pharaon

PILOTS FOR HEALTHY AND ACTIVE AGEING

Grant Agreement: 857188

D4.5 Service orchestration support tools-intermediate

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:	**D4.5**
Deliverable title:	Service orchestration support tools-intermediate
Deliverable version:	1.0
Work Package number:	WP4
Work Package title:	Technology Support Tools
Due Date of delivery:	M36 (November 2022)
Actual Date of delivery:	M36 (November 2022)
Dissemination level:	Public (PU)
Type	Report + Other (O)
Editor(s):	Andrej Grgurić (ENT)
Contributor(s):	Andrej Grgurić (ENT) Boris van Schooten (RRD) Danny Pape (ASC) Elisabete Pitarma (CDC) Jose Luis Alfonso (ICE) Jure Lampe (SENLAB) Miguel Ángel Beteta (CETEM) Miran Mošmondor (ENT) Ricardo Perez de Zabalza (MIWENERGIA) Rodríguez Vázquez, Jorge Juan (INDRA)
Reviewer(s):	Erika Rovini (UNIFI)
Project name:	Pilots for Healthy and Active Ageing
Project Acronym	PHArA-ON
Project starting date:	01/12/2019
Project duration:	48 months
Rights:	PHArA-ON Consortium

Document history

Version	Date	Beneficiary	Description
0.1	1.9.2021	ENT	Initialized document
0.2	2.09.2021	ASC	Added Evaluation reports for Logging, Monitoring and Orchestration
0.3	8.09.2021	INDRA	Monitoring orchestration in Pharaon
0.4	29.09.2021	ENT	Update to OpenAPI3 and inputs to Interactive documentation of REST APIs in Pharaon
0.5	15.10.2021	ENT	Documenting Pharaon (new) API design-first process
0.6	25.10.2021	ENT	GraphQL as orchestration layer chapter input
0.61	11.1.2022		GraphQL as orchestration layer chapter update
0.7	3.8.2022	ENT	Editor updates, updates to quality properties of Pharaon APIs chapter
0.8	18.8.2022	ENT	Chapters Service registration in Pharaon, API orchestration, Appendix A
0.9	19.9.2022	IND	CaaS chapter added
0.91	20.9.2022	ENT	Added chapter Endpoints serving the Pharaon API standardized descriptions and ENT input
0.92	23.9.2022	MIWENERGIA	uGrid info added
0.921	25.10.2022	ICE	Chapters related to Service Orchestration
0.93	12.10.2022	UNIFI	Internal Revision of the document
0.94	17.10.2022	ENT	Editor overall updates
0.95	19.10.2022	CETEM	Amicare info added
0.96	19.10.2022	RRD	RRD related input added
0.97	19.10.2022	INDRA	Review of sections 3 & 7
0.98	20.10.2022	MAIN	Input Adsysco and MAIN in tables
0.99	14.11.2022	ENT	Editor updates, fixes in section 4 on polishing Ascora inputs according to internal reviewer comments
0.991	16.11.2022	CDC	Overall proofread and review
0.992	17.11.2022	SEN	SenLab related input added, comments added
0.993	21.11.2022	ASC	Ascora inputs
0.994	21.11.2022	ENT	Final updates
1.0	29.11.2022	ENT	Wrap up for submission

PM efforts per beneficiary having contributed to the deliverable.

#	Partner	PM effort in this deliverable
1	UNIFI	0.25
7	CETEM	0.1
10	MIWENERGIA	0.1
15	INDRA	1
18	CDC	0.1
27	ASC	0.1
28	ENT	1
32	ICE	1
37	SENLAB	0.1
TOTAL	Pharaon Consortium	3.65

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

Executive Summary

This deliverable reports on the work carried out within the tasks of the fourth work package (WP4) of the PHArA-ON (in further text "Pharaon") project.

This deliverable is the second one within task T4.2. aiming to provide support for service orchestration activities in the Pharaon project. We define Pharaon service orchestration as the processes, procedures, guidelines, and tools implemented to manage multiple platforms, technologies (Pharaon internal and external onboarded via WP6 open calls), services and APIs. The goal is to create a seamless user experience that enables co-creation across multiple platforms, technologies, and technology providers. In the process, a tight coordination has been established with task T4.3 (parallel deliverable D4.8 reports on the T4.3 activities) but also other WP4 and cross-technical WP level conducted primarily within the SWAT1 task force lead by the project TM (Technical Manager).

This deliverable is a result of work done after a comprehensive analysis of the Pharaon deployment setup and organization to understand the specifics, limitations and opportunities. It furthers the proposed and advocated OpenAPI-based interactive and always up-to-date documentation of the Pharaon REST API. In comparison with D4.4 where this OpenAPI has been proposed, has been largely used during the integration work. For example, it provides guides and examples how to expose and later use the OpenAPI specification for the generation of the stub client code to streamline and facilitate the technical integration work has been successfully used in the integration work activities related to Slovenian and Italy pilots.

Since in the process of integrating new Pharaon APIs are sometimes also being developed this deliverable summarizes the process of designing the new APIs in Pharaon. It also identifies critical quality properties of the Pharaon APIs. To explore further and give hands-on experience and examples of open-source data query and manipulation language for APIs, GraphQL was explored in more detail and findings were discussed and shared within Pharaon. This deliverable reports on the GraphQL as orchestration layers for downstream APIs and gives comparisons of REST APIs and GraphQL with pros and cons, tools and services and summarized the API conversion and aggregation from existing Pharaon endpoints via conversion of GraphQL Schema to new Pharaon GraphQL endpoints (proxies).

Container orchestration tools, monitoring orchestration tools are also reported in this deliverable together with the service registry considerations for Pharaon (an activity that was continued in T4.3).

To provide support with high-level management of the technical integration and service orchestration work the benchmarking of the project management tools was done within this deliverable as a result of the discussions and a request from cross technical WP SWAT1 task force led by the project Technical Manager.

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

This deliverable documents the support activities related to Pharaon API design, documentation and orchestration. Design first approach is documented for new Pharaon APIs, selected quality properties of Pharaon APIs are summarized, and the newest version of the OpenAPI standard is presented and adopted within Pharaon (since in comparison with the previous version the new libraries serving the OpenAPI-based interactive REST API descriptions have emerged). GraphQL was explored in more detail and findings were discussed and shared within Pharaon. This deliverable reports on the GraphQL as orchestration layers for downstream APIs and gives comparisons of REST APIs and GraphQL with pros and cons, tools and services and summarized the API conversion and aggregation from existing Pharaon endpoints via conversion of GraphQL Schema to new Pharaon GraphQL endpoints. Container orchestration tools are also reported in this deliverable together with the service registry considerations for Pharaon. This document reports on the extensive in-depth analysis and evaluation that took place in the fields of Logging and Monitoring to enable the Pharaon environment to connect to and use this as a holistic and central possibility for overviewing the status of the Pharaon system.

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
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 second one within task T4.2. aiming to provide support for technical activities in the Pharaon (primarily conducted in WP3 and WP5) with a focus on orchestration support.

Pharaon selected input technologies for each pilot are cataloged within D3.1, where the gap analysis with relation to requirements coming from each Pharaon pilot is also done to identify customizations to be done. To facilitate the integration work and interoperability the proper documentation and instructions are made available within Pharaon Technical One Stop Show (described in more detail within D4.2). The work within this task (T4.2) continued to facilitate the coordination between Pharaon technical providers and more concretely facilitate the Pharaon services orchestration.

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

1.2 Relation to other tasks and deliverables

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 1.1).

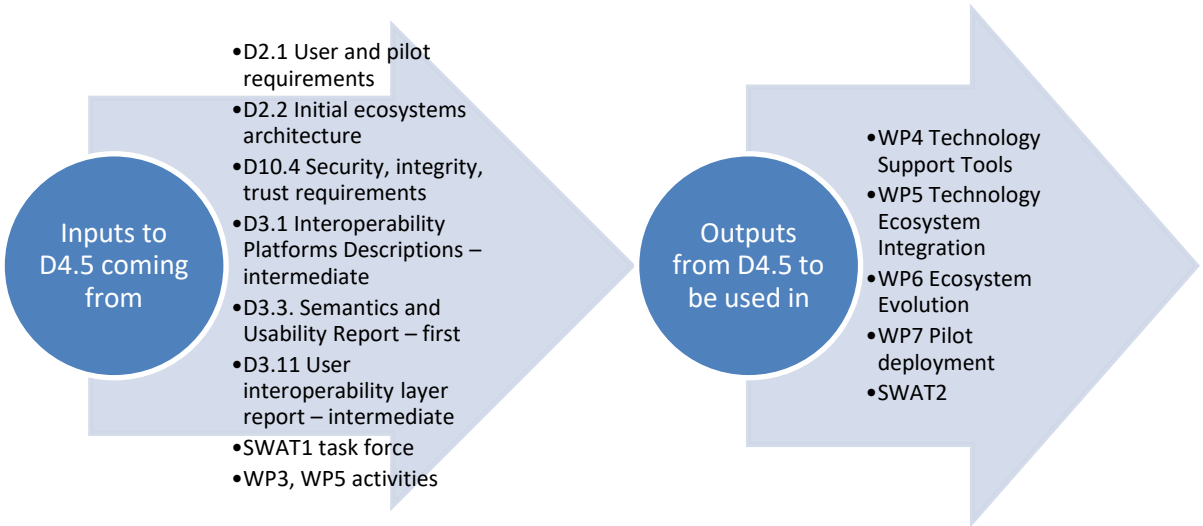


Figure 1.1 Inputs and outputs of D4.5 concerning other project work

1.3 Structure of the deliverable

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.

In chapter 2 Pharaon API design, documentation and orchestration are elaborated. A new API design-first process is explained, properties of Pharaon APIs are given, Interactive documentation of Pharaon REST APIs is explained with additional investigation and analysis of GraphQL as an orchestration layer for downstream APIs.

Chapter 3 elaborates on containers orchestration and further analyzes Kubernetes in relation to Pharaon.

Chapter 4 elaborated the monitoring orchestration tools.

Chapter 5 discusses registries in Pharaon, summarizes some discussions in Pharaon and gives tools comparison.

Chapter 6 gives some insight into concrete API orchestration activities done within SLO and IT pilots.

Chapter 7 summarizes work on benchmarking project management tools.

Chapter 8 concludes the report.

1.4 Terminology

Some of the terms we use here we defined as follows:

- **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 catalog** 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 portal** - a place where developers can go to access a company's (or in our case project Pharaon's) APIs
- **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
- **Service orchestration** in Pharaon is envisioned as the process, procedure, guidelines, and tools implemented to manage multiple platforms, technologies (Pharaon internal and external 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 D4.11).

2 Pharaon API design, documentation and orchestration

Following on the previous work regarding the REST API recommendations (documented in the previous version of this deliverable vD4.4.) a designated chapter has been added to Pharaon's Developer's Handbook (hosted on: https://gitlab.com/pharaongroup/developers-handbook/-/wikis/Best-practices/REST-API-best-practices).

While the exact functional and quality scope of the APIs is a matter of designated input technologies and Pharaon's WP3, where the interoperability scenarios are enabled to be finally integrated within Pharaon's WP5, work reported here gives some recommendations and common practices to ease this work and to give recommendations for a) design and implementation of the new APIs that are being created to enable Pharaon use cases and scenarios and b) to document existing APIs following well-established standards and tools to ease the integration and orchestration process and make it more efficient.

2.1 Pharaon (new) API design-first process

As previously mentioned, many APIs used in Pharaon are inherited from input technologies and are not to be rewritten. Efforts are instead invested in extensions and upgrades to support Pharaon use cases and scenarios. With that respect, integration is also facilitated by the work reported in this deliverable in terms of recommending methods to increase efficiency, such as applying interactive documentation (see chapter 2.2) for the existing REST APIs. However, some of the APIs are also to be developed from scratch (which may happen if the old APIs quality is simply insufficient for Pharaon, which is to be determined within WP3; or if the API needed merely does not exist). In the latter case, the recent *API design-first approach* by James Higginboarham using the "**Align-Define-Design-Refine" (ADDR)** process is recommended as suitable for Pharaon. 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 API 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 is to deliver the desired API with desired quality and to prevent breaking changes

and large code refactoring. It also allows consumers to start the integration process without waiting for full implementations to become ready and deployed in a staging (integration) environment.

The APIs are to be delivered in iterations having in mind that breaking changes should be avoided as much as possible and always in coordination with the API consumer.

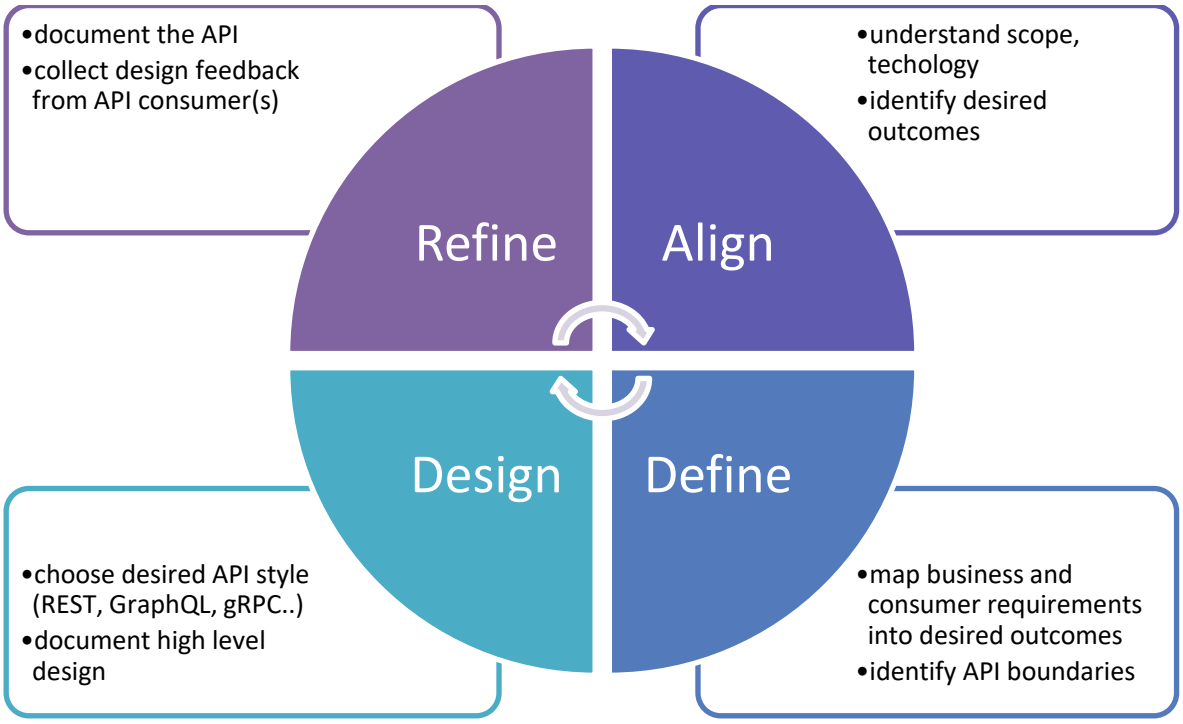


Figure 2.1 ADDR (Align-Define-Design-Refine**) process for Pharaon's new APIs**

2.2 Quality properties of Pharaon APIs

The essential prerequisite of integration and orchestration is that the input solutions and services are *consumable* in terms that they are capable of being used or engaged with. In this sense, the property of being compliant with standards, delivering what they promise and being accessible is vital. The following figure (Figure 2.2) illustrates and summarizes not only desirable but also in most situations, critical properties of the APIs that are to be integrated into the Pharaon ecosystem. 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.

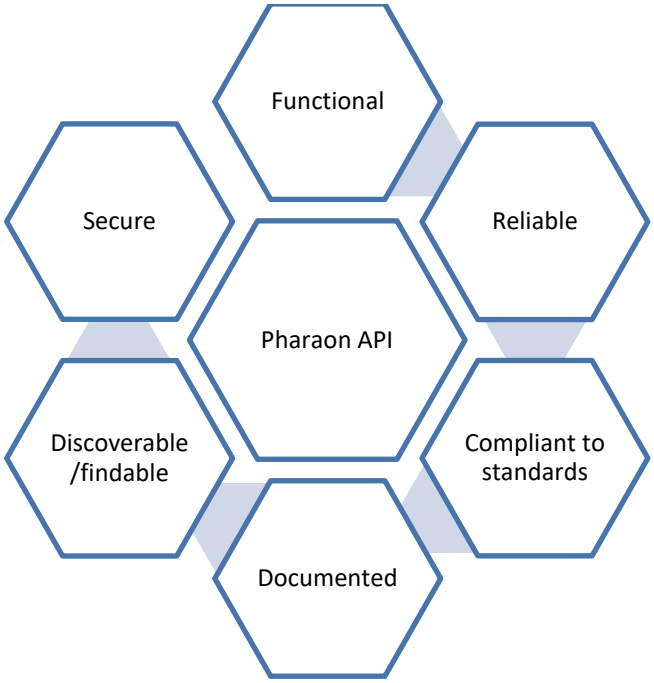


Figure 2.2 Quality properties of Pharaon APIs

The functional aspect is in most cases predefined and made available with the selection of the input technology solution. Within Pharaon, the goal is to advance the maturity level of Pharaon input technologies APIs so that all APIs (and especially APIs exposed on the public IP addresses where Pharaon integration is happening) are:

- Reliable: to be able to connect to the API and consume its functionality;
- compliant to standards: in both the protocol and data formats used;
- properly documented
- findable: to share the publicly exposed URLs of the Pharaon APIs;
- secure: to have secure endpoints, including authentication mechanisms.

2.3 Interactive documentation of REST APIs in Pharaon

Initial considerations on the process of applying OpenAPI in Pharaon are described in Developers Handbook (served as a wiki at https://gitlab.com/pharaongroup/developers-handbook and described in D4.1, updated in D4.2).

In the previous version swagger, 2.0 was used, while now we use Swagger UI to visualize the OpenAPI description of our REST interfaces. The OpenAPI Specification defines a standard interface to RESTful APIs. 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. It can be used with both a top-down and bottom-up API development approach.

The following screenshots (Figure 2.3, Figure 2.4) show the older and newer version, respectively. Both figures show selected microservice REST API from the SmartHabits system which is used in Slovenian and Italian pilot sites. They are served from ENT private cloud server used as a development environment. And used for integration work (described in WP5) hosted within the integration test sandbox (as described in D4.11 released at the same time as this deliverable).

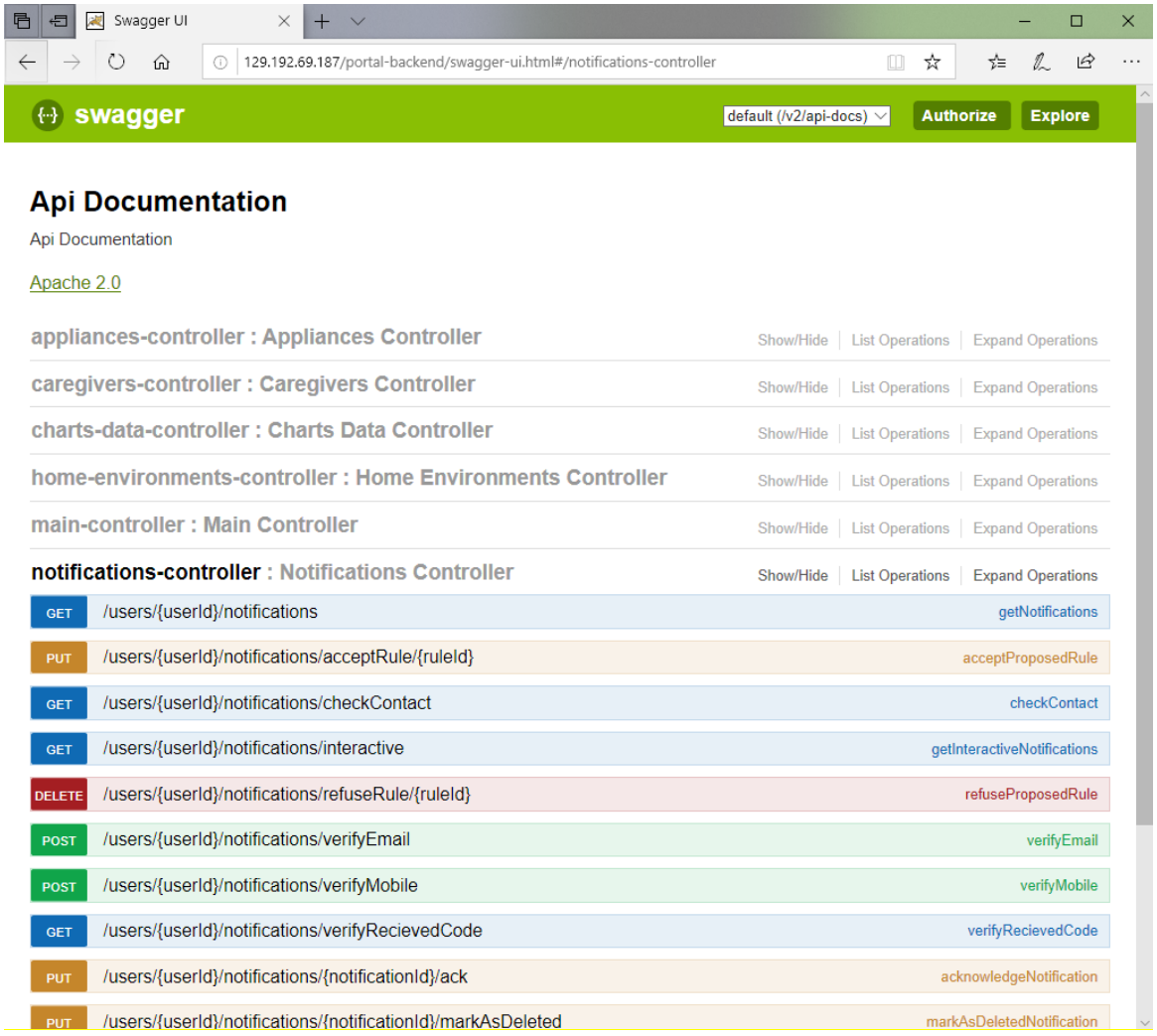


Figure 2.3 REST API interactive documentation for SmartHabits based on Swagger2.0

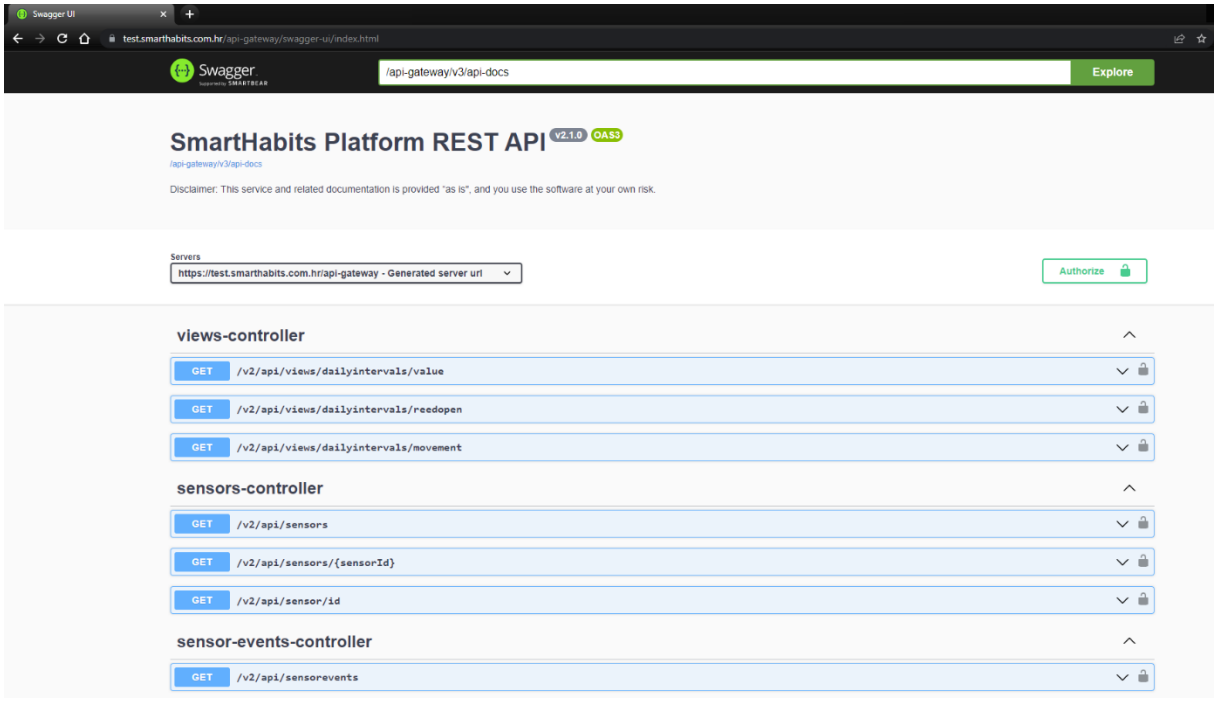
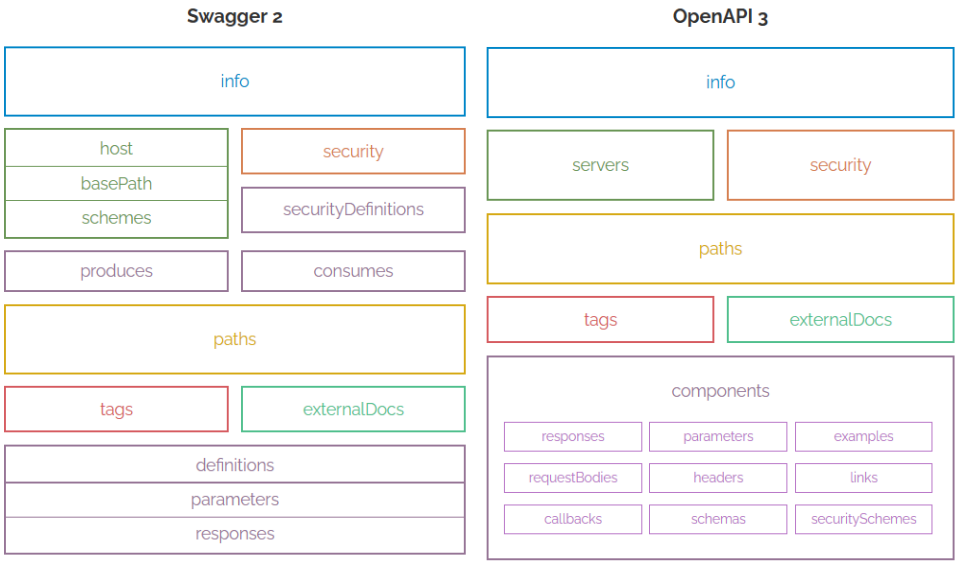


Figure 2.4 REST API interactive documentation for SmartHabits based on Swagger3.0

2.3.1 Swagger2 vs. OpenAPI3

The *OpenAPI* specification was formerly known as the *Swagger* Specification so here we compare the swagger 2 vs OpenAPI3. Adoption of OpenAPI3 was done to switch to the newest specification



More notable changes in the new version are:

- The API's base URL is now defined in a single 'url' attribute instead of the previous separation into 'host', 'basepath' and 'schemas',

```
"servers": [
    {
        "url": "http://129.192.68.219:80",
        "description": "Inferred Url"
    }
],
```

- The new 'requestBody' attribute supersedes the previous 'formData' parameter type and 'body' attribute and is now more flexible in allowing the specification of different types of content with various media types.
- Some renames and changes in the structure.
- New features added are *Callbacks* and *Links*:
    - Callbacks or webhooks[1]: used to avoid frequent polling and to push information to API consumers in realtime. A consumer registers a URL for a specific event, and the provider then posts information to this URL when the event of interest occurs.
    - Links[2]: give info on possible related API calls. Based on the initial API call, a new API call can be done that returns the content. This kind of relationship can be defined using Links in OpenAPI. Hypermedia[3] is a *dynamic* way to express links to related resources and actions within an API response, whereas this specification feature is a *static* way to specify these relationships.
- For setting up a client to access the OpenAPI software components, OpenAPI generators can be used, which can be found online at https://openapi-generator.tech/.
- The software generates clients, servers, and documentation from OpenAPI 2.0/3.x documents. It supports 50+ client generators (for different programming languages and frameworks, ranging from Haskell to AngularJS), which can be easily used to generate code to interact with any server which exposes an OpenAPI document. Those clients can even be customized, to fit the developers special needs. Many of these generators use Inversion of Control, to support updating the client side, without much influence on the code base. Because of the standard followed, the effort to integrate those Pharaon components to third-party software is reduced by a huge factor.

2.4 Endpoints serving the Pharaon API standardized descriptions

The following table summarizes the endpoints serving the Swagger or OpenAPI YAML description of the Pharaon APIs. Those descriptions can also be used as input to ICE Orchestrator (described in the related D4.8 deliverable)

Table 2.1 Endpoints serving the Pharaon API standardized descriptions (dev or integration sandbox as described in D4.11)

Pharaon solution	technology	Swagger endpoint

[1] https://swagger.io/docs/specification/callbacks/

[2] Using links, you can describe how various values returned by one operation can be used as input for other operations, for more info see: https://swagger.io/docs/specification/links/

[3] With HATEOAS (**Hypermedia as the Engine of Application State**) a client interacts with a network application whose application servers provide information dynamically through hypermedia https://www.wikiwand.com/en/HATEOAS

Globalcare	NA
HS.Helios Platform	NA
IoTool Platform	https://demo.iotool.io/docs/
IoChat Platform	https://demo.iochat.io/docs/ (Work in progress Nov 2022)
MISS Activity	
Dutch Intake Wizard	https://portals.rrdweb.nl/pharaon/wizard/webpages-openapi.yaml
PACO - Web application	https://servlets.rrdweb.nl/r2d2project/paco/
RRD eHealth Platform	https://servlets.rrdweb.nl/r2d2/
Amicare	http://amicare-swagger-bucket2.s3-website.eu-central-1.amazonaws.com/
uGrid	https://api-pharaon.miwenergia.com/swagger/index.html
SmartHabits Platform	https://test.smarthabits.com.hr/api-gateway/swagger-ui/
ICE Orchestration	NA
Discovery	https://backend-pharaon-dev.ascora.eu/swagger/index.html
AdSysCo RegiCare	https://pharaon.dev.adsysco.nl/docs/regiapi.v2.yaml

Note that 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

GraphQL is *an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data.* It 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/). The specification is hosted on GitHub (https://github.com/graphql/graphql-spec). The GraphQL landscape is visualized in the following figure (Figure 2.5).

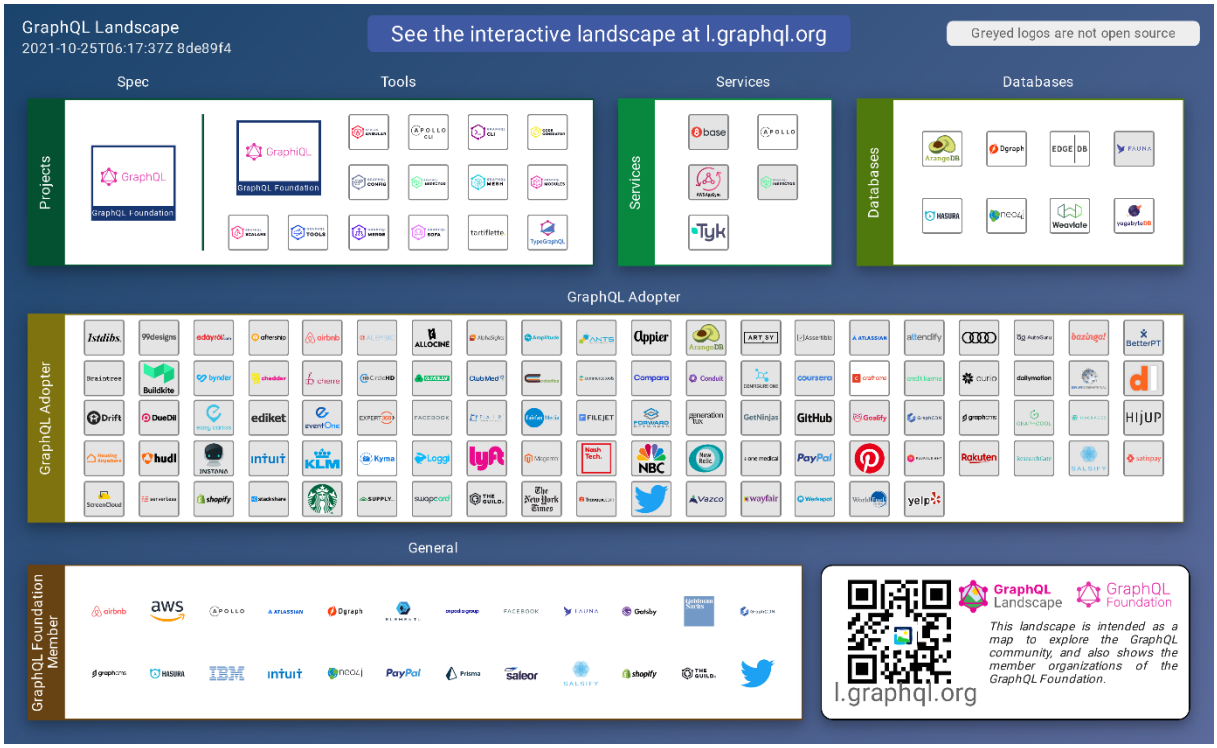


Figure 2.5 GraphQL landscape [source: https://landscape.graphql.org/ 25.10.2021]

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 2.6) shows how the Pharaon API façade is used to orchestrate across multiple fine-grained sets of (Pharaon) API calls and moves from the antipattern of overloading Pharaon API consumers with making multiple calls. Hiding the complexity of underlying APIs makes the experience of API consumers much better and the whole process much more streamlined and efficient.

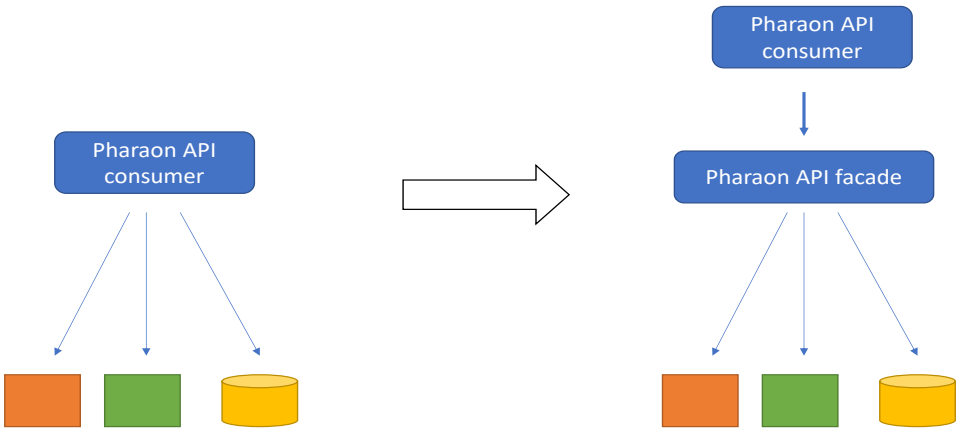


Figure 2.6 Pharaon API façade orchestration

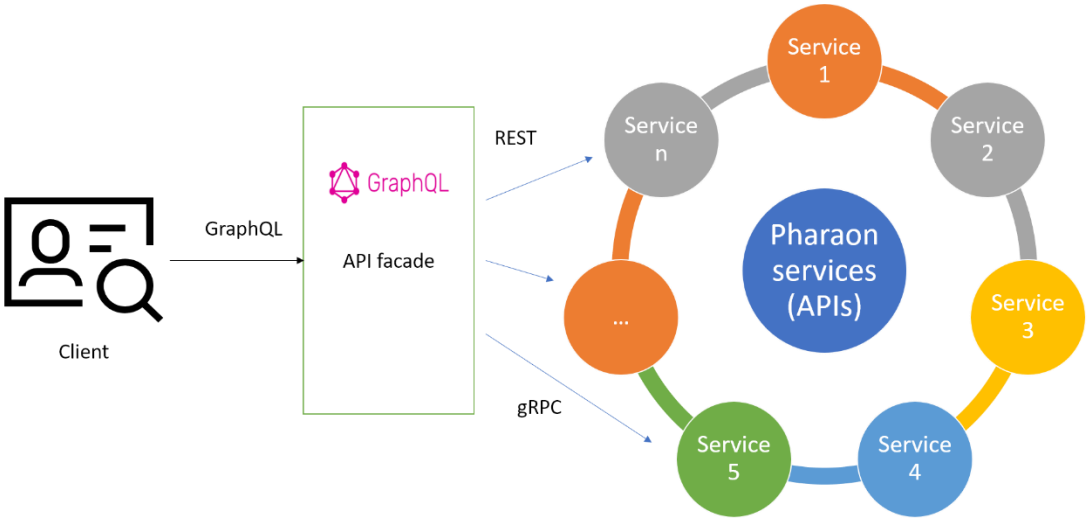


Figure 2.7 GraphQL Pharaon API façade

Apart from the BFF pattern, we see its usefulness in building new features directly within GraphQL and using it as a business logic layer as well.

2.5.1 Pros of using GraphQL

Pros of using GraphQL:

- Developer productivity
- Faster delivery
- Access to the entire schema (since all operations are on the same endpoint)
- API catalog
- Fetching just the needed info (no overfetching and no underfetching)
- Fetching in just 1 call (unlike in REST, where to reach specific info you need more calls)
- Creating a request by defining the exact wanted resources (via POST to the server) and receiving a response matching the format of our request
- Human readable queries
- Evolving APIs without using versions
- Use any programming language
- API discoverability (better handled than in REST level 3 and HATEOAS) since all data can be fetched just from the root endpoint

2.5.2 GraphQL tools and services

There are many tools and services related to GraphQL including the ones stated at https://graphql.org/code/ (where tools such as GraphQL Code Generator[4], GraphiQL[5]) and services such as Altair -as an alternative to Postman see - are listed).

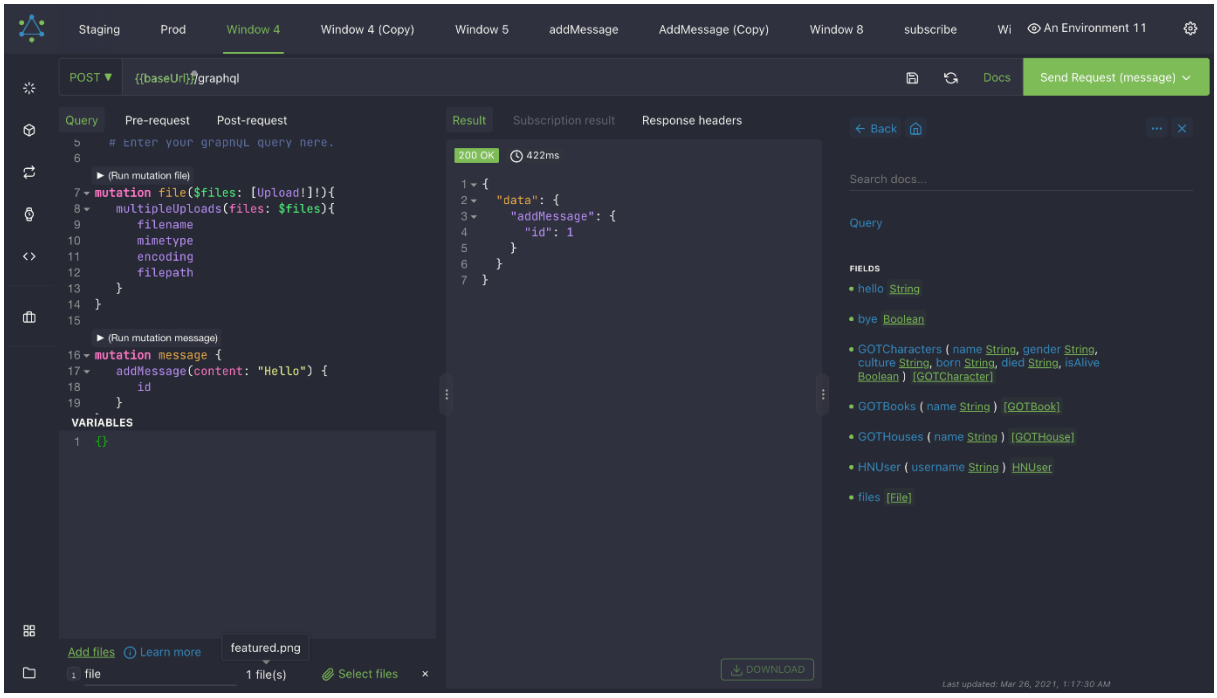


Figure 2.8 Altair GraphQL Client [image from https://altair.sirmuel.design/ 5 Nov 2021]

2.5.3 REST vs GraphQL comparison

A short comparison of REST and GraphQL-based APIs is shown in the following table (Table 2.2).

Table 2.2 GraphQL vs REST comparison

	GraphQL	REST
Architecture	Client-driven	Server driven
Data fetching	Specific data in a single API call	Fixed data with multiple API calls
Data exchange	At single endpoint	At multiple endpoints
Organized	Schema and types	Endpoints
API evolution	Without version	With versions or breaking changes

[4] https://www.graphql-code-generator.com/ Generate code from your GraphQL schema and operations with a simple CLI; GitHub: https://github.com/dotansimha/graphql-code-generator MIT License

[5] https://github.com/graphql/graphiql GraphiQL is the reference implementation of this monorepo, GraphQL IDE, an official project under the GraphQL Foundation. The code uses the permissive MIT license.

Learning curve	High	Medium
Functions	Called resolvers. Used for a field within a type	Used for specific URLs
Operations	Query mutation subscription. read operations "queries" and write operations "mutations."	CRUD (Create, Read, Update, Delete)
Concept of resources	Yes, identified as URLs	Yes, identified as URLs
Protocol and data format	Typically HTTP, JSON (with Gzip encouraged in production)	HTTP, JSON
Queries	Automatic validation and type checking	Better for complex queries
Use cases	Multiple microservices	Resource-driven applications
File upload	No	Yes
Self-documenting	Yes	Needs OpenOPI or similar
Development speed	High	Medium
Performance	Fast	Multiple calls take more time
Community	Medium	Large
Security	While GraphQL offers flexibility in building APIs, it involves complex configurations that may expose applications to security vulnerabilities	More common and known to Pharaon partners

In a recent (September 2021) paper Lawi et al[6]. proposed a new research methodology to evaluate the performance of REST and GraphQL API services. The performance evaluation used basic measures of QoS (Quality of Services), i.e., response time, throughput, CPU load, and memory usage. Their results showed REST is still faster up to 50.50% in response time and 37.16% for throughput, while GraphQL is very efficient in resource utilization, i.e., 37.26% for CPU load and 39.74% for memory utilization. The authors concluded that GraphQL is the right choice when data requirements change frequently, and resource utilization is the most important consideration. They summarize the results in the following figure (Figure **2**.**9**).

[6] Lawi, Armin, Benny L.E. Panggabean, and Takaichi Yoshida. 2021. "Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System" *Computers* 10, no. 11: 138. https://doi.org/10.3390/computers10110138

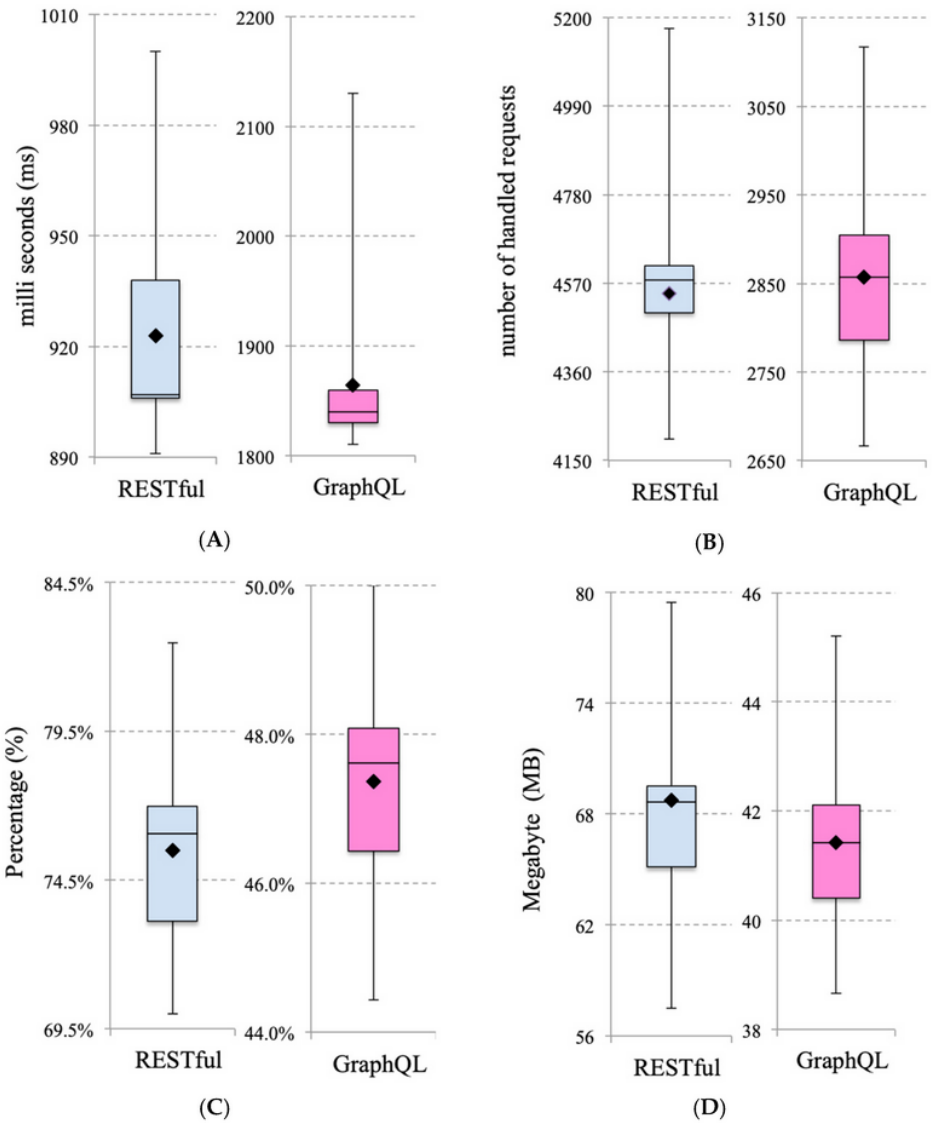


Figure 2.9 Boxplot of evaluation results for the performance measurement on REST and GraphQL services; (A) response time (ms), (B) throughput (handled requests), (C) CPU Load (%), and (D) memory utilization (MB) [image from https://www.mdpi.com/2073-431X/10/11/138#]

2.5.4 API conversion and aggregation

The following figure (Figure 2.10) summarizes the steps of API conversion from REST API to GraphQL API.

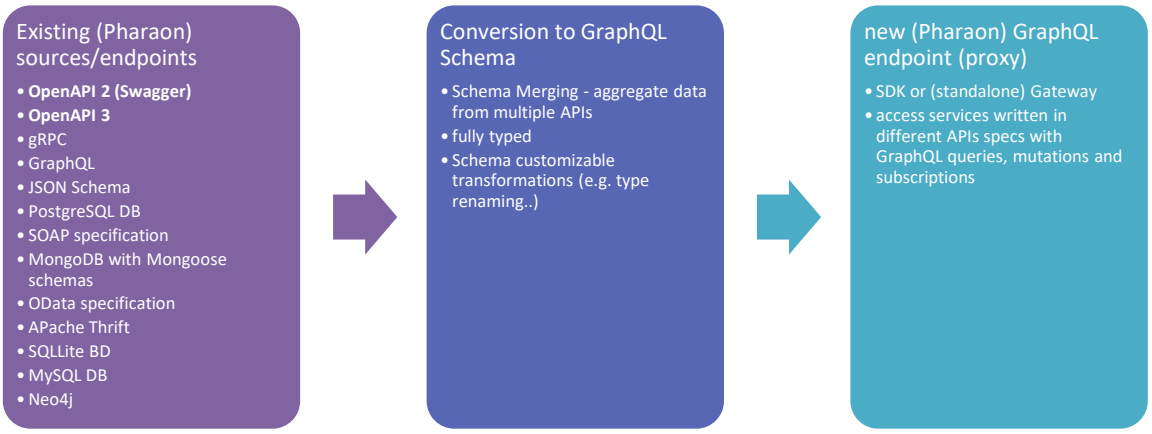


Figure 2.10 API conversion and aggregation from Existing REST-based Pharaon APIs to GraphQL endpoints (serving as proxies to underlying REST APIs)

3 Container orchestration tools

3.1 Containers as a service (CaaS)

Container as a Service (CaaS) is somewhere in between Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). It is a form of container-based virtualisation 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 vendor provides the framework, or orchestration platform, on which the containers are deployed and managed; and it is through this orchestration that critical IT functions are automated. This model is especially useful for developers, as it allows them to design scalable and more secure containerized applications. Users can purchase only the resources they want (scheduling functions, load balancing, etc.), which saves money and increases efficiency.

The core resources of CaaS are containers: a well-known deployment mechanism for cloud-native applications and microservices. Also, CaaS increases portability between hybrid or multicloud environments.

Among the advantages offered by CaaS, the following can be highlighted:

- **Portability:** Containerized applications have everything they need to run and can be deployed in multiple environments, including private and public clouds. Portability means flexibility, as workloads can be moved more easily between environments and providers.
- **Scalability:** Containers are characterized by horizontal scaling, which means that a user can multiply identical containers within a single cluster to expand as needed. By using and running only what you need and when you need it, you can reduce costs considerably.
- **Efficiency:** Containers require fewer resources than virtual machines (VMs), as they do not require a separate operating system. In addition, they require fewer hardware systems without an operating system, and you can run multiple of them on a single server, which reduces costs.

- **Increased security:** Containers are isolated from each other, so if one container is compromised, the others are not affected.
- **Speed:** Because they are not dependent on the operating system, it only takes a few seconds to start and stop a container. This feature also enables faster operational speeds and development, as well as a smoother and more agile user experience.

The following schema shows how CaaS works. Pods are a set of one or more containers deployed under a single host. All containers deployed within the same pod will share storage and network resources with each other.

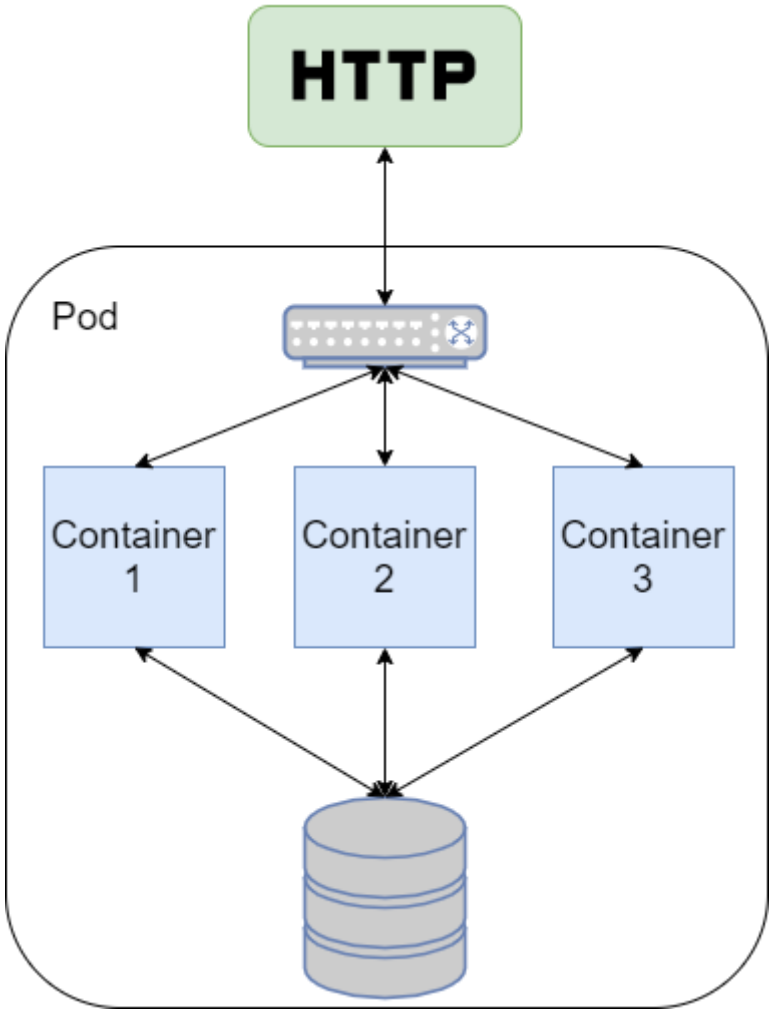


Figure 3.1: Example of CaaS distribution with several containers sharing a single storage system, all of them within a single pod

CaaS allows for the definition and automation of the end-to-end life cycle of the containers, which is managed in a highly automated way. The specifics can be customized, specifying which status is initial, which is final, and what statuses can be reached from each of the available statuses, like in the following diagram:

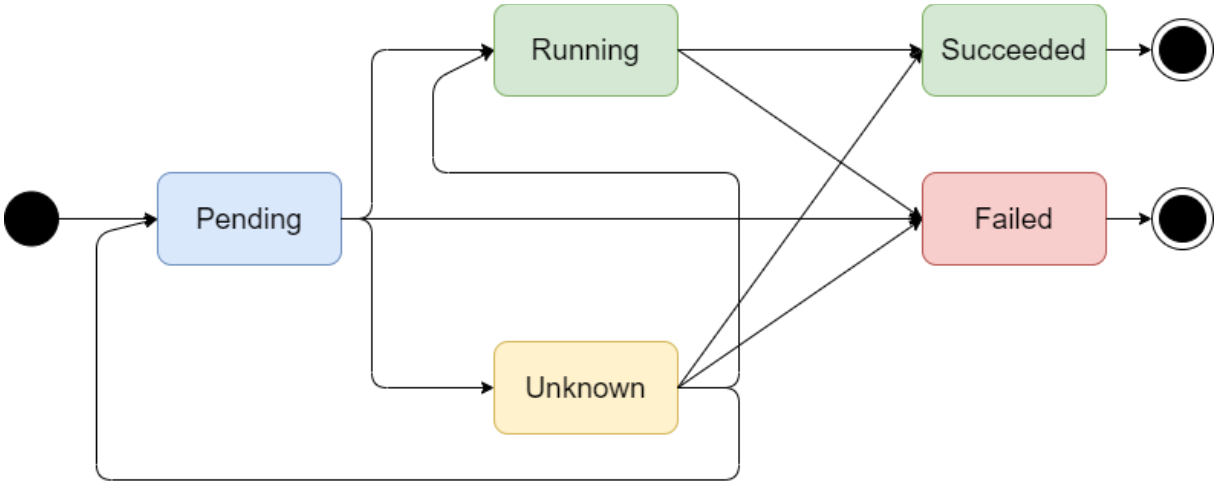


Figure 3.2: Container life cycle, with initial status Pending; final statuses Succeeded and Failed; and possible flows

3.2 Kubernetes distributions analysis

Kubernetes is an open-source platform for managing workloads and services. It makes it easy to automate, configure and deploy our software and applications. Being a widespread technology that has also been used to develop other technologies, Pharaon partners have noticed its versatility, and it is used in the Andalusian pilot for deployments, which are based in Indra's Onesait Platform; as well as by ICE's Interoperability Data Tool. Also, SenLab has prepared demo and test clusters based on Kubernetes for IoTool and IoChat platforms, although the production servers for Slovenian and Italian pilots stay on the classic platform (separate VPS servers for IoTool and IoChat). The main reason is that especially IoTool is tightly bonded with other platforms, like SmartHabits and during the pilots phase, it is not wise to move them to other service providers and technology stacks. The partners have also noticed that several CI/CD technologies under consideration are easily supported by Kubernetes (See D5.1 section 4.7 and appendix A for more details on this).

Kubernetes, sometimes called K8, provides us with a container-centric management environment with which to orchestrate, compute, network and storage infrastructure for workloads.

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** is in charge of 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):

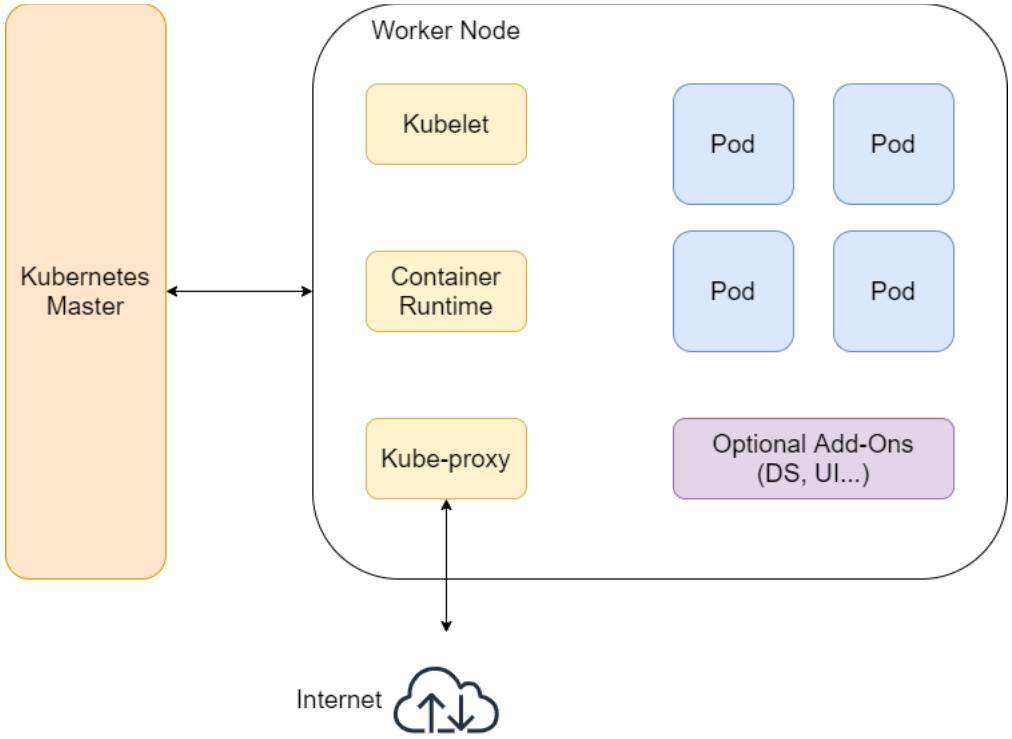


Figure 3.3: Kubernetes Architecture

Orchestration systems for microservices were investigated with the goal to analyse the integration of monitoring and logging into multi-cloud systems. For this purpose, K8s distributions have been compared and evaluated as well as the deployment of a joint control layer to propagate KPIs and log information between multiple, decentralized clusters. Figure 3.4 shows the visualisation of the evaluated setup of the decentralized clusters.

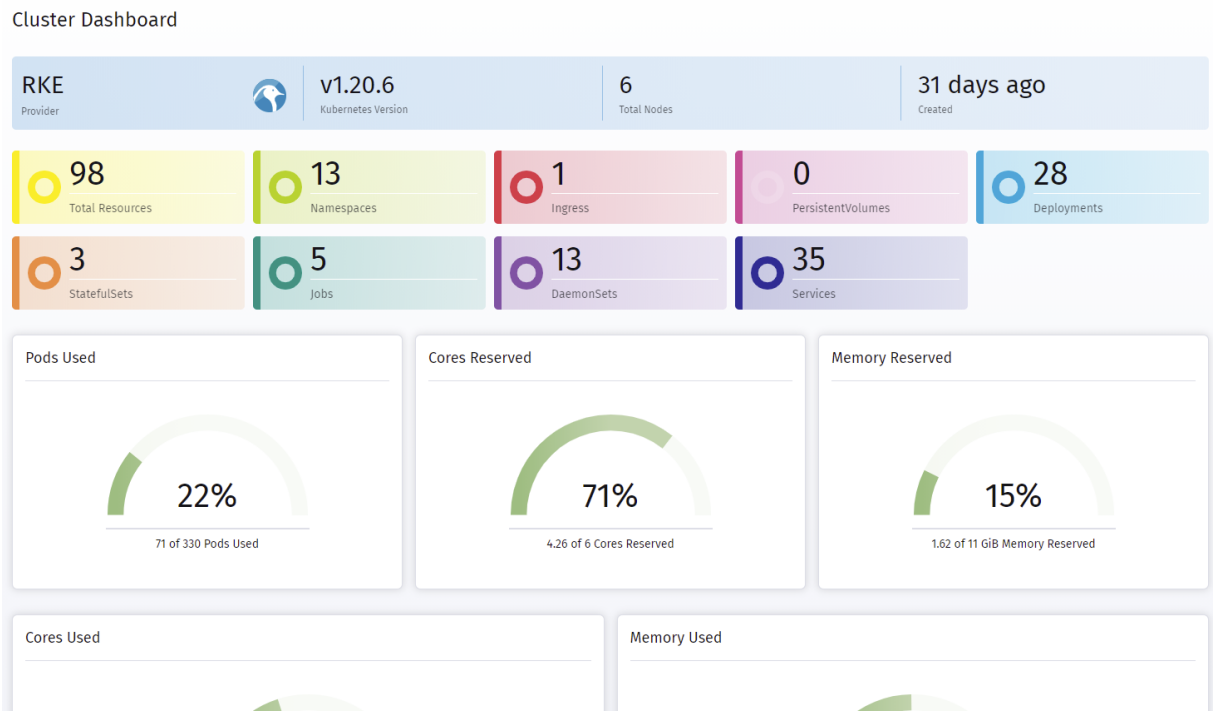


Figure 3.4: K8s Cluster Dashboard from Rancher, providing real-time details of a cluster such as percentage of used pods, number of nodes, date of creation, software version, etc., within the Andalusian pilot

4 Monitoring Orchestration Tools

Several monitoring approaches for Pharaon software and hardware environments were analysed. The first approach resulted in a dedicated lightweight monitoring solution, developed in node.js and deployed as docker containers. **Figure 4.1 and Figure 4.2** show the graphical user interface of this solution.

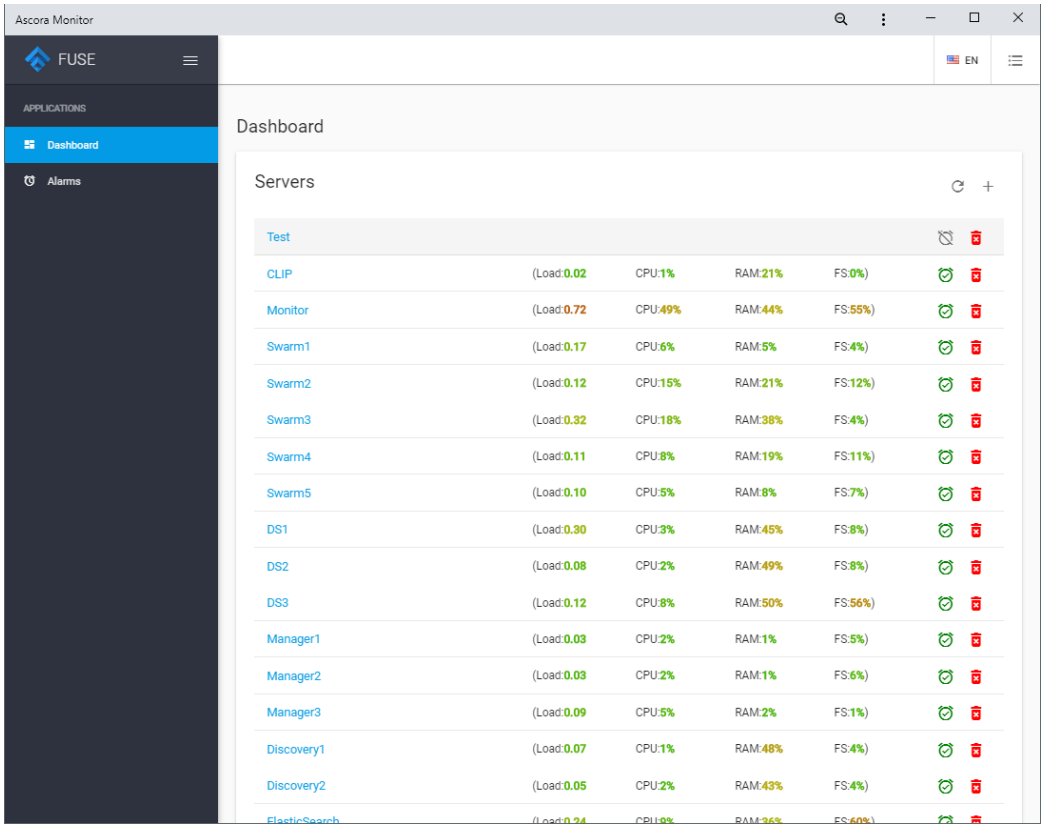


Figure 4.1: Lightweight Monitoring Overview

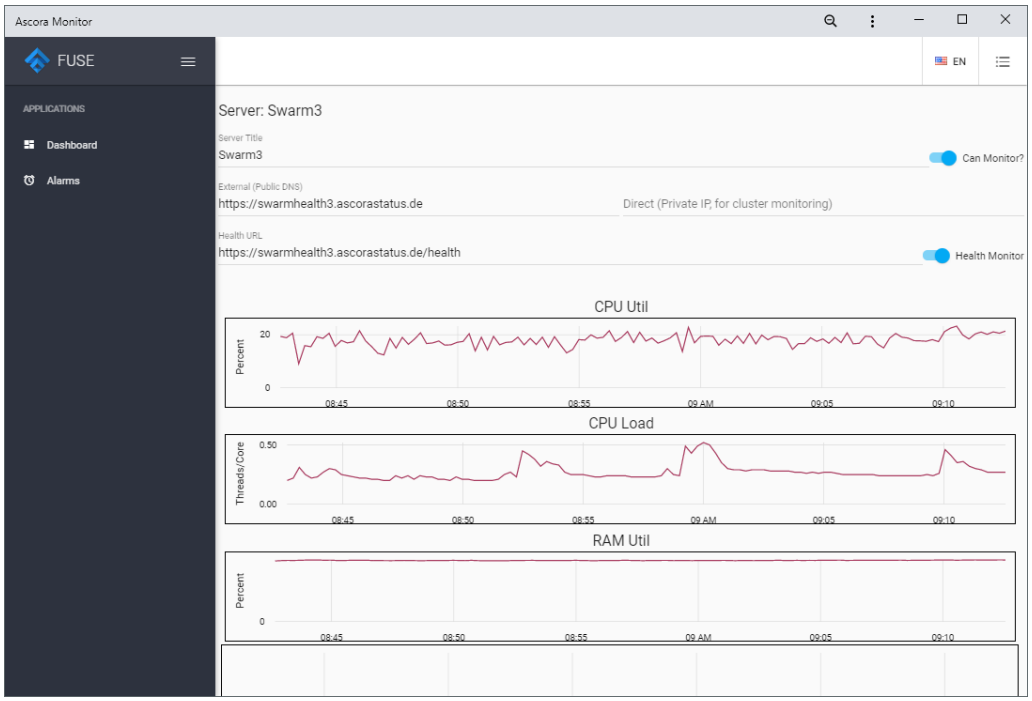


Figure 4.2: Lightweight Monitoring Details

Evaluation of a monitoring stack based on Prometheus and Grafana to monitor KPIs of distributed systems. In this scenario, monitoring of KPIs for collecting more than 40 different KPI types from a cluster of 51 servers. This solution has been put in place since February 2021 and since then

successfully monitoring the KPIs (see Figure 4.3). Setup an additional multi-stage alerting system and successfully tested it in one event. The future focus will be on the further implementation of the alerting system and especially in its integration into orchestration systems.

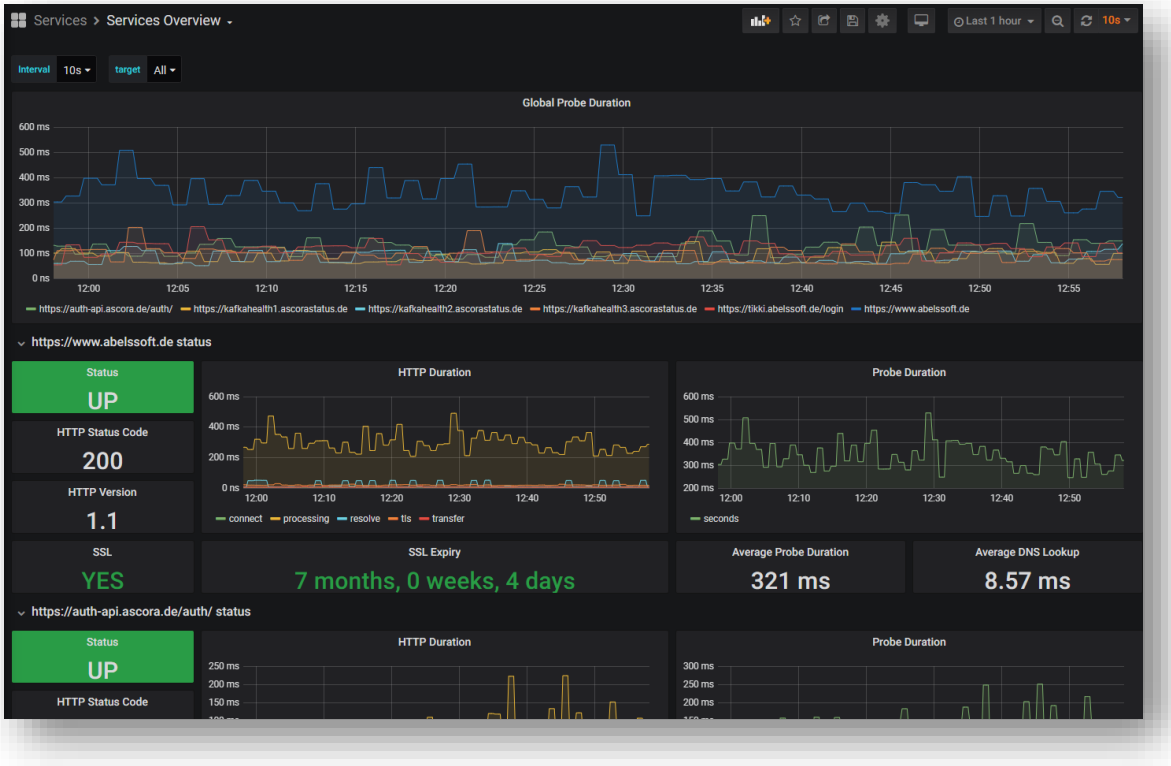


Figure 4.3: Grafana Visualisation during the evaluation

4.1 Logging Management

A detailed evaluation and a test for the log management and for this purpose was done in a cluster with the ELK stack. ELK is a composition of **E**lasticSearch, **L**ogstash and **K**ibana. Each component fulfills 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) provides the Pharaon Consortium with a good option to run a holistic log management tool for its components.

The Log system in the first half of the period was setup and has battle-tested the scalability within the second half by adding more than 1.560 Mio (1.5 Billion) real-world log data sets into the system for simulating the practical use within possible distributed Pharaon scenarios.

The following screenshot shows the setup (company internal loggings are redacted):

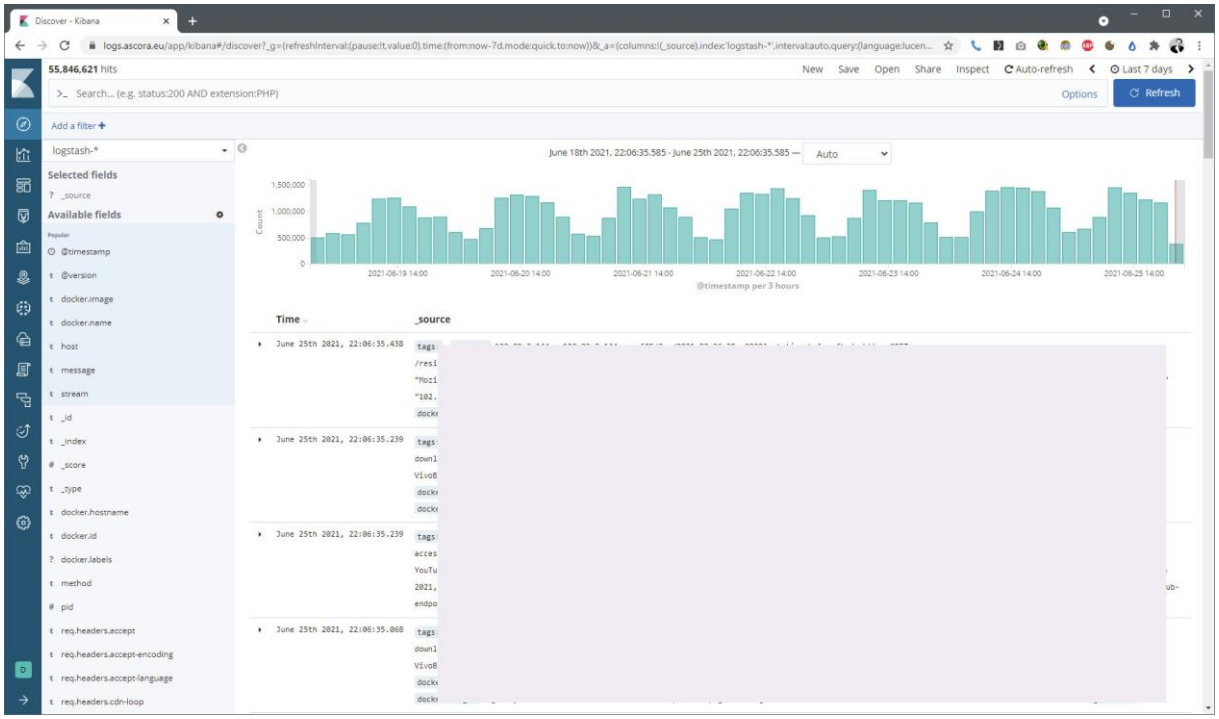


Figure 4.4: ELK Stack Setup

Horizontal scalability capability was improved to better cope with high-load fault-tolerant scenarios.

5 Registries in Pharaon

5.1 Pharaon specifics

Each Pharaon platform or technology exposes different remote services (accessible via APIs) at a specific public web location. Pharaon services need to call one another but their location (IPs) change depending on the deployment environment they belong to (more info on this in D4.11). At the beginning of the integration work in Pharaon, the number of connections between technologies and the number of exposed service instances increased as well. However, when those integrations were realized only the content of the messages within those connections changed (if there is no additional special requirement to set up a new connection). Additional connections were realized in Pharaon pilots to accommodate for the additional onboarded technologies (from WP6 cascade calls) so here there might be a need to potentially expose new Pharaon services serving those new connections and data flows. The number of such connections is not big and is not done arbitrarily or dynamically since first all the prerequisites (such as legal and GDPR-related requirements) have to be met, proper consents and agreements signed, and only then some (minimally required) data can be exposed via Pharaon endpoints. To enable the dynamic and automatic process to connect new services over the public Internet to serve any Pharaon data from any Pharaon pilot would be a breach of privacy and ethical concerns, so some of the work in the technical direction in this sense might never be deployed in full Pharaon production environment but only in the integration (test) environment where the data being exchanged is not the data from the real users. The Pharaon follows the GDPR principles and only the needed information is exchanged following the consent and the legal aspects. In theory, much more is possible in technical terms but since in Pharaon pilots we deal with real user data we limit the data exchange strictly to what is legal and allowed.

5.2 Service registry considerations for Pharaon

The conceptual illustration of the service registry in Pharaon is given in the following figure (Figure 5.1).

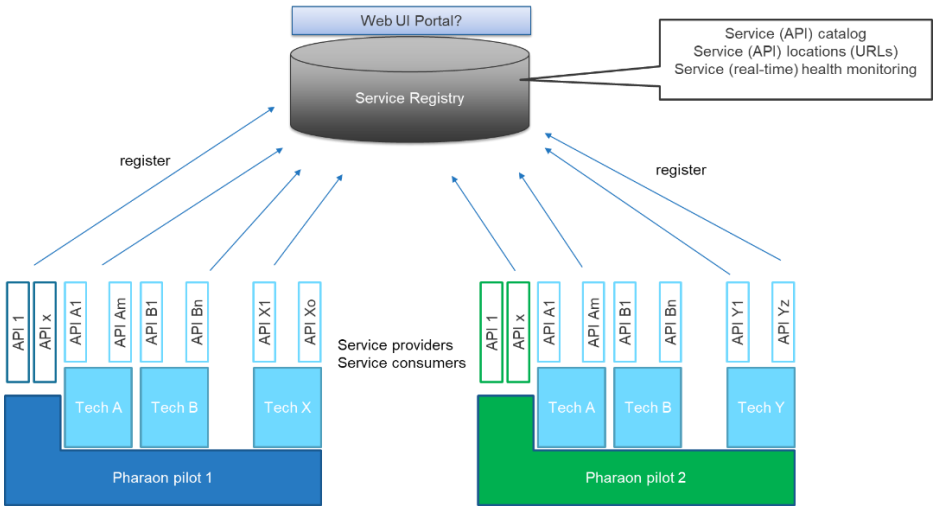


Figure 5.1 Service registration in Pharaon

Common service registration types and properties are summarized in the following figure (Figure 5.2).

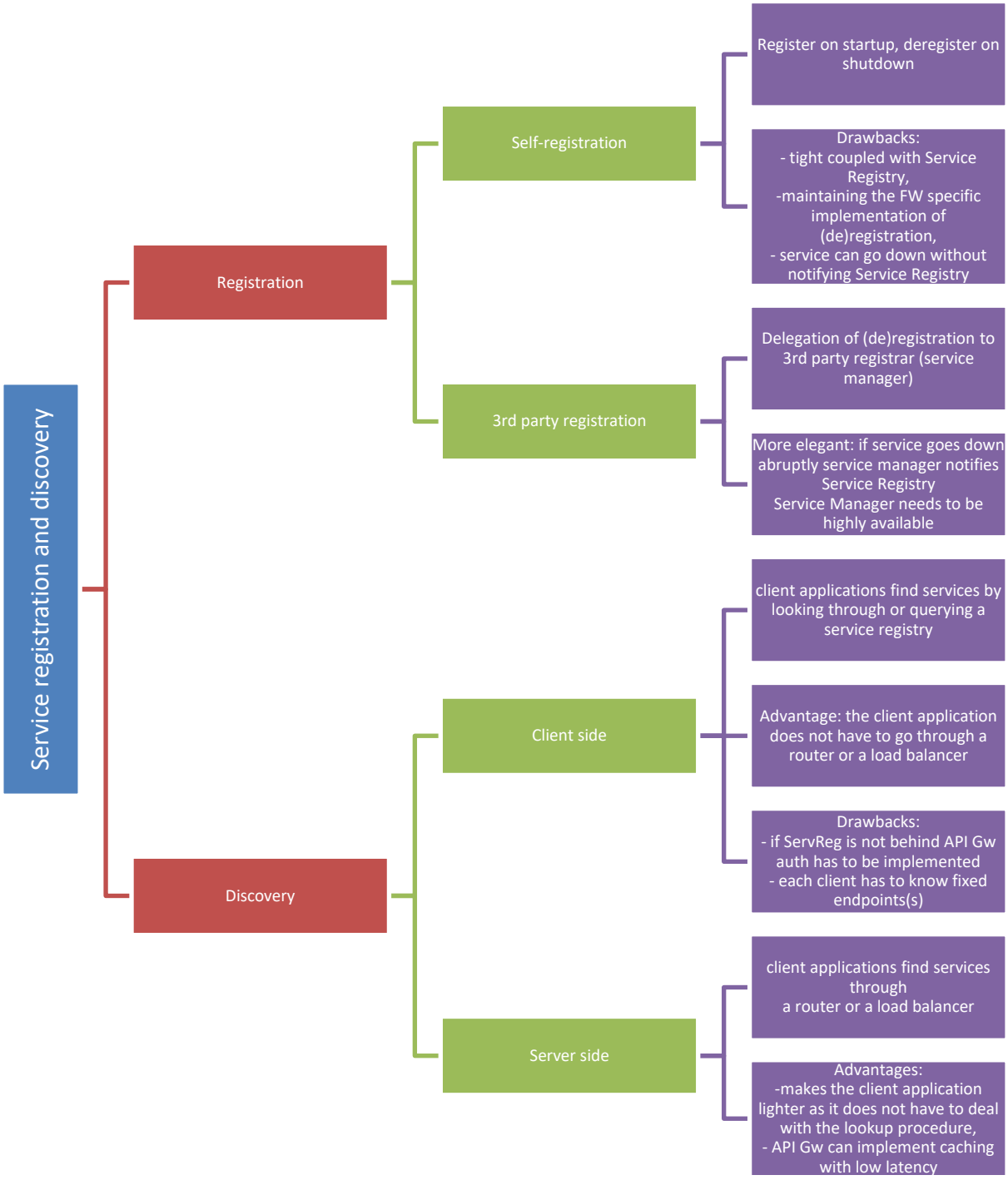


Figure 5.2 Service registration and discovery: types and properties

A more in-depth view of the Client-side and Server-side service discovery is given in the following table Table 5.1).

Table 5.1 Client-side and Server-side service discovery comparison

Service Discovery	Pros	Cons
Client-side	• load-balancing decisions can be application-specific • less complicated and can be handled easily • does not have to route the client application or request through a router or load balancer • avoids the extra steps in the middle	• couples the client with the service registry, so client-side discovery logic has to be implemented for every client/programming fw • makes application management complicated as the microservices architecture works with diverse technologies, frameworks, and tools of different languages • the client has to make two calls to reach the target microservice
Server-side	• does not involve the client in exploring • creates an abstract between the client and server-side • No need to implement the discovery logic separately for each language and framework that the service clients use • client needs to make only one call to request services, without having to be involved in looking up available instances	• needs to be set up and managed • need to implement the load balancing mechanism for the central server

5.2.1 Tools comparison

	Consul	zookeeper	etcd	eureka
Home page	https://www.consul.io/	https://zookeeper.apache.org/	https://etcd.io/	
Last version (in August 2022)		3.8.0	3.5	https://spring.io/projects/spring-cloud-netflix
GitHub page	https://github.com/hashicorp/consul	https://github.com/apache/zookeeper	https://github.com/etcd-io/etcd	https://github.com/Netflix/eureka
License	Mozilla Public License 2.0	Apache 2.0	Apache 2.0	Apache 2.0
Service health check	Service status, memory, hard disk, etc.	(Weak) long connection, keepalive	Connect Heartbeat	Configurable support
watch support	Full quantity/support long polling	stand by	Support long polling	Support long polling/most incremental
Self-monitoring	metrics	—	metrics	metrics
Safety	acl/https	acl	https support (weak)	—
spring cloud integration	supported	supported	supported	supported

5.3 Container registry in Pharaon

With the Docker Container Registry integrated into GitLab, every GitLab project can have its own space to store its container (including Docker) images (more info https://docs.gitlab.com/ee/user/packages/container_registry/, available also for free community edition of GitLab used in Pharaon).

In Pharaon, we use GitLab which gives GitLab a private container registry out of the box, as shown in the following figure (Figure 5.3).

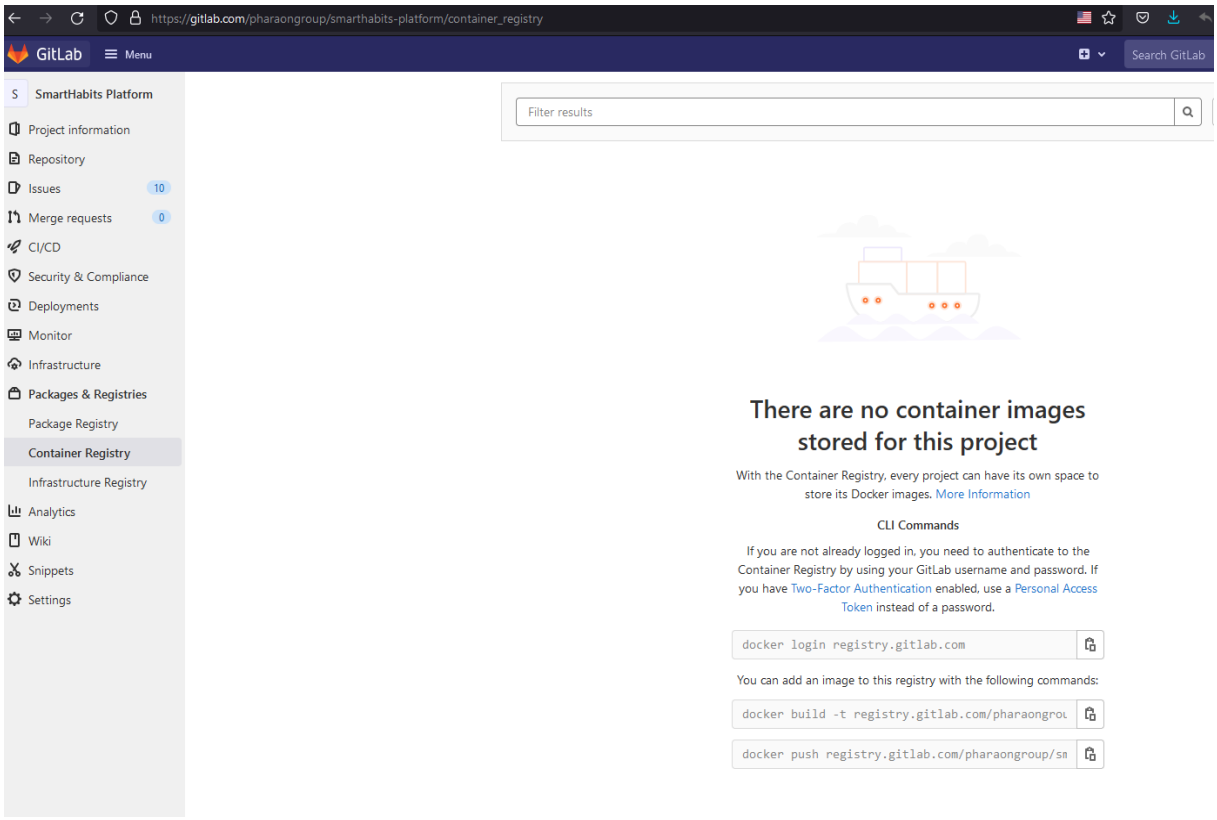


Figure 5.3 GitLab Container Registry

To push the docker image to the Container registry the command is as follows:

```
docker push pharaon.registry_name/group_name/project_name/image_name
```

6 API and service orchestration in Pharaon

6.1 API orchestration in SmartHabits Platform (used in SLO and IT pilots)

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.

Since the main emphasis in Pharaon is on the integration of different (mature) technologies, the technology's internal service and API orchestration is here out of scope. The integration endpoints in Pharaon and external technology interfaces are described in more detail in D5.1 and D5.2 and

summary visualizations are given also in this document in Appendix A: High level view of the public integration interfaces in Pharaon.

To make the integration work as seamless as possible a single API endpoint was created to be exposed and offered to both data providers (data sources for SH Platform; IoTool Platform) and data consumers (Discovery Platform).

Pharaon technology solution	SmartHabits Platform integration sandbox	SmartHabits Platform production sandbox
Data receiving endpoint(s)	test.smarthabits.com.hr:8883 (mqtts)	smarthabits.com.hr:8883 (mqtts)
Data exposing endpoint(s)	test.smarthabits.com.hr (https)	smarthabits.com.hr (https)

6.2 Service Orchestration

For Service Orchestration, in Pharaon, we have implemented custom software (entirely described in D4.8) that works with Camunda (Open Source BPMN Engine).

For Service Orchestration we envision the ability for a domain expert user to visually describe a business process, connecting elements together (using BPM notation), including services from Service Directory, user tasks, timers and other BPMN elements.

In design-time, we offer the Process Designer, which is a complete tool for creating a business process from scratch using BPMN notation. The BPMN engine (Camunda) works with an XML file that is the output of the process, according to the design provided by the user. Process Designer works closely with the Designer API to manipulate the XML file that is sent to Camunda when the user finishes designing the process and wants to start using the process in run-time.

In design-time, there is also a connection with the Service Registry, to allow the user to select the available/registered services in the platform and connect them. It is possible, for example, get the output of a given service and pass that data as the input of another service.

When in run-time, a user has published the process and Camunda takes care of executing it, following the specification designed by the user. It works closely with the Task Manager to provide input/outputs from humans (users) and with the API Gateway for service-to-service authentication (if required).

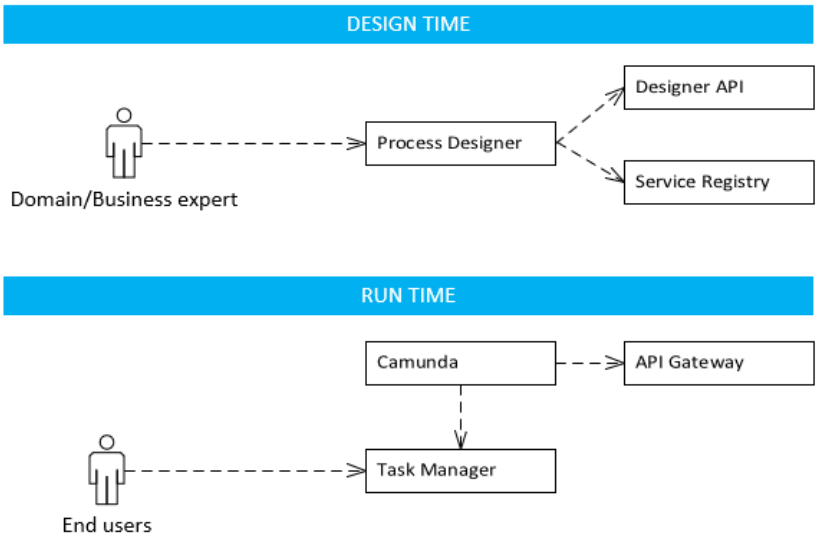


Figure 7.1 Service Orchestration overall architecture

7 Benchmarking of project management tools

Tools for project management are not the primary focus of WP4, however, since they are essential for the alignment of the technical work and "orchestration" of the technical work (being done in WP3, WP4 and WP5), the analysis and some considerations are reported in this deliverable to make them explicit. These tools can also be considered as supporting tools for overall Pharaon project management being under the umbrella of WP1.

To give the complete picture of the tools survey in Pharaon, it is worth repeating in this deliverable as well that the tools survey for all DevSecOps Service Development Life Cycle phases was done within D4.1 and is reported within Pharaon's Developers Handbook at the following link: https://gitlab.com/pharaongroup/developers-handbook/-/wikis/home#tools-survey

7.1 Needs

The Pharaon project, due to its convolution, timespan and profusion of professionals and pursuits, requires appropriate and accurate monitoring and control, in particular for the developments. This can be achieved with the use of specialized tools, many of which are available in the market so that they do not require the expenditure of development effort within the scope of the Pharaon project in activities that would not be the priority of Pharaon.

This approach has already proved fruitful with the creation of a dedicated repository to upload the specific developments made during the project, GitLab,[7] in January 2021 (Month 14 of the project). After considering several possible repositories, the project partners agreed to use GitLab, with a differentiating factor for that choice being its easy integration with Git and other technologies used for programming. The repository started seeing use to store the source code, executables, and further

[7] https://gitlab.com/pharaongroup/

material that proved development effort. ENT created the environment and performed management tasks on it, while other day-to-day activities on GitLab are performed by Indra. The partners involved in developments contribute with content.[8]

As this is demonstrated as a success case, the partners considered the advantages of using other management tools in aspects where Pharaon can use more accurate monitoring of its progress. Specifically, these two needs are identified:

- A tool to implement the agile methodology, as defined in Work Packages 3 and 5, which is also compliant with the mitigation plans of the project.[9]
- A tool to monitor the status and progress of the developments, to control both the technological progress of each pilot and the activity of each partner (while only the stable *results* of that same development are uploaded to the GitLab).

7.2 Tool selection criteria

A benchmarking has been made, trying to identify the tools covering the features required by Pharaon.

The preferences for both tools include:

- Free use/**open source** tools are favoured over proprietary ones.
- **Special licenses** for open source projects, free of cost, etc. will be considered.
- Easy **integration** with other technologies is a priority, considering that the project uses a variety of technologies in the different pilots.
- **Easy to use** to ensure wide adoption, especially by the Project partners.
- The inclusion of a **friendly interface** is useful, as it also contributes to the adoption and is related to the next point.
- A moderately-steep **learning curve**, considering that the partners have to get familiar with the processes.
- **Adoption in communities** with widespread operating systems will contribute to further continuity of the results.
- Generation of **executive reports** is a must, so that management tasks can get input from it.

The benchmarking was made starting with Wikipedia,[10] then further researching every potential candidate in their own webpage and documentation to obtain detail and proof on the specific fields. The focus is on the reference points that apply most to Pharaon.

[8] See the development summary in *Deliverable 3.1 Interoperability Platforms Descriptions* for examples.

[9] See project proposal Part B, pages 50, 53, and 67-69 respectively.

[10] https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems and https://en.wikipedia.org/wiki/Comparison_of_project_management_software

7.3 Development programming and monitoring

The number of fields that were studied is bigger than the amount that can be easily red in pages of the current format. Thus, the original table has been divided in several sub-tables with related criteria (e.g., a general table; a table of Kanban features; a table of Scrum features, etc.), with the first column always having the same software solutions in the same order for easy reference.

In some cases, the information is presented as a hyperlink, because that information is not intuitive to be presented in just a few words. E.g.: The cost of Blossom technology is not by product, but by project, and prospective customers must contact the provider. The link leads to the contact form.

The acronym "N/C" stands for "Not confirmed." Data in the table refer to information from December 2021.

Software	Main features	Main cons	Git integration	Web-based	SaaS	License	Cost	Authentication
MS Excel	Widespread. Easy to use. Easy to adopt. Some partners will have already paid for it.	Local; does not integrate automatically. Not oriented for the specific features.	No	No	No	Proprietary	•Basic: 4.20€/user month. Limited features. •Standard: 10.50€/user month. More features. •Premium: 16.90€/user month. Advanced protection and management. •Applications 365: 8.80€/user month. Limited, different plan.	Windows, Kerberos, SQL Server, OLEDB/ODBC
Google Sheets	Easy to use. Easy to adopt. Collaborative use. Available online.	Does not integrate automatically. Not oriented for the specific features.	Yes, not native	Yes	Yes	Proprietary	•Free	Oauth, API, native
Jira	Favored for agile. Highly customizable. Integration with	UI can be cluttered and confusing. Limited file size oupload. Reports are not reusable	Yes	Yes	Yes	Proprietary	•Free: 10 users, 1 site, 1 project. •Standard: $70 month, 10,000 users.	Form, OpenID, OAuth, LDAP, Shibboleth via

	third parties. Support for Roadmap. Useful for different user types.	(graphics cannot be downloaded as image). Integration with other applications. Difficult in mobile.					•Premium: $140 month, multi-project, advanced features. •Enterprise: ask.	plugin, others via Crowd
Clickup	Customizable: Boards, workspaces and lists. Simultaneous work.	Steep learning curve.	No	Yes	Yes	Proprietary (Free)	•Free: 100 MB storage. •Unlimited: $5/user and month. •Business: $9/user and month. •Enterprise: ask.	token, OAuth
Redmine	High compatibility. Extensible with plugins.	Difficult to use. Steep learning curve. Non-customizable workflows.	No	Yes	Yes	GPL	•Standard: $65, single server, lifetime use, 1-year periodic updates. •RedmineUP: $130, supports all plugins. •Business: Ask. Unlimtied server license.	Form, OpenID, LDAP, Shibboleth, others
OpenProject	Modular. Configurable.	Steep learning curve.	Yes	No	No	GPL	•€357 per user and year + €150 installation service optional + €980/month premium service.	Form, public key, 2 factor, LDAP
Asana	Higly customizable. Recursivity in tasks. Dashboard: Customizable, efficient, easy.	Free up to only 15 users (but open source). Workflow is not efficient. Does not support Scrum or Kanban.	Yes	Yes	Yes	Proprietary	•Free: basic, limited features, 15 users. •Premium: €13.49 month, more users and features. •Business: €30.49 user, more features. •Enterprise: ask.	Oauth, token, OpenID

Pivotal Tracker	Easy to track progress. Open API to expand. Cheaper than JIRA.	Does not integrate easily with other tools. Does not support Gantts. Does not support dashboards. User interface is slower and more difficult than Jira.	Yes (GitHub confirmed)	Yes	Yes	Proprietary	•Free: 15 users, 5 projects. •Startup: $10/month, 10 users. •Standard: $6.50/user and month, volume discount. •Enterprise: ask.	Two factor
SpiraTeam	Several graphs, charts and reports with no config or customization. Built-it manager of HHRR & time tracker. Document versioning, tagging and linker. Integrated testing. Support for regulated processes and workflows.	Not support visual workflows. Not support Gantt. Not support extensive customization. Not support extensive integration. Steep learning curve.	Yes (Plugin)	Yes	Yes	Proprietary	•Starting from $42/month, subscription. •$46.66 per concurrent user per month, cloud; save 10% billed annually. •$1,260,00 per concurrent user, download.	LDAP, others
Blossom	Oriented to software development. Design focused on clarity.	Expensive. Cannot interact with task (add, remove, assign, etc) Requires integration with Slack.	No	Yes	Yes	Proprietary	•A sk.	N/C
Assembla Tickets	Easy to use. High security.	Documentation is boresome, should have videos.	Yes	Yes	Yes	Proprietary, hosted, free for open	•Starter: $12/user and month. Max 5 users, 5 GB, limited features. •Enterprise cloud: $19/user	SecureGit, Form, OpenID, OAuth, LDAP.

						source projects.	and month. Max 500 GB, unlimited users, more features. •Enterprise self-hosted: From $960/year for 5 users.	

	Main features					Project Management								
Software	**Document management**	**Workflow system**	**Reporting & analyses**	**Budget management**	**Time tracking**	**Agile**	**Collaborative software**	**Gantt charts**	**Milestone tracker**	**Project portfolio management**	**Issue track system**	**Scheduling**	**Resource management**	**Traditional methodologies**
MS Excel	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Google Sheets	Yes	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Jira	Yes, with extension	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes, with extension	Yes
Clickup	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Premium	Yes	Yes	Yes	Yes
Redmine	No	No	No	No	Yes	Yes	No	Yes	Yes	No	No	No	No	No
OpenProject	Yes	Yes, customizable	Yes	Yes	N/C	N/C	N/C	N/C	No	No	No	No	No	No
Asana	No	Yes	Yes	No	Yes	No	Yes	Yes, with extension	Yes	Yes	No	Yes	Yes	No

Pivotal Tracker	Yes	Yes	Yes	No	No	Yes	Yes	No	No	No	Yes	Yes	No	N/C
SpiraTeam	Yes	Limited	Yes, dash-board	No	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes
Blossom	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No	Yes	No	No	Yes
Assembla Tickets	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

	KANBAN FEATURES									
Software	**Kanban board**	**Collaboration Tools**	**Dependency Tracking**	**KPI Monitoring**	**Milestone Tracking**	**Multi-Board**	**Prioritization**	**Roadmapping**	**Task Management**	**Time Tracking**
MS Excel	Yes, not native	No	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	No
Google Sheets	Yes, not native	Yes	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	No
Jira	Yes	Yes	Yes	Yes, with extension	Yes	Yes	Yes (not intuitive)	Yes	Yes	Yes
Clickup	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Redmine	Yes	N/C	N/C	N/C	N/C	Yes	N/C	N/C	N/C	N/C
OpenProject	Yes	Yes	No	No	No	No	Yes	Yes	Yes	No
Asana	No	No	No	No	No	No	No	No	No	No
Pivotal Tracker	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C
SpiraTeam	Yes	Yes	No	No	No	No	No	No	No	No
Blossom	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C
Assembla Tickets	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C

	SCRUM FEATURES												
Software	**Supports Scrum**	**Backlog Mana-gement**	**Collabo-ration Board**	**Daily Reports**	**Iteration Manage-ment**	**KPI Moni-toring**	**Mile-stone Track**	**Prioriti-zation**	**Progress Tracking**	**Release Planning**	**Road-map**	**Sprint Plan**	**Task Mana-gement**
MS Excel	Yes, not native	No	No	No	No	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native
Google Sheets	Yes, not native	No	No	No	No	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	Yes, not native	No
Jira	Yes	Yes	Yes	Yes	Yes	Yes, with extension	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Clickup	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Redmine	N/C	No	No	No	No	No	No	No	No	No	No	No	No
OpenProject	Yes	No	Yes	No	No	No	No	No	Yes	Yes	Yes	No	Yes
Asana	No	No	No	No	No	No	No	No	No	No	No	No	No
Pivotal Tracker	Yes	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C
SpiraTeam	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No
Blossom	No	No	No	No	No	No	No	No	No	No	No	No	No
Assembla Tickets	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C

Table 7.1: Development programming and monitoring tools.

7.3.1 Suggested Development programming monitoring tool

After this analysis, the suggested development and progress monitoring tool is **Jira**.[11] Developed by Atlassian Corporation Plc and released in 2002, Jira is described as an issue tracking product that also allows for bug monitoring and agile project management. Jira covers most of the functionalities for project monitoring, including, in particular, dashboard creation and KPIs, along with tools oriented to agile development.

Jira is considered to be easy to integrate, when compared to other alternatives. It is widely documented, with examples of most activities, and it has a big community, that provides support for developers and users.

Although Jira is technically a proprietary software application, there are free-use options that can be applied within the scope of Pharaon.

The technical requirements for Jira are moderate:

- Server:
  - Java: JRE / JDK: Java 8 or later.
  - HDD: 10 GB.
  - CPU: Quadcore 2 GHz.
  - RAM: 6 GB.
- Client:
  - Web browser supporting JavaScript.

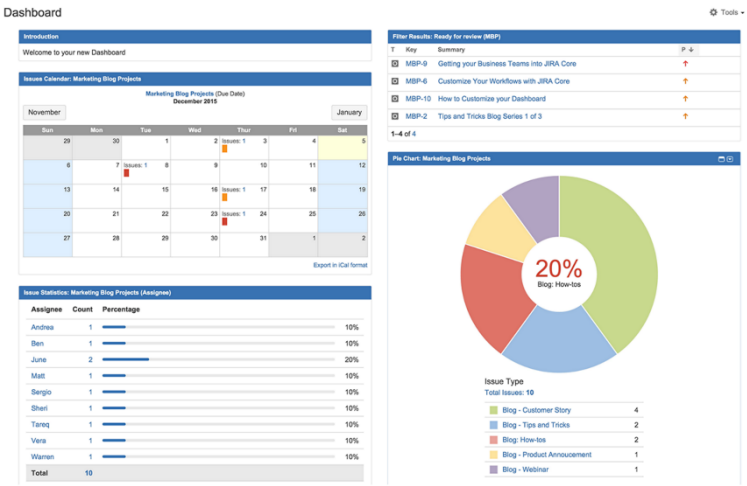


Figure 7.1: Example of Jira dashboard.

[11] https://www.atlassian.com/software/jira

7.4 Agile methodology tools

As in the previous table, the number of analysed fields is bigger than the amount that can be easily red in pages of the current format. Thus, the original table has been divided in two sub-tables, each with related criteria, with the first column always having the same software solutions in the same order.

Again, in some cases, the information is presented as a hyperlink, because that information is not intuitive to be presented in just a few words; and the acronym "N/C" stands for "Not confirmed." Data in the table refer to information from December 2021.

Software	Main features	Main cons	Git integration	License	Cost	Web based	Kanban support
Kanbanize	Oriented to teamwork.	Expensive.	Yes	Proprietary	•Free: 3 projects, 20 MB files. •Pro: €8.75/month. No limits, more features. •Enterprise: €20.75/month. Advanced features. •Corporation: Ask.	Yes	Yes
Trello	Visual, collaborative, easy to use. Attractive, useful UI. Simple to keep track of items in orderly way. Customizations are fast for everyone. Oriented to software development.	Considered the best integration tool.	Yes	Proprietary	•Free: 10 boards, 50 automated commands/month. •Business: $10/user month. More features and unlimited users, unlimited storage. •Enterprise: Ask.	Yes	Yes
Zoho Projects	several options, Gantt, Kanban, etc. Reports, time tracking and templates.	Customization sub-par. Not very intuitive.	Yes	Proprietary	•Free: 10 boards, 10 MB, 50 automated commands/month. •Business: $10/user month. More features and unlimited users, storage. •Enterprise: Ask.	Yes	Yes

Jira	Oriented to software development. Favored for agile. Highly customizable. Integration with third parties. Support for Roadmap. Useful for different user types.	UI can be cluttered and confusing. Limited file size oupload. Reports are not reusable (graphics cannot be downloaded as image). Integration with other applications. Difficult in mobile.	Yes	Proprietary	•Free: 10 users, 1 site, 1 project. •Standard $70 month, 10,000 users. •Premium $140 month, multi-project, advanced features. •Enterprise, ask.	Yes	Yes
MeisterTask	Easy to learn. Easy to coordiante different projects. Orderly interface. Versions customized to different needs.	Pricing. Specifically oriented to automation. Slow load. Some features still missing (e.g. detailed calendar).	Yes	Proprietary	•Free: basic. 3 projects. •Pro: €8.25/month. More features, unlimited projects, higher limits. •Business: €20.75/month. All the features. •Enterprise: Ask.	Yes	Yes
Asana	Useful, ergonomic design. Useful to share files. Useful calendars.	Lacks software development capacities Notifications may not appear properly.	Yes, externally	Proprietary	•Free: basic, limited features, 15 users. •Premium: €13.49 month, more users and features. •Business: €30.49 user, more features. •Enterprise: ask.	Yes	Yes
Leankit	Good for Kanban and agile.	No mobile version.	Yes	Proprietary	•$20 per user and month, billed annually.	Yes	Yes

	Good workflow views.						
Smartsheet	Multiple task views. Easy export. Messaing system. Dashboard automatically updated. Work with different sheets at once.	Changing or updating a system requires changes in each instance.	Yes	Open	•Individual: 13€/month. 10 users, limited skills. •Business: 22€/user and month. Unlimited users. More features. Min 3 licenses. •Enterprise: Ask. •Premier: Ask.	Yes	Yes
Blossom	Oriented to software development. Design focused on clarity.	Expensive. Cannot interact with task (add, remove, assign, etc) Requirse integration with Slack.	No	Proprietary	•Ask.	Yes	Yes
MS Azure Boards	Visual and interactive. Very customizable. Provides cloud storage. Scalable.	Not all the features are easily accessible.	No	Proprietary	•$5.06 per user and month. •First 5 users for free.	Yes	No
HeySpace	Tracks %-complete, milestone, status; file sharing; idea management.	No mobile version; PC can be improved.	Yes, externally	Proprietary	•Free: Up to 5 users, 10 GB. •Premium: $5/user month, more storage, more features.	Yes	No
Projektron BCS	Works with agile and traditional.	Steep learning curve. New features are slow to appear.	No	Proprietary	•$20 per month (no free version but trial available).	Yes	No

Taiga	Easy UI and dashboard.	Expensive.	Yes, externally	GPL (freemium)	•Basic: Free, 3 members, 1 project, 300 MB storage. •Premium: $5/user and month ($7 if monthly). Unlimited members, projects, storage 10GB. •On premise: Ask.	Yes	No
Tuleap	Customizable. Good tracking capacity.	Old-fashioned plugins. Works well for up to 20-30 people.	Yes	GPL	•(Cloud) myTuleap: 9.9€/user and month; min 10 users; all features. •Premium Cloud: 24€/user and month; more services (support, configurable, hosting, etc.) •Ultimate Cloud: Ask. •On premises Access: 11€/user month; minimal features. •On premises Expert: 18€/user month; more features. •On premises Managed: 25€/user month; all the feautes.	Yes	No
Monday.com	User-friendly and customizable Very flexible. Automation in processes. Good search engine.	Not easy to share customization. Requires a lot of CPU. Data loss problem detected.	Yes	Proprietary	•Individual: free, 2 users. •Basic: €8/user month, more storage. •Standard: €10/user month, Chronogram, view, calendar, 250 automations and integrations, combined board. •Pro: €16/user month, private baords, chart, time tracking, higher limits for automations, integrations and combined board. •Corporate: Ask.	Yes	Yes
Kanboard	Functionalities are solid. Good documentation.	Difficult to see overview with priorities and progress. Reports are sub-par.	Yes	Open	•100% free. •Donations accepted.	Yes	Yes
ClickUp	Free. Hierarchic approach	Steep learning curve.	Yes	Proprietary (Free)	•Free: 100 MB storage. •Unlimited: $5/user and month.	Yes	Yes

	to organize. Import from external apps.	Non-intuitive interface.			•Business: $9/user and month. •Enterprise: ask.		
Wrike	Good for team collaboration. Good reports and dashboard. Good configuration capacities.	New features and updates are not properly announced.	Yes	Proprietary	•Free: Minimum features. •Professional: $9.80/user month. More features (Gantt, integrations, more storage). •Business: $24.80/user month. More features: Calendars, reports, time track. •Enterprise: Ask.	Cloud-based	Yes
Kissflow Project	Good workflow design. Good access to history.	Interface and UI sub-par.	Yes	Proprietary	•Basic: $16/user month. Min 10 users. Minimum features. •Advanced: $22/user month. Min 10 users. Advanced features (audit log, advanced integrations, etc.) •Fully Loaded: $30/user month. Min 50 users. More feautres.	Yes	Yes

	Scrum features				Scrum Story features					Scrum Task features			Inte-gration
Software	**Scrum board**	**Custom columns**	**Burndown chart**	**Sprint goal**	**Accep-tance criteria**	**Story points**	**Planning poker**	**Story descrip-tion & com ments**	**Change track**	**Tasks within stories**	**Task claim**	**Auto-story comple-tion**	**Data Export**
Kanbanize	Yes	Yes	Yes	N/C	Yes	N/C	N/C	N/C	N/C	Yes	N/C	N/C	Yes
Trello	Yes	Yes	Via plugin	Yes	Yes	Via plugin	Via plugin	Yes	Yes	Yes	Yes	Via plugin	Yes
Zoho Projects	Yes	Yes	Yes	Yes	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	Yes
Jira	Yes	Yes	Yes	Yes	Via custom field	Yes	Via plugin	Yes	Yes	Yes	Yes	N/C	Yes
MeisterTask	Yes	Yes	No	N/C	Yes	N/C	N/C	Yes	N/C	N/C	N/C	N/C	Yes
Asana	Yes	Yes	Yes	Yes	Yes	N/C	N/C	Yes	N/C	N/C	N/C	N/C	Yes
Leankit	Yes	Yes	Yes	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	Yes
Smartsheet	Yes	Yes	Yes	N/C	Yes	N/C	N/C	Yes	N/C	Yes	N/C	N/C	Yes
Blossom	Yes	Yes	Yes, substitute	N/C	Yes	Yes	N/C	Yes	N/C	Yes	Yes	N/C	N/C
MS Azure Boards	Yes	Yes	Yes	Sprint name	Yes	Yes	N/C	Yes	N/C	Yes	Yes	Yes, with exten-sion	Yes
HeySpace	Yes	Yes	No	Sprint descrip-tion	Yes	No	No	Yes	Yes	Yes	No	No	No
Projektron BCS	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes
Taiga	Yes	Yes	Yes	Yes	Via custom field	Yes	Via plugin	Yes	Yes	Yes	Yes	Yes	Yes

Tuleap	Yes	Yes	Yes	Sprint descrip-tion	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Monday.com	Yes	Yes	Yes	Yes	Yes	Yes	N/C	Yes	N/C	Yes	Yes	N/C	Yes
Kanboard	No	No	No	No	No	No	No	No	No	No	No	No	Yes, command line
ClickUp	Yes	Yes	Yes	Yes	Yes	Yes	N/C	Yes	N/C	Yes	Yes	N/C	Yes
Wrike	Yes	Yes	Yes	N/C	N/C	N/C	N/C	N/C	N/C	Yes	Yes	N/C	Yes
Kissflow Project	Yes	Yes	Yes	N/C	Yes	N/C	N/C	Yes	N/C	Yes	Yes	Yes	Yes

Table 7.2: Agile methodology tools.

7.4.1 Suggested Agile methodology tool

Trello,[12] first released in 2011, is a web-based Kanban-style agile methodology tool, and it is the most used and extended tool for its purpose. Initially created by Fog Creek Software and released in 2011, Trello was spun out into a separate company which was later sold to Atlassian. This contributes to integration with other Atlassian tools, such as Jira: As of 2021, Trello features are integrated in Jira.

Trello is designed to be user-friendly and allows for a simple interface. It is also easy to customize and adapt to the customer's needs.

The technical requirements are easy because Trello is a web-based service provided by Atlassian, thus it only includes this:

- Client: Web app that can be executed in a browser in either a personal computer, smartphone or tablet.

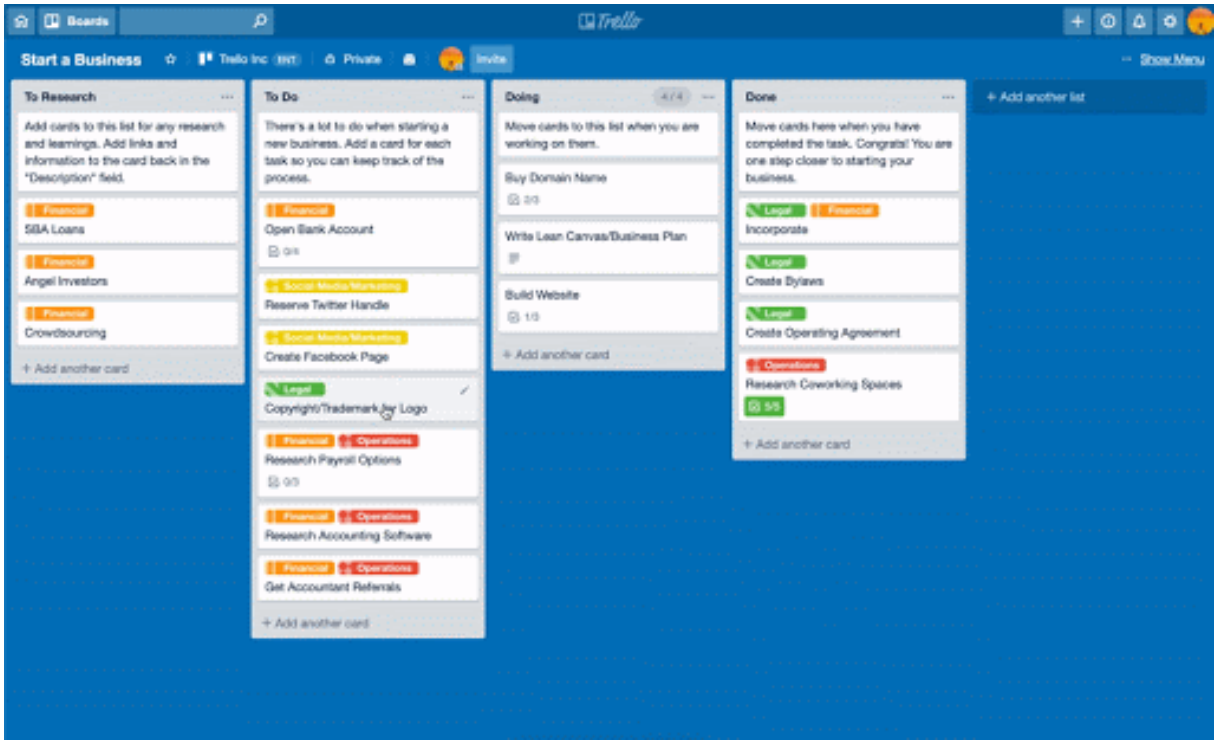


Figure 7.2: Example of Trello board

[12] https://trello.com/

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

7.5 Outcome

A cross-WP strategy was defined to incorporate these tools to the project, with one or more technical partners with effort in WP4 taking care of the following as part of its WP4 activity: Configuration of the tools, deployment, technical maintenance and training.

Meanwhile, other aspects were discussed, including among these how to deal with limitations of the chosen versions (e.g., limited number of users in the free versions), the hosting of the tools within the scope of Pharaon's project (e.g., budget allocation), and in particular the administration of the tools. On this latter subject, there was a proposal of a three-level management system:

- **Technical Manager:** Maximum responsible for monitoring the project, must have access to reports on anything.
- **Pilot coordinators:** Responsible for the monitoring of each of the pilots and to ensure that the information on that pilot is updated.
- **Technical providers:** Responsible for updating the work performance and the repository.

In the end, the technical implementation proved to be more complicated than expected and would have required human resources with greater skill than those who were initially assigned, and the definition of the management required time resources that were better allocated in higher-priority tasks. As the implementation of these management tools was not mandatory for the success of the project, in the end the project scratched the idea in favor of using GitLab’s integrated tools for those purposes. That work is nevertheless documented here, so as to provide proof of where a small amount of the WP4 effort was spent.

8 Conclusions

This deliverable reports on the further considerations and work done concerning service orchestration in Pharaon. The Deliverable furthers the proposed and advocated OpenAPI-based interactive and always up-to-date documentation of the Pharaon REST API. It summarizes the process of integrating new Pharaon APIs and the process of designing the new APIs in Pharaon. It also identifies critical quality properties of the Pharaon APIs. It gives results of the GraphQL exploration for the Pharaon. It reports on the container orchestration tools, monitoring orchestration tools, and service registry considerations for Pharaon. To provide support with high-level management of the technical integration and service orchestration work the benchmarking of the project management tools was done within this deliverable as a result of the discussions and a request from cross technical WP SWAT1 task force led by the project Technical Manager.

Appendix A: High level view of the public integration interfaces in Pharaon

Following images are taken from D5.2[13] (describing high-level architectures of each Pharaon pilot) and are referenced in later chapters of this document.

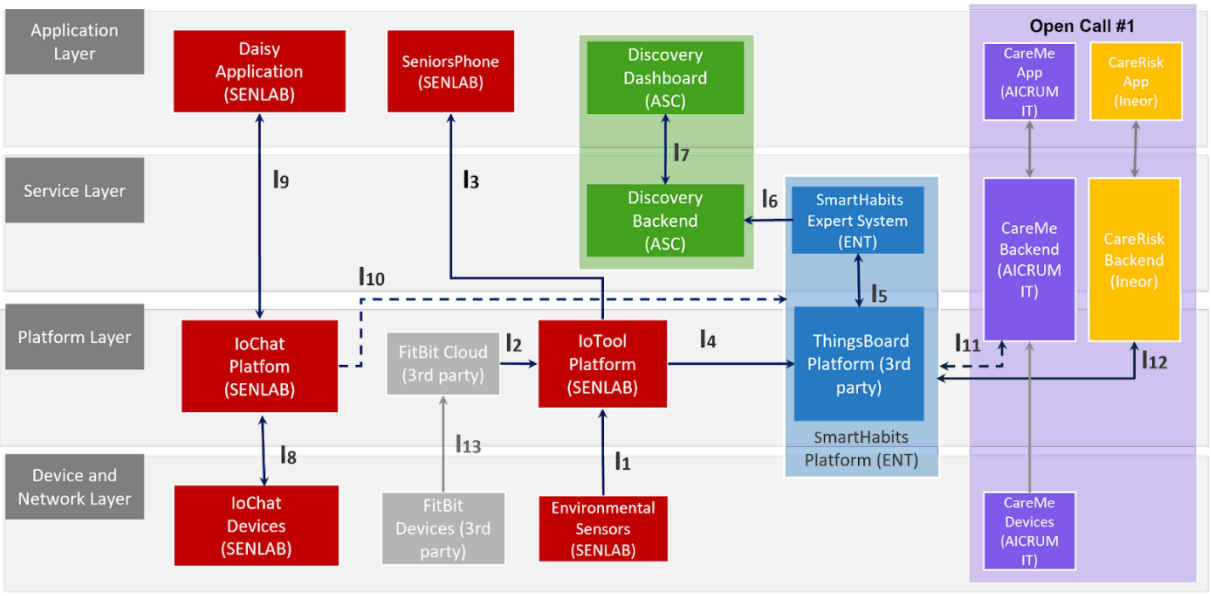


Figure 6: Slovenian pilot high-level architecture (from D5.2)

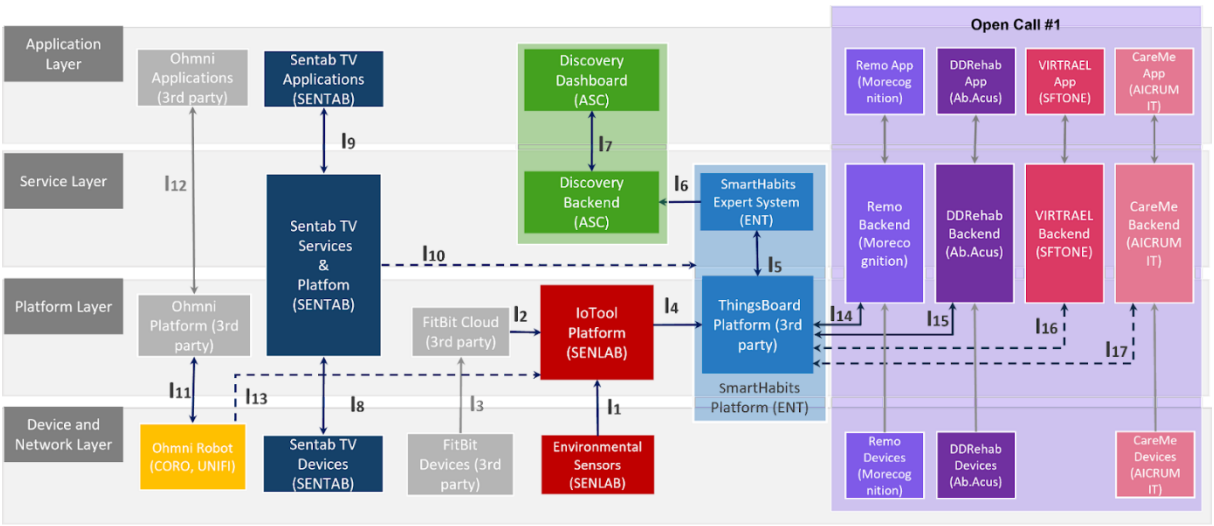


Figure 5: Italian pilot high-level architecture (from D5.2)

[13] Version of document on Nov 29th 2022 (not final submitted version)

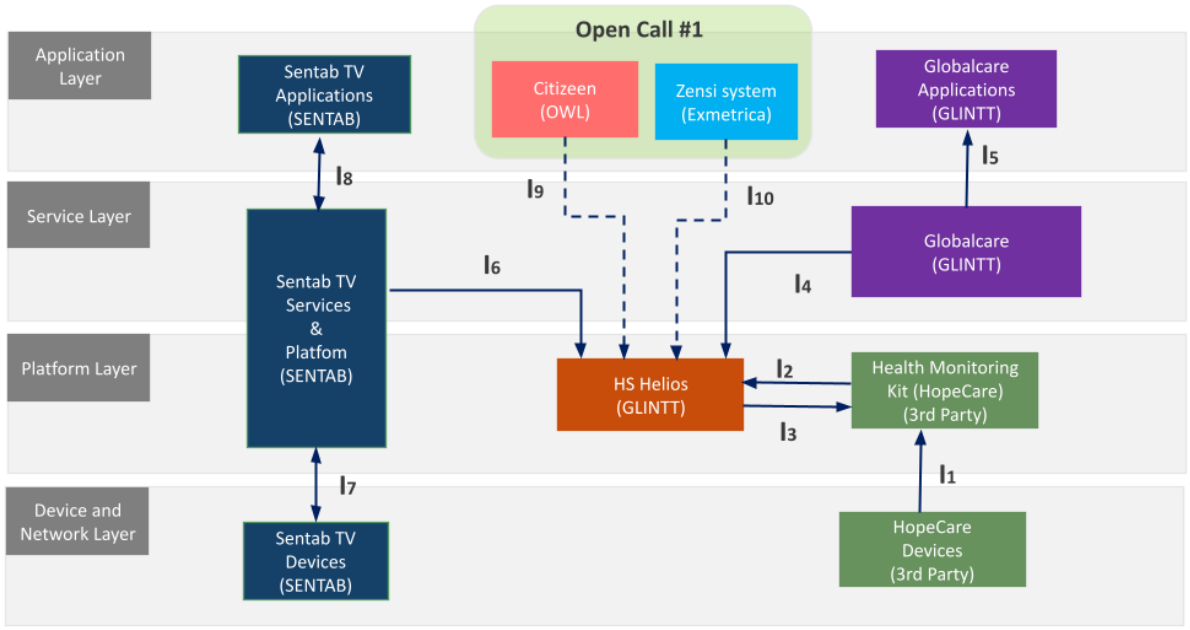


Figure 7: Portuguese pilot high-level architecture (from D5.2)

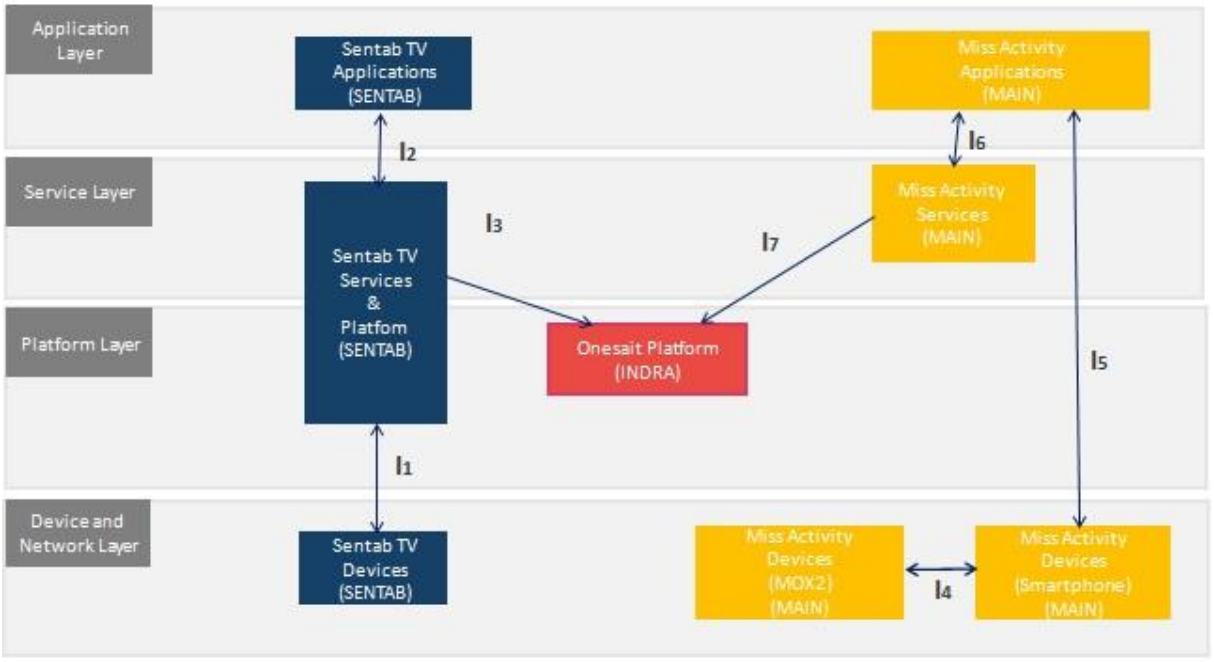


Figure 8: Andalusian pilot high-level architecture (from D5.2)

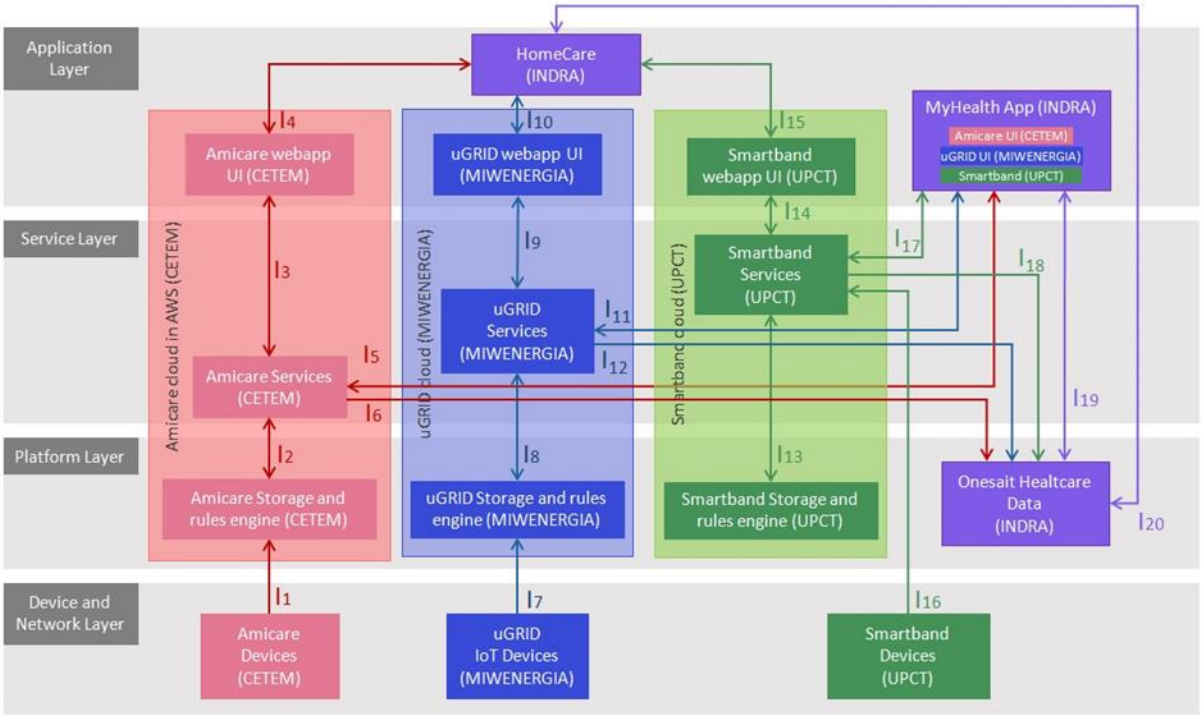


Figure 9: Murcia pilot high-level architecture (from D5.2)

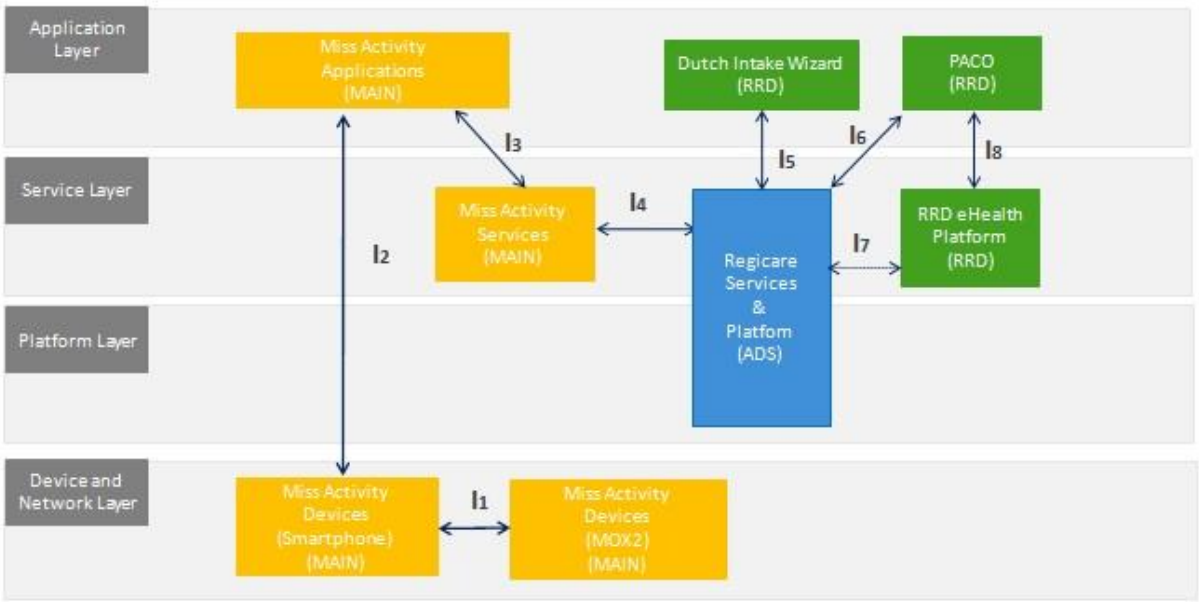


Figure 10: Dutch pilot high-level architecture (from D5.2)