PILOTS FOR HEALTHY AND ACTIVE AGEING

Grant Agreement: 857188

D4.4 Service orchestration support tools - first

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:	**D4.4**
Deliverable title:	Service orchestration support tools - first
Deliverable version:	0.8
Work Package number:	WP4
Work Package title:	Technology Support Tools
Due Date of delivery:	M16 (March 2021)
Actual date of delivery:	M16 (March 2021)
Dissemination level:	Public (PU)
Type	Other (O)
Editor(s):	Andrej Grgurić (ENT)
Contributor(s):	Andrej Grgurić (ENT) Danny Pape (ASC)
Reviewer(s):	Miran Mošmondor (ENT) Rafael Maestre Ferriz (CETEM) Miguel Ángel Beteta (CETEM)
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	26.10.2020	ENT	Table of Contents proposal and initial content considerations for each chapter
0.2	15.02.2021	ENT	Service orchestration in Pharaon, OpenAPI in Pharaon, Containers orchestration and applicability in Pharaon. Document introductory chapters.
0.3	18.02.2021	ASC	Monitoring orchestration in Pharaon
0.4	08.03.2021	ENT	Overall quality improvements. REST APIs and related description languages chapters added. Policies-based control and OPA consideration for Pharaon.
0.5	16.03.2021	ENT	Service orchestration chapter update. Updates based on internal review. Progress beyond state-of-the-art chapter content added.
0.6	29.03.2021	ENT	Abbreviations chapter update. Benefits of OPA in Pharaon and Using OPA with Docker chapters added.
0.7	30.03.2021	ENT	Internal review updates
0.8	31.03.2021	ENT	Wrapping up the document

PM efforts per beneficiary having contributed to the deliverable.

#	Partner	PM effort in D4.4
28	ENT	1.5
29	ASC	0.4
7	CETEM	0.2
TOT	Pharaon Consortium	2.1

Acknowledgement: This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 857188.	**Disclaimer**: The content of this publication is the sole responsibility of the authors, and in no way represents the view of the European Commission or its services.

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.

The focus of the work reported in this deliverable is to report on the initial work done regarding technology support tools with respect to service orchestration.

Giving that the Pharaon system architecture is being defined in parallel and that WP5 work has been delayed, the scope of this deliverable is reduced (considering initial project planning in September 2020) and the efforts planned in this task will be minimal in this first deliverable iteration, thus preserving the resources for the next phase when concrete needs and priorities will be clearer and effort spending more effective.

Progress beyond the state of the art and play

This document reports on the initial in-depth analysis of the selected tools related to the process of service orchestration within Pharaon. After the specifics of both the realization and the deployment of the Pharaon input technologies became more apparent, as well as with the definition of Pharaon reference architecture and the work on concrete architectures of Pharaon pilots, additional information crucial for this task became known. One such piece of information is that the input technologies during the work in WP3 will be hosted by each partner separately. In general, only the APIs are to be exposed for the integration. While this is being done in parallel with the activities reported in this document, the following steps will inevitably have to include WP5 planning to provide the best possible project value. Having this in mind, this document reports on the common ("best") and state-of-the-art processes, in-depth analysis of the selected and prioritized tools, and initial concrete work to be used as a lighthouse example for future activities (e.g., adopting and making available the interactive REST API (REpresentational State Transfer Application Programming Interface) documentation based on the most popular OpenAPI specification).

Contents

Figures

Tables

Acronyms & Abbreviations

Term	Description
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CPU	Central Processing Unit
DevSecOps	Development, Security and Operations
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
JSON	JavaScript Object Notation
k8s	Kubernetes
LDAP/AD	Lightweight Directory Access Protocol/Active Directory
OData	Open Data Protocol
OPA	Open Policy Agent
OS	Operating System
PromQL	Prometheus Query Language
RAML	RESTful API Modeling Language
REST	REpresentational State Transfer
RPC	Remote Procedure Call
SOAPs	Service Orchestrion and Automation Platforms
WADL	Web Application Description Language
WP	Work Package
WP2	Pharaon WP2 Pilot and User Requirements, Ecosystem Architecture
WP3	Pharaon WP3 Secure Interoperability Solution
WP4	Pharaon WP4 Technology Support Tools
WP5	Pharaon WP5 Technology Ecosystem Integration
WP6	Pharaon WP6 Ecosystem Evolution
WSDL	Web Services Description Language
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1 Introduction

1.1 Overview

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 in order to identify customizations to be done. As interoperability is in the focus most obvious and straightforward way of connecting different platforms and solutions will be via web APIs. For this to be possible, the proper documentation and web addresses have to be available. Such documentation with possible concrete adapters or SDKs will be available via Developer Handbook (described in D4.1).

Task T4.2, where this deliverable is the first step, will focus on facilitating the coordination of the communication between Pharaon services and orchestration as straightforward as possible. Special attention will be put towards investing efforts to achieve the biggest impact and added value.

Pharaon ecosystem, encompassing several AAL ecosystems realized via different Pharaon pilots, is defined as a sociotechnical system. When focusing only on the software part residing in the cloud it can be said that it is a "system of systems".

1.2 Relation to other tasks and deliverables

The main inputs to this deliverable come from WP2 and WP3 activities, in the future also more input will be collected from WP5.

Apart from the activities from deliverables:

- D2.1 User and pilot requirements
- D2.2 Pharaon initial ecosystems architecture

activities reported in this deliverable are in high correlation to parallel activities in WP3 and WP4 and, more specifically with deliverables (due at the same time, M15):

- D3.1 Interoperability Platforms Descriptions – intermediate (Type: Report)
- D3.3. Semantics and Usability Report – first (Type: Report)
- D3.11 User interoperability layer report – intermediate (Type: Other)
- D4.1 Developer guidelines and templates – first (Type: Other)

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 service orchestration in relation to Pharaon is elaborated. In chapter 3, OpenAPI as REST API documentation specification, tools and usage in Pharaon is explained. Chapter 4 elaborates on containers, container orchestration and gives an analysis of the Kubernetes and Docker Swarm in relation to Pharaon. Chapter 5 addresses monitoring orchestration tools and chapter 6 elaborates and analyzes policy-based control in relation with possible use in Pharaon.

2 Service orchestration in Pharaon

In most definitions of orchestration, the (web) services are in control of a central (web) service which coordinates their operation. The services can, in this way, be connected without being aware that they are part of a more extensive (business) process.

Gartner defines Service Orchestrion and Automation Platforms (SOAPs) as a "workload automation and orchestration tools that enable IT to *design and implement business services through a combination of workflow orchestration, run book automation and resource provisioning across an organization's hybrid digital infrastructure."*[1] SOAPs provide orchestration for applications, IT infrastructure, network services, DevOps, microservices and other use cases with the most common features and capabilities being:

- Graphical workflow designers for assembling cross-platform processes,
- Monitoring and alerting aiming to reduce mean time to remediation, help improve SLAs, monitor workloads in real-time, issues alarms, offer actionable insights, analyze historical system data with ML/AI,
- Resource provisioning across cloud environments according to the need dynamically.

For a Pharaon ecosystem to be aligned when serving the end users, multiple siloed input technologies have to be integrated. Having this in mind, the **Pharaon service orchestration** is defined 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.

Some of the aims of this task are to make possible the orchestration on multiple layers within Pharaon. As the Pharaon pilot technologies have been selected within WP2 their connection is being established within WP3 and initial analysis shows there will be no need to realize the fully automated orchestration of all Pharaon input technologies. The amount of effort to be invested in such automated orchestration would significantly surpass the more pragmatic approach towards facilitating the orchestration by minimizing the redundancies, optimizing and streamlining repeating processes, offering good documentation, code and actionable steps. Apart from that, the ability to monitor service-level performance is high on the list of priorities.

The activities are directed in the definition of the right strategy and processes supported with the most effective tools to facilitate the integration of reliable and high-quality experience across a distributed Pharaon ecosystem.

The essential prerequisite of integration and orchestration is that the input solutions and services are *consumable*. In this sense, the property of being compliant to standards, delivering what they promise and being accessible is vital. The following figure (Figure 2.1) 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.

[1] https://www.advsyscon.com/blog/service-orchestration-automation/

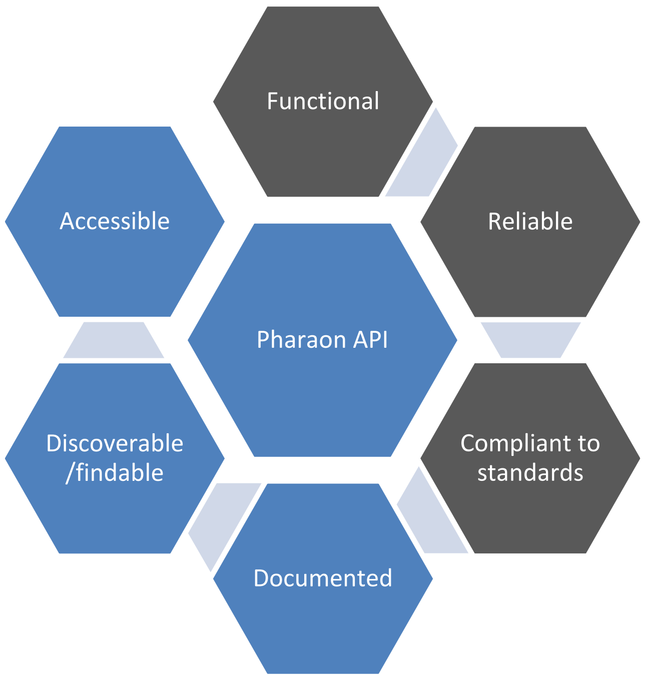


Figure 2.1 Properties of Pharaon APIs

The functional aspect is in most cases predefined and made available with the selection of the input technology solution, as it should be the case with reliability and standards compliance. Within WP4 and the task related to this deliverable, the goal is to advance the later properties such as documentation (vital to make the integration possible), findability (to make the address such as URL of the API explicit) and accessibility (to make the process and all needed information such as authorization, explicit to be able to connect to the API and consume its functionality).

3 REST API documentation

3.1 RESTful APIs

API (Application Programming Interface) allows the two applications (or services) to interact using a defined set of rules or commands. On the web, apart from the main APIs, there are also web service APIs that include SOAP (Simple Object Access Protocol), XML-RPC, JSON-RPC, REST (Representational State Transfer). Unlike the former, REST is not a protocol but a set of architectural principles (software architectural style) which uses a subset of HTTP and it is commonly used to create interactive applications that use Web services. While REST is an "architecture style", RESTful is about putting that style to practice and it typically refers to web services implementing such architecture.

REST APIs are a standardized architecture style for building web APIs using HTTP methods (Figure 3.1).

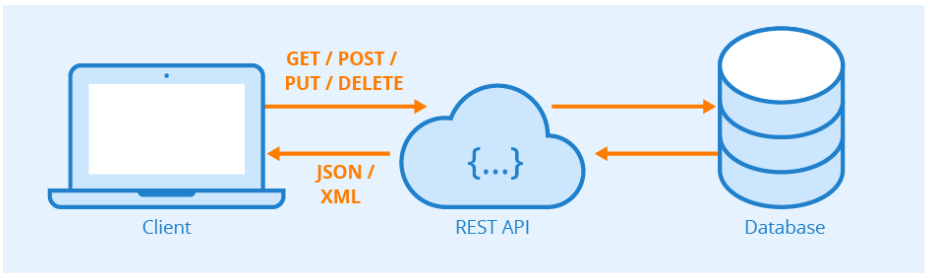


Figure 3.1 REST API [image taken from Seobility[2]]

When a RESTful API is called, the server will *transfer* a *representation* of the requested resource's *state* (representation of the state can be in JSON, XML or HTML format) to the client system.

3.2 RESTful APIs description languages

RESTful (representational state transfer) **API** (application programming interface) **DLs** (description languages) are formal languages designed to provide a (structured) description of RESTful web APIs. Some examples include Web Services Description Language (WSDL), Web Application Description Language (WADL), Open Data Protocol (OData), RESTful API Modeling Language (RAML)[3], API Blueprint[4] and heavily adopted OpenAPI Specification.

The following figure shows the large adoption rate (82%) of the REST-based OpenAPI specification from the State of API 2020 report[5][6], based on responses from 1,500 API developers, architects, testers, and product leads collected from May to June 2020.

---

[2] https://www.seobility.net/en/wiki/REST_API

[3] https://raml.org/

[4] https://apiblueprint.org/

[5] https://nordicapis.com/breaking-down-smartbears-2020-state-of-api-report/

[6]

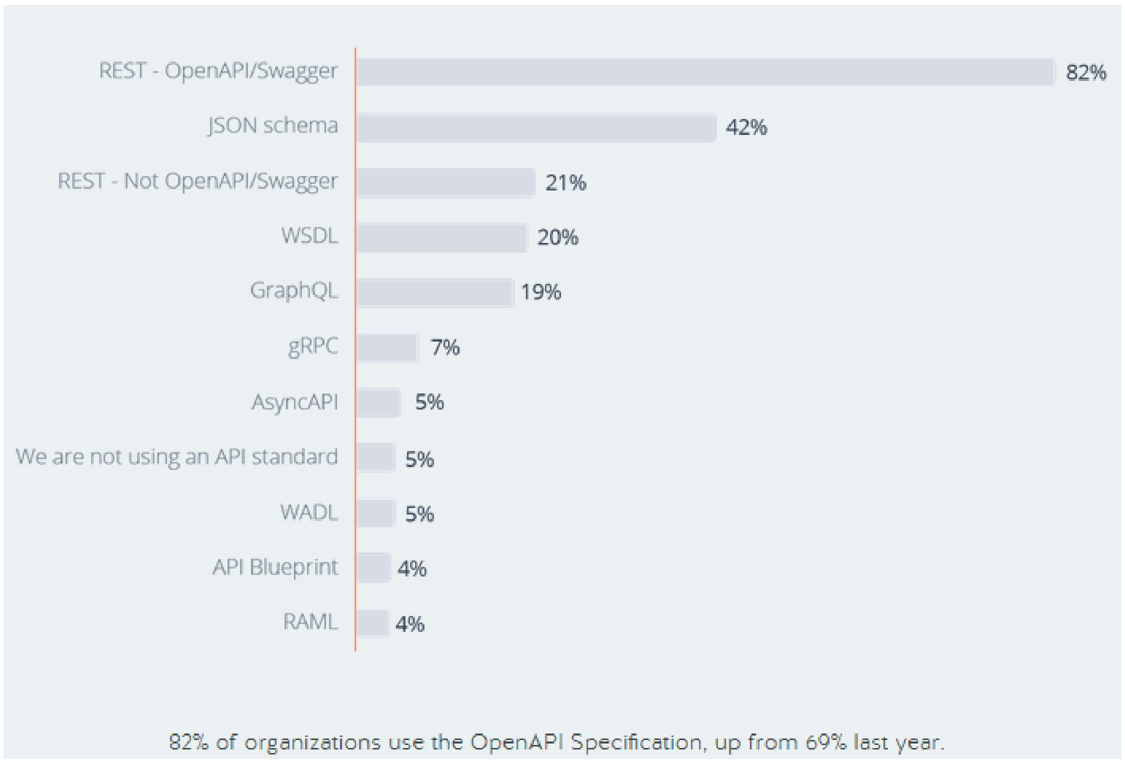


Figure 3.2 State of API 2020 Report[7]

3.3 OpenAPI Specification

The **OpenAPI Specification (currently v3.0.3)**[8], formerly (2009-2017) known as the **Swagger Specification**, is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services.

Some benefits of using OpenAPI include:

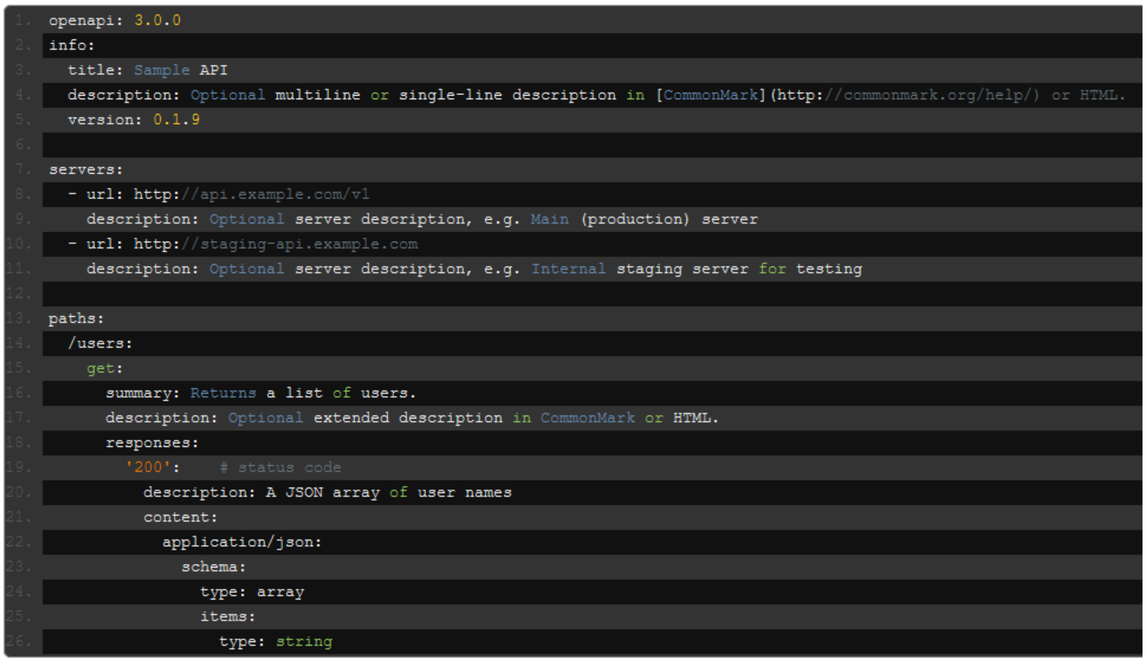
- Language-agnostic interface for describing RESTful APIs,
- Machine-readable and easily interpretable by humans,
- Design first approach: whole API with types and examples for every endpoint can be defined before the start of implementation work. With additional tools, mock-APIs can be generated for testing until full implementation becomes available, thus speeding up the work and making agreements explicit
- Code generators: most languages are supported by their code-generators
- Tooling: big ecosystem of tools created under the Swagger brand that can be used to streamline the work (generating API documentation, tests, mock servers)
- Widely accepted: many developers and organization use it,
- Stable: known under Swagger name until 2017. OpenAPI Initiative is part of the Linux Foundation, which increases trustworthiness,

[7] https://smartbear.com/resources/ebooks/the-state-of-api-2020-report/

[8] https://www.openapis.org, https://swagger.io/specification/, https://spec.openapis.org/oas/v3.0.3

- Reliable single source of truth for APIs: clients and API developers/providers have a good point of reference through the API contract which OpenAPI describes.

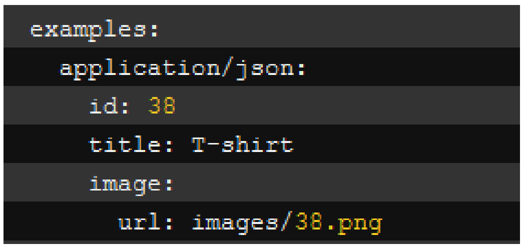
OpenAPI definitions can be written in JSON or in YAML (YAML Ain't Markup Language, as a superset of JSON). An example of OpenAPI 3.0 definition in YAML is given in the following figure (Figure 3.3).



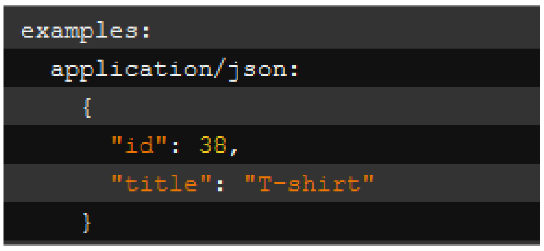
```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Figure 3.3 A sample OpenAPI 3.0 definition written in YAML [image from swagger.io[9]]

YAML and JSON are interchangeable as shown by the example in the following figure (Figure 3.4).



```yaml
examples:
  application/json:
    id: 38
    title: T-shirt
    image:
      url: images/38.png
```



```
examples:
  application/json:
    {
      "id": 38,
      "title": "T-shirt"
    }
```

Figure 3.4 OpenAPI specification in YAML (left) and JSON (right)

3.4 OpenAPI tools

There are different tools around OpenAPI. The most notable Open Source are[10]:

- Swagger Editor[11]: open source editor to design, define and document RESTful APIs
- Swagger UI[12]: visualizes OpenAPI specification definition of RESTful APIs in an interactive UI. Example of a Swagger for Petstore application can be found at following web link https://petstore.swagger.io/#/ [last accessed 8 March 2021].

[9] https://swagger.io/docs/specification/basic-structure/

[10] https://swagger.io/docs/open-source-tools/

[11] https://github.com/swagger-api/swagger-editor

[12] https://github.com/swagger-api/swagger-ui

- Swagger Codegen[13]: open-source code-generator to build server stubs and client SDKs directly from a Swagger defined RESTful API

One repository is https://openapi.tools/, where the tools are grouped into categories as[14]:

- *Converters: tools to convert to and from OpenAPI and other API description formats.*
- *Data Validators: Check to see if API requests and responses are lining up with the API description.*
- *Description Validators: Check if the API description is a valid OpenAPI.*
- *Documentation: Render API Description as HTML (or maybe a PDF) so slightly less technical people can figure out how to work with the API.*
- *DSL: Writing YAML by hand is no fun, and maybe you don't want a GUI, so use a Domain Specific Language to write OpenAPI in your language of choice.*
- *GUI Editors: Visual editors help you design APIs without needing to memorize the entire OpenAPI specification.*
- *Learning: Whether you're trying to get documentation for a third party API based on traffic, or are trying to switch to design-first at an organization with no OpenAPI at all, learning can help you move your API spec forward and keep it up to date.*
- *Miscellaneous: Anything else that does stuff with OpenAPI but hasn't quite got enough to warrant its own category.*
- *Mock Servers: Fake servers that take description documents as input, then route incoming HTTP requests to example responses or dynamically generates examples.*
- *Parsers: Loads and read OpenAPI descriptions, so you can work with them programmatically.*
- *SDK Generators: Generate code to give to consumers to help them avoid interacting at a HTTP level.*
- *Security: By poking around your OpenAPI description, some tools can look out for attack vectors you might not have noticed.*
- *Server Implementations: Easily create and implement resources and routes for your APIs.*
- *Testing: Quickly execute API requests and validate responses on the fly through command line or GUI interfaces.*
- *Text Editors: Text editors give you visual feedback while you write OpenAPI, so you can see what docs might look like.*

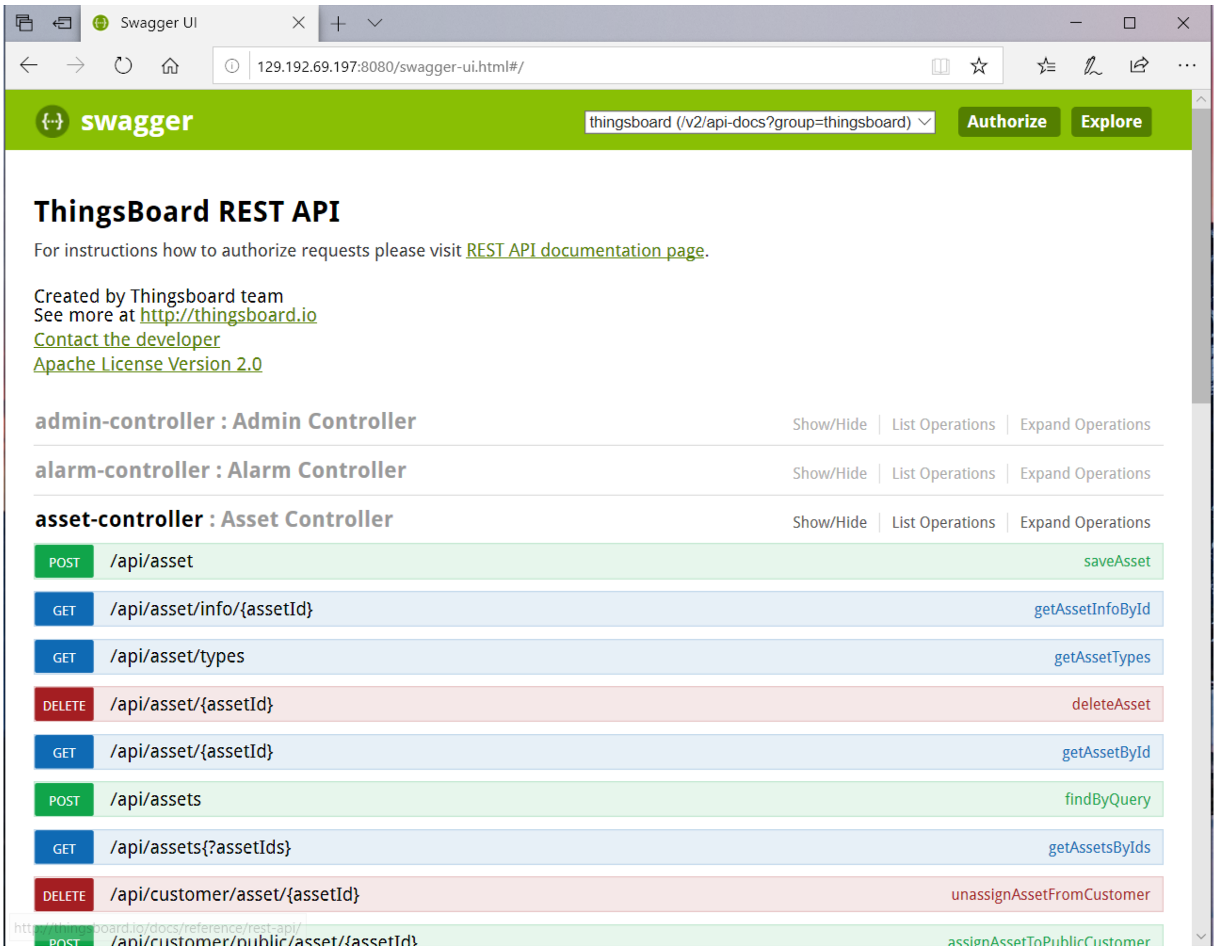
[13] https://github.com/swagger-api/swagger-codegen

[14] Open.API tools: https://openapi.tools/ [Accessed 10.2.2021]

3.5 OpenAPI in Pharaon

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

The concrete application of the OpenAPI for initial Pharaon input technologies is done and screenshots are shown in the following figures (Figure 3.5, Figure 3.6). They are served from ENT private cloud server used as a development environment.


Swagger UI
129.192.69.197:8080/swagger-ui.html#/
swagger
thingsboard (/v2/api-docs?group=thingsboard)
Authorize
Explore

ThingsBoard REST API

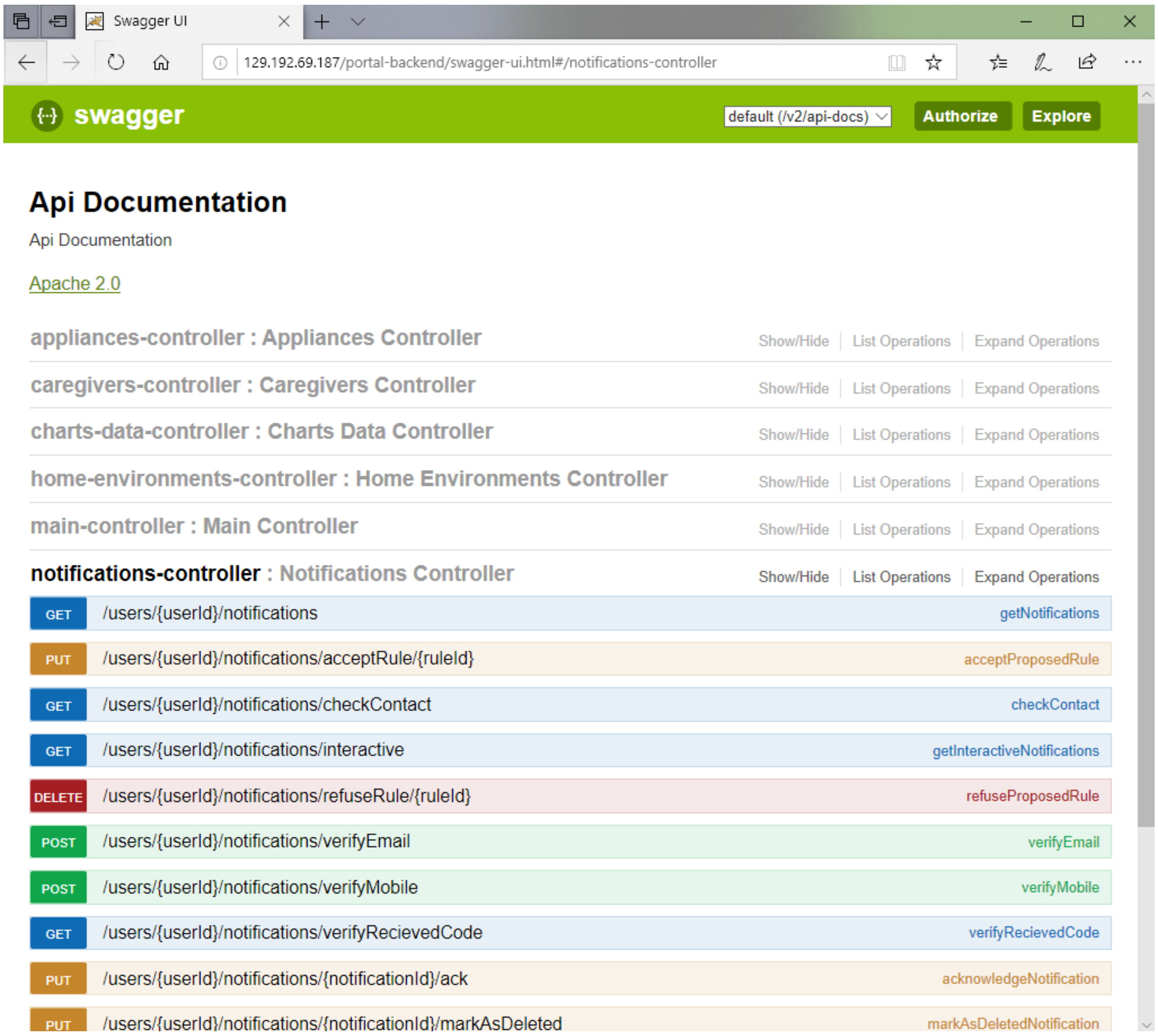
For instructions how to authorize requests please visit REST API documentation page.

Created by Thingsboard team
See more at http://thingsboard.io
Contact the developer
Apache License Version 2.0

admin-controller : Admin Controller — Show/Hide | List Operations | Expand Operations
alarm-controller : Alarm Controller — Show/Hide | List Operations | Expand Operations
asset-controller : Asset Controller — Show/Hide | List Operations | Expand Operations

POST	/api/asset	saveAsset
GET	/api/asset/info/{assetId}	getAssetInfoById
GET	/api/asset/types	getAssetTypes
DELETE	/api/asset/{assetId}	deleteAsset
GET	/api/asset/{assetId}	getAssetById
POST	/api/assets	findByQuery
GET	/api/assets{?assetIds}	getAssetsByIds
DELETE	/api/customer/asset/{assetId}	unassignAssetFromCustomer
POST	/api/customer/public/asset/{assetId}	assignAssetToPublicCustomer

Figure 3.5 OpenAPI UI for ThingsBoard (used in Slovenian and Italian pilot)


Swagger UI
129.192.69.187/portal-backend/swagger-ui.html#/notifications-controller
swagger
default (/v2/api-docs)
Authorize
Explore

Api Documentation

Api Documentation

Apache 2.0

appliances-controller : Appliances Controller — Show/Hide | List Operations | Expand Operations

caregivers-controller : Caregivers Controller — Show/Hide | List Operations | Expand Operations

charts-data-controller : Charts Data Controller — Show/Hide | List Operations | Expand Operations

home-environments-controller : Home Environments Controller — Show/Hide | List Operations | Expand Operations

main-controller : Main Controller — Show/Hide | List Operations | Expand Operations

notifications-controller : Notifications Controller — Show/Hide | List Operations | Expand Operations

GET	/users/{userId}/notifications	getNotifications
PUT	/users/{userId}/notifications/acceptRule/{ruleId}	acceptProposedRule
GET	/users/{userId}/notifications/checkContact	checkContact
GET	/users/{userId}/notifications/interactive	getInteractiveNotifications
DELETE	/users/{userId}/notifications/refuseRule/{ruleId}	refuseProposedRule
POST	/users/{userId}/notifications/verifyEmail	verifyEmail
POST	/users/{userId}/notifications/verifyMobile	verifyMobile
GET	/users/{userId}/notifications/verifyRecievedCode	verifyRecievedCode
PUT	/users/{userId}/notifications/{notificationId}/ack	acknowledgeNotification
PUT	/users/{userId}/notifications/{notificationId}/markAsDeleted	markAsDeletedNotification

Figure 3.6 OpenAPI UI for SmartHabits (used in Slovenian and Italian pilots)

4 Container orchestration tools

4.1 Containers and container orchestration

Unlike virtual machines, which virtualize hardware, containers virtualize operating systems (CPU, memory, storage, and network resources at the OS-level) and are more portable and resource-efficient (Figure 4.1).

Notable benefits of using containers include:

- process, memory, filesystem isolation,
- horizontal elasticity,
- increased portability: on different operating systems and platforms,
- greater efficiency: quicker to be deployed, scaled,
- less overhead: lighter than virtual machines as they do not include operating system images,
- more appropriate for DevSecOps and CI/CD: they run the same regardless of where they are deployed and accelerate development, test and production cycles by having all dependencies packaged together, thus being very popular with microservices architectures,

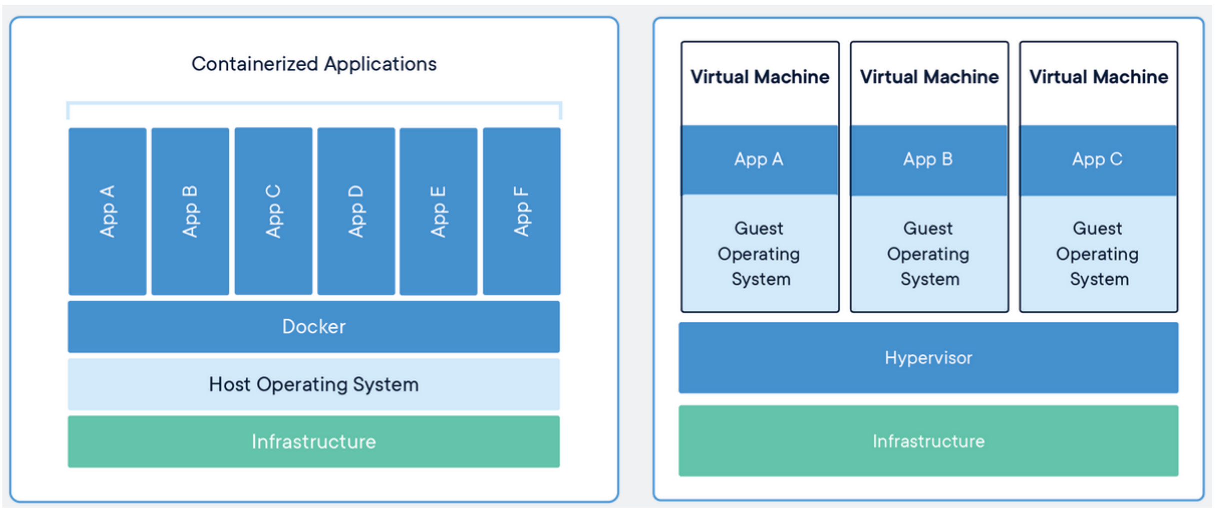


Figure 4.1 Containers vs. Virtual Machines [image taken from[15]]

Docker is the most popular open platform for developing, shipping, and running applications. Docker provides the ability to "*package* and *run an application in a loosely isolated environment called a container*"[16]. Docker uses client-server architecture (Figure 4.2), where the *client* talks to the Docker *daemon* (responsible for building, running, and distributing Docker containers). A container *image* is a read-only blueprint (template) for building Docker containers stored in a Docker *registry* (DockerHub[17] as a public registry is being hosted on a public cloud). The container is a runnable instance of an image.

[15] https://www.docker.com/resources/what-container

[16] https://docs.docker.com/get-started/overview/

[17] https://hub.docker.com/

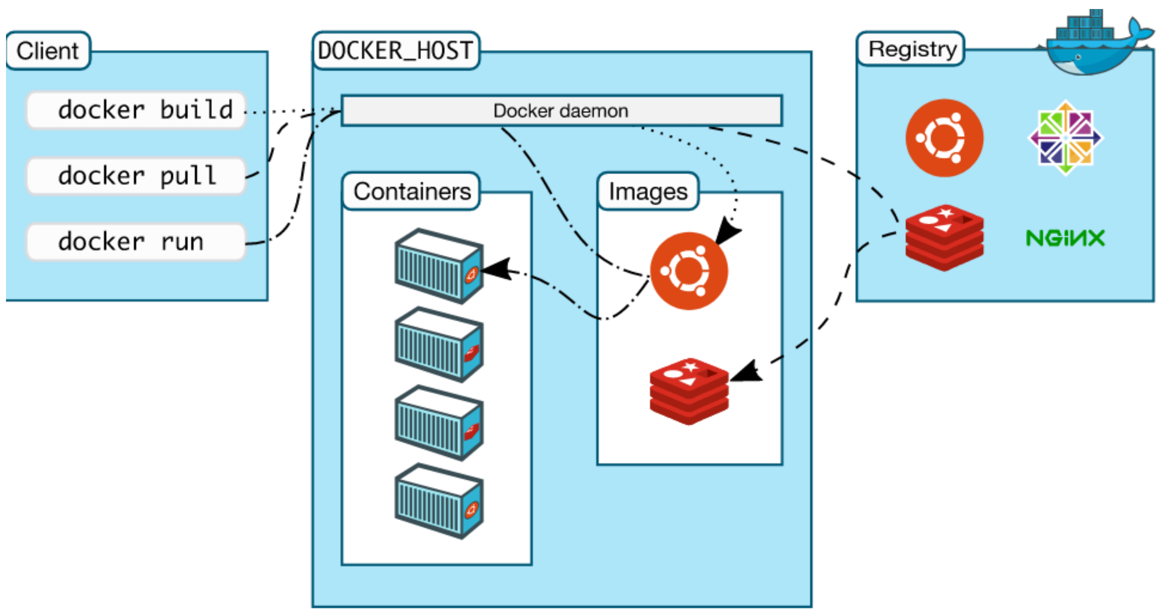


Figure 4.2 Docker architecture [image taken from [18]]

Container orchestration focuses on managing the lifecycles of containers. Including provisioning and deployment, scaling, allocation of resources between containers, load balancing, secure communication between containers, health monitoring of containers and hosts on which containers run, etc.

[18] https://docs.docker.com/get-started/overview/

4.2 Kubernetes and Pharaon

Kubernetes[19] (a.k.a. k8s), as an open-source orchestration tool for containerized services, aims to manage a cluster of hosts on which containers are deployed, monitored and scaled.

The following figure (Figure 4.3) shows a k8s cluster consisting of a set or worker machines (nodes) that run containerized applications. Control Plane manages worker nodes and makes a global decision about the cluster.

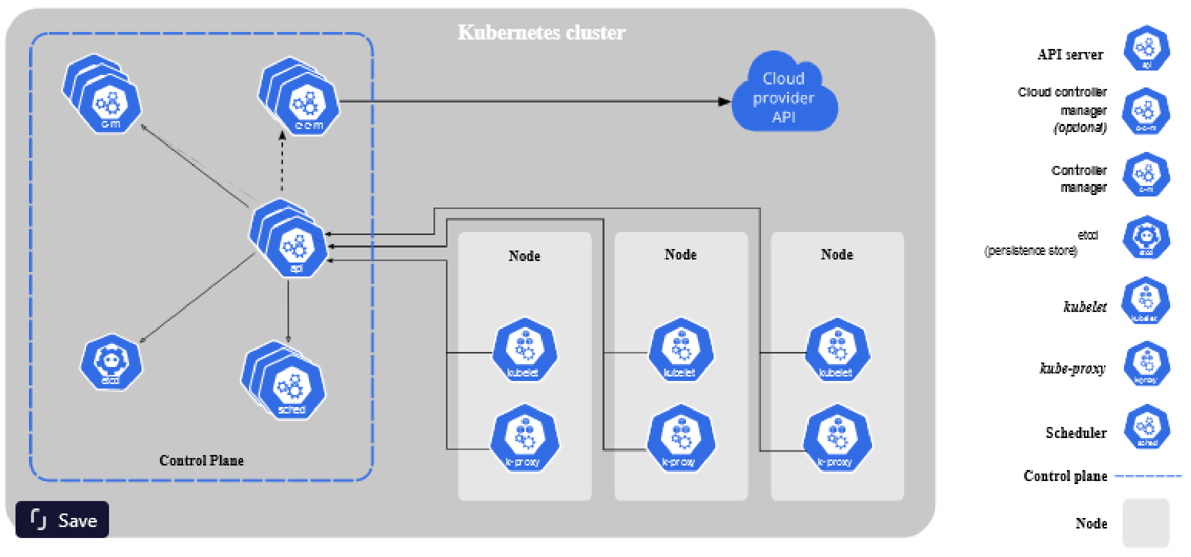


Figure 4.3 Diagram of a Kubernetes cluster [image taken from official documentation[20]]

Kubernetes features include automated rollouts and rollbacks, service discovery and load balancing, storage orchestration, self-healing, secret and configuration management.

The following table (Table 4.1) gives some considerations (as pros as cons) on using k8s in Pharaon.

Table 4.1 Using k8s in Pharaon

Pros	Cons
• Improved deployment • Having **infrastructure as data:** All the resources in Kubernetes (including Pods, Configurations, Deployments, Volumes…), can be expressed in a simple YAML file • Possibility to keep K8S YAML files in Git repositories and to use **GitOps** (Git Operations Version Control) to increase transparency, avoid ambiguity • **Scalability** becomes easier	• A steep learning curve, vast ecosystem, a huge number of 3rd party providers • k8s manages only containers, not the VMs running them so VM maintenance, cloud load balancers, storage still needs to be done regardless of using k8s • Not all Pharaon input technologies are containerized and wrapping them into containers may involve a lot of effort, an effort that should primarily be spent on Pharaon integration work (focusing largely on integration via web APIs)

[19] https://kubernetes.io/

[20] https://kubernetes.io/docs/concepts/overview/components/

• **Security** enforcement becomes easier (using tools such as conftest[21] allowing writing tests for k8s configurations r Open Policy Agent[22] to check security policies) • Easy integration with different cloud providers	• many Pharaon input technologies already run on private servers or will be deployed via different cloud providers. Pharaon pilot sites will have different deployment architectures and will include a combination of public, private and hybrid cloud infrastructure environments. • Most of the Pharaon participants do not have experience with k8s, which may lead to bottleneck problems and delays if using k8s across all pilot sites • Managed k8s solutions give more out of the box but also requires careful analysis of the possible best-fit for Pharaon

4.3 Docker Swarm and Pharaon

Docker Swarm[23], as an open-source container orchestration platform, is more lightweight than k8s. Docker Swarm turns a set of Docker hosts into a virtual, single host. It is a more straightforward solution but also easier to learn and quicker to start with.

With easier setup, Docker Swarm could be a good fit for a Pharaon technology provider giving more control and easier running of different Docker containers. All technology providers and developers within Pharaon will decide whether to use some tool or not, and no tool will be enforced.

[21] https://www.conftest.dev/

[22] https://www.openpolicyagent.org/

[23] https://docs.docker.com/get-started/swarm-deploy/

5 Monitoring Orchestration Tools

Monitoring tools are needed to orchestrate data from various sources. For this specific use case, Prometheus and Grafana are an established solution to orchestrate data from different sources and to monitor them consistently. The following subsections provide more details about Prometheus and Grafana.

5.1 Data analysis with Prometheus

Prometheus is reliable open-source monitoring and alerting tool that currently has a very active developer and user community. This enables the consortium to have a continuous stable version that can be worked with. Prometheus scrapes data from different sources and records them as numeric time-series data. The recorded real-time metrics can then be queried from the internal time-series database by applications via HTTP and enables real-time alerts. In addition to that, Prometheus offers also

- **Queries** with an own query language, PromQL (Prometheus Query Language), which can be used to select and aggregate data. PromQL has been specially adapted to the convention with a time-series database and offers time-related queries.
- **Visualization** nodes that enable to visualize data via a built-in expression browser and/ or a Grafana integration
- **Time-Series storage** in an efficient custom format
- **Alerting** based on PromQL to provide notifications for alerts, etc.
- Many **existing integrations** of exporters that are needed to analyse data.
- **Extensions**, such as custom alert managers, visualisation tools, microservice support, etc.

A Prometheus server is standalone, which is not depending on network storage or other remote services. For developers, it supports a broad range of client libraries for all common programming languages (official and unofficial).

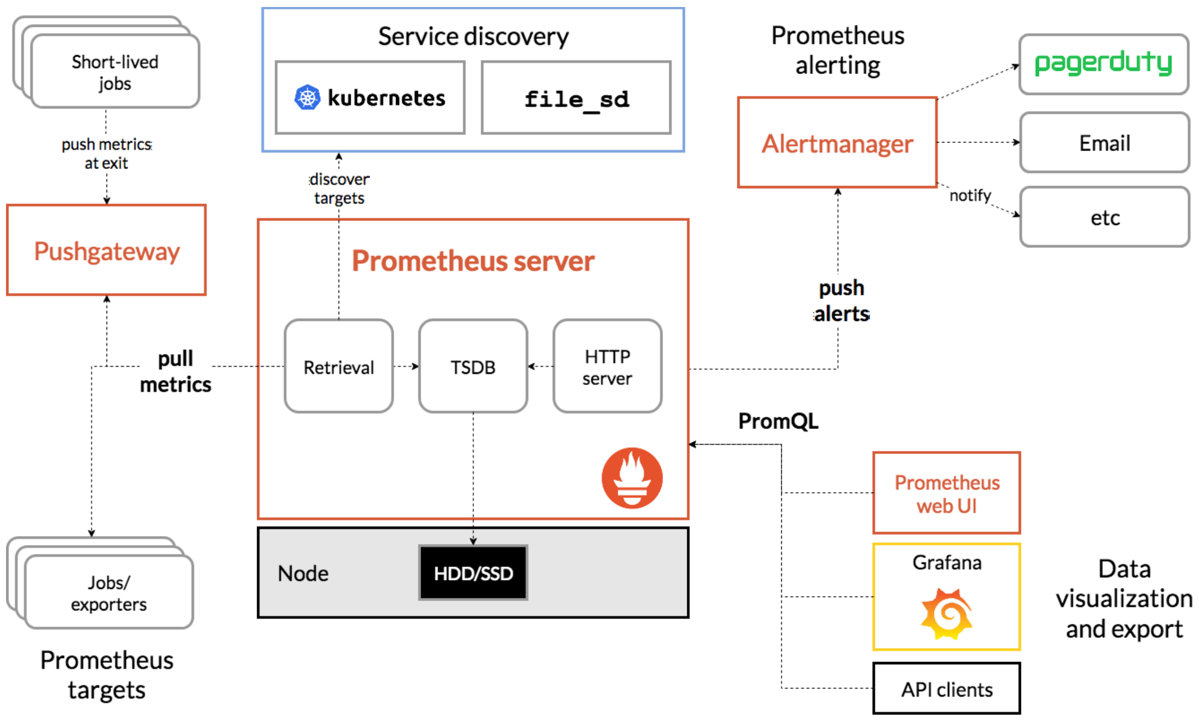


Figure 5.1: Prometheus architecture [image from[24]]

5.2 Data Visualisation with Grafana

Prometheus is fundamentally not intended as a dashboarding solution. Through the integration of Grafana, specific queries can be displayed graphically and compiled into dashboards. Grafana is a multi-platform open-source web application to visualize data from different data sources in charts, graphs, etc. (see Figure 5.2). It must be connected to a data source, such as Prometheus (see Section 5.1) and can be configured to intended needs. Grafana provides a broad range of functionality to handle data differently. It allows exploring data in different ways, e.g., through ad-hoc queries and dynamic drilldowns. Views can individually be split to get an optimized view, different time ranges can be compared separately, and data sources can be mixed to compare the data in the same graph. Out of these customizable possibilities, thresholds can be defined to notify the user for alerts as soon as the threshold has been met. Grafana has built-in support for Prometheus, which includes:

- a query editor with metric name lookup
- templating queries for generic dashboards
- alias patterns for short, readable series names

There are also existing a broad range of plugins for Grafana that offers GUI panels, additional data sources, etc.

[24] https://www.programmersought.com/article/16474076685/

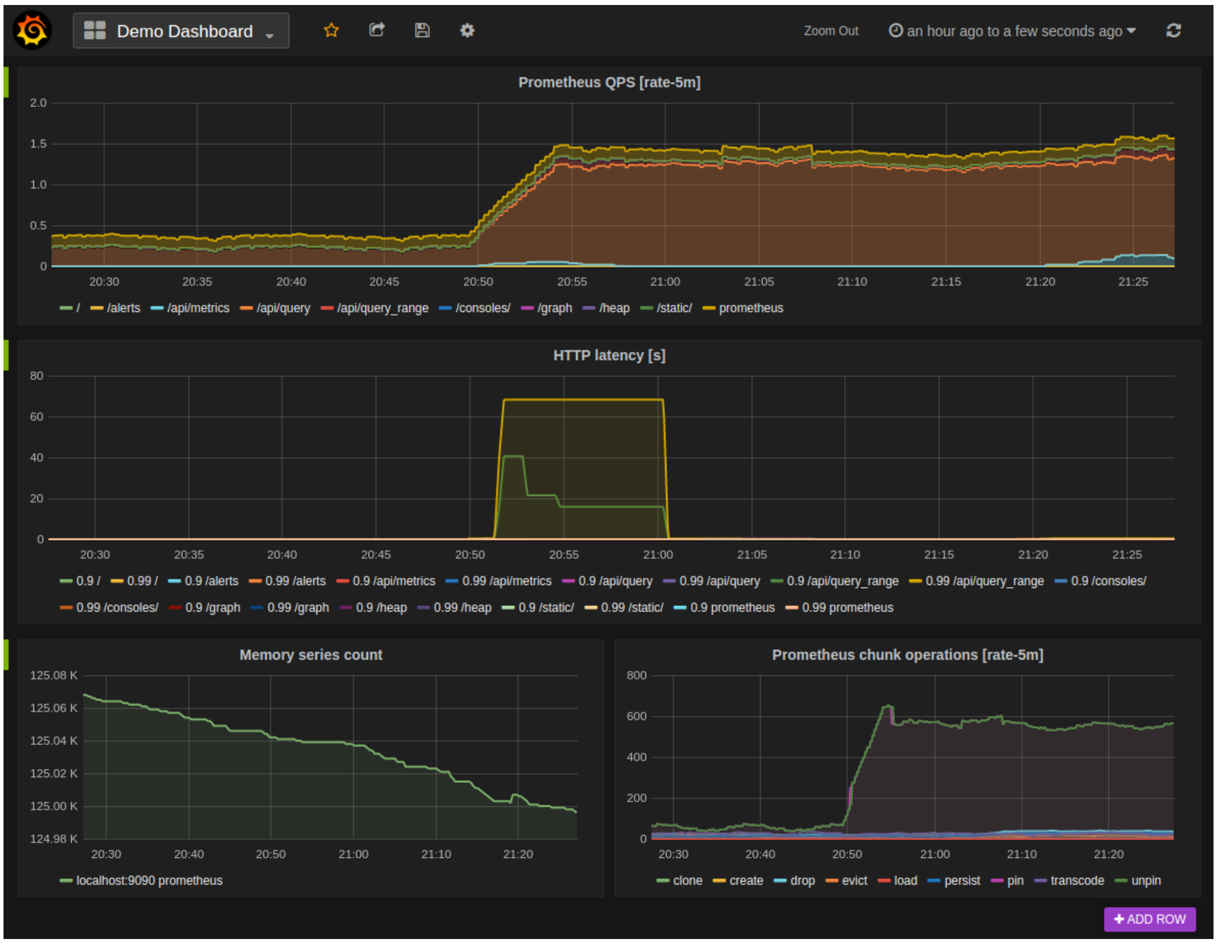

Demo Dashboard
Zoom Out
an hour ago to a few seconds ago
Prometheus QPS [rate-5m]
/ /alerts /api/metrics /api/query /api/query_range /consoles/ /graph /heap /static/ prometheus
HTTP latency [s]
0.9 / 0.99 / 0.9 /alerts 0.99 /alerts 0.9 /api/metrics 0.99 /api/metrics 0.9 /api/query 0.99 /api/query 0.9 /api/query_range 0.99 /api/query_range 0.9 /consoles/
0.99 /consoles/ 0.9 /graph 0.99 /graph 0.9 /heap 0.99 /heap 0.9 /static/ 0.99 /static/ 0.9 prometheus 0.99 prometheus
Memory series count
localhost:9090 prometheus
Prometheus chunk operations [rate-5m]
clone create drop evict load persist pin transcode unpin
ADD ROW

Figure 5.2: Grafana example

5.3 Monitoring and Pharaon

Pros	Cons
• Prometheus is containerized standalone toolkit, that is not dependent on other microservices. • This technology is mainly based on web requests, that supports both containerized microservices and webservices. • A customizable toolkit to analyze data in various ways fits to all Pharaon pilots • The use of time-series data in Prometheus fits to the health records and other data that are used in Pharaon. • Both Prometheus, as well as Grafana, are highly extendable with plugins, which allows to use established third-party developments instead of new developments.	• The required query language PromQL is very specific, that must be learned to handle data scrapers. • Most of the Pharaon participants do not have experience with this technology which may lead to bottleneck problems and delays if using Prometheus and Grafana across all pilot sites. • The Prometheus server must fetch its own measurement data from the various servers. Time points must be defined by the participants themselves. • Additional Plugin solutions give more out of the box but also requires careful analysis of the possible best-fit for Pharaon

6 Policy-based control for cloud native environments

6.1 Open Policy Agent (OPA)

OPA is an open-source, general-purpose policy engine that helps us to implement policies as code using high-level declarative language Rego that resembles JavaScript (and is inspired by old query language Datalog).

OPA decouples policy decision-making from policy enforcement. When your software needs to make policy decisions it queries OPA and supplies structured data (e.g., JSON) as input. OPA generates policy decisions by evaluating the query input and against policies and data. OPA and Rego are domain-agnostic so you can describe almost any kind of invariant in your policies[25]*.*

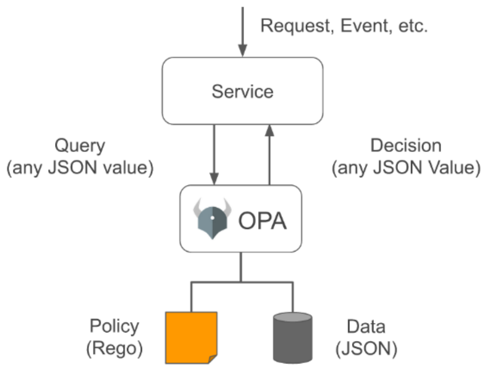


Figure 6.1 OPA overview [image from OPA documentation[26]]

OPA REST API exposes endpoints for managing (adding, removing, modifying) policy modules.

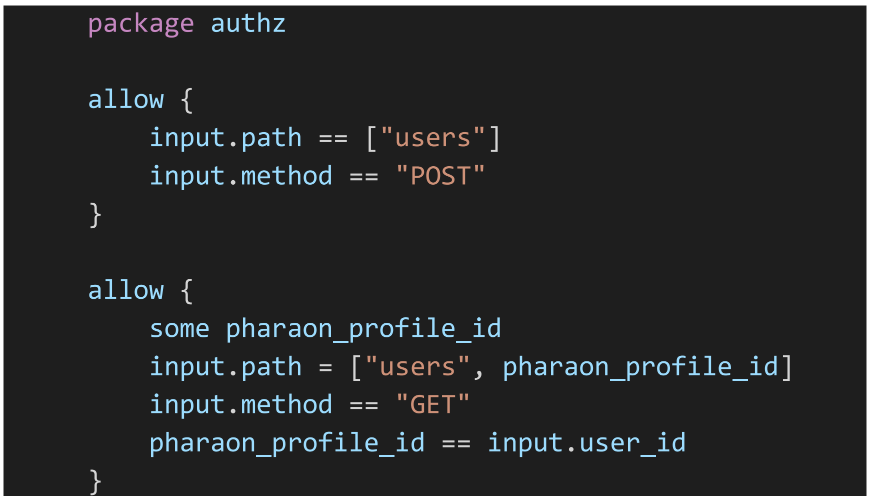
Some of the organizations that adoped OPA are Netflix (to control access to its internal APIs), Chef (for IAM capabilities), Pinterest, Goldman Sacks etc.

6.2 OPA and Pharaon

Since OPA exposes domain-agnostic APIs, Pharaon services cloud them to manage and enforce different policies. OPA will be further analyzed to see if and where it could be best applied within Pharaon. OPA policy example for Pharaon is shown in the following figure (Figure 6.2).

[25] https://www.openpolicyagent.org/docs/latest/

[26] https://www.openpolicyagent.org/docs/latest/



```
package authz

allow {
    input.path == ["users"]
    input.method == "POST"
}

allow {
    some pharaon_profile_id
    input.path = ["users", pharaon_profile_id]
    input.method == "GET"
    pharaon_profile_id == input.user_id
}
```

Figure 6.2 OPA policy example for Pharaon

6.2.1 Benefits of using OPA in Pharaon

Both security and compliance are high on the list of Pharaon priorities. Placing access controls throughout Pharaon infrastructure, systems and services comes at a cost of overhead which can degrade the performance and ultimately impact the customer (end user) experience. Common solutions in enterprises are LDAP/AD (Lightweight Directory Access Protocol/Active Directory) used for the authentication and authorization of every request across organization. Such setup only provides data but not the decision-making using that data meaning that every service or application still has to code the decision making logic.

OPA allows "security policy as a code" validations that evaluate the data in the context of different organization's security and compliance policies, which means that different Pharaon pilots residing in different countries can have their own policies. The benefit of externalizing such concerns from the application allows managing the security in more generic way.

OPA can be used with k8s, access control across Pharaon services, policy driven CI/CD pipelines.

6.2.2 JWT tokens and OPA

OpenAPIs in Pharaon (described in 0) are secured with JSON Web Tokens (JWTs) (as described in Pharaon deliverable D3.6) and Pharaon JWTs could be given to OPA and use OPA specialized support for JWTs to extract the information needed to make a policy decision as shown in the following figure (Figure 6.3).

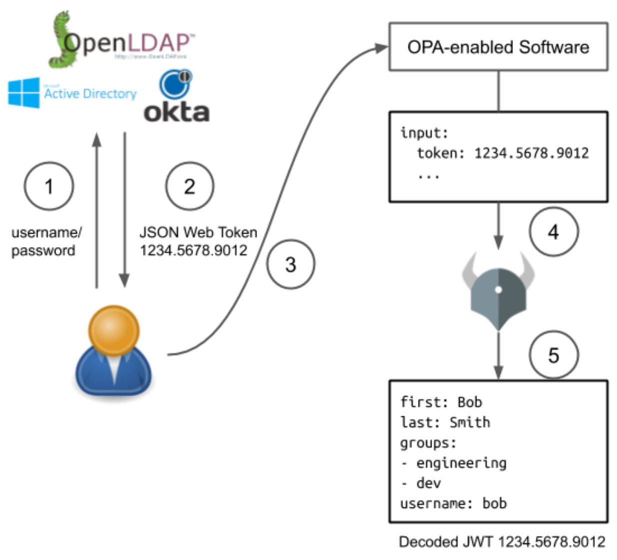


Figure 6.3 JSON Web Token flow [image from OPA documentation[27]]

6.2.3 Using OPA with Docker

Finer access control using OPA can be done via Docker plugin infrastructure[28] (using opa-docker-authz plugin[29]). Since Docker is used in Pharaon OPA can help here as well.

6.2.4 Evaluation of policies via REST API

For evaluation of policies there are different ways, one being REST APIs that return JSON over HTTP.

6.2.5 Collecting status reports

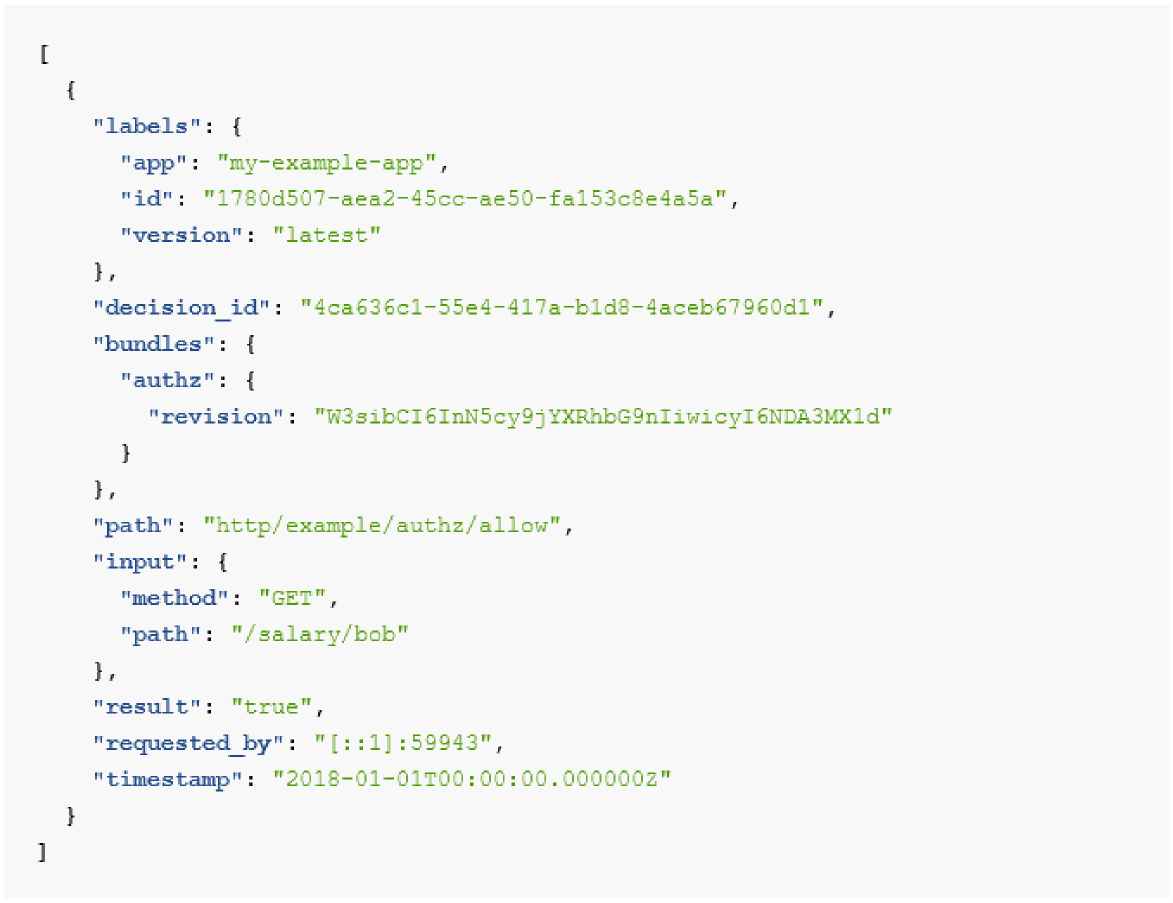
OPA can periodically report status updates to remote HTTP servers[30].

6.2.6 Collecting log of policy decisions

OPA can periodically report decision logs to remote HTTP servers via gzip compressed JSON array as shown in following figure (Figure 6.4) where each array element (event) represents a policy decision.

Sensitive data (such as passwords) can be masked by OPA masking policy.

[27] https://www.openpolicyagent.org/docs/latest/external-data/
[28] https://www.openpolicyagent.org/docs/v0.11.0/docker-authorization/
[29] https://github.com/open-policy-agent/opa-docker-authz
[30] https://www.openpolicyagent.org/docs/latest/management/#status



```json
[
  {
    "labels": {
      "app": "my-example-app",
      "id": "1780d507-aea2-45cc-ae50-fa153c8e4a5a",
      "version": "latest"
    },
    "decision_id": "4ca636c1-55e4-417a-b1d8-4aceb67960d1",
    "bundles": {
      "authz": {
        "revision": "W3sibCI6InN5cy9jYXRhbG9nIiwicyI6NDA3MX1d"
      }
    },
    "path": "http/example/authz/allow",
    "input": {
      "method": "GET",
      "path": "/salary/bob"
    },
    "result": "true",
    "requested_by": "[::1]:59943",
    "timestamp": "2018-01-01T00:00:00.000000Z"
  }
]
```

Figure 6.4 OPA decision log [image from OPA documentation[31]]

6.2.7 Health API

OPA Health API endpoint can be executed to verify that the OPA Pharaon server is operational.

6.2.8 Prometheus API

The Prometheus (usage in Pharaon explained in 5.1) endpoint is enabled by default OPA is run as a server. OPA exposes an HTTP endpoint that can be used to collect performance metrics for all API calls. To enable metric collection from OPA the `prometheus.yml` config has to be set:

```yaml
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: "opa"
    metrics_path: "/metrics"
    static_configs:
    - targets:
      - "localhost:8181"
```

[31] https://www.openpolicyagent.org/docs/latest/external-data/

6.2.9 Editor and IDE support

OPA can be integrated into different IDEs and editors[32], including The VSCode (https://code.visualstudio.com/) selected to be used in Pharaon, where it gives a plugin to develop, test, debug and analyze policies for the Open Policy Agent (available at https://marketplace.visualstudio.com/items?itemName=tsandall.opa).

Rego playground is available at the following web link: https://play.openpolicyagent.org/

7 Conclusions

This deliverable reports on the initial considerations and work done concerning service orchestration in Pharaon. As more and more information is being gathered from technology partners providing the pilots' technologies, more knowledge is gained that will be used for the planning of activities. Special consideration related to the Pharaon ecosystem, concrete architectures of the Pharaon six pilots, deployment architectures, ownership of the cloud environments and access to the infrastructure are also being analyzed as the project evolves. All input information is continuously being analyzed in order to understand the needs better and ultimately where the most added value can be delivered. In such analysis, not only the technology but also business aspects, including restricted access to some resources and infrastructure from technology providers (Pharaon partners) is being taken into account. With all this in mind, and considering the limited resources available, the contributions, tools and approaches with the highest potential are and will be prioritized.

[32] https://www.openpolicyagent.org/docs/latest/editor-and-ide-support/