PHYSICS

oPTIMIZED HYBRID SPACE-TIME SERVICE CONTINUUM IN FAAS

D6.5 – PHYSICS APPLICATION PROTOTYPE V1

Lead Beneficiary	ISPRINT
Work Package Ref.	WP6 – UC Adaptation, Experimentation, Evaluation
Task Ref.	T6.3 – Use Cases Adaptation & Experimentation
Deliverable Title	D6.5 – PHYSICS Application Prototype V1
Due Date	2022-05-31
Delivered Date	2022-05-31
Revision Number	3.0
Dissemination Level	Public (PU)
Type	Demonstrator (DEM)
Document Status	Release
Review Status	Internally Reviewed and Quality Assurance Reviewed
Document Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	Mr. Stefano Foglietta

H2020 ICT 40 2020 Research and Innovation Action

This project has received funding from the European Union's horizon 2020 research and innovation programme under grant agreement no 101017047

CONTRIBUTING PARTNERS

Partner Acronym	Role[1]	Name Surname[2]
iSPRINT	Lead Beneficiary	Harm op den Akker, Aristodemos Pnevmatikakis, Stathis Kanavos, George Labropoulos, Dimitris Bekiaris, Panos Babalis
FTDS		André Hennecke, Niklas Franke
CYBE		Théophile Lohier
DFKI		Volkan Gezer, Carsten Harms
UPM		
INNOV		
GFT		

REVISION HISTORY

Version	Date	Partner(s)	Description
0.1	2021-09-07	iSPRINT	Initial setup of document and initial ToC suggestions
0.2	2022-04-12	iSPRINT	Updated ToC
0.5	2022-05-02	iSPRINT	Version for internal progress update
1.0	2022-05-04	Isprint DFKI CYBE	Integrated contribution for Smart Manufacturing Use Case (DFKI), initial contribution for Smart Agriculture Use Case (CYBE)
1.1	2022-05-13	iSPRINT	Alignment of design- and prototype sections between 3 use cases.
1.2	2022-05-16	iSPRINT, DFKI, CYBE	Updated Smart Manufacturing sections, added overview of PHYSICS architecture, executive summary and conclusions, update Smart Agriculture sections.
2.0	2022-05-20	UPM DFKI CYBE	Internal review of document
3.0	2022-05-27	INNOV GFT	QA of the document.

[1] Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance
[2] Can be left void

LIST OF ABBREVIATIONS

Term	Explanation
BPMN	Business Process Model and Notation
CLA	Command Line Argument
CSP	Cloud Service Provider
CYBE	CybeleTech
D	Deliverable
DEM	Demonstrator
DFKI	Deutsches Forschungszentrum für Künstliche Intelligenz [German Research Centre for Artificial Intelligence]
ETL	Extract, Transform, Load
FaaS	Function as a Service
GPU	Graphics Processing Unit
HTTP	HyperText Transfer Protocol
iSPRINT	Innovation Sprint
JSON	JavaScript Object Notation
NRT	Near-Real-Time
OW	OpenWhisk
QC	Quality Control
RAMP	Reusable Artefacts Marketplace Platform
RIA	Research and Innovation Action
SaaS	Software as a Service
T	Task
WP	Work Package

EXECUTIVE SUMMARY

The goal of this deliverable, which is of type *Demonstrator*, is to accompany the three demonstrators built for the three pilots: Smart Manufacturing, eHealth, and Smart Agriculture and document the internal prototype milestone of the project.

These use-cases cover three very distinct application scenarios (i.e., manufacturing, health, and agriculture) that cover three major areas of European everyday life and economic activity. We thus demonstrate the benefits of the PHYSICS platform in a broad range of application scenarios and show how to improve agility and adaption by applying more advanced computing models and cover a wide and diverse range of available edge resources (e.g., small IoT sensors, mobile devices to powerful Edge nodes).

The current document D6.5: PHYSICS Application Prototype V1 is the first of a series of 2 deliverables that mark the "Version 1" and "Version 2" releases of these prototypes, and are being produced in the context of Task T6.3: Use Cases Adaptation and Experimentation.

In this document you will find, the following elements:

- A brief overview of the PHYSICS Project and the PHYSICS Platform – in order to provide the necessary context for the remainder of the document we describe the high level aims of the PHYSICS project, as well as the high level PHYSICS technical architecture, consisting of Infrastructure Layer, Continuum Deployment Layer, and Application Developer Layer – the last of which allows the development of the three prototypes as described in this document;
- An overview and short description of design decisions that were made in the process of producing the demonstrators for the three use cases. For all three use cases we provide a short summary of their aims and objectives, and highlight the major Functional Requirements to be fulfilled either in this version, or the next prototype.
- Descriptions of the three demonstrators produced, covering the use cases of Smart Manufacturing, eHealth, and Smart Agriculture. For each use case, it is described how Node-RED flows are designed and use to design the application flow, and how the use cases are deployed in the PHYSICS Platform. For each use case, the fulfillment of functional requirements is discussed, as well as a roadmap looking forward to the next phase(s) of development.

The next version of this deliverable is D6.6: PHYSICS Application Prototype V2, which is due to be released in M34 of the project (October 2023).

CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION

This document D6.5: PHYSICS Application Prototype V1 is the accompaniment to the first version of three separate demonstrators that are delivered under this Task 6.3 Use Cases Adaptation & Experimentation. As the name of the task implies, the activities in this slice of the PHYSICS project deal with adapting the Use Cases (*Smart Manufacturing*, *eHealth*, and *Smart Agriculture*) to the PHYSICS Platform-to-be in an experimental fashion. The three use cases mentioned have in some detail been defined in T6.2: Use Case Scenarios, as described in D6.3: PHYSICS Application Scenarios Definition V1 [1], which will later be refined in a follow-up "V2" document. In order for the current document to be understandable in isolation, we provide a short summary of each of the use cases in the relevant subsection of Section 3, but we refer to the details regarding requirements for each of the Use Cases to the D6.3 document.

Although requirements have been captured in a very systematic way, this Use Case "Adaptation & Experimentation" is indeed a more experimental process. In particular due to the dynamic nature of the PHYSICS platform in the early stages of the project's development in which new features and functionalities are implemented continuously. Nevertheless, and again in order to maintain a document that is readable in and of itself, we aim to provide a quick overview of the PHYSICS platform and architecture in Section 2 of this document.

The core contents of this project deliverable describe the current version of the application demonstrators as developed by the three use case partners. This is described in Section 4 of this document.

1.1 Objectives of the Deliverable

The objective of this deliverable is to demonstrate three use case prototypes, in the Smart Manufacturing, eHealth, and Smart Agriculture domains respectively, and how they use the PHYSICS Platform and platform components. This document accompanies those prototypes and aims to describe the prototypes, and how they use the PHYSICS Platform components, as well as provide additional meaningful context information.

1.2 Insights from other Tasks and Deliverables

The image below shows the timeline of activities for Task 6.3 under which this document is delivered. As shown, this document builds on the previously delivered documents D2.4: *PHYSICS Reference Architecture Specification V1* [2] – describing the initial PHYSICS reference architecture and D6.1: *Prototype of the Integrated PHYSICS solution framework and RAMP V1* [3] – describing a recent state of the integrated PHYSICS platform, as well as D6.3: Application scenarios definition V1 [1] - describing the definitions and requirements for the use cases.

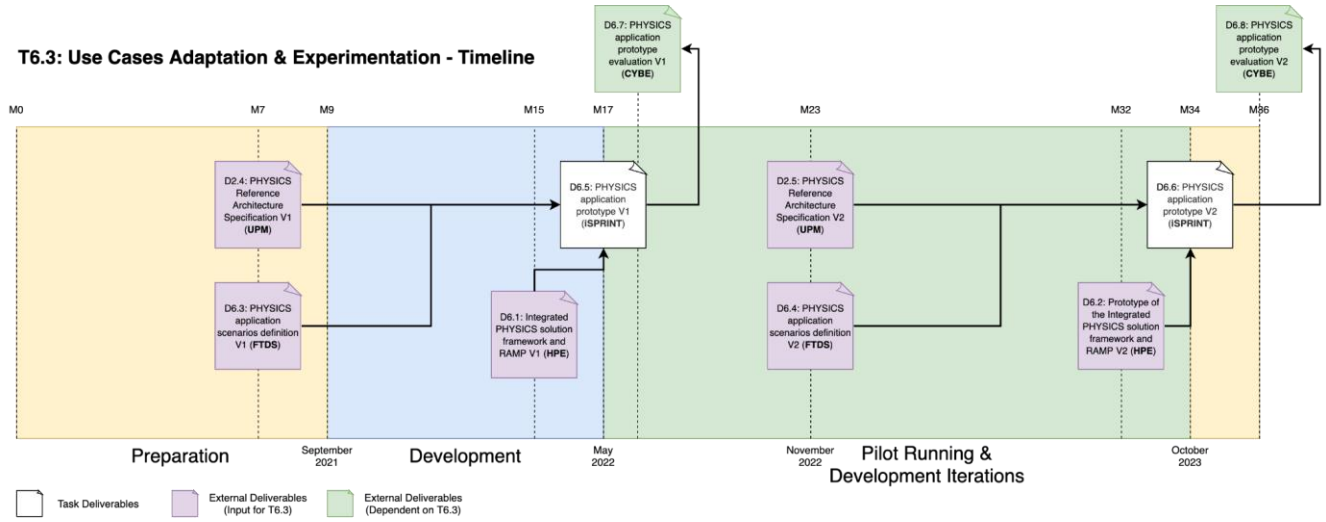

Figure 1: Timeline for T6.3 including dependencies between the various relevant deliverables.

1.3 Structure

The remainder of this document is structured as follows. Section 2 gives a brief overview of the PHYSICS architecture. This overview is not meant to be exhaustive, but merely provides some context to better understand the contents of the following sections. Section 3 provides a short overview as well as any potentially relevant design specifications of the three different use cases: Smart Manufacturing, eHealth and Smart Agriculture. In Section 4, the actual prototype demonstrators for the use cases are described, and finally conclusions and an outlook for future work is provided in Section 5.

2 OVERVIEW OF PHYSICS ARCHITECTURE

The PHYSICS platform architecture was first described in D2.4: PHYSICS Reference Architecture Specification V1 [2] and is still being developed and finetuned until the final architecture deliverable will be released in M23 of the project (November 2022), and possibly beyond that. Below we provide a short summary of the overall objectives for the PHYSICS project and platform (§2.1), as well as the current state of the platform architecture (§2.2), in order to help better understand the prototypes as described below in Section 4.

2.1 The PHYSICS Project

PHYSICS empowers European Cloud Service Providers (CSPs) to exploit the most modern, scalable and cost-effective cloud model (FaaS), operated across multiple service and hardware types, provider locations, edge, and multi-cloud resources. To this end, it applies a unified continuum approach, including functional and operational management across sites and service stacks, performance through the relativity of space (location of execution) and time (of execution), enhanced by semantics of application components and services. PHYSICS applies this scope via a vertical solution consisting of:

- A *Cloud Design Environment*, enabling design of visual workflows for applications, exploiting provided generalized Cloud design patterns functionalities with existing application components, easily integrated and used with FaaS platforms, including incorporation of application-level control logic and adaptation to the FaaS model;
- An *Optimized Platform Level FaaS Service*, enabling CSPs to acquire a cross-site FaaS platform middleware including multiconstraint deployment optimization, runtime orchestration and reconfiguration capabilities, optimizing FaaS application placement and execution as well as state handling within functions, while cooperating with provider-local policies;
- A *Backend Optimization Toolkit*, enabling CSPs to enhance their baseline resources performance, tackling issues such as cold-start problems, multi-tenant interference and data locality through automated and multi-purpose techniques.

Furthermore, PHYSICS will produce an Artefacts Marketplace (RAMP) (see [3]), in which internal and external entities (developers, researchers, etc.) will be able to contribute fine-grained reusable artifacts (such as functions, flows, or controllers). PHYSICS will validate the outcomes in 3 real-world applications (eHealth, Agriculture and Manufacturing), making a business, societal and environmental impact on the lives of EU citizens.

2.2 Basic Architecture

The following summary of the PHYSICS Architecture has been adopted from [3].

The current version (M17 – May 2022) of the prototype implementation of the integrated PHYSICS solution framework can be referred to as the PHYSICS Minimum Viable Platform (MVP). The main components of the PHYSICS architecture, implemented in the PHYSICS MVP, are shown in Figure 2 below.

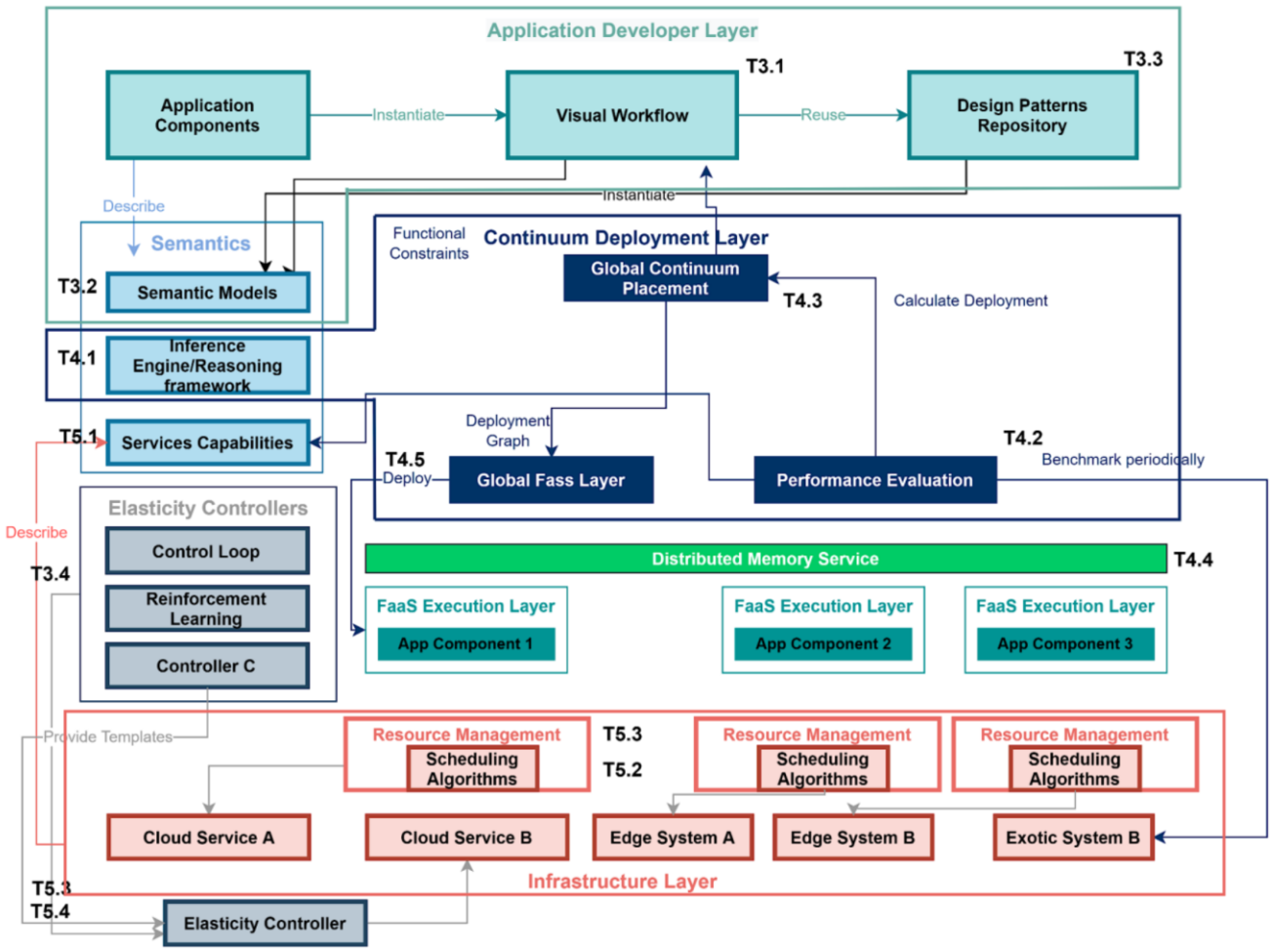

Figure 2: The PHYSICS "Minimum Viable Platform" Prototype Architecture.

Figure 2 above presents three layers from top to bottom: (1) the *Application Developer Layer*, (2) the *Continuum Deployment Layer* and (3) the *Infrastructure Layer*, which correspond to the developments in the three technical work packages of the PHYSICS project:

- WP3: Functional and Semantic Continuum Services Design Framework (Application Layer)
- WP4: Cloud Platform Services for Global Space-Time Continuum Interplay (Continuum Deployment Layer)
- WP5: Extended Infrastructure Services with Adaptable Algorithms (Infrastructure Layer)

The top layer, Application Developer Layer, is the entry point for users that design their applications using a Visual Workflow tool. The design of applications is eased by reusing common design patterns such as split-join for function parallelisation, batch processing, data collection, and more, provided by the Design Patterns Repository. Application components (e.g., functions) can be semantically annotated providing information to lower layers that may affect the placement, deployment, operation and configuration of the application (Semantic Models). Application components may have elasticity controllers that regulate the algorithms and resources needed for scaling a component.

The Continuum Deployment Layer oversees providing uniform access to the diverse cloud services provided by one or more cloud providers. The Global Continuum Placement is in charge of deciding on the most suitable deployment of applications taking into account the performance of the services, costs, affinity

constraints of components: for that purpose, it receives the list of candidate services that the Reasoning Framework has filtered taking into account the application graph needs and the performance of the services provided by the cloud services and edge devices Performance Evaluation component. The placement of the components is done by the Global FaaS Layer component. The Global FaaS layer abstracts the usage of different data centers from one or more cloud providers. The management of data shared by functions of applications is provided at this level by the Distributed Memory Service.

The Infrastructure Layer provides a view and interface for enabling an optimized operation of the edge and cloud services utilized for the realization of the application service graph. To this end, the Service Capabilities component depicts and models the abilities of each service and resource type. The analysis of different algorithmic approaches for adaptive and real-time provider level scheduling (Scheduling algorithms) so that resources are adapted to current application needs while maintaining overall QoS levels is done by the Resource Management component. The co-allocation strategies component provides, on behalf of the provider strategies, optimizations to maximise performance.

3 USE CASES: OVERVIEW AND DESIGN

As explained in the introduction section of this deliverable document, the main contents of the document are split into two parts. This section provides short overviews of the three different use cases and gives room to document any design decisions or functional specifications that were used to create the actual prototypes that are described in Section §4.

In the three sub-sections below, we provide for each of the three use cases, a short summary or synopsis to refresh the reader on the context of the use case, as well as any design or specifications that may be of interest or required to understand certain development choices that were made in the development of the prototypes. Note that each of the use cases define their own design and functional specifications, based on the use-case specific requirements that were defined earlier in D6.3 [1].

Due to the experimental nature of this task (T6.3), additional specifications, designs, and perhaps even requirements – as well as updates to the ones presented here – can be expected in the next iteration(s) of this task, to be reported in the follow-up document D6.6 to this deliverable D6.5.

The three different use cases are included in the following sections:

- 3.1: Use Case “Smart Manufacturing”
- 3.2: Use Case “eHealth”
- 3.3: Use Case “Smart Agriculture”

3.1 Use Case: “Smart Manufacturing”

3.1.1 Synopsis

SmartFactory-KL provides an Industry 4.0-compliant and manufacturer-independent demonstrator for the PHYSICS-Project. The integration of SmartFactory-KL with the PHYSICS platform enables the decoupling of the available services in the production line. The initial version of the pilot plant already follows a service-oriented approach, which made it easier to convert it into a Function-as-a-Service (FaaS) system. Within the Smart Manufacturing use case, two scenarios were defined. One of the scenarios implement a failover case. In case of the local Quality Control (QC) service failure, the system is expected to forward the QC request to the PHYSICS-Platform and continue the QC without downtime. The second use case develops a more complex QC service following FaaS approach, which is expected to increase the certainty level[3], in case the local QC service fails to provide an adequate value. Initially, the local QC service utilizing PHYSICS at the edge with low computing resources will be used for a faster analysis. If the certainty level is not satisfactory, then, the system will forward the QC data to the complex QC service and perform computations with more available resources.

An important factor in both use cases is the priority of the Local and Cloud version of the services. In both, the local services have precedence. If the local QC service do not function properly or not at all, then the QC service in the Cloud will be used. The PHYSICS-Platform (which will be used at the edge and Cloud) and FaaS approach enable higher availability which was not available at the initial version of the pilot plant.

3.1.2 Design & Specifications

As stated above, for the Smart Manufacturing pilot, two scenarios were defined. As the first prototype, the second scenario was chosen since it had minimum dependency with the development progress of the PHYSICS components.

[3] Or “score” of AI-based results.

The “to-be” BPMN diagrams for the scenario have been defined in Deliverable 6.3 [1]. For easier trackability, the diagram of the second scenario is also given in Figure 3 below. To realize the use cases, the functional requirements shown in the Table 1 are defined.

Table 1: Functional Requirements of the DFKI Use Case.

Code	Functional Requirements
FRS-UC1-01	An inference service for quality control must be provided to be invoked by DFKI.
FRS-UC1-02	The inference service is to be deployed according to the principles of the PHYSICS project through the OpenWhisk platform.
FRS-UC1-03	The inference service is to be utilized via the endpoints exposed by Node-RED flows.
FRS-UC1-04	The inference will take place in a custom docker-based OpenWhisk action.
FRS-UC1-05	The inference service is to be invoked by specifying the inputs (image and other quality related data) to infer upon and the model to be used.
FRS-UC1-06	The inference service tolerates the server failures by utilizing the PHYSICS platform (Edge & Cloud).
FRS-UC1-07	The inference service runs preferably on the local Edge.

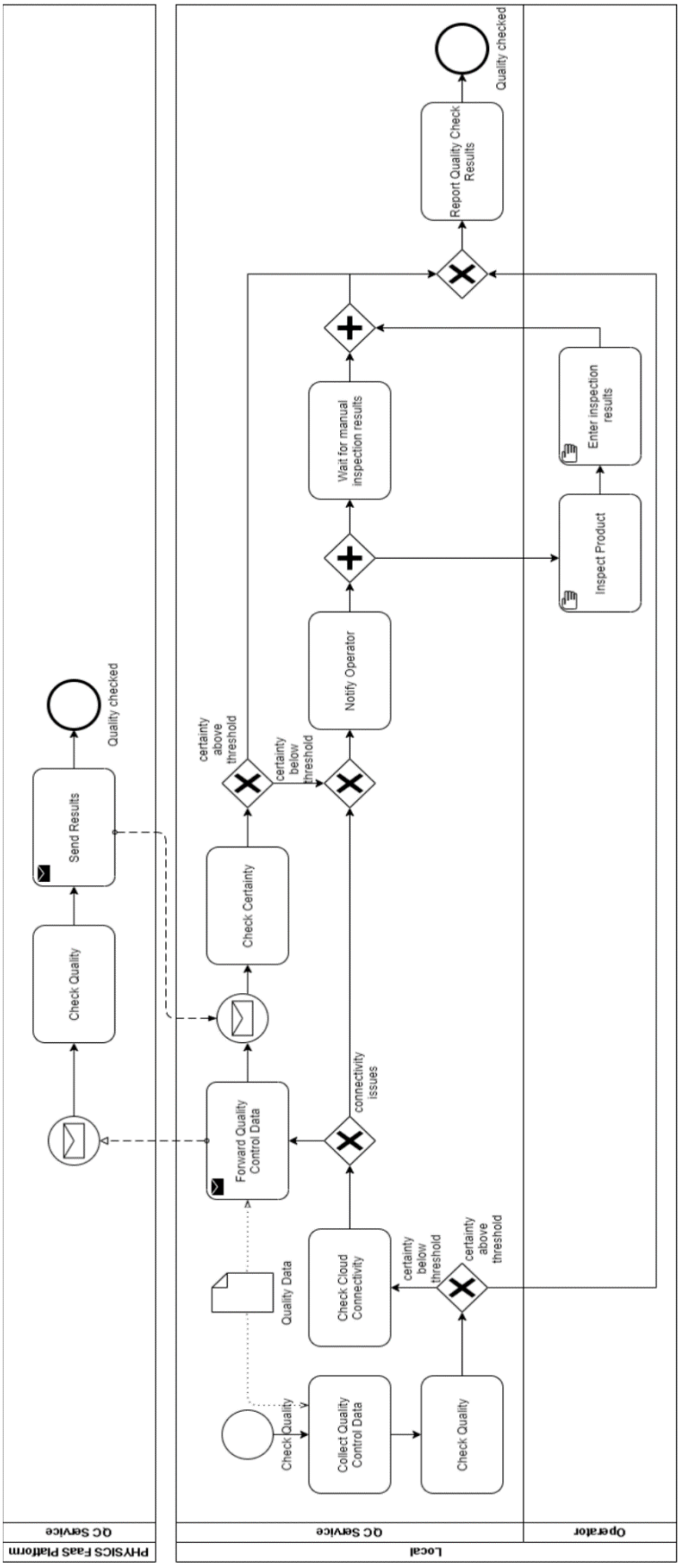

Figure 3: BPMN Diagram of Scenario #2 of the Smart Manufacturing Pilot.

3.2 Use Case: “eHealth”

3.2.1 Synopsis

The aim of the “eHealth” Pilot is to improve the performance and maintainability of the Healthentia platform, developed by Innovation Sprint (iSPRINT), by using Function-as-a-Service (FaaS) technologies as provided by the PHYSICS Platform for some of the smart services on offer. Healthentia is an eClinical Software-as-a-Service (SaaS) platform, consisting of a mobile app for patients/citizens, a web portal for healthcare professionals and researchers, and a server-platform for data storage and processing (see Figure 4 for an impression of the mobile app and its functionalities – as taken from [1]).

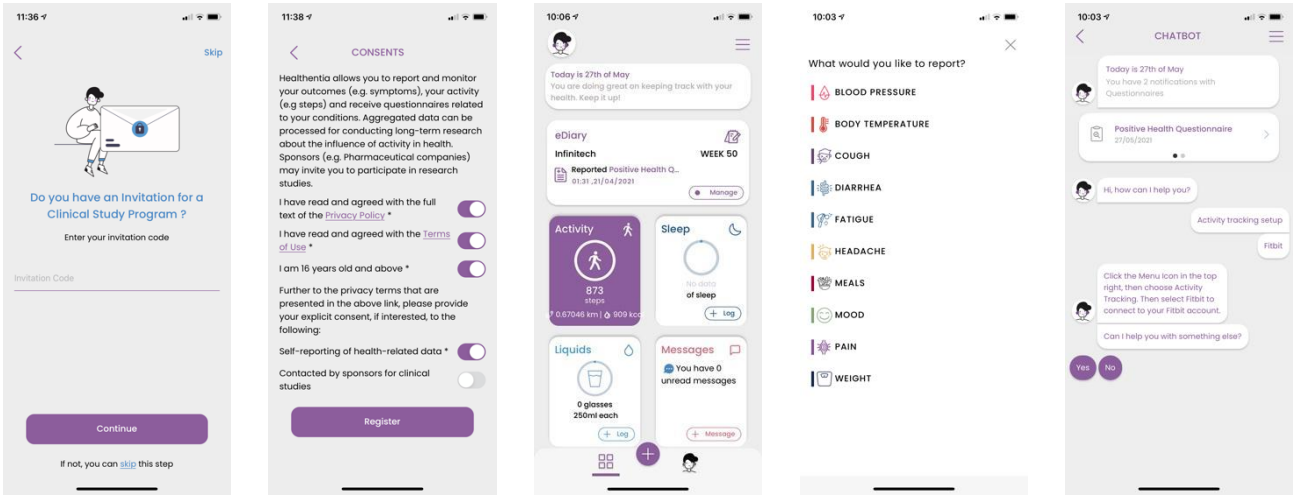

Figure 4: Screenshots of the Healthentia mobile application.

Deliverable 6.3 [1] provides all details and requirements for this Use Case. In order to better understand the provided functional specifications and prototype description (see §4) we include a summary of a typical usage scenario here:

An individual interested in monitoring or improving their health or lifestyle will download the Healthentia mobile application to their phone. The user provides an email address and password and finalizes their account creation by consenting to their data being used for research purposes. Once in the application, the user can link their Fitbit or Garmin account to Healthentia to start providing activity and sleep data. Additionally, they can report various symptoms and events and will be able to regularly answer questionnaires related to their overall health status. After sufficient data has been collected for a particular user, the Healthentia platform will start to offer its inference services. Depending on the specific study configuration, this inference can be e.g., a prediction of future health status. The predictions that are made in an online fashion serve as input to a virtual coaching component – an conversational agent that can discuss the prediction with the user in natural language dialogues.

3.2.2 Design & Specifications

The goal of the eHealth use case is thus to demonstrate the benefits of the PHYSICS FaaS approach on inference carried out for eHealth applications for Healthentia, the eClinical platform of Innovation Sprint. To achieve this, a number of use case specific functional requirements have been derived, as shown in

Table 2 below. These functional requirements are used as practical goals for the development of the first prototype described in Section 4.2 and as indications of its progress.

Table 2: Functional requirements for the eHealth use case.

Code	Functional Requirements
FRS-UC2-01	An inference service must be provided to be invoked by Healthentia.
FRS-UC2-02	The inference service is to be deployed according to the principles of the PHYSICS project through the OpenWhisk platform.
FRS-UC2-03	The inference service is to be utilized via the endpoints exposed by Node-RED flows
FRS-UC2-04	The inference flow will utilize an inference script written in Python
FRS-UC2-05	The inference script is to be invoked by specifying the vectors to infer upon and the model to be used

The fulfilment of these requirements in the current implementation of the eHealth use case is discussed in Section 4.2.

3.3 Use Case: "Smart Agriculture"

3.3.1 Synopsis

The smart agriculture pilot aims to provide growers enhancing greenhouse management scenarios. To achieve this goal, it is necessary to have: 1) A reliable tool to gather data collected in the greenhouse; 2) high performing simulation and optimization; 3) Up-to-date agronomic model parametrization obtained through calibration on empiric measurements. Figure 5 shows a global overview of the CybeleTech solution for greenhouses.

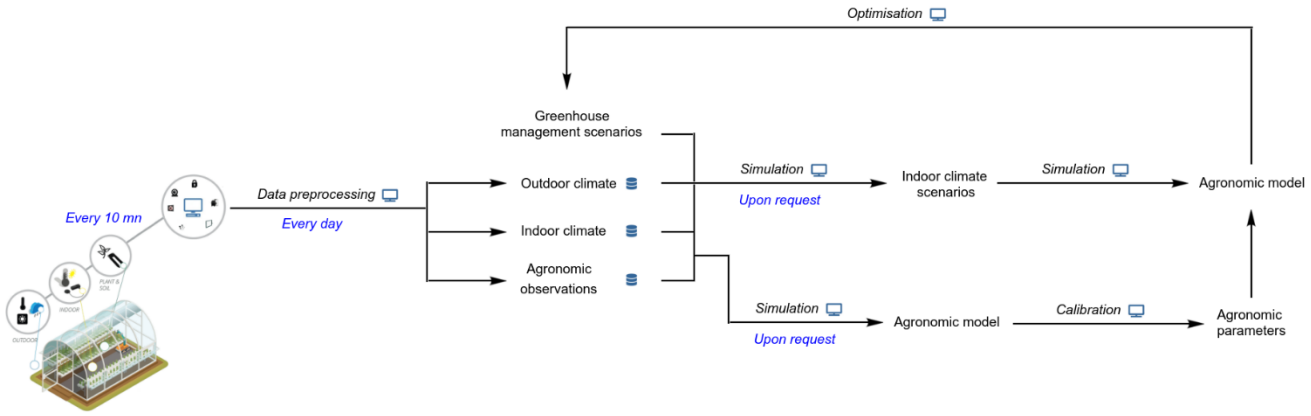

Figure 5: Overview of the Cybeletech solution for greenhouses.

Deliverable 6.3 [1] provides all details and requirements for this Use Case. In order to better understand the provided functional specifications and prototype description (see §4.3) we include a summary of a typical usage scenario here:

Growers interested in monitoring plant development in their greenhouses and in improving the management of the environmental conditions will contact Cybeletech. Cybeltech will audit the computer infrastructure, adapt the solution to the greenhouse constraints and deploy it. Once Cybeletech's solution is deployed in the greenhouse, the data collected by greenhouse sensors are automatically retrieved, preprocessed and stored in Cybeletech databases. The grower can visualize these data through the Cybeletech platform in near real time (NRT) to follow the environmental conditions in the greenhouse. Moreover, the system uses agronomic models developed by Cybeletech to give an overview of the plant status (e.g., healthy, heating of the leaves, water stress) in NRT. On the other hand, the grower can provide greenhouse management scenarios. Dedicated statistical models are then used to convert these scenarios in environmental conditions, which are in turn used to simulate plant development. Finally, the growers

can ask for optimal greenhouse management scenarios. In this case, several scenarios are automatically generated and the bests scenarios according to the tradeoff between plant development and environmental cost are returned.

3.3.2 Design & Specifications

The goal of the smart agriculture use case is thus to demonstrate: 1) how PHYSICS components can be deployed and used in the context of fog computing; and 2) the benefits of the PHYSICS FaaS approach on NRT simulation of plant development and optimization of greenhouse management. To achieve this, several use case specific functional requirements have been derived, as shown in Table 3 below.

As the first prototype, the emphasis was on adapting the pipeline for data collection.

Table 3: Functional requirements for the Smart Agriculture use case.

Code	Functional Requirements
FRS-UC3-01	A data collection pipeline that takes as input the Python script for data pre-processing must be provided.
FRS-UC3-02	The data collection pipeline is triggered automatically at regular time step and store data locally when connection is lost.
FRS-UC3-03	A minimal version of the docker image allowing to run the data collection pipeline is built during the deployment phase and can be pull from the edge.
FRS-UC3-04	A simulation / optimization flow must be provided to be invoked by Cybeletech greenhouse management suite.
FRS-UC3-05	The simulation / optimization service is to be deployed according to the principles of the PHYSICS project through the OpenWhisk platform.
FRS-UC3-07	The optimization service must take advantage of parallelization

The fulfilment of these requirements in the current implementation of the Smart Agriculture use case is discussed in Section 4.3.

4 PROTOTYPE DESCRIPTIONS

4.1 Use Case “Smart Manufacturing”

Based on the BPMN diagram presented in Section §3.1.2, a flow was designed with the PHYSICS design environment in Node-RED as shown in Figure 6. This flow contains two OpenWhisk (OW) Actions: 1) Quality Control, 2) Complex Quality Control. OW Action (1) the same quality control as in “as-is” scenario – packaged as a custom OW Action-compatible docker-container. The OW Action (2) performs, as the name suggests, a more complex quality control requiring GPU acceleration. Both OW Actions contain proprietary code and AI models developed prior to PHYSICS project and thus will not be disclosed. The actions just encapsulate those to be usable within PHYSICS platform following its requirements.

4.1.1 Quality Control PHYSICS Flow

The flow starts after it receives a base64-encoded image data in JSON-format through a POST request to “/qc”. An example of the request can be seen in Code Snippet 1 below:

```
{
  "mime": "image/jpeg",
  "encoding": "base64",
  "image": "base64-encoded input image",
  "expected": ["UsbPenDrive_2x4_Blue", "FlatStone_2x4_Black"]
}
```

Code Snippet 1: Example JSON input for the quality control inference service.

After each QC operation, the certainty of the results is checked. An example output is given in Code Snippet 2 below:

```
{
  "encoding": "base64",
  "image": "base64-encoded output image"
  "mime": "image/jpeg",
  "results": [
    {
      "class": "UsbPenDrive_2x4_Blue",
      "score": 0.9822760820388794,
      "x0": 4,
      "x1": 750,
      "y0": 686,
      "y1": 888
    },
    {
      "class": "FlatStone_2x4_Black",
      "score": 0.6782341003417969,
      "x0": 0,
      "x1": 776,
      "y0": 418,
      "y1": 696
    }
  ],
  "version": "1.0.0"
}
```

Code Snippet 2: Example JSON output for the quality control inference service.

The resulting array scores are inspected, and the minimum score is used for “certainty”. Based on the threshold, in the following manner:

1. After QC (1)
    a) If certainty is above the threshold, the quality result is checked,
    b) If certainty is below the threshold, the image data is forwarded to (2).
2. After QC (2)
    a) If certainty is above the threshold, the quality result is checked,
    b) If certainty is below the threshold, a manual inspection will be required.

Based on the conditions written above, three different status codes are generated. The computation results are summarized in Table 4 below.

Table 4: Summary of Status Codes of Use Case #2 in Smart Manufacturing Pilot.

	Quality OK	**Quality Not OK**
Certainty OK	AllOK	Certainty OK, Quality Not OK
Certainty Not OK	CertaintyNotOK	

If certainty is not OK, the operator will be notified by the caller of the Node-RED flow.

Figure 6 below shows a prototype of the second scenario of manufacturing use case depicted in BPMN diagram (See Figure 3). The sub flows were used for reusability and status reported for debugging. Note that the notification of the operator is handled outside of this flow.

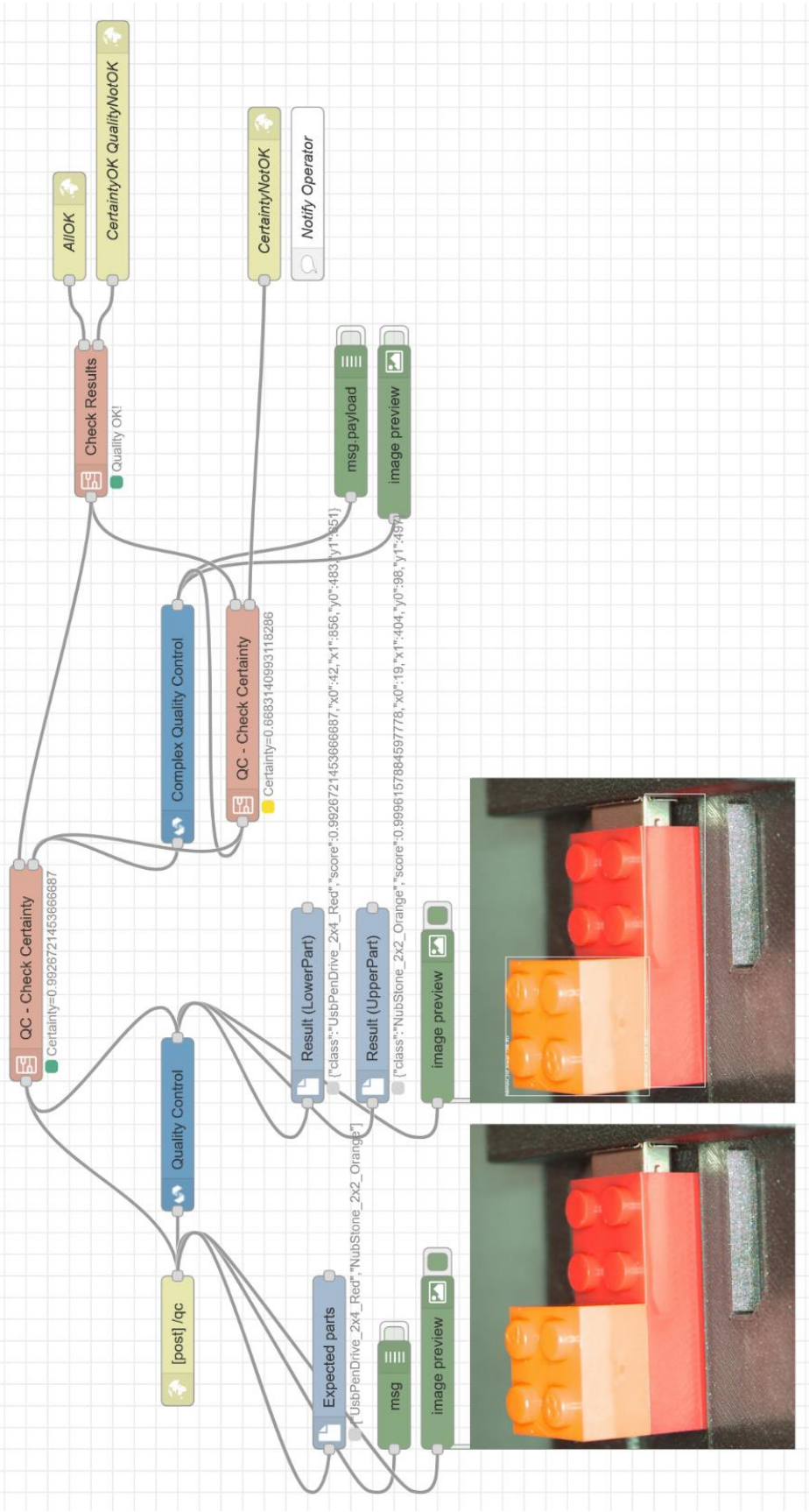

Figure 6: Smart Manufacturing Scenario #2, Prototype Node-RED Flow.

In Figure 7 below, the "Check Quality" sub flow is shown. It needs both the original message and the quality check result message, but Node-RED does not support nodes with multiple inputs, hence the *join* node is used for working around that limitation. Status is red if an error occurred, yellow if certainty is below the threshold and green otherwise.

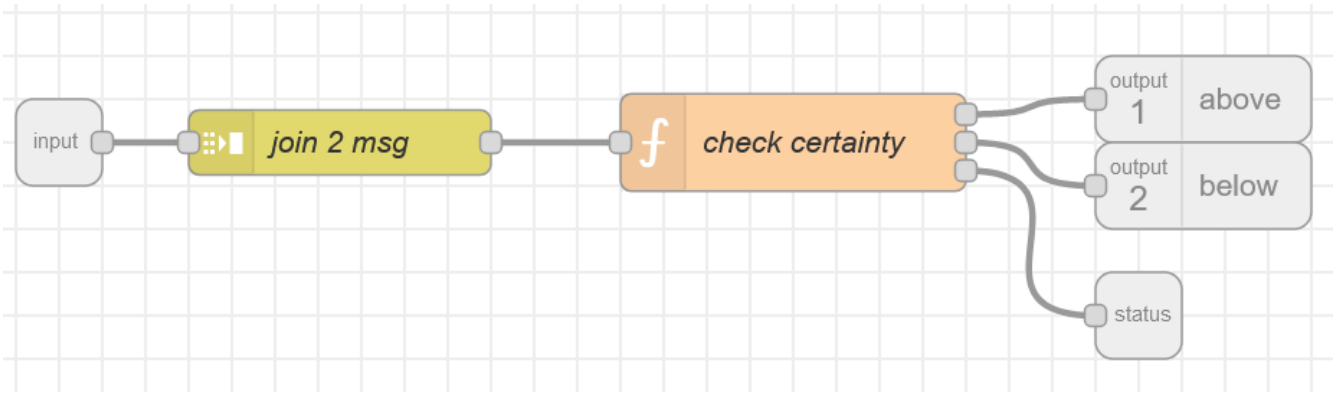

Figure 7: "Check Certainty" sub flow.

The "Check status" sub flow is illustrated in Figure 8 below. The status is red if an error occurred, yellow if quality is not OK and green if it is OK.

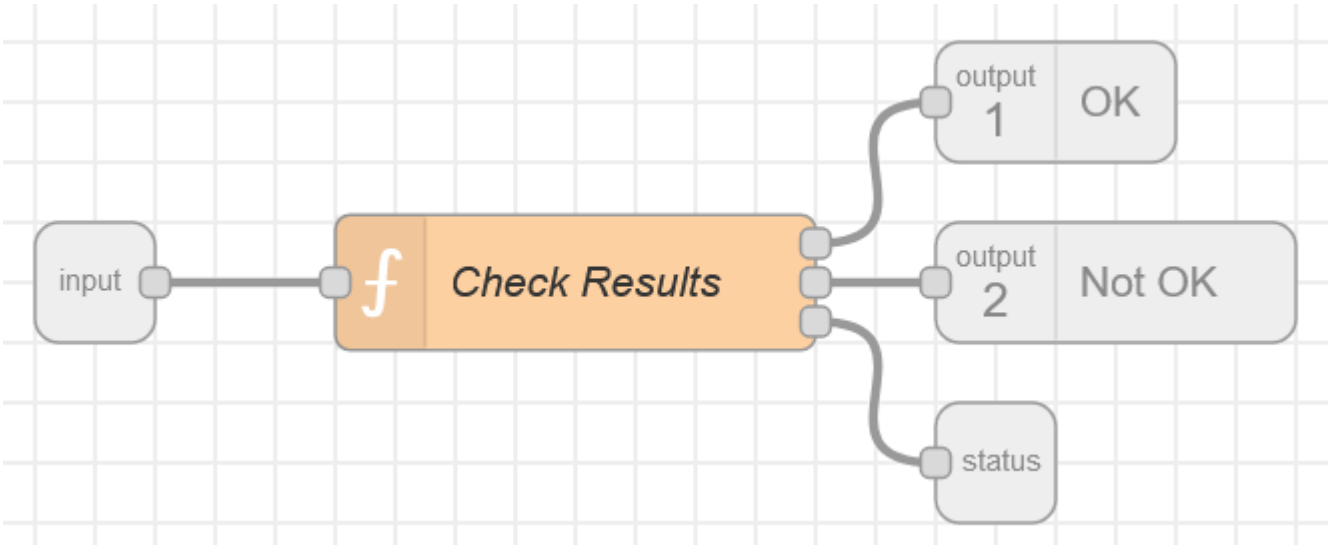

Figure 8: "Check Results" sub flow.

4.1.2 Concluding remarks for the first version quality control inference service

The source material for the prototype described here can be found on the internal PHYSICS repository:

https://repo.apps.ocphub.physics-faas.eu/PHYSICS/test/src/dfki

Access to this repository is limited to members of the PHYSICS Consortium, but may be granted upon request.

Based on the prototype and the functional requirements defined in §3.1, in this section, we summarize what is fulfilled and what is planned in the upcoming periods. The fulfilment statuses of the requirements are shown in the Table 5 below.

Table 5: Fulfilment of the functional specifications for the DFKI use case.

Code	Functional Specification Fulfilment
FRS-UC1-01	Completed. Endpoint is available at Kubernetes cluster with POST to "/qc".
FRS-UC1-02	Completed. Prototypes of Node-RED flows and quality control inference as OW-Action deployed.
FRS-UC1-03	Partially fulfilled. A small prototype was implemented to test the functionality and send image data to the endpoint.
FRS-UC1-04	Completed. The inference takes place in a custom docker-based OpenWhisk action.
FRS-UC1-05	On hold until the local Edge is connected to the Cloud to form the whole PHYSICS platform. Depends on PHYSICS functionality and components.
FRS-UC1-06	Not yet started. Prerequisite is to have FRS-UC1-05 completed.

In the upcoming prototype versions, it is planned to fully utilize the related components of the PHYSICS platform to exploit the FaaS benefits. The planning for these upcoming activities is provided in Table 6 below.

Table 6: Development Roadmap for the Smart Manufacturing use case.

	2022							2023			
Tasks	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**January**	**February**	**March**	**April**
Integration of first version of the PHYSICS patterns											
Collection of training data for complex AI quality control (QC)											
Implementation of the complex AI QC Service											
Conversion of complex AI QC Service into FaaS											
Connection of local PHYSICS Deployment to Cloud-based PHYSICS Platform											
Software adjustments regarding local-to-Cloud connection											
Integration of pattern(s) enabling secure communication between local and Cloud											
Test and validation											

4.2 Use Case "eHealth"

The eHealth use case is utilizing a local deployment of the PHYSICS design environment to prepare the necessary flows, and then invokes their version deployed on OpenWhisk to run inference on healthcare data, given pre-trained ML models. The use of the PHYSICS design environment to implement and build the inference flow and the use of Python for the actual inference script as discussed in Section 4.2.1. The experimentation that can currently be carried out is demonstrated in Section 4.2.2. This demonstration is followed in Section 4.2.3 by instructions on running and using the system, as well as by a description of how the necessary docker image is built. Finally, the fulfilment of the use case functional specifications and the next steps towards it are considered in the concluding Section 4.2.4.

4.2.1 The eHealth PHYSICS flow

Once the local implementation of the PHYSICS Design Environment is up and running, one can access the Admin Panel to see the flows exposed by Node-RED, as shown in Figure 9.

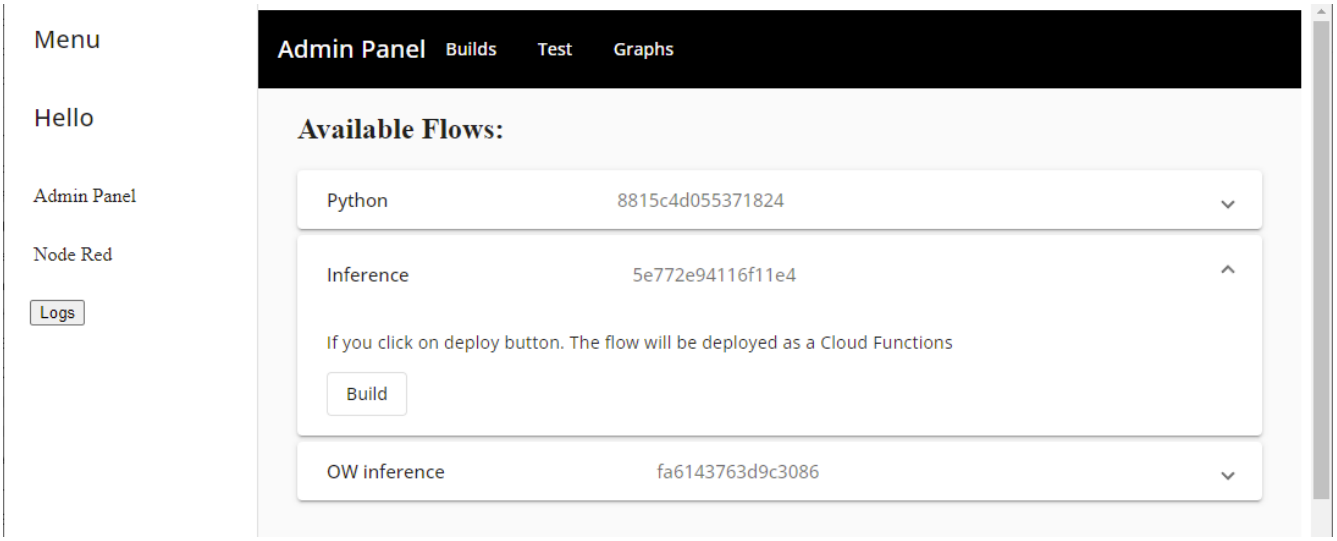

Figure 9: Admin Panel showing the flows exposed by Node-RED in the eHealth local workflow.

Selecting any of the flows allows the user to build the flow using Jenkins (see Figure 10) and then have the resulting image deployed at OpenWhisk, whereupon the flow is available for inference both from the Node-RED environment, but also via external invocation.

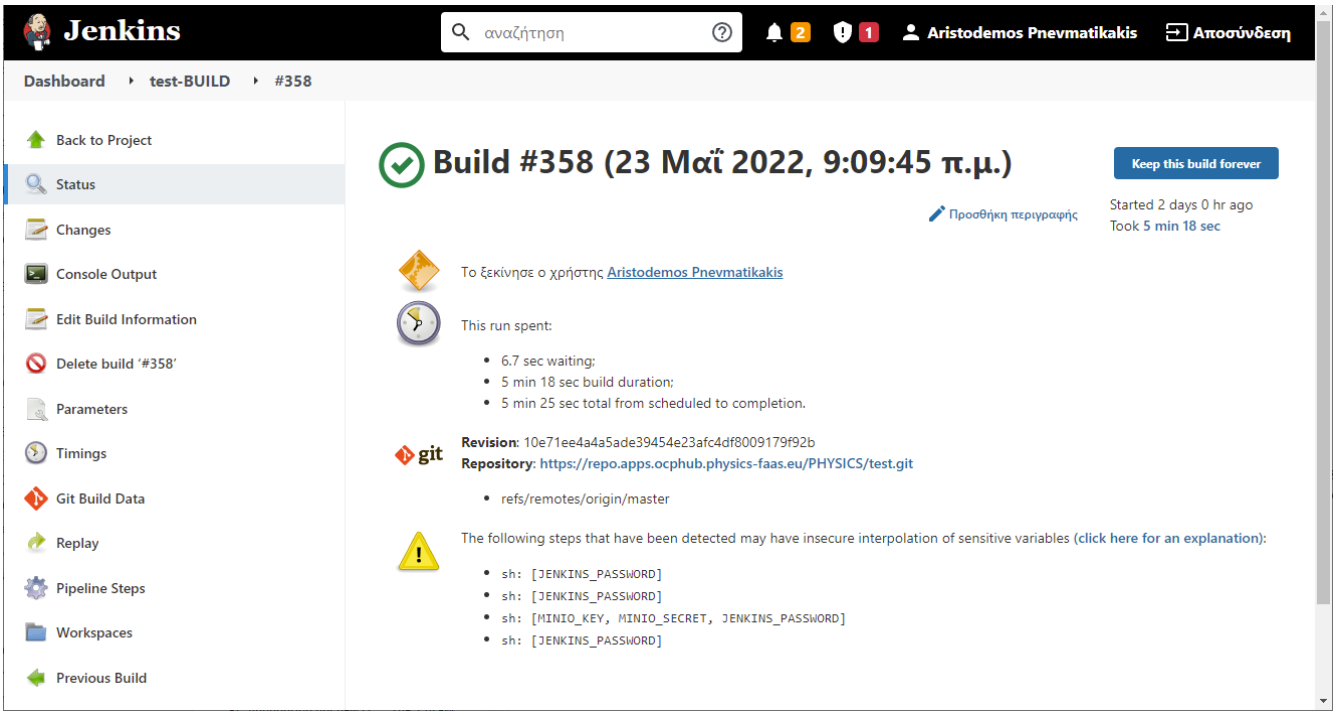

Figure 10: Jenkins build job initiated by the PHYSICS design environment (Admin panel) for the inference flow.

Selecting Node-RED from the menu, one accesses the Node-RED flow editor, where all the flows in the local implementation (in this case the flows of the eHelath use case) are loaded, together with the PHYSICS Node-RED components. This is shown in Figure 11.

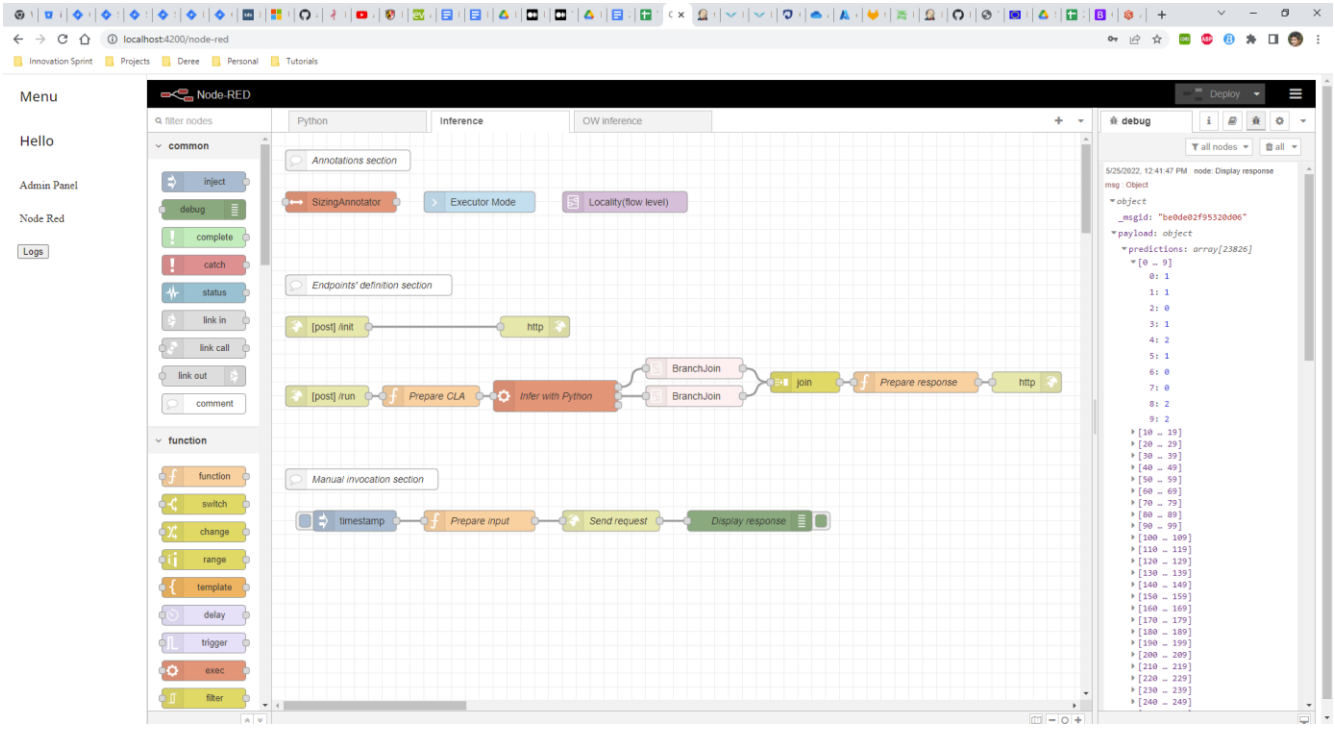

Figure 11: Node-RED interface depicting in inference flow in the eHealth local implementation.

The Python flow in the first tab contains experimental flows. The OW inference flow in the third tab is discussed in the experimentation section. Here the focus is on the Inference flow in the middle tab, shown in Figure 11.

The flow is divided in three sections. The annotations section on the top utilizes PHYSICS annotations to allow the user to add information on the behavior of the flow as deployed in OpenWhisk. The endpoints definition section defines the two endpoints exposed by the flow, adapted to the Openwhisk Action specification:

- POST init handles initialization and is currently a stub (empty top row sub-flow), and
- POST run performs the inference (middle row sub-flow).

At the heart of the run endpoint lies the "Infer with Python" execution node, invoking the Python inference script, passing it the command-line arguments prepared by the "Prepare CLA" (Command Line Argument) Javascript function node that manipulates the "message" object into the "cla" one. The "Prepare response" Javascript function node considers the standard and error output streams of the execution node, as they are concatenated together using the PHYSICS Branch-Join pattern. The configuration of all three nodes is shown in Figure 12.

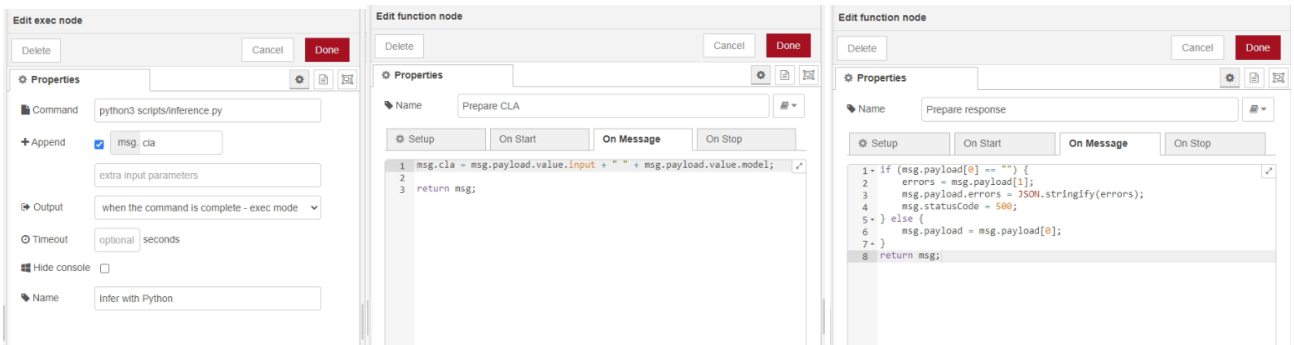

Figure 12: Properties of the "Infer with Python" execution node (left), the "Prepare CLA" function node (middle) and the "Prepare response" function node (right).

The manual invocation section of the inference flow, as well as the OW inference flow in the third tab are considered in the experimentation section.

4.2.2 Experimentation on the eHealth PHYSICS flow

The inference flow can be executed within the Node-RED environment by manually activating the timestamp node in the manual invocation section of the inference flow. This is quite useful for debugging the flow at the creation stage, without having to undergo the OpenWhisk deployment phase.

The current implementation of the inference expects two input parameters: the name of the joblib file containing the input vectors to be processed, and the name of the model to be used for inference in the `inference.py` Python script. This outputs the inferences, one per provided vector. A second Python script, `inference_test.py`, is provided that also tests the inference using the known inference results, providing the classification accuracy.

Manually invoking the POST run of the flow results to successful inference as indicated by the inference results at the logs shown in the right column of Figure 11.

In a future version of this flow, the input vectors might be directly provided in the POST message instead of a filename, depending on the needs of the experimentation on the flow.

Any flow deployed in OpenWhisk can be executed by POSTing at its init and run endpoints from outside the PHYSICS design environment, or within an OpenWhisk experimentation flow. This is the OW inference flow in the third tab, shown in Figure 13.

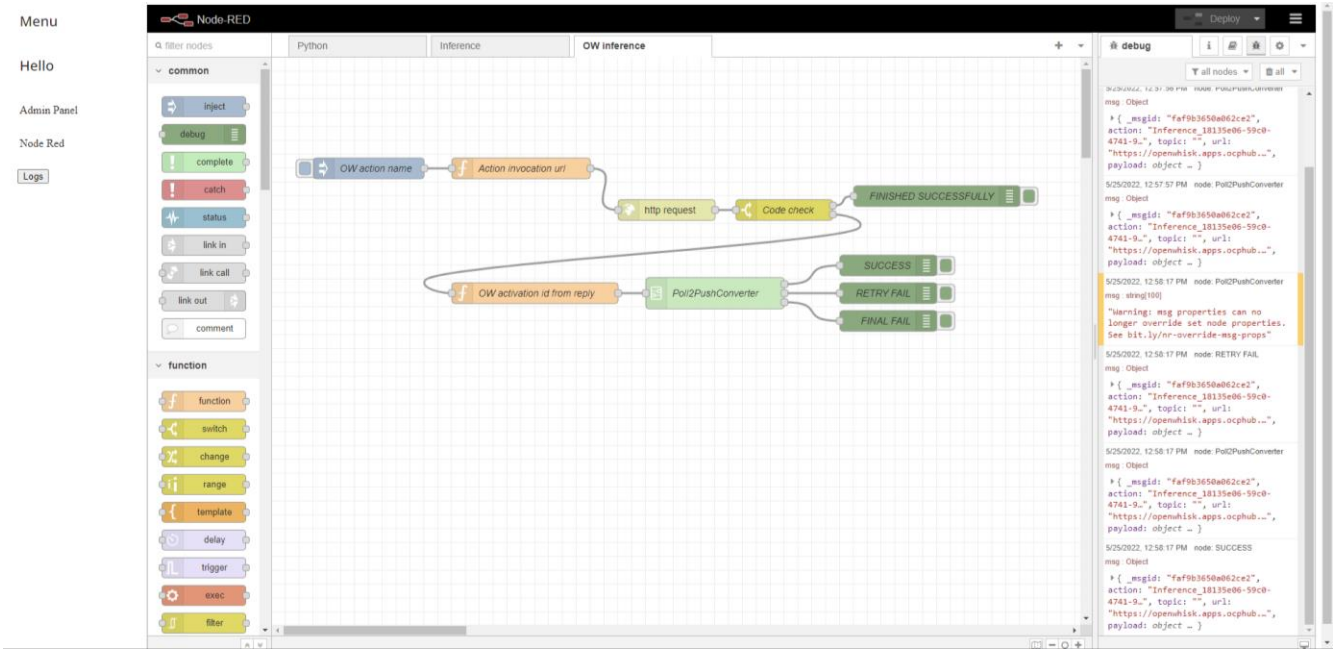

Figure 13: The OpenWhisk experimentation flow for the deployed Inference flow of Figure 11.

The flow is designed for manual-only invocation via the inject node "OW action name", where the deployed action name is given. The user can find this action name in the Jenkins build. The POST URL is reconstructed from this action name, and the expected parameters of the endpoint are given in the "Action invocation url" Javascript function. The log of executing the deployed flow is shown on the right column of Figure 13, where the final success is indicated.

4.2.3 eHealth use-case local design environment instructions

This section provides the installation and usage instructions for the local PHYSICS design environment for the eHealth use case. The system can be found in the ehealth branch of the PHYSICS Design Environment project in Gogs:

https://repo.apps.ocphub.physics-faas.eu/PHYSICS/test/src/ehealth

Access to this repository is limited to members of the PHYSICS Consortium, but may be granted upon request.

A screenshot of the repository is shown in Figure 14.

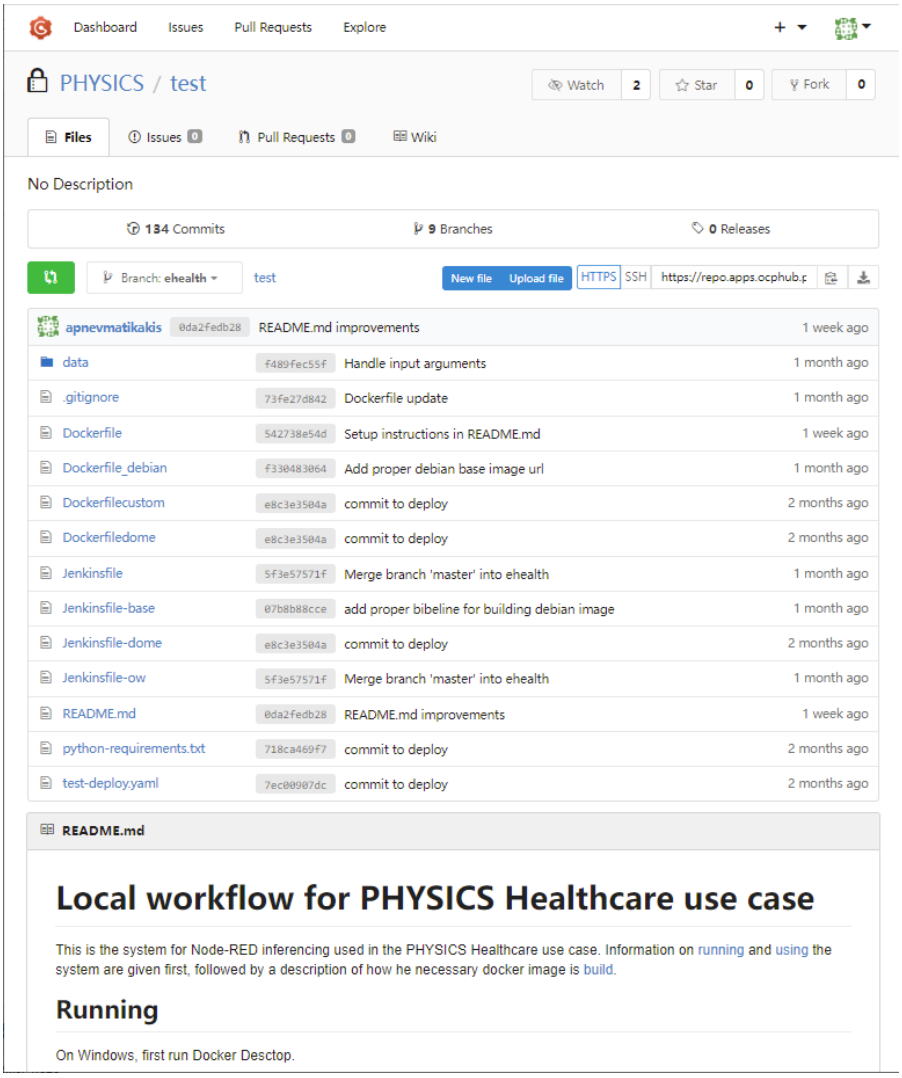

Figure 14: Screenshot of the Gogs eHealth repository.

In order to run the adapted PHYSICS Design Environment for the eHealth use case locally, first pull the repository in an ehealth directory. Add to the parent of the ehealth directory two files. The `credentials.env` environment file should have a number of tokens defined, as given by the PHYSICS project (see Code Snippet 3).

```
JENKINS_USERNAME=XXX
JENKINS_TOKEN= XXX
GIT_NODE_RED_TOKEN= XXX
```

```
GIT_NODE_RED_PATH=/repository
MINIO_ACCESS_KEY= XXX
MINIO_SECRET_KEY= XXX
MONGODB_PASSWORD= XXX
MONGODB_USER= XXX
JENKINS_PIPELINE_TOKEN= XXX
API_KEY= XXX
```

Code Snippet 3: Example credentials.env file with the required tokens to run the eHealth inference system.

The `docker-compose.yml` configuration file gives the information on how to build the three images of the system (see Code Snippet 4), namely the admin panel UI, the Node-RED and the SFG backend.

```
version: "3.9"
services:
  ui:
    container_name: ui
    image:          registry.apps.ocphub.physics-faas.eu/design-environment/control-ui/design-environment-
ui:latest
    ports:
      - "4200:4200"
  node-red:
    container_name: node-red
    build: ./<ehealth directory>
    volumes:
      - ./<ehealth directory>/data:/data
    ports:
      - "1880:1880"
  sfg:
    container_name: sfg
    image: registry.apps.ocphub.physics-faas.eu/design-environment/sfg/design-environment:latest
    volumes:
      - ./<ehealth directory>:/repository
    ports:
      - "3001:3001"
    links:
      - node-red
    env_file:
      - credentials.env
```

Code Snippet 4: Docker-compose.yml configuration file.

Where `<ehealth directory>` is replaced with the actual ehealth directory name.

Then run Docker Desktop. Open a terminal and:

- Go to the parent directory of the `<ehealth directory>`
- `docker-compose up --build`

The Node-RED image that is built also includes Python, the necessary libraries, the scripts for inference and the models. Details on using and building the system follow.

To use the system, go to `http://localhost:4200/` for the Admin Panel, or to `http://localhost:4200/node-red` for the Node-RED instance the created flows already loaded. After any useful modification of the flows, click on "Deploy" to actually store the changes in `data/flows.json`.

Deployed flows can then be built via the Admin panel (via Jenkins behind the scene) and be registered in OpenWhisk.

The Python scripts and the model data to be used in the flows are in `data/scripts`. In detail, the contents are:

- requirements.txt: It is used to setup the Python environment by installing the necessary libraries
- *.py: The different scripts to be used in the Node-RED flows

- *.joblib: Data to be used for inference and model metadata (complete model in the case of a Random Forest one)
- Neural Network models in directories

The Dockerfile found at the root of the project is used to setup the Node-RED and Python image needed for the use case. The starting point is the Debian image from the PHYSICS consortium:

```
FROM registry.apps.ocphub.physics-faas.eu/wp3/debian-base:latest
```

In there, as ROOT user, first the scripts directory is created and populated:

```
USER root
RUN mkdir -p /usr/src/node-red/scripts
COPY data/scripts /usr/src/node-red/scripts
```

Then Python is installed:

```
RUN apt install -y python3-dev python3-pip
RUN pip3 install --upgrade pip
```

To be followed by the 3rd party library requirements:

```
RUN pip3 install -r scripts/requirements.txt
```

Finally, the properties of the data folder are set and the working user is set to node-red:

```
RUN chown -R node-red /data
RUN chmod -R 775 /data
USER node-red
```

4.2.4 Concluding remarks for first version

After having described the first version of the "eHealth Prototype", we now look back at the initial functional specifications that were defined in §3.2 above. For each of the Functional Specifications, we discuss if and how the specifications are fulfilled, and – in case of a partial fulfillment – the work that remains to be done in the next iteration(s) of the prototype. The fulfillment of the functional specifications introduced in

Table 2 is discussed in Table 7.

Table 7: Fulfilment of the functional specifications for the eHealth use case.

Code	Functional Specification Fulfilment
FRS-UC2-01	Partially fulfilled. The focus of the first prototype has been on local implementation and deployment of the design environment. A complete fulfilment of the specification is expected early in the second prototype development phase as online versions of the relevant platform components become available.
FRS-UC2-02	Fulfilled. Deployed flows can then be built via the admin panel, whereupon they are registered in OpenWhisk.
FRS-UC2-03	Fulfilled via the run endpoint exposed by the Node-RED inference flow.
FRS-UC2-04	Fulfilled, since two Python scripts are provided, one for inference and another for both inference and testing, if the correct inference results are known.
FRS-UC2-05	Fulfilled. Maybe to be revisited if the input vectors should be provided directly and not in a file.

Future versions of the flow will be provided to facilitate the PHYSICS experiments and to utilize more PHYSICS patterns, as needed to showcase the benefits of the PHYSICS FaaS approach. The roadmap for future developments is given in Table 8 below.

Table 8: Development Roadmap for the eHealth use case.

	2022							2023			
Tasks	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**January**	**February**	**March**	**April**
Evaluation of inference node (for D6.6)											
Optimization of OpenWhisk deployment via PHYSICS annotations											
Optimization of service (PHYSICS flows for live input, input aggregation)											
Optimization of Machine Learning model											
Using the FaaS Service: Integration with Healthentia											

4.3 Use Case “Smart Agriculture”

4.3.1 Pipeline definition using Node-Red

The pipeline is implemented as a Node-RED flow and packaged as a subflow (Edge ETL Service). The implementation appears in Figure 15. Initially an input is provided so that the developer can plug in any kind of means to obtain the primary data value, encapsulated in the payload field of the message. Then any custom ETL logic can be applied through one or more functions and apply any needed transformation, filtering, or other operation on the data. Once this is finalized, the generic part of the pattern begins. Given that any output nodes, such as the HTTP out node used in this example, may substitute the contents of the msg.payload field with the results of the call that pushes the data to the central system, it is necessary to maintain the original data for future use (in case the transmission fails). For this reason, these are moved in the "Keep contents'' function in the msg.originalpayload field. This function is also responsible for inserting a retry counter in the message, as well as differentiating the origin of the message (new data that have arrived or past data that have failed and have been stored locally).

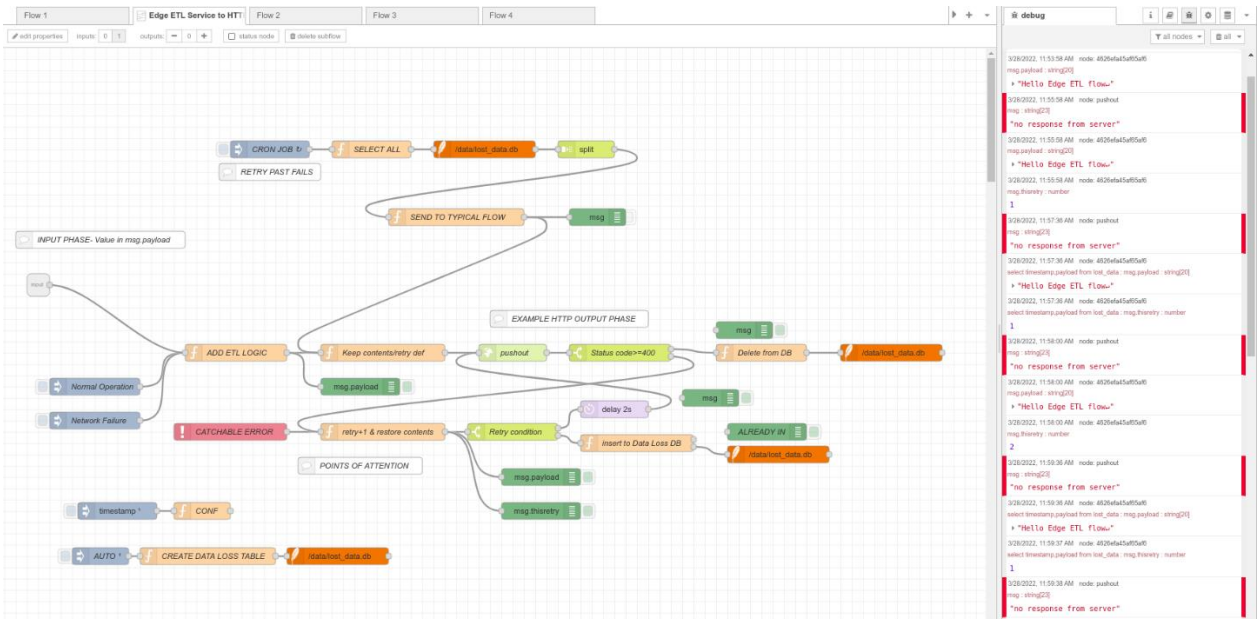

Figure 15: Implementation Flow for the Edge ETL Pattern.

Beside the adaptation of the Python script managing the parsing of the file containing the data collected by the sensors passed as an argument of the Edge ETL flow, some parameters must be defined including the frequency of pipeline triggering, the number of retries before storage of data in the local database, and the authentication information for API connection to Cybeletech database (Figure 16).

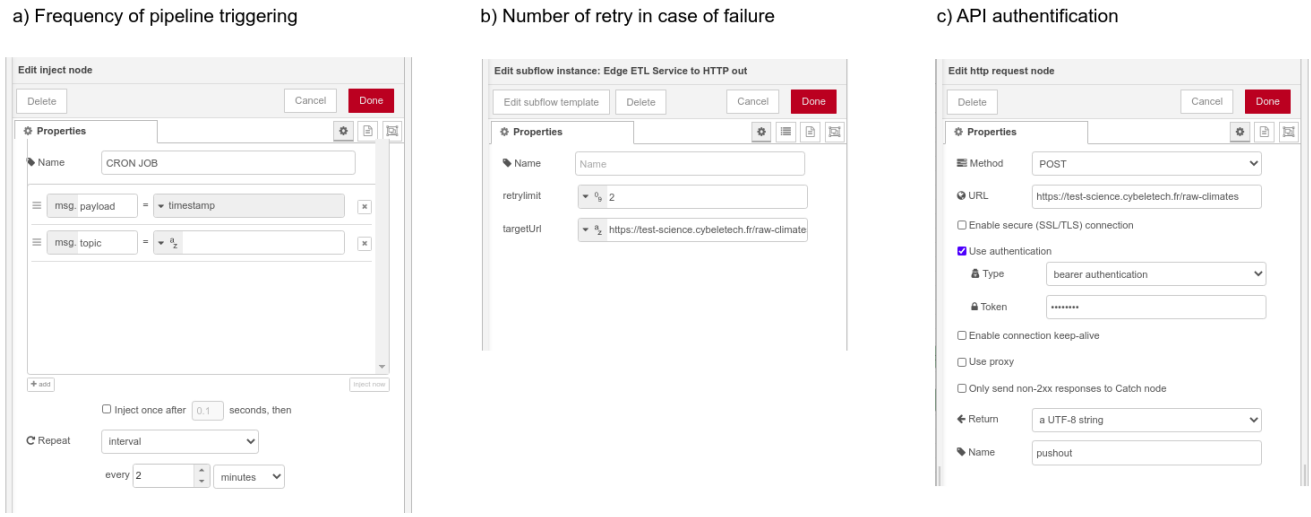

Figure 16: Settings of Node-RED ETL flow for data collection pipeline.

4.3.2 Adaptation of legacy codes

According to the greenhouse supervisor specifications, the data collection logic must be adapted. As today Cybeletech application enables to deal with two types of supervisors.

One automatically generates a plain *.txt* file at regular time intervals. In this case a Python script is run at regular time intervals with a cron. This script:

1. Read the plain *.txt* file.
2. Perform preprocessing operations to ensure compliance between sensor data collected by the supervisor and Cybeletech data model.

3. Check the existence of the preprocessed data in Cybeletech database
4. Insert the data via a dedicated SQLite driver developed by Cybeletech if they are not already in the database.

In case of connection failure between the supervisor and Cybeletech server, the plain *.txt* file is archived and will be parsed the next time the script runs.

The other provides an access to the data collected by the sensor via an API. In this case a Python script is run at regular time intervals with a cron. This script:

1. Send a request to the supervisor API.
2. Check if sensor data have been archived during the previous runs and read the archive files if they exist.
3. Perform preprocessing operations to ensure compliance between sensor data obtained with the API, and eventually from the archived files, and Cybeletech data model.
4. Insert the data via a dedicated SQLite driver developed by Cybeletech.

In case of connection failure between the supervisor and Cybeletech server the data obtained with the API are archived in a *.json* file, which will be parsed the next time the script runs.

During the adaptation phase the supervisor dependent part of the data collection logic has been extracted of the legacy codes and adapted so that it can be run using the Edge ETL Pattern.

Moreover, an API for data transfer from the supervisor to Cybeletech server has been developed so that the *http node* can be used.

Once this adaptation implemented the Python script can be passed as argument to the generic Edge ETL Pattern (see Figure 17 below).

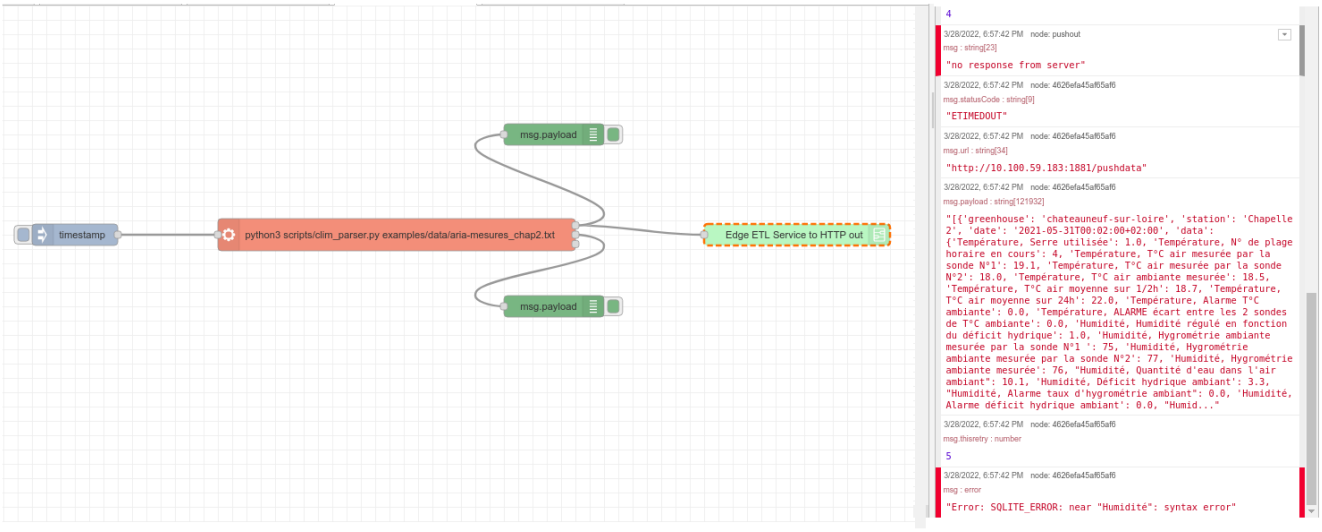

Figure 17: ETL flow testing.

As the data collection pipeline logic is defined in a generic pattern, it allows to decorrelate the part of the code dedicated to data gathering from the sensors and data preprocessing, from the part dedicated to the job running and connection failure management. This has considerably simplified the Python code base used for data collection and will ease their maintenance and adaptation to new contexts.

4.3.3 Pipeline testing using environment design

The pipeline of data collection has been created and tested using the PHYSICS Design Environment. For this test the Python script performing data parsing and a .txt file such as those produce by the supervisor of type 1 have been uploaded in the /data repository of the test project.

Three scenarios have been tested:

1. In the first scenario it was assumed that no connection failure occurs during the data collection procedure. It allows to validate the integration of the Python script in the Edge ETL Pattern and the configuration of the nodes.
2. In the second scenario it was assumed that temporary connection failure occurs during the data collection procedure. It allows to validate that the retry procedure allows to overcome transient connection troubleshooting.
3. In the third scenario it was assumed that extended connection loss occurs during the data collection procedure. It allows to validate that while the API cannot be reached the data are stored locally with no duplicate and that these data are sent to the distant database as soon as the connection is restored.

The outcome of the first scenario is presented in Figure 18. In this case the pipeline succeeded, and all the data collected by the sensors have been preprocessed and sent to Cybeletech database using the API at the first try.

Figure 18: Example of data returned by the greenhouse supervisor and outcome of the Node-RED ETL flow.

The outcome of the second scenario is presented in Figure 19. In this scenario the pipeline failed three times, and at the fourth attempt, when the connection has been restored, data collected by the sensors have been preprocessed and sent to Cybeletech database using the API.

Figure 19: Example of data stored in the local database in case of connection failure and outcome of the Node-RED ETL flow.

In scenario 3 the pipeline failed five times, which is the maximum number of retries. The data are then stored in a local SQlite instance. After a while the pipeline is re-run and the data stored in the local database as well as the new data were sent to Cybeletech database.

4.3.4 Concluding remarks for first version

The source material for the prototype as desribed here can be found on the PHYSICS project internal Git repository, here:

https://repo.apps.ocphub.physics-faas.eu/PHYSICS/test/src/smart_agriculture/

Access to this repository is limited to members of the PHYSICS Consortium, but may be granted upon request.

After having described the first version of the “smart agriculture Prototype”, we now look back at the initial functional specifications that were defined in §3.3 above. For each of the Functional Specifications, we discuss if and how the specifications are fulfilled, and – in case of a partial fulfillment – the work that remains to be done in the next iteration(s) of the prototype. The fulfillment of the functional specifications introduced in 2 is discussed in Table 4 below.

Table 9: Fulfillment of the Functional requirements for the Smart Agriculture use case.

Code	Functional Requirements
FRS-UC3-01	Fulfilled. A first script for data collection has been integrated in the Node-RED ETL flow.
FRS-UC3-02	Fulfilled. The pipeline runs as long as the docker is up with a user-defined time step and is resilient to connection failure.
FRS-UC3-03	Fulfilled. A first deployment has been performed on Cybeletech by using the PHYSICS Design Environment. Moreover, the image built during the deployment phase has been pull and used to deploy the data collection pipeline.
FRS-UC3-04	Work in progress. The focus of the first prototype has been on the data collection part of the pipeline. Beside the adaption of legacy codes used to perform simulation and optimization. Cybeletech is currently working on the definition of a deployment procedure for Python codes using C++ internal libraries in the context of PHYSICS.

FRS-UC3-05	Work in progress. Similar to FRS-UC3-04, the implementation for Prototype V1 has focused on data collection pipeline. Deployment of the simulation /optimization flow using OpenWhisk is a next step that can be taken early in the second prototype development phaseusing a simplified version of crop growth model. These models will be complexified during the intensification phase.
FRS-UC3-06	Work in progress. Similar to FRS-UC3-04, the implementation for Prototype V1 has focused on data collection pipeline. Adaptation of the optimization procedure for compliance with the Node-RED SplitJoin pattern is in progress.

And finally, looking forward to the next development phase(s), we foresee the following development roadmap (see Table 10).

Table 10: Development Roadmap for the Smart Agriculture use case.

	2022							2023			
Tasks	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**January**	**February**	**March**	**April**
Run of pure python light model as function											
Run of pure python light model as a service											
Optimization procedure using splitJoin pattern with light model											
Integration of C++ complex models in PHYSICS docker image											
Run of C++ complex models as a service											
Run of C++ complex models as a function											
Optimization procedure using splitJoin pattern with complete model											
Adaptation of calibration pipeline											

5 CONCLUSIONS

This document accompanies the delivery of the very first three demonstrators that have been developed to run on top of the PHYSICS Platform. The Smart Manufacturing use case demonstrates a quality control scenario, the eHealth use case an online prediction scenario, while the Smart Agriculture use case focuses on data collection pipelines, all using Node-RED flows to model these processes as a series of functions.

These first demonstrators are snapshots of ongoing iterative and "experimental" work to implement the different use cases in a "FaaSified" manner using the PHYSICS Platform. As such, they are not to be considered "final" in any way. The Task 6.3, focusing on *Use Cases Adaptation & Experimentation* continues to run until M34 (October 2023), nearly the end of the project, at which point in time the follow-up deliverable to this document (D6.6: PHYSICS Application Prototype V2) will describe the final demonstrators including further additions and functionalities provided by the PHYSICS platform.

This deliverable describes the adaptation of the application components in the PHYSICS environment, the functional requirements per each use case and the current or foreseen usage of the PHYSICS tools. The first prototypes are documented using the visual workflow design of PHYSICS and they all represent different utilization of the developed components. In addition, a detailed roadmap of implementation and integration of further functionalities is presented by each use case to show what will be done in the timespan from June 2022 to April 2023. The next version of this deliverable, as well as the others part of WP6, will show the development of the use cases in respect to expected results.

6 BIBLIOGRAPHY

[1] N. Franke, A. Hennecke, V. Gezer, C. Harms, H. op den Akker, A. Pnevmatikakis, G. Labropoulos, T. Lohier, M. Patiño, Y. Poulakis and M. Touloupou, "D6.3 - Application Scenarios Definition V1," PHYSICS Consortium, 2021.

[2] M. Patiño, A. Azqueta, L. Mengual, T. Li, G. Kousiouris, S. Tsarsitalidis, E. Boutas, T. Stamati, C. Giannakos, J. Salomon, L. Tomas, A. Mamelli, D. Costantino, Y. Georgiou and Pelegr, "D2.4 - PHYSICS Reference Architecture Specification V1," PHYSICS Consortium, 2021.

[3] A. Mamelli, D. Costantino, J. Salomon, L. Tomas Bolivar, A. Castillo Nieto and C. Sánchez Fernández, "D6.1 - Prototype of the Integrated PHYSICS solution framework and RAMP V1," PHYSICS Consortium, 2022.

DISCLAIMER

COPYRIGHT MESSAGE