OPTIMIZED HYBRID SPACE-TIME SERVICE CONTINUUM IN FAAS

D6.6 – PHYSICS APPLICATION PROTOTYPE V2

Lead Beneficiary	iSPRINT
Work Package Ref.	WP6 – UC Adaptation, Experimentation, Evaluation
Task Ref.	T6.3 – Use Cases Adaptation & Experimentation
Deliverable Title	D6.6 – PHYSICS Application Prototype V2
Due Date	2023-11-30
Delivered Date	2023-11-30
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

CONTRIBUTING PARTNERS

Partner Acronym	Role[1]	Name Surname[2]
iSPRINT	Lead Beneficiary	Aristodemos Pnevmatikakis, George Labropoulos
FTDS		André Hennecke, Niklas Franke
CYBE		Théophile Lohier
DFKI		Volkan Gezer, Carsten Harms, Maciej Kolek
UPM		
INNOV		Ariana Polyviou
GFT		

REVISION HISTORY

Version	Date	Partner(s)	Description
0.1	2023-10-04	iSPRINT	Initial version based on D6.5
0.2	2023-10-31	DFKI	Updated Smart Manufacturing use case
0.3	2023-11-01	iSPRINT	Updated eHealth use case
0.4	2023-11-02	CYBE	First version of updates to smart agriculture use case
0.5	2023-11-04	iSPRINT	Harmonization, sections common to all use cases
0.6	2023-11-08	CYBE	Final version of updates to smart agriculture use case
1.0	2023-11-09	iSPRINT	Ready for internal review
1.1	2023-11-19	iSPRINT	Incorporating general review comments and improving section 4.2.2
1.2	2023-11-20	DFKI	Incorporating review comments for smart manufacturing
1.3	2023-11-22	CYBE	Incorporating review comments for smart agriculture
2.0	2023-11-29	iSPRINT	Version for QA
2.1	2023-11-30	INNOV	QA
2.2	2023-11-30	iSPRINT	Integrated QA comments
3.0	2023-11-30	GFT	Version ready for the submission

[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 adoption by applying more advanced computing models and cover a wide and diverse range of available edge resources (e.g., small IoT sensors, mobile devices, or Edge nodes in diverse clusters).

The current document D6.6: PHYSICS Application Prototype V2 is the final 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 – 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 that have been fulfilled.
- 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 used to implement the application flow, and how the use cases are deployed in the PHYSICS Platform. For each use case, the fulfillment of functional requirements is detailed, leading to the final versions of the services for evaluation.

CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION

This document, D6.6: PHYSICS Application Prototype V2, accompanies the final 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 D6.4: PHYSICS Application Scenarios Definition V2 (Franke, et al., 2022). 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 D6.4 document. We also built this document on top of its predecessor, D6.5: PHYSICS Application Prototype V1 (op den Akker, et al., 2022), resulting in a single document containing all relevant information on experimentation.

Although requirements have been captured in a very systematic way, this Use Case "Adaptation & Experimentation" is indeed a more experimental process. 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 to maintain a document that is self-contained, we aim to provide a quick overview of the PHYSICS platform and architecture in Section 2 of this document.

The core contents of this deliverable describe the final 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 is part of the deliverable, accompanying those prototypes and serving as a guide for the preparation and experimentation leading to their delivery for evaluation.

1.2 Insights from other Tasks and Deliverables

Figure 1 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 D2.5: PHYSICS Reference Architecture Specification V2 (Patiño, et al., 2022) – describing the reference architecture of PHYSICS and D6.2: Prototype of the Integrated PHYSICS solution framework and RAMP V2 (Mamelli, et al., 2023) – describing a recent state of the integrated PHYSICS platform, as well as D6.4: Application scenarios definition V2 (Franke, et al., 2022) - describing the definitions and requirements for the use cases.

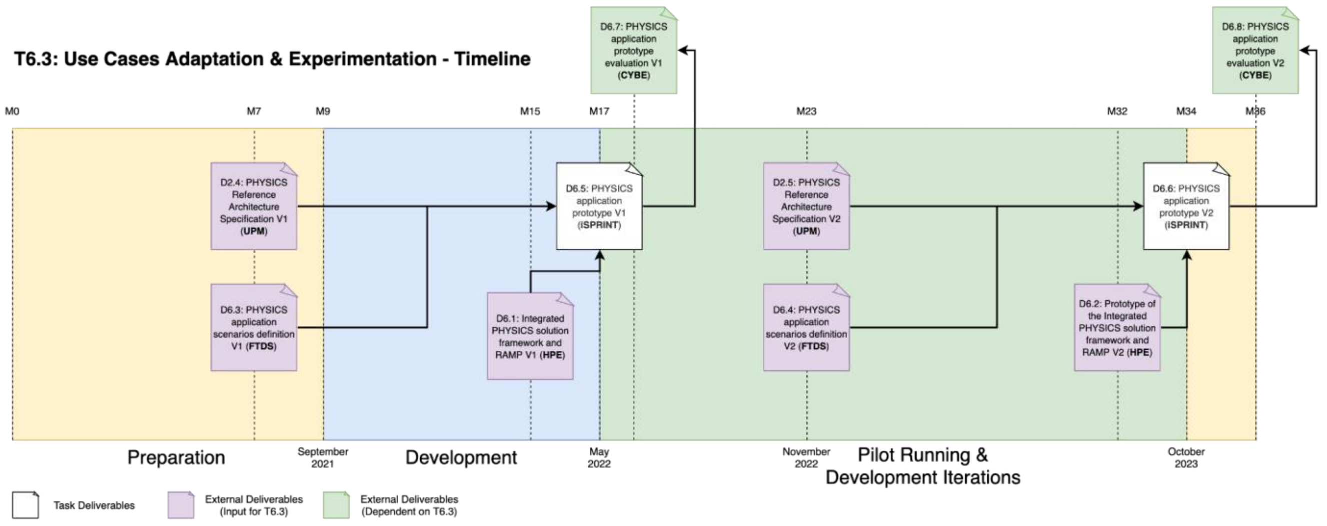

T6.3: Use Cases Adaptation & Experimentation - Timeline
M0
M7
M9
M15
M17
M23
M32
M34
M36
D6.7: PHYSICS application prototype evaluation V1 (CYBE)
D6.8: PHYSICS application prototype evaluation V2 (CYBE)
D2.4: PHYSICS Reference Architecture Specification V1 (UPM)
D6.3: PHYSICS application scenarios definition V1 (FTDS)
D6.1: Integrated PHYSICS solution framework and RAMP V1 (HPE)
D6.5: PHYSICS application prototype V1 (iSPRINT)
D2.5: PHYSICS Reference Architecture Specification V2 (UPM)
D6.4: PHYSICS application scenarios definition V2 (FTDS)
D6.2: Prototype of the Integrated PHYSICS solution framework and RAMP V2 (HPE)
D6.6: PHYSICS application prototype V2 (iSPRINT)
Preparation
September 2021
Development
May 2022
November 2022
Pilot Running & Development Iterations
October 2023
Task Deliverables
External Deliverables (Input for T6.3)
External Deliverables (Dependent on T6.3)

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.

1.4 Changes since D6.5

The changes in this document compared to the previous version (op den Akker, et al., 2022) are grouped per use case. For the smart manufacturing, the changes are as follows:

- Minor changes in section 3.1.
- Input & Output JSON specifications updated in section 4.1.1.
- Updated existing flow to final version in section 4.1.1.
- Added newly implemented flow in section 4.1.2.
- Updated function requirement fulfillment section 4.1.3.

For the eHealth use case, the main area of update is the optimization and finalization of the inference scenario, as well as the introduction of two more scenarios on phenotyping and synthesis. In detail the following sections have been updated as explained:

- Introduced patient phenotyping and data synthesis cases alongside inferencing as goals of the use case in section 3.2.1.
- Updated the specifications of the use case in section 3.2.2.
- Updated sections 4.2.1 and 4.2.2, to include the flows on phenotyping and synthesis. Also updated the figures therein to reflect the latest version of the design environment.
- Completely re-written section 4.2.4, to reflect the final experimentation results.

Finally, for the smart agriculture use case the main area of update is the introduction of two more pipelines. The changes are as follows:

- Minor updates in section 4.3.1 (introduction of subsections for conformity)
- Implementation, deployment and testing of simulation pipeline as a service in a new section 4.3.2.
- Implementation, deployment and testing of calibration pipeline as a service in a new section 4.3.3.
- Parallelization of the calibration pipeline taking advantage of FaaS in section 4.3.3.

There are also horizontal changes across the use cases, that are as follows:

- The introduction section 1 is updated.
- This section (1.4) is new.
- The references have been updated throughout the document.
- The concluding section 5 is re-written.

2 OVERVIEW OF PHYSICS ARCHITECTURE

The PHYSICS platform architecture is described in D2.5: PHYSICS Reference Architecture Specification V2 (Patiño, et al., 2022). In the rest of this section we provide a short summary of the overall objectives for the PHYSICS project and platform (section 2.1), as well as the platform architecture (section 2.2), in order to help better understand the prototypes as described 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, 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 (Mamelli, et al., 2023)), 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 (Mamelli, et al., 2023). The main components of the PHYSICS architecture are shown in Figure 2. There three layers are depicted 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 parallelization, 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.

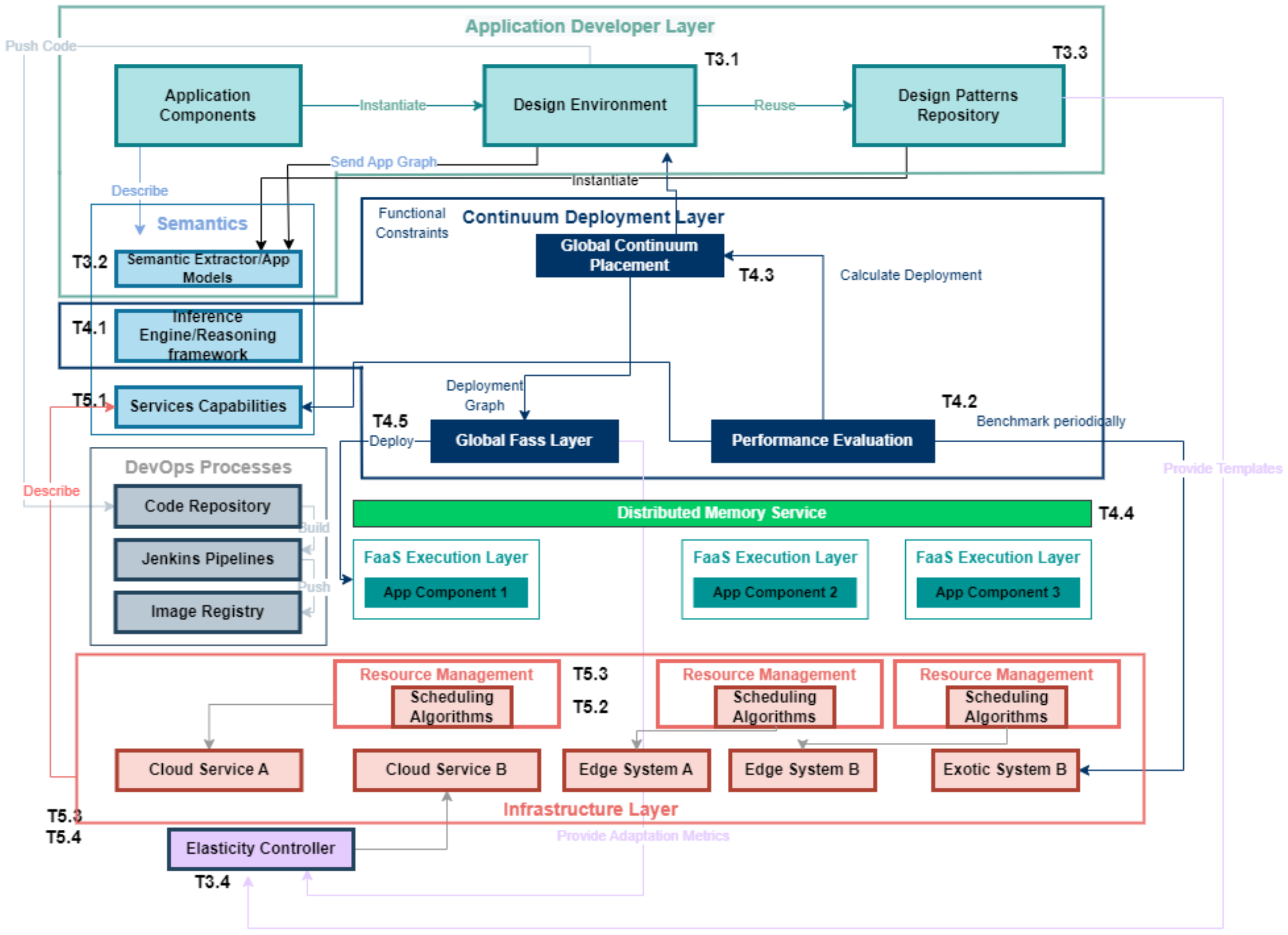


Figure 2: The PHYSICS Software Components' Architecture.

The Continuum Deployment Layer oversees providing uniform access to the diverse cloud services provided by one or more cloud providers. The Global Continuum Placement oversees deciding on the most suitable deployment of applications taking into account the performance of the services, costs and 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 maximize 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 following three sub-sections, 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 finalized earlier in D6.4 (Franke, et al., 2022).

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 implements 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 does 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. The first prototype implemented only the second scenario since it had minimum dependency with the development progress of the PHYSICS components. Now that all required PHYSICS components are available, both smart manufacturing scenarios are implemented.

The "to-be" BPMN diagrams for the scenario have been defined in Deliverable 6.3 (Franke, et al., 2022). For easier trackability, the diagram of the second scenario is also given in Figure 3.

To realize the Smart Manufacturing use cases, the functional requirements shown in the Table 1 are defined.

[3] Or "score" of AI-based results.

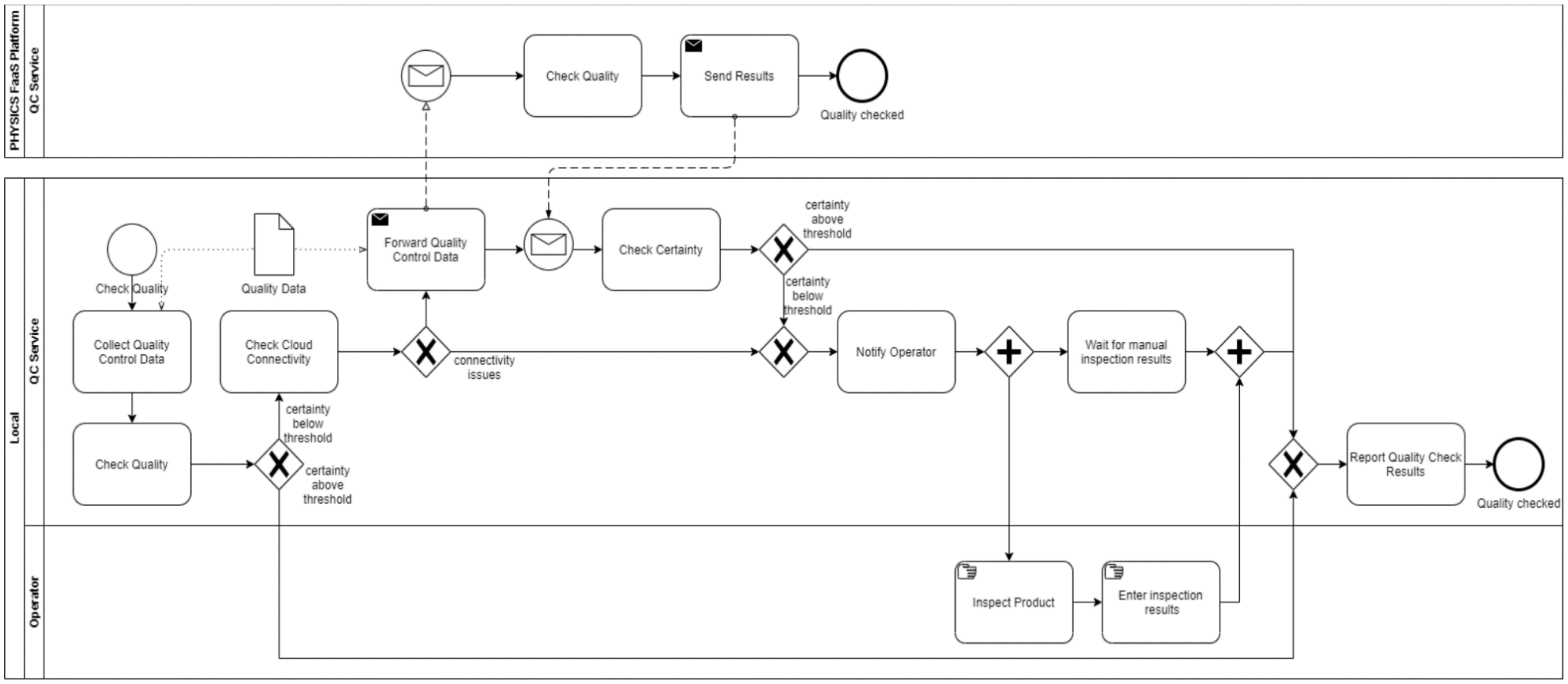


Figure 3: BPMN Diagram of Scenario #2 of the Smart Manufacturing Pilot.

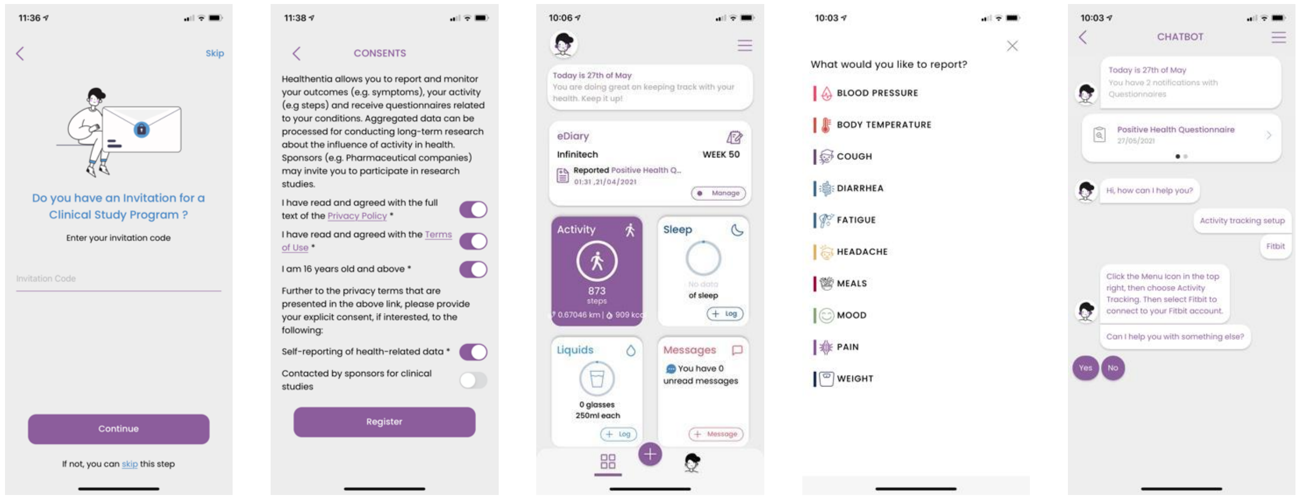
Table 1: Functional Requirements of the Smart Manufacturing Use Case.

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.

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 (Franke, et al., 2022)).


11:36
Skip
Do you have an Invitation for a Clinical Study Program ?
Enter your invitation code
Invitation Code
Continue
If not, you can skip this step
11:38
CONSENTS
Healthentia allows you to report and monitor your outcomes (e.g. symptoms), your activity (e.g steps) and receive questionnaires related to your conditions. Aggregated data can be processed for conducting long-term research about the influence of activity in health. Sponsors (e.g. Pharmaceutical companies) may invite you to participate in research studies.

I have read and agreed with the full text of the Privacy Policy *

I have read and agreed with the Terms of Use *

I am 16 years old and above *

Further to the privacy terms that are presented in the above link, please provide your explicit consent, if interested, to the following:

Self-reporting of health-related data *

Contacted by sponsors for clinical studies

Today is 27th of May
You have 2 notifications with Questionnaires

Positive Health Questionnaire
27/05/2021

Hi, how can I help you?

Activity tracking setup

Fitbit

Click the Menu Icon in the top right, then choose Activity Tracking. Then select Fitbit to connect to your Fitbit account.

Can I help you with something else?

Figure 4: Screenshots of the Healthentia mobile application.

Deliverable 6.3 (Franke, et al., 2022) provides all details and requirements for this Use Case. In order to better understand the provided functional specifications and prototype description (see section 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 – a conversational agent that can discuss the prediction with the user in natural language dialogues.

A healthcare professional is interested in grouping their patients to handle them. Generative models are learnt to describe the derived clusters of patients. The models corresponding to clusters that healthcare professionals find meaningful are termed phenotypes. Patients are then grouped into those clusters based on the similarity of their data to the different generative data, a process called phenotyping.

A data engineer is interested in working with Healthentia data. To do so, data is synthesized from the generative models.

3.2.2 Design & Specifications

The goal of the eHealth use case is thus to demonstrate the benefits of the PHYSICS FaaS approach on three scenarios:

- Inference for patients
- Patient phenotyping
- Data synthesis

To implement all three of them, several use case specific functional requirements have been derived, as shown in Table 2. 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	Service must be provided to be invoked by third parties
FRS-UC2-02	The services are to be deployed according to the principles of the PHYSICS project through OpenWhisk
FRS-UC2-03	The services are to be utilized via the endpoints exposed by functions deployed from Node-RED flows
FRS-UC2-04	The flows will utilize ML scripts written in Python
FRS-UC2-05	The ML scripts are to be invoked by specifying the model(s) to be used and if applicable, the vectors to infer upon

The fulfillment 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.

D6.4 (Franke, et al., 2022) provides all details and requirements for this Use Case. To better understand the provided functional specifications and prototype description (see section 4) we include a summary of a typical usage scenario here.

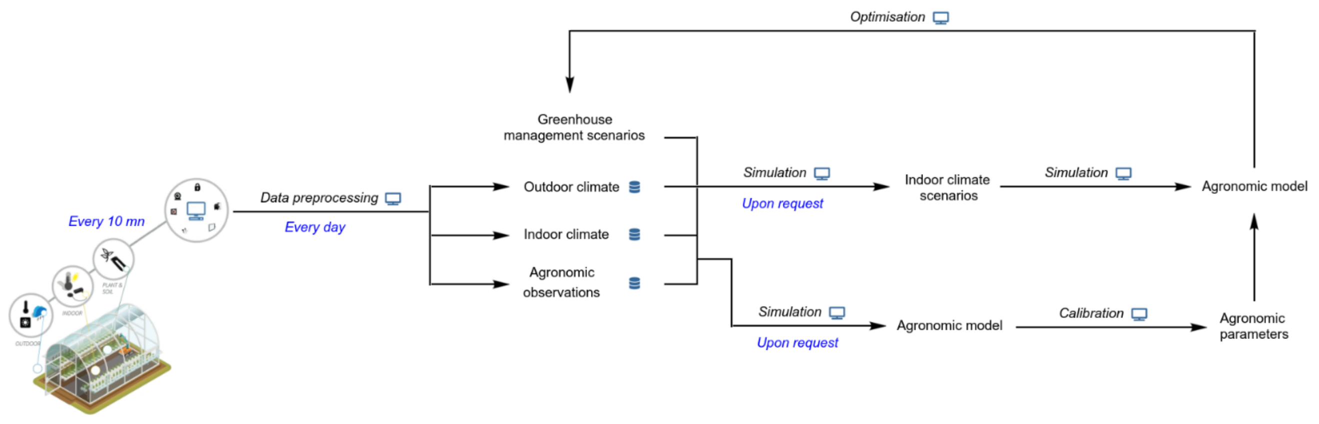


Figure 5: Overview of the Cybeletech solution for greenhouses.

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 best 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.

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

As the first prototype, the emphasis was on adapting the pipeline for data collection. The fulfillment 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 and are imported to PHYSICS with the "Custom Image Upload" functionality.

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 "/run" (standard interface for any OpenWhisk function). An example of the request can be seen in Code Snippet 1. Apart from the base64-encoded image, it also contains the expected product configuration depending on the customer order.

```
 {
   "mime": "image/jpeg",
   "encoding": "base64",
   "image": "base64-encoded input image",
   "expected": [
     {
       "name": "UsbPenDrive 2x4 Blue",
       "type": "USB-PenDrive",
       "color": "blue",
       "size": "2x4",
       "horizontal position": 0,
       "vertical position": 0
     },
     {
       "name": "FlatStone 2x4 Black",
       "type": "FlatStone",
       "color": "black",
       "size": "2x4",
       "horizontal position": 0,
       "vertical_position": 1
     }
   ]
}
```

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

After each quality control operation, the certainty of the results is checked. An example output is given in Code Snippet 2. Apart from the processed base64-encoded image, it contains the Object Detection results (bounding boxes, labels, and confidence of prediction) for each object as well as the time taken in milliseconds for performance measurement purposes.

```
 {
   "complex": true,
   "encoding": "base64",
   "image": "base64-encoded output image"
   "mime": "image/jpeg",
   "results": [
     {
       "confidence": 0.9248585104942322,
       "label": "UsbPenDrive 2x4 Blue",
       "x0": 2,
       "x1": 778,
       "y0": 687,
       "y1": 877
     },
     {
       "confidence": 0.9240514039993286,
       "label": "FlatStone 2x4 Black",
       "x0": 0,
       "x1": 777,
       "y0": 413,
       "y1": 701
     }
   ],
   "time taken": 0.66082,
   "version": "1.3.0"
}
```

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

The resulting array scores are inspected, and the minimum confidence of all results is used for "certainty". Based on the threshold, in the following manner:

1. After simple QC
   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 complex QC.
2. After complex QC
   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.

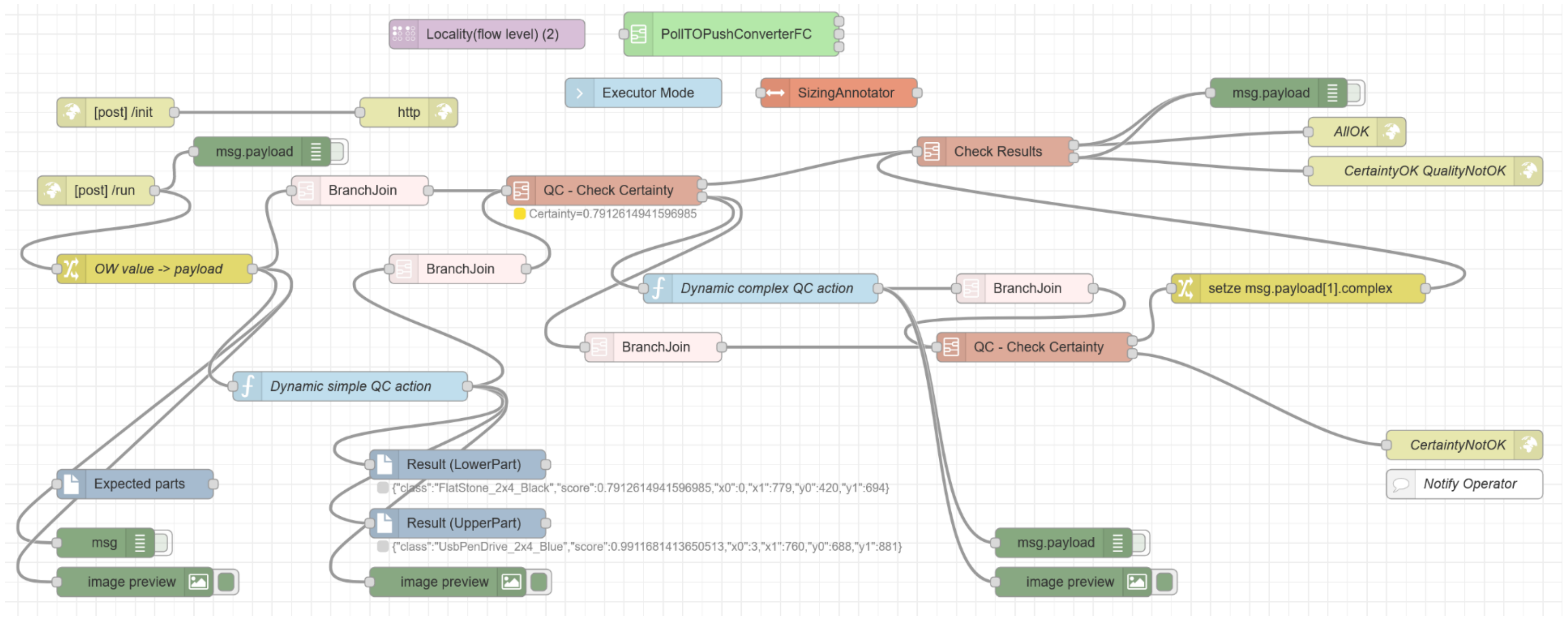
Based on the conditions written above, three different status codes are generated. The computation results are summarized in Table 4.

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 shows a prototype of the second scenario of manufacturing use case depicted in BPMN diagram (See Figure 3). The two "Dynamic simple/complex QC action" nodes are used to call the actual QC functions (custom images) dynamically via PHYSICS "Dynamic OW action". The sub flow "QC – Check Certainty" checks whether the certainty is above the threshold. The "Check Results" sub flow checks whether the results of the QC function match the expected product parts. Sub flows were used for reusability and status reported for debugging. Note that the notification of the operator is handled outside of this flow.



```
{"class":"FlatStone_2x4_Black","score":0.7912614941596985,"x0":0,"x1":779,"y0":420,"y1":694}
{"class":"UsbPenDrive_2x4_Blue","score":0.9911681413650513,"x0":3,"x1":760,"y0":688,"y1":881}
```

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

In Figure 7, the “QC - Check Certainty” 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 occurs, yellow if certainty is below the threshold and green otherwise.

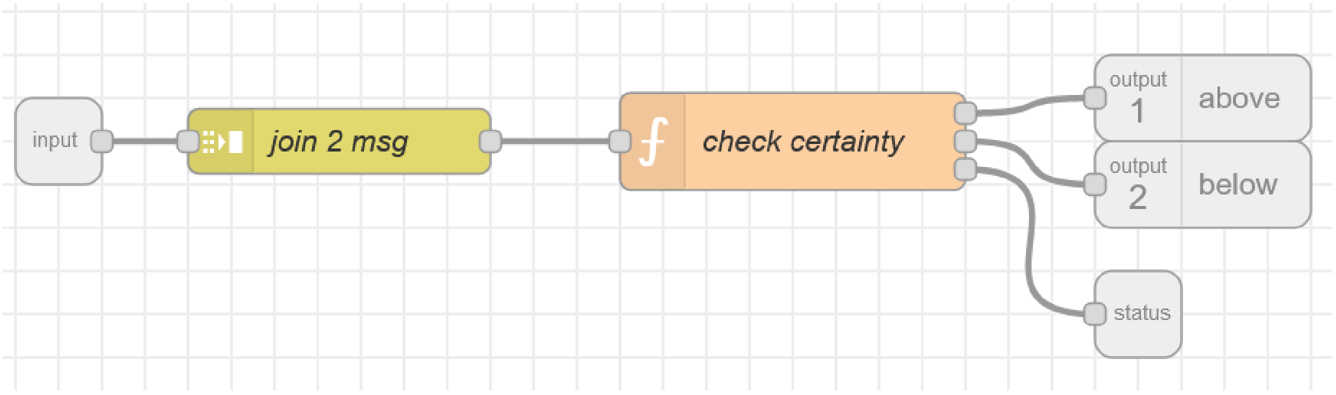


Figure 7: “QC - Check Certainty” sub flow.

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

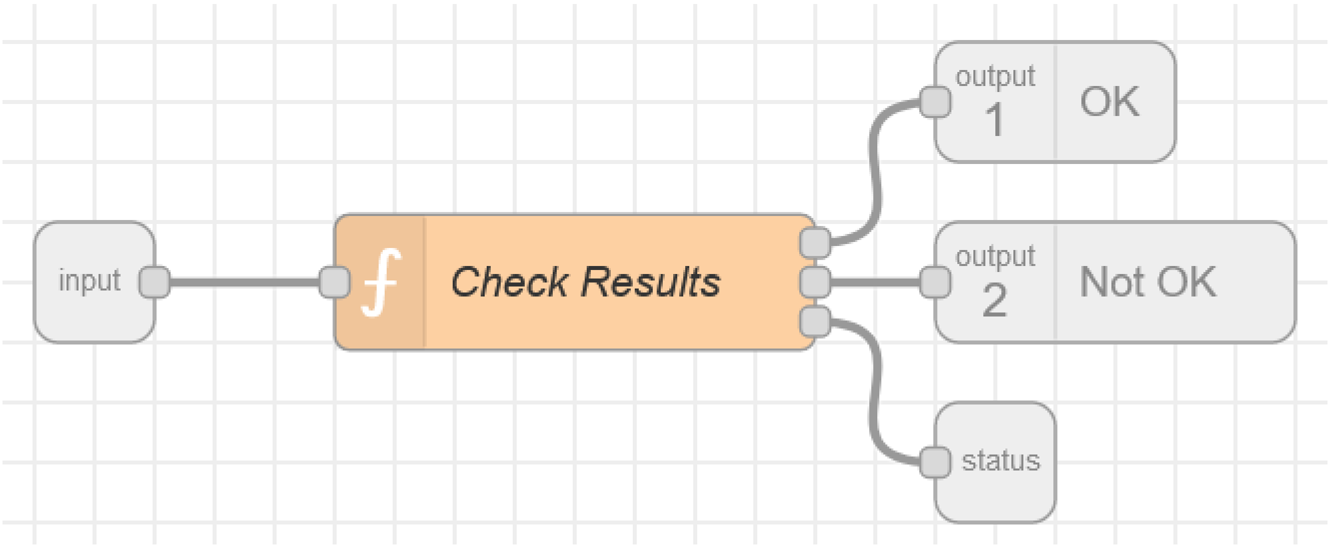


Figure 8: “Check Results” sub flow.

4.1.2 Fail-Over PHYSICS Flow

The complete Quality Check function discussed in section 4.1.1 is deployed both on the local edge instance of PHYSICS as well as in the cloud. The flow deciding which instance to use is shown in Figure 9. The “Edge OW MONITOR AND LOGGER” PHYSICS component monitors the local OpenWhisk instance. The “Router” PHYSICS component uses this information to route incoming requests according to its configuration. It is setup such that if the edge instance is not available or overloaded, the cloud instance is used instead (see in Figure 10). Further information about the router component can be found in Deliverable D3.2 Section 4.12. Because the OW monitor needs to run constantly, this flow is deployed as a service rather than a function.

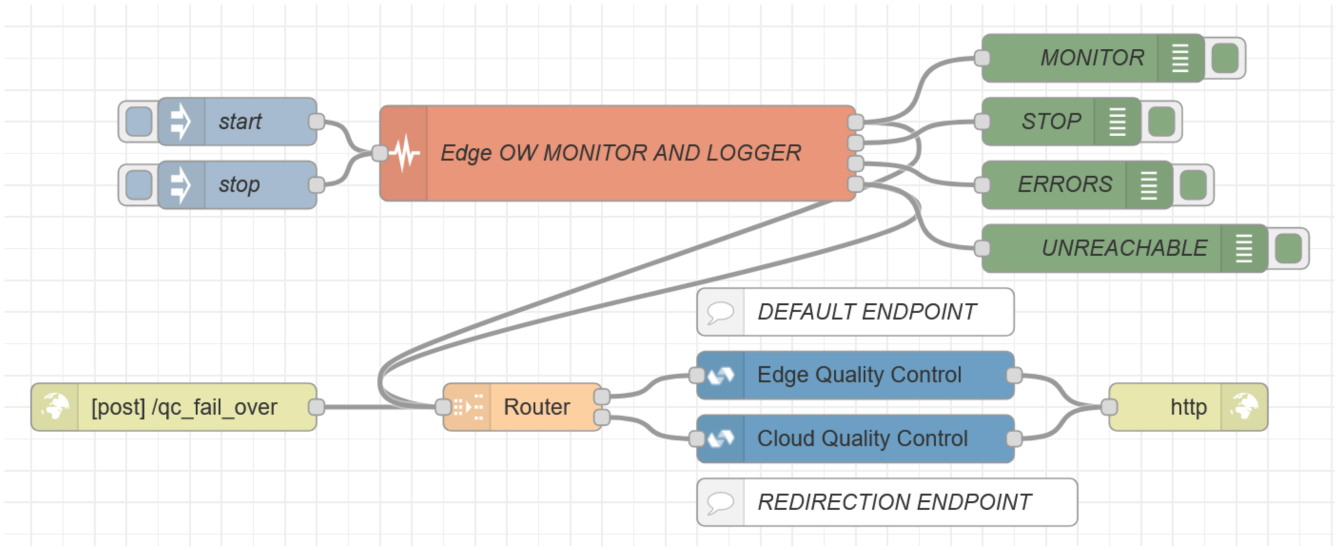
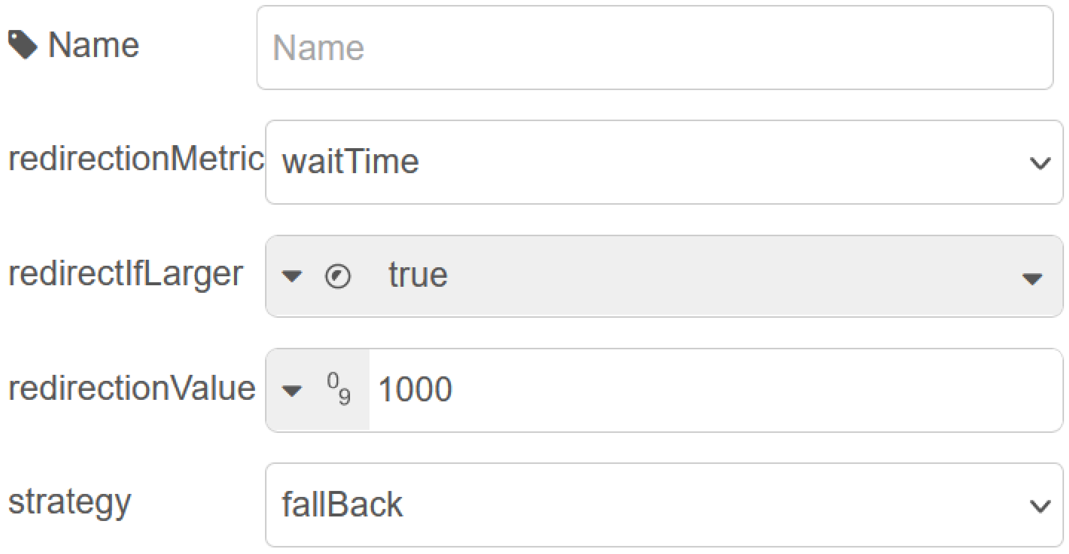


Figure 9 Fail-Over flow for Smart Manufacturing Use Case.



Name	Name
redirectionMetric	waitTime
redirectIfLarger	true
redirectionValue	1000
strategy	fallBack

Figure 10 Fail-Over Router setup for Smart Manufacturing Use Case.

4.1.3 Concluding remarks

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 prototypes and the functional requirements defined in section 3.1.2, a summary of fulfilment statuses of the requirements is shown in Table 5.

Table 5: Fulfillment of the functional requirements for the Smart Manufacturing use case.

Code	Functional Specification Fulfilment
FRS-UC1-01	Fulfilled. Endpoint is available at Kubernetes cluster with POST to "/qc".
FRS-UC1-02	Fulfilled. Final version of Node-RED flows and quality control inference as OW-Action deployed.
FRS-UC1-03	Fulfilled. Integrated into the Node-RED flows.
FRS-UC1-04	Fulfilled. The inference takes place in a custom docker-based OpenWhisk action.
FRS-UC1-05	Fulfilled. Two separate functions for simple and complex QC are used instead of specifying the model.
FRS-UC1-06	Fulfilled. Both edge and cloud PHYSICS deployments use identical functions. Fail-Over is implemented using the PHYSICS Router Pattern.
FRS-UC1-07	Fulfilled. PHYSICS Router is setup to prefer edge deployment unless it is not available or overloaded.

The smart manufacturing use case has successfully fulfilled the respective functional requirements. Several components of the PHYSICS platform are utilized to exploit the FaaS benefits.

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 fulfillment of the use case functional specifications is 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 11.

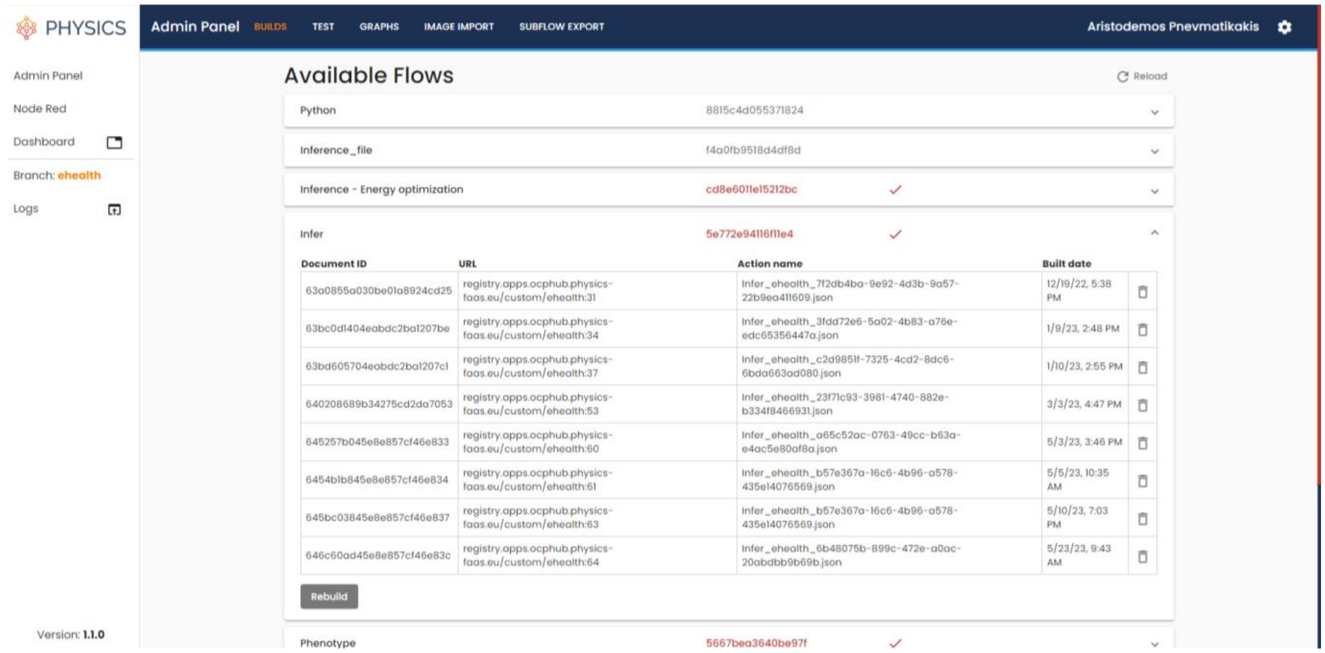
Selecting any of the flows allows the user to build the flow using Jenkins (see Figure 12) and then have the resulting image deployed at OpenWhisk, whereupon the flow is available for inference both from the Node-RED environment, but also via external invocation.

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 eHealth use case) are loaded, together with the PHYSICS Node-RED components. This is shown in Figure 13.

Three of the flows of the editor correspond to the inference, phenotyping and synthesis cases. Each of those flows are 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 14.



Available Flows

Python 8815c4d055371824

Inference_file f4a0fb9518d4df8d

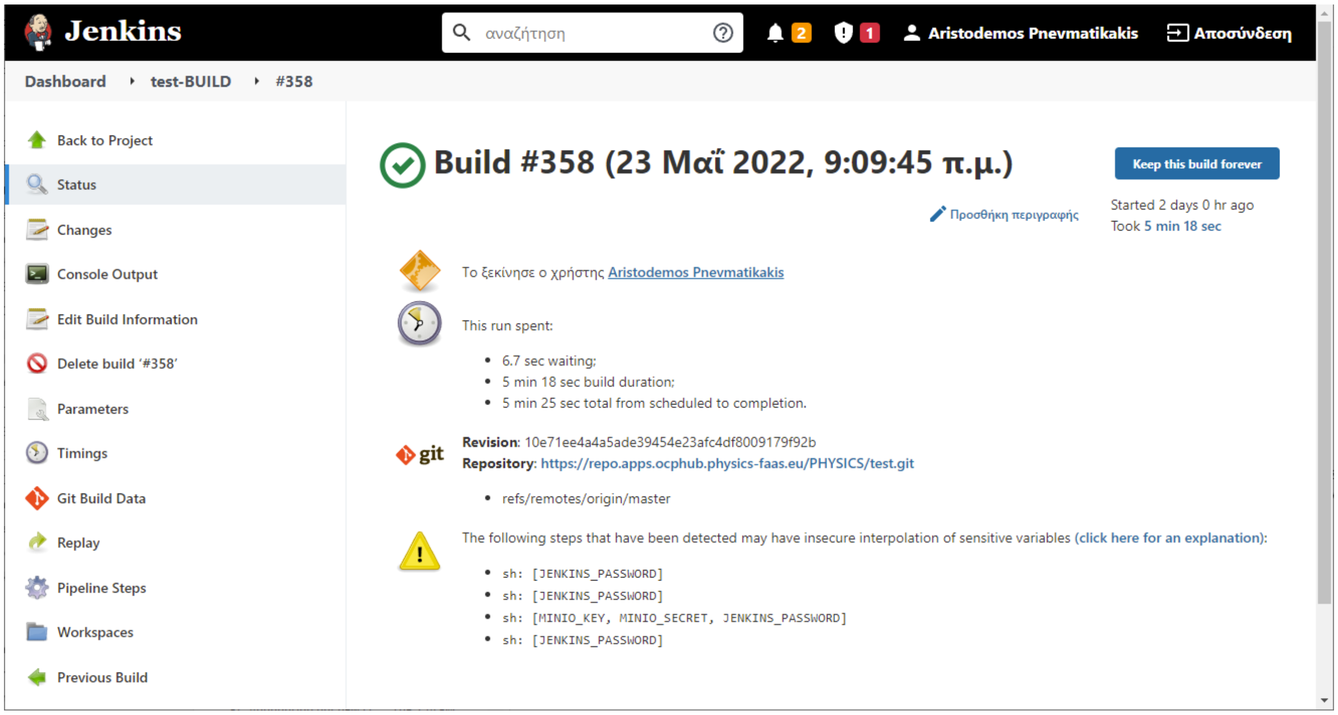
Inference - Energy optimization cd8e6011e15212bc ✓

Infer 5e772e94116f11e4 ✓

Document ID	URL	Action name	Built date	
63a0855a030be01a8924cd25	registry.apps.ocphub.physics-faas.eu/custom/ehealth:31	Infer_ehealth_7f2db4ba-9e92-4d3b-9a57-22b9ea411609.json	12/19/22, 5:38 PM	🗑
63bc0d1404eabdc2ba1207be	registry.apps.ocphub.physics-faas.eu/custom/ehealth:34	Infer_ehealth_3fdd72e6-5a02-4b83-a76e-edc65356447a.json	1/9/23, 2:48 PM	🗑
63bd605704eabdc2ba1207c1	registry.apps.ocphub.physics-faas.eu/custom/ehealth:37	Infer_ehealth_c2d9851f-7325-4cd2-8dc6-6bda663ad080.json	1/10/23, 2:55 PM	🗑
640208689b34275cd2da7053	registry.apps.ocphub.physics-faas.eu/custom/ehealth:53	Infer_ehealth_23f71c93-3981-4740-882e-b334f8466931.json	3/3/23, 4:47 PM	🗑
645257b045e8e857cf46e833	registry.apps.ocphub.physics-faas.eu/custom/ehealth:60	Infer_ehealth_a65c52ac-0763-49cc-b63a-e4ac5e80af8a.json	5/3/23, 3:46 PM	🗑
6454b1b845e8e857cf46e834	registry.apps.ocphub.physics-faas.eu/custom/ehealth:61	Infer_ehealth_b57e367a-16c6-4b96-a578-435e14076569.json	5/5/23, 10:35 AM	🗑
645bc03845e8e857cf46e837	registry.apps.ocphub.physics-faas.eu/custom/ehealth:63	Infer_ehealth_b57e367a-16c6-4b96-a578-435e14076569.json	5/10/23, 7:03 PM	🗑
646c60ad45e8e857cf46e83c	registry.apps.ocphub.physics-faas.eu/custom/ehealth:64	Infer_ehealth_6b48075b-899c-472e-a0ac-20abdbb9b69b.json	5/23/23, 9:43 AM	🗑

Phenotype 5667bea3640be97f ✓

Figure 11: Admin Panel showing the flows exposed by Node-RED in the eHealth local workflow.



Build #358 (23 Μαΐ 2022, 9:09:45 π.μ.)

Started 2 days 0 hr ago
Took 5 min 18 sec

Το ξεκίνησε ο χρήστης Aristodemos Pnevmatikakis

This run spent:

- 6.7 sec waiting;
- 5 min 18 sec build duration;
- 5 min 25 sec total from scheduled to completion.

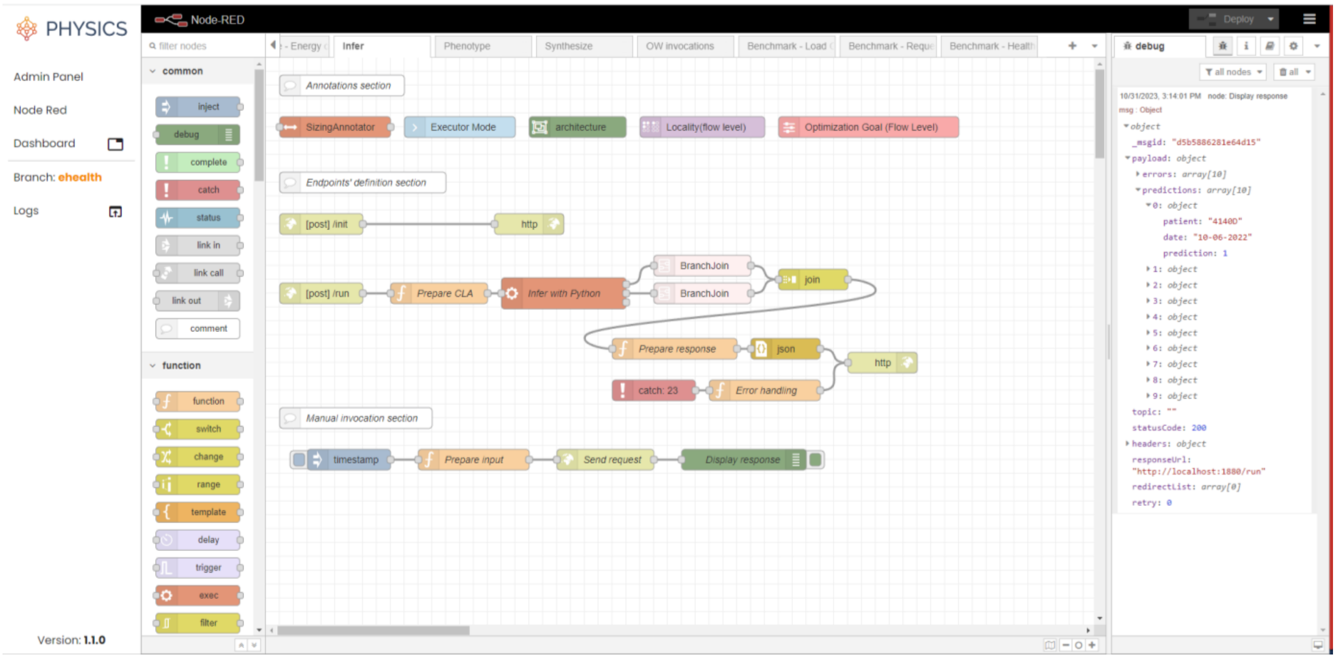
Revision: 10e71ee4a4a5ade39454e23afc4df8009179f92b
Repository: https://repo.apps.ocphub.physics-faas.eu/PHYSICS/test.git

- refs/remotes/origin/master

The following steps that have been detected may have insecure interpolation of sensitive variables (click here for an explanation):

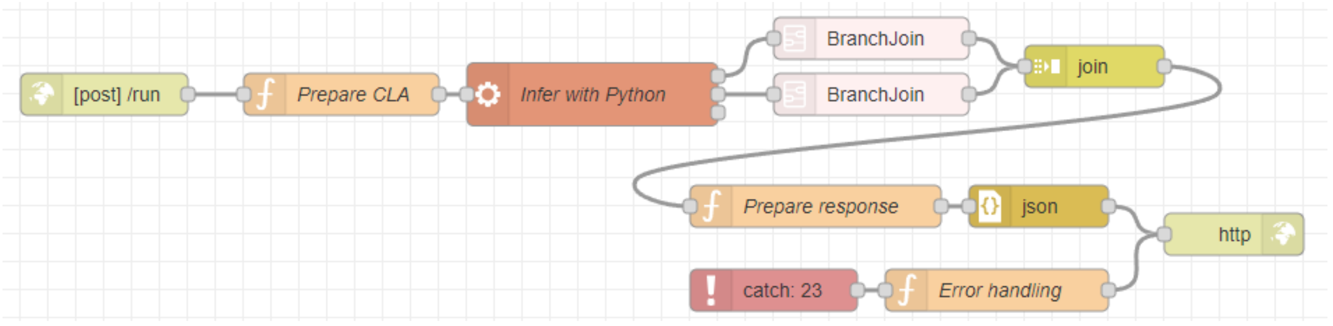
- `sh: [JENKINS_PASSWORD]`
- `sh: [JENKINS_PASSWORD]`
- `sh: [MINIO_KEY, MINIO_SECRET, JENKINS_PASSWORD]`
- `sh: [JENKINS_PASSWORD]`

Figure 12: Jenkins build job initiated by the PHYSICS design environment (Admin panel) for the inference flow.



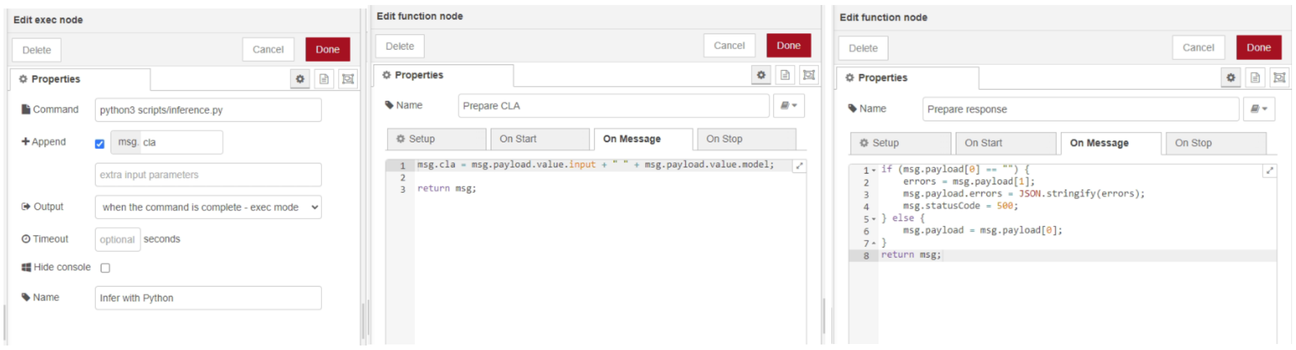
```
10/31/2023, 3:14:01 PM   node: Display response
msg : Object
▾object
  _msgid: "d5b5886281e64d15"
 ▾payload: object
   ▸errors: array[10]
   ▾predictions: array[10]
     ▾0: object
        patient: "4140D"
        date: "10-06-2022"
        prediction: 1
     ▸1: object
     ▸2: object
     ▸3: object
     ▸4: object
     ▸5: object
     ▸6: object
     ▸7: object
     ▸8: object
     ▸9: object
  topic: ""
  statusCode: 200
 ▸headers: object
  responseUrl:
  "http://localhost:1880/run"
  redirectList: array[0]
  retry: 0
```

(a)



(b)

Figure 13: Node-RED interface depicting in inference flow in the eHealth local implementation (a) and zoom in the main flow (b).



Command: python3 scripts/inference.py
Append: ✓ msg. cla
extra input parameters
Output: when the command is complete - exec mode
Timeout: optional seconds
Hide console: ☐
Name: Infer with Python

Name: Prepare CLA

```
msg.cla = msg.payload.value.input + " " + msg.payload.value.model;

return msg;
```

Name: Prepare response

```
if (msg.payload[0] == "") {
    errors = msg.payload[1];
    msg.payload.errors = JSON.stringify(errors);
    msg.statusCode = 500;
} else {
    msg.payload = msg.payload[0];
}
return msg;
```

Figure 14: 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 flows, as well as the OW inference flow are considered in the experimentation section.

4.2.2 Experimentation on the eHealth PHYSICS flows

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

The inference flow expects two input parameters: the input vectors to infer upon (given in `msg.payload.value.input`), and the name of the model to be used (given in `msg.payload.value.model`) for inference in the `inference.py` Python script. The JSON string to be passed to `msg.payload.value.input` for the inference is as follows:

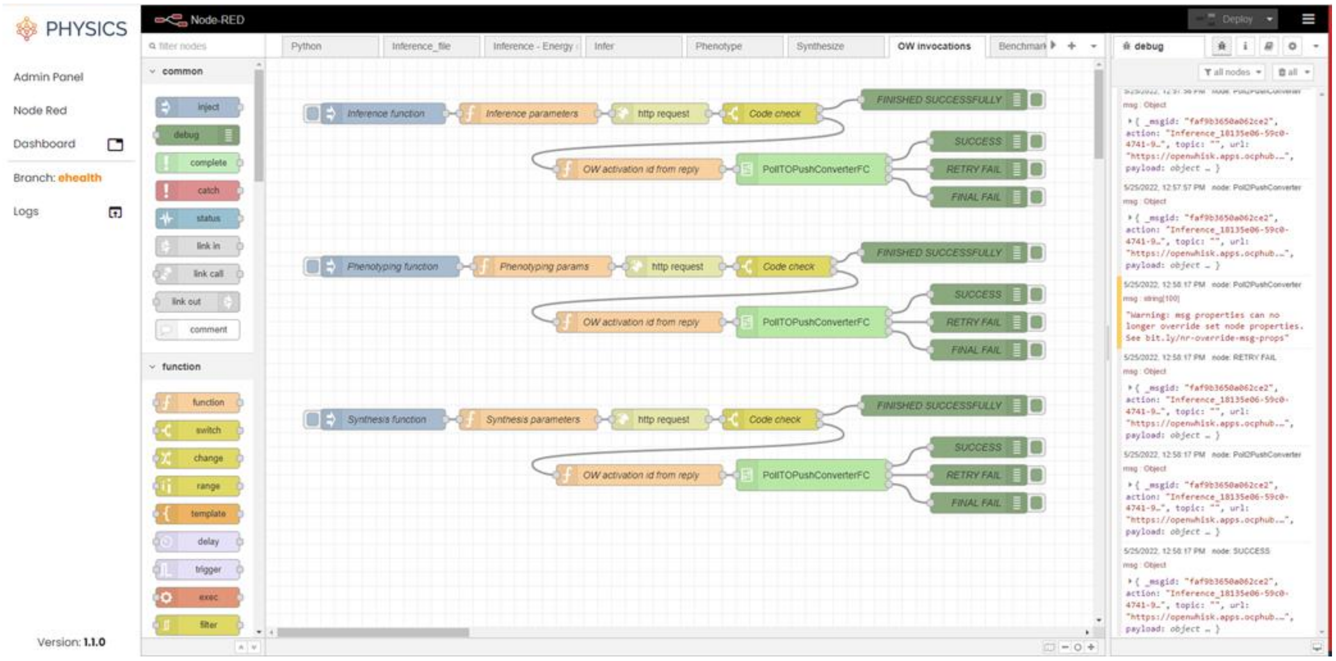
```
[
  {
    "patient":"4140D",
    "date":"10-06-2022",
    "vector":[-0.1756308,0.3731949,0.2202068,0.8069529,-0.1448357,0.8299854,-0.3774286,
              1.6622097,-0.6561496,-0.1429273,-0.3793805,0.7972746,0.1187435,-0.0117112,0.3693147]
  },
  …
]
```

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 in successful inference as indicated by the inference results at the logs shown in the right column of Figure 13.

The phenotyping flow expects the input vectors to be phenotyped and the name of the joblib file containing all the generative models to compare each vector against. The synthesis flow expects the joblib file with the generative models, an array of the initial phenotype of each of the synthetic patients (the length of the array thus determines the number of synthetic patients), the number of time steps to infer for each synthetic patient and the phenotype transition probability matrix.

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, shown in Figure 15.



```
msg : Object
▸{ _msgid: "faf9b3650a062ce2",
action: "Inference_18135e06-59c0-
4741-9…", topic: "", url:
"https://openwhisk.apps.ocphub…",
payload: object … }
5/25/2022, 12:57:57 PM  node: Poll2PushConverter
msg : Object
▸{ _msgid: "faf9b3650a062ce2",
action: "Inference_18135e06-59c0-
4741-9…", topic: "", url:
"https://openwhisk.apps.ocphub…",
payload: object … }
5/25/2022, 12:58:17 PM  node: Poll2PushConverter
msg : string[100]
"Warning: msg properties can no
longer override set node properties.
See bit.ly/nr-override-msg-props"
5/25/2022, 12:58:17 PM  node: RETRY FAIL
msg : Object
▸{ _msgid: "faf9b3650a062ce2",
action: "Inference_18135e06-59c0-
4741-9…", topic: "", url:
"https://openwhisk.apps.ocphub…",
payload: object … }
5/25/2022, 12:58:17 PM  node: Poll2PushConverter
msg : Object
▸{ _msgid: "faf9b3650a062ce2",
action: "Inference_18135e06-59c0-
4741-9…", topic: "", url:
"https://openwhisk.apps.ocphub…",
payload: object … }
5/25/2022, 12:58:17 PM  node: SUCCESS
msg : Object
▸{ _msgid: "faf9b3650a062ce2",
action: "Inference_18135e06-59c0-
4741-9…", topic: "", url:
"https://openwhisk.apps.ocphub…",
payload: object … }
```

Figure 15: The OpenWhisk experimentation flow for the deployed flows of the three cases.

The flow is designed for manual-only invocation via any of the three inject nodes "Inference function", "Phenotyping function", or "Synthesis function", where the respective 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 15, where the final success is indicated.

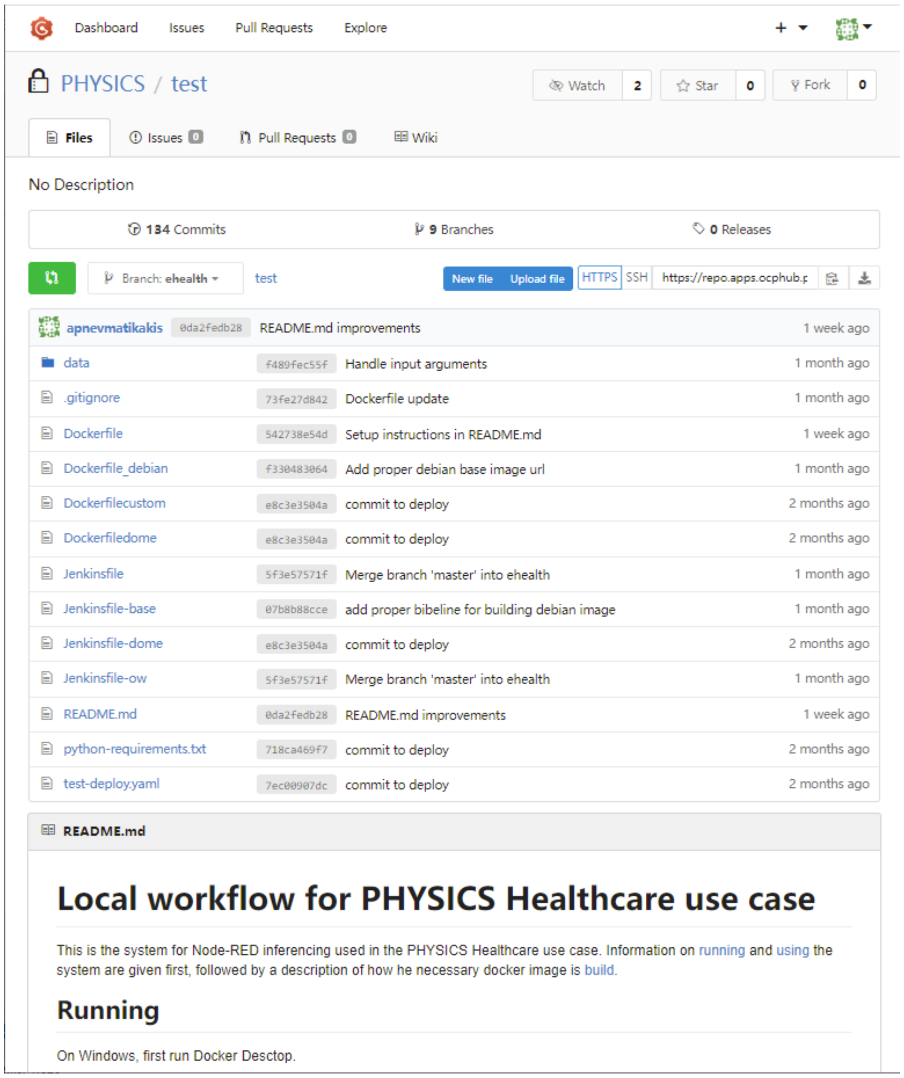
4.2.3 eHealth use-case local design environment instructions

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 16.



No Description

134 Commits | 9 Branches | 0 Releases

Branch: ehealth | test | New file | Upload file | HTTPS | SSH | https://repo.apps.ocphub.p

apnevmatikakis	0da2fedb28	README.md improvements	1 week ago
data	f489fec55f	Handle input arguments	1 month ago
.gitignore	73fe27d842	Dockerfile update	1 month ago
Dockerfile	542738e54d	Setup instructions in README.md	1 week ago
Dockerfile_debian	f330483064	Add proper debian base image url	1 month ago
Dockerfilecustom	e8c3e3504a	commit to deploy	2 months ago
Dockerfiledome	e8c3e3504a	commit to deploy	2 months ago
Jenkinsfile	5f3e57571f	Merge branch 'master' into ehealth	1 month ago
Jenkinsfile-base	07b8b88cce	add proper bibeline for building debian image	1 month ago
Jenkinsfile-dome	e8c3e3504a	commit to deploy	2 months ago
Jenkinsfile-ow	5f3e57571f	Merge branch 'master' into ehealth	1 month ago
README.md	0da2fedb28	README.md improvements	1 week ago
python-requirements.txt	718ca469f7	commit to deploy	2 months ago
test-deploy.yaml	7ec00907dc	commit to deploy	2 months ago

README.md

Local workflow for PHYSICS Healthcare use case

This is the system for Node-RED inferencing used in the PHYSICS Healthcare use case. Information on running and using the system are given first, followed by a description of how he necessary docker image is build.

Running

On Windows, first run Docker Desctop.

Figure 16: Screenshot of the Gogs eHealth repository.

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 where the created flows are 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 set up 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

After having described the eHealth use case flows and the implementation of services from them, we now look back at the functional specifications that were defined in section 3.2.2. For each of the Functional Specifications, we discuss how the specifications are fulfilled. The fulfillment of the functional specifications introduced in Table 2 is discussed in Table 5.

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

Code	Functional Specification Fulfilment
FRS-UC2-01	Fulfilled. Three services have been generated from the respective flows, and URLs are provided for their invocation. Any 3rd party system can utilize these URLs.
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, phenotyping and synthesis flows.

FRS-UC2-04	Fulfilled, since a Python script is provided for each of the three cases of interest (inference, phenotyping and synthesis). Specifically for inference, two scripts are provided, one just for inference and another for both inference and testing, to be used in the case the correct inference results are known.
FRS-UC2-05	Fulfilled: Each service requires its own set of parameters to be POSTed. These parameters are passed to the underlying Python scripts.

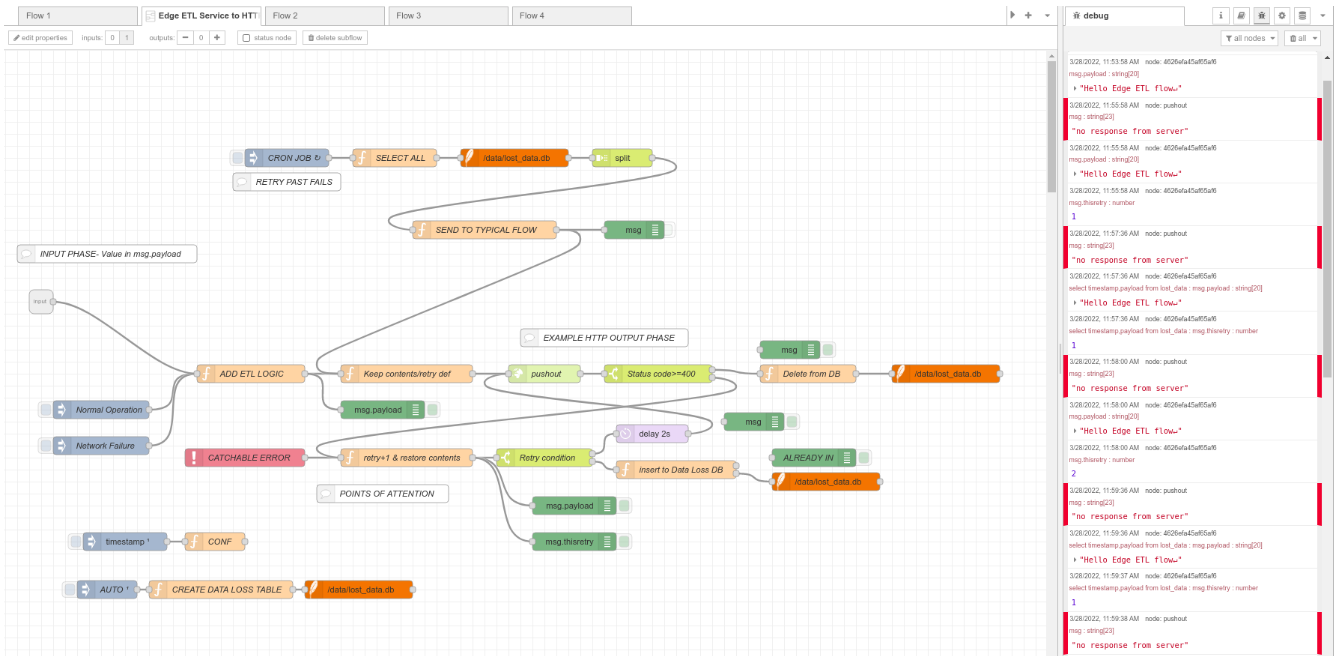
The eHealth use case has successfully fulfilled the respective functional requirements. Several components of the PHYSICS platform are utilized to exploit the FaaS benefits.

4.3 Use Case "Smart Agriculture"

4.3.1 Data collection pipeline

4.3.1.1 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 17. 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, see (Kousiouris, 2023) for more details).



```
3/28/2022, 11:53:58 AM   node: 4626efa45af65af6
msg.payload : string[20]
 "Hello Edge ETL flow↵"
3/28/2022, 11:55:58 AM   node: pushout
msg : string[23]
 "no response from server"
3/28/2022, 11:55:58 AM   node: 4626efa45af65af6
msg.payload : string[20]
 "Hello Edge ETL flow↵"
3/28/2022, 11:55:58 AM   node: 4626efa45af65af6
msg.thisretry : number
 1
3/28/2022, 11:57:36 AM   node: pushout
msg : string[23]
 "no response from server"
3/28/2022, 11:57:36 AM   node: 4626efa45af65af6
select timestamp,payload from lost_data : msg.payload : string[20]
 "Hello Edge ETL flow↵"
3/28/2022, 11:57:36 AM   node: 4626efa45af65af6
select timestamp,payload from lost_data : msg.thisretry : number
 1
3/28/2022, 11:58:00 AM   node: pushout
msg : string[23]
 "no response from server"
3/28/2022, 11:58:00 AM   node: 4626efa45af65af6
msg.payload : string[20]
 "Hello Edge ETL flow↵"
3/28/2022, 11:58:00 AM   node: 4626efa45af65af6
msg.thisretry : number
 2
3/28/2022, 11:59:36 AM   node: pushout
msg : string[23]
 "no response from server"
3/28/2022, 11:59:36 AM   node: 4626efa45af65af6
select timestamp,payload from lost_data : msg.payload : string[20]
 "Hello Edge ETL flow↵"
3/28/2022, 11:59:37 AM   node: 4626efa45af65af6
select timestamp,payload from lost_data : msg.thisretry : number
 1
3/28/2022, 11:59:38 AM   node: pushout
msg : string[23]
 "no response from server"
```

Figure 17: Implementation Flow for the Edge ETL Pattern.

Besides 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 18).

4.3.1.2 <u>Adaptation of legacy codes</u>

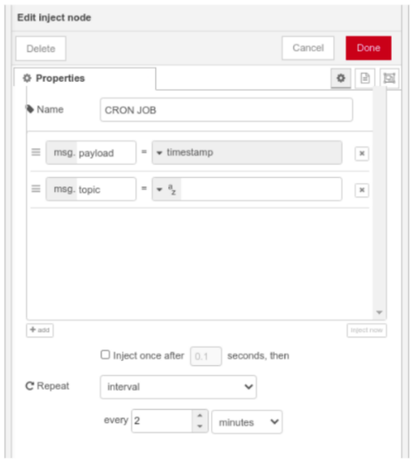
According to the greenhouse supervisor specifications, the data collection logic must be adapted. As today Cybeletech application enables to deal with two types of supervisors.

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.

a)Frequency of pipeline triggering (node CRON JOB)



Edit inject node

Properties

Name: CRON JOB

msg. payload = timestamp

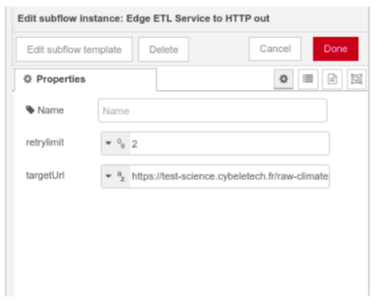
msg. topic =

☐ Inject once after 0.1 seconds, then

Repeat: interval

every 2 minutes

b) Number of retry in case of failure (node input)



Edit subflow instance: Edge ETL Service to HTTP out

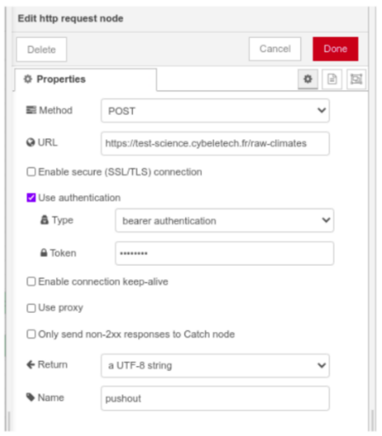
Properties

Name: Name

retrylimit: 2

targetUrl: https://test-science.cybeletech.fr/raw-climate

c) API authentification (node pushout)



Edit http request node

Properties

Method: POST

URL: https://test-science.cybeletech.fr/raw-climates

☐ Enable secure (SSL/TLS) connection

☑ Use authentication

Type: bearer authentication

Token: •••••••••

☐ Enable connection keep-alive

☐ Use proxy

☐ Only send non-2xx responses to Catch node

Return: a UTF-8 string

Name: pushout

Figure 18: Settings of Node-RED ETL flow for data collection pipeline.

The other provides 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.

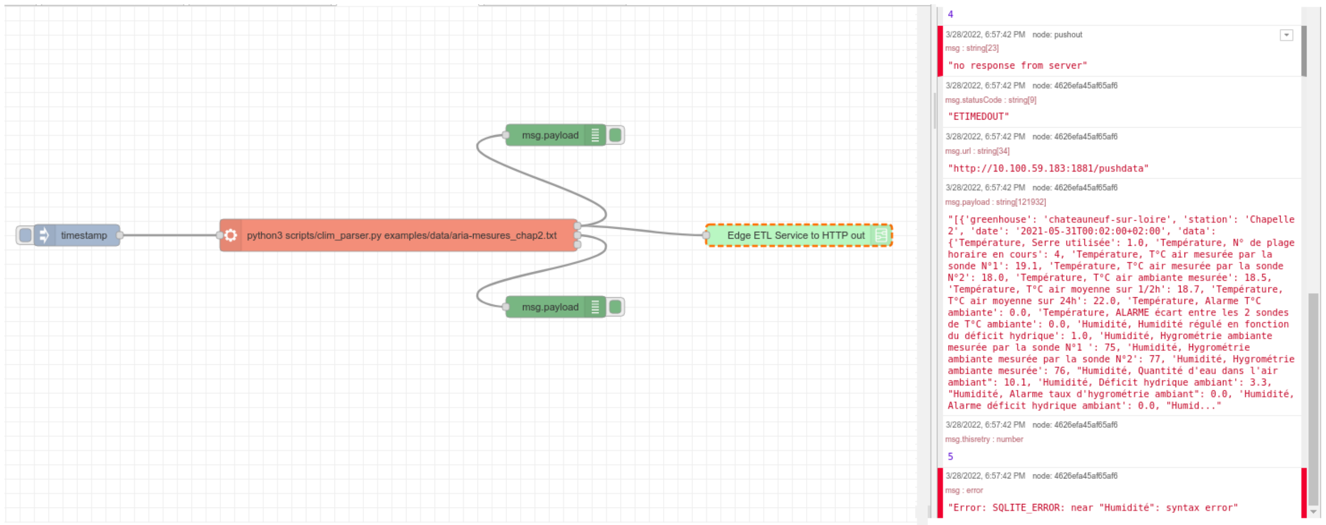
Moreover, an API for data transfer from the supervisor to Cybeletech server has been developed so that the *http node* can be used.

Once this adaptation is implemented, the Python script can be passed as argument to the generic Edge ETL Pattern (see Figure 19).

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.

4.3.1.3 Pipeline testing using the design environment

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 produced by the supervisor of type 1 have been uploaded in the /data repository of the test project. This repository is shared with the docker container hosting the Design Environment and allows the use of scripts and data in the Node-RED flows.


timestamp
python3 scripts/clim_parser.py examples/data/aria-mesures_chap2.txt
msg.payload
msg.payload
Edge ETL Service to HTTP out

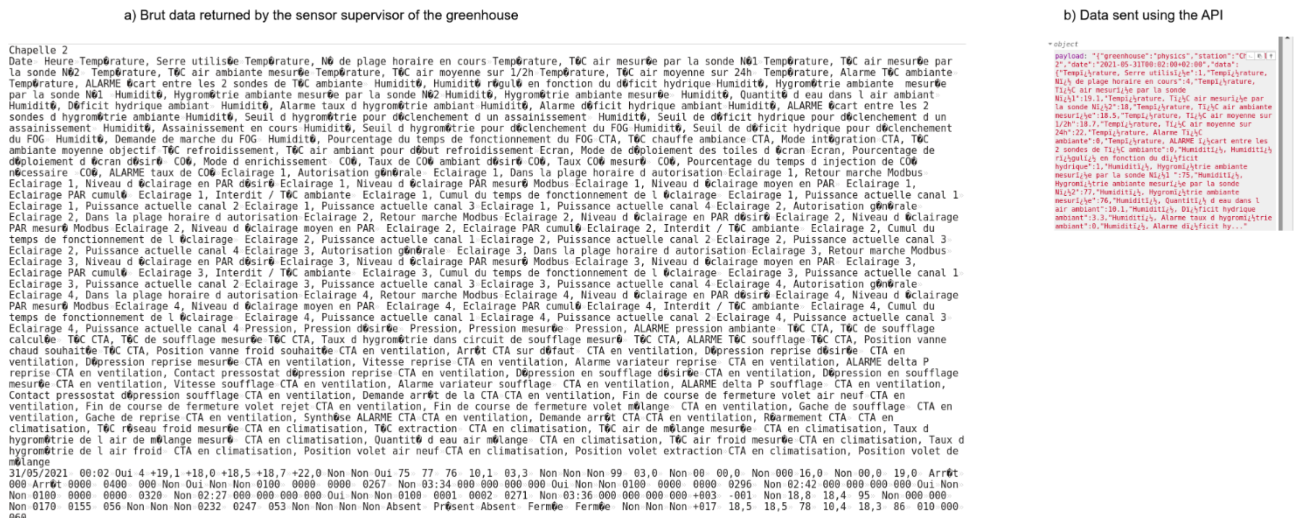
```
4
3/28/2022, 6:57:42 PM   node: pushout
msg : string[23]
"no response from server"
3/28/2022, 6:57:42 PM   node: 4626efa45af65af6
msg.statusCode : string[9]
"ETIMEDOUT"
3/28/2022, 6:57:42 PM   node: 4626efa45af65af6
msg.url : string[34]
"http://10.100.59.183:1881/pushdata"
3/28/2022, 6:57:42 PM   node: 4626efa45af65af6
msg.payload : string[121932]
"[{'greenhouse': 'chateauneuf-sur-loire', 'station': 'Chapelle
2', 'date': '2021-05-31T00:02:00+02:00', 'data':
{'Température, Serre utilisée': 1.0, 'Température, N° de plage
horaire en cours': 4, 'Température, T°C air mesurée par la
sonde N°1': 19.1, 'Température, T°C air mesurée par la sonde
N°2': 18.0, 'Température, T°C air ambiante mesurée': 18.5,
'Température, T°C air moyenne sur 1/2h': 18.7, 'Température,
T°C air moyenne sur 24h': 22.0, 'Température, Alarme T°C
ambiante': 0.0, 'Température, ALARME écart entre les 2 sondes
de T°C ambiante': 0.0, 'Humidité, Humidité régulé en fonction
du déficit hydrique': 1.0, 'Humidité, Hygrométrie ambiante
mesurée par la sonde N°1 ': 75, 'Humidité, Hygrométrie
ambiante mesurée par la sonde N°2': 77, 'Humidité, Hygrométrie
ambiante mesurée': 76, "Humidité, Quantité d'eau dans l'air
ambiant": 10.1, 'Humidité, Déficit hydrique ambiant': 3.3,
"Humidité, Alarme taux d'hygrométrie ambiant": 0.0, 'Humidité,
Alarme déficit hydrique ambiant': 0.0, "Humid..."
3/28/2022, 6:57:42 PM   node: 4626efa45af65af6
msg.thisretry : number
5
3/28/2022, 6:57:42 PM   node: 4626efa45af65af6
msg : error
"Error: SQLITE_ERROR: near "Humidité": syntax error"
```

Figure 19: ETL flow testing.

Three scenarios have been tested:

1. In the first scenario it was assumed that no connection failure occurs during the data collection procedure. It validates 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 validates 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 validates 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 20. 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.

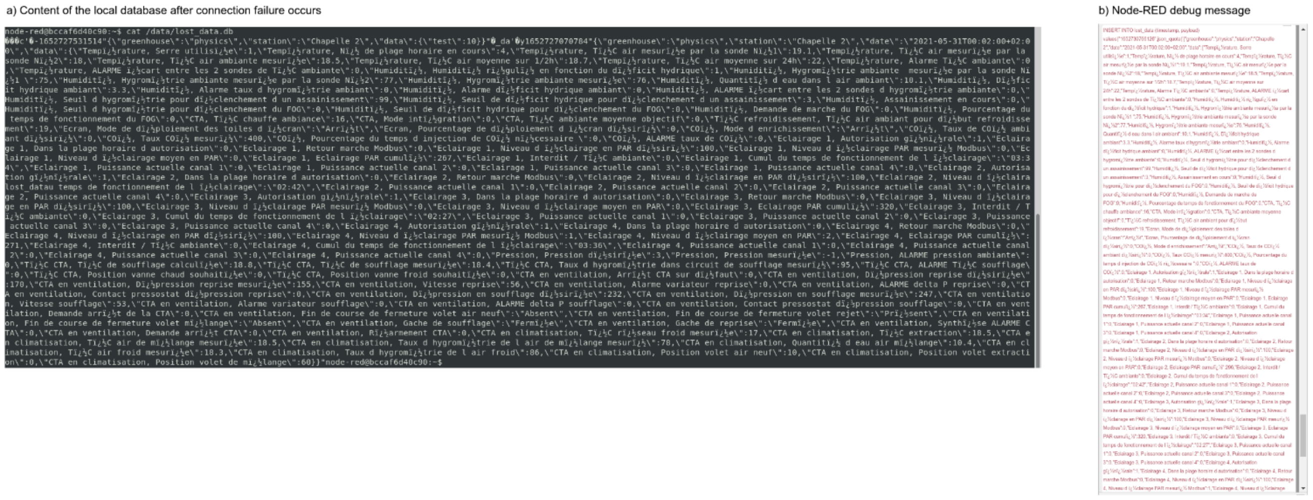

a) Brut data returned by the sensor supervisor of the greenhouse
b) Data sent using the API

```
Chapelle 2
Date  Heure Temp�rature, Serre utilis�e Temp�rature, N� de plage horaire en cours Temp�rature, T�C air mesur�e par la sonde N�1 Temp�rature, T�C air mesur�e par
la sonde N�2  Temp�rature, T�C air ambiante mesur�e Temp�rature, T�C air moyenne sur 1/2h Temp�rature, T�C air moyenne sur 24h  Temp�rature, Alarme T�C ambiante
Temp�rature, ALARME �cart entre les 2 sondes de T�C ambiante  Humidit�, Humidit� r�gul� en fonction du d�ficit hydrique Humidit�, Hygrom�trie ambiante  mesur�e
par la sonde N�1  Humidit�, Hygrom�trie ambiante mesur�e par la sonde N�2 Humidit�, Hygrom�trie ambiante mesur�e  Humidit�, Quantit� d eau dans l air ambiant
Humidit�, D�ficit hydrique ambiant  Humidit�, Alarme taux d hygrom�trie ambiant Humidit�, Alarme d�ficit hydrique ambiant Humidit�, ALARME �cart entre les 2
sondes d hygrom�trie ambiante Humidit�, Seuil d hygrom�trie pour d�clenchement d un assainissement  Humidit�, Seuil de d�ficit hydrique pour d�clenchement d un
assainissement  Humidit�, Assainissement en cours Humidit�, Seuil d hygrom�trie pour d�clenchement du FOG Humidit�, Seuil de d�ficit hydrique pour d�clenchement
du FOG  Humidit�, Demande de marche du FOG  Humidit�, Pourcentage du temps de fonctionnement du FOG CTA, T�C chauffe ambiance CTA, Mode int�gration CTA, T�C
ambiante moyenne objectif T�C refroidissement, T�C air ambiant pour d�but refroidissement Ecran, Mode de d�ploiement des toiles d �cran Ecran, Pourcentage de
d�ploiement d �cran d�sir�  CO�, Mode d enrichissement  CO�, Taux de CO� ambiant d�sir� CO�, Taux CO� mesur�  CO�, Pourcentage du temps d injection de CO�
n�cessaire  CO�, ALARME taux de CO� Eclairage 1, Autorisation g�n�rale  Eclairage 1, Dans la plage horaire d autorisation Eclairage 1, Retour marche Modbus
Eclairage 1, Niveau d �clairage en PAR d�sir� Eclairage 1, Niveau d �clairage PAR mesur� Modbus Eclairage 1, Niveau d �clairage moyen en PAR  Eclairage 1,
Eclairage PAR cumul�  Eclairage 1, Interdit / T�C ambiante  Eclairage 1, Cumul du temps de fonctionnement de l �clairage  Eclairage 1, Puissance actuelle canal 1
Eclairage 1, Puissance actuelle canal 2 Eclairage 1, Puissance actuelle canal 3 Eclairage 1, Puissance actuelle canal 4 Eclairage 2, Autorisation g�n�rale
Eclairage 2, Dans la plage horaire d autorisation Eclairage 2, Retour marche Modbus Eclairage 2, Niveau d �clairage en PAR d�sir� Eclairage 2, Niveau d �clairage
PAR mesur� Modbus Eclairage 2, Niveau d �clairage moyen en PAR  Eclairage 2, Eclairage PAR cumul� Eclairage 2, Interdit / T�C ambiante  Eclairage 2, Cumul du
temps de fonctionnement de l �clairage  Eclairage 2, Puissance actuelle canal 1 Eclairage 2, Puissance actuelle canal 2 Eclairage 2, Puissance actuelle canal 3
Eclairage 2, Puissance actuelle canal 4 Eclairage 3, Autorisation g�n�rale  Eclairage 3, Dans la plage horaire d autorisation Eclairage 3, Retour marche Modbus
Eclairage 3, Niveau d �clairage en PAR d�sir� Eclairage 3, Niveau d �clairage PAR mesur� Modbus Eclairage 3, Niveau d �clairage moyen en PAR  Eclairage 3,
Eclairage PAR cumul�  Eclairage 3, Interdit / T�C ambiante  Eclairage 3, Cumul du temps de fonctionnement de l �clairage  Eclairage 3, Puissance actuelle canal 1
Eclairage 3, Puissance actuelle canal 2 Eclairage 3, Puissance actuelle canal 3 Eclairage 3, Puissance actuelle canal 4 Eclairage 4, Autorisation g�n�rale
Eclairage 4, Dans la plage horaire d autorisation Eclairage 4, Retour marche Modbus Eclairage 4, Niveau d �clairage en PAR d�sir� Eclairage 4, Niveau d �clairage
PAR mesur� Modbus Eclairage 4, Niveau d �clairage moyen en PAR  Eclairage 4, Eclairage PAR cumul� Eclairage 4, Interdit / T�C ambiante  Eclairage 4, Cumul du
temps de fonctionnement de l �clairage  Eclairage 4, Puissance actuelle canal 1 Eclairage 4, Puissance actuelle canal 2 Eclairage 4, Puissance actuelle canal 3
Eclairage 4, Puissance actuelle canal 4 Pression, Pression d�sir�e  Pression, Pression mesur�e  Pression, ALARME pression ambiante  T�C CTA, T�C de soufflage
calcul�e  T�C CTA, T�C de soufflage mesur�e T�C CTA, Taux d hygrom�trie dans circuit de soufflage mesur�  T�C CTA, ALARME T�C soufflage T�C CTA, Position vanne
chaud souhait�e T�C CTA, Position vanne froid souhait�e CTA en ventilation, Arr�t CTA sur d�faut  CTA en ventilation, D�pression reprise d�sir�e  CTA en
ventilation, D�pression reprise mesur�e CTA en ventilation, Vitesse reprise CTA en ventilation, Alarme variateur reprise  CTA en ventilation, ALARME delta P
reprise CTA en ventilation, Contact pressostat d�pression reprise CTA en ventilation, D�pression en soufflage d�sir�e CTA en ventilation, D�pression en soufflage
mesur�e CTA en ventilation, Vitesse soufflage CTA en ventilation, Alarme variateur soufflage  CTA en ventilation, ALARME delta P soufflage  CTA en ventilation,
Contact pressostat d�pression soufflage CTA en ventilation, Demande arr�t de la CTA CTA en ventilation, Fin de course de fermeture volet air neuf CTA en
ventilation, Fin de course de fermeture volet rejet CTA en ventilation, Fin de course de fermeture volet m�lange  CTA en ventilation, Gache de soufflage  CTA en
ventilation, Gache de reprise CTA en ventilation, Synth�se ALARME CTA CTA en ventilation, Demande arr�t CTA CTA en ventilation, R�armement CTA  CTA en
climatisation, T�C r�seau froid mesur�e CTA en climatisation, T�C extraction  CTA en climatisation, T�C air de m�lange mesur�e  CTA en climatisation, Taux d
hygrom�trie de l air de m�lange mesur�  CTA en climatisation, Quantit� d eau air m�lange  CTA en climatisation, T�C air froid mesur�e CTA en climatisation, Taux d
hygrom�trie de l air froid  CTA en climatisation, Position volet air neuf CTA en climatisation, Position volet extraction CTA en climatisation, Position volet de
m�lange
31/05/2021  00:02 Oui 4 +19,1 +18,0 +18,5 +18,7 +22,0 Non Non Oui 75  77  76  10,1  03,3  Non Non Non 99  03,0  Non 00  00,0  Non 000 16,0  Non 00,0  19,0  Arr�t
000 Arr�t 0000  0400  000 Non Oui Non Non 0100  0000  0000  0267  Non 03:34 000 000 000 000 Oui Non Non 0100  0000  0000  0296  Non 02:42 000 000 000 000 Oui Non
Non 0100  0000  0000  0320  Non 02:27 000 000 000 000 Oui Non Non 0100  0001  0002  0271  Non 03:36 000 000 000 000 +003  -001  Non 18,8  18,4  95  Non 000 000
Non 0170  0155  056 Non Non Non 0232  0247  053 Non Non Non Non Absent  Pr�sent Absent  Ferm�e  Ferm�e  Non Non Non +017  18,5  18,5  78  10,4  18,3  86  010 000
060
```

```
object
payload: "{"greenhouse":"physics","station":"Ch
2","date":"2021-05-31T00:02:00+02:00","data":
{"Tempï¿½rature, Serre utilisï¿½e":1,"Tempï¿½rature,
Nï¿½ de plage horaire en cours":4,"Tempï¿½rature,
Tï¿½C air mesurï¿½e par la sonde
Nï¿½1":19.1,"Tempï¿½rature, Tï¿½C air mesurï¿½e par
la sonde Nï¿½2":18,"Tempï¿½rature, Tï¿½C air ambiante
mesurï¿½e":18.5,"Tempï¿½rature, Tï¿½C air moyenne sur
1/2h":18.7,"Tempï¿½rature, Tï¿½C air moyenne sur
24h":22,"Tempï¿½rature, Alarme Tï¿½C
ambiante":0,"Tempï¿½rature, ALARME ï¿½cart entre les
2 sondes de Tï¿½C ambiante":0,"Humiditï¿½, Humiditï¿½
rï¿½gulï¿½ en fonction du dï¿½ficit
hydrique":1,"Humiditï¿½, Hygromï¿½trie ambiante
mesurï¿½e par la sonde Nï¿½1 ":75,"Humiditï¿½,
Hygromï¿½trie ambiante mesurï¿½e par la sonde
Nï¿½2":77,"Humiditï¿½, Hygromï¿½trie ambiante
mesurï¿½e":76,"Humiditï¿½, Quantitï¿½ d eau dans l
air ambiant":10.1,"Humiditï¿½, Dï¿½ficit hydrique
ambiant":3.3,"Humiditï¿½, Alarme taux d hygromï¿½trie
ambiant":0,"Humiditï¿½, Alarme dï¿½ficit hy..."
```

Figure 20: 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 21. 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.


a) Content of the local database after connection failure occurs
b) Node-RED debug message

```
node-red@bccaf6d40c90:~$ cat /data/lost_data.db
���c'�-1652727531514"{\"greenhouse\":\"physics\",\"station\":\"Chapelle 2\",\"data\":{\"test\":10}}"�_da'�y1652727070784"{\"greenhouse\":\"physics\",\"station\":\"Chapelle 2\",\"date\":\"2021-05-31T00:02:00+02:0
0\",\"data\":{\"Tempï¿½rature, Serre utilisï¿½e\":1,\"Tempï¿½rature, Nï¿½ de plage horaire en cours\":4,\"Tempï¿½rature, Tï¿½C air mesurï¿½e par la sonde Nï¿½1\":19.1,\"Tempï¿½rature, Tï¿½C air mesurï¿½e par la
sonde Nï¿½2\":18,\"Tempï¿½rature, Tï¿½C air ambiante mesurï¿½e\":18.5,\"Tempï¿½rature, Tï¿½C air moyenne sur 1/2h\":18.7,\"Tempï¿½rature, Tï¿½C air moyenne sur 24h\":22,\"Tempï¿½rature, Alarme Tï¿½C ambiante\":0
,\"Tempï¿½rature, ALARME ï¿½cart entre les 2 sondes de Tï¿½C ambiante\":0,\"Humiditï¿½, Humiditï¿½ rï¿½gulï¿½ en fonction du dï¿½ficit hydrique\":1,\"Humiditï¿½, Hygromï¿½trie ambiante  mesurï¿½e par la sonde Nï
¿½1 \":75,\"Humiditï¿½, Hygromï¿½trie ambiante mesurï¿½e par la sonde Nï¿½2\":77,\"Humiditï¿½, Hygromï¿½trie ambiante mesurï¿½e\":76,\"Humiditï¿½, Quantitï¿½ d eau dans l air ambiant\":10.1,\"Humiditï¿½, Dï¿½fic
it hydrique ambiant\":3.3,\"Humiditï¿½, Alarme taux d hygromï¿½trie ambiant\":0,\"Humiditï¿½, Alarme dï¿½ficit hydrique ambiant\":0,\"Humiditï¿½, ALARME ï¿½cart entre les 2 sondes d hygromï¿½trie ambiante\":0,\"
Humiditï¿½, Seuil d hygromï¿½trie pour dï¿½clenchement d un assainissement\":99,\"Humiditï¿½, Seuil de dï¿½ficit hydrique pour dï¿½clenchement d un assainissement\":3,\"Humiditï¿½, Assainissement en cours\":0,\"
Humiditï¿½, Seuil d hygromï¿½trie pour dï¿½clenchement du FOG\":0,\"Humiditï¿½, Seuil de dï¿½ficit hydrique pour dï¿½clenchement du FOG\":0,\"Humiditï¿½, Demande de marche du FOG\":0,\"Humiditï¿½, Pourcentage du
 temps de fonctionnement du FOG\":0,\"CTA, Tï¿½C chauffe ambiance\":16,\"CTA, Mode intï¿½gration\":0,\"CTA, Tï¿½C ambiante moyenne objectif\":0,\"Tï¿½C refroidissement, Tï¿½C air ambiant pour dï¿½but refroidisse
ment\":19,\"Ecran, Mode de dï¿½ploiement des toiles d ï¿½cran\":\"Arrï¿½t\",\"Ecran, Pourcentage de dï¿½ploiement d ï¿½cran dï¿½sirï¿½\":0,\"COï¿½, Mode d enrichissement\":\"Arrï¿½t\",\"COï¿½, Taux de COï¿½ ambi
ant dï¿½sirï¿½\":0,\"COï¿½, Taux COï¿½ mesurï¿½\":400,\"COï¿½, Pourcentage du temps d injection de COï¿½ nï¿½cessaire \":0,\"COï¿½, ALARME taux de COï¿½\":0,\"Eclairage 1, Autorisation gï¿½nï¿½rale\":1,\"Eclaira
ge 1, Dans la plage horaire d autorisation\":0,\"Eclairage 1, Retour marche Modbus\":0,\"Eclairage 1, Niveau d ï¿½clairage en PAR dï¿½sirï¿½\":100,\"Eclairage 1, Niveau d ï¿½clairage PAR mesurï¿½ Modbus\":0,\"Ec
lairage 1, Niveau d ï¿½clairage moyen en PAR\":0,\"Eclairage 1, Eclairage PAR cumulï¿½\":267,\"Eclairage 1, Interdit / Tï¿½C ambiante\":0,\"Eclairage 1, Cumul du temps de fonctionnement de l ï¿½clairage\":\"03:3
4\",\"Eclairage 1, Puissance actuelle canal 1\":0,\"Eclairage 1, Puissance actuelle canal 2\":0,\"Eclairage 1, Puissance actuelle canal 3\":0,\"Eclairage 1, Puissance actuelle canal 4\":0,\"Eclairage 2, Autorisa
tion gï¿½nï¿½rale\":1,\"Eclairage 2, Dans la plage horaire d autorisation\":0,\"Eclairage 2, Retour marche Modbus\":0,\"Eclairage 2, Niveau d ï¿½clairage en PAR dï¿½sirï¿½\":100,\"Eclairage 2, Niveau d ï¿½claira
lost_datau temps de fonctionnement de l ï¿½clairage\":\"02:42\",\"Eclairage 2, Puissance actuelle canal 1\":0,\"Eclairage 2, Puissance actuelle canal 2\":0,\"Eclairage 2, Puissance actuelle canal 3\":0,\"Eclaira
ge 2, Puissance actuelle canal 4\":0,\"Eclairage 3, Autorisation gï¿½nï¿½rale\":1,\"Eclairage 3, Dans la plage horaire d autorisation\":0,\"Eclairage 3, Retour marche Modbus\":0,\"Eclairage 3, Niveau d ï¿½claira
ge en PAR dï¿½sirï¿½\":100,\"Eclairage 3, Niveau d ï¿½clairage PAR mesurï¿½ Modbus\":0,\"Eclairage 3, Niveau d ï¿½clairage moyen en PAR\":0,\"Eclairage 3, Eclairage PAR cumulï¿½\":320,\"Eclairage 3, Interdit / T
ï¿½C ambiante\":0,\"Eclairage 3, Cumul du temps de fonctionnement de l ï¿½clairage\":\"02:27\",\"Eclairage 3, Puissance actuelle canal 1\":0,\"Eclairage 3, Puissance actuelle canal 2\":0,\"Eclairage 3, Puissance
 actuelle canal 3\":0,\"Eclairage 3, Puissance actuelle canal 4\":0,\"Eclairage 4, Autorisation gï¿½nï¿½rale\":1,\"Eclairage 4, Dans la plage horaire d autorisation\":0,\"Eclairage 4, Retour marche Modbus\":0,\"
Eclairage 4, Niveau d ï¿½clairage en PAR dï¿½sirï¿½\":100,\"Eclairage 4, Niveau d ï¿½clairage PAR mesurï¿½ Modbus\":1,\"Eclairage 4, Niveau d ï¿½clairage moyen en PAR\":2,\"Eclairage 4, Eclairage PAR cumulï¿½\":
271,\"Eclairage 4, Interdit / Tï¿½C ambiante\":0,\"Eclairage 4, Cumul du temps de fonctionnement de l ï¿½clairage\":\"03:36\",\"Eclairage 4, Puissance actuelle canal 1\":0,\"Eclairage 4, Puissance actuelle canal
 2\":0,\"Eclairage 4, Puissance actuelle canal 3\":0,\"Eclairage 4, Puissance actuelle canal 4\":0,\"Pression, Pression dï¿½sirï¿½e\":3,\"Pression, Pression mesurï¿½e\":-1,\"Pression, ALARME pression ambiante\":
0,\"Tï¿½C CTA, Tï¿½C de soufflage calculï¿½e\":18.8,\"Tï¿½C CTA, Tï¿½C de soufflage mesurï¿½e\":18.4,\"Tï¿½C CTA, Taux d hygromï¿½trie dans circuit de soufflage mesurï¿½\":95,\"Tï¿½C CTA, ALARME Tï¿½C soufflage\
":0,\"Tï¿½C CTA, Position vanne chaud souhaitï¿½e\":0,\"Tï¿½C CTA, Position vanne froid souhaitï¿½e\":0,\"CTA en ventilation, Arrï¿½t CTA sur dï¿½faut\":0,\"CTA en ventilation, Dï¿½pression reprise dï¿½sirï¿½e\"
:170,\"CTA en ventilation, Dï¿½pression reprise mesurï¿½e\":155,\"CTA en ventilation, Vitesse reprise\":56,\"CTA en ventilation, Alarme variateur reprise\":0,\"CTA en ventilation, ALARME delta P reprise\":0,\"CT
A en ventilation, Contact pressostat dï¿½pression reprise\":0,\"CTA en ventilation, Dï¿½pression en soufflage dï¿½sirï¿½e\":232,\"CTA en ventilation, Dï¿½pression en soufflage mesurï¿½e\":247,\"CTA en ventilatio
n, Vitesse soufflage\":53,\"CTA en ventilation, Alarme variateur soufflage\":0,\"CTA en ventilation, ALARME delta P soufflage\":0,\"CTA en ventilation, Contact pressostat dï¿½pression soufflage\":0,\"CTA en vent
ilation, Demande arrï¿½t de la CTA\":0,\"CTA en ventilation, Fin de course de fermeture volet air neuf\":\"Absent\",\"CTA en ventilation, Fin de course de fermeture volet rejet\":\"Prï¿½sent\",\"CTA en ventilati
on, Fin de course de fermeture volet mï¿½lange\":\"Absent\",\"CTA en ventilation, Gache de soufflage\":\"Fermï¿½e\",\"CTA en ventilation, Gache de reprise\":\"Fermï¿½e\",\"CTA en ventilation, Synthï¿½se ALARME C
TA\":0,\"CTA en ventilation, Demande arrï¿½t CTA\":0,\"CTA en ventilation, Rï¿½armement CTA\":0,\"CTA en climatisation, Tï¿½C rï¿½seau froid mesurï¿½e\":17,\"CTA en climatisation, Tï¿½C extraction\":18.5,\"CTA e
n climatisation, Tï¿½C air de mï¿½lange mesurï¿½e\":18.5,\"CTA en climatisation, Taux d hygromï¿½trie de l air de mï¿½lange mesurï¿½\":78,\"CTA en climatisation, Quantitï¿½ d eau air mï¿½lange\":10.4,\"CTA en cl
imatisation, Tï¿½C air froid mesurï¿½e\":18.3,\"CTA en climatisation, Taux d hygromï¿½trie de l air froid\":86,\"CTA en climatisation, Position volet air neuf\":10,\"CTA en climatisation, Position volet extracti
on\":0,\"CTA en climatisation, Position volet de mï¿½lange\":60}}"node-red@bccaf6d40c90:~$
```

```
INSERT INTO lost_data (timestamp, payload)
values ("1652730700120",json_quote({"greenhouse":"physics","station":"Chapelle
2","date":"2021-05-31T00:02:00+02:00","data":{"Tempï¿½rature, Serre
utilisï¿½e":1,"Tempï¿½rature, Nï¿½ de plage horaire en cours":4,"Tempï¿½rature, Tï¿½C
air mesurï¿½e par la sonde Nï¿½1":19.1,"Tempï¿½rature, Tï¿½C air mesurï¿½e par la
sonde Nï¿½2":18,"Tempï¿½rature, Tï¿½C air ambiante mesurï¿½e":18.5,"Tempï¿½rature,
Tï¿½C air moyenne sur 1/2h":18.7,"Tempï¿½rature, Tï¿½C air moyenne sur
24h":22,"Tempï¿½rature, Alarme Tï¿½C ambiante":0,"Tempï¿½rature, ALARME ï¿½cart
entre les 2 sondes de Tï¿½C ambiante":0,"Humiditï¿½, Humiditï¿½ rï¿½gulï¿½ en
fonction du dï¿½ficit hydrique":1,"Humiditï¿½, Hygromï¿½trie ambiante mesurï¿½e par la
sonde Nï¿½1 ":75,"Humiditï¿½, Hygromï¿½trie ambiante mesurï¿½e par la sonde
Nï¿½2":77,"Humiditï¿½, Hygromï¿½trie ambiante mesurï¿½e":76,"Humiditï¿½,
Quantitï¿½ d eau dans l air ambiant":10.1,"Humiditï¿½, Dï¿½ficit hydrique
ambiant":3.3,"Humiditï¿½, Alarme taux d hygromï¿½trie ambiant":0,"Humiditï¿½, Alarme
dï¿½ficit hydrique ambiant":0,"Humiditï¿½, ALARME ï¿½cart entre les 2 sondes d
hygromï¿½trie ambiante":0,"Humiditï¿½, Seuil d hygromï¿½trie pour dï¿½clenchement d
un assainissement":99,"Humiditï¿½, Seuil de dï¿½ficit hydrique pour dï¿½clenchement d
un assainissement":3,"Humiditï¿½, Assainissement en cours":0,"Humiditï¿½, Seuil d
hygromï¿½trie pour dï¿½clenchement du FOG":0,"Humiditï¿½, Seuil de dï¿½ficit hydrique
pour dï¿½clenchement du FOG":0,"Humiditï¿½, Demande de marche du
FOG":0,"Humiditï¿½, Pourcentage du temps de fonctionnement du FOG":0,"CTA, Tï¿½C
chauffe ambiance":16,"CTA, Mode intï¿½gration":0,"CTA, Tï¿½C ambiante moyenne
objectif":0,"Tï¿½C refroidissement, Tï¿½C air ambiant pour dï¿½but
refroidissement":19,"Ecran, Mode de dï¿½ploiement des toiles d
ï¿½cran":"Arrï¿½t","Ecran, Pourcentage de dï¿½ploiement d ï¿½cran
dï¿½sirï¿½":0,"COï¿½, Mode d enrichissement":"Arrï¿½t","COï¿½, Taux de COï¿½
ambiant dï¿½sirï¿½":0,"COï¿½, Taux COï¿½ mesurï¿½":400,"COï¿½, Pourcentage du
temps d injection de COï¿½ nï¿½cessaire ":0,"COï¿½, ALARME taux de
COï¿½":0,"Eclairage 1, Autorisation gï¿½nï¿½rale":1,"Eclairage 1, Dans la plage horaire d
autorisation":0,"Eclairage 1, Retour marche Modbus":0,"Eclairage 1, Niveau d ï¿½clairage
en PAR dï¿½sirï¿½":100,"Eclairage 1, Niveau d ï¿½clairage PAR mesurï¿½
Modbus":0,"Eclairage 1, Niveau d ï¿½clairage moyen en PAR":0,"Eclairage 1, Eclairage
PAR cumulï¿½":267,"Eclairage 1, Interdit / Tï¿½C ambiante":0,"Eclairage 1, Cumul du
temps de fonctionnement de l ï¿½clairage":"03:34","Eclairage 1, Puissance actuelle canal
1":0,"Eclairage 1, Puissance actuelle canal 2":0,"Eclairage 1, Puissance actuelle canal
3":0,"Eclairage 1, Puissance actuelle canal 4":0,"Eclairage 2, Autorisation
gï¿½nï¿½rale":1,"Eclairage 2, Dans la plage horaire d autorisation":0,"Eclairage 2, Retour
marche Modbus":0,"Eclairage 2, Niveau d ï¿½clairage en PAR dï¿½sirï¿½":100,"Eclairage
[illegible]
```

Figure 21: 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 is 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.

4.3.2 Simulation pipeline

4.3.2.1 Pipeline definition using Node-Red

The pipeline is implemented as a Node-RED flow as described in Figure 22. This flow allows to build the input required by the python script called in the "Run agronomical simulation" node from the message payload (Figure 23-A). The output of the simulation script is collected using the BranchJoin pattern, which allows to aggregate the results of the simulation together with the logging output used for monitoring in a single message. The node labeled "Prepare response" reads this message and gives the output its final shape (Figure 23-B).

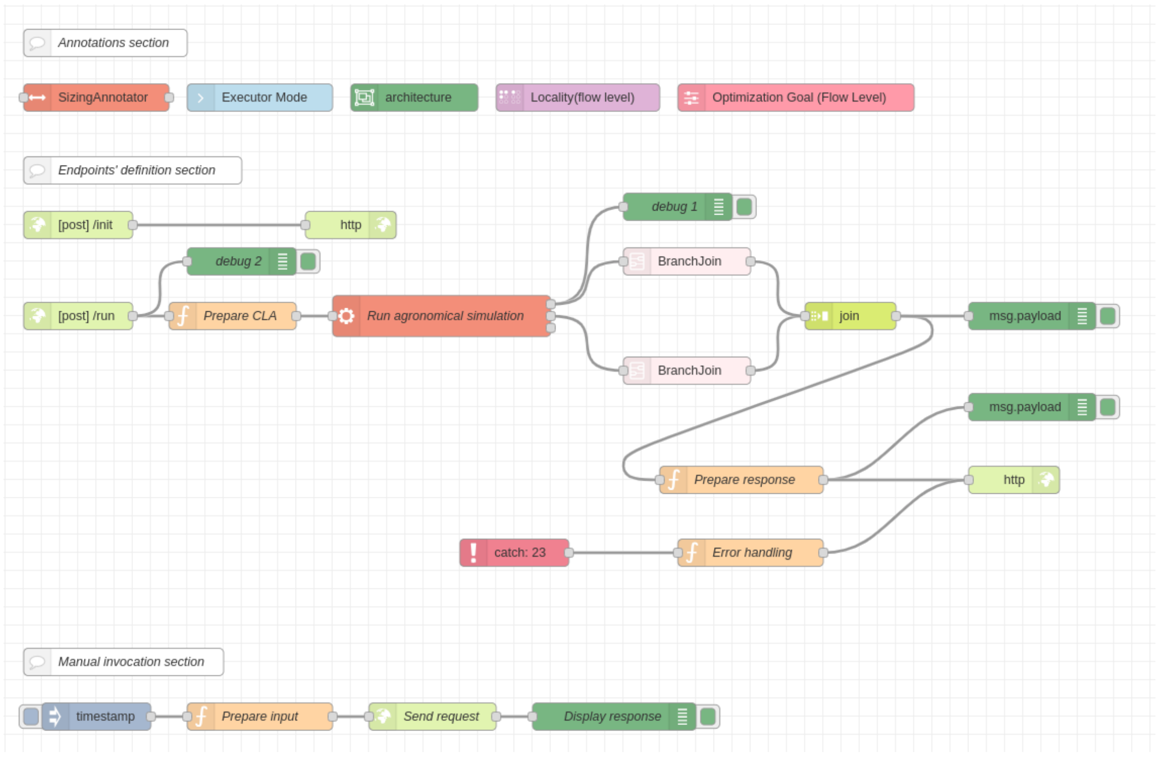
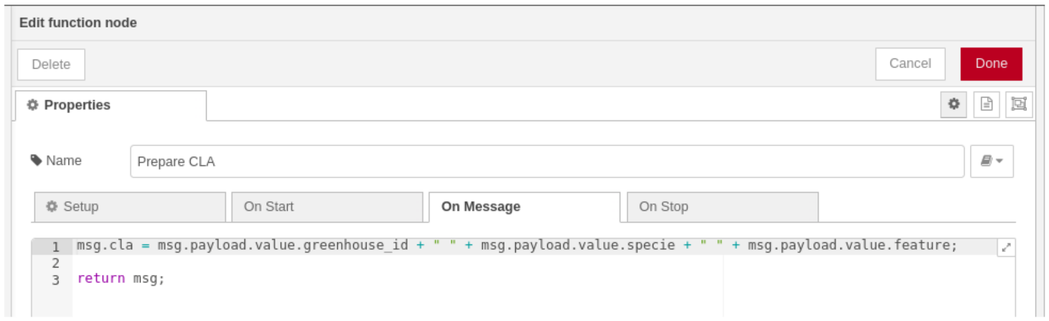


Figure 22: Node-RED flow describing the simulation pipeline.

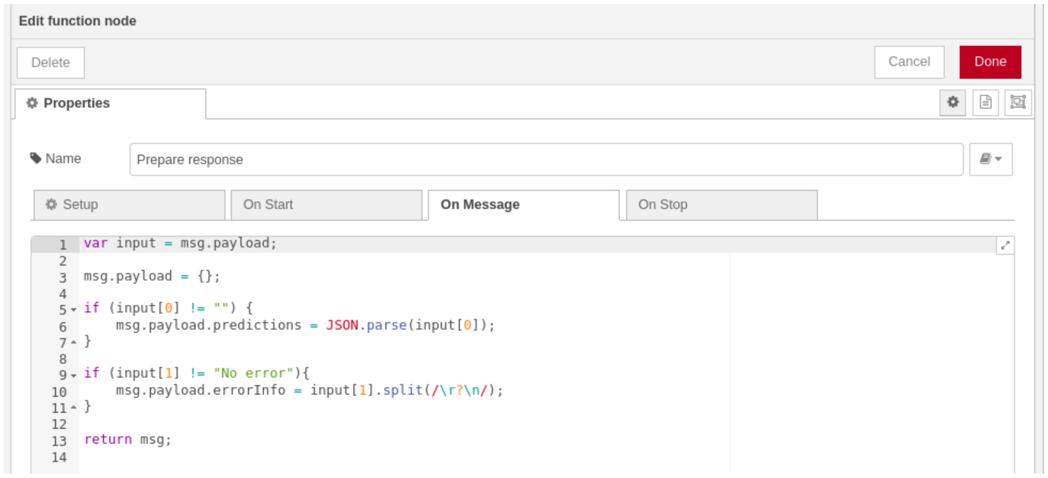
A. Function node for input preparation



Name: Prepare CLA

```
msg.cla = msg.payload.value.greenhouse_id + " " + msg.payload.value.specie + " " + msg.payload.value.feature;

return msg;
```

B. Function node for output preparation



Name: Prepare response

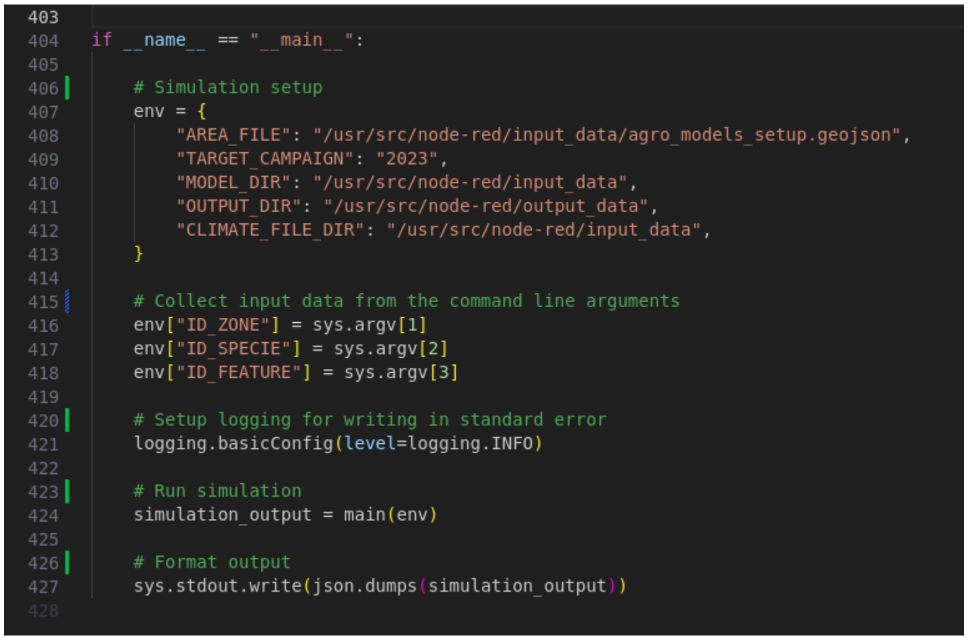
```
var input = msg.payload;

msg.payload = {};

if (input[0] != "") {
    msg.payload.predictions = JSON.parse(input[0]);
}

if (input[1] != "No error"){
    msg.payload.errorInfo = input[1].split(/\r?\n/);
}

return msg;

```

Figure 23: Node-RED function node for specification of I/O of simulation pipeline.

4.3.2.2 Adaptation of legacy codes

The integration of the Python script in the Node-RED flow is straightforward. The main change introduced is the adaptation of the input / output management: the inputs are collected using the *sys* module, which enables to get the command line arguments passed to the python script; the output is formatted as a json string and returned using the standard output channel (Figure 24).

Moreover, we introduced the use of the *logging* module to collect the logs of the script execution. This allows us to write info, warnings and errors in the standard error channel and to collect them at the end of the execution.



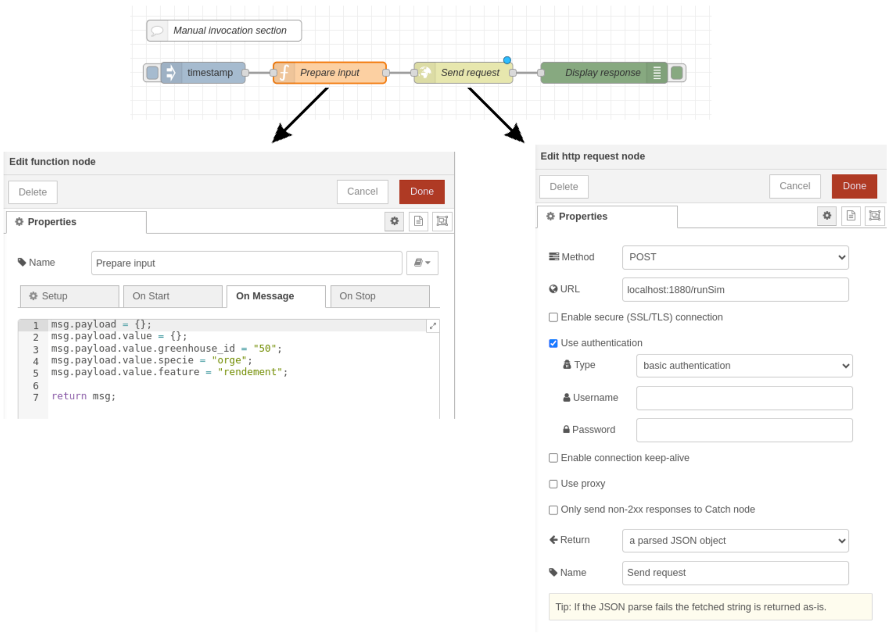
```python

if __name__ == "__main__":

    # Simulation setup
    env = {
        "AREA_FILE": "/usr/src/node-red/input_data/agro_models_setup.geojson",
        "TARGET_CAMPAIGN": "2023",
        "MODEL_DIR": "/usr/src/node-red/input_data",
        "OUTPUT_DIR": "/usr/src/node-red/output_data",
        "CLIMATE_FILE_DIR": "/usr/src/node-red/input_data",
    }

    # Collect input data from the command line arguments
    env["ID_ZONE"] = sys.argv[1]
    env["ID_SPECIE"] = sys.argv[2]
    env["ID_FEATURE"] = sys.argv[3]

    # Setup logging for writing in standard error
    logging.basicConfig(level=logging.INFO)

    # Run simulation
    simulation_output = main(env)

    # Format output
    sys.stdout.write(json.dumps(simulation_output))
```

Figure 24: Adaptation of Python script for simulation pipeline execution.

4.3.2.3 Pipeline deployment and testing

First, the simulation pipeline has been tested locally using the Node-RED panel of the Design Environment. The manual invocation of the flow is done using the http request node with a specific URL. The preparation of the input message is done using the function node (Figure 25).

The different steps of the pipeline can be monitored using the debug nodes, which output messages in the debug window of the Node-RED user interface. In a typical run, the user provides the id of its greenhouse, the species grown and the properties of interest. The simulation script looks for model parameters according to the species and uses them to provide an estimation of the target properties according to environmental conditions in the greenhouse (Figure 26). Here the climate conditions and models' parameters are made available through plain files embedded in the docker image.

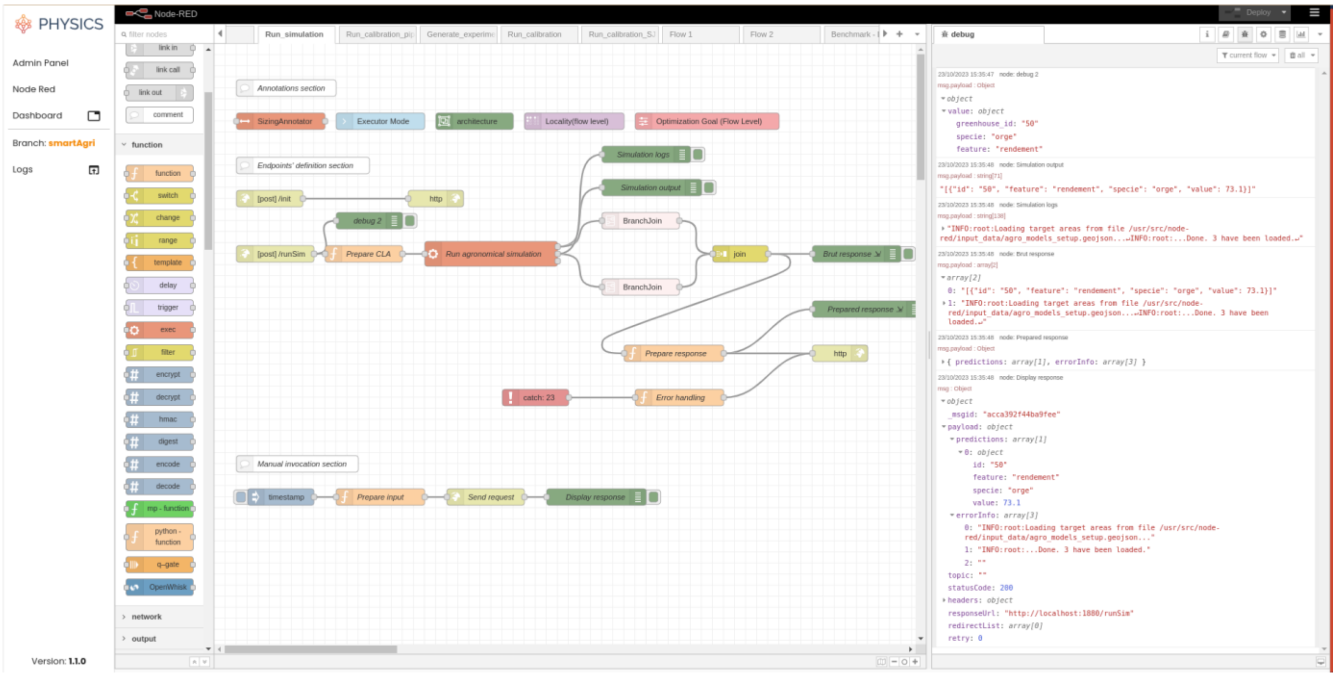


Name: Prepare input

```
msg.payload = {};
msg.payload.value = {};
msg.payload.value.greenhouse_id = "50";
msg.payload.value.specie = "orge";
msg.payload.value.feature = "rendement";

return msg;
```

Method: POST
URL: localhost:1880/runSim
☐ Enable secure (SSL/TLS) connection
☑ Use authentication
Type: basic authentication
Username:
Password:
☐ Enable connection keep-alive
☐ Use proxy
☐ Only send non-2xx responses to Catch node
Return: a parsed JSON object
Name: Send request
Tip: If the JSON parse fails the fetched string is returned as-is.

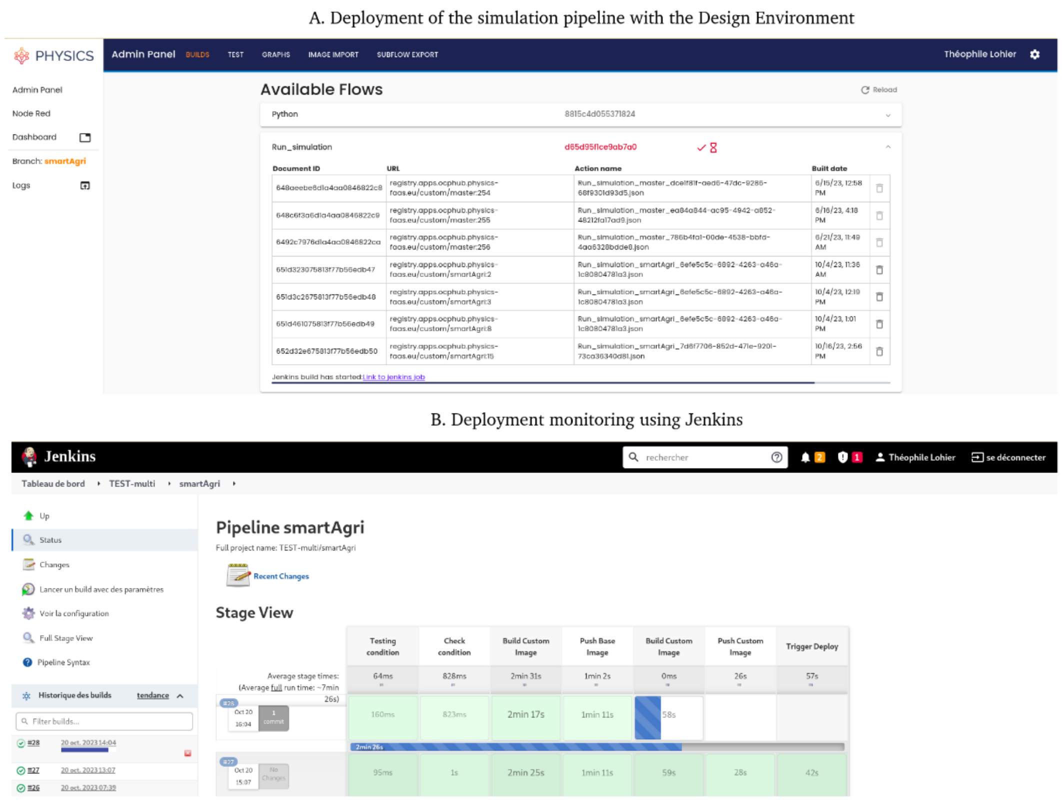
Figure 25: Adaptation of Python script for simulation pipeline execution.



```
23/10/2023 15:35:47  node: debug 2
msg.payload : Object
▾object
 ▾value: object
    greenhouse_id: "50"
    specie: "orge"
    feature: "rendement"
23/10/2023 15:35:48  node: Simulation output
msg.payload : string[71]
"[{"id": "50", "feature": "rendement", "specie": "orge", "value": 73.1}]"
23/10/2023 15:35:48  node: Simulation logs
msg.payload : string[138]
▸"INFO:root:Loading target areas from file /usr/src/node-
red/input_data/agro_models_setup.geojson...↵INFO:root:...Done. 3 have been loaded.↵"
23/10/2023 15:35:48  node: Brut response
msg.payload : array[2]
▾array[2]
  0: "[{"id": "50", "feature": "rendement", "specie": "orge", "value": 73.1}]"
 ▸1: "INFO:root:Loading target areas from file /usr/src/node-
  red/input_data/agro_models_setup.geojson...↵INFO:root:...Done. 3 have been
  loaded.↵"
23/10/2023 15:35:48  node: Prepared response
msg.payload : Object
▸{ predictions: array[1], errorInfo: array[3] }
23/10/2023 15:35:48  node: Display response
msg : Object
▾object
  _msgid: "acca392f44ba9fee"
 ▾payload: object
   ▾predictions: array[1]
     ▾0: object
        id: "50"
        feature: "rendement"
        specie: "orge"
        value: 73.1
   ▾errorInfo: array[3]
      0: "INFO:root:Loading target areas from file /usr/src/node-
      red/input_data/agro_models_setup.geojson..."
      1: "INFO:root:...Done. 3 have been loaded."
      2: ""
  topic: ""
  statusCode: 200
 ▸headers: object
  responseUrl: "http://localhost:1880/runSim"
  redirectList: array[0]
  retry: 0
```

Figure 26: Local testing of the simulation pipeline using the Node-RED debug window.

Once the local tests passed, the simulation pipeline has been deployed using the admin panel of the Design Environment (Figure 27-A). The steps of the deployment process can be monitored using Jenkin (Figure 27-B).


A. Deployment of the simulation pipeline with the Design Environment

Available Flows

Python 8815c4d055371824

Run_simulation d65d95f1ce9ab7a0

Document ID	URL	Action name	Built date
648aeebe6d1a4aa0846822c8	registry.apps.ocphub.physics-faas.eu/custom/master:254	Run_simulation_master_dce1f81f-aed6-47dc-9286-68f9301d93d5.json	6/15/23, 12:58 PM
648c6f3a6d1a4aa0846822c9	registry.apps.ocphub.physics-faas.eu/custom/master:255	Run_simulation_master_ea84a844-ac95-4942-a852-48212fa17ad9.json	6/16/23, 4:18 PM
6492c7976d1a4aa0846822ca	registry.apps.ocphub.physics-faas.eu/custom/master:256	Run_simulation_master_786b4fa1-00de-4538-bbfd-4aa6328bdde8.json	6/21/23, 11:49 AM
651d323075813f77b56edb47	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:2	Run_simulation_smartAgri_6efe5c5c-6892-4263-a46a-1c80804781a3.json	10/4/23, 11:36 AM
651d3c2675813f77b56edb48	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:3	Run_simulation_smartAgri_6efe5c5c-6892-4263-a46a-1c80804781a3.json	10/4/23, 12:19 PM
651d461075813f77b56edb49	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:8	Run_simulation_smartAgri_6efe5c5c-6892-4263-a46a-1c80804781a3.json	10/4/23, 1:01 PM
652d32e675813f77b56edb50	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:15	Run_simulation_smartAgri_7d8f7706-852d-471e-9201-73ca36340d81.json	10/16/23, 2:56 PM

Jenkins build has started:Link to jenkins job

B. Deployment monitoring using Jenkins

#28 20 oct. 2023 14:04
#27 20 oct. 2023 13:07
#26 20 oct. 2023 07:39

Pipeline smartAgri

Full project name: TEST-multi/smartAgri

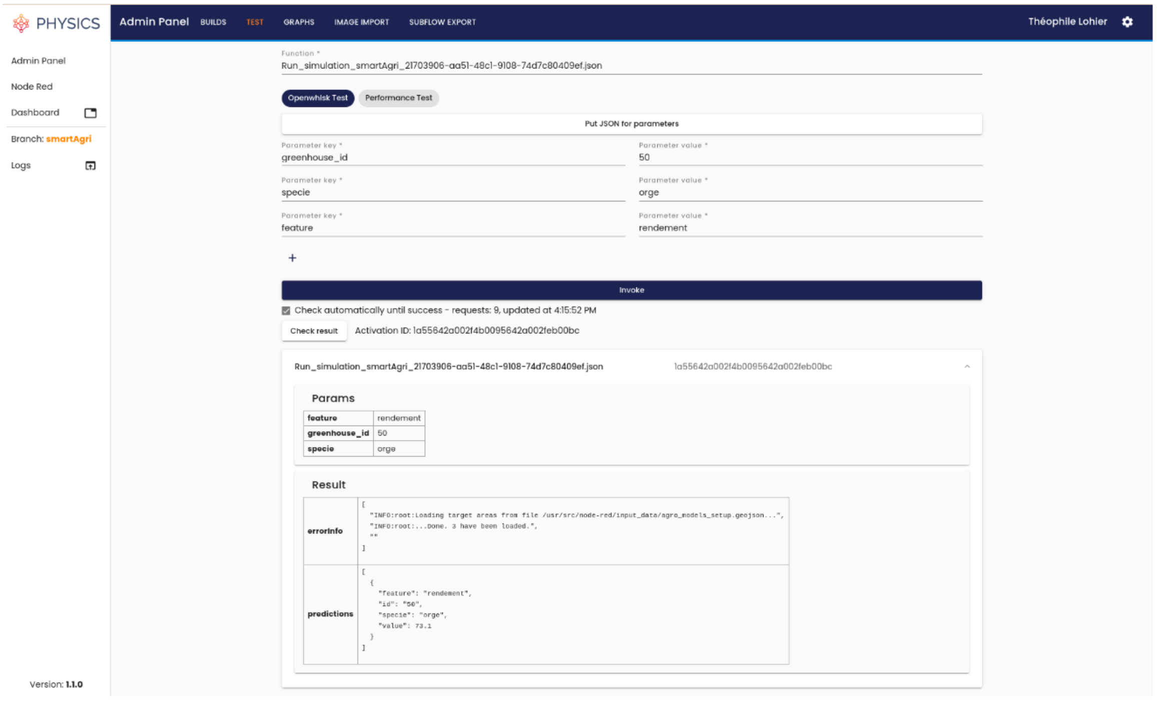
Stage View

	Testing condition	Check condition	Build Custom Image	Push Base Image	Build Custom Image	Push Custom Image	Trigger Deploy
Average stage times: (Average full run time: ~7min 26s)	64ms	828ms	2min 31s	1min 2s	0ms	26s	57s
#28 Oct 20 16:04 1 commit	160ms	823ms	2min 17s	1min 11s	58s		
#27 Oct 20 15:07 No Changes	95ms	1s	2min 25s	1min 11s	59s	28s	42s

2min 26s

Figure 27: Simulation pipeline deployment using the Design Environment (A) and deployment monitoring with Jenkins (B).

Once the deployment is completed, the Design Environment allows us to test the workflow as FaaS. The input of the pipeline can be specified via a form and the output is provided in a table when execution succeeds (Figure 28).



Function *
Run_simulation_smartAgri_21703906-aa51-48c1-9108-74d7c80409ef.json

Parameter key * greenhouse_id	Parameter value * 50
Parameter key * specie	Parameter value * orge
Parameter key * feature	Parameter value * rendement

Check automatically until success - requests: 9, updated at 4:15:52 PM

Activation ID: 1a55642a002f4b0095642a002feb00bc

Run_simulation_smartAgri_21703906-aa51-48c1-9108-74d7c80409ef.json 1a55642a002f4b0095642a002feb00bc

Params

feature	rendement
greenhouse_id	50
specie	orge

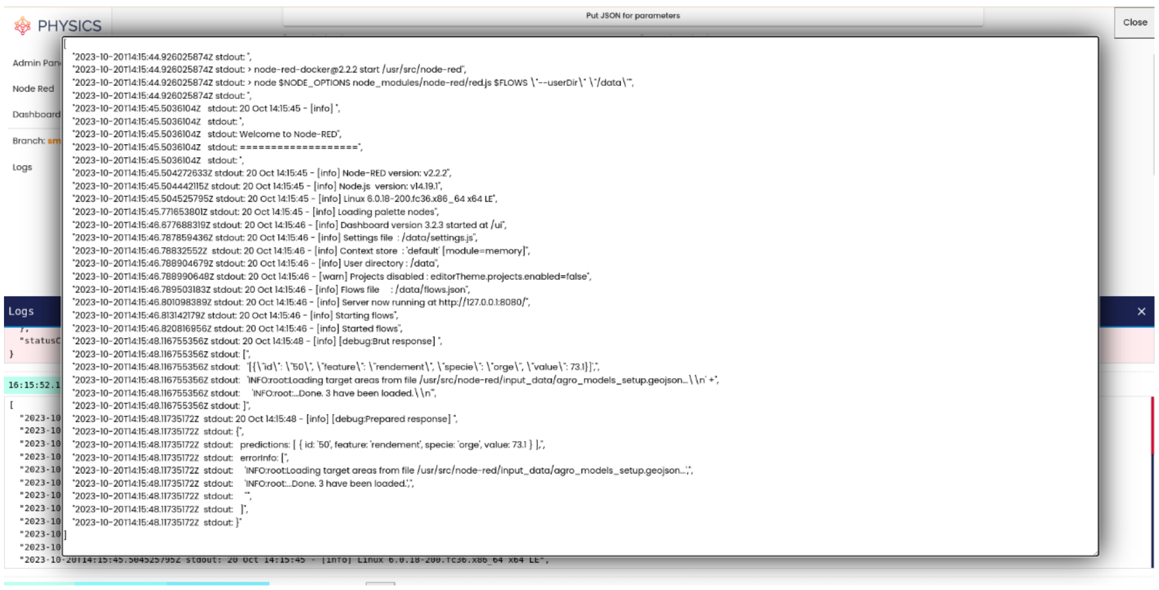
Result

errorInfo
```
[
  "INFO:root:Loading target areas from file /usr/src/node-red/input_data/agro_models_setup.geojson...",
  "INFO:root:...Done. 3 have been loaded.",
  ""
]
```

predictions
```
[
  {
    "feature": "rendement",
    "id": "50",
    "specie": "orge",
    "value": 73.1
  }
]
```

Figure 28: Local testing of the simulation pipeline using the Node-RED debug window.

Moreover, the admin panel provides access to detailed logs in which the debug messages defined in the flow are made available (Figure XX).



```
[
 "2023-10-20T14:15:44.926025874Z stdout: ",
 "2023-10-20T14:15:44.926025874Z stdout: > node-red-docker@2.2.2 start /usr/src/node-red",
 "2023-10-20T14:15:44.926025874Z stdout: > node $NODE_OPTIONS node_modules/node-red/red.js $FLOWS \"--userDir\" \"/data\"",
 "2023-10-20T14:15:44.926025874Z stdout: ",
 "2023-10-20T14:15:45.5036104Z  stdout: 20 Oct 14:15:45 - [info] ",
 "2023-10-20T14:15:45.5036104Z  stdout: ",
 "2023-10-20T14:15:45.5036104Z  stdout: Welcome to Node-RED",
 "2023-10-20T14:15:45.5036104Z  stdout: ===================",
 "2023-10-20T14:15:45.5036104Z  stdout: ",
 "2023-10-20T14:15:45.504272633Z stdout: 20 Oct 14:15:45 - [info] Node-RED version: v2.2.2",
 "2023-10-20T14:15:45.504442115Z stdout: 20 Oct 14:15:45 - [info] Node.js  version: v14.19.1",
 "2023-10-20T14:15:45.504525795Z stdout: 20 Oct 14:15:45 - [info] Linux 6.0.18-200.fc36.x86_64 x64 LE",
 "2023-10-20T14:15:45.771653801Z stdout: 20 Oct 14:15:45 - [info] Loading palette nodes",
 "2023-10-20T14:15:46.677688319Z stdout: 20 Oct 14:15:46 - [info] Dashboard version 3.2.3 started at /ui",
 "2023-10-20T14:15:46.787859436Z stdout: 20 Oct 14:15:46 - [info] Settings file  : /data/settings.js",
 "2023-10-20T14:15:46.78832552Z  stdout: 20 Oct 14:15:46 - [info] Context store  : 'default' [module=memory]",
 "2023-10-20T14:15:46.788904679Z stdout: 20 Oct 14:15:46 - [info] User directory : /data",
 "2023-10-20T14:15:46.788990648Z stdout: 20 Oct 14:15:46 - [warn] Projects disabled : editorTheme.projects.enabled=false",
 "2023-10-20T14:15:46.789503183Z stdout: 20 Oct 14:15:46 - [info] Flows file     : /data/flows.json",
 "2023-10-20T14:15:46.801098389Z stdout: 20 Oct 14:15:46 - [info] Server now running at http://127.0.0.1:8080/",
 "2023-10-20T14:15:46.813142179Z stdout: 20 Oct 14:15:46 - [info] Starting flows",
 "2023-10-20T14:15:46.820816956Z stdout: 20 Oct 14:15:46 - [info] Started flows",
 "2023-10-20T14:15:48.116755356Z stdout: 20 Oct 14:15:48 - [info] [debug:Brut response] ",
 "2023-10-20T14:15:48.116755356Z stdout: [",
 "2023-10-20T14:15:48.116755356Z stdout:   '[{\"id\": \"50\", \"feature\": \"rendement\", \"specie\": \"orge\", \"value\": 73.1}]',",
 "2023-10-20T14:15:48.116755356Z stdout:   'INFO:root:Loading target areas from file /usr/src/node-red/input_data/agro_models_setup.geojson...\\n' +",
 "2023-10-20T14:15:48.116755356Z stdout:     'INFO:root:...Done. 3 have been loaded.\\n'",
 "2023-10-20T14:15:48.116755356Z stdout: ]",
 "2023-10-20T14:15:48.11735172Z  stdout: 20 Oct 14:15:48 - [info] [debug:Prepared response] ",
 "2023-10-20T14:15:48.11735172Z  stdout: {",
 "2023-10-20T14:15:48.11735172Z  stdout:   predictions: [ { id: '50', feature: 'rendement', specie: 'orge', value: 73.1 } ],",
 "2023-10-20T14:15:48.11735172Z  stdout:   errorInfo: [",
 "2023-10-20T14:15:48.11735172Z  stdout:     'INFO:root:Loading target areas from file /usr/src/node-red/input_data/agro_models_setup.geojson...',",
 "2023-10-20T14:15:48.11735172Z  stdout:     'INFO:root:...Done. 3 have been loaded.',",
 "2023-10-20T14:15:48.11735172Z  stdout:     ''",
 "2023-10-20T14:15:48.11735172Z  stdout:   ]",
 "2023-10-20T14:15:48.11735172Z  stdout: }"
]
```

Figure 29: Local testing of the simulation pipeline using the Node-RED debug window.

4.3.3 Calibration pipeline

4.3.3.1 Pipeline definition using Node-Red

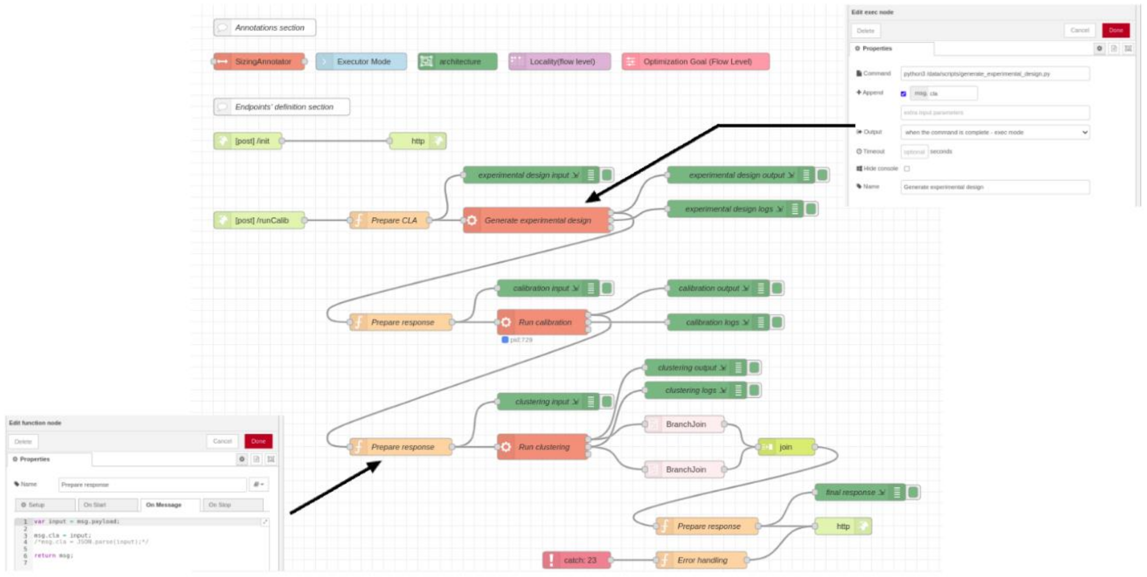
The pipeline is made up of three main steps:

1. Generation of a design of experiment. In its simplest form it consists of a grid where each node represents a model parameter set.
2. Running the simulation pipeline for each parameter set and evaluating the results regarding in situ observations.
3. Clustering of the model parameters set based on the evaluation with the objective of identifying groups of parameters sets which reproduce part or full plant behavior.

We propose three implementations for this pipeline:

1. The first implementation is used as a baseline for implementation and testing of the different components required. All the steps are embedded in one flow and the parameter set evaluations (step 2) are run sequentially.
2. In the second implementation we seek to take advantage of the FaaS approach by parallelizing the run of the simulation pipeline. All the steps are embedded in one flow and the parameter set evaluations are run in parallel using the split and join pattern.
3. In the third implementation we seek to integrate multi-processing at the function level: the parameter set evaluations will be distributed in FaaS mode as in the second implementation, but we add a level of parallelization with each function being able to distribute the evaluations over the processors available on the pod. This distribution is enabled by the split and join pattern with specific parametrization.

The first and third steps of the pipeline should not be parallelized since they are respectively input generation and output gathering and analyzing. The second step is highly parallelizable since each simulation can be run and evaluated independently from the others. In the first version of the flow each step is defined as an independent exec node triggering a dedicated Python script. The communication between the nodes is ensured by function nodes, writing the output of the previous node in the input field of the message retrieved by the current node (Figure 30).



Edit exec node

- Command: python3 /data/scripts/generate_experimental_design.py
- Append: [✓] msg. cla
- extra input parameters
- Output: when the command is complete - exec mode
- Timeout: optional seconds
- Hide console: ☐
- Name: Generate experimental design

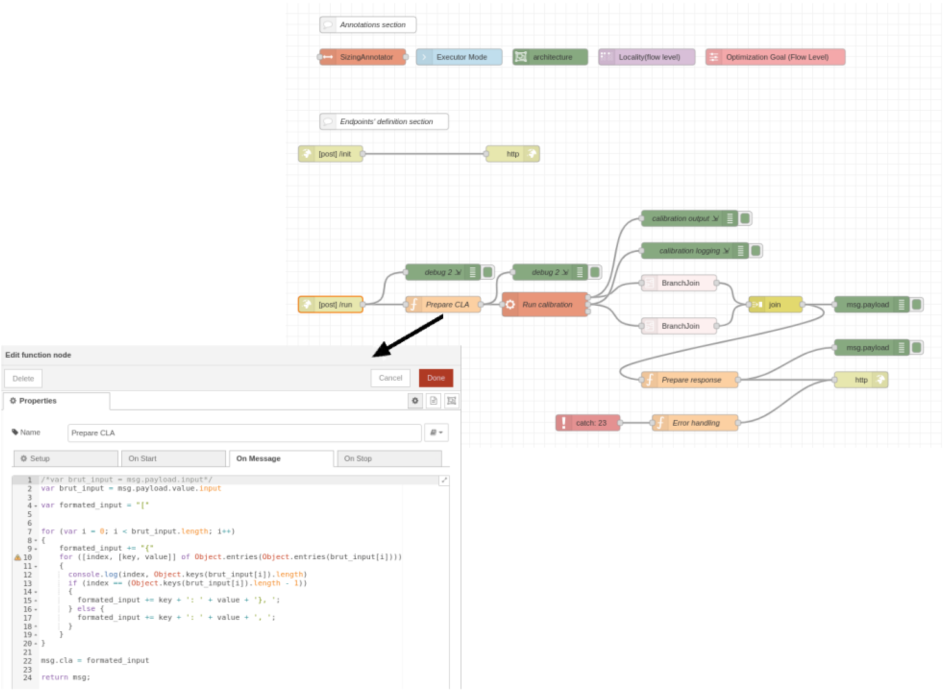
Edit function node

- Name: Prepare response
- Setup | On Start | On Message | On Stop

```
var input = msg.payload;

msg.cla = input;
/*msg.cla = JSON.parse(input);*/

return msg;

```

Figure 30: Node-RED flow describing the calibration pipeline.

To take advantage of the split and join pattern we first defined a dedicated parameter set evaluation flow which takes as input a vector with every element being a node of the design of experiment. The formatting of the input vector as a string is operated by a dedicated node. This input string is then decoded by the calibration script, implemented in Python and triggered using the exec node (Figure 31).



Edit function node

- Name: Prepare CLA
- Setup | On Start | On Message | On Stop

```
/*var brut_input = msg.payload.input*/
var brut_input = msg.payload.value.input

var formated_input = "["


for (var i = 0; i < brut_input.length; i++)
{
    formated_input += "{"
    for ([index, [key, value]] of Object.entries(Object.entries(brut_input[i])))
    {
      console.log(index, Object.keys(brut_input[i]).length)
      if (index == (Object.keys(brut_input[i]).length - 1))
      {
        formated_input += key + ': ' + value + '}, ';
      } else {
        formated_input += key + ': ' + value + ', ';
      }
    }
}

msg.cla = formated_input

return msg;
```

Figure 31: Node-RED flow allowing to run calibration from an array of nodes such as generated by the Python script for experimental design generation.

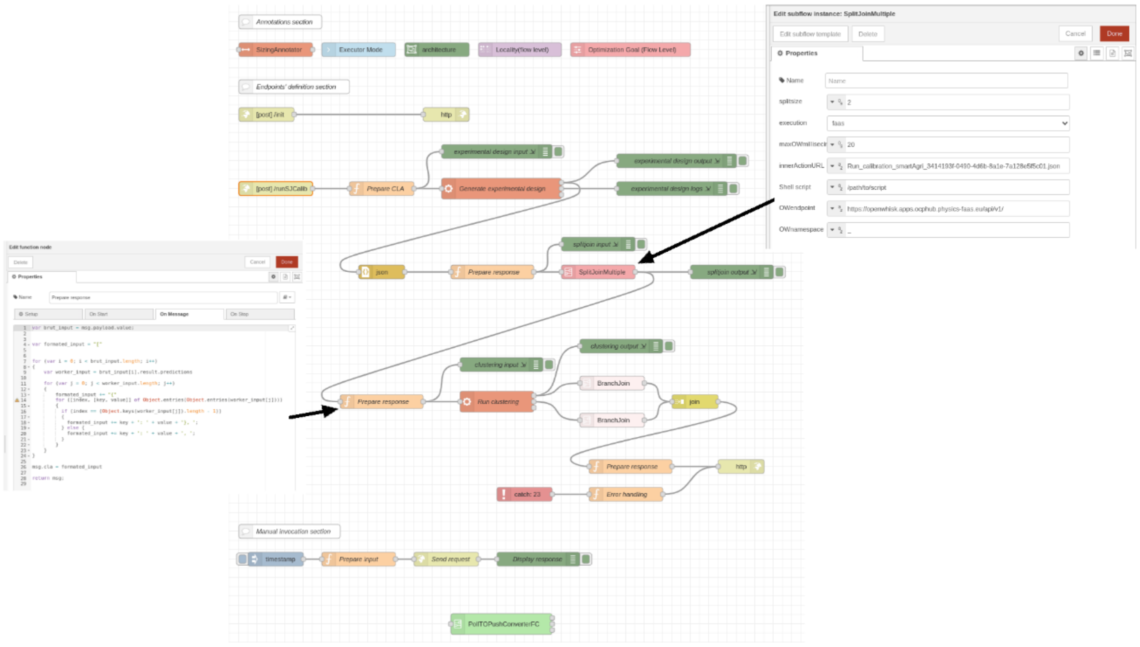
Then we can call this calibration flow using the split and join node. The routine allowing to prepare the input for the calibration script from the message sent by the split and join node is embedded in the calibration flow (Figure 32). On the other hand, the output of the split and join node needs to be aggregated before being sent to the clustering node (Figure 32). The parametrization of the split and join node is straightforward and can be done using the Node-RED UI. It required the URL of the function to run in parallel (innerActionURL field in the node setup) which in our case is described by the calibration flow and the number of splits to perform (splitsize field in the node setup).

The split and join pattern allow us to take advantage of FaaS approach to run the parameter set evaluation in parallel. This pattern also includes an option to distribute the evaluations over the processors of a server. In the third implementation of the calibration pipeline, we combine FaaS and multiprocessing.

To achieve this double parallelization, we adapt the dedicated parameter set evaluation flow used in the second version. The main difference lies in the calling of python script describing the parameter set evaluation procedure. In the second implementation the script was run using an exec node (see node called Run calibration, XXXXXX31) while here the script is run in parallel over the processors thanks to the split and join pattern. The parameterization of the split and join node is very straightforward with the command to run the script being specified through the Shell script field of the subflow. We also need to specify that we want to run it using multiprocessing (field execution of the subflow) and the number of evaluations we want to run on each processor (field splitsize of the subflow, Figure 33).

To ensure consistency between the FaaS and the FaaS-multiprocessing pipeline we added a formatting sequence after the split and join node.

To use the FaaS-multiprocessing version of the calibration pipeline we only need to change the innerActionURL field in the split and join pattern so that it points on the multiprocessing version of the parameter set evaluation function (Figure 32).

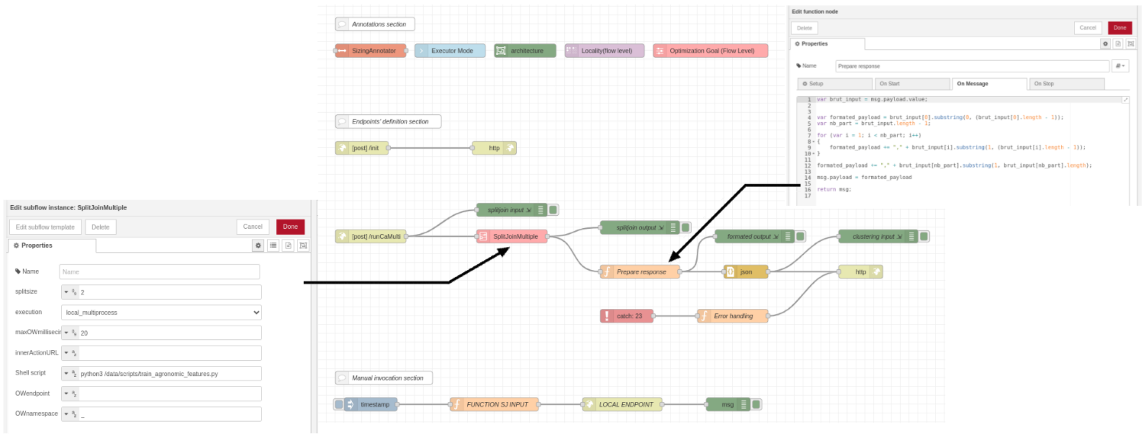


Name: Name
splitsize: 2
execution: faas
maxOWmillisec: 20
innerActionURL: Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json
Shell script: /path/to/script
OWendpoint: https://openwhisk.apps.ocphub.physics-faas.eu/api/v1/
OWnamespace: _

Name: Prepare response

```
var brut_input = msg.payload.value;


var formated_input = "["


for (var i = 0; i < brut_input.length; i++)
{
    var worker_input = brut_input[i].result.predictions

    for (var j = 0; j < worker_input.length; j++)
    {
        formated_input += "{"
        for ([index, [key, value]] of Object.entries(Object.entries(worker_input[j])))
        {
          if (index == (Object.keys(worker_input[j]).length - 1))
          {
            formated_input += key + ': ' + value + '}, ';
          } else {
            formated_input += key + ': ' + value + ', ';
          }
        }
    }
}

msg.cla = formated_input

return msg;
```

Figure 32: Calibration pipeline as a Node-RED flow using the split and join pattern for parallelization.



Name: Name
splitsize: 2
execution: local_multiprocess
maxOWmillisecin: 20
innerActionURL:
Shell script: python3 /data/scripts/train_agronomic_features.py
OWendpoint:
OWnamespace: _

Name: Prepare response

```
var brut_input = msg.payload.value;


var formated_payload = brut_input[0].substring(0, (brut_input[0].length - 1));
var nb_part = brut_input.length - 1;

for (var i = 1; i < nb_part; i++)
{
    formated_payload += "," + brut_input[i].substring(1, (brut_input[i].length - 1));
}

formated_payload += "," + brut_input[nb_part].substring(1, brut_input[nb_part].length);

msg.payload = formated_payload

return msg;
```

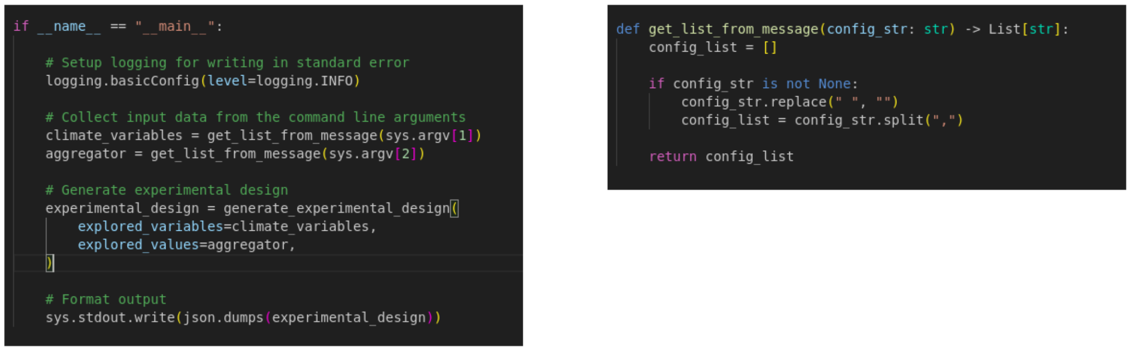
Figure 33: Node-RED flow allowing to run parameter set evaluation from an array of nodes such as generated by the Python script for experimental design generation with parallelization over processors.

4.3.3.2 Adaptation of legacy codes

To take advantage of FaaS the three steps of the calibration pipeline should be run separately. In this way each step could be run as a function with optimal placement according to resources needed and the simulations run during the calibration process could be parallelized using the split and join pattern. The legacy codes, consisting in one main script has then been split into three distinct scripts with ad-hoc I/O management and logging.

The script for experimental design generation has been implemented based on the legacy codes describing the calibration pipeline. It takes as input two lists, one describing the climate variables to use in the model and the other the way these variables are post-processed (Figure 34). The parameters to evaluate depend on the target agronomic variables, which are defined in the file referenced as AREA_FILE (read by the script for parameter set merit evaluation, Figure 35.A).

A. Main for Experimental design generation **B. Input formatting function**



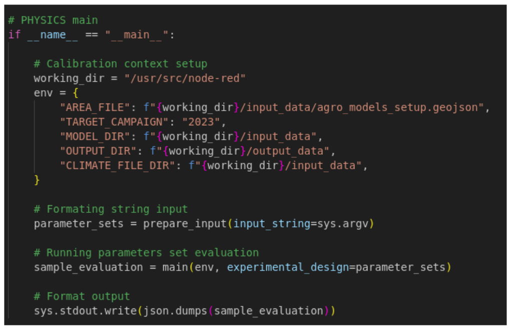
```
if __name__ == "__main__":

    # Setup logging for writing in standard error
    logging.basicConfig(level=logging.INFO)

    # Collect input data from the command line arguments
    climate_variables = get_list_from_message(sys.argv[1])
    aggregator = get_list_from_message(sys.argv[2])

    # Generate experimental design
    experimental_design = generate_experimental_design(
        explored_variables=climate_variables,
        explored_values=aggregator,
    )

    # Format output
    sys.stdout.write(json.dumps(experimental_design))
```

```
def get_list_from_message(config_str: str) -> List[str]:
    config_list = []

    if config_str is not None:
        config_str.replace(" ", "")
        config_list = config_str.split(",")

    return config_list
```

Figure 34: Adaptation of legacy codes for experimental design evaluation.

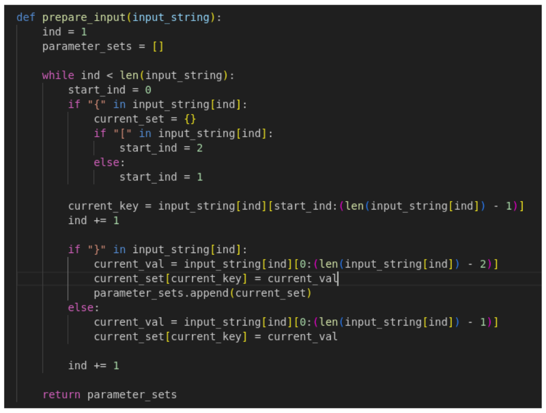
The script for parameter set merit evaluation integrates a main function which run the simulations with one or several parameters sets and perform the estimation its merit according to in situ measurements provided in the file referenced in the code as AREA_FILE (Figure 35.A). The parameter set list is built from the string provided as input using the prepare_input function (Figure 35.B).

A. Calibration main function



```python
# PHYSICS main
if __name__ == "__main__":

    # Calibration context setup
    working_dir = "/usr/src/node-red"
    env = {
        "AREA_FILE": f"{working_dir}/input_data/agro_models_setup.geojson",
        "TARGET_CAMPAIGN": "2023",
        "MODEL_DIR": f"{working_dir}/input_data",
        "OUTPUT_DIR": f"{working_dir}/output_data",
        "CLIMATE_FILE_DIR": f"{working_dir}/input_data",
    }

    # Formating string input
    parameter_sets = prepare_input(input_string=sys.argv)

    # Running parameters set evaluation
    sample_evaluation = main(env, experimental_design=parameter_sets)

    # Format output
    sys.stdout.write(json.dumps(sample_evaluation))
```

B. Parameter sets formatting



```python
def prepare_input(input_string):
    ind = 1
    parameter_sets = []

    while ind < len(input_string):
        start_ind = 0
        if "{" in input_string[ind]:
            current_set = {}
            if "[" in input_string[ind]:
                start_ind = 2
            else:
                start_ind = 1

        current_key = input_string[ind][start_ind:(len(input_string[ind]) - 1)]
        ind += 1

        if "}" in input_string[ind]:
            current_val = input_string[ind][0:(len(input_string[ind]) - 2)]
            current_set[current_key] = current_val
            parameter_sets.append(current_set)
        else:
            current_val = input_string[ind][0:(len(input_string[ind]) - 1)]
            current_set[current_key] = current_val

        ind += 1

    return parameter_sets
```

Figure 35: Adaptation of legacy codes for parameter set merit evaluation.

The clustering of parameter sets according to their merit is a novel feature of the calibration pipeline implemented in PHYSICS. It allows to build a probability distribution for each target agronomic features, increasing the reliability and robustness of the method. The script integrates a function for building the dataset provided as input to the clustering algorithm from the output of SplitJoin node (Figure 36).

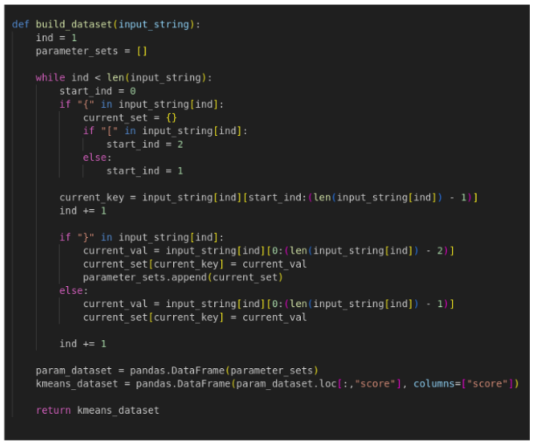
4.3.3.3 <u>Pipeline deployment and testing</u>

First, the calibration pipeline has been tested as a function to ensure that the I/O management between the different nodes of the pipeline is correct. Moreover, this flow will provide a baseline for performance evaluation.

Then we deploy (Figure 37.A) and test the parameter set evaluation flow both locally and using the Design Environment (Figure 37.B) to ensure that messages sent by the split and join node will be handled correctly.

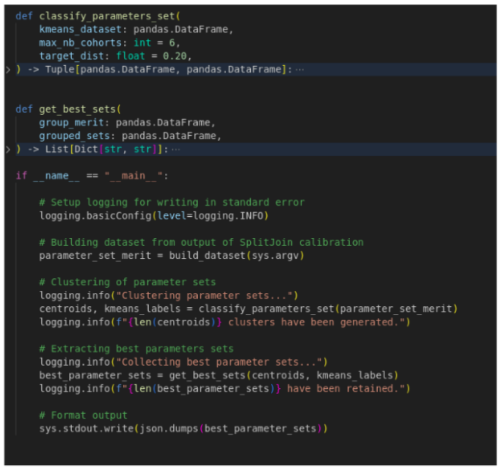
Once the parameter set evaluation flow has been validated, the FaaS parallelized calibration pipeline has been tested. In real condition, this pipeline will be triggered by a Cybeletech engineer when needed and there is no gain in deploying it as a function. The testing has then been performed using the Node-RED panel of the Design Environment with parameter set evaluation being split and run as FaaS.

A. Dataset construction function



```python
def build_dataset(input_string):
    ind = 1
    parameter_sets = []

    while ind < len(input_string):
        start_ind = 0
        if "{" in input_string[ind]:
            current_set = {}
            if "[" in input_string[ind]:
                start_ind = 2
            else:
                start_ind = 1

        current_key = input_string[ind][start_ind:(len(input_string[ind]) - 1)]
        ind += 1

        if "}" in input_string[ind]:
            current_val = input_string[ind][0:(len(input_string[ind]) - 2)]
            current_set[current_key] = current_val
            parameter_sets.append(current_set)
        else:
            current_val = input_string[ind][0:(len(input_string[ind]) - 1)]
            current_set[current_key] = current_val

        ind += 1

    param_dataset = pandas.DataFrame(parameter_sets)
    kmeans_dataset = pandas.DataFrame(param_dataset.loc[:,"score"], columns=["score"])

    return kmeans_dataset
```

B. Main function for Parameter set selection



```python
def classify_parameters_set(
    kmeans_dataset: pandas.DataFrame,
    max_nb_cohorts: int = 6,
    target_dist: float = 0.20,
) -> Tuple[pandas.DataFrame, pandas.DataFrame]: ···


def get_best_sets(
    group_merit: pandas.DataFrame,
    grouped_sets: pandas.DataFrame,
) -> List[Dict[str, str]]: ···

if __name__ == "__main__":

    # Setup logging for writing in standard error
    logging.basicConfig(level=logging.INFO)

    # Building dataset from output of SplitJoin calibration
    parameter_set_merit = build_dataset(sys.argv)

    # Clustering of parameter sets
    logging.info("Clustering parameter sets...")
    centroids, kmeans_labels = classify_parameters_set(parameter_set_merit)
    logging.info(f"{len(centroids)} clusters have been generated.")

    # Extracting best parameters sets
    logging.info("Collecting best parameter sets...")
    best_parameter_sets = get_best_sets(centroids, kmeans_labels)
    logging.info(f"{len(best_parameter_sets)} have been retained.")

    # Format output
    sys.stdout.write(json.dumps(best_parameter_sets))
```

Figure 36: Adaptation of legacy codes for parameter set selection.

In the running example depicted in Figure 38, a parameter set grid including 27 nodes is generated by the Generate experimental design node (Figure 39). The string returned by this node is parsed in order to create an array of parameter sets which is sent to the SplitJoinMultiple node. The nodes are divided into three groups and three functions are invoked as FaaS, each processing one of the groups (Figure 40). The three evaluations outputted by the SplitJoinMultiple node are aggregated in a single string which is sent to the Run clustering node. The parameter sets are sorted according to their score and the best parameters sets are returned (Figure 41).

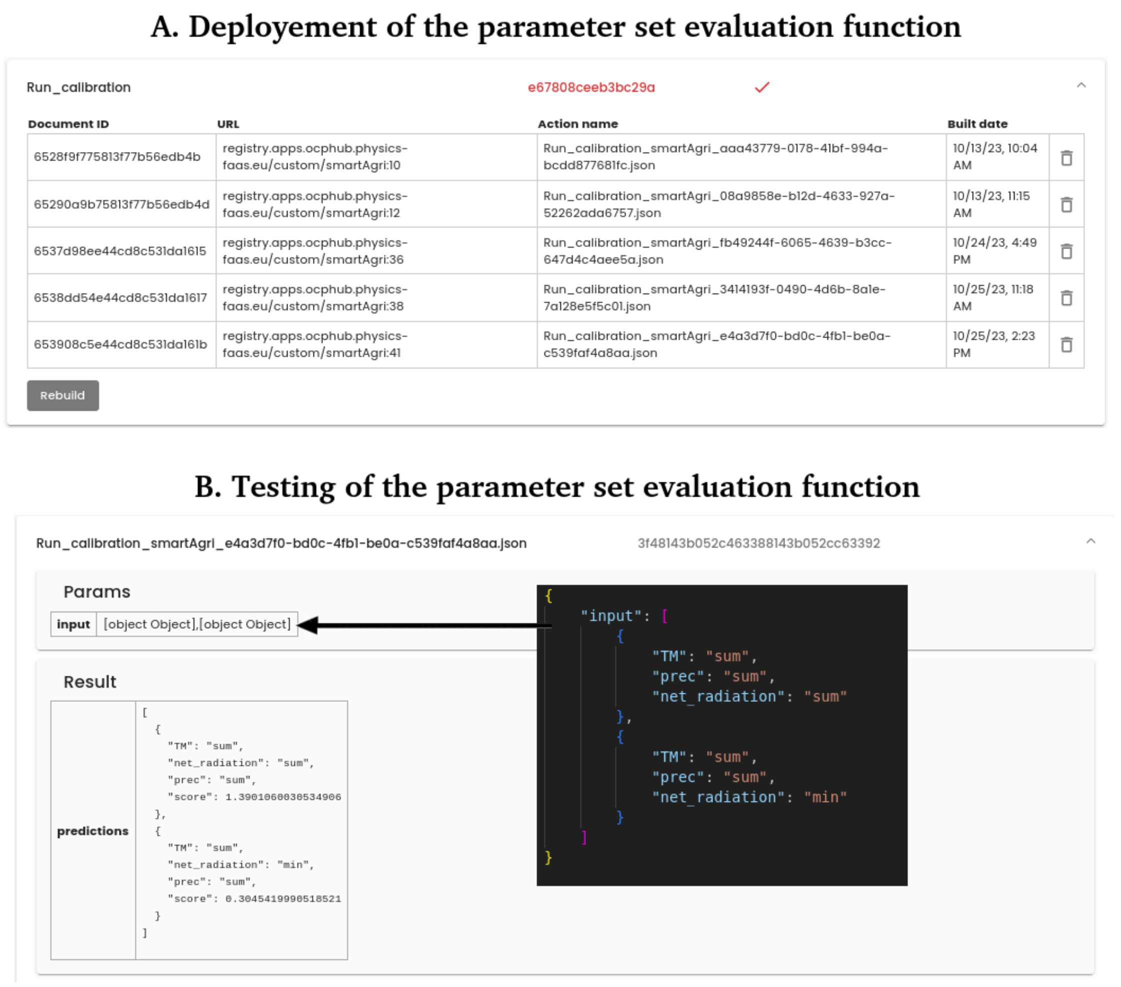
The same experiment has been conducted for the multiprocessing version of the parameter set evaluation with deployment and testing of the multiprocessed flow.

4.3.4 Concluding remarks

The source material for the prototype as described 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.


A. Deployement of the parameter set evaluation function
Run_calibration
e67808ceeb3bc29a
Rebuild
B. Testing of the parameter set evaluation function
Run_calibration_smartAgri_e4a3d7f0-bd0c-4fb1-be0a-c539faf4a8aa.json
3f48143b052c463388143b052cc63392
Params
Result

Document ID	URL	Action name	Built date
6528f9f775813f77b56edb4b	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:10	Run_calibration_smartAgri_aaa43779-0178-41bf-994a-bcdd87768lfc.json	10/13/23, 10:04 AM
65290a9b75813f77b56edb4d	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:12	Run_calibration_smartAgri_08a9858e-b12d-4633-927a-52262ada6757.json	10/13/23, 11:15 AM
6537d98ee44cd8c531da1615	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:36	Run_calibration_smartAgri_fb49244f-6065-4639-b3cc-647d4c4aee5a.json	10/24/23, 4:49 PM
6538dd54e44cd8c531da1617	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:38	Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json	10/25/23, 11:18 AM
653908c5e44cd8c531da161b	registry.apps.ocphub.physics-faas.eu/custom/smartAgri:41	Run_calibration_smartAgri_e4a3d7f0-bd0c-4fb1-be0a-c539faf4a8aa.json	10/25/23, 2:23 PM

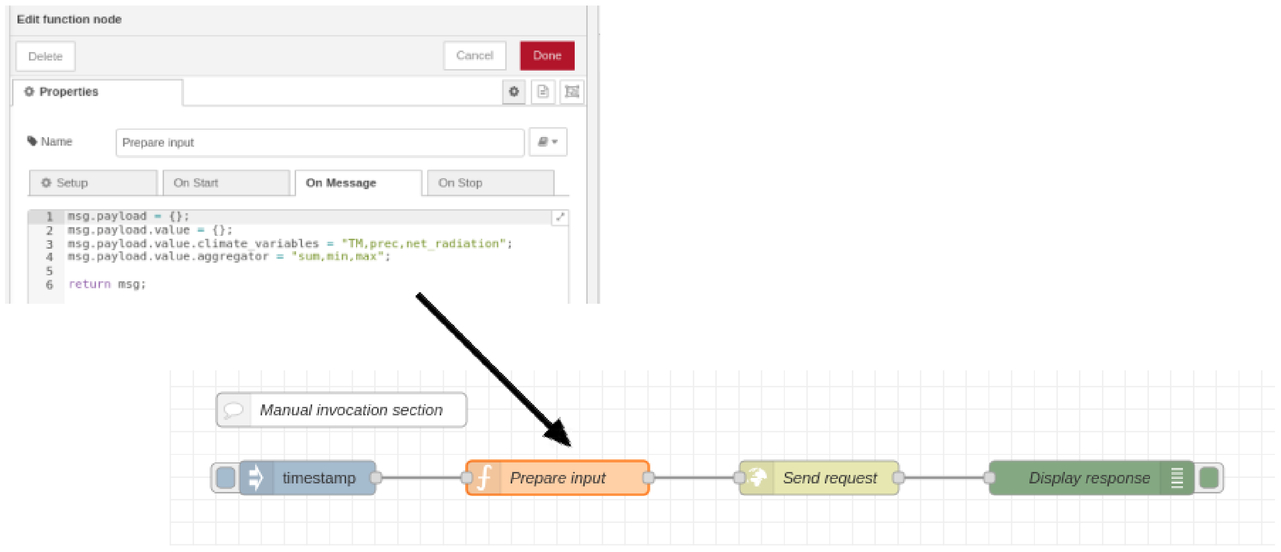
input	[object Object],[object Object]

```
{
    "input": [
        {
            "TM": "sum",
            "prec": "sum",
            "net_radiation": "sum"
        },
        {
            "TM": "sum",
            "prec": "sum",
            "net_radiation": "min"
        }
    ]
}
```

predictions

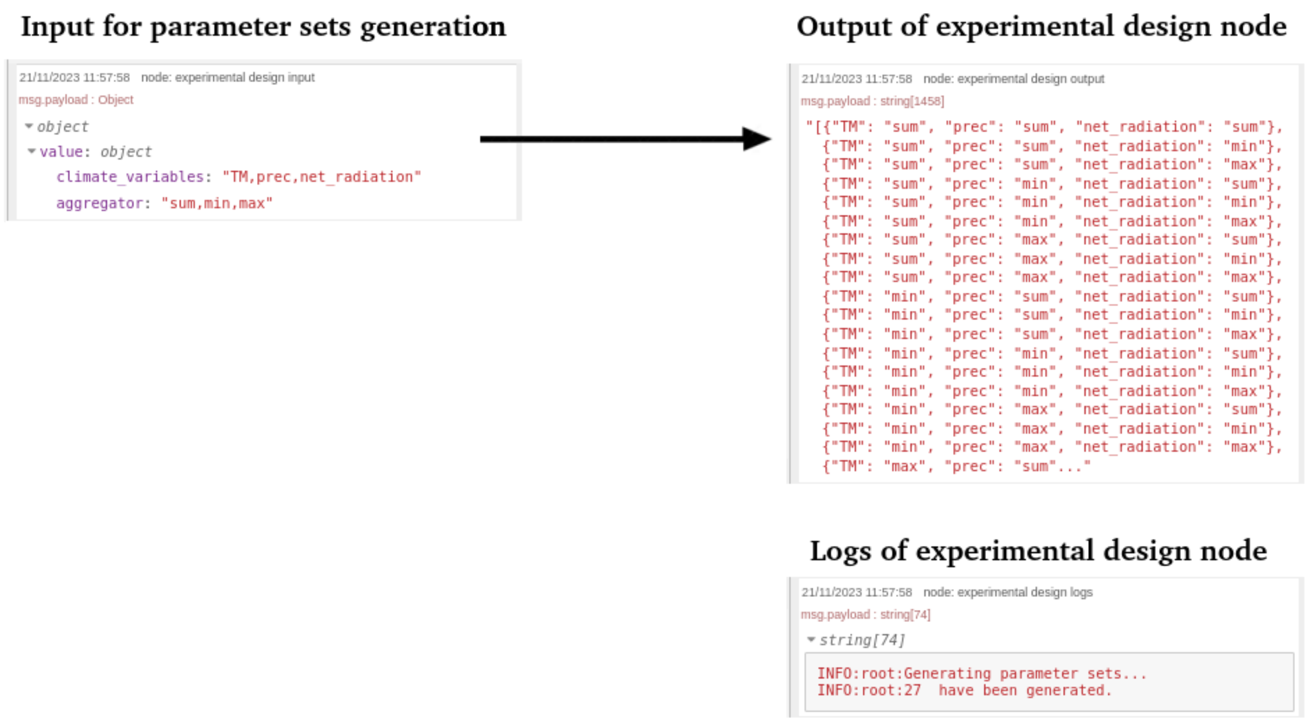
```
[
  {
    "TM": "sum",
    "net_radiation": "sum",
    "prec": "sum",
    "score": 1.3901060030534906
  },
  {
    "TM": "sum",
    "net_radiation": "min",
    "prec": "sum",
    "score": 0.3045419990518521
  }
]
```

Figure 37: Parameter set evaluation flow deployment (A) and testing (B) using the Design Environment.



```
msg.payload = {};
msg.payload.value = {};
msg.payload.value.climate_variables = "TM,prec,net_radiation";
msg.payload.value.aggregator = "sum,min,max";

return msg;
```

Figure 38: Flow for local invocation of the split and join calibration pipeline.


Input for parameter sets generation

```
21/11/2023 11:57:58   node: experimental design input
msg.payload : Object
▾object
 ▾value: object
    climate_variables: "TM,prec,net_radiation"
    aggregator: "sum,min,max"
```

Output of experimental design node

```
21/11/2023 11:57:58   node: experimental design output
msg.payload : string[1458]
"[{"TM": "sum", "prec": "sum", "net_radiation": "sum"},
  {"TM": "sum", "prec": "sum", "net_radiation": "min"},
  {"TM": "sum", "prec": "sum", "net_radiation": "max"},
  {"TM": "sum", "prec": "min", "net_radiation": "sum"},
  {"TM": "sum", "prec": "min", "net_radiation": "min"},
  {"TM": "sum", "prec": "min", "net_radiation": "max"},
  {"TM": "sum", "prec": "max", "net_radiation": "sum"},
  {"TM": "sum", "prec": "max", "net_radiation": "min"},
  {"TM": "sum", "prec": "max", "net_radiation": "max"},
  {"TM": "min", "prec": "sum", "net_radiation": "sum"},
  {"TM": "min", "prec": "sum", "net_radiation": "min"},
  {"TM": "min", "prec": "sum", "net_radiation": "max"},
  {"TM": "min", "prec": "min", "net_radiation": "sum"},
  {"TM": "min", "prec": "min", "net_radiation": "min"},
  {"TM": "min", "prec": "min", "net_radiation": "max"},
  {"TM": "min", "prec": "max", "net_radiation": "sum"},
  {"TM": "min", "prec": "max", "net_radiation": "min"},
  {"TM": "min", "prec": "max", "net_radiation": "max"},
  {"TM": "max", "prec": "sum"..."
```

Logs of experimental design node

```
21/11/2023 11:57:58   node: experimental design logs
msg.payload : string[74]
▾string[74]
 INFO:root:Generating parameter sets...
 INFO:root:27  have been generated.
```

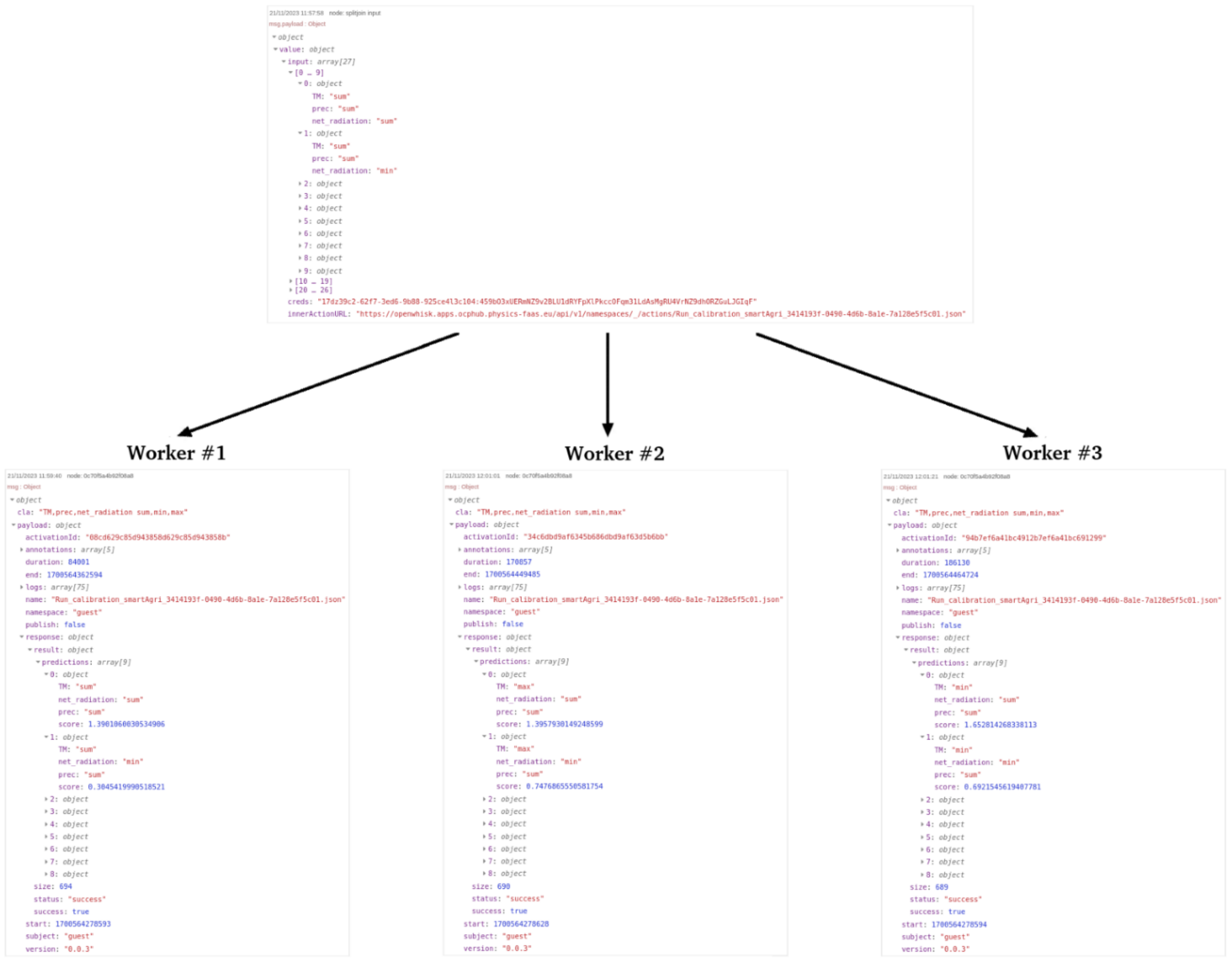
Figure 39: Input, output and logs of the Generate experimental design node for the running example.

After having described the first version of the "smart agriculture Prototype", we now look back at the initial functional specifications that were defined in section 3.3. 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 3 is discussed in Table 6.

Table 6: 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 if 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	Fulfilled. The simulation and calibration pipelines can be invoked using the test panel of the Design Environment or the endpoint exposed by the Node-Red flow.
FRS-UC3-05	Fulfilled. The simulation and calibration pipelines have been deployed as FaaS using the DE and benefits from annotations and patterns developed in PHYSICS.
FRS-UC3-06	Fulfilled. The split and join pattern has been integrated in the calibration flow in order to parallelized the runs, allowing to reduce the runtime.

In the smart Agriculture Use Case, three applications have been adapted and tested using the components developed in PHYSICS. The data collection pipeline, adapted during the first period, takes advantage of the Edge-ETL pattern to increase the reliability of the process and reduce the adaptation time from one greenhouse to another. The deployment on the edge is facilitated by the Design Environment interface which allows to build and retrieve an image embedding all the required components. The simulation pipeline has been adapted so that it can be easily deployed and run as FaaS, significantly reducing the infrastructure costs. Finally, the calibration pipeline has been adapted to integrate the split and join pattern, which enables high level of parallelization using the FaaS paradigm. These two last applications have been successfully deployed in the cloud and tested using the Design Environment.



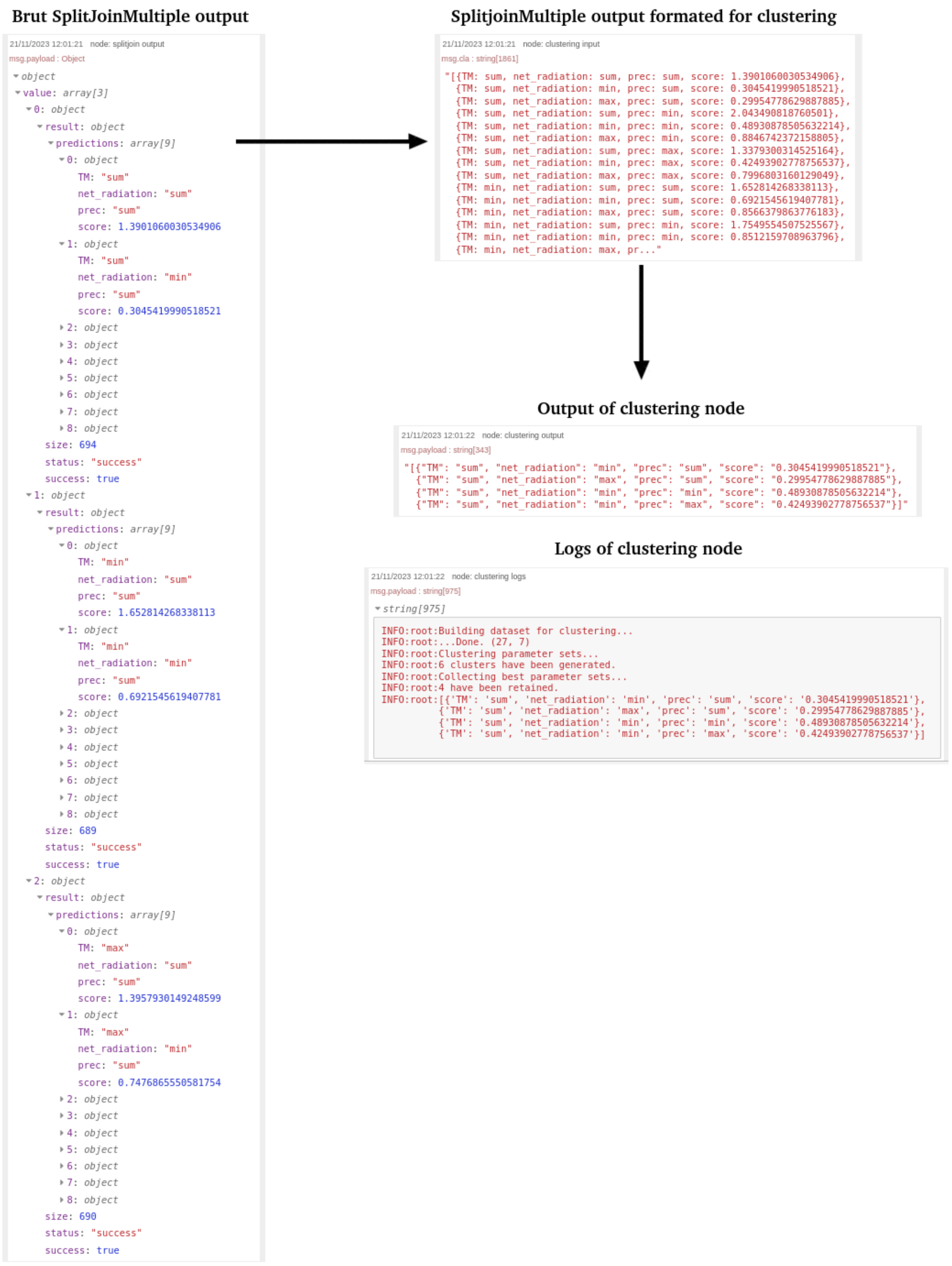
```
21/11/2023 11:57:58   node: splitjoin input
msg.payload : Object
▾object
 ▾value: object
   ▾input: array[27]
    ▾[0 … 9]
      ▾0: object
         TM: "sum"
         prec: "sum"
         net_radiation: "sum"
      ▾1: object
         TM: "sum"
         prec: "sum"
         net_radiation: "min"
      ▸2: object
      ▸3: object
      ▸4: object
      ▸5: object
      ▸6: object
      ▸7: object
      ▸8: object
      ▸9: object
    ▸[10 … 19]
    ▸[20 … 26]
   creds: "17dz39c2-62f7-3ed6-9b88-925ce4l3c104:459b03xUERmNZ9v2BLU1dRYFpXlPkccOFqm31LdAsMgRU4VrNZ9dh0RZGuLJGIqF"
   innerActionURL: "https://openwhisk.apps.ocphub.physics-faas.eu/api/v1/namespaces/_/actions/Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json"
```

```
21/11/2023 11:59:40   node: 0c70f5a4b92f08a8
msg : Object
▾object
  cla: "TM,prec,net_radiation sum,min,max"
 ▾payload: object
    activationId: "08cd629c85d943858d629c85d943858b"
   ▸annotations: array[5]
    duration: 84001
    end: 1700564362594
   ▸logs: array[75]
    name: "Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json"
    namespace: "guest"
    publish: false
   ▾response: object
     ▾result: object
       ▾predictions: array[9]
         ▾0: object
            TM: "sum"
            net_radiation: "sum"
            prec: "sum"
            score: 1.3901060030534906
         ▾1: object
            TM: "sum"
            net_radiation: "min"
            prec: "sum"
            score: 0.3045419990518521
         ▸2: object
         ▸3: object
         ▸4: object
         ▸5: object
         ▸6: object
         ▸7: object
         ▸8: object
      size: 694
      status: "success"
      success: true
    start: 1700564278593
    subject: "guest"
    version: "0.0.3"
```

```
21/11/2023 12:01:01   node: 0c70f5a4b92f08a8
msg : Object
▾object
  cla: "TM,prec,net_radiation sum,min,max"
 ▾payload: object
    activationId: "34c6dbd9af6345b686dbd9af63d5b6bb"
   ▸annotations: array[5]
    duration: 170857
    end: 1700564449485
   ▸logs: array[75]
    name: "Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json"
    namespace: "guest"
    publish: false
   ▾response: object
     ▾result: object
       ▾predictions: array[9]
         ▾0: object
            TM: "max"
            net_radiation: "sum"
            prec: "sum"
            score: 1.3957930149248599
         ▾1: object
            TM: "max"
            net_radiation: "min"
            prec: "sum"
            score: 0.7476865550581754
         ▸2: object
         ▸3: object
         ▸4: object
         ▸5: object
         ▸6: object
         ▸7: object
         ▸8: object
      size: 690
      status: "success"
      success: true
    start: 1700564278628
    subject: "guest"
    version: "0.0.3"
```

```
21/11/2023 12:01:21   node: 0c70f5a4b92f08a8
msg : Object
▾object
  cla: "TM,prec,net_radiation sum,min,max"
 ▾payload: object
    activationId: "94b7ef6a41bc4912b7ef6a41bc691299"
   ▸annotations: array[5]
    duration: 186130
    end: 1700564464724
   ▸logs: array[75]
    name: "Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json"
    namespace: "guest"
    publish: false
   ▾response: object
     ▾result: object
       ▾predictions: array[9]
         ▾0: object
            TM: "min"
            net_radiation: "sum"
            prec: "sum"
            score: 1.652814268338113
         ▾1: object
            TM: "min"
            net_radiation: "min"
            prec: "sum"
            score: 0.6921545619407781
         ▸2: object
         ▸3: object
         ▸4: object
         ▸5: object
         ▸6: object
         ▸7: object
         ▸8: object
      size: 689
      status: "success"
      success: true
    start: 1700564278594
    subject: "guest"
    version: "0.0.3"
```

Figure 40: Output of the Generate experimental design node formatted for the SplitJoinMultiple node and output of each worker. In this context a worker is an instance of the parameter set evaluation function deployed and invoked as FaaS.


Brut SplitJoinMultiple output
SplitjoinMultiple output formated for clustering
Output of clustering node
Logs of clustering node

Brut SplitJoinMultiple output

```
21/11/2023 12:01:21   node: splitjoin output
msg.payload : Object
▾object
 ▾value: array[3]
  ▾0: object
   ▾result: object
    ▾predictions: array[9]
     ▾0: object
        TM: "sum"
        net_radiation: "sum"
        prec: "sum"
        score: 1.3901060030534906
     ▾1: object
        TM: "sum"
        net_radiation: "min"
        prec: "sum"
        score: 0.3045419990518521
     ▸2: object
     ▸3: object
     ▸4: object
     ▸5: object
     ▸6: object
     ▸7: object
     ▸8: object
   size: 694
   status: "success"
   success: true
  ▾1: object
   ▾result: object
    ▾predictions: array[9]
     ▾0: object
        TM: "min"
        net_radiation: "sum"
        prec: "sum"
        score: 1.652814268338113
     ▾1: object
        TM: "min"
        net_radiation: "min"
        prec: "sum"
        score: 0.6921545619407781
     ▸2: object
     ▸3: object
     ▸4: object
     ▸5: object
     ▸6: object
     ▸7: object
     ▸8: object
   size: 689
   status: "success"
   success: true
  ▾2: object
   ▾result: object
    ▾predictions: array[9]
     ▾0: object
        TM: "max"
        net_radiation: "sum"
        prec: "sum"
        score: 1.3957930149248599
     ▾1: object
        TM: "max"
        net_radiation: "min"
        prec: "sum"
        score: 0.7476865550581754
     ▸2: object
     ▸3: object
     ▸4: object
     ▸5: object
     ▸6: object
     ▸7: object
     ▸8: object
   size: 690
   status: "success"
   success: true
```

SplitjoinMultiple output formated for clustering

```
21/11/2023 12:01:21   node: clustering input
msg.cla : string[1861]
"[{TM: sum, net_radiation: sum, prec: sum, score: 1.3901060030534906},
  {TM: sum, net_radiation: min, prec: sum, score: 0.3045419990518521},
  {TM: sum, net_radiation: max, prec: sum, score: 0.29954778629887885},
  {TM: sum, net_radiation: sum, prec: min, score: 2.043490818760501},
  {TM: sum, net_radiation: min, prec: min, score: 0.48930878505632214},
  {TM: sum, net_radiation: max, prec: min, score: 0.8846742372158805},
  {TM: sum, net_radiation: sum, prec: max, score: 1.3379300314525164},
  {TM: sum, net_radiation: min, prec: max, score: 0.42493902778756537},
  {TM: sum, net_radiation: max, prec: max, score: 0.7996803160129049},
  {TM: min, net_radiation: sum, prec: sum, score: 1.652814268338113},
  {TM: min, net_radiation: min, prec: sum, score: 0.6921545619407781},
  {TM: min, net_radiation: max, prec: sum, score: 0.8566379863776183},
  {TM: min, net_radiation: sum, prec: min, score: 1.7549554507525567},
  {TM: min, net_radiation: min, prec: min, score: 0.8512159708963796},
  {TM: min, net_radiation: max, pr..."
```

Output of clustering node

```
21/11/2023 12:01:22   node: clustering output
msg.payload : string[343]
"[{"TM": "sum", "net_radiation": "min", "prec": "sum", "score": "0.3045419990518521"},
  {"TM": "sum", "net_radiation": "max", "prec": "sum", "score": "0.29954778629887885"},
  {"TM": "sum", "net_radiation": "min", "prec": "min", "score": "0.48930878505632214"},
  {"TM": "sum", "net_radiation": "min", "prec": "max", "score": "0.42493902778756537"}]"
```

Logs of clustering node

```
21/11/2023 12:01:22   node: clustering logs
msg.payload : string[975]
▾string[975]
INFO:root:Building dataset for clustering...
INFO:root:...Done. (27, 7)
INFO:root:Clustering parameter sets...
INFO:root:6 clusters have been generated.
INFO:root:Collecting best parameter sets...
INFO:root:4 have been retained.
INFO:root:[{'TM': 'sum', 'net_radiation': 'min', 'prec': 'sum', 'score': '0.3045419990518521'},
          {'TM': 'sum', 'net_radiation': 'max', 'prec': 'sum', 'score': '0.29954778629887885'},
          {'TM': 'sum', 'net_radiation': 'min', 'prec': 'min', 'score': '0.48930878505632214'},
          {'TM': 'sum', 'net_radiation': 'min', 'prec': 'max', 'score': '0.42493902778756537'}]
```

Figure 41: Raw output of the SplitJoinMultiple node and formatted string used as input of the Run clustering node, output of the clustering and corresponding logs.

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 demonstrates three scenarios (inference, phenotyping and synthesis), 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 demonstrators are the final work of experimenting with the different use cases in a "FaaSified" manner using 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 usage of the PHYSICS tools. The prototypes are documented using the design environment of PHYSICS and they all represent different utilization of the developed PHYSICS components.

All three use cases have finalized their experimentation activities, meaning that the flows have been created for all their scenarios, and they have been tested locally. Services have been deployed from these flows, and the services have also been tested. The outcome of the experimentation (and thus of Task 6.3, Use Cases Adaptation and Experimentation) is a total of 8 scenario services across the 3 use cases, all of them ready for evaluation in the context of Task 6.4, Use Cases Evaluation).

6 BIBLIOGRAPHY

Franke, N., Hennecke, A., Gezer, V., Harms, C., op den Akker, H., Pnevmatikakis, A., . . . Touloupou, M. (2022). *D6.4 - Application Scenarios Definition V2.* PHYSICS Consortium.

Kousiouris, G. (2023). *D3.2 - Functional and semantic continuum services design framework Scientific Report and Prototype Description V2.* PHYSICS Consortium.

Mamelli, A., Costantino, D., Salomon, J., Tomas Bolivar, L., Castillo Nieto, A., & Sánchez Fernández, C. (2023). *D6.2 - Prototype of the Integrated PHYSICS solution framework and RAMP V2.* PHYSICS Consortium.

op den Akker, H., Pnevmatikakis, A., Kanavos, S., Labropoulos, G., Bekiaris, D., Babalis, P., . . . Harms, C. (2022). *D6.5: PHYSICS Application Prototype V1.* PHYSICS Consortium.

Patiño, M., Azqueta, A., Mengual, L., Li, T., Kousiouris, G., Tsarsitalidis, S., . . . Pelegr. (2022). *D2.5 - PHYSICS Reference Architecture Specification V2.* PHYSICS Consortium.