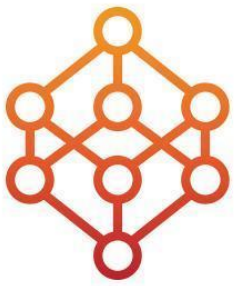


PHYSICS

oPTIMIZED HYBRID SPACE-TIME SERVICE CONTINUUM IN FAAS

D6.8 – PHYSICS APPLICATION PROTOTYPES EVALUATION V2

Lead Beneficiary	CYBELETECH
Work Package Ref.	WP6 – Use Cases Adaptation, Experimentation, Evaluation
Task Ref.	T6.4 – Use Cases Evaluation
Deliverable Title	D6.8 – PHYSICS Application Prototypes Evaluation V2
Due Date	2023-12-31
Delivered Date	
Revision Number	1.0
Dissemination Level	Public (PU)
Type	Other (ORDP)
Document Status	Draft
Review Status	Internally Reviewed
Document Acceptance	
EC Project Officer	Mr. Stefano Foglietta

H2020 ICT 40 2020 Research and Innovation Action

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

CONTRIBUTING PARTNERS

Partner Acronym	Role[1]	Name Surname[2]
CYBE	Lead Beneficiary	Lohier, Théophile
FTDS	Contributor	Franke, Niklas
iSPRINT	Contributor	Pnevmatikakis, Aristodemos
iSPRINT	Contributor	op den Akker, Harm
iSPRINT	Contributor	Kanavos, Stathis
DFKI	Contributor	Gezer, Volkan
DFKI	Contributor	Harms, Carsten
DFKI	Contributor	Kolek, Maciej
GFT	Internal Reviewer	Maurizio Megliola
INNOV	Internal Reviewer	George Fatouros
INNOV	Quality Assurance	George Fatouros

REVISION HISTORY

Version	Date	Partner(s)	Description
0.1	2023-11-20	CYBE	Initial setup of document and initial ToC suggestions
0.2	2023-12-08	DFKI	Updated smart manufacturing KPI evaluation
0.3	2023-12-10	iSPRINT	Updated eHealth KPI evaluation
0.4	2023-12-11	CYBE	Updated smart agriculture KPI evaluation
1..0	2023-12-12	CYBE, DFKI, iSPRINT	Internal review version
1.1	2023-12-15	GFT	Version reviewed
1.2	2023-12-16	INNOV	Version reviewed
2.0	2023-12-18	CYBE, DFKI, iSPRINT	Internal review comments implemented
2.1	2023-12-19	INNOV	Quality Assurance review
3.0	2023-12-21	CYBE, DFKI, iSPRINT	Quality Assurance review comments implemented

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

LIST OF ABBREVIATIONS

API	Application Programmer's Interface
BG	Business Goal
CI/CD	Continuous Integration/Continuous Delivery
DaaS	Desktop as a service
DE	Design Environment
ETL	Extract, transform, load
GPU	Graphical Processing Unit
FaaS	Function as a Service
KPI	Key Performance Indicator
ORDP	Open Research Data Pilot
OW	OpenWhisk
OKR	Objectives and Key Results
QA	Quality Assurance
QC	Quality Control
SaaS	Software as a service
SUS	System Usability Scale

EXECUTIVE SUMMARY

The scope of this deliverable is to present the results of application prototype evaluation for the three Use Cases: Smart Manufacturing, eHealth, and Smart Agriculture.

The purpose of "*Task 6.4: Use Case Evaluation"* is to:

- Evaluate the functionalities achieved in comparison to requirements defined in WP2
- Evaluate the application-level KPIs defined in T6.2
- Evaluate the project outputs regarding usability, flexibility, and adaptability
- Evaluate the application prototypes against the feedback received by external stakeholders

According to the Use Cases, the objectives, and then the KPIs, are different: In the Smart Manufacturing Use Case, the main concern is the reliability of the service, while in eHealth, the principal objective is to ensure a near real time response regardless of the load. In addition, the Smart Agriculture Use Case integrates an objective regarding computation time.

On the other hand, each Use Case presents its own specificities regarding application deployment and will then mobilize different PHYSICS components. In the first phase of the project, the Smart Agriculture Use Case focused on the deployment on the Edge, while during the intensification period emphasis has been put on Cloud deployment of workflows with high computational needs. The eHealth Use Case needs a deployment in the Cloud as FaaS, while Smart Manufacturing Use Case requires a hybrid solution with some functions running on the Edge while some others are run as FaaS.

The deliverable "*D6.8 PHYSICS Application Prototype Evaluation V2*":

- Describes the common methodology used to convert Use Case objectives defined in T6.2 into measurable performance indicators, called KPIs
- Details the process of KPIs definition and its outcomes for each Use Case
- Draws baseline evaluations for selected KPIs from currently deployed applications
- Reports on first evaluations for selected KPIs based on application prototypes developed in T6.3 with emphasis on available functionalities
- Describes a method to evaluate the usability and adaptability of PHYSICS components
- Analyzes the associated results
- Provides an overview of the preparation for the impact intensification period

During the first phase, a set of KPIs has been defined for each Use Case and evaluated using currently deployed applications and PHYSICS application prototypes. The comparison of these two evaluations highlights the benefits of the PHYSICS Design Environment and components, including the increase in reliability and the facilitation and acceleration of application development and deployment. Flexibility and adaptability have been demonstrated by developing and deploying three very different application prototypes based on PHYSICS components. The usability of the PHYSICS Design Environment and of some distinct components has been evaluated through a state-of the art survey, showing that users with different backgrounds are able to take advantage of these components, but also that there is room for improving the user-friendliness of the Design Environment.

This is the next version of the above deliverable, i.e., "*D6.8 PHYSICS Application Prototype Evaluation V2*", which is due to be released in M36 of the project (December 2023). It details the outcomes of the impact intensification period, including the evaluation of KPIs related to the more complex version of the application prototypes and the reevaluation of the first set of KPIs with the fully integrated PHYSICS platform.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

1.1 Objectives of the Deliverable

The objective of this deliverable is to evaluate the PHYSICS platform and the platform components from a user perspective. This deliverable provides the formal definition of the KPIs, its evaluation for the currently deployed Use Case applications, and a complete evaluation using the PHYSICS applications prototypes.

1.2 Insights from other Tasks and Deliverables

As depicted in Figure 1, deliverable *D6.8: Application prototype evaluation V2* builds on the previously delivered documents *D6.4: Application scenarios definition V2* and *D6.6: Application prototype V2*. The currently deployed Use Case applications used to estimate the baseline KPIs are detailed in *D6.6,* and their PHYSICS-compliant counterparts are introduced in *D6.4* and detailed in *D6.6*. *D6.8* also compares the achieved functionalities with the requirements posed during design time, mainly in *WP2* and in *D2.3: State of the Art Analysis and Requirements Definition V2.*

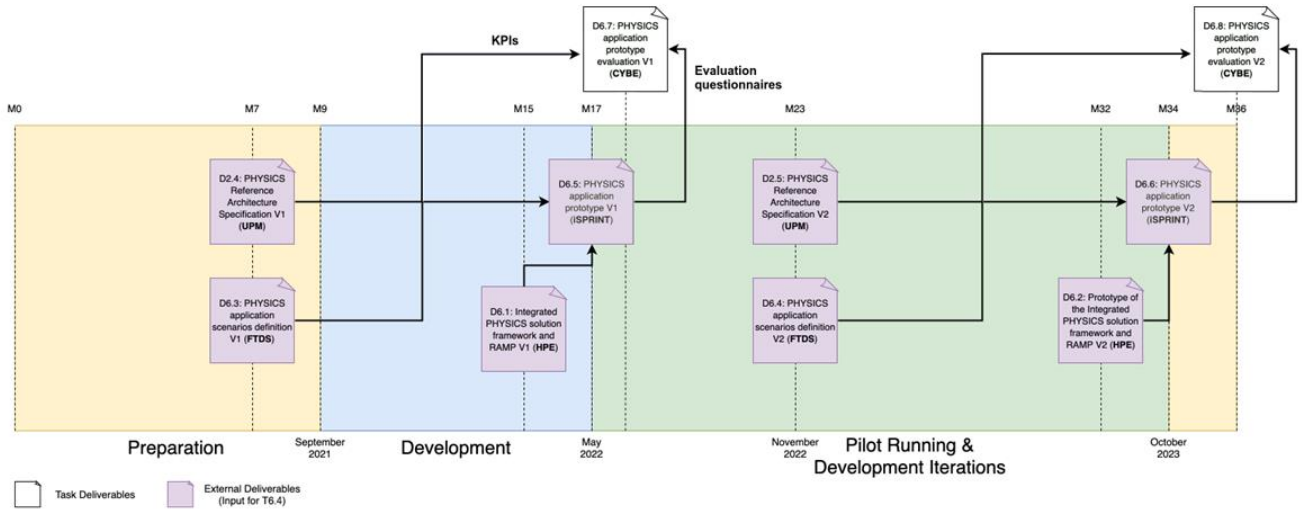


Figure 1: Overview on how D6.8 and previous WP2 and WP6 Deliverables relate to each other in terms of the requirement engineering of the PHYSICS project.

Moreover, to facilitate the reading of this document, the outcome of Task 6.2 about the definition of applications KPIs have been integrated in Section 2.

1.3 Methodology

First, the Use Case objectives regarding PHYSICS have been formalized. These objectives are very dependent on the application, and in addition, each Use Case has its own set of objectives. Workshops have been organized to introduce a common methodology allowing to associate each objective to a measurable performance metric. These metrics, called Key Performance Indicators, have been the basis for the PHYSICS platform evaluation.

To quantify the gain induced by using the PHYSICS platform, the KPIs have been evaluated on the currently deployed Use Case infrastructure and with the available PHYSICS components. The comparison of these two sets of measurements enables to quantify the achievements of the Use Case objectives.

Another key and transversal evaluation criterion of the PHYSICS platform from a user perspective is its usability. Considering the large number of usability evaluation methods, standardized usability questionnaires are valuable tools intended for the assessment of perceived usability (Hornbæk, 2006). A large variety of standardized usability questionnaires for evaluating software systems are available.

Based on the comparative review of (Assila & Ezzedine, 2016) the System Usability Scale has been chosen as a baseline. It has been completed to capture relevant background information about the users and to assess system capabilities and flaws.

Flexibility and adaptability are addressed at the Use Case level by monitoring the efforts required to develop and deploy each application prototype. And at a global level through the development and deployment of application prototypes from three fields: manufacturing, health, and agriculture, with different requirements and constraints. In this context, the performance metric at the global level is the success in application prototype development and deployment.

1.4 Structure

The reminder of the document is structured as follows. Section 2 describes the methodology elaborated during Task 6.2 to define Use Case specific KPIs. Section 3 gives a brief overview of PHYSICS components that have been used to experiment and evaluate the application prototypes.

Sections 4, 5 and 6 are dedicated to the pilots Smart Manufacturing, eHealth, and Smart Agriculture respectively. These sections have the same structure: In subsection 1, Use Case specific KPIs are detailed; in subsection 2, the KPIs evaluation on the infrastructure used in production by the pilots are presented with the aim to establish reference metrics that are then compared to those measured when using the PHYSICS platform. This metrics comparison is detailed in subsection 3, and the cost/benefit trade-offs achieved using PHYSICS are discussed,

The questionnaire used to evaluate PHYSICS platform's usability is introduced in section 7.1 and collected user feedback are analyzed in section 7.2. Finally, conclusions and an outlook for future work is provided in Section 8.

2. KPI DEFINITION

2.1 KPI Design Theory

Companies have the reason to either maximize their operation, optimize activities, or make better decisions. Therefore, they use metrics per definition measures for evaluating efficiency, performance, advantage, or quality. Metrics are simply numbers. Per definition, a metric always has one single dimension. If a metric unites multiple dimensions, it would not be comprehensible why the value of the metric is low. A measure should be tight to only one specific goal. To shape a broader picture of an operation, it makes sense to have a set of metrics that measure all relevant facets of an entire operation. Key Performance indicators are metrics that are used to gauge how good or bad a company or a business achieves its goals. Companies use a huge set of KPIs and rarely get rid of unnecessary ones. KPIs can have motivational effects, can convince others, and allow management from remote.

When wrong things are measured, it will lead to bad decisions, bad behavior and unhappy customers. People and specifically managers can also misuse metrics to their own benefit, to the disadvantage of others or to deceive others. It is crucial that users recognize that KPIs can provide the value to guide people for better decisions or to change behavior and can not only be used to assess, evaluate and analyze.

2.2 Methodology

Developing a good metric requires commitment of all stakeholders. There is not a catalog from which managers can pick the perfect KPI. However, there exists a very good metric that fits an individual purpose. This metric must be measurable, achievable, easy to understand, fraud-proof, and strategically aligned (Davilla, 2018) . In terms of this project, we assessed existing methodologies to create a suitable approach in a technical environment that aligns performance measures for technical components and the pilots' business. During Task T6.2 we were already collecting desired performance measures of the three pilots. The results were reported in Deliverable 6.3 in the Sections 3.19, 3.29 and 3.39. It was determined that good KPIs were not possible to be created with a brainstorming or a list. The desired KPIs as collected with a questionnaire contained following issues: They contained more than one goal, or they contained fuzzy definitions of a performance, like "must be higher", "less or none" or "should be short". Number one issue was that desired KPIs were formulated as goals instead of measurable indicators for the achievement of a goal. Figure 2 shows the process that was created and covered within a workshop which each pilot.

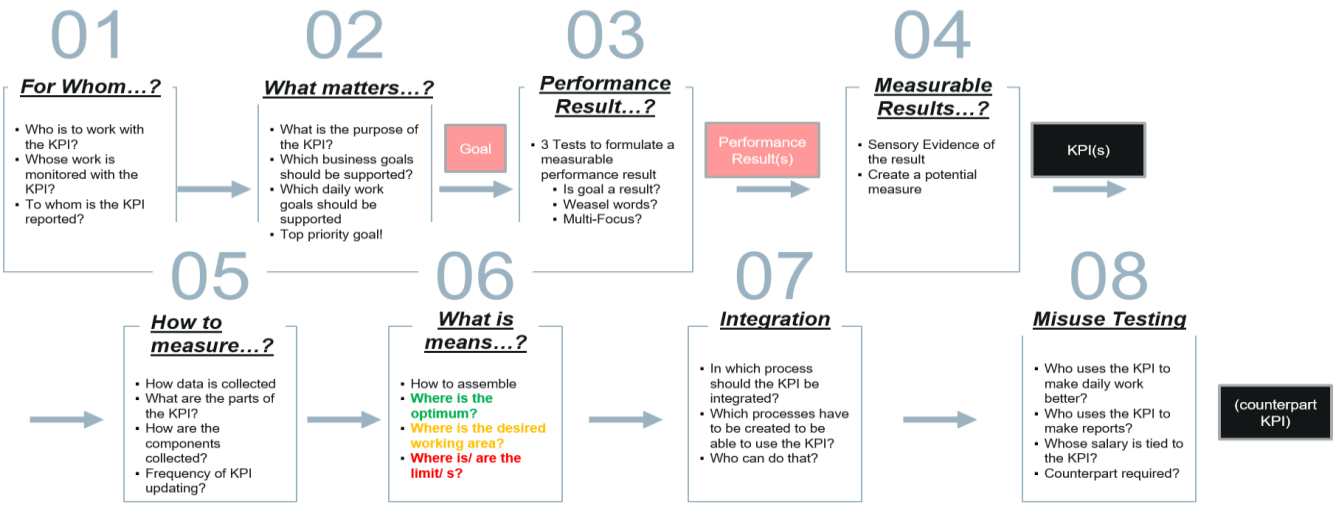


Figure 2: Process used to derive measurable indicators (KPI) from phrased objectives.

In order to create measures that are suitable for the pilots in the context of PHYSICS, we customized existing methodologies for our needs. Two steps in the KPI Design Process were adopted from the Performance

Measurement Process as described by (Barr, 2014). One step is about mapping of measurable results and the translation of goals into results. The other is about finding obvious evidence that a goal was achieved and how to measure the level of goal achievement. Another step took the idea of (Doerr, 2018) of measuring a KPI at the bottleneck of a process.

We focused in the first place during step number one on the users of the KPI to be developed to identify the reporting line of the KPI. We developed specific questions to identify the connection between the operational level that is working with the KPI and the business level that must draw decisions based on the performance. Per the Lean methodology and the Theory of Constraints, a KPI should be assigned to the bottleneck of a process (Doerr, 2018). Therefore, the second step has the goal to identify what matters most for both the business and the operational level and what purpose the KPI should fulfill. This most relevant goal got prioritized via a voting of the workshop members and represented the bottleneck of the process. The prioritized goals needed to be formulated as performance results. To transform the goals into performance results, specific tests were used. The first test qualifies the reformulation into a result. With the second test there was ensured that weasel words get replaced with clearly defined definitions and a third test ensured that the result only includes one focus. The outcome after going with the original goals through this test qualified the goal to be a valid performance result. If a multi-focus gets detected the results get separated and individually handled as separate performance results after step three. Step four was chosen to be included into the methodology because not in all cases the achievement of a result can be detected obviously but only with the observation of an indicator used as sensory evidence. If the performance result cannot meet the requirement of providing sensory evidence on its own, evidence must be found or created within the workshop. After step four, the performance measure got transformed into a key performance indicator that measures the achievement of a goal that aligns business with operation at a bottleneck of a process. The second row in the process model characterizes the steps that consider the rules of the handling of the KPI. Step five identifies parts of the KPI, how they are collected and if the reporting interval of the KPI is in relation with the effort to create the KPI. It checks if the parts of the KPI can be measured directly, must be calculated from other data, or can be drawn from an indirect indicator. Sometimes there will be combinations possible. Step six consists of two parts. The first considers from this baseline of the KPI components the mathematics and the relationship between the components to build the KPI. In the Workshop there were sticky notes used to build the formula. The second part is about the definition boundaries between a desired optimum, a working area that is acceptable and the area which is intolerable. After the mathematics the KPI needs to be assigned to a responsible owner. Step seven audits if an existing process can own the responsibility of the KPI creation or if a new one needs to be established. If an existing one can own the KPI integration it is checked what adjustments in the process need to be made and who will be the person or respectively the role that is responsible. Step number eight is enormously important for the actual implementation in the company, because the collection of KPIs will in any case be associated with people. There are the people who work with the KPI, others who are responsible for the process that the KPI measures, others who report based on the KPIs and possibly also those whose salary is linked to the KPI. During the workshop, participants from the relevant areas of the company will be sensitized to how KPIs can be abused for their own benefit and to the detriment of others. This task secures the misuse of the KPIs by covering possible scenarios with counterpart KPIs. For example, quantitative results can be supplemented by qualitative ones.

3. EVALUATED COMPONENTS

In the context of Use Case evaluation, the users access to, and experiment with different PHYSICS components through the Design Environment. The next paragraph provides a short description of the PHYSICS Design Environment and the components made available (see (Patiño, et al., 2021) for more details).

The PHYSICS Design Environment is the main entry point for the application developer when interacting with the PHYSICS platform. In this environment the latter needs to visually design and implement their application, by creating new code segments, importing existing ones, or re-using generic, available implementations (in the form of patterns) available from the PHYSICS platform. A key element is the ability to dictate workflows of operations among these diverse components, that in the end will be implemented during runtime, so that different elements of the application can be deployed according to their envisioned operation (i.e., as microservices or as functions, or a combination of the two).

The overall architectural diagram of the WP3 entry point for the developer appears in the following figure (Figure 3). This also includes interactions with the semantic block (Application and Service Semantic Models, Inference Engine), the Design Patterns Repository, and the FaaS Platform Deployment.

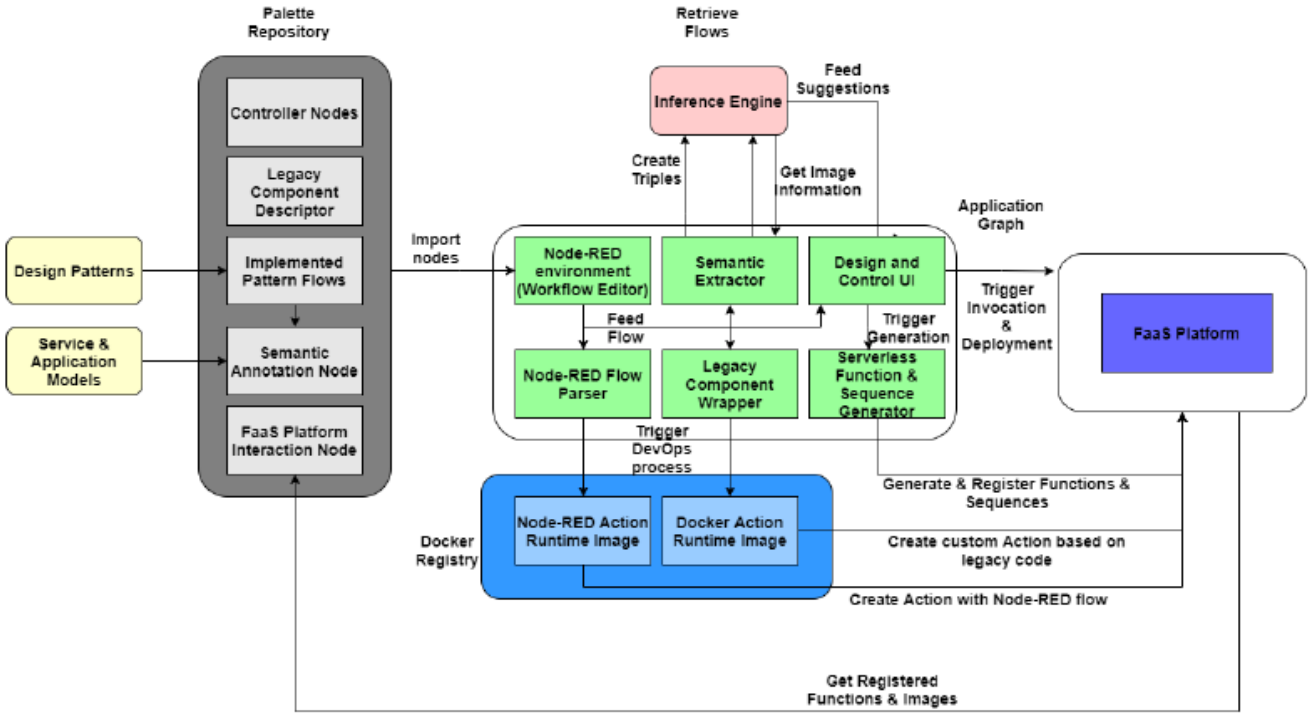


Figure 3: Design Environment Components and Interactions with other elements of the PHYSICS platform.

In the Smart Agriculture Use Case, the main functionalities tested during the evaluation phase are:

- Integration of Python codes in existing flows, namely the Edge-ETL flow (see (Patiño, et al., 2021), (Fatouros, et al., 2023) and (Kousiouris, et al., 2023) for more details)
- Parametrization of the flow using the Node-Red insterface
- Adaptation, build and deployment of the docker image in which the pipeline will be run on the Edge

In the eHealth Use Case, the main functionalities tested during the evaluation phase are:

- Design of flows describing the application with Node-Red
- Integration of Python codes in existing flows, namely the branchJoin flow (see (Patiño, et al., 2021) for more details)
- Execution of the application as FaaS through the deployment of flows using the DE

In the Smart Manufacturing Use Case, the main functionalities tested during the evaluation phase are:

- Design of flows describing the application with Node-Red
- Integration of Python codes in designed flows
- Execution of the application as FaaS using OpenWhisk
- Testing of design environment and the build functionality with the quality control (QC) flow

4. PILOT "SMART MANUFACTURING"

4.1 KPI definition

4.1.1 Description of key performance indicators

To evaluate the improvements after deployment of the PHYSICS platform on the Smart Manufacturing Use Case, the following KPIs were defined:

- **KPI01 - Time for handling a request**: Measures the average request/response time for 8 requests with PHYSICS and before PHYSICS.
- **KPI02 - Scalability**: Measures request/response time for 8 parallel requests.
- **KPI03 - Number of software logs per QA'ed product**: Counts the number of errors with and without failover functionality of PHYSICS while using QC function.
- **KPI04 - Reliability:** Measures the availability of the QC function and reliability of responses.
- **KPI05 - Interference latency:** Same as KPI01, but only calculates the time to complete the QC action, without considering the overhead of OpenWhisk.
- **KPI06 - Data protection:** Indicates how the transferred data is protected.
- **KPI07 - Number of PHYSICS invocations in software logs per QA'ed product**: Similar to KPI03, but this measures the ratio between the invocations on PHYSICS and the local QC service.
- **KPI08 - Cost reduction:** The bill based on the `time` multiplied by the `cost/time`.
- **KPI09 - Performance benefits:** Although for the manufacturing Use Case, the reliability (KPI04) is more important than performance benefits, this KPI will inspire us to further use FaaS technology in our upcoming demonstrators as well as the functionalities of the existing demonstrators.

4.1.2 KPI Workshop outcomes

From the complete list of the KPIs written above, during the workshop for the KPI definition process, KPI03 and KPI07 were selected as the highest priority for the Smart Manufacturing Use Cases. They were assessed in this deliverable.

Using the methodology from §2.2, in Step #1, the actors within DFKI (and its test lab) were identified by answering the following questions:

- Who is to work with the KPI?
  - Software Developer
  - Maintenance User
  - Production Planner
- Whose work is monitored with this KPI?
  - Software Developer
  - Maintenance User
- To whom is the KPI reported?
  - Production Planner
  - Superiors
  - Management

In Step #2, the following business goals (BGs) have been defined:

1. High production rate
2. High availability (of the software infrastructure)
3. Less maintenance time (in software failure)

The high availability (BG2) was chosen as the main goal, as it is crucial for the test lab. If the BG2 is met, the BG1 and the BG3 will also be affected indirectly, thus meeting these goals as well.

In Step #3, the prioritized goals were formulated as measurable performance results. This step enabled to identify two of them:

- Using PHYSICS should decrease occurrences of software faults by at least 40% compared to without PHYSICS.
- Using PHYSICS should decrease manual QA requests by at least 40% compared to without PHYSICS.

Since the performance results defined in Step #3 cannot be measured directly, in Step #4 they were reformulated as the following KPIs:

- # of error messages in software logs per QA'ed products (KPI03)
- # of PHYSICS invocations in software logs per QA'ed products (KPI07)

Step #5 dealt with how the measurements were supposed to be carried out. Based on the analysis, it was determined that the data was both calculated and directly measured by checking the error logs of the software. A daily check would also suffice. This KPI was first intended to be for Quality Control (service), however, it was also identified that it could be used for different usages.

Following Step #5, in Step #6, the optimal value was determined to be 0%, which is the ratio of error messages in software logs, divided by the number of total QA'd products (where # of QA'd products > 0). The desired value range was expected to be between 0% and 10%.

The last steps (namely Step #7 and Step #8) were to identify whether there was a (business)-process that the KPI should be integrated in. However, for the Smart Manufacturing Use Case, it was not the case. Nevertheless, it was confirmed that software developers and maintenance users can collect the logs and discuss the results with their superiors and management to evaluate their status. If necessary, this could have led to making decisions to improve the software.

4.2 KPI evaluation in the current infrastructure

The current infrastructure at SmartFactory-KL is implemented as a service-oriented architecture. As seen in Figure 4, each software component to complete the production is loosely decoupled from each other. This enables to convert the appropriate services into functions, to fully exploit the PHYSICS functionality. Each hardware module added to the demonstrator is also using the same idea of this software architecture. This means that the hardware modules are able to use the FaaS approach to fulfill their requests or respond to the requests from other hardware modules. This Use Case utilizes the QC Module, which controls the product for defects before giving it to the customer.

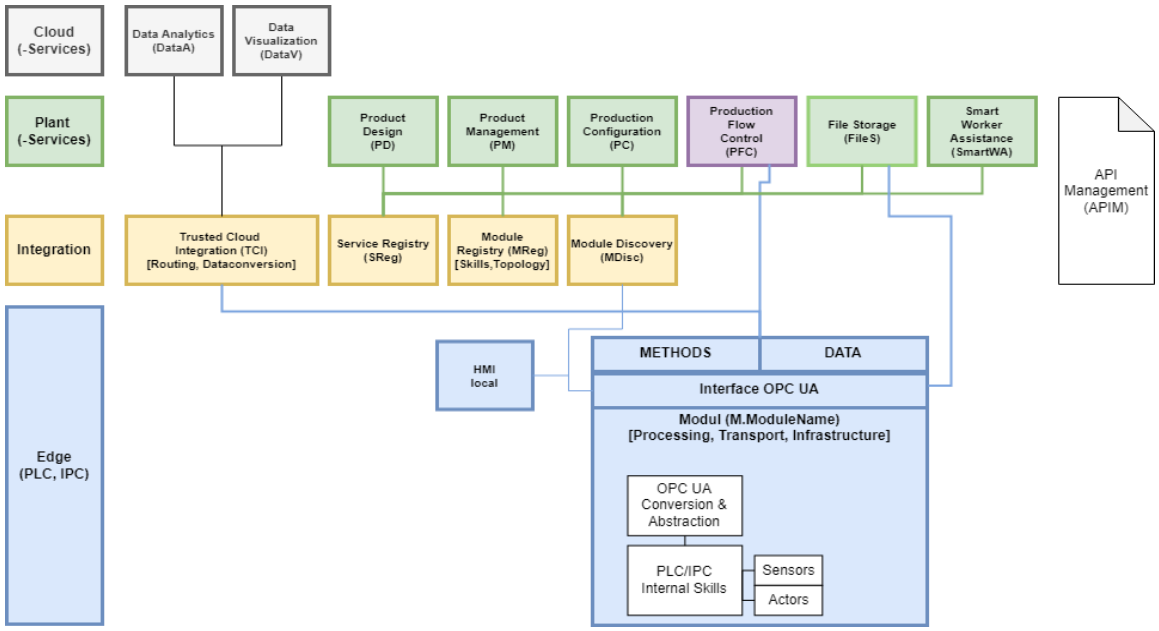


Figure 4: The architectural overview of the SmartFactory-KL demonstrator.

For the second version of the deliverable, the KPIs were evaluated by measuring the times using an internal tool. The baseline for this KPI was obtained using the following scenario:

- **S01)** Python script to request QC → Docker QC → Python in Docker (inference time) → Result

Table 1: KPI evaluation with remote docker (as-is, without PHYSICS components).

KPI	Objective	Current evaluation
KPI01	Median response time ≤ 1 seconds	0.383 s
KPI02	Not applicable, existing system was not built with scalability in mind	(3.032 s)
KPI05	Median interference time < 0.5 seconds	0.056 s

4.3 KPI evaluation using PHYSICS components.

For the second version of the deliverable, the KPIs were evaluated by measuring the times using an internal tool. To measure these KPIs using PHYSICS components, the following scenarios were identified:

- **S02)** Python script → PHYSICS **local Edge**-deployed QC flow with 2 docker QC functions → Result
- **S03)** Python script → PHYSICS **Cloud**-deployed QC flow with 2 docker QC functions → Result

S02 uses the local PHYSICS deployment on the Edge within DFKI premises, while S03 uses the Cloud PHYSICS deployment. While the first version of this deliverable explored the cold- and warm-start behavior of PHYSICS, for the second version, we focus in different aspects such as overall performance and availability in line with the much larger timeframe of the measurements.

We carried out the **3-week tests** (from September 4 to 23, 2023) of the PHYSICS platform to assess the goals targeted by the Smart Manufacturing Use Cases (see Deliverable 6.4 for complete explanation):

1. Deployment of substitute service in the Cloud
2. High Confidence Quality Control

The evaluation phase was held **from 8:00 a.m. to 8:00 p.m., Monday through Saturday**. The test consisted of calling the High Confidence Quality Control PHYSICS Cloud deployed function, resulting in approximately **20000 requests per week**. This gave the total number of 60177 requests with 59425 **successful responses (98.75%)**, even though PHYSICS was still in beta testing. Figure 5 shows the complete timeframe for the evaluation phase and all total and action times for each request. The main outliers at the beginning of each new day are caused by the required cold-starts of the functions.

Only one major problem with the PHYSICS platform occurred on September 4th which, once noticed, was fixed relatively quickly early next day. This outage is heavily reflected in the weekly results where the successful response rate was 96.47% in the first week. The remaining weeks were problem free, with 99.82% and 99.96% successful response rate for the second and third week respectively. All in all, the availability of the PHYSICS Cloud deployment meets the Smart Manufacturing objectives regarding KPI04.

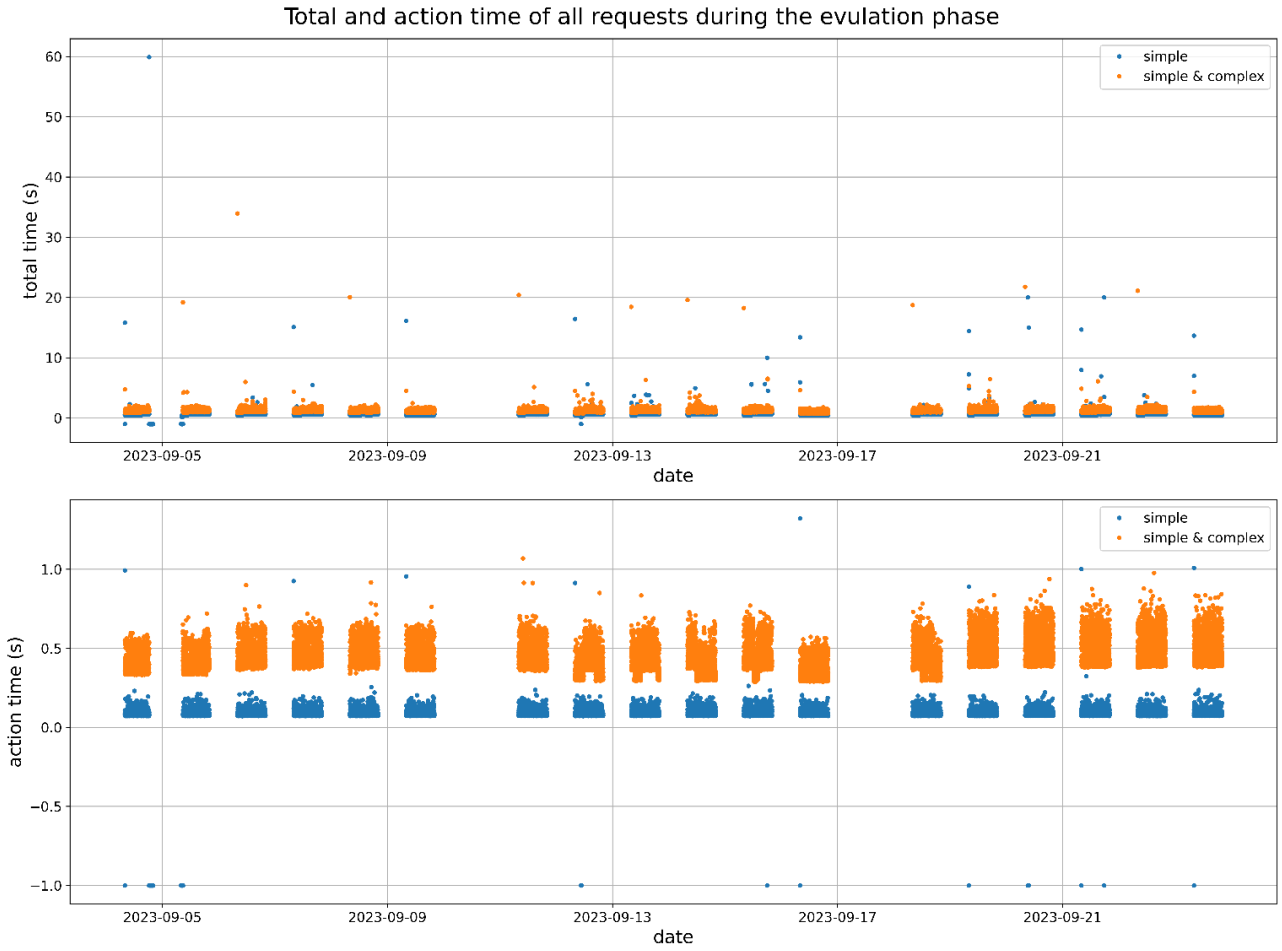


Figure 5: Total and action time of all requests during the evaluation phase (2023-09-04 till 2023-09-23). A time of -1 indicates a problem (either related to the PHYSICS platform or the connection to it).

The histogram in Figure 6 depicts all successful request total response time during the evaluation phase. While requests requiring the more complex QC function take slightly longer than the objective of less or equal one second per request for KPI01, it is only used in 43.33% cases (KPI07), hence the median total response time is 0.747 seconds which meets the objective.

Figure 7 depicts a typical day of requests during the evaluation phase, except for a short period of continuously failed requests. This was caused by a load generator demonstration (another feature of the PHYSICS Design Environment), that overwhelmed the system during its runtime. The load generator is meant to generate as much load as possible, however it had the unintended consequence of interrupting every other request. This however is to be expected, since the Cloud infrastructure that PHYSICS is running on during the project runtime is limited.

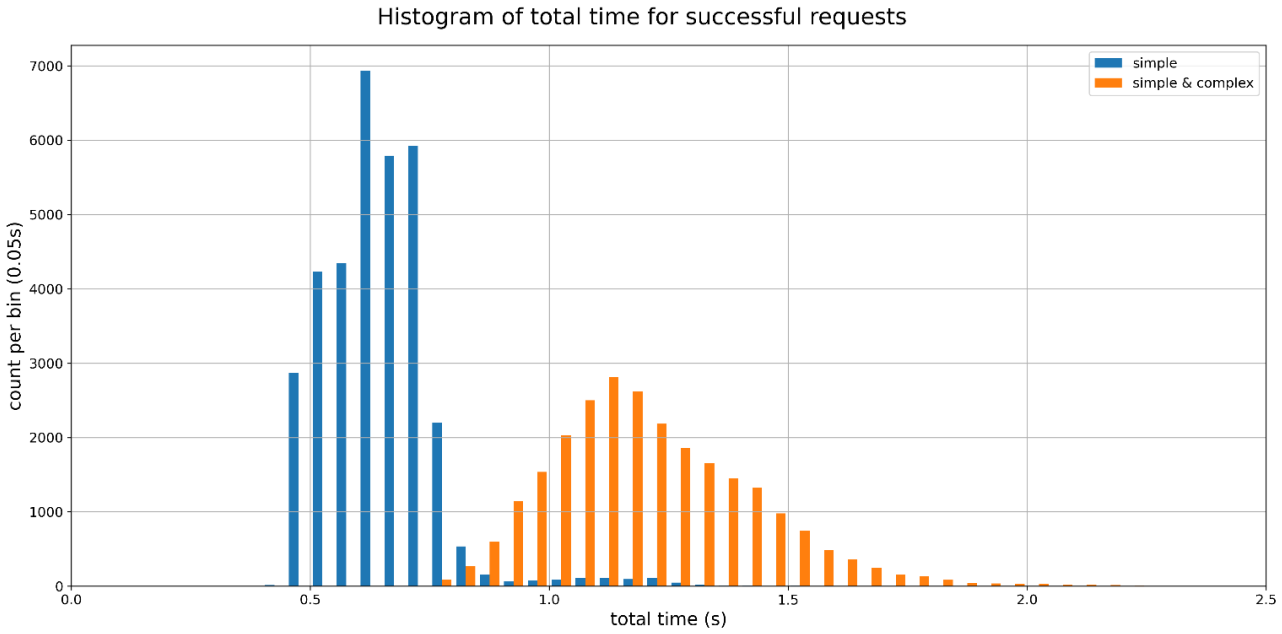


Figure 6: Histogram of total time for successful requests during the evaluation phase.

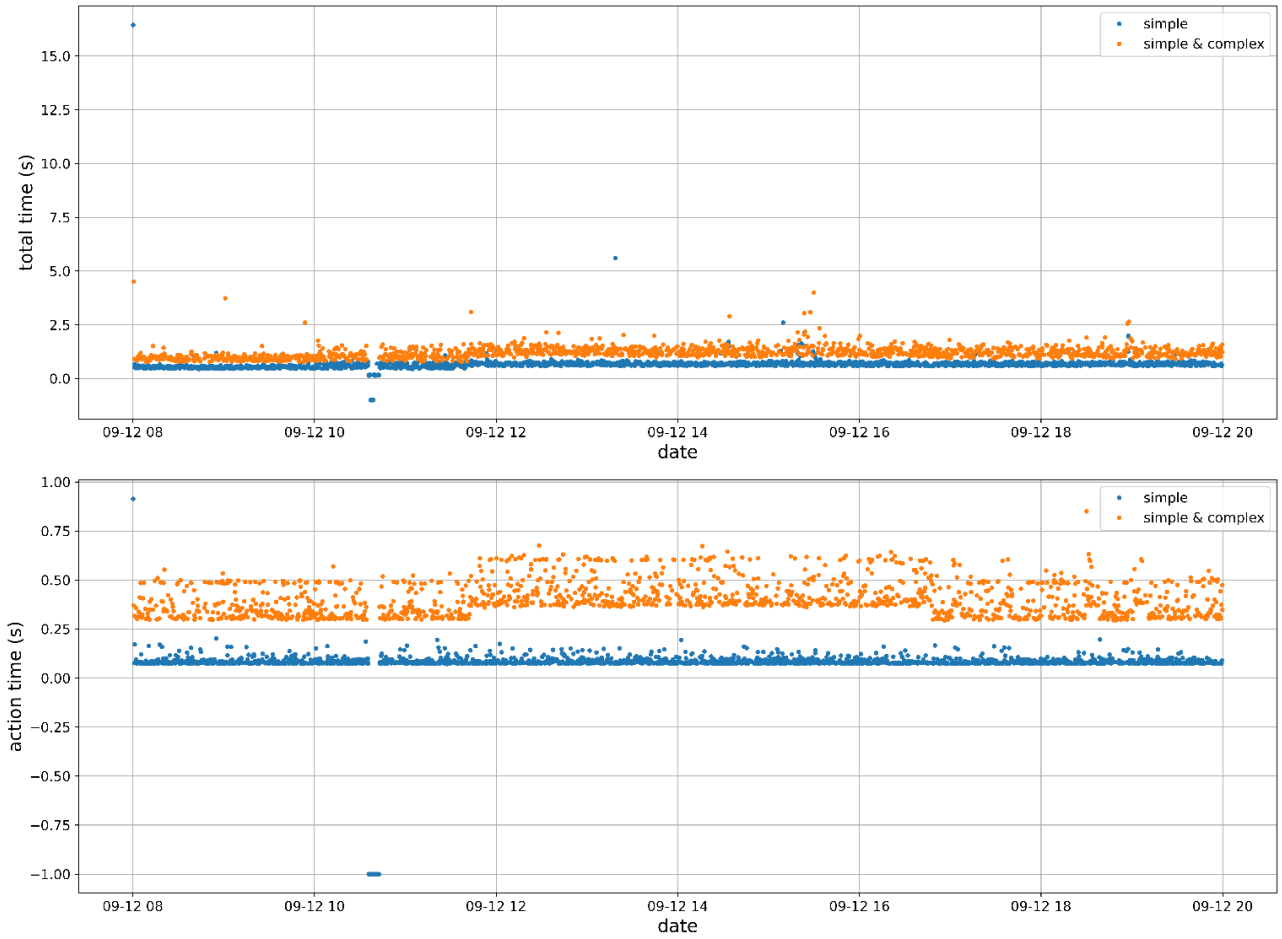


Figure 7: Outlier Day because of "Load-Generation" demonstration

Regarding KPI03, during the third week, our local Edge PHYSICS deployment had a successful response rate of 99.88% (23 errors). With the Failover to the Cloud deployment, the actual successful response rate increased to 99.96% (7 errors).

Table 2: Smart manufacturing KPI evaluation using PHYSICS components. All evaluations done in "warm" state with worst case (both simple and complex QC functions are used).

KPI	Objective	Evaluation context	Current evaluation
KPI01	Median response time ≤ 1 second	S02	0.575 s
		S03	0.747 s (3rd week)
KPI02	Total time (parallel with 8 requests) < Total time (sequential)	S03 (parallel)	11.803 s
		S03 (sequential)	8 * 0.747 s = 5.976 s
KPI03	# of errors as close to 0 as possible	Without Failover	23 (3rd week)
		With Failover	7 (3rd week)
KPI04	Availability > 99%	S02	99.88% (3rd week)
		S03	99.96% (3rd week)
KPI05	Median interference time < 0.5 seconds	S02	0.244 s
		S03	0.451 s
KPI06	Keep data secure	S02	Data never leaves company network
		S03	Data transfer is encrypted, and data is not stored.
KPI07	Minimize manual QC by Operators, more PHYSICS invocations mean less manual QC.	# of PHYSICS invocations	8663 (3rd week)
		Ratio	43.33% (3rd week)
KPI08	Minimize running costs using PHYSICS Cloud	Worst case (always)	~3.35 USD
		Average case (just failover)	~0 USD
KPI09	Performance Benefits when the certainty of the simple QC is insufficient	With PHYSICS	4232 s total (3rd week)
		Manual inspections[3]	10s * 8663 = 86630 s
		Improvement[4]	20.47x

In the as-is (non-PHYSICS) scenario, the local deployment on a dedicated virtual machine was running 24/7, causing significant power consumption and needing extra maintenance, totaling cost of roughly 60€ per month. Without help from experienced DevOps engineers, this deployment required nine person-days.

With the PHYSICS infrastructure, since the machines used for production are not running 24/7, there is a potential for cost savings. The locally deployed PHYSICS on the Edge is running on a shared server and is only insignificantly contributing to power consumption while not in use and incurs only minimal additional

[3] Manual inspections – it is assumed that the manual inspection takes on average 10s.
[4] Improvement – theoretical improvement based on the assumption regarding manual inspection.

maintenance costs equal to approximately 10€. At the time of the experiment, the platform was still undergoing fine-tuning, necessitating assistance from other project partners to set up the developing and testing environment locally. Consequently, it's challenging to precisely determine the amount of time required to deploy PHYSICS at the Edge. However, starting from scratch, the installation of PHYSICS Design Environment + stand-alone OpenWhisk (minimal setup of PHYSICS) is estimated to take less than one day for an expert, especially with the help of available automation scripts.

The PHYSICS Cloud deployment only incurs costs while in use (included in Table 2). Since no commercial pricing of PHYSICS is available yet, the cost of using FaaS was estimated using competitor AWS Lambda in Frankfurt (Europe). Thus, in the Smart Manufacturing case with approximately 20000 requests per week taking in the worst case 5s each total to roughly $3.35 (3.08€) monthly. Deployment of the Smart Manufacturing functions to PHYSICS Cloud was done in roughly two person-days. This process was considerably easier than deploying it locally, especially since previous experience gained from the local deployment was applicable. Estimated time is less than 2 hours including the import of already existing functions from the previous Edge deployment.

Regarding data security, the PHYSICS Cloud offers encrypted data transfer, and no data is stored in the functions because they are stateless by design (unless the user wants to implement it using the PHYSICS component). Additionally, if the user has the requirement of data not leaving the company network, it is also possible to deploy and use PHYSICS locally and separated from the Internet.

4.4 Conclusion

As discussed in the previous section, the availability of implemented Smart Manufacturing functions is the highest priority. PHYSICS achieved satisfactory results both in the Edge Deployment case as well as in the combined Edge and Cloud fail-over case, with each having roughly 99.9% availability during the evaluation phase.

Although the performance of the PHYSICS Cloud environment (S03) was found to be slightly slower than the local deployment (S02), as defined by the Smart Manufacturing Use Cases, the reliability and availability is more important than the performance, as long as downtime is minimized, and it is still high enough not to disturb the production. However, the performance results are expected to be enhanced if PHYSICS is used commercially, since currently it is restricted by running on limited Cloud resources due to minimizing cost.

PHYSICS decreases both deployment and running costs compared to using a dedicated virtual machine. The PHYSICS Edge deployment reduced costs from approximately 60€ to roughly 10€. Additionally, utilizing the PHYSICS Cloud deployment as a fail-over incurs basically no costs on average. Only in the worst-case, estimated costs are around 3€ if every request is handled in the Cloud.

Finally, the PHYSICS environment helps reduce the number of tedious and time-consuming tasks carried out by the test lab worker. While similar performance benefits could be achieved without using PHYSICS, the development and operational costs would be significantly higher.

5. PILOT "EHEALTH"

5.1 KPI definition

In the following subsections we describe the process of deriving KPIs for the eHealth Use Case, resulting in a Table of measurable KPIs at the end of this subsection.

5.1.1 Description of key performance indicators

The starting point for defining the eHealth Use Case KPIs were the following two performance measures that were deemed relevant for the pilot at the start of this process:

- The **time to infer** on some input vectors given a model should scale better than linearly with the number of involved vectors.
- The **time to redeploy** the service utilizing more resources should be short and the process should not require human supervision.

5.1.2 KPI Workshop outcomes

Starting the KPI definition process as defined in §2.2 (Step #1), we have identified the following set of key actors involved in these two KPIs:

- *Who is to work with the KPI?*
  - Head of Machine Learning
  - Machine Learning Engineer
  - Head of Software Development Team (Service Manager)
- *Whose (direct) work is monitored with this KPI?*
  - Head of Machine Learning
  - Machine Learning Engineer
- *To whom is the KPI reported?*
  - CEO

The actors and their roles as defined above are the same for all relevant KPI's related to the PHYSICS eHealth Use Case.

As a next step (Step #2), the question of which business goals should be supported was defined as follows. Note that from this point on in the KPI-definition workshop, the original KPI definitions (i.e., "time to infer", and "time to deploy") were let go in order to formulate a set of more exact KPIs as defined here. At the end of this section in Table 3, we re-visit these initial statements in order to re-introduce them in the final set of KPIs for the eHealth Use Case.

- **BG1:** "***Platform works in a stable way***" – A very important high level KPI is that the addition of smart services, such as the online inferencing should never negatively impact the overall system performance. Additionally, for this business goal, the following daily work goals were defined:
  - Prediction pipeline should be optimized
  - Improve the prediction time
- **BG2:** "***Add the machine learning features to platform***" – Perhaps obvious, but it is important that the ML features (online inferencing, and later perhaps online model training) are integrated into the platform. Additionally, for this business goal, the following daily work goal was defined:
  - Improve the prediction accuracy

- **BG3:** "***Lower the price of an individual prediction***" – Looking ahead, as the number of users of the platform grows, performance of the online inferencing cannot become an excessive cost element of the platform, therefore the price ($) of individual predictions should be kept to a minimum. Eventually this business goal was considered secondary and was not worked out in further detail.

From these three, BG1 and BG2 were selected for further clarification, and both were marked as "Optimization" KPIs on the scale of (*Optimization – Orientation – Motivation*).

In the next step (Step #3), both KPI phrasings were clarified as follows:

- **KPI-1: Mobile/Webportal Application Requests are handled in 99% of the time within 1 second**
- **KPI-2: Add automatic prediction of configurable health outcomes as a platform feature**

Since both KPIs are already considered very well measurable, Step #4 didn't yield any new results. The final four steps (Step #5 – Step #8) have been followed for each of these 2 candidate KPIs, resulting in the following information:

KPI-1: Mobile/Web portal Application Requests are handled in 99% of the time within 1 second

- *Step #5: How data is collected:*
  - Calculated
- *How often data needs to be updated:*
  - Daily
- *What are the parts of the KPI:*
  - Every Page Load Time (automatically collected)
  - Statistics of Page Load Times (automatically collected)
- *What it means (Step #6):*
  - Ideally, the amount of page load times <1s / total amount of page loads is somewhere between 0.9 and 0.99.
- *Step #7: In which (business-)process should the KPI be integrated:*
  - Integrate into the OKR process handled through JIRA[5].
- *What has to be done therefore:*
  - Automatic calculation has to be implemented
  - Automatic measurement of load times
  - Automatic reporting of statistics to an e-mail inbox
  - Manually updating JIRA
- *Which processes have to be created to be able to use the KPI:*
  - None.
- *Who can do that:*
  - Head of SW Development updates the JIRA Board.
- *Step #8: Who uses the KPI to make daily work better:*

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

    - Head of SW Development
- *Who uses the KPI to make reports:*
    - Head of SW Development
- *Whose salary/compensation is tied to the KPI:*
    - Head of SW Development

KPI-2: Add automatic prediction of configurable health outcomes as a platform feature

- *Step #5: How data is collected:*
    - Direct measured
- *How often data needs to be updated:*
    - Weekly
- *What are the parts of the KPI:*
    - Presence of the Feature (manually collected)
- *What it means (Step #6):* There is relevant progress on any of the implementation's subtasks.
- *Step #7: In which (business-)process should the KPI be integrated:*
    - Software Development and Planning
- *What has to be done therefore:*
    - Implement the reporting in the JIRA Agile Tool
    - Talk with participants about the purpose of the KPI
- *Which processes have to be created to be able to use the KPI:*
    - None.
- *Who can do that:*
    - Software Development Lead
    - Product Owner
    - Software Developer
- *Step #8: Who uses the KPI to make daily work better:*
    - Head of Machine Learning
- *Who uses the KPI to make reports:*
    - Head of Machine Learning
- *Whose salary/compensation is tied to the KPI:*
    - Head of Machine Learning
    - Head of SW Development

5.1.3 Building measurable performance results

This defines KPI-1 and KPI-2 as they were defined in the collaborative KPI Workshop held in January 2022. The current set of KPIs for the eHealth Use Case is summarized in Table 3 below.

Table 3: List of KPIs for the eHealth Use Case.

#	KPI	Clarification
KPI-1	Platform requests are handled in 90% of the time within 1 second	Currently the requests arrive in bundles every couple of hours. It is important that these requests do not block the platform
KPI-2	Functionalities offered by the functions drive platform features	Have usable services out of the eHealth functions, that can be invoked by the platform
KPI-3	The average invocation time of using a FaaS should be the same or less compared to the traditional deployment	Cold and hot starts of the FaaS will be considered when presented with bursts of requests.
KPI-4	The time to re-deploy the service (update the code or a model) should be shorter compared to the traditional deployment.	The process of redeploying service code or models should be simplified ty PHYSICS.
KPI-5	The use of PHYSICS should lead to cost reductions.	Cost is important for the uptake of PHYSICS. We consider both initial costs (learning PHYSICS) and running costs (hosting the services).

5.2 KPI evaluation in the current infrastructure

The current implementation of the Healthentia Smart Services at Azure exposes API endpoints that expect the same input as their new DaaS counterparts detailed in D6.6. The service is a Flask Python web application that also features a UI for monitoring purposes. The service does not have any storage needs. The input data and the results are fetched from and stored in the database of the Healthentia platform. It is dockerized and deployed as part of the Healthentia Kubernetes cluster in its own pod.

The service is typically invoked once per day for each subject in the Healthentia platform. The feature vectors are composed when the information for any day is complete. Since this can happen at any time in the following day, vector composition is attempted every two hours, whereupon some new vectors are prepared. These are combined in bundles of ten and are sent for inference.

The models are implemented using different machine learning algorithms, their majority being Neural Networks or Random Forests. Our current inference needs are met without the need of any GPU acceleration.

5.2.1 Handling requests

KPI-1 is fulfilled by the current traditional deployment. All requests result to starting the processing for inference well within the 1 second threshold.

5.2.2 Feature inclusion

KPI-2 is fulfilled by the current traditional deployment. The deployment is offering the wanted functionality via API endpoints.

5.2.3 Executing requests

KPI-3 is about comparing traditional deployment and FaaS, but the two approaches cannot be compared like that. FaaS introduces an inherent delay for bringing the service online. This is a cold start. Once online, a service remains so for some time after the completion of the request. In PHYSICS this is about 12 minutes. Any subsequent requests in that period are processed by the service in hot start. A hot-started service has the same execution time on the same hardware no matter if it is traditionally deployed or FaaS.

5.2.4 Deployment complexity

KPI-4 is about comparing the deployment of updated functionality in the current Healthentia platform to the needed steps when PHYSICS is used. In Healthentia, service code updates are handled by pushing the new code in the equivalent branch, whereupon the equivalent CI/CD pipeline builds the docker image and deploys it. Model updates are not allowed. Model additions are done by Healthentia's ML services: Any newly learnt model is stored in a data lake, while metadata about it are stored in Healthentia's database.

5.2.5 Cost

KPI-5 is about the initial costs and the hosting ones. Initial costs relate to learning how to set up deployment pipelines (in Jenkins) and actually putting them in place. A knowledgeable engineer can become a novice DevOps engineer in about 10 person days. A mid-level DevOps engineer can create a new pipeline of the complexity of the services at hand in less than 2 person days. Hosting costs for the service's pod are currently at around $80,00 per month.

5.3 KPI evaluation using PHYSICS components

The FaaS deployment is achieved using the PHYSICS DE to deploy services from Node-RED flows, as described in D6.6. For example, the flow for inference is shown in Figure 8, adapted from D6.6.

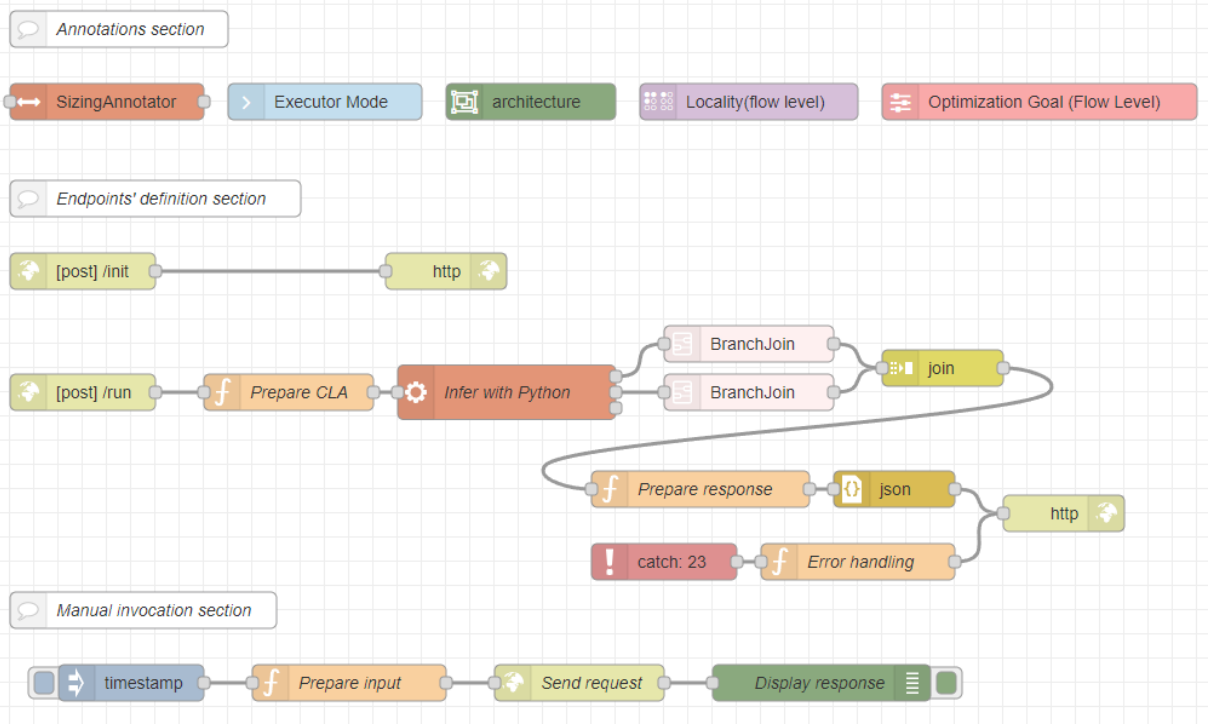


Figure 8: Flow for inference service.

Single requests are sent to the FaaS version of the services using another Node-RED flow, as shown in Figure 9, adapted from D6.6.

Multiple requests can be sent using the PHYSICS load generator. The flow for doing so is shown in Figure 11. This flow benchmarks the timing of the execution of the function, as shown in Figure 10.

5.3.1 Handling requests

For the evaluation of KPI-1, the load generation flow is called once, and the results are shown in Figure 12. The average start latency is far below the 1 sec limit, at 214ms. But this suffers from a large standard deviation of 485ms. While the start latency is clearly not Gaussian-distributed, assuming so leads to approximately 95% of the requests having started faster than 1 second, so the KPI is reached.

The results obtained from the PHYSICS Load Generator appear in Figure 11, based on the delay definitions of Figure 11. Memory is in MB, timestamps in Unix Time milliseconds, the set and achieved rate refer to messages per minute while all the delays are in milliseconds. In this case the target is the inference function on AWS.

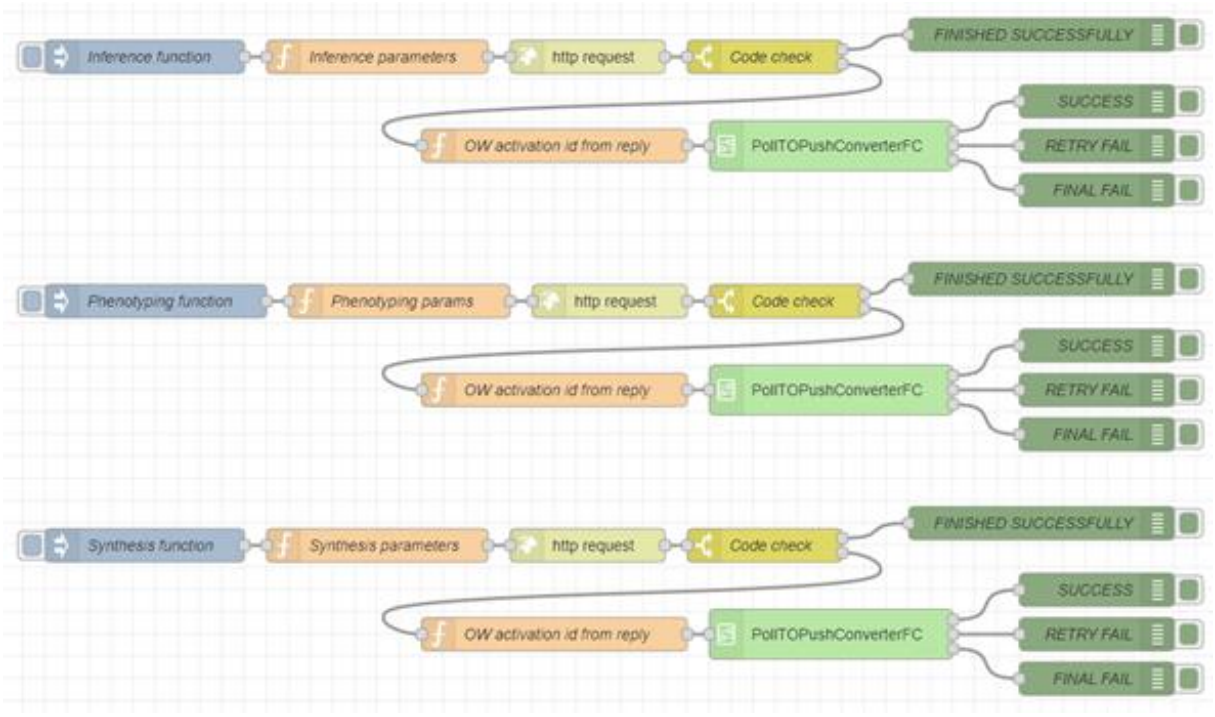


Figure 9: Flow utilizing the deployed functions.

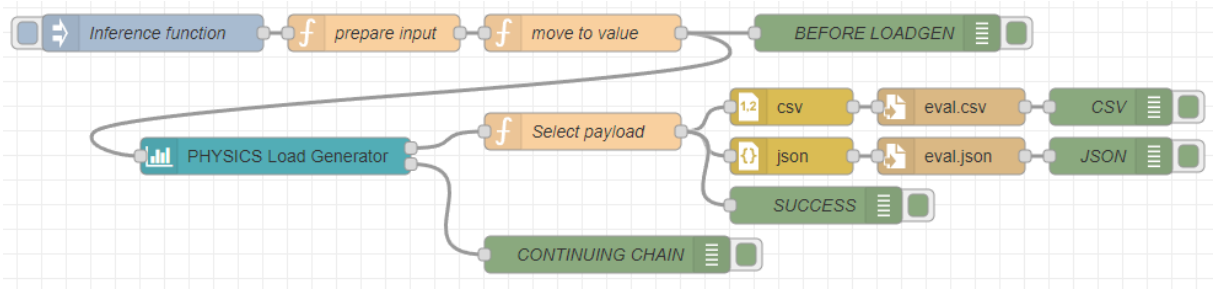


Figure 10: Flow for load generation.

5.3.2 Feature inclusion

KPI-2 is fulfilled by the SaaS deployment of the services, since it is offering the wanted functionality via API endpoints.

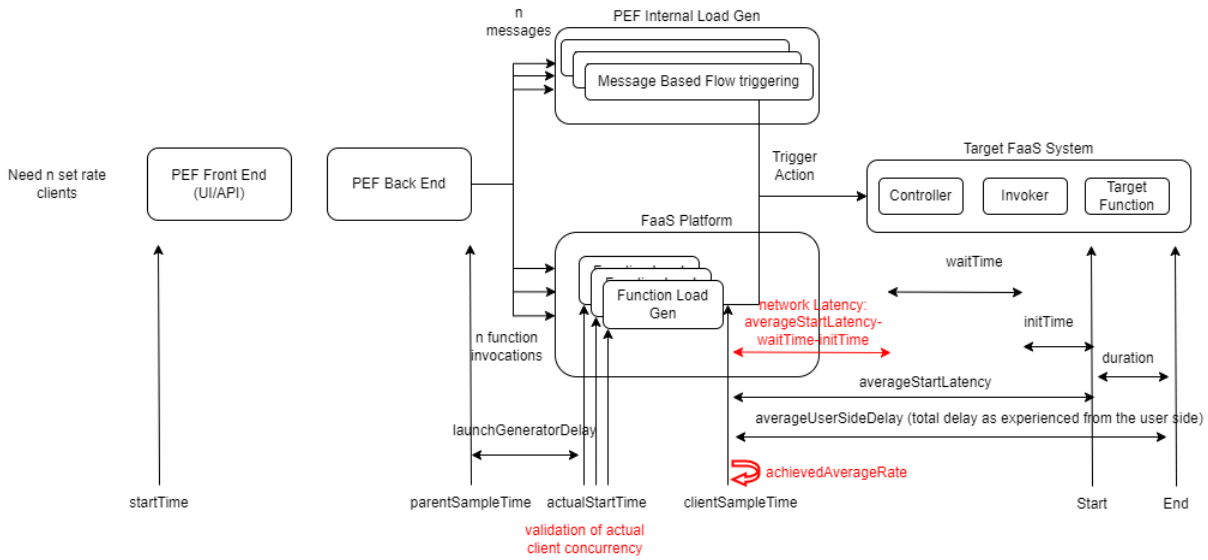


Figure 11: Load generation flow used for benchmarking the functions.

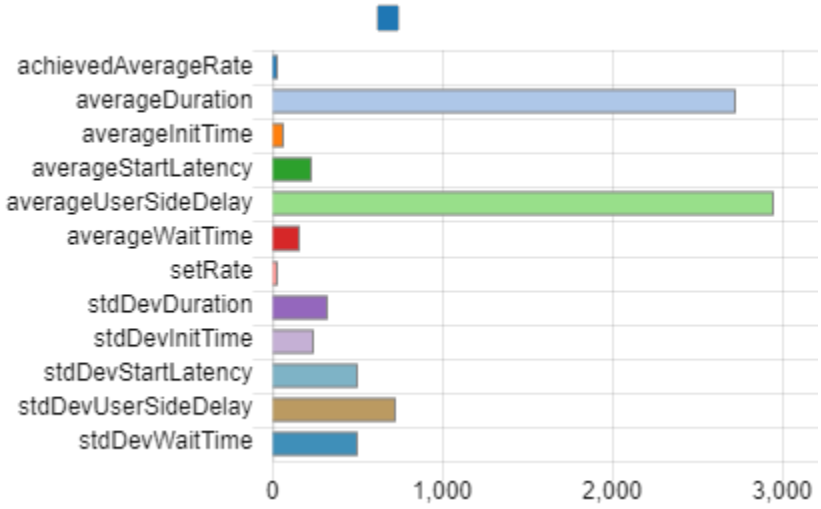


Figure 12: Load generation results.

5.3.3 Executing requests

More experiments have been carried out, with the parameters selected so that we always get the same warm container for execution, to investigate its behavior over time. This is shown in Figure 13. This can be used to decide for example if the memory used is ok, or if some other aspect of the image can be optimized. For example, in the following case the filesystem usage seems to increase over time, indicating that at some part of the flow relevant files may need to be deleted.

In order to compare between a typical service-based deployment and a FaaS based one, the created model inference container was deployed as a service container on a 4-core 8GB VM (internal testbed). On the same platform, an Openwhisk installation was used with the registered function. Then three different set rates were applied (12, 30 and 60 messages per minute), each with 10 inputs needing prediction, to investigate the effect in each case. 100 values from each case were approximately obtained. The results appear in Table 4. One of the main benefits achieved is the increased scalability of the FaaS solution, which is able to maintain responsiveness at the highest rate, in contrast to the server-based version.

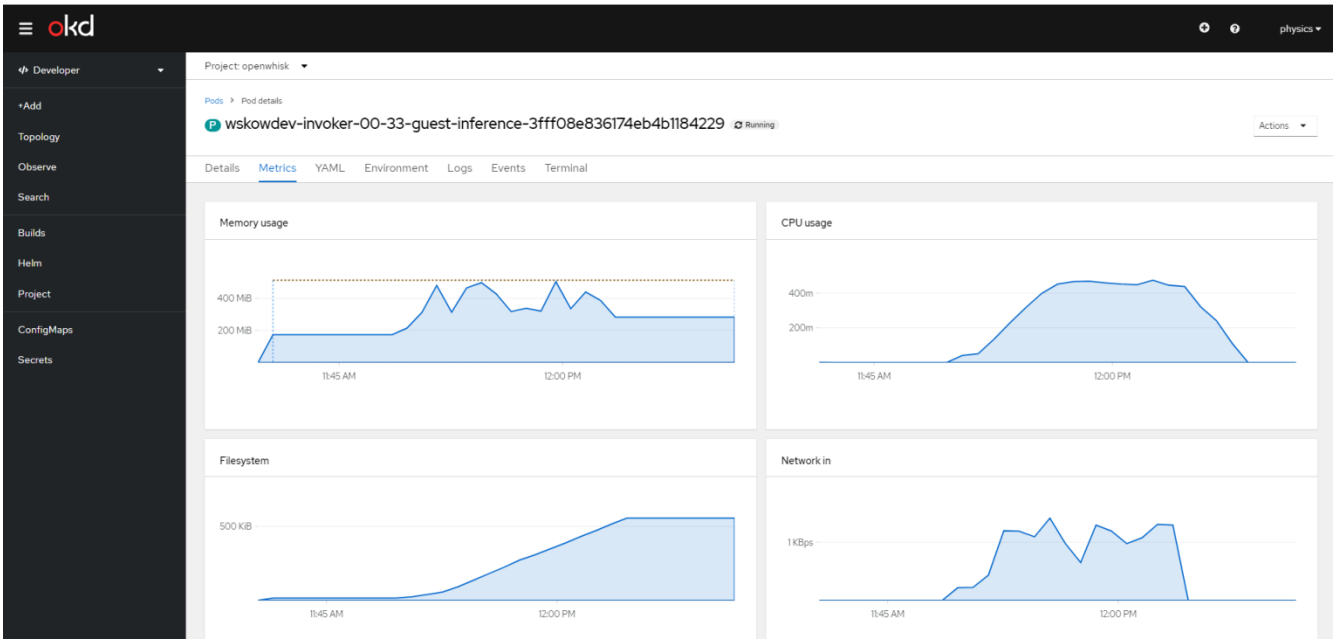


Figure 13: Example container usage metrics from the OKD dashboard

Table 4: Comparative results between a service-based version and a FaaS-based version on a local testbed

Type of Execution-Rate	Average User Side Delay (ms)
Server-12	6309.56
Server-30	6608.54
Server-60	Timeout, server crashing
FaaS-12	6646.71
FaaS-30	21940.93
FaaS-60	90766.40

The evaluation proceeds looking into the average rate of processed requests per minute as a function of delay between requests, considering both cold and hot start conditions. The average time to process each request is 2.6 sec, so delays more than that result to a rate of processed requests that is equal to the incoming rate, matching that of an infinite system (one of infinite resources). This is shown in Figure 14. For delays smaller than 2 sec, the performance of the system deviates from the infinite one, with the rate of processed requests dropping below the expected. There, the performance of the system under hot and cold start deviates. The system under hot start still manages to remain close to the infinite system down to 1 sec of delay, i.e., about three requests being processed simultaneously. The cold start system already deviates a lot, but also fails to infer, even for requests that do arrive for processing (see the dotted lines indicating successfully inferred for messages). Performance collapses completely for an intra-request delay of 750ms for the cold start, and 500ms for the hot start.

To alleviate the problem, the PHYSICS Request Aggregator can be used. It collects multiple requests into a larger one, actually receiving larger inference requests more sparsely in time. It is evident from Figure 14 that at an original request rate of 1 per second, problems start even for the hot system. Using the PHYSICS Request Aggregator we can manage much larger original request rates, without any unprocessed or failed requests. This is shown in Table 5. Note that for all these scenarios the processing delay for the original requests is increased or average, but there also is a large standard deviation. This is expected, since when using the aggregator, requests simply wait for the buffer to fill, are packed together, are sent for processing, the results are unpacked and populate the respective responses. Requests early in the buffer suffer longer delays as they wait for the buffer to fill in. The histogram of the processing delay for all scenarios in Table 5 are shown in Figure 15.

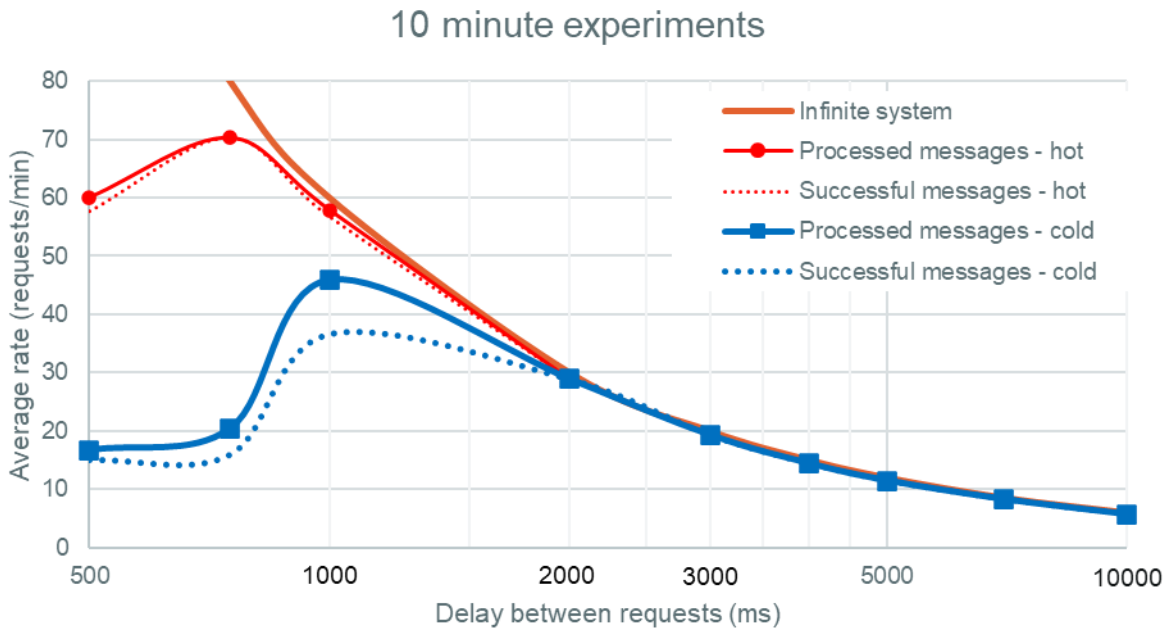


Figure 14: Average rate of processed requests as a function of the delay between the incoming requests.

Table 5: Request Aggregator scenarios for handling request rates of 1 Hz and above with different batch sizes aggregated together.

Rate (Hz)	Batch size	Duration (ms)	
		Mean	STD
1	2	7,135	4,429
2	2	18,315	6,830
2	10	6,758	1,335
5	10	17,668	3,707

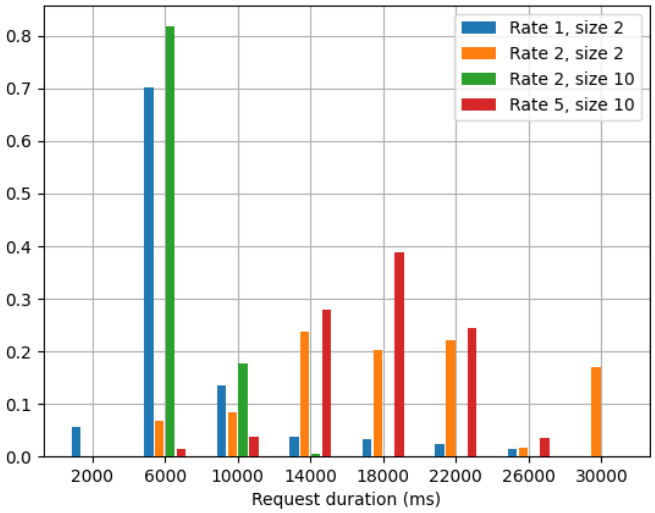


Figure 15: Histogram of request durations when aggregated under the four different scenarios of Table XX.

5.3.4 Deployment complexity

KPI-4 is investigates updating the services, either their code or their underlying models. For the eHealth pilot both require re-building the Docker image that contains both the Python scripts and the model files. The image is built by Jenkins automatically by the PHYSICS design environment. Placing the models needs to be done manually in the model directory to be used for building the image, so this does not benefit from the automation already in place in the company.

Nonetheless, KPI-4 is achieved, since the update complexity is really similar.

5.3.5 Cost

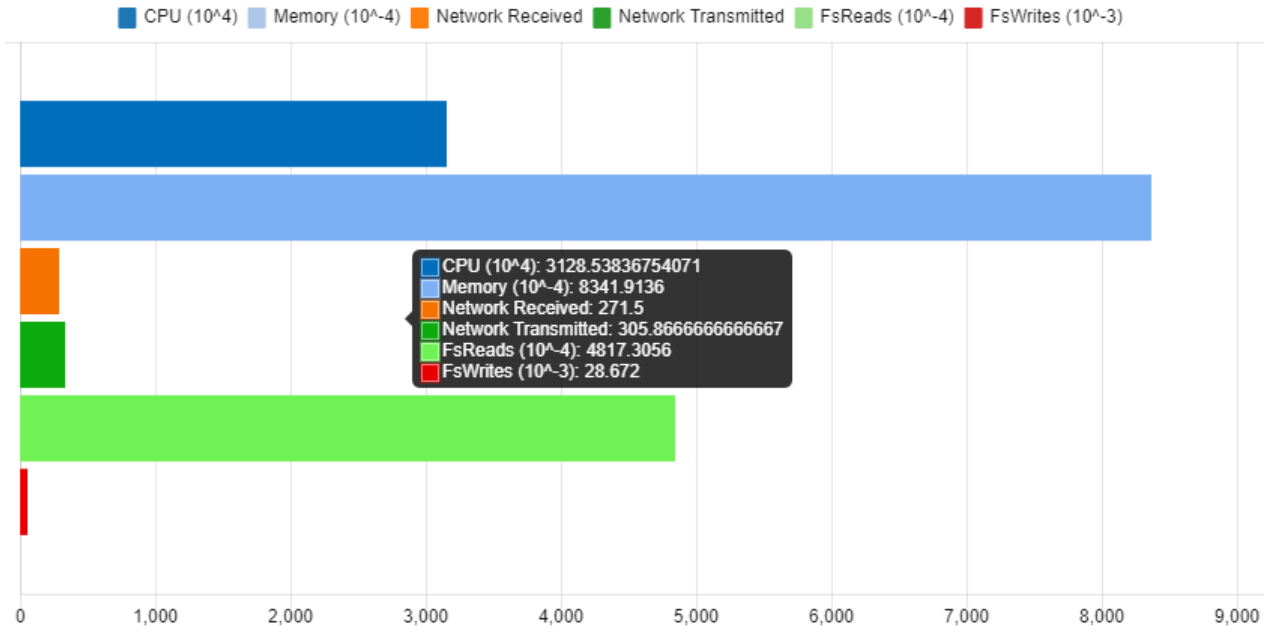
As already discussed, the initial and the hosting costs are considered. The initial cost includes learning to deploy the PHYSICS way, i.e., learning the Design Environment. It is difficult to estimate the time needed for learning the DE for the eHealth pilot engineers, since the process was progressive, and the DE was under development. Focusing on the online version of the DE, an engineer knowledgeable in Docker and Node-RED, was able to deploy simple services in a day.

Regarding the hosting costs, we consider deployment in AWS, located @ Europe/Frankfurt. Given the bursty nature of the requests, each needing 2.6 sec to execute for ten subjects, and the relaxed time requirements - each subject requires one inference per day, the cost of the FaaS based version is $1 per month for the current needs of 1.5k patients, or $6.8 for a future need of 10k patients. The recurring hosting costs in the FaaS case are an order of magnitude lower than that of the server-based deployment.

Obviously, KPI-5 is easily met by FaaS deployment using PHYSICS.

5.4 Service resources

The required resources for running the eHealth services are also considered. Focus is placed in comparing one of the inference scenarios (the one for prediction) and the synthesis scenario. The needs of the services implementing the two scenarios are compared in Figure 16. The significant differences are in information transmitted over the network (synthesis has much larger outcome volume than inference) and in filesystem reads (models are read from the filesystem, and the inference random forest model is much larger than the Gaussian Mixture Models used by synthesis).



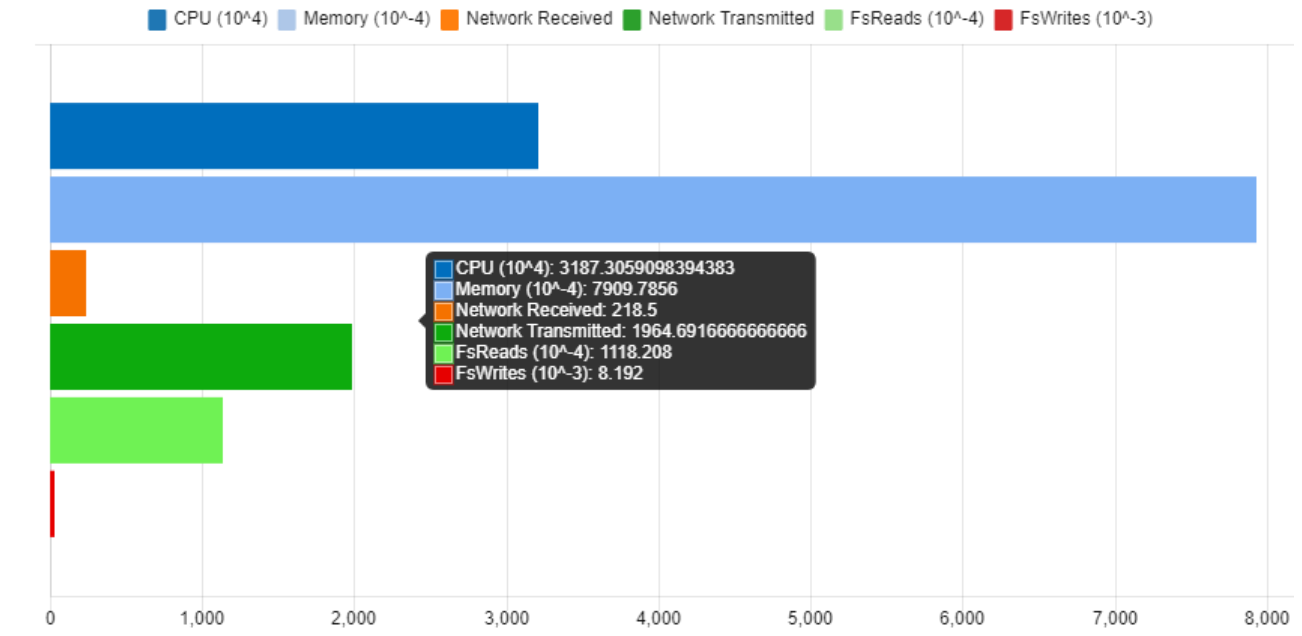


Figure 16: Resource needs for the inference (top) and the synthesis (bottom) scenarios of the eHealth pilot.

Finally, the needs of the two eHealth services are compared to that of every other service deployed by PHYSICS. The result is shown in Figure 17. Both eHealth services are of the most demanding in terms of CPU and memory, while they are rather low in terms of filesystem writes and input information received via the network. They differ in filesystem reads, where inference is very demanding, but synthesis is rather low, and in transmitted information over the network, where synthesis is very demanding, but inference is just above average.

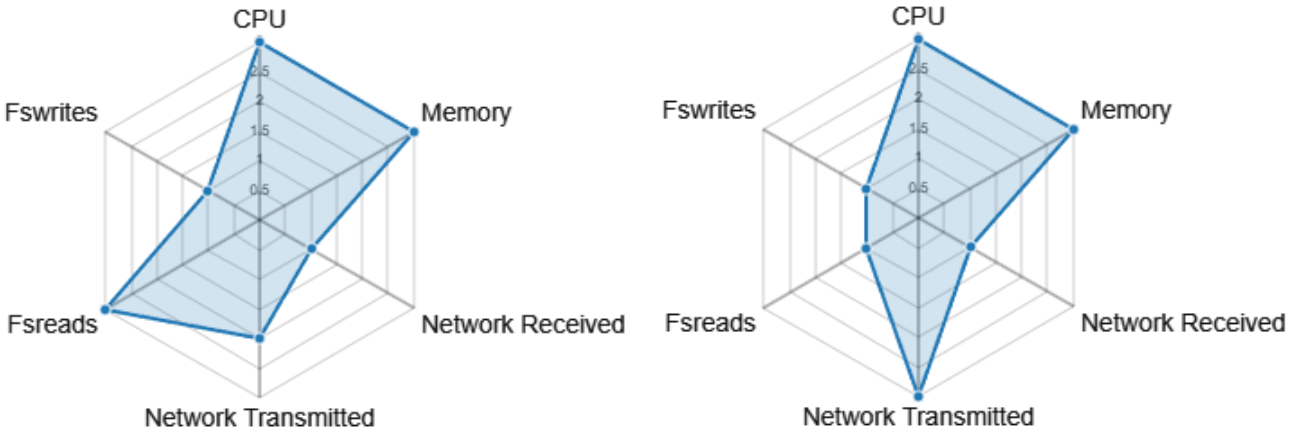


Figure 17: Comparison of the resource needs of the eHealth scenarios (inference on the left, synthesis on the right) to those of all other services deployed using PHYSICS.

6. PILOT "SMART AGRICULTURE"

6.1 KPI definition

6.1.1 Description of key performance indicators

Several objectives have been defined to evaluate the performance improvements achieved through the PHYSICS platform and project products (Franke, et al., 2021). For each one of the objectives mentioned, we give a brief description:

- **Amount of data lost**: the data collection pipeline currently deployed in greenhouses is highly sensitive to connection failures. When connection failures happen, the data measured by the sensors and exposed by the supervisor of the greenhouse are lost. The aim of this indicator is to evaluate the gain in robustness induced by the integration of PHYSICS components.
- **Update on plant status:** the data collected in the greenhouse are used as input of the numerical plant growth model. The outputs of these models are provided to the growers through visualization tools allowing them to track hourly changes in the plant status and to support decision making in greenhouse management. In case of connection failure to the supervisor, environmental data are unavailable, and plant status cannot be updated, leading to service interruption.
- **Time to perform optimization**: numerical plant growth models coupled with climate scenarios allow to explore different greenhouse management policies. To be relevant, the numerical simulation of the scenarios must be run with up-to-date environmental data and the response time of the system must be as short as possible.
- **Adaptation time to a new context**: according to the greenhouse infrastructure on which the pipeline must be deployed, several adjustments could be necessary.

6.1.2 KPI Workshop outcomes

From the full list of the objectives written above, during our workshop for the KPI definition process, we have selected "Adaptation time to new context" as our highest priority during the PHYSICS project. To reflect the methodology in §2.2, as Step #1, here are the identified actors:

- Who is to work with the KPI?
  - Commercial
  - Head of greenhouse project
  - Head of R&D
- Whose work is monitored with this KPI?
  - Engineer in charge of deployment
  - Software developers
- To whom is the KPI reported?
  - Commercial
  - Head of greenhouse project
  - Head of R&D

For the Step #2, we have defined the following business goals (BGs):

- Scaling up from a couple of greenhouses to tens or hundreds
- Reducing the cost associated with deployment of the application on the Edge
- Ease junior installation and then recruitment

From the three BGs above, high availability (BG2) was chosen as the main goal. If BG2 is met, BG1 and BG3 will also indirectly be affected, thus meeting our goals.

In Step #3, goals and the initial KPIs were rephrased to create measurable performance result. After this step, the following result has been identified:

- Using PHYSICS allow to adapt and deploy the application in a variety of greenhouse context with minimal effort in terms of development.

To even dig deeper and create a potential measure, in Step #4, the results are further evaluated. The created measures were the following:

- Extra-time needed to deploy the solution in a new context

Step #5 dealt with how the measurements are supposed to be. Based on the analysis, it is determined that the data is directly measured by checking the efforts required to adapt the application to new greenhouse context. The analysis performed lead to the formalization of what defines a context, which includes:

- The type of greenhouse supervisor
- The type of sensors used in the greenhouse
- The network configuration
- The operating system present in the greenhouse

Moreover, two kinds of adaptations could be needed to address a new context:

- The adaptation of the environment deployed
- The adaptation of the methods for data collection and preprocessing

Following Step #5, in Step #6, the optimal value is determined to be 0, which is the extra-time needed to deploy the application in a new context, estimated from the time needed to deploy the application in a fully mastered context. Desired value range is expected to be between 0 and 2 man-days.

The last steps (namely Step #7 and Step #8) were to identify whether there is a (business)-process that the KPI should be integrated in. The reduction of the time needed for adapting the application to a new context has a direct impact on the billing plan and on the developer team planning.

Once measured on a set of contexts to define an achieved upper limit, the KPI could be easily integrated in business scenarios by the commercial director and in the developer, team tasks scheduling by the head of greenhouse projects together with head of R&D.

6.1.3 Building measurable performance results

Following the methodology experimented during the KPI Workshop we derived measurable performance results for each KPI and associated objectives (see Table 6).

Table 6: Measurable performance results and PHYSICS objectives for smart agriculture Use Case.

KPI	Metrics	Objective
Amount of data lost	Number of connection failure Number of data lost	No data lost no matter how many connections failure occurs
Update on plant status	Number of interruptions Interruption time	No interruption of service
Time to perform optimization	Response time	Optimization must be performed in less than 1h
Adaptation time to new contexts	Time to adapt the environment Time to adapt the data collection procedure	Adaptation time must be smaller than 2 man-days

6.2 KPI evaluation in the current infrastructure

6.2.1 Amount of data lost

To evaluate this indicator the data collection process has been monitored in two types of greenhouses belonging to R&D partners of Cybeletech. Monitoring has been performed from October 2021 to June 2022 in the first, and from July 2021 to September 2021 in the second. Time series collected during these periods are shown in Figure 18 and Figure 19.

Figure 18: Temporal series describing the climate variables collected in greenhouse type 1 with the currently deployed data collection pipeline over the experimentation period.

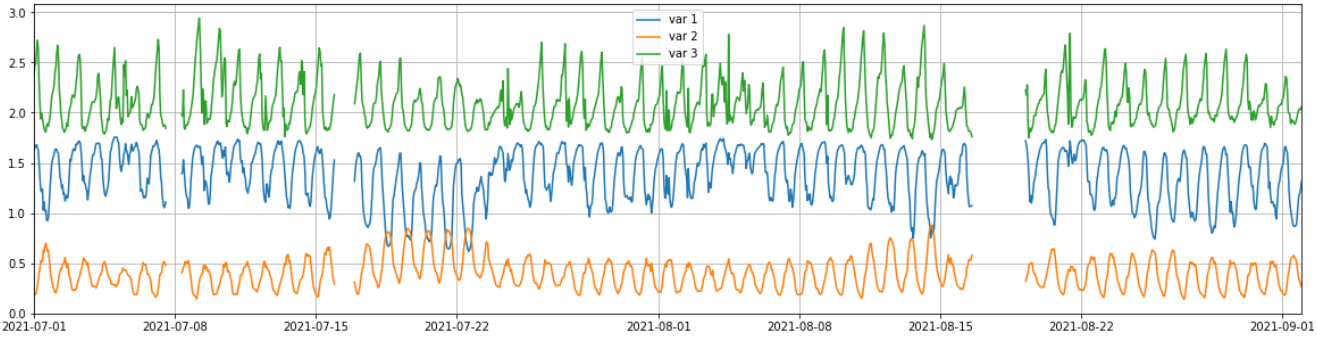


Figure 19: Temporal series describing the climate variables collected in greenhouse type 2 with the currently deployed data collection pipeline over the experimentation period.

The performance indicator used to quantify the amount of data lost integrates:

1. The number of connection failure during the monitoring period
2. The number of data point lost
3. The amount of data lost, expressed in kilobytes (kB)

During the monitoring period, six connection failures were encountered for greenhouse Type 1. These failures resulted in the loss of 27400 data point or 220 kB. For greenhouse Type 2, three connection failures were encountered, resulting in the loss of 8000 data point or 63 kB (Table 7).

Table 7: Number of connection failures and corresponding amount of data lost, expressed in number of data point and volume, for greenhouse type 1 and type 2 over the experimentation period.

	Failure number	Lost data point	Lost data amount
Greenhouse Type 1			
October 2021	0	0	0 kB
November 2021	1	5000	40 kB
December 2021	1	2900	23 kB
January 2022	1	5800	46 kB
February 2022	1	8600	69 kB
March 2022	1	1500	11 kB
April 2022	0	0	0 kB
May 2022	1	3600	29 kB
TOTAL	**6**	**27400**	**220 kB**
Greenhouse Type 2			
July 2021	2	500	4 kB
August 2021	1	7500	59 kB
TOTAL	**3**	**8000**	**63 kB**

6.2.2 Update on plant status

To evaluate this indicator the simulation process has been monitored in two types of greenhouses belonging to R&D partners of Cybeletech. Monitoring has been performed from October 2021 to June 2022 in the first, and from July 2021 to September 2021 in the second.

The performance indicator used to quantify the failure of plant status update process is composed of two metrics:

1. The number of service interruption during the monitoring period
2. The interruption times

During the monitoring period six service interruptions occurred for greenhouse Type 1, with interruption duration ranging from 2 to 12 days. For greenhouse Type 2, three interruptions occurred with interruption duration ranging from less than one day to four days (Table 8).

The total interruption duration is 38 days out of eight months (round 15%) for greenhouse Type 1 and four days out of two months (round 6.5%) for greenhouse Type 2.

Table 8: Number of interruption and corresponding interruption time of the plant status monitoring service, for greenhouse type 1 and type 2 over the experimentation period.

	Number of interruptions	Interruption time
October 2021	0	0 days
November 2021	1	7 days
December 2021	1	4 days

January 2022	1	8 days
February 2022	1	12 days
March 2022	1	2 days
April 2022	0	-
May 2022	1	5 days
TOTAL	**6**	**38 days**
July 2021	2	< 1 day
August 2021	1	3 days
TOTAL	**3**	**4 days**

6.2.3 Optimization process performance

To evaluate this indicator, we focused on the calibration pipeline. The main difference between management optimization and calibration pipeline are the model inputs explored. In the case of management optimization, we aim at exploring the impact of climate conditions in the greenhouse on plant performance, while in the case of calibration we aim at exploring the impact of parameter sets on model accuracy.

The parallelization logic is identical: simulations are distributed over workers and model outputs are evaluated according to some metrics specific to the Use Case. The results regarding computational performances are then expected to be similar.

The reference performance is evaluated by monitoring the runtime according to different scenarios. On the one hand, we evaluated the runtime variations with the size of the design of the experiment when running the pipeline without parallelization. On the other hand, we evaluated the speedup when using the parallelized version of the pipeline. The distribution of parameter sets evaluation is implemented with MPI, and the pipeline is run on demand on a cluster with 16 nodes.

The experimental design is a combination of climate variables, aggregators and time window. The objective is to identify the combinations and to estimate the associated model parameters allowing to reproduce the outcome of a set of physiological processes as precisely as possible.

For example, we can aggregate the temperature in the greenhouse over an hour by summing the temperatures sent by the sensors every 10 minutes and the net radiations received by the leaves of the plant over 2 hours by averaging the values sent by the sensors every 30 minutes. These two-input variables are fed to a function that estimates biomass production over a period defined according to the *in-situ* observations available. The outcome of this function is compared to the observations and the merit of the triplets is quantified through the Root Mean Square Error. This evaluation procedure is repeated over all the points of the experimental design to find the most suitable combination.

According to the physiological process for which we seek to estimate the triplets and associated model parameters; the runtime of an estimation can slightly vary. In the experimentation presented in this document, we focused on the model describing the yield of the plant. The number of time windows explored is fixed to 10, the number of climate variables is varied between 2 and 5, and the number of aggregators is varied between 1 and 3.

According to the design of experiment, the runtime of the parameter set evaluation procedure ranges from 2.5 seconds when two climate variables and one aggregator are explored, to 50 minutes when 5 climate variables and 3 kinds of aggregators are explored. The runtime increases exponentially with both climate variables and aggregators (Figure 20.A).

In real case scenario, we aim at exploring up to 10 climate variables and 2 to 5 aggregators over 5 to 10 different physiological models. To perform the parameter set estimation for one model with 8 climate variables and 3 aggregators, we need approximately 4 days of computation. To be able to propose accurate models to the growers within an acceptable timeframe, a parallelized version of the calibration procedure is used.

The parameter sets are split according to the number of processors and the evaluations are distributed using MPI. The calibration pipeline is deployed on a dedicated server with 16 cores. With 16 cores, MPI enables a speedup close to 15 (Figure 20.B), which reduces the calibration time to about 7 hours by the physiological model.

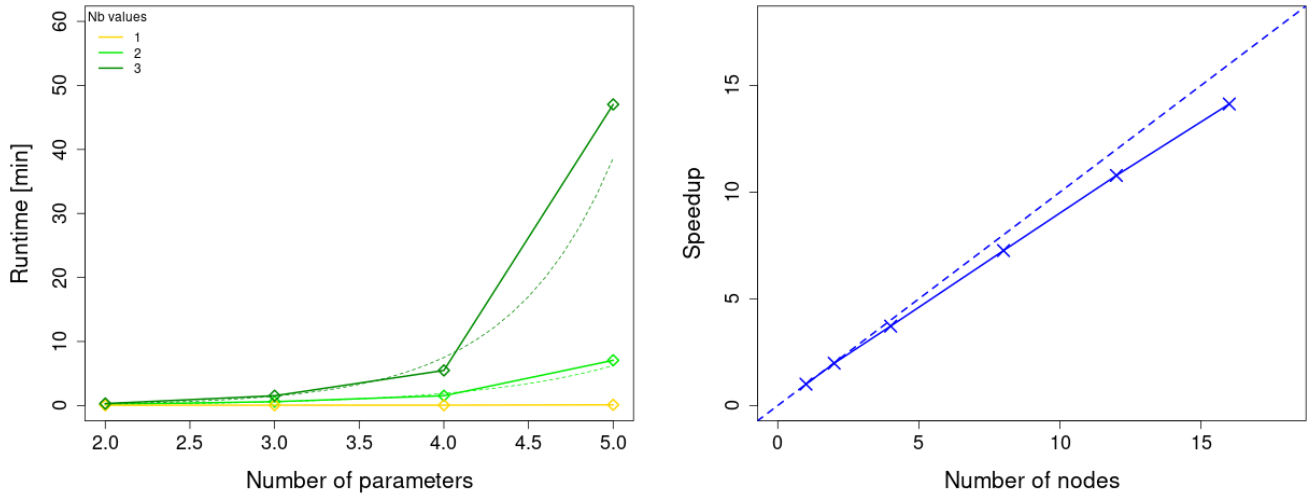


Figure 20: Runtime of the parameter set evaluation procedure according to the design of experiment (A) and MPI scalability evaluated through the speedup (B).

6.2.4 Adaptation time to new contexts

The KPI workshop enables to formalize a measurable metric on the adaptation time which is the sum of times needed for several distinct tasks. However, when the adaptation of Cybeletech application from type 1 to type 2 greenhouse has been performed, the needed time has not been monitored with enough details to estimate the constructed indicator.

As a reference evaluation it is then proposed to use a very coarse estimation of the effort invested in the adaptation of the application from type 1 to type 2 greenhouse. The estimated time for adapting the application is over 15 man-days.

6.2.5 Reduction of costs

The data collection pipeline is currently deployed on the greenhouse's supervisor, which remains on 24 hour/7 day. The main cost reduction is then a direct outcome of the reduction of adaptation time to new contexts.

Simulation and pipeline is currently deployed on Cybeletech servers with some other applications. For the time being, update on plant status is performed automatically every hour to ensure that the growers access to up-to-date results.

As for now, when calibration is needed, a server with 16 cores is leased, the calibration pipeline is deployed on this server and the Cybeletech engineer trigger the calibrations according to growers' objectives and *in situ* measurements provided. The minimal length of the lease is one month, and the cost is around 400 euros.

To calibrate one physiological model, we need to run around 50000 parameter set evaluations. In general, a grower is working on two to three different varieties and is interested in up to five physiological processes.

For one grower, we then need to run around 3 (varieties) x 5 (physiological models) x 50000 = 750000 parameter set evaluation, which corresponds roughly to 5 days of computation when parallelizing on 16 cores.

To reach nominal use of the server, we then need to have six growers asking for the integration of new varieties and/or physiological processes in a month. In this case, we must ensure that *in situ* measurements are completed for the six growers before starting the server, and we cannot guarantee the same delivery time to all because we need to run the calibration one after the other. In case of delay, we need to extend the lease of the server, which induces an extra 200 euros cost per month.

As for now, this kind of servers are under-used since we have at most two to three growers needing to integrate new varieties and/or physiological processes per month. As we highly relate to *in situ* measurements provided by the growers, we regularly need to extend the lease to match the growers' operational constraints.

6.3 KPI evaluation using PHYSICS components.

6.3.1 Amount of data lost

To evaluate this indicator, the script for sensor data parsing has been adapted and integrated in the Edge-ETL flow. The flow has then been parametrized and deployed on a Cybeletech computer, acting like a greenhouse supervisor, following the local installation procedure (Zarzycki & Labropoulos, 2022). We then simulated connection failure of length ranging from 1 hour to 1 day by randomly deactivating the internet connection on the computer.

After 2 weeks of experimentation, the amount of data lost due to connection failure is 0. All the data have been sent to the distant database and made available to run model simulations (Fatouros, et al., 2023).

Data lost can also be due to the greenhouse supervisor restarting (for example for maintenance operation). It would be of great interest for Cybeletech to include an automated local deployment procedure of the Design Environment to ensure continuity of service in case of supervisor maintenance.

6.3.2 Update on plant status

The logic behind the update on plant status in Cybeletech offer has changed since the beginning of the project. Initially, it was planned that the simulation pipeline will be deployed in the greenhouse supervisor so that growers can trigger the simulation using a local software. However, the Cybeletech decision support system is now provided through a web platform, including dashboards providing information on climate conditions in the greenhouse, plant status estimated using Cybeletech simulation framework and cost monitoring tools.

The simulation pipeline has then been deployed in the Cloud as FaaS using the Design Environment, so that it can be invoked on growers' demand to provide up-to-date estimation on plant development and needs. We envision to usage scenarios regarding invocation of the simulation function: the invocation could be made on a regular basis during the growing season to ensure up-to-date dashboards with no visible delay for the grower, or it could be made on grower's demand with some delay due to the function execution and associated dashboard update.

Using the testing interface of the Design Environment, we estimate this delay to be between 20 and 30 seconds in case of a cold start and between 2 and 5 seconds with a warm start. Cold start could be acceptable in the first scenario, but warm start would be needed in the second one. The cost/benefit implications are discussed in section 6.2.5.

6.3.3 Optimization process performance

The calibration pipeline has been adapted and deployed using the Design Environment. Three implementations of the pipeline have been proposed:

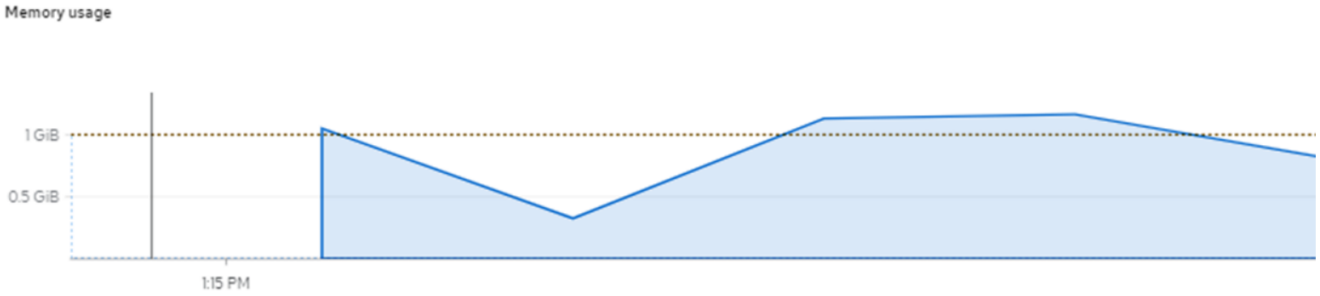
1. A standard implementation with no parallelization
2. An implementation using the split and join pattern to parallelize the parameter set evaluations using FaaS
3. An implementation using two split and join instances, one to parallelize the parameter set evaluation using FaaS, and the other to take advantage of multiprocessing capabilities of each function.

The performances of these three workflows are compared together, and with the current infrastructure evaluation based on runtime and speedup.

First, we run a performance analysis on the parameter set evaluation function, which is the most time consuming, to dimension the size of the splits and of the pods hosting the execution. This performance analysis is run using the Design Environment and the profile of the function highlights a memory leak in the parameters set evaluation procedure with memory consumption reaching 1Go (Figure 21.A).

Profiling the parameter set evaluation procedure shows that the memory leak results from the use of the cross-validation procedure of the Python library Scikit-learn. We then implement a home-made cross-validation procedure which allows to reduce significantly the memory consumption of the parameter set evaluation procedure (Figure 21.B), and then to run it on smaller pods.

A. Memory consumption of the parameter set evaluation with Scikit-learn CV



B. Memory consumption of the parameter set evaluation with home-made CV

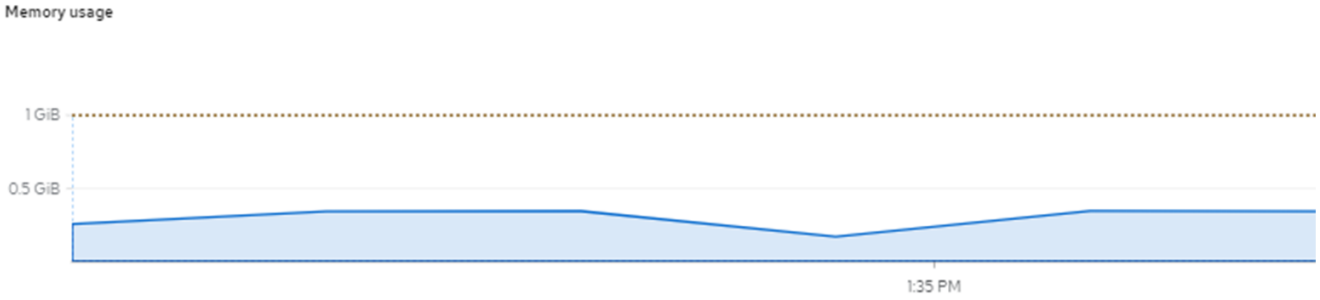


Figure 21: Memory consumption of the parameters set evaluation procedure with (A) and without (B) use of the Scikit-learn cross-validation (CV) function.

We then analyzed the differences between the runtime of the parameter set evaluation procedure over three scenarios:

1. Local execution of the Python script used to perform parameter set evaluation.
2. Execution on Node-RED server using the local version of the Design Environment.
3. Execution in the Cloud as FaaS triggered using the test interface of the Design Environment.

The execution on Node-RED server is about 50% slower than the local execution irrespective of the number of parameter sets evaluated, while the execution as FaaS induces a further increase of around 25% for 10 parameter sets evaluation and 10% for a larger number of sets (Figure 22).

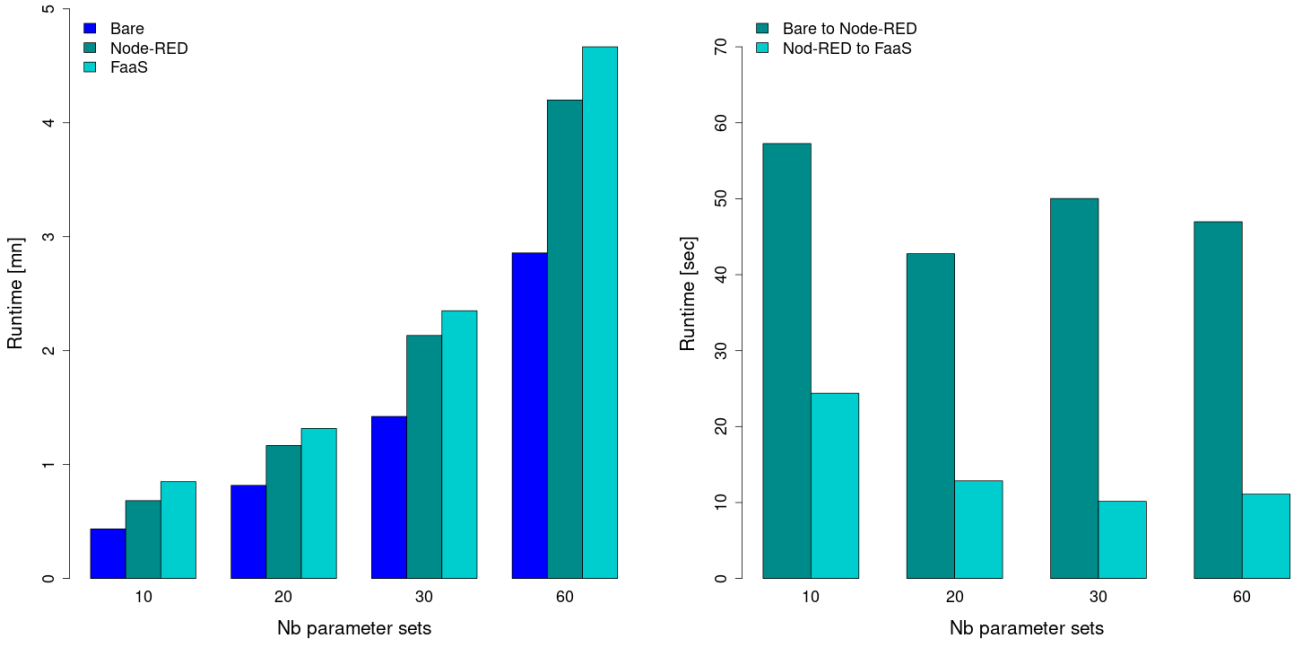


Figure 22: Runtime for parameter set evaluation procedure using bare Python script in local system, Python script embedded in Node-RED flow and run on Node-Red server and Node-RED flow run as FaaS in the Cloud (A). The difference between local system and Node-RED.

To evaluate the gain induced by FaaS parallelization, we run the split and join calibration flow with 40 parameter sets and split size of 40, 20, 10 and 5, which corresponds to 1, 2, 4 and 8 pods, respectively. The flow is run from the local Design Environment using Node-RED server with the parameter set evaluation procedure being called as FaaS using the split and join pattern.

Distributing parameter set evaluation as FaaS along two pods enables to increase the speed of the parameter set evaluation phase by 50%. However, increasing further the number of pods increases only slightly the runtime (Figure 23).

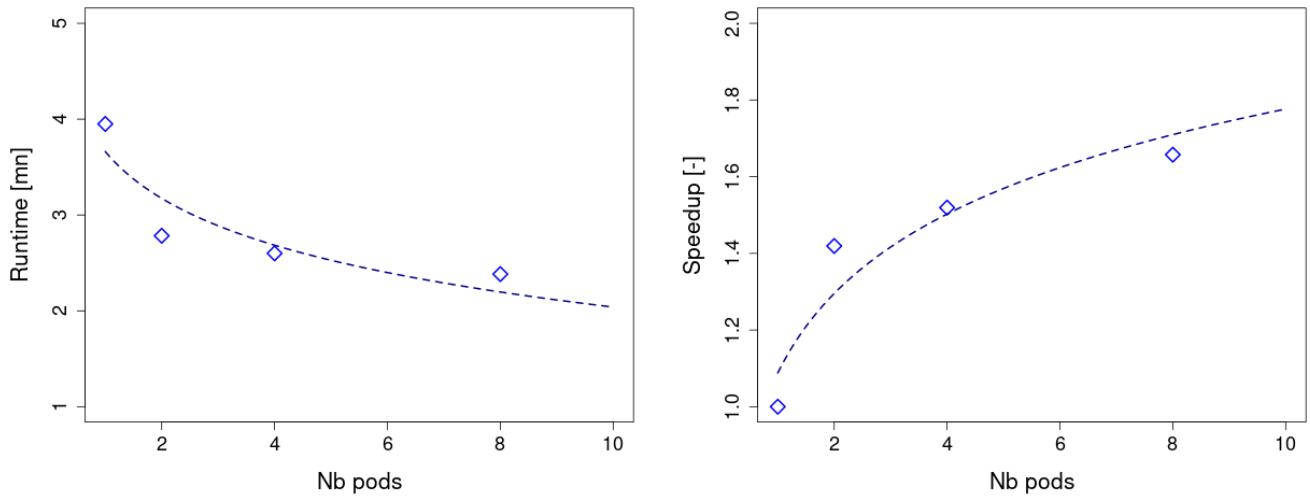


Figure 23: Parameter set evaluations speeding using the FaaS option of the split and join pattern.

To evaluate the gain induced by multiprocessing, we run the split and join calibration flow with 120 parameter sets and split size of 120, 60, 40, 30, 20, 15 and 12, which corresponds to 1, 2, 3, 4, 6, 8 and 10 processors, respectively. The flow is run from the local Design Environment using Node-RED server with parameter set evaluation procedure being called via the Python script using the split and join pattern.

Using the multiprocessing option of the split and join pattern allows to efficiently reduce the runtime. For the 120 parameters set, evaluation takes around 10 minutes with one processor and less than 3 minutes when running on 10 processors, yielding a speedup of 33% (Figure 24).

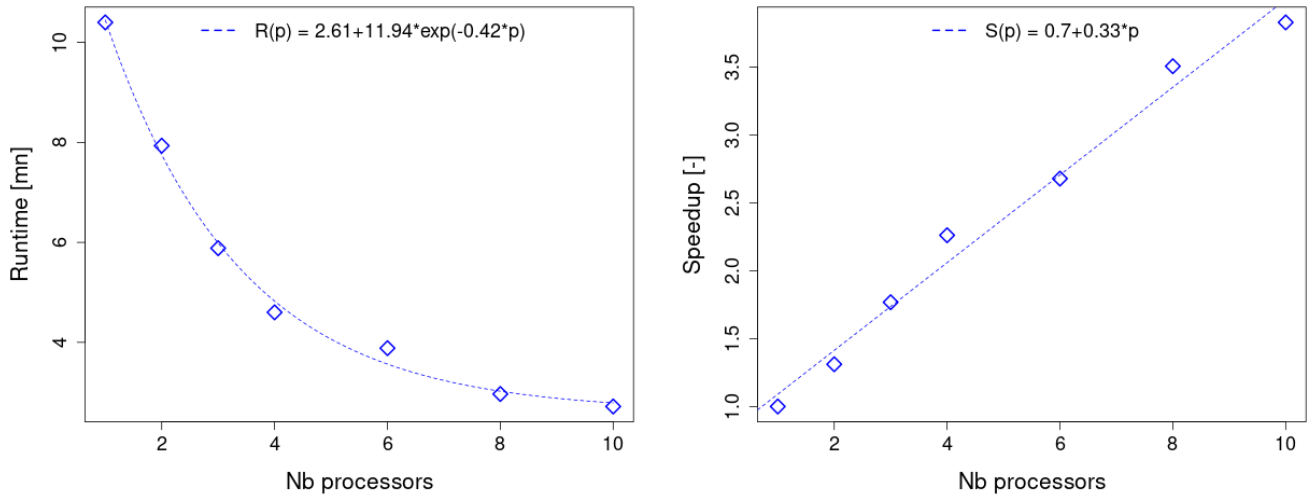


Figure 24: Parameter set evaluations speeding using the multiprocessing option of the split and join pattern.

6.3.4 Adaptation time to new contexts

During the first phase, we mainly evaluate the adaptation time for the data collection procedure, assuming that the Design Environment is deployed on the greenhouse supervisor. To perform this evaluation, Python scripts generating files with different format and content have been implemented. The time required to adapt the procedure of data parsing, which is used as input of the Edge-ETL flow, and to adapt and deploy the newly constructed flow has been monitored.

According to the complexity of the data structure and the format of the file, the complete adaptation time ranges from a few minutes to a few hours, which is far lower than the time required to perform the same adaptation within the existing application. This is mainly achieved thanks to the use of the Edge-ETL pattern, which remove the need to develop and configure bash wrappers, and to the docker image made available through the Design Environment, which can be directly deployed on the Edge (Table 9).

Table 9: Tasks and associated times needed to deploy the data collection pipeline developed for greenhouse type 1 in greenhouse type 2, compared to times needed to deploy the PHYSICS prototype on the testing server.

Task	For partner greenhouse	For the prototype
Python script for data collection and storage	1 to 2 man-days	< 1 man-days
Python script for connection failure handling	3 to 5 man-days	0 man-day
Bash scripts for task scheduling	3 to 5 man-days	0 man-day
Deployment procedure	4 to 6 man-days	< 1 man-days
TOTAL	11 to 18 man-days	< 2 man-days

During the intensification phase, we focused on simulation and calibration pipelines with deployment in the Cloud as FaaS. As for the data collection pipeline, once the baseline Node-RED flow has been defined, the adaptation of the different nodes, which in our case are an invocation of Python scripts, is straightforward.

The refactoring cost of the pipelines has been relatively high for Cybeletech since our legacy codes were not based on the functional programming paradigm. However, Cybeletech is already reaping the benefits of this approach since we enhance our offer to aromatic herbs with very low development costs regarding pipelines adaptation. Moreover, it facilitates the continuous improvement of the different nodes, including, for example, experimental design generation and parameter set selection with very low dependencies on the plant growth models.

Finally, the parallelization capabilities offered by the split and join pattern, which can be used with any FaaS function, or Python script for multiprocessing, without any code adaptation is a real improvement compared to tools such as openMP or MPI which requires invasive changes in the codes and are likely to introduce some complexity. Besides these technical considerations, it is worth noting that HPC specialists are a scarce resource, and even more in the smart agriculture ecosystem.

6.3.5 Reduction of costs

The previous section demonstrates the gain in deployment time enabled by the use of the PHYSICS Design Environment. In terms of cost, it corresponds to a reduction of more than 50% of the cost associated to the deployment of the data collection pipeline in a new type of greenhouse.

If the reduction costs induced by the refactoring of Cybeletech DSS code base is more difficult to evaluate, we can already feel the difference with facilitated continuous improvement of the different modules of the DSS and faster installation of new recruits.

On the other, with the full integration of simulation and optimization pipelines in the prototype during the impact intensification phase, these services can now run on demand in the Cloud. This implies that the maintenance of Cybeletech computing servers will no longer be necessary leading to a significant reduction of operational costs.

For simulation, we explore two sets of scenarios: in the first one, we assume that the simulation is run daily for each greenhouse, as it is currently the case; in the second scenario, we assume that the simulation can be run on grower's demand with a varying maximal number of requests by the grower and by day.

In the first scenario, the number of requests by month ranges from 30 when the service is made available for one greenhouse to 300000 when it is made available for 10000 greenhouses. In the second scenario, we explore the same range of variations in greenhouses number with a maximal number of requests varying between one and ten.

The monthly cost of the service is estimated using AWS lambda calculator, with simulation length being set to 30 seconds and memory required to 256 Mo. In scenario 1, the monthly cost of the service for Cybeletech ranges from 0 when the service is made available in less than 2000 greenhouses to around 22 euros in when the service is made available in 10000 greenhouses (Figure 25, green line). The monthly cost per greenhouse is then less than one cent.

When increasing the maximal number of requests per day to 5, the monthly cost by greenhouse remains smaller than one cent while the service is made available in less than 1000 greenhouses. For a higher number of greenhouses, the monthly cost is around 1.5 cents per. Increasing the maximal number of requests to 10 per day induces an increase of approximately1.5 cents by greenhouse (Figure 25).

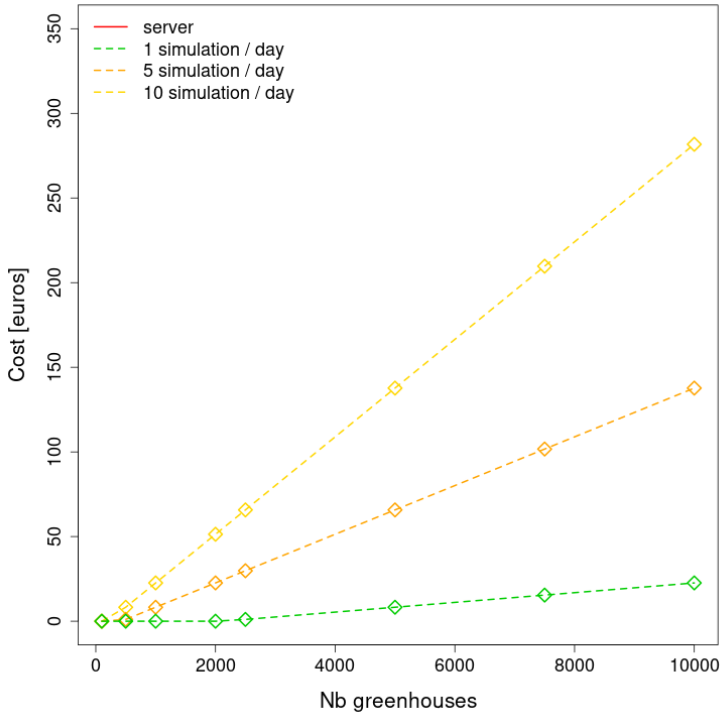


Figure 25: Monthly cost of the simulation service for Cybeletech according to the number of greenhouses for which the service is made available and to the number of requests per day allowed.

For calibration, we explore four scenarios of FaaS parallelization:

1) 5 parameter sets by function using 256 Mo for 2 minutes.
2) 10 parameter sets by function using 256 Mo for 2 minutes.
3) 25 parameter sets by function using 524 Mo for 4 minutes.
4) 50 parameter sets by function using 1024 Mo for 8 minutes.
5) 50 parameter sets by function with multiprocessing using 1024 Mo for 2 minutes.

The AWS calculator allows to explore the trade-off between number of requests from one side and function memory and runtime from the other side. The number of parameters set evaluations is simply calculated as the number of requests multiplied by the number of evaluations performed by each worker. This study shows that the best trade-off is around 10 parameter sets by function (Figure 26). This configuration allows to have a cost by grower of around 30 euros, which is half the server-based price (400 euros divided by 6 growers), assuming maximal efficiency in usage.

It is interesting to note that the multiprocessing option allows to reduce the time needed to complete a request and should then allow to increase the number of evaluations performed by one function without increasing the cost. However, the number of processors available to run a function cannot be set using AWS lambda, we then assume a similar cost for single processor and multi-processors runners, which can be an optimistic assumption.

As we are not limited by the number of cores, we can expect much faster response time with FaaS approach, and it allows to trigger the calibration pipeline as soon as *in situ* measurements are available without prioritization among growers, as it is the case with server-based approach.

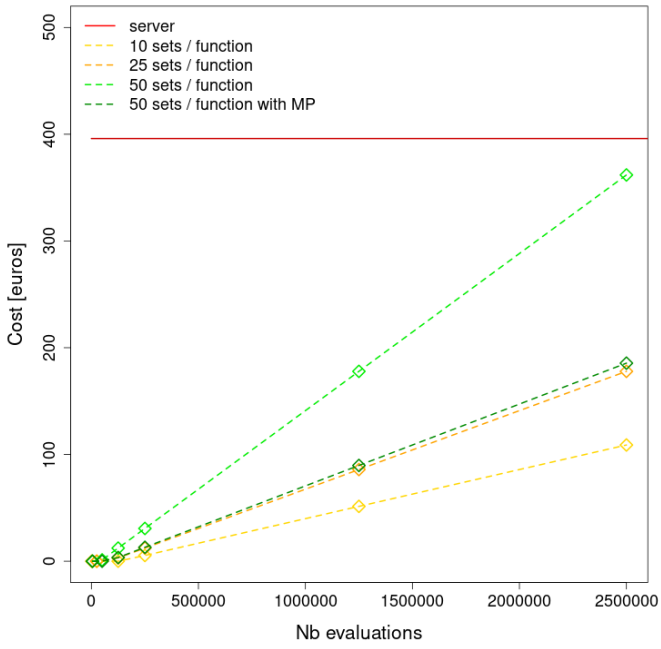


Figure 26: Cost of calibration using FaaS parallelization according to the number of parameters sets evaluations performed and to the number of parameter sets evaluated by each function.

6.4 Concluding remarks

The application prototype developed with PHYSICS components enabled to achieve the objective regarding the data collection pipeline with no data lost during the experimentation period. Moreover, the time needed to adapt the data collection pipeline to the context of PHYSICS was significantly lower than the time needed to adapt the pipeline developed for greenhouse type 1 to greenhouse type 2 (Table 10). The speedup of application deployment enabled by the use of PHYSICS components will induce a reduction in costs for the growers.

The simulation and calibration pipelines have been fully integrated in the application prototype and evaluated during the impact intensification phase. The simulation pipeline has been deployed as FaaS and can now be run on demand. The invocation model for this function is still being discussed with business partners but in any case, we foresee significant cost reduction and facilitation in the development process. The calibration pipeline, which is triggered on demand by Cybeletech agronomic engineer, has been packaged as a Node-RED flow and takes advantage of FaaS and multiprocessing parallelization offered by the split and join pattern to reduce the runtime. If we expect a reduction in the costs associated to calibration, the main benefit of using FaaS is the flexibility and scalability allowed by running thousands of functions in parallel with no deployment effort.

The computation capabilities of the testbed did not allow to run calibration in production conditions. However, the proof of concept performed with small samples enables to foresee the benefits of FaaS for applications requiring massive parallelization of simulations with limited resource requirement. Moreover, the performance monitoring tools developed in PHYSICS will facilitate the identification of bottlenecks and will help developers to build thrifty functions in terms of resource consumption.

Table 10: Measurable performance results and PHYSICS objectives for smart agriculture Use Case evaluated with PHYSICS application prototype and compared to the evaluation performed on the currently deployed applications.

KPI	**Objective**	**In partners greenhouses**	**With application prototype**
Amount of data lost	No data lost no matter how many connections failure occurs	500 to 8000 data points lost / month	No data point lost during the testing period
Update on plant status	No interruption of service	Up to 10 days of interruption / month	Dashboards continuously available and up to date
Time to perform optimization	Optimization must be performed in less than 1h	> 4 hours	Runtime reduction up to 50 times with similar costs
Adaptation time to new contexts	Adaptation time must be smaller than 2 man-days	> 15 man-days to adapt from type 1 to type 2	< 2 man-days to deploy the prototype

7. USABILITY EVALUATION

The previous section shows how the components embedded in the design environment enable

7.1 Methodology

7.1.1 The System Usability Scale

The System Usability Scale (SUS) provides a reliable tool for measuring the usability. It consists of a ten items questionnaire with five response options for respondents; from Strongly agree to Strongly disagree (Figure 27). Originally created by (Brook, 1996), SUS has become an industry standard, with references in over 1300 articles and publications ((Assila & Ezzedine, 2016)). The noted benefits of using SUS include that:

- It is a very easy scale to administer to participants
- It can be used on small sample sizes with reliable results
- It is valid – it can effectively differentiate between usable and unusable systems

Figure 27: Standard version of the System Usability Scale.

	The System Usability Scale Standard Version		Strongly disagree				Strongly agree
			1	2	3	4	5
1	I think that I would like to use this system.		O	O	O	O	O
2	I found the system unnecessarily complex.		O	O	O	O	O
3	I thought the system was easy to use.		O	O	O	O	O
4	I think that I would need the support of a technical person to be able to use this system.		O	O	O	O	O
5	I found the various functions in the system were well integrated.		O	O	O	O	O
6	I thought there was too much inconsistency in this system.		O	O	O	O	O
7	I would imagine that most people would learn to use this system very quickly.		O	O	O	O	O
8	I found the system very cumbersome to use.		O	O	O	O	O
9	I felt very confident using the system.		O	O	O	O	O
10	I needed to learn a lot of things before I could get going with this system.		O	O	O	O	O

7.1.2 Scoring and interpretation

The total score is 100 and each of the questions has a weight of 10 points. For each of the respondents the SUS score is calculated according to the formula:

$$\text{SUS Score} = (X + Y) \times 2.5$$

Where:

- X = sum of the points for all odd-numbered questions
- Y = 20 – Sum of the points for all even-numbered questions
- Points range from 0 when response is "*Strongly disagree*" to 4 when response is "*Strongly agree*"

As even-numbered questions are all in a negative tone, subtracting the points of each question from 4 ensures that:

- If the response is "*Strongly agree"*, the minimum point, which is 0 for each question, is given.
- If the response is "*Strongly disagree"*, the maximum point which is 4 is given.

The sum obtained is then multiplied by 2.5, enabling a score out of a scale of 100 to be assigned to each respondent.

The SUS score allows to easily compare product usability from iteration-to-iteration. On the other hand, (Bangor, Kortum, & Miller, 2009) lead a study to associate an "absolute usability" to a SUS score. They used a seven-point, adjective-anchored Likert scale to determine if a word or phrase could be associated with a small range of SUS scores. The obtained mapping of SUS score and qualitative evaluation of the usability is detailed in Table 11.

Table 11: Descriptive Statistics of SUS Scores for Adjective Ratings (from (Bangor, Kortum, & Miller, 2009))

Adjective	Mean SUS score	Standard Deviation
Worst Imaginable	12.5	13.1
Awful	20.3	11.3
Poor	35.7	12.6
OK	50.9	13.8
Good	71.4	11.6
Excellent	85.5	10.4
Best Imaginable	90.9	13.4

7.1.3 Adaptation to PHYSICS context

The interface allowing end-users to access to PHYSICS components is the so-called Design Environment (DE). The SUS has then been applied to evaluate the usability of the DE (section 4, Figure 28 in Appendix).

Moreover, it has been completed with a short questionnaire on user background with the objective to identify the main areas for improvement according to target users of PHYSICS platform (section 1-3, Figure 28 in Appendix).

Finally, to get a more precise understanding of the global evaluation of the DE performed through SUS survey, a set of questions regarding specific components has been proposed to the end users (section 5-7, Figure 28 in Appendix).

7.2 Results

For the first iteration, the survey has been submitted to the Use Case partners involved in the experimentation of PHYSICS components. Two members of the eHealth team, two members of the smart manufacturing team and one member of the smart agriculture team have responded to the survey.

Based on the user background survey, each Use Case respondents have its own specificities:

- eHealth respondents have little experience in building and using docker images and a substantial experience in event driven applications and application deployment (not serverless).

- Smart Agriculture respondents have some experience in building and using docker images but no experience in event driven applications or application deployment (not serverless).
- Smart Manufacturing respondents have sound experience in building and using docker images and in event driven applications and has some background regarding application deployment (not serverless).

None of the respondents have sound experience in:

1. Using automation servers
2. Functional programming concepts
3. Creating, deploying, and executing serverless applications

The respondents have all experimented with the Design Environment with different levels of intensity. One respondent evaluated it as "very high", while others evaluated it as "high".

The total SUS score range from 30, which according to (Bangor, Kortum, & Miller, 2009) corresponds to "poor usability", to 72.5, which correspond to "good usability" (Table 11). The mean value across respondents is 51.25 and correspond to "OK usability" (Table 12).

Table 12: Minimal, maximal, and mean SUS scores obtained for each statement of the survey (out of ten) and for the global evaluation (out of 100) in the end of the first period (V1) and after the intensification period (V2).

Questions	**Min**		**Max**		**Mean**	
	V1	V2	V1	V2	V1	V2
1. I think that I would use the DE	0	5	7.5	10	3.1	7.5
2. I found the DE unnecessarily complex	2.5	5	7.5	7.5	5.6	6.7
3. I thought the DE was easy to use	2.5	5	10	7.5	6.3	5.8
4. I think that I would need the support of a technical person to be able to use the DE	0	2.5	10	5	4.4	4.2
5. I found the various functions in the DE were well integrated	5	2.5	7.5	7.5	5.6	5
6. I thought there was too much inconsistency in the DE	5	5	10	7.5	7.5	6.7
7. I would imagine that most developers would learn to use the DE very quickly	0	7.5	7.5	10	5.0	8.3
8. I found the DE very cumbersome to use	0	5	10	7.5	5.6	6.7
9. I felt very confident using the DE	5	5	7.5	7.5	5.6	6.7
10. I need to learn a lot of things before I could get going with the DE	0	5	5	7.5	2.5	6.7
TOTAL SCORE	30	47.5	72.5	77.5	51.25	64.3

The users found the DE consistent (statement 6) and easy to use (statement 3), except for one respondent who disagree with this statement (Table 12).

The lowest scores have been obtained for statements 1, 4 and 10, which means that:

1. The users don't think they would use the DE frequently.
2. The users think that they would need the support of a technical person to be able to use the DE.
3. The users consider that they need to learn a lot of things before being confident with the DE.

During the intensification period, the improvements of the Design Environment made it more use friendly and reliable, which is highlighted by the significant increase in the score obtained for questions 1., 7., 10., and to a less extent in that obtained for questions 2., 8., and 9.

For questions 3., 4., 5., and 6., the score did not significantly vary during the intensification period, even if the minimum over the participant has increased, pointing out that the main flaws of the DE have been fixed. The average score obtained for questions 3. and 6. was already good in the first period evaluation, likely because the backbone of the Design Environment did not change significantly. However, the average score for question 4., and 5. remains below the average after the intensification period. This is probably due to the fact that the infrastructure used for UC experimentation and evaluation, and PHYSICS platform development were not separated, leading to errors during experimentation that cannot be understand easily by end-user. These kinds of errors should not happen on a production infrastructure, alleviating the feeling that technical expertise is needed to use the Design Environment.

Overall, the total average SUS score increased between the end of first and second phases from 51.25 to 64.3, which move its usability from "OK" to "good" (Bangor, Kortum, & Miller, 2009). Moreover, for the most challenging evaluator, the DE obtained a score of 47.5, compared to 30 at the end of the first phase, while for the less challenging evaluator the DE obtained a score of 77.5, which bring its usability close to excellent.

Regarding specific components, in the end of the first period, most of the respondent estimate that Node-RED ease the transition to the functional programming concepts and that the patterns can be adapted to support modifications needed for the application. To a slightly lesser extent they estimate that the provided patterns aid in abstracting details and offers useful ready-made functionality. In contrast all the respondents have a neutral opinion on annotations, and most of them consider that using the DE does not ease the deployment and testing of their applications (Table 13).

In the end of the second period the score obtained for question 3. significantly increased (+2 points), likely because the number of patterns available increased, but also because their development has been guided by Use Cases, and evaluator get more familiar with the pattern-oriented coding style promoted by Node-RED. The score for questions 5. and 6. increases as well, confirming the DE adoption by evaluators pointed out by the SUS survey.

Scores for question 2. remains the same, suggesting that the adoption of annotation by evaluators is still not very strong. This could be because annotations do not have a strong impact on experimentations performed. Indeed, the expected added value of annotations lies in cost and energy optimization, which have not been extensively explored during UC evaluation.

Scores for questions 1. and 4. did not significantly change either, remaining slightly higher than the median score. The high variance in the scores obtained for question 1. suggests that the transition to functional programming is not mandatory for using the PHYSICS platform but is eased by Node-RED when addressed. Likewise, the score obtained for question 4. remains constant highlight different level of pattern adaptation according to the Use Case. When needed, evaluators considered the adaptation relatively easy to pull off.

Table 13: Minimal, maximal, and mean SUS scores obtained for each statement of the survey (out of ten) and for the global evaluation (out of 60) in the end of the first period (V1) and after the intensification period (V2).

Questions	**Min**		**Max**		**Mean**	
	V1	V2	V1	V2	V1	V2
1. Node-RED eases the transition to the functional programming concepts	2.5	2.5	10	10	6.2	5.8
2. The annotations provided by the PHYSICS environment are useful	5	5	5	5	5.0	5.0
3. The provided patterns aid in abstracting details and offering ready-made functionality I find useful	2.5	5	10	10	5.6	7.5
4. The patterns can be adapted in order to support modifications that I need	5	5	7.5	7.5	6.2	6.7
5. The DE ease the deployment of my applications	0	5	7.5	7.5	5.0	6.7
6. The DE ease the testing of my applications	0	2.5	7.5	10	3.7	6.7
TOTAL SCORE	15.0	30.0	42.5	47.5	31.9	38.4

7.3 Concluding remarks

In the end of the first period, the survey highlights the consistency of the DE and the facilities offered by Node-RED regarding application adaptation. However, users do not feel very confident in using the DE in total autonomy and seem to have difficulties projecting themselves into extensive usage. At this stage of the project, various enhancements on the DE and components are still ongoing and Use Case end-users do not apprehend fully all the functionalities offered by PHYSICS DE, probably explaining their low confidence in mastering and exploiting the DE.

After the intensification period, the evaluators reported greater confidence in the use of the Design Environment with good adoption of the use of patterns and visual programming tools. To perform Use Case experimentations and evaluations, the evaluators have extensively used the deployment and testing capabilities of the DE and pointed out the gain induced by these integrated tools. The SUS survey also confirms the diversity of usage of the patterns and visual programming tools across Use Cases and shows that the flexibility of the Design Environment enables addressing this diversity.

To move from “good” to “excellent” usability, the PHYSICS platform needs to migrate from a development infrastructure with inherent limitations and fuzziness, to a production grade infrastructure. This should alleviate the feeling that technical expertise is required to use the Design Environment and put emphasis on some of the functionalities that have been less extensively explored by the UC, such as the annotations.

A first step in this direction as already been made with the release of the Cloud version of the Design Environment. The cloud version significantly eases the onboarding of new users and increases the market attractiveness of the DE since the local installation phase, which required some expertise in docker management and knowledge about development tools underlying the DE, are no longer needed. In the same vein, no action from users is needed to benefit from the updates and new features of the DE since they are made available automatically.

8. CONCLUSIONS

This final version of the Use Case Evaluation deliverable describes a set of KPIs constructed from the objectives defined in T6.2 following state-of-the-art methodology. Each Use Case has constructed its own set of KPIs to measure the achievement of objectives related to application adaptation and deployment with the PHYSICS platform and components.

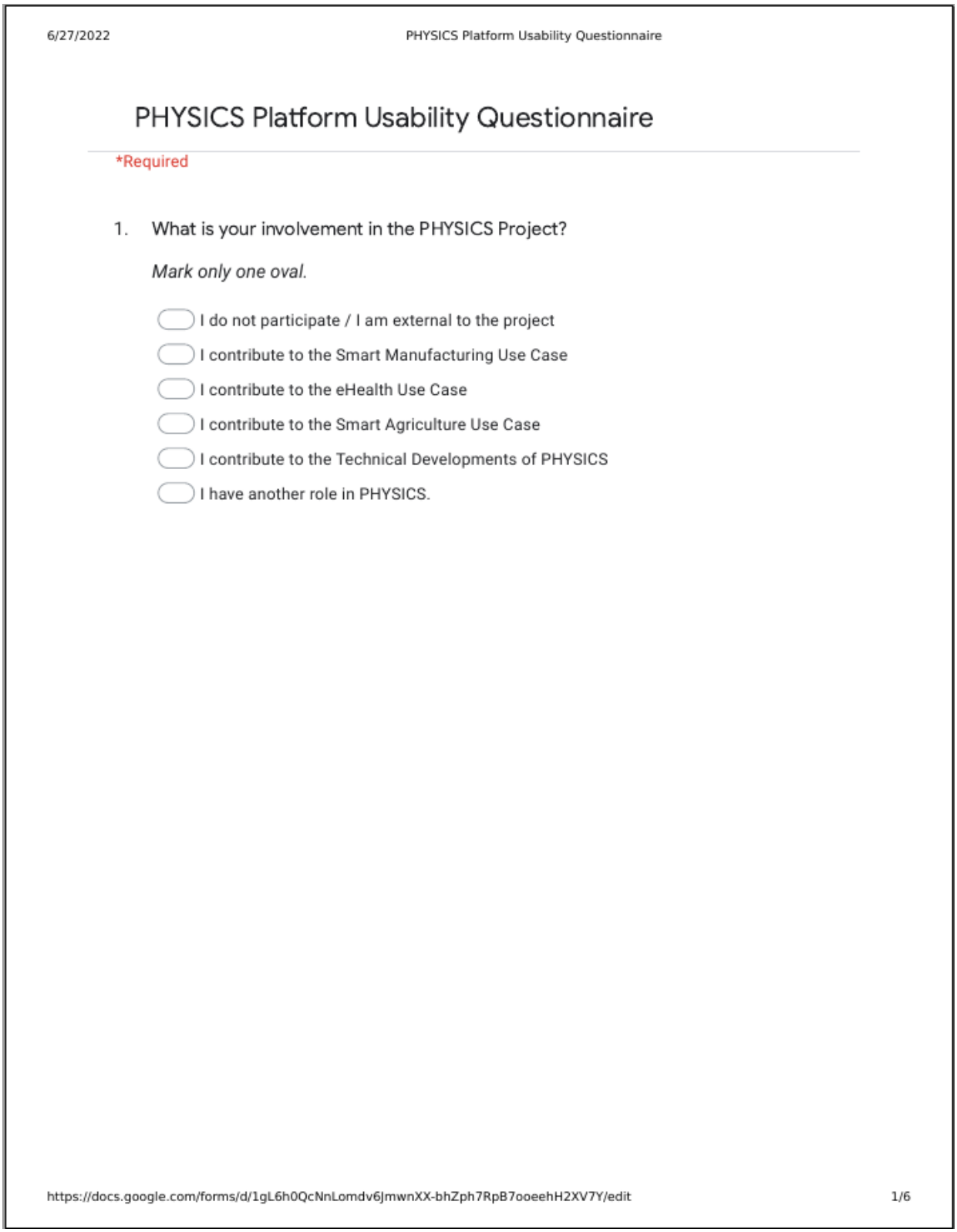
A subset of these indicators has been selected according to the available PHYSICS components and to the prototypes built during T6.3. They have been evaluated with the currently deployed applications to draw a starting point and use the PHYSICS Design Environment as well as specific components. The comparison of the two sets of evaluations enables to give a snapshot of the objectives achieved during the first phase of the project. It shows that:

- The use of PHYSICS Design Environment facilitates the adaptation of applications to new contexts, profiling of the functions for optimal allocation of resources, and alleviates the effort needed for deploying a new service.
- The use of the Edge-ETL pattern significantly increases the reliability of data collection.
- The increase in application runtime due to component interactions remains in an acceptable range for evaluated applications.
- FaaS increases the scalability of the services, allowing them to maintain responsiveness at the highest request rate, in contrast to the server-based version.
- FaaS can be used to massively and easily parallelize function invocation while collecting function outputs for further processing.
- FaaS enables significant cost reduction, particularly when applications can be run on demand with punctual associated infrastructure costs.

On the other hand, an evaluation of the usability of the PHYSICS Design Environment and components based on the System Usability Scale has been conducted. This survey highlights the consistency of the DE and the facilities offered by Node-RED regarding application adaptation. At the end of the first period, users do not feel very confident in using the DE with total autonomy and seem to have difficulties to project themselves into an extensive usage. Since then, many developments on the DE and components have been made; and Use Case end-users apprehend better the functionalities offered by PHYSICS DE, probably explaining their increased confidence in mastering, and exploiting the DE.

9. APPENDIX

Figure 28: Survey proposed to Use Case users to evaluate the usability of PHYSICS Design Environment and components.



6/27/2022 PHYSICS Platform Usability Questionnaire

PHYSICS Platform Usability Questionnaire

*Required

1. What is your involvement in the PHYSICS Project?

Mark only one oval.

- ◯ I do not participate / I am external to the project
- ◯ I contribute to the Smart Manufacturing Use Case
- ◯ I contribute to the eHealth Use Case
- ◯ I contribute to the Smart Agriculture Use Case
- ◯ I contribute to the Technical Developments of PHYSICS
- ◯ I have another role in PHYSICS.

https://docs.google.com/forms/d/1gL6h0QcNnLomdv6JmwnXX-bhZph7RpB7ooeehH2XV7Y/edit 1/6

2. The following statements are related to your previous experience with various relevant software development aspects. Please state in how far you agree with the statements below. *

Mark only one oval per row.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I have experience in building docker images	◯	◯	◯	◯	◯
I have experience in using docker images	◯	◯	◯	◯	◯
I have experience in using automation servers	◯	◯	◯	◯	◯
I have experience in functional programming concepts	◯	◯	◯	◯	◯
I have experience in event driven applications	◯	◯	◯	◯	◯
I have experience in deploying (not serverless) applications	◯	◯	◯	◯	◯
I have experience in creating, deploying, and executing serverless applications	◯	◯	◯	◯	◯
I have experience in using visual programming tools	◯	◯	◯	◯	◯

PHYSICS Design Environment

The PHYSICS Design Environment enables the design of visual workflows of applications. It allows the developer to exploit a set of provided generalised 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.

3. Which statement best describes your current level of experience with using the PHYSICS Design Environment?

Mark only one oval.

- ◯ I don't know what it is.
- ◯ I know what it is, but I have not used it.
- ◯ I have experimented with it once, or twice.
- ◯ I have used it to build something.
- ◯ I have used it a lot.
- ◯ I have helped build or design it.

4. The following statements relate to the PHYSICS Design Environment (DE). Please indicate in how far you agree with the statements below. *

Mark only one oval per row.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I think that I would use the DE frequently.	◯	◯	◯	◯	◯
I found the DE unnecessarily complex.	◯	◯	◯	◯	◯
I thought the DE was easy to use.	◯	◯	◯	◯	◯
I think that I would need the support of a technical person to be able to use the DE.	◯	◯	◯	◯	◯
I found the various functions in the DE were well integrated.	◯	◯	◯	◯	◯
I thought there was too much inconsistency in the DE.	◯	◯	◯	◯	◯
I would imagine that most developers would learn to use the DE very quickly.	◯	◯	◯	◯	◯
I found the DE very cumbersome to use.	◯	◯	◯	◯	◯
I felt very confident using the DE.	◯	◯	◯	◯	◯
I need to learn a lot of things before I could get going with the DE.	◯	◯	◯	◯	◯

PHYSICS Components

The following questions are about specific PHYSICS components.

5. The following statements relate to the use of various other PHYSICS components. Please indicate in how far you agree with the statements below. *

Mark only one oval per row.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Node-RED eases the transition to the functional programming concepts	◯	◯	◯	◯	◯
The annotations provided by the PHYSICS environment are useful	◯	◯	◯	◯	◯
The provided patterns aid in abstracting details and offering ready-made functionality I find useful	◯	◯	◯	◯	◯
The patterns can be adapted in order to support modifications that I need	◯	◯	◯	◯	◯
The DE ease the deployment of my applications	◯	◯	◯	◯	◯
The DE ease the testing of my applications	◯	◯	◯	◯	◯

REFERENCES

Assila, A., & Ezzedine, H. (2016). Standardized usability questionnaires: Features and quality focus. *Electronic Journal of Computer Science and Information Technology*.

Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, 114-123.

Barr, S. (2014). *Practical performance measurement: Using the PuMP blueprint for fast, easy and engaging KPIs.* Samford: PuMP Press. Retrieved from https://www.staceybarr.com/.

Brook, J. (1996). *SUS: a" quick and dirty" usability scale." Usability evaluation in industry.*

Davilla, E. (2018). *Die Unternehmensleistung messen*. Retrieved from https://www.linkedin.com/learning/die-unternehmensleistung-messen/warum-wir-messen?autoAdvance=true&autoSkip=false&autoplay=true&resume=false&u=83641554

Doerr, J. (2018). *Measure What Matters.* London: Penguin Random House.

Fatouros, G., Kousiouris, G., Lohier, T., Makridis, G., Polyviou, A., Soldatos, J., & Kyriazis, D. (2023). Enhancing Smart Agriculture Scenarios with Low-code, Pattern-oriented functionalities for Cloud/Edge collaboration. *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, (pp. 285-292).

Franke, N., Hennecke, A., Gezer, V., Harms, C., Akker, H. o., Pnevmatikakis, A., . . . Touloupou, M. (2021). *D6.3 - Application Scenarios Definition V1.* PHYSICS Consortium.

Hornbæk, K. (2006). Current practice in measuring usability: Challenges to usability studies and research. *International journal of human-computer studies*, 79-102.

Kousiouris, G., Ambroziak, S., Zarzycki, B., Costantino, D., Tsarsitalidis, S., Katevas, V., . . . Stamati, T. (2023). A Pattern-based Function and Workflow Visual Environment for FaaS Development across the Continuum. *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, (pp. 165-172).

Patiño, M., Azqueta, A., Mengual, L., Li, T., Kousiouris, G., Tsarsitalidis, S., . . . Georgiou, Y. (2021). *D2.4 - PHYSICS Reference Architecture Specification V1.* PHYSICS Consortium.

Zarzycki, B., & Labropoulos, G. (2022). Local Installation Guide. PHYSICS CONSORTIUM.

DISCLAIMER

COPYRIGHT MESSAGE