PHYSICS

OPTIMIZED HYBRID SPACE-TIME SERVICE CONTINUUM IN FAAS

D2.3 – STATE OF THE ART ANALYSIS AND REQUIREMENTS DEFINITION V2

Lead Beneficiary	INQ
Work Package Ref.	WP2 – Requirements, Architecture and Technical Coordination
Task Ref.	T2.2 - Requirements & State of the Art Analysis
Deliverable Title	D2.3 - State of the Art Analysis and Requirements Definition V2
Due Date	2022-09-30
Delivered Date	2022-10-07
Revision Number	3.0
Dissemination Level	Public (PU)
Type	Report (R)
Document Status	Release
Review Status	Internally Reviewed and Quality Assurance Reviewed
Document Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	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]
ATOS	Contributor, Internal Reviewer	Ana Soto Jiménez, Lucas Pelegrín Caparros
BYTE	Contributor, Internal Reviewer	Yiannis Poulakis
DFKI	Contributor, Quality Assurance	Volkan Gezer, Carsten Harms, Arnold Herget
FTDS	Contributor	Niklas Franke, André Hennecke
GFT	Contributor	Fabrizio Di Peppo, Maurizio Megliola
HPE	Contributor	Alessandro Mamelli, Domenico Costantino
HUA	Contributor	George Kousiouris, Chris Giannakos, Angeliki Anagnostopoulou, Stelios Tsarsitalidis
INNOV	Contributor	George Fatouros
INQ	Lead Beneficiary	Eleni Argyrioy, Damian Strycharczuk
ISPRINT	Contributor	Harm op den Akker
RHT	Contributor	Luis Tomas, Idan Levi, Josh Salomon
RYAX	Contributor	Yannis Georgiou
UPM	Contributor	Marta Patiño, Ainhoa Azqueta

REVISION HISTORY

Version	Date	Partner(s)	Description
0.1	2022-06-21	INQ	ToC Version
0.1	2022-08-08	ALL	First Round of Contributions
1.0	2022-08-30	ATOS, BYTE	1st Version for Internal Review
1.1	2022-08-31	ALL	Second Round of Contributions
1.2	2022-09-11	ALL	Third Round of Contributions
2.0	2022-09-25	ATOS, BYTE	2nd Version for Internal Review
2.1	2022-09-28	INQ	Address comments from reviewers
2.2	2022-10-03	DFKI	Quality Assurance
3.0	2022-10-05	INQ, GFT	Version for the submission

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

LIST OF ABBREVIATIONS

2FA	Two-Factor Authentication
ACM	RedHat Advanced Cluster Management
AKS	Azure Kubernetes Service
API	Application Programming Interface
AWS	Amazon Web Services
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
DAG	Directed Acyclic Graph
DL	Deep Learning
DNN	Deep Neural Net
DoS	Denial of Service
DRAM	Dynamic Random Access Memory
DSL	Domain Specific Languages
eDoS	Economic Denial of Sustainability
EKS	Amazon Elastic Container Service for Kubernetes
FaaS	Function as a Service
GCP	Google Cloud Platform
GKE	Google Kubernetes Engine
GPG	GNU Privacy Guard
HDD	Hard-Disk Drive
HPA	Horizontal Pod Autoscaler
HSM	Hardware Security Module
HTTP/S	HyperText Transfer Protocol / Secure
I/O	Input/Output
IAM	Identity and Access Management
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
KMS	Key Management Service
LDAP	Lightweight Directory Access Protocol
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MARLA	MApReduce on Lambda
MCSC	Multi-Cloud Service Composition
MILP	Mixed-Integer Linear Programming
ML	Machine Learning
MoSCoW	Must-have, Should-have, Could-have, Will-not-have
MSE	Mean Squared Error
NIST	National Institute of Standards and Technology
NVMe	Non-Volatile Memory Express
OIDC	Open ID Connect
OWASP	Open Web Application Security Project
QoS	Quality of Service

R^2	Coefficient of Determination
RAMP	Reusable Artefacts MarketPlace
RDF	Resource Description Framework
REST	REpresentational State Transfer
RMSE	Root Mean Squared Error
RWX	Read Write Many
S.M.A.R.T.	Specific, Measurable, Achievable, Relevant, Timely
SAML	Security Assertion Markup Language
SDK	Software Development Kit
SGX	Software Guard Extensions
SIG	Special Interest Group
SLA	Service-Level Agreement
SOA	Service Oriented Architecture
SPT	Shortest Processing Time
SSD	Solid-State Drive
SSH	Secure Shell protocol
SSL	Secure Sockets Layer
SSO	Single Sign-On
TLS	Transport Layer Security
TRUSTEE	daTa pRivacy and cloUd Security clustEr Europe
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSS	Cross Site Scripting
XXE	XML External Entity
YAML	YAML Ain't Markup Language

EXECUTIVE SUMMARY

The aim of this deliverable is to provide an upgraded and thoroughly analysed state of the art analysis and requirements definition on the updated requirements for the functional and non-functional components of the PHYSICS architecture. In the previous deliverable the analysis of the requirements was documented using the S.M.A.R.T. (ISO25010, MoSCoW) template of well-known and established standards, something that ensures that these requirements are well defined, understood and in scope. In this deliverable each of the documented requirements will be checked and evaluated towards the corresponding use cases andhow these components satisfy the needs and necessities for each use case.

The state-of-the-art analysis follows the PHYSICS architectural components to ensure that there is a full documentation and a know-how established for each part of the technical aspects of the PHYSICS project. These architectural components are directly derived from the envisioned PHYSICS architecture as it is described in the proposal. To ensure the transition from the initial architecture to the final promised solution, this document analysed all possible technological candidates to be used throughout the project. The requirements elicitation and specification aim at covering the entire PHYSICS project by allowing all partners to participate in the documentation process and considering their own vision of the project. Each requirement is documented within a template that uses well known and established standards (S.M.A.R.T., ISO25010, MoSCoW), something that ensures that these requirements are well defined, understood and in scope. Each of these requirements will be checked and tracked in the PHYSICS project lifecycle with the target of satisfying all the must-have and should-have requirements and most of the could-have ones.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION

The deliverable D2.3 State of the Art Analysis and Requirements Definition v2 is one of the foundational deliverables that documents both (i) all the relevant state-of-the-art technologies and research works and (ii) the initially identified functional and non-functional requirements of the PHYSICS project. D2.3 will provide input for the updated versions of both the reference architecture and scenarios definition, in D2.5 and D6.4 respectively. The relationship of D2.3 with its neighbouring deliverables is depicted in Figure 1.

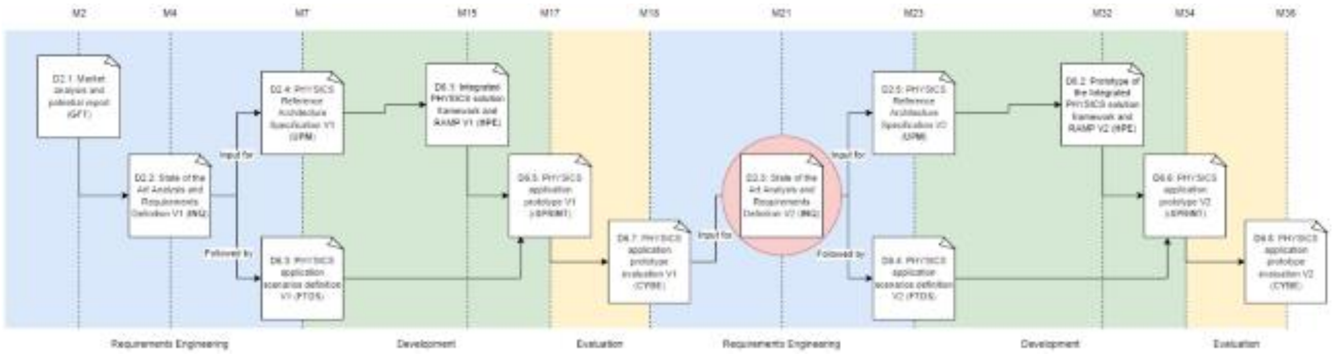

M2
M4
M7
M15
M17
M18
M21
M23
M32
M34
M36
D2.1: Market analysis and potential impact (GFT)
D2.2: State of the Art Analysis and Requirements Definition V1 (INQ)
Input for
Followed by
D2.4: PHYSICS Reference Architecture Specification V1 (UPM)
D6.3: PHYSICS application scenarios definition V1 (PTOS)
D6.1: Integrated PHYSICS solution framework and RAMP V1 (HPE)
D6.5: PHYSICS application prototype V1 (SPRINT)
D6.7: PHYSICS application prototype evaluation V1 (CYBE)
Input for
D2.3: State of the Art Analysis and Requirements Definition V2 (INQ)
Input for
Followed by
D2.5: PHYSICS Reference Architecture Specification V2 (UPM)
D6.4: PHYSICS application scenarios definition V2 (PTOS)
D6.2: Prototype of the Integrated PHYSICS solution framework and RAMP V2 (HPE)
D6.6: PHYSICS application prototype V2 (SPRINT)
D6.8: PHYSICS application prototype evaluation V2 (CYBE)
Requirements Engineering
Development
Evaluation
Requirements Engineering
Development
Evaluation

Figure 1 - Relationship of D2.3 with other deliverables of PHYSICS

1.1 Objectives of the Deliverable

The objectives of this deliverable are to provide the most recent and well-established relevant state-of-the-art technologies for each architectural component of PHYSICS and to aggregate all relevant functional and non-functional requirements that are predicted for the PHYSICS project. In more detail the objectives are the following:

- ➢ Identify all state-of-the-art areas that need to be analysed
- ➢ Provide a thorough analysis and presentation of all technological advances in the selected state-of-the-art areas
- ➢ Ensure that all partners have a saying in the development of the PHYSICS project by allowing them to specify their own requirements
- ➢ Ensure that all requirements are of high quality, well understood and relevant to the project
- ➢ Document and analyse the aggregated requirements
- ➢ Provide a reference document for the rest of the project in terms of state of the art and requirements

1.2 Insights from other Tasks and Deliverables

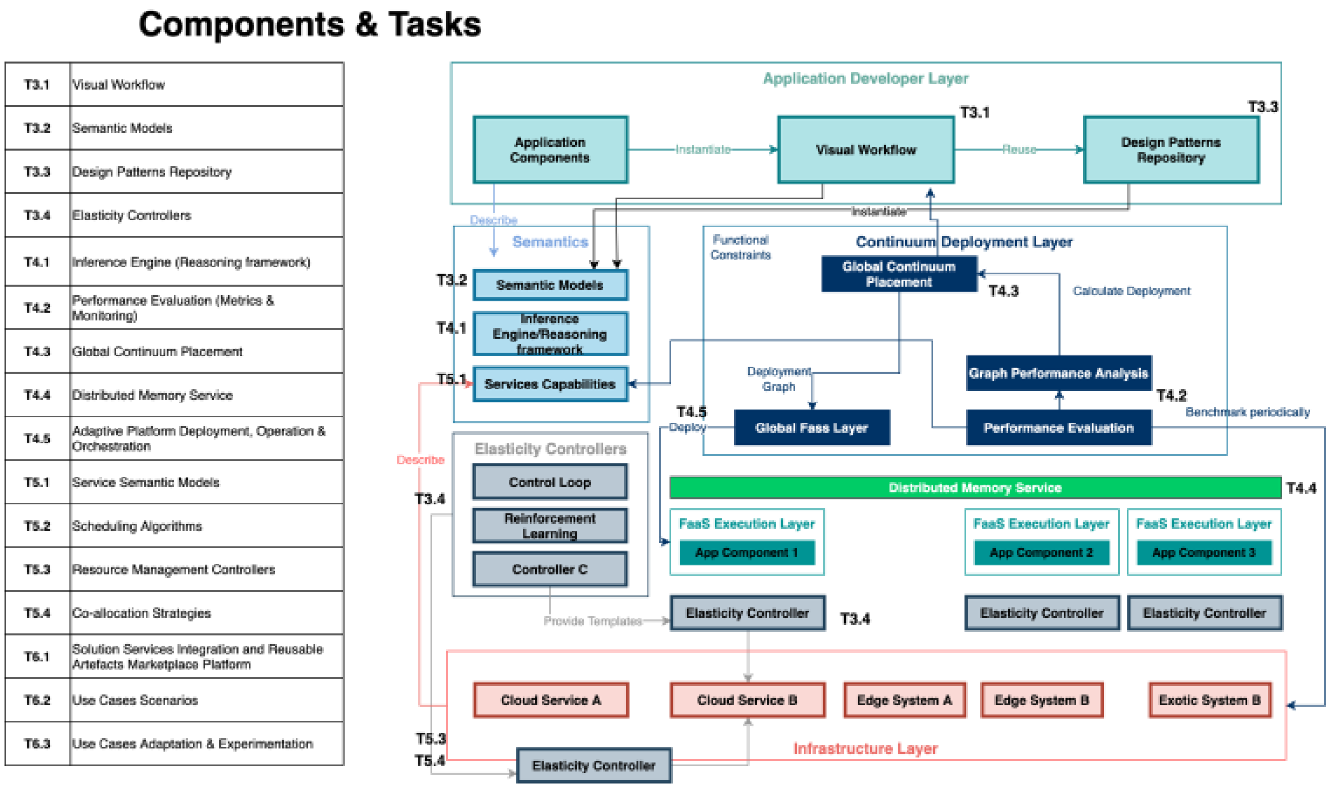
The main inputs of this deliverable are D2.2 "State of the art analysis and requirements definition v1" and D6.7 "PHYSICS application prototypes evaluation v1". D2.2 provides an initial definition of the requirements, upon which D2.3 is built and written with the appropriate enhancements where necessary. Based on the outcomes of D6.7 as well, D2.3 provides an updated state-of-the-art analysis and requirements to ensure a high coverage of all aspects of the PHYSICS project.

1.3 Structure

The rest of this deliverable is structured according to the aforementioned two main pillars. Chapter 2 focuses on the state-of-the-art analysis and is divided in the architectural components – research areas of PHYSICS. In Chapter 3, there is a thorough documentation of all requirements divided in their corresponding tasks. This division is not mandatory, and requirements could be applied to multiple tasks as the project matures and is better understood. A "Requirements Traceability Matrix" is also presented, providing an overview of the requirements. Finally, the conclusion summarizes the lessons learnt from the deliverable and provides a short overview of the entire D2.3 deliverable.

2 STATE OF THE ART ANALYSIS V2

The state-of-the-art analysis is based on the architectural components of PHYSICS. Each sub-section of this section is dedicated to one component of PHYSICS with the sole exception of FaaS security which was added as an overlay component that should be taken into consideration throughout the project. The architecture of PHYSICS with all its components is depicted in Figure 2, a figure that was taken directly from T2.3 to ensure that there is a close relationship between T2.2 and T2.3 and the results of this deliverable are useful for the rest of the project. The only exceptions that are not included in the state-of-the-art analysis are the components from WP6 (T6.1, T6.2 and T6.3) which are about the integration and the use cases, and their analysis is covered by all the previous components. The rest of this section follows the aforementioned structure, with one subsection for each of the PHYSICS components.



Components & Tasks

T3.1	Visual Workflow
T3.2	Semantic Models
T3.3	Design Patterns Repository
T3.4	Elasticity Controllers
T4.1	Inference Engine (Reasoning framework)
T4.2	Performance Evaluation (Metrics & Monitoring)
T4.3	Global Continuum Placement
T4.4	Distributed Memory Service
T4.5	Adaptive Platform Deployment, Operation & Orchestration
T5.1	Service Semantic Models
T5.2	Scheduling Algorithms
T5.3	Resource Management Controllers
T5.4	Co-allocation Strategies
T6.1	Solution Services Integration and Reusable Artefacts Marketplace Platform
T6.2	Use Cases Scenarios
T6.3	Use Cases Adaptation & Experimentation

Figure 2 - The PHYSICS Components Architecture

2.1 Visual Workflow

Visual environments, known also as "Visual Workflow Designer", have emerged in recent years as a user-friendly tool that can speed up application development by abstracting the details that are not directly relevant to the user. Typically, these environments are based on flow programming, based on asynchronous event driven languages such as JavaScript, and offer palettes of readymade nodes that incorporate the major functionalities needed. Function code is applied to the input message (triggering function execution and providing the function input data) transforming its contents based on the function logic and passing it to the next node in line. Furthermore, the Visual Workflow Designer tools encompass means of extension for these nodes as well as external repositories in which such nodes or in general flows can be stored and shared by the community. Environments such as open-source Node-RED for event driven applications and KNIME (mixture of open and proprietary models) for data science flows have emerged, indicating that the need for easier development and deployment of application flows is very relevant and user demanded. Typically, such frameworks are designed for a specific domain (e.g., Node-RED for the IoT, KNIME for data

science) and include neither an end-to-end rationale, that is from design to development and deployment in one single step, nor ready-made patterns for exploiting the cloud model (yet they can be extended to).
In terms of major open source FaaS platforms, these typically do not come with a UI for workflow definition, except for Apache Airflow [1] which also incorporates plugins to interface with cloud services or processes. One drawback of Airflow is that these operators are typically provider-specific and thus cannot be reused while amplifying the vendor lock-in. Additionally, these operators do not include advanced and abstracted cloud design patterns. Fission workflows [2] are mainly programmatically defined while heavily linked with the Kubernetes environment. Proprietary solutions also exist with an extensive list of accompanying services such as the IBM Cloud (formerly Bluemix) environment (and Blueworks) as well as Google Composer (for managing Airflow related workflows) that offer integrated services design, deployment and composition. However, all the previously mentioned solutions are tightly coupled with the associated vendor lock-in.
Let us have a look at some of the most diffused Open-Source Visual Workflow Designer environments to understand how they work and to get the best of them in relation to the purpose of the PHYSICS visual workflow designer, that is to easily design, configure, and deploy FaaS model applications.

Node-RED [3]

Node-RED provides a browser-based flow editor with which flows can be linked using a selection of nodes in the palette. Flows can be easily deployed in the runtime environment. JavaScript functions can be defined using an advanced editor included in the tool. An integrated library allows functions, templates and flows to be stored for reuse. The runtime environment is based on Node.js, taking advantage of its event-driven and non-blocking model. Flows created in Node-RED are stored in JSON format that can be easily imported and exported for sharing with others.

Apache Airflow [1]

Apache Airflow is a workflow design and management tool. It allows the planning, scheduling and automation of the flow of data through nodes. Graphs of Directed Acyclic Graphs (DAGs) type represent the direction of the data, while the output of one node (task) is usually the input of the next node. The acyclic nature of the graph means that the data cannot go backwards.
Airflow allows the dynamic creation of pipelines, written in python language. It is extensible, and the user can create its own set of operators, helping their level of abstraction and understanding. Multiple Amazon Web Services (AWS), Google Cloud applications, and Microsoft Azure capabilities can be integrated into the Airflow workflow environment.

Apache Taverna [4]

While Apache Airflow is mainly used to create regular workflows, Apache Taverna is often used to create scientific workflows. Apache Taverna is represented by a set of the Taverna Engine, the Taverna Workbench and the Taverna Server. The set of the above components is useful for scientists with limited programming languages and/or coding knowledge to build complex data streams on which to perform analyses. Data may come from a variety of public and private sources, involving many fields like geography, medicine, and sports.

Camunda [5]

Camunda is an open-source workflow management tool that allows workflows to be designed in an extremely simplified and efficient way. Built on the three core principles of design, automation and improvement, Camunda allows the workflow design process to be continuously optimized. The workflows produced by Camunda are particularly suitable for complex organizations, ensuring maximum visibility of information to users. Built on a lightweight Java Application Program Interface (API) stack, the platform is reliable and scalable. Camunda is integrated in cloud environments that are accessible from multiple platforms.

Cflow [6]

Cflow is a cloud-based (available on AWS) and open-source workflow management software. Workflows are designed and maintained mainly without writing code, through the availability of a big number of components that can be gathered from the tool's libraries, even with predefined workflows. The tool is

suitable for organizations of different sizes. It allows easy integration with many third-party applications such as SAP.
Knime [7]
Knime is a tool for mainly building data science workflows. It allows to create visual workflows with an intuitive, drag and drop style graphical interface, without the need for coding - including dragging and dropping nodes and components from the KNIME Hub. It allows to blend tools from different domains with KNIME native nodes in a single workflow, including scripting in R & Python, machine learning, or connectors to Apache Spark.
There are many other tools that allow the creation of applications using visual technology, with different features and aimed often to a specific application. Many of the tools are commercial; in the analysis, we have based ourselves exclusively on open-source tools, as they fall within the philosophy of the PHYSICS project.

2.2 Semantic Models for Application Characteristics Description

When it comes to the application descriptions, there must be sufficient expressiveness, so as to effectively describe an application or pattern. This includes its components, as well as the requirements of each component in terms of resources or (cloud) services, performance, dependencies between components and external dependencies. To achieve data uniformity and guarantee the usability of the data by the inference engine (T4.1), the application characteristics (T3.1 output) are transformed and semantically enriched to conform to a metamodel/ontology. The semantic models for application characteristics description are expressed in JSON-LD [8], a format that is consumable by conventional services, as it is pure JSON, as well as semantic web services as it encodes RDF triples that follow the OWL vocabulary [9]. The core goal of T3.2 is the definition of the vocabulary / ontology to be followed by the PHYSICS application workflow models created in the application developer layer and propagated to the lower layers for deployment. As such, the aim is to reuse existing software component descriptions or related concepts, and semantically represent the metadata annotations for the links with the FaaS platform.
Requirements of an application that have to do with resources can use terms from the Ontology for Cloud Computing Instances [10] as well as the Ontology for Service Level Agreements [11]. For locality requirements, the most usable are the GeoJSON-LD [12] vocabulary, which lays out coordinates or geographical areas in a semantics-enabled way. In addition to that, the Vocabulary for Regions and Zones on Cloud Computing [13] can be used to specify the suitable regions of Data Centres where application components can run.
Works on semantic models for multi-cloud service compositions (MCSC) have been developed in recent years. Relevant cases need to include specific services or unique requirements (e.g., object stores etc.) in the application description. The application description has the potential to help identifying matching parameters from the application descriptions with the services and resources described in T5.1 with the use of the inference engine from T4.1. Examples of such works follow. A semantic engine for porting applications to the cloud and among clouds is presented in [14], which later became part of the mOSAIC framework. To aid in modelling cross-cloud deployments and optimise cloud deployments based on QoS metrics, requirements, prices and other SLA parameters, CloudPick was introduced as an operational platform, which is based on the aforementioned ontologies [15].
In [16], a unified cloud service description is proposed, which is meant to be able to accurately represent any cloud service, by consolidating all common characteristics and organising them in nine dimensions/sub-ontologies:

- **Service subontology**, presents the general information about a cloud service (type, deployment model, category, evaluation, service reusability, etc.).
- **Functional subontology**, defines the functionality as the set of operations offered by a cloud service.
- **Technical subontology**, presents the technical aspects, i.e., the way a cloud service is accessed.
- **Participant subontology** defines the different actors (e.g., providers, consumers) participating in the description, composition, and invocation of cloud services.

- **Interaction subontology** describes the services' behavioural aspects, and how cloud providers and consumers interact with services.
- **Service-level subontology** comprises the QoS capabilities of each service (e.g., security, reliability, compliance).
- **Legal subontology**, presents the legal aspects and restrictions of the cloud service's usage.
- **Pricing subontology**, refers information about the fees and pricing models for consuming a cloud service.
- **Foundation subontology**, describes the general concepts (e.g., artifact, resource, location, time). Additionally, it addresses the resources control and visibility, the dynamic changes in the environment, and the environmental constraints.

This description can be used for automated service selection, based on characteristics and QoS constraints, expressed in Semantic Web Rule Language. In the same work, an algorithm for selecting a combination of clouds is introduced, which considers both service semantics and multi-cloud settings, optimizing the usage of Multi-Cloud Environments.

Requirements that have to do with other resources, such as Edge Computing and IoT, can be expressed through the recently standardised W3C Web of Things. The Web of Things is not just a trend, but also a concentrated effort by W3C to create standards that would reduce the fragmentation of the IoT domain, by unifying many terms into a small set of ontologies. In the WoT architecture [17], the way to describe, expose, as well as consume a "Thing" are laid out. As such, terms from the WoT Thing Description [18] and Binding Templates [19] may be useful for laying out application requirements that are satisfied from special IoT equipment at the Edge.

In the realm of linked data and open vocabularies, there are not existing ontologies that include terms about the kind of programming workflows that Node-RED and PHYSICS have per se. However, there are ontologies that define class hierarchies and characteristics of more generic workflows, such as dataflows, which most typically are Directed Acyclic Graphs. The modelling of the PHYSICS application class hierarchy is based upon Node-RED, using the following external ontologies as a guide, with the goal of connecting to or reusing terms from related ontologies wherever possible.

The Visual Modelling Tool Model (vmm for short)[3] defines a vocabulary that includes the characteristics of a modelling tool and originates from the study of UML software modelling tools. Although vmm falls outside the scope of the application description, it provides a guide for modelling terms related to application workflow design, in a more abstract and reusable way. The Wfdesc ontology [20] describes an abstract workflow description structure and it is meant as an upper ontology for more specific workflow definitions and a way to express abstract workflows. The Wfprov ontology [21] helps link the descriptions of Wfdesc, to form a provenance trace of the execution of a workflow. The invocation of the steps of a workflow execution are described by the Workflow Invocation Ontology [22], which provides useful insights in the representation of an actual workflow execution. The kinds of data-intensive activities that are found in workflows, which are essentially data operations, and the ways in which the activities are implemented within these workflows, are found in the Workflow Motif Ontology [23]. Linking the workflow-related ontologies with occurrences of equivalent terms in the manifests of FaaS platforms and visual workflow programming tools appears to be possible and may prove beneficial in the semantic modelling of applications in PHYSICS.

In general, reusing terms from these ontologies, and following the ways their terms are linked, can help improve the descriptions of workflows and application patterns and their components. The ensemble of these works provides insights as to how to organise semantic terms within the PHYSICS application semantic models.

2.3 Cloud Design Patterns Repository

In the following paragraphs, related work is presented around common Cloud Patterns in Function-as-a-service (FaaS) platforms and microservice environments that covers various areas of application

[3] https://lov.linkeddata.es/dataset/lov/vocabs/vmm

development such as computational patterns, security patterns, microservices patterns, user authentication patterns, ai-machine learning patterns and more generic serverless design patte rns.

2.3.1 Function-as-a-Service, Cloud patterns

As PHYSICS aims to provide an environment that will make application development easier through visual editors and make the same applications work efficiently by optimizing the execution and cost of a service, it is necessary to provide some common, reusable design and cloud patterns. These patterns will be common programming patterns, or architecture patterns, that an application developer usually needs (such as computing models, data encryption, security architecture, request management etc) and they will be available through PHYSICS development environment. The implementation will be in a native function model which is expected to harvest more efficiently the relevant capabilities of the FaaS layer. I t is important to mention that these patterns must be implemented on languages and frameworks that are supported by the FaaS framework (eg., OpenWhisk [24]) and it should be noted the efficiency of each language for certain tasks [25]. Following we will analyse several candidate patterns that will be considered for the PHYSICS development environment.

2.3.2 Computational patterns

In today's data-drive applications it is common to collect great amounts of data daily from multiple sources which then requires processing and analysing. MapReduce, a programming model for data intensive computing inspired by functional programming [26], is one of the most common programming models for analysing large-scale data. Regarding the implementation of MapReduce models for serve rless computing, few works could be found in the literature. AWS has its own reference architecture for serverless MapReduce [27] in which the user provides the mapper and reducer functions, a Coordinator Lambda function that orchestrates the execution, and the already partitioned input data. Additionally, more libraries and frameworks for serverless MapReduce have been implemented such as Ooso [28] and Corral [29]. Finally, a more complete framework has been proposed in MARLA (MApReduce on LAmbda) [30]. MARLA is a framework designed to work on AWS Lambda and allows the execution of Python -based MapReduce jobs without requiring pre-partitioning of the input data. In MARLA the coordinator Lambda calculates the optimal size of the data partitions taking into consideration the user-defined number of chunks and invokes the first mapper Lambda function with an environment variable that stores the dimension of each data chunk size. The first mapper Lambda function starts a logarithmic reduction approach so that it invokes the second mapper and then, each Lambda function will invoke another two. This procedure is recursively repeated until all mappers have been invoked.
The above model can be extended in such a way to offer frameworks and services that will support g eneric parallel serverless execution of functions, providing the application developers the capability to just write the functional code and the FaaS platform will be responsible to parallelize the execution and aggregate the results.

2.3.3 Security related patterns

As in Cloud Computing, and in extension Serverless Computing, exists a shared -responsibility model regarding the security of applications. Often, cloud providers provide tenants with security services and patterns to integrate on their workflows but also invest in securing their own infrastructure. Regarding the applications developed using the FaaS model, securing the code running in a function and the data coming in/out of it, complying with the corporate and privacy rules is responsibility of the ap plication developers. Aditionally, an important aspect on FaaS, unlike Cloud computing applications, is to provide mechanisms that detect and prevent attacks leveraging on incorrect function execution order to subvert application logic. A framework has been proposed in [31], SecLambda, which covers the peculiar security needs of serverless applications (exg Flow control) and gives the developer the capability to define custom security functions to protect the application. SecLambda consists of three main components:

- *Secure runtime:* A function runtime that generates events and reports them to the Guard component. An event might be send/receive messages, SSL/TLS connections or create/modify/delete files. The runtime blocks an operation until it receives a decision from the guard and enforces the decision (ALLOW/DENY).

- *Controller:* The controller provides an interface for the application developer to manage security functions, policies, and configurations.
- *Guard:* The guard is a runsec module which executes a set a set of security functions created by the application developer to process the received events based on the application security policies

This framework can be used to implement security functions for modelling and monitoring application behaviours, obfuscating credentials in requests, rate limiting and even data encryption.

A more general approach is mentioned in [32] where the use of 6 main Serverless Design Patterns (periodic invocation, event driven, data transformation, data streaming, state machine, and bundled pattern) are proposed as a Serverless Threat-Intelligence Platform that analyses various data sources in the cloud, notifies suspicious events and takes responsive actions against them.

In the context of cloud design patterns for security, all major providers define some design patterns or specifications for secure and well-functioning cloud applications [33][34][35][36]. Although these patterns are very useful when it comes to securing an application and must be taken into consideration, there is a lack of cloud design patterns that focus on providing security and privacy functionalities as serverless components. These components could provide easy to use abstraction for encryption, privacy blurring of fields, management of secrets and other services in the same context. These components can significantly enhance the usage of security in such applications both in terms of ease of use but also from a functional correctness viewpoint.

Furthermore, there are some examples [37][38][39] of forensics as a service. These examples are especially useful given the nature of the forensics science and the methodology of forensically correct collection of data. They include methodologies for provable correct collection, privacy and encryption that, if broken down, could prove useful in creating security-focused cloud design patterns to be used in serverless applications

2.3.4 Microservices patterns

In recent years microservices-based solutions are gaining more and more momentum in application development as an extension of service-oriented architecture (SOA) by enabling high service reusability, reliability, improved scalability and availability, heterogeneity, and platform independence [40]. As microservices break the logic of an application into a number of independent services that run as separate processes, it is quite obvious that their very nature is aligned with the FaaS paradigm. Common microservices patterns [41] are:

- API Gateway Design Pattern: A gateway pattern which exposes a number of sub-services as a single entry point
- Chain of Responsibility Design Pattern: This design pattern consists of a collection of sub-services designed to work together with a specific order in order to process a request.
- Asynchronous Messaging Design Pattern: Pattern that enables asynchronous messaging and event-driven communications.

More microservices patterns are proposed at VSLive blog [42] such as Proxy Microservice Design Pattern, Branch Microservice Design Pattern and Shared Data Microservice Design Pattern.

2.3.5 User authentication patterns

In modern web applications the need for user authentication and management is a common issue with many different approaches. Regarding FaaS applications it is important to note that the authentication patterns must be stateless as FaaS itself is stateless. Perhaps one of the most common techniques for stateless authentication is JWT tokens, where after user authentication (for example user login), each HTTP request will carry the generated JWT token to the endpoint and the token will be validated from a serverless function. A key advantage of JWT tokens is that it can also be used to store additional user information directly in the token, not just the access credentials.

2.3.6 Serverless design patterns

A basic feature of serverless environments is to provide the ability for an application developer to compose, orchestrate and execute serverless functions in sequences or workflows based on data (for example based on a message or an event). In the literature we can find frameworks [43] that propose solutions for sequence management where a workflow management system orchestrates the function execution and is responsible for maintaining the appropriate execution order. Another approach is for each serverless function to know which is the next function that should be called, based on a decision, and so to create a function chain where each serverless function, when finished execution, calls the next one.

2.3.7 General Cloud patterns

A very thorough and interesting catalogue of design patterns in cloud environm ents is detailed in [44]. This includes patterns separated in three large categories (Data Management, Messaging and Design and Implementation) and relate to how the services may expose interfaces, functionality organization, service design etc. Some of them are already existent in typical FaaS environments (e.g. Throttling for limiting number of concurrent function invocation, Queue-based load levelling through the inclusion of Kafka messaging in a typical OpenWhisk architecture setup etc.) or aim to cover for limitations of these environments (e.g. Function Chain in order to continue a function execution after the maximum function execution time is reached). Interesting patterns can also be extracted from the literature regarding parametric service creation for adaptation to dynamic request patterns [45] or from a combination of request management and FaaS platform configurations for minimizing other issues such as the cold start problem.

2.3.8 Artificial intelligence and machine learning patterns

As AI and machine learning are some of the major technology trends, work has been done to propose solutions for AI development on Cloud and Serverless platforms. The main goal is to provide an easy-to-use framework for developing AI and machine learning applications without having developers worry about scalability and provisioning. Such framework has been proposed in [46] where special function annotations and a data API hide platform data management tasks from the developer. Furthermore, some techniques have been proposed [47] to transform existing AI applications with standard architectures into applications that work on Serverless platforms. These techniques have to do with 1) Reducing the footprint of the AI libraries and frameworks used in application codebase, 2) Dynamic loading and injection of AI models into temporary container runtime memory 3) A 2-Step Framework ML Process (train and running machine learning models with different frameworks) and 4) Improving the handling of data lookup and storage, through innovative partitioning and indexing techniques. With the aforementioned techniques, a pattern can be implemented to provide developers a way to easily build AI and Machine Learning applications without worrying about Serverless platform's restrictions.

2.3.9 Orchestration patterns and frameworks

Function orchestration (or function choreography as more specially mentioned) is the ability to regulate the execution of a more complex logic involving a number of functions, based on a business logic and scenario. For this reason, a number of frameworks are available for dictating and executing function orchestrations.

From a visual workflow creation point of view, Kubeflow [223] includes a relevant language for pipeline definition and an editor extension for visual definitions of workflows. The defined workflow resembles more to a static definition of steps, without a respective runtime, orchestrating the execution of one task after the other, while the inputs and outputs are passed through external object storage services. AWS Step functions supports visual programming style and extended operators for function workflows (e.g. state management ones) however it is tied to a single provider (AWS Lambda). Google Cloud Functions are based on text based yaml files for the definition of a workflow. The same yaml approach applies for the AFCL approach presented in [224]. In general yaml based approaches can become very complex when the size or connections in the workflow scale, although in this case the solution comes also with a rich set of available constructs that can significantly speed up application creation.

[225] has moved the execution of scientific workflows to a FaaS model with Hyperflow and compares it to the traditional IaaS approach from a cost point of view, while it proposes a number of architecture alternatives for workflow orchestration. One commendable research effort was the extension of Hyperflow to support execution of workflow tasks with AWS Lambda and Google Cloud Functions, where

performance and cost was impacted by size and priority of cloud functions.In [226], SWEEP, a cloud-agnostic workflow management system, which is built on the serverless execution model, is introduced and evaluated. Throughout the findings and results it is clear that it allows the direct mapping of functions and containers to tasks in workflows, while providing a layer of structure and orchestration on top of serverless execution frameworks. Based on the aforementioned, works like [224] with AFCL introducing programming for workflows on a high-level of abstraction for reducing invocation delays and avoidance of delays of blocking functions, and Beldi [227], a library and runtime system that makes transactional and fault-tolerable workflows possible without having to deal with load-balancing, have been conducted in this sense. In [228], Durable Functions, which is a programming model for serverless workflows, and Netherite which is a distributed execution engine, are proposed, while they provide interesting results on Throughput and Latency aspects.

2.4 Elasticity Controllers

The elasticity controllers will be integrated into the infrastructure layer (through the API extensions provided in resource manager controllers (task T5.3). They should be abstracted so that they can be configured and used in different use cases.
Currently Kubernetes has different scalability controllers, two for the application abstraction layer (the *Horizontal Pod Autoscaler* and the *Vertical Pod Autoscaler*) and one for the infrastructure layer (Cluster Autoscaler).
This task focuses on the application layer, and more specifically on the number of replicas, therefore on the horizontal one. In a nutshell, the Horizontal Pod Autoscaler (HPA) [48] automatically scales the number of pods in a replication controller, deployment, replica set or stateful set based on observed CPU utilization. It is implemented as a Kubernetes API resource and a controller. The behaviour of the controller can be adjusted through the API and loops its functionality periodically (with a default value of 15 seconds). Hence, HPA is a tool to ensure that critical applications are elastic and can scale out to meet increasing demand as well scale down to ensure optimal resource usage.
There are different types of metrics that HPA can use to perform more accurate autoscaling actions [40]: The default "*per-pod resource metrics*" (i), i.e., CPU, which are obtained from the resource metrics API for each pod. Upon target utilization set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod. The controller then calculates the average utilization across all targeted pods and generates a ratio that is used to scale the number replicas. The "*per-pod custom metrics*" (ii), similar to the per-pod resource metrics but working with raw values instead of utilization values. The "*external metrics*" (iii), using them allows the cluster to autoscale applications based on any metric available in the monitoring system. This is useful for applications running on Kubernetes that may need to autoscale based on metrics that do not have an obvious relationship to any object in the Kubernetes cluster.
However, there may be other metrics that would be interesting to include, such as the **"price".** For instance, when multiple deployment options are available, select the one which is cheaper for the customer. This metric is useful mainly for public cloud deployments. This can be achieved by better packing pods into instances, using ML based metrics on the relations between pods resource consumption (rather than using max resource consumption) and by choosing the cheapest instance types that fit the requirements.
The controller manager works as a control loop by quering the resource utilization against the metrics specified in each HorizontalPodAutoscaler definition. It obtains the metrics from either the resource metrics API (for per-pod resource metrics), the cloud provider API (for cost calculations) or the custom metrics API (for all other metrics). For object metrics and external metrics, a single metric is fetched, which is compared to the target value to obtain the desired/needed ratio. The ratio is obtained by the Horizontal Pod Autoscaler controller by using the next formula:

*desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]*

In addition to the custom metrics, Kubernetes also has support for configurable scaling behaviour as well as for multiple metrics (autoscaling/v2beta2 API). For example, for the former, it is possible to configure autoscaling to skip certain actions when it is "close" to the target by configuring the tolerance flag. Related

to multiple metrics, the HPA controller will evaluate each metric, and propose a new scale based on that metric.
On the other hand, the scaling actions can also be made for container specific metrics (unlike pod specific metrics), i.e., by tracking individual containers across a set of pods. This let us to configure scaling thresholds for the containers that is more important in a particular pod. For example, a web application and a logging sidecar can be scaled based on the resource usage of the web application, ignoring the sidecar container and its resource usage.
There are extra mechanisms that together with HPA can be used to ensure that other controllers (such as the cluster autoscaler one) do not remove pods required for a given application. To that end *Pod Disruption Budget* [49] can be utilized. It ensures nodes with specific pods of an application cannot be removed until those pods have been scaled out to a different node. This avoids disruptions to critical pods and ensures that a desired number of them is always running.
The PHYSICS project will leverage and enhance HPA to provide extra optimized controllers based on external/custom metrics. An important aspect which is today outside of Kubernetes scope is the total cost of ownership for a specific cluster, machine sets, or pods. In public clouds, this cost can be very critical for the customers. In public clouds when we look at the set of deployed pods as a network, we can cover this network by multiple combinations of instances (usually VMs, but it can also be bare metal machines), with different performance characteristics and different cost. As part of the PHYSICS project, we want to minimize the cost of ownership by using some additional mechanisms:

- ➢ The sse of pod state size-aware algorithms for executing more pods on some nodes (e.g., preferring AWS spot instances). This is simple for stateless pods (pods that do not keep state on the hosting instance, hence pod-state-size is zero), but it becomes more challenging for stateful workloads. The project aims to carefully control the pods state and use horizontal scaling to run additional workloads on spot instances. Preliminary work in this direction (for a long running computational batch process use case) is implemented by the *FastFreeze* project [50].
- ➢ Create deployments that will assure that the network will not become the bottleneck of the performance. This require locating the components which communicate with each other in the same instance and deploy them on instances with smaller network bandwidth (typically smaller and cheaper instances).

2.5 Inference Engine (Reasoning framework)

As far as the Inference Engine is concerned, both the input and the output of the model should be defined. As input of the Inference Engine, two semantic models (i.e., T3.2, T5.1) will be considered: the application description and the available cloud resources. The output will be a *deployment graph* where each function of the application will be connected to certain resources capable of running the given function. Thus, the inference engine objective (task T4.1) is to produce a knowledge graph (KG) which given a fact *function(predict, 128mb)* and r*esource(e2micro, 1gb)* would be able to complete the missing link *ableToRun(predict, e2micro).* In addition, it is worthwhile mentioning that, to the best of our knowledge, there are no regulated ontologies for FaaS and Serverless applications and as a result, the semantic descriptions will be developed in the context of PHYSICS. Additionally, the Deployment Graph should be able to be updated during the runtime as properties regarding the performance of the deployed application change.
The Deployment graph will be completed as a Resource Description Framework (RDF) triple store. In RDF, data is linked via a subject-predicate-object structure (Figure 3), where the subject is a node, the predicate is an edge, and the object is either another node or a literal. Multiple triples could be modelled as a graph of data consisting of nodes and edges, named by a Unique Resource Identifier (URI) which typically is an HTTP URI. A unique asset of RDF is that the HTTP URIs can be published on the world wide web, and therefore be used by others. In the context of Semantic Ontologies, SPARQL has been adopted as the standard query language based on triple patterns. Being a global standard, RDF is the most frequently used graph model in the smart home domain and has been used in cases related to energy [51], home automation [52][53], health [54], activity-recognition [55] and IoT integration [52][56].

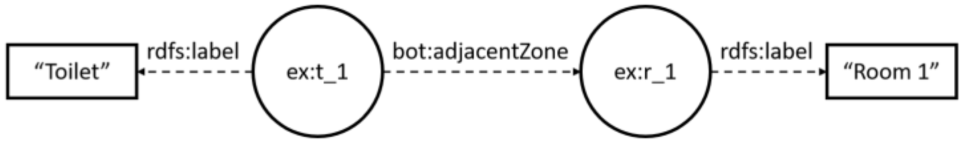


Figure 3 - An example of linked building data using an RDF model

RDF stores typically support ontology modelling languages such as RDFS and OWL2 RL. The scope of the ontologies' usage is twofold. Firstly, they may serve as an interoperability framework where different stakeholders model their data. Secondly, ontologies store domain knowledge that allows machines to do inferences to better interpret the data or to derive new insights from them. The top well-established RDF stores in terms of community adoption are listed in Table 1.

Table 1 - Summarization of Triple Stores

Name	**Description**	**License**	**API**
GraphDB [57]	Enterprise-ready RDF and graph database with efficient reasoning, cluster and external index synchronization support. It also supports SQL JDBC access to Knowledge Graph and GraphQL over SPARQL.	Free version is limited to two concurrent queries	GeoSPARQL, GraphQL, Java API, JDBC, RDF4J, RDFS, RIO, Sail API Sesame REST HTTP Protocol, SPARQL 1.1
Blazegraph [58]	High-performance graph database supporting Semantic Web (RDF/SPARQL) and Graph Database (tinkerpop3, blueprints, vertex-centric) APIs with scale-out and High Availability.	Open Source	Java API,RESTful HTTP API SPARQL QUERY, SPARQL UPDATE, TinkerPop 3
Stardog [59]	Enterprise Knowledge Graph platform and graph DBMS with high availability, high performance reasoning, and virtualization	Commercial	GraphQL query language, HTTP API, Jena RDF API, OWL, RDF4J API, Sesame REST HTTP, SNARL,SPARQL, Spring Data, Stardog Studio, TinkerPop 3
Virtuoso [60]	Virtuoso is a multi-model hybrid-RDBMS that supports management of data represented as relational tables and/or property graphs	Commercial	ADO.NET, GeoSPARQL, HTTP API, JDBC, Jena RDF API, ODBC, OLE DB RDF4J API, RESTful HTTP API, Sesame REST HTTP SOAP, SPARQL 1.1, WebDAV XPath, XQuery, XSLT
JanusGraph [61]	A Graph DBMS optimized for distributed clusters	Open Source	Java API, TinkerPop
Apache Jena - TDB [62]	An RDF storage and query DBMS, shipped as an optional-use component of the Apache Jena framework	Open Source	Fuseki, Jena RDF API, RIO
AnzoGraph [63]	Scalable graph database built for online analytics and data harmonization with MPP scaling, high-performance analytical algorithms and reasoning, and virtualization	Commercial	Apache Mule, gRPC, JDBC Kafka, OData access for BI tools, OpenCypher, RESTful HTTP API, SPARQL
AllegroGraph [64]	High performance, persistent RDF store with additional support for Graph DBMS	Commercial	RESTful HTTP API, SPARQL

Furthermore, a semantic reasoner will be utilized to infer the required logical consequences for the development of the Deployment graph. For instance, after loading the ontologies (i.e., application, resource) to the Inference Engine, queries (i.e., which resources are gpuEnabled?) can be answered using one of the available reasoners.
Current reasoners can handle a comprehensive set of RDFs and OWL vocabularies and most RDF data formats. A reasoner concludes facts from semantic data and ontologies based on predefined rules. Common reasoning and inference engines such as Jena Inference subsystem [62], Pellet [65], RacerPro [66], HermiT [67], RIF4J [68] , and Fact++ [69] are based on different rule languages and have support for ontologies and

OWL. The triple stores described in Table 1 provide built-in reasoning engines, while some of them are able to work with external reasoners.

In recent years, there is surging interest in designing machine learning / deep learning ML/DL algorithms for complex reasoning tasks, especially in large KGs where the countless nodes and links have posed great challenges to traditional logic-based algorithms. Specifically, various deep neural networks (DNNs) architectures have been leveraged as link prediction models for the completion of knowledge graphs [70]. These methods [71][72][73][74][75][76][77] heavily rely on the subsymbolic representation of entities and relations learned through maximization of a scoring objective function over valid factual triples. Thus, the current success of such deep models hinges primarily on the power of those subsymbolic continuous real-valued representations in encoding the similarity/relatedness of entities and relations. Recent attempts have focused on neural multi-hop reasoners [78][79][80][81][82] to equip the model to deal with more complex reasoning where multi-hop inference is required. More recently, DeepPath [83] and MINERVA [84], frame the path-finding problem as a Markov Decision Process (MDP) and utilize reinforcement learning (RL) to maximize the expected return.

2.6 Metrics & Monitoring

In the following paragraphs, related work is presented around Function-as-a-Service (FaaS) benchmarking research that has already been conducted (or is currently on-going) related to performance monitoring and cost efficiency, research on container performance comparison and benchmarking (with specific metrics) but also current open issues.

2.6.1 Function-as-a-Service, Cost models, Metrics & Benchmarking

The number of available FaaS platforms increases with the frequent tendency nowadays to use serverless architecture, thus, a high demand for benchmarking FaaS platforms exists. In the paper [85] a literature review is presented in support of benchmarking FaaS platforms. Due to extremely rapid development and slow publication in FaaS, a lack of benchmarks is observed for measuring effects on key points such as cost efficiency as well as benchmarks that observe functions not isolated from the whole but in the composed environment of a cloud service.

The use of FaaS environments is delivered by many cloud service providers nowadays. Although the problem is that it is hard to measure performance of Cloud Services because they behave like black boxes. In papers [86], [87] an architectural design of a new FaaS benchmarking tool is presented which allows users to evaluate their cloud functions performance, since users need to clarify whether more resources are needed to deliver services in higher qualities. This benchmarking framework consists of two components, a Java-based application and a JavaScript proxy cloud function. This framework can be used to identify limitations and restrictions on top of the FaaS infrastructure helping the cloud service consumers to evaluate and identify a more holistic view of the performance of the platforms, aside from typical benchmarking.

Another simulation tool, SimFaas, is proposed in [88] which aids developers to create optimised FaaS applications in terms of cost and performance. The simulator is written in Python language and available on GitHub. Use cases for the simulator include "Steady-State analysis", "Transient Analysis", "What-If analysis" and "Cost Calculation", plus an experimentation on AWS Lambda. SimFaas benefits include i) validating new ideas for cloud service providers in SimFaas before applying them (which is cheaper in both cost and time aspects) ii) providing users with fine-grained control over the cost-performance trade off by alternating the platform parameters.

In [89] authors present a microbenchmark to monitor 2 aspects of FaaS: i) Differences in observable behaviour related to memory of each FaaS implementation by the providers ii) Complex pricing models currently being in use derived from the number of function invocations across functions belonging to the user together with function execution duration. Three very common algorithmic tasks (Fast Fourier Transformation, matrix multiplication, and a simple sleep as a baseline) are applied to this matter, which are implemented on the Node.js environment as the common denominator across the FaaS solutions under consideration. The results provide some insights on the relationship between cost and performance on a cloud environment.

FaaS models transition from singular applications to compositions of smaller services that are more granular and distributed. Thus, new FaaS models offer many new opportunities while raising new performance challenges. [90] identified six performance-related challenges related to FaaS modelling from a performance engineering aspect: reduction of performance overheads, performance isolation, scheduling policies, performance prediction, cost-performance engineering, evaluation and comparison of FaaS platforms through suitable benchmarks. [91] also tries to list some performance-related challenges along with a proposed approach to solve them. Most notable challenges are: 1) Software engineering challenges (meaning developer experience can be overcome by better testing, tooling, functionality and education) 2) System (operational) challenges that emerge mostly from the dynamic characteristics of Serverless models. A proposition made is an improvement in cost prediction models as well as lifecycle management. 3) Performance engineering challenges which are unwanted overheads, questionable performance and diversity in cost-performance solution models.

With FaaS models being our prior and outmost general interest, we cannot ignore microservice architecture technology. Microservice is defined as an architectural style for software design as opposed to the monolithic style. Thus, software applications are dissected to smaller, simpler functioning components that communicate with each other throughout network requests. [92] draws a detailed analysis and summary on performance optimization, design approaches and open issues for microservices. PHYSICS could most likely benefit from the topics of benchmarking approaches, performance optimization techniques and monitoring of anomaly detections from this article.

[93] addresses the research issues behind FaaS. A large literature gap (mismatch between academic and industrial sources on tested platform configurations) is identified while providing recommendations that can take place immediately. The paper's main findings are that AWS Lambda is the most evaluated platform , microbenchmarks are the most common type of benchmarks and that application benchmarks are currently evaluated on a single platform.

[94] is another paper referring to performance evaluation metrics, benchmarks and open issues in the general spectrum of Cloud Computing. The aspects of resource allocation, resource provisioning, task scheduling, load balancing, task placement, data caching and service discovery are some of the topics that play an important role. One interesting point lies in the categorization of metric benchmarks in 1) Monitoring - related metrics (resource load, throughput, resource lifetime, maximum running resources, response time, fault tolerance, energy consumption etc.) 2) Analysing-related metrics (MAPE, MAE, MSE, R^2, RMSE, average, median) 3) Planning-related metrics (adaptation time & scalability, decision making, competition Ratio) 4) Execution-related metrics (provisioned and de-commissioned resources metrics, technique overhead or lightness, contradictory actions). Many of those could be used (as base, and then adapted) to test the project functionalities, depths and limits.

2.6.2 Containers, Benchmarking & Metrics

Virtualization is nowadays the core component of cloud computing that allows multiple tenants to run their heterogeneous applications in isolated environments. Containers are a solid viable alternative for VMs in cloud infrastructure services as they provide virtualization advantages with near bare-metal performance as they bind all the necessary software in the form of images that can be easily deployed in any environment. In [95] an experimental study is presented on the performance evaluation of Docker containers running a heterogeneous set of microservices concurrently, providing more insights on their power and limits. The extensive testing was materialised with many benchmarking tools (Linpack, STREAM, Bonnie++, Netperf) looking at CPU performance, memory evaluation, disk I/O evaluation and network performance (floating point operations per second, data and disk throughput, random seeks, network throughput).

Another interesting publication [96] conducts experimental evaluations on containers to investigate their performance, while running on hosted , cloud-native managed Kubernetes environments offered by cloud providers, by monitoring system resources such as CPU, memory, disk and network with the benchmarking tools and metrics for each resource (SysBench for CPU load, Y-cruncher for CPU load + efficiency, STREAM for data throughput, SysBench for data throughput with random I/O, Bonnie++ for data throughput with sequential I/O, Nuttcp for data throughput and Netperf for network latency). The different cloud environments that are under tests consist of Amazon Elastic Container Service for Kubernetes (EKS), Azure

Kubernetes Service (AKS), Google Kubernetes Engine (GKE). The gain on this paper is twofold: a good evaluation on the functionalities of Kubernetes with different service providers, (presenting pros and cons on the environment to be used) and the importance of the used container benchmarks as a basis for our work in the benchmarking aspect.
An extensive literature review on container technologies and their related challenges is presented, as well as performance (in comparison to VMs and bare-metal performance), orchestration (deployment and dynamic control of multi container packaged applications) and security (container isolation, confidentiality of containerized data, and network security). Performance prediction models and scheduling challenges for production of optimal schedules (cost efficiency) are also presented.
Overall, environments like serverless computing tend to evolve at the speed of light nowadays, although the research and applications related to them are still at an early stage. This is easily notable the lack of solid benchmarking tools focused on FaaS environments, absence of highly accurate cost models and on the unsolved open issues that exist.

2.6.3 Orchestration overheads examination

Orchestration overheads intersect two out of the 3 performance challenges identified in [229], namely the request overheads and the function lifecycle management aspects. One exception investigating performance issues in depth is Netherite [228], in which a distributed executi on engine is presented. Netherite applies speculation for minimizing delays from state management, which aids in increasing the orchestration throughput and workflow latency.
In [230], a performance analysis is conducted for fork-join executions between Amazon Step Functions, Azure Durable Functions and IBM Composer, in the form of overheads from multiple concurrent executions. In [231], the pure orchestration needs are measured in sequences of operations for the same providers as [230]. Interesting findings are reported with relation primarily to the state transition delays affected by the function input size and how this affects the total overhead. Other approaches investigate more futuristic implementations deploying the orchestrators at the Smart NIC l evel, for minimizing latency in function orchestrations [232].

2.7 Global Continuum Placement

PHYSICS will provide a uniform access, management and optimization layer for the usage of the underlying hybrid edge-cloud computing infrastructure. In this context one of the important tasks is the workflow placement across the distributed and diverse edge and cloud resources along with possible optimizations and needed adaptations. In this context, this section provides a first analysis regarding the state of the art of workflow placement upon the global continuum.
This study [97] upon orchestrators discusses the various advances that have been made regarding scheduling. Kubernetes [98], [99] and Mesos [100], two of the most advanced open-source orchestrators. Kubernetes orchestrator enables the support of Software Defined Infrastructures and resources disaggregation by leveraging on container-based deployments and particular drivers based on standardized interfaces (Container Runtime Interface [101], Container Storage Interface [102], Container Network Interface [103] and the device plugins framework [104]). These interfaces enable the definition of abstractions for finer-grain control of computation, state, and communications in multi-tenant environments along with optimal usage of the underlying hardware resources.
However, even if Kubernetes is today production-level for typical cloud data centres, the default distribution is not adapted for the constrained edge capabilities nor for multi-cluster deployments, such as integrating different layers of compute resources (edge, fog and cloud). Efforts are currently ongoing to better adapt Kubernetes for the edge, such as those done by the IoT-Edge working group [105] which are mainly focused on guidelines and best practices with the current ecosystem. The opensource proje cts kubeedge [106] and virtual-kubelet [107], driven by vendors provide interesting designs to incorporate the edge with the cloud using Kubernetes, enabling a simpler integration of cloud and edge based on a group of complexity abstraction features. Opensource solutions such as k3s [108] where Kubernetes heavyweight internal procedures have been stripped down is another alternative that simplifies the autonomy of single edge devices using the same Kubernetes API. Another opensource alternative that could b e interesting for

the deployment of individual autonomous edge resources is Canonicals' microk8s [109] which can be evaluated for the mobile edge resources case, needing to orchestrate tasks and workflows autonomously when disconnected from network. Further analysis is needed as we go forward on determining the best tools to be used for the edge resource management.

In a similar way, the multi-cluster special interest group (SIG) community of Kubernetes works on the federation v2 project, named kubefed [110] which focuses on integrating multiple clusters under a federation while providing a generic scheduling engine that, based on policies, can make decisions on how to place arbitrary Kubernetes API objects. The project is currently under development, but th e goal is to eventually support the execution of workloads across multiple clusters which is one of our requirements. Other alternatives that could be used to provide the communication layer across services that run on multiple Kubernetes clusters are Submariner [111] and Istio [112] but they do not offer the capabilities of scheduling which has to be provided apart. The cloud federation reference architecture by NIST [113] can be also considered as a base when designing the architecture of our global continuum.

Asuncao et al [114] studied resource management challenges regarding hybrid deployments including IoT and Edge. They consider that managing task scheduling and allocation of heterogeneous resources along with adapting an application to current resource and network conditions will require the development of new schedulers and that allocations must be dynamic enough to support migration. Yuan in [115] does a thorough study of task scheduling in hybrid edge-cloud environments and proposes various algorithms optimizing energy, performance and cost while considering various constraints such deadlines and Q oS.

Maia et al in [116] considered the offline placement problem of IoT services supporting horizontal and vertical scaling in a hybrid edge-cloud environment. They tried to solve the joint problem of service placement and load distribution to minimize the delay in execution (QoS). They formulated the problem with Mixed-Integer Linear Programming (MILP) method which has high computational complexity, so they considered solutions either through MILP approaches for optimal solution or greedy and genetic methods which provide good placement with less computational complexity and faster results. In our case we would like to consider both offline and online placement optimizations.

Wang et al in [117] proposed another genetic-based scheduling algorithm under a deadline constraint to enable the execution of tasks upon edge or cloud resources with a goal to minimize the execution time of all tasks. They have enhanced the genetic algorithm with a catastrophic variant to increase the mutation probability to stay away from the current optimal, which may not be the ideal optimal solution, and allow a re-optimization in a later stage. They have provided experimentations and performance evaluations of their algorithms upon the CloudSim simulator.

We consider that scheduling on the global continuum layer is considered as a meta-scheduling to take the high-level decision on how the different components/patterns of the workflow need to be placed across the available platforms. Different ways to deal with this environment can be studied but the most adapted would be to consider a two-phase scheduling where the high-level decision will be taken on the global continuum layer while the second scheduling will be done on each local scheduler. In that scenario the final scheduling will be based on the meta-scheduling proposition which will be directed to the local schedulers of each platform to perform the local placement considering the local characteristics and availabilities while trying to fulfil the global scheduling proposition. In addition, we need to consider that the applications to be executed will be (at least partially) based on serverless architecture (FaaS programming mod el).

The execution of serverless workflows [118] will stress the scheduler of the system since a large number of tasks will need to be mapped to resources. This can be also considered by the orchestrator and provide optimized high throughput techniques which will minimize the scheduling time while respecting constraints such as data locality.

Das et al in [119] presented scheduling algorithms and a framework to execute serverless applications over a hybrid public-private cloud in a way to minimize the cost of public cloud use, while remaining under a user-specified makespan constraint. The choice of how to offload tasks to the public cloud is made based on their order in the priority queue considering the following methods: Highest Cost First order (HCF) and Shortest Processing Time order (SPT). The experimentation took place using 3 different serverless applications dealing with matrix, image or video processing in each case, while the platform was composed

out of OpenFaaS for the private cloud and AWS Lambda for the public cloud. Their results showed that their algorithms and framework achieved a speedup of more around 1.8 times (in average) for matrix and video processing applications over an approach that uses only the private cloud, at a cost that is ar ound 40% (in average) of an approach that uses only the public cloud.
Another important aspect that needs to be considered for the global continuum scheduling is adaptability and the capacity to perform adjustments on placement based on new parameters. Kub ernetes has an inherent support of self-healing and auto-scaling, but it cannot be leveraged on the federation level amongst clusters. To enable this a cross-layer federated monitoring strategy will be needed along with strategies on how to decide which cluster to select for the offloading and how. The orchestration techniques [120] to optimize autoscaling within Borg the predecessor of Kubernetes at Google can be used as a reference in our investigation for autoscaling improvements.
Of course, the study of scheduling algorithms cannot be performed efficiently without having ways to experiment and evaluate the developed strategies in simulated environments. Addressing the challenge of cloud and edge computing simulations, another study and platform, develop ed by a member of Ryax team, is a simulated edge platform developed on top of Batsim/SimGrid along with several scheduling policies solutions upon a real use-case [121]. This solution was used to simulate a real edge platform use-case based on smart heaters allocated over smart buildings. The simulated infrastructure was composed by nodes representing the smart buildings and computational resources representing the smart heaters. They developed two-level scheduling policies that first decide the building th at a job should be executed and then to decide in which smart heater. Upon such platform they proposed three different scheduling policies based on data locality and compared them with the default policy applied in the initial smart buildings use case as a baseline. To perform the comparisons, they used several workloads extracted from the real platform. In addition, Batsim allows the injection of external events and plugins. The events can abstract real world behaviours for edge platforms such as loss of connection to some nodes or more nodes becoming available. The plugin can add more features to the platform, such as a storage controller plugin that dealt with the data movements. Paper [122] depicts the implementation of the simulator and provides more information about other cloud and edge platform simulators, scheduling policies and metrics. We foresee to use a variation of the above simulator, adapted to our context, in order to simulate our new scheduling strategies for the global continuum in PHYSICS.

2.8 Distributed Memory Service

FaaS allows building and deploying applications in the cloud in which the unit of computation is a function that the cloud provider scales as needed. The application developer concentrates in the business logic which mainly consists of stateless functions with minimal I/O and communication. Other limitations include the maximum memory allocated to functions and the time to execute a function for instance, AWS Lambda limits the execution time of a function to 15 minutes and maximum memory to 3GB. Moreover, state across function invocations cannot be shared unless the state is made persistent using a remote storage service which worsens performance. Transforming legacy applications to follow this model is challenging.
The In-memory Distributed State Service will oversee maintaining state across function invocations in memory, therefore avoiding the latency penalty of persistent storage. The cache will keep the state in memory so that it can be quickly accessed between fu nctions invocations. The cache must be distributed, and its state replicated so functions can quickly access the state regardless of where the function is executed. Elasticity is also a requirement for the in-memory distributed state service in order to be deployed in a FaaS/cloud environment in which functions are scaled as needed. In a FaaS/serverless environment a large amount of functions can be launched and produce a high number of requests to the in -memory state service. Once the functions complete, the data service should be scaled down.
The goals for in-memory distributed state service are: 1) it should provide high throughput for a wide range of objects sizes. 2) It should be cost efficient and ensure a low response time with a low cost of the usa ge of resources. 3) It should provide data consistency. 4) It should provide automatic elasticity. 5) Simple interface (get & put) to accommodate different applications.

There are a very few proposals on handling state in FaaS environments. During the last years some solutions have been presented such as:
Pocket is a distributed ephemeral storage system for storing ephemeral data for analytics in the context of serverless computing [123]. Analytics run in several phases and need to store intermediate data produced by each phase before the final output is produced. Pocket provides three types of storage: DRAM, NVMe Flash storage, and a generic block storage (HDD or SSD) to meet the I/O demands of the application.
Pocket architecture consists of three components: one centralized controller, one or more metadata servers, and multiple data plane storage servers (Figure 4). The controller allocates storage resources and dynamically scales the other two types of components: metadata servers and storage nodes. Metadata servers enforce data placement policies and server client requests accessing the storage nodes.
When a job is registered, it launches lambdas. Lambdas first connect to their assigned metadata server. Lambdas write data by first contacting the metadata server to obtain the storage server's IP address and connecting to the storage server to write data. Pocket supports elasticity, the controller monitors the system health and decides whether to deploy or remove new metadata servers or storage servers. To support metadata server failures, all metadata servers log their data operations to a shared NFS so that new metadata servers can access and replay the log.

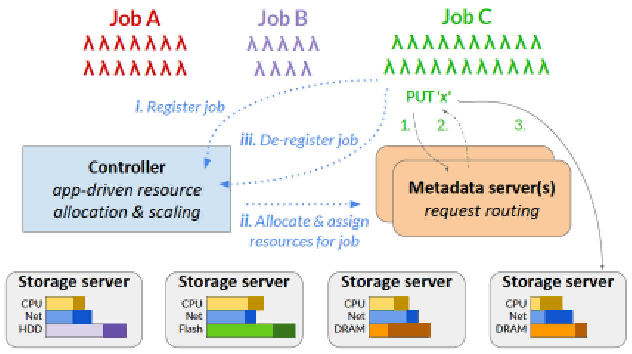


Figure 4 - Pocket system architecture and job deployment steps [123]

The controller fault tolerance mechanism is not presented. The paper states that it can be achieved by master-slave replication. Pocket uses Kubernetes container orchestration to launch and remove metadata and storage servers running in Docker containers. Metadata and storage servers are implemented on top of Apache Crail distributed data store [124]. The latency of Pocket is higher than the one of Redis [125] with a similar throughput.
In PHYSICS we aim at designing similar goals although, we do not only target ephemeral data. We will also provide different consistency levels.
CRUCIAL is a system for programming high-concurrency stateful applications with serverless architectures [126]. It implements a model that resembles shared distributed memory (Distributed Shared Objects, DSO). Data is organized as objects found in programming languages such as Java. Objects can be ephemeral or persistent and CRUCIAL replicates the data. State machine replication guarantees fault-tolerance of objects. The consistency criterium of the DSO is linearizability. That is, concurrent method invocations behave as if they were executed sequentially.
Figure 5 shows the components of CRUCIAL and how clients interact with the objects (either directly from the client or functions, CloudThread in CRUCIAL terminology).
Data objects are built on top of the Infinispan in-memory data grid [127]. CloudThreads are implemented as AWS Lambdas. CRUCIAL does not provide auto-scaling of objects and the replication model used by CRUCIAL is very expensive in terms of performance (based on state machine replication and implemented in Infinispan using group communication).

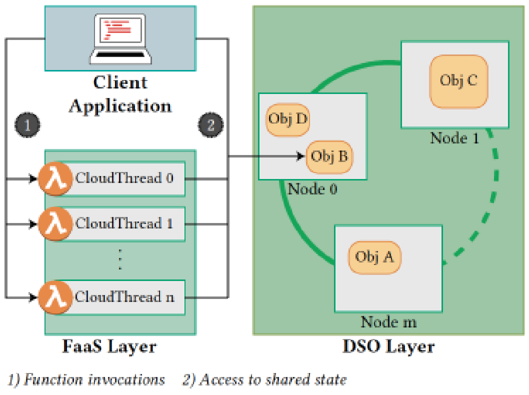


Figure 5 - Overall architecture of Crucial [126]

InfiniCache is an object cache for FaaS applications [128] optimized for large objects (MBs up to GBs) with strong locality, which are less frequently accessed than small objects. InfiniCache uses the functions memory to reduce the cost of accessing the objects (traditional caches are billed per hour in contrast to functions that are charged per invocation). InfiniCache is made up of three components: a client library, a proxy server and the Lambda cache pool (Figure 6). The client library is responsible for the cache invalidation. The proxy manages a set of lambdas and acts as a server accepting connections from Lambda nodes. A Lambda node cannot act as a server due to the limitations of functions. InfiniCache is based on AWS Lambda. The proxy keeps the mapping between objects and Lambda nodes. It also coordinates data migration.

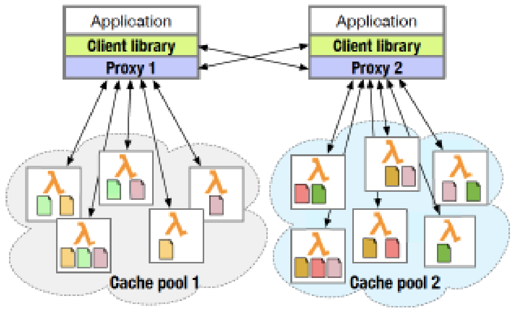


Figure 6 - InfiniCache architecture overview [128]

InfiniCache provides high availability of data by a combination of invoking functions periodically (to avoid that the function can be finished) and replication of functions. The performance of Iof this solution under changing loads is not yet evaluated and the performance with small objects (less than 10MBs^{-1}) is worse than that of AWS ElastiCache [129].
Cloudburst is an autoscaling stateful FaaS [130]. Cloudburst is not based on other FaaS, it completely designs a new FaaS system that collocates data with functions to provide high performance. Cloudburst is built on top of AnnaDB [131], a key-value data store that provides fault tolerance and automatic scaling. Each virtual machine may run several functions (Executors) and keeps an internal cache that asynchronously stores the state in AnnaDB. Cloudburst is the only system that takes care of consistency of the cached data providing repeatable reads and causal consistency. Its runtime scales independently of the AnnaDB. There are four components in Cloudburst: function executors, caches, function schedulers and a monitoring and resource management system (Figure 7). In PHYSICS we aim at providing consistency by integrating the in-memory service with a FaaS platform instead of designing the whole FaaS system.

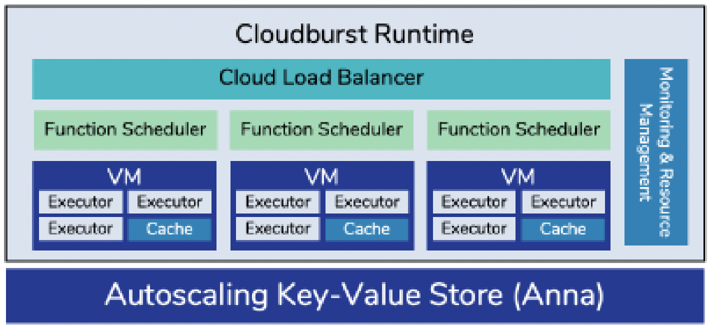


Figure 7 - Cloudburst architecture overview [130]

The different approaches providing stateful computation in FaaS environments either do not consider data consistency or cannot be used with existing FaaS platforms. Moreover, those approaches target or large objects or, if they allow small to medium size objects, the scaling is independent of the FaaS platform introducing two levels of independently scaled planes.

2.9 Adaptive Platform Deployment, Operation & Orchestration

One of the main objectives of PHYSICS is to provide a FaaS Platform able to operate, orchestrate, and deploy service applications adaptatively on different infrastructures from several cloud providers. Current implementations are facing several constraints limiting their potential:

- **Resource catalogue:** Not all cloud providers offer the same resources or services in their catalogue, therefore, it can happen that the PHYSICS user needs something specific that is not present in a given provider.
- **Deployment:** Although most of the cloud providers offer the same basic resources in their catalogue, they are not provisioned the same way, that means, that all providers expose a different interface on how resources are created, administrated and operated.
- **Operations:** Given the deployment diversity, there would be different procedures on how to execute the required operations to deploy and maintain the platform.
- **Orchestration:** On top of the previous points, the orchestration stands for making the platform work automatically. To be able to execute the operations with the minimum human interaction or no interaction at all. Ideally, the platform would automatically detect points of failure either in a proactive or reactive way and apply the required operation action.

Container based technologies provide the needed application isolation required to develop pl atforms in a heterogeneous environment. Such technologies did ramp up development and shorten delivery times of any solution since requirements could be isolated inside the container's definition. Moreover, container technologies provide a simplified way to deploy and operate different solutions while making them much easier to maintain.

The following platforms presented are taking this into account in their design by featuring a simple and homogeneous way to deploy, operate and orchestrate them by using te chnologies such as Docker Container and Kubernetes Middleware.

Apache OpenWhisk [132] is an open source, distributed serverless platform that executes functions in response to events at any scale. By using a Docker container technology, it can manage the infrastructure and scale according to the workload. The main elements of OpenWhisk and their interactions are depicted in Figure 8:

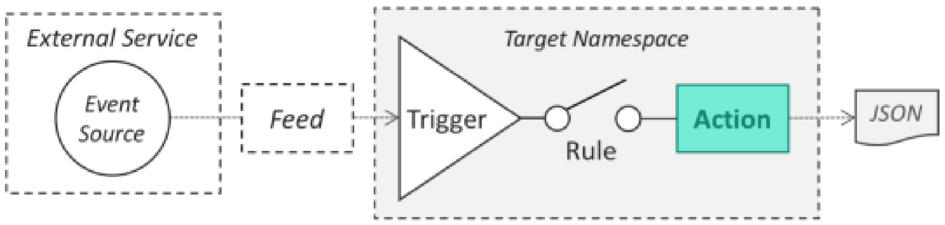


Figure 8 - OpenWhisk Programming Model [132]

- **Actions**: Isolated stateless functions that encapsulate application logic
- **Triggers & Rules:** The reactive element from an even source that start the execution of an action.
- **Sequences:** A definition of succession of functions actions that works together with a specific order and logic.

According the OpenWhisk documentation it supports many deployment options either locally and within cloud infrastructures, including container frameworks such as Kubernetes or OpenShift Mesos.
OpenFaaS [133] is an open source event-driven platform like OpenWhisk that deploys functions and microservices to container-based frameworks such as Kubernetes or OpenShift Mesos. The OpenFaas architecture is depicted in Figure 9.

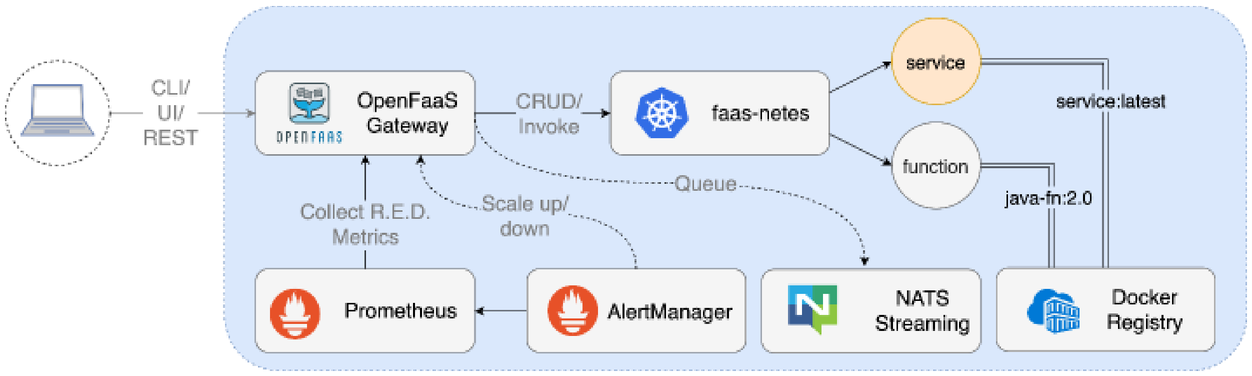


Figure 9 - OpenFaaS Architecture [133]

By using an OpenFaas Gateway endpoint, it is possible to trigger the call of services and functions hosted inside the platform together with other complementary solutions that extend the functionality for the log tracing, metrics collection and asynchronous calls.
Knative [134] is built on top of Kubernetes and abstracts away its complex details, making it extremely simple to deploy services on top in a serverless way. It focuses on an API with higher level abstractions and enables autoscaling up/down (including to zero) of applications based on their load. As it is basically an extension to kubernetes (set of controllers and custom resource definitions) it can easily get access to the base k8s improvements as well as be managed by third party cluster administrator tools (GitOps for instance). It has 2 core components: Knative Serving and Knative Eventing. While Knative Serving easily manages stateless services on Kubernetes by reducing the developer effort required for autoscaling, networking, and rollouts, Knative Eventing easily routes events between on-cluster and off-cluster components by exposing event routing as configuration rather than embedded in code. PHYSICS project could contribute to knative with more advanced scaling options (such as predictions or defining the desired minimum over time) as well as with extra APIs needed.
Kubeflow is an AI stack of technologies enabling easier incorporation of tasks based on Tensorflow, MPI, Map/Reduce and other relevant heavy weight application subsystems. It includes a set of functionalities (Jupyter notebooks for different runtimes, workflow definition through Kubernetes-based Argo pipelines [135], execution through a Knative platform layer based on Kubernetes [134]). It supports readymade

operators for the main environments such as MPI or Tensorflow, enabling easier execution and experiment implementation. A developer can write Python code primarily for AI tasks, package it and then deploy it (through Knative plugins) directly on a Kubernetes cluster environment. Defined functions can also be reused, even in some elementary workflows. Importing existing arbitrary code does not seem very flexible [136]. Although Kubeflow is not directly a FaaS platform, its execution engine based on Knative enables its consideration as a candidate tool. The architecture of Kubeflow can be found in Figure 10.

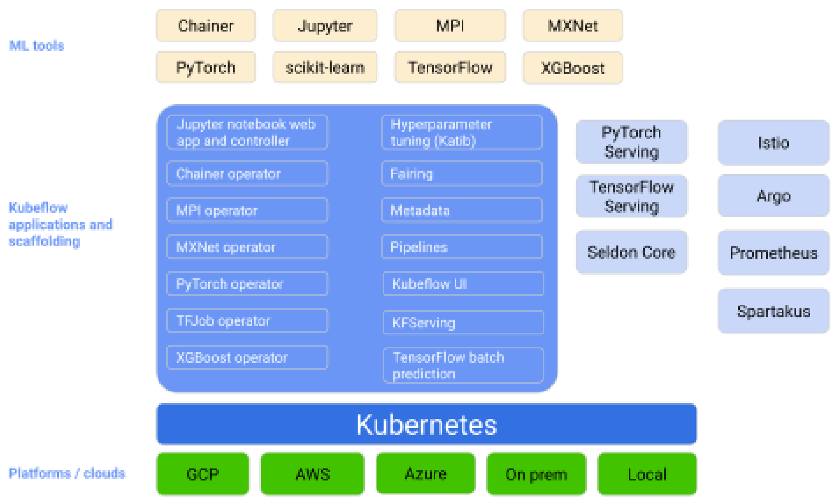


Figure 10 - Kubeflow Architecture.

2.9.1 Comparison between the aforementioned platforms.

Based on the anticipated needed functionality from the main FaaS platform, as well as the identified requirements of the following sections, the following key characteristics of interest for the platform are listed below and compared with the aforementioned platforms:

1). *Run any type of container as the result of an event (allow legacy code).*

Both OpenWhisk and OpenFaaS have a process for registering an arbitrary container image and trigger it with events. Kubeflow can include arbitrary scripts as executables, based on the Argo notation, although it does not have an event layer. It has however a REST API that can be triggered by respective external event mechanisms. Knative also runs any type of container, as it is fully integrated into Kubernetes.

2). *Support for sequences of functions (i.e. workflows)*.

OpenWhisk has the advantage in this case, given that it has a specific functionality in place (sequence operator). It also has a readymade Node-RED node that can be used to define function workflows in Node-RED. OpenFaaS has an external plugin (FaaS-flow) for declaring sequences of functions while Kubeflow has the pipeline definition language as well as an editor extension (Elyra) through which pipelines can be visually defined. Elyra is a further abstraction level (includes notebooks, flow creation, pipeline building and deployment to an available Kubeflow endpoint), however the concept of workflow is that of a static sequence of operations (with information passing from one step to another via intermediate cloud object storage files). Therefore it lacks the dynamicity of a combination like Node-RED and OpenWhisk/OpenFaaS. that is based on a message level execution of the workflow through the respective runtime mechanism. Furthermore, Knative eventing/triggering mechanisms can be used to support workflows a fact that kubeflow has on top of Knative.

3). *Different endpoints for triggering events (for better flexibility).*

OpenWhisk has an advantage in this case, covering a variety of endpoints and flexible structure between triggers, rules and actions while OpenFaaS has only HTTP based triggers. Knative also supports different eventing mechanisms, that can be used: "Source to Sink", "Channel and Subscription", and "Broker and Trigger".

4). *Backend support for Kubernetes (since it is the mainstream container orchestration system).*
All the tools support Kubernetes (possibly among others) as the baseline container management platform.
5). *Open-source nature and size/activity of community.*
All tools have a sufficiently large community, as well as backing by major companies that have helped in their evolution.
6). *Lifetime limitation of function (max execution time for a function).*
OpenFaaS, Knative and Kubeflow do not have any limitation on function execution, OpenWhisk has a maximum limit. In order to overcome it, alternative mechanisms would be needed, such as the Function Chain pattern.
7). *Ability to change parameters such as autoscaling rationale, concurrency, number of prewarm containers etc dynamically and during runtime.*
In this case OpenFaaS has a clear advantage, since it can be configured with different scaling parameters at the function level, including a REST API to change them during runtime. These parameters include aspects such as min and max container replicas used as well as scaling factor with which to increase them based on identified activation increase [137]. OpenWhisk on the other hand can limit the number of concurrent function activation across the cluster while Knative of Kubeflow is the only one that can be configured to execute multiple functions concurrently in the same container. This however is not expected to provide significant benefits given the computationally intensive workloads that Kubeflow primarily targets.

2.10 Service Semantics

The emergence of cloud and edge computing paradigms has enabled cost reduction and high resource availability for modern applications, by utilizing features that employ a resources-on-demand schema. Deployment and elasticity control of these applications is usually managed by tools that are compatible with the respective vendor infrastructure. However, this specificity of the tools along with the plethora of cloud vendors hinders the ability to migrate applications through cloud providers and application modelling in infrastructures that require multicloud solutions, thus defining a vendor lock-in problem.
To address these challenges, several Domain Specific Languages (DSL) and standards can be utilized to create ontologies based on semantic descriptions for cloud modelling, application management and monitoring. The main goal of the service semantics component in PHYSICS is to utilize these DSL and in turn describe the available resources in the time of application modelling. This will enable the creation of application topologies that in turn can be further analyzed for resource management and allocation.
The Web Ontology Language (OWL) [9] is one of the most popular standards for semantic descriptions that was initially developed to account for the specific needs of World Wide Web. It has been utilized within the Cloud Paradigm extensively in [138][139] to address cloud portability, resource management and discovery. OpenStack Heat, the main project in the OpenStack Orchestration program utilizes *Heat templates* to operate. These templates capture features such as the compute resources, network configuration, scaling rules etc in a YAML file. Additionally, a cloud application modelling and execution language (CAMEL) was developed during the PaaSage EU project. CAMEL is a multi-domain specific language for cloud application management. It utilizes CloudML [140] and defines new Scalability Rule Languages. Finally, OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA**)** is a DSL that aims to improve cloud portability and interoperability using semantic topologies. It can be utilized in both XML and YAML and defines two categories of each entity available, types and templates. Types are reusable entities that contain implicit knowledge while templates explicitly describe the application information.
The application of ontologies, service semantics and knowledge graphs are not limited to cloud applications but also extend to edge and IoT systems. The papers [141][142] study how ontologies can be used for modelling and reasoning in edge and IoT. In addition, IoT-Lite Ontology [143] aims to represent resources, entities and services that find applicability in the edge.
The service semantic component will explore the previously mentioned research results and standards in order to utilize their dynamics and capabilities. Ultimately, this knowledge of the state-of-the-art

approaches, along with a series of advancements and contributions will lead to a functionality that enables the modern resource management and allocation in the edge ecosystem and the FaaS driven applications.

2.11 Scheduling Algorithms

Serverless Computing has emerged as a new paradigm of abstraction, platform and implementation of cloud functions. It is thought to be the evolution of the Cloud Computing model in the sense of use of micro services and containers. Containers look to be one of the best options to work with isolated and controlled environments. With them it is possible to pack a whole environment and to deploy it anywhere. Within such environment, of course, it is added the application which will be executed when such container would be deployed. For instance, an application can be an entire program, or a part of that – such as a set of functions that can run separated but coordinated to join the main program in the end - or even small functions that do not communicate to any other. From the first scenario we could say that we are talking about a monolithic architecture application, from the second one, a micro-services architecture application and from the third one we are talking about function as a service modelled application. Please, notice that we are not discussing the efficiency of these approaches, the focus is to emphasize that containers can pack applications for several proposes and in different type of environment configurations. Once that is pointed, something is required to manage those containers. Kubernetes is one of the options for that. It provides a container management upon platforms. It is possible to deploy, schedule, execute them and interconnect them.

2.11.1 Serverless Platforms and Kubernetes

As mentioned above, FaaS applications can benefit from containerization, which means they can also be managed by Kubernetes-based solutions previously target to other or similar problems. Serverless Computing evolves the FaaS concept avoiding the server infrastructure management. So, if by one side FaaS applications benefit from Kubernetes based platforms, from the other side, they still handle the server's infrastructure. At this point, there were developed several Serverless Platforms on top of Kubernetes such as Kubeless [144], OpenWhisk [24] and OpenFaaS [133]. These platforms automatically handle the Kubernetes configuration side to make it easier for the developers to upload, deploy and execute their functions. For that, they create all the services and operators needed as pods within such clusters. When an action is needed to be done, they use these pods. For instance, when OpenWhisk is deployed upon a Kubernetes cluster, it creates pods to manage its administration, alarms, gateways, schedulers and so on. It also creates pods to connect to external services -with which it was developed - such as Kafka, NGINX and CouchDB. Similarly, OpenFaaS creates its own pods for helping its management. Going into more details for OpenWhisk, it abstracts a function as an Action, which is called by a Trigger and follows a Rule. Developers can deploy their functions and define such triggers and rules. With all of that, they can also combine actions - functions - with Events. OpenWhisk architecture is based on a) a NGINX mechanism to handle HTTPs requests; b) a CouchDB mechanism to save the platform information such as the Actions defined; c) a Controller to decide what to do with the HTTP requests received and with the information saved on the CouchDB. In addition, it sends Actions to be executed in the Invokers; d) a Kafka mechanism to manage the messages among the Controller and the Invokers; and finally, e) the Invokers that execute the Actions within containers.

2.11.2 Kubernetes Standard Scheduler

The decisions regarding to where to execute pods are taken by schedulers. Kube-scheduler is the default scheduler of Kubernetes and its role is to dispatch pods to nodes based on a scheduling policy [145]. Depending on the metric focused by the platform, sometimes the best allocation is not achieved with such scheduler. Besides Kubernetes allowing the developers to modify its default one, there are some available options such other open-source schedulers for Kubernetes such as Volcano [146], YuniKorn [147], [148], Safe Scheduler [149] and Multiple ClusterDispatcher [150]. Each one addresses to some of the different focus on scheduling objectives. Since all the tools mentioned above are based on Kubernetes and its default scheduler, we will depict its mechanism.

The Kubernetes scheduling process is based on some steps where filtering and scoring are the two main parts. Basically, before scheduling a pod to a node, the kube-scheduler filters all available nodes and if there is any, it scores them to select the most valuable one. If kube-scheduler algorithm is not the most suitable option for some specific case, Kubernetes allows developers to change it. Policies [151] are the way that Kubernetes implements the filtering and scoring phases for the scheduler. Anyone can access and change its parameters. Furthermore, Kubernetes split its scheduling process in stages, for instance, QueueSort, Filter, Score, Bind, Reserve and others. Each Stage represents a scheduling step, and they are exposed by Extension Points. Extension Points behaviours are implemented by Plugins [152]. A Plugin can be used by one or more Extension Points. In the end, the set of Extension Points and its Plugins compose a Profile [153]. A Profile is a mechanism that allows developers to configure Plugins to implement different scheduling behaviours for different Stages. If not provided at pod creation, Kubernetes considers its default Profile, the default-scheduler. There is also the possibility to develop several Profiles and to use them within different pods as a multiple-scheduler mechanism [154]. For instance, an example is the QueueSort Extension Point. As an Extension Point it defines a scheduling Stage. It provides ordering functions to sort the pods in the schedule queue. To do so the QueueSort Extension Point uses the PrioritySort Plugin, that implements the default priority-based sorting. With many others, QueueSort Extension Point, and consequently the PrioritySortPlugin, compose the default-scheduler Profile. Since Kubernetes environment evolves and changes over time, some resources might become under or over usage. For instance, a very simple example is if a pod fails, and it is duplicated for fault tolerant reasons. If the failed pod becomes available again the cluster would be running two instances of the same pod. For this and other reasons, there is a mechanism to avoid such under or over resources usage named descheduler [155]. The goal of the descheduler mechanism is to find pods that can be moved and evict them. This does not mean that the descheduler will replace the evicted pods, but if needed, that can be done by the scheduler itself.

2.11.3 Scheduling Algorithms for FaaS

Scheduling algorithms can be used for several reasons, such as to minimize functions response time, to save costs, to reduce data movements, or energy consumption and so on. To address the problem of functions locality requirement, Aumala et al. proposed a scheduling policy based on the packages needed to execute a function [156]. The package-aware scheduling proposed (PASch) considers the package affinity during scheduling. So, a worker node which already executed a specific function can re-use execution environments with preloaded packages. They showed that the cache hit rate was improved, as well as the individual functions execution and turnaround time. Besides, they do not achieve the best load balance results, due to the priority given to the packages locality instead of the resources availability.

To address the response time minimization challenge, Stein presented an approach based on a non-cooperative game-theoretic load balancing, implemented on top of OpenWhisk [157]. The concept of non-cooperative load balancing is that every user within the distributed scenario has information regarding service time and allocation from the hosts. With that information, they calculate an optimal split of its own perceived arrival rate in response to other users. The distributed controller architecture used by OpenWhisk shares common host state information such as the number of concurrently active invocations on each host. Combining the concept of non-cooperative load balancing and the architecture of OpenWhisk, Stein presents a non-cooperative on-line allocation heuristic (NOAH), where each function contains an expected average waiting time to be handled by an event. For each function it estimates the required number of in-stances to respect such average waiting time.

The dispatcher first searches for idle instances to schedule the functions and to get the minimum completion time possible, otherwise it balances the request based on the function’s allocation. When a function is allocated to a site, its instances are not directly spawned. An instance pool manages the requests and take the decisions locally. Suresh et. Gandhi also used OpenWhisk to implement a scheduling policy, named FnSched, focused on costs reduction. Their main idea is that they try to use as less resources - named invokers - as possible [158]. For that, they developed and combined two algorithms, the CPU-shares regulation and the greedy one. The so-called CPU-shares regulation algorithm, it regulates how much CPU the instances will use. They define a latency ratio that measures the quality of the service and verify it over time. If an instance achieves the latency ratio, it receives more CPU-shares - resources. The greedy algorithm

will take care of allocating and scaling up the instances. It checks the available memory of the host, and just if needed, more invokers are used. This way they avoid using many machines, invokers, and consequently reduce costs. They point that it also reduces the cold start latency since they prefer to use the same invoker as long as necessary if possible, and therefore keep the containers. In addition, to avoid cold start latency, they duplicate the container during an invocation to another invoker even if it will not be used. This way, whenever scale up is needed, some invokers will have a warm pool of containers.
The scheduling policies were implemented on top of OpenWhisk which provides a REST interface to a ccept requests and provide response to them. The CPU-shares regulation algorithm belongs to each Invoker and the greedy algorithm belongs to the Controller mechanism. With a different focus, James et. Schien present a model and the results of the implementation of a scheduling algorithm based on low carbon emissions [159].

2.11.4 Simulations for Serverless Platforms

Although a lot is said about improvement of metrics by evolving the scheduling phase of the functions, one of the main challenges of this subject is reducing the cost to perform and evaluate all the possibilities on in-production platforms or services. Such cost can be understood in several ways such as money, time, resources and many others. To address these problems, simulation is a key practice and solution. Within simulations one can easily perform and compare different scheduling policies without using any of the above-mentioned resources. Besides, it is needed to have accurate results to be fair with real-world environments and there are not many tools with such accuracy. Either in a real deployment or in a simulated one, benchmarks are needed [160]–[162]. Kim et. Lee [160] presents a set with micro and application benchmarks for serverless platforms. The micro-benchmarks help measure the performance of target resources with functions call, such as matrix multiplication, linpack, iperf3, etc.
The application-benchmarks provide an entire application with realistic scenarios, dealing with data-oriented flow and several resources together. Some examples are image/video processing, logistic regression, face detection, word generation, etc. In this scenario, SimGrid [163] is a framework that allows the development of simulators to be used to prototype, evaluate and compare system designs, platform configurations and algorithmic approaches. Batsim [164] is a resource and job management system (RJMS) simulator developed on top of SimGrid. It allows the development and study of scheduling algorithms. The design of Batsim helps researchers to apply and compare different scheduling polices to different platforms due to its decoupled development. Batkube [165] is a Kubernetes cluster simulator developed on top of Batsim. It benefits from Batsim platform and scheduling decoupled implementation, simulating different platforms and workloads connecting them to different scheduling policies.
The first work developed with Batkube uses the default scheduler of Kubernetes. Mahmoudi et. Khazaei present the SimFaas platform [88]. It is an open source serverless platform which allows the study of scheduling policies. The platform deals with some aspects of serverless computing such as function instances states, cold/ warm start-up, auto-scaling etc. Different to Batsim that allows the implementation of whole platforms, SimFaas abstraction uses four parameters to characterize a platform: expiration threshold, which they point that is usually constant for public serverless platforms such as Isagoge Cloud etc.; arrival process, that describes the frequency that the jobs arrive; warm and cold service process, representing the response time took by the platform to reply cold and war requests respectively. To characterize a workload, they consider the instances arrival rate and the response time for warm and cold requests. Within the simulations, a practice to validate a scheduling policy as generic instead of being useful only on specific case, is to compare such results with benchmarks.
Those are studies that examine the performance of general workloads and platforms. One example is presented by Utiugov et al. that demonstrates a technique to reduce functions cold start latency based on snapshots [166]. They save the current state of a VM in the disk and use it when needed. With that they capture the state of the virtual machine monitor (VMM) and the guest-physical memory contents. It reduces the cold-start latency, in addition to require no main memory during the idle periods. They provide a tool named vHive, an open-source framework for serverless experimentation. It is based on Containerd [167] for the container's orchestration and on Firecracker [168] for provisioning. Using vHive, they evaluated the functions from FunctionBench [160].

2.12 Resource Management Controllers

New controllers and interfaces will be added to the infrastructure layer (OpenShift, Kubernetes, Serverless framework, etc.).to provide the needed knobs for the scheduling mechanisms implemented (Chapter 2.11) as well as for the multi-cluster orchestration layer (Chapter 2.9) or application-level controllers (Chapter 2.4) to enable improved fine-grain scheduling decisions as well as optimized resource allocation between clusters and within each cluster over time. Moreover, the target is to contribute those API/Controllers extensions to the related upstream versions of each software communities.

As regards to *in-cluster interfaces and controllers*, there exist several options available that PHYSICS project can leverage in and optimize them.

1) *Scheduling Policies and Multi-Schedulers*

Kubernetes comes with a default filter and weight scheduler. However, if the default scheduler is not enough, it provides you with the knobs to implement your own [169]. In fact, you can even run multiple schedulers simultaneously (alongside the default one) and instruct Kubernetes which sch eduler to use for each of the pods. Basically, it allows to run your scheduler as another pod running on the system. New pods will use the new scheduler by adding the desired scheduler at the pod spec (*schedulerName* field). PHYSICS will extend this to provide new scheduling mechanisms (Chapter 2.11) as well as configurable options for them and for the upper management layer to easily select between them.

2) *Descheduler Operator*

After the initial scheduling process, there may be changes on the infrastructure that may end up in a nonoptimal status. For example, new nodes may have been added or deleted, or the resource usage in one of the nodes is higher/lower than when an application was initially scheduled which forced a less optimal pod distribution. In order to react to that, work has been done to provide a "*descheduler*" for Kubernetes [170]. This is work on its initial steps that PHYSICS project could leverage and extend to provide the needed APIs and descheduler profiles for other components (such as Chapter 2.13) to trigger a more optimized resource management over time. The descheduler operator is in charge of enforcing the profile configured every X seconds by performing new scheduling decisions through terminating pods in some nodes and creating them in the new ones. Currently there are three profiles implemented:

AffinityAndTaints: Basic profile that removes running pods that violate node and pod affinity, and node taints.

TopologyAndDuplicates: Attempts to balance pod distribution based on topology constraints definitions and evicting duplicate copies of the same pod running on the same node.

LifecycleAndUtilization: Focuses on pod lifecycle and node resource consumption. It attempts to evict pods from nodes with high utilization that can fit onto other nodes with less load. High and Low utilization is measured based on CPU, memory or pod capacity percentages.

3) *Performance Operator*

In order to provide more predictable performance, there is a need for some extra low-level configuration (kernel tuning, CPU pinning, hugepages, NUMA awareness, etc.). These low-level configurations should be abstracted away behind simpler APIs for the upper layers (such as the co-allocation techniques developed in 2.13). In order to do that, PHYSICS could leverage the work being done around the Performance Operator [171] to ensure that nodes are configured with the needed kernel flags (e.g., to configure it with real time kernel as well as hugepages), as well as the isolation between different applications on the same no de can be enforced by assigning different cores to different applications (NUMA aware). The performance operator provides an easy way to configure and manage some of those capabilities on OpenShift:

- ➢ Reserve a set of cores per node to either run or not run other workloads
- ➢ Ensure QoS for a given pod by creating it with the same memory limit and request, as well as the same CPU limit and request.
- ➢ Annotations on a pod to configure (enable/disable) CPU load balancing

Regarding *multi cluster management*, there is a need for applications (pods) running in different clusters to be able to reach other components/applications leaving in a different cluster, enabling a *common networking layer across Kubernetes clusters*. Kubernetes has defined the *Multi-Cluster Services API* [172], which specifies both the terminology as well as the expected process to expose a service across Kubernetes

Clusters. Submariner [111] is one upstream project which targets to cover this existing gap, implementing the mentioned multi-cluster services API, and providing more flexibility and extra options at the application placement layer. Submariner is a tool built to connect overlay networks of different Kubernetes clusters (with different CNIs/Networking plugins) by establishing encrypted tunnel s between each Kubernetes cluster and ensuring proper service discovery and traffic redirection between clusters. Currently Submariner is in the pre-alpha stage, but the PHYSICS project will leverage the already existing functionality and work together with the upstream community, providing new requirements and contributing enhancements.

Despite the option to have applications with a shared networking layer across clusters, there is still a need for *orchestration decisions* about what applications to deploy on what clusters, how many replicas on each, etc., as well as to have a simple way of managing and visualizing it. There are several tools to enable cross cluster management, for instance ManageIQ [173] or Red Hat Advance Cluster Management (ACM) [174], based on Open Cluster Management [175]. ACM for Kubernetes provides a single view to manage Kubernetes clusters, providing end-to-end management visibility and control to manage clusters and application life cycle, including security and compliance for the entire Kubernetes domain across multiple datacenters and public clouds. It supports easy provisioning of OpenShift clusters on several cloud providers (AWS, GCP, Azure), and on-premise (OpenStack, baremetal, vSphere). In addition, it allows to enforce policies at the target clusters using Kubernetes supported custom resource definitions. P HYSICS can leverage and enhance the APIs provided to better support the multi-cluster orchestration layer (Chapter 2.9) needs.

In addition to the above, and as organizations embark on the hybrid cloud, new challenges arise when managing applications at different clusters, such as the complexity of deploying and delivering the applications in a consistent and predictable way. Currently there is a trend about managing the applications (continuos delivery) in a declarative way, following the *GitOps model* [176]. GitOps is a way of continuously (re)deploying cloud native applications based on having a Git repository as the source of true and containing the declarative descriptions about the infrastructure needs for the applications. It allows full transparency through Git audit capabilities and provides a straightforward mechanism to roll back to any desired version across multiple OpenShift and Kubernetes clusters. There are several tools running on top of Kubernetes providing the GitOps functionality, such as Ar goCD [177] and kustomize [178].

2.13 Co-allocation Strategies

The co-allocation strategies component will investigate how to group services in a physical node for improving overall performance. The co-allocation component will use the monitoring information of the services such as CPU usage, memory, network usage… in order to identify complementarity of services. This information will be used by the adaptable scheduling algorithms. The different effects of co-allocation between different users and workloads will be studied to obtain the best co-allocation strategy.

Applications are divided into different functions that are executed in containers. A container needs a set of resources (CPU, memory). In some cases, containers have dependencies between them and re quire to be collocated on the same node to reduce network latency. A pod groups several containers, and pods have limited resources which limits the number of containers in a pod. A physical node can host one or more pods depending on the amount of resour ces the pods require.

Figure 11 shows an application that is made up of six functions that will be deployed in a 4-node cluster. The function execution can be represented as a directed acyclic graph where nodes are functions and arrows represent the execution order. When a function completes, it sends its results to their neighbor functions. The cluster is made up of 4 nodes with different resources. In one of the node s a NFS is deployed within a pod. This pod is a *Read Write Many (RWX) persistent volume [145],* needed to read and write shared data with other pods. The co-allocation service will provide rules for deciding on the co-allocation of pods in a physical node. It will propose a distribution of functions (pods) among the available nodes and a co-allocation strategy that takes advantage of the available resources and does not create bottlenecks.

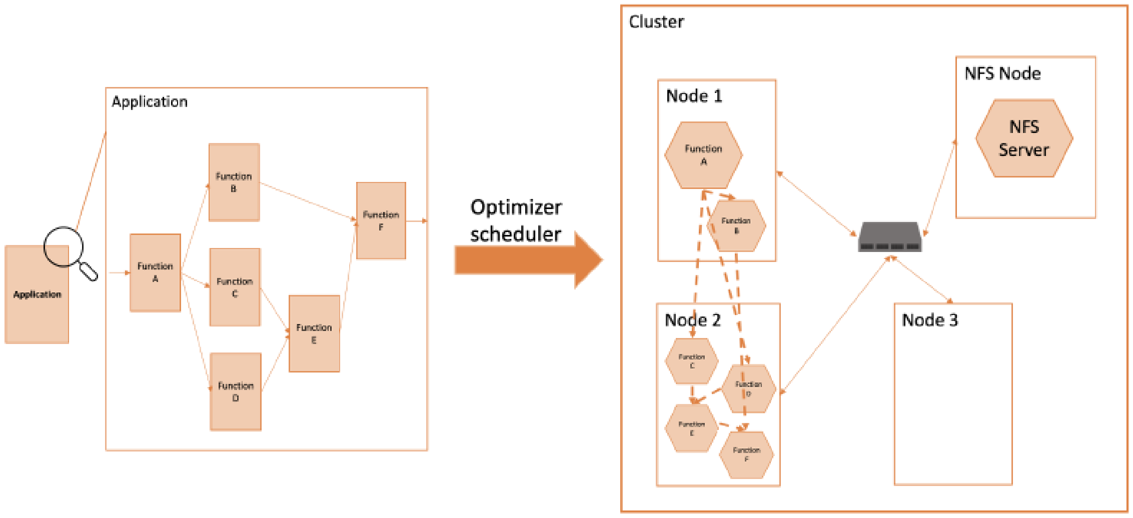


Figure 11 - Optimizer scheduler example

The co-allocation strategy will be triggered at least when: 1) a new application is deployed, 2) an application scales out/down. There are several tools to allocate application pods in different hosts, such as Kube-scheduler [145], describe in Section 2.11 and Openshift scheduler [179], [180], that is similar to the previous one.
Another way to allocate pods to nodes is using affinity and *anti-affinity rules* (Openshift [181], Kubernetes [182]). The *affinity rules* define the preferred nodes on which a pod will be executed. On the other hand, the *anti-affinity rules* prevent a pod to be scheduled on a node. Pods are tagged with a selector label. The selector label is not unique, in fact, there can be multiple pods with the same selection label allowing the identification of a set of pods. If an affinity rule is defined, the scheduler will place a pod in the same node as other pods with the same selector label. Otherwise, it remains as pending to be placed. The first pod within a selector label doesn't have to have any rules and it will be placed considering the scheduler configuration. In case of the *affinity rules* there are two types: required, that which means the rule must be enforced; and preferred, which means that the rule shouldn't be enforced. Pod selector label and affinity/anti-affinity rules are configured in the podspec yaml file. Multiple rules can be configured and can be affinity or anti-affinity, but cannot overlap; otherwise the pod will never be scheduled.
In addition, Openshift allows the creation of *infrastructure topological levels* by assigning custom labels to nodes [183], grouping a set of nodes in levels such as cluster, room, building, etc. Affinity and anti-affinity rules can be used to place different pods in the same level (set) of nodes. The *affinity rules,* in this case, allow the administrator to define the set of nodes where the pod must be placed using a selector level. This way, all service pods are scheduled on nodes within the same level. If there are no more nodes available in the selected level, the rest of pods to be deployed are not scheduled. On the other hand, the *anti-affinity rules,* also called *spread policies*, allow pods of the same service to be placed over different levels. This type of policy is well suited for high availability purposes since it distributes pods evenly across available nodes.
Both Kubernetes and Openshift have tools to define different co-allocation strategies for new pods taking into account the requirements in terms of resources of each pod. More over, the co-allocation service has to take into account dependencies between to pods and the resource consumption for avoiding bottlenecks and to create the proper co-allocation strategy.

2.14 FaaS Security

The security implications of FaaS in terms of benefits and concerns is still an open topic that requires additional research [184]. The serverless architecture in a general perspective, has mainly positive outcomes when it comes to security due to the fact that the underlying system is patched, updated and managed by the cloud provider. This fact effectively removes all system security responsibilities from the application developers and leaves only the application-level security to them. Besides this significant benefit, the serverless architecture has some security concerns that mainly stem from its complex, stateless and segregated nature that will be presented through this section that is divided in several subsections that focus on the different identified concerns.

2.14.1 Application security

In the architecture of FaaS, the organisation that needs to run the code does not need to know or care about the underlying operational issues of how many servers/containers are required, networking and storage issues associated with the operation of the code, or with maintenance, updating, patching, and securing the servers supporting the functionality—that is all the responsibility of the cloud provider. [185]

While the usage of FaaS removes any security issues and responsibilities from the organisations, this is only true in part; while the organisation has delegated the applied security associated with the servers and the host OS to the cloud provider (and most of the times the security issues of central common applications such as databases), the organisation is still fully responsible for the security of their deployed function-application.

This function-application security includes but is not limited to [185]: (i) the actual code that runs in each function, (ii) the connection and communication between function components in the context of the application, (iii) the authentication and authorization of functional components, users and other internal or external entities, (iv) the protection and validation of data and communications and (v) the analysis and protection from vulnerabilities in code, modules, and libraries that the application calls and uses including third-party code or services.

Given these facts, the OWASP organisation has provided a top-10 list with the most common security vulnerabilities that provides an analysis of common attack vectors in the context of FaaS serverless applications [186]. These attacks mainly focus on the application security given the fact that system security is off-loaded to the cloud provider. The list is the following:

Injection (major security concern): Refers to attacks where the input to one function can be controlled or manipulated by an attacker. In the case of FaaS, the input does not only come from API calls but also from internal cloud triggers (cloud storage events, stream data processing, database changes, code modifications, notifications etc.); this fact should be considered when calculating the attack surface of a FaaS application.

*Broken Authentication (moderate security concern)**:*** This type of vulnerability is concerned with ways that the authentication can be maliciously manipulated. This includes forgotten stored credentials, authentication triggers that can be toggled in an unforeseen manner, entirely open APIs, public cloud storages etc. These vulnerabilities mainly stem from weak and not carefully analysed identity and access control application design. This issue is especially relevant in serverless design where there are multiple potential entry points, services, events and triggers with no continuous flow, something that requires multiple points of automated authentication.

Sensitive Data Exposure (Moderate Security Concern): Most of the methods used in traditional architectures, such as stealing keys, performing man-in-the-middle (MitM) attacks, and stealing readable data at rest or in transit, still apply to serverless applications. This type of attacks can come from faulty or non-existent usage of encryption or from the expectation that temporary files (/tmp) will be delete d automatically.

*XML External Entities (Low Security Concern)**:*** This is a class of vulnerabilities manifests from the parsing of defined XML External Entities from malicious payloads. This issue is internal to the XML specification, where it is allowed for XML files to define external resources that could potentially be manipulated by attackers.

*Broken Access Control (Moderate Security Concern)**:*** This refers to the bad implementation or usage of access control mechanisms. In FaaS architectures, the management of admin credentials to the system is not crucial since they are automatically managed by the provider. What is crucial in such architectures is the management of access to resources that can be done through the service provider.

Security Misconfiguration (Major Security Concern): This category is focused on any security misconfiguration that includes configuration of access to resources, over-privileged functions and most importantly the human error.

Cross-Site Scripting (Moderate Security Concern): It is an extended form of traditional cross-side scripting (XSS) since the source of attacks could also come from other triggers (emails, IoT, cloud storage etc
Insecure Deserialization (Major Security Concern): This is the traditional form of deserialization vulnerabilities in dynamic languages (Python, JavaScript, Java, .NET).
Using Components with Known Vulnerabilities (Major Security Concern): Usage of dependencies and third-party libraries is common for FaaS applications. Usage of external components that is known to have vulnerabilities or using them from questionable sources will lead to introducing vulnerabilities to the entire application.
Insufficient logging and monitoring (Major Security Concern): This problem mainly focuses on the undetectability of attacks due to the lack of logging and monitoring. This is even more relevant in FaaS architecture where monitoring and logging is even more difficult and complicated.
Despite these ten vulnerabilities, the list also identifies other types of vulnerabilities that are common amongst serverless architecture. These include (i) denial of service (minor), where resources are peaked to the set limit by the administrator, (ii) denial of wallet (moderate), where the resources are escalated to such a degree that the organization is not able to pay for the requested fee, (iii) insecure shared space (moderate), where remounted shared drives are not properly cleaned from previous runs of the container and (iv) Business logic / flow manipulation (major), that targets at exploiting the higher level logic implemented by the serverless components of the application.

2.14.2 SGX, Trusted Computing and Blockchain for FaaS Security and Privacy

The majority of solutions trying to enhance security and privacy properties of serverless architectures, use the Intel SGX as a trust anchor, a hardware-backed trusted execution environment, which can be utilized for running functions in private memory regions called enclaves [187]. The usage of SGX enclaves is proposed in [188] where the authors propose a secure lambda function model where each function that handles sensitive data or is security critical is run under protected SGX enclaves. With this, the assurance provided both in terms of security and of privacy are enhanced. SGX is also used by [189] in the proposed protocol that the system uses to ensure confidentiality and integrity of data, and integrity of function chains. The additional benefit of this solution is that to overcome performance and latency issues that exist sometimes in SGX applications, the authors applied several SGX-specific optimizations to the runtime system, such as SGXv2 to speed up the enclave start-up and perform batch enclave page cache augmentation. In [190] the authors propose the usage of SGX in an advanced key distribution and verification protocol that allow for seamless verification of functions and assurance for their proper and untampered functionality that focuses on the privacy and security of the data handled by the function.
The usage of blockchains has been proposed many times as a solution for privacy in the context of the cloud, especially when it comes to the protection of healthcare data [191][192]. This has been observed mainly due to the tendency of healthcare organisations to offload their responsibilities towards security and privacy to the cloud providers. The proposed usage of blockchains is to essentially pass the ownership of the data to the true owners (the patients) and this way adding a layer of security and privacy on the application level.

2.14.3 Serverless Components Communications security

When it comes to the communication between application components in a FaaS architecture, developers can choose to implement all the code that manages the calling of their functions and their i integration with any third-party services used or use cloud-service provided communication solutions. By choosing to implement their own communication solutions and carefully restricting their calls against the FaaS providers API to the commonly used subset of supported APIs across the FaaS providers that they plan to support, the developer can preserve maximum flexibility in dealing with FaaS providers and so can minimise the effort associated with moving code between FaaS providers, reducing provider lock-in.
This flexibility makes the organisations fully responsible for maintaining the security of the communications to and from the functions and the state datastore (which should be protected with TLS)— with appropriate key management. On the other hand, using a cloud service provided communication

solution, delegates this security responsibility to the cloud provider but it removes the aforementioned flexibility and locks the application to the specific provider.

2.14.4 TRUSTEE Project Cluster

The TRUSTEE project cluster is a consortium of projects that all have the common denominator of security and privacy in the cloud. TRUSTEE (daTa pRivacy and cloUd Security clustEr Europe) is an aggregation of results of 11 research projects funded by the European Union that was established within the Common Dissemination Booster initiative. The TRUSTEE initiative is coordinated by the CREDENTIAL [193] project, and is composed of the following projects: MUSA [194], PRISMACLOUD [195], SecureCloud [196], SERECA [197], SPECS [198], SUNFISH [199], SWITCH [200], TREDISEC [201], UNICORN [202], and WITDOM [203], which are all focused in different domains of cloud security and privacy, ranging from secure and privacy-friendly authentication over encrypted and distributed solutions for data sharing and cloud storage to data integrity, authenticity, and availability. In the context of the TRUSTEE consortium, there is a variety of readily available solutions in multiple contexts that could be taken into account for the PHYSICS project when it comes to security. The following is a concise list of the categories that these solutions cover: (i) identity management, (ii) cloud security management, (iii) cloud management, (iv) secure data sharing, (v) application development and (vi) SLA management.

2.14.5 Identity and access management

Most cloud providers implement internally critical security functionalities for developers. One of the central services among these are the cloud identification and authorization management (IAM) functionality, policy and role management, and access control management. In ANNEX I there is a table (Table 2) that provides a bird’s eye view of these services from the three major cloud providers, namely: Amazon AWS, Google Cloud Platform, and Microsoft Azure. These services also have firewall capabilities, which customers can configure to restrict improper access.

A novel identity and access management system as part of the overall FaaS architecture is proposed in the work of [204] as part of the H2020 SUNFISH project that created the proposed cloud federation service. This service allows organisations to enforce attribute-based access control policies on their data in a privacy-preserving fashion using their federated solutions. The end users are granted access to federated data when their identity attributes match the policies, but without revealing their attributes in clear. The entire solution is based on two trust anchors, namely, blockchain and Intel SGX hardware platform to guarantee integrity of the policy evaluation process, a common approach when it comes to security and privacy in the context of FaaS.

2.14.6 Secret Storage and Management

The proper deployment of serverless components alongside access control and secure storage rely upon secrets that must be installed within them. Some form of trustworthy secret management and storage, typically backed by cloud hardware security modules, is needed. Most of the major cloud providers offer some kind of secret management: Google Cloud platform does this with Cloud KMS, Amazon provides this with AWS Secrets Manager, and Azure provides this functionality in Azure Key Vault. Furthermore, in all of these services, there is the choice of encrypting lower-level secrets under a protected key that is stored in an appropriate database, being decrypted under the protected key, as needed. Automated provisioning of secrets to VM’s/containers is challenging as there are many underlying security concerns. One option for example is to use the keys from the command line directly, but command line arguments are logged and easily accessible from the history. This is the reason why storing and injecting secrets in drives that are mapped to the target or the use of products such as HashiCorp’s Vault to inject secrets is preferable.

Despite the services provided by the cloud services, there is an array of solutions that are service-agnostic and allow to avoid a vendor lock-in. Of course, this is the classic trade-off between ease-of-use/integration and avoiding a lock-in. An advantage of using the cloud provider functionality is that the provider has integrated their security controls with the secret management functionality, something that simplifies the developer’s task in managing and deploying secrets to components. Following, there is a list of possible solutions for secret management.

There are in-house solutions from each major cloud provider such as AWS Secrets Manager [205], Google Cloud Secret Manager [206] and Azure Key Vault [207]. All of these tools are quite versatile within the application domain of each provider with tight integrations with the services of the provider, something that allows for easier to deploy secret management services. On the other hand, usage of such solutions will lead to vendor lock-in. The solution Akeyless Vault [208] provides a zero-trust scheme for multi-cloud secret management and propagation of secrets. This solution is very focused on privacy and provable zero knowledge of the secrets from the provider, but it is not open-source and comes at a price for commercial usage.
The open-source solution Vault from HashiCorp [209] is a solution that allows for a cloud-centric secret management across well-known providers (AWS, Google Cloud, Azure) that allows for storing, moving, replicating secrets across cloud providers and networks while also providing tight access control to said secrets. Square Keyewhiz [210] is an open-source solution for secret management (such as GPG keyrings, database credentials, TLS certificates and keys, symmetric keys, API tokens, and SSH keys for external services). This solution is provider-agnostic since it provides just a JSON API that the application-level gateways can use to interact with. Confidant [211] is an open-source secret management tool that aims at ease of use and security of the secrets. It is tightly integrated with AWS since it uses DynamoDB for the storage of secrets. Docker Secrets [212] is a service deeply integrated with Docker Swarm and allows for management and deployment of secrets to docker containers in swarms. This service provides an application-level abstraction of secrets management since it utilises Docker, and thus it allows for ease of use and tight integration with the developed application without enforcing vendor lock-in. Knox from Pinterest [213] is an open source secret management solution that is provider-agnostic. The management of secrets is offloaded to the application level, where developers have to create the appropriate hooks for their business logic that they need.

2.14.7 DoS attacks — Economic DoS

Depending on how the services are configured, traditional DoS attacks can still be applicable even in a FaaS architecture. More specifically, cloud providers allow for policy specifications that enforce resource limits on functions (such as execution times, CPU cycles and memory usage). Given this capability, administrators could enforce strict policies to avoid overcharging their accounts; a fact that essentially opens the door for traditional DoS attacks. That is, attackers could just perform a DoS attack, and reach the enforced policy limit, effectively crippling the function.
If there are no restrictions on resources from policies, then there is a manifestation of a new type of attack, the economic denial of sustainability (eDoS) attack. In this scenario, the attacker has such an overwhelming botnet size in his disposal that an attack would escalate the FaaS-based application resources to such a degree that it essentially becomes economically unsustainable for the organisation. There are many proposed research solutions such as [214] and [215] that focus on monitoring the behaviour of the overall application. An additional solution would be to create cloud-specific policies that detect possibly malicious behaviours and block any further access attempts from the source(s) of the detected attack.

2.14.8 Additional Considerations

In addition, and in conjunction to the aforementioned security challenges, there are a few more considerations when it comes to security in the context of serverless architectures. The dynamicity and complexity of the FaaS components could lead to left-out APIs that are not properly secured, care must be taken so that each communication is properly protected and controlled while also ensuring that any input to each component is properly validated. Another concern is the usage of DevOps and agile methodologies that continuously integrate changes and modify the architecture of an application; this concept in FaaS could leave residual and unused (legacy) components that are prone to or could lead to unforeseen vulnerabilities. Furthermore, given the stateless nature of serverless architectures; there should be a secure methodology for the management and storage of the required state in any used databases. This should also be taken into consideration in the shared memory proposed by PHYSICS.
Continuing in the context of the PHYSICS project, the usage of the reusable artefacts marketplace platform (RAMP) could lead to reusable vulnerabilities that are propagated through all the users of the artefact. This

stands true for all marketplaces and automated paired with manual security analysis should be performed on each artefact to ensure that no serious vulnerabilities exist. Finally, the global deployment strategy and auto scaling architecture of the PHYSICS project should also be concerned with malicious manipulations of the autoscaling and locality mechanisms. More specifically, the algorithms used to determine the factors of these mechanisms, could be fed manipulated data in order to change their behaviour.

2.14.9 Anonymization & Encryption Services

While cloud computing provides a flexible, on-demand and dynamic environment to the user, it poses many privacy concerns as the provider can access the data stored in the cloud at any time. This is the case for the edge computing model as well, as the user's data is shared across many distributed nodes connected through the Internet. Sharing these data introduces privacy concerns and major reliability issues if no proper security measures are implemented by the service provider. The survey in [216] highlights the various privacy threats present when handling sensitive data:

- **Intrusion** – Any action that can directly or indirectly invade an individual's or an organisation's private affairs.
- **Public Disclosure** – The release of private or previously unknown information to the public regarding an individual or an organisation.
- **False Light** – A form of public disclosure of false or malicious statements. Usually accomplished by distorting the truth or using fictional facts.
- **Appropriation** – Referring to the appropriation of an individual's or organisation's identity or other private data without authorization or knowledge. Especially in the digital era, this happens with online accounts or profiles.

This is especially the case when the data sharing involves health data or personally identifiable information, where efficient and reliable security mechanisms need to be implemented to preserve the user's privacy [217]. Several third-party open-source anonymization software have been evaluated [218] that can be utilised to address these privacy issues, these are presented below:

- **ARX Data Anonymization Tool**: A popular open source and cross-platform tool, supporting different privacy models like k-anonymity or Differential Privacy and can be used for up to 50 dimensions (e.g., attributes) on millions of data records.
- **Amnesia**: A data anonymization tool that supports k-anonymity and km-anonymity. It has a hierarchy creator and editor that allows the user to tailor the anonymization requirements to and balance between privacy and data utility.
- **Anonimatron**: A tool that pseudonymizes datasets. It can be used to generate pseudonymized production data to find a bug or do performance tests outside of the client's production environment.
- **Presidio**: A tool that helps to ensure that sensitive data are properly managed and governed, providing fast analysis and anonymization modules for private entities both in text and images. The Personal Information Identifier analysis module involves identifiers such as credit card numbers, names, locations, social security numbers, bitcoin wallets, US phone numbers, financial data and more.

An important aspect of the architecture is its confidentiality capabilities on data transmissions. It must be noted that FaaS developers should consider how their user's data is transferred between each process or service and evaluate the needs and privacy requirements of the underlying data. Furthermore, the integrity of the underlying data is an aspect that should be considered, alongside confidentiality, to provide a fundamental security overlay on PHYSICS. Both security aspects can be addressed through reusable components and flows that can be properly overlayed on top of any other service or process..

2.14.10 Smart Contracts

In the cloud context, the term serverless is often associated with FaaS, a particular serverless component type that allows hosting business logic in serverless architectures. As shown in [219] the distributed peer-to-peer nature of Blockchain makes it an interesting consideration for enhancing the capabilities of serverless architectures as the resource allocation and resource management tasks do not necessarily need to be performed directly by the users. In [220] a blockchain-based serverless platform is introduced which takes advantage of the capacity provided by underutilised personal computers to run serverless tasks, considering the modern massive computational resource requirements. Modern cloud application developers rely on a wide range of available cloud service models that allow for flexibility by making trade-offs between out-of-the-box integrations and user control over the infrastructure. Especially with FaaS, developers can deploy custom code blocks that can be triggered by events from multiple provider-managed cloud service offerings. These cloud service offerings include emerging new paradigms such as BaaS [221] which can combine the high computing power of cloud computing, the pervasiveness of IoT and the decentralisation of blockchain ensuring the openness and transparency of the system.
Distributed Ledger Technology (DLT) is a protocol that enables the secure functioning of a decentralised digital database, allowing for storage of all information in a secure and integral manner using cryptographic functions. These distributed ledgers can be accessed using "keys" with their corresponding cryptographic signatures. Once the information is stored inside the ledger, it becomes an immutable database that is governed by the rules of the decentralised network. One of the most well-known uses of DLT is blockchain technology.
In a blockchain network the participants can deploy functional code blocks that can be invoked to perform automatic procedures in the ledger. These code blocks, called Smart Contracts (SC), provide secure and transparent means for accessing and changing data inside the ledger by utilising cryptographic signatures. Their corresponding "key" inside the ledger is the address at which the SC is deployed.
To invoke a Smart Contract, a client application formulates a cryptographically signed request message, known as a transaction, containing entries like the address of the Smart Contract, the signature of the specific function to be invoked and the arguments passed to it, and sends it to one of the network peers, known as blockchain nodes. The node then validates the transaction, and includes it in a distributed, blockchain-specific consensus mechanism that ensures all honest peers agree on the contents of the transaction and its global order among other transactions. The result of this mechanism is a block of transactions that is appended at the end of a fully replicated list of blocks, which are chained by their hashes. This list is known as the blockchain data structure, and the way it is organised, in addition to the consensus mechanism, both ensure that the stored transactions are practically immutable.
In this context Smart Contracts can enable useful properties for the PHYSICS architecture, ensuring immutability for transactions made, facilitating accountable interactions, and delegating the business logic on digital assets. From a study conducted in [222] for the adoption of Smart Contracts from different organisations, the research shows that transparency and trust were the main benefits from incorporating them to their business models and several limitations are identified for the capabilities of blockchain technology which are mainly focused on the existing consensus protocols.

3 REQUIREMENTS ELICITATION

3.1 Elicitation Methodology

The methodology used to gather the requirements was based on the vision of each partner for the PHYSICS project. More specifically, each partner was asked to create internally draft visions and uses of the PHYSICS solution and based on this, they extracted requirements (functional and not) that fulfil this vision and allow for a seamless implementation of the envisioned scenario. This allows each partner to participate in the building of the project from its foundation which in turn will allow for a seamless execution of the project plan, given that the requirements of each partner are taken into consideration from the beginning. These requirements will evolve and expand as the project matures, something that will be depicted in the next version of this deliverable. Each partner was asked to document their created requirements within a template that follows well understood and mature standards.
The aim of the envisioned methodology is to simulate the ISO29148 standard. What we want to do is to allow the partners to initially create an internal requirement-driven vision of the project and on top of this to create informed and detailed requirements that can be used in a technical level for the creation of the product. These requirements will be further enhanced and detailed in the second iteration of the deliverable, something that will result in more specific and detailed requirements. This way, we partially skip the first abstract requirements of ISO29148 by having the internal visions of each partner and then we create directly in the first version detailed requirements which will be used as a basis for the mid-term of the project and for the second version of the requirements and essentially having two detailed versions of requirements, something that allows for a rapid initial development of the project. This procedure is depicted in Figure 12.

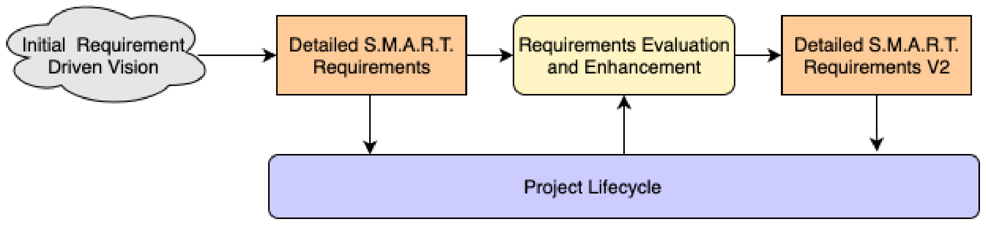

Initial Requirement Driven Vision
Detailed S.M.A.R.T. Requirements
Requirements Evaluation and Enhancement
Detailed S.M.A.R.T. Requirements V2
Project Lifecycle

Figure 12 - PHYSICS Requirements Gathering Methodology

3.1.1 Structure and Standards

The overall approach when it comes to requirement specification follows well known and proven standards. The basis of the overall approach is the usage of the widely used S.M.A.R.T. tool. This tool is a mnemonic that gives five criteria that aim to guide the proper identification and selection of objectives and requirements. These criteria, according to the tool, the defined requirements must be (i) **Specific** in their initial definition (ii) **Measurable** with tight milestones and results, (iii) **Attainable**, (iv) **Relevant** to the overall plan and (v) **Timely** with specific delivery dates. This tool will allow for a detailed specification, management and tracking of each requirement to ensure that project objectives are met.
In addition to the aforementioned methodology, the MoSCoW approach was used to prioritize the requirements within the context of the Specific field. More specifically, this approach defines four levels of priority so as to allow a unified prioritization amongst all requirements. The levels are (i) M for must have, (ii) S for should have, (iii) C for could have and (iv) W for will not have but could be a future enhancement. Furthermore, the ISO25010 was used to create a homogenous categorization of the requirements. The ISO25010 specification defines a set of metrics for software quality, these metrics perfectly align withy and are translated to requirement categories. Given the additional complexity of a project, we added the data category that aims to cover any requirements that focus on data. All in all, the following categories are defined:

- Functional Suitability (FUNC): For requirements that define functional and behavioural attributes.

- Data (DATA): For requirements that define data dependencies.
- Usability (USE): For requirements that define ease of use attributes.
- Reliability (REL): For requirements that define attributes that focus on reliability and dependability.
- Security (SEC): For security-focused requirements.
- Performance Efficiency (PERF): For requirements that describe performance needs.
- Compatibility (COMP): For interoperability requirements.
- Maintainability (MAINT): For requirements that describe ease of maintenance of the solution.
- Portability (PORT): For requirements that describe needs for portability and replaceability.

3.1.2 Template

In order to attain consistency and uniformity of the aggregated requirements, a template was used by all partners that modelled the aforementioned structure and standards. The template has the following fields based on the S.M.A.R.T. approach:

Specific: The aim of this group of fields is to make the requirement specific and to ensure that it is on point.

- **ID:** A unique identifier for the requirement. It has the structure Req-TASK-ALPHANUM where Req is a constant, TASK is the task number related to the requirement and ALPHANUM a small and descriptive alphanumeric value. (e.g., Req-5.1-ResKnow)
- **(Optional) Dependencies:** This is a list of possible dependencies for the requirement. The dependencies are listed with their IDs.
- **Type:** The type of the requirement. The type should be one of the pre-specified types which are based on the ISO 25010 standard.
- **Short Name:** A short but descriptive name for the requirement.
- **(Optional) Actors:** Any actors involved in the requirement. This field is optional since some requirements might not have any easy to identify actors. The aim of this is to better specify the requirement and possibly make it easier to understand.
- **Description:** A description of the requirement.
- **(Optional) Additional Information:** Any additional information for the requirement.
- **Priority:** The importance of the requirement. It follows the MoSCoW approach to specify the priority of the requirement.

Measurable: The aim of this field is to specify any metrics and milestones that will be used as measures of goal achievement.

Attainable: This field defines the attainability of the requirement. This should align with the priority of the requirement.

Relevant: This field will investigate the relevancy of the requirement to the project. The aim here is to ensure that the requirement is as close to the project as possible.

Timely: The final field of the S.M.A.R.T. approach specifies the time by which the requirement should be met. Here there can be many definitions that mainly refer to the project such as task completion dates, milestones, project months etc.

The focus of this template is to be as concise as possible while maintaining an optimized amount of information that will allow it to be easily translatable to technical terms for the implementation tasks to understand. The entire template is available in ANNEX I.

3.1.3 Openness and Flexibility

Although the skeleton of a requirement is pre-determined and essentially its use is mandatory, the actual contents of the deliverable can be flexible. More specifically, the short name, the description and additional information areas are by their nature arbitrary inputs that have small limitations. Furthermore, the actors area is an optional input that serves as an additional definition layer that could be used to better describe the requirement or add information that will be useful during the implementation phase. The M(easurable) area allows authors to specify how the requirement will be measured in open text; the same goes for A(chievable), R(elevant) and T(timely) where authors specify how achievable and how relevant to the project the requirements are and when the requirements should be satisfied in open text. Finally, although

in the dependencies it is recommended to add the IDs of the requirements that the requirement depends on, authors are free to define other dependencies so as to be more flexible and have better capabilities to exactly define their requirements.
The intuition behind this choice is to allow authors to be as descriptive or abstract as they require for the requirement they are specifying. This was chosen due to the fact that different kinds of requirements have different needs and can be specified in more detailed or vague manner. For example, one requ irement could specify a strict policy that requires the usage of specific algorithms and technologies, while other requirements are not so technical and might require conformity to some laws and regulations. Finally, these are the requirements as they are understood in month M4 of the project, as the project matures and it is better understood, these requirements might change, evolve, deprecate or new requirements might occur. This is why a second version of this deliverable is scheduled for month M21, wher e the partners of the PHYSICS project will have the chance to depict their updated vision of the project in a more concrete manner. These final requirements will be compared with their initial version and tracked until the end of the project for their fulfilment.

3.2 PHYSICS Requirements – Second Version

In this chapter there is the documentation of each requirement as they are provided by each partner. Each requirement was grouped by the PHYSICS layer that they affect the most. It is worth noting here that a requirement could potentially affect multiple layers or even layers that were not predicted in this initial state. These changes will be taken into account in the second version of the deliverable and will be investigated throughout the project.

3.2.1 Cloud Design Environment Requirements

Req-3.1-WorkflowDef

Section		Description
S	ID	Req-3.1-WorkflowDef
	Dependencies	-
	Type	FUNC: Functional Completeness
	Short name	Define workflows on FAAS platform
	Actors	FAAS platform, Visual Design Environment
	Description	FaaS platform needs to have workflow definition abilities to allow development of cloud patterns and applications that require specific execution sequence as it is represented in the development Visual environment. The Visual Environment needs to translate the designed flow in the respective workflow definition specification of the platform.
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	This requirement will be identified as complete, if the FaaS platform can accept, in any format, the definition of an application workflow and execute different functions or runtimes in sequence or through any arbitrary defined structure (including branches etc.).	
A	Many FaaS platforms (such as Openwhisk or IBM cloud) provide the ability to create workflows by calling an action from another action. For this reason, we believe this is an achievable requirement.	
R	The goal of this requirement is to allow the development applications and cloud patterns in a flow-like way that will represent what a developer designs in the Visual Environment.	

T	This should be one of the firsts priorities as Visual Cloud Application Workflow Design Environment and Cloud Design Patterns as FaaS Template Nodes highly depend on it. Therefore, it should be available by Phase 1 of the project (M13 prototype-M15 Integration). The ability for any arbitrary structure and branch can be implemented by Phase 2 of the project.

Req-3.1-SupportedRuntimes

	Section	**Description**
S	ID	Req-3.1-SupportedRuntimes
	Dependencies	
	Type	USE: Operability
	Short name	Supported runtimes on Visual Editor
	Actors	Visual Editor
	Description	The Visual Editing environment should support more languages, other than NodeJS, based on the supported runtimes of the FaaS platform.
	Additional Information	This means that the Visual Editing environment should also do the appropriate syntax highlighting and checking to help developers with their development.
	Priority (MoSCoW)	Should-have
M	This requirement will be identified as complete, if the Visual Environment allows the development in more than just one of the supported runtimes.	
A	For this requirement to be implemented it is required to make appropriate changes in the Visual Editor that will be used (for example Node-RED). Furthermore, many code editors today allow development on multiple runtimes and offer the appropriate syntax highlighting and error checking to help the developer. For this reason, we believe this is an achievable requirement.	
R	The goal of this requirement is to allow application developers to develop their applications on multiple runtimes, based on the supported runtimes from the FaaS platform.	
T	Although the Visual Environment should support the same runtimes that the FaaS engine supports this is not a very-high priority. This requirement can be assessed after Phase 1 of the project (M13 prototype-M15 Integration)	

Req-3.1-UploadCustomImages

	Section	**Description**
S	ID	Req-3.1-UploadCustomImages
	Dependencies	Req-3.1-CustomDockerImages
	Type	USE
	Short name	Upload custom images functionality
	Actors	Visual Environment
	Description	If the FaaS platform supports it, the Visual Environment should provide developers the ability to upload their own custom docker images and use them as function nodes.
	Additional Information	
	Priority (MoSCoW)	Should-have

M	This requirement will be identified as complete, if the Visual Environment allows them to upload custom docker images and use them as function nodes.
A	For this requirement to be implemented it is required to make appropriate changes in the Visual Editor that will be used(for example Node-RED). Registry facilities are also required in order to store the uploaded image.
R	The goal of this requirement is to allow application developers to upload their own custom docker images, if the FaaS platform supports it, and use them as functions. With this we provide developers greater freedom on their applications and ease of porting of existing applications.
T	This requirement can be handled in Phase 2 of the project (M30-32).

Req-3.1-MultiTenancy

	Section	**Description**
S	ID	Req-3.1-MultiTenancy
	Dependencies	
	Type	PERF
	Short name	Multi Tenancy
	Actors	Visual Design Environment
	Description	The Design Environment should have the possibility to define specific branch names different for each user. The branch name will be used in the build process for that user flow. It allows one backend service for handling building and deploying processes for multiple users.
	Additional Information	
	Priority (MoSCoW)	Must have
M	Input for the branch name must be placed on the UI. Entered branch name must be remembered by the browser. That name must be sent to the build process and deployment process.	
A	This requirement is achievable by extending necessary backend requests by branch name property.	
R	Purpose for this feature is to have one job defined on Jenkins that is handling builds from multiple branches, and also one backend service processing data from multiple users.	
T	This is the first priority, needed to invoke a job on Jenkins that will build flow for separated branches.	

Req-3.1-LogsService

	Section	
S	ID	Req-3.1-LogsService
	Dependencies	
	Type	PERF
	Short name	Logs Service
	Actors	Visual Design Environment

	Description	There should be one place for collecting logs coming from each microservice. Logs should have the possibility of passing JSON objects for debugging purposes. Viewing logs should be available from the panel on the UI.
	Additional Information	
	Priority (MoSCoW)	Should have
M	Service collecting logs should be created. Other microservices should be connected to it.	
A	This requirement can be fulfilled creating a new custom implementation of service collecting logs, or using an open source one, for example Grafana Logs.	
R	This requirement aims to make the application easier for debugging, tracking the processes engaging multiple microservices.	
T	It should be finished before the application starts to be used by multiple users.	

Req-3.1-BuildsHistory

	Section	
S	ID	Req-3.1-BuildsHistory
	Dependencies	
	Type	PERF
	Short name	Builds History
	Actors	Visual Design Environment
	Description	Built flows page displays information about builds done by the Design Environment. List is presented in the table that contains information about time, branch and name of the function that will be deployed on the OpenWhisk.
	Additional Information	
	Priority (MoSCoW)	Should have
M	Page with flow build information should display a table for each flow with details like time, branch and function name.	
A	This requirement is achievable by storing additional properties for the builds in the database.	
R	The goal of this requirement is to have information needed to differentiate results of builds for different users displayed on the UI.	
T	It should be ready after the multi-tenancy feature.	

Req-3.3-PatternDocumentation

	Section	**Description**
S	ID	Req-3.3-PatternDocumentation
	Dependencies	Req-3.1-WorkflowDef,Req-3.2-RequirementsCoverage
	Type	FUNC: Functional Completeness
	Short name	Documentation and Semantic Enrichness of Design Patterns

	Actors	Visual Environment, Semantic Framework, application developer
	Description	The exposed patterns from T3.3 need to be completely documented so that the application developer is aware of their structure, operation, effects and outcomes as well as how to incorporate them in their application. Furthermore, semantic tags that can be included in the PHYSICS ontology and therefore enable reasoning over the needed functionalities that can be enhanced by the pattern are needed.
	Additional Information	
	Priority (MoSCoW)	Must-have
M	Each pattern should be documented with characteristics such as: ● One indicative use case or testing flow case where applicable ● Expected benefits (if quantifiable) ● Anticipated limitations or values/cases in which the pattern no longer becomes useful, if applicable ● Adaptation needed by the application (if any) and the way the pattern is configured (via the incoming message or the provided UI) This information should also be visible to the application developer at the visual environment layer.	
A	The requirement is achievable, since it relates to the richness and completeness of information that is available to the application developer and should be completed by the pattern developers. The only aspect that may not be attainable in some cases is the boundaries of operation after which the pattern no longer becomes useful. For these cases indicative comments or rules of thumb may be used.	
R	The goal of this requirement is to enable the use of readymade functionality adapted to the cloud and/or FaaS design principles in the application structure with limited knowledge or effort by the application developer. To this end it is linked to Objective - 2: "*Enhance transparency, abstraction and application development reuse through intuitive flow programming approaches incorporating typical cloud design patterns structures*" of the project.	
T	This activity is expected to go hand in hand with the actual availability of the patterns. Therefore, based on the pattern production, it is expected to have completed in Phase 1 of the project (M13 prototype-M15 Integration) for the patterns that are made available in M13, and accordingly for the ones that are delivered in Phase 2 of the project (M30-32).	

<u>Req-3.3-PatternApplication</u>

	Section	**Description**
S	ID	Req-3.3-PatternApplication
	Dependencies	Req-3.1-WorkflowDef
	Type	FUNC: Functional Completeness
	Short name	Application of Design Patterns by Application Developers
	Actors	Visual Environment, Application Developer, FaaS platform
	Description	The exposed patterns from T3.3 need to be seamlessly integrated into the application structure. To this end, they need to be exposed in the Visual environment and directly integrated into the application workflow, while configurable, if applicable, through the environment.

		The mechanism for enforcing the pattern should be seamlessly deployed along the application in the FaaS platform.
	Additional Information	
	Priority (MoSCoW)	Must-have
M	Each pattern should be successfully deployed and operated along the normal application components, not affecting their functional correctness.	
A	For this requirement to be implemented, a specific docker image is needed per pattern that contains the implementation logic. Also, a relevant node is needed at the Visual editor. In some cases, a minor configuration step may be needed in order to set it up.	
R	The goal of this requirement is to enable the use of readymade functionality adapted to the cloud and/or FaaS design principles in the application structure with limited knowledge or effort by the application developer. To this end it is linked to Objective – 2: "*Enhance transparency, abstraction and application development reuse through intuitive flow programming approaches incorporating typical cloud design patterns structures*" of the project.	
T	This activity is expected to go hand in hand with the actual availability of the patterns. Therefore, based on the pattern production, it is expected to have completed in Phase 1 of the project (M13 prototype-M15 Integration) for the patterns that are made available in M13, and accordingly for the ones that are delivered in Phase 2 of the project (M30-32).	

Req-3.3-ParallelContainerExecution

	Section	**Description**
S	ID	Req-3.3-ParallelContainerExecution
	Dependencies	
	Type	USE
	Short name	Spawn multiple containers to execute same function with different data in parallel
	Actors	FaaS-Platform
	Description	FaaS-Platform should provide the ability on a pattern to request the creation of multiple containers that execute the same function with different data to achieve parallel processing(MapReduce patterns & MPI patterns).
	Additional Information	
	Priority (MoSCoW)	Could-have
M	This requirement will be identified as complete, if FaaS-Platform provides the ability for a pattern to specify the number of containers that will be created for a specific parallel execution workflow.	
A	Although obstacles can be found some FaaS platforms support this feature and allow concurrent function executions.	
R	The goal of this requirement is to allow the implementation of parallel execution workflows by avoiding multiprocessing inside containers which is not recommended	
T	This activity is expected to go hand in hand with the availability of some patterns that require parallel execution. Therefore, based on the pattern production, it is expected to have	

	completed in Phase 1 of the project (M13 prototype-M15 Integration) for the patterns that are made available in M13, and accordingly for the ones that are delivered in Phase 2 of the project (M30-32).

Req-4.4-funcNode

	Section	Description
S	ID	Req-4.4-funcNode
	Dependencies	Req-3.1-WorkflowDef
	Type	PORT
	Short name	Incorporate as a functional node
	Actors	-
	Description	Incorporate as a functional node in the design environment of T3.1 in order to enable its usage by the application-level function blocks.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Verify the correct deployment of the in-memory state service as a functional node.	
A	This requirement is achievable, as it is one of the main goals of task T4.4.	
R	The objective of this requirement is to integrate the in-memory state service with the rest of components and allow its use by the application-level function blocks.	
T	D4.2 M30	

3.2.2 Semantic Framework Requirements

Req-3.2-WorkflowCoverage

	Section	Description
S	ID	Req-3.2-WorkflowCoverage
	Dependencies	Req-3.1-WorkflowDef
	Type	DATA
	Short name	Coverage of Workflow Programming Attributes
	Actors	FaaS platform
	Description	The PHYSICS Ontologies/Metamodels must include every attribute of functional workflow programming, in a way ingestible by Semantic Web tools.
	Additional Information	Reuse of workflow Ontologies in the inception of the application description metamodel.
	Priority (MoSCoW)	Must-have
M	This requirement will be identified as complete, if the attributes of the workflows are all included in the resulting PHYSICS ontology about the application characteristics description.	
A	This requirement is achievable through knowledge engineering. It is a matter of matching the semantic terms/classes that have to do with the workflow, making sure that they are all derived by the visual workflow tool, and connecting them with terms that express constraints or requirements for functions/nodes.	
R	This requirement is about the creation of the basic PHYSICS ontology about the application characteristics description of the metamodel for the workflows. This will help create the	

	basic graph of the workflow, and its attributes. It will also be the basis for the annotation of the workflow's components with additional attributes, constraints and requirements.
T	This is of one of the highest priority requirements, as it is an essential part of the semantic description of the functional workflows. It should be one of the first things to be implemented in the PHYSICS ontology.

Req-3.2-RequirementsCoverage

	Section	**Description**
S	ID	Req-3.2-RequirementsCoverage
	Dependencies	Req-3.2-WorkflowCoverage
	Type	DATA
	Short name	Coverage of function/component requirement attributes
	Actors	FaaS platform
	Description	The PHYSICS Ontologies/Metamodels must include sufficient attributes for the requirements a workflow component may have regarding hardware, software or location.
	Additional Information	Specifically provide properties and possible value domains for hardware, software or location requirements a workflow component has. Due to the relationship with T5.1, in the next version there might be additional dependencies from T5.1.
	Priority (MoSCoW)	Must-have
M	This requirement will be identified as complete, if the requirements expressed in the PHYSICS application characteristics descriptions cover the needs of the project's use cases, at least.	
A	This requirement is achievable through knowledge engineering and gathering the attributes that may be required in each indicative use-case scenario.	
R	The requirements an application workflow component may have are covered as much as possible. Links with the service descriptions of Task 5.1 must also be created.	
T	This is one of the highest priority requirements, as it is an essential part of the semantic description of the requirements of the components of the functional workflows. It should be one of the first things to be implemented in the PHYSICS ontology.	

Req-3.2-ConstraintsCoverage

	Section	**Description**
S	ID	Req-3.2-ConstraintsCoverage
	Dependencies	Req-3.2-WorkflowCoverage
	Type	DATA
	Short name	Coverage of QoS and performance constraint attributes
	Actors	FaaS platform
	Description	The PHYSICS Ontologies/Metamodels must include sufficient QoS attributes that a workflow component may have.
	Additional Information	Performance metrics or Cloud SLA terms have to be reused and organized as insertable values in the workflow components. Due to the relationship with T5.1 and the optimizer, in the next version there might be additional dependencies from these components.
	Priority (MoSCoW)	Must-have

M	This requirement will be identified as complete, if the constraints expressed in the PHYSICS application characteristics descriptions cover the needs of the project's use cases, at least.
A	This requirement is achievable through knowledge engineering, inclusion of QoS parameters found in T5.1 service description, and investigating the constraints used in the performance optimization process.
R	This requirement ensures that application descriptions include QoS and performance-indicator values that will be used accordingly by both the reasoning framework and the optimizer of PHYSICS.
T	This is one of the highest priority requirements, as it is an essential part of the semantic description of the requirements of the components of the functional workflows. It should be one of the first things to be implemented in the PHYSICS ontology.

Req-3.2-LinkWithVocabularies

	Section	Description
S	ID	Req-3.2-LinkWithVocabularies
	Dependencies	-
	Type	DATA
	Short name	Link PHYSICS ontology terms with other Vocabularies
	Actors	FaaS platform
	Description	The PHYSICS Ontologies should have links with other ontologies and follow the linked data paradigm.
	Additional Information	Target common ontologies to reuse terms from, or make links to, when creating the PHYSICS application characteristics ontologies. Due to the relationship with T5.1, in the next version there might be dependencies from T5.1.
	Priority (MoSCoW)	Should-have
M	Create a link with at least one existing ontology per domain included in the PHYSICS application characteristics descriptions.	
A	This requirement is achievable through knowledge engineering and analyzing ontologies of related domains to the ones PHYSICS encompasses.	
R	This requirement helps in both of the following ways: Assist in the knowledge engineering process, by discovering terms, and relationships between them, that have been thought of by other researchers. Makes the resulting PHYSICS ontology reusable in future work related to the semantic web, and modelling of logic in general.	
T	This is an essential part of the semantic description of the requirements of the components of the functional workflows. It should be one of the first things to be implemented in the PHYSICS ontology.	

Req-3.2-ReasoningCapability

	Section	**Description**
S	ID	Req-3.2-ReasoningCapability
	Dependencies	-
	Type	USE
	Short name	Reasoning capabilities using the PHYSICS ontology
	Actors	FaaS platform

	Description	The PHYSICS Ontologies/Metamodels must be effectively usable within reasoners.
	Additional Information	Application description models should, in combination with service descriptions, be usable in the reasoning framework of PHYSICS. Due to the relationship with T4.1 and T5.1, in the next version there might be dependencies from these tasks.
	Priority (MoSCoW)	Must-have
M	The PHYSICS Ontologies/Metamodels must be usable as terminological boxes of knowledge bases with reasoners and should be capable of constituting an effective entailment regime on SPARQL queries of individuals, yielding more knowledge as assertions.	
A	This requirement is achievable through: knowledge engineering, the testing on common reasoners and SPARQL querying over example individuals that follow the classes of the resulting Ontology, given the OWL Ontology itself as an entailment regime.	
R	A SPARQL query over example data/individuals, yields more assertions that are a result of a basic OWL entailment regime, than the data originally created. This will ensure the usability of the model within a reasoner that extends beyond simple OWL reasoning.	
T	This requirement comes at a second stage, as the effort will be put into the inclusion of the resulting ontology in the reasoning engine as its terminological box (TBOX). It is however essential for the reasoning engine.	

Req-3.2-ExpressivityRichness

	Section	**Description**
S	ID	Req-3.2-ExpressivityRichness
	Dependencies	Req-3.2-WorkflowCoverage, Req-3.2-RequirementsCoverage, Req-3.2-ReasoningCapability
	Type	USE
	Short name	High Expressivity: Attribute Richness & Relationship Richness
	Actors	FaaS platform
	Description	The PHYSICS Ontologies/Metamodels should achieve at least acceptable attribute richness and relationship richness, in order to indicate expressiveness.
	Additional Information	Attribute richness is an indicator of how rich the description of a class is. Relationship richness is an indicator of the achievable variety through connections, and a high level of achievable detail.
	Priority (MoSCoW)	Should-have
M	The PHYSICS Ontologies/Metamodels should achieve attribute richness and relationship richness same as or higher than the average of the other linked ontologies in its domain.	
A	This requirement is achievable through: knowledge engineering, creation of meaningful predicates/properties between classes/terms wherever possible and the testing on common reasoners	
R	Make the resulting ontology as expressive as possible, in order to ensure that additional knowledge and assertions lead to the reduction of the effective search space of the optimizer, through a reasoning process.	
T	This requirement comes at a second stage, as the effort will be put into the inclusion of the resulting ontology in the reasoning engine as its terminological box (TBOX). It is however essential for the reasoning engine.	

Req-4.1-Adaptation

	Section	Description
S	ID	Req-4.1-Adaptation
	Dependencies	-
	Type	FUNC
	Short name	Runtime Adaptation
	Actors	Semantics, Service Semantic Models, Global Continuum Placement
	Description	The deployment graph, produced by the Inference Engine, will be updated during runtime based on the performance of the utilized resources.
	Additional Information	Due to the relationship with T3.2, T4.3 and T5.1, in the next version there might be dependencies from these tasks.
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Frequency of required changes in the deployment graph.	
A	Runtime Adaptation requires appropriate system architecture in order for updated inputs to trigger a new deployment graph calculation.	
R	This requirement ensures that misbehavior in the deployment of a given function will be handled appropriately.	
T	Initial version at M13. Integration with the other components at M15. Updated version at M30. Reintegration with the other components at M32.	

Req-4.1-Inputs

	Section	Description
S	ID	Req-4.1-Inputs
	Dependencies	Req-5.1-Compatibility
	Type	DATA
	Short name	Inference Engine (Reasoning framework)
	Actors	Semantics, Service Semantic Models, Global Continuum Placement
	Description	The Inference Engine takes as input two semantic models (i.e., T3.2, T5.1), one describing the application to be deployed and another describing the available cloud resources. The output is the deployment graph where each function of the application will be connected to certain resources capable of running the given function.
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Domain coverage by ontologies, Semantic distance/similarity of graph nodes	
A	Application and Services Ontologies will be developed in the context of PHYSICS; thus, they could be aligned in order to optimize the required reasoning. Dataset with resources and deployments of FaaS are difficult to find, however some partners or other EU projects might provide such data.	
R	Availability of data is mandatory in order to start developing the Inference engine. The data format (e.g., JSON-LD, XML) of the inputs and the output of T4.1 should be defined. Dataset	

	with resources and deployments of FaaS applications are required in order to develop ML-based Reasoning.
T	Initial version of ontologies at M8. Dataset with resources and deployments at M15. Updated version of ontologies at M17. Dataset with resources and deployments from PHYSICS platform at M20.

Req-4.1-Latency

	Section	**Description**
S	ID	Req-4.1-Latency
	Dependencies	Req-4.1-Reasoning
	Type	PERF
	Short name	Inference Engine Efficiency
	Actors	
	Description	The Reasoning performed by the Inference Engine should be both valid and instant in order for PHYSICS to operate efficiently.
	Additional Information	Due to the relationship with T4.3, in the next version there might be additional dependencies from T4.3.
	Priority (MoSCoW)	S: Should-have. Desirable requirement.
M	Correctness of this requirement will be determined by its applicability within the PHYSICS ecosystem.	
A	Given that application and services ontologies are properly designed this requirement will be achievable.	
R	This requirement will enhance the overall performance of PHYSICS.	
T	After the initial deployment of the Inference engine .	

Req-4.1-ML_Reasoning

	Section	**Description**
S	ID	Req-4.1-ML_Reasoning
	Dependencies	Req-4.1-Inputs
	Type	FUNC
	Short name	Machine Learning based Reasoner
	Actors	Global Continuum Placement
	Description	The reasoning between requirements and application resources is done with Machine Learning techniques.
	Additional Information	
	Priority (MoSCoW)	C: Could-have. Optional requirement
M	Extra utility in T4.3 in comparison to Req-4.1-Reasoning.	
A	ML-based Reasoning requires R&D and proper data.	
R	ML based Reasoning will be capable of eliminating the search space of the T4.3.	
T		

Req-4.1-Reasoning

	Section	**Description**
S	ID	Req-4.1-Reasoning

	Dependencies	Req-4.1-Inputs
	Type	FUNC
	Short name	Inference Engine (Reasoning framework)
	Actors	Semantics, Service Semantic Models, Global Continuum Placement
	Description	The Inference Engine takes as input two semantic models (i.e., T3.2, T5.1), one describing the application to be deployed and another describing the available cloud resources. The output is the deployment graph where each function of the application will be connected to certain resources capable of running the given function.
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Required time for reasoning (Number of SPARQL queries, query mean response time)	
A	Rule-based Reasoning will be implemented by an open-source framework for building Linked-Data applications (e.g. Apache Jena).	
R	This requirement will provide the tools to use in a functional context the semantic descriptions created by T-3.2 and T-5.1, through exposing them in a service-oriented manner as well as offer reasoning capabilities and semantic inference. Req-4.1-Reasoning will also aid T-4.3 in the determination of the benefit of using a specific service type instead of another and it will act as a first level filter to minimize the number of candidate services that may be used for the deployment of a given application graph, enhancing the optimization process of T-4.3.	
T	Initial version at M13. Integration with the other components at M15. Updated version at M30. Reintegration with the other components at M32.	

Req-5.1-ResKnow

	Section	**Description**
S	ID	Req-5.1-ResKnow
	Dependencies	-
	Type	DATA: Data preconditions
	Short name	Input information on resources
	Actors	-
	Description	For the service semantics component to be operational, the component needs to be aware of which resources to describe and their respective characteristics
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	This requirement will be identified as complete, if the service semantics component can receive information that will directly or indirectly lead to the semantics modelling.	
A	Although it is thought to be achievable, certain obstacles can occur such as how to acquire the properties from different cloud vendors and edge devices. The variety of resource types will most probably require a unique approach to collecting these data via a single middleware. Additionally information on domains such as environmental impact, i.e. energy consumption	

	can be very difficult to obtain. As such the component should include methods for manual input of the respective fields in the ontology.
R	The scope of this requirement is to collect resource properties. This is the backbone of the service semantics component that will in turn enable a functional modelling.
T	This should be the first priority of the component, given that every other functionality of the component depends on it. As a result, this requirement will be assessed during the first year of the project.

Req-5.1-Interface

	Section	**Description**
S	ID	Req-5.1-Interface
	Dependencies	-
	Type	USE: Operability
	Short name	Component's operational interface.
	Actors	Application developer
	Description	An interface that interested actors can find an overview of the resources available.
	Additional Information	
	Priority (MoSCoW)	C: Could-have. Optional requirement.
M	-	
A	This requirement is thought to be achievable.	
R	The main goal is to provide an additional graphical interface through a web application where resources and their respective properties can be conceptualized graphically.	
T	This requirement will be addressed during the second year of the project after the initial design has concluded and a prototype have been developed.	

Req-5.1-Compatibility

	Section	**Description**
S	ID	Req-5.1-Compatibility
	Dependencies	Req-4.1-Inputs
	Type	COMP: Interoperability t
	Short name	Compatible outcome with co-existent components
	Actors	Inference Engine
	Description	The format of this component's outcome should allow its use by other components interested in the service semantics.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Correctness of this requirement will be determined by its applicability within the PHYSICS ecosystem.	
A	With the required amount of communication and design between involved partners this requirement will be achievable.	
R	This requirement ensures that the components involved in utilizing the semantics description will have a defined format of inputs and outputs that allows interoperability.	

T	This requirement will be focused on at the mid to late early stages of design. Hopefully this period will take place during the 6th to 8th month of the project.

Req-5.1-Portability

	Section	**Description**
S	ID	Req-5.1-Portability
	Dependencies	-
	Type	PORT
	Short name	Portability of the Service Semantic Reusables
	Actors	
	Description	Creation of reusable entities that describe the core components of resources to be possibly utilized in an edge application.
	Additional Information	
	Priority (MoSCoW)	S: Should-have. Desirable requirement
M	The portability of the results of this task will be verified by the creation of service semantics for a variety of applications in the edge that utilize a broad spectrum of resources.	
A	Although the complexity of modelling services can be daunting, a set of state-of-the-art methodologies and technologies along with exhaustive research will enable fulfilment of this scenario.	
R	The fulfilment of this requirement is needed in order to extend the resource management components of the PHYSICS programs so as to be extendable in cases that new resources are implemented in relevant scenarios.	
T	This requirement is to be fulfilled upon the first year of the project as it is dependent with the resource allocation process of task 4.2.	

Req-5.1-GraphSeparability

	Section	**Description**
S	ID	Req-5.1-GraphSeparability
	Dependencies	-
	Type	DATA: Data preconditions
	Short name	Identification of different ontology graphs
	Actors	-
	Description	The service semantics component will capture information on more than one different clusters. As such , the produced graphs from semantics modelling should be separable and assigned unique identifiers.
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	This requirement will be identified as complete, if the service semantics component can produce multiple but uniquely identified graphs of each cluster examined .	
A	No obstacles of great impact can be foreseen as there do exist techniques to implement in order to produce unique identifiers for each cluster	

R	The scope of this requirement is to manage output of the component, i.e. provide unique identifiers on produced semantic graphs.
T	This should a high priority, given that components that ingest information from the service semantics will need to identify the different options for deployment in a clear and structured way.

Req-5.1-SemCap

	Section	**Description**
S	ID	Req-5.1-SemCap
	Dependencies	-
	Type	FUNC: Functional Completeness
	Short name	Resource capabilities in service semantics
	Actors	-
	Description	Service semantics should capture all the available resources and their respective capabilities in a way that is meaningful for resource management and allocation through the semantics reasoner.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	The quality of results will be tested through direct applicability in the PHYSICS use cases and other relevant scenarios.	
A	Through the past years several domain specific languages have been utilized in order to create application topologies by defining semantic descriptions of the relevant components. In PHYSICS such an approach is thought to be achievable to realize and implement.	
R	Creating service semantics will enable PHYSICS to manage resources in a meaningful way that allows deployment of applications in the edge and enables a meaningful resource allocation.	
T	This requirement will be a work in progress for the first 18 months of the project in collaboration with the relevant tasks from WP3 and WP4.	

3.2.3 FaaS and Container Platform Requirements

Req-4.4-state

	Section	**Description**
S	ID	Req-4.4-state
	Dependencies	-
	Type	FUNC
	Short name	State between function invocations
	Actors	-
	Description	The distributed management system must maintain the state between function invocations.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Plan and design a validation process to verify that different functions share the state in order to obtain a result combining the results obtained from the previous functions.	
A	This requirement is achievable, as it is one of the main goals of task T4.4.	
R	This requirement aims to share state between different FaaS functions.	

T	D4.1 M13

Req-4.4-interplay

	Section	**Description**
S	ID	Req-4.4-interplay
	Dependencies	-
	Type	FUNC
	Short name	Interplay between the in-memory state and the persistent storage layer
	Actors	-
	Description	Enable numerous functionalities of interplay between the in-memory state and the persistent storage layer.
	Additional Information	-
	Priority (MoSCoW)	C: Could-have
M	Plan and design a validation process to verify the interplay between both the in-memory state and persistent storage layer.	
A	This requirement is achievable, as it is one of the main goals of task T4.4.	
R	This requirement aims to define and implement a process to integrate the state storage using the in-memory state and the persistent storage layer.	
T	Initial version was reported in D4.1 at M13. An updated version will be produced at M30	

Req-4.4-tradeOffs

	Section	**Description**
S	ID	Req-4.4-tradeOffs
	Dependencies	-
	Type	USE
	Short name	Trade-offs between consistency and performance
	Actors	-
	Description	Investigate trade-offs between consistency and performance.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement
M	Write a document with the investigation analysing the trade-offs.	
A	This requirement is achievable, as it is one of the main goals of task T4.4.	
R	This requirement amins to study the trade-offs between consistency and performance and how this affects the development of this task.	
T	D4.1 M13	

Req-4.4-perf

	Section	**Description**
S	ID	Req-4.4-perf
	Dependencies	Req-4.4-state Req-4.4-interplay
	Type	PERF
	Short name	Activation performance

	Actors	-
	Description	Dynamic caching mechanisms and persistent storage functionalities activation performance.
	Additional Information	-
	Priority (MoSCoW)	C: Could-have
M	Measurement of the activation time of functions	
A	This requirement is achievable, as it is one of the main goals of task T4.4.	
R	The goal of this requirement is to evaluate the time it takes for functions to configure with data stored in cache or persistent storage to be ready to run.	
T	D4.2 M30	

<u>Req-4.4-access</u>

	Section	**Description**
S	ID	Req-4.4-access
	Dependencies	-
	Type	USE
	Short name	In-memory access patterns.
	Actors	-
	Description	Different access patterns must be investigated and included such as single client-multiple access, multiple client-single access, multiple client-multiple access.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement

M	**Check state access using different access patterns.**
A	This requirement is achievable, as it is one of the main goals of task T4.4.
R	The goal of the requirement is to implement and verify that state access from different functions is working correctly and all functions write or read the expected data.
T	D4.1 M30

<u>Req-4.5-PersStorage</u>

	Section	**Description**
S	ID	Req-4.5-PersStorage
	Dependencies	Req-3.1-WorkflowDef, Req-3.3-PatternApplication
	Type	FUNC: Functional Completeness
	Short name	Persistent storage on FaaS platform
	Actors	FaaS platform
	Description	FaaS platform should provide persistent storage (Like AWS S3 or Minio) for applications, that also supports notifications based on the content changes.
	Additional Information	-
	Priority (MoSCoW)	Could-have

M	This requirement will be identified as complete, if the FaaS platform can provide the application developers the capability to read, write and update persistent data in FaaS platform during function execution. Notification mechanisms on top of the data are also needed in order to enable triggering of functions based on content availability or change.
A	Many FaaS platforms (such as Openwhisk, AWS Lambda) provide the capability for a function to use persistent volumes, Object Storage or external DBs to handle persistent data so we believe this is an achievable requirement.
R	The goal of this requirement is to allow applications to handle persistent data during their runtime to apply their business logic.
T	Application development can be done without relying on persistent data or with using external resources (such as an external database). For this reason, we believe this requirement does not have a high priority and can be completed in Phase 2 of the project (M30-32).

Req-5.4-optimization

	Section	**Description**
S	ID	Req-5.4-optimization
	Dependencies	
	Type	FUNC
	Short name	Definition and implementation of an optimization process
	Actors	-
	Description	Definition and implementation of an optimization process inside the provider resources
	Additional Information	-
	Priority (MoSCoW)	C: Could-have
M	Compare our developed implementation process with other available systems.	
A	This requirement is achievable, as it is one of the main goals of task T5.4.	
R	The aim of this requirement is to define and implement an optimization process inside the resource's provider.	
T	D5.2 M30	

Req-5.4-co-allocation

	Section	**Description**
S	ID	Req-5.4-co-allocation
	Dependencies	Req-5.4-workloads
	Type	FUNC
	Short name	Co-allocation service
	Actors	-
	Description	This requirement will provide rules for the co-allocation of services in a physical node in order to optimize the performance of the services and do not create contention
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.

M	Compare the performance when co-allocation is used with other strategies for assigning services to physical nodes.
A	This requirement is achievable, as it is one of the main goals of task T5.4.
R	The aim of this requirement is to define rules for co-allocating services in a physical node in order to optimize the performance of all services deployed in the node.
T	D5.2 M30

Req-5.4-workloads

	Section	**Description**
S	ID	Req-5.4-workloads
	Dependencies	
	Type	USE
	Short name	Identify computational nature of the workloads
	Actors	The actors involved in this scenario
	Description	Identify the computational nature of the workloads running in a physical node. In order to define the co-allocation rules.
	Additional Information	
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M		
A	This requirement is achievable, as it is one of the main goals of task T5.4.	
R	The goal of this requirement is to analyse the different natural workloads and bottlenecks to obtain accurate placement decisions.	
T	D5.1 M13	

Req-5.4-AImodels

	Section	**Description**
S	ID	Req-5.4-AImodels
	Dependencies	Req-5.4-UsageModellingOpt
	Type	USE
	Short name	Effects of different deployed services and models combinations
	Actors	-
	Description	Investigate the effects of various combinations of the deployed services and models
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Plan and implement a way to evaluate the different combinations of implemented services and models	
A	This requirement is achievable, as it is one of the main goals of task T5.4.	
R	The goal of this requirement is to study the effects of the different combination of the deployed services and models	
T	D5.1 M13	

Req-5.4-UsageModellingOpt

	Section	**Description**

S	ID	Req-5.4-UsageModellingOpt
	Dependencies	Req-4.5-FaaSandIaaSMonitoring, Req-4.5-FaaSRuntimeAdaptation
	Type	PERF: Time Behavior
	Short name	Exploit Monitoring Data for Modelling and Optimization purposes
	Actors	FaaS Platform, Container Orchestrator Platform
	Description	Exploitation of performance information may be used in a modelling process in order to optimize management practices either at the platform or orchestrator levels.
	Additional Information	The modelling may refer to aspects such as anticipation of request invocation patterns, re-allocation of request sequence in order to optimize warm starts and/or dynamic setting of FaaS platform parameters available from Req-4.5
	Priority (MoSCoW)	Should-have
M	Modelling attempts should produce a Mean Absolute Percentage Error in the area of <20% and/or an according improvement in performance.	
A	The requirement is achievable, following a selection of relevant available benchmark tests. Registration of the respective functions can be performed once, while the workflow of the execution can be handled also through the visual environment of WP3. Careful consideration of benchmarking needs and sequence may be needed based on the selected benchmarks and relevant stages of measurement.	
R	The goal of this requirement is to evaluate the performance of a FaaS platform provider and aid in the selection processes. It can also be used as a mean to evaluate resource management practices by the provider, and it is related to Objective - 3: "*Workflow distribution, functional incorporation and runtime management across the continuum (optimization, placement and reconfiguration)*" and Objective - 4: "*Provider-Local, fine grained runtime management and adaptation through extension of relevant provider interfaces*" of the project.	
T	This activity is expected to be available during Phase 2 of the project (M30 prototype-M32 Integration).	

Req-5.4-validation

	Section	**Description**
S	ID	Req-5.4-validation
	Dependencies	-
	Type	PERF
	Short name	Quantify the performance gains of different deployments of services
	Actors	-
	Description	Investigate the effects of various combinations of services deployments.
	Additional Information	Due to the relationship with T5.2 and T5.3 there might be dependencies from these requirements.
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Plan and implement a way to evaluate the performance of different combinations of deployments	
A	This requirement is achievable, it is one of the main goals of task T5.4.	

R	The goal of this requirement is to study the effects in the performance of different combinations of deployments of services
T	D5.1 M13

3.2.4 Cross Layer Requirements

Req-3.4-Privacy

	Section	**Description**
S	ID	Req-3.4-Privacy
	Dependencies	-
	Type	SEC
	Short name	Privacy for individuals and their data. Compliance with privacy laws and regulations.
	Actors	-
	Description	The handling of data (storage, transmission and processing) should protect the privacy of the users and follow all applying laws and regulations.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	A plan, design and implementation that proves the measures taken to holistically protect the privacy of data-owners. (Mandatory) Compliancy with privacy laws and regulations (Mandatory) Results from a privacy impact assessment (Tentative) Privacy maturity model results (Tentative)	
A	Given the complexity of the PHYSICS architecture, this requirement is achievable, but care must be taken on each step so that privacy is incorporated by-design.	
R	To protect the privacy of the data-owners. This requirement aims to satisfy privacy-related regulatory and legal obligations of the PHYSICS architecture.	
T	This requirement needs to be active throughout the lifecycle of the project. A privacy status assessment must be done before major milestones.	

Req-3.4-SmartContracts

	Section	**Description**
S	ID	Req-3.4-SmartContracts
	Dependencies	-
	Type	SEC
	Short name	External code invocation from Blockchain Network.
	Actors	-
	Description	Smart Contracts deployed in a blockchain network provide the means for delegating data procedures where user authentication and authorization is needed
	Additional Information	-
	Priority (MoSCoW)	S: Should-Have.
M	A cloud design pattern allowing the deployment and interaction with smart contracts.	

A	The needs of this requirement are easy to implement and incorporate.
R	Extend the PHYSICS architecture to leverage the distributed nature of Smart Contracts.
T	This requirement needs to be addressed in the context of pilot cases.

Req-3.4-SmartContractTemplates

	Section	**Description**
S	ID	Req-3.4-SmartContractTemplates
	Dependencies	Req-3.4-SmartContracts
	Type	SEC
	Short name	Templates for automated Smart Contract deployment.
	Actors	-
	Description	A collection of contract templates to automate the procedure of deploying a smart contract to a blockchain network
	Additional Information	-
	Priority (MoSCoW)	C: Could-Have.
M	A cloud design pattern with several contract templates: Authorization Policy Contract Timed Policy Contract Access Control Contract Marketplace Contract	
A	The needs of this requirement are easy to implement and incorporate.	
R	Creation of a template repository for reusability.	
T	This requirement needs to be addressed in the context of pilot cases.	

Req-3.4-CodeAnalysis

	Section	**Description**
S	ID	Req-3.4-CodeAnalysis
	Dependencies	-
	Type	SEC
	Short name	Code and binary analysis of internal and external functions
	Actors	-
	Description	The deployed code and used external libraries should be analyzed for known or potential vulnerabilities. In the best case, this will be an automated process that will allow for a quick security health check of each deployed application. This could be implemented in the form of a cloud design pattern.
	Additional Information	-
	Priority (MoSCoW)	C: Could-have. Optional requirement.
M	A cloud design pattern alongside with functions that execute code analysis or binary analysis of the deployed FaaS components.	
A	Given the readily available tools for automated vulnerability analysis, it is feasible to deploy them or potentially modify them in a FaaS architecture which will serve as a drop-in component for security enhancement.	

R	To enhance the overall security of deployed applications in the context of PHYSICS. This service can also be used in the RAMP where each reusable artefact will be scanned and verified for its security and integrity.
T	This requirement needs to be initially defined in the context of T3.3 and T3.4 while its implementation and evaluation will be checked in WP6.

Req-3.4-DeprMan

	Section	**Description**
S	ID	Req-3.4-DeprMan
	Dependencies	-
	Type	SEC
	Short name	Deprecated Component Management
	Actors	-
	Description	This requirement specifies the need for a method and process for the deprecation and removal of old FaaS components. This is based on the fact that due to rapid development of components, old ones could easily be forgotten, something that poses new security risks.
	Additional Information	-
	Priority (MoSCoW)	C: Could-have. Optional requirement.
M	A cloud design pattern or a methodology for the policy enforcement that manages the deprecated components.	
A	The needs of this requirement are easy to implement and incorporate.	
R	To ensure that all components are not deprecated and exist within the overall architecture for a purpose.	
T	This requirement needs to finish in the context of T3.4	

Req-3.4-Encryption

	Section	**Description**
S	ID	Req-3.4-Encryption
	Dependencies	-
	Type	SEC
	Short name	Encryption Usage
	Actors	-
	Description	Strong encryption must be used in all cases where sensitive data are handled in transit or in storage.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement. Additionally, a could-have requirement is the usage of post-quantum resistant cryptography,
M	The method for measuring this requirement is the algorithms and the corresponding key lengths used for the encryption, decryption, signing, verification, key exchange and hashing throughout the project.	
A	All strong encryption schemes are easy to implement and use. Care must be taken in their configuration so that their guarantees are maintained and not weakened.	

R	The protection of sensitive data in rest and in transit through ought PHYSICS.
T	This requirement needs to be addressed in the context of T3.4.

Req-3.4-Secrets

	Section	**Description**
S	ID	Req-3.4-Secrets
	Dependencies	-
	Type	SEC
	Short name	Secrets deployment to components
	Actors	-
	Description	The secrets to be deployed in each component should be done in a protected and secure manner.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	A method and process for deploying secrets to FaaS PHYSICS components	
A	Given the abundancy of solutions and technologies for secure secret deployment, this requirement is achievable.	
R	To ensure the secrecy of the to-be deployed secrets in each FaaS component.	
T	This requirement needs to be addressed in the context of T3.4.	

Req-3.4-SecureComms

	Section	**Description**
S	ID	Req-3.4-SecureComms
	Dependencies	Req-3.4-Encryption
	Type	SEC
	Short name	Communication security between components
	Actors	-
	Description	The communication between each component and components and other entities need to be authenticated, controlled and secured.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	An identity and access management methodology to be enforced throughout deployed applications. Strong communication and authentication protocols to be used between all components and entities.	
A	Given the complexity of the PHYSICS architecture, this requirement is achievable, but care must be taken on each step so that secure communications and proper access management is used between all components and entities.	
R	To protect the integrity, authenticity and secrecy of all data in transit while also ensuring proper access and identity management in the entire deployed application.	
T	This requirement needs to be addressed in task T3.4.	

Req-4.2-FaaSBenchmarking

	Section	**Description**

S	ID	Req-4.2-FaaSBenchmarking
	Dependencies	Req-4.5-FaaSandIaaSMonitoring
	Type	PERF: Time Behavior
	Short name	Benchmark functions execution towards the FaaS Platform
	Actors	FaaS Platform
	Description	The mechanisms of T4.2 need to execute performance benchmarks towards the FaaS platform in order to evaluate its effectiveness and/or resource management approaches.
	Additional Information	
	Priority (MoSCoW)	Must-have
M	The benchmarking process will be successful if all of the following points are met: ● The benchmarking framework is able to benchmark the user provided functions ● The benchmarking framework undertakes the full lifecycle of benchmark execution (i.e. launching of the benchmark, orchestration of its operation, gathering of results) ● A benchmark test can be run as the result of a service invocation (Benchmarking as a Service)	
A	The requirement is achievable, following a selection of relevant available benchmark tests. Registration of the respective functions can be performed once, while the workflow of the execution can be handled also through the visual environment of WP3. Careful consideration of benchmarking needs and sequence may be needed based on the selected benchmarks and relevant stages of measurement.	
R	The goal of this requirement is to evaluate the performance of a FaaS platform provider and aid in the selection processes. It can also be used as a mean to evaluate resource management practices by the provider, and it is related to Objective - 3: "*Workflow distribution, functional incorporation and runtime management across the continuum (optimization, placement and reconfiguration)*" and Objective - 4: "*Provider-Local, fine grained runtime management and adaptation through extension of relevant provider interfaces*" of the project.	
T	This activity is expected to have finalized for at least one category of benchmarks in Phase 1 of the project (M13 prototype-M15 Integration).	

Req-4.2-CostAssociation

	Section	Description
S	ID	Req-4.2-CostAssociation
	Dependencies	Req-4.2-FaaSBenchmarking, Req-4.5-FaaSandIaaSMonitoring
	Type	DATA: Data Requirements
	Short name	Integration of cost models in performance aspects
	Actors	End user, FaaS platform
	Description	The mechanisms of T4.2 need to associate performance metrics attained through the benchmarking executions with the relevant cost models available in the FaaS domain, in order to indicate to users or selection mechanisms the cost aspect in their selection.
	Additional Information	
	Priority (MoSCoW)	Should-have

M	Cost estimation should be linked with a benchmark execution sequence and presented to the user. along with the relevant performance metrics of an experiment.
A	The requirement is achievable, following a selection of relevant available benchmark tests. Registration of the respective functions can be performed once, while the workflow of the execution can be handled also through the visual environment of WP3. Careful consideration of benchmarking needs and sequence may be needed based on the selected benchmarks and relevant stages of measurement.
R	The goal of this requirement is to evaluate the performance of a FaaS platform provider and aid in the selection processes. It can also be used as a mean to evaluate resource management practices by the provider, and it is related to Objective - 3: "*Workflow distribution, functional incorporation and runtime management across the continuum (optimization, placement and reconfiguration)*" and Objective - 4: "*Provider-Local, fine grained runtime management and adaptation through extension of relevant provider interfaces*" of the project.
T	This activity is expected to have finalized for at least one category of benchmarks in Phase 1 of the project (M13 prototype-M15 Integration).

Req-4.2-MeasurementPropagation

	Section	**Description**
S	ID	Req-4.2-MeasurementPropagation
	Dependencies	Req-4.2-FaaSBenchmarking, Req-4.5-FaaSandIaaSMonitoring, Req-4.2-CostAssociation
	Type	FUNC-Functional Appropriateness
	Short name	Benchmark measurements through REST API
	Actors	FaaS Platform
	Description	The mechanisms of T4.2 need to make performance and cost estimations available through suitable REST APIs for other components of the platform to retrieve them.
	Additional Information	
	Priority (MoSCoW)	Must-have
M	Benchmark results need to be properly propagated to external components in order to retrieve collective and meaningful statistics of benchmarks. Therefore either pull or push methods need to be implemented that will cover metrics such as average times, deviation, percentiles of values.	
A	The requirement is achievable, following the availability of the raw measurement data from the dependent requirements. After this stage, the API creation is a typical layer in front of the measurements DB.	
R	The goal of this requirement is to evaluate the performance of a FaaS/IaaS platform provider and aid in the selection processes. It can also be used as a mean to evaluate resource management practices by the provider, and it is related to Objective - 3: "*Workflow distribution, functional incorporation and runtime management across the continuum (optimization, placement and reconfiguration)*" and Objective - 4: "*Provider-Local, fine grained runtime management and adaptation through extension of relevant provider interfaces*" of the project.	
T	This activity is expected to have finalized at the second stage of the project due to the dependencies from the depending requirements (M30 prototype-M32 Integration).	

Req-4.4-elasticity

	Section	Description
S	ID	Req-4.4-elasticity
	Dependencies	-
	Type	REL
	Short name	Elasticity
	Actors	-
	Description	Implement the distributed data management service elasticity for both reducing and expanding the exploitation resource plane available by the application.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement
M	Plan and implement a verification process to check the correct elasticity of the in-memory state service.	
A	This requirement is achievable, as it is one of the main goals of task T4.4.	
R	The aim of the requirement is to design and implement a mechanism to monitor the in-memory state service and decide if it has to be scaled out or down to satisfy the SLOs	
T	D4.2 M30	

Req-4.5-CustomDockerImages

	Section	Description
S	ID	Req-4.5-CustomDockerImages
	Dependencies	Req-3.1-WorkflowDef, Req-3.3-ParallelContainerExecution
	Type	PORT: Adaptability
	Short name	Execution of custom docker images
	Actors	FaaS platform
	Description	FaaS platform could allow the execution of custom docker images from developers, with a specific structure, as a function.
	Additional Information	
	Priority (MoSCoW)	Should-have
M	This requirement will be identified as complete, if the FaaS platform can accept a custom docker image, that is compliant with a specific spec, and execute it as a function no matter the runtime.	
A	Many FaaS platforms(such as Openwhisk or OpenFaaS) support the usage of custom Docker images as an action runtime to handle the issue of having external application dependencies too large to deploy.	
R	The goal of this requirement is to allow application developers to freely develop their applications without restricting them with supported runtimes. Furthermore, it eases existing application migration to the FaaS paradigm for components that cannot be easily ported to a function logic.	
T	This requirement should be taken under consideration for the FaaS platform selection in Phase 1 of the project (M13 Prototype-M15 integration).	

Req-4.5-FaaSandIaaSMonitoring

	Section	Description
S	ID	Req-4.5-FaaSandIaaSMonitoring
	Dependencies	Req-4.2-FaaSBenchmarking
	Type	DATA: Data Requirements
	Short name	Benchmark functions monitoring data availability
	Actors	FaaS Platform, Orchestration Platform
	Description	The FaaS platform as well as the Container Orchestration platform need to expose collected monitoring metrics from benchmark functions execution.
	Additional Information	
	Priority (MoSCoW)	Must-have
M	FaaS Metrics: Function invocations, execution time, memory size, cold/warm start case Container Metrics: I/O wait, cache hits, user time, memory size etc. The aforementioned metrics should be made available based on REST endpoints from which they can be retrieved.	
A	The majority of available platforms come with built-in mechanisms for collecting the needed metrics. Thus, the requirement is deemed as achievable, if according plugins or interfaces are used and exposed towards the benchmarking mechanisms.	
R	The goal of this requirement is to evaluate the performance of a FaaS platform provider and aid in the selection processes. It can also be used as a mean to evaluate resource management practices by the provider, and it is related to Objective - 3: "*Workflow distribution, functional incorporation and runtime management across the continuum (optimization, placement and reconfiguration)*" and Objective - 4: "*Provider-Local, fine grained runtime management and adaptation through extension of relevant provider interfaces*" of the project.	
T	Addressed	

Req-4.5-FaaSRuntimeAdaptation

	Section	Description
S	ID	Req-4.5-FaaSRuntimeAdaptation
	Dependencies	-
	Type	FUNC: Functional Appropriateness
	Short name	FaaS Platform runtime reconfiguration
	Actors	FaaS Platform
	Description	The selected FaaS platform needs to have means of setting dynamically a number of parameters with relation to the operation of the platform, especially with relation to the host cluster upon which it operates.
	Additional Information	These parameters can be investigated by according mechanisms in the context of T4.2 or 5.4 in order to optimize platform operation
	Priority (MoSCoW)	Should-have
M	Parameters in question may be the cluster size, , autoscaling factors linked with application or platform related metrics etc.	

A	The requirement is achievable, provided a relevant selection of the baseline FaaS platform takes this characteristic under consideration.
R	The goal of this requirement is to enable mechanisms created in the context of various tasks (e.g., T4.2, T4.5, T5.2, T5.4) to set parameters accordingly in order to optimize various aspects of the platform operation and related outcomes (e.g., increase of number of hot/warm starts, reduction of wait time etc.). It is related to Objective - 3: "*Workflow distribution, functional incorporation and runtime management across the continuum (optimization, placement and reconfiguration)*" and Objective - 4: "*Provider-Local, fine grained runtime management and adaptation through extension of relevant provider interfaces*" of the project.
T	Partially addressed.

Req-4.5-placementDecision

	Section	**Description**
S	ID	Req-4.5-placementDecision
	Dependencies	
	Type	FUNC
	Short name	Allow placement decisions of functions at a physical node level
	Actors	-
	Description	Design and implement an automated placement decision maker of virtual resources at the physical node level and reducing the noisy neighbour effect.
	Additional Information	Due to the relationship with T5.2 and T5.3 there might be dependencies from these requirements.
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Validate that the placement decision component decision provides better performance	
A	This requirement is achievable, as it is one of the main goals of task T5.4.	
R	The aim of this requirement is to achieve an automated placement decision system taking into account the AI models available at T5.2 and implemented at T5.3	
T	D5.2 M30	

3.2.5 Use Case Requirements

Req-6.1-stateless

	Section	**Description**
S	ID	Req-6.1-stateless
	Dependencies	TBD
	Type	PORT
	Short name	Stateless Function
	Actors	TBD
	Description	Each function must be stateless.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	During the design phase is necessary to control that each developed function is not dependent on any other one	

A	This requirement is achievable, but it must be taken into account from the beginning of the development
R	This requirement aims to satisfy the first principle of FAAS. If a function is designed in a way that can hold state, it is using the wrong architecture.
T	This requirement needs to be active through out the lifecycle of the project.

Req-6.1-single action

	Section	**Description**
S	ID	Req-6.1-single action
	Dependencies	Req-6.1-stateless
	Type	COMP
	Short name	Each function provides a single action
	Actors	TDB
	Description	Each function has to perform one and only one action. The microservices based approach for creating an application requires the refactoring of the application into a collection of modular microservices easier to develop and deploy. In this vision, the microservice performs only one action, defined as a function.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Plan to create a function in order that a single request yields a single response	
A	Given the complexity of the PHYSICS architecture, this requirement is achievable, but care must be taken on each step so that the single function is incorporated by-design.	
R	This is a corollary of stateless	
T	This requirement needs to be active throughout the lifecycle of the project	

Req-6.1-lightweight

	Section	**Description**
S	ID	Req-6.1-lightweight
	Dependencies	Req-6.1-single action
	Type	PERF
	Short name	Each function is lightweight.
	Actors	TBD
	Description	A very important aspect of FAAS is the load time that must be as short as possible.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	It is relevant to use as few libraries as possible while writing a function: this fact will carry out the use of less memory.	
A	Using simple and lightweight functions will reduce the complexity of the PHYSICS architecture. This requirement is achievable but must be taken in account from the beginning of each step of the development	
R	Keep it simple	

T	This requirement needs to be active through out the lifecycle of the project

Req-6.1- OpenAPI

	Section	**Description**
S	ID	Req-6.1-OpenAPI
	Dependencies	TBD
	Type	COMP/USE
	Short name	Function exposes an OpenAPI.
	Actors	TBD
	Description	The Function interoperability can be addressed using the OpenAPI (e.g., through Swagger[4]) approach in order expose its functionality to other functions.
	Additional Information	-
	Priority (MoSCoW)	C: Could have
M	If all the functions that are designed and used to realize the PHYSICS-enabled application workflow have a standard communication interface (such as OpenAPI), the application workflow design will be easier	
A	The standardization of the interoperability can be achieved if all functions uses a standard pattern defined in the PHYSICS architecture, but this requires having a clear vision of all use-cases because some of them may not use a standard OpenAPI for their needs	
R	This approach is relevant for the interoperability and portability of the written functions	
T	This requirement needs to be active throughout the lifecycle of the project	

Req-6.1- Centralized logging system

	Section	**Description**
S	ID	"Req-6.1- Centralized logging system"
	Dependencies	Access log
	Type	MAINT/REL
	Short name	Centralized logging system
	Actors	FAAS platform
	Description	The solution must provide a way to centralize all logs from all disturbed components in any location. This system should be specific and lightweight
	Additional Information	-
	Priority (MoSCoW)	M: Must have
M	If this system is in place, troubleshooting for the entire platform will be easier and faster	

[4] https://swagger.io/

A	All components within PHYSIC must conform to twelve-factor[5] concepts, so the logs must be redirected to stdout. This makes it very easy to take these logs and redirect them to a centralized system
R	This approach is relevant for the troubleshooting
T	This requirement needs to be active throughout the lifecycle of the project

Req-6.1- Backup/Restore platform

	Section	**Description**
S	ID	"Req-6.1- Backup/Restore platform"
	Dependencies	TBD
	Type	MAINT
	Short name	Backup system
	Actors	FAAS platform
	Description	The platform should implement a disaster recovery system that allows to restore the correct functionality of the platform in case of a catastrophic event
	Additional Information	-
	Priority (MoSCoW)	C: Could have
M	The backup system is crucial to recovery from unpredictable disaster events. In order to evaluate how it is useful is important to define the RTO[6] (Recovery Time Objectives) and RPO[7] (Recovery Point Objectives) values.	
A	Backup definition and planning for critical components	
R	It is relevant to the resilience of the platform	
T	This requirement needs to be active throughout the lifecycle of the project	

Req-6.1-Login functionality

	Section	**Description**
S	ID	"Req-6.1-Login functionality"
	Dependencies	TBD
	Type	SEC
	Short name	DE Legin
	Actors	DE

[5] https://12factor.net/logs

[6] The maximum allowable time to recovery after data loss. If the recovery time objective is five hours, then it must be possible to restore data up to the recovery point objective within five hours.

[7] The maximum allowable data loss as a point in time. If the recovery point objective is two hours, then the maximum allowable amount of data loss that is acceptable is two hours of work.

	Description	The design environment should implement a login functionality to track the user activity and manage the function authorization, like the build branch that can be used
	Additional Information	-
	Priority (MoSCoW)	C: Could have
M	The number of log with a indicated user	
A	For the base idea to give at the user a portal for creating is own flow we need to protect its job with the login functionality	
R	It is relevant for manage the distribution of the design enviroment to a multi users	
T	The login functionality is a basic step for future integration in the DE, we think that the next milestone MS 11 - 31/3/23 is the more appropriate for the integration of this requirement	

Req-6.2-Health

	Section	**Description**
S	ID	Req-6.2-Health
	Dependencies	-
	Type	REL
	Short name	Health Check
	Actors	Function, Load Balancer, Environment
	Description	The architecture or orchestrator must be able to check the health of the application or environment
	Additional Information	
	Priority (MoSCoW)	Mandatory requirement
M	If a function/application/environment doesn’t pass the health check the PHYSICS architecture must notice this.	
A	Should be a core element in the PHYSICS architecture. Therefore, its mandatory to achieve the goals of PHYSICS	
R	For use-case Smart Manufacturing, its required to redeploy functions, if the current one isn’t healthy anymore	
T	Not required to provide this at the beginning, but should be achieved in the mid of the project	

Req-6.2-Load

	Section	**Description**
S	ID	Req-6.2-Load
	Dependencies	-
	Type	PERF
	Short name	Load Balancing
	Actors	FaaS, Environment, Orchestrator
	Description	Balancing the load to multiple functions (edge/cloud) if required to fulfill the performance
	Additional Information	

	Priority (MoSCoW)	Should-have. Desirable requirement.
M	If the CPU-load of an environment is >80%. The PHYSICS architecture must be able to deploy a new function in a different environment and balance the load.	
A	Might be difficult, depending on the frameworks to do this between cloud and edge environments	
R	Smart Manufacturing: E.g., increasing the production rate and still using the same application e.g., for prediction of machine failures or visual inspection might result in too high load of the FaaS	
T	Should be achieved until beginning of the implementation of the use-case.	

Req-6.2-Privacy

	Section	**Description**
S	ID	Req-6.2-Privacy
	Dependencies	-
	Type	SEC
	Short name	All data must be secure and private
	Actors	Load-Balancer, Orchestrator, Edge or Cloud Server (Deployment Location)
	Description	The handling of data must be secured, encrypted in all situations.
	Additional Information	-
	Priority (MoSCoW)	M: Must-have. Mandatory requirement.
M	Encryption in transit and rest at all time Current state of the art configuration in the deployed environment	
A	Is archivable if configuration of environment is done correct	
R	Highly relevant as data from manufacturing are highly sensible.	
T	This requirement needs to be active throughout the lifecycle of the project.	

Req-6.3-AccessLog

	Section	**Description**
S	ID	Req-6.3-AccessLog
	Dependencies	Req-6.3-AuthorizedAccess
	Type	SEC-Non-repudiation
	Short name	Access Log
	Actors	-
	Description	The system must have a log in which every logon is recorded, failed ones also.
	Additional Information	This is an internal regulatory requirement (REG-005), based on 21 CFR Part 11, Section 11.300 (d), and EU Annex 11, Section 8.
	Priority (MoSCoW)	Must-have
M	A series of tests should be designed to perform successful and unsuccessful login attempts throughout various endpoints of the platform – all attempts should be traceable to the log.	
A	This requirement is achievable.	
R	See 21 CFR Part 11, Section 11.300(d).	
T	To be determined	

Req-6.3-AlteredRecords

	Section	Description
S	ID	Req-6.3-AlteredRecords
	Dependencies	-
	Type	SEC
	Short name	Find Altered Records
	Actors	-
	Description	The system shall be able to find all invalid and altered records (changed by external applications or tools).
	Additional Information	This is an internal regulatory requirement (REG-002), based on 21 CFR Part 11, Section 11.10 (a).
	Priority (MoSCoW)	Must-have
M	There should be a complete list of all types of data that fall under this regulation. This list of data types should be mapped onto database records. Several test cases should be defined for altering those relevant data types through internal/external means. All alterations should be able to be retraced.	
A	This requirement is achievable.	
R	This requirement relates to privacy and security of personal (health) data. It is important to know who modified critical health information and when.	
T		

Req-6.3-AuditTrailExport

	Section	Description
S	ID	Req-6.3-AuditTrailExport
	Dependencies	Req-6.3-AuditTrailLogs
	Type	FUNC
	Short name	Audit Trail Export
	Actors	Investigators, Organisation-Administrators
	Description	The system must be able to provide audit trail copies in a standard electronic format (XML, PDF).
	Additional Information	This is an internal regulatory requirement (REG-024), based on 21 CFR Part 11, Section 11.10 (e), and EU Annex 11, Section 10.
	Priority (MoSCoW)	Must-have
M	Authorized users of the web portal (Investigators, Organisation-Administrators) shall be able to execute an export function that exports all relevant information related to their currently assigned role (e.g., an Investigator within a specific Study shall be able to export all audit log records related to that specific Study only).	
A	This requirement is achievable.	
R	See 21 CFR Part 11, Section 11.10 (e).	
T	To be determined	

Req-6.3-AuditTrailLogs

	Section	Description
S	ID	Req-6.3-AuditTrailLogs

	Dependencies	Req-6.3-AlteredRecords Req-6.3-AuthorizedAccess
	Type	SEC-Accountability
	Short name	Audit Trail Logs
	Actors	All
	Description	The system must provide automated audit trail logs. Inside them all the information related to record creation, modification and deletion must be recorded. For each operation the system record, the following information shall be used: Username of the user Date and time stamps Type of action on records Record values before and after the action was applied by the user
	Additional Information	This is an internal regulatory requirement (REG-020, REG-021), based on 21 CFR Part 11, Section 11.10 (e), and EU Annex 11, Section 10.
	Priority (MoSCoW)	Must-have
M	A series of tests should be designed that execute a number of information altering steps through various paths in the system – all actions taken in these test routes should be traceable through the Audit Log.	
A	This requirement is achievable.	
R	See 21 CFR Part 11, Section 11.10 (e).	
T	To be determined	

Req-6.3-AuthorizedAccess

	Section	**Description**
S	ID	Req-6.3-AuthorizedAccess
	Dependencies	-
	Type	SEC-Confidentiality
	Short name	Authorized Access
	Actors	All
	Description	The system will be accessed only by authorized users providing a username and password, or authorized external services.
	Additional Information	This is an internal regulatory requirement (REG-005), based on 21 CFR Part 11, Section 11.10 (d), and EU Annex 11, Sections 13 and 14.
	Priority (MoSCoW)	Must-have
M	All platform endpoints must have the required authentication mechanisms in place.	
A	This requirement is achievable.	
R	The system stores personal and health-related data, so no unauthorized access should be able to take place.	
T	To be determined	

Req-6.3-GenerateRecords

	Section	**Description**
S	ID	Req-6.3-GenerateRecords

	Dependencies	-
	Type	FUNC
	Short name	Generate Records
	Actors	End-User, Investigator
	Description	The system shall be able to generate full copies of all stored electronic records. All copies must be generated both in a readable form and in a standard electronic format (PDF). All copies must be available for inspection checking.
	Additional Information	This is an internal regulatory requirement (REG-003), based on 21 CFR Part 11, Section 11.10 (b), 11.10 (c), and EU Annex 11, Section 12.
	Priority (MoSCoW)	Must-have
M	Primary End-Users as well as Investigators shall be able to request generation of all stored electronic records (i.e., relevant personal and health information). These records shall be delivered in PDF format.	
A	This requirement is achievable.	
R	For the PHYSICS use case, the primary reason for this requirement is the end-user's GDPR right to obtain all information stored about him/her.	
T	To be determined	

Req-6.3-TimeSynchronisation

	Section	**Description**
S	ID	Req-6.3-TimeSynchronisation
	Dependencies	-
	Type	PORT-Adaptability
	Short name	Time Synchronisation
	Actors	-
	Description	Every component involved in the system must be time synchronized.
	Additional Information	This is an internal regulatory requirement (REG-026), based on 21 CFR Part 11, Section 11.10 (e), and EU Annex 11, Section 10.
	Priority (MoSCoW)	Must-have
M	All events that fall under the audit trail requirement (Req-6.3-AuditTrailLogs) must be timestamped. All timestamps originating from potentially distributed service components must be synchronised with a degree of accuracy that ensures that the order of audit log events is maintained.	
A	To be determined	
R	See 21 CFR Part 11, Section 11.10 (e).	
T	To be determined	

3.3 Requirements Traceability Matrix

The requirements traceability matrix provides a high-level description of the defined requirements, including:

- Status: dictates whether a **new** requirement has been added, an existing requirement has been **updated** or an existing requirement maintained its properties **as-is**.
- MoSCoW: any changes that may have taken place regarding the prioritization of the requirements.

- Evaluation: This column describes the completion status of a requirement. More specifically:
  - Not continued: The requirement is no longer considered necessary.
  - Pending: The processes that must be completed in order to fulfil the requirement are in progress.
  - Addressed: The requirement has been met, but there is room for more development that will potentially contribute further in the completion of this requirement.
  - To be extended: The scope of the requirement will be expanded.
  - Completed: The requirement has already been met.

The matrix is presented through 5 tables, one for each corresponding category.

CLOUD ENVIRONMENT REQUIREMENTS			
	STATUS	**MoSCoW**	**EVALUATION**
Req-3.1-WorkflowDef	AS-IS	Must have	Addressed through the Node-RED flow orchestration mode, pending
Req-3.1-SupportedRuntimes	AS-IS	Should have	Addressed through the incorporation of a generic runtime, pending to check if other runtimes will be needed in the course of the project
Req-3.1-UploadCustomImages	AS-IS	Won't have	Not continued due to the Custom Dockerfile feature of the Design Environment, from which the developer can create their own base custom image. Hence this requirement has been deemed obsolete
Req-3.1-Multi-Tenancy	NEW	Must have	Completed based on the branch separation
Req-3.1-LogsService	NEW	Must have	Addressed, to be extended with log info from any needed component, including platform services
Req-3.1-BuildsHistory	NEW	Should have	Completed and available in the UI
Req-3.3-PatternDocumentation	UPDATED	Must have	Addressed up to now through documentation and examples included in the provided Node-RED subflows, monitoring for future period to ensure that any new patterns will also come with documentation
Req-3.3-PatternApplication	AS-IS	Must have	Addressed since all patterns are provided as Node-RED subflows, monitoring in the next period for the newly produced patterns
Req-3.3-ParallelContainerExecution	AS-IS	Could have	Completed through the SplitJoin and SplitJoin Multiple patterns

Req-4.4-funcNode	AS-IS	Must have	Pending

SEMANTICS REQUIREMENTS			
	STATUS	MoSCoW	EVALUATION
Req-3.2-WorkflowCoverage	AS-IS	Must have	Completed, through the incorporation of all main workflow features in the Application ontology and the inclusion of the Wfdesc external ontology
Req-3.2-RequirementsCoverage	AS-IS	Must have	Addressed for the requirements up to now, method to easily extend annotations has also been devised to cater for new additions needed in the following period
Req-3.2-ConstraintsCoverage	AS-IS	Must have	Similar to the above
Req-3.2-LinkWithVocabularies	AS-IS	Should have	Addressed through the inclusion of the Wfdesc ontology and the irao ontology The following period a link to the concepts of the resource ontology from T5.1 is the goal.
Req-3.2-ReasoningCapability	AS-IS	Must have	Pending
Req-3.2-ExpressivityRichness	AS-IS	Should have	Addressed. Currently, the resulting AR is 0.85 while the RR is 0.42. The descriptive logic complexity of the ontology is SROIQ (D), the maximum of OWL2. We will also monitor these metrics in the evolution of the ontology
Req-4.1-Adaptation	AS-IS	Must have	Pending. Implementation started during the 2nd period of the project.
Req 4.1 - Inputs	UPDATED	Must have	Addressed. The update in this requirement refers to an additional data input from WP5 with deployment information.
Req 4.1 - Latency	AS-IS	Should have	Addressed up to now, monitoring for future period. Evaluation based on the response time of the Reasoning Framework during parallel requests.
Req 4.1 - ML_Reasoning	UPDATED	Could have	Pending. This requirement is not needed, as semantic reasoning with rules is sufficient for the platform needs.
Req 4.1 - Reasoning	AS-IS	Must have	Addressed. Semantic rules, ontologies and knowledge graph are integrated in a single service facilitating resource filtering.

Req 5.1 - ResKnow	UPDATED	Must have	Addressed up to now. The component extracts necessary information from the Kubernetes API. Additional information to be gathered such as energy specifications in the future.
Req 5.1 - Interface	AS-IS	Could have	Completed. Resource Semantics can be reached visually in the task's 5.1 interface.
Req 5.1 - Compatibility	AS-IS	Must have	Addressed by enabling communication with the reasoning framework. Further adjustments will be implemented to digest information from other components.
Req 5.1 Portability	AS-IS	Should have	Addressed, the component has been tested on testbeds deployed on two different cloud providers. Future testing needs to take place to ensure compatibility with edge clusters.
Req 5.1 - GraphSeparability	NEW	Must have	Completed. The graphs can easily be separated by the unique identifier of each, which is assigned to the cluster entity of the ontology.
Req-5.1-SemCap	AS-IS	Must have	Completed, the resource ontology is fully fledged and encompasses the necessary information for the operation of reasoning framework and in turn deployment optimization.

FAAS / CONTAINER PLATFORM REQUIREMENTS			
	STATUS	**MoSCoW**	**EVALUATION**
Req-4.4-state	AS-IS	Must have	Initial version delivered at M13. It will be updated at M30
Req-4.4-interplay	AS-IS	Must have	Initial version delivered at M13. It may be updated at M30
Req-4.4-tradeOffs	AS-IS	Must have	Pending
Req-4.4-perf	AS-IS	Must have	Pending
Req-4.4-access	AS-IS	Must have	Initial version delivered at M13. It will be updated at M30
Req-4.5-PersStorage	AS-IS	Could have	Pending
Req-5.4-optimization	AS-IS	Must have	Pending
Req-5.4-co-allocation	AS-IS	Must have	Initial version delivered at

			M13. It will be updated at M30
Req-5.4-workloads	AS-IS	Must have	Pending
Req-5.4-AImodels	AS-IS	Must have	Pending
Req-5.4-validation	AS-IS	Must have	Pending
Req-5.4-UsageModellingOpt	AS-IS	Should have	Addressed in the context of the Request Aggregator model presenting MAPE of ~11%, below the threshold of 20% set in the requirement metric. To be extended to cover for other cases of interest (e.g. configuration of the SJ pattern).

CROSS LAYER REQUIREMENTS			
	STATUS	MoSCoW	EVALUATION
Req-3.4-Privacy	AS-IS	Must have	Addressed
Req-3.4-CodeAnalysis	AS-IS	Could have	Pending
Req-3.4-DeprMan	AS-IS	Could have	Pending
Req-3.4-Encryption	AS-IS	Must have	Addressed
Req-3.4-Secrets	AS-IS	Must have	Addressed, to be extended
Req-3.4-SecureComms	AS-IS	Must have	Addressed, to be extended
Req-4.2-FaaSBenchmarking	UPDATED	Must have	Completed through the implementation of the PHYSICS load generator function. Any target function can be directly benchmarked and measured through a function invocation of the Generator (benchmarking as a service). The update relates to the fact that any user function and not just a pre-existing benchmark function can be used.
Req-4.2-CostAssociation	AS-IS	Should have	Pending. Cost association depends on the execution time, which was a dependency from Req-4.2-FaaSBenchmarking. The latter has been completed

			so in the following period this can be addressed.
Req-4.2-MeasurementPropagation	UPDATED	Must have	Partially addressed, the PHYSICS load generator results should be pushed to the respective mechanisms (e.g. RF) and populate directly the performance semantics. The update refers to the inclusion of a way to push the data instead of pulling them. Statistics include the metrics foreseen (averages, deviation etc)
Req-4.4-elasticity	AS-IS	Must have	Pending
Req-4.5-CustomDockerImages	AS-IS	Should have	Pending
Req-4.7-FaaSandIaaSMonitoring	AS-IS	Must have	Addressed with the usage of Prometheus with Openwhisk in the context of WP5, and the statistics of the PHYSICS Load Generator, including cold starts, wait times, init times,duration, latency etc.
Req-4.7-FaaSRuntimeAdaptation	UPDATED	Should have	Partially addressed. K8S autoscalers have been applied on the cluster, pending link between cluster or app metric and elasticity decisions. The update refers to the type of elasticity targeted (cluster versus OW parameters, since the latter need restart of the OW setup)
Req-4.5-placementDecision	AS-IS	Must have	Pending
Req-3.4-SmartContracts	NEW	Should have	Addressed, to be extended
Req-3.4-SmartContractTemplates	NEW	Could have	Addressed, to be extended

USE CASE REQUIREMENTS			
	STATUS	MoSCoW	EVALUATION
Req-6.1-stateless	AS-IS	Must have	In fact, addressed FAAs created by Openwhisk are stateless by default
Req-6.1-single action	AS-IS	Must have	The function created always has an action, but in some cases it is necessary to

			combine multiple functions to create a complex action
Req-6.1-lightweight	AS-IS	Must have	The maximum code size for the action is 48MB
Req-6.1-OpenAPI	AS-IS	Could have	Pending. We have planned to implement a new API to extend an OpenWhisk functionality, but it has not yet been implemented. Anyway we will follow the OpenAPI methodology
Req-6.1- Centralized logging system	NEW	Must have	To be evaluated at the end of the 2nd implementation cycle
Req-6.1- Backup/Restore platform	NEW	Could have	To be evaluated at the end of the 2nd implementation cycle
Req-6.1-Login functionality	NEW	Could have	To be evaluated at the end of the 2nd implementation cycle
Req-6.2-Health	AS-IS	Must have	Pending
Req-6.2-Load	AS-IS	Should have	Pending
Req-6.2-Privacy	AS-IS	Must have	Pending
Req-6.3-AccessLog	AS-IS	Must have	Pending
Req-6.3-AlteredRecords	AS-IS	Must have	Pending
Req-6.3-AuditTrailExport	AS-IS	Must have	Pending
Req-6.3-AuditTrailLogs	AS-IS	Must have	Pending
Req-6.3-AuthorizedAccess	AS-IS	Must have	Pending
Req-6.3-GenerateRecords	AS-IS	Must have	Pending
Req-6.3-TimeSynchronisation	AS-IS	Must have	Pending

4 REQUIREMENTS DISCUSSION

All in all, the PHYSICS consortium has gathered 72 (previously 63 in D2.2) requirements, with a breakdown of: 10 requirements for the cloud design environment, 17 requirements for the semantic framework, 12 requirements for the FaaS and container platform, 16 requirements that are cross-layer and 17 requirements for use cases. Furthermore, the number of requirements per category is the following:

- FUNC – Functional Suitability Requirements: 17
- USE – Usability Requirements: 10
- SEC – Security Requirements: 14
- DATA – Data Requirements: 9
- PERF – Performance Efficiency Requirements: 10
- PORT – Portability Requirements: 5
- COMP – Compatibility Requirements: 3
- REL – Reliability Requirements: 2
- MAINT - Maintenance Requirements: 2

The evolution of the existing requirements and the addition of new ones, compared to the D2.2, are an indication of the level of understanding of the project by the consortium. With this deliverable, we finalize the requirements definition and lay strong foundations upon which the implementation and the corresponding results will come to fruition.

5 CONCLUSIONS

This deliverable extends D2.2, providing a complete version regarding the *updated* documentation of the state-of-the-art analysis of certain technological axes of the PHYSICS project alongside the *update of existing and introduction of new* functional and non-functional requirements, as they are envisioned by the consortium of PHYSICS. In the state-of-the-art analysis, all major architectural components of PHYSICS are analysed with major, well-known as well as novel technological solutions and research works. Through this documentation, the PHYSICS project will be able to continue its course without any issues given the fact that the foundational technologies that it will be based on is well understood and can be found in one place. Furthermore, there is an analysis for the state of the art of security in the context of FaaS, a component that although is not an architectural component of PHYSICS but it is an overlay-component that covers each aspect and axis of PHYSICS since security and privacy is crucial not only for legal and regulation duties but also for the ethical values and responsibilities to end users.

Furthermore, in the requirements elicitation section of this deliverable, there is the depiction of the process behind the aggregation of requirements, a specification of each requirement and a quick overview of these requirements. The methodology used to gather the requirements was based on the vision of each partner of the PHYSICS project so that everyone has a saying for the development of the PHYSICS project at an early stage (M4) so as to avoid any issues later in the project. The tracking and management of the requirements is based on well-established standards, best practices and methodologies (S.M.A.R.T., ISO 25010 and MoSCoW), something that will ensure that these requirements are well-established, relevant to the project, achievable, measurable, timely, uniformly documented and trackable. Finally, the overview of these requirements is acceptable and will allow the project to move forward on a strong basis.

This deliverable had the purpose of creating a strong foundational know-how and knowledge on both the technologies and the requirements of the project. Given the presented thorough analysis of both of these concepts in the context of the PHYSICS project, we aimed to ensure that the project is based on a strong knowledge and understanding of the related technologies and the project as a whole. With this information in hand, the following tasks and deliverables, especially the ones in work package 2 and 6, will be able to proceed with their operations in a smooth and unobstructed manner, which will in turn build step-by-step the envisioned PHYSICS architecture.

REFERENCES

[1] "Home," *Apache Airflow*. / (accessed Apr. 21, 2021).

[2] *fission/fission-workflows*. Fission, 2021.

[3] "Node-RED." https://nodered.org/ (accessed Apr. 21, 2021).

[4] "Taverna - Apache Incubator." https://incubator.apache.org/projects/taverna.html (accessed Apr. 21, 2021).

[5] "Workflow and Decision Automation Platform | Camunda." https://camunda.com/ (accessed Apr. 21, 2021).

[6] admin, "Workflow Automation Software & Business Process Management," *Cflow*. https://www.cflowapps.com/ (accessed Apr. 21, 2021).

[7] "KNIME | Open for Innovation." https://www.knime.com/ (accessed Apr. 21, 2021).

[8] "JSON-LD - JSON for Linking Data." https://json-ld.org/ (accessed Apr. 02, 2021).

[9] G. Antoniou and F. Van Harmelen, "Web ontology language: Owl," in *Handbook on ontologies*, Springer, 2004, pp. 67–92.

[10] "Ontology for Cloud Computing Instances," *Ontology for Cloud Computing Instances*. http://cookingbigdata.com/linkeddata/ccinstances/.

[11] "Ontology for Service Level Agreement for Cloud Computing," *Ontology for Service Level Agreement for Cloud Computing*. http://cookingbigdata.com/linkeddata/ccsla/.

[12] "GeoJSON-LD." https://geojson.org/geojson-ld/ (accessed Apr. 24, 2020).

[13] "Vocabulary for Regions and Zones on Cloud Computing," *Vocabulary for Regions and Zones on Cloud Computing*. http://cookingbigdata.com/linkeddata/ccregions/.

[14] G. Cretella and B. Di Martino, "A semantic engine for porting applications to the cloud and among clouds: A SEMANTIC ENGINE FOR PORTING APPLICATIONS TO THE CLOUD AND AMONG CLOUDS," *Softw. Pract. Exp.*, vol. 45, no. 12, pp. 1619–1637, Dec. 2015, doi: 10.1002/spe.2304.

[15] A. V. Dastjerdi, S. K. Garg, O. F. Rana, and R. Buyya, "CloudPick: a framework for QoS-aware and ontology-based service deployment across clouds: CloudPick: a framework for QoS-aware and ontology-based service deployment across clouds," *Softw. Pract. Exp.*, vol. 45, no. 2, pp. 197–231, Feb. 2015, doi: 10.1002/spe.2288.

[16] S. Ghazouani, H. Mezni, and Y. Slimani, "Bringing semantics to multicloud service compositions," *Softw. Pract. Exp.*, vol. 50, no. 4, pp. 447–469, Apr. 2020, doi: 10.1002/spe.2789.

[17] "Web of Things (WoT) Architecture." https://www.w3.org/TR/wot-architecture/ (accessed Apr. 22, 2020).

[18] "Web of Things (WoT) Thing Description." https://www.w3.org/TR/wot-thing-description/ (accessed Apr. 22, 2020).

[19] "Web of Things (WoT) Binding Templates." https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/ (accessed Apr. 22, 2020).

[20] "The Wfdesc ontology," *The Wfdesc ontology*. http://purl.org/wf4ever/wfdesc.

[21] "The Wfprov Ontology," *The Wfprov Ontology*. http://purl.org/wf4ever/wfprov.

[22] "Workflow Invocation Ontology," *Workflow Invocation Ontology*. http://purl.org/net/wf-invocation.

[23] "The Workflow Motif Ontology," *The Workflow Motif Ontology*. http://purl.org/net/wf-motifs.

[24] M. Sadowski and L. Frantzell, "Apache OpenWhisk–Open Source Project," in *Serverless Swift*, Springer, 2020, pp. 37–57.

[25] R. Pereira *et al.*, "Ranking programming languages by energy efficiency," *Sci. Comput. Program.*, vol. 205, p. 102609, 2021.

[26] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating MapReduce on Virtual Machines: The Hadoop Case," in *Cloud Computing*, Berlin, Heidelberg, 2009, pp. 519–528, doi: 10.1007/978-3-642-10665-1_47.

[27] "Ad Hoc Big Data Processing Made Simple with Serverless MapReduce," *Amazon Web Services*, Nov. 04, 2016. https://aws.amazon.com/blogs/compute/ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce/ (accessed Apr. 02, 2021).

[28] *d2si-oss/ooso*. D2SI-OSS, 2020.

[29] B. Congdon, *bcongdon/corral*. 2021.
[30] S. Pallickara, "Some Recent Advances in Utility and Cloud Computing," *Future Gener. Comput. Syst.*, vol. 56, no. C, pp. 315–316, Mar. 2016, doi: 10.1016/j.future.2015.11.018.
[31] D. S. Jegan, L. Wang, S. Bhagat, T. Ristenpart, and M. Swift, "Guarding Serverless Applications with SecLambda," *ArXiv Prepr. ArXiv201105322*, 2020.
[32] S. Hong, A. Srivastava, W. Shambrook, and T. Dumitraș, "Go serverless: Securing cloud via serverless design patterns," 2018.
[33] "AWS Well-Architected - Build secure, efficient cloud applications," *Amazon Web Services, Inc.* https://aws.amazon.com/architecture/well-architected/ (accessed Mar. 30, 2021).
[34] "Security by Design - Amazon Web Services (AWS)," *Amazon Web Services, Inc.* https://aws.amazon.com/compliance/security-by-design/ (accessed Mar. 30, 2021).
[35] "Patterns for scalable and resilient apps | Solutions," *Google Cloud.* https://cloud.google.com/solutions/scalable-and-resilient-apps (accessed Mar. 30, 2021).
[36] dragon119, "Security patterns - Cloud Design Patterns." https://docs.microsoft.com/en-us/azure/architecture/framework/security/security-patterns (accessed Mar. 30, 2021).
[37] D. Didone and R. J. de Queiroz, "Forensic as a service-FaaS," in *Proceedings of the Sixth International Conference on Forensic Computer Science (ICoFCS)*, 2011, pp. 202–210.
[38] S. Nanda and R. A. Hansen, "Forensics as a service: Three-tier architecture for cloud based forensic analysis," in *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2016, pp. 178–183.
[39] Y. Wen, X. Man, K. Le, and W. Shi, "Forensics-as-a-service (faas): computer forensic workflow management and processing using cloud," in *The Fifth International Conferences on Pervasive Patterns and Applications*, 2013, pp. 1–7.
[40] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A Systematic grey literature review," *J. Syst. Softw.*, vol. 146, pp. 215–232, 2018.
[41] A. Akbulut and H. G. Perros, "Performance analysis of microservice design patterns," *IEEE Internet Comput.*, vol. 23, no. 6, pp. 19–27, 2019.
[42] "Go Fast by Going Micro: Microservices Design Patterns You Should Know -," *Visual Studio Live!: Training Conferences and Events for Enterprise Microsoft .NET and Azure Developers.* https://vslive.com/blogs/news-and-tips/2018/02/go-fast-by-going-micro-microservices-design-patterns-you-should-know.aspx (accessed Apr. 02, 2021).
[43] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *International Conference on Service-Oriented Computing*, 2017, pp. 706–721.
[44] dragon119, "Cloud design patterns - Azure Architecture Center." https://docs.microsoft.com/en-us/azure/architecture/patterns/index-patterns (accessed Apr. 02, 2021).
[45] G. Kousiouris *et al.*, "Parametric design and performance analysis of a decoupled service-oriented prediction framework based on embedded numerical software," *IEEE Trans. Serv. Comput.*, vol. 6, no. 4, pp. 511–524, 2012.
[46] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, "Towards a serverless platform for edge ${$AI$}$," 2019.
[47] A. Christidis, S. Moschoyiannis, C.-H. Hsu, and R. Davies, "Enabling serverless deployment of large-scale ai workloads," *IEEE Access*, vol. 8, pp. 70150–70161, 2020.
[48] "Horizontal Pod Autoscaler," *Kubernetes.* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (accessed Apr. 20, 2021).
[49] "Specifying a Disruption Budget for your Application," *Kubernetes.* https://kubernetes.io/docs/tasks/run-application/configure-pdb/ (accessed Apr. 20, 2021).
[50] *twosigma/fastfreeze*. Two Sigma, 2021.
[51] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, 2012.
[52] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, "Semantics for the Internet of Things: early progress and back to the future," *Int. J. Semantic Web Inf. Syst. IJSWIS*, vol. 8, no. 1, pp. 1–21, 2012.

[53] A. Gyrard, C. Bonnet, and K. Boudaoud, “Enrich machine-to-machine data with semantic web technologies for cross-domain applications,” in *2014 IEEE world forum on internet of things (WF-IoT)*, 2014, pp. 559–564.
[54] P. R. Woznowski, E. L. Tonkin, and P. A. Flach, “Activities of daily living ontology for ubiquitous systems: Development and evaluation,” *Sensors*, vol. 18, no. 7, p. 2361, 2018.
[55] L. Chen, C. D. Nugent, and H. Wang, “A knowledge-driven approach to activity recognition in smart homes,” *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 6, pp. 961–974, 2011.
[56] M. Strohbach, L. A. Saavedra, P. Smirnov, and S. Legostaieva, “Smart home crawler: Towards a framework for semi-automatic IoT sensor integration,” in *2019 Global IoT Summit (GIoTS)*, 2019, pp. 1–6.
[57] “GraphDB Downloads and Resources.” https://graphdb.ontotext.com/ (accessed Apr. 01, 2021).
[58] “Blazegraph Database.” https://blazegraph.com/ (accessed Apr. 01, 2021).
[59] “The Enterprise Knowledge Graph Platform | Stardog.” https://www.stardog.com/ (accessed Apr. 01, 2021).
[60] “OpenLink Software: Virtuoso Homepage.” https://virtuoso.openlinksw.com/ (accessed Apr. 01, 2021).
[61] “JanusGraph.” https://janusgraph.org/ (accessed Apr. 01, 2021).
[62] “Apache Jena - Reasoners and rule engines: Jena inference support.” https://jena.apache.org/documentation/inference/ (accessed Apr. 01, 2021).
[63] “AnzoGraph® DB.” https://www.cambridgesemantics.com/anzograph/ (accessed Apr. 01, 2021).
[64] “AllegroGraph.” https://allegrograph.com/ (accessed Apr. 01, 2021).
[65] “Pellet - Semantic Web Standards.” https://www.w3.org/2001/sw/wiki/Pellet (accessed Apr. 01, 2021).
[66] “RacerPro.” https://franz.com/agraph/racer/ (accessed Apr. 01, 2021).
[67] “HermiT Reasoner: Home.” http://www.hermit-reasoner.com/ (accessed Apr. 01, 2021).
[68] “RIF4J.” http://rif4j.sourceforge.net/ (accessed Apr. 01, 2021).
[69] “OWL : FaCT++.” http://owl.man.ac.uk/factplusplus/ (accessed Apr. 01, 2021).
[70] M. Ebrahimi, M. K. Sarker, F. Bianchi, N. Xie, D. Doran, and P. Hitzler, “Reasoning over RDF knowledge bases using deep learning,” *ArXiv Prepr. ArXiv181104132*, 2018.
[71] M. Nickel, V. Tresp, and H.-P. Kriegel, “Factorizing yago: scalable machine learning for linked data,” in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 271–280.
[72] S. Riedel, L. Yao, A. McCallum, and B. M. Marlin, “Relation extraction with matrix factorization and universal schemas,” in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 74–84.
[73] R. Socher, D. Chen, C. D. Manning, and A. Ng, “Reasoning with neural tensor networks for knowledge base completion,” in *Advances in neural information processing systems*, 2013, pp. 926–934.
[74] K.-W. Chang, W. Yih, B. Yang, and C. Meek, “Typed tensor decomposition of knowledge bases for relation extraction,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1568–1579.
[75] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, “Embedding entities and relations for learning and inference in knowledge bases,” *ArXiv Prepr. ArXiv14126575*, 2014.
[76] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon, “Representing text for joint embedding of text and knowledge bases,” in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1499–1509.
[77] T. P. Trouillon and G. M. Bouchard, “Complex embeddings for simple link prediction,” Nov. 23, 2017.
[78] A. Neelakantan, B. Roth, and A. McCallum, “Compositional vector space models for knowledge base completion,” *ArXiv Prepr. ArXiv150406662*, 2015.
[79] B. Peng, Z. Lu, H. Li, and K.-F. Wong, “Towards neural network-based reasoning,” *ArXiv Prepr. ArXiv150805508*, 2015.
[80] R. Das, A. Neelakantan, D. Belanger, and A. McCallum, “Chains of reasoning over entities, relations, and text using recurrent neural networks,” *ArXiv Prepr. ArXiv160701426*, 2016.

[81] D. Weissenborn, “Separating answers from queries for neural reading comprehension,” *ArXiv Prepr. ArXiv160703316*, 2016.
[82] Y. Shen, P.-S. Huang, J. Gao, and W. Chen, “Reasonet: Learning to stop reading in machine comprehension,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1047–1055.
[83] W. Xiong, T. Hoang, and W. Y. Wang, “Deeppath: A reinforcement learning method for knowledge graph reasoning,” *ArXiv Prepr. ArXiv170706690*, 2017.
[84] R. Das, A. Godbole, S. Dhuliawala, M. Zaheer, and A. McCallum, “A Simple Approach to Case-Based Reasoning in Knowledge Bases,” *ArXiv Prepr. ArXiv200614198*, 2020.
[85] J. Kuhlenkamp and S. Werner, “Benchmarking FaaS platforms: Call for community participation,” in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 189–194.
[86] R. Pellegrini, I. Ivkic, and M. Tauber, “Function-as-a-Service Benchmarking Framework,” *ArXiv Prepr. ArXiv190511707*, 2019.
[87] R. Pellegrini, I. Ivkic, and M. Tauber, “Towards a security-aware benchmarking framework for function-as-a-service,” *ArXiv Prepr. ArXiv190507228*, 2019.
[88] N. Mahmoudi and H. Khazaei, “SimFaaS: A Performance Simulator for Serverless Computing Platforms,” *ArXiv Prepr. ArXiv210208904*, 2021.
[89] T. Back and V. Andrikopoulos, “Using a microbenchmark to compare function as a service solutions,” in *European Conference on Service-Oriented and Cloud Computing*, 2018, pp. 146–160.
[90] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, “A SPEC RG cloud group’s vision on the performance challenges of FaaS cloud architectures,” in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 21–24.
[91] E. van Eyk and A. Iosup, “Addressing performance challenges in serverless computing,” *Proc ICT OPEN*, 2018.
[92] R. Zeng, X. Hou, L. Zhang, C. Li, W. Zheng, and M. Guo, “Performance Optimization for Cloud Computing Systems in the Microservice Era: State-of-the-Art and Research Opportunities.”
[93] J. Scheuner and P. Leitner, “Function-as-a-Service performance evaluation: A multivocal literature review,” *J. Syst. Softw.*, vol. 170, p. 110708, 2020.
[94] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, “Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research,” *Internet Things*, p. 100273, 2020.
[95] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan, “A holistic evaluation of docker containers for interfering microservices,” in *2018 IEEE International Conference on Services Computing (SCC)*, 2018, pp. 33–40.
[96] A. P. Ferreira and R. Sinnott, “A performance evaluation of containers running on managed kubernetes services,” in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019, pp. 199–208.
[97] M. Schwarzkopf and P. Bailis, “Research for practice: cluster scheduling for datacenters,” *Commun. ACM*, vol. 61, no. 5, pp. 50–53, 2018.
[98] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade,” *Queue*, vol. 14, no. 1, pp. 70–93, 2016.
[99] “Production-Grade Container Orchestration,” *Kubernetes*. https://kubernetes.io/ (accessed Apr. 16, 2021).
[100] B. Hindman *et al.*, “Mesos: A platform for fine-grained resource sharing in the data center.,” in *NSDI*, 2011, vol. 11, no. 2011, pp. 22–22.
[101] “Introducing Container Runtime Interface (CRI) in Kubernetes,” *Kubernetes*, Dec. 19, 2016. https://kubernetes.io/blog/2016/12/Container-Runtime-Interface-Cri-In-Kubernetes/ (accessed Apr. 16, 2021).
[102] “Container Storage Interface (CSI) for Kubernetes GA,” *Kubernetes*, Jan. 15, 2019. https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/ (accessed Apr. 16, 2021).

[103] *containernetworking/cni*. CNI, 2021.
[104] "Device Plugins," *Kubernetes*. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/ (accessed Apr. 16, 2021).
[105] "kubernetes/community," *GitHub*. https://github.com/kubernetes/community (accessed Apr. 16, 2021).
[106] KubeEdge, "KubeEdge," *KubeEdge*. /en/ (accessed Apr. 16, 2021).
[107] *virtual-kubelet/virtual-kubelet*. virtual kubelet, 2021.
[108] "K3s: Lightweight Kubernetes." https://k3s.io/ (accessed Apr. 16, 2021).
[109] "MicroK8s - Zero-ops Kubernetes for developers, edge and IoT | MicroK8s," *microk8s.io*. http://microk8s.io (accessed Apr. 16, 2021).
[110] *kubernetes-sigs/kubefed*. Kubernetes SIGs, 2021.
[111]: : "Submariner k8s project documentation website." https://submariner.io/ (accessed Apr. 16, 2021).
[112] 15 Minute Read, "Deployment Models," *Istio*. /latest/docs/ops/deployment/deployment-models/ (accessed Apr. 16, 2021).
[113] R. B. Bohn, C. A. Lee, and M. Michel, "The NIST Cloud Federation Reference Architecture," Feb. 2020, Accessed: Apr. 16, 2021. [Online]. Available: https://www.nist.gov/publications/nist-cloud-federation-reference-architecture.
[114] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *J. Netw. Comput. Appl.*, vol. 103, pp. 1–17, 2018.
[115] H. Yuan, "Energy and performance-optimized scheduling of tasks in distributed cloud and edge computing systems," 2020.
[116] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, and M. F. de Castro, "Optimized placement of scalable iot services in edge computing," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 189–197.
[117] S. Wang, Y. Li, S. Pang, Q. Lu, S. Wang, and J. Zhao, "A Task Scheduling Strategy in Edge-Cloud Collaborative Scenario Based on Deadline," *Sci. Program.*, vol. 2020, p. e3967847, Mar. 2020, doi: 10.1155/2020/3967847.
[118] E. Jonas *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *ArXiv Prepr. ArXiv190203383*, 2019.
[119] A. Das, A. Leaf, C. A. Varela, and S. Patterson, "Skedulix: Hybrid Cloud Scheduling for Cost-Efficient Execution of Serverless Applications," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, Oct. 2020, pp. 609–618, doi: 10.1109/CLOUD49709.2020.00090.
[120] K. Rzadca *et al.*, "Autopilot: workload autoscaling at Google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, New York, NY, USA, Apr. 2020, pp. 1–16, doi: 10.1145/3342195.3387524.
[121] A. A. Da Silva *et al.*, "Evaluating Computation and Data Placements in Edge Infrastructures through a Common Simulator," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 67–74.
[122] A. Anderson da Silva, "Investigating Job Allocation Policies in Edge Computing Platforms," Master's Thesis, Grenoble University.
[123] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *13th ${$USENIX$}$ Symposium on Operating Systems Design and Implementation (${$OSDI$}$ 18)*, 2018, pp. 427–444.
[124] "The Apache Crail (Incubating) Project: Overview." https://crail.apache.org/ (accessed Apr. 01, 2021).
[125] "Redis." https://redis.io/ (accessed Apr. 01, 2021).
[126] D. Barcelona-Pons, M. Sánchez-Artigas, G. París, P. Sutra, and P. García-López, "On the faas track: Building stateful distributed applications with serverless architectures," in *Proceedings of the 20th International Middleware Conference*, 2019, pp. 41–54.
[127] "Infinispan." https://infinispan.org/ (accessed Apr. 01, 2021).

[128] A. Wang *et al.*, “Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache,” in *18th ${$USENIX$}$ Conference on File and Storage Technologies (${$FAST$}$ 20)*, 2020, pp. 267–281.
[129] “Amazon ElastiCache- In-memory data store and cache,” *Amazon Web Services, Inc.* https://aws.amazon.com/elasticache/ (accessed Apr. 01, 2021).
[130] V. Sreekanti *et al.*, “Cloudburst: Stateful functions-as-a-service,” *ArXiv Prepr. ArXiv200104592*, 2020.
[131] C. Wu, J. Faleiro, Y. Lin, and J. Hellerstein, “Anna: A kvs for any scale,” *IEEE Trans. Knowl. Data Eng.*, 2019.
[132] “Apache OpenWhisk is a serverless, open source cloud platform.” https://openwhisk.apache.org/ (accessed Apr. 22, 2021).
[133] “Home | OpenFaaS - Serverless Functions Made Simple.” https://www.openfaas.com/ (accessed Apr. 16, 2021).
[134] “Knative,” *Knative*. https://knative.dev/ (accessed Apr. 20, 2021).
[135] “Workflows & Pipelines | Argo.” https://argoproj.github.io/projects/argo/ (accessed Apr. 28, 2021).
[136] “Kubeflow Overview,” *Kubeflow*. /docs/started/kubeflow-overview/ (accessed Apr. 27, 2021).
[137] “Autoscaling - OpenFaaS.” https://docs.openfaas.com/architecture/autoscaling/ (accessed Apr. 27, 2021).
[138] B. Di Martino, G. Cretella, and A. Esposito, “Towards a unified owl ontology of cloud vendors’ appliances and services at paas and saas level,” in *2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems*, 2014, pp. 570–575.
[139] F. Moscato, R. Aversa, B. Di Martino, T.-F. Fortiş, and V. Munteanu, “An analysis of mosaic ontology for cloud resources annotation,” in *2011 federated conference on computer science and information systems (FedCSIS)*, 2011, pp. 973–980.
[140] E. Brandtzæg, “CloudML: A DSL for model-based realization of applications in the cloud,” Master’s Thesis, 2012.
[141] K. Sahlmann and T. Schwotzer, “Ontology-based virtual IoT devices for edge computing,” in *Proceedings of the 8th International Conference on the Internet of Things*, 2018, pp. 1–7.
[142] G. Chen, T. Jiang, M. Wang, X. Tang, and W. Ji, “Modeling and reasoning of IoT architecture in semantic ontology dimension,” *Comput. Commun.*, vol. 153, pp. 580–594, 2020.
[143] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor, “IoT-Lite: a lightweight semantic model for the Internet of Things,” in *2016 INTL IEEE conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress (uic/atc/scalcom/cbdcom/iop/smartworld)*, 2016, pp. 90–97.
[144] “Kubeless.” https://kubeless.io/ (accessed Apr. 16, 2021).
[145] “Kubernetes Scheduler,” *Kubernetes*. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ (accessed Apr. 01, 2021).
[146] “Volcano.” https://volcano.sh/en/ (accessed Apr. 16, 2021).
[147] “Welcome to Apache YuniKorn (Incubating) | Apache YuniKorn (Incubating).” https://yunikorn.apache.org/ (accessed Apr. 16, 2021).
[148] *apache/incubator-yunikorn-core*. The Apache Software Foundation, 2021.
[149] *IBM/kube-safe-scheduler*. International Business Machines, 2021.
[150] *IBM/multi-cluster-app-dispatcher*. International Business Machines, 2021.
[151] “Scheduling Policies,” *Kubernetes*. https://kubernetes.io/docs/reference/scheduling/policies/ (accessed Apr. 16, 2021).
[152] *kubernetes-sigs/scheduler-plugins*. Kubernetes SIGs, 2021.
[153] “Scheduler Configuration,” *Kubernetes*. https://kubernetes.io/docs/reference/scheduling/config/ (accessed Apr. 16, 2021).
[154] “Configure Multiple Schedulers,” *Kubernetes*. https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/ (accessed Apr. 16, 2021).
[155] *kubernetes-sigs/descheduler*. Kubernetes SIGs, 2021.

[156] G. Aumala, E. Boza, L. Ortiz-Avilés, G. Totoy, and C. Abad, “Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms,” in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 282–291, doi: 10.1109/CCGRID.2019.00042.
[157] M. Stein, “The serverless scheduling problem and NOAH,” *ArXiv Prepr. ArXiv180906100*, 2018.
[158] A. Suresh and A. Gandhi, “Fnsched: An efficient scheduler for serverless functions,” in *Proceedings of the 5th International Workshop on Serverless Computing*, 2019, pp. 19–24.
[159] A. James and D. Schien, “A Low Carbon Kubernetes Scheduler.,” 2019.
[160] J. Kim and K. Lee, “FunctionBench: A Suite of Workloads for Serverless Cloud Function Service,” in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Jul. 2019, pp. 502–504, doi: 10.1109/CLOUD.2019.00091.
[161] J. Kim and K. Lee, “Practical Cloud Workloads for Serverless FaaS,” in *Proceedings of the ACM Symposium on Cloud Computing*, New York, NY, USA, Nov. 2019, p. 477, doi: 10.1145/3357223.3365439.
[162] T. Yu *et al.*, “Characterizing serverless platforms with serverlessbench,” in *Proceedings of the 11th ACM Symposium on Cloud Computing*, New York, NY, USA, Oct. 2020, pp. 30–44, doi: 10.1145/3419111.3421280.
[163] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms,” *J. Parallel Distrib. Comput.*, vol. 74, no. 10, p. 2899, Jun. 2014, doi: 10.1016/j.jpdc.2014.06.008.
[164] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, “Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator,” Chicago, United States, May 2016, Accessed: Apr. 16, 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01333471.
[165] “Thlr/master-memoir,” *GitHub*. https://github.com/Thlr/master-memoir (accessed Apr. 16, 2021).
[166] D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, and B. Grot, “Benchmarking, Analysis, and Optimization of Serverless Function Snapshots,” *ArXiv210109355 Cs*, Feb. 2021, doi: 10.1145/3445814.3446714.
[167] “containerd – An industry-standard container runtime with an emphasis on simplicity, robustness and portability.” https://containerd.io/ (accessed Apr. 29, 2021).
[168] “Firecracker.” https://firecracker-microvm.github.io/ (accessed Apr. 29, 2021).
[169] “Configure Multiple Schedulers,” *Kubernetes*. https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/ (accessed Apr. 20, 2021).
[170] *openshift/cluster-kube-descheduler-operator*. OpenShift, 2021.
[171] “Performance Addon Operator for low latency nodes | Scalability and performance | OpenShift Container Platform 4.6.” https://docs.openshift.com/container-platform/4.6/scalability_and_performance/cnf-performance-addon-operator-for-low-latency-nodes.html (accessed Apr. 20, 2021).
[172] “kubernetes/enhancements,” *GitHub*. https://github.com/kubernetes/enhancements (accessed Apr. 20, 2021).
[173] “ManageIQ.” http://manageiq.org/ (accessed Apr. 20, 2021).
[174] “Red Hat Advanced Cluster Management for Kubernetes.” https://www.redhat.com/en/technologies/management/advanced-cluster-management (accessed Apr. 20, 2021).
[175] “Open Cluster Management,” *GitHub*. https://github.com/open-cluster-management (accessed Apr. 22, 2021).
[176] “GitOps | GitOps is Continuous Deployment for cloud native applications.” https://www.gitops.tech/ (accessed Apr. 20, 2021).
[177] “Argo CD - Declarative GitOps CD for Kubernetes.” https://argoproj.github.io/argo-cd/ (accessed Apr. 20, 2021).
[178] “Kustomize - Kubernetes native configuration management.” https://kustomize.io/ (accessed Apr. 22, 2021).
[179] “Welcome | About | OpenShift Container Platform 4.7.” https://docs.openshift.com/container-platform/4.7/welcome/index.html (accessed Apr. 01, 2021).

[180] “Configuring the default scheduler to control pod placement - Controlling pod placement onto nodes (scheduling) | Nodes | OpenShift Container Platform 4.7.” https://docs.openshift.com/container-platform/4.7/nodes/scheduling/nodes-scheduler-default.html#nodes-scheduler-default (accessed Apr. 01, 2021).
[181] “Placing pods relative to other pods using pod affinity and anti-affinity rules - Controlling pod placement onto nodes (scheduling) | Nodes | OpenShift Container Platform 4.7.” https://docs.openshift.com/container-platform/4.7/nodes/scheduling/nodes-scheduler-pod-affinity.html (accessed Apr. 23, 2021).
[182] “Assigning Pods to Nodes,” *Kubernetes*. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/ (accessed Apr. 23, 2021).
[183] “Scheduler | Cluster Administration | OpenShift Container Platform 3.4.” https://docs.openshift.com/container-platform/3.4/admin_guide/scheduler.html#infrastructure-topological-levels (accessed Apr. 23, 2021).
[184] E. Van Eyk, A. Iosup, S. Seif, and M. Thömmes, “The SPEC cloud group’s research vision on FaaS and serverless architectures,” in *Proceedings of the 2nd International Workshop on Serverless Computing*, 2017, pp. 1–4.
[185] J. Michener, “Security Issues With Functions as a Service,” *IT Prof.*, vol. 22, no. 5, pp. 24–31, 2020.
[186] “OWASP Serverless Top 10.” https://owasp.org/www-project-serverless-top-10/ (accessed Mar. 23, 2021).
[187] V. Yussupov, U. Breitenbücher, F. Leymann, and M. Wurster, “A systematic mapping study on engineering function-as-a-service platforms and tools,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 229–240.
[188] S. Brenner and R. Kapitza, “Trust more, serverless,” in *Proceedings of the 12th ACM International Conference on Systems and Storage*, 2019, pp. 33–43.
[189] B. Trach, O. Oleksenko, F. Gregor, P. Bhatotia, and C. Fetzer, “Clemmys: Towards secure remote execution in FaaS,” in *Proceedings of the 12th ACM International Conference on Systems and Storage*, 2019, pp. 44–54.
[190] F. Alder, N. Asokan, A. Kurnikov, A. Paverd, and M. Steiner, “S-faas: Trustworthy and accountable function-as-a-service using intel SGX,” in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019, pp. 185–199.
[191] A. Al Omar, M. Z. A. Bhuiyan, A. Basu, S. Kiyomoto, and M. S. Rahman, “Privacy-friendly platform for healthcare data in cloud based on blockchain environment,” *Future Gener. Comput. Syst.*, vol. 95, pp. 511–521, 2019.
[192] C. Esposito, A. De Santis, G. Tortora, H. Chang, and K.-K. R. Choo, “Blockchain: A panacea for healthcare cloud-based data security and privacy?,” *IEEE Cloud Comput.*, vol. 5, no. 1, pp. 31–37, 2018.
[193] “Credential.” https://credential.eu/ (accessed Mar. 26, 2021).
[194] “MUSA Project Website.” https://www.musa-project.eu/ (accessed Mar. 26, 2021).
[195] qktheme, “PRISMACLOUD – PRIvacy and Security MAintaining services in the CLOUD.” https://prismacloud.eu/ (accessed Mar. 26, 2021).
[196] “Secure Big Data Processing in Untrusted Clouds | SecureCloud Project | H2020 | CORDIS | European Commission.” https://cordis.europa.eu/project/id/690111 (accessed Mar. 26, 2021).
[197] “Sereca Project.” https://www.serecaproject.eu/ (accessed Mar. 26, 2021).
[198] “Secure Provisioning of Cloud Services based on SLA management | SPECS Project | FP7 | CORDIS | European Commission.” https://cordis.europa.eu/project/id/610795 (accessed Mar. 26, 2021).
[199] “SecUre iNFormatIon SHaring in federated heterogeneous private clouds – Servizi ed Infrastrutture Cloud Avanzate.” http://www.sunfishproject.eu/ (accessed Mar. 26, 2021).
[200] “SWITCH.” http://www.switchproject.eu/ (accessed Mar. 26, 2021).
[201] “Home | Tredisec.” http://tredisec.eu/ (accessed Mar. 26, 2021).
[202] “UNICORN project.” http://unicorn-project.eu/ (accessed Mar. 26, 2021).
[203] “About Witdom | Witdom.” http://witdom.eu/ (accessed Mar. 26, 2021).

[204] S. Alansari, F. Paci, A. Margheri, and V. Sassone, "Privacy-Preserving Access Control in Cloud Federations," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, Jun. 2017, pp. 757–760, doi: 10.1109/CLOUD.2017.108.
[205] "AWS Secrets Manager | Rotate, Manage, Retrieve Secrets | Amazon Web Services (AWS)," *Amazon Web Services, Inc.* https://aws.amazon.com/secrets-manager/ (accessed Mar. 29, 2021).
[206] "Secret Manager," *Google Cloud*. https://cloud.google.com/secret-manager (accessed Mar. 29, 2021).
[207] "Key Vault | Microsoft Azure." https://azure.microsoft.com/en-us/services/key-vault/ (accessed Mar. 29, 2021).
[208] "Akeyless Vault for Secrets Management," *Akeyless*. https://www.akeyless.io/product-secrets-management/ (accessed Mar. 29, 2021).
[209] "Vault by HashiCorp," *Vault by HashiCorp*. https://www.vaultproject.io/ (accessed Mar. 29, 2021).
[210] "Keywhiz." https://square.github.io/keywhiz/ (accessed Mar. 29, 2021).
[211] "Confidant: Your secret keeper." https://lyft.github.io/confidant/ (accessed Mar. 29, 2021).
[212] "Manage sensitive data with Docker secrets," *Docker Documentation*, Mar. 26, 2021. https://docs.docker.com/engine/swarm/secrets/ (accessed Mar. 29, 2021).
[213] *pinterest/knox*. Pinterest, 2021.
[214] M. A. Sotelo Monge, J. Maestre Vidal, and G. Martínez Pérez, "Detection of economic denial of sustainability (EDoS) threats in self-organizing networks," *Comput. Commun.*, vol. 145, pp. 284–308, Sep. 2019, doi: 10.1016/j.comcom.2019.07.002.
[215] A. Shawahna, M. Abu-Amara, A. S. H. Mahmoud, and Y. Osais, "EDoS-ADS: An Enhanced Mitigation Technique Against Economic Denial of Sustainability (EDoS) Attacks," *IEEE Trans. Cloud Comput.*, vol. 8, no. 3, pp. 790–804, Jul. 2020, doi: 10.1109/TCC.2018.2805907.
[216] Silva, Paulo, Edmundo Monteiro, and Paulo Simoes. "Privacy in the cloud: A survey of existing solutions and research challenges." *IEEE Access* 9 (2021): 10473-10497. (accessed Sep. 27,2022)
[217] Azeez, Nureni Ayofe, and Charles Van der Vyver. "Security and privacy issues in e-health cloud-based system: A comprehensive content analysis." *Egyptian Informatics Journal* 20.2 (2019): 97-108.
[218] Tomás J, Rasteiro D, Bernardino J. Data Anonymization: An Experimental Evaluation Using Open-Source Tools. Future Internet. 2022; 14(6):167. https://doi.org/10.3390/fi14060167
[219] Yussupov, Vladimir, et al. "On the serverless nature of blockchains and smart contracts." arXiv preprint arXiv:2011.12729 (2020).
[220] Ghaemi, Sara, Hamzeh Khazaei, and Petr Musilek. "Chainfaas: An open blockchain-based serverless platform." IEEE Access 8 (2020): 131760-131778.
[221] Jie, S. O. N. G., et al. "Research advances on blockchain-as-a-service: Architectures, applications and challenges." Digital Communications and Networks (2021).
[222] Udokwu, Chibuzor, et al. "The state of the art for blockchain-enabled smart-contract applications in the organization." 2018 Ivannikov Ispras Open Conference (ISPRAS). IEEE, 2018.
[223] E. Bisong, "Kubeflow and kubeflow pipelines," in Building Machine Learning and Deep Learning Models on Google Cloud Platform, pp. 671–685, Springer, 2019.
[224] Ristov, Sasko, Stefan Pedratscher, and Thomas Fahringer. "AFCL: An abstract function choreography language for serverless workflow specification." Future Generation Computer Systems 114 (2021): 368-382..
[225] Malawski, Maciej, et al. "Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions." Future Generation Computer Systems 110 (2020): 502-514.
[226] John, Aji, et al. "SWEEP: accelerating scientific research through scalable serverless workflows." Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion. 2019..
[227] Zhang, Haoran, et al. "Fault-tolerant and transactional stateful serverless workflows." 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI20), 2020.
[228] Burckhardt, Sebastian, et al. "Serverless workflows with durable functions and netherite." arXiv preprint arXiv:2103.00033 (2021

[229] Erwin van Eyk, Alexandru Iosup, Cristina L. Abad, Johannes Grohmann, and Simon Eismann. 2018. A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18). Association for Computing Machinery, New York, NY, USA, 21–24. DOI:https://doi.org/10.1145/3185768.3186308
[230] Barcelona-Pons, P. Garcıa-Lopez, ́A. Ruiz, A. Gomez-Gomez,G. Parıs, and M. Sanchez-Artigas, “Faas orchestration of parallel workloads,” in Proc. of the 5th International Workshop on Serverless Computing, pp. 25–30, 2019
[231] López, P.G., Sánchez-Artigas, M., París, G., Pons, D.B., Ollobarren, Á.R. and Pinto, D.A., 2018, December. Comparison of faas orchestration systems. In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion) (pp. 148-153). IEEE
[232] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2021. Speedo: Fast dispatch and orchestration of serverless workflows. Proceedings of the ACM Symposium on Cloud Computing. Association for Computing Machinery, New York, NY, USA, 585–599. DOI:https://doi.org/10.1145/3472883.3486982.

ANNEX I. IDENTITY AND ACCESS MANAGEMENT SERVICES

Table 2 - Identity and access, policy and role management services of cloud providers

Identity and Access Management	Amazon Web Services	➢ AWS IAM ➢ AWS for Microsoft Active Directory o AD Connector o Simple AD o Managed Microsoft AD ➢ Federation via Identity Providers (OIDC & SAML2)
	Google Cloud Platform	➢ Google Cloud Identity ➢ Federation via third party IdPs (OIDC & SAML2) ➢ Cloud Directory Sync via LDAP
	Microsoft Azure	➢ Azure Active Directory ➢ Federation via IdPs (OIDC & SAML2)
Policy and Role Management	Amazon Web Services	➢ AWS IAM ➢ Access Policies configuration per user/group/role
	Google Cloud Platform	➢ Permissions assigned on a per project basis ➢ Users can belong to multiple projects ➢ All permissions are applied to all resources of the project ➢ Standard roles can be customized and applied to users and groups
	Microsoft Azure	➢ Permissions are applied on Subscriptions, resource groups and individual resources. All resources belong to one group that in turn can be configured in hierarchies. ➢ Common roles exist but additional ones can be configured.
Access Control Management	Amazon Web Services	➢ Account keys – god key for account ➢ Access keys ➢ Temporary security credentials (OIDC & SAML)
	Google Cloud Platform	➢ Account keys – god key for account ➢ Service account keys ➢ Storage access token ➢ API keys ➢ User account tokens
	Microsoft Azure	➢ Account keys – god key for account ➢ Storage access keys – root key for storage ➢ Connection strings ➢ AD auth ➢ MSI tokens ➢ SAS tokens ➢ Mutual TLS

ANNEX II. QUESTIONNAIRE TEMPLATE

The template used and followed by all partners is the following.

	Section	Description
S	ID	Req-TASK-x (unique id) Where TASK = task number x = descriptive short identifier Example: "Req-3.2-Priv" for a privacy requirement that maps to task 3.2
	Dependencies	Requirement ID of the dependencies that this requirement depends on.
	Type	Choose from the list: - FUNC: Functional Suitability - DATA: Data - USE: Usability - REL: Reliability - SEC: Security - PERF: Performance Efficiency - COMP: Compatibility - MAINT: Maintainability - PORT: Portability (For examples see the table below)
	Short name	Meaningful and not too long
	Actors	The actors involved in this scenario
	Description	General description
	Additional Information	
	Priority (MoSCoW)	This allow to identify the priority of the requirements; it can be updated in the different iterations: M: Must-have. Mandatory requirement. S: Should-have. Desirable requirement. C: Could-have. Optional requirement. W: Will-not-have. Possible future enhancement
M	Strong encouragement for defining concrete measures. Means of measuring the goal achievement. Milestones, metrics etc.	
A	How achievable is the requirement? This will have to take into account the Objectives, the Priority and any possible foreseen obstacles.	
R	Objectives of this requirement. Why is the goal worthwhile? What needs does it fulfill and how relevant are they to the project and/or use-case scenario? Does it satisfy any future dependencies (to be filled with the requirement IDs)?	
T	Strong encouragement for defining concrete timelines (Such as milestones, delivery deadlines according to work plan etc.). When should this requirement be achieved? This goes hand-in-hand with the future dependencies and their timelines.	

The categories used within the template are the following. They are derived from the ISO25010 standard.

	Category	Examples	Name
Functional	Functional Suitability	➢ Functional Completeness ➢ Functional Correctness ➢ Functional Appropriateness	FUNC
	Data	➢ Data requirements ➢ Data preconditions ➢ Data postconditions	DATA
Non-functional	Usability	➢ Appropriateness Recognizability ➢ Learnability ➢ Operability ➢ User Error Protection ➢ User Interface Aesthetics ➢ Accessibility	USE
	Reliability	➢ Maturity ➢ Availability ➢ Fault Tolerance ➢ Recoverability	REL
	Security	➢ Confidentiality ➢ Integrity ➢ Non-repudiation ➢ Authenticity ➢ Accountability	SEC
	Performance Efficiency	➢ Time Behavior ➢ Resource Utilization ➢ Capacity	PERF
	Compatibility	➢ Co-existence ➢ Interoperability	COMP
	Maintainability	➢ Modularity ➢ Reusability ➢ Analyzability ➢ Modifiability ➢ Testability	MAINT
	Portability	➢ Adaptability ➢ Installability ➢ Replaceability	PORT

DISCLAIMER

COPYRIGHT MESSAGE