NebulOuS

A Meta Operating System
for Brokering Hyper-Distributed Applications
on Cloud Computing Continuums

D2.1 Requirements and Conceptual Architecture of the NebulOuS Meta-OS

Funded by
the European Union

Project Acronym/Title	NebulOuS: A Meta Operating System for Brokering Hyper-Distributed Applications on Cloud Computing Continuums
Grant Agreement No.:	101070516
Call	World leading data and computing technologies (HORIZON-CL4-2021-DATA-01)
Project duration	36 months | 1 September 2022 – 31 August 2025
Deliverable title	Requirements and Conceptual Architecture of the NebulOuS Meta-OS
Deliverable reference	D2.1
Version	1.0
WP	2
Delivery Date	01/3/2023
Dissemination level	Public
Deliverable lead	UBI
Authors	Christos-Alexandros Sarros, Giannis Ledakis, Panagiotis Gouvas (UBI), Andreas Tsagkaropoulos, Dimitris Apostolou, Ioannis Patiniotakis, Yiannis Verginadis (ICCS), Radosław Piliszek, Paweł Skrzypek (7bulls), Gregoris Savvas, Fotis Paraskevopoulos (EXZ), Ali Fahs (AE), David Llordes Palou, Robert Sanfeliu Prat, Mario Reyes de los Mozos (EUT), Evangelos Barmpas, Simeon Veloudis, Iraklis Paraskakis (SEERC), Amir Taherkordi, Rudolf Schlatte, Geir Horn (UiO), Cristina Hilario Gomez (TiD), Pedro Teixeira, Sofia Rosas (UW), Michal Klosinski (AF), Maria Karageorgiou (AUG), Moritz von Stietencron (BIBA), Amir Azimian (FIRE)
Reviewers	Rita Santiago (UW), Mario Reyes (EUT)
Abstract	Report on the current work in all the relevant research areas of NebulOuS project and capture of relevant requirements. Bird's view on the general architecture of the system and should lay a foundation for the integration. Each component will be described through its functional and non-functional requirements
Keywords	Meta-OS, Cloud computing, Edge computing, Fog Computing, IoT

Dissemination Level:	
PU	Public, fully open
Type	
R	Document, report (excluding the periodic and final reports)

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them. | Grant Agreement No.: 101070516

Version History

Version	Date	Owner	Author(s)	Changes to previous version
0.1	2023-01-05	UBI	Christos-Alexandros Sarros	Table of Contents
0.2	2023-01-20	TiD	Cristina Hilario Gomez	Initial draft of High-level Requirements section
0.3	2023-01-27	UiO	Geir Horn	Initial draft on SotA section
0.4	2023-02-03	UBI	Christos-Alexandros Sarros	Initial draft of Architecture section
0.5	2023-02-10	UBI	Andreas Tsagkaropoulos, Dimitris Apostolou, Ioannis Patiniotakis, Yiannis Verginadis (ICCS), Radosław Piliszek, Paweł Skrzypek (7bulls), Gregoris Savvas, Fotis Paraskevopoulos (EXZ), Ali Fahs (AE), David Llordes Palou, Robert Sanfeliu Prat, Mario Reyes de los Mozos (EUT), Evangelos Barmpas, Simeon Veloudis, Iraklis Paraskakis (SEERC), Amir Taherkordi, Rudolf Schlatte, Geir Horn (UiO), Christos-Alexandros Sarros, Giannis Ledakis (UBI)	Additional input on SotA areas and Architecture components
0.6	2023-02-17	UBI	Christos-Alexandros Sarros	First full draft
0.7	2023-02-22	UBI	Rita Santiago (UW), Mario Reyes (EUT)	Internal reviews
0.8	2023-02-24	UBI	Pedro Teixeira, Sofia Rosas (UW), Michal Klosinski (AF), Maria Karageorgiou (AUG), Moritz von Stietencron (BIBA), Amir Azimian (FIRE)	Added use case descriptions
0.9	2023-02-28	UBI	Christos-Alexandros Sarros	Revised full draft
1.0	2023-03-01	UBI	Maria Navarro Abellán	Submitted version

Funded by the European Union

Table of Contents

Funded by the European Union

Funded by
the European Union

List of Figures

List of Tables

Funded by
the European Union

List of Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ABAC	Attribute-based access control
ACK	Alibaba Container service for Kubernetes
AI	Artificial Intelligence
AIS	Artificial Immune Systems
AKS	Azure Kubernetes Service
ALFA	Abbreviated Language For Authorization
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARIMA	Integrated Moving Average
ARM	Advanced RISC Machines
ARMA	Auto Regressive Moving Average
AuG	Augmenta
AWS	Amazon Web Services
BaaS	Backend-as-a-Service
BC	Blockchain
BGP	Border Gateway Protocol
BLE	Bluetooth Low Energy
BQA	Brokerage Quality Assurance
CaaS	Container-as-a-Service
CAMEL	Cloud Application Modelling and Execution Language
CAPI	Cluster API
CCTV	Closed-circuit television
CEP	Complex Event Processing
CFSB	Cloud & Fog Service Brokerage
CIM	Context Information Management
CIS	Center for Internet Security
CLI	Command Line Interface
CNCF	Cloud-Native Computing Foundation
CNI	Container Network Interface
CNN	Convolutional Neural Networks
CoAP	Constrained Application Protocol
CORD	Central Office Re-architected as a Datacenter
CP	Critical Path
CPU	Central Processing Unit
CRD	Custom Resource Definitions
CRI	Container Runtime Interface
CRI	Container Runtime Interface
DAC	Discretionary Access Control
DAG	Directed Acyclic Graph
DB	Database
dB	decibel

Funded by the European Union

DC/OS	Distributed Cloud Operating System
DCA	Dendritic Cell Algorithm
DKP	D2iQ Enterprise Kubernetes Platform
DNS	Domain Name Service
DPDK	Data Plane Development Kit
DSL	Domain-Specific Language
DSP	Data Stream Processing
DUM	Distribució Urbana de Mercaderies (Urban goods distribution)
eBPF	extended Berkeley Packet Filter
EC2	(Amazon) Elastic Compute Cloud
EIP	Enterprise Integration Patterns
EKS	Elastic Kubernetes Service
EMS	Event Management System
ESB	Enterprise Service Bus
ETSI	European Telecommunications Standards Institute
FaaS	Function-as-a-Service
FIPS	Federal Information Processing Standard
GA	Grant Agreement
GAMMA	Gateway API for Mesh Management and Administration
GAN	Generative Adversarial Networks
GB	GigaByte
GHz	GigaHertz
GKE	Google Kubernetes Engine
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
H2020	Horizon 2020
HA	High Availability
HCI	Hyper-Converged Infrastructure
HPA	Horizontal Pod Autoscaler
HPC	High-Performance Computing
HTTP	HyperText Transfer Protocol
HW	Hardware
I/O	Input/Output
IaaS	Infrastructure-as-a-Service
IAM	Identity and Access Management
IBAC	Identity Based Access Control
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IoTMA	Internet of Things Model and Analytics
IP	Internet Protocol
IPE	Interworking Proxy Entities
IT	Information Technology
JMX	Java Management Extensions
JSON	JavaScript Object Notation

Funded by the European Union

JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
K8s	Kubernetes
KEDA	Kubernetes-based Event Driven Autoscaling
KP	Knowledge Processors
KPI	Key Performance Indicator
L1/L2/Lx	Layer 1/Layer 2/Layer x
LKE	Linode Kubernetes Engine
MAC	Mandatory Access Control
MCC	Mobile Cloud Computing
MCDM	MultiCriteria Decision Making
MEC	Mobile Edge Computing
MKE	Mirantis Kubernetes Engine
ML	Machine Learning
MQTT	MQ Telemetry Transport
MrB	Mercabarna
mTLS	mutual TLS
NDI	Several non-destructive inspection methods
NFC	Near Field Communication
NFV	Network Function Virtualization
NGAC	Next Generation Access Control
NGSI-LD	Next Generation Service Interfaces - Linked Data
NVGRE	Network Virtualization using Generic Routing Encapsulation
OAM	Open Application Model
OASIS	Organization for the Advancement of Structured Information Standards
OPC UA	OPC Unified Architecture
ORDGE	Organisational Requirements Definition Governance and Enforcement
OS	Operating System
OSI	Open Systems Interconnection
OVN	Open Virtual Network
OvS	Open vSwitch
OWL	Web Ontology Language
PaaS	Platform-as-a-Service
PAP	Policy Administration Point
PCA	Principal Component Analysis
PCP	Partial Critical Path
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RBAC	Role-Based Access Control
RBAC	Role-based access control
RDBMS	Relational Database Management Systems
RDF	Resource Description Framework

Funded by the European Union

REST	Representational state transfer
RFID	Radio-Frequency Identification
RKE	Rancher Kubernetes Engine
RM	Resource Manager
RNN	Recurrent Neural Network
RPC	Remote Procedure Call
SaaS	Software-as-a-Service
SAL	Scheduling Abstraction Layer
SAREF	Smart Applications Reference
SCB	Support and Confidence-based Monitoring
SCE	Smart Contract Encapsulator
SDK	Software Development Kit
SDN	Software-Defined Networking
SIB	Semantic Information Brokers
SLA	Service-Level Agreements
SLO	Service Level Objectives
SME	Small and Medium Enterprise
SMG	Semantic Mediation Gateways
SMI	Service Mesh Interface
SOA	Service-Oriented Architecture
SOSA	Sensor, Observation, Sample, and Actuator ontology
SotA	State-of-the-Art
SQL	Structured Query Language
SSN	Semantic Sensor Network ontology
SVM	Support Vector Machines
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOSCA	Topology and Orchestration Specification for Cloud Applications
TSA	Time Series Analysis
TSDBMS	Time Series Database Management Systems
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNet	(Azure) Virtual Network
VPA	Vertical Pod Autoscaler
VPC	Virtual Private Cloud
VPN	Virtual Private Network
VXLAN	Virtual Extensible Local Area Network
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication
WLAN	Wireless Local Area Networks
WP	Work Package
WPAN	Wireless Personal Area Networks
XACML	Access Control Markup Language

XDP	eXpress Data Path
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language
Zenoh	Zero Overhead Network Protocol

Funded by the European Union

Executive Summary

The NebulOuS project aims to design and implement a novel Meta Operating System which enables secure and optimal application provisioning and reconfigurations over the cloud computing continuum. To this end, disparate resources from the far edge to public and private cloud offerings, as well as processing nodes with significant capacity in between, will be exploited to augment modern hyper-distributed applications. Our vision is to enable transient fog brokerage ecosystems that seamlessly exploit edge and fog nodes, in conjunction with multi-cloud resources, to cope with the requirements posed by low latency applications.

This document provides the system requirements, scientific and technological state of the art and architecture of the NebulOuS Meta-OS. Driven by the use cases, we first present the high-level system requirements. Subsequently, we survey the relevant scientific and technological state-of-the-art and document the current work in each relevant area. Last, we provide the conceptual and logical views of the NebulOuS architecture, along with its constituent components and interrelations. Each component is described in relation to the functional and non-functional requirements it covers.

As this is the first technical deliverable of the NebulOuS project, which presents the general architecture of the system, it provides the main point of reference for the technical implementation and integration work that follows. The physical architecture that details the actual implementation along with the respective integration work, will be reported in D6.1 "*1st Release of the NebulOuS Integrated Platform & Use Case Planning*" (M18) and *D6.2 "Final Release of the NebulOuS Integrated Platform & Pilot Demonstrators Evaluation Report*" (M36).

Funded by the European Union

1. Introduction

1.1 Deliverable objectives

The objective of NebulOuS is to create a Meta-Operating System for brokering Hyper-distributed applications on cloud computing continuums. To achieve this, the design of an architecture that is capable to support fog brokerage in different cloud computing continuum settings is needed.

This architecture is presented at the conceptual and logical level in the present deliverable. Furthermore, this deliverable includes all work that was performed to arrive at the NebulOuS architecture, including its high-level requirements and the current state of the respective scientific and technical fields.

Based on the above, D2.1 has three inter-related objectives: First, to capture the high-level system requirements for the NebulOuS Meta-OS, by focusing on the use cases to be supported and analysing their individual requirements. Second, to report the scientific and technological state-of-the-art in all research areas that are relevant to the NebulOuS project. Third, based on the previous analysis, to present the NebulOuS conceptual and logical architecture, which lays the foundation for the technical implementation which will be performed during the following months. In this respect, it describes the individual components which constitute the NebulOuS platform along with their interactions, as well as their relation to the functional and non-functional requirements.

1.2 Relation to other Deliverables and WPs

Deliverable D2.1 presents the work carried out in the context of ***Task 2.1***: *"Scientific and technological State-of-the-art analysis"*, ***Task 2.2***: *"High-level requirements analysis"* and ***Task 2.3***: *"NebulOuS Platform Architecture"*. By arriving at the NebulOuS architecture, D2.1 is a reference point for the work to be performed in all technical Work Packages (WP2-WP5), in the context of which the individual components and mechanisms will be developed. As such, D2.1 forms the **1st project Milestone.**

Furthermore, D2.1 is closely related to the work performed in the context of WP6 "*Integrated Meta-OS & Pilot Demonstrators*". This is especially true of the integration work, which effectively begins with the definition of the architecture. In this respect, two particular deliverables are closely related to D2.1: *Deliverable D6.1 "1st Release of the NebulOuS Integrated Platform & Use Case Planning"* and *Deliverable D6.2 "Final Release of the NebulOuS Integrated Platform & Pilot Demonstrators Evaluation Report"*. As this deliverable presents the logical architecture of the NebulOuS Meta-OS, it does not incorporate the technical decisions made by the respective tasks. The technical architecture - which incorporates those results - will be formulated and reported on D6.1 and D6.2.

1.3 Deliverable structure

The following structure is followed by this deliverable:

- Section 1 introduces the deliverable objectives, its relation to other deliverables and work packages and its overall structure
- Section 2 defines the high-level system requirements (functional and non-functional). For presentation purposes, it first provides an overview of the use cases and subsequently provides the methodology that was followed.
- Section 3 analyses the scientific and technological state-of-the-art. It begins by presenting the consortium methodology and subsequently focuses on the individual areas that are relevant to NebulOuS.
- Section 4 details the NebulOuS architecture. It first provides an overview of the methodology followed towards its definition, before proceeding to the presentation of the conceptual and logical view. After introducing the

architecture, the section focuses on the individual components and their interrelation, discussing how each one covers the requirements identified in Section 2.
- Finally, Section 5 concludes the document.

2. High-level requirements

This section presents the high-level requirements for the NebulOuS platform. First, the use cases that will validate the platform will be described. Next, the methodology used to gather the requirements will be introduced. Finally, the system requirements will be presented.

As described in the proposal, there are 4 pilot demonstrators, with the numbering shown in the first column of Table 1. The 2nd pilot covers 2 main objectives, which are (i) Development of Mercabarna's internal traffic Digital Twin to support Mercabarna's intra-logistics operations management, and (ii) Capture and management of the data on trucks' destination (city areas) to improve the last-mile goods distribution in the city. Every objective will be addressed in a separate application. For this reason, the pilot has been divided into the 2 use cases described in the second column of Table 1.

Overall, 6 use cases will be developed, since the 1st pilot also involves 2 use cases. To be able to identify the requirements relevant to each application, an identifier has been assigned to each use case, that is different from those originally assigned in the GA. The list of pilots and use case identifiers are described in the table below.

Table 1: Use case identifiers

Pilot Demonstrator	Use case ID
1st Pilot: Maintenance	UC1.1: Windmill maintenance
	UC1.2: Computer vision for city maintenance
2nd Pilot: Efficient, Connected and Sustainable management of infrastructure, freights and resources	UC2.1: Intra-logistics operations improvement
	UC2.2: Last mile delivery optimization
3rd Pilot: Precision Agriculture	UC3
4th Pilot: Crisis Management	UC4

2.1 Use Case Descriptions

This subsection describes each use case that will evaluate the platform.

UC1.1: Windmill Maintenance

This Use Case exploits AI models for identifying windmill blade damage based on images collected by drones (both actual and prospective damages will be considered). Given bandwidth and connection stability constraints coupled with the limited flight time of a drone which is typically below 30 minutes, the selection of relevant images (including objects that need to be further analyzed or general non-repetitive images of clear regions) occurs at the edge (ground station) and only relevant data are transmitted over the edge-cloud channel. Edge processing will also call for a new acquisition (detailed images of certain regions) when required. Inspection time and operator effort can be significantly reduced at IoT level (on the drone, exploiting power/weight-efficient GPU modules, already available as prototypes today) by providing image quality feedback to the operator, allowing less conservative flight. Additionally, real-time feedback from AI-powered image analysis can be used as input for a drone's autonomous mission control unit. Tools provided by NebulOuS will provide means to deploy, monitor and orchestrate platform components effectively.

Funded by
the European Union

UC1.2: Computer Vision for City Maintenance

This use case exploits computer vision solutions to detect damage in public buildings and infrastructures. The current solution is based on edge-deployed computer vision algorithms to detect damage, publishing this information into a cloud-centric persistence layer. The output of this use case – alarmistics on damage on buildings – should be provided by city managers and maintenance crews as fast as possible, hence the need for a low latency solution within this use case. In addition, it is expected that the current computer vision algorithms need to be retrained, updated or new models added, with the necessary repositories needing a seamless scaling and configuring across edge and cloud resources. The NebulOuS project will provide the tools to decrease the overall necessary time to train a new computer vision model and deploy a new instance or test and train new models.

UC2.1: Intra-logistics Operations Improvement

Mercabarna (MrB) is a small living city, where many vehicles of all types (from big trucks to vans, cars or forklifts) work together to carry on MrB's customers' operations. Effective management of internal traffic is of utmost importance to enable smooth operations and prevent jams or bottlenecks that would delay the lifecycle of the entire MrB. With this objective, a real-time traffic monitoring system for intra-logistics operations will be developed. The system will be based on a network of strategically located CCTV cameras recording the movements of vehicles along MrB roads. AI algorithms will analyse the live streams of these cameras to identify the vehicles (read plate numbers and vehicle typology). The actions they take (continue straight on a road, turn right, turn left, etc..). Crossing the events detected by the different cameras, the solution will be able to track the routes of the vehicles inside MrB. This information presented by MrB managers will allow them to evaluate the traffic in real-time and take decisions on how to mitigate congestion. Moreover, a Decision Support System will be developed to allow these managers to simulate traffic management policies (e.g: make one street to be one direction only) and evaluate the impact on the traffic.

UC2.2: Last Mile Delivery Optimization

Mercabarna (MrB) is a food city that operates 24 hours a day to guarantee the supply of fresh food to the public. The facility houses 600 companies specialising in distributing, preparing, importing and exporting fresh and frozen products. More than 7,000 small trucks leave MrB's facilities every day and go to the city and its premises to distribute goods.

For this, this case study will focus on developing a real-time system to monitor transport vehicles and propose optimal routes for efficient goods delivery. Before the delivery vehicles leave MrB's facilities, the system will be responsible for planning the optimal delivery route for each of them, taking into account: i) the locations they have to visit and their time constraints; ii) the traffic conditions; iii) the expected occupation of the parking areas for delivery vehicles near the delivery locations (DUM areas). Once the vehicles leave MrB's facilities, the system will constantly monitor the fulfilment of the delivery route to detect any deviation from the said planning. This monitoring will be performed thanks to the analysis of real-time GPS data provided by vehicles. Moreover, changes to external conditions that might affect the viability of the proposed plan will be observed (e.g: unexpected traffic volume or DUM area usage). Should any of these events occur (delays on the schedule or changes in the predictions), the system will evaluate the necessity to trigger an automatic re-planification of the vehicle's routes to mitigate the possible delays.

UC3: Precision Agriculture

Augmenta (AuG) is a company that provides live precision agriculture solutions using IoT devices to many customers worldwide. Some of the most critical services that provide are the post-processing of operation data and its transformation into useful and high-quality visual information with the aim of each user being able to monitor the progress of their crops also to identify possible problems in their fields. The implementation of these services has been done through various data processing pipelines, depending on the number and size of the tasks, can become lengthy and time-consuming. Therefore, the current use case aims to exploit the existing infrastructure of IoT devices when they remain idle for a significant time in the field. The ultimate goal is the reduction of the tasks assigned to the cloud provider, which will bring significant cost decrease but also an optimization of the results in terms of time and the number of completed tasks. Given the network bandwidth and connection stability, NebulOuS will provide the demanding task scheduling that will take over the distribution of processes on active devices, ensuring a smooth completion of the tasks and secure data transmission.

Funded by the European Union

UC4: Crisis Management

The coordination of response crews, materials and equipment from different stakeholders is one of the major challenges in handling large-scale disasters like floods, earthquakes or wildfires. For the operational management of the situation, obtaining a clear picture of the situation in the concerned area is paramount, to allow for the efficient and safe deployment of the available resources. The technological potential for data collection and processing has rapidly increased in the last decade and is increasingly employed in most sectors. In disaster situations, however, the established structures for communication and, thus, processing capabilities are disrupted and cannot easily be used. Planning tools for day-to-day emergency services can thus often only be utilised to a limited extent. This use case will experiment with the flexible NebulOuS fog computing platform to enable broad use of communication and computing even in disaster situations by. Thus enabling the utilisation of AI algorithms in the edge-cloud-continuum to process the available data to verify the information and significantly enhance the situational awareness of the crisis management staff and response teams.

2.2 Methodology

The requirements presented in this document have been collected during several brainstorming sessions and online meetings with use case partners and the consortium. The information gathered has taken into account the architectural requirements of the platform, considering several aspects of the use cases, such as technical challenges, infrastructure requirements or security and privacy aspects.

The methodology involved, as a first step, the collection of the key requirements of each use case using a template and, as a second step, dividing them into functional and non-functional. As defined in the proposal, functional requirements specify the design-time and run-time capabilities of the platform. Non-functional requirements specify characteristics of security, privacy, scalability, reliability, elasticity, and system robustness.

2.2.1 Key Performance Indicators (KPIs)

This subsection describes the set of Key Performance Indicators (KPIs) defined for each use case. Each of them are linked to the list of requirements described in Section 2.3.

Table 2 presents each KPI, defined by the following columns:

- **KPI ID:** Link each requirement to a KPI. The naming refers to a particular use case KPI_UCX_Y, where X is the use case ID and Y is the requirement ID.
- **Description:** Brief description of the KPI.
- **Baseline:** Value that describes the current performance of the application.
- **Target:** Value that describes the expected improvement after using the NebulOuS platform.

Table 2: Key Performance Indicators

ID	Description	Baseline	Target
KPI_UC1.1_1	Time to generate notification about one of data pipeline components failure or abnormal operation: <1 minute	n/a	1 min
KPI_ UC1.1_2	Complete pipeline deployment time: < 5 minutes	n/a	5 min
KPI_ UC1.1_3	Number of images processed during single asset inspection: >300	0	300
KPI_ UC1.1_4	Response time for quality assessment model: <2s	n/a	2s
KPI_ UC1.1_5	Initial feedback on major damages: <5 min after inspection is completed.	1 day	5 min
KPI_ UC1.1_6	Reduction in data transferred to the cloud (defined as the ratio of the difference between the size of gathered data set and data sent to the cloud, to the size of the full data set): 30%	0 %	30 %

Funded by the European Union

KPI_ UC1.1_7	Time to restore normal operation after connectivity is restored (in case of temporarily lost connection): <1 minute	n/a	1 min
KPI_UC1.2_1	Vertical scaling: reducing mean time to detect by 50%	n/a	1 min
KPI_UC1.2_2	Horizontal scaling: decreasing the mean time to repair by 50%	n/a	1 min
KPI_UC1.2_3	Reducing data transfer from the edge to the cloud by 90%	n/a	1 s
KPI_UC1.2_4	Reduce number of necessary edge nodes by 10%	4	3
KPI_UC1.2_5	Reduce number of necessary cloud resources by 10%	2	1
KPI_UC2.1_1	Reducing data transfer from the edge to the cloud (NebulOuS KPI)	448Mbps of video sent to the cloud (estimated)	0Mbps of video sent to the cloud
KPI_UC2.1_2	Reduce number of necessary cloud resources (NebulOuS KPI)	n/a	Video streams processed on edge
KPI_UC2.1_3	Improve road usage patterns awareness (APP KPI)	Qualitative knowledge from managers	Historical quantitative data available
KPI_UC2.1_4	Less than 5 minutes to identify potential traffic issues (APP KPI)	15 min	5 min
KPI_UC2.2_1	Detect delays in delivery route and trigger a re-planification in less than 2 minutes (APP KPI)	15 min	2 min
KPI_UC2.2_2	Reduce average delivery route duration by 10% (APP KPI)	3h	2h40 min
KPI_UC2.2_3	Reduction of data transfer from the edge/fog to the cloud. Less than 5% of inconsistent data traces are sent (NebulOuS KPI)	n/a	<5%
KPI_UC2.2_4	Increase the number of tasks executed in fog in (NebulOuS KPI)	None	Vehicle routing should happen at fog.
KPI_UC2.2_5	NebulOuS takes less than a minute to allocate computational power for the application (NebulOuS KPI)	n/a	<1 min
KPI_UC3_1	Decrease cost of operation by 10%	1 day	10% cost reduction
KPI_UC3_2	Decrease time of completing a task by 20%	40 tasks	20% less time
KPI_UC3_3	Increase total completed tasks by 20%	2 hours	20% more completed tasks
KPI_UC4_1	Increase of total information processed digitally in disaster management	2 %	10 %
KPI_UC4_2	Increase Local data communication coverage during the early stages of disaster situations	5 %	60 %
KPI_UC4_3	The Acquisition of individual situational information (e.g., weather info), shall be automated and personnel effort shall be eliminated. Time to acquire data points.	15-60 mins	1 min
KPI_UC4_4	Increase the capacity of Average number of individual situational information points available (e.g., water level)	30	1000
KPI_UC4_5	Complementary information automatically derived in disaster management	0	100

2.2.2 Description of Key Requirements

Each requirement is defined by the following columns:

- **ID:** Column containing the ID for each requirement. The naming is composed of two parts: <ReqType>_<ReqID>
  - <ReqType>: Identifies the type of requirement:
    1. **F:** Functional requirement
    2. **NF:** Non-Functional requirement
  - <ReqID>: Is the number that identifies the requirement

- **Description:** Brief explanation of the requirement

- **Requirement Level:** indicates the level of priority of the requirement:
  - **M:** Mandatory
  - **R:** Recommended to be met
  - **O:** Optional

- **KPI ID:** Link each of the requirements to a KPI in section 2.1.1.

2.3 System Requirements

This section presents the list of the requirements for the NebulOuS platform, divided into functional (Table 3) and non-functional (Table 4) requirements.

2.3.1 Functional requirements

Table 3: Functional System Requirements

ID	Description	Req. Level	KPI ID
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows. It ensures workloads are distributed on the appropriate computing continuum (just edge, just cloud, edge + cloud)	M	KPI_ UC1.1_3, KPI_ UC1.1_4, KPI_ UC1.1_5, KPI_ UC1.1_6, KPI_UC1.2_3, KPI_UC1.2_4, KPI_UC1.2_5, KPI_ UC2.1_2, KPI_UC2.2_1, KPI_UC2.2_3, KPI_UC2.2_4, KPI_UC2.2_5, KPI_UC3_1, KPI_UC3_2, KPI_UC3_3, KPI_UC4_1, KPI_UC4_2, KPI_UC4_3, KPI_UC4_4, KPI_UC4_5
F_02	The NebulOuS platform offers a lightweight resilient event pub/sub mechanism with persistence, access control and consumer-group capacities to orchestrate communication between the different modules of the solution. The system shall provide events publish and subscribe based communication “bus” for the interaction of components	M/R	KPI_UC2.1_1, KPI_UC2.1_2, KPI_UC2.2_1, KPI_UC4_1, KPI_UC4_2 KPI_UC4_3
F_03	The system shall provide a graphic user interface for the administration of the deployed components	M	KPI_UC1.1_2, KPI_UC3_1, KPI_UC3_2, KPI_UC3_3, KPI_UC4_1, KPI_UC4_2

Funded by the European Union

F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components/containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	KPI_UC2.1_1, KPI_UC2.1_2, KPI_UC2.1_4, KPI_UC2.2_1, KPI_UC2.2_3, KPI_UC2.2_4, KPI_UC2.2_5, KPI_UC3_1, KPI_UC3_2, KPI_UC3_3, KPI_UC4_1, KPI_UC4_2, KPI_UC4_4
F_05	NebulOuS should constantly monitor the QoS required by the different application components (as specified by their SLA) and apply necessary actions to remediate QoS violations.	M	KPI_UC2.1_4, KPI_UC2.2_1, KPI_UC3_3, KPI_UC3_2, KPI_UC3_1
F_06	The NebulOuS platform provides infrastructure monitoring capabilities.	M	KPI_UC1.1_1
F_07	The NebulOuS platform provides application components lifecycle management for all components of the application regardless of where a component is deployed (cloud, edge, IoT device). It is able to start, stop components and react to monitoring alerts (if a component requires restarting for example).	M	KPI_UC1.1_1
F_08	The NebulOuS platform provides a deployment mechanism that makes it easy to deploy and manage data processing pipelines across various resource types (cloud/edge/fog).	M	KPI_UC1.1_2
F_09	The NebulOuS platform provides near-real-time communication between application components.	O	KPI_UC2.1_4, KPI_UC2.2_1
F_10	NebulOuS offers a mechanism to partition streaming analysis jobs between a variable number of workers (similar to Spark partitioning)	R	KPI_UC2.2_1, KPI_UC2.2_3,KPI_UC2.2_4
F_11	NebulOuS should offer a mechanism to deploy and invoke serverless computation functions.	R	KPI_UC2.2_3, KPI_UC2.2_4,KPI_UC2.2_5
F_12	Monitor and predict downtime of registered resources to maximize time/cost ratio. NebulOuS should be able to accommodate applications that prioritize resources with lower down time, in comparison with the opposite.	R	KPI_UC3_1, KPI_UC3_2
F_13	Handle paused or dropped tasks	O	KPI_UC3_1, KPI_UC3_2KPI_UC3_3
F_14	The system shall be able to be started and operated during run-time with minimal prior training	M	KPI_UC4_1, KPI_UC4_2

2.3.2 Non-functional requirements

Table 4: Non-functional System Requirements

ID	Description	Req. Level	KPI ID
NF_01	Fast decisions from NebulOuS, regarding the resources to allocate and fast execution of these decisions (fast communication, internal processing to take decisions etc.).	R/M	KPI_UC1.2_1, KPI_UC2.1_4 KPI_UC2.2_4, KPI_UC2.2_5, KPI_UC3_1, KPI_UC3_2, KPI_UC3_3, KPI_UC4_3

Funded by the European Union

NF_02	The NebulOuS platform will be able to operate even if available bandwidth between edge infrastructure and the cloud is limited or connection is temporarily lost.	R	KPI_UC1.1_7, KPI_UC3_1, KPI_UC3_2, KPI_UC3_3 KPI_UC4_1, KPI_UC4_2, KPI_UC4_5
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	M	KPI_UC2.1_2, KPI_UC2.2_3, KPI_UC2.2_4, KPI_UC3_1, KPI_UC3_2 KPI_UC3_3, KPI_UC4_1, KPI_UC4_2
NF_04	NebulOuS platform should be able to deploy applications in a Raspberry Pi 3 Model B+ or similar ARM architecture.	R	KPI_UC2.1_1, KPI_UC2.1_2,
NF_05	Data transferred between edge nodes and between edge nodes and clouds should be secured	M	KPI_UC2.1_1, KPI_UC2.2_4
NF_06	The system provides elasticity capabilities along with the cloud continuum so HW resources are provisioned or freed depending on the workload	M	KPI_UC2.2_4, KPI_UC2.2_5
NF_07	The system shall be able to cope with a sudden loss of connection to individual computational units	M	KPI_UC4_1, KPI_UC4_5
NF_08	The system shall be able to adapt the maximum load of the edge devices to the available power supply.	R	KPI_UC4_1, KPI_UC4_2
NF_09	The NebulOuS platform provides near-real-time processing of big data	M/R	KPI_UC2.1_4, KPI_UC2.2_1, KPI_UC2.2_2, KPI_UC2.2_3
NF_10	The NebulOuS platform will allow effective scaling of the cloud resources (in the edge/cloud computing).	R	KPI_UC1.1_3, KPI_UC1.1_5 KPI_UC1.2_1, KPI_UC1.2_2, KPI_UC1.2_3, KPI_UC2.1_2, KPI_UC2.2_4, KPI_UC2.2_5, KPI_UC3_1, KPI_UC3_2, KPI_UC3_3, KPI_UC3_5, KPI_UC4_1, KPI_UC4_4, KPI_UC4_5
NF_11	The NebulOuS platform ensures data privacy when transferring data from one node to another.	R	KPI_UC1.2_3, KPI_UC2.1_1, KPI_UC2.2_3, KPI_UC2.2_4, KPI_UC3_1, KPI_UC3_2, KPI_UC3_

Funded by the European Union

3. State-of-the-art analysis

3.1 Methodology

Most partners in the project have contributed with their expertise in the state-of-the-art analysis, structured in two parts:

1. First a broad information gathering resulting in an extensive concept map shared among the partners using the Mindomo[1] platform. The purpose is to establish a live infrastructure that can be used by the project team throughout the project duration and also for the more specific academic state-of-the-art in research to be analysed in the technical work packages. The results of this continuous analysis will be reported in the thematic deliverables later.
2. A more focused analysis of technologies and tools necessary for defining the NebulOuS architecture and starting the implementation. Over the decade of Cloud computing several tools, platforms, and technologies have been created and many of these could help NebulOuS implement a set of specific features and functionalities. Putting them in the concept map allowed us to discuss relations and overlap among the different technologies and identify the most central technologies NebulOuS should integrate or interface.

These technologies most relevant for the NebulOuS architecture are analysed and discussed in the following subsections. The placement of the technologies according to high-level topics is our best classification, even though some technologies are relevant to many topics, and there will inevitably be some overlaps among the subsections.

3.2 Analysis

3.2.1 Meta-Operating Systems for the Cloud Continuum

In this section, we aim to provide the current state-of-the-art with respect to the development of a Meta-OS for the Cloud Continuum. To this end, we first provide an overview of Kubernetes, which is the today's de-facto cloud service orchestrator and will play a central role in the NebulOuS platform. Subsequently, we shift our focus to other similar attempts to develop some form of a Meta-OS for the cloud, such as KubeSphere, ONEedge, Mesosphere DC/OS and Apache Mesos, as well as the recent EU initiative of SIMPL.

3.2.1.1 Enabling Technologies

Kubernetes

Kubernetes [1] is an open-source framework for managing containerized workloads and services, originally designed by Google and now maintained by the Cloud Native Computing Foundation (CNCF). It is considered the de facto standard for deploying and operating containerized applications and is frequently dubbed as the "OS of the Cloud". Moreover, with the increasing maturity of edge computing as a part of the cloud continuum, new Kubernetes distributions have been specifically developed for the edge.

As NebulOuS aims to develop a Meta-OS for the Cloud Computing Continuum, Kubernetes support comes as a natural choice and forms an important part of the project. A large ecosystem has been built around Kubernetes and the CNCF, with a plethora of tools that provide several functionalities that are supplementary to each other (such as security, observability/telemetry, service meshes, container networking, application modelling, etc.). The entire cloud-native landscape can be found in the corresponding CNCF website[2]. In this section, we aspire to provide a general overview of the internal workings of Kubernetes, as well as aspects that are relevant to the project such as application modelling, scheduling, authentication/authorization, monitoring, etc. We note that multi-cloud and edge support are considered to

[1] https://www.mindomo.com
[2] https://landscape.cncf.io/

be standalone research areas in the scope of the project and, as a result, Kubernetes support for multi-cloud and edge/IoT environments will be analysed as a part of the respective sections (sections 3.2.2 and 3.2.3).

Kubernetes offers several functionalities to manage containerized workloads. Some of the main ones include:

1. Service discovery and Load Balancing
2. Storage Orchestration
3. Automated rollouts and rollbacks
4. Automatic bin packing
5. Self-Healing
6. Secret and configuration management

A key concept in Kubernetes is that it follows a *declarative* approach to achieve its goals, meaning that given a description about the system's desired state (**cluster**), the framework tracks any drifts between the desired and current state, and applies all the necessary actions to ensure consistency between them.

There are two major logical entities in a Kubernetes cluster, which consist of various components.

1. **Control Plane**
2. **Worker Nodes**

Both entities support redundancy to ensure High-Availability (HA) requirements.

The **Control Plane** is where the core administrative components reside:

- **API Server**: An HTTP REST API Server, which exposes the Kubernetes API. It provides endpoints for all the resources managed by the framework. Communication between the framework's components is carried out through the API.
- **etcd**: A distributed key-value store, in which the desired and current state of all the cluster's resources are stored. When interacting with the **API-Server,** the etcd database backends the results.
- **Controller Manager:** The controller monitors the current state of the cluster via calls made to the API Server, and changes the current state to match the desired state described in the cluster's declarative configuration.
- **Scheduler**: Detects pending pods that have not been assigned to a node yet. Performs a series of Scheduling Policies to find the most suitable node for the pod to be assigned.
- **Cloud Controller Manager**: Enables linking a Kubernetes cluster with a cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with the cluster.

The worker nodes are machines that run Kubernetes workflows and to which pods are assigned. Their core components are:

- **Kubelet:** The primary agent responsible for running and monitoring the containers in a Pod. Any event derived from a container action is propagated to the API Server.
- **Kube-Proxy:** The component of Kubernetes that handles routing traffic for services within the cluster.

<u>Extensibility</u>

Kubernetes provides an interface for supporting user defined objects to extend the framework's capabilities. It allows the creation and registration of **Custom Resource Definitions** (CRDs) in the API Server, as well as monitoring through custom **Controllers** (namely **Operators**).

This seamless integration with the ecosystem has already enabled the establishment of a marketplace, which supports many well-known application solutions provided as **CRDs**, ready to be used and integrated into any Kubernetes cluster.

<u>Application Modelling Approach:</u>

The native approach of describing a Kubernetes object is via a **.yaml** file. The syntax is related to the Server's API endpoint with which this object is associated. After the application of the **.yaml** file, the cluster's desired state will be altered through the respective API call.

<u>Kustomize</u> provides post-processing template-like functionality to structure the native **.yaml** files.

Funded by
the European Union

Helm is a Kubernetes package manager which uses the **Chart** format**.** It enables an easier definition, installation, and upgrade of Kubernetes resources, by containing a collection of files that describe a related set relative to an application. Charts can be published through a public or private repository and installed directly to a Kubernetes cluster. They can and are ultimately translated to native .yaml files.
Worth Mentioning: *Open Application Model*, is an abstraction model describing an application as a **specific type**, for instance a Web Service.
The application's **.yaml** definition shall declare only the resources that are directly associated with this type of application, and not the extra resources needed by Kubernetes e.g. to configure the network rules, scale the app etc. This approach distinguishes between the definition of an application and the Kubernetes' specific resources that the devops team is responsible for. KubeVela implements the Open Application Model for Kubernetes, which provides **CRDs** for abstracting different kinds of well-established application types.

Authentication-Authorization:
Kubernetes provides out-of-the-box **RBAC** authorization for communicating with the API Server. It can be backed by:
1. Cluster's CA signed certificates
2. User Stores
3. File System
4. Token authentication for service accounts

It enables the creation of roles and role bindings for authorising defined roles with various Kubernetes resources.

Computable Resources:
Two types of computable resources supported by Kubernetes:
- **CPU:** Measured in *cpu* units. 1 CPU unit equals 1 physical CPU core or 1 virtual core. In a contended environment, the system will favour processes with higher requested CPUs with more CPU time.
- **Memory:** Measured in bytes.

When defining a container, a **request** and a **limit** parameter of these resources can be applied. The **scheduler** ensures that, for each resource type, the sum of the scheduled containers' requests is supported by the capacity of the node running the workload. When the container is executed, the **kubelet** agent passes the specified values to the container runtime interface **(CRI)**, which applies and enforces those policies based on a defined driver, like a **cgroup** driver in a linux environment. During execution, the kubelet constantly monitors and exposes these resources.

Extending Computable Resources:
CPU and Memory are the two resources supported natively by Kubernetes. Custom computable resources can be registered through the API Server and extend the same logic of request/limit while scheduling a container for deployment. Two kinds of custom resources can be set as quotas during scheduling:
- Node-Level extended resources
- Cluster-Level extended resources

Monitoring and Metrics:
Out of the box Kubernetes uses **cAdvisor (integrated into the Kubelet's binary).** A container monitoring agent, developed by Google, for exposing container metrics. The following metrics are exposed by cAdvisor:
- CPU
- Memory
- Disc
- Network

The de facto addon monitoring system for Kubernetes is **Prometheus**. An open-source application with a dimensional data model, flexible query language, efficient time series database and alerting approach. It can be stacked with Grafana to provide a more comprehensive depiction of the gathered data through dashboards.

Funded by the European Union

Liveness, Readiness and StartUp Probes:
Kubelet uses liveness probes to check the health of a container. Readiness probes indicate when a container is ready to accept traffic. Startup probes are used to know when a container application has started. All the above can be provided as custom commands in a container's configuration file, which is periodically triggered to monitor a container's status.

Automatic Optimization:
Kubernetes provides optimization using the following resources:

- ***Vertical Pod Autoscaler:***
  Scales up/down the CPU or Memory limit values declared in a pod. The **VPA Recommender** tracks the metrics history, OOM events, and the VPA deployment spec and suggests fair requests. The limits are raised/lowered based on the *limits-requests* ratio defined in the VPA spec. Scaling up requires recreation of the pods, so the **VPA Updater** is responsible for evicting these pods and applying the **VPA Recommender** deduced action, depending on the configuration update mode. The **VPA Admission Controller** intercepts the redeploying of the pod, injecting the recommended actions.

- ***Horizontal Pod Autoscaler:***
  Scales in/out the number of pods based on **CPU** and **Memory** utilisation. HPA natively depends on **Metrics Server** for monitoring and taking auto-scaling actions. The scaling decision can be based on any custom-provided metric, by installing appropriate resources in the Kubernetes cluster, like **Prometheus**, and applying actions regarding thresholds, events or on time schedules.

- ***Cluster Autoscaler:***
  In contrast with the former autoscalers, which operate at the pod level, the cluster autoscaler scales out/in by adjusting the available number of nodes. It takes action when:
  - A Pod is deemed un-schedulable, by the scheduler, because there are not enough nodes in the Pool.
  - There are nodes in the cluster that have been underutilised for an extended period of time and their pods can be placed on other existing nodes.

There are different implementations for the **Cluster Autoscaler** depending on the infrastructure provider and must be installed manually to the cluster, as the node creation process is not part of the native container orchestration.

Scheduling
Kubernetes scheduler selects a node for assigning a pod in a 2-step operation:

1. Filtering
2. Scoring

Filtering based on Node constraints is achieved by assigning specific labels or taints to the available nodes. Pods can be configured to tolerate these taints based on defined toleration conditions, making the node deployable only for specific Pods. A Pod can imperatively select a node by providing a specified matching label, through a NodeSelector. The node/pod affinity/anti-affinity filtering is even more expressive than the latter:

- *Node affinity* functions like the nodeSelector but is more expressive and allows for specifying soft rules.
- *Inter-pod affinity/anti-affinity* allows constraining of Pods against labels of other Pods, with regard to a node topology key, e.g. based on the name of the host, or a specific failure zone.

User-made scheduling configurations can be created and injected in various steps of the filtering and scoring process, allowing the definition of custom logic to the pod assignment operation.

Kubernetes and control theory
Kubernetes adopts the basic notions of control theory[3], i.e., the state has two sides to it: desired and actual, and the goal of active k8s components is to ensure that the desired state is met with the actual state (which means that the desired state must be stored, and the actual state must be monitored). This is achieved with control loops[4], a concept applied widely in

[3] https://en.wikipedia.org/wiki/Control_theory
[4] https://en.wikipedia.org/wiki/Control_loop

Funded by
the European Union

autonomic computing[5]. These active k8s components are called **controllers** – they observe changes to the desired and actual state and ensure the actual state meets the desired state, rinse and repeat (this is known as a closed control loop). There is also a related term of **operators** which are controllers responsible for managing domain-specific resources, such as specific application deployments. There is not a strictly technical difference between controllers and operators, but the general naming convention is that the core ones (available in vanilla k8s) are called controllers and the extra ones are called operators – this might be because most of the external ones deal with deployments of single applications (such as etcd, prometheus) and they act as replacements for human operators (hence the name). All in all, controllers and operators are just computer programs which observe and react to the state they were programmed to work with.

Kubernetes resources
Kubernetes manages a set of **resources** in this control loop fashion. Each resource is created with the description of the desired state (declaration[6], intent[7]). An example of a pre-existing resource kind is a **Deployment** which declares a deployment of a single (micro)service which consists of multiple replicas of a set of co-scheduled containers. The main definitional part of a Deployment is a **Pod** template. A Pod is another resource that is instantiated because of k8s controllers processing a Deployment. In fact, the plural – controllers – here is well-deserved because there is yet another intermediary resource – a **ReplicaSet** that gets created by the Deployments controller. Only after that, the ReplicaSets controllers create the necessary pods. The closed control loop ensures that there are as many (and not more) Pods and ReplicaSets as there should be. We can say that the three resource kinds given here are related and form a clear hierarchy: a Deployment relates to one or more ReplicaSets and a ReplicaSet to one or more Pods. Thus, the hierarchy is this:

- Deployment
    - ReplicaSet
        - Pod

However, any dependencies are possible between the resources, not necessarily strict hierarchies.

Kubernetes custom resource kinds
Kubernetes allows operators to extend the set of managed resource kinds by letting operators define Custom Resource Definitions (CRDs)[8] which themselves are a kind of managed resource, albeit controller-less – they only affect the operation of the API server. With new CRDs defined, appropriate custom resources will be accepted by the API server for management. However, without further work, there is no logic beyond schema validation that applies to these new resources. A new controller/operator must be run against the API server to be able to react to changes to the resources, monitor them and enact necessary amendments to ensure the desired state is met (see the previous paragraph on *Kubernetes and control theory*). The amendments could range from modifying other resources in k8s API (like the Deployments controller works) or changing the state of the world (like the kubelet does for Pods – it ensures that locally-scheduled Pods are running on the node).
(If CRDs are not enough [the linked page lists all their properties, including limitations], then there is a possibility to use the API aggregation layer[9] which means using the same principles as the k8s API but building the custom parts independently so that they offer all that is necessary at the API level. Using either approach does not preclude using the controller/operator pattern.)

Examples of interesting meta-level CRD-providing projects
Crossplane [2] provides meta CRDs that allow operators to define supported cloud providers and their services, letting users create native resources based on these cloud-provider services[10]. This works at a meta level and requires the actual providers to exist, such as the AWS provider, and credentials to it[11]. The k8s user can request e.g., an AWS RDS-backed

[5] https://en.wikipedia.org/wiki/Autonomic_computing
[6] https://en.wikipedia.org/wiki/Declarative_programming
[7] https://datatracker.ietf.org/doc/html/rfc9315
[8] https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
[9] https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation/
[10] https://doc.crds.dev/github.com/crossplane/crossplane@v1.10.1
[11] https://docs.crossplane.io/v1.10/getting-started/install-configure/

Funded by the European Union

PostgreSQL instance using the familiar k8s API, and the integration with the rest of the k8-based platform is already there[12].
KubeVela [3] ports the Open Application Model (OAM) to the k8s world[13]. The OAM [4] is a meta-model which allows to define components and applicable traits which can then be used to model the actual application. The components and traits have their implementations which are then realised by KubeVela. This allows for high level of reuse and abstraction with built-in automation for the actual realisation of „workload" resources such as Pods. Also note the traits pattern is analogous to the decorator pattern[14] from programming. One can utilise both KubeVela and Crossplane together – KubeVela's components or traits can be instantiated using Crossplane's resources.

Possible outcome: Kubernetes resources as the entry point to NebulOuS
NebulOuS could very well be driven using such CRDs in k8s. In fact, even if NebulOuS went with the basis of CAMEL [5] or TOSCA [6] in our DSL study, these could be remapped/remodelled to work in this model of thinking. Using k8s custom resources would have two major benefits: first and foremost, the basic working formula would be the same for existing k8s adopters – create resources and wait for magic to happen. Second, NebulOuS would be able to reuse any other well-developed resource to create an end-to-end path of instantiation control (vide KubeVela and Crossplane working together yet not referencing any other explicitly in their implementations). All the information necessary for NebulOuS to work could be defined in terms of collaborating custom resources, interpreted by NebulOuS controllers.

Multi-Cloud / Multi-Cluster Support:
Every node which is exposed via the internet can be registered in a Kubernetes Cluster, independently of which cloud provider it was created on, as long as it can communicate with the **API Server**. This approach assumes that the nodes are pre-configured in each cloud provider and then registered in the Kubernetes cluster, providing no intelligent automation and furthermore restricting the cluster autoscaling capabilities, as node creation is not feasible through different cloud providers by a single Kubernetes cluster. Cloud providers' specific features are also not leveraged in this kind of configuration.
A cluster configuration should be as little as possible coupled to a specific cloud provider's requirement, making the migration to another a trivial process and thus enabling multi-cloud support and avoiding vendor lock-in, while leveraging all the intelligence provided by Kubernetes. Catering to this need many technologies and platforms have arisen, providing management of Kubernetes clusters through different providers via a common interface, abstracting as many configurations as allowed to enable this kind of vendor interoperability.

Different distributions of Kubernetes:
The official Kubernetes distribution, described in the previous section is also known as "upstream Kubernetes" and abbreviated as *k8s. In addition to k8s, there exist many other Kubernetes distributions, providing different flavours with default configuration options, extending the native components or providing custom CRDs, in favour of supported conventions, like security, compatibility, monitoring, scaling etc. often backed by a maintainer's individual managing platform, which leverages and manages the aforementioned features.*
CNCF provides certification of Kubernetes software conformance, ensuring that every vendor's version of Kubernetes supports the required APIs, as do open-source community versions. For organisations using Kubernetes, conformance enables interoperability between all Kubernetes installations, providing flexibility between vendors.

In this context, other distributions have also been developed. One such instance is k3s, which is a certified distribution developed by Rancher and maintained by the CNCF. K3s is specifically developed for the edge, as such it is described in the Section 3.2.3 of this deliverable.

[12] https://docs.crossplane.io/v1.10/getting-started/provision-infrastructure/
[13] https://kubevela.io/docs/platform-engineers/oam/x-definition
[14] https://en.wikipedia.org/wiki/Decorator_pattern

Funded by the European Union

3.2.1.2 Available platforms

In this section, we provide an overview of the main platforms that have been developed as a Cloud OS/Meta-OS, analysing their approaches to grasp the key takings for own our platform's design and development. Particular focus is being placed in the SIMPL platform as an EU initiative to establish cloud-to-edge federations[15].

KubeSphere

a) **Optimization and Intelligence**

Kubesphere [7] does not offer any optimization functionality, as it mainly acts as a frontend to Kubernetes. Therefore, although it allows the creation of a new instance of an application, it will not perform any actions to lower the cost of implementing a processing topology.

b) **Application modelling approach**

Kubesphere offers the opportunity to install applications using both helm charts and application templates. Application templates allow for a visual construction of applications and can be retrieved from the Kubesphere application store. To create an application template, first a helm chart of the application should be uploaded to Kubesphere. Subsequently, the application can be deployed.

c) **Deployment orchestration**

Kubesphere builds on solid Kubernetes technology, and is also an open-source product. It features a modern user interface, but also allows a command line interface to kubectl. Kubesphere provides a visualization of the resource consumption allowing DevOps to plan cloud application usage. Kubesphere features integration with other Kubernetes-related technologies such as OpenPitrix [8] and KubeEdge [9], featuring an application store and Edge node management respectively.
Unfortunately, as a product-oriented towards cloud usage Kubesphere still leaves some to be desired. Namely, the installation process is not always as smooth as expected. Moreover, no optimization or scaling opportunities are exploited within the UI. In addition, Kubesphere requires 2 cores and 4GB of RAM per node[16], which may not always be available.

ONEedge

ONEedge [10] is an innovation project launched by OpenNebula in 2019 under the Horizon 2020 SME Instrument programme. It was successfully finished on the 31st of March 2022 and basically enhanced OpenNebula's cloud solution [11] with new features on multi-cloud computing and with an objective to offer a vendor-neutral platform for corporate users wishing to combine resources from public clouds and edge providers. This platform called Edgify claims to offer industrial readiness and maturity in a rapidly growing edge computing market. Although, Edgify is not branded as a meta-operating system for the cloud computing continuum, it is very relevant as it provides a vendor-neutral, application-agnostic and automated software-defined platform for managing resources in the cloud computing continuum. It enables the building of private edge computing platforms based on resources leased on demand in close proximity to the application users and devices. Edgify builds a hosted cloud, using a fully managed OpenNebula front-end, to automatically deploy as a service a distributed edge multi-cloud environment using resources from on-premises locations, edge and cloud

[15] https://digital-strategy.ec.europa.eu/en/news/simpl-cloud-edge-federations-and-data-spaces-made-simple
[16] https://kubesphere.io/docs/v3.3/quick-start/all-in-one-on-linux/

Funded by the European Union

resources from AWS[17], Google Cloud[18], Digital Ocean[19], Vultr[20] and Equinix[21]. It offers SLAs over the optimized configuration resources instances combined that may involve VMs, containers (as it encompasses centralized k8s clusters management) and microVMs for serverless deployments. Among its noteworthy use cases is the support of low latency edge applications for Telefonica's edge computing CORD (Central Office Re-architected as a Datacenter) and its implementation of a distributed Greengrass Edge Cloud leveraging the features of AWS IoT Greengrass[22]. In comparison to NebulOuS, it is focused only on the integration of infrastructural multi-cloud and edge resources, without the support for flexible preferences expression for the formulation of cloud continuum environments in an ad-hoc way. Moreover, it doesn't support the advanced and optimized management of application components and data that NebulOus envisions offering.

Mesosphere DC/OS and Apache Mesos

Finally, there is one platform worth mentioning despite the fact it is no longer being offered. At the time, it was a very promising solution for an OS-like approach to decentralised, cloud-native applications – the DC/OS (Distributed Cloud Operating System) Mesosphere Platform [12] supported by D2iQ and based on the Apache Mesos orchestrator, the "distributed systems kernel". [13] The end of Mesosphere DC/OS in October 2021 was caused by the enormous growth of the popularity of Mesos's competitor – Kubernetes. The D2iQ has decided that continuing on Mesos is a no-go and has switched to offering solely the D2iQ Enterprise Kubernetes Platform (DKP). [14] A notable benefit of using Mesos is the two-level scheduling which at first considers a pool of all resources available to all frameworks provided in the particular installation and then relies on the scheduling of the framework itself. Typical frameworks include Apache Hadoop, Apache Spark and Apache Kafka. Mesos is written in C++ and relies on Apache Zookeeper to provide high availability, and Apache Marathon to provide container orchestration. Mesos has not seen a release since November 2020 (with the planned release cycle of one every 3 months) due to the dropout of the major contributor – D2iQ. Similarly, Marathon has not seen a release since September 2019. DC/OS was enhancing the Mesos ecosystem with additional components, e.g. for advanced scheduling (Metronome) and a service catalogue for readily-installable applications.rlless

SIMPL - Smart Middleware Platform

SIMPL is the implementation of a procurement for developing a Smart Middleware Platform (SMP) that will enable cloud-to-edge federations and support interoperability between existing and new European data spaces [15].
The main challenges that the project addresses are:

- Interconnect data, applications, services and infrastructure across data spaces with secure, resilient, energy efficient, and accessible cloud-to-edge capabilities
- Make data more accessible to the public and enable processing through shared tools and assets, which are discovered and accessed via the SIMPL platform
- Apply a secured business model with the use of SLAs, licenses, contracts and invoicing to the available resources, that is constantly monitored and evaluated

A data space can be considered as an ecosystem of data sharing services within a particular industry sector. Data spaces consist of a set of distributed actors where each holds a piece of data in a decentralized fashion. SIMPL's architecture focuses primarily on data spaces, considering them as the main logical topology component. The overall architecture consists of the following key points [16]:

[17] https://aws.amazon.com/
[18] https://cloud.google.com/
[19] https://www.digitalocean.com/
[20] https://www.vultr.com/
[21] https://www.equinix.com/
[22] https://aws.amazon.com/greengrass/

Funded by the European Union

- The middleware provides common services on which data spaces can be built. SMP components span across four different actor types:
  - Data Provider
  - Infrastructure Provider
  - Application Provider
  - End User

Enabling them to securely and efficiently share and use their respective provided resources (data, application and infrastructure)

- Data Space specific services add the ontologies, standards, or quality enforcements needed by the data space
- Interoperability between data spaces enabled through SMP

Inside a data space, besides the aforementioned actors, also exist the following centralized components that enable the creation of the data-centered ecosystem [16]:

- Data, Infrastructure and Application catalogues which provide the cataloguing service for end users to discover shared resources
- Vocabulary providers which provide the definition of metadata representation, vocabularies and ontologies.
- Identity authorities manage the identities of the data space participants and provide proofs that other participants can use for authentication and authorization.

Through the federated ecosystem of data spaces, the following entities with their respective role can be distinguished:

- **Data providers** hold valuable data assets that they share with the SMP ecosystem
- **Infrastructure providers** allow users to run applications on their infrastructure (e.g., VMs, Containers, Serverless)
- **Application providers** share their applications and algorithms towards interested end users
- **End users** consume data, applications, and infrastructure
- **Data space governance** is the entity responsible for defining policies of each data space that all the actors should comply with in order to participate.

Application wise SIMPL is a **software agent** with a decentralized model, in which users are responsible for provisioning their own resources, as well as exposing them through the agent's interface. The agents communicate via peer-to-peer protocols and enable the exposure and utilization of each actor's resources to the ecosystem.
In this way SIMPL is conceived as a middleware, which exists between the Infrastructure and Data layers, but without tight coupling to them. [17].
To further distinguish SIMPL's functionalities, the platform is divided in 4 different layers [16]:

- **Data Service Layer:** Through this layer users can provide, discover and consume available data and applications through the accessed data spaces. Native tools will also ship pre-installed for common functions like visualization, anonymization and analytics**.** **SIMPL** supported data orchestration and distributed execution capabilities that allow actors to pool together data from different sources and manage partial sets of data across infrastructure providers when executing distributed applications.
  The combination of these capabilities allows end users to gather data from different providers and spread it over distributed infrastructure where data is fed into an application.
- **Infrastructure Layer:** Through this layer users can provide, discover and consume available infrastructure as well as Platform-as-a-Service services through the accessed data spaces**.**
  SIMPL provides infrastructure orchestration for automating the provisioning of the infrastructure services to enable the various infrastructure providers to interconnect and get exposed via a standard interface. Distributed execution is also supported, allowing users to deploy applications and execute computations closer to the data.
- **Administration Layer:** The administration layer vertically spans the data and infrastructure layer.
  It provides services that are required for the well-functioning of those layers, as well as the SMP as a whole, by enabling operations like federation management, security, networking, auditing, access control and trust, contracts, monitoring and reporting.
- **Governance Layer:** Finally, the governance layer will provide help and support features, regarding the overall platform.

Funded by
the European Union

Among the business requirements that are bound to be implemented by the **SIMPL** platform are the following, categorized by field of interest [18]:

Deployment:

- Authorized users shall be able to run the models/algorithms through the SMP in their chosen IT infrastructure
- The SMP shall enable the orchestration across multiple providers
- The SMP shall enable state-of-the-art data management between cloud and edge, enabling seamless ultra-fast data workload balancing between them.
- The SMP shall seamlessly accept applications/services made available via the marketplace

Scaling:

- The SMP shall automatically scale up & down specific services usage (e.g., storage, VM or HPC) without previous user requests up to the limits defined by the provider.

Infrastructure:

- Infrastructure is advertised in a data space by the SMP Agent and registered across the whole data space federation.
- The middleware shall offer an abstraction layer to give access to (edge, cloud) infrastructure services (computing & storage services)

SLAs:

- SLAs decided between actors should be monitored, evaluated and actions taken.
- Different performance tiers shall be suggested by the SMP, where each level provides a certain availability, data throughput, latency, turnaround time and other quality aspects as needed, which the infrastructure/application/data providers shall adhere to.

Monitoring:

- While the infrastructure is utilized by the end user, the consumption is monitored. This consumption includes metrics such as CPU usage, memory usage or compute time.
- The infrastructure usage, Performance monitoring, QoS metric, and Energy metrics components supply the end user with a view of its consumption.
- The middleware shall be able to monitor the SMP performance (both technical and environmental) and ensure performance and quality of service in the execution of applications across multiple cloud and edge providers.

3.2.2 Multi-cloud platforms

As multi-cloud operation is a main objective of our project and an end goal for the NebulOuS architecture, we devote this section to analyse the current state-of-the-art. We mainly focus on the current state-of-the-art platforms and tools that can be used to technically realize this goal, as well as NebulOuS' relation to previous relevant EU research projects (i.e., PrestoCloud).

Rancher

Rancher [19] is an open-source Kubernetes managing platform running on top of Kubernetes offering a multi-cloud, multi-cluster solution. It can be used to provision clusters and nodes with **k3s** or **RKE2**, on many different infrastructure providers, as well as manage cloud-hosted Kubernetes instances, like GKE, EKS etc. Clusters running **k8s** can also be registered in the Rancher platform. It was originally developed by Rancher, which was later acquired by SUSE.

<u>A brief analysis of the **RKE2 distribution**:</u>
RKE2 is one of the two Kubernetes distributions backing the Rancher platform. As a CNCF-certified distribution it provides all the features and intelligence that the upstream Kubernetes version supports, including YAML files, charts and

Funded by the European Union

templates for defining the workloads and the desired state, autoscaling options provided by native pod autoscalers, security policies etc.

This distribution comes in a single binary pack extending Kubernetes with custom features. Some of the extensions that are provided by RKE2 are:

- A helm-controller which auto detects helm charts in the file system and executes them.
- CoreDNS for providing DNS features to the cluster..
- DNSCache in node level
- Nginx-Ingress Controller
- Canal as the default CNI

RKE2 focuses mainly on security. It is configured to pass CIS Kubernetes benchmark scans with minimal effort and configuration, enables FIPS 140-2 compliance for cryptographic modules and uses **trivy** (a versatile security scanner) to scan regularly for common vulnerabilities and exposures on the images used in the cluster. This is achieved by providing the cluster with default **PodSecurity** and **NetworkSecurity** policies.

Rancher supports the deployment of each of these distributions according to the user's choice. Furthermore, the platform provides a powerful GUI for abstracting many of Kubernetes operations and workflows like deployments, services, ingresses, pod autoscalers, storageClasses, configMaps, persistentVolumes, installations of 3rd party apps through a marketplace with YAML and helm charts etc.
Ultimately the GUI provides a user-friendly way to utilise most of Kubernetes' features, while also giving the option to use kubectl commands for managing through a CLI, or by editing the resource's YAML files directly. Search and name-space filtering is also available through the GUI. Additionally, to manage Kubernetes' components, a default dashboard is provided for monitoring the resources used by each managed cluster in terms of nodes, pods, memory and CPU usage, events and health.

Multi-Cloud/Multi-Cluster support
The cluster management and provision are backed by Cluster API (CAPI), a Kubernetes project which provides infrastructure drivers for most cloud providers. Rancher is integrated natively with Cluster API and provides an abstraction of the implementation through its own CRDs.

Rancher can manage the lifecycle of a cluster as well as the nodes that comprise it. It uses CRDs for defining the cluster itself (**Cluster CRD**), as well as resources for managing the nodes, namely **machine** and **machine pools**.
These operations, backed up by Cluster API internal implementation, leverage the Kubernetes controlling element, ensuring that a desired set of Machines will always run on a registered cluster. Rancher comes with many predefined drivers for deploying a hosted cluster to a cloud provider. The most popular ones consist of:

- Amazon EKS
- Azure AKS
- Google GKE
- Linode LKE
- Alibaba ACK

Users can also provide their own driver to extend those natively supported by Rancher, by implementing their own cluster interface.
As mentioned above the Machine CRDs are used to provision nodes in a cloud provider. Rancher natively provides infrastructure drivers for most cloud or on-prem instances machines (bare-metal and VMs) like:

- Amazon EC2
- Azure
- Google
- DigitalOcean
- Linode
- OpenStack
- vSphere

Each node corresponds to a Machine object and is managed internally by Cluster API, which is in turn abstracted by Rancher. Machine Pools can be configured for managing nodes (machines) with the same characteristics. The GUI provides an option to scale those Machine Pools as needed, by providing or destroying nodes automatically in the chosen infrastructure provider. Cloud provider-specific credentials can be stored in Rancher to conveniently reuse across different cluster configurations.

Cluster Autoscaling through Rancher's autoscaler.

As mentioned in the upstream Kubernetes section a cluster autoscaler is supported additionally to the native PodAutoscalers. In the case of Rancher, the cluster autoscaler implementation can be deployed to automatically scale in/out the Machine Pool resources, in which the number of available nodes is defined.
Due to the Node drivers supported, the platform is aware of how to create and destroy nodes in its managed clusters, either be a Cloud Provider or an On-Premise Hypervisor.
The autoscaler monitors the node's capacity and scales in/out based on the utilisation of resources like memory, CPU, GPU and max pods.

Rancher native extensions and integrations with other Kubernetes projects:
Some specific tools are also integrated directly with Rancher, enabling management through the UI and installation from the provided application marketplace. Following are the main concerns of what problems these tools try to solve:

Monitoring:
A monitoring stack is provided and can be deployed to any cluster, which consists of Prometheus for scraping and storing metrics, Alert Manager for notifying on events based on those metrics and Grafana for visualising and querying the metrics.

Multi-cluster deployments with GitOps through Rancher's Fleet:
Fleet is an open-source Kubernetes project developed by Rancher. It enables the deployment of applications to multiple clusters from a single management point. Rancher's multi-cluster capabilities integrate seamlessly with Fleet, combining the multi-cluster management interface from Rancher server with the multi-cluster deployment features of Fleet.
Users can leverage continuous delivery to deploy applications to the Kubernetes clusters without any manual operation by following GitOps practices. Meaning that the only source of truth for an application's state is a tracked Git repository. Fleet integrates with Git, tracks change and automatically rolls any updates that occur in the remote repository to the managed clusters. Furthermore, managed clusters can be grouped and tagged for fine-grained selection during the deployment of workloads.

Security:
Rancher extends the RBAC functionality of Kubernetes to enforce control policies to the platform's CRDs. Once the user logs in to Rancher, their access rights within the system are determined by *global permissions*, as well as namespaces-specific *project roles*. This feature enables a federated Access Control policy, spanning through multiple clusters and different cluster-specific namespaces.

Kubernetes enabled VM provisioning with Harvester:
Harvester [20] is an open-source hyper-converged infrastructure (HCI) software built on Kubernetes. Harvester is installed on bare metal servers and provides integrated virtualization and distributed storage capabilities. Although Harvester operates using Kubernetes, this dependency is concealed in its functionality, requiring zero Kubernetes knowledge from the user.
Harvester uses a declarative way to define and provision VMs. It is internally powered by **KubeVirt**, [21], a CNCF project that provides virtual machine templating and provisioning through the Kubernetes API. It integrates seamlessly with Rancher and once installed, it can be operated through Rancher's interface. The combination of the two platforms can enable on-premises cluster provisioning to VMs created and managed by Harvester, while leveraging all Rancher's capabilities like autoscaling the VM instances or managing through Machine Pools.

Funded by the European Union

PrEstoCloud

PrEstoCloud [22] is a Horizon 2020 project, which aimed as well to take advantage of combined cloud and edge resources, using a model-driven engineering approach. However, Nebulous significantly surpasses Prestocloud in terms of promised functionality.
Firstly, Nebulous envisages a meticulous MultiCriteria Decision Making (MCDM) approach to select resources and advertise resources through fog service brokerage while Prestocloud targeted the selection of processing resources based on considering cost, latency and bias through a weighted linear function. While Prestocloud only considered the definition of resource constraints, Nebulous also adds regulatory, security and privacy constraints within its scope. The targeted MCDM approach to consider requirements and preferences related to the discovery of services and resources, coupled with the option to define advanced constraints, greatly enhances the programmability of applications deployed through the new Nebulous platform. By contrast, the constraints of Prestocloud were only coarsely defined, and the placement criteria were evaluated using a much simpler approach.

Relating to the utilities offered to the application by the platform, the encapsulation of its SLAs in smart contracts is a feature uniquely targeted by Nebulous. To compare, PrEstoCloud offered SLA support, using SLAs expressed in JSON format and transmitted to interested components using conventional encryption techniques (at best).
Moreover, concerning the detection of SLA violations and the automatic application placement reconfiguration, Nebulous will employ distributed complex event processing, capable of being executed at the edge. This is a major improvement over Prestocloud which offered the equivalent functionality through the SDM component which could only process events in a centralized fashion. To aid the detection of SLAs (and their prediction), Nebulous will utilize AI algorithms. In Prestocloud, the detection of SLAs considered the use only of current metric values, and the prediction of workload-related metrics (but not prediction of SLAs).

Nebulous will also develop the ORDGE mechanism which in turn entails the development of semantic models (able to capture higher-level organizational policies) and advanced semantic processing (able to detect contradiction or subsumption between application provisioning policies or preferences and the compliance of lower-level application provisioning requirements with higher-level organizational policies (Task 3.2). In Prestocloud, the definition and usage of semantic technologies was significantly more limited. Moreover, the usage of digital twins will be explored in Nebulous as part of Task 4.6 whereas Prestocloud did not focus on this area.
Finally, significant additions which are only featured by Nebulous include the addition of the Scheduling Abstraction Layer (SAL) which will be extended to consider extra heterogeneous resources, transcription of an application description from textual form to a visual representation, and fault-tolerant monitoring probes.

3.2.3 Edge/IoT Platforms

This section focuses on the Edge/IoT side of the continuum, presenting relevant platforms. In this respect, we provide an overview of the technical state-of-the-art, presenting both the open-source tools (mostly related to the Kubernetes ecosystem) and available commercial platforms.

When considering the edge, we mainly refer to the transition from the cloud to the deployment of resources and services closer to the end-users. The overall goal is to bring computing resources and services closer to the edge devices, either by deploying these resources in regional data centres or by deploying them directly at the edge. Consequently, and by design, the edge infrastructure is extremely distributed to cover all the users that are by nature distributed over the coverage area. This complex distribution mandates the creation of edge platforms to automate the management of these distributed resources. edge platforms must be designed to handle the challenges of managing and orchestrating computing resources in a highly distributed environment. These challenges include managing and allocating resources in real time, handling network latencies and failures, and ensuring security and data privacy.

StarlingX [23] is one of these Edge platforms that provides a comprehensive set of tools and components for deploying, managing, and scaling edge computing resources. It is designed to attach resources from the different cloud providers and private providers using OpenStack, and Kubernetes cluster. However, the autoscaling decisions is limited to what is

Funded by
the European Union

provided using Kubernetes and OpenStack. As a result, StarlingX only supports application-level autoscaling by for example assigning more pods to the replica set and can't decide to deploy a new VM based on a surge in traffic.

A similar approach was taken by **K0s** [24]**,** a Kubernetes distribution designed for simplicity and ease of use. It aims to provide a fast and lightweight solution for deploying and managing containers in both cloud and edge computing environments. One of the main goals achieved by K0s is lightness such that it can run on very limited nodes (as low as 1 CPU core and 1GB of RAM) like those expected to be running on the edge. It uses Prometheus as its default monitoring system to collect monitoring logs about resource utilization, application performance, and network traffic. However, the node addition to the cluster is done manually. In addition, K0s is considered an Immature project and may still be undergoing changes and improvements, which could result in bugs or other issues.

OpenYurt [25] is a CNCF-hosted project that provides an extension for Kubernetes to support edge computing. Similar to K0S, OpenYurt is considered an early project (created in 2020). However, It extends the functionality of Kubernetes to provide an edge cluster management solution that can run on various edge devices and enable seamless integration with centralized Kubernetes clusters. It categorizes the nodes in different "nodepool" and it's capable of migrating these nodes between these to give an impression of resource pool autoscaling, but in reality, no new nodes are added to the cluster.

Portainer [26] is lightweight and open-source container management platform. It aims at accelerating container technology adoption by reducing operational complexity and addressing the security challenges of running containers. Portainer defines itself as a universal platform based on the fact that it supports multi-cloud usage (including AWS, Azure, GCP, Linode, etc.), It also supports on-premises resources including edge nodes (supports ArmV7 and Armv8). The platform can coexist with several container management orchestrators like Docker Swarm, Nomad, and Kubernetes. As a result, ensuring a deployment that cannot be locked into a single technology or vendor.

However, since Portainer lacks a lot of important features, it heavily depends on other cloud technologies to implement these features, for example it depends completely on Kubernetes and Nomad to manage containers autoscaling and consequently a user of Portainer will be bounded to use only one of this technology as the autoscaling rules are not transferable from one technology to another. Consequently, we can argue the claim of independence from a single technology.

In a similar fashion to that of Portainer, Mirantis Kubernetes Engine (MKE) is a container orchestration engine that similarly aims at reducing container management complexity by creating a platform for deploying containers at scale [27]. It supports Kubernetes as well as Docker Swarm, provides a full stack of open-source technologies like calcio and nginx. However, unlike Portainer, MKE attempts at hiding the underlying infrastructure and keeping the focus on the application. It reserves the infrastructure management to another Mirantis software's like Lens for the monitoring of the infrastructure and ZerOps for automatic reconfiguration and autoscaling for the applications containers. However, this management is heavily dependent on containers and cannot control physical resources.

In NebulOuS we aim at creating a holistic solution for applications spanning all the layers of on-demand computing, we plan on having this solution not only defining rules for autoscaling the containers of the application, but we will extend that with autoscaling the infrastructure based on boundaries defined by the application developers and system administrators. Our solution will provide support for all the commonly used virtualization techniques, and will provide translators to deploy the same application definition in all the supported infrastructures and technologies.

KubeEdge

KubeEdge [9] is a Kubernetes-based open-source software aiming to facilitate the orchestration of applications using the edge, and the management of devices at the edge.
The edge part of KubeEdge allows the execution of business logic at the edge, and offers - through EdgeMesh - simple service discovery and traffic proxying functions. Device properties are maintained in a database, and can be set or updated dynamically based on the (desired or current) status of the device. The cloud part of KubeEdge is mainly oriented towards the administration of edge devices and applications on the edge. As KubeEdge is based on Kubernetes, it can benefit from the monitoring and management tools which have been developed for this platform.

Regarding the deployment of applications themselves, KubeEdge offers an MQTT broker which may be used by the application in addition to HTTP calls. Applications need to be containerized in order to benefit from KubeEdge, which then allows the DevOps to select the execution location (edge or cloud). Examples of KubeEdge applications can be found in the GitHub repository of the project[23].
However, it should be mentioned that the setup of KubeEdge is not trivial, and the documentation could certainly be improved. Moreover, integration with the serverless paradigm is not pursued, missing an opportunity to be strategically placed in the market.

K3s

K3s is a certified Kubernetes distribution developed by Rancher and now maintained by CNCF. k3s focuses on limiting the resources needed to run Kubernetes on a single node. This distribution is a refined version of k8s, which has removed a lot of "bloat" code, while maintaining the core functions intact, allowing the limit of resources while providing high availability, scalability, security, and other full-blown[24] k8s benefits.

Depending on the version, the single binary file can range from under 40MB to 100MB in size and run on less than 512MB of RAM. Taking into consideration the redundancy needed for HA, a kubernetes cluster can be resource intensive. In many cases, especially when deploying at the edge, the k8s resource requirements can be prohibitive. This makes k3s a great choice for the edge, without ruling out the use in cloud or on-prem environments.

This differentiation of resource usage comes from the fact that k3s contains fewer dependencies, cloud provider integrations, add-ons, and other components that are not absolutely necessary for installing and running Kubernetes. The conventions offered are a predefined CNI (**Flannel**[25]) for pod networking (L2 overlaid on a L3 network) and an embedded SqlLite database replacing the default etcd, containerd as CRI, Traefik Proxy as ingress controller and manifests to install non-pre-packaged core components. Of course, these components can be replaced with any other Kubernetes-backed technology and are only provided for convention.
The main advantages of k3s are:

- Lightweight: All of the components run together as a single process, packed in a single binary.
- Open Source
- Certified Kubernetes Distribution, ensuring compatibility and interoperability with different vendors.
- Easier and faster installation and deployment.
- Supports low-resource environments
- Flexibility in database choice, through Kine, an etcdshim translating the native Kubernetes API etcd calls to different database interfaces

Disadvantages:

- Does not ship with a distributed database. So, to ensure HA, an external distributed DB must be configured to use with the k3s servers.
- A lot of components do not come out of the box but need to be imported. This can lead to a missing chain in dependencies, when trying to install an external application that depends on a native k8s component.

As an official Kubernetes distribution all the features and intelligence described in the upstream Kubernetes section are supported.

AWS IoT Greengrass

AWS IoT Greengrass [28] is an open-source Internet of Things (IoT) edge runtime for developing complex IoT applications that run between IoT devices the edge and the cloud. The project offers a set of reusable components to build these

[23] https://github.com/kubeedge/kubeedge
[24] https://www.cloudzero.com/blog/kubernetes-architecture
[25] https://github.com/flannel-io/flannel

Funded by the European Union

applications: databases (InfluxDB, Postgresql), communication protocols (Bluetooth, LoRaWAN, MQTT, Modbus TCP), Node-Red programming environment, etc… It also offers the possibility to execute Lambda functions that make use of device resources (sensors, actuators, serial-ports, etc…). Finally, it offers virtualization capabilities through the deployment and management of Docker containers. All these components can be executed on Greengrass core devices with ARM or x86 architectures running Windows and Linux-based distributions.
Greengrass core devices can work as hubs for IoT devices running the AWS IoT Devices SDK. Using secured local communications, the IoT devices can transmit data to the core devices for their analysis, and storage. In turn, Greengrass core devices can trigger actions to be executed by these IoT devices. Thus, working totally isolated from the cloud. On top of that, policies for processing data locally and/or sending it to the AWS Cloud services can be defined. AWS IoT Cloud Service is a paid cloud service that allows users to manage their IoT Greengrass core devices and the applications they are running.

Azure IoT Edge
Similarly, Microsoft leads the development of the Azure IoT Edge [29], a device-focused open-source runtime that enables to deploy and monitor IoT application logic using Docker containers called IoT Edge modules. Some popular software solutions are already containerized to run on top of the runtime (SQLserver, Redis, Node-Red, etc…) and the user can bring its own application containers. Azure IoT Edge runtime can run on Windows and Linux machines. Also, a paid web control panel (Azure IoT Central) is provided to allow management: to prepare the workload to be executed by devices, send it to the devices and monitor its execution.

3.2.4 Serverless
Serverless computing is a seemingly oxymoronic term that describes the use of computing resources without managing details related to the lower levels of the stack, such as the operating system and actual servers (be them virtual or physical) [30], [31]. The use of serverless computing is growing and is predicted to finally become the dominant deployment model. Thus, it is vital for a project like NebulOuS to consider support for serverless computing offers.

The most common application of serverless computing is the Function-as-a-Service (FaaS) approach to computing where the programmer (FaaS's end user) is presented with an interface to deploy a piece of software (a function) without the need to worry about the details of the execution environment – no resource reservation nor preplanning is needed (although service providers might offer discounts for upfront resource reservations). This substantially simplifies the deployment cycle for developers.

A more general serverless computing definition includes also other models of services, which nowadays include: Backend-as-a-Service (BaaS), Platform-as-a-Service (PaaS) and Container-as-a-Service (CaaS). By the general definition, also the Software-as-a-Service (SaaS) fits the bill but it is not common to see it mentioned as a serverless solution, possibly because the serverless term is targeted to programmers as end users, not general software users as is the case in SaaS.

PaaS can be viewed as an early iteration of FaaS and indeed the two approaches share a common concept of an interface to deploy a piece of code. The distinction between the two is that the FaaS model goes further on abstracting away the underlying platform, providing multiple event triggers on when the function in FaaS would be run vs always-running PaaS applications. This allows FaaS deployments to scale to zero and avoid all costs related to the deployed service as long as no triggers are hit. Typical triggers include: HTTP-based requests, messages in a message queue or time-based schedule. One downside of FaaS is that it does not follow a typical programming model, instead the function is run in a service-provider-specific context. This model also affects the way that data is managed, enforcing cloud-native best practices: all data must be stored externally to the application (function), in a separate, dedicated, persistent storage service.

BaaS solutions complement FaaS in providing these services that are not normally feasible with FaaS. This way the user is still saved from managing the servers, operating systems and specialised software on them. Instead, the user is able to request API endpoints to be available and to utilise the service provider's storage services. Examples of BaaS offerings include SQL and NoSQL databases, message queues, memory stores and alike.

Funded by the European Union

Finally, serverless CaaS is an emerging trend that allows for more flexibility and less lock-in while reaping the benefits of FaaS. Serverless take at CaaS tries to balance out the advantages and disadvantages of non-serverless Infrastructure-as-a-Service (IaaS) cloud offerings (which allow a lot of flexibility) and FaaS which, in comparison, has severe limitations as to the way the software is deployed. By abstracting away the container orchestration and all the layers below it, serverless CaaS allows the programmer to focus on delivering the runtime in addition to the application without worrying about the underlying servers and their supporting software. It is worth noting here that CaaS has two meanings, with materials before near the end of 2017 using the term "CaaS" only to mean non-serverless Kubernetes-as-a-Service offerings which manage the control plane of Kubernetes but still base themselves on IaaS and server management, without the abstraction necessary for the serverless approach. The latest serverless CaaS offerings of the three major cloud providers were made available in 2021 and keep evolving.

By the very notion of serverless, it is not a primary deployment option considered on-premises since the majority of benefits come from yielding control to the service provider which is the local operator in case of an on-premise deployment. Despite that, on-premise serverless solutions are growing in popularity. One reason is that, especially considering free and open source solutions, serverless might be one of the options for smaller service providers which rely on such software to deliver their services. Another reason is that programmers liked the new model of computing and are looking for solutions which would deliver a similar experience regardless of whether the target is in the cloud or on-premises. Furthermore, the situation is augmented by the seemingly synergic interaction between serverless and containerisation trends.

The already-established Kubernetes ecosystem offers multiple solutions for FaaS (which is more like CaaS due to the underlying container technology), the most prominent one being Knative [32]. Service providers are also offering hybrid solutions which involve FaaS on-premises, major example being Google Anthos [33] which offers an on-premise FaaS with Knative. Apart from Knative, other popular Kubernetes-native solutions include OpenFaaS [34], Fission [35] and Camel-K [36]. A notable Kubernetes-compatible, yet not Kubernetes-dependent FaaS solution is Apache OpenWhisk [37]. Due to the widespread adoption among different environments, Knative looks most promising as a future-proof solution but all mentioned solutions have solid user bases based on github statistics. There are also some newer solutions which do not directly name themselves serverless or FaaS, yet they share the basic characteristics and are thus counted in as serverless solutions in the cloud-native solutions landscape by CNCF. A notable example of such solution is the combination of KEDA (Kubernetes-based Event Driven Autoscaling) [38] and dapr (a portable, event-driven runtime for building distributed applications across cloud and edge) [39]. The OpenFunction project [40], related to Kubesphere mentioned earlier in this SotA, combines Knative, KEDA and dapr (among others) to deliver an opinionated FaaS stack.[41]

3.2.5 Resource Management

Resource management in computer science refers to the process of allocating, monitoring, and controlling the underlying resources. These resources can consist of the computing power calculated as the number of cores and the size of the memory, the storage capacity measured by the disk space, and network resources measured by the assigned bandwidth. Typically, the management of resources is done through a combination of manual and automated processes. However, with the current insurgence of automation, manual management is falling in popularity for more advanced automated tools that provide a full cycle of allocating, monitoring, detecting, decision-making, and finally updating.

In cloud computing, although we can practically consider the attainability of unlimited resources, the management of the provisioned resources remains a topic of extreme importance. This can be attributed to the pay-as-you-go model which allows one to pay only for the used resources as function of the usage time. As a result, an optimized approach in resource management should enable system administrators to ensure the operation of the application within an agreeable performance while keeping the cloud provider bill as low as possible.

Funded by
the European Union

Consequently, in cloud computing, resource management is a cost/performance trade-off that attempts to reduce the cost of operation while maintaining the performance calculated as function of metrics decided by the application developer and the Quality-of-Experience (QoE) of the users.

In contrast, for edge computing and likely so for on-premises infrastructures, resource management aims at solving another optimization problem. For instance, the resources are owned and not rented which abolishes the cost from the optimization problem. However, the resources are limited which dictactes that resources are managed effectively through scheduling and consolidation to achieve the maximum resource utilization and, therefore, the highest possible performance within such resource limitation.

Elements of Resource Management

As an initial conclusion, we can see that resource management is not a one-size-fits-all type of a problem, but in contrast, an optimization problem derived from the underlying infrastructure and the application requirements. In the state of the art of resource management, authors have chosen to target different elements of resource management, mentioning some:

(1) **Infrastructure type:** several infrastructures can fall under the definition of on-demand computing, in the state of the art we can see resource management was addressed based on the type of the infrastructure like cloud computing [42] [43], edge computing [44], fog computing [45], grid computing [46], etc…
(2) **Application model:** Application running on top of the on-demand computing infrastructures cover a wide spectrum of functionality, design, and requirements [47]. Consequently, resource management mechanisms should be adjusted based on the application model. For example, client-server applications should be managed differently from microservices [48]. Also, in terms of requirement, latency-sensitive applications [49] should be deployed differently to stream processing applications [50] and so on. Here we can identify a platform-centric approach based on optimizing resource utilization, and an application-centric approach based on optimizing the performance of the application.
(3) **Virtualization techniques:** All of the on-demand computing infrastructures rely on a layer of virtualization on top of which applications are deployed. The mainstream virtualization techniques currently in use are virtual machines and containers. The virtualization technique in use can influence the resource management. For instance, VM resource management takes an interest in how the deployment time of VMs can be optimized [51]. On the contrary, in a container-based setting, the deployment time is negligible and consequently it's overlooked.
(4) **Workload type:** the type of expected workload influences the choice of resource management techniques. In the literature, some researchers consider an application request as an encapsulated task that must execute without previously provisioned resources [52]. In other cases, the request is considered a set of instructions that will be executed in an already deployed application resource [53]. This distinction drives two very different styles for resource management.
(5) **Optimization objective:** stemming from all the previous approaches, the resource management objective can cover different topics like cost [54], resource utilization [55], energy consumption [56], deployment time [57], latency minimization [49], network usage [58], etc.

In addition to the different elements in resource management, we can identify two distinct layers of resource management. The first layer comprises the management of the underlying resources, meaning the actual physical resources in use while the second layer comprises the management of the application components by assigning the proper resources to them. In the rest of this document, we will refer to the first layer as the resource pool management while the second will be referred to as application resource management.

Application resource management:

The management of applications ensures their QoS requirements. Application resource management includes the treatment of the workloads and creating the routes for these workloads, the placement and autoscaling of the components, health checks on the applications state, and handling application provider requests to change the state of the deployment.

To better understand this type of resource management, we can consider a general application model in Figure 1. In this model the application has 3 services that can be replicated/non-replicated and monolithic/distributed, the services are deployed on several architectures spanning several layers and over different machines. By design each service has a Broker (can also be referred to as a load balancer) that acts as the point of contact for the service and redirects users' requests to the correct destinations of the service.

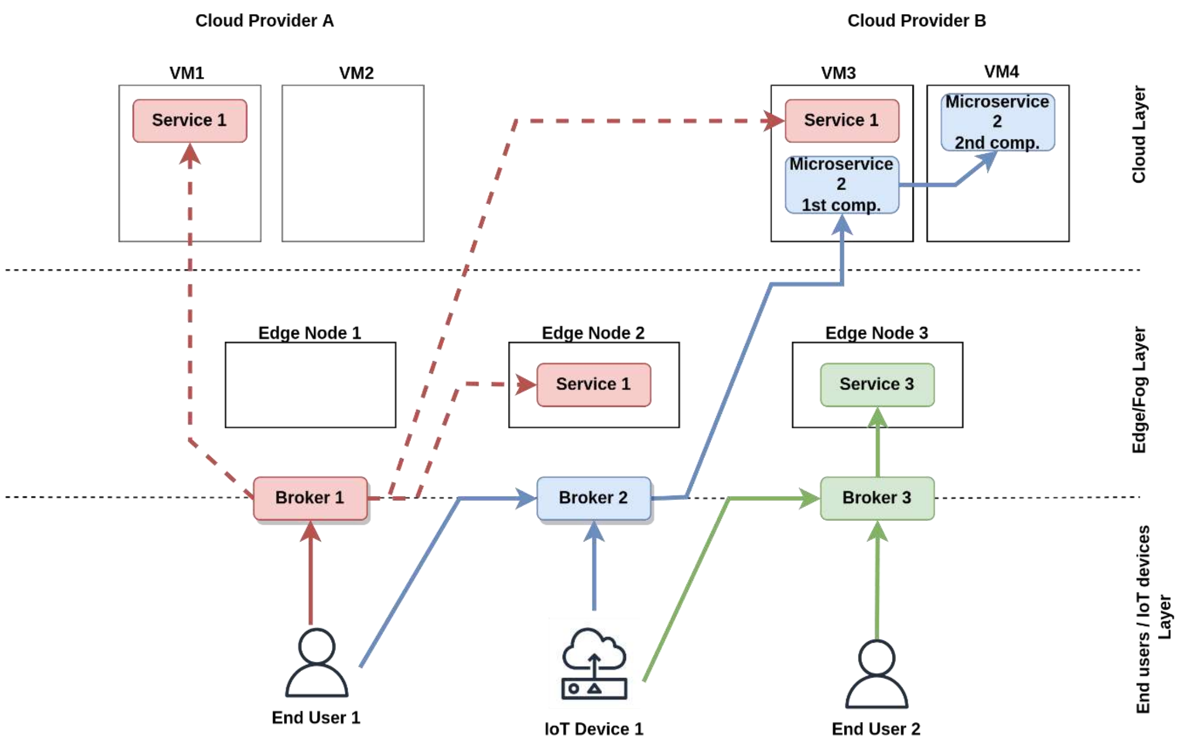

Figure 1: Application Model

Based on this Model we can define three levels of application resource management:

- **Workload Routing**: The routing of the End users' request takes place at the broker of each service and constitute the first optimization decision to be considered. In the case of broker 1, one of the replicas should be selected to compute the request based on some criteria like availability, lowest round trip time, and request priority.
  In the state of the art, a number of researchers have targeted the optimization problem of task offloading, in other meaning, deciding on which layer the request should be processed. This concept is popular in Mobile Cloud Computing (MCC), where the application decides whether to execute on the user device or to be offloaded to a cloud data center [59]. With the introduction of Mobile Edge Computing (MEC), this concept gained similar traction, however with access to two different types of resources, the edge resources and the cloud data centers [60]. The major difference between the several papers remains the objective of the optimization problem, Vu et al. offload services in fog radio access networks to optimize the energy consumption and offload latency [56]. Yousefpour et al. Propose a delay-minimizing offloading policy for fog nodes to reduce service delay for the IoT nodes [52]. Mukherjee et al. reduce the overall task completion latency by solving a quadratic programming constraint optimization problem [61].

  In the other hand, other researchers took the approach of selecting not only the layer where the request will be processed but the exact node and service. This approach provides a finer granularity for heterogeneous infrastructures like that of fog computing and edge computing since the nodes can be unique in terms of their resources, their latency to end users, even their energy consumption.
  Consequently, the edge and fog layers can't be treated as homogeneous layers. In the literature, a number of routing techniques have been suggested to optimize latency, energy consumption, and backhaul network traffic while maintaining a balanced load [62] [63]. Kadhim et al. presented an energy-efficient multicast routing protocol

based on SDN and fog computing [64]. Puthal et al. propose a secure and sustainable load balancing technique which aims to distribute the load to the less-loaded edge data centers [65]. Similarly, Beraldi et al. propose a cooperative load balancing technique to distribute the load over different edge data centers to reduce the blocking probability and the task's overall time [66]. Kapsalis et al. propose a fog-aware publish-subscribe system which aims to deliver messages to the best possible node according to a combination of network latency, resource utilization and battery state [67]. Finally, Proxy-mity addresses the necessary trade-off between reducing the user-to-replica latencies and balancing the load equally across service replicas [53].

- **Scheduling and Placement**: Although routing mechanisms can help in optimizing the application resources, however, routing is limited by the location of these resources. For instance, in the case of latency-sensitive applications, where all the resources are located far from the origin of traffic the problem exceeds the capabilities or routing mechanisms and falls under the optimization problem of how to correctly place the resources. This subject was heavily investigated in the literature, starting from simple mechanisms that control in which layer the application services will be deployed, to more advanced mechanisms that more precisely predict the fittest node to be allocated the application service. The NebulOuS platform targets applications that are expected to span several cloud providers, edge nodes, fog nodes, al, all the way to on-premises. As a result, we limit the scope of this optimization problem to algorithms that first consider this continuum of resources and second algorithms that do not consider the different layers as homogeneous.
  Starting from this scope, we can start categorizing the work based on the application. Data-intensive application placement algorithms optimized the response time [68]–[71], while monolithic applications focused predominantly on network usage [58], [72], [73]. In contrast, service-oriented applications aimed at optimizing an array of metrics: *Li et al.* [74] present a replica placement algorithm to enhance data availability, *Jemaa et al.* [55] present mechanisms to optimize resource utilization, to prevent services overload, and to avoid violation of Service Level Agreement (SLA) requirements. A range of other papers attempted the proximity-awareness and latency sensitivity problem [47], [72]–[75]. The proximity awareness problem reflects the intention of locating the application resources as closely as possible from the sources of traffic, which by design requires one to monitor the traffic and then include this input as a metric of the optimization problem.
  Since the sources of traffic are not static by any means, the solution to the placement problem should be dynamic, such that the resources can be reallocated to convey the new system state. In contrast, among the mentioned papers, only [71] [74], [49], [49], [76] have considered dynamicity, while only [49], [49], [70], [71], [74] are designed for replicated applications.

- **Autoscaling**:
  On-demand computing applications are expected to face traffic variations as a function of geographical location and time. The placement mechanisms help mitigate the overloading of the application by allocating resources based on the sources of traffic, yet an application can still be overloaded in case the overall system load is greater than the overall application resources. The placement thus hits a performance wall induced by the limitation of resources per application. In such a scenario, the solution corresponds to increasing the resources allowed for the application. To better understand, consider a Kubernetes application composed of 2 replicated pods, if the overall load is greater than the capacity of these 2 pods, no matter how you place your replicas, the application will reach an overloaded state. The solution in this case is increasing the size of the replica set, thus autoscaling the resources allocated for the application. Autoscaling has been extensively studied over the previous decades [78] [79], and its application in the context of cloud continuum platforms remains nascent. This can be attributed to the popularity of offloading as an alternative to autoscaling [44]. Such that autoscaling is not handled at the edge and fog layers. Although such an approach is valid for certain types of applications, it does not offer a feasible solution for application components that require deployment on the edge.

  Nevertheless, the literature includes a wide range of autoscaling techniques in similar platforms. Autoscalers designed for general-purpose Kubernetes clusters aim at providing a seamless service for the application users [80], [81]. Geo-distributed computing environments autoscalers aim at selecting the nearest available resource [82], [83]. Few papers propose auto-scaling systems designed for fog computing platforms. *Zheng et al.* propose to vary the number of Kubernetes pods according to the load, but do not address the question of placement nor efficient routing between the end users and their closest resources [84]. On the other hand, *ENORM* aims to reduce latency between users and computing devices, and the network traffic to the cloud [85]. However, it chooses the resources regardless of their location, and essentially considers every fog node as equivalent to one another. Finally, *Voilà* [49] considers the autoscaling of Kubernetes pods according to proximity-awareness metrics, but remains limited to a single Kubernetes cluster.

In NebulOuS we aim at autoscaling that covers all the layers of the cloud continuum, an application component should be able to migrate from one cloud provider to another and from one layer based on what will provide a better performance at an optimized cost.

3.2.6 Interoperable data collection and management

The GSM Association [86] proposes a generalized architectural framework for the delivery of Big Data services based on the Internet of Things. This architecture can be summarized in the following components:

- The IoT service layer for handling the bidirectional communication with the IoT devices.
- A Context Data Layer responsible for handling bidirectional communication with external systems (other than IoT devices) for extracting information.
- The Data and Protocol Mediator responsible for transforming data formats of the incoming data (IoT or other sources) to and from a harmonized format.
- Data & Control Broker responsible for enabling access to harmonized data entities through a query and subscribe API.
- IoT Big Data Store to provide data storage for the managed data.
- BigData processing for analyzing context information.

For the IoT service layer, several communication protocols appear to be broadly used in the IoT ecosystem [87]: OPC-UA, MQTT, AMQP, REST and CoAP. These protocols fall in one of the two following paradigms: client-server or publish-subscribe. In a client-server pattern the data producer must open a communication channel with each of the receivers of the data (usually only one) and send the message. In contrast, the publish-subscribe pattern decouples the concept of publishers (the systems that publish messages) and subscribers (the systems interested in receiving these messages) using intermediate servers (message brokers). To articulate the communication, messages are logically grouped into topics. Publishers write data on topics and consumers subscribed to the same topic receive these messages. Any system can be both a publisher and a subscriber for an indeterminate number of topics. In comparison with client-server communication, the publish-subscribe paradigm is a decoupled and asynchronous way of communication between different parts of a system, allowing for loose coupling and reducing dependencies between components.

Using the previously mentioned communication protocols, IoT devices produce information often related to events (e.g. a lightbulb has been lid, a push button has been pressed, etc…) and observations (e.g. the room temperature is 25ºC, the noise volume is 30dB, etc…). For both types of data, apart from the actual payload (temperature, noise volume, button press status, etc..), equally important is the time instant associated with that payload. This type of time-stamped data requires for novel approaches when it comes to the storage and query mechanisms compared with traditional storage systems. Traditional Relational Database Management Systems (RDBMS) are optimized for storing relationships between data (a student is enrolled in a course that is taught by two teachers and the lectures happen each Monday and Friday in room CS23). Applications managing such information have the necessity to query the stored information using different perspectives (what courses does a teacher teach? What lectures take place in room CS23), and updating that information while preserving its consistency. In contrast IoT applications require a different kind of query semantics (e.g.: what is the hourly average temperature of room CS23 during last year?). To cope with this necessity, Time Series Database Management Systems (TSDBMS) appeared. TSDBMS are optimized for ingesting large amounts of time-stamped data and offer advanced time-related query semantics (time average, time bucketing, etc..). Many options exist for TSDBMS [88], among these InfluxDB[26], Kdb+[27] and OpenTSDB[28] to name a few. In general, all time-series databases listed offer the

[26] https://github.com/influxdata/influxdb
[27] https://github.com/timeseries/kdb
[28] http://opentsdb.net/

Funded by the European Union

mechanism for storing big amounts of time-stamped data (either single-valued or multi-valued) to later retrieve it using advanced query semantics that include time aggregations. To support the ingestion workload, often these database services offer the opportunity to be deployed in clustered environments.

Zenoh[29] (Zero Overhead Network Protocol) aims at closing the gap between the data transmission and storage/retrieval aspects of IoT data management. It is a pub-sub protocol that includes storage capabilities in the protocol brokers. On top of that, Zenoh offers a lightweight query mechanism for clients to retrieve the stored data. This query mechanism is seamlessly integrated into the pub/sub mechanism traditionally used by clients. In contrast, some implementations of pub/sub protocols offer storage capabilities but do not support querying as an integrated system capability[30]. This is the case of EMQX, an open-source MQTT broker that allows storage systems to be installed as plugins and manage the storage of information using SQL and No-SQL databases.

Besides managing the actual data generated by the IoT devices (temperature reading, pressure, etc..) the GSM IoT Big Data reference architecture acknowledges the importance of context information and links to the ETSI Context Information Management (CIM) [89] a framework for modelling context information in the intelligent solution space. In CIM the context is understood as any relevant information about physical/logical entities including their properties (temperature, location, or any other such parameter), and their relationships with other entities (e.g: an entity belongs to another, an entity is located inside another, etc..). To model the context information, CIM proposes the use of NGSI-LD framework. In NGSI-LD, entities are structured like a graph of properties and relations with other entities. The representation of this information is presented in JSON-LD [90], an extension of JSON that defines the reserved keyword "@context" for including the reference to NGSI-LD meta-model. NGSI-LD also states how entity attributes, types and relations should be encoded. Moreover, NGSI-LD proposes a REST API (client-server paradigm) for implementing the exchange of context information between the producers and the context broker (the component responsible for holding the context status) and between the context broker and subscriptions to context information. This API includes operations for publishing new context data, querying the existing data and subscribing to changes in the context. As part of the FIWARE[31] initiative, some NGSI-LD compliant context brokers have been implemented: Scorpio[32], Orion-LD[33].

The data protocol mediation layer of the GSM architecture handles the transformation of the incoming data (IoT or other sources) to and from a harmonized format. This transformation includes: the conversion of payload encoding (e.g: from XML to JSON); mapping of data structures (e.g: move data from the "temp" field provided by the IoT sensor to the "temperature" field used in the unified data model); units conversion (e.g: from the temperature value in Fahrenheit provided by the IoT sensor to the value expressed in Celsius expected by the platform); etc…

These transformations can be implemented following the often referred as Data Stream Processing (DSP) paradigm. This paradigm pursues the modelling of data transformation pipelines as direct acyclic graphs of nodes that can either be a source (where the information comes from); an operation (that transforms the data); or a sink (where the results are sent/stored). Typically, a DSP application topology is composed by a single source (e.g: temperature readings stream from sensors distributed across a building), many operations with arbitrary dependencies between them (e.g: convert the initial XML payload to JSON; convert from Fahrenheit to Celsius; trigger an alarm if the temperature from a room/wing of the building rises; etc..) and one or many sinks for the information (databases, pub-sub mechanisms, etc…). By utilizing this approach, application developers benefit from the re-usability of the application components. For instance, one can easily implement a piece of code that is provided with a description of the XML to JSON fields mapping, receives XML payloads and outputs their JSON counterpart. This software element can be used to convert data coming from many different IoT devices using an XML communication protocol.

Although a single execution of the transformation pipelines usually consumes low computing resources (CPU time), in applications where the number of IoT devices and/or the publishing frequency are large, the execution of the

[29] https://zenoh.io/
[30] https://zenoh.io/docs/overview/zenoh-in-action/#queries-in-zenoh
[31] https://www.fiware.org/
[32] https://github.com/ScorpioBroker/ScorpioBroker
[33] https://github.com/FIWARE/context.Orion-LD

Funded by the European Union

transformation pipelines can no longer be handled by a single CPU thread, and it must be distributed. In that situation, DSP can benefit from the separation of the computation units in different operators, as their execution can run in parallel. This parallelism can be achieved in three forms: Task parallelism (separated operations/group of operations from the DAG can run in parallel); pipeline parallelism (consecutive operators on the same path of the DAG process consecutive elements from the data stream at the same time) and data parallelism (multiples instances of the same operator processing portions of the data stream).

At deployment time developers can define the DSP topology, and the parallelism level of each operator (e.g.: how the stream can be split between multiple instances of the same operator). Once the applications are running, the usual stochastic volume of the data streams to be processed causes fluctuations in the workload. Moreover, failures of the computation nodes or the networks that connect them are not exotic. In this situation, it is necessary to have a close monitoring of the DSP to check for failures and QoS fulfilment and provide an intelligent mechanism capable of dynamically adapting the DSP execution model in response. As summarized by [91], several approaches for tackling this issue are being studied in the literature (Topology adaptation, deployment adaptation, processing adaptation, overload management, fault tolerance, adaptation, and infrastructure adaptation) relying on different techniques: mathematical optimization and game theory, control theory, graph theory, stochastic modelling (and, in particular, queueing theory), heuristics, and ML.

Currently, several declarative frameworks for defining and executing DSP exists: Apache Nifi[34] Apache Airflow[35], FogFlow[36], ZenohFlow[37] among others. These declarative frameworks require the administrator to express the pipelines, as structured text files or even through a GUI, and provide the code that implements each of the declared operations. With that information at hand, the framework is responsible to span the necessary workers that will handle each of the steps of the flow. Usually, the orchestration frameworks can take advantage of computation clusters so that the workload is distributed among a pool of workers, making this process totally transparent to the developer of the workflow.

3.2.7 Semantic Modelling for Interoperability

Interoperability is the ability of a system to communicate with other systems. In NebulOuS, we focus on interoperability in the IoT and the Cloud Continuum. We present below state of the art in frameworks and semantic technologies relevant to our focus.

Interoperability Frameworks and Platforms

Several approaches targeting interoperability in distributed systems, such as SeDim [92] , OSDA [93], MUSDAC [94], INDISS [95], uMiddle [96], and CONNECT [97], have been proposed. These generally exploit open system integration paradigms, notably Service-Oriented Architecture (SOA) and Enterprise Service Bus (ESB), to bridge inherently heterogeneous interaction paradigms such as client-server, publish-subscribe, and tuple space [98]. These approaches, however, are not targeted to the IoT or to the Cloud Continuum, and hence ignore the scale of heterogeneity encountered in such contexts.

More recently, approaches that focus explicitly on the IoT and/or the Cloud Continuum have been Proposed. These generally unfold along several dimensions [99]: the *device level*, regarding computational and communication capacity; the *syntactic level*, regarding data format, schemas, and interfaces; the *networking level*, regarding internetworking protocols; the *platform level*, regarding underlying OSs and runtimes; and the *semantic level*, regarding, data and information models. An overview of such approaches is provided below.

[34] https://nifi.apache.org/
[35] https://airflow.apache.org/
[36] https://fogflow.readthedocs.io/en/latest/index.html
[37] https://github.com/eclipse-zenoh/zenoh-flow

Funded by the European Union

AllJoyn [100] is a full-stack[38] framework with wide multiplatform support (Android, Arduino, iOS, Linux, OS X, Windows, OpenWRT). It achieves interoperability through a common protocol that enables service discovery and interaction. Interaction is based on the D-Bus specification, an RPC mechanism extended by AllJoyn to operate across remote devices, and is realised over the AllJoyn bus: a logical bus that comprises a set of AllJoyn routers. AllJoyn is written in C++, and provides C, Java, JavaScript, C#, and Obj-C bindings for creating AllJoyn apps, i.e., apps that can interoperate with different AllJoyn resources. Non-AllJoyn resources may be discovered and interacted with via resource-specific software bridges.

IoTivity [101] is another full-stack framework with wide multiplatform support (Android, Arduino, Linux, Tizen, and Yocto). Unlike AllJoyn, however, IoTivity adopts a RESTful style of interaction (based on CoAP). IoTivity is written in C and provides C++ and Java bindings.

oneM2M [102] achieves interoperability across different machine-to-machine solutions through standardization, i.e., by developing a global specification that integrates existing platforms, frameworks, and standards. It provides a homogenisation layer that mediates, by virtue of semantic mediation gateways (SMGs), between IoT resources and applications. SMGs use semantic annotations and inferential reasoning to translate between diverse data representations across different devices or systems; this is based on a minimal base ontology that is readily linked with external 'domain-specific' ontologies. SMGs are utilised by Interworking Proxy Entities (IPEs): application entities that provide interfaces between oneM2M and non-oneM2M compliant networks or systems.

NGSI-LD (Next Generation Service Interface for the Internet of Things - Linked Data) [103] provides a data model and communication framework for representing and exchanging, respectively, information between devices, applications, and data sources in the IoT. Semantic interoperability is achieved through a flexible and extensible RDF-based[39] ontology that represents IoT entity types, their properties, and the relationships between them. This allows different systems to represent and exchange information in a standardized way, which is of paramount importance for the development of IoT applications and services. NGSI-LD offers a RESTful API for exchanging information thus simplifying its integration into IoT applications.

In addition to the above solutions, several frameworks that focus on facilitating application development in the IoT have been proposed. The IDeA framework [104] provides abstractions of IoT devices based on the IoT-A reference model [105], along with a tool – the IoT AppFramework – that enables application development based on these abstractions; heterogeneity between hardware and software components is addressed through SysML4IoT, a SysML profile that provides stereotypes for abstractly describing a system in terms of devices and services. In a similar vein, IoTLink [106] provides a model-driven development toolkit based on IoT-A; it assists developers in overcoming heterogeneity by providing a layered set of abstractions for defining physical connections, sensor data pre-processing and aggregation, virtual entities (proxies, essentially, of devices), and crucially, integration of virtual entities within applications.

The framework in [107] proposes a web application (a single-page app) for rapid, browser-based prototyping of IoT applications. The focus is on aiding developers in bridging interactions between GUI-based control panels and the devices that are controlled through them. The framework is based on virtual entities that act as in-browser device emulators and interact with the application through virtual ports that emulate physical I/O ports, and the use of a proxy server that enables interaction with these emulators through the virtual ports. Communication is managed by the WebRTC protocol stack, and by a proxy server in case WebRTC is not supported.

Omni [108] provides an application framework for seamless device-to-device interaction. It facilitates building applications that use heterogeneous communication channels to intelligently discover useful devices in a vicinity, and to transfer

[38] https://spring.io/projects/spring-boot
[39] Resource Description Framework.

Funded by the European Union

content directly between these devices in an efficient and realistic (in terms of energy and time) manner. The framework is based on the tenet that device-to-device communication entails inherently different communication tasks: periodic tasks for sharing small amounts of data in a localised area (e.g., network-level neighbour discovery and application-level service discovery), interspersed by sporadic application-specific tasks that tend to share larger amounts of data.

BIG IoT (Big Data for IoT) [109] is a platform for facilitating the development, deployment, and management of IoT applications that generate and process big data. It provides a set of tools and a framework for collecting, analysing, and visualizing data from IoT devices and sensor networks. It also aims at facilitating the integration of IoT data with other data sources, such as social media, weather, or traffic data, to enable new and innovative use cases. The platform also provides features such as scalability, security, and real-time data processing, which are critical for IoT applications that handle big data.

Mortar [110] is an open-source testbed for portable building analytics, and a platform for storing, describing, updating, discovering, and retrieving building data. It allows developers to test their building analytics algorithms and models on a variety of data sets. It provides tools to easily collect and clean data, visualize the results of their algorithms, and compare them with other algorithms and models. It provides a way for developers to deploy their algorithms in real-world buildings, giving them the opportunity to test their algorithms on real-world data and improve the energy efficiency and overall performance of the building. To reconcile the heterogeneities across different buildings, the Brick ontology is used [111].

In a similar vein, LinkLab [112] provides a set of tools, and a framework, for collecting, analysing, and visualizing building energy data, as well as for testing and evaluating building energy management and control algorithms. LinkLab is designed to be highly configurable and customizable, so that it can be adapted to the needs of different buildings, research projects, and energy management strategies.

Smart-M3 is a platform for information sharing [113]. It comprises two main components: Semantic Information Brokers (SIBs) and Knowledge Processors (KPs). SIBs are ontological triple stores that persist knowledge about resources in a smart space i.e., in "a *named search extent of information*". KPs are the "*active part*" of smart spaces; they bundle together application logic and APIs (the so-called KPIs) that facilitate developers in accessing information in SIBs. KPs are both the providers of the information stored in SIBs, and its processors; based on this information, KPs may take externally observable actions. Applications in Smart-M3 are therefore 'mashups' of different KPs. KPs are loosely coupled and interact with each other only indirectly through the information in the SIBs.

DynaSense [114] is a middleware system that allows user applications to be agnostic of the data sources or sensors in use. It provides a unified approach for accessing data from various sources. It dynamically decides how to acquire data from available sources, as well as how to deliver it to requesting user applications. Crucially, DynaSense allows applications to use high-level data without any code modifications. SOUL [115] is an edge-cloud system for mobile applications in a sensor-rich world. It is designed to handle large amounts of data generated in sensor-rich environments. SOUL provides a set of tools and a framework for developing, deploying, and managing mobile applications, including data collection, data analysis, and data visualization tools. Similar to DynaSense, SOUL allows for portable and reusable applications that consume high-level data without code modifications.

In [116], the authors present a semantic interoperability approach for the automated deployment of generic IoT applications i.e., of applications that can run over distinct, albeit functionally similar, device sets. They advocate the use of a semantic registry for IoT devices that conceals technological, as well as semantic, heterogeneities, the latter stemming from the use of heterogeneous domain ontologies for annotating IoT data. Ontology alignment is used to reconcile domain ontologies. The approach is based on (a prior version of) the SSN ontology.

XWARE [117] is a customizable interoperability framework aiming at enabling communication between services running over diverse, typically proprietary, middleware platforms within a particular domain (e.g., a home environment, a healthcare system, etc.). XWARE advocates a modular design to facilitate customisation towards a domain, thus enabling, in principle, its applicability across many domains. It employs plugins to communicate with legacy middleware; plugins provide such functionalities as service advertisement, lookup, and invocation. It also employs mediators to translate messages between different plugins.

In [118] the SemIoTic ecosystem is presented for facilitating the development of IoT applications that are agnostic to the underlying IoT infrastructure, and can thus interoperate with diverse, albeit functionally similar, IoT devices (e.g., diverse temperature sensors that adhere to different data formats and data exchange protocols). This paves the way for the development of reusable IoT applications that can be readily ported across diverse IoT spaces. At the semantic layer, SemIoTic deals with issues of interoperability through the incorporation of SEMIC: a SOSA/SSN-based meta-ontology used for describing both static and dynamic aspects of resources in an IoT space. SemIoTic provides programmatic support and algorithms to specify user-defined actions expressed in terms of semantically meaningful concepts from the SEMIC metamodel, and translate between them and low-level sensor and actuator data. To achieve interoperability at the data exchange layer, SemIoTic supports wrappers for IoT devices which consist of a common interface to enable SemIoTic to communicate with them (northbound interaction), and device/manufacturer/model-specific code that encapsulates the low-level (southbound) interaction. Also, SemIoTic defines a specification methodology for virtual sensors which enables semantic interpretation of low-level sensor data and provides application-oriented access to the IoT space resources.

Ontologies

The "Semantic Sensor Networks" (SSN) ontology [119], [120] is a comprehensive W3C recommendation[40] for providing a formal representation of: sensors and their properties (sensing modalities, units and ranges of measurement); actuators; the types of phenomena - or features of interest - being observed or affected by actuations; the procedures involved in realising observations and actuations. SSN incorporates a lightweight, self-contained, core ontology called SOSA (Sensor, Observation, Sample, and Actuator). Through their different, albeit complementary, scopes and degrees of axiomatization, SSN and SOSA are together able to provide interoperability across a wide gamut of applications and use cases, ranging from satellite imagery to social sensing and citizen science. Both SSN and SOSA are RDF-based and are therefore extensible, hence reusable. IoT-Lite [121] is an instantiation of SSN, a lightweight semantic model that includes the least number of concepts required for classifying IoT data. IoT-Lite is not intended to be a fully-fledged ontology but rather a lightweight core that can be extended, if needed, with application-specific semantic models.

The Smart Applications Reference (SAREF) [122, p. 103] ontology is a reference ontology for smart appliances. It provides a common vocabulary for describing the functionalities, features, and services of smart appliances, as well as the communication and data exchange protocols that these devices use. SAREF is intended to enable interoperability and support the development of smart home and building automation applications; to this end, it provides a modular representation of the service that an appliance provides in terms of its functions and the actual commands that invoke these functions. Notably, SAREF is narrower-in-scope than SSN and is thus based on a more domain-specific nexus of interrelated concepts.

The IoT-A ontology [105] is part of the wider "IoT Architectural Reference Model (ARM)". It provides a formal representation of the key components and their relationships in an IoT architecture. The ontology is modular comprising different facets or "models": the *domain* model that includes fundamental concepts such as services and virtual entities, the *entity* model representing digital twins, the *resource* model carrying device-specific information, the *service description* model that describes the services offered by an IoT device, the *event* model describing the events and changes that a device may engage in, the *functional* model that encodes lower-level concerns such as protocols and device management, as well as concepts for describing data flows and intercomponent interactions. The IoT-A ontology does not provide concepts for readily supporting context awareness.

[40] https://www.w3.org/TR/vocab-ssn/

Funded by the European Union

The NGSI-LD ontology [123] is a formal representation of the concepts and relationships of the NGSI-LD data model that aims at allowing the exchange of data across IoT devices and systems. It includes the usual prose for the representation of IoT entities and the properties thereof, as well as a nexus of interrelated concepts for representing and sharing contextual awareness. It fails to support actuation environments.

In [124], an ontology is proposed for enabling the development of 'generic' IoT applications that are agnostic of the underlying sensing or actuating devices that they interact with. The ontology enables seamless device-to-application communication by abstracting away from vendor-specific device details through the annotation, and classification, of the data streams that these devices generate/accept. The ontology is based on an old version of SSN (prior to the integration of SOSA) extended with the authors' own ontological actuation model. In a similar vein, the SEMIC ontology [27] aspires to introduce an interoperability layer that bridges the world of IoT applications with the realm of sensing and actuating devices. SEMIC extends SSN through a nexus of interrelated concepts that allow for the modelling of virtual (software) sensors and their interrelations with underlying physical sensors, as well as of virtual observations. Virtual observations are aggregations of physical observations that collectively provide the kind of higher-level information sought by IoT applications (e.g., deriving room occupancy information from camera and wi-fi access point data). IoTMA (Internet of Things Model and Analytics) [32] is another SSN-based ontology that provides a common vocabulary for describing the capabilities and properties of IoT devices. IoTMA emphasises on formalising context awareness i.e., on including concepts for reviewing observations gathered in a particular IoT context and determining/prioritising any future actuations in that context based on that awareness. Clearly, this prioritisation is of utmost importance for supporting low-latency applications deployed in the Cloud Continuum.

OWL-Q [125] is an ontology for the formal representation of QoS-based attributes of web services. It treats a web service as a set of QoS attributes, each associated with one or more metrics, where an attribute is analogous to a "feature of interest" in SSN or SAREF. Through its extension, Q-SLA, it provides constructs for service-level objective representation and negotiation, service-level agreement (SLA) representation including violation penalties, and performance monitoring based on SLAs. QoS description is also attempted through the QoSOnto-CSS ontology [126] which, however, advocates a different approach to modelling by representing different QoS criteria as distinct classes.

CoCoOn [127] is an ontology that provides a standardized vocabulary for describing concepts and their relationships in the realm of cloud computing, specifically in the context of IaaS. It aims at providing a common understanding of cloud computing concepts to aid the discovery and consumption of infrastructural services. It is an extension of the SSN ontology. A similar ontology is mOSAIC [128] which aims at reconciling the heterogeneity in terms used by cloud service providers to enable discovery and negotiation of cloud services across different cloud platforms.

3.2.8 Cloud Service Brokerage

In a cloud environment, a cloud broker is an important entity that works as an independent middleware between cloud customers and providers to address the issues related to satisfying the customer preferences and the service provider profits through negotiation between them [129]. Recent domain survey papers [130], [131] identified the following key challenges and future research streams related to cloud brokerage:

- **Adaptive and Fluid Deployment.** Cloud brokerage is challenged by the need for methodologies to benchmark containers so that the heterogeneous cloud and edge resources can be compared and ranked (Varghese et al. 2016). Furthermore, the fact that containers do not provide the same level of isolation as VMs (containers share the operating system) means that the degree of performance interference is higher compared to the VMs [132], which makes satisfying the SLA more challenging. Other challenges relate to the management of the container lifecycle in the cross-cloud context and quantifying the cost of migration, among others.
- **Intelligent Decision-Making**. Currently, decision-making at the brokerage level relies heavily on multi-criteria decision making (MCDM) methods [133]. MCDM methods can be extended to provide better support for decision making at the brokerage level. Examples include multi-objective meta-heuristics such as multi-objective evolutionary algorithm and multi-objective particle swarm optimization, fuzzy optimization, and multi-objective optimization methods combined with neural networks. MCDM methods should also consider the priority of service provision. The provision of differentiated services can better meet the needs of different users, giving

Funded by the European Union

higher priority to requests from customers with high service quality requirements, and on the other hand offering more competitive prices for requests from customers that can tolerate unstable service quality. Furthermore, machine-learning techniques can help to quantify the extent to which providers fulfil their offerings and the extent to which the applications are satisfied in practice. Also, machine learning models can be deployed to adapt deployments proactively, i.e., before the adaptation becomes necessary. A relevant challenge is to accurately predict when requirements will change and the feasibility of adapting the deployment at that time. Furthermore, feedback on the performance of the adopted learning approaches is necessary, as learning in different ways can lead to different results.

- **Resource management:** Current resource pool management for cloud brokers often relies on prediction-based approaches. The accuracy of the predictions greatly affects the performance of these approaches. Therefore, methods that do not rely on predictions and can provide performance guarantees and interpretability deserve more research in the future.
- **Pricing**: For pricing problems, the following directions are worthy of research investigation. Firstly, the fairness of pricing for multiple cloud service providers from a federated cloud perspective is vital. Furthermore, a study of pricing based on the true performance of cloud services from a user perspective is needed. In addition, pricing dynamically to maximize cloud broker profits from the perspective of cloud broker resource pool management is also important.
- **Governance and QoS:** Providing cloud services with guaranteed performance based on realistic evaluations through cloud brokers with QoS monitoring capabilities. Currently, the performance of services offered by different cloud service providers varies even though they have similar performance promises, so how cloud brokers evaluate and dynamically select and manage resources based on that evaluation deserves more research. Further research is needed in methods for governing the use of cloud services, including mechanisms for enforcing service level agreements and resolving disputes. Moreover, techniques are the brokerage level are needed for ensuring the security and privacy of data and applications when they are hosted on third-party cloud platforms.
- **Ability to solve high-dimensional problems**: Most of the current research is based on experiments conducted in simulated scenarios, and the problem dimensions considered in some studies are much smaller than those to be considered in real scenarios. Therefore, solving the problem of resource availability in a larger dimension remains a challenge.
- **Reducing latency based on fog computing**: Latency reduction is one of the key optimization goals in current cloud broker research, but current research mainly focuses on reducing latency by selecting data centers in different locations. The fog computing paradigm optimizes latency by reducing the transfer of data by distributing jobs to fog servers that are closer to end users. Latency will continue to be a major topic in future cloud broker research and could be further investigated by considering the adoption of new paradigms. Further research is needed in methods for monitoring and optimizing the performance of cloud services, and for ensuring their availability and reliability.
- **Discovering and combining cloud services to meet complex needs**: Inter-cloud environments and environmental conditions at the implementation time of broker in the inter-cloud environments are more variable and unpredictable than single-cloud environments. Therefore, when a broker performs orchestration tasks in addition to selecting the service, it should appropriately combine services and increase the resistance to failure in the broker.
- **Ability to deal with cloud provider failures:** The multi-cloud nature of cloud brokers is that they are not dependent on a single cloud provider, while cloud provider failures can still occur and be very costly today. Therefore, failure management through cloud brokers is a viable method to provide a more stable service to users. Especially in the case of combining services, in case of failure to run only one service, the process of running the broker will fail. Therefore, management and detection of failures during the implementation of service composition is vital.

Nebulous will primarily focus on intelligent decision-making at the brokerage level, while advancing several of the aforementioned challenges that are relevant to the project objectives and use case needs, such as adaptive and fluid deployment and the ability to solve high-dimensional problems. On the other hand, we will not focus on challenges such as pricing and brokerage governance which in our view should be addressed at a stage which is closer to production/commercialisation.

Most relevant works on brokerage decision making consider quantitative metrics, with less attention put qualitative aspects [131]. NebulOus will go beyond SOTA by introducing cloud and fog brokerage capabilities in a similar manner that OS intermediates between computing resources and software. Specifically, it will support cloud computing continuum providers (organisational units, telecom providers etc.) and consumers to advertise the available resources for the ad-hoc formulation of local continuums and define their preferences, respectively, in both qualitative and qualitative ways (e.g., using linguistic terms). Furthermore, NebulOuS will be able to match, on demand, the above aspects based on an MCDM approach that copes with precise and imprecise criteria to broker the ad-hoc generation of micro local cloud continuums.

3.2.9 Networking and Communication

In this subsection, we cover communication aspects in the Cloud-to-Edge-to-IoT continuum. For presentation purposes, we have divided the state-of-the-art into two major areas: cloud networking (covering aspects related to tunnelling, overlays and network virtualization) and IoT communication (covering wireless IoT protocols). In addition, we include aspects relevant to networking and communication in the cloud- and edge-native world by presenting recent advancements related to the most popular platforms used in the ecosystem: Linux and Kubernetes networking (which are vertical to the continuum). In particular, we present tools and specifications which enable container networking inside and between Kubernetes clusters, as well as service-to-service communications in the microservices architectural style by presenting state-of-the-art service mesh solutions. Some related higher-level functionalities, such as IoT data management, are covered in Section 3.2.9. In any case, all of the presented technologies are carefully taken into account by the NebulOuS project, and the best will be selected to achieve high-performant, end-to-end communications in fog computing environments.

Cloud networking

In the cloud, virtualized resources may exist in different physical machines and sites, but need to be handled as if they were a part of the same local network. For this reason, datacenter networks are typically established using techniques such as network tunnelling to create secure overlay networks which interconnect the various compute resources. This way an isolated, logical network is created that can span multiple physical locations. To achieve this, different technologies can be used.

One solution is using VPNs, which have evolved to be multipoint-to-multipoint overlays and are supported by existing public IaaS platforms (e.g., AWS's VPCs[41] and Azure's VNets[42]). Another solution is the use of VXLAN [134] (which was designed to extend the capabilities of legacy VLANs) and other similar technologies such as NVGRE [135] (which uses a different encapsulation method). Geneve is the latest addition to this family, aiming for a flexible design which can be easily extensible to accommodate various requirements. In any case, the typical approach is to encapsulate MAC frames and transport them over an IP network to form virtual networks between the resources.

As can be easily inferred, the latest advancements in cloud networking are strictly coupled with advances in virtualization and adhere to the general trend for the softwarization of hardware devices and their corresponding functionalities. As a result, there is an interplay between cloud networking and the research areas of Software-Defined Networking (SDN) [136] and Network Function Virtualization (NFV) [137].

SDN separates a network's control plane and its data plane; early incarnations focused on programmable control planes but the latest focus is on data-plane programmability. The latter is also a convergence point with NFV approaches, which attempt to virtualize network functions typically performed by dedicated equipment and deploy them into general purpose hardware. As the two approaches are usually coupled to establish modern virtualized and programmable networks, they are jointly referred to as SDN/NFV [138] [139]. As is to be expected, many current cloud/edge networking solutions use SDN/NFV techniques to deploy and manage networks between computational resources.

[41] https://aws.amazon.com/vpc/
[42] https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview

Funded by the European Union

Several tools incorporate the aforementioned functionalities and can be used for the creation of secure network overlays. Some of the available solutions used commercially include Open vSwitch (OvS)[43], Open Virtual Network (OVN)[44] and VPN Gateways[140].

At this point, we will shift our focus to Linux kernel networking (which is especially important for cloud/edge networks). Recent advancements in the field are closely tied to the IO Visor project[45], an open-source Linux Foundation Collaborative Project. The aim of IO Visor is to solve the operational impact created by the coupling of input/output functionality with networking services, by effectively decoupling them. With the usage of its data plane, new tools can be developed using "networking functions" that achieve interoperability between different manufacturers, offering better performance compared to existing solutions such as iptables[46].

Our main point of interest lies on two specific, closely-related technologies: eBPF and XDP [141], which are currently used in many state-of-the-art tools[47]. eBPF can run sandboxed programs in the Linux kernel. It is used to extend the kernel capabilities safely and efficiently without requiring any change to its source code. Running sandboxed programs within the operating system, developers can utilize eBPF to add additional capabilities to the operating system at runtime. The efficiency provided by the Just-in-Time compiler provides execution performance close to that of natively compiled in-kernel code. This fact makes eBPF ideal for packet processing, making it a great building block to offer efficient cloud-native networking, security and monitoring functionalities.

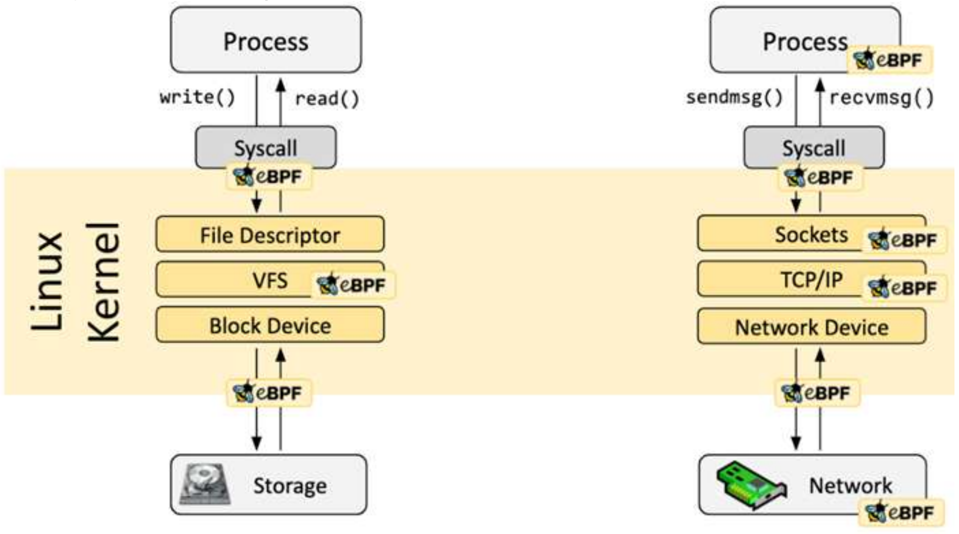

Figure 2: eBPF hooks[48]

XDP stands for eXpress Data Path and provides a framework for BPF that enables high-performance programmable packet processing in the Linux kernel. It runs the BPF program at the earliest possible point in the software, namely at the moment the network driver receives the packet [142]. XDP works in concert with the Linux kernel and its infrastructure, meaning the kernel is not bypassed as in various networking frameworks that operate in user space only (e.g. dpdk[49]).

[43] https://www.openvswitch.org/
[44] https://www.ovn.org/en/
[45] https://www.iovisor.org/
[46] https://www.netfilter.org/projects/iptables/index.html
[47] https://ebpf.io/applications/
[48] https://ebpf.io/what-is-ebpf
[49] https://www.dpdk.org/

Funded by the European Union

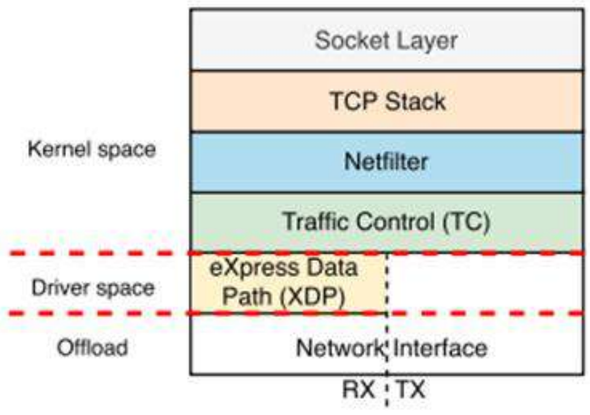

Figure 3: XDP overview [141]

Lately, those offered capabilities have been utilized extensively in the cloud-native landscape. With the recent widespread adoption of OS-level virtualization (i.e., containerization) and Kubernetes as the de-facto container orchestration platform, several tools have been created to enable communication between services on Kubernetes clusters. Such tools are commonly referred to as “network plugins” or “CNI plugins” [143].

Kubernetes network plugins

Kubernetes networking solutions are built upon the standardized Container Network Interface (CNI)[50]. CNI is a Cloud Native Computing Foundation (CNCF)[51] specification and collection of libraries. Those libraries are used to write plugins for configuring network interfaces in Linux containers. The plugin specification defines an interface for configuring the network, provisioning IP addresses, and maintaining connectivity with multiple hosts.

CNI plugins are composed of two components: the CNI daemon and the CNI binary files. The CNI daemon, which runs in user space, is responsible for setting up the host network devices (e.g., bridge, overlay tunnel endpoint), the tunnelling options (e.g., VXLAN, UDP, etc.) and iptables (e.g., network policies). The CNI binary configures the network of a Pod, i.e., sets up virtual ethernet (veth) links and Pod IP addresses. It is called by the container runtime only when setting up the network for a newly-created Pod.

The different plugins vary in their design and way towards facilitating Pod communication, as well as in their support for network policies, i.e., the means to enforce rules indicating which network traffic is allowed and which Pods can communicate with each other.

To implement specific packet forwarding, routing, and networking policies, CNI Plugins typically leveraged ‘iptables’, a userspace interface to set up, maintain and inspect the tables of IP packet filter (Netfilter) rules in the Linux kernel. Iptables use some “hook points” in the network stack to implement packet filtering, NAT, and security-related policies.

Developers using the CNI standard can create network plugins to interoperate with a variety of container runtimes. CNI networks can use an encapsulated network model, such as Virtual Extensible LAN (VXLAN), or an unencapsulated (decapsulated) network model, such as Border Gateway Protocol (BGP) [144]. CNI is not explicitly tied to Kubernetes, as the latter is just one of the runtime environments that adhere to the CNI specification.

CNI has attracted large support and many popular plugins have been created for Kubernetes in the cloud-native ecosystem. Popular ones include Calico [145], Cilium [146] and Flannel [147], whereas an extensive list can be found on the CNI

[50] https://www.cni.dev/docs/spec/
[51] https://www.cncf.io/

Funded by the European Union

official github page[52]. It has to be noted that many of these CNI plugins offer joint solutions for both container networking and security.

Calico brands itself as a networking and network security solution for containers, virtual machines, and native host-based workloads. It supports a wide range of platforms, including Kubernetes, OpenShift[53], OpenStack[54] and bare metal services. Calico can be used both for networking and security, in contrast to some other plugins (e.g. Flannel). This is achieved by supporting network policy configuration, using the NetworkPolicy API. Linux-native tools are used to facilitate traffic routing and network policy enforcement, while a BGP daemon is hosted for distributing routes to other nodes. Even though Calico originally used iptables, it has recently added support for eBPF aiming to leverage its advantages for high-performance networking [55].

However, the first and most mature eBPF-based plugin is Cilium, an open-source project that has been designed to address the scalability, security and visibility requirements of container workloads. It has been embraced by Google in its Google Kubernetes Engine (GKE)[56], by AWS in EKS Anywhere[57], by Alibaba[58] and others.

Cilium, which can be used for both container networking and security, offers OSI layer 7 (HTTP protocol) awareness and enforcement of network policies on OSI layers 3 to 7 using an identity-based security model decoupled from network addressing. Furthermore, it can be used in combination with other CNI plugins by leveraging a feature called CNI chaining. Such features can be exploited to offer advanced security features, in addition to those described in the security section.

Another relevant CNI plugin is Kube-OVN [148]. Kube-OVN was designed to integrate OVN-based network virtualization into Kubernetes, bringing SDN-like functionality to Kubernetes. One notable feature is that it can use CNI chaining to integrate with Cilium[59], to combine the control-plane network abstraction capabilities of Kube-OVN with the high-performance monitoring and security capabilities that come with eBPF. Furthermore, another important aspect is that it offers native support for VMs. As such, it can be used by KubeVirt [21] to manage VM-based workloads along with container clusters in a shared environment, under Kubernetes.

Last, with respect to multi-cluster networking, tools such as Submariner [149] have been developed. Submariner was created to connect overlay networks of different Kubernetes clusters, by establishing encrypted tunnels between them. It is designed to be compatible with most CNI plugins.

Service meshes

Another important and still growing in popularity aspect of modern service communication is the adoption of service mesh solutions. The advent of service mesh software is tied to the popularity of microservice-based architectures [16] where (relatively) small chunks of functionality are divided among different networked computer programs. These programs need to communicate and doing so reliably is easier said than done. Similar to frameworks, such as web frameworks, which handle certain repeatable aspects for the creators of the applications so that they can focus on core functionality, service mesh software delivers a framework for reliable communication among services.

[52] https://github.com/containernetworking/cni/
[53] https://www.redhat.com/en/technologies/cloud-computing/openshift
[54] https://www.openstack.org/
[55] https://www.tigera.io/blog/introducing-the-calico-ebpf-dataplane/
[56] https://cloud.google.com/blog/products/containers-kubernetes/bringing-ebpf-and-cilium-to-google-kubernetes-engine
[57] https://isovalent.com/blog/post/2021-09-aws-eks-anywhere-chooses-cilium/
[58] https://www.alibabacloud.com/blog/how-does-alibaba-cloud-build-high-performance-cloud-native-pod-networks-in-production-environments_596590
[59] https://kubeovn.github.io/docs/v1.10.x/en/advance/with-cilium/

Funded by the European Union

However, the difference is that the service mesh is transparent to the application developer and decoupled from services' implementation, unlike a framework. Service mesh solutions relieve developers from hard-coding the communication reliability logic into their programs and present application operators with a single pane of glass for inspecting the inter-service communication activities. The reliability feature of service meshes includes backoffs and retries as well as security features for robust secure connections between two-way-authenticated services, employing techniques such as mutual TLS (mTLS).

Service meshes terminology maps similar concepts from general communication parlance: there is both the service mesh control plane (with management features included) and service mesh data plane. The control plane entities are responsible for configuring the data plane entities so that the desired communication patterns are possible. The popularity of Kubernetes in the microservices' world means that the majority of service mesh implementations are at least compatible with Kubernetes if not simply Kubernetes native. Furthermore, the popularity of microservice architectures and thus service meshes resulted in multiple not strictly compatible approaches to service meshes.

To alleviate this situation, a Service Mesh Interface (SMI) [150] standard has been proposed for Kubernetes and donated to the CNCF by Microsoft. All major open-source service mesh providers, such as Istio [18] and Linkerd [151], have joined the effort to build a consistent operator-oriented interface for defining and managing service meshes, regardless of the internal implementation (like what proxy server is used and whether XDP/eBPF is utilised). Recently (in 2022), the SMI has joined forces with a yet more general Kubernetes Gateway API [152], [153] project which handles now service meshes in their GAMMA (Gateway API for Mesh Management and Administration) initiative [154].

On the implementation side, the service meshes' data plane is usually wired as managed proxies which intercept services' traffic both on ingress and egress (from the service's perspective; the data movement considered is usually east-west rather than north-south, but under the GAMMA initiative north-south will also be supported). Thus, such proxies see all traffic and can manipulate it in various ways.

IoT Communication

Wireless communication is the preferred method of communication in the IoT for it provides the flexibility and mobility typically required by IoT applications. ***Figure 4*** presents an overview of diffused wireless technologies used in the IoT classified by signal range; a brief overview of these technologies is in order.

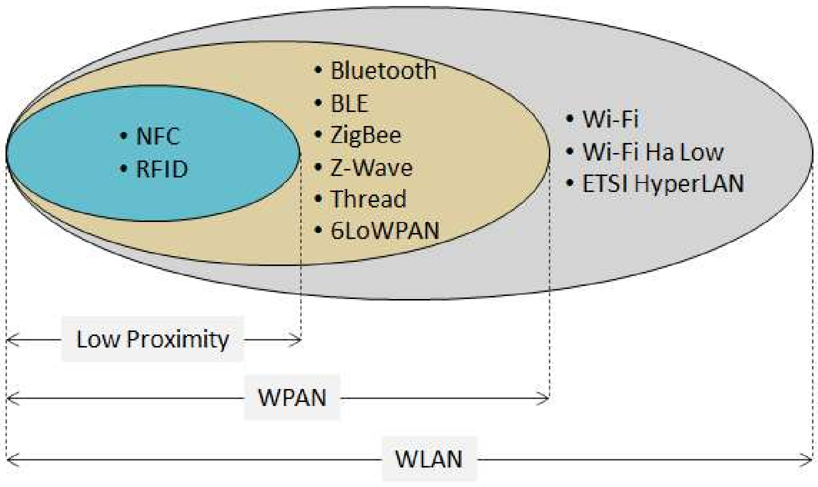

Figure 4: Wireless technologies used in the IoT

NFC (Near Field Communication) [155] and RFID (Radio-Frequency Identification) [156] provide short-range connectivity - in the order of centimetres in the case of NFC, and of a few metres in the case of RFID. NFC supports higher data rates than RFID, provides encryption of data transmissions, and requires synchronous interaction. RFID on the other hand makes no security provisions (relies on external physical security measures to protect transmitted data), and supports both synchronous and asynchronous interaction. NFC is more power-consuming than RFID.

Turning to Wireless Personal Area Networks (WPANs), Bluetooth [157] is an open standard used for exchanging data over short distances. It is designed for high-datarate communication that can replace wire connections between nearby devices; in its most widely used mode, transmission power is limited to 2.5 milliwatts, yielding a range of up to 10 metres. BLE (Bluetooth Low Energy)[157] is a low-power version of Bluetooth designed for low-data rate communications in the IoT. Both Bluetooth and BLE provide encryption and authentication to protect data transmissions. ZigBee [158], Z-Wave [159], Thread [160], and 6LoWPAN [161], [162] are based on IEEE.802.15.4. ZigBee is an open standard that provides low-cost, low-power, low-data rate communication in a mesh network of up to 65000 connected devices. Its data rate is typically lower than that of BLE, and its range is generally shorter; it is, however, more power-efficient. Z-Wave is a proprietary technology that offers a higher data rate than ZigBee, but a shorter range (typically limited to 30-50 metres), and a relatively higher power consumption. Both ZigBee and Z-Wave provide encryption and authentication to protect data transmissions. The recently announced Thread, is a mesh networking protocol that provides secure communications between IoT devices. It is a low-power, IPv6-based technology that provides reliability through self-healing in case of device failures. 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) is an open standard for transporting IPv6 packets over low-power, lossy networks. It can be used both for device-to-device interaction, and as a gateway for providing Internet connectivity. 6LoWPAN supports a relatively low data rate (up to 20 Kbps) and can hold up large networks with potentially billions of nodes.

Moving to Wireless Local Area Networks, Wi-Fi Ha Low [163], [164] provides a low-power, long-range wireless communication solution for IoT networks. It is based on the IEEE 802.11ah standard and operates in the sub-gigahertz frequency band, allowing for longer range and better penetration through obstacles. To reduce power consumption, Wi-Fi Ha Low features low-power sleep modes and corresponding wake-up mechanisms. ETSI HiperLAN [165] is a European technology standard for wireless local area networks (WLANs). It provides high-speed data transfer rates for supporting multimedia applications (notably video and audio streaming). HiperLAN operates in the 5 GHz frequency band and provides reliable connectivity. Table 5 summarises the main characteristics of the protocols outlined above.

Table 5: Main characteristics of wireless IoT protocols

Protocol	Range		Data rate		Frequency	
	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
NFC	2cm		106 Kbps	424 Kbps	13.56 MHz	
RFID	Depends on frequency but typically in the order of centimetres		Depends on frequency but typically in the order of few Kbps		125 Mhz	5.8 Ghz
Bluetooth	Depends on implementation		1 Mbps	3 Mbps	2.4 GHz	
BLE			125 Kbps	2 Mbps	2.4 GHz	
ZigBee			100 Kbps	250 Kbps	868 MHz	2.4 GHz
Z-Wave	IEEE 802.15.4-based radios, which have a typical range of around 30-100 meters in a home environment		40 Kbps	100 Kbps	Under 1 GHz	
Thread			Depends on HW	250 Kbps	2.4 GHz	
6LoWPAN	10m	100m	20 Kbps	250 Kbps	2.4 GHz	
Wi-Fi Ha Low	1Km		0.3Mb/s	347 Mb/s	Under 1Ghz	
ETSI HiperLAN	100m		23.59 Mb/s		5 GHz	

3.2.10 Cloud and Fog Monitoring

Monitoring is crucial for the efficient performance of cloud and fog computing systems. It can be divided into hosting resources or platform and application monitoring. Resource monitoring involves aggregating data from physical or virtual environments to maintain proper resource availability and detect any abnormal situations. Detecting such situations through monitoring should be the input to fast adaptation decision making and execution by (multi-)cloud and fog management platforms [166]. Software that tracks and publishes this kind of data is also known as monitoring probes and it constitutes a fundamental block of any monitoring system. Using such probes, network and computation aspects are usually published (e.g., Collectd [60], Icinga [61], Nagios [62], Netdata[63], etc.). This is implemented through operating system calls made by a local monitoring agent that collects the data in a predefined recurrent period in order to make them available for processing. Application monitoring measures the health and performance of all application components, providing valuable information to improve the efficiency and timing of computing jobs. The goal is to extract application patterns and ensure service level objectives (SLOs) are met. Therefore, a monitoring probe refers to either a daemon operating on the monitored host or an agent scraping data from monitoring APIs exposed by the resources being monitored (e.g., Java Management Extensions (JMX)) [167]. Depending on: i) how distributed the application components are; ii) whether single or multiple cloud providers are used; and iii) if fog resources or other edge devices are involved, the collection and propagation of monitoring data become perplexing [168], [169]. Cross-layer monitoring is necessary for multicloud and fog computing, but it can be challenging due to dispersed application components and the use of multiple hosting locations that may enable heterogeneous hosting resources (i.e., private/public/hybrid clouds, fog devices). Monitoring probes should be automatically set up and connected to provide an overview of the current context of applications and their dispersed hosting infrastructure.

In addition to metrics collection and their respective challenges, another fundamental aspect of any monitoring functionality is the ability to process incoming monitoring data streams. This processing comprises filtering and aggregation of monitoring data as well as events management and processing with respect to SLOs and service level agreements (SLAs) threshold violation. The challenge is to provide timely, accurate, and trustworthy detection of such violations or recognition of any other anomalies to allow a cloud computing continuum management platform to maintain the desired quality of service (QoS). So, different monitoring topologies have been discussed in the literature [168]–[170] encompassing centralised, hierarchical or even distributed approaches. Such topologies involve both the propagation of data coming from dispersed monitoring probes and the processing of the various monitoring streams. Specifically, according to Stefanidis et al. [168] and Masip et al. [171] a centralized monitoring topology involves only one monitoring server that receives the data flow from all dispersed agents; a hierarchical [170] (or decentralised [171]) topology (e.g., the Event Management System [168], [169]) suggests a set of nodes with their own "local" monitoring server (also known as aggregator) able to act on the data (filtering, persisting, etc.) received from nearby resources, while it can collaborate with other servers in a pre-defined way; and last a distributed topology has been defined [170] by involving the interaction of all monitoring components that share their view of monitoring resources and together maintain a common, synced view of the whole application environment, without using any centralised entity.

In [172] we analysed well-known commercial and open-source monitoring tools to discuss their fit, mainly as monitoring probes, for an advanced monitoring system able to cope with applications deployed across the Cloud Computing Continuum. In another recent study, Costa et al. [170], focused on fog monitoring solutions that could, even partially, cope with the characteristics and challenges of monitoring in the cloud computing continuum. According to the same review, monitoring tools such as Support and Confidence-based Monitoring (SCB) [173], Prometheus[64], Switch [174] and Rule base [175] can cope with several of the challenges of fog monitoring since they are lightweight, multi-platform and accept on-the-fly configurations. Nevertheless, we ascertained [170], [172] that the majority of the available monitoring

[60] https://collectd.org/
[61] https://icinga.com/
[62] https://www.nagios.org/
[63] https://www.netdata.cloud/
[64] https://prometheus.io/

Funded by the European Union

tools follow a centralised approach (e.g., Amazon CloudWatch[65], Azure Cloud Monitoring[66], Datadog[67], OpenStack Telemetry[68], PyMon [176], TEEMon [177], etc.), which is not always adequate for the cloud computing continuum. They usually involve monitoring agents that collect performance-related metric values, in an active manner (intrusive to the monitoring target [178]). Only a few of them (Icinga, SequenceIQ [172], FMonE, SCB, FogMon [170], Event Management System (EMS) [168], [169]) support some sort of distribution regarding the processing topology (i.e., Hierarchical) and therefore may provide a basic support to dynamically distributed monitoring topologies like the ones NebulOus aims to facilitate. Such an approach allows for lower latency and more efficient use of resources, as data is processed closer to the monitored source. This is quite valuable for any low latency application environment where real-time data processing is critical. We consider the support for dynamic and distributed monitoring topology management as imperative for the dispersed and vigorous cloud computing continuum environments.

Moreover, according to Costa et al. [170] the use of the push-based communication model (e.g., Opsview[69], Icinga, Datadog, Collectd, PyMon, Osmotic monitoring [179], etc.) for monitoring may lead to problematic situations, if scalability aspects are not solved, resulting in huge amount of data being injected into the monitoring server. In such cases, communication channels and processing capacity for monitoring purposes can become overloaded and potentially cause inefficacy, data loss or even complete unavailability of the whole monitoring system [177]. It is evident that scalability and fault-tolerance aspects should be considered. Otherwise if not clearly addressed (Pymon, Osmotic monitoring and Switch) the risk of failure is significantly increased. In addition, plenty of these tools focus mainly on the aggregation of monitoring data to mainly visualise them and let the DevOps decide on mitigation actions, based on the current status of the application [172]. Only a few of them support advanced processing capabilities and root cause analysis (i.e., Datadog, Dynatrace, New Relic), which if extended appropriately they can materialise the goals of white-box monitoring [170]. Last, the majority focus on hosting resources and network monitoring which implies the diversity required to cope with different application monitoring types [172]. Since, deploying and maintaining applications over multiple cloud computing continuums, with their inherent distributed nature, one may think that the proper processing topology would always be a distributed scheme. According to Ward et al. [180], distributed topologies have inherent scalability improvements over centralized ones, but they introduce a set of different challenges such as monitoring system bootstrap, node lookup process and monitoring data replication and synchronization [170]. We add to these findings all the security aspects that should be considered when propagating monitoring data across cloud, fog, and edge resources, spanning the strict boundaries of vendors and end-users organisations.

The last point to this analysis acknowledges the fact that as the distribution of cloud application components is increased and the heterogeneity of the hosting resources is expanded, the processing requirements of diversified (in volume, velocity and type) monitoring streams are also amplified. This leads to the requirement for sophisticated tools that follow the complex event processing (CEP) paradigm [168], [169], [176] for digesting and processing multitudes of event streams. Of course, there are already existing centralised CEP approaches for monitoring purposes, but they usually crave for huge bandwidth and computational capabilities, while they lack robustness and scalability, generating a single point of failure [168], [169]. Therefore, distributed CEP architectures [168], [181] [182] are more appropriate in dynamic environments like in ad-hoc cloud continuums. Nevertheless, such efforts are currently bound to the use of simple infrastructures, a fact that limits, the capability to detect reconfiguration opportunities in multi-clouds and edge resources.

[65] https://aws.amazon.com/cloudwatch/
[66] https://docs.microsoft.com/en-us/azure/azure-monitor/overview
[67] https://www.datadoghq.com/
[68] https://github.com/openstack/ceilometer-specs
[69] https://www.opsview.com/

Funded by the European Union

3.2.11 Anomaly Detection

From the data collected by the set of monitoring probes deployed, one of the most relevant and valuable processes is anomaly detection. Anomaly detection [183] is a broad problem, which has been investigated for many years and is applicable in a wide range of domains, with its peculiarities and limitations. Beyond data analysis and artificial intelligence, anomaly detection can be applied in the cybersecurity domain, predictive maintenance, failure prevention, automation, or e-health [184]. Anomaly detection can be defined as the process of identifying data patterns that significantly deviate from expected behaviour, where, in most cases, these anomalies are unusual or difficult to model or predict. The identification of anomalies is vital to be reactive, mitigating and responding to a wide range of issues that can have costly consequences if not addressed in time, such as detecting theft of credit cards, preventing system failures, or anticipating the occurrence of cancer.

In a continuum computing context, the anomaly detection process relies on the data collected by the deployed monitoring probes system, where the type of input data has to be considered, as it requires certain techniques, with associated challenges. Topics such as dimensionality, number of attributes, or relationships between different instances of the data (sequential, spatial, graphical, etc.). The anomaly detection process is further complicated when the data includes other types of data such as evolving data, streaming data, correlated data, heterogeneous data, contaminated data, and big data.

Anomaly detection has usually been approached from a statistical point of view, but the explosion of machine learning opens up new opportunities, especially thanks to the large volume of data that can be gathered for model training processes, which are very attractive in domains where new data patterns are arising that make it difficult to use static models [185], [186]. Statistical techniques assume that data points are generated and distributed according to a statistical model, where any deviation from this distribution is considered an anomaly, with the advantage that the results can be explained and interpreted. These types of techniques can be classified as parametric (Gaussian model, Regression model, Gaussian mixture modelling, Hidden Markov Models, Hypothesis testing...), or non-parametric (k-NN based, Parzen density estimation, string matching, and clustering) [187], [188]. A very common case is when the data have a sequential relationship, generating a time series. Algorithms such as Time Series Analysis (TSA), Auto Regressive Moving Average (ARMA), Integrated Moving Average (ARIMA), Kalman Filtering, etc. are commonly applied in order to analyse the time series and extract statistical information to detect an anomaly [189]. However, we obtain better results when solutions with synergies between statistical models (ARIMA) and deep learning models, allowing for unsupervised anomaly detection [190]. Another conventional approach applied in anomaly detection is the reduction of the dimensionality of the data space, assuming that this reduction can separate normal and anomalous cases in a very clear way. This reduction process can be based on Principal Component Analysis (PCA) [191], or on graphs [192], where the graphs are selected to include structural information about the measured data (proximity). This dimensionality reduction represents a great advantage for high dimensional data processing, even more so when other types of anomaly detection techniques are subsequently applied. The above techniques require a good situational awareness in order to derive a well-quantifiable measure of the anomaly. On the other hand, when the real scenarios increase their time-variability, these techniques do not perform satisfactorily in front of the time-variability of the data models. This means that it is necessary to introduce data-driven techniques that allow for more flexible adjustments and that can respond to the high dynamism and complexity of the data space gathered by the monitoring system.

Data-driven techniques are usually based on learning models, since there is no mathematical model, but there is a large volume of data that allows the most appropriate learning process to be carried out. These types of techniques can be classified into three categories of learning: supervised, unsupervised and reinforcement learning, with a considerable number of success stories. In the case of anomaly detection, supervised learning faces very special challenges compared to its application in other cases. A well-known method for anomaly detection is cost-sensitive learning, such as Metacost [193], Proximity-Based Classifiers [194], Support Vector Machines (SVMs) [195], Decision Trees [196] and Rule-Based Classifiers [197]. The difficulty and cost of the training data labelling process limits the volume of labelled data available so that the introduction of semi-supervised learning is necessary, where a limited set of labelled data and a large volume of unlabelled

Funded by the European Union

data is available [198]. In such approaches, machine learning is applied to a classifier model for the 'normal' data, so that anomalies are incorporated and evolve dynamically. Autoencoders [199] can be used for the learning stage, training the model with the normal data, learning to reconstruct the input with a very small reconstruction error score, and it is at the time of detection that anomalies with a higher reconstruction error are introduced, defining a threshold to identify the anomalous data. On the other hand, unsupervised learning algorithms assume that the anomalies are significantly different from the normal cases [200], with the clustering method being the most widespread and applied, where the anomalies do not belong to any cluster or to a very small cluster [201]. Clustering algorithms can be based on similarity measures or probabilistic modelling, estimating the probability of each data instance [202]. Another interesting approach in anomaly detection is reinforcement learning, being the most similar method to the way humans learn, focusing on assigning positive or negative reinforcement depending on the result obtained, while adapting to the objective of maximising the defined goal, anomaly detection. This type of learning has significant potential in different domains; focusing on anomaly detection, it has been applied in a distributed manner in a hierarchical monitoring probe system architecture [203], as well as in anomaly detection for time series [204]. Finally, there are deep learning methods that enable us to identify patterns that are useful for making predictions about new inputs. In the area of anomaly detection, several types of deep learning networks have been proposed, such as Convolutional Neural Networks (CNN) [205], [206], Autoencoders [207], [208], Generative Adversarial Networks (GAN) [209], [210], Restricted Boltzmann Machine [199], [211] and the Recurrent Neural Network (RNN) [212], [213].

All learning techniques introduce a number of features to the anomaly detection process, but they also involve significant complexity, where learning must be performed on sufficiently resourced components, either at the edge, fog, or cloud, and where there is a growing demand for accurate anomaly detection in streaming and real-time systems. The evolution of both hardware and software resources, as well as the emerging new architectures, make it possible to handle streaming data and to respond to the real-time requirements of applications. These architectural proposals will allow data processing in two different modes: off-line processing and on-line processing. Off-line processing permits the use of complex techniques without limitations of time and computational resources. On the other hand, on-line processing must be performed with limited resources and provide results in a defined time frame, where data must be processed as it is received [184]. The use of one or the other processing approach depends on the requirements of the system. The online approach will be applied when it is required to process streaming data on the fly, in real or near real time. On the other hand, the offline approach will allow more complex tasks to be carried out, when a large volume of data is available and when an immediate response is not required. In the NebulOus context, the real challenge will be to efficiently extract useful information from the large volume of data gathered by the monitoring probes deployed in the infrastructure, limiting considerably the transmission, storage and processing of the necessary data in a continuum computing model, and to apply the most appropriate processing approach (offline or online) to respond to the requirements defined, both in terms of response time and available resources.

3.2.12 Security and Privacy

Edge computing technology helps real-time applications by reducing latency and response time, extending cloud computing capabilities to the edge of the network. With this technology, data is instantly analysed and processed at the edge of the network where it is collected. In this way, data is processed in real time at physically close edges to support data flow acceleration [214]. Currently, many of the applications in IoT environments require real-time response, but IoT computing and storage deployment in the cloud may not be IoT-friendly due to network latency. The solution to reducing latency is to move those resources closer to the IoT sensors that are providing the data or waiting for a response in real time [215]. The appearance of IoT systems and solutions provides more open ecosystems with highly-interconnected heterogeneous devices. This fact has generated many challenges in the field of IoT security [216].

In this context, the security and privacy in the context of NebulOuS assume three main functions. The first one is to ensure the security and privacy of communications. This is established by the creation of secure overlay networks which

Funded by the European Union

encrypt traffic end-to-end. The second one is to establish data protection and privacy, which is achieved by providing logical access control to generated data. The third one is to detect and mitigate threats or potential intruders, which is enabled by the timely detection of such events and appropriate cybersecurity actions.

The state-of-the-art with respect to securing communications can be found in Section 3.2.9. In this section, we outline approaches to tackle the other two aspects by providing the background and recent advancements on a) Access Control mechanisms and b) Intrusion Detection, focusing on Artificial Immune Systems.

We note that closely related to the latter is anomaly detection, but is methodologically treated as a separate field as it is also related to general monitoring aspects which lie outside the scope of cybersecurity. Therefore, the related state-of-the-art has been presented in the previous section.

Access Control

Access Control Mechanisms realize various logical access control models. Those models provide the framework and a set of boundary conditions upon which the objects, subjects, operations, and rules may be combined to generate and enforce an access control decision.

There are several models and mechanisms, each one having its own advantages and limitations. Details on their application to the cloud and IoT edge of the continuum can be found in several papers [217] [218]. In this section, we first provide an overview of the most popular ones, note their advantages and disadvantages, assess their suitability for NebulOuS and present the current technical options (languages, specifications and tools) to implement them.

With respect to the different Access Control models, the main ones are the following:

- **Discretionary Access Control (DAC).** In DAC, the owner of the object specifies which subjects can access the object. Most operating systems such as Windows, Linux, Macintosh and most flavours of Unix are based on DAC models.
- **Mandatory Access Control (MAC).** In MAC it is not the users, but the system that specifies which subjects can access specific data objects. This model is based on security labels. Subjects are given a security clearance (secret, top-secret, confidential, etc.), while data objects are given a security classification (secret, top-secret, confidential, etc.).
- **Identity Based Access Control (IBAC)** uses mechanisms such as access control lists (ACLs) to capture the identities of those allowed to access an object. In the IBAC model, the authorization decisions are made statically prior to any specific access request and result in the subject being added to the ACL.
- **Role-based access control (RBAC)** employs pre-defined roles that carry a specific set of privileges associated with them and to which subjects are assigned.
- **Attribute-based access control (ABAC)** uses attributes, and policies that express boolean rule sets that can evaluate many different attributes before allowing access. ABAC, therefore, avoids the need for capabilities (operation/object pairs) to be directly assigned to subject requesters or to their roles or groups before the request is made. IBAC and RBAC can be seen as special cases of ABAC, with IBAC using the attribute of "identity" and RBAC using the attribute of "role".

The adaptability and expressiveness of ABAC make it ideal for protecting data, thus it is currently the approach that is considered by NebulOuS for implementing its Access Control mechanisms.
In more detail, ABAC allows dynamicity in policy creation, as well as separation of concerns among policy definition and enforcement. Such policies shall be able to update dynamically based on specific constraints that are not a-priori related to requestors. This is in contrast to both a MAC scheme, in which the policy definition point is unique and the resources

have static properties based on which allowance/disallowance is provided, or DAC schemes where the properties are totally static.
Furthermore, in IBAC and Access Control Lists the exhaustive list of authorization decisions should be defined prior to any request. Therefore, even though they support extensible attributes for subjects & objects, the dynamicity offered comes with a severe drawback.
On the other hand, RBAC and ABAC seem more capable to satisfy the needs of NebulOuS. Since RBAC can be seen as a special case of ABAC (in which one subject attribute is addressed as Role), we will hereafter focus on ABAC.
ABAC removes the need for capabilities (operation/object pairs) to be directly assigned to subject requesters or to their roles or groups before the request is made [219]. Instead, when a subject requests access, an ABAC-compliant engine makes an access control decision based on a) the assigned attributes of the requester, b) the assigned attributes of the object, c) the environment conditions, and d) a set of policies that are specified in terms of those attributes and conditions.

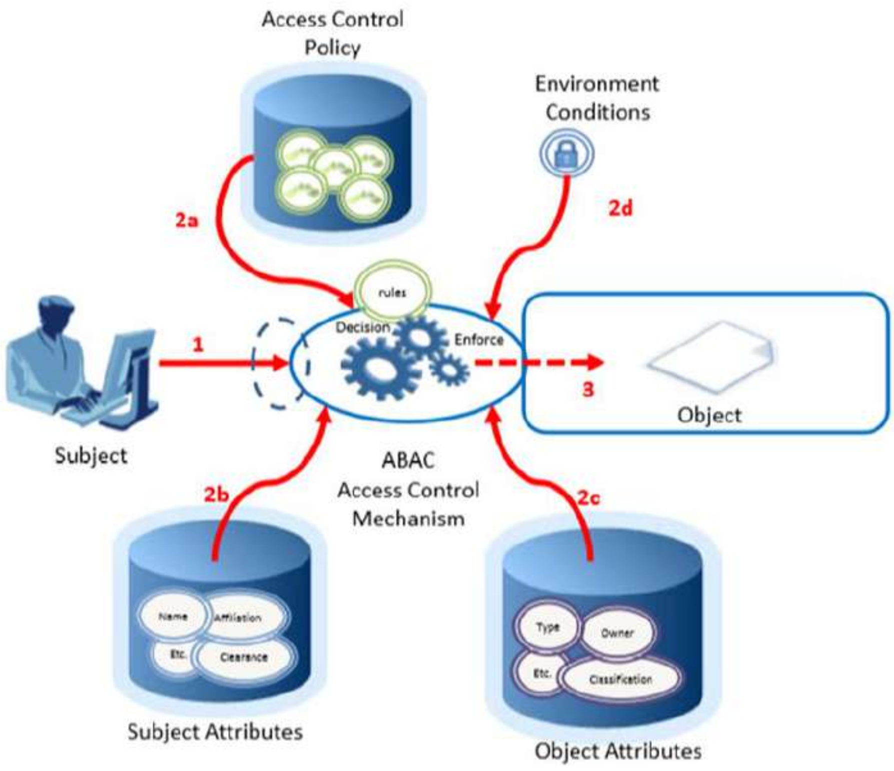

Figure 5: Attribute-Based Access Control (ABAC)

The key standards that implement ABAC are two: the OASIS standard of extensible Access Control Markup Language (XACML) and the Next Generation Access Control (a.k.a. NGAC) standard [220]. Although the former two are the most notable, worthy of mention is also the Abbreviated Language For Authorization (ALFA) [221]. ALFA is a pseudocode language that respects the XACML model and contains the same structural elements as XACML i.e. PolicySet, Policy, and Rule. Its difference is the use of JSON (instead of XML) to define access-control policies, while it can map directly into XACML.

XACML is an OASIS [222] standard that describes both a policy language and an access control decision request/response language. Both languages use XSD [223] notations; hence policy definition and request/response elements are serialized as XML elements.

On one hand, the policy language details the general access control requirements and provides extension points for new functions, data types, combining logic, etc. On the other hand, the request/response language allows the formation of

Funded by the European Union

queries that ask whether a given action should be allowed. The response shall include an answer on the request allowance using one of the following four values:

- Permit
- Deny
- Indeterminate (an error occurred or some required value was missing, so a decision cannot be made)
- Not Applicable (the request can't be answered by this service).

The specification defines five main components that handle access decisions: Policy Enforcement Point (PEP), Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Information Point (PIP), and a Context Handler. With respect to their functionalities, the role of each component is the following:

- **The Policy Administration Point (PAP)** is the repository for the policies and provides the policies to the Policy Decision Point (PDP);
- **The Policy Enforcement Point (PEP)** is the interface of the whole environment to the outside world. It receives the access requests and evaluates them with the help of the other actors and permits or denies access to the resource;
- **Policy Decision Point (PDP)** is the main decision point for the access requests. It collects all the necessary information from other actors and calculates a decision;
- **Policy Information Point (PIP)** is the point where the necessary attributes for the policy evaluation are retrieved from several external or internal actors. The attributes can be retrieved from the resource to be accessed, environment (e.g. time), subjects, and so forth.

Artificial Immune Systems

In this context, the detection of anomalies caused by cyberattacks becomes an increasingly laborious task to identify known anomalies, new zero-day anomalies, or signs of system instability after these anomalies. These new security challenges require new approaches and tools and smarter solutions, with bio-inspired ones having the best position.

Artificial intelligence and in particular learning algorithms appear to be the most appropriate for some cybersecurity purposes. Using learning algorithms, the rapid and accurate detection of malicious behaviour is more feasible than ever. A special branch of this type, includes algorithms inspired by nature and biotechnology. Such algorithms followed from the model's nature, biology, social systems and life sciences. Some examples [224] include genetic algorithms, swarm intelligence, artificial immune systems, evolutionary algorithms and artificial neural networks. These algorithms have a great advantage over traditional learning algorithms, they focus on optimization: nature acts as a method to generate something as perfect as possible, or to choose the most suitable samples from a population. In practice, this family of algorithms applies these principles in the form of optimization and search for the best solution to the assigned problem. In anomaly detection, the main objective is to identify malicious behaviour, so these algorithms use the most appropriate mechanisms to detect malicious anomalies [225]. Another beneficial use of bio-inspired algorithms is the optimization of possible features used in attack detection, for example selecting an optimal set of features for effective malware detection, but also reducing complexity and computational load. Also, bio-inspired algorithms are very flexible, as they can accept a mix of variables in terms of type and duration.

Bio-inspired computing is gaining prominence over time as these algorithms are intelligent, as well as learning and adapting like biological organisms [226]. The lines of biological research about algorithms are being directed towards the combination of several of these algorithms to apply them in different specific functions within the same complex context, in order to obtain better results, since depending on the application that is required one algorithm or another will adapt better, or what is the same, obtaining an intelligent behaviour with better results depending on the context.

Funded by the European Union

One of the most used and promising approaches in these topics is the use of Artificial Neural Networks, which can solve many problems including clustering, regression, classification, and prediction problems. In particular, they have been widely used in malware through pattern recognition and intrusion detection [227]. On the other hand, the evolutionary algorithms are used to solve single-objective, multi-objective, combinatorial and/or non-deterministic problems, optimizing characteristics or parameters to classify attacks [228]. Also, other approach are swarm intelligence algorithms, the self-learning capacity and the adaptability that they present towards external variations in real time turn out to be key factors of these algorithms to understand the attractiveness and interest in different fields of application [229]. Additionally, another approach are artificial immune systems (AIS), providing valuable features: security, resilience, distribution and memory [230]. To that end, its insight capacity for the identification and characterization of anomalies in a context of dynamic systems and rapidly evolving threats is a critical factor.

The main function of a biological immune system is to protect the body from foreign molecules known as antigens. It has a strong pattern recognition ability that can be used to distinguish between foreign cells entering the body (non-self or antigen) and the body's (self) cells. Immune systems have many characteristics such as uniqueness, autonomy, stranger recognition, distributed sensing, and noise tolerance. Most of the living organisms have an immune system and the IoT, despite being a computational system, introduces some novel features that make it resemble a living organism [231]. First, IoT networks consist of a multitude of IoT devices (cells) distributed over a wide geographic area, which can be static or mobile. Second, IoT devices can sense the environment with their built-in sensors and interacting with it through different types of actuators. This changes the context around the device, which in turn introduces new changes at all levels of the IoT implementation. Third, typical IoT deployments lack sufficient computing and processing power to defend against and adapt to all kinds of anomalies. Finally, just as an organ is vital to the proper functioning of the human body, malfunctioning IoT implementations can pose a serious threat to the functioning of an entire system. Maintaining IoT security is a serious challenge, mainly due to the complexity in integrating security mechanisms and the existence of vulnerable and misconfigured IoT systems.

Inspired by specific immunological theories that explain the function and behaviour of the immune system [232], the following four techniques/algorithms arose:

- Clonal Selection: Inspired by the clonal selection theory of acquired immunity that explains how B and T lymphocytes improve their response to antigens over time, called affinity maturation. These algorithms focus on the Darwinian attributes of the theory in which selection is driven by the affinity of antigen-antibody interactions, reproduction is driven by cell division, and variation is driven by somatic hypermutation. Clonal selection algorithms are most applied to the optimization and pattern recognition domains, some of which resemble parallel climbing and the genetic algorithm without the recombination operator [233].
- Negative Selection: inspired by the positive and negative selection processes that occur during T cell maturation in the thymus called T cell tolerance. Negative selection refers to the identification and elimination (apoptosis) of autoreactive cells, that is that is, T cells that can select and attack own tissues. This class of algorithms is typically used for classification and pattern recognition problem domains where the problem space is modelled on the complement of available knowledge. For example, in the case of an anomaly detection domain, the algorithm prepares a set of exemplary pattern detectors trained on normal (not anomalous) patterns that model and detect invisible or anomalous patterns [234].
- Immune Network: inspired by the "idiotypic" network theory proposed by Niels Kaj Jerne that describes the regulation of the immune system by "anti-idiotypic" antibodies (antibodies that select for other antibodies). This class of algorithms focuses on the network graph structures involved where antibodies (or antibody-producing cells) represent the nodes and the training algorithm involves growing or pruning edges between nodes based on affinity (similarity). in the representation space of problems). Immune network algorithms have been

Funded by the European Union

used in domains of clustering, data visualization, control, and optimization, and share properties with artificial neural networks [235].

- Dendritic Cells: Inspired by the immune system developed with a multi-scale approach. This algorithm is based on an abstract dendritic cell (DC) model. The Dendritic Cell Algorithm (DCA) is abstracted and implemented through a process of examining and modelling various aspects of DC function, from the molecular networks present within the cell to the behaviour exhibited by a population of cells as a whole. Within DCA, the information is granulated into different layers, which is achieved through multi-scale processing. It has turned out to be one of the best approaches for solving intrusion detection problems, representing DCs as detectors of different control sites in the body, as well as mediators to induce a variety of immune responses [236].

Based on the human immune system, assimilating in this way to the characteristics that they present; highly evolutionary, functioning in a parallel and distributive manner, robust and being able to easily adapt to any environment, the concept of AIS is built, in which they want to extend the capacities provided by the immune system and which turn out to be very useful in contexts such as in detecting anomalies [237]. The main aspects of the immune system that make this concept interesting and try to contextualize it in computational paradigms are recognition, feature extraction, diversity, learning, memory, distributed detection, self-regulation, meta dynamics and immune network [238]. In this type of dynamic environment, defence mechanisms of the same type must be implemented, that is, at runtime. In this way, the approach is focused on monitoring and analysing the characteristics of the system to identify attacks and errors at an early stage, having the ability to analyse the cybernetic and physical behaviours of the system, comparing the observed behaviour with the predictions or adequate behaviour. It can be safely stated that nature is the most appropriate entity to solve difficult and complex problems, since it can find the most appropriate solution.

Funded by
the European Union

4. The NebulOuS architecture

4.1 Methodology

The NebulOuS architecture was defined in the context of Task 2.3. Its design incorporates the input provided by Tasks T2.1 "*Scientific and technological State-of-the-art analysis*" and T2.2 "*High-level requirements analysis*", the results of which are reported in Sections 2 and 3 of this Deliverable.

To define our system's architecture, we have followed a design approach which depicts our system using 3 different levels of abstraction: The conceptual design, the logical design and the physical design. Each one of those designs depicts our platform in an increasing level of detail.

The conceptual design is a high-level view which aims to depict the main concepts and functionalities being offered and contains no implementation details. It is mostly non-technical and can communicate the general concept to non-experts. The logical design includes the logical components which comprise our system and is closely tied to the software artifacts that are going to be implemented. It describes the basic flow of information between the components, but does not yet bind them to any specific implementation technologies or tools. The physical design is a low-level depiction of the actual implementation of our system, including all technical details. We note that this Deliverable only reports the conceptual and logical NebulOuS architecture. The physical architecture will be reported in a later stage of the project, as it is tied to the results of technical tasks which have not yet started.

For the design process, we followed a top-down approach: First, we designed the conceptual architecture of our system, grouping the individual functionalities that are going to be supported by the NebulOuS Meta-OS into coarse-grained blocks. Subsequently, we began to de-compose those building blocks into more fine-grained components, adding further details until the final logical architecture was specified. Furthermore, our approach was an iterative one. Based on the updated input received by the requirements and SotA analysis which was conducted in parallel, we integrated the corresponding findings into our architecture and released new, refined, versions regularly.

Funded by the European Union

4.2 Architecture

4.2.1 Conceptual architecture

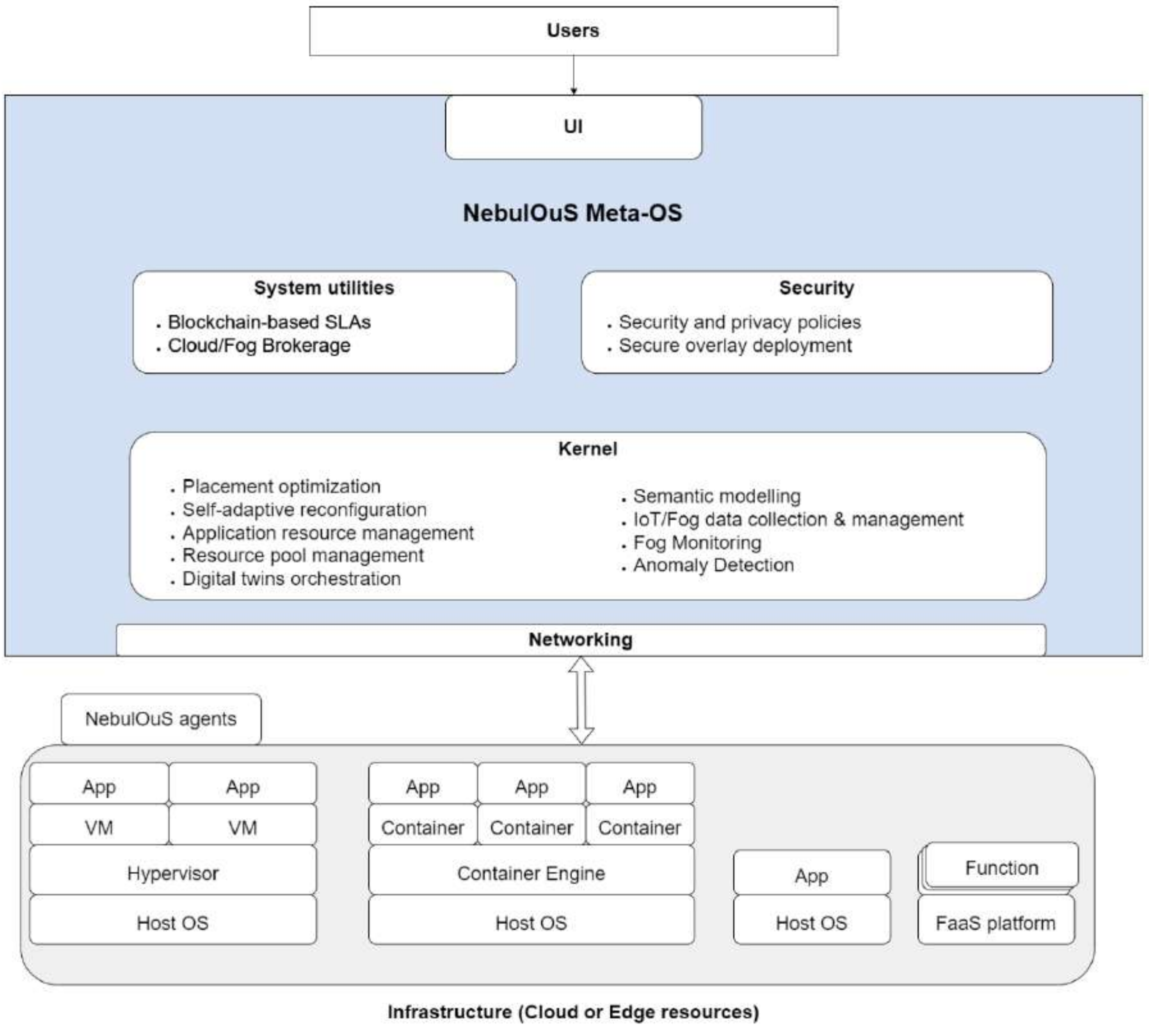

Figure 6: The NebulOuS conceptual architecture - overview of supported functionality

Figure 6 depicts the NebulOuS conceptual architecture. In this figure we provide a high-level overview of the NebulOuS Meta-OS and present the functionalities that will be provided by each part of the Meta-OS. The figure is composed by two main parts: a) the NebulOuS Meta-OS and b) the different types of underlying resources that can be managed by NebulOuS.

With respect to the underlying infrastructure, NebulOuS will be able to deploy applications across the entire Cloud-to-Edge Continuum, supporting a wide variety of resources from the Data Center to the User Edge[70]. This includes the entire spectrum of available options, from container-based deployments to VM-based ones, even "raw" bare metal deployments. Furthermore, NebulOuS will not only accommodate "serverful" approaches, but also the serverless paradigm, allowing users to optimize the execution of their functions using our Meta-OS.

The NebulOuS Meta-OS itself is comprised of the following components: the kernel, UI, networking, security and utilities. The security mechanisms are used to ensure secure access to the resources, also including privacy aspects when desired by

[70] https://github.com/State-of-the-Edge/glossary/blob/master/edge-glossary.md#user-edge

Funded by
the European Union

the NebulOuS users and expressed in the form of relevant policies. The UI enables user interaction with the Meta-OS and will include a Graphical User Interface. The networking mechanisms interconnect the different resources and ensure that communication can be successfully and efficiently established between the devices (e.g. by creating secure tunnels and overlay networks). The NebulOuS kernel forms the core of the architecture and can be conceived as a logically-centralized control plane, which continuously monitors the resources and dictates the service orchestration process on the entire continuum. Finally, two system utilities will be offered by the NebulOuS Meta-OS to its users: a) the ability to establish blockchain-based Service Level Agreements and b) the ability to leverage the Cloud & Fog service brokerage mechanism for the establishment of ad-hoc clouds which involve the infrastructure of different organizational units or telecom providers, according to the user preferences.

4.2.2 Logical architecture

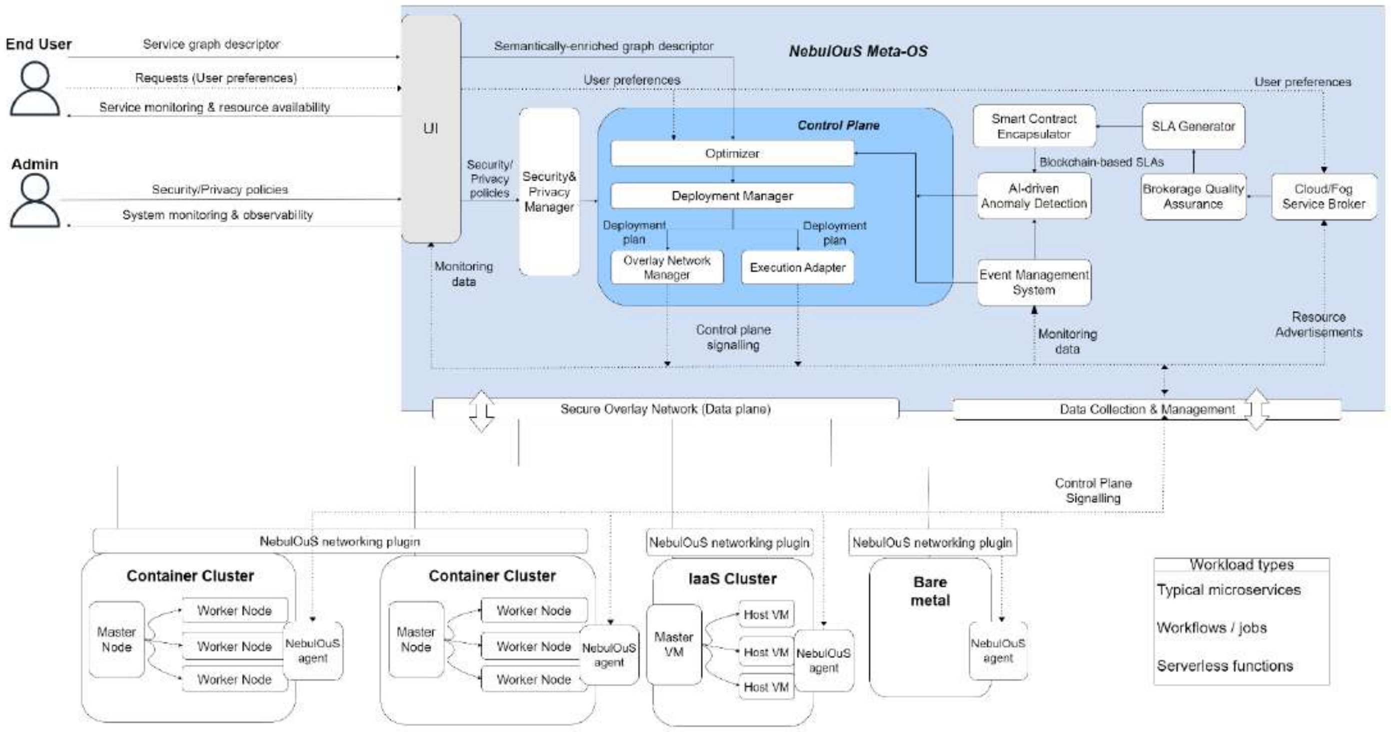

Figure 7: The NebulOuS logical architecture

Figure 7 depicts the NebulOuS logical architecture. This figure provides an overview of the different components which comprise our platform, as well as the main flows between them. We will hereafter give a general overview of its design, reserving specific details on the role and functionality of the individual components for Section 4.3.

Before delving into the actual architecture, we will first present the different actors that interact with NebulOuS. To this end, we consider two main types of NebulOuS users:

- A regular "end user", which assumes the role of a cloud continuum consumer and uses NebulOuS to deploy a workload in the cloud-to-edge continuum (typically a software developer or DevOps). Those users provide NebulOuS with a service description, along with all related requirements and constraints (e.g. the hardware architecture that the service can be deployed on). They also declare their preferences regarding service brokerage in the form of requests that are going to be handled by the NebulOuS Meta-OS.
- An "admin" user, related to an entity that has deployed NebulOuS on its premises and advertises its available resources to be brokered. Instances of such entities in the context of NebulOuS can be telecom providers which onboard compute resources from their edge locations, organizational units which make their private resources

available for intra or inter-organisational collaborative infrastructure sharing scenarios, etc. Those users provide NebulOuS with various policies (e.g. security, access control) to be enforced by the Meta-OS.

Both of the aforementioned actors use NebulOuS to intermediate between computing resources and software, the same way that a typical Operating System is being used. The NebulOuS users interact with the platform via the NebulOuS User Interface (UI). This UI will incorporate graphical interaction (i.e., a GUI) and will provide each user with all the necessary information on the state of the deployed services and available resources. To this end, we will consider various tools for the observability, telemetry and monitoring aspects and provide visualizations of the relevant data to the users using a dashboard-like approach.

To manage security and privacy aspects, a Security and Privacy Manager lies between the UI and Control Plane. This component receives user-defined security and privacy policies by the UI and undertakes all necessary actions to ensure that the policies will be evaluated, deployed and enforced. Those policies can provide fine-grained access control to protect user access to sensitive data, as well as network security by restricting communication between application containers.

In the center of the NebulOuS Meta-OS lies its Control Plane. At this point, it is worth to note that NebulOuS employs a layered approach to orchestration, assuming it is performed on two different levels: a local and on a global one. Local orchestration (or L1) is performed by a service orchestrator which manages the deployment and execution of a distributed application within a single cluster, which will typically span a single resource provider (e.g. cloud provider). In the case of containerized workloads, Kubernetes may play the role of the local service orchestrator.

However, NebulOuS as a Meta-OS aims to unify resource management spanning different resource providers under a single control plane, aiming to globally optimize application placement on a multi-cloud, multi-cluster setting from the cloud to the edge. Such a setting may consist of several local orchestrators being deployed in the different cloud providers and/or clusters. As a result, a global layer of orchestration is also needed (L2).

Layer-2 orchestration functionality is exactly what is provided by the NebulOuS logically-centralized Control Plane, which acts as the Meta-OS kernel by intermediating between the various infrastructure and application resources located in different clusters and optimizing their placement on a global level. As such, it is envisioned that although the two different levels will have a certain degree of autonomy in their operation, the global NebulOuS Control Plane will be able to override local orchestrator decisions, if needed. This approach allows L2 orchestration to achieve globally-optimal setups when available, while L1 can still operate locally even if communication to the second layer is unavailable. Such an approach aims to realize the Fog Computing paradigm by combining the performance benefits of centralized approaches (usually employed in the cloud) with the resiliency offered by distributed/decentralized ones (frequently employed at the edge)[239].

As a central component, the NebulOuS Control Plane is responsible for the service deployment, orchestration, optimization and real-time reconfiguration across the entire cloud-to-edge continuum. Based on the distributed applications' design time requirements, it deploys the service graph on the available cloud/edge resources and continuously monitors the latter (triggering reconfigurations at runtime, if needed) to ensure that the services operate optimally. Application placement is optimized in a dynamic manner (when relevant), so that all applications meet their performance targets and the infrastructure is not over-subscribed. It evaluates the severity of situations forwarded to it by the AI-driven anomaly detection mechanism, or the EMS and triggers reconfiguration action(s) suggesting either application-level or platform level adaptations. Furthermore, it also allows for real-time scheduling of functions and workflow tasks, using traditional and AI-based scheduling approaches. To achieve this functionality, it is composed of various modules: the Optimizer, the Deployment Manager, the Overlay Network Manager and the Execution Adapter. The function of each individual component is elaborated in Section 4.3.

Funded by the European Union

There are also several other components which comprise the NebulOuS Meta-OS and intermediate between the NebulOuS Agents and Control Plane. Those modules implement different functionalities such as Data Collection and Management, Event Management, Anomaly Detection, blockchain-based SLAs etc. The most important one is the brokerage functionality, which is served by two components: the Cloud/Fog Service Broker and the Brokerage Quality Assurance mechanism. By receiving advertisements for the infrastructure and services that can be made available, as well as the preferences of end users with respect to their services and the available resources being brokered, it enables the formation of ad hoc clouds between different organizations, or different units of the same organization.

Last, communication on the NebulOuS Meta-OS is mainly concerned with two aspects: connectivity establishment between the infrastructure resources and data management in the entire continuum. To this end, we include two main components which realize this functionality: the Overlay Network Manager and the Data Collection and Management component. The former establishes tunnels between the different physical and virtual devices, to create secure overlay networks. The latter assumes the role of a communications middleware, capturing raw IoT data streams and managing data flow details, while also handling the exchange of control plane information between the distributed agents and the logically-centralized orchestration components of our Meta-OS. The data plane will be established using current state-of—the-art virtual networking approaches, focusing on programmable and software-defined networks.

4.2.3 Interactions and flows

In this Section, we focus on providing some more specific details on the way the different NebulOuS components are connected and interact with each other. To do so, we first provide a UML diagram which includes the logical architecture along with the main interfaces that are exposed by the components, with respect to their internal interaction. Subsequently, we provide some sequence diagrams which demonstrate information flows between the components on three main scenarios: a) service deployment by a user, b) service re-configuration by the NebulOuS platform and c) cloud service brokerage (including SLA generation and compliance monitoring). Last, we present our approach on inter-component communication within the NebulOuS Meta-OS, focusing on the mechanisms that are currently considered for this purpose.

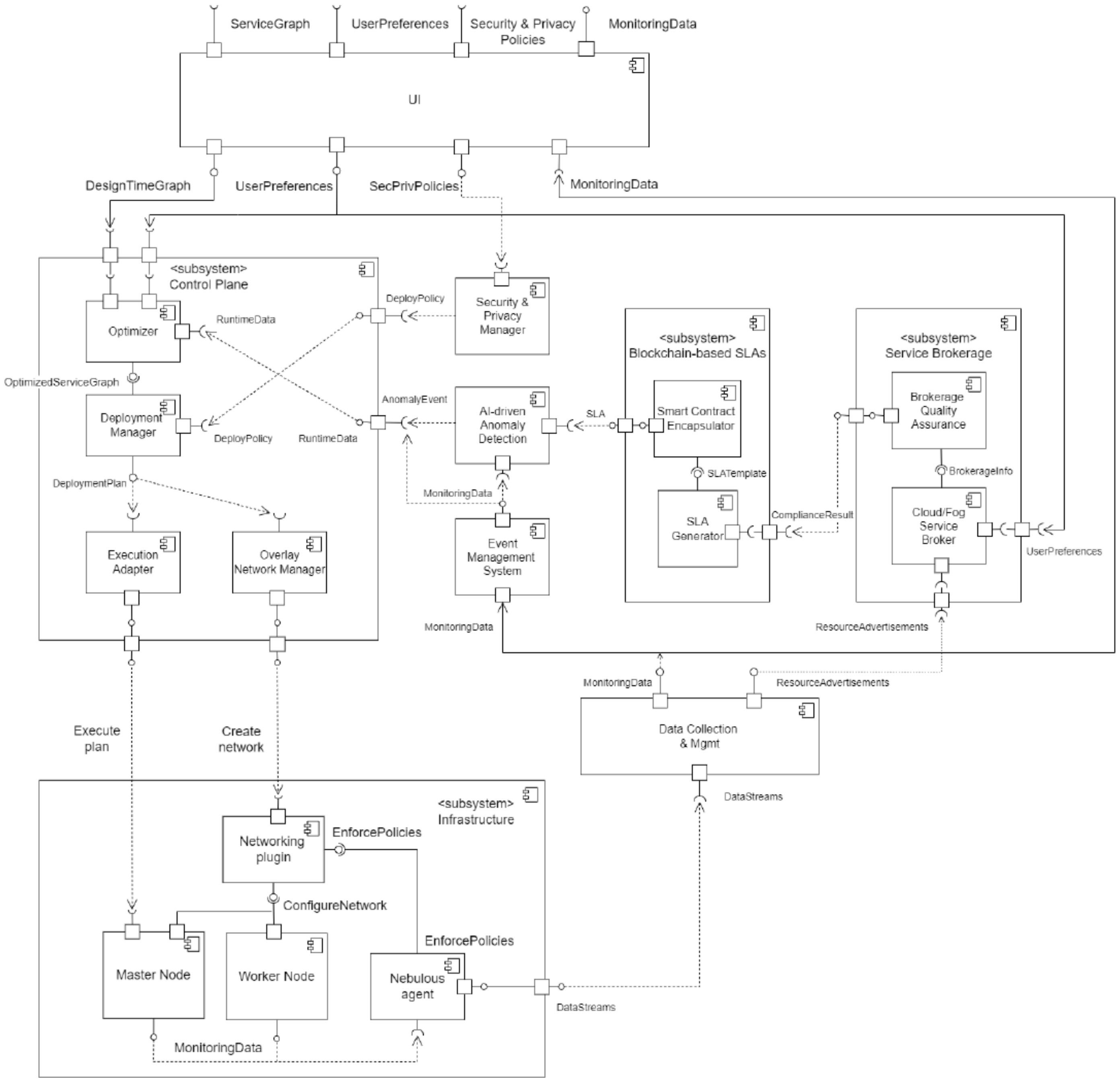

Figure 8: Component interactions in the NebulOuS architecture

The NebulOuS end users interact with the Meta-OS through the UI. Via the UI, users can provide the service graph, describe their preferences and constraints regarding the application placement, define security and privacy policies using the respective interfaces that will be provided. Furthermore, the end users will be able to view observability information regarding the status of their services and the available resources.

The UI forwards the design-time graph received by a user to the Control Plane, specifically received by the Optimizer component. User preferences are also forwarded to the Optimizer, as well as to the Cloud/Fog Service Broker. Apart from the aforementioned design-time information, the Optimizer will also include an interface to receive runtime information about the system status. Based on those inputs, the Optimizer will perform the optimization process and output an Optimized Service Graph, which will be provided by the component.

Funded by the European Union

The Optimized Service Graph is required as an input by the Deployment Manager. The latter also exposes an interface to retrieve security and privacy policies. Based on those, the Deployment Manager generates a deployment plan that is used by the Execution Adapter and Overlay Network Manager components.

The Execution Adapter and Overlay Network Manager interact directly with different parts of the underlying infrastructure, the former managing the resource pool and the later managing the secure overlay network.

Moving on to the infrastructure being managed by NebulOuS, this will typically consist of compute clusters which are comprised by master and worker nodes. A NebulOuS agent will be deployed in those resources for policy enforcement and monitoring data collection purposes. Once collected by the NebulOuS agent, the monitoring data will be streamed to the Data Collection and Management component.

The Data Collection and Management component can then separate those streams, based on their purpose. Two main branches are currently considered: One includes resource advertisements, which will be sent by the NebulOuS agent to indicate resource availability information. This information will be received by the Cloud/Fog Service Broker and used to establish the brokerage functionality. The second one includes the 'pure' monitoring information, which is directed towards a) the end users, for observability purposes and b) the control plane, to enable real-time application reconfigurations.

In the observability case, the monitoring information is received by the UI and provided to the users by a dedicated interface. In the real-time reconfiguration case, the monitoring information is received by the Event Management System (EMS). The EMS, in turn, exposes that information to the AI-driven Anomaly Detection component, as well as to the control plane directly. The Anomaly Detection uses that information to detect possible anomaly events which, upon detection, are also forwarded to the control plane. Both are eventually received by the Optimizer module which will, in turn, decide whether there is a need for re-configurations and further optimizations.

Apart from the previous loop, there is also a second internal loop related to Cloud/Fog Service Brokerage. As already stated, this loop starts with the Cloud/Fog Service Broker receiving the user preferences and resource advertisements. Based on that input, the Cloud/Fog Service Broker makes the appropriate decisions and then exposes that information to the Brokerage Quality Assurance component. The latter performs the necessary checks and exposes the compliance results. The quality-ensured brokerage information is then received by the SLA Generator, which uses that information to generate SLA templates. Those templates are ultimately received by the Smart Contract Encapsulator component, which generates the finalized blockchain-based SLAs.

Those SLAs are exposed and used by the AI-based Anomaly Detection, along with the available monitoring information, to assess SLA compliance. In case SLA violations are detected, an anomaly event is generated and passed over to the control plane for any further actions.

Sequence diagrams:

To better illustrate the basic information flows between the NebulOuS components, we hereafter provide three sequence diagrams. The scenarios depicted are:

- Service deployment by an end user
- Service re-configuration by the NebulOuS platform, based on the available monitoring information.
- Cloud service brokerage, including the process of generating blockchain-based SLAs and monitoring SLA compliance.

Funded by the European Union

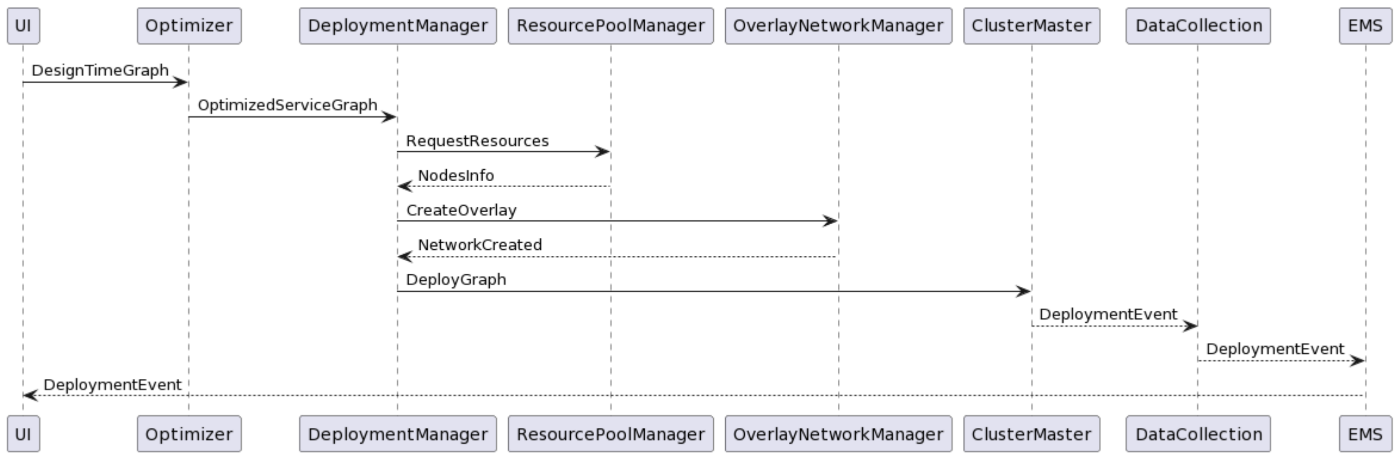

Figure 9: Service deployment sequence diagram

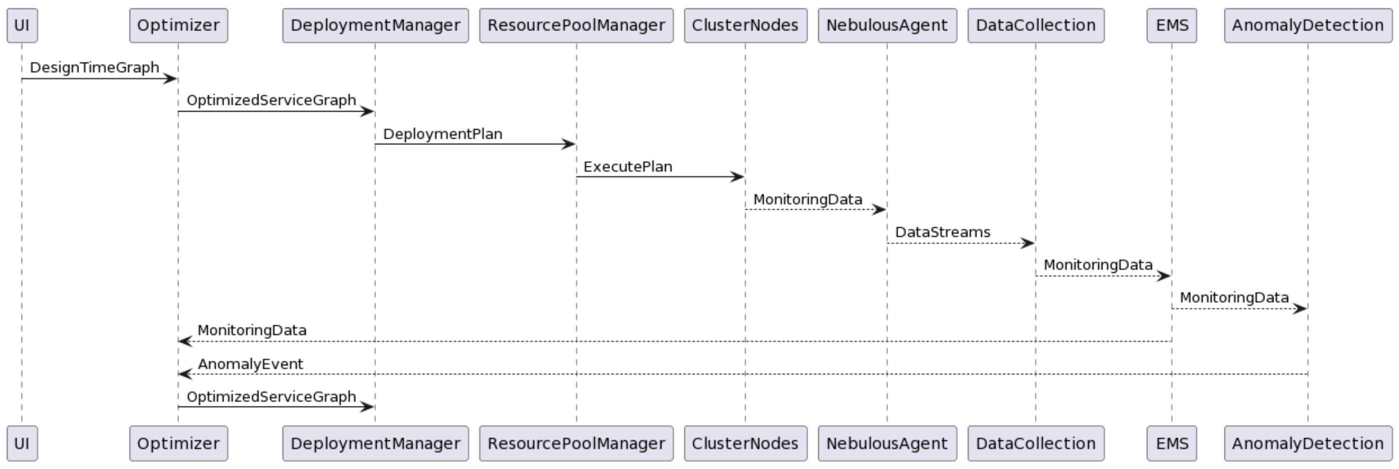

Figure 10: Real-time reconfiguration sequence diagram

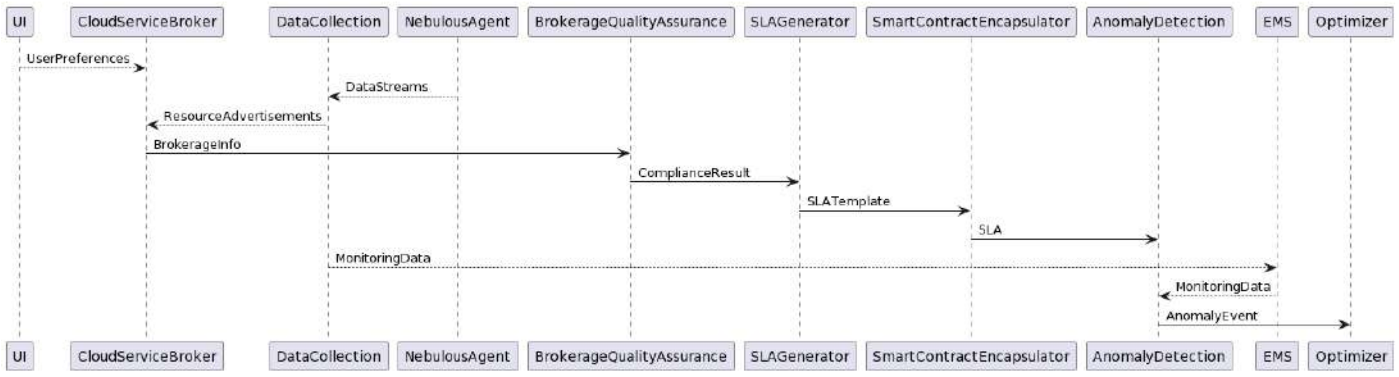

Figure 11: Cloud service brokerage and SLAs sequence diagram

General approach to component interaction

There are multiple ways in which the communication among NebulOuS components can happen. The various services need to exchange data and commands. Typically, the rough classification of communication patterns is between synchronous and asynchronous. Synchronous are often chosen due to their simplicity and seemingly easier state management. However, they suffer from flexibility and reliability issues. Asynchronous communication patterns can be made more flexible and robust if they are properly designed and supported by the applications. However, they require that the state is managed in a more sophisticated way: either outsourced or handled inside of the application. There are also ways to improve the shortcomings of synchronous communications by employing solutions like service meshes (described in SotA) which offload some of the associated issues, like queuing and smart backoff. In NebulOuS, we are currently investigating the approach to be followed for each component, yet we aim to arrive at a clean approach, proposing a single flavour of communications. Among the two, asynchronous communication is the option that is currently considered, owing to its generally larger potential to support scalable message passing between heterogeneous components.

Even with asynchronous communication, the interface design process does not end there. The actual implementation of asynchronous communication might differ widely. One common option is to use a Pub/Sub mechanism for publishers and subscribers. This might be preferable especially for the monitoring data that is collected in real time. In Pub/Sub, again, there is a choice between brokered and non-brokered message queues. The brokered/non-brokered choice affects the reliability and scalability of the final solution and thus the decision must not be taken lightly. Similarly, the next step would be to choose the protocol, typical examples being AMQP and MQTT, and the actual implementation of clients (and brokers if applicable). Finally, it's worth noting that K8s API-based approach is also a form of Pub/Sub when considered from the watched resources perspective (as opposed to the control loop).

4.3 Components

4.3.1 User Interface (UI)

The User Interface component will be the main point of entry for end users and system administrators to interact with the NebulOuS platform. It will provide a user-friendly interface that allows for intuitive interaction with the system, reducing the learning curve and making it easier to understand and use the capabilities provided by the Meta-OS. It is related to *Task 3.5: NebulOuS Meta-OS Graphical User Interface.*

a. Role & Functionality

As the process of deploying and managing applications can be complex, a GUI will help simplify the experience for end-users, by providing tools that will enable the definition of their workloads and respective requirements, the management of the deployed services, as well as visualization of metrics and alerts that will be deducted through the monitoring process. We can categorize the functionalities that will be available through the UI in the following manner:

- **Configuration**

The GUI will provide a range of customization and configuration options, allowing users to tailor the service to their specific needs and requirements, thus generating SLAs which will be tracked and validated (via QoS monitoring).

- **Monitoring**

It will allow end users to have a clear and concise overview of their deployments, making it possible to manage and monitor the health and performance of their applications, the usage of consumed resources, like cpu/ram/bandwidth, as well as be alerted of any deviations of the defined SLAs, or any related event particularly to their requirements.

- **Visualization**

The GUI will display a visual representation of the deployment architecture, providing a clear understanding of the components and relationships involved. The monitored resources capacity/utilization will also be visualized through graphs accessible by respective dashboards.

- **Security**

Administrators of the platform will be able to define their security policies throught the DSL, for their provided infrastructure that is registered and advertised for consumption by the end-users.

During the development of the user interface we aim to create a controller layer that will expose most of the logic as an API, allowing 3rd party systems to integrate and programmatically utilize the NebulOuS and extend the functionality of their systems in their own manner.

b. Requirements Mapping

Table 6: User Interface - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_03	The system shall provide a graphic user interface for the administration of the deployed components	M	A GUI will be implemented that will provide administrative functionalities to the deployed components
F_06	The NebulOuS platform provides infrastructure monitoring capabilities.	M	Data visualization of the monitored infrastructure will be provided through the Graphical User Interface via graphs in Dashboards.
F_07	The NebulOuS platform provides application components lifecycle management for all components of the application regardless of where the component is deployed (cloud, edge, IoT device). It is able to start, stop components and react to monitoring alerts (if a component requires restarting for example).	M	Users will be able to manage the deployed components through the User Interface.
F_08	The NebulOuS platform provides a deployment mechanism that makes it easy to deploy and manage data processing pipelines across various resource types (cloud/edge/fog).	M	Users will be able to define their workloads and pipelines via configuration files and descriptive models provided through the UI.

4.3.2 Security and Privacy Manager

The Security and Privacy Manager is the NebulOuS component that manages the various security and privacy policies defined by admin users. It is related to Task 4.5 Security and Privacy-by-design in data streams propagation.

a. Role & Functionality

Preserving security and privacy in cloud and fog environments is challenging due to the broad attack surface that stems from the nature of applications deployed in such environments. Those applications typically include multiple independent microservices that use virtualization techniques to run over multi-tenant resources. Current state-of-the-art solutions for preserving security and privacy in the cloud require considerable developer effort to implement authentication and authorization mechanisms, as well as appropriate data encryption schemes. This is a fault-prone process that requires support by the infrastructure provider. As such, their use in applications whose components might reside in different clouds or edge locations is not an option.

To cover this need, NebulOuS will implement the Security and Privacy Manager, a policy engine for ad-hoc multi-cloud and fog environments. This component lies between the UI and Control Plane; it receives user-defined security and privacy policies by the UI and undertakes all necessary actions to ensure that the policies will be evaluated, deployed and enforced. Security and privacy by-design will be realized by offering a) a context-aware attribute-based access control (ABAC)

Funded by the European Union

enforcement mechanism to restrict access to deployed application components; b) a service for articulating access control policies at the code-level of the deployed application through appropriate security annotations.

To achieve the aforementioned security and confidentiality of the services and the provisioned resources, we are going to introduce an annotation mechanism that will facilitate developers in defining access policies for application components that are deployed on cloud, fog and edge resources; these policies will naturally reflect the security requirements that developers are interested in imposing on their applications. Using such an approach, the access rights of the platform and application will be bound to a number of dynamic policies.

The Security and Privacy Manager will examine the annotations and will enforce privacy/security policies at certain PEP (Privacy Enforcement Points) of an application or workflow task executed at Fog. Modelling and execution of XACML compliant policies will be supported to protect access to data, as well as eBPF-based network security policies which will be used to restrict pod communication in Kubernetes environments.

The security and privacy mechanisms will be based on and extend our previous work on the PaaSword [240] and PUZZLE [241] H2020 projects. State of the art tools will be leveraged for the implementation, such as WSO2 Balana [242], Keycloak [243] and the Drools rule engine [244], along with recent developments in the cloud-native ecosystem (e.g. Open Policy Agent [245] and Cilium Policies [246]).

b. Requirements Mapping

Table 7: Security and Privacy Manager - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_02	The NebulOuS platform offers a lightweight resilient event pub/sub mechanism with persistence, access control and consumer-group capacities to orchestrate communication between the different modules of the solution. The system shall provide events publish and subscribe based communication "bus" for interaction of components	M/R	The Security and Privacy Manager will realize the aforementioned access control capabilities, which will be used by the pub/sub system .
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	M	The Security and Privacy Manager will offer Identity and Access Management (IAM) capabilities. We will use ABAC/RBAC to provide fine-grained access control based on user attributes (e.g. roles)
NF_05	Data transferred between edge nodes and between edge nodes and clouds should be secured	M	Although this requirement will be mainly covered by the Overlay Network Manager component, the network security policies deployed by the Security and Privacy manager will be used supplementarily to secure traffic flows between virtualized resources (using whitelists/blacklists).
NF_11	The NebulOuS platform ensures data privacy when transferring data from one node to another.	R	Data protection and privacy will be realized by providing logical access control to generated data. The Security and Privacy Manager will be the main NebulOuS component that realizes those access control mechanisms.

4.3.3 Optimizer

The Optimizer component is a part of the NebulOuS control plane. It relates to Tasks 3.3 *Applications Placement Optimization*, Task 3.4 *Real-time scheduling of functions & workflow tasks*, and Task 5.4 *Self-adaptive and proactive reconfiguration enhancement*.

a. Role & Functionality

A typical NebulOuS application consists fundamentally of a set of containers pre-grouped into microservices where each microservice implements a given transformation of input data to output data. The role of the components collectively referred to as *the Optimizer* is threefold:

1. To decide on the placement of the microservices ranging from the core Cloud data centres to the edge devices.
2. To decide on the multiplicity of microservices and duplicate microservices that are bottlenecks in the application.
3. To provide the application with more resources with the right capabilities in the right location to execute the application's microservices.

Functionality

A rational decision will maximise the decisions makers *utility* [247], and it is therefore mandatory that the optimizer has a way to assess the utility seen from each application owner of a particular application configuration prior to enacting this configuration. The best way to represent the application owner's utility is arguably through a representative utility function [248]. This will allow the optimizer to find the combined set of resources and application configuration that maximises the application's utility.

A *workflow* application has temporal constraints among its microservices. This makes the optimisation problem more difficult as the temporal constraints impose an ordering of the microservices, and consequently temporal bounds on the resources needed. Workflow applications are typically scheduled with a *Critical Path* (CP) heuristic [249], and the *partial critical path* (PCP) approach can be used to meet and distribute a given execution deadline over the microservices [250]. Serverless functions can be treated differently since they do not suffer from initial initialisation lags, and so they can be deployed almost as of need. If the same application workflow is repeatedly deployed, reinforcement learning can be used to further shorten the scheduling time, making the serverless function deployment almost instantaneous.

Detecting the need for reconfiguration

The Event Management System (see Section 4.3.12) continuously monitors the state of an application and the resources used to execute the application. The metric values used in the utility function or in the optimisation problem constraints are collectively referred to as the *application's execution context*. This context changes whenever a monitoring event occurs that changes directly or indirectly one of the metric values of the context. The monitoring events recorded can be discrete events or sampled continuous function values. An example of the former is a new user starting to use the application, and an example of the latter can be a reading of the CPU load every second, i.e., the CPU load is sampled with a frequency of one Hertz.

There are two types of constraints:

1. Resource constraints based on agreements with the resource providers. These are often captured as Service Level Agreements (SLAs), and whenever a Service Level Indicator (SLI) metric is updated, it will trigger a severity estimation to evaluate if an SLA violation is probable.
2. Topological constraints related to the application's deployed topology independent of the resources agreed with the providers hosting the different components, but possibly linked to monitored metric values. Consider for example a microservice capable of serving a certain number of users. The number of users is a monitored metric value. If the upper bound of users per microservice is violated by more users arriving, new instances of the microservice must be deployed. Another example is the constraints that there should be twice as many microservices of type 'A' than of type 'B', and so whenever a new microservice of type 'B' is added because of other constraints, two instances of microservice 'A' must be added. Other topological constraints can be data related, so that if the monitored input volume from a certain data source exceeds a given threshold a new microservice processing the data must be added.

Funded by the European Union

A violated constraint indicates that the current deployment is no longer feasible and appropriate adaptation actions must be decided as a reconfiguration is needed.

It should be noted that anomaly detection relates to the individual metric values, and should be done prior to assessing the constraints and the feasibility of the current deployment. Abnormal events should be ignored for the purpose of optimisation, unless there are explicit constraints directly relating to the detection of abnormal events.

Reactive and Proactive Optimization

The assessment of the constraints described above has no concept of time. The constraints are simply evaluated whenever a metric value changes if that metric value is used in at least one of the constraints. This allows the optimization to be done in two ways:

1 Reactive to the change in a metric value at the time the change happening. This is a safe mode because it the observed values are as good as the accuracy of the sensors monitoring the values. However, as it may take time to find a new configuration optimal for the current application execution context, and thereafter some time to adapt and reconfigure the running application, the response there will be a lag before the new state of the application is active. During this reconfiguration lag, other events may happen forcing new, immediate reconfigurations responding to these.

2 Proactive optimisation is basically a reactive optimisation to a forecasted future state of the application execution environment. In other words, all of the metric values involved in the constraints and the utility function will be forecasted to a future time point to allow ample time for the optimisation and the reconfiguration so that the adapted application is already running by the time it is needed. There are two issues related to proactive optimisation: The first relates to the forecasting itself. One cannot expect that the predicted metric values will perfectly match the future metric value, and this may cause wrong optimisation actions to be taken resulting in an enacted application configuration not matching the future need. An example of this is a situation where a few events all indicates an increase in the number of users of an application. Then it is easy to predict that this increase will continue, and give more resources to the application and replicate the application services. However, it was a brief temporary increase in the number of users as some of the active users will soon end using the application. Then, potentially costly reconfiguration actions have been taken unnecessary. The second issue is related as the forecasting may trigger instabilities if the predictions are very much off target, there may not be an allowed feasible configuration for the predicted future situation. The reconfiguration will then be the best possible, but not the one that would satisfy the predicted needs.

Stateless or stateful optimization

Stateless optimizers will start from the current application configuration when it is detected that the current configuration is no longer feasible by violating one or more constraints of the problem. It will then search the feasible solution space for the configuration that maximises the application's utility function, and pass this on to the application adapter to reconfigure the running application accordingly. It must be acknowledged that the optimization problem can be non-linear, and most likely the problems are discrete owning to enumerated domains of the decision variables. For instance, one cannot deploy 3.1415 cores. The problem is that discrete decision variables lead to combinatorial optimization problems with exponential complexity. To illustrate, if one should find the maximum of a function of two discrete variables, one can only be sure to have the maximal value by trying all the combinations of the values possible for the two variables and pic the solution with the largest function value. The result is that the time to find a solution rapidly increases with the number of variables of the configuration and the size of the values possible for these variables.
From this discussion it is clear that there may not be sufficient time to run the optimizer until it has found the optimal value. Giving the optimizer a time budget may not work as one has no guarantee that the solution held by the optimizer at the time of termination is the best one seen until that time unless the algorithm of the optimizer is so called 'any time' algorithm that can be terminated at any time returning the best seen solution. The best known family of 'any time' algorithms is the evolutionary algorithms.

Stateful optimizers can be seen as extreme versions of 'any time' optimizers. These optimizers will run continuously taking into consideration new metric values as they arrive. However, this implies that they must deploy some kind of reinforcement learning since the application execution context changes continuously, the optimizer must be able to change opinion about the what the best configuration is for the current feasible region. Since the current search space may exclude

Funded by the European Union

the currently best seen configuration, it will be necessary to keep essentially the full history of best solutions, and back-track to return the best feasible solution when a new configuration is needed. Hence, the almost O(1) optimization time comes at the cost of increased memory consumption, and for larger problems this may be prohibitive.

Multi-objective optimization

The fundamental utility of many Cloud users is to minimise cost. However, the minimal cost is obviously resulting from not deploying the application at all. Hence, there must also be a utility related to the performance of the application and the overall utility is a trade off between cost and performance. There can of course also be other utility dimensions related to other criteria such as minimal data communication, or dimensions related to security. One may think that one can maximize individually the utility dimensions, but one will quickly come to points where one cannot increase the value of one dimension without decreasing the value in other dimensions. Returning to the example of performance and cost, it is possible to make a big application deployment with maximal performance, but then one fails to minimise the cost. The set of points where one cannot make all dimensions more optimal is called the Pareto front. It is an open challenge for the application owner to balance the different utility dimensions into a single utility value, and it can be proven that the is no way to do this ensuring to find all the points on the Pareto front if one only seeks to maximize the scalarized utility [251]. Furthermore, the balance between the utility dimensions can also be contextual. An example of this is an application with an execution deadline. The application owner may be willing to accept more cost to get better performance the closer the clock ticks to the deadline. The best would therefore be to use an optimizer algorithm that is able to find the Pareto front, or closely approximate it, and then make a decision on the scalarized overall utility when it is needed.

Multi-tenant optimization

The Cloud providers promise unlimited elasticity, and so it will always be possible to deploy multiple applications to the Cloud ensuring that all applications get the resources they need to maximise the application's utility. This is no longer the case for distributed resources from the Cloud to the edge devices. These have limited resources, and when these resources are requested by several applications, they start competing over the available resources. It is not sure that an allocation of the resources exists that maximises the utility of all deployed applications. To see this, consider two components from two different applications. Both of these components request 4 cores for optimal execution, but the edge server has only 6 cores, and only one of the applications can maximise its utility by deploying to this edge server. The question is who to decide which application's component to deploy to the server. NebulOuS will adopt the concept of "virtual cost" of the resources, and use combinatorial Walresian auctions to set a price ensuring that the resources will be allocated to the applications willing to pay this price as the utility of these applications will be least sensitive to the price of resources.

Components and integration

The optimizer is not a single component but a set of interacting components. The detailed architecture of the set of components inside the optimizer and their interaction will be detailed as part of the work in Work Package 3. However, the main external interfaces are shown in the architecture diagram above:

- When an application is deployed to NebulOuS the optimizer receives the *design time graph*, which is basically the topological application model describing the application components, their resource requirements, their variability, and possibly the data flow and the workflow of the application.
- The optimizer also receives the *user's preferences* in terms of the deployment utility and the constraints of the optimisation problem.
- Based on these inputs, the optimizer finds an *optimized service graph* specifying exactly which microservice to deploy to which available resource. These decisions are then enacted by the deployment manager.
- During application run time, the monitoring data is collected by the event monitoring system. The event monitoring system will forward the monitoring values describing the current execution context, or the predicted execution context if proactive adaptation is used. Based on the provided execution context, the optimizer will compute the optimized service graph for the current situation and forward this to the deployment manager, which will make the difference between the new optimized service graph and the currently running service graph, and create a deployment plan which in this case will act as an adaptation plan.

Funded by
the European Union

b. Requirements Mapping

Table 8: Optimizer - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud)	M	The application graph and the user preferences will be used by the Optimizer to compute an optimised service graph from which the deployment manger creates a deployment plan enacted by the Execution Adapter. The resources used will be the ones satisfying the optimization constraints and maximizing the application utility.
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	The Optimizer will take into account the mentioned factors provided that appropriate monitoring sensors are installed, as well as relevant resources declared to be available for the processing of a particular software component/container
F_07	The NebulOuS platform provides application components lifecyle management for all components of the application regardless of where component is deployed (cloud, edge, IoT device). Is able to start, stop components and react to monitoring alerts (if component requires restarting for example).	M	The Optimizer will issue appropriate directives to start and stop processing components, when this is appropriate based on the monitoring input and analysis of the situation. Workflow adaptations may also be proposed as appropriate. Moreover, in the case of fog resources, failover (e.g replacement of an underperforming or unavailable edge node) will be attempted.
F_08	The NebulOuS platform provides a deployment mechanism that makes it easy to deploy and manage data processing pipelines across various resource types (cloud/edge/fog).	M	Provided with concrete requirements and the application graphs and deployment constraints for the deployment of the processing pipeline, the NebulOuS platform will allow the easy deployment of data processing pipelines, while also providing utilities for their management.
F_11	NebulOuS should offer a mechanism to deploy and invoke serverless computation functions.	R	Serverless functions are just another type of application components in the optimized service graph produced by the optimizer component, and functions only require special handling when they are enacted on a resource, i.e., during deployment.
F_12	Monitor and predict downtime of registered resources to maximize time/cost ratio. NebulOuS should be able to accommodate applications that prioritize resources with lower down time, in comparison with the opposite.	R	Downtime will be considered if the relevant metrics are included in the optimisation problem, as it as relates to the application execution context. In this case, the application preferences and its utility functions will be taken into account during the optimization process.
F_13	Handle paused or dropped tasks	O	Paused or dropped tasks in a workflow invalidates the current workflow the deployment and the revised workflow must be completely re-deployed. Hence, if there are monitored metrics to detect this situation, the event management system and anomaly detection components will indicate to the optimizer that the current service graph is invalid, and a new optimised service graph will be produced.

F_14	The system shall be able to be started and operated during run-time with minimal prior training	M	The Optimizer component will rely on the input of the DevOps concerning the appropriate operational range of the application and will onwards be able to appropriately manage autonomously the application.
NF_02	The NebulOuS platform will be able to operate even if available bandwidth between edge infrastructure and the cloud is limited or connection is temporarily lost.	R	The Optimizer component shall not rely on a persistent network connection between processing instances at the edge and the cloud. This is a requirement for the data collection and monitoring system. The quality of any connection must be measured in real time.
NF_04	NebulOuS platform should be able to deploy applications in a Raspberry Pi 3 Model B+ or similar ARM architecture.	R	The Optimizer component does not prescribe any particular hardware models and can decide on deployment to any declared resource.
NF_06	The system provides elasticity capabilities along with the cloud continuum so HW resources are provisioned or freed depending on the workload	M	The Optimizer will decide on appropriate scaling in and scaling actions in response to the current and the predicted workload provided that the appropriate sensors are in place for the monitoring system.
NF_07	The system shall be able to cope with a sudden loss of connection to individual computational units	M	The Optimizer component is only using the metric values provided by the monitoring system, and this system must decide on how to estimate or interpolate missing values.
NF_08	The system shall be able to adapt the maximum load of the edge devices to the available power supply.	R	The Optimizer component shall take into account the available power supply of edge devices to schedule processing tasks on them
NF_09	The NebulOuS platform provides near-real-time processing of big data	M/R	The Optimizer component will adopt an appropriate workflow to ensure that no manual intervention will be required to reconfigure big-data processing
NF_10	The NebulOuS platform will allow effective scaling of the cloud resources (in the edge/cloud computing).	R	The Optimizer component decide on the optimized application configuration for the current context, and the differences between the current and optimized application deployment plans will be computed by the deployment manager to request effective scaling actions as appropriate to be done by the execution adapter.

4.3.4 Deployment Manager

The NebulOuS platform optimizes the deployment of the application based on an array of available infrastructures that ranges from bare-metal, public and private cloud virtual machines, container orchestration engines, edge and fog nodes, and distributed serverless platforms. This aggregation of different infrastructures requires NebulOuS to cope with the particular way to communicate decisions to each one of them. This design complexity can be eased by a separation of concerns where the decision-making component selects the target infrastructure and the action and then pass it to a deployment controller that translates the decision according to the targeted infrastructure.

This deployment controller is realized in NebulOuS by the Deployment Manager component, which constitutes a single point of contact for the underlying infrastructure by receiving unified instructions to execute an action. It receives an optimized service graph from the Optimizer and initiates the application deployment process by creating a deployment plan. The Execution Adapter (which is responsible for managing the resource pool) is contacted first, responding with details regarding the resources to be used (e.g. node IP addresses). Subsequently, the Deployment Manager sends the deployment plan to the Overlay Network Manager, which creates the secure overlay network between the resources. The implementation of this component is related to *Task 4.1: Deployment & Orchestration in heterogeneous environments.*

Funded by the European Union

a. Role & Functionality

The NebulOuS Deployment Manager implementation will be based on Proactive's SAL (Scheduling Abstraction Layer). SAL was created in the context of Morphemic EU project and enables a decision maker component to easily adapt with a resource pool manager (i.e., the Execution Adapter in NebulOuS).

SAL runs as service and it exposes a REST API that accepts requests from authorized decision maker components. These requests can be:

- Adding/deleting/updating an infrastructure.
- Adding/deleting/updating nodes.
- Scaling up or down the application resources.
- Reconfiguring the application using a completely new deployment strategy
- Connecting bare-metal and edge nodes using ssh connections.
- Initializing the nodes with startup scripts
- Creating a database of available node candidates that an application can use, including the candidate's hardware, image and location.

In the scope of NebulOuS, SAL will be extended to include support of 3 main features:

- Support of serverless platforms: The details are yet to be established but serverless is in nature infrastructure-less as users don't manage the infrastructure needed – the infrastructure should have been pre-provisioned for them already and utilised indirectly using the appropriate interfaces, specific for the serverless platform. Surely, new capabilities will need to be added to be able to register and utilise the serverless deployment interfaces.
- Support of overlay networks: By design, the overlay network manager depends on inputs from the deployment, this input being the IP addresses of the connected and deployed nodes (in addition to the list of exposed ports).
- Support of container orchestration engines: A new set of endpoints will be created to allow the creation and reconfiguration of container clusters (e.g. Kubernetes-based ones).

b. Requirements Mapping

Table 9: Deployment Manager - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud).	M	First, by providing a complete list of the available node candidates, and second By providing the capability of deploying heterogeneous resources and finally, by exposing the nodes in a secure way.
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	R	By providing a list of node candidates that includes information about their hardware, image (Operating system and version), and location. And by providing the ability to attach volumes to cloud VMs.

F_07	The NebulOuS platform provides application components lifecyle management for all components of the application regardles of where component is deployed (cloud, edge, IoT device). Is able to start components and stop components and react to monitoriing alerts (if component requires resterarting for example).	M	By providing the capability to provision the nodes and execute actions on top of them.
F_08	The NebulOuS platform provides a deployment mechanism that makes it easy to deploy and manage data processing pipelines across various resource types (cloud/edge/fog).	M	The Deployment Manager main objective is satisfying this this requirement by acting as a single point of contact and receiving abstract instructions that will be translated based on the target infrastructure
F_11	NebulOuS should offer a mechanism to deploy and invoke serverless computation functions.	R	By providing the necessary tools, nodes, and information for the serverless platform.
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	R	By provisioning the cloud nodes with the correct security group and by exposing new ports upon a request.
NF_06	The system provides elasticity capabilities along with the cloud continuum so HW resources are provisioned or freed depending on the workload.	M	By providing endpoints to scale or reconfigure the deployment
NF_10	The NebulOuS platform will allow effective scaling of the cloud resources (in the edge/cloud computing).	R	By providing endpoints that allows the optimizer to execute decisions on the scale of the infrastructure.

4.3.5 Execution Adapter

The Execution Adapter is the component that adapts an initial deployment plan to the available infrastructure resources. In NebulOuS, it receives the deployment plan from the Deployment Manager and, based on the available resource pool, undertakes all actions needed to execute the service on those resources. It is related to *Tasks T4.1: Deployment & Orchestration in heterogeneous environments* and *T4.2: Serverless support.*

a. Role & Functionality

The Execution Adapter effectively acts as a resource pool manager, which communicates with the underlying infrastructure to provision the available resources. It is responsible for various related actions, such as connecting to an existing cluster, creating a new one upon demand, scaling up/down the virtual machines used by a container orchestration engine on a cloud provider's infrastructure, etc.

The implementation of this component will be based on Activeon's Proactive Resource Manager (RM), which will be extended with additional functionality to support the project needs. Proactive RM is the part of ProActive Workflows & Scheduling suite[71] that is responsible for the automatic management of cloud resources, by establishing a single point of contact between the orchestration and a wide set of infrastructures. One of its features which is important for NebulOuS

[71] ProActive Workflows & Scheduling is a Java-based cross-platform workflow scheduler and resource manager that can run workflow tasks in multiple languages and multiple environments. https://www.activeeon.com/products/workflows-scheduling/

Funded by the European Union

is its ability to support edge nodes, by providing node agents compatible with different system architectures including ARM, as well as hardware-accelerated nodes that can be connected through the provided node sources.

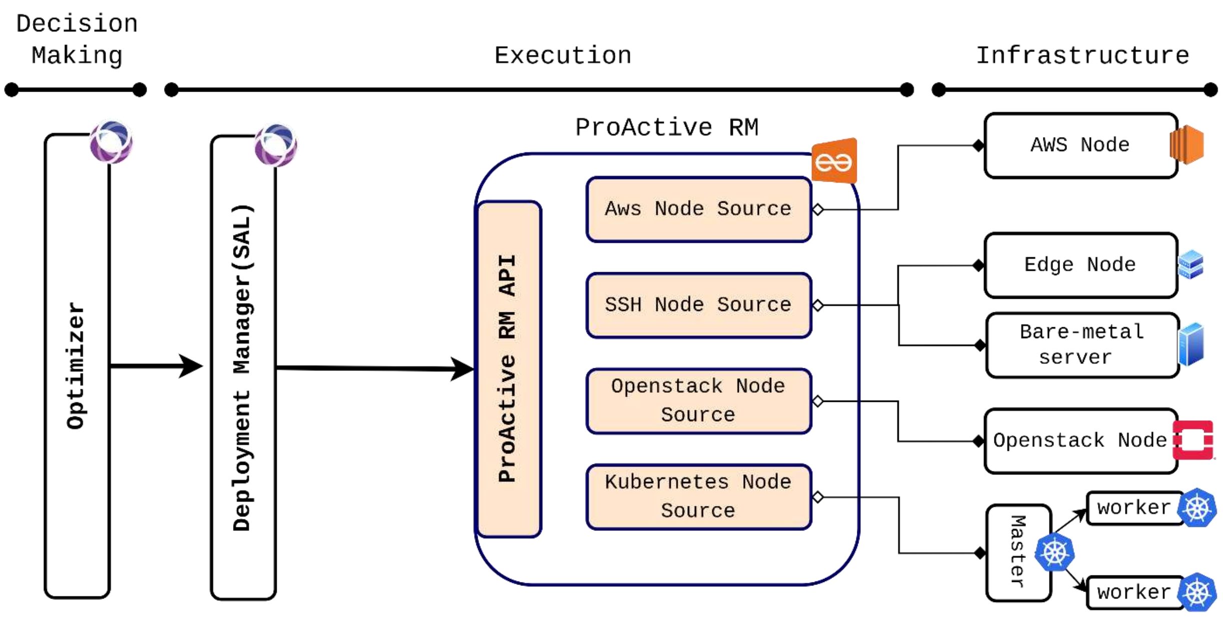

Figure 12: Resource management in NebulOuS

To manage the computational resources (compute hosts), the Execution Adapter will use the concept of node sources. Each node source controls a certain provider such that all the nodes that belongs to the same node source share the same deployment mechanism and policy. In other words, each node source is designed to handle the communication with its associated infrastructure.

The process of adding a node to the Execution Adapter involves the creation of a node agent[72]. The deployment of the node agent depends on the target machine's operating system, its architecture, and the underlying infrastructure. However, the process of communication to the connected nodes is then unified through the node agents. As shown in ***Figure 12***, using the node sources and node agents the Execution Adapter creates an abstraction layer on top of heterogeneous resources allowing the Optimizer to control the nodes using unified instructions regardless of their operating system, architecture, or provider.

b. Requirements Mapping

Table 10: Execution Adapter - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud).	M	By providing the capability of deploying heterogeneous resources on different public and pivate cloud, edge nodes, and bare metal
F_07	The NebulOuS platform provides application components lifecyle management for all components of the application regardless of where the component is deployed (cloud, edge, IoT device). It is able to start components and stop components and react to monitoring alerts (if component requires restarting for example).	M	By providing the capability to provision the nodes and execute actions on top of them.

[72] The current implementation of Proactive RM deploys its own agents. During the project, we will investigate the feasibility of supporting this functionality using a single NebulOuS agent that incorporates all required functionality.

Funded by the European Union

F_11	NebulOuS should offer a mechanism to deploy and invoke serverless computation functions.	R	By providing the capability to deploy a serverless platform and call the serverless platform's interfaces to deploy and invoke functions.
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	R	By allowing the control of the cloud security groups.
NF_04	NebulOuS platform should be able to deploy applications in a Raspberry Pi 3 Model B+ or similar ARM architecture.	R	By supporting ARMv7 and ARMV8 and by providing a very light node agent that can run on very limited resources (down to only 1 core and 1GB of RAM like an RPI 3B+)
NF_06	The system provides elasticity capabilities along with the cloud continuum so HW resources are provisioned or freed depending on the workload.	M	By providing Node sources that allows for dynamic update of the deployed resources.
NF_10	The NebulOuS platform will allow effective scaling of the cloud resources (in the edge/cloud computing).	R	By providing the capability of managing running resources.

4.3.6 Overlay Network Manager

The Overlay Network Manager component is a part of the NebulOuS control plane. It is related to Task 4.4: Automatic deployment of secure network overlay and implements the corresponding functionality.

a. Role & Functionality

The role of this component is to create a secure overlay network which interconnects compute resources with one another to support the multi-cloud and cloud-to-edge vision of NebulOuS. It receives a deployment plan from the Deployment Manager and, based on that, automatically creates the corresponding network connections between the endpoints.

The Overlay Network Manager implements three main functionalities: First, it establishes connections between the different resources (physical or virtual), enabling component-to-component and service-to-service communication. Second, it ensures that the traffic will be encrypted and the information being exchanged in the channels is secure. Third, it manages the various network connections and all related connectivity aspects.

Our solution will create the overlay network by leveraging the most recent advancements in the SDN/NFV field (Open vSwitch (OvS), VPN gateways) and the IO Visor Project (e.g., XPD/eBPF), taking into account the programming capabilities of modern switches. The relevant state-of-the-art is provided in Section 3.2.9.

b. Requirements Mapping

Table 11: Overlay Network Manager - relation to system requirements

Req. ID	Title	Req. Level	Means to address the requirement
F_07	The NebulOuS platform provides application components lifecyle management for all components of the application regardles of where component is deployed (cloud, edge, IoT device). Is able to start components and stop components and react to monitoriing alerts (if component requires restarting for example).	M	The Overlay Network Manager provides and manages connectivity to devices across the entire cloud continuum, interconnecting compute resources with one another, irrespective of their physical location. This way, it enables the efficient management of application

			components deployed in various infrastructure types.
F_09	The NebulOuS platform provides near-real-time communication between application components.	M	The Overlay Network Manager will address this requirement by deploying high-performance networks which utilize the latest advancements in network virtualization and programmability (e.g. eBPF/XDP).
F_14	The system shall be able to be started and operated during run-time with minimal prior training.	M	The secure network overlay will be deployed automatically, reducing the need for manual system configuration.
NF_02	The NebulOuS platform will be able to operate even if available bandwidth between edge infrastructure and the cloud is limited or connection is temporarily lost.	R	The deployed overlay network will support ad hoc and mesh networking techniques, which can provide resiliency and ensure operational continuity in the case of connectivity disruptions.
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	M	Traffic sent across the network will be always be encrypted end-to-end. Furthermore, we will incorporate security enforcement mechanisms and utilize access control techniques (e.g. RBAC, ABAC).
NF_05	Data transferred between edge nodes and between edge nodes and clouds should be secured.	M	The traffic sent across the network will be always encrypted end-to-end.
NF_07	The system shall be able to cope with a sudden loss of connection to individual computational units.	M	The deployed overlay network will support ad hoc and mesh networking techniques, which can provide resiliency and ensure operational continuity in the case of connectivity disruptions.

4.3.7 Data Collection & Management

The Data Collection and Management component is related to data management, both in the context of managing the monitoring data collected from the IoT/Edge/Cloud compute resources, as well as with respect to the data exchanged internally between the NebulOuS components. It relates to *Task 5.3: Interoperable IoT/Fog data collection and management* and *Task 5.5: Asynchronous Message-based API definition & implementation.*

Four distinct sub-components will comprise this module: A pub/sub mechanism, a time-series IoT data store, an IoT data flow pipelines orchestration mechanism and an inter-component communications middleware. Even though the individual sub-components are abstracted in the logical architecture, they are presented here in more detail and will form a part of the technical architecture.

Sub-component 1: Pub/sub mechanism

In the core of the Data Collection and Management module lies a pub/sub mechanism. Its goal is to allow the communication between modules of the NebulOuS platform as well as modules of the applications running on top of NebulOuS. It will also serve as an entry point for IoT data required by any of these applications.

It is related to tasks *T5.3: Interoperable IoT/Fog data collection and management* and *T5.5: Asynchronous Message-based API definition & implementation*. It will be utilized by the Event Management System.

a. Role & Functionality

The role of this component is to create a secure, efficient communications channels within NebulOuS distributed architecture that will utilize a publish-subscribe paradigm to allow:

- NebulOuS components to publish and subscribe to event streams.
- IoT devices or application components to publish event streams.
- Application components to subscribe to these information streams.

To facilitate the integration of devices/systems that can't connect directly to the publish-subscribe broker, a proxy to handle the communication with the platform will also be included.

The management of the PUB/SUB mechanism will automatically be handled by a component responsible for reading the reified ARP model offered for each brokered application/IoT device and create the necessary pub/sub topics.

Some candidate technologies to cover this necessity have already been identified during the SoTA review (EMQX, Zenoh) and others will be investigated.

b. Requirements Mapping

Table 12: Pub/sub mechanism - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud)	M	The PUB/SUB mechanism will be the communication backbone between workers dedicated to the execution of the data processing flows.
F_02	The NebulOuS platform offers a lightweight resilient event pub/sub mechanism with persistence, access control and consumer-group capacities to orchestrate communication between the different modules of the solution. The system shall provide events publish and subscribe based communication "bus" for interaction of components	M/R	The PUB/SUB mechanism will allow the communiaction of events between different components of the applications running on top of NebulOuS.
F_09	The NebulOuS platform provides near-real-time communication between application components.	O	A PUB/SUB technology that minimizes communication overhead will be selected.
F_10	NebulOuS offers a mechanism to partition streaming analysis jobs between a variable number of workers (similar to Spark partitioning)	R	The PUB/SUB mechanism will be the communication backbone between workers dedicated to the execution of the data processing flows.
NF_01	Fast decisions from the Nebulous, regarding the resources to allocate and fast execution of these decisions (fast communication, internal processing to take decisions etc).	R/M	A PUB/SUB technology that minimizes communication overhead will be selected.
NF_05	Data transfered between edge nodes and between edge nodes and clouds should be secured	M	The PUB/SUB mechanism will offer encrypted communications (mTLS).
NF_09	The NebulOuS platform provides near-real-time processing of big data	M/R	The PUB/SUB mechanism shall suport the transmission of big-data event streams.
NF_11	The NebulOuS platform ensures data privacy when transferring data from one node to another.	R	The PUB/SUB mechanism will offer encrypted communications (mTLS).

Sub-component 2: IoT time series data store

IoT devices connected to the NebulOuS platform will generate streams of events and measurements that need to be stored for later retrieval. To cover this necessity, a distributed time series database will be deployed as part of the NebulOuS platform

a. Role & Functionality

This database will be able to persist the time-series related data generated by the IoT devices. Using a well-defined query mechanism, other application components running on the NebulOuS platform will be able to retrieve this data. Databases identified on the SoTA: InfluxDB , Kdb+ and OpenTSDB, among others will be investigated to cover this necessity.

b. Requirements Mapping

Table 13: IoT time series data store - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_02	The NebulOuS platform offers a lightweight resilient event pub/sub mechanism with persistence, access control and consumer-group capacities to orchestrate communication between the different modules of the solution. The system shall provide events publish and subscribe based communication "bus" for interaction of components	M/R	The IoT time series data store will persist events writen to the pub/sub mechanism.
NF_09	The NebulOuS platform provides near-real-time processing of big data	M/R	The datastore will be able to ingest big ammounts of data on a fast pace.

Sub-component 3: IoT Data processing pipelines orchestration tool

Interoperability aspects of the data collection and management can require the execution of transformation pipelines that include payload encoding conversion, maping of data structures and conversion of values among others.

a. Role & Functionality

To ease the development, deployment and management of these transformation pipelines, the NebulOuS platform will include a framework that allows users to declaratively express these data transformation pipelines by identifying their main components (data sources, transformation operations and data consumers) and interlace them to conform data transformation flows. Once defined, the orchestration of these data transformation pipelines will be handled by NebulOuS, allocating computational resources whenever needed to divide the workload. Finally, the user will be able to monitor the execution of these pipelines and detect any error occurring on them.

Apache Nifi, Apache Airflow, FogFlow, ZenohFlow, among other frameworks for data transformation pipeline creation identified during the SoTA analysis will be evaluated for this purpose.

Funded by the European Union

b. Requirements Mapping

Table 14: IoT data processing pipelines orchestration tool - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud)	M	Users will be able to declaratively express data transformation pipelines that will be able to run on the whole NebulOuS computing infrastructure (restricted to SLAs).
F_08	The NebulOuS platform provides a deployment mechanism that makes it easy to deploy and manage data processing pipelines across various resource types (cloud/edge/fog).	M	Declarative processing pipelines will ease the deployment of data transformation flows.
NF_09	The NebulOuS platform provides near-real-time processing of big data	M/R	Once deployed, the data transformation pipelines will be able to utilize NebulOuS computing power to distribute the workload.
NF_11	The NebulOuS platform ensures data privacy when transferring data from one node to another.	R	Communication between nodes of the data transformation pipelines will be encrypted.

Sub-component 4: **Inter-component Communications Middleware**

The communication middleware is a message-oriented software that will enable the connection between the various components defined in the NebulOuS architecture. It will act as an intermediary layer between the sender and receiver of messages, allowing for decoupled and interoperable data transmission between the different actors of the system, resulting in the creation of a homogenous communication gateway.

a. Role & Functionality

In a system with multiple components, such as NebulOuS, the message-oriented middleware will provide message routing, by transferring messages from one component to another, based on message content or destinations. It provides a publish-subscribe pattern for asynchronous message delivery, allowing components to subscribe to specific topics or utilize service bus queues for receiving only the messages that are relevant to them. In case a synchronous communication approach is needed by pre-existing components, the middleware can support the use of Enterprise Integration Patterns (EIPs) to achieve this.

In general, the middleware will provide interoperability between the different actors by supporting various messaging protocols (e.g. AMQP, MQTT, etc.), depending on the capabilities of each transacted component. Provided that some components may not support a messaging protocol or are already exposing functionality in a non-messaging manner, we will also investigate the transformation between non-message-based protocols like HTTP or XML-RPC, which can allow for a more unified and seamless integration.

The communication will be robust and resilient, as the middleware will provide message delivery guarantees to the interested parties, by utilizing mechanisms like redelivery, ordering, and transaction management.
Security is also a crucial portion of the communication, as unauthorized access may lead to privacy issues or even malicious acts by disrupting the proper functioning of the system. The middleware will provide authentication/authorization capabilities, as well as message encryption where it is considered necessary.

Funded by the European Union

The inter-component middleware will also be utilized to gain insight into the internal behavior of the system and its distributed components, as most of them will communicate through it. We can achieve observability by monitoring, logging and tracing the messages as they flow through the system. Specifically, the component will provide:

- Monitoring: through exposed metrics including the number of messages in the queues, the rate of message consumption, and the message processing times. These metrics can be used to monitor middleware's own performance, as well as the performance of each transacted component.
- Logging: logged messages by the middleware can provide valuable insights into the behavior of the system and get detailed information about errors, warnings and other events that occur during the message processing.
- Tracing: enhanced tracking for messages of particular importance or groups of messages, can be used if deemed necessary, to follow critical information as it is moves through the system

The previous functionalities can be integrated with visualization tools to provide a more comprehensive view of the system's behavior and help identify performance issues or errors. Ultimately, using a message-oriented middleware will allow for the creation of a scalable, flexible, and loosely coupled system.

b. Requirements Mapping

Table 15: Intercomponent Communications Middleware - relation to system requirements

ID	Description	Req. Level	Means to address requirements
F_02	The NebulOuS platform offers a lightweight resilient event pub/sub mechanism with persistence, access control and consumer-group capacities to orchestrate communication between the different modules of the solution. The system shall provide events publish and subscribe based communication "bus" for interaction of components	M/R	A message-oriented middleware will be deployed with authentication/authorization and encrypting mechanisms to enable the communication between NebulOuS components
NF_11	The NebulOuS platform ensures data privacy when transferring data from one node to another.	R	The communication between NebulOuS components will be encrypted,

4.3.8 Cloud/Fog Service Broker

The Cloud & Fog Service Brokerage will be an important aspect of the NebulOuS meta-operating system for coping with the massive number of resources that can be considered in cloud computing continuum deployments. This research and development work is related to *Task 3.1: MCDM-based Cloud & Fog Service Brokerage.*

a. Role & Functionality

The aim of this mechanism is to reduce the variability space when considering cloud computing continuum resources for hosting application component instances. NebulOuS Cloud & Fog Service Brokerage (CFSB) will introduce brokerage capabilities, beyond the state-of-the-art analysed in Section 3.2.8, in a similar manner that OS intermediates between computing resources and software. Specifically, it will support cloud computing continuum providers and owners (organisational units, telecom providers etc.) to advertise the available hosting resources. These advertisements will combine quantitative and qualitative characteristics of their offerings (i.e., cloud, fog, edge resources that are willing to share across organisation units) that will be used for the ad-hoc formulation of local continuums according to the prerequisites of the applications. Furthermore, a graphical user interface (GUI) will be offered to application owners or DevOps to declare their preferences, in a more loose and realistic way that retains human preferences' inherent vagueness (e.g., using linguistic terms). Based on this information NebulOuS will be able to match, on demand, the preferences and resources advertisements, based on a multi-criteria decision-making (MCDM) approach that copes with precise (related to quantitative resources characteristics) and imprecise (related to qualitative aspects) criteria to enable the ad-hoc generation of micro local cloud continuums.

Therefore, NebulOus CFSB will research and develop the following functionalities:

- Announcement of infrastructural resources and services that can be made available in intra or inter-organisational collaborative infrastructure sharing scenarios;
- Declaration of application-related preferences in a way that permits the expression of quantitative and qualitative aspects
- Consideration of multiple precise and imprecise criteria for ranking cloud and fog services to recommend which subsets of them should formulate the required local computing continuums, involving cloud, fog and edge resources.

b. Requirements Mapping

Table 16: Cloud/Fog Service Broker - relation to system requirements

Req. ID	Title	Req. Level	How the requirement is addressed
F_03	The system shall provide a graphic user interface for the administration of the deployed components	M	The CFSB will use a GUI to allow end-users to declare their preferences, on the way that ad hoc cloud continuums should be considered, in a way that retains their inherent vagueness (e.g., using linguistic terms).
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	CFSB will allow organisational units or telecom providers to advertise the infrastructure and services that can be made available in intra or inter-organisational collaborative infrastructure sharing scenarios. Such factors can be considered as constraints when considering ad hoc cloud computing continuums.
NF_06	The system provides elasticity capabilities along with the cloud continuum so HW resources are provisioned or freed depending on the workload	M	CFSB will offer recommendation capabilities through a multi-criteria decision-making solution for offering fog service ranking according to the preferences and the advertised HW resources characteristics.
NF_10	The NebulOuS platform will allow effective scaling of the cloud resources (in the edge/cloud computing).	R	NebulOuS Cloud & Fog Service Brokerage (CFSB) will provide a preferences matchmaking mechanism that will allow the creation of ad hoc cloud computing continuums. This will provide the hosting variability space, comprising resources that can be considered upon scaling decisions.

4.3.9 SLA Generator

NebulOuS's brokerage capability naturally rests upon its capacity to contract SLAs between resource providers and requesters; to this end, the SLA Generator is provided. The component is related to *Task 2.6: SLAs in Cloud Computing Continuum Brokerage*

a. Role & Functionality

The SLA Generator is responsible for the automatic creation of SLA templates. It is based on the NebulOuS ontology, a meta-ontology featuring two primary facets:

- The *resources facet*, which offers adequate support for describing entities in the cloud continuum (devices, observations, actuations, people, etc.) and the properties thereof.
- The *QoS facet*, which offers adequate support for describing application provisioning requirements and preferences.

Funded by the European Union

The resources facet is based on SEMIC [252]: an ontology that extends the diffused and comprehensive SSN/SOSA model [120]. This facet is instantiated by resource providers to describe the cloud continuum resources brokered by NebulOuS. The QoS facet extends Q-SLA [125]: an ontology based on OWL-Q offering adequate support for modelling QoS-related properties, and for facilitating automatic SLA negotiation. This facet is instantiated by application developers to describe application provisioning requirements. Based on these two facets, the SLA Generator automatically determines, through ontological reasoning, the feasibility of QoS requirements.

The main functionalities of the SLA Generator are:

- Ontological modelling of cloud continuum resources and their properties of interest.
- Ontological modelling of QoS requirements and preferences regarding application provisioning.
- Semantic reasoning to determine the ability of cloud continuum resources to satisfy QoS requirements.
- Generation of SLA templates specifying QoS requirements and violation penalties.

b. Requirements Mapping

Table 17: SLA Generator - relation to system requirements

Req. ID	Title	Req. Level	How the requirement is addressed
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	These factors are extracted from the knowledge artefacts (concepts) that are included in the resources facet of the meta-ontology underpinning the SLA Generator.
F_05	NebulOuS should constantly monitor the QoS required by the different application components (as specified by their SLA) and apply necessary actions to remediate QoS violations.	M	QoS requirements are ontologically expressed in the QoS facet of the meta-ontology underpinning the SLA Generator

4.3.10 Brokerage Quality Assurance

An important characteristic of NebulOuS is its ability to ensure the quality of the brokerage services that it offers. To this end, the Brokerage Quality Assurance (BQA) mechanism is provided. The component is related to the work performed in the context of *Task 3.2: Quality Assurance in Fog Brokerage*.

a. Role & Functionality

The BQA mechanism assures brokerage quality by ensuring abidance of application provisioning requirements and preferences with **organisational policies:** higher-level requirements that reflect an organisation's business and/or security standpoint with respect to application provisioning. For example, consider an application called "application X" that samples intermediate-frequency signals emitted by a radar. An organisational policy may articulate that application X belongs to a class of applications that must always be provisioned over compute nodes that feature a clock rate of at least 100MHz (e.g., to avoid aliasing effects); the policy may also maintain that these nodes should not be under the control of certain organisations (which are considered untrusted). Such an organisational policy clearly places constraints on the manner in which application X may be provisioned. The BQA mechanism is responsible to ensure, through adequate ontological reasoning, that such higher-level constraints are indeed respected by application X's lower-level provisioning requirements expressed by instantiating the QoS facet of the NebulOuS meta-ontology (see Section 3.2.7).

The main functionalities of the BQA mechanism are:

- Specification of ontological model for describing organisation policies.

Funded by the European Union

- Specification of organisation policies; the BQA mechanism shall enable stakeholders[73] to instantiate the ontological model by articulating:
    - The knowledge artefacts that *must*, *may*, or *must not* be embodied in lower-level application provisioning requirements and preferences;
    - The allowable values, or value ranges, that these knowledge artefacts may assume.
- Semantic reasoning to automatically determine whether lower-level application provisioning requirements abide by higher-level organisational policies.
- Governance of organisational policies; automatic detection of inter-policy relations such as policy contradiction and subsumption.

The BQA mechanism shall be based upon the higher-level ontology conceptualised in the PaaSword project and reported in [253].

b. Requirements Mapping

Table 18: Brokerage Quality Assurance - relation to system requirements

Req. ID	**Title**	**Req. Level**	**How the requirement is addressed**
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	The BQA mechanism articulates constraints on how exactly these factors should be taken into account when deciding deployment location, thus ensuring quality in fog brokerage.
F_05	NebulOuS should constantly monitor the QoS required by the different application components (as specified by their SLA) and apply necessary actions to remediate QoS violations.	M	The BQA mechanism ensures that such monitoring takes place in accordance with pertinent organisational policies, hence ensuring quality in fog brokerage.

4.3.11 Smart Contract Encapsulator

The Smart Contract Encapsulator (SCE) component is a component of the NebulOuS platform which encapsulates SLAs in smart contract which features expressivity of semantic models with immutability aspects that blockchain can support. It is related to Task 4.3: From SLAs to Smart Contracts.

a. Role & Functionality

The SCE component is proposed to handle SLAs in NebulOuS in an efficient, secure and scalable way. To this end, SCE's main functionality is to automatically transform SLAs into smart contract functions. It is termed as *Blockchain-enabled SLA. SCE* provides a mechanism for infusing into smart contracts both the SLA rules and the monitoring algorithms used to determine whether these rules are satisfied by application execution. SCE is also in charge of enabling smart contracts to invoke other services and third parties for performing arbitrage when SLAs are violated. SCE also encompasses a sensing/monitoring loop that regularly feeds the information into the smart contracts. This component has a key role in interacting between smart contracts running on blockchain, sensing loop, and off-chain external services for maintaining SLAs.

We list below the main functionalities of SCE:

- enabling the automatic transformation of SLAs into smart contract functions;

[73] By "stakeholders" here we refer to people that typically occupy higher organisation positions and are responsible for determining and pursuing an organisation's business interests and policies, as well as its overall stance towards security, e.g. CTOs. security officers, Data Protection Officers (DPOs), etc.

Funded by the European Union

- infusing the SLA rules and the monitoring algorithms into smart contracts which are used to determine whether these rules are satisfied by application execution.
- designing new interfaces and mechanisms for obtaining feedback from the monitoring algorithms and interacting with off-chain services and other third-party services.

There exist a number of blockchain development technologies that can host and run smart contracts using various programming languages. We list below the prominent ones:

1. Ethereum as a decentralized BC platform runs smart contracts and provides a cryptocurrency called Ether (ETH) as a reward for executing smart contracts. Ethereum is written in Solidity which is a high-level programming language specifically designed for smart contracts.
2. Hyperledger Fabric is a permissioned blockchain platform that supports smart contract development. Hyperledger Fabric contracts are written in Chaincode which is a programming language based on Go.
3. EOS is a blockchain platform that provides a smart contract platform (written in C++) which enables developers to develop scalable, high-performance decentralized applications.
4. TRON is a BC platform that supports smart contracts. TRON contracts are written in Solidity similar to Ethereum.
5. Cardano is a BC platform that supports smart contracts and provides a cryptocurrency called ADA. Smart contracts in Cardano are written in Plutus which is a functional programming language.
6. NEO is a BC platform that offers a cryptocurrency called NEO, and supports smart contract development, where contracts on NEO can be developed in multiple programming languages, including C#, Python, and Java.

With respect to the implementation technologies and tools, we intend to extend one of the existing frameworks (Hyperledger or Ethereum) with new interfaces and mechanisms for obtaining feedback from the monitoring algorithms and interacting with off-chain services. Hyperledger is a very popular BC platform used widely both by industry and academia. Moreover, unlike public blockchains like Ethereum, Hyperledger is a permissioned blockchain, meaning that only authorized participants can access and validate transactions. On the other hand, Ethereum is also one of the most widely used blockchain platforms. Solidity as the programming language for developing smart contracts in Ethereum is easy to learn and provides a number of tools and libraries to make smart contract development more efficient.

b. Requirements Mapping

Table 19: Smart Contract Encapsulator - relation to system requirements

Req. ID	Title	Req. Level	How the requirement is addressed
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud)	M	SCE facilitates the allocation of resources across different platforms based on the pre-defined SLAs.
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	All these parameters can be included in SLAs related to the deployment platforms providers. SCE can facilitate the execution and monitoring of SLAs.
F_08	The NebulOuS platform provides a deployment mechanism that makes it easy to deploy and manage data processing pipelines across various resource types (cloud/edge/fog).	M	SCE facilitates the allocation of resources across different platforms based on the pre-defined SLAs

NF_06	The system provides elasticity capabilities along with the cloud continuum so HW resources are provisioned or freed depending on the workload	M	SLAs and the corresponding smart contracts will enable further allocation of resources or freeing the unused resources
NF_09	The NebulOuS platform provides near-real-time processing of big data	M/R	SCE allows creation and running of smart contracts which enables triggering of big data processing SLAs (violation if far-real-time processing of big data)

4.3.12 Event Management System

The Event Management System (EMS) is part of NebulOuS platform, spanning several places in the infrastructure. It is related to Task 5.1: Efficient, Secure and Fault-tolerant Monitoring of Fog-enabled applications.

a. Role & Functionality

The role of EMS is to deploy and maintain the monitoring functionality of NebulOuS on infrastructure nodes, which will observe the necessary Quality of Service (QoS) metrics both of the platform and of the deployed applications. Based on application description models, EMS will deploy and configure a number of monitoring agents that collect, filter, process and propagate the monitoring metrics values. The collected and computed monitoring metrics are then published in platform's pub/sub message broker hence any interested subscriber can use them.
Monitoring metrics can either be collected from infrastructure nodes and application components, or can be computed using certain processing rules. The monitoring metrics of interest, the processing rules that generate computed metrics, as well as information about processing locality and privacy level will be extracted/deduced from application descriptions or (global) system configuration.

The main functionalities of EMS are:

- Extract/Deduce monitoring requirements from application description models, as well as from system configuration
- Deploy monitoring agents and configure them into a network of collaborating nodes that process and propagate, in a context-aware manner, monitoring metrics coming from candidate or already hosting nodes located in the cloud computing continuum.
- Detect SLO violations (specified in Application description models)
- Persist and publish metrics and SLO violation events to platform's pub/sub message broker
- Continuously monitor the health status of monitoring agents and them to autonomously recover when failures are detected.

EMS will be based on and extend the corresponding monitoring system of Melodic and Morphemic projects [168].

Funded by the European Union

c. Requirements Mapping

Table 20: Event Management System - relation to system requirements

Req. ID	Title	Req. Level	How the requirement is addressed
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud)	M	EMS will collect and process monitoring data 'at' or 'near' their source (Edge/Fog), and fall back to Cloud, according to application requirements. It will also orchestrate the network of monitoring agents for distributing the monitoring data processing to the appropriate nodes.
F_04	In addition to the Processing Capacity the system shall take into consideration the following factors when deciding on the deployment location of software components / containers: Bandwidth, Storage Capacity, Power Availability, Physical Location of edge device	M	EMS will deploy and adapt the monitoring system operation based on the context, which can include available storage, power level, location, and device type.
F_05	NebulOuS should constantly monitor the QoS required by the different application components (as specified by their SLA) and apply necessary actions to remediate QoS violations.	M	The EMS monitoring agents will collect and compute certain QoS metrics out of the box (like health or performance indicators). The EMS will detect application- and platform-specific SLO violations and trigger reconfiguration by sending SLO violation events in the platform's pub/sub message broker.
F_06	The NebulOuS platform provides infrastructure monitoring capabilities.	M	EMS will autonomously deploy, configure, maintain and recover monitoring agents, based on the platform and applications' descriptions and requirements.
F_12	Monitor and predict downtime of registered resources to maximize time/cost ratio. NebulOuS should be able to accommodate applications that prioritize resources with lower down time, in comparison with the opposite.	R	EMS will collect and publish the related down time metrics.
NF_02	The NebulOuS platform will be able to operate even if available bandwidth between edge infrastructure and the cloud is limited or connection is temporarily lost.	R	EMS monitoring agents will use low bandwidth and feature fast transmission times, but this also depends on the applications' monitoring requirements. Moreover, monitoring agents will be able to recover from intermittent network connectivity.
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	M	EMS monitoring agents will include Id/Authentication information in monitoring data they collect and transmit.
NF_05	Data transfered between edge nodes and between edge nodes and clouds should be secured	M	EMS monitoring agents will use encrypted communications and storage for monitoring data when needed.
NF_07	The system shall be able to cope with a sudden loss of connection to individual computational units	M	EMS monitoring agents will be able to continue working and recover from (short) network disruptions.
NF_08	The system shall be able to adapt the maximum load of the edge devices to the available power supply.	R	EMS will adapt monitoring data processing based on the context, including edge device load and power supply.

NF_10	The NebulOuS platform will allow effective scaling of the cloud resources (in the edge/cloud computing).	R	EMS will be able to monitor (by deploying the required agents) even nodes that do not host application components yet but they are considered as hosting candidates, as being part of ad-hoc cloud computing continuums.
NF_11	The NebulOuS platform ensures data privacy when transferring data from one node to another.	R	EMS will transmit, in a context-aware manner, monitoring data to ensure the required data privacy required by the application, data source or the data type (e.g., propagate monitoring data only in encrypted form or completely avoid persisting them outside organisation boundaries.

4.3.13 AI-driven Anomaly Detection

The AI-driven Anomaly Detection engine will ensure Quality of Service (QoS), by detecting scenarios that perform differently from what they are expected to. It is related to Task 5.2: Flexible AI-driven anomaly detection at the edge.

a. Role & Functionality

The role of the AI-driven anomaly detection engine is to ensure QoS, being able to identify deviation from "normal" behaviour (i.e., low probability events, anomaly) of IoT devices. Such anomalies can be generated by lack of mobile network coverage, software or network misconfigurations, anomalous user behaviour, intruders, etc. It will consider the real time processed monitoring data coming from the NebulOuS event management system (EMS), so this will be accommodated as close to the edge as possible, and possibly leverage federated learning approaches to: i) improve response time, ii) reduce network traffic, and iii) increase fault-tolerance among the network thanks to better autonomous decision in collaboration with the evolution of the EMS, the application data management, and the self-adaptive reconfiguration capabilities. To that end, it will be considered different AI algorithms (from simple machine learning techniques to advanced deep learning technologies, and immunological-inspired systems), and use supervised and/or unsupervised approaches depending on the available data and ground truth information.

b. Requirements Mapping

Table 21: Anomaly Detection - relation to system requirements

Req. ID	**Title**	**Req. Level**	**How the requirement is addressed**
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud).	M	The AI-based anomaly detection engine will process surveillance data "at" or "near" the Edge, distributing the workload more efficiently and effectively deployed, with the goal of detecting deviations from normal device behaviour.
F_02	The NebulOuS platform offers a lightweight resilient event pub/sub mechanism with persistence, access control and consumer-group capacities to orchestrate communication between the different modules of the solution. The system shall provide events publish and subscribe based communication "bus" for interaction of components.	R	The AI-powered anomaly detection engine will be able to manage/monitor data/alerts through the pub/sub event mechanism, handling alerts and taking appropriate actions to maintain the resilience of the platform.
F_05	NebulOuS should constantly monitor the QoS required by the different application components (as specified by their SLA) and apply necessary actions to remediate QoS violations.	M	The AI-driven anomaly detection engine will detect different kind of anomalies and trigger the right events in the platform's pub/sub message broker to solve/advertise.

F_07	The NebulOuS platform provides application components lifecycle management for all components of the application regardless of where component is deployed (cloud, edge, IoT device). Can start components and stop components and react to monitoring alerts (if component requires restarting for example).	M	The AI-driven anomaly detection engine will be able to stop components and react to monitoring alerts to ensure/solve the security of the platform (CIA triad, confidentiality, integrity and availability).
F_09	The NebulOuS platform provides near-real-time communication between application components.	O	The AI-driven anomaly detection engine will consider the real time processed monitoring data.
F_13	Handle paused or dropped tasks.	O	The AI-driven anomaly detection engine will be able to handle activities that are or could be detected like anomalies.
NF_01	Fast decisions from the Nebulous, regarding the resources to allocate and fast execution of these decisions (fast communication, internal processing to take decisions etc).	M	The AI-based anomaly detection engine will consider the surveillance data processed in a distributed manner to achieve a real-time response.
NF_02	The NebulOuS platform will be able to operate even if available bandwidth between edge infrastructure and the cloud is limited or connection is temporarily lost.	R	The AI-based anomaly detection engine will be able to guarantee its functionality even in such circumstances, because certain actions are performed on devices deployed on devices and edge.
NF_04	NebulOuS platform should be able to deploy applications in a Raspberry Pi 3 Model B+ or similar ARM architecture.	R	The AI-driven anomaly detection engine will be lightweight to be able to deploy in IoT devices at the edge.
NF_05	Data transferred between edge nodes and between edge nodes and clouds should be secured.	M	The AI-based anomaly detection engine includes the necessary countermeasures to ensure the integrity and security of the reported and processed data.
NF_07	The system shall be able to cope with a sudden loss of connection to individual computational units.	M	The AI-based anomaly detection engine will be able to guarantee its functionality even in such circumstances, because certain actions are performed on devices deployed on devices and edge.

4.3.14 NebulOuS agent

The NebulOuS agent is a software agent that is used to realize our distributed control plane. It is deployed in the compute resources being managed by the Meta-OS and is related to several project tasks.

a. Role & Functionality

The agent is part of the NebulOuS control plane and is deployed side-by-side with the service components. It connects the NebulOuS logically centralized control plane with the distributed applications and enables information exchange between the orchestrator and the services being orchestrated, including control signals.

In particular, an agent implements three main functionalities: First, it collects monitoring data from the services and compute resources, acting as a data source for the Data Collection and Management component. In this respect, it acts as a monitoring probe and provides observability information. Second, it can act as a processing agent which supports event processing at the edge. This can alleviate network stress by reducing the need to stream all monitoring data to a centralized monitoring mechanism. Third, the agent enables policy enforcement in a distributed environment, acting as an action point and enabling fine-grained access control based on user-provided security and privacy policies.

The agent will be developed using Java; different implementations may be realized for different infrastructure types (e.g. for cloud and edge devices). In any case, all agents will implement the same component functionality. Microservice development frameworks will be considered where appropriate, e.g. Quarkus[74] and Spring[75], to realize the agents in a cloud-native way. Information on the current state-of-the-art cloud/fog monitoring approaches can be found in *Section 3.2.9 "Cloud and Fog Monitoring"*, while security aspects are covered in *Section 3.2.2*.

b. Requirements Mapping

Table 22: NebulOuS agent - relation to system requirements

Req. ID	Title	Req. Level	How the requirement is addressed
F_01	The NebulOuS platform provides orchestration mechanisms and data streaming capabilities that allow implementing flexible data processing workflows and ensures workloads are distributed on the appropriated computing continuum (just edge, just cloud, edge + cloud)	M	The agent enables the orchestration of distributed applications, acting as a monitoring data stream and enabling distributed event processing.
F_05	NebulOuS should constantly monitor the QoS required by the different application components (as specified by their SLA) and apply necessary actions to remediate QoS violations.	M	The agent will be capable of monitoring the application's QoS and streaming all relevant data to the Data Management module.
F_06	The NebulOuS platform provides infrastructure monitoring capabilities.	M	The agent will be able to provide not only application monitoring data, but also metrics tied to the underlying infrastructure (e.g. CPU usage, memory usage etc.)
F_12	Monitor and predict downtime of registered resources to maximize time/cost ratio. NebulOuS should be able to accommodate applications that prioritize resources with lower down time, in comparison with the opposite.	R	The agent will be capable of monitoring the downtime of resources and propagating the relevant information to the Data Management module.
NF_03	The platform provides secure access to the resources (clouds, edge resources etc.) (i.e provides user Identification/Authentication, User/application manager roles etc.)	M	The agent will include policy enforcement capabilities, enabling fine-grained access control.

[74] https://quarkus.io/

Funded by the European Union

5. Conclusions

In this document, we have detailed the high-level requirements for the NebulOuS platform, the scientific and technological state-of-the-art work relevant research areas, as well as the conceptual and technical architecture of our system. Its functional components have been detailed, along with an overview of the interactions between them.

The requirements definition process started with the extraction of use case requirements, focusing on the project pilots: a) Windmill and City Maintenance, b) Efficient, Connected and Sustainable management of infrastructure, freights and resources, c) Precision Agriculture, d) Crisis Management. Based on those, the overall system requirements were formulated.

Subsequently, we identified the research areas that are relevant to the project and reviewed the current state and recent advancements in each field: meta-operating systems for the cloud continuum, multi-cloud platforms, edge/IoT platforms, serverless computing, resource management, interoperable data collection and management, semantic modelling for interoperability, cloud/fog service brokerage, networking and communication, cloud/fog monitoring, anomaly detection, security and privacy.

Last, we presented the NebulOuS conceptual and architecture in the form of component and sequence diagrams. This architecture will be used as a starting point for the technical implementation and integration work that will take place in the following months. The physical architecture that will report on the details of the technical implementation will be provided in two iterations, the initial one in deliverable D6.1 (M18) and the final one in D6.2 (M36).

Funded by
the European Union

References

[1] 'Kubernetes'. [Online]. Available: https://kubernetes.io/
[2] 'Crossplane'. [Online]. Available: https://www.crossplane.io/
[3] 'KubeVela'. [Online]. Available: https://kubevela.io/
[4] 'OAM | Open Application Model Specification'. https://oam.dev/ (accessed Oct. 19, 2022).
[5] A. P. Achilleos *et al.*, 'The cloud application modelling and execution language', *Journal of Cloud computing*, vol. 8, pp. 1–25, 2019.
[6] A. Tsagkaropoulos, Y. Verginadis, M. Compastié, D. Apostolou, and G. Mentzas, 'Extending TOSCA for Edge and Fog Deployment Support', *Electronics*, vol. 10, no. 6, 2021, doi: 10.3390/electronics10060737.
[7] 'Kubesphere'. [Online]. Available: https://kubesphere.io/
[8] 'OpenPitrix'. [Online]. Available: https://github.com/openpitrix/openpitrix
[9] 'KubeEdge'. [Online]. Available: https://kubeedge.io/en/
[10] 'ONEedge - EU Next Generation Edge Cloud Platform', Mar. 21, 2022. https://oneedge.io/ (accessed Feb. 13, 2023).
[11] 'OpenNebula'. [Online]. Available: https://opennebula.io/
[12] 'The Definitive Platform for Modern Apps', *DC/OS*. https://dcos.io/ (accessed Feb. 10, 2023).
[13] 'Apache Mesos', *Apache Mesos*. https://mesos.apache.org/ (accessed Feb. 10, 2023).
[14] 'D2iQ DCOS Platform | Distributed Cloud Operating System'. https://d2iq.com/products/dcos (accessed Feb. 10, 2023).
[15] 'SIMPL: Cloud-to-edge federations and data spaces made simple'. [Online]. Available: https://digital-strategy.ec.europa.eu/en/news/simpl-cloud-edge-federations-and-data-spaces-made-simple
[16] 'SIMPLE Architecture Vision document'. European Commission, Mar. 30, 2022. [Online]. Available: https://ec.europa.eu/newsroom/dae/redirection/document/86241
[17] 'SIMPL - Clarifications'. European Commission. [Online]. Available: https://ec.europa.eu/newsroom/dae/redirection/document/91768
[18] 'SIMPL Final Business Requirements'. European Commission. [Online]. Available: https://ec.europa.eu/newsroom/dae/redirection/document/86235
[19] 'SUSE Rancher'. [Online]. Available: https://www.rancher.com/
[20] 'Harvester'. [Online]. Available: https://harvesterhci.io/
[21] 'KubeVirt'. [Online]. Available: https://kubevirt.io/
[22] Y. Verginadis, I. Alshabani, G. Mentzas, and N. Stojanovic, 'PrEstoCloud: Proactive Cloud Resources Management at the Edge for Efficient Real-Time Big Data Processing.', presented at the CLOSER, 2017, pp. 583–589.
[23] 'StarlingX Documentation'. https://docs.starlingx.io/ (accessed Feb. 08, 2023).
[24] 'K0s Documentation'. https://docs.k0sproject.io/v1.26.0+k0s.0/ (accessed Feb. 08, 2023).
[25] 'OpenYurt documentation', Feb. 01, 2023. https://openyurt.io/docs/ (accessed Feb. 08, 2023).
[26] 'Portainer Documentation'. https://docs.portainer.io/ (accessed Feb. 08, 2023).
[27] MirantisIT, 'Download Mirantis Kubernetes Engine Documentation', *Mirantis*. https://www.mirantis.com (accessed Feb. 08, 2023).
[28] 'AWS IoT Greengrass'. [Online]. Available: https://aws.amazon.com/greengrass/
[29] 'Azure IoT Edge'. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge?view=iotedge-1.4
[30] E. Jonas *et al.*, 'Cloud Programming Simplified: A Berkeley View on Serverless Computing'. arXiv, Feb. 09, 2019. doi: 10.48550/arXiv.1902.03383.
[31] 'CNCF WG-Serverless Whitepaper v1.0'. CNCF Serverless WG, Feb. 14, 2018.
[32] 'Knative Serving'. Knative, Feb. 06, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/knative/serving

[33] 'Anthos technical overview', *Google Cloud*. https://cloud.google.com/anthos/docs/concepts/overview (accessed Feb. 06, 2023).

[34] 'openfaas/faas'. OpenFaaS, Feb. 06, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/openfaas/faas

[35] 'Fission: Serverless Functions for Kubernetes'. Fission, Feb. 06, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/fission/fission

[36] '🐫 + ☁ = Apache Camel K'. The Apache Software Foundation, Feb. 03, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/apache/camel-k

[37] 'OpenWhisk'. The Apache Software Foundation, Feb. 06, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/apache/openwhisk

[38] 'kedacore/keda'. KEDA, Feb. 06, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/kedacore/keda

[39] 'dapr/dapr'. Dapr, Feb. 06, 2023. Accessed: Feb. 06, 2023. [Online]. Available: https://github.com/dapr/dapr

[40] 'OpenFunction: Build a Modern Cloud-Native Serverless Computing Platform'. https://kubesphere.io/blogs/faas-openfunction/ (accessed Oct. 11, 2022).

[41] 'How Dapr helps to build a cloud-agnostic FaaS platform', May 14, 2022. https://blog.dapr.io/posts/2022/05/14/how-dapr-helps-to-build-a-cloud-agnostic-faas-platform/ (accessed Oct. 11, 2022).

[42] A. J. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, 'Efficient resource management for Cloud computing environments', in *International Conference on Green Computing*, Aug. 2010, pp. 357–364. doi: 10.1109/GREENCOMP.2010.5598294.

[43] S. Mustafa, B. Nazir, A. Hayat, A. ur R. Khan, and S. A. Madani, 'Resource management in cloud computing: Taxonomy, prospects, and challenges', *Computers & Electrical Engineering*, vol. 47, pp. 186–203, Oct. 2015, doi: 10.1016/j.compeleceng.2015.07.021.

[44] K. Toczé and S. Nadjm-Tehrani, 'A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing', *Wireless Communications and Mobile Computing*, vol. 2018, p. e7476201, Jun. 2018, doi: 10.1155/2018/7476201.

[45] A. Yousefpour *et al.*, 'All one needs to know about fog computing and related edge computing paradigms: A complete survey', *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.

[46] M. B. Qureshi *et al.*, 'Survey on Grid Resource Allocation Mechanisms', *J Grid Computing*, vol. 12, no. 2, pp. 399–441, Jun. 2014, doi: 10.1007/s10723-014-9292-9.

[47] A. Ahmed *et al.*, 'Fog Computing Applications: Taxonomy and Requirements'. arXiv, Jul. 26, 2019. doi: 10.48550/arXiv.1907.11621.

[48] C.-C. Liu, C.-C. Huang, C.-W. Tseng, Y.-T. Yang, and L.-D. Chou, 'Service Resource Management in Edge Computing Based on Microservices', in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, Aug. 2019, pp. 388–392. doi: 10.1109/SmartIoT.2019.00068.

[49] A. J. Fahs, G. Pierre, and E. Elmroth, 'Voilà: Tail-Latency-Aware Fog Application Replicas Autoscaler', in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Nov. 2020, pp. 1–8. doi: 10.1109/MASCOTS50786.2020.9285953.

[50] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, 'Model-based Stream Processing Auto-scaling in Geo-Distributed Environments', in *2021 International Conference on Computer Communications and Networks (ICCCN)*, Jul. 2021, pp. 1–10. doi: 10.1109/ICCCN52240.2021.9522236.

[51] B. Xu, Z. Peng, F. Xiao, A. M. Gates, and J.-P. Yu, 'Dynamic deployment of virtual machines in cloud computing using multi-objective optimization', *Soft Comput*, vol. 19, no. 8, pp. 2265–2273, Aug. 2015, doi: 10.1007/s00500-014-1406-6.

Funded by
the European Union

[52] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, 'On Reducing IoT Service Delay via Fog Offloading', *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, Apr. 2018, doi: 10.1109/JIOT.2017.2788802.
[53] A. J. Fahs and G. Pierre, 'Proximity-Aware Traffic Routing in Distributed Fog Computing Platforms', in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 478–487. doi: 10.1109/CCGRID.2019.00062.
[54] P. Osypanka and P. Nawrocki, 'Resource Usage Cost Optimization in Cloud Computing Using Machine Learning', *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 2079–2089, Jul. 2022, doi: 10.1109/TCC.2020.3015769.
[55] F. Ben Jemaa, G. Pujolle, and M. Pariente, 'QoS-Aware VNF Placement Optimization in Edge-Central Carrier Cloud Architecture', in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–7. doi: 10.1109/GLOCOM.2016.7842188.
[56] D.-N. Vu *et al.*, 'Joint energy and latency optimization for upstream IoT offloading services in fog radio access networks', *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 4, p. e3497, 2019, doi: 10.1002/ett.3497.
[57] 'A survey on techniques for improving the energy efficiency of large-scale distributed systems | ACM Computing Surveys'. https://dl.acm.org/doi/abs/10.1145/2532637 (accessed Feb. 03, 2023).
[58] Y.-J. Yu, T.-C. Chiu, A.-C. Pang, M.-F. Chen, and J. Liu, 'Virtual machine placement for backhaul traffic minimization in fog radio access networks', in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7. doi: 10.1109/ICC.2017.7996500.
[59] S. K. Abd, S. A. R. Al-Haddad, F. Hashim, A. B. H. J. Abdullah, and S. Yussof, 'Energy-Aware Fault Tolerant Task offloading of Mobile Cloud Computing', in *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, Apr. 2017, pp. 161–164. doi: 10.1109/MobileCloud.2017.26.
[60] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, 'Multi-Server Multi-User Multi-Task Computation Offloading for Mobile Edge Computing Networks', *Sensors*, vol. 19, no. 6, Art. no. 6, Jan. 2019, doi: 10.3390/s19061446.
[61] M. Mukherjee *et al.*, 'Latency-Driven Parallel Task Data Offloading in Fog Computing Networks for Industrial Applications', *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6050–6058, Sep. 2020, doi: 10.1109/TII.2019.2957129.
[62] F. Y. Okay and S. Ozdemir, 'Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-Based Solution', *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4871–4889, Dec. 2018, doi: 10.1109/JIOT.2018.2882781.
[63] H. Pydi and G. N. Iyer, 'Analytical Review and Study on Load Balancing in Edge Computing Platform', in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, Mar. 2020, pp. 180–187. doi: 10.1109/ICCMC48092.2020.ICCMC-00036.
[64] A. J. Kadhim and S. A. H. Seno, 'Energy-efficient multicast routing protocol based on SDN and fog computing for vehicular networks', *Ad Hoc Networks*, vol. 84, pp. 68–81, Mar. 2019, doi: 10.1016/j.adhoc.2018.09.018.
[65] D. Puthal, M. S. Obaidat, P. Nanda, M. Prasad, S. P. Mohanty, and A. Y. Zomaya, 'Secure and Sustainable Load Balancing of Edge Data Centers in Fog Computing', *IEEE Communications Magazine*, vol. 56, no. 5, pp. 60–65, May 2018, doi: 10.1109/MCOM.2018.1700795.
[66] R. Beraldi, A. Mtibaa, and H. Alnuweiri, 'Cooperative load balancing scheme for edge computing resources', in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 94–100. doi: 10.1109/FMEC.2017.7946414.
[67] A. Kapsalis, P. Kasnesis, I. S. Venieris, D. I. Kaklamani, and C. Z. Patrikakis, 'A Cooperative Fog Approach for Effective Workload Balancing', *IEEE Cloud Computing*, vol. 4, no. 2, pp. 36–45, Mar. 2017, doi: 10.1109/MCC.2017.25.

Funded by
the European Union

[68] J. Liu, B. Bai, J. Zhang, and K. B. Letaief, 'Cache Placement in Fog-RANs: From Centralized to Distributed Algorithms', *IEEE Transactions on Wireless Communications*, vol. 16, no. 11, pp. 7039–7051, Nov. 2017, doi: 10.1109/TWC.2017.2737015.

[69] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand, 'iFogStor: An IoT Data Placement Strategy for Fog Infrastructure', in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 97–104. doi: 10.1109/ICFEC.2017.15.

[70] A. Aral and T. Ovatman, 'A Decentralized Replica Placement Algorithm for Edge Computing', *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, Jun. 2018, doi: 10.1109/TNSM.2017.2788945.

[71] Y. Shao, C. Li, and H. Tang, 'A data replica placement strategy for IoT workflows in collaborative edge and cloud environments', *Computer Networks*, vol. 148, pp. 46–59, Jan. 2019, doi: 10.1016/j.comnet.2018.10.017.

[72] K. Li and J. Nabrzyski, 'Traffic-Aware Virtual Machine Placement in Cloudlet Mesh with Adaptive Bandwidth', in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2017, pp. 49–56. doi: 10.1109/CloudCom.2017.47.

[73] L. Zhao, J. Liu, Y. Shi, W. Sun, and H. Guo, 'Optimal Placement of Virtual Machines in Mobile Edge Computing', in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec. 2017, pp. 1–6. doi: 10.1109/GLOCOM.2017.8254084.

[74] C. Li, Y. Wang, H. Tang, Y. Zhang, Y. Xin, and Y. Luo, 'Flexible replica placement for enhancing the availability in edge computing environment', *Computer Communications*, vol. 146, pp. 1–14, Oct. 2019, doi: 10.1016/j.comcom.2019.07.013.

[75] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, 'Towards QoS-Aware Fog Service Placement', in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 89–96. doi: 10.1109/ICFEC.2017.12.

[76] H. Tang, C. Li, J. Bai, J. Tang, and Y. Luo, 'Dynamic resource allocation strategy for latency-critical and computation-intensive applications in cloud–edge environment', *Computer Communications*, vol. 134, pp. 70–82, Jan. 2019, doi: 10.1016/j.comcom.2018.11.011.

[77] K. Ray, A. Banerjee, and N. C. Narendra, 'Proactive Microservice Placement and Migration for Mobile Edge Computing', in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, Nov. 2020, pp. 28–41. doi: 10.1109/SEC50012.2020.00010.

[78] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, 'A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments', *J Grid Computing*, vol. 12, no. 4, pp. 559–592, Dec. 2014, doi: 10.1007/s10723-014-9314-7.

[79] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, 'Elasticity in cloud computing: a survey', *Ann. Telecommun.*, vol. 70, no. 7, pp. 289–309, Aug. 2015, doi: 10.1007/s12243-014-0450-7.

[80] G. Rattihalli, M. Govindaraju, H. Lu, and D. Tiwari, 'Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes', in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Jul. 2019, pp. 33–40. doi: 10.1109/CLOUD.2019.00018.

[81] S. Taherizadeh and M. Grobelnik, 'Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications', *Advances in Engineering Software*, vol. 140, p. 102734, Feb. 2020, doi: 10.1016/j.advengsoft.2019.102734.

[82] J. Zhu, Z. Zheng, Y. Zhou, and M. R. Lyu, 'Scaling Service-Oriented Applications into Geo-distributed Clouds', in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, Mar. 2013, pp. 335–340. doi: 10.1109/SOSE.2013.56.

[83] C. Qu, 'Auto-scaling and deployment of web applications in distributed computing clouds', 2016. Accessed: Feb. 07, 2023. [Online]. Available: https://www.semanticscholar.org/paper/Auto-scaling-and-deployment-of-web-applications-in-Qu/d16bb76802bcccd26eb9d0b75a9b89e2fbd0c21c

[84] W.-S. Zheng and L.-H. Yen, 'Auto-scaling in Kubernetes-Based Fog Computing Platform', in *New Trends in Computer Technologies and Applications*, Singapore, 2019, pp. 338–345. doi: 10.1007/978-981-13-9190-3_35.
[85] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, 'ENORM: A Framework For Edge NOde Resource Management', *IEEE Transactions on Services Computing*, vol. 13, no. 6, pp. 1086–1099, Nov. 2020, doi: 10.1109/TSC.2017.2753775.
[86] GSM Association, 'IoT Big Data Framework Architecture v2.0', 2018. [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2018/11/CLP.25-v2.0.pdf
[87] T. Salman and R. Jain, 'A survey of protocols and standards for internet of things', *arXiv preprint arXiv:1903.11549*, 2019.
[88] 'DB-Engines Ranking', *DB-Engines*. https://db-engines.com/en/ranking/time+series+dbms (accessed Feb. 14, 2023).
[89] D. Bees, L. Frost, M. Bauer, M. Fisher, and W. Li, 'NGSI-LD API: For Context Information Management', *ETSI White Paper*, no. 31, 2019.
[90] 'JSON-LD 1.1'. https://www.w3.org/TR/json-ld11/ (accessed Feb. 14, 2023).
[91] V. Cardellini, F. Lo Presti, M. Nardelli, and G. R. Russo, 'Runtime adaptation of data stream processing systems: The state of the art', *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, 2022.
[92] C. Flores, P. Grace, and G. S. Blair, 'Sedim: A middleware framework for interoperable service discovery in heterogeneous networks', *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 6, no. 1, pp. 1–8, 2011.
[93] N. Limam *et al.*, 'OSDA: Open service discovery architecture for efficient cross-domain service provisioning', *Computer Communications*, vol. 30, no. 3, pp. 546–563, 2007.
[94] P.-G. Raverdy, V. Issarny, R. Chibout, and A. de La Chapelle, 'A multi-protocol approach to service discovery and access in pervasive environments', presented at the 2006 3rd Annual International Conference on Mobile and Ubiquitous Systems-Workshops, 2006, pp. 1–9.
[95] Y.-D. Bromberg and V. Issarny, 'INDISS: Interoperable discovery system for networked services', presented at the Middleware 2005: ACM/IFIP/USENIX 6th International Middleware Conference, Grenoble, France, November 28-December 2, 2005. Proceedings 6, 2005, pp. 164–183.
[96] J. Nakazawa, H. Tokuda, W. K. Edwards, and U. Ramachandran, 'A bridging framework for universal interoperability in pervasive systems', presented at the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), 2006, pp. 3–3.
[97] A. Bennaceur, E. Andriescu, R. S. Cardoso, and V. Issarny, 'A unifying perspective on protocol mediation: interoperability in the future internet', *Journal of Internet Services and Applications*, vol. 6, pp. 1–15, 2015.
[98] G. S. Blair, M. Paolucci, P. Grace, and N. Georgantas, 'Interoperability in complex distributed systems', *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures 11*, pp. 1–26, 2011.
[99] M. Noura, M. Atiquzzaman, and M. Gaedke, 'Interoperability in internet of things: Taxonomies and open challenges', *Mobile networks and applications*, vol. 24, pp. 796–809, 2019.
[100] 'AllJoyn reference implementation'. [Online]. Available: https://openconnectivity.org/technology/reference-implementation/alljoyn/
[101] 'IoTivity reference implementation'. [Online]. Available: https://openconnectivity.org/technology/reference-implementation/
[102] 'TS-0012-V3.7.3 (OneM2M base ontology)'.
[103] 'ETSI GS CIM 006', Jul. 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/006/01.01.01_60/gs_CIM006v010101p.pdf
[104] B. Costa, P. F. Pires, and F. C. Delicato, 'Modeling iot applications with sysml4iot', presented at the 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2016, pp. 157–164.

Funded by the European Union

[105] M. Bauer, M. Boussard, N. Bui, and F. Carrez, 'Internet of Things – Architecture IoT-A Deliverable D1.5 – Final architectural reference model for the IoT v3.0', Technical report D1.5, Jul. 2013.
[106] F. Pramudianto *et al.*, 'Iot link: An internet of things prototyping toolkit', presented at the 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops, 2014, pp. 1–9.
[107] J. Janak and H. Schulzrinne, 'Framework for rapid prototyping of distributed IoT applications powered by WebRTC', presented at the 2016 Principles, Systems and Applications of IP Telecommunications (IPTComm), 2016, pp. 1–7.
[108] T. Kalbarczyk and C. Julien, 'Omni: An application framework for seamless device-to-device interaction in the wild', presented at the Proceedings of the 19th International Middleware Conference, 2018, pp. 161–173.
[109] A. Bröring *et al.*, 'Enabling IoT ecosystems through platform interoperability', *IEEE software*, vol. 34, no. 1, pp. 54–61, 2017.
[110] G. Fierro *et al.*, 'Mortar: An open testbed for portable building analytics', *ACM Transactions on Sensor Networks (TOSN)*, vol. 16, no. 1, pp. 1–31, 2019.
[111] 'Introduction — Brick Ontology Documentation'. https://docs.brickschema.org/intro.html (accessed Feb. 14, 2023).
[112] Y. Gao, J. Zhang, G. Guan, and W. Dong, 'LinkLab: A scalable and heterogeneous testbed for remotely developing and experimenting IoT applications', presented at the 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), 2020, pp. 176–188.
[113] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, 'Smart-M3 information sharing platform', presented at the The IEEE symposium on Computers and Communications, 2010, pp. 1041–1046.
[114] P. Lade, Y. Upadhyay, K. Dantu, and S. Y. Ko, 'Developing adaptive quantified-self applications using dynasense', presented at the 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), 2016, pp. 61–71.
[115] M. Jang, H. Lee, K. Schwan, and K. Bhardwaj, 'SOUL: an edge-cloud system for mobile applications in a sensor-rich world', presented at the 2016 IEEE/ACM Symposium on Edge Computing (SEC), 2016, pp. 155–167.
[116] K. Kotis and A. Katasonov, 'Semantic interoperability on the web of things: The semantic smart gateway framework', presented at the 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, 2012, pp. 630–635.
[117] F. M. Roth, C. Becker, G. Vega, and P. Lalanda, 'XWARE—A customizable interoperability framework for pervasive computing systems', *Pervasive and Mobile Computing*, vol. 47, pp. 13–30, Jul. 2018, doi: 10.1016/j.pmcj.2018.03.005.
[118] R. Yus, G. Bouloukakis, S. Mehrotra, and N. Venkatasubramanian, 'The SEMIoTIC Ecosystem: A Semantic Bridge between IoT Devices and Smart Spaces', *ACM Trans. Internet Technol.*, vol. 22, no. 3, pp. 1–33, Aug. 2022, doi: 10.1145/3527241.
[119] M. Compton *et al.*, 'The SSN ontology of the W3C semantic sensor network incubator group', *Journal of Web Semantics*, vol. 17, pp. 25–32, Dec. 2012, doi: 10.1016/j.websem.2012.05.003.
[120] A. Haller *et al.*, 'The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation', *SW*, vol. 10, no. 1, pp. 9–32, Dec. 2018, doi: 10.3233/SW-180320.
[121] 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)*, Toulouse, Jul. 2016, pp. 90–97. doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0035.

Funded by the European Union

[122] L. Daniele, R. Garcia-Castro, M. Lefrançois, and M. Poveda-Villalon, 'ETSI TS 103 264 (SAREF spec)'. ETSI, Feb. 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/03.01.01_60/ts_103264v030101p.pdf

[123] G. Privat and A. Medvedev, 'Guidelines for Modelling with NGSI-LD', *ETSI White Paper. Available online: https://www. etsi. org/images/files/ETSIWhitePapers/etsi_wp_42_NGSI_LD. pdf (accessed on 21 June 2022)*, 2021.

[124] K. Kotis and A. Katasonov, 'An ontology for the automated deployment of applications in heterogeneous IoT environments', [Online]. Available: https://www.semantic-web-journal.net/sites/default/files/swj247_0.pdf

[125] K. Kritikos and D. Plexousakis, 'OWL-Q for Semantic QoS-based Web Service Description and Discovery', *Foundation of Research and Technology, Heraklion, Greece*, [Online]. Available: https://publications.ics.forth.gr/_publications/10.1.1.93.9067.pdf

[126] B. Rajendiran and J. Kanniappan, 'A Generic Model for Identifying QoS Parameters Interrelations in Cloud Services Selection Ontology during Runtime', *Symmetry*, vol. 13, no. 4, p. 563, Mar. 2021, doi: 10.3390/sym13040563.

[127] Q. Zhang, A. Haller, and Q. Wang, 'CoCoOn: Cloud Computing Ontology for IaaS Price and Performance Comparison', in *The Semantic Web – ISWC 2019*, vol. 11779, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Eds. Cham: Springer International Publishing, 2019, pp. 325–341. doi: 10.1007/978-3-030-30796-7_21.

[128] F. Moscato, R. Aversa, B. Di Martino, T.-F. Fortiş, and V. Munteanu, 'An analysis of mosaic ontology for cloud resources annotation', presented at the 2011 federated conference on computer science and information systems (FedCSIS), 2011, pp. 973–980.

[129] H. Taheri *et al.*, 'Cloud broker: a systematic mapping study', *arXiv preprint arXiv:2102.12717*, 2021.

[130] A. Elhabbash, F. Samreen, J. Hadley, and Y. Elkhatib, 'Cloud brokerage: A systematic survey', *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–28, 2019.

[131] X. Li, L. Pan, and S. Liu, 'A survey of resource provisioning problem in cloud brokers', *Journal of Network and Computer Applications*, p. 103384, 2022.

[132] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, 'Spotcheck: Designing a derivative iaas cloud on the spot market', presented at the Proceedings of the Tenth European Conference on Computer Systems, 2015, pp. 1–15.

[133] I. Patiniotakis, Y. Verginadis, and G. Mentzas, 'PuLSaR: preference-based cloud service selection for cloud service brokers', *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–14, 2015.

[134] M. Mahalingam *et al.*, 'Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks', 2070–1721, 2014.

[135] P. Garg and Y. Wang, 'NVGRE: Network virtualization using generic routing encapsulation', 2070–1721, 2015.

[136] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, 'A survey of software-defined networking: Past, present, and future of programmable networks', *IEEE Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[137] B. Yi, X. Wang, K. Li, and M. Huang, 'A comprehensive survey of network function virtualization', *Computer Networks*, vol. 133, pp. 212–262, 2018.

[138] M. S. Bonfim, K. L. Dias, and S. F. Fernandes, 'Integrated NFV/SDN architectures: A systematic literature review', *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–39, 2019.

[139] P. P. Ray and N. Kumar, 'SDN/NFV architectures for edge-cloud oriented IoT: A systematic review', *Computer Communications*, vol. 169, pp. 129–153, 2021.

[140] M. Pudelko, P. Emmerich, S. Gallenmüller, and G. Carle, 'Performance analysis of VPN gateways', presented at the 2020 IFIP Networking Conference (Networking), 2020, pp. 325–333.

Funded by the European Union

[141] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, 'Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications', *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.
[142] 'BPF and XDP Reference Guide — Cilium 1.12.6 documentation'. https://docs.cilium.io/en/stable/bpf/ (accessed Feb. 14, 2023).
[143] R. Kumar and M. C. Trivedi, 'Networking analysis and performance comparison of Kubernetes CNI plugins', presented at the Advances in Computer, Communication and Computational Sciences: Proceedings of IC4S 2019, 2021, pp. 99–109.
[144] Y. Rekhter, T. Li, and S. Hares, 'A border gateway protocol 4 (BGP-4)', 2070–1721, 2006.
[145] 'Calico CNI plugin'. [Online]. Available: https://www.tigera.io/project-calico/
[146] 'Cilium - Linux Native, API-Aware Networking and Security for Containers'. Accessed: Dec. 06, 2022. [Online]. Available: https://cilium.io
[147] 'Flannel'. [Online]. Available: https://github.com/flannel-io/flannel
[148] 'Kube-OVN'. [Online]. Available: https://github.com/kubeovn/kube-ovn
[149] 'Submariner'. [Online]. Available: https://submariner.io/
[150] 'SMI | A standard interface for service meshes on Kubernetes'. https://smi-spec.io/ (accessed Feb. 03, 2023).
[151] 'Linkerd'. https://linkerd.io/ (accessed Feb. 14, 2023).
[152] 'SMI | A standard interface for service meshes on Kubernetes'. https://smi-spec.io/blog/announcing-smi-gateway-api-gamma/ (accessed Feb. 03, 2023).
[153] 'Introduction - Kubernetes Gateway API'. https://gateway-api.sigs.k8s.io/ (accessed Feb. 03, 2023).
[154] 'GAMMA - Kubernetes Gateway API'. https://gateway-api.sigs.k8s.io/contributing/gamma/ (accessed Feb. 03, 2023).
[155] 'NFC overview'. https://nfc-forum.org/learn/nfc-technology/
[156] R. Want, 'An Introduction to RFID Technology', *IEEE Pervasive Comput.*, vol. 5, no. 1, pp. 25–33, Jan. 2006, doi: 10.1109/MPRV.2006.2.
[157] 'Bluetooth overview'. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
[158] 'Zigbee', *ZigBee*. https://csa-iot.org/all-solutions/zigbee/zigbee-faq/
[159] 'Z-wave overview'. https://www.z-wave.com/learn
[160] 'Thread'. https://www.threadgroup.org/
[161] N. Kushalnagar, G. Montenegro, and C. Schumacher, 'IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals', RFC Editor, RFC4919, Aug. 2007. doi: 10.17487/rfc4919.
[162] H. A. A. Al-Kashoash and A. H. Kemp, 'Comparison of 6LoWPAN and LPWAN for the Internet of Things', *Australian Journal of Electrical and Electronics Engineering*, vol. 13, no. 4, pp. 268–274, Oct. 2016, doi: 10.1080/1448837X.2017.1409920.
[163] W. Sun, Department of ECE and INMC, Seoul National University, Seoul, South Korea, M. Choi, Department of ECE and INMC, Seoul National University, Seoul, South Korea, S. Choi, and Department of ECE and INMC, Seoul National University, Seoul, South Korea, 'IEEE 802.11ah: A Long Range 802.11 WLAN at Sub 1 GHz', *JICTS*, vol. 1, no. 1, pp. 83–108, 2017, doi: 10.13052/jicts2245-800X.115.
[164] 'Wi-Fi Halow'. https://www.wi-fi.org/discover-wi-fi/wi-fi-certified-halow
[165] 'ETSI TR 101 683'. Feb. 2000. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/101600_101699/101683/01.01.01_60/tr_101683v010101p.pdf
[166] K. Kritikos, P. Skrzypek, and F. Zahid, 'Are cloud platforms ready for multi-cloud?', presented at the Service-Oriented and Cloud Computing: 8th IFIP WG 2.14 European Conference, ESOCC 2020, Heraklion, Crete, Greece, September 28–30, 2020, Proceedings 8, 2020, pp. 56–73.
[167] U. Demirbaga *et al.*, 'Autodiagn: An automated real-time diagnosis framework for big data systems', *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1035–1048, 2021.

Funded by the European Union

[168] V. Stefanidis, Y. Verginadis, I. Patiniotakis, and G. Mentzas, 'Distributed complex event processing in multiclouds', presented at the Service-Oriented and Cloud Computing: 7th IFIP WG 2.14 European Conference, ESOCC 2018, Como, Italy, September 12-14, 2018, Proceedings 7, 2018, pp. 105–119.
[169] D. Baur, F. Griesinger, Y. Verginadis, V. Stefanidis, and I. Patiniotakis, 'A model driven engineering approach for flexible and distributed monitoring of cross-cloud applications', presented at the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), 2018, pp. 31–40.
[170] B. Costa, J. Bachiega, L. R. Carvalho, M. Rosa, and A. Araujo, 'Monitoring fog computing: A review, taxonomy and open challenges', *Computer Networks*, vol. 215, p. 109189, Oct. 2022, doi: 10.1016/j.comnet.2022.109189.
[171] X. Masip, E. Marín, J. Garcia, and S. Sànchez, 'Collaborative Mechanism for Hybrid Fog-Cloud Scenarios', *Fog and Fogonomics: Challenges and Practices of Fog Computing, Communication, Networking, Strategy, and Economics*, pp. 7–60, 2020.
[172] Yiannis Verginadis, 'A Review of Monitoring Probes for Cloud Computing Continuum', presented at the 5th International Workshop on Recent Advances for Multi-Clouds and Mobile Edge Computing (M2EC), 2023.
[173] S. K. Battula, S. Garg, J. Montgomery, and B. Kang, 'An efficient resource monitoring service for fog computing environments', *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 709–722, 2019.
[174] S. Taherizadeh, V. Stankovski, and M. Grobelnik, 'A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers', *Sensors*, vol. 18, no. 9, p. 2938, 2018.
[175] A. Bali and A. Gherbi, 'Rule based lightweight approach for resources monitoring on IoT edge devices', presented at the Proceedings of the 5th International workshop on container technologies and container clouds, 2019, pp. 43–48.
[176] M. Großmann and C. Klug, 'Monitoring container services at the network edge', presented at the 2017 29th International Teletraffic Congress (ITC 29), 2017, vol. 1, pp. 130–133.
[177] R. Krahn *et al.*, 'TEEMon: A continuous performance monitoring framework for TEEs', presented at the Proceedings of the 21st International Middleware Conference, 2020, pp. 178–192.
[178] A. Morton, 'Active and passive metrics and methods (with hybrid types in-between)', 2070–1721, 2016.
[179] A. Souza, N. Cacho, A. Noor, P. P. Jayaraman, A. Romanovsky, and R. Ranjan, 'Osmotic monitoring of microservices between the edge and cloud', presented at the 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2018, pp. 758–765.
[180] J. S. Ward and A. Barker, 'Observing the clouds: a survey and taxonomy of cloud monitoring', *Journal of Cloud Computing*, vol. 3, no. 1, pp. 1–30, 2014.
[181] W. A. Higashino, 'Complex event processing as a service in multi-cloud environments', 2016.
[182] M. Hirzel, 'Partition and compose: Parallel complex event processing', presented at the Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, 2012, pp. 191–200.
[183] V. Chandola, A. Banerjee, and V. Kumar, 'Anomaly detection: A survey', *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.
[184] C. C. Aggarwal and C. C. Aggarwal, *An introduction to outlier analysis*. Springer, 2017.
[185] L. Atzori, A. Iera, and G. Morabito, 'The internet of things: A survey', *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[186] G. Fortino and P. Trunfio, *Internet of things based on smart objects: Technology, middleware and applications*. Springer, 2014.
[187] M. Markou and S. Singh, 'Novelty detection: a review—part 1: statistical approaches', *Signal processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[188] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, 'A review of novelty detection', *Signal processing*, vol. 99, pp. 215–249, 2014.
[189] N. Mohamudally and M. Peermamode-Mohaboob, 'Building an anomaly detection engine (ADE) for Iot smart applications', *Procedia computer science*, vol. 134, pp. 10–17, 2018.
[190] M. Munir, S. A. Siddiqui, M. A. Chattha, A. Dengel, and S. Ahmed, 'FuseAD: unsupervised anomaly detection in streaming sensors data by fusing statistical and deep learning models', *Sensors*, vol. 19, no. 11, p. 2451, 2019.
[191] L. Huang, X. Nguyen, M. Garofalakis, M. Jordan, A. Joseph, and N. Taft, 'In-network PCA and anomaly detection', *Advances in neural information processing systems*, vol. 19, 2006.
[192] H. E. Egilmez and A. Ortega, 'Spectral anomaly detection using graph-based filtering for wireless sensor networks', presented at the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 1085–1089.
[193] P. Domingos, 'Metacost: A general method for making classifiers cost-sensitive', presented at the Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, 1999, pp. 155–164.
[194] I. Mani and I. Zhang, 'kNN approach to unbalanced data distributions: a case study involving information extraction', presented at the Proceedings of workshop on learning from imbalanced datasets, 2003, vol. 126, pp. 1–7.
[195] C. Aggarwal and C. Zhai, 'Springer Science & Business Media; 2012', *Mining Text Data.[Google Scholar]*.
[196] K. M. Ting, 'An instance-weighting method to induce cost-sensitive trees', *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 659–665, 2002.
[197] M. V. Joshi, R. C. Agarwal, and V. Kumar, 'Mining needle in a haystack: classifying rare classes via two-phase rule induction', presented at the Proceedings of the 2001 ACM SIGMOD international conference on Management of data, 2001, pp. 91–102.
[198] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, 'A deep learning approach for network intrusion detection system', presented at the Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), 2016, pp. 21–26.
[199] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, 'Network anomaly detection with the restricted Boltzmann machine', *Neurocomputing*, vol. 122, pp. 13–23, 2013.
[200] W. Lee *et al.*, 'Real time data mining-based intrusion detection', presented at the Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01, 2001, vol. 1, pp. 89–100.
[201] S. Rajasegarar, C. Leckie, M. Palaniswami, and J. C. Bezdek, 'Distributed anomaly detection in wireless sensor networks', presented at the 2006 10th IEEE Singapore international conference on communication systems, 2006, pp. 1–5.
[202] X. R. Wang, J. T. Lizier, O. Obst, M. Prokopenko, and P. Wang, 'Spatiotemporal anomaly detection in gas monitoring sensor networks', presented at the Wireless Sensor Networks: 5th European Conference, EWSN 2008, Bologna, Italy, January 30-February 1, 2008. Proceedings, 2008, pp. 90–105.
[203] A. Servin and D. Kudenko, 'Multi-agent reinforcement learning for intrusion detection', presented at the Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning: 5th, 6th, and 7th European Symposium, ALAMAS 2005-2007 on Adaptive and Learning Agents and Multi-Agent Systems, Revised Selected Papers, 2008, pp. 211–223.
[204] M. Oh and G. Iyengar, 'Sequential anomaly detection using inverse reinforcement learning', presented at the Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & data mining, 2019, pp. 1480–1490.
[205] Y. Chen, S. Su, and H. Yang, 'Convolutional neural network analysis of recurrence plots for anomaly detection', *International Journal of Bifurcation and Chaos*, vol. 30, no. 01, p. 2050002, 2020.
[206] O. Janssens *et al.*, 'Convolutional neural network based fault detection for rotating machinery', *Journal of Sound and Vibration*, vol. 377, pp. 331–345, 2016.

[207] P. Baldi, ‘Autoencoders, unsupervised learning, and deep architectures’, presented at the Proceedings of ICML workshop on unsupervised and transfer learning, 2012, pp. 37–49.
[208] T. Luo and S. G. Nagarajan, ‘Distributed anomaly detection using autoencoder neural networks in WSN for IoT’, presented at the 2018 ieee international conference on communications (icc), 2018, pp. 1–6.
[209] N. Patel, A. N. Saridena, A. Choromanska, P. Krishnamurthy, and F. Khorrami, ‘Adversarial learning-based on-line anomaly monitoring for assured autonomy’, presented at the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 6149–6154.
[210] I. Goodfellow *et al.*, ‘Generative adversarial networks’, *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.
[211] M. Z. Alom, V. Bontupalli, and T. M. Taha, ‘Intrusion detection using deep belief networks’, presented at the 2015 National Aerospace and Electronics Conference (NAECON), 2015, pp. 339–344.
[212] A. Sari, ‘A review of anomaly detection systems in cloud networks and survey of cloud security measures in cloud storage applications’, *Journal of Information Security*, vol. 6, no. 02, p. 142, 2015.
[213] Z. C. Lipton, J. Berkowitz, and C. Elkan, ‘A critical review of recurrent neural networks for sequence learning’, *arXiv preprint arXiv:1506.00019*, 2015.
[214] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, ‘The role of edge computing in internet of things’, *IEEE communications magazine*, vol. 56, no. 11, pp. 110–115, 2018.
[215] C. Vandana and A. A. Chikkamannur, ‘IOT future in Edge Computing’, *International Journal of Advanced Engineering Research and Science*, vol. 3, no. 12, p. 236962, 2016.
[216] J. Deogirikar and A. Vidhate, ‘Security attacks in IoT: A survey’, presented at the 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2017, pp. 32–37.
[217] K. Punithasurya and S. J. Priya, ‘Analysis of different access control mechanism in cloud’, *International Journal of Applied Information Systems*, vol. 4, no. 2, pp. 34–39, 2012.
[218] E. Bertin, D. Hussein, C. Sengul, and V. Frey, ‘Access control in the Internet of Things: a survey of existing approaches and open research questions’, *Annals of telecommunications*, vol. 74, pp. 375–388, 2019.
[219] V. C. Hu *et al.*, ‘Guide to attribute based access control (abac) definition and considerations (draft)’, *NIST special publication*, vol. 800, no. 162, pp. 1–54, 2013.
[220] ANSI, ‘Next Generation Access Control - Functional Architecture (NGAC-FA)’. InterNational Committee for Information Technology Standards [incits]ANSI, 2018. [Online]. Available: https://webstore.ansi.org/standards/incits/incits4992018
[221] OASIS, ‘The Abbreviated Language for Authorization Version 1.0 Working Draft’. [Online]. Available: https://www.oasis-open.org/committees/download.php/55228/alfa-for-xacml-v1.0-wd01.doc
[222] ‘OASIS Standards’. [Online]. Available: https://www.oasis-open.org/standards/
[223] ‘W3C XML Schema Definition Language (XSD)’. [Online]. Available: https://www.w3.org/TR/xmlschema11-1/
[224] H. Firpi, ‘Handbook of Bioinspired Algorithms and Applications Edited by Stephan Olariu and Albert Y.’, 2007.
[225] A. Procopiou and N. Komninos, ‘Bio/Nature-inspired algorithms in AI for malicious activity detection’, 2019.
[226] A. K. Kar, ‘Bio inspired computing–a review of algorithms and scope of applications’, *Expert Systems with Applications*, vol. 59, pp. 20–32, 2016.
[227] N. Dash, R. Priyadarshini, B. K. Mishra, and R. Misra, ‘Bio-inspired computing through artificial neural network’, in *Fuzzy Systems: Concepts, Methodologies, Tools, and Applications*, IGI Global, 2017, pp. 1285–1313.
[228] V. Jain and M. Agrawal, ‘Applying genetic algorithm in intrusion detection system of iot applications’, presented at the 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), 2020, pp. 284–287.
[229] A. Thakkar and R. Lohiya, ‘Role of swarm and evolutionary algorithms for intrusion detection system: A survey’, *Swarm and evolutionary computation*, vol. 53, p. 100631, 2020.

[230] P. Chaudhary and K. Kumar, *ARTIFICIAL IMMUNE SYSTEM: ALGORITHMS AND APPLICATIONS REVIEW*. 2018. [Online]. Available: https://www.ssarsc.org/documents/ais.pdf

[231] R. Roman, R. Rios, J. A. Onieva, and J. Lopez, 'Immune system for the internet of things using edge technologies', *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4774–4781, 2018.

[232] J. R. Al-Enezi, M. F. Abbod, and S. Alsharhan, 'Artificial immune systems-models, algorithms and applications', 2010.

[233] L. N. De Castro and F. J. Von Zuben, 'Learning and optimization using the clonal selection principle', *IEEE transactions on evolutionary computation*, vol. 6, no. 3, pp. 239–251, 2002, doi: 10.1109/TEVC.2002.1011539.

[234] S.-I. T. Tosin and J. R. Gbenga, 'Negative selection algorithm based intrusion detection model', presented at the 2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON), 2020, pp. 202–206.

[235] X. Shen, X. Z. Gao, and R. Bie, 'Artificial immune networks: Models and applications', *International Journal of Computational Intelligence Systems*, vol. 1, no. 2, pp. 168–176, 2008, doi: 10.1109/ICCIAS.2006.294161.

[236] F. Gu, J. Greensmith, and U. Aickelin, 'The dendritic cell algorithm for intrusion detection', in *Biologically Inspired Networking and Sensing: Algorithms and Architectures*, IGI Global, 2012, pp. 84–102. [Online]. Available: https://doi.org/10.4018/978-1-61350-092-7.ch005

[237] J. Timmis, T. Knight, L. N. de Castro, and E. Hart, 'An overview of artificial immune systems', *Computation in cells and tissues: Perspectives and tools of thought*, pp. 51–91, 2004, doi: 10.1007/978-3-662-06369-9_4.

[238] L. N. De Castro and F. J. Von Zuben, 'Artificial immune systems: Part I–basic theory and applications', *Universidade Estadual de Campinas, Dezembro de, Tech. Rep*, vol. 210, no. 1, 1999, [Online]. Available: https://www.researchgate.net/profile/Leandro-De-Castro/publication/228712042_Artificial_Immune_Systems_Part_I-Basic_Theory_and_Applications/links/0fcfd5075b7c95b9cd000000/Artificial-Immune-Systems-Part-I-Basic-Theory-and-Applications.pdf

[239] C.-H. Hong and B. Varghese, 'Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms', *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[240] Y. Verginadis *et al.*, 'Context-aware policy enforcement for PaaS-enabled access control', *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 276–291, 2019.

[241] 'H2020 PUZZLE Project | Home l Towards a Sophisticated SIEM', Apr. 12, 2020. https://puzzle-h2020.com/ (accessed Feb. 23, 2023).

[242] 'WSO2 Balana Implementation'. WSO2, Feb. 22, 2023. Accessed: Feb. 22, 2023. [Online]. Available: https://github.com/wso2/balana

[243] 'Keycloak'. Keycloak, Feb. 23, 2023. Accessed: Feb. 23, 2023. [Online]. Available: https://github.com/keycloak/keycloak

[244] 'Drools - Business Rules Management System (Java^{TM}, Open Source)'. Accessed: Feb. 22, 2023. [Online]. Available: https://drools.org/

[245] 'Open Policy Agent'. Accessed: Feb. 22, 2023. [Online]. Available: https://openpolicyagent.org/

[246] 'Overview of Network Policy — Cilium 1.13.90 documentation'. https://docs.cilium.io/en/latest/security/policy/ (accessed Feb. 22, 2023).

[247] Itzhak Gilboa, *Rational choice*. Cambridge, MA, USA: MIT Press, 2010.

[248] Jeffrey O. Kephart and Rajarshi Das, 'Achieving Self-Management via Utility Functions', *IEEE Internet Computing*, vol. 11, no. 1, pp. 40–48, Jan. 2007, doi: 10.1109/MIC.2007.2.

[249] Yu-Kwong Kwok and Ishfaq Ahmad, 'Static scheduling algorithms for allocating directed task graphs to multiprocessors', *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, Dec. 1999, doi: 10.1145/344588.344618.

Funded by the European Union

[250] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J. Epema, 'Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths', *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, Aug. 2012, doi: 10.1109/TPDS.2011.303.

[251] Matthias Ehrgott, *Multicriteria Optimization*, Second. Berlin Heidelberg: Springer-Verlag, 2005. Accessed: Dec. 09, 2018. [Online]. Available: //www.springer.com/gp/book/9783540213987

[252] R. Yus, G. Bouloukakis, S. Mehrotra, and N. Venkatasubramanian, 'The semiotic ecosystem: A semantic bridge between IoT devices and smart spaces', *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 3, pp. 1–33, 2022.

[253] S. Veloudis *et al.*, 'Achieving security-by-design through ontology-driven attribute-based access control in cloud environments', *Future Generation Computer Systems*, vol. 93, pp. 373–391, 2019.