ZEROSWARM

D4.2 - Distributed stream computing within the Edge-Cloud

ZERO-enabling Smart networked control framework for Agile cyber physical production systems of systems

Project funded by Horizon Europe, Grant Agreement #101057083

Topic	HORIZON-CL4-2021-TWIN-TRANSITION-01-08
Project Title	ZERO-enabling Smart networked control framework for Agile cyber physical production systems of systems
Project Number	101057083
Project Acronym	Zero-SWARM
Contractual Delivery Date	M15
Actual Delivery Date	M16
Contributing WP	WP4
Project Start Date	01/06/2022
Project Duration	30 Months
Dissemination Level	Public
Editor	RWG
Contributors	NE-SE - AIM

Author List

Leading Author (Editor)			
Surname	Initials	Beneficiary Name	Contact email
Maccioni	RM	RWG	raffaele.maccioni@rwings.tech
Co-authors (in alphabetic order)			
Surname	Initials	Beneficiary Name	Contact email
Fritz	AF	NX-SE	artur.fritz@se.com
Lepore	LM	RWG	mario.lepore@mathbiology.tech
Serra	DS	RWG	domenico.serra@modelite.tech
Contributors (in alphabetic order)			
Surname	Initials	Beneficiary Name	Contact email
Abadía	AA	AIM	antonio.abadia@aimen.es
Deshmukh	SD	NX-SE	shreya.deshmukh@se.com
Méndez	RM	AIM	roi.mendez@aimen.es

Reviewers List

List of reviewers (in alphabetic order)			
Surname	Initials	Beneficiary Name	Contact email
Andolfi	MA	NXW	m.andolfi@nextworks.it
Khodashenas	PK	HWE	pouria.khodashenas@huawei.com
Lazaridis	GL	CERTH	glazaridis@iti.gr
Udayanto Atmojo	UA	FhG	udayanto.atmojo@aalto.fi

Document History

Document History			
Version	Date	Author	Remarks
00.00	15/02/2023	R Maccioni	Initialization
00.01	15/03/2023	R Maccioni	Chapters Definition
00.02 /00.11	14/08/2023	all involved partners	Intermediates versions with incremental contributions
00.12	25/08/2023	R Maccioni	Version ready for review
00.12A	05/09/2023	P Khodashenas	Comments from Pouria Khodashenas
00.12B	05/09/2023	M Andolfi	Comments from Matteo Andolfi
00.13	06/09/2023	R Maccioni	Integration of comments and review
1.0	22/09/2023	A Drosou	Final submission

DISCLAIMER OF WARRANTIES

This document has been prepared by Zero-SWARM project partners as an account of work carried out within the framework of the contract no 101057083.

Neither Project Coordinator, nor any signatory party of Zero-SWARM Project Consortium Agreement, nor any person acting on behalf of any of them:

- makes any warranty or representation whatsoever, express, or implied,
    - with respect to the use of any information, apparatus, method, process, or similar item disclosed in this document, including merchantability and fitness for a particular purpose, or
    - that such use does not infringe on or interfere with privately owned rights, including any party's intellectual property, or
- that this document is suitable to any particular user's circumstance; or
- assumes responsibility for any damages or other liability whatsoever (including any consequential damages, even if Project Coordinator or any representative of a signatory party of the Zero-SWARM Project Consortium Agreement, has been advised of the possibility of such damages) resulting from your selection or use of this document or any information, apparatus, method, process, or similar item disclosed in this document.

Zero-SWARM has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101057083. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

Executive Summary

The manufacturing industry is witnessing a transformative shift towards automation, where seamless data gathering, processing, and learning are pivotal in driving operational efficiency.

We aim to achieve a generic approach for distributed intelligent control applications in the cloud-to-edge continuum, enabling:

- the CPSoS paradigm
- standardization of engineering and development processes,
- run time intelligence, by real-time and near-real-time decision-making functionalities,
- distributed intelligent agents based on AI algorithms and related data.

The central focus is on defining an approach and an engineering platform that leverages a synergy of technology layers to facilitate and standardize the delivery of cloud-to-edge applications, embedding intelligent agents and their distributions, such for example agents based on machine learning algorithms for real-time decision making.

Considering the opportunities and the challenges, in this analysis, we explore the integration of key technologies such as to develop a comprehensive engineering framework that bridges the gap between cloud-based analytics and real-time edge data, keeping into consideration the backbone provided by the IEC 61499.

The convergence of these technologies enables a robust and adaptable ecosystem, enhancing data continuity, high performance and facilitating agile real or near-real-time decision-making and control processes by operators, robots, and computing resources.

The Zero-Swarm platform encapsulates an integrated, future-proof solution for data streaming eligible for near and real-time decision-making tailored for modern manufacturing. Utilizing technologies such as Kafka, MQTT, and OPC-UA, it offers a unified architecture for data control, streaming, and interoperability. Key innovation elements, like Multi-Layered Adaptivity and Metamodeling and Federated Learning, position the platform as competitive and scalable, permitting it to respond to the specific solution' requirement. End-users' benefits include enhanced operational efficiency, cost reduction, near-real-time quality control, and risk mitigation facilitated by the distribution of intelligent agents. Aims beyond the state-of-the-art:

- Seamless exchange and exposure of information across the cloud-edge continuum for AI-based applications enabling real time decision making
- On-the-fly data to knowledge conversion with automatic mapping to information models, e.g., based on OPC UA
- Harmonized development platform enabling standardization and high development efficiency.

The zero-warm Team is pioneering exploring the combined use of technological layers, such as (Kafka and MQTT) and protocols, such as OPC UA, and standards, such as IEC 61499, on a cloud-to-edge architecture for a tangible deployment of distributed intelligent agents, enabling real-time and near-real-time decision making.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Description
5G	Fifth-Generation Wireless Communications
AE	Alarm And Events
AFoF	AALTO Factory of the Future
AMR	Autonomous Mobile Robots
AI	Artificial Intelligence
AIC	Automotive Intelligence Centre
AIIC	AALTO Industrial Internet Campus
AR	Augmented Reality
CAT	Composite Automation Type
CPSoS	Cyber-Physical System of Systems
CUC	Centralized User Configuration
DA	Data Access
DFA	Demonstration Factory Aachen (DFA)
DLFi	Distributed Learning Framework
DSS	Dss Dynamic Spectrum Allocation
E2E	End-To-End
eMBB	Enhanced Mobile Broadband
gPTP	Generalized Precision Time Protocol
HAD	Historical Data Access
ICT	Information Communication Technologies
IDS	International Data Spaces
IICF	Industrial Internet Connectivity Framework
IIoT	Industrial Internet Of Things
IIRA	Industrial Internet Reference Architecture
IMC	Intelligent Mechatronic Components
ITU	International Telecommunication Union
LBO	And Local Breakout
MAS	Multi-Agent Systems
MES	Manufacturing Execution Systems
mIoT	Massive Internet of Things
mMTC	Massive Machine Type Communication
MR	Mixed Reality
NASA	National Aeronautics and Space Administration NASA
NASA	Telematic Data Collector
NPN	Non-Public Networks
OLE	Object Linking and Embedding
OPC	Open Platform Communication

OPC-UA	Opc Unified Architecture
PLC	Programmable Logic Controllers
PTZ	Production Technology Center (PTZ)
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SNPNs	Stand-Alone Npns
SOAP	Simple Object Access Protocol
SoS	Systems Of Systems
TSN	Time Sensitive Networking
UA	Unified Architecture
UACP	Unified Architecture Connection Protocol
UES	User Equipment
UPF	User Plane Function
uRLLC	Ultra-Reliable Low Latency Communication
VR	Virtual Reality

1 Introduction

1.1 Purpose of the document

Intelligent agents deployed within the Edge-Cloud continuum offer a strategic advantage by capitalizing on the strengths of both localized Edge computing and robust Cloud resources.

The primary benefits can be summarized in:

1. **Latency Reduction and Real-Time Insights:** Placing intelligent agents at the Edge enables rapid decision-making by reducing data transfer latency. Real-time insights derived from local processing enhance manufacturing processes, especially in time-sensitive scenarios like quality control and preventive maintenance.
2. **Bandwidth Optimization:** Distributing intelligence across the Edge-Cloud continuum optimizes network bandwidth usage. Data preprocessing and preliminary analysis can be performed at the Edge, significantly reducing the amount of data transmitted to the Cloud, which is particularly relevant in bandwidth-constrained environments.
3. **Privacy and Security Enhancement:** By processing data locally at the Edge, manufacturing facilities can enhance privacy and security. Critical information stays within the premises, minimizing exposure to external threats and compliance risks.
4. **Scalability and Resource Management:** Effective distribution of intelligent agents requires careful consideration of resource constraints. Edge devices with limited computational capacity can offload resource-intensive tasks to the Cloud, ensuring efficient utilization of available resources.

However, this approach also presents key challenges such as:

1. **Algorithm Selection and Deployment:** Choosing the right algorithms to deploy at the Edge versus the Cloud requires a comprehensive understanding of processing needs, latency sensitivity, and resource availability. Optimizing algorithm distribution ensures optimal performance.
2. **Asynchronous Learning and Adaptation:** Enabling asynchronous learning of predictive models is crucial for manufacturing applications. This involves continuous model updates in response to changing data streams, operational conditions, and evolving patterns. Adaptation mechanisms must be finely tuned to ensure accuracy and relevance.
3. **Data Synchronization and Consistency:** Coordinating data synchronization and maintaining consistency across distributed intelligent agents can be complex. Effective management of data updates, versioning, and synchronization mechanisms is essential to prevent discrepancies and errors.
4. **Network Reliability and Robustness:** The success of this approach hinges on reliable network connectivity. Seamless communication between Edge devices and the Cloud is vital for timely data exchange and decision-making. Redundancy and failover mechanisms mitigate disruptions caused by connectivity issues.

The technology and the methodological approach must consider such challenges in addition to standardizing the development phases and reducing unnecessary complexity and related costs.

The Cloud-to-Edge data streams and intelligent components distribution requires combining technologies and protocols, for example:

- OPC-UA is the machine-to-machine communication protocol used to exchange data between industrial automation systems. OPC-UA is used to collect data from sensors and machines on the edge.
- A distributed streaming platform (e.g. Kafka) that is used to process and analyze data in real time. The streaming platform acts as a message broker between the edge devices and the cloud platform.
- The Edge Platform is responsible for processing and analyzing the data collected from edge devices using OPC-UA. This platform can be deployed on the edge, allowing for real-time processing and analysis of data. The edge platform can also act as a gateway between the edge devices and the cloud platform.
- The Cloud Platform is responsible for storing and analyzing data received from edge devices (for example, through Kafka). This platform can be used to provide insights into the data collected from edge devices and can also be used to trigger actions based on the data received.

This document provides an incisive overview of state-of-the-art, related to the distributed stream computing for continuous gathering and learning of data in Cloud-to-Edge architecture. From the such overview, which includes the available technologies and the current limitations, the document provides a deeper understanding about how to use IEC 61499 over OPC-UA, Kafka and MTQQ.

The concepts reported in the document are needed to define the most suitable technology layers constituting the zero-Swarm Platform and zero-Swarm Applications.

1.2 Structure of the document

The document is structured as follows:

- **Chapter 1** introduction to the document and the context.
- **Chapter 2** analysis of the state of the art.
- **Chapter 3** the zero-Swarm Innovation and cloud-to-edge conceptual architecture.
- **Chapter 4** beyond the state of the Art. Insight about IEC 61499 using OPC-UA over Kafka & MQTT.
- **Chapter 5** Conclusion.
- **Chapter 6** References.

2 Distributed Streaming Computing in the Cloud-to-Edge continuum - State of the Art & Current limits

2.1 Edge-nodes and Cyber Physical Systems of Systems: a brief background knowledge for the scope of zero-Swarm

In cloud edge architecture [1], the edge encompasses the network infrastructure and computing resources situated closer to the end-user or data source, as opposed to centralized resources in a remote data center or the cloud. The purpose of the edge is to bring computing capabilities and data

storage closer to the network's edge, resulting in reduced latency, enhanced response times, and real-time data processing capabilities [2][3].

Traditional cloud computing [4] involves transmitting data from edge devices, such as sensors, IoT devices, routers, gateways, edge servers, and smartphones, to remote data centers or the cloud for processing and analysis. However, edge computing [5] allows for some processing, storage, and analysis tasks to be conducted at or near the edge devices themselves. Within the scope of the zero-Swarm, an edge node can be considered any physical device with on-board capabilities for data processing and transmission, such as a robot with on-board capabilities. So, for example, any robot with on-board capabilities is an edge node.

Figure 1 illustrating the components of an edge node might include the following elements [6]:

- Application Processor, which executes application logics, is responsible for managing and coordinating the various components of the edge node.
- Sensors collect data from the surrounding environment or from end-user devices.
- AI Application incorporates machine learning models, artificial intelligence algorithms, or advanced analytics capabilities to make intelligent decisions locally.
- Network Hardware facilitates communication with other edge nodes, cloud resources, or end-user devices, supporting wired or wireless connectivity options such as Ethernet, Wi-Fi, or cellular networks.
- Edge-DB provides local storage capacity for storing and caching relevant data, allowing for faster access and reducing the need for constant communication with the cloud.

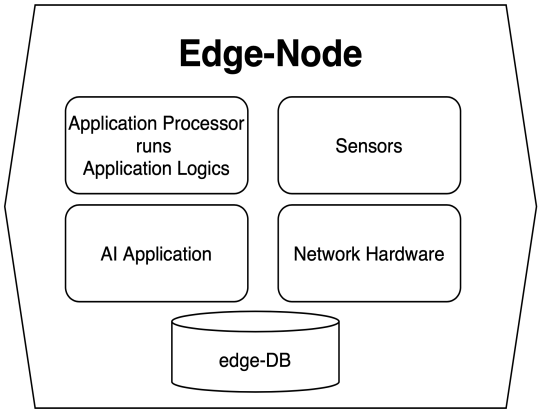

Figure 1: Edge-Node components view

Edge nodes are integral to the cloud-to-edge paradigm [7], as they facilitate distributed computing and bring computational capabilities closer to data sources or end-user devices. Key characteristics of an edge node include proximity to data sources, bandwidth optimization, computing power, improved reliability, storage capacity, connectivity, real-time processing, low latency, intelligent decision-making, and security and privacy. Specifically, [8], edge nodes are an essential component of edge computing as they are positioned in close proximity to the data source or end-user devices. They collect, process, and analyze data in real-time, reducing latency and network congestion. Edge computing optimizes bandwidth usage by performing initial data processing locally and transmitting only relevant or summarized data to the central location. Edge nodes have sufficient computing power to perform local data processing and analysis tasks and often have specialized accelerators to handle

computational workloads efficiently. They enhance reliability by minimizing reliance on a single remote data centre, and if the connection to the cloud is lost, edge devices can still function and process data locally. Edge nodes typically have local storage capacity to store and cache relevant data, allowing for faster access and reducing the need for constant communication with the cloud. Edge nodes support wired or wireless connectivity options, such as Ethernet, Wi-Fi, or cellular networks, depending on the deployment scenario. They are capable of performing real-time data processing and analysis, making them suitable for time-sensitive applications, and can quickly respond to local events, trigger actions, or provide immediate feedback to end-users. Edge nodes may incorporate machine learning models [9], artificial intelligence algorithms, or advanced analytics capabilities to make intelligent decisions locally, allowing for efficient use of cloud resources and enabling edge computing applications like predictive maintenance, anomaly detection, or real-time optimization. They enforce robust security measures to protect sensitive data and ensure data privacy, including encryption, access control, secure bootstrapping, and regular security updates to mitigate vulnerabilities.

Furthermore, they should be designed to scale horizontally or vertically based on the requirements of the edge computing environment [10]. They should support flexible configurations to accommodate varying workloads and changing demands. In other words, edge computing infrastructure should be able to add or remove edge nodes as needed to handle increased or decreased workloads. In the end, edge nodes should have management and orchestration capabilities to facilitate efficient deployment, configuration, monitoring, and software updates. Centralized management systems can help in administering a large number of distributed edge nodes effectively.

Edge nodes can be organized in a hierarchical structure, with multiple layers serving specific purposes or processing levels. This allows for the distribution of computing resources and responsibilities across different levels of the edge infrastructure. Programmable Logic Controllers (PLCs) are typically not considered edge nodes [11], as they primarily focus on real-time control and executing pre-programmed logic to manage industrial processes. However, advanced PLCs or PLC-based systems with additional computational capabilities and connectivity options, integrated with edge computing principles, may be considered edge nodes in certain contexts.

A local computing server responsible for processing and transferring data close to the field can also be considered an edge node within the context of the generalization concept useful for the zero-Swarm.

Whereas, a cyber physical system of systems [12] refers to a collection of individual systems or components working together to achieve a common goal or provide unified functionality. Each system within the cyber physical system of systems contributes to the overall system's capabilities and can be organized hierarchically. In this context, any edge node can be considered a system-node within the entire system. In a cyber system of systems [13], edge nodes can act as system-nodes and contribute to the overall system's capabilities. They can be organized hierarchically, with each edge node responsible for a specific subset of the system's functionality. For example, a network of edge nodes could be responsible for monitoring and controlling a manufacturing plant's various processes, with each edge node responsible for a specific subset of machines or equipment. edge nodes can enhance the reliability and security of a cyber system of systems. By minimizing reliance on a single remote data centre, edge nodes can provide redundancy and fault tolerance, ensuring that the system operates continuously even if one or more edge nodes fail. Edge nodes can also enforce robust security measures to protect sensitive data and ensure data privacy, such as encryption, access control, secure bootstrapping, and regular security updates to mitigate vulnerabilities.

2.2 Cloud-to-Edge technologies: a review of the main platforms

Cloud to Edge development technologies [14][15] refer to a set of tools, platforms, and frameworks that enable developers to create applications and services that leverage the strengths of cloud computing and edge computing environments. Cloud computing [16] centralizes computational resources and data in remote data centers, while edge computing processes data closer to the source, typically at the network's edge or near end devices as shown in Figure 2. These technologies allow for seamless integration and interaction between the two environments, enabling the creation of distributed and intelligent applications that capitalize on the strengths of both cloud and edge computing. These technologies enable seamless integration and interaction between the two environments, allowing developers to create distributed and intelligent applications that capitalize on the strengths of both cloud and edge computing. When combined with 5G technology [17], this concept opens up opportunities for manufacturing applications and systems involving intensive automation, robotics, and drones.

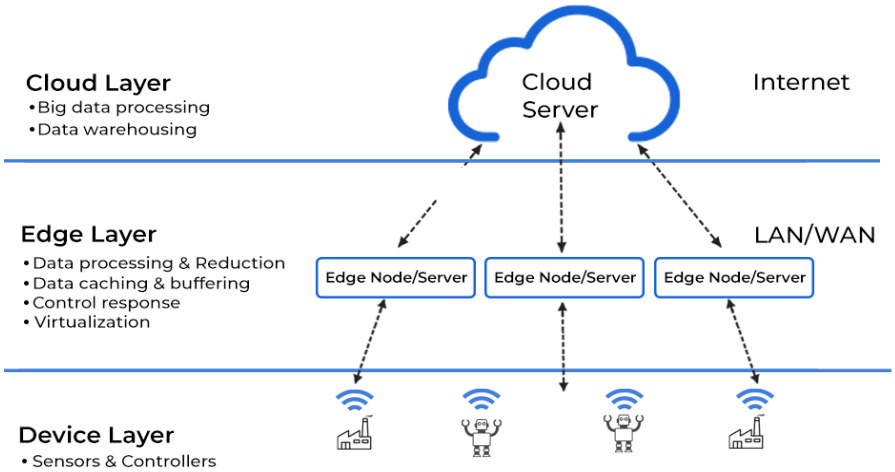

Figure 2: Cloud to edge conceptual architecture

As reported in other deliverables of the Zero Swarm project, numerous technologies contribute to a Cloud to Edge architecture. Cloud platforms such as AWS, Azure, and GCP offer a broad array of tools for developing and deploying applications in the cloud. These platforms provide capabilities like computing, storage, databases, analytics, and machine learning, which can be used to construct cloud-based components of Cloud to Edge applications. Edge platforms such as AWS Greengrass, Azure IoT Edge, and Google Edge TPU offer software frameworks and tools for deploying and managing applications at the network's edge. These platforms enable developers to execute code and process data on edge devices, such as edge servers, gateways, and IoT devices, supporting edge computing capabilities.

Containerization technologies like Docker and orchestration platforms like Kubernetes are widely employed in Cloud to Edge development. Containers allow for packaging applications and their dependencies into portable units, enabling consistent deployment across cloud and edge environments. Kubernetes offers a robust and scalable framework for managing containerized applications across cloud and edge clusters, simplifying application deployment and management in a distributed setting.

Edge computing frameworks such as Apache Edgent, Eclipse ioFog, and TensorFlow Lite for Edge provide programming frameworks and libraries specifically tailored for edge computing. These frameworks provide tools and APIs for developing edge applications capable of processing data locally on edge devices, facilitating real-time and low-latency processing capabilities. Machine learning models can be implemented at the edge to enable intelligent data processing closer to the source. Technologies like TensorFlow Lite, ONNX Runtime, and NVIDIA TensorRT offer libraries and tools for deploying machine learning models on edge devices, allowing for real-time inference and decision-making capabilities.

Connectivity plays a vital role in Cloud to Edge development. Technologies such as 5G, MQTT, and edge gateways ensure reliable and low-latency communication between cloud and edge environments. These technologies facilitate efficient and seamless data exchange, control, and coordination between cloud and edge components of a distributed application.

Robust security and privacy measures are essential for Cloud to Edge development. Technologies like Secure Enclaves, Trusted Execution Environments (TEEs), and Secure Boot guarantee the security and protection of edge devices and applications from threats. Additionally, data encryption, access controls, and authentication mechanisms are implemented to safeguard data and maintain privacy in distributed cloud-to-edge environments.

A brief review of the main platforms follows. AWS Greengrass [18] is an edge computing platform provided by Amazon Web Services (AWS). It allows developers to deploy Lambda functions, Docker containers, and machine learning models on edge devices. Greengrass also provides local messaging, caching, and data synchronization capabilities, allowing for offline operation and reduced latency. It integrates with other AWS services, enabling seamless communication and coordination between edge and cloud components. Greengrass also offers features such as automatic device management, security, and monitoring, making it easier to deploy and manage applications at the edge.

Azure IoT Edge [19] is an edge computing platform provided by Microsoft Azure. It allows developers to deploy containerized modules, including Docker containers and Azure Functions, on edge devices. Azure IoT Edge also provides features such as local data processing, device management, and security. It integrates with other Azure services, allowing for seamless communication and data processing between edge and cloud components. Azure IoT Edge also supports machine learning capabilities with Azure Machine Learning, enabling deployment of machine learning models at the edge for real-time inferencing.

Google Edge TPU [20] is a hardware accelerator designed specifically for edge computing. It provides high-performance, low-power processing capabilities for machine learning inferencing at the edge. The Edge TPU integrates with TensorFlow, a popular machine learning framework, enabling developers to deploy machine learning models at the edge for local inferencing. The Edge TPU is optimized for edge devices, such as IoT devices and edge servers, and provides hardware acceleration for efficient and fast inferencing capabilities.

Apache Edgent [21] is an open-source edge computing framework that provides a programming model and runtime for edge devices. It supports a wide range of devices and platforms, including Raspberry Pi, BeagleBone Black, and Android. It provides local compute, messaging, and data management capabilities, and can be integrated with other Apache projects, such as Kafka and Spark. Edgent offers a lightweight and modular framework that supports running analytics and processing data at the edge

of the network, closer to the data source.

Edgent provides connectors to various data sources, such as sensors, devices, and gateways, and supports a wide range of programming languages, including Java, Python, and JavaScript.

Edgent focuses on enabling low-latency and real-time data processing at the edge, with features such as data aggregation, filtering, and transformation.

Edgent also supports extensibility through its modular architecture, allowing developers to add custom analytics modules and integrate with other edge technologies.

EdgeX Foundry [22] is an open-source edge computing framework that provides a modular platform for building and deploying edge computing applications. It supports a wide range of devices and platforms, including Linux, Windows, and ARM-based devices. It provides local compute, messaging, and data management capabilities, and can be integrated with other open-source projects, such as Docker and Kubernetes.

Eclipse ioFog [23] is an open-source edge computing framework that provides a platform for developing, deploying, and managing edge applications and services.

ioFog offers a distributed architecture with a hierarchical approach, enabling the deployment and coordination of microservices across a network of edge devices and gateways.

ioFog provides features such as local data processing, message routing, and fog computing capabilities, allowing for processing data and running microservices at the edge.

ioFog also includes management capabilities for deploying, scaling, and monitoring edge applications, with a focus on providing a secure and scalable edge computing platform.

With a specific focus on Machine learning, the following platforms are the most commonly used. TensorFlow Lite for Edge [24] is a machine learning framework provided by Google that enables running machine learning models on edge devices. TensorFlow Lite for Edge is designed specifically for edge computing, with optimizations for running machine learning inference on resource-constrained devices, such as IoT devices and edge servers.

TensorFlow Lite for Edge provides a wide range of pre-trained models for tasks such as image recognition, object detection, and speech recognition, as well as tools for converting and deploying custom machine learning models.

TensorFlow Lite for Edge also offers support for hardware acceleration, allowing for efficient and fast inferencing on edge devices with dedicated hardware, such as GPUs and TPUs.

TensorFlow Lite offers features such as model quantization, which reduces the size of the models while maintaining inference accuracy, and hardware acceleration, which allows for efficient inferencing on edge devices with dedicated hardware, such as GPUs and TPUs.

ONNX Runtime [25] is an open-source runtime that is part of the Open Neural Network Exchange (ONNX) project, which aims to provide interoperability between different machine learning frameworks. ONNX Runtime supports running machine learning models on edge devices, as well as in the cloud, and provides optimizations for edge deployment, such as model quantization and hardware acceleration. ONNX Runtime is designed to be extensible and supports a wide range of machine learning frameworks, including TensorFlow, PyTorch, and scikit-learn, allowing for seamless

integration of models from different frameworks. ONNX Runtime also provides support for deploying machine learning models in different programming languages, such as C++, Python, and Java, making it versatile for edge deployment in various environments.

NVIDIA TensorRT [26] is an inference optimization runtime provided by NVIDIA that is designed to accelerate deep learning inferencing on NVIDIA GPUs. TensorRT provides optimizations such as layer fusion, precision calibration, and dynamic tensor memory management, which enable efficient and fast inferencing of deep learning models on edge devices with NVIDIA GPUs. TensorRT supports popular deep learning frameworks, including TensorFlow, PyTorch, and ONNX, and provides a TensorRT Python API for seamless integration with these frameworks.

TensorRT also includes support for INT8 and FP16 precision inference, which can further optimize the performance of machine learning models on edge devices with NVIDIA GPUs.

Performance and feature comparison of these Machine Learning at the Edge technologies depends on various factors, such as the specific use case, hardware resources available on the edge devices (e.g., GPU or CPU), and the type of machine learning models being deployed. Each technology has its own strengths and trade-offs. For example, TensorFlow Lite is focused on efficient inferencing on resource-constrained devices with a wide range of pre-trained models, ONNX Runtime provides interoperability with different machine learning frameworks and support for multiple programming languages, and NVIDIA TensorRT is optimized for inferencing on NVIDIA GPUs. It's important to evaluate the performance, features, and compatibility of these technologies based on the specific needs of your edge computing application.

Performance comparison of edge platforms depends on various factors, such as the specific use case, workload, and hardware resources available on the edge devices. Each platform has its own strengths and trade-offs. For example, AWS Greengrass provides a wide range of features and integrations with other AWS services, making it a comprehensive and scalable choice for edge computing. Azure IoT Edge offers strong integration with the Microsoft Azure ecosystem and provides capabilities for deploying and managing containerized modules. Google Edge TPU is focused on machine learning inferencing with dedicated hardware acceleration, making it a suitable choice for AI-driven edge applications.

Performance benchmarks and comparisons are typically workload-dependent and may vary based on specific requirements and configurations. It's important to evaluate the performance and suitability of edge platforms based on the specific needs. Considering the goal of zero-Swarm it should be given priority to features that enable intensive use of data streams and AI applications.

In summary, Cloud to Edge development technologies comprise a diverse range of tools, frameworks, and platforms that enable the creation of distributed applications spanning cloud and edge environments. These technologies supply the necessary tools and capabilities to build intelligent, real-time, and scalable applications that harness the strengths of both cloud and edge computing. The zeroSwarm project is designed to test a multi-layered infrastructure that combines the most suitable platforms, enabling the management of Cyber-Physical Systems of Systems (CSoS) through the EIC 6-1499 extension.

2.3 Cloud-Edge architectures: pros and cons

As reported in previous sections, Cloud-to-Edge architecture refers to a distributed computing model where data processing and computation are performed both in centralized cloud data centers and at the edge, which includes devices or nodes at the network's periphery. While this approach offers several advantages such as reduced latency, bandwidth optimization, and enhanced privacy and security, it also comes with its own set of challenges that must be addressed when designing and implementing solutions, Figure 3. The main advantages and typical problems of these architectures are then reported [27].

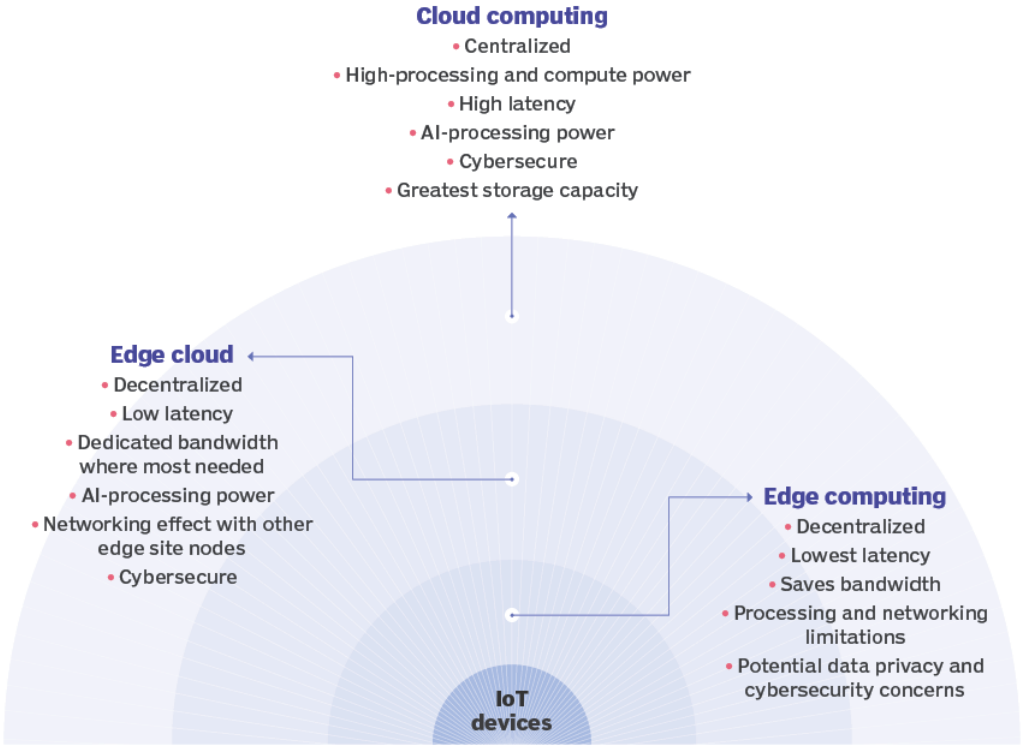

Figure 3: Cloud to edge main characteristics

Edge devices are geographically closer to the data source, which can significantly reduce the latency in processing time-sensitive data. For applications that require real-time or near-real-time responses, such as industrial automation, autonomous vehicles, or augmented reality, Cloud-to-Edge architecture can provide substantial performance benefits by minimizing data transfer delays and improving overall system responsiveness. Cloud-to-Edge architecture can help optimize network bandwidth by processing data at the edge before transmitting it to the cloud. This reduces the amount of data that needs to be transmitted to the cloud, thereby reducing the bandwidth requirements and associated costs. This can be especially beneficial in scenarios where bandwidth is limited or expensive, such as in remote or rural areas. Edge devices can process sensitive data locally without transmitting it to the cloud, which can help address privacy and security concerns. By keeping data locally on edge devices, Cloud-to-Edge architecture can mitigate risks associated with data breaches, unauthorized access, and compliance issues, as data is not transmitted over the network or stored in a centralized cloud.

However, there are also crucial issues that can impact the performance of Cloud-to-Edge architecture and must be considered when designing a zero-Swarm solution.

Edge devices typically have limited computational resources compared to the cloud, which can impact

the scalability of the system. Complex or resource-intensive applications may face limitations in terms of processing capabilities, memory, and storage at the edge. This can require careful distribution of tasks and workload management to ensure efficient utilization of resources. Deciding which algorithms should be deployed at the edge and which should be executed in the cloud can be challenging. Some algorithms may require significant computational resources and may be better suited for cloud execution, while others may need low-latency processing and are more suitable for edge execution. Efficiently managing the distribution of algorithms across the cloud and edge, and dynamically adapting the distribution based on changing conditions, can impact the overall system performance. In Cloud-to-Edge architecture, data may be processed and stored at different locations, leading to challenges in data synchronization and consistency. Ensuring that data remains consistent across cloud and edge, and managing data updates, caching, and versioning, can be complex and impact system performance. Cloud-to-Edge architecture relies heavily on network connectivity between the cloud and edge devices. Unreliable or intermittent network connections can disrupt data processing, result in delays, and impact system performance. Ensuring reliable and robust network connectivity, especially in remote or harsh environments, is crucial for the success of Cloud-to-Edge architecture.

2.4 An introduction to OPC-UA

OPC-UA (Open Platform Communications Unified Architecture) is a widely adopted industrial communication protocol that provides a standardized and interoperable way to exchange data between industrial devices, sensors, machines, and systems [28]. OPC-UA plays a significant role in the integration of industrial systems with edge computing and cloud computing environments, enabling efficient and secure data communication across the Cloud-to-Edge architecture, as described in Figure 4.

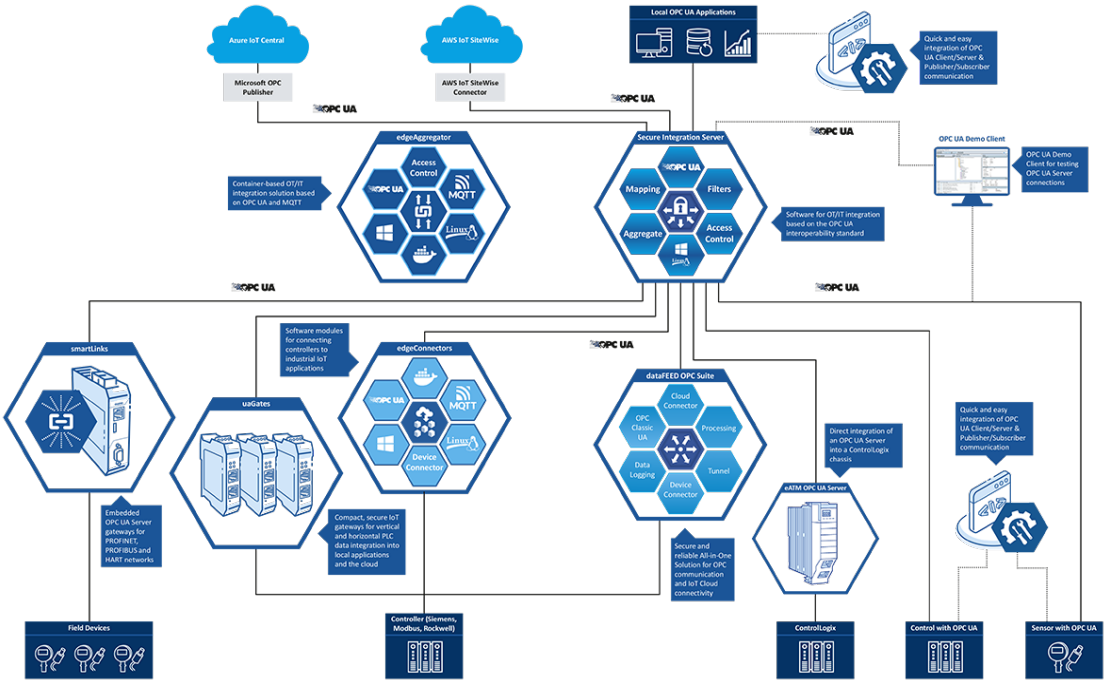

Figure 4: An example of OPC-UA architecture

OPC-UA [29] is designed to be platform-independent, vendor-neutral, and open, making it widely adopted across various industries such as manufacturing, energy, transportation, and building

automation. OPC-UA supports a range of programming languages and operating systems, allowing for flexible and adaptable implementation in different environments.

One of the primary roles of OPC-UA in the Cloud-to-Edge architecture is data acquisition. OPC-UA servers can be deployed on edge devices or gateways, enabling them to communicate with various industrial protocols and collect data from sensors, machines, and other devices. This data can then be processed, analyzed, and acted upon locally at the edge or transmitted to the cloud for further processing.

OPC-UA provides a standardized way to represent data from different industrial systems, regardless of the underlying protocols or data formats. OPC-UA defines a unified data model, which includes a hierarchical address space and a set of standardized data types, allowing for seamless data integration and interoperability across different devices and systems. This enables a consistent representation of data across the Cloud-to-Edge architecture, simplifying data processing and analysis.

OPC-UA includes robust security features to ensure the secure transmission and storage of data in the Cloud-to-Edge architecture. OPC-UA supports a variety of security mechanisms, including Transport Layer Security (TLS), Secure Sockets Layer (SSL), and authentication via certificates or username/password. This helps protect sensitive industrial data and ensures the integrity and confidentiality of data exchanged between edge devices, gateways, and the cloud.

OPC-UA supports a flexible and scalable Publish/Subscribe (Pub/Sub) model, where data can be published by OPC-UA servers and subscribed to by OPC-UA clients. This allows for efficient and asynchronous data communication between edge devices, gateways, and cloud systems, reducing the overhead of continuous polling and enabling real-time or near-real-time data processing and analysis.

Finally, OPC-UA provides interoperability between different vendors, devices, and systems, making it easier to integrate and exchange data across the Cloud-to-Edge architecture. OPC-UA defines a common set of communication services and data modeling conventions, enabling seamless integration of different industrial systems regardless of the underlying technologies or protocols.

Overall, OPC-UA [30][31] serves as a key enabling technology in the Cloud-to-Edge architecture, providing a standardized, secure, and efficient way to exchange data between industrial systems and cloud computing environments.

2.5 The role of Gate-way & Protocols such as MQTT

MQTT (Message Queuing Telemetry Transport) [32] is a standardized messaging protocol, or set of rules, used for machine-to-machine communication. It is commonly used by smart sensors, wearable devices, and other Internet of Things (IoT) devices that need to transmit and receive data over networks with limited resources and bandwidth. These IoT devices utilize MQTT for data transmission because it is easy to implement and efficiently communicates IoT data. MQTT supports messaging between devices and the cloud, as well as between the cloud and devices.

The MQTT protocol has become a standard for IoT data transmission and offers several benefits as described in Table 1.

Benefit	Description
Lightweight and	Implementing MQTT on IoT devices requires minimal resources, making it

Efficiency	suitable for even small microcontrollers. For example, a minimal MQTT control message can consist of just two bytes of data. Additionally, MQTT message headers are small in size, optimizing network bandwidth.
Scalability	MQTT implementation requires minimal code and consumes very little power during operations. The protocol also provides built-in functions to support communication with a large number of IoT devices. This enables the implementation of MQTT to connect with millions of these devices.
Reliability	Many IoT devices connect over unreliable cellular networks with limited bandwidth and high latency. MQTT incorporates built-in features that reduce the time for IoT devices to reconnect to the cloud.
Security	MQTT simplifies message encryption and device/user authentication for developers using modern authentication protocols such as OAuth, TLS1.3, Client Managed Certificates, and more.
Good Support	Several programming languages, such as Python, provide extensive support for MQTT implementation. Hence, developers can quickly implement MQTT with minimal coding in any application.

Table 1: MQTT benefits

The MQTT protocol was invented in 1999 for use in the oil and gas industry. Engineers needed a protocol that offered minimal bandwidth usage and minimal battery drain to monitor pipelines via satellite. Initially, the protocol was known as Message Queuing Telemetry Transport due to the IBM MQ series products it supported in its early stages. In 2010, IBM released MQTT 3.1 as an open and free protocol for anyone to use, which was later submitted to the Organization for the Advancement of Structured Information Standards (OASIS) for maintenance in 2013. In 2019, OASIS released the updated MQTT version 5. MQTT is now no longer an acronym but is considered the official name of the protocol.

The MQTT protocol operates based on the principles of the publish-subscribe model. In traditional network communication, clients and servers communicate directly with each other. Clients request resources or data from the server, which processes the requests and sends responses. However, MQTT uses a publish-subscribe scheme to decouple the message sender (publisher) from the message receiver (subscriber). Instead, a third component called the message broker manages the communication between publishers and subscribers. The broker's task is to filter all incoming messages from publishers and distribute them correctly to subscribers.

Before we delve deeper into MQTT, it is crucial to understand the publish-subscribe pattern, also known as the pub-sub pattern. This pattern enables decoupling between the client that publishes a message and the client or clients that receive the message. Clients are oblivious to the existence of other clients, and a publisher can send messages of a specific type, which are then received only by the clients interested in those types of messages. This pattern relies on a broker, also referred to as a server, which acts as an intermediary. All clients establish connections with the broker, and the client responsible for sending a message through the broker is called the publisher. The broker filters incoming messages and distributes them to the clients subscribed to the relevant message types. Clients that register themselves as subscribers establish connections with the broker accordingly. To

visualize this concept, let's consider the following scenario:

An Autonomous Mobile Robot (AMR) equipped with a battery sensor is configured to communicate with other devices.

In this scenario, an AMR acts as the publisher and sends a message with a payload of relevant sensor data to the broker. It is important to note that the payload refers to the data encapsulated within a message, including its associated topic. Subsequently, the broker distributes the message to the two clients that have subscribed to the "AMR" topic: Subscriber 1, representing which can be another AMR, and Subscriber 2, representing a Monitoring System. As shown in Figure 5: MQTT Publish-Subscriber.

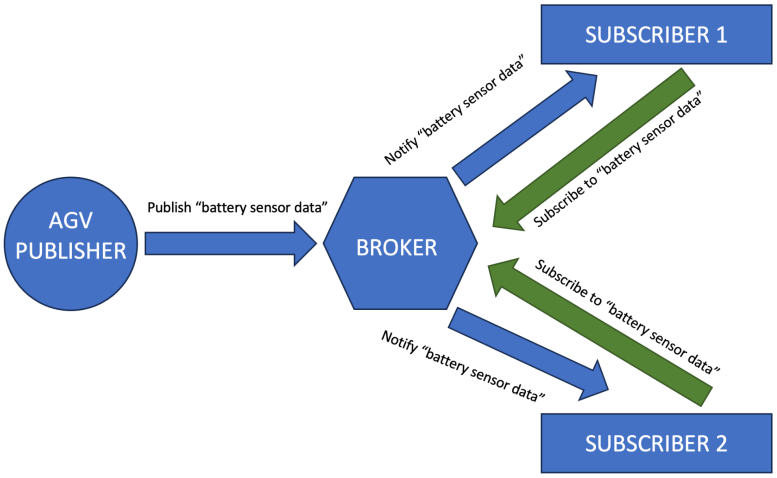

Figure 5: MQTT Publish-Subscriber

The publish-subscribe pattern ensures that publishers and subscribers are decoupled, meaning they are unaware of each other's existence. Furthermore, publishers and subscribers do not need to operate simultaneously. Publishers can send messages while subscribers receive them at a later time. Additionally, the publish operation is asynchronous and not synchronized with the receive operation. Publishers request the broker to publish a message, and different clients subscribed to the corresponding topics may receive the message at different times. Publishers have the option to send messages asynchronously, avoiding being blocked until clients receive them. Alternatively, they can choose to send messages synchronously, waiting for a successful operation from the broker before proceeding with further execution.

MQTT plays a crucial role in defining our architecture, allowing us to manage AMRs. MQTT, enables efficient communication and coordination between the various components. In this context, an AMR can be seen as an edge device that subscribes to the system.

MQTT provides a lightweight and scalable communication framework that facilitates real-time data exchange between AMRs, the management system, and other devices. It enables seamless integration and coordination within the system. In this way we obtain a decoupled and flexible communication pattern. AMRs act as publishers, sending updates on their status, tasks, or requests, while the management system and other components act as subscribers, receiving relevant information.

It is essential to set up an MQTT broker, establish the necessary network infrastructure, and integrate MQTT functionality into the AMRs' software or firmware. By leveraging MQTT's capabilities, we can

efficiently manage and coordinate the fleet of AMRs, enabling real-time data exchange, task assignments, and remote monitoring.

Edge gateways play a crucial role in enabling hybrid connectivity by providing the interface between edge devices and various connectivity technologies, such as 5G, MQTT, and other protocols, ensuring seamless communication and data processing between edge and cloud environments
Edge gateways can be deployed in different forms, such as dedicated hardware devices or virtualized software instances, depending on the specific use case and requirements.

2.6 Kafka introduction and comparison with common alternatives

Apache Kafka is a powerful open-source event streaming platform widely used for building scalable, real-time data pipelines. It has gained significant popularity in recent years and has become a cornerstone of modern data infrastructure. Kafka provides a distributed, fault-tolerant system that enables high-throughput, low-latency data streaming, making it an ideal choice for applications that require real-time data processing and analysis. At the core of Kafka's architecture is the pub-sub model [36]. Producers publish messages to topics, and consumers subscribe to those topics to receive the messages in real time. This decoupling of producers and consumers allows for high scalability and flexibility in data distribution. Kafka ensures durability by persisting messages to disk, providing fault tolerance and enabling data recovery even in the event of failures. Kafka's robustness, scalability, and versatility have made it the go-to choose for many organizations dealing with high volumes of streaming data. It has found applications in various industries, including finance, e-commerce, social media, and more. Its ability to handle large-scale data ingestion and processing has made it a fundamental component of data-driven architectures.

Figure 6 shows the Kafka model. It consists of a producer that write data to topics, consumers that read data from topics, and a broker that handle storage and replication of data. Kafka enables high-throughput, fault-tolerant, and scalable data processing in real-time that can be available to the consumer.

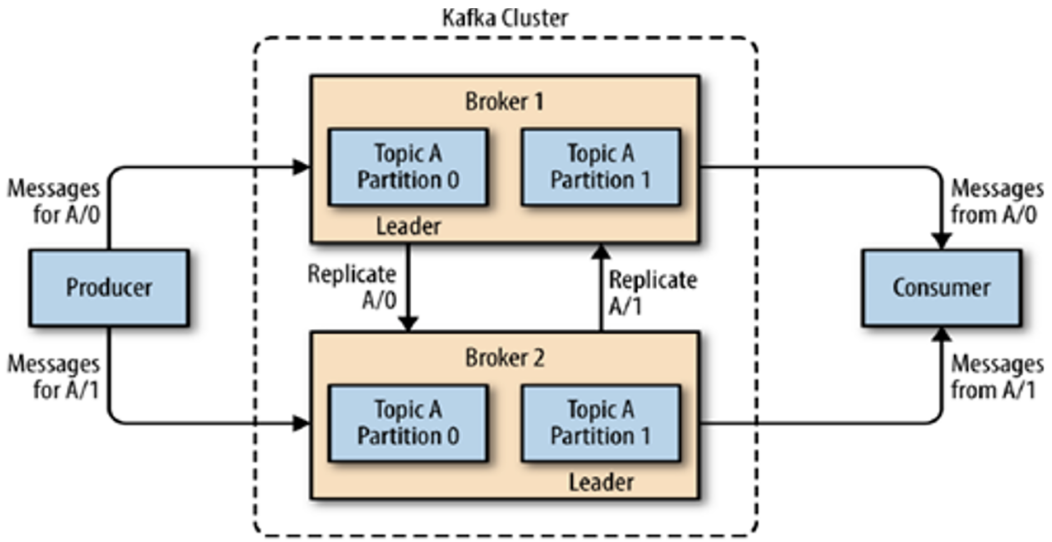

Figure 6: Kafka Conceptual Architecture

Kafka's ability to collect data from various edge devices, sensors, and monitoring systems is a

fundamental functionality that greatly enhances edge system monitoring. With Kafka, these devices can publish data directly to Kafka topics using Kafka producers. This direct publication mechanism eliminates the need for complex data integration processes and allows for efficient and centralized data collection [35][36][37]. The data collected can include system metrics, sensor readings, log files, or any other relevant information from the edge environment. By leveraging Kafka for data collection, organizations can overcome the challenges associated with heterogeneity in edge systems. Edge devices and sensors often use different protocols, formats, or data structures, making data integration a complex task. However, Kafka's flexibility in handling various data formats and protocols simplifies this process. It acts as a unified data bus that can receive and store data from disparate sources, enabling seamless integration and ensuring that no data is lost.

Furthermore, real-time streaming is a crucial functionality provided by Kafka that enables edge system monitoring to keep pace with the dynamic nature of edge environments. Kafka's distributed streaming capabilities allow data to be streamed in real-time from edge devices to centralized monitoring systems. This real-time streaming capability ensures that the monitoring system has immediate visibility into the state of the edge systems, allowing for timely response to critical events or anomalies [38]. The consumption of data through Kafka consumers further enhances real-time monitoring capabilities. Kafka consumers can perform real-time analysis, aggregation, or filtering on the streaming data, enabling organizations to gain valuable insights and detect patterns or anomalies in real-time. For example, by applying stream processing algorithms within the Kafka ecosystem, organizations can identify potential equipment failures, trigger proactive maintenance actions, or detect anomalies that may indicate security breaches.

In edge system monitoring, where intermittent network connectivity or device failures are common, ensuring the reliability of data capture and processing is essential. Kafka's fault-tolerant design addresses these challenges effectively. Kafka allows for data replication and distribution across multiple Kafka brokers, ensuring that even if a broker or a network connection fails, the data is still available for processing and consumption. The replication of data across Kafka brokers provides resiliency to edge system monitoring applications. Each Kafka broker can act as a leader or a follower for different partitions of a Kafka topic [39]. If a leader broker fails, one of the follower brokers automatically takes over, ensuring continuous data availability. This fault-tolerant design mitigates the risk of data loss and ensures that monitoring systems can rely on the data collected from edge devices, even in challenging and dynamic edge environments.

Scalability is also a critical aspect of edge system monitoring, as the number of edge devices and the volume of data generated can increase rapidly. Kafka's distributed nature makes it highly scalable, allowing it to handle high volumes of data from numerous edge devices. As the number of devices or data sources increases, additional Kafka brokers can be added to the cluster to handle the increased load effectively [40]. The ability to add more Kafka brokers to the cluster without impacting the overall system performance ensures that edge system monitoring can scale seamlessly as the deployment grows. This scalability is particularly valuable in scenarios where the number of edge devices may vary dynamically, such as in industrial IoT deployments or smart city environments. Kafka's scalability enables organizations to accommodate the evolving needs of edge system monitoring without compromising performance or data integrity.

Kafka's role as a data ingestion layer enables seamless integration between edge systems and cloud-based monitoring and analytics platforms. In edge-to-cloud integration, edge devices publish data to

Kafka, which then acts as a bridge, forwarding the data to cloud services for further analysis, long-term storage, or visualization. This integration brings several advantages to edge system monitoring [41]. First, it allows organizations to leverage the power of cloud-based analytics and machine learning algorithms to gain deeper insights from the data collected at the edge. Cloud platforms can provide advanced analytics capabilities, such as predictive maintenance, anomaly detection, or optimization algorithms, which can be applied to the data ingested through Kafka. Second, edge-to-cloud integration enables long-term storage of data in scalable and durable cloud storage systems. While edge devices often have limited storage capacities, cloud storage can accommodate large volumes of historical data, allowing for comprehensive analysis and retrospective investigations. This is particularly valuable for compliance, auditing, or performance optimization purposes. Third, cloud-based visualization and reporting tools can leverage the data ingested through Kafka to provide rich dashboards, real-time monitoring interfaces, or automated reporting. This enhances situational awareness and decision-making capabilities, enabling stakeholders to have a holistic view of the edge system's performance, health, and operational metrics.

Kafka's integration with stream processing frameworks, such as Apache Kafka Streams or Apache Flink, opens up opportunities for performing on-the-edge data processing and analytics. This capability empowers edge devices to perform real-time data transformations, aggregations, or anomaly detection directly within the Kafka ecosystem before sending processed data to centralized systems [42]. By leveraging Kafka's integration with stream processing frameworks, organizations can address latency, bandwidth, or privacy constraints associated with transmitting large volumes of data to centralized processing infrastructures. Edge devices can apply custom business logic or predefined algorithms to the streaming data, enabling real-time decision-making and immediate action at the edge. For example, an edge device can analyze sensor data to detect abnormal patterns, trigger local alarms, or adjust its own behavior autonomously, without relying on centralized systems. The on-the-edge data processing capability provided by Kafka fosters distributed intelligence and decision-making, reducing dependency on centralized resources and network connectivity. It enables organizations to utilize the computational power available at the edge efficiently and empowers edge devices to respond rapidly to local events or changing conditions.

While Kafka remains a popular choice, several alternatives have emerged in recent years, offering similar or enhanced functionalities.

2.7 Comparison with common alternatives

Apache Kafka has established itself as a leading event streaming platform, but in recent years, several alternatives have emerged in the market. This section will provide a detailed comparison of Kafka with some of the common alternatives, namely Apache Pulsar, Redpanda, StreamSets, and Estuary Flow.

Apache Pulsar is an open-source event streaming platform that shares similar functionality with Kafka. Like Kafka, Pulsar offers strong durability and scalability, making it suitable for handling large volumes of data. One key advantage of Pulsar is its support for multi-layered topics, which provides greater flexibility and simplifies data organization. By dividing topics into multiple layers, Pulsar allows for more granular control over data routing and enables efficient data processing workflows [43]. Additionally, Pulsar provides native support for both streaming and batch processing, which expands its use case possibilities compared to Kafka. This versatility makes Pulsar suitable for scenarios that

require both real-time processing and offline analytics. Furthermore, Pulsar offers multi-tenancy support, allowing different users or organizations to share a single Pulsar cluster while maintaining data isolation. This feature is particularly useful in multi-tenant environments where data privacy and resource allocation are crucial considerations.

Redpanda is a streaming platform built as a Kafka-compatible alternative. It aims to address some of the limitations and complexities associated with Kafka while maintaining compatibility with Kafka's APIs and ecosystem. Redpanda offers improved performance, lower latency, and simpler administration, making it an attractive option for organizations with demanding real-time applications. One of the key advantages of Redpanda is its ability to leverage modern hardware features and techniques to deliver enhanced performance. It takes advantage of technologies such as kernel bypass and zero-copy networking to reduce latency and improve throughput. These optimizations make Redpanda well-suited for high-throughput, low-latency use cases, where timely data processing is critical. Moreover, Redpanda aims to simplify administration tasks by providing a more intuitive interface for managing clusters and topics. It offers a simplified deployment model, making it easier to set up and configure compared to Kafka. Redpanda's compatibility with Kafka's APIs allows organizations to migrate their existing Kafka applications to Redpanda seamlessly, reducing the barrier to adoption for those already invested in Kafka.

While not a direct alternative to Kafka, StreamSets is a data integration platform that can work with Kafka, among other technologies. StreamSets provides a visual interface for designing, executing, and managing data pipelines. It simplifies the process of ingesting and processing data from various sources, including Kafka, by offering built-in connectors and transformations. StreamSets focuses on making data integration and pipeline development easier, especially for organizations that may not have dedicated engineering resources for managing complex streaming platforms. Its visual interface allows users to design data flows using a drag-and-drop approach, reducing the need for manual coding. This abstraction layer makes it accessible to a wider range of users, including data analysts and less technically inclined individuals, enabling them to create and manage data pipelines efficiently [45]. In addition to its ease of use, StreamSets offers robust functionality for data governance, monitoring, and error handling. It provides features like data lineage, data quality checks, and error tracking, which are crucial for maintaining data integrity and reliability in streaming pipelines. By integrating with Kafka, StreamSets offers organizations a streamlined solution for managing their data integration needs.

Estuary Flow is a relatively new player in the real-time data infrastructure space, aiming to provide a simpler and more frictionless alternative to Kafka. It offers an easy-to-use, cloud-native event streaming platform designed to handle large-scale data ingestion and processing. Estuary Flow abstracts away much of the underlying complexity associated with Kafka while still delivering reliable and performant data streaming capabilities. One of Estuary Flow's primary focuses is on simplicity. It provides a user-friendly interface and intuitive workflows for managing data pipelines, making it accessible to a broader range of users. The platform handles the complexities of topics, partitions, and consumer groups behind the scenes, allowing users to focus on their data processing logic instead of managing infrastructure details. Scalability is another area where Estuary Flow excels. It is built to handle large-scale data workloads, offering horizontal scalability by automatically scaling resources based on demand. This elastic scaling capability enables organizations to handle varying data volumes without manual intervention, ensuring smooth operations during peak loads. Estuary Flow also

emphasizes its cloud-native design, leveraging the scalability and resilience of cloud platforms. It seamlessly integrates with popular cloud providers, allowing users to take advantage of managed services for storage, compute, and monitoring. By utilizing cloud-native capabilities, Estuary Flow reduces the operational overhead of managing infrastructure and simplifies the deployment and management process.

2.8 Kafka introduction and comparison with common alternatives Current Limits of the Cloud-to-Edge technologies

Cloud-to-Edge technologies have gained popularity due to the rise of IoT and edge computing use cases. However, certain challenges need to be addressed for the effective implementation of these technologies. One significant limitation is the limited processing power of edge devices compared to cloud servers. This constraint restricts the amount of data that can be processed at the edge and may necessitate sending data to the cloud for further analysis. Additionally, edge devices may have limited storage capacity, making it challenging to store large amounts of data locally. This limitation also hampers the ability to perform local analytics, often requiring data to be sent to the cloud for storage. Connectivity issues pose another obstacle for Cloud-to-Edge technologies. Edge devices often operate in environments with limited or intermittent connectivity, which makes it difficult to send data to the cloud for processing and analysis. This limitation also affects real-time analytics at the edge.

Security concerns are a significant consideration when implementing Cloud-to-Edge technologies. Edge devices are more vulnerable to security threats, including hacking and data breaches, which can compromise the overall network security. Therefore, additional security measures need to be in place to safeguard edge devices and data. Integration challenges arise when dealing with various edge devices and cloud platforms. The process of integrating different devices and platforms can be complex, requiring significant investment in time and resources to ensure compatibility and interoperability [64]. Cost is another limitation associated with Cloud-to-Edge technologies. Implementing these technologies involves substantial capital and operating costs, especially when deploying and managing large-scale edge computing infrastructure.

To overcome these limitations, a combination of technical and non-technical solutions is required. From a technical perspective, deploying suitable edge computing infrastructure like AWS Greengrass, Microsoft Azure IoT Edge, and Google Cloud IoT Edge can provide local compute, messaging, and machine learning capabilities, even in the absence of an internet connection. Improved connectivity can be achieved through the use of high-speed wireless networks such as 5G. Hybrid networking architectures combining edge computing with cloud computing can address connectivity challenges by enabling data transfer between edge devices and the cloud [65]. Security measures, including encryption, access control, and data integrity checks, need to be implemented to mitigate security concerns. Device management solutions such as remote monitoring and over-the-air updates can further enhance security. Integration platforms like Apache NiFi can simplify the integration of diverse edge devices and cloud platforms by providing connectors and adapters for different data sources and platforms. Cost optimization strategies like leveraging open-source software, containerization, automation, and serverless computing can help reduce expenses and enhance efficiency.

In the context of the Zero Swarm Trials, certain limits need to be considered when implementing cloud-to-edge technologies. Latency can impact real-time decision-making and responsiveness, while

bandwidth limitations can pose challenges when multiple AMRs generate continuous data streams. Optimizing algorithms for resource-constrained AMRs is crucial, and robust security measures must be in place to protect the AMR fleet and sensitive data. Scalability, maintenance, and updates are also important factors to consider in the context of an expanding AMR fleet. By addressing these limits and tailoring solutions to the specific requirements of the AMR fleet, the benefits of cloud-to-edge technologies can be harnessed while overcoming the challenges related to latency, bandwidth, edge device capabilities, security, scalability, and maintenance.

3 Zero-SWARM Distributed Streaming Computing in the Cloud-to-Edge continuum. Propose Architecture

3.1 Kafka & Edge System Monitoring using MQTT

In today's interconnected world, the monitoring of edge systems has become increasingly important. Edge systems encompass a wide range of devices, sensors, and monitoring systems that are deployed at the periphery of a network, closer to the data source. These systems generate vast amounts of data that need to be collected, analyzed, and monitored in real-time to ensure efficient operation and timely decision-making. This is where Kafka, a distributed streaming platform, comes into play. Kafka provides a robust and scalable solution for edge system monitoring, offering a variety of functionalities that are well-suited for this purpose. One of the key functionalities of Kafka in this context is data collection. Kafka can seamlessly receive data from various edge devices, sensors, and monitoring systems. These devices can publish data directly to Kafka topics using Kafka producers, allowing for a centralized and efficient data collection mechanism. Furthermore, Kafka's distributed streaming capabilities enable real-time streaming of data from edge devices to centralized monitoring systems [47]. By consuming the data through Kafka consumers, real-time analysis, aggregation, or filtering can be performed, providing immediate visibility into the state of the edge systems. This real-time streaming capability is crucial for effective monitoring and rapid response to any issues or anomalies detected at the edge.

Another important aspect of Kafka in edge system monitoring is its fault-tolerant design. Kafka ensures that data is reliably captured and processed, even in the presence of intermittent network connectivity or device failures. By allowing data replication and distribution across multiple Kafka brokers, Kafka provides resiliency to edge system monitoring applications, minimizing the risk of data loss or system downtime. Scalability is yet another strength of Kafka. As the number of edge devices or data sources increases, additional Kafka brokers can be added to the cluster, allowing the system to handle the growing volume of data. This scalability ensures that edge system monitoring can accommodate deployments of any size, from small-scale setups to large-scale enterprise environments [48]. Moreover, Kafka facilitates edge-to-cloud integration, serving as a data ingestion layer that connects edge systems to cloud-based monitoring and analytics platforms. With Kafka, edge devices can publish data to Kafka topics, which can then be forwarded to cloud services for further analysis, long-term storage, or visualization. This integration enables hybrid monitoring solutions that leverage both edge and cloud resources, combining the benefits of real-time monitoring at the edge with the power of cloud-based analytics. Additionally, Kafka's integration with stream processing frameworks like Apache Kafka Streams or Apache Flink allows for on-the-edge data processing and analytics. This means that

edge devices can perform real-time data transformations, aggregations, or anomaly detection directly within the Kafka ecosystem before sending processed data to centralized systems. This capability enables edge systems to derive insights and make informed decisions in real-time, without the need for data to be transported to a separate processing infrastructure.

Combining Kafka and MQTT in a manufacturing cloud-edge architecture offers a robust and scalable solution for data streaming and communication. While Kafka excels in data streaming, integration, persistence, and stream processing, MQTT shines in lightweight and efficient communication, the publish-subscribe model, and edge-to-cloud connectivity. In zero-Swarm, by leveraging the strengths of both technologies, a comprehensive and efficient architecture can be established for manufacturing environments [49]. Kafka plays a critical role in data streaming and integration. It can collect, stream, and integrate large volumes of data from various sources, including edge devices, sensors, and systems distributed across the production line or intra-logistics devices. Kafka's ability to handle high data volumes and its support for real-time streaming make it an ideal choice for collecting data from manufacturing assets and enabling seamless integration with downstream systems.

Furthermore, Kafka provides durable storage of data streams, allowing for the retention of data for a longer period. This feature is invaluable for historical analysis, replaying events, and serving as a reliable data source for downstream systems. By leveraging Kafka's data persistence capabilities, manufacturers can gain insights from historical data and perform in-depth analysis of manufacturing processes. In addition, Kafka's integration with stream processing frameworks such as Kafka Streams or Apache Flink enables real-time data transformations, analytics, and complex event processing. These frameworks can be utilized to perform data enrichment, aggregation, or anomaly detection on the collected manufacturing data. By leveraging Kafka's stream processing capabilities, manufacturers can derive real-time insights, detect anomalies, and trigger timely actions for process optimization and quality control [50]. On the other hand, MQTT serves as a lightweight and efficient communication protocol, specifically designed for constrained environments. It is well-suited for edge devices with limited resources and intermittent connectivity, which are often found in manufacturing environments. MQTT enables efficient communication between edge devices and cloud or edge servers, ensuring reliable transmission of real-time data and device management commands. The publish-subscribe model of MQTT allows devices to publish messages to specific topics, and interested subscribers receive those messages. This decoupled communication model enables efficient data distribution across the manufacturing system. MQTT brokers receive data from edge devices and publish it to relevant Kafka topics, enabling seamless integration between MQTT and Kafka [51]. This integration ensures that data published by edge devices can be efficiently consumed and processed by Kafka consumers or stream processing frameworks. Moreover, MQTT facilitates edge-to-cloud connectivity, enabling communication between edge devices and cloud services. Edge devices can publish data to MQTT brokers, which then forward the data to subscribed consumers, including cloud-based analytics platforms or Kafka for further processing. This integration allows for the seamless flow of data between edge devices and cloud-based systems, enabling comprehensive monitoring, analysis, and decision-making in manufacturing environments.

As shown in Figure 7, Kafka and MQTT are integrated together to accomplish the data needs of the consumer.

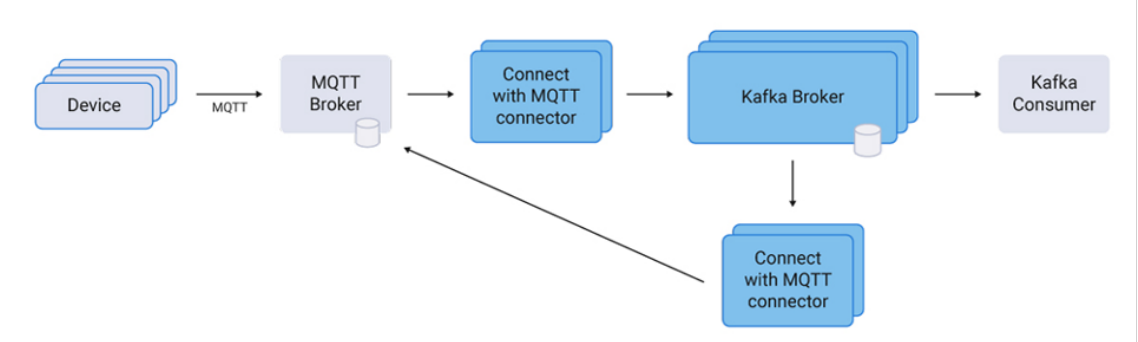

Figure 7: Kafka and MQTT integration

3.2 Integration of Kafka into the Zero Swarm System

The integration of Kafka into the Zero Swarm system, for example referring to the management of the Automated Guided Vehicle (AMR) fleet, set of subscribers, enhances the system's messaging and communication capabilities. Kafka serves as a reliable and scalable messaging system for handling data streams and facilitating communication between the AMR and the subscribers.

Kafka plays a crucial role in facilitating real-time data streaming from the AMR within the Zero Swarm system. The AMR generates various data types such as sensor readings, status updates, or other relevant information. These data streams can be published as messages to specific Kafka topics, ensuring that the data is available for consumption by the subscribers in real-time [52]. By leveraging Kafka for data streaming, the Zero Swarm system benefits from the inherent advantages of Kafka's distributed streaming capabilities. The AMR can publish data to Kafka topics, and Kafka takes care of efficiently distributing the data across the Kafka brokers. This ensures that the data is reliably delivered to the subscribers, allowing them to receive real-time updates and stay synchronized with the AMR's status and activities.

To handle the messaging infrastructure of the Zero Swarm system, a Kafka cluster is set up. The cluster consists of multiple Kafka brokers running on different servers, providing fault tolerance and scalability. The AMR acts as a Kafka producer, publishing messages to the Kafka cluster, and the subscribers connect to the cluster as Kafka consumers. The Kafka cluster's fault-tolerant design ensures the reliability of message delivery even in the presence of failures. Each message published by the AMR is stored in one or more Kafka brokers, with replication across multiple brokers. If a broker fails, the data is still accessible from other brokers, ensuring uninterrupted communication between the AMR and the subscribers [53]. Moreover, the Kafka cluster's scalability is a significant advantage in the Zero Swarm system. As the number of subscribers or the data volume increases, additional Kafka brokers can be added to the cluster. This allows the system to handle the growing load and ensures that the messaging infrastructure can scale seamlessly to accommodate the increasing demands of the Zero Swarm system.

In Kafka, topics are used to categorize and organize messages. In the Zero Swarm system, Kafka topics can be defined to categorize the different types of messages generated, for example, by the AMR. For example, topics can be created for sensor readings, status updates, or specific events. This categorization enables efficient message routing and selective consumption by the subscribers based on their interests and requirements. Furthermore, Kafka topics can be divided into multiple partitions, which allow for parallel processing and distribution of messages across the Kafka brokers. Each

partition can be assigned to a specific Kafka broker, enabling load balancing and increasing the system's throughput. Partitioning also ensures that messages within a topic can be processed in parallel, enhancing the overall efficiency of the Zero Swarm system.

The set of subscribers in the Zero Swarm system connect to the Kafka cluster as Kafka consumers. They subscribe to the relevant Kafka topics based on their interests and requirements. As the AMR publishes messages to the Kafka topics, Kafka ensures that the messages are delivered to the subscribed consumers in real-time. This enables the subscribers to receive timely updates and stay informed about the AMR's status and activities. The subscribers can process the received messages according to their specific needs. They can perform real-time analysis, trigger actions based on certain events, or update their own internal state based on the received data [54]. Kafka's efficient message distribution mechanism ensures that each subscriber receives all the messages from the subscribed topics, allowing for synchronized communication and coordinated actions within the Zero Swarm system.

The Zero Swarm system can leverage several key features of Kafka to enhance its messaging and communication capabilities. First, Kafka's scalability ensures that the system can handle increasing data volumes or accommodate additional subscribers in the future. This scalability is particularly valuable in scenarios where the Zero Swarm system may expand, such as adding more AMRs or integrating with a larger ecosystem of devices and applications. Second, Kafka's fault-tolerant design guarantees reliable message delivery even in the face of failures. If a Kafka broker or a network connection experiences a temporary disruption, Kafka ensures that the messages are still accessible from other brokers, preventing data loss and maintaining continuous communication within the Zero Swarm system. Additionally, Kafka's support for data retention is advantageous for the Zero Swarm system. Subscribers can consume messages even if they join the system later or experience temporary disruptions. Kafka retains messages for a configurable period, allowing new subscribers to catch up on missed messages and ensuring data availability for subscribers that may have intermittent connectivity. The combination of Kafka's scalability, fault tolerance, and data retention features provides a robust foundation for reliable and efficient messaging and communication within the Zero Swarm system. It enables seamless data streaming, message delivery, and synchronization between the AMR and the subscribers, facilitating effective coordination and collaboration in the context of the system's objectives.

3.3 OPC-UA and the combination with Kafka and MQTT

The combination of OPC-UA, Kafka, and MQTT in Cloud-to-Edge architectures offers a range of possibilities for efficient and effective data communication. One approach is using OPC-UA to acquire data from edge devices and systems, which can then be published to Kafka topics using OPC-UA servers as Kafka producers. Kafka's distributed streaming platform excels at handling large data streams in real-time, allowing for scalable and high-throughput communication between the edge and the cloud [55]. By organizing data streams into Kafka topics, ingestion and processing in the cloud become more efficient, supporting analytics, machine learning, and other applications.

Another integration option involves publishing OPC-UA data to MQTT topics through OPC-UA servers acting as MQTT publishers. MQTT, a lightweight messaging protocol widely used in IoT and edge computing, facilitates low-overhead and efficient data transmission. This bi-directional communication protocol is suitable for sending data from the edge to the cloud and receiving commands or instructions from the cloud to the edge. MQTT brokers can effectively manage and route MQTT

messages, ensuring scalable and reliable communication between the edge and the cloud.

Furthermore, Kafka or MQTT can transmit data to OPC-UA servers acting as subscribers, allowing bidirectional communication. This enables data to flow from the edge to the cloud via Kafka or MQTT for cloud-based processing or analysis. The results can then be sent back to the edge via OPC-UA, enabling near-real-time decision-making based on cloud-based analytics or machine learning models [56]. This approach empowers the edge with valuable insights and allows for local actions based on the cloud's processing capabilities. In zero-Swarm, by combining OPC-UA with Kafka and MQTT, a scalable and flexible Cloud-to-Edge architecture can be created. OPC-UA ensures standardization, security, and interoperability for industrial data, while Kafka and MQTT provide scalable and efficient messaging capabilities for data ingestion, processing, and transmission. The specific combination and configuration of these technologies depend on the requirements and use cases of the Cloud-to-Edge architecture in question.

In the context of the Zero Swarm project, the combination of OPC-UA, MQTT, and Kafka proves to be a robust and efficient system for managing the fleet of AMRs (Automated Guided Vehicles). OPC-UA integration establishes standardized and secure connectivity with the AMRs, enabling access to real-time data such as position, status, and battery levels. MQTT facilitates lightweight and seamless communication between the algorithm and the AMRs, enabling control over movements, routes, and tasks [57]. Kafka serves as a distributed streaming platform for processing the high-volume AMR data, supporting real-time monitoring, historical analysis, and advanced analytics. This integrated architecture empowers the algorithm to effectively manage and optimize the fleet's operations, enhancing overall performance and productivity. Furthermore, establishing an effective connection between edge and cloud environments [58] using IT protocols like Kafka over MQTT requires careful consideration of various factors. These include the specific requirements of the use case, technical capabilities of the edge and cloud systems, and the overall architecture and infrastructure of the organization. While the following guidelines are not exhaustive, they provide a starting point for designing a robust edge-cloud connection using Kafka over MQTT. First and foremost, it is essential to choose the right protocol based on your requirements and constraints. Kafka excels at handling high volumes of data in real-time, making it suitable for scenarios like IoT applications. On the other hand, MQTT is designed for constrained environments with limited processing power and bandwidth. Select the protocol that aligns best with your specific needs. Efficient data serialization is crucial when transmitting data between edge and cloud environments. This involves minimizing bandwidth usage and reducing processing overhead. To achieve this, opt for a lightweight and efficient serialization format such as Protocol Buffers, which is supported by both Kafka and MQTT. Implement proper data serialization and deserialization techniques in your edge and cloud applications. Connectivity in edge devices can be unreliable due to remote locations or intermittent network issues. It is vital to design a robust and fault-tolerant edge-cloud connection to ensure reliable data delivery. Implement mechanisms like message queuing, buffering, and retries to overcome connectivity disruptions and maintain data integrity.

Security should be a top priority when transmitting data between edge and cloud environments. Edge devices are often vulnerable to security threats, and data protection becomes paramount. Implement security best practices such as data encryption, authentication, and authorization to safeguard your data. Utilize secure communication protocols like SSL/TLS for transmitting data between edge and cloud. Considering the limited computing resources in edge environments, optimizing the performance

of the edge-cloud connection is crucial. Minimize the overhead of data serialization, compression, and encryption to reduce resource consumption. Efficient messaging patterns like batching and compression can further reduce network data transmission. Additionally, employ edge caching and data aggregation techniques to minimize data sent to the cloud and optimize processing logic for reduced latency and improved performance. Moreover, scalability and extensibility are essential factors to consider when designing the edge-cloud connection. Ensure the architecture can handle increasing data volumes and a growing number of devices. Plan for future changes in requirements and technology by designing the connection to be flexible and adaptable. Thorough testing and monitoring are crucial to ensure the performance, reliability, and security of the edge-cloud connection. Conduct comprehensive testing in realistic environments to validate the effectiveness of the design. Set up robust monitoring and logging mechanisms to detect and diagnose issues in real-time. Regularly review and update the edge-cloud connection to align with changing requirements and ensure optimal performance.

3.4 A conceptual harmonized Edge-to-Cloud Architecture

The Figure 8 depicts a conceptual system architecture for Zero Swarm project comprising several interconnected layers: MQTT Broker, Cloud, Cloud to Edge, Edge Getaway and Edges Devices. Each of these components plays a crucial role in enabling efficient data communication and processing within the system. At the center of the diagram is the MQTT Broker, which serves as a message broker, facilitating the communication between various devices and applications. As reported above, MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol commonly used in IoT (Internet of Things) applications due to its low overhead and efficient data transfer capabilities. The MQTT Broker acts as a central hub, receiving and distributing messages among the different components of the system.

Starting from the bottom, the Edges Devices section in the diagram represents a collection of edge nodes, which are distributed devices or sensors located close to the data source or the edge of the network. These edge nodes are equipped with various capabilities, such as PLC logic, robots, and also edge AI (Artificial Intelligence). Whereas, PLC logic refers to programmable logic controllers, which are industrial control systems used for automation and monitoring of machinery or processes. Robots within the edge nodes can perform tasks autonomously or with minimal human intervention (shopfloor operators for example). Edge AI refers to the deployment of AI algorithms and models directly on edge devices, enabling real-time processing and decision-making capabilities, using ONNX technology.

The diagram also includes an edge gateway, which serves as a crucial component in the system architecture. An edge gateway is a device that acts as a bridge between the edge nodes and the broader network, facilitating communication and data transfer between the two. It provides a secure and reliable connection between the edge environment and the external networks or cloud infrastructure. The edge gateway plays a vital role in the system by enabling connectivity and interoperability between the diverse range of edge nodes and the rest of the network. It serves as a centralized point for managing and controlling the flow of data between the edge environment and the external systems. This layer has also a system monitoring and data stream management module based on Kafka. Kafka is a distributed streaming platform that enables the collection, storage, and real-time processing of large-scale data streams. It acts as a highly scalable and fault-tolerant backbone for

data streaming within the edge server environment. By incorporating Kafka into the architecture, the system gains the ability to monitor and analyze data flowing through the edge nodes in real-time, enabling efficient data processing and decision-making at the edge. Note that all modules communicate downwards using the OPC-UA protocol.

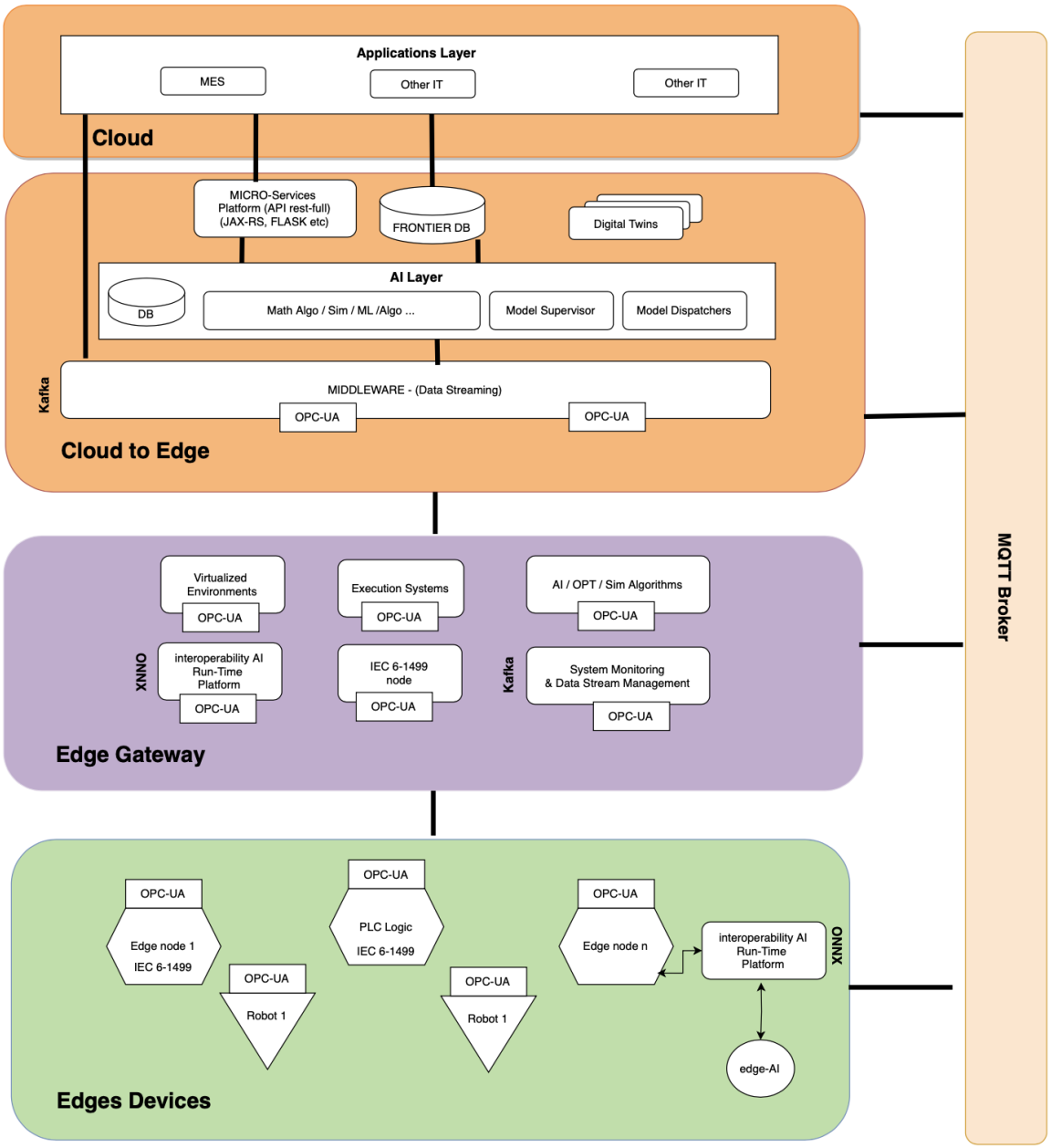

Figure 8: Zero Swarm Conceptual Architecture

The Cloud components, belonging to Cloud and Cloud to Edge layers, in the diagram represent a centralized and scalable computing infrastructure that provides additional processing power, storage, and advanced services for the system. Those layers are divided into three main modules: the application layer, AI layer, and middleware for data streaming. The application layer within the cloud encompasses various software applications and services that utilize the data generated by the edge nodes and the edge server environment. These applications can range from data analytics and

visualization tools to higher-level applications specific to the use case of the system. The application layer enables users to access and interact with the system, extracting insights and making informed decisions based on the data collected from the edge nodes.

Deliverable D4.4: Federated data infra & toolkit for data-driven model v1 [66], provides a possible implementation architecture of the Cloud and Cloud to Edge layers, showing the high-level architecture of several components that can be implemented to satisfy the requirements of storing the data gathered from the shop floor and use them to train AI/ML models.

The Frontier DB of the Cloud to Edge Layer can be implemented using the Collecting Platform referenced in [66], that aims to gather various data types from the MQTT broker showed in Figure 8 and direct them to designated destinations. Its primary objective is to streamline the integration between the Industry 4.0 domain, which employs its unique formats, data types, and protocols, and the IT/OT domain, which encompasses different types of information. The data gathered can be made available to another component describe in [66], the MLOps Framework, that can be used in the AI sub-layer in the Edge to Cloud layer of figure 8.

The AI layer in the cloud is responsible for hosting and executing advanced AI algorithms and models. By leveraging the computational resources of the cloud, the AI layer can perform complex data analysis, machine learning, and predictive modeling tasks. This layer can enable sophisticated analytics and decision-making capabilities by processing and interpreting the data collected from the edge nodes. The middleware, represented in the cloud component, serves as a bridge between the edge nodes and the cloud. It facilitates the seamless and efficient transfer of data from the edge environment to the cloud for further processing and analysis. The middleware layer ensures reliable and secure data streaming, enabling the continuous flow of data from the edge nodes to the cloud components. Hence, the diagram illustrates a system architecture that combines edge computing capabilities with cloud infrastructure. In these levels, communication can take place via Restful calls and each individual module can use its own DB for storing the data required for proper functioning.

3.5 Cloud-to-Edge data streaming & intelligent object distribution, the best practice to deliver Zero-SWARM's solutions

The main concept of the Zero-SWARM project, which focuses on delivering a Cloud-to-Edge solution and distributing intelligent agents to support various functionalities such as prediction, predictive maintenance, math-optimizers, scheduling, and digital twins, requires a well-defined and systematic approach. To successfully implement this approach, several crucial steps need to be followed. The first step is to define the requirements of the Cloud-to-Edge architecture. It is essential to clearly identify the specific use cases that the application needs to support, such as prediction, predictive maintenance, math-optimization, scheduling, and digital twins. Additionally, understanding the data sources, processing requirements, and latency constraints of the edge computing environment is vital in this phase [60]. Once the requirements are defined, the next step is to select appropriate edge devices that align with the identified needs. Factors such as processing power, memory, connectivity, and power consumption should be considered when choosing edge servers, gateways, or IoT devices. These devices should be capable of running intelligent agents, math-optimization algorithms, and supporting the desired use cases.

After selecting the edge devices, it is crucial to choose an appropriate edge computing framework that

provides the necessary capabilities for deploying, managing, and orchestrating intelligent agents at the edge. Evaluating popular frameworks like Apache Edgent, Eclipse ioFog, and TensorFlow Lite for Edge based on features, performance, and ease of use can help in making an informed decision. The development of intelligent agents, math-optimization algorithms, and digital twins is the next step in the process. These components should be designed using suitable machine learning or statistical techniques for prediction, predictive maintenance, and scheduling [61]. Optimizing the agents for edge computing environments in terms of size, complexity, and processing requirements is essential. Compatibility with the selected edge framework is also a crucial consideration. Once the intelligent agents are developed, they need to be deployed to the edge devices using the chosen edge computing framework. Secure deployment practices, including encryption, authentication, and access controls, should be followed to ensure data and computation confidentiality and integrity. In cases where models are trained in the cloud, distributed deployment to the edge may be necessary.

To ensure proper functioning of the deployed agents, monitoring and management capabilities should be implemented. Real-time monitoring of performance, health, and status is crucial. Mechanisms for remote configuration, updates, and maintenance should also be in place. Integration of the edge agents with the cloud is an important aspect of the Cloud-to-Edge architecture. This integration enables centralized management, coordination, and data analysis. Data synchronization, aggregation, and analytics mechanisms should be implemented to leverage the collected data for higher-level processing, decision making, and system optimization in the cloud. Testing and validation are necessary to ensure the performance, accuracy, and reliability of the deployed intelligent agents, math-optimization algorithms, and digital twins in the edge computing environment [62]. Different testing methodologies, including simulation, emulation, and real-world testing, should be used to verify that the agents meet the defined requirements. The Cloud-to-Edge architecture should be continuously improved through iterative iterations based on feedback from testing, monitoring, and performance analysis. Real-world data and performance metrics should drive the refinement of intelligent agents, math-optimization algorithms, and digital twins. Staying up-to-date with edge computing technologies and best practices is crucial for optimization and enhancement. Security and privacy considerations are of utmost importance throughout the Cloud-to-Edge architecture. Robust measures such as encryption, authentication, access controls, and other security mechanisms should be implemented to protect data and computations at all stages of the architecture. Also, scalability and resilience are essential aspects to consider. The architecture should be designed to handle varying workloads, changing edge environments, and potential failures. Load balancing, fault tolerance, and redundancy mechanisms should be in place to ensure high availability and reliability of the intelligent agents and math-optimization algorithms.

Documentation and knowledge sharing are valuable practices for future reference and knowledge transfer. Properly documenting the architecture, design decisions, and implementation details ensures that the knowledge is preserved and can be utilized by others. Implementing cloud-to-edge data streaming and intelligent object distribution in a fleet management system, such as one that manages a fleet of Automated Guided Vehicles (AMRs), offers significant benefits. By adopting these approaches, latency can be minimized, bandwidth usage can be optimized, scalability can be enhanced, and reliability can be improved [63]. Real-time decision-making, efficient data management, and effective task allocation within the AMR fleet can be achieved. Ultimately, leveraging cloud-to-edge technologies enables the optimization of system performance,

responsiveness, and overall management of AMR operations.

The zero-Swarm team can play a role of pioneering in exploring the combined technological layers described in the deliverable, to enable real-time decision-making by intelligent agents in cloud-to-edge architecture, also considering the novel aspects described in the next chapter and their motivations.

4 Beyond the state of the Art. Edge-to-Cloud Distributed Stream Computing Using IEC 61499 and OPC-UA over Kafka and MQTT

4.1 How the zero-Swarm Cloud-to-Edge Distributed Streaming platform enable novel applications

In the previous chapter, we have seen how, in zero-Swarm, technological layers could be combined to create a harmonized infrastructure enabling, in the cloud-to-edge continuum, distribution of intelligent agents that solicit data streaming, for example for learning processes and to allow accurate and real and near-real-time decision making without forgetting the CPSoS paradigm.

Now, we explore how this architecture enables novel applications whose characteristics depend on the particular case.

In the zero-swarm "Distributed stream computing" platform, engineers could/should consider implementing the following element to set new benchmarks for distributed intelligent control applications.

Multi-Layered Adaptivity (MLA): MLA introduces a novel algorithmic approach that allows for adaptivity at both cloud and edge levels, optimizing data flow and computation tasks in real-time. This is a departure from static task allocations in conventional systems.

The Multi-Layered Adaptivity concept envisions a dynamic, self-optimizing architecture where computational tasks and data streams can be adaptively managed at both cloud and edge levels. This flexibility contrasts sharply with traditional systems where tasks are statically allocated, leading to inefficiencies in resource utilization.

The key ingredients to apply MLA are:

- Adaptive Task Scheduler: for example, a machine learning-based scheduler evaluates current loads, bandwidth, and processing capabilities at both edge and cloud levels. Depending on the requirements and available resources, it dynamically reallocates tasks.
- Data Prioritization Engine: Intelligent algorithms prioritize which data should be processed immediately at the edge and which can be deferred for cloud-based analytics. For example, data related to immediate failure will be prioritized over routine operational statistics.
- Load Balancer: Continuously monitors resource utilization and shifts loads between edge and cloud in real-time, allowing for seamless scalability.
- As a practical example, we can imagine a manufacturing plant with a complex array of sensors and machines engaged in various tasks (cutting, welding, assembly, etc). The Multi-Layered Adaptivity concept can be applied, for example, as follows:

- Real-time Monitoring: Sensors continuously monitor various parameters such as temperature, pressure, and material flow. Cameras, in the process, can also be a source of data.
- Edge-Level Decisions: When an overheating issue is detected in a welding machine, the edge-level algorithms immediately halt the operation and notify the control room, thus averting potential equipment failure, process/product defects, or any hazard.
- Cloud-Level Analysis: The cloud-level system analyzes the aggregate data to identify patterns or trends. It may recognize that issues occur more frequently at certain times, signaling the need for preventive maintenance schedules.
- Adaptive Task Re-Allocation: During peak hours, when edge devices are overloaded, the scheduler may reroute some less critical data analysis tasks to the cloud, freeing up edge resources for real-time monitoring and control.

By incorporating Multi-Layered Adaptivity in the zero-swarm proposed framework, the manufacturing plant gains real-time responsiveness and benefits from broader trends and efficiencies discerned at the cloud level.

Dynamic Agent Allocation: Machine learning algorithms can be included in the platform to dynamically allocate intelligent agents in response to system requirements, improving overall system efficiency.

Edge-Centric Analytics and Federated Machine Learning: Developing novel edge analytics algorithms that can make local decisions without necessarily requiring cloud communication, thus reducing latency and improving system responsiveness. For example, levering machine learning and federated ML to enable edge devices to train models locally, thus decreasing data transfer volumes and improving real-time capabilities (see D4.3)

CPSoS-Ready Modules combined with Metamodeling for Standardization and Performances: While introducing plug-and-play modules specifically designed for automatic discovery allows for seamless integration and Standardization (see D5.2), metamodeling entails creating abstract, higher-level models that encapsulate the essential elements and relationships. This metamodel serves as a template/representation serving the engineering &/or development process and the operations. In delivery 4.3, the concept of metamodels is applied to the analytical agents.

Data Lineage and Provenance: Implementing traceability features into the data streams enables every piece of data to be audited and validated for its entire lifecycle.

The novel aspects of the zero-Swarm platforms provide a holistic, adaptable, and future-proof solution that caters to the complexity of modern manufacturing processes and the need for real-time analytics and control.

It offers compelling advantages for vendors looking to provide a cutting-edge, integrated solution and for manufacturers aiming to optimize their operations and gain deeper insights into their processes.

About the Engineering /Solution Provider, we can mention:

Comprehensive Solution: Leveraging Kafka, MQTT, and OPC-UA provides a unified platform for data streaming, control, and interoperability, reducing the complexity for end-users and increasing the platform's appeal.

Competitive Edge: a unique, high-performance solution. Features like Multi-Layered Adaptivity and Metamodeling can distinguish the solution, providing unique selling points.

Scalability: The platforms facilitate easy scaling, allowing you to support small and large manufacturing processes.

Interoperability: Standardization ensures the solution can easily integrate with other systems, reducing friction during implementation.

Resource Optimization: Dynamic task allocation and data prioritization algorithms can significantly reduce computational and bandwidth costs, improving overall ROI.

Operational Efficiency: Advanced algorithms for real-time and near-real-time decision-making functionalities enable efficient use of resources, again strengthening the platform's ROI proposition.

Future-Proofing: The architecture is designed to adapt to emerging technologies, thereby ensuring the platform remains current and minimizes obsolescence risks.

Customizability and Scale: The applied paradigm of CPSoS and the standard IEC 61499 provide a standardization framework enabling customization and scalability, making the platform suitable for various sizes, types, and dimensions of manufacturing enterprises.

From the end users (Manufacturers) point of view, the zero-swarm platform and paradigm provide some clear advantages even if the single benefit's importance depends on the type of process, level of automation, and dimension. We can mention:

Operational Efficiency: Cloud-to-edge continuum ensured, and real-time decision-making also enabled at the edge level, minimizing downtimes, near-real-time process and product quality control, and optimizing machines and robots utilization.

Cost Savings: The ability to dynamically allocate tasks between Edge and Cloud optimizes resource usage, reducing operational costs.

Standardization is a standardized approach to system engineering, significantly reducing the complexity and cost of integrating disparate systems.

Data-Driven Insights: Advanced analytics capabilities, both at the Edge and in the cloud, enable data-driven decision-making for preventive maintenance, quality control, and process optimization.

Risk Mitigation: Features like real-time monitoring and dynamic reallocation of tasks can significantly reduce operational risks such as machine failures or security vulnerabilities.

Optimized Computing Resource Allocation: Multi-layered adaptivity ensures that resources are used where they are most needed, optimizing both edge and cloud computing capabilities.

Real-time Data-Driven Intelligence: The platform provides real-time analytics and decision-making, significantly improving manufacturing processes and mitigating risks. Moreover, the intelligent agents and digital twin, directly connected to the platform as components, and the distributed computing capabilities enable deep insights into manufacturing processes, supporting prediction (eg, predictive maintenance), process control and optimization as well as quality assurance.

Streamlined Integration: Thanks to the use of IEC 61499 and OPC-UA over Kafka and MQTT, the platform ensures seamless interoperability with existing manufacturing systems, reducing integration costs and complexities.

5G Connectivity: The inclusion of 5G technology ensures low-latency, high-reliability communications,

enabling more effective real-time control and data streaming across the manufacturing floor.

4.2 OPC-UA to Kafka

There are different ways to integrate OPC-UA data into an Apache Kafka cluster.

Kafka comes natively with five core APIs: Admin, Producer, Consumer, Kafka Streams and Kafka Connect API. Kafka Connect API is a tool to build and run reusable data import/export connectors that consume or produce streams of events from and to external systems and applications so they can integrate with Kafka. The API is designed for scalability and fault-tolerantly ingesting big data collections from heterogeneous sources into Kafka topics with low latency. Kafka Connect can run in a distributed execution mode, where in handles load balancing and fall outs, provided that your connector extends the Connector API. It's a pluggable framework, enabling developers to implement their own connectors providing an API in many programming languages. There is, however, a strong community already providing hundreds of reusable connectors, for instance Apache PLC4X. PLC4X is a set of open-source libraries for communicating with industrial grade PLCs using industrial protocols in a uniform way. PLC4X Kafka connectors enable communication among industrial protocols (including OPC-UA) and Kafka.

Beyond the connectors, there are a variety of architecture components available in Kafka Ecosystem: Broker, KSQL (a streaming SQL engine for Kafka), ZooKeeper (high-level service to maintain and manage the Kafka cluster), Kafka Connect Mirror (to copy topics from one cluster to another).

Assuming we have edge-nodes with OPC-UA servers running in the shopfloor aggregating data from different sensors/actuators, we are evaluating three integration options for streaming data from/to Kafka which are shown in Figure 9, all using the current available PLC4X OPC-UA Kafka connector:

1. Stream OPC-UA data from the server directly to a Kafka cluster in the cloud through a Kafka Connect OPC-UA connector. Data is streamed from the edge to the cloud as OPC-UA traffic.
2. Use the Kafka Connect OPC-UA connector in the edge and stream data from edge to cloud as to Kafka traffic.
3. Deploy a Kafka Broker in the edge and use a Kafka Connect Mirror in the cloud to replicate the information. Data is streamed from edge to cloud a Kafka traffic. This option provides data persistence and stream processing capabilities in the edge.

Most extended way to stream data from edge to cloud is to use MQTT as lightweight communication protocol. The novel OPC-UA Pub/Sub communication mechanism could offer similar performance than MQTT, however, a compatible Kafka Connector supporting this mechanism should be used. The selection among option 1 and 2 is not clear in terms of performance and would depend on the application. The main difference between option 1, 2 and option 3 is the use of the Kafka Broker in the edge, which provides data persistence, and can be beneficial in many applications with unstable edge-cloud communications.

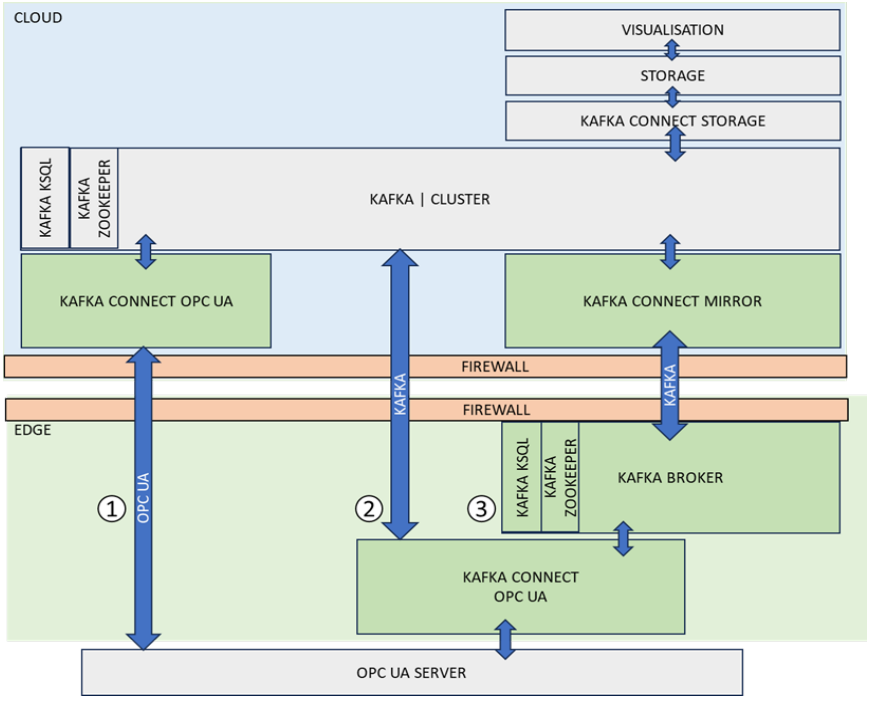

Figure 9: OPC-UA to Kafka architecture options

4.3 MQTT and Kafka

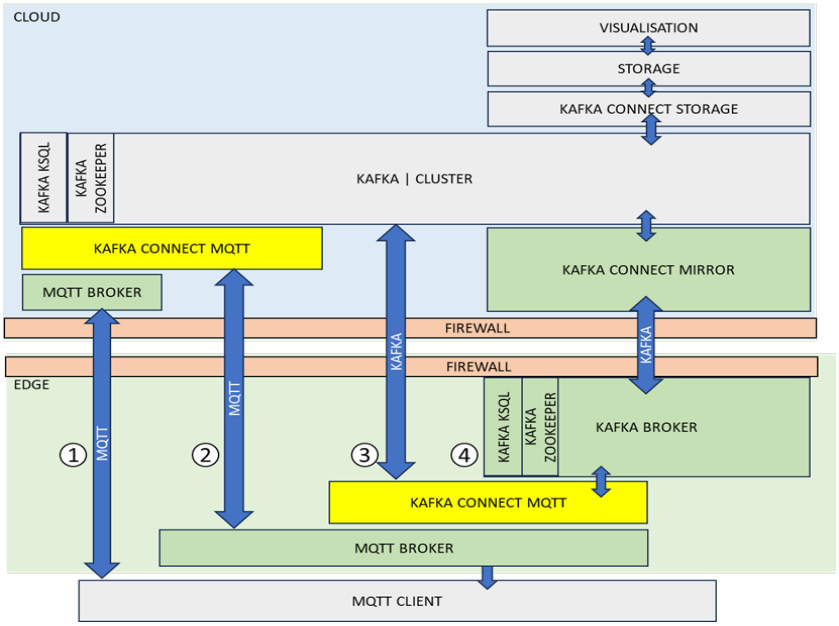

Figure 10: MQTT to Kafka architecture options

Similarly, to the bridging options for integrating OPC-UA and Kafka already discussed in Section XX, here we evaluate different approaches for integrating MQTT with Kafka (Figure 10). An equivalent to the Kafka Connect OPC-UA connector is the Kafka Connect MQTT connector. The connector implements an MQTT client to read/write data from an MQTT Broker and translate it to Kafka. There are, however, very few open-source implementations available of these types of connectors, and the ones available have limitations (e.g., constraints in the number of topics). Some work is foreseen in the

project to come up with a robust and stable MQTT connector. Alternatively, there are some proprietary solutions solving this integration by using MQTT brokers with integrated Kaka features. We don't consider the use of these proprietary solutions in Zero-Swarm. Assuming a similar scenario where we have edge-nodes with MQTT clients to publish/subscribe MQTT sensor data in a broker, we consider the following options based on open-source components:

1. Use an MQTT broker in the Cloud. The data is streamed from edge (MQTT client) to the cloud (MQTT broker) as MQTT traffic (TCP/IP).
2. Use the MQTT Broker in the edge. The data is streamed from the edge MQTT Broker to the Cloud (Kafka MQTT connector) as MQTT traffic.
3. Use the MQTT Kafka connector on the edge. Data is streamed from edge to cloud as Kafka traffic.
4. Deploy a Kafka Broker on the edge, and the Kafka Connect Mirror in the cloud. Data is streamed as Kafka traffic from edge to cloud. This solution offers data persistence and stream processing capabilities in the edge.

The best solution will always depend on the application requirements. Most common approaches adopted in IoT applications with distributed low power smart sensor is option 1. However, in industrial applications usually including an edge computing layer, option 4 could be preferable thanks to the data persistence feature. In all cases, the main technology gap is the development of a robust Kafka Connect MQTT connector.

4.4 IEC 61499 over OPC-UA to Kafka

4.4.1 OPC-UA in the IEC 61499 environment

The focus here is the integration of IEC 61499 with OPCUA.
This objective was achieved by creating Schneider Electric Ecostruxure Automation Expert (EAE) SFBs. These Function Blocks are dedicated to data transmission via the OPC UA communication protocol. These FBs use the OPC UA information model to exchange data between EAE applications and between EAE applications and other industrial applications.
EAE allows to create an industrial control application based on the IEC 61499 standard, and additionally allows the configuration of OPC UA tags for the usage as OPC UA server. Besides this, OPC UA Client Function Blocks for the IEC61499 application acting as OPC UA Clients, able to connect to an OPC UA server in front of Kafka. This flexibility is necessary to connect IEC61499 over OPC UA to Kafka, which is a new approach for IEC61499 application, connecting to the datacentre layer, and is a focus within this deliverable and Zero-SWARM project.
OPC UA defines a comprehensive information model for publishing and managing meta-information, and a system context to simplify automation engineering and systems integration. As implemented by the EcoRT **OPC UA server**, the OPC UA information model maps to the data structure of the IEC61499 application. In this way, the server can present OPC UA information in an object oriented, re-usable way that is consistent with the application design. OPC UA data is device-based, which means that each device presents its own **OPC UA server** and data model.

4.4.2 OPC UA Server

To communicate with third-party equipment, the IEC-61499 application can behave as an OPC-UA

server on the one hand or as an OPC-UA client on the other.

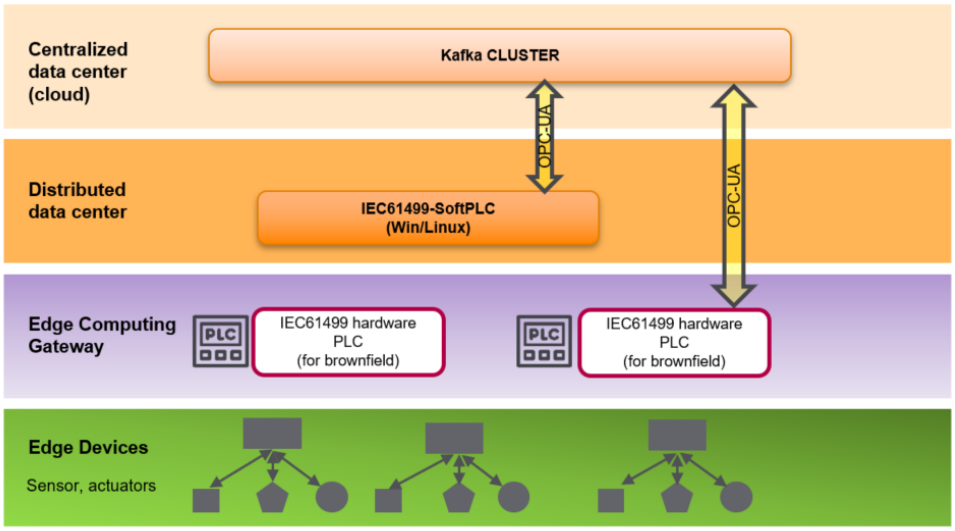

Figure 11: OPC UA to Kafka connection

As an OPC-UA server EAE uses a Composite Automation Type called CAT. From this CAT the variables are exposed in such a way that these are shared with the third party.

4.4.3 OPC UA Client

OPC UA client functionality for EAE is implemented by means of OPC UA client service function blocks, which are incorporated in the Service folder of the Standard.OPCUAClient library. Refer to the topic OPC UA Client.
The OPC UA client function blocks include:

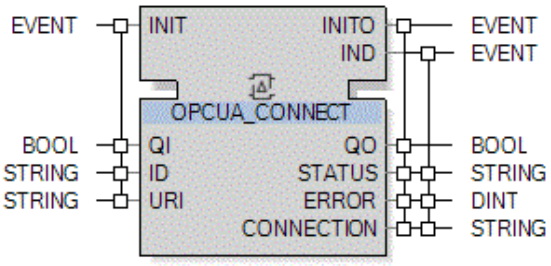

Figure 12: SIFB OPCUA CONNECT

OPCUA_CONNECT: Establishes a connection to an OPC UA server.

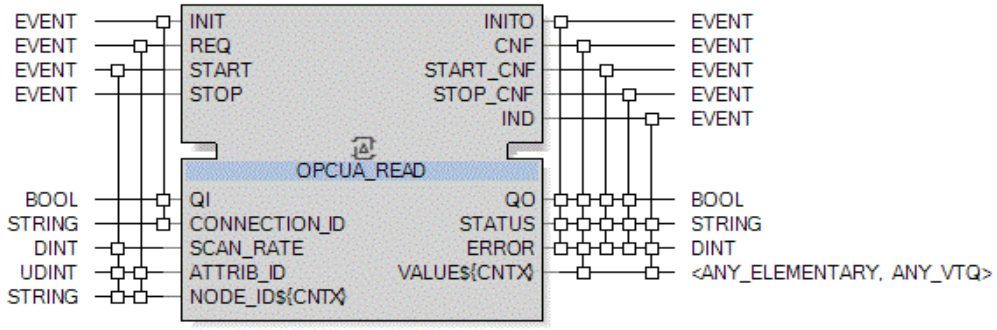

Figure 13: SIFB OPCUA READ

OPCUA_READ: Reads a Value attribute of up to 32 variables. Reading mode can be cyclic or acyclic. In cyclic mode, this function block can poll the Value attribute of a node, and can read any attribute value by specifying the attribute ID for the attrib_id input. Supports all elementary data types and Value, Time, Quality (VTQ) types in IEC61499, but structures and arrays are not supported.

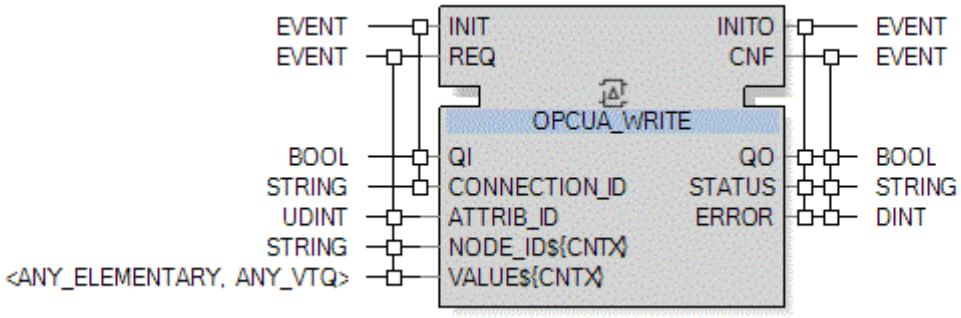

Figure 14: SIFB OPCUA WRITE

•**OPCUA_WRITE**: Updates Value attribute of up to 32 variables. Supports the elementary data types in IEC61499. Accepts VTQ input, provided the OPC UA server also supports VTQ input.

• **OPCUA_CALL:** Invokes any method in the address space. Up to 32 input arguments of any elementary data types are supported. Up to 32 output arguments of any elementary types are supported.

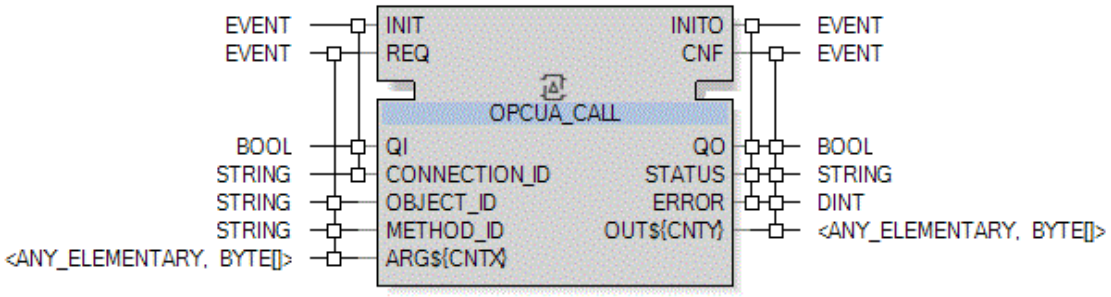

Figure 15: SIFB OPCUA CALL

• **OPCUA_MONITOR_EVENT**: This function block:

- Monitors alarms and events.
- Supports OPC UA data access connections.
- Supports method calls.

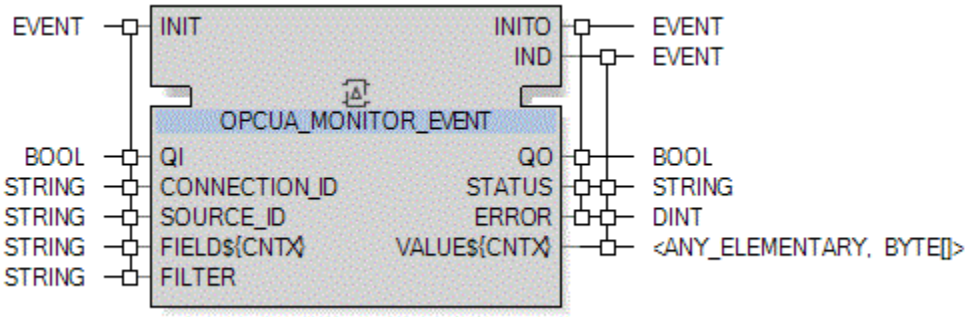

Figure 16: SIFB OPCUA MONITOR EVENT

IN IEC 61499 engineering environment (EAE), a Composite Automation Type CAT is created for each

asset present in edge devices layer shown in Fig.1. CAT can represent sensors, machines, or actuators with OPC UA functionality. Once a CAT is created with the function, it can be reused in an application created in the Ecostructure Automation Expert. For topics and payload generation CAT in CAT structure is used. This way it saves time for an automation engineer to create a solution. After the application is created, the data gathered by the edge devices can be transferred to Kafka cluster in cloud via OPC UA.

4.5 EC 61499 over MQTT to Kafka

4.5.1 MQTT in the IEC 61499 environment

The IEC 61499 runtime system implements the Mongoose library. This is a networking library for C/C++, which implements APIs for several protocols, among others also MQTT.
This library implements the MQTT protocol with all its functionalities as SIFBs (Service Interface Function Blocks). A SIFB (Service Interface Function Block) provides an interface for a service that is programmed inside of the FB and can be used 'out of the box' when creating IEC 61499 applications. For the communication between the IEC 61499 platform and the IT level, such SIFBs will be used to exchange data via MQTT. There are several SFBs for MQTT communication:

- Publish: Sends a MQTT message to a connected MQTT broker
- Subscribe: Subscribes and receives a MQTT message from a connected broker.
- Connect: Establish a connection to the configured MQTT broker
- Broker: Usage of an internal MQTT broker running on the IEC61499 platform.

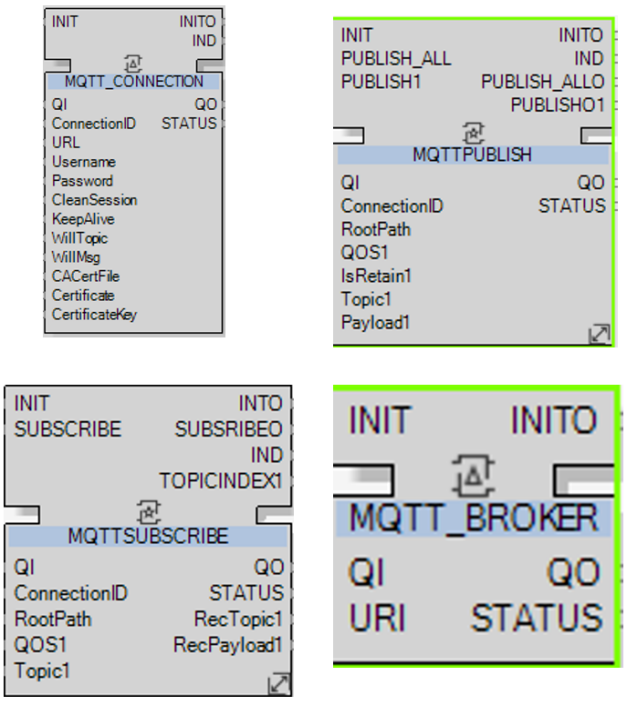

Figure 17: MQTT SIFB implementation

The SIFBs for MQTT are supporting the QoS functionality and in combination with other FBs that are

used for encrypting messages, they provide an additional security feature that is defined on the application level.

The physical objects of the real plant are designed in an object-oriented way with the IEC 61499-based extension for CAT in CAT nested hierarchy. This structure reflects the CAT in CAT hierarchy in the following way:

This structure reflects the CAT in CAT hierarchy in the following way:

- **MQTT topic 1** is a version of the protocol
- **MQTT topic 2** is the location name: Reepack
  - In our example it is the IEC 61499 application name representing the Reepack factory
- **MQTT topic 3** is the message type: Read/write
  - It can be Read or Write messages depending on data being published to or to be subscribed to
- **MQTT topic 4** is Plant: Production Line
  - In our example it is mapped to the IEC 61499 application which presents the Production line of Reepack
- **MQTT topic 5** is Machine/Object/Instance-name: ReeEco
  - The Pasteuriser machine is presented with an instance of a CAT machine object
- **MQTT topic 6** is Object /Actor Name: Chain/Sealer/Film
  - Chain and Sealer are presented with an instance of a CAT object which is inside the CAT machine object creating the CAT in CAT structure
  - Under the topic name 6 the MQTT message payload transfers all the parts data from the dedicated objects
- **MQTT topic 7** is the payload
  - Payload is the data gathered from the HW present on the shop floor. The payload is the value sent under a specific topic name to the MQTT broker

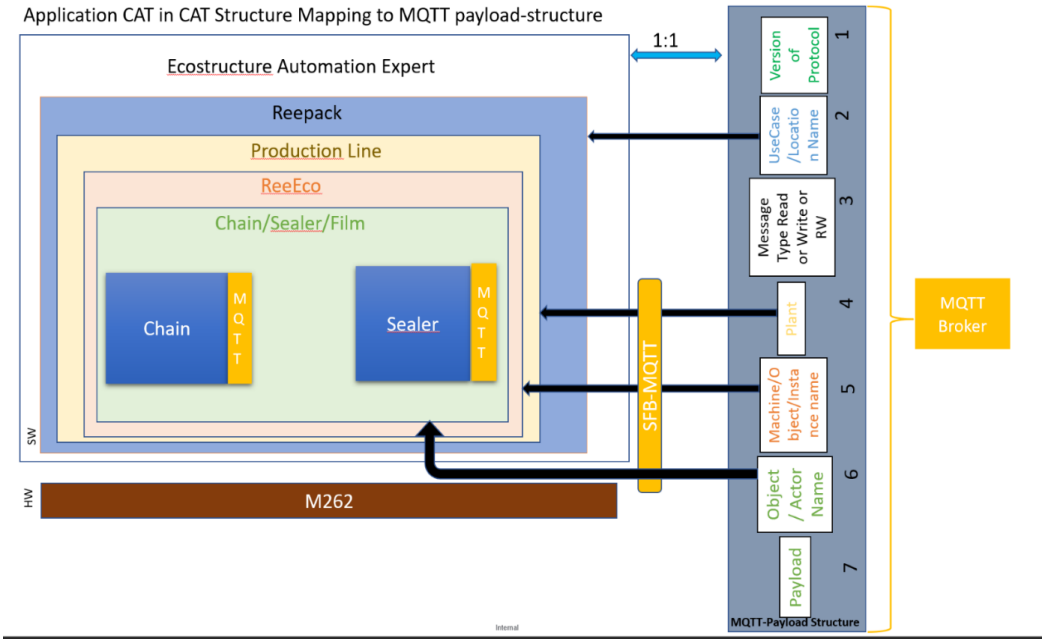

Figure 18: CAT in CAT mapping to MQTT Payload-structure in IEC61499

With the EcoStruxture Automation Expert, which is an engineering studio used to create IEC61499 applications using IEC61499 Function blocks, a concept is developed to simplify the mapping between the IEC61499 Composite Automation Type (CAT) and the MQTT communication, where the automation engineer needs nothing to know about the MQTT structure.
Figure 18 shows the CAT in CAT structure mapping of the application created in the IEC 61499 platform. The conceptual view gives an idea of the IEC6**1499** application and its connection to the MQTT topic's. "Repack" is the name of the IEC61499 application using the IEC61499 platform. The application consists of a CAT created for a Production Line, containing objects in which all the machines of a production line are present. One of the machines is ReeEco represented as a CAT in above Fig 1, which contains parts of machines like "Chain", "Sealer" or "Film". For the MQTT Payload a JSON SFB in the IEC61499 environment provides the possibility to form a message in JSON format and send/receive that message to the MQTT broker using the MQTT publish/subscribe SFBs, where the appropriate Data Inputs and Data Outputs of a dedicated CAT-Object are included to transport them over one message.

4.5.2 Connecting the IEC61499 automation platform over MQTT with the Kafka Environment

Apache Kafka is a popular open-source streaming platform that makes it easy to share data between systems and applications. In T4.2 the goal is to create a communication between Kafka and the IEC61499 platform over MQTT communication protocol, because this MQTT protocol is already available in the IEC61499 platform and is going to be extended with the above mapping functionality on the CAT level. The final structure of the payload with the MQTT topic structure will be adapted to the needs of the Kafka MQTT structure and topics.
There are different architectural ways to implement the bridge between IEC61499 over MQTT to Kafka.

1. **Kafka Connect for MQTT**: Kafka Connect is a Framework extension by Apache Kafka which can connect external systems to Kafka for message exchange. The Kafka connector acts as an MQTT Client that can subscribe to various topics to collect MQTT messages from the MQTT broker and write them to Kafka Cluster as a native Kafka client
2. **MQTT Proxy**: MQTT proxy offers an approach where MQTT messages are streamed into the Kafka cluster without an MQTT broker. This approach is easily scalable since the MQTT proxy application is stateless. In IEC 61499 platform we can adapt by having a Kafka cluster on the external platform and directly streaming the messages from the device to the Kafka cluster.
3. **MQTT Custom Bridge**: Alternative to the two options above, it's possible to develop an application. The application can take over the transport of data from the IEC61499 node to Kafka and forward data from the Kafka cluster to IEC 61499 node.
4. **MQTT Broker Extension:** Another possibility is to provide the MQTT broker the option of writing messages directly to Kafka with an extension.

Considering the task partners of the Apache Kafka platform, it is decided to integrate existing open-source solutions, which are available by option 1.

4.6 IEC61499 to cloud over HTTP

We developed a data-streaming method for the cloud storage and visualization services in IEC 61499 applications. The approach is based on LEAP (La Trobe Energy Analytics Platform). LEAP is a cloud platform oriented toward analysts and monitoring of energy data. The method was applied to a

building management system (BMS). BMS is a testbed for the simulation of a smart house systems behavior, managed with IEC 61499 distributed application. BMS wall is equipped with illumination and proximity sensors, curtains, lamps and smart electric meters.

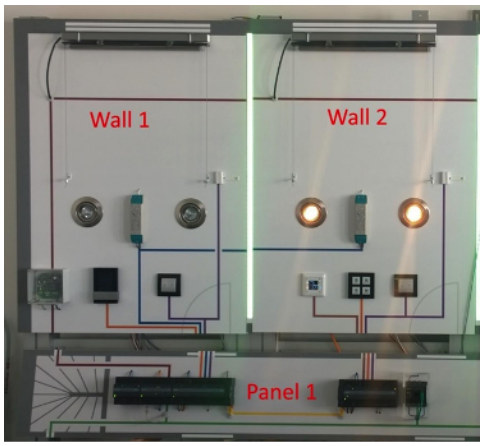

Figure 19: BMS wall

The structure of the controlled network is shown on Figure 19 IEC 61499 application deployed into four control devices controls BMS wall. The gateway device re-transmits data received from an IEC61499-compliant device to the Cloud in HTTP format. The visualization device is a computer. It visualizes the data of interest.

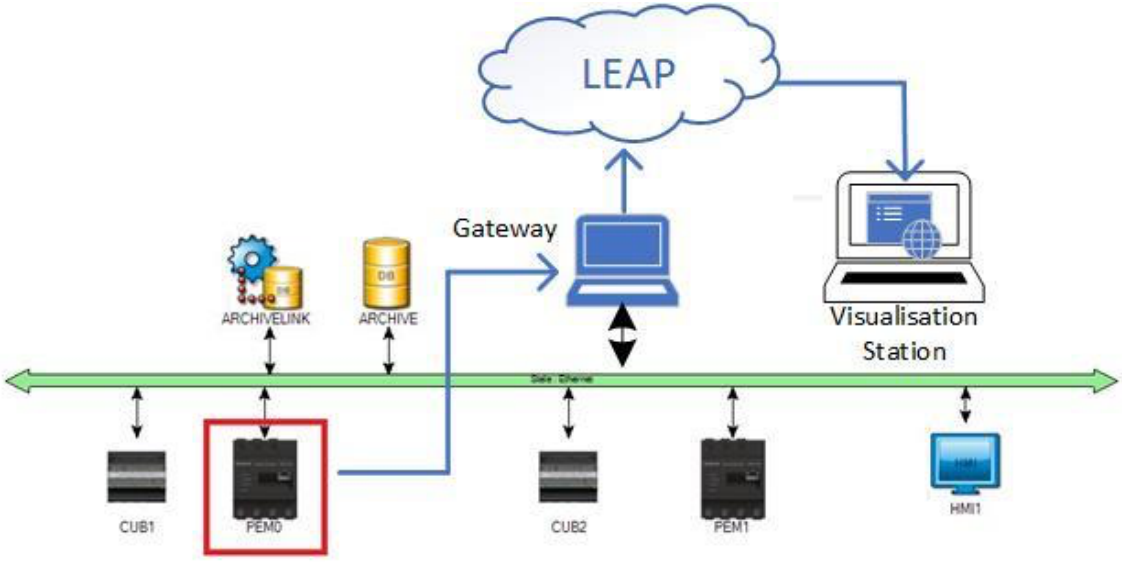

Figure 20: controlled network structure

Let's take a look at the BMS wall control application. Suppose the active power measured by the smart meter is of interest to be observed through an external web-based visualization application. The application is extended with NETIO function block. NETIO communicates the data assigned to its parameter SD over the network. The parameter ENDPOINT is address of the receiving party. ANY2ANY block converts data of interest and assigns in to the input SD of the NETIO. The gateway receives the measurement data in STRING format using a UDP socket and sends messages to the LEAP. The gateway receives the measurement data in STRING format using a UDP socket and sends messages to the LEAP

in the format. LEAP aggregates disparate data sources through its data lake and lata the data can be processed by the analytic engine.

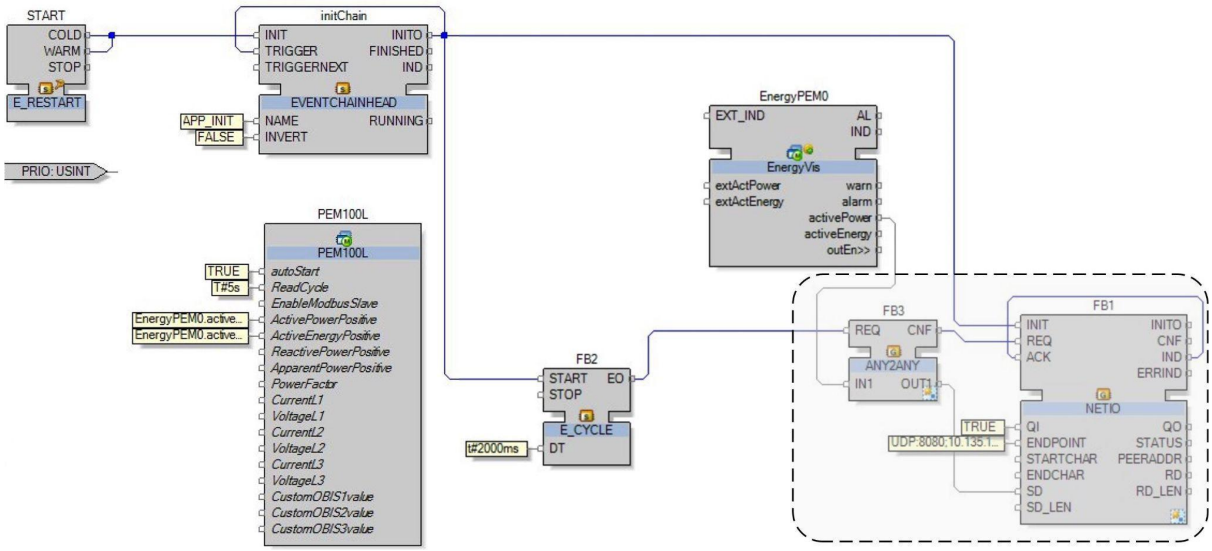

Figure 21: BMS wall control application

5 Conclusions

Intelligent agents deployed within the Edge-Cloud continuum offer a strategic advantage such as:

- Reduced data transfer latency.
- Distributing intelligence across the Edge-Cloud continuum optimizes network bandwidth usage. Data preprocessing and data analysis can be performed at the Edge, reducing the amount of data transmitted to the Cloud;
- Privacy and Security Enhancement (critical information stays within the premises, minimizing exposure to risks)
- Effective distribution of intelligent agents optimizing computational capacity.

We have seen that challenges are also present, such as:

- Algorithm Selection and Deployment;
- Enabling asynchronous learning involves continuous model updates in response to changing data streams, operational conditions, and evolving patterns.
- Coordinating data synchronization and maintaining consistency across distributed intelligent agents can be complex.
- Seamless communication between Edge devices and the Cloud is vital for timely data exchange and decision-making.

Achieving the full potential of this paradigm necessitates meticulous balancing of these aspects. The appropriate technology layers and a robust architecture represent a core enabler for performance standardization of the development and delivery of applications. In this direction, we have seen how Kafka, MQTT, OPC-UA, and the IEC 61499 environment provide a possible robust and standard framework.

In conclusion, the Zero-SWARM platform serves as a disruptive innovation in the manufacturing sector. It presents an opportunity to deliver a competitive, scalable, and future-proof product that integrates

seamlessly across systems. It offers a pathway to operational excellence by enabling real-time decision-making, reducing costs, and mitigating risks. It favors design and development standardization.

Advanced features like Multi-Layered Adaptivity, the possibility to distribute intelligent agents and learning processes, and the metamodeling concepts introduced in D4.3 contribute to the platform's uniqueness and relevance, solidifying its potential as a transformative force in modern manufacturing.

References

[1] Wang, N., Varghese, B., Matthaiou, M., & Nikolopoulos, D. S. (2017). ENORM: A framework for edge node resource management. IEEE transactions on services computing, 13(6), 1086-1099.

[2] Varghese, B., Wang, N., Li, J., & Nikolopoulos, D. S. (2017). Edge-as-a-service: Towards distributed cloud architectures. arXiv preprint arXiv:1710.10090.

[3] Chang, H., Hari, A., Mukherjee, S., & Lakshman, T. V. (2014, April). Bringing the cloud to the edge. In 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 346-351). IEEE.

[4] Sunyaev, A., & Sunyaev, A. (2020). Cloud computing. Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies, 195-236.

[5] Cao, K., Liu, Y., Meng, G., & Sun, Q. (2020). An overview on edge computing research. IEEE access, 8, 85714-85728.

[6] Hassan, N., Yau, K. L. A., & Wu, C. (2019). Edge computing in 5G: A review. IEEE Access, 7, 127276-127289.

[7] Pan, J., & McElhannon, J. (2017). Future edge cloud and edge computing for internet of things applications. IEEE Internet of Things Journal, 5(1), 439-449.

[8] Asim, M., Wang, Y., Wang, K., & Huang, P. Q. (2020). A review on computational intelligence techniques in cloud and edge computing. IEEE Transactions on Emerging Topics in Computational Intelligence, 4(6), 742-763.

[9] [Rodrigues, T. K., Suto, K., Nishiyama, H., Liu, J., & Kato, N. (2019). Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective. IEEE Communications Surveys & Tutorials, 22(1), 38-67.

[10] Duc, T. L., Leiva, R. G., Casari, P., & Östberg, P. O. (2019). Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. ACM Computing Surveys (CSUR), 52(5), 1-39.

[11] Stankovski, S., Ostojić, G., Šaponjić, M., Stanojević, M., & Babić, M. (2020, March). Using micro/mini PLC/PAC in the Edge Computing Architecture. In 2020 19th International Symposium Infoteh-Jahorina (Infoteh) (pp. 1-4). IEEE.

[12] Jazdi, N. (2014, May). Cyber physical systems in the context of Industry 4.0. In 2014 IEEE international conference on automation, quality and testing, robotics (pp. 1-4). IEEE.

[13] de C Henshaw, M. J. (2016). Systems of systems, cyber-physical systems, the internet-of-things… whatever next?. Insight, 19(3), 51-54.

[14] Pahl, C., & Lee, B. (2015, August). Containers and clusters for edge cloud architectures--a technology review. In 2015 3rd international conference on future internet of things and cloud(pp. 379-386). IEEE.

[15] Lee, E. A., Hartmann, B., Kubiatowicz, J., Rosing, T. S., Wawrzynek, J., Wessel, D., ... & Rowe, A. (2014). The swarm at the edge of the cloud. IEEE Design & Test, 31(3), 8-20.

[16] Pan, J., & McElhannon, J. (2017). Future edge cloud and edge computing for internet of things applications. IEEE Internet of Things Journal, 5(1), 439-449.

[17] Hassan, N., Yau, K. L. A., & Wu, C. (2019). Edge computing in 5G: A review. IEEE Access, 7, 127276-127289.

[18] https://aws.amazon.com/greengrass/?nc1=h_ls
[19] https://azure.microsoft.com/en-gb/products/iot-hub/
[20] https://cloud.google.com/edge-tpu
[21] https://incubator.apache.org/projects/edgent.html
[22]https://www.edgexfoundry.org/
[23]https://iofog.org/
[24]https://www.tensorflow.org/lite
[25]https://onnxruntime.ai/
[26]https://developer.nvidia.com/tensorrt
[27]Caprolu, M., Di Pietro, R., Lombardi, F., & Raponi, S. (2019, July). Edge computing perspectives: architectures, technologies, and open security issues. In 2019 IEEE International Conference on Edge Computing (EDGE) (pp. 116-123). IEEE.
[28]Leitner, S. H., & Mahnke, W. (2006). OPC UA–service-oriented architecture for industrial applications. ABB Corporate Research Center, 48(61-66), 22.
[29]Schwarz, M. H., & Börcsök, J. (2013, October). A survey on OPC and OPC-UA: About the standard, developments and investigations. In 2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT) (pp. 1-6). IEEE.
[30] Cavalieri, S., & Chiacchio, F. (2013). Analysis of OPC UA performances. Computer Standards & Interfaces, 36(1), 165-177.
[31]Estuary Flow. (n.d.). Estuary. Retrieved June 14, 2023, from https://estuary.dev/product/
[32]Hillar, G. C. (2017). MQTT Essentials - A Lightweight IoT Protocol. Packt Publishing.
[33]Kafka. (n.d.). Apache Kafka ([31], n.d.). Retrieved June 14, 2023, from https://kafka.apache.org/
[34]Pulsar. (n.d.). Apache Pulsar | Apache Pulsar. Retrieved June 14, 2023, from https://pulsar.apache.org/
[35]RedPanda. (n.d.). Redpanda | The streaming data platform for developers. Retrieved June 14, 2023, from https://redpanda.com/
[36]StreamSets. (n.d.). StreamSets: Data Integration Platform for Enterprise Companies. Retrieved June 14, 2023, from https://streamsets.com/
[37]Dunning T, Friedman E. Streaming architecture: new designs using Apache Kafka and MapR streams. " O'Reilly Media, Inc."; 2016 May 10.
[38]Vohra D. Apache kafka. Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools. 2016:339-47.
[39]"Edge computing: Extending cloud computing to the edge of the network" by Shi, W. et al. (IEEE Computer, 2016)
[40]Aung T, Min HY, Maw AH. Coordinate checkpoint mechanism on real-time messaging system in kafka pipeline architecture. In2019 International Conference on Advanced Information Technologies (ICAIT) 2019 Nov 6 (pp. 37-42). IEEE.
[41]Torres DR, Martín C, Rubio B, Díaz M. An open source framework based on Kafka-ML for Distributed DNN inference over the Cloud-to-Things continuum. Journal of Systems Architecture. 2021 Sep 1;118:102214.
[42]Hisarligil BB. Franz Kafka in the Design Studio: A Hermeneutic-Phenomenological Approach to Architectural Design Education. International Journal of Art & Design Education. 2012

Oct;31(3):256-64.
[43]Wang Z, Dai W, Wang F, Deng H, Wei S, Zhang X, Liang B. Kafka and its using in high-throughput and reliable message distribution. In2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS) 2015 Nov 1 (pp. 117-120). IEEE.
[44]Le Noac'H P, Costan A, Bougé L. A performance evaluation of Apache Kafka in support of big data streaming applications. In2017 IEEE International Conference on Big Data (Big Data) 2017 Dec 11 (pp. 4803-4806). IEEE.
[45]Poss R, Lankamp M, Uddin MI, Sýkora J, Kafka L. Heterogeneous integration to simplify many-core architecture simulations. InProceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools 2012 Jan 23 (pp. 17-24).
[46]Gütlein M, Djanatliev A. Modeling and Simulation as a Service using Apache Kafka. InSIMULTECH 2020 (pp. 171-180).
[47]Miloradović M, Milovanović A. Data streaming architecture based on Apache Kafka and GitHub for tracking students' activity in higher education software development courses. InE-business technologies conference proceedings 2022 Jun 18 (Vol. 2, No. 1, pp. 132-136).
[48]Fennell C. A Communication Architecture for Transportation Digital Twins using Apache Kafka.
[49]Shree R, Choudhury T, Gupta SC, Kumar P. KAFKA: The modern platform for data management and analysis in big data domain. In2017 2nd international conference on telecommunication and networks (TEL-NET) 2017 Aug 10 (pp. 1-5). IEEE.
[50]Gütlein M, Djanatliev A. On-demand simulation of future mobility based on apache kafka. InInternational Conference on Simulation and Modeling Methodologies, Technologies and Applications 2020 Jul 8 (pp. 18-41). Cham: Springer International Publishing.
[51]Hugo Å, Morin B, Svantorp K. Bridging MQTT and Kafka to support C-ITS: A feasibility study. In2020 21stIEEE International Conference on Mobile Data Management (MDM) 2020 Jun 30 (pp. 371-376). IEEE.
[52]Indrasiri K, Indrasiri K. Enterprise Messaging with JMS, AMQP, MQTT, and Kafka. Beginning WSO2 ESB. 2016:133-60.
[53]Lan D, Liu Y, Taherkordi A, Eliassen F, Delbruel S, Lei L. A federated fog-cloud framework for data processing and orchestration: a case study in smart cities. InProceedings of the 36th annual ACM symposium on applied computing 2021 Mar 22 (pp. 729-736).
[54]Soderi M, Kamath V, Breslin JG. Toward an API-driven infinite cyber-screen for custom real-time display of big data streams. In2022 IEEE International Conference on Smart Computing (SMARTCOMP) 2022 Jun 20 (pp. 153-155). IEEE.
[55]Gruener S, Koziolek H, Rückert J. Towards resilient IoT messaging: an experience report analyzing MQTT brokers. In2021 IEEE 18th International Conference on Software Architecture (ICSA) 2021 Mar 22 (pp. 69-79). IEEE.
[56]Chonata Villamarín JF. End-to-End IoT System Integration for Real Time Apps using MQTT and KAFKA for collecting and streaming data from Fog to Cloud (Doctoral dissertation, ETSIS_Telecomunicacion).
[57]Soderi M, Gerard J. Bluetooth Low Energy Peripherals and Edge-to-Cloud Data Assessment as a Service, with Docker, Node-RED, MQTT, Scala, Spark, Kafka, HDFS, and Android SDK: a demo.
[58]de Oliveira DL, Veloso AF, Sobral JV, Rabêlo RA, Rodrigues JJ, Solic P. Performance evaluation of

mqtt brokers in the internet of things for smart cities. In2019 4th International Conference on Smart and Sustainable Technologies (SpliTech) 2019 Jun 18 (pp. 1-6). IEEE.

[59]Ge X, Zhu H, Wang C, Yuan Z, Zhu Y. Design and implementation of reactive distributed Internet of Things platform based on actor model. In2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) 2019 Mar 15 (pp. 1993-1996). IEEE.

[60]Liang WY, Yuan Y, Lin HJ. A performance study on the throughput and latency of zenoh, mqtt, kafka, and dds. arXiv preprint arXiv:2303.09419. 2023 Mar 16.

[61]Keswani B, Mohapatra AG, Keswani P, Khanna A, Gupta D, Rodrigues J. Improving weather dependent zone specific irrigation control scheme in IoT and big data enabled self driven precision agriculture mechanism. Enterprise Information Systems. 2020 Nov 25;14(9-10):1494-515.

[62]Renart EG, Balouek-Thomert D, Parashar M. Edge based data-driven pipelines (technical report). arXiv preprint arXiv:1808.01353. 2018 Aug 3.

[63]Vitorino JP, Simão J, Datia N, Pato M. IRONEDGE: Stream Processing Architecture for Edge Applications. Algorithms. 2023 Feb 17;16(2):123.

[64]Thanh LN, Phien NN, Vo HK, Luong HH, Anh TD, Tuan KN, Son HX. UIP2SOP: a unique IoT network applying single sign-on and message queue protocol. International Journal of Advanced Computer Science and Applications. 2021;12(6).

[65]Lv H. Highly Reliable Control System based on Reactive Event Stream. In2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) 2021 Jun 18 (Vol. 4, pp. 963-966). IEEE.

[66]Zero-SWARM “D4.4 Federated data infra & toolkit for data-driven model.v1”