

Project acronym:	Lasers4MaaS
Project title:	Laser-as-a-Service Digital Platform with Dynamic Beam Shaping for Acceleration of Smart, Decentralised and Sustainable Factory of the Future
Call	HORIZON-CL4-2024-TWIN-TRANSITION-01-03 Manufacturing-as-Service: technologies for customised, flexible, and decentralised production on demand
Grant agreement No:	101178719
Work-package 1:	Definition of project requirements
Deliverable D1.3:	Current standards and approaches for interoperability of data streams
Owner:	Futurice
Due date:	28 th February 2025

Type		
R	Document, report (excluding the periodic and final reports)	x
DEM	Demonstrator, pilot, prototype, plan designs	
DEC	Websites, patents filing, press & media actions, videos, etc	
DATA	Data sets, microdata, etc	
DMP	Data management plan	
ETHICS	Deliverables related to ethics issues	
SECURITY	Deliverable related to security issues	
OTHER	Software, technical diagram, algorithms, models, etc	

Dissemination level		
PU	Public, fully open, e.g. project website	x
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/EU-R	EU RESTRICTED under the Commission Decision No2015/444	

Disclaimers: this project has received funding from the European Union's HORIZON EUROPE research and innovation programme under grant agreement No. 101178719. The content of this publication is the sole responsibility of the Consortium partners listed herein and does not necessarily represent the view of the European Commission or its services.

HISTORY OF CHANGES

Version	Publication date	Change
1.0	07/02/2025	Document created
1.1	13/02/2025	First content draft
2.0	21/02/2025	Final version completed

LIST OF AUTHORS

Name	Organization	Email
Antti Partanen	FUTURICE	antti.partanen@futurice.com
Diarmaid de Búrca	FUTURICE	diarmaid.de.burca@futurice.com
Mary Jackson	FUTURICE	mary.jackson@futurice.com

TABLE OF CONTENT

- 1. Executive summary 6**
- 2. Introduction 7**
- 3. Definition of interoperability of data streams 8**
 - 3.1. Different Message Architectures 8**
 - 3.1.1. Polling 8**
 - 3.1.2. Message Queues..... 8**
 - 3.1.3. Pub/Sub..... 8**
 - 3.1.4. Request-Response..... 9**
 - 3.2. Cloud-Factory Communication Architectures 9**
 - 3.2.1. Database Synchronization 9**
 - 3.2.2. Edge Computing and Gateways 10**
 - 3.2.3. Blockchain Integration 10**
 - 3.2.4. Cloud-Native APIs & Services 10**
- 4. Data Format and Serialization Standards 12**
 - 4.1. JSON (JavaScript Object Notation)..... 12**
 - 4.2. XML (Extensible Markup Language)..... 12**
 - 4.3. Apache Avro 12**
 - 4.4. Protocol Buffers (Protobuf) 13**
- 5. Data Streaming and Messaging Protocol Standards..... 14**
 - 5.1. OPC UA (Open Platform Communications Unified Architecture) 14**
 - 5.2. MQTT (Message Queuing Telemetry Transport)..... 14**
 - 5.3. AMQP (Advanced Message Queuing Protocol) 14**
 - 5.4. Kafka (Apache Kafka) 15**
- 6. API and Communication Protocols..... 16**
 - 6.1. gRPC 16**
 - 6.2. REST API 16**
 - 6.3. GraphQL..... 16**
 - 6.4. WebSockets..... 17**
- 7. Security and Data Privacy Standards 18**
 - 7.1. TLS (Transport Layer Security, RFC 8446) 18**
 - 7.2. AES (Advanced Encryption Standard, ISO/IEC 18033-3)..... 18**
 - 7.3. Zero Trust Architecture (NIST SP 800-207) 18**
 - 7.4. OAuth 2.0 (For API & Microservices Security) 18**
 - 7.5. IEC 62443 (Industrial Cybersecurity Standard)..... 19**
- 8. Conclusions 20**
- 9. Sources 21**

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
FIFO	First-In, First-Out
RPC	Remote Procedure Calls
HTTPS	Hypertext Transfer Protocol Secure
IACS	Industrial Automation and Control Systems
IEC	International Electrotechnical Commission
IDL	Interface Definition Language
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
NIST	National Institute of Standards and Technology
OAuth	Open Authorization
OPC UA	Open Platform Communications Unified Architecture
Pub/Sub	Publish-Subscribe
QoS	Quality of Service
REST	Representational State Transfer
TLS	Transport Layer Security
URI	Uniform Resource Identifier
XML	Extensible Markup Language

1. Executive summary

This paper explores the critical aspects of interoperability for data streams within the context of the manufacturing. The paper examines various message architectures, including polling, message queues, publish-subscribe, and request-response, analyzing their strengths and weaknesses in industrial settings. Furthermore, it delves into cloud-factory communication architectures, covering database synchronization, edge computing, blockchain integration, and cloud-native APIs. The importance of data format and serialization standards is highlighted, with a discussion of JSON, XML, Apache Avro, and Protocol Buffers. The paper also investigates data streaming and messaging protocol standards such as OPC UA, MQTT, AMQP, and Kafka, as well as API and communication protocols like gRPC, REST APIs, GraphQL, and WebSockets. Finally, it addresses the crucial domain of security and data privacy, examining standards like TLS, AES, Zero Trust Architecture, OAuth 2.0, and IEC 62443. This comprehensive overview provides a foundation for informed decision-making regarding interoperability within the Lasers4MaaS project, ensuring seamless integration and secure data exchange across all components.

2. Introduction

The Lasers4MaaS program necessitates a robust and interoperable data ecosystem to achieve its objectives. This paper provides a comprehensive overview of current standards and approaches relevant to data stream interoperability. It aims to equip stakeholders with the knowledge necessary to select and implement appropriate technologies for seamless data exchange and communication within the Lasers4MaaS platform. The paper covers a range of topics, from fundamental message architectures to advanced security and privacy standards. By examining the strengths and weaknesses of various approaches, this document serves as a valuable resource for architects and developers working on the Lasers4MaaS project. The efficient and secure flow of data is paramount to the program's success, and this paper seeks to facilitate informed decisions that will ensure the platform's long-term viability and effectiveness.

3. Definition of interoperability of data streams

3.1. Different Message Architectures

Efficient and reliable communication between different components is crucial in industrial environments. The choice of message architecture plays a vital role in ensuring seamless data exchange and coordination between various elements. This section explores different message architectures and covers the strengths and weaknesses of each of them in an industrial context.

Four key message architectures are considered here: Polling, Publish-Subscribe (Pub/Sub), Message Queues, and Request-Response. Each architecture offers a unique approach to facilitating communication and data exchange between components within a system. By understanding the characteristics of these architectures, developers can make informed decisions about which approach best suits the specific communication needs of a project, ensuring efficient and reliable data flow between various components.

3.1.1. Polling

Polling is a message architecture that involves a receiver actively querying a sender for new data. This mechanism can be implemented in two primary ways:

- **Pull polling:** The receiver explicitly requests data from the sender at defined intervals. This is akin to checking your mailbox for letters regularly.
- **Push polling:** The sender informs the receiver that new data is available, prompting the receiver to initiate a retrieval request. This is like receiving a notification that you have new mail.

The effectiveness of polling hinges on factors such as the frequency of data updates, the volume of data transferred, and the desired responsiveness. Frequent polling can lead to unnecessary overhead if data updates are infrequent, while infrequent polling can result in delays in receiving crucial information.

In an industrial context, polling could be employed for tasks such as monitoring the status of devices or retrieving sensor data from manufacturing equipment. The choice between pull and push polling would depend on the specific requirements of the data stream, such as the urgency of data updates and the capacity of the communication channels.

3.1.2. Message Queues

Message queues provide a mechanism for asynchronous communication between applications or components. Senders place messages into a queue, where they are stored until a receiver retrieves them. This buffered approach enhances reliability by ensuring that messages are not lost even if the receiver is temporarily unavailable. Additionally, message queues promote decoupling, allowing senders and receivers to operate independently without affecting each other's performance. Different types of message queues cater to various needs:

- **FIFO (First-In, First-Out) queues:** Messages are processed in the order they are received, ensuring that the sequence of events is preserved.
- **Priority queues:** Messages are assigned priorities, allowing high-priority messages to be processed ahead of lower-priority ones.

In an industrial context, message queues could be employed for managing tasks between different devices, such as queuing up processing requests, storing intermediate results, or coordinating the activities of different components within the system. The choice of queue type would depend on the specific requirements of the data stream, such as the need for strict ordering or the prioritization of certain tasks.

3.1.3. Pub/Sub

Publish-subscribe (pub/sub) is an asynchronous messaging paradigm where senders (publishers) transmit messages to a central topic or channel without needing to know the identities of the receivers (subscribers). Subscribers express interest in specific topics and receive all messages routed through those topics. This decoupled approach fosters scalability and flexibility, as publishers and subscribers can operate independently without direct knowledge of each other.

The pub/sub model relies on message brokers to manage topics and subscriptions, ensuring efficient message delivery and handling potential issues like message duplication or loss. Message brokers act as intermediaries, receiving messages from publishers and distributing them to the appropriate subscribers. As all of the messages are passed through the broker, a publisher can have any number of subscribers and does not have any information about how many subscribers it has. Because of this, the number of subscribers and publishers can be scaled independently of each other, improving the efficiency of communication. Pub/sub can be thought of as an extension of a message queue, where each publisher has a 1 to many subscribers instead of a 1 to 1 relationship.

Within an industrial context, pub/sub could be utilized for disseminating real-time updates on production progress, broadcasting alerts about equipment malfunctions, or distributing processing tasks across a network of devices. The use of topics allows for selective message filtering, ensuring that subscribers only receive information relevant to their specific roles or tasks.

3.1.4. Request-Response

Request-response is a synchronous communication pattern where a sender transmits a request message and awaits a corresponding response message from the receiver. This two-way exchange is essential for scenarios where the sender requires confirmation or data from the receiver before proceeding.

The request-response model can be implemented using various messaging channels, such as point-to-point connections or publish-subscribe systems. However, the response channel is typically point-to-point to ensure that the response is delivered directly back to the original requester.

Message correlation is crucial in request-response interactions to ensure that responses are correctly matched to their corresponding requests. This involves including unique identifiers in both the request and response messages, allowing the sender to identify which response corresponds to which request.

Within an industrial context, request-response could be utilized for tasks such as querying the device configuration, initiating a specific task, or retrieving the results of a completed operation. The synchronous nature of request-response ensures that the sender receives immediate feedback on the status of the request.

3.2. Cloud-Factory Communication Architectures

In industrial context, the interaction between the cloud and the factory floor is paramount for achieving seamless data exchange, remote monitoring, and centralized control. This section delves into various communication architectures that bridge the gap between the cloud and the factory, enabling efficient and reliable data flow for enhanced operational efficiency and decision-making.

3.2.1. Database Synchronization

Database synchronization is a fundamental aspect of cloud-factory communication, ensuring that data is consistently replicated and updated between the cloud and on-premises systems. This involves establishing mechanisms to propagate changes made in one database to the other, maintaining data integrity and consistency across the entire system.

Various techniques can be employed for database synchronization, including:

- **One-way synchronization:** Changes are propagated from the source database (e.g., factory floor) to the destination database (e.g., cloud).
- **Two-way synchronization:** Changes are bi-directionally replicated between the source and destination databases.
- **Real-time synchronization:** Changes are propagated as they occur, ensuring near-instantaneous consistency.
- **Scheduled synchronization:** Changes are propagated at predefined intervals, offering a balance between consistency and resource utilization.

The choice of synchronization technique depends on factors such as the frequency of data updates, the volume of data transferred, and the desired level of consistency. Careful consideration of these factors is crucial for ensuring efficient and reliable data synchronization between the cloud and the factory floor.

3.2.2. Edge Computing and Gateways

Edge computing involves placing computational resources closer to the data source, reducing latency and bandwidth consumption. In the context of cloud-factory communication, edge devices and gateways play a crucial role in pre-processing and filtering data before it is transmitted to the cloud. This approach minimizes the amount of data sent to the cloud, improving efficiency, reducing costs and improving data security.

Edge gateways act as intermediaries between the factory floor and the cloud, providing functionalities such as data aggregation, protocol conversion, and security enforcement. They can also perform local processing and decision-making, enabling real-time responses to events on the factory floor.

The use of edge computing and gateways in the industrial context can enhance the responsiveness and efficiency, particularly for tasks that require real-time processing or local decision-making. By distributing computational tasks between the edge and the cloud, a system can achieve optimal performance and resource utilization.

3.2.3. Blockchain Integration

Blockchain technology offers a decentralized and immutable ledger for recording transactions and tracking assets. In the manufacturing sector, blockchain can be leveraged to enhance transparency, security, and traceability across the supply chain.

The benefits of integrating blockchain into cloud-factory communication include:

- **Enhanced security:** Blockchain's cryptographic techniques ensure the integrity and immutability of data, protecting against tampering and unauthorized access.
- **Improved traceability:** Blockchain provides a transparent and auditable record of all transactions, enabling the tracking of materials, components, and finished products throughout the manufacturing process.
- **Increased trust:** Blockchain's decentralized nature fosters trust among stakeholders, as data is not controlled by a single entity.

3.2.4. Cloud-Native APIs & Services

Cloud-native APIs and services are designed to leverage the scalability, flexibility, and resilience of cloud computing environments. These APIs and services provide standardized interfaces for accessing cloud resources, enabling seamless integration between cloud-based applications and on-premises systems.

In the context of cloud-factory communication, cloud-native APIs and services can facilitate:

- **Data exchange:** APIs can be used to expose data from factory floor systems to cloud-based applications, enabling remote monitoring, analysis, and control.
- **Application integration:** Cloud-native services can be integrated with on-premises applications, enabling hybrid deployments and extending the functionality of existing systems.
- **Scalability and resilience:** Cloud-native architectures are designed to scale on demand, ensuring that the system can handle fluctuating workloads and maintain high availability.

By utilizing cloud-native APIs and services, factories can benefit from the agility and scalability of the cloud, while maintaining seamless integration with existing floor systems. This approach enables rapid adaption to changing needs and helps to ensure continuous operation even in the face of disruptions.

4. Data Format and Serialization Standards

In the realm of data exchange and interoperability, the choice of data format and serialization standard plays a crucial role in ensuring efficient and reliable communication between different components. There are a lot of data format standards and this chapter explores only the most prominent data format and serialization standards, evaluating their strengths and weaknesses.

4.1. JSON (JavaScript Object Notation)

JSON is a lightweight, text-based data interchange format that has gained widespread popularity due to its human-readable syntax and ease of use. It is based on a subset of the JavaScript programming language, but it is language-independent, making it suitable for data exchange across various platforms and technologies.

JSON's simplicity and readability make it an attractive option for representing data structures in a concise and easily understandable manner. Its widespread adoption and support across various programming languages and tools further contribute to its appeal.

In an industry setting, JSON can be employed for tasks such as exchanging configuration parameters or representing processing instructions. Its lightweight nature and ease of parsing make it well-suited for scenarios where human readability and ease of development are important.

4.2. XML (Extensible Markup Language)

XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is widely used for data exchange and interoperability, particularly in web services and enterprise applications.

XML's extensibility and flexibility allow for the representation of complex data structures and relationships. Its support for namespaces and schemas further enhances its ability to represent data in a structured and well-defined manner.

While XML's verbose nature can lead to increased bandwidth consumption and processing overhead compared to more concise formats like JSON or binary formats, its rich features and established standards make it a suitable choice for scenarios where complex data structures and data validation are paramount.

In an industrial context, XML can be utilized for tasks such as exchanging detailed machine configurations, representing complex processing workflows, or transmitting large datasets with intricate relationships.

4.3. Apache Avro

Apache Avro is a data serialization system that provides a compact, fast, and binary data format. It relies on schemas to define the structure of data, ensuring efficient encoding and decoding while maintaining data integrity. Schemas are sent with each message, ensuring that the message can always be decoded.

Avro's schema-based approach offers several advantages:

- **Compactness:** Avro's binary encoding results in smaller message sizes compared to text-based formats like JSON or XML, reducing bandwidth consumption and storage requirements.
- **Performance:** Avro's schema-based encoding and decoding are highly efficient, minimizing processing overhead and enabling faster data exchange.
- **Data integrity:** Avro's schemas enforce data validation, ensuring that data conforms to the expected structure and preventing data corruption.

Apache Avro is employed for scenarios where high throughput, low latency, and data integrity are critical, such as transmitting real-time sensor data, exchanging large datasets, or storing historical processing logs.

While the serialisation system itself is open source, the serialisation protocol is generally used with systems such as Kafka or Apache Spark, which has a risk of vendor lock-in.

4.4. Protocol Buffers (Protobuf)

Protocol Buffers (Protobuf) is a language-agnostic mechanism for serializing structured data. It offers a compact and efficient binary format, coupled with a language-neutral interface definition language (IDL) for defining data structures.

Protobuf's key features include:

- **Efficiency:** Protobuf's binary encoding results in smaller message sizes and faster processing compared to text-based formats.
- **Language-neutrality:** Protobuf's IDL allows for the definition of data structures that can be used across various programming languages, promoting interoperability.
- **Extensibility:** Protobuf's schema evolution capabilities allow for the modification of data structures without breaking compatibility with existing systems.

Protobuf can be utilized for scenarios where cross-platform compatibility, efficient data exchange, and schema evolution are important considerations, such as communicating between different components written in different languages or ensuring backward compatibility with legacy systems. It is most used with the gRPC streaming protocol.

5. Data Streaming and Messaging Protocol Standards

Efficient and reliable data streaming and messaging are essential for real-time communication and control within the industrial setting. This chapter explores various data streaming and messaging protocol standards, each offering unique features and capabilities tailored to different communication needs.

5.1. OPC UA (Open Platform Communications Unified Architecture)

OPC UA is a platform-independent, service-oriented architecture for industrial communication. It provides a standardized framework for data exchange between various devices and systems, enabling seamless interoperability across different vendors and platforms.

Key features of OPC UA include:

- **Platform independence:** OPC UA can be implemented on various operating systems and hardware platforms, promoting interoperability across diverse devices and systems.
- **Security:** OPC UA incorporates robust security mechanisms, including authentication, authorization, and encryption, ensuring secure communication and data protection.
- **Information modelling:** OPC UA allows for the definition of information models, enabling the representation of complex data structures and relationships.
- **Wide industrial adoption:** OPC UA has been adopted by a wide range of manufacturers of smart devices, making it a good choice for machines which natively support it.

OPC UA can be employed for tasks such as connecting to industrial equipment, exchanging sensor data, or controlling manufacturing processes. Its platform independence and security features make it well-suited for integrating diverse devices and systems within the platform.

5.2. MQTT (Message Queuing Telemetry Transport)

MQTT is a lightweight, publish-subscribe messaging protocol designed for resource-constrained devices and low-bandwidth networks. It is widely used in Internet of Things (IoT) applications, enabling efficient communication between devices and cloud platforms.

Key features of MQTT include:

- **Lightweight:** MQTT's small message overhead and efficient encoding minimize bandwidth consumption, making it suitable for low-bandwidth networks and resource-constrained devices.
- **Publish-subscribe:** MQTT's publish-subscribe pattern enables efficient one-to-many communication, allowing devices to publish data to topics and other devices to subscribe to those topics to receive the data.
- **Quality of Service (QoS):** MQTT offers different QoS levels, allowing applications to choose the level of message delivery guarantee based on their needs.

MQTT could be utilized for tasks such as collecting sensor data from edge devices, transmitting real-time updates, or enabling remote monitoring and control of equipment. Its lightweight nature and publish-subscribe pattern make it well-suited for connecting many devices and efficiently distributing data within the platform.

5.3. AMQP (Advanced Message Queuing Protocol)

AMQP is an open standard application layer protocol for message-oriented middleware. It provides a standardized framework for reliable message exchange between applications, ensuring message delivery and guaranteeing message order.

Key features of AMQP include:

- **Reliability:** AMQP offers robust message delivery guarantees, including persistent message storage and acknowledgment mechanisms, ensuring that messages are not lost in transit.
- **Flexibility:** AMQP supports various messaging patterns, including point-to-point, publish-subscribe, and request-response, providing flexibility for different communication needs.
- **Interoperability:** AMQP is an open standard, promoting interoperability between different messaging systems and vendors.

AMQP can be employed for tasks such as exchanging messages between different components, ensuring reliable delivery of critical data, or integrating with existing messaging infrastructure. Its reliability and flexibility make it well-suited for scenarios where message delivery guarantees and support for various messaging patterns are essential.

5.4. Kafka (Apache Kafka)

Kafka is a high-throughput, distributed streaming platform designed for handling real-time data feeds. It provides a scalable and fault-tolerant infrastructure for ingesting, processing, and storing streams of data.

Key features of Kafka include:

- **High throughput:** Kafka's distributed architecture and efficient data storage mechanisms enable it to handle high volumes of data with low latency.
- **Scalability:** Kafka can be easily scaled horizontally to accommodate growing data volumes and traffic.
- **Fault-tolerance:** Kafka's replication and failover mechanisms ensure data durability and high availability.

Kafka could be utilized for tasks such as ingesting large volumes of sensor data, processing real-time events, or building data pipelines for analytics and machine learning. Its high throughput, scalability, and fault-tolerance make it well-suited for handling the demanding data streaming requirements.

6. API and Communication Protocols

APIs (Application Programming Interfaces) and communication protocols are fundamental building blocks for enabling seamless interaction and data exchange between different software components. This chapter explores various API and communication protocol standards, each offering unique approaches to facilitating communication and data exchange.

6.1. gRPC

gRPC is a modern, open-source, high-performance Remote Procedure Call (RPC) framework that enables client and server applications to communicate transparently. It leverages Protocol Buffers (Protobuf) as its Interface Definition Language (IDL) and underlying message interchange format, providing efficient and type-safe communication.

Key features of gRPC include:

- **Performance:** gRPC's use of Protobuf and HTTP/2 results in efficient and low-latency communication, making it suitable for high-performance applications.
- **Streaming:** gRPC supports bidirectional streaming, allowing for real-time data exchange and asynchronous communication patterns.
- **Code generation:** gRPC provides tools for generating client and server code in various programming languages, simplifying development and promoting interoperability.

In industrial setting gRPC could be employed for tasks such as establishing communication channels between microservices, enabling real-time data streaming from sensors, or implementing remote procedure calls for controlling equipment.

6.2. REST API

REST (Representational State Transfer) API is a widely adopted architectural style for designing networked applications. It leverages the HTTP protocol and its methods (GET, POST, PUT, DELETE) to interact with resources, providing a standardized and flexible approach to building web services.

Key features of REST APIs include:

- **Scalability:** REST APIs are stateless, meaning that each request contains all the information necessary to process it, enabling horizontal scalability and load balancing.
- **Flexibility:** REST APIs can use various data formats, such as JSON and XML, providing flexibility in data representation and exchange.
- **Uniform interface:** REST APIs adhere to a set of constraints, such as uniform resource identifiers (URIs) and standard HTTP methods, promoting consistency and ease of use.

REST APIs could be utilized for tasks such as exposing data from the platform to external applications, enabling user interaction through web interfaces, or integrating with third-party services.

6.3. GraphQL

GraphQL is a query language and runtime for APIs that provides a more efficient, powerful, and flexible alternative to REST APIs. It allows clients to request exactly the data they need, reducing over-fetching and under-fetching of data, and enabling more efficient communication at the expense of a higher server computational cost.

Key features of GraphQL include:

- **Precise data fetching:** GraphQL allows clients to specify the exact data they need, reducing data transfer and improving performance.
- **Strong typing:** GraphQL uses a schema to define the types of data available, enabling data validation and reducing errors.
- **Introspection:** GraphQL allows clients to query the schema, providing self-documenting capabilities and simplifying integration.

GraphQL can be employed for tasks such as providing a flexible API for querying data from the platform, enabling clients to retrieve specific data subsets, or supporting complex data relationships and nested queries.

6.4. WebSockets

WebSockets is a communication protocol that provides full-duplex communication channels over a single TCP connection. It enables real-time, bidirectional communication between clients and servers, making it suitable for applications that require instant updates and interactive experiences.

Key features of WebSockets include:

- **Real-time communication:** WebSockets enables low-latency, bidirectional communication, making it ideal for applications that require instant updates, such as chat applications, online games, and real-time dashboards.
- **Efficiency:** WebSockets reduces overhead compared to traditional HTTP polling, improving performance and reducing bandwidth consumption.
- **Persistence:** WebSockets maintains a persistent connection between client and server, enabling continuous data exchange and reducing the need for frequent connection establishment.

WebSockets can be utilized for tasks such as providing real-time updates on processing progress, enabling interactive control of equipment, or supporting collaborative features within the platform.

7. Security and Data Privacy Standards

Security and data privacy are paramount concerns in industry, given the sensitive nature of manufacturing data and the need to protect intellectual property. This chapter explores various security and data privacy standards, highlighting their importance in ensuring the confidentiality, integrity, and availability of the data.

7.1. TLS (Transport Layer Security, RFC 8446)

TLS is a cryptographic protocol designed to provide secure communication over a computer network. It is widely used to secure web traffic (HTTPS) and other internet-based communications. TLS ensures data confidentiality, integrity, and authenticity through encryption, digital signatures, and certificate-based authentication.

TLS can be employed to secure communication channels between various components, such as the cloud platform and factory floor devices, or between different microservices within the platform. By encrypting data in transit, TLS protects sensitive information from unauthorized access and tampering.

7.2. AES (Advanced Encryption Standard, ISO/IEC 18033-3)

AES is a symmetric-key encryption algorithm widely adopted as the standard for encrypting sensitive data. It is used in various applications, including securing data at rest and in transit. AES offers strong encryption capabilities, with various key lengths (128, 192, or 256 bits) providing different levels of security.

AES can be employed to encrypt sensitive data stored in databases, transmitted between components, or processed within the platform. By encrypting data, AES protects it from unauthorized access and disclosure, even if the storage or communication channels are compromised.

7.3. Zero Trust Architecture (NIST SP 800-207)

Zero Trust Architecture is a security framework that assumes no user or device can be trusted by default, regardless of their location or network. It emphasizes strict identity verification, access control, and continuous authorization to protect resources and data.

Adopting a Zero Trust approach can enhance security by:

- **Minimizing attack surface:** By limiting access to only authorized users and devices, the potential attack surface is reduced.
- **Preventing lateral movement:** If a breach occurs, Zero Trust principles limit the attacker's ability to move laterally within the network, minimizing damage.
- **Enhancing data protection:** By enforcing strict access controls and encryption, sensitive data is protected even if the network perimeter is compromised.

7.4. OAuth 2.0 (For API & Microservices Security)

OAuth 2.0 is an authorization framework that enables third-party applications to obtain limited access to user resources without exposing their credentials. It provides a secure and standardized way to delegate access to APIs and microservices.

OAuth 2.0 can be employed to secure APIs and microservices, allowing authorized users and applications to access specific resources while preventing unauthorized access. This ensures that only authenticated and authorized entities can interact with the platform's functionalities.

OAuth 2.0 allows authentication of users through service providers, and most large tech companies (e.g. Google, Amazon, Microsoft) allow authentication through their own service provider. This removes the need to store user passwords or identification tokens, and greatly decreases the risk of unauthorised access, as users no longer need to have a separate password for the system. Many of these service providers also allow the user permissions to be centrally managed on their platform, removing another source of security leaks.

7.5. IEC 62443 (Industrial Cybersecurity Standard)

IEC 62443 is a series of standards specifically designed to address cybersecurity for industrial automation and control systems (IACS). It provides a comprehensive framework for securing industrial environments, including guidelines for risk assessment, security architecture, and security lifecycle management.

Adhering to IEC 62443 standards can enhance the security of a platform's industrial components, such as industrial processing units, sensors, and control systems. By implementing the security controls and best practices outlined in IEC 62443, the project can mitigate cybersecurity risks and protect critical infrastructure from attacks.

8. Conclusions

The interoperability of data streams is a complex but crucial aspect of the Lasers4MaaS program. This paper has explored a wide range of standards and approaches, highlighting the importance of careful consideration when selecting technologies for data exchange and communication. The choice of message architecture, data format, communication protocol, and security measures will significantly impact the platform's performance, scalability, and security. As demonstrated, there is no one-size-fits-all solution. The optimal approach will depend on the specific requirements of the project, including the volume and velocity of data, the criticality of real-time communication, and the sensitivity of the information being exchanged. It is recommended that a thorough analysis of these requirements be conducted to inform technology selection. Furthermore, staying abreast of evolving standards and best practices in the field of data interoperability is essential for ensuring the long-term success of a solution. This paper provides a solid foundation for making informed decisions and building a robust and interoperable data ecosystem for the program.

9. Sources

5.2 MQTT

<https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/>

<https://mqtt.org/>

5.3 AQMT

<https://www.amqp.org>

5.4 Kafka

<https://kafka.apache.org>

6.1 gRPC

<https://grpc.io/docs/>

6.3 GraphQL

<https://www.apollographql.com/docs>

6.4 Websockets

<https://datatracker.ietf.org/doc/html/rfc6455>

7.1 TLS

<https://datatracker.ietf.org/doc/html/rfc8446>

7.2 AES

<https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

7.3 Zero Trust Architecture

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf>

7.4 OAuth 2.0

<https://datatracker.ietf.org/doc/html/rfc6749>

7.5 IEC 62443

<https://gca.isa.org/hubfs/ISAGCA%20Quick%20Start%20Guide%20FINAL.pdf>