Digital Twins for Green Shipping

D1.1: *Value-oriented Analysis in enabling Shipping Decarbonisation*

Document Information

Grant Agreement No	101056799	**Acronym**	DT4GS
Full Title	Open collaboration and open Digital Twin infrastructure for Green Smart Shipping		
Call	HORIZON-CL5-2021-D5-01: Clean and competitive solutions for all transport modes		
Topic	HORIZON-CL5-2021-D5-01-13	**Type of action**	RIA
Coordinator	INLECOM GROUP		
Project URL	https://dt4gs.eu/		
Start Date	01/06/2022	**Duration**	36 months
Deliverable	D2.2	**Work Package**	WP 2
Document Type	OTHER	**Dissemination Level**	PU
Lead beneficiary	FINCANTIERI NEXTECH (FINC)		
Responsible author	Alessandro Caviglia (FINC)		
Contractual due date	28/02/2023	**Actual submission date**	28/02/2023

Disclaimer and acknowledgements

Funded by the European Union

This project has received funding from the Horizon Europe framework programme under Grant Agreement No 101056799

Disclaimer

Copyright message

Document information

Contributors	
Contributor Name	**Organisation (Acronym)**
Alessandro Caviglia	FINCANTIERI NEXTECH (FINC)
Takis Katsoulakos	INLECOM GROUP (INLE)
Bill Karakostas	INLECOM GROUP (INLE)
Antonis Antonopoulos	KONNECTA SYSTEMS LIMITED (KNT)
Dimitris Skoutas	ATHENA RESEARCH CENTER (ATH)
Austin Kana	Technical University of Delft (TUD)
Konstantinos Papoutsis	EURONAV (ENAV)

Document history				
Version	**Date**	**%**	**Changes**	**Author**
0.1	01/09/2022	1%	ToC Draft	Alessandro Caviglia (FINC)
0.2	16/12/2022	50%	Added content to chapters	Alessandro Caviglia (FINC)
	16/01/2022		Provided contributions	KNT, ATH
0.3	17/01/2023	80%	Added more content to all chapters	Alessandro Caviglia (FINC)
0.8	17/2/2023	95%	Restructuring and new content	INLE
0.9	22/2/2023		Review, Refinement	KNT, TUD, ENAV
1.0	24/02/2023	100%	Final version	Alessandro Caviglia (FINC)

Quality Control (includes peer & quality control reviewing)			
Date	**Version**	**Name (Organisation)**	**Role & Scope**
02/09/2022	0.1	Efstathios Zavvos (VLTN)	QM ToC Approval
18/12/2022	0.2	Efstathios Zavvos (VLTN)	50% Draft approval
07/01/2023	0.3	Efstathios Zavvos (VLTN)	80% Draft approval
24/02/2023	0.9	Bill Karakostas (INLE)	100% PM Review
27/02/2023	1.0	Efstathios Zavvos (VLTN)	Final Version Approval

`

Executive summary

Deliverable 2.2 presents the proposed structure and components of the DT4GS Dataspace. This document represents version 1 of the two deliverables that will be released in the scope of the DT4GS project. The second version, describing the final structure of the Dataspace will be release on the 34th month of the project. Certain aspects of this document build upon and expand the "Transferable DT4GS architecture" (D2.1), which is currently in the finalization stage and is scheduled for release in the 12th month.

Within a maritime context, the Digital Twin enables services mainly focused on decision support/ decision-making, for the purpose of, at company level, optimizing operations and at industry level facilitating the greening transition. DT4GS Dataspace has two main functionalities; to establish the bidirectional data link between the physical vessel, its digital counterpart and components providing the data required for the creation and evolution of the Digital Twin, and additionally to support sharing of data between shipping actors/stakeholders.

The primary objectives of this document are to establish the current state of the art, outline the technical requirements, conduct a comprehensive survey of relevant technologies, and provide an initial definition of the constituent elements comprising the DT4GS dataspace. This is conducted accounting for the requirements defined by the consortium partners but also the state of the art and the intended actors that will actively interact with the Dataspace both for configuration tasks and for the actual utilisation of the Digital Twin. The document presents the needs that the Dataspace have to address to ensure the communication, store and computational capabilities necessary to allow for the operational use of the Digital Twin.

Concerning deliverable's organisation, the next section is devoted to the discussion of IT architectures for shipping/maritime, in particular architectures that enable the Internet of Things (and in this context, the Internet of Ships). Then, the Section 3 discusses the data requirements for the DT4GS project pilots, in terms of data types, storage and processing requirements etc. This leads to the overview of the Dataspace architecture, and the analysis and evaluation of technical choices for data storage, processing and communicating, and finally in Section 4, the selection of the technological solutions that implement them. Section 5 provides more detailed description of the dataspace components, starting from a central component, a Message Broker, developed in Apache Kafka that is responsible for the message exchanges utilised for data transmission within the Dataspace. The technology implied for the communication ship-shore is described highlighting the customisation needs due to the different configurations of the ship informative system and IoT infrastructure between the different ships that will be utilised as Living Labs during the project. Lastly, a brief description of the possible external data sources is presented. Section 6 discusses the different analytics applications envisaged in the Living Labs, and how the Dataspace supports them.

This document, which is delivered in the first part of the project, reflects developments that are currently in progress towards the implement the first version of the Dataspace.

Contents

List of Figures

List of Tables

Glossary of terms and acronyms used

Table 1 Glossary of acronyms and terms.

Acronym / Term	**Description**
ANN	Artificial Neural Network
API	Application Programming Interface
AWS	Amazon Web Services
CNN	Convolutional Neural Networks
DS	Dataspace
DT	Digital Twin
DT4GS	Digital Twin for Green Shipping
ERP	Enterprise Resource Planning
GCP	Google Cloud Platforms
HF data	High Frequency Data
HQ	Headquarters
IDS	International Dataspace
IIOT	Industrial Internet of Things
IoS	Internet of Ships
IoT	Internet of Things
KG	Knowledge Graph
LL	Living Labs
LSTM	Long short-term memory
ML	Machine Learning
PMS	Property Management System
REST	Representational State Transfer
RDBMS	Relational Database Management System
RNN	Recurrent Neural Network
SAND	Streaming Subsequence Anomaly Detection
SotA	State of the Art
VM	Virtual Machine
WP	Work Package

1 Introduction

1.1 DT4GS Overview

As digitalisation in the shipping industry has been maturing over the recent years, DT adoption will be dependent on establishing trusted and convincing DT application exemplars and ensuring that ship operators and other industry stakeholders can set up their own DTs based on their own business models, building their own confidential knowledge at reasonable cost. This requirement is at the heart of the DT4GS approach as illustrated in the Figure 1.

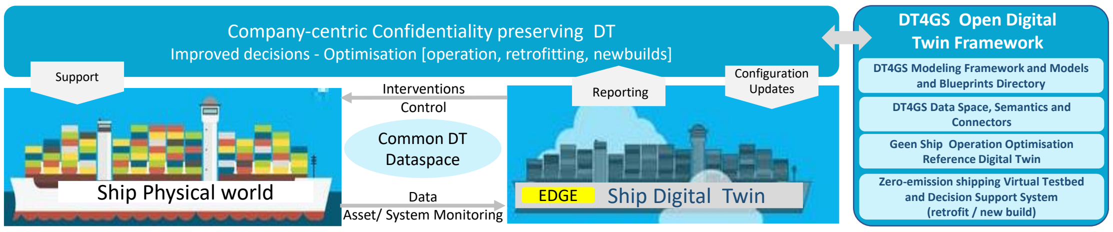
Company-centric Confidentiality preserving DT
Improved decisions - Optimisation [operation, retrofitting, newbuilds]
Support
Interventions
Control
Reporting
Configuration Updates
Common DT Dataspace
Ship Physical world
Data
Asset/ System Monitoring
EDGE
Ship Digital Twin
DT4GS Open Digital Twin Framework
DT4GS Modeling Framework and Models and Blueprints Directory
DT4GS Data Space, Semantics and Connectors
Geen Ship Operation Optimisation Reference Digital Twin
Zero-emission shipping Virtual Testbed and Decision Support System (retrofit / new build)

Figure 1 DT4GS approach

DT4GS is aimed at providing a virtual representation of ships and physical transport entities with a bi-directional communication links from sensing to actuation/control and data driven simulation and AI based decision support. In DT4GS extra emphasis will be given to DT applications onboard the ship utilising advanced IoT computing infrastructure and machine learning techniques. DT4GS therefore, creates a common point of reference in the digital world for shipping vessels, which different stakeholders can access and utilise and adapt in line with their own internal business needs.

To reach its goals DT4GS is divided into 6 WPs each with different goals, tasks, and deliverables.

This deliverable details the work in progress in Work Package 2 and in particular Task 2.2, which is led by the partner Fincantieri NexTech, responsible for the definition and implementation of the DT4GS Dataspace. T2.2 is divided into sub-tasks, each aimed at implementing the key components for data storage and communication between the various entities and applications making up the Dataspace.

Given that this deliverable is written during the first part of the project, while the development of the Dataspace components is still in progress and overall DT4GS Architecture is not yet finalised, the contents of the deliverable will be revised and refined in the next version. The purpose of this deliverable is therefore to provide architectural perspectives and report on components needed to progress with the WP2 early prototyping. However, open topics are underlined in the body of the document and will be described in their final implementation in the context of D2.3 "DT4GS (Green Shipping) Dataspace v2" due month 34 of the project.

1.2 Mapping DT4GS Outputs

The purpose of this section is to map DT4GS Grant Agreement commitments, both within the formal Deliverable and Task description, against the current document.

Table 2 Adherence to DT4GS Grant Agreement deliverable and work description.

DT4GS GA Component Title	DT4GS GA Component Outline	Respective Document section(s)	Justification
DELIVERABLE			
***D2.2* DT4GS (Green Shipping) Dataspace v1**	Connectors Module 1st Version, IoT infrastructure, Internal and External Connectors matching selected priority use cases. This deliverable includes the outputs of T2.2.	Section 5 (chapters 5.1,5.2,5,3, 5.4)	Those chapters describe all the components and technologies to be implemented to develop the DT4GS DS as per GA.
TASK			
***ST2.2.1* Provide a central connectors module**	Enable (a) End-to-end encrypted device authentication management; (b) Streaming and Queuing message exchanges (Apache Kafka, Apache Pulsar); (c) Deployment automation (Kubernetes) supporting cluster configurations for increased message loads and efficient processing; (d) provide a management and configuration User Interface (UI) supporting embedded monitoring and visualisation (e.g. Grafana, ELK); (e)	Section 5 (chapters 5.1,5.2,5.5,5.6,5.7,5.8, 5.11,5.17)	In this chapter is presented the actual solution as projected by DT4GS partners to augment the DT4GS Dataspace with the communication capabilities required as defined in the GA. The functionalities of the Central Connector module are described also with respect to the connection with other components of the dataspace.

	integrate CHARIoT's blockchain-based PKI integration for sensor / gateway authentication and blockchain-aided encryption of IoT endpoints; (f) implement Smart contracts for process automation (certification process, contract process), secure IOT, DT data transparency and trust.		
ST2.2.2 Create the DT4GS IoT infrastructure	Support automated localised data capture, processing, and event-based data flows for better visibility, performance management, and enhanced automation of vessel operations. Implement components for time series processing and analysis, including anomaly detection, change detection, and forecasting, as well as complex event processing by combining data from multiple sensors.	Section 5 (chapters 5.12, 5.14), Section 6	An overall description of the possible algorithms and approaches as per the SotA literature is presented. Some tools supporting the evaluation of the outputs are presented. The evaluation is still ongoing by the project partners, working on the available data to grant the required data quality to enable the DT4GS DT operational functioning.
ST2.2.3. Develop Internal connectors for ship subsystems	Based on the LL requirements and the central connectors module, supply the necessary connectors to link the ship subsystems to the DT4GS infrastructure. Develop and deploy specific APIs (wrappers) to these systems enhanced and semantically enriched via metadata of the DT4GS ontology. Link to the central connectors' module, configuring blockchain enabled connectivity deploying event-based data streaming components**.**	Section 5 (subchapter 5.7.1, chapters 5.10,5.11,5.14)	The solutions for the implementation described in this chapter reflect the actual implementation described in the GA of the project. Security topics, depending on the requirements of the LL, still in gathering phase, will be addressed as per GA description.

ST2.2.4. External Connectors	Based on the LL requirements supply the necessary connectors to setup the links between the central connector's module and external data resources. APIs developed and deployed in the context of the LLs will become available via an API Registry. APIs and metadata descriptions will be semantically enriched, using the DT4GS ontology to enable and facilitate semantic search and interoperability across the LL deployed federated solutions.	Section 3 (subchapter 3.6.2), Section 5 (chapter 5.15,5.16,5.17)	The solutions for the implementation described in this character reflect the actual implementation described in the GA of the project. A first batch of possible external sources to be considered to enrich the DT4GS Dataspace data are presented. Section 3.6.2 explains a solution to implement and leverage external data in the context of the DT operational functioning.

1.3 Deliverable Overview and Report Structure

In this section, a description of the Deliverable's Structure is provided outlining the respective Chapters and their content.

Section 1 Introduction: Overview of DT4GS, Mapping of DT4GS Outputs and overview and structure of the deliverable.

Section 2 Review of the State of the Art (Maritime data characteristics, Ship information architectures, Internet of ships, the international Data Spaces (IDS) Standards), and the overall Information Architecture for Dataspace.

Section 3 Data Requirements for DT4GS including data aspects to be considered, actors participating to the Dataspace, Data Requirements from Living Labs and overview of Data types in DT4GS and their processing requirements. Also, the system architecture of DT4GS Dataspace, data fabric and Knowledge Graphs.

Section 4 Evaluation of data processing infrastructures and selection criteria, including Knowledge Graph technology Selection, IoT/time series database selection, and messaging infrastructures selection.

Section 5 Detailed description of the Dataspace architecture including the Central Connector Module , the Messaging System, the Kafka Cluster, Internal and external Connectors. Also, considerations for data cataloguing and lineage, security, and device authentication and encryption. Also, discussion of platform deployment and monitoring aspects. Additionally, data pre-processing and cleansing, including data from external sources.

Section 6 Analytics application including Time series-processing and Analysis of Sensors Data for anomaly detection. Additionally, discussion on Machine Learning deployment environments to host the analytics applications.

Section 7 Conclusions of the work described in this Deliverable.

Section 8 References

2 State of the Art review

2.1 Maritime data characteristics

Maritime data comes from a variety of sources including the ship bridge data network, conventional automation systems, new 'e-ship' systems, performance monitoring, reporting, Automatic Identification System (AIS), Vessel Traffic Services (VTS), weather and other environmental monitoring, port data and others. This creates issues related to data ownership, heterogeneity, quality etc, that a data processing infrastructure such as the one to be developed in DT4GS needs to address.

Due to the mobility of ships and intermittent communications, data quality may suffer resulting in incomplete, inaccurate, or unreliable data. These issues hinder the seamless use of ship data in decision making applications. Overall, the large volume and variety of data are turning data mining, big data analytics, and data visualization into significantly challenging issues in the maritime domain due to high computation and communication complexities. In addition, the integration of data management technologies that span multiple ships and ports is still an open challenge mainly because of unreliable and slow transmissions as well as incompatible application programming interfaces. The ability to perform timely and cost-effective analytical processing of such large datasets to extract deep insights is a key ingredient for success. These insights can drive automated processes for optimizing ship and fleet operations, and importantly knowledge transfer on decarbonisation solutions and complying with regulation for reducing emissions.

2.2 Ship information architectures

Many maritime information architectures have been developed in the past years, as shown in Figure 2 from the EU MUNIN project.

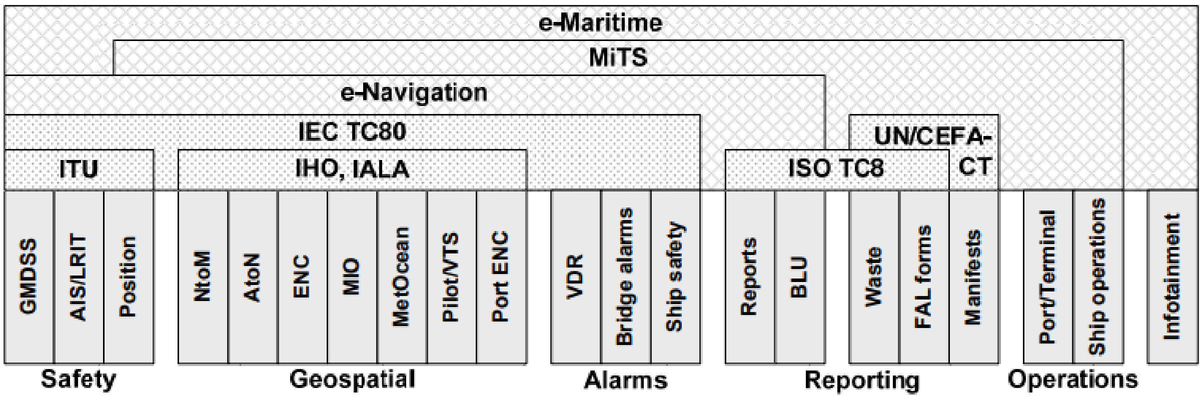

Figure 2 Different maritime information models (source EU MUNIN Maritime Unmanned Navigation through Intelligence in Networks Project)

Each of these models covers a different aspect of maritime such as geospatial information (e.g., ENC), safety (e.g., AIS) or reporting (e.g., Manifests, FAL forms). Because of that, the structure and processing requirements of such information models vary greatly.

2.3 Internet of ships

The Internet of Ships (IoS) paradigm[1] has recently emerged following the concept of e-navigation and the advancements of Internet of Things (IoT) technologies. According to IoT, things/devices/objects are transformed from conventional to smart by sensing, communicating and processing. In the maritime world, these things can be any physical device or infrastructure associated with a ship, a port, or the transportation itself. Figure 3 illustrates how smart sensors and devices, networks middleware and applications realise the Internet of Ships.

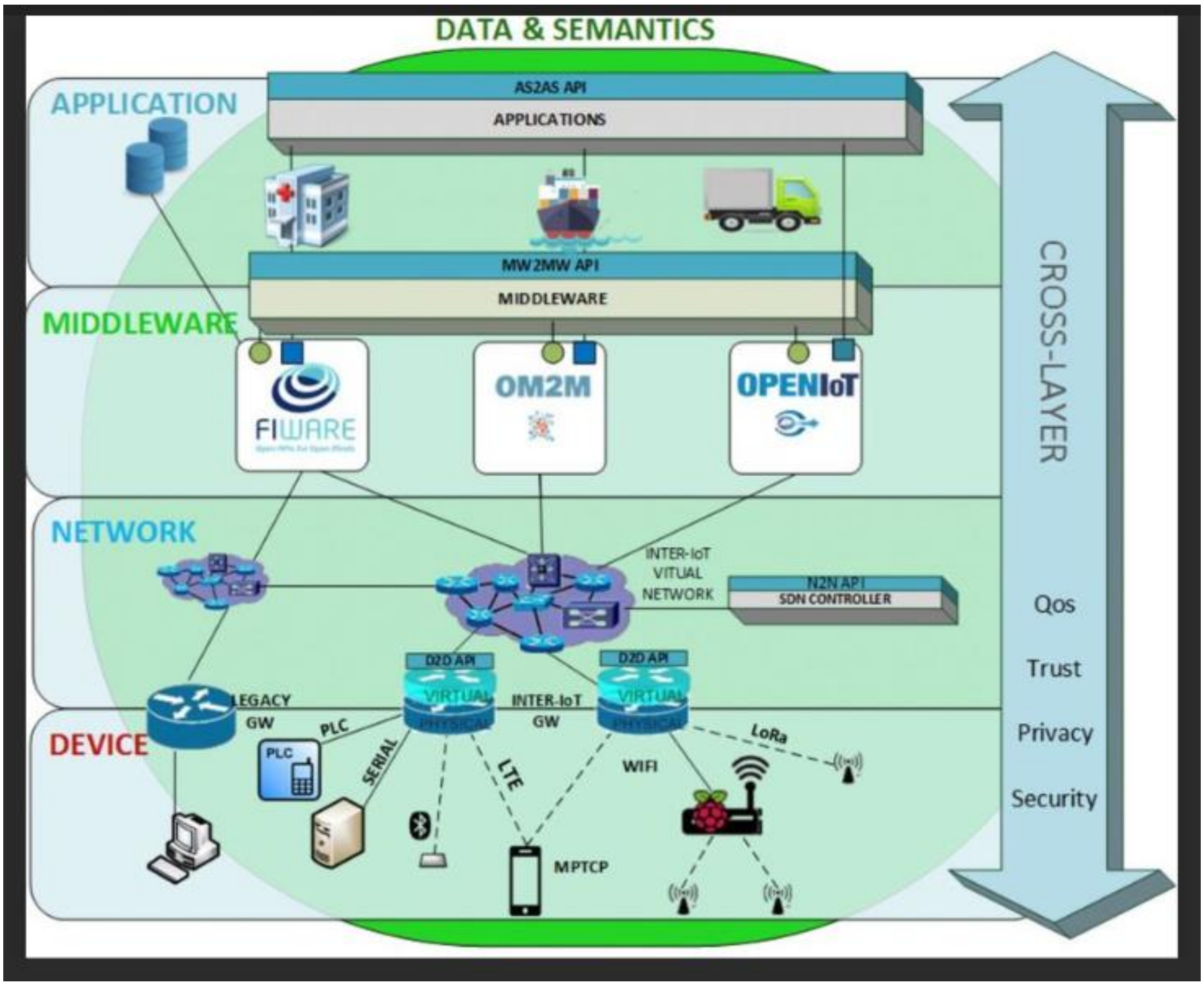

Figure 3 Multitiered IoT architecture [2]

[1] Sheraz Aslam, Michalis P. Michaelides, and Herodotos Herodotou, Internet of Ships: A Survey on Architectures, Emerging Applications, and Challenges. IEEE INTERNET OF THINGS JOURNAL, VOL., NO., MAY 2020

[2] Giménez, Pablo & Llop, Miguel & Olivares, Eneko & Palau, Carlos & Montesinos, Miguel & Llorente, Miguel. (2020). Interoperability of IoT platforms in the port sector.

2.4 The international Data Spaces (IDS) Standards [3]

An obstacle for a wider deployment of ship data use and sharing in an industrial context is the lack of trust in data and its sharing mechanisms. To counter this problem, the International Data Space Association (IDSA) has developed an architecture to define a standard for data exchange on a trusted and self-regulated basis.

The International Data Spaces (IDS) standard creates the basis for the future of a global digital economy. It combines a technical architecture and governance models to facilitate the secure and standardized exchange and easy linking of data in data spaces. The IDS standard guarantees data sovereignty. It allows companies and individuals to self-determine how, when and at what price their data is used along the value chain, thus enabling new intelligent services and innovative business processes. Companies can share any data in any ecosystem, thus transforming the digital economy worldwide.

One key aspect of the IDS standards is the use of a data sovereignty framework, which ensures that data is stored and processed in compliance with the laws and regulations of the country where the data originates. This is achieved using data controllers, who are responsible for ensuring that data is processed in accordance with local laws and regulations.

Another important aspect of the IDS standards is the use of data protection and privacy by design principles. This means that privacy and security considerations are built into the data sharing process from the outset, rather than being an afterthought. This is achieved using techniques such as data anonymization and pseudonymization, as well as the use of secure communication protocols.

The IDS standards also provide guidelines for data interoperability, which is essential for effective data sharing and collaboration. This includes the use of common data formats and protocols, as well as the use of semantic data models to ensure that data can be easily understood and used by different organizations. In addition, the IDS standards also provide guidelines for data governance, which is essential for ensuring that data is used responsibly and ethically. This includes the use of data use agreements and data access policies, as well as the use of data quality and data lineage techniques to ensure that data is accurate and reliable.

In order to achieve the DT4GS goals, it is important to apply the IDS standards in a way that ensures interoperability between partners and enables the exchange of information while maintaining data sovereignty but still emphasizing the open nature of the project.

2.4.1 IDS in the maritime domain

The maritime domain is facing an increasing amount of data generation from different sources such as ships, ports, coastal monitoring systems, among others. The data generated by these sources is valuable for improving the efficiency and safety of maritime operations, but also for addressing environmental and security challenges. However, the data is often siloed, inconsistent, and difficult to share and use.

The IDS framework can help to overcome these challenges by providing a secure and privacy-compliant way of sharing and using maritime data. The IDS can be adapted to the maritime domain by defining specific data models, data controllers and connectors that will enable the interoperability of different data sources, services and applications. The IDS can provide a governance framework to ensure that the data is used ethically and responsibly, and that the data sovereignty and privacy regulations are respected.

[3] International Data Spaces (fraunhofer.de)

One key aspect of the IDS adaptation to the maritime domain is the use of a data sovereignty framework, which ensures that data is stored and processed in compliance with the laws and regulations of the country where the data originates. This is essential for the maritime domain as it often involves data from different countries and international waters. Another important aspect of the IDS adaptation to the maritime domain is the use of data protection and privacy by design principles. This means that privacy and security considerations are built into the data sharing process from the outset, rather than being an afterthought. This is essential for the maritime domain as it often involves sensitive data such as personal data and location data.

The IDS can also provide data interoperability, which is essential for effective data sharing and collaboration in the maritime domain. This includes the use of common data formats and protocols, as well as the use of semantic data models to ensure that data can be easily understood and used by different organizations.

In addition, the IDS can provide guidelines for data governance, which is essential for ensuring that data is used responsibly and ethically in the maritime domain. This includes the use of data use agreements and data access policies, as well as the use of data quality and data lineage techniques to ensure that data is accurate and reliable.

IDS connector

The IDS Connector is the central technical component for secure and trusted data exchange. The connector sends your data directly to the recipient from your device or database in a trusted, certified data space, so the original data provider always maintains control over the data and sets the conditions for its use. The connector uses technology that puts your data inside a sort of virtual "container," which ensures that it's used only as agreed upon per the terms set by the parties involved. Data exposed via IDS Connector is published with data endpoints description to an IDS meta-data broker. This approach allows data consumers to available data sources in terms of structure, quality, eventual obsolescence and other various information.

Figure 4 clarifies the data exchange modalities between different data providers via the IDS Connector application.

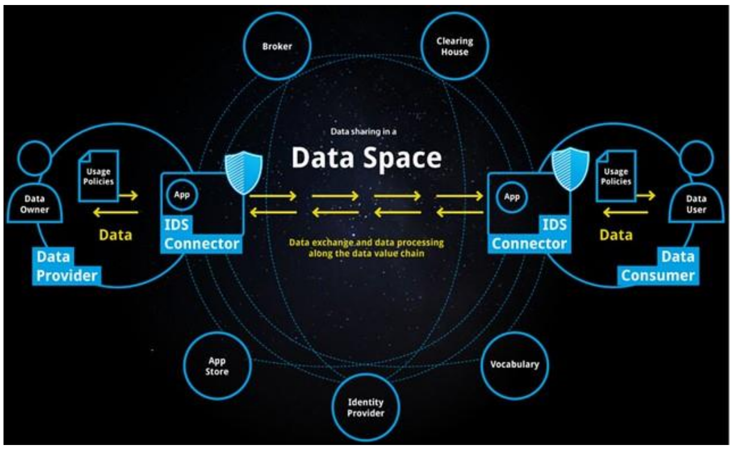

Figure 4 The IDS Connector concept[4]

[4] Source: https://newsroom.eclipse.org/eclipse-newsletter/2021/october/eclipse-Dataspace-connector-trusted-data-sharing-sovereignty

The IDS Connector Library consists of several different types of connectors, including:

1. Data connectors: These connectors provide access to data sources, such as databases and file systems, and enable the retrieval, storage and transformation of data.
2. Service connectors: These connectors provide access to services, such as machine learning or analytics services, and enable the execution of specific tasks on the data.
3. Communication connectors: These connectors provide secure communication channels between different systems and services and enable the exchange of data and metadata.
4. Authentication and authorization connectors: These connectors provide authentication and authorization mechanisms for accessing data and services.

Each connector in the IDS Connector Library has a well-defined interface and is designed to be reusable and composable. This allows different connectors to be combined to create complex workflows for data sharing and collaboration.

The IDS Connector Library is typically used by data controllers, who are responsible for ensuring that data is processed and shared in compliance with local laws and regulations. The library provides a set of functionalities to automate tasks such as data lineage, data quality, data use agreements and data access policies, which are crucial for data governance.

Libraries for the implementation are available for the main programming languages, such as Java, Python, C++ and C#.

2.5 The Information Architecture for DT4GS Dataspace

Figure 5, shows the conceptual information architecture of the Dataspace, inspired by the Internet of Ships paradigm. It can be considered in terms of several layers. The bottom layer (Sensing) consists of the IoT infrastructure onboard the ship that collects and transmits sensor data. The layer above (Inter-network) is responsible for transmitting the data to both onboard and on shore systems using different network and communication technologies. The Data fabric layer is responsible for organising, cataloguing and interlinking the diverse data, and providing them with semantics (metadata) so that can be discovered and utilised by the Services and applications layer. This layer is responsible for the different analytics and other applications running onboard and onshore systems. These applications and services are utilised by the DT4GS actors shown in Figure 6.

The below conceptual architecture will be further analysed and become concrete in the forthcoming sections of this Deliverable.

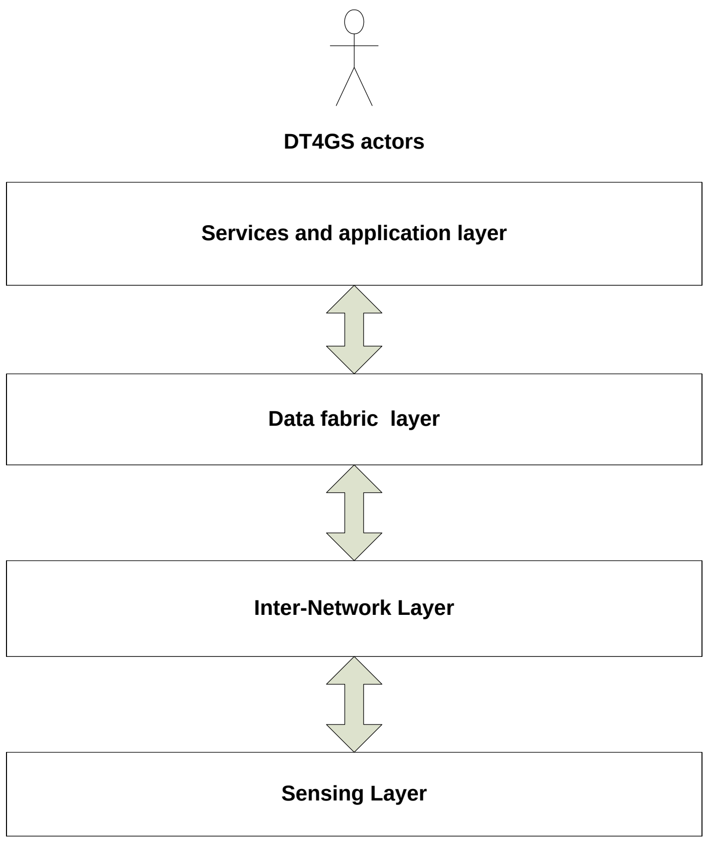

Figure 5 Dataspace information architecture.

3 Data Requirements for DT4GS

3.1 Data aspects to be considered

There are different types of data on the Internet of Things including RFID (Radio Frequency Identification) data, environmental data, location data, multidimensional time series Sensor data and actuator status and control order data. In addition, many of such data are unstructured or semi-structured and may lack precise definitions of their schemata. Lack of strong frameworks for sensor and sensor data security also create problems of lack of ownership and lineage knowledge. Moreover, the sheer volume of sensor data, their stream-based arrival and need to fuse them and process them, often in real time create particular requirements for a suitable data processing architecture.

3.2 Actors participating to the Dataspace

Similar to the IoT data diversity, there is a diversity of data users (consumers). Data producers and consumers may be either on the shore or on the vessel. These are systems including digital twin (DT) type of systems.

The Dataspace facilitates the different actors with the following:

- Acquire data required for the DT operation
- Contribute to the Dataspace with the available information produced by the DT
- Interact with the DT
- Configuring the DT

Figure 6 shows the two software major components and the various actors. These are categorized in Table 3 with respect to the relationship with a DT4GS instance and their role.

The vessel system and the shore must be able to exchange data in a fast and reliable manner to enable real time data processing, allowing for the DT configuration and the display of information both on board and on shore.

Table 3 Summary of Dataspace internal and external actors' roles

Category	**Actors**	**Relationship**
Office end users	Office Engineer Office Operator Chartering Operator Purchaser Crewing Operator	Interact with the DT in a monitoring or decision support role.
Vessel end users	Captain Engineer Desk Officer	Receive collected information and evaluate operational advice by the DT

IT personnel	IT Engineers	Keep the infrastructure working, as well as all the software applications.
Internal data sources	Sensor and other onboard systems	Ship maintenance, control/ supervisory and operation systems
External data sources	Organizations outside of the Ship Owner's.	Provide data to enrich all the information collected on the ships.
Authorities	Organizations outside of the Ship Owner's organisation such as port authorities	They need to receive data from the ship for e.g., regulatory compliance
Model providers	Organizations outside of the Ship Owner's.	They can provide the Dataspace with models that can potentially be leveraged to improve the simulation capabilities of the DT.
Original equipment manufacturer	Organizations outside of the Ship Owner's.	They can provide with components to be potentially integrated into the DS to implement particular functionalities

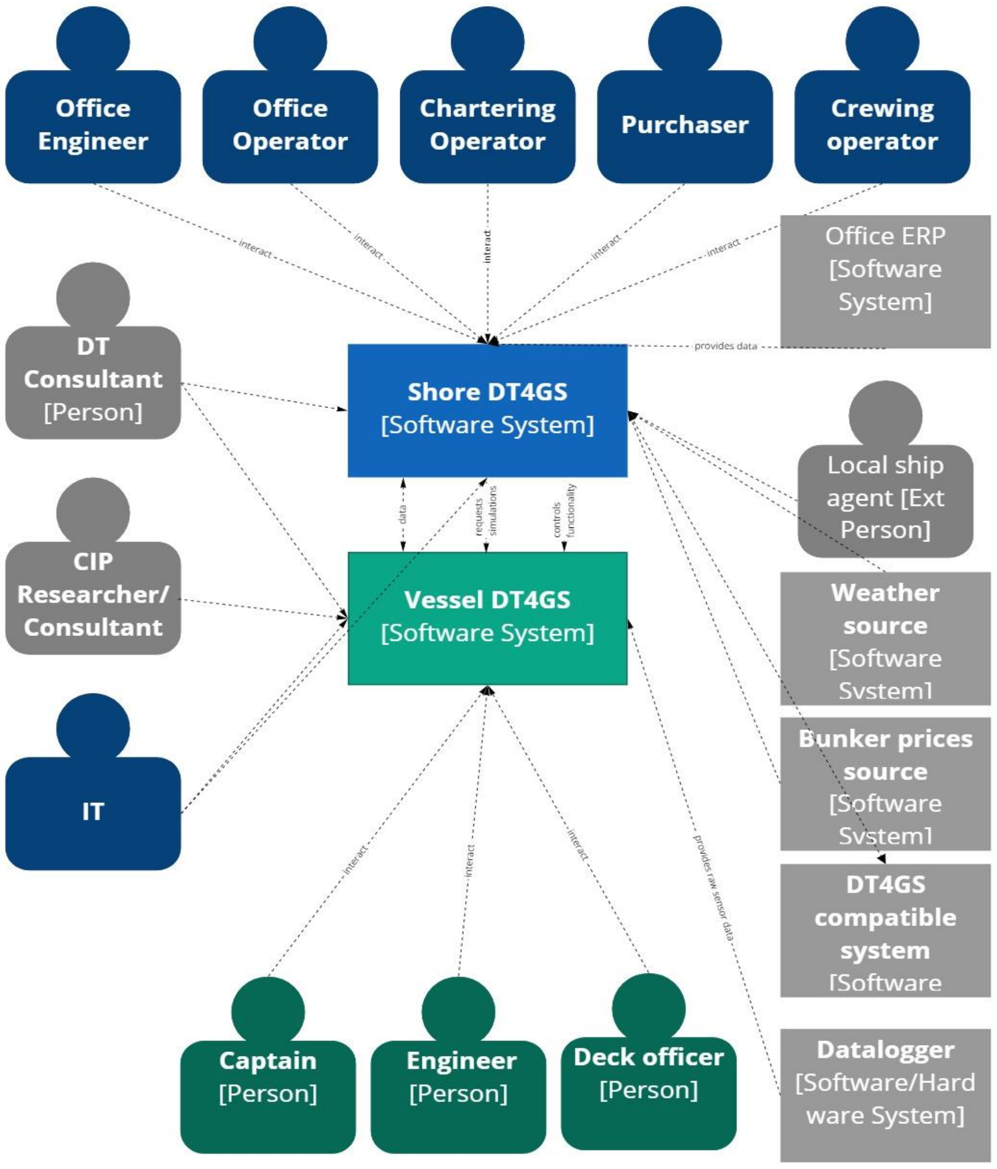

Figure 6 DT4GS ecosystem actors

3.3 Data Requirements from Living Labs

The way values are acquired from on-board sensors varies depending on the ship on which the digital twin system is installed. In this section, a brief review is provided of the on-board data acquisition system as an interface to the DT4GS internal connectors. Given that circumstance, the internal connectors built for the project have to be able to adapt to acquire and process data in different ship infrastructures.

The Living Labs partners and the respective ships are presented and described in deliverable D1.1 of this project.

The DANAOS' EXPRESS ATHENS and EURONAV Alex's ship infrastructure expose an API that can be queried to retrieve real time shipping data collected by the ship's sensors. The internal connector will gather data with a given, configurable, sampling, parse them accordingly in order to make them ready to be sent to the central connector node in a predefined, structured node for the storing. The ship owners will provide the project with the API documentation in order to allow the system to collect data given the decided connector configuration.

BALEÀRIA's Sicilia and Marie Curie ships present a similar configuration with a server that collects and stores on board data and consequently can be queried via a REST API to collect information. Also, in this case the Internal Connector will interface with the API, parse the gathered data and consequently post them on a Kafka topic to send it to the Dataspace.

In the STARBULK's ship case, Maharaj, there is not an infrastructure capable of exposing directly data via an API. The IoT sensor infrastructure is connected to an alarm and monitoring system. In this particular case, the Internal Connector have to interface itself with this particular system to be able to collect real data sensor to provide to the Dataspace.

While this document is being written, contacts are ongoing between the DT4GS T2.2 participants and STARBULK engineers to define how the system will be interfaced.

3.4 Overview of Data types in DT4GS and their processing requirements

Table 4 Data types and requirements in DT4GS

Data name	**Data origin**	**Inbound or outbound data**	**Data characteristics e.g stream/discrete structured/unstructured etc**
Ship location, motion and heading sensors	IoT sensors	Both	Stream, structured
Weather information data	Weather information services	Inbound	Discrete, structured
Hull condition diagnostics	Hull IoT sensors	Outbound	Discrete/stream
Engine condition diagnostics	Engine IoT sensors	Both	Discrete/stream
Fuel consumption	Fuel monitoring sensors	Outbound	Discrete structured
Cargo condition monitoring sensors	Sensors for monitoring cargo condition e.g., temperature etc.	Outbound	Discrete structured/unstructured

Table 4 Data types and requirements in DT4GS summarises the data types that are considered in DT4GS and provide requirements for the design of the Dataspace data processing capabilities. Such capabilities in terms of data storage, processing, distribution, security etc are further described in the following sections of this Deliverable.

3.5 Overview of the System Architecture of DT4GS Dataspace

A Dataspace is a data infrastructure that organizes information into a meaningful and easily accessible format. It aims to store, manage and retrieve data in an efficient and organized manner. The purpose of a Dataspace is to provide a centralized location for data, enabling multiple users to access, manipulate and analyse information quickly and easily, leading to better data-driven decision making and insights.

The data to be processed and digested in a Dataspace may be of different nature and origin, therefore one of the key characteristics to be assured in a Dataspace is a common formatting of the data itself to grant easy accessibility to it. It is essential that security is ensured in the passage of data between the various components of the Dataspace. Finally, a fast retrieval of data has to be a focal goal in the design and development of the Dataspace, in particular in the DT4GS use case, to allow for a real-time processing and consequently enable applications like a real time optimization tool.

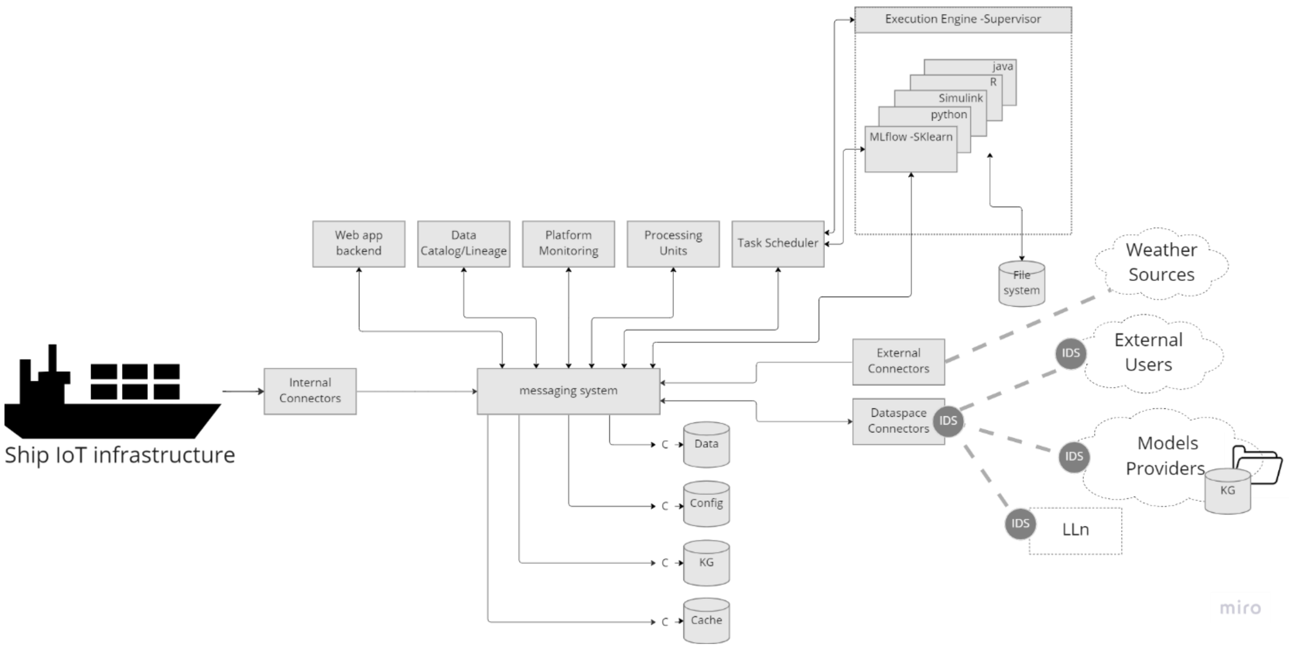

Figure 7 DT4GS Dataspace components and link with components of the DT4GS DT architecture

The logical components of the DT4GS Dataspace architecture are:

- Central Connector;
- Data storage;
- Internal Connectors;
- External Connectors;
- Dataspace Connectors.

Each of those entities as visualised in Figure 7 can be decomposed into smaller blocks that implement particular functionalities of the Dataspace and that are developed leveraging the state-of-the-art technologies.

Those entities, shown in Figure 7, will be briefly introduced in this chapter and then subsequently better explained in the following sections of this document.

The Central Connector

The Central Connector has the purpose to interconnect the components of the system while ensuring security and transparency. Allows various components of the DS to exchange data and information via message exchanges. The technology of choice for the implementation in the DT4GS DS is Apache Kafka that implements a publish-subscribe (pub-sub) model. More on the topic will be described in section 5.2. In Figure 7, the messaging system node is shown, responsible for actual message routing between components, Data catalog and Lineage, platform monitoring, processing units and the task scheduler, (analysed in Sections 5.5 and 5.6) that triggers the various applications to enable particular routines and simulation execution within the DT. The Central Connector Node represents the core of the Dataspace, allowing for data centralization.

Data storage

As already stated, the DT4GS ecosystem is composed of different kind of data to be shared and processed between its internal software applications and components. Different data storages utilities are applied in the DS. Data streams collected by on-board sensors are stored in a time-series data format utilising a NoSQL database called InfluxDB (discussed in Section 4.2) by the node *Data*. A configuration database (node *Config*) is required to store configurations data for all the software involved in the digital twin and associated services.

The models built by the DT4GS partners, in particular in the scope of WP1, are stored in File System. Different storing formats are in evaluation while completing the DS architecture definition. In particular, the adoption of the Functional Mock-up Interface (FMI) is to be considered as the best solution, with it being supported by 170+ tools and maintained as a Modelica Association Project[5]. FMI is an open standard that defines a container and an interface to exchange dynamic simulation models using a combination of XML files, binaries and C code, distributed as a ZIP file.

The node KG (further discussed in Section 4.1) represents the data fabric of the entire DT4GS Dataspace, using a graph data structure that is used to represent knowledge in the form of nodes and edges.

In order to ensure better performance of the DT in data processing, a cache-like memory (node Cache) will be set up that will contain at runtime the last data used, to allow quick access and reusability, some reference data that are particularly useful and intensively used by all applications, and finally useful data for the process.

Internal Connectors

The Internal connectors (analysed in Section 5.3) are responsible for the exchange of sensory and other data from the Vessel System and the Shore System. Sensor data will be recovered from the on-board systems available on the ship, partially processed and analysed and then exported towards the Central Connector node for the actual storing of the information, to make them available and usable to the Digital Twin applications. In line with previous statements, the technology chosen for information exchange will be Apache Kafka (further analysed in Section 5.2), so as to ensure consistency and interoperability between the Vessel System and the Shore System, hosting the central connector.

External Connectors

Not all the information required for the DT to operate is directly available from on-board and internal data. Based on the Living Labs requirements, necessary connectors, External connectors, (analysed in

[5] MA Projects — Modelica Association

Section 5.4) are created to enable the digital twin systems to retrieve information from external sources, such as weather-data APIs, fuel availability and prices data, information relative to other ships in navigation, etc.

Dataspace Connectors

Given the niche aspect of digitalization in the maritime industry when viewed from a broader industrial perspective, it may be a wise strategy to adapt solutions from broader domains to the maritime context in order to reap benefits from adoption, support, and future developments. The IDS standards (discussed in Section 2.4), applied in the maritime context can allow for a more structured data handling and sharing between separate entities, part of different organizations. As showed in Figure 7, IDS connectors can allow other users, external from the organization to interact and retrieve data from the Dataspace based on rules defined in the connector itself. Models, Knowledge graph and other information can be exchanged with other Dataspaces together with new potential use cases to be considered and managed by the DT4GS DT to achieve the goals set by the partners.

3.6 DT4GS Dataspace

3.6.1 Data fabric

Because the DT4GS Dataspace is tasked to support a very broad type of ship data with diverse structure, format, ownership, processing requirements and so on, it requires a dedicated infrastructure that can make such data easily discoverable, accessible, integratable and usable. In data management literature this infrastructure is called data fabric[6]. Using a data fabric, data from disparate sources can be found, understood and integrated to support analytics applications faster and at a lower cost.

One technology that can realise data fabric is the Knowledge Graph. The next section discusses selecting a suitable Knowledge Graph technology to implement a data fabric in Dataspaces.

3.6.2 Knowledge Graph

Given the different software applications, services and tools that cooperate to allow the DT4GS system to effectively map the actual ship behaviour and support the operational utilization of the digital twin itself, the Project proposed the use of Knowledge Graphs to support the data flow and the orchestration of the computational steps required for a given operation.

Knowledge graphs (KG) are a powerful tool for representing and reasoning about large and complex sets of data. From a general point of view, they are a type of graph data structure that is used to represent knowledge in the form of nodes and edges. KG are becoming increasingly important in recent years as the amount of data available on the internet has grown.

There are several key features of knowledge graphs, including:

- Representation of entities: KG represents entities in the form of nodes, which can be anything from people, organizations, and locations to concepts and ideas.
- Representation of relationships: they represent relationships between entities in the form of edges, which can be anything from "is-a" relationships to more complex relationships such as "part-of" or "works-for".

[6] Using Data Fabric Architecture to Modernize Data Integration (gartner.com)

- Semantic representation: they use a consistent vocabulary and ontology to represent entities and relationships, which allows for automated reasoning and inferencing.
- Scalability: KG can be scaled to handle large amounts of data and are able to handle millions of nodes and edges.

In the scope of the DT4GS KG are utilized to describe a data elaboration pipeline, highlighting the inputs data to gather from the data storage, the models and tools to utilize for data processing, and the actual outputs to be produced and stored in the Dataspace itself. An example of this concept is presented in Figure 8 below.

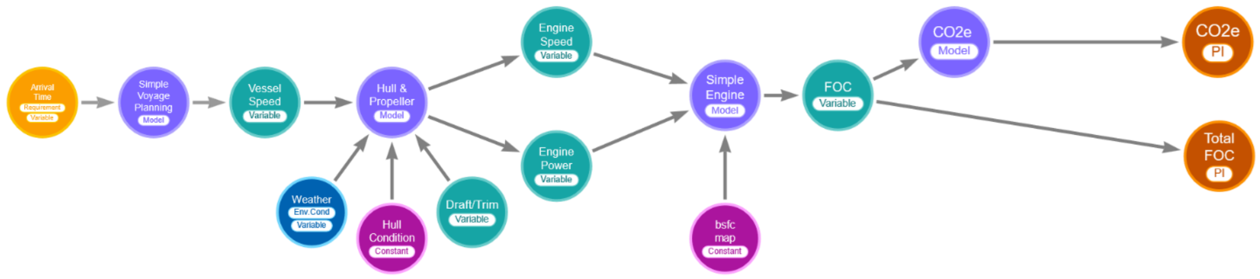

Figure 8 Example of Knowledge Graph to describe the computational steps to obtain CO2e and FOC given the required models and data

4 Evaluation of data processing infrastructures and selection criteria

4.1 Knowledge Graph Selection

There are several software solutions that implement knowledge graphs, including:

Neo4j[7]*:* Neo4j is an open-source graph database that can be used to create and store knowledge graphs. It supports Cypher, a graph-oriented query language that allows you to traverse and query your graph data. Neo4j also provides a web-based user interface and a set of APIs for interacting with the graph data.

GraphDB[8]*:* GraphDB is a proprietary triplestore that allows you to store, manage and query RDF data. It has aset of features such as reasoning, geospatial and temporal data handling, and text search. GraphDB is often used in semantic web and linked data projects.

GraphQL[9]*:* GraphQL is a query language that allows you to define the structure of your data and the operations that can be performed on it. It is often used to build APIs and can be used to create and store knowledge graphs.

These solutions are in evaluation by the DT4GS partners to select the one that best fits the system needs.

4.2 IoT Database Selection

In order to properly handle IoT data generated by the different ship sensors, databases play a critical role. As a consequence, the selection of the right database is crucial. Key factors for selecting a database for storing an IoT applications[10] include indexing, size, scaling, reliability when dealing with vast numbers of data, user-friendly schemas, languages for querying, convergence and heterogeneity, aggregation of time series, archiving, security and cost effectiveness.

The following is a list of candidate IoT databases:

CrateDB[11] is a distributed SQL database management system. It is designed for high scalability since it is open source and written in Java, and it includes components from Facebook Presto, Apache Lucene, Elasticsearch, and Netty.

MongoDB[12] is a document-oriented database software that is available as a free and open source cross-platform framework. It is classified as a NoSQL database application. MongoDB makes use of JSON-style documents with schemas.

RethinkDB[13]: RethinkDB is an open-source database, built-from-the-ground-up portable JSON database for the real-time Network

[7] https://neo4j.com
[8] Ontotext GraphDB
[9] GraphQL | A query language for your API
[10] https://iot4beginners.com/top-5-databases-to-store-iot-data/)
[11] CrateDB – Distributed SQL Database Enabling Data Insights at Scale
[12] MongoDB: The Developer Data Platform | MongoDB
[13] RethinkDB: the open-source database for the realtime web

SQLite[14]: SQLite Database Engine is a process library that offers a transactional SQL database engine that is serverless (standalone). Because of its portability and small footprint, it has had a huge impact on game and mobile application growth.

IoTDB Apache[15]: IoTDB is an IoT native database with high performance for data management and analysis, deployable on the edge and the cloud.

4.2.1 Time series databases

Time series databases (TSDBs) allow the storing of data over a period of time. Time series enables the inserting or appending data in comparison to updating or deleting data. TSDBs can be used in IoT devices and system software to obtain metrics for predictive analytical purposes. TSDB are used for storing sensor logging data that can be used in analysis to aid in predictive maintenance. However, TSDBs can suffer from data volume limitation, difficulty in optimizing read and write operations to prevent overlapping, and difficulty in handling static data[16].

Some well-known time series databases include:

- InfluxDB [17]
- Graphite[18]
- TimescaleDB[19]
- Apache Druid[20]
- RRDTool[21]

4.2.2 Rationale for selecting Influx DB

The technology of choice for the storage of data driven application is InfluxDB. InfluxDB is a NoSQL database optimized for the ingestion of time-series data like the one we will retrieve from ship IoT sensors.

InfluxDB is a time series database built to handle high write and query loads. InfluxDB is a custom high-performance datastore written specifically for time-stamped data, and especially helpful for use cases such as DevOps monitoring, IoT monitoring, and real-time analytics. Data can be permanently conserved, but the DBMS is configurable to potentially setup expiration data logics for part of the data stored. While InfluxDB also offers a SQL-like query language for interacting with data, granting good performances on data querying and easy composition interpretation of the requests.

We selected InfluxDB for storing sensor data in the Dataspace for the following reasons:

1) InfuxDB allows series to be indexed.

[14] SQLite Home Page
[15] https://iotdb.apache.org
[16] Time Series Database (TSDB) (opengenus.org)
[17] https://influxdata.com
[18] https://graphiteapp.com
[19] Time-series data simplified | Timescale
[20] Druid | Database for modern analytics applications (apache.org)
[21] RRDtool - About RRDtool (oetiker.ch)

2) InfluxDB allows automated data down sampling and has a SQL-like query language.

3) InfluxDB has built-in linear interpolation for missing data.

4) InfluxDB allows for continuous queries in order to calculate aggregates.

4.2.3 InfuxDB Technology Stack

The whole InfluxDB tool suite is named TICK Stack. TICK stands for Telegraf, InfluxDB, Chronograf, and Kapacitor, which are integrated in a cohesive architecture, or "stack." Together these technologies provide a platform that can capture, monitor, store, and visualize all data in a time series, allowing for informed business decisions in real-time.

Below are briefly presented the components of the TICK stack:

Telegraf

Telegraf is a plugin-driven server agent for collecting and reporting metrics. Telegraf plugins source a variety of metrics directly from the systems it runs on, pulling metrics from third-party APIs or even to listen for metrics via a StatsD and Kafka consumer service. It also has output plugins to send metrics to a variety of other datastores, services, and message queues, including InfluxDB, Graphite, OpenTSDB, Datadog, Librato, Kafka, MQTT, NSQ and many others.

InfluxDB

The TSDB itself.

Chronograf

Chronograf is the administrative user interface and visualization engine of the stack. It allows to setup and maintain the monitoring and alerting for the infrastructure. It also includes templates and libraries that allow the user to rapidly build dashboards with real-time visualizations of your data and to easily create alerting and automation rules.

Kapacitor

Kapacitor is a native data processing engine. In DT4GS we can potentially leverage Kapacitor capabilities on TS data elaboration to implement the logics and algorithms of the Dataspace. It can process both stream and batch data from InfluxDB. Kapacitor lets the user plug in a custom logic or user-defined functions to process alerts with dynamic thresholds, match metrics for patterns, compute statistical anomalies, and perform specific actions based on these alerts, like dynamic load rebalancing. Kapacitor integrates with HipChat, OpsGenie, Alerta, Sensu, PagerDuty, Slack and more.

InfluxDB natively supports Kafka as an intermediate buffer before TS data is persisted for processing, analysis, and use in other applications.

Cache memory and Configuration data

The amount of data processed by the various tools that will cooperate to implement the Digital Twin application can be pretty huge. Given the real time requirements of some computation at the base of the system, fast access time to some data is required. One solution to this problem is to use a cache memory system, which stores frequently accessed data in a fast and easily accessible location. This approach can significantly improve the performance of a data space by reducing the number of requests to the primary data store and consequently the amount of data that needs to be transferred over the network.

4.2.4 Rationale for selecting Mongo DB

Apart from sensor data, a variety of other types of ship data needs to be handled by Dataspace, as explained in previous sections. For this application a NoSQL database like MongoDB is the natural and more suited choice. There are, in fact, various reasons to lean towards to a NoSQL database to store that kind of data, such as more flexibility, scalability, better performances and even implementation cost reduction. Therefore, MongoDB was selected for the following reasons:

- Flexibility: MongoDB is designed to handle dynamic, unstructured data and can easily accommodate changes to the schema. Configuration data often requires the ability to add, modify or remove fields quickly, making it a good fit.
- Scalability: MongoDB databases are horizontally scalable, allowing them to easily handle large amounts of configuration data, which often need to be updated frequently.
- Performance: MongoDB is optimized for fast and efficient retrieval of data, making them well suited for use cases that involve frequent reads and updates of configuration data.
- Cost: compared to traditional relational databases these solutions are usually less expensive.

MongoDB is document-oriented to allow for easy storage and retrieval of data, and it supports a wide range of data types. It also has built-in support for sharding, which allows for horizontal scaling and improved performance. MongoDB capacity to store some data directly in memory allows for the implementation of high-performance applications that require low-latency access to data.

4.3 Messaging infrastructures selection criteria

4.3.1 Messaging frameworks

The following are some popular messaging frameworks for IoT[22]:

1) MQTT (Message Queue Telemetry Transport)
2) AMQP (Advanced Message Queue Protocol)
3) DDS (Data Distributed Service)
4) XMPP (Extensible Messaging and Presence Protocol)
5) CoAP (Constrained Application Protocol)

To select a message processing framework for the needs of the Dataspace use cases the following needs to be considered[23]:

- Performance: The framework should be able to handle the required volume and rate of messages, and it should have low overhead and latency.
- Reliability: The framework should support delivery guarantees and error handling mechanisms to ensure that messages are delivered reliably.
- Security: The framework should support security features such as encryption and authentication to protect against unauthorized access and tampering.

[22] https://build5nines.com/top-iot-messaging-protocols/
[23] https://build5nines.com/top-iot-messaging-protocols/

- Compatibility: The framework should be compatible with the devices and infrastructure that are being used in the system.
- Ease of implementation: The framework should be easy to implement and integrate with the rest of the system.
- Scalable data movement and processing: the framework can handle backpressure and can process increasing throughput
- Agile development and loose coupling: different sources and sinks should have their own decoupled domains. Different teams can develop, maintain, and change integration to devices and machines without being dependent on other sources or the sink systems that process and analyze the data.
- Innovative development**:** new and different technologies and concepts can be used depending on the flexibility and requirements of a specific use case.

But several challenges increase the complexity of IoT integration architectures:

- Complex infrastructure and operations that often cannot be changed.
- Integration with many different technologies like MQTT or OPC Unified Architecture (OPC UA) while also adhering to legacy and proprietary standards
- Unstable communication due to unreliable IoT networks, resulting in high cost and investment in the edge

4.3.2 Stream Processing architectures

A streaming architecture is a defined set of technologies that work together to handle stream processing, i.e., series of data at the time the data is created. In many modern deployments, some streaming architectures include workflows for both stream processing and batch processing

Two popular stream processing architectures are the Kappa and Lambda Architectures. The Kappa Architecture is considered a simpler alternative to the Lambda architecture as it uses the same technology stack to handle both real-time stream processing and batch processing. Both architectures handle the storage of historical data to enable large-scale analytics. The main difference with the Kappa Architecture is that all data is treated as a stream.

4.3.3 Kafka

The Kappa Architecture is typically built around Apache Kafka along with a high-speed stream processing engine.

The reason for selecting Kafka for the DataSpace messaging[24] needs is that traditional message brokers like Java Messaging Service (JMS)[25], Apache ActiveMQ[26], RabbitMQ[27], and others are not designed to handle high volumes of messages and provide fault tolerance. Apache Kafka on the other hand, is designed to handle large volumes of messages and provide fault tolerance. It can be used as the central nervous system of a distributed architecture that delivers data to multiple systems. Apache Kafka is a

[24] Why Kafka is the Future of Messaging | Engineering Education (EngEd) Program | Section
[25] Getting Started with Java Message Service (JMS) (oracle.com)
[26] ActiveMQ (apache.org)
[27] Messaging that just works — RabbitMQ

distributed, partitioned, replicated commit log service. It helps you process a large amount of data through the use of various Kafka consumers. Kafka is available as a standalone server that you can install on your local machine and various other applications can consume it. The primary reason behind developing Apache Kafka was to serve companies with fast and efficient data pipelines.

With a sufficiently fast stream processing engine, Kafka can be used for both batch and real time stream processing.

5 Detailed Architecture description

5.1 The Central Connector Module

The Central Connector has the purpose to interconnect the components of the system while ensuring security and transparency. Allows various components of the Dataspace to exchange data and information via message exchanges. The technology of choice for the implementation in the DT4GS DS is Apache Kafka that implements a publish-subscribe (pub-sub) model, as explained in the previous section.

As already stated, the DT4GS ecosystem is comprised of different kind of data to be shared and processed between its internal software applications and components. Different data storages utilities are applied in the Dataspace. In particular, data streams collected by on-board sensors are stored in a time-series data format utilising the NoSQL database InfluxDB (reviewed in section 4). A configuration database is required to store configurations data for all the software involved in the digital twin and associated services.

In order to ensure better performance in data processing, a cache-like memory will be set up that will contain at runtime the last data used, to allow quick access and reusability, some reference data that are particularly useful and intensively used by all applications, and finally useful data for the process.

The purpose of the Central Connector Module is to interconnect the components of the Dataspace while ensuring security and transparency.

In Figure 9 is shown the architectural diagram of the DT4GS infrastructure. The items in blue are the components of the central connector module. A "Message Broker" receives data and requests and properly routes them the intended destinations. The "platform monitoring" is a UI to visualize system performance metrics and provide interface for the configuration of the remaining components. The "Data Catalogue/Lineage" component provides a detailed inventory of all data assets and their destination, utilization and transformation over their lifetime.

In the following paragraphs are presented the following:

- An overview of the messaging system,
- The Data Catalog and Lineage description,
- A preview of the capabilities and graphic output of the Platform Monitoring Interface,
- An overview of the possible security implications and of the evaluated approaches,
- The chosen technology for the deployment of all the Dataspace components.

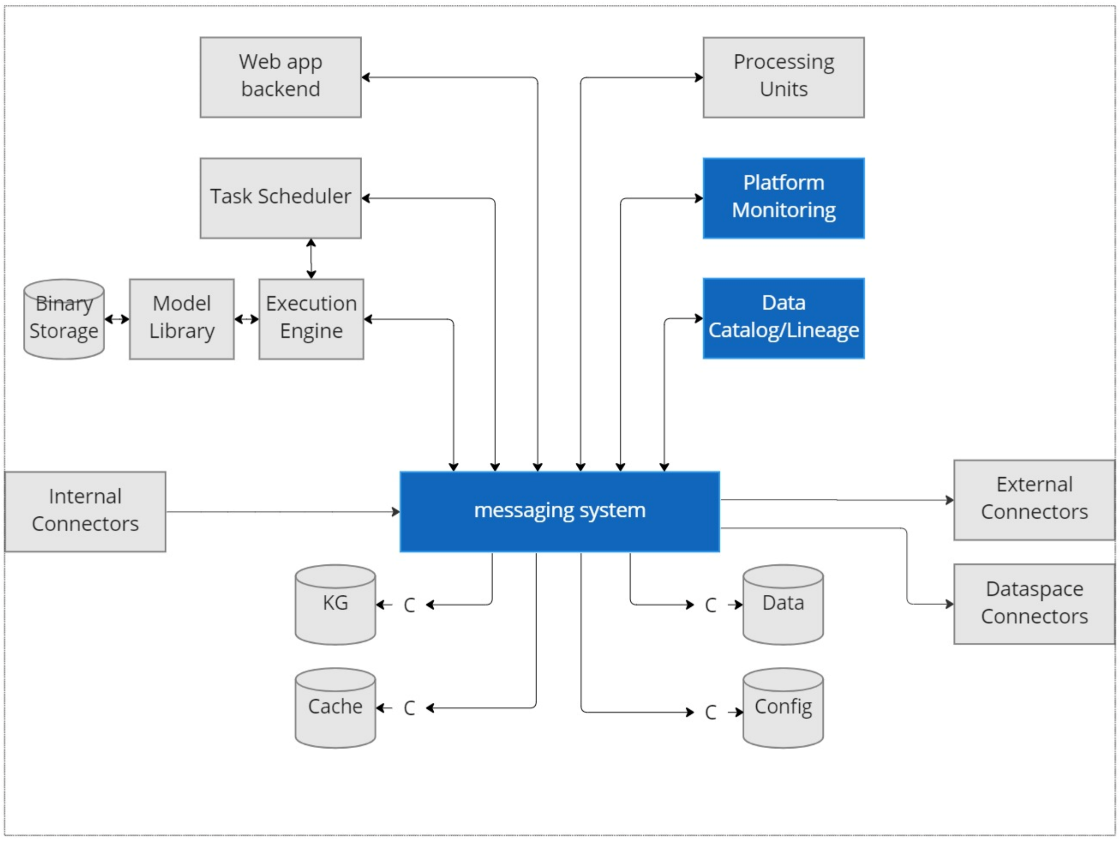

Figure 9 Architectural diagram of the DT4GS infrastructure with highlighted the components of the central connector module.

5.2 The Messaging System

As illustrated in Figure 9, data from the internal and external connectors, are ingested into the platform using a distributed event streaming mechanism. As explained in Section 4.3 the chosen technology is Apache Kafka (Kafka), a well-established open-source event streaming platform that is often preferred for applications with real time data.

The Kafka cluster is administered by Zookeeper service (discussed in the next section); this service keeps track of the cluster's metadata, such as the nodes, topics, partitions and so on. These will now be explained in more detail.

5.2.1 The Kafka Cluster

A Kafka cluster is a group of Kafka nodes that work together to handle the incoming data streams. In a Kafka cluster, there are typically multiple broker servers. Each broker is responsible for managing a portion of the data streams, called a partition. Each partition is made up of a sequence of messages, called a log. Producers write data to the cluster by sending messages to a specific topic, and consumers read from the cluster by subscribing to one or more topics.

One key feature of Kafka is its ability to handle high levels of data ingestion and processing. This is achieved through horizontal scalability, where new brokers can be added to the cluster as needed to handle increased traffic. Additionally, Kafka uses a technique called replication to ensure that data is not lost in the event of a broker failure. Each partition is replicated across multiple brokers, so if one broker goes down, the others can continue to serve the data.

Another important aspect of Kafka is its support for real-time data processing. This is achieved through the use of a commit log, where each message is written to disk as soon as it is received. Data Consumer services can then read from the log and process the data in real-time. Additionally, Kafka provides support for complex data pipelines through its support for stream processing, where data can be transformed and processed as it is ingested.

A vital component of a Kafka cluster is a distributed coordination service called ZooKeeper. ZooKeeper is a centralized service that is responsible for maintaining configuration information and providing distributed synchronization. In a Kafka cluster, ZooKeeper is used to manage the configuration of the brokers, track the location of partitions, and maintain a record of the overall health of the system. When a new broker is added to the cluster, it registers with ZooKeeper and receives its configuration information. Additionally, if a broker's configuration is changed, the change is propagated to ZooKeeper, and all other brokers in the cluster are notified of the change. Each partition in a Kafka cluster is replicated across multiple brokers, and ZooKeeper keeps track of which broker is the leader for each partition. This information is used by the producers and consumers to determine where to send and receive data. In the event of a broker failure, ZooKeeper will also coordinate the re-election of a new leader for the affected partitions.

Finally, ZooKeeper also plays a critical role in maintaining the overall health of the system. It stores a record of the current state of the cluster, including the status of each broker and partition. This information is used by the clients to determine the health of the system and route their requests accordingly.

In the DT4GS implementation, the central node, composed of the broker servers is in charge of granting the distribution and persistent storing of record and messages.

Although only three components of the Dataspace are highlighted in blue in Figure 9 (as nodes fully dedicated to the implementation of the Central Connector Module), all components of the DS will be implemented as Kafka nodes, be they pure and simple consumers, or producers of data, as indicated by the arrows in the figure. This architecture allows for fast data retrieval times for all the DS-related applications that make up the DT4GS system environment. To achieve this communication capabilities the components will be empowered via Kafka Connect. To better explain this concept, we can analyze how the time series sensor data will be stored in the Dataspace once collected from the actual ship IoT infrastructure. The ship, as a producer, will publish on a dedicated topic, routinely given the chosen sampling interval, sensor data. The messages will be stored by the Central Connector node in order to make them available to the subscribers of the topic. Once triggered, or routinely, the InfluxDB instance will leverage the available dedicated Kafka Connect API to read from the topic and acquire the time series for processing and storing.

5.3 Internal Connectors Nodes

The Internal connectors are responsible for the exchange of sensory and other data from the Vessel System and the Shore System. Sensor data will be recovered from the on-board systems available on the ship, partially processed and analysed and then exported towards the Central Connector node for the actual storing of the information, to make them available and usable to the Digital Twin applications. In line with previous statements, the technology chosen for information exchange will be Apache Kafka, so as to ensure consistency and interoperability between the Vessel System and the Shore System, hosting the central connector.

5.4 External Connectors Nodes

Not all the information required for the Digital Twin to operate is directly available from on-board and internal data. Based on the Living Labs requirements, necessary connectors are created to enable the digital twin systems to retrieve information from external sources, such as weather-data APIs,

5.4.1 Dataspace Connectors Nodes

The Dataspace Connector is an IDS connector that is currently being maintained by Sovity[28]. The connector was originally developed at the Fraunhofer ISST. With the help of the Dataspace Connector, existing software can easily be extended by IDS connector functionalities in order to integrate them into an IDS data ecosystem. Furthermore, it is possible to use the Dataspace Connector as a basis for the development of own software that is to be connected to an IDS data ecosystem. The Dataspace Connector uses the recent IDS Information Model version and the IDS Messaging Services for message handling with other IDS components. For managing datasets by means of their metadata as IDS resources, the Dataspace Connector provides a REST API. After an initial registration, IDS resources are persisted to an internal or external database of the connector. External data sources can be connected via REST endpoints, allowing the Dataspace Connector to act as an intermediary between the IDS data ecosystem and the actual data source.

5.5 Data Catalog and Lineage

The other modules of the Dataspace, such as the storage layer, the processing units, the model execution engine and the external connectors, communicate with each other through Kafka. A benefit from applying this topology is that it is possible to monitor and record the consumption point of produced data on Kafka. It is also possible to restrict or allow the distribution of specific information according to rules. The operation is facilitated by the existence of a central data catalog, listing the data sources along with their metadata.

Data catalog and lineage are important concepts in managing and understanding large Dataspaces. A data catalog is a system that keeps track of all the data assets within a Dataspace, including their location, format, and metadata. This allows users to easily discover, understand, and use the data within the Dataspace. Data lineage, on the other hand, refers to the history of a particular data asset, including where it came from, how it was transformed, and where it is used. This information is crucial for understanding the provenance and trustworthiness of a particular piece of data, as well as for identifying and troubleshooting issues that may arise.

With a comprehensive data catalog, users can easily find and access the data they need, while data lineage allows them to understand how that data was created and how it has been used. Together, these two concepts help to ensure that data is accurate, relevant, and trustworthy.

[28] https://sovity.de

Various technologies are available for the implementation. Some popular options include Apache Atlas[29], Alation[30], Collibra[31], and Informatica MDM[32]. These tools provide a variety of features, such as data discovery, data lineage visualization, and metadata management, that can help organizations effectively manage and understand their Dataspaces.

For these purposes it is examined the utilization of Apache Atlas as the DT4GS data governance tool. Apache Atlas provides open metadata management and governance capabilities for systems to build a catalog of their data assets, classify and govern these assets and provide collaboration capabilities around these data assets for data scientists, analysts and the data governance team.

The metadata repository is built on top of a NoSQL database, such as Apache HBase[33] or Apache Cassandra[34], which allows for scalability and high performance.

Atlas also includes a set of RESTful APIs that can be used to access and manage the metadata stored in the repository. These APIs can be used to create, read, update, and delete metadata, as well as to perform advanced searches and queries on the metadata itself.

In addition to the core metadata repository, Apache Atlas also includes a number of other components that provide additional functionality. For example, it includes a data discovery and lineage component, which allows users to track the lineage of data assets through various stages of the data pipeline. This allows organizations to understand how data is used and transformed in their systems, and to identify potential issues or errors.

Apache Atlas also provides a UI that allows user to easily navigate and interact with the metadata in the repository. The UI allows users to view metadata, perform searches, and create and manage policies, among other things.

5.6 The Platform Monitoring Interface

The data ingestion to the Kafka topics will be visualized with a platform (e.g., Grafana) so that the data rate, volumetrics and availability can be precisely defined and monitored. Moreover, the incoming real-world data can be used in conjunction with historical data for the simulations but also as a validation measure on the output of a simulation, provided that the models allow for such input. Furthermore, specific topics can pass information to the dashboard together with the scheduler API so that the user is updated on the status of the simulations that are being executed. In addition, information regarding the simulation execution history and in cases output data will be stored in Kafka topics. For example, it is possible to create a line chart that shows the number of messages per second that are being processed by a Kafka topic, or a bar chart that shows the number of messages per topic over time. The user can also create tables that show the number of messages per topic, partition, or consumer group.

Finally, the machine's resources along with metrics from the Kafka subsystem and the storage infrastructure are monitored through Prometheus exporters. The collected data can then be visualized with dedicated dashboards (Grafana, Prometheus) in a variety of different graphs and with alerting functionality built in.

[29] https://atlas.apache.org
[30] Learn About Alation - The Leader in Data Intelligence | Alation
[31] Collibra: Data Catalog, Data Governance & Data Quality | Collibra
[32] Master Data Management (MDM) Solutions and Tools | Informatica UK
[33] https://hbase.apache.org/
[34] https://cassandra.apache.org/

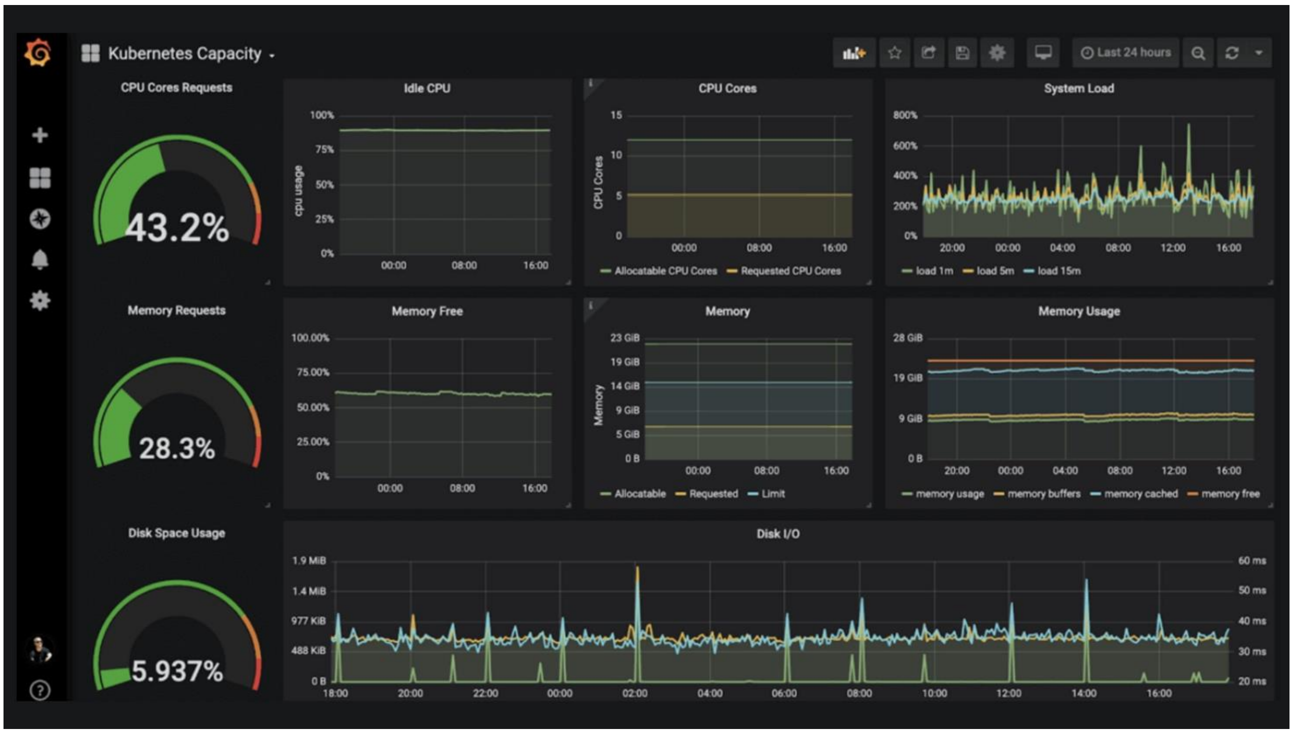

Figure 10 Example of Grafana Dashboard to monitor a Kubernetes cluster

The dashboard, shown as an example in Figure 10, is customizable so it can be leveraged to create a UI to integrate in the DT4GS Dataspace as a tool to allow the user to keep control of every meaningful aspect to track in the DT4GS ecosystem.

5.7 Security Considerations

5.7.1 Device authentication and encryption

In view of the forthcoming regulation concerning ships' emissions, the application of a blockchain technology can provide the means for an autonomous, reliable, and trustful verification system to report environmental indexes to the authorities. However, the implementation of such a system possibly requires a decentralized topology, i.e. positioning of the blockchain intelligence closed to the data source, possibly on the internal connector instead of the central connector (Figure 11). In this case the central connector just passes through the communication.

The solution, defined in GA and currently being studied to evaluate its application based on the inherent deployment needs of the DT4GS infrastructure, is the integration of CHARIoT's blockchain-based PKI for sensor/gateway authentication and blockchain-assisted encryption of IoT endpoints.

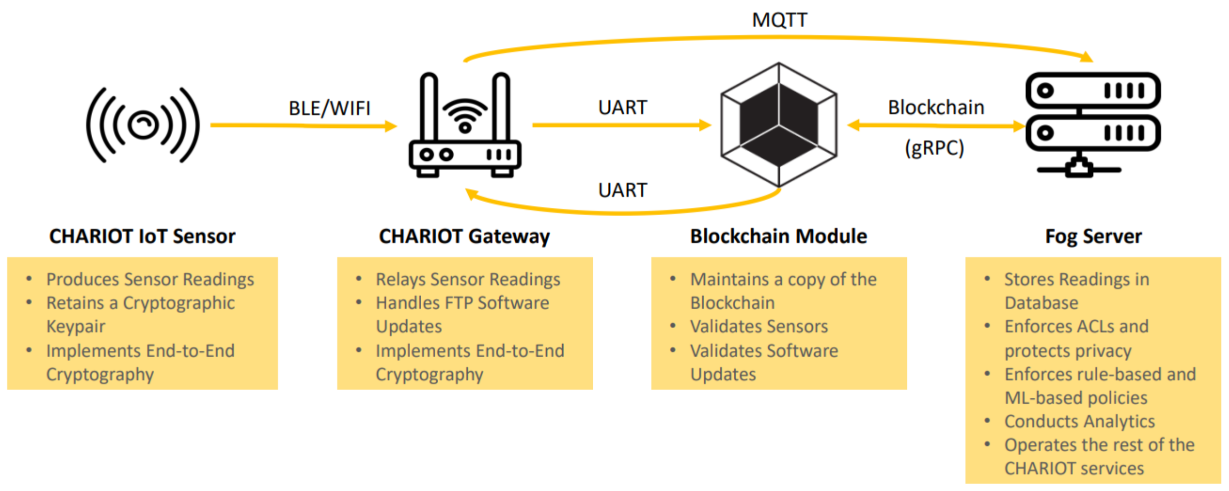
MQTT
BLE/WIFI
UART
UART
Blockchain
(gRPC)
CHARIOT IoT Sensor
Produces Sensor Readings
Retains a Cryptographic Keypair
Implements End-to-End Cryptography
CHARIOT Gateway
Relays Sensor Readings
Handles FTP Software Updates
Implements End-to-End Cryptography
Blockchain Module
Maintains a copy of the Blockchain
Validates Sensors
Validates Software Updates
Fog Server
Stores Readings in Database
Enforces ACLs and protects privacy
Enforces rule-based and ML-based policies
Conducts Analytics
Operates the rest of the CHARIOT services

Figure 11 Example of sensor authentication and data exchange architecture with CHARIOT[35]

Below are listed some of the CHARIoT blockchain-related functionalities available[36]:

- End-to-end network solution with advanced capabilities for devices (sensors, gateways, etc.) authentication (via keys' embedding) combined with blockchain and encryption technologies.
- Combined authentication solution of blockchain with PKI (Public Key Infrastructure) technologies in the actual network devices (sensors and gateways).
- Blockchain-aided encryption between all IoT network endpoints (sensor/gateway/FOG).
- Mobile application for sensor provisioning in the IoT network utilizing the four-eye principle.
- Blockchain-based state management for sensors (decommissioned, faulty, compromised etc.).
- CHARIOT sensors (WiFi & Bluetooth) with high processing capabilities (supporting encryption, blockchain etc.)

Given the heterogeneous IoT infrastructure already in place on the ships involved in the DT4GS Living Labs (discussed in Section 3.3), applicability of the CHARIOT's solution is under investigation, therefore a final implementation, at the time that the present report is compiled, is not yet conceived.

From the perspective of the central connector, it is required to ensure authentication of the connected subsystems, especially the internal and external connectors, in order to avoid system manipulation by providing artificial inputs.

Authentication can also secure system from erroneous inputs that may result to exhaustion of the available resources (similar to DDOS symptoms).

Apart from protecting system from artificial or false inputs, care is taken to secure the data communication from unauthorized access. The means to succeed this task is to encrypt the communication.

[35] https://www.chariotproject.eu
[36] https://www.chariotproject.eu/wp-content/uploads/2020/11/WS3_Blockchain-as-an-enabler-for-IoT-Data-Security-Safety-and-Privacy.pdf

Both encryption and authentication are foreseen for any communication between the different building blocks of the system.

The current version of Kafka includes the following features[i] that, that can be used either separately or together, increasing security in a Kafka cluster:

1. Authentication of connections to brokers from clients (producers and consumers), other brokers and tools, using either SSL or SASL.
2. Authentication of connections from brokers to ZooKeeper
3. Encryption of data transferred between brokers and clients, between brokers, or between brokers and tools using SSL
4. Authorization of read / write operations by clients
5. Authorization is pluggable and integration with external authorization services is supported

Application of these security measures is optional - non-secured clusters are supported, as well as a mix of authenticated, unauthenticated, encrypted and non-encrypted clients.

As per the actual direction evaluated by the DT4GS partners, as already stated in this section, internal and external connector must be implemented such as to guarantee for a secure authentication while exchanging data towards the central connector node. Given the data exchanged between ship and shore might be sensitive, encryption capabilities will be granted to all the Kafka nodes compromising the DS. These security features will be implemented, as a first iteration, using the Kafka features described above.

5.7.2 Sensor and Gateway Authentication

In view of the forthcoming regulation concerning ships' emissions, the application of a blockchain technology can provide the means for an autonomous, reliable, and trustful verification system to report environmental indexes to the authorities. However, the implementation of such a system possibly requires a decentralized topology, i.e. positioning of the blockchain intelligence closed to the data source, possibly on the internal connector instead of the central connector. In this case the central connector just passes through the communication. At the time that the present report is compiled, the matter is under investigation, but a solution is not yet conceived.

5.8 Deployment

A goal of the project is to allow end users to easily implement and deploy their own version of the DT4GS infrastructure. The technology evaluated as the best for our framework is Kubernetes (htps://Kubernetes.io).

Kubernetes is an open-source container orchestration system that automates the deployment, scaling, and management of containerized applications. It is widely used in organizations of all sizes, across various industries, to manage their containerized workloads.

One of the main benefits of Kubernetes is its ability to automate and script the deployment of applications, which can greatly reduce the time and effort required to deploy and manage applications. This can be especially beneficial for organizations that have complex and dynamic workloads, or that need to deploy and manage applications across multiple environments. These scripts are managed by a version control system and are part of a Continuous Integration and Continuous Deployment pipeline (CI/CD). This pipeline will monitor the code repositories of the various components and any changes will trigger re-testing and operator instructed redeployment of the platform.

Grafana dashboards provide also monitoring for the Kubernetes infrastructure.

To explain the choice of Kubernetes as a technology for deployment, the software and its operation are briefly presented below.

A Kubernetes deployment typically includes several components and tools, including:

- Kubernetes cluster: A set of machines that run the Kubernetes control plane and worker nodes. The control plane manages the overall state of the cluster, while the worker nodes run the containerized applications.
- Container runtime: The software that is used to run the containerized applications. This can be Docker, containerd, or another container runtime.
- Kubernetes API: The API that is used to interact with the Kubernetes cluster. This is typically accessed via the Kubernetes command-line interface (kubectl) or a Kubernetes client library.
- Kubernetes manifests: YAML files that define the desired state of the deployment, including the containerized applications and the resources that they require.
- kubectl: The Kubernetes command-line interface that is used to interact with the Kubernetes API. It can be used to create, update, and delete resources, as well as to view the current state of the cluster.

5.9 Data storage

As introduced in section Data aspects to be considered, our Dataspace processes heterogeneous types of data and thus it is required for it to be able to store all these information in an effective format to allow fast data retrieving and processing.

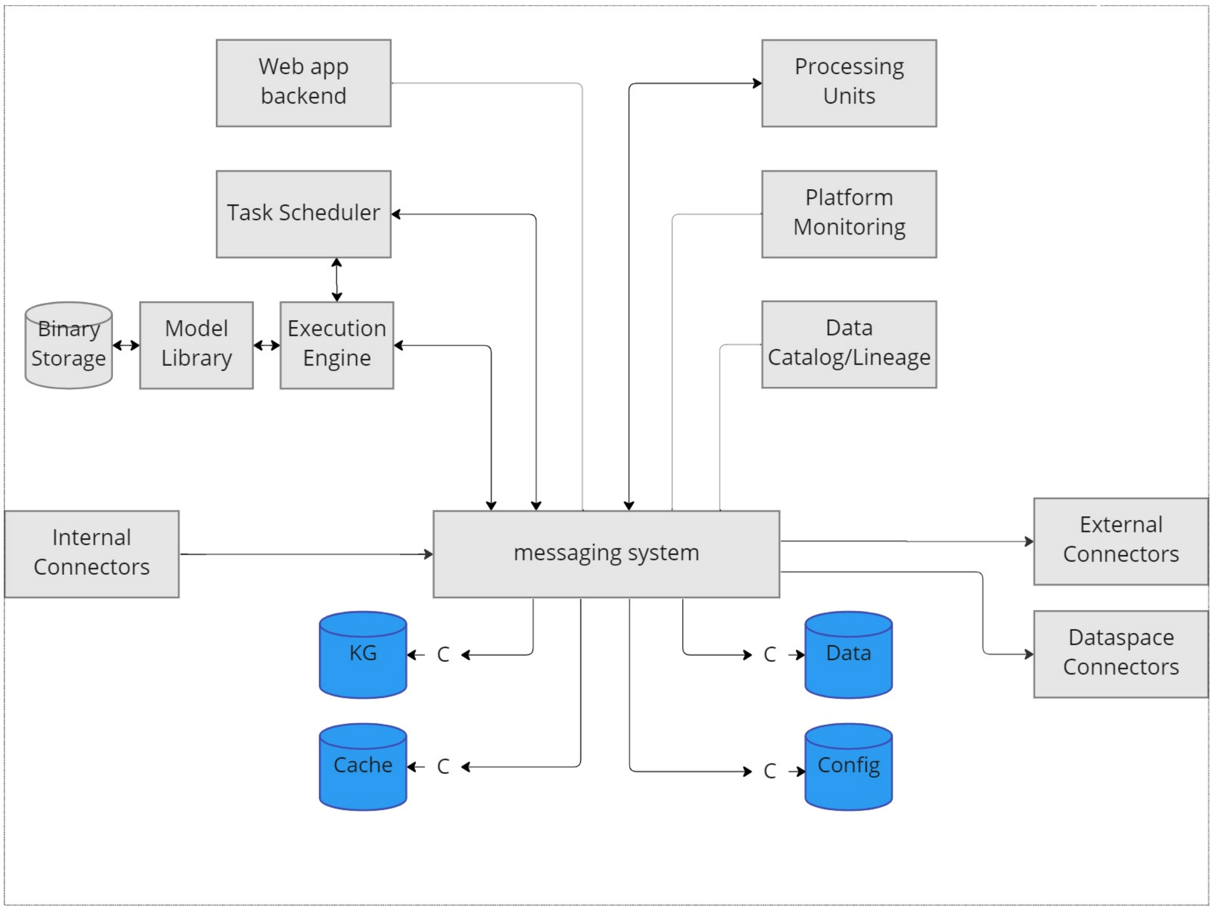

Figure 12 Architectural diagram of the DT4GS infrastructure with highlighted the components of the data storage.

The chosen solutions for data storing and processing are presented in this document in the chapter Evaluation of data processing infrastructures and selection criteria.

5.10 Internal Connectors

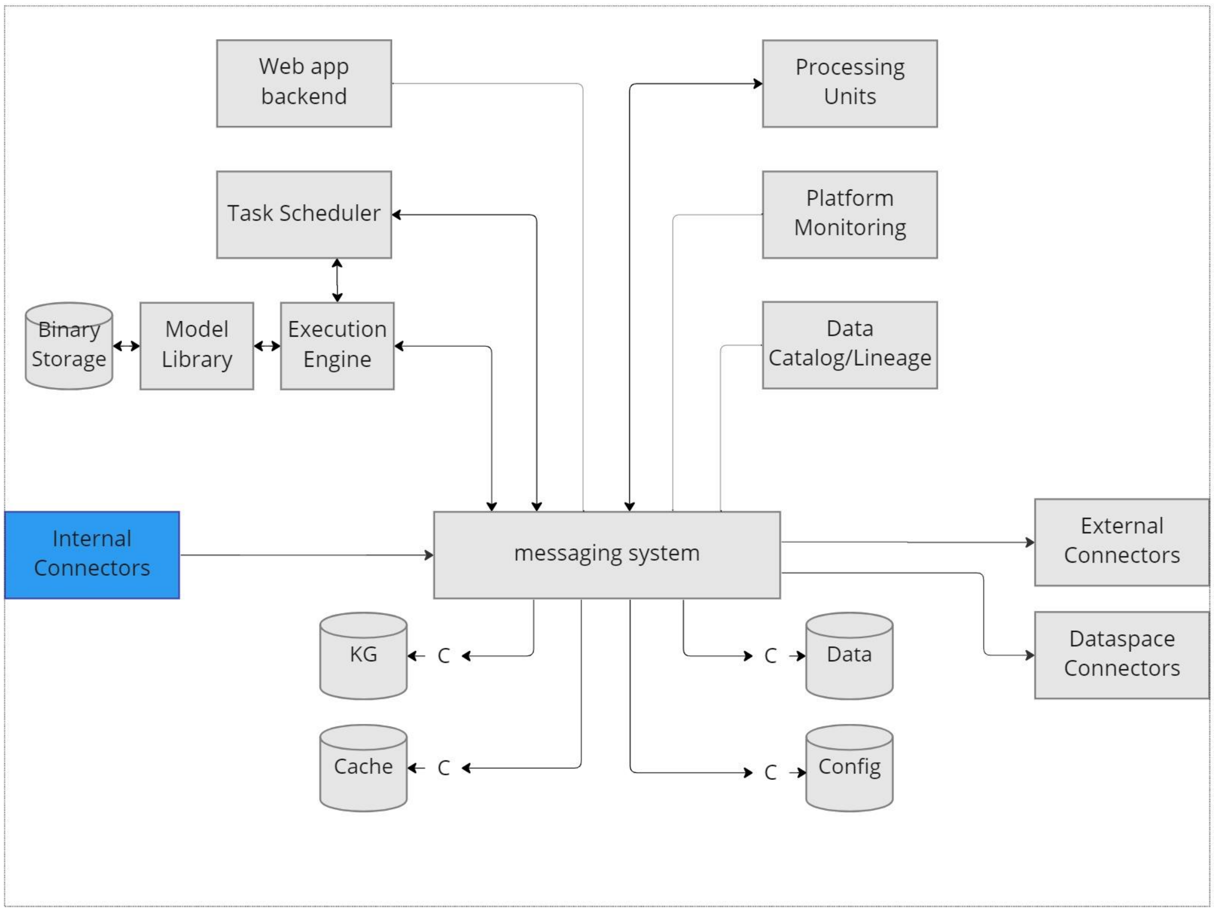

Figure 13 Architectural diagram of the DT4GS infrastructure with highlighted the components of the Internal Connectors.

Given the actual sensor infrastructure already installed on board of the LL ships, the Internal connectors, presented, will be customized in order to ensure the digital twin has the capabilities to interface with the ship's information system, if present, or with the signal management system from the sensors behind the on-board real-time monitoring system as in Starbulk's ship case. This approach will allow the Digital Twin to include the whole hardware sensor infrastructure of the ships into the Dataspace making it a common IoT infrastructure capable of connecting devices on board and ashore, enabling users involved in the digital twin to monitor in real time the status of the ship.

The Internal connectors, shown in Figure 13 Architectural diagram of the DT4GS infrastructure with highlighted the components of the Internal Connectors.are the components of the DT4GS infrastructure located on the ship and dedicated to the gathering of the IoT sensor data and consequently to the transmission of data from the ship itself to the ship-owner’s headquarters, where the central connector node and the Dataspace structure are located. Consequently, the minimum required capabilities for this component are to be able to adapt to the ship infrastructure and collect data from its informative system or IoT structure, organize them in a structured way and consequently provide them to the DT4GS

Dataspace for storage.For an easy, automated deployment of the solution the Internal Connector will be provided as a Docker Container[37].

5.11 Internal Connectors integration with the Central Connector Module

As described in this document, all the main components of the DT4GS system will be enabled as Apache Kafka consumers/producers to allow the messaging system to connect. The internal connectors are no exception to that rule.

5.12 Blockchain Enabled Connectivity

Blockchain technology has recently gained a lot of attention as a way to provide secure, decentralized solutions for a variety of applications. One area where blockchain technology has the potential to make a significant impact is in the field of IoT connectivity, granting the security and integrity of the data being transmitted. Blockchain technology provides a solution for this challenge by allowing for the creation of decentralized networks where each node has a copy of the blockchain, providing a tamper-proof record of all transactions. By incorporating blockchain technology into an Apache Kafka-based IoT connectivity solution, it is possible to ensure that the data being transmitted is secure and cannot be tampered with. One way to implement this functionality is by using a blockchain-based distributed ledger to store the metadata of the messages transmitted in the Kafka cluster. This metadata would include information such as the timestamp, the sender, and the recipient of the message. By storing this information on a blockchain, the system can ensure that it is tamper-proof and can be used to verify the authenticity of the data being transmitted. Another approach is to use smart contract to validate the authenticity of the devices that are sending data to the Kafka cluster.

As already stated in section 5.7.1 this approach requires a decentralized topology, i.e., positioning of the blockchain intelligence closed to the data source, possibly on the internal connector instead of the central connector, and is still investigated as by the DT4GS partners as a possible implementation choice.

5.13 Data Streaming Modalities

The Internal Connector in DT4GS are implemented as Kafka producers. The Kafka producer is conceptually much simpler than the consumer, as it does not need group coordination. A producer partitioner maps each message to a topic partition and then sends a production request to the leader of that partition. The partitioners supplied with Kafka ensure that all messages with the same non-empty key are sent to the same partition.

5.14 On board data cleaning

In order to clean up the acquired data to be transmitted to the Central Connector Node, it could be evaluated to provide some data pre-processing and cleaning capabilities to the Internal Connectors. At this stage of the project, this solution is in getting discussed by the DT4GS partners. To achieve this result a local instance of InfluxDB must be instantiated in the Internal Connector Node.

The data will be saved in the local Influxdb instance, and some data cleaning routines will be performed on them, including outlier detection, as descripted in Section 6, and potentially data validation with a

[37] https://docker.com

fixed threshold of acceptability of the sensor value. This approach could eventually allow to lighten the burden of information exchanged between ship and shore by ensuring that only useful data is sent on the relevant topic. There are different ways to stream data from InfluxDB to an Apache Kafka topic, but one possible approach is to use a Kafka Connect sink connector for InfluxDB. The data stream can be triggered using the Kafka Connect command-line tool but can also be initiated and managed automatically by a service.

5.15 External Connectors

As already stated in section 1, the actual implementation and operation of a ship Digital Twin requires additional data that are not directly collectable from the ship system via Internal Connectors. Data like weather data, port traffic data, fuel availability/prices, etc might be accessible via other data sources and accessible in alternative ways like via APIs. For this reason, the DS needs to be augmented with external connectors, able to digest those external data and consequently enrich them with the required metadata before making them available to the Digital Twin ecosystem through the Central Connector Node. An interface with HQ ERP, PMS and other system might be needed too.

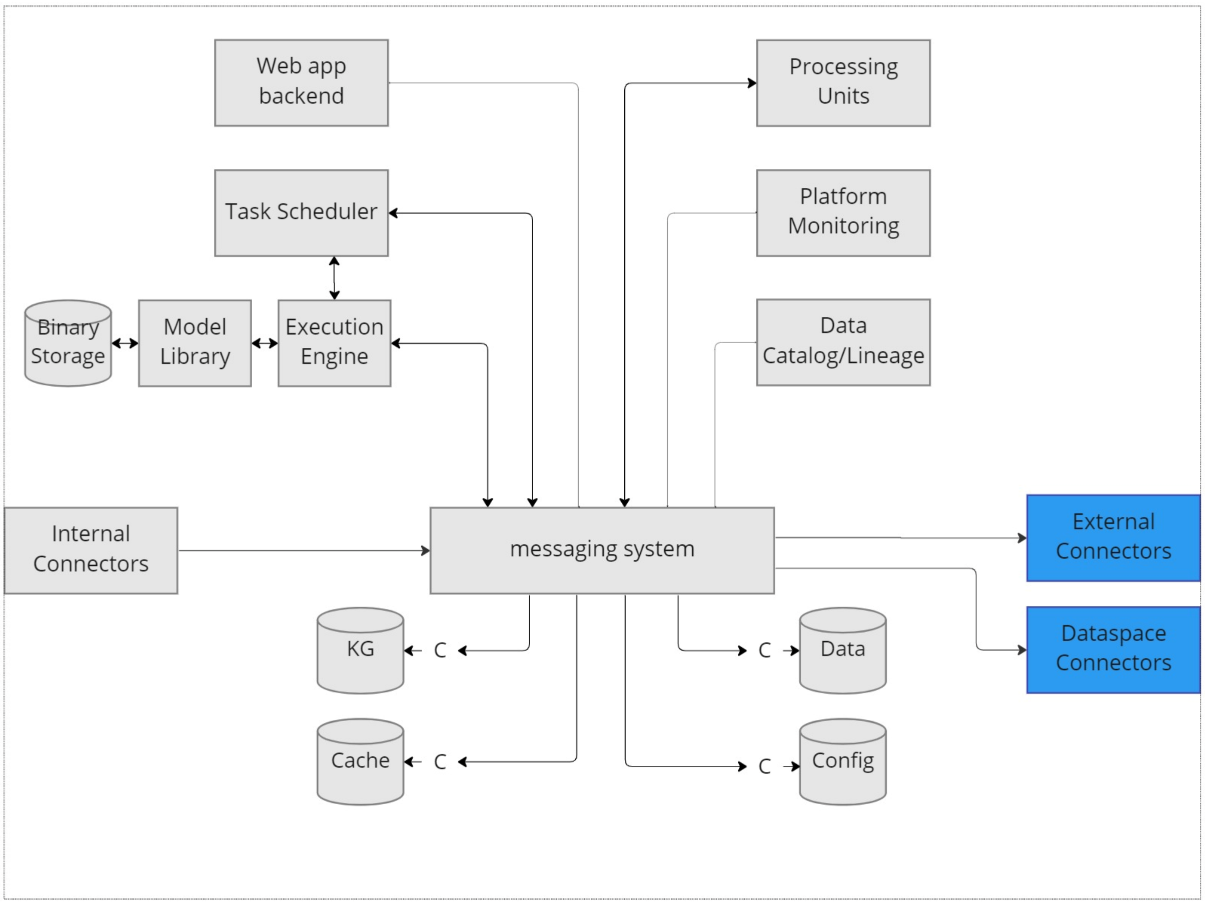

Figure 14 Architectural diagram of the DT4GS infrastructure with highlighted the components of the external connector modules.

The data linking between the external sources and the DT4GS DS is implemented via Apache Kafka as per the other data sources, to enable compatibility also in this context. As per the Internal Connectors the applications are built leveraging Docker creating standalone containers to allow DT4GS users to easily deploy the application.

In Figure 14 Architectural diagram of the DT4GS infrastructure with highlighted the components of the external connector modules.are highlighted the components of the Dataspace to be considered as External Connectors. The first block named External Connector represents generally every API and application built to allow retrieving data from external sources out of the DS itself. The other highlighted block, Dataspace Connectors, represents the actual implementation of the connectors between different Dataspaces as per the IDS Dataspace Standards.

5.16 External data

In this sub-chapter are presented some of the external data that might be useful to be leveraged in the DT4GS project. The main and more interesting external information could be classified as:

- Weather data (for performance assessment).
- Weather data (for routing).
- Fuel availability/price.
- Headquarters systems such as ERP, PMS, etc.

In particular, as a matter of example, we cite the most important environmental conditions that could affect the choice of the optimal ship's route. This data has been found to some of the most important weather sources, such as World Weather Online[38], Copernicus[39], Meteoblue[40] and NOAA[41]. It is important to mention that in some platforms, this data is explicitly available, whereas in others, subscription may be needed. At the time that the present report is compiled, the external sources are evaluated to assess which ones could represent an added value for the project, enabling the models developed within the project to more faithfully simulate the real behaviour of the ship. The choice will be made in discussion with the consortium partners. In the following part of this document, the weather data is organized per source and some tables are provided to summarize the available information.

5.16.1 World Weather Online

The Premium Historical Marine Weather REST API method allows you to access marine data for a given longitude and latitude, as well as tidal data. The Historical Marine Weather API returns weather elements such as temperature, precipitation (rainfall), weather description, weather icon, wind speed, Tidal data, significant wave height, swell height, swell direction and swell period. Provides up to 7 days of forecast. The data provided from the World Weather Online, are presented in the *Table 5* below:

Table 5 World Weather Online API available data

Weather element	
1	**Date**
2	**Maximum Temperature of the day in degree Celcius**

[38] World Weather Forecasts | World Weather Online
[39] Homepage | Copernicus
[40] https://www.meteoblue.com/
[41] Homepage | National Oceanic and Atmospheric Administration (noaa.gov)

3	**Maximum Temperature of the day in degree Fahrenheit**
4	**Minimum Temperature of the day in degree Celcius**
5	**Minimum Temperature of the day in degree Fahrenheit**
Astronomy element	
1	**Local sunrise time**
2	**Local sunset time**
3	**Local moonrise time**
4	**Local moonset time**
5	**Moon phase**
6	**Moon illumination**
Hourly element	
1	**Local forecast time**
2	**Temperature in degree Celcius**
3	**Temperature in degree Fahrenheit**
4	**Wind speed in miles per hour**
5	**Wind speed in kilometres per hour**
6	**Wind direction in degrees**
7	**Wind direction in 16-point compass**
8	**Weather condition code**
9	**Weather condition description**
10	**URL to weather icon**
11	**Precipitation in millimetres**
12	**Humidity in percentage**
13	**Visibility in kilometres**
14	**Atmospheric pressure in millibars**
15	**Cloud cover in percentage**
16	**Significant wave height in metres**
17	**Swell wave height in metres**
18	**Swell wave height in feet**
19	**Swell direction in degree**
20	**Swell direction in 16-point compass**
21	**Swell period in seconds**
22	**Water temperature in Celcius**
23	**Water temperature in Fahrenheit**
24	**UV index**
Tides element	

1	**Local tide time**
2	**Tide height in meter**
3	**Type of tide, i.e, high, low or normal**

5.16.2 Meteoblue Packages API

Meteoblue offers numerous weather variables which are grouped into history and forecast packages. There are different packages for specific use cases like agriculture, renewable energy and many others. The data packages are available for different historical and forecasting time-ranges in the CSV and JSON formats. The data provided from the Meteoblue Packages API, are presented in the Table 6 below:

Table 6 Meteoblue Packages API available data

Precipitation	
1	**Accumulated Precipitation**
2	**Daily Precipitation**
3	**Hourly Precipitation**
4	**Accumulated Snowfall**
5	**Daily Snowfall**
6	**Hourly Snowfall**
Sea	
1	**Wave Height (significant, swell, wind)**
2	**Wave Period (swell, wind)**
3	**Wave peak period (swell, wind)**
4	**Ocean currents**
5	**Sea water salinity**
6	**Sea ice cover**
7	**Water surface temperature**

5.16.3 National Oceanic and Atmospheric Administration API

NCDC's Climate Data Online (CDO) offers web services that provide access to current data. This API is for developers looking to create their own scripts or programs that use the CDO database of weather and climate data. An access token is required to use the API, and each token will be limited to five requests per second and 10,000 requests per day. The data provided from the CDO database, are presented in the Table 7 below:

Table 7 CDO database available data

Real time data

1	**Water Level**
2	**Air temperature**
3	**Water Temperature**
4	**Barometric pressure**
5	**Winds**
6	**Relative humidity**
7	**Visibility**
8	**Next tide (when and if high or low) ~ every 6h**
Currents	
1	**Currents velocity (some meters below the surface)**
Wave buoys (ports)	
1	**Significant wave height**
2	**Peak direction**
3	**Wave period**
4	**Water temperature**

5.17 External Connectors integration with the Central Connector Module

As stated in the previous paragraph, external information is usually available via API calls. Given that assumption, it is possible to identify a lot of implementation parallels between the Internal and the External connectors.

6 Analytics Applications

The sensor infrastructure on a ship includes a variety of sensors that are used to monitor various systems and conditions on board the ship. These sensors include engine sensors, which monitor the performance, temperature, pressure, and vibration of the ship's engines, navigation sensors, such as GPS and gyroscopes, which track the ship's position, heading, and speed, and environmental sensors, such as temperature, humidity, and air quality sensors, which monitor the conditions on board. Additionally, there are cargo sensors that monitor the weight, temperature, and condition of the ship's cargo, weather sensors, such as wind, rain, and wave sensors, that gather data on the ship's surroundings, and ballast and bilge sensors that detect any changes in the ship's water levels to ensure stability. Other sensors, such as fire and gas detectors, are included for safety purposes, and deck and hull stress sensors monitor the ship's structure and detect any signs of stress or damage. These sensors work together to gather data, which is then transmitted to edge devices or directly to the cloud for analysis and action.

The information collected on the different ships made available by the LL partners to be used as a test bed presents many common data but at the same time there are differences to be managed. In the context of DT4GS, the data of interest are vessel data that arrive in the form of sensor observations. These observations are captured by onboard sensor technologies and may measure quantities such as velocity, fuel consumption, acceleration, engine temperature, etc. Since these observations are ordered based on time and arrive at fixed time intervals *t*, a straightforward way to model them is through time series data. Since multiple sensor updates are received at each timestamp, the produced time series are characterized as multivariate.

A first approach to the whole data collected onboard must be to select from the available pool of monitored signals those useful for processing within the Digital Twin. One of the goals of the project at this stage is to provide a common interface between ship and shore, both from the communication standpoint but also relatively to the actual sensors' information collected for the DT's operational needs. While many common aspects, as expected, can be found in the IoT structure for all the ships in the LL, there are differences in the way data is collected and consequently in the way our DT can interface with them for data retrieving.

The table reported in in annex 1 of the DT4GS deliverable D1.1 produced in the scope of the DT4GS project, which at the time of writing this document has already been published by the project consortium, presents the information on available sensors gathered by the partners in the work of task 1.1.

While approaching the collected data, it is also expected that there will be incorrect, corrupted, or incomplete data points within the captured observations; hence, a data cleaning process is typically necessary prior to any analysis. Standard practices at the minimum perform the detection and removal of erroneous values, as well as missing values imputation. Furthermore, depending on the type of analysis we wish to perform, there may be further preprocessing steps required, such as normalization or dimensionality reduction. These steps could be partially performed on board before sending the data to shore, or handled completely in the land offices, easing the hardware requirements of the machines installed on board. This aspect is one of the topics being evaluated with the Living Labs partners.

Once the data preprocessing procedure is completed, the data should be in a form appropriate to be used as input to various mathematical models (e.g., machine learning models) that will detect potential irregularities, extract valuable patterns or predict future behaviours. These computational steps must be processed before time series data are made available for the simulation tasks underlying the operation of the DT4GS digital twin.

The IoT Infrastructure is not compromised only by the actual sensors installed on-board but also by the edge devices, such as gateway computers, a communication system and user interfaces, such as dashboards, that visualize the data and allow for control of the ship systems.

Event streaming platforms play a central role in ingesting, storing, and processing real-time data (e.g. sensor observations) in a scalable and resilient manner. In our case, an event (also record or message) is a notification that "something happened" in the vessel, i.e., an update on the state of the ship. An event stream is a continuous unbounded series of such events. The start of the stream may have occurred before we started processing the stream. The end of the stream is at some unknown point in the future, i.e when the vessel reaches a port. Each event in a stream carries a timestamp that denotes the point when the event occurred. Events are ordered based on this timestamp. In general, the definition of time series data aligns with that of event streams.

Typically, these systems function according to the popular producer-consumer paradigm. Simply put, in the core of this paradigm there are 2 (computer) processes, one that sends events (or messages), also called the producer, and one that receives events (or messages), also called the consumer.

The architecture just described fits perfectly in the Apache Kafka producer-consumer communication paradigm, as already described in section 4.3 validating the choice of this technology in the scope of this project.

6.1 Localized Data Capture and processing

Scalability and the capacity to quickly consume data are the main database requirements for IoT apps. As previously stated, NoSQL systems are ideal for IoT since they are designed with significant horizontal scalability. Some other popular characteristic of NoSQL databases is the effective use of data storage in memory, which would be highly useful for writing throughput and latency. RDBMS systems appear to be incomplete because they were not intended to process the quantity of data, or the rate produced by the data.

For the data ingestion in the form of time series data the choice for the DT4GS DB implementation is InfluxDB, presented in Section 4.2.

6.2 Time series-processing and Analysis on Sensors Data

In this Section we are going to examine anomaly detection as a technique to preprocess data before storing it in the DT4GS Dataspace for further utilisation.

The SoTA techniques briefly described in this chapter are those that are being evaluated by the partners while analyzing the available historical high frequency data made available by the LL partners. In addition, given that a lot of methods employed in these two areas make use of machine learning models, we deem it necessary to understand both how and why a model makes a certain prediction.

Thus, a summary of approaches for explaining machine learning models is presented last.

6.2.1 Anomaly Detection

An anomaly can be thought of as an unexpected change in the state of a system, which is outside of its local or global norm. Pang (2021) divides time series anomalies into three types, in particular point, contextual, and collective anomalies. We briefly outline them below:

Point anomalies refer to data points that deviate remarkably from the rest of the data. Point anomalies usually make up for a small percentage of the total data observations. The affected system after experiencing these short anomaly intervals returns to its previous normal state. Point anomalies can also represent statistical noise or noise produced by faulty sensing equipment.

Contextual anomalies refer to data points within the expected range of the distribution, but which deviate from the expected data distribution, given a specific context. Contextual anomalies if taken in isolation may be within the range of expected values, but when taken in the context of the surrounding observations constitute anomalies.

Collective anomalies refer to sequences of points that do not repeat a typical previously observed pattern. Individual observations within a collective anomaly may or may not be anomalous, it is only when they appear as a group that they arouse suspicion.

The first two categories, namely, point and contextual anomalies, are referred to as point-based anomalies, whereas collective anomalies are referred to as subsequence anomalies.

Anomaly detection problems are supposed, by definition, to handle and process unpredictable rare events characterized by many unknown factors such as their structures, distributions and irregularities. Anomalies are typically rare data instances, contrasting to normal instances that often account for an overwhelming proportion of the data.

Schmidl (2022) studies 71 algorithms, categorizing them according to various criteria. One viable categorization is by considering the method family into which each algorithm can be grouped. These method families characterize the algorithms by their general approach of determining the abnormality of specific points or subsequences within the time series. The resulting six method families are summarized below:

- *Forecasting methods:* use a continuously learned model to forecast a number of time steps based on a current context window. The values for the forecasted data points depend solely on the time series' data points in the preceding context window and the previously learned model. The forecasted points are then compared to the observed values in the original time series to determine how anomalous the observed values are.
- *Reconstruction methods:* build a model of normal behavior by encoding subsequences of a normal training time series in a low-dimensional latent space. To detect anomalies in a test time series, subsequences from the test series are reconstructed from the latent space, and the reconstructed subsequences' values are then compared to the original, observed series values.
- *Encoding methods:* are similar to reconstruction methods in that they also encode subsequences of a time series in a low-dimensional latent space. However, they do not attempt to reconstruct the subsequences from the latent space but compute the anomaly score directly from the latent space representations.
- *Distance methods:* use specialized distance metrics to compare points or subsequences of a time series with each other. Anomalous subsequences are expected to have larger distances to other

subsequences than subsequences with normal behavior. Cluster-based distance methods cluster similar subsequences together and then compute the distances to dense areas.

- *Distribution methods:* fit a distribution model to the data. Anomalies are detected using frequency rather than distance. The anomaly scores are usually measured using probabilities and likelihoods or subsequences with respect to the prior calculated distributions.
- *Isolation tree methods:* build an ensemble of random trees that partition the sample of the test time series. For the tree construction, the methods recursively select random features and random split values as tree nodes to eventually isolate the samples in the tree leaves. The number of splits required to isolate a sample is a measure described by the average path length over all random trees in the ensemble. Because anomalous samples are easier to separate than normal samples, they are on average closer to the tree root and have noticeably shorter paths. For this reason, path lengths are characteristic of the normality of samples and, hence, their reciprocal value translates into anomaly scores.

Anomaly detection algorithms can also be classified by their learning type, i.e unsupervised, supervised, and semi-supervised. However, both Pang (2021) and Schmidl (2022) point out that supervised learning approaches are quite unpopular in practice and thus the majority of techniques developed in anomaly detection research is based mainly on semi-supervised and purely unsupervised settings and methods.

Unsupervised Methods do not require a set of labeled samples to fit the parameters of a particular machine-learning model. There are multiple algorithms proposed to approach anomaly detection in an unsupervised fashion. Below are presented some, that are actually evaluated by the partners.

- *SAND* (Streaming Subsequence Anomaly Detection), proposed by Boniol (2021). SAND supports real-time analytics, as the algorithm is online and does not require access to the entire dataset. Anomalies are detected based on their distance to a data structure that represents normal behavior. Moreover, SAND does not require domain knowledge and thus can detect domain-agnostic anomalies. Finally, the algorithm is able to adapt to distribution drifts, i.e., to changes in the data generation process.
- *Convolution Ensembles*: Another state-of-the-art unsupervised anomaly detection method, proposed by Campos (2022). The ensemble employs multiple basic anomaly detection models built on convolutional sequence-to-sequence autoencoders. An autoencoder consists of an encoding phase that compresses a time series T into a compact representation and a decoding phase that reconstructs an output time series T' from this representation. The representation is only able to capture patterns that reflect normal behavior in the original time series and not anomalies. The difference between observations in T and in T' is called the reconstruction error. Intuitively, the higher the reconstruction error which means less likely to be from the input distribution, the higher the anomaly score. A threshold can be set to discriminate anomaly from normality.
- Anomaly detection based on **transformers** has been introduced for example by Tuli (2022). Transformers proved to be more efficient than LSTMs, mostly due to parallel computing in their architecture. Tuli (2022) proposed to combine transformers with neural generative models for better reconstruction models in anomaly detection. In particular, they propose an adversarial training procedure to amplify reconstruction errors. GAN adversarial training is processed by two transformer encoders and two transformer decoders to gain stability. A review of the application

of Transformers to important time series tasks, including forecasting, anomaly detection, and classification, is presented by Wen (2022).

6.3 Forecasting

Machine Learning Architectures for Time Series Forecasting are particularly suitable for finding the appropriate complex nonlinear mathematical function to turn an input into an output. Hence, machine learning models provide a means to learn temporal dynamics in a purely data-driven manner. Here are briefly presented some possible approaches to the Time Series Forecasting task:

- *Artificial Neural Networks* (ANNs) can be employed for nonlinear processes that have an unknown functional relationship and as a result, are difficult to fit. The main concept with ANNs is that inputs are passed through one or more hidden layers each of which consists of hidden units, or nodes before they reach the output variable.

In time series forecasting, as in other domains, the emergence of competing neural network architectures has relegated simple ANNs to the background.

- *Convolutional Neural Networks* (CNNs) are a specific kind of deep neural networks, proposed for and mostly dedicated to image analysis and aimed at preserving spatial relationships in the data, with very few connections between the layers. Having connections from all nodes of one layer to all nodes in the subsequent layer, as in regular ANNs, proved extremely inefficient and thus CNNs arose from the observation that a careful pruning of the connections, based on domain knowledge, boosts performance.
- *Recurrent Neural Networks* (RNNs) were designed to handle sequential information as stated in Lipton (2015). Typically, RNNs are capable of making predictions over many time steps, in time series. An RNN achieves the same task at each step (with varying inputs): the sequence (x1, x2 , · · · , xt, xt+1, · · ·) is input to the RNN, element by element (one step at a time). An RNN performs the same task for each element of a sequence, with the output being dependent on the previous computations. The weights are context-dependent and dynamically updated for each time step.
- *Long short-term memory networks* (LSTMs) are the most widely-used subclass of RNNs, as they perform better than RNNS in capturing long dependencies. LSTMs are intrinsically RNNs in which changes were introduced in the computation of hidden states and outputs, using the inputs. They were developed to address some limitations that occurred during the training process of regular RNNs.
- *Transformers*, like RNNs, were designed to handle sequential data and tackle problems in natural language processing (for instance, translation). Transformers were repurposed to address forecasting in time series. The encoder and decoder form the two parts of the transformer’s architecture. The encoder mainly consists of an input layer, an input encoding, a positional encoding mechanism, and a stack of identical encoder layers. In the seminal work of Vaswani (2017), the decoder is composed of an input layer, an input encoding, a positional encoding mechanism, a stack of identical decoder layers and an output layer.
- *Autoforecast* (Optimal Forecasting Model Selection*)* (Abdallah 2022) is a meta-learning approach that allows for the quick inference of the best time-series forecasting model for an unseen dataset. Given a new time series dataset, AutoForecast selects the best performing forecasting

algorithm and its associated hyperparameters without having to first train or evaluate all the models on the new time series data to select the best one. The meta-learner is trained on the models' performances on historical datasets and the time-series meta-features of these datasets.

6.4 ML Tracking Tools

To enable DT4GS to properly track the performance of developed machine learning models and their version history, we consider third-party software that could assist us in managing their whole lifecycle. After all, current machine learning methods often require thorough experimentation. This translates to training a particular model instance, usually for prolonged periods of time, by feeding it with a large set of data observations, then evaluating the model and measuring its performance by using standardized metrics and finally trying to interpret the results and draw valid conclusions.

There are several tools available for tracking and managing machine learning experiments. To our knowledge, two of the most complete ones are MLflow and Weights & Biases. These two solutions are the ones currently being studied by the DT4GS consortium in order to identify the most suitable solution for the project's needs. In this chapter we present a brief description of the capabilities and structures of the tools.

6.4.1 Mlflow

MLflow[42] is an open-source platform that helps manage the whole machine learning lifecycle. Its features include tracking experiments, packaging code into reproducible runs, and sharing and deploying models. MLflow currently offers four components:

- MLflow Tracking: An API to log parameters, code versions, metrics, and output files when conducting machine learning experiments. The results of each experiment can be then visualized and compared using an interactive UI. Tracking can also be used in any environment (for example, a standalone script or a notebook) to log results to local files or to a server, then compare multiple runs.
- MLflow Projects: A code packaging format for reproducible runs using Conda and Docker. Each project is simply a directory with code or a Git repository and uses a descriptor file to specify its dependencies and how to run the code. An MLflow Project is defined by a simple YAML file called MLproject. Projects can specify their dependencies through a Conda environment. A project may also have multiple entry points for invoking runs, with named parameters. You can run projects using the command-line tool, either from local files or from a Git repository. MLflow will automatically set up the right environment for the project and run it. In addition, if you use the MLflow Tracking API in a Project, MLflow will remember the project version executed (that is, the Git commit) and any parameters. You can then easily rerun the exact same code.
- MLflow Models: A convention for packaging machine learning models in multiple formats called "flavors". MLflow offers a variety of tools to help you deploy different flavors of models. Each MLflow Model is saved as a directory containing arbitrary files and an MLmodel descriptor file that lists the flavors it can be used in. In this way, models (from any ML library) can be easily

[42] https://mlflow.org/

deployed to batch and real-time scoring on platforms such as Docker, Apache Spark, Azure ML, and AWS SageMaker.

- MLflow Model Registry: A centralized model store, set of APIs, and UI, to collaboratively manage the full lifecycle of MLflow Models. It provides model lineage (which MLflow experiment and run produced the model), model versioning, stage transitions (for example from staging to production), and annotations.

6.4.2 Weights & Biases

Weights and Biases [43]is a platform for machine learning experiment tracking, dataset versioning, and model management. Its interactive dashboards facilitate visualizing model performance, training metrics, and model predictions. The functionality supported by Weights and Biases is divided into 6 main components:

- Experiments: Tracks, compares, and visualizes ML experiments. Logged model hyperparameters, output, and metrics are streamed live into interactive graphs and tables. In this way, it is easy to evaluate and compare the performance of multiple ML models simultaneously. CPU and GPU usage can also be monitored, in order to identify potential training bottlenecks and avoid wasting expensive resources. There is rich support for logging media of various forms, including images, video, audio, and 3D objects. Dataset versioning is also provided with deduplication and diffing handled internally by Weights & Biases.
- Reports: Allows for organizing and embedding visualizations, describing findings, sharing updates with collaborators, and more. More specifically, updates and outcomes of developed machine learning projects can be shared efficiently. Consequently, explaining how a particular model works or whether its subsequent versions improved becomes more straightforward by integrating graphs and other visualizations into shared reports. Furthermore, there is support for making live comments, and taking snapshots of work logs. Finally, reports can be easily exported as a LaTeX zip file or converted to PDF.
- Artifacts: Provides dataset versioning, model versioning, and tracking dependencies and results across machine learning pipelines. An artifact in this context is a versioned folder of data. Entire datasets can be stored directly in artifacts, or artifact references can be used to point to data in other systems like S3, GCP, or other user-defined systems. Dependency graphs can be built on users' premises or in the cloud to trace the flow of data through pipelines. This ensures that keeping track of which datasets are used as input to the tested models is as effortless as possible. Moreover, this component tracks the evolution of users' data over time and stores checkpoints of the best-performing models.
- Tables: Logs, queries, and analyzes tabular data. In essence, it helps users to better understand their datasets, visualize model predictions, and share insights in a central dashboard. Changes can be precisely compared across models, epochs, or individual examples. Importantly, this aids in realizing higher-level patterns in the data, and capturing and communicating valuable insights with visual samples.

[43] https://wandb.ai

- Sweeps: Automates hyperparameter search and explores the space of possible models. Sweeps basically combines the benefits of automated hyperparameter search with visualization-rich, interactive experiment tracking. Users can pick from popular search methods such as Bayesian, grid search, and random to search the hyperparameter space. It also supports scaling and parallelizing Sweep jobs across one or more machines.
- Models: The W&B Model Registry is used as a central system of record for models. It creates Registered Models to organize the best model versions for a given task. Besides, it tracks models moving into staging and production. A detailed history of all changes is maintained, including which user moved a model to production.

7 Conclusions

DT4GS is aimed at making Digital Twin technology readily available to the shipping industry to support accelerated transition to zero emissions. Within this aim, Work package 2 aims to develop an Open Ship Operational Optimization Digital Twining Infrastructure, built on top of a Waterborne Sector Dataspace to support shipping companies to build their own ship specific DTs.

This Deliverable has defined the user requirements, high level architecture and implementation technologies selection for the Dataspace.

First, task 2.2 whose output is described in this Deliverable, surveyed the Industry requirements and information architectures for shipping and maritime, focusing in particular on the latest paradigms such as Internet of Things and Internet of Ships. The task was informed also by international industry standards such as Industrial Data Spaces.

Next, the task analysed the user data requirements as described by the Project's Living Labs in Deliverable 1.1. From that analysis, the task focused on data processing requirements, including storage, real time, volumes, security and so on. This led to the definition of a high-level software architecture, in terms of ship and shore-based software modules, to realise the required functionality of the Dataspace and adhere to the users' nonfunctional requirements for data security, sovereignty, and so on.

Based on that high level architecture, the task selected implementation technologies that meet the Project's requirements. The main criteria for the selection of technologies for data storage, communications and processing, was that such technologies would have to be already proven in maritime and/or other industry domains. Additionally, such technologies would have to meet current and future maintenance, evolution and cost criteria, in order to be acceptable by the Project's stakeholders.

Following that, the Deliverable discussed the deployment and utilisation of the dataspace architecture in maritime analytics applications.

The overall strategy employed in this Deliverable, therefore, has been to adopt suitable solutions from broader domains to the maritime context, in order to reap benefits from adoption, support, and future developments. The Dataspace infrastructure described in this Deliverable provides a solid foundation for further developments, in the remaining of the DT4GS Project, specifically, a refinement of the architecture and reference implementations of the architecture in the Project's Living Labs.

Important work dimensions in the next six months are the potential development of the DT4GS dataspace not only to support the LL use cases but also the broader knowledge transfer on decarbonisation technologies. For this close cooperation with T3.1, the DT4GS knowledge hub incorporating outputs from T1.1, T1.2 and T1.3 as well asT2.4 DT4GS Model Blueprints and Open Model Library is planned.

Further, mainly through the knowledge hub task and DT4GS Alliance activities the industry perspective of the dataspace will be emphasized, utilising data available in interconnected Digital Twins (DTs). Interconnected DTs represent 1) ships equipped with one or more decarbonisation solutions or 2) decarbonisation equipment test facilities or related DTs. The interface of such nodes are connectors to be developed and reported in the v2 deliverable.

8 References

1. Apache Kafka, security, https://kafka.apache.org/documentation/#security
2. *MongoDB. Internet of Things. url: https://www.mongodb .com / use - cases/internet-of-things (visited on 04/12/2017).*
3. *Dan Sullivan, James Sulllivan. Find the IoT database that best fits your enterprise's needs (Source: https://internetofthingsagenda.techtarget.com/feature/Find-the-IoT-database-that-best-fits-your-enterprises-needs)*
4. *Mustafa Abdallah, Ryan Rossi, Kanak Mahadik, Sungchul Kim, Handong Zhao, and Saurabh Bagchi. "Autoforecast: Automatic time-series forecasting model selection." In Proceedings of the 31st ACM International Conference on Information and Knowledge Management, CIKM '22, page 5–14, New York, NY, USA, 2022. Association for Computing Machinery.*
5. *Paul Boniol, John Paparrizos, Yuhao Kang, Themis Palpanas, Ruey S. Tsay, Aaron J. Elmore, and Michael J. Franklin. "Theseus: Navigating the labyrinth of time-series anomaly detection." Proc. VLDB Endow., 15(12):3702–3705, sep 2022.*
6. *Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. "SAND: streaming subsequence anomaly detection." Proc. VLDB Endow., 14(10):1717–1729, 2021.*
7. *David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. "Unsupervised time series outlier detection with diversity-driven convolutional ensembles." Proc. VLDB Endow., 15(3):611–623, feb 2022.*
8. *Sohee Cho, Wonjoon Chang, Ginkyeng Lee, and Jaesik Choi. "Interpreting internal activation patterns in deep temporal neural networks by finding prototypes." In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '21, page 158–166, New York, NY, USA, 2021. Association for Computing Machinery.*
9. *Yiru Chen and Silu Huang. "TSexplain: Surfacing evolving explanations for time series." In Proceedings of the 2021 International Conference on Management of Data, SIGMOD '21, page 2686–2690, New York, NY, USA, 2021. Association for Computing Machinery.*
10. *Fatoumata Dama and Christine Sinoquet. "Time Series Analysis and Modeling to Forecast: a Survey." (2021).*
11. *Jan G. De Gooijer and Rob Hyndman. (2005). "25 Years of IIF Time Series Forecasting: A Selective Review." SSRN Electronic Journal. 10.2139/ssrn.748904.*
12. *Riccardo Guidotti and Anna Monreale. "Designing shapelets for interpretable data agnostic classification." In Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society, AIES '21, page 532–542, New York, NY, USA, 2021. Association for Computing Machinery.*
13. *Ilaria Gandin, Arjuna Scagnetto, Simona Romani, and Giulia Barbati. "Interpretability of time-series deep learning models: A study in cardiovascular patients admitted to intensive care unit." Journal of Biomedical Informatics, 121:103876, 2021.*
14. *Abdelouahab Khelifati, Mourad Khayati, Philippe Cudre-Mauroux, Adrian Hanni, Qian Liu, and Manfred Hauswirth. "Vadetis: An explainable evaluator for anomaly detection techniques." 2021 IEEE 37th International Conference on Data Engineering (ICDE), pages 2661–2664, 2021.*

15. *Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. "Lstm fully convolutional networks for time series classification." IEEE Access, 6:1662–1669, 2018.*
16. *Kwei-Herng Lai, Daochen Zha, Junjie Xu, Yue Zhao, Guanchu Wang, and XiaHu. "Revisiting time series outlier detection: Definitions and benchmarks." In Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021.*
17. *Zachary Lipton. (2015). "A Critical Review of Recurrent Neural Networks for Sequence Learning."*
18. *Zhenyu Liu, Zhengtong Zhu, Jing Gao and Cheng Xu. "Forecast Methods for Time Series Data: A Survey," in IEEE Access, vol. 9, pp. 91896-91912, 2021, doi: 10.1109/ACCESS.2021.3091162.*
19. *Lkhagvadorj Munkhdalai, Tsendsuren Munkhdalai, Kwang Ho Park, Tsatsral Amarbayasgalan, Erdenebileg Batbaatar, Hyun Woo Park, and Keun Ho Ryu. "An end-to-end adaptive input selection with dynamic weights for forecasting multivariate time series." IEEE Access, 7:99099–99114, 2019.*
20. *Roberto Medico, Joeri Ruyssinck, Dirk Deschrijver, and Tom Dhaene. "Learning multivariate shapelets with multi-layer neural networks for interpretable timeseries classification." Adv. Data Anal. Classif., 15(4):911–936, dec 2021.*
21. *Milad Moradi and Matthias Samwald. "Post-hoc explanation of black-box classifiers using confident itemsets." Expert Syst. Appl., 165:113941, 2021.*
22. *Qianli Ma, Wanqing Zhuang, Sen Li, Desen Huang, and Garrison Cottrell. "Adversarial dynamic shapelet networks." Proceedings of the AAAI Conference on Artificial Intelligence, 34(04):5069–5076, Apr. 2020.*
23. *Felipe Oviedo, Zekun Ren, Shijing Sun, Charles M. Settens, Zhe Liu, Noor Titan Putri Hartono, Savitha Ramasamy, Brian L. DeCost, Siyu Isaac Parker Tian, Giuseppe Romano, Aaron Gilad Kusne, and Tonio Buonassisi. "Fast and interpretable classification of small x-ray diffraction datasets using data augmentation and deep neural networks." npj Computational Materials, 5:1–9, 2019.*
24. *John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S. Tsay, Themis Palpanas, and Michael J. Franklin. "TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection." Proc. VLDB Endow., 15(8):1697–1711, 2022.*
25. *Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. "Learning homophily couplings from non-iid data for joint feature selection and noise-resilient outlier detection." In Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI'17, page 2585–2591. AAAI Press, 2017.*
26. *Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. ACM Comput. Surv., 54(2), mar 2021.*
27. *Leonardos Pantiskas, Kees Verstoep, and Henri Bal. "Interpretable multivariate time series forecasting with temporal attention convolutional neural networks." In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1687–1694, 2020.*
28. *Thomas Rojat, Raphael Puget, David Filliat, Javier Del Ser, Rodolphe Gelin, and Natalia Diaz-Rodriguez. "Explainable artificial intelligence (xai) on timeseries data: A survey." ArXiv, abs/2104.00950, 2021.*
29. *Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?: Explaining the predictions of any classifier." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.*

30. *Md Amran Siddiqui, Alan Fern, Thomas G. Dietterich, and Weng-Keen Wong. "Sequential feature explanations for anomaly detection." ACM Trans. Knowl. Discov. Data, 13(1), jan 2019.*
31. *Amal Saadallah, Maryam Tavakol, and Katharina Morik. "An actor-critic ensemble aggregation model for time-series forecasting." In 2021 IEEE 37th International Conference on Data Engineering (ICDE), pages 2255–2260, 2021.*
32. *Anirban Sarkar, Deepak Vijaykeerthy, Anindya Sarkar, and Vineeth N. Balasubramanian. "A framework for learning antehoc explainable models via concepts." 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10276–10285, 2022.*
33. *Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. "Anomaly detection in time series: A comprehensive evaluation." Proc. VLDB Endow., 15(9):1779–1797, jul 2022.*
34. *Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Tranad. "Deep transformer networks for anomaly detection in multivariate time series data." Proc. VLDB Endow., 15(6):1201–1214, jun 2022.*
35. *Andreas Theissler, Francesco Spinnato, Udo Schlegel, and Riccardo Guidotti. "Explainable ai for time series classification: A review, taxonomy and research directions." IEEE Access, 10:100700–100724, 2022.*
36. *Gilles Vandewiele, Femke Ongenae, and Filip De Turck. "Gendis: Genetic discovery of shapelets." Sensors, 21(4), 2021.*
37. *Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention is all you need." In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). 6000–6010.*
38. *Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. "Transformers in time series: A survey.", 2022.*
39. *Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. "Timeeval: A benchmarking toolkit for time series anomaly detection algorithms." Proc. VLDB Endow., 15(12):3678–3681, sep 2022.*
40. *Internation Data Spaces, https://internationalDataspaces.org/*
41. Multisensor system for fire detection and situational awareness Susan Rose-Pehrsson, Christian Minor, Kevin Johnson, Jeffrey Owrutsky, Stephen Wales, Daniel Steinhurst, and Daniel Gottuk

Annex I – Base code for data stream from InfluxDB database to an external source via Kafka Connect

The code presented in this annex is a first base implementation in Java of the data stream from a InfluxDB database to an external source utilising Kafka Connect libraries. InfluxDBKafkaConnector class extends the SourceTask class from the Kafka Connect API. The start method is used to connect to the InfluxDB database using the connection information provided in the task configuration. The poll method is used to execute the query and convert the results to SourceRecord objects, which can be consumed by a Kafka topic. The stop method is used to close the connection to the InfluxDB database. Note that the conversion from QueryResult to SourceRecord is showed as an example implementation and will be needed to be adapted given the LL data to process.

```java
import org.apache.kafka.connect.source.SourceRecord;
import org.apache.kafka.connect.source.SourceTask;
import org.influxdb.InfluxDB;
import org.influxdb.InfluxDBFactory;
import org.influxdb.dto.Query;
import org.influxdb.dto.QueryResult;

import java.util.List;
import java.util.Map;

public class InfluxDBKafkaConnector extends SourceTask {

  private InfluxDB influxDB;
  private String databaseName;
  private String query;

  @Override
  public void start(Map<String, String> props) {
    String url = props.get("influxdb.url");
    String username = props.get("influxdb.username");
    String password = props.get("influxdb.password");
    databaseName = props.get("influxdb.database");
    query = props.get("influxdb.query");

    influxDB = InfluxDBFactory.connect(url, username, password);
  }


  @Override
  public List<SourceRecord> poll() throws InterruptedException {
   QueryResult result = influxDB.query(new Query(query, databaseName));
   List<SourceRecord> records = new ArrayList<>();
   for (QueryResult.Result res : result.getResults()) {
    for (QueryResult.Series series : res.getSeries()) {
     List<List<Object>> values = series.getValues();
     for (List<Object> value : values) {
      Map<String, Object> sourcePartition = new HashMap<>();
      sourcePartition.put("database", databaseName);
      sourcePartition.put("series", series.getName());
      Map<String, Object> sourceOffset = new HashMap<>();
      sourceOffset.put("index", value.get(0));
      SourceRecord record = new SourceRecord(
        sourcePartition,
        sourceOffset,
        "influxdb",
        null,
        series.getColumns(),
        value.subList(1, value.size()));
      records.add(record);
     }
    }
   }
   return records;
  }

 @Override
 public void stop() {
  influxDB.close();
 }
```