DT4GS

Digital Twins for Green Shipping

D2.8: *DT4GS Edge and Orchestration Infrastructure (v1)*

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	D2.8	**Work Package**	WP 2
Document Type	OTHER	**Dissemination Level**	PU
Lead beneficiary	IBM		
Responsible author	Fearghal O'Donncha		
Contractual due date	30/11/2023	**Actual submission date**	30/11/2023

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

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the granting authority can be held responsible for them.

While the information contained in the document is believed to be accurate, the authors or any other participant in the DT4GS consortium make no warranty of any kind with regard to this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the DT4GS Consortium nor any of its members, their officers, employees, or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the DT4GS Consortium nor any of its members, their officers, employees, or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

Document information

Contributors	
Contributor Name	**Organisation (Acronym)**
Fearghal O'Donncha	IBM
John D. Sheehan	IBM
Sofiane Zemouri	IBM

Document history				
Version	**Date**	**%**	**Changes**	**Author**
0.1	[01/09/2022]	5%	Created the ToC draft	Sofiane Zemouri (IBM)
0.2	[31/05/2023]	25%	Added background material	Fearghal O'Donncha (IBM)
0.3	[01/07/2023]	45%	Added architecture details and use case examples and documentation	John D. Sheehan (IBM)
0.4	[20/09/2023	55%	Added background material related to edge literature review and motivation	Fearghal O'Donncha (IBM)
0.5	[28/10/2023]	65%	Added overview of Edge architecture	Fearghal O'Donncha / John D Sheehan (IBM)
0.6	[3/11/2023]	75%	Added details on the creation of policies for deployment	Fearghal O'Donncha (IBM)
0.7	[10/11/2023]	85%	Added details on AI-backed contributions	Fearghal O'Donncha (IBM)
0.8	[20/11/2023]	85%	Improving structure, adding exec summary and conclusion, adding mapping to requirements	Fearghal O'Donncha (IBM)
0.9	[25/11/2023]	95%	Incorporating reviewers comments	John D Sheehan (IBM)
1.0	[30/11/2023]	100%	Final version	Fearghal O'Donncha

Quality Control (includes peer & quality control reviewing)			
Date	**Version**	**Name (Organisation)**	**Role & Scope**
[30/09/2022]	0.2	WP2	QM ToC Approval
[20/11/2023]	0.9	Dimitris Skoutas (ATH)	Reviewer 1
[27/11/2023]	0.9	Bill Karakostas (INLECOM)	Reviewer 2

Executive summary

This report outlines the edge and cloud technical architecture for the DT4GS project. Architecture choices and decisions are shaped by the specific requirements of the four living labs use cases. Over the first 18 months of the DT4GS project, we collated these requirements through numerous meetings, reports, and deliverables from each of the living labs. In response, we evaluated multiple technological solutions, and identified the open-source solution – OpenHorizon – as the most fitting architectural framework.

The report delves into an in-depth analysis of the technical nuances of edge and cloud computing within the maritime sector. Driven by diverse use cases from the four living labs, which include tanker, container ship, roll-on/roll-off passenger, and bulker services, it scrutinises the unique needs and requirements specific to different types of shipping services. This encompasses a detailed technical comparison of various cloud and edge computing solutions, such as Kubernetes, IoT Azure Edge, KubeEdge, Red Hat Advanced Cluster Management, and KubeStellar, all evaluated against the backdrop of user-driven requirements.

The report provides a detailed description of the OpenHorizon platform for managing the service software lifecycle of containerized workloads and related machine learning assets. In particular it focuses on the requirements to address application placement at scale and manage operations in conditions with limited or sporadic connectivity.

A significant element of this deliverable is the analysis of computing needs for advanced digital twin functionalities in the shipping industry. It considers the diverse characteristics and requirements of the four living labs, focusing on scalability, resilience, usability, and system administration. This includes considerations of typical shipping routes, operational needs, skill availability, security, and connectivity.

In its concluding section, this deliverable explores the role of AI in two key areas: firstly, in simplifying the management and orchestration of applications across distributed cloud and edge computing environments within the shipping industry. This includes examining how innovation can meet the demands for scalability and the resulting increases in productivity. Secondly, it looks at how AI can aid in developing a scalable model library that is versatile enough to meet the diverse needs of the shipping sector.

This report represents the first version of the deliverable. The upcoming version (D2.9) to be delivered in M34 will provide a more detailed exploration into enhancements powered by AI, specifically designed to cater to the distinct requirements and challenges of creating digital twins for green shipping.

Contents

1 Introduction 9
1.1 Mapping DT4GS Outputs and Task Description 10
1.2 Deliverable Overview and Report Structure 12
1.3 DT4GS HQ-Ship interaction scenario 12
1.3.1 Cloud Vs Edge Computing 13
1.3.2 Why Edge Computing in the Maritime Industry 13
2 DT4GS Cloud to Edge Architecture 16
2.1 State of the Art in Edge Computing 16
2.2 Evaluating existing edge-computing solutions 18
2.2.1 Azure IoT Edge 18
2.2.2 Red Hat Advanced Cluster Management (RHACM) 19
2.2.3 KubeEdge 19
2.2.4 KubeStellar 20
2.2.5 Open Horizon 20
2.2.6 DT4GS Edge Solution 21
2.3 Full Stack Edge Solution 22
2.3.1 Edge Infrastructure and compute 23
2.3.2 Edge platform and orchestration 24
3 DT4GS Headquarters Infrastructure 27
3.1 Overview of the headquarters testbeds (LLs) 27
3.2 Service requirements and limitations 28
3.3 Service Blueprint – Provisioned services on the headquarters 31
4 DT4GS Far Edge (Vessel) Infrastructure 34
4.1 Overview of the ship testbeds (LLs) 34
4.2 Far Edge Constraints 35
4.3 Testing Environment 35
4.4 Service Blueprint – Deployable library on the ship 35
5 DT4GS Enhanced Intelligence in data collection, service provisioning and orchestration 39
5.1 Data collection optimisation – state of the art 39
5.2 AI-assisted service provisioning and orchestration solutions 41
6 Conclusions 44

References 45
Annex I 47
Quickstart, setup server 47
Quickstart, adding an additional edge node 49
install docker on edge node 49
install open-horizon cli and daemon on edge node 49
configure horizon daemon on edge node 49
Troubleshooting 50
Deploying a container to an edge node - 1 51
Deploying a container to an edge node 52
Deploying multiple dependent containers to an edge node 55
Updating a deployed container 60

List of figures

Figure 1 DT4GS approach. The core digital twin is deployed in the cloud and represents a comprehensive digital replica of ship processes, components, and their real-time and predictive aspects. Simultaneously, selected mission-critical components are simulated directly on the Edge – on-board ship compute resources – ensuring continuous availability, minimal latency, and zero-to-low data gravity constraints. 9
Figure 2: High-level topology for a typical edge computing setup. Taken from https://open-horizon.github.io/ 25
Figure 3: Data required for formulation of digital twin (from Deliverable 1.1) 28
Figure 4: provides a schematic of the Cloud-Edge DT4GS architecture that adapts to the required flexibility. Starting, on the ship, an edge orchestration solution is deployed. This can deploy and manage different applications or edge data services. Each edge data service is a containerised application deployed on the ship infrastructure. It addresses capabilities such as storage, monitoring, prediction, and optimisation. On-ship deployments are typically compute- and communication-constrained. Hence, orchestration of services must consider those factors. Each ship or edge node must securely communicate with the cloud HQ and with a container registry that may be located at HQ, public cloud, or somewhere else. Critically, edge nodes are resilient to connectivity loss between ship and HQ. Hence, a resilient container management solution is required that acts when feasible rather than on a purely schedule- or event-based approach. Finally, a DevOps framework is required to manage updates and changes to the application library deployed on the ship. 31
Figure 5: Open Horizon is a client/server application with a centralized server and remote edge devices connected over a computer network. It provides targeted placement of a service (containerized application) or collection of services to a specific registered device, or the entire collection of registered devices under its control. It also allows for targeted updates of deployed services, again with a granularity of a single device up to the entire collection. If a deployed service is suitable constructed. An Open Horizon edge node is a Linux device running the Open Horizon Edge Agent software. 33
Figure 6: A data unit testing framework for DT4GS that asserts whether the time series data is withing the prescribed bounds. The data covers a 1-year period at 5-minute intervals. While many sensor tags are returned. For demonstration purposes, we consider a subset. The min and max thresholds are defined based on values provided by the Living Labs. 40
Figure 7: Results from Great Expectations data quality testing framework. Results show that for the 9 expectations specified (as shown in Figure 6), 5 failed due to returning values outside the expected range. Note, of course that doesn't indicate low quality data but rather that some data values should be further interrogated and possibly excluded from model development. 41

List of tables

Table 1 Glossary of acronyms and terms. 8
Table 2 Adherence to DT4GS Grant Agreement deliverable and work description. 10
Table 3: Summary of the advantages and disadvantages of shipping 15

Glossary of terms and acronyms used

Table 1 Glossary of acronyms and terms.

Acronym / Term	Description
AI	Artificial Intelligence
CBM	Condition Based Maintenance
CI/CD	Continuous Integration / Continuous Deployment
DT	Digital Twin
EAM	Edge Application Manager
IoT	Internet of Things
KG	Knowledge Graph
LL	Living Lab
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
OH	Open Horizon
RHACM	Red Hat Advanced Cluster Management

1 Introduction

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 below.

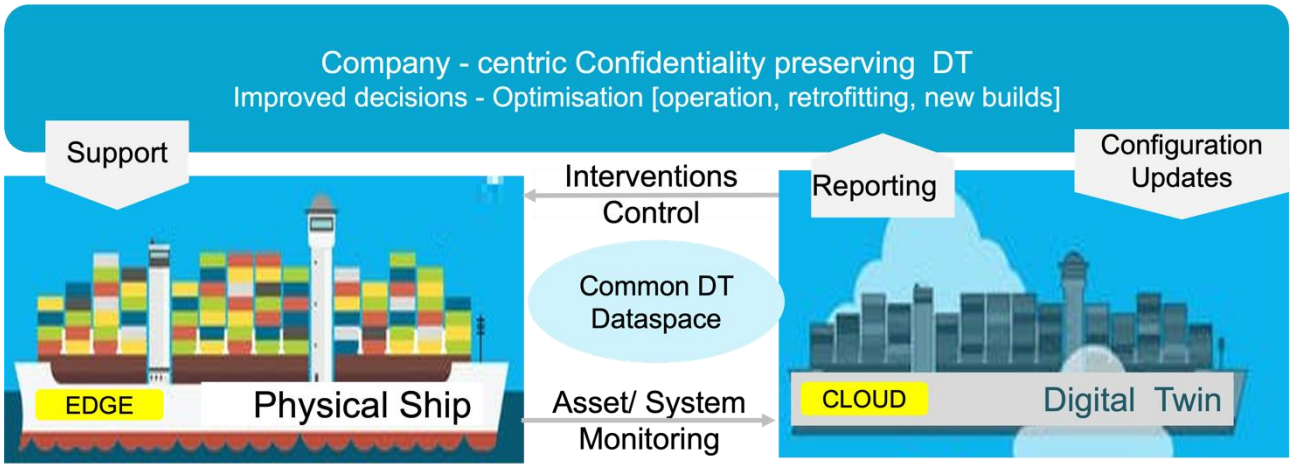

Figure 1 DT4GS approach. The core digital twin is deployed in the cloud and represents a comprehensive digital replica of ship processes, components, and their real-time and predictive aspects. Simultaneously, selected mission-critical components are simulated directly on the Edge – on-board ship compute resources – ensuring continuous availability, minimal latency, and zero-to-low data gravity constraints.

DT4GS will provide a virtual representation of ships and physical transport entities with a bi-directional communication links from sensing to actuation/control, to data driven simulation and AI based decision support. Results will be directly communicated to people who will implement necessary actions. In DT4GS extra emphasis will be given to:

- DT applications onboard the ship utilising advanced IoT and edge computing infrastructure.
- Using labelled data or self-learning capabilities for AI/ML training to provide the ground truth for accurate predictions (supervised learning), and where there is need to learn from experience to provide the reward function (reinforcement learning).
- The deployment of powerful new foundation models built on the transformers architecture that are revolutionizing many aspects of natural language processing and computer vision
- Creating 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.

1.1 Mapping DT4GS Outputs and Task Description

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

DT4GS GA Component Title	DT4GS GA Component Outline	Respective Document Chapter(s)	Justification
DELIVERABLE			
D2.8 DT4GS Headquarters, Edge and Orchestration Infrastructure	*Cloud based infrastructure for services provisioning, deployable library on vessels. Hosting and orchestration services including tools for deployment automation.*	*All chapters*	*Report considers this from two perspectives: 1) what are the technical requirements of shipping based on living labs and 2) how do the different technology solutions available address those*
TASK			
ST2.5.1 DT4GS Service provisioning at the headquarters (HQ)	*Development and provision of cloud-type infrastructure and services necessary for data collection at HQ, and for model / simulation / optimisation development and deployment which will bring together the outputs of WP1 and WP2 tasks.*	*Chapters 3 & 4*	*These provide a detailed analysis of the living labs and their unique requirements for each. We then describe the HQ management hub and the edge node deployed on the ship. The management and orchestration services across these are then described*
ST2.5.2 DT4GS Far edge (vessel) infrastructure	*Define the reference architecture for DT4GS, to be implemented in accordance with shipping industry standards. Design the DT4GS data schema (ontology), based on the DT LL expectations and resulting data mapping. Establish a common understanding and*	*Chapter 4*	*Chapter 4 describes the policies for deploying services on the edge. Importantly this doesn't restrict to DT4GS technologies. A key need highlighted during the project (e.g. Advisory Board Meeting Sept 27th 2023), was the ability*

	representation of DT4GS models via a metadata Knowledge Graph (KG), which will create a common metamodel and vocabulary for specific instantiations of DT4GS (e.g., in the LLs). The KG thus will act as a single source of truth for the available models, databases, and intelligence, describing the interdependences amongst physical and digital components (connectors), as well as the individual state of operations for each system component.		*to include existing open-source and closed-source models and capabilities from different stakeholders.* *The DT4GS technical architecture is agnostic to any containerised applications and scalable via a flexible deployment and orchestration policy*
ST2.5.3 DT4GS Hosting and Orchestration of services	*Enable the creation and instantiation of service deployment processes both at HQ and on-board the ship based on the model and data blueprints stored in the OML, including the retrieval of datasets, providing access to a set of registered recipes through user-friendly interfaces to set the execution parameters. Macro level composition services will ease the deployment of the envisioned configurations (dataset sources, execution engines/locations, and their interactions) and applications by users, simplifying the onboarding process.*	*Chapter 3 and 4*	*These chapters describe the creation of deployment policies detailing the deployment of containerised applications on Edge or Cloud. These are agnostic to any containerised application and the DT4GS architecture and dataspace can be readily deployed either in whole or in part based on the container-first development approach adopted across the DT4GS project.*
ST2.5.4 DT4GS Enhanced Intelligence in data collection, service provisioning and orchestration	*AI algorithms will be researched and will be introduced to augment the cloud and edge infrastructure, improving responsiveness, automation, reactive and predictive orchestration, self-healing, meeting increased capacity demands and thereby avoiding potential service disruptions, and improving the overall data collection, service provisioning and orchestration processes.*	*Chapter 5*	*Chapter 5 discusses the role of AI to:* *1) Streamline the creation of orchestration policy documents at scale.* *2) Automate the creation and deployment of AI foundation models that enhance the DT4GS model library.*

1.2 Deliverable Overview and Report Structure

This deliverable describes the edge and cloud solution for the DT4GS project. In brief, it considers user requirements, existing state of the art in distributed multicloud computing, computing in the cloud for shipping, computing at the far edge for shipping, and AI-backed innovation to enhance digital twin developments for shipping.

In the first chapter, we introduce edge and cloud computing and describe the motivations of each. Further, we give an overview of why edge and cloud computing are a key value-add for shipping. Chapter 2 provides an overview of state of the art in edge computing. We interrogate several different solutions and evaluate them in the context of the DT4GS living labs. A key consideration for shipping is the remote locations and regions it extends to. Hence, Chapter 2 discusses the role of zero-touch solutions that can be particularly attractive for in industries operating under spartan conditions.

Chapter 3 provides an overview of the technical architecture in the cloud or at shipping headquarters. This chapter provides a detailed overview of the diverse use cases from the four living labs, which include tanker, container ship, roll-on/roll-off passenger, and bulker services. It scrutinizes the unique needs and requirements specific to different types of shipping services. It provides a high-level overview of the technical architecture required to integrate operations on the ship (edge) with headquarter decision making (cloud) and external services and application development (model library). Extending from this, we provide a highly detailed description of the OpenHorizon technical architecture for shipping. This includes considerations around service management, application placement, and scalability across 100s or 1000s of ships.

Chapter 4 discusses the technical architecture on the edge. It details the constraints and restrictions of operating in compute- and communication-constrained regions together with the myriad requirements of data processing on the edge such as IoT, machine learning, and security. We detail the requirements and specifications to enable effective deployments at scale. Chapter 4 provides a detailed description of deploying applications (with a full description provided in the Appendices).

Chapter 5 speaks to the role of AI and AIOps to enhance development on the edge. Specific topics considered are data quality on the edge and structures to ensure a consistent standard of data quality; the use of AI to automate the deployment and orchestration of applications on the cloud; and the role of AI to simplify and streamline model development for digital twin applications.

Finally, the Conclusions section summarises the deliverable and speaks to future material that will be covered in part B of this deliverable (due M34).

1.3 DT4GS HQ-Ship interaction scenario

Recent years has seen the evolution and maturation of IoT technology from primarily a data collection and transmission solution to devices with significant compute capabilities and the ability to deploy workloads close to the data. In traditional IoT setups, data generated by connected devices are transmitted to the cloud for processing and analysis. This approach worked well for certain applications but introduced challenges like latency, network congestion, and increased reliance on always-available cloud connectivity. As data volumes continued to expand, this challenge becomes more pressing. In response, edge computing emerged as a viable solution to address these issues.

Edge computing minimises latency by bringing computational workloads closer to the data source and the location where actions need to be taken – removing the distance between collecting data, processing

it, and making those results available for use in decision making. Data is generated from diverse sources and at different scales, encompassing equipment, devices, individuals, and processes.

1.3.1 Cloud Vs Edge Computing

The primary difference between cloud computing and edge computing is the location where data processing occurs. In cloud computing, data is processed on a central server such as Amazon EC2 instances. This is usually located far away from the source of information. Hybrid cloud is a mixed computing environment where applications are run using a combination of computing, storage, and services in different environments – public clouds and private clouds, including on-premises data centres or edge locations.

Hybrid cloud computing approaches are widespread because almost no one today relies entirely on a single public cloud. Hybrid cloud solutions offer the flexibility to seamlessly migrate and manage workloads across diverse cloud environments, empowering organizations to tailor their infrastructure to meet specific business requirements. By adopting hybrid cloud platforms, organizations gain the ability to lower costs, mitigate risks, avoid vendor lock-in, and leverage existing cloud-native developer skills and CI/CD pipelines to drive successful digital transformation initiatives.

In today's landscape, the hybrid cloud approach has become a prevalent infrastructure setup. As organizations undergo cloud migrations, hybrid cloud implementations naturally emerge, enabling a gradual and systematic transition of applications and data. With hybrid cloud environments, enterprises can continue utilizing on-premises services while harnessing the advantages of public cloud providers like AWS, Azure, and GCP, which offer flexible options for data storage and application access (Google 2023).

Edge computing extends the hybrid cloud paradigm further. It emerged to address the unique requirements of enterprise and consumer applications. Although they possess individual traits, edge computing and hybrid cloud can collaborate to establish a comprehensive and adaptable computing infrastructure. A notable aspect of contemporary edge computing solutions is their adoption of cloud native development practices specifically designed for the edge. By leveraging cloud native development practices, applications can be built and deployed using the same skills and tools that have been honed for developing cloud native applications in hyper-scale cloud environments or private data centres. This allows for the seamless extension of these practices to the edge, enabling organizations to leverage their existing expertise and resources for edge computing deployments.

Hence, aspects such as distributed computing, data processing, management, and integration, as well as workload placement and optimisation are enabled in an accelerated and scalable manner, agnostic of where applications are deployed.

1.3.2 Why Edge Computing in the Maritime Industry

In a recent report (Gardner, Kenney, and Chubb 2021), Inmarsat detailed that average daily data consumption per ship nearly tripled, from 3.4 to 9.8 gigabytes between January 2020 and March 2021. In the same report, authors suggest that the global maritime digital products and services market in 2021 is 18% bigger than previous forecasts predicted, and that growth is running three years ahead of pre-pandemic forecasts. The pandemic was a huge driver for digitalisation of the shipping industry. Consequently, advances are required to ensure that these datasets are being leveraged optimally for improved efficiency and decision.

Broadly edge computing is a distributed computing paradigm that aims to circumvent obstacles related to connectivity, latency, autonomy, and resilience. The definition of the "edge" is flexible and context

dependent. For example, in the case of a shipping company, the edge may encompass activities at docks where shipments are loaded and unloaded, or it may include on-board processing of ship systems and operations. In both scenarios, processing and analysis occur in near real-time, leading to data-driven decision-making. While the company's headquarters with the main data centre may be located miles away, the edge represents the crucial point where data is collected, processed, and managed to derive insights, irrespective of latency challenges. In our case, we consider the ship to be the **Edge Server** where compute tasks are deployed and managed at HQ. Each ship may have additional Edge Devices such as cameras, sensors, and actuators which communicate with the Edge Server where these data are processed, interpreted, aggregated, analysed, and communicated to the Cloud. There are however other examples of edge computing in the maritime sector.

A prominent example is activities by Maersk who are leveraging edge computing to streamline supply chain logistics for shipping (Rooney 2022). Recognising that supply chains are physically connected but digitally fragmented, Maersk are leveraging edge computing, 5G networks, and IoT devices at its terminals to elevate the efficiency, quality, and visibility of the container ships Maersk uses to transport cargo across the oceans. Similarly, the Port of Rotterdam is leveraging edge computing and digital twin technologies to enhance connectivity and automation between port operations and assets (Nast 2019). On the other hand, Yara and technology company KONGSBERG have teamed up to build the world's first autonomous and zero-emission container vessel: Yara Birkeland. Building over several years towards fully crewless autonomy, Birkeland leverages edge computing to ensure full vessel autonomy agnostic of connectivity, location, or environmental conditions (Bratić et al. 2019)

Edge computing is increasingly critical for shipping. Ship transport accounts for around 90% of global trade in goods and raw materials (OECD 2023), making the efficiency and speed of large container ships critical to the value chains and production processes of countries worldwide. However, the environmental impact of shipping, including emissions, fuel efficiency, and noise pollution, is becoming increasingly apparent. To address these concerns, emerging technologies like artificial intelligence (AI) and digital twin are appealing for their potential to improve the efficiency, management, and environmental sustainability of shipping routes.

Whilst digitalisation of the transport sector is key to unlocking efficiencies, this is currently hindered by data silos, technological limitations, barriers of complexity, and availability of skilled personnel. The maritime logistics industry has been described as physically connected but digitally fragmented (Kapoor 2023). DT4GS promises to enable stakeholders in shipping to actively **embrace the full spectrum of Digital Twins innovations** to support **smart green shipping** in both the upgrade of existing ships, as well as the building of new vessels.

Application development today is predominantly cloud-native. With the growth of public, private, and on-prem cloud offerings, these concepts have evolved towards a multicloud ecosystem where the developer builds their applications in the same manner regardless of whether they will be deployed in a centralized cloud (HQ) or decentralized edge (on a ship) environment. This introduces the cloud continuum which extends from centralized public cloud services to decentralized edge computing devices where DT4GS solutions are deployed. At one end, ship owners and stakeholders can leverage public cloud services provided by major cloud providers that offer massive, centralized computing power and storage resources that can be accessed from anywhere. On the other end, edge computing enables deployment closer to where the data is generated to provide compute at every step of the ship system. This provides unique benefits and advantages in terms of data sovereignty, security, and privacy, resilience to outages and connectivity constraints, innovation opportunities, flexibility to different providers, as well as

inherent autonomy that are critically important to shipping. Table 3 summarises some of the pros and cons of cloud and edge computing.

Table 3: Summary of the advantages and disadvantages of shipping

Computing Paradigm	Advantages	Disadvantages
Cloud Computing	**Scalability**: Can easily scale resources up or down based on demand. **Cost-Effective:** Reduces the need for on-site hardware and maintenance. **Accessibility**: Allows remote access to computing resources from anywhere. **Advanced Services:** Offers a wide range of services including AI, analytics, and IoT. **Disaster Recovery**: Provides robust data backup and recovery solutions.	**Latency Issues:** Can experience delays due to distance from the server. **Ongoing Costs**: Subscription-based model can be expensive over time. **Security Concerns**: Potential vulnerabilities from shared resources. **Internet Dependency:** Requires a stable internet connection for access. **Limited Customization:** May not fully meet specific operational requirements.
Edge Computing	**Reduced Latency**: Processes data closer to the source, reducing delay. **Bandwidth Savings:** Limits the amount of data that must travel to the cloud, reducing bandwidth usage. **Improved Privacy:** Data can be processed locally, enhancing privacy and security. **Real-Time Processing:** Ideal for applications requiring immediate data processing. **Operational Reliability:** Less reliant on central network, reducing the risk of remote service outages.	**Limited Resources:** Edge devices often have less processing power and storage. **Management Complexity:** More challenging to manage numerous edge devices. **Security Risks:** Increased number of devices can lead to greater security vulnerabilities. **Higher Upfront Costs:** Initial setup and deployment can be costly. **Limited Functionality:** May not offer the full range of services available in cloud computing.

2 DT4GS Cloud to Edge Architecture

There are multiple complexities inherent to ship operations and decision making. Whilst the central complexity may be the difficulty and uncertainty involved in decision making in the marine environment, there are several technological challenges to overcome:

- **Scale:** There are thousands of ships with limited or no technical expertise on board. The solutions implemented must be self-healing and turn-key, requiring minimal intervention and maintenance.
- **Heterogeneity:** Each ship represents a dynamic and unique environment, making it challenging to find a one-size-fits-all model. Solutions need to be adaptable and flexible to cater to the diverse needs and characteristics of each vessel.
- **Data Gravity:** Ships and their instruments generate vast amounts of logs, events, and metric data, often in different formats. Additionally, the sensitivity of some of this data adds another layer of complexity. Handling and processing this data in a secure and efficient manner is crucial.
- **Resource Constraints:** Ships serve as the ultimate edge nodes, characterized by limited communication, computational, and energy resources. These constraints must be considered when designing solutions, ensuring they can operate effectively within the ship's resource limitations.

Fundamentally, edge and cloud technologies promise a solution that seamlessly infuse AI and simulation across the entire spectrum of shipping operations. Agnostic of connectivity or compute capabilities, it provides a continuously available digital assistant to guide all aspects of decision and automation.

2.1 State of the Art in Edge Computing

A forecast by International Data Corporation (IDC) estimates that there will be 41.6 billion IoT devices in 2025, capable of generating 79.4 zettabytes (ZB) of data (Hojlo 2021). The latency, scalability, and compute efficiency of edge computing provide obvious appeal to extract insight from these vast volumes of data. However, there are also multiple other considerations driving edge adoption including security, operational costs, resilience, and flexibility (Rathore et al. 2022).

The global edge computing industry is experiencing rapid growth, driven by advancements in technologies such as 5G, IoT, and the Industrial Internet (Rathore et al. 2022). Edge computing entails extending a consistent computing environment from the datacentre to physical locations proximate to users and data. Similar to a hybrid cloud strategy that enables organizations to run workloads across their internal datacentres and public cloud infrastructure, an edge strategy expands the reach of a cloud environment to numerous additional locations. This approach allows for a distributed computing infrastructure, bringing computing capabilities closer to the edge for enhanced responsiveness, performance, and resilience.

There are three categories of edge use cases (Schabell 2022):

- The first is called **enterprise edge**, and it allows customers to extend application services to remote locations. It entails a core enterprise data store located in a datacentre or as a cloud resource.
- The second is **operations edge**, which focuses on analysing inputs in real time (from Internet of Things sensors, for example) to provide immediate decisions that result in actions. For performance reasons, this generally happens onsite. This kind of edge is a place to gather, process, and act on data.

- The third category is **provider edge**, which manages a network for others, as in the case of telecommunications service providers. This type of edge focuses on creating a reliable, low-latency network with computing environments close to mobile and fixed users.

For the maritime industry, the first two are of relevance. As the volumes of shipping data expands rapidly, it's clear that much of the processing that currently happens in the cloud will happen on the edge to avoid intractable data transfer rates and to ensure data is exploited fully. Similarly, to achieve the benefits of digitalisation and associated efficiency, edge computing is a core component to enhance operational decision making. In the context of industry applications, edge computing involves distribution of some application tasks to a distributed part of the operations. It can be broadly decomposed into two categories:

- **Edge Server**: This typically refers to IT equipment specifically designed for computing tasks at the edge and operating as an extension to the computation tasks performed in the cloud. It can take the form of a half rack comprising 4 or 8 blades, or it can be an industrial PC. Essentially, it is a piece of IT equipment dedicated to edge computing operations.
- **Edge Device**: An edge device is a piece of equipment built for a specific purpose, such as an assembly machine, a turbine, or a car. While these devices were initially designed for their primary functions, they also possess computing capacity. In fact, many devices that were traditionally considered IoT devices now incorporate increased computational power. For instance, an average car today is equipped with approximately 50 CPUs. These devices often run on the Linux operating system, allowing for the deployment of software containers, and enabling edge computing applications to be executed directly on the devices themselves.

Scalable maritime cloud and edge architecture involves distributing compute across 1000s of heterogeneous ships distributed across the globe. Deploying workflows across both cloud and edge resources presents several fundamental challenges for technology solutions:

- Consider the scale of the challenge at hand. With an estimated 15 billion devices in the world today, enterprises are faced with the task of managing thousands of devices within their operations.
- Diversity poses another significant factor to address. These devices come in various forms, each serving a different purpose and running on different operating systems. Managing the diverse range of devices and the specific tasks they perform can be a complex endeavour. In the case of shipping, this can be thought of as the diversity across different geographies and regulatory requirements, different shipping companies and their data sovereignty and process management demands, different ship types and their variation across compute availability and digital sophistication, and the different stakeholders of shipping across individual ships, shipping companies, OEMs, port operations, multi-modal logistics, and regulatory bodies. Different stakeholders require different process optimisations, and different views of ship condition, operations, and efficiency gaps.
- Security is a critical concern when dealing with edge devices. Unlike traditional IT data centres, these devices operate outside the confines of controlled environments. They lack the physical barriers, uniformity, and consistency typically found in hybrid cloud environments that aid in ensuring security. Protecting edge devices from tampering and unauthorized access becomes a priority. This is particularly critical for shipping where unauthorized access to ship data, processes,

and operations can extend across the spectrum from a sensitive personal and commercial information data risk to a national security risk from unauthorized control of mammoth vessels.

- Building a robust ecosystem is essential. The role of edge computing is rapidly expanding and will have a profound impact on enterprise computing, much like how mobile technology has influenced consumer behaviour. Establishing a comprehensive ecosystem that integrates and supports edge devices and their functionalities will be crucial for organizations to maximize the benefits and potential of edge computing. In the shipping industry, there are myriad open-source and proprietary models that are key resource for the digitalisation and modernisation of shipping operations. A prominent example is the Open Simulation Platform (Smogeli et al. 2020), which has become a reference for simulation model development in the maritime industry. It is critical that any digital twin solution interfaces seamlessly with existing frameworks and solutions.
- Maintenance and ongoing support present some unique challenges. Software and AI components are complex and require period patches and updates. Software performance can also be impacted by environmental conditions. Having a rigorous approach for software life-cycle management and drift detection is essential to achieve optimal performance and avoid catastrophic failure where nearby IT support personnel is not available. It is not realistic to expect shipping operators to have ready access to system administrators. Zero-touch provisioning and remote patching are essential to the success of digital twin solutions for shipping.

2.2 Evaluating existing edge-computing solutions

Edge computing is considered part of the cloud continuum and existing cluster management solutions such as Kubernetes can address many of the requirements. However, aspects such as device volumes, heterogeneity, connectivity, and availability make a dedicated management solution preferable. Several solutions have been proposed to address the specific needs of edge computing: including Azure IoT Edge, RedHat Advanced Cluster Management, KubeEdge, KubeStellar, and OpenHorizon.

2.2.1 Azure IoT Edge

Azure IoT Edge is a service that allows deployment of cloud intelligence directly onto local devices by moving analytics and certain cloud functionalities to the edge. This service is designed to run on a wide variety of devices with heterogeneous operating systems, and it helps in reducing latency, allowing for quick local decision-making, and reducing bandwidth use by processing data locally on the edge device itself. The core of Azure IoT Edge is to enable AI, complex event processing, and analytics closer to the physical world where data resides, thereby optimizing the performance of the IoT ecosystem.

At its heart, Azure IoT Edge is anchored by a runtime that operates on each IoT device, managing the deployment and execution of modules, which are container-based implementations of specific workloads or services. These modules can be services such as Azure Stream Analytics, custom Azure Functions, or even your own solution. IoT Edge devices connect to Azure IoT Hub, which provides a centralized way to manage and orchestrate devices and their modules, including the ability to update them remotely. This ensures that devices can operate effectively in offline scenarios, with state and data automatically synchronized once connectivity is restored.

Moreover, Azure IoT Edge integrates seamlessly with various Azure services, ensuring a unified approach to managing device identity, security, and other cloud interactions. The security is enterprise-grade, providing end-to-end protection for both data at rest and data in motion. For developers and companies

looking to implement IoT solutions with advanced on-site processing and immediate responsiveness, Azure IoT Edge presents a robust and versatile platform.

2.2.2 Red Hat Advanced Cluster Management (RHACM)

Red Hat Advanced Cluster Management for Kubernetes provides a centralized platform for managing Kubernetes clusters across a variety of cloud and on-premises environments. In the context of edge computing, RHACM extends its management capabilities to edge deployments, which often involve many clusters spread across disparate locations with limited resources and connectivity.

RHACM simplifies the complexities of running Kubernetes at the edge by offering consistent, automated cluster creation, configuration, and application deployment. It provides the tools necessary to address the unique challenges of edge computing, such as the need for low-latency processing, bandwidth limitations, and the requirement for local autonomy in the event of network outages. With RHACM, operators can manage and scale workloads on thousands of clusters with ease, enforce governance and compliance across all environments, and automate the deployment of applications and updates with advanced policy management and GitOps workflows.

The platform's ability to offer visibility and control over Kubernetes workloads is particularly beneficial for edge scenarios, where maintaining performance and reliability across numerous remote locations is crucial. By leveraging RHACM's comprehensive set of features, organizations can effectively deploy, monitor, and govern their edge computing resources as an integrated part of their overall cloud and data centre strategies, thus reducing the operational burden and increasing efficiency.

2.2.3 KubeEdge

KubeEdge is an open-source system for extending native containerized application orchestration capabilities to hosts at the edge. It is built upon Kubernetes and provides core infrastructure support for network, application deployment, and metadata synchronization between the cloud and edge. It also supports MQTT and allows developers to author custom logic and enable resource constraint devices to communicate at the edge.

The key components of KubeEdge are:

- **CloudCore**: This is the cloud side of KubeEdge which contains various components of the cloud part of KubeEdge. It contains the modules that interact with the cloud platform, like the controller manager, API server, etc. These components are responsible for managing the edge nodes, ensuring that the policies and rules defined on the cloud are enforced on the edge.
- **EdgeCore**: is the edge side counterpart of the CloudCore. It runs on the edge nodes and handles the workloads, synchronization of configuration, ensuring the workloads are running on the edge node as expected. EdgeCore makes the edge node autonomous and ensures the node can still operate in scenarios where the connectivity to the cloud is not stable.
- **EdgeMesh**: provides service mesh capabilities for edge devices. This allows for service discovery and load balancing across the edge nodes, which is critical in a distributed edge environment where devices might have to communicate with one another frequently.
- **DeviceTwin**: maintains the state of edge devices, so the status of the devices can be reported to the cloud, and changes on the cloud side can be synchronized with the edge devices.

- **EdgeHub**: is a web socket client responsible for interacting with CloudCore to handle the communication between the cloud and the edge. It supports MQTT, which means it can also act as a broker for devices that want to communicate through MQTT.
- **MetaManager**: is the message processor between EdgeCore and EdgeHub. It's responsible for storing and synchronizing metadata from the cloud to the edge and vice versa.

KubeEdge allows users to deploy containerized applications to edge devices and manage these applications in a Kubernetes-like fashion while addressing the challenges posed by the edge environment. It can support multiple edge nodes, enabling users to scale their applications across numerous devices.

With the increase in IoT devices and the need for real-time, low-latency processing, KubeEdge represents a significant advancement in the edge computing paradigm, blending the best practices of cloud-native computing with the unique demands of the edge.

2.2.4 KubeStellar

The KubeStellar initiative is dedicated to tackling the complexities of managing configurations across multiple cluster setups, such as those found in edge computing, multi-cloud, and hybrid cloud scenarios. It aims to streamline and address a multitude of factors, including organizational structure, underlying infrastructure, platform management, delineation of duties, architectural integration, and security issues. Furthermore, KubeStellar is designed to manage operations in clusters that experience intermittent connectivity, ensuring that edge environments can function independently of the main infrastructure and other edge sites when necessary.

KubeStellar, while preserving the unaltered essence of Kubernetes (eschewing forking), is tailored to refine edge computing deployments:

- Facilitates the central deployment and orchestration of Kubernetes resources throughout an array of clusters.
- Implements resilient strategies for managing interruptions in cluster connectivity with minimal disruption.
- Engineered for scalable performance, the system supports both one-to-many and many-to-one deployment models.
- Its architecture is modular, designed to maintain interoperability with a suite of prevalent industry-standard tools.

Launched in May 2023 from the KCP-Edge initiative, KubeStellar emerges as an advanced innovation aimed at enhancing the deployment processes within edge computing environments.

2.2.5 Open Horizon

The Linux Foundation's Open Horizon is an open-source platform aimed at facilitating the deployment and management of containerized workloads and related services at the edge of a network. Open Horizon is designed to simplify the orchestration of edge deployments, making it easier for organisations to manage their applications across a multitude of edge devices and environments.

Open Horizon allows for automated deployment and updates of AI models and other workloads directly onto edge devices, which is crucial for scenarios where latency, bandwidth, or privacy constraints preclude the use of cloud computing resources. With its robust and scalable architecture, Open Horizon enables users to securely manage the lifecycle of their edge workloads from a central location, while also supporting autonomous operation at the edge. This means that, even in scenarios with intermittent

connectivity to the central management, edge nodes can continue to function and make decisions independently.

This technology is particularly relevant as the Internet of Things (IoT) expands, with billions of devices generating data that need to be processed in real-time or near-real-time. Industries such as manufacturing, retail, healthcare, and telecommunications can benefit significantly from Open Horizon's ability to manage complex, distributed networks of IoT devices and edge servers. With its open-source approach, Open Horizon also fosters collaboration and innovation, allowing developers and organizations to contribute to the ecosystem and tailor solutions that meet their specific needs without being locked into a single vendor or platform.

The commercial solution IBM Edge Application Manager (EAM) is built on top of Open Horizon. IBM EAM includes added features for enhanced scalability, security, and management, along with an integrated user interface for easier management of edge applications. Further, it includes professional support and services for customers who require more sophisticated capabilities and support.

2.2.6 DT4GS Edge Solution

The imperative for integrating cloud and edge computing solutions to underpin maritime digital twin ecosystems is evident. A survey of extant market solutions indicates that several conform partially to the stipulated project specifications. Given the strategic priority to utilize open-source frameworks, thereby precluding proprietary entanglement, both Azure IoT Edge and Red Hat Advanced Cluster Management (RHACM) were precluded from consideration. Subsequent evaluation of the KubeStellar initiative determined its developmental phase to be nascent, rendering it suboptimal for large-scale implementation. Despite its significant potential, reservations persist regarding its integration as a fundamental component of the project's infrastructure pending its progression to a more advanced stage of development.

This reduced the list to KubeEdge and Open Horizon. To that end, we analysed the functional objectives and architectural frameworks of Open Horizon (OH) and KubeEdge (KE) in greater detail. Both are designed to facilitate the distribution and management of containerized applications across a network of edge devices. A comparison can be done across objectives, architecture, project status, and runtime monitoring:

Objective Alignment:

Both OH and KE share the fundamental goal of enabling automated, targeted container deployment to individual or groups of edge devices within a fleet, obviating the need for manual intervention or direct administrative access to each edge node.

Operational Architecture:

The operational design of both systems adheres to a hub-and-spoke architecture, where an edge agent on the device communicates with a central hub, conducting orchestration tasks and relaying instructions to the local Docker runtime environment.

OH's architecture is comprised of an agglomerate of distributed microservices forming a management hub. These services are network-distributed and, contingent upon a stable network interface, are responsible for a suite of operational capabilities including account management, service policy enforcement, and deployment management.

Conversely, KE operates as an extension of Kubernetes, rendering it intrinsically dependent on the Kubernetes system for its management hub functionalities. The absence of Kubernetes nullifies KE's operational capability.

Developmental Status:

Both projects are subject to continuous development as evidenced by the activity within their respective GitHub repositories. Open Horizon exhibits a higher degree of maturity, given its role as the underpinning technology of IBM's Edge Application Manager (IEAM), thereby asserting its production-ready status. On the other hand, KE, while still in the incubation phase within the Cloud Native Computing Foundation, has a robust contributor base and has been validated through production use cases, albeit on a limited scale.

Monitoring Capabilities:

A critical differential between OH and KE resides in their monitoring functionalities. OH does not inherently provide monitoring tools for its deployed containers. It necessitates users to define key performance indicators, establish monitoring intervals, and develop additional containerized solutions for such surveillance tasks. Nonetheless, OH can deploy such monitoring containers.

KE offers built-in monitoring capabilities, specifically for basic CPU and memory usage statistics of the containers it manages. This information can be accessed sporadically by system administrators or can be incorporated into automated monitoring processes.

Conclusion:

Both Open Horizon and KubeEdge represent robust frameworks for edge computing orchestration, with their own distinct operational paradigms and developmental trajectories. The choice between them depends on the specific requirements of the edge computing environment, including considerations around Kubernetes dependency, the maturity of the solution, and the monitoring requirements of the deployment.

Considering the mission critical nature of shipping and the absolute need for resilience, we selected Open Horizon for its greater maturity and close synergy with the enterprise solution IBM Edge Application Manager.

2.3 Full Stack Edge Solution

Edge solutions require multifaceted capabilities, including the orchestration of heterogeneous workloads, creation of a scalable, extensible, and robust data management pipeline, and seamless deployment of data inference and AI on the edge. The latter consideration is especially critical as many organisations become AI-driven enterprise and infuse generative AI capabilities across their operations (Yusuf 2023). The requirements of workload orchestration, data operations, and AI infusion, include:

- **Edge orchestration:** A fundamental requirement is to enable seamless deployment of containerised applications to multicloud and edge environments. Deploying on 1000s of nodes in a scalable, replicable manner requires deep automation for advanced management of 1000s of different software applications across 1000s of compute nodes distributed across the entire footprint of an organisation's operating remit. Solutions such as Open Horizon ("Open Horizon" 2023) enables the autonomous management of more than 10,000 edge devices simultaneously. Core architecture considerations include:

  - The provisioning of Open Horizon includes the management hub that runs in an instance of OpenShift Container Platform installed in a data centre or headquarter facility. The management hub is where the management of all remote edge nodes (edge devices and edge clusters) occurs.
  - Managed edge nodes can then be installed in remote on-premises locations to make application workloads local to where critical business operations physically occur, such as docks, ships, factories, warehouses, retail outlets, distribution centres, and more.

- **Data Operations (DataOps):** DT4GS Edge consist of disparate devices such as engines, propellors, hulls, berths, shipping containers, etc., that are all generating data. Resilient edge solutions require robust data management and organisational strategies for collecting and handling data, ensuring compliance with data sovereignty regulations, and providing flexible data quality solutions facilitating the training and deployment of AI models. Further, cognisant of data latency restrictions, data processing on the edge must translate from vast volumes of raw data to digestible subsets of high-quality, high-value features that can be used for AI model finetuning and prediction. As an example, computer-vision based hull monitoring solutions generate large volumes of data, while only small subsets of this data may be relevant for guiding hull cleaning and maintenance.
- **AI Operations (AIOps):** AI-backed decision making will play a critical role in improving the sustainability, safety, and efficiency of shipping. Ship captains and stakeholders can use AI to process large data volumes and make decisions related to route selection, navigation, port arrival and logistics, weather-enforced disruption, and mitigation. Further, many aspects can be fully automated such as power management, HVAC system optimisation, cargo and inventory management, predictive maintenance, crew management, and risk analytics. As AI begins to play a central role in shipping, it is critical that 1) ships possess the compute infrastructure to fully exploit these advantages and 2) the AI layer provides reliable inference monitoring mechanisms to assess model performance, detect potential errors or uncertainties, and instil confidence in the decision-making process.

2.3.1 Edge Infrastructure and compute

As edge matures, a critical concern for stakeholders is infrastructure considerations such as server selection for different workloads, security of distributed devices, and loading necessary software onto hardware devices to support various edge applications such as visual inspection, voice interaction, and inference.

Compute capabilities within the shipping and logistics space is an evolving state. While container ships may have some computing equipment for basic tasks such as monitoring and controlling ship systems, these capabilities are generally limited in scope and processing power. The computing infrastructure on a container ship is designed to support essential functions like engine control, navigation, communication, and safety systems.

However, it's worth noting that with the increasing adoption of digital technologies and automation in the shipping industry, there is a growing trend towards incorporating more advanced computing capabilities on certain types of vessels. For example, larger and more advanced container ships may have additional computing systems for tasks like cargo management, route optimization, and fuel efficiency monitoring.

Overall, while there may be some level of compute capability on a typical container ship, it is typically limited and focused on specific operational requirements rather than extensive computing tasks. The bulk of compute-intensive tasks, data processing, and analytics are more commonly performed in onshore data centres or edge computing systems that support the shipping operations. Further, the mission-critical nature of existing applications and risk-averse decision making, means shipping stakeholders are often slow to provide access to onboard computer resources for deploying new applications. This makes it critical to develop stakeholder trust but also to provide solutions tailored to the specific needs of a shipping party across dimensions such as security, cost, scalability, flexibility, and ease of deployment.

Several companies offer edge solutions that provide integrated hardware and software for edge computing deployments. Some examples include:

- Dell EMC Edge Gateway provides a compact and ruggedized device designed for edge computing. It comes with off-the-shelf, pre-configured, pre-certified, ready-to-use products, for different industries or applications (Dell 2023)
- Intel NUC Edge is a compact edge device that promises an out-of-the-box solution ideal for running any critical applications on-premise with immediate high availability (Intel 2023).
- Microsoft Azure Stack Edge is a solution that combines hardware and software to bring AI, analytics, and IoT capabilities to the edge. It includes an on-premises appliance that can be deployed in disconnected or low-connectivity environments and integrates with Azure services for seamless cloud-edge integration (Azure 2023).

When one considers compute infrastructure for shipping, it is important to consider the types of compute capabilities that may be deployed: edge servers and edge devices (described in more detail in Section 1).

2.3.2 Edge platform and orchestration

Edge, by its very nature, generally involves 1000s of devices. Hence, concepts from the data centre do not translate directly to the edge. Instead, it requires software capabilities to monitor and update a limitless number of edge devices from across the world and new security technology and protocols to keep everything safe (Hines 2020).

An edge computing platform contains many components. These include:

- **Edge Management and Orchestration:** This component handles the management, configuration, and physical deployment of edge devices and placement of applications that will run on them. It ensures efficient resource allocation, software updates, and monitoring of edge infrastructure.
- **Security and Authentication:** Edge computing software incorporates security measures to protect edge devices, data, and communications. This includes authentication, encryption, access control, and threat detection capabilities.
- **Edge Gateway:** The edge gateway serves as a bridge between the edge devices and the central cloud or data centre. It provides connectivity, protocol translation, and data aggregation capabilities.
- **Containerization and Virtualization:** To enable portability and ease of deployment, containerisation of software workloads is critical. Software containers or virtual machines encapsulate applications and their dependencies, making them easier to manage and deploy at the edge.

Edge management and orchestration are critical components of edge computing. Edge management refers to the management of edge devices, including device provisioning, configuration, and monitoring. Edge orchestration refers to the deployment of network resources across an edge network by controlling

how network resources flow between devices and applications in an edge environment to produce a more responsive and smartly optimized network.

As an example, Open Horizon deploys an orchestrator called "management hub" in an instance of OpenShift Container Platform installed in the data centre. The management hub is responsible for controlling the placement of containers to all remote edge nodes, both edge servers and edge devices.

Figure 2 provides an overview of an edge computing architecture. The two most critical responsibilities of the orchestrator are the deployment and monitoring of workloads.

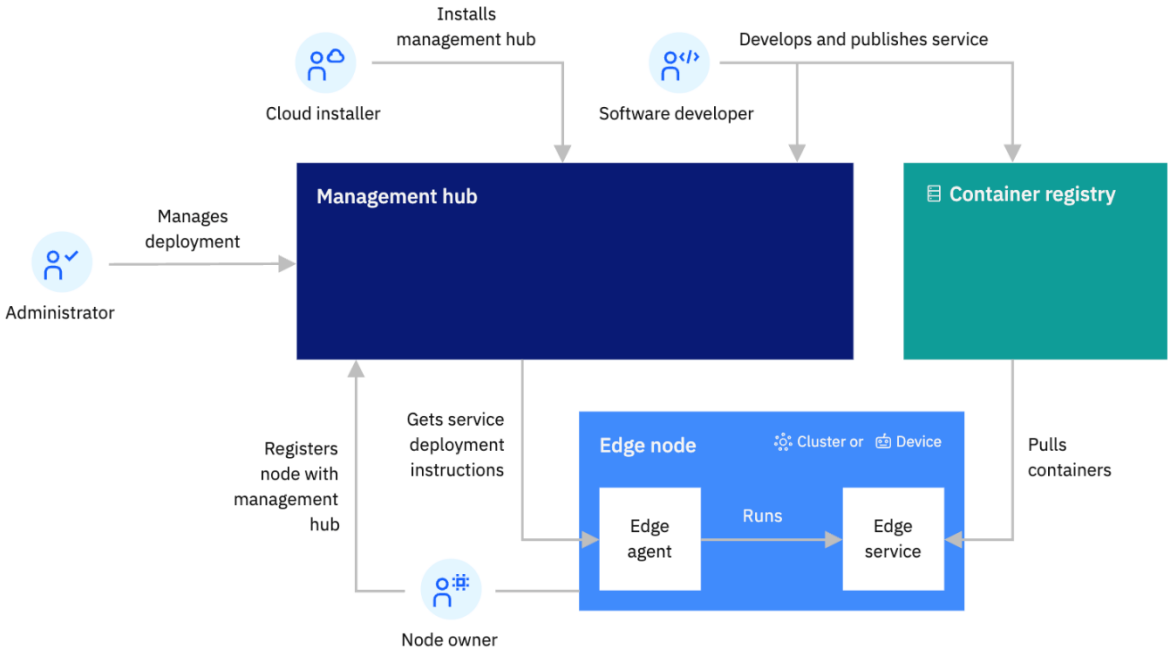

Figure 2: High-level topology for a typical edge computing setup. Taken from https://open-horizon.github.io/

Fundamentally, the edge deployment is based on containerisation of workloads. A Cloud Operations team installs the management hub components. Specific applications are developed by data scientists and domain experts and containerised, which are then published to the management hub edge library. Administrators define the deployment policies that control where edge services are deployed.

Edge orchestrators use various monitoring techniques to ensure that edge devices are functioning correctly and that network resources are being used efficiently. Some examples of monitoring techniques used by edge orchestrators include:

- Device monitoring: Monitor the status of edge devices to ensure that they are functioning correctly. This includes monitoring device health, connectivity, and performance.
- Application monitoring: Monitor the performance of applications running at the edge of the network to ensure that they are meeting performance requirements.

Additionally, monitoring can be extended by user provided services to consider,

- Network monitoring: Monitor network traffic to ensure that network resources are being used efficiently and that there are no bottlenecks in the network. This can also include monitoring of network traffic for security threats and vulnerabilities.
- Data monitoring: Edge orchestrators monitor data flows between devices and applications to ensure that data is being processed correctly and that there are no data integrity issues.

These monitoring techniques help edge orchestrators to identify issues before they become critical and to optimize network resources for better performance. In the context of shipping, it is critical to consider "**Day 0/Day 1/Day 2**" of the software lifecycle:

- Day 0 activities encompass the initial definition, design, procurement and provisioning of the hardware and software solution. For factory pre-install, this stage includes the initial installation and testing of the hardware and software in the factory prior to shipping it to its destination. Additional testing is also performed when the device is wired into the network at its destination and powered up.
- Day 1 marks the actual deployment or launch of the software solution. It represents the first day of production use or operation. On Day 1, the software is made available to users or customers, and the system goes live. This phase involves activities such as testing, data migration, user onboarding, and initial training. The goal of Day 1 is to successfully deploy software created during Day 0 and transition from the development phase to the operational phase and ensure that the software meets the required functionality and performance standards.
- Day 2 refers to the ongoing operational phase of the software after it has been deployed and is in active use. It represents the period of maintenance, support, and continuous improvement. During Day 2, organisations focus on tasks such as monitoring, troubleshooting, bug fixes, performance optimization, regular updates, and feature enhancements. The emphasis is on managing and maintaining the software to ensure its stability, reliability, security, and efficiency throughout its lifecycle.

In the case of shipping, the vast number of deployed edges and the geographical location of each deployment generally makes it prohibitively expensive to provide local IT support at each location to monitor, maintain and update these deployments. Instead, edge technologies must provide a comprehensive solution for administration, managing, monitoring, and securing an almost limitless number of edge servers and devices.

The deployment of thousands of Edge devices means that each of those devices are potential entry points for hackers and security breaches. There are some critical considerations to security on the edge. Some of these include (Iyengar 2023):

- The enterprise should have the ability to check whether the edge nodes are operating properly by comparing the current configuration of various resources against the desired state.
- The communication between an edge agent and management hub should be signed and encrypted.
- Each container run on an edge endpoint should be verified against the official container to check for tampering.
- Each container should be running in its own "docker" network with self-regulated (application dependencies) connectivity between the components.
- Each edge agent should check that it is running the latest version of the container when the endpoint is connected to a network.
- The enterprise should create a cryptographic signing key pair and have a plan to rotate them.

In the shipping industry, security plays a vital role as any unauthorized access or control over a vessel's ecosystem can have catastrophic consequences. Ensuring robust security measures is of utmost importance to prevent potential threats and protect the safety and integrity of maritime operations.

3 DT4GS Headquarters Infrastructure

3.1 Overview of the headquarters testbeds (LLs)

The DT4GS project is focused on crafting strategies for reducing carbon emissions in the maritime sector. This effort involves creating, experimenting with, and confirming innovations through four dynamic testing environments (living labs), represented by EURONAV, DANAOS, BALEÀRIA, and STAR BULK, each symbolizing a different segment of maritime transport (tankers, container ships, ferries, and bulk carriers, respectively). Addressing the significant task of reducing the maritime industry's carbon footprint hinges on tackling three principal aspects:

- Digital Twin (DT) technology for enhancing the operational efficiency of maritime transportation.
- DT applications for the selection and planning of carbon reduction retrofitting strategies for vessels.
- DT-supported tools for the planning of new vessel builds targeting zero emissions by the year 2050.

Consequently, the DT4GS project demands innovative technological solutions that are versatile enough to meet a diverse set of needs for various types of vessels. Deliverable 1.1 breaks down the potential contributions of digital twin technologies in relation to the aforementioned aspects:

- Crafting and implementing Digital Twins for Operational Optimization:
    - Optimisation of voyages. This approach aims to enhance vessel operations and boost efficiency, predominantly through the reduction of fuel oil use, thereby lessening the environmental impact.
    - Advanced event detection for predictive maintenance. This includes: 1. reactive maintenance through the replacement or repair of faulty parts; 2. scheduled routine maintenance; 3. condition-driven maintenance relying on regular checks; and 4. predictive maintenance based on continuous surveillance.
    - Digital Twins for selection and preparation of decarbonization retrofitting initiatives for ships.
- Creation of digital twins to guide the adoption of new, carbon-neutral power generation methods.
    - Digital twins modelling the processes of energy production and conversion in fuel cells, generators, and steam systems to push the boundaries and creativity of future maritime technology.
    - Digital twins for batteries and supercapacitors, aiming to boost the prospects of vessel electrification.
    - Digital twins focused on thermal energy recovery and conversion systems to lower energy consumption.
    - Development of digital twins for eco-friendly propulsion methods, including wings, sails, rotors, kites, and others.
- Digital Twins to enable a zero-emission 2050 Vision for New Vessel Construction:

- Utilizing accumulated operational data and experience to drive the lifecycle enhancement of carbon-neutral ship designs, ensuring they are produced at optimal scales and reduced costs.
- Digital twins for the operation of autonomous, environmentally friendly ships.
- Smart shipbuilding digital twins, encompassing all facets of a ship's lifespan, such as its design, construction, and management.

The formulation of digital twin solutions will be steered by various data inputs. The primary data will be sourced from the vessels themselves, but additional data from office infrastructure and external internet-based data will also be crucial. Figure 3 summarises the key datasets across three different types.

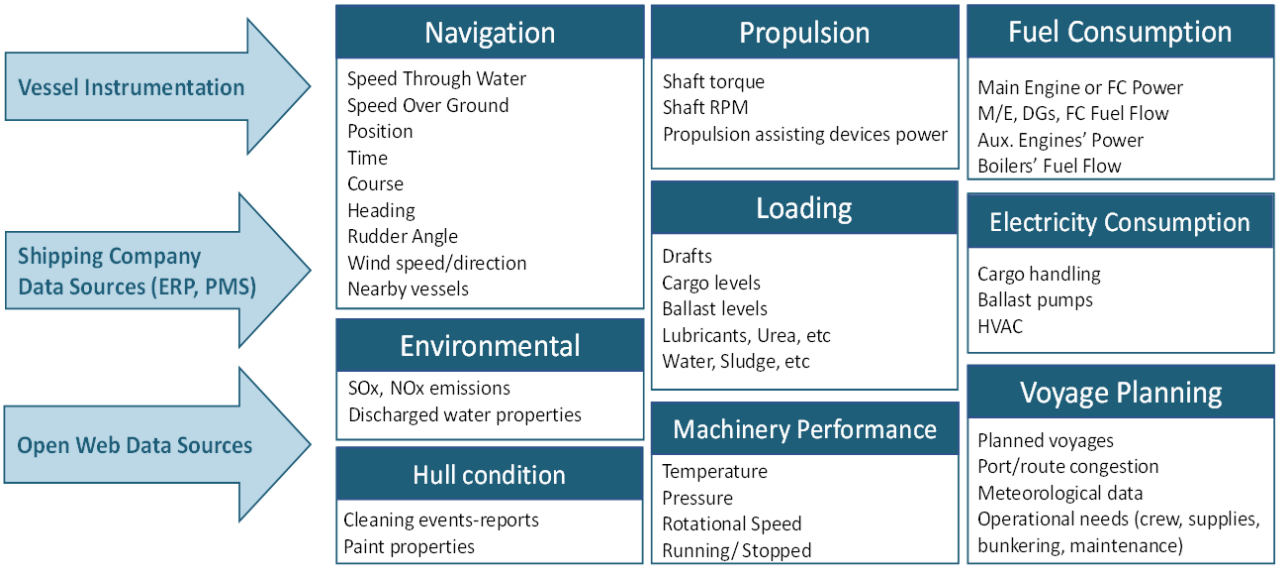

Figure 3: Data required for formulation of digital twin (from Deliverable 1.1)

Hence, requirements from a DT4GS technology perspective are contingent on the ability to handle data from disparate sources and generate actionable insight across many aspects of ship building, management, and operations.

3.2 Service requirements and limitations

Deliverable 2.1 provided a detailed description of the service requirements of the DT4GS living labs that was collected from a series of workshops and user stories. This information is described in more detail in deliverables 4.2 – 4.5 and briefly summarised here. Considering, each in turn, the use cases for the living labs are:

1. *Euronav*
   - Navigation management, route, and hull/propeller optimisation
   - Integrated machinery performance management and remote control
   - Integrated vessel energy production, distribution recovery and management
   - Vessel performance improvements using Just In Time (JIT) arrivals
2. *DANAOS*
   - Navigation management, route, and hull/propeller optimisation.
   - Robotic data and 3D DT construction.

- Evaluation, simulation, and optimization of the flow around the propeller, including possible energy saving devices.
- Machinery performance metrics and management.
- Integrated energy production, distribution, recovery, and management.
- Vessel performance improvements using JIT arrivals.

3. *Balearia*
   - Navigation management, route, and hull/propeller optimisation
   - Integrated machinery performance management and remote control
   - Integrated vessel energy production, distribution recovery and management
   - Vessel performance improvements
4. *StarBulk*
   - Navigation management, route, and hull/propeller optimisation
   - Robotic data and 3D DT construction
   - Integrated machinery performance management and remote control
   - Integrated vessel energy production, distribution recovery and management
   - Integrated biofouling assessment and management

The distinct LLs exhibit notable cooperative potential despite their differences in ship types. Key targets for the living labs include navigation, asset management, energy generation, distribution, and recovery, and the advancement of automation.

Developing a robust digital twin framework mandates the amalgamation of varied data streams and a comprehensive library of models that can simulate the evolving dynamics of diverse vessels and establish baseline performance indicators. This framework must also seamlessly connect with current maritime industry models and solutions, such as the Open Simulation Platform and the Functional Mockup Interface. Consultations with industry users have underscored several necessities for the maritime sector:

- Vessels are equipped with numerous sensors, each with unique identifiers often defined by the OEM. To handle this data efficiently on a large scale, a universal data lexicon is essential. This could be in the form of a reference ontology or a machine learning method that can decipher the sensor tags' underlying meanings.
- The DT4GS solution must be compatible with existing models and frameworks. Given the intricate nature of maritime operations, an expandable open model library must be able to engage with both proprietary and open-source models to reflect the myriad facets of shipping.
- DT4GS should facilitate both ownership and deployment phases of models. Some users will need to implement existing models within the DT4GS infrastructure, while others may seek out-of-the-box solutions from other suppliers.
- It is imperative for models to accurately mirror the real-world mechanics of ship functions. A digital twin setup should include modules for monitoring and evaluating models, ideally integrating physics-based principles and limitations to reinforce known interdependencies.
- DT4GS models must precisely depict the standard conditions of ships to swiftly diagnose deviations and incidents. Condition-based maintenance (CBM), which prescribes maintenance

only upon signs of deteriorating performance or imminent failure, relies on real-time condition monitoring.

- Digital twins should be used to assess the operational implications of novel design alternatives, such as different fuel options. This is particularly true regarding electrification, helping shipowners assess the potential savings from electrification or hybrid engines, and informing electrified navigation to optimise efficiencies and charging schedules
- The adaptability of digital twins is crucial; they must conform to specific vessel conditions, regulatory standards, and internal corporate goals and policies. This involves model precision and the management of uncertainty, ensuring that the model's accuracy is both transparent and suited to the application's importance, with critical missions demanding greater confidence in decision-making.
- Applications must be developed along the "write once, run anywhere" principle. This includes the containerisation of applications to streamline development and aligning with industry tools such as the Functional Mockup Interface.
- DTs must be adaptable to models of differing levels of complexity. From simple statistical processing models to sophisticated 3D simulation models, a holistic DT includes models of different scale, fidelity, and accuracy. The DT4GS framework must be amenable to the deployment of different sized models dependent on situation, user preference, and acceptable model uncertainties.

End-user interaction evidently shows a need for adaptable digital twin technologies tailored to meet individual shipping needs. Responding to these needs, requires a flexible compute infrastructure that adapts to the scale, fidelity, and speed of a given digital twin implementation. It is clear that digital twin for green shipping is a continuum of solutions and it requires an associated continuum of compute resources.

Figure 4 illustrates the essential capabilities and their interactions among onboard computing systems, cloud services, and a container registry or model library. At its core, the setup includes an edge node on each ship, a cloud headquarters (which could be public, private, or hybrid), and a container registry to facilitate application deployments. Critical aspects to consider are the adaptability of application deployments to the computing resources available on each ship, a robust framework for application deployment and management capable of handling intermittent connectivity, and a secure connection between individual nodes and the cloud.

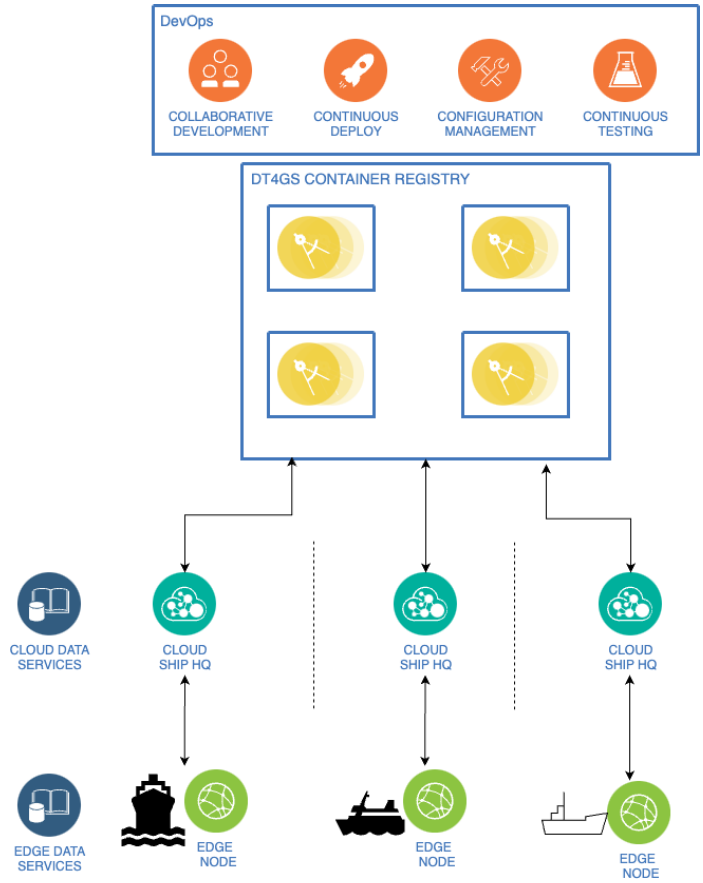

Figure 4: provides a schematic of the Cloud-Edge DT4GS architecture that adapts to the required flexibility. Starting, on the ship, an edge orchestration solution is deployed. This can deploy and manage different applications or edge data services. Each edge data service is a containerised application deployed on the ship infrastructure. It addresses capabilities such as storage, monitoring, prediction, and optimisation. On-ship deployments are typically compute- and communication-constrained. Hence, orchestration of services must consider those factors. Each ship or edge node must securely communicate with the cloud HQ and with a container registry that may be located at HQ, public cloud, or somewhere else. Critically, edge nodes are resilient to connectivity loss between ship and HQ. Hence, a resilient container management solution is required that acts when feasible rather than on a purely schedule- or event-based approach. Finally, a DevOps framework is required to manage updates and changes to the application library deployed on the ship.

3.3 Service Blueprint – Provisioned services on the headquarters

DT4GS builds applications to aid decision making in ship design, management, and operations that will drive reductions in carbon emissions. Since companies typically have more than one type of ship – and ships of the same class may have differing characteristics – these applications will require customization and targeted deployment. Also, applications will be continuously iterated, refined, and updated to reflect changing dynamics. It requires a solution that targets application deployment (and its associated configuration) to a specific ship, a subset of ships, or perhaps all the ships in a company's fleet. Additionally, it is highly desirable to provide critical updates to applications when they are posted rather than waiting for a ship to return to port. Accordingly, such a deployment system would need to operate in a fragile network environment.

For DT4GS we leverage Open Horizon (OH) as an open-source open-architecture solution to deploy applications across available compute resources. OH deploys an orchestrator called "management hub" in an instance of OpenShift Container Platform installed in a data centre or on the locally managed compute resources. The management hub is responsible for controlling the placement of containers to all remote edge nodes, both edge servers and edge devices.

Open Horizon is a client/server application with a centralized server (management hub) and remote edge devices, connected over a computer network that may be constrained in terms of latency or availability. It provides targeted placement of a service (containerized application) or collection of services to a specific registered device, a subset of registered devices, or the entire collection of registered devices under its control. It also allows for targeted updates of deployed services, again with a granularity of a single device up to the entire collection. If a deployed service is suitably constructed, Open Horizon can also facilitate the delivery of data (e.g., update inference models, modify configuration parameters, security patches) to specific services. It can achieve these goals whilst operating over a fragile computer network.

The server section of Open Horizon is composed of a collection of microservices, with the primary components being:

- **exchange-api**: This acts as the primary communication point between the server and its collection of edge devices.
- **cloud sync service**: is responsible for making available any data/models to edge devices
- **agreement bot (agbot):** is responsible for reconciling so called Open-Horizon policies and deciding which services should be deployed to which devices. The server section also leverages some additional microservices for data storage (postgres, mongo) and advanced usage.

In contrast to typical centralized Internet of Things (IoT) platforms and cloud-based control systems, the edge control plane is mostly decentralized. Each role within the system has a limited scope of authority so that each role has only the minimum level of authority that is needed to complete associated tasks. No single authority can assert control over the entire system. A user or role cannot gain access to all nodes in the system by compromising any single host or software component. For Open Horizon, the control plane is implemented by three different software entities (IBM 2021):

- **Horizon agents:** Outnumber all the other actors in Open Horizon. An agent runs on each of the managed edge nodes. Each agent has authority to manage only that one, single, edge node. The agent advertises its public key in Horizon exchange and negotiates with remote agbot processes to manage the local node's software. The agent only expects to receive communications from the agbots that are responsible for deployment patterns within the agent's organization.
- **Horizon agbots (agreement robots):** Are processes that can run anywhere. By default, the processes run automatically. Agbot instances are the second most common actors in Horizon. Each agbot is responsible for only the deployment patterns that are assigned to that agbot. Deployment patterns consist primarily of policies, and a software service manifest. A single agbot instance can manage multiple deployment patterns for an organization. Deployment patterns are published by developers in the context of an Open Horizon managed user organization. The deployment patterns are served by agbots to Horizon agents. When an edge node is registered with Horizon exchange, a deployment pattern for the organization is assigned to the edge node. The agent on that edge node accepts offers only from agbots that present that specific deployment pattern from that specific organization. The agbot is a delivery vehicle for deployment patterns, but the deployment pattern itself must be acceptable to the policies that are set on the edge node by the edge node owner. The deployment pattern must pass signature validation, or the pattern is not accepted by the agent.
- **Horizon exchange:** Refers to a centralized, but geographically replicated and load balanced, service that enables the distributed agents and agbots to join and negotiate agreements. Horizon

exchange also functions as a shared database of metadata for users, organizations, edge nodes, and all published services, policies, and deployment patterns.

To maintain anonymity, the agent and agbot processes share only their public keys throughout the entire discovery and negotiation process. All parties within Open Horizon treat each other as an untrusted entity by default. The parties share information and collaborate only when trust is established, negotiations between the parties are complete, and a formal agreement is established.

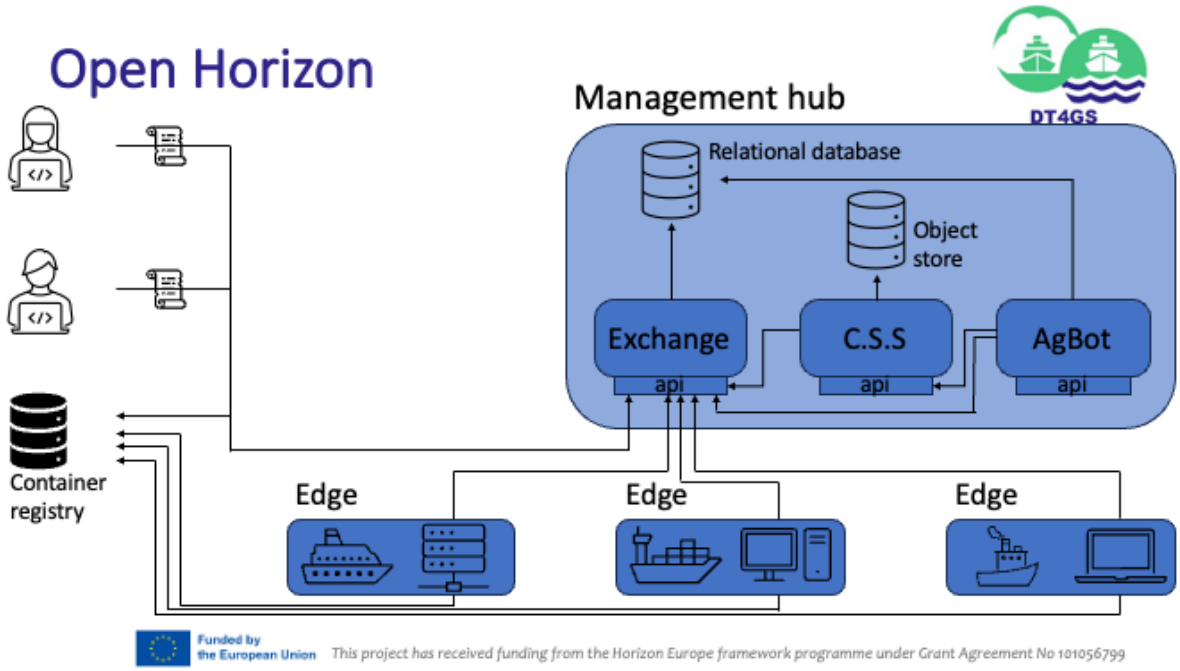

Figure 5: Open Horizon is a client/server application with a centralized server and remote edge devices connected over a computer network. It provides targeted placement of a service (containerized application) or collection of services to a specific registered device, or the entire collection of registered devices under its control. It also allows for targeted updates of deployed services, again with a granularity of a single device up to the entire collection. If a deployed service is suitable constructed. An Open Horizon edge node is a Linux device running the Open Horizon Edge Agent software.

4 DT4GS Far Edge (Vessel) Infrastructure

4.1 Overview of the ship testbeds (LLs)

The living labs, span four distinct types of vessels: tankers, container ships, roll-on/roll-off passenger ships, and bulk carriers. Across this diversity, there are shared factors to consider:

- **Integration with IoT Devices:** Modern ships are outfitted with numerous sensors. The digital twin must be able to integrate with IoT devices to receive real-time data.
- **Data Analysis and Machine Learning:** Utilizing machine learning algorithms can help in predictive maintenance and in analysing data for operational improvements.
- **Regulatory Compliance:** The digital twin must ensure that the ship complies with all international maritime regulations, including those pertaining to emissions and discharge.
- **Cybersecurity:** As digital twins are connected systems, robust cybersecurity measures must be in place to protect them from hacking and other cyber threats.
- **Scalability and Flexibility:** The solution should be scalable to adapt to various ship sizes and flexible enough to integrate new technologies as they become available.

There are further considerations unique to the individual ship types, for example:

- Tankers carry liquid cargo, which can be volatile or environmentally hazardous. The digital twin should include precise monitoring systems for cargo levels, temperature, pressure, and cargo movement within the tanks due to the ship's motion.
- With the size of modern container ships, operational efficiency is critical. The digital twin can be used to optimize routing, speed, and loading/unloading operations to minimize time spent in port.
- RoPax digital twin should model HVAC and other environmental control systems to ensure passenger comfort.
- Bulk carriers often use ballast to maintain stability when unladen. The digital twin should model the ballasting system and its interaction with the vessel's stability.

The different ship types also influence decisions made regarding the digital twin deployment infrastructure. For example, tankers, with sensitive cargo, might require more immediate onboard edge processing, while container ships might manage with a combination of both edge and cloud due to the less sensitive nature of their cargo. The requirements for connectivity can vary greatly, with container ships and bulkers needing less frequent updates than tankers or RoPax ships, which might need constant monitoring for safety reasons. Different ship types may have different bandwidth needs. For instance, passenger ships will have higher bandwidth requirements due to passenger usage, on top of operational data transmission. Compute infrastructure on container ships and bulkers, which often follow transoceanic routes, ideally require self-diagnostic capabilities to minimize the need for in-person maintenance. Aspects such as transporting passengers or hazardous materials as in the case of RoPax and tankers, respectively, may also require robust data integrity solutions to ensure regulatory compliance.

These disparate considerations influence decision making related to the processing of digital twin components, on-board, on-prem, or in the cloud.

4.2 Far Edge Constraints

Deploying DT4GS DTs requires consideration of three components (which are described in more detail in Deliverable 2.1:

- **DT4GS data space:** Generally, lack of interoperability and fragmentation of the different data sharing platforms is a significant concern for supply and logistics operators. A Shipping Dataspace provides the data infrastructure, specifically data connectors, for creating, managing, and interacting with ship digital twins. External data sources may be also linked to the dataspace provided their compliance to the adopted protocols. The main purpose of the Shipping Dataspace is to enable information exchange between the digital twins, in a uniform and secure way, providing them with collective knowledge.
- **DT4GS knowledge graph:** A Knowledge Graph (KG) is a structured representation of knowledge. This provides a unified representation of vessel assets, as well as a semantic layer for the use and encoding of functional metamodel concept.
- **DT4GS model library:** Any model properly registered in the open model library, i.e. conforming with the framework, can be used in any vessel, if it is applicable as denoted by "model applicability".

An effective digital twin requires the seamless integration of these components across the cloud continuum from the edge (ship) to the cloud (either on-prem or public). Aspects such as the data space, knowledge graph, and model library require efficient storage, deployment, update, and management, and seamless availability whether processing happens on the edge or in the cloud.

4.3 Testing Environment

In the domain of software engineering, the delineation between development and production environments is a fundamental tenet for ensuring the reliability and stability of applications upon deployment. Specifically, when considering the deployment of applications within the context of a shipping industry – where the applications are situated on edge devices on the ship – the need for a separate development environment becomes paramount. The development environment is a controlled and experimental space that allows for rigorous testing, debugging, and feature development without impacting the operational stability of the production environment. This segregation is critical, as the production environment in such a setting must exhibit robust performance and constant availability; any disruption could lead to significant logistical challenges, safety concerns, and financial loss. Furthermore, the edge ship environment imposes unique constraints and requirements, such as limited connectivity, which necessitates careful planning and simulation within the development phase to ensure seamless on-deck operations. By maintaining this separation, developers can iterate and innovate without risk to the active shipping operations, ultimately fostering a more efficient, secure, and reliable software life cycle.

In DT4GS, we hosted a virtual environment that allowed us to simulate different compute configurations, representing different computational processing power, different bandwidth availability, different failure modes, and different communication periodicity.

4.4 Service Blueprint – Deployable library on the ship

Open Horizon's effectiveness hinges on its utilization of policy documents. These documents, structured as JSON files with key/value pairs, are interpreted by Open Horizon's 'agbot'. This setup allows nodes

(where services are deployed) to specify their own requirements, establishing Open Horizon's policy mechanism as bidirectional. In this environment, nodes function autonomously, not merely as passive recipients. This leads to policies offering a more nuanced level of control over the deployment of services and models compared to patterns. When policies are active, the management hub actively seeks out suitable nodes for service deployment and continuously monitors existing nodes to ensure they adhere to the established policies. A node is considered 'in policy' if the policies governing the node, service, and deployment that led to its initial service deployment are still valid, or if any changes in these policies do not disrupt their compatibility. The adoption of such a policy-driven approach facilitates a clearer division of responsibilities, enabling edge node owners, service developers, and business operators to define and manage their respective policies independently.

There are three key policy documents:

- edge node policy
- service policy
- deployment policy

The edge node policy is registered by an Edge Node when it initially joins an Open Horizon network. It contains two required sections properties and constraints. See example:

```
{
  "properties": [
    {
      "name": "shipID",
      "value": "ship-id"
    },
    {
      "name": "classID",
      "value": "group-id"
    }
  ],
  "constraints": [
    "capacity > 15"
  ]
}
```

In an edge node policy, `properties` define a list of `name` and `value` pairs. The arguments for these can be any text string and are user defined. There is also a list of read-only system-added properties. These contain information about computer resource, architecture, and other machine information. The final parameter, `constraints` are a list of text encoded constraints which can be used to explicitly declare limitations and requirements. This can be used to ensure that applications are not deployed to nodes that do not have the requisite capabilities.

Open Horizon does not deploy naked applications to Edge Nodes. Instead, application developers containerize their applications using a tool such as docker and push the result to a container registry. They then register the container with Open Horizon using a `service policy`. A service policy contains information such as name, version, and architecture of the application as well as descriptive details on the service that the application provides. An example service policy is:

```
{
  "org": "myorg",
  "label": "sample service text description",
  "url": "service-name",
  "version": "0.0.1",
  "arch": "amd64",
  "public": true,
  "sharable": "singleton",
  "requiredServices": [],
  "userInput": [],
  "deployment": {
    "services": {
      "service-name": {
        "image": "docker-registry/amd64-sample-service:0.0.1"
      }
    }
  }
}
```

The final piece is the `deployment policy`. This contains a reference to the service to be deployed including versioning and their priority, a list of constraints which is used to identify potential edge nodes, and user input for the container delivered as environment variable in the container. For example,

```
{
  "label": "",
  "description": "",
  "service": {
    "name": "service-name",
    "org": "myorg",
    "arch": "amd64",
    "serviceVersions": [
      {
        "version": "0.0.1",
        "priority": {
          "priority_value": 3,
          "retries": 2,
          "retry_durations": 60
        }
      }
    ]
  },
  "properties": [],
  "constraints": [
    "shipID == \"ship-id\""
  ],
  "userInput": [
    {
      "serviceOrgid": "myorg",
      "serviceUrl": "service-name",
      "serviceVersionRange": "0.0.1",
      "inputs": [
        {
          "name": "ENV_NAME1",
          "value": "ENV_VALUE1"
        },
        {
          "name": "ENV_NAME2",
          "value": "ENV_VALUE2"
        }
      ]
    }
  ]
}
```

Once the user defines and submits a deployment policy it is evaluated by the agbot. Based on the deployment policy it determines which version of which service should be deployed to which edge node by examining each device's node policy. The agbot then informs the exchange-api of its deliberations. The next time an Edge Node contacts the exchange-api, it may receive notification of a new service to run. The Edge Node then contacts the container registry directly, retrieves the container, and runs it.

5 DT4GS Enhanced Intelligence in data collection, service provisioning and orchestration

5.1 Data collection optimisation – state of the art

A central part of any software development pipeline is a DevOps framework that provides a unified software development (Dev) and software operation (Ops) approach. It promotes a collaborative and automated approach to build, test, and release software faster and more reliably. It emphasizes continuous improvement, efficient workflows, and the integration of development and operations teams to enhance the quality and speed of delivering software.

Similarly in a complex use case like digital twin for shipping a robust data quality pipeline is critical to enhance the fidelity of data and identify anomalies. DataOps is an agile methodology focused on improving the speed, accuracy, and quality of data analytics by integrating engineering, scientific, analytic, and domain information to streamline the design, development, and maintenance of data pipelines. It emphasizes automation, continuous integration and delivery, and collaboration to manage the lifecycle of data flows and enable more efficient and effective data-driven decision-making.

Validating, documenting, and profiling data are essential practices in data management that play a crucial role in ensuring the quality and integrity of data, which is the cornerstone of reliable analytics and informed decision-making. Data validation involves checking data for accuracy and quality against predefined criteria or standards, which helps in identifying and rectifying errors or inconsistencies at an early stage. This process is vital in preventing the propagation of erroneous data through the pipeline, which could lead to misleading analysis results, flawed business decisions, and loss of credibility. Documenting data, on the other hand, involves keeping detailed records of data sources, structures, transformations, and associated validations. This not only provides clarity and context to the data but also enhances collaboration among team members, ensuring everyone has a consistent understanding of the data's characteristics and lineage.

Each of the living labs offer comprehensive details about their data, including defined minimum and maximum values for each sensor tag, which is instrumental in enhancing data validity and ensuring that the data received from sensors adhere to these specified ranges. For these projects, we utilized Great Expectations, an open-source Python library designed for data validation, documentation, and profiling, to confirm the quality and integrity of our data. This library employs checkpoints, a method that is both transparent and automatable, to verify if the data meets the set or calculated expectations. An example of an expectations suite created for DT4GS is depicted in Figure 6. Here, the minimum and maximum values were initially determined based on the information from the living labs, although they can also be established by calculating statistical outlier boundaries.

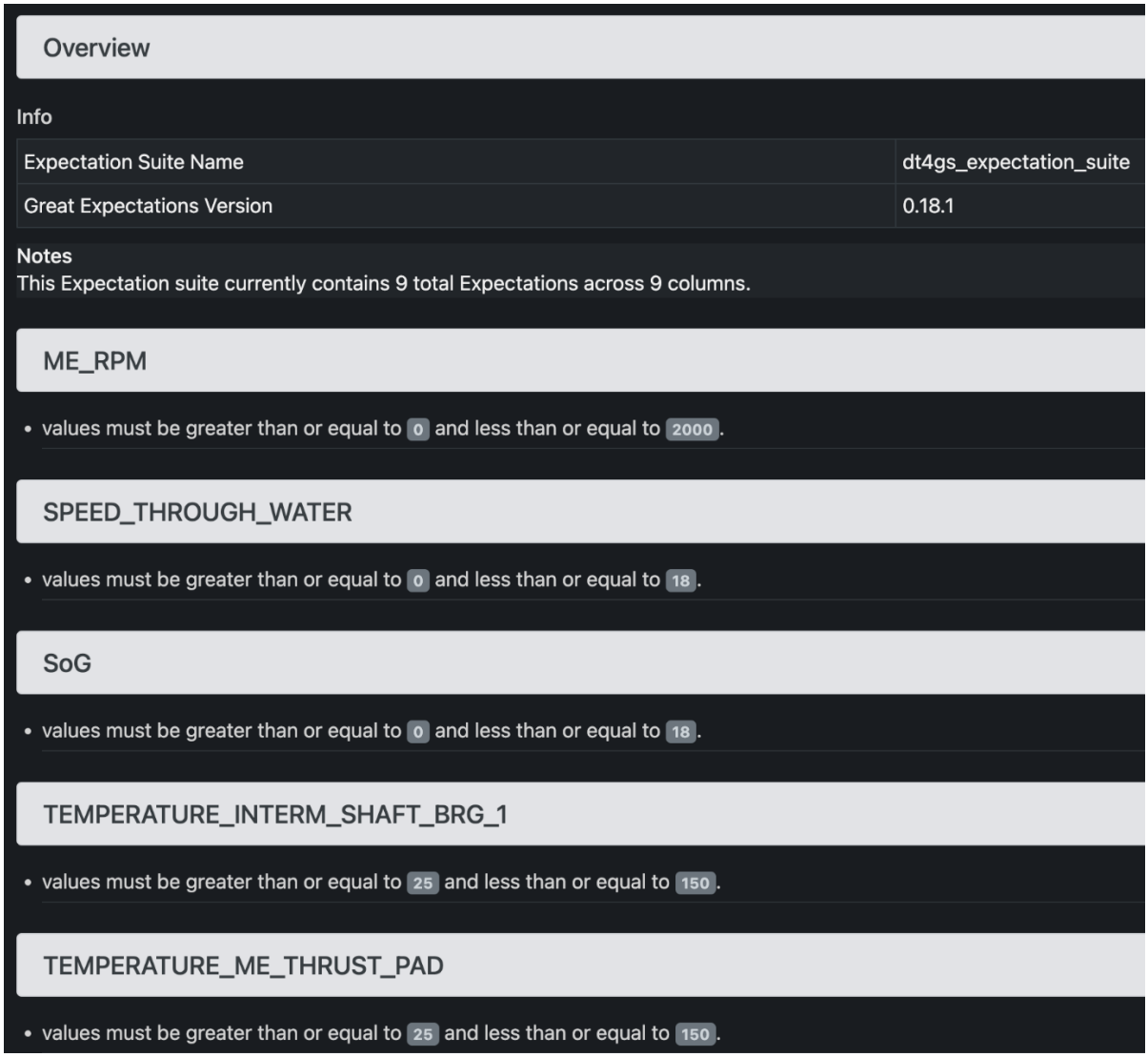

Figure 6: A data unit testing framework for DT4GS that asserts whether the time series data is withing the prescribed bounds. The data covers a 1-year period at 5-minute intervals. While many sensor tags are returned. For demonstration purposes, we consider a subset. The min and max thresholds are defined based on values provided by the Living Labs.

Figure 7 presents results from one iteration of a data quality assessment. The results from the Great Expectations pipeline, revealing that 5 out of 9 datasets failed due to values outside the expected bounds, underscore the common challenge of dealing with imperfect data in real-world scenarios. This outcome highlights the inherent variability and unpredictability of data, emphasizing the necessity of having robust strategies in place for managing and intelligently filtering anomalous data. It's an opportunity to refine our data processing and validation methods, ensuring that they are not only stringent but also flexible enough to discern between true data quality issues and acceptable variances. Such instances reinforce the importance of continuously adapting our data handling approaches, including refining the criteria used for anomalies and outliers. By doing so, we can maintain the integrity and reliability of our datasets, while accommodating the nuanced nature of real-world data. This balanced approach is key to extracting meaningful and actionable insights from the data, even when it doesn't strictly adhere to predefined expectations.

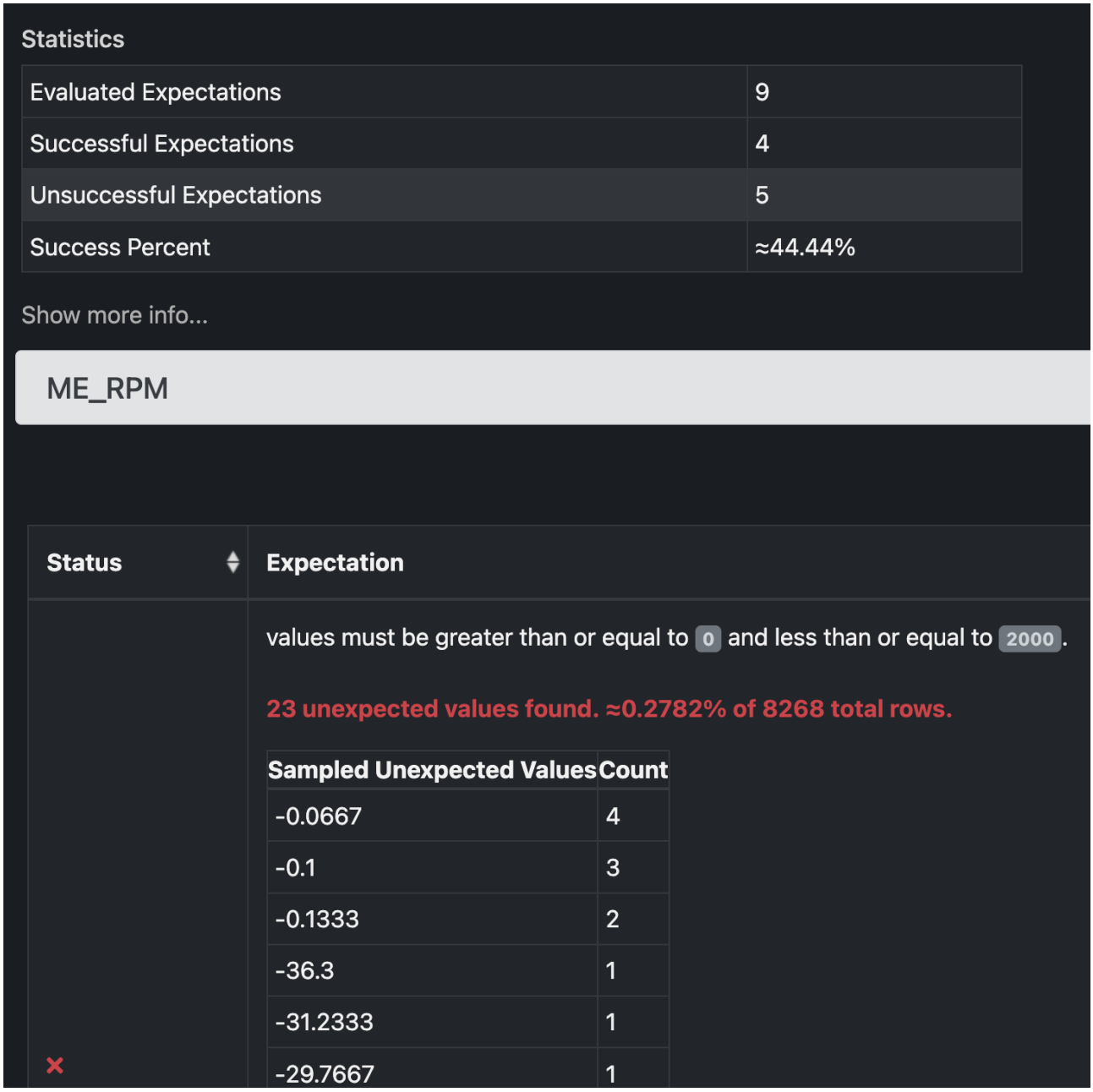

Figure 7: Results from Great Expectations data quality testing framework. Results show that for the 9 expectations specified (as shown in Figure 6), 5 failed due to returning values outside the expected range. Note, of course that doesn't indicate low quality data but rather that some data values should be further interrogated and possibly excluded from model development.

5.2 AI-assisted service provisioning and orchestration solutions

Since the DT4GS project began in May 2022, the field of AI has experienced a tremendous surge of interest strongly influenced by the launch of ChatGPT. The field has been described as reaching its "Netscape moment," indicating a significant breakthrough (Kahn 2023). This has led to the rapid development and deployment of large AI models by organizations, with a focus on automating and enhancing various aspects of their operations.

Foundation models, also known as pre-trained language models, have emerged as a fundamental component in this AI revolution (Bommasani et al. 2021). These models serve as the building blocks for a

wide range of natural language processing (NLP) tasks. They are trained on extensive amounts of text data from the internet and are specifically designed to comprehend and generate human-like text.

Built on transformer architectures, foundation models excel at capturing contextual relationships between words, enabling them to generate coherent and contextually relevant responses. They serve as the starting point for developing task-specific models, thus earning the term "foundation." While they have gained recognition for their performance in language-related tasks, they can also be customized for various domains beyond NLP, including computer vision, IT operations, and industry applications. For instance, IBM Research has developed a geospatial foundation model that learns from raw satellite imagery to create customized maps of natural disasters and environmental changes (Raghavan and Shim 2021). This pre-trained model can be fine-tuned for downstream applications like disaster response, supply chain logistics, and agriculture.

By leveraging foundation models, developers and researchers can save significant time and resources that would otherwise be required to build and train models from scratch. These models provide a robust starting point with generalizable capabilities, which can be further refined and adapted to specific applications or industries.

In the context of DT4GS, foundation models can be a central tool to address the scale and heterogeneity of applications. This can be best addressed by considering two use cases:

1. **Streamlining deployments on the edge**: The key to Open Horizon's functionality is the use of JSON policy documents. The core documents are edge node, service, and deployment policy, detailing all the edge nodes being managed, the applications stored in the relevant container registry, and the process to match applications to edge node, respectively. As the number of edge nodes and applications grow, creating, updating, and managing these JSON files becomes cumbersome. Large language models, like GPT, have the potential to significantly streamline the process of writing these files. By understanding the syntax and semantics of configuration files, these models can automatically generate accurate and efficient scripts based on user-provided specifications or requirements. For instance, when deploying multiple container applications to multiple ships with different characteristics and compute infrastructure, a user could describe the desired configuration in natural language, and the AI model could translate this description into a well-structured policy document. This capability not only saves time and reduces the potential for human error but also allows for quick customization and scalability in complex deployment environments. Language models can further assist by validating configurations against best practices, suggesting optimizations, and ensuring compatibility with specific deployment environments.
2. **Automating AI model development**: Developing digital twin capabilities for shipping requires an extensive model library to represent the disparate processes that take place on a ship. One requires the ability to translate the 100s of datasets coming from different IoT devices into predictions informing on the condition, status, and implications of these observations. Increasingly, transformer architecture are demonstrating ability to generalise across time series data from multiple domains. Examples of this include:
    - Transformer models demonstrate beyond state-of-the-art performance across multiple timeseries applications domains (Nie et al. 2023).
    - Tokenising time series as numerical strings and feeding to best-of-breed LLMs (e.g. GPT-3 and LLaMA-2) to make next-token prediction demonstrates performance on-par with bespoke time series models (Gruver et al. 2023).

- (Jin et al. 2023) leveraged a Prompt-as-a-Prefix approach that acts as a prefix to enrich the input context and guide the transformation of reprogrammed time series patches.
- (Zhou et al. 2023) repurposed an existing pre-trained foundation model to various timeseries forecasting tasks by redesigning and training the input embedding layer
- (Chang, Peng, and Chen 2023) leveraged pre-trained foundation model (LLMs) to enhance time-series forecasting building on the growing interest in multi-modal approaches that unify Natural Language Processing and Computer Vision.
- (Cao et al. 2023) developed a pool of prompts stored as distinct key-value pairs to leverage related past experiences, where similar input time series tend to retrieve the same group of prompts from the pool together with classic time series forecasting techniques to improve prediction.

In both application scenarios, leveraging pretrained foundation models or the fundamental transformer architecture can expedite and enhance the development of digital twins. These elements will constitute primary areas of concentration during the latter half of the project.

6 Conclusions

This deliverable describes the edge and cloud technical architecture for the DT4GS project. Fundamentally, the technical architecture is driven by the living labs use cases and their specific requirements. We explored the requirements across each of the living labs based on several meetings, reports, and deliverables produced during the first 18 months of the DT4GS project.

Based on these requirements, we reviewed multiple technology solutions, including Kubernetes, Red Hat Advanced Cluster Management, KubeEdge, Azure IoT, and Kubestellar. Based on the requirements analysis across the four living labs, the OpenHorizon open-source solution provides closest alignment to DT4GS objectives.

The deliverable details the technical specifications of Open Horizon, the key requirements for shipping, the selected implementation for DT4GS, considerations to deploy DT4GS applications on the edge (ship) and in the cloud (HQ), the process to instantiate Edge and Cloud resource, system administration and management, orchestration of resources in the cloud, and finally orchestration and management of resources on the Edge. We assess aspects such as usability and scalability for the project. A particular requirement for shipping is that digital solutions are easy to install and manage, ideally with zero or minimal expertise required at the ship facility.

A key contribution of this deliverable is the analysis of compute requirements for sophisticated digital twin capabilities for shipping. It considers the different characteristics of the four living labs and the requirements in terms of scalability, resilience, usability, and system admin cognisant of factors such as typical route, operational requirements, available skills and personnel, security considerations, and connectivity.

This deliverable focuses on the cloud and edge architecture to support digital twin technologies for shipping. This extends across supporting the DT4GS data space, knowledge graph implementation, and the model library. We detail the capabilities to support these components. Further, we explore the

role of AI to enhance digital twin penetration in shipping. Specifically, we focus on two aspects:

- The opportunity to leverage AI to enhance the deployment, placement, and orchestration of applications on edge and cloud resources. This is a combination of pretrained foundation models and natural language prompts to streamline generation of policy documents that controls the placement of applications on the edge and cloud architecture.
- The use of pretrained foundation models to enhance the digital twin model library for shipping. Creating foundation models with the capability to handle multiple different tasks can reduce the total number of models required and simplify deployment of models by combining natural language prompts with prediction tasks.

This document is the first version of the deliverable. In the second version, D2.9 (M34), we will provide more details on AI-backed enhancements to address the specific needs and requirements of digital twin for green shipping.

References

Azure. 2023. "Azure Stack Edge | Microsoft Azure." 2023. https://azure.microsoft.com/en-us/products/azure-stack/edge.

Bommasani, Rishi, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, and Emma Brunskill. 2021. "On the Opportunities and Risks of Foundation Models." *arXiv Preprint arXiv:2108.07258*.

Bratić, Karlo, Ivan Pavić, Sr\djan Vukša, and Ladislav Stazić. 2019. "A Review of Autonomous and Remotely Controlled Ships in Maritime Sector." *Transactions on Maritime Science* 8 (02): 253–65.

Cao, Defu, Furong Jia, Sercan O. Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. 2023. "TEMPO: Prompt-Based Generative Pre-Trained Transformer for Time Series Forecasting." arXiv. http://arxiv.org/abs/2310.04948.

Chang, Ching, Wen-Chih Peng, and Tien-Fu Chen. 2023. "LLM4TS: Two-Stage Fine-Tuning for Time-Series Forecasting with Pre-Trained LLMs." arXiv. http://arxiv.org/abs/2308.08469.

Dell. 2023. "New Dell Edge Gateway 5200 | Dell Ireland." Dell. 2023. https://www.dell.com/en-ie/shop/gateways-embedded-computing/dell-emc-edge-gateway-5200/spd/dell-edge-gateway-5200/emea_edgegateway5200.

Gardner, Nic, Matthew Kenney, and Nick Chubb. 2021. "A Changed World: The State of Digital Transformation in a Post COVID-19 Maritime Industry." Inmarsat Research Programme. https://www.inmarsat.com/en/insights/maritime/2021/state-of-digital-transformation-post-covid-maritime.html.

Google. 2023. "What Is a Hybrid Cloud?" Google Cloud. 2023. https://cloud.google.com/learn/what-is-hybrid-cloud.

Gruver, Nate, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. 2023. "Large Language Models Are Zero-Shot Time Series Forecasters." arXiv. http://arxiv.org/abs/2310.07820.

Hines, Michael. 2020. "What Does the Future Hold for Edge Computing?" 2020. https://builtin.com/cloud-computing/future-edge-computing.

Hojlo, Jeffrey. 2021. "Future of Industry Ecosystems: Shared Insights & Data | IDC Blog." 2021. https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/.

IBM. 2021. "IBM Edge Application Manager." March 2, 2021. https://www.ibm.com/docs/en/eam/4.1.

Intel. 2023. "Intel NUC Enterprise Edge Compute Edition Built with Scale Computing." Scale Computing. 2023. https://www.scalecomputing.com/landing-pages/intel-nuc-enterprise-edge-compute-edition.

Iyengar, Ashok. 2023. "Security at the Edge." March 24, 2023. https://www.ibm.com/cloud/blog/security-at-the-edge.

Jin, Ming, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, et al. 2023. "Time-LLM: Time Series Forecasting by Reprogramming Large Language Models." arXiv. http://arxiv.org/abs/2310.01728.

Kahn, Jeremy. 2023. "The inside Story of ChatGPT: How OpenAI Founder Sam Altman Built the World's Hottest Technology with Billions from Microsoft." Fortune. 2023. https://fortune.com/longform/chatgpt-openai-sam-altman-microsoft/.

Kapoor Navneet. 2023. "Technology at Maersk." 2023. https://www.maersk.com/about/technology.

Nast, Condé. 2019. "Rotterdam Is Building the Most Automated Port in the World." *Wired UK*, 2019. https://www.wired.co.uk/article/rotterdam-port-ships-automation.

Nie, Yuqi, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. "A Time Series Is Worth 64 Words: Long-Term Forecasting with Transformers." arXiv. http://arxiv.org/abs/2211.14730.

OECD. 2023. "Ocean Shipping and Shipbuilding - OECD." 2023. https://www.oecd.org/ocean/topics/ocean-shipping/.

"Open Horizon." 2023. Open Horizon. 2023. https://open-horizon.github.io/.

Raghavan, Sririam, and Christina Shim. 2021. "A New AI Model Could Help Track and Adapt to Climate Change." IBM Research Blog. February 9, 2021. https://research.ibm.com/blog/geospatial-models-nasa-ai#fn-1.

Rathore, Vijay Singh, Vijeta Kumawat, B. Umamaheswari, and Priyanka Mitra. 2022. "Edge Computing: State of Art with Current Challenges and Future Opportunities." In *Rising Threats in Expert Applications and Solutions*, edited by Vijay Singh Rathore, Subhash Chander Sharma, Joao Manuel R.S. Tavares, Catarina Moreira, and B. Surendiran, 131–38. Lecture Notes in Networks and Systems. Singapore: Springer Nature. https://doi.org/10.1007/978-981-19-1122-4_15.

Rooney, Paula. 2022. "Maersk Embraces Edge Computing to Revolutionise Supply Chain." 2022. https://www.arnnet.com.au/article/704006/maersk-embraces-edge-computing-revolutionise-supply-chain/.

Schabell, Eric D. 2022. "5 Reference Architecture Designs for Edge Computing." Enable Architect. Red Hat, Inc. May 4, 2022. https://www.redhat.com/architect/edge-portfolio-architecture.

Smogeli, Øyvind Rasmussen, Kristine Bruun Ludvigsen, Levi Jamt, Bjørnar Vik, Håvard Nordahl, Lars Tandle Kyllingstad, Kevin Koosup Yum, and Houxiang Zhang. 2020. "Open Simulation Platform– An Open-Source Project for Maritime System Co-Simulation." In *19th International Conference on Computer and IT Applications in the Maritime Industries*. Technische Universität Hamburg-Harburg. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2723258.

Yusuf, Kareem. 2023. "Introducing Watsonx: The Future of AI for Business." *IBM Blog* (blog). May 9, 2023. https://www.ibm.com/blog/introducing-watsonx-the-future-of-ai-for-business/.

Zhou, Tian, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. 2023. "One Fits All: Power General Time Series Analysis by Pretrained LM." *arXiv Preprint arXiv:2302.11939*.

Annex I

This section describes the install of the DT4GS cloud and edge solution for shipping. It includes:

- quickstart install of non production environment open horizon server
- quickstart install/registration of open horizon client on an edge node
- deploy a 'hello world' container
- deploy a long running container
- deploy a collection of containers
- update a deployed container

Quickstart, setup server

based on https://open-horizon.github.io/quick-start/

Create an ubuntu 22.04 amd64 vm, calling it `mgmt.vm.com` (say). No need to install docker, the quickstart script will manage that.

- note: the "server" is a collection of containers which communicate with each other over the network.
- note: `user` is a user account on the vm with `sudo` privileges.

```
ssh user@mgmt.vm.com
sudo su
curl -sSL https://raw.githubusercontent.com/open-horizon/devops/master/mgmt-hub/deploy-mgmt-hub.sh -o ./deploy-mgmt-hub.sh
```

edit `./deploy-mgmt-hub.sh`, changing,

- `export HZN_LISTEN_IP=${HZN_LISTEN_IP:-127.0.0.1}` to `export HZN_LISTEN_IP=${HZN_LISTEN_IP:-0.0.0.0}`
- `export MONGO_IMAGE_TAG=${MONGO_IMAGE_TAG:-latest}` to `export MONGO_IMAGE_TAG=4.0.6`

Then,

```
chown user:user ./deploy-mgmt-hub.sh
chmod 755 ./deploy-mgmt-hub.sh
./deploy-mgmt-hub.sh
usermod -aG docker user
```

Record the final few lines of output, and exit out of the `root` shell back to the user shell

```
----------- Summary of what was done:
  1. Started Horizon management hub services: agbot, exchange, postgres DB, CSS, mongo DB, vault
  2. Created exchange resources: system org (IBM) admin user, user org (myorg) and admin user, and agbot
    Automatically generated these passwords/tokens:
      EXCHANGE_ROOT_PW=pFb0...
      EXCHANGE_HUB_ADMIN_PW=HZZu...
      EXCHANGE_SYSTEM_ADMIN_PW=W2bu...
      AGBOT_TOKEN=K13w...
      EXCHANGE_USER_ADMIN_PW=HCuP...
      HZN_DEVICE_TOKEN=gk1Y...
    Important: save these generated passwords/tokens in a safe place. You will not be able to query them from Horizon.
```

```
  3. Installed and configured the Horizon agent and CLI (hzn)
  4. Created a Horizon developer key pair
  5. Installed the Horizon examples
  6. Created and registered an edge node to run the helloworld example edge service
  7. Created a vault instance: http://0.0.0.0:8200/ui/vault/auth?with=token
    Automatically generated this key/token:
      VAULT_UNSEAL_KEY=4M4m...
      VAULT_ROOT_TOKEN=hvs....
    Important: save this generated key/token in a safe place. You will not be able to query them
from Horizon.
  8. Added the hzn auto-completion file to ~/.bashrc (but you need to source that again for it to
take effect in this shell session)

For what to do next, see: https://github.com/open-horizon/devops/blob/master/mgmt-hub/README.md#a
ll-in-1-what-next
Before running the commands in the What To Do Next section, copy/paste/run these commands in your
terminal:
 export HZN_ORG_ID=myorg
 export HZN_EXCHANGE_USER_AUTH=admin:HCuP...
```

Run `docker ps` to show what was started,

```
user@mgmt:~$ docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED
STATUS                  PORTS
NAMES
fcf1fb17afea   openhorizon/amd64_agbot:latest                 "/bin/sh -c /usr/hor…"   21 hours ag
o   Up 20 hours (healthy)   127.0.0.1:3110->8080/tcp, 0.0.0.0:3111->8083/tcp
agbot
cf65e14383e2   openhorizon/amd64_vault:latest                 "entrypoint.sh server"   21 hours ag
o   Up 21 hours (healthy)   0.0.0.0:8200->8200/tcp
vault
90070de59fad   openhorizon/amd64_cloud-sync-service:latest    "/usr/edge-sync-serv…"   21 hours ag
o   Up 21 hours (healthy)   0.0.0.0:9443->8080/tcp
css-api
0c8fd9e9cdf9   openhorizon/sdo-owner-services:latest          "/bin/sh -c $WORKDIR…"   21 hours ag
o   Up 21 hours (healthy)   0.0.0.0:8040->8040/tcp, 0.0.0.0:8042->8042/tcp, 0.0.0.0:9008->9008/tc
p   sdo-owner-services
1786e00554eb   openhorizon/amd64_exchange-api:latest          "/bin/sh -c '/usr/bi…"   21 hours ag
o   Up 21 hours (healthy)   8083/tcp, 0.0.0.0:3090->8080/tcp
exchange-api
42ec032ffa13   postgres:13                                    "docker-entrypoint.s…"   21 hours ag
o   Up 21 hours (healthy)   5432/tcp
postgres
228c26566df8   mongo:4.0.6                                    "docker-entrypoint.s…"   21 hours ag
o   Up 21 hours (healthy)   27017/tcp
mongo
```

To interact with the `open-horizon` management hub,

```
export HZN_ORG_ID=myorg
export HZN_EXCHANGE_USER_AUTH=admin:HCuP...

hzn exchange status
```

Other commands,

```
hzn  --help
hzn  <subcommand>  --help

# exchange specific commands
hzn  exchange  --help
```

Quickstart, adding an additional edge node

Create another ubuntu 22.04 amd64 vm, calling it `edge.vm.com` (say). On this vm, `docker`, the `hzn` command line client, and the `hzn` deamon will be installed.

install docker on edge node

```
# install docker
ssh user@edge.vm.com
sudo su

apt-get  update
apt-get  upgrade

apt-get  remove  docker  docker-engine  docker.io  containerd  runc
apt-get  install  ca-certificates  curl  gnupg

install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o /etc/apt/keyrings/dock
er.gpg
chmod a+r /etc/apt/keyrings/docker.gpg

echo   "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://
download.docker.com/linux/ubuntu "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" |   s
udo tee /etc/apt/sources.list.d/docker.list > /dev/null

apt-get  update
apt-get  install  docker-ce  docker-ce-cli  containerd.io  docker-buildx-plugin  docker-compose-p
lugin
usermod  -aG  docker  user
```

install open-horizon cli and daemon on edge node

```
# install hzn cli and daemon (tested with version 2.30.0)
ssh user@edge.vm.com

mkdir pkg
cd pkg
curl  -LO  https://github.com/open-horizon/anax/releases/download/v2.30.0-1435/horizon-agent-linu
x-deb-amd64.tar.gz

tar -xf horizon-agent-linux-deb-amd64.tar.gz

sudo apt install jq
sudo dpkg -i horizon-cli_2.30.0-1435_amd64.deb
sudo dpkg -i horizon_2.30.0-1435_amd64.deb

sudo systemctl status horizon
sudo systemctl stop horizon
```

configure horizon daemon on edge node

edit the config file `/etc/default/horizon`,

```
HZN_EXCHANGE_URL=http://mgmt.vm.com:3090/v1
HZN_FSS_CSSURL=http://mgmt.vm.com:9443
HZN_AGBOT_URL=http://mgmt.vm.com:3111
HZN_SDO_SVC_URL=http://mgmt.vm.com:9008/api
```

```
HZN_DEVICE_ID=edgevm
ANAX_LOG_LEVEL=3
```

and restart,

```
sudo systemctl start horizon
sudo systemctl status horizon
```

Use the command line tool to check,

```
export HZN_ORG_ID=myorg
export HZN_EXCHANGE_USER_AUTH=admin:HCuP...

hzn exchange status
```

Remember this is running on the newly created edge node, connecting to the management hub on `mgmt.vm.com`

Troubleshooting

- ensure ports 3090, 9443, 3111, 9008 are open on `mgmt.vm.com`

Register this edge node with the management hub. This is usually a one time operation carried out on the edge node,

```
mkdir ~/hzn
cd ~/hzn
cat << EOF > node-policy.json
{
  "properties": [
    {
      "name": "edge.id",
      "value": "edgevm"
    }
  ],
  "constraints": []
}
EOF

hzn register --policy node-policy.json
```

- note: `properties` and `constraints` in the node policy document allow you to describe the edge node or set of edge nodes.

On `mgmt.vm.com`, you can check for registered nodes,

```
export HZN_ORG_ID=myorg
export HZN_EXCHANGE_USER_AUTH=admin:HCuP...

hzn exchange node list
[
  "myorg/node1",
  "myorg/edgevm"
]
```

- note: `myorg/node1` is an artifact of the quickstart script.

Deploying a container to an edge node - 1

To begin with, we will deploy the docker `hello-world` container to the edge node via open horizon. This container simply starts, emits a message to `stdout`, and exits successfully. Log onto the management node,

```
ssh user@mgmt.vm.com
mkdir ~/hello-world
cd ~/hello-world
```

Create a `service.json` file to describe the service you wish to deploy and publish it to the open horizon exchange. This will download the image locally (from the docker registry) and sign the image.

```
cat << EOF > service.json
{
  "org": "myorg",
  "label": "docker hello-world container",
  "url": "docker-hello-world",
  "version": "1.0.0",
  "arch": "amd64",
  "public": true,
  "sharable": "singleton",
  "requiredServices": [],
  "userInput": [],
  "deployment": {
    "services": {
      "docker-hello-world": {
        "image": "hello-world:latest"
      }
    }
  }
}

EOF

hzn  exchange  service  publish  -f service.json  --pull-image
```

The output of the `publish` should resemble,

```
Signing service...
Pulling hello-world:latest...
latest: Pulling from library/hello-world
719385e32844: Pulling fs layer
719385e32844: Verifying Checksum
719385e32844: Download complete
719385e32844: Pull complete
Digest: sha256:88ec0acaa3ec199d3b7eaf73588f4518c25f9d34f58ce9a0df68429c5af48e8d
Status: Downloaded newer image for hello-world:latest
Using 'hello-world@sha256:88ec0acaa3ec199d3b7eaf73588f4518c25f9d34f58ce9a0df68429c5af48e8d' in 'd
eployment' field instead of 'hello-world:latest'
Creating docker-hello-world_1.0.0_amd64 in the Exchange...
Storing default.public.key with the service in the Exchange...
```

Create a `deployment.json` policy file to target which edge nodes recieve the container,

```
cd ~/hello-world

cat << EOF > deployment.json
{
```

```
  "label": "",
  "description": "",
  "service": {
    "name": "docker-hello-world",
    "org": "myorg",
    "arch": "amd64",
    "serviceVersions": [
      {
        "version": "1.0.0",
        "priority": {
          "priority_value": 3,
          "retries": 2,
          "retry_durations": 60
        }
      }
    ]
  },
  "properties": [],
  "constraints": [
    "edge.id == \"edgevm\""
  ],
  "userInput": []
}
EOF

hzn exchange deployment addpolicy -f deployment.json hello-world
hzn exchange deployment listpolicy
```

Log on the the edge node and watch for agreement,

```
ssh user@edge.vm.com
watch hzn agreement list
```

Check if the container is running on the edge,

```
ssh user@edge.vm.com
watch docker ps
```

You should see the container start, run for a few moments, and then exit. Once open-horizon identifies that the container is no longer running (and thus the deployment policy is not satisfied), the container is restarted.

To remove the deployment from the exchange,

```
hzn exchange deployment removepolicy hello-world
```

Deploying a container to an edge node

- note: this example requires access to a remote docker registory

Other than showing how to deploy a container and that open horizon restarts a container that terminates, the `hello-world` container is of little use. This example will deploy a mqtt client on the edge node which transmits data to a mqtt broker. If you do not have a mqtt broker, a container with one can be started on the `mgmt.vm.com` machine, as follows,

```
ssh user@mgmt.vm.com
cat << EOF > ./mosquitto.conf
listener 1883 0.0.0.0
allow_anonymous true
EOF
```

```
docker run --rm --name mosquitto -v /home/user/mosquitto.conf:/mosquitto/config/mosquitto.con
f -p 1883:1883 eclipse-mosquitto:latest
```

On `mgmt.vm.com`, you can subscribe to this broker with `mosquitto_sub -v -h mgmt.vm.com -t '#'` (to install client if not present: `sudo apt install mosquitto-clients`)

Create an mqtt client service and deploy it to the edge node.

```
ssh user@mgmt.vm.com
mkdir ~/mqtt-client
cd ~/mqtt-client

# create Dockerfile
cat << EOF > Dockerfile
FROM debian:stable-slim

RUN apt-get update  && apt-get upgrade -y  && apt-get install -y mosquitto-clients

COPY run.sh /run.sh

ENTRYPOINT ["/run.sh"]
EOF

# create shell script which runs in container
cat << EOF > run.sh
#!/bin/bash

export MHOST=${MHOST:-127.0.0.1}
export TOPIC=${TOPIC:-fallback}
export MSG=${MSG:-message}


while [ 1 ]
do
  STAMP=$(date)
  echo "001 ${MHOST} ${TOPIC} ${MSG}"
  mosquitto_pub -h ${MHOST}  -t ${TOPIC}  -m "001 ${STAMP} ${MSG}"

  sleep 5
done
EOF

chmod 755 run.sh

docker build -t mqttpub:001 .
docker tag mqttpub:001 remote-registry:port/mqttpub:001
docker push remote-registry:port/mqttpub:001
```

Create a service description for open horizon,

```
cd ~/mqtt-client

# replace REMOTE-REGISTRY:PORT
cat << EOF > service.json
{
  "org": "myorg",
  "label": "publish mqtt message to broker",
  "url": "publish-mqtt",
  "version": "1.0.0",
  "arch": "amd64",
  "public": true,
```

```
  "sharable": "singleton",
  "requiredServices": [],
  "userInput": [],
  "deployment": {
    "services": {
      "publish-mqtt": {
        "image": "REMOTE-REGISTRY:PORT/mqttpub:001"
      }
    }
  }
}
EOF

hzn  exchange  service  publish  -f service.json  --pull-image
```

Create a deployment policy for this service,

```
cd ~/mqtt-client

cat << EOF > deployment.json
{
  "label": "",
  "description": "",
  "service": {
    "name": "publish-mqtt",
    "org": "myorg",
    "arch": "amd64",
    "serviceVersions": [
      {
        "version": "1.0.0",
        "priority": {
          "priority_value": 3,
          "retries": 2,
          "retry_durations": 60
        }
      }
    ]
  },
  "properties": [],
  "constraints": [
    "edge.id == \"edgevm\""
  ],
  "userInput": [
    {
      "serviceOrgid": "myorg",
      "serviceUrl": "publish-mqtt",
      "serviceVersionRange": "1.0.0",
      "inputs": [
        {
          "name": "TOPIC",
          "value": "edgevm"
        },
        {
          "name": "MSG",
          "value": "new message"
        },
        {
          "name": "MHOST",
          "value": "mgmt.vm.com"
        }
      ]
    }
  ]
}
EOF
```

```
hzn exchange deployment addpolicy  -f deployment.json  mpub
hzn exchange deployment listpolicy
```

Log on to the edge node and watch for agreement,

```
ssh user@edge.vm.com
watch hzn agreement list
```

Check if the container is running on the edge,

```
ssh user@edge.vm.com
watch docker ps
```

Check the mqtt broker for delivered messages

```
mosquitto_sub  -h mgmt.vm.com  -t '#'
```

To cancel the deployment,

```
hzn exchange  deployment removepolicy  mpub
```

Deploying multiple dependent containers to an edge node

As a simple example we will deploy three containers, `source`, `node`, and `sink` to an edge node. `source` sends a message to `node`, which appends data to the message before forwarding it to `sink`, which tranmits the message it receives to a remote mqtt broker. This will require the creation of the following files (in directory `~/pipeline`),

```
source.df    # Dockerfile
source.json  # service file
source.sh    # simple application to emit data

node.df      # Dockerfile
node.json    # service file
node.sh      # simple application to update and relay data

sink.df      # DockerFile
sink.json    # service file
sink.sh      # simple application to recieve and relay data to a mqtt broker

deployment.json  # deployment policy file
```

- note: if the services require network access to one another, they cannot simple be deployed as open horizon creates an unique isolated docker network for each service. Therefore in order to allow communication between services, dependencies (`requiredServices`) must be defined in each service file

The contents of the source related files are,

```
==> source.df <==
FROM debian:stable-slim

RUN apt-get update   && apt-get upgrade  -y   && apt-get install  -y  netcat-openbsd

COPY source.sh /source.sh

ENTRYPOINT ["/source.sh"]
```

```
==> source.json <==
{
  "org": "myorg",
  "label": "pipeline: source",
  "url": "source",
  "version": "0.0.1",
  "arch": "amd64",
  "public": true,
  "sharable": "singleton",
  "requiredServices": [],
  "userInput": [],
  "deployment": {
    "services": {
      "pipeline-source": {
        "image": "REMOTE-REGISTRY:PORT/source:latest"
      }
    }
  }
}

==> source.sh <==
#!/bin/bash

export ADDRESS=${ADDRESS:-127.0.0.1}

while [ 1 ]
do
    msg=$(date)
    echo $msg
    echo $msg | netcat -N ${ADDRESS} 1234
    sleep 5
done
```

The contents of the node related files are,

```
==> node.df <==
FROM debian:stable-slim

RUN apt-get update   && apt-get upgrade  -y   && apt-get install  -y  netcat-openbsd

COPY node.sh /node.sh

ENTRYPOINT ["/node.sh"]

==> node.json <==
{
  "org": "myorg",
  "label": "pipeline: node",
  "url": "node",
  "version": "0.0.1",
  "arch": "amd64",
  "public": true,
  "sharable": "singleton",
  "requiredServices": [
    {
      "org": "myorg",
      "url": "source",
      "version": "0.0.1",
      "arch": "amd64"
    }
  ],
  "userInput": [],
```

```
  "deployment": {
    "services": {
      "pipeline-node": {
        "image": "REMOTE-REGISTRY:PORT/node:latest"
      }
    }
  }
}

==> node.sh <==
#!/bin/bash


export ADDRESS=${ADDRESS:-127.0.0.1}


while [ 1 ]
do
    msg=$(nc  -l  -p 1234)
    echo $msg

    echo "$msg + node" | netcat  -N ${ADDRESS}  1235
done
```

The contents of the sink related files are,

```
==> sink.df <==
FROM debian:stable-slim

RUN apt-get update   && apt-get upgrade  -y   && apt-get install  -y  mosquitto-clients  netcat-o
penbsd

COPY sink.sh /sink.sh

ENTRYPOINT ["/sink.sh"]

==> sink.json <==
{
  "org": "myorg",
  "label": "pipeline: sink",
  "url": "sink",
  "version": "0.0.1",
  "arch": "amd64",
  "public": true,
  "sharable": "singleton",
  "requiredServices": [
    {
      "org": "myorg",
      "url": "node",
      "version": "0.0.1",
      "arch": "amd64"
    }
  ],
  "userInput": [],
  "deployment": {
    "services": {
      "pipeline-sink": {
        "image": "REMOTE-REGISTRY:PORT/sink:latest"
      }
    }
  }
}
```

```
==> sink.sh <==
#!/bin/bash


export MHOST=${MHOST:-127.0.0.1}
export TOPIC=${TOPIC:-fallback}


while [ 1 ]
do
    msg=$(nc  -l  -p 1235)
    echo $msg

    mosquitto_pub  -h ${MHOST}   -t ${TOPIC}   -m "001 ${msg}"
done
```

And finally the contents of `deployment.json`,

```
{
  "label": "",
  "description": "",
  "service": {
    "name": "sink",
    "org": "myorg",
    "arch": "amd64",
    "serviceVersions": [
      {
        "version": "0.0.1",
        "priority": {
          "priority_value": 3,
          "retries": 2,
          "retry_durations": 60
        }
      }
    ]
  },
  "properties": [],
  "constraints": [
    "edge.id == \"edgevm\""
  ],
  "userInput": [
    {
      "serviceOrgid": "myorg",
      "serviceUrl": "source",
      "serviceVersionRange": "0.0.1",
      "inputs": [
        {
          "name": "ADDRESS",
          "value": "pipeline-node"
        }
      ]
    },
    {
      "serviceOrgid": "myorg",
      "serviceUrl": "node",
      "serviceVersionRange": "0.0.1",
      "inputs": [
        {
          "name": "ADDRESS",
          "value": "pipeline-sink"
        }
      ]
    },
    {
```

```
      "serviceOrgid": "myorg",
      "serviceUrl": "sink",
      "serviceVersionRange": "0.0.1",
      "inputs": [
        {
          "name": "TOPIC",
          "value": "edgevm"
        },
        {
          "name": "MHOST",
          "value": "mgmt.vm.com"
        }
      ]
    }
  ]
}
```

Deployment is as before,

- build, tag, and push containers,
- publish services to exchange
- add deployment policy

```
ssh user@mgmt.vm.com

export HZN_ORG_ID=myorg
export HZN_EXCHANGE_USER_AUTH=admin:HCuP...

cd ~/pipeline

docker build  -f source.df  -t source:latest  .
docker build  -f node.df  -t node:latest  .
docker build  -f sink.df  -t sink:latest  .

docker tag  source:latest  REMOTE-REGISTRY:PORT/source:latest
docker push REMOTE-REGISTRY:PORT/source:latest

docker tag  node:latest  REMOTE-REGISTRY:PORT/node:latest
docker push REMOTE-REGISTRY:PORT/node:latest

docker tag  sink:latest  REMOTE-REGISTRY:PORT/sink:latest
docker push REMOTE-REGISTRY:PORT/sink:latest

hzn  exchange  service  publish  -f source.json  --pull-image
hzn  exchange  service  publish  -f node.json  --pull-image
hzn  exchange  service  publish  -f sink.json  --pull-image

hzn exchange deployment addpolicy  -f deployment.json  pipeline
hzn exchange deployment listpolicy
```

Log on to the edge node and watch for agreement,

```
ssh user@edge.vm.com
watch hzn agreement list
```

Check if the container is running on the edge,

```
ssh user@edge.vm.com
watch docker ps

# also check docker networks created
docker network ls
```

Check the mqtt broker for delivered messages

```
mosquitto_sub  -h mgmt.vm.com  -t '#'
```

To cancel the deployment,

```
hzn exchange  deployment removepolicy  pipeline
```

Updating a deployed container

Returning to the mqtt example, if it is not still deployed, redeploy it.

```
ssh user@mgmt.vm.com
cd ~/mqtt-client

hzn  exchange  deployment  addpolicy  -f deployment.json  mqtt
```

To update this container, edit `run.sh` changing `001` to `002`,

```
#!/bin/bash

export MHOST=${MHOST:-127.0.0.1}
export TOPIC=${TOPIC:-fallback}
export MSG=${MSG:-message}


while [ 1 ]
do
  STAMP=$(date)
  echo "002  ${MHOST}  ${TOPIC}  ${MSG}"
  mosquitto_pub  -h ${MHOST}   -t ${TOPIC}   -m "002  ${STAMP}  ${MSG}"

  sleep 5
done
```

Build, tag, and push container,

```
docker build  -t mqttpub:002  .
docker tag  mqttpub:002  REMOTE-REGISTRY:PORT/mqttpub:002
docker push REMOTE-REGISTRY:PORT/mqttpub:002
```

Update version and docker image url in `service.json`,

```
{
  "org": "myorg",
  "label": "publish mqtt message to broker",
  "url": "publish-mqtt",
  "version": "2.0.0",
  "arch": "amd64",
  "public": true,
  "sharable": "singleton",
  "requiredServices": [],
  "userInput": [],
  "deployment": {
    "services": {
      "publish-mqtt": {
        "image": "REMOTE-REGISTRY:PORT/mqttpub:002"
      }
    }
  }
}
```

Publish service,

```
hzn exchange service publish -f service.json --pull-image
```

Update `deployment.json`, appending to `serviceVersions` and `userInput`. The `serviceVersions.priority.priority_value` determines which container is started (lower value == more important). Failure to start the container will result in fallback to the next `priority_value`.

```
{
  "label": "",
  "description": "",
  "service": {
    "name": "publish-mqtt",
    "org": "myorg",
    "arch": "amd64",
    "serviceVersions": [
      {
        "version": "2.0.0",
        "priority": {
          "priority_value": 1,
          "retries": 2,
          "retry_durations": 60
        }
      },
      {
        "version": "1.0.0",
        "priority": {
          "priority_value": 3,
          "retries": 2,
          "retry_durations": 60
        }
      }
    ]
  },
  "properties": [],
  "constraints": [
    "edge.id == \"edgevm\""
  ],
  "userInput": [
    {
      "serviceOrgid": "myorg",
      "serviceUrl": "publish-mqtt",
      "serviceVersionRange": "2.0.0",
      "inputs": [
        {
          "name": "TOPIC",
          "value": "edgeVM"
        },
        {
          "name": "MSG",
          "value": "new message"
        },
        {
          "name": "MHOST",
          "value": "mgmt.vm.com"
        }
      ]
    },
    {
      "serviceOrgid": "myorg",
      "serviceUrl": "publish-mqtt",
      "serviceVersionRange": "1.0.0",
      "inputs": [
        {
```

```
          "name": "TOPIC",
          "value": "edgevm"
        },
        {
          "name": "MSG",
          "value": "new message"
        },
        {
          "name": "MHOST",
          "value": "mgmt.vm.com"
        }
      ]
    }
  ]
}
```

Deploy,

```
hzn exchange deployment addpolicy  -f deployment.json  mqtt
hzn exchange deployment listpolicy
```

Log on to the edge node and watch for agreement,

```
ssh user@edge.vm.com
watch hzn agreement list
```

Check if the container is running on the edge,

```
ssh user@edge.vm.com
watch docker ps
```

Check the mqtt broker for updated delivered messages (`edgevm 001 TIMESTAMP` -> `edgeVM 002 TIMESTAMP`)

```
mosquitto_sub  -h mgmt.vm.com  -t '#'
```

To cancel the deployment,

```
hzn exchange  deployment removepolicy  mqtt
```