Received: 12 August 2020 | Revised: 8 December 2020 | Accepted: 16 March 2021

DOI: 10.1002/spe.2973

RESEARCH ARTICLE

WILEY

Design and evaluation of a scalable Internet of Things backend for smart ports

Vincent Bracke | **Merlijn Sebrechts** | **Bart Moons** | **Jeroen Hoebeke** | **Filip De Turck** | **Bruno Volckaert**

IDLAB, Department of Information Technology, Ghent University, Ghent, Belgium

Correspondence
Vincent Bracke, IDLAB, Department of Information Technology, Ghent University, Ghent, Belgium.
Email: vincent.bracke@ugent.be

Abstract

Internet of Things (IoT) technologies, when adequately integrated, cater for logistics optimisation and operations' environmental impact monitoring, both key aspects for today's EU ports management. This article presents Obelisk, a scalable and multi-tenant cloud-based IoT integration platform used in the EU H2020 PortForward project. The landscape of IoT protocols being particularly fragmented, the first role of Obelisk is to provide uniform access to data originating from a myriad of devices and protocols. Interoperability is achieved through adapters that provide flexibility and evolvability in protocol and format mapping. Additionally, due to ports operating in a hub model with various interacting actors, a second role of Obelisk is to secure access to data. This is achieved through encryption and isolation for data transport and processing, respectively, while user access control is ensured through authentication and authorisation standards. Finally, as ports *IoTisation* will further evolve, a third need for Obelisk is to scale with the data volumes it must ingest and process. Platform scalability is achieved by means of a reactive micro-services based design. Those three essential characteristics are detailed in this article with a specific focus on how to achieve IoT data platform scalability. By means of an air quality monitoring use-case deployed in the city of Antwerp, the scalability of the platform is evaluated. The evaluation shows that the proposed reactive micro-service based design allows for horizontal scaling of the platform as well as for logarithmic time complexity of its service time.

KEYWORDS

cloud-based integration, IoT, IoT data hub, IoT platform, smart port

1 | INTRODUCTION

Ports are economic lungs for the region they are located in. Further, they play a central role in today's global economy with continuously growing trade volumes and strict just-in-time delivery imperatives requiring uninterrupted operationality.

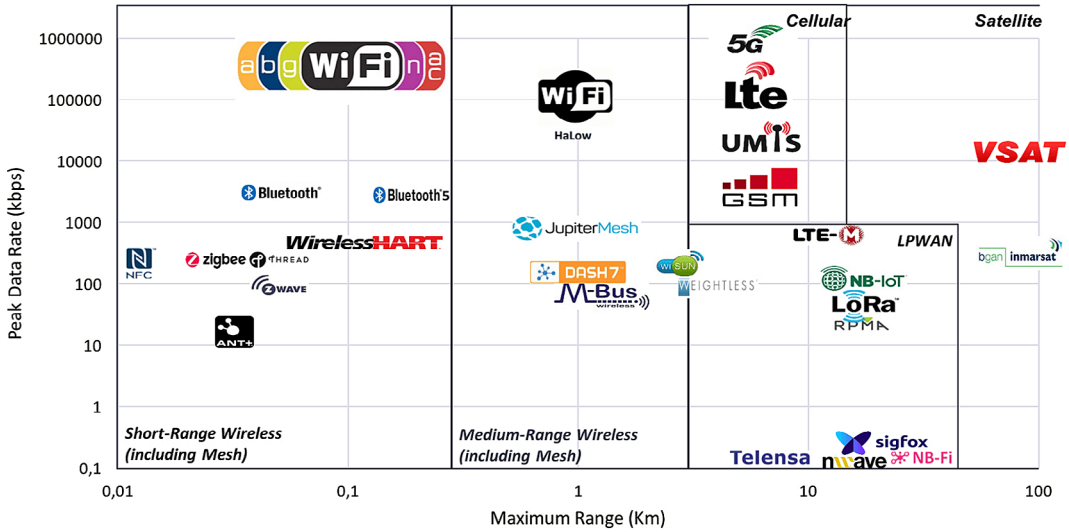

FIGURE 1 Comparison of main IoT wireless technologies;[1] a fragmented and complex landscape [Color figure can be viewed at wileyonlinelibrary.com]

Operational excellence and environmental footprint are not only buzz words, they also and mostly are survival conditions in today's economic competition and the real differentiating factors. Ports are no exception to this rule and the hub model they operate in makes it even more complex to stay in control of the end-to-end management of their open value chain.

Modern information and communication technologies such as the Internet of Things (IoT), by interfacing the 'physical world' with the 'virtual world', may help ports facing the challenges they are confronted with by improving efficiency and providing accrued awareness in their operations. Indeed, port activities encompass a wild variety of use-cases that can benefit from the adoption of IoT. Amongst those use-cases, and not exhaustively: asset tracking, continuous metering of environmental conditions and energy consumption and lastly remote assistance in operations.

Ports of the future will be 'IoT-enabled', giving rise to the so-called 'Smart Port'. Though, the uninterrupted flow of data originated from the multitude of deployed 'things' needs to be integrated within the port Information System. To achieve this an integration layer that offers interoperability, multi-tenancy and scalability capabilities is necessary.

Interoperability is needed to offer uniform access to data amongst the wide variety of IoT wireless protocols. Figure 1 positions main protocols according to their theoretical maximum data rate and range.[*] As illustrated there is no silver bullet but rather a fragmented landscape. This comes from the usually constrained nature of the transmitting embedded devices with optimised form factor and battery lifetime trade-off. Wireless technology selection is thus use-case dependent and ultimately results from a multi-dimensional compromise that covers various aspects (e.g., scalability, range, coverage, deployment, battery lifetime, Quality of Service, up-link vs down-link, payload length, latency, cost efficiency for licensed protocols and protocols using frequencies under licensed spectrum).

Scalability is needed to ingest variable volumes of data as well as to support future growth in terms of IoT devices deployment and volume of activities. In order to shape the order of magnitude of today's EU mid-sized ports annual volume of activity, Table 1 summarises main figures extracted from public reports of two such ports (Autoritá di Sistema Portuale del Mar Tirreno Settentrionale (AdSP MTS),[2] Italy and Autoridad Portuaria de Baleares (APB),[3] Spain). Those reports noticeably indicate that premises under management cover millions of square metres, that there are hundreds of thousands of containers, commercial vehicles for the sale and Ro-Ro[†] cargo and millions of tons of dry and liquid bulk that yearly transit by the port.

[*] Protocols are positioned in the chart according to their maximum theoretical range and data rate. However both cannot be achieved at the same time: the higher the data rate, the lower the range (and vice-versa).

[†] 'Roll-on/Roll-off' (Ro-Ro) designates wheeled cargo, such as trucks, trailers and railroad cars that are driven on and off the ship on their own wheels. This is in contrast to 'Lift-on/Lift-off' (LoLo) cargo that requires a crane to be loaded to and unloaded from the ship.

TABLE 1 Example of ports activities volume

2018	AdSP MTS	APB
Total traffic (tons)	**44,081,063**	**16,205,000**
– Liquid bulk	9, 562, 275	1, 696, 000
– Dry bulk	2, 048, 291	1, 415, 000
– Containerised cargo	8, 538, 918	660, 000
– Ro-Ro	22, 171, 749	12, 328, 000
– Other general cargo	1, 759, 830	106, 000
Other figures (numbers)		
Ro-Ro	728, 655	603, 725
Commercial vehicles	666, 651	917, 112
Containers (TEUs)	734, 085	121, 437
Passengers	9, 774, 533	8, 942, 448
Vessels	34, 912	48, 366
Land area (sqm)	2, 500, 000	2, 043, 709

Multi-tenancy is needed to enforce data isolation required by the multitude of actors operating within ports premises. Port authorities[‡]ensure the landlord, regulator, operator and community manager functions for the port(s) they are accountable for. However, the way they realise those functions highly depends on their ownership (public vs private),[§] their objectives[¶] and their juridical structure and management style.[#]Nevertheless, despite their differences and specificities, a general trend is observed: most EU's port authorities remain under public ownership while moving towards more independent private-like management. Table 2 illustrates the impact of this trend with most of ports' services being executed by private operators who are generally granted the use of (or simply the access to) port land through lease agreements or public domain concessions. Thus port authorities delegate the execution of the operator function to private operators, leading to complex mixtures of roles and responsibilities that most often are port specific. This requires fine-grained access rights to data stored in and processed by the IoT integration platform.

This article presents Obelisk, a cloud-based IoT integration platform that offers the interoperability, multi-tenancy and scalability capabilities required by the nature and volume of ports activities.[‖] It provides an overview of the design choices made in an IoT integration project within the context of IoT-enabled ports. The provided solution is based on open-source software packages and frameworks and can serve as a 'how-to' model for the design and implementation of future IoT integration platforms. The article provides an evaluation of the scalability of the proposed IoT data hub and also deals with device connectivity as well as how to achieve end-to-end data isolation. To the best of our knowledge, no prior work describes this with on the one hand a comparable level of design details and on the other hand a holistic integrated view from device over IoT data hub to application consuming data from a fleet of devices.

The remainder of this article is organised as follows. Section 2 positions this article in its context, summarizing the current state of the art in the domain, after which Section 3 introduces the operational challenges today's ports are facing. Section 4 presents the architecture and implementation of Obelisk and Section 5 proposes, by means of an air quality monitoring use-case, an in-depth analysis of the scalability capability of the platform. Section 6 discusses tracks for further research while Section 7 provides concluding remarks.

[‡]The port authority is 'the entity which, whether or not in conjunction with other activities, has as its objective under national law or regulation the administration and management of the port infrastructures, and the co-ordination and control of the activities of the different operators present in the port'.[4]

[§]In 2016, 87% were publicly owned while only 6% were fully privately owned (mainly in the UK). The remaining 7% having a mixed private-public ownership.[5]

[¶]In 2016, 71% of EU's port authorities were mission-driven entities where profit/cost recovery is a must but not the only conside, 15% were non-economic public bodies run with general interest objectives and finally the remaining 14% were profit-maximising businesses.[5]

[#]In 2016, 51% were structured as independent commercial entities and 44% as independent public bodies. The last 5% being under another structure.[5]

[‖]While suiting the requirements of 'Smart Ports' use-case, Obelisk platform is not exclusively dedicated to it and is used as well in other IoT applications contexts (e.g., smart city, smart home, industry 4.0).

TABLE 2 EU's ports services distribution amongst actors[5]

Service	Port Authority	Government	Private	Other
PROVISION OF PORT SERVICES TO SHIPS				
Pilotage outside the port area	9%	35%	**42%**	14%
Pilotage inside the port area	19%	22%	**47%**	11%
Towage outside the port area	3%	3%	**88%**	6%
Towage inside the port area	12%	1%	**83%**	4%
Mooring	27%	2%	**65%**	5%
Waste reception facilities	40%	0%	**58%**	2%
Onshore power supply	**62%**	0%	34%	4%
Bunkering	4%	0%	**94%**	2%
PROVISION OF SERVICES TO CARGO				
Cargo handling on board ship	16%	0%	**80%**	4%
Cargo handling ship-shore	23%	0%	**74%**	3%
Cargo handling shore-inland transport	14%	0%	**84%**	2%
Logistics services	11%	0%	**88%**	1%
Warehousing services	18%	0%	**81%**	1%
Road haulage	3%	1%	**93%**	2%
Rail operation	8%	10%	**74%**	8%
Inland barging	0%	0%	**100%**	0%

2 | RELATED WORK

Literature on ports logistics automation abounds denoting its complexity and the vivid economical and societal interests it generates. It ranges from sensing and communication technologies reviews applied to port use-cases[6] to Decision Support Systems (DSS)[7] and Virtual Twins (VT).[8] Botti et al.[9] even propose a holistic reconsideration of the port value chain by means of Information and Communication Technologies (ICT).

Heilig and Voss[10] demonstrate the role of integrated information systems for enhancing the performance of ports as well as to comply with regulatory requirements. For the authors, the evolution and availability of mobile devices provide many opportunities in the logistics sectors and specifically in the port industry. Yet, they report that the adoption of mobile devices in port communities is still in its infancy, which also applies to research in this area. Still according to the authors, the importance of integrated information systems will consequently continue to grow and the main focus should be put on the integration of data sources by means of highly scalable and distributed computing cloud-platforms.

More recently, comparable statement has been expressed by Yau et al.[11] who observe that in spite of the urgent need to deploy and support various smart applications, there is relatively little work on the design of IoT platforms for smart applications in smart port. Still according to the authors, further investigations could be pursued to design IoT platforms supporting smart applications.

For logistics auspicious automation, successful IoT integration has thus become a no-brainer. This explains the recent craze for IoT and more specifically for IoT cloud platforms. Internet Giants all propose such platforms through Platform as a Service (PaaS) commercial offerings. Despite free tier limitations, Pierleoni et al.[12] evaluate the scalability of Amazon Web Services (AWS) IoT Core, Google Cloud IoT Core and Microsoft (MS) Azure IoT Hub under three scenarios[**] with different loads in terms of message sent per second. The performance indicator under consideration in those different scenarios is the service-time, that is, the time consumed by the platform to perform the message brokering activity.[††]

[**] Point-to-point (one producer, one consumer), fan-in (multiple producers, one consumer) and fan-out (one producer multiple consumers).
[††] This indicator is comparable to the concept of 'Internal Streaming Time' (IST) introduced in Section 5.3 of this article. Importantly though, the results obtained by Pierleoni et al.[12] can hardly be compared to the results presented in Sections 5.3.1 and 5.3.2 of this article as test conditions (transport protocol, data-set, infrastructure size) substantially diverge.

However, vendors of IoT cloud platforms come with complex costs structures,[13] legal and jurisdictional challenges as well as technical lock-in.[14] Opara-Martins et al.[14] assert that cloud software vendors lock in customers in several ways: (1) by designing a system incompatible with software developed by other vendors; (2) by using proprietary standards or closed architectures that lack interoperability with other applications; and (3) by licensing the software under exclusive conditions. The authors further specify that vendor lock-in problem in cloud computing is characterised by expensive and time-consuming migration of application and data to alternative providers which deters organisations adopting cloud technology.

Regarding open-source alternatives, a comparable study has been conducted by Ismail et al.[15] for two such IoT platforms, namely ThingsBoard and SiteWhere, on a setup composed of one local server. Two indicators are measured: throughput in terms of messages per second and average response time for both the Message Queuing Telemetry Transport (MQTT) protocol[‡‡]and the Hypertext Transfer Protocol (HTTP). The authors analyze both scalability and stability but with other interpretation than provided in this article. Their interpretation of stability mainly concerns CPU usage while their interpretation of scalability only concerns the response time and throughput in terms of messages per seconds. Though there is no explicit mention of horizontal scaling of individual components (micro-services) with the associated impact on overall platform performance, which is discussed in this article.

Nevertheless, cloud-based IoT platforms performances benchmarking remains complex as it requires to be conducted on comparable hardware setup with similar requests characteristics (i.e., protocols, payload size, request rates). While efforts to evaluate the performance of specific IoT platforms has begun, a fine-grained understanding of their performance is severely lacking in the state-of-the-art research and a benchmark that is reusable across IoT platforms is still lacking.[16] Further, literature recurrently identifies interoperability, multi-tenancy and scalability as open challenges in cloud-based IoT integration.[17-21]

More specifically, IoT integration in a Smart Port context is discussed by Suárez et al.[22] and Fernández et al.[23] that propose an open-source Smart Port implementation based on Fiware Orion Context Broker and Generic Enablers (GEs). However, the proposed solution doesn't really address the hereinabove mentioned interoperability and multi-tenancy dimensions. In Reference 24, Fiware Orion Context Broker (and GEs) is also used, in an Industry 4.0 context; while multi-tenancy and interoperability are discussed the authors report some scalability limitations explained by their non cloud-based setup. Araujo et al.[16] report from their extensive analysis of Fiware scaling capabilities that their results clearly identify bottlenecks as well as limitations in the current Fiware architecture.

To the best of our knowledge, no prior work provides comparable accounts that can serve as 'how-to' model for future work in the same field. Indeed, this article provides an in-depth description of Obelisk's architecture[§§]amended by thorough explanations on design, implementation and technology choices. Further, this description is complemented by empirical validation where both platform stability and scalability are demonstrated. Finally, while Obelisk is currently used in different IoT contexts (smart city,[26-28] industry 4.0,[29] home automation, etc.), it has not yet been extensively described in academic publications.

3 | TOWARDS SMART PORTS: A BRIEF OVERVIEW OF PORTS OPERATIONAL CHALLENGES

How to increase performance and at the same time reduce costs while guaranteeing the highest standards in safety, health and environment? To answer this key question to today's port authorities, three main axes are emerging to shape what shall be called 'The Port of the Future' that will have to be:

- *Smart*, in order to achieve better monitoring and control of the freight flows, by integrating novel IoT technologies;
- *Interconnected*, in order to improve exchange of information between port and port community as well as to provide insightful collaboration with partners (terminal operators, cargo carriers, cruise ships companies, maintenance technicians, etc.) throughout the value chain;
- *Green*, through the adoption of green technologies in order to reduce the environmental impact of port operations and save resources.

[‡‡]Configured with the lowest Quality of Service, that is, 'At Most Once' delivery.

[§§]Noticeably, the architecture defined in this article for the PortForward project, as illustrated by Figure 2, matches the reference architecture provided by Guth et al.[25] with Obelisk as 'IoT Integration Middleware (IoTIM)' layer.

In order to successfully cope with this demanding reality, IoT technologies, by interfacing the 'physical world' with the 'virtual world', allow to improve efficiency and accrue awareness in ports operations. Indeed port activities encompass a wild variety of use-cases that can benefit from the adoption of IoT. As explained by Molavi et al.,[30] with the global economy demanding maritime transportation, ports have faced increasing pressure to optimise their performance in terms of economic, environmental, energy and functional challenges that impact their sustainability. The authors regroup key elements and associated issues within five distinct categories, namely: operations, environment, energy, safety and security. According to the authors, in response to the existing problems, ports are adopting technology-based solutions. Implementation of such solutions to mitigate recent problems is known to be switching to smart ports. Typical use-cases are:

- Operations: drone-based inventory management of containers, automated customs clearance/gating by container/vehicle recognition, remote assistance and/or guidance of operators in complex tasks with, for example, Augmented Reality (AR) devices, and so on.
- Environment: based on weather forecast and observed air and/or water quality, operations' scheduling can potentially be adapted (as much as it possibly can be) as to respect nuisances thresholds agreed with or defined by the surrounding municipality, and so on.
- Energy: optimised scheduling of operations positively impacts fuel consumption and associated emissions (cranes (un)loading ships, movement of trailers and trucks picking up containers, etc.), smart-lighting with significant reduction of electricity consumption, and so on.
- Safety: continuous monitoring of infrastructure alteration and assets usage for on-time preventive maintenance, AR glasses to guide vessel captains when manoeuvring, berthing (mooring a ship in its allotted place) or even engaging waterways with complex bathymetry (underwater topography), and so on.
- Security: automatic number/license plate recognition (ANPR/ALPR) systems at port road gates to instantly detect and record vehicle entrance and exit, premises inspection with real-time object detection by means of unmanned aerial vehicles (UAVs)[31] to safeguard onsite goods, assets and people, and so on.

Noticeably though, the hereinabove presented use-cases should not be perceived like as many isolated opportunities, but rather as a holistic approach which, when successfully implemented and integrated, delivers insightful information for a more efficient management of the port through decision support systems, 360° dashboards, and so on. Alternatively, instead of being grouped by category, the use-cases could also have been presented by end-to-end business-case, integrating various dimensions, for example, reduction of time spent in port (asset tracking, automated customs clearance and gating, etc.), energy consumption reduction (metering, smart-lighting, scheduling optimisations, etc.).

Yau et al.[11] identify eight distinct categories[¶¶] of smart ports open issues that would benefit from further research, all of them relying on IoT technologies as technical enabler. PortForward project aims at implementing novel solutions for various of those categories with Obelisk as IoT integration platform upon which future developments can leverage.

4 | OBELISK, THE IOT INTEGRATION PLATFORM

Obelisk has been designed as a scalable IoT data hub. The design and implementation details are presented in this section.

4.1 | Proposed functional architecture

Figure 2, hereafter detailed, illustrates a high-level representation of the functional architecture of the PortForward project with, in blue, the components discussed in following sections. The typical data-flow is initiated at 'data producers' side (on the very left of the diagram), that in an IoT context typically consists of connected devices. Nevertheless, more

[¶¶] Exploring new applications, Collaboration platform for different ports, Enhanced Terminal Operating System, Enhancing ports by collaborating with smart ships, Completely digitised autonomous smart ports, Big data analytics, Using Blockchain for logistics and finally Cyber security.

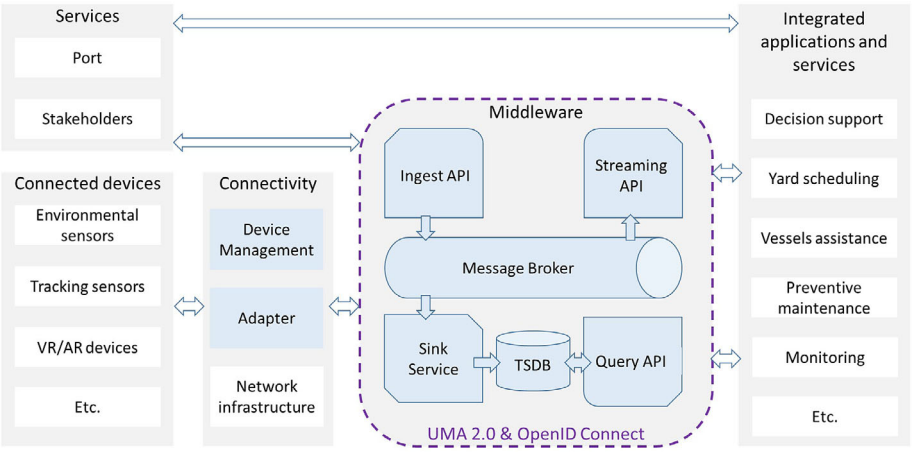

FIGURE 2 Proposed functional architecture for IoT-enabled ports [Color figure can be viewed at wileyonlinelibrary.com]

traditional Information Systems may complement this category as well (e.g., a weather forecast Application Programming Interface (API), a cruise line company API, the Information System of the municipality).

In essence, if a system, whatever its nature, provides valuable information for the efficient management of port activities, it should be integrated into the solution in order to feed the final service or application (also called the 'data consumer', on the very right of the diagram) with the data it needs.

In traditional internet applications, this inter-system communication is now quite standardised through the use of HTTP(s) REpresentational State Transfer (REST) APIs (represented by the arrow on the top of the diagram) and shall therefore not be further discussed in this article. However, when it comes to connected devices and more specifically to IoT, standardisation is more the exception than the rule. Wireless connectivity protocols are indeed plenty, sometimes vendor specific. This comes from the necessity to best accommodate the use-cases (and their associated constraints) they are intended for (e.g., a sensor in an unmanned vehicle will typically send a lot of messages per second over a short distance (range in centimetres) and battery capacity won't be the first constraint, while a high-sea buoy used to collect wind statistics over time will have to send a few messages per hour or even day but at a very high distance (range in kilometres) and with high constraints on battery lifetime).

To provide the expected level of interoperability, the connectivity layer illustrated in Figure 2 makes use of the Adapter architectural pattern by connecting protocol specific adapters on the network infrastructure.

Obelisk, the middleware layer (central component in Figure 2) exposes an API for data publishing (the *Ingest* API). It also provides two data access mechanisms to the consuming applications, being (i) a *Query*API providing the retrieval of historical data (with filtering and pagination mechanisms) and (ii) a *Streaming* API for data pipelining, that is, consuming the data as they enter the platform. Both mechanisms are complementary to allow historical data fetching and (near) real-time eventing. It is up to the consuming applications to use them as it best fits their functional needs.

Each piece of data, called *metric* (detailed in Section 4.2.3), is somehow duplicated when entering the platform: one instance being persisted into a time-series database (TSDB) for later fetching (through the *Query* API), the other, ephemeral, being piped to the *Streaming* API for instantaneous notification. Internally, most of the data exchanges between components happens through a message broker. Last but not least, as this platform supports multi-tenancy, an Identity and Access Management (IAM) solution, at the edge of the platform, ensures authentication and authorisation through OpenID Connect and User-Managed Access (UMA) 2.0 standards respectively. Data isolation is enforced through the concept of *scope* that is a platform specific data labelling mechanism (detailed in Section 4.2.2). The following subsection details the above mentioned capabilities and concepts and substantiates them with underlying technology choices and implementation.

4.2 | Key concepts and implementation details

4.2.1 | Interoperability

Obelisk allows secured data ingestion, storage, streaming and retrieval through HTTPs REST APIs. It sits as a middleware layer (as illustrated in Figure 2) between the data producers and the data consumers. However, due to the multitude of

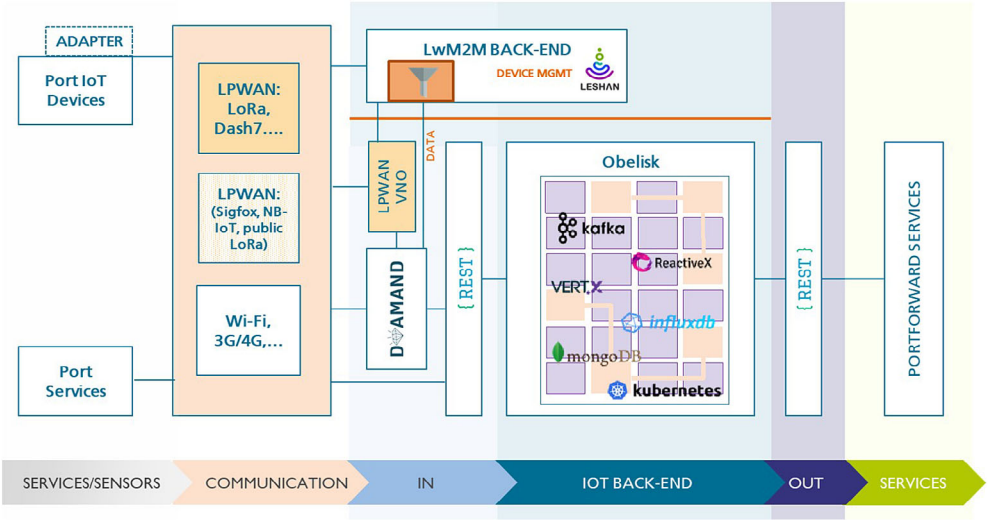

FIGURE 3 High level overview of the connectivity layer allowing interoperability amongst varying sources [Color figure can be viewed at wileyonlinelibrary.com]

IoT wireless connectivity protocols, it is clear that the ideal IoT application-enablement platform goes beyond the intake of data via APIs and requires protocols interoperability for both device management and data access. This is exactly the scope of the connectivity layer. As illustrated in Figure 3, this does not take place through a 'one fits all' approach but rather depends on the type of device/service that generates IoT data and the used communication protocol. In order to correctly apprehend the connectivity layer, its main components are hereafter briefly introduced.

Deploying IoT devices at ports scale requires industry standards to cater for their remote management. Due to their constrained nature (limited computing power, memory, batteries and/or network bandwidth), a simple, low-cost protocol for delivering remote management and service enablement that leverages on open Internet standards, that can work over limited bandwidth wireless connections and that has a limited implementation footprint is required. Consequently, in the context of the PortForward project, the Lightweight M2M (LwM2M) specification[32] from the Open Mobile Alliance (OMA) is used as IoT device management solution as:

- It is a free, open specification that leverages upon open Internet standards.
- Relying on the Constrained Application Protocol (CoAP), it is able to cater for constrained devices.[33]
- It has an extensible data model, leaving room for expansion towards various market areas.
- Open-source implementations for both the device and platform side are available (e.g., Eclipse Leshan, Eclipse Wakaama, AVSystem Anjay).
- There is a growing number of commercially available platforms that support LwM2M for performing device management (e.g., ARM mBed, AVSystem Coiote, Microsoft Azur Hub, Samsung Artik Cloud, Nokia IoT connector, Sierra Wireless, Bosch IoT Suite)
- Novel IoT modules, particularly Narrowband IoT (NB-IoT), are being shipped with firmware that implements LwM2M.

It is very well possible however that ports have some already deployed or to be deployed IoT devices that do not have support for LwM2M. Two approaches, both valid, are possible: the device management solution shipped with those devices is used and, in parallel, the data is injected within Obelisk through DYAMAND (introduced in next paragraph) or alternatively, those devices are converted into LwM2M compliant devices by performing a translation between the proprietary device management API and LwM2M API. In this second approach, the LwM2M backend forwards the data to DYAMAND before ultimately reaching Obelisk. To do so Hoebeke et al.[34] introduce an easily deployable LwM2M Management Adapter that is able to convert legacy APIs into LwM2M compliant APIs. The adapter is realised as a Docker container, that can be either deployed on top of the device itself, if sufficiently capable, or on a Cloud platform.

DYAMAND (DYnamic, Adaptive MAnagement of Networks and Devices) is a framework that offers a solution to overcome interoperability issues amongst the fragmented landscape of heterogeneous IoT protocols in a plug-and-play manner.[35] It provides the necessary abstraction for applications to remotely communicate with devices, regardless of

their technology.[36] The framework allows interoperability amongst virtually all protocols,[##] provided that their management API offers Discovery, Eventing and Control. In the context of the PortForward project, it is used as an adapter to Obelisk's *Ingest* API for all devices and services needing data format and/or protocol translation to access it.

Finally, the Virtual Network Operator (VNO)[34] is a multimodal cloud-based Low-Power Wide-Area Network (LPWAN) operator that homogenises and unifies both device management and data flows from different LPWAN technologies and by this offers the desired interoperability for those technologies.[||||] It facilitates the roll-out and usage of devices that can switch from one technology to another over time. It also offers interoperability to fleets of devices with different LPWAN technologies and allows device isolation when the same LPWAN network infrastructure is shared amongst different organisations. On the one hand, it unifies all communication, independent of the data exchange method that is being used, and provides a single point of attachment towards an IoT cloud platform or device management platform. On the other hand, in case ports roll-out their own LPWAN equipment, it can homogenise the way to manage such private deployments by means of LwM2M. Finally, the VNO caters all non-IP based interactions with LPWAN equipment (e.g., LoRaWAN Network Server) or platforms (e.g., Sigfox cloud) and, in the PortForward implementation, hands over follow-up processing to either the LwM2M back-end or DYAMAND, depending whether LwM2M compliance is met or not. Putting the pieces back together, some routing principles apply within the connectivity layer:

- All data flows originating from LwM2M devices go to LwM2M back-end and then from LwM2M back-end to DYAMAND before reaching Obelisk.
- Data flows originating from LPWAN devices may need to pass by the VNO. This is the case if multimodal LPWAN capability is required.
- Data flows originating from non-LwM2M LPWAN devices bypass the LwM2M backend and directly target DYAMAND (possibly through the VNO if needed). This can be the case for instance for NB-IoT devices using IP communication.
- Data flows originating from services may pass by DYAMAND.

To exemplify those principles, various paths through the architecture are possible, such as:

1. An existing port service injecting IoT data over a typical Internet connection to Obelisk. This path is illustrated in Figure 4(A).
2. An existing port service injecting IoT data over a typical Internet connection to the IoT platform, but with an intermediate translation by DYAMAND from data format X to the the defined format of Obelisk's *Ingest* REST API. This path is illustrated in Figure 4(B).
3. A LwM2M compliant IoT device that communicates over Wi-Fi with the LwM2M back-end, from where its data is passed to Obelisk through DYAMAND. This path is illustrated in Figure 4(C).
4. A LwM2M compliant IoT device uses LPWAN connectivity and non-IP data exchange, upon which its data passes via the LPWAN VNO to reach the LwM2M backend and subsequently Obelisk through DYAMAND. This path is illustrated in Figure 4(D).
5. A non-LwM2M device that uses LPWAN connectivity and non-IP data exchange, upon which its data passes via the LPWAN VNO to go to DYAMAND to finally reach Obelisk. This path is illustrated in Figure 4(E).

4.2.2 | Multi-tenancy

As explained in the introduction, ports operate in a hub model, with numerous actors, requiring multi-tenancy. Special care must thus be taken in (i) securing the data transport to and from Obelisk, (ii) isolating the data processing and storage amongst tenants within Obelisk, (iii) fairly distributing Obelisk's resources consumption amongst tenants and (iv) monitoring and alerting to detect and act upon anomalies. To meet those requirements, Obelisk makes use of proven and widely adopted standards and protocols, further presented in this section.

[##] Zigbee, EnOcean, UPnP, Z-Wave, Philips hue, LwM2M, NGSI are examples of protocols already supported by DYAMAND.

[||||] At the current implementation state, the VNO is able to support five different communication technologies (LoRaWAN, SigFox, DASH7, NB-IoT and BLE[37]) as well as three different modes of communication (end-to-end IP, SCHC and non-IP communication).

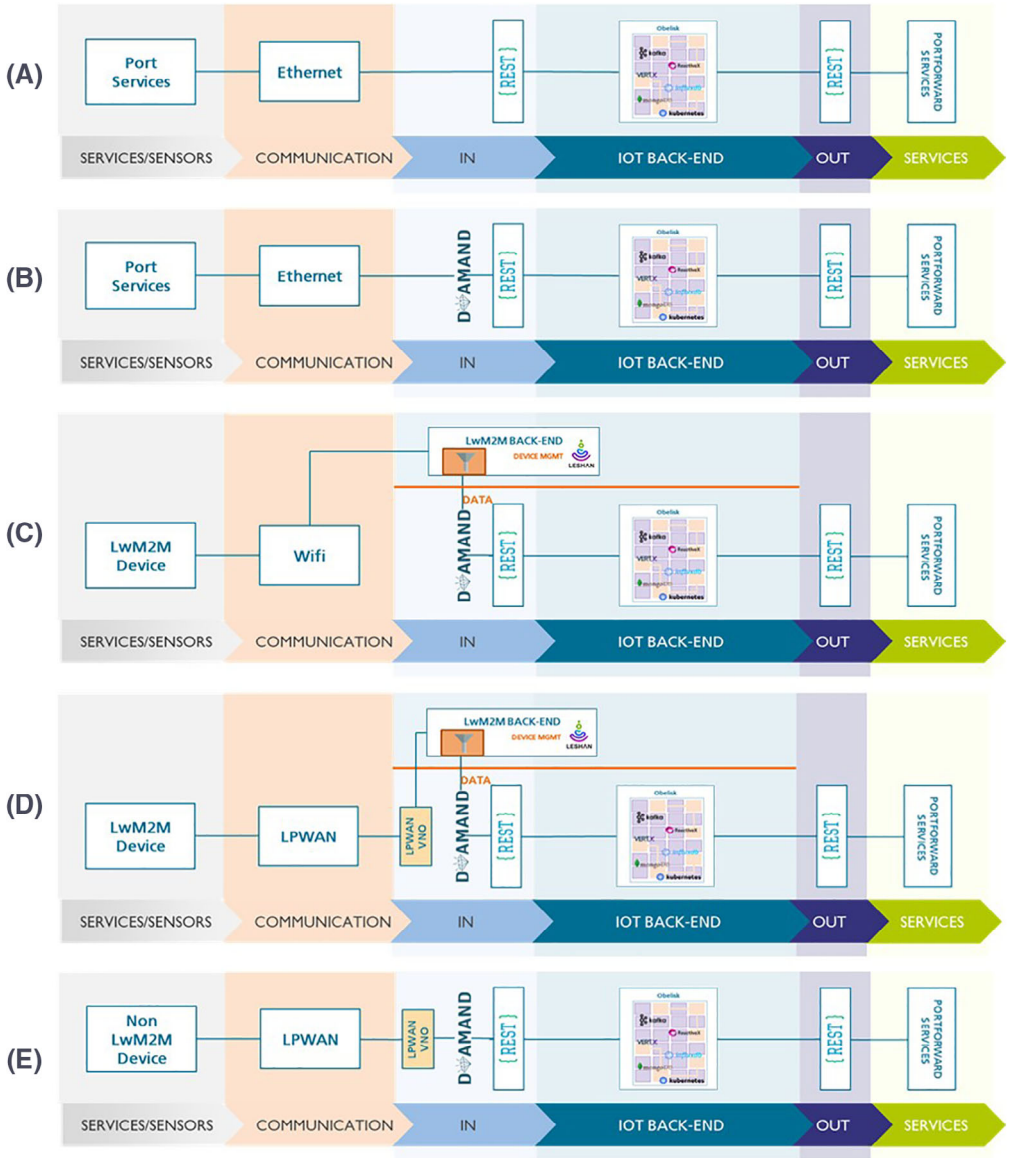

FIGURE 4 Overview of communication paths through the PortForward IoT data hub architecture [Color figure can be viewed at wileyonlinelibrary.com]

With regards to data transport, Obelisk ensures server authentication, data confidentiality and integrity towards its clients (devices, applications, end users) through the use of the Transport Layer Security (TLS) protocol, a proven and widely adopted cryptographic protocol designed to secure communication over a computer network, and thus exposes its APIs only as HTTP**s** endpoints.

With regards to data processing and storage, multi-tenancy is enforced within Obelisk through the combination of (i) a flexible data isolation mechanism called *scope*, (ii) UMA 2.0, a fine-grained authorisation management standard and (iii) OpenId Connect, an open authentication management standard. Those three dimensions are hereafter further detailed.

Obelisk internally ensures data isolation through the concept of *scopes* which represent logical data sets with configurable perimeter. Parameters used for perimeter refinement may be time-based (bounded: only records for a given time period can be accessed or unbounded: only records as of or until a given moment can be accessed), location-based (only records emitted from a specific geographic area can be accessed), metrics-based (only records for a (set of) metric(s) can be accessed), tag-based (only records with associated tag expression can be accessed) or any combination of the parameters listed above. A single piece of data may be accessed through multiple scopes, which can thus be understood as a labelling mechanism aiming at isolating data between different contexts of use. Data access APIs offered by Obelisk for data ingestion, querying and streaming all require scope mention in their request URL (e.g., for the *Streaming* API: .../scopes/*{scopeId}*/streaming...).

Users (devices, end-users and applications) are granted access rights to scopes (which API calls they can execute) through UMA 2.0, a cloud-oriented authorisation management standard. Through its decentralised nature, it fits particularly well within an IoT application context. Indeed, it provides resource owners with the ability to directly manage access rights to the resources (the devices, their operations and their data) they own through a flexible and fine grained delegations approach, exempting centralised management of complex access-rights matrices.[***]

Additionally, users (devices, end-users and applications) authentication is performed through OpenID Connect standard, with Obelisk as Relying Party. As a result, Obelisk doesn't manage directly identities but rather relies on other Identity Providers.[†††]Main advantages of this standard is that it alleviates the Relying Party (here Obelisk) from hosting a specific authentication system and more importantly, through single sign-on (SSO) and identity propagation afterwards, greatly eases cloud applications integration and ultimately the end-user experience.

Finally, to ensure fair resource distribution amongst tenants, the platform also provides a network traffic throttling mechanism that will temporarily degrade API bandwidth for over-consuming tenants or users (devices, end-users and applications). Moreover, a configurable monitoring and alerting system, based on the widely adopted Prometheus[‡‡‡]and Grafana[§§§]complementary open-source packages, allows to detect anomalies for subsequent corrective action.

4.2.3 | Scalability

Obelisk has been designed and implemented by means of an event-based, asynchronous micro-services software architecture. Micro-service software architectures promote application development as a set of distributed small and independent services each one of those focusing on a specific task. Appropriate service decomposition leverages on loose coupling for the inter-relationships amongst them while ensuring high cohesion of purpose within them. Micro-services commonly interact through asynchronous messages (events) which strengthens loose coupling and is enabled through reactive development frameworks (e.g., reactiveX) or by an intermediate message bus infrastructure, or a combination of both. Micro-services architectures are typically associated with the following benefits of particular interest for cloud-based IoT platforms:[38]

- Technology heterogeneity is enabled by the independence of micro-services amongst them and allows coexistence of different technologies used by different components in the system.
- Resilience and ease of deployment are enabled by services decomposition that promotes frequent and controlled changes in the system.
- Composability enabling the creation of new system capabilities by composing and reusing existing services.
- Scaling with micro-services can be achieved by horizontal duplication (adding more parallel instances of the same service), by data partitioning (services only manage the data they need to perform their tasks) and by functional decomposition (limiting their purpose).

Obelisk's composing micro-services are encapsulated within containers that da Silva et al.[39] define as a technology that provides OS-level virtualisation to isolate processes and define system usage limits for resources such as CPU, memory, disk I/O and network. It acts as an abstraction at the application layer that packages code and dependencies (application code, runtime, system libraries, settings, etc.). Multiple containers can thus simultaneously run on the same machine and share the operating system kernel with other containers, each running as isolated processes in user space. Container technologies offer unprecedented agility in developing and running applications in a cloud environment,[40] especially when combined with a micro-services architecture.

Obelisk's containers are managed within a Kubernetes cluster. Kubernetes is an open-source system for managing and orchestrating clusters of containers. It provides tools for scheduling and deploying containers in the cluster according to applications scaling needs. Kubernetes is designed to be extensible and fault-tolerant by allowing containers

[***]However, at the time of writing this article, although relying on UMA 2.0, no specific user interface nor API has been developed within Obelisk to support such decentralised management.
[†††]At the time of writing this article, Google is the only supported Identity Provider.
[‡‡‡]Prometheus is a cloud-based open-source monitoring and alerting system.
[§§§]Grafana is an open-source metrics analytics and visualisation suite.

to restart and move across the cluster as needed.[39] Both Docker (as container technology) and Kubernetes (as containers scheduling and orchestration technology) have been chosen for their ease of use, high maturity and dominant market position.[39,41]

Obelisk further relies on a set of widely adopted open-source packages for the infrastructure part (IAM, load balancers, message broker, databases as well as logging and monitoring) as well as on dedicated software components implementing the desired logic and data flows. More specifically, the platform uses:

- *Apache Kafka* as message broker. It inherently decouples data consumers from data producers and through its 'publish/subscribe' pattern that allows asynchronous messaging by ensuring delivery of all messages from a topic to all consumer groups that subscribed to that topic. This mechanism allows different applications (consumer groups) to consume at their own pace and process their own way the same messages. In addition to this key pattern for event-based micro-services platforms Kafka complements it with a more fine grained mechanism allowing members of the same consumer group to share the load amongst them, paving the way for micro-services scalability: topic messages are spread across partitions defined for that topic. Consumer group members are assigned to zero, one or more topic partitions and a topic partition can only be assigned to one member of each consumer group. Thus messages can be consumed at multiple consumer groups (decoupling and asynchronous messaging through'publish/subscribe') and within each of those, the load is spread across the members (scalability).[¶¶¶]Widely used as distributed event streaming platform by cloud applications[###], it provides fast and reliable delivery of considerable amounts of messages. Though recently another open-source distributed message broker, Apache Pulsar, has gained in popularity: while offering comparable functionalities as Apache Kafka it promises in addition easier and higher scalability through its stateless nature. Indeed, unlike Kafka, Pulsar splits brokering from data storage, the latter being delegated to Apache BookKeeper, a reliable, efficient and resilient distributed write-ahead log (WAL) system. However, unlike Kafka, Pulsar still lacks enterprise-grade commercial backing today.
- *InfluxDB* as time series database, optimised for fast storage and retrieval of time series data. InfluxDB is a widely adopted open-source TSDB specialised in the management of time-stamped data (sensor measure change over time), and (near) real time data ingestion (sensors continuously stream data). Obelisk refers to those sensors data as *metrics*. A metric is a type of measurement like temperature, humidity, etc. Metrics are stored in the TSDB together with the related emitting device/sensor id, the timestamp and the coordinates (position) of each measurement.[|||||]InfluxDB fulfills Obelisk's *metrics* main storage requirements in terms of supported data types (booleans, integers up to 64 bits, floating point numbers, string values and arrays of the aforementioned types), available operations (querying, filtering, ordering, aggregates and deletion) and compression rate versus access efficiency trade-off. Additionally, it also fulfills Obelisk's specific requirements for time-series IoT based data storage, hereafter shortly listed:
  (a) Data timestamping must not be set automatically at storage time by the TSDB but rather is provided along with the data to be stored. TSDB must accept 'past' timestamps due to the delay between the measurement and the moment it is effectively stored as well as 'future' timestamps to also support desynchronised devices. Additionally, the data store must support data from high frequency clock rate devices (>1kHz). This implies that millisecond based timestamps are inadequate to represent the individual data points and that the data store must have at least support for microsecond based timestamps. Further, new data events can be produced at any moment in time and should not be limited to a fixed interval.
  (b) In addition to timestamps, the data store must also support geospatial positioning for any data point and allow to link tags or labels (e.g., emitting device identifier, data isolation context (scope) identifier) to data points for efficient filtering purpose.
  (c) Data flows within Obelisk follow the at-least-once delivery principle. This implies that the same data can be written multiple times. The data store should be able to deal with this gracefully with minimal impact on performance.

[¶¶¶]Having more members in a consumer group than the number of partitions results in having unassigned 'starving' instances and conversely having more partitions than the number of members results in having members consuming messages from more than one partition. There should be at least as many partitions as there are members in each linked consumer groups for optimal performance.
[###]Kafka is being used by tens of thousands of organisations, including over a third of the Fortune 500 companies, from internet giants to car manufacturers to stock exchanges and has more than 5 million lifetime downloads[42]
[|||||]A more complete description of Obelisk's concepts may be found at: https://obelisk.ilabt.imec.be/api/v2/docs/.

However horizontal scalability and high availability are only proposed under commercial terms and are not available for the open-source version of the product. This limitation risks soon or later to become an issue as it represents a scalability bottleneck for the platform. Consequently this technology might be replaced in the future by a full-fledged open-source package that would meet the hereinabove listed requirements including horizontal scalability and high availability.

- *Vert.X* as development framework for the micro-services. Vert.x is a library for building reactive applications on top of the Java Virtual Machine (JVM). This framework has been chosen because it is lightweight, fast, event-driven and non blocking. It uses the *ReactiveX* library for composing asynchronous and event-based programs and implements the so-called *Multi-Reactor* pattern for events handling. In a standard reactor implementation (like it is the case in Node.js for instance), a specific thread called the event loop continuously delivers events to the different handlers as they arrive. Because no piece of code is blocking in asynchronous programming, an event loop can potentially deliver huge amounts of events in a short amount of time. However in such a configuration, there is only one event loop thread, consequently only running on one single core at any point in time. This may quickly become a limitation when having to scale an application over a multi-core server as it would require to (re-)design the application in a multi-processing fashion. Vert.x works differently: instead of a single event loop, Vert.x maintains several event loops (by default, the number of available cores) while ensuring that any particular handler will never be executed concurrently by always calling it using the exact same event loop.

Figure 5 depicts the data flow within the platform. The *Ingest* API expects as request body a JavaScript Object Notation (JSON) array representing a batch of 1..n metric events. Once the entire request is received, it directly sends an acknowledgment back to the emitting device before internally publishing *n* individual events to the *metrics.events* Kafka topic. The *sink* service as well as the *scope streamer* service are both subscribers of this topic and consequently consume the enqueued messages. As they are part of two distinct consumer groups they both receive and process at their own pace all messages. The *sink* service accumulates those individual metric events and performs a batch write when one of the two following conditions is met: the last batch write did happen *x* milliseconds ago or the amount of buffered events equals *y*. Both *x* and *y* are configurable parameters of the *sink* service. The *scope streamer* service also consumes those individual metric events and based on their respective scope forwards them to the appropriate topic: one topic being defined per scope (*metrics.events.***s**where **s** is the scope name). When a client application is willing to consume those streamed events, it calls the *Streaming* API (with a HTTP GET request) which continuously fills the HTTP GET response with the metric events it gets from the metrics.events.*scope* topic as they arrive.

This design sensibly restricts the overall scalability potential of the platform. Indeed, if technically there is no restriction on the number of parallel instances of the *streaming* micro-service, and even if each of those instances can simultaneously consume different streaming flows, by design only one at a time can consume events from a given scope topic as all are members of the same consumer group. Distributing the scope specific topic on more partitions is not an option either and would generate even worse results: indeed, each Kafka consumer (all members

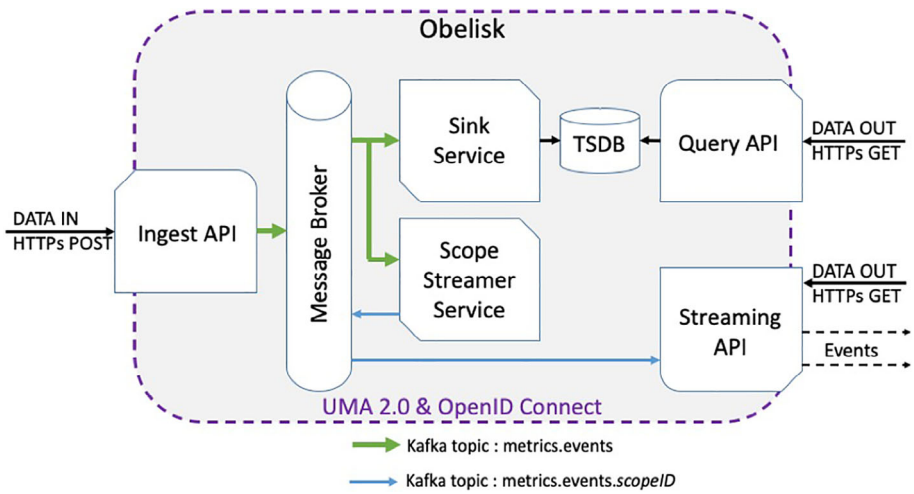

FIGURE 5 Obelisk data flow from ingest side to query and streaming side [Color figure can be viewed at wileyonlinelibrary.com]

of the same consumer group) would only receive a fraction (about $1/p$ where p is the number of partitions) of the topic messages.

Architecturally speaking, and as previously introduced, the *Streaming* API must be first contacted by the client application (through an HTTP GET request), thus only targeting one instance of the micro-service. This design only supports one open streaming flow at a time for a given scope, only consuming messages from one partition. At the time of writing this article, this design is being reconsidered and will be improved in a future release of the platform.

Finally, the *Query* API provides filtering and aggregation mechanisms. When being queried it fetches and responds the requested data from the time series database. A pagination system limiting the number of returned records per request completes this API.

Efficient scaling is achieved by a platform when its service time does not increase more rapidly than the growth of the load it has to process. In other words, its time complexity is expected to not significantly exceed a linear function, expressed as $O(n)$ where n is the load to be processed. Through its micro-services software architecture as well as the use of asynchronous event-based communication amongst them, and despite the two previously explained limitations (single *InfluxDB* instance and single streaming flow per scope), Obelisk offers the scalability required for the 'Smart Port' use-case. Indeed, results in Section 5.3.1 show that Obelisk exhibits a $O(\log n)$ time complexity, which means that its service time follows a logarithmic function, thus increasing less rapidly with the load it has to process than with a linear function. The next section demonstrates this, by means of an in-depth analysis of Obelisk' scalability performance.

5 | EVALUATION OF PORTFORWARD'S IOT BACKEND

As explained in Section 4, Obelisk, as cloud IoT data hub, provides secured integration of IoT data that is stored for and streamed to the consuming applications. Those applications benefit thus from this centralised abstraction layer to uniformly query historical events or be notified by those events as they occur. Furthermore this platform also acts as a buffer, able to absorb load peaks, thus shielding the consuming applications from those peaks. However, those advantages come at a cost as such a platform induces extra delay in the communication between devices and consuming applications (platform overhead) and implies a specific IT infrastructure that must be setup and maintained over time. Therefore special attention must be put on limiting the impact of those drawbacks as much as they possibly can. This section analyzes those drawbacks through an air quality monitoring use-case, which is introduced in the next subsection, before dissecting platform overhead (also called service-time) through three specific performance indicators.

5.1 | Description of the use case

Since ages and even more in today's global economy, ports play a vital economic role for the region they are located in providing direct and indirect jobs[****] as well as for trading with the rest of the world. However, those port activities generate an environmental impact for the surrounding community and complexify its relationship with the port. Consequently, and besides EU environmental objectives and regulations, today's EU ports feel more and more concerned about their environmental footprint and therefore need to measure and follow its evolution by taking corrective actions.

The performance analysis of the Obelisk platform is based on a dataset originating from the mobile air quality monitoring case-study[44] that has been developed for the 'City-of-Things' (CoT)[††††], the 'Smart City' project of the city of Antwerp in Belgium. For this case-study, air quality sensors have been mounted on the roofs of some delivery cars from the Belgian postal services 'bpost'. The sensors measure typical gasses such as Volatile Organic Compound as well as climate data such as temperature and humidity data, annotated with GPS locations. As such, they allow gathering real-time air quality information with broad city coverage, as opposed to an approach with static sensors which provide only limited local information. As each postal car is continuously driving around the city, on a longer time frame (i.e., a day), the set of sensors can cover the entire city in terms of measurements.

[****] Approximately 2.5 million full-time equivalents in Europe in 2014.[43]
[††††] https://www.imec-int.com/en/cityofthings

(A) Batch of aggregated metrics (286 bytes)


```
[
  [
    1586253168425,
    "bpost_car.metrics::number[6]",
    "bpost_car_00001",
    [
      31.04,
      2.5502517,
      4.34723,
      3.9945107,
      74.696106,
      5.3054137
    ],
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168426,
    "bpost_car.metrics::number[6]",
    "bpost_car_00001",
    [
      31.04,
      2.2674253,
      3.506697,
      3.4550116,
      73.582214,
      5.3054137
    ],
    [
      4.416488,
      51.238956
    ]
  ]
]
```


(B) Batch of individual metrics (916 bytes)


```
[
  [
    1586253168425,
    "light.lux",
    "bpost_car_00001",
    31.04,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168425,
    "air.pm1",
    "bpost_car_00001",
    2.5502517,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168425,
    "air.pm10",
    "bpost_car_00001",
    4.34723,
    [
      4.416488,
      51.238956
    ]
  ],
(...)
```

```
(...)
  [
    1586253168425,
    "air.pm25",
    "bpost_car_00001",
    3.9945107,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168425,
    "humidity.rh",
    "bpost_car_00001",
    74.696106,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168425,
    "temp.celcius",
    "bpost_car_00001",
    5.3054137,
    [
      4.416488,
      51.238956
    ]
  ],
(...)
```

```
(...)
  [
    1586253168426,
    "light.lux",
    "bpost_car_00001",
    31.04,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168426,
    "air.pm1",
    "bpost_car_00001",
    2.2674253,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168426,
    "air.pm10",
    "bpost_car_00001",
    3.506697,
    [
      4.416488,
      51.238956
    ]
  ],
(...)
```

```
(...)
  [
    1586253168426,
    "air.pm25",
    "bpost_car_00001",
    3.4550116,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168426,
    "humidity.rh",
    "bpost_car_00001",
    73.582214,
    [
      4.416488,
      51.238956
    ]
  ],
  [
    1586253168426,
    "temp.celcius",
    "bpost_car_00001",
    5.3054137,
    [
      4.416488,
      51.238956
    ]
  ]
]
```


FIGURE 6 Benefits of data aggregation in terms of network bandwidth and energy consumption [Color figure can be viewed at wileyonlinelibrary.com]

Figure 6(B) illustrates the typical JSON array structure for batch inserts as supported by Obelisk's *Ingest* API with a metric at each index position of the array. Metrics are defined as JSON arrays as well with the timestamp of the measure followed by the metric identifier and the originating device after which comes the measured value and finally the longitude and latitude coordinates of the measure as a JSON array. Although not used in the use-case, the elevation from the floor can complete the coordinates. In order to reduce network payload and consequently increase battery lifetime of the data emitting devices, the six different metrics can alternatively be regrouped as one JSON array at the cost of reduced querying and filtering flexibility. This approach is illustrated in Figure 6(A) and has been used for the hereafter described experiments.

As air quality remains since 2013 the first priority on the 'Top 10 environmental priorities' of the EU port sector.[43] the choice of this case-study is relevant to measure data throughput performance of Obelisk platform.

5.2 | Description of the testbed

The servers used to run the Obelisk performance experiments were all executed on the imec iLab.t testbed (Virtual Wall 2), part of the Fed4FIRE+ testbeds federation.[‡‡‡‡]Two independent slices (i.e., virtual experiment containers) were created: one hosting the Obelisk platform (the purple servers in Figure 7) and the other one hosting both the devices

[‡‡‡‡]https://www.fed4fire.eu/

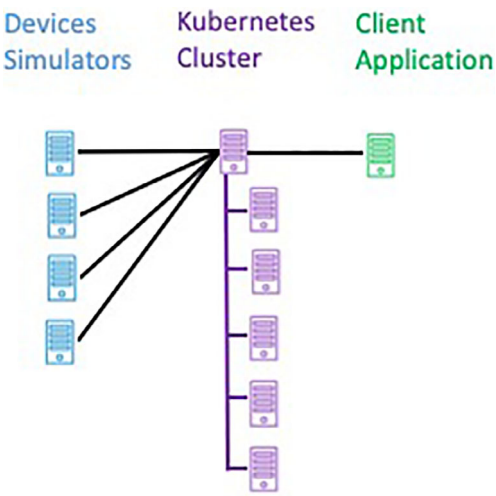

FIGURE 7 Testbed topology [Color figure can be viewed at wileyonlinelibrary.com]

simulators (developed in Vert.x v3.8.4 and hosted on the blue servers in Figure 7) simulating bpost postal cars air quality sensors and the client application (developed in Vert.x v3.8.4 and hosted on the green servers in Figure 7) receiving the events from Obelisk. All hosts are physical servers equipped with a 4 cores CPU (Intel E3-1220v3 (3.1GHz)), 16GB of RAM, a hard disk drive of 250GB and a 10 gigabit network connection.[§§§§] All servers use Ubuntu 16.04.1 LTS as operating system and are synced through the same time servers infrastructure.

5.3 | Analysis of platform performance

The selected approach to analyze platform performance and overall stability is to examine its behaviour under different loads. Those loads may vary in term of number of emitting devices as well in number of messages (metrics) sent per device.

A first experiment has thus been conducted with 16 successive runs of 30 seconds each.[¶¶¶¶] Those 16 consecutive runs are individually allocated a unique 'number of devices - messages size' mix (with 1, 2, 10 or 50 devices each sending one batch of 1, 2, 10 or 50 messages (metrics) per second to the platform). This first experiment focuses thus mainly on the analysis of platform stability: how the load affects overall platform performance.

Next, platform scalability is analyzed through different setups: the number of Kafka partitions are being increased as well as the number of instances of the *ingest*, *sink* and *scope-streamer* micro-services. In this second set of experiments, the number of metrics sent by batch is fixed to two; only the number of emitting devices is increased. By horizontally scaling the platform, the overall throughput of the platform is expected to increase.

The performance indicators used to assess both platform stability and scalability are:

- The round-trip time (**RTT**): the observed delay at emitting device side that is being spent between the start of a (batch of) metric(s) emission and the end of the reception of platform acknowledgment message. This indicator is of special importance in an IoT context, since radio should be used as frugally as possible due to its impact on battery lifetime.
- The internal streaming time (**IST**): the observed delay at platform side that is being spent between the reception of a (batch of) metric(s) and its restitution to a consuming application. In a data streaming context, it represents the platform overhead and can be seen as the time that is needed for a message to traverse the platform.
- The internal persistence time (**IPT**): the observed delay at platform side that is being spent between the reception of a (batch of) metric(s) and its persistence within the time series database of the platform.

[§§§§] For informational purposes, a comparable setup hosted at Amazon Web Services (AWS) for 3 years and composed of 6 EC2 standard reserved servers with comparable configuration, is estimated at a monthly fee of 342.17 USD, according to their online pricing calculator. This does not take into account the data storage needs that highly depend on metrics data volumetry and required persistence duration.

[¶¶¶¶] This duration is long enough to ensure the results are representative; indeed the platform supports higher sporadic peaks, here the purpose is to assess how sustainable the platform remains under such 'permanent' load.

Finally, the experiments that were run all respected a constant load pattern, where devices emit their (batch of) metric(s) every second. In order to not have a peak at the beginning of each second but rather to represent a constant load for the platform, devices are started in the first second of the experiment each with a configurable number of milliseconds delay. Of course, this distribution stabilises with higher number of devices. More importantly, it allows straightforward extrapolation for cases where the number of devices is higher with a proportionally lower emission frequency. For instance, the performance indicators obtained by the platform when 1000 devices emit a batch of 2 metrics per second will not drastically vary from a situation where 10,000 devices would send a batch of 2 metrics every 10 seconds.

5.3.1 | Analysis of platform stability: impact of different loads on platform performance

This section analyzes the platform stability, that is, how the chosen performance indicators (RTT, IST, IPT) evolve under different loads. To this end, 16 different experiments were executed respectively simulating 1, 2, 10 and 50 devices that send batches of 1, 2, 10 and 50 metrics every second, and this during 30 consecutive seconds. Main outcome are described below and visualised in Figure 8.

The RTT remains stable and under the 100 ms over time regardless of load. It mostly is influenced by the number of devices rather than the message size (number of metrics); however this difference is not really perceivable between 1, 2 and 10 devices and becomes more tangible for the four experiments with 50 devices. Of course the intrinsic value of the RTT is greatly influenced by the network, which in traditional IoT applications will most likely imply a significantly higher value; the interest of this indicator thus mainly resides in its relative stability over time and this under different loads offering high predictability on time of acknowledgment arrival and consequently allowing fine tuning, at device side, of radio time activation.

The IST remains quite stable for a given load, meaning the platform can sustainably ingest and render the events flow over time (no buffering effect). However this indicator slightly increases with the number of metrics contained in each

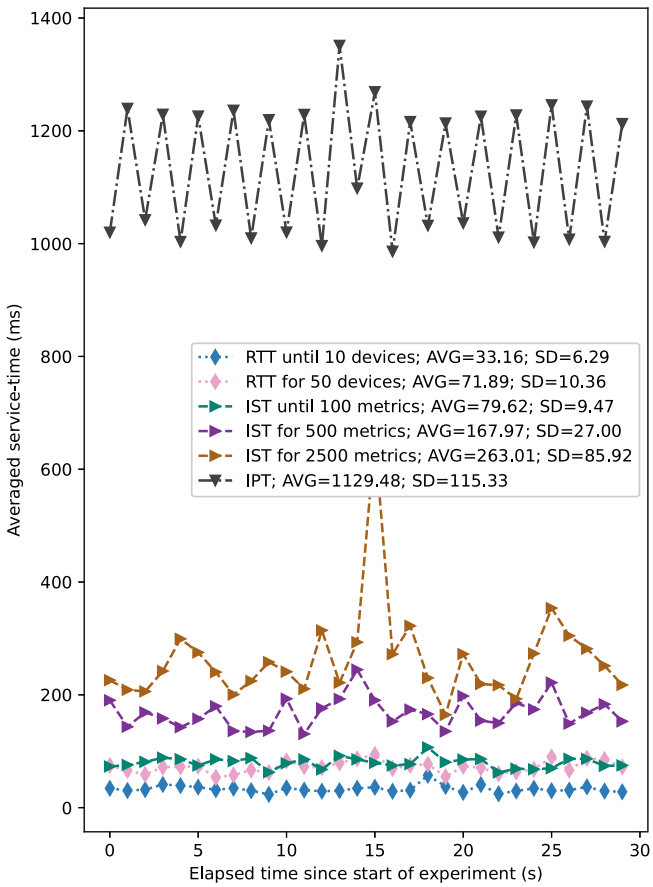

FIGURE 8 Analysis of platform stability under different loads [Color figure can be viewed at wileyonlinelibrary.com]

batch. Out of the 16 conducted experiments, three distinct categories appear: the first one regrouping experiments where the platform had to ingest until 100 metrics per second (that is, the experiments with 1 and 2 devices, the experiments with 10 devices sending batches of 1, 2 and 10 metrics per second and lastly the experiments with 50 devices sending batches of 1 and 2 metrics per second); this category behaves pretty much like the highest RTT observed and averages a delay of ≈80 ms. A second category, averaging ≈160 ms (twice as much as the first category) regroups experiments where the platform had to ingest 500 metrics per seconds (i.e., the experiment with 10 devices sending batches of 50 metrics every second and the experiment with 50 devices sending batches of 10 metrics every second). Lastly, the experiment with 50 devices each sending batches of 50 metrics per second, resulting in a 2500 metrics per second ingest rate at platform side, defines the third category and averages ≈260ms, which is 60% more than previous category while the number of ingested metrics is 5 times higher.

The IPT offers comparable results regardless the ingested load. The sink micro-service, buffers every metric to be persisted and flushes this buffer to the time-series database once every 2 seconds or as soon as it reaches 5000 buffered metrics before that delay (cfr. Section 4.2.3). This mechanism has been implemented as batch writes as those are by far more efficient than individual ones and a maximum delay of 2 seconds offers appropriate trade-off for our use-case. So an averaged IPT slightly higher than 1 second is being observed (≈1130 ms) as there are metrics arriving just after a flush that have to wait for 2 seconds before being stored but there also are metrics that arrive just before a flush and only wait a few milliseconds). The ≈130 residual milliseconds are explained by the averaged time spent between a metric arrival time at the ingest micro-service and its arrival time at the sink micro-service in addition to the time needed to flush and persist a buffer of metrics. Through this buffering mechanism this indicator remains extremely stable over time, whatever the load. Also, even though this delay is much higher than the ones observed both for RTT and IST, it remains acceptable for typical historical data querying contexts. For more demanding contexts it can be parameterised and tuned (by changing the default flushing time and size values).

The results obtained by this first experiment can also be used to model and explain the time complexity of the three performance indicators.

- The *RTT* evolves according to a logarithmic function for a growing number of concurrent metrics to be processed, as illustrated in Figure 9. This results in a $O(\log n)$ time complexity for data ingestion. This is explained by the fact the *Ingest* API directly acknowledges messages upon receipt before internally publishing each individual metric they contain. This performance indicator will thus increase for messages containing more metrics, but not linearly as it would be the case if message acknowledgment would only be sent after the processing of all metrics contained in the message.
- The *IST* also evolves according to a logarithmic function for a growing number of concurrent metrics to be processed, as illustrated in Figure 9. This results in a $O(\log n)$ time complexity for platform traversal time. This is explained by the asynchronous nature of the development framework used. As an example, when the *Ingest* API needs to send the n metrics it received to the Kafka message broker, it does not need to loop n times on a send and wait acknowledgment logic as it would be the case in synchronous programming but rather executes n times a send operation and may use the leftover CPU time for other activities like processing acknowledgments of previously sent metrics or accepting new metrics from devices. This performance indicator will thus increase for messages containing more metrics, but not linearly as would be the case in synchronous programming.
- The *IPT*, illustrated in Figure 8, evolves according to a constant function for a growing number of concurrent metrics. This results in a $O(1)$ time complexity for data persistence. This is explained by the buffering mechanism of the *Sink* service that collects metrics before executing a batch write operation to the database. This performance indicator will remain constant whatever the number of concurrent metrics.

In this section, it has been demonstrated that the three performance indicators remain stable over time for a given load. Additionally, for a growing load, both the RTT and IST exhibit a $O(\log n)$ time complexity pattern while the IPT models a $O(1)$ time complexity pattern. These results are in line with the scalability properties claimed in Section 4.2.3. This is of key importance in an IoT context where the load to be processed can exhibit vastly different information generation patterns over time (earthquake sensors, tracking devices, etc.). The next section analyzes the ability of the platform to scale by increasing the number of concurrent devices (fixing the number of metrics to 2 in order to limit the number of cases) and this under different configurations, by multiplying the *Ingest*, *Sink*, *Scope Streamer* and *Streaming* micro-services as well as the *metrics.events* topic partitions.

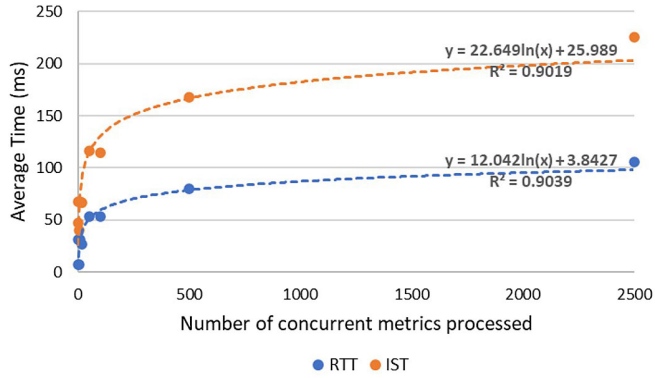

FIGURE 9 Evolution of RTT and IST under growing number of concurrent metrics [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 3 Components-Nodes distribution for each experiment

Component	Node_0	Node_1	Node_2	Node_3	Node_4
Ingest	1,2,3	-	2,3	3	-
Sink	-	1,2,3	3	2,3	-
Scope-Str.	-	-	1,2,3	2,3	3
Streaming	-	-	-	-	1,2,3
InfluxDB	-	1,2,3	-	-	-
Kafka	-	-	1,2,3	1,2,3	1,2,3
m.events	-	-	1,2,3	2,3	3
m.e.*scope*	-	-	-	-	1,2,3

5.3.2 | Analysis of platform scalability: impact of different setups, under different loads, on platform performance

This section analyses the platform behaviour and limits when scaling micro-services up, allowing higher parallelism in data treatment. To this end, the performance indicators will now be evaluated and compared amongst three different platform setups (1 instance of each micro-service, 2 instances and 3 instances)[####] and this with successive loads of 500, 1000, 1500, 2000, 2500 and 3000 devices sending 2 metrics per second. Additionally, experiments of 50 devices sending 50 metrics per second have also been conducted in order to analyze the impact of another emission pattern (less devices sending more metrics). Table 3 illustrates main components distribution amongst the Kubernetes cluster for the three tested setups.

Figure 10 illustrates this comparative analysis (with the number of metrics per second on the x-axis and the time in milliseconds on the y-axis), of which the main conclusions can be hereafter summarised:

- Config 1 succeeds in supporting up to 1500 devices, each sending a batch of 2 metrics per second.
- Config 2 succeeds in supporting up to 2500 devices, each sending a batch of 2 metrics per second.
- Config 3 succeeds in supporting up to 3000 devices, each sending a batch of 2 metrics per second.
- The mean and median of **IPT** across all experiments remain fairly close together (≈1200ms). This symmetrical distribution is explained by the buffering mechanism described above.
- The **RTT** remains stable across all experiments (it slightly increases with load but remains between 75 and 300 ms), meaning that the network accessibility to the ingest API is guaranteed to remain stable over time, whatever the load and the configuration. Noticeably though, config 2 and config 3 both reveal a slightly higher RTT than config 1 for same loads, which is explained by the routing logic needed once more than one instance of the ingest API is published.

[####] going higher than 3 parallel instances negatively impacted performance in our testbed configuration, no doubt that with a more powerful cluster, overall performance could benefit from a higher scaling, keeping however in mind the intrinsic limitation of the non duplicable 'events.metrics.*scope*' topic.

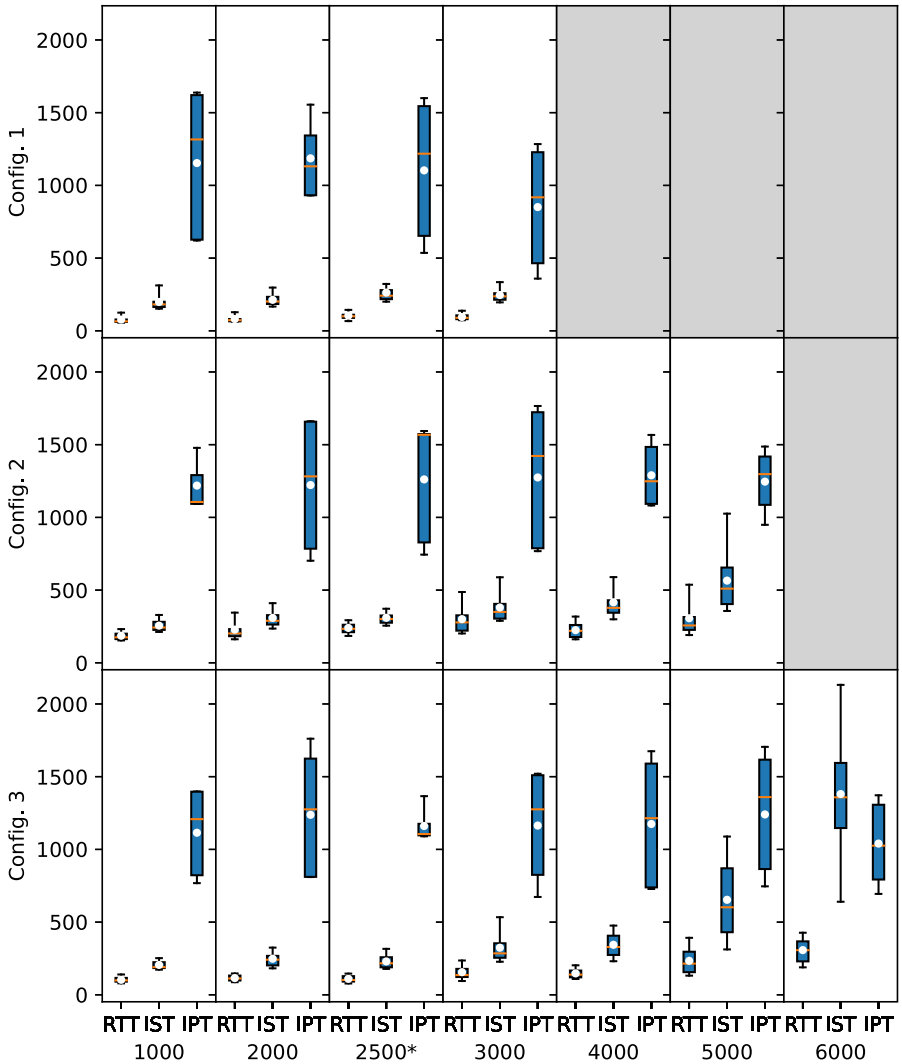

FIGURE 10 Analysis of platform scalability—measured service time for different configurations under growing load [Color figure can be viewed at wileyonlinelibrary.com]

- The **IST** remains stable under different loads in config 1. However, in configurations 2 and 3, the IST time complexity degrades from a logarithmic function to an exponential one from 5000 metrics onwards and thus becomes less stable while reaching the maximum supported load for the given configuration. At 6000 metrics ingested per second, in Config 3, the averaged platform traversal time observed nears 1,5 seconds which may be an issue for applications. Furthermore, the smaller degree of stability of this indicator (at max load) results in a lower performance predictability.

By horizontally scaling its composing micro-services the platform noticeably doubles its maximum processing capability (from 3000 metrics processed per second to 6000) with unchanged underlying infrastructure. However, while more metrics can be ingested in parallel when multiplying the number of micro-services instances it comes at the cost of IST. This is explained by the needed routing across multiplied components in the cluster. Indeed, when scaling containers out, they get deployed on (potentially) different nodes. So with a bit of chance a metric may traverse the platform on co-located (same node) containers all with relatively low load at that time (best case) or conversely in worst case, at each step of the pipeline the data may be sent to a container hosted on another node of the cluster which at that very moment might be highly loaded. This effect impacts thus IST predictability. So having more workers helps ingesting more data in parallel but doesn't allow faster platform traversal.

6 | FUTURE WORK

Section 4 presented, through its architecture and implementation, Obelisk's answers to the interoperability, multi-tenancy and scalability raised challenges for the 'Smart Port' IoT integration. Further development could however still be

envisioned at interoperability side by for instance exposing a CoAP data ingestion interface in addition to the current HTTP one in order to support direct connectivity for CoAP (and LwM2M) devices without the need for an extra adapter layer. In this scenario, the Delegated CoAP Authentication and Authorisation Framework (DCAF) could be used to provide Authentication and Authorisation management for the CoAP interface and by this still guarantee end-to-end multi-tenancy. Further research could possibly allow DCAF integration with the HTTP solution currently in place for homogenised management. Another area for further research, concerns the current *Streaming* API architecture and its underlying constraint with regards to scalability as expressed in Section 4.2.3.

Section 5 demonstrated, through an air quality monitoring use-case, Obelisk's effective stability and scalability properties. In order to further reduce the IST, further research can here also be envisioned with for instance the development of optimisation models for dynamic (re-)assignation of containers to cluster nodes based on the observed exchanged volumes amongst those containers as today Obelisk implements a static configuration setup.

Additionally, technology intelligence continuously brings further opportunities for improvements. As pointed out in Section 4.2.3, while not being technically validated yet, platform scalability might benefit from replacing the message broker Apache Kafka by Apache Pulsar as the latter decouples data brokering from storage. Replacing InfluxDB by another open-source TSDB fullfilling Obelisk's requirements would also alleviate the current scalability limitation imposed by the single node constraint of the InfluxDB license.

Finally, benchmarking Obelisk performance with other IoT cloud platforms would definitely bring a complementary perspective on the obtained results that are discussed in section 5.3. As an example, prior work from Araujo et al.[16] on Fiware platform performance could be used to this end.

7 | CONCLUSION

This article presents Obelisk, a scalable and multi-tenant cloud-based IoT integration platform used in the EU H2020 PortForward project, focusing on three key capabilities required for successful IoT integration within a port environment. Those three capabilities are (i) interoperability amongst the wide variety of wireless technologies and data formats, (ii) multi-tenancy to ensure appropriate data isolation amongst the various stakeholders operating within port premises and (iii) scalability to adapt platform performance according to data volumes patterns and consequently ensure stability for response time and service time (composed of the 'Internal Streaming Time' and the 'Internal Persistence Time') under varying loads. The way those three key capabilities have been designed and implemented within Obelisk is explained and documented as well as, for the scalability capability specifically, demonstrated by means of an air quality monitoring use case. The evaluation shows that the proposed reactive micro-service based design allows for horizontal scaling of the platform as well as for logarithmic time complexity of its service time. Finally, improvement tracks have been identified in those three dimensions to further enrich Obelisk services.

ACKNOWLEDGEMENT

The authors are particularly grateful for the assistance provided by the development team of the Obelisk platform (Thomas Dupont, Wannes Kerckhove, Sarah Kerkhove, Joachim Nielandt and Jasper Vaneessen) during the realisation of this research. The project PortForward has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 769267. - https://www.portforward-project.eu.

DATA AVAILABILITY STATEMENT

Data available on request from the authors.

ORCID

Vincent Bracke https://orcid.org/0000-0003-0913-6433
Merlijn Sebrechts https://orcid.org/0000-0002-4093-7338
Bart Moons https://orcid.org/0000-0003-4977-8916
Jeroen Hoebeke https://orcid.org/0000-0003-2039-007X
Filip De Turck https://orcid.org/0000-0003-4824-1199
Bruno Volckaert https://orcid.org/0000-0003-0575-5894

REFERENCES

1. Pasqua E. (IoT-Analytics), IoT connectivity market; 2018. https://iot-analytics.com/iot-segments/iot-connectivity/. Accessed May 26, 2020.
2. Autoritá di Sistema Portuale del Mar Tirreno Settentrionale, Relazione Annuale Sull' Attivita' Svolta Nell' Anno 2018 Dall' Autorita' Di Sistema Portuale Del Mar Tirreno Settentrionale; 2019. https://www.portialtotirreno.it/wp-content/uploads/relazioni_annuali/Relazione%20annuale%202018.pdf. April 30, 2020.
3. Autoridad Portuaria de Baleares, Plan de Empresa 2020 de la Autoridad Portuaria de Baleares; 2019. http://www.portsdebalears.com/sites/default/files/PLAN%20DE%20EMPRESA%%20DE%20BALEARES%202020%20signed.pdf. April 30, 2020.
4. Verhoeven P. European port governance - report of an enquiry into the current governance of european seaports; 2010. https://www.espo.be/media/espopublications/espofactfindingreport2010.pdf. May 26, 2020.
5. European Sea Ports Organization (ESPO), Trends in eu ports governance; 2016https://www.espo.be/media/Trends_in_EU_ports_governance_2016_FINAL_VERSION.pdf. May 26, 2020.
6. Yang Y, Zhong M, Yao H, Yu F, Fu X, Postolache O. Internet of Things for smart ports: technologies and challenges. *IEEE Instrum Meas Mag*. 2018;21:34-43.
7. Irannezhad E, Prato CG, Hickman M. An intelligent decision support system prototype for hinterland port logistics. *DecisSupport Syst*. 2020;130:113227.
8. W. Hofmann, F. Branding, Implementation of an IoT- and cloud-based digital twin for real-time decision support in port operations, *IFAC-PapersOnLine* 52 (2019) 2104 – 2109.9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.
9. Botti A, Monda A, Pellicano M, Torre C. The Re-conceptualization of the port supply chain as a smart port service system: the case of the port of salerno, systems; vol. 5 2017.
10. Heilig L, Voss S. Information systems in seaports: a categorization and overview. *Inf Technol Manag*. 2017;18:179-201.
11. Yau KA, Peng S, Qadir J, Low Y, Ling MH. Towards smart port infrastructures: enhancing port activities using information and communications technology. *IEEE Access*. 2020;8:83387-83404.
12. Pierleoni P, Concetti R, Belli A, Palma L. Amazon, Google and microsoft solutions for iot: architectures and a performance comparison. *IEEE Access*. 2020;8:5455-5470.
13. Dhirani LL, Newe T, Nizamani S. Can IoT escape cloud QoS and cost pitfalls. Paper presented at: Proceedings of the 2018 12th International Conference on Sensing Technology (ICST), Limerick, Ireland; 2018:65-70.
14. Opara-Martins J, Sahandi R, Tian F, Critical review of vendor lock-in and its impact on adoption of cloud computing. Paper presented at: Proceedings of the International Conference on Information Society (i-Society 2014), London, UK; 2014:92-97.
15. Ismail AA, Hamza HS, Kotb AM. Performance evaluation of open source IoT platforms. Paper presented at: Proceedings of the 2018 IEEE Global Conference on Internet of Things (GCIoT), Alexandria, Egypt; 2018:1-5.
16. Araujo V, Mitra K, Saguna S, ÅAhlund C. Performance evaluation of FIWARE: a cloud-based IoT platform for smart cities. *J Parall Distrib Comput*. 2019;132:250-261.
17. Patel K, Patel S. Internet of Things-IoT: definition characteristics, architecture, enabling technologies, application & future challenges. *Int J Eng Sci Comput*. 2016;6:6122-6131.
18. Farhan L, Shukur ST, Alissa AE, Alrweg M, Raza U, Kharel R. A survey on the challenges and opportunities of the Internet of Things (IoT). Paper presented at: Proceedings of the 2017 11th International Conference on Sensing Technology (ICST), Sidney, Australia; 2017:1-5.
19. Atlam HF, Alenezi A, Alharthi A, Walters RJ, Wills GB. Integration of cloud computing with internet of things: challenges and open issues. Paper presented at: Proceedings of the 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, UK; 2017:670-675.
20. Udoh IS, Kotonya G. Developing IoT applications: challenges and frameworks. *IET Cyber-Phys Syst Theory Appl*. 2018;3:65-72.
21. Kavre M, Gadekar A, Gadhade Y. Internet of Things (IoT): a survey. Paper presented at: Proceedings of the 2019 IEEE Pune Section International Conference (PuneCon), Pune, India; 2018:1-6.
22. Suárez JP, Trujillo A, Domínguez C, Santana JM, Fernández P, Managing and 3D visualization of real-time big geo-referenced data from Las Palmas port through a flexible open source computer architecture. Paper presented at: Proceedings of the 2015 1st International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM), Barcelona, Spain; 2015:1-11.
23. Fernández P, Santana JM, Ortega S, et al. Smartport: a platform for sensor data monitoring in a seaport based on FIWARE. *Sensors*. 2016;16:417.
24. Alonso A, Pozo A, Cantera JM, De la Vega F, Hierro JJ. Industrial data space architecture implementation using FIWARE. *Sensors*. 2018;18:2226.
25. Guth J, Breitenbücher U, Falkenthal M, et al. A detailed analysis of IoT platform architectures: concepts, similarities, and differences. In: Di Martino B, Li K-C, Yang LT, Esposito A, eds. *Internet of Everything: Algorithms Methodologies Technologies and Perspectives*. Singapore, Asia: Springer; 2018:81-101.
26. Latre S, Leroux P, Coenen T, Braem B, Ballon P, Demeester P. City of things: an integrated and multi-technology testbed for IoT smart city experiments. Paper presented at: Proceedings of the 2016 IEEE International Smart Cities Conference (ISC2), Trento, Italy; 2016:1-8.
27. Braem B, Latre S, Leroux P, Demeester P, Coenen T, Ballon P. Designing a smart city playground: Real-time air quality measurements and visualization in the City of Things testbed. Paper presented at: Proceedings of the 2016 IEEE International Smart Cities Conference (ISC2), Trento, Italy; 2016:1-2.
28. Santos J, Vanhove T, Sebrechts M, et al. City of things: enabling resource provisioning in smart cities. *IEEE Commun Mag*. 2018;56:177-183.

29. Moens P, Bracke V, Soete C, et al. Scalable fleet monitoring and visualization for smart machine maintenance and industrial IoT applications. *Sensors*. 2020;20:4308.
30. Molavi A, Lim G, Race B. A framework for building a smart port and smart port index. *Int J Sustain Transp*. 2019;14:686-700.
31. Tijtgat N, Van Ranst W, Volckaert B, Goedeme T, De Turck F. Embedded real-time object detection for a UAV warning system. Paper presented at: Proceedings of the 2017 IEEE International Conference on Computer Vision Workshops (ICCVW 2017), Venice, Italy; 2017:2110-2118.
32. OMA LwM2M specifications; 2019. https://www.openmobilealliance.org/release/LightweightM2M/. Accessed May 14, 2020.
33. Ishaq I, Carels D, Teklemariam G, et al. IETF standardization in the field of the Internet of Things (IoT): A survey. *J Sens Actuat Netw*. 2013;2:235-287.
34. Hoebeke J, Haxhibeqiri J, Moons B, Van Eeghem M, Rossey J, Karaaaç A, Famaey J. A cloud-based virtual network operator for managing multimodal LPWA networks and devices. Paper presented at: Proceedings of the 2018 3rd Cloudification of the Internet of Things (CIOT), Paris, France; 2018:1-8.
35. Nelis J, Verschueren T, Verslype D, Develder C. DYAMAND: dynamic, adaptive management of networks and devices. In: Pfeifer T, Jayasumana A, Turgut D, eds. *Proceedings of the Conference on Local Computer Networks*. Clearwater Beach, FL: IEEE; 2012:192-195.
36. Vandaele H, Nelis J, Verbelen T, Develder C. Remote management of a large set of heterogeneous devices using existing IoT interoperability platforms. In: Mandler B. et al. eds. *Internet of Things: IOT Infrastructures, IOT 360*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Cham: Springer; 2016;170:450-461. https://doi.org/10.1007/978-3-319-47075-7_49.
37. Basu SS, Haxhibeqiri J, Baert M, et al. An end-to-end LwM2M-based communication architecture for multimodal NB-IoT/BLE devices. *Sensors*. 2020;20:17.
38. Krylovskiy A, Jahn M, Patti E. Designing a smart city internet of things platform with microservice architecture. Paper presented at: Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud, Rome, Italy; 2015:25-30.
39. da Silva VG, Kirikova M, Alksnis G. Containers for virtualization: an overview. *Appl Comput Syst*. 2018;23:21-27.
40. Maenhaut P-J, Volckaert B, Ongenae V, De Turck F. Resource management in a containerized cloud: status and challenges. *J Netw Syst Manag*. 2020;28:197-246.
41. Flexera RightScale 2019 state of the cloud report from flexera; 2019. https://resources.flexera.com/web/media/documents/rightscale-2019-state%-of-the-cloud-report-from-flexera.pdf. May 20, 2020.
42. Narkhede N, Shapira G, Palino T. *Kafka: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, Inc; 2017.
43. Dooms M, Notteboom T, van der Lugt L, et al. European port industry sustainability report 2017. http://www.portopia.eu/wp-content/uploads/2017/11/FINAL-Sustainability-%Report-2017.pdf. May 06, 2020.
44. Santos J, Wauters T, Volckaert B, De Turck F. Fog computing: enabling the management and orchestration of smart city applications in 5G networks. *Entropy*. 2018;20:4:1-4:26.

How to cite this article: Bracke V, Sebrechts M, Moons B, Hoebeke J, De Turck F, Volckaert B. Design and evaluation of a scalable Internet of Things backend for smart ports. *Softw Pract Exper*. 2021;1–23. https://doi.org/10.1002/spe.2973