UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FRANCISCO PAIVA KNEBEL

# An open Digital Twin framework based on microservices in the cloud

Porto Alegre
2020

FRANCISCO PAIVA KNEBEL

# An open Digital Twin framework based on microservices in the cloud

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering

Advisor: Prof. Dr. Juliano Araújo Wickboldt

Porto Alegre
2020

**CIP — CATALOGING-IN-PUBLICATION**

Knebel, Francisco Paiva

An open Digital Twin framework based on microservices in the cloud / Francisco Paiva Knebel. – Porto Alegre: 2020.

80 f.

Advisor: Juliano Araújo Wickboldt

Trabalho de conclusão de curso (Graduação) – Universidade Federal do Rio Grande do Sul, Escola de Engenharia. Curso de Engenharia de Computação, Porto Alegre, BR–RS, 2020.

1. Digital twin. 2. Open source. 3. Cloud computing. 4. Fog computing. 5. Microservices. I. Wickboldt, Juliano Araújo, orient. II. Título.

*Dedico este trabalho às memórias de meu pai, Norberto, e minha mãe, Silvia.*

*À minha família.*

## ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Juliano Araújo Wickboldt, not only for the masterful guidance that resulted in this work but for advocating for Digital Twins and introducing this concept to me. I thank you for the assistance and the incredible availability. This work would not exist without him and the constant research we have done and hopefully will continue doing. I also thank my colleague in our endeavor, Rafael Trevisan, for another invaluable hand in expanding our research.

I would like to acknowledge all fellow members and friends of the UFRGS Computer Networks group, especially those involved in the JEMS3 project, which I am involved, for all the help and suggestions received, to further improve this work and other developed while acquiring this degree.

To my colleagues and cofounders of IDE, the UFRGS Institute of Informatics junior enterprise, where I learned so much about how to be who I needed to be, as a computer engineer. We aimed towards building a place where the institute's students could freely test until they succeed, experiment with their interest in computer science, providing experience and a helping hand to our community while building exciting solutions for modern problems. I had the pleasure of sharing my time with dozens of brilliant colleagues, bringing several projects to life, and I can not leave the "Mural of Bolsas" project out of this acknowledgment, where we built the official solution used by UFRGS in advertising and increasing accessibility to academic scholarships, to solve problems our members and other colleagues were facing.

To Rodrigo Madruga and Rodrigo Neves, friendships I will take from here to life, I have no words to condense it to a single page. To Rafael Calçada, a constant presence in developing this work, his contributions were essential in the maturity of this work. To Gabriel Ammes, I would like to thank you for several discussions about so many things, that helped me evolve this project to what it is and what it will be. To everyone else from the "Turma do Pagode", I thank you all. I want to acknowledge all others which I have not specified because one page is too small to list all and how their influence affected this work, so I thank you.

Finally, I would like to thank most of all for the help, care and love of my family, my brothers Norberto and João Guilherme. I wish that I could share this with our parents, but destiny had other plans. Everything I am and everything I will be is due to their support.

## AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer ao meu orientador, Juliano Araújo Wickboldt, não apenas pela orientação magistral que resultou neste trabalho, mas por defender os *Digital Twins* e me apresentar esse conceito. Agradeço a ajuda e a incrível disponibilidade. Este trabalho não existiria sem ele e as pesquisas constantes que temos feito e, espero, continuaremos fazendo. Agradeço também ao meu colega em nosso empreendimento, Rafael Trevisan, por outra mão inestimável na expansão de nossa pesquisa.

Gostaria de agradecer a todos os companheiros e amigos do grupo de Redes de Computadores da UFRGS, em especial aos envolvidos no projeto JEMS3, do qual estou envolvido, por todas as ajudas e sugestões recebidas, para aprimorar ainda mais este trabalho e outros desenvolvidos durante a aquisição deste diploma.

Aos meus colegas e cofundadores da IDE, empresa júnior do Instituto de Informática da UFRGS, onde aprendi muito sobre como ser quem precisava ser, como engenheiro da computação. Nosso objetivo era construir um lugar onde os alunos do instituto pudessem testar livremente até que tivessem sucesso, experimentar com seu interesse em computação, fornecendo experiência e uma mão amiga para nossa comunidade enquanto construíam soluções interessantes para problemas modernos. Tive o prazer de dividir meu tempo com dezenas de colegas brilhantes, dando vida a vários projetos, e não posso deixar o projeto "Mural das Bolsas" fora deste agradecimento, onde construímos a solução oficial utilizada pela UFRGS em publicidade e acessibilidade para bolsas acadêmicas, para resolver problemas que nossos membros e outros colegas estavam enfrentando.

À Rodrigo Madruga e Rodrigo Neves, amizades que vou levar daqui para a vida, não tenho palavras para condensar numa só página. À Rafael Calçada, presença constante no desenvolvimento deste trabalho, as suas contribuições foram essenciais no amadurecimento deste. À Gabriel Ammes, gostaria de agradecer as várias discussões sobre tantas coisas, que me ajudaram a evoluir este projeto para o que é e o que será. Para todos os outros da Turma do Pagode, eu agradeço por todos vocês. Quero agradecer a todos os outros que não especifiquei porque uma página é muito pequena para listar todos e como sua influência afetou este trabalho, por isso, agradeço.

Por fim, gostaria de agradecer acima de tudo a ajuda, carinho e amor da minha família, meus irmãos Norberto e João Guilherme. Eu gostaria de compartilhar isso com nossos pais, mas o destino teve outros planos. Tudo o que sou e tudo que serei se deve ao apoio deles.

*“An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil’s behaviour.”*

— Alan Turing, Computing Machinery and Intelligence

## ABSTRACT

Digital Twins are a concept of designing two interconnected mirrored spaces, modeling a real space with one virtual, each reflecting the other, sharing information, and making predictions based on simulations. In practice, Digital Twin platforms are often built as closed systems, limiting its operability with other applications, due to the lack of support in interconnecting these systems. An open-source approach enables interoperability and reduces the costs of design and implementation. To reflect their real counterpart, Digital Twins are composed of a large number of sensors, resulting in the need to store and analyze large amounts of data. Therefore, modern large scale Digital Twin systems rely on offloading computation towards cloud-based architectures, due to shared pools of hardware resources, but the significant large data volume can generate increased latency and slower response times. To reduce these effects, bringing processing closer to the edge devices, in a fog computing scenario, reduces the overall system latency, allowing for faster computing of the Digital Twin, and allowing for faster response times to the real physical system. This work proposes the creation of an open and expansible Digital Twin framework, built on a stack of cloud-based microservices, allowing for flexibility and reduced complexity for integrating with third-party applications. Tests performed using the framework in an emulated environment resulted in up to 64% reduction in average message transmission to the Digital Twin when deploying in fog computing nodes, compared to a cloud-only approach.



**Keywords:** Digital twin. open source. cloud computing. fog computing. microservices.

**Um framework aberto de Gêmeo Digital baseado em microsserviços na nuvem**

## RESUMO

Gêmeos Digitais são um conceito de projetar dois espaços espelhados interconectados, modelando um espaço real com um virtual, cada um refletindo o outro, compartilhando informações e fazendo previsões com base em simulações. Na prática, plataforms de Gêmeos Digitais são muitas vezes construídas como sistemas fechados, limitando sua operabilidade com outras aplicações, devido à falta de suporte na interligação desses sistemas. Uma abordagem de código aberto permite a interoperabilidade e reduz os custos de planejamento e implementação. Para refletir sua contraparte real, os Gêmeos Digitais são compostos por um grande número de sensores, resultando na necessidade de armazenar e analisar grandes quantidades de dados. Para isso, sistemas baseados em Gêmeos Digitais modernos de grande escala contam com a transferência de computação para arquiteturas baseadas em nuvem, devido ao compartilhamento de recursos de hardware, mas o significativo volume de dados pode gerar latência aumentada e tempos de resposta mais lentos. Para reduzir esses efeitos, aproximar o processamento aos dispositivos de borda, em um cenário de computação em névoa, reduz a latência geral do sistema, permitindo uma computação mais rápida do Gêmeo Digital e permitindo respostas mais rápidas ao sistema físico real. Este trabalho propõe a criação de um *framework* de Gêmeos Digitais aberto e expansível, construído sobre uma pilha de microsserviços baseados em nuvem, permitindo flexibilidade e complexidade reduzida para integração com aplicativos de terceiros. Testes executados com o *framework* em um ambiente emulado resultaram em uma redução de até 64% na transmissão média de mensagens para o Gêmeo Digital ao implantar em nós de computação em névoa, em comparação com uma abordagem apenas em nuvem.

**Palavras-chave:** Gêmeo digital, código aberto, computação em nuvem, computação em névoa, microsserviços.

## LIST OF FIGURES

## LIST OF TABLES

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| AI | Artificial Intelligence |
| AWS | Amazon Web Services |
| API | Application Programming Interface |
| CPS | Cyber-Physical System |
| CPU | Central Processing Unit |
| DAS | Data Acquisition Systems |
| DT | Digital Twin |
| GB | Gigabyte |
| IIoT | Industrial Internet of Things |
| IoT | Internet of Things |
| ML | Machine Learning |
| MQTT | Message Queuing Telemetry Transport |
| REST | Representational State Transfer |
| SaaS | Software as a Service |
| SoC | System-on-a-chip |

**CONTENTS**

# 1 INTRODUCTION

Digital Twins (DT) are models of physical systems, containing two interconnected mirrored spaces, one real and one virtual, which share real-time information, generating possible outcomes of the real system via simulations executed by the digital space. The concept of a DT, first introduced in 2002 (GRIEVES, 2016), has been used in various scenarios, from cloning entire factories, building smart cars, and space exploration.

A DT should accurately simulate, analyze, and predict real-world events and situations, via the collection of real physical data, mirroring it into a virtual space. DTs can not only anticipate and predict but can issue proactive measures to avoid problems by combining real-time data with the knowledge of other DT instances. Simulations can provide forecasts to their outcomes, allowing for humans to choose, knowing the consequences of said actions, and settle for the optimal solution (VOELL *et al.*, 2018).

In essence, a DT is a computer program that acquires real-world data about a physical system and generates simulations on how that system will be affected by this data. Long before the terminology, Digital Twins were used in action by NASA, to monitor and maintain machines in outer space, a situation where hands-on contact with devices would be impossible, allowing for remote diagnosis and fixing problems that are presented (GRIEVES; VICKERS, 2017). Space capsules were duplicated, mirroring the devices in orbit for diagnosing problems, but this presents the logical consequence of increased costs. DTs solve this problem due to being exclusively digital solutions, where a physical duplication of systems is not required, only the definition of the requirements that compose the physical system.

In practice, DT platforms are often built as closed and commercial systems, which by nature limits its interoperability between DT and other applications, like systems built for the Web or even other closed systems, due to lack of support in connecting systems with diverse specifications. Open-source implementations of DTs (DAMJANOVIC-BEHRENDT; BEHRENDT, 2019) enables interoperability and reduces costs of designing and implementing new solutions while providing a high-level microservices DT architecture, allowing flexibility and reduced complexity, maintaining each service working on small, isolated, and specific tasks while allowing the integration of heterogeneous software. While the proposed architecture is inspired by Smart Manufacturing and Smart Automotive standards (DAMJANOVIC-BEHRENDT, 2018), there are no specifications that would limit the adaptation of this architecture for different models of Digital Twins.

To reflect their physical counterparts, DTs are composed of a large number of sensors, responsible for acquiring the used data. The rapid expansion of sensors powered by the Internet of Things (IoT) is what makes DTs possible, but with this increased data generation, centralized applications oftentimes experience high latency and poor connectivity situations which affects the performance of applications. With the advent of IoT devices and its increased data volume in cloud systems, moving all data to the cloud does not satisfy time-sensitive application requirements. Fog computing enables computing, storage, networking, and data management on network nodes in closer proximity to edge devices, allowing for processing to happen not only in the cloud but also occurring between the edge and the cloud (YOUSEFPOUR *et al.*, 2019).

Literature related to this theme already exists, but is limited to abstract and conceptual proposals for these architectures. Therefore, this work will propose, implement, and analyze an open and extensible DT framework, built on a stack of cloud computing microservices, tested in two different scenarios: Case Study #1, which discusses the effect on added latency of cloud-based services and our solution of deploying the framework elements in a fog computing layer, bringing processing closer to the edge to reduce these impacts, and Case Study #2, where we analyze the communication limitations imposed by our implementation and how systems can work around these limits.

Bringing processing closer to the edge, to handle the DT message exchange, effectively reduced latency for the system, allowing for the processing unit to receive data from the source devices and compute its suggestion to the real system with a shorter response time. Scaling to the cloud enables ease of sharing hardware in a centralized location, but bringing resources closer to the edge enables data preprocessing, reducing overall traffic and latency. Deploying the framework considering these aspects, in a controlled emulated environment, resulted in up to 64% reduction in average message transmission from client devices to the DT when deploying in fog computing nodes, compared to a cloud-only approach. Simulations performed in this work consider deadline-based communications as their focus. Historic data, used to analyze previous system behaviour, do not present timing limitations and can be transmitted in the background, without deploying in a fog scenario. This work focuses on the communication aspect of the DT implementation. Data analysis and behaviour prediction is left as future work, where communication is made full cycle, considering data acquisition, decisions are computed by the DT, suggesting an action to the physical system.

# 2 BACKGROUND INFORMATION

The following sections will describe and define some key concepts in understanding what and how Digital Twins function. It will explain the concepts of cloud, fog, and edge computing and why are they important to DTs, defining DT and IoT as overlapping concepts and how DTs propose the evolution of IoT. This chapter closes with a look into other works in the DT scenario, how some solutions already exist, and how they are used.

## 2.1 Digital Twins

Digital Twins are not just a digital model or interface of a physical object. A twin receives input from a variety of sensors from their real-world counterpart. With this data, the DT can simulate the physical object it mirrors in real-time, feeding the real system with insights into their performance and potential problems that might occur, due to incorrect or invalid behaviors.

Another usage of a DT is on the design stage of a system, in which the twin would serve as the real system prototype, before any physical system is built, further reducing project costs. According to Voell *et al.* (2018), Digital Twins are defined by a couple of features:

- **Comprehensiveness**: Digital Twins must be able to handle data from multiple different types, to aggregate data from various sources. Multiple features of physical objects may be integrated into Digital Twins, ranging from basic identification to a fully detailed representation of objects.
- **Linkage**: Digital Twins are connected to multiple physical objects, but they can also be connected to other Digital Twins, being a part of an even greater system of linked representations of systems.
- **Interoperability**: Digital Twins must be flexible and allow for communication between components built by a wide range of different manufacturers. Digital Twins must also understand problems and determine paths to solve them by communicating with other Digital Twins, instead of operating in isolation.
- **Instantiation**: Differentiation between abstract and fully functional Digital Twins. Non-instantiated Digital Twins only contain methods and features they all share, while instantiated Digital Twins contain functionality obtained from live data, not

shared with other Twins.

- **Evolution and Traceability**: Digital Twins evolve, just as physical objects, and must keep track of their evolution. After acquiring knowledge, whether it is the addition of new components or from sensed data, Digital Twins keep track of their changes, allowing for an understanding of how the system or physical object evolved into its current state.

For Digital Twins to have the means of performing efficiently, they must know the current and future states of the system, as well as the ability to identify and evaluate disruptions, defining and acting on their solutions (KORTH; ZAJAC; SCHWEDE, 2018).

The implementation of self-aware systems like Digital Twins create modern control systems that freely adapt to their environment, not only mimicking but understanding the reasons behind the behavior of objects (STOJANOVIC; MILENOVIC, 2018). Systems like this provide both the insight of the models and implicit knowledge of objects, derived from their past behavior, only requiring the availability of raw data, unsupervised by foreign agents. The purpose of a Digital Twin is not the same as being a virtual prototype of physical systems, useful in aiding in the development process of products, but another level of modeling, where digital and physical coexists in a continuous feedback loop.

Building complex systems was never the limitation in building technology like Digital Twins. How to obtain, store, and define what data to collect is inherently a cost-based consequence. While the hardware for data acquisition and transmission may not be cheap in extreme environments, the cost of microcontrollers and other low-cost System-on-a-chip (SoC) boards enables the ease of connectivity between these devices via the IoT, making the DT concept possible.

The implementation of a cloud-based microservices DT provides the means for effortless integration between its parts and ease of access for the end-users, significantly accelerating product development and manufacturing. Components from the platform can be expanded and scaled on-demand, with independent development between each module, where enhancements made to each of the microservices are easily inserted with minimal changes to the entirety of the system (JOSEPH; CHANDRASEKARAN, 2019). The virtual component of a Digital Twin system is not necessarily implemented physically alongside the native physical devices. Due to its digital nature, it can be implemented physically apart from its real counterpart, in a decentralized manner (CARDIN, 2019).

Modern DT-based systems can not only implement simple, well established, busi-

ness logic, but also an intelligent system which can think and provide advanced feedback for its users, with the integration of Artificial Intelligence (AI) via Machine Learning (ML). DTs must support a collection of different models to accurately describe the device in their full life-cycles. As so, they require a collection of services to effectively simulate, analyze, and predict physical world events and situations that it mirrors.

## 2.2 Cloud, Fog, and Edge computing

For a DT to mirror any physical device, it must respect its real-time system timing requirements. The correctness of the system behavior depends not only on the logical results but also on the timing constraints defined for this system (KOPETZ, 2011). Applications in smart industries (KHAN *et al.*, 2020; WAN *et al.*, 2018), homes (VERMA; SOOD, 2018), cities (TANG *et al.*, 2015; PERERA *et al.*, 2017) and healthcare (NIKOLOUDAKIS *et al.*, 2017) already exist for implementing fog computing in real-time systems, where being time-critical is essential in preventing accidents and to provide a better and more reliable experience, due to reduced latency. Integrating fog computing with IoT and Digital Twins enables mirroring physical and virtual spaces while achieving the necessary timing requirements of real-time systems. For DTs, it is still challenging to manage, process, and store all this data in a highly distributed environment (DIZDAREVIĆ *et al.*, 2019; KHAN *et al.*, 2020).

Cloud computing provides means for convenient, on-demand access to shared computing resources (MELL; GRANCE, *et al.*, 2011), transferring the responsibility of data processing and storage to remote data centers, allowing for geographical distribution, which yields in increased system redundancy and reliability (MARINESCU, 2017; DENG *et al.*, 2010). Table 2.1 presents a multi-aspect comparison between cloud and fog, presenting strong and weak points of the two deployment scenarios. Fog differentiates itself from the cloud due to its nodes being deployed in large numbers at less centralized locations, compared to cloud data centers. Edge computing moves processing nearest to where it is needed, allowing computation closer to the source, reducing cloud traffic and service latency, improving response times (CHEN *et al.*, 2019).

Fog differentiates itself from the cloud due to decentralized nature. Fog computing extends cloud computing into the physical world and their multi-connected devices, ensuring data availability where and when it is needed (ATLAM; WALTERS; WILLS, 2018), while edge computing represents the nearest it can be placed regarding the data

source. This work excludes edge deployment, due to edge being composed mostly of devices without complete control from the cloud provider, presenting problems in regards to system reliability and availability. Fog enables latency reduction while still being in a more controlled environment.

Table 2.1 – Comparison between cloud and fog computing.

| **Items** | **Cloud Computing** | **Fog Computing** |
|---|---|---|
| Latency | High | Low |
| Hardware | Scalable storage and computing power | Limited storage and computing power |
| Location of server nodes | Within the Internet | At the edge of the local network |
| Distance between client and server | Multiple hops | One hop |
| Working environment | Warehouse with AC | Outdoor or indoor |
| Security measures | Defined | Hard to define |
| Attack on data | Less probability | High probability |
| Deployment | Centralized | Distributed |
| Location awareness | No | Yes |

Source: (ATLAM; WALTERS; WILLS, 2018)

## 2.3 Internet of Things and Industry

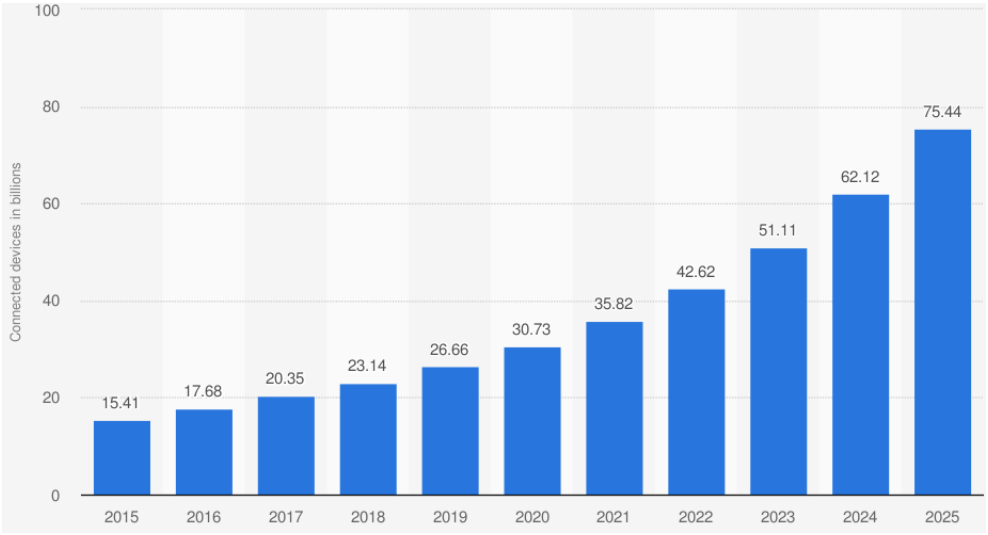
The Internet of Things is a long-established term for devices connected to the internet (ASHTON *et al.*, 2009), acquiring and sharing data without direct input from human beings, purely through these devices. Since its initial definition, the number of devices with internet access has exponentially grown in recent years. According to some research, the IoT market is forecasted to grow from 15.4 billion devices in 2015 to 75.4 billion in 2025 (LUCERO, 2016), as shown in Figure 2.1.

With the increasing number of devices and the promise of low-power embedded sensors providing efficient signal processing and communication, the IoT concept provides an opportunity to bridge the physical world and cyberspace, being the backbone enabling technology of Digital Twins (HE; GUO; ZHENG, 2018).

The Industrial Internet of Things (IIoT) is a derivation of the initial IoT definition, but detailing industrial processes. When researching this topic, two terms are important to be defined: Cyber-Physical Systems (CPS) and Industry 4.0.

A CPS could be defined as a system comprised of digital, physical, and human components, which are engineered to be interconnected and to work together (GRIFFOR *et al.*, 2017). DT, CPS, and IoT share a significant overlap between concepts, but DTs could be defined as an example of a CPS, which is a System of Systems, with combined cyber and physical parts that share data and evolve through this communication. Fig-

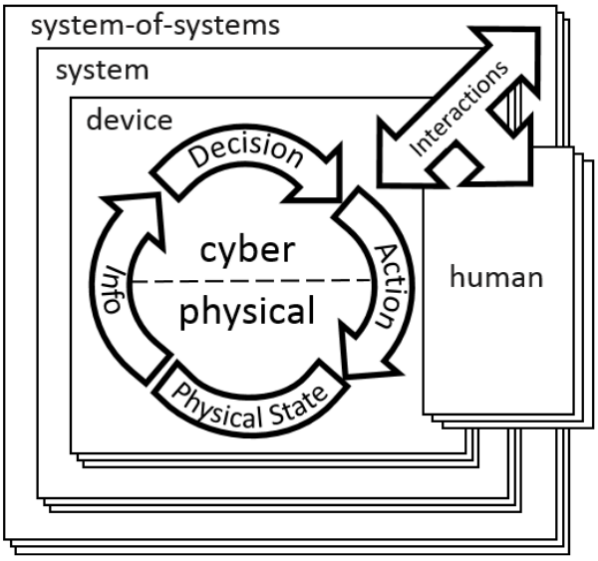
Figure 2.1 – Internet of Things connected devices worldwide from 2015 to 2025 (in billions)




Source: Image provided by author, data from (LUCERO, 2016), p. 5.

ure 2.2 presents a conceptual model for a CPS, where the cyber and physical parts of a device interact with each other through decisions from the cyber part, which generates actions performed by the physical, generating information which is fed back to the cyber part, closing the device information loop.

Figure 2.2 – NIST Cyber-Physical System conceptual model




Source: (GRIFFOR *et al.*, 2017), p. 6.

> CPS integrate computation, communication, sensing, and actuation with physical systems to fulfill time-sensitive functions with varying degrees of interaction with the environment, including human interaction (GRIFFOR *et al.*, 2017).

As for Industry 4.0, the term is used to define the most recent revolution in production, for the context of the burst of electronic and automated manufacturing, entirely

relying on the use of CPS for obtaining sensor data and communication, aiming in increasing efficiency, productivity, safety, and transparency of industrial processes (BOYES *et al.*, n.d.).

Linking IoT, Industry 4.0, and CPS we can define IIoT as the usage of information technologies to build a framework for improving industrial processes, through the use of automated data acquisition, product interconnection, and knowledge generation, powered by a multitude of different devices connected to the Internet, which instantly react intelligently to the changing environment, without requiring human interaction. A DT-powered system could implement all the concepts described for this scenario, improving manufacturing processes through increased knowledge and real-time responses, reducing production costs and waste (FULLER *et al.*, 2020).

Digital Twins could be defined as "what is after the IoT" since by definition IoT is purely acquiring a massive, ever-growing, number of data from multiple sources. With more devices connected and generating more information, organizations can compare these values and obtain a better idea of the functionality of their entire system. A DT is an idea of effectively sorting and combining this data into palatable information, both for machines that will keep improving themselves and generating more data and for any person interested in the process, that wish for their real system to function most efficiently.

## 2.4 Related Work

The Digital Twin concept is not relatively new, but newer technologies, like low-cost sensors for IoT and the development of efficient and intelligent Deep Learning, result in becoming technology enablers for DTs, making fully digital simulations of real-life objects not only a possibility but an approachable reality.

Studies combining CPS, DTs, and fog computing are not a new concept in related academic literature. The benefits of fog computing, which include low latency, locality, and scalability, are well known. CPS using DTs in the cloud, fog, or edge environments have been proposed (QI *et al.*, 2018) (KIM, 2019) as three levels of systems: unit, system, and system of systems (or service) level. The proposal by Qi (QI *et al.*, 2018) highlights that cloud computing enables on-demand resources and computational sharing, while fog computing shifts computation, storage, and networking of the cloud to the edge network, and edge computing allows processing geographically closer to the data source. Kim (KIM, 2019) states that the key driver to moving towards edge/fog comput-

ing is time-sensitive communication, which is a required feature for real-time systems. It also mentions that distinctions between the edge, fog, and cloud layer need to be further clarified to be better explored in different setups.

Mohan and Kangasharju (MOHAN; KANGASHARJU, 2016) propose a decentralized edge-fog IoT device network, where the edge is built with loosely coupled general-purpose devices (*i.e.*, phones, computers), with a horizontal connection between all the edge devices. The fog layer, in this approach, works as a summation of networking-related devices (*i.e.*, network routers, switches) where edge devices can offload intensive computations to. This model also includes a cloud background component, used only for centralized data storage, without providing any computational capacity. This edge-fog (and cloud) approach would not suffice in a high computation scenario, since it has no means of scaling back towards the shared resource pool of the cloud if the edge and fog become overload with tasks.

Ahmad and Afzal (AHMAD; AFZAL, 2019) state that although there are solutions for this domain, many aspects are still unexplored in practical scenarios, lacking an architecture that fully implements these principles.

The Digital Twin area of research still has many unexplored study areas, lacking on an actual deployment of a real implementation of DTs which utilizes the benefits of a cloud-fog architecture, presenting how it was built, its components and with experiment results proving its benefits. There are plenty of arguments suggesting the benefits of a DT implementation, from manufacturing to logistics in all industry sectors, but mainly motivational. Open solutions are scarce, both free or commercial, being treated only as assumptions or studies with no real, accessible implementations. This work aims to analyze what is needed for building a general DT, which can be extended to model any real physical device while being a reliable and open platform.

## 2.5 Use Cases

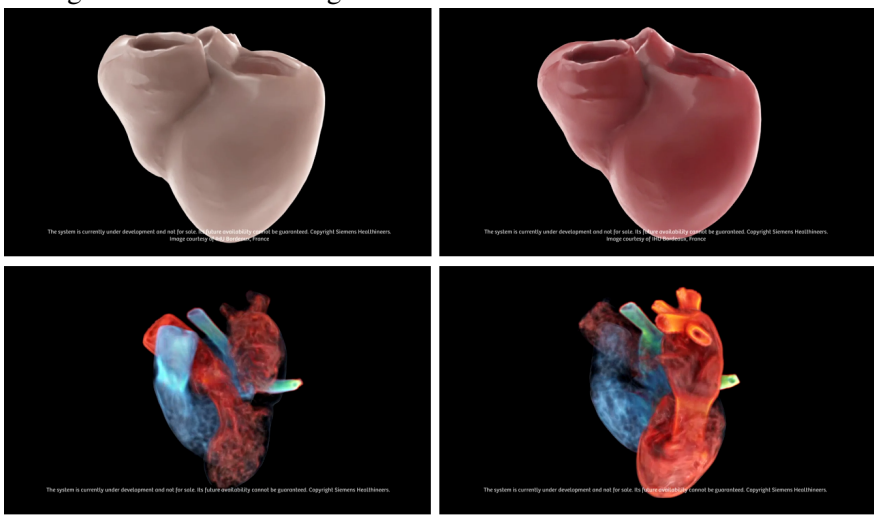
There are plenty of arguments suggesting the implementation of DTs in multiple areas, from manufacturing to logistics in all industry sectors. Any complex system, that is dependent and generates an output based on real-time sensor data could be made more reliable and efficiently by using Digital Twins.

### 2.5.1 Healthcare

DTs can identify problems with equipment in various medical fields. The vast amount of data generated could be used to analyze patterns, and coupled with the patient's medical history stored in the Cloud, could provide more information and insights leading to a successful and quicker diagnosis, taking into account the patient's unique medical history into account (KIELAR, 2019).

Work has been done in the field of DTs to engineer virtual organs. For instance, Siemens has modeled the human heart via the usage of DTs (SIEMENS, 2018), by using massive data sets from its medical devices and complex algorithms, to enable planning and prediction of recovery after procedures, simulating the physiological processes of a real human heart. This virtual heart can be controlled by sending electrical signals, simulating muscle contractions, as shown in Figure 2.3.

Figure 2.3 – Siemens Digital Twin visualization of the human heart.



Source: Video footage of Digital Twin of the heart, Siemens.[1]

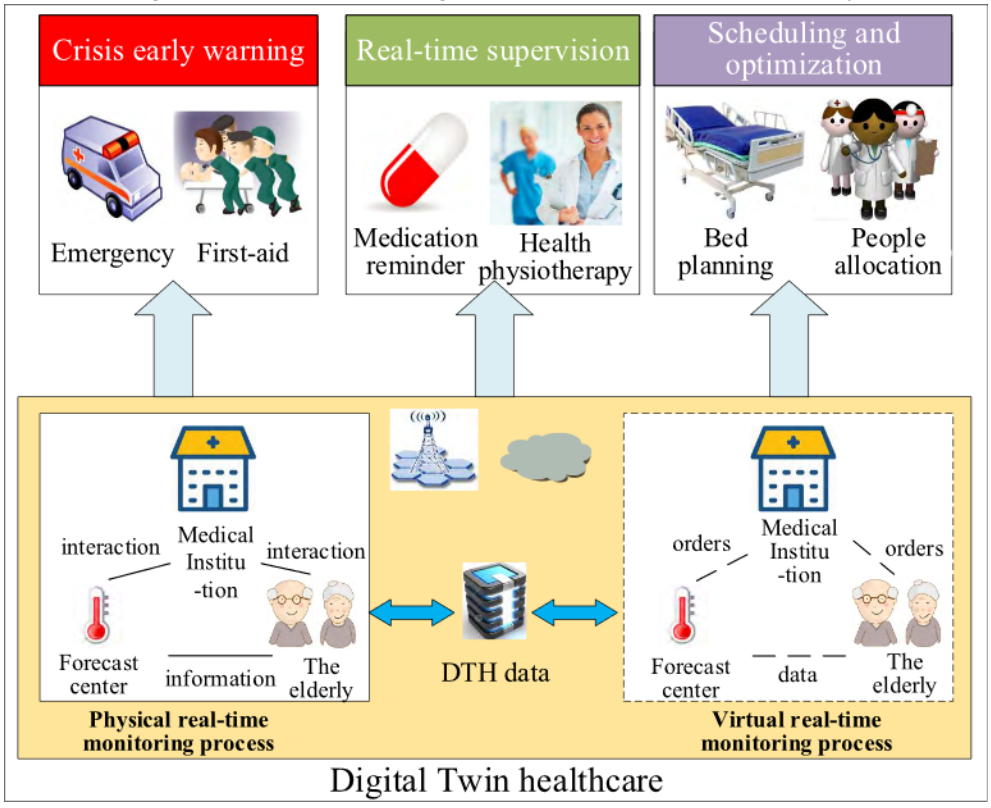
For individuals, DTs could enable the concept of personalized medicine: a virtual representation of an individual, whom every known drug could be tested and its behavior analyzed, reaching the optimal treatment for the patient. It would monitor the real person, alerting before any medical condition, to most effectively execute preventive measures.

In hospitals, DTs can be used to handle emergency scenarios, simulating how best to handle staff, patient beds, and access to testing rooms, with minimal downtime. Medical facility administration of resources becomes easier and automated, allowing for cheaper and improved access to medical care.

[1] Available from: `https://www.siemens-healthineers.com/press-room/press-videos/im-20181204001shs.html`. Visited on: 5 Oct. 2020.

Liu *et al.* (2019) proposes a cloud-based Digital Twin for providing healthcare services for the elderly, by monitoring, diagnosing, and predicting the health of individuals with data acquired via wearable medical devices, towards achieving personal health management. In the author's implementation, exposed in Figure 2.4, the Digital Twin acts on three separate fronts: early reaction to a crisis, real-time supervision, and optimization of scheduling. Data required for this process are weather forecasts, real-time physiological data from patients (e.g. blood pressure, heartbeat, current temperature), healthcare records provided by medical institutions, and data obtained from the Digital Twin models, enabling quicker response times in the healthcare assist of elderly patients.

Figure 2.4 – Healthcare Digital Twin framework for the elderly.




Source: (LIU *et al.*, 2019), p. 49098.

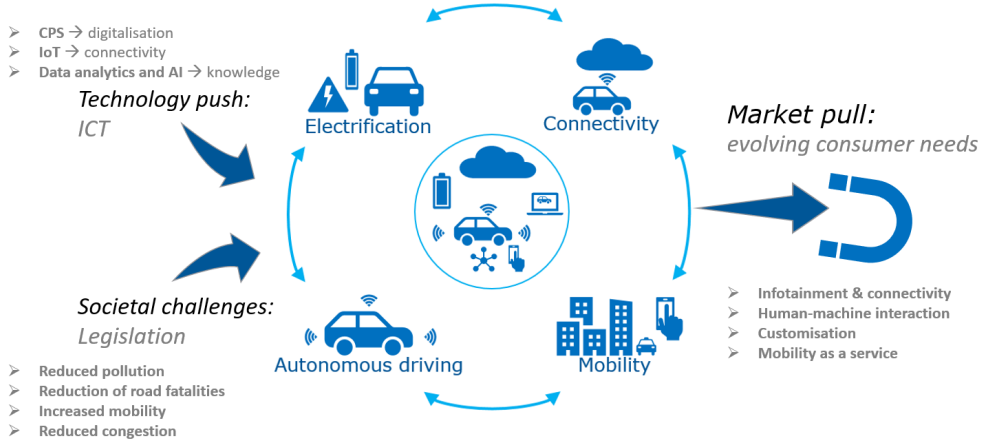
### 2.5.2 Automotive

Modern cars are designed with a variety of different, interconnected components. DTs for the automobile sector allows for a virtual model of a vehicle, capturing and analyzing the entirety of its parts, their performances, and how each part communicates. Development of new vehicles with DTs would include testing how the environment in real scenarios affects the car performance and allow for engineers to identify problems

and possible failures before the actual production process begin.

Smart cars, by definition, requires the integration of IoT and CPS to function, as shown in Figure 2.5. Sensors in cars, integrated with the aid of DTs, could record and understand the current state of the vehicle. The driver could check the status of its parts, receiving warnings if maintenance in specific parts is required. Manufacturers could use vehicle data to construct new parts, refining their products with data from their parts provided by drivers in real environments, allowing for improvement in safety and reducing costs for manufacturers.

Figure 2.5 – Automotive trend for connected driving.




Source: (VELEDAR; DAMJANOVIC-BEHRENDT; MACHER, 2019), p. 416.

### 2.5.3 Manufacturing & Engineering

Planning of manufacturing processes, acquiring sufficient process information, and the subsequent development of said process requires the largest overall time-consumption in manufacturing (UHLEMANN; LEHMANN; STEINHILPER, 2017). However, fully automated techniques for planning are not common practices and resulting data is manually analyzed with the aid of simulations.

The use of DTs for a production process enables coupling production systems with its digital equivalent as a means for optimization and full automation of data acquisition (UHLEMANN; SCHOCK, *et al.*, 2017). Virtual replicas enable testing new designs in an efficient form, simulating whole supply chains without shutting down production, which would result in a profit loss. Engineers can freely experiment and research, with no risk of reducing productivity.

### 2.5.4 Oil & Gas Industry

In Oil and Gas, definitions for DTs are still scarce and non-technical (CAMERON; WAALER; KUMULAINER, 2018), only being used as a commercial term, with no definition of an all-purpose DT, which could integrate solutions from multiple vendors, from many data sources, and simulation models.

For most industries, availability and monetization of data were the main motivators for digitalization, but for Oil and Gas (IRVING, 2018), industry leaders saw no advantage in changing their business model: oil prices were high, providing no reason for improvement. There were no proven use cases with apparent benefits against known methods that already worked.

Data science concepts were introduced but motivated by hype, distracting the focus that these technologies must be built with a solid foundation. The Oil and Gas industry has data generated from decades of use from physical equipment and assets, with little standardization between manufacturers, which provides no easy way of accessing and computing this data, which could be resolved with cloud storage and the usage of open software platforms, reversing the use of proprietary technology and allowing for partners, suppliers and experts to collaborate effectively (PARRY, 2018).

### 2.5.5 Cloud Services

For the bibliographic search made for this work, the only commercial and advertised Digital Twin platform found was Azure Digital Twins, currently defined as a preview, so the final product could be modified to what was reviewed. The service is described as a way to create digital models of environments to gain insights that can drive better products, generate operation optimization, and reduce costs (MICROSOFT, 2020a). The platform offers free trial access for a limited time, but instead of a platform walkthrough, the user is sent to the Azure Cloud Services dashboard, providing no introduction to the Digital Twin service.

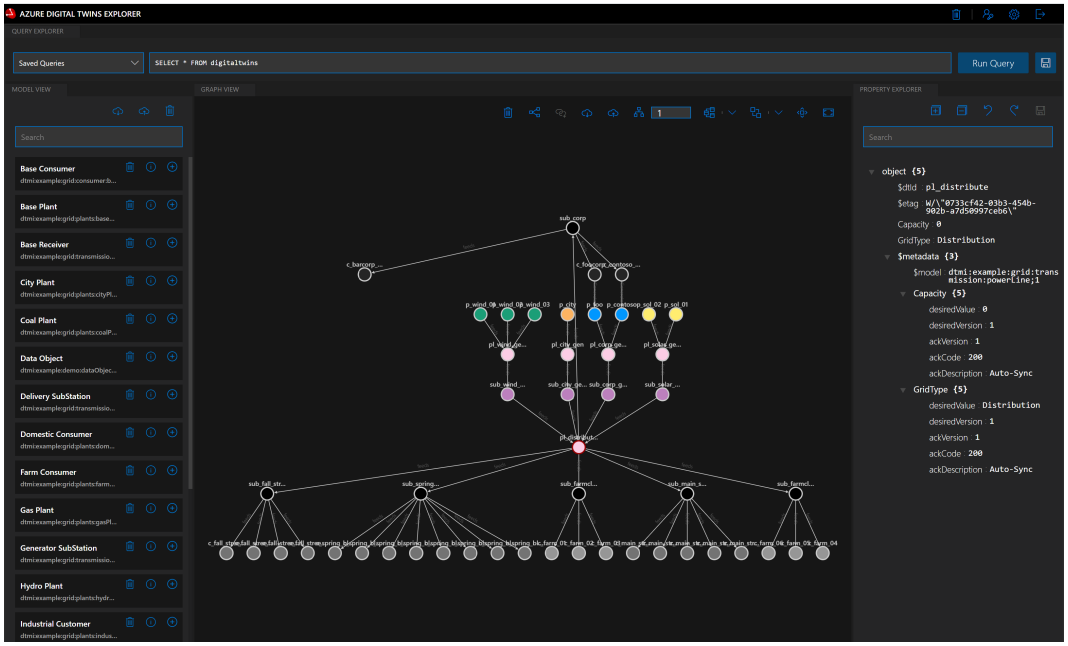
After searching the Microsoft-provided documentation[2], some small conceptual tutorials are presented, showing examples with the scope limited to designing Smart Buildings, without guides on how to define and create realistic representations of Twins

[2]Available from: `https://docs.microsoft.com/en-us/azure/digital-twins/`. Visited on: 27 Nov. 2019.

and how to work with it. The Digital Twin documentation is defined mostly by an automatically generated API documentation, without much detail on system functionality.

Most time spent on the platform was testing the APIs via trial and error, and with configuring other Azure Cloud Services dependencies: plenty of other services are needed to be linked with the Digital Twin service, like logging and data storage, to provide basic functionality, and even so, documentation on these are scarce and outdated. A Digital Twin platform like Azure Digital Twins should abstract generic configurations: all Twins will need databases and execution logs. This data must be easily reached by the developers and users, who need it for computation by the Digital Twin.

Figure 2.6 – Azure Digital Twins (ADT) explorer




Source: (MICROSOFT, 2020b)

Despite these problems, Azure Digital Twins allowed for easily creating new spaces and sensors with an open-source graph viewer (MICROSOFT, 2020b), providing a lightweight and adaptable front-end for the provided Digital Twin service. Figure 2.6 displays how modeling works in this tool. It did not allow for much other than creating new spaces, devices, or sensors, with the collection of these physical environments and associated assets in the graph becoming an ontology that describes that scenario virtually, being a tool where you can start to create your model. Azure Digital Twins is currently only a preview version and the final product is certain to change, but the difficulty in configuration, testing, and lack of integration with tools outside the closed Azure Cloud environment shows that it is not a definitive solution to the problems presented by this work.

# 3 PROPOSAL

The following proposed framework is built on a stack of different microservices. In contrast to monolithic software, built as one massive codebase, the microservices architectural approach of this solution allows for our DT to be developed as separate, self-contained, small pieces of software, responsible for solving each task. This isolated nature of each process allows for each application to be deployed individually, on-demand, each part working independently. Changes introduced to a specific part of the system only affect that module, and so only that part needs to be updated in case of software change. This also allows for microservices to be scaled via a software deployment tool (*i.e.*, Docker, Kubernetes) only on required services, reducing hardware resource waste.

This chapter will detail the design of the Digital Twin framework, discussing the challenges in defining how any software of this nature should be built ethically and openly, due to the systems and data it could be used with. Another discussion about the language of preference including an analysis of modern system bugs is included. The remainder of the chapter to follow discusses the elements belonging to the microservices architecture of the final framework design, from third-party services used, the modeling used to construct Digital Twins, and how those elements were programmed. The chapter closes with examples of how the defined services communicate for the deployed DT.

## 3.1 Motivation

Conceptually, Digital Twins impose a few ethical problems. The usage of raw real-time data is essential to the definition of a Digital Twin, to accurately match their physical counterparts. This allows for data pattern analysis, and with the growing number of cheap and accessible IoT-powered sensors, raw data from any source could be easily gathered by any interested party.

Digital Twins are not limited to reflect physical machines. In essence, DTs are a collection of multiple entry points of data, gathering any live information of a system, which could be an oil drilling rig, an autonomous smart car, or reflect the behavior of an entire city, all depending on the amount of data acquired, the historical data used and the algorithms that define the DT prediction behavior.

Integrating IoT and AI via ML create living digital simulation models that update and change as their physical counterparts change, continuously learning and updating

itself using sensor data relevant to its operating condition. In essence, a system of systems like this could be used to effectively understand the works of any collective system, how it behaves, what to expect of it giving certain scenarios, and means to obtain the input required for a system to function expectedly.

With this problem in mind, a Digital Twin of any system which collects a massive number of public data must have must be liable to the acquired data and the related computing process which manipulates it. Software scaled to the complexity of manipulating large amounts of public provided data must be accountable, transparent, apprehensible, and open.

- **Accountable**: DTs and all stakeholders involved in its usage must be held accountable for its actions and the consequences it generates. DTs should act most ethically and conscientiously as possible.
- **Transparent**: DTs should operate in a manner that is is easy for others to see what actions are performed and the motivations behind these actions. All actions performed by complex systems of this nature should not hide the nature of its computation.
- **Apprehensible**: Since DTs can be used not only as industry simulation tools but also as a major part of the future of entire living and evolving cities, DTs must be designed in a way to simplify its implementation to have the capacity of being understood by all involved. Algorithms should not be incapable of being explained, interpreted, or accounted for.
- **Open**: In computer software, open-source software refers to the underlying source code being freely available, allowing for usage, analysis, and modification. In computer security, debate still exists on the merit of full disclosure of vulnerabilities versus offering security by obscurity. For DTs, open systems also are essential, for enabling interoperability of different components.

Building an open platform allows for increased security in the platform through exposure. All the code is available for inspection and end-users can build the code themselves. Security through blind trust in software companies is not an ideal approach for software, as there is no guarantee it will work as expected. Open-source also allows anyone to fix errors as fast as possible, without needing to wait for a vendor to acknowledge the problem and decide if they will fix it. Open-source software is not inherently more secure than closed source, but they can be audited and fixed as vulnerabilities are detected,

by any party interested in maintaining the software. Due to the DT nature of connecting a multitude of different devices, each with their own specifications and standards, open systems enable easier interoperability between components.

Large-scale software is not inexplicable and parties involved with its development and deployment must be held accountable for actions that are a result of its usage. Algorithms are a definition of instructions that a machine will follow, without any analysis of what the content of these instructions represents. Until the time comes that software is not directly developed by humans and is fully automated by sentient AI (which will be software created by accountable humans), systems that act on, manipulate and intervene in acquired private human data must not simply be labeled as "complex", but should be required to have a responsible and transparent development through its entire life-cycle.

This project is designed and developed with these requirements in mind. All the project decisions, code repositories, and sample data used for testing are open[1] and can be freely audited by any person with access to GitHub. All work relating to this project is open and transparent, so that any future party that is involved in using or expanding its implementation, either into an upgrade or a newer and better Digital Twin framework implementation, can know every single aspect that drives its creation.

## 3.2 Programming Language

Before getting further into how the DT framework was initially built, an explanation is required of some aspects of its creation. One definition of programming languages is a set of rules that can be used to command a computer. The usage of high-level languages enables us to write efficient programs, using features native to that language that was built to simplify and produce higher quality code. Features that were built because available solutions in other languages did not have a satisfactory performance or required non-ideal implementations, resulting in hard to maintain or inexplicable code.

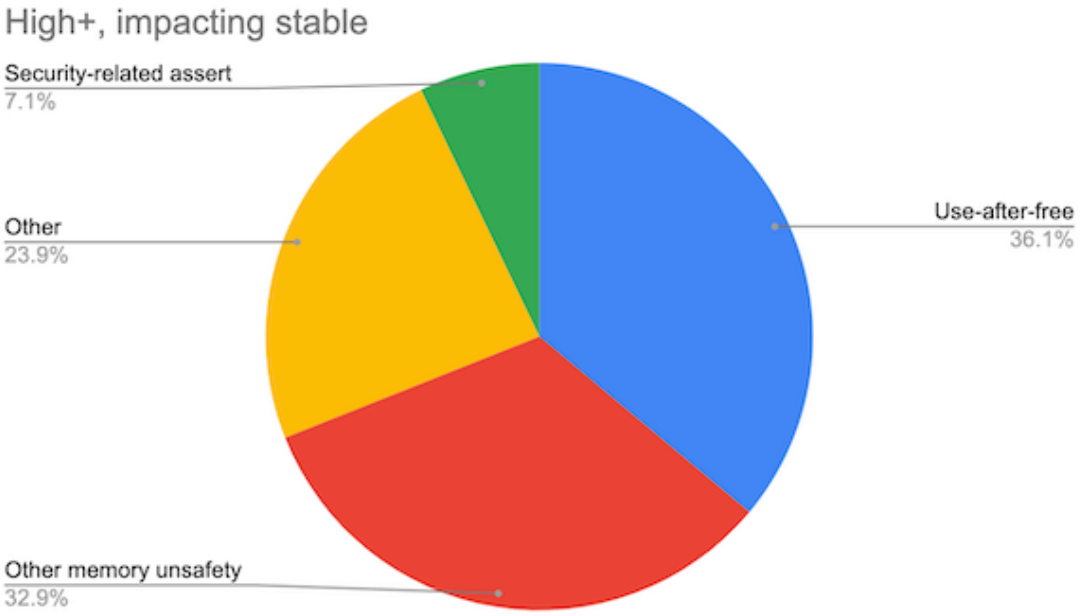
Because of this, personal preference may not be an ideal approach to building software. Looking into known popularity sources, like the TIOBE Programming Community index, shows the popularity of languages according to software engineers, courses, and other third party (TIOBE PROGRAMMING COMMUNITY INDEX. . . , 2020). With this data in mind and only this data, one programmer could be induced into thinking that the

[1] GitHub organization containing all repositories related to the project found at `https://github.com/Open-Digital-Twin`.

most popular language should be the most adequate when building a new software system. This analysis ignores the main purpose of having different programming languages, which is having features suited to the problems they are aimed at solving.

Working on software is not simple and bugs are unavoidable. Research by Google engineers of the Chromium Project (GOOGLE, 2020) found that 70% of their high severity security bugs are related to memory safety, through mistakes introduced with C/C++ pointers, leaving breaches for attacks by leaving access to what should be private and unreachable data. Figure 3.1 describes the types of memory safety bugs in the Chromium project. This number is identical to stats revealed by Microsoft, across their software catalog (CIMPANU, 2019).

Figure 3.1 – Chromium project fixed memory safety bugs




Source: (GOOGLE, 2020)

Both companies are dealing with the same problem in their software, due to vulnerabilities inserted by human error in their code. To deal with the same issue, Mozilla created Rust, an open-source language built from scratch to be focused on speed, parallelism, and memory safety, being less prone to memory exploits at an architectural level (MATSAKIS; KLOCK, 2014), enforcing safety without overhead caused by both runtime and garbage collection (BALASUBRAMANIAN *et al.*, 2017). The Chromium project and Microsoft, relating to this problem, also study the substitution of vulnerable parts of their software with Rust or are developing internal solutions that are Rust-inspired (GOOGLE, 2020; CIMPANU, 2019).

Given the points discussed above, to develop a modern system that is fast, but safe, all code relating to the Digital Twin architecture built in this work was written in Rust.

### 3.3 Third-party Digital Twin services

To fully implement the DT framework, third party open-source software was used in some services of the framework. One challenge in designing a real-time, temporal data-generating system is to decide the right storage engine for time-series data (DAMJANOVIC-BEHRENDT; BEHRENDT, 2019), which must be able to query and aggregate a large amount of sensor data, from multiple sources, distributed geographically. In regards to data storage, the platform uses the ScyllaDB database. Instead of a pure time-series database like InfluxDB, ScyllaDB is a highly-performant NoSQL general wide-column store solution, with a selectable replication factor, intended for low latency and high throughput to applications (SUNEJA, 2019; MAHGOUB *et al.*, 2017). This allows for our solution to have the benefits of fast querying in data with high support for data replication, necessary for the distributed DT scenario.

Another crucial element for a DT implementation is Data Acquisition Systems (DAS). For communication between the physical devices (*i.e.*, sensors, devices) and the DT instance, our framework uses the Message Queue Telemetry Transport (MQTT) protocol. For DTs, data acquisition overlap with the concept of telemetry: the collection of measurements and their automatic transmission for monitoring. The usage of MQTT for telemetry in IoT devices is an established and real-world tested concept, being used in Microsoft Azure IoT Hub (MICROSOFT, 2018), Amazon Web Services (AWS) IoT (BARR, 2015) and Facebook Messenger (ZHANG, 2011).

For the experiments executed, the platform uses the Eclipse Mosquitto message broker, version 1.6.10. This software was chosen due to its high popularity as an MQTT broker and for being an open-source implementation, built to be open and simple, intended for situations with a need for lightweight messaging, due to devices with limited resources (BANKS *et al.*, 2019; LIGHT, 2017). The MQTT broker works by ordering received messages in a queue and distributing it to its subscribers. To not reach the maximum message queue, which would discard all messages received on limit reached, the broker was configured to not have a limit, other than the physical memory restriction of where the broker was deployed.

### 3.4 Digital Twin model objects

For building the framework, certain objects were defined to model the Digital Twin. These objects allow users to represent the existing physical environment as digital models and extend them with additional elements.

- User: responsible for creating, updating and modifying their twin instances.
- Twin: central structure of the instanced DT. All modeled objects are associated with a twin instance, which represents the digitalized physical system.
- Element: a generic object used to define a part of the twin. An element could be a device, which contains sensors, or it could represent any other physical space, like buildings, or other real physical elements. Elements are directly associated with a twin and can be associated with other elements, to create more complex systems, all depending on the system modeled[2].
- Source: a generic component of an element. Used to define data entry points inside of elements of a twin instance. Figure 3.4 describes how communication for the twin instance and the real devices are dependent on the data sources.

### 3.5 Repository Organization

For hosting the project codebase and testing scripts, the "Open Digital Twin" organization was created on GitHub[3] and is freely accessible by any person with access to GitHub. The architecture is an open-source software built as a collection of individual microservices, available under the GNU General Public License v3.0. Currently, the organization hosts 11 repositories related to this project. In the following sections, the different repository types and individual repositories will be described and related.

[2]Currently, elements are used as the location where data sources are located. Future implementations could contain more useful information relating to the modeling of the actual physical element, for defining their entire life-cycle, and additional information used for physics simulations.

[3]GitHub organization containing all repositories related to the project found at `https://github.com/Open-Digital-Twin`

### 3.5.1 Architecture repositories

Repositories which names start with "dt" are implementations of the Digital Twin architecture. Figure 3.2 exposes the relationships between the repositories, showing how the digital elements share common modules.

- *dt-client* and *dt-client-bytes*: Testing repositories to simulate data acquisition via publishing to an MQTT broker, allowing for customization of the number of messages published, its frequency, and payload. They represent the "real" aspect of the system and act as elements with data sources [4];
- *dt-instance* and *dt-instance-webserver*: instance of the created Digital Twins. The *dt-instance* includes communication with the message broker and storage of acquired data, while *dt-instance-webserver* is an optional service containing a Representational State Transfer (REST) Application Programming Interface (API) for configuring the instance elements, data sources, and other configuration related to the DT instance[5]. The DT instance is the final destination for the client messages, after being distributed by the broker. They represent the "digital" aspect of the system;
- *dt-master*: Central Digital Twin microservice, where users can create and deploy new DT instances;
- *dt-common*: Shared modules used in multiple projects, to be used as a Git submodule, for reuse of DT-related code between the components;
- *dt-sharedconfig*: Shared configuration used in multiple projects, like database initialization.

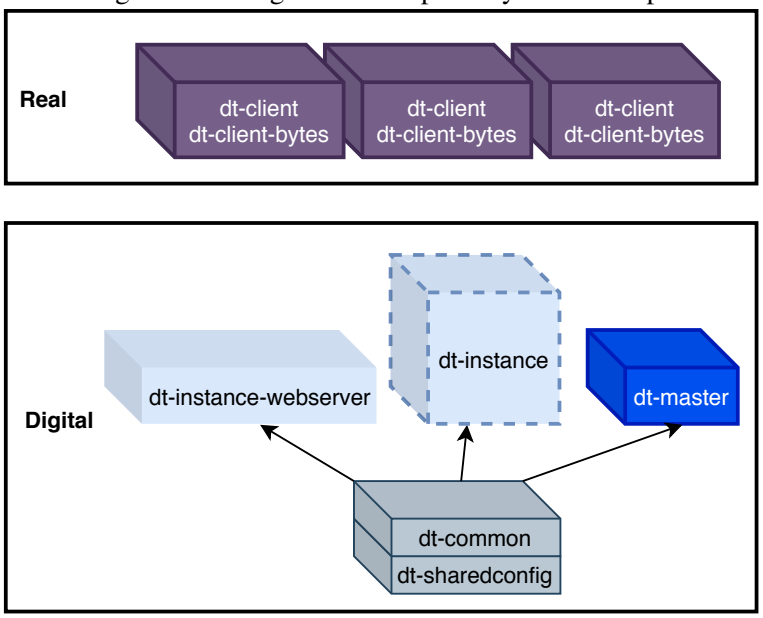
### 3.5.2 Article testing repositories

Repositories which names start with "article-test" are code used directly in testing for the case studies presented by this work. These include scripts for exporting and parsing raw data for analysis shown in each study.

[4]Since this was a simulation, the *"dt-client"* and *"dt-client-bytes"* repositories act as the devices tested, communicating with the DT instance using the *rumqttc* MQTT client, found at `https://github.com/bytebeamio/rumqtt`.

[5]The DT instance is a prototype of the digital aspect of the system, providing means for acquisition, storage, and processing of data, and is where the physical product is linked throughout its entire lifecycle (GRIEVES; VICKERS, 2017).

Figure 3.2 – Organization repository relationships




Source: Image provided by author

- *article-test-fog*: Scripts used for testing Case Study #1 - Fog Computing, in the cloud and fog deployment scenarios of the architecture.
- *article-test-mqtt-sender* and *article-test-mqtt-receiver*: Scripts used for testing Case Study #2 - MQTT Analysis, which studies the usage of the MQTT protocol for Digital Twins.

### 3.5.3 General repositories

Repositories that do not have a specific category, but are relevant to the project and the organization.

- *Summary*: Meta repository, for documenting the organization. Repository for documenting the different repositories associated with the Open Digital Twin group projects.

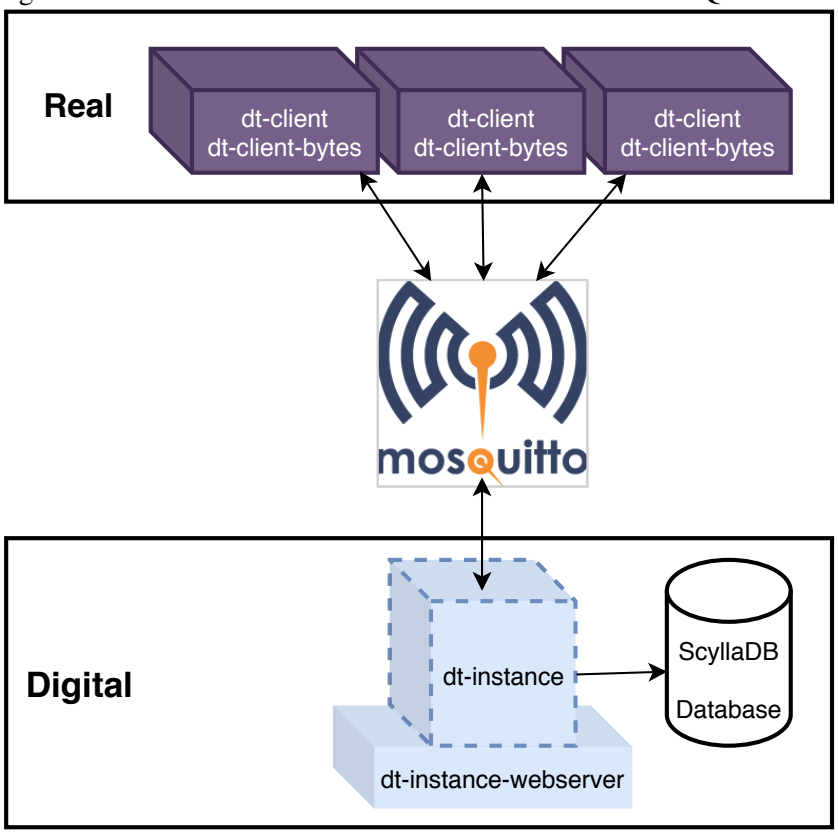
## 3.6 Architecture usage

To perform the experiments, the client devices were simulated with a custom program that sends sequentially numbered messages, the "*dt-client-bytes*", with a configurable payload and publication interval, to the broker, with a specific topic known by the

subscribed twin instance. Numbered messages allow us to analyze the timing aspects of each message, from its publication by the client until it is processed by the twin instance.

Figure 3.3 presents an illustration of how the communication between the "real" physical devices and the Digital Twin communicate. After deploying the instance web server, the user can model its twin with as many elements required, which in turn can have multiple data sources. The REST API will return the exact topic where the twin will subscribe and the *dt-instance* and client services must be configured to communicate using the same MQTT broker [6].

Figure 3.3 – Devices to Twin instance communication via MQTT broker.

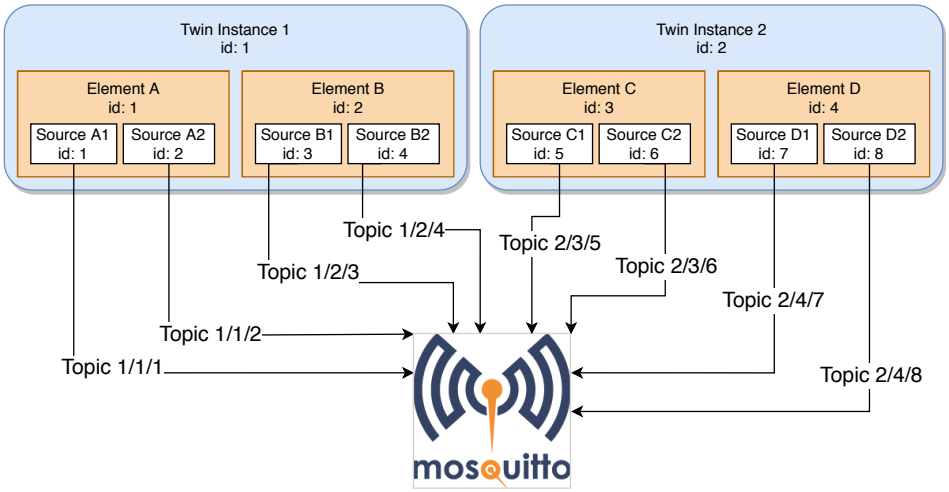



Source: Image provided by author

Figure 3.4 illustrates the relations between each of the twin model objects in constructing communication. Each data source will have a unique topic, dependent on how the twin is modeled, in the format of "twin id/element id/source id". Real devices can then publish messages to the broker in this topic, to insert data to these sources, and the twin instance will subscribe to these topics awaiting new data.

[6]For the effects of our experiments, this is done by configuring an environment variable in each service and manually deploying. In a more advanced scenario, like the framework being used as a Software as a Service (SaaS) platform, all this configuration would be seamless to the users, who would use the *dt-master* service to create and deploy their twins and only worry about publishing MQTT messages to an address, returned by the API.

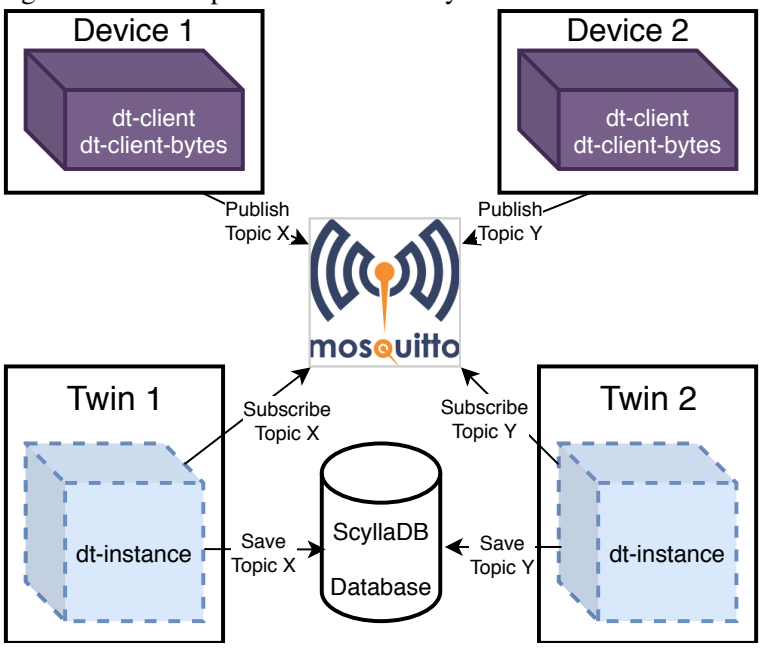
Figure 3.4 – Twin model objects to MQTT publish/subscribe topics.




Source: Image provided by author

Figure 3.5 illustrates the flow of messages from multiple devices. In this scenario, each device represents an element with a single data source. To communicate data to the twin instance, the device publishes messages to the specific topic of that source to the MQTT broker. Previous to publishing, each twin instance is subscribed to all the registered data sources for their twin. When a message is published to the subscribed topic, the instance stores the transmitted data into its associated database.

Figure 3.5 – Multiple Twin instances system communication flow.




Source: Image provided by author

# 4 EXPERIMENTS

This chapter will present two different studies implemented with the built Digital Twin framework, explaining each experiment's motivation and scenario, factors, and discussing the results obtained.

## 4.1 Case Study #1 - Fog Computing

To match their physical counterparts, DTs acquire data through a large number of sensors. The rapid expansion of IoT-powered sensors is what makes DTs possible, but with increased data generation, centralized applications can experience high latency and poor connectivity, affecting application performance. Cloud services can be used to reduce these problems, but moving all data to the cloud may not satisfy time-sensitive application requirements. To further improve and minimize this situation, the concept of fog computing enables computing, storage, networking, and data management on network nodes closer to edge devices, allowing for processing to exist not only in the cloud but between these two layers (YOUSEFPOUR *et al.*, 2019).

Several areas can be improved with the usage of fog computing, including smart industries (KHAN *et al.*, 2020; WAN *et al.*, 2018), homes (VERMA; SOOD, 2018), cities (TANG *et al.*, 2015; PERERA *et al.*, 2017) and healthcare (NIKOLOUDAKIS *et al.*, 2017), real-time systems with time-critical processing to avoid accidents and to provide a better and more reliable experience without suffering from cloud latency. Integrating fog computing with IoT and DTs enables mirroring physical and virtual spaces while achieving the necessary timing requirements of real-time systems.

Balancing the computational workload between multiple fog nodes, closer to the edge, enables the network to be more efficient in handling timing critical tasks, by distributing operations between different devices and reducing communications between edge and cloud. Research is still required so that several connected services and devices can scale and correctly execute real-time operations in a timely fashion. It is still challenging to manage, process, and store all this data in a highly distributed environment (DIZDAREVIĆ *et al.*, 2019; KHAN *et al.*, 2020).

The literature review revealed that the use of cloud and fog computing to build DTs and CPS is not new, but they are presented mainly as abstract, non-experimented scenarios. This case study analyzes the usage of a cloud-fog architecture that can manage

a large scale number of real-time Digital Twins, increasing performance while reducing latency, to meet timing requirements. Thus, the contribution of this study is to see how the Digital Twin framework deployed in a fog computing layer can help applications that must meet real-time requirements. To meet real-time requirements, the Digital Twin architecture microservices, being virtual systems, can be widely distributed between cloud, fog, and edge computing layers, depending on the scenario of the corresponding physical system and the available network resources. This study presents the idea of using both cloud and fog layers, but excluding edge, due to its possible short and temporary availability, using a more controlled layer, in this case, the fog, enables the objective of latency reduction while still being supervisable. This work focuses solely on DT communication. Analysis and prediction are left as future work, in a full cycle communication scenario, with data acquisition, simulations generating decisions, and suggesting an action to the real physical system. The full study can be found in Appendix A.

### 4.1.1 Experiment Setup

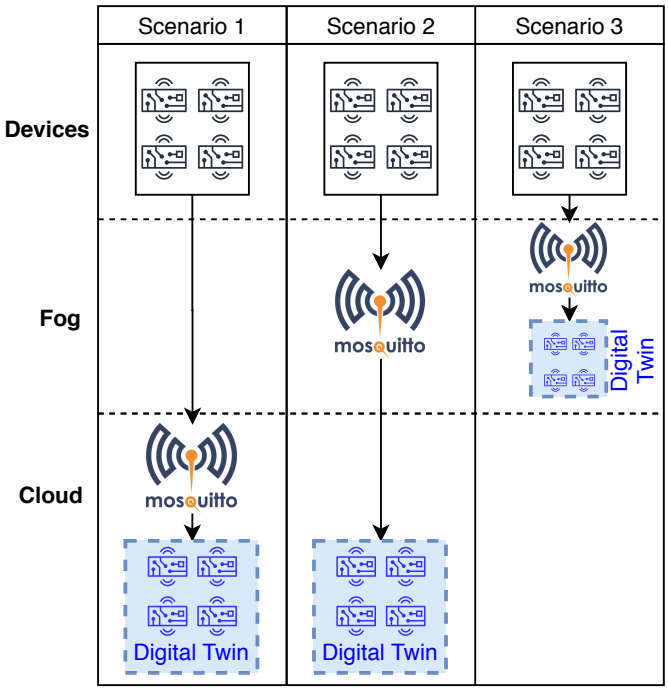
The first step, before defining the deployment scenarios, was to list the experiment factors, variables that will be used or measured in the tests.

- *Number of data sources:* twin elements can have multiple data sources, in the form of multiple individual devices, which by themselves can have multiple sensors. For simplification of the problem in this paper, an element is defined as one general part or device of the physical system, which has only one data source. To emulate multiple data sources, it is possible to increase the number of elements accordingly. Each source then sends a certain amount of messages to the broker.
- *Message payload size:* number of bytes sent in the MQTT message payload generated by "*dt-client-bytes*", which will affect bandwidth usage and processing time for the message broker and the twin instance.
- *Message frequency:* the estimated time between messages sent by the data sources.
- *Transmission latency:* time for the message to travel in two steps: 1) from the client to the broker; and 2) from the broker to the twin instance. This latency is directly affected by the distribution of the architecture between cloud and fog.

To perform the study on cloud and fog, three different experiment scenarios are proposed. These three scenarios are illustrated in Figure 4.1 and detailed as follows:

- *Scenario 1 - Cloud only:* Processing is done exclusively on a single cloud node. The MQTT broker, where the client publishes and the twin instance subscribes to a data source, resides on this cloud node.
- *Scenario 2 - Fog and Cloud:* The MQTT broker is located on a fog node, closer to the edge, while the twin instance resides on the cloud.
- *Scenario 3 - Fog only:* Both broker and the twin instance resides on a fog node.

Figure 4.1 – Case Study #1 - Cloud and fog deployment scenarios.




Source: Image provided by author

Each scenario was individually tested under three situations: (A) varying the number of messages sent by each data source; (B) varying the frequency of messages transmissions; and (C) varying the payload of the sent messages.

- *Situation A - Source and number of message:* Situation A varied the amount of messages sent by each source, in sets of 100, 1,000, 10,000, and 100,000 messages by each source. To test concurrency, the number of data sources was set to 1, 3, and 5. The delay between when each message was sent by the data sources was fixed to 10ms, with a payload of 64 bytes.
- *Situation B - Message frequency:* Situation B tested the effects of the frequency messages sent by the data source, sending messages after 10, 20, 40, 80, 160, 320, 640, 1280, and 2560 milliseconds. In this situation, a total of 1000 messages were sent by only one source, each with a payload of 64 bytes.

- *Situation C - Message payload size:* Situation C tested different payload sizes on the messages. The payloads were 8, 16, 32, 64, 128, 256, and 512 bytes. Each test sent a total of 1000 messages from one source, with a fixed delay of 80ms between each message.

For running these experiment scenarios, a server running an Intel Xeon E5-2420 (with 6 cores and 12 threads) and 32 GB of RAM was used. The experiments were executed on three individual Kernel-based Virtual Machines simulating real devices, each named according to their use on the defined scenarios: Cloud, Fog, and Client. The Cloud virtual machine runs with two virtual Central Processing Units (CPU) and 4 gigabytes (GB) of memory. The Fog and Client are by definition devices with more constrained resource capacities, with only one virtual CPU each and 1 GB of memory.

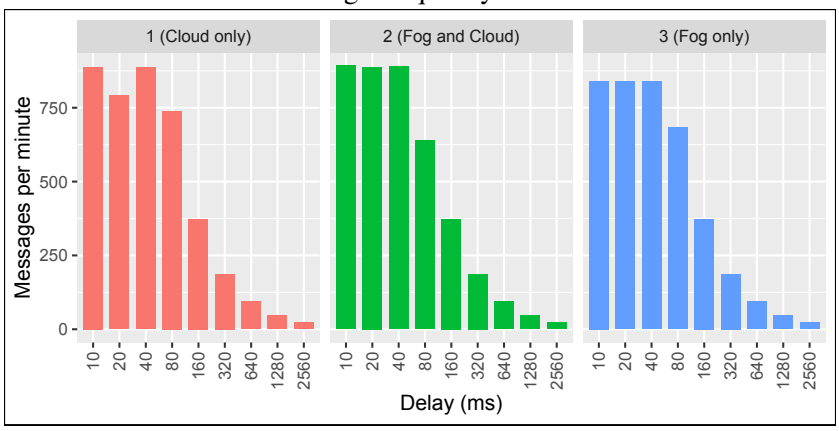
In regards to network limitations, network latency was injected in the communications between the machines, with the usage of *tc-netem* (HEMMINGER; LUDOVICI; PFEIFER, 2011). In scenario 1, a delay of 100ms, with a 10ms variation, was added in the Client. In scenarios 2 and 3, two different delays were added: 40ms, varying 10ms, in the Client, and 20ms, varying 5ms, in the Fog. These delays were inserted to simulate existing latency between fog and cloud components, with fog having lower hardware resources but lower latency.

### 4.1.2 Results

As expected by definition, bringing processing closer to the edge, to handle our message exchange, effectively reduced latency for the system. DTs, composed of a large number of sensors, could experience high latency and poor connectivity. Proposing processing being done closer to the edge is a possible strategy, since being purely dependent on the cloud may not satisfy time-sensitive requirements (YOUSEFPOUR *et al.*, 2019). The processing unit can receive the data from the source devices and compute its suggestion to the physical system with a shorter response time.

When testing situation B, varying the message frequency, an important detail about how the MQTT broker throttles messages were found. As illustrated by Figure 4.2, which shows the number of messages receives per minute in each scenario, varying the message frequency, the broker has a limit on the number of messages it can receive and publish to its subscribers in the order of 800 messages per minute. This number was reached even

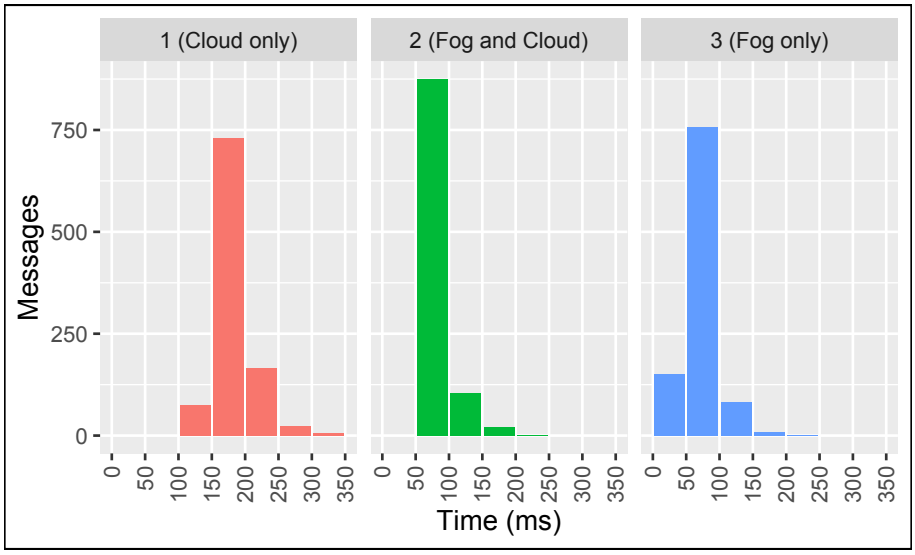
Figure 4.2 – Case Study #1 - Messages per minute received by the DT instance, per scenario, in the message frequency situation.




Source: Image provided by author

though the broker is set to have no limits on the message queue. This message rate is a hard limit set by the broker to prioritize receiving publications and to not overload client subscribers, via limiting the number of in-flight messages (messages queue to be sent from the broker to its clients). Eclipse Mosquitto allows for configuration of this number, but doing so would cause message order to break or overloading subscribers, and should be tested depending on the deployment scenario of the real devices.

Figure 4.3 – Case Study #1 - Timing histogram of 1000 messages, from client to DT, in each scenario.

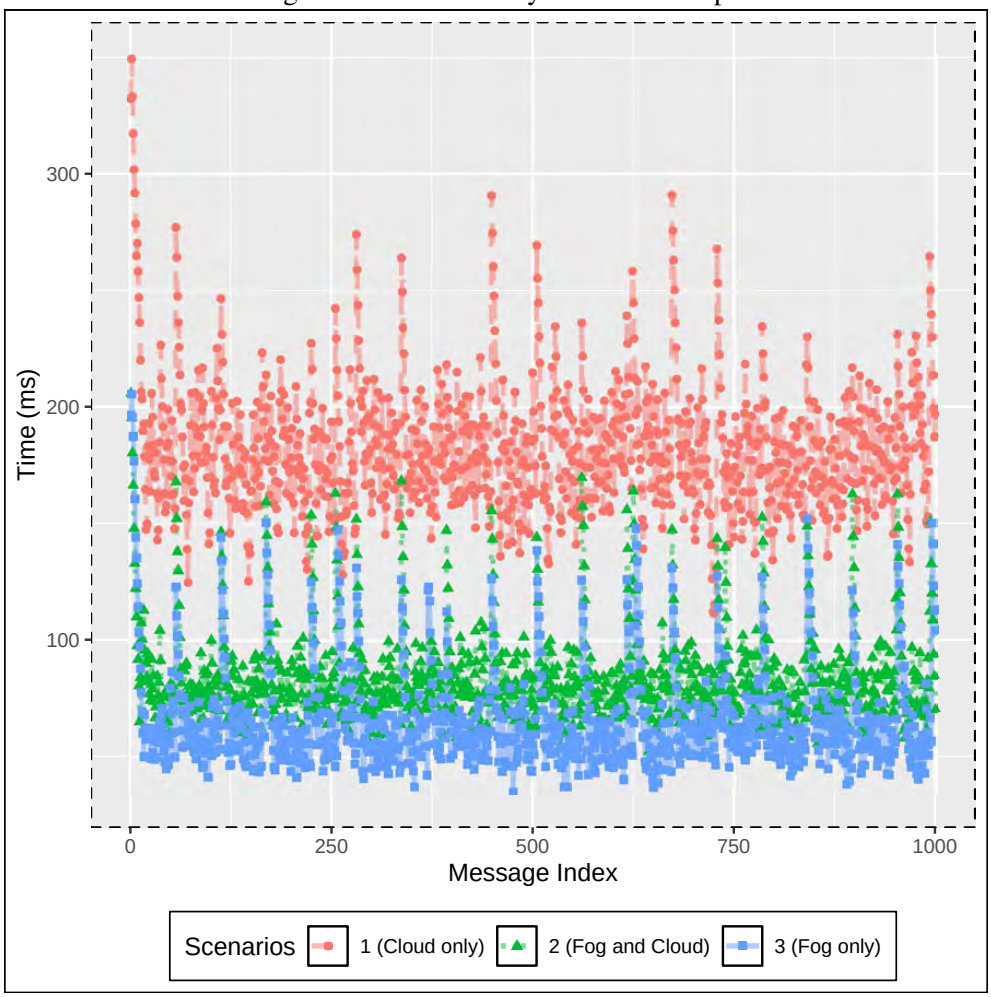



Source: Image provided by author

Figure 4.3 presents a timing histogram of 1000 messages in the three scenarios, on situation C with a payload of 64 bytes and 80ms interval between each message, from the time it is published by the client until received by the DT instance (subscriber). Results from this experiment show the time of each publication. Scenario 1 presents messages

with a higher time compared to the two other scenarios, where most messages are received in the 50 to 100 ms interval, in this experiment. Considering a real-time system with a communication requirement of 200ms in the Cloud only scenario (1), 805 of the 1000 messages would respect this limitation, representing 80.5% of the total messages sent by the client. Meanwhile, only one message would not arrive in time for this imposed system limitation in the Fog and Cloud (2) and Fog only (3) scenarios, respectively. Further reducing this limitation to 150ms, 92.4% of messages sent in scenario 1 would be lost, while 0.21% and 0.09% would not respect the requirement by scenarios 2 and 3.

Figure 4.4 – Case Study #1 - Timing per message, from client to DT, in each scenario, sending 1000 messages with a 80ms delay between each publication




Source: Image provided by author

More benefits depend on the architecture scale of implementation in the fog and cloud layers. The usage of cloud computing enables devices to store data in a centralized location, but large amounts of data being transmitted, natural to the concept of Digital Twins with a large scale network of sensors, implies an increasing delay, affecting response times from the server. By having resources closer to the edge, it is possible to

preprocess data as soon as possible, before centralizing the data in the cloud.

Figure 4.4 makes this clear by showing the time each message takes from being sent by the client until received by the twin instance in each deployment scenario. The situation shown in this image sends 1000 messages, each one with a delay between each publication of 80ms. Table 4.1 provides a summation of the average times of this experiment, providing a general overview of all the published messages. Using the Fog and Cloud scenario (2) and the Fog only scenario (3) reduced average times per message by 54% and 64% respectively, compared to the Cloud only scenario (1). Maximum times were also reduced by 41% for scenarios 2 and 3, as were the minimum times by 53% and 69%.

Table 4.1 – Case Study #1: Average times per message, from client to twin, for the 80ms delay experiment situation.

| Time per message (ms) | Average | Std Deviation | Max | Min |
|---|---|---|---|---|
| Scenario 1 | 182.45 | 27.932 | 349.35 | 111.82 |
| Scenario 2 | 84.43 | 20.417 | 205.83 | 52.51 |
| Scenario 3 | 66.08 | 22.626 | 205.63 | 35.13 |

Source: Table provided by author

## 4.2 Case Study #2 - MQTT Analysis

For the second case study of the Digital Twin framework, further studies were made in the context of determining the limitations and overall usability of MQTT for Digital Twins. This study resulted in a published paper, written in Portuguese, which can be found in Appendix B.

For DTs to function, one matter that requires definition is the necessity of communication between the DT instance and the multitude of IoT-enabled sensors that input data. MQTT, being a lightweight and economic protocol for usage in low capacity devices (BANKS *et al.*, 2019), is a possible alternative. Depending on a single element in the architecture, the MQTT broker, to mediate all communication between clients could represent a system bottleneck, being directly reliant on its performance, and being a single point of failure. A study on how the broker behaves is required to assess its viability for real-time, large scale, DT-based systems, to obtain the most efficient manner on how to distribute multiple broker instances and reach higher system scalability.

### 4.2.1 Experiment Setup

This study consisted of an analysis of the MQTT broker performance for scenarios where the volume of messages was higher than it could process. This situation is recreated in several different contexts to determine which configuration is the most optimized for the DT instance. To test these scenarios, two framework elements were used, other than the MQTT broker: the *dt-client-bytes* and the *dt-instance*.

A server running an Intel Xeon E5-2420 (with 6 cores and 12 threads) and 32 GB of RAM was used as the experiment environment. The experiments were executed on three different types of Kernel-based Virtual Machines simulating real devices: Type 1 machines, with 2 GB of memory and 1 virtual CPU, Type 2 machines, with 4 GB of memory and 2 virtual CPUs, and Type 3 machines, with 8 GB of memory and 4 virtual CPUs.

Considering this environment, the first conducted experiment published 1000 MQTT messages with a fixed payload of 64 bytes, from the client to the DT instance, at decreasing intervals of time, to determine the message output capacity of a single broker. This was analyzed with the number of queued messages of the broker in different moments, during its execution.

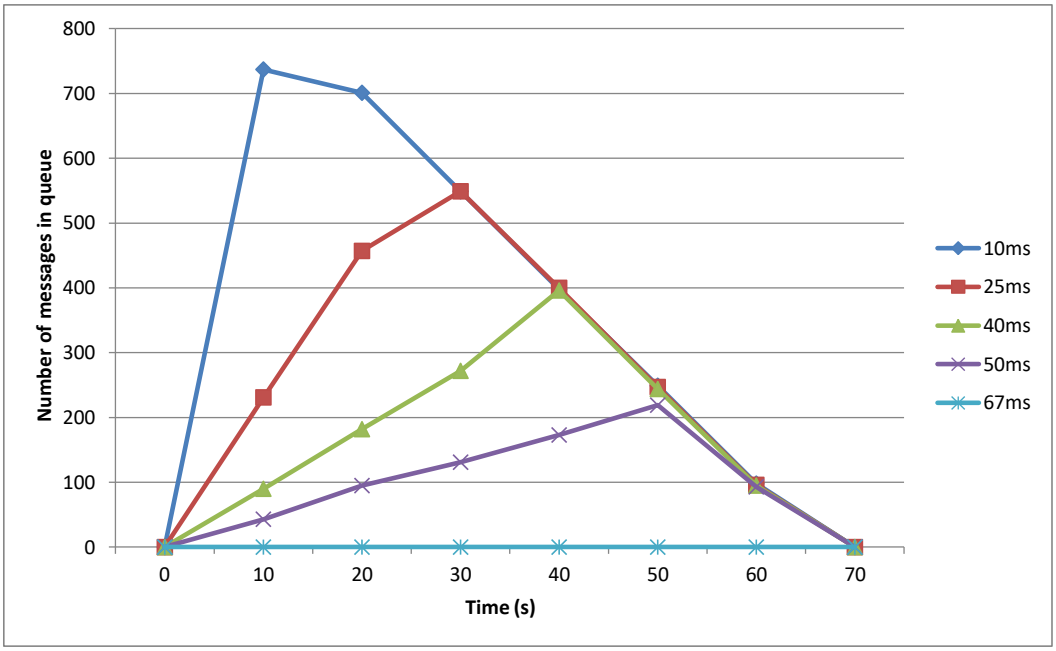
The solution to remove the limitation on the output flow of messages from the broker is to distribute its message load to other brokers. Eclipse Mosquitto is not a multi-threaded broker, but its lightweight nature allows for instantiating several parallel brokers in the same device. The second experiment assessed the performance difference of sending the same number of messages when distributed between multiple brokers, in two different scenarios: Centralized, where all the brokers run in the same virtual machine with higher computing resources and Distributed, where brokers are allocated to different, lower capacity, virtual machines.

### 4.2.2 Results

For the first experiment, the goal was to determine the maximum rate of messages that could be sent by only one broker. In this experiment, 1000 messages were sent from the client device to the Mosquitto Broker in gradually smaller intervals. With the analysis of the number of messages in the queue at the broker, the maximum output flow of messages could be determined. These tests were executed in a type 3 machine, with all

services centralized.

Figure 4.5 – Case Study #2 - Experiment 1 queue size on broker, with different intervals between messages sent by client.




Source: Image provided by author

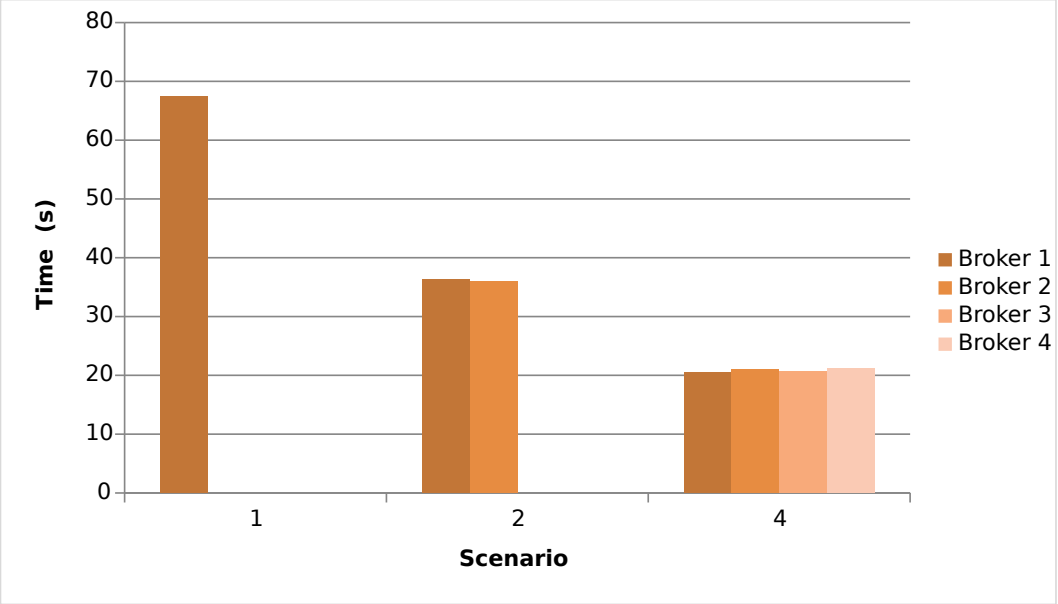
Assessing the described behavior of Figure 4.5, we can see that the total time it took the broker to effectively output all 1000 received messages was in the range of 67 seconds. Smaller intervals than 67 ms, shown in the figure, will generate a queue inside the broker, which results in having no benefit in publishing messages with a smaller time frame than that. With these values obtained, we can determine that Eclipse Mosquitto, with its default settings, can at most handle publishing of 15 messages per second, in these conditions. For intervals above that of 67 ms, there is no queue generation, since its output rate is higher than its input.

For the second experiment, the messages were sent with a fixed 10 ms interval, which would generate a large queue, but we wanted to determine if this message load, when distributed to multiple brokers, could reduce the total time to send all messages, via a higher output frequency. For the centralized scenario, machines of type 2 were used in tests with 2 brokers and a machine of type 3 for the tests with 4 brokers. In the distributed scenario, each broker was deployed using a machine of type 1.

In the centralized scenario, with 1, 2, or 4 brokers running on the same machine, the performance of each broker was lowered, but the total time for all messages to be processed was reduced. Figure 4.6 displays the values obtained for this experiment scenario,

showing the total time to send all messages.

Figure 4.6 – Case Study #2 - Experiment 2 Centralized total time spent per broker scenario.




Source: Image provided by author

Through this data, we can see the advantages of clustering our broker, but also that the sum time of the clustered brokers increases with the number of brokers deployed. This information infers that this tactic will not reduce broker times forever and must be tested according to the deployment scenario of the specific DT, taking into account network requirements and hardware resources available. Table 4.2 breaks down the experiment, showing per broker scenario the total time, the sum time to send (sum of the individual times each broker spent to finish), the frequency per broker (the average messages per second sent by each broker), and the broker cluster frequency (the average messages per second sent by all brokers combined).

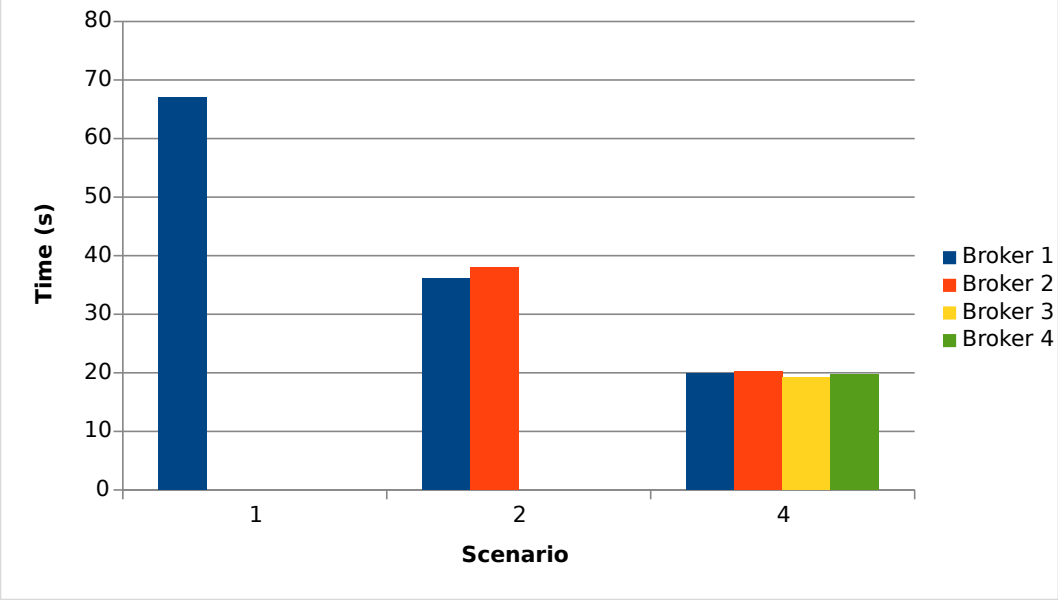
Table 4.2 – Case Study #2 - Experiment 2 Centralized broker times.

| Brokers | Total time (s) | Sum time (s) | Frequency per broker (message / s) | Broker cluster frequency (message / s) |
|---|---|---|---|---|
| 1 | 67.34 | 67.34 | 14.85 | 14.85 |
| 2 | 36.125 | 72.25 | 13.84 | 27.68 |
| 4 | 20.79 | 83.19 | 12.02 | 48.08 |

Source: Table provided by author

For the distributed part of experiment 2, the scenario changes to use different machines per broker but using lower capacity machines. Figure 4.7 displays the total time the total messages sent took, in each deployment with a different number of brokers.

Figure 4.7 – Case Study #2 - Experiment 2 Distributed total time spent per broker scenario.




Source: Image provided by author

Table 4.3 – Case Study #2 - Experiment 2 Distributed broker times.

| Brokers | Total time (s) | Sum time (s) | Frequency per broker (message / s) | Broker cluster frequency (message / s) |
|---|---|---|---|---|
| 1 | 67.34 | 67.34 | 14.85 | 14.85 |
| 2 | 37.03 | 74.07 | 13.50 | 27.00 |
| 4 | 19.715 | 78.86 | 12.68 | 50.72 |

Source: Table provided by author

Comparing tables 4.2 and 4.3, we can see that the results from both deployments are quite similar in some aspects. Both share improved total times and similar message frequency increases. All values are shown where the results of the mean values are found in 5 executions of each experiment.

Through the results acquired in this case study, we have discovered there is a frequency limitation on the output of messages from the Mosquitto MQTT broker, achieving the maximum value of 15 messages per second per broker. To reduce the effects of this limitation, we deployed multiple parallel brokers in two different scenarios (centralized and distributed). Both scenarios similarly allowed for increased frequency and further studies are required to obtain an ideal distribution of brokers for all cases. The experiments did not factor in outside factors that could impact transmissions, like network latency, jitter, or packet losses. Future experiments could take these factors into account to more accurately determine the overall behavior of the broker and its consequences to a more realistic DT deployment.

# 5 CONCLUSION

This work investigated and discussed what defines Digital Twins, researching the terminology, and the state of the art in the research area. One major problem found was the absence of public, open implementations, being limited to abstract conceptual proposals, leading to the suggestion of building a microservices cloud-based Digital Twin framework. For software to scale in the manner expected for DTs, deployment in the cloud is suggested, but with the increased data volume, moving all data to the cloud may not satisfy time-sensitive application requirements, so the usage of a cloud-fog architecture is suggested, to reduce overall latency. After defining the importance of DTs and their building blocks, a proposal is presented, defining the elements required to create a functioning DT instance in the cloud-fog environment.

Case study #1 investigated how the usage of a cloud-fog architecture for Digital Twins can help in meeting the real-time requirements of these systems. Tests were executed in three different scenarios, changing the deployment locations of the architectural elements where the broker and twin reside in the cloud, where the broker resides in a fog node, and the twin on the cloud, and lastly where both the broker and twin share the same fog node. The experiments tested these different scenarios on how they would handle the number of messages, different frequency of messages sent, and payload size. From these experiments, fog computing is shown as a viable alternative to reduce the effects of this problem, reducing the transmission delay to meet real-time requirements. It was concluded that the distribution of these DT elements closer to the edge reduces communication delays for the end application, allowing faster response by bringing preprocessing into distributed locations. This particular distribution allows the DT to have a best-effort attempt at meeting real-time systems timing specifications. Real-time systems are not necessarily systems with short deadlines. Non-critical data, like physical system telemetry or historic data, can be transmitted without timing deadlines. Only critical, alert, or error-specific messages need priority in communication.

Case study #2 analyzed the impact of using MQTT and Eclipse Mosquitto as the DT framework communication enabler. Several experiments were presented, reaching even more specific values of message limitations, testing the performance limits of the broker. With limits reached, we deployed the broker in several different ways, both in a centralized and distributed manner, allowing for more messages to reach the DT instance without overloading the message broker.

This project is still a work in progress. A Digital Twin platform must be extensible and collaborative, implementing more solutions to meet real scenarios. The first step in creating the framework, which is modeling, providing data acquisition and storage, is complete, but there is a multitude of other components required for fully implementing a Digital Twin framework, for instance enabling feedback from the twin to its client devices and to provide good visual feedback of the created twin for the users involved in the process.

Having all the required data flow from the client devices, the DT can run application-specific algorithms handling this input and be able to suggest actions to the physical systems respecting their real-time requirements. Some limitations could be addressed, as the problem regarding the real-time priority policy for the MQTT broker: message handling should prioritize real-time subscribers. The MQTT protocol has no definition of priority for subscriptions, but a broker-defined configuration indicating priority topics could be used, dedicating more resources to attempt a best-effort approach to meeting real-time requirements. External factors in regards to network communication could be analyzed to check their impact on real-life, non-controlled scenarios.

## REFERENCES

AHMAD, Sultan; AFZAL, Mohammad Mazhar. Deployment of Fog and Edge Computing in IoT for Cyber-Physical Infrastructures in the 5G Era. *In*. INTERNATIONAL Conference on Sustainable Communication Networks and Application. Erode, India: Springer, 2019. p. 351–359.

ASHTON, Kevin *et al.* That 'internet of things' thing. **RFID journal**, v. 22, n. 7, p. 97–114, 2009.

ATLAM, Hany F; WALTERS, Robert J; WILLS, Gary B. Fog Computing and the Internet of Things: A Review. **Big Data and Cognitive Computing**, Multidisciplinary Digital Publishing Institute, v. 2, n. 2, p. 10, 2018.

BALASUBRAMANIAN, Abhiram *et al.* System Programming in Rust: Beyond Safety. *In*. PROCEEDINGS of the 16th Workshop on Hot Topics in Operating Systems. Whistler, BC, Canada: Association for Computing Machinery, 2017. (HotOS '17), p. 156–161.

BANKS, Andrew *et al.* **MQTT Version 5.0**. Mar. 2019. Available from: `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html`. Visited on: 9 Aug. 2020.

BARR, Jeff. **AWS IoT – Cloud Services for Connected Devices**. 2015. Available from: `https://aws.amazon.com/blogs/aws/aws-iot-cloud-services-for-connected-devices/`. Visited on: 30 Oct. 2020.

BOYES, Hugh *et al.* The industrial internet of things (IIoT): An analysis framework. **Computers in Industry**.

CAMERON, David; WAALER, Arild; KUMULAINER, Tiina. Oil and Gas digital twins after twenty years. How can they be made sustainable, maintainable and useful? *In*. 59TH CONFERENCE ON SIMULATION AND MODELLING. Oslo, Norway: Linköping University Electronic Press, 2018.

CARDIN, Olivier. Classification of cyber-physical production systems applications: Proposition of an analysis framework. **Computers in Industry**, Elsevier, v. 104, p. 11–21, 2019.

CHEN, Yishan *et al.* Deploying data-intensive applications with multiple services components on edge. **Mobile Networks and Applications**, Springer, p. 1–16, 2019.

CIMPANU, Catalin. **Microsoft: 70 percent of all security bugs are memory safety issues**. 2019. Available from:

https://web.archive.org/web/20201031031718/https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/. Visited on: 30 Oct. 2020.

DAMJANOVIC-BEHRENDT, Violeta. A Digital Twin Architecture for Security, Privacy and Safety. **ERCIM News**, v. 115, p. 25–26, 2018.

DAMJANOVIC-BEHRENDT, Violeta; BEHRENDT, Wernher. An open source approach to the design and implementation of Digital Twins for Smart Manufacturing. **International Journal of Computer Integrated Manufacturing**, Taylor & Francis, v. 32, n. 4-5, p. 366–384, 2019.

DENG, Jing *et al.* Fault-tolerant and reliable computation in cloud computing. *In*. 2010 IEEE Globecom Workshops. Miami, FL, USA: IEEE, 2010. p. 1601–1605.

DIZDAREVIĆ, Jasenka *et al.* A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. **ACM Computing Surveys**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 6, 2019.

FULLER, A. *et al.* Digital Twin: Enabling Technologies, Challenges and Open Research. **IEEE Access**, v. 8, p. 108952–108971, 2020.

GOOGLE. **Memory Safety**. 2020. Available from: https://web.archive.org/web/20201031025218/https://www.chromium.org/Home/chromium-security/memory-safety. Visited on: 30 Oct. 2020.

GRIEVES, Michael. **Origins of the Digital Twin Concept**. [*S. l.*], 2016.

GRIEVES, Michael; VICKERS, John. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. *In*. **Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches**. Ed. by Franz-Josef Kahlen, Shannon Flumerfelt and Anabela Alves. [*S. l.*]: Springer International Publishing, 2017. p. 85–113.

GRIFFOR, Edward R *et al.* Framework for cyber-physical systems: Volume 1, overview. **Report No. 1500-201**, Gaithersburg, MD, 2017. DOI: 10.6028/NIST.SP.1500-201.

HE, Y.; GUO, J.; ZHENG, X. From Surveillance to Digital Twin: Challenges and Recent Advances of Signal Processing for Industrial Internet of Things. **IEEE Signal Processing Magazine**, v. 35, n. 5, p. 120–129, 2018.

HEMMINGER, Stephen; LUDOVICI, Fabio; PFEIFER, Hagen Paul. **NetEm - Network Emulator**. Nov. 2011. Available from: https://man7.org/linux/man-pages/man8/tc-netem.8.html. Visited on: 22 Aug. 2020.

IRVING, Duncan. Causing E&P problems with Digitalisation. *In* JEFFERY, Karl (Ed.). **Solving E&P problems with digitalisation**. London, UK: Digital Energy Journal, Nov. 2018.

JOSEPH, Christina Terese; CHANDRASEKARAN, K. Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. **Software: Practice and ExperienKce**, Wiley Online Library, v. 49, n. 10, p. 1448–1484, 2019.

KHAN, WZ *et al.* Industrial internet of things: Recent advances, enabling technologies and open challenges. **Computers & Electrical Engineering**, Elsevier, v. 81, p. 106522, 2020.

KIELAR, Tomasz. **Digital Twin Explained – The Next Thing After IoT**. 2019. Available from: https://www.pgs-soft.com/blog/digital-twin-explained-the-next-thing-after-iot. Visited on: 27 Nov. 2019.

KIM, Taioun. Cyber Physical Systems Framework of Edge-, Fog-, and Cloud-Computing. *In*. PROCEEDINGS of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications. Las Vegas, NV, USA: 2019. p. 10–14.

KOPETZ, Hermann. **Real-time systems: design principles for distributed embedded applications**. [*S. l.*]: Springer Science & Business Media, 2011.

KORTH, Benjamin; ZAJAC, Markus; SCHWEDE, Christian. Simulation-ready digital twin for realtime management of logistics systems. *In*. 2018 IEEE INTERNATIONAL CONFERENCE ON BIG DATA. Seattle, WA, USA: IEEE, 2018.

LIGHT, Roger A. Mosquitto: server and client implementation of the MQTT protocol. **Journal of Open Source Software**, The Open Journal, v. 2, n. 13, p. 265, 2017. Available from: https://doi.org/10.21105/joss.00265.

LIU, Ying *et al.* A Novel Cloud-Based Framework for the Elderly Healthcare Services Using Digital Twin. **IEEE Access**, IEEE, v. 7, p. 49088–49101, 2019.

LUCERO, Sam. **IoT platforms: enabling the Internet of Things**. Phoenix, AZ, USA, 2016.

MAHGOUB, A. *et al.* Suitability of NoSQL systems — Cassandra and ScyllaDB — For IoT workloads. *In*. 2017 9TH INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEMS AND NETWORKS. Bengaluru, India: Springer, 2017. p. 476–479. DOI: 10.1109/COMSNETS.2017.7945437.

MARINESCU, Dan C. **Cloud computing: theory and practice**. [*S. l.*]: Morgan Kaufmann, 2017.

MATSAKIS, Nicholas D.; KLOCK, Felix S. The Rust Language. *In*. PROCEEDINGS of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology. Portland, Oregon, USA: Association for Computing Machinery, 2014. p. 103–104.

MELL, Peter; GRANCE, Tim, *et al.* The NIST definition of cloud computing. **National Institute of Science and Technology, Special Publication, 800**, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, v. 145, 2011.

MICROSOFT. **Azure Digital Twins**. 2020. Available from: https://azure.microsoft.com/en-us/services/digital-twins/. Visited on: 27 Oct. 2020.

MICROSOFT. **Azure Digital Twins (ADT) explorer**. 2020. Available from: https://github.com/Azure-Samples/digital-twins-explorer. Visited on: 27 Oct. 2020.

MICROSOFT. **Communicate with your IoT hub using the MQTT protocol**. 2018. Available from: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support. Visited on: 30 Oct. 2020.

MOHAN, Nitinder; KANGASHARJU, Jussi. Edge-Fog cloud: A distributed cloud for Internet of Things computations. *In*. 2016 Cloudification of the Internet of Things (CIoT). Paris, France: IEEE, 2016. p. 1–6.

NIKOLOUDAKIS, Yannis *et al.* An NF V-powered emergency system for smart enhanced living environments. *In*. 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks. Berlin, Germany: IEEE, 2017. p. 258–263.

PARRY, Peter. Next generation digital organization and capabilities. *In* JEFFERY, Karl (Ed.). **Solving E&P problems with digitalisation**. London, UK: Digital Energy Journal, Nov. 2018.

PERERA, Charith *et al.* Fog computing for sustainable smart cities: A survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 50, n. 3, p. 1–43, 2017.

QI, Qinglin *et al.* Modeling of Cyber-Physical Systems and Digital Twin Based on Edge Computing, Fog Computing and Cloud Computing Towards Smart Manufacturing. *In* AMERICAN SOCIETY OF MECHANICAL ENGINEERS DIGITAL COLLECTION. ASME 2018 13th International Manufacturing Science and Engineering Conference. College Station, TX, USA: ASME, 2018.

SIEMENS. **Exploring the possibilities offered by digital twins in medical technology**. Erlangen, DE, Nov. 2018. p. 4.

STOJANOVIC, Nenac; MILENOVIC, Dejan. Data-driven Digital Twin approach for process optimization: an industry use case. *In*. 2018 IEEE INTERNATIONAL CONFERENCE ON BIG DATA. Seattle, WA, USA: IEEE, 2018.

SUNEJA, N. ScyllaDB Optimizes Database Architecture to Maximize Hardware Performance. **IEEE Software**, v. 36, n. 4, p. 96–100, 2019.

TANG, Bo *et al.* A hierarchical distributed fog computing architecture for big data analysis in smart cities. *In*. PROCEEDINGS OF THE ASE BIGDATA & SOCIALINFORMATICS 2015. Kaohsiung, Taiwan: Association for Computing Machinery, 2015. p. 1–6.

TIOBE PROGRAMMING COMMUNITY INDEX. Oct. 2020. Available from: `https://www.tiobe.com`. Visited on: 27 Oct. 2020.

UHLEMANN, Thomas; LEHMANN, Christian; STEINHILPER, Rolf. The Digital Twin: Realizing the Cyber-Physical Production System for Industry 4.0. *In*. 24TH Conference on Life Cycle Engineering. Kamakura, Japan: Elsevier Procedia, 2017.

UHLEMANN, Thomas; SCHOCK, Christoph, *et al.* The Digital Twin: Demonstrating the potential of real time data acquisition in production systems. *In*. 7TH CONFERENCE ON LEARNING FACTORIES. Darmstadt, Germany: Elsevier Procedia, 2017.

VELEDAR, Omar; DAMJANOVIC-BEHRENDT, Violeta; MACHER, Georg. Digital Twins for Dependability Improvement of Autonomous Driving. *In* WALKER, Alastair; O'CONNOR, Rory V.; MESSNARZ, Richard (Eds.). **Systems, Software and Services Process Improvement**. Edinburgh, UK: Springer International Publishing, 2019. p. 415–426. ISBN 978-3-030-28005-5.

VERMA, Prabal; SOOD, Sandeep K. Fog assisted-IoT enabled patient health monitoring in smart homes. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 3, p. 1789–1796, 2018.

VOELL, Christian *et al.* How Digital Twins Enable the Next Level of PLM – A Guide for the Concept and the Implementation in the Internet of Everything Era. *In*. PRODUCT LIFECYCLE MANAGEMENT TO SUPPORT INDUSTRY 4.0. Turin, Spain: Springer International Publishing, 2018. p. 238–249.

WAN, Jiafu *et al.* Fog computing for energy-aware load balancing and scheduling in smart factory. **IEEE Transactions on Industrial Informatics**, IEEE, v. 14, n. 10, p. 4548–4556, 2018.

YOUSEFPOUR, Ashkan *et al.* All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, v. 98, p. 289–330, 2019. ISSN 1383-7621. Available from: `http://www.sciencedirect.com/science/article/pii/S1383762118306349`. Visited on: 5 Oct. 2020.

ZHANG, Lucy. **Building Facebook Messenger**. 2011. Available from: `https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920`. Visited on: 30 Oct. 2020.

# APPENDIX A — A CLOUD-FOG COMPUTING ARCHITECTURE FOR REAL TIME DIGITAL TWINS

The following appended document is currently a working paper, submitted and in a review phase, not yet peer-reviewed and thus pending approval. It is a paper written in the CMP158 - Real Time Systems course of the Postgraduate Program in Computing, of the Institute of Informatics of the Federal University of Rio Grande do Sul, ministered by Prof. Dr. Edison Pignaton de Freitas, in 2020, which I enrolled as an Special Student.

The paper describes the usage of the Digital Twin framework in a cloud/fog deployment environment, an attempt to reduce latency in communication for real time systems, which Digital Twins could be, depending on modeling.

# A Cloud-Fog Computing Architecture for Real Time Digital Twins

Francisco Paiva Knebel*, Juliano Araujo Wickboldt, Edison Pignaton de Freitas

*Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil*

**Abstract**

Digital Twin systems are designed as two interconnected mirrored spaces, one real and one virtual, each reflecting the other, sharing information, and making predictions based on analysis and simulations. The correct behavior of a real-time Digital Twin depends not only on the logical results of computation but also on the timing constraints. To cope with the large amounts of data that need to be stored and analyzed, modern large scale Digital Twin deployments often rely on cloud-based architectures. A significant portion of the overall response time of a Digital Twin is spent on moving data from the edge to the cloud. Therefore, implementing Digital Twins using cloud-fog architectures emerges as an alternative to bring computing power closer to the edge, reducing latency and allowing faster response times. This paper studies how suitable the use of a cloud-fog architecture is to handle the real-time requirements of Digital Twins. Based on a realistic implementation and deployment of Digital Twin software components, it is possible to conclude that the distribution of Digital Twins in a fog computing setup can reduce response time, meeting its real-time application requirements.

*Keywords:* Digital Twin, Real Time, Cloud Computing, Fog Computing

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

*Corresponding author

*Email addresses:* francisco.knebel@inf.ufrgs.br (Francisco Paiva Knebel), jwickboldt@inf.ufrgs.br (Juliano Araujo Wickboldt), epfreitas@inf.ufrgs.br (Edison Pignaton de Freitas)

## 1. Introduction

The Digital Twin concept was first introduced in 2002 as a model of a system containing two interconnected mirrored spaces, one real and one virtual, each space reflecting the other while sharing real-time information [1]. Digital Twins must be designed as real-time systems so that the virtual part can reflect the real one accurately. The correctness of the system behavior depends not only on the logical results but also on the timing constraints defined for this system [2].

Essentially, a Digital Twin is a computer program that acquires real-world data about a physical system and generates simulations on how that system will be affected by this data. Long before the terminology, Digital Twins were used in action by NASA, in order to monitor and maintain machines in outer space, a situation where hands-on contact with devices would be impossible, allowing for remote diagnosis and fixing problems that are could occur [3].

To reflect their physical counterparts, Digital Twins are composed by a large number of sensors, responsible for acquiring the used data. The rapid expansion of sensors powered by the Internet of Things (IoT) is what makes Digital Twins possible, but with this increased data generation, centralized applications oftentimes experience high latency and poor connectivity situations which affects the performance of applications. With the advent of IoT devices and its increased data volume in cloud systems, moving all data to the cloud does not satisfy time-sensitive application requirements [4]. Fog computing enables computing, storage, networking, and data management on network nodes in closer proximity to edge devices, allowing the processing to happen not only in the cloud, but also between the edge and the cloud [4].

Applications in several areas can be improved with the usage of fog computing, including smart industries [5, 6], homes [7], cities [8, 9] and healthcare [10], all real-time systems with time-critical processing to avoid accidents and to provide a better and more reliable experience without suffering from cloud latency. Integrating fog computing with IoT and Digital Twins enables mirroring physical and virtual spaces while achieving the necessary timing requirements of real-time systems.

Among the many challenges of implementing a fog computing network, computation offloading and data preprocessing are critical in terms of scalability and guarantee of real-time requirements [11]. Balancing the computational workload between multiple fog nodes, closer to the edge devices, enables the network to be more efficient in handling timing critical tasks, by distributing operations between different devices and reducing communications between edge and cloud. For Digital Twins operate under time-sensitive conditions, research

is still required so that a number of connected services and devices can scale and correctly execute real-time operations in timely fashion. It is still challenging to manage, process, and store all this data in a highly distributed environment [12, 5].

Literature related to the usage of cloud and fog in IoT-related themes is already existent and discussed, but are limited to abstract and conceptual proposals for these architectures, lacking on practical demonstrations, which show absolute differences between the advantages of each deployment scenario. In light of these challenges and possibilities, this paper proposes the usage of a cloud-fog architecture that can manage a large scale number of real-time Digital Twins, increasing performance while reducing latency, to meet timing requirements. Thus, the contribution of this work is the study of an distributed architecture to support Digital Twin applications that must meet real-time requirements, avoiding timing problems that usually happen in centralized approaches.

The remainder of this paper is structured as follows. Section 2 provides background information for Digital Twin and cloud, fog, and edge computing concepts. Section 3 discusses related work. Section 4 describes the proposed solution. Section 5 presents the experiments, the methodology, and the obtained results. Section 6 discusses the results obtained in the experimental scenarios. Section 7 concludes the paper and presents directions for future work.

## 2. Background

### *2.1. Digital Twins*

Digital Twins are systems designed to work with rich data provided by both the insight of the models and implicit knowledge of objects, products of their behavior, only requiring the availability of raw data about their physical counterparts. To achieve this, Digital Twins must accurately simulate, analyze, and predict real-world events and situations, via the collection of real-time physical data, mirroring it into its virtual space. Digital Twins can not only anticipate and predict but can issue proactive measures to avoid problems by combining real-time data with the knowledge of other Digital Twins instances. Simulations can forecast the outcomes, allowing human operators to choose, knowing the consequences of the possible actions, and set the optimal solution [13].

The implementation of self-aware Digital Twins creates modern control systems that adapt to their environment, not only mimicking but understanding the reasons behind the behavior of their physical counterparts [14]. The virtual part of a Digital Twin system is not necessarily physically implemented along with the native physical devices. Conversely, it can be implemented physically apart

from its real counterpart in a distributed and evolving manner, dynamically learning patterns, and helping predict possible disruptions throughout the life of the system [15].

### *2.2. Cloud, Fog, and Edge Computing*

Cloud computing enables convenient, on-demand access to a shared pool of computing resources [16]. Cloud computing transfers the responsibility of data processing and storage to remote data centers, via the Internet. It allows resources to be geographically distributed, increasing data redundancy and reliability [17, 18]. Fog nodes are distributed computing entities, formed by at least one networked physical processing device, able to execute distributed tasks [19]. Edge computing moves processing nearest to where it is needed, allowing computation closer to the source, reducing cloud traffic and service latency, improving response times [20].

Fog differentiates itself from the cloud due to its nodes being deployed in large numbers at less centralized locations, compared to centralized cloud data centers. In essence, fog computing is an extension of cloud computing into the physical world and their multi-connected devices, ensuring data availability where and when it is needed [21], while edge computing represents the nearest the processing resources can be placed regarding the data source.

This work presents the idea of using both cloud and fog layers, but excluding edge. Since the edge is composed mostly of devices without complete control from the cloud provider, concerns are presented in regards to system reliability and availability. It can be used to reduce latency and enable more computing power, but due to its possible short and temporary availability, using a more controlled layer, in this case, the fog enables the objective of latency reduction while still being supervisable.

## 3. Related Work

Many studies combining cyber-physical systems, Digital Twins, and fog computing exist, but to the best of our knowledge, not in the context of real-time systems. Cyber-physical systems using Digital Twins in the cloud, fog, or edge environments have been proposed by Qi *et al.* [22] and Kim [23] as three levels of systems: unit, system, and system of systems (or service) level. The proposal by Qi *et al.* [22] highlights that cloud computing enables on-demand resources and computational sharing, while fog computing shifts computation, storage, and networking of the cloud to the edge network, and edge computing allows processing at closer proximity to the data source, which can benefit cyber-physical

systems. Kim [23] states that the key driver to moving towards edge/fog computing is time-sensitive communication, which is a required feature for real-time systems. It also mentions that distinctions between the edge, fog, and cloud layer need to be further clarified to be better explored in different setups.

A decentralized model for an edge-fog network with IoT devices is proposed by Mohan and Kangasharju [24]. The proposal has the edge built with voluntary and loosely coupled general-purpose devices, one or two hops from the IoT resources, and horizontal connection between the edge devices. The fog layer, in this model, is a consolidation of networking devices (*i.e.*, routers, switches) where edge devices can offload computationally intensive tasks. The model includes a cloud component, but it is used only for centralized data storage, with no computational capabilities. This approach would not suffice in a scenario with high computational requirements, since it has no fallback to the cloud, in cases where the edge and fog are overloaded with tasks.

The benefits of fog and edge computing, which include low latency, locality, and scalability, are well known. However, Ahmad and Afzal [25] state that although there are solutions for this domain, many aspects are still unexplored in practical scenarios, lacking an architecture that fully implements these principles. Despite this landscape, there are examples of real situations where fog computing was used to build cyber-physical systems [26], to accelerate production processes. In the scenario described by Fernández-Caramés *et al.* [26], experiments revealed that fog nodes executed their tasks faster than the same tasks in a purely cloud-dependent environment. The authors report that these nodes can process more data while under high-load situations, in which real-time processing is constrained.

In short, given the studies presented above, currently, the Digital Twin literature still has many unexplored study areas, lacking on an actual deployment of a real implementation of Digital Twins which utilizes the benefits of a cloud-fog architecture, presenting how it was built, its components and with experiment results proving its benefits.

## 4. Proposal

This section initially describes in detail the problem addressed in this work. Then, it provides an overview of how the Digital Twin architecture was designed, its components, and details on how each of the components is used in the experiments presented in Section 5.

### *4.1. Addressed Problem*

The literature review revealed that the use of cloud and fog computing to build Digital Twins is not an entirely new idea. However, the studies surveyed presented mainly abstract concepts, establishing the overall idea with notions on what to expect from a cloud-based Digital Twin deployment, but without realistic implementations. An actual implementation and extensive experimentation would allow a direct analysis of how systems like this can benefit from the use of cloud, fog, and edge interchangeably, and how this could affect the performance of this network of systems.

The three layers edge-fog-cloud architecture has been presented in multiple investigations [22, 23], but none of which presents this architecture in the context of real-time systems or analyzing how much each layer could add or reduce latency. Another problem present in this situation is the scenario of a public cloud/fog: the network resources are shared between heterogeneous systems. Systems in this network require resource competition management, defining process priority for real-time systems to execute their operations without being frozen because other systems have access to the network resources. The modification of existing systems to support Digital Twins implies the addition of a new real-time approach, which would share resources of a network that could already be near its computational limits.

Non-shared cloud and fog Digital Twin network scenarios presented in other works do not accurately describe all real situations, because they demonstrate scenarios where companies would build an entire infrastructure with the sole purpose of supporting Digital Twins. Real situations where they would be implemented most likely already have a working network where the Digital Twin would be inserted, requiring minimal physical change. Cloud and fog, by definition, enable reusable and cheap computation by handling multiple different and independent devices. Networks with a single purpose do not make sense because idle resources would be wasted, going against the whole concept of cloud computing.

These problems imply two situations: 1) the usage of cloud, fog, and edge computing in the context of workload variability; and 2) the definition of priority for different devices in the network to allow priority computation to be allocated to time-critical systems. Sharing resources in a network requires the definition of a priority policy between devices, to allocate the necessary computation to Digital Twins or other systems. In light of this landscape, the problem addressed in this paper is the investigation of how appropriate a cloud-fog architecture is to support real-time Digital Twins.

*4.2. Digital Twin Design Overview*

Modern systems built with Digital Twins in mind can implement business logic, ranging from simple computing software, only storing and processing data, to complex systems that can "think" and provide advanced insight for its users. The use of artificial intelligence techniques in all types of tasks is the next step in digitalization and process automation, and the use of cloud-based technologies is an essential tool for building scalable systems for any domain.

For Digital Twins to accurately simulate, analyze, and predict events and situations, they must collect a variety of data from the physical mirrored device or system and process it effectively and timely. Digital Twins must support a collection of different models to accurately describe devices in their full life-cycle phases, which may include simulation and predictive models. As so, Digital Twins require to be equipped with a collection of services to effectively monitor and simulate the physical world it mirrors and computes relevant decisions.

The architecture we designed and implemented is an open-source[1] software built as a collection of individual microservices, available under the GNU General Public License v3.0. For the experiments performed in this work, the architecture contains microservices for the client devices, simulating the physical parts, a Digital Twin instance, which is the digital counterpart, and a message broker to intermediate the communication. This work focuses on the communication aspect of the Digital Twin and will not detail internal processing services. These elements are the minimum components required for the Digital Twin communication, emulating the physical and digital parts, from the model proposed by Damjanovic-Behrendt[27].

To execute the experiments, the client devices were simulated with a custom program that sends sequentially numbered messages, with a configurable payload and publication interval, to an MQTT broker, which uses a topic-based publish/subscribe pattern for communication. This allows decoupling from publisher and subscriber, allowing for each service to run independently and with higher scalability. Each client device publishes their messages with a specific topic known by the subscribed twin instance, identifying their publications as being from that specific client. Numbered messages allow us to analyze the timing aspects of each message, from its publication by the client until it is processed by the twin instance.

The Digital Twin instance is a prototype of the digital part of the system, which implements the required mechanisms for acquisition, storage, and processing of data obtained from its physical counterpart, and is where the physical

[1] Source code available at `https://github.com/Open-Digital-Twin`.

product is linked throughout its entire life-cycle [3]. The Digital Twin instance is the final destination for the client messages after they are distributed by the broker. In this work, messages sent in the opposite direction, *i.e.* from the twin instance to real devices, were not experimented with and require analysis by future work.

The current implementation uses the Eclipse Mosquitto message broker, version 1.6.10, as it is an open-source message broker implementation of the MQTT protocol, purposely built to be open, simple, and light weight, characteristics heavily associated with IoT contexts [28]. Eclipse Mosquitto is intended for use in situations where there is a need for lightweight messaging, particularly on constrained devices with limited resources [29]. The broker was configured with its default settings, except for the max message queue limitations which were removed, since the broker would discard all messages received if this limit was reached, only retaining the physical restriction of memory limitations of where the broker is being executed.

To meet real-time requirements, the fact that the Digital Twin components of the architecture we designed and implemented are microservice-based allows for a variety of distribution scenarios. These components can be distributed between cloud, fog, and edge computing layers, depending on the needs of the corresponding physical system and the available network resources.

## 5. Experimental Setup and Methodology

This section describes the performed experiments. It also describes the primary experiment factors, variables that affect the experiment response, which directly impacts the performance of the proposal in the defined scenarios. Factors will be tested in different scenarios, to analyze their impact on the architecture.

### *5.1. Experiment Scenarios*

To perform the proposed study, three different experiment scenarios are proposed. These three scenarios are illustrated in Figure 1 and detailed as follows:

- *Scenario 1 - Cloud only:* Processing is done exclusively on a single cloud node. The MQTT broker, where the client publishes and the twin instance subscribes to a data source, resides on this cloud node.
- *Scenario 2 - Fog and Cloud:* The MQTT broker is located on a fog node, closer to the edge, while the twin instance resides on the cloud.
- *Scenario 3 - Fog only:* Both broker and the twin instance resides on a fog node.

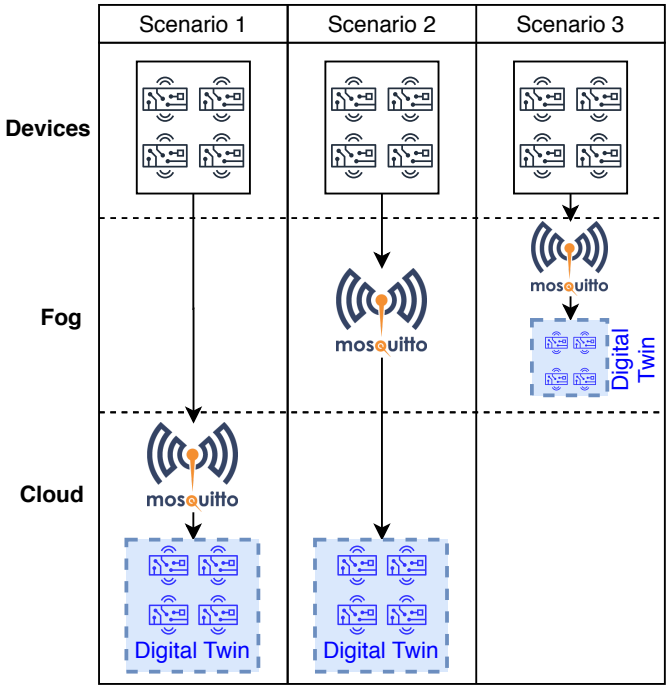



Figure 1: Illustration of the three different experiment scenarios, with their corresponding distributions.

### 5.2. *Experiment Factors*

- *Number of data sources:* twin elements can have multiple data sources, in the form of multiple individual devices, which by themselves can have multiple sensors. For simplification of the problem in this paper, an element is defined as one general part or device of the physical system, which has only one data source. To emulate multiple data sources, it is possible to increase the number of elements accordingly. Each source then sends a certain amount of messages to the broker.

- *Message payload size:* the number of bytes sent in the MQTT message payload generated by the client devices, which will affect bandwidth usage and processing time for the message broker and the twin instance.

- *Source message frequency:* the estimated time between messages sent by the data sources, or client devices, depending on the types of sensors that define the physical system.

- *Transmission latency:* time for the message to travel in two scenarios: 1)

from the client to the broker; and 2) from the broker to the twin instance. This latency is directly affected by the distribution of the architecture between cloud and fog.

Each scenario was individually tested under three situations: (A) varying the number of messages sent by each data source; (B) varying the frequency of messages transmissions; and (C) varying the payload of the sent messages.

- *Situation A - Source and number of message:* Situation A varied the amount of messages sent by each data source, in sets of 100, 1,000, 10,000, and 100,000 messages by each source. To test concurrency, the number of data sources was set to 1, 3, and 5, which would add up to a maximum of 500,000 messages to be handled by the broker and the Digital Twin instance. The delay between when each message was sent by the data sources was fixed to 10ms, with a payload of 64 bytes.

- *Situation B - Message frequency:* Situation B varied the wait interval between each message sent by the data source, in sets of 10, 20, 40, 80, 160, 320, 640, 1280, and 2560 milliseconds. In this situation, a total of 1000 messages were sent by only one source, each with a payload of 64 bytes.

- *Situation C - Message payload size:* Situation C tests variations of payload sizes on the messages sent by the data source. The payloads were set to 8, 16, 32, 64, 128, 256, and 512 bytes. Each test sent a total of 1000 messages from one source, with a fixed delay of 80ms between each message.

### *5.3. Experiment Hardware Specifications*

A server running an Intel Xeon E5-2420 (with 6 cores and 12 threads) and 32 GB of RAM was used as the experiment environment. The experiments were executed on three individual Kernel-based Virtual Machines simulating real devices, each named according to their use on the defined scenarios: Cloud, Fog, and Client. The Cloud virtual machine runs with two virtual CPUs and 4GB of memory. The Fog and Client are by definition devices with more constrained resource capacities, so they only have one virtual CPU each, with 1GB of memory.

Regarding network limitations, the only limit inserted was a simulated network latency in the communication between the machines, varying according to each scenario, injected by the usage of *tc-netem* [30]. In scenario 1, a delay of 100ms, with a 10ms variation, was added in the Client. In scenarios 2 and 3, two different delays were added: 40ms, varying 10ms, in the Client, and 20ms, varying 5ms, in the Fog. Delays were injected into the scenarios to simulate the

different delays when devices communicate with fog and cloud components, not only with fog having lower hardware capabilities but also having a lower delay. Considering just two machines, without any communication delay difference, the one with more computational power would have a faster response, but computing in the edge/fog layers is the concept that communication can be the bottleneck in how specific distributed systems function.

## 6. Results

Bringing processing closer to the edge, to handle message exchange, reduces latency for the application. Digital Twins, being composed with a multitude of distributed sensors, could experience high latency and poor connectivity. Proposing processing being done closer to edge is a possible strategy, since being purely dependent on the cloud may not satisfy time-sensitive requirements [4]. The processing unit can receive the data from the source devices and compute its suggestion to the physical system with a shorter response time.

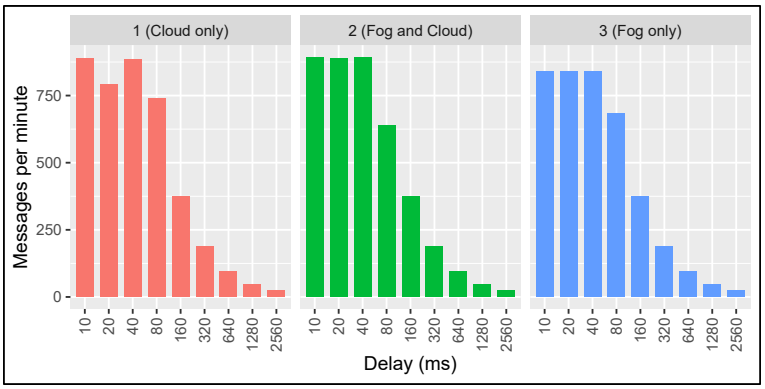



Figure 2: Messages per minute received by the Digital Twin instance, in each scenario, in the message frequency situation.

An important observation about how the MQTT broker throttles messages was found when testing Situation B, varying the message frequency, in the described scenarios. As evidenced by Figure 2, which shows the amount of messages received per minute in each scenario, varying the delay between publications, the broker has a limit on the number of messages it can receive and publish to its subscribers in the order of 800 messages per minute. We did not set a limit to the message queue in order to prevent the broker from dropping messages under high load situations. Thus, this message per minute rate is a hard limit set by

the broker in order to prioritize receiving publications and to not overload subscribers through limiting the number of in-flight messages. Eclipse Mosquitto can be configured to increase this in-flight message limitation or even remove it. However, doing so could cause message ordering to break or it could overload subscribers and should be tested in a case by case scenario, according to the number of messages sent by the real device.

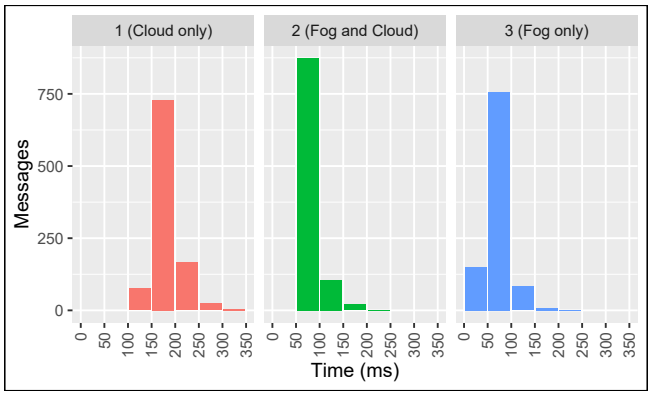



Figure 3: Timing histogram of 1000 messages, from client publish until reaching the Digital Twin, in each scenario.

Figure 3 presents a timing histogram of 1000 messages in the three scenarios, on situation C with a payload of 64 bytes and 80ms interval between each message, from the time it is published by the client until received by the Digital Twin instance (subscriber). Results from this experiment show not just the average timing of messages, but the time of each individual publication. Scenario 1 presents messages with a higher time compared to the two other scenarios, where most messages are received in the 50 to 100 ms interval, in this experiment. Considering a real-time system with a communication requirement of 200ms in the Cloud only scenario (1), 805 of the 1000 messages would respect this limitation, representing 80.5% of the total messages sent by the client. Meanwhile, only one message would not arrive in time for this imposed system limitation in the Fog and Cloud (2) and Fog only (3) scenarios, respectively. Further reducing this limitation to 150ms, 92.4% of messages sent in scenario 1 would be lost, while 0.21% and 0.09% would not respect the requirement by scenarios 2 and 3.

More benefits depend on the architecture scale of implementation in the fog/cloud levels. By having resources closer to the edge, it is possible to perform preprocessing as soon as possible, before centralizing the data in the cloud. Figure 4 makes this clear by showing the time each message takes from being sent by the client until received by the twin instance in each deployment scenario.

The situation shown in this image sends 1000 messages, each one with a delay between each publication of 80ms. Table 1 provides a summation of the average times of this experiment, providing a general overview of all the published messages.

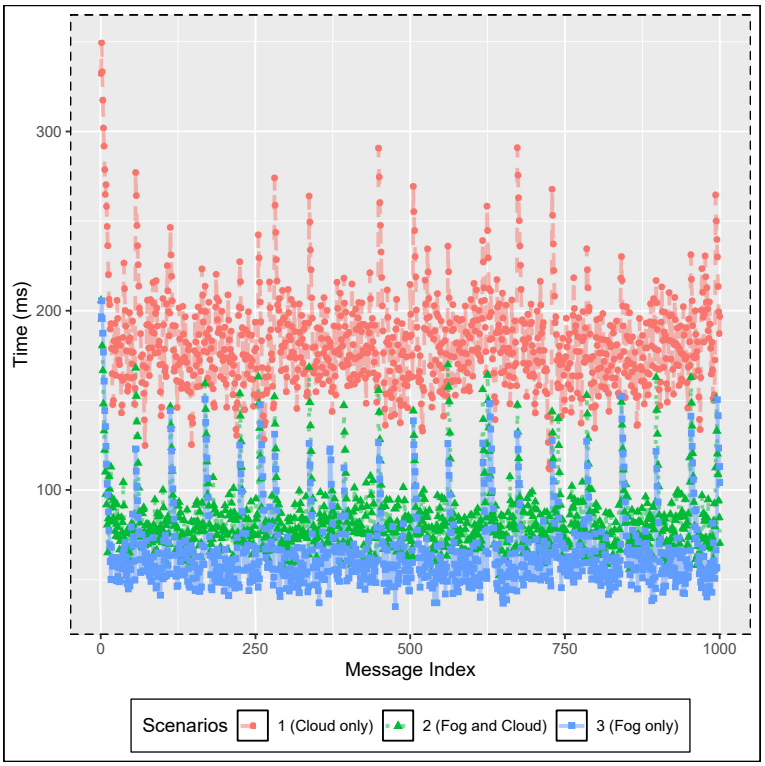



Figure 4: Timing per message, from client publish to the Digital Twin receiving, in each scenario, sending 1000 messages with a 80ms delay between each publication.

Table 1: Average times per message, from client to twin, for the 80ms delay experiment situation.

| Time per message (ms) | Average | Std Deviation | Max | Min |
|---|---|---|---|---|
| Scenario 1 | 182.45 | 27.932 | 349.35 | 111.82 |
| Scenario 2 | 84.43 | 20.417 | 205.83 | 52.51 |
| Scenario 3 | 66.08 | 22.626 | 205.63 | 35.13 |

The usage of cloud computing enables devices to store data in a centralized location, but large amounts of data being transmitted, natural to the concept of Digital Twins with a large scale network of sensors, implies an increasing delay, affecting response times from the server.

As shown in Table 1, deploying to the Fog and Cloud scenario (2) and the Fog only scenario (3) reduced average times per message by 54% and 64% respectively, compared to the Cloud only scenario (1). Maximum times were also reduced by 41% for scenarios 2 and 3, as were the minimum times by 53% and 69%.

As a response from the results presented in this work, fog computing is shown as a viable alternative to reduce the effects of this problem, reducing the transmission delay in order to match real-time requirements. Further reductions depend on improving lower layers of the network components, via extra connectivity and reduced latency in general.

## 7. Conclusion and future work

This paper investigated how the usage of a cloud-fog architecture for Digital Twins can help in meeting real-time requirements of these systems.

The tested architecture includes a client, for sending sensor data, the MQTT broker, for handling messages sent by the client, and the Digital Twin instance, which is subscribed by the broker. Tests were executed in three different scenarios, changing the deployment locations of the architectural elements where the broker and twin reside in the cloud, where the broker resides in a fog node, and the twin on the cloud, and lastly where both the broker and twin share the same fog node.

The experiments tested these different scenarios on how they would handle the number of messages, different frequency of messages sent, and payload size. From these experiments, it was concluded that the distribution of these Digital Twin elements closer to the edge reduce communication delays for the end application, allowing faster response by bringing preprocessing into distributed locations. This particular distribution allows the Digital Twin to meet real-time systems timing specifications.

Having all the required data flow from the client devices, the Digital Twin can run application-specific algorithms handling this input and be able to suggest actions to the physical systems respecting their real-time requirements.

As future work, some limitations could be addressed, as the problem regarding the real-time priority policy for the MQTT broker: message handling should prioritize real-time subscribers. The MQTT protocol has no definition of priority for subscriptions, but a broker-defined configuration indicating priority topics could be used, dedicating more resources to attempt a best-effort approach into meeting real-time requirements.

## Acknowledgment

This work was supported in part by the Brazilian National Council for Scientific and Technological Development (CNPq) and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. We also thank the funding of CNPq, Research Productivity Scholarship grants ref. 313893/2018-7 and ref. 306614/2017-0.

## References

[1] M. Grieves, Origins of the Digital Twin Concept, Technical Report, Florida Institute of Technology / NASA, 2016.

[2] H. Kopetz, Real-time systems: design principles for distributed embedded applications, Springer Science & Business Media, 2011.

[3] M. Grieves, J. Vickers, Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems, Springer International Publishing, 2017, pp. 85–113.

[4] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture 98 (2019) 289 – 330. URL: `http://www.sciencedirect.com/science/article/pii/S1383762118306349`.

[5] W. Khan, M. Rehman, H. Zangoti, M. Afzal, N. Armi, K. Salah, Industrial internet of things: Recent advances, enabling technologies and open challenges, Computers & Electrical Engineering 81 (2020) 106522.

[6] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, C. Liu, Fog computing for energy-aware load balancing and scheduling in smart factory, IEEE Transactions on Industrial Informatics 14 (2018) 4548–4556.

[7] P. Verma, S. K. Sood, Fog assisted-IoT enabled patient health monitoring in smart homes, IEEE Internet of Things Journal 5 (2018) 1789–1796.

[8] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, Q. Yang, A hierarchical distributed fog computing architecture for big data analysis in smart cities, in: Proceedings of the ASE BigData & SocialInformatics 2015, 2015, pp. 1–6.

[9] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, A. V. Vasilakos, Fog computing for sustainable smart cities: A survey, ACM Computing Surveys (CSUR) 50 (2017) 1–43.

[10] Y. Nikoloudakis, E. Markakis, G. Mastorakis, E. Pallis, C. Skianis, An NFV-powered emergency system for smart enhanced living environments, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2017, pp. 258–263.

[11] C. Li, Y. Xue, J. Wang, W. Zhang, T. Li, Edge-oriented computing paradigms: A survey on architecture design and system management, ACM Computing Surveys (CSUR) 51 (2018) 1–34.

[12] J. Dizdarević, F. Carpio, A. Jukan, X. Masip-Bruin, A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration, ACM Comput. Surv. 51 (2019).

[13] C. Voell, P. Chatterjee, A. Rauch, J. Golovatchev, How Digital Twins Enable the Next Level of PLM – A Guide for the Concept and the Implementation in the Internet of Everything

Era, in: Product Lifecycle Management to Support Industry 4.0, Springer International Publishing, 2018, pp. 238–249.
[14] N. Stojanovic, D. Milenovic, Data-driven Digital Twin approach for process optimization: an industry use case, in: 2018 IEEE International Conference on Big Data, 2018.
[15] O. Cardin, Classification of cyber-physical production systems applications: Proposition of an analysis framework, Computers in Industry 104 (2019) 11–21. doi:https://doi.org/10.1016/j.compind.2018.10.002.
[16] P. Mell, T. Grance, et al., The NIST definition of cloud computing, National Institute of Science and Technology, Special Publication, 800 145 (2011).
[17] D. C. Marinescu, Cloud computing: theory and practice, Morgan Kaufmann, 2017.
[18] J. Deng, S. C.-H. Huang, Y. S. Han, J. H. Deng, Fault-tolerant and reliable computation in cloud computing, in: 2010 IEEE Globecom Workshops, IEEE, 2010, pp. 1601–1605.
[19] E. M. Tordera, X. Masip-Bruin, J. Garcia-Alminana, A. Jukan, G.-J. Ren, J. Zhu, J. Farré, What is a fog node? a tutorial on current concepts towards a common definition, arXiv preprint arXiv:1611.09193 (2016).
[20] Y. Chen, S. Deng, H. Ma, J. Yin, Deploying data-intensive applications with multiple services components on edge, Mobile Networks and Applications (2019) 1–16.
[21] H. F. Atlam, R. J. Walters, G. B. Wills, Fog Computing and the Internet of Things: A Review, Big Data and Cognitive Computing 2 (2018) 10.
[22] Q. Qi, D. Zhao, T. W. Liao, F. Tao, Modeling of Cyber-Physical Systems and Digital Twin Based on Edge Computing, Fog Computing and Cloud Computing Towards Smart Manufacturing, in: ASME 2018 13th International Manufacturing Science and Engineering Conference, American Society of Mechanical Engineers Digital Collection, 2018.
[23] T. Kim, Cyber Physical Systems Framework of Edge-, Fog-, and Cloud-Computing, in: Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2019, pp. 10–14.
[24] N. Mohan, J. Kangasharju, Edge-Fog cloud: A distributed cloud for Internet of Things computations, in: 2016 Cloudification of the Internet of Things (CIoT), IEEE, 2016, pp. 1–6.
[25] S. Ahmad, M. M. Afzal, Deployment of Fog and Edge Computing in IoT for Cyber-Physical Infrastructures in the 5G Era, in: International Conference on Sustainable Communication Networks and Application, Springer, 2019, pp. 351–359.
[26] T. M. Fernández-Caramés, P. Fraga-Lamas, M. Suárez-Albela, M. A. Díaz-Bouza, A Fog Computing Based Cyber-Physical System for the Automation of Pipe-Related Tasks in the Industry 4.0 Shipyard, Sensors 18 (2018) 1961.
[27] V. Damjanovic-Behrendt, W. Behrendt, An open source approach to the design and implementation of digital twins for smart manufacturing, International Journal of Computer Integrated Manufacturing 32 (2019) 366–384.
[28] A. Banks, E. Briggs, K. Borgendale, R. Gupta, MQTT Version 5.0, 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, accessed: 2020-08-09.
[29] R. A. Light, Mosquitto: server and client implementation of the MQTT protocol, Journal of Open Source Software 2 (2017) 265. URL: https://doi.org/10.21105/joss.00265.
[30] S. Hemminger, F. Ludovici, H. P. Pfeifer, NetEm - Network Emulator, 2011. URL: https://man7.org/linux/man-pages/man8/tc-netem.8.html, accessed: 2020-08-22.

# APPENDIX B — UMA AVALIAÇÃO DO USO DE MQTT PARA A IMPLEMENTAÇÃO DE DIGITAL TWINS

The following document is a paper written, submitted, reviewed and approved at ERRC 2020 (Escola Regional de Redes de Computadores). This work was done to further understand the limitations and learn what to expect from MQTT-based communication for our Digital Twin framework and to allow for the best relationship between client and twin instances as possible.

The paper analyses how Eclipse Mosquitto handles a large number of messages sent in a small interval and reaches the best use case, obtaining the maximum message rate per broker. This is further extended by deploying multiple brokers, in a centralized and in a distributed manner, so they can work in parallel and allow for an increased total message rate in the system.

# Uma Avaliação do uso de MQTT para a Implementação de Digital Twins

**Rafael Trevisan**[1], **Francisco Paiva Knebel**[1], **Juliano Araújo Wickboldt**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{rafael.trevisan,francisco.knebel,jwickboldt}@inf.ufrgs.br

***Resumo.*** *O conceito de Digital Twin tem se tornado cada dia mais importante como um método para monitorar elementos e prevenir possíveis defeitos. Uma das tecnologias que pode ser usada para implementar a comunicação de dados desse software é o MQTT. Este trabalho avaliou o uso dessa tecnologia nesse contexto e constatou que existe uma limitação do broker na capacidade de envio de mensagens, que pode ser contornada através da utilização de múltiplos brokers em paralelo. Também foi examinado o comportamento dos brokers em situações onde eles estão centralizados em uma máquina com mais recursos computacionais e quando eles estão distribuídos em máquinas exclusivas, porém com menos recursos. Nessas situações constatamos que os brokers apresentam um comportamento mais estável quando seus serviços estão centralizados.*

## 1. Introdução

Com o progresso da informática em áreas como a Internet das Coisas (IoT) e inteligência artificial, a indústria tem adotado soluções cada vez mais sofisticadas para a manutenção e monitoramento de seus produtos e estruturas. Uma destas novas soluções é chamada Digital Twin (DT). A definição de DT surgiu em 2002 e tem como objetivo replicar algum elemento que existe em um espaço físico, em um espaço virtual correspondente [Grieves 2016]. Para fazer isso, o DT é atualizado em tempo real com dados vindos de sensores, que captam as características físicas do elemento e as transmitem. O DT então analisa os dados recebidos e é capaz de dar recomendações sobre o uso e manutenção desse elemento. Dessa forma o software pode revelar problemas em potencial para os seus usuários, para que eles sejam resolvidos de forma proativa [Boschert and Rosen 2016].

Para criar uma estrutura como essa, uma das questões que precisa ser resolvida é a necessidade da comunicação entre o DT e os sensores que fornecem dados. Para este fim, o protocolo Message Queue Telemetry Transport (MQTT) é indicado, por ser leve, econômico e compatível com o uso de aparelhos de baixa capacidade computacional [Banks et al. 2019]. O MQTT funciona através de um sistema de *publish*/*subscribe* regulado por um mediador (*broker*). Neste sistema, primeiramente, os clientes que precisam receber dados (*subscribers*) registram os tópicos de seu interesse no *broker*. Em seguida, os clientes que precisam publicar dados (*publishers*) enviam mensagens ao *broker*, que fica então responsável por entregar as informações aos *subscribers*.

Embora o MQTT seja um protocolo considerado confiável para entrega de mensagens, a dependência de um elemento na arquitetura (o *broker*) para mediar toda a

comunicação entre os clientes pode representar um gargalo no sistema. A agilidade para o envio de mensagens depende diretamente do desempenho desse elemento mediador. A aquisição de dados em DTs deve ser projetada como um sistema de tempo real para que a parte virtual possa refletir o seu equivalente real com precisão. O comportamento correto do sistema depende não apenas dos resultados lógicos, mas também do respeito às restrições de tempo real definidas para aquele sistema. Nesse sentido, um estudo sobre o comportamento do *broker* é importante para avaliar a viabilidade do uso desse tipo de comunicação em DTs de larga escala.

Neste trabalho é analisado o comportamento do *broker* em situações onde ele é posto sob estresse considerando a arquitetura prevista pelos DTs e o volume de dados gerado por eles. Estes são cenários onde o *broker* está sendo sobrecarregado e experimentos buscando métodos de lidar com esse fluxo de mensagens distribuindo-as entre mais *brokers* e mais máquinas. Através desses experimentos pode-se observar o comportamento do *broker* em diferentes situações e avaliar quais seriam os impactos no funcionamento e nos requisitos de tempo real de um DT. Através dos resultados obtidos é possível sugerir quais são as formas mais adequadas para a implementação da comunicação do DT e definir suas limitações.

O restante desse trabalho está estruturado da seguinte forma. A seção 2 discute o *background* e os trabalhos relacionados. Na seção 3 são apresentadas a metodologia e os experimentos que foram realizados. Os resultados são discutidos na seção 4 e a seção 5 relata as conclusões obtidas através desses resultados.

## 2. Trabalhos Relacionados

O conceito de Digital Twin surgiu do processo de *Product Lifecycle Management* (PLM), proposto como uma representação não estática de um produto durante todo o seu ciclo de vida desde a sua fabricação até o fim de seu uso [Grieves 2016]. Desde a criação deste conceito, já era prevista a separação dos espaços reais dos espaços virtuais e o fluxo de dados indo do espaço real para o virtual. Com o tempo o PLM evoluiu para o conceito mais moderno de DT, mantendo as suas principais ideias, mas se adequando as novas demandas do mercado.

Estudos recentes já discutem a possibilidade de implementar um DT com tecnologias de código aberto atuais [Damjanovic-Behrendt and Behrendt 2019]. Segundo Damjanovic-Behrendt e Behrendt, um dos elementos fundamentais para a implementação de um DT são os sistemas de aquisição de dados, do inglês *Data Acquisition Systems* (DAS). Nesse contexto, o MQTT é visto como uma boa alternativa para regular o fluxo das mensagens por ser otimizado para conectar clientes de baixa capacidade computacional com um servidor. Estudos já fizeram análises de desempenho do *broker* MQTT, onde foi possível observar disparidades, sendo elas através da comparação entre diferentes implementações de *broker* ou configurações do protocolo [Mishra 2018]. Porém, esse tipo de análise ainda deixa em aberto outros aspectos que podem influenciar na comunicação entre um sensor e seu DT, como opções de distribuição e balanceamento de carga entre diversos *brokers* a fim de obter escalabilidade na comunicação. Este trabalho irá explorar mais a fundo essas questões com o objetivo de encontrar a maneira mais eficiente de distribuir os elementos da arquitetura de um DT.

## 3. Metodologia

A avaliação proposta neste trabalho consiste em analisar o comportamento do *broker* MQTT para o envio de volumes de mensagens maiores do que ele é capaz de processar. Esta situação será recriada em diferentes contextos a fim de se determinar qual a configuração mais otimizada para a implementação de um Digital Twin. Para realizar estes experimentos, além do *broker* foram utilizadas outras duas estruturas, o cliente MQTT e a instância de DT.

### 3.1. *Broker* MQTT

Como *broker* de MQTT foi utilizado o Eclipse Mosquitto versão 1.6.10. Embora existam outras implementações de *brokers* MQTT voltados para a escalabilidade [Pipatsakulroj et al. 2017, Jutadhamakorn et al. 2017] ou para uso em dispositivos embarcados [Espinosa-Aranda et al. 2015], o Mosquitto foi escolhido devido à sua popularidade e ser uma implementação já integrada em produtos comerciais, para uso em dispositivos com recursos limitados [Light 2017]. Ele trabalha ordenando as mensagens recebidas em uma fila e as distribuindo para seus destinatários. Esse software recebe as mensagens e as coloca em uma fila com tamanho limitado, portanto se executarmos esse experimento com as configurações padrão deste *broker*, haveria perda considerável de mensagens. Para contornar isso, as configurações do *broker* foram alteradas para que a limitação de tamanho máximo da fila fosse removido, restando apenas a restrição física da quantidade de memória do computador onde o *broker* executa. A agilidade no envio das mensagens depende do quão eficiente o Eclipse Mosquitto é para processar as regras de encaminhamento e esvaziar a fila.

### 3.2. Clientes MQTT

Para realizar o experimento foi feita a implementação de um cliente MQTT que envia mensagens de forma controlada. Esse programa cria mensagens do tipo *publish* numeradas sequencialmente com um tamanho de *payload* configurável e as envia ao *broker* utilizando um tópico específico e um intervalo constante também configurável. O momento onde cada uma dessas mensagens é enviada é monitorado através de um *log* gerado localmente pelo cliente.

### 3.3. Instância de DT

A instância de DT é um protótipo da parte virtual de um twin[1] que implementa os mecanismos necessários para aquisição, armazenamento e processamento de dados oriundos da parte física do DT. Neste estudo, a instância de DT é utilizada como destino das mensagens enviadas pelos clientes MQTT após elas serem distribuídas pelo *broker*. A instância de DT registra as mensagens recebidas em um banco de dados local e gravando também o momento quando uma mensagem é recebida. Neste trabalho não consideramos nenhum tipo de processamento das mensagens recebidas nem envio no sentido inverso (da instância de DT para o cliente), apesar de serem operações possíveis e necessárias, ficam, por hora, fora do escopo do trabalho.

[1]Código aberto disponível em `https://github.com/Open-Digital-Twin`.

### 3.4. Configuração do Ambiente de Experimentação

Como ambiente de experimentação foi utilizado um servidor com processador Intel Xeon E5-2420 (com 6 núcleos físicos e 12 *threads*) e 32 GB de RAM. Nesse servidor foram instanciadas máquinas virtuais, conectadas a uma rede local também virtual estabelecida através de uma Linux Bridge, e com três configurações de recursos computacionais: Máquinas do tipo 1, com 2 GB de memória e 1 VCPU, máquinas do tipo 2, com 4 GB de memória e 2 VCPUs e Máquinas do tipo 3, com 8 GB de memória e 4 VCPUs.

Considerando esse ambiente, foi conduzido um primeiro experimento enviando 1000 mensagens MQTT *publish*, com *payload* fixado em 64 bytes, do cliente MQTT para uma instância de DT em intervalos de tempo cada vez menores, a fim de definir qual capacidade de envio de mensagem de um único *broker*. Para determinar isso foi analisado o número de mensagens da fila do *broker* em diversos instantes de seu funcionamento.

Para contornar a limitação no fluxo de saída de mensagens de um *broker* individual é possível distribuir sua carga de mensagens para outros *brokers* que funcionem em paralelo considerando que o Eclipse Mosquitto é um processo leve. Em um segundo experimento foi avaliada a diferença de desempenho no envio do mesmo número de mensagens quando elas são distribuídas entre mais *brokers*, considerando dois cenários: (*Centralizado*) onde eles estão centralizados em uma máquina virtual apenas, mas com aumento de recursos computacionais ou (*Distribuído*) onde os *brokers* estão distribuídos em máquinas virtuais com menos recursos, porém exclusivas.

Como esses experimentos foram realizados em um servidor controlado, os resultados obtidos não representam os valores que seriam encontrados ao se executar esses testes com dispositivos reais. Porém, com esses experimentos é possível tirar conclusões sólidas sobre os casos estudados por os cenários se assemelharem muito com a realidade.

## 4. Resultados

### 4.1. Experimento 1

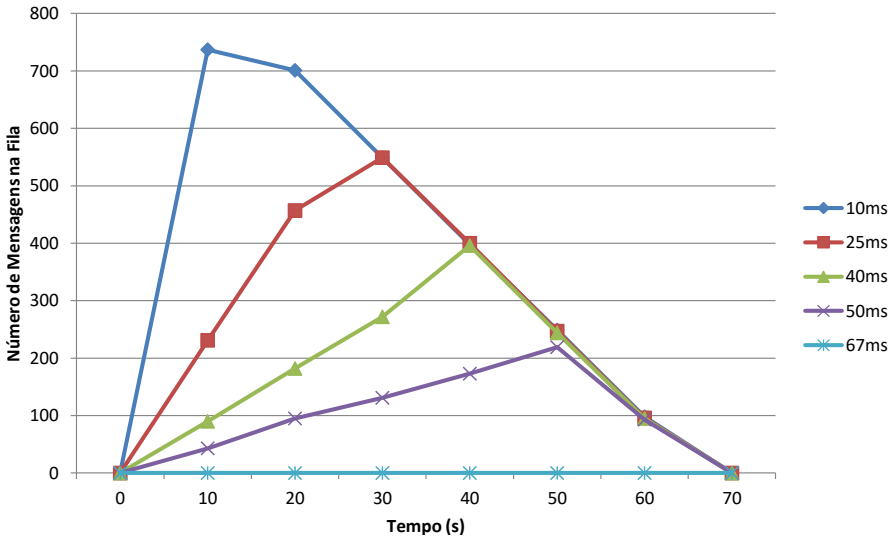



**Figura 1. Gráfico que demonstra o tamanho da fila ao longo do tempo.**

Baseado na abordagem discutida na seção 3, foi realizado um primeiro experimento para determinar a taxa máxima de envio de mensagens quando se opera com apenas um *broker*. Neste experimento foram enviadas 1000 mensagens pelo MQTT em intervalos gradativamente menores e foi analisado o comportamento do tamanho da fila em diferentes intervalos de tempo. Com essa análise podemos perceber que em cada uma dessas situações é possível determinar a taxa de saída máxima de mensagens do *broker*. Esses testes foram realizados em uma máquina do tipo 3, com todos os serviços centralizados.

Avaliando o comportamento descrito na Figura 1, podemos verificar que o *broker* é capaz de enviar as 1000 mensagens em torno de 67 segundos para intervalos de envio menores que 67ms. Com esses valores conseguimos determinar que o Eclipse Mosquitto consegue enviar no máximo aproximadamente 15 mensagens por segundo nessas condições. Para intervalos de envio maiores que 67 ms, não ocorre a criação de fila e as mensagens são encaminhadas antes das próximas serem recebidas.

### 4.2. Experimento 2

Para tentar aumentar a vasão de mensagens foi conduzido outro experimento. Nele as mensagens foram enviadas com uma frequência fixa de 10ms, porém essa carga foi distribuídas para mais *brokers* que funcionam em paralelo. Com esse experimento pretendíamos avaliar como eles se comportariam ao resolver as filas criadas. Como base foi utilizado o cenário onde existe apenas um *broker* em uma máquina do tipo 1. A partir deste cenário, foram criados outros dois, o Centralizado e o Distribuído, para verificar se seria mais eficiente aglomerar todos os *brokers* em uma máquina com mais recursos ou distribuí-los em máquinas menores, sempre mantendo uma equivalência de recursos entre os dois cenários. Para o cenário Centralizado foi usada uma máquina do tipo 2 para realizar os testes com 2 *brokers* e uma máquina tipo 3 para realizar os testes com 4. O cenário Distribuído utilizou uma máquina do tipo 1 para cada.

#### 4.2.1. Cenário Centralizado

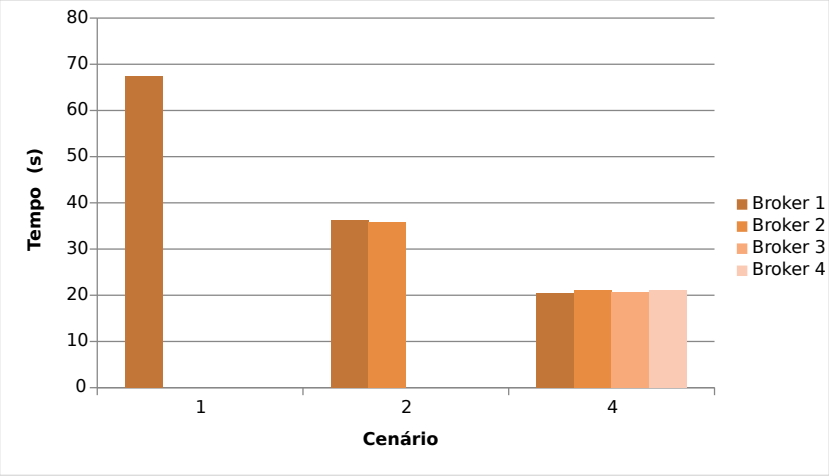



**Figura 2. Gráfico que mostra o tempo de operação dos brokers no cenário centralizado.**

Com a análise do comportamento dos diferentes *brokers* apresentado na figura 2, podemos perceber que embora distribuir as mensagens seja um bom método para diminuir a quantidade de tempo total de envio de mensagens, a performance individual dos *brokers* é prejudicada. Quando foi utilizado apenas um ele apresentou uma taxa de envio de 14.85 mensagens por segundo. Avaliando a implementação de 2 *brokers* a frequência de envio foi em média 13.84 (27.68 no total). Avaliando 4, ela caiu para 12.02 (48.08 no total).

### 4.2.2. Cenário Distribuído

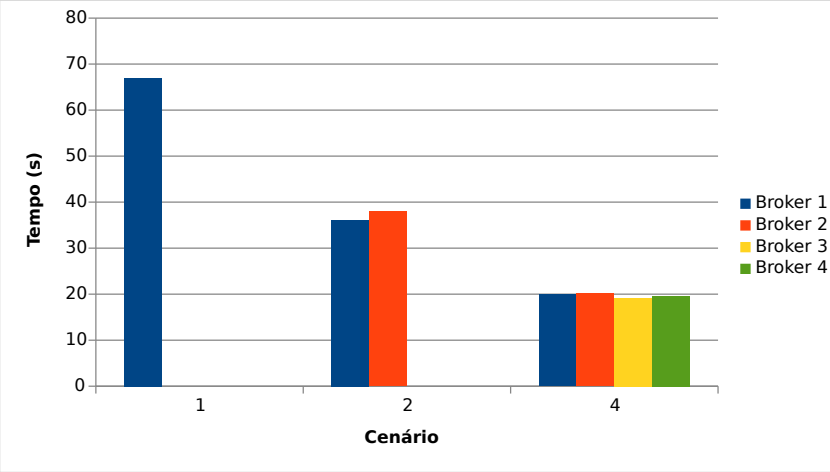



**Figura 3. Gráfico que mostra o tempo de operacão dos brokers no cenário distribuído.**

Como podemos observar na figura 3, quando dispostos em um ambiente distribuído as mensagens são processadas em uma velocidade similar ao cenário Centralizado. A configuração de dois *brokers* apresentou uma taxa de 13.50 mensagens por segundo (27.00 no total), enquanto a configuração de quatro *brokers* apresentou 12.68 (50.72 no total).

Embora os cenários Centralizado e Distribuído apresentem resultados parecidos, o cenário Distribuído apresenta uma variância consideravelmente maior no tempo tomado por cada um de seus *brokers*. No cenário Centralizado as variâncias apresentadas são de 7.51 e 0.61 para 2 e 4 *brokers* respectivamente. No Cenário Distribuído esses valores aumentaram para 14.36 e 5.24. Esses valores foram encontrados com 5 iterações do experimento.

## 5. Conclusão

Neste trabalho, estudamos o MQTT com a intenção de usá-lo para o desenvolvimento de um DT. Foram realizados experimentos envolvendo o envio de mensagens em diferentes taxas para determinar as limitações de desempenho de um cenário com um único *broker*. Além disso, foram avaliadas duas configurações viáveis para a arquitetura desse sistema, centralizada ou distribuída, visando melhorar o seu desempenho. Através dos resultados descobrimos que existe um limite da frequência de envio de mensagens através do *broker*, que no caso da configuração do ambiente de experimentação foi de aproximadamente 15

mensagens por segundo. Contornamos essa limitação empregando diversos *brokers* em paralelo em dois cenários (centralizado e distribuído). Ambos os cenários apresentaram resultados similares, porém constatamos que adotar uma arquitetura centralizada tende a gerar uma variância menor nos resultados criando um ambiente mais estável. Embora aglomerar os *brokers* gere um ambiente mais estável, essa organização cria um único ponto de falha no sistema.

Os experimentos realizados não levaram em conta algumas configurações do MQTT que podem influenciar na agilidade do envio de mensagens, como a qualidade do serviço e a persistência das mensagens, mesmo esse fatores tendo baixo impacto no desempenho do *broker*. Eles também não levaram em conta fatores externos que podem impactar na transmissão das mensagens como latência na rede, *jitter* ou perda de pacotes. Como trabalhos futuros, pretendemos realizar mais experimentos levando em conta esses fatores para determinar de forma mais precisa o comportamento do *broker* e suas consequências na implementação de um DT em situações mais realísticas.

## Referências


Banks, A., Briggs, E., Borgendale, K., and Gupta, R. (2019). MQTT Version 5.0. `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html`. Online; acessado em 09 Ago, 2020.

Boschert, S. and Rosen, R. (2016). Digital twin—the simulation aspect. In *Mechatronic futures*, pages 59–74. Springer.

Damjanovic-Behrendt, V. and Behrendt, W. (2019). An open source approach to the design and implementation of Digital Twins for Smart Manufacturing. *International Journal of Computer Integrated Manufacturing*, 32(4-5):366–384.

Espinosa-Aranda, J. L., Vallez, N., Sanchez-Bueno, C., Aguado-Araujo, D., Bueno, G., and Deniz, O. (2015). Pulga, a tiny open-source MQTT broker for flexible and secure IoT deployments. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 690–694.

Grieves, M. (2016). Origins of the Digital Twin Concept. Technical report, Florida Institute of Technology / NASA.

Jutadhamakorn, P., Pillavas, T., Visoottiviseth, V., Takano, R., Haga, J., and Kobayashi, D. (2017). A scalable and low-cost MQTT broker clustering system. In *2017 2nd International Conference on Information Technology (INCIT)*, pages 1–5.

Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2(13):265.

Mishra, B. (2018). Performance evaluation of MQTT broker servers. In *International Conference on Computational Science and Its Applications*, pages 599–609. Springer.

Pipatsakulroj, W., Visoottiviseth, V., and Takano, R. (2017). mumq: A lightweight and scalable mqtt broker. In *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE.