Farhad Hajnoruzi

Design and Implementation of a Microservices-Based Digital Twin Architecture

Master's thesis in Simulation and Visualization
Supervisor: Ricardo da Silva Torres
Co-supervisor: Arne Styve
September 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU

ABSTRACT

By bridging the gap between the physical and digital worlds, digital twins stand out as a valuable asset for informed decision-making based on complex simulations and predictive analytics. As they navigate the increasing complexity of real-time data processing systems, digital twins necessitate the employment of architectures that uphold scalability and reliability. This thesis embarks on a comprehensive investigation of the design and implementation of a microservices-based digital twin solution and aims at enhancing the growing need for scalable and reliable architectures in such systems. In the pursuit of that, this thesis is anchored on three primary goals. It first proposes a solid microservices-based architectural framework for digital twins. The thesis then investigates the relationship between the use of microservices architecture and the scalability and reliability of the digital twin solution. The third goal of this study is to identify the challenges and trade-offs faced during the design, implementation, and deployment of the proposed model. Seven main microservices are defined in the architecture design among which five more critical ones (data acquisition, digital twin management, simulation, command and control, and visualization) are implemented and evaluated. To validate the suggested design, an extensive test case is conducted employing a digital twin of a wind turbine. This test mirrors real-world conditions by employing a variety of simulation modes, guaranteeing a thorough assessment of the system in conditions applicable to real-world scenarios. The thesis offers useful insights into the efficiency of the suggested microservices-based architecture in achieving scalability and reliability in digital twin models. While providing a sound practical foundation, it thoroughly examines and underscores the associated challenges and trade-offs, presenting a balanced and informed road map for future advancements in this field.

PREFACE

This thesis is the summit of my work on the subject of Simulation and Visualization, more specifically, in the area of real-time data processing using microservices-based digital twin solutions. My interest in this study subject was triggered by the rising need for robust, scalable systems, especially in the renewable energy industry.

I would like to express my deepest gratitude to my supervisors Ricardo Torres and Arne Styve, whose expertise and guidance were invaluable throughout this research. Thanks are also due to my family and friends for their unwavering support and encouragement that made this work possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **API** Application Programming Interface
- **DB** Database
- **DT** Digital Twin
- **FMI** Functional Mockup Interface
- **FMU** Functional Mockup Unit
- **IoT** Internet of Things
- **PLM** Product Lifecycle Management
- **RQ** Research Question
- **SQL** Structured Query Language
- **YAML** YAML Ain't Markup Language

CHAPTER

ONE

INTRODUCTION

1.1 Motivation

The concept of the "Digital Twin" has evolved as a key tool in the age of rapid technology advances and Industry 4.0, linking the physical and digital domains. Digital twins, which are digital clones of actual entities, enable real-time data processing, allowing enterprises to foresee, visualize, and manage problems before they occur [1]. The digital twin acts as a mirror, reflecting the real-time status, working conditions, and properties of its physical counterpart.

This interconnection enables unparalleled data analysis and system monitoring, paving the way for enhanced predictive maintenance, real-time optimization, and the development of more robust and resilient systems [2]. The interaction between digital twins and other cutting-edge technologies, such as big data analytics and artificial intelligence further propels the capacities of real-time monitoring, remote control, and data-driven decision-making [3].

These complex, interconnected, and dynamic real-time data processing systems typical of digital twins necessitate exceptional scalability, reliability, and adaptability to manage the rapidly changing data streams effectively [4]. Traditional monolithic architectures, where the entire application is tightly integrated and run as a single service, while once adequate, now face substantial challenges in meeting these demands. In these architectures, even a small change in the application necessitates redeploying the entire software, leading to challenges in scaling, maintenance, and reliability [5]. These challenges highlight a crucial need for an architectural shift to accommodate the growing complexity and real-time demands of modern digital systems [6].

Microservices architecture is a design pattern that breaks down a traditional monolithic system into smaller, self-contained services, each performing a specific business function [7]. Such decoupled systems facilitate seamless integration and adaptation to changing needs, ensuring long-term sustainability and evolution in line with technological advancements [6]. When combined with the scalable and fault-tolerant design of microservices, digital twins have the potential to change industries ranging from manufacturing to healthcare and from agriculture to city management, by offering robust and adaptable solutions for real-time data processing and analysis[8].

With their modular and decentralized design, microservices provide a viable

answer to the aforementioned difficulties. However, developing a scalable and reliable microservices-based digital twin system is not easy. Such a system necessitates careful planning, appropriate technology selection, and thorough implementation [9].

1.2 Research Approach

Designing and implementing a microservices-based digital twin solution is the main objective of this thesis. In order to achieve this general goal, we aim to address two important research questions:

RQ$_1$: What will be the benefits of microservices architecture to the implementation of digital twins for real-time data processing?

This question forms the foundation of our research, exploring the main benefits that microservices architecture can offer in enhancing the scalability and reliability of digital twin systems.

RQ$_2$: What are the foundational design principles and methodologies for constructing a robust microservices-based architectural framework for digital twins?

This question aims to clarify the fundamental design principles and methodologies that form the basis of an efficient architectural design, ensuring the successful deployment and utilization of digital twins in a variety of real-world scenarios.

RQ$_3$: What will be the main obstacles and trade-offs that we will face when designing and putting our microservices-based digital twin architecture into practice?

This question seeks to bring insight into the difficulties that digital twin developers may encounter so they can navigate and make well-informed judgments throughout the architectural design and implementation processes.

We will carefully examine these issues in order to understand their complexity and the trade-offs that different design choices involved. By doing this, we hope to make insightful contributions that can direct future efforts in the field of microservices-based digital twin solutions.

By employing a digital twin of a wind turbine as a test case, the study aims to glean practical insights and provide a comprehensive understanding of the microservices adoption in digital twin development for real-time data processing. By investigating these questions, we aim to shed light on the critical aspects of microservices architecture in the context of digital twin development and real-time data processing.

This document is organized according to the following structured sections:

1. **Background Concepts:** This section presents an exhaustive review of existing literature on digital twins, microservices, and their intersection, setting the research basis by providing context and background. Additionally, it includes a thorough examination of the project's fundamental concepts, ranging from the components of digital twins to the principles governing microservices.

2. **Microservices-based Design and Architecture:** Here, we delve into the blueprint of our solution. This section illuminates the architectural choices, design patterns adopted, and the rationale behind each decision.

3. **Microservices-based Implementation:** The metamorphosis of our design into a working solution. Every line of code, every deployed service, and the testing methodologies employed to ensure system robustness are discussed.

4. **Results and Discussion:** An empirical assessment of our system. We critically analyze its performance, scalability, and reliability, especially concerning real-time data processing.

5. **Conclusion and Future Work:** Reflecting on the journey, we summarize the project's achievements, and its implications, and chart out potential future directions.

In the following chapters, we will go through each of these components in detail, shining light on the complexities of creating a microservices-based digital twin solution.

CHAPTER

TWO

BACKGROUND CONCEPTS

This chapter explores the fundamental concepts needed to create a solid digital twin system, focusing on the microservices architecture in real-time data processing contexts. In order to clarify the importance of key software concepts like scalability and dependability in modern software solutions, we initiate our discussion by providing an overview of these concepts.

The reader is then introduced to the revolutionary world of microservices architecture. We examine its inherent properties, clarify the numerous advantages it offers over conventional monolithic constructions, and delve into its complex operation. We also discuss Docker containerization and its valuable impact on leveraging microservices architecture.

Next, we go on to the topic of digital twins as we end our discussion. These entities, which integrate the physical and digital worlds effortlessly, serve as examples of technical innovation. The symbiotic relationship between digital twins and microservices architecture becomes clear as we progress through this chapter, setting the scene for the following chapters in which we explore the practical side of developing and analyzing a digital twin solution supported by microservices.

2.1 Scalability and Reliability in Software Context

Scalability and reliability are two fundamental pillars in software engineering, especially in today's ever-evolving digital landscape.

2.1.1 Scalability

At its core, scalability refers to the ability of a system to handle an increase in workload without compromising on performance [10]. It is not just about "making things bigger" but ensuring that as the system grows, it can still deliver consistent, if not improved, levels of service.

There are two major types when speaking about scalability:

- **Vertical Scalability:** This involves increasing the capacity of a single server, usually by adding more resources like memory, storage, or faster CPUs [11]. While this method is straightforward, there is an upper limit to how much you can upgrade a single machine.

- **Horizontal Scalability:** This involves adding more servers to a system and distributing the workload among them. Modern distributed systems, cloud computing platforms, and databases often leverage this type of scalability [12].

Factors affecting scalability include database design, system architecture, software algorithms, and even organizational aspects.

2.1.2 Reliability

Reliability can be defined as the probability that a system will function without failure over a specified period under stated conditions [13]. In software terms, the system can operate consistently and as intended.

Several ways can be used to achieve reliability in software products:

- **Redundancy:** Employing backup components or systems that can take over during failures, ensuring system availability.
- **Failover Mechanisms:** These mechanisms detect system failures and switch to a backup or standby system, reducing system downtime [14].
- **Continuous Monitoring and Testing:** By constantly monitoring systems and rigorously testing new updates and features, one can identify and rectify points of failure [15].

2.1.3 The Interplay between Scalability and Reliability

One might think that by simply scaling a system (adding more machines or resources), it would inherently become more reliable. While there is some truth to this, it is a simplification. As systems scale, especially horizontally, they become more complex. More machines mean more potential points of failure. Hence, while scalability can contribute to reliability, it introduces new challenges that need addressing.

The explosion of the internet and mobile devices has made the challenge of scaling systems while maintaining (or even improving) their reliability even more acute. Websites and applications that were once serving thousands are now catering to millions, and sometimes even billions. The stakes have never been higher. A single hour of downtime can result in losses of significant revenues and can negatively affect a company's reputation [16].

2.2 Microservices Architecture

Microservices architecture is not just a trend but an impactful approach in modern software development paradigms [17]. This strategy promises faster software product releases by maximizing automation. It is a modern software development approach that revolutionizes the design and implementation of large-scale applications. It comprises a collection of small, autonomous services, each running in its own process and communicating with lightweight mechanisms such as HTTP/REST or message queues. This architectural style has gained significant

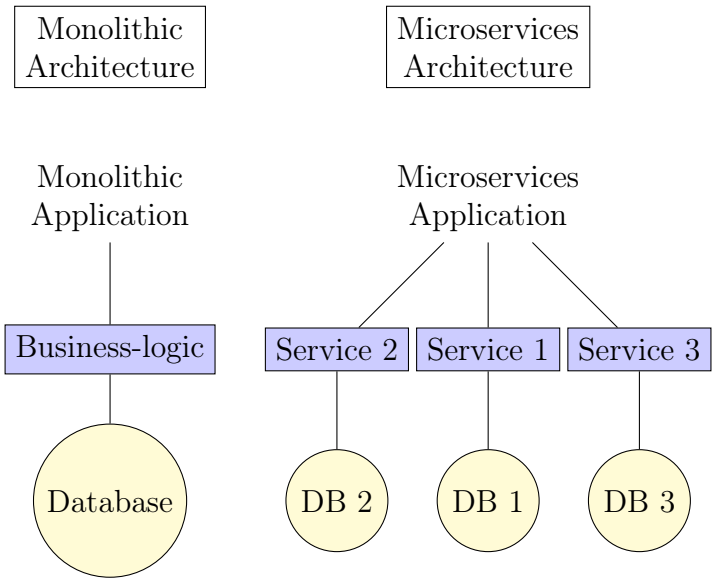

Figure 2.2.1: Comparison of a monolithic (left) and microservice architecture (right).

attention in recent years due to its ability to address the challenges of large-scale and complex systems by promoting modularity, scalability, and flexibility [7].

In contrast to traditional monolithic architectures, where the entire application is tightly coupled and deployed as a single unit, microservices architecture encourages the decomposition of monolithic applications into smaller, independent services. Each microservice is responsible for a specific business capability, such as user management, inventory management, or payment processing. These services can be developed, deployed, and scaled independently, allowing for faster development cycles and improved maintainability. Figure 2.2.1 illustrates the comparison between these two architectural styles.

Microservices architecture encourages the decomposition of monolithic applications into smaller, independent services, each responsible for a specific business capability. These services can be developed, deployed, and scaled independently, allowing for faster development cycles and improved maintainability. The loosely coupled nature of microservices enables teams to work on different services concurrently, facilitating faster innovation and reducing dependencies.

One of the key advantages of microservices architecture is its scalability. By distributing the system's functionalities across multiple services, each service can be scaled independently based on its specific needs. This flexibility allows organizations to handle varying workloads effectively and efficiently, ensuring optimal performance even under high demand.

Another benefit of microservices architecture is its ability to enhance fault tolerance and resilience. Since each microservice is isolated and operates independently, failures in one service do not necessarily impact the entire system. Services can be designed to be resilient by implementing retry mechanisms, circuit breakers, and fallback strategies, ensuring the overall system's availability and reliability.

2.2.1 Docker and Microservices: A Seamless Integration

A game-changer in application development and deployment is the Docker technology. Docker offers a consistent environment for apps wherever they are executed as an open-source containerization solution. This makes sure that differences between the settings used for development, testing, and production are kept to a minimum. Applications can be packed with all the components they require, including the libraries and other dependencies, because of docker's fundamental ability to isolate software within containers. This ensures that the program will execute consistently regardless of any external circumstances [18].

In addition, the microservices architecture breaks applications into more manageable parts that can function independently. Docker's capacity to containerize each of these services separately is what makes the combination of Docker and microservices so attractive [19]. Their combination makes it possible for each service to operate in a standardized environment and to be scaled, updated, or redeployed independently of the others. Many of the difficulties with dependency management and service orchestration in a microservices-based system are resolved by the usage of Docker containers [18].

Together, Docker and microservices have the potential to reshape the landscape of software development paradigms. By using those technologies, developers are presented with an efficient and streamlined workflow, wherein the technologies automate and optimize various development tasks, allowing developers to focus on writing code and developing applications, rather than being overwhelmed by deployment issues and related tasks. This combination leverages the modularity of microservices with the environmental consistency of Docker, paving the way for more robust, scalable, and maintainable software solutions.

2.3 Digital Twins

Digital twins have attracted substantial interest and traction recently, mostly because of their potential impact on a number of industries. The vision of digital twins has its roots in product lifecycle management (PLM) [20]. This vision emphasizes the use of a digital replica for every physical product to manage its data throughout its lifecycle. In the paper written by Glaessgen and Stargel in 2012 [21], a digital twin is described as an integrated multi-physics, multiscale, probabilistic simulation of a system or a vehicle that combines the best physical models currently available, sensor updates, fleet history, etc. to replicate the behavior of its corresponding flying twin.

Despite its PLM origins, the rise of Digital Twins is closely linked to the Internet of Things (IoT) advancements. The IoT emphasizes the connection between physical objects and their digital counterparts [22]. These digital counterparts allow physical entities to have context awareness, communicate, act, and exchange information. Saddik [23] defines digital twins with this perspective as a digital replica of a living or non-living physical entity. By bridging the physical and the virtual world, data is transmitted seamlessly allowing the virtual entity to exist simultaneously with the physical entity.

Digital Twins, from both PLM and IoT viewpoints, have been described in various ways. A commonly accepted notion is that a Digital Twin is a digital

representation of a system or asset, reflecting its real-world behavior [24, 25]. However, there is not a universally accepted definition yet.

A reliable and efficient architecture is essential for research using digital twins. In this context, the study conducted by Minerva et al. [2] stands out particularly. Their study provides a thorough exploration of the complex technological properties and structural paradigms inherent to digital twins. Their research provides invaluable advice by identifying and outlining the fundamental software principles that form the basis of Digital Twins in their entirety.

The Digital Twin concept substantially enhances our engagement with physical systems, enabling better understanding and optimization. While these systems often rely on simulation models, challenges arise when considering modular multi-domain systems like those in renewable energy. Moreover, with the continuous growth in the size of wind turbines and their increasing deployment offshore, the operation and maintenance of these turbines have become more challenging. The digital infrastructure, which is based on the Industry 4.0 concept, like the predictive digital twin, emerges as a key solution. Such technology not only facilitates data collection, visualization, and wind power analytics at both individual turbine and wind farm levels but also predicts potential failures of wind turbine components [26].

Digital twins inherently lean on simulation to replicate real-world scenarios and predict future behavior of their physical counterparts. A vital component of this simulation aspect can be ensuring interoperable interfaces and modular components for a precise and dynamic representation. One such interface is the Functional Mock-up Interface (FMI) [27] Standard. While its prominence is noted in the automotive industry, its potential in broader applications remains vast [28]. An innovative approach taps into this potential by constructing digital twins using individual Functional Mock-up Units (FMUs). These units come equipped with predefined model interfaces that enhance the digital twin's simulation capabilities across various industries and applications [26]. The evolution of such standards signifies the continuous convergence of simulation technologies and digital twin methodologies.

Various architectural solutions have been proposed in the literature for managing digital twins and their associated information sources. These solutions provide insights into how the different layers and components of a digital twin architecture can be organized. Some notable architectural solutions include:

- Architecture proposed in [29] for managing digital twins: This architecture consists of four layers: Information Providers, Model Providers, Digital Twin Providers, and Applications. It focuses on collecting data, processing information, creating and managing digital twins, and providing applications for accessing and manipulating the digital twins.

- Architecture presented in [24] for city digital twins: This architecture consists of five layers: Data Acquisition, Transmission, Digital Modeling, Data/-Model Integration, and Service. It emphasizes the integration of different sub-DTs, data acquisition and transmission, digital modeling, and providing various services to stakeholders. The structure of the proposed architecture in this paper is provided in Figure 2.3.1.

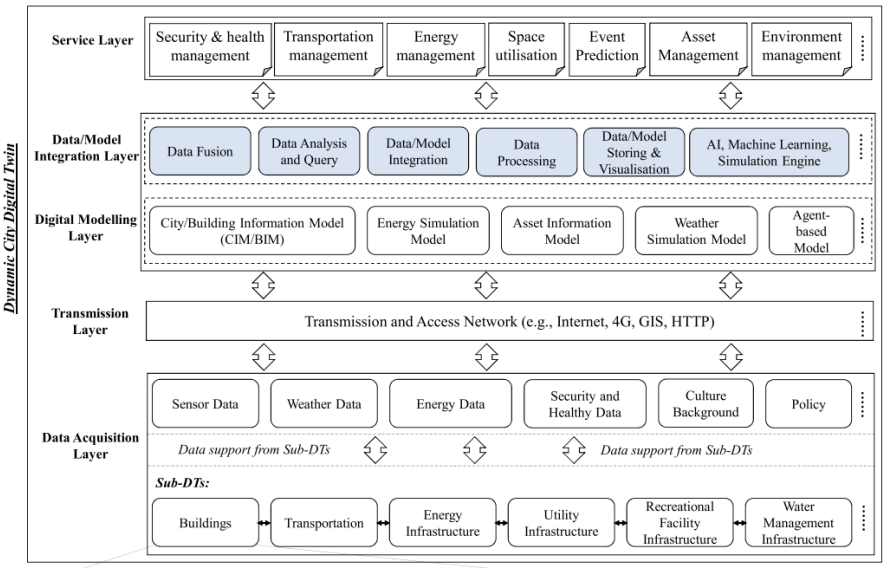

Figure 2.3.1: Five-layer DT architecture proposed in [24].

- Verdouw et al. [1] introduced an architecture for DTs that unfolds across nine designated layers. An overview of the proposed architecture is provided in Figure 2.3.2.

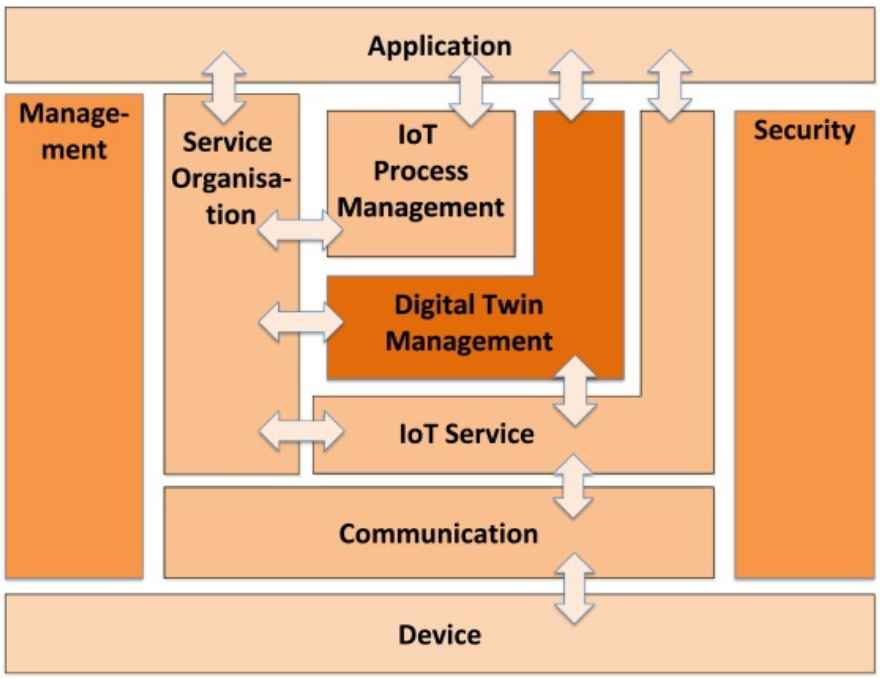

Figure 2.3.2: Overview of the nine-layer DT architecture proposed in [1].

In this architecture, the Device Layer is the foundation, embedding hardware components like sensors and actuators into physical objects for dynamic property measurements and remote operations. The Communication Layer assures seamless interactions and data transit between devices and IoT services, promoting end-to-end communication across diverse networking environments. The IoT Service Layer streamlines information retrieval and delivery, enhancing sensor and actuator control. The Digital Twin Manage-

ment Layer offers comprehensive insights into the Digital Twin information, ensuring efficient association and monitoring of physical objects. The IoT Process Management Layer enables the seamless deployment of IoT-aware processes, backed by the Service Organization Layer's orchestrated services. Security concerns are adeptly managed in the Security Layer, ensuring robust authorization, authentication, and identity management. The Management Layer oversees system configuration, fault reporting, and state determination, ensuring seamless operations. Lastly, the Application Layer emerges as the intelligence hub, empowering specific control tasks with diverse technologies, and ensuring effective interaction with Digital Twins across varied user interfaces.

- Architecture provided in [30] for managing a network of digital twins: This architecture consists of three layers: Device Layer, Cloud Platform Layer, and Application Layer. It focuses on managing physical objects with digital twins, providing a platform for managing the digital twins and their interactions, and offering applications for accessing and manipulating the digital twins.

These architectural solutions provide valuable insights into the organization and composition of digital twin systems, enabling effective management, integration, and utilization of digital twins.

CHAPTER

THREE

MICROSERVICES-BASED DESIGN AND ARCHITECTURE

3.1 Introduction

In this chapter, we focus on the layers of microservices-based design and architecture for digital twins. We commence our exploration by following a systematic process for designing the architecture of an application that handles requests. Next, we examine the common capabilities and specific use cases of digital twins. This section forms the groundwork upon which the rest of the chapter is built, painting a clear picture of the design methods of the architectural framework.

Transitioning from this approach, we proceed from our strategy to a wider perspective. The services and microservices that can be incorporated into a digital twin solution are examined in this section. Collectively, these sections offer a panoramic view, covering both the detailed complexities and the broader strokes of creating a robust microservices-based architectural framework for digital twins. As we navigate through the chapter, readers will gain a clear roadmap of the interconnections and dependencies among various design elements and architectural choices.

3.2 Systematic Design Process

This section describes the structured methodology we adopt for crafting a robust architectural design tailored for digital twin applications. At the heart of our approach lies the principle of addressing the specific challenges and requirements presented by digital twins, while simultaneously ensuring flexibility, scalability, and maintainability.

3.2.1 Business Requirements, System Functionalities, and Technical Constraints

In order to design an effective system, it is important to consider the business requirements, system functionalities, and technical constraints. The business requirements define the goals and objectives of the system, while the system functionalities describe the specific features and capabilities that the system should

provide. Additionally, the technical constraints outline the limitations and restrictions that need to be taken into account during the design process. By understanding and addressing these factors, the system design can be aligned with the needs of the business and ensure successful implementation.

3.2.2 Bounded Context

The concept of "Bounded Context" is a crucial aspect of domain-driven design (DDD) in software architecture [31]. A Bounded Context represents a conceptual boundary around a specific domain, where a distinct model and language are applied. Using Bounded Contexts facilitates handling complexity in large software systems by separating concerns and allowing each Bounded Context to have its own model and language without interfering with others [32].

In the context of microservices architecture, Bounded Contexts play a significant role in identifying the boundaries between services [7]. When designing a microservices architecture, architects need to identify the various business domains that the system will serve and define the Bounded Contexts for each domain. This ensures that each microservice is aligned with the needs of the business and focuses on a specific area of the organization. By defining Bounded Contexts based on business requirements and capabilities, complexity can be reduced, and the system becomes more adaptable to changing business requirements over time.

3.2.3 A Systematic Process for Designing the Architecture of an Application that Handles Requests

Designing the architecture of an application that handles requests requires a systematic process to ensure modularity, maintainability, and scalability. The following steps outline this process [7]:

1. **Identifying the system operations:** The first step involves identifying the key requests that the application needs to handle and abstracting them into system operations. These operations can be either commands, which update data, or queries, which retrieve data. An abstract domain model can be used to define the behavior of each command, providing a clear understanding of the system's functionalities.

2. **Decomposing the application into services:** The second step is to decompose the application into services. There are different strategies for achieving this, such as organizing services around business capabilities or domain-driven design subdomains. The goal is to create services that are organized around business concepts rather than technical concepts, promoting better maintainability and reusability.

3. **Determining each service's API:** The third step involves determining each service's API, including assigning system operations to services and defining how services collaborate. This may require additional operations to be defined to facilitate communication between services. It is recommended to use appropriate inter-process communication (IPC) mechanisms to implement each service's API, ensuring seamless interaction between services.

By following this systematic process, the architecture of the application can be designed in a modular and scalable manner, making it easier to manage and adapt to changing business requirements.

3.2.3.1 Identifying the System Operations (An Example)

Creating a high-level domain model for the application is the initial step in identifying the system operations. Although each service has its own domain model, establishing a high-level domain model helps describe how the system activities behave. Standard methods, such as consulting domain experts and examining the nouns in stories and situations, can be employed to construct the domain model.

For instance, in the context of a general digital twin solution, a simple story like "Real-time monitoring of a physical system" can be expanded into a scenario. This scenario involves continuously collecting sensor data from the physical system, analyzing the data in real-time, and generating insights or alerts based on the analysis.

Algorithm 1 outlines an example of how the "Real-time monitoring of a physical system" scenario could be expanded:

Algorithm 1 Real-time monitoring of a physical system

Given: a production system
And: a set of sensors measuring key parameters of the system
And: a system to collect and store data from the sensors
And: a dashboard to display the collected data
When: the system is in operation
Then: the sensors continuously collect data
And: the data is stored in the data collection system
And: the dashboard displays the real-time status of the system
And: the dashboard sends alerts to maintenance personnel when the system's status reaches predefined thresholds
And: the data can be analyzed to identify trends and predict maintenance needs

The nouns in the scenario can hint at the existence of various classes, such as a physical system, sensor, data point, monitoring system, alert, and notification. These classes represent the entities and concepts within the system and help in defining the system operations effectively.

3.2.4 Common Capabilities and Use Cases

In order to expand the systematic design approach discussed in the preceding section, it is essential to dive into further detail about the practical considerations that support our architectural decisions. The discovery of common capabilities related to robust software systems and the clarification of essential use cases that guide our design are fundamental components of our methodology. This guarantees that our design is not only conceptually sound but also practically orientated and in line with requirements found in the real world.

"Common Capabilities" are the fundamental features that every system must have, in terms of software architecture. These features include integration, scala-

bility, and security, and they guarantee the software's reliable and effective operation [33]. On the other hand, "Use Cases" are detailed accounts of how a system interacts with external entities to accomplish particular goals. They outline the series of steps and requirements needed for the program to carry out a task, providing a clear picture of the system's behavior and ensuring that it adheres to the functionality and goals that were intended. The foundation for creating software architectures that are robust, effective, and in line with stakeholder and user expectations is formed by shared capabilities and use cases.

In the context of digital twins, "Common Capabilities" refer to the general functionalities that digital twin systems provide irrespective of the specific use case or application. These capabilities typically include real-time data collection and analysis, simulation, visualization, and integration, among others. These functionalities form the backbone of digital twin systems, enabling them to replicate, predict, and analyze the behavior and performance of physical entities in a virtual environment.

On the other hand, "Use Cases" pertain to the practical and specific applications of digital twin technology in various domains and contexts. Use cases illustrate how the common capabilities of digital twins can be leveraged to solve real-world problems, optimize systems, and improve decision-making in diverse. More clarification can be obtained by taking a look at some examples.

As an example, in [26], a predictive digital twin is developed for wind farms, and the following capabilities and use cases can be obtained from it:

- **Predictive maintenance:** The ability to predict failures of wind turbine components, enabling proactive maintenance activities and reducing potential failures.
- **Real-time data collection and analysis:** The platform allows users to collect, visualize, and analyze data in real time, improving predictive capabilities and enabling better decision-making.
- **Improved reliability:** By providing predictive maintenance, the platform can improve the reliability of wind turbines and wind farms.
- **Visualization:** The platform offers different result presentations through 2D and 3D visualization, and augmented reality, which can be chosen depending on the desired objectives and requirements.
- **Integration:** The platform is developed based on the OPC-UA, making it easy to adopt and integrate directly into energy companies' existing systems.
- **Teaching and research:** The platform is also used in teaching and research at the Department of ICT and Natural Sciences NTNU, which can be a capability of the system.

For a city digital twin as in [34], potential capabilities and use cases could include data management, visualization, situational awareness, planning and prediction, and integration and collaboration. These capabilities enable stakeholders to efficiently and securely manage, analyze, and visualize data, monitor and respond to real-time events, simulate and forecast the impact of different scenarios

and interventions, and collaborate with various stakeholders and data sources to support effective decision-making and planning.

According to the examples, these common capabilities can be obtained for digital twin solutions:

- **Simulation capabilities:** the ability to predict failures or simulate scenarios to enable proactive maintenance and better decision-making.
- **Real-time data collection and analysis:** the ability to collect, process, and analyze real-time data streams from various sources to improve situational awareness and predictive capabilities.
- **Visualization:** the ability to create and display 2D or 3D models of the system or environment to improve understanding and decision-making.
- **Integration:** the ability to integrate and collaborate with various stakeholders, data sources, and software tools to support effective decision-making and planning.

3.3 Defining Microservices for Current Architectural Framework

Based on the identified capabilities and use cases, a range of services can be considered for a general digital twin solution. These services can be organized into a microservices architecture, allowing for modularity, scalability, and flexibility. Some possible services for a digital twin solution include:

- Data Acquisition: Collecting data from physical devices and sensors in real-time.
- Data Integration: Integrating data from multiple sources into a unified view of the system.
- Analytics: Analyzing data to identify patterns, trends, and anomalies.
- Simulation: Creating a virtual representation of the physical system and running simulations to predict performance.
- Visualization: Providing a graphical representation of the system and its data to aid in decision-making.
- Control: Using the virtual model to control the physical system in real time.
- Collaboration: Allowing multiple stakeholders to access and interact with the digital twin solution.
- Security: Ensuring the solution is secure and protecting sensitive data and intellectual property.

In addition to these services, other candidate microservices can be considered based on the specific requirements and domain of the digital twin solution. These include configuration management, event management, workflow management, integration with enterprise systems, knowledge management, optimization, predictive quality, and resource management.

In our digital twin solution, seven microservices are taken into account based on different scenarios and use cases, which are presented in Table 3.3.1.

Table 3.3.1: Microservice description.

Microservice	**Description**
Data Acquisition and Streaming	This service handles real-time and batch data acquisition from various sources such as sensors, devices, and databases, and streams it to the data processing layer.
Data Processing and Analytics	This service processes and analyzes the incoming data in real-time and in batch mode using various techniques such as machine learning, deep learning, and statistical analysis.
Digital Twin Modeling	This service creates and maintains the digital twin model of the physical system using data from various sources, and updates it in real-time as new data becomes available.
Simulation	This service utilizes the digital twin model and runs simulations to predict performance.
Command and Control	This service enables the communication between the digital twin and the physical system, allowing the digital twin to send commands and receive feedback.
Visualization and Reporting	This service provides real-time and historical visualizations of the data, including dashboards, reports, and alerts.
Security and Access Control	This service manages the security of the microservices architecture, including authentication, authorization, and encryption of data.

3.4 Conclusion

Digital twins offer a range of capabilities and use cases, enabling organizations to gain valuable insights, improve decision-making, and enhance system performance. By adopting a microservices architecture, digital twin solutions can be developed with modularity, scalability, and flexibility in mind. The identified services and architectural solutions provide a foundation for designing and implementing effective digital twin systems.

In this chapter, we have explored the common capabilities and specific use cases of digital twins, discussed the services and microservices that can be incorporated into a digital twin solution, and examined architectural solutions proposed in the literature. By leveraging the power of digital twins and microservices, organizations can unlock new opportunities and achieve better outcomes in their respective domains.

CHAPTER

FOUR

MICROSERVICES-BASED IMPLEMENTATION

4.1 Introduction

This chapter explores the practical realization of the conceptual architecture previously covered. The main objective is to give an insightful account of the technical procedures and approaches used to translate the theoretical constructs into a concrete, practical reality.

This chapter focuses on the seamless integration of microservices, Docker containerization, Kubernetes orchestration, and the use of RabbitMQ as a message broker for inter-service communication. Microservices were chosen as the architectural base because of their inherent scalability and adaptability, which correspond well with the dynamic requirements of real-time digital twin systems. By assuring portability, resource optimization, and effective management of microservice installations, Docker and Kubernetes further improve implementation. The use of RabbitMQ, a powerful and well-known message broker, enables the smooth interchange of information across microservices and adds to the digital twin ecosystem's real-time nature.

The details of the deployment procedure, the complex nature of microservice relationships, and the orchestration done by Kubernetes are revealed in the next sections. Through the journey outlined in this chapter, the study aims to demonstrate the technical aspects required for the successful implementation of the proposed microservice-based solution.

4.2 Architectural Overview

As discussed in the previous chapter, the microservices architecture designed for the real-time digital twin solution serves as the foundation for the subsequent implementation and testing phases. To provide context for the following sections, a brief overview of the architecture and design is reiterated here, with emphasis on the aspects most relevant to implementation and testing.

The microservices architecture is structured to encapsulate specific functionalities within individual services, enabling modularity, scalability, and parallel development. Each microservice is designed to fulfill a distinct role within the digital twin ecosystem, facilitating real-time data synchronization, analysis, and

visualization. This architecture promotes flexibility and adaptability, crucial for accommodating evolving requirements and diverse use cases.

Containerization, executed through Docker, enhances the portability of microservices across various environments, minimizing the "it works on my machine" dilemma and ensuring consistency between development, testing, and production environments. As an orchestration platform, Kubernetes manages the deployment, scaling, and management of microservices, promoting smooth operation and efficient resource use.

Additionally, RabbitMQ, the chosen message broker, plays a pivotal role in enabling communication between microservices in an asynchronous and decoupled manner. This facilitates the exchange of data and commands crucial for real-time synchronization and collaboration within the digital twin ecosystem.

The following sections cover the implementation details of the designated microservices with this architectural base in mind.

4.3 Implementation Details

In this section, we examine in depth the microservices that were chosen to be used in our real-time digital twin system. Initially, our architecture plan included a group of seven potential microservices, each of which addressed to a different functional need in the context of the digital twin environment. However, after careful thought and deliberation, we determined that five of these seven should be implemented first due to their crucial necessity and the project's limitations.

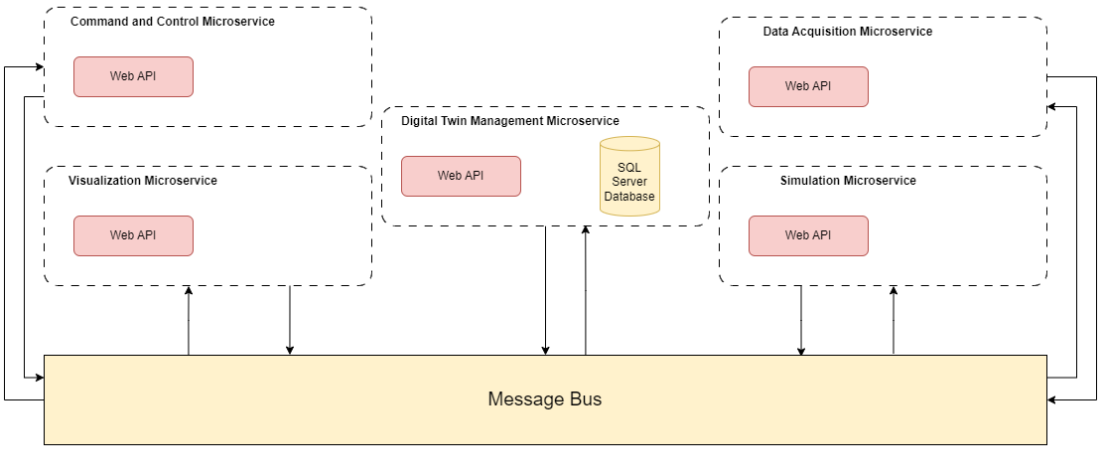

Figure 4.3.1: Diagram depicting the microservices implemented in the solution and the connections to the message bus (RabbitMQ).

Figure 4.3.1 shows the diagram illustrating these microservices and the message bus. The selected microservices for the current phase are Data Acquisition, Digital Twin Management, Simulation, Command and Control, and Visualization. Each of these was chosen because of their pivotal role within the digital twin ecosystem. The Data Acquisition service acts as the gateway, bridging the digital twin with real-world data, thus ensuring seamless synchronization. Digital

Twin Management forms the structural spine of the system, managing and maintaining the digital replica of the physical counterpart. The Simulation service is vital, providing insights into system performance and enabling predictive analyses. Command and Control orchestrates the integral interactions between the digital replica and its physical counterpart. Finally, the Visualization microservice stands as the user's window to the system, allowing real-time insights and interactive engagements with the digital twin.

Weighing their immediate relevance and the concrete impact they could give within our time constraint, we chose these five from the original seven. As the project moves forward, other designated microservices can be developed and added to the solution.

4.3.1 Data Acquisition Microservice

The Data Acquisition Microservice serves as the entry point to our real-time digital twin solution. It is responsible for collecting and aggregating data from different sources, such as sensors, devices, and databases. The decision to prioritize the development of this microservice is grounded in its fundamental importance. Data acquisition forms the basis upon which the entire digital twin ecosystem relies. By ingesting both real-time and batch data, this microservice lays the foundation for subsequent processing, analysis, and simulation. Figure 4.3.2 shows a diagram illustrating how this microservice is utilized in the solution. As shown in this figure, data is perceived from the sensors by this microservice and a message as an event is created to be published to the message bus regarding the received data.

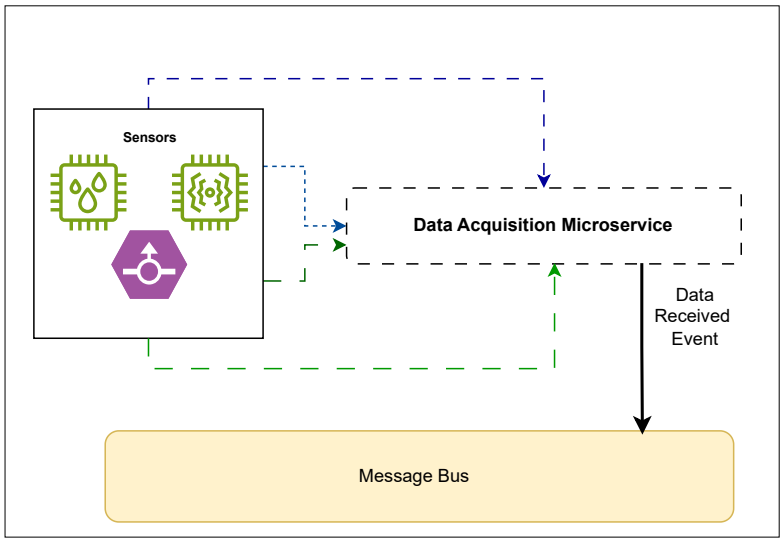

Figure 4.3.2: A simple diagram illustrating how Data Acquisition Microservice is utilized in the solution

Our implementation of the Data Acquisition Microservice is executed using .Net 6, with a specific focus on utilizing C# for its robust capabilities. .NET 6 offers enhanced support for asynchronous programming, allowing the microservice to handle multiple data streams concurrently while maintaining responsiveness. Leveraging the latest features of C#, the microservice effectively interacts with diverse data sources, ensuring efficient and reliable data ingestion. A link to

the Github repository of the implementation of this microservice is provided in Appendix A.

4.3.2 Digital Twin Management Microservice

The Digital Twin Management Microservice holds a pivotal role in our real-time digital twin solution. This microservice is responsible for creating and maintaining a virtual representation of the physical system. By harnessing the power of .NET 6 and C#, we have designed a robust system that encompasses the creation of digital twin projects based on distinct types and the seamless storage of project information within an SQL Server database.

4.3.2.1 Project Creation and Database Storage

In the context of the Digital Twin Management Microservice, projects serve as the building blocks of our virtual representation. Projects are created based on specific types that define the nature and scope of the digital twin. Leveraging the extensibility of .NET and the expressive capabilities of C#, we have implemented a project creation mechanism that allows stakeholders to define the purpose and attributes of each project type.

The Digital Twin Management Microservice employs the Code-First approach to database design, which aligns well with the agile nature of microservices development. As part of this approach, our microservice's C# models mirror the structure of the SQL Server database. Through a carefully crafted set of C# classes and data annotations, we define the attributes of each project type. These annotations guide the Code-First process to generate corresponding database tables and relationships, ensuring a seamless translation between our application's object model and the database schema. Figure 4.3.3 depicts how project and digital twin creation is applied in Digital Twin Management Microservice. Projects and digital twin instances are created in the user interface shown in the figure and using a Rest API is transmitted to the SQL Server database.

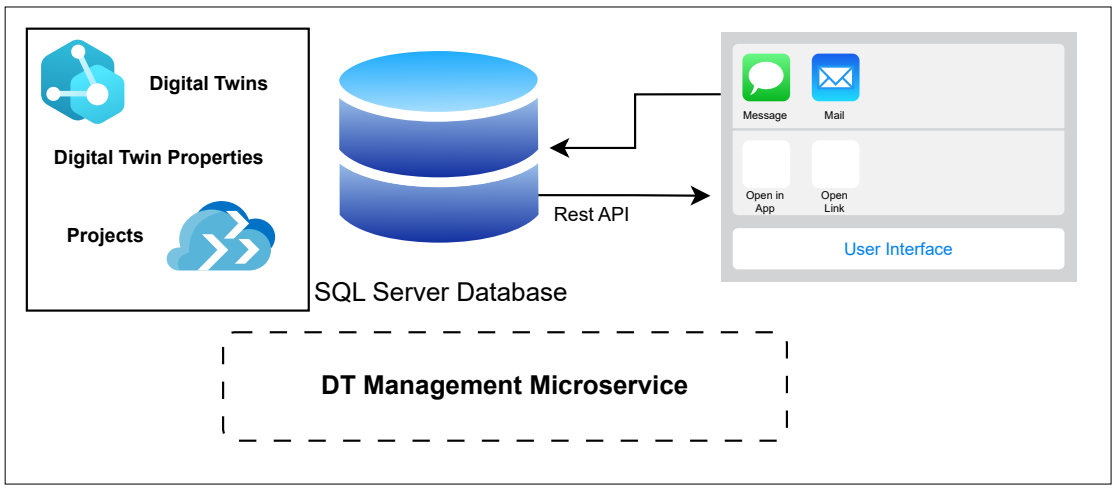

Figure 4.3.3: Diagram illustrating how project and digital twin creation is applied in Digital Twin Management Microservice.

Upon project creation, the microservice captures essential metadata, such as project type, creation timestamp, and owner details. This information is then seamlessly persisted into the SQL Server database, preserving project integrity and establishing a robust foundation for subsequent real-time updates. The use of the Code-First approach simplifies database schema evolution, allowing us to iteratively enhance project attributes and relationships without compromising data integrity.

4.3.2.2 SQL Server and Database Interaction

Our choice of SQL Server as the database management system is rooted in its reliability, scalability, and seamless integration with the .NET ecosystem. The Digital Twin Management Microservice interacts with the SQL Server database using Entity Framework Core, a modern and extensible ORM framework provided by the .NET 6 ecosystem. Entity Framework Core simplifies data access and manipulation by providing a high-level abstraction over database interactions.

With Entity Framework Core, we define a DbContext that encapsulates the interactions between our microservice and the SQL Server database. This DbContext enables seamless querying, insertion, updating, and deletion of project records. The DbContext also supports LINQ queries, allowing us to retrieve project data using expressive and readable code. The integration of Entity Framework Core ensures that our microservice's logic remains focused on business processes, while the underlying data access complexities are abstracted.

4.3.3 Simulation Microservice

The Simulation Microservice holds a critical role within the real-time digital twin solution, employing the digital twin model to predict the performance of the physical system. Our focus in this phase is on laying the foundation for future simulation enhancements, including the integration of different simulators and the utilization of Functional Mock-up Interface (FMI) standards for improved extensibility and accuracy.

4.3.3.1 Foundational Simulation and Static Simulator

The Simulation Microservice initiates predictive simulations to provide stakeholders with insights into potential future behaviors of the physical system, based on real-time data from the digital twin. Our initial approach involves the incorporation of a static simulator. This simulator employs foundational simulation techniques, such as mathematical models and heuristic algorithms, to approximate system trajectories. While this approach may not capture intricate complexities, it serves as a stepping stone for more sophisticated simulation methodologies.

Through the use of .NET and c#, we have implemented the static simulator within the microservice. The simulator interprets data from the digital twin and generates predictions that offer valuable insights into system performance. The foundational simulator provides stakeholders with an initial understanding of the system's behavior and aids in decision-making.

4.3.3.2 Extensibility through FMI/FMU Integration

To establish a robust foundation for future enhancements, the Simulation Microservice architecture is designed to integrate various simulators using the FMI standard. FMI is an open standard for model exchange and co-simulation that allows diverse simulation tools to interact seamlessly. By adhering to FMI standards, we ensure compatibility with different simulation engines and facilitate the integration of domain-specific simulators.

Our microservice's extensibility is demonstrated by the integration of FMI/FMU compatibility. This architectural decision empowers future iterations to incorporate advanced simulation tools, including domain-specific simulators and commercial simulation software. Through the use of standardized interfaces, the Simulation Microservice can dynamically load and execute different simulation engines, enhancing accuracy and enabling sophisticated predictive capabilities.

4.3.4 Command and Control Microservice

The Command and Control Microservice serves as a critical communication bridge between the digital twin and the physical system. While the initial design emphasized synchronization, the complexity of integrating with actual hardware and compatibility considerations led us to focus on building a foundational framework.

4.3.4.1 Foundational Communication Framework

This Microservice facilitates bidirectional communication between the digital twin and simulated devices, allowing dynamic adjustments and real-time control. The microservice's foundational communication framework is designed to accommodate a range of devices, systems, and protocols. It establishes lightweight communication protocols that enable seamless interaction while considering extensibility for future integration.

Our implementation of the Command and Control Microservice utilizes .NET and C#. This enables us to create communication channels that adhere to industry standards and best practices, ensuring reliability and security. The microservice's modular design promotes maintainability, allowing for the addition of custom communication protocols as needed.

4.3.4.2 Integration with Digital Twin

To enable real-time control and adjustments, the Command and Control Microservice integrates tightly with the Digital Twin Management Microservice. The digital twin provides the necessary context and insights for informed decision-making, while the Command and Control Microservice enables the translation of commands into actionable changes within the physical system.

Through well-defined APIs and message formats, the microservices seamlessly exchange information. The Command and Control Microservice can receive commands from the digital twin and translate them into appropriate actions, simulating the impact of these actions on the digital twin and physical system. This integration empowers stakeholders with the ability to interact with the digital twin, make informed adjustments, and observe their effects.

4.3.5 Data Visualization Microservice

The Data Visualization Microservice plays a crucial role in our real-time digital twin solution, providing stakeholders with meaningful insights through interactive visualizations. While the original design aimed for comprehensive dashboards and reports, the current focus is on core visualization functionalities, with an emphasis on user engagement and accessibility.

4.3.5.1 Interactive Visualization Interfaces

The user's capacity to engage with, analyze, and comprehend complicated data sets and trends is improved by the Data Visualization Microservice, which makes use of a variety of technologies to produce dynamic and interactive visualization interfaces. These technologies include:

- **HTML (HyperText Markup Language)** is used for creating the structured content for the web interfaces. It forms the backbone, providing the basic structure and layout for the visual elements [35].
- **CSS (Cascading Style Sheets)** is deployed to enhance the presentation of the HTML elements, offering stylization and design improvements that make the interface more visually appealing and user-friendly [36].
- **Angular** is a renowned front-end web framework, instrumental in developing dynamic dashboards, real-time charts, and interactive components. It allows for the seamless creation and management of complex front-end logic, ensuring that the data is presented in a clear, concise, and interactive manner [37].

These technologies together lay the foundation for robust data visualization capabilities, making it easier for stakeholders to engage with and understand the data presented to them. Moreover, to elevate the data visualization experience further, Unity 3D [38] is integrated for enhanced 3D visualization purposes. Unity 3D, a powerful cross-platform game engine, empowers the creation of immersive and interactive 3D visualizations, offering stakeholders a more engaging and in-depth view of the data.

Our microservice's implementation integrates seamlessly with the rest of the ecosystem, receiving real-time and historical data streams from other microservices. Through APIs and well-defined message formats, the Data Visualization Microservice ensures that visualizations are synchronized with the most up-to-date information from the digital twin and simulation microservices.

4.3.5.2 Real-time and Historical Insights

This Microservice provides stakeholders with both real-time and historical insights into the behavior of the physical system. Real-time visualizations offer immediate feedback on system status and changes, enabling stakeholders to make informed decisions in real-time. Historical visualizations, on the other hand, allow stakeholders to analyze trends and patterns over time, facilitating data-driven planning and optimization. Figure 4.3.4 illustrates different technologies utilized when implementing Visualization Microservice.

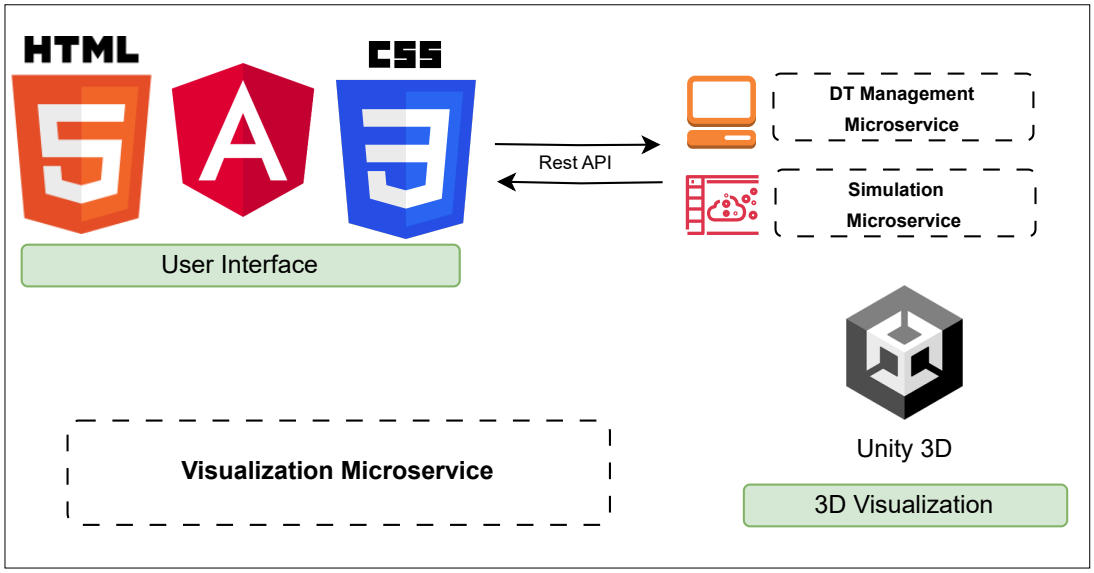

Figure 4.3.4: A diagram depicting different technologies utilized when implementing Visualization Microservice.

By leveraging HTML and CSS for layout and styling, and Angular for dynamic content, the microservice delivers a rich and responsive user experience. The visualizations are designed to be intuitive, allowing stakeholders to interact with data, customize views, and gain actionable insights without requiring extensive technical expertise.

4.3.5.3 Customizable Dashboards and Reports

While the initial implementation focuses on core visualization capabilities, the architecture of the Data Visualization Microservice is designed for extensibility. Future iterations can expand on this foundation to include customizable dashboards, detailed reports, and additional visualization types. This modularity ensures that the microservice can evolve to meet the evolving needs of stakeholders and adapt to new visualization requirements.

4.3.6 Future Microservices Implementation

While our current focus has been on the implementation of the Data Acquisition, Digital Twin Management, Simulation, Command and Control, and Data Visualization microservices, the architectural design of the real-time digital twin solution encompasses a broader spectrum of functionalities. Several additional microservices are envisioned to complete the comprehensive ecosystem. Due to various factors, including resource limitations and the complexity of advanced algorithms, these microservices are planned for future implementation.

4.3.6.1 Data Processing and Analytics

The Data Processing and Analytics microservice will be responsible for processing and analyzing incoming data in real-time and batch modes. Machine learning, deep learning, and statistical analysis techniques will be employed to extract

insights, identify trends, and predict system behaviors. This microservice will play a crucial role in converting raw data into actionable information, supporting decision-making and optimization strategies.

4.3.6.2 Security and Access Control

The Security and Access Control microservice will be responsible for managing the security aspects of the microservices architecture. This includes user authentication, authorization mechanisms, encryption of data at rest and in transit, and auditing. As the solution grows, securing data and maintaining privacy will become increasingly important, and this microservice will ensure compliance with industry standards and best practices.

4.3.6.3 Other Future Enhancements

In addition to these planned microservices, other enhancements are anticipated. These could include fine-tuning and optimizing existing microservices, expanding integrations with external systems and APIs, and incorporating advanced visualization techniques to provide richer insights. Furthermore, as new technologies emerge and capabilities evolve, the microservices architecture will remain adaptable, enabling the integration of cutting-edge tools and algorithms.

The phased approach to microservices implementation allows us to prioritize the most critical functionalities while building a solid foundation for future expansion. By taking a modular approach, our solution remains flexible, accommodating changes in requirements, technology advancements, and stakeholder needs. In the subsequent sections, we will delve into the comprehensive testing methodologies employed to validate the reliability, scalability, and performance of the implemented microservices and the broader digital twin ecosystem.

4.4 Integration with RabbitMQ Message Broker

The seamless communication and coordination of microservices within the real-time digital twin solution are facilitated by the integration with RabbitMQ [39], a powerful message broker. RabbitMQ plays a crucial role in ensuring efficient data exchange, reliable message delivery, and asynchronous communication among the microservices, contributing to the robustness and scalability of the entire ecosystem.

4.4.1 Role of RabbitMQ in the Architecture

RabbitMQ serves as a central hub that facilitates communication between microservices in a decoupled and asynchronous manner. It acts as an intermediary, receiving messages from producer microservices and delivering them to consumer microservices. This decoupling eliminates direct dependencies between microservices, allowing them to operate independently and evolve without affecting one another.

4.4.2 Message Exchange Mechanism

Messages are exchanged between microservices through RabbitMQ using the publish-subscribe pattern. When a microservice produces a message, it publishes it to a specific exchange. Consumer microservices interested in certain types of messages bind to the exchange and receive messages as they are published. This mechanism ensures that each microservice only processes the messages relevant to its functionality, maintaining a clear separation of concerns.

The message content typically includes information needed for coordination, data synchronization, or triggering specific actions. This includes updates from the Data Acquisition Microservice, commands from the Command and Control Microservice, simulation results from the Simulation Microservice, and more. RabbitMQ ensures that these messages are reliably delivered to the intended consumers, even in scenarios of high traffic or temporary service unavailability.

4.4.3 Benefits of Using RabbitMQ as Message Broker

The integration with RabbitMQ offers several significant benefits within the context of our real-time digital twin solution:

- Asynchronous Communication: RabbitMQ enables asynchronous communication between microservices. This decoupling prevents bottlenecks and enhances the responsiveness of the entire system. Microservices can continue processing requests and messages without waiting for immediate responses.
- Reliability: RabbitMQ ensures reliable message delivery through its message queuing mechanisms. Messages are stored in queues until they are successfully consumed, preventing data loss in case of temporary failures or microservice unavailability.
- Scalability: As the solution scales, RabbitMQ supports load distribution and efficient message routing. Microservices can be replicated and distributed across multiple instances without affecting message exchange patterns.
- Flexibility: RabbitMQ supports various messaging patterns, such as publish-subscribe, request-response, and work queues. This flexibility allows microservices to communicate in ways that best suit their communication needs.
- Fault Tolerance: RabbitMQ provides mechanisms for clustering and high availability, minimizing the risk of single points of failure. This ensures continued communication even if some instances experience issues.
- Message Prioritization: RabbitMQ allows prioritization of messages, enabling critical messages to be processed ahead of others. This is particularly useful when dealing with time-sensitive information or urgent commands.

In summary, the integration with RabbitMQ enhances the real-time digital twin solution's communication, coordination, and scalability. By adopting asynchronous messaging patterns, we ensure efficient data exchange while maintaining reliability and flexibility. As we proceed to the subsequent sections, we will delve

into the comprehensive testing methodologies employed to validate the effectiveness and reliability of the integrated architecture, including RabbitMQ's role in ensuring seamless microservices communication.

4.5 Deployment with Kubernetes

The deployment of microservices within the real-time digital twin solution is orchestrated using Kubernetes [40], a powerful container orchestration platform. Kubernetes streamlines the deployment, scaling, and management of containerized applications, enhancing the solution's reliability, scalability, and ease of maintenance.

Kubernetes, an open-source platform for orchestrating containers, plays a pivotal role in the deployment and management of microservices within our real-time digital twin solution. This section delves into the key Kubernetes concepts and explains how different microservices can be represented as various Kubernetes objects. Kubernetes efficiently manages the lifecycle and networking of containerized applications, providing automatic recovery and scaling. It abstracts away the complexities of managing individual containers, ensuring that applications remain responsive and reliable. When considering the adoption of Kubernetes, it is crucial to assess whether the benefits align with your architectural needs.

For monolithic applications, Kubernetes might not immediately provide significant value. However, as we transition towards microservices, the need for service orchestration, including connection management and scalability, becomes more pronounced [6].

4.5.1 Kubernetes Objects and Microservices

This section explores how Kubernetes and microservices might work together. To deploy, scale, and manage applications, Kubernetes offers a complete collection of primitives, also known as objects. Understanding how Kubernetes deployments, pods, services, and ingress work will help us build an architecture that makes the most of both technologies and will produce a system that is reliable and fast. The details of each of these important Kubernetes objects and how they interact with the microservices in our digital twin solution are explained in the following subsections.

4.5.1.1 Deployments for Scalability and Reliability

Kubernetes Deployments serve as blueprints for maintaining the desired state of microservice pods. By creating multiple replicas of a pod and distributing them across nodes, Deployments enhance both performance and reliability. If a pod crashes, other replicas can seamlessly handle incoming requests, ensuring uninterrupted service availability.

In our real-time digital twin solution, each microservice, such as the Data Acquisition Microservice or the Simulation Microservice, can be encapsulated within a Deployment. This guarantees a specified number of replicas are always running, optimizing performance and enabling graceful recovery from failures.

4.5.1.2 Pods for Microservice Isolation

Pods, the smallest deployable units in Kubernetes, encapsulate one or more containers within a shared environment. For microservices, pods ensure isolation, resource allocation, and a consistent execution environment. Each pod can host a single microservice or multiple related microservices. In our architecture, we can encapsulate microservices like the Data Acquisition Microservice or the Visualization Microservice within separate pods. This isolation prevents conflicts and resource contention while providing a dedicated execution environment.

4.5.1.3 Services for Communication

Kubernetes Services provide a stable endpoint for accessing microservices. They enable load balancing, service discovery, and reliable communication between microservices. Services abstract away the complexities of individual pod IP addresses and provide a consistent entry point. For instance, the Data Acquisition Microservice and the Simulation Microservice can be exposed as Kubernetes Services. This ensures other microservices can interact with them through stable endpoints, regardless of the underlying pod distribution.

4.5.1.4 Ingress for External Access

Kubernetes Ingress acts as an entry point for external traffic into the cluster. It can be configured as an API gateway, routing requests to different microservices based on specific rules. Ingress supports features like SSL termination, load balancing, and path-based routing. In our solution, the Ingress resource can provide external access to microservices, including the Data Visualization Microservice. By defining routing rules, we can ensure that incoming requests are directed to the appropriate microservice based on the requested URLs or paths.

4.5.2 Performance and Reliability Gains

The inherent nature of Kubernetes orchestrations, such as Deployments, contributes significantly to improved performance and reliability. Deployments facilitate scaling, distributing load across multiple replicas of a pod, while also ensuring the availability of backups if any pod crashes. This translates to enhanced responsiveness, minimized downtime, and efficient resource utilization. In summary, Kubernetes integration empowers our real-time digital twin solution with orchestrated microservices deployment, communication, and scalability. These concepts bolster the solution's performance, reliability, and ease of management, ultimately aligning with our architectural goals.

4.5.3 Containerization with Docker

Before diving into the Kubernetes deployment process, it is important to highlight the role of Docker in containerization. Docker allows us to package microservices and their dependencies into isolated containers, ensuring consistent environments across development, testing, and production stages. Containers encapsulate the

microservices, including their runtime, libraries, and settings, providing a reliable and portable execution environment.

Each microservice in the real-time digital twin solution is containerized using Docker images. These images are created based on predefined Dockerfiles, specifying the necessary configuration and dependencies. By containerizing microservices, we eliminate potential conflicts between application components and ensure seamless deployment across different environments.

4.5.4 Kubernetes Configurations

The deployment and management of microservices in Kubernetes involve creating configuration files that define the desired state of resources. These configurations are written in YAML (YAML Ain't Markup Language) format, a human-readable data serialization standard that can be used for all programming languages. It is specifically designed to be easy to read and write due to its clear visual presentation and support for complex data structures. In the context of Kubernetes, YAML files specify complex details, such as pod definitions, services, resource constraints, and environment variables. This structured format allows administrators and developers to declare configurations and automate the deployment process, ensuring consistent and reproducible results across various environments and infrastructures.

Kubernetes configurations include pod templates, service specifications, and deployment strategies. These configurations enable us to automate the deployment process and ensure consistency across different environments, from development to production. As part of our deployment process, we have developed a set of Kubernetes configuration files that define the deployment, services, and scaling behavior of each microservice. These configurations ensure that the microservices are deployed consistently, are accessible through services, and can be scaled efficiently.

The integration of Kubernetes and Docker revolutionizes the deployment and management of microservices within the real-time digital twin solution. By containerizing microservices and utilizing Kubernetes abstractions, we achieve enhanced scalability, availability, and ease of management.

4.6 Conclusion

In conclusion, this chapter's journey provided a comprehensive perspective on the process of moving from a hypothetical microservices-based digital twin architecture to its practical manifestation. This project's foundation has evolved as the interplay of microservices, Docker, Kubernetes, and RabbitMQ, reflecting the demanding technical basis of a successful digital twin system. This project's breadth and accuracy are made clear by the rigorous deployment methods, extensive network of microservice interactions, and grasp of Kubernetes orchestration. This chapter serves as bridging the gap between theoretical design and practical implementation by lifting the curtain on these details.

CHAPTER

FIVE

RESULTS AND DISCUSSION

5.1 Introduction

In this chapter, we evaluate and discuss the results obtained from the implementation of a microservices-based digital twin solution focused on real-time data processing, using a wind turbine as our primary test case.

5.2 Case Study: Wind Turbine Digital Twin

For the purpose of capturing renewable energy, wind turbines are necessary. However, real-time maintenance and modifications, which can be greatly enhanced using digital twin technology, typically determine their effectiveness and lifespan. Consequently, our emphasis on creating a digital twin of a wind turbine is both essential and a step toward the improvement of green technology.

In our case, the wind turbine's digital twin acts as its virtual counterpart, mirroring its every aspect and behavior in real time. This twin is not just a static 3D model; it is a living entity that captures the intricate data from the physical turbine through sensors, processes this data, and reflects the turbine's current state, health, and performance. It can predict wear and tear, suggest preventive maintenance, and even simulate various conditions to forecast how the turbine might react. One frame of visualization of the test case is depicted in figure 5.2.1.

One of the core motivations behind focusing on wind turbines for our digital twin test case stems from the challenges associated with turbine maintenance. These giant structures, especially those located offshore, are often subjected to harsh environmental conditions—be it saltwater corrosion, heavy winds, or temperature fluctuations. Regular manual inspections, while necessary, are resource-intensive and might not always pinpoint hidden issues. A wind turbine digital twin, with its continuous monitoring capability, can provide early warnings for parts showing signs of wear or failure, ensuring timely interventions and reducing unplanned downtime.

Furthermore, the digital twin's simulation capability allows operators to test various operational scenarios without risking the actual physical asset. For instance, how would the turbine react to a sudden increase in wind speed? Or how would a minor pitch adjustment in the blades affect the energy output? Such

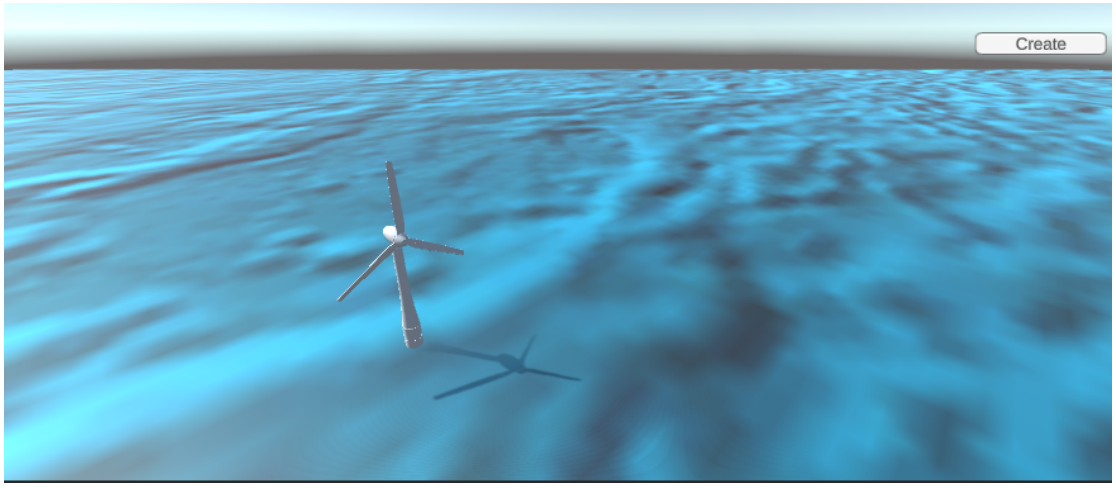

Figure 5.2.1: One frame of the 3D visualization of the test case in Unity 3D.

virtual experiments provide invaluable insights, leading to more informed decision-making.

5.3 Results

This section presents a thorough description of the outcomes that proceeded the deployment and integration of several microservices, which resulted in the construction of our digital twin of a wind turbine. We examine each microservice's functionality, and performance metrics, emphasizing their interdependencies and how they all work together to enhance the real-time capabilities of the digital twin. These outcomes show how reliable and effective our system is, from data acquisition to visualization and simulation. The potential and worth of our microservice-based approach in creating an efficient and responsive digital twin for wind turbines are highlighted by this organized presentation of the findings.

5.3.1 Data Acquisition Service

In the real-world application, our Data Acquisition Service is designed to efficiently gather real-time wind data directly from the turbine's sensors. The importance of this service cannot be overstated, as real-time data is fundamental to understanding and optimizing the turbine's performance. Typically, the turbine's sensors would capture critical parameters like wind speed and direction. This information, when captured at consistent intervals, aids in understanding the turbine's instantaneous performance and any external factors influencing it.

However, for the scope of our project and in the absence of real sensors, we have employed a data generation algorithm. This algorithm is designed to emulate the behavior of actual wind sensors by generating random values that are reflective of typical wind speeds and directions. The randomization ensures that the simulated data is not monotonous and mirrors the variability one would expect from actual wind conditions.

The Gaussian Normal Distribution is used here for creating these random values. The algorithm is summarized as:

Given:

- μ = mean wind speed (e.g., 6 m/s);
- σ = standard deviation of wind speeds (e.g., 1 m/s).

The generated wind speed v can be computed as:

$$v = \mu + \sigma \times \mathcal{N}(0, 1) \tag{5.1}$$

The wind direction θ is:

$$\theta = \text{Random}(0, 360) \tag{5.2}$$

where:

- $\mathcal{N}(0, 1)$ is a random number drawn from the standard normal distribution (with a mean of 0 and a standard deviation of 1).
- $\text{Random}(a, b)$ is a function that generates a random number between a and b.

Below is a sample data table, generated using our algorithm, that showcases the wind data collected over a short duration:

Table 5.3.1: Samples of generated wind data.

Sample	Wind Speed (m/s)	Direction (θ in degrees)
1	7.2	45
2	5.5	320
3	6.3	180
4	5.9	90
5	6.8	270

As a substitute assessment, this algorithmically generated data supports our digital twin model. Therefore, even in the absence of actual sensor data, we can validate and improve the performance of our digital twin. While not a replacement for actual wind data, this offers a workable solution for our digital twin model's first testing and validation phases.

5.3.2 Digital Twin Management

The Digital Twin Management service is essential in handling our specific test case, which is centered around creating a digital twin of a wind turbine. The key objective of the test was to effectively establish, manage, and monitor a wind turbine's digital representation, considering its vital parameters.

Each wind turbine digital twin in our test case was defined by three primary attributes:

- **Rotor Diameter:** This measures the span of the turbine blades and provides insights into the potential energy capture capacity of the turbine.

- **Power Rating:** It indicates the maximum power output capability of the turbine, which is essential for grid integration and power distribution.
- **Hub Height:** This measures the distance from the turbine's base to its central hub, affecting its exposure to wind patterns and consequently its efficiency.

These attributes form the basic profile of each wind turbine digital twin and are crucial for any subsequent simulations, predictions, or performance analyses. Table 5.3.2 provides attributes of the Wind Turbine Digital Twin in this test case.

Table 5.3.2: Attributes of the wind turbine digital twin in the test case.

Attribute	Description	Value
Rotor Diameter	Span of turbine blades	120m
Power Rating	Max power output	3.5MW
Hub Height	Base to central hub distance	85m

With the project-based structure, our test case was implemented by creating a unique project titled "Wind Turbine Digital Twin Test." Within this project, we housed the digital twin representation of our wind turbine, complete with its rotor diameter, power rating, and hub height parameters. The modular approach allowed for easy incorporation of additional turbines in the future, should they possess different attribute configurations or be situated in varying locations.

5.3.3 Simulation Service

In the quest to replicate the functioning and efficiency of a wind turbine, our test case hinges upon a structured and straightforward mathematical model [41]. This model meticulously assimilates primary parameters, specifically the rotor diameter and wind speed, each holding profound implications on the turbine's power production capacity.

The rotor diameter is central to determining the rotor's radius, r, expressed as:

$$r = \frac{D}{2}, \tag{5.3}$$

where D symbolizes the rotor diameter. The wind's interaction with the rotor blades, across a certain surface area, is the engine behind the turbine's power generation. This surface, termed as the rotor's swept area A, is computed from the radius:

$$A = \pi r^2 = \pi \left(\frac{D}{2}\right)^2 \tag{5.4}$$

A pivotal component in predicting the power output is the Power Coefficient, C_p, which varies based on the tip-speed ratio (TSR). In our simplified model, we have considered the following value as an estimate for C_p [41]:

$$C_p = 0.4 \tag{5.5}$$

With C_p in hand, the turbine's power output, P_{output}, becomes discernible via the formula [41]:

$$P_{\text{output}} = \frac{1}{2} \times C_p \times A \times V^3 \tag{5.6}$$

Our simulation microservice, which can function in two modes, emerges as the fundamental component of this system. The first, known as a **continuous simulation**, provides a flexible pause-and-resume function along with a close to real-time simulation of how a wind turbine operates. This mode takes in real-time data, reflecting the effects of changing conditions on the performance of the turbine right away. Users who use the pause-and-resume feature can briefly halt simulations, examine intermediate findings, recalibrate parameters, and then restart the simulation. Figure 5.3.1 shows wind data and corresponding power generation over an extended period in continuous mode.

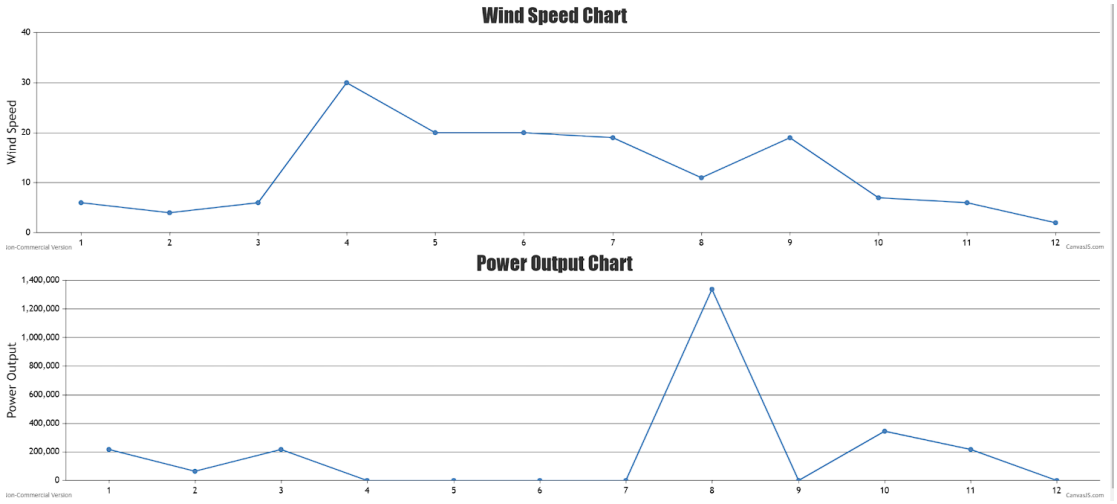

Figure 5.3.1: Continuous simulation mode: Line diagrams showcasing wind data and corresponding power generation over an extended period (wind speed in km/h and output power in Watts, the horizontal axis is time in seconds).

The **instantaneous simulation**, in contrast, is history-centric and uses archived data. This mode is helpful when looking backward to see how the turbine performed in the past or under earlier circumstances. The simulation microservice quickly processes and forms insights after feeding past wind data into our model, creating a clear picture of the turbine's previous performances. Figure 5.3.1 shows wind data and corresponding power generation based on historical data in instantaneous mode.

5.3.4 Command and Control Service

The Command and Control service is fundamental to the digital twin's interactive capabilities, establishing a two-way communication channel between the digital replica and the physical wind turbine system. This bi-directional link is critical not just for passive monitoring but for proactively influencing turbine behavior based on insights gleaned from simulations and analytics.

One of the main functionalities of this service is to send operational commands to the wind turbine. For instance, if the simulations predict adverse weather con-

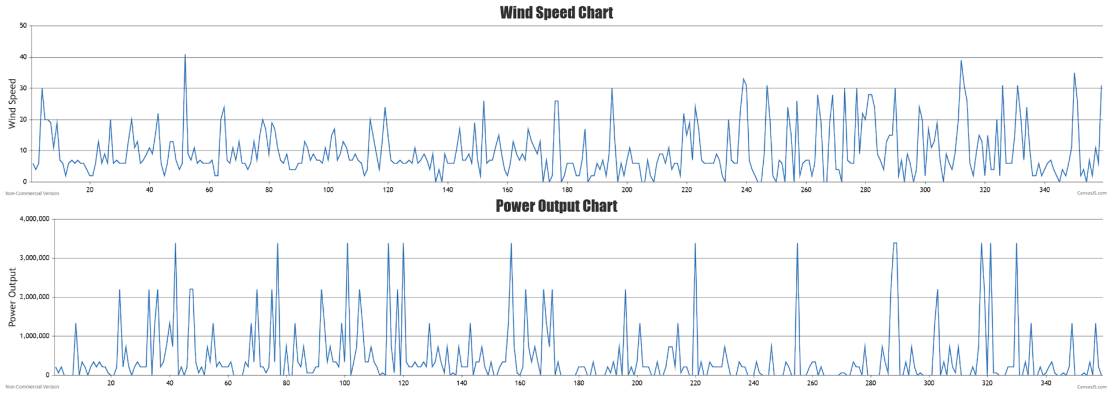

Figure 5.3.2: Instantaneous simulation mode: Line diagrams representing wind data and the immediate power generation response based on historical data (wind speed in km/h and output power in Watts, the horizontal axis is time in seconds).

ditions, the service can relay a command to temporarily halt turbine operations or adjust the blade angles to mitigate potential damage. Conversely, during optimal wind conditions, the service can maximize power generation by suggesting or automatically making real-time tweaks to the turbine's operational parameters.

5.3.5 Visualization Service

The Visualization service plays a pivotal role in intuitively presenting the intricate interplay between the real-time wind data and the corresponding turbine power output. Leveraging state-of-the-art graphical rendering, this service brings forth detailed visual representations, allowing stakeholders to grasp the intricate nuances of turbine performance quickly.

A primary graphical representation dominates this service. A time-series chart that displays the wind speed alongside the power output. This dual-axis line chart enables users to immediately discern the correlation between wind input and energy output. The x-axis typically represents the time continuum, be it in seconds, minutes, hours, or even days, depending on the user's selection. The left y-axis corresponds to wind speed values, while the right y-axis represents the power generated by the turbine. The diagrams are provided in Figures 5.3.1 and 5.3.2.

Furthermore, historical data comparisons are facilitated through overlaid graphs, where users can juxtapose the current turbine performance against a chosen past timeframe. Such comparisons are invaluable in assessing the long-term health and efficiency of the turbine, potentially highlighting wear and tear or other mechanical inefficiencies that might be creeping in over time.

5.4 Discussion

5.4.1 Scalability and Reliability

The microservices-based architecture adopted for the digital twin solution has been central to our investigation. The wind turbine test case, serving as a concrete implementation of the architecture, provided invaluable insights:

Scalability: We may autonomously expand each service in response to demand by dividing functionality into separate microservices (such as Data Acquisition, Digital Twin Management, Simulation, etc.). For instance, during times of high data influx, the Data Acquisition service, which collects real-time wind data, can be scaled up without disrupting other services. In our wind turbine test case, where the system handled varied volumes of sensor data, this modular scalability could clearly be illustrated.

Reliability: The decoupled nature of the microservices ensures that a failure in one service does not cripple the entire system. This principle was tested in the wind turbine case, where even if there were anomalies in the data streaming or simulation service, the Digital Twin Management or Visualization service remained unaffected.

5.4.2 Identifying Obstacles and Trade-offs

The practical application of the microservices-based architecture in the wind turbine test case revealed multiple challenges and trade-offs. Every system has some issues. One challenge we faced was making sure that data was always up-to-date. When there is a lot of data coming in, making sure the digital twin matches the real turbine can be tough.

Another problem was with our model for predicting turbine output. It works for basic situations, but sometimes it is not accurate when many things affect the turbine. Here are some of the trade-offs we could discover in this project:

Data Consistency: While microservices offer improved scalability, managing consistent data across services can be challenging. For instance, ensuring that the simulated data in the Visualization service matches the latest acquired wind data posed synchronization challenges.

Network Overhead: As each microservice communicates over the network, there's added latency. This was evident in the command and control operations for the wind turbine, where real-time decisions are paramount.

Trade-off Between Scalability and Complexity: While the system can scale efficiently, it comes at the cost of added complexity. Monitoring, maintaining, and orchestrating multiple microservices, as opposed to a monolithic architecture, required sophisticated tools and techniques.

Resource Management: Allocating appropriate resources to each microservice, especially in a real-time scenario like the wind turbine case, was a challenge. Striking a balance between resource allocation and service performance required continuous monitoring and fine-tuning.

5.5 Conclusions

By practically implementing the microservices-based architecture for the wind turbine digital twin, we were able to delve deep into the benefits and challenges of such a system. The test case served as an effective canvas, enabling us to explore the nuances of microservices in a real-world scenario, thereby addressing our thesis goals comprehensively.

CHAPTER

SIX

CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

The goal of this study was to develop an effective digital twin solution by utilizing the flexibility and modularity of microservices for the efficient processing of real-time data. The solution was built to test the real-time data processing capabilities using a wind turbine as the test subject. The results confirm the system's significant operational advantages and successfully demonstrate real-time data handling and processing.

In alignment with the initial research questions, the adoption of a microservices architecture indeed contributed positively to the scalability and reliability of the digital twin solution which was mentioned as the main concern in the second question. The isolation of various tasks, such as data gathering, management, simulation, and visualization into separate services facilitated unyielding flexibility. The modular approach proved instrumental in enhancing manageability, a distinct advantage over conventional methods, and endorsed the efficient utilization of digital twins for our case study.

The exploration into the creation of a microservices-based architectural framework for digital twins has yielded substantial insights and advancements. The innovative architectural framework presented in this thesis was carefully designed to improve the usefulness and effectiveness of digital twin systems and a was detailed response for the first research question of the thesis . The proposed framework, built upon principled design approaches and innovative methodologies, stands as a robust and scalable solution. It considerably reduces current difficulties and increases the general effectiveness and reliability of digital twin systems in practical applications.

However, despite the successful achievement of the goals, the analysis revealed possible areas for improvement. A significant challenge lies in maintaining data consistency, especially with a large data influx. The encountered obstacles and trade-offs, though highlighting the solution's limitations, offer an invaluable learning curve, extending insights into the complexities involved and providing a clear perspective on the complicated balance required in design decisions and this sheds insight into the third research question of the study.

But overall, despite the highlighted areas for improvement, the study underscores the promising potential of employing a microservices-based digital twin

model. The conclusions and discoveries derived from this study have the potential to act as a solid basis and a compass for directing further exploratory efforts and developments in the dynamic field of digital twin technologies.

6.2 Future Work

Looking ahead, there is a lot of room for improvement and growth:

- **Enhanced Scalability and Fault Tolerance Analysis:** Improve the suggested microservices-based digital twin architecture's scalability and fault tolerance. Look into more sophisticated and effective methods and algorithms that can support the architecture's scalability and reliability even more. Investigation on the paper written by Blinowski et al [42] might be helpful in addressing this topic.

- **Comprehensive Architectural Evaluation:** Perform a more comprehensive evaluation of the suggested microservices-based architectural framework in various real-world scenarios. Examine its flexibility, effectiveness, and resilience for a variety of industries and applications beyond wind turbines, as this could uncover particular difficulties and solutions relevant to those situations.

- **Improvement in Data Consistency:** Address the identified challenge of maintaining data consistency, especially with a large data influx. Research advanced data management and synchronization techniques to ensure seamless and consistent operation of the digital twin system.

- **User-Centered Design Enhancements:** Gather detailed feedback from potential end-users, especially digital twin developers, and stakeholders to ascertain the usability and practicality of the proposed digital twin system. Utilize the feedback to make informed enhancements to the system interface, features, and functionality to ensure it effectively meets the user and industry requirements.

In the end, our work is just one step towards enhancing the efficiency of digital twin models through the innovative use of microservices architecture. The path is now paved for further exploration and refinement, promising advancements in scalability, reliability, and efficient data management. Digital twin solutions, such as the one suggested in this work, can significantly improve and optimize a variety of industries by promoting innovation, efficiency, and sustainability with additional research and development.

REFERENCES

[1] Cor Verdouw et al. "Digital twins in smart farming". In: *Agricultural Systems* 189 (2021), p. 103046.

[2] Roberto Minerva, Gyu Myoung Lee, and Noel Crespi. "Digital twin in the IoT context: a survey on technical features, scenarios, and architectural models". In: *Proceedings of the IEEE* 108.10 (2020), pp. 1785–1824.

[3] Milan Groshev et al. "Toward intelligent cyber-physical systems: Digital twin meets artificial intelligence". In: *IEEE Communications Magazine* 59.8 (2021), pp. 14–20.

[4] Riyaz Ahamed Ariyaluran Habeeb et al. "Real-time big data processing for anomaly detection: A survey". In: *International Journal of Information Management* 45 (2019), pp. 289–307.

[5] Omar Al-Debagy and Peter Martinek. "A comparative review of microservices and monolithic architectures". In: *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE. 2018, pp. 000149–000154.

[6] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen. "Challenges when moving from monolith to microservice architecture". In: *Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17*. Springer. 2018, pp. 32–47.

[7] Chris Richardson. *Microservices Patterns: With examples in Java*. Manning Publications, 2020.

[8] Alfio Lombardo et al. "Design, implementation, and testing of a microservices-based Digital Twins framework for network management and control". In: *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE. 2022, pp. 590–595.

[9] Christian Esposito, Aniello Castiglione, and Kim-Kwang Raymond Choo. "Challenges in delivering software in the cloud as microservices". In: *IEEE Cloud Computing* 3.5 (2016), pp. 10–14.

[10] André B Bondi. "Characteristics of scalability and their impact on performance". In: *Proceedings of the 2nd international workshop on Software and performance*. 2000, pp. 195–203.

[11] Zhen Xiao, Qi Chen, and Haipeng Luo. "Automatic scaling of internet applications for cloud computing services". In: *IEEE transactions on computers* 63.5 (2012), pp. 1111–1123.

[12] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The datacenter as a computer: Designing warehouse-scale machines*. Springer Nature, 2019.

[13] Algirdas Avizienis et al. "Basic concepts and taxonomy of dependable and secure computing". In: *IEEE transactions on dependable and secure computing* 1.1 (2004), pp. 11–33.

[14] Matti A Hiltunen, Richard D Schlichting, and Carlos A Ugarte. "Enhancing survivability of security services using redundancy". In: *2001 International Conference on Dependable Systems and Networks*. IEEE. 2001, pp. 173–182.

[15] Michael R Lyu et al. *Handbook of software reliability engineering*. Vol. 222. IEEE computer society press Los Alamitos, 1996.

[16] James R Hamilton et al. "On Designing and Deploying Internet-Scale Services." In: *LISA*. Vol. 18. 2007. 2007, pp. 1–18.

[17] S Newman. *Building Microservices: Designing Fine-Grained Systems, 1ra edición. Sebastopol, Ciudad de California*. 2015.

[18] David Jaramillo, Duy V Nguyen, and Robert Smart. "Leveraging microservices architecture by using Docker technology". In: *SoutheastCon 2016*. IEEE. 2016, pp. 1–5.

[19] Karl Matthias and Sean P Kane. *Docker: Up & Running: Shipping Reliable Containers in Production*. " O'Reilly Media, Inc.", 2015.

[20] Michael W Grieves. "Product lifecycle management: the new paradigm for enterprises". In: *International Journal of Product Development* 2.1-2 (2005), pp. 71–84.

[21] Edward Glaessgen and David Stargel. "The digital twin paradigm for future NASA and US Air Force vehicles". In: *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*. 2012, p. 1818.

[22] Harald Sundmaeker et al. "Vision and challenges for realising the Internet of Things". In: *Cluster of European research projects on the internet of things, European Commision* 3.3 (2010), pp. 34–36.

[23] Abdulmotaleb El Saddik. "Digital twins: The convergence of multimedia technologies". In: *IEEE multimedia* 25.2 (2018), pp. 87–92.

[24] Qiang Lu et al. "Developing a digital twin at building and city levels: Case study of West Cambridge campus". In: *Journal of Management in Engineering* 36.3 (2020), p. 05020004.

[25] Fei Tao et al. "Digital twin-driven product design, manufacturing and service with big data". In: *The International Journal of Advanced Manufacturing Technology* 94 (2018), pp. 3563–3576.

[26] Amirashkan Haghshenas et al. "Predictive digital twin for offshore wind farms". In: *Energy Informatics* 6.1 (2023), pp. 1–26.

[27] *Functional Mock-up Interface Specification*. 2010. URL: https://fmi-standard.org/docs/3.0/ (visited on 11/25/2022).

[28] Marcus Wiens, Tobias Meyer, and Philipp Thomas. "The potential of FMI for the development of digital twins for large modular multi-domain systems". In: *Modelica Conferences*. 2021, pp. 235–240.

[29] Somayeh Malakuti et al. "A four-layer architecture pattern for constructing and managing digital twins". In: *Software Architecture: 13th European Conference, ECSA 2019, Paris, France, September 9–13, 2019, Proceedings*. Springer, 2019, pp. 231–246.

[30] Zhihan Wang et al. "Mobility digital twin: Concept, architecture, case study, and future challenges". In: *IEEE Internet of Things Journal* 9.18 (Mar. 2022), pp. 17452–17467.

[31] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[32] Florian Rademacher, Jonas Sorgalla, and Sabine Sachweh. "Challenges of domain-driven microservice design: A model-driven perspective". In: *IEEE Software* 35.3 (2018), pp. 36–43.

[33] Brian Levy. "The common capability approach to new service development". In: *BT Technology Journal* 23.1 (2005), pp. 48–54.

[34] Essam Shahat, Chang Tae Hyun, and Cheolho Yeom. "City digital twin potentials: A review and research agenda". In: *Sustainability* 13.6 (2021), p. 3386.

[35] Ian Hickson and David Hyatt. "Html5". In: *W3C Working Draft WD-Html5-20110525* (2011), p. 53.

[36] Ben Frain. *Responsive web design with HTML5 and CSS3*. Packt Publishing Ltd, 2012.

[37] Google. *Angular*. 2023. URL: https://angular.io/.

[38] Alex Okita. *Learning C# programming with Unity 3D*. AK Peters/CRC Press, 2019.

[39] David Dossot. *RabbitMQ essentials*. Packt Publishing Ltd, 2014.

[40] Linux Foundation. *Kubernetes*. 2023. URL: https://kubernetes.io/.

[41] Tony Burton et al. *Wind energy handbook*. John Wiley & Sons, 2011.

[42] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. "Monolithic vs. microservice architecture: A performance and scalability evaluation". In: *IEEE Access* 10 (2022), pp. 20357–20374.

A - GITHUB REPOSITORY

The following linked Github repositories contain all of the code and latex files used in this work.

Github repository links

- **Management Microservice**
  https://github.com/farhadhnz/FDT.Management (As of Sep. 2023).
- **Simulation Microservice**
  https://github.com/farhadhnz/FDT.Simulation (As of Sep. 2023).
- **Data Aquisition Microservice**
  https://github.com/farhadhnz/FDT.DataAquisition (As of Sep. 2023).
- **Command and Control Microservice**
  https://github.com/farhadhnz/FDT.CommandControl (As of Sep. 2023).
- **Visualization Microservice**
  https://github.com/farhadhnz/FDT.Visualization.UI (As of Sep. 2023).
- **Kubernetes Implementation**
  https://github.com/farhadhnz/FDT.K8S (As of Sep. 2023).
- **Thesis Latex Implementation**
  https://github.com/farhadhnz/Thesis-template-NTNU (As of Sep. 2023).