DEMOCRATIZING DIGITAL TWINS – A MANAGEMENT PLATFORM FOR THE DISTRIBUTION AND SHARING OF DIGITAL TWIN ASSETS

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
In Partial Fulfilment of the Requirements
For the Degree of Master of Science
In the Department of Computer Science
University of Saskatchewan
Saskatoon

By

CHRISTOPHER I. UGBOMAH

© Copyright Christopher I. Ugobah, April 2024. All rights reserved.
Unless otherwise noted, the copyright of the material in this thesis belongs to the author.
PERMISSION TO USE

In presenting this thesis/dissertation in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying this thesis/dissertation in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors. They supervised my thesis/dissertation work or, in their absence, the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication, or use of this thesis/dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis/dissertation.

DISCLAIMER

Reference in this thesis/dissertation to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favouring by the University of Saskatchewan. The views and opinions of the author expressed herein do not state or reflect those of the University of Saskatchewan and shall not be used for advertising or product endorsement purposes.

Requests for permission to copy or to make other uses of materials in this thesis/dissertation in whole or part should be addressed to:

Head of the Department of Computer Science
Department of Computer Science
University of Saskatchewan
176 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada
ABSTRACT

Digital Twins (DT) - virtual models designed to accurately reflect physical objects, systems, or processes - has received widespread adoption in its use to manage, monitor and respond to state changes of physical objects or environments such as factories, health monitoring devices, smart homes, and even cities; however, the primary focus of most research efforts have been on the challenges associated with sharing of data generated by the DT assets, with less emphasis on the distribution and sharing of the DT models themselves. This gap overlooks the access to models and the protection of intellectual property rights of the creators.

This study introduces a management platform designed for the distribution and sharing of DT models, facilitating control, personalization, and aggregation of DT Instances for specific objectives and generalized control. This is achieved without compromising the embedded knowledge or user autonomy over data sharing.

A key objective of this study is the exploration of the concept of Composite (Aggregate) Digital Twins – the logical combination of several (atomic) Digital Twin Instances to facilitate the management and control of a large number of heterogenous atomic DTs, without direct access to the instances themselves. By representing the integration of several different Digital Twin Instances present in a Composite Digital Twin as a JSON object, the study proposes to simplify the complex interaction of several interconnected, but disparate, systems into a single, unified view.

This study presents the implementation of a Rust-based proxy/middleware web server, utilizing webassembly-based containerized microservices, that allows for the deployment and hosting of DT models packaged as Open Container Initiative (OCI) images - an open industry standards around container formats and runtimes - and stored in repositories like Docker Hub in a vendor-agnostic environment. The services exposed by these models are accessible via RESTful API endpoints and executable on container runtime environments such as Docker Engine. The platform features a web portal for deploying and managing DT models, enabling controlled sharing between model creators (Owners) and users (Subscribers) through configurable access and data sharing policies.

Our evaluation focuses on the performance (latency and throughput) of accessing DT model services directly versus through our middleware, highlighting the trade-offs in access costs for enhanced security, data validation, and access control. The results affirm that the benefits of the middleware framework justify the increased access costs – in terms of latency and throughput, for providing a secure and controlled environment for sharing DT models and data.
ACKNOWLEDGEMENTS

This thesis stands as a testament to the invaluable support and guidance provided by a distinguished group of individuals to whom I am profoundly indebted.

Foremost, I extend my deepest appreciation to my supervisor, Professor Ralph Deters, whose expert guidance and patient mentorship were instrumental throughout the course of my studies and research. Professor Deters, your insightful counsel has illuminated my path forward, and for this, I am eternally grateful.

I am equally thankful to Professor Julita Vassileva for her unwavering support and quiet confidence throughout my academic journey. Your encouraging words served as a vital source of motivation during challenging times, and I am deeply grateful for your role in my development. Special recognition is due to Professor Ramakrishna Gokaraju, a key member of my thesis advisory and examination committee. His insightful feedback and astute observations have significantly enhanced the quality of my work, and his contributions cannot be overstated.

I must also express my profound gratitude to my peers in the Multi-User Adaptive Distributed Mobile and Ubiquitous Computing (MADMUC) Lab. Your readiness to assist at crucial moments was indispensable. Thank you to Maurine, Shakiba, and all the members of the Department of Computer Science for your genuine concern and support for my well-being and success. My heartfelt thanks go out to my friends and family, whose steadfast support and prayers have been the cornerstone of my achievements. Special thanks are extended to the Ugborah family, Sunday and Nadia Ovemurai, and Rick and Sharon Murza, whose encouragement, prayers, and humour have continually refreshed my spirit.

Lastly, I am immensely grateful to my wife, Amaka, and my children, Giovanni, Chinyenum (Andrea), and Francesco. Your patience and understanding during my periods of absence and stress were the bedrock of my perseverance. Your love sustains me, and I cherish each of you profoundly.

In sum, this journey would not have been possible without the collective support of each individual mentioned, and many others who have contributed in various capacities. I am eternally appreciative of all your efforts.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMISSION TO USE</td>
<td>I</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>II</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>III</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>IV</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>VI</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>VII</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>IX</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER 2 PROBLEM DEFINITION</td>
<td>16</td>
</tr>
<tr>
<td>2.1 Research Questions</td>
<td>18</td>
</tr>
<tr>
<td>2.2 Research Objectives</td>
<td>19</td>
</tr>
<tr>
<td>CHAPTER 3 LITERATURE REVIEW</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Digital Twins – Origins and Brief History</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Digital Twins Related Concepts</td>
<td>20</td>
</tr>
<tr>
<td>3.2.1 Digital Models (DM)</td>
<td>21</td>
</tr>
<tr>
<td>3.2.2 Digital Shadows (DS)</td>
<td>21</td>
</tr>
<tr>
<td>3.2.3 Behavioural Models</td>
<td>21</td>
</tr>
<tr>
<td>3.3 Digital Twins in Action: Real-World Applications</td>
<td>22</td>
</tr>
<tr>
<td>3.3.1 Industrial Applications of Digital Twins</td>
<td>23</td>
</tr>
<tr>
<td>3.3.2 Healthcare, Personalised Medicine and Digital Twins</td>
<td>24</td>
</tr>
<tr>
<td>3.3.3 Digital Twins and Smart Cities</td>
<td>25</td>
</tr>
<tr>
<td>3.4 Digital Twin Access – Model and Data Sharing</td>
<td>26</td>
</tr>
<tr>
<td>3.5 Digital Twins Packaging, Distribution and Deployment – OCI Prospects</td>
<td>27</td>
</tr>
<tr>
<td>3.6 DT Data Sharing and Usage Control</td>
<td>28</td>
</tr>
<tr>
<td>3.7 Summary</td>
<td>29</td>
</tr>
<tr>
<td>CHAPTER 4 DESIGN AND ARCHITECTURE</td>
<td>31</td>
</tr>
<tr>
<td>4.1 Layers of Abstraction</td>
<td>32</td>
</tr>
<tr>
<td>4.1.1 Client Layer</td>
<td>32</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1. Summary of Literature Review ................................................................. 30
Table 4.1. Roles and Responsibilities ....................................................................... 34
Table 6.1. Experiments Setup ................................................................................... 58
Table 6.2. Application Performance Index (Apdex) Default Ratings ......................... 59
Table 6.3. Experiment Infrastructure ........................................................................ 59
Table 6.4. Relationship Between the Experiments and the Research Objective .......... 60
Table 6.5. Experiment 1 Performance Summary Report ............................................. 61
Table 6.6. Experiment 2.1 Performance Summary Report for HTTP GET Requests .... 66
Table 6.7. Experiment 2.2 Performance Summary Report for HTTP POST Requests .... 69
Table 6.8. Experiment 3 Performance Summary Aggregate Twin ................................ 73
Table 6.9. Summary Performance: Application Performance Indices (Apdex) for the DTMP .... 77
Table C.1. Infrastructure Cost for Solution and 1000 DT Instances ............................. 98
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 1.1</td>
<td>Digital model, shadow and twin</td>
<td>10</td>
</tr>
<tr>
<td>Fig. 1.2</td>
<td>A Smart Home with multiple DTs</td>
<td>12</td>
</tr>
<tr>
<td>Fig. 1.3</td>
<td>A Schematic of various interacting DTs in a Smart Home System</td>
<td>13</td>
</tr>
<tr>
<td>Fig. 2.1</td>
<td>Challenges with existing DT Management Platforms</td>
<td>16</td>
</tr>
<tr>
<td>Fig. 3.1</td>
<td>Applications of Digital Twin Technology</td>
<td>22</td>
</tr>
<tr>
<td>Fig. 3.2</td>
<td>The use of DTs to remotely update feedback controllers</td>
<td>24</td>
</tr>
<tr>
<td>Fig. 4.1</td>
<td>Proposed Solution Architecture</td>
<td>32</td>
</tr>
<tr>
<td>Fig. 4.2</td>
<td>A Translation of an “Aggregation Rule” to a JSON-formatted Instruction Set</td>
<td>37</td>
</tr>
<tr>
<td>Fig. 4.3</td>
<td>A Flowchart Representation of the Generated JSON Instruction Set</td>
<td>37</td>
</tr>
<tr>
<td>Fig. 5.1</td>
<td>Digital Twins Management Platform (DTMP) Landing Page</td>
<td>43</td>
</tr>
<tr>
<td>Fig. 5.2</td>
<td>A Screenshot of a List of Referenced Rust Libraries</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 5.3</td>
<td>DT Model Owners Sign-Up Interface</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 5.4</td>
<td>User Sign-In and Subscription Sequence Diagram</td>
<td>46</td>
</tr>
<tr>
<td>Fig. 5.5</td>
<td>DT Model Deployment Wizard Interface – Basic Model Setup</td>
<td>47</td>
</tr>
<tr>
<td>Fig. 5.6</td>
<td>DT Model Deployment Wizard Interface – Model Policy Setup</td>
<td>48</td>
</tr>
<tr>
<td>Fig. 5.7</td>
<td>DT Model Deployment Wizard Interface – Data Sharing</td>
<td>50</td>
</tr>
<tr>
<td>Fig. 5.8</td>
<td>DT Model Deployment Wizard Interface – Parameter Value Definition</td>
<td>51</td>
</tr>
<tr>
<td>Fig. 5.9</td>
<td>Screenshot showing a list of a DT Owner’s Models</td>
<td>51</td>
</tr>
<tr>
<td>Fig. 5.10</td>
<td>Screenshot showing Published DT Models</td>
<td>52</td>
</tr>
<tr>
<td>Fig. 5.11</td>
<td>User Personalizing a Twin</td>
<td>53</td>
</tr>
<tr>
<td>Fig. 5.12</td>
<td>User Interaction with a DT Instance Service</td>
<td>54</td>
</tr>
<tr>
<td>Fig. 5.13</td>
<td>Payload of a Composite Digital Twin Instance Creation</td>
<td>55</td>
</tr>
<tr>
<td>Fig. 5.14</td>
<td>The Use of a Digital Twin Instance with AI Assistant Enabled</td>
<td>56</td>
</tr>
<tr>
<td>Fig. 6.1</td>
<td>Apdex Score Calculation Formula</td>
<td>59</td>
</tr>
<tr>
<td>Fig. 6.2</td>
<td>Bar Graph Comparison of the Average Latency (ms)</td>
<td>61</td>
</tr>
<tr>
<td>Fig. 6.3</td>
<td>Line Graph Comparison of the Average Latency (ms)</td>
<td>62</td>
</tr>
</tbody>
</table>
Fig. 6.4. Bar Graph Comparison of the Average Throughput (requests/sec) ............................................. 63
Fig. 6.5. Line Graph Comparison of the Average Throughput (requests/sec) ............................................. 63
Fig. 6.6. Bar Graph Comparison of the Average Error Rate (%) ............................................................... 64
Fig. 6.7. Line Graph Comparison of the Average Error Rate (%) .............................................................. 64
Fig. 6.8. Bar Graph Comparison of the Average Latency (ms) for HTTP GET Requests ............................... 66
Fig. 6.9. Line Graph Comparison of the Average Latency (ms) for HTTP GET Requests ............................. 67
Fig. 6.10. Bar Graph Comparison of the Average Throughput (req/sec) for HTTP GET Requests ...................... 67
Fig. 6.11. Line Graph Comparison of the Average Throughput (req/sec) for HTTP GET Requests .............. 68
Fig. 6.12. Bar Graph Comparison of the Average Error Rate (%) for HTTP GET Requests ............................ 68
Fig. 6.13. Line Graph Comparison of the Average Error Rate (%) for HTTP GET Requests .......................... 69
Fig. 6.14. Bar Graph Comparison of the Average Latency (ms) for HTTP POST Requests ............................. 70
Fig. 6.15. Line Graph Comparison of the Average Latency (ms) for HTTP POST Requests ............................ 70
Fig. 6.16. Bar Graph Comparison of the Average Throughput (req/sec) for HTTP POST ............................. 71
Fig. 6.17. Line Graph Comparison of the Average Throughput (req/sec) for HTTP POST ............................. 71
Fig. 6.18. Bar Graph Comparison of the Average Error Rate (%) for HTTP POST Requests ............................ 72
Fig. 6.19. Line Graph Comparison of the Average Error Rate (%) for HTTP POST Requests ........................... 72
Fig. 6.20. Bar Graph Comparison of the Aggregate Twin Average Latency (ms) ............................................ 74
Fig. 6.21. Line Graph Comparison of the Aggregate Twin Average Latency (ms) ......................................... 74
Fig. 6.22. Bar Graph Comparison of the Aggregate Twin Average Throughput (requests/sec) ...................... 75
Fig. 6.23. Line Graph Comparison of the Aggregate Twin Average Throughput (requests/sec) ...................... 75
Fig. 6.24. Bar Graph Comparison of the Aggregate Twin Average Error Rate (%) ...................................... 76
Fig. 6.25. Line Graph Comparison of the Aggregate Twin Average Error Rate (%) ...................................... 76
Fig. C.1. PostgreSQL Database Flexible Server Pricing .................................................................................. 98
Fig. C.2. OpenAI GPT-3.5 Turbo Model Pricing ......................................................................................... 99
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>DT</td>
<td>Digital Twins</td>
</tr>
<tr>
<td>DTI</td>
<td>Digital Twin Instance</td>
</tr>
<tr>
<td>Apdex</td>
<td>Application Performance Index</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CI/CD</td>
<td>Continuous Integration/Continuous Delivery</td>
</tr>
<tr>
<td>CPS</td>
<td>Cyber-Physical System</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>DTaaS</td>
<td>Digital Twins as a Service</td>
</tr>
<tr>
<td>DTM</td>
<td>Digital Twin Model</td>
</tr>
<tr>
<td>DTMP</td>
<td>Digital Twins Management Platform</td>
</tr>
<tr>
<td>GE</td>
<td>General Electric</td>
</tr>
<tr>
<td>GPT</td>
<td>Generative Pre-trained Transformer</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>JSON</td>
<td>Javascript Object Notation</td>
</tr>
<tr>
<td>LLM</td>
<td>Large Language Model</td>
</tr>
<tr>
<td>OCI</td>
<td>Open Container Initiative</td>
</tr>
<tr>
<td>PNO</td>
<td>Privacy Non-sensitive Object</td>
</tr>
<tr>
<td>PSO</td>
<td>Privacy-Sensitive Object</td>
</tr>
<tr>
<td>RBAC</td>
<td>Role-Based Access Control</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UCON</td>
<td>Usage Control Model</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WASM</td>
<td>WebAssembly</td>
</tr>
<tr>
<td>WoT TD</td>
<td>Web of Things Thing Description</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

The concept of the Digital Twin [1] has its origins in the pioneering work of Michael Grieves, alongside John Vickers at NASA, marking the beginning of a transformative approach to bridging the physical and digital worlds. Since its inception, the Digital Twin concept has evolved, adopting various interpretations across different domains and applications. This evolution has, however, led to misconceptions and a lack of consensus on its definition, underscoring the complexity and multifaceted nature of this innovative idea. At its core, the Digital Twin is traditionally understood as a dynamic, bidirectional linkage between a physical entity and its virtual counterpart, facilitating real-time simulation, analysis, and control.

Amidst this backdrop, it's important to distinguish between related but distinct concepts that have emerged in the field. One such concept is the Digital Model (DM), which serves as a digital representation of either an existing or planned physical object. Designed primarily as a simulation tool, the Digital Model lacks automatic data exchange with its physical counterpart, setting a clear boundary in terms of interactivity and real-time updating. Another key term in this discourse is the Digital Shadow (DS), which represents a physical object in a digital format, characterized by a unidirectional data flow from the physical entity to its digital reflection. Unlike the Digital Model, the Digital Shadow's value lies in its ability to support data-driven insights, offering a comprehensive view of the physical entity's status and performance without the capability for direct intervention.

In contrast to both the Digital Model and Digital Shadow, the Digital Twin concept elevates the relationship between the physical and virtual worlds to a level of mutual influence and continuous synchronization. It
encompasses a two-way flow of information, where each entity—the physical and the virtual—can inform and potentially alter the state of the other. This reciprocal interaction not only enhances the accuracy and utility of the virtual model but also opens up new possibilities for predictive analytics, operational optimization, and informed decision-making. Through the lens of the Digital Twin, we gain a more integrated and actionable perspective on the interplay between physical reality and its digital reflection, setting the stage for advancements in simulation, monitoring, and control across various sectors.

Building on the foundational concepts introduced by Michael Grieves and John Vickers, the landscape of Digital Twins is further enriched by the delineation of sub-categories, such as the Digital Twin Prototype (DTP), Digital Twin Instance (DTI), and the Digital Twin Aggregate (DTA) [2], each offering unique perspectives and functionalities within the digital-physical paradigm.

The Digital Twin Prototype (DTP) serves as the model encompassing the essential informational elements required to replicate its virtual design in the physical realm. This prototype acts as a blueprint, detailing the specifications and attributes necessary to create a physical counterpart that mirrors the virtual model accurately. The DTP is foundational, providing the basis from which specific instances of the product can be realized.

In contrast, the Digital Twin Instance (DTI), or atomic Digital Twins, pertains to a unique, corresponding physical product that maintains a direct and ongoing connection with its digital counterpart throughout the product’s lifecycle. The DTI represents the manifestation of the DTP in the physical world, embodying the specific characteristics and operational data of an individual unit. This one-to-one correspondence ensures a detailed and dynamic reflection of the product's state and performance over time.

Expanding upon these concepts, Grieves and Vickers introduce the Digital Twin Aggregate (DTA), also known as the Composite Digital Twin. The DTA represents a higher-order integration of multiple Digital Twin Instances, encapsulating the collective attributes and interactions of several distinct units. This composite may be conceptualized as a logical structure, potentially formalized through data structures like JSON objects, facilitating access to and interaction with the underlying DTIs or other composites. Through either ad-hoc or proactive querying, the DTA provides insights into the synergistic behaviours and emergent properties arising from the interplay among its constituent DTIs. Composite DTs represent an approach to modelling a complex system of interconnected parts in a more natural manner.

This hierarchical structuring—from the foundational DTP through the specific DTI to the integrative DTA—illustrates the versatility and depth of the Digital Twin concept. It highlights the ability of Digital Twins to operate at different scales and levels of complexity, from individual components to complex
systems, enabling a comprehensive and nuanced understanding of both discrete elements and their collective dynamics.

In an era where the confluence of technology and environmental consciousness is leading to transformative changes in societal infrastructure, the role of Digital Twins (DT) emerges as a cornerstone in this evolution. The burgeoning integration of Internet of Things (IoT) devices and the nuanced application of Artificial Intelligence (AI) indeed play substantial roles in advancing our capabilities. Yet, it is the strategic utilization of DTs that offers a profound leap in how we manage and interact with complex systems. For example, the heightened awareness of climate change necessitates a paradigm shift in how we engage with our environment. Central to this shift is the evolution of our urban habitats into 'smart cities,' where energy efficiency is not just an option, but a necessity. The potential of DTs in orchestrating a more sustainable and efficient future, particularly within the context of smart home systems presents a viable use case for our proposal. This paper presents a platform that transcends traditional boundaries, employing DTs not merely as tools for specific applications, such as energy efficiency, but as universal enablers of participatory innovation and policy-making.

Fig. 1.2. A Smart Home with multiple DTs
The smart home diagram presented exemplifies an ecosystem where different components, such as solar panels, electricity supply mains, smart home systems, heat pumps, and electric furnaces, work in unison to optimize energy use. Each component can be individually represented by a DT instance (DTI), capturing its unique data, behaviour, and interactions. However, the true power of DTs emerges when these instances are aggregated to form a comprehensive model that elucidates the inter-relationships and collective behaviour of the entire system. This aggregated DT instance not only provides a holistic view of a home's energy profile or carbon footprint but also serves as a microcosmic model of a smart city's potential for energy management.

By aggregating different DT instances, we can construct complex models that accurately reflect the energy dynamics of a smart home. This amalgamation allows for sophisticated analysis and control, enabling homeowners to become active participants in energy conservation and climate change mitigation. When scaled, this model fosters a community of energy-conscious citizens, each contributing vital data from their DT instances. This shared pool of information becomes a powerful tool in the hands of urban planners and policymakers, driving informed decisions that edge us closer to sustainability goals.

While the narrative of climate change mitigation and carbon footprint reduction provides a compelling use case, our platform's potential is far-reaching. The versatility of our platform is in its capacity to empower citizens beyond energy management. By enabling individuals to contribute data from their DT instances, it facilitates a granular yet expansive canvas of real-time information. This data is not only instrumental in driving sustainable practices but also serve as a democratic tool that fuels policy discourse and development across various domains. Be it urban planning, public health, transportation, or environmental conservation,
the insights garnered from aggregated DTs provide a basis for informed decisions that resonate with the needs and contributions of the populace.

The proposed solution, grounded in a Rust-based middleware with webassembly-based containerized microservices, is pivotal in actualizing this vision. It offers a secure, scalable, and flexible framework that allows for the deployment of DT models packaged as OCI images. The platform ensures that the data sovereignty of individuals is respected, while facilitating a pool of information that powers community-driven policy formulation and societal advancement.

This middleware architecture does not merely support the logical aggregation of DT instances for energy systems but also extends to any domain where DTs can be applied. It is the bedrock upon which DT instances from diverse sectors can be harmonized, thus enabling a multidimensional perspective that is crucial for holistic policy-making. By fostering such a dynamic and inclusive environment, our platform positions DTs as a catalyst for civic engagement and a beacon for the co-creation of a sustainable and responsive society.

In this implementation, users interact with containerized Digital Twin Instances—each running from a Digital Twin Model OCI image—through a web interface facilitated by the middleware platform. This platform acts as a proxy, channelling various HTTP requests to the digital twins. It ensures that each request complies with predefined policies by conducting access and parameter checks. Furthermore, it obfuscates usage data in accordance with the data owner's specifications and stores the data as necessary.

Users interact with the functionalities of Digital Twins, such as changing a heat pump's compressor speed or toggling the device's operational mode between heating and cooling, primarily through HTTP/RESTful endpoints provided by the Digital Twin model. Specifically, a user communicates with a particular Digital Twin instance through a unique web address and port allocated to that instance. This allocation, along with management and deallocation of resources, is handled internally by the platform, facilitating seamless user interaction with the Digital Twins.

In summary, the platform recognizes and harnesses the power of DTs as a transformative agent for societal good. It extends the purview of DTs from specific use cases to a universal framework that empowers individuals to contribute to the collective intelligence that shapes policies and innovation. In doing so, it
paves the way for a future where technology is not only a driver of efficiency but also a platform for inclusive participation and progressive change.

The rest of this study is organized as follows:

- Chapter 2 introduces the problem statement, enumerating the research questions and objectives.
- Chapter 3 presents the literature review, providing a brief historical background to Digital Twins, highlighting some of the use cases and applications of DTs, as well as presenting the efforts of related works in the areas of Digital Twin Model and Data Sharing, Security, and Usage Control.
- Chapter 4 discusses the system architecture, presenting the different layers of system abstraction and participating components, as well as some of the technologies used.
- Chapter 5 presents the system's implementation, capturing its entire development process. It describes the DT deployment process as well as highlights several of the solution features implemented to address or support the stated project objectives.
- Chapter 6 evaluates the system's performance, presenting data gathered from the various experiments and evaluations.
- Chapter 7 summarizes the research, presents our findings, charts a course for future work and improvements.
CHAPTER 2

PROBLEM DEFINITION

The landscape of commercial digital twin management platforms is riddled with significant impediments. Chief among these challenges are the proprietary nature of these solutions, with each vendor serving its own interpretation of DT Models, imposing considerable portability and interoperability restrictions, compelling developers to undertake the laborious task of customizing DT models for various platforms, a process that is both time-consuming and resource-intensive (Refer to Fig. 2.1). This limitation not only impedes productivity but also places constraints on the widespread adoption of DT technology, notably among individuals and smaller enterprises.

Furthermore, the open-source alternatives, while offering alleviation in terms of cost, are not immune to the challenges inherent in a lack of standard DT model definitions, convoluted sharing procedures, insufficient intellectual property safeguards, and the absence of definitive protocols to manage policy infractions and usage noncompliance. These shortcomings obstruct the full harnessing of digital twin technology's transformative prospects. In response to these complexities, this study proposes the creation of an open-source digital twin management platform that embodies cost-efficiency, user-centricity, and a universal protocol for the packaging, distribution, and execution of DT assets. This initiative significantly simplifies the deployment and dissemination of digital twins, paving the way for "Digital Twins as a Service (DTaaS)". By addressing the aforementioned limitations, the proposed platform endeavours to actualize the untapped potential of DTs, enabling entities of all sizes to achieve operational excellence, augmented performance, and profound insights.

Fig. 2.1. Challenges with existing DT Management Platforms
This study outlines key challenges encountered in the current digital twin management platforms:

1. **DT Asset Hosting**: A primary issue is the absence of widely accepted standards for developing DT assets, leading to platforms that often result in vendor lock-in due to these constraints.

2. **DT Asset Portability and Interoperability**: The diversity in digital twin implementations across vendors, each built on proprietary technologies, hampers portability between systems and interoperability among these technologies. This diversity complicates the integration of multiple Digital Twin Instances (DTIs) into a cohesive composite.

3. **DT Data Ownership**: There is a lack of clarity surrounding data ownership within existing platforms, raising questions about who owns the data, who has access, and the purposes for which data can be accessed.

The complexities inherent in these systems often necessitate specialized expertise, thus hindering the creation and utilization of DT Models (DTMs). This intricacy, coupled with substantial costs and the hazard of vendor lock-in, confines the technology's scalability and accessibility. The establishment of a universally open standard empowers a broad spectrum of users to tap into a rich repository of DT models, thereby fostering an ecosystem conducive to innovation and informed decision-making in the face of global environmental imperatives. (See Figure. 2.2 for a schematic representation).

The development of the platform and the proposal for a universally acceptable format for digital twin models is inherently resistant to vendor lock-in and accentuates interoperability, ensuring seamless amalgamation with existing tech ecosystems.

This study is predicated on the belief that by addressing the prevailing constraints, we can unleash the boundless possibilities inherent in digital twin technology, allowing entities to realize enhanced operational efficiency, superior performance metrics, and deep insights into virtualized processes. The endgame is to democratize digital twin technology, positioning it as an indispensable tool for organizations to stimulate innovation across the spectrum, as well as making it a tool to empower citizens and give them a participatory role in driving policies.
The solution is implemented with the goal of meeting the unique needs of:

- **DT Model Owners**: The architects and knowledge custodians behind these models, warranting protection for their intellectual property and potential commercial privileges.
- **DT Twin Users**: The principal stakeholders and data contributors to the system.

While formulating this solution, paramount importance is accorded to the sanctity of collaboration artifacts – user data and proprietary models/IP.

### 2.1 Research Questions

1. What mechanisms can be instituted to share DT models without compromising the intellectual property of the model creators while eliminating deployment and usage complexities?

2. Is it feasible to implement owner-specific policies to govern access to individual DT models consistently, while enforcing certain levels of access control over usage data generated by DT Twin users?

3. Despite the intrinsic overhead owing to policy enforcement, can we delineate and achieve performance benchmarks for the middleware solution?
2.2 Research Objectives

1. Develop a Flexible System for Creating DT Instances: Create a system capable of facilitating the deployment of Digital Twin Instances (DTIs) as containers across various container runtime environments, ensuring versatility and adaptability in digital twin creation.

2. Assess Platform Performance with Policy Enforcement: Conduct an evaluation on the impact of model-specific and data governance policies of the proposed platform’s performance, specifically focusing on latency and throughput when accessing digital twin models.

3. Analyse Performance Impact of Composite DTs: Investigate the latency and throughput performance associated with accessing Composite Digital Twins (DTAs) to determine if aggregating multiple DTIs significantly affects overall system performance. The goal is to confirm that the integration of several DTIs into a composite structure does not degrade system efficiency.
CHAPTER 3

LITERATURE REVIEW

This chapter presents a brief history on the origins of the concept of Digital Twins (DTs), and discusses some of the modern day applications of DTs across various domains. Furthermore, a review of some earlier works and proposals on the secure sharing of DT assets, as well as possible security considerations are covered.

3.1 Digital Twins – Origins and Brief History

The concept of Digital Twins dates back to the early 2000s and has seen a significant surge in interest and research in recent years. The concept of Digital Twins can be traced back to Dr. Michael Grieves, who coined the term in 2002 during his work at the University of Michigan. Grieves proposed the idea of a virtual representation of physical products, serving as a digital surrogate for real-world assets. Since then, other definitions and descriptions of Digital Twins have since been proposed. [1]

The adoption of Digital Twins within industrial settings gained momentum in the mid-2000s. General Electric (GE) was among the pioneers in leveraging Digital Twins for predictive maintenance, asset monitoring, and optimization of complex systems. Since then, the Digital Twin has found application in several different areas including healthcare, aviation and smart cities, just to mention a few examples.

According to Grieves and Vickers, Digital Twins (DTs) can be categorized into two distinct types: Prototypes and Instances. A prototype DT serves as a virtual representation used to optimize the final design of a product. This is especially valuable for expensive and intricate items that cannot be physically experimented with during the development phase. On the other hand, a DT instance refers to a virtual representation of a product that has been manufactured and deployed, providing insights into its behavior and performance beyond the production lines [3].

3.2 Digital Twins Related Concepts

In the comprehensive review by Fuller et al., titled “Digital Twin: Enabling Technologies, Challenges, and Open Research,” [1] the authors address the notions of Digital Models (DM) and Digital Shadows (DS), labelling them as misconceptions within the Digital Twin discourse. Nonetheless, these concepts underline traditional applications of digital twins, such as visualization, simulation, and interaction. [4]
3.2.1 Digital Models (DM)

Digital Models are aligned with Grieves' and Vickers' concept of a Digital Twin Prototype [2]. Predominantly utilized in the product development phase, these models serve to simulate and demonstrate the functionality or behaviour of a product still under development [4]. Characterized by their detachment from any physical counterpart, Digital Models broaden the digital twin concept to include scenarios where the corresponding physical entity may not yet exist.

3.2.2 Digital Shadows (DS)

Digital Shadows establish a unidirectional link from a physical asset to its virtual representation, embodying the original envisioned utility of the Digital Twin [5]. This concept initially aimed to enhance monitoring and maintenance planning for physical assets through a virtual system, distinguished from Digital Models by a direct connection to the physical entity via a "physical-to-virtual" link. This connection often involves receiving operational and usage data from the physical asset through IoT sensors. In academic literature, any entity capable of collecting, generating, or processing data to sufficiently describe the physical entity's state might be considered its Digital Shadow, hinting at its integral role within the broader Digital Twin framework [4].

Jones et al. [6] highlight a less-discussed aspect: the "human-in-the-loop" in scenarios lacking a "virtual-to-physical" connection. Here, humans complete the feedback loop by manually implementing changes to the physical entity. Modern interpretations of digital twins now underscore the significance of automated data exchanges between physical and virtual entities, enriching the interaction model (see Fig. 1.1).

3.2.3 Behavioural Models

An emerging trend, most notably in Industry 4.0, is the possible enhancement of digital twin models through the integration of Artificial Intelligence (AI) capabilities. Behavioural digital twins refer to a concept in technology and data analytics where a virtual model or simulation is created to mimic the behaviours, preferences, and decision-making processes of individuals or groups. This concept is derived from the broader idea of digital twins, which are primarily used in industries like manufacturing, urban planning, and healthcare to create digital replicas of physical systems or processes for simulation, analysis, and optimization purposes.
In the context of behavioural digital twins, the focus shifts from physical objects or systems to human behaviour. These models can integrate a wide array of data sources, including but not limited to social media activity, browsing histories, purchase records, and even real-time location data, to create a comprehensive and dynamic representation of a person's or group's behavioural patterns, usually not reproducible by traditional agents based on finite state machines.

An application of this is in Task Learning, where AI agents employ Imitation Learning (IL) and Regression Logistics (RL) algorithms. These agents are trained via observation of human operations, refining their behaviours in a simulated digital twin environment before actual deployment in real-world settings. This approach underscores the potential of behavioural digital twins in facilitating complex learning and decision-making processes, bridging the gap between virtual simulations and tangible operational improvements [7].

3.3 Digital Twins in Action: Real-World Applications

Digital Twins are finding increasing applications in various different domains where they enable real-time monitoring, simulation, and optimization of various processes [1]. These versatile virtual counterparts find applications in diverse domains, each with specific use cases.
3.3.1 Industrial Applications of Digital Twins

Digital twin (DT) is one of the most promising enabling technologies for realizing smart manufacturing and Industry 4.0 [9]. By harnessing the power of Artificial Intelligence (AI) and Cyber-Physical Systems (CPSs), Digital Twins have become integral in processing and analysing vast amounts of data, facilitating decentralized, autonomous decision support systems. These systems contribute to enhanced productivity, reduced downtime, and increased efficiency in manufacturing processes.

One of the key challenges addressed by Digital Twins and AI in this context is how to effectively utilize real-time sensor data streams in Digital Twins to improve remote control operations and maintenance. Manufacturers can remotely monitor and manage equipment, identify potential issues, and schedule maintenance activities proactively, minimizing unexpected breakdowns and optimizing resource allocation. Additionally, optimizing computing resources for digital twins in the cloud-to-thing continuum is crucial to satisfy key performance indicators such as latency and security requirements [7]. This ensures that the digital twin models operate efficiently, delivering real-time insights and supporting seamless interactions between the physical and virtual worlds.

Moreover, a significant objective for Digital Twins in the manufacturing domain is to improve the safety and well-being of workers within factory floors. By integrating Digital Twins and AI, it becomes possible to predict the state of various manufacturing machinery, thereby leading to improved safety and risk reduction. For instance, predictive maintenance techniques can detect early signs of component failures, allowing manufacturers to replace or repair parts before they pose safety risks.

When considering human-machine collaboration scenarios, where humans and machines work closely together, safety is of paramount importance. Failures of either humans or machines can result in accidents. While human errors might be highly unpredictable, machine failures are often associated with the state of their various components, making them more foreseeable. Identifying potential machine failures through Digital Twins and AI is feasible, and the utilization of Machine Learning-based solutions further enhances the accuracy of prediction and classification problems.

To ensure the highest level of safety, control and command mechanisms must be built into the digital twin model. This enables a deeper understanding of the device state and serves as an interface through which machines can be remotely powered down in the event that they pose any safety risk to their human operators or collaborators. Designing digital twin systems for safety is a major cornerstone, especially as modern industrial settings increasingly foster the need for human-machine collaboration. Creating systems that "reduce short-term memory load" [10] for human operators and transfer that burden to the digital twin/AI
model could prove to be a significant advancement in creating safer working environments on the manufacturing floor.

![Fig. 3.2. The use of DTs to remotely update feedback controllers [11]](image)

3.3.2 Healthcare, Personalised Medicine and Digital Twins

In the field of modern medicine, the potential for Digital Twins and AI to revolutionize healthcare is undeniable. The concept of "personalized" medicine has gained momentum, aiming to tailor treatments to individual patients based on their unique characteristics. It is a well-known fact that a significant percentage of patients with common diseases do not respond effectively to standard medication [12]. The complexity of diseases and the genetic variations between patients necessitate more precise and targeted treatment approaches [13].

With the application of Digital Twins in personalized medicine, the goal is to model the patient as completely as possible, creating a computational replica of the patient. This virtual representation enables healthcare professionals to simulate and test thousands of drug combinations and interactions quickly and cost-effectively. Through this process, diagnostics and treatment plans can be optimized to suit individual patients' needs, potentially improving treatment outcomes and reducing adverse reactions to medications.

Moreover, the integration of Digital Twins and AI in healthcare has led to various other applications, such as telemedicine, hospital management, and wearable technologies [14]. Telemedicine has seen significant advancements, allowing patients to access medical consultations and services remotely, thereby reducing the need for physical visits to healthcare facilities. Hospital management has also benefited from Digital Twins, with real-time monitoring of hospital resources, patient flow, and equipment utilization, leading to improved operational efficiency and patient care.
Wearable technologies, such as fitness trackers and health monitoring devices, are increasingly becoming part of individuals' daily lives. These wearables collect valuable health data that can be integrated with Digital Twins to provide personalized health insights and recommendations. Furthermore, with the advent of 5G networks and Internet of Things (IoT) devices, the potential for leveraging Digital Twins in healthcare is bound to expand even further.

One of the greatest socio-ethical benefits of Digital Twins in healthcare is realized in the area of personalized medicine. The ability to provide better prevention, treatment, and understanding of diseases through Digital Twins has a positive impact on individual happiness [15]. Improved prevention and early detection of diseases can lead to better health outcomes, reduced healthcare costs, and an overall improvement in the quality of life for patients. Additionally, Digital Twins can play a crucial role in drug discovery and development, accelerating the process of bringing new medications to the market and addressing unmet medical needs.

3.3.3 Digital Twins and Smart Cities

The concept of digital models for smart cities has evolved significantly since its inception, transitioning from 3D visualization tools [16] to "living" replicas animated by vast arrays of sensor data. Digital Twins, coupled with AI-based technologies, have enabled several smart city applications that contribute to enhancing the quality of life for urban residents.

Water monitoring is one such application where Digital Twins play a vital role. By integrating real-time data from sensors installed in water distribution systems, Digital Twins can analyze water quality, detect leakages, and optimize water usage, thereby promoting water conservation and efficient resource management. Predictive policing is another compelling application, where Digital Twins can assist law enforcement agencies in predicting and preventing crimes based on historical data and current trends.

Occupancy detection, counting, and tracking are instrumental in smart city planning and management. Digital Twins can help monitor the utilization of public spaces and transportation systems, leading to better urban planning and resource allocation.

Smart city applications have led to numerous possibilities for improving social welfare by optimizing resource allocation, load forecasting, and timely maintenance of city-wide utilities. For instance, optimized resource allocation ensures that essential services, such as public transportation and emergency response, operate efficiently and cater to the needs of the urban population. Load forecasting allows cities to predict and manage peak demands for utilities, such as electricity, ensuring a stable and reliable supply.
Digital Twins also play a significant role in public information dissemination and governance transparency. By engaging citizens in the decision-making process, Digital Twins foster participatory governance, allowing residents to influence public policy through virtual feedback loops [17]. Citizens' feedback can be used to experiment with different policy scenarios and assess their impact on various aspects of the city, contributing to evidence-based policy-making.

3.4 Digital Twin Access – Model and Data Sharing

In the era of Digital Twins, ensuring seamless access and sharing of DT models and data is a critical aspect for enabling collaboration and innovation. Several efforts have been made in academia and commercial sectors to facilitate the distribution and networking of Digital Twins.

Twinbase, an open-source Digital Twin Web server proposed by J. Autiosalo et al. [18], is one such initiative that promotes controlled sharing of Digital Twin documents over the web. The server stores publicly available DT meta-level description documents in a Git repository, which can be modified using Git workflows. This enables users to access Digital Twin features via static web pages or client libraries, enhancing accessibility and collaborative usage.

Commercial solutions, such as Microsoft's Azure Digital Twins [19], and open-source projects like Platform Industrie 4.0's Asset Administration Shell (AAS) [20] and W3C's Web of Things Thing Description (WoT TD), aim to standardize the description of Digital Twin metadata. By establishing standardized protocols for describing Digital Twins, these initiatives aim to achieve interoperability, allowing Digital Twins to seamlessly integrate across various platforms and industries.

Additionally, blockchain technology has emerged as a promising solution for secure data sharing of Digital Twins among multiple, non-trusting parties [21]. Blockchain's decentralized and immutable nature provides an ideal framework for ensuring data integrity, privacy, and controlled access. M. Dietz et al. [21] proposed a distributed ledger approach to secure data sharing, emphasizing the maintenance of confidentiality and integrity of shared data throughout the DT asset's lifecycle. By using a Distributed Ledger and Distributed Hash Table (DHT), stakeholders can jointly own and manage the DT asset, maintaining control over access rights and data manipulation.

Moreover, the concept of digital twin sharing platforms based on blockchain technology has shown promise in promoting cyber collaboration and knowledge exchange. M. Li et al. [22] presented a blockchain-based digital twin sharing platform that fosters software copyright protection and simplifies the integration of manufacturing resources in decentralized and distributed environments. By utilizing blockchain
technology, this platform facilitates communication and exchange of knowledge, bridging the gap in infrastructure intelligence between leading companies and small and micro-businesses in terms of cyber-physical-social collaboration.

3.5 Digital Twins Packaging, Distribution and Deployment – OCI Prospects

The studies "Performance Evaluation of Container Runtimes" [23] and "Using Docker in High Performance Computing Applications" [24] collectively underscore the evolving landscape of cloud computing, emphasizing the shift towards containerization technologies, particularly Docker, and their impact on high-performance computing (HPC) applications. These investigations reveal the inherent strengths and weaknesses of containerized environments compared to traditional virtual machines (VMs) and the implications for deploying digital twins and HPC applications.

The "Performance Evaluation of Container Runtimes" paper delves into the performance nuances of containerd and CRI-O container runtimes with OCI-compliant runtimes like runc and gVisor. It highlights containerd's superior performance in CPU utilization, memory latency, and scalability, which are critical for the resource-intensive operations typical of digital twins. This study not only showcases the efficiency of containerization in cloud environments but also points towards containerd as a viable runtime for deploying digital twins, owing to its performance and resource management capabilities.

On the other hand, the "Using Docker in High Performance Computing Applications" paper expands on the application of Docker containers in HPC scenarios, demonstrating Docker's advantages in terms of reduced overhead and resource efficiency compared to VMs. It presents Docker as a promising solution for HPC applications, offering significant performance improvements and resource utilization benefits. The ability of Docker to share the OS kernel and libraries with the host, while ensuring isolated execution environments, aligns with the needs for deploying scalable and efficient digital twins.

Integrating insights from both papers, it becomes evident that container technologies, particularly Docker, represent a compelling shift in the deployment of HPC applications and digital twins. Docker's lightweight nature, combined with the performance and scalability offered by container runtimes like containerd, underscores the practicality of containers in meeting the demands of digital twin simulations and HPC workloads. This synergy between container technology and the requirements for digital twins highlights a clear conclusion: containerization, spearheaded by Docker and supported by OCI standards, stands as a robust foundation for the future of cloud-based simulations and computations.
The Open Container Initiative's role in standardizing container formats and runtimes ensures interoperability and portability across different cloud platforms, further solidifying the position of containers as the go-to technology for developing, deploying, and managing digital twins in the cloud. As such, the convergence of these studies with the OCI's efforts points to a future where cloud computing, powered by containerization, enables more efficient, scalable, and secure digital twin deployments.

### 3.6 DT Data Sharing and Usage Control

The integration of Digital Twins and AI-powered technologies into various sectors highlights the increasing importance of these innovations. Alongside their growth, however, rises a concern critical to their sustainability and ethical application: the secure and ethical management of data. Establishing a framework that ensures precise control over digital resource access is essential for fostering trust and compliance in the era of Digital Twins.

Jones et al. have significantly contributed to the conversation on Digital Twins by identifying key research gaps and future challenges, one of which centres on Data Ownership [6]. This concern is increasingly relevant in today's digital landscape, marked by heightened awareness of data ethics, personal information security, and the commercial implications of data use. The European Union's General Data Protection Regulation (GDPR) [26] exemplifies legislative efforts to address these concerns, imposing strict penalties for data breaches and non-compliance. The GDPR's definitions of "Personal Data" and the stipulations regarding "Consent" from data subjects—those individuals whose data are being processed—underscore the critical need for clarity and control in data handling practices.

While traditional access control methods, such as Role-Based Access Control (RBAC) [27], provide a structured framework for managing user permissions and roles, they fall short in addressing the nuanced requirements of data protection and privacy. Although RBAC systems offer a degree of flexibility and simplicity in managing system resource access, the evolving digital landscape demands a more comprehensive model. This model must not only delineate access rights but also clearly define the rights and responsibilities of all contractual parties involved in data handling and usage.

To bridge this gap, it is imperative to move beyond conventional security models and embrace frameworks that are capable of addressing the complexities of data ownership, privacy, and ethical use. Such frameworks should not only enforce access rights but also ensure that all parties adhere to ethical data usage principles, thereby safeguarding personal information in the context of Digital Twins and beyond.
The UCON$_{ABC}$ Usage Control Model, proposed by Park and Sandhu [28], introduces a more comprehensive framework that integrates Authorizations (A), oBligations (B), and Conditions (C) for usage control. By incorporating subject attributes, object attributes, obligations, and conditions into the decision process, UCON$_{ABC}$ enables richer and finer control capabilities.

One of the key advantages of the UCON$_{ABC}$ model is the consideration of attribute mutability as a consequence of access. In traditional access control, attribute changes were primarily the result of administrative actions. However, the UCON$_{ABC}$ model allows certain subject or object attributes to be updateable (mutable) before, during, or after usage, enhancing flexibility and adaptability in dynamic environments.

The concept of privacy-sensitive and non-sensitive objects is crucial in the UCON$_{ABC}$ model. Privacy-sensitive objects (PSOs), similar to the EU’s definition of personal data, contain individually identifiable information that requires careful handling to prevent privacy breaches. On the other hand, privacy non-sensitive objects do not raise similar privacy concerns and are less subject to control restrictions. By classifying objects based on their sensitivity, UCON$_{ABC}$ ensures a balanced approach to control and privacy protection.

Additionally, the concept of a reverse UCON system is introduced to address the mutual protection of rights between consumers, providers, and identifye subjects. In this context, consumers may also have obligations towards the provider, creating a dynamic interplay of rights and responsibilities among all parties involved.

Implementing a UCON system that ensures adequate control mechanisms on the rights and usage of rights for all three subjects—consumer, provider, and identifye—can be further strengthened by involving third-party entities. Third-party management of the UCON system helps establish trust and fairness, providing an impartial approach to control and privacy protection.

3.7 Summary

In conclusion, Digital Twins have evolved from virtual representations of physical systems to transformative technologies with applications across diverse domains. In Industry 4.0, Digital Twins, coupled with AI, have revolutionized manufacturing by enabling real-time monitoring, predictive maintenance, and enhanced worker safety.

In healthcare, the concept of personalized medicine has gained momentum through the integration of Digital Twins, allowing for computational replicas of patients to optimize diagnostics and treatment plans. Smart
cities have also benefited from Digital Twins and AI, leading to improved resource optimization, public safety, and participatory governance.

To enable seamless access and sharing of DT models and data, efforts have been made to standardize the description of Digital Twin metadata, such as with Twinbase and Azure Digital Twins. Blockchain technology has emerged as a promising solution for secure data sharing among multiple, non-trusting parties.

The Linux Foundation's Open Container Initiative (OCI) is pivotal in streamlining the lifecycle management of Digital Twins, offering a unified standard for their packaging, distribution, and deployment. Its lightweight architecture ensures efficient operation, while its performance capabilities support dynamic Digital Twin environments. OCI's scalability facilitates adaptation to varying operational demands, and its portability guarantees seamless transitions across different computing environments. This comprehensive framework not only enhances the efficiency and adaptability of Digital Twins but also fosters innovation by broadening their applicability across various sectors.

The UCON\textsubscript{ABC} Usage Control Model presents a comprehensive approach to access control, ensuring fine-grained control over digital resources and privacy protection. As the field of Digital Twins continues to advance, embracing these robust frameworks for sharing and control will be essential to harnessing the full potential of this transformative technology across industries and communities. Table 3.1 presents a summary of the reviewed papers, publications and articles.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Conclusions and Discoveries</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review of Digital Twins</td>
<td>Insights into the origins of Digital Twins, related concepts and different realisations of the digital twin, current trends in the application of DTs.</td>
<td>15</td>
</tr>
<tr>
<td>Review of Digital Twin Data Sharing</td>
<td>Insights into various attempts in academic literature at the sharing of digital twins artifacts.</td>
<td>5</td>
</tr>
<tr>
<td>Review of OCI Containers and Images as deployment alternatives to Virtual Machines for cloud-based apps.</td>
<td>Insights into the performance, scalability, portability and interoperability benefits of containers and container runtime environments.</td>
<td>3</td>
</tr>
<tr>
<td>Review of Data Ownership, Rights and Access Management</td>
<td>Insights into the laws governing the handling of processing of user data. The relevance of Role-Based Access Control (RBAC) and the significance of the Usage Control (UCON) security models.</td>
<td>4</td>
</tr>
</tbody>
</table>
CHAPTER 4

DESIGN AND ARCHITECTURE

This chapter provides an overview of the proposed architecture. It delineates the various components, which are distributed across different application layers. These components are meticulously designed to offer a robust system with scalability at its core.

The architecture of our Digital Twin Management Platform is crafted as a web-enabled system utilizing a microservices approach. The adoption of the HTTP (Hypertext Transfer Protocol) with REST (Representational State Transfer) architecture ensures a lightweight, stateless, and scalable interaction model among services, which is crucial for the system's robustness and efficiency.

HTTP, the backbone of data communication on the web, offers a well-established, platform-independent protocol that enables our microservices to communicate over the internet. The use of REST architectural principles on top of HTTP further enhances our platform's scalability and modularity by allowing each microservice to be developed, deployed, and scaled independently.

REST's stateless nature allows each microservice interaction to be self-contained, with requests from client to server containing all necessary information, thus simplifying the server design and enhancing scalability and reliability. By leveraging standard HTTP methods (GET, POST, PUT, DELETE), RESTful services can easily perform Create, Read, Update, and Delete (CRUD) operations, aligning seamlessly with the needs of a digital twin management system that requires dynamic and responsive data manipulation.

In contrast to other communication protocols, HTTP and REST provide the flexibility needed to support a wide range of client devices and to integrate with external systems and services. They facilitate easier debugging, a vast ecosystem of tools, and the ability to cache requests for better performance. Furthermore, the security model of HTTPS, which encrypts data in transit, can be readily applied, providing a secure channel for sensitive operations and data transfer.

On the next page, we offer an in-depth exploration of each architectural layer.
4.1 Layers of Abstraction

At the centre of our architecture are the layers of abstraction, which structure the services available at each level of our system. This hierarchy starts with user-facing features and culminates in the foundational RDBMS data infrastructure.

4.1.1 Client Layer

The Client Layer represents the interface through which users directly engage with our application. This solution serves a browser-based web application, while also offering support for a native mobile application as well. These clients communicate with the system's back-end via the API Gateway Layer. The client layer represents the front-end of the solution from which users interact with the platform and their DTIs exposed as RESTful endpoints.
4.1.2 API Gateway Layer

The API Gateway Layer acts as the entry point to our system's back-end. Comprising a proxy service and a cloud caching service, this layer plays a pivotal role in managing request routing, authentication, authorization, and other cross-cutting business concerns. Its primary function is akin to that of a reverse proxy, efficiently directing incoming requests to the relevant microservices.

The cloud caching service is a crucial component within this layer, leveraging Redis as the technology of choice. This cloud cache stores frequently requested data, including encrypted authentication and authorization security tokens, as well as short-lived user-enforced DT model policy definitions. The utilization of cloud-based caching greatly enhances system responsiveness, leading to an improved overall user experience.

**Proxy Service**

The Proxy Service, implemented in Rust, is a microservice specifically dedicated to managing event routing from the front-end applications to the appropriate business services within our microservices architecture. This service collaborates with the "User Management Microservice" in the Application Layer to handle the generation and maintenance of session security tokens, data hashing, encryption, and decryption. It also routes user-enforceable policies generated by the "Model & Policy Management Microservice” to the “Cloud Cache Service” to improve runtime performance.

Additionally, frequently requested but slowly-changing data is routed by this service to our Cloud Cache service, which relies on Redis as a cloud storage solution.

**Cloud Cache Service**

The Cloud Cache Service is an open-source, Redis-based, in-memory storage solution. It acts as a distributed, in-memory key-value database and cache. This service is designed to store frequently requested but slowly-changing data, delivering low-latency read and write operations to enhance application performance and user experience.

In the "Smart Home" scenario, the system often handles non-state-changing requests, such as retrieving the current energy output from solar panels or monitoring the consumption level of the heat pump. To enhance responsiveness, these types of GET requests can be effectively cached within the architecture. By setting predetermined intervals for cache refreshes, the system ensures that users receive timely updates while optimizing the overall efficiency and speed of response to user inquiries.
4.1.3 Application Layer

The Application Layer is the core of our system's business logic. This layer orchestrates the flow of calls between different services, manages data transformations, and facilitates seamless communication between various system components. It is comprised of a collection of small, autonomous, WebAssembly (WASM)-enabled services, each designed to handle specific aspects of business logic and domain functionality.

Audit & Logging Microservice

The Audit & Logging Microservice plays a pivotal role in maintaining a comprehensive log of "system" events in its corresponding database. These "system" events encompass operations or actions triggered in response to user requests, each of which may be relevant for tracking system use and state changes. The data recorded by this microservice and stored in its database becomes a valuable resource for future queries and analysis, supporting functions such as abuse detection, SLA tracking, or usage billing.

Every instance when a new Digital Twin (DT) is generated for devices such as the heat pump, solar panel, furnace, or any other component within our Smart Home suite, a corresponding log entry is created. This logging also captures key operational events, such as an attempt to modify temperature readings or compressor velocities to dangerously high values, ensuring a comprehensive record of significant activities related to each entity.

User Management Microservice

The User Management Microservice serves as the micro-application within our framework responsible for all user-provisioning activities. This includes user registration/signup, sign-in, authentication, authorization, and general system access control. This microservice embodies the incorporation of Role-Based Access Control (RBAC) security architecture within our framework. The specific roles and corresponding responsibilities are detailed in Table 4.1, as outlined below.

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>Digital Twin Model Owners. Members of this group, who are often the owners of the models, own the Intellectual Property rights of the DT Model.</td>
<td>Deploy and Configure Model on DT Platform Publish and Unpublish Models Define Data Sharing and Access Policies Deactivate DT Instances of errant users</td>
</tr>
<tr>
<td>User</td>
<td>Digital Twin Subscribers. Members of this group subscribe to published DT Models. They are owners of DT instances, thus making them owners of DT usage data.</td>
<td>Create, Start, Stop and Delete Twin Instances Interact with owned DT instances Define Data Sharing for PSOs</td>
</tr>
<tr>
<td>System Admin</td>
<td>Platform Administrator. Manages and administers the platform.</td>
<td>Manages all platform-related operations and support requests.</td>
</tr>
</tbody>
</table>
DT Model & Policy Management Microservice

The DT Model and Policy Management Microservice is dedicated to managing the business logic required to set up and define a digital twin resource or asset on our management platform. Utilizing the "API Gateway" Layer's "Proxy Service," it empowers DT model owners to define the source of DT OCI images, configure runtime properties that allow for model personalization, and establish the necessary policies governing the access and use of digital twin resources.

In the Smart Home context, certain model properties facilitate the pairing of a Digital Twin (DT) with its corresponding physical entity. These properties include establishing a link to the IoT hub for our diverse smart home devices, setting the frequency at which a user can query a DT, and defining a range of permissible operational data values which are essential for controlling or modifying the state of a specific DT instance.

Twin Management Microservice

The Twin Management Microservice provides the set of features required to instantiate, and personalize – through configuration – a digital twin model instance, necessary for achieving a personalized connection between a specific DT instance and its physical counterpart, where required/applicable. This microservice provides the functionality required to create the logical composite we call our “Smart Home” DT by allowing for the combination of several atomic DTs into a single unit.

Container Runtime Management Microservice

The Container Runtime Management Microservice is tasked with handling the interaction with specified container runtime environments where deployed models reside. It functions as the conduit for routing HTTP requests to the container runtime engine, which may be provided by either the platform owner or the model owner. This microservice oversees the management of created containers, networks, and other container-related operations conducted remotely.

Parameter Checker Microservice

The Parameter Checker Microservice plays a crucial role in enhancing the safety and security of Digital Twin (DT) Models. It functions as a validation layer, ensuring the use of valid and secure query parameter values during interactions with a DT asset. This service operates in tandem with the Audit & Logging Microservice, triggering logs whenever there are violations of parameter value specifications. Such logging is vital for identifying and addressing potential abuses of the system, thereby maintaining its integrity and reliability.
Data Aggregator Microservice

The Data Aggregator Microservice is designed to aggregate results of all DT instances that are defined as members of an aggregate/composite DT. The module is a calculation engine that applies an “aggregation instruction” set specified in JSON format on the array-based collection of combined DT instance outputs. The aggregation instruction is either manually specified or generated from natural language as part of the services of the “AI Augmentation Microservice”, and the aggregated result represents a relation among the participating atomic DTs.

An example of our Smart Home's composite Digital Twin (DT) could be the aggregated measure of the home's energy efficiency. This metric would be derived by processing outputs from the heat pump, solar panels, and furnace atomic DTs. The process would follow the predefined aggregation instructions that reflect the cumulative impact of the various interacting components within our Smart Home system.

AI Augmentation Microservice

The AI Augmentation Microservice significantly enhances the value obtained from Digital Twin (DT) assets. This microservice integrates with a Generative AI engine, enriching the responses obtained from DT models. For instance, within the interface of our heat pump Digital Twin (DT), we could engage with the integrated AI to analyze the heat pump's output. We could explore various input parameters and solicit recommendations on their optimal values to attain specific objectives, such as adjusting the combination of variables in our heat pump model to heat our home more energy-efficiently.

This module also helps translate aggregate/composite DT instance “aggregation rules” into JSON-formatted “aggregation instructions”, containing a set of mathematical operations and operands, used by the “Data Aggregator” microservice to compute the results of logically combining several DT instances. By leveraging the advanced capabilities of Artificial Intelligence and Large Language Models (LLMs), this service expands the functionalities of our DT management platform, providing users with richer, more insightful interactions with DT assets.

The aggregation rule below representing the Smart Home DT is translated into JSON format as shown in Fig. 4.2 with a process flow representation shown in Fig. 4.3.

```
Generate a JSON instruction set for a data processing operation with the following specifications defined as the relationship among the heat pump, solar panels, and electric furnace DTs as the 'Energy Efficiency' of our Smart Home. The energy efficiency is calculated as follows –

`Energy_Efficiency = (summation(power_output)*panel_efficiency) – ((power_consumption * (running_time_seconds / 60))) * (desired_temperature - indoor_temperature)`
```
While not part of the current implementation, it's worth mentioning other microservices that can be incorporated into our solution:
Abuse Detection Microservice
The Abuse Detection Microservice is designed to monitor the use of DT models, assess policy and SLA compliance, and provide alerts and advice to the platform provider or model owner in cases of user violations that could potentially constitute an abuse of the system.

SLA Management Microservice
The SLA Management Microservice offers features for defining service level agreements (SLAs) that govern the access and usage of DT resources. SLAs essentially serve as contractual agreements between model owners, platform providers, and model users. These agreements can vary in complexity, depending on the specific requirements of the contracting parties.

Billing and Payment Management Microservice
The Billing and Payment Management Microservice is responsible for managing billing and payment information associated with accessing specific DT models.

In an ideal setup, each microservice is designed to have its standalone database. This approach promotes decoupling, scalability, and ease of maintenance, facilitating independent development, deployment, and scaling. In practice, our current implementation employs a simplified approach, utilizing a single centralized database for all available microservices.

The rest of the discussion of our solution architecture assumes this level of separation of microservices (i.e. each existing with its own standalone databases, and other subsequent abstractions), as shown in Fig. 4.1.

4.1.4 Data and Information Layer
The Data and Information Layer is a critical component of our architecture, encompassing a collection of Relational Database Management Systems (RDBMSs). This layer offers persistent storage for application data. In alignment with microservices principles, each microservice ideally has its database to maintain data integrity within its specific domain. DT usage data is persisted, when indicated, based on policy settings collaboratively defined by the model owner, and the DT user.

4.2 Design Considerations
We delve into the foundational design considerations that shaped our architecture, which were instrumental in driving our decisions. Understanding these ensures the long-term efficacy and adaptability of our solution.
4.2.1 Standardization Considerations

A principal aim of this study is to forge a path toward a universally accepted standard for the packaging and dissemination of Digital Twin Models and Instances. In this pursuit, the Open Container Initiative (OCI) serves as a pivotal framework. It establishes a lean, transparent governance model dedicated to formulating open industry standards encompassing container formats and their runtime environments.

The adoption of OCI containers signifies a leap forward in deploying digital twins, providing a uniform environment that ensures consistency across different computing environments. This is instrumental for digital twins, which require reliable and repeatable deployment methods to mirror and predict the state of physical assets accurately. OCI containers encapsulate the digital twin's software, libraries, and all its dependencies in a compact, isolated package. This encapsulation not only simplifies version control and distribution but also streamlines the update process, allowing for seamless evolution of digital twins in tandem with their physical counterparts.

Compared to traditional Virtual Machines (VMs), OCI containers offer a litany of deployment advantages that are particularly conducive to the dynamic nature of cloud computing:

- **Reduced Overhead**: OCI containers share the host system's kernel, rather than requiring a full operating system for each instance, significantly reducing the computational overhead. This lightweight nature translates to a higher density of containers per server and minimizes the resource footprint.
- **Faster Startup Times**: Containers can be started almost instantaneously, as there is no need to boot an OS. This rapid startup capability is vital for digital twins, allowing real-time responsiveness and agility in operational environments.
- **Improved Performance**: By eschewing the need for hypervisors and running directly on the host OS, containers can offer performance that is close to bare-metal servers, ensuring that digital twins operate with minimal latency and maximal efficiency.
- **Portability Across Environments**: OCI containers can run consistently across different platforms, from a developer's laptop to production servers, regardless of the host environment. This is crucial for digital twins, which may be developed in diverse ecosystems but require uniform operation.
- **Fine-grained Scaling and Management**: Containers enable microservice-based architectures, where services can be independently scaled. This granular control is advantageous for managing the complex, interrelated services that comprise digital twin ecosystems.
- **Resource Isolation**: Although containers are less isolated than VMs, modern container runtimes provide robust isolation features that can be almost as effective as VMs for certain use cases, without the associated overhead.
- **Ecosystem and Tooling:** The container ecosystem offers an extensive suite of tools for orchestration (e.g., Kubernetes), security, monitoring, and networking, which are continually evolving and improving, thanks to the open-source community's contributions.

In summary, the shift toward OCI containers for digital twins represents not only an alignment with cloud-native principles but also a strategic decision to leverage a flexible, efficient, and scalable method for the standardization and operational management of digital twin technologies.

### 4.2.2 Performance Considerations

Introducing a middleware service for the sharing of digital twin models/assets inevitably adds layers of complexity. A primary objective, therefore, is to optimize latency during request lifecycles. The selected technologies are chosen both for their alignment with the middleware service's principles and their ability to maintain system responsiveness.

While the overall system architecture has been designed to emphasize performance, the following considerations have been made based on the foregoing:

- **Rust Programming Language**
  
  Rust is a multi-paradigm, general-purpose programming language designed with an emphasis on performance, type safety and concurrency. It enforces memory and thread safety through its rich type system and ownership, discarding the need for automated memory management techniques such as garbage collection. This, and the absence of a runtime, makes Rust very fast and memory-efficient – making it a prime candidate for critical services run on embedded devices. Given its focus on performance and safety, Rust becomes indispensable when building services that demand responsiveness and reliability, especially in an environment with embedded devices.

- **WebAssembly (WASM)**
  
  WebAssembly (Wasm for short) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages such as Rust, enabling deployment on the web for client and server applications. The Wasm stack machine is designed to be encoded in a size- and load-time-efficiency binary format, with the aims of executing at native speed by taking advantage of common hardware capabilities available on a wide range of platforms. Its memory-safe, sandboxed execution environment now finds uses in web applications, for example embedded in Web browsers, as well as non-web embedded environments.
WebAssembly, with its emphasis on speed and portability, is crucial for ensuring that applications maintain native-level performance across diverse platforms. Its integration with languages like Rust also facilitates a streamlined deployment process.

- **WasmEdge**
  WasmEdge is a lightweight, high-performance, and extensible WebAssembly runtime for cloud-native, edge, and decentralized applications. The WasmEdge Runtime provides a well-defined execution sandbox for its contained WebAssembly bytecode program. The runtime offers isolation and protection for operating system resources (e.g. file system, sockets, environment variables, processes) and memory space.

In our architecture, WasmEdge's lightweight and high-performance runtime is pivotal in maintaining speed and ensuring the secure execution of WebAssembly bytecode.

- **Docker + Wasm**
  Docker Desktop now ships with the WasmEdge Runtime embedded. This allows us to build, share and run very lightweight containers (i.e., a “scratch” empty container with only the .wasm file without any Linux OS libraries or files) through the Docker tools. Docker containers built for Wasm are 10 times comparatively smaller in size to a Linux container, and they startup 10 times faster as the Wasm containers do not need to bundle and start Linux libraries and services.
  This synergy between Docker and Wasm epitomizes the blend of efficiency and performance. Lightweight Wasm containers further streamline the middleware's responsiveness, emphasizing our commitment to a fast and efficient system.

- **Redis Cache**
  Redis, an in-memory data structure store, provides rapid data access, making it ideal for use cases requiring high-speed operations, such as caching, real-time analytics, and session storage. Additionally, Redis supports various data structures like strings, hashes, lists, and sets, enabling versatile data operations. Its ability to act as a message broker further enhances its applicability in diverse scenarios. In our system, Redis acts as the go-to for rapid data access, supporting high-frequency operations that are paramount for ensuring smooth user experiences.

### 4.2.3 Deployment and Scalability Considerations

Anticipating future demands and ensuring the middleware's adaptability is paramount. Additionally, seamless integration with CD and CI pipelines ensures that our solution remains agile and up-to-date.
• Decomposition into a Microservices Architecture

The decomposition of our solution into smaller, independent services, each responsible for specific functionality, while communicating with each other over a network using the HTTP/REST protocol, enhances the scalability of our solution. Asides facilitating parallel development within large teams and supporting the use of diverse technologies, each individual service can be scaled autonomously, based on existing demands. The distributed nature of horizontally scaling microservices often means there is built-in redundancy, where if one service fails, others continue to function, ensuring higher availability.

By adopting a microservices approach, we not only ensure scalability but also resilience. The system can adapt to surges in demand, and any potential service disruptions become localized, ensuring a seamless user experience.

• PostgreSQL RDBMS

PostgreSQL is an advanced, open-source relational database that supports both SQL (relational) and JSON (non-relational) querying. Its robustness, extensibility, and adherence to SQL standards make it a preferred choice for applications requiring complex queries and transaction safety. PostgreSQL also offers advanced data types, powerful indexing techniques, and support for extensions, catering to a wide array of use cases.

Our choice of PostgreSQL stems from its versatility. As our system juggles diverse data demands, PostgreSQL's advanced querying and indexing mechanisms ensure data integrity and quick retrievals.

By understanding our system's architectural nuances and the rationale behind each design choice, we gain a comprehensive appreciation of its strength and adaptability. Our Digital Twin Management Platform, fortified with advanced technologies, stands out as a solution that's primed for the future – scalable, efficient, and dependable.
CHAPTER 5

IMPLEMENTATION

5.1 Introduction

In this chapter, we describe the practical aspects of realizing our digital twin management platform. Building upon the architectural blueprint laid out in Chapter 4, we embarked on a journey to transform theoretical concepts into a robust, functional system. The core objective was to address the challenges identified in the problem statement: ensuring the protection of intellectual property rights of model owners, the enforcement of user-defined policies as it relates to the creation and use of personalised instances of a digital twin artifact, considerations for data storage and sharing, and the need to have these, and other functionalities, made available to a digital twin model user without the introduction of prohibitive latencies/delays in digital twin utilization. Other ancillary benefits realised in this implementation include reducing costs of digital twin access/ownership, simplifying functionalities, and mitigating the risk of vendor lock-in, while ensuring data integrity and privacy.

![Digital Twins Management Platform (DTMP) Landing Page](image)

**Fig. 5.1.** Digital Twins Management Platform (DTMP) Landing Page

5.2 Development Environment and Tools

Our development began with the selection of Rust as the primary programming language. Chosen for its emphasis on performance, safety, and concurrency, Rust was particularly suited for our middleware,
designed to efficiently proxy user requests to executable digital assets. The need for high responsiveness in our environment was a critical factor in this decision.

For containerization, we opted for Docker, which provided a consistent runtime environment across various deployment scenarios. This choice greatly facilitated our development and testing processes, allowing for rapid prototyping and iterative improvements.

A key consideration for our middleware was performance optimization. To this end, we compiled several microservices into WebAssembly (WASM) modules. These were then packaged in Docker containers running on the WasmEdge runtime, combining efficiency with the lightweight nature of WebAssembly.

Our project harnessed a robust suite of tools and libraries from the Rust ecosystem, including:

i. Tokio: An asynchronous runtime for Rust, crucial for building fast, scalable network applications. Tokio's asynchronous code execution is essential for efficiently managing numerous network requests.

ii. Hyper: An HTTP library for Rust, ideal for creating high-volume HTTP servers and clients. Built on Tokio, Hyper enhances our capability to handle concurrent HTTP requests, forming the core of our API or service layer.

iii. Warp: A web server framework that works seamlessly with Tokio. Known for its simplicity, Warp is used for setting up RESTful APIs, allowing easy creation of routes and handlers in our web applications.

iv. Reqwest: This HTTP client, also built on Tokio, is utilized for making HTTP requests to external services and APIs, enabling integration with various web services.

v. Bollard: A Docker API client for Rust, Bollard is instrumental in interacting with Docker for container management and service orchestration, streamlining deployment, and automating Docker-related development tasks.

vi. Serde: A serialization/deserialization framework, Serde is indispensable for handling JSON data, a common format in web development. It is used for parsing incoming JSON data from requests and formatting responses.

vii. SQLx: An asynchronous, pure-Rust SQL crate compatible with multiple databases and built atop Tokio. SQLx is utilized for database interactions, allowing async execution of queries, data fetching, and transaction handling.

In our ecosystem, Tokio serves as the backbone for asynchronous operations, with Hyper and Warp handling HTTP server functionalities. Reqwest manages outbound HTTP requests, Bollard facilitates Docker integration, Serde handles data serialization, and SQLx takes care of database interactions. This
combination leverages Rust's performance and safety features, making it an exemplary choice for contemporary web development.

![Cargo.toml](image)

**Fig. 5.2.** A Screenshot of a List of Referenced Rust Libraries

### 5.3 Usage Scenarios

The implemented features and functionalities necessary to achieving our objectives would be presented through the view of two distinct access roles – Owner (DT Model creator/owner) and User (DT instance creator/subscriber/user) across several usage scenarios. At signup, a user is directed to an appropriate signup page depending on the role the user wishes to assume in the system. On successful signup, all users a given access through a single sign-in/login page.

![DT Management Platform](image)

**Fig. 5.3.** DT Model Owners Sign-Up Interface
5.3.1 DT Model Provisioning and Management

The platform exhibits the following features and functionalities specifically for users with the 'Owner' access role:

- **Capturing Basic DT Model Information**

  DT model owners are tasked with documenting fundamental details about their models. This includes providing a comprehensive description of the model, outlining its key parameters, and specifying the access points for various model services available through HTTP/RESTful endpoints. The information captured at this stage is critical as it equips users with the necessary knowledge to understand and effectively interact with any given DT model.

  Additionally, it's important to note the integration of an AI model based on the GPT (Generative Pre-trained Transformer) architecture in this context. The descriptions provided by the DT model owner lay the groundwork for setting the interaction context with this AI model. The GPT-based AI is adept at generating coherent and contextually relevant text, which enhances its ability to
respond to queries and prompts in a conversational manner. This augmentation significantly enriches the user experience, providing a more intuitive and informative interface for engaging with the DT model instance.

![DT Model Deployment Wizard Interface – Basic Model Setup](image)

Fig. 5.5. DT Model Deployment Wizard Interface – Basic Model Setup

During DT model provisioning/deployment, the DT model owner specifies the image reference to a containerized version of the Digital Twin Model, which serves as a template for creating Digital Twin Instances, stored in a cloud-based repository such as Docker Hub, Google Container Registry, and others. Image references are defined using the format “<username>/repository:tag”, where “username” is the Docker Hub username of the model owner, “repository” is the name of the repository, and “tag” specifies the version of the image. This crucial piece of data, as well as every single container instance spawned from this image are completely shielded from, and inaccessible to the DT model user, thereby fostering the protection and privacy of the DT asset which ties in with one of the goals of this project.

When deploying the Digital Twin (DT) model of the heat pump from our Smart Home case study, the model owner stipulates the following configurations:

- A suite of parameters that influence the DT's state, such as compressor speed, operating mode, and target temperature settings.
• An associated spectrum of acceptable values for these parameters—for instance, the compressor speed must be within a range from 0 to 100, adjustable in increments of 10, while the desired temperature should fall within 15 to 28 degrees Celsius.

• Specific twinning and environmental variables that are essential for binding a given instance of the digital entity with its real-world counterpart, such as the IoT hub address connected to the heat pump, or the identifiers for the sensors on the solar panels.

• **DT Model Policy Definition**

A critical aspect of this project is ensuring access to Digital Twin (DT) assets is governed by "usage policies" established by the DT Model owner. This component of our platform allows for the specification of owner-defined DT policies, grouped around two major strategies viz.

i. **Regulating Access to DT Services Exposed via HTTP/RESTful Service Endpoints:** This approach to access control and usage compliance involves a set of policies that control how frequently users can access DT resource services. These services are exposed through specific HTTP/RESTful endpoints. The policies define limits on the number of times these services can be accessed within a given period. An example policy could be: 'Users are allowed to query the heat pump’s “Get Energy Consumption” service endpoint up to 50 times per day.' Furthermore, these access policies are subject to periodic resets, as determined by the DT model owner. The reset schedule can be daily, weekly, monthly, or set to 'Never', which means the access limit, once reached, is not reset.

ii. **Regulating Access to DT Model and Twin Instances:** These rules determine the number of twin instances a user can create. An extension to this feature could be to impose limitations on their operational duration such as restricting the number of hours a twin instance can run or specifying the times of day when it is accessible. Essentially, these rules are designed to regulate the overall availability of the DT asset, ensuring balanced and fair usage among users.

![Fig. 5.6. DT Model Deployment Wizard Interface – Model Policy Setup](image)
- **Policy Violations and Access Denial**
  In our digital twin (DT) platform, any violations of access and usage policies lead to immediate denial of access to the respective DT asset. These incidents of policy breach are systematically logged within the system for potential retrospective actions. Such actions might include measures like user deactivation. Additionally, the accumulated data from these logged violations over time serves a crucial role in identifying patterns of systemic abuse. By analysing these trends, we can enhance the platform's security and integrity, ensuring fair and compliant usage by all participants.

- **DT Data Sharing and User Privacy**
  Throughout the lifecycle of a Digital Twin Instance (DTI), it generates and logs various types of 'usage' data on the platform, especially during user engagements with the system. Notably, this includes data from interactions with instances of the Digital Twin model, often in the form of state-altering user queries to the Digital Twin's service endpoints. The potential for sharing this data with the model owner is contingent upon the conditions established in the provisioning phase of the model. Typically, the data shared encompasses the values of various model parameters that users utilize when querying the model's service endpoints.

  In alignment with the UCON$_{ABC}$ Usage Control Model [28], our platform empowers users to designate specific data items as either PSOs (Privacy Sensitive Objects) or PNOs (Privacy Non-sensitive Objects) during the model provisioning process. This classification dictates data sharing permissions: PNOs are deemed automatically shareable, often viewed as the user's contributions towards future enhancements of the DT model, whereas PSOs are typically not shared by default.

  However, users retain the flexibility to modify these settings for PSOs during the lifespan of a DT instance, tailoring their privacy preferences as needed. In scenarios where a DT model does not necessitate data sharing, user queries to service endpoints are not logged for sharing. Conversely, when data sharing is active, any data items marked as sensitive are automatically obfuscated by the platform prior to storage.
DT Model Deployment Wizard

Parameter Definition

Name: Compressor Speed
Description: The speed of the heat pump's compressor
Data Type: Integer
Parameter Type: Multi-value
Is Sensitive?: No

Valid Values

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Param Type</th>
<th>Sensitive?</th>
<th># Valid Values</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compressor Speed</td>
<td>The speed of the heat pump's comp...</td>
<td>Integer</td>
<td>Multiple</td>
<td>false</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Operation Mode</td>
<td>The heat pump's operational mode...</td>
<td>String</td>
<td>Multiple</td>
<td>false</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.7. DT Model Deployment Wizard Interface – Data Sharing

- **DT Model Safety and Security Through Parameter-Checking**

In our digital twin (DT) platform, safeguarding the integrity and safety of DT assets encompasses more than just enforcing predefined policies. It also involves protecting against actions that may undermine the system's integrity. One potential risk involves utilizing information from DT asset probes to identify system vulnerabilities. Although the primary responsibility for ensuring DT safety and security lies with the DT model owner, who may implement security rules as suggested by Eckhart et al [29], our platform offers an additional layer of protection. This is achieved by enabling DT model owners to set limits on model parameter values, such as defining a minimum cooling temperature or specifying a range of compressor speed for a heat pump DT.

The “Parameter-Checker” microservice plays a crucial role during runtime. It parses parameter values in queries sent to a DT instance to verify if they fall within the predefined limits. Any attempts to submit values outside these boundaries are not only prevented but also logged for retrospective analysis. This log serves as a valuable input for an “abuse detection” algorithm or microservice, which further strengthens the platform's security against potential misuse or exploitation.
After successfully provisioning a digital twin (DT) model on the platform by specifying all basic model information, parameter values, and twinning/environment variables required, the model is made accessible to potential subscribers the owner by 'publishing' it on the platform.
Every model published by owners on our platform is showcased in the 'Marketplace' section. This feature allows visitors and potential subscribers to easily browse through the models and view basic information about each one. To access more detailed information about these models or to subscribe, visitors must sign up for a 'User' access role on the platform.

- **Instantiating a DT Model – Twinning**

  The subscription process for a published DT (Digital Twin) Model involves creating a personalized instance, effectively forming the user's own digital twin instance. Personalization of the DT model can be achieved in one of two ways:

  i. **Direct Copy**: This method creates a one-to-one replica of the original model. Instances generated in this manner are exact duplicates, sharing any specified environment variable settings and values with other instances of the same model. For users, the process is straightforward: a simple click on the 'Subscribe' button is all that's required.

  ii. **Configuration**: This approach not only copies the underlying model but also allows users to input specific, personalized values for any environment variables defined in the model. This customization is crucial for establishing a personalized connection between the user's DT and its physical counterpart, where applicable. In this case, the subscription and twinning process is more involved. The user is required to provide substitute values for some, or potentially all, environment variables. These variables, along with their corresponding values, become integral components of the twin container created at the end of this process.
Each digital twin instance – instantiated as docker containers - remains exclusive to its owner and is accessible via specific URL endpoints and port combinations allocated and managed by the platform.

**DT Instance Operations**

After subscribing to a published model, the user gains operational control over their digital twin (DT) instance. The platform supports several key operations throughout the asset's lifecycle:

i. **Starting a Twin Instance**: Users can initiate their DT instance by issuing the 'run container' command. This action transitions the model into an active, running state.

ii. **Using a Twin Instance**: Once the container is running, users can interact with various service endpoints defined in the model to utilize its functionalities and achieve the intended outcomes. Usage queries issued to a DTI are usually state-altering (HTTP POST/PUT) e.g. change the compressor speed, toggle the heating mode etc, or non-state-altering (HTTP GET) e.g. get the energy consumption of the heat pump.

iii. **Stopping a Twin Instance**: Users also have the capability to issue the 'stop container' command, which halves their DT instance. This feature is particularly useful for managing container or twin instance usage in accordance with policies that may include uptime quotas.

iv. **Removing/Deleting a Twin Instance**: This operation concludes the lifecycle of the specific DT asset instance. It involves deleting the provisioned OCI container and releasing the associated system resources. It's important to note that once a twin instance is removed, the
action is irreversible. However, the instance will still be listed on the user's dashboard for record-keeping purposes.

Fig. 5.12. User Interaction with a DT Instance Service

- **Creating an Composite DT Instance**

A major feature of interest in our application is the ability to logically combine several DT instances into one single aggregate. This offers the potential for monitoring the state of several individual DT instances and combining their output, based on a defined “aggregation rule” translated into a JSON-format instruction set, into a singular, coherent result that reflects the interrelationships among the member DT instances in our aggregate.

Composites are created by selecting specific services from owned DT instances exposed by their respective RESTful endpoints and specifying the rules by which the results from these individual services would be combined into a singular output. Under the current implementation, only DT instance services exposed as HTTP GET endpoints (non-state altering requests) can be combined into an aggregate. The payload for the creation of our composite DT is shown in Fig. 5.14.
5.3.3 DT Value Enhancement Through AI Integration

Until now, user interactions with digital twin (DT) model instances have primarily revolved around querying various service endpoints. Although the complexity of these DT models varies across different application domains and reflects diverse levels of abstraction and implementation by developers, the integration of Artificial Intelligence (AI) capabilities can significantly enhance the value derived from these interactions. Our platform introduces an innovative approach by integrating a Large Language Model (LLM) – a sophisticated AI system capable of understanding, generating, and interacting with human language. This integration is meticulously aligned with the context provided by the model owner during the model provisioning phase, thereby enriching the output without altering the DT model's in-built logic.

This AI integration facilitates a dynamic three-way information exchange among the user, the DT instance, and the LLM. It enables users to not only receive responses from the DT model instance but also to augment and expand these responses in relation to specific queries. In this interactive ecosystem, outputs from the DT instance, particularly in response to certain parameter inputs, act as contextual 'prompts' for the LLM. This synergy allows the LLM to provide a broader and more nuanced understanding of the subject matter under discussion.

Moreover, the LLM brings additional intrinsic benefits, such as advanced text generation, language translation, summarization of complex information, question-answering capabilities, and conversational interaction. These features collectively empower the LLM to function as an 'AI
Assistant’, offering users an enriched, interactive experience. This assistant can guide, clarify, and extend the insights gained from DT models, making the platform not just a tool for technical analysis but also a comprehensive solution for understanding and application in various contexts.

By embedding this LLM within our platform, we aim to bridge the gap between complex DT model outputs and user comprehension, thereby unlocking new dimensions of value and usability in the realm of digital twins.

Fig. 5.14. The Use of a Digital Twin Instance with AI Assistant Enabled

Another benefit of our integration with an LLM, as previously mentioned, lies in the value the AI model brings to our “Data Aggregator” microservice – the cornerstone of our platform’s ability to generate meaningful outputs from the logical combination of several DT instances. More specifically, the generation of parse-able JSON instructions from natural human language texts greatly enhances our application useability. An alternative to this approach would have been to either return the data in its raw form – passing the responsibility of translation to a meaningful output to the user, or lean heavily on the AI to parse and interpret every output from our aggregated twin instances, leading to high network chatter and increased platform cost.

5.4 Challenges and Solutions

Our implementation encountered a range of challenges, notably in achieving satisfactory performance, enhancing system scalability, maintaining data privacy, and simplifying deployment processes.
**Microservices Architecture**: The adoption of a microservices architecture proved instrumental in overcoming these obstacles. It allowed us to decompose the solution into a loosely coupled system of purpose-specific modules. This modularity enabled independent management, scaling, and updating of each module without causing a complete service outage, thus enhancing system scalability and resilience.

**WebAssembly Integration Challenge**: A significant challenge was our ambition to compile a substantial portion of our microservices into WebAssembly (WASM) modules. We encountered compatibility issues with several core libraries in the Rust ecosystem, which were not initially Wasm-compatible. This necessitated a strategic separation of functionalities: identifying those that could be effectively compiled into WASM and those that could not. As a result, over 60% of our solution was successfully compiled into WASM binaries running on the WasmEdge runtime. This approach not only optimized performance but also maintained the lightweight nature of our services.

**Architectural Adjustments and Testing**: To address these challenges, we implemented systematic architectural adjustments and conducted rigorous testing. These measures were crucial in ensuring that our platform not only met performance benchmarks but also maintained high standards of data privacy and ease of deployment. Our commitment to continuous evaluation and improvement resulted in a platform that is both robust and secure, capable of adapting to evolving needs and challenges.

5.5 **Summary**

This chapter highlights the successful transformation of the theoretical architecture into a practical digital twin management platform. The implementation emphasizes performance optimization, scalability, and user-centric features while ensuring data privacy and security. The integration of AI and careful consideration of user roles and policies significantly enhance the platform's capability, making it a robust and adaptable solution in the field of digital twin technology. Our platform represents a noteworthy contribution in the realm of digital twin sharing, providing a comprehensive, scalable, and secure solution that caters to the diverse needs of stakeholders within the digital twin ecosystem.

In the subsequent chapter, we will present the outcomes of our framework's evaluation, offering insights into its effectiveness and practical applicability.
CHAPTER 6

EXPERIMENTS AND EVALUATIONS

In this section, we evaluate the suitability of our solution by showing the results of some experiments with a view to seeing if our research objectives have been achieved, and if not, what future adaptations might be required to deliver on promised benefits.

6.1 Methodology and Experiment Setup

Adopting a “Quantitative” approach to evaluating our solution, with the use of JMeter, we evaluate the Latency, Throughput and Error Rate (measured as a percentage of failed transactions) of accessing various services of our platform under different transaction loads e.g. 100, 200, 300 ... 1,000 with a load ramp-up of 1 second. Results are computed as the averages of our measures of interest after several independent runs i.e. repeating our experiments for each workload a minimum of 3 to 5 times.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Scenarios</th>
<th>Arrival Rate (secs)</th>
<th>Sent bytes</th>
<th>Min. # Samples</th>
<th>Max. # Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 - Develop a Flexible System for Creating DT Instances</td>
<td>Scenario 1: Creating DTI using 1 instance of the Container Runtime MS</td>
<td>1</td>
<td>492</td>
<td>25</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>Scenario 2: Creating DTI using 2 instances of the Container Runtime MS</td>
<td>1</td>
<td>492</td>
<td>25</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>Scenario 3: Creating DTI using 3 instances of the Container Runtime MS</td>
<td>1</td>
<td>492</td>
<td>25</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>HTTP GET (Direct DT Access) Performance</td>
<td>1</td>
<td>199</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>HTTP GET (Proxy DT Access) Performance</td>
<td>1</td>
<td>507</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>HTTP POST (Direct DT Access) Performance</td>
<td>1</td>
<td>250</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>HTTP POST (Proxy DT Access) Performance</td>
<td>1</td>
<td>538</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Strategy 1: Accessing Composite DTs without Aggregating results</td>
<td>1</td>
<td>491</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Strategy 2: Accessing Composite DTs Aggregating results using LLM</td>
<td>1</td>
<td>486</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Strategy 3: Accessing Composite DTs Aggregating results using Data Aggregator MS</td>
<td>1</td>
<td>492</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

In JMeter, and performance testing in general, latency refers to the amount of time it takes for a request to travel from the client to the server and for the server to start sending a response, while throughput refers to the number of requests that can be handled in a given time frame.
**Application Performance Index (Apdex) Ratings**

We will also provide the Apdex ratings [30], a single metric showing the performance of our application and the errors that impact user experience.

Given a Tolerated threshold that we specify, the Application Performance Index returns a score between 0 and 1. The Apdex formula is the number of satisfied samples plus half of the tolerating samples plus none of the frustrated samples, divided by all the samples:

\[
\text{Apdex}_t = \frac{\text{SatisfiedCount} + (0.5 \cdot \text{ToleratingCount}) + 0 \cdot \text{FrustratedCount}}{\text{TotalSamples}}
\]

*Fig. 6.1. Apdex Score Calculation Formula*

The default Apdex ratings are shown in the table below.

**Table 6.2. Application Performance Index (Apdex) Default Ratings**

<table>
<thead>
<tr>
<th>Score</th>
<th>Performance level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.94–1.0</td>
<td>Excellent</td>
</tr>
<tr>
<td>0.85–0.94</td>
<td>Good</td>
</tr>
<tr>
<td>0.7–0.85</td>
<td>Fair</td>
</tr>
<tr>
<td>0.5–0.7</td>
<td>Poor</td>
</tr>
<tr>
<td>&lt; 0.5</td>
<td>Unacceptable</td>
</tr>
</tbody>
</table>

**Experiment Infrastructure**

The infrastructure used for our experiments was built on the Microsoft Azure Platform and the resource specifications are as detailed in the table below.

**Table 6.3. Experiment Infrastructure**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Type</th>
<th>Purpose</th>
<th>OS</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMP</td>
<td>Virtual Machine</td>
<td>Hosts the DTMP Solution running in several Docker Containers for our microservices</td>
<td>Windows 11</td>
<td>Standard E2bs v5 Server with 2 vCPUs, 16 GiB memory, 127 GiB disk space</td>
</tr>
<tr>
<td>JMeter-Server</td>
<td>Virtual Machine</td>
<td>Hosts the JMeter application</td>
<td>Windows 11</td>
<td>Standard E2bs v5 Server with 2 vCPUs, 16 GiB memory, 127 GiB disk space</td>
</tr>
<tr>
<td>DTMP-RunEnv</td>
<td>Virtual Machine</td>
<td>Serves as the Platform's Runtime Engine Server responsible for running DT instances</td>
<td>Linux</td>
<td>Standard E4-2ds v4 Server with 2 vCPUs, 32 GiB memory, 30 GiB disk space</td>
</tr>
<tr>
<td>DTMW-Server</td>
<td>Azure DB for PostgreSQL</td>
<td>Hosts the application database</td>
<td>N/A</td>
<td>General Purpose, D4ads_v5, 4 vCores, 16 GiB RAM, 128 GiB storage</td>
</tr>
</tbody>
</table>
Three experiments have been set up to evaluate the performance of our solution, and the relationship between the experiments and the research objectives from Section 2.2 are shown in the table below.

<table>
<thead>
<tr>
<th>Research Objective</th>
<th>Experiments</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 - Design a system that supports DT instance creation on a container runtime environment</td>
<td>6.2.1 - Experiments to measure the Performance of DT instance creation across 3 load-balanced scenarios</td>
<td>Load-Balanced Scenario 1: 1 instance of the Container Runtime Microservice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Load-Balanced Scenario 2: 2 instances of the Container Runtime Microservice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Load-Balanced Scenario 3: 3 instances of the Container Runtime Microservice</td>
</tr>
<tr>
<td>2.2.2 - Evaluate the Performance of our proposed platform</td>
<td>6.2.2 - Experiments to measure the Performance of accessing a DT instance Directly vs via our Platform Proxy</td>
<td>Compare Performance on HTTP GET Requests</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare Performance on HTTP POST Requests</td>
</tr>
<tr>
<td>2.2.3 - Evaluate the Performance of an Aggregate DT instance</td>
<td>6.2.3 - Experiments to measure the Performance of accessing an Aggregate DT using 3 different data aggregation strategies</td>
<td>Strategy 1: No Aggregation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Strategy 2: Aggregate data using LLM API</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Strategy 3: Aggregate Data using our platform's Data Aggregation Microservice</td>
</tr>
</tbody>
</table>

### 6.2 Experiments and Results

The following sections present the results of our evaluations of our platform based on the above-stated sets of experiments, within the constraints of our infrastructure.

#### 6.2.1 Experiments to Measure Performance Across 3 Load-Balanced Scenarios

Creating a Digital Twin Instance, such as for a heat pump or solar panels, involves the instantiation of an OCI container. Activation of each scenario occurs during the subscription phase, wherein the DT Platform's endpoint is invoked to instantiate a DT container based on a specific model. It's important to note that while some subscription requests might include particular personalization parameters for the DT instance within their payload—utilizing default values in their absence—other instances of DT creation may proceed without necessitating any personalization.

This set of experiments is designed to assess the average latency, throughput, and error rates across three distinct load-balanced scenarios for the creation of DT container instances within our infrastructure. Using the performance metrics of the first scenario as the baseline, our objective is to show that performance can be enhanced by simply scaling the number of Container Runtime Engine microservice instances, particularly during high demand periods.
Table 6.5. Experiment 1 Performance Summary Report

<table>
<thead>
<tr>
<th># Samples</th>
<th>Average Latency (ms)</th>
<th>Average Throughput (requests/sec)</th>
<th>Average Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scenario 1</td>
<td>Scenario 2</td>
<td>Scenario 3</td>
</tr>
<tr>
<td>25</td>
<td>2784</td>
<td>2086</td>
<td>2292</td>
</tr>
<tr>
<td>50</td>
<td>2557</td>
<td>2093</td>
<td>2117</td>
</tr>
<tr>
<td>75</td>
<td>2770</td>
<td>2448</td>
<td>2244</td>
</tr>
<tr>
<td>100</td>
<td>2825</td>
<td>2593</td>
<td>2314</td>
</tr>
<tr>
<td>125</td>
<td>2853</td>
<td>2649</td>
<td>2388</td>
</tr>
<tr>
<td>150</td>
<td>3028</td>
<td>2738</td>
<td>2508</td>
</tr>
<tr>
<td>175</td>
<td>3547</td>
<td>2847</td>
<td>2657</td>
</tr>
<tr>
<td>200</td>
<td>3709</td>
<td>3244</td>
<td>2778</td>
</tr>
<tr>
<td>225</td>
<td>4083</td>
<td>3206</td>
<td>2990</td>
</tr>
<tr>
<td>250</td>
<td>4418</td>
<td>3318</td>
<td>3143</td>
</tr>
<tr>
<td>275</td>
<td>4084</td>
<td>3354</td>
<td>3102</td>
</tr>
</tbody>
</table>

Analysis of Average Latency:

Fig. 6.2. Bar Graph Comparison of the Average Latency (ms)
As the number of samples increases, the average latency tends to increase in all three scenarios, particularly after 200 samples. This could be due to the system being increasingly loaded or possibly due to the accumulation of processing time as more tasks are executed.

Scenario 3 – the load balancing option with 3 running instances of the Container Runtime Microservice - generally has the lowest latency of all 3 scenarios investigated. This suggests that the load balancing in Scenario 3 is the most efficient for reducing latency.

The graphs in Fig. 6.2 and 6.3 provide a visual representation of the latency trends. They confirm that Scenario 3 performs best in terms of latency, particularly at higher sample sizes.

**Analysis of Average Throughput:**
The average throughput across the scenarios starts at around 0.96 requests/sec and improves slightly as the number of samples increase with each scenario reaching a peak throughput at 275 samples. Of all three scenarios, Scenario 3 generally maintains the highest and more consistent throughput. This indicates that the system's capacity to handle requests is stable.

With each scenario reaching their peaks at 275 samples, this might not be unconnected to the increased number of errors at the mark as well, possibly indicating that failed requests are being returned at a much faster rate.
Figs. 6.4 and 6.5 show that throughput is consistent across all scenarios, with only minor differences. The consistency of throughput suggests that the system scales well in handling requests.
Analysis of Average Error Rate:

The error rates remain at 0% for lower sample sizes across all scenarios but begin to increase at around 100 samples, reaching a high of 8% at 275 samples. As the number of samples increases, the error rate begins to show variability.
Scenarios 1 and 3 maintain a lower and more consistent error rate, with Scenario 3 outperforming the others as the sample size increases to 175. However, all three scenarios seem to indicate a maximum of no more than 225 running containers on the runtime server for optimal performance.

**Summary:**
- Scenario 3 appears to be the most robust configuration, providing the best latency, throughput and error rate, especially at higher sample sizes. This confirms our premise that the overall system performance with regards to creating Digital Twin instances (as docker containers) can be improved with the addition of more instances of our Container Runtime Microservice.
- Scenario 2 may have issues that need investigation, particularly because of the higher error rates.
- Scenario 1 exhibits the worst performance in terms of latency and shows significant reliability issues at higher loads, as demonstrated by the high error rates.
- The system maintains good throughput across all scenarios.

**Conclusion:**
From this analysis, Scenario 3 would generally be the recommended setup, especially for larger scales, given its balance between latency, throughput, and error rate. Scenario 1 seems to be the least desirable given its consistently higher latency and poor error rate performance at higher sample sizes.

**6.2.2 Experiments to Measure Performance of Direct versus Proxy DT Endpoints Access**

Accessing the energy consumption data or other parameters of the heat pump Digital Twin (DT) involves calling its corresponding HTTP GET endpoints. Similarly, modifying the heat pump's settings, such as compressor speed or temperature, requires invoking services through HTTP POST/PUT endpoints. These operations are central to the experiments conducted on the platform.

The experiments are fundamental to the platform's objective of enabling broad access to DT models. By creating distinct workload simulations for individual users interacting with their own DT instances, the goal is to thoroughly assess the platform's resilience and its compliance with established access policies, especially under heavy load conditions.

The experiments involve a comparative analysis of average latency, throughput, and error rates across three scenarios for accessing a DT container instance endpoint: direct access versus access via our proxy platform, with direct access serving as the performance benchmark. The objective is to demonstrate that any added performance overhead from the proxy scenario is outweighed by the benefits of enhanced
security, data validation, and policy enforcement provided by the proxy through its microservices in the request-response cycle.

Table 6.6. Experiment 2.1 Performance Summary Report for HTTP GET Requests

<table>
<thead>
<tr>
<th># Samples</th>
<th>Average Latency (ms)</th>
<th>Average Throughput (requests/sec)</th>
<th>Average Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct</td>
<td>Proxy (No Caching)</td>
<td>Proxy (Cached)</td>
</tr>
<tr>
<td>100</td>
<td>146</td>
<td>123</td>
<td>80</td>
</tr>
<tr>
<td>200</td>
<td>146</td>
<td>135</td>
<td>83</td>
</tr>
<tr>
<td>300</td>
<td>147</td>
<td>146</td>
<td>82</td>
</tr>
<tr>
<td>400</td>
<td>150</td>
<td>149</td>
<td>83</td>
</tr>
<tr>
<td>500</td>
<td>145</td>
<td>166</td>
<td>86</td>
</tr>
<tr>
<td>600</td>
<td>146</td>
<td>136</td>
<td>83</td>
</tr>
<tr>
<td>700</td>
<td>148</td>
<td>133</td>
<td>84</td>
</tr>
<tr>
<td>800</td>
<td>149</td>
<td>193</td>
<td>85</td>
</tr>
<tr>
<td>900</td>
<td>151</td>
<td>136</td>
<td>86</td>
</tr>
<tr>
<td>1000</td>
<td>147</td>
<td>158</td>
<td>85</td>
</tr>
</tbody>
</table>

Analysis of Average Latency:

Fig. 6.8. Bar Graph Comparison of the Average Latency (ms) for HTTP GET Requests
Proxy with caching consistently provides the lowest latency, significantly improving performance compared to direct access and proxy without caching. Direct access latency is relatively stable as the number of samples increases, whereas proxy without caching shows more variability, particularly at 800 samples where latency spikes significantly.

**Analysis of Average Throughput:**

![HTTP GET Requests Average Throughput](image)

![Fig. 6.10. Bar Graph Comparison of the Average Throughput (req/sec) for HTTP GET Requests](image)
Throughput remains remarkably consistent across all samples and scenarios at around 1.00 requests/sec, except for proxy without caching that has minor fluctuations in throughput with a notable dip at the 400-sample point. Proxy with caching generally matches or slightly exceeds the throughput of direct access indicating effective request handling even with the overhead of caching.

**Analysis of Average Error Rate:**

![HTTP GET Requests Average Error Rate](image-url)
The error rate is 0% across all scenarios and sample sizes, except for mid-level sample sizes for the proxy scenarios – a situation that may warrant further investigations. Nonetheless, these errors occur less than 0.5% of the time, indicating a system with high reliability.

**Summary:**
- **Proxy with Caching:** This method shows the best performance, with significantly lower latencies. The presence of caching seems to efficiently reduce the time taken to fulfil requests, likely due to the reuse of previously retrieved data.
- **Proxy without Caching:** This is the least consistent, with variable latency that occasionally spikes, suggesting potential performance issues under certain conditions.

**Conclusion:**
Based on this analysis, using a proxy with caching appears to be the most efficient way to handle HTTP GET requests, as it combines low latency with high throughput and reliability. The consistency of throughput across all scenarios suggests the system's capacity is not a bottleneck, and the less than 0.5% error rate reflects high reliability.

**Table 6.7. Experiment 2.2 Performance Summary Report for HTTP POST Requests**

<table>
<thead>
<tr>
<th># Samples</th>
<th>Latency</th>
<th>Throughput</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct</td>
<td>Proxy</td>
<td>Direct</td>
</tr>
<tr>
<td>100</td>
<td>136</td>
<td>251</td>
<td>1.0109</td>
</tr>
<tr>
<td>200</td>
<td>137</td>
<td>247</td>
<td>1.0055</td>
</tr>
<tr>
<td>300</td>
<td>136</td>
<td>242</td>
<td>1.0037</td>
</tr>
<tr>
<td>400</td>
<td>136</td>
<td>244</td>
<td>1.0028</td>
</tr>
<tr>
<td>500</td>
<td>136</td>
<td>247</td>
<td>1.0024</td>
</tr>
<tr>
<td>600</td>
<td>136</td>
<td>244</td>
<td>1.0021</td>
</tr>
<tr>
<td>700</td>
<td>136</td>
<td>246</td>
<td>1.0018</td>
</tr>
<tr>
<td>800</td>
<td>138</td>
<td>244</td>
<td>1.0017</td>
</tr>
<tr>
<td>900</td>
<td>148</td>
<td>301</td>
<td>1.0014</td>
</tr>
<tr>
<td>1000</td>
<td>145</td>
<td>318</td>
<td>1.0015</td>
</tr>
</tbody>
</table>
Analysis of Average Latency:

Fig. 6.14. Bar Graph Comparison of the Average Latency (ms) for HTTP POST Requests

Fig. 6.15. Line Graph Comparison of the Average Latency (ms) for HTTP POST Requests

The average latency remains fairly constant for both scenarios, with direct access starting at 136 ms for 100 samples and slightly increasing to 145 microseconds for 1000 samples while proxy access average latency values are almost double that for direct access.
Analysis of Average Throughput:

Throughput for direct access slightly decreases from 1.0109 requests/sec for 100 samples to 1.0015 requests/sec for 1000 samples, indicating a small decline as the number of samples increases. Proxy access also shows a similar trend but starts with a slightly lower throughput at 1.0097 requests/sec for 100 samples, decreasing to 1.0012 requests/sec for 1000 samples.
Analysis of Average Error Rate:

**Fig. 6.18.** Bar Graph Comparison of the Average Error Rate (%) for HTTP POST Requests

**Fig. 6.19.** Line Graph Comparison of the Average Error Rate (%) for HTTP POST Requests

The error rate is zero for most sample sizes for both direct and proxy accesses. However, there is a notable 0.28% error rate for proxy access with 500 samples.

**Summary:**
- While proxy access shows a slight increase in latency and decrease in throughput (and an almost imperceptible error rate), these performance compromises are associated with the following added platform benefits of security, data validation and access policy enforcement.

**Conclusion:**
Given these added benefits, the slightly increased latency and decreased throughput associated with the proxy service are often considered acceptable given the critical importance of security and compliance.
Overall, the data shows that within acceptable thresholds for performance metrics like latency and throughput in the context of their security requirements and regulatory obligations, the security and control provided by our proxy platform are worth the trade-off in raw performance metrics.

6.2.3 Experiments to Measure Performance Across 3 Data Aggregation Scenarios

The Smart Home composite Digital Twin (DT), designed to assess the home’s energy efficiency by querying the individual atomic DTs—namely, the heat pump, solar panel, and furnace, combines their outputs into a single, aggregated metric that reflects the interplay among these atomic DTs as specified by the DT user.

The objective of these experiments is to evaluate the average latency, throughput, and error rates in three distinct scenarios of data aggregation when invoking service endpoints on Aggregated DT instances. The baseline scenario, involving no aggregation, sets the performance benchmark. The experiments aim to show that the third scenario, which utilizes a Data Aggregator Microservice, achieves an acceptable level of performance overhead compared to the baseline non-aggregation scenario. Moreover, it aims to demonstrate superior performance relative to the second scenario, which employs a more complex aggregation mechanism (LLM), without sacrificing any functionality.

Table 6.8. Experiment 3 Performance Summary Aggregate Twin

<table>
<thead>
<tr>
<th># Samples</th>
<th>Unaggregated</th>
<th>LLM-Aggregated</th>
<th>DTMP MS</th>
<th>Average Latency (ms)</th>
<th>Unaggregated</th>
<th>LLM-Aggregated</th>
<th>DTMP MS</th>
<th>Average Throughput (requests/sec)</th>
<th>Unaggregated</th>
<th>LLM-Aggregated</th>
<th>DTMP MS</th>
<th>Average Error Rate (%)</th>
<th>Unaggregated</th>
<th>LLM-Aggregated</th>
<th>DTMP MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>89</td>
<td>4981</td>
<td>117</td>
<td>1.01</td>
<td>0.96</td>
<td>1.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>88</td>
<td>4897</td>
<td>112</td>
<td>1.01</td>
<td>0.91</td>
<td>1.01</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>86</td>
<td>5126</td>
<td>119</td>
<td>1.00</td>
<td>0.78</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>88</td>
<td>4782</td>
<td>121</td>
<td>1.00</td>
<td>0.98</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>88</td>
<td>5084</td>
<td>121</td>
<td>1.00</td>
<td>0.84</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>88</td>
<td>5588</td>
<td>115</td>
<td>1.00</td>
<td>0.75</td>
<td>1.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>88</td>
<td>7999</td>
<td>118</td>
<td>1.00</td>
<td>0.73</td>
<td>1.00</td>
<td>0.00</td>
<td>3.67</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>88</td>
<td>8234</td>
<td>114</td>
<td>1.00</td>
<td>0.74</td>
<td>1.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>88</td>
<td>3753</td>
<td>118</td>
<td>1.00</td>
<td>0.62</td>
<td>1.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>88</td>
<td>3892</td>
<td>114</td>
<td>1.00</td>
<td>0.90</td>
<td>1.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analysis of Average Latency:

The Unaggregated strategy maintains a consistently low latency across all sample sizes (around 88 ms).
The LLM-Aggregated strategy exhibits extremely high latencies, suggesting a substantial processing overhead incurred when invoking the LLM API to aggregate the results.
The DTMP Microservice strategy shows moderate latency, higher than the Unaggregated but significantly lower than the LLM-Aggregated strategy.
The latency graphs in Figs. 6.20 and 6.21 emphasize the stark difference between the Unaggregated/DTMP Microservices strategy’s consistently low latency and the LLM-Aggregated strategy’s very high latency.
Average Throughput:

The Unaggregated and DTMP Microservice strategies maintain a high and steady throughput of 1.01 requests/sec across all sample sizes. The LLM-Aggregated strategy's throughput decreases as the number
of samples increases, which may indicate scalability issues with using that aggregation strategy for high volume requests.

The throughput for the Unaggregated and DTMP Microservice strategies, as visually presented in Figures 6.22 and 6.23, remains consistent across sample sizes, whereas the LLM-Aggregated strategy’s throughput shows more variability.

**Analysis of Average Error Rate:**

The Unaggregated and DTMP Microservice strategies show an error rate of 0% across all sample sizes, which is ideal and an indication of high reliability. The LLM-Aggregated strategy has a spike in error rate at 700 samples (3.67%) (see Figures 6.24 and 6.25), suggesting that this strategy may become unreliable under heavier loads.

![Bar Graph Comparison of the Aggregate Twin Average Error Rate (%)](image1)

**Fig. 6.24.** Bar Graph Comparison of the Aggregate Twin Average Error Rate (%)

![Line Graph Comparison of the Aggregate Twin Average Error Rate (%)](image2)

**Fig. 6.25.** Line Graph Comparison of the Aggregate Twin Average Error Rate (%)
Summary:
- The Unaggregated strategy is highly efficient, with the best latency and error rate but lacks the benefits of data aggregation.
- The LLM-Aggregated strategy is inefficient in terms of latency and shows potential reliability issues at higher sample sizes.
- The DTMP Microservice is a compromise between the two, with reasonable latency, good throughput, and no error rate, which might be the best overall choice if aggregation is necessary and the slight increase in latency is acceptable.

Conclusion:
The results of our experiment show that that the DTMP Microservice option offers a balance for achieving reasonable performance outcomes while retaining the benefits of delivering a consolidated output.

6.3 Application Performance Summary - Apdex Score and Performance Ratings

Table 6.9. Summary Performance: Application Performance Indices (Apdex) for the DTMP

Given a Tolerated threshold for the four different chosen configurations of our DTMP platform, the tables above present the application performance using the APDEX Score.
1. **Experiment 1 - Scenario 3 (DT Creation with 3 Container Runtime Microservice Instances):**
   The Apdex scores start perfect (0.993 or higher), suggesting excellent performance for up to 150 samples, aligning with the Apdex target of 3 (assumed to represent a high-performance benchmark). However, as the number of samples increases beyond 175, we see a gradual decline in the Apdex scores, slipping to "GOOD" at 175 and 200 samples and further down to "FAIR" at 225 samples and beyond. This decline indicates that as the system is stressed with more samples, user satisfaction diminishes, possibly due to increased response times or lowered throughput.

2. **Experiment 2 - GET Scenario 3 (Proxy Cached):** This scenario consistently shows an "EXCELLENT" Apdex score across all sample sizes. It suggests that GET requests are handled efficiently by the application when caching is enabled, ensuring user satisfaction remains high.

3. **Experiment 2 - POST Scenario 2 (Proxy):** The scenario starts with perfect Apdex scores for sample sizes up to 500. However, at 600 samples, the Apdex score marginally drops to 0.996, still maintaining an "EXCELLENT" rating but indicating the beginning of a performance impact. Beyond this point, the Apdex scores remain stable, suggesting the system copes well with increased loads during POST requests.

4. **Experiment 3 - Scenario 3 (Aggregation Strategy using Data Aggregator Microservice):** In this scenario, the system maintains an "EXCELLENT" Apdex score across all sample sizes, reflecting a consistently high level of user satisfaction. This implies that the application is responsive, stable, and likely to be well-received by its users, regardless of the number of requests processed.

**Summary:**
The Apdex ratings shed light on the application's user experience, with higher scores denoting superior performance. The data reveal that the application generally excels, especially in handling GET requests and within the context of Experiment 3 - Scenario 3. Nonetheless, a noticeable performance decline in Experiment 1 - Scenario 3 as the workload intensifies points to possible limitations in our test infrastructure's resources.

Reflecting on the outcomes of the conducted experiments—particularly those involving the use of three or more Container Runtime Microservices for Digital Twin (DT) creation, alongside the implementation of caching and a Data Aggregator Microservice for processing GET requests for both atomic and composite DT instances—we have successfully shown that our solution achieves the research goals outlined in section 2.2 of this document. This advancement is a significant step in addressing the complexities of DT hosting,
packaging, distribution, and access, all while upholding data privacy and safeguarding the intellectual property of model designers.
7.1 Conclusion

This study sets forth a pioneering framework—an open-source Digital Twin Management Platform (DTMP)—to navigate the complexities associated with creating, sharing, and managing Digital Twin Models (DTMs) while simultaneously upholding the governance of intellectual property rights for model creators and data rights for model subscribers. An innovative feature of this platform is the facility for Digital Twin Aggregation, which enables the logical consolidation of multiple DT instances to provide an integrated perspective of a system comprising interacting DTs.

Our system champions the use of Open Container Initiative (OCI) images, compatible with any OCI-conformant runtime engine, as a standardized method for the packaging and distribution of DTMs. This approach simplifies the publication process for creators, from the initial offering of a model on our platform to the eventual decommissioning of a twin instance, thereby encapsulating the entire lifecycle management of DT instances.

In the preliminary chapters, our study delineates the challenges prompting our research and defines our investigative trajectory. Chapter 3 delves into the concept of Digital Twins, elucidating their escalating significance and versatile applications. We underscore the role of DTs in fostering inclusive policy-making and enabling citizens to assert their stake in pressing global issues, such as climate change.

Chapter 4 details the architecture and design of our proposed system, dissecting the various components of our middleware. Chapter 5 then provides an exhaustive exposition of the implemented solution, articulating its array of features and capabilities.

Chapter 6 encompasses a series of experiments to assess system performance against several benchmarks. The promising outcomes of these evaluations underscore the platform's capacity to support a substantial user base.

The outcomes of the experiments confirm that the solution successfully fulfills the objectives outlined in section 2.2 of this document, achieving an "Excellent" rating in perceived user experience as determined by the Apdex scoring system. Furthermore, when comparing average latency and throughput across various test scenarios, the results consistently align well with the established benchmarks.
7.2 Contribution

The creation of a digital twin is an endeavour rich in knowledge and necessitates protection for the intellectual labour invested. A salient contribution of this study is the insulation it offers to subscribers of DT models: they can fully utilize all features and functionalities, yet are barred from any direct interaction with the underlying infrastructure. This ensures that the internal mechanisms of the digital twin models remain confidential and secure.

Further, the development of a middleware service that can amalgamate multiple DT instances into a cohesive model represents a significant advancement. This integration enables the synthesis of individual outputs into a singular, coherent result.

Notably, this study has harnessed the capabilities of Artificial Intelligence (AI) and Large Language Models (LLMs) to augment the value derived from DT models. By leveraging a "conversational" interface with publicly available AI models, our system facilitates a three-way exchange of information, yielding novel insights. The employment of AI for converting natural language into JSON-formatted instruction sets is another meritorious aspect of our platform.

7.3 Future Work

The overarching aim is to render access to DTs universal, laying down a foundation that allows citizens to actively engage in public discourse and address community concerns. Future efforts will be channelled towards magnifying user access and extracting greater utility from the rich insights encapsulated in diverse DT models, which are becoming increasingly integral to daily life. The successful deployment of our solution signifies a meaningful stride in this pursuit.

Prospective avenues for further research might encompass:

- Enhancing the functionality of composite Digital Twins (DTs) within the platform, aiming to integrate the services of all constituent atomic DTs. This advancement will position composite DTs as a unified control hub for all aggregated Digital Twin Instances (DTIs), streamlining management and interaction across the system. Effectively, the Composite DT functionality would be enhanced to leverage all existing services of constituent DT Instances.

- Advancing our platform to incorporate the concept of Digital Twin Aggregation "Templates" offering predefined configurations for combining DT instances to achieve specific outcomes. Leveraging the experiences of model developers, pre-defined combinations of DTIs would be supported by the platform to enhance value and utility of a user’s investment in DT acquisitions.
• Expanding the platform to conceptualize a "Community Twin"—a communal DT representation of public assets managed by corporations or other such stewards but made transparent to the public, enhancing visibility and transparency. This extension would represent a DT that incorporates the idea of “multiple stakeholders” and serve as a “living” contract between communities and service providers e.g. a DT representation of power utility equipment visible to members of the public served by that utility.

• Progressing the dialogue towards establishing a universally endorsed Digital Twin Description Language, facilitating standardization and ease of DT model interpretation and integration. This essentially would imply participation in ongoing discussions on adopting a common and universally accepted description language for the Digital Twin.

This study represents a substantial leap in the practical application of DT technology, with the ambition of democratizing its use and unleashing its full potential for societal benefit.
REFERENCES


“I (Legislative acts) REGULATIONS REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)”.


Appendix A

This section shows excerpts of the docker compose file used to deploy the solution.

```
version: "3.8"
services:
  app-user-mgmt:
    image: ugbomah/wasm-user-mgmt:0.1
    container_name: app-user-mgmt
    platform: wasi/wasm
    runtime: io.containerd.wasmedge.v1
    environment:
      - DATABASE_URL=postgresql://postgres:your-db-password@dtmw-server.postgres.database.azure.com:5432/dtmw
      - RUST_BACKTRACE=1
      - DNS_SERVER=127.0.0.11:53
    restart: unless-stopped

  app-model-mgmt:
    image: ugbomah/wasm-model-policy-mgmt:0.1
    container_name: app-model-mgmt
    platform: wasi/wasm
    runtime: io.containerd.wasmedge.v1
    environment:
      - DATABASE_URL=postgresql://postgres:your-db-password@dtmw-server.postgres.database.azure.com:5432/dtmw
      - RUST_BACKTRACE=1
      - DNS_SERVER=127.0.0.11:53
    restart: unless-stopped

  app-twin-mgmt:
    image: ugbomah/wasm-twin-mgmt:0.1
    container_name: app-twin-mgmt
    platform: wasi/wasm
    runtime: io.containerd.wasmedge.v1
    environment:
      - DATABASE_URL=postgresql://postgres:your-db-password@dtmw-server.postgres.database.azure.com:5432/dtmw
      - RUST_BACKTRACE=1
      - DNS_SERVER=127.0.0.11:53
    restart: unless-stopped
```
nginx-proxy:
  image: nginx:latest
  container_name: nginx-proxy
  ports:
    - "3034:3034"
  volumes:
    - ./runtime-mgmt-1/nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
    - app-runtime-mgmt1
    - app-runtime-mgmt2
    - app-runtime-mgmt3
  restart: unless-stopped

app-runtime-mgmt1:
  image: ugbomah/dtmw-runtime-mgmt:0.1
  container_name: app-runtime-mgmt1
  environment:
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  restart: unless-stopped

app-runtime-mgmt2:
  image: ugbomah/dtmw-runtime-mgmt:0.1
  container_name: app-runtime-mgmt2
  environment:
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  restart: unless-stopped

app-runtime-mgmt3:
  image: ugbomah/dtmw-runtime-mgmt:0.1
  container_name: app-runtime-mgmt3
  environment:
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  restart: unless-stopped
app-logging-mgmt:
  image: ugbomah/wasm-logging-mgmt:0.1
  container_name: app-logging-mgmt
  platform: wasi/wasm
  runtime: io.containerd.wasmedge.v1
  environment:
    - DATABASE_URL=postgresql://postgres:your-db-password@dtmw-server.postgres.database.azure.com:5432/dtmw
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  restart: unless-stopped

app-chatgpt-api:
  image: ugbomah/dtmw-chatgpt-api:0.1
  container_name: app-chatgpt-api
  platform: linux/amd64
  environment:
    - RUST_BACKTRACE=1
    - TOKEN_DURATION=3600
    - REDIS_URL=redis://redis:6379
    - OPENAI_API_KEY=your-open-ai-key
    - OPENAI_API_URL=https://api.openai.com/v1/chat/completions
    - JWT_SECRET=your-jwt-secret
  depends_on:
    - redis
  restart: unless-stopped

app-param-checker:
  image: ugbomah/wasm-param-checker:0.1
  container_name: app-param-checker
  platform: wasi/wasm
  runtime: io.containerd.wasmedge.v1
  environment:
    - REDIS_URL=redis://redis:6379
    - MODEL_MGMT_API_URL=http://app-model-mgmt:3032/api/v1
    - JWT_SECRET=your-jwt-secret
    - TOKEN_DURATION=3600
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  depends_on:
    - app-model-mgmt
  restart: unless-stopped

redis:
  image: redis:latest
  ports:
    - 6379:6379
  command: ["redis-server", "--appendonly", "yes"]
  volumes:
    - redis-data:/data
  restart: unless-stopped
app-data-aggregator:
  image: ugbomah/wasm-data-aggregator:0.1
  container_name: app-data-aggregator
  platform: wasi/wasm
  runtime: io.containerd.wasmedge.v1
  environment:
    - REDIS_URL=redis://redis:6379
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  depends_on:
    - redis
  restart: unless-stopped

app-proxy:
  image: ugbomah/dtmw-proxy:0.1
  container_name: app-proxy
  environment:
    - JWT_SECRET=your-jwt-secret
    - TOKEN_DURATION=3600
    - USER_MGMT_API_URL=http://app-user-mgmt:3031/api/v1
    - MODEL_MGMT_API_URL=http://app-model-mgmt:3032/api/v1
    - TWIN_MGMT_API_URL=http://app-twin-mgmt:3033/api/v1
    - RUNTIME_MGMT_API_URL=http://nginx-proxy:3034/api/v1
    - LOGGING_MGMT_API_URL=http://app-logging-mgmt:3035/api/v1
    - CHATGPT_API_URL=http://app-chatgpt-api:3036/api/v1
    - PARAM_MGMT_API_URL=http://app-param-checker:3037/api/v1
    - AGGREGATOR_API_URL=http://app-data-aggregator:3038/api/v1
    - FRONT_END_API=http://dtmp.canadaeast.cloudapp.azure.com:8080
    - TWIN_RUNTIME_ENV=http://dtmp-renv.canadaeast.cloudapp.azure.com
    - REDIS_URL=redis://redis:6379
    - RUST_BACKTRACE=1
    - DNS_SERVER=127.0.0.11:53
  ports:
    - 3030:3030
  depends_on:
    - app-user-mgmt
  restart: unless-stopped

app-dtmp-fe:
  image: ugbomah/dtmp-fe:0.1
  container_name: app-dtmp-fe
  environment:
    - VITE_PROXY_APP_API_BASE_URL=http://app-proxy:3030
    - RUST_BACKTRACE=1
  ports:
    - 8080:5173
  depends_on:
    - app-proxy
  restart: unless-stopped

volumes:
  redis-data:
Appendix B

This section shows excerpts of the docker compose file used to deploy the solution.

```sql
CREATE TABLE IF NOT EXISTS core_role
(
    id uuid NOT NULL,
    name character varying UNIQUE COLLATE pg_catalog."default" NOT NULL,
    description character varying COLLATE pg_catalog."default",
    system_role boolean NOT NULL DEFAULT false,
    status boolean NOT NULL DEFAULT true,
    created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by uuid,
    updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_by uuid,
    deleted_at timestamp(6) with time zone,
    deleted_by uuid,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS core_user
(
    id uuid NOT NULL,
    first_name character varying COLLATE pg_catalog."default" NOT NULL,
    last_name character varying COLLATE pg_catalog."default" NOT NULL,
    email character varying UNIQUE COLLATE pg_catalog."default" NOT NULL,
    password character varying COLLATE pg_catalog."default" NOT NULL,
    role_id uuid NOT NULL,
    status boolean NOT NULL DEFAULT true,
    created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by uuid,
    updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_by uuid,
    deleted_at timestamp(6) with time zone,
    deleted_by uuid,
    FOREIGN KEY (role_id)
        REFERENCES core_role (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS core_model_component_architecture
(
    id SERIAL PRIMARY KEY,
    name VARCHAR(64) NOT NULL UNIQUE
);
```
CREATE TABLE IF NOT EXISTS core_model
(
    id uuid NOT NULL,
    name character varying UNIQUE COLLATE pg_catalog."default" NOT NULL,
    description character varying COLLATE pg_catalog."default" NOT NULL,
    picture character varying COLLATE pg_catalog."default",
    is_published boolean NOT NULL DEFAULT false,
    enable_data_sharing boolean NOT NULL DEFAULT false,
    cache_duration integer NOT NULL DEFAULT 0,
    created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by uuid,
    updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_by uuid,
    deleted_at timestamp(6) with time zone,
    deleted_by uuid,
    CONSTRAINT core_model_pkey PRIMARY KEY (id)
);
CREATE TABLE IF NOT EXISTS core_policy (  id uuid NOT NULL,  name character varying COLLATE pg_catalog."default" NOT NULL,  description character varying COLLATE pg_catalog."default" NOT NULL,  model_id uuid NOT NULL,  policy_version integer NOT NULL DEFAULT 1,  block_after integer NOT NULL DEFAULT 0,  max_allowed_subscriptions integer NOT NULL DEFAULT 0,  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,  created_by uuid,  updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,  updated_by uuid,  deleted_at timestamp(6) with time zone,  deleted_by uuid,  CONSTRAINT core_policy_pkey PRIMARY KEY (id),  CONSTRAINT "core_twin_modelId_fkey" FOREIGN KEY (model_id) REFERENCES core_model (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT "core_policy_userId_fkey" FOREIGN KEY (created_by) REFERENCES core_user (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION
);

CREATE TABLE IF NOT EXISTS core_model_service (  id uuid NOT NULL,  model_id uuid NOT NULL,  service_name character varying COLLATE pg_catalog."default" NOT NULL,  description character varying COLLATE pg_catalog."default" NOT NULL,  service_endpoint character varying COLLATE pg_catalog."default" NOT NULL,  endpoint_verb character varying COLLATE pg_catalog."default" NOT NULL,  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,  created_by uuid,  updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,  updated_by uuid,  deleted_at timestamp(6) with time zone,  deleted_by uuid,  CONSTRAINT core_model_service_pkey PRIMARY KEY (id),  CONSTRAINT "core_model_service_fkey" FOREIGN KEY (model_id) REFERENCES core_model (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT "core_model_service_userId_fkey" FOREIGN KEY (created_by) REFERENCES core_user (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION
);
CREATE TABLE IF NOT EXISTS core_policy_action
(
  id uuid NOT NULL,
  policy_id uuid NOT NULL,
  endpoint character varying COLLATE pg_catalog."default" NOT NULL,
  description character varying COLLATE pg_catalog."default" NOT NULL,
  endpoint_verb character varying COLLATE pg_catalog."default" NOT NULL,
  action_count integer NOT NULL DEFAULT 0,
  reset_frequency_id integer NOT NULL DEFAULT 1,
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_by uuid,
  updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_by uuid,
  deleted_at timestamp(6) with time zone,
  deleted_by uuid,
  CONSTRAINT core_policy_action_pkey PRIMARY KEY (id),
  CONSTRAINT "core_userId_fkey" FOREIGN KEY (created_by)
    REFERENCES core_user (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_action_policyId_fkey" FOREIGN KEY (policy_id)
    REFERENCES core_policy (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_policy_resetFreqId_fkey" FOREIGN KEY (reset_frequency_id)
    REFERENCES core_action_reset_frequency (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);
CREATE TYPE data_type AS ENUM ('Integer', 'Decimal', 'String');
CREATE TYPE parameter_type AS ENUM ('Single', 'Multiple');
CREATE TABLE core_model_parameter
(
  id uuid NOT NULL,
  model_id uuid NOT NULL,
  service_id uuid NOT NULL,
  parameter_name VARCHAR(255) NOT NULL,
  description character varying COLLATE pg_catalog."default" NOT NULL,
  data_type data_type NOT NULL,
  parameter_type parameter_type NOT NULL,
  is_sensitive boolean NOT NULL DEFAULT true,
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_by uuid,
  CONSTRAINT core_model_param_pkey PRIMARY KEY (id),
  CONSTRAINT "core_model_param_fkey" FOREIGN KEY (model_id)
    REFERENCES core_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_model_service_fkey" FOREIGN KEY (service_id)
    REFERENCES core_model_service (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);
INSERT INTO core_twin_status (name) VALUES ('Stopped');
INSERT INTO core_twin_status (name) VALUES ('Started');
INSERT INTO core_twin_status (name) VALUES ('Deactivated');
INSERT INTO core_twin_status (name) VALUES ('Deleted');

CREATE TABLE IF NOT EXISTS core_twin
(
  id uuid NOT NULL,
  name character varying COLLATE pg_catalog."default" NOT NULL,
  model_id uuid NOT NULL,
  policy_id uuid,
  twin_status_id integer NOT NULL DEFAULT 1,
  network_name character varying COLLATE pg_catalog."default",
  http_port integer,
  enable_data_sharing boolean NOT NULL DEFAULT false,
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_by uuid NOT NULL,
  updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_by uuid,
  CONSTRAINT core_twin_pkey PRIMARY KEY (id),
  CONSTRAINT "core_twin_modelId_fkey" FOREIGN KEY (model_id)
    REFERENCES core_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_twin_policyId_fkey" FOREIGN KEY (policy_id)
    REFERENCES core_policy (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_twin_createdby_fkey" FOREIGN KEY (created_by)
    REFERENCES core_user (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_twin_status_fkey" FOREIGN KEY (twin_status_id)
    REFERENCES core_twin_status (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);
CREATE TABLE IF NOT EXISTS core_twin_service
(
  id uuid NOT NULL,
  twin_id uuid NOT NULL,
  service_name character varying COLLATE pg_catalog."default" NOT NULL,
  description character varying COLLATE pg_catalog."default" NOT NULL,
  service_endpoint character varying COLLATE pg_catalog."default" NOT NULL,
  endpoint_verb character varying COLLATE pg_catalog."default" NOT NULL,
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_by uuid,
  updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_by uuid,
  deleted_at timestamp(6) with time zone,
  deleted_by uuid,
  CONSTRAINT core_twin_service_pkey PRIMARY KEY (id),
  CONSTRAINT "core_twin_service_fkey" FOREIGN KEY (twin_id)
    REFERENCES core_twin (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);
CREATE TABLE IF NOT EXISTS core_twin_parameter (
    id uuid NOT NULL,
    twin_id uuid NOT NULL,
    service_id uuid NOT NULL,
    parameter_name VARCHAR(64) NOT NULL,
    parameter_data_type VARCHAR(64) NOT NULL,
    is_sensitive boolean NOT NULL DEFAULT true,
    is_shared boolean NOT NULL DEFAULT false,
    created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by uuid NOT NULL,
    updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_by uuid,
    CONSTRAINT core_twin_parameter_pkey PRIMARY KEY (id),
    CONSTRAINT "core_twin_param_fkey" FOREIGN KEY (twin_id)
        REFERENCES core_twin (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT "core_twin_service_fkey" FOREIGN KEY (service_id)
        REFERENCES core_twin_service (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);
CREATE TABLE IF NOT EXISTS core_model_env_variable (
    id uuid NOT NULL,
    component_id uuid NOT NULL,
    env_variable_name VARCHAR(255) NOT NULL,
    default_value VARCHAR(255) NOT NULL,
    created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by uuid,
    CONSTRAINT core_model_env_var_pkey PRIMARY KEY (id),
    CONSTRAINT "core_model_env_var_fkey" FOREIGN KEY (component_id)
        REFERENCES core_model_component (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);
CREATE TABLE IF NOT EXISTS core_shared_model_data
(
  id uuid NOT NULL,
  model_id uuid NOT NULL,
  input_data character varying COLLATE pg_catalog."default",
  output_data character varying COLLATE pg_catalog."default",
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT core_shared_model_data_pkey PRIMARY KEY (id),
  CONSTRAINT "core_model_fkey" FOREIGN KEY (model_id)
    REFERENCES core_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);

CREATE TABLE IF NOT EXISTS core_log
(
  id uuid NOT NULL,
  model_id uuid,
  twin_id uuid,
  action_performed character varying COLLATE pg_catalog."default",
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_by uuid NOT NULL,
  CONSTRAINT core_log_pkey PRIMARY KEY (id),
  CONSTRAINT "core_log_model_fkey" FOREIGN KEY (model_id)
    REFERENCES core_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT "core_log_user_fkey" FOREIGN KEY (created_by)
    REFERENCES core_user (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);

CREATE TABLE IF NOT EXISTS core_twin_component
(
  id uuid NOT NULL,
  name VARCHAR(64) NOT NULL,
  component_alias VARCHAR(64),
  host_port integer,
  container_name character varying COLLATE pg_catalog."default",
  container_port integer,
  is_exposed boolean NOT NULL DEFAULT false,
  twin_id uuid NOT NULL,
  architecture_id integer NOT NULL,
  image_source character varying COLLATE pg_catalog."default" NOT NULL,
  created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_by uuid NOT NULL,
  updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_by uuid,
  CONSTRAINT core_twin_component_pkey PRIMARY KEY (id),
  CONSTRAINT "core_twin_comp_fkey" FOREIGN KEY (twin_id)
    REFERENCES core_twin (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
CREATE TABLE IF NOT EXISTS core_twin_aggregate_member (  
id uuid NOT NULL,  
aggregate_id uuid NOT NULL,  
twin_id uuid NOT NULL,  
twin_endpoint character varying COLLATE pg_catalog."default" NOT NULL,  
created_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,  
created_by uuid NOT NULL,  
updated_at timestamp(6) with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,  
updated_by uuid,  
CONSTRAINT core_twin_aggregate_member_pkey PRIMARY KEY (id),  
CONSTRAINT "core_twin_aggregate_member_twin_fkey" FOREIGN KEY (twin_id)  
REFERENCES core_twin (id) MATCH SIMPLE  
ON UPDATE NO ACTION  
ON DELETE NO ACTION,  
CONSTRAINT "core_twin_aggregate_member_aggregate_fkey" FOREIGN KEY (aggregate_id)  
REFERENCES core_twin_aggregate (id) MATCH SIMPLE  
ON UPDATE NO ACTION  
ON DELETE NO ACTION,  
CONSTRAINT "core_twin_aggregate_member_createdby_fkey" FOREIGN KEY (created_by)  
REFERENCES core_user (id) MATCH SIMPLE  
ON UPDATE NO ACTION  
ON DELETE NO ACTION);
Appendix C

This section shows the cost estimates of deploying the solution on the Microsoft Azure Platform, as well as cost associated with leveraging Open AI’s ChatGPT API for our integrated AI enhancements.

![Microsoft Azure PostgreSQL Database](image)

Fig. C.1. PostgreSQL Database Flexible Server Pricing

The table below presents the summary costs of the infrastructure setup.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Purpose</th>
<th>Number of Instances</th>
<th>Specification</th>
<th>Cost per Instance</th>
<th>Cost/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Machine</td>
<td>Hosts the DTMP Solution running in several Docker Containers for our microservices and the Docker Runtime Environment for Twin Instances</td>
<td>1 Application Server and 5 Runtime Environment Servers (to accommodate 1000 DT instances)</td>
<td>Standard E2bs v5 Server with 2 vCPUs, 16 GiB memory, 127 GiB disk space</td>
<td>US$120.45/month</td>
<td>US$0.17/hour</td>
</tr>
<tr>
<td>Azure DB for PostgreSQL</td>
<td>Hosts the application database</td>
<td>N/A</td>
<td>General Purpose, D4ads_v5, 4 vCores, 16 GiB RAM, 128 GiB storage</td>
<td>US$274.60/month</td>
<td>N/A</td>
</tr>
</tbody>
</table>
**GPT-3.5 Turbo**

GPT-3.5 Turbo models are capable and cost-effective. gpt-3.5-turbo-012S is the flagship model of this family, supports a 16K context window and is optimized for dialog. gpt-3.5-turbo-instruct is an Instruct model and only supports a 4K context window. Learn about GPT-3.5 Turbo [^1]

<table>
<thead>
<tr>
<th>Model</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpt-3.5-turbo-012S</td>
<td>$0.50 / 1M tokens</td>
<td>$1.50 / 1M tokens</td>
</tr>
<tr>
<td>gpt-3.5-turbo-instruct</td>
<td>$1.50 / 1M tokens</td>
<td>$2.00 / 1M tokens</td>
</tr>
</tbody>
</table>

**Fig. C.2.** OpenAI GPT-3.5 Turbo Model Pricing [23]