Testing data logging tools in DataOps for Digital Twins

Testning av loggningsverktyg i DataOps för Digitala Tvillingar

Adrian Bakken Sundmoen
<Adrian.basu@hotmail.com>

Faculty of Health, Science and Technology

Master thesis in Computer Science

Second Cycle, 30 hp (ECTS)

Supervisor: Dr. Bestoun S. Ahmed, Karlstad University, bestoun@kau.se

Examiner: Prof. Dr. Andreas J. Kassler, Karlstad University, andreas.kassler@kau.se

Karlstad, February 27th, 2023

Abstract

Industry 4.0 and a global trend in digital transformation have brought new ideas and emerging technologies to the surface. Data has become a key asset for businesses, and streamlining data and automating data life cycles have become increasingly important. This industrial revolution is centered around cyber-physical systems, and it sets forth that new technologies will change how a business traditionally operates. However, the problem is a lack of tools, systems, and methods to realize this revolution. Thus, there is a strong demand for finding solutions that move businesses toward Industry 4.0. A new technology known as Digital Twin (DT) has emerged from this. This technology aims to improve the business value of big data by digitally representing physical entities. To operate successfully with this technology, other enabling technologies and tools are needed, providing DTs with high-quality data that accurately represent the system in which the twin models are used. This can be a problem as the data might originate from different sources and often do not follow the same format and standards. Furthermore, data must also be readily collected in a timely manner. To deal with problems such as these, a new term known as Data Operations (DataOps) has surfaced. DataOps is a set of practices and processes that aims to improve the communication, integration, and automation of data flow within data landscapes and organizations. This thesis introduces a methodology to investigate whether a standardized data logging tool can be used as a DataOps solution to collect, process, and make data available for DTs. This is done by investigating the current literature and applying testing methodologies to the tool. More specifically, a combination of load, performance, and stress tests are performed to assess the ability of the tool to collect large amounts of data. The focus is on investigating whether this can be done in a timely manner. It is concluded that the tool does posses features that are of importance for DataOps and DTs, and that it could be a viable option for data gathering to certain DTs on its own. However; as a result of internal mechanics of the tool, it is not timely enough for use as a DataOps solution in general. Further research regarding improvements of its timeliness, other similar tools, and testing in a real environment consisting of a real DT is proposed and motivated.

Keywords

Master's Thesis, DataOps, Digital Twin, Logging Tools, Testing Methodology

Sammanfattning

Industri 4.0 och en global trend inom digital transformation har resulterat i att nya idéer och teknologier dykt upp. Data har blivit en allt mer viktig tillgång för företag och samhälle, och en effektivisering samt automation av datas livscykler är av ökat intresse. Denna industriella revolution är centrerad kring cyber-fysiska system och innebär att möjliggörande teknologier i grunden kommer att förändra hur företag traditionellt fungerar. Trots detta så saknas tillförlitliga verktyg, system och metoder för att fullt ut förverkliga denna revolution. Till följd av detta har nya termer och teknologier dykt upp. Två av dessa är Data Operations (DataOps) och den digitala tvillingen (DT). För att tekniker som den digitala tvillingen ska fungera optimalt måste den ha tillgång till högkvalitativ data som tillräckligt noggrant representerar systemet den modellerar. Detta är ofta ett problem då data kan härstamma från många olika system, som följer olika format och standarder. Det finns oftast också krav på att datat skall processeras och göras tillgänglig för tvillingen tillräckligt snabbt. Detta examensarbete undersöker om ett standardiserat loggningsverktig kan användas som en DataOps lösning för att samla in, processera, och göra data tillgänglig för digitala tvillingar. Detta görs genom att undersöka nuvarande litteratur, och genom att applicera diverse testningsmetodologier på verktyget. Testningen undersöker verktygets förmåga att samla in stora mängder data, och fokuserar på om detta kan göras tidsenligt. Undersökningen visar att verktyget har egenskaper som relaterar väl till de kvalitéer som önskas hos ett DataOps verktyg, och de krav som ställs på Digitala Tvillingar. Testningen visar att verktyget kan processera data med ett konsekvent resultat, men att verktyget har begränsningar som resulterar i att data inte kan processeras i realtid. Sammanfattningsvis visar undersökningen att verktyget kan vara ett lämpligt verktyg för vissa Digitala Tvillingar i sig självt, men inte som en DataOps lösning generellt. Ytterligare forskning gällande förbättring av verktygets tidsenlighet, andra liknande verktyg, och testning i en verklig miljö föreslås.

Nyckelord

Master Examensarbete, DataOps, Digital Tvilling, Loggningsverktyg, Testmetoder

Acknowledgements

Firstly, I would like to express my gratitude to Kazoku IT AB and all their very talented colleagues for giving me the opportunity, tools, and material to do this thesis. Special thanks to Robert Mayer, Do Hellbom, and Ingemar Elefant for their persistent help and patience throughout the process.

I also wish to express my sincere gratitude and a big thank you to my supervisor Bestoun S. Ahmed for his persistent and insightful advice, help, and critique throughout the completion of this thesis. Without his knowledge and experience, this work would not have been possible. Also, a big thank you to Karlstad University; not only for the opportunity to write this thesis, but also for providing me with invaluable knowledge and professional reception over the past five years of my education.

Acronyms

DT	Digital Twin
DataOps	Data Operations
SUT	System Under Test
psutil	python-system-and-process-utilities
REST	Representational state transfer
HTTP	Hypertext Transfer Protocol
WCF	Windows Communication Foundation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents introductory information about the thesis. The chapter starts with background information and a problem description of the topic, along with the objectives and goals of the thesis. The ethical aspects of the project, along with its stakeholders, will also be presented. Finally, the thesis outline and its delimitations will be discussed.

1.1 Background

Data have become a critical asset for competitiveness in today's applications, in which data-driven solutions significantly increase the success of a business. Streamlined data and analytical processes have become crucial for companies looking to improve their business performance [4] [14]. However, with an ever-growing amount of generated data, consuming all of it has become a problem. As a result, there is an increased need to shape enterprise data and analytical processes, to effectively consume this data [4]. Subsequently, new ideas and concepts have emerged. One of these is DataOps, which addresses the mentioned problems by automating data orchestration throughout an organization.

Alongside this, an industrial revolution commonly known as Industry 4.0 is happening. This revolution is about rapid change in industrial technologies centered around cyber-physical systems. By combining real-time connections between physical and digital systems with new technologies to enable innovation, Industry 4.0 is hypothesized to change the way industries will operate in the future [1]. Further strengthening competitive priorities for businesses. One of the new emerging technologies related to Industry 4.0 is the Digital Twin (DT). A DT is a virtual copy of a physical system that connects real systems to a virtual one to collect, analyze, and simulate data in a virtual model [12]. However, there is a lack of tools, methods, and systems powerful and sophisticated enough to fully realize this revolution and its technologies [17], such as the DT.

1.2 Problem Description

To operate successfully, a DT must have access to available high-quality data that accurately represent the system it models. DTs must also be able to collect data from different sources, often not following the same format and standards. A tool that solves these problems would be an attractive component for a DT. That tool could collect, standardize, and process data from different sources in one centralized platform and then make it available for retrieval for a DT in real-time. Additionally, it is essential that the data-gathering process is implemented in a way that automates and streamlines the entire data life cycle. Shortening the end-to-end cycle time of data analytics will allow business value to be achieved more effectively. The above characteristics will be managed by ensuring that a tool used for gathering data to a DT is implemented as a DataOps solution.

1.3 Thesis Objective

The aim of this thesis is to investigate whether a standardized logging tool can be used as a DataOps solution to gather and process data for a DT to use. This will be done by providing and applying a testing methodology to such a tool and evaluating its ability to gather, standardize, and process data for a generalized DT. The focus is on evaluating the timeliness of this tool. To facilitate this aim, a case study is conducted on a tool named Nodinite. Additionally, a literature review is conducted to gather information about the discussed topics.

1.4 Thesis Goals

This thesis aims to show a systematic method that evaluates a data logging tool in the context of DataOps for DTs. To create accurate and reliable DTs, it is essential to have robust and efficient data logging tools in place. Through this process, the hope is to provide valuable insight and recommendations for organizations looking to implement data logging tools in their DataOps for DTs. By identifying the most effective tools, we can help organizations optimize their data management processes and improve the accuracy and reliability of their DTs. Ultimately, the goal is to contribute to the development and advancement of DataOps for DTs and help organizations harness this technology's full potential. Furthermore, the following research questions will be used as a basis to achieve the goals of the study:

1. What are the characteristics of a DataOps solution and the data requirements of a DT?
2. Can a data logging tool be used as a DataOps solution to collect and make data available for a DT?
3. What is the timeliness of the tool under test?

1.5 Ethics and Sustainability

This thesis aims to contribute in the field of Industry 4.0, enhancing the efficiency of organizations and their data management processes. This could, under the right circumstances, result in improved sustainability. For example by lowering the energy consumption of a manufacturing process, after having applied and used technologies such as those discussed in this paper.

The data used in this thesis do not contain personal, private, or sensitive information.

1.6 Methodology

In this thesis, a data logging tool named Nodinite is evaluated as a case study for its ability to act as a DataOps solution for gathering data to DTs. A literature study is conducted to find the characteristics of a DataOps solution and the data requirements of DTs. A testing methodology is built using the gathered information, to perform the evaluation. More specifically, the thesis focuses on evaluating the tool's ability to gather and process data in a timely manner for DTs as a DataOps solution.

The testing is conducted by generating data for logging to the tool while varying the different parameters related to the data. Then it is observed how the tool behaves under these different parameters while measuring the processing times of the logged data. The tool is evaluated when analyzing and relating the testing data with the information collected from the literature study, and its timeliness.

1.7 Stakeholders

This project is carried out in collaboration with Kazoku IT AB. Kazoku is a consulting company specializing in the integration of IT systems. One of the tools used by Kazoku is Nodinite, a standardized logging and monitoring tool that centralizes information from different IT systems and integrates it into one platform. With growing interest in the industry regarding DataOps and DTs, Kazoku IT wants to investigate if and how Nodinite can be used as a DataOps solution for DTs.

1.8 Delimitations

The evaluation conducted in this thesis is based on theoretical requirements found in the literature to be important for DTs in general. This allows the tool to be evaluated for the industry in general rather than for specific industrial scenarios and environments. However, further research is still motivated in which Nodinite is applied to an actual and specific DT, where data requirements are derived from a real-life environment. Extended testing with more complex data could also contribute to

an improved evaluation. Further, as Nodinite is used as a generalization for logging tools in this study, research regarding other logging tools is also motivated.

1.9 Outline

This thesis is structured as follows: Chapter 2 presents the information necessary to understand the following chapters through background and related work. A literature study will be conducted to present and define DataOps and DTs. Furthermore, the characteristics of DataOps and the data requirements of DTs will be discussed, laying the foundations for the tests and evaluation carried out in the case study. The chapter will also present testing theories and methodologies, along with related works and a conclusion of the chapter. Following this, in chapter 3 a description of the used tool is given. In Chapter 4, a description of the design and implementation process is given. Detailed information on the methodology will be presented using written descriptions, figures, and graphs. Chapter 5 will present the study's results and the tests applied to Nodinite. Lastly, a discussion of the results, followed by a conclusion of the thesis, will be given in Chapter 6.

Chapter 2

Background and Related Work

Industry 4.0 conceptualizes rapid change in industrial technologies centered on cyber-physical systems. This revolution means that real-time connections between physical and digital systems combined with new enabling technologies are likely to change how industries traditionally operate, advancing competitive priorities such as cost, flexibility, speed, and quality [11]. However, the lack of robust tools poses a significant obstacle to reaching the full potential of Industry 4.0. More specifically, there is a lack of formal methods and systems crucial to implementing this revolution [17].

A term of relevance in the implementation of Industry 4.0 is DataOps, which aims to automate data orchestration throughout a business. Furthermore, a technology emerging directly from Industry 4.0 is the DT. This technology is a virtual representation of a physical object that spans its entire life cycle, essentially addressing one of the core ideas of Industry 4.0 - building cyber-physical systems. This chapter includes a literature study on related topics to better understand the aforementioned terms, technologies, and the demands on tools that enable them. After this, related works are reviewed followed by a conclusion of the chapter and its significance for the thesis.

2.1 DataOps

This section presents the definitions of DataOps and identifies its characteristics. The process of implementing a fully integrated DataOps solution in an organization is also discussed.

2.1.1 Definitions of DataOps

Upon reviewing the term DataOps, no standardized definition is found. How DataOps is described depends on the stakeholder and their interest. As an example, below are two different definitions given by business leaders in DataOps:

”DataOps is the orchestration of people, processes, and technology to deliver trusted,

business-ready data to data citizens, operations, and applications throughout the data lifecycle. With properly governed data, companies can comply with complex regulations and data privacy and ensure the accuracy of the AI model by monitoring data quality [1]*."*

"DataOps is a collaborative data management practice focused on improving the communication, integration, and automation of data flows between data managers and data consumers across an organization. The goal of DataOps is to deliver value faster by creating a predictable delivery and managing changes in data, data models, and related artifacts. DataOps uses technology to automate the design, deployment, and management of data delivery with appropriate levels of governance and uses metadata to improve data usability and value in a dynamic environment[2]*."*

By also reviewing the current literature related to DataOps, the above definitions can be extended and complemented to define the term better. In a study [4] conducted to move toward a standardized definition of DataOps, current literature about DataOps is investigated. The study investigates which components and characteristics are essential for DataOps solutions. It is concluded that most of the authors of the reviewed literature emphasizes continuous improvement and a culture of collaboration and trust as an underlying goal of DataOps. It is also emphasized that end-to-end thinking is a core objective. Another vital component found within DataOps is the data pipeline, which describes a process-oriented structure in which data are transferred through multiple stages of a process. The stages mentioned are data extraction, transformation, and visualization. It is also revealed that data-driven improvements, testing, and monitoring are the fundamental principles of DataOps. Furthermore, the study extracted common goals and principles related to the term in the literature. To determine which components were considered the most important, eight DataOps experts were interviewed and asked to note which components are the most valuable. The result can be seen in table 2.1.1. The interviewees are denoted with the letter C followed by a numbering in the table.

Based on this result, [4] derives the following definition:

"DataOps is a set of practices, processes, and technologies that combine an integrated and process-oriented perspective on data with automation and methods from agile software engineering to improve quality, speed, and collaboration and promote a culture of continuous improvement."

2.1.2 DataOps in an industrial environment

Today's production environments generate substantial amounts of data. This is especially the case in industrial environments, where data is continuously generated near machinery (sensors and controllers) and from various IT systems and on-site data centers. With the rise of Industry 4.0, the need for digitization and improved

[1] https://www.ibm.com/se-en/analytics/dataops
[2] https://www.gartner.com/en/information-technology/glossary/dataops

Table 2.1.1: Components in DataOps and their importance, according to experts interviewed [4]

	C1	C2	C3	C4	C5	C6	C7	C8
Goals:								
Continuous improvement	X	X	X	X	X		X	
Orchestration		X				X	X	X
Empowerment of citizen users		X			X		X	
Agility & Speed	X	X	X	X	X	X	X	X
Collaboration & Trust		X		X	X		X	
Principles:								
Reuse of artifacts	X	X						X
Automation	X	X	X	X	X	X	X	X
Integrated end-to-end thinking		X	X			X	X	X
Short cycles & Incremental change	X	X	X			X		
Analytics as code	X	X	X			X		
Testing	X	X		X		X	X	X
Monitoring	X	X		X	X	X	X	X
Data-driven improvement	X	X				X		
Process-oriented Data pipelines		X	X	X			X	X

efficiency in data consumption has also increased [5]. The importance of DataOps in the industrial setting is further emphasized. By looking at current literature [5] of DataOps in manufacturing and industrial settings, in an attempt to further extend the previously defined components of DataOps, a set of essential components can be identified, better aligned with the context of this thesis:

- **Standardization of data:** To get full value from any production data, an analysis must be performed on different machines, processes, and data sources. This kind of data landscape generally does not provide any standardization of its data. To handle this, a standard model must be established within the industrial DataOps solution.
- **Connect to industrial and IT systems:** IT systems and devices in industrial environments communicate through many different protocols, extensively using APIs and custom integration. Therefore, an industrial DataOps solution must seamlessly integrate with devices and data sources using industry standards.
- **Provide scale and security:** Compared with typical transaction data, industrial data differ. It usually comes from hundreds or thousands of devices and must be captured, contextualized, and delivered efficiently. For analytics and/or visualization to work optimally, it is also essential that the data meet real-time constraints that can arise when data must be available to applications in a timely manner. DataOps solutions must be secured and delivered discretely to applications in the industrial solution.

Understanding how the implementation and application process of DataOps is carried

out in a real-world environment is essential. It is not enough to know only the definitions and target attributes of DataOps and then build it. Real environments are often very complex, and implementing new technologies is not done in an instant. This is discussed next.

2.1.3 Towards DataOps

The implementation process of DataOps in an organization is not completed immediately. It is something that is built over time, evolving from basic data analysis in incremental steps. Despite the fact that the topic has received increasing attention in the industry, there are no clear definitions of how a fundamental data collection process (ad hoc) evolves into a fully automated data analytic process such as DataOps [10]. To highlight this and better define the steps leading to a fully integrated DataOps solution, [10] investigated the process of moving from ad-hoc data analytics to DataOps. The study was conducted through a multi-vocal literature study combined with a case study in a large mobile communication organization. Five phases, including DataOps itself, were identified. The phases can be seen in Figure 2.1.1, and are described in the bullet list below:

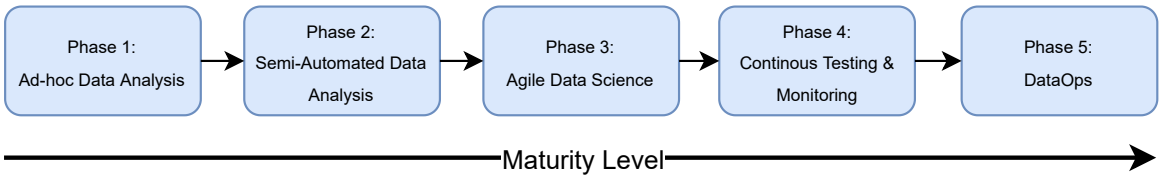

Figure 2.1.1: Phases [10] of moving from ad-hoc data analysis to DataOps

- **Ad-hoc data analysis** means that reports on data are created on demand, and in which the reports are highly customized to the specific scenario in question. In an ad-hoc solution, the user decides which source to retrieve data from and how to present it. Insights can range from simple data tables to more advanced visualization features. Ad-hoc solutions should also be able to deal with different data sources in a flexible and scalable way. The ad-hoc analysis is helpful when there are requirements to deliver immediate results, and the analyst knows what to look for. However, to make a good business decision, it is always better to have adequate data engineering, collection, and more extensive analysis. A big challenge with ad-hoc solutions is that the data collected from different sources will not be centralized to a single access point. The data are represented in data silos (raw data presented in systems that are isolated from each other, for example, is accessible by one department but not another), which prevents customers and users from getting the full picture of their data.
- **Semi-automated data analysis** is a more efficient and automated way of

collecting and processing data and can be implemented using data pipelines. A data pipeline is a process (or processing elements) of moving data from a source to a destination. During the process of moving the data, it is transformed and optimized, preparing the data for its destination and analysis. Thus, well-designed pipelines are necessary to efficiently evaluate, test, ingest, transform, validate, and publish large-scale data. Data technologies for data collection, engineering, processing, analysis, and visualization are also crucial through pipelines. However, there are some challenges in the data pipeline. One is that some activities are not automated. For example, monitoring is often done manually, and when an issue is found, the problem is also often solved manually. Problems that occur in a data pipeline are inefficient in terms of solving.

- **Agile data science** includes agile and DevOps methodologies, where product development and deployment processes are well defined. Teams can develop complete, tested, and functional code in short periods of time. As customer requirements often change over time, developers must evolve with these requirements. In agile methodologies, insights are delivered to customers after each sprint (short cycles), where feedback is provided, and rework is initiated if required. Some challenges in this phase are that without continuous automated testing, many man-hours are spent surveilling data pipelines. To provide quality insights, data must be available in a timely manner. Thus, monitoring and testing of data pipelines should be conducted continuously to acquire quality data that can be part of insights delivered to customers so that action can be taken immediately.
- **Continuous testing & monitoring** are essential elements of data pipelines, especially in systems that handle real-time data. As problems can be detected before the data cross successive elements of a pipeline, the problem can be prevented from escalating. If a problem is identified at the end of a pipeline, it is much harder to identify where it originates from. Furthermore, to meet quality constraints, it is important to have continuous automated unit tests running in the pipeline or parts of the pipeline. With automated alerts, teams can be notified when something is not right within the pipeline.
- **DataOps** shortens the end-to-end cycle time of data analysis, from data collection to obtaining insights from it. DataOps incorporates data pipelines to create valuable insights and innovation pipelines to push new analytics into the data pipelines. Continuous integration and delivery are the main goals of DataOps, as well as monitoring the data life cycle process.

2.2 Digital Twins

This section will present information on DTs. The section begins with an introductory section describing different definitions of the technology, followed by discussing information regarding the data requirements of the technology. As the aim of this

thesis is to evaluate a tool for data gathering to a DT, and not build a DT, focus is on the data requirements of the technology.

2.2.1 Definitions

With the DT being a relatively new concept that is currently being explored, there are no standardized definitions of what a DT is, as concluded in [15]. However, some common traits reappear in literature, and are also prevalent in definitions given by industry leaders on the subject. Below are three different definitions gathered from industry leaders in DTs:

"A digital twin is a virtual representation of a physical product or process, used to understand and predict the physical counterpart's performance characteristics. Digital twins are used throughout the product life cycle to simulate, predict and optimize the product and production system before investing in physical prototypes and assets [3]*."*

"A digital twin is a virtual representation of an object or system that spans its life cycle, is updated from real-time data, and uses simulation, machine learning and reasoning to help decision-making [4]*."*

"A digital twin is a virtual representation of a physical object or system – but it is much more than a high-tech lookalike. Digital twins use data, machine learning, and the Internet of Things (IoT) to help companies optimize, innovate, and deliver new services [5]*."*

Looking further into the definitions of DTs according to industry leaders, table 2.2.1 shows a set of DT characteristics and whether or not the characteristic is present in the industry leaders description of a DT [6] [7] [8] [9] [10] [11]. This highlights what is commonly regarded as important for the technology. In conclusion, a DT is a virtual model representing a physical counterpart throughout its lifecycle. This counterpart could be any arbitrary physical object, such as a phone, a building, an engine, or an industrial production system. In theory, there are no limits as to what a DT model is. A DT could essentially be a virtual representation of the entire industrial ground of a company, or a single production unit in a manufacturing plant. The DT should also use real-time data, which means that the data used should be gathered and used by the DT in a timely manner. However, no time constraints define what timely means, and the answer depends on the use case of the DT.

[3] https://www.plm.automation.siemens.com/global/en/our-story/glossary/digital-twin/24465
[4] https://www.ibm.com/blogs/internet-of-things/iot-cheat-sheet-digital-twin/
[5] https://www.sap.com/sweden/products/supply-chain-management/digital-twin.html
[6] https://www.plm.automation.siemens.com/global/en/our-story/glossary/digital-twin/24465
[7] https://www.sap.com/sweden/products/supply-chain-management/digital-twin.html
[8] https://www.ge.com/digital/applications/digital-twin
[9] https://www.ge.com/research/offering/digital-twin-creation
[10] https://docs.microsoft.com/en-us/azure/digital-twins/
[11] https://www.ibm.com/topics/what-is-a-digital-twin

Table 2.2.1: DT characteristics according to industry leaders

	C1	C2	C3	C4	C5
Virtual rep. Of physical entity	X	X	X	X	X
Spans lifecycle	X	X	X	X	
Measure performance	X	X	X	X	X
Predict performance	X	X		X	X
Real-time Data	X	X	X	X	X
Historical data					X
Bidirectional data flow	X				X
Uses ML	X	X	X	X	
Uses IoT/Sensor data	X	X	X	X	X

In addition, a DT is used to collect, analyze, and simulate data that can be used to improve performance and gain insight into the system/object it models. What performance means and on which metrics it is based also depends on the use case of the DT. Furthermore, DTs should help understand how a system performs and predict how it will perform in the future. When data derived from the modeled system/object are analyzed, predictions about the future can be made. A method that is prevalent among companies to perform data analysis is machine learning (ML). It is important to note that the key characteristic is the analysis itself, whereas ML is only a method to perform the analysis. That is, the effect of ML is what is desired, not the technique itself.

With the definitions, attributes, and goals of DTs described, it is also of interest to explore the requirements prevalent for the construction of the technology in the first place. What are the requirements for the data used and for the enabling technologies?

2.2.2 Data requirements, principles and enabling technologies for Digital Twins

To operate successfully, a DT must have access to available high-quality data that accurately represent the system it models. As a result of developments in technologies such as the Internet of Things (IoT), cloud computing, edge computing, big data, and artificial intelligence (AI), large amounts of data are being continuously generated. This is especially prevalent in industrial environments where operations are conducted around the clock and data are constantly being generated from operational devices. It is essential to determine what data requirements and principles should be followed to use the data efficiently. In [18], the requirements and principles for DT data are explored. The following segments concludes some of the essential topics that are discussed in the paper:

Comprehensive data gathering is necessary to enhance the accuracy, efficiency, and adaptability of the characteristics of DT. The comprehensive data in this context

refer to normal and abnormal states, common and rare events, and certain and uncertain events. A DT developed only based on data collected directly from a physical entity will probably face specific problems when the scenarios of lower probability finally occur, and the data belonging to these scenarios cannot be adequately handled. In the same sense, DTs built only using virtual models and simulated data might have problems handling disturbances and data that vary over time that often arise in data gathered from physical environments. As a result, DTs should be powered using both available data from a physical entity and simulated/virtual data. This is of benefit and interest when for example simulating a scenario on the DT that one does not want to try on the physical entity itself.

Real-time data is another important aspect of DTs. The data flowing between DT and the entity it models should be transferred in a timely manner, allowing for real-time interaction and representation of the modeled entity. The term timely depends on the scenario and the system to be modeled; for certain scenarios, it may not be of great importance to have a real-time representation DT, and for other scenarios, it might be crucial to have an accurate, timely representation. To restore consistency, there should also be a way to detect inconsistencies in transmission times. Phrased differently, there should be some means of monitoring the data.

Data universality is an obstacle to implementing DTs, as large environments with different types of data landscapes, application scenarios, data types, structures, interfaces, and communication protocols often result in low universality of data. Therefore, it is important to effectively transform and standardize the data to take advantage of data gathered from multiple sources. This will make data exchange with a DT and its modeled entity(s) more efficient.

Furthermore, there is also research on enabling technologies for DTs. For example, in [13], work is carried out to review methodologies and techniques related to building DTs, aiming to cover challenges and enable technologies in the research area. One of the enabling technologies identified in this paper is data pre-processing. This includes gathering and compressing data on the fly while offering capabilities such as automatic outlier detection. Another identified key components are big data, which means infrastructure to store and process large volumes of data. Lastly, the enabling techniques must also be able to deliver data on time. That is, it must be processed in a timely manner without any latency.

2.3 Load, Performance, and Stress Testing

Large-scale software systems and tools must support concurrent access and usage from multiple sources. This is also the case for the tools used within DTs to collect data, as the data can come from many different sources. Studies have shown that failures in these systems are often caused by their inability to scale to meet usage demands rather than failures due to feature bugs [16]. To ensure the quality of these systems, testing techniques can be applied to evaluate them. Three of these techniques are load,

stress, and performance testing. These terms are often used interchangeably but can be defined as distinct techniques that share a relationship. Below is a description of each technique, as defined in [7]:

- **Load Testing** is a process in which the behavior of a system is assessed to detect load-related problems. The rate at which the SUT is submitted to requests is the load. Problems evaluated can be functional problems, such as deadlocks, buffer overflows, memory leaks, or non-functional problems. These problems violate quality-related requirements such as reliability, stability, or robustness.
- **Performance Testing** is a process in which performance-related aspects of a software system are measured. Such metrics include response time, processing time, and resource utilization. In contrast to load testing, performance tests are broader. Performance testing can verify performance requirements (pass-fail criteria) but can also be exploratory (no clear pass-fail criteria).
- **Stress Testing** is a process in which a SUT is placed under extreme conditions to verify its robustness and/or detect load-related problems. Test conditions can, for example, be reached by applying load, limiting computing resources, or producing failures.

2.4 Related Work

As presented in previous parts of this chapter, there is broad and an increasing amount of research regarding the concept of DTs and their data requirements. There is less research regarding the gathering of data for the twins. This conclusion is also made in [3], where the process of making DTs a reality is discussed. The paper concludes that there is a need for more research regarding data collection and processing methods for DT data. More specifically, research is needed to improve the traditional methods of collecting and processing data, and to implement a communication interface between real environments and physical twins. Other papers recognizes similar problems as well. One such paper is [9], where it is stated that the DT is a concept of many promises and potentials, facing many challenges. One such challenge is the complexity in data accumulation and lack of processing power to support DTs. The study also states that a communication medium between the real environment and the DT is essential, but that the choice of this medium will be totally dependent on the communication requirements of the DT. This strengthens the idea that there are no specific enabling technology that is commonly accepted as a standard for DTs. Another challenge of DTs and current literature is the lack of research regarding general implementations of DTs and their enabling technologies. As stated in [6], the majority of current literature focuses on conceptual development of DTs frameworks and tools that are designed for specific implementation areas. Thus, more research regarding general frameworks and enabling tools that could be used to implement DTs is motivated.

Current literature regarding DataOps has also been reviewed in this chapter,

concluding definitions and characteristics of the term. However, it has not yet been discussed what kind of tools that are available to implement DataOps. A similar question was posed in [8], where a goal was to provide a comprehensive overview on tools and their suitability for DataOps. The paper discusses different tools to provide ideas and shed light on the large variety of tools that exists, but in a very general manner without any actual tests and evaluations. It is emphasized that a single tool does not necessarily need to provide a full fledged DataOps solution, but that it can aid in some stages of DataOps. In short, different types of tools are thought to be useful for different kinds of situations. The paper does not provide any actual testing, with any specific tool, and it is concluded that a topic for future research is to evaluate the performance of different tools in different use cases, in a real project. In other terms, using a real tool and evaluating its performance as a DataOps solution is a topic that could provide research value.

2.5 Conclusionary Remarks

Having described both DataOps and DTs, some conclusionary remarks are given to connect their characteristics and data requirements to the topic of the thesis, and to provide motivation for the testing that is conducted.

2.5.1 DataOps

Beginning with DataOps, it is clear that some key characteristics of a DataOps solution is the ability to standardize data, connect to industrial and IT systems, provide scalability, and meet real-time constraints. This means that a logging tool used for the purpose of implementing DataOps should be able to handle and process small amounts of data, but also large amounts. This data should also be standardized. As data originating from different kinds of sources might have different formatting, it is important to standardize it to efficiently take use of it. A DataOps solution should also have the ability to gather data from different sources – ranging from hundreds to thousands of units. As the data landscape in an industrial setting often consists of different types of devices (machine controllers, PLCs, sensors, etc), protocols, APIs and custom solutions, it is important that an efficient DataOps tool is able to connect and gather data from different kinds of sources. Gathering data from many different sources also implies an increased amount of logged data that the tool must handle. Another very important characteristic is the ability to process data in a timely matter, meeting real-time constraints that might be put on the system – even if the load is increased.

To evaluate a logging tool for its ability to operate as a DataOps solution, the above characteristics must be related to the qualities and features of the tool. This thesis aims to evaluate how suitable a logging tool might be for being used in DataOps, by analyzing the features of the investigated tool and relating it to the above mentioned characteristics. Further, a practical testing process is conducted to investigate how

timely the tool is in processing data. Timeliness is crucial in DataOps, and it should integrate seamlessly into the system where it is used. A tool which can not provide timeliness is not an optimal DataOps solution – even if other qualities and characteristics are fulfilled. Thus, focus is put on timeliness.

2.5.2 Digital Twins

The data requirements for enabling a DT relates well to the characteristics of a DataOps solution. Firstly, data used for a DT can originate from many different kinds of sources. A tool used for gathering data to a DT must therefore be able to connect and gather data from many different end points. Standardizing this data could enable a DT to more effectively take use of it. Further, data used should be comprehensive in the sense that it does not only consist of real data. As a result of DTs often being used for simulation and analyzation, enabling data gathering tools should also be able to deliver synthetic data to a DT – enabling the DT to evaluate different events that might not happen very often in a real-life setting. Lastly, timeliness is also very important for DTs.

Similarly to what was concluded for DataOps, a logging tool used for enabling DTs should have the features and qualities which enables the characteristics and requirements mentioned above. Again, this thesis aims to evaluate how suitable a logging tool might be for being used as a enabling technology for DTs. This will be done by evaluating the features of a logging tool and relating it to the characteristics and data requirements for DTs. A comprehensive testing process is also conducted to test the tool for its ability to process data in a timely manner. Again, this quality is of great importance and timeliness will dictate what kind of DTs the tool could be suitable for.

Chapter 3

The Logging Tool

This chapter will describe logging tools, with a focus on Nodinite – the SUT of the thesis. A description of its logging mechanics and other functionalities offered by the tool is given.

3.1 Introduction

A data logger is a device or software system that automatically gathers and stores data, often originating from real-life sensors, devices, or IT systems. They are often used to collect data for quality control, predictive maintenance, and troubleshooting purposes. In its simplest form, these data are stored in a spreadsheet where a person analyses them manually [2]. In other cases, the data might be processed more sophisticatedly through third-party software or within the logger itself. In environments such as industrial ones, many different IT systems communicate and depend on each other, whether they concern orders, production statuses, sensor data, or something else. A logging tool will log events that occur within such an environment. The data will be logged to a standardized platform for the data consumer to gain control and insight. Often, features are offered to monitor critical data flows, where a monitoring system can send alerts if something is wrong or a specific event is registered. Essentially, a logging tool will centralize information flows in an environment from many different systems to a single platform. Nodinite is one of those logging tools with features mentioned above [1].

3.2 The Logging Service and Log API

For the logged data to be presented in a user friendly manner and for it standardized, the data require some form of processing. In Nodinite, there is a Logging Service that is responsible for processing and reprocessing the events and messages that are being logged. The service comprises one or more Log Agents, which will create log events

[1] https://www.nodinite.com/features/

from one or more integration brokers that carry information about events and an optional payload. Events are communicated to the tool through a log API that enables end-to-end logging across many system integration platforms. With the API built in Representational state transfer (REST) and Windows Communication Foundation (WCF), it is possible to create custom solutions that allow logging of arbitrary data. The ability to build custom solutions to record data from any arbitrary data source is important to make a recording tool scalable and applicable to as many systems as possible. REST and WCF will support the following protocols to communicate with the API [2]:

- Hypertext Transfer Protocol (HTTP) (REST and WCF)
- HTTPS (REST and WCF)
- MSMQ (WCF)
- TCP (WCF)

Figure 3.2.1: JSON Log Event details.

When sending custom log events to the log API, a format that can be used is JSON. These JSON log events include three distinct parts: *Event details* (mandatory), a *payload* (optional), and *context properties* (optional), as seen in fig. 3.2.1. The payload can, for example, also be a JSON file, which will then be logged into Nodinite. Within Nodinite, all or individual data fields of this JSON file can be explicitly stored, as described in the next section. In fig. 3.2.2, a simplified visual representation of the logging architecture is given, using custom solutions built using the log API.

[2]https://www.nodinite.com/features/

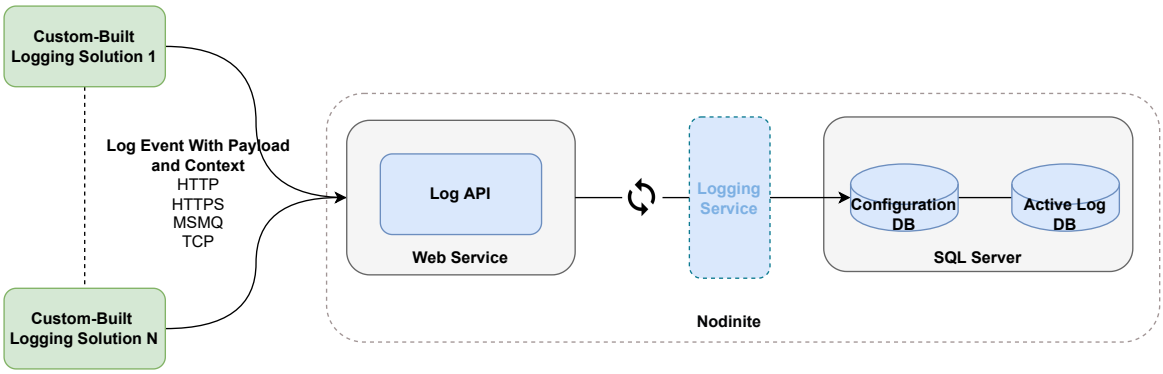

Figure 3.2.2: Log API overview

The Logging Service within Nodinite is set up to gather logs through the API based on two parameters. The first specifies how often the service should look for new messages, and the second specifies how many messages the service should process in each iteration. The system parameters are called *MessageProcessingTimerInterval* and *MessageProcessingBatchSize*. This means that logs sent to the Log API will only be fetched by the Logging Service and processed in batches fetched with certain intervals.

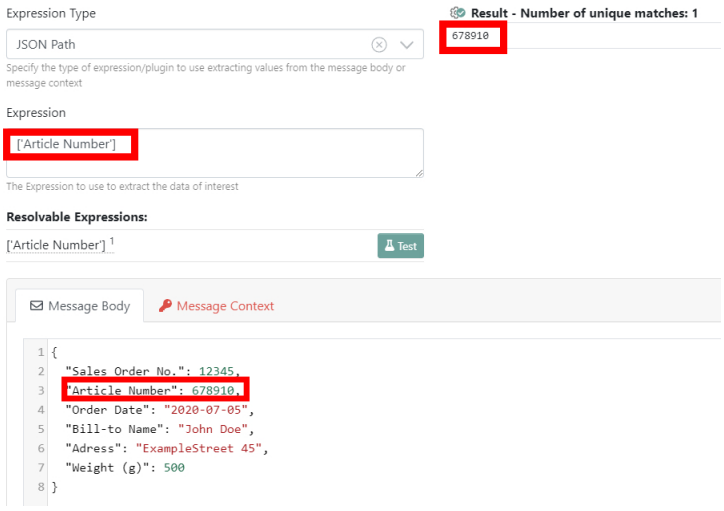

Figure 3.2.3: Illustration of the Search Field Functionality in Nodinite

3.3 Search Fields

Another feature offered by Nodinite is the feature to explicitly fetch specific data fields in the logged data. This is done using *Search Fields*. A *Search Field* facilitates search conditions. The functionality extracts values from log-event payloads and explicitly stores them in a database. To fetch these values, a *Search Field Expression* is applied to a message type, for example JSON, to extract a specific value. Figure fig. 3.2.3 is an example that illustrates this feature. As can be seen, a specific value can be extracted from the JSON file. In this case, the value is stored as *Integer*, but Nodinite also

supports *Text, Long Integer, Real numbers with two decimals, GUID, Double, Date time with offset* and *Bitwise Integer*. The *Search Field Expression* will persist for all similar logs/events from the same source (or multiple sources), which means that the value (*Article Number* in this example) will be extracted for all logged events of similar type. It will be explicitly stored in Nodinite's log-database, as seen in fig. 3.3.1. This means that logged data, either entire logs or just specific fields, can be stored in Nodinite no matter where it originates from and its original formatting. In addition, there is also an option to monitor the values logged in Nodinite. Using this monitoring system, Nodinite can notify the user if a data field reaches a specific value or exits a specified interval.

	SearchFieldValueId	EventId	SearchFieldId	Value
1	C1667B70-67FC-EC11-8A19-0022489B06A7	7421EA7B-62FC-EC11-8A19-0022489B06A7	21	678910

Figure 3.3.1: Search Field value stored in the log database

Chapter 4

Design and implementation

The following chapter will provide a practical description of how the thesis work was implemented, the goals of the implementation, and the design choices. The testing process aims to perform a combination of load, performance and stress testing methodologies to evaluate the SUT, Nodinite, for its suitability as a DataOps solution used for data collection to DTs. Specifically, the testing will evaluate the timeliness of the tool. The log processing times (timeliness) are measured under different loads in an exploratory sense. That means that no clear pass/fail criteria are used. Lastly, a stress test places the SUT under more extreme conditions. When conducting these tests, it is also assessed how Nodinite behaves when the logging frequency is increased and multiple instances feed data to the tool.

4.1 Testbed

A test bed is built to conduct rigorous, transparent, and replicable testing processes. This testbed consists of a Nodinite instance, an environment on which it can run, and custom software to test the SUTwith. The instance runs on a production server provided by Kazoku IT AB. As these components are located on a server, there is also a need to establish a remote desktop connection. Included in the test bed is also the Nodinite Web Client. The testbed is depicted in fig. 4.1.1, followed by a description of its components.

- **Nodinite Instance:** An instance of Nodinite specifically dedicated to the testing of this project was set up on the server provided by Kazoku IT AB. This allowed uninterrupted testing to be carried out, producing results that would not be affected by external events or disturbances.
- **Personal Computer and Remote Desk Connection:** An authenticated remote-desk connection was used to reach the server.
- **Nodinite Web Client:** Nodinite Web Client acts as an interface for the Nodinite instance. Allows access to the instance and its functionalities to see, configure,

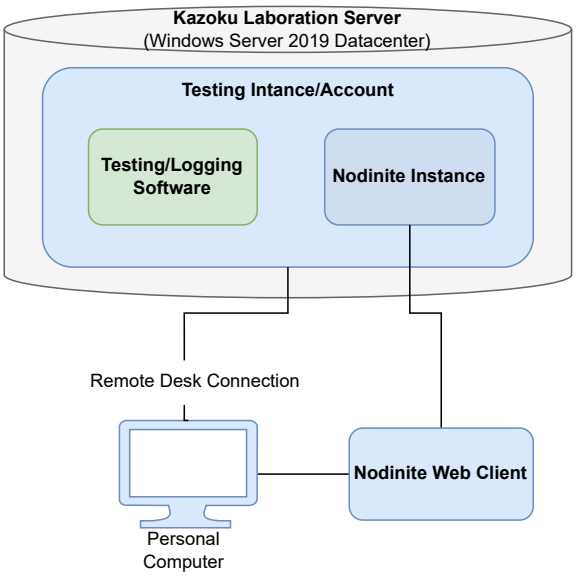

Figure 4.1.1: Architectural overview of test bed

and monitor Nodinite as the testing process is conducted.

- **Testing Software:** Custom testing code was built using Python. This code was executed using Visual Studio code on the server.

The next paragraph will explain how the test is conducted and why the testing strategy is designed the way it is. Alternative designs are also discussed. In addition, some comments are made on the test bed and the reasons for its architecture.

4.2 Logging and Measuring Setup

As described in Section 2.6.4, the tool offers support for using custom-built solutions to communicate with the log API. This enables complete control over what is logged and when an event is logged. It is a time-efficient option, as direct HTTP requests to the log API can be made using REST, instantly sending the desired data/logs to the tool. Thus, building a custom solution that sends automatically generated data directly to the Log API using REST is the base technique for logging during testing. Additionally, the built-in log-generating/testing software is run on the same server as the logging tool's instance rather than remotely. Although there would be some benefits to running the tests locally and having the logging tool located on a server, such as evaluating the tool as a solution implemented in the cloud, we want to conduct a full-fledged stress test not affected by external variables.

Testing is centered around measuring the processing time of logged data. That is, the time it takes from receiving an event to the point where it is logged, ready, and processed in the log database. As mentioned above, the data are generated through a custom-built solution. This system can vary the sizes of the events, their contents/amount of parameters, the frequency at which they are sent, how many

events to send, and how many instances that generate and send events concurrently. In fig. 4.2.1, the architecture of the testing process is shown.

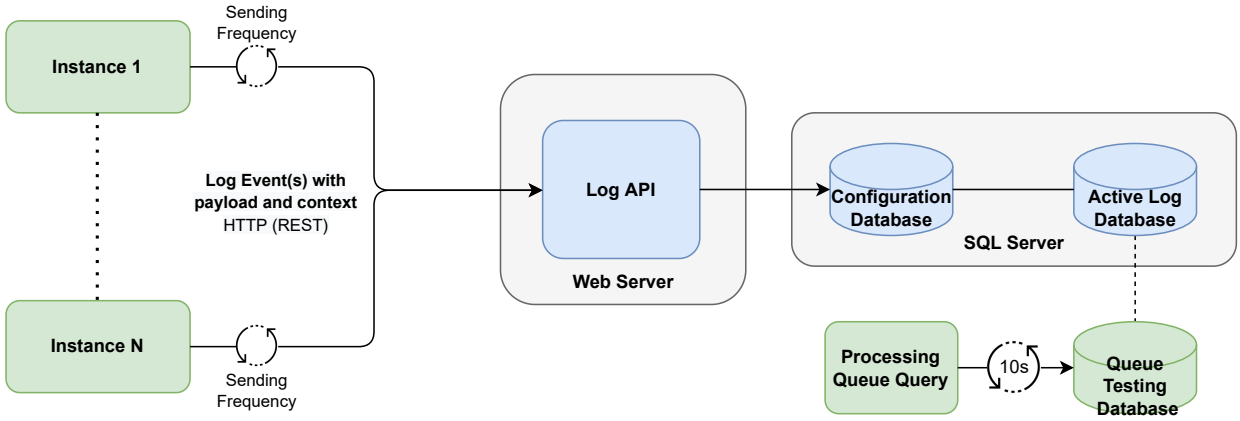

Figure 4.2.1: Architectural overview of the testing process

In addition to generating and sending events to Nodinite, there is also a way to measure the processing times. Each event receives a time stamp within the SUT when it arrives in the system, labeled *Created*, and a second when it is finished processing and ready in the active log database, labeled *Changed*. There is also another label called *LogDateTime*, which includes the time in which the data was initially created in the testing software. In fig. 4.2.3 and fig. 4.2.2 these timestamps are shown. We can derive a processing time by subtracting the completion time with the arrival time (*Changed - Created*). By measuring only the timestamps within SUT, the testing is isolated to the SUT in a black-box manner: only focusing on internal processing times, and not taking into account the time between *LogDateTime* and *Created*. This also minimizes the effect that external factors, such as the efficiency of the event generator, have on the tests.

Furthermore, a way to measure the current amount of unprocessed logs within the tool is set up. A health check of the logging process can be performed by regularly observing how many logs are unprocessed. A new database contains the current amount of unprocessed logs in SUT. Additionally, a query is built that, with an interval of 10 seconds, fetches this amount from the database along with a timestamp. In this way, eventual correlations between the number of unprocessed logs and processing times can be observed. The process can be seen in fig. 4.2.1.

Size	LogDateTime	Created	Changed
646	2022-06-16 16:58:37.3140000 +00:00	2022-06-16 16:58:37.3274070 +02:00	2022-06-16 17:08:59.4877097 +02:00
648	2022-06-16 16:58:37.1890000 +00:00	2022-06-16 16:58:37.2024237 +02:00	2022-06-16 17:08:59.4659777 +02:00
647	2022-06-16 16:58:37.0640000 +00:00	2022-06-16 16:58:37.0774249 +02:00	2022-06-16 17:08:59.4503195 +02:00
647	2022-06-16 16:58:36.9390000 +00:00	2022-06-16 16:58:36.9524529 +02:00	2022-06-16 17:08:59.4503195 +02:00
646	2022-06-16 16:58:36.7990000 +00:00	2022-06-16 16:58:36.8117869 +02:00	2022-06-16 17:08:59.4346945 +02:00

Figure 4.2.2: Snapshot of Nodinite Database during stress tests, showing log timestamps

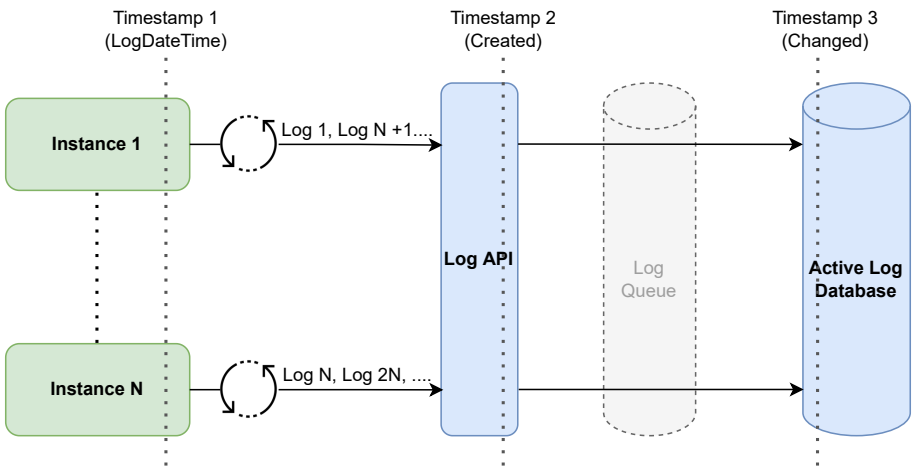

Figure 4.2.3: Testing process

4.3 The Logs and Testing Variables

An important design decision is what the logs should include: what kind of information will be sent and logged to the SUT. As the tests in this project aim to evaluate the SUT for DTs in general and not a specific use case, the data logged should be of relevance, and the tests should apply to the general DT. The data should also be available and time-efficient to gather/create, as generating events in these tests will be done rapidly and frequently. Therefore, it is decided to log system utilization data. The Python library python-system-and-process-utilities (psutil) fetches these values. Additionally, the events are generated in JSON file format. This format is lightweight and can be generated efficiently and commonly used.

Additionally, there is a set of critical variables in the testing process. These variables can be split into two groups: a set of input (independent) variables and a set of output (dependent) variables. The input variables are used to vary, modify, or tweak the testing mechanics, and the output variables are the outputs resulting from the input variables. The output variables are the metrics used to measure the performance and behavior of the SUT in the tests. In fig. 4.3.1, these variables are visualized.

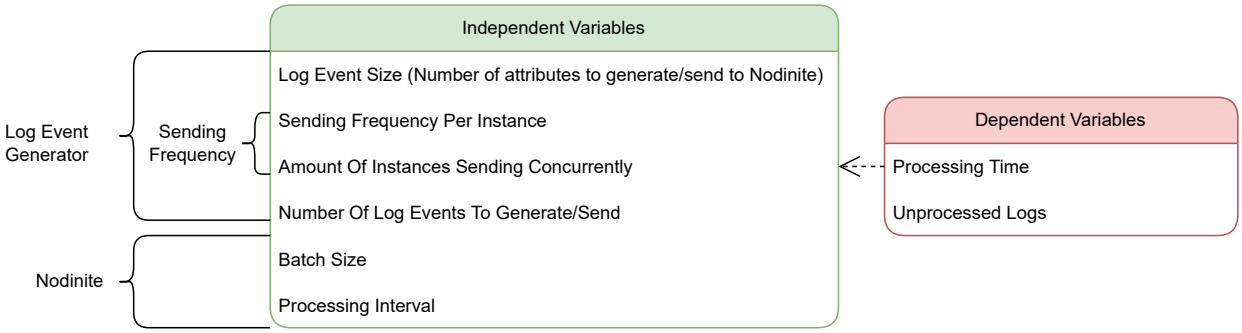

Figure 4.3.1: Independent vs Dependent Variables

As previously described, the software built to test the SUT can vary the sizes of the logs, the frequency at which they are sent, how many logs to send, and how many instances

sending concurrently. The sizes are set by varying the number of attributes of the system (presented in the above listing) included in the logs and thus also stored in SUT. More specifically, three types of logs $\approx 165, 335$ and ≈ 650 bytes are used. As we will see, these sizes and their differences were enough to show a change in behavior in the SUT, measure its timeliness and identify its qualities. Other sizes and log complexities could have been used, but this would also require the log generator to be more efficient. In fig. 4.3.2, the log attributes are shown, where each attribute is represented as it is in the Python code used to generate them. By dividing the logs in this way, testing can be conducted to show eventual differences in the SUTs processing related to log size.

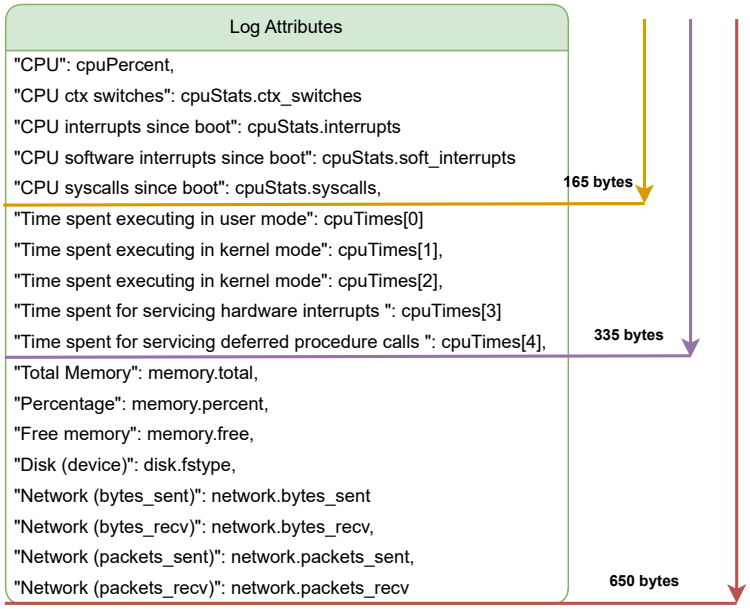

Figure 4.3.2: Log Attributes and Log Sizes

The second variable is the sending frequency. Two sub-variables determine this variable: the sending frequency per log-generating instance and the number of instances sending concurrently. This means that the sending frequency can be increased either by scaling vertically or horizontally. Vertically increases the sending speed of the individual instance and horizontally increases the number of instances; however, the end result is the same for both. From the SUTs point of view, the effect is the same. Finally, the total number of logs generated in each testing session can be decided. The variable is set by deciding how many logs each log-generating instance should generate. There is also a set of independent testing variables found within the SUT, which will be described in the next section.

4.4 Nodinite Setup

Most of the design and mechanics to perform the tests are outside of SUT. However, some settings are made within the tool to prepare it for tests. Firstly, a *Log Agent* is set up in Nodinite that listens for events sent by the custom solution (see 3.2). Second, the

Search Field feature (see 3.3) is set up to collect and store the attributes from the JSON files being logged. In fig. 4.4.1 the JSON file is seen and in fig. 4.4.2 these attributes have been stored in Nodinite.


```
{
  "CPU": 75.0,
  "CPU ctx switches": 2594803469,
  "CPU interrupts since boot": 2167795998,
  "CPU software interrupts since boot": 0,
  "CPU syscalls since boot": 3812879041,
  "Time spent executing in user mode": 638714.07812499988,
  "Time spent executing in kernel mode": 6499335.984375,
  "Time spent for servicing hardware interrupts ": 4435.78125,
  "Time spent for servicing deferred procedure calls ": 4827.15625,
  "Total Memory": 17179398144,
  "Percentage": 70.0,
  "Free memory": 5146042368,
  "Disk (device)": "NTFS",
  "Network (bytes_sent)": 41348982952,
  "Network (bytes_recv)": 302719702014,
  "Network (packets_sent)": 94645229,
  "Network (packets_recv)": 269556980
}
```


Figure 4.4.1: JSON file (650 bytes) containing the attributes to log

Furthermore,
there are two independent testing variables within Nodinite. These system parameters decide the batch size and processing interval, named *MessageProcessingBatchSize* and *MessageProcessingTimerInterval* in Nodinite. Varying the batch size will alter how many logs the Logging Service will fetch and process in each iteration. And the processing interval specifies how often the Logging Service will look for new logs to fetch and process. These variables and their functionalities are described in 3.2. As a base set, the batch size is set to 100 logs and the processing interval to 15s – the standard value of the SUT. This is also the starting variable used for the testing.

Search Field	Values
CPU Load	75
Memory Percentage	70
Free Memory	5146042368
Disk (device)	NTFS
Network (Bytes Sent)	41348982952
Network (bytes_recv)	302719702014
Network (packets_sent)	94645229
Network (packets_recv)	269556980
Total_Memory	17179398144
CPU ctx switches	2594803469
CPU interrupts since boot	2167795998
CPU software interrupts since boot	0
CPU syscalls since boot	3812879041
Time spent executing in user mode	638714.078125
Time spent executing in kernel mode	6499335.984375
Time spent for servicing hardware interrupts	4435.78125
Time spent for servicing deferred procedure calls	4827.15625

Figure 4.4.2: Attributes from JSON (650 bytes) file stored in Nodinite using the *Search Field* feature

Chapter 5

Evaluation and Result

This chapter will answer the research questions of this thesis. First, the result of the literature study will be presented, followed by the testing results and, finally, an overall evaluation. As the information for answering the first question has already been gathered in Chapter 2, this chapter will focus on the two latter questions.

1. What are the characteristics of a DataOps solution and the data requirements of a DT?
2. Can a data logging tool be used as a DataOps solution to collect and make data available for a DT?
3. What is the timeliness of the tool under test?

The tests presented in this chapter are conducted with the primary goal of evaluating the timeliness of the SUT, and to evaluate its suitability as a DataOps solution for DTs. Focus is put on the characteristic of timeliness in the SUT, as this a quality that is not as easily verified by directly analysing the features offered by the tool and its architecture. It is a quality that is not documented, as opposed to other features of the tool which can be connected to important qualities of DataOps solutions and DTs. We know that the tool can gather and process data in certain ways, but not how efficiently – and thus it must be further investigated. Additionally, timeliness is crucial for a DataOps solution. How efficiently the tool can process data will also dictate which DTs it could be applicable for.

5.1 Results from the Literature Study

One of the goals of this paper was to find the characteristics of DataOps solutions. In the literature study that was conducted, we found that one very important component of DataOps was speed, essentially meaning that a DataOps solution should be fast and improve speed within the data system that it is implemented. An example of this could be processing times, the solution must be able to process its data very efficiently, seamlessly integrating into the system where it is deployed. Continuous improvement,

automation, and monitoring were other principles named by experts in the field to be important in DataOps. In addition, to better understand DataOps within Industry 4.0, we look at DataOps in manufacturing and industrial settings. Here, a very important component was the ability to standardize the data. As processing must be done on data originating from many different sources in industrial environments, it is important that the data is standardized and a standard model established. It is also important that the DataOps solution is able to connect to different industrial IT systems, to gather the data in the first place. Lastly, it is also concluded that a tool used for DataOps must provide scalability, the ability to gather data from hundreds to thousands of different sources concurrently. Again, it is stated that this must be done while meeting real-time constraints. The tool must not suffer in its processing efficiency even though an increasing amount of data is being logged, ultimately increasing the load on the system.

Moving on, it was also investigated what the requirements on the data used for DTs were. These requirements are well related to the characteristics of DataOps. Firstly, it was again important that the data provided to a DT is fed to it in real-time. However, it can be argued that this might not be the case for all DTs. For example, a DT only used for analytical purposes will be less sensitive to time than a DT used to communicate and provide direct feedback to devices on the production floor in manufacturing. However, it must be determined how timely an enabling tool for DTs can process data. If one does not know its timeliness, it is also very hard to determine what kind of DT it could be used for. Another important ability for a DT, or a tool gathering data for one, is to detect inconsistencies in the data. This can be done by, for example, monitoring a certain parameter and sending out alerts if a specified change is detected. This is well related to the monitoring ability mentioned for DataOps as well. Just as for DataOps solutions it is also crucial that the data are standardized, again as a result of the data originating from many different sources. If a DT is to gather data from a tool, it is much more efficient if those data are standardized and follow the same format. The DT will then be able to take advantage of this data much more efficiently. Lastly, the data provided should both be real data and artificial data if needed. This enables DT to operate on both actual data from a real physical entity and to be fed with artifical data for analytical and testing purposes.

5.2 Testing results

This section presents the results of the tests.

5.2.1 Load and Performance Testing

The first tests were carried out by varying the parameters *Log Event Size* and *Sending Frequency*. Furthermore, the tests were carried out in three blocks, each of which had a different sending frequency. In each block, three log types of different *Log Event Size* and contents were used for the testing. The result from the first block is

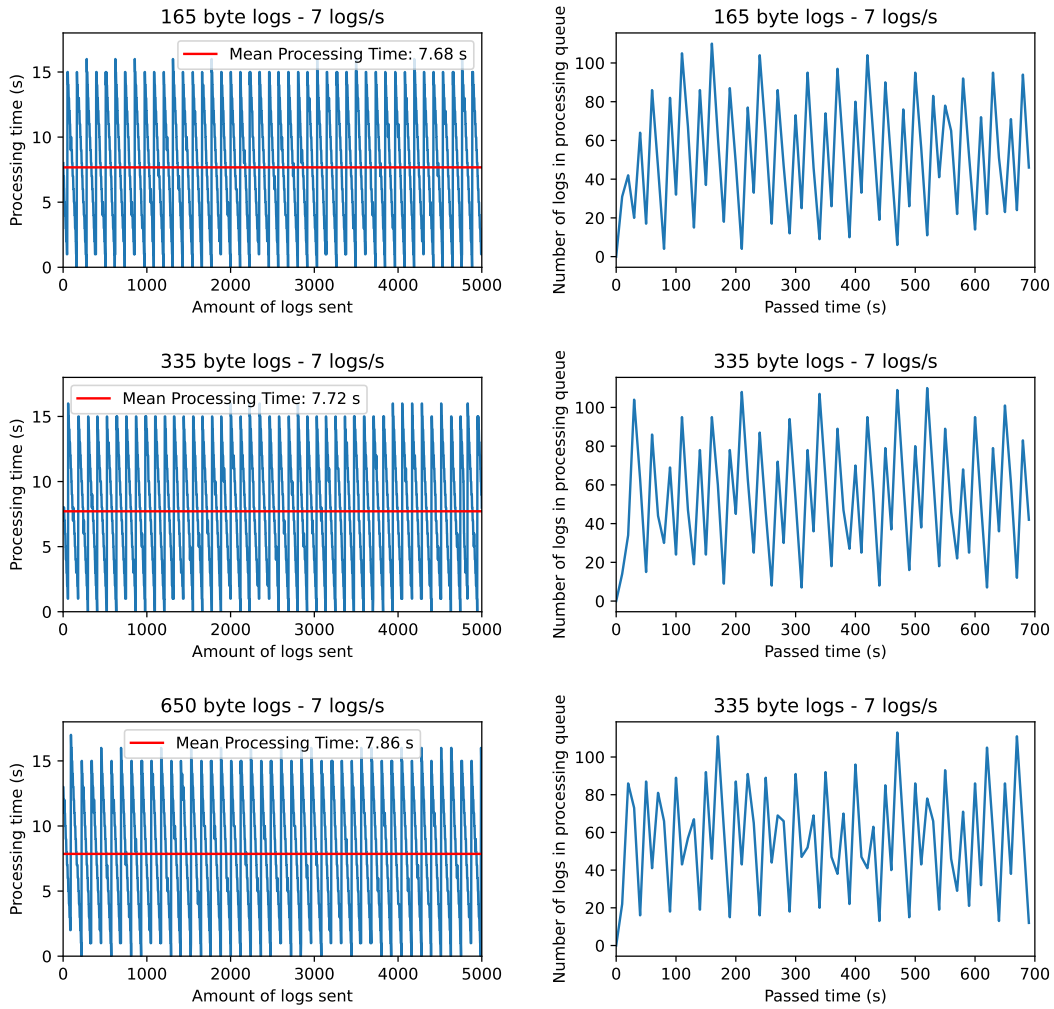

Figure 5.2.1: Sending frequency: 7 logs/s, Amount of logs sent: 5,000 , Instances sending: 1, Batch size: 100 logs, Processing Interval: 15s

shown in fig. 5.2.1. In column one, the processing times of the logs are shown for log sizes of 165, 335, and 650 bytes. In the second column, the amount of unprocessed logs within the SUT is shown for each of the log sizes. The sending frequency per instance is 7 log events per second, and only one instance is sending concurrently. The batch size (*MessageProcessingBatchSize*) is set to 100 logs, and the processing interval (*MessageProcessingTimerInterval*) to 15s – the standard values used by the tool.

Looking at the first tests (shown in fig. 5.2.1), the average processing time ranges from 7.68s for 165-byte logs to 7.86s for 650-byte logs. The processing times for the largest logs are approximately 2.3% higher than for the smallest ones. When comparing the logs of 335 with 165 bytes, the difference is even smaller. In general, the differences between the sizes are small. With such small differences, it is hard to say with confidence that the size difference is the cause of the processing time difference

for these particular results. However, as other tests will show, log size can substantially impact the processing time. Another thing to note is that the average processing times do not change over time, and that the processing times on an individual level oscillates within a fixed interval. This means that the tool provides stable and consistent average processing times (for the parameters in this specific test), while the individual processing times fluctuates. Looking at fig. 5.2.1 again, we see the processing times ranging from well below 1s to slightly above 15s, with the average being around 7.5-7.9s. This behaviour is concluded to be a result of how Nodinite *Logging Service* operates. As described in 3.2, Nodinite will fetch log events from the API with a regular interval set by the *MessageProcessingTimerInterval* variable. This means that certain logs will take a longer time to process than others, depending on when it is sent to the API in relation to when the *Logging Service* will fetch logs from the queue. A log that is sent directly after a fetch has recently been made will take a longer time to process than a log that is sent directly before a fetch is made. Essentially, some log events will spend more time in a non-productive state than others. This kind of architecture and internal structure of the tool will result in high average processing times – which we have seen already in this first test. It can also be seen that the number of unprocessed logs (column 2) follows the same pattern as the processing times. We also see the effect of the data being fetched in intervals: The amount of logs waiting to be processed oscillates in a pattern that correlates well with the processing times. However, this pattern is not as regular. The reason for this is that the current number of unprocessed logs is checked every 10 seconds in the tests. If this interval were faster, the pattern would likely be more regular and would correlate better with processing times. The take-away is that processing times correlate with the amount of unprocessed logs. Finally, when you look at the processing times as absolute values, they are very high even at these initial tests where the logging frequencies are relatively low. Although some individual log processes are at the lower end of the 0-1s range, an average of 7.6-7.9s is very high for efficient use in DataOps.

The sending frequency was then increased to 35 logs per second to observe any change in behavior. In fig. 5.2.2 the results of this are shown. As can be seen, the result is very similar to the first test, with the main difference being the processing times. For all log sizes, an increase in processing times is observed. More specifically, the processing times increase to 8.55-8.81s, compared to 7.68-7.86s in the first test. This is an increase of roughly 11-12% – while the increase in logging frequency from 7 to 35 logs per second corresponds to a 400% increase. This suggests that the tool can scale to a higher logging frequency quite well. However, as later tests will show, this behavior will be broken. Looking at the processing times in absolute values again, they are still very high in the context of DataOps.

For the third block of tests, shown in fig. 5.2.3, a clear difference in behavior is observed. First, the pattern for the processing times is different. In these tests, the pattern of regular highs and lows is still observed, but much less frequent, with only a small amount of local maxima and minima. As an example, only two minima of the processing times are observed for the 165-byte logs compared with 25 in fig. 5.2.2.

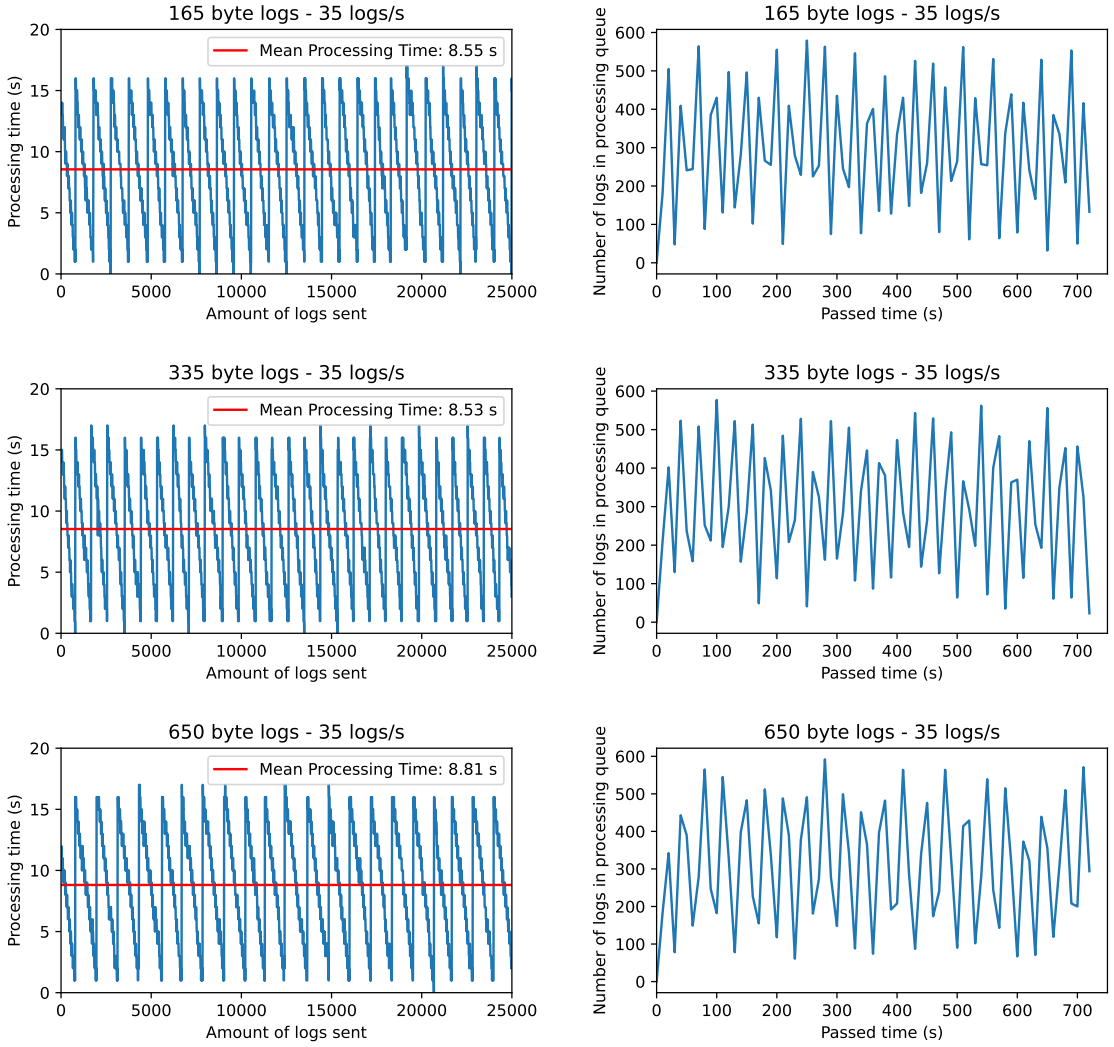

Figure 5.2.2: Sending frequency: 35 logs/s, Amount of logs sent: 25,000, Instances sending: 5, Batch size: 100 logs, Processing Interval: 15s

The processing times still oscillate around the mean processing time, but in a much less fluctuating pattern. Looking at the 165- and 335-byte logs compared with those in fig. 5.2.2, processing times are increased from roughly 8.5s to 9.1-11.1s on average – an increase of 7-30%. This increase is higher than what was observed between the first two tests in fig. 5.2.1 and fig. 5.2.2, although the increase in logging frequency (in percentage) for this test was lower. 77-89% (35 to 62-66 logs per second) compared with 400% (7 to 35 logs per second). This suggests that the SUT is getting less efficient in scaling to higher logging frequencies. It is also noted that the processing times for the 335-byte logs are on average faster than those of the 165-byte logs - 9.06s compared to 11.12s. This means that for this test the lower frequency resulted in a high processing time. The reason for this is unclear, and even though multiple tests were run, the result persisted. However, this behavior does not change the overall outcome or conclusion of the tests and is thus not investigated further. Lastly, when looking at the number

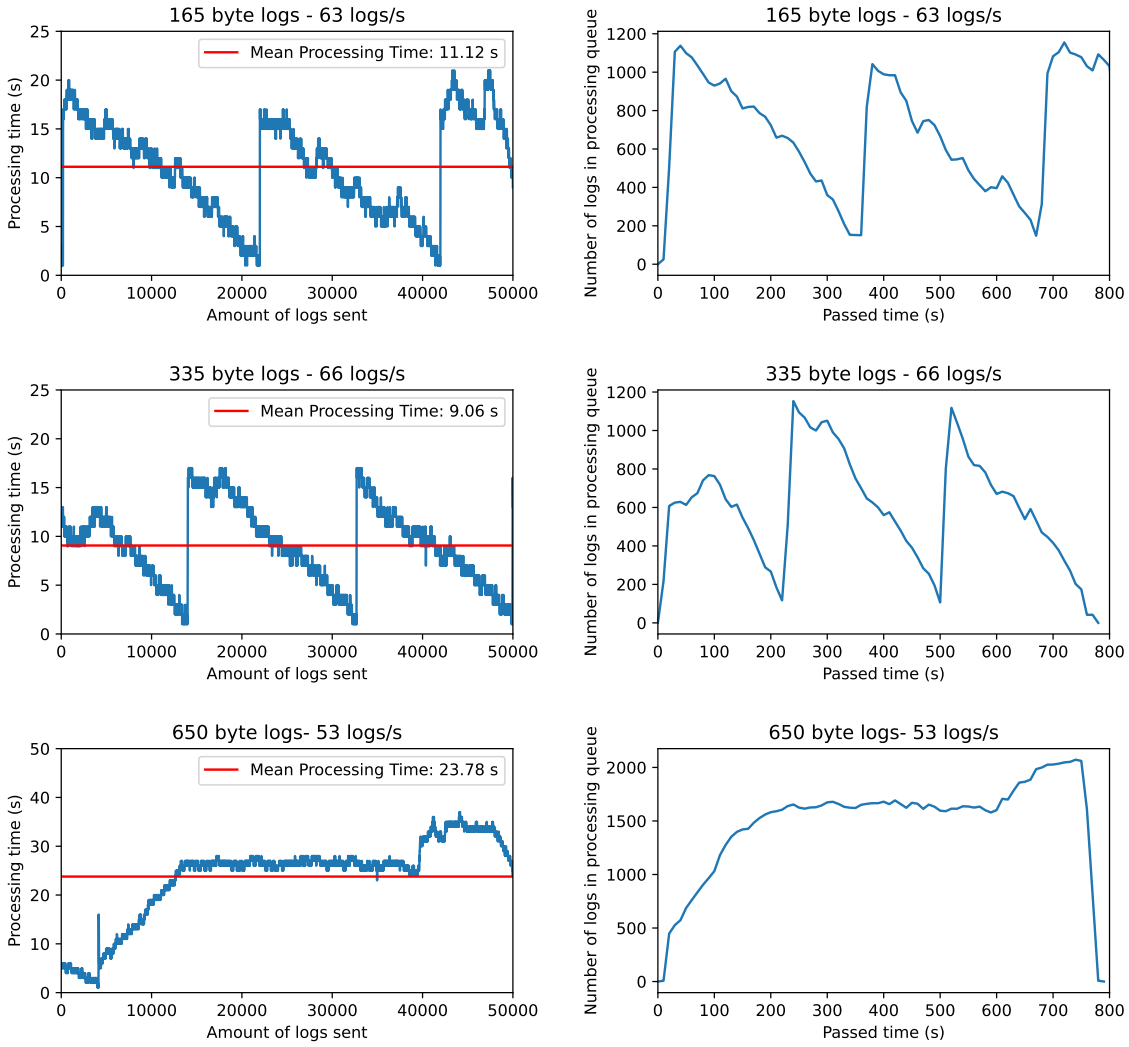

Figure 5.2.3: Sending frequency: 53-66 log events/s, Amount of logs sent: 50,000, Instances sending: 10, Batch size: 100 logs, Processing Interval: 15s

of unprocessed logs, the correlation is more clearly seen in this block compared with previous tests. The pattern is largely the same when comparing column 1 with column 2, again suggesting that processing time correlates with the amount of unprocessed logs.

Moving on, a great difference and a complete change of behavior is observed in the bottom row of fig. 5.2.3. As can be seen, the mean processing time is substantially higher than for the 165- and 335-byte logs. Comparing the 335 and 650 byte logs, the average processing time increases roughly 161% (9.1s to 23.8s), even though the logging frequency is a bit lower for the logs of the higher size. This suggests that the log sizes affect the SUTs processing efficiency quite substantially, even though this behavior has not been observed on this scale until now. It can also be seen that the processing time continues to increase with the number of logs sent. They no longer oscillate around a

mean value but rather continuously increase over time. Looking at the graph showing the number of unprocessed logs, it can be seen that the queue starts to decrease at roughly the same point that the processing times stop increasing. Phrased differently, processing becomes faster only when the queue finally starts emptying. This is also the point where the log generator stopped logging into SUT. Essentially, this means that the processing times only started to decrease because no more logs arrived at the tool. What would happen if the logging continued? In the next section, this is further investigated. It should also be noted that the average processing time here is far above what is optimal in the context of DataOps.

5.2.2 Stress Test

In fig. 5.2.3, it was observed that processing times continuously increased when sending with a high frequency, 53 logs per second, for 650-byte logs. This behavior continued until the sending process stopped, upon which the processing times finally started to decrease. What would happen if the log generator continued to send logs for an extended time, under high frequencies? To investigate this, an extended stress test was performed. The test was carried out by sending 100,000 logs of 650 bytes size at three different sending frequencies. The result is shown in fig. 5.2.4. When sending with a frequency of 36 logs per second, the processing times fluctuate within a static interval that is not changed over time, resulting in an average processing time that is stable during the entire sending process. This is consistent with the previous corresponding tests. As seen, the frequency of 36 logs per second produced an average processing time of roughly 8.8s. This processing time is consistent with the corresponding test conducted in fig. 5.2.2, where only 25,000 logs were sent. Thus, under this load the processing times does not increase over time even if a higher amount of logs are sent in total. This suggests that the tool provides consistency and stability for logs and frequencies of this type. However, at some point, this behavior deteriorates. Looking at the frequency of 62 logs per second, it can be observed that the processing times increase over time, in accordance with previous corresponding tests. The average processing time is roughly 250s, and the median processing time is 312s. These values are extremely high and confirm the behavior indicated in the previous tests that processing times will continuously increase as long as they are being sent to SUT, after a certain threshold in frequency and log size has been exceeded. The difference between average and median processing times should also be noted. With a median value that is substantially higher than the average value, it can be concluded that of the 100,000 logs, most are on the higher side in terms of processing time. Looking at the graph of 62 logs per second, we can see this. The processing times increase the most for the first roughly 40,000 logs. During the previous tests, we could see a peak in the graph, after which the processing times starts to decrease. We see this in fig. 5.2.4 as well, but it is less prominent. Again looking at the graph of 62 logs per second, we see a peak (this is also the point where the logging is stopped). After this peak, the processing times decrease until all of the 100,000 logs have been processed. However, the decrease is not as severe as in the previous tests. This suggests that the

tool struggles with its processing efficiency even after logs have stopped being sent to it. This is hypothesized to be the result of an increased computational power being required to store and handle the large amount of data that is in the queue.

Looking at the highest logging frequency of 90 logs per second, a similar result but with higher numbers is presented. An observed difference is that the processing times never stop increasing. This happens even though at some point on the graph the log event generator stops sending logs. This means that the processing times continue to increase even after the log generator has stopped sending and the queue is being emptied. Lastly, an important topic also is whether this result (processing times) can be improved. This is investigated in the next section.

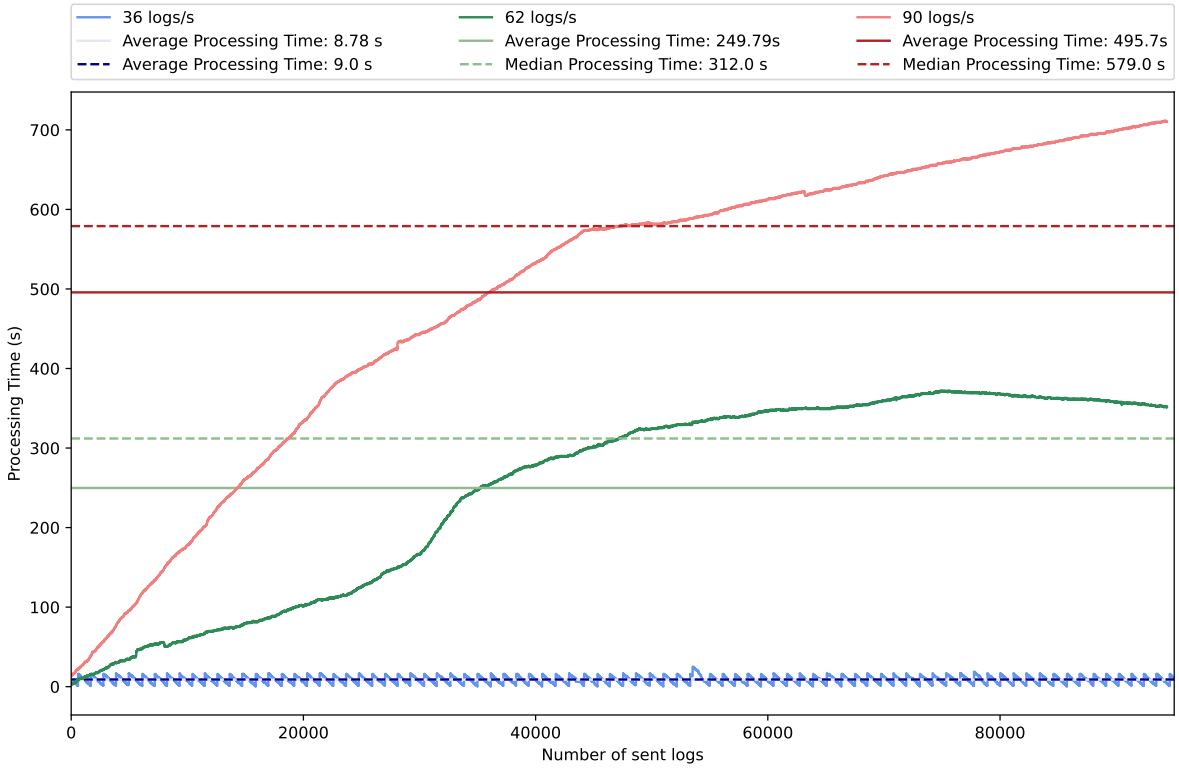

Figure 5.2.4: Instances sending: 5, 10 and 15, Batch size: 100 logs, Processing Interval: 15s

5.2.3 Improving Processing Times

Two independent testing variables have not yet been investigated: *MessageProcessingTimerInterval* and *MessageProcessingBatchSize*. In this section, these variables are tuned to determine if they can improve processing times. This is done by first varying *MessageProcessingBatchSize* while keeping *MessageProcessingTimerInterval* constant, and vice versa. In fig. 5.2.5, the results are shown. Looking at the top row, it can be observed that a larger batch size results in substantially higher processing time. By, for example, comparing a batch size of 100 to 1000, the time difference is roughly 123%. This indicates that a lower batch size

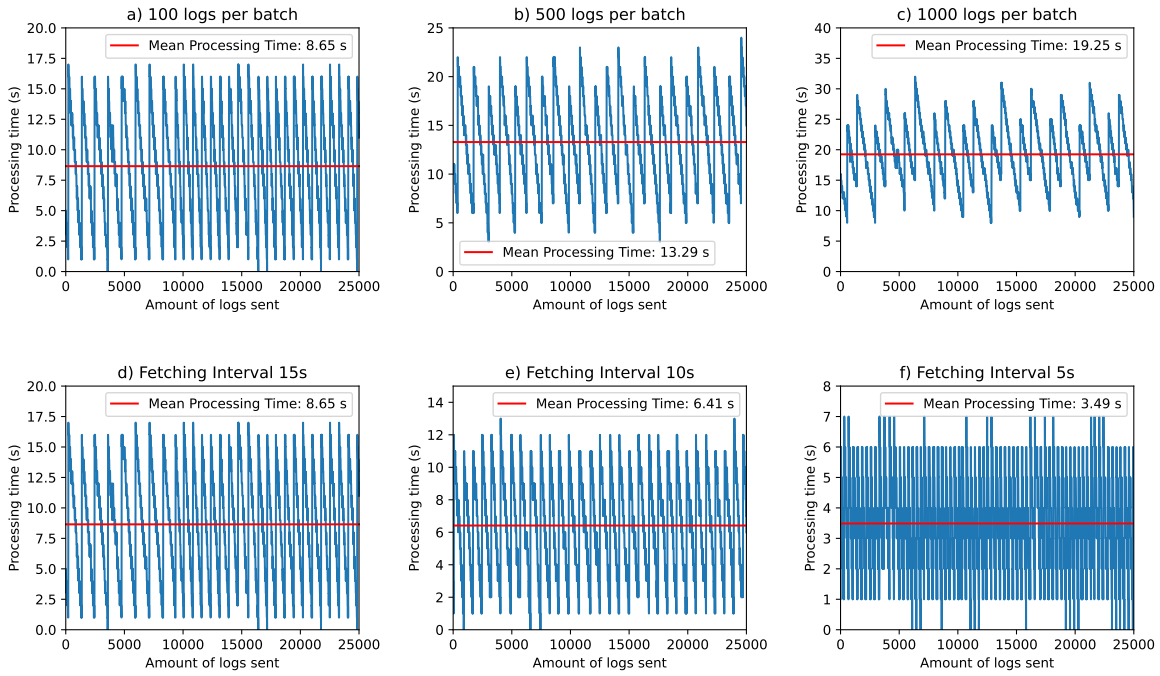

Figure 5.2.5: Sending Frequency: 35 logs/s, Amount of logs sent: 25,000, Instances sending: 5

is more efficient in fetching and processing logs. Continuing, looking at the bottom row, a substantial improvement can be observed when the standard fetching interval of 15s is changed. Comparing an interval of 15s to 5s (no lower frequency is allowed in the SUT), the processing time is roughly 148% higher for the 15s interval – 3.49s vs. 8.65s. This aligns well with what has been discussed earlier, that a liming factor in the internal structure and architecture of the tool is the way that logs are fetched in intervals. The design of fetching logs for processing in intervals rather than directly can result in a major bottleneck for SUT in terms of timeliness. Continuing, as decreasing this variable results in a faster processing time, a question that arises is if this can improve the problem of continuously increasing processing times for higher sending frequencies. In fig. 5.2.6, the tests of fig. 5.2.4 were re-conducted with a processing interval of 5s instead of a processing interval of 15s. A clear improvement can be observed; however, the characteristic of increasingly faster processing times is still present. It can also be seen that the highest sending frequency still results in processing times that continue to increase even after log-events stop being fed to the SUT. Thus, a faster fetching frequency does not solve the problem of high and continuously increasing processing times – the bottleneck (fetching interval) persists.

5.3 Evaluation

This section will evaluate the test results and discuss the SUTs ability to collect and process data in a timely manner. Subsequently, the tool is evaluated as a DataOps

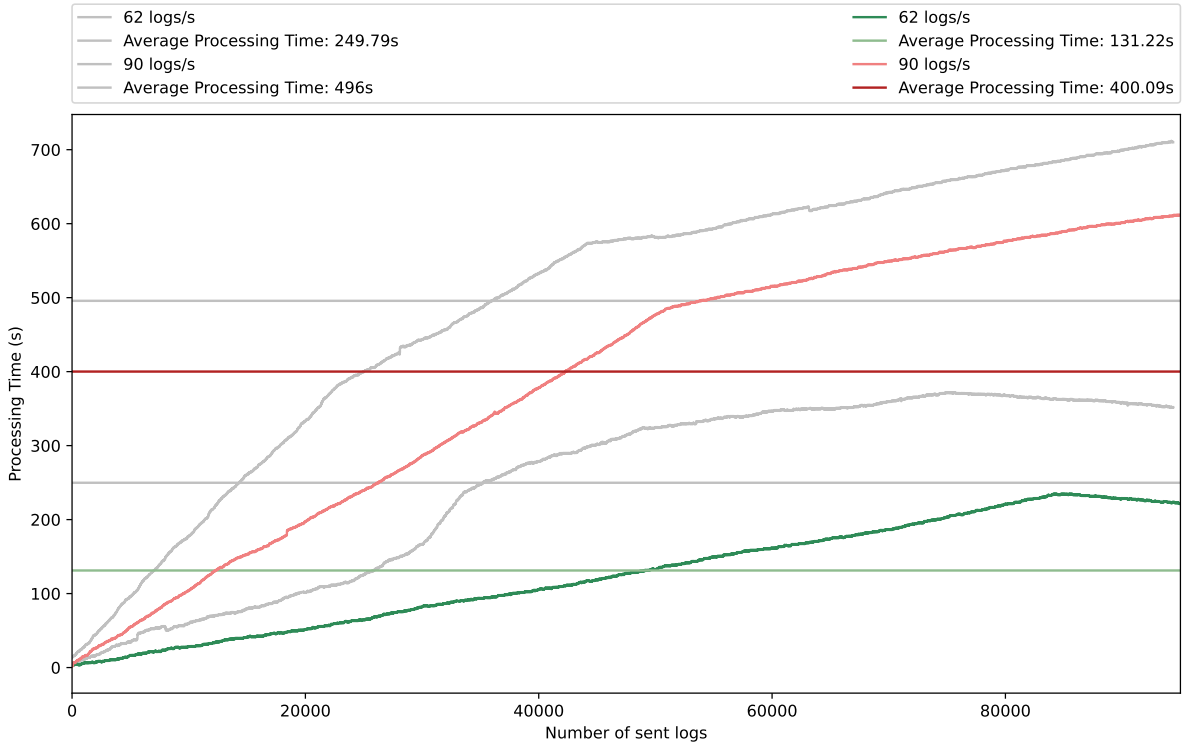

Figure 5.2.6: Instances sending: 5, 10 and 15, Batch size: 100 logs, Processing Interval: 5s

solution for DTs, based on the results collected and the information derived from the current literature in Chapter 2. Lastly, limitations of the testing process and the thesis in general are presented.

5.3.1 Evaluation of Testing Results

Starting with the results presented in fig. 5.2.1, we concluded that the average processing times ranged from 7.68s for the 165-byte logs, to 7.86s for the 650-byte logs – when the logging frequency was 7 logs per second. This is a 2.3% increase in processing time and an increase of 294% in log size. The same behavior was observed in the next test, shown in fig. 5.2.2. Here, the average processing times ranged from 8.55s to 8.81s for a logging frequency of 35 logs per second. This is a 3% increase in processing time, again for an increase of 294% in log size. If we compare the results from fig. 5.2.1 with fig. 5.2.2, the processing times are increased from 7.68-7.86s to 8.55-8.81s while the logging frequency is increased from 7 to 35 logs per second. This means that even though the logging frequency was increased with 400%, the processing times only increased roughly 11-12%. These first initial results, where the processing efficiency is only slightly decreased, even though both log sizes and logging frequency are increased on a much higher scale, suggest that the tool can handle both logging of larger sized logs and an increase in logging frequency quite well. Increasing the logging frequency is especially interesting, as one of the qualities that

were important for DataOps and DTs is scalability. When a tool logs data from more devices and data sources, the logging frequency should also increase, and it must be able to handle this. However, this quality and hitherto observed behavior would soon deteriorate as the load was increased. Continuing, still looking at the first two initial results, we also observed that the processing times on an individual level fluctuates around and deviates from the average processing times quite substantially in a regular pattern. This is concluded to be a result of how the SUT fetches its logs in regular intervals, more specifically 15s in these tests (which is the standard value of the tool). If logs are only fetched at regular intervals from a queue, some logs will inherently have a higher processing time than others, and vice versa. Let's illustrate this with an example: A batch of logs have just been gathered for processing by the tool. Immediately after this fetch is made, another log enters the queue. If the processing interval is 15s, this log will now have to wait roughly 15s before being fetched for processing. When 14s have passed, another log enters the queue. This log will only have to spend 1s in an idling state before being fetched for processing. If we look at the graphs, we see that the highest processing times only slightly exceed 15s. This suggests that as soon as the logs are fetched for processing, the processing itself is done very fast. Even if a log, as in the example above, has to spend 15s waiting for the next batch, its processing time will only slightly exceed 15s. A conclusion here is that a majority of the processing time is spent in an idling, non-productive state. This presents a major bottleneck in the ability of the SUT to process data in a timely manner and will severely limit its timely capabilities. Additionally, looking at the processing times as absolute values, it can be argued that they are very high even at sending frequencies such as these. With an average of 7.68-8.81s, the tool would be very limited for use in DataOps – where timeliness is crucial and the tool must be seamlessly integrated into the systems where it is applied. Instead, looking at the tool explicitly for use in DTs, there are still a lot of application areas where processing times as high as these would be too slow. For example, in time-critical production lines in manufacturing. However, in applications where time is not as critical, these processing times could be acceptable, especially considering that they do not increase over time (for these particular results). When using DTs strictly for simulation and testing purposes rather than applying them directly to a physical entity, it could also be argued that time is not as critical. However, looking at the tool from the perspective of DataOps for DTs, the initial tests indicate that the tool is not a viable option. As we have concluded, a DataOps solution must seamlessly integrate into the system where it is deployed. Adding processing delays in the range of seconds would not do this.

Continuing, a behavioral change was observed in fig. 5.2.3 where the logging frequency increased to 52 and 63/66 logs per second. For logs of 165 and 335 bytes, an increase in the average processing time was identified. However not substantial, there is still a behavioral change in how the SUT processes the data. Not as many maxima and minima are observed, and the previously discussed fluctuation around an average value is not as prominent and frequent. The main focus is not on the behavior itself, but on its effect, which is not changed in a critical matter. The average processing times

are still, at large, about the same as in previous results. However, a critical change can be identified for the 650-byte logs, where a substantial increase in the processing times, as well as a change of behavior, was observed. It is also seen that processing times continue to increase with time. Only when the log generator stops sending logs and the processing queue is offloaded is a decrease in processing time observed. This is not a good sign, as it suggests that the SUT would become increasingly ineffective in processing data as long as data are logged. This was further investigated in the following stress test. As observed in this test, fig. 5.2.4, high sending frequencies continued to produce logs with increasingly longer processing times. For the highest frequency of 90 logs per second, it could also be observed that the values continued to increase even after the log event generator had stopped sending logs. This means that even though the processing queue did not continue to fill up, the processing times still increased – until all of them were processed. For a sending frequency of 62 logs per second, an average processing time of 250 was reached when sending 100,000 logs. For a frequency of 90 logs per second, this number reached almost 500s. This is very high, and under these circumstances, it would be very difficult to apply the logging tool for collecting data to even moderately time sensitive systems. Looking at the lowest sending frequency of 36 logs per second, the previous behavior of no increase in average processing time is still present. The processing time per log is roughly the same as in fig. 5.2.2, even though 100,000 logs were sent.

Following this, it was investigated whether processing times could be improved. As concluded in fig. 5.2.5, a lower value for *MessageProcessingTimerInterval* would reduce processing times. For 335-byte logs sent with a frequency of 35 logs per second, the average processing time could be lowered from 8.65s to 3.49s when changing the fetching interval from 15s to 5s. This is substantially better and provides improved timeliness. Although this is an improved timeliness, as a DataOps solution, it is still too high. Even if the fetching interval is lowered to 5s, there would still be a ”5s bottleneck”. Processing data in the range of seconds is not optimal for a DataOps solution that should integrate seamlessly where it is implemented. However, as mentioned, for use specifically within DT, it could be a viable option for those twins that are not dependent on real-time data. Therefore, it was essential to test whether the very slow processing times presented earlier in the stress tests could be reduced to a more suitable level. Thus, the last test applied a fetching interval of 5s on the same variables used in fig. 5.2.4 to see if processing times could be reduced. As was concluded, the values were lowered but remained at a high level and followed the same behavior as for the 15-second interval tests. With such high values, the only actual application area would be those that do not depend on timely behavior at all. But, as argued earlier, a full-fledged DataOps solution for general use should not process in seconds, and this would be too slow.

To conclude this evaluation, the initial tests indicate that the SUT does provide a certain scalability. The logging frequency can be increased quite substantially without it loosing its overall processing efficiency – until reaching a certain threshold. However, the baseline for how efficiently the tool processes data to begin with is not optimal

in terms of DataOps. Even in the initial tests the SUT does not process the data in real-time. Further, as logging frequencies were increased, the tool started to lose its processing efficiency and the processing times increased even more. Although the logging frequencies could be considered very high, they highlight critical problems within the tool in terms of timeliness.

5.3.2 Logging tools as a DataOps solution for Digital Twins

It is established that a core functionality of a DataOps solution is the ability to standardize data. It is also established that it is important that the data used for DT be standardized, considering that the twin might operate on data originating from many different sources. As the tool can log data of different formats and store the information in a standardized format to a database, the characteristic of standardization is present in the tool. Furthermore, using the *Search Field* system in the SUT, specific data fields that have been sent to the tool through a *Payload* can be stored as standardized values. This means that one can also store specific fields from a log, rather than the whole log itself. Furthermore, monitoring was another important feature. In the SUT, a feature that can monitor specific data fields and send alerts if these fields reached a certain threshold is present. This can be considered a form of monitoring. However, it must be pointed out that this mechanic was not explicitly tested.

It is concluded that a DataOps solution should be scalable and be applicable for connecting to different types of IT systems and devices. This is also an essential aspect for DTs. As has been discussed, the tool offers the functionality to connect and record data from different types of source, either using prebuilt log agents or building custom solutions that logs data. Although this indicates that the tool is scalable, being scalable also infers that the tool must be able to handle an increased load (logging frequency). As the tests of this thesis have shown, the SUT provides a certain scalability. However, at some point when a threshold is exceeded, this quality deteriorates. When the logging frequencies (in combination with the log sizes) reach a certain point, the processing capabilities of the SUT are substantially affected and the processing capabilities become increasingly worse. Lastly, a crucial quality for DataOps is timeliness. This is also the quality which was focused during the tests. And, as we concluded, SUT has a major bottleneck in its architecture and internal structure that prevents it from processing data in real time. During the moderate loads in the tests, the average processing times ranged from roughly 3.5 to 11 s, depending on which parameters were used. This could be a viable timeliness for some DTs, but for a tool to be used as a full-fledged solution in DataOps it should process data much more efficiently than in the range of seconds. It should integrate seamlessly into the system where it is applied. Thus, an argument can be made that the SUT used in this thesis, Nodinite, is not a viable option for being used as a DataOps solution. However, it could be a viable tool for gathering data to some DTs, more specifically those twins that does not depend on timely data.

Generalizing the findings of the study and the statements above, it can be concluded

that logging tools of this type have some features which coincide well with the characteristics of DataOps solutions and the data requirements of DTs. However, as we have shown for this specific tool, there are strong limitations in its ability to process data in a timely manner. This affects its suitability for being used as a DataOps solution negatively. Furthermore, it will limit its potential use for data collection to DTs, as most DT s depend on real-time data. It must also be mentioned that although the results of this study present processing times that can be regarded as high for the discussed areas of application, it does not necessarily mean that all similar logging tools will provide equal results. It could, however, show to be a recurring problem, as logging tools are not necessarily built with the intention of presenting logs in real time, but to rather show them in a presentable, clean, accurate, and well-processed way. This is not a final conclusion, however, and is a statement which would require more research. In short, the result indicates that the tool could be used as a data gathering solution for certain non-time critical DTs, but is not a viable option for being used as a DataOps solution. At least in its current state.

5.3.3 The timeliness of the SUT

Having discussed the results, it is clear that the SUT in this study does not provide optimal timeliness. The presented average processing times range from 3.5s to more than 500s. Although the extreme logging frequencies used during the stress test might not be a realistic logging frequency, it can be seen already under moderate loads that processing times are in the range of roughly 3.5s to 11s on average (depending on different parameters). As has also been discussed, the main reason that the SUT is not very efficient in timely processing is an effect of how it obtains logs for processing. If, in theory, the processing interval present in the tool were to be eliminated and processing be done immediately as data is logged, processing times would be much faster. In conclusion, the tool processes data in the range of seconds and thus never in real-time. With an average in these tests reaching no lower than 3.5s. Not looking at the average, but rather on individual logs, the processing times oscillate around the average, reaching both lower and higher values than the average.

5.3.4 Proposed Improvements For The SUT

As has been discussed, the SUT is limited in how quickly it can process logs. The main reason for the high processing times is the way the tool fetches data for processing in intervals. If batches of logs are only fetched in certain time intervals for processing, the time between generating the data and it being completely processed will be greatly affected by this interval. For example, if the batch fetching interval is set to 15s and a log is sent to the tool directly after a batch has just been fetched, it will have a minimum processing time of roughly 15s – as this is the time it takes for the next batch to be fetched. In the test cases, we can see that the processing time rarely exceeds the processing interval time by a substantial amount. If the processing interval is 15s, the maximum processing times will also be around 15s. This means that between the

data being generated and the data being completely processed, most of the time will be spent waiting for the tool to fetch these data from the batch. When data is finally fetched, it is processed in a time that is likely much faster than the time it has spent in an idle state. This is a flaw in the system and a major bottleneck in looking at the tool from the perspective of timely processing. Improving this and possibly removing the fetching interval completely could improve processing efficiency.

5.4 Limitations of the Thesis

One limitation of this study is the lack of testing multiple logging tools. Using only one SUT requires generalizations to be made. Further studies should look at other logging tools as well to get a broader understanding of how they work and operate. Additionally, although some features present in the tool can be related to important qualities of DTs and DataOps, they have not been explicitly tested and verified. For example, it is described how Nodinite can connect to different IT systems for logging data and how it can monitor and send out alerts if some threshold is exceeded, but it is not explicitly tested in the thesis. However, as has been stated, it is still motivated to focus on timeliness, as that will be a main determining factor. Another limitation is that no investigation has been done as to how easily accessible the gathered data is. Although the processing times for the collection and processing of the data within the SUT are determined, it is not determined how efficiently DT could fetch these data. The lack of a real use case also affects the evaluation of the SUT as a DataOps solution, as this analysis is based on a general picture of such solutions. By conducting a study in a real use case, where the tool is used as DataOps in a real environment with real DT, the limitations and problems described above would be further evaluated and explored. One of the favorable effects of this would be that the tool could be evaluated for its contribution to the system where it is implemented, as a whole. As DataOps is often about making an entire environment more effective, for example, introducing data orchestration and pipelines, a real environment would be very beneficial. Another topic of improvement would be conducting tests on more complex logs. For example, by logging large XML files, instead of small JSON files. Lastly, a topic of interest is also how hardware will affect the results. Hardware will obviously affect computational capabilities, and this would certainly be a topic for further research. However, as we have concluded, a main limitation of the processing capabilities of the SUT in this thesis is due to software limitations rather than computational limits. The final result shown in this thesis should, therefore, not be affected by hardware to an influencing degree. Nonetheless, it is a limitation that hardware has not been more investigated and integrated into the tests.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In the literature study we found that a very important quality for DataOps in particular, but also DTs, is the ability to process data in real-time. Continuous improvement, automation, monitoring, data orchestration, and data pipelines have also been concluded to be important qualities for DataOps. Further investigating DataOps in Industry 4.0, another crucial component is the ability to standardize data. This is also of great importance for DTs, considering the environments in which they operate, where the data can originate from many different sources, requiring the data to be standardized to take advantage of it efficiently. Related to this component is the ability to connect to different kinds of device and to gather the data to begin with. This is also important for both DataOps and DTs. Lastly, both technologies should provide scalability. This means that a DataOps tool in DTs must be able to log data from many different devices simultaneously, all while not losing efficiency when an increasing amount of data is being logged.

It is concluded that the processing times vary depending on a set of parameters of the SUT and the logs. The variables that affect the processing times are; logging frequency, log sizes, the number of logs sent, how often the tool fetches logs from its queue and how many logs to fetch per batch. Depending on these parameters, average processing times can range from 3.5 to 500 seconds (for the conducted tests). The results also suggest that the latter could increase if the logging was continued, i.e. the number of logs sent was higher. There is a threshold in which, when exceeded, the processing times become increasingly higher. It cannot be concluded exactly when this threshold is exceeded, but the main parameters that affect it are the logging frequency and the log sizes. If this threshold is not exceeded, the processing times do not increase over time, but fluctuate within a fixed interval. This behavior is a result of how the tool fetches its data in intervals. It is also shown that a faster fetch interval will positively affect processing times, which means that the tool runs more efficiently when fetching data batches with a higher frequency. The fact that processing times continue to

increase after a certain threshold could be mitigated (but not eliminated) by lowering this fetching interval. For example, by having a fetching interval of 5 seconds instead of 15 seconds, the average processing time could be reduced from roughly 500 seconds to 400 seconds in one of the stress tests. Similarly, processing times could be reduced from roughly 9 seconds to 3.5 seconds for moderate logging frequencies. For low-to-moderate logging frequencies, the average processing times are consistent, while individual processing times fluctuate within a fixed interval, but never real-time. Thus, the timeliness of the tool evaluated is in the range of seconds, where average processing times are never lower than roughly 3.5 seconds for the conducted test cases. It is concluded that the reason the tool cannot process data faster is because of the way it fetches data in batches. If data are not fetched and processed immediately as it reaches the tool, there will always be a bottleneck or a lower limit as to how fast data can be processed. For example, data that reaches the tool right after a batch of logs is fetched will have to wait in a queue until the next batch. Thus, the timeliness of the tool largely depends on how regularly the tool can fetch the batches – creating a bottleneck. In conclusion, the tool processes data in the range of seconds and never in real-time.

We can conclude that the tool has some characteristics that are well related to the characteristics found in DataOps and the requirements of the data in DT. For example, the tool enables data to be logged from any arbitrary data source, standardizing, and storing them to a database, important qualities for both DataOps and DTs. It is also possible to store specific parameters in the log files, these data being standardized as well. As custom solutions for logging can be built, the tool also allows for the recording of artificial data. That is, data that do not originate from physical data sources. This is important for DTs for analytical and predictive purposes. Although not explicitly tested, the tool also has features for monitoring log parameters. However, even though some features fulfill important qualities in DataOps for DTs, we have shown that the tool does not process the data in real time. The result strongly suggests that for use in DataOps, the processing times are not fast enough. Therefore, since timely processing is crucial for DataOps, the tool cannot be concluded to be a viable solution DataOps for DTs. It could, however, be a viable option on its own for data gathering to certain DTs – that is, not as a DataOps solution in general. For those DT cases where it is not critical to have data processed immediately, but with consistency and where data are recorded with a moderate frequency, the tool could be a viable option. However, this would require more research.

6.2 Future Work

Considering the result, a crucial component in continuing this work is to investigate whether the timeliness of the SUT can be improved. As we have mentioned, timeliness is a bottleneck because logs are fetched in batches at regular intervals. If a solution could be found to fetch data immediately as it is sent to the tool, the processing times could be improved. Another prospect for future studies is to investigate whether similar tools have the same problem. As only one SUT has been subject to

tests and evaluation, generalizations have been made. Conducting more extensive research that includes a wide set of logging tools would provide a better overview of such tools. Additionally, a study conducted in a real environment would be very beneficial. For example, DataOps is something that is implemented with the intention of changing and improving the entire system where it is implemented. Applying a tool to a real environment would provide a better way to evaluate the tool in terms of data orchestration, pipelines, and other characteristics of DataOps. Additionally, implementing a real DT would allow testing of the ease with which the logged data are accessible. In this study, we have logged data, but we have not tested how accessible it is to use for a real DT. However, for the above to have a purpose, the timeliness must first be improved or another tool found.

Bibliography

[1] Abdelmajied, FathyElsayed Youssef. “Industry 4.0 and its implications: Concept, opportunities, and future directions”. In: (2022).

[2] Connolly, Christine. “A review of data logging systems, software and applications”. In: *Sensor Review* (2010).

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

[4] Ereth, Julian. “DataOps-Towards a Definition.” In: *LWDA* 2191 (2018), pp. 104–112.

[5] Harrington, John. “DataOps: Fundamental for Industrial Transformation.” In: *InTech* 68.1 (2021), pp. 28–32. ISSN: 0192303X.

[6] Hu, Weifei, Zhang, Tongzhou, Deng, Xiaoyu, Liu, Zhenyu, and Tan, Jianrong. “Digital twin: A state-of-the-art review of its enabling technologies, applications and challenges”. In: *Journal of Intelligent Manufacturing and Special Equipment* (2021).

[7] Jiang, Zhen Ming and Hassan, Ahmed E. “A Survey on Load Testing of Large-Scale Software Systems”. In: *IEEE Transactions on Software Engineering* 41.11 (2015), pp. 1091–1118. DOI: 10.1109/TSE.2015.2445340.

[8] Mainali, Kiran, Ehrlinger, Lisa, Matskin, Mihhail, and Himmelbauer, Johannes. “Discovering DataOps: a comprehensive review of definitions, use cases, and tools”. In: *DATA ANALYTICS 2021 The Tenth International Conference on Data Analytics*. 2021.

[9] Mihai, Stefan, Yaqoob, Mahnoor, Hung, Dang V., Davis, William, Towakel, Praveer, Raza, Mohsin, Karamanoglu, Mehmet, Barn, Balbir, Shetve, Dattaprasad, Prasad, Raja V., Venkataraman, Hrishikesh, Trestian, Ramona, and Nguyen, Huan X. “Digital Twins: A Survey on Enabling Technologies, Challenges, Trends and Future Prospects”. In: *IEEE Communications Surveys Tutorials* 24.4 (2022), pp. 2255–2291. DOI: 10.1109/COMST.2022.3208773.

[10] Munappy, Aiswarya Raj, Mattos, David Issa, Bosch, Jan, Olsson, Helena Holmström, and Dakkak, Anas. “From ad-hoc data analytics to dataops”. In: *Proceedings of the International Conference on Software and System Processes*. 2020, pp. 165–174.

[11] Olsen, Tava Lennon and Tomlin, Brian. “Industry 4.0: Opportunities and challenges for operations management”. In: *Manufacturing & Service Operations Management* 22.1 (2020), pp. 113–122.

[12] Pires, Flávia, Cachada, Ana, Barbosa, José, Moreira, António Paulo, and Leitão, Paulo. “Digital twin in industry 4.0: Technologies, applications and challenges”. In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. Vol. 1. IEEE. 2019, pp. 721–726.

[13] Rasheed, Adil, San, Omer, and Kvamsdal, Trond. “Digital twin: Values, challenges and enablers from a modeling perspective”. In: *Ieee Access* 8 (2020), pp. 21980–22012.

[14] Seddon, Peter B, Constantinidis, Dora, Tamm, Toomas, and Dod, Harjot. “How does business analytics contribute to business value?” In: *Information Systems Journal* 27.3 (2017), pp. 237–269.

[15] Uhlenkamp, Jan-Frederik, Hribernik, Karl, Wellsandt, Stefan, and Thoben, Klaus-Dieter. “Digital Twin Applications: A first systemization of their dimensions”. In: *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE. 2019, pp. 1–8.

[16] Weyuker, E.J. and Vokolos, F.I. “Experience with performance testing of software systems: issues, an approach, and case study”. In: *IEEE Transactions on Software Engineering* 26.12 (2000), pp. 1147–1156. DOI: 10.1109/32.888628.

[17] Xu, Li Da, Xu, Eric L, and Li, Ling. “Industry 4.0: state of the art and future trends”. In: *International journal of production research* 56.8 (2018), pp. 2941–2962.

[18] Zhang, Meng, Tao, Fei, Huang, Biqing, Liu, Ang, Wang, Lihui, Anwer, Nabil, and Nee, AYC. “Digital twin data: methods and key technologies”. In: *Digital Twin* 1.2 (2022), p. 2.