UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Usage Statistics

João Marcelo Machado Milagaia

Mestrado em Informática

Trabalho de Projeto orientado por:
Prof. Márcia Cristina Afonso Barros

2022

Acknowledgments

I would like to start by thanking the Faculty of Sciences of the University of Lisbon for giving me the opportunity to pursue learning Informatics and develop a Master's Project Work in Informatics. This experience enabled my growth as a Software Developer and furthered my interest in Informatics. I would also like to give a special thanks to Prof. Márcia Cristina Afonso Barros, without whose support, patience and guidance this project would not be possible. I would also like to thank Quidgest for the opportunity to have this project experience within a professional context and Rodrigo Serafim, Chief Technology Officer (CTO) at Quidgest, for guiding me as well as the Research and Development (RND) team for welcoming and helping me throughout the project. I am also grateful for all the friends who supported me throughout this project, specially Catarina Parente and Francisco Cavaco. Finally, I would like to thank my sisters for their love and support and my parents for always believing in me, for all their hard work and effort so that I could continue studying and for always pushing me to do better. I am who I am because of them and I will forever be grateful.

À minha família.

Resumo

À medida que a tecnologia avança, os sistemas de informação podem satisfazer mais necessidades da humanidade e ajudar a resolver problemas cada vez mais complexos, resultando em sistemas cada vez mais complexos. No entanto, para manter e melhorar esses sistemas, é necessário monitorá-los de forma automática. É aqui que entra o conceito de Observabilidade. Observabilidade, a capacidade de um sistema ser monitorável é definida como a capacidade de inferir os estados internos de um sistema examinando o seu output. Para implementar a observabilidade num sistema de informação, existem três tipos de dados de telemetria - logs, métricas e traces. Em suma, olhando para logs, métricas e/ou traces, é possível inferir o que está a acontecer dentro de um sistema para, por exemplo, detetar problemas ou ineficiências. Além disso, logs, métricas e traces podem ser resumidos e representados graficamente usando um dashboard, o que permite obter rapidamente informações sobre várias métricas e facilita a visualização de dados, economizando tempo e esforço.

A Quidgest é uma empresa tecnológica global e, desde a sua fundação em 1988, é pioneira no uso de IA aplicada à modelagem e geração automática de software. Genio, a plataforma para geração automática de código, é desenvolvida pela Quidgest desde 1990, sendo um dos seus principais recursos. É uma plataforma extreme low-code de desenvolvimento de software orientada a modelos. O Genio foi concebido para não programadores e permite que especialistas e analistas funcionais construam e suportem sistemas de informação, abrindo caminho para combinar o know-how tecnológico com o conhecimento do negócio. O Genio gera modelos que descrevem completamente o sistema final e são enviados para geradores de tecnologia específicos (por exemplo, Web Site em Asp.Net MVC) que cumprem a implementação desse modelo numa aplicação. Isto reduz o número de passos manuais propensos a erros e evita testes demorados, melhorando a produtividade em cerca de 10 vezes em comparação com plataformas de low code.

A Quidgest precisa de avaliar o desempenho dos sistemas gerados pelo Genio. Além disso, eles também precisam avaliar quais funcionalidades ou menus são usados. Até agora, esse tipo de informação era inferido por queries manuais à base de dados. Este é um processo difícil e ineficiente, razão pela qual uma junção de todas estas informações seria útil na hora de tomar a decisão de alterar ou remover funcionalidades. Os dashboards seriam especialmente úteis nesta situação para fazer debug, manter e analisar sistemas.

Não só isso, mas as estatísticas de uso podem fornecer uma visão sobre quais funcionalidades são mais usadas, o que é especialmente útil no processo de tomada de decisão de modificação ou remoção de determinadas funcionalidades. O objetivo deste trabalho é integrar um dashboard com estatísticas de uso e desempenho na interface de administração dos sistemas gerados pelo Genio. Para isso, um agente de coleção de métricas será desenvolvido e integrado a uma base de dados para persistir os dados. Além disso, será desenvolvido um dashboard, com capacidades de downsampling.

Sendo que o objetivo final deste projeto é a coleção, armazenamento e visualização de métricas de qualquer sistema gerado pelo Genio, um sistema específico gerado pelo Genio é usado para gerar métricas simuladas para fins de prototipagem. Esse sistema em particular é o QuidServer, um serviço windows desenvolvido e utilizado pela Quidgest que lê e configura processos de longa duração e chama web services externos que executam esses processos. Aqui, o QuidServer atua tanto como scheduler quanto como message broker e um MockWebApi envia mensagens para o QuidServer, o que o leva a fazer log de eventos. O QuidServer faz log de eventos pois tem um event provider integrado, que chama a API de logging. O tempo de processamento de mensagens foi desenvolvido como métrica pela empresa. Cada canal de mensagem foi atribuído a uma instância de Stopwatch, uma instância de EventCounter e uma instância de EventCounter geral partilhada por todos os canais de mensagem. O stopwatch é iniciado antes do processamento de cada mensagem e após o processamento da mensagem, o stopwatch é interrompido e o tempo decorrido é gravado em ambos os EventCounters e ao final de cada janela de tempo (definida para 5 segundos) os EventCounters fazem log de eventos ETW contendo o número de mensagens processadas e um resumo dos tempos de processamento (tempo máximo, médio, mínimo e stdev).

Posto isto, foi desenvolvido um agente de coleção de eventos em C Sharp utilizando o Microsoft Visual Studio para coletar os eventos. Este agente usa a biblioteca TraceEvent e é composto por um event controller e um event consumer. O controller cria uma sessão de rastreamento de eventos e permite que o event provider inicie o logging, e o consumer lê os eventos em tempo real, converte-os em data points e grava-os num bucket do InfluxDB usando a sua writing API.

Tendo o agente de coleção a enviar dados para o InfluxDB, foi criado um dashboard utilizando a ferramenta Boards do InfluxDB. As queries construídas no Data Explorer podem ser exportadas para diferentes células de um dashboard com a ferramenta Boards. Isto permite a visualização simultânea de diferentes dados e para melhor interação com as células do painel, é possível criar variáveis dinâmicas que permitem alterar componentes específicos da célula, como um valor de tag, sem editá-los. Outra funcionalidade do InfluxDB é a ferramenta Tasks que permite criar queries contínuas que são executadas automaticamente e periodicamente, o que é especialmente útil para downsampling, pois é possível executar periodicamente uma query que agrega dados em janelas de tempo e

armazena os valores agregados num bucket com um período de retenção maior.

Como o InfluxDB é principalmente uma base de dados, este não é otimizado para a visualização de dados, pois é mais limitante do que o Grafana, que é construído especificamente para a análise e visualização de dados temporais. No entanto, ao explorar os dashboards do InfluxDB, foi possível utilizar esse conhecimento e utilizar o Grafana para recriar o dashboard criado no InfluxDB. Os dashboards do Grafana têm painéis que possuem três funcionalidades diferentes: query, transformação e alerta. Isto permite remover a parte de menor desempenho de uma query, o processamento dos dados. Em vez disso, a query pode simplesmente buscar os dados necessários (o que torna a consulta muito mais rápida e eficiente) e o recurso de transformação do Grafana foi desenhado para processar com eficiência os dados consultados. Assim, o InfluxDB agora é usado apenas para armazenar e reduzir a amostragem de dados, enquanto o Grafana usa a linguagem Flux para fazer queries e visualizar os dados num dashboard interativo.

Até agora, a coleção, persistência e visualização são testadas apenas para o tempo de processamento das mensagens recebidas, métrica que foi implementada pela empresa. A próxima etapa deste projeto é então entender como as métricas são coletadas de um sistema: fazendo medições e agregando os dados em métricas que podem ser enviadas por eventos ETW para serem capturadas pelo agente de coleção de métricas. Para isso, é implementada uma nova métrica, o número de tarefas de scheduling invocadas, bem como o tempo de processamento dessas tarefas. A implementação desta métrica foi muito semelhante à métrica do tempo de processamento da mensagem, pois continuamos a medir um tempo de processamento. A única diferença é que, como as tarefas podem ser concorrentes, cada uma deve ter um novo Stopwatch distinto, em vez de cada canal ter um Stopwatch atribuído. Isto garante que, mesmo que duas tarefas diferentes do mesmo canal estejam a ser processadas ao mesmo tempo, cada uma delas ainda terá seu próprio stopwatch. Por fim, o Docker é usado para correr o InfluxDB num container e o Grafana noutro, o que permite a automação da instalação e configuração do InfluxDB e do Grafana usando ficheiros de configuração. Após isso, é desenvolvido um script para automatizar a primeira configuração do InfluxDB e do Grafana, facilitando a continuação deste projeto. Trabalho futuro incluiria a adição de mais métricas, como erros frequentes, duração de pedidos, duração de sessões e número de utilizadores por dia, além de integrar o dashboard criado no Genio com o objetivo de que cada sistema gerado por ele tenha o seu próprio dashboard estatístico.

Palavras-chave: TSDB, InfluxDB, Dashboard, Grafana, Docker

Abstract

By looking at logs, metrics and traces, it is possible to infer what is going on inside a system, in order to detect problems or inefficiencies. Quidgest is a global technological company and since its establishment in 1988, it has pioneered the use of AI applied to modelling and automatic generation of software. Genio is a tool that allows functional specialists and analysts to build and support information systems. Quidgest needs to evaluate the performance of the systems generated by Genio. The goal of this work is to integrate a dashboard with usage and performance statistics into the administration interface of the solutions generated by Genio. To generate mock metrics QuidServer is used, a windows service developed and used by Quidgest. This service reads and configures long duration processes and calls external services that run those processes. With this, an event collection agent was developed in C Sharp. This agent reads events in real time, parses them and writes them into an InfluxDB bucket. With InfluxDB, it is possible to create continuous queries that run automatically and periodically to downsample the data as needed. Grafana is then used to create dashboard, which allows for simultaneous visualization of different data. Having the processing time of received messages as a metric implemented by the company, a new metric is implemented, the number of invoked scheduling tasks, as well as the processing time of said tasks. This is done to understand how metrics are collected from a system: measuring data and aggregating it into metrics that can be sent through ETW events to be captured by the metric collection agent. Finally, Docker is used to run InfluxDB in one container and Grafana in another, which allows for the automation of the installation and configuration of InfluxDB and Grafana.

Keywords: TSDB, InfluxDB, Dashboard, Grafana, Docker

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This Master's Project Work in Informatics is the result of an internship of nine months at Quidgest - Management Consultants, SA. In this project, a dashboard of usage statistics is developed for Quidgest's software development platform, Genio. The goal of this work is to integrate a dashboard with usage and performance statistics into the administration interface of the systems generated by Genio. To do this, a metric collection agent will be developed and integrated with a timeseries database to persist the data. In addition to this, a dashboard will be developed, along with downsampling capabilities.

1.1 Context

Quidgest is a global software engineering company that has pioneered the concept of AI applied to automatic software generation since its establishment in 1988[18]. Genio is an extreme low-code model-driven software development platform developed by Quidgest, being one of its main resources. Genio is conceived for non-programmers, and it allows functional specialists and analysts to build and support information systems. It generates models that fully describe the final system. These models are sent to specific technology generators (e.g., Web Site in Asp.Net MVC) that fulfil the implementation of that model into an application. This reduces the number of manual error-prone steps and avoids time-consuming testing, improving productivity[6].

1.2 Motivation

As technology advances, information systems can satisfy more of humanity's needs and help solve increasingly complex problems, resulting in increasingly complex systems. However, to maintain and improve these systems, there is a need to monitor them in an automatic way. This is where the concept of Observability comes in. Observability, the ability of a system to be monitorable is defined as the ability to infer the internal states of a system by examining its external outputs. To implement observability in an information

system there are three types of telemetry data - logs, metrics, and traces. In short, by looking at logs, metrics and/or traces, it is possible to infer what is going on inside a system, in order to, for example, detect problems or inefficiencies. In addition, logs, metrics, and traces can be summarized and represented graphically by using a dashboard. This allows obtaining information on multiple metrics at a glance and makes data visualization easier, saving time and effort.

Quidgest needs to evaluate the performance of the systems generated by Genio. Not only this, but they also need to assess what functionalities or menus are used. Until now, this type of information has been inferred by manual queries to the database. This is a difficult and inefficient process, which is why a junction of all this information would be useful when making a decision to change features or remove them. Dashboards would be especially useful in this situation to debug, maintain and analyse systems. Not only that, but usage statistics can give an insight into what features are used the most, which is especially useful in the decision-making process of the modification or removal of certain functionalities.

1.3 Goals

The goal of this work is to integrate a dashboard with usage and performance statistics into the administration interface of the systems generated by Genio. To do this, a metric collection agent will be developed and integrated with a timeseries database to persist the data. In addition to this, a dashboard will be developed, along with downsampling capabilities. Furthermore, this project aims to explore metrics to monitor and test collection, persistence, and visualization tools for metrics.

1.4 Contributions

The contributions made for this project include:

- Metric collection agent prototype that supports integration with InfluxDB's TSDB
- Downsampling query for the collected metrics written in Flux language
- Grafana dashboard that displays the collected data
- New metric to be collected: Number of invoked scheduling tasks
- Docker containers for InfluxDB and Grafana, including all necessary configuration files
- Scripts to start and stop metric collection agent and containers

- Script to create data buckets for InfluxDB
- Script to import downsampling task into InfluxDB
- Documentation file that explains how to run metric collection agent, as well as how it works

1.5 Structure of the document

This document is organised in 6 chapters as follows:

- Chapter 1 - Introduction: Introduces the context for the project, as well as the motivation and goals behind it, followed by the contributions of the project and the structure of the document.
- Chapter 2 - Background: Provides more information about the company where the project work took place, as well as concepts needed to understand metric collection, persistence, and visualization, in addition to the tools that were used.
- Chapter 3 - Related Work: Discusses other works of research in the literature that are related to the various aspects of the project.
- Chapter 4 - Methods: Indicates what methods were used throughout the project.
- Chapter 5 - Results and Discussion: Exhibits and discusses the results obtained by following the methods previously mentioned.
- Chapter 6 - Conclusion: Summarizes the main points of the project and discusses possible future work for this project.

Chapter 2

Background

In this chapter, a background to the project is presented, including more information about the company where the project took place, as well as concepts needed to understand metric collection, persistence, and visualization, in addition to the tools that were used.

2.1 Quidgest

Quidgest is a global technological company and since its establishment in 1988, it has pioneered the use of AI applied to modelling and automatic generation of software[18]. Genio, the platform for automatic generation of code, has been developed by Quidgest since 1990, being one of its main resources. It is an extreme low-code model-driven software development platform that combines technological independence, code standardization and delivery speed in multiple languages and technologies. In fact, it supports the most recent web-based architectures, technologies, and databases, as well as previous modelled Back Office solutions to ensure legacy support and fast upgrading. Genio is conceived for non-programmers, and it allows functional specialists and analysts to build and support information systems, clearing the way for combining technological know-how with business knowledge. It generates models that fully describe the final system and are sent to specific technology generators (e.g., Web Site in Asp.Net MVC) that fulfil the implementation of that model into an application. This reduces the number of manual error-prone steps and avoids time-consuming testing, improving productivity by about 10 times in comparison with low code platforms[6].

Since one of the goals of the project is to collect, store and visualize metrics from any system generated by Genio, the first step is to test the collection, storage, and visualization of metrics from one particular system. Thus, the test subject used in this step is QuidServer, a windows service developed and used by Quidgest that reads and configures long duration processes and calls external web services that run those processes. Thus, QuidServer acts as both a scheduler and a message broker. A MockWebApi sends messages to QuidServer, running in parallel with it. This is what triggers QuidServer to log

events.

2.2 Logs, metrics, and traces

Monitoring the performance of the systems generated by Genio is crucial to debug common errors and maintain the system. It is also useful to know information like user logins and logouts to determine the affluence of people in a certain system at a certain time to predict a pattern and allocate more resources if and when necessary. In addition, the ability to know which features are most or least used in a system can be incredibly useful to understand the general user experience, acting as user's feedback. If a feature is the least used in a system it can be because it is not as visible or easy to use as the others, indicating to the developers the need of improving the feature, or it can be because the feature is not as useful for the users, indicating the possibility of removing that feature altogether.

In the same way, knowing which features are most used in a system may give clues about the next features to be implemented. These are just some examples of the benefits of system monitoring. However, the monitoring of a system is only possible if the system itself is monitorable.

The ability of a system to be monitorable is called observability. It is defined as the ability to infer the internal states of a system by examining its external outputs. To implement observability in a system there are three types of telemetry data - logs, metrics, and traces - often called "the three pillars of observability"[25]. Logs are immutable records that are time stamped, give contextual information about a certain event, and can be represented in three different formats: plain text, structured and binary. The information present in logs can include the time at which an event occurred and which user or endpoint was associated with it, as well as an error, warning or information message that describes what happened. Metrics are sets of numeric values that describe a process or activity over time and allow for longer data retention, since numbers are optimized for processing, storing, compressing, and retrieving. For example, metrics might track how many transactions an application handles per second or measure how many CPU or memory resources are consumed on a server. Traces represent a series of causally related distributed events which can give insight into both the path traversed by a request as well as the structure of a request. Traces are similar to logs in the sense that they describe at what time each process occurred, however traces are more structured since a trace of a request includes all the traces of the requests called by the original request.

The biggest challenge here is that the volume of collected data greatly increases over time, therefore for this project we will use three strategies for mitigating this problem: collecting metrics that, unlike logs or traces, can be conveniently aggregated and visualized in dashboards; storing the (timestamped) data in a time series database specialized in data compression; and downsampling (older) data, which metrics (as numeric values) are also

optimized for. Another way to solve this problem would be by sampling, as implemented in Google's Dapper project[34]. However, this approach is better fit for companies of the same magnitude as Google.

2.3 Metric monitoring and collection

To collect events logged by QuidServer, a collection agent is developed in C Sharp[2] using Microsoft Visual Studio[21]. This agent uses the TraceEvent library to collect and process event data[16]. This library was initially built to parse Event Tracing for Windows (ETW) events that the Windows operating system (OS) generates and is meant to be used together with the EventSource class that provides the ability to create ETW events[1].

ETW is a widely used and efficient logging framework with structured payloads and ETW based logging systems are broken down into three components: providers, controllers, and consumers. An event provider is wired into the application to be monitored (in this case QuidServer) and it calls the logging API to provide events. It is common for the controller and consumer to be combined, ergo in this project they constitute the agent developed in C sharp. The controller creates an event tracing session and enables the provider to start logging, and the consumer reads events from the session in real time.

EventSource is an efficient mechanism to log events, but it loses performance as the frequency of events increases. To manage that increase, the EventCounter class was used in the place of EventSource[4]. Instead of writing an event for every measurement, EventCounter records a set of values during a set interval and, at the end of each interval, a statistical summary for the set is computed to write an event. This way, instead of storing each individual processing time, the maximum, mean, and minimum processing time is stored, as well as the number of processed messages.

2.3.1 Telegraf

Telegraf is an open-source plugin-driven server agent for metric and event collection. The extensive library of input and output plugins make Telegraf easily extendable, and it also allows for custom plugin development[19]. The TICK stack is a loosely coupled and tightly integrated collection of open-source products developed to manage high quantities of time series data for metric analysis, and it consists of Telegraf, InfluxDB, Chronograf and Kapacitor[13]. By being part of the TICK stack, Telegraf is designed to work seamlessly with InfluxDB, having both input and output plugins available to collect data from and/or to InfluxDB[15].

2.4 Metric storing

Time series are sequences of values of a variable taken at successive equally spaced time intervals and since timestamped metrics are collected in this project, it is imperative to have a Time series database (TSDB) which is optimized for time series data[30] and efficiently compresses data[32], which is useful to control the volume of collected data since as data grows older, granularity becomes less important.

TSDBs have become increasingly popular with the emergence of the Internet of Things (IoT)[31]. IoT englobes a network of physical objects that are embedded with software, sensors, actuators, and other technologies that help to connect and exchange data using devices and systems over the Internet[35]. Since IoT sensors collect vast amounts of (timestamped) data, it makes sense that TSDBs would be imperative to IoT, however they are also used in DevOps monitoring and real time data analysis. TSDB use cases include, but are not limited to, software system monitoring, eventing apps that track data about user interaction and business intelligence tools.

The TSDB used in this project is InfluxDB, an open-source schemaless TSDB developed by InfluxData[14]. It is written in Go[7] and provides an SQL-like query language, Flux[5]. InfluxDB has been ranked the number one TSDB since 2016[10] on DB-Engines[3], an independent website that ranks DBs based on search engine popularity, frequency of technical discussions, number of job offers and number of mentions in social media.

InfluxDB stores its data in a bucket, which combines the concept of a database and a retention period, and it writes data points by using line protocol, a text-based format consisting of the measurement, tag set, field set and timestamp. Each point is required to have one and only one measurement, which is a string that indicates what the point is measuring. The tag set is optional, contains all the tag key-value pairs of the data point and both the key and value are strings. A point must have at least one field in the field set, which is required and contains all the field key-value pairs of the data point, in which the key is a string, and the value can be a string, float, integer or boolean. Finally, a point has only one timestamp, which is optional, since if it is not provided, InfluxDB will use the system time of the host machine.

Another key concept in InfluxDB is series. A series key is a set of points that share a measurement, tag set and field key. A series includes timestamps and field values for a given series key. An important thing to note here is that tags are indexed, and fields are not. This means that a query that filters by field values must scan all field values, making queries on tags more performant. Consequently, it is recommended that commonly queried metadata should be stored in tags and not fields. However, tags that contain highly variable information lead to a large number of unique series in the DB. This phenomenon is known as high series cardinality, and it is the main cause of high memory usage. To avoid this issue, highly variable information should be stored as a string field instead of a

tag.

2.5 Metric visualization interface

InfluxDB user interface (UI) allows data visualization; however, it is used for storing the data. The visualization is usually provided by another tool, Grafana[8], which is the go-to open-source software for time series analytics and visualization[31]. Grafana was built by Grafana Labs[9] and integrates a large number of data sources, including InfluxDB. In fact, the two are commonly used together.

On top of InfluxDB's features, Grafana's dashboards have panels that have three different functionalities: query, transform and alert. This allows the removal of the least performant part of a query, which is processing the data. Instead, the query can simply fetch the needed data (which makes the query much faster and performant) and Grafana's transform feature is designed to efficiently process the queried data.

2.6 Docker

Docker is a group of products developed by "Docker, Inc" in 2013 that uses OS-level virtualization to deliver software in individualized containers[11]. This open-source engine allows the wrapping of any application and its dependencies, allowing it to run anywhere, as well as automating the configuration of applications.

The base of docker are Docker files, which describe how to build a docker image, a snapshot of a piece of software. The image is immutable and is used to run one or more containers which are the actual software running.

Chapter 3

Related Work

Monitoring a system's performance is crucial to debug common errors and maintain the system as well as to gain insight on usage patterns. This section will discuss related work around three topics that are relevant to system monitoring: metric collection, metric storing and metric visualization.

3.1 Metric collection

There are two main categories regarding metric collection methods. One involves having an agent residing in each individual component that pushes metrics while the other consists of having a centralized collector that pulls metrics from components[33]. The TICK Stack[13] and Prometheus[17] are two popular monitoring systems that are examples of these two categories, respectively. In our project, a metric collection agent is developed to be compared with Telegraf, TICK's very own metric collection software, which has been shown to be a lightweight and suitable real-time monitoring agent[33].

However, the biggest challenge in system monitoring is that the volume of collected data greatly increases over time. One solution to this problem is sampling, as implemented in Google's Dapper project[34]. Dapper is described as a Large-Scale Distributed Systems Tracing Infrastructure. However, this approach is better fit for companies of the same magnitude as Google. Thus, other approaches are used, such as the aggregation of collected metrics, the use of a TSDB to store and compress data; and downsampling the data.

3.2 Metric storing

A relational database is a flexible, easy, and established solution for data persistence. Yet, it has the disadvantage of not being scalable as data increases[29]. This disadvantage leads to the creation and increasing use of time series NoSQL databases to deal with increasingly copious amounts of time-series data[27]. Indeed, TSDBs have been shown

to be optimized for time series data[30] and to efficiently compress data[32]. Not only that, but TSDBs have been increasingly used in association with the emergence of the Internet of Things (IoT)[31].

Our project uses InfluxDB, an open-source schemaless TSDB that has been the most used TSDB since 2016[10]. InfluxDB's primary benefit has been shown to be the ability of value aggregation without the need of manual interference[26] in addition to the ability of being accessed by Grafana, the go-to open-source software for time series analytics and visualization[31].

3.3 Metric visualization

Data visualization is a crucial part of any type of logging framework as it allows users to examine the data in detail and interactive statistic dashboards are especially useful for zooming in and/or filtering data. One example of interactive dashboards associated with a logging framework is the creation of a dashboard for the statistical analysis and visualization of user logs[23] from LogUI, a framework-agnostic JavaScript library[28] designed to simplify the process of logging for web pages.

In this project, the dashboard is constructed with the widely used Grafana which has recently been used as the display environment of the new monitoring infrastructure that CERN[12] has provided for the ATLAS Distributed Computing project[24]. This new infrastructure aims to replace the one that has been used for the last 10 years. It was based on a collection of CERN-provided custom dashboards and performed effectively, however the progressive difficulty of the system maintenance demanded for a new infrastructure[22].

Chapter 4

Methods

This section will describe not only the steps taken in the development of this project but also all the knowledge that was necessary to be acquired in the process, since in the beginning, the project consists of researching and learning new tools. The main stages of the project are an initial analysis of the problem and Genio's architecture, the development of a prototype for metric collection, persistence, and visualization, the implementation of a new metric and the use of Docker containerization, as seen in figure 4.1.

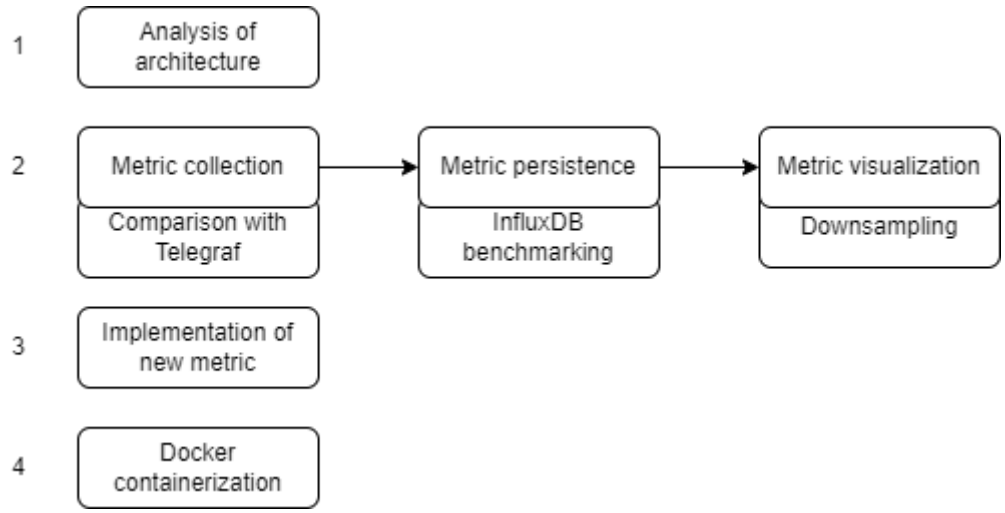

Figure 4.1: Project Flow Schema

Before starting the project, it is imperative to learn and understand how to use Genio and how it works as an extreme low-code model-driven software development platform to get a better understanding of it, as well as analysing the architecture of the systems generated by Genio and the structure of the databases created for them.

The ultimate goal of this project is the collection, storage and visualization of metrics from any system generated by Genio, but for prototyping purposes, one particular system generated by Genio, QuidServer, is used to generate mock metrics. After analysing the QuidServer's architecture, an event collection agent is developed in C Sharp using Mi-

crosoft Visual Studio to collect events logged by QuidServer, parse them, and store them in InfluxDB, a TSDB.

Having the collection agent sending data to InfluxDB, the next step is to learn how InfluxDB works, learn its language Flux, and explore what InfluxDB can do and after exploring the data with different queries, a dashboard is created using InfluxDB's Boards tool. In addition, InfluxDB is benchmarked to understand the storage space taken by each metric, how it stores data and how InfluxDB compares with other TSDBs. This allows for a better understanding of InfluxDB, which is crucial to optimize the persistence of the data.

The last functionality to explore in InfluxDB is the Tasks tool which allows to create continuous queries that run automatically and periodically, which is especially useful for downsampling since it is possible to periodically run a query that aggregates data within windows of time and stores the aggregate values in a bucket with a larger retention period.

Since InfluxDB is mainly a TSDB, it is not optimized for data visualization, as it is more limiting than Grafana, which is specifically built for time series analytics and visualization. However, by exploring InfluxDB's dashboards, it is possible to use this knowledge and Grafana to recreate the dashboard created in InfluxDB. With this, InfluxDB is now only used to store and downsample data, while Grafana uses Flux to query the data and visualize it in an interactive dashboard.

Having a prototype metric collection agent completed, the next step is to explore and use Telegraf, InfluxData's own data collection agent, to collect the metrics and send them to InfluxDB to compare the benefits, development time, and complexity of the agent developed in C Sharp vs Telegraf. These factors will determine which method should be used to collect the data in the final product.

Until this point, the collection, persistence, and visualization are only being tested for the processing time of received messages, a metric that was previously implemented by the company. The next step in this project is then to understand how metrics are collected from a system, measuring data, and aggregating it into metrics that can be sent through ETW events to be captured by the metric collection agent. To do this, a new metric is implemented, the number of invoked scheduling tasks, as well as the processing time of said tasks.

Finally, Docker is used to run InfluxDB in one container and Grafana in another, which allows for the automation of the installation and configuration of InfluxDB and Grafana by using configuration files. After that, a script is constructed to automate the InfluxDB and Grafana first setup, facilitating the continuation of this project.

Chapter 5

Results and Discussion

In this chapter the results of the project will be presented and discussed, following the methods described previously, as seen in figure 4.1, in addition to any unplanned procedure that was implemented in response to either a result or an unexpected problem.

5.1 Analysis of architecture

Before starting the project, it is imperative to learn and understand how to use Genio and how it works as an extreme low-code model-driven software development platform to get a better understanding of it, as well as analysing the architecture of the systems generated by Genio and the structure of the databases created for them. This analysis allowed the understanding of the points of contact between the user and the system, as to identify what kind of metrics could be extracted, as well as where, in the code, they would be implemented.

The ultimate goal of this project is the collection, storage, and visualization of metrics from any system generated by Genio, but for prototyping purposes, one particular system generated by Genio, QuidServer is used to generate mock metrics. Here, QuidServer acts as both a scheduler and a message broker and a MockWebApi sends messages to QuidServer, running in parallel with it. This is what triggers QuidServer to log events. QuidServer does this by having an event provider wired into it, that calls the logging API. Thus, to develop an event collection agent, the architectures of QuidServer, the MockWebApi and the events logged by QuidServer were analysed, not only to understand how the message processing time was already developed as a metric by the company, but also to be able to replicate the procedure to create new metrics. Each message channel was assigned to a Stopwatch instance, an EventCounter instance and a general EventCounter instance shared by all message channels. The Stopwatch would be started before the processing of each message and after the message was processed, the Stopwatch was stopped and the elapsed time was written in both EventCounters and by the end of each time window (set to 5 seconds) the EventCounters would log ETW events containing the number of

processed messages and a summary of the processing times (max, mean, min and stdev). Having both EventCounters allowed to have data for each specific message channel, as well as for all channels.

5.2 Metric collection

With the understanding of the mechanism and structure of the events logged by QuidServer, an event collection agent was developed in C Sharp using Microsoft Visual Studio to collect said events. This agent uses the TraceEvent library and is composed of an event controller and an event consumer. The controller creates an event tracing session and enables the provider to start logging, and the consumer reads events from the session in real time, parses them into data points using line protocol and writes them into an InfluxDB bucket using its write API. However, to persist the data in InfluxDB, it had to be installed, started, and set up to create an InfluxDB user client that can write data. This includes creating an organization name, a bucket, and an Operator API token.

5.3 Metric persistence

Having the collection agent sending data to InfluxDB, the next step was to learn how InfluxDB works, learn its language Flux, and explore what InfluxDB can do. InfluxDB has a Data Explorer tool that allows for querying, processing, and visualizing data. When building a query, it is necessary to define the data source (like a bucket), the time range and data filters like measurement, field keys and tag values. A query can be built in the Query Builder or the Script Editor. Querying with the Query Builder is as simple as selecting the bucket, time range and filters in the UI. However, this way of building queries can be limiting. To allow for more freedom, the Script Editor can be used, in which the query is manually written using the Flux language. In addition to filtering, InfluxDB also allows for the shaping and processing of data. Shaping consists of modifying the data structure to prepare it for processing and includes grouping, pivoting, dropping, and keeping specific columns, while processing includes aggregate functions, selecting and rewriting rows. Finally, InfluxDB also provides different visualization types like graph, heatmap, histogram or table, as well as the ability to choose the time window in which to aggregate data.

5.4 Metric visualization

After exploring the data with different queries, a dashboard was created using InfluxDB's Boards tool, as shown in figure 5.1. Queries built in the Data Explorer can be exported into different cells of a dashboard with the Boards tool. This allows for the simultaneous

visualization of different data and to better interact with the dashboard cells, it is possible to create dashboard variables which allow to alter specific cell components, like a tag value, without editing them.

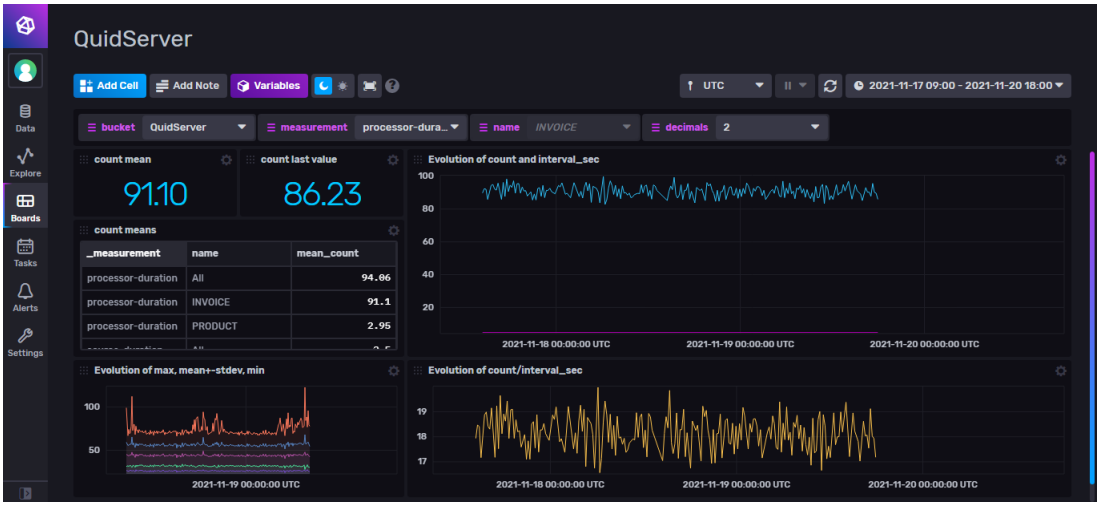

Figure 5.1: Interactive dashboard developed using InfluxDB

In addition, InfluxDB is benchmarked to understand the storage space taken by each metric, how it stores data and how InfluxDB compares with other TSDBs. This allows for a better understanding of InfluxDB, which is crucial to optimize the persistence of the data.

The last functionality explored in InfluxDB was the Tasks tool which allows to create continuous queries that run automatically and periodically. This is especially useful for downsampling since it is possible to periodically run a query that aggregates data within windows of time and stores the aggregate values in a bucket with a larger retention period.

Since InfluxDB is mainly a TSDB, it is not optimized for data visualization, as it is more limiting than Grafana, which is specifically built for time series analytics and visualization. However, by exploring InfluxDB's dashboards, it was possible to use this knowledge and use Grafana to recreate the dashboard created in InfluxDB. Grafana's dashboards have panels that have three different functionalities: query, transform and alert. This allows to remove the least performant part of a query, which is processing the data. Instead, the query can simply fetch the needed data (which makes the query much faster and performant) and Grafana's transform feature is designed to efficiently process the queried data. With this, InfluxDB is now only used to store and downsample data, while Grafana uses Flux to query the data and visualize it in an interactive dashboard, as shown in figure 5.2.

The developed dashboard contains 7 panels. The first shows a time series graph with the evolution of the number of processed messages through time; the second panel is also a time series graph, but it shows the throughput rate of messages, which is calculated by dividing the number of processed messages by the time interval; the third panel shows

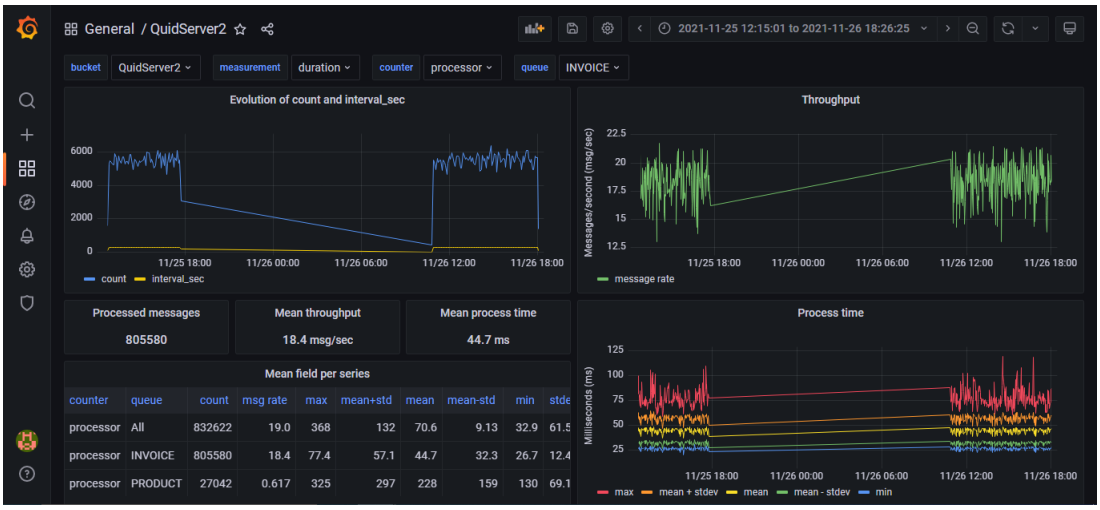

Figure 5.2: Interactive dashboard developed using Grafana

the total number of processed messages; the fourth shows the mean throughput rate of messages; the fifth shows the mean processing time; the sixth shows a time series graph that shows the evolution of the max, mean, and min processing time, as well as the standard deviation (stdev), mean + stdev and mean - stdev (this is useful to know the range in which the processing time of most messages is at a given moment); finally, the seventh panel shows a summary table of the various metrics.

5.5 Comparison with Telegraf

Having a prototype metric collection agent completed, the next step is to explore and use Telegraf, InfluxData's own data collection agent, to collect the metrics and send them to InfluxDB to compare the benefits, development time, and complexity of the agent developed in C Sharp vs Telegraf. These factors will determine which method should be used to collect the data in the final product.

Telegraf input and output plugins are enabled and configured in Telegraf's configuration file (telegraf.conf), so the InfluxDB (OSS v2.0) UI was used to automatically generate a Telegraf configuration. This configuration uses the outputs.influxdb_v2 plugin to write metrics to InfluxDB and the system monitoring group of plugins was picked to test Telegraf (v1.21). The plugins that constitute the system monitoring group of plugins are the following: inputs.cpu, inputs.disk, inputs.diskio, inputs.mem, inputs.net, inputs.processes, inputs.swap and inputs.system. These plugins gather metrics about CPU and disk usage, disk IO, system memory, network interface usage, number of processes grouped by status, swap memory usage and general stats on system load, uptime and number of users logged in. However, as the inputs.processes plugin is specific for Linux, it had to be replaced by the inputs.win_perf_counters and inputs.win_services plugins which

are specific for Windows and gather metrics about Performance Counters and Windows services, respectively. With this, InfluxDB automatically generates a dashboard to visualize the system monitoring data, as seen in figure 5.3 which confirms that both Telegraf and its plugins are working as expected.

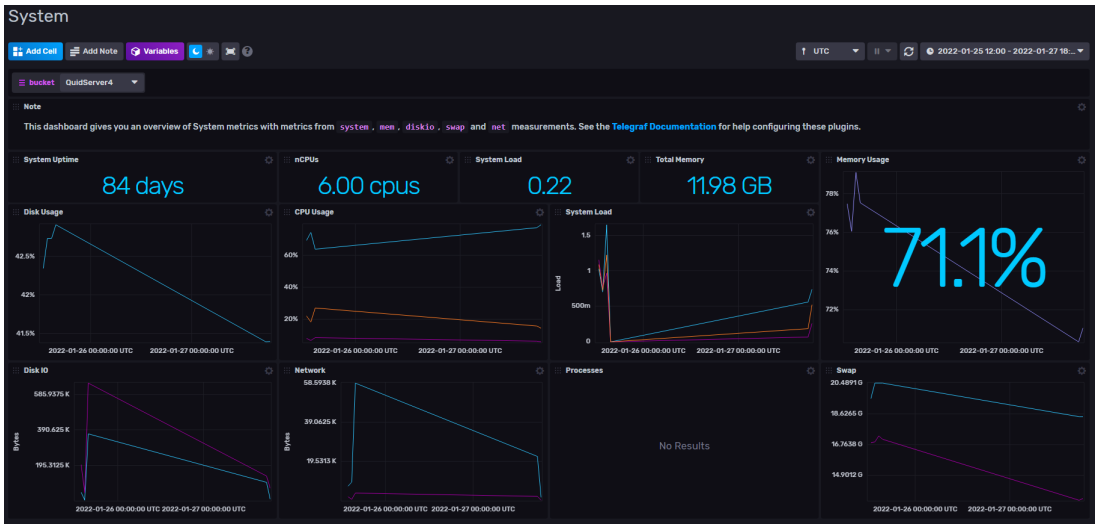

Figure 5.3: Dashboard automatically generated by InfluxDB to visualize the system monitoring data

Having tested the transfer of data between Telegraf and InfluxDB, the next step is to test the collection of ETW events with Telegraf. Unfortunately, there is not an input plugin specific for ETW event collection available, so an external plugin would have to be developed for this use case. External plugins can run through Telegraf's inputs.execd plugin, a plugin designed to run external programs that output formatted metrics on the process's STDOUT and stay running. The plugin can be built from a sample Telegraf configuration provided by the documentation[20] which is written in Go and should be modified to collect ETW events. A summary of the comparison between Telegraf and QMetrics, the metric collection agent developed in C Sharp, is available in table 5.1.

Metric collection agent	QMetrics	Telegraf
Benefits	Customizable code tailored to the project's use case	Flexibility of configuring and using different plugins
Development time	<6 days	7+ days
Complexity	Simple program in C#	Complex plugin written in Go

Table 5.1: Summary table of comparison between Telegraf and QMetrics, the metric collection agent developed in C Sharp.

The advantages of using Telegraf would be the flexibility of configuring and using

different plugins, which would be useful if the project required the use of one or more of the available plugins. However, since the project's use case is not supported and since the process of using Telegraf proved to be much more complex and time consuming than the development of the metric collection agent developed in C Sharp, it was decided that Telegraf would not be the metric collection agent used.

5.6 Implementation of new metric

Until this point, the collection, persistence, and visualization are only being tested for the processing time of received messages, a metric that was previously implemented by the company. The next step in this project is then to understand how metrics are collected from a system, measuring data, and aggregating it into metrics that can be sent through ETW events to be captured by the metric collection agent. To do this, a new metric is implemented, the number of invoked scheduling tasks, as well as the processing time of said tasks.

The implementation of this metric was very similar to the metric of the message processing time, since processing time is still being measured. The only difference is that since tasks can be concurrent, each one must have a new distinct Stopwatch instead of each channel having it is assigned Stopwatch. This is to ensure that even if two different tasks of the same channel are being processed at the same time, each of them will still have their own Stopwatch.

5.7 Docker containerization

Finally, Docker is used to run InfluxDB in one container and Grafana in another, which allows for the automation of the installation and configuration of InfluxDB and Grafana by using configuration files. From the company point of view, Docker has the disadvantage of having to be installed by the client, who may not agree to this. Besides, the docker engine must be running in addition to the metric collection agent, which implies a high consumption of resources. However, not using Docker would mean the installation of two applications (InfluxDB and Grafana) instead of one (docker engine), on top of the manual installation of both applications, which not only translates to more time spent in configuration, but it also leads to a higher number of errors or differences in each configuration. After that, a script is constructed to automate the InfluxDB and Grafana first setup, facilitating the continuation of this project.

Chapter 6

Conclusion

Throughout this project an event collection agent was developed to collect events logged by QuidServer and write them into InfluxDB as data points. This step was not only useful to understand the structure of the events logged by QuidServer, but also to understand the structure of the data points written into InfluxDB. The dashboard feature provided by InfluxDB was used to design dashboards and explore the collected data and using this knowledge, Grafana was used to create a dashboard that exploits the advantages of both InfluxDB and Grafana. The result is a metric collection, persistence and visualization framework which allows to examine the data in real-time and detail as well as to store and downsample older data to compare the data over days, months, or years. In addition to this, docker containers for InfluxDB and Grafana were developed in order to wrap these applications and their dependencies, as well as automating the configuration process. Finally, the monitoring of one more metric was implemented, so future work would include the addition of more metrics, such as frequent errors, request duration, session duration and number of users per day as well as to integrate the created dashboard in Genio with the aim of every system generated by it to have its own statistical dashboard.

Bibliography

[1] About event tracing - win32 apps — microsoft docs. [Online]. Accessed: 2021-12-22. Available: https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing.

[2] C sharp docs - get started, tutorials, reference. — microsoft docs. [Online]. Accessed: 2021-12-22. Available: https://docs.microsoft.com/en-us/dotnet/csharp/.

[3] Db-engines ranking - popularity ranking of database management systems. [Online]. Accessed: 2021-12-22. Available: https://db-engines.com/en/ranking.

[4] Eventcounter class (system.diagnostics.tracing) — microsoft docs. [Online]. Accessed: 2021-12-22. Available: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.tracing.eventcounter?view=net-6.0.

[5] Flux 0.x documentation. [Online]. Accessed: 2021-12-22. Available: https://docs.influxdata.com/flux/v0.x/.

[6] Genio platform — quidgest. [Online]. Accessed: 2021-12-02. Available: https://genio.quidgest.com/.

[7] The go programming language. [Online]. Accessed: 2021-12-22. Available: https://go.dev/.

[8] Grafana documentation — grafana labs. [Online]. Accessed: 2021-12-22. Available: https://grafana.com/docs/grafana/latest/.

[9] Grafana: The open observability platform — grafana labs. [Online]. Accessed: 2021-12-22. Available: https://grafana.com/.

[10] historical trend of time series dbms popularity. [Online]. Accessed: 2021-12-22. Available: https://db-engines.com/en/ranking$_trend/time + series + dbms$.

[11] Home - docker. [Online]. Accessed: 2022-09-25. Available: https://www.docker.com/.

[12] Home — cern. [Online]. Accessed: 2022-01-02. Available: https://home.cern/.

[13] Influxdb 1.x: Open source time series platform — influxdata. [Online]. Accessed: 2021-12-28. Available: https://www.influxdata.com/time-series-platform/.

[14] Influxdb: Open source time series database — influxdata. [Online]. Accessed: 2021-12-22. Available: https://www.influxdata.com/.

[15] Influxdb oss 2.0 documentation. [Online]. Accessed: 2021-12-22. Available: https://docs.influxdata.com/influxdb/v2.0/.

[16] perfview/traceeventlibrary.md at main · microsoft/perfview · github. [Online]. Accessed: 2021-12-22. Available: https://github.com/microsoft/perfview/blob/main/documentation/TraceEvent/TraceEventLibrary.md.

[17] Prometheus - monitoring system time series database. [Online]. Accessed: 2021-12-28. Available: https://prometheus.io/.

[18] Quidgest: Future-ready software. [Online]. Accessed: 2021-12-02. Available: https://quidgest.com/.

[19] Telegraf open source server agent — influxdb. [Online]. Accessed: 2021-12-22. Available: https://www.influxdata.com/time-series-platform/telegraf/.

[20] telegraf/inputs.md at master · influxdata/telegraf · github. [Online]. Accessed: 2022-09-25. Available: https://github.com/influxdata/telegraf/blob/master/docs/INPUTS.md.

[21] Visual studio: Ide and code editor for software developers and teams. [Online]. Accessed: 2021-12-22. Available: https://visualstudio.microsoft.com/.

[22] Thomas Beermann, Aleksandr Alekseev, Dario Baberis, Sabine Crépé-Renaudin, Jo-Hannes Elmsheuser, Ivan Glushkov, Michal Svatos, Armen Vartapetian, Petr Vokac, and Helmut Wolters. Implementation of atlas distributed computing monitor-ing dashboards using influxdb and grafana. 2020.

[23] Hugo Van Dijk and Claudia Hauff. Expanding logui: Adding screen capturing and a statistical analysis dashboard for web-based experiments, 2021.

[24] Johannes Elmsheuser and Alessandro Di Girolamo. Overview of the atlas distributed computing system. 2019.

[25] Radoslav Gatev. Observability: Logs, metrics, and traces. *Introducing Distributed Application Runtime (Dapr)*, pages 233–252, 2021.

[26] Mohammad Abu Kausar. Suitability of influxdb database for iot applications. *Article in International Journal of Innovative Technology and Exploring Engineering*, pages 2278–3075, 2019.

[27] Gunasekaran Manogaran, Daphne Lopez, Chandu Thota, Kaja M Abbas, Saumyadipta Pyne, and Revathi Sundarasekar. Innovative healthcare systems for the 21st century. understanding complex systems. pages 263–284, 2017.

[28] David Maxwell and Claudia Hauff. Logui : Contemporary logging infrastructure for web-based experiments. 2021.

[29] Franck Michel. Integrating heterogeneous data sources in the web of data. 2017.

[30] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover. Exact discovery of time series motifs. 2009.

[31] Syeda Noor, Zehra Naqvi, Sofia Yfantidou, Esteban Zimányi, and Zim´ Zimányi. Time series databases and influxdb. 2017.

[32] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla. *Proceedings of the VLDB Endowment*, 8:1816–1827, 8 2015.

[33] Prapaporn Rattanatamrong, Yoottana Boonpalit, Siwakorn Suwanjinda, Ayuth Mangmeesap, Shava Smallen, Ken Subraties, Vahid Daneshmand, and Jason Haga. Overhead study of telegraf as a real-time monitoring agent. *JCSSE 2020 - 17th International Joint Conference on Computer Science and Software Engineering*, pages 42–46, 11 2020.

[34] Benjamin H Sigelman, André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Google technical report dapper dapper, a large-scale distributed systems tracing infrastructure. 2010.

[35] Felix Wortmann and Kristina Flü. Internet of things technology and value added. 2015.