**Digital Twin Monitoring System**

Internship Report

Budurovici Cosmin-Andrei

*13/01/2025*

Semester 6 – Internship

Venlo, Limburg, Netherlands

# Internship Report Information

Fontys University of Applied Sciences

School of Technology and Logistics

Post Office Box 141, 5900 AC, Venlo, Netherlands

|  |  |
| --- | --- |
| Type of report:  Student Name:  Student Number:  Study:  Period:  Company name:  Address:  Postal code + City:  Country:  Telephone:  Company supervisor:  Supervising Lecturer:  Company Confidential:  Number of words: | Internship Report  Cosmin Budurovici  4514270  Informatics – Software Engineering  September 2024 – January 2025  Fontys School of Technology and Logistics  Tegelseweg 255  5912 BG Venlo  Netherlands  +31 885089271  Dr. Gregor Schwake  Dr. Gregor Schwake  No  9875 words |

# Abstract

*The SPoHF project is a project that aims to improve the quality of fruit and vegetable cultivation by creating a digital twin a program that can monitor and then simulate plants providing valuable information which can be used to improve the cultivation.*

*One of the partners of SPoHF Fontys Venlo has commissioned the Digital Twin Monitoring System to monitor digital twin.*

*The project is organized using waterfall for the initial phases and agile for the actual implementation. Jira is used for keeping track of tasks.*

*Through a series of interviews, it was determined that the project should monitor distributed traces, errors and hardware of Digital Twin. While offering an easily understandable web interface.*

*Trough criteria-based selection the Prometheus Node Exporter, Prometheus and Grafana tools were selected for use in this project. The Monitoring application was then designed and implemented.*

*At the end of development, the Monitoring Application was functional but unable to monitor traces. It did not monitor all of digital twin and the provided interface could only be understood by those with technical knowledge. The all-other features were successfully implemented.*

# Statement of Authenticity

I, the undersigned, hereby certify that I have compiled and written the attached document / piece of work and the underlying work without assistance from anyone except the specifically assigned academic supervisors and examiners. This work is solely my own, and I am solely responsible for the content, organization, and making of this document / piece of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document / piece of work will not be accepted for evaluation or for the award of academic credits if it is determined that it has not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document / piece of work and the underlying work will not be accepted for evaluation or for the award of academic credits if it is determined that it embodies plagiarism.

|  |  |
| --- | --- |
| Name:  Student number:  Place/Date:  Signature: | Cosmin Budurovici  4514270  Venlo, 13/01/2025  …………………………. |

Table of Contents

[Internship Report Information II](#_Toc187766335)

[Abstract III](#_Toc187766336)

[Statement of Authenticity IV](#_Toc187766337)

[List of Figures VII](#_Toc187766338)

[List of Tables VIII](#_Toc187766339)

[Glossary and List of Abbreviations IX](#_Toc187766340)

[1 Introduction 1](#_Toc187766341)

[2 Project Management 2](#_Toc187766342)

[2.1 Problem Description 2](#_Toc187766343)

[2.2 Objective 2](#_Toc187766344)

[2.3 Scope 3](#_Toc187766345)

[2.4 Potential Risks 3](#_Toc187766346)

[2.5 Stakeholders 5](#_Toc187766347)

[2.6 Organization 5](#_Toc187766348)

[2.6.1 Approach 6](#_Toc187766349)

[2.6.2 Methodology 7](#_Toc187766350)

[3 Analysis 8](#_Toc187766351)

[4 Design 9](#_Toc187766352)

[4.1 Tool Selection 9](#_Toc187766353)

[4.2 High-Level Overview 14](#_Toc187766354)

[4.3 Development environment 15](#_Toc187766355)

[5 Implementation 15](#_Toc187766356)

[5.1 Sprint 1: Hardware monitoring 15](#_Toc187766357)

[5.2 Sprint 2: Error monitoring 17](#_Toc187766358)

[5.3 Sprint 3: Trace monitoring 21](#_Toc187766359)

[6 Conclusion 22](#_Toc187766360)

[References 23](#_Toc187766361)

[Appendix 24](#_Toc187766362)

[Appendix A: Interview Questions 24](#_Toc187766363)

[First interview (Cristiane Salz) 24](#_Toc187766364)

[Second interview (Tim Riebner) 24](#_Toc187766365)

[Third interview (Compass Agro) 24](#_Toc187766366)

[Appendix B: Tool Selection Criteria 24](#_Toc187766367)

[General Criteria (Applies to all areas) 24](#_Toc187766368)

[Monitoring data storage 26](#_Toc187766369)

[Hardware monitoring 26](#_Toc187766370)

[Monitoring visualization 26](#_Toc187766371)

[Appendix C: Unchosen Tools 27](#_Toc187766372)

[Unchosen monitoring data storage 27](#_Toc187766373)

[Unchosen hardware monitoring tools 28](#_Toc187766374)

[Unchosen monitoring visualization 28](#_Toc187766375)

[Appendix D: Score Explanations 28](#_Toc187766376)

[Monitoring Data Storage 28](#_Toc187766377)

[Hardware Monitoring 29](#_Toc187766378)

[Monitoring Visualization 29](#_Toc187766379)

[Appendix E: Configuration Environment 30](#_Toc187766380)

[Prometheus node exporter 30](#_Toc187766381)

[Apache spark 31](#_Toc187766382)

[Configuration for Loki 32](#_Toc187766383)

[Configuration for Prometheus 33](#_Toc187766384)

[Prometheus 33](#_Toc187766385)

[Loki+Promtail 35](#_Toc187766386)

[Grafana 38](#_Toc187766387)

# List of Figures

[Figure 1: SPoHF structure 1](#_Toc187766388)

[Figure 2: Risks impact/probability 3](#_Toc187766389)

[Figure 3:Aproach overview 6](#_Toc187766390)

[Figure 4: Architectural diagram 1.0 14](#_Toc187766391)

[Figure 5: Hardware monitoring mockup 16](#_Toc187766392)

[Figure 6: Hardware monitoring dashboard 17](#_Toc187766393)

[Figure 7: Architectural diagram 2.0 18](#_Toc187766394)

[Figure 8: Error monitoring mockup 19](#_Toc187766395)

[Figure 9: Error monitoring interface 20](#_Toc187766396)

[Figure 10:Updated Error Monitoring with Trace Monitoring 21](#_Toc187766397)

# List of Tables

[Table 1: Glossary and list of abbreviations IX](#_Toc187766398)

[Table 2: List of risks 3](#_Toc187766399)

[Table 3: Stakeholders 5](#_Toc187766400)

[Table 4: User stories 9](#_Toc187766401)

[Table 5: Possible data storage tools 12](#_Toc187766402)

[Table 6: Posible hardware monitoring tools 12](#_Toc187766403)

[Table 7: Posible visualization tools 13](#_Toc187766404)

[Table 8: Sprint List 15](#_Toc187766405)

[Table 9: Unselected data storage tools 27](#_Toc187766406)

[Table 10: Unselected hardware monitoring tools 28](#_Toc187766407)

[Table 11: Unselected visualization tools 28](#_Toc187766408)

# Glossary and List of Abbreviations

|  |  |
| --- | --- |
| Term | Definition |
| Monitoring System | A shortened version of the Digital Twin Monitoring System |
| User story | An informal and natural description of a feature of an application |
| Job | An operation done by Apache Spark on a piece of data |
| U.S. | User story |
| Distributed trace | Represents the timeline and all the actions that occur between an initial request to the application and its completion. |

Table 1: Glossary and list of abbreviations

# 1 Introduction

The purpose of this document is to provide a report on the development of Digital Twin Monitoring System.

This chapter will provide background information as well as an introduction to both the SPoHF project, Digital Twin as well as the Digital Twin Monitoring System.

SPoHF is a Dutch German project co-financed by the European Union. It focuses on the improvement of fruit and vegetable cultivation, by creating “Digital Twins”, programs that can monitor and then simulate plants providing valuable information that can be used in agriculture to use less resources, energy, pesticides and fertilizers while reducing environmental impact and improving food quality.

The SPoHF project has various partners including Fontys University of Applied Sciences, Compass Agro, Proc Evolution, Bright labs, Yookr, Oecotrophologie, Gemit and Appcom.

A diagram of a diagram

Description automatically generated

Figure 1: SPoHF structure

Figure 1 provides a high-level overview of the SPoHF project and all its components. Below there is an explanation of each component and its role in the larger picture:

* Lightning Monitoring, Climate Monitoring, Insect Monitoring, Soil Analysis, Weather Monitoring and Climate Monitoring are all programs that use various tools to monitor various information about a plant’s environment and send it to Digital Twin
* Digital Twin receives information from various programs and stores and organizes it.
* Prescriptive twin role is to use AI, data science and information provided by Digital twin to prescribe things that can be done to improve cultivation.
* Predictive twin will use AI, data science and information provided by Digital Twin to predict how plants and their environment will change over time.
* Monitoring represents the Digital Twin Monitoring System; it was commissioned by Fontys Venlo (more precisely the informatics department), and its role is to monitor Digital Twin allowing users to know how well the program functions and when there is a problem.

# 2 Project Management

This chapter will provide information on the objectives of the project as well as how these objectives will be caried out.

## 2.1 Problem Description

The Digital Twin application receives data from a variety of sensors from various farms and then processes the data, ordering it and putting into a database, before making it available for analysis by AI and people.

For the digital twin project to be used effectively its users need to be able to know if the project is functioning at any given time. The users need to know when any part of the system has problems and what impact these problems have on the larger system so that they can fix them and take measures to mitigate any problems caused by the failure. While this information can be gathered by manually looking at the system this is often slow, error prone and often reveals problems too late.

## 2.2 Objective

The objective of the Digital Twin Monitoring System is to provide a way to see the current state of Digital Twin as well as give details about performance and notify users of any problems that occur. It accomplishes this by collecting data from Digital Twin (it does not directly collect data from other subprojects such as Insect Monitoring and Climate Monitoring) and displaying it through a visual interface in real time.

The application will mainly be used by the Proc Evolution company, making them the customers for this project.

## 2.3 Scope

To accomplish our objective the monitoring system will:

* Provide information about what tasks is digital twin performing at any given time.
* Provide information about the current state of the hardware at any given time.
* Show when any part of the system fails and what consequence this has.
* Display this information through a visual interface in real time.

The monitoring system will not:

* Send notifications to users when it detects a failure.
* Have functionality to solve the problems it detects (it’s solely a detection tool)

It’s important to mention that this project will only cover the creation of the application not it’s updates and maintenance.

## 2.4 Potential Risks

There are several risks that need to be considered, prevented and possibly mitigated. This chapter will discuss each risk in detail, how to prevent and mitigate them and how probable or impactful each risk is.

Table 2 is a list of all risks that could cause problems for the Monitoring system, their probability of occurring and their impact. Figure 2 is the same information outlined in a more visual form.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Probability | Impact |
| 1 | Unfinished application | Low | High |
| 2 | Insufficient hardware | Low | Medium |

Table 2: List of risks

A diagram of a graph

Description automatically generated with medium confidence

Figure 2: Risks impact/probability

The individual risks outlined in this section will be discussed in more detail below:

**Name:** Unfinished application

**Description:** For a Monitoring Application to be used there needs to be another application that will be monitored. In our case this application is Digital Twin. Since Digital Twin is not yet completed there is a chance that it will never be completed or more likely that it will have significant delays leaving the Monitoring System without an application to monitor.

**Prevention:** Due to the development of Digital Twin being the responsibility of a different team there aren’t any significant measures to prevent the incompletion of Digital Twin. Communication will be maintained with the Digital Twin developers to know as early as possible if a problem occurs.

**Mitigation:** In the case that Digital Twin can’t be completed in time a barebones demo of digital twin will be created to monitor.

**Name:** Insufficient hardware

**Description:** At the current moment both the capabilities of the server the application will be running on as well as how much of those capabilities will be occupied by other applications running on the same server. As a result, there is a chance that the hardware will be unable to support the Digital Twin Monitoring System.

**Prevention:** The only measure we can take is to make sure our application is as efficient as possible and doesn’t occupy to much memory.

**Mitigation:** There are two measures that can be taken to minimize this risk once it occurs. First is to communicate with our customer and request more powerful hardware. Second would be to try to rework the application to take less resources.

## 2.5 Stakeholders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Company | Influence | Impact | How to engage |
| Budurovici Razvan | Fontys | Low | High | Frequent meetings |
| Gregory Schwake | Fontys | High | High | Frequent meetings |
| Tim Reibner | Proc evolution | High | High | Frequent meetings |
| Cristiane Salz | Fontys | Low | Low | Occasional meetings |
| Joel Eckhardt | Fontys | Low | High | Occasional meetings |

Table : Stakeholders

Table 3 is a list of the stakeholders of this project, which company they work for and what their authority over the direction of the project and impact they have.

Each stakeholder present is explained in more detail below:

* Razvan Budurovici is the **developer of the Digital Twin Insect Counting Integration**. His code is what needs to be monitored for this project making communication with him essential.
* Gregory Schwake is the **project manager.** He checks the progress of the project as well as guiding it and giving feedback. He must be always kept up to date.
* Tim Reibner is the **customer representative** for Proc Evolution. Since his company will be using the Monitoring System after completion, he has the say on what features need to be implemented
* Cristiane Salz **developer of Insect Counting** his application provides data to Digital Twin as a result understanding his application will help in understanding Digital Twin.
* Joel Eckhardt is the lead developer on the generic interface for Digital Twin. The Generic Interface is the part of digital twin responsible for receiving data from insect counting and passing it on to the rest of digital twin. As such his application also needs to be monitored, the monitoring of his application is a significantly lower priority than the rest of Digital Twin developed by Razvan Budurovici.

## 2.6 Organization

In the following chapter the approach of this project, the methodology and the timeline will be detailed.

### 2.6.1 Approach

A diagram of a project

Description automatically generated

Figure :Aproach overview

As seen in figure 3 this project will be split into five different phases: Introduction, Analysis, Design, Implementation and Rollout. For any one of these phases and deliverables to be considered done the work needs to be approved by the customer representative and/or the project manager. Each of the five phases and their deliverables are explained below:

1. Introduction. This phase consists of the developer of this project being introduced to and familiarizing with the project. This phase of the project is considered done after all the initial meetings are done.
2. Analysis. This phase consists of assessing what the application needs to accomplish and why. The deliverable of this phase is the product backlog.
   1. Product backlog is considered done after the creation of a series of user stories.
3. Design. This phase consists of taking the information gathered in the analysis phase and using it to determine, in broad strokes, how this project will accomplish these goals. This phase has three deliverables:
   1. The tool selection refers to the research and the selection of necessary tools for use in the completion of this project. It is considered done when we have a list of tools with justification on how each one will be used to complete the features outlined in the user stories
   2. The high-level overview consists of making one or more diagrams that will serve as blueprints for our project. It is considered done when we have at least one diagram, the high-level architecture diagram. The project might require other diagrams for particularly complex parts of the project.
   3. The development environment is considered done when the following has been done: set up the project framework, installed and configured all the used tools and tested the basic framework of the project with a demo.
4. The implementation represents the coding of our application. It is considered done when all the features outlined in the user stories have been completed and tested.
5. Rollout consists of configuration the application an installing it a Proc Evolution server and it’s functional.

## 2.6.2 Methodology

This project will be managed using a combination of both waterfall and agile. The analysis, design and rollout will be done using a waterfall stile while the implementation will be done using the sprints of agile. While waterfall is better for the initial stages of the project because of its simplicity, the Monitoring application is depended on Digital Twin ass a result it is better to use agile to adapt to any changes in Digital Twin.

For managing the project will be using Jira. The management program will be used for keeping track of the project board, the timeline, individual tasks, subtasks and the project backlog. Jira was chosen for two reasons. First other stakeholders are also using it for their parts of the project. Second Jira is a good all-around tool that makes it easy to keep track of timelines.

A diagram of a product selection

Description automatically generated

In figure 4 there are all the necessary deliverables of the project and how long each one will take to complete. How each deliverable will be done is detailed below:

* For the introduction the developer will be introduced to the application and stakeholders through a series of meetings.
* To complete the product, backlog the developer will conduct 3 interviews one with Compass Agro, to provide information about what the application will be used for, one with the developer of insect counting, to provide more information on the workings of digital twin and one with the customer representative to establish what features are needed. For each interview a list of questions will be created in advance. The information from the interview will be used to create a series of user stories.
* The tool selection is the first deliverable of the design phase, and it consists of multiple steps. First, determine different areas that tools are needed to be used for. Second, utilize the existing use cases to create a series of criteria that can be used for determining which tools to select. Third find a series of tools that could be used for the monitoring application and rate them depending on how well they fulfill the criteria. Only trusted sources will be used to find tools and determine their characteristics. Lastly, the tools that will be used are selected.
* High-level overview consists of first making the high-level architecture diagram, which will include the selected tools and their application. To better understand more complex parts of the project, such as the database, more diagrams besides the first might be made.
* The development environment and implementation will be caried out using the Java programing language, the various tools that have been previously selected and the Mac operating system. To ensure the quality of the code we will use two measures: test driven development and frequent code review by Razvan Budurovici (the developer of digital twin) and the project manager.
* The configuration (part of the rollout) will be caried out by first showcasing the monitoring application to the customer and then installing it one of their devices

# 3 Analysis

The chapter covers the analysis part of the project. More precisely creation of the product backlog through a series of user stories.

To create the user stories there has been tree interviews with various stakeholders. Before each interview a list of questions was made to ensure that all relevant points were brought up to interviewed. A full list of all prepared interview questions can be found in Appendix A: Interview Questions.

The first interview was with the developer of insect counting. This interview offered us detailed information on how the digital twin will receive information from Insect Counting, but not significant information on how our application needs to perform.

The second interview was with the customer representative. As the customer this interview provided the bulk of the information we have used for our user stories.

The third interview was conducted significantly later due to delays and involved the developers going directly to Compass Agro. This interview provided useful insights into what the Digital Twin (the application that will be monitored) was used for.

Using the information gathered throughout the interviews we here able to make the following list of user stories:

1. As a developer I want to be able to see the trace of each individual batch of information so that I can assess the efficiency of each process.
2. As a developer or manager, I want to be able to see data from the last 4 weeks so that I can keep track of the systems functionality over time.
3. As a developer I want to be able to see the data of currently processing baches in real time so that I can assess the current state of the system.
4. As a developer I want to see the current state of the server the main application is running on so that I can see the state of the system as a hole.
5. As a developer I want to be able to see the error message produced by the application when there is a problem so that I can have a better idea of how to solve the problem.
6. As a manager I want to see the consequences of any failure in an understandable way so that I can understand what went wrong and what consequences that failure has.
7. As a manager and developer, I want to be able to access the monitoring application with any device that can connect to the internet so that I can more easily access the data that I need.

Table 4: User stories

This list of uses was then approved by both the customer representative and the making this phase of the project complete.

# 4 Design

This chapter will go over the development phase of this project. This phase involves tree major deliverables the selection of tools, the creation of high-level architecture and the environment set up.

## 4.1 Tool Selection

While it is possible to code the functionality of this program from scratch using already preexisting tools for the same purpose is easier and less error prone.

The first step of the process was figuring out in what areas we can use a tool instead of custom code. After analyzing our user stories and our knowledge of the program well be monitoring, we have determined three areas where we can use tools:

* **Monitoring data storage:** To display the collected metrics of the application for 4 weeks there needs to be some way to store for the visualization to have data to pull from after the fact.
* **Hardware monitoring:** Some of the important metrics of the application are the remaining storage, ram and CPU usage. There are many tools that can be used to get the hardware metrics of multiple nodes easily.
* **Monitoring Visualization:** Using an already preexisting tool to display the information in a web interface will be significantly easier

The second step is determining what sources we will be using for researching different existing tools. Using only trusted sources is important in to make sure the selection is not affected by misinformation and as objective as possible. Below there are listed the sources well be using:

* **Stack share** is the most popular website that specifically deals with finding different tools for applications. It is used by many companies and developers and shows what companies utilize which tools which is a useful metric to determine if a tool is widely used professionally.
* **The websites of the tools themselves** will be the primary source of information regarding the features and capabilities of each tool
* **Slant** a website designed for comparing different software tools and their upsides and downsides.
* **LibHunt** a website that provides curated lists, alternatives and comparisons for different tools. Is reliable doe to its size. Is a good indicator of a tools ecosystem since it includes GitHub metrics.

The third step is using knowledge of the project and user stories to create a series of criteria that can be used to select the best tool for the project. There are two types of criteria normal and **knockout criteria**. The normal criteria are each given a score that they can have. The higher the score the better the tool in that aspect. The **knockout criteria** can either be true or false and immediately disqualifies a tool from being chosen if it’s false.

Below there is a list of all criteria used for this project.

* General Criteria (Applies to all areas)
  + Ecosystem
  + Free of cost
  + Compatibility
  + Wildcard
* Monitoring data storage
  + 4-week storage
  + Time series
* Hardware monitoring
  + Basic hardware monitoring
* Monitoring visualization
  + Web interface
  + Real time display
  + Display features

Of these criteria Speed is notably absent as a criterion from the data storage area. Because the database will not be handling particularly large amounts of data and the fact that measuring the exact performance of a database for this project will be very difficult it was decided to not use speed as a criterion for this project.

Below we have an example of one of our criteria in more detail:

**Name:** Ecosystem

**Explanation:** The more widely used a tool is the easier it is to find information on how to use it such as tutorial or blogs and the easier it is to solve problems since there is a higher chance that other people have had a similar problem and solved it. For this criterion we will prioritize projects that have been popular recently since this reduces the chance that the tool is outdated.

**How this criterion is measured:** This criteria’s score will be generated using two metrics the number of stars on GitHub and the activity score on LibHunt.

The number of stars on GitHub is a good indicator of how widely popular a project is. For this criterion the points will be added in the following way:

* 5 points 40.000+ stars
* 4 points 30.000-40.000 stars
* 3 points 20.000-30.000 stars
* 2 points 10.000-20.000 stars
* 1 point 5.000-10.000 stars
* 0 points 0-5.000 stars

The lib hunt activity score is based on recent metrics such as commit frequency, issue and pull request activity and community engagement (number of discussion and responses to issues). This makes the LibHunt activity score a good indicator of the how currently popular and up to date the tool is. The points for this criterion from this metric will be added in the following way:

* 5 points 9-10 score
* 4 points 7-9 score
* 3 points 5-7 score
* 2 points 3-5 score
* 1 point 1-3 score
* 0 points 0-1 score

The points of this metrics are added together to form the final score of the criteria which will be between 0-10.

The full list of criteria in more detail can be found in Appendix B: Tool Selection Criteria.

The fourth step is searching for tools and then rating them according to the criteria.

Below there are tables of the tools that have been found that pass all the **knockout criteria.** A full list of all tools that have failed to meet our knockout criteria can be found in Appendix C: Unchosen tools.

**Possible tools for data storage:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Good ecosystem | Works with other tools | Wildcard | Total |
| Prometheus | 10 | 6 | 2 | 18 |
| Timescale DB | 7 | 3 | 1 | 11 |
| Quest DB | 7 | 0 | 0 | 7 |
| Influx DB | 8 | 3 | 0 | 11 |

Table : Possible data storage tools

**Possible tools for hardware monitoring:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Good ecosystem | Works with other tools | Wildcard | Total |
| Prometheus Node Exporter | 6 | 3 | 0 | 9 |
| Telegraph | 7 | 3 | 0 | 10 |
| Zabbix | 6 | 3 | 0 | 9 |

Table : Posible hardware monitoring tools

**Possible tools for visualization:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Good ecosystem | Compatibility | Display features | Wildcard | Total |
| Grafana | 10 | 3 | 10 | 2 | 25 |
| Apache Superset | 10 | 3 | 7 | 0 | 21 |
| Metabase | 9 | 3 | 6 | 0 | 18 |

Table : Posible visualization tools

A full explanation of each score a tool received can be found in Appendix D: Score Explanations.

The fifth step is choosing which tools will be used.

For each category the highest score that is held by Prometheus, Telegraph and Grafana. Prometheus and Grafana work well with each other and are easy to use, but Telegraph does not work with Prometheus. Because of this the actual option with the highest score is the one that works with Prometheus, the Prometheus node exporter.

The final selection is Prometheus node exporter, Prometheus and Grafana. These choices were approved by the project manager and customer representative.

## 4.2 High-Level Overview

After analyzing what is known about the project at this stage, it has been determined that there isn’t any part of the project where a diagram would be necessary besides the high-level overview.

The high-level overview was made by studding our user stories and our used tools to make a design for the architecture of the monitoring application.

A diagram of a computer system

Description automatically generated

Figure 4: Architectural diagram 1.0

Figure 5 is the high-level overview. Data is first collected by an application such as insect counting (other data collecting applications are Lightning Monitoring, Temperature Monitoring and so on). The data then goes trough Digital Twin. As the data passes through Digital Twin the Monitoring Data Collectors collect data about the process as well as data about the condition of the hardware. The monitoring data is then stored and later visualized.

## 4.3 Development environment

To complete the development environment the environment described in the high-level overview was set up. At this point the project could collect basic metrics data from hardware and Apache Spark. A full description on how to set up the environment was set up in Appendix E: Configuration Environment. Note that this description was later updated after the initial completion of the environment.

# 5 Implementation

This section will cover in detail all the sprints of the implementation their results. As well as how the sprints where organized.

For keeping track of sprints and tasks Jira is used. First a list of all necessary sprints is planed based on the user stories. Then each sprint is split into a series of smaller tasks and executed.

To properly organize the sprints, we have meet with both the customer representative and project manager to determine the importance of each individual user story and used that to organize a series of sprints. The following sprints have been organized in order of execution:

|  |  |  |
| --- | --- | --- |
| ID | Name | Description |
| 1 | Hardware Monitoring | The objective of this sprint is the creation of a visual interface that can give the customer data about the current state of the servers the application is running on. |
| 2 | Error Monitoring | The objective of this sprint is the creation of a user interface that can be used to alert the user of any errors and or problems the system encounters when executing jobs. |
| 3 | Trace Monitoring | The objective of this sprint is creating an interface that shows the user traces of all data that is entered in Digital twin. |
| 4 | Management interface | The objective of this sprint is taking data presented in the other interfaces and using it to create an interface that can utilized by users that aren’t developers. |

Table 8: Sprint List

## 5.1 Sprint 1: Hardware monitoring

The objective of this sprint is the creation of a visual interface that can give the customer data about the current state of the servers the application is running on. More precisely information about the CPU, RAM and memory current usage of each device as well as these metrics over a period selected by the user.

To achieve the objective for this sprint the work was split the work into the following tasks:

1. Desing an interface for the hardware monitoring data.

2. Document the process in this document.

For this specific sprint setting up the database and the monitoring of the hardware was unnecessary as these steps were completed during the development environment.

A screenshot of a computer

Description automatically generated

Figure : Hardware monitoring mockup

To design the interface a mockup was first created. This can be seen in figure 5. The tree or more traffic lights at the top represent different servers. They are red if the server is unavailable. They are yellow if the server is having over 80% CPU, RAM or Memory usage and green otherwise. Below we have a dropdown that selects which server the user wants to see in more detail. Below that we have the current RAM, CPU and Memory usage of the server as well as graphs showing these statistics over a period selected by the user in another dropdown.

A screenshot of a computer

Description automatically generated

Figure : Hardware monitoring dashboard

Using the initial mockup and Grafana we have created a functional web dashboard as seen in Figure 6. Grafana features have been used to replace the traffic lights with server images and the loading bars with speedometers to make the image clearer.

This sprint was ended with a fully functional hardware monitoring dashboard that was approved by the customer and the project manager.

## 5.2 Sprint 2: Error monitoring

The objective of this sprint is the creation of a user interface that can be used to alert the user of any errors and or problems the system encounters when executing jobs. For now, this is done by only monitoring the errors and warnings of Apace Spark ass it is the only application that is ready to be monitored.

In the initial planning for this sprint, it was discovered that Prometheus, the database that was being used for the collection of hardware monitoring data is extremely inadequate for storing error and warning information from logs. As a result, it was decided to utilize Loki a database designed by the Grafana team specifically for log retention. This specific tool was chosen for the following reasons:

* It’s relation with Grafana makes it very easy to use with it guaranteeing that we can display the information quickly and effectively.
* It’s molded after Prometheus making it very easy to use and not requiring the developers to learn much new information to learn how to use it.
* It comes with Promtail an application designed to extract log data from a file or folder a submit it to Loki. This makes it very easy to extract data from Apache Spark.

A diagram of a software system

Description automatically generated

Figure 7: Architectural diagram 2.0

Due to the addition of Loki our initial high-level diagram was updated as seen in figure 7. The first change is that Prometheus only needing to collect metrics can collect data directly from the generic interface and storage and processing. Instead of using a java application we use can use Promtail to collect data and transmit it to Loki. All this data is then transferred to the Grafana visualization.

To complete this sprint, we have divided the work into multiple smaller steps:

1. Configure Apache Spark to write the logs to a rolling file so it can be read by Promtail.
2. Set up Promtail to collect data from Apache Spark.
3. Set up Prometheus to collect data.
4. Connect Grafana with Prometheus.
5. Design a dashboard showcasing both the collected errors and warnings.

A screenshot of a computer

Description automatically generated

Figure 8: Error monitoring mockup

To create a dashboard showcasing the logs a mockup has been created as seen in figure 8. The interface shows a button used to select the time to in which the errors and warning where produced. Besides that, there are for fields two showing the warnings and logs of Apache Spark master and two showing the logs of the Apache Spark worker. This was done since the master and worker are different components of Apache spark with different roles so mixing up the logs would have created confusion for the user.

A screenshot of a computer program

Description automatically generated

Figure 9: Error monitoring interface

Figure 9 shows a picture of the fully working log monitoring dashboard.

This sprint all our goals have been accomplished. A log viewing interface has been created for digital twin. Of note is the fact that monitoring for the general interface has not been set up and the developers of the Monitoring system are waiting for a meeting with the general interface team to establish how monitoring of the general interface can be done efficiently.

One problem that we did encounter was that for Loki to properly store its data the container running it has been given root level access which poses a small security risk. This is planned to be remedied in a future sprint.

The work of this sprint has been approved by both the project manager and the customer representative.

## Sprint 3: Trace monitoring

The objective of this sprint is creating a user interface capable of showing the user distributed traces.

The first step of this sprint is to determine if monitoring distributed traces is possible for Apache Spark without modifying application. To answer the question the following where researched: Apache spark and its monitoring features and other Apache Spark monitoring systems and how they monitor distributed traces. The result of this research is the fact that without instrumenting the application (which will modify it substantially) it is not possible to monitor the distributed traces of the application.

The closest thing that can be done is show the duration of each spark job trough monitoring logs. This was done by modifying our error monitoring interface to also monitor the completion of jobs and their duration. An image of the updated UI can be seen in Figure 10.

A screenshot of a computer program

Description automatically generated

Figure 10:Updated Error Monitoring with Trace Monitoring

The implementation of this feature took significantly less time than expected. To not waste time, the developer has also implemented two other features:

* Configuring both databases (Loki and Prometheus) to automatically delete data older than 4 weeks.
* Linking the two dashboards so that you can easily go from one to the other in a single click.

This sprint did not fully accomplish its goals and but managed to implement other features. The work was aproved by the customer representative and the project manager who agreed this it was more important to not modify digital twin than have a comprehensive tracing system.

# 6 Conclusion

This project has at the time of writhing this document managed to implement most of the features outlined in the user stories. The completed user stories are 2,3,4,5,7 and the incomplete user stories are 1,6.

Currently the Digital Twin Monitoring System can:

* Monitor metric and log data from the storage and processing part of Digital Twin (it monitors Apache Spark). (necessary for all user stories)
* Show information on the state of the used hardware trough a web interface. (U.S. 4,7)
* Show information on the current running of the Digital Twin application including errors, warnings and data on the time it took for the application to complete certain tasks.

(U.S. 5,7)

* Show collected data up to 4 weeks old and delete data older than that. (U.S. 2)
* Show any changes to the system within 1 minute of them happening. (U.S. 3)

The monitoring system is currently unable to:

* Show full traces of the data as it passes through Digital Twin (U.S. 1). It is unable to do this without modifying the Digital Twin application, which is something that should be avoided at this stage in development.
* Display the information through a Management Interface in such a way that it can be understood by those without significant knowledge. (U.S. 6) This feature was not completed due to lack of time.
* Monitor the general interface of Digital Twin in any way. (necessary in all user stories). This feature could not be completed due to lack of communication between the developers of this project and the developers of the general interface.

As a hole the project is in a functional but unfinished state. The current version of this project has been deemed satisfactory by the project manager and customer representative.

Below is a list of recommendations on what should be done next to improve the Monitoring System:

* Find a way to remove root level access from the Loki docker image
* Implement monitoring for the general interface
* Rework the dashboards to offer the user more data and be easier to use.
* Implement the monitoring of distributed traces by modifying Digital Twin.
* Design a management interface with that makes it possible for users without technical knowledge to monitor the application.
* Give the interface a web address so that it can be easily accessed through the internet.

# References

Resources used in the report:

1. StackShare: <https://stackshare.io>
2. Slant: <https://www.slant.co>
3. Libhunt: <https://www.libhunt.com>
4. Prometheus: <https://prometheus.io>
5. Prometheus node exporter: <https://prometheus.io/docs/guides/node-exporter/>
6. Grafana: <https://grafana.com/docs/grafana/latest/>
7. Tutorial used for setting up node exporter on mac: <https://medium.com/@mishra.anshuman6/how-node-exporter-of-prometheus-works-on-macos-servers-ace131d51d56>
8. Used as reference to set up Prometheus: <https://grafana.com/docs/grafana-cloud/send-data/metrics/metrics-prometheus/prometheus-config-examples/docker-compose-linux/>
9. Used to run Grafana: <https://grafana.com/docs/grafana/latest/setup-grafana/installation/docker/> and https://grafana.com/tutorials/
10. Used for Loki: https://grafana.com/docs/loki/latest/setup/install/docker/

# Appendix

## Appendix A: Interview Questions

### First interview (Cristiane Salz)

* How does Insect Counting transfer data to Digital Twin?
* What data does Insect Counting transfer to Digital Twin?
* How often does Insect Counting transfer data to Digital Twin?

### Second interview (Tim Riebner)

* What kinds of information does the monitoring system need to collect about the operation of digital twin.
* Does the monitoring system also need to monitor the state of the hardware Digital Twin is running on.
* Should the monitoring system update the information shown in real time or only periodically.
* What types of users access the monitoring system and what can each of them see.
* How will the monitoring system display information to users?
* How long should the monitoring information be saved for?

### Third interview (Compass Agro)

* What does Compass Agro do?
* How will digital twin be used to help Compass Agro?
* Will Compass Agro directly use the monitoring system?

## Appendix B: Tool Selection Criteria

### General Criteria (Applies to all areas)

**Name:** Ecosystem

**Explanation:** The more widely used a tool is the easier it is to find information on how to use it such as tutorial or blogs and the easier it is to solve problems since there is a higher chance that other people have had a similar problem and solved it. For this criterion we will prioritize projects that have been popular recently since this reduces the chance that the tool is outdated.

**How this criterion is measured:** This criteria’s score will be generated using two metrics the number of stars on GitHub and the activity score on LibHunt.

The number of stars on GitHub is a good indicator of how widely popular a project is. For this criterion the points will be added in the following way:

* 5 points 40.000+ stars
* 4 points 30.000-40.000 stars
* 3 points 20.000-30.000 stars
* 2 points 10.000-20.000 stars
* 1 point 5.000-10.000 stars
* 0 points 0-5.000 stars

The lib hunt activity score is based on recent metrics such as commit frequency, issue and pull request activity and community engagement (number of discussion and responses to issues). This makes the LibHunt activity score a good indicator of the how currently popular and up to date the tool is. The points for this criterion from this metric will be added in the following way:

* 5 points 9-10 score
* 4 points 7-9 score
* 3 points 5-7 score
* 2 points 3-5 score
* 1 point 1-3 score
* 0 points 0-1 score

The points of this metrics are added together to form the final score of the criteria which will be between 0-10.

**Name:** Free of cost

**Explanation:** The company does not want to pay for any additional tools that will be using in the creation of the monitoring application as a result we will only be using free and open-source options. This is a **knockout criterion**.

**How this criterion is measured:** True or false. False if it cost money to use.

**Name:** Compatibility

**Explanation:** This criterion is meant to represent if the tool selected works well with other tools that will be used. We define “works well” as having features or plugins that make it easier to use alongside other tools in the creation of the application.

**How this criterion is measured:** 3 points will be assigned for each other tool the selected tool “works well” with. These points will only be added if the tool it works well with another tool that is utilized.

**Name:** Wildcard

**Explanation:** This is meant to represent any unique features or problems a tool has that are not covered by other criteria.

**How this criterion is measured:** A score from -3 to +3 point depending on the specific tool.

### Monitoring data storage

**Name:** 4-week storage

**Explanation:** The data storage must be capable of storing only the last 4 weeks of data and automatically deleting older data as to not occupy too much space. Is a **knockout criterion**.

**How this criterion is measured:** True or false. False if it can’t automatically delete data older than 4 weeks.

**Name:** Time series

**Explanation:** For this project we require any database used to be a time series database. This is because time series databases make it easier to keep track of the timing of different metrics making the excellent for a monitoring solution. This is a **knockout criterion**.

**How this criterion is measured:** True or false. True if the tool is a time series database.

**Name:**  Speed

**Explanation:** Because the database will not be handling particularly large amounts of data and the fact that measuring the exact performance for this project will be very difficult it was decided to not use speed as a criterion for this project.

**How this criterion is measured:** Not applicable

### Hardware monitoring

**Name:**  Basic Hardware Monitoring

**Explanation:** The tool used must be able to monitor the remaining RAM, memory and the CPU usage and transmit it to a time series database. This is a **knockout criterion**.

**How this criterion is measured:** True or false. True if it can monitor all tree metrics.

### Monitoring visualization

**Name:**  Web Display

**Explanation:** For ease of use the tool must be able to display the information through a web interface to make it easy to access for the customers. This is a knockout criterion.

**How this criterion is measured:** True or false. True if it can display the information rough a web interface.

**Name:**  Real Time Display

**Explanation:** To quickly address problems, the visual interface must be able to change in real time to reflect changes in the monitored application. This is a **knockout criterion**.

**How this criterion is measured:** True or false. True if it can modify itself in real time.

**Name:**  Display Features

**Explanation:** The tool should have the features necessary to display the data in concise and easy to understand way.

**How this criterion is measured:** This criterion is measured based on the available feature of the tools. Each helpful feature will have several points assigned to it and the final score will be made of the sum of the features the tool has:

* Can display pop ups - 1 point
* Can display text messages – 3 points
* Can switch between multiple pages – 1 point
* Can display traces – 3 points
* Can display graphs – 2 points

## Appendix C: Unchosen Tools

Below we have a list of tools we have not chosen and one knockout criterion they failed

### Unchosen monitoring data storage

|  |  |
| --- | --- |
| Tool | Failed knockout criteria |
| PostgresSQL | Time series |
| Oracle | Time series |
| MySQL | Time series |
| Cockroach DB | Time series |
| VoltDB | Time series |
| YugabyteDB | Time series |
| CouchDB | Time series |
| MongoDB | Time series |
| Dgraph | Time series |
| Neo4j | Time series |
| ArgoDB | Time series |
| Memgraph | Time series |
| Pinecone | Time series |
| Milvus | Time series |
| DolphinDB | Free to use |
| Redis | Time series |
| Memcached | Time series |
| Amazon Dinamo Db | Time series |
| Key DB | Time series |
| ArangoDB | Time series |
| Fauna | Time series |
| SurealDB | Time series |

Table : Unselected data storage tools

## Unchosen hardware monitoring tools

|  |  |
| --- | --- |
| Tool | Failed knockout criteria |
| Checkmk | Free to use |
| Splunk enterprise | Free to use |
| Zabix | Free to use |
| PRTG network monitor | Free to use |
| HWmonitro | Free to use |

Table : Unselected hardware monitoring tools

## Unchosen monitoring visualization

|  |  |
| --- | --- |
| Tool | Failed knockout criteria |
| Datadog | Free to use |
| Tableau | Free to use |
| Power BI | Free to use |
| Looker | Free to use |
| Mode analytics | Free to use |
| Qlik sense | Free to use |
| Zoho analytics | Free to use |
| GoodData | Free to use |
| Domo | Free to use |

Table : Unselected visualization tools

## Appendix D: Score Explanations

### Monitoring Data Storage

**Tool Name:** Prometheus

**Good ecosystem:** 5 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Grafana and 3 additional for working with Prometheus node explorer for a total of 6 points if both tools are picked.

**Wildcard:** 2 points thanks to the fact that Prometheus is considered easier to use than its alternatives

**Tool Name:** Timescale DB

**Good ecosystem:** 2 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Telegraph and 3 points for working with Zabbix because Telegraph and Zabbix cannot be both selected Timescale DB receives the only tree points.

**Wildcard:** 1 point for the fact that Timescale DB uses the Postgres SQL as it’s query language that the developer of this project is already familiar with.

**Tool Name:** Quest DB

**Good ecosystem:** 2 points for GitHub stars and 5 points for activity score

**Works with other tools:** does not work particularly well with other tools

**Wildcard:** Nothing notable

**Tool Name:** Influx DB

**Good ecosystem:** 3 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for telegraph

**Wildcard:** Nothing notable

### Hardware Monitoring

**Name:** Prometheus Node Exporter

**Good ecosystem:** 2 points for GitHub stars and 4 points for activity score

**Works with other tools:** 3 points for working with Prometheus

**Wildcard:** Nothing notable

**Name:** Telegraph

**Good ecosystem:** 2 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Timescale DB

**Wildcard:** Nothing notable

**Name:** Zabbix

**Good ecosystem:** 1 point for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Timescale DB

**Wildcard:** Nothing notable

### Monitoring Visualization

**Name:** Grafana

**Good ecosystem:** 5 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Prometheus

**Display features:** 10 points for having all display features

**Wildcard:** 2 points thanks to the fact that Grafana is considered easier to use than its alternatives

**Name:** Apache Superset

**Good ecosystem:**5 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Timescale DB

**Display features:** 7 points for all display features except trace display

**Wildcard:** Nothing notable

**Name:** Metabase

**Good ecosystem:** 4 points for GitHub stars and 5 points for activity score

**Works with other tools:** 3 points for working with Timescale DB

**Display features:** 6 points for all display features except trace display and pop-ups

**Wildcard:** Nothing notable

## Appendix E: Configuration Environment

The purpose of this document is to provide a step by step on how to set up and configure the environment of the Digital Twin Monitoring Application on the mac operating system. We will go over each tool used and how to set it up. It is highly recommended that the tools are installed in the below order. This document had to be modified multiple times after the initial environment set up to accommodate all the features of our application.

This configuration tutorial assumes that the user has docker desktop installed.

# Prometheus node exporter

The Prometheus Node Exporter has the role of collecting data about the computers operation and then sending it to Prometheus. It needs to be installed and configured on each computer that needs to be monitored.

Below are the steps need it to install node exporter on a mac.

1. Go to <https://github.com/prometheus/node_exporter/releases> and download the newest release of the darwin-amd64 version then extract it using this command:

tar -xvf node\_exporter-{version}.darwin-amd64.tar.gz

1. Go into the extracted folder using a terminal and then move the node explorer executable to /usr/local/bin/ using the following command:

sudo mv node\_exporter /usr/local/bin/

1. Node explorer can now be started. But it won’t automatically start when the computer starts. To do that it is necessary to make a **launchd** configuration.
2. Open a new terminal type the following command twice:

cd ..

1. Create a file in Library/LaunchDaemons/ called node\_exporter.plist using the following command:

sudo nano Library/LaunchDaemons/node\_exporter.plist

1. Give the file permissions using:

chmod 777 node\_exporter

1. Open the newly created file with a text editor and paste the following:

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
<dict>  
 <key>KeepAlive</key>  
 <true/>  
 <key>Label</key>  
 <string>node\_exporter</string>  
 <key>ProgramArguments</key>  
 <array>  
 <string>/usr/local/bin/node\_exporter</string>  
 </array>  
 <key>RunAtLoad</key>  
 <true/>  
</dict>  
</plist>

1. Run the following command to start node exporter:

sudo launchctl load /Library/LaunchDaemons/node\_exporter.plist

1. If an error due to permissions on file is shown, run the following commands to provision permissions on the file.

chmod 600 /Library/LaunchDaemons/node\_exporter.plist  
  
sudo chown root /Library/LaunchDaemons/node\_exporter.plist  
  
sudo chgrp wheel /Library/LaunchDaemons/node\_exporter.plist

1. After grating the permission run the command again to start node exporter.
2. Check if its functioning by going to <http://localhost:9100/metrics>

# Apache spark

Apache spark is one of the applications we must monitor. How to set up the application properly will be covered in the configuration document made by Razvan Budurovici. This will only detail additional steps that are needed to be taken after the installation to ensure data can be collected from Apache Spark.

## Configuration for Loki

1. When creating spark-master use the following command to mount a container to spark-master.

docker run -d --name spark-master --network spark-network --mount type=volume,source=promatail\_spark-master\_logs,target=/opt/bitnami/spark/logs -h spark-master -p 8080:8080 -p 7077:7077 bitnami/spark:latest /opt/bitnami/spark/bin/spark-class org.apache.spark.deploy.master.Master

1. Do the same for the spark worker.

docker run -d --name spark-worker --network spark-network --mount type=volume,source=promatail\_spark-worker\_logs,target=/opt/bitnami/spark/logs -h spark-worker -p 8081:8081 bitnami/spark:latest /opt/bitnami/spark/bin/spark-class org.apache.spark.deploy.worker.Worker spark://spark-master:7077

1. If the containers for Apache Spark are already started delete it and remake it using the previous command or mount the container in a different way.
2. Create a file called log4j2.propreties. Using a text editor (such as nano). Add the following to the file:

rootLogger.level = info

rootLogger.appenderRefs = console, file

rootLogger.appenderRef.console.ref = ConsoleAppender

rootLogger.appenderRef.file.ref = RollingFile

appender.console.type = Console

appender.console.name = ConsoleAppender

appender.console.layout.type = PatternLayout

appender.console.layout.pattern = %-5p %c:%L - %m%n

appender.RollingFile.type = RollingFile

appender.RollingFile.name = RollingFile

appender.RollingFile.fileName = /opt/bitnami/spark/logs/spark.log

appender.RollingFile.filePattern = /opt/bitnami/spark/logs/spark-%d{yyyy-MM-dd-HH}.log

appender.RollingFile.layout.type = PatternLayout

appender.RollingFile.layout.pattern = %-5p %c:%L - %m%n

appender.RollingFile.policies.type = Policies

appender.RollingFile.policies.size.type = SizeBasedTriggeringPolicy

appender.RollingFile.policies.size.size = 10MB

appender.RollingFile.strategy.type = DefaultRolloverStrategy

appender.RollingFile.strategy.max = 10

1. Go into a terminal and go into the folder with the log4j2.propreties file.
2. Use the following commands to copy the file to Apache spark worker and master:

docker cp log4j2.properties spark-master:/opt/bitnami/spark/conf/

docker cp log4j2.properties spark-worker:/opt/bitnami/spark/conf/

1. Restart Apache Spark to update the changes.

## Configuration for Prometheus

1. Create a file called metrics.propreties. Using a text editor (such as nano). Add the following to the file:

\*.sink.prometheusServlet.class=org.apache.spark.metrics.sink.PrometheusServlet

\*.sink.prometheusServlet.path=/metrics

1. Go into a terminal and go into the folder with the metrics.propreties file.
2. Use the following commands to copy the file to Apache spark master and worker:

docker cp metrics.properties spark-master:/opt/bitnami/spark/conf/

docker cp metrics.properties spark-worker:/opt/bitnami/spark/conf/

1. Restart Apache Spark to update the changes.

# Prometheus

Prometheus role is to store metrics data from multiple sources so it can be further used for different sources.

Below are the steps need it to install node exporter on a mac:

1. Create a folder for storing docker compose files this will be used to store the files well be using to run Prometheus and other aps in docker.
2. Create a folder in the original storage folder called Prometheus or another descriptive name.
3. Create a file in the Prometheus folder create a file called docker-compose.yml . The easiest way to do this is by opening a terminal going to the Prometheus folder and then running the following command:

nano docker-compose.yml

1. After running the command paste the following into nano to write into the file:

services:

prometheus:

image: prom/prometheus

container\_name: prometheus

command:

- '--config.file=/etc/prometheus/prometheus-config.yml'

- '--storage.tsdb.retention.time=4w'

ports:

- 9090:9090

restart: unless-stopped

volumes:

- ./prometheus-config.yml:/etc/prometheus/prometheus-config.yml

- prom\_data:/prometheus

volumes:

prom\_data:

1. file by using control O and then exit with control X.
2. Create a file called prometheus-config.yml in the Prometheus folder. This can be done with the command:

nano prometheus-config.yml

1. After running the command paste the following into nano to write into the file:

global:

scrape\_interval: 15s

scrape\_timeout: 10s

evaluation\_interval: 15s

alerting:

alertmanagers:

- static\_configs:

- targets: []

scheme: http

timeout: 10s

api\_version: v2

scrape\_configs:

- job\_name: prometheus

honor\_timestamps: true

scrape\_interval: 15s

scrape\_timeout: 10s

metrics\_path: /metrics

scheme: http

static\_configs:

- targets:

- localhost:9090

- job\_name: hardwareMonitoring

static\_configs:

- targets: ['host.docker.internal:9100']

- targets: ['145.93.88.194:9100']

- job\_name: spark

metrics\_path: /metrics

static\_configs:

- targets: ['host.docker.internal:8080']

1. Save the changes to the file by using control O and then exit with control X.
2. Use the terminal to go into the file with the folder with the docker-compose.yml file and run the flowing command to start Prometheus:

docker compose up -d

1. To verify that Prometheus is running go to <http://localhost:9090/> there should be an Ui allowing you to run queries and check the connection with hardware monitoring.

# Loki+Promtail

Promtail is used to collect logs from Apache spark and send them to Loki while Loki is used to store logs and send them to Grafana.

1. Create a folder for storing docker compose files this will be used to store the files well be using to run Prometheus and other aps in docker.
2. Create a folder in the original storage folder called Loki+Promtail or another descriptive name.
3. Create a file in the Prometheus folder create a file called docker-compose.yml . The easiest way to do this is by opening a terminal going to the Prometheus folder and then running the following command:

nano docker-compose.yml

1. After running the command paste the following into nano to write into the file:

services:

loki:

image: grafana/loki

container\_name: loki

user: "root"

volumes:

- ./loki-config.yaml:/mnt/config/loki-config.yaml

- loki\_data:/loki/data

ports:

- "3100:3100"

command:

- '--config.file=/mnt/config/loki-config.yaml'

promtail:

image: grafana/promtail

container\_name: promtail

volumes:

- ./promtail-config.yaml:/mnt/config/promtail-config.yaml

- promatail\_spark-master\_logs:/monitor\_logs/spark-master/

- promatail\_spark-worker\_logs:/monitor\_logs/spark-worker/

depends\_on:

- loki

command: -config.file=/mnt/config/promtail-config.yaml

volumes:

promatail\_spark-master\_logs:

external: true

promatail\_spark-worker\_logs:

external: true

loki\_data:

1. Save the changes to the file by using control O and then exit with control X.
2. Create a file called promtail-config.yaml in the Loki+Promtail folder. This can be done with the command:

nano promtail-config.yaml

1. After running the command paste the following into nano to write into the file:

server:

http\_listen\_port: 9080

grpc\_listen\_port: 0

positions:

filename: /tmp/positions.yaml

clients:

- url: http://loki:3100/loki/api/v1/push

scrape\_configs:

- job\_name: system

static\_configs:

- targets:

- localhost

labels:

job: varlogs

\_\_path\_\_: /var/log/\*log

- targets:

- localhost

labels:

job: spark-master

\_\_path\_\_: /monitor\_logs/spark-master/\*log

- targets:

- localhost

labels:

job: spark-worker

\_\_path\_\_: /monitor\_logs/spark-worker/\*log

1. Save the changes to the file by using control O and then exit with control X.
2. Create a file called loki-config.yaml in the Loki+Promtail folder. This can be done with the command:

nano loki-config.yaml

1. After running the command paste the following into nano to write into the file:

auth\_enabled: false

server:

http\_listen\_port: 3100

common:

ring:

instance\_addr: 127.0.0.1

kvstore:

store: inmemory

replication\_factor: 1

path\_prefix: /loki

schema\_config:

configs:

- from: "2023-01-05"

index:

period: 24h

prefix: index\_

object\_store: filesystem

schema: v13

store: tsdb

storage\_config:

tsdb\_shipper:

active\_index\_directory: /loki/data/tsdb-index

cache\_location: /loki/data/tsdb-cache

query\_scheduler:

max\_outstanding\_requests\_per\_tenant: 32768

querier:

max\_concurrent: 16

compactor:

working\_directory: /data/retention

compaction\_interval: 10m

retention\_enabled: true

retention\_delete\_delay: 2h

retention\_delete\_worker\_count: 150

delete\_request\_store: filesystem

limits\_config:

retention\_period: 672h

1. Save the changes to the file by using control O and then exit with control X.
2. Use the terminal to go into the file with the folder with the docker-compose.yml file and run the flowing command to start Prometheus:

docker compose up -d

1. To verify that Loki is running go to <http://localhost:3100/>ready there should be a website with a message.

# Grafana

Grafana is used to properly display our gathered monitoring data through a web interface.

Below are the steps required to start Grafana on a mac:

1. In the Docker folder created when installing Prometheus create a folder called Grafana (this is to make organizing easier).
2. In the Grafana folder create a file called docker-compose.yml. The easiest way to do this is by opening a terminal going to the Prometheus folder and then running the following command:

nano docker-compose.yml

1. After running the command paste the following into the nano terminal:

services:

grafana:

image: grafana/grafana

container\_name: grafana

restart: unless-stopped

ports:

- '3000:3000'

volumes:

- grafana-storage:/var/lib/grafana

volumes:

grafana-storage: {}

1. Save the changes to the file using control X.
2. In the Grafana folder run the following command to start Grafana:

docker compose up -d

1. To verify that Grafana go to <http://localhost:3000/>. Input the username admin and password admin to access Grafana.
2. To connect Prometheus to Grafana go to data sources > Add new data source select Prometheus and then in connection write:

http://host.docker.internal:9090

1. Save and test the connection.
2. To connect Loki to Grafana go to data sources > Add new data source select Loki and then in connection write:

http://host.docker.internal:3100

1. Save and test the connection.