INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Leveraging Data Monitoring to Aid Decision-Making in Large Scale Continuous Integration Systems

DIOGO ANTÓNIO FERREIRA TERRANTEZ
outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Leveraging Data Monitoring to Aid Decision-Making in Large Scale Continuous Integration Systems

Diogo Terrantez

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Dr. Nuno Bettencourt

Porto, October 17, 2022

Abstract

Following the Android Open Source Project (AOSP) initiative, Bayerische Motoren Werke GmbH (BMW) has started developing the head units for their latest generation vehicles with an Android-based operating system. While providing multiple benefits, it brought about a new development methodology much different than developing for AUTOSAR, the industry standard and previous methodology. To accommodate this new methodology, a new and highly complex Continuous Integration (CI) infrastructure was designed from the ground up.

Managing such an infrastructure has proved arduous and existing monitoring solutions failed to meet the project's demands. As such, a need has emerged for a new, more complete monitoring solution, that can accurately measure and provide insight into the state of this infrastructure.

This work aims to provide such monitoring system that, by aggregating relevant metrics, is able to formulate meaningful KPIs that may be leveraged during the decision-making process for infrastructure maintenance and future project growth.

The research section for this work consists in the investigation on how to best define metrics and KPIs, and relevant monitoring tools and other technologies, that may be used to deploy and configure the required monitoring solution. Besides this investigation, A literature review on relevant topics and collaborator interviews were conducted in order to identify problems with the CI infrastructure and metrics that should be measured as part of the solution.

A proof-of-concept was developed as a way to verify the solution's viability, resulting in the implementation of some metrics and definition of KPIs. A possible AWS (Amazon Web Services) deployment was also entertained.

Results from the implementation suggest that the problem can be solved trough the implemented solution, though some limitations afflicted the development workflow. These are, however, already being addressed and should not pose a problem in the future.

The main contributions from the work revolve around features implemented for the monitoring solution, which are already improving daily workflow for a number of collaborators.

Keywords: Continuous Integration, Metrics, KPIs, Data Monitoring

Resumo

O grupo BMW (Bayerische Motoren Werke GmbH), seguindo a iniciativa AOSP (Android Open Source Project), começõu o desenvolvimento das "head units" para os seus veículos de nova geração, com um novo sistema operativo baseado em Android. Esta mudança, apesar de trazer vários benefícios, acarreta também nova metodologia de desenvolvimento, a qual difere bastante da préviamente usada e padrão industrial, AUTOSAR. Para acomodar esta nova tecnologia, um novo sistema de infraestrutura foi projetado desde raíz.

Gerir uma infraestrutura deste calibre provou ser uma tarefa árdua e os sistemas de monitorização existentes não foram capazes de atender a todas as necessidades do projeto. Desta forma, surgiu a necessidade para uma nova solução de monitorização completa, capaz de, com precisão, medir e informar do estado desta infraestrutura.

Este trabalho visa fornecer tal sistema que, através da agregação de méticas relevantes, é capaz de formular KPIs com significância, que possam ser aproveitados aquando o processo de decisão para manutenção da infraestrutura e crescimento futuro do projeto.

A fase de pesquisa para este trabalho consiste na investigação em como melhor definir metricas e KPIs, e outras ferramentas de monitorização e tecnologias relevantes, que possam ser utilizadas para implantar e configurar a solução necessária. Além desta investigação, foi realizada uma revisão da literatura sobre temas relevantes, e entrevistas a colaboradores de modo a identificar problemas com a infraestrutura, e definir métricas que devem ser medidas como parte da solução.

Foi desenvolvida uma prova de conceito como forma de verificar a viabilidade da solução, resultando na implementação de algumas métricas e definição de KPIs. Foi também considerada uma possível implantação na núvem da AWS (Amazon Web Services).

Os resultados da implementação sugerem que o problema pode ser resolvido através da solução implementada, embora algumas limitações tenham afligido o fluxo do desenvolvimento do trabalho. No entanto, estas já estão a ser abordadas e não deverão originar problemas no futuro.

As principais contribuições do trabalho giram em torno das funcionalidades implementadas para a solução de monitorização, que, neste momento, já melhoram o fluxo de trabalho diário de diversos colaboradores.

Acknowledgement

I would like to thank all who helped me trough this educational journey and gave me their time when I needed it.

Special thanks to my friends and colleagues that directly aided in the making of this document, as well as my professor and supervisor, Dr. Nuno Bettencourt that supported me trough my lack of communication and timely deliveries.

The biggest thanks of all go to my family, for worrying about me and providing all the support and motivation I needed to complete my education up to this point.

Contents

List of Figures

List of Tables

List of Source Code

List of Acronyms

AHP	Analytic Hierarchy Process.
AMI	Amazon Machine Image.
AOSP	Android Open-Source Project.
API	Application Programming Interface.
APM	Application performance monitoring.
APT	Advanced Package Tool.
ARN	Amazon Resource Name.
AWS	Amazon Web Services.
BMW	Bayerische Motoren Werke GmbH.
CI	Continuous Integration.
CLI	Command Line Interface.
DSRM	Design Science Research Methodology.
EC2	Elastic Compute Cloud.
FFE	Fuzzy Front End.
IaC	Infrastructure as Code.
IP	Internet Protocol.
MQTT	MQ Telemetry Transport.
NCD	New Concept Developmen.
QEF	Quality Evaluation Framework.
SAVE	Society of American Value Engineers.
SDKs	Software Development Kits.
SMTP	Simple Mail Transfer Protocol.
SSH	Secure Shell.
URL	Uniform Resource Locators.
VPC	Virtual Private Cloud.
VPN	Virtual Private Network.
WSL2	Windows Subsystem for Linux.

Chapter 1

Introduction

A car is much more than a machine which takes someone from point A to point B. Automotive vehicles are turning into computers, with capabilities akin to those of a smartphone. As more of the vehicle functions (*e.g.*, acceleration and deceleration control, autonomous driving, *etc.*.) become digitized, the reliability of the control system and the enjoyment provided by the infotainment system have an ever increasing weight in the retail value perceived by the costumer. Customers have come to expect an increasingly better experience from their multiple thousand euros computer and demand faster and more frequent update cycles with new features and improvements.

Unlike Tesla, conventional OEMs are not used to these demands and conventional automotive software platforms (*i.e.*, AUTOSAR), struggle to respond to customer needs in a timely manner (Martinez-Fernandez et al. 2015). To tackle this problem, Bayerische Motoren Werke GmbH (BMW) is implementing a new, Android-based operating system that more easily fulfils their customer's expectations, such as, faster and more sophisticated integrations, and reduced turnaround time. Moreover, the Android Open-Source Project (AOSP), which is being used as the new development platform, already includes standardized tools that are on par with modern software development (Al-Ani 2012).

BMW has only recently started development with AOSP and, similarly to the rest of automotive industry, is not yet used to the aforementioned platform. For instance, traditional AUTOSAR-based systems occupy only a few Megabytes in size while a full Android-based system requires almost a dozen of Gigabytes. This brings new challenges to the table as the complexity of the software increases rapidly and in great magnitude.

1.1 Problem

The practice of Continuous Integration (CI), although an old concept, has only recently started to gain mass adoption and still has a long ways to go in order to reach a level of standardization enough to apply it to most projects.

In a large scale, complex project, that is the development of a car's infotainment system, the hurdles of building and maintaining a proper CI infrastructure, grow to a higher order of magnitude. The time required to merge changes into the code base, the size of said changes, lengthy review process, long build and test pipelines execution, *etc.*, are just a few of the challenges that apply to this infrastructure.

Another issue that emanates from such a complex infrastructure, the one of interest to this work, is setting up proper monitoring in order to understand the state of said infrastructure, and provide the necessary information to leverage data-driven decisions.

1.2 Goals

The scope of this work consists in evaluating how the BMW Group in its next-generation Android-based head-unit (infotainment system) is addressing these points and propose not only metrics and KPIs[1] but also derive optimizations to the development/integration process upon such data.

The goal is to bring to light the status quo by understanding the most prominent pain points for developers and line managers (*i.e.*, problems with the CI infrastructure), and quantify said problems by developing a complete monitoring solution, capable of aggregating relevant metrics, to later formulate meaningful KPIs. This implementation must be scalable so it may eventually encompass every development team and project. With these curated KPIs, it will then be possible to make an objective analysis of the CI infrastructure and its problems and, aid/influence or even provide higher level, educated decision making, culminating in a more sustainable and optimized present and future development in the AOSP and other projects.

1.3 Research Questions

Considering the objectives defined in the previous section, it is possible to infer a set of research questions that, when analyzed and answered, will aid the construction of this document:

1. **RQ1** - Which metrics are most suitable to ascertain the state of a continuous integration infrastructure? (In the context of the BMW's new AOSP project)
2. **RQ2** - How can metrics and KPIs depict an accurate representation of the infrastructure's problems and weaknesses/vulnerabilities?
3. **RQ3** - How can we leverage data monitoring to improve the development of a complex system? (The AOSP CI infrastructure)

1.4 Hypotheses

As a means to answer the research questions and evaluate the result of this research, the following hypotheses will be analyzed and corroborated in the **Assessement** chapter:

1. **H1** - The infrastructure's state can be accurately represented by a set of metrics and KPIs.
2. **H2** - The defined metrics and KPIs can be leveraged when making data-driven decisions for the future of the project.

1.5 Research Methodology

The purpose of this dissertation is to tackle the problem by designing and implementing an appropriate solution. The research related to the metric management system was conducted following the Design Science Research Methodology (DSRM).

The DSRM methodology provides guidelines that direct the research in an efficient manner:

[1] KPI.org 2022.

1. **Problem Identification and Motivation**, providing a background for the work, addressed in the section 1.1.
2. **Definition of Objectives** to solve the problem, addressed in the section 1.2.
3. **Analysis and Design** of the solution, addressed in the chapters 3 and 4.
4. **Demonstration** of the implemented design alternative, addressed in the chapter 5.
5. **Evaluation** of the solution, addressed in the chapter 6.
6. **Communication** of the results, addressed in the chapter 7.

The other component for the research, required to define the metrics and KPIs that should be implemented, is mostly focused in the Survey research method, by conducting interviews with collaborators, as a way to identify the most prominent pain points that should be addressed. Besides the collaborator surveys, a small scale literature review took place, analysing relevant documents and academic papers, in order to infer other relevant metrics, KPIs, and good practices when implementing a monitoring solution.

1.6 Thesis Structure

This document is composed of 7 chapters.

The first chapter provides a context and describes the problem being addressed. Based on the problem, several objectives and research questions have been identified, and a research methodology defined.

The following chapter is dedicated to the **State of the art**, containing an overview and analysis of all the relevant theoretical topics, relevant work, and technologies.

The **value analysis** chapter is dedicated to studying the value of implementation alternatives for the solution and consists in applying various methodologies to ascertain said value.

The next chapter, **design** is divided into two sections. The first section, requirement analysis, establishes the requirements, functional and non-functional, to be fulfilled during the design and implementation of the solution. The second section is dedicated to the design of a possible solution to the problem, considering the requirements defined previously.

The **implementation** chapter describes the implementation process required to develop the monitoring system and metric/KPI gathering/calculation and shows evidence of the carried work.

The **experiments** chapter is comprised of tests and experiments that assess the outcome of the implementation and the result of the work.

Finally, the **conclusion** chapter summarizes the outcomes from the previous chapters and the work done as a whole, stating the contributions, limitations, and future work. It also contains the answer to the research questions established in the introduction, as well as some personal remarks.

Chapter 2

State Of The Art

This chapter's goal is to explain and analyze all relevant concepts related to this dissertation. It consists of five different sections:

1. **Theoretical Context** - Overview of the relevant theoretical concepts.
2. **Related work** - Literature review of related work.
3. **Thechnologies** - Technologies studied/used for this work.
4. **CI infrastructure** - Overview of the CI infrastructure in study.
5. **Methodologies** - Description of the methodologies applied to the thesis.

2.1 Theoretical Context

This section provides a brief overview and explanation of the theoretical concepts relevant for this work.

2.1.1 Data monitoring

A system's composers should place a high priority in ensuring its correct operation. Organizations should have instruments capable of analysing their systems' components, preferably in real time, so that faults may be identified and corrected as quickly as possible.

Defining data monitoring

Data monitoring, in its definition, is a business practice in which critical business data is routinely checked against quality control rules to ensure it is always of high quality and meets previously established standards for formatting and consistency (Informatica 2022).

In the context of this work, data monitoring is applied in monitoring a CI infrastructure, as a way to ascertain its state and characteristics.

Why practice data monitoring

The purpose of data monitoring is to understand a system's behavior, discover anomalies, and immediately notify relevant personnel should a failure occur. Ideally, it should also aid in preventing failures by analysing and reporting on anomalous trends that may eventually result in one.

Besides enabling operators to respond to undesirable degraded states of the system as a whole, it can also assist in identifying the root causes of persistent issues, and ensuring they do not reoccur in the future. As such, data monitoring is crucial when aiming for a robust, highly available system.

Data monitoring with software

Analysing large amounts of data in an analog fashion is time consuming or even practically impossible, hence the development of data monitoring software, which provide tools such as dashboards, alarms, and reports, that provide support or completely automate data analysis.

Monitoring and alerts

While monitoring systems are useful for active interpretation and investigation, the value proposition of a complete monitoring solution lies in its automation capabilities, allowing administrators to disengage from the system. Alerts allow for the definition of rules that, given certain (metric-based) conditions, automatically execute specified actions, while relying on the passive monitoring of the software to watch for these conditions (Ellingwood 2017).

The main purpose of alerting is bringing attention to the current status of the system. The alert should contain information about what is wrong and where to find additional information.

More complex infrastructures require the use of several severity levels, so that the responsible teams or individuals may be notified when appropriate. For instance, a decrease in network stability may warrant a work ticket, while a full shutdown requires an immediate response.

Visualizing data monitoring

Visualizations in a monitoring system provide a powerful, visual, analytic tool to ascertain the state and performance of the monitored system. These are commonly portrayed by dashboards, tables, charts, and graphs.

Similarly to alerts, visualizations are not a simple domain. As the consumer of visualizations, users can sometimes suffer from apophenia, a term coined by Klaus Conrad in his published monograph "The onset of schizophrenia: an attempt to form an analysis of delusion." Apophenia is where meaningful patterns are perceived from random data, which in turn can lead to sudden jumps from association to causation (Lightbend 2022).

Some of the key concerns that should be considered when implementing visualizations, are:

- Data must be clearly shown
- Graphs and such, should cause the viewer to think about substance not visuals
- Smoothing should not distort the data
- Data sets that include lots of data should be coherent
- Changing granularity should not impact comprehension

Approaching data monitoring

The first step to monitoring data is establishing data quality metrics or criteria, that are tied to specific business objectives. After establishing the groundwork, it becomes possible to compare the results over time, allowing for improvement and deeper understanding of how data can best be used (Experian 2022).

Concrete approaches to data monitoring are entertained in the **Technological** section.

2.1.2 Metrics and KPIs

Metrics and KPIs are important indicators that enable data monitoring and, if correctly defined, provide an actionable way to achieve overall business strategies and goals.

Defining Metrics and KPIs

Key Performance Indicators are the metrics by which business critical initiatives, objectives, or goals, can be gauged. The operative word in the phrase is "key", meaning they have special or significant meaning (Perez 2022). KPIs act as measurable benchmarks against defined goals.

As an example, if a business goal is to "increase sales by 15% over the next two quarters", the KPIs to gauge that may include, but are not be limited to: new customer acquisition, customer churn, and upselling success rate.

In short, **a KPI** can be made up of **multiple metrics**.

While KPIs measure progress toward specific goals, metrics are measurements of overall business health. Metrics may be loosely tied to specific targeted objectives, but are not, normally, the most appropriate guides to track said objectives.

For example, when tracking the number of website visitors, unless it is tied to a specific key business objective, it is a metric, not a KPI.

Figure 2.1 better depicts metrics and KPIs as a subset of *Data* as a hole. Additionally, a data point is a piece of information that describes one unit of observation, at one point in time, at the data collection level. It most commonly appears as one cell in a data table.

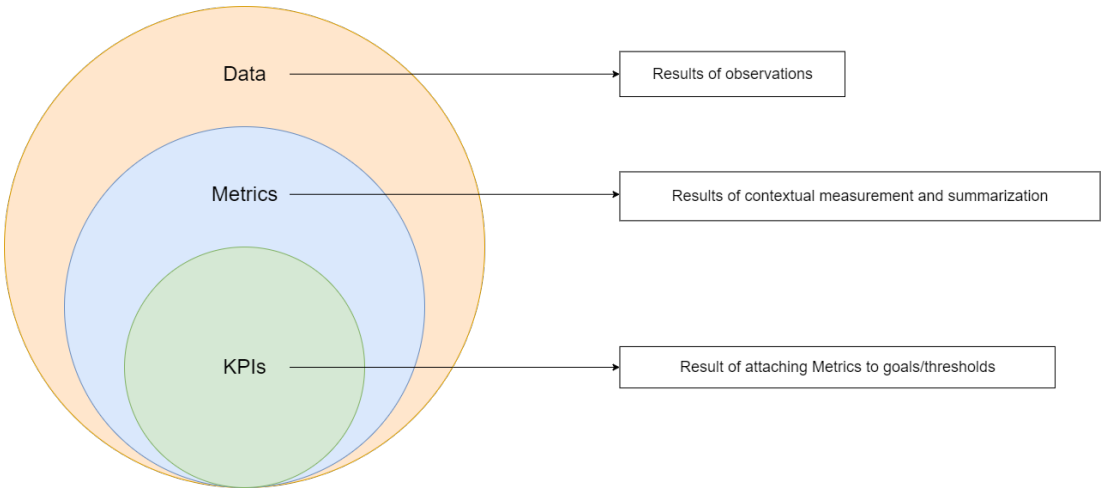

Figure 2.1: KPIs as a subset of Metrics and Data

Metrics, KPIs and data monitoring

Metrics and KPIs, specially the latter, are crucial indicators when monitoring a system since, as mentioned previously, enable the tracking of business objectives.

The above section, **Approaching data monitoring**, hints at the establishment of metrics and KPIs as the first step of monitoring data. That is so, because monitoring data is not possible without knowing what to look for. There is only meaning in gathering specific data when it can be converted into useful metrics/KPIs.

As such it can be said that data monitoring correlates directly to metric and KPI tracking, which in turn correlate to tracking business objectives.

Importance of monitoring metrics and KPIs

The importance of monitoring metrics and KPIs is much the same as the **importance of data monitoring** in itself. That is because of the correlation between the two (as stated above).

Generally, KPIs help project managers in (Stackify 2021):

- Controlling the project progress
- Reducing project cost
- Increasing the ROI (Return on investment)
- Correcting the distribution of tasks
- Managing workload

Moreover, tracking metrics and KPIs ensures that employees are aware of what is important to the business, by showing them what the business is being measured against.

Selecting the correct Metrics and KPIs to monitor

An effective way of choosing the right metrics and KPIs to implement, is by using the SMART framework (Doran 1981):

1. **Specific** - Goals must be specific so everyone understands what needs to be done. Unclear objectives lead to an unorganized development process with lots of unnecessary work and changes. A clear and definitive objective points the team to the right focal point, removing redundant processes during development.
2. **Measurable** - A KPI must give a clear expression of what solution or process needs to be measured. A comprehensive and descriptive performance measurement renders more accurate metric reports.
3. **Assignable** - A KPI should be assigned to a member or a team. Each KPI group should have "owners" to actively drive progress. It is best not to constantly change the owner of the KPI to avoid confusion and mismanagement.
4. **Realistic** - A KPI must be achievable and its evaluation should be realistic rather than idealistic. For example, if the ultimate goal is to climb Mount Everest, heading to Everest without prior climbing experience would be unrealistic, and failing to reach the top should not imply a complete failure.

5. **Time-bound** - A KPI must have a specific time-frame. A metric needs to have a defined tracking frequency. It should answer the question: "Is there a specific time-frame on when the goal can be achieved?"

There exist variants of this framework in which "**A**" stands for "Attainable" (similar to "Realistic") and "**R**" stands for "Relevant", meaning it should be relevant for the business.

2.1.3 Observability

Observability measures how well the internal states of a system can be inferred from knowledge of its external outputs.

An observability platform allows developers to simultaneously observe (or gain deeper insight into) the health and status of different applications and resources across the IT infrastructure. By garnering insights from each system's data, IT teams can proactively detect abnormalities, analyze issues, and resolve problems (Turner 2022).

2.1.4 Application performance monitoring

Application performance monitoring (APM) allows the tracking the performance of software applications to identify and drill down into issues that occur during development and runtime.

APM metrics revolve heavily around performance, measuring, for example, the number of transactions per second that the application processes and the total response time for each of those transactions. APM can also be used to measure the performance of devices executing those transactions as well as assessing hardware performance issues that may result in bottlenecks. APM tools are primarily designed to directly measure application performance on a granular level, identifying problematic dependencies in the infrastructure (Splunk 2022c).

Using APM solutions, it is possible to monitor whether the IT environment meets performance standards and identify bugs and potential issues before they cause an impact, or reduce recovery time should failures occur.

APM, Monitoring and Observability

The key task of DevOps teams is to ensure reliability, availability, and performance across the IT infrastructure. Observability solutions enable DevOps teams to proactively detect anomalies, analyze issues, and resolve problems by garnering real-time insights into the health and status of their systems, servers, applications, and other resources (Magnusson 2022).

In DevOps, observability and monitoring go hand in hand, the main difference being that monitoring tools reveal performance issues or anomalies a DevOps team can anticipate while observability infrastructure takes care of multifaceted, often unanticipated issues, such as those arising from the interplay between complex applications in distributed technology environments.

1. Monitoring collects and analyzes predetermined data pulled from individual systems.
2. Observability aggregates all data produced by all IT systems.

APM is a type of monitoring designed specifically for tracking end-to-end transactions within particular applications. APM combines monitoring with telemetry data (collection of data, including logs, metrics, and traces, across disparate systems) to enhance the user experience, perform availability monitoring, and improve performance.

When monitoring a complex system, it is the union of all these concepts that enables the:

1. Monitoring the health and status of the systems using metrics, logs, and traces.
2. Detection and reporting of anomalies.
3. Provisioning of data required to quickly troubleshoot and solve issues.

2.2 Related Work

This section pertains to an overview of the related work of interest to this project, containing the result of the literature review mentioned in the **Research Methodology** section.

2.2.1 Reviewing literature in CI monitoring

This subsection is dedicated to explain the details of the literature review conducted for this work, including the steps taken to define the search scope, relevant material found, and extrapolated results.

Search protocol

Before the search is conducted, it is necessary to define a protocol and a search scope. This includes the relevant research questions, inclusion/exclusion criteria, and search strings.

The purpose of this review is answering the first research question defined for this work:

1. **RQ1** - Which metrics are most suitable to ascertain the state of a continuous integration infrastructure? (In the context of the BMW's new AOSP project)

The population for the research are the documents whose properties match the criteria in the following list:

- Article written in the English language
- Topic Related to CI or monitoring
- Full document available and properly sourced

The studied documents were sourced from the ACM and IEEE libraries and found trough combining two sets of keywords, connected by the **AND** term:

- "*Continuous integration*", "*Continuous improvement*", "*Continuous integration process*", "*Continuous integration quality*"
- "*Data monitoring*", "*System monitoring*", "*Metric*", "*KPI*",

Screening

Following the defined search protocol, nine articles complied with the requirements and passed a simple title inspection to ascertain relevancy.

After analysing the abstract for each one, four articles were excluded, the reason being that, while related to continuous integration, they were not relevant for this study, when considering the monitoring component. The topics for these articles are "Anti-patterns in continuous integration" (two articles), "Best practices for continuous integration" (two articles).

One more article was also excluded, as it consists in a deep dive into the topic of "Metric-Correlation Models", their problems and solutions, not being directly related to system monitoring.

The remaining four articles were further inspected and their topics are summarized in following section. As this is a small scale review, no other articles were analysed from potentially relevant references from the selected documents.

Relevant topics

The topics for the four articles that passed the screening can be summarized as follows:

1. **System monitoring for performance anomaly localization** - Anomaly localization trough the use of performance probes with rule based activation (Ehlers et al. 2011).
2. **Build failure analysis** - Holistic approach to identifying the causes for build failures (Rausch et al. 2017).
3. **Lightweight and scalable monitoring implementation of large scale computing systems** - Implementation details for a lightweight and scalable monitoring solution, plus deployment in two systems (Agelastos et al. 2014).
4. **Optimizations for automatic monitoring systems** - Integrated framework for minimizing false positives and maximizing the monitoring coverage for system faults (Tang et al. 2013).

Results

The relevant topics brought to light by the review, resulted in a number of ideas that can be applied to this project, also serving as validation for some ideas gathered from the interviews with collaborators (more in the **Colaborator Interviews** section).

Notably, the anomaly localization and build failure identification topics are of great importance for this project (more in the **Functional Requirements** section).

As for the remaining two articles, these provide pointers as to how effectively deal with some aspects of an automated a monitoring solution (*i.e.*, erroneous alerting).

While the topics and ideas present in the studied articles are relevant for this thesis, their actual content does not provide much value in the context of this project. This may be a result of the still niche nature of the continuous integration topic, compounded by the unique aspects of the CI infrastructure to be monitored.

2.2.2 Other relevant work

Other investigations were also conducted regarding possibly relevant work not in the scope of the literature review. Possible candidates included a number of theses, dissertations and other scholarly articles not present in the ACM and IEEE databases.

The result of this search proved to be no more fortuitous that the literature review, as the articles found had little to no relevance considering the niche topic for this work.

2.3 Technological

This chapter is focused in analysing the existing technologies that enable infrastructure monitoring in a way that suits the **requirements** for this work. It features complete packages marketed for this purpose, and standalone technologies that achieve the same result when combined.

2.3.1 Prometheus

Prometheus is an open source systems monitoring and alerting toolkit originally built at SoundCloud[1]. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company (Prometheus 2022).

Prometheus is community-driving, meaning that the project benefits form constant development and support, and any missing feature can be added during the deployment of the solution.

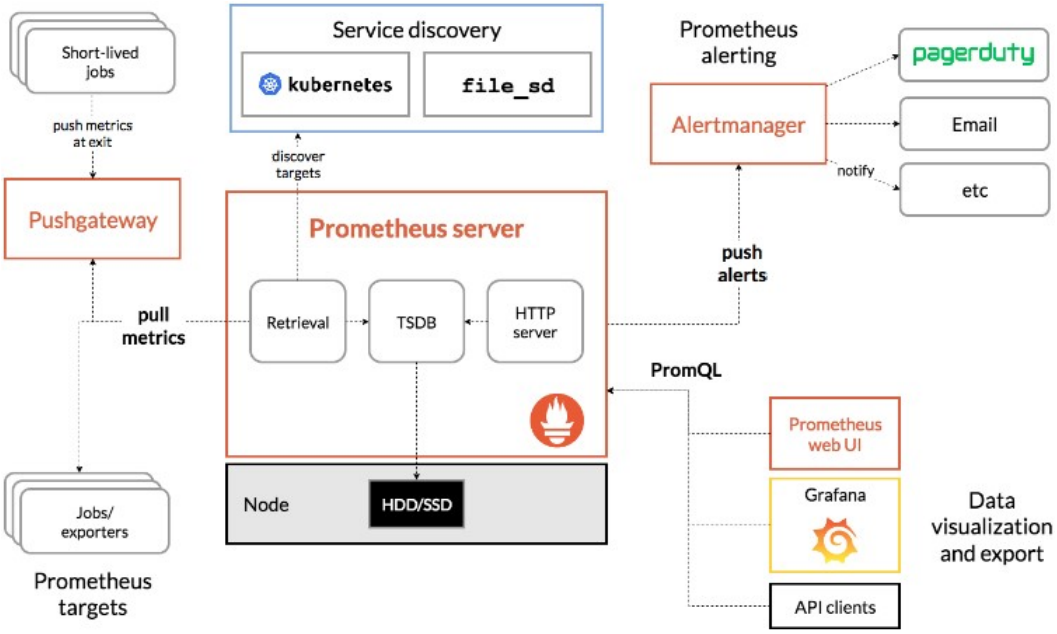

FIGURE 2.2: Prometheus architecture (Prometheus 2022)

[1]SoundCloud n.d.

The Prometheus ecosystem consists of multiple components, all of which are displayed in Figure 2.2, which contains its architecture diagram. From those, the most notable for this project are:

- The main Prometheus server which scrapes and stores time-series data
- PromQL, a flexible query language to leverage the data
- A push gateway for supporting short-lived jobs
- An alert manager to handle alerts
- The Prometheus Node Exporter

Time series data is a collection of data points obtained through repeated measurements over time. When plotting the points on a graph, one of the axes is always time. Time series metrics refer to a piece of data that is tracked at an increment in time.

The Prometheus Pushgateway is an intermediary service which allows for pushing metrics to the server, from jobs that cannot be scraped. As such it can be used to push user generated metrics that cannot be easily scraped.

The Prometheus Node Exporter is a piece of software that exposes and provides from the machine (or "node"), hardware and OS-level system metrics (*e.g.*, memory, disk, network and CPU usage), to a Prometheus server. It can also be used to fetch statistics from applications running on those nodes and convert them into metrics that Prometheus can utilize.

Grafana

Grafana is an open source data-visualization platform that allows in-depth templating of massive amounts of data, compatible with multiple of data sources (including Prometheus).

Grafana enables visualization of data at any level of granularity by allowing:

- Data **Paneling** of any kind, such as histograms, heatmaps, graphs, tables, *etc.*
- Integration with multiple **Plugins**, providing complex forms data rendering in real time (*e.g.*, data representation on a world map tile) (Grafana 2022)
- Creation and management of **Alerts** in a centralized manner (instead of scattering on the multiple data sources)
- Data**Transformations** by combining and performing calculations on data from different queries and data sources.

Prometheus and Grafana, both tools built for system metrics in a time-series format, are commonly deployed together, providing superior data visualization and analysis over the basic Prometheus web user interface.

2.3.2 The ELK Stack

The ELK stack, or Elastic stack is a collection of free and open technologies for data ingestion, enrichment, storage, analysis, and visualization:

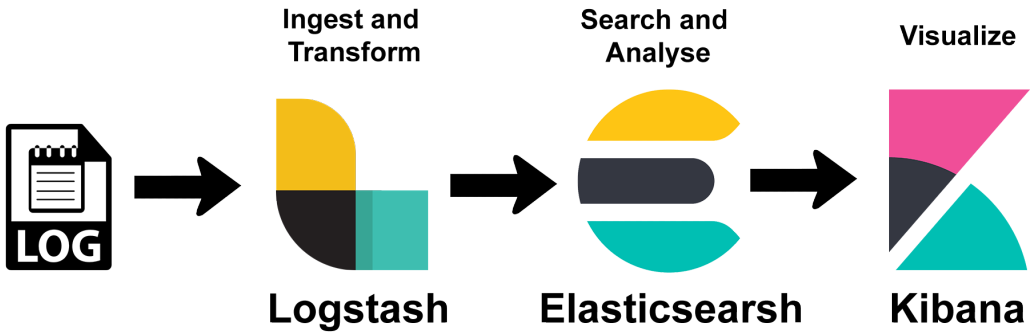

FIGURE 2.3: Simple ELK stack architecture

1. **Elasticsearch** - Search and analytics engine for all types of data. It is the central component of the ELK stack and known for its simple REST APIs, distributed nature, speed, and scalability (Elasticsearch 2022b).
2. **Logstash** - Server-side data processing pipeline that ingests data from multiple sources, transforms it, and then sends it to Elasticsearch (Logstash 2022).
3. **Kibana** - User interface that allows visualization and navigation for Elasticsearch data and the Elastic Stack (Kibana 2022).

The ELK stack provides all the same features as Prometheus, including pushing custom metrics to the server.

Elasticsearch is a very robust tool with a broad range of applications. For the purpose of this work, it can be used for log analytics, ingesting and analyzing log data in near-real-time and in a scalable manner.

How Elasticsearch works

Elasticsearch is a distributed document store. Instead of storing information as rows of columnar data, Elasticsearch stores complex data structures that have been serialized as JSON documents (Elasticsearch 2022a).

Within one second of being stored, a document is completely searchable and indexed. Inverted indexes, a type of data structure used by Elasticsearch, provide quick full-text searches. Every unique term that appears in each document is included in an inverted index, along with the documents in which it appears.

Every document in an index is a collection of fields which are key-value pairs that contain the desired data. Indexes can be thought of as efficient collections of documents. Every field in Elasticsearch has a specific, efficient data structure and is by default fully indexed.

Elasticsearch provides a REST API for managing its configuration, indexing, and searching data. When searching data, the API supports structured and full text queries, as well as a combination of both. Structures queries work similarly to SQL queries while a full text query finds all documents matching a given string, sorted by relevance. Search requests can aggregate data to build complex summaries of data, giving insight into key metrics, patterns, and trends.

Trough the use of machine learning, Elasticsearch can also analyze time series data and identify anomalous patterns in a data set.

2.3.3 Splunk

Splunk is a software platform widely used for monitoring, searching, analyzing and visualizing machine-generated data in real time. It provides instant visualizations, real-time actionable alerts, centralized controls, and scalability options.

Splunk also offers an APM package, able to "ingest, analyze and store all data in real time, at any scale, to detect all issues and troubleshoot them significantly faster" (Splunk 2022a).

After a licensing fee, Splunk enterprise edition allows for deployement of the service on-premise and, like the previous technologies, provides all the features mentioned in the **Data monitoring** section, such as:

1. Dashboards and Visualizations
2. Monitoring and Alerting
3. Reporting and Metrics
4. Performance, Scale, and Manageability
5. Integrations and Add-ons
6. Training on the platform

Splunk works similarly to Prometheus in that it uses forwarders to collect data from remote machines in real-time, forwarding it to the Indexer, which processes incoming data, indexing and storing it (like Elasticsearch).

End users interact with Splunk through the "Search Head". It allows users to do searches, analysis, and visualizations (the same as Grafana/Kibana).

Figure 2.4 depicts the architecture for the Splunk enterprise monitoring solution.

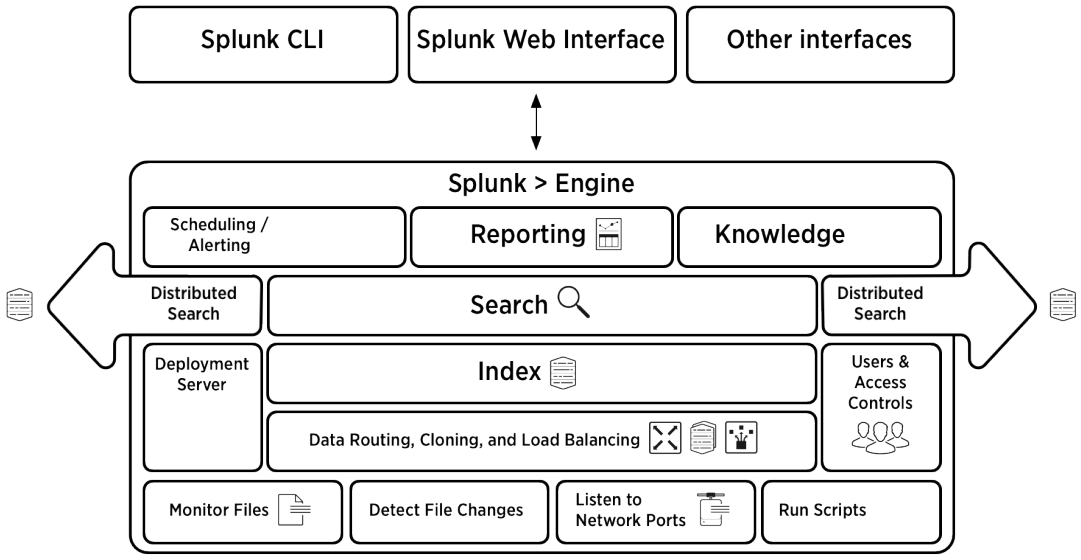

FIGURE 2.4: Splunk enterprise architecture (Splunk 2022b)

2.3.4 Comparisons

Table 2.1 illustrates the notable differences between the tools above.

Table 2.1: ELK, Prometheus and Splunk comparisons

	ELK	**Prometheus**	**Splunk**
Distribution	Open source	Open source and community developed	Proprietary with licensing cost
Packaging	Three components	Two components	Single package
Amount of Intagrations with third party tools	Low	High	High
Configuration	Complex	Simple	Simple
Scaling	Simple Horizontal and vertical scaling	Simple Vertical and complex Horizontal scaling	Simple Horizontal and vertical scaling
Log analysis	Yes	No	Yes
Professional support	Available for a cost	Available for a cost	Support hours included on cost

Alternatives

A number of other monitoring tools were also investigated for this work but ultimately dropped due to not meeting enough **Requirements** or being inferior alternatives. Notable mentions are:

1. **Datadog** - Splunk alternative, more popular but not able to meet all requirements.
2. **Sematext** - Commercial, Managed Elasticsearsh and Kibana deployment and Logstash replacement.
3. **Netdata** - Prometheus alternative, cloud only deployment and smaller community .
4. **Apache Solr** - Elasticsearch alternative with out-of-date documentation and diminishing popularity.

Prometheus + Elasticsearch

The Elastic Stack can ingest data from Prometheus, unifying Prometheus metrics with Elasticsearch logs and APM data, and correlating them in Kibana.

Following the **Value Analysis** component in the thesis, a solution built upon ELK with Prometheus support was deemed a suitable alternative for this project. As such, more information about these platforms architecture, features, and inner workings is available in the **Design** and **Implementation** chapters.

2.3.5 Amazon Web Services

Amazon Web Services (AWS), as the name implies, is a collection of cloud computing products and services provided by Amazon, on a metered pay-as-you-go basis.

As a requirement for this work, AWS must be used when deployment of software is necessary.

AWS Infrastructure

AWS has a massive global infrastructure separated in geographical regions, and further divided into availability zones. The geographical region chosen for deployment will have an impact in the:

- Latency to the cloud services
- Cost of the infrastructure
- Government regulations (Service availability or limitations)

The availability zones within a region (as least two for each) are physically separated from each other and act as redundant systems.

AWS Management Console

AWS provides a web user interface and mobile applications to access and manage its services. Resources can also be accessed through various Software Development Kits (SDKs), enabling application development with AWS as the back-end. A Command Line Interface (CLI) is provided as well with support for automation scripts.

AWS Provided Services

AWS has multiple product categories, some examples being Compute, Storage, Database, Networking, *etc.*

The notable components for this work belong to the Compute and Networking categories, the **EC2** and **VPC**.

The Amazon Elastic Compute Cloud (EC2) is a resizeable, secure, compute service that behaves as a typical hosted server. It is possible to select the operation system (*e.g.*, Ubuntu, Windows, *etc.*) and resources (*i.e.*, CPU, storage, memory), and change those resources at any moment and automatically trough the web Application Programming Interface (API). The EC2 value stems from its auto scaling properties. AWS Auto Scaling monitors deployed applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost.

The Amazon Virtual Private Cloud (VPC) is the network environment in the cloud, behaving similar to a Virtual Private Network (VPN) inside the AWS cloud. The VPC gives complete control of the network configuration, allowing the configuration of the normal network items (*e.g.*, Internet Protocol (IP) address ranges, subnet creation, route table creation, *etc.*).

Terraform

Terraform is an Infrastructure as Code (IaC) tool that enables building, changing, and versioning infrastructure safely and efficiently (HashiCorp 2022). Terraform enables the code-based description of a whole infrastructure. It aids in the simultaneous construction and provider-agnostic management of all resources

When managing IT resources, IaC replaces manual effort for lines of code, using source code to automate infrastructure management rather than manually configuring cloud nodes or physical hardware. The main benefit of such an approach is its speed an simplicity, especially when reproducibility is desired, enabling provisioning of an entire infrastructure by running a script.

Terraform has built-in version control, enabling multiple developers to configure the infrastructure simultaneously, and also simplifying the review process for changes in the code.

This tool is used in this work to provision all the necessary infrastructure in AWS and integrate it in the AOSP private cloud.

2.4 The CI infrastructure

This section pertains to an overview of the CI system of study for this work.

2.4.1 Zuul

Zuul CI, not to be confused with Netflix Zuul, is a free and open source program that drives continuous integration, delivery, and deployment systems with a focus on project gating and interrelated projects. Together with Ansible, Zuul is the core of the CI infrastructure in study.

As a gating system, Zuul does not test only a proposed change, it tests the proposed future state of multiple branches and repositories with any number of in-flight changes, and their dependencies. This is its biggest value add in managing a complex system.

Similarly to other CI/CD tools, Zuul is a service which listens to events from various code review systems, executes jobs based on those events, and reports the results back to the code review system. The primary interface for Zuul is the code review system (e.g. GitHub). A web interface is also available for inspecting the current builds status' and browsing build history, with multiple filtering options (Zuul 2022).

To better understand this and the upcoming concepts, a simple example workflow is available in the following section.

Zuul concepts and workflow

Zuul is organized around the concept of a pipeline. In Zuul, a pipeline encompasses a workflow process which can be applied to one or more projects. For instance, a "*check*" pipeline might describe the actions which should cause newly proposed changes to projects to be tested. A "*gate*" pipeline might implement Project Gating to automate merging changes to projects only if their tests pass.

In Zuul, a pipeline executes all its tasks (in Zuul they are named "*jobs*") parallelly, unless dependencies are specified. A single pipeline execution is named a "*buildset*")

Figure 2.5 illustrates a buildset execution in Zuul.

FIGURE 2.5: Zuul pipeline execution

Listing 2.1 contains the definition of a *job*, which is executed as part of a buildset.

```
- job:
    name: apinext-make-hello-world-app
    description: |
        Build the app using the common app build base job
    parent: apinext-make-app
    vars:
        gradle_targets: "build fullCoverageReport"
        use_remote_cache: True
        ...
```

LISTING 2.1: Zuul job definition

Notice the "*parent*" key, that refers to another job, which is a dependency of this one. In this case, the "apinext-make-app" job is in another repository that will be executed before this one.

The parent job can then have it's own parents and added variables. Eventually in the chain there must be a "*run*" key, indicating the Ansible playbooks that will execute the actual tasks. Listing 2.2 contains a snippet from the parent for the job above.

```
- job:
    name: apinext-make-app
    description: ...
    parent: apinext-common-settings
    pre-run: # playbooks that execute before the run phase
        ...
    run: # main playbook(s)
        - playbooks/apinext-check.yaml
        - playbooks/apinext-make-app.yaml
    post-run: # playbooks that run after main
        ...
        - playbooks/apinext-collect-logs.yaml
```

LISTING 2.2: Zuul parent job

Listing 2.3 contains an example of a simple "check" pipeline for a given project.

```
- project:
    check:
        jobs:
        - noop
        - zuullint
        - apinext-create-app-cache-sdk
        - apinext-apps-docker-build-and-push:
            vars:
              docker_image_tag: "checkbuild-{{ zuul.change }}"
        ...
```

LISTING 2.3: Zuul pipeline definition

Once a pipeline has been defined, any number of projects may be associated with it, each one specifying what jobs should be run for that project in a given pipeline.

Pipelines have associated *triggers* which are descriptions of events which should cause something to be queued into a pipeline. An example of a trigger would be creating a pull request on a GitHub project.

This triggers are one of the features that may be provided by the *Drivers*. These drivers can be code review systems (*e.g.*, GitHub, GitLab) or other services/protocols such as Simple Mail Transfer Protocol (SMTP), MQ Telemetry Transport (MQTT).

Drivers may support different functions depending on their purpose and they may be:

1. **Sources** - Host git repositories for projects.
2. **Triggers** - Emits events to which Zuul may respond.
3. **Reporters** - Outputs information when a pipeline is finished processing an item.

Continuing the workflow, once all of the jobs for an item in a pipeline have been run, the pipeline's *reporters* are responsible for reporting the results of all of the jobs.

Figure 2.6 shows the report of the job defined above, which was triggered by updating a pull request.

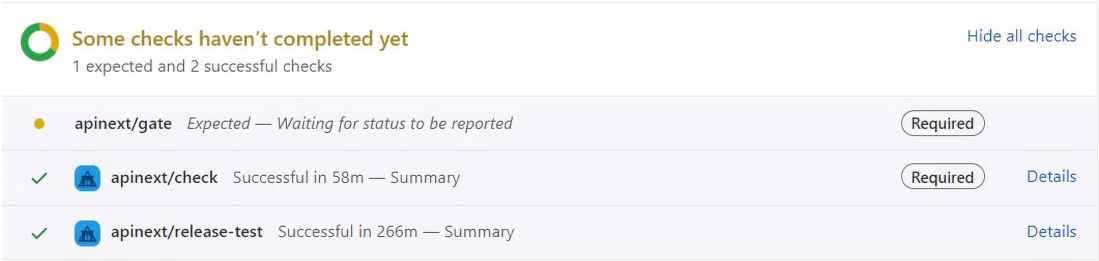

FIGURE 2.6: Check job reporting

2.4.2 Ansible

Ansible is an IT automation engine that automates cloud provisioning (allocation of a cloud provider's resources and services to a customer), configuration management, deployment and orchestration tool.

It is designed for multi-tier deployments and models IT infrastructures by describing how all of the systems inter-relate, rather than just managing one system at a time.

It uses no agents and no additional custom security infrastructure, simplifying it's deployment and uses the YAML language (in the form of Ansible Playbooks) that allow the description of automation jobs in a readable way. (R. H. Ansible 2022)

Ansible uses Jinja2 templating[2] to enable dynamic expressions and access to variables. All templating happens on the Ansible controller (machine that defines the code to execute) before the task is sent and executed on the target machine(s).

The Listing 2.4 represents the "*apinext-collect-logs*" Ansible playbook that appears in the snippet from 2.2

[2]Pallets 2022.

```
- hosts: all
  tasks:
    - name: Set default execution logs
      set_fact:
        exec_logs:
          - "reports"

    - name: Add extra log paths
      set_fact:
        exec_logs: "{{ exec_logs + collect_logs.split(' ') }}"
      when: collect_logs | default('') | length > 0

    - name: Copy execution logs
      collect_logs: (custom module)
        src: "{{ build_project_dir }}"
        dest: "{{ zuul_logs_dir }}"
        paths: "{{ exec_logs }}"
```

Listing 2.4: Ansible task definition

Ansible also allows for the definition of custom "*modules*" (the "collect_logs" keyword in the snippet), which are custom libraries that can be coded in Python, providing a great deal of implementation flexibility.

The listing contains the keywords "*hosts*" and "*tasks*". A "host" is a remote machine in which the all the "tasks" (blocks of code) defined in the playbook will be executed. Each "task" starts with the "*name*" keyword and can be executed conditionally with the usage of the "*when*" keyword.

It is also important to mention the Ansible "*roles*". a "role" is a piece of code (similar to a Java or Python function) that can be executed before, after, or as part of a task. Roles can be defined locally or installed into an environment before execution. They may also expect parameters for execution.

Listing 2.5 represents a simple playbook with a task and two roles, the first of which was installed from a public source[3] and the second one is implemented locally.

```
- hosts: prometheus
  tasks:
    - name: Ping machine
      ping:

  roles:
    - role: cloudalchemy.prometheus
    - role: ansible-grafana
      vars:
        grafana_security:
          admin_user: admin
          admin_password: password
```

Listing 2.5: Ansible role example

The "ansible-grafana" role receives variables which will later be used during execution.

The "*hosts*" keyword also has a new scope, which contains the specific IPs for the target remote machines where the tasks/roles will be executed, also referred to as "*nodes*".

[3] Ansible 2022.

2.4.3 Other Monitoring Systems available

In BMW's infrastructure there are two other monitoring solutions already in deployment (related to the AOSP project).

One is build with Elasticsearch and Kibana (no Logstash), managing logs and metadata for all Zuul jobs. This is currently used to aggregate logs for failed jobs, which will then be analyzed by a DevOps team.

One of the **functional requirements** for this theses consists on automatically analyze such failures and quantify their causes as metrics. As such, the data provided by this system is desirable.

The second monitoring system consists of a Prometheus/Grafana stack, monitoring resource load (*e.g.*, available virtual machines, network traffic, *etc.*) in a broad scale, with the AOSP being only one of the projects under monitoring.

By aggregating and correlating data from the two monitoring systems, it would be possible obtain all desired metrics and KPIs.

2.5 Methodologies

This section describes the methodologies applied in the course of this thesis.

2.5.1 New Concept Development Model

According to Peter Koen (Koen et al. 2001), the innovation process can be divided in three parts, those being:

1. **Fuzzy Front End** - The time between when a new product opportunity is first evaluated and when the product idea is deemed ready for "formal" development and "Presents one of the biggest opportunities for improvement".
2. **New Product Development** - "Disciplined and goal-oriented with a project plan", covers the complete process of bringing a new product to market, following with a formal and structured approach.
3. **Commercialization** of the product in the market.

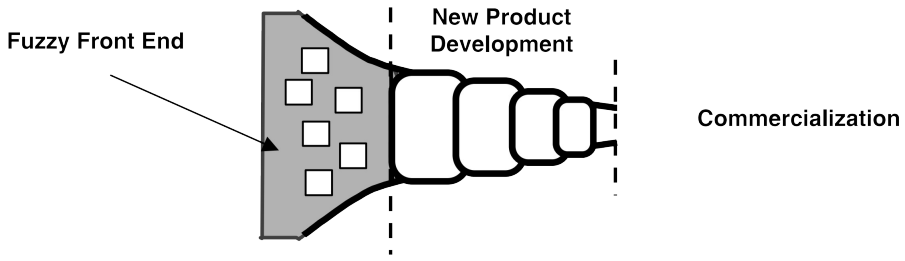

Figure 2.7: Innovation Process diagram (Koen et al. 2001)

While studying the Fuzzy Front End (FFE), Koen and his team noticed the lack of standardization in it's practice across companies due to "a lack of common terms and definitions for key elements of the FFE". To address this shortcoming, the NCD model was developed, with the intention to "provide insight and a common terminology for the FFE".

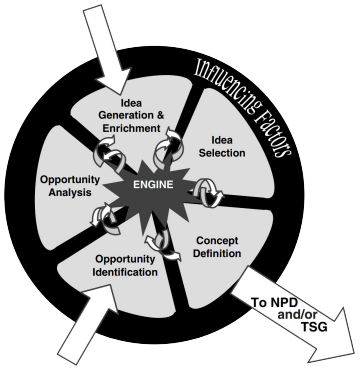

FIGURE 2.8: NCD model (Koen et al. 2001)

The three key parts of the NCD model, shown in Figure 2.8, are the:

1. **Engine / bull's-eye portion**, representing the leadership, culture, and business strategy of the organization.
2. **Inner spoke area**, defining the five controllable activity elements (opportunity identification, opportunity analysis, idea generation and enrichment, idea selection, and concept definition) of the FFE, which are driven by the engine.
3. **Influencing factors**, consisting of the organizational capabilities and external factors (distribution channels, law, government policy, customers, competitors, and political and economic climate), and the enabling sciences (internal and external) that may be involved.

2.5.2 Analytic Hierarchy Process

AHP is a mathematical, multi-criteria approach for organizing and analyzing complex decisions, developed by Thomas L. Saaty, 1980. It is trough this process that a solution to the problem will be selected (Saaty 1980).

This method entails the usage of a three-level hierarchical structure, depicted by the Figure 2.9.

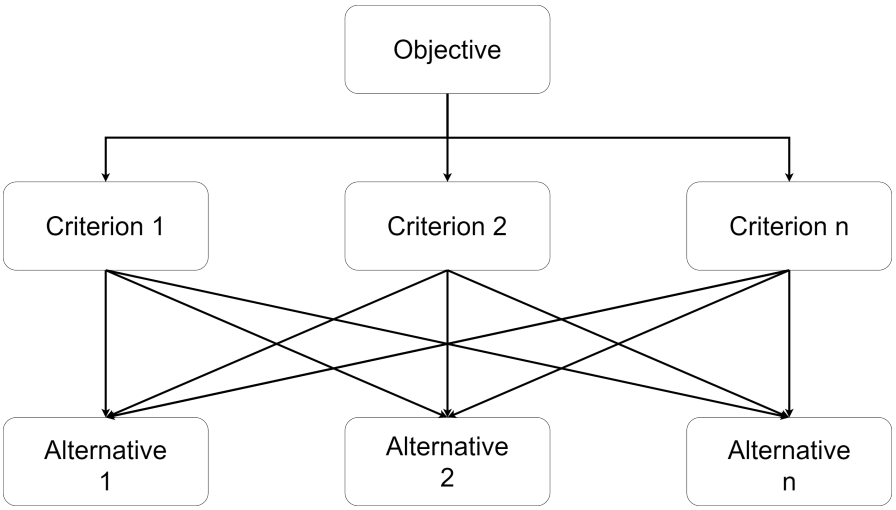

FIGURE 2.9: AHP hierarchical decision tree

2.5.3 Value proposition

A product value proposition is a statement that articulates the product's features, uses, and differentiators while taking into account the customers' problems, wants and needs (Enfroy 2022).

To better identify the value proposition for this project, the Elevator Pitch and Value Proposition Canvas tools, were applied.

Elevator Pitch

The elevator pitch is a tool used to position a product succinctly by describing (Moore 2008):

- The target segment the product is aimed at.
- The business problem it solves.
- the category it fits into.
- A unique value proposition – how it is differentiated from the competition.
- Why it is the "best buy for this situation"

Value Proposition Canvas

The Value Proposition Canvas, Figure 2.10, is an illustrative tool by Alexander Osterwalder that shows the link between the **Value Map** (left square) **Customer Profile** (right circle).

The Value (Proposition) Map describes the features of a specific value proposition in a business model in a more structured and detailed way. It breaks the value proposition down into products and services, pain relievers, and gain creators.

The Customer (Segment) Profile describes a specific customer segment in your business model in a more structured and detailed way. It breaks the customer down into its jobs, pains, and gains. (Osterwalder and Papadakos 2014)

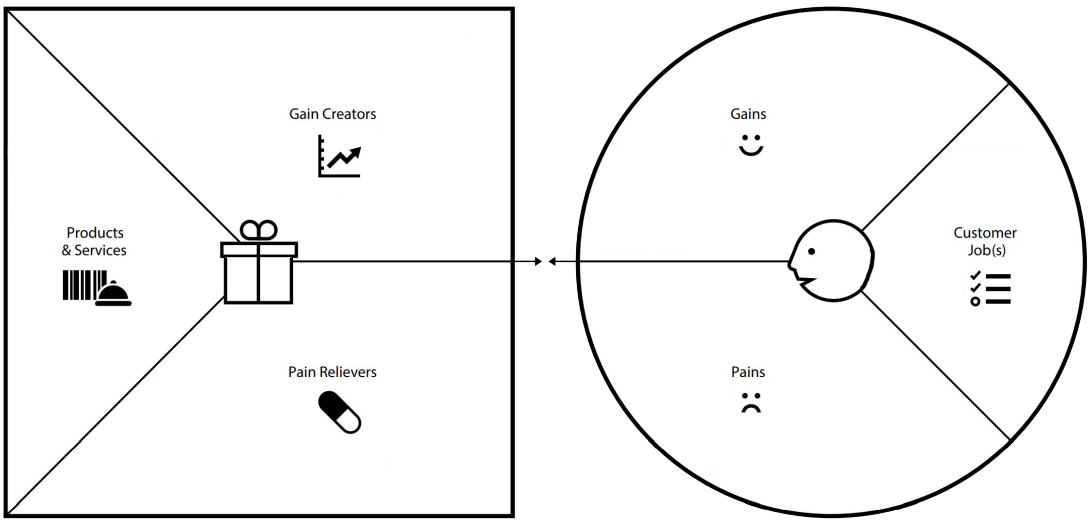

FIGURE 2.10: Value proposition Canvas

2.5.4 SWOT Analysis

SWOT analysis is a framework or technique used to analyse the strength (S), weakness (W), opportunities (O) and threats (T) of a business, strategy or product/service. It both the internal (S,W) and external (O,T) factors related to the subject.

Product SWOT analysis[4] is "an assessment methodology that you can apply to your current products or services to better understand them and, if needed, reposition them in the market".

2.6 Summary

This chapter provides insight to all relevant theoretical concepts and technologies in the context of this dissertation.

Starting from an overview of the theoretical concepts related to monitoring, it goes trough a literature review in relevant academic articles, resulting in some findings in how to properly design a monitoring solution for a complex infrastructure.

Then, according to this findings, as well as project requirements, there is an analysis on the technologies most appropriate for a solution, and how they may be utilized.

From there, ensues a description about the workings of the CI infrastructure subject to the monitoring solution, as well as other pre-existing monitoring systems that may be leveraged to support this work.

The chapter ends with an overview of the methodologies applied during the **Value Analysis** phase of the thesis.

[4]SWOT 2022.

Chapter 3

Value Analysis

According to Society of American Value Engineers (SAVE)[1], the Value Analysis (VA) or Value Methodology (VM), is a "systematic and structured approach for improving projects, products, processes, services and organizations. VM, which is also known as Value Engineering, is used to analyze and improve manufacturing products and processes, design and construction projects, business and administrative processes, and both public and private sector services and organizations."

Value is the reliable performance of functions to meet customer needs at the lowest overall cost and it can be calculated as such:

$$Value = \frac{FunctionPerformance}{Resources} \tag{3.1}$$

Where **Function Performance** describes the capacity of the subject under study to provide key customer or user functions and **Resources** are the expenditure needed to create it in terms of monetary or time-based units.

The Value analysis process consists of the steps in Figure 3.1:

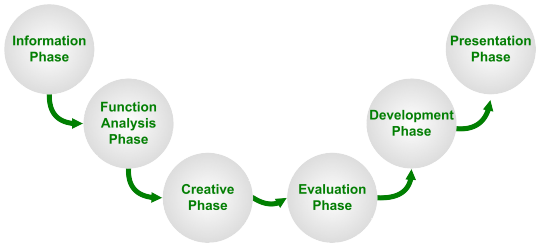

FIGURE 3.1: Value analysis process (Canada 2022)

3.1 Information Phase

The Information or Orientation Phase is the first phase of the Value Analysis. The purpose of this is to gain an understanding of the problem and any solutions that have been proposed. Project related information and data is reviewed to help the team establish a thorough understanding of the objectives and the context. The complexity of the project, the amount of information and time available will influence the level of effort, and time devoted to the Information Phase (Canada 2022).

[1]SAVE 2022.

This chapter will not refer to that phase, since the problem to be analysed has already been defined for this project.

3.2 Functional Analysis Phase

Function Analysis is a technique used to identify and understand the needs of the project, product or service, (what does it do, what must it do).

This chapter, which includes the identification and functional analysis, will be explored under the New Concept Developmen (NCD) model.

3.2.1 Opportunity Identification

"In this element the organization identifies opportunities that it might want to pursue" (Koen et al. 2001).

BMW has identified the need for better monitoring of their CI infrastructure, as a way to identify and solve problems that it may suffer from. As such, this work took that opportunity to analyse the infrastructure and implement a solution that best fits this it's needs.

3.2.2 Opportunity Analysis

"In this element, an opportunity is assessed to confirm that it is worth pursuing" (Koen et al. 2001).

The **Theoretical Context** section in the State of The Art chapter states the importance of data monitoring as well as KPIs and metrics as a whole. Moreover, the **Technological** section features multiple viable alternatives for the solution, each one with its own value.

To better analyse the opportunity as well as the **Influencing factors** of the NCD model, a SWOT analysis was conducted.

The Table 3.1 represents the SWOT analysis for this project.

Table 3.1: SWOT analysis

Strengths
1. Sizable documentation about the CI insfrastructure in question.
2. Pre-existing monitoring tools that can be studied and leveraged.
3. Potential for better managerial decisions and infrastructure improvement.
Weaknesses
1. High monetary cost for data storage systems.
2. High effort in maintaining monitoring infrastructure.
Opportunities
1. Lack of a suitable monitoring solution, prompting the opportunity for this work.
Threats
1. Sizeable amount of complete monitoring solutions in the market.

3.2.3 Idea Generation and Enrichment

"The element of idea generation and enrichment concerns the birth, development, and maturation of a concrete idea" (Koen et al. 2001). It is equivalent to the **Creative Phase** in the value analysis process.

Following the analysis of all requirements and available technologies, the proposed ideas are as such:

1. Implement a solution using a commercial product from the **Technological** section (Splunk).
2. Implement a modular solution utilizing components studied in the **Technological** section (ELK + Prometheus).
3. Improve and rely on the **Other Monitoring Systems** already in production.

3.2.4 Idea Selection

"In most instances, the problem is not coming up with new ideas. Even when businesses are being downsized, there is no shortage of new ideas. The problem for most businesses is in selecting which ideas to pursue in order to achieve the most business value" (Koen et al. 2001). The Idea Selection instance is equivalent to the **Evaluation Phase** in the value analysis process.

Similarly to the Idea generation, the Idea selection is an interactive process, meaning valuation for the ideas may change depending on new criteria and information, and may also be updated or replaced in compliance with the Idea generation element.

The idea selection for this work follows the Analytic Hierarchy Process (AHP) method.

When applying the AHP, the objective, alternatives and criteria must be defined. The objective has already been defined as "Selecting the best infrastructure monitoring system idea". Likewise, the alternatives are also defined in the **Idea Generation and Enrichment** section.

As such, the relevant criteria must be defined:

- **Security**: Overall security of the implementation, including confidentiality, availability, integrity and non-repudiation.
- **Performance**: Performance of the system, including ingestion and indexing speed, and responsiveness.
- **Suitability**: Umbrella term symbolizing how well the alternative complies with the requirements for the solution.
- **Complexity**: How simple is the implementation of the solution, in terms of development effort and maintenance post-deployment.

The Figure 3.2 illustrates the decision tree, when adapted to the context of this thesis. The alternatives are those from the **Idea Generation and Enrichment** section.

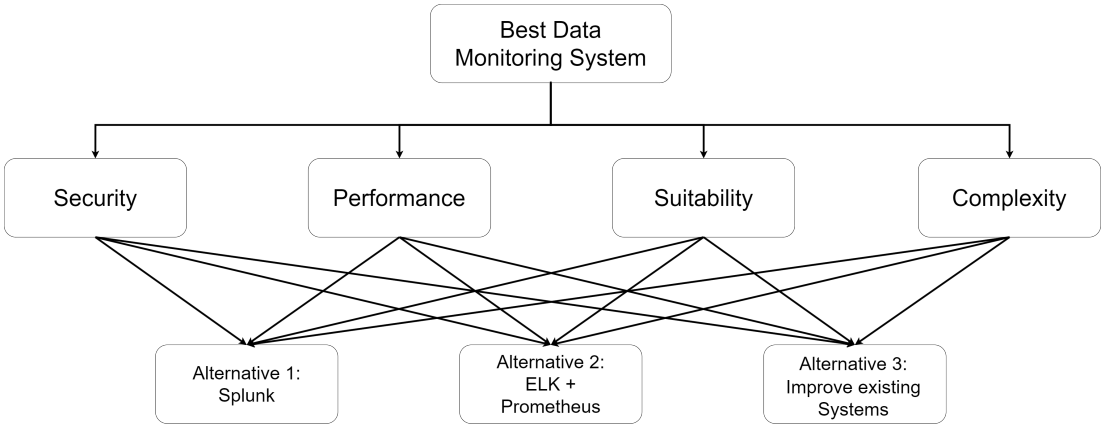

FIGURE 3.2: AHP hierarchical decision tree - adapted

After defining the hierarchical tree, the next phase consists in establishing the relative priority for each criterion pair, as per the fundamental scale defined by Saaty available in Table 3.2.

TABLE 3.2: Fundamental scale (Saaty 1980)

Intensity of importance	**Definition**	**Explanation**
1	Equal importance	Two activities contribute equally to the objective
3	Moderate importance of one over another	Experience and judgment strongly favor one activity over another
5	Essential of strong importance	Experience and judgment strongly favor one activity over another
7	Very strong importance	An activity is strongly favored and its dominance demonstrated in practice
9	Extreme importance	The evidence favoring one activity over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between the two adjacent judgment	When compromise is needed

Utilizing this scale to compare each pair, the comparison matrix in Table 3.3 ensues:

Table 3.3: Criteria comparison matrix

Criteria	Security	Performance	Suitability	Complexity
Security	1	3	1/2	2
Performance	1/3	1	1/5	1/2
Suitability	2	5	1	3
Complexity	1/2	2	1/3	1

After obtaining the comparison matrix, it's values must be normalized. Normalization is achieved by adding all the values in a column and dividing each one by the result.

An example for security criterion, in the security column would be:

$$Value = \frac{1}{1 + \frac{1}{3} + 2 + \frac{1}{2}} = \frac{1}{\frac{23}{6}} = \frac{6}{23} \approx 0.26 \tag{3.2}$$

After normalizing the values, the next step is calculating the relative priority vector for each criterion, which is simply the sum of all values in a row.

The Table 3.4 contains both the normalized values and the priority vector.

Table 3.4: Normalized criteria comparison matrix and Priority Vector

Criteria	Security	Performance	Suitability	Complexity	Priority Vector
Security	0.26	0.27	0.25	0.31	0.27
Performance	0.09	0.09	0.1	0.08	0.09
Suitability	0.52	0.45	0.49	0.46	0.48
Complexity	0.13	0.18	0.16	0.15	0.16

The priority vector indicates the relative weight of each criterion.The fourth phase is to calculate the consistency of the relative priorities.

First, the consistency matrix (Table 3.5) is calculated by multiplying the normalized matrix by a single column matrix constituted by the priority vectors.

Table 3.5: Consistency matrix

Criteria	Consistency Value
Security	1.1
Performance	0.36
Suitability	1.95
Complexity	0.63

Then, the $\lambda_{\max}$ must be calculated by averaging the division of each consistency value by the correspondent priority vector:

$$\lambda_{\max} = \frac{1.1/0.27 + 0.36/0.09 + 1.95/0.48 + 0.63/0.16}{4} \approx 4 \tag{3.3}$$

Finally, the *Consistency Index* (CI) can be calculated:

$$CI = \frac{\lambda_{\max} - n}{n-1} = \frac{4.01022 - 4}{3} = 0.00341 \tag{3.4}$$

After obtaining the CI, the final value for this phase can be achieved, the *Consistency Ration* (CR). This value is obtained by dividing the CI by the *Random Consistency Index*, a tabulated value (Table 3.6) dependent on the number of criteria:

Table 3.6: Random Consistency Index (Saaty 1980)

1	2	3	**4**	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	**0.90**	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

$$RC = \frac{CI}{RCI} = \frac{0.00341}{0.90} = 0.00379$$

Note that the RC **must** be under **0.1**, else the relative priorities are inconsistent and cannot be used. The calculated RC is under 0.1, and so, the next stage begins.

Phase five consists in developing the consistency matrix for each criterion and alternative pair, the same way they were previously calculated for each criterion pair.

The resulting tables (Table 3.7, 3.8, 3.9, 3.10) are as such:

Table 3.7: Comparison Matrix between ideas and Security criterion

Security / **Idea**	**1**	**2**	**3**
1	1	1/3	1/4
2	3	1	1/2
3	4	2	1

Table 3.8: Comparison Matrix between ideas and Performance criterion

Performance / **Idea**	**1**	**2**	**3**
1	1	1/3	1/2
2	3	1	2
3	2	1/2	1

Table 3.9: Comparison Matrix between ideas and Suitability criterion

Suitability / **Idea**	**1**	**2**	**3**
1	1	1/4	1/3
2	4	1	2
3	3	1/2	1

Table 3.10: Comparison Matrix between ideas and Complexity criterion

Complexity / **Idea**	**1**	**2**	**3**
1	1	3	1/3
2	1/3	1	1/7
3	3	7	1

Normalizing the matrices and calculating the priority vector results in the following tables (Table 3.11, 3.12, 3.13, 3.14):

Table 3.11: Normalized Comparison Matrix between ideas and Security criterion and Priority Vector

Security / **Idea**	**1**	**2**	**3**	**Priority Vector**
1	0.13	0.1	0.14	0.12
2	0.38	0.3	0.29	0.32
3	0.5	0.6	0.57	0.56

Table 3.12: Normalized Comparison Matrix between ideas and Performance criterion and Priority Vector

Performance / **Idea**	**1**	**2**	**3**	**Priority Vector**
1	0.17	0.18	0.14	0.16
2	0.5	0.54	0.57	0.54
3	0.33	0.27	0.29	0.3

Table 3.13: Normalized Comparison Matrix between ideas and Suitability criterion and Priority Vector

Suitability / **Idea**	**1**	**2**	**3**	**Priority Vector**
1	0.13	0.14	0.1	0.12
2	0.5	0.57	0.6	0.56
3	0.38	0.29	0.3	0.32

Table 3.14: Normalized Comparison Matrix between ideas and Complexity criterion and Priority Vector

Complexity / **Idea**	**1**	**2**	**3**	**Priority Vector**
1	0.23	0.27	0.23	0.24
2	0.08	0.09	0.1	0.09
3	0.69	0.64	0.68	0.67

With all the necessary values calculated, the next phase consists in obtaining the *Composite Priority* (CP) for the ideas/alternatives.

A new matrix should be formed with a row for each alternative and a column for each criterion, the cells being populated with the priority vectors for each pair calculated previously.

This matrix is then multiplied by the criteria priority value vector from phase three.

And so, Table 3.15 indicates the Composite Priority for each alternative:

Table 3.15: Composite Priority matrix

Idea	**Composite Priority**
1	0.15
2	0.42
3	**0.44**

Finally the process is completed and the "best" alternative (with the highest value) has been found, **Alternative 3**, according to the criteria defined.

Result

As a result of the evaluation phase, the third alternative, "Improve and rely on the other monitoring systems already in production", was found to be the most appropriate solution.

Note that the composite priority for alternatives 2 and 3 is very similar, suggesting that the second alternative could become the most optional, as the project matures or new information comes to light.

The result of the value analysis corroborates initial (pre-analysis), ideas for a solution (after an overview of the problem, goal, and requirements), where improving and utilizing the current monitoring systems was deemed sufficient for the current state of the project, and flexible enough to accommodate its short term growth in complexity.

3.3 Value proposition

This section applies the value proposition to this solution as described in the Value proposition section.

3.3.1 Elevator Pitch

The Table 3.16 describes the elevator pitch for this project.

Table 3.16: Elevator pitch

For	The BMW Group
who	lacks proper CI infrastructure monitoring
the	monitoring system solution
is a	log and metric management system
that	monitors the CI infrastructure
unlike	other solutions that require migration effort
our	solution provides the most value in terms of cost to performance and development time

3.3.2 Value Proposition Canvas

The Figure 3.3 illustrates the value proposition of the solution.

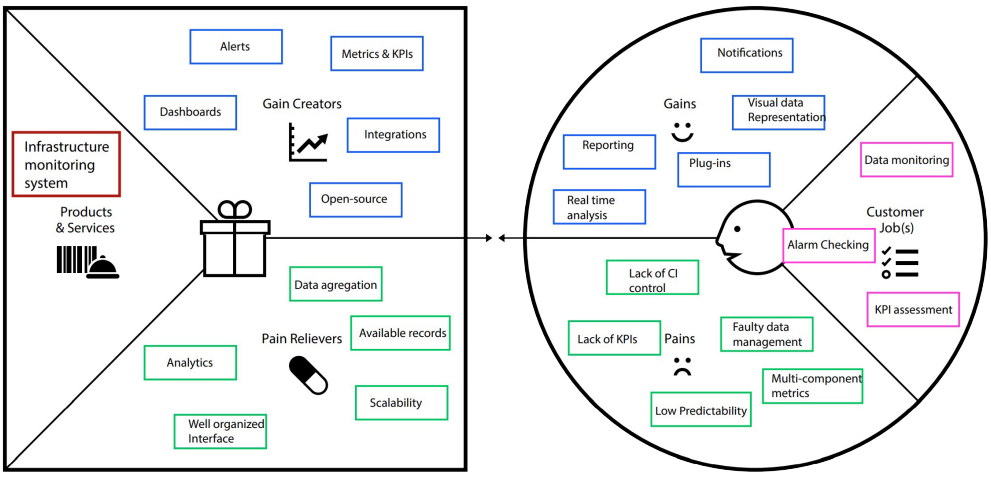

Figure 3.3: Value proposition Canvas

3.4 Summary

This chapter describes the value analysis and innovation process for this work, from identifying and analysing the opportunity to the definition of the solution. The value analysis process follows the NCD model and applies the AHP approach. The value of the solution defined was also illustrated trough the Elevator Pitch and Value Proposition Canvas tools.

Chapter 4

Design

This chapter describes the entire analysis and design process leading to the implementation, from an overview of the CI infrastructure, trough raising the requirements for the solution, closing with an architecture for implementing the solution and possible alternatives.

4.1 CI Architecture

The Figure 4.1 depicts a high level view of the relevant CI infrastructure components.

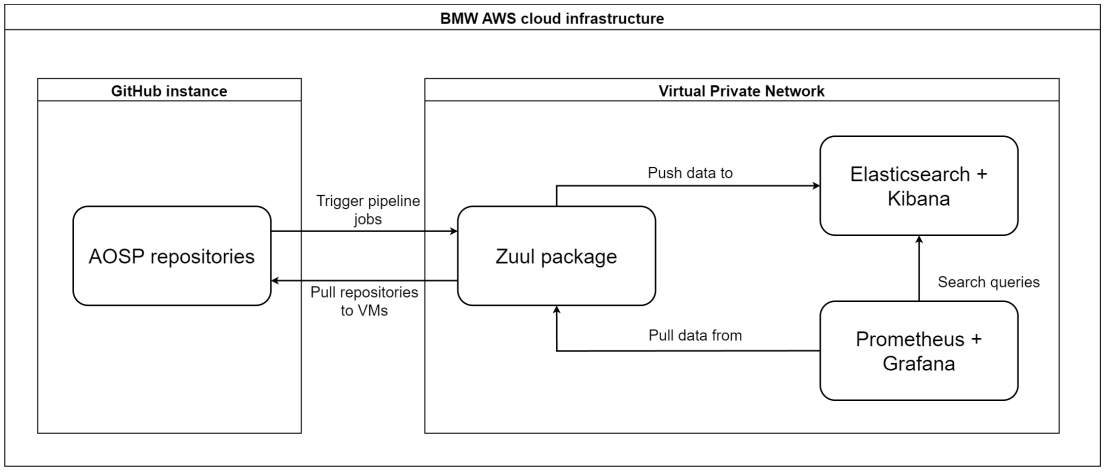

Figure 4.1: CI infrastructure architecture

The "business logic" component, which is the actual project infrastructure to be monitored, consists of multiple GitHub repositories that interact with the **Zuul** package. These repositories contain the job definitions and pipeline configurations that will be executed by Zuul. All the tasks for the jobs (*e.g.*, generate the package that will be installed in the cars), are also defined in the repositories.

The "Zuul package" consists of a Zuul instance and, in this diagram, also represents the virtual machines it controls (the ones used to execute the jobs). Zuul responds to the triggers from the repositories by scheduling the jobs in their pipeline configuration and then execute each one in a virtual machine. Every job starts with a base configuration, then clone specified repositories to the machine and execute their tasks (Ansible playbooks).

The Zuul package is connected to the **other monitoring systems** mentioned. Each virtual machine has a **Prometheus** Node Exporter installed, allowing the server instance to scrape live data during job execution. After a job completes, its execution logs and machine telemetry data are stored and ingested by the **Elasticsearch** instance.

Prometheus also queries the Elasticsearch web API to get relevant metrics that are used for alerts.

4.2 Requirements Engineering

The requirements engineering process starts by identifying the actors that take part in the solution and understand the functional requirements. The collaborator interviews mentioned in the **Research Methodology** are one of the ways to identify requirements, as well as the results from the **Literature Review**.

4.2.1 Collaborator Interviews

The interviews for this work were conducted on a focused group of collaborators with stakes in this solution (*i.e.*, the actors), including team members, colleagues that implemented the existing monitoring tools, and line managers for the project.

The selected collaborators were subject to a semi-structured interview (interview with specific questions and open answers), either in group meetings or one-on-one. The questions were simple, consisting only of:

- What metrics should be collected, and for what reason?
- What constraints apply to the solution?

The point of asking for a reason coupled to a metric idea is to reduce the number of "vanity metrics", statistics that look good on paper but do not translate to meaningful business results.

4.2.2 Functional Requirements

Functional requirements describe what the solution must be able to achieve. They can be expressed as user stories and in the context of this project, always follow same pattern:

1. As a CI engineer/Project Line manager, I want to know X about my infrastructure, with a Y degree of granularity.

It turned out from the interviews that project line managers want to know the very same of information about the infrastructure that, the CI engineers, who work directly with it, also want to know. The only difference being that line managers indicated clear business motive for wanting such metrics.

The results can be illustrated trough the use case diagram in Figure 4.2.

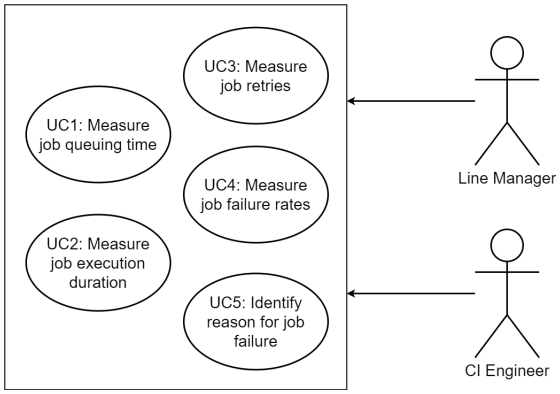

FIGURE 4.2: Use case diagram

4.2.3 Use Cases

All use cases, follow the same sequence pattern of opening their respective dashboard and checking the desired value(s). Since values are updated automatically after each job execution, there are no actions required by the actors.

All use cases assume prior authentication into the BMW network.

The sequence diagram in Figure 4.3 represents the normal system behaviour:

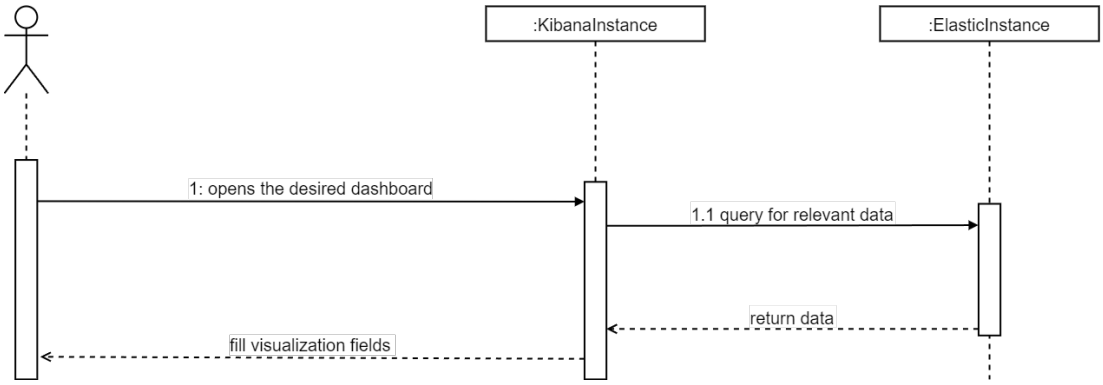

FIGURE 4.3: General sequence diagram

UC1: Measure job queuing time

This UC consists in checking the queuing time for a job prior to its execution.

With this feature it is possible to measure how infrastructure load trends over time and identify scheduling problems or lack of virtual machines.

UC2: Measure job execution duration

This UC consists in checking the execution duration for a job. It should be subdivided into the time required for executing each task.

With this feature it is possible to measure job performance and locate points of degradation within the infrastructure.

This UC is consistent with the first relevant topic resulting from the literature review.

UC3: Measure job retries

This UC consists in checking the amount of retries on a job, before the execution is complete. Should the number of retries exceed a threshold, an alert should be issued.

With this feature it is possible to react to potential infrastructure issues before they become blockers.

UC4: Measure job failure rates

This UC consists in counting the amount of failures for a job in a given period of time.

With this feature it is possible to identify a lack of reliability in a job that must be investigated.

UC5: Identify reason for job failure

This UC consists in identifying the reason for a job failure and classify it as either infrastructure or content related (whether it failed because of problems with the infrastructure or due to bad code being pushed). Infrastructure related failures should be counted and an alert issued after a specified threshold.

With this feature it is possible pinpoint problems within the infrastructure that must be investigated.

This UC is consistent with the second relevant topic resulting from the literature review.

As the only UC that requires processing and implementation apart from dashboards on Kibana, a sequence diagram is available to illustrate the behaviour:

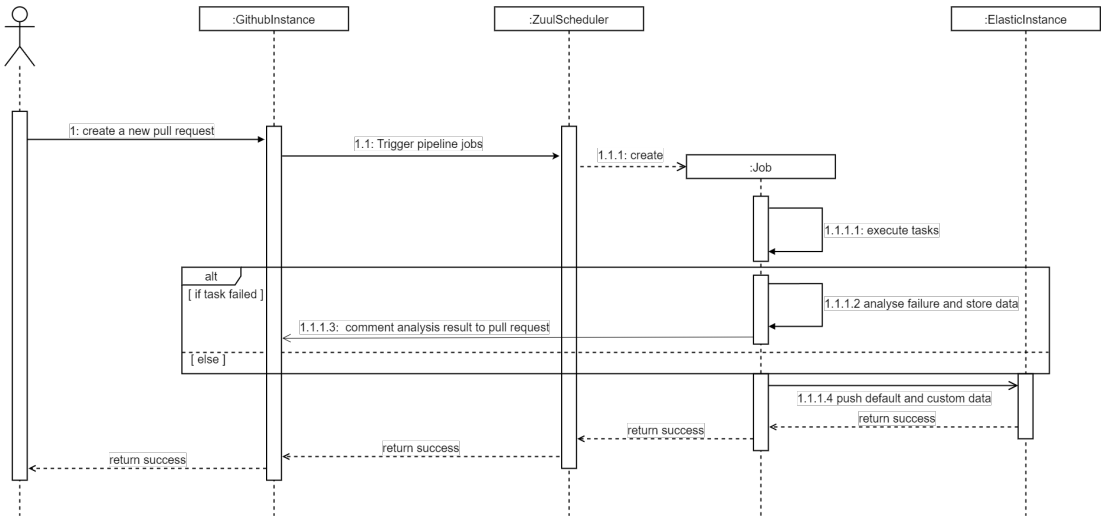

FIGURE 4.4: Job failure analysis sequence diagram

When a user creates a new pull request on a GitHub repository, Zuul triggers a pipeline execution (buildset) according to the configuration file on that repository. The same will happen when attempting to merge a pull request to the main branch, releasing a new version for an artifact, or periodically according to the repository configuration.

The Zuul scheduler starts all the jobs defined in the pipeline in separate virtual machines, in which their tasks begin execution. If an error occurs during the task execution

phase, Zuul marks the job as failed and an analysis of the execution log data (until that point) ensues. The result from the analysis is then stored in a specific file, and a message stating the result is published in the pull request comments. Before the job terminates, all data stored in the previous file is pushed to Elasticsearch.

4.2.4 Non-functional Requirements

Non-functional requirements are the requirements that describe system behaviour and quality attributes (measurable characteristics of a system that are of interest to the user), along with possible constraints. The **FURPS+**[1] model is adequate for this task.

Functionality

- User authentication is required to access the system
- Secure communication between system components must be ensured

Usability

- Simple and intuitive dashboards to easily access relevant information

Reliability

- The System must have complete up-time apart from scheduled maintenance windows
- Failing components should be able to auto-recover

Performance

- Job data used for metrics should be available in real time during execution, or right after completion (depending on the type of data)
- The system must be able to ingest, analyse, and present data form all the jobs that could be executing simultaneously

Supportability

- The system must be scalable to acompany the project growth and future inclusion of other projects.

Design Constraints

- The system must not be connect to components outside the BMW network, meaning only technologies that support on-premise or private cloud deployment can be used.
- Only established technologies with active communities and support may be used.

Implementation Constraints

- Alerts should be sent to a Microsoft Teams channel, with adequate severity indication.

[1] AL-Badareen et al. 2011.

Concerns

- Architecture design, system configuration and deployment steps, and implemented code should be sufficiently documented.

4.3 Architecture

From the **result** from the value analysis phase, the ideal solution revolves around the existing monitoring tools, which would not require any changes to the overall architecture in Figure 4.1.

Said tools already comply with all the requirements (functional and non-functional). However, there are some limitations:

- The tools are managed by other teams and only they can add new features.
- Alerting trough the Kibana dashboard is not configured, so all alert implementations must be requested to the respective team.

In the case that these limitations become blocking to the monitoring workflow, it is important to prepare an alternative solution, which, from the alternatives and result in value analysis section, would be alternative 2: "Implement a modular solution utilizing the studied components" (ELK stack with Prometheus support).

4.3.1 Design alternative

When discussing alternative 2 with the teams responsible for the existing monitoring tools, it came to light that, since it utilizes the same technologies (Elasticsearch and Prometheus), all the data available to their instances can also be provided to the new ones, resulting in the following architecture:

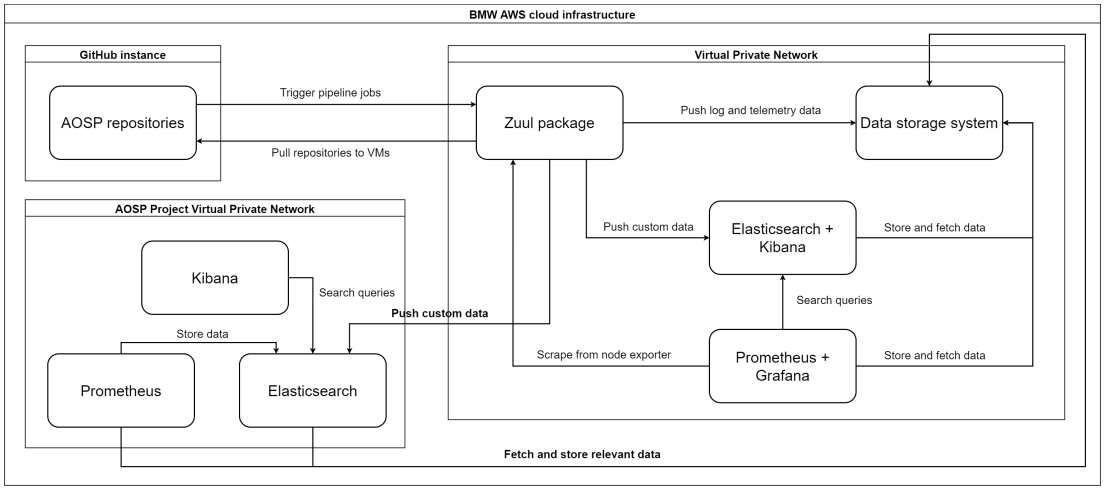

FIGURE 4.5: Architecture for alternative 2

The diagram in Figure 4.5 provides a deeper level of detail than the previous 4.1, which is now necessary to explain the design alternative.

All the log data from Zuul is stored in a data storage system, managed by the same team that maintains the provided monitoring tools. For the purpose of this solution, it is considered a black box.

There are three new components, which represent the Elasticsearch, Prometheus, and Kibana instances private to the AOSP project. These components are deployed within a private network (Amazon VPC), that can only be accessed by authorized members.

The provisioning and deployment of this servers is handled by **Terraform** (more details are available in the **Implementation** section).

The Elasticsearch and Prometheus components communicate with the provided data storage system, effectively obtaining the same data available to the other monitoring tools. The Prometheus instance can also directly send data to Elasticsearch, providing direct access to its live metrics.

The same tooling that pushes custom data to the existing Elasticsearch instance can also be used with the new one.

4.3.2 Architecture comparison

The two alternatives for the solution have a similar in overall design, by virtue of sharing most of their tooling. That said, there are major implications stemming from the differences.

Monitoring tools

The monitoring tooling in the first alternative is completely maintained by a separate team. While that causes the limitations already mentioned, there is no need to maintain any private monitoring instances. The accountability for the scalability and reliability requirements falls in the other team.

Infrastructure provisioning

Similarly to the point above, the first alternative has no need for cloud infrastructure provisioning. The existing instances are already deployed, secure, and ready to use. This reduces the overall workload of maintaining a cloud infrastructure and also saves on costs.

Functionality

The second alternative has the edge in terms of functionality. Managing the entire monitoring stack provides a degree of flexibility that was not achievable before. The second alternative enables full control when developing new features, integrating with add-ons, and configuring the alerts.

Development effort

By virtue of relying on already deployed tooling, the first alternative poses the lowest bar of entry to experiment with the monitoring tools and implement a proof-of-concept. The total development effort is also considerably lower, even after accounting with problems that may occur due to the lack of flexibility.

The second alternative is more involved and requires a lot of configuration but those problems are alleviated by all the technical knowledge already present within the organization.

4.3.3 Chosen architecture

The first alternative was deemed as "optimal" in the value analysis section prior to the complete analysis. The same holds true after defining all the requirements and analysing the alternatives.

The main driver for this decision is the development effort.

Considering the characteristics of this work, it is possible to implement a proof-of-concept, with alternative 1, while loosing little to no invested effort. That is true because of the similarities between the two alternatives. It is possible to implement some of the use cases and seamlessly transition to the second alternative, should that be necessary in the future.

Along the implementation section are brief comments on how to make a transition to the second alternative. It also includes a section dedicated to provisioning the infrastructure that would be required.

4.4 Summary

This chapter describes the functional and non-functional requirements, with relevant design artifacts, that apply to the solution for the thesis' problem.

The requirements are based on interviews with collaborators and the conducted literature review.

It also features different design alternatives and the reasoning behind the chosen solution: implementing a proof-of-concept utilizing the existing monitoring tools.

Chapter 5

Implementation

This chapter is dedicated to showcasing the implementation process for the chosen solution, demonstrating the adherence to the requirements engineering process.

Note that the implementation in this section is **not** final, instead serving as a proof-of-concept to, in the future, build a complete monitoring solution, optimal for the project.

What this chapter features then, is all the possible preparations for the final implementation, from altering the CI infrastructure to generate and expose necessary metrics, to provisioning the monitoring infrastructure in AWS. The existing monitoring systems will be used to collect some of the metrics and create relevant dashboards. As such, this implementation serves as a perfect proof of concept for the final solution, exposing possible oversights in the requirements, design, or the chosen architecture.

5.1 Technological Stack

The technological stack for the implementation consists of the technologies mentioned in the **Technological** section in the state of the art chapter.

The programming languages used are Python (for algorithms) and Jinja2 (for templating in Ansible), which are the conventions for the project. Due to the nature of the work, most of the implementation section consists in the configuration of Zuul and Ansible YAML files, and Kibana dashboards.

The implementation also features a section about the deploying procedure for AWS, which may become useful should there be the need to transition to the second design alternative for the solution. In that case, provisioning and deployment is achieved trough Terraform and Ansible is used to setup the newly created servers.

The operating systems used during development are Windows 11 and Ubuntu 20.04 trough Windows Subsystem for Linux (WSL2).

5.2 Elasticsearch and Kibana management

This section describes the necessary steps taken to prepare the CI infrastructure to push custom metrics to the Elasticsearch and setting up Kibana dashboards to aggregate them as per the functional requirements.

5.2.1 Preparing the CI

Before updating the CI infrastructure, there is a need to understand the tooling that allows pushing data to Elasticsearch.

The team responsible for the monitoring packages provides a GitHub repository with the required logic to push custom information to any Elasticsearch instance hosted within the dedicated BMW network.

At this point, it is relevant to mention that, the team providing the monitoring tools is the same one that provides the entire physical infrastructure and manages most platforms related to the AOSP (and other) projects, including the the GitHub and Zuul instances, storage systems, and the AWS Cloud infrastructure.

They are referred to as the CodeCraft[1] team.

CodeCraft provides multiple other GitHub repositories with general CI configuration and, the one containing the logic to push data to Elasticsearch, is cloned into the virtual machine during the initial setup of any jobs. That logic is accessible trough an Ansible role, which acts as a gateway to the provided features.

The provided documentation states that there are two phases in the process. The first one is to create a specific file containing key/value pairs for all data that should be pushed. The aforementioned role expects a variable containing those pairs.

The Listing 5.1 provides an example on how to execute the role.

```
- name: Store data for the Elasticsearch database
  include_role:
    name: store-data-in-file
  vars:
    index: example-index
    key_value_pairs:
      foo: bar
```

Listing 5.1: Store data in Elasticsearch example

The snippet consists of an Ansible task with the "*include_role*" keyword, which executes an available role with a given name. The name for the provided role is "store-data-in-file".

In the variables section there are two variables are defined, a dictionary containing the custom "key_value_pairs" and the **Elasticsearch "index"**, which must be configured prior to being used.

To illustrate the entire workflow involved in pushing custom data to Elasticsearch, this section will follow the implementation of **Use Case 5**.

Configure a custom index

The first step to pushing custom data is to configure a new index. A configuration repository is provided for this purpose and new indexes are integrated by adding a respective configuration file.

The most relevant part when configuring a new index is the mapping section.

[1] *CodeCraft – driving BMW's Next Generation Embedded Systems Development* 2020.

Mapping is the process of defining how a *document* (which represents a Zuul *job*), and the fields it contains, are stored and indexed by Elasticsearch. Each field has a data type (*e.g.*, text, boolean, number, *etc.*.) that defines how they are interpreted by the system. In this case, the new "failures" field is an object, which similarly to object oriented programming, is a data type that contains custom properties.

The Listing 5.2 contains a snippet of the mapping section for the new index. The "failures" object has two properties: "infrastructure" of boolean type, and "list", a keyword type.

```
{
    "index_patterns": [
        "epd-apinext-failures-*"
    ],
    ...
    "mappings": {
        "dynamic": false,
        "properties": {
          ...
          "failures": {
            "type": "object",
            "properties": {
              "infrastructure": {
                "type": "boolean"
              },
              "list": {
                "type": "keyword"
              }
            }
          },
          ...
        }
    }
}
```

Listing 5.2: Index configuration file

The complete file is available as A and contains other fields that are automatically filled with telemetry data from the job. The index pattern is also defined in this file.

The purpose of this use case is identifying job failures as either infrastructure related, or not. The "infrastructure" and "list" properties will be set according to the analysis result.

Build failure analysis

In Zuul, a task failure during the main execution (*run*) phase of a job, does not cause the entire workflow to abort. Instead, it will set a global flag expressing an execution failure and skip to the *post-run* phase. The Listing 2.2 shows the three phases in the job configuration.

All jobs defined in the AOSP repositories have the same parent job at the top of the hierarchy. The *post-run* phase of that parent job only executes after the ones from all its child jobs.

The *post-run* phase of the "base" job is the ideal place to configure the failure analysis, since all the jobs will eventually execute it.

```
- job:
    name: apinext-common-settings
    description: Base job for AOSP projects
    parent: ...
    abstract: true
    pre-run: ...
    post-run:
      - playbooks/push-to-elasticsearch.yaml
    ...
```

Listing 5.3: Project base job

The "*abstract*" keyword in Listing 5.3 indicates that the job should only be depended on and is not meant for execution. A new "push-to-eslasticsearch" playbook is defined in the *post-run* section.

```
- hosts: all
  roles:
    - role: push-failures-to-elasticsearch
```

Listing 5.4: Push to Elasticsearch playbook

This playbook serves only as a bridge form the job to execute a role, which contains more of the logic. The reason for such an approach is to aid with modularity and maintainability when new features are added.

The "push-failures-to-elasticsearch" playbook is defined in Listing 5.5

```
- block:
    - name: Analyse build failure
      include_role:
        name: extract-build-failure
      vars:
        max_length: 20
        report_to_pr: true
        report_last_error: false

    - name: Print extracted build failures
      debug:
        var: extracted_build_failures

    - name: Identify failure types
      apinext_check_build_failures:
        extracted_failures: "{{ extracted_build_failures }}"
      register: apinext_failures

    - name: Insert data into the ElasticSearch database
      include_role:
        name: store-data-in-file
      vars:
        index: epd-apinext-failures
        key_value_pairs:
          retries: "{{ apinext_failures.identified_failures }}"

  when: not (zuul_success | bool)
```

Listing 5.5: Push failures to Elasticsearch role

The "*block*" keyword is used to conditionally execute the tasks within. In this case, the tasks are only executed when the "zuul_success" variable is *False* (the global flag that is changed then the main execution fails).

The first task executes a role provided by the CodeCraft team. It analyses the current execution's logs and returns a result. The algorithm is implemented in Python and works my matching the log file against predefined error patterns. A default error pattern file is provided but it can be replaced by a custom one to better match the projects' needs. After execution, it comments the result in the GitHub pull request (Listing 5.1) and also sets a new global variable, "extracted_build_failures".

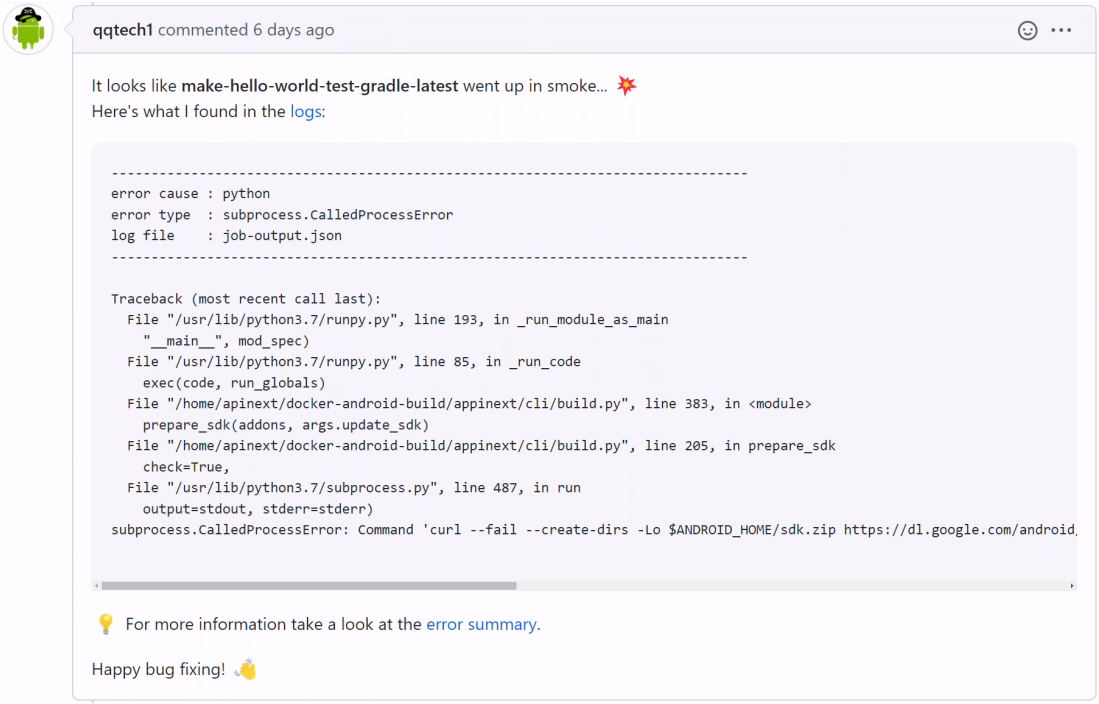

FIGURE 5.1: GitHub error comment

The value from "extracted_build_failures" is then printed for debugging. It consists of a dictionary with all possible failure types as keys, and a boolean value indicating their occurrence.

```
{
  "extracted_build_failures": {
    "ansible": {
      "templating": false
    },
    "cmake": true,
    ...
    "infrastructure": {
      "git": true,
      "docker": false,
      ...
    }
    "unknown_errors": false
  }
}
```

LISTING 5.6: Extracted failures snippet

The next task makes use of a custom Ansible module that was implemented to complement this use case. Its purpose is to parse the extracted failure dictionary and create an object that will be sent to Elasticsearch. As input, it uses the "extracted_build_failures" variable that was registered in the previous task.

```
INFRASTRUCTURE_ERROR_KEYWORDS = ["infrastructure", "unknown_errors"]

def run_module() -> None:
    module_args = dict(
        extracted_failures=dict(type="dict", required=True),
    )
    module = ansible.module_utils.basic.AnsibleModule(argument_spec=
    module_args)
    failures = []
    check_failures(
        module.params["extracted_failures"],
        failures,
    )
    infrastructure_failure = any(
        error for error in INFRASTRUCTURE_ERROR_KEYWORDS if error in
    failures
    )
    result = {
        "identified_failures": {
            "infrastructure": infrastructure_failure,
            "list": failures,
        },
    }
    module.exit_json(**result)
```

Listing 5.7: Extracted failures snippet

A simple recursive algorithm is able to transverse the entire dictionary and return a list with the positive failures. Then it generates a return variable to Zuul, which can be accessed during the rest of the execution.

```
def check_failures(extracted_failures: dict, failures: list) -> None:
    """
    Fill the failures list with all the failure types

    :param extracted_failures: dict with extracted failures
    :param failures: list with found failure types
    """
    for k, v in extracted_failures.items():
        initial_failures = len(failures)
        if isinstance(v, dict):
            check_failures(v, failures)
            if len(failures) > initial_failures:
                failures.append(k)
        else:
            if v is True:
                failures.append(k)
```

Listing 5.8: Extracted failures snippet

Finally, it is possible to use the role provided to push data to Elasticsearch (Listing 5.1), using the newly created "identified_failures" object and configured index pattern. This role stores data in a specific file and sets a global flag that triggers another role, which later pushes the actual data to Elasticsearch.

This second role may be used manually when pushing data to a custom Elasticsearch instance is desired. In that case, it requires the Uniform Resource Locators (URL) of the instance and access credentials (which can be configured as Zuul secrets[2]).

```
- name: Push elastic metrics
  include_role:
    name: store-data-in-es
  vars:
    elasticsearch:
      host: url/to/custom/Elasticsearch/instance
      username: ""
      password: ""
```

LISTING 5.9: Push data in Elasticsearch example

In the case of opting for the **design alternative 2**, the entire workflow described can be leveraged by utilizing this role.

5.2.2 Kibana Job data

After creating the new index, failing jobs automatically push the analysis result to Elasticsearch. The index is then populated with information about those jobs.

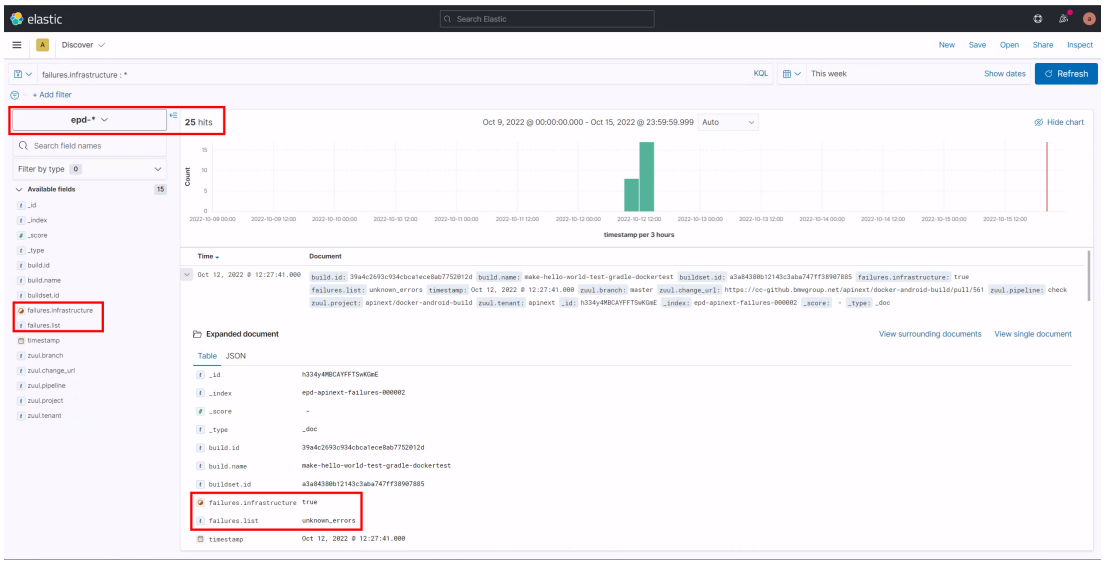

FIGURE 5.2: Elasticsearch failed jobs

The Figure 5.2 illustrates the Kibana landing page for the new index. In the top-most highlighted section, exists a dropdown reading "epd-*", which is the prefix for all custom indexes in the AOSP project. To the right, is the count off all jobs in that category (25), after filtering for those that possess the "*failuers.infrastructure*" field (above the highlighted section).

Down to the next highlighted section, lay all the available fields for the "epd-*" index prefix. While all fields are available, a job may provide only part of them. The newly added fields can be seen inside the rectangle.

[2]Zuul n.d.

The final highlighted section is part of the "*document*" that represents a job. There, it is possible to see the result of the failure evaluation which, in this case, indicates an infrastructure error of "unknown cause". The properties above the highlighted section provide the metadata for the job and, along with the two new properties, can be used for filtering and aggregation purposes.

Kibana Dashboarding

To finalize the use case requirements, a suitable visualization is required. Kibana provides an intuitive user interface to create visualizations, with several general purpose types.

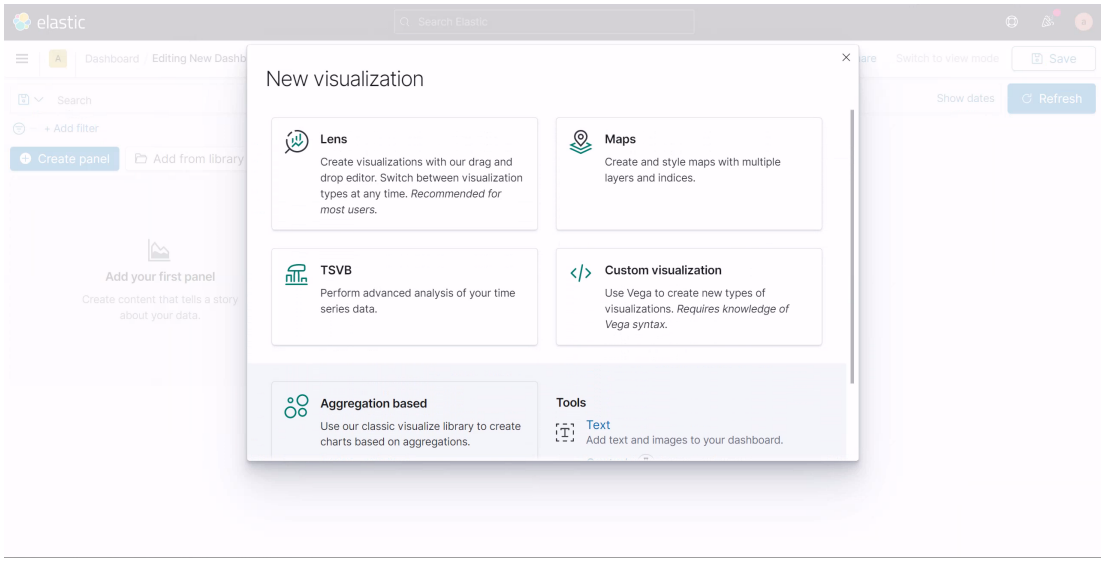

FIGURE 5.3: Kibana visualizations

For this purpose, a "*Lens*" type visualization can accurately illustrate the relevant information for all, infrastructure related, failed jobs.

The Figure 5.4, depicts the configuration process for the failures visualization. It is based on a "*Data table*", which lists all the jobs from the custom "epd*" indexes. The list is filtered for only jobs in which the "*failures.infrastructure*" property is set as *True*. The relevant fields are then added to the table, in this case:

- The Job name (*build.name*)
- The list of identified errors (*failures.list*)
- The count of failures

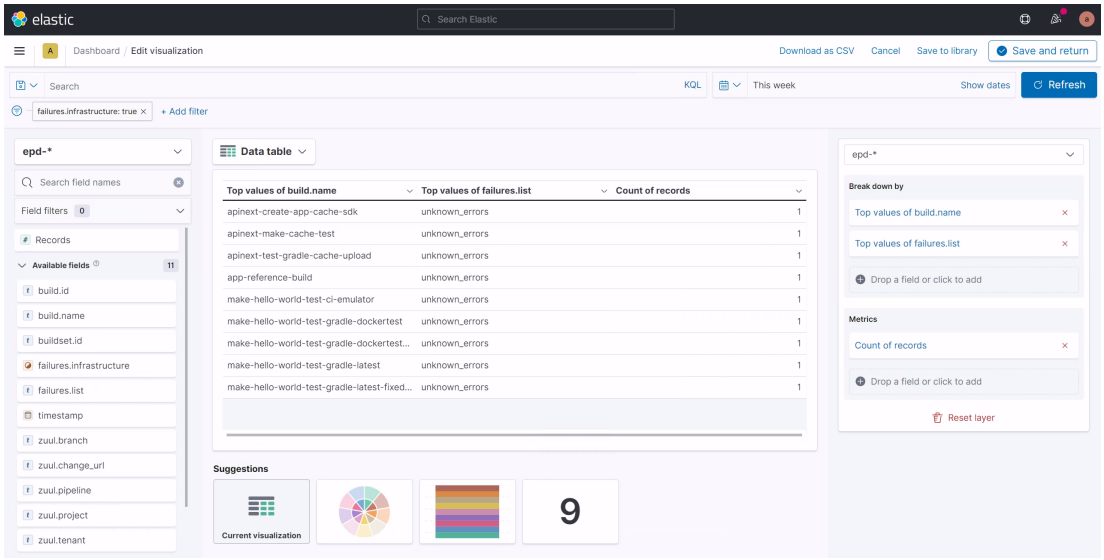

FIGURE 5.4: Kibana *Lens* visualization

Figure 5.5 renders the dashboard with the newly created visualization, providing options for further filtering based on properties or time-frame.

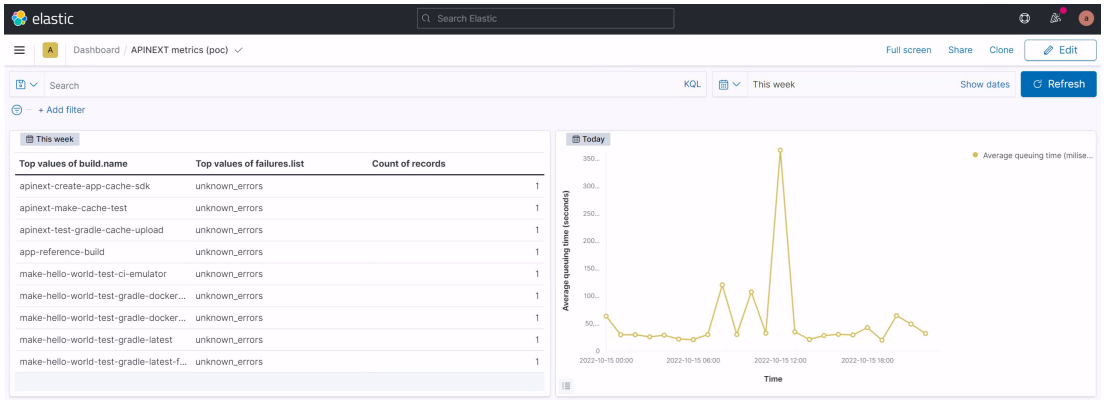

FIGURE 5.5: Kibana dashboard

Another visualization lays beside the failures table. It indicates, throughout the day, the average queue time for buildsets in Zuul. This visualization is related to **use case 1** and is created from a simple line chart where, the X axis represents the passage of time and the Y axis represents the average queue time for build sets.

5.3 AWS Deployment

This section describes the provisioning for an AWS Cloud deployment of the monitoring tools, relying on Amazon EC2 compute instances.

Access to the infrastructure is restricted to some employees, who may act in the system based on the **roles** possessed (a set of permissions that grant access to actions and resources in AWS). The AOSP project's cloud infrastructure can only be accessed by employees possessing specific roles.

5.3.1 Terraform Provisioning

Infrastructure provisioning is managed by **Terraform**. The first step is to configure Terraform to access the BMW AWS infrastructure.

Prerequisites

To authenticate against AWS, BMW provides a command line tool for login that stores an access token valid for a period of time.

```
terrantez:~$ bmwaws login
(browser pop-up for authentication)
INFO - bmwaws.login (line: 103): Credentials stored in shared
    credentials file: ~/.aws/credentials in profile: bmwsso
```

Listing 5.10: BWM AWS login

An exemple credentials file can be seen in Listing 5.11. Text in full upper case indicates that the respective value has been redacted from the listing.

```
[bmwsso]
aws_access_key_id = ACCESS_ID
aws_secret_access_key = SECRET_KEY
aws_session_token = SESSION_TOKEN
valid_until = 2022-10-12 23:59:23+00:00
```

Listing 5.11: AWS credentials file

Terraform also provides a CLI that can be obtained as an Advanced Package Tool (APT) package.

```
terrantez:~$ sudo apt install terraform
Reading package lists... Done
Building dependency tree
Reading state information... Done
terraform is already the newest version (1.3.1).
```

Listing 5.12: Terraform installation

The command "*$ terraform init*" initializes a working directory containing Terraform configuration files (similar to "*$ git init*") and is the first command that should be executed when starting (or cloning) a new configuration.

Infrastructure configuration

The configuration for this section features the base components required for an eventual final deployment. These are:

1. **Provider** - The could provider Terraform will interact with
2. **Resource group** - Define a resource group in the cloud
3. **Security group** - Provide details about a security group in the cloud
4. **EC2 instance** - Configure the EC2 instance resources and access

The example configuration consists of a single Terraform (.tf) file, but more complex configurations may use multiple files that communicate trough variables. The full configuration file is available as B. Starting from the Provider, it requires a cloud

region ("eu-central-1" in this case) and an AWS profile for authentication, which can be obtained from the credentials file in Listing 5.11.

```
provider "aws" {
  profile = "aosp_profile"
  region = "eu-central-1"

  default_tags {
    tags = {
      project    = "monitoring-poc"
      managed_by = "Terraform"
    }
  }
}
```

Listing 5.13: Terraform provider configuration

The "aosp_profile" has been added to the credentials file:

```
[bmwsso]
...

[aosp_profile]
region = eu-central-1
role_arn = USER_ROLE
source_profile = bmwsso
```

Listing 5.14: AWS updated credentials

The "*role_arn*" key requires an Amazon Resource Name (ARN) for the user role that will interact with the system. Every resource in AWS has a unique resource name that can act as an identifier.

The "*default_tags*" dictionary defines some tags that will be used to identify the configured resources.

Following with the resource group, this component defines an "*aws_resource_group*", a service that enables the management and automation of tasks in multiple resources at the same time, for example, simultaneously restart all the EC2 instances.

```
resource "aws_resourcegroups_group" "resource_group" {
  name = "monitoring-poc"

  resource_query {
    query = <<JSON
    {
    "ResourceTypeFilters": [
        "AWS::AllSupported"
    ],
    "TagFilters": [
        {
        "Key": "project",
        "Values": ["monitoring-poc"]
        }
    ]
    }
JSON
  }
}
```

Listing 5.15: Terraform resource group configuration

The Listing 5.15 defines a new resource group in AWS. The "*resource_query*" key specifies the JSON query string that AWS will use when searching for the resources to place in a group.

The next component, AWS security group, controls the traffic that is allowed to reach and leave the resources that are associated with it. In this context, it is used to control inbound and outbound traffic for the EC2 instance.

The Listing 5.16 depicts part of these rules, with the remainder available in the B.

```
resource "aws_security_group" "poc_security_group" {
  name   = "poc-security-group"
  vpc_id = "AOSP-VPC"

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "TCP"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Listing 5.16: Terraform security group configuration

The "*vpc_id*" key is optional and defines the private network in which the security group will be inserted.

Each "*egress*" and "*ingress*" block specify an outbound or inbound rule, respectively. In the example, the egress block allows full outbound access to the network and the ingress block allows only access to port number 22 with the TCP protocol, allowing Secure Shell (SSH) connections. All network connectivity to and from EC2 instances are blocked by default.

Finally, the EC2 instance is defined in the Listing 5.17:

```
resource "aws_instance" "poc_ec2_instance" {
  tags = {
    Name = "monitoring-poc"
  }
  ami                    = "ami-06148e0e81e5187c8"
  instance_type          = "t2.micro"
  key_name               = "Diogo Terrantez"
  vpc_security_group_ids = [aws_security_group.poc_security_group.id]
  subnet_id              = "SUBNET_ID"
}
```

Listing 5.17: Terraform EC2 instance configuration

The configurations used for the EC2 machine are as such:

1. **ami** - Amazon Machine Image (AMI) that will be installed in the instance (Ubuntu 20.04 in this example)

2. **instance_type** - storage capacity tier (a free tier in this example)
3. **key_name** - alias for the public SSH key that will be added to the instance
4. **vpc_security_group_ids** - id for a specific security group (Listing 5.16)
5. **subnet_id** - id for the sub network

Infrastructure deployment

To deploy the new configuration, Terraform provides the "*$ terraform plan*" command. It creates an execution plan that provides a preview of the changes that Terraform plans to make in the infrastructure. By default, the command:

- Reads the current state of any already-existing remote objects, ensuring the Terraform state is up-to-date
- Compares the current configuration to the prior state and noting any differences
- Proposes a set of actions that, if applied, updates the remote objects to match the configuration

The Listing 5.18 contains part of the output of the command, when applied to the configuration above.

```
terrantez:~/aws/terraform$ terraform plan -out example_plan

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-06148e0e81e5187c8"
      + arn                                  = (known after apply)
      + availability_zone                    = (known after apply)
      ...
    }

  # aws_resourcegroups_group.resource_group will be created
  + resource "aws_resourcegroups_group" "resource_group" {
      + arn      = (known after apply)
      + id       = (known after apply)
      + name     = "monitoring-poc"
      ...
    }

  # aws_security_group.poc_security_group will be created
  + resource "aws_security_group" "poc_security_group" {
      + arn                    = (known after apply)
      + description            = "Managed by Terraform"
      + egress                 = [
          ...
        ]
      + ingress                = [
          ...
        ]
      ...
    }
Plan: 3 to add, 0 to change, 0 to destroy.
Saved the plan to: example_plan
```

Listing 5.18: Terraform plan result

The "*-out*" option specifies a file where the plan for the configuration will be stored. The final deployment step is to execute the "*$ terraform apply example_plan*" command, applying the changes above.

After deployment, a new EC2 instance can be seen in the AWS management console.

FIGURE 5.6: New EC2 instance

5.3.2 EC2 Configuration

After confirming the deployment and checking the instance IP address, it is necessary to configure the monitoring tools. For this example, a Prometheus server is installed using Ansible.

A Configuration approach using Ansible is ideal for the same reason why Terraform is preferred for deployment. It provides a quick, simple, and **reusable** approach when it is desirable to configure multiple machines at the same time or multiple times.

To execute playbooks inside the new instance, Ansible makes use of the SSH server that is automatically installed in every EC2, authenticating against the public key that was configured.

When using the Ansible CLI, it is possible to provide an "*inventory*" file, that defines variables for execution. In this example, a "prometheus" group is defined, that contains the hosts to execute the playbook.

```
1 [prometheus]
2 10.3.79.245 ansible_ssh_private_key_file=/home/terrantez/.ssh/
    aws_ssh_key.pem ansible_ssh_user=ubuntu
```

LISTING 5.19: Ansible inventory

For this example, a single host was defined, the new EC2 instance that was deployed. A host is defined by an IP address and access credentials necessary for authentication.

The Listing 5.20 illustrates the playbook that configures the Prometheus server:

```
1 - hosts: prometheus
2   roles:
3     - role: cloudalchemy.prometheus
```

LISTING 5.20: Prometheus setup playbook

It contains the "*hosts*" key, which will receive the value configured in 5.19, and the role "*cloudalchemy.prometheus*"[3], a public, custom role that installs a Prometheus instance in the host.

A snippet from the execution of this playbook can be seen in Listing 5.21.

[3] Ansible 2022.

```
terrantez:~/aws$ ansible-playbook -vv ./playbooks/apinext-monitoring-
    setup.yaml -i ./aws.ini

PLAYBOOK: apinext-monitoring-setup.yaml
------------------------------------------------------------
1 plays in ./playbooks/apinext-monitoring-setup.yaml

PLAY [prometheus] ---------------------------------------

TASK [Gathering Facts] ----------------------------------
task path: /home/terrantez/aws/playbooks/apinext-monitoring-setup.yaml:1
ok: [10.3.79.245]
...
...
PLAY RECAP ----------------------------------------------
10.3.79.245: ok=23 changed=0 unreachable=0 failed=0 skipped=11 rescued=0
     ignored=0
```

Listing 5.21: Prometheus installation

The playbook is executed with the "*$ ansible-playbook*" command, which as options receives the playbook to execute and the inventory file.

The redacted section in the snippet contains the output of all the tasks that execute as part of the role.

Following the execution, the Prometheus instance setup is complete and a landing page is available in the EC2 instance, exposed on port 9090 (the default Prometheus port).

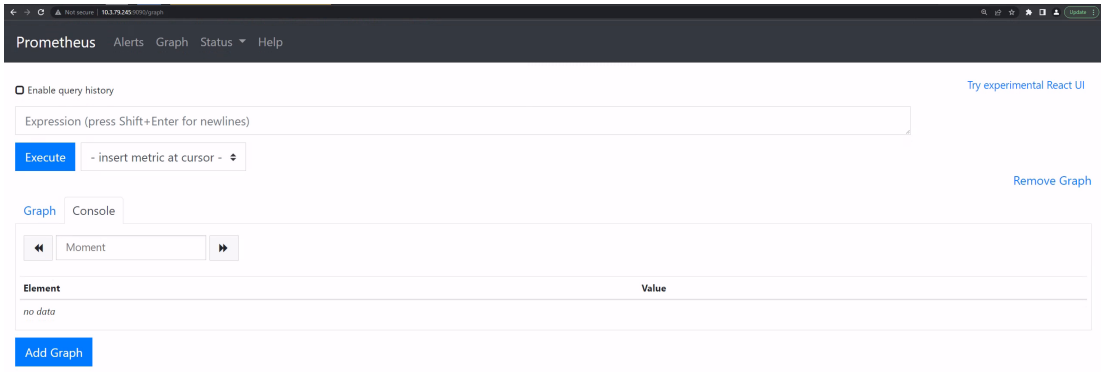

Figure 5.7: Prometheus landing page

5.4 Summary

This chapter presents implementation details for the chosen solution and an alternative that may be adopted in the future. Part of the effort lies in adapting the CI infrastructure in a way that is compatible with both monitoring alternatives. Then, the complete implementation of a use case is showcased, with visible results. The final section features basic AWS provisioning for a possible deployment of private monitoring instances.

Chapter 6

Experiments

This chapter relates to the evaluation of the overall work. The project has two distinct components, the monitoring system as a whole, and its takeaways (defined metrics, KPIs and extrapolations from their results). That is because the goal, as defined in the **Goals** section was " developing a monitoring system and aggregating relevant metrics to formulate meaningful KPIs", as well as making "an objective analysis of the system and its problems and aid/influence or even provide higher level, educated, decision making, culminating in a more sustainable and optimized present and future developmen".

As such, the evaluation must also encompass both segments. Due to the nature of this segments, they must be evaluated following both a quantitative approach (for the monitoring system), and a qualitative approach (for it's takeaways).

6.1 Indicators

As a way to evaluate the monitoring system, some indicators were defined. This indicators are the same as the ones used as the criteria in the **Analytic Hierarchy Process** (except the time constraints).

- **Security**: Overall security of the implementation, including confidentiality, availability, integrity and non-repudiation.
- **Performance**: Performance of the system, including ingestion and indexing speed and responsiveness.
- **Suitability**: Umbrella term symbolizing how well the alternative complies with the requirements for the solution.

Note that the "Suitability" indicator includes compliance with the functional and non-functional requirements, as well as any other quality attributes that may apply.

6.2 Assessment

This section pertains to the assessment methodology applied to this work, explaining the hypotheses and approaches .

6.2.1 Hypotheses

An hypothesis is a declaration that must be corroborated, following some kind of testing approach. The following hypotheses, which have been defined in the **Hypotheses (1.4)** section, will be evaluated in compliance with the previous indicators.

1. **H1** - The system's state can be accurately represented by a set of metrics and KPIs.
2. **H2** - The defined metrics and KPIs can be measured and leveraged for data-driven decisions.

The first hypothesis states that the entire CI infrastructure state can be accurately represented by a set of metrics and KPIs, which means that it's problems and weaknesses can be identified and quantified trough this means.

The second hypothesis states that, utilizing measured values from metrics and KPIs in the system, one has the necessary information to make data-driven, optimal decisions which will improve the quality of the system under analysis.

6.2.2 Methodology

As already stated, the assessment of this work, must follow different approaches, depending on the segment (hypothesis) under evaluation.

The following table REF represents the evaluation methods used for each hypothesis.

Segment / **Indicator**	**H1**	**H2**
Quantitative	Software testing / QEF	n. a.
Qualitative	n. a.	Interviews and Hypothesis Testing

Table 6.1: Evaluation methods

Interviews

Qualitative interviews or intensive interviews are semi-structured since the researcher has a particular topic for the respondent, but open ended-questions. The objective is to directly hear the respondents' and analyse their opinions.

The evaluation for **H2** will follow this approach with a questionnaire that is yet to be defined.

Quality Evaluation Framework

The Quality Evaluation Framework (QEF) is a quantitative evaluation method that judged the quality of a software solution according to specified criteria (Escudeiro and Bidarra 2008).

This approach will be detailed at the time of application.

Testing

This evaluation will follow, when applicable, the Verification and Validation model[1] (V-Model).

[1] tutorialspoint 2022b.

The V-Model is an extension of the waterfall[2] model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase.

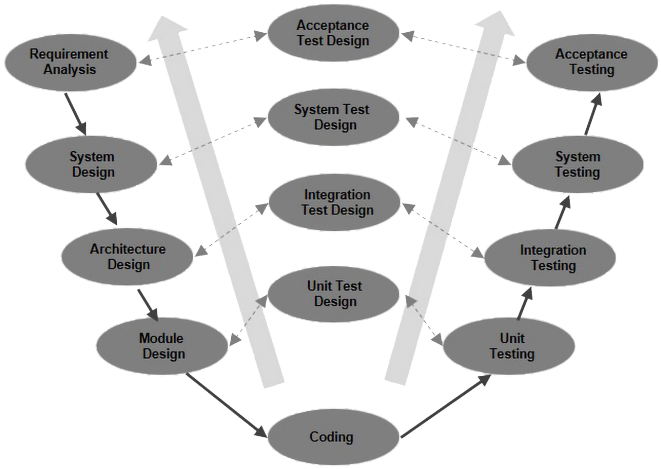

FIGURE 6.1: V-Model testing phases

Hypothesis Testing

Hypothesis testing[3] is a systematic procedure for deciding whether the results of a research study support a particular theory which applies to a population.

It is composed of two hypotheses:

- H_0 - The **null hypothesis** which is the opposite of the research hypothesis and expresses that there is no relationship between variables, or no differences between groups.
- H_A - The **research hypothesis** which is proposed.

If the null hypothesis is rejected, then the alternative to the null hypothesis is accepted.

In hypothesis testing, a value is established to determine if the null hypothesis is accepted or rejected, as well as whether the result is statistically significant:

- A critical value is the score the sample would need to decide against the null hypothesis.
- A probability value is used to assess the significance of the statistical test.

6.3 KPI Definitions

KPIs are the measurements by which business-critical activities, objectives, or goals can be evaluated.

Although not KPIs per se, when implemented, the act as business gauging metrics.

Their correlation to business strategies is based on:

[2] tutorialspoint 2022a.
[3] *Libguides: Maths: Hypothesis testing* 2022.

- **UC1 - Measure job queuing time**: Knowing how long jobs are queued for before starting execution is important to the business because it indicates a possible lack of enough physical resources (virtual machines) to execute all the jobs in question. As a result, it may be necessary to increase their number based on the average time that jobs spend in queue.
- **UC2 - Measure job execution duration**: Measuring a job's execution time is important to the business because it indicates, trough historical data, the possible deterioration in the state of the infrastructure. If execution duration's start taking longer, it is an indication of a potential issue introduced recently that should be investigated. This metric helps in maintaining the efficiency of the infrastructure.
- **UC3 and UC4 - Measure job retries and failure rates**: These two use cases are important to the business because they indicate the overall infrastructure reliability. If there are no (infrastructure related) job failures or retries, it is safe to assume that the infrastructure in in a good state. On the other hand, has jobs start executing retries and eventually fail, it suggests that some component in the infrastructure is experiencing problems and can be investigated before any major outage occurs.
- **UC5 - Identify reason for job failure**: Knowing the reason of a job failure is important to the business because it indicates which components are experiencing issues at any given time. When the root cause of a problem is known, it is simpler to concentrate efforts on finding solutions.

By aggregating this metrics it is possible to obtain deeper business insight, resulting in some possible KPIs:

1. **Overall infrastructure health** - By combining the reliability data from UC3 and UC4 with the failure analysis from UC5, It is possible to, at any moment, verify the state of each infrastructure component and generate an alert for the responsible team, should any start indicating a large number of retries or failures.
2. **Physical infrastructure** - With the data from UC1 it is possible to indicate how "physical infrastructure" should be allocated. For example, virtual machines allocated to a project may be lend to another one during times with low average queue time (more machines are available that jobs being executed).

Chapter 7

Conclusion

The purpose of this chapter is to communicate the findings resulting from this work. It ties together all the previous chapters in an attempt to describe all the accomplishments, limitations, and future work that characterize the solution to the problem statement.

The thesis concludes with some final remarks about the work topic.

7.1 Research Questions

This section is dedicated to answering the questions that drove the research for this work.

RQ1 - Which metrics are most suitable to ascertain the state of a continuous integration infrastructure? (In the context of the BMW's new AOSP project)

The metrics that best measure the state of the CI infrastructure, specific to the AOSP project, are those defined in the **use cases** section.

The initial assumption for this research question was that, general metrics resulting from the literature review and other such sources would be able to adequately ascertain the state of a related system. That quickly turned out to not be the case. To measure the state of a system, the base requirement is understanding what exactly it is trying to achieve and what needs to be monitored. As such, save for the odd coincidence, metrics measuring one system do not translate to others.

RQ2 - How can metrics and KPIs depict an accurate representation of the infrastructure's problems and weaknesses/vulnerabilities?

From the results of this work, the KPI that tracks "overall infrastructure health" contains a metric that measures the amount of infrastructure related failures (use case 5). If that metric consistently indicates a high amount of infrastructure failures of a specific type, then it is providing clear insight into a weakness/vulnerability in the infrastructure.

RQ3 - How can we leverage data monitoring to improve the development of a complex system? (The AOSP CI infrastructure)

Trough the definition and tracking of KPIs, it is possible to provide relevant information to steer the future development of the system. The "physical infrastructure" KPI in the **KPI Definitions** section provides such an example.

7.2 Contributions

The contributions from this work are clear and directly impact the BMW AOSP project.

With the build failure analysis implementation, team members no longer need to manually analyse all failures for specific projects. Until now a rotating task existed where a team member would classify infrastructure related failures by type and report on them in a project-wide weekly meeting. The workload has now been reduced, allowing more time for more valuable tasks.

The use cases implemented and defined KPIs enable the CI team to quickly identify and solve issues, and provide useful information that project line managers can leverage when assigning tasks, among other project related decisions.

Finally the thesis document itself may be used as a documentation source when similar problems arise, either within the organization, or by the CI community at large.

7.3 Limitations

The main limitations of this work lie with the inability to deploy alerts. The lack of flexibility brought upon by the chosen design solution hindered part of the implementation but not to a degree of invalidating the results. Visualizations were perfectly usable with the amount of data provided and the team accountable for the monitoring tools pledged to implement alerts that are requested, and to investigate the possibility of enabling alerting trough Kibana.

Though a characteristic of the project under analysis, rather than an inherent limitation of the thesis, the amount of metrics to be collected, and therefore use cases, identified during the analysis phase, was small. Compounding their overall lack of implementation complexity, the result is a lack of diversification when demonstrating the implementation process and a small amount of KPI definitions. This is, however expected to change as the project matures.

7.4 Future Work

Future work for this thesis revolves around whether a transition to second monitoring alternative is necessary. While the groundwork has already been laid, deeper investigation is required into the configuration for the private monitoring instances, should they be necessary. Though their compliance with the requirements was already verified, and configuration can be obtained from the ones already in production, is not guaranteed that it will be so in the future as the project grows or more get included in the scope. In time, the number of KPIs to be monitored is expected to increase, requiring further implementation.

7.5 Personal Remarks

The topic for this project is fairly interesting. Challenging as well, not in the implementation side (though there were some limitations) but in the investigative phase. Documentation in this topic is very sparse and hardly useful in most situations, resulting in a poor result for the literature review. The amount of metrics that can be

implemented may be great but KPIs not so much. To define one, deep knowledge about the business concepts is required, and it certainly cannot standardized.

During the investigation portion of this thesis, It came to be that, while monitoring solutions for complex projects are far from unusual, they tend to be developed in-house, therefore being tightly coupled to the domain, lacking in the realm of standardization. Furthermore they are mostly closed to the public, even inside the same organization, as was the case for the other monitoring solutions deployed by BMW, which access was mostly blocked to anyone not belonging to the team, impeding investigation at a deeper level.

As such, I can only hope that this work can shine a light in this domain and provide a starting point for others wishing to implement something similar in the future.

Bibliography

Agelastos, Anthony et al. (2014). "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '14. New Orleans, Louisana: IEEE Press, pp. 154–165. ISBN: 9781479955008. DOI: 10.1109/SC.2014.18. URL: https://doi.org/10.1109/SC.2014.18.

Al-Ani, Tarik (2012). "Android In-Vehicle Infotainment System (AIVI)". In: URL: http://hdl.handle.net/10523/2114 (visited on 02/26/2022).

Ansible (2022). URL: https://galaxy.ansible.com/cloudalchemy/prometheus (visited on 10/02/2022).

Ansible, Red Hat (2022). *How Ansible Works*. URL: https://www.ansible.com/overview/how-ansible-works (visited on 02/26/2022).

AL-Badareen, Anas et al. (Feb. 2011). "Users' Perspective of Software Quality". In.

Canada, Value Analysis (2022). *VA VE OVERVIEW*. URL: https://www.valueanalysis.ca/aboutva.php (visited on 02/26/2022).

CodeCraft – driving BMW's Next Generation Embedded Systems Development (Dec. 2020). URL: https://jfrog.com/user-conference/codecraft-driving-bmws-next-generation-embedded-systems-development/.

Doran, George T. (1981). "There's a S.M.A.R.T. way to write managements's goals and objectives." In: *Management Review., 70(11)*, pp. 35–36.

Ehlers, Jens et al. (2011). "Self-Adaptive Software System Monitoring for Performance Anomaly Localization". In: *Proceedings of the 8th ACM International Conference on Autonomic Computing*. ICAC '11. Karlsruhe, Germany: Association for Computing Machinery, pp. 197–200. ISBN: 9781450306072. DOI: 10.1145/1998582.1998628. URL: https://doi.org/10.1145/1998582.1998628.

Elasticsearch (2022a). *Data in: Documents and indices: Elasticsearch Guide [master]*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/master/documents-indices.html (visited on 10/11/2022).

– (2022b). *What is Elasticsearch?* URL: https://www.elastic.co/what-is/elasticsearch (visited on 10/02/2022).

Ellingwood, Justin (Dec. 2017). *An introduction to metrics, monitoring, and alerting*. URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting (visited on 02/26/2022).

Enfroy, Adam (Feb. 2022). *Your ultimate guide to defining your product's value proposition*. URL: https://www.qualtrics.com/experience-management/product/value-proposition/ (visited on 02/11/2022).

Escudeiro, Paula and José Bidarra (Jan. 2008). "Quantitative Evaluation Framework (QEF)". In: *Conselho Editorial/Consejo Editorial*, p. 16.

Experian (2022). *Data monitoring*. URL: https://www.edq.com/glossary/data-monitoring/ (visited on 02/26/2022).

Grafana (2022). *Worldmap panel*. URL: https://grafana.com/grafana/plugins/grafana-worldmap-panel/ (visited on 10/03/2022).

HashiCorp (2022). *Terraform documentation*. URL: https://www.terraform.io/docs (visited on 10/05/2022).

Informatica (2022). *What is data monitoring: Definition: Informatica UK*. URL: https://www.informatica.com/gb/services-and-training/glossary-of-terms/data-monitoring-definition.html (visited on 05/30/2022).

Kibana (2022). *Kibana: Explore, visualize, Discover Data*. URL: https://www.elastic.co/kibana/ (visited on 10/02/2022).

Koen, Peter et al. (2001). "Providing Clarity and A Common Language to the "Fuzzy Front End"". In: *Research-Technology Management* 44.2, pp. 46–55. DOI: 10.1080/08956308.2001.11671418. eprint: https://doi.org/10.1080/08956308.2001.11671418. URL: https://doi.org/10.1080/08956308.2001.11671418 (visited on 02/26/2022).

KPI.org (2022). *KPI Basics*. URL: https://kpi.org/KPI-Basics (visited on 02/26/2022).

Libguides: Maths: Hypothesis testing (2022). URL: https://latrobe.libguides.com/maths/hypothesis-testing (visited on 02/11/2022).

Lightbend (2022). *Monitoring architecture*. URL: https://developer.lightbend.com/guides/monitoring-at-scale/monitoring-architecture/architecture.html (visited on 02/26/2022).

Logstash (2022). *Logstash: Collect, parse, transform logs*. URL: https://www.elastic.co/logstash/ (visited on 10/02/2022).

Magnusson, Andrew (2022). *Observability vs. monitoring: Understanding the difference*. URL: https://www.strongdm.com/blog/observability-vs-monitoring (visited on 10/02/2022).

Martinez-Fernandez, Silverio et al. (2015). "A Survey on the Benefits and Drawbacks of AUTOSAR". In: *Proceedings of the First International Workshop on Automotive Software Architecture*. WASA '15. Montréal, QC, Canada: Association for Computing Machinery, pp. 19–26. ISBN: 9781450334440. DOI: 10.1145/2752489.2752493. URL: https://doi.org/10.1145/2752489.2752493 (visited on 02/26/2022).

Moore, Geoffrey A. (2008). "Elevator pitch". In: *Crossing the chasm marketing and selling disruptive products to mainstream customers*. Collins Business Essentials.

Osterwalder, Alexander and Trish Papadakos (2014). "Value Proposition Design". In: *Get started with ... value proposition design: How to create products and services customers want*. Wiley.

Pallets (2022). *Template designer documentation¶*. URL: https://jinja.palletsprojects.com/en/latest/templates/ (visited on 10/05/2022).

Perez, Susan (2022). *Understanding the difference between Kpis and metrics*. URL: https://www.brightgauge.com/blog/understanding-kpis-and-metrics (visited on 02/26/2022).

Prometheus, Prometheus (2022). *Overview: Prometheus*. URL: https://prometheus.io/docs/introduction/overview/ (visited on 02/26/2022).

Rausch, Thomas et al. (2017). "An Empirical Analysis of Build Failures in the Continuous Integration Workflows of Java-Based Open-Source Software". In: *Proceedings of the 14th International Conference on Mining Software Repositories*. MSR '17. Buenos Aires, Argentina: IEEE Press, pp. 345–355. ISBN: 9781538615447. DOI: 10.1109/MSR.2017.54. URL: https://doi.org/10.1109/MSR.2017.54.

Saaty, T.L. (1980). *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Advanced book program. McGraw-Hill International Book Company. ISBN: 9780070543713. URL: https://books.google.pt/books?id=Xxi7AAAAIAAJ (visited on 02/11/2022).

SAVE (2022). *About Value Engineering - Save International*. URL: https://www.value-eng.org/page/AboutVM (visited on 02/26/2022).

SoundCloud (n.d.). *Prometheus: Monitoring at SoundCloud*. URL: https://developers.soundcloud.com/blog/prometheus-monitoring-at-soundcloud.

Splunk (2022a). *Application performance monitoring*. URL: https://www.splunk.com/en_us/products/apm-application-performance-monitoring.html (visited on 10/02/2022).

– (2022b). *Splunk Enterprise Architecture and Processes*. URL: https://docs.splunk.com/Documentation/Splunk/8.0.0/Installation/Splunksarchitectureandwhatgetsinstalled (visited on 10/02/2022).

– (2022c). *What is Application Performance Monitoring*. URL: https://www.splunk.com/en_us/data-insider/what-is-application-performance-monitoring.html (visited on 10/02/2022).

Stackify (Apr. 2021). *Metrics Monitoring: Choosing the right KPIs*. URL: https://stackify.com/metrics-monitoring-choosing-the-right-kpis/ (visited on 02/26/2022).

SWOT (2022). *SWOT analysis on a product or service*. URL: https://www.swotanalysis.com/blog/swot-analysis-on-a-product-or-service (visited on 02/26/2022).

Tang, Liang et al. (2013). "An Integrated Framework for Optimizing Automatic Monitoring Systems in Large IT Infrastructures". In: KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, pp. 1249–1257. ISBN: 9781450321747. DOI: 10.1145/2487575.2488209. URL: https://doi.org/10.1145/2487575.2488209.

Turner, John (2022). *What is observability? beyond logs, metrics, and traces*. URL: https://www.strongdm.com/observability (visited on 10/02/2022).

tutorialspoint (2022a). URL: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm (visited on 02/11/2022).

– (2022b). *SDLC - V-Model*. URL: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm (visited on 02/11/2022).

Zuul (2022). *About Zuul*. URL: https://zuul-ci.org/docs/zuul/latest/about.html (visited on 02/26/2022).

– (n.d.). *Zuul secrets*. URL: https://zuul-ci.org/docs/zuul/latest/config/secret.html.

Appendix A

Elasticsearch index full configuration file

```
{
  "index_patterns": [
    "epd-apinext-failures-*"
  ],
  "settings": {
    "index": {
      "lifecycle": {
        "name": "epd-apinext",
        "rollover_alias": "epd-apinext-failures"
      },
      "number_of_shards": "4"
    }
  },
  "mappings": {
    "dynamic": false,
    "properties": {
      "build": {
        "type": "object",
        "properties": {
          "id": {
            "type": "keyword"
          },
          "name": {
            "type": "keyword"
          }
        }
      },
      "failures": {
        "type": "object",
        "properties": {
          "infrastructure": {
            "type": "boolean"
          },
          "list": {
            "type": "keyword"
          }
        }
      },
      "buildset": {
        "type": "object",
        "properties": {
          "id": {
            "type": "keyword"
          }
```

```
        }
      },
      "zuul": {
        "type": "object",
        "properties": {
          "change_url": {
            "type": "keyword"
          },
          "pipeline": {
            "type": "keyword"
          },
          "project": {
            "type": "keyword"
          },
          "branch": {
            "type": "keyword"
          },
          "tenant": {
            "type": "keyword"
          }
        }
      },
      "timestamp": {
        "type": "date"
      }
    }
  }
}
```

Listing A.1: Index configuration file

Appendix B

Terraform full configuration file

```
provider "aws" {
  profile = "aosp_profile"
  region = "eu-central-1"

  default_tags {
    tags = {
      project    = "monitoring-poc"
      managed_by = "Terraform"
    }
  }
}

resource "aws_resourcegroups_group" "resource_group" {
  name = "monitoring-poc"

  resource_query {
    query = <<JSON
    {
    "ResourceTypeFilters": [
        "AWS::AllSupported"
    ],
    "TagFilters": [
        {
        "Key": "project",
        "Values": ["monitoring-poc"]
        }
    ]
    }
JSON
  }
}

resource "aws_security_group" "poc_security_group" {
  name   = "poc-security-group"
  vpc_id = "AOSP-VPC"

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 9090
    to_port     = 9090
    protocol    = "TCP"
```

```
48      cidr_blocks = ["0.0.0.0/0"]
49    }
50
51    ingress {
52      from_port   = 3000
53      to_port     = 3000
54      protocol    = "TCP"
55      cidr_blocks = ["0.0.0.0/0"]
56    }
57
58    ingress {
59      from_port   = 22
60      to_port     = 22
61      protocol    = "TCP"
62      cidr_blocks = ["0.0.0.0/0"]
63    }
64  }
65
66  resource "aws_instance" "web" {
67    tags = {
68      Name = "monitoring-poc"
69    }
70    ami                    = "ami-06148e0e81e5187c8"
71    instance_type          = "t2.micro"
72    key_name               = "Diogo Terrantez"
73    vpc_security_group_ids = [aws_security_group.poc_security_group.id]
74    subnet_id              = "SUBNET_ID"
75  }
```

Listing B.1: Terraform configuration file