Analysing and alerting on application logs within Kubernetes infrastructure

Danila Koryugin
d.koryugin@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

Software applications designed as a set of micro-services constantly require supervision. Logging of applications can help development teams explore and discover errors, exceptions, mistakes, and inefficiencies in the workflow within micro-services. Kubernetes is the most popular technology to maintain, manage and work with micro-services. Nevertheless, Kubernetes technology lacks in ability to effectively and efficiently provide logging information to DevOps developers. To cope with this, there exist various solutions to provide powerful logging capabilities. However, in this paper main focus will be on two of the most popular stacks of logging solutions that can also be integrated into Kubernetes infrastructure, namely Elastic stack (also known as ELK stack) and PLG stack with Prometheus. In addition to logging analysis, these stacks provide an opportunity to alert on these logs based on specified rules, e.g. alert on errors. The goal of this research paper is to understand which stack is the most suitable for providing application logging within Kubernetes infrastructure based on chosen metrics for a Java application, as well as define which stack is more relevant in terms of implementation based on a literature review. The results showed that there is no significant difference between chosen metrics. However, the results of the analysis based on the literature review showed that there is a large prevalence and usage of ELK over PLG in various research papers.

KEYWORDS

Kubernetes, Logging, ELK stack, Elastic search, PLG stack, Loki, Grafana, K8s, Elastic Stack, Docker, Promtail, Logstash, Kibana, Prometheus

ACM Reference Format:
Danila Koryugin. 2023. Analysing and alerting on application logs within Kubernetes infrastructure. In *Proceedings of 39th Twente Student Conference on IT (TScIT).* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Software applications play a huge role in people's lives. Since the beginning of software development, developers were focusing on building monolithic applications, which have a single code base that includes multiple services [1]. These services can communicate with external systems or consumers via different interfaces like Web services, TCP connection, and REST API. Moreover, they support communication with each other, for instance, web application service most of the time requires database service to store data. However, with the growth of companies providing more and more software solutions, the need for a different approach to software development became extremely important. Due to the change from a more conservative development method (Waterfall) to iterative and with emphasis on responding to change over following a plan (Agile) [13], companies tend to switch from monolithic software to micro-services. That is where Kubernetes technology appears as one of the best for managing and scaling micro-services applications. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerised applications [6]. Nevertheless, it is necessary to supervise the state of the services, such as discovering potential errors, exceptions, the overall behaviour of a micro-service application, and other important information, that would help development teams to understand what should be debugged. Logging is one of the solutions to cope with these problems. Log is defined as a record of performance, events, or day-to-day activities by Meriam-Webster Dictionary [8]. It is designed to help to analyse malfunctioning of the application's behavior and to identify the part of application source code that leads to an undesired behavior [3]. Kubernetes is a very popular technology, but it does not contain an efficient solution to effectively and easily apply logging technologies in its infrastructure to analyse and store application logs. Although Kubernetes has a built-in solution for gathering such data, e.g. writing to standard output and standard error streams [5], it does not provide the possibility to process these logs, filter, cluster, and visualise them. To cope with this, there exist various solutions, and the most popular are the Elastic stack (Elastic search, Logstash, Kibana, Metricbeat) [9, 14] and the PLG stack (Promtail, Loki, Grafana) [7, 10, 11] with Prometheus. These approaches can help enhance logging and fill the gap of missing functionality in a Kubernetes setup. Moreover, these stacks provide opportunities to alert on application logs, the best examples would be notifying users in Slack, Jira, and Email. However, the application of logging stacks integrated into Kubernetes is popular among DevOps developers and in general among programmers, it lacks scientific research. The goal of this research paper is to analyse and compare these stacks in terms of system metrics. The main focus of this research will be on the implementation of both stacks within Kubernetes infrastructure for a simple Java application on a physical device MacBook Air 18. Moreover, the literature review analysis will be conducted to discover which stack is more relevant in terms of implementation for different use cases.

2 BACKGROUND

2.1 Terminology

Before continuing with the problem statement section, it is necessary to establish the terminology that will be used further in this paper, so the reader not familiar with DevOps has an idea of what was used during the research. Therefore, looking ahead, in this section technologies used will be listed with their tools and supported with respective documentation from official sources. It is worth mentioning that in this section only the description of technologies and tools will be observed, the steps taken to implement them will be described in more detail later in this paper. The list of used technologies and tools:

- Docker
- Kubernetes
- ELK
- PLG & Prometheus
- Java application

2.1.1 ***Docker****.* Docker [4] is an open platform for developing, shipping and running applications on a physical/cloud machine or just a node. It provides the ability to package the application called containerisation. It is worth mentioning that there exist various alternatives to it, such as Buildah, LXD, etc. Docker consists of different objects, each with the objective to make Docker work.

- *Image* is a read-only template with instructions for creating a Docker container.
- *Container* is a runnable instance of an image. It can be created, started, stopped, moved, or deleted using the Docker API or CLI.
- *Docker-compose* is a tool for defining and running multi-container Docker applications. In Docker-compose, a YAML format file is used to configure the application's services.
- *Docker hub* is a hosted repository service provided by Docker for finding and sharing container images, as well as pushing and pulling images.

2.1.2 ***Kubernetes****.* Kubernetes or just K8s, as defined in the Introduction section, is an open-source system for automating the deployment, scaling, and management of containerised applications [6]. Here and after Kubernetes will be referred to as K8s, the official short name. In this section, only the K8s objects used in this research will be described.

- *Pod* is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co- scheduled, and run in a shared context.
- *Kube-proxy* is a K8s network proxy that runs on each node. This reflects services as defined in the K8s API on each node and can do simple TCP, UDP, etc.
- *Minikube* is a lightweight K8s implementation that creates a VM on a local machine and deploys a simple cluster containing only one node.

2.1.3 ***ELK****.* ELK or Elastic stack [14] is a stack of logging solutions made by Elasticsearch company. There exist different products, however, in this paper, only 4 of them will be used.

- *Elasticsearch* is a distributed, RESTful search and analytics engine. Moreover, it centrally stores the data for lightning fast search, fine-tuned relevancy, and powerful analytics that scale with ease.
- *Kibana* is a free and open frontend application that provides search and data visualisation capabilities for data indexed in Elasticsearch and acts as the user interface for monitoring, managing, and securing an Elastic Stack cluster.
- *Logstash* is a free and open server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to chosen stash.
- *Metricbeat* is a data shipper for collecting and shipping various system and service metrics to a specified output destination.

2.1.4 ***PLG & Prometheus****.* PLG stack is a stack of logging solutions provided by Grafana labs company [11]. In addition to that, for purpose of retrieving the system metrics, Prometheus technology provided by SoundCloud company will be used [7].

- *Grafana* is an open-source frontend application to query, visualise, alert on, and understand data pipelined to it.
- *Loki* is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus. It is designed to be very cost effective and easy to operate.
- *Promtail* is an agent which ships the contents of local logs to a private Grafana Loki instance. It primarily is able to discover targets, attach labels to log streams, and push them to the Loki instance.
- *Prometheus* is a free software application used for event monitoring and alerting. It records metrics in a time series database built using an HTTP pull model, with flexible queries and real-time alerting.

2.1.5 ***Java application****.* Java application will be used for testing purposes and represent the logs' spamming application with different log levels, namely info, warn, debug, and error, which is the most popular and widely used. The program was created specifically for this research to pipeline a large number of logs with different chances of appearing. The probability of the log with info level is 50%, both warn and debug levels are with a probability of 20%, and the error logs are with a probability of 10%. Moreover, each level has its number of times to be printed into the console log, namely info - 100, debug - 50, warn - 30, and error - 10. The process iterates infinitely until manually stopped with a delay of 5 seconds between each log's print. The application runs twice in different pods with different configurations.

3 PROBLEM STATEMENT

DevOps requires a more integrated solution for infrastructure and operations, and therefore the need of analysing logs increases. Finding the right solution appears to be challenging and complex, as existing solutions require different resources of a node, have limitations on where they can be deployed, and require substantive knowledge of the development team for proper implementation. In the monolithic applications, the logs are aggregated in one place, while in micro-service architecture the applications have their own logs aggregated in different places in non-persisting containers.

The ELK and PLG stacks can help solve this problem, but they lack scientific research.

The problem statement will lead to the following research question:

What is the difference between ELK and PLG stacks, based on the performance and implementation relevance?

To answer this question, it is necessary to subdivide it into smaller sub-questions. Each subquestion will explore the problem statement in depth observed from different perspectives, namely:

(1) Is there a significant difference between stacks in terms of performance?
(2) What is the prevalence and usage of the ELK stack and PLG stack in various research papers, and how do they compare in terms of their implementations across different domains and use cases?

4 RESEARCH METHODS

This research will be divided into 2 stages according to the subquestions mentioned above. The results of each subquestion will help to answer the research question of this paper. For the 1st subquestion, it is necessary to analyse the performance of each stack deployed on a shared single physical node when integrated into K8s infrastructure for a Java application. To conduct such analysis, specific metrics of a physical node are chosen, including a hypothesis to be tested with these metrics with the aforementioned logger application, providing a practical foundation for the research. For the 2nd subquestion, a systematic literature review will be conducted to understand the relevance of each stack based on different implementation scenarios. A literature review will help to discover which stack is more popular for various implementations. Thus, the focus will be made on the number of appearances in chosen databases, as well as providing an overview of the most popular IT-related sectors where the combination of K8s and either of the stack is used and deployed. It is necessary to mention, that only research papers will be used in the literature review and not the documentation of mentioned technologies and tools.

5 PERFORMANCE ANALYSIS

The performance analysis will consist of 3 hypothesises per each chosen metric to be tested. These metrics are essential for understanding the differences between stacks in terms of performance. The chosen metrics are:

- *CPU Usage*: measure the percentage of the total CPU capacity that is being used when the stacks are deployed. Lower CPU capacity indicates that the stack requires fewer node resources.
- *Indexing Latency*: measure the time taken by each stack to process and index log events. Lower latency indicates faster processing and indexing capabilities.
- *System Load*: system load is a measure of the amount of computational work that a computer system performs. In this case, the processes are running K8s and stacks.

To avoid confusion, it is necessary to understand the difference between CPU Usage and System Load. CPU usage refers to the percentage of time the CPU is actively executing tasks or processing instructions, while System Load represents the amount of work being performed by both running and waiting processes. Moreover, the System Load takes into account not only CPU usage but also other system resources, such as disk I/O, network activity, and memory usage. All three metrics are available in Kibana for the ELK stack and in Grafana for PLG. Each hypothesis will be tested with a suitable statistical method. For research, it is intended to conduct hypothesis testing with two data sets. There exist two methods, i.e. two-sample t-test and paired t-test. Generally, the two-sample t-test is used when the data of two samples are statistically independent, while the paired t-test is used when data is in the form of matched pairs [15]. The method is chosen to be a 2-sample t-test, since the gathered data sets are independently deployed on the node. Moreover, the advantage of this method is that it will help to understand if there is a significant difference between each stack in terms of each metric. All tests will be conducted using the MacBook Air 18 as the node where stacks will be integrated. The data samples will be gathered in a timeframe of 10 minutes.

5.1 Setup Limitations

Before continuing with the setup section, it is necessary to mention the limitations faced during the integration of stacks on the node. The initial plan was to integrate each stack into the K8s minikube cluster and run the Java application at the same time, as stacks are better in performance and simplicity of integration when deployed within the same environment. However, during the integration of the ELK stack into the minikube, it was found that there are not enough resources on the node, as it requires higher CPUs and RAM for deployment. Therefore, it was decided to run the stack as the Docker-compose file within Docker and connect it to the Java application within the minikube. To keep the stacks within the same environment, i.e. in Docker, it was decided to also run PLG in Docker as a Docker-compose file. However, during the implementation of Promtail and the lack of documentation on it, as well as the time limitation given on research, it was not possible to perform the connection between the Java application inside of the minikube to the Promtail running inside Docker. To make PLG work and save time, it was decided to run PLG inside the minikube, as was intended initially. This limitation does not necessarily have a great impact on the performance, but for the sake of research accuracy, it would have been better to have the application run in the same environment.

5.2 Setup

In this section, the setup, made for testing the hypothesises, will be shortly described to give an overview of how Docker, K8s, and both stacks operate with each other. First of all, the Docker desktop application was installed on the node. After that, the K8s' minikube Docker image was pulled into the Docker using the CLI in the node's terminal. As it was mentioned in the limitations section, the ELK stack was deployed within Docker via the Docker-compose file, containing the configurations for Elasticsearch, Kibana, Logstash, and Metricbeat. After that, using the Helm package manager the configuration file containing configurations for Grafana, Loki, Promtail, and Prometheus was deployed as pods within minikube using the CLI in the terminal. After that two different Java applications were dockerised (containerised using Docker), pushed to the Docker hub repository, and then pulled with the help of a configuration file as an image to run a Java application pod within minikube. As it was mentioned earlier, both applications have the same functionality, but different configurations according to Promtail and Logstash documentation. The connection between Logstash and Java application was established using the TCP protocol and managed by kube-proxy for service discovery. The connection between Promtail and Java application was established automatically, as Promtail and Java application are running on the same cluster, i.e. minikube. The deployment of each stack was successful, so Logstash and Promtail can gather Java logs and pipeline them to respective services.

5.3 Hypothesis Description

In this section the description of the hypothesis will be shown and the steps to perform the testing procedure will be listed. The testing procedure will be the same for each metric with respective data sets.

(1) Model: two data samples will be gathered while the application is running within the given timeframe with respect to each stack per each metric

(2) Hypothesis:
   (a) H_0: the mean (μ_{elk}) metric's value of ELK is equal to the mean (μ_{plg}) metric's value of PLG ($\mu_{elk} = \mu_{plg}$)
   (b) H_1: the mean (μ_{elk}) metric's value of ELK is significantly different from the mean (μ_{plg}) metric's value of PLG ($\mu_{elk} \neq \mu_{plg}$)

(3) Significance level: $\alpha = 5\%$

(4) Notation:
   (a) Sample size - n_x
   (b) Sample mean - $\bar{x}_x$
   (c) Sample standard deviation - s_x

(5) Test statistics:
Pooled standard deviation is a measure of the variation, spread or dispersion of the data around the mean, s_p:

$$s_p = \sqrt{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2/(n_1 + n_2 - 1)}$$

Test statistic is a statistical test that is used to compare the means of two groups, t:

$$t = (\bar{x}_1 - \bar{x}_2)/s_p\sqrt{1/n_1 + 1/n_2}$$

(6) Compute p-value: reject H_0 = if the p-value $\leq \alpha = 5\%$

(7) Draw statistical conclusion.

5.4 Results

In this subsection, the results and calculations of statistical tests for each metric will be presented (see Table 1). The data gathered is shown in Appendix A at the end of the paper (see Fig. 3). All calculations are made according to formulas stated in the testing procedure.

	Metrics		
Statistical values	**CPU Usage**	**Latency Indexing**	**System Load**
n	n_{elk} = 10, n_{plg} = 10	n_{elk} = 41, n_{plg} = 41	n_{elk} = 17, n_{plg} = 17
x̄	$\bar{x}_{elk}$ = 40.72, $\bar{x}_{plg}$ = 43.92	$\bar{x}_{elk}$ = 9.3956, $\bar{x}_{plg}$ = 7.8802	$\bar{x}_{elk}$ = 23.0317, $\bar{x}_{plg}$ = 23.5705
s	s_{elk} = 15.659, x_{plg} = 11.701	s_{elk} = 12.529, s_{plg} = 11.431	s_{elk} = 6.3111, s_{plg} = 7.5003
s_p	s_p = 13.822	s_p = 11.9925	s_p = 6.9312
t	t = -0.517682	t = 0.572097	t = -0.226644
p-value	p = 0.610984	p = 0.56886	p = 0.822142
Conclusion	Reject H_1	Reject H_1	Reject H_1

Table 1: Results overview

After the conduction of the hypothesis testing with the stated testing procedure using a two-sample t-test and from the results of testing each metric, it can be concluded that concerning the chosen metrics, there is no significant difference between stacks deployed on the shared physical node. Both stacks were deployed independently and could not affect the performance of each other during the data gathering. All the data were gathered as mentioned above in a timeframe of 10 minutes. The results were expected, as they were deployed on a shared physical node with limited resources and a single cluster. In addition, the stacks are similar in terms of approaches to building a logging solution, i.e. search engines on non-relational databases, using JSON format files for faster and more efficient parsing, etc. Two of the chosen metrics are related to the performance of the physical node when stacks are deployed, and the latency metric is mostly related to the processing powers of the stack itself. From the results, it is seen that even for different kinds of metrics there is no sufficient evidence that there is a significant difference between stacks. Nevertheless, section 8 will describe how various aspects of deployment could have affected the results.

6 LITERATURE REVIEW

In this section literature review analysis will be conducted. The purpose of the analysis is to explore which stacks are more relevant in terms of practical implementation on various cases and to provide an overview of these cases. This will help to provide a theoretical foundation for the research. As it is was mentioned before, the method is chosen to be a literature review, as it will help provide a summary and synthesis of the existing knowledge.

6.1 Search Terms

To conduct a literature review it is necessary to state the search terms. In this research, the focus is mainly on two aspects: K8s and both stacks. Since the objective is to analyse existing literature, a state which stack is more relevant in terms of implementation, and in what are these cases, the search terms would contain "Kubernetes or K8s **and**" any of the following terms:

- ELK
- PLG
- Elasticsearch
- Logstash
- Kibana
- Promtail
- Grafana
- Loki
- Prometheus

6.2 Database Search

In order to locate the relevant research papers, the search was conducted in the following databases: IEEE, ScienceDirect, Wiley, and JSTOR. Moreover, the filters were limited to finding papers starting from 2014, the year when Kubernetes was officially published as open-source software, and to research papers written in English.

6.3 Inclusion & Exclusion Criteria

The inclusion and exclusion criteria are defined to find papers focused on the implementation of ELK or PLG within Kubernetes infrastructure in IT-related sector (see Table 2).

Inclusion criteria	Exclusion criteria
Research must contain only research articles	Research must exclude conference proceedings, news, documentations, and book chapters
Research must contain an example of either of stack implementation within K8s	-
Research must be done in English	-
Research must be done after 2014	-
Full-text is available	-

Table 2: Inclusion & Exclusion Criteria Overview

6.4 Screening Process

The screening process was conducted with an emphasis on inclusion and exclusion criteria and used the aforementioned search terms. Moreover, to filter the gathered research papers PRISMA four-phase flow diagram [12] will be used and a research paper by Carrera-Rivera A. (2022) was used as an additional guidance [2]. All articles were imported into the Mendeley software for screening.

Following the four-phase flow diagram, the following steps were conducted (see Fig. 1). The search resulted in 177 articles from the aforementioned search terms. After that, 69 papers were deleted as duplicates. Next, the papers were assessed and checked for eligibility based on inclusion and exclusion criteria. Out of 108 papers that were left after removing the duplicates, 76 of the papers were removed as assessed as not eligible for this study. Therefore, only 32 papers will be analysed in the literature review.

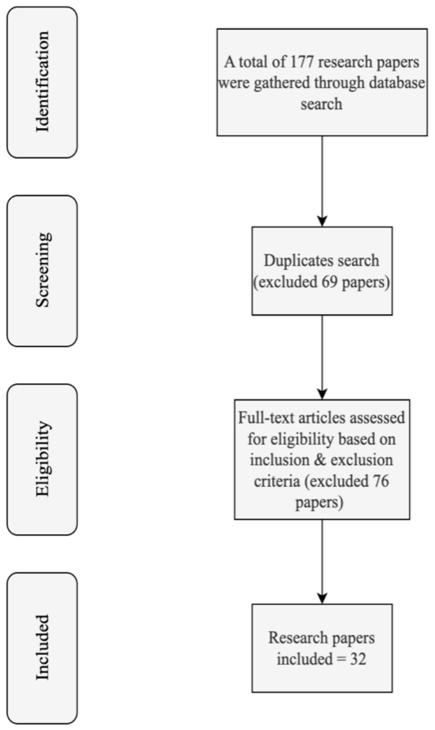

Identification
A total of 177 research papers were gathered through database search
Screening
Duplicates search (excluded 69 papers)
Eligibility
Full-text articles assessed for eligibility based on inclusion & exclusion criteria (excluded 76 papers)
Included
Research papers included = 32

Figure 1: Four-phase flow diagram

6.5 Results of Usage Prevalence

The analysis of the included research papers shows that all of the papers after 2017, with the biggest rate of 30.03% were published in 2021. All of these papers contain information on the implementation of Kubernetes and either stacks or any technology from these stacks. In the analysis, the results were categorised into either the full stack or some of the technologies from these stacks, or the mix of technologies from both stacks. The analysis concludes that in 53.1% ELK stack or any technology included in the stack was used, in 31.3% PLG stack or any technology included in the stack was used including Prometheus, and in 15.6% the mix of technologies included in both stacks were used (see Fig. 2), which mostly was the mix of Elasticsearch and Prometheus. It can be seen that in over a half of use cases, the ELK stack was implemented for particular solutions.

6.6 Results of Stacks' Implementation

In this section, the included research papers were analysed in terms of the use cases in which the stacks were implemented. The analysis showed that the use cases' topics are extremely diverse and capture different domains. The largest number of papers were related to the Monitoring and Cloud-edge Environment (3 papers per each). The next were Security Architecture, Data Analysis, Networking, Blockchain, and VMi Management (2 papers per each). Lastly, the use cases‘ topics that were mentioned only once in the research papers' list and could not be clustered further are Trigger

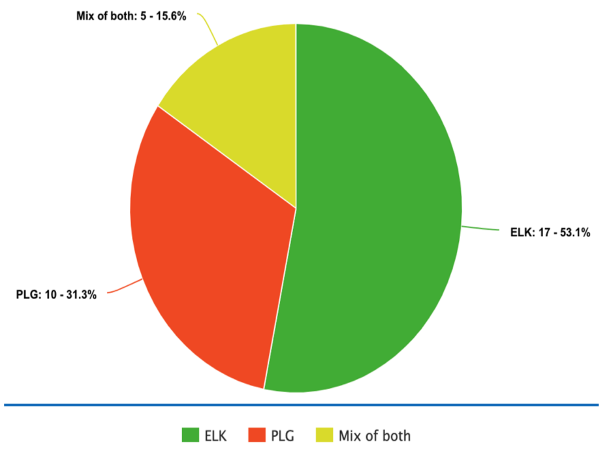


Figure 2: Pie chart of technologies used in papers

System (Physics related), Data Visualisation, Data Management, Automatic DFD Extraction, Machine Learning, Log analysis, Malware Exposure, Cloud Scheduling, Cloudification Middleware, Workload Simulation, Integration with Raspberry Pi, Auto Scaling System, and Science Platform (Astronomy) (1 paper per each). From such a distribution, we can derive possible patterns if there is any trend for a particular stack usage for particular use case (see Table 3), use cases mentioned only once will not be considered.

	Stack Usage per 1 paper		
Monitoring	Prometheus & Grafana	Mix of both full stacks	Prometheus & Grafana
Cloud-edge environment	Elasticsearch & Kibana	Elasticsearch & Kibana	Grafana
Security Architecture	Full ELK stack	Elasticsearch	-
Data Analysis	Full ELK stack	Full ELK stack	-
Networking	Prometheus & Grafana	Elasticsearch & Kibana	-
Blockchain	Elasticsearch & Kibana	Elasticsearch & Logstash	-
VMi Management	Full ELK stack	Prometheus	-

Table 3: Stack usage across domains

It can be observed from the table that the prevailing stack is ELK. For instance, in Data Analysis-, Security Architecture-, and Blockchain-related use cases the authors of the research prefer using either a full ELK stack or a combination of ELK products. In the case of PLG, it can be seen that this stack and its products separately prevail in Monitoring-related use cases. For the other use cases, it can be seen that there is no specific preference over the stack, so both ELK and PLG products were used to implement particular use cases.

To summarise, the literature review was conducted, and out of 177 located research articles, 32 were eligible to be analysed concerning the inclusion and exclusion criteria. Based on the analysis, there is a large prevalence and usage of ELK over PLG in various research papers (over a half = 53.1%). Moreover, when compared to specific use cases, the ELK still shows a higher rate of usability and implementation in similar and different use case domains.

7 RECOMMENDATIONS

In this section, the recommendations concerning the stacks will be discussed. ELK and PLG have different implementations in terms of the languages they were written in, thus they provide different functionality as well as compatibility with other technologies. Therefore, to implement a particular solution for a specific use case, it is necessary to understand how each product within the stack works, which functionality it provides, and which resources are needed for deployment. It is needless to say, that the substantive knowledge of Docker and Kubernetes is required from the development team to be able to manage the containers and pods and configure them. From a personal perspective, the ELK stack seems more complete as all solutions are part of a single stack and have their necessary functionality, which helps to reach a goal without separate products. Other products can be deployed as well if needed, but usually, it is not the case. However, the PLG stack provides a larger flexibility in terms of different implementations, as a great number of products can be connected to Grafana, and these products can be mixed to achieve a required objective. Another advantage of Grafana against Kibana is the dashboard flexibility since it provides the opportunity to create dashboards with required metrics and parameters. Nevertheless, to be able to create such a dashboard it is required to know Grafana QL language, which is very time-consuming and was a limitation during this research. It is certain that stacks serve the common solution and are extremely popular among developers, to be able to efficiently and correctly deploy them, it is required to refer to its documentation.

8 EVALUATION

This section elaborates on the limitations faced during the research. The main limitation is related to the given timeframe of the research. Within the 8 weeks provided to conduct the research, most were spent on getting familiar with Docker, Kubernetes, ELK, and PLG products, and for the deployment of it on the node. Docker and Kubernetes are the backbones of DevOps development, and it is crucial to obtain enough knowledge to be able to deploy applications as well as configure them. Networking knowledge is required as well, since all the Docker and Kubernetes objects, e.g. minikube, have their networking parameters, such as IP addresses, ports, etc. Without this knowledge the implementation becomes challenging.

(Regarding the ^{1st} subquestion)

- The lack of MacBook Air 18 resources led to the inability to do quick manipulations with the Docker and Kubernetes objects, such as starting a pod/container, stopping it, restarting, and deleting. For instance, sometimes the process of stopping the set of containers where ELK was deployed could take up to 15-20 minutes or stuck in an infinite loop of being stopped.

- ELK (Kibana) and PLG (Grafana) provide different implementations of metrics provision. Kibana has a default set of dashboards with important metrics to explore the performance of either CPU usage or Networking, Grafana does not have such default dashboards. Such dashboards can only be either manually created (within a given timeframe it would impossible) or found suitable in the dashboard marketplace (dashboards are free). Nevertheless, searching for dashboards that would match the ones Kibana provides was challenging, as most of them require their own tools setup, such as InfluxDB and Telegraf, which are not included in this research.

(Regarding the 2^{nd} subquestion)

- The timeframe provided for this research led to the reduced size of literature review sample size. Without this limitation, the research with a larger number of research papers might have been statistically more accurate and, thus, be more representative not only in terms of the stacks' usage and implementation but also in which domains these stacks were deployed.

9 CONCLUSION

In this paper, the two most popular stacks of logging solutions were considered and analysed from two different perspectives. For the practical foundation, the statistical analysis was conducted based on the chosen metrics to establish if there is any significant difference between the stacks in terms of performance. The chosen statistical method was a two-sample t-test. The results of the hypothesis testing showed, that for each metric there is no significant difference between the stacks. For the theoretical foundation, the literature review was conducted to establish which stack prevails in terms of usability in various research papers, and compare them in terms of their implementations across different domains and use cases. The analysis showed that the ELK stack prevails in terms of usability across different research papers showing 53.1%, whereas the PLG showed only 31.3%, and the mix of the products implemented showed 15.6%. From the clustering of the use cases where the stacks were implemented, it was discovered that the ELK also prevails in terms of usability, especially in such cases as Data Analysis, Security Architecture, and Blockchain. Nevertheless, the PLG products were used with the same amount of appearance as ELK (for instance, in Networking it is 50/50: Grafana and Prometheus and Elasticsearch and Kibana). In conclusion, answering the main research question of this paper, the results showed that in terms of practice, there is no significant difference between the stacks, and at the same time there is a large prevalence and usage of ELK over PLG in various research papers, as well as when compared to specific use cases, the ELK still shows a higher rate of usability and implementation in similar and different use case domains.

9.1 Future Work

The extended timeframe and support from DevOps professionals would hugely improve the accuracy of results.

- The extended timeframe would highly increase the sample size to conduct a stronger literature review.
- It is necessary to have a team of proficient developers to be able to ask for a piece of advice and recommendation on how to properly set up the technologies and deploy them.

This would enhance the analysis and show more useful and valuable results. Moreover, in the future, such analysis would help to formulate a methodology for developers, such as guidance on how to deploy and in which domains which stack is more suitable.

REFERENCES

[1] Omar Al-Debagy and Peter Martinek. 2018. A Comparative Review of Microservices and Monolithic Architectures. 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), 000149–000154. https://doi.org/10.1109/CINTI.2018.8928192

[2] Angela Carrera-Rivera, William Ochoa-Agurto, Felix Larrinaga, and Ganix Lasa. 2022. How-to conduct a systematic literature review: A quick guide for computer science research. *MethodsX* (2022). https://doi.org/10.1016/j.mex.2022.101895

[3] Monika Dávideková and Michal Gregu Ml. 2016. Software Application Logging: Aspects to Consider by Implementing Knowledge Management. In *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE, 102–107. https://doi.org/10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00087

[4] Docker development team. 2023. *Docker Official Documentation*. https://docs.docker.com/

[5] Kubernetes development team. 2023. *Kubernetes logging solutions*. https://kubernetes.io/docs/concepts/cluster-administration/logging/

[6] Kubernetes development team. 2023. *Kubernetes Official Documentation*. https://kubernetes.io/

[7] SoundCloud development team. 2023. *Prometheus Official Documentation*. https://prometheus.io/docs/introduction/overview/

[8] Merriam-Webster Dictionary. 2023. Definition of Log. http://www.merriam-webster.com/dictionary/log

[9] Yuvraj Gupta and Ravi Kumar Gupta. 2017. *Mastering Elastic Stack*. Packt Publishing Ltd. https://books.google.nl/books?id=EVQoDwAAQBAJ&lpg=PP1&ots=-0K4beEdeo&dq=Mastering%20Elastic%20Stack%20By%20Yuvraj%20Gupta%2C%20Ravi%20Kumar%20Gupta&lr&pg=PP1#v=onepage&q=Mastering%20Elastic%20Stack%20By%20Yuvraj%20Gupta,%20Ravi%20Kumar%20Gupta&f=false

[10] Matti Holopainen. 2021. Monitoring Container Environment with Prometheus and Grafana. (2021). https://www.theseus.fi/bitstream/handle/10024/497467/Holopainen_Matti.pdf?sequence=2

[11] Grafana lab development team. 2023. *Grafana lab Official Documentation*. https://grafana.com/docs/

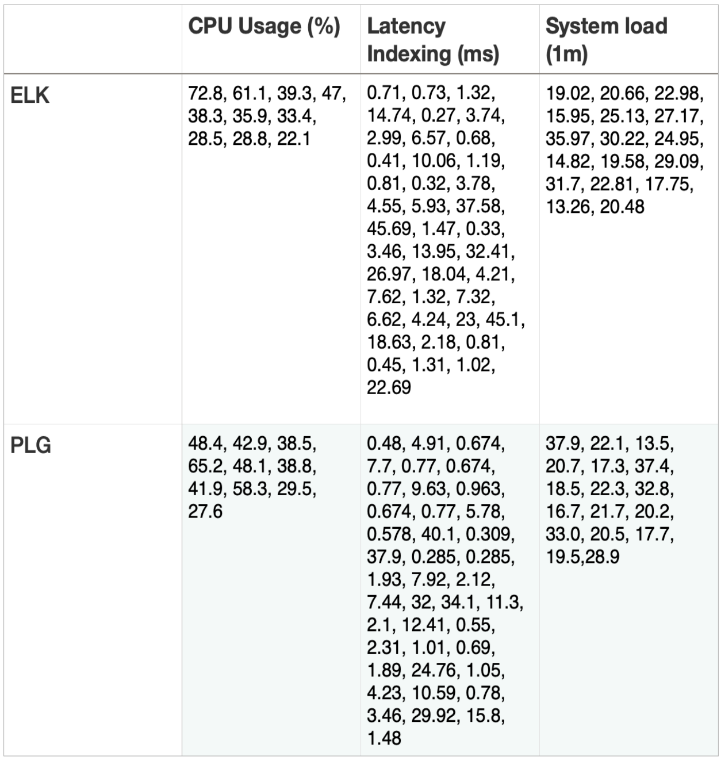
[12] David Moher, Alessandro Liberati, Jennifer Tetzlaff, and Douglas G. Altman. 2010. Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. *International Journal of Surgery* 8, 5 (2010), 336–341. https://doi.org/10.1016/j.ijsu.2010.02.007

[13] James Shore and Shane Warden. 2021. *The art of agile development*. O'Reilly Media, Inc. https://books.google.nl/books?id=kXZIEAAAQBAJ&lpg=PP1&ots=M1TeAuVz4b&dq=agile%20development&lr&pg=PP1#v=onepage&q=agile%20development&f=false

[14] Elastic stack development team. 2023. *Elastic stack Official Documentation*. https://www.elastic.co/guide/index.html

[15] Manfei Xu, Drew Fralick, Julia Z Zheng, Bokai Wang, Xin M Tu, and Changyong Feng. 2017. The differences and similarities between two-sample T-test and paired T-test. *Shanghai Arch. Psychiatry* 29, 3 (June 2017), 184–188. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5579465/

A DATA OVERVIEW



	CPU Usage (%)	Latency Indexing (ms)	System load (1m)
ELK	72.8, 61.1, 39.3, 47, 38.3, 35.9, 33.4, 28.5, 28.8, 22.1	0.71, 0.73, 1.32, 14.74, 0.27, 3.74, 2.99, 6.57, 0.68, 0.41, 10.06, 1.19, 0.81, 0.32, 3.78, 4.55, 5.93, 37.58, 45.69, 1.47, 0.33, 3.46, 13.95, 32.41, 26.97, 18.04, 4.21, 7.62, 1.32, 7.32, 6.62, 4.24, 23, 45.1, 18.63, 2.18, 0.81, 0.45, 1.31, 1.02, 22.69	19.02, 20.66, 22.98, 15.95, 25.13, 27.17, 35.97, 30.22, 24.95, 14.82, 19.58, 29.09, 31.7, 22.81, 17.75, 13.26, 20.48
PLG	48.4, 42.9, 38.5, 65.2, 48.1, 38.8, 41.9, 58.3, 29.5, 27.6	0.48, 4.91, 0.674, 7.7, 0.77, 0.674, 0.77, 9.63, 0.963, 0.674, 0.77, 5.78, 0.578, 40.1, 0.309, 37.9, 0.285, 0.285, 1.93, 7.92, 2.12, 7.44, 32, 34.1, 11.3, 2.1, 12.41, 0.55, 2.31, 1.01, 0.69, 1.89, 24.76, 1.05, 4.23, 10.59, 0.78, 3.46, 29.92, 15.8, 1.48	37.9, 22.1, 13.5, 20.7, 17.3, 37.4, 18.5, 22.3, 32.8, 16.7, 21.7, 20.2, 33.0, 20.5, 17.7, 19.5,28.9

Figure 3: Data used in the Hypothesis testing