LEVERAGING MICROSERVICE END-TO-END TRANSPARENCY IN DEVELOPMENT

Lappeenranta–Lahti University of Technology LUT

Bachelor's programme in Software Engineering

2022

Aatu Laitinen

Examiner: Assistant Professor Antti Knutas

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

School of Engineering Science

Software Engineering

Aatu Laitinen

Leveraging microservice end-to-end transparency in development

Software engineering bachelor's thesis

41 pages, 5 figures and 2 tables

Examiner: Assistant Professor Antti Knutas

Keywords: microservice, transparency, monitoring, development, alerting, survey

Microservice architecture has emerged as a widely adopted way of breaking software into independent components that are more agile to develop and manage. However, as the microservice systems grow larger they are faced with increasing complexity and difficulty to govern. This has created the need for monitoring microservices to form better transparency of the system and how its components work together.

In this work we researched scientific literature to find out how microservice developers can utilize the different aspects of microservice monitoring to improve their development process. We research what aspects of monitoring can bring the most value to the development and furthermore to the customers of the services. The findings from the literature review are summarized to a survey. The survey is conducted on a small group of developers working in various development teams of a Finnish software development company. To create a comprehensive understanding on how the monitoring aids microservice development in practice, the results are collated and analyzed.

The results indicate that microservice monitoring has various uses for improving microservice development. Both the literature review and the survey state that collecting metrics, logs and traces through monitoring and combining them increases the speed and quality at which the microservices can be continuously integrated and deployed. Because the survey was conducted with only 15 developers who all are part of the same company, it affects the validity of the research results and thus they should be taken as directional.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikka

Aatu Laitinen

Mikropalveluiden toiminnan end-to-end läpinäkyvyyden hyödyntäminen kehityksessä

Tietotekniikan kandidaatintyö 2022

41 sivua, 5 kuvaa ja 2 taulukkoa

Tarkastaja: Apulaisprofessori Antti Knutas

Avainsanat: mikropalvelu, läpinäkyvyys, monitorointi, kehitys, hälytys, kysely

Mikropalveluarkkitehtuuri on viime vuosina noussut yleisesti käytetyksi tavaksi pilkkoa ohjelmistoja pienempiin itsenäisiin osiin, jotka ovat ketterämpiä kehittää ja käsitellä. Näiden mikropalveluohjelmistojen kasvaessa suuremmiksi, myös niiden monimutkaisuus kasvaa ja ohjelmistokokonaisuudesta tulee vaikeampi hallita. Tämä on luonut tarpeen monitoroida mikropalveluiden toimintaa, jotta voidaan paremmin hahmottaa ja hallita ohjelmistokokonaisuutta ja sitä miten eri osat toimivat yhteen.

Tässä työssä käydään läpi tieteellistä kirjallisuutta mikropalveluiden monitoroinnista ja selvitetään kuinka kehittäjät voivat hyödyntää mikropalveluiden monitorointia palveluiden kehityksessä. Työssä tarkastellaan sitä, mitkä monitoroinnin osa-alueet edistävät parhaiten kehitystä ja tätä kautta tuovat eniten arvoa palveluiden asiakkaille. Kirjallisuuskatsauksen tulokset tiivistetään kyselyksi. Tämä kysely jaetaan pienelle ryhmälle kehittäjiä, jotka kaikki toimivat suomalaisen ohjelmistoyrityksen eri kehityssoluissa. Kirjallisuuskatsauksen ja kyselyn tuloksia käydään läpi, jotta saadaan luotua kattava käsitys siitä, miten mikropalveluiden monitorointia voidaan käytännössä hyödyntää palveluiden kehityksessä.

Tulosten perusteella mikropalveluiden monitorointi edistää palveluiden kehitystä usealla eri tavalla. Kirjallisuuskatsaus ja kysely molemmat toteavat, että metriikoiden, sekä seuranta- ja lokitietojen kerääminen ja yhdistäminen edistää palveluiden jatkuvan integroinnin ja julkaisemisen nopeutta ja laatua. Koska kysely toteutettiin vain 15 kehittäjän kanssa, jotka kaikki toimivat samassa yrityksessä, tuloksia voidaan pitää vain suuntaa antavina.

LIST OF ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
SaaS	Software as a Service
SME	Small and Medium-sized Enterprise
UI	User Interface

TABLE OF CONTENT

1 Introduction

Monoliths have been an architectural standard in software development since the beginning of web development. Monoliths are single-based applications that are built and deployed all at once. Since the monoliths are single-based the entire system needs to be stopped and restarted to make any changes to them. This architecture makes it difficult to scale, understand and maintain monoliths. In recent years, microservices have emerged as an efficient way to manage and scale monoliths. (Brondolin & Santambrogio 2020) Microservices make it possible to create and deploy components independent from the monolith and have the monolith use them to extend its functionality (Eski & Buzluca 2018). This also allows companies to scale their monoliths without the dependency to use the same outdated languages and technologies that might be used in the monolith. Microservice components can also be developed and deployed without having to stop the monolith or any other service.

The speed at which companies need to be able to answer customer needs and issues has increased tremendously. The implementation of microservice architecture enables companies to utilize continuous integration and delivery to answer these needs. (Chen 2018) Microservices usage of cloud-based containers allows the services to be run in their own isolated environments. This architecture allows companies to make changes to microservices and deploy them in rapid succession. Because of the isolated nature of these containers, they need to communicate between other components and containers to form a unified system (Sabharwal & Pandey 2020, pp. 1-2). This can create large and complex communication pipelines where the information travels from one container to another through multiple different routes. The increasing complexity of these information routes combined with the short lifespan of containers and the high elasticity of the services is what creates a need for monitoring microservices (Pina et al. 2018).

The amount of information that can be monitored about microservices is high and there are multiple domains in the system where data can be monitored (Chakraborty & Kundan 2021, pp 11-14). For example, various metrics can be monitored from the containers on which the service runs. Logs about failed responses are attainable from the application domain. Also, trace data about the movement of information can be gathered from the services domain. Whether this collected data is applicable depends on how the data can be presented, and who is inspecting the data. An infrastructure architect responsible for integrating the microservices can have vastly dissimilar needs for the monitored data than a developer responsible for the functionality of the microservice. All this leads to the research problem of this work:

i. How can microservice end-to-end transparency be leveraged to improve the continuous integration and delivery of these services?

There are differences in environment and functionality for microservices. In this work, we will focus more specifically on API (Application Programming Interface) microservices, running on AWS (Amazon Web Services) cloud-based containers. We also concentrate on the developer side of the monitoring-related problem and try to identify ways for them to utilize monitored data to aid microservice development. This leads to the subproblems of this study:

i. What are the most essential factors of microservice monitoring from the microservice developer's point of view?

ii. How can different monitored information and its presentation benefit developers when developing API microservices?

This work will be done in cooperation with a Finnish software company specializing in financial management and enterprise resource planning. The company provides SaaS (Software as a Service) solutions. They have adopted microservice architecture to help with managing monolithic applications. In this work, developers working at this company's development teams are surveyed to get answers for the research problems. To preserve the anonymity of the company, it will be referred to as Finnish SaaS company in this work.

2 Literature review of microservices and monitoring tools

This chapter takes a look into the microservice architecture and how microservices function. We will also inspect the technologies used to monitor microservices at the Finnish SaaS company and examine previous research on the subject.

2.1 Microservice architecture

Microservices are small-scale independent software components that are individually deployed, managed, and scaled. Microservices work for a single well-defined responsibility or purpose. Microservices communicate through lightweight messages, often HTTP (Hypertext Transfer Protocol) API calls, to form larger distributed systems (Bucchiarone et al. 2020, p. 7). The autonomy of microservices allows for high scalability, since the microservices can be updated and new services can be deployed without the need to restart or redeploy the other services in the same application. Because microservices are run in isolated environments and communication is done in a comprehensive matter, microservices can be implemented in any technology. This means that they are not bound by the language or framework of the service that uses it. (Rajasekharaiah 2021, pp 13-19.)

To allow independent, instant, and on-demand instantiation of microservice applications, containerization is used for running microservices. Containerization is considered mandatory for running cloud-based microservice applications. (Rajasekharaiah 2021, pp 123-125.) Containers are used to package and deploy small parts of applications into the cloud using lightweight virtualization. Most container solutions are Linux-based. Container engines, with Docker being the most popular, are used to manage dependencies and deployments of the containers. Docker uses images made from layered file systems to form application services. These services are formed by groups of containers of the same image. Docker handles the scaling of the service by increasing and decreasing the number of containers of the same image. (Pahl et al. 2019.)

Kubernetes is a platform used for managing containerized microservices. Kubernetes is used to ensure that there is no downtime for the containers in an application. Kubernetes can distribute network traffic to stabilize the application deployment. It can also be used to automatically mount different storage systems to the application. Kubernetes can automatically deploy new containers and remove existing ones to appropriately distribute the resources. Sensitive information such as tokens and secrets can also be managed by Kubernetes without the need to rebuild container images. Kubernetes will also restart and replace malfunctioning containers to maintain the health of the containers. (Kubernetes 2021a.)

Kubernetes uses pods to manage containers. Pods are a group of one or more containers that share network resources and storage space. Pods contents are run in a shared context with concurrent scheduling and shared location. The containers in the same pod can communicate with each other and correlate their termination. A controller for workload resources is what handles the deployment and health of the pods. (Kubernetes 2021b.)

2.2 Microservice monitoring

There are many different tools for monitoring microservices and presenting monitored data. This work is going to focus on the tools used at the Finnish SaaS company and the data they gather. The information monitored from microservices is divided into three different types. These types are log information, metrics, and trace data.

Grafana is an open-source observability tool for time-series data query and visualization. Grafana uses query editor and independent data storage to query time-series-based data from multiple different sources. It allows the visualization of the data using dashboards. Grafana visualizes metrics in different forms and provides a large number of different visualization options. Grafana also provides an alerting tool that enables the use of various thresholds to recognize critical situations. Upon encountering a critical situation Grafana can send alerts

about it through different communication channels such as email or text messages. (Chakraborty & Kundan 2021, pp. 187-188)

Grafana dashboard is a UI (User Interface) consisting of organized panels that are used to represent the metrics data. Every panel can be configured to utilize any configured Grafana Data Source. The dashboards can be constructed and customized to display the preferred graphs and to suit your needs. Dashboards also provide several time-range options to filter and manage the displayed results. (Grafana 2022)

Promtail is a Grafana client that is used to transfer local log information to a private cloud instance of Grafana Loki. It can be deployed to find targets, that are discovered from log lines produced by the monitored application. The discovered targets can be parsed to a more accurate format and labeled for improved identification. Promtail continuously reads the logs from the processed targets and ships them in batches to the Grafana Loki instance. (Grafana 2022.)

Prometheus is a toolkit for system monitoring and alerting. Prometheus collects metrics as data points indexed in time order. The metrics information is stored by timestamps, and optionally with key-value pairs. Prometheus metrics provide analytical information about the state of Kubernetes pods, service response time, and the performance of HTTP endpoints (Sabharwal & Pandey 2020, pp. 88-89). Prometheus provides PromQL query language that can be utilized to query metrics from Prometheus in a variety of forms. Prometheus can also create and push alerts through Alertmanager to send notifications about system anomalies. A visualization of how Prometheus works and how Grafana can utilize its data through PromQL is displayed in figure 1. (Prometheus 2022.)

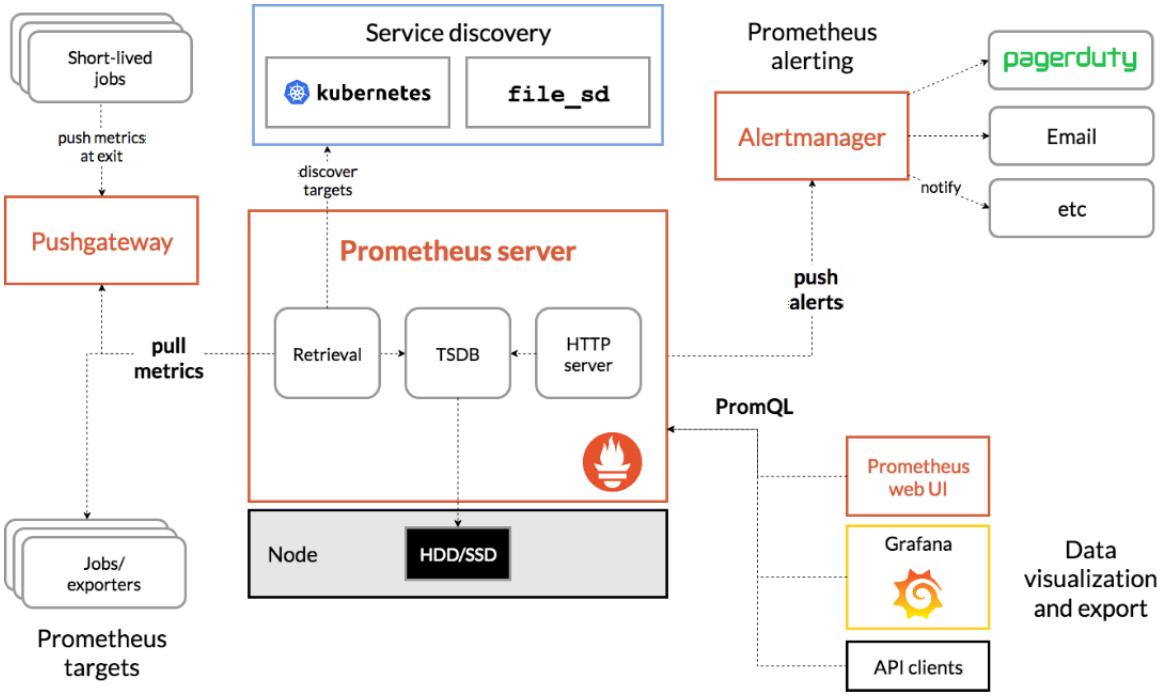

Figure 1. Prometheus architecture illustration (Prometheus 2022)

OpenTelemetry provides a way to generate, emit, collect, process, and export end-to-end telemetry data from microservices. OpenTelemetry Collector is used to collect trace data and create and manage spans that show the end-to-end latency for an entire request. This happens through the "tracer" interface. More specifically, a trace is a tree of spans with a single root span and all its child spans. Tempo, a Grafana tracing backend, receives trace data from OpenTelemetry and allows Grafana to utilize it. (Grafana 2022) OpenTelemetry Collector can be deployed as a Collector instance running with the application or with the same host, or multiple instances can be run as an independent service. (OpenTelemetry 2021)

2.3 Previous research on microservices and monitoring

Microservices and their monitoring are still relatively recent technologies, but they have quickly become widely adopted in the industry. This has also spawned numerous books and articles on the topic. A lot of this literature focuses on the architecture and implementation of microservices. "Microservices Science and Engineering" is a book by Bucchiarone et al. (2020) that centers around microservice architecture. The six-part book gives insight into the

benefits and challenges of microservice migration. It also tackles common anti-patterns and problems with microservices. In addition, the book provides case studies and research conducted on the successful implementation of microservices. A peer-reviewed magazine and scientific journal IEEE (Institute of Electrical and Electronics Engineers) software from IEEE computer society has various articles on microservices such as “Microservices” by Larrucea et al. (2018). The article presents a compact overview of microservices and the technologies behind them. It also provides five key guidelines for microservice integration.

There are also various literature pieces that study the monitoring side of microservices and focus on the tools and implementations to achieve it. Chakraborty and Kundan (2021) introduce modern monitoring methods for containerized cloud-native applications in their book “Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software”. They first provide knowledge of cloud-native applications and the fundamentals of container monitoring on Kubernetes orchestrated environments. The latter part of their work provides insight and implementation examples for popular open-source tools like Prometheus and Grafana that are also present in this work. Sabharwal and Pandey (2020) in their book “Monitoring Microservices and Containerized Applications” give an overview of running microservices in containers with Docker and Kubernetes architectures. They provide a guide on how to construct an environment for microservices to be run on and how to set up Prometheus and Alert manager for recording information about the microservice containers. Usage of Prometheus’ query language (PromQL) is demonstrated along with Grafana Dashboard. They also present the use of two different monitoring solutions in Dyntrace and Sysdig as well as address the automation and orchestration of container monitoring.

3 Research design

This chapter focuses on reviewing literature around microservice monitoring to find background information on the matter. That research is then utilized to conduct a field study which aims to gather further insight into the research problems. Both the literature review and the field study work as a base for discussion around answering the research problems.

3.1 Literature review as a research method

In this work a literature review is undertaken to get a grasp of the best microservice monitoring practices and ways to utilize them. A literature review is a summarization and evaluation of literature on a specific topic (Knopf 2006). According to Branley, Seale & Zacharias (2004), there are three stages in a literature review. The first stage of the literature review in this work is to search scientific literature about microservice monitoring. This is done by using the Google Scholar search engine and LUT Primo scientific library. The second stage consists of reading through the abstracts and summaries of the found literature and discovering materials that contain useful information for the research. In the third stage, the materials that have been determined useful will be read with more focus to gain a deeper understanding of the subject. When new aspects are found through the reviewed literature that seem potential and need further research, the three steps are repeated to deepen the understanding of that aspect. The main structure of the research process is presented in figure 2.

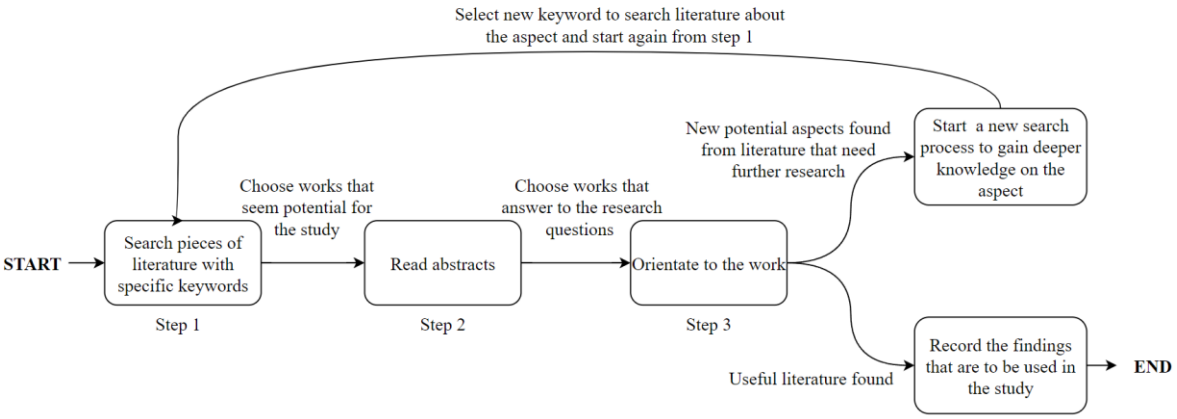

Figure 2. Literature review process

Using carefully selected keywords to identify important literature is the most common way when conducting literature reviews (Cronin, Ryan & Coughlan 2008). In this work, the main keywords to be used are “microservices” and “monitoring”. These keywords should narrow the literature to reflect the main components of the research. To further narrow down the literature, the keywords “containerized”, “development” and “tracing” are used to find pieces surrounding the research problems. In the case of small individual problems that require more particular research material, keywords linked to the problem will be used.

The literature review resulted in 19 different references that proved to be useful for this study. All the reviewed literature and the number of times they were referenced in this study can be seen in table 1. The list includes only references that were made in subsection 3.2 of this study.

Table 1: Reviewed literature by number of references

Number	Reviewed literature	Number of references
1.	Chakraborty & Kundan 2021	11
2.	Li et al. 2022	4
3.	Ewaschuk 2013	2
4.	Beyer et al. 2016	2
5.	Riti 2018	2
6.	Clément et al 2022	1
7.	Gribaudo, Iacono, & Manini 2017	1
8.	Hunter 2017	1
9.	Jamshidi et al. 2018	1
10.	Liu et al. 2019	1
11.	Mace 2017	1
12.	Mi et al. 2012	1
13.	Rooney & Heuvel 2004	1
14.	Sabharwal & Pandey 2020	1
15.	Santana et al. 2019	1
16.	Swersky 2018	1
17.	Sayfan 2017	1
18.	Watts et al. 2019	1
19.	Yocum 2021	1

The most referenced reviews used in this study were the book by Chakraborty and Kundan (2021) and the article by Li et al. (2022). Chakraborty's and Kundan's (2021) book was referenced ten times which means that the literature review is significantly influenced by their work. Other works were almost entirely referenced only once. Many of these references were done to compare the findings from Chakraborty's and Kundan's (2021) book with the rest of the references focusing on other aspects absent in the book.

3.2 Literature review on utilizing monitoring to improve development

To be able to answer the question of whether monitoring can be used to improve microservice development, we will look at different ways of benefiting from microservice

monitoring. We will also look at challenges found in microservice monitoring as well as common needs for a monitoring system.

3.2.1 Challenges in microservice monitoring

With the introduction of containerization, modern cloud-based microservices environments have evolved to be increasingly complex. The ever-changing structure of microservice applications is too hard to monitor using standard tools. Monitoring systems need to be able to gather data from transient services in addition to monitoring constant services. (Sabharwal & Pandey 2020, p. 15) In order to get a clear picture of the health of the microservice-architectural application and how it operates, a high enough transparency of the application as a complete system is required. To achieve this transparency, observability is needed. Observability is a concept that focuses on clarifying the internal operations of a system using the available external information the system provides. (Chakraborty & Kundan 2021, pp. 25-26) Three pillars of Observability, as Chakraborty and Kundan (2021, pp. 27-28) describe them, are metrics, traces and logs. All these, when combined, provide an insight into the distributed microservice system. They also help with narrowing down the areas that should in practice be looked into.

The amount of information that the current monitoring technologies provide from microservices can become overwhelmingly large for developers to filter what information is crucial and what is redundant (Jamshidi et al. 2018). What adds even more complexity to this challenge is the possibility of conflicts in quality requirements for different microservices. Developers require tailored monitoring visualizations that filter the essential information from the redundant in their specific microservice environment. They also need clear requirements to distinguish when to act on the gathered information.

There are three ways of implementing log information monitoring into a microservice system. The most intrusive and effort demanding of these ways is manually coding the logging system into the source code. This is an easy and flexible way to implement logging,

but it can become burdensome to implement and manage with larger systems. A less intrusive technique is to use a tracing framework. They require less effort to use since developers only need to implement the invocations the framework uses. But the frameworks are less flexible with handling individual needs for different services. Dynamic binary instrumentation is a non-intrusive technique and thus requires no additional effort from the developer. A runtime agent is used to dynamically inject instrumentation code that provides the logging functionalities. The downside to this technique is that it is language-dependent and can only be used with languages that support it. Dynamic binary instrumentation also lacks flexibility. (Li et al. 2022) Knowing which technique to use requires deep knowledge about the monitored system and what kind of needs there are for logging information.

3.2.2 Common monitoring needs

Riti (2018, p. 170) defines two core questions that a microservice monitoring system should be able to answer. The first question to be answered is what is broken within the microservice system and the second is why it is broken. Chakraborty and Kundan (2021, p. 16) detail three common needs for a microservice monitoring system which are early problem detection, performance measuring and availability ensuring. Being able to quickly detect problems and why they are occurring, most preferably before they have even happened, is a core need for a microservice monitoring system. In order to have efficient pre-emptive or reactive countermeasures to problems within the microservice system, gathering data about the service is mandatory.

The applications can hold a large number of different microservices that need a virtualized network to interact with each other. Because of this, the substandard performance of a singular service can influence the performance of the entire system. (Gribaudo, Iacono & Manini 2017) This is why it is important to ensure that the services meet their desired performance levels. The visualization of the performance trends and patterns is used to investigate the locations and times for performance drops in the microservice ecosystem. This can result in faster resolving of the performance problems which is why performance measuring is a common need in microservice monitoring.

Akin to performance drops affecting the entire system, the outages of singular microservices also affect the functionality of the entire system. This is because the system cannot access all the microservices it needs to function properly. Since the outage means that the application can be partially or even completely unusable for users, availability ensuring is arguably the most important need for microservice monitoring. (Chakraborty & Kundan 2021, p. 17) Availability monitoring can be used as a pre-emptive method to detect microservices with a high chance of outage in the future. With this monitoring data, predictive container health maintenance can be performed or automated. Kubernetes can be configured to detect and report pod and container problems with the Node Problem Detector (Sayfan 2017, p. 59).

One key feature that is sought after in microservice monitoring systems is the ability to set up alerts for various events. Specific thresholds can be established for monitored data that when crossed will raise an alert. The alert is sent to the assigned personnel and information feeds to notify of the unusual event in the microservice system. (Swersky 2018) This makes it possible for the developers responsible for the service to receive notification in real-time if something has gone wrong in the system or requires additional attention.

According to findings made by Ewaschuk (2013), alerting should be aimed to inform about imminent or occurring problems in a service. The number of different alerts should be kept at as few as possible and the alerts should represent problems that directly affect the end-users. This prevents the alerting channels from flooding with unnecessary alerts. Ewaschuk (2013) also states that the underlying problem should always be classified in the alert to help with figuring out the causes for the problems. The causes themselves should usually not be alerted, because their amount makes it increasingly complicated to catch them all, and you would still need to know what the cause affects.

3.2.3 Metrics utilization

A study conducted by Watts et al. (2019) delves into the metrics gathered from containers by Prometheus and how they can help at detecting anomalies and creating preventive actions based on them. The first set of metrics can be gathered from Dockers daemon engine. Prometheus can document information about the Docker system with "engine_daemon_engine_info", which can provide useful information for finding anomalies. Anomalies in docker containers can cause an increase in resources allocated to the affected container. A metric called "engine_daemon_engine_memory_bytes" displays how much resources have been allocated from the Docker engines host. Spikes in allocated resources can indicate an anomaly and forward to its resolution. Events in the Docker engine are another metric that can help with identifying possible problems within containers. Drops or spikes in the number of total events can be investigated with "engine_daemon_events_total."

Another set of metrics with potentially beneficial information can be collected from the processes running underneath the Docker containers. Similar to the resources allocated for docker containers, the virtual memory used by them can also be used to detect abnormalities. The amount of virtual memory used by overall containers is extracted with "process_virtual_memory_bytes" in Prometheus. This can reveal unwanted or faulty memory usage in containers. The last metric to be inspected from the study is the number of HTTP requests handled by the inspected system. The total amount of requests can be fetched with "http_requests_total." This can be used to determine unusual spikes and traffic inside the system.

Gathered metrics can be divided into two different categories, work metrics and resource metrics (Chakraborty & Kundan 2021, pp 31-34). The benefits gained from metrics can differ depending on the category of the metric. Work metrics can be used to evaluate the traffic in a service with throughput. Throughput indicates how much work is done in a time period by the monitored component. The number of errors occurring in a specific time period is also a work metric and it can be beneficial in detecting where errors are occurring. Another

metric falling under the work metrics category is latency. Latency can be analyzed to notice if there are fluctuations in performance and where they are occurring.

While work metrics provide information about the service workload and performance, resource metrics give a view of the resource allocation for the services. Resource metrics provide information about the utilization and availability of resources for services along with resource saturation. Information provided by the work metrics better reflects how the end users experience the service. Because of this work metrics tend to be more valuable for the improvement of the service. However, resource metrics can also give valuable background information for the work metrics.

To further emphasize which of these metrics should be given the most attention, Beyer et al. (2016, p. 60) define latency, traffic, errors and saturation as "The Four Golden Signals" of metrics. Riti (2018, p. 173) states that these four metrics work as a good base for an efficient monitoring system. Beyer et al. (2016, p. 60) stress that latency can also be useful for tracking the response time of failed requests in addition to tracking successful ones. Chakraborty and Kundan (2021, p. 37) even state that a thorough examination of the latency of a failed request is more important than successful requests. Traffic can be measured differently depending on your type of microservice with the number of HTTP requests being the most usual. The number of sessions, transactions or the rate of network input/output can also be measured as traffic. When it comes to tracking errors, it should be considered whether you are tracking solely requests with error status or do you also consider successful requests with faulty content. Depending on your choice, they may have different uses in aiding development.

Another way to provide developers with immediate information about the state of a microservice is with the use of the RED method (Chakraborty & Kundan 2021, pp 35-36). RED here stands for request rate, error rate and duration. Request rate refers to the number of incoming requests, such as HTTP requests, to the service in a period of time. Spikes in the number of requests can be investigated to better understand the reasons behind them. With the number of requests, it can then be examined how many of those requests have

failed. This refers to the error rate part of the RED method. Seeing a lot of errors of a certain status code can give clarity to the state of the service. Duration gives an answer to how long it takes for the service to provide a response. Long response times are a clear indication that something might be wrong within the service. The benefit of the RED method is that it reflects how end users experience the service (Yocum, 2021). It also helps with faster identification and solving of service-related problems. Since the RED method only focuses on requests and responses, it is the most beneficial when combined with other monitoring methods.

3.2.4 Tracing utilization

Tracing is a pivotal part of microservice monitoring when trying to get a better understanding of the behavior and performance of the microservice system. (Santana et al. 2019). Tracing provides a more coalesce overview of the route a request travels within the microservice system. This can help with understanding dependencies between services and with analyzing the latency of the request in various parts of its route. (Mace 2017) When combined, all this tracing information can be used to increase the microservice's performance and more efficiently resolve problems.

In a study conducted by Clément et al (2022) OpenTelemetry provided monitoring data was used to identify bottlenecks in distributed cloud systems. A key factor in discovering these bottlenecks was to use the trace data to model a graph that represents the communication pipelines in the distributed system. Two types of centrality rankings were used to identify possible bottlenecks. The first one was to count the number of communication connections that each service has. This would indicate how many services depended on a particular service and thus could be used to highlight a possible bottleneck. The second ranking method would count the number of communication paths that a specific service is connected to. This would highlight which services work as bridges between different groups of services and could in this way cause bottlenecks in the system.

The identification of bottlenecks gives insight if there are services that need increased reliability, or if the system needs to be redesigned to reduce the bottlenecks. The handling of bottlenecks is more in line with the management of the system infrastructure and thus helps the system architects and operation managers more than service developers. On the other hand, the identification of a bottleneck service could provide to be useful to developers. Additional focus could be given by the developers to services identified as bottlenecks. This could lead to improved quality of the bottleneck service and increase its reliability.

Service dependency analysis is a technique that can be used to provide an overview of all the dependencies in the microservice system (Li et al. 2022). It uses a flow graph to visualize the dependencies within the microservice. An example of this is shown in figure 3. This type of visualization can be achieved in Grafana by using the FlowCharting plugin (Grafana 2022).

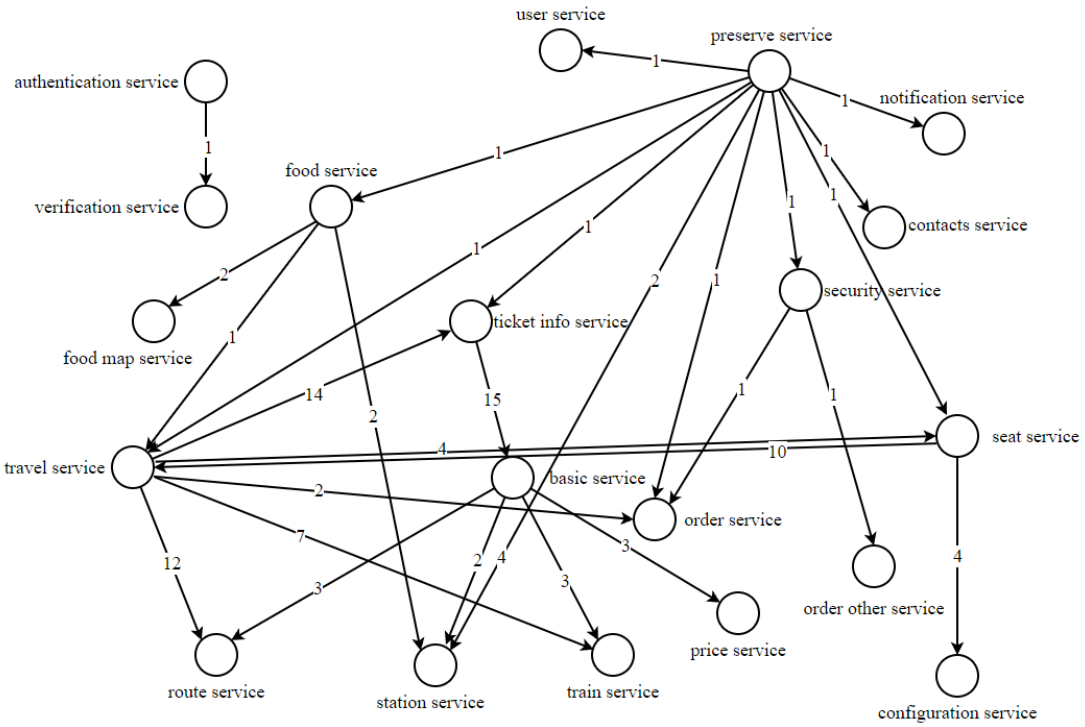

Figure 3. Service dependency analysis (Mimicking Li et al. 2022)

Service dependency analysis can be used to monitor the effects resulting from updates to a microservice. Keeping track of the microservices' dependencies can also be useful for managing the microservices and increasing their autonomy by reducing unnecessary dependencies. Dependencies can be divided into two different types to help identify criticality of a dependency (Liu et al. 2019). These types are strong dependencies and weak dependencies. If a failure occurs in an API call and it causes all subsequent calls to not invoke, the call is considered to have a strong dependency. In a weak dependency's case, the subsequent calls can still be called. Therefore, failures in strong dependency calls can have much more severe consequences, and thus should require more attention from developers.

3.2.5 Leveraging logs

Log information can be gathered to better understand the application and its performance as well as to collect data related to the system under the application. This makes logging a pivotal part of microservice monitoring. (Chakraborty & Kundan 2021, p. 90) Being able to gather logs from all the services in the distributed systems is an important aspect of microservice monitoring, especially when there are multiple teams working on different services in the same system (Hunter 2017, p. 131). Since errors in another service can lead to failures in your service and vice versa, being able to inspect the logs across the entire system helps at dealing with dependencies across the services.

One main use case for logs with microservices is their usefulness in conducting root cause analyses (Chakraborty & Kundan 2021, p. 90). Root cause analysis is a general tool for identifying what kind of event has occurred, how it has occurred, as well as why it has occurred (Rooney & Heuvel 2004). In the case of microservice monitoring, the root cause analysis can be used to detect anomalies and problems and find explanations for why they have occurred. Logs are not the only useful tool for helping with root cause analysis as traces and metrics can also be leveraged to aid with it (Li et al. 2022). This is why combining all the pillars of observability will give the highest benefit for conducting a root cause analysis.

When collecting log information from Kubernetes to Grafana, a key feature that can be utilized is the filtering of the logs (Chakraborty & Kundan 2021 pp. 139-140). The logs can be categorized with different levels of severity. Filtering the logs based on their severity helps with distinguishing whether the gathered logs can provide to be useful. Various other filters such as different identifiers can also be used to speed up the discovery of the required log information.

Log information about clients' requests to the microservice system can be written in a form of access logs (Chakraborty & Kundan 2021 pp. 137-139). Access logs can provide information like the total time used to process a request or the time taken from connection to receiving the whole response as well as the time taken to send the response. This information can be used to detect if slow response times or outages are present in the microservice system. By monitoring the requests over time, they can be separated into normal requests and anomalous requests to help with distinguishing poor request performance (Mi et al. 2012). Knowing which part of the request is causing the performance drop can help with pinpointing the cause behind the slow response time in the request.

3.3 Field study as a research method

Field study is a research method where the research is conducted in a specific real-world environment without the researcher having any active control over the research setting or variables (Stol & Fitzgerald 2018). The most straightforward tools for conducting research in the software engineering field are interviews and questionnaires. These are both techniques that require direct contact with the target population. The less involvement a field study technique requires, the more reliable the results are. On the contrary, more involvement leads to higher flexibility and cognition in the results. (Lethbridge, Sim & Singer 2005) The technique used in this work is a questionnaire. Since a questionnaire requires direct involvement with the target group, the results are assumed to be less reliable but more comprehensive.

Questionnaires are a collection of written questions that can be used for a cost and time-effective gathering of views on a certain subject from a large group of people (Shull et al. 2008, pp. 15-16). The questionnaire will be produced based on knowledge gathered from the literature review. The questionnaire is conducted in seven steps as described by Kasunic (2005, p. 6). The first step is to identify what the research objectives are. The second step is to identify the target audience of the survey. The third step is to decide how the questionnaire is going to be distributed. The fourth step includes designing and writing the questionnaire. The fifth step is to pilot test the questionnaire and the sixth is to distribute the questionnaire to the target audience. The final step is to analyze the results and write down the findings.

3.4 Field study on monitoring needs

To get a realistic understanding of monitoring needs in a concrete software development company, a field study is conducted. The field study is conducted as survey research. The survey research is created based on the previous literature review to further narrow down the most pivotal areas in microservice monitoring. The survey aims to distinguish what areas of microservice monitoring are seen as the most beneficial in aiding development in a concrete software development company. Another goal of the survey is to find support for the monitoring advantages recognized in the earlier literature review as well as find new potential ways to improve development with monitoring. Since field research is conducted inside one specific company the results may not reflect the monitoring needs present in other software companies. So, the results of this research should not be taken as an accurate representation of the whole software development field. Instead, the results work more as a directive view on monitoring needs in companies that develop and maintain microservices.

Target audience for the field study in this work is a Finnish SaaS company. The company is an SME (Small and Medium-sized Enterprise) and has recently adopted the development of multiple different microservices which makes them a good candidate for the field study. The Finnish SaaS company is chosen as a target audience for their request for this research. A web questionnaire is chosen as an easy and effective way to collect the Finnish SaaS

company’s software engineers' views without the need for physical contact or a matching timeslot in both parties' schedules. Google Forms is used as a platform for conducting the questionnaire because its use is already established at the target company.

Based on the findings in the literature review, the survey is divided into five main categories. The first three categories focus on the three pillars of observability identified in Chakraborty's and Kundan’s (2021, pp. 27-28) book and the first question of each category focuses on the effects of that pillar of observability. The first category is metrics, the second is traces and the third is logs. Errors were pointed out as a pivotal point for monitoring by multiple sources. Therefore, the fourth category focuses on different types of errors and their value in improving development. The last category includes questions that don’t specifically fall under any of the previous four categories, and they deal with the overall usefulness of microservice monitoring.

The created survey consists of 21 questions as shown in table 2. Questions two to four in the first category are based on the splitting of metrics into two categories, as done in the Chakraborty's and Kundan’s (2021, pp 31-34) book. Questions five and 17 focus on the alerting introduced by studies from Swersky (2018) and Ewaschuk (2013). Questions seven and eight are based on the findings from studies by Clément et al (2022) and Li et al. (2022). Chakraborty's and Kundan’s (2021, p. 90; 2021, pp. 139-140) and Hunter's (2017, p. 131) books act as a foundation for questions ten and 11. Questions 12 to 15 are also derived from Chakraborty’s and Kundan’s (2021) book as well as the book from Beyer et al. (2016, p. 60). Question 16 compares the impact of all three pillars of observability. Questions 18 and 19 combine the research done by Gribaudo, Iacono and Manini (2017), Chakraborty and Kundan (2021, p. 16) and Riti (2018). Question 20 is based on the RED method pointed out in works from Yocum (2021) and Chakraborty and Kundan (2021, pp. 35-36). The final question is used to collect overall thoughts on the benefits of microservice monitoring from the respondents.

Table 2: Survey questions

Number	Type	Question
Question category 1: Metrics		
Q1	Fixed scale	How likely would you need microservice METRICS to help with development?
Q2	Fixed scale	How likely would you need work related metrics to help with development?
Q3	Fixed scale	How likely would you need resource related metrics to help with development?
Q4	Free form	What type of metrics could you find useful (E.g., total events, memory usage)?
Q5.1	Fixed scale	How impactful could getting alerts about metrics be for faster fixing and improving of microservices?
Q5.2	Free form	What type of metrics should you receive alerts for, in your opinion?
Question category 2: Traces		
Q6	Fixed scale	How likely would you need microservice TRACES to help with development?
Q7.1	Fixed scale	Does the identification of bottlenecks in microservice systems have an effect on your service development process?
Q7.2	Free form	What type of effect would it have in your opinion?
Q8	Fixed scale	What is trace data's impact on finding and fixing problems in microservice functionality?
Question category 3: Logs		
Q9	Fixed scale	How likely would you need microservice LOGS to help with development?
Q10	Fixed scale	What is monitored log information's impact on finding and fixing microservice functionality?
Q11.1	Fixed scale	What kind of need is there to be able to filter the monitored log information?
Q11.2	Free form	What type of filters could you find useful (E.g., customer company identifier)?
Question category 4: Errors		
Q12	Fixed scale	How much would monitoring failed request help with development?
Q13	Fixed scale	How much would monitoring successful requests with faulty content help with development?
Q14	Fixed scale	How much would monitoring failed request latency help with development?
Q15	Free form	What do you see as the largest gain from monitoring errors?
Question category 5: Overall		
Q16	Pick one	Which part of microservice observability do you find most beneficial for improving microservice development?

Q17	Free form	What would you see as the best way to receive alerts about events captured with monitoring?
Q18.1	Fixed scale	What is the need to analyze request response times for improving the performance of microservice logic?
Q18.2	Free form	How could request response times help with development?
Q19	Fixed scale	How impactful could monitoring data be for faster improvement of services?
Q20.1	Fixed scale	How likely could monitoring data help with solving issues pointed out through customer service?
Q20.2	Free form	In what way could it prove to be helpful?
Q21	Free form	What do you feel would be the most important feature of microservice monitoring that would advance your microservice API development?

Some questions are answered in free form, but the vast majority have a fixed scale. The scale used to evaluate the need for specific monitoring data in the questions ranges from one to five. One in the scale represents that there is little to no value or impact from that aspect of monitoring in microservice development. Three on the scale represents that it is uncertain whether that aspect of monitored information has value or impact. And the highest value of five represents that there is a significant value or impact gained from that aspect of microservice monitoring. To get a more detailed view of the monitoring needs, some of the questions have follow-up questions that are also answered in free form.

The questionnaire is pilot tested first by one of the software engineers in the target audience. After successful pilot testing, the questionnaire is distributed to the target audience through email and the Finnish SaaS company's primary communication channel. The questionnaire is promoted on multiple occasions to ensure that enough responses are recorded to form sufficiently reliable results. Responses are recorded for a one-week period after which all the responses are analyzed.

4 Results

The survey was answered by 15 developers from nine different development teams. The vast majority of the developers had moderate experience in microservice development. Questions one, six and nine focused on the benefits of metrics, traces and logs, respectively. All these parts were seen as highly or significantly useful by over 70 percent of the respondents. Out of the three, traces were seen as most impactful with 85 percent of the respondents seeing it as highly beneficial for development. Out of the 85 percent, over half saw it as significantly beneficial as shown in figure 2. Metrics were seen as least impactful with only 20 percent seeing them as significantly impactful.

As shown in figure 4, all factors in the survey were mostly seen as highly beneficial by the developers. None of the factors were seen as completely useless. Monitoring of failed requests' latency was the only factor that over 20 percent of the respondents saw to have only a small impact on the development. Thus, it turned out to be the least beneficial factor to utilize. On the other hand, identification of bottlenecks and filtering of log information were seen as the most beneficial factors. 85 percent of the respondents saw the identification of bottlenecks with tracing as highly beneficial with the remaining 15 percent seeing it have moderate benefit. Being able to filter log information was seen useful by 80 percent of the respondents with the remaining seeing it as moderately useful.

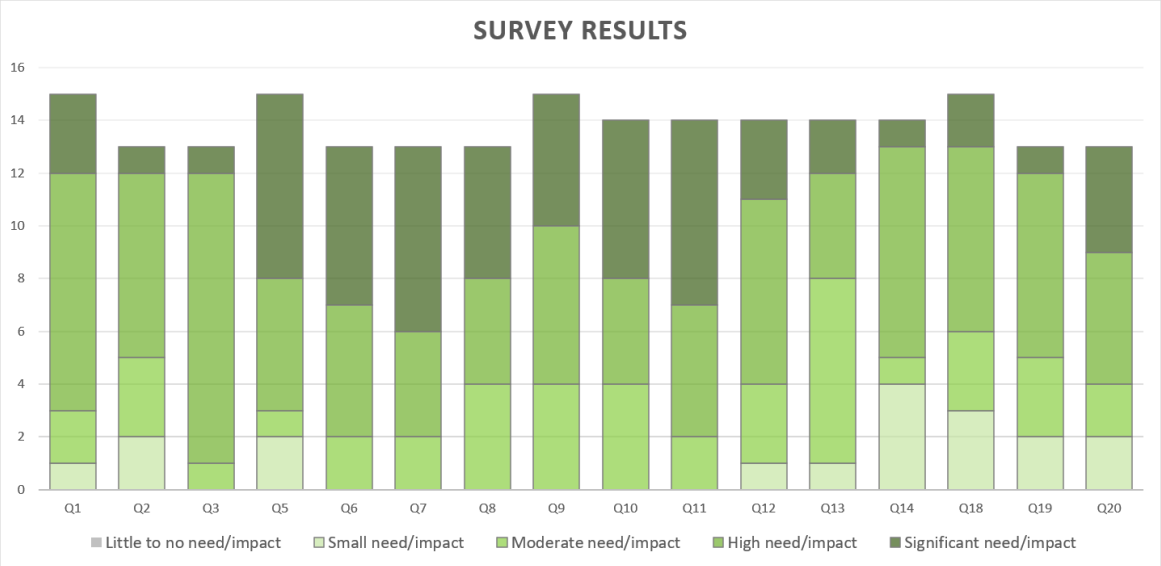

Figure 4. Survey results (Based on questions in table 2)

Figure 5 shows which parts of the microservice observability were seen as most beneficial or impactful for improving microservice development. Out of the 15 developers six saw logs as the most beneficial, and another seven saw all the parts to be equally beneficial. Only one developer considered traces as the most useful part, and none of the developers saw metrics as most beneficial.

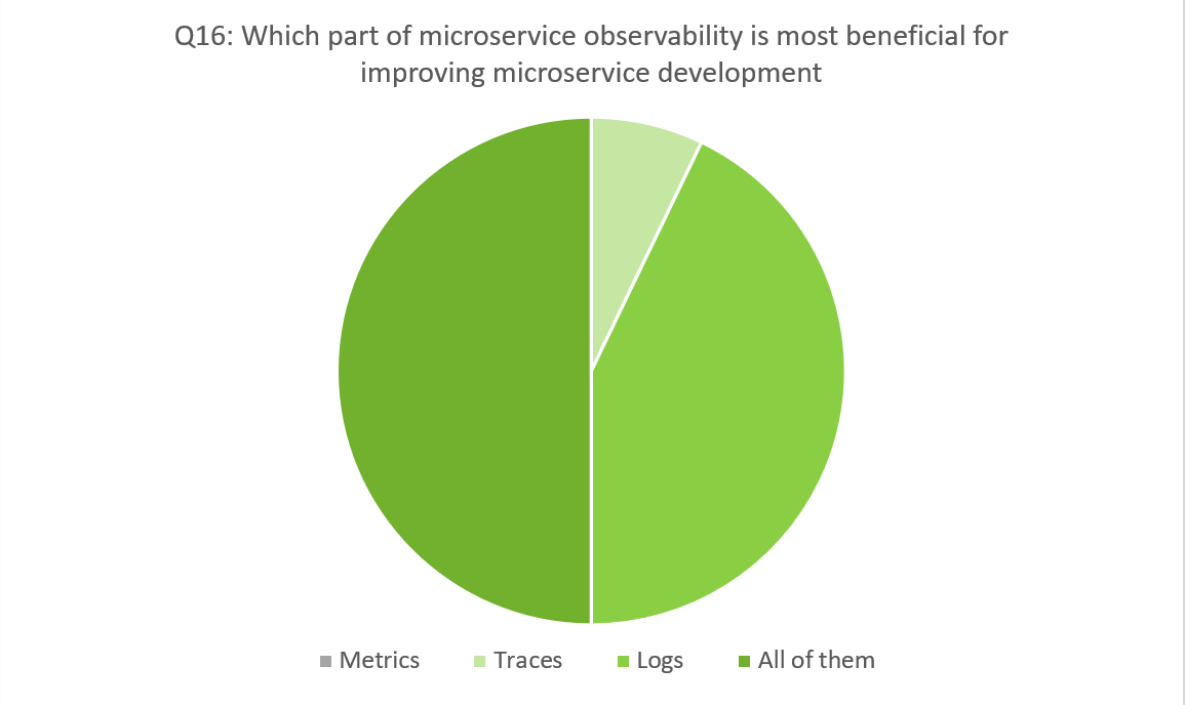

Figure 5. Question 16 results

Having the microservice monitoring system send alerts about various captured events was seen to benefit the speed at which developers can identify end-user facing problems and resolve them in the literature review. In the survey, 80 percent of the respondents saw the alerting to be highly beneficial for faster resolving of microservice problems. Based on both the literature review and the survey, the most common metrics to receive alerts from were availability, latency and errors. Receiving critical alerts through email or company messaging platforms like Google Slack saw the most support among the survey respondents. For less critical alerts, highlighting them in a monitoring dashboard was widely supported.

5 Discussion

Since the sample size of the survey is only 15 developers with all being from the same company, the results cannot be used to reflect the global monitoring needs for microservice development. However, they do give a comprehensive overview of what could be utilized for possible increase of customer value through continuous integration of the microservices.

5.1 Utilization in development

Out of the three pillars of observability, the vast majority of the responses in question 16 saw either logs being the most beneficial or all equally beneficial for improving microservice development. Based on this result we could argue that logs have the highest impact on improving microservice development and helping with resolving issues related to them, but none of the other parts should be left out. However, the results from the individual questions about the benefits of each aspect point otherwise. Based on the individual questions, tracing was seen as the most beneficial feature out of the three.

On the contrary to the literature review, resource related metrics were seen as more useful compared to the work related metrics. This could indicate that there is a larger need to ensure the availability of the services through fixing the resource utilization than there is to enhance the performance of the service. It could also mean that logs and traces are seen as a better tool for improving the performance of the services, which would make work metrics less valuable for that matter, whereas resource metrics would be more pivotal for ensuring availability. Availability ensuring was also recognized earlier as one of the main common needs for microservice monitoring which could further support this view of valuing resource metrics over work metrics. Nevertheless, both resource and work metrics were not seen as impactful for improving development as logs and traces.

Detection of bottlenecks was seen as a very useful tool for improving development. Similar to the findings in the literature review, the main use for identifying bottlenecks was to be able to detect possible performance problems. Identification of bottlenecks through tracing should be particularly useful to help avoid the bottlenecks by revamping the microservice functionality. This means that tracing helps with pinpointing the root problems for the bottlenecks, which leads to faster fixing of the performance problems or finding ways to go around them.

In the literature review, monitoring microservice response times was seen as useful tool for fixing and improving microservice performance issues. It was emphasized that monitoring failed requests latency could be even more pivotal than monitoring the latency of successful ones. However, in the survey monitoring failed requests latency was seen as the least useful part of microservice monitoring.

5.2 Value to customers through continuous integration

Monitoring data was mostly seen as a useful tool for faster resolving of issues related to end-users. Monitoring was seen as a beneficial tool for responding to these issues as well as helping to find and fix them even before they affect the end-users. One key feature that was highlighted in the survey responses was the ability to filter out end-user related logs from the monitoring data. Being able to filter out the logs with specific request paths was also seen as hugely beneficial for resolving end-user related issues.

The findings from the literature review and the survey questions focused on dealing with errors and issues that end-users would face. However, the survey responses pointed out another potential way of utilizing monitoring. According to the responses, monitoring could be used to learn how the end-users are using the microservices. This would be useful to analyze if the services are used as intended or if they need to be revoked.

The literature review pointed out the RED method as a useful method to leverage metrics in microservice development. The potential of the RED method was further backed by the survey where the three components of the RED method were pointed out as useful by multiple respondents. Since the RED method reflects how the customer or end-user experiences the microservice, it is especially beneficial for increasing the value to customers. Being able to make faster fixes to request and response related problems reduces the negative effect they would have on customers. Since the metrics used in the RED method are very straightforward and easy to monitor, their monitoring can be easily automated. This allows us to set up alerts for when there are spikes in these metrics and thus increase the developers' ability to respond even further.

In the literature review, alerting was generally found to have an impact on the speed at which required changes can be made to the microservices. Based on the responses to question 17 in the survey, the way of receiving alerts should differ based on the criticality of the alert. For alerts that have less impact on the microservices functionality and are not imminent to the user, it is best to highlight the alert in the monitoring dashboard. This way developers do not get flooded with alerts that do not require immediate attention.

6 Conclusion

Monitoring and combining the three pillars of observability, which are metrics, traces and logs, is what allows developers to achieve microservice end-to-end transparency. Combining these aspects of monitoring allows developers to utilize different analytical methods to improve their microservice development. Focusing on monitoring end-user related issues and anomalies enables developers to include faster fixes to these issues in their continuous integration and delivery pipelines.

We looked at several ways and methods of utilizing microservice monitoring to aid microservice development. The three pillars of observability should allow companies and development teams to get a lot of valuable information about their services' health and state. The monitored information can then be used to conduct root cause analyses that improves the speed at which problems can be detected and resolved within the microservice. The information can also be used to conduct service dependency analyses and detect possible bottlenecks in the microservice systems to help enhance the services' performance. Setting up alerts for issues that affect end users overall improves the developers' ability to respond to these issues. Thus, it diminishes the loss of value the end users could face from these issues. Keeping the alerting simple and using different ways of alerting depending on the criticality of the issues should provide a clear indication to developers of how and when to act to these issues. Implementing all these practices should lead to better continuous integration and delivery of the microservices.

The survey results supported many things found in the literature review. The RED method introduced in Chakraborty's and Kundan's (2021, pp 35-36) book as well as in Yocum's (2021) article saw a lot of support in the survey responses. Using microservice tracing for conducting service dependency analyses to find bottlenecks in the microservice system was found pivotal for improving development by Li et al (2022) and Clément et al (2022). The survey respondents also saw this as a significant benefit for improving the services. The use of logs for resolving customer or end-user related problems proved to have significant

benefits by both the literature review and the survey responses. The survey responses also pointed out the potential of identifying customer behavior through logs. This allows teams to see how the services are being used and if it matches their intent or if the service needs refactoring.

This research touched briefly on the use of dashboards like Grafana to visualize the monitoring data in various ways. To further investigate how microservice monitoring can be utilized in the best possible way, monitoring visualization should be further researched. The efficiency at which the monitoring data can be analyzed is linked to the way it can be presented. Therefore, finding out what are the best graphs for visualizing different pieces of information could further increase the potential of microservice monitoring. Thus, knowing how to build dashboards to display and combine the monitoring data in the most efficient way should be the next logical step toward mastering microservice monitoring for the betterment of service development.

References

Beyer, B. et al., 2016. ‘Monitoring Distributed Systems’ in Site reliability engineering: How Google runs production systems. O'Reilly Media, Inc., pp 55-66.

Brondolin, R. & Santambrogio, M. D., 2020. A Black-box Monitoring Approach to Measure Microservices Runtime Performance. ACM transactions on architecture and code optimization. 17 (4), pp. 1-26.

Branley, D., Seale, C., & Zacharias, T., 2004. Doing a literature review. Researching society and culture, pp. 145-162.

Bucchiarone, A. et al., 2020. Microservices Science and Engineering. 1st ed. 2020. Cham: Springer International Publishing.

Chakraborty, M. & Kundan, A. P., 2021. Monitoring cloud-native applications: lead agile operations confidently using open source software. 1st ed. 2021. Apress.

Chen, L., 2018. Microservices: architecting for continuous delivery and DevOps. In 2018 IEEE International conference on software architecture (ICSA) IEEE.

Clément, C. et al., 2022. A Tracing Based Model to Identify Bottlenecks in Physically Distributed Applications. In 2022 International Conference on Information Networking (ICOIN), Jeju-si, South Korea. pp. 226-231.

Cronin, P., Ryan, F., & Coughlan, M., 2008. Undertaking a literature review: a step-by-step approach. British journal of nursing, 17(1), pp. 38-43.

Eski, S. & Buzluca, F., 2018. An automatic extraction approach: transition to microservices architecture from monolithic application. In Proceedings of the 19th International Conference on agile software development. 2018 ACM. pp. 1-6.

Ewaschuk, R., 2013. My Philosophy on Alerting. [online]. Available at: https://docs.google.com/document/d/199PqyG3UsyXlwieHaqbGiWVa8eMWi8zzAn0YfcApr8Q/edit#heading=h.fs3knmjt7fjy [Accessed 8 April 2022].

Grafana, 2022. Documentation. [online]. Available at: https://grafana.com/docs/ [Accessed 5 April 2022].

Gribaudo, M., Iacono, M., & Manini, D., 2017. Performance evaluation of massively distributed microservices based applications. In 31st European Conference on Modelling and Simulation. 2017 European Council for Modelling and Simulation (ECMS), pp. 598-604.

Hunter II, T., 2017. Advanced Microservices A Hands-on Approach to Microservice Infrastructure and Tooling. Berkeley, CA: Apress.

Jamshidi, P, et al., 2018. Microservices: The journey so far and challenges ahead. IEEE Software, 35(3), pp. 24-35.

Kasunic, M., 2005. Designing an effective survey. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

Knopf, J. W., 2006. Doing a literature review. PS: Political Science & Politics, 39(1), pp. 127-132.

Kubernetes, 2021a. What is Kubernetes? [online]. Available at: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Accessed 23 February 2022].

Kubernetes, 2021b. Pods. [online]. Available at: https://kubernetes.io/docs/concepts/workloads/pods/ [Accessed 23 February 2022].

Larrucea, X. et al., 2018. Microservices. IEEE software. 35 (3), pp. 96-100.

Lethbridge, T. C., Sim, S. E., & Singer, J., 2005. Studying software engineers: Data collection techniques for software field studies. Empirical software engineering, 10(3), pp. 311-341.

Li, B, et al., 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering, 27(1), pp. 1-28.

Liu, H. et al., 2019. JCallGraph: Tracing microservices in very large scale container cloud platforms. In International Conference on Cloud Computing. Springer, Cham. pp. 287-302.

Mace, J., 2017. End-to-end tracing: Adoption and use cases. [survey]. Brown University.

Mi, H. et al., 2012. Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs. Science China. Information sciences. 55(12), pp. 2757-2773.

OpenTelemetry, 2021. Collector documentation. [online]. Available at: https://opentelemetry.io/docs/collector/ [Accessed 23 February 2022].

Pahl, C. et al., 2019. Cloud Container Technologies: A State-of-the-Art Review. IEEE transactions on cloud computing. 7(3), pp. 677-692.

Pina, F. et al., 2018. Nonintrusive monitoring of microservice-based systems. In IEEE 17th International Symposium on Network Computing and Applications (NCA). pp. 1-8. IEEE.

Prometheus, 2022. Prometheus overview. [online]. Available at: https://prometheus.io/docs/introduction/overview/ [Accessed 22 February 2022].

Rajasekharaiah, C., 2021. Cloud-based microservices. 1st ed. Suwanee, GA: Apress.

Riti, P., 2018. Pro DevOps with Google Cloud Platform. With Docker, Jenkins, and Kubernetes. Berkeley, CA: Apress.

Rooney, J. J., & Heuvel, L. N. V., 2004. Root cause analysis for beginners. Quality progress, 37(7), pp. 45-56.

Sabharwal, N. & Pandey, P., 2020. Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager. 1st ed. 2020. Berkeley, CA: Apress L. P.

Santana, M. et al., 2019. Transparent tracing of microservice-based applications. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. pp. 1252-1259.

Sayfan, G., 2017. Mastering kubernetes. Packt Publishing Ltd.

Shull, F. et al., 2008. Guide to Advanced Empirical Software Engineering. London: Springer-Verlag London Limited.

Stol, K.-J. & Fitzgerald, B., 2018. The ABC of Software Engineering Research. ACM transactions on software engineering and methodology. 27 (3), pp. 1-51.

Swersky, D., 2018. The Hows, Whys and Whats of Monitoring Microservices. The New Stack. [online]. Available at: https://thenewstack.io/the-hows-whys-and-whats-of-monitoring-microservices/ [Accessed 8 April 2022].

Watts, T. et al., 2019. Insight from a docker container introspection. In Hawaii International Conference on System Sciences 2019.

Yocum, T., 2021. The RED method: A new strategy for monitoring microservices. InfoWorld, [online]. Available at: https://www.infoworld.com/article/3638693/the-red-method-a-new-strategy-for-monitoring-microservices.html [Accessed 10 April 2022].