TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Annabel Pugi 222791IAIB

Development of a Sensitive Data Logging Service

Bachelor's Thesis

Supervisor: Tarvo Treier
MSc

Co-Supervisor: Kim Naciscionis
MA

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Annabel Pugi 222791IAIB

Sensitiivsete andmete logimise teenuse arendamine

Bakalaureusetöö

Juhendaja: Tarvo Treier

MSc

Kaasjuhendaja: Kim Naciscionis

MA

Tallinn 2025

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis and that this thesis has not been presented for examination or submitted for defense anywhere else. All used materials, references to the literature, and work of others have been cited.

Author: Annabel Pugi

04.06.2025

Abstract

The goal of this bachelor's thesis is the development of a logging service for sensitive data. The purpose of logging is to provide information about system behavior. This information can be used for debugging. However, when the data being logged is sensitive, it is additionally important to ensure the security of customer data.

As part of this work, a logging system was developed for email sending feature in company X. The system consisted of Kafka, a data-filtering service, and two databases for storing the data. One database was used for storing metadata and the other for sensitive data, which had a separate access policy applied. The metadata database was added to Grafana as a data source to enable data visualization, searching, and analysis.

The resulting system is a potential example of a logging system for sensitive data. According to feedback from developers and support engineers, the system met the requirements set by company X. It could be used for solving customer-related issues and it did not affect other services.

The thesis is in English and contains 34 pages of text, 6 chapters, 11 figures, 4 tables.

Annotatsioon
Sensitiivsete andmete logimise teenuse arendamine

Käesoleva bakalaureusetöö eesmärgiks on sensitiivsete andmete logimise teenuse arendamine. Logimise eesmärk on pakkuda teavet süsteemi käitumise kohta. Seda teavet saab rakendada erinevate probleemide lahendamisel. Kui logitavad andmed on sensitiivsed tuleb nende logimisel tagada ka kliendi andmete turvalisus.

Töö käigus valmis logimissüsteem emailide saatmise info kogumiseks firmas x. Süsteem koosnes Kafkast, andmeid filtreerivast teenusest ning kahest andmebaasist, kuhu andmed salvestati. Üks andmebaas oli metainfo talletamiseks ja teine sensitiivsete andmete jaoks, millele rakendati eraldi ligipääsupiirang. Metainfot sisaldav andmebaas lisati andmeallikana Grafanasse, et oleks võimalik andmeid visualiseerida, otsida ja analüüsida. Tulemusena valminud süsteem on üks potensiaalne näide sensitiivsete andmete logimise süsteemist. Nagu selgus arendajate ja tugiinseneride tagasisidest täitis antud süsteem firma x poolt esitatud nõuded süsteemile. Süsteemi oli võimalik kasutada kliendi probleemide lahendamisel ning see ei mõjutanud teisi teenuseid.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 34 leheküljel, 6 peatükki, 11 joonist, 4 tabelit.

List of abbreviations and terms

CPU	Central Processing Unit
DB	Database
SE	Support Engineer

Table of contents

List of figures

List of tables

1 Introduction

Logging is essential in software development as it enables system health assessment, problem detection, and security risk identification. Additionally, logging helps to collect valuable information that helps to respond quickly to problems and identify root causes more easily [1]. On the other hand, companies must ensure that only authorized individuals have access to customer data. According to the European Union General Data Protection Regulation (GDPR), organizations must guarantee the security of customer data and limit access to it. Therefore, developing and implementing systems that manage the logging of sensitive data is especially important. [1] [2]

The aim of this thesis is to develop a sensitive data logging system for email sending feature in company X. Such a logging system is important for preventing or reducing incident resolution time by providing information for debugging as well as helping SEs with solving customer problems. The system must be able to separate sensitive data from nonsensitive data. Additionally, the system should provide data visualization and follow guidelines set by GDPR.

The author came in contact with the problem while working as a software developer for the company X. The development for the solution started in June 2024 and finsihed in November.

1.1 Goals

The goal of this thesis is to develop a sensitive data logging system that can be used for:

- Identifying and resolving issues and errors.
- Analysing stored data to determine the need for optimizing traffic, speed, and overall service performance.
- Visualizing data for a better overview.

System requirements are:

- It should be as reliable as possible, with a simple architecture and low maintenance costs.
- It must not affect other services.
- Access to sensitive data must be restricted.

For the employer, it is important that the system allows logging and verification of what an email looked like at the moment it was sent, making it easier to solve user issues and reducing the time spent by customer support. Additionally, the system should provide access to statistical data on email sending via Grafana.

1.2 Structure of the work

This work consists of six chapters. The first chapter introduces the work, the second chapter gives an overview of the background, and the third chapter gives an overview of the planning, requirements, and alternative solutions. The fourth chapter describes the implementation process, and the fifth chapter analyses whether the requirements were fulfilled, what could have been done better, and what can be done next. The sixth chapter includes the summary of the work.

2 Background

This chapter provides context for understanding the problem domain of logging sensitive data in distributed systems. It begins with a general overview of logging: what logs are, why they matter, and what challenges they present, especially in systems dealing with sensitive data. It also introduces relevant legal and regulatory frameworks like GDPR and how they impact logging practices. Additionally, it covers key concepts related to the thesis and provides context about the company.

2.1 General Problem Overview

This section gives an overview of the general problem being solved, which is setting up a logging system for sensitive data. For that, it is important to first understand what logging is, why it is necessary, and what the common problems related to it are.

2.1.1 What is logging in software systems?

Logging is a technique for collecting information about system behavior [3]. Log message (also called a log entry) is something generated by the system or device to note that something has happened. Typically a log message consists of timestamp, source and data. Timestamp shows when the message was generated, source shows the system that generated the message and data is the content of the log. There is no standard for the data format, it depends on how the source of the log has implemented its logging system. [4]

2.1.2 Why is logging important in distributed systems?

Logging provides information about how systems are behaving. There are multiple reasons why logging is considered to be important. First, for debugging: logs can be used to understand or confirm the causes of problems. Second, logs support the optimization and debugging of system performance by helping to analyze how resources are utilized over time by specific components. Third, for security: logs can aid in identifying attacks,

breaches, and application misuse based on user behavior. Finally, logs can be applied in predictive analysis. This means that it is possible to identify potential customers, plan and manage workloads, and schedule resources more effectively based on log data. [5]

2.1.3 Challenges of Logging and Logging Sensitive Data

Logging also comes with challenges. First, (as mentioned earlier) it can be hard to predict the exact format of the data that will be logged as it can be different based on applications, devices and in this specific context providers. Second, the logs can be difficult to access or find. People interested in the information in logs might struggle with finding the appropriate details. If the volume of logs and number of services is large, expert knowledge may be required to locate the correct information among all the data. Additionally, if the necessary logs are generated by multiple services or components, analyzing them becomes more difficult. Third, logging can cause performance overhead and, if implemented ineffectively, may slow down the system.[5] [6] There are additional challenges that arise when logging sensitive information. Even when logging, it is still important to ensure that the users' privacy is protected. It is important to understand that protecting users' privacy is critical for several reasons. It is necessary not only to comply with regulatory requirements and avoid potential fines, but also to maintain customer trust and organisation reputation, and prevent potential harm caused by data leaks. Such harm may include identity theft and financial losses. To protect user privacy, one must first understand what data needs to be protected. This leads to the second challenge which is that it may be difficult to identify which data is considered sensitive. Despite the large amount of works on general data privacy, there is no clear consensus on which log attributes should be treated as sensitive. Users may also be identifiable through seemingly harmless data, such as the type of device they use. Moreover, there is insufficient information on what log attributes might in combination lead to user re-identification. Therefore, context becomes crucial. Whether a certain parameter is sensitive or not depends on the context. For example, if a known group of users all answer a quiz about their device type and only one of them uses a Mac, that user becomes re-identifiable even without their name being revealed. [7] [8]

On the other hand, anonymizing logs can also reduce their usefulness. According to a study by, when comparing the accuracy of two machine learning models on a dataset before and after anonymization, there is a decrease of 1–4% in accuracy after anonymization. [9]

2.2 Legal and Security Context

2.2.1 What is Sensitive data?

In the convention for the Protection of Individuals with regard to Automated Processing of Personal Data "personal data" is defined as any information relating to an identified or identifiable individual. Individual becomes identifiable if they can be directly or indirectly be identified without spending unreasonable amount time, effort or resources. The nature of personal data is considered to be ever-changing with the fast development of technology. “Identifiable” means not just knowing someone’s legal or official identity, but also anything that can make a person stand out from others and thus be treated differently. Even if a fake name or digital ID is used, but the person can still be identified, the data isn’t truly anonymous and therefore is covered by the provisions of the Convention for the Protection of Individuals with regard to Automated Processing of Personal Data. [2] [10]

2.2.2 GDPR and Logging

The European Union General Data Protection Regulation (GDPR) is a data privacy regulation established by the European Union in 2018 that applies to the European Economic Area. The GDPR sets clear rules for how personal data should be collected, used, stored, and shared to protect individuals’ privacy in the EU and EEA. Its definition of personal data includes both direct information, like names, and indirect details. If an indirect detail can be used to identify someone, it is considered to be personal data. The regulation also recognizes that somebody can use data from devices and apps like cookie IDs to create detailed profiles, which might reveal someone’s identity when combined with other information.[2]

GDPR sets multiple principles for the processing of personal data, which can be summarized as seven key points. First, the data should be processed lawfully, fairly, and in a transparent manner with the data subject. Next, personal data should only be collected for legitimate and specified purposes and only processed in line with these purposes. Third, the data collected should be relevant and limited to what is necessary for the processing purposes. Fourth, organizations must ensure that personal data is correct, and if the data is wrong, people have the right to ask for it to be corrected. Fifth, organizations need to consider what data they store and for how long. Data should not be stored longer than needed.

Sixth, organizations must implement security measures to protect sensitive data that has been collected and make sure that access to it is limited. Last, organizations need to take responsibility for what they do with the collected data and how they comply with other principles. [2]

2.3 Technical Concepts and Terminology

This section provides context for technical concepts used in this thesis like event-driven architecture, Kafka and log management.

2.3.1 Event-driven architecture

In an event-driven architecture, services respond to events. An event can either happen inside or outside of your business and trigger one or more services to act, or it can be emitted by a service to inform other systems or users. These events may indicate problems, opportunities, or changes, and can lead to automated actions, business processes, or further communication across the system. [11]

2.3.2 Kafka

Apache Kafka is an open-source distributed event streaming platform consisting of servers and clients. Some of these servers (brokers) form the storage layer, and other servers are responsible for importing and exporting data as event streams. Clients are used by distributed applications and microservices to read, write, and process streams of events in parallel.[12]

Topics in Kafka are like folders in a filesystem. They contain events. Producers create events into topics, and consumers subscribe to topics and read events from there. Topics are divided into partitions, and when an event is published, it is added to one of the topic's partitions. When two events have the same event key, they are written to the same partition. This enables client applications to simultaneously read from and write to multiple brokers.[12]

2.3.3 Log management

Log management is a practice where the generated log messages are collected and used for data analysis to provide insights of runtime behavior. [13]

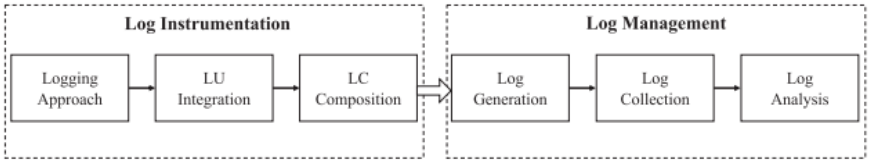

Figure 1. The overall process of software logging. [13]

As shown in Figure 1, software logging has two main parts: log instrumentation and log management. The log instrumentation phase happens during development and is about adding and managing the code that creates log messages. It includes: Logging Approach, Logging Utility Integration, and Logging Code Composition. In the logging approach step, one chooses how to log. In the logging utility integration step, one adds the logging tools into the system, and in logging code composition, writes the actual logging to the code. Log management is a practice where the generated log messages are collected and used for data analysis to provide insights into runtime behavior. It also consists of three steps: Log Generation, Log Collection, and Log Analysis.[13]

This thesis will focus on log instrumentation to enable effective log management later on.

2.4 About the company

The author works as a software developer at a cloud-based Software-as-a-Service company that offers a Customer Relationship Management (CRM) platform to its clients. Specifically, the author is part of a team within the company that focuses on the platform's email management product.

2.4.1 More about the problem in the company

There was an incident in the company related to email sending. The emails were sent out with partially missing content and solving the problem took a long time because it was hard to find the root cause of the problem. This because there are multiple services that handle email sending both from front-end as well as from back-end and it was hard to pinpoint

which services might have caused the issue. The issue was also not easily reproduceable meaning the team had to wait for customer to come in with a case to get more information about the bug and to test proposed solutions. The periods between new cases could be quite long which would lead to logs for previous cases being deleted making it hard to find patterns between the cases.

The email body could also be influenced by the provider, as the user can see the sent email after it has been synced in from the provider. Due to that, there was a wish to see the email content before it was sent to the provider. Existing logging systems (Loki) were not a fitting solution because email contains sensitive data, and access to it should be limited. Second, the retention period of logs was not enough, as the incident was not happening constantly, and the cases were often older than a month.

The second part of the problem involves support engineers and an SE-helper (a rotating role for one of the developers on the team) who try to solve different problems clients reach out with. They also face the problem that the issue happened more than a month ago, and it is hard to debug it as there is not enough information since the company's log retention is a month.

3 Analysis

This chapter gives an overview of the overall planning process, identifies key stakeholders and their roles, analyzes the functional and technical requirements, compares alternative solutions, and presents the chosen approach for implementing a secure logging system.

3.1 Planning Process

Before developing the solution, a planning phase was conducted to clarify the scope and direction of the project. The planning process was inspired by the steps described in the [13], along with the goals recommended in the [4]. However, it is also important to take into consideration that the survey and the book focused more on building general logging systems which aim to cover the entire codebase with enough logs and manage all these different logs. In contrast, the logging system developed in this thesis is meant for logging a very specific type of information: data related to sent emails. It is not intended to replace existing logging in services, but to provide extra logging with longer data retention and to gather related information in one place.

As explained in Chapter 2.3.3, the logging instrumentation phase consists of three main steps, each with several substeps. This chapter will focus on the first two steps and address two key questions from the third. Based on this, the steps covered in this chapter are:

- choosing logging approach
- selecting a logging utility
- deciding on configuration for the selected logging utility
- deciding where to log
- choosing what to log

The question of how to log will be addressed in the following chapter. [13]

In [4] the suggested stages were:

- pre-planning - understanding why you need the logging system and what problems it is solving
- stakeholder identification and access mapping – to determine who needs to use or contribute to the system.
- requirement gathering – based on stakeholder needs and system goals.
- evaluation of tools – to assess what tools are available and offer a good solution.
- selecting tool and Architecture planning – sketching out the data flow, components, and access controls.

The last goal is partially addressed in the next chapter. Several goals from [4], however, are not included in this analysis, as they are either too specific to general-purpose logging systems or pertain to building a system entirely from scratch.

By combining the two sets of goals, we arrive at the following planning steps:

- pre-planning
- choosing a logging approach
- identifying stakeholders
- gathering and analyzing requirements
- evaluating available tools
- selecting a tool and planning the architecture
  - deciding how to configure the chosen Logging Utility
  - choosing where to log
  - choosing what to log

While the project followed most of these steps, the early planning stage could have been given more attention. Some decisions, like who should be responsible for granting access to the data and service naming, were only worked out during the implementation. These could have been handled more smoothly with better planning at the start.

3.2 Pre-planning and choosing logging approach

It is important to start with preplanning in order to clearly understand the problems the log management system is intended to solve. This step was already covered in the previous chapter. In summary, the main goal is to address the issues identified in section 2.4.1. [4]

The selection of the logging approach in this case is relatively straightforward. There are three different logging approaches: Conventional Logging, Rule-based Logging, and Distributed Tracing. Given that the logging system is meant for logging at one specific time in response to a specific event - email being sent, Conventional Logging is the most appropriate choice. Rule-based logging works best in modular logging, not for instrumentation at a specific code location. Distributed tracing is mainly useful for correlating actions across multiple services or machines, and is not necessary for a single logging point in one service. Since neither of these conditions applies here, Conventional Logging presents the most practical and efficient solution.[13]

3.3 Stakeholders And Access Mapping

To better understand who need to be involved in the proccess of creating a solution, it is important to list the teams who will be involved in this process and how.

stakeholder	responsibility
Email team	services related to email management and sending, email product
Email team SEs	solving client problems related to email management and sending
privacy	ensuring clients' rights are met as per data protection regulations
info security	system design is in line with regulations
infrastructure	manages DBs and Kafka structures

Table 1. Stakeholders and Responsibilities

Table 1 shows that five different teams are involved in this project. First, the email team is responsible for developing and maintaining services related to the email management product. Developers from this team will also create the new logging service and act as one of its main users to gain insights into sending speeds. The second team, the Support Engineers, who handle client issues, will be the main users of the logging system. Next, the privacy and information security team plays a key role in ensuring that client data is protected and that all logging practices comply with regulations. Finally, the infrastructure team will assist in setting up the necessary components, such as new databases and Kafka topics.

3.4 Requirements Analysis

This section analyses the system requirements for the new logging solution. The requirements are defined based on use cases that display the needs of different client groups. Also data sensitivity, stakeholder priorities, and technical limitations are taken into consideration.

3.4.1 Goal Description

To better understand what are your system requirements, one should start with defining what are the purposes different groups of clients will use the system. As there are two groups of clients for the new logging system, the these purposes or goals can be made based on clients and problems listed in the previous chapter. [4]

Purpose 1: Debugging and Customer Support
Objective: Help support engineers identify the root cause of email issues (e.g., incorrect email title).
Data Needed:

- Metadata: header IDs, company IDs, user IDs, thread IDs, message IDs
- Delivery status and reason for failure
- Email type (regular, group, automation)
- Email html, recipients and subjects
- attachment information

Key Requirements:

- Ability to check individual email content at the point of sending
- Access to metadata months after sending that could be useful when debugging

Who needs this: SEs, SE-helper

Purpose 2: Incident Resolution
Objective: Help solve incidents related to email sending (e.g understanding if problem is coming from platform or provider side)
Data Needed:

- Metadata: header IDs, company IDs, user IDs, thread IDs, message IDs

- Delivery status and reason for failure
- Email type (regular, group, automation)
- Email html, recipients and subjects
- attachment information
- provider, host

Key Requirements:

- Ability to check individual email content at the point of sending
- Ability to visualize data and analyse it based on hosts and providers

Who needs this: email team developers

Purpose 3: Performance Monitoring and Analytics

Objective: Help analysing and understanding sending performance to determine the need for optimizing traffic, speed, and overall service performance.

Data Needed:

- Metadata: header IDs, company IDs, user IDs, thread IDs, message IDs
- Delivery status and reason for failure
- Email type (regular, group, automation)
- attachment sizes
- provider, host, domain
- sending speeds (provider/platform)

Key Requirements:

- Ability to visualize and analyse data based on sending performance (success, speed) and filter it based on parameters like host, domain, mailboxid
- Have an overview of highest and avarage attachment sizes to see if it influences sending

Who needs this: email team developers

Based on the use cases, one reason mentioned multiple times is being able to see the email as it was sent out from the platform. That means that data related to the email contents

should be saved. That includes actual email content, subjects, and recipients. Attachment contents can be excluded. However, attachment names and sizes are listed in the use cases, so these should be included. Another requirement mentioned in the use cases, is the ability to analyse and visualize metadata.

3.4.2 Stakeholder needs

- **Support Engineers:** Require access to sensitive data as well as metadata for detailed troubleshooting.
- **Email Team Developers:** Require possibility to filter data for performance analysis and visualization.
- **Infrastructure Team:** Responsible for managing Kafka topics and databases, from their side, it is important that their best practises are followed from db and kafka level
- **Security and Privacy Teams:** Require that data protection is ensured through controlled access and compliance with regulations. It is important to mention that we did not start collecting any new data with this solution.

3.4.3 Requirements based on data sensitivity

Next, how can your product be in compliance with security and keep clients data safe. For that the the access to sensitive data should be limited to certain people. Previously it was explained in 2.2.1 and 2.1.3 what is considered to be sensitive data. Now we can apply this knowledge to the data we will be logging to understand if this classifies as sensitive.

In the context of the proposed logging system, data sensitivity is assessed based on the potential to identify individuals or expose information that could harm clients if misused. As shown in Table 2, fields such as email HTML content, subjects, recipients, and attachment names are considered sensitive. These elements may contain personally identifiable information, confidential communication, or business-critical content, and thus require strict access control. On the other hand, identifiers such as user ID, company ID, thread ID, or message ID are classified as non-sensitive because these IDs alone do not reveal any specific user identity without using additional internal tooling to which access is limited. Similarly, metadata fields including provider, domain, host, email sending type, delivery status, failure reasons, attachment sizes, and sending speed are considered

field	sensitivity level
Email html content and subject	sensitive
in reply message IDs	non-sensitive
recipients	sensitive
provider	non-sensitive
domain and host	non-sensitive
attachment names	sensitive
attachment sizes	non-sensitive
IDs like user ID and company ID	non-sensitive
delivery status	non-sensitive
failure reason	non-sensitive
email sending type	non-sensitive
sending speed	non-sensitive

Table 2. Sensitive and non-sensitive data

non-sensitive, as they do not directly expose private information. Nevertheless, context remains critical: while individual metadata fields may not be sensitive, their combination or correlation with other datasets could increase privacy risks. Therefore, the system must be designed so that only authorized employers can access sensitive information.

Based on this analysis, it is possible to conclude that the data is partially sensitive. Additionally, taking into consideration stakeholder requirements, the non-sensitive data should be easily accessable for visualisation. Therefore, it makes sense to separate sensitive and non-sensitive data, allowing controlled access while supporting both usability and data protection requirements.

3.4.4 Technical requirements

Technical system requirements are:

- It should be as reliable as possible, with a simple architecture and low maintenance costs. As the service is not high-priority (not directly affecting customers), it should have low maintanence costs.
- It must not affect other services. As mentioned in chapter 2.1.3, one of the problems

associated with logging is that it causes overhead and can potentially slow down services. Because of that, the new logging service should influence existing sending flow as little as possible.

3.5 Related works

This section gives an overview of some of the suggested solutions for (sensitive) data logging systems in other works.

One of the suggested solutions is using blockchain. For example, Li et al. introduced a blockchain-based tool called TripleP for log management. However, another source also highlighted that with blockchain comes the challenge of finding a balance between minimizing blockchain redundancy to improve latency and maintaining a sufficient number of secure nodes to retain full control over the contents of the blockchain. [7] [14]

The other solution is using immutable databases. Multiple early database systems offered an immutable database option. However, immutable databases also introduced challenges, such as the inability to use key fields, which made retrieval and analysis of the stored data difficult and slow, which is quite important in this case. [14]

3.6 Evaluation Of Existing Systems

There are over 3000 logging utilities being used for logging. This section will give an overview of some of the most known logging tools that could be potentially used for setting up a sensitive data logging system for email sending. [15]

Starting with Loki and Grafana, which are already used by the employer. Loki is a log aggregation system optimized for cost-effective storage. It is designed to work well with Grafana for visualization and stores logs in a compressed, index-free format. However, this solution is not suitable for solving this problem because the data retention periods are not sufficiently long, and extending them is not desirable due to security concerns regarding customer data. Additionally, restricting developer access to the data while still providing an overview of other logs is more complex with Loki. Also the data that needs to be logged is quite big and could slow down Loki and make it difficult to manage other logs. Grafana is an open-source data visualization and monitoring platform, that lets you create custom

dashboards and supports alerting. It would be a fitting solution for visualising the metadata as it allows the creation of different dashboards and using variable based filtering on them. [16] [17]

Another alternative solution is ELK stack consisting of Elasticsearch, Logstash and Kibana. According to [18] Loki and Grafana are more resource-efficient in terms of CPU and memory. Additionally, since Loki and Grafana are already implemented by the employer and offer lower operational costs, introducing an additional third-party logging stack would not be practical or cost-effective.

Next alternative solution is Graylog. Graylog is a centralized logging solution that includes log collection, storage, analysis, and alerting. It is built on top of Elasticsearch and MongoDB. It offers role-based access control and audit logs. Again as it is a full stack logging platform it would not be would not be practical or cost effective. [19]

Table 3 gives an overview of the pros and cons of alternative solutions.

Tool	Pros	Cons
Loki and Grafana	- already implemented in the company - cost-effective	- too short retention periods - difficult to set up different access policies
ELK	- Well-know, good documentation	-Other logging system implemented in company X - not cost effective solution
Graylog	- all in one tool	-Other logging system implemented in company X - not cost effective solution

Table 3. Alternative Solutions

4 Realisation

This chapter provides an overview of the implementation process of the logging system. For confidentiality reasons, service, Kafka topic, Kafka cluster, database, table and field names have been anonymized. The implementation was done together with an another developer from the email team.

4.1 Introduction To Chosen Approach

As the amount of data that needs to be stored is large, and the requirements expect a low maintanence cost solution, the cheapest option would be store data in a database. In this case databases would be the place where to save and store logs. Based on the use cases and requirements the saved data should include email html bodies, subjects, recipients, as well as email sending speeds and attachment sizes and other parameters mentioned in the Table 2. It was also mentioned that the data should be separated based on sensitivity, so dividing it into two databases would be a logical solution.

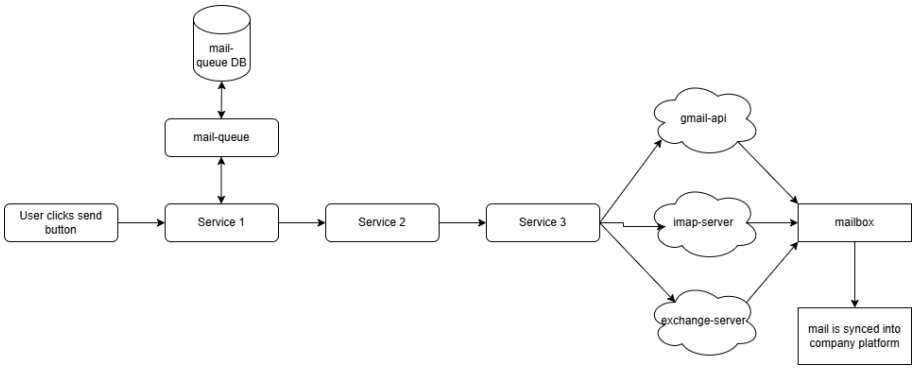

Figure 2. Existing sending flow in company X

As a next step, it is important to understand where should the logs be created. Figure 2 shows the current architecture of the sending feature backend. It would make most sense to add logging to Service 3 as this is the last service before the email is sent to provider and would contain the most information about the sending process.

4.2 Designing Architecture

Taking into consideration the existing architecture for email sending and the requirements mentioned in the previous chapter, the logging system should not influence the sending flow. Making requests directly to the databases from Service 3 could slow down sending performance. Additionally, before data is logged, it must be filtered and separated based on sensitivity. Therefore, it is reasonable to implement a new service for saving data to the databases. To enable this service to receive data from Service 3, Kafka, which is already in use within the company, can be utilized. In this setup, Service 3 would publish events to a Kafka topic and continue its operations without noticeably affecting its performance.

Using Kafka also makes sense because it allows logs to be queued. For example, if a large number of emails are sent simultaneously, Kafka can buffer the log messages in cases where emails are sent faster than they can be written to the database.

It is also important to consider that there are two different kinds of data that need to be processed. One is metadata, which should be easily accessible to all email team developers and SEs. The other is sensitive data, which a person should request access to. Since Kafka supports multiple consumers on a single topic, it makes sense to use just one Kafka topic for both types of data.

In the new service, there are two separate consumers: one for metadata and one for sensitive data. Each consumer processes, formats, and logs the data into its respective database. The final architecture is shown in Figure 3.

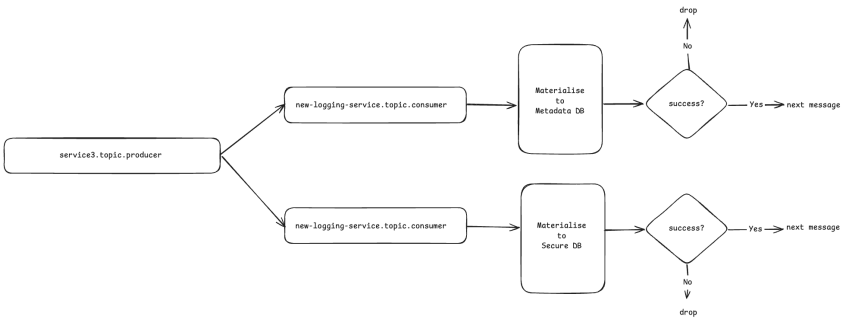

Figure 3. Architecture for the Logging System

The initial idea was to listen to database changes to verify whether the data had been saved correctly. If not, the message would have been redirected to a retry topic (Figures 4, 5).

However, after discussion, and considering the service requirements, namely, that it should be easily maintainable and have as simple structure as possible, it was decided that no retry topics would be used and there would be no listening to database changes. Another reason the retry topic was disregarded was the high pace of work in the team. Since this service does not directly impact customers, issues would likely receive lower priority. In such cases, messages in the retry topic could remain unfixed instead of being addressed immediately.

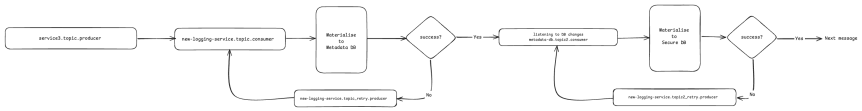

Figure 4. Suggested Architecture. Example 1

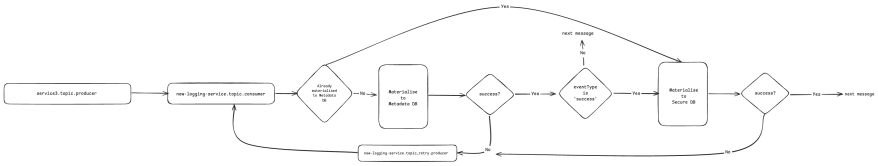

Figure 5. Suggested Architecture. Example 2

4.3 Implementing Data Collection

This section gives an overview of the process of implementing data collection. This process included modifying excisting services, creating the base for a new service.

4.3.1 Modifying Existing Services

First, it was necessary to modify existing services to make all necessary data accessible in one place. One of the required data metrics was email sending time. If an email was scheduled, it was important to exclude the time between pressing the send button and the actual scheduled time. To achieve this, a parameter called sendingStartedTime was added to Service 1 (Figure 2), so that the sending duration could later be calculated in Service 3. Some modifications were also made to Service 3 to calculate the sending time from the platform and the provider's side. Prometheus was used to set up initial metrics for email sending time, and Grafana was used to create graphs based on this information. These graphs were needed since the development of the logging system was expected to take

some time.

4.3.2 Base for the New Service

The top programming languages used in the company are TypeScript, JavaScript, Go, Python, and PHP. The technology stack of the author's team consists of approximately 63.11% JavaScript, 32.3% TypeScript, 1.2% Go, and 3.39% other languages. Therefore, it would be reasonable to select a language already present in the existing stack, as this simplifies maintenance and ensures that developers are familiar with the chosen language. Go (or Golang) is an open-source programming language developed by Google. While it offers great performance, it has less integration support compared to TypeScript and JavaScript. Between the two, TypeScript is a more suitable option, as it is strongly typed and can help detect unexpected behavior in code, thereby lowering the likelihood of bugs. [20][21]

Fastify is a fast and low-overhead web framework. It was chosen as it was already in use and configured within the company. A company-specific Fastify template was available that offered built-in support for Logger, schema validation, plugin decorators, and Kafka handlers. Using this template to create the service base automatically generated various configuration files like deployment YAML files and GitHub configuration files. That made deployment easier later on. Additionally, the template handled the installation of the necessary packages required for service setup.

4.3.3 Creating a Kafka Topic

The first task was to choose an existing cluster where the new topic would be created. Since there was a dedicated cluster for email related services, it made sense to use that. Next, the partition count had to be defined. Considering the volume of data expected to pass through the topic, it was decided to set the partition count to 8. Based on the data volume passing through Service 3, the retention policy was set to two days or 10 GB, leaving some buffer in case the data volume would increase unexpectedly. After that, user roles and access rights were defined. Service 3 needed write access as it would contain the producers connected to the topic, and the new service required read access as it would start consuming the messages. The infrastructure team created new Kafka topics for production and test regions based on these specifications.

4.3.4 Adding Kafka Producers

After the Kafka topic had been created, producers could be added to Service 3. Since Service 3 is a JavaScript service, one option was to use the KafkaJS package. However, the company already had an internal Kafka library built on top of KafkaJS. This library included preconfigured settings, error handling, and partition selection. Therefore, it made more sense to use it instead.

The producer logic is located in a separate file. It contains two functions: one for initializing the producer and another for publishing messages. If publishing a message to a Kafka topic fails, the error is caught and logged to Loki. Additionally, Prometheus counters are implemented to track the number of successful and failed message publishing attempts to Kafka. Before data is produced to the Kafka topic, it is filtered from requests and parameters inside Service 3. The message payload consists of three parts: eventType, metadata and either messageData or error. The first part shows whether email sending was suggested or not. If the Kafka message structure were compared to a log structure, this could be considered similar to the verbosity level. The metadata part contains the metadata information, including different IDs and sending speeds. If email sending was successful, the messageData part is added to the Kafka event containing the email content and recipients. However, if the sending fails, error messages with the cause for failure are added instead. Figure 6 displays an example of a Kafka message content about a failed email sending event.

Figure 6. Example of a failed Kafka message body

To validate if the messages were being produced correctly and to better analyse if the selected topic sizes were correct, Kafka topic size changes were measured in all regions over the course of a week (Appendix 6). The collected data included dates, regions, topic sizes, and the age of the oldest message in each topic. Measuring the age of the oldest message was important because, although the retention period was set to two days, Kafka does not delete messages individually, and this caused some initial confusion. Kafka divides messages into segments, which are deleted based on time or disk usage. A segment containing more new than expired messages will not be deleted.

4.3.5 Adding Consumers to the New Service

To handle the incoming Kafka messages, two separate consumers were implemented within the new service: one for non-sensitive metadata and another for sensitive data. Each consumer had its own dedicated handler file to keep the logic modular and easier to maintain.

The **metadata consumer** is responsible for processing general email-related metadata that will be later logged to the metadata DB. This includes calculating the total size of email attachments (if present) and appending relevant metadata such as error messages in case sending the email message failed. The total attachment size is calculated by summing the individual attachment sizes provided in the Kafka message.

The **secure data consumer** handles potentially sensitive content such as email subjects, recipients, and HTML bodies and also the metadata that will be later logged to the secure data DB.

Both consumers have defined schemas to validate that incoming data contains all the necessary information. The schemas were implemented using `@sinclair/typebox`, which builds JSON Schema objects in memory and maps them to TypeScript types. It can be used to define more complex schematics and helps to ensure that only valid and expected data structures are passed on to the service layer. Also, a validation error handler was added, and when the payload format is not in line with the schema, errors are thrown and logged until the broken message gets fixed. [22]

4.3.6 Creating Databases

The infrastructure team created the databases. However, before the setup could begin, several decisions needed to be made, for example, how much data would be stored, which services would require access, and which regions would be included. Two separate databases were needed, one for secure data and another for metadata. A separate database instance would be created for each production and test region. The service that required write access to the databases was the new service developed for the logging system, as this service was responsible for saving data into the databases.

For the metadata database, it was estimated that approximately 1.5 GB of information per region per month would be written to that database. This information would be required to be retained for 12 months, as this information would be used to analyze email sending performance throughout the year. For the secure database, the expected storage need was around 6–8 GB per region per month. The databases were implemented using MySQL, which aligns with the company's standard technology stack.

After the infrastructure setup, a developer from the email team created the schemas for the metadata and secure data tables. The data was divided between the two databases based on the sensitivity level assessment described in Section 3.4.3 and also considering the wishes from support engineers and developers. For example, even though the error message was not considered sensitive, based on the developer's feedback, there was no need to write this information to the metadata database. The final structure of the database tables can be seen on Figure 7. A composite primary key consisting of ID and timestamp was added to each table to optimize database performance.

4.3.7 Renaming the Service

Due to the service name being out of line with team naming practices, the service needed to be renamed. The renaming process included archiving the previous service repository, updating the Readme, removing the project from SonarQube, removing the service instances from Kubernetes and coordinating with relevant teams to remove associated resources like credentials.

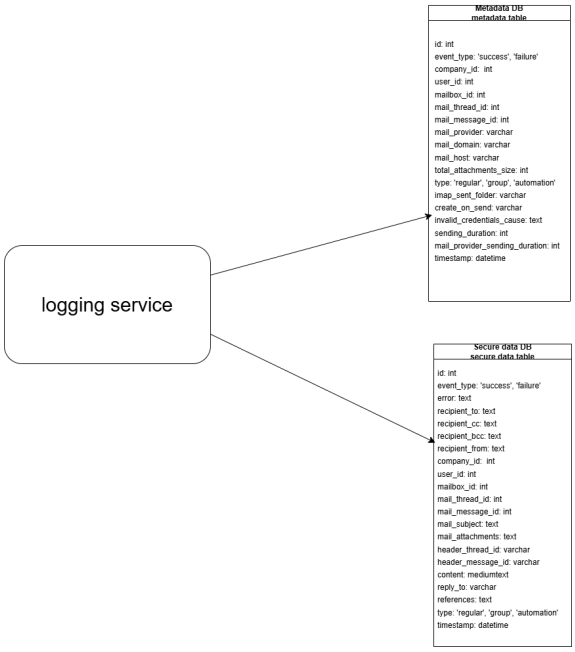

Figure 7. Database structure

4.3.8 Writing to Databases

Another developer from the email team implemented writing data to the databases using knex.js, because the Fastify template offered built-in support for it. They used Fastify plugins to set up database connections and created repository and manager layers to organize and handle the logic for writing data to databases.

First, the logging started only in one region and only to metadata database. After verifying that everything functioned correctly and was logged in the correct format, logging was then enabled for the remaining regions after two weeks. The same approach was used for the secure data database. The rollout was slightly delayed because one of the databases was migrated to a private cluster due to the large volume of data it was expected to handle.

4.4 Data Deletion and Data Deletion Validation

The initial idea was to handle data deletion on the database level using procedures and events (Figure 8). This approach was considered, since it seemed the most straightforward solution.

However, as this approach was against company architecture decisions, the next plan was to use events alone. On the downside, events can consume significant space and negatively

```
DROP PROCEDURE IF EXISTS delete_rows_older_than_3_months;

DELIMITER //

CREATE PROCEDURE delete_rows_older_than_3_months()
BEGIN
    DELETE FROM `sensitive_data`
    WHERE `timestamp` < NOW() - INTERVAL 90 DAY;
END//

SET GLOBAL event_scheduler = ON//

DROP EVENT IF EXISTS Delete_rows_older_than_3_months_hourly_event//

CREATE EVENT Delete_rows_older_than_3_months_hourly_event
ON SCHEDULE EVERY 1 HOUR
STARTS (NOW())
DO
BEGIN
    CALL delete_rows_older_than_3_months();
END//
```

Figure 8. Example of procedures and events

impact database performance. In the end, it was decided that handling the deletion in the service would be safer, as users might not have all the necessary rights from the database side, and replication could be broken if too many delete requests were made. Handling deletion in the service would also be more aligned with company architecture decisions. However, as the amount of data in the databases was to be quite large, it made sense to group data on the database level to make deletion easier. For that purpose, partitions were used to group data by months. Therefore, the partitions were added to the table schemas by another developer in the email team.

To make queries for data deleting in the service, corresponding logic was added to the repositories and managers. For that rawquery method was needed because truncating specific partitions is not supported by the standard query builder in Knex (Figure 9).

The implemented logic also included calculating the partitions that needed to be deleted.

```
db.rawQuery(`ALTER TABLE \`${TABLE_SECURE_DATA}\` TRUNCATE
    PARTITION
${partitionsToBeCleaned.join()}`
```

Figure 9. Query for truncating partitions

Once the functions for truncating partitions on the service side were implemented, a solution was needed to trigger the deletion process at specific intervals. Cron jobs are a commonly used solution in cases like this. Initially, the deletion was scheduled to occur on the first day of every month. The schedule was later updated, but that is beyond the scope of this thesis. The cron package was added to the service, and the schedule was defined using standard cron syntax.

4.4.1 Validating Data Deletion

The first plan to check whether deletion was successful was to count all rows in the table that were outside the calculated threshold with the worry that some rows were not belonging to partitions (Figure 10).

```
db.knex
    .table<TableName>(TABLE_SECURE_DATA)
    .count('* as total')
    .where('timestamp', '<', db.raw(`NOW() - INTERVAL ? MONTH`, [
        thresholdInMonths]))
    .first();
```

Figure 10. Query for selecting undeleted rows

However, this query began failing due to timeout caused by the large volume of data in the databases. Since there was no documented case where it would be possible for a row to not belong to a partition, there was no need to mitigate this risk. Therefore, it was decided to use partitions for checking deletion success as well, but with a larger time threshold and by running the check at a later time. Same logic, as for calculating the partitions that should be truncated, was implemented. The cron job was planned to run twice a month.

When logs were not deleted on time, it was important to notify the email team developers. Because of that, alerting needed to be set up. A Prometheus gauge was used to collect

metrics about the undeleted logs. Since the template used for service already included Prometheus support, setting up the metrics was straightforward. The Prometheus metrics were then sent to Grafana, where the alerts could be configured. The alert was set to trigger when the number of undeleted logs would be bigger than zero.

4.5 Implementing Access Control

As shown in Chapter 3.3, the support engineers of the email team were intended to be responsible for reviewing access to the secure data database, as they were expected to be its primary users. However, this approach did not align with the company's default access policy, and therefore a new access policy needed to be created. Although this was initially expected to be a quick process, it turned out to be more complex. Access to the sensitive database was first granted to the email team, as giving access to support engineers required making changes to the internal tooling. These changes were implemented by the security team and once the necessary fixes were completed, access could finally be granted to the SEs.

The process of setting up access involved discussions with all teams mentioned in Chapter 3.3. During one of these discussions, a suggestion was made. This was to enable direct access to the database via an internal tool used within the company. However, since this tool currently does not support MySQL queries and adding the support was expected to be a complicated process, the idea was not pursued further. Additionally, the tool should have been comfortable for the support engineers to use and offered support for using saved queries, for example. Therefore, transitioning to a new tool could have required more preparation than initially anticipated.

4.6 Creating Dashboards with Grafana

Grafana was already used in the company for visualisation, making it a good choice for fulfilling the visualisation part of the goals mentioned in chapter X. First so that Grafana could be used as a visualisation tool, the metadata databases needed to be added as data sources to Grafana, each region as a separate source. This was done by the infrastructure team. After that, dashboards could be created based on the databases.

Another developer from the email team was responsible for creating the dashboards. One dashboard included a graph for successful and failed sends, a query station, and various graphs for comparing average sending times by mail-providers and mail-hosts. It also displayed email sending counts per mail-provider, mail-domain, and mail-host, as well as graphs for average attachments sizes and the largest attachments. In the top bar, it is possible to enter the parameters to search by, and based on them, a SQL query is made to the database, and the results are displayed under the query station. There is a separate dashboard for each region, as it is no longer possible to use multiple data sources within a single dashboard and filter based on source. Regions, however, are an important parameter for filtering data. Part of a dashboard can be seen on Figure 11

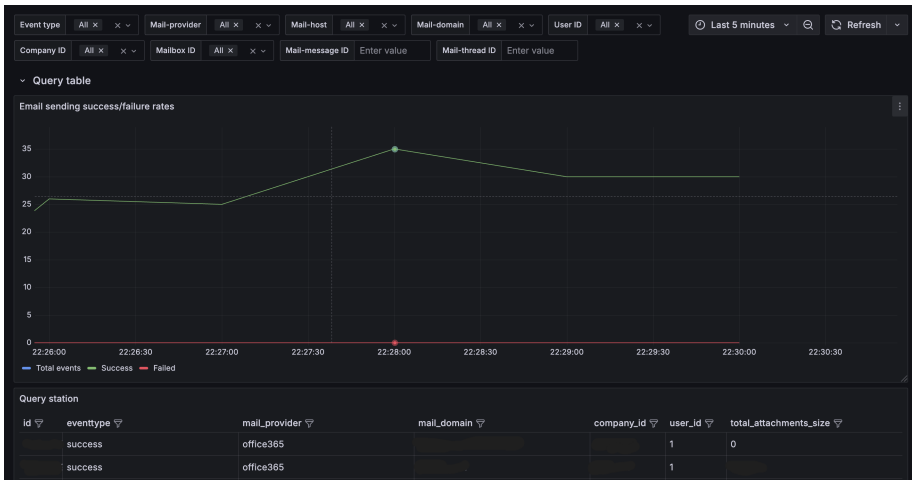

Figure 11. Part of Dashboard

When testing the dashboards, it was discovered that fetching data took a lot of time. In an attempt to improve the querying speed, indexes were added to the tables. The index consisted of company ID and user ID as these are the two most used parameters by support engineers and developers when searching for information. As a result, the querying speed improved a bit, but there is still room for improvement.

4.7 Writing Tests

In order to validate that the logic was functioning correctly and would do so after changes were done later on, automated tests were added to the service. There were tests testing the whole flow starting from injecting a Kafka message to the service and checking that it gets logged correctly to mocked databases. There were also unit tests, that focused on specific

components. Altogether there were 18 tests that covered 98.23% of the lines not excluded by the test configuration files. The tests were written using jest because there is built-in support in the Fastify template.

As a lot of the functionality was related to time, e.g. the service included methods with the purpose of validating if the database contained logs older than the threshold, a challenge arose on how to write such tests that would not need updating every few months because the time had gone out of limit. For that, a special function was added among test set-up functions that would fill Secure DB with rows for testing. It took in four arguments: db, correctRows number, oldRows number, and age threshold in months. Based on the arguments it would generate a given number of correct and old rows in the given database. However, the function was not perfect and had in fact many flaws. For example, if the threshold in days would be smaller than the number of correct rows, then incorrect rows would be produced to the databases. Also when calculating the threshold from months to days it did not take into consideration that different months have different lengths.

5 Validation

As part of this thesis, a secure logging system was developed to provide a better debugging tool for support engineers to solve email sending-related issues at company X. The development of the project began in June 2024 and continued until November. Although there was an initial plan for the final desired outcome, the early planning phase could have been more thorough, which led to certain decisions being only worked out during the implementation. Five different teams were involved in the process of developing the logging system. However, it was primarily implemented by the email team. In addition to the thesis author, another developer from the email team contributed to the project, primarily focusing on implementing data writing to the databases and creating the dashboards for querying and visualizing metadata.

5.1 Comparison Against Initial Goals

In chapter 3.4, specific requirements were brought out that the logging system should fulfill. These were mostly related to logging system usage and were validated by talking with support engineers and developers and collecting their feedback. Several use cases were identified that the logging system should support. The first was that it should assist support engineers in solving client issues. Slack conversations following the release of the logging service show that it has been helpful in resolving at least one case related to a missing attachment, by clarifying whether the issue originated from the provider or the platform. In the collected feedback it was also said that the system has been useful in quickly identifing the composition of the message. Previously cases like these required a lot of back and fourth between the technical team, developers and customers but now support engineers are able to check this information in one place. However, there have also been cases where the system did not provide the needed insights, such as different problems related to email parsing. As for the second use case, no relevant incident has occurred yet and therefore it is not possible to validate if the purpose got fulfilled as expected. Regarding the third use case, the dashboards offer a clear overview of sending metrics and provides the option to

filter the results as needed.

During the implementation process, stakeholder needs were followed. Support engineers have access to sending related metadata and sensitive data. Email team developers can filter data to analyse performance. Best practices at the DB level were followed. Access to sensitive data was limited.

Technical requirements can be validated by analysing the available data about service performance and again by collecting developer feedback. There was no noticeable drop in sending speed performance after the logging system was implemented.

The reliability of the service can be doubted, as there was an issue where logs did not get saved to the databases for a period of time because the incoming messages were not in line with the message schemas. This went unnoticed for some period as there was not enough alerting set up. This was also brought out in the feedback collected from the support engineers. On the other hand, in terms of querying speed, it was said to be better than some other internal databases. Additionally, the alert for undeleted logs has not been triggered. From the code aspect, it was said that the code is well structured, but may have some naming inconsistencies.

5.2 What could have been done better?

One important thing that could have been done better is the planning at the beginning of the project. If it had been more thorough, it would have helped to save quite a lot of work, for example, by avoiding renaming the service. Better definitions of the goals and requirements in the beginning would also have helped with clearer communication.

5.3 what can be done next

One possible next step would be to implement the recommendations made in the collected feedback. This includes adding additional information to logs for example automation ids, updating the ReadMe with instructions on how to use the Grafana dashboards as well as well as with a link, settting up alerts based on the information in the metadata db for specific hosts failing for example. Additionally, database performance could still be improved, as the dashboards can sometimes be quite slow.

A possible bigger project would be to create interface for the sensitive data via internal tooling in the company X to make it more comfortable for support engineers to access this data.

6 Summary

The purpose of this thesis was to develop a secure logging system that could be used to debug email-sending-related issues on the platform of company X.

After the planning stage, this was implemented by creating a system that consisted of Kafka, a service for filtering the data, and two databases where the data was saved. One of the databases was for storing metadata and the other was for sensitive data with a special access policy set up. The database containing metadata was added as a data source to Grafana for visualising, searching, and analysing the data.

To validate whether the system met its goals, feedback was collected from support engineers and developers. Based on their feedback, the system was useful for solving some client issues, though not all. It did not slow down email sending or interfere with other services. The system reliability was questionable as one issue related to the service went unnoticed due to insufficient alerting. In the future, the system could be improved by logging more information, such as automation IDs, adding more alerting, or by implementing an interface for the sensitive data via internal tooling.

References

[1] Amazon Web Services. *What are log files?* https://aws.amazon.com/what-is/log-files/. Accessed: 23.04.2025. 2025.

[2] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation)*. https://eur-lex.europa.eu/eli/reg/2016/679/oj. Accessed: 2025-05-02. Apr. 2016.

[3] Antonio Pecchia et al. "Industry Practices and Event Logging: Assessment of a Critical Software Development Process". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. 2015, pp. 169–178. DOI: 10.1109/ICSE.2015.145.

[4] Kevin Schmidt, Chris Phillips, and Anton Chuvakin. *Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management*. Newnes, 2012.

[5] Saurabh Chhajed. *Learning ELK stack: build mesmerizing visualizations, and analytics from your logs and data using Elastisearch, Logstash, and Kibana*. Packt Publishing Ltd, 2015.

[6] Rui Ding et al. "Log2: A cost-aware logging mechanism for performance diagnosis". In: *2015 USENIX annual technical conference (USENIX ATC 15)*. 2015, pp. 139–150.

[7] Roozbeh Aghili, Heng Li, and Foutse Khomh. "An Empirical Study of Sensitive Information in Logs". In: *arXiv preprint arXiv:2409.11313* (2024).

[8] Guidewire Software. *Logging Sensitive Information (PII)*. Accessed: 2025-05-15. URL: https://docs.guidewire.com/security/secure-coding-guidance/logging-sensitive-information-PII/.

[9] Liam Daly Manocchio et al. "A configurable anonymisation approach for network flow data: Balancing utility and privacy". In: *Computers and Electrical Engineering* 118 (2024), p. 109465.

[10] Council of Europe. *Explanatory Report to the Protocol Amending the Convention for the Protection of Individuals with Regard to Automatic Processing of Personal Data (CETS No. 223)*. https://rm.coe.int/cets-223-explanatory-report-to-the-protocol-amending-the-convention-fo/16808ac91a. Accessed: 2025-05-02. 2018.

[11] Brenda M Michelson. "Event-driven architecture overview". In: *Patricia Seybold Group* 2.12 (2006). Accessed: 2025-05-01, pp. 10–1571.

[12] Apache Software Foundation. *Apache Kafka - Introduction*. Accessed: 2025-05-17. URL: https://kafka.apache.org/intro.

[13] Boyuan Chen and Zhen Ming (Jack) Jiang. “A Survey of Software Log Instrumentation”. In: *ACM Comput. Surv.* 54.4 (2021). ISSN: 0360-0300. URL: https://doi.org/10.1145/3448976.

[14] Andreas Aßmuth et al. “A secure and privacy-friendly logging scheme”. In: *arXiv preprint arXiv:2405.11341* (2024). Accessed: 2025-05-13.

[15] Boyuan Chen and Zhen Ming Jiang. “Studying the use of java logging utilities in the wild”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 397–408.

[16] Grafana Labs. *Loki: Like Prometheus, but for logs*. Accessed: 18.05.2025. 2024. URL: https://grafana.com/oss/loki/.

[17] Grafana Labs. *Grafana: The open observability platform*. Accessed: 18.05.2025. 2024. URL: https://grafana.com/grafana/?pg=hp&plcmt=lt-box-data-visualization.

[18] Joakim Eriksson and Anawil Karavek. *A comparative analysis of log management solutions: ELK stack versus PLG stack*. 2023.

[19] Inc. Graylog. *About Graylog*. Accessed: 18.05.2025. 2024. URL: https://graylog.org/about/.

[20] TypeScript Team. *TypeScript in 5 Minutes*. Accessed: 2025-05-18. 2025. URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

[21] The Go Authors. *Getting Started - A Tour of Go*. https://go.dev/doc/tutorial/getting-started. Accessed: 2025-05-18. 2024. URL: https://go.dev/doc/tutorial/getting-started.

[22] Sinclair. *@sinclair/typebox*. https://www.npmjs.com/package/@sinclair/typebox. Accessed: 01.06.2025. 2025.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Annabel Pugi

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Development of a Sensitive Data Logging Service", supervised by Tarvo Treier and Kim Naciscionis
    1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
    1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright
2. I am aware that the author also retains the rights specified in clause 1 of the nonexclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.06.2025

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive licence shall not be valid for the period.

Appendix 2 – Kafka Topic Size Measurements

Kafka Topic Size Measurements

	Region 1	**Region 2**	**Region 3**	**Region 4**	**Region 5**	**Region 6**
Size						
05.08.24	1.118 GB	397.029 MB	1.404 GB	730.888 MB	1.176 GB	729.879 MB
06.08.24	1.747 GB	1.151 GB	1.96 GB	1008.713 MB	1.719 GB	1.395 GB
07.08.24	1.302 GB	1.838 GB	1.507 GB	1.293 GB	1.266 GB	1.988 GB
08.08.24	1.884 GB	1.391 GB	1.122 GB	676.829 MB	1.781 GB	1.476 GB
09.08.24	1.355 GB	1.837 GB	1.525 GB	989.768 MB	1.163 GB	1.813 GB
Oldest message						
05.08.24	4 days	3 days	4 days	4 days	4 days	4 days
06.08.24	5 days	4 days	5 days	5 days	5 days	4 days
07.08.24	3 days	5 days	2 days	6 days	2 days	5 days
08.08.24	4 days	3 days	2 days	2 days	3 days	2 days
09.08.24	5 days	4 days	2 days	3 days	4 days	3 days