UNIVERSITÀ DI TRENTO

Department of Information Engineering and Computer Science

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics and Computer Science

Master's Degree in
Cyber Security

Final Dissertation

# $k$-Anonymization Module Prepared for Change

*A flexible architecture for the easy replaceability of algorithms and data sources with proof-of-concept implementation*

Student
*Kristóf Majkut*

Supervisor
*Prof. Andrea Rossato*
*University of Trento*

Academic year 2022/2023

# Abstract

As part of their services, Würth Phoenix has been helping its customers comply with the General Data Protection Regulations. However, building up compliance requires a considerable amount of investment in terms of both time and money. Given that the regulation does not apply to anonymous data, instead of setting up a compliant data management system of high complexity, companies might opt for data anonymization, thus freeing themselves of any obligatory compliance measures. This is why anonymization has become of interest to Würth Phoenix as a potential pathway for extending their GDPR-related service portfolio. This thesis proposes a flexible architecture for an anonymization module based on the $k$-anonymity model. The flexibility of the architecture makes it possible to easily replace both the algorithm and the database technology so that the changing requirements of the company and its customers can be promptly addressed. A proof-of-concept implementation serves as a demonstration of how this architecture works in practice. In addition to the theoretical discussion, it highlights practical challenges, limitations and trade-offs to be taken into account in future implementations.

# Contents

# Chapter 1

# Introduction

As a software service company, Würth Phoenix offers a wide range of services to its customers, among others in the field of system management and cybersecurity. The software product of the company, NetEye, is a holistic system management tool that makes the centralised monitoring and controlling of the company IT infrastructure possible.

With the coming into force of the GDPR, i.e. General Data Protection Regulation of the European Union, in 2016, all companies managing personal data needed to put serious effort into strengthening their data protection and privacy measures. Thanks to the system monitoring, log management and alerting capabilities already included in NetEye, Würth Phoenix could help its customers to make their operations GDPR-compliant. Besides offering already available services, the company has also been interested in extending NetEye with additional functionality that facilitate other aspects of compliance with the regulation. For example, the R&D team leveraged the blockchain technology in order to introduce strong guarantees for log integrity.

Anonymization is one of the fields that has gained traction due to the GDPR. The regulation does not apply to anonymized data. This means that once a dataset has been rendered anonymous, the owner of the data is not obliged to build up the complex compliance measures any more. Given that the regulation has meant a heavy burden for the majority of the companies, anonymization, as a potential "escape route", has raised their curiosity. And this is also how Würth Phoenix got interested in the integration of anonymization into their service portfolio.

This thesis project aims to come up with an anonymization solution integrated into NetEye. When it comes to use cases, it is log anonymization that currently seems the most relevant for Würth Phoenix and its customers. Since NetEye uses Elasticsearch for log management, initially the idea was to focus entirely on anonymization inside Elasticsearch. However, taking into account that the company might decide to pivot to another log management tool, which is a realistic scenario with the rapidly evolving technological landscape of today, it seemed more reasonable to consider a higher abstraction level. Therefore, this thesis proposes an anonymization module with a flexible architecture that makes the replacement of both the data source and the anonymization algorithm easy.

When considering the various use cases where rendering data anonymous makes sense, an anonymization taxonomy has started to take shape. This helps to categorise the use cases, and also facilitates the discussion about which anonymization model is relevant in which scenario. Using the proposed taxonomy, the sort of log anonymization that Würth Phoenix is most interested in belongs to static, one-time, server-side anonymization. As this can be addressed with $k$-anonymity, the proposal for the anonymization module focuses entirely on this model.

The thesis project goes on to provide a proof-of-concept implementation of the proposed anonymization module. The goal of this proof of concept is twofold. On the one hand, it aims to demonstrate the feasibility of the proposed architecture, and prove that the abstractions indeed work and the components are easily replaceable. On the other hand, it helps to find and pinpoint any practical weaknesses, challenges or limitations that have not been take into account in the theoretical proposal. Besides these two points, the proof-of-concept implementation can also help the company initiate an internal discussion about anonymization. Developers and other stakeholders can get a real feel of how data anonymization works, they can test their assumptions and eventually decide which path to follow when attempting to integrate anonymization into NetEye.

The proof-of-concept implementation comes with two database connectors and two algorithms. One of the databases that the algorithms can interact with is Elasticsearch. This one is of the greatest interest for Würth Phoenix, as they offer their log management services using this technology. The other database for which a connector is implemented in the proof of concept is MySQL. Considering that the most popular data management systems are still SQL-based[3], it is a reasonable choice to include in the proof of concept. When it comes to the algorithms, an implementation of Datafly and Mondrian are provided. Datafly is one of the earliest $k$-anonymization algorithms. Mondrian was proposed some years later. It is a greedy algorithm, still frequently

referenced in scientific papers. Furthermore, the proof of concept is preconfigured for the anonymization of two datasets. One of them is the well-known UCI adults dataset, commonly used to demonstrate the functioning and performance of anonymization algorithms. To showcase the anonymization of data types other than the usual numerical, categorical and hierarchical ones, the proof of concept comes with a configuration file for the artificially generated web logs shipped with the Kibana tool. Even though semantically this anonymization does not make sense, it is sufficient for demonstration purposes.

The chapters of the thesis go through the above mentioned topics in detail and are structured as follows: Chapter 2 introduces the context of the work in general. It briefly presents the history of Würth Phoenix and its main activities. Then, NetEye, the system management product of the company, is presented, along with a short introduction into the Elasticsearch-Logstash-Kibana Stack which constitutes the log management component of NetEye. The GDPR and anonymization is presented next, followed by a section on $k$-anonymity with the two algorithms used in the proof-of-concept implementation.

Chapter 3 lays out the main motivation of the work, introducing an anonymization taxonomy, relevant use cases, and state of the art anonymization solutions. It then goes on to narrow down the scope of the problem and clarifies the exact problem to address.

Chapter 4 is dedicated to the proposal of a solution for the previously defined problem. It presents the flexible architecture of the anonymization module, along with the details of the two main components.

Chapter 5 describes the details of the proof-of-concept implementation. The sections elaborate on the implementation of the Elasticsearch and MySQL backends, as well as the Datafly and Mondrian algorithms. They highlight what were the main challenges that had emerged along the way and how they had been addressed. The datasets and the corresponding configuration files are also presented in detail. Finally, the anonymization of new data types is introduced.

Chapter 6 summarises the most important takeaways from the implementation of the proof of concept on the one hand, and from the generation of and working with the anonymous datasets on the other hand.

Chapter 7 talks about the limitations of the thesis, and Chapter 8 goes on to propose some directions for further developing the presented solution. Finally, Chapter 9 concludes the work with a brief summary of what has been achieved in the thesis project.

# Chapter 2

# Context

## 2.1 The company

The Würth company was established, with a primary focus on selling screws, in 1945 by Adolf Würth[45]. It was his son, Reinhold Würth, who, taking over the company, turned it into a global trading business, and eventually an international group. The company has its origins in the sale and distribution of assembly and fastening material, and, as a world market leader by today, "the Würth Line companies continue to serve this segment [...]. In addition, the Würth Group comprises the Allied Companies trading under their own name. Their business activities are either closely related to the core business or diversified and are developed further and expanded successfully"[46].

An exquisite example of the latter is Würth Phoenix[47], founded in 2000 as part of the Würth Group. Initially, the company provided services solely for the Würth Group, continuously expanding its activities and also starting to develop their own, internal solutions. Their products and services were eventually made available to the external market in 2007. The customers of Würth Phoenix are primarily located in Italy and Germany, but the company provides services to enterprises in various European countries, and also internationally to other members of the Würth Group[33].

Würth Phoenix is a software service company, today offering a wide range of solutions in the field of Business Intelligence (BI), Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), System Management and Cybersecurity. The company boasts of a series of strategic partnerships with companies like Microsoft, Atlassian or Elasticsearch. There is a constant collaboration with these partners to provide services of the highest quality and to cater for all demands of the Würth Phoenix customers.

## 2.2 NetEye

Their system management solution, NetEye[44], was originally developed for the Würth Group. The entire IT infrastructure of the organization can be mapped into, monitored and controlled through the product. Once the customer onboarding is done and NetEye has been deployed successfully, Würth Phoenix, on demand, provides further consulting and assistance to maintain the current setup or address any newly emerging requirements.

NetEye builds upon various open source tools. Among them is Icinga[21], a powerful infrastructure monitoring tool that provides NetEye's foundation and basic frame. "Icinga's modular architecture allows on the one hand to reuse all of its modules within NetEye, on the other hand it enables the NetEye team to develop new modules and integrate them seamlessly within the existing infrastructure"[16].

NetEye incorporates five major categories of service. The first one, constituting the core functionality of the product, is monitoring. Building upon the capabilities of Icinga, monitoring can be activated for entire systems, specific networks or desired services. The monitored objects provide an overview of, and when required, more detailed insights into the health of the IT infrastructure of the organization through dashboards and performance graphs. The option to create detection rules helps to spot anomalies and act quickly on the malfunctioning of some system components.

The second major category is IT operation analytics (ITOA). Features like complex event processing facilitate the investigation of correlations and cause-effect relationships. ITOA "complements the monitoring functionalities by collecting telemetry data from network traffic and from systems and applications"[16].

Thirdly, NetEye also comes with application performance monitoring (APM), leveraging the respective module of Elastic[6]. As the name suggests, this functionality zooms in on the performance and overall health of applications. By quickly identifying anomalies and spotting abnormal behavior, APM helps the organization react and address the root cause of the problem, and thus assists in providing a reliable, high-quality experience for end users.

The fourth pillar of the product offers service management and incident response. The creation and management of organizational IT assets, remote control over hosts, software-as-a-service solutions and ticketing systems provide additional assistance in streamlining the management of the IT infrastructure.

Last, but not least, NetEye is shipped with a security event and information management (SIEM) solution, building upon the open source Elasticsearch-Logstash-Kibana technology stack (ELK Stack). Working with SIEMs starts with the collection and processing of machine logs. They are first ingested, normalized according to a schema and enriched with any additional information. At this point, the logs are ready to be queried, visualized, and analyzed manually or in an automated manner. Through predefined detection rules, SIEMs assist in finding and keeping track of anomalies, threats and vulnerabilities present in the monitored systems. Sophisticated alerting mechanisms make it possible to react in time and facilitate the investigation of any suspicious behavior occurring on the machines under supervision.

## 2.3 The ELK Stack

The ELK Stack[42], comprising Elasticsearch, Kibana and Logstash, is the result of years of evolution. The first component, that eventually triggered the evolution, is Elasticsearch, a document-based, distributed database. Document-based, that is it stores unstructured data. Its distributed nature means that multiple instances of Elasticsearch can be run at the same time on multiple machines. On the one hand, this provides high performance and scalability when it comes to data ingestion, manipulation and querying. On the other hand, the database being distributed across multiple instances comes with robustness, reliability and high availability: even in the case of a subset of the underlying machines breaking down, the ones still alive can seamlessly take over the workload.

Elasticsearch is built on top of an open source search engine called Lucene. The Lucene library comes with "powerful indexing and search features, as well as spell checking, hit highlighting and advanced analysis [...] capabilities"[4]. Communication with Elasticsearch happens through a REST API over HTTP. The ingestion of new documents, modification or deletion of existing ones, querying and security settings are all carried out via the web API, transmitting data in JSON format.

Kibana provides an intuitive graphical user interface (GUI) for Elasticsearch. The majority of the functionalities available through the API is directly accessible through the Kibana GUI. The visual overview helps to get a holistic understanding of the monitored systems, thus facilitating for example the management of Elasticsearch indices and security roles. In addition, building on the query API of Elasticsearch, Kibana comes with a powerful toolset for data visualization through charts of all sorts. Charts, representing different views of the data, can then be added to dashboards. Applying filters to your dashboard will update all of the charts, thus letting the user carry out a visual drill down into the desired aspect of the data.

Logstash serves as a powerful collector, transformer and forwarder of data. It supports a wide variety of data sources and formats, from reading files and listening on a port for Syslog messages to directly pulling documents from Elasticsearch indices. Once the data has arrived, it is pushed through pipelines based on predefined conditions. Logstash can carry out practically any sort of data transformation. Once the processing is finished, data is then sent to an output. Elasticsearch is only one potential destination, through Logstash it is possible to write the result of the transformations to a file or forward it to any web server via the desired protocol, like UDP, TCP or syslog.

The original ELK Stack was extended with the concept of so-called Beats - and at a loss where to fit the "B" into the ELK acronym, the development team decided to refer to the technology stack as the "Elastic Stack" from that point on. Beats are light-weight data shippers that are meant to read or listen for incoming data. Having carried out some basic preprocessing, for instance the parsing of the timestamp in Syslog messages, Beats can forward the documents to Logstash for further processing or directly send them to Elasticsearch for ingestion.

Beats allow the additional transformation and enrichment of data before forwarding it. Its real strength, however, comes not from the option to manually manipulate data, but from prepackaged modules, so-called integrations. Endpoint security software, firewalls or Windows services generate logs in widely varying, custom formats. It is not rare that documentation, or at least a detailed one, is not publicly available. Thus, the manual parsing and normalization of various logs is usually cumbersome and time-consuming. Vendors address this problem precisely, by writing and publishing their own integrations, which are practically vendor- and product-specific log parsers.

Beats might be installed on each machine that is meant to be monitored. Activating and managing integrations after the installation requires directly accessing each of these machines around the organization. And this, from a scalability point of view, is problematic. This issue led up to the latest evolution in the Elastic Stack, namely the introduction of Elastic Agents and Fleets. Elastic Agents provide an abstraction over Beats which facilitates their configuration and the workflow in general. Fleets provide central access to all Elastic Agents installed on the monitored machines of the organization, and make it possible to add or remove integrations

and modify their configuration directly through Kibana.

The Elastic Stack comes with an immense amount of functionalities. These are spread across different components of the technology stack. Certain capabilities are even duplicated and can be found in various parts of the stack.For example, the enrichment of logs is available through Beats and Logstash processors, but the same functionality can be implemented through an ingest pipeline directly in Elasticsearch. The point of entry into the stack is again a choice of the operations team. As Figure 2.1 shows, there are multiple ways for logs to reach Elasticsearch.

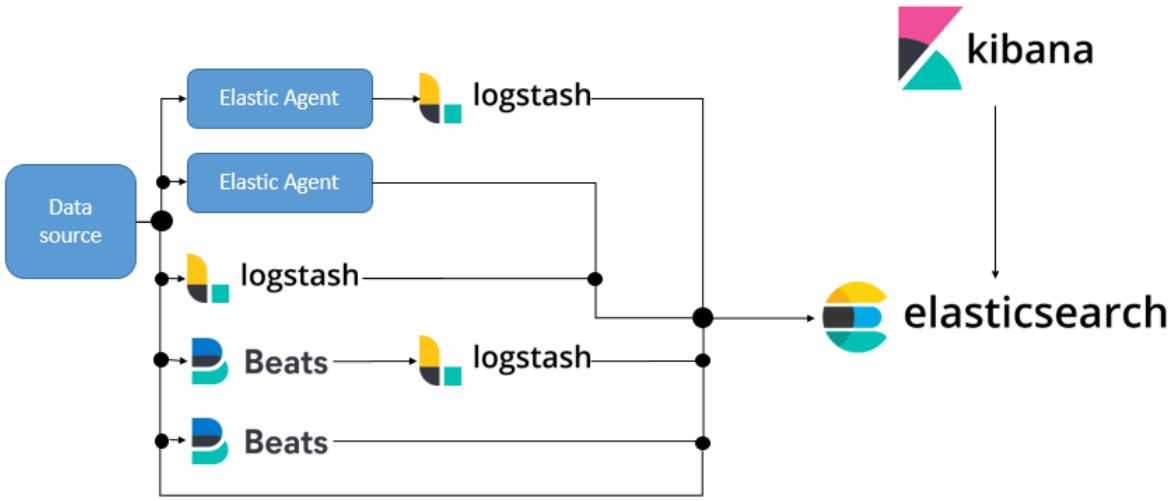


Figure 2.1: Various ways for ingesting data into Elasticsearch

They can arrive through a Beat or an Elastic Agent. However, if needed for some specific reason, these two can forward logs first to Logstash. And a log source can also directly send its output to Elasticsearch. Thus, the Elastic Stack provides huge flexibility so that organizations can tailor their setup according to their exact needs and address as many use cases as efficiently as possible.

## 2.4 GDPR

In Europe, privacy was first discussed and declared as a fundamental human right during the European Convention on Human Rights in 1950. It took almost a half century for the next formal milestone to arrive. With the rapid advancements in technology, the concept of privacy had to go through profound revisions. As personal data processing was increasing continuously and seemed to be getting out of hand, the European Union answered with the Data Protection Directive of 1995, also known as Directive 95/46/EC[14]. Restating that privacy was indeed a fundamental human right, and emphasizing the harmonization of the internal market as a main motivation, the directive set out a series of guidelines. Based on these the member states were then expected to enact national laws restricting the processing of personal data and guaranteeing the right to privacy to all of their citizens.

By 2012, it became apparent that the directive had not been able to sufficiently address its original aims. On the one hand, nobody anticipated the pace of technological development. Personal computers, the internet, finally followed by smartphones have become accessible for anyone around the European Union, and, importantly, at an affordable price. In 15 years, the majority of the European population stepped into cyberspace in one way or another. Browsing the internet, using digital services, shopping online, posting on social media sites and chatting through various applications have become everyday activities. Accordingly, the magnitude of the data being generated per second exploded. And a significant portion of this data was personal.

On the other hand, the harmonization of the internal market was not particularly successful either. Given that the directive was only meant to show a direction that the national legislation should then have interpreted and implemented, member states ended up with rather differing privacy laws. Keeping track of and adhering to these turned out to be at times a direct obstacle, but at least a heavy burden for companies with the intention of providing cross-border digital services.

Thus, the European Commission started working on a draft in 2012, keeping in mind the trend of rapid technological change and the excessive data processing, neither of which showed any signs of slowing down in the years to come. Legal certainty across the Union, as a means of facilitating cross-border services and cooperation, also became of vital importance. This led to the pivoting from the directive format to the definition of a

regulation, which was finally approved and published in 2016 under the, by now well-known, infamous name: the General Data Protection Regulation[37], or the GDPR in short.

The primary focus is on data protection and privacy. Besides, the regulation also emphasizes that companies must provide more transparency about their data management practices towards their customers. Customers must be made aware of what sort of data is being collected about them, how it is collected, which parties have access to it and are involved in the processing, what the customers' exact rights are and how to exercise them. And all this must be communicated in a clear, simple and comprehensible manner, avoiding legal jargon or endless complex sentences that usually hinder comprehension and make people consent to terms they do not even understand. In general, access to, information about and deletion of one's own personal data must be available on a short notice and without excessive administrative steps. Users must have the opportunity to reject privacy policies. Furthermore, short of legitimate interest service providers cannot track many user activities before the users give explicit permission by opting in such tracking on a consent form.

Companies were granted two years of preparation time, thus the GDPR only became applicable in 2018, at this point repealing the directive of 1995. The regulation is rather extensive. It sets out complex, but at the same time abstract requirements that allow plenty of room for interpretation. The rationale behind this is to remain technology-agnostic so that the regulation remains applicable even as the technical context keeps changing. All in all, however, this extensiveness, complexity and abstraction left quite some uncertainty around how to approach compliance. The two-year preparation period was indeed necessary for companies to experiment with and eventually clarify what constitutes compliance in practice.

### 2.4.1 Terminology

The regulation comes with a list of definitions clarifying the terminology used throughout the document. For the upcoming discussions of this work the following four concepts are the most relevant and are worth being elaborated on: personal data, processing, data processor and data controller.

According to the GDPR, "'personal data' means any information relating to an identified or identifiable natural person ('data subject')". As alluded to by the bracket at the end, another piece of terminology is resolved here right away: "data subject" refers to an identified or identifiable natural person. The paragraph then goes on to define what makes a natural person identified or identifiable: it is someone "who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person". In short, a piece of information must be regarded as personal data once the individual, whom the data describes, can be identified through some combination of the information available at the given time and place.

Processing is defined as any sort of manual or automated operation carried out on personal data, such as "collection, [...] structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, [...] erasure or destruction". That is, it concerns, not surprisingly, the entire data lifecycle from ingestion to deletion. The GDPR also makes an important distinction between two types of entities that have access to personal data. A "'controller' means the natural or legal person, [...] which, alone or jointly with others, determines the purposes and means of the processing of personal data; [...]". Once the exact purposes and the means are clarified, it is the "processor", who, as the name suggests, processes, i.e. works with and manipulates the data on behalf of the controller. This distinction is relevant, there are significant differences in the obligations concerning one and the other.

Outsourcing to third-party digital service providers has become a common practice. For instance, website operators can receive advanced insights into the behavior of their users through third-party analytics tools. Or, webshops can share the purchase history of their customers with external recommendation service providers in the hope of thus improving their advertisement campaigns. The controllers are the companies outsourcing, while the processors are third-party service providers. The GDPR also explicitly addresses working with and handing over the personal data of European citizens to entities operating in non-EU countries. Even in such cases, as stated by Recital 101, "the level of protection of natural persons ensured in the Union by this Regulation should not be undermined". Chapter V explicitly clarifies what measures data controllers must make to ensure that processors in third countries or international organizations provide the same guarantees as set out and required by the GDPR and that European citizens can exercise their rights and enjoy the same level of data protection.

### 2.4.2 The "way out"

Heeding the claim of the frequently recited "data is the new gold", companies, regardless of their size, have been trying to collect and analyze as much of their customers' data as possible. However, simply the registration of information about employees is already considered personal data processing. Accordingly, the GDPR applies to virtually every organization. And the wide range of obligations turned out to be a huge challenge for many, from a technical as well as a financial point of view.

Thus, it is no wonder that, with the promise of freeing data controllers from GDPR compliance, Recital 26 of the regulation has drawn plenty of attention. It states that "data protection should [...] not apply to anonymous information". The paragraph defines anonymous information as one "which does not relate to an identified or identifiable natural person" and any "personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable".

While the definition of personal data already explains the notion of identifiability, Recital 26 lays some further ground to help with the interpretation of the concept: "[t]o determine whether a natural person is identifiable, account should be taken of all the means reasonably likely to be used, [...], either by the controller or by another person to identify the natural person directly or indirectly." Two main points are worth unpacking in this sentence. Firstly, when evaluating identifiability, it is not only third parties, but also the data controllers themselves that must be taken into consideration, as they can potentially identify individuals, regardless of whether it happens intentionally or by happenstance. Secondly, "all the means reasonably likely" refers to "all objective factors, such as the costs of and the amount of time required for identification, taking into consideration the available technology at the time of the processing and technological developments". Arguably a rather challenging requirement that introduces yet another layer of abstraction and vagueness.

The text of the regulation goes on to point out the important distinction between anonymization and pseudonymization. While in the process of anonymization all direct and indirect ways of identifying an individual are made impossible, the data subject belonging to a pseudonymous record is easily reidentified in one simple step. In the process of pseudonymization direct identifiers are replaced with another unique identifier. For instance, the ID number, name and birth date are removed from every record, and a new, seemingly random set of characters, derived from these three values, is added as a new attribute. The mapping between the original values and the newly generated identifier is only known to the data controller. The regulation regards pseudonymization as a means of privacy protection, since people with access to and working on pseudonymous data are not able to identify the underlying individuals directly. Still, the data controller or anyone managing to get access to the "pseudonym-to-direct identifiers" mapping is capable of deanonymizing the dataset and accessing all the information about the data subjects. For this reason the GDPR does not consider the output of pseudonymization as anonymous data, and accordingly, the regulation applies to any pseudonymous dataset.

As quickly as anonymization, presented to be a means to freedom from GDPR compliance, might raise one's hopes, reading Recital 26 from beginning to end makes it clear that anonymization is not a simple and quick workaround. Similarly to other parts of the regulation, the wording of the definitions above leave quite some room for interpretation. With the intention of remaining as universal as possible, and with the introduction of technical concepts into the legal domain, the regulators undoubtedly set a hard challenge for those who must make sense of the GDPR.

Some practical clarification is provided by the Data Protection Working Party 216 of the European Data Protection Board. They published an opinion on anonymization techniques in 2014[32] with useful, more technical guidelines on, among others, what criteria must be met for a dataset to be considered properly anonymized. Before looking into this, however, let us take a look at what anonymization actually is.

## 2.5 Anonymization

With the anonymization of personal data the goal is to make sure that the information, output at the end of the procedure, cannot be linked back to the individual it originally belongs to. This requirement can occur, for example, in the field of research. Detailed medical databases about patients are invaluable sources of information, especially with the rapid development of machine learning. However, it is rather hard to find any individual that would willingly make their medical history public. Therefore, by anonymizing such data, researchers are able to hide the identity of individuals while retaining the information they are interested in. Other than a layer of privacy protection, anonymization also serves as a means of risk mitigation against cyber attacks. If the organization only stores an anonymous version of its clients' data, even in case of a successful intrusion and subsequent extracting of databases, personal data is not leaked in any way. This can save the company the loss of their clients' trust on the one hand, and immense fines on the other hand, that would be applicable due to the GDPR.

To make further discussion and the demonstration of concepts easier, let us assume that the Table 2.1 below, containing personal information about four individuals, has been extracted from a medical database. The columns are called attributes, representing one piece of information about the individuals, while the lines, called records, show all the data per individual contained in the database.

When it comes to personal data, attributes can be divided into two main categories: personally identifiable and sensitive information. Personally identifiable information (PII) can be further separated into direct and indirect. Direct identifiers are attributes based on which an individual can be directly singled out, such as the number of one's identity card, or in some cases, your name. In the sample dataset above, for example, Julia Miller and Sarah Parker can be singled out by their name with complete certainty, thus for them the name

| ID | name | date of birth | address | weight | marital status | illness |
|---|---|---|---|---|---|---|
| 1 | Julia M | 1970.01.22. | 12,A St,York | 62 | never married | tuberculosis |
| 2 | John S | 1968.04.12 | 24,B St,Hull | 82 | divorced | liver cancer |
| 3 | Sarah P | 1977.12.12. | 36,C St,York | 70 | married - civil | lung cancer |
| 4 | Peter T | 1972.12.02. | 48,C St,York | 75 | married - AF | kidney fail. |
| 5 | John S | 1974.05.01. | 60,B St,Hull | 90 | widowed | liver cancer |
| 6 | Carl B | 1980.09.03. | 72,C St,York | 85 | married - civil | heart dis. |
| 7 | Frank W | 1969.08.30. | 80,D St,York | 65 | never married | liver fail. |
| 8 | Julia M | 1970.01.22. | 12,A St,York | 62 | never married | tuberculosis |

Table 2.1: Personal information and illness of individuals from a fictional medical database

attribute is a direct identifier. However, in the case of the two individuals called John S., the same attribute is not enough to tell exactly which record belongs to whom. For them, their name, along with their date of birth, address, weight and marital status are indirect identifiers. Having these pieces of information separately, it is not possible to single out an individual, but combining more of them can already prove to be sufficient for identification. Indirect identifiers are also called quasi-identifiers.

Personally identifiable information is thus any information that makes the direct or indirect identification of a natural person possible. Sensitive data, on the other hand, covers special categories of personal data that "merit specific protection as the context of their processing could create significant risks to the fundamental rights and freedoms" of individuals. Any personal data is deemed sensitive that "reveal racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the [...] genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation". In Table 2.1 the illness attribute contains sensitive, health-related information.

The first obvious step in anonymization is the complete removal of any direct identifier. From this point on, there are essentially three ways to publish an anonymized view of the dataset. First, the data can be anonymized record-by-record, the output of which is anonymized microdata. It looks similar to the input, but some attributes are removed and the value of others are modified in a way to prevent the identification of individuals. For instance, an anonymized view of Table 2.1 above could look like Table 2.2. It is a 2-anonymous dataset, the definition and details of which is provided in the next subsection. As a second option, anonymity is also achievable through publishing only an aggregate view of the data. Referring again to Table 2.1, instead of making microdata publicly available, the aggregation statement can be published: "50% of the hospital's patients have cancer". The best-known model for aggregation-based anonymization is differential privacy[15]. Thirdly, synthetic data can serve as yet another means of anonymization. In this case, through mathematical methods the distribution of the attributes is extracted, and using these properties of the original dataset new, artificial data can be generated. Despite being practically fake, synthetic data shows the same statistical characteristics as the dataset it is based upon.

| sex | date of birth | address | weight | marital status | illness |
|---|---|---|---|---|---|
| * | 1969-71 | York | 62-65 | never married | tuberculosis |
| male | 1965-75 | B St, Hull | 82-90 | * | liver cancer |
| * | 1970-81 | C St, York | 70-85 | married | lung cancer |
| * | 1970-81 | C St, York | 70-85 | married | kidney failure |
| male | 1965-75 | B St, Hull | 82-90 | * | liver cancer |
| * | 1970-81 | C St, York | 70-85 | married | heart disease |
| * | 1969-71 | York | 62-65 | never married | liver failure |

Table 2.2: A potential anonymized view of Table 2.1

### 2.5.1 Anonymization techniques

The focus of this work is on the first way of anonymization, i.e. publishing an anonymized view in the form of microdata. There are multiple anonymization techniques for the transformation and modification of the attribute values, the five most common ones being data masking, generalization, suppression, perturbation and data swapping[11].

During data masking, some portion of the attribute value is "masked", i.e. some digits or characters are replaced with other values. For example, instead of publishing a five digit ZIP code, like 41837, the last

three digits can be masked with a "*" sign, resulting in the value "41***". This still allows the approximate identification of the geographical region, but with lower precision, thus providing more privacy to the data subjects. Data masking in the example above is irreversible. However, encrypting the original value and storing the encryption key results in a reversible data masking. The encryption of the same value yields the same masked value, thus the relationships among records are retained. If the last three digits of the 41837 ZIP code is encrypted to "41abc", each record with this ZIP code will have the exact same encrypted value.

The second technique is generalization where some concrete attribute values are grouped together, thus constructing "generalized" information that hides the original one. Let's assume the dataset includes four records with the ZIP codes 41837, 48596, 57586 and 59647. The grouping "40000-60000" encompasses all four records. For reduced information loss, but also reduced privacy guarantees, two smaller groups can be created "40000-49999" and "50000-59999". There is some overlapping between generalization and data masking: the group of "40000-49999" has the same semantic meaning as the masked value of "4****". However, generalization is more expressive and flexible, "40000-60000" cannot be expressed with data masking any more.

Thirdly, anonymization is achievable through the complete removal or "suppression" of the attribute value. This technique practically intersects with the extreme forms of data masking and generalization. All information is lost both when masking out all digits in the ZIP code to obtain the "*****" value, and when creating one group, like "0-99999", to encompass the entire range of possible ZIP codes.

Data perturbation is yet another way for protecting the privacy of data subjects. In this case, some small, random noise is added to the original attribute value that for instance turns the ZIP code of 41837 into 41866. Thus, the "contours" of the data subjects are blurred, the perturbed information does not describe the original individuals any more. The resolution of the data is still maintained, and through well-designed random noise the statistical properties of the dataset remain the same. Thus, data analysts can keep working on seemingly raw data with the characteristics of the original one, and possibly extract the same insights that are present in the original dataset.

The final technique is data swapping. It does not directly manipulate data on an attribute-level, but swaps the values belonging to two different records. By swapping the ZIP codes 41837 and 78596, the resolution of the information also in this case remains high. However, from a statistical point of view, this technique might introduce significant distortions when considering the example of ZIP codes. This must be taken into account before deciding for this technique. For instance, trajectory anonymization is one domain where data swapping has already proved to be useful[38].

Depending on the use case and the requirements, any of the options above might come into handy. Regardless of the choice, however, anonymization ultimately always comes down to finding a trade-off between privacy and utility. The more the published information is distorted in the hope of hiding the underlying individuals, the less detailed, and thus the less usable the anonymized view becomes for data analytics. The engineering task is therefore developing models and algorithms that are capable of reaching the right balance: the privacy guarantees must be strong while keeping the resolution of the data as high as possible.

### 2.5.2 Opinion of the Data Protection Working Party

The Data Protection Working Party 216 of the European Data Protection Board points to three "essential risks" that must be taken into consideration during any anonymization: singling out, linkability and inference. Singling out refers to the possibility of pointing to the records of one exact individual in a dataset claimed to be anonymous. Linkability examines whether it is possible to link anonymized records of an individual or a group of individuals with other records belonging to the same individual or group of individuals in the same dataset or potentially in another one. The third risk to consider comes from inference attacks: one should evaluate if it is possible to derive, "with significant probability", the value of an attribute from a set of other attributes.

The Working Party states that "a solution against these three risks would be robust against re-identification performed by the most likely and reasonable means the data controller and any third party may employ". Thus, when evaluating whether the level of anonymity of a dataset complies with the notion of anonymity according to the GDPR, one can primarily rely on these three criteria.

## 2.6 $k$-anonymity

One of the models for microdata-level anonymization is $k$-anonymity, originally proposed by Samarati and Sweeney[39]. The basic idea is to group records into "buckets" of size $k$, so that it is not possible to tell the records inside a group apart from one another. As each group contains $k$ or more elements, someone aiming at the reidentification of an individual has at best $1/k$ probability of guessing which record belongs to their target, even if they can precisely point to the target's group. Thus, the privacy guarantee can be fine-tuned through the adjustment of the parameter $k$: the higher the value, the stronger the protection of privacy, but also the lower the resolution of the data.

Building a $k$-anonymous dataset, similar to any anonymization procedure and as presented already in the previous section, starts with the categorization of the attributes into direct, (indirect or) quasi-identifiers and sensitive attributes. All direct identifiers are removed and sensitive attributes are retained in their original form. The task at this point is to create the groups or the "buckets" from the quasi-identifiers. These groups are called equivalence classes. From the anonymization techniques presented in the previous section, the $k$-anonymity model relies primarily on generalization and partly on suppression.

Table 2.2 represents a $k$-anonymous, more precisely a 2-anonymous view of the dataset in Table 2.1. Deriving the sex of the individual from their name and using the other four quasi-identifier attributes, namely the date of birth, address, weight and marital status, three equivalence classes can be generated: *(*; 1969-71; York; 62-65; never married)*, *(male; 1965-75; B Street, Hull; 82-90; *)*, *(*; 1970-81; C Street, York; 70-85; married)*. The "*" symbol means that the value of the attribute is completely suppressed. The first two equivalence classes cover two, while the last one three records from the original dataset. That is, each "group" contains at least 2 elements which makes the anonymized dataset in Table 2.2 a 2-anonymous one.

While it might seem relatively straightforward to come up with a generalization just by looking at the data on a small example, translating it into an algorithm or code can prove to be trickier. Various categories of attributes can be differentiated when aiming at the generalization of data. The most common attribute types are numerical, categorical, hierarchical and date.

Numerical values, like the age or the salary of an individual, can be generalized rather simply, by placing them into ranges. In Table 2.1, the one numerical attribute is the weight: the values 70, 75 and 85 are generalized by replacing them with a range of "70-85".

Categorical attributes can be split into two further categories based on whether there is a natural ordering among the values. The level of education, for instance, comes with a natural ordering: primary school, secondary school, high school, bachelor's degree, master's degree, PhD, etc. Such categorical attributes can be encoded into numerical values, starting from 1 and incrementing by one for each new value: primary school - 1, secondary school - 2, high school - 3, etc. The value of the encoding is that from this point on these categorical attributes can be treated as numerical ones and the generalization again becomes trivial. If a natural ordering does not exist, as in the case of race, there is no way to generalize the values. The only two options are completely suppressing or retaining them in their original form during the process of anonymization.

The third type of attribute is called hierarchical. Such attributes are similar to categorical ones without a natural ordering. The difference is that a hierarchy tree can be defined for generalization. Let's take geographical areas as an example. From a specific address with street and house number, it is easy to zoom out and start reducing the precision of the data. In every step of the generalization the resolution of data is reduced. First, remove the house number, and publish only the name of the street. Then keep zooming out and share the residential area only, then the district, the city, the region, the country and finally the continent. It is not trivial (although in the case of addresses it might be feasible) to create an algorithm that is capable of this "zooming out" automatically for any arbitrary input it receives. However, it is possible to define a tree, a so-called generalization hierarchy, that has the original values in its leaf nodes and generalized values in its internal node. Such a tree is already easy to traverse programmatically, and thus makes the generalization of this sort of attribute possible. Figure 2.2 presents a generalization hierarchy for the marital status attribute of the frequently used UCI adult dataset[8].

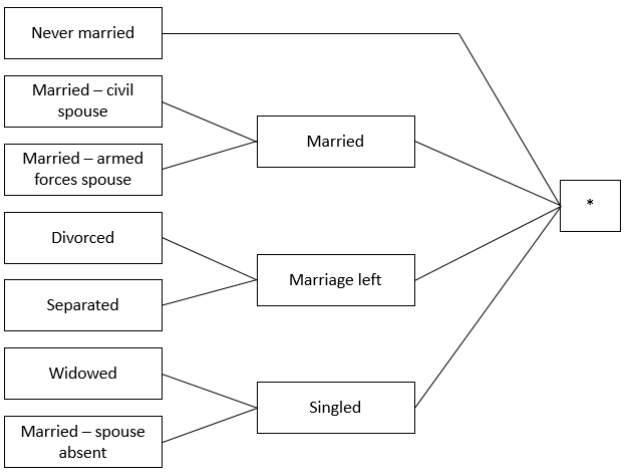


Figure 2.2: A generalization hierarchy defined for the marital status attribute of the frequently used UCI adult dataset

Finally, dates seem to be somewhere between numerical and hierarchical attributes. As long as the date field

only contains the year and perhaps the month, it is not particularly challenging to manage the generalization logic. When it is timestamps that must be generalized, with days, hours, minutes and seconds also included, the solution is less trivial.

### 2.6.1 Algorithms

Optimal $k$-anonymization means producing the maximal number of equivalence classes while keeping their size as close to $k$ as possible. Achieving this was proven to be NP-hard shortly after the proposal of the anonymity model[30]. At that point the main objective of researchers became to find and construct algorithms that are capable of approximating optimal $k$-anonymization as efficiently as possible.

Several algorithms have already been published and the research is still going on[23]. Depending on the efficiency of the algorithm used and the underlying dataset, the quality of the created equivalence classes might show great differences. Certain algorithms, for example, support the definition of overlapping equivalence classes that can reduce the information loss during the anonymization. If the dataset comes with outliers, they force the definition of equivalence classes that cover huge chunks of the domain. An unsophisticated generalization approach can produce equivalence classes that contain a number of elements way above $k$.

DataFly[40] and Mondrian[24] are two early $k$-anonymization algorithms that are used for demonstration purposes in this thesis project. The former follows a bottom-up, while the latter a top-down approach for carrying out the anonymization.

#### Datafly

The bottom-up approach of DataFly means that on launching the generalization process it creates an equivalence class for each individual record. The algorithm goes on to then merge these equivalence classes into new ones with increasing sizes, until all of them contain at least $k$ elements.

Datafly starts with grouping all records together that have identical quasi-identifier attribute values, thus creating a so-called frequency list. This contains all the unique combinations of quasi-identifier values present in the dataset, i.e. practically the equivalence classes inherently present in the dataset without any generalization step. For each equivalence class in the frequency list the number of records covered is also stored.

In each consecutive step, the algorithm checks whether there exists any items in the frequency list with less than $k$ elements, that is if there are any equivalence classes that still do not fulfill the expected privacy guarantee. If so, the algorithm finds the attribute with the most distinct values among the items of the frequency list, and generalizes this attribute in all of the records of the dataset. The algorithm runs as long as there are less than $k$ elements that do not belong to any equivalence class with a size greater than $k$. At this point these elements are simply suppressed, i.e. thrown away, and the algorithm terminates.The current state of the frequency list is returned, which represents the generated equivalence classes.

The original algorithm proposes some additional parameters through which the generalization process can. This work focuses on a simplified version, presented as pseudo-code in the paper of Sweeney[41]. Figure 2.3 shows a visual representation of how the Datafly algorithm generates the equivalence classes. Each circle represents a data point with two attributes: the first attribute takes values from the x axis and the second one from the y axis. Accordingly, the algorithm can generalize along either of these two axes.

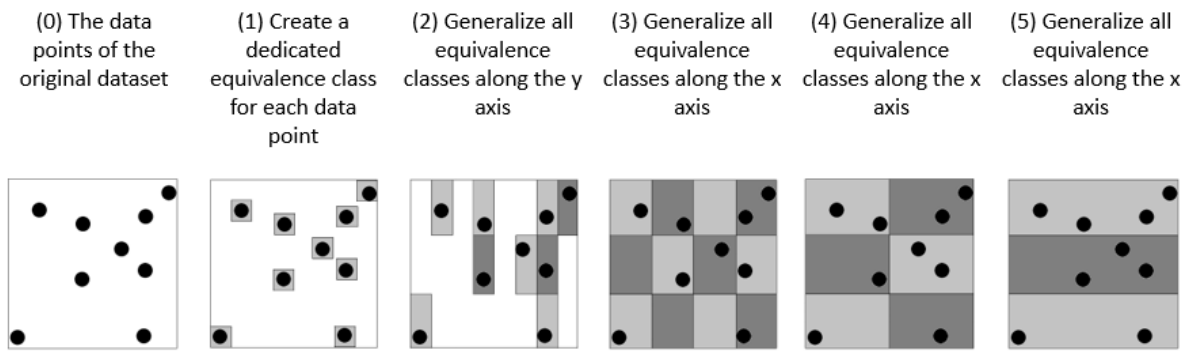


Figure 2.3: A visual representation of how the Datafly algorithm generates the equivalence classes through generalization

**Mondrian**

Mondrian's top-down approach refers to the fact that the algorithm starts with one equivalence class encompassing the entire dataset, and goes on to split this one equivalence class into multiple smaller ones. It is a greedy, recursive algorithm.

Its "greediness" means that the algorithm uses the heuristic of always picking the "widest" attribute, assuming that this will result in the most balanced split of the current equivalence class. The width of an attribute must be defined per attribute type. For numerical attributes it translates to the range covered. For instance, a generalized value of "70-80" has a width of 10. The width of the hierarchical attribute can be defined as the number of leaf nodes belonging to the current generalized value.

The algorithm is recursive: it practically operates similar to a depth-first tree traversal. It takes one equivalence class and splits it into two: let's call the first newly output equivalence class "left" and the second one "right". The algorithm goes on to split the "left" equivalence class, again obtaining a new "left" and "right" one. It moves on to do another split on the "left" one, and so on, until reaching a "left" equivalence class that cannot be split further. A split in invalid if the size of at least one of the two output equivalence classes is less than $k$. Then, it tries to split the "right" one, drilling down further in the "tree" through recursive calls until there are valid splits. Otherwise it moves back up in the "tree", that is along the recursive call stack.

Figure 2.4 shows a visual representation of how the Mondrian splits are executed. Similarly to Figure 2.3, each circle represents a data point with two attributes taking value from the x and the y axes.The algorithm can split the equivalence classes along either of these two axes.

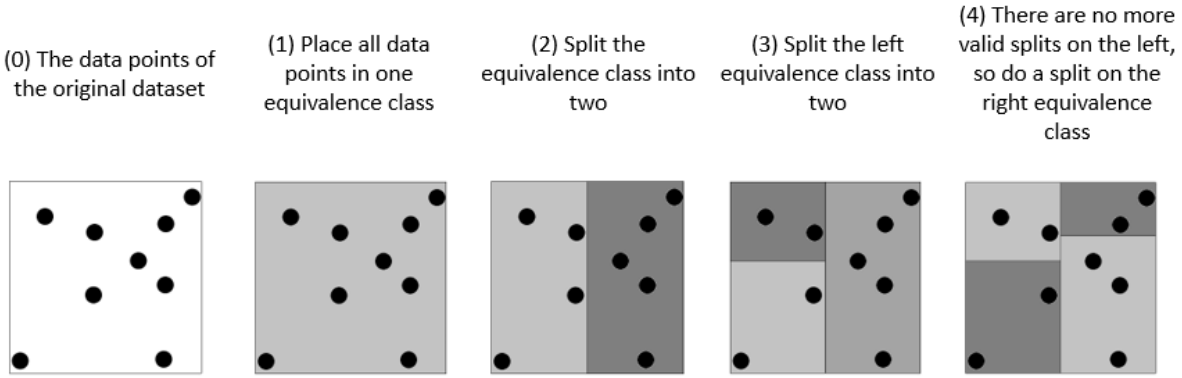


Figure 2.4: A visual representation of how the Mondrian algorithm generates the equivalence classes through the creation and splitting of partitions

**Comparison of the two algorithms**

The DataFly algorithm is way more rigid than Mondrian. The generalization in Datafly is synchronized across all equivalence classes: it forces all equivalence classes to carry out the same generalization step regardless of their current state and number of elements. DataFly can easily produce equivalence classes with very low resolution and a size way above $k$. Mondrian, however, by its "drill-down" approach introduces way more flexibility into the generalization procedure. It is capable of producing equivalence classes that approximate an optimal $k$-anonymization.

Comparing the steps and outcome of Figure 2.3 and 2.4 gives an intuition of rigidness of Datafly and the flexibility of Mondrian. Both algorithms "run" on the same set of data points and aim to produce a 2-anonymous dataset. Mondrian manages to split the eight data points perfectly into four equivalence classes. However, Datafly produces three bigger equivalence classes that result in a lower resolution outcome with more information loss.

### 2.6.2 Weaknesses and updated models

The Data Protection Working Party 216 of the European Data Protection Board also takes $k$-anonymity under the scope and points out common mistakes and weaknesses in connection to the anonymity model. While $k$-anonymity protects against singling out attacks, it is less robust when observed from the perspective of linkability and inference. As the Working Party points out, the major problem occurs when all records of one equivalence class have the same sensitive attribute values. Considering Table 2.1, the two people of the equivalence class *(male; 1965-75; B Street, Hull; 82-90; *)* both have liver cancer. Despite the anonymization attempt, if someone

knows that their target belongs to this specific equivalence class, they can with complete certainty say what the sensitive attribute value of their target is. The anonymity and the privacy of the individual is broken.

This shortcoming of the $k$-anonymity was addressed by an updated model, called $l$-diversity[27]. Other than hiding individuals in groups of size greater than or equal to $k$, $l$-diversity also makes sure that each equivalence class contains at least $l$ diverse sensitive attribute values. This prevents the most basic inference attack, presented above as the major flaw of $k$-anonymity. However, $l$-diversity is not without its weaknesses either. Probabilistic inference attacks are still possible against this model. Let's assume an $l$-diverse dataset, with $k = 20$ and $l = 5$: each equivalence class contains at least 20 records and at least 5 different values for the sensitive attribute. With, for example, 28 records in the equivalence class, it might happen that one sensitive attribute value is present 24 times, while each of the other four values occur only once. In this scenario, someone looking at the data can tell with high probability (namely 24/28) which sensitive attribute value an individual from this equivalence class possesses.

$T$-closeness[25] aims to prevent the above mentioned probabilistic inference attack against $l$-diverse datasets. It makes sure that not only does each equivalence class have more than $k$ members with more than $l$ diverse sensitive attribute values, but that the distribution of the sensitive attribute values also approximates that of the entire dataset. The parameter $t$ represents a threshold: a $t$-close dataset guarantees that the distance between the distribution of the sensitive attribute values in the original dataset and in any of the equivalence classes remains within the defined threshold.

The Working Party, while acknowledging the merits of the $l$-diversity and $t$-closeness, still emphasizes their concern that a linkability-based attack is possible against these models, given that they are based on the clustering mechanism of $k$-anonymity. Once it is clear which equivalence class an individual belongs to, anyone has approximately $1/k$ chance of correctly guessing which record belongs to their target, which is far greater than $1/N$, where $N$ is the size of the entire dataset.

# Chapter 3

# Motivation

Talking to one of the colleagues about various services that the company offers in the field of security, he mentioned that Würth Phoenix also assists customers with GDPR compliance from a technical point of view. A webinar of the company presented in 2020, but still available on their website, introduces the requirements of the regulation and presents how to explicitly address these with the help of the tools offered by NetEye.

Around the same time that my colleagues shared this webinar with me, one of the customers showed interest in the Elastic Blockchain Proxy of Würth Phoenix. It is an extension of the Elastic Stack, developed by the research and development team of the company. The proxy places each incoming log in a blockchain so that there is no way to manipulate the ingested information later on. This provides strong guarantees for data integrity during upcoming audits. The main purpose of the Elastic Blockchain Proxy was again facilitating GDPR compliance.

## 3.1 GDPR compliance with NetEye

With the coming into force of the GDPR in 2018, service providers and tools promising to facilitate the building up of compliance came into the centre of attention. The ecosystem of NetEye also contains a series of components and functionalities through which the requirements of the regulation, like data protection by design, the principle of accountability and the obligation of the timely reporting of data breaches, can be explicitly addressed.

Data protection by design, as set out by Article 25 of the GDPR, expects organisations to design and carry out their data processing activities with security and privacy in mind from the very beginning. Security Information and Event Management, that is SIEM solutions provide technical measures that can serve as a strong pillar of data protection. Their primary aim is to track and record the activities of users all around the IT infrastructure of the company. Any time data is accessed, changed or deleted, the account information along with the exact set of operations are stored. The more sensitive or valuable the data, the more detailed the records can be. The integrity of the information stored is secured through various mechanisms, like the definition of security roles. The Elastic SIEM Module integrated into NetEye provides the framework for designing an architecture that makes responsible data governance possible, with security and privacy guarantees in place.

As the website of the European Data Protection Supervisors explains, "[t]he General Data Protection Regulation integrates accountability as a principle which requires that organisations put in place appropriate technical and organisational measures and be able to demonstrate what they did and its effectiveness when requested"[1]. Once the technical guarantees are established through the Elastic SIEM Module in NetEye, a centralised, holistic overview of the entire SIEM system is made available through the graphical user interface of Kibana. Thus, it becomes easy for the Data Protection Officer of the organisation, the point of contact during GDPR audits, to demonstrate compliance and present the set of measures put in place.

As frequently mentioned, compliance is not fulfilled by a one-time deployment of a well-designed system. Instead, it is an ongoing procedure that necessitates the continuous supervision, maintenance and updating of the means of data processing and protection. The SIEM module assists in the designing, implementing and maintaining the security measures across the organisation. Starting to track newly deployed IT assets and activating the corresponding detection rules, tweaking security roles or involving the latest machine learning models for regular security checks are all available through the centralised view of Kibana.

"In the case of a personal data breach, the controller shall without undue delay and, where feasible, not later than 72 hours after having become aware of it, notify the personal data breach to the supervisory authority [...]", states Article 33 of the regulation[37]. When a data breach occurs, a series of questions must be answered about the IT assets around the organisation. Information about the physical devices, the softwares installed, network connections and access rights are essential to locate and isolate vulnerable points of the infrastructure. NetEye is also shipped with IT Asset Management capabilities that assists in yet another aspect of GDPR

compliance: addressing the obligation of timely data breach reporting.

There are a number of functionalities offered out of the box by NetEye through which compliance with the requirements of the regulation can be explicitly addressed. Furthermore, Würth Phoenix is actively looking to further extend its services that will assist their customers in complying with other obligations set out by the GDPR. A good example is the Elastic Blockchain Proxy developed by the R&D team of the company. In the case of some log sources, it might be required to prove their originality and their unmodifiable nature in general. With the Elastic Blockchain Proxy activated, the specified logs must pass through it. Each of them is signed and placed in a blockchain. Thus, it becomes possible to demonstrate during upcoming audits that all logs of the specified type have been kept in their original form. This, again, adheres to the principle of accountability.

## 3.2 Compliance through anonymization

My bachelor thesis also revolves around the data protection regulation and proposes a protocol for client-side anonymization[29]. The company's apparent interest in and its activities around the GDPR made me wonder whether they also have some services related to anonymization. As it turns out, anonymization is not part of the Würth Phoenix portfolio, primarily because the Elastic ecosystem does not come with supporting tooling in this field.

Nevertheless, the colleagues also mention that anonymization has been a recurring topic of discussion at the company. Promising freedom from GDPR compliance, anonymization attracted quite some attention after the publishing of the regulation. As colleagues from Würth Phoenix recall, this was also reflected in increased demand and interest in the concept from their customers. However, the NetEye ecosystem does not offer any solution for anonymization, thus these demands could not be catered for.

Even though the interest in anonymization gained quite some traction from the part of organisations, it disappeared almost just as quickly as it came. One of the colleagues suggests that, as far as their customers are concerned, there is one main reason behind the general loss of interest: the integration of fine-grained access control into Elasticsearch shortly after the GDPR became applicable.

It is not only the visibility of an index that can be defined per user role. Attribute- and document-level access control has also been made available in Elasticsearch. It is even possible to filter out documents of an index that a user is not allowed to see based on a predefined query. This also alludes to the use cases organisations deemed relevant in connection with anonymization: restricting access to and visibility of data. This demand is indeed easily satisfied through a fine-grained access control mechanism. However, this is also not the primary aim of anonymization.

Despite the decreased demand for anonymization, Würth Phoenix has not removed the topic from its agenda. Taking into account the Elastic Blockchain Proxy as a solid proof of the company being willing to invest in the development of new products and services, I proposed the development of an anonymization solution inside the Elastic Stack, leveraging the experience I gained during the bachelors. The company welcomed the idea.

## 3.3 Scoping

In my bachelor thesis, I propose a protocol that allows the client device to anonymize the data it intends to share with the server before sending it to. Both parties benefit from this. On the one hand, the clients do not need to give away personal data, thus their privacy fully remains protected. On the other hand, if the server only receives anonymous information, there is no need for the implementation of additional technical and organisational measures to render the data processing GDPR compliant.

Even though the thesis includes a proof of concept implementation, the work is more of a theoretical nature. The initial idea, therefore, was to provide a practical implementation based on the proposed protocol, embedding it in the NetEye, and more specifically the Elastic ecosystem. Given that during log collection there are plenty of client machines sending data with potentially personal information to a central server, the proposal seemed reasonable.

### 3.3.1 Use cases

Before moving on with the discussions, it was essential to clarify some use cases that are of relevance to the company. The most frequent goals of data anonymization are data monetization, publishing of open data and building trust. The first and the third one are essential and provide great value for practically any company. Accordingly, there is a set of anonymization use cases that can attract the attention of Würth Phoenix customers.

#### Data retention beyond the allowed time frame

In general, the regulation expects data controllers to store personal data for the shortest period of time possible. According to Article 5(1)(e), information that "permits identification of data subjects" should be kept "for no longer than is necessary for the purposes for which the personal data are processed". The article goes on to add that "personal data may be stored for longer periods insofar as the personal data will be processed solely for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes". However, for improving some business processes or the services provided, companies often mean to hold onto the information they obtained from their customers. This demand for information is constantly on the rise, since artificial intelligence, and more specifically machine learning models require as much data as possible for increasing the accuracy of their output.

One way to resolve the contradiction between the GDPR obligations and the need for data is to anonymize customer data. Anonymization provides a means to remove the time limitation on data storage. By rendering personal data anonymous and thus lifting the dataset out of the context of the regulation, data controllers are allowed to retain valuable historical data. Big technological companies, for example Yahoo[12] and Google[20], already applies this practice.

As one colleague points out, not only clients, but also Würth Phoenix itself could make use of extended information storage times. The company has a security operations centre, which provides SOC services to external companies. In case of an incident or detected anomaly in some part of the system, anonymized historical data can serve as a useful source of information for forensics and trend analysis, for example.

#### Sharing data with third parties

Sharing personal data with third parties, while possible, has strict constraints according to the GDPR. Data subjects must be informed about the circumstances of the processing and the involvement of a third party therein. Such processing must have a legal basis and the data subject must explicitly consent to it. Furthermore, an entire chapter, namely Chapter V of the regulation is dedicated to requirements of transferring data to companies or international organisations based outside of the European Union.

Instead of implementing complex compliance measures and building all of the security and privacy guarantees into the third-party data sharing procedure, data controllers can opt for the sharing of only anonymous information. This can significantly reduce the financial investments and time required for teaming up with an external organisation. The ultimate goal can be manyfold: outsourcing, creating partnerships and organising hackathons, for instance. Whichever the case, not having to dedicate resources to establishing compliance with the regulation provides more flexibility and removes several hurdles from the way of cooperating with any third party.

#### Additional layer of data protection

Unless it is essential to store the personal data in its original form for the business purposes of the company, anonymization can also provide an extra layer of data protection, and a means of risk minimization. The benefit of having only anonymous data is that even in the case of a data breach, personal information cannot get leaked. Additionally, data access can be provided to a wider set of employees that might facilitate everyday business processes and the workflow in general.

Of course, it is not possible to anonymize all the data that the company stores. Still, aiming to keep as many of the datasets anonymous as possible can help the company reduce risks and compliance-related costs. Furthermore, communicating it might also help to build trust and attract security-aware customers.

### 3.3.2 A taxonomy of anonymization

When considering anonymization use cases, some patterns start to emerge. There is a difference if the anonymization is carried out on the side of the client or the server. Some use cases require static, while others dynamic anonymization.. And when talking about the static one, anonymization can be carried in two ways: it is possible to anonymize an already ingested dataset in one go, or keep anonymizing data points continuously, as they arrive into the system. Organising these patterns into a taxonomy helps with understanding and discussing the various scenarios that come up.

Server-side anonymization is the most trivial. Probably it is the one that comes to mind when hearing about data anonymization in general. There is a machine collecting and storing incoming information in step one, which it then goes on to anonymize in step two. This counts as data processing: before producing the anonymized version, the data must be stored, preprocessed and then pushed through the anonymization procedure. As a result, this machine and the entire workflow must be compliant with the regulation.

The question might arise whether it is possible to spare even these compliance measures on the server side. This is where client-side anonymization comes into play. If data is already rendered anonymous on the device

of the user, then only anonymous data reaches the server machine. Thus, the server side practically never deals with any personal data processing, and accordingly does not have an obligation to comply any more. A similar solution is already present in Apple devices[5]. The company is interested in application and hardware usage information of various sorts that can be extracted from the devices of users. In order to hide the identity of users and guarantee their privacy, Apples relies on local differential privacy for forwarding only anonymous device usage data to their servers. Client-side anonymization can not only free the processor from compliance measures, but it also provides higher privacy and more control over data for end users by not letting any personal information leave their devices. Thus, it can also serve as a means to build trust with privacy-aware customers.

"Static" and "dynamic" refers to the data view obtained after having conducted the anonymization. In the case of the former, anonymous microdata is created. The outcome of the anonymization is static, one can access and work directly on the anonymous dataset on a record-level. One example of static anonymization is using any algorithm based on $k$-anonymity.

In contrast, dynamic anonymization does not produce one final anonymous dataset. Instead, an anonymization layer is placed on top of the original dataset. From that point on, each user query must pass through this layer. And, most importantly, the anonymization is carried out on a query-by-query basis. Differential privacy follows this approach, for instance. What makes dynamic anonymization powerful is its ability to tailor the anonymization to the needs of the current query. This ensures that the returned data is of the highest resolution possible. It is important to note, however, that noise addition, which is at the core of dynamic anonymization, has its limitations, as well. For one, after a certain amount of queries the added noise cancels out. This makes it necessary to define a privacy budget that puts a hard limit on the number of queries that are allowed to be made to the dataset[10]. Once this limit is exceeded, the privacy guarantees practically disappear. Thus, any subsequent queries must be blocked, and practically the dataset becomes unusable. With static anonymization, although the produced microdata is of lower quality, it has no limitation in terms of querying.

There is one more important distinction to make inside the domain of static anonymization: whether it is carried out once, or it goes on continuously. Let's assume a hospital wants to publish a subset of their patient data for research purposes, or a company organises a hackathon where the participants must extract insights from customer data. Since the data to share is personal, in both cases it must be anonymized first. These are typical cases, where a dataset or a segment thereof is anonymized once, and this chunk of data is sufficient for the intended purposes.

In some cases, when data keeps coming in continuously, the requirement is to create an anonymous view of the information right away. Data points must then be anonymized one-by-one or in small batches. The major difference compared to the one-time subtype of static anonymization is the availability of the data to digest. With one-time anonymization, the complete dataset is at hand: the statistical properties of the data can be taken into account and used to generate an anonymous view. The same statistical properties cannot be anticipated when anonymizing data as it arrives, since these properties are being shaped by each incoming data point. Therefore, managing anonymization in such a scenario is far from trivial.

Taking into consideration the various combinations from the above presented taxonomy, six different ways, presented in Table 3.1, emerge for approaching anonymization. All of them have their own needs and requirements, and might make good use of differing anonymity models. For example, $k$-anonymity aligns well with the microdata-level operations of static anonymization. In comparison, dynamic anonymization is usually based on some form of differential privacy. The attacker models and potential vulnerabilities to consider changes considerably as the discussion moves from client-side to server-side anonymization. Similarly, the assumptions to make vary widely depending on whether it is one closed dataset, or continuously incoming information that must be anonymized. Accordingly, each cell of Table 3.1 necessitates a dedicated solution.

| - | Static | | Dynamic |
|---|---|---|---|
| - | One-time | Continuous | Query-time |
| Client-side | | | |
| Server-side | | | |

Table 3.1: Combinations derived from the above presented taxonomy

### Practical examples

Beside the high-level use cases presented in the previous section, when zooming in from the perspective of this taxonomy, it is possible to propose some further, more practical use cases. Considering them might also help in grasping the distinctions between the various anonymization approaches. Since it is the anonymization of logs that is of interest for Würth Phoenix, the examples are taken from this domain.

The simplest scenario is the one-off anonymization on the server side. One use case, already mentioned in Chapter 2, is the retention of machine logs after the company would not have a legal basis to store personal

information any longer. The information is already located on some machines on the premises of the organisation. All the data points that should otherwise be deleted are rendered anonymous, and the original data is thrown away.

If a security operations centre works with highly sensitive data, the data controller, taking into account privacy-related concerns, might prefer to make only anonymous data available to the analysts. In this case, however, the anonymization would need to be carried out at least in near real-time in order not to disrupt the operations of the security team. Using static anonymization might still be possible. The server waits for the accumulation of a larger quantity of data, and carries out the anonymization in batches. However, the smaller the amount of data being anonymized, the worse the resolution of the anonymous data. This means that static anonymization might reduce the utility of the information to an extent that renders the incoming data practically unusable. A technique that allows the anonymization of incoming data points on the fly, and continuously populates the anonymous dataset would instead come in handy for this use case.

Client-side static anonymization can prove useful for employers, if they want to receive insights about the activities their employees indulge in during work. An employer, however, might not have the legal basis for such data processing. Although probably the consent of the employees would be still necessary, by carrying out the anonymization on the client machines, this might ease the mind of the employees and reduce their reluctance to share their activity logs. If the employer intends to get such statistics for a one-month period, the client machines can carry out a one-time anonymization at the end of the 30 days.

A demand for static, continuous anonymization on the client side might emerge when there is a need for accessing the data as soon as possible. Still considering employee activity logs, an employer might want to test how people around the company react to certain situations, for example to a simulated cyber attack. Continuously anonymizing logs and forwarding them might provide a more realistic view of the timeline and the employees' activities, as opposed to every machine sending the anonymized data at the end of the experiment.

Analysts who do not require a microdata-level drill down into data, but instead only need answers for aggregate queries, could make good use of dynamic anonymization. On the server side, this can be applied to any dataset that contains personal information. When it comes to client-side dynamic anonymization, Apple's data collection practices, as mentioned in the previous section, demonstrate a real-life example. It would be possible to collect end user or employee activity logs in a similar manner.

### 3.3.3 A better understanding of the needs of the company

The list of potential use cases along with the defined taxonomy facilitated considerably the conceptualization of and discussions about anonymization. Soon it became clear that client-side anonymization would not bring too much value for Würth Phoenix. Accordingly, focusing on the proposal of my bachelor thesis turned out to be the wrong direction.

Nevertheless, the use case from the domain of server-side anonymization comes with immediate applicability in the field of data retention. The desire to store valuable application usage data for business purposes after the allowed time frame can realistically occur for the majority of companies. Consequently, the focus of the thesis pivoted from anonymization on the client side to a server-side one.

## 3.4 State of the art solutions

Once the scope of the project is readjusted according to the needs and interest of the company, the next step is to have a look around and see what solutions exist inside the Elastic Stack or on the market in general for the anonymization of personal data. This helps in gaining a more profound understanding of the problem domain and provides additional insights, inspiration and ideas for an eventual solution. First and foremost, however, it assists in the identification of the gaps in terms of available services, and thus can further focus the project scope.

### 3.4.1 Inside the Elastic ecosystem

Unfortunately, the Elastic Stack does not come with any tooling for the complete anonymization of personal information. Despite the misleading naming of some solutions, the best one can achieve inside the ecosystem is pseudonymization.

Around the time of the GDPR becoming applicable, Anonymize-it, "the general purpose tool for data privacy", was published[19]. While the name sounds promising, the tool simply replaces the specified fields of documents with fake data with the same semantics. Leveraging the faker Python library, a wide number of data types, like email addresses, names or credit card numbers, can be generated in a semantically correct format. This is practically masking field values with synthetic data. As also the disclaimer of the article points

out, the tool "is not intended to be used for anonymization requirements of GDPR policies, but rather to aid pseudonymization efforts".

SearchGuard is an open-source security plugin for the Elastic Stack. It offers a functionality called "field anonymization". It allows the replacement of string fields with the hash of the original value or with another string derived from the original one through some string operation. Although the field is indeed anonymized, its value is practically completely suppressed. The fields with the same value will have the same hash, so some relationships in the data are retained, but the data utility is reduced almost to zero.

The same hashing functionality is already natively achievable through the fingerprint processor in the Elastic Stack. One of the articles in the Elastic blog presents how to leverage it for pseudonymization of documents in Elasticsearch[26].

The two currently available features are thus the suppression of fields and pseudonymization. While both of them serve as an additional layer of data protection, provide privacy guarantees and thus support GDPR compliance, they certainly cannot be used for producing anonymous datasets from Elastic indices.

### 3.4.2 External tools

The implementation of various anonymization algorithms are readily available in open repositories. Unfortunately, these are usually simple proof of concept projects that are rather limited in terms of scalability and flexibility. There are, however, three tools that come with more robustness: Amnesia[13], ARX[7] and Aircloak[2].

Both Amnesia and ARX are open-source anonymization tools. They offer some ways for loading data. Amnesia works with the $k$- and $k^m$-anonymity models. During the configuration phase, the user categorises the attributes, defines generalisation hierarchies and can specify the desired anonymization techniques per attribute. All this functionality is also available in ARX, however this tool offers a much richer toolkit in general compared to Amnesia. ARX is shipped with a wide range of anonymity models that the user can pick from. Various metrics are available for the evaluation of the quality of the anonymized data. Besides, ARX is also made available as a Java library, thus the tools can be embedded in other projects, as well. Accordingly, ARX is considered to be more robust and to offer better usability in general[43].

Combining well-tested anonymization models, like $k$-anonymity and differential privacy, Aircloak has come up with a unique solution. An anonymization layer is put on top of the target dataset through which each query must pass through. Similarly to differential privacy, Aircloaks works with adding noise to the query results. There is one major disadvantage of differential privacy: it comes with a privacy budget that is depleted after a certain amount of queries. From that point on, answering any additional query would mean that a potential attacker could break the privacy guarantees through statistical methods. Thus, as the privacy budget is spent, the underlying database becomes unusable. In contrast, Aircloak has managed to engineer a specific way of noise addition, thanks to which the noise added to the results does not cancel out, the way it does in differential privacy. This removes the limitation on the number of available queries, while maintaining the privacy guarantees.

#### Shortcomings

Unfortunately, it is not trivial how Würth Phoenix could offer an anonymization service to its customers based on these external tools. The primary issue is scalability. For example, to use the two open-source solutions, data would first need to be extracted in a CSV format and then ingested into the anonymization tool. They do not offer a connector to Elasticsearch. In addition, they have limitations in terms of the dataset size they can handle[34].

When it comes to Aircloak, deploying their solution is rather complex and time-consuming. For Würth Phoenix customers, a more light-weight anonymization solution could easily be sufficient. Another concern might be that partnering with an external company would make Würth Phoenix completely reliant on a third-party service provider.

## 3.5 The problem statement

Given the various items of the taxonomy require specific, dedicated solutions, it only makes sense to pick one and focus the attention entirely on that. As discussions with some colleagues at Würth Phoenix reveal, the most relevant use cases for the company are from the domain of static, one-time, server-side anonymization. Let's take the example of data retention again. Whether deploying NetEye with a new Würth Phoenix customer or ingesting a new data source for an existing one, data retention, as a fundamental component of data governance, is an unavoidable topic of discussion. The most trivial way to include anonymization in the service portfolio of the company is to replace the deletion of data with anonymization thereof in the data retention policy.

Accordingly, this item of the taxonomy can bring the most value to Würth Phoenix. Besides, the static, one-time, server-side anonymization is also the most clear and straightforward way of anonymization. It is the easiest to grasp. Once a solution exists for the basic problem, the entire domain, in this case that of anonymization, becomes more clear. New problems, ideas and potential use cases can emerge in the process. From that point on, it is thus easier to venture into the other directions.

From the point of view of Würth Phoenix, the currently existing anonymization tools have the three main downsides: inflexibility, lack of scalability, and the need to involve a third party. Firstly, inflexibility refers to the limited set of supported data sources. Secondly, available solutions do not scale well as the amount of data to be anonymized increases. Inside NetEye it is the anonymization of logs that is of interest. Taking into account the speed at which logs are generated, and the resulting size of the datasets to be anonymized, scalability is crucial. Thirdly, partnering with and outsourcing anonymization to a third party, although a common practice, would necessitate additional compliance measures for the period of the data processing. When the ultimate goal of introducing anonymization into the service portfolio of the company is precisely alleviating all the GDPR compliance-related efforts, introducing another layer of legal obligations might take a toll on the usability, relevance and added value of the whole endeavour. Therefore, beside striving for a flexible and scalable solution, providing it inside the NetEye ecosystem would be of great value.

The thesis project aims to find a solution that addresses all of the requirements derived from the analysis of the use cases on the one hand and the currently existing solutions on the other one: a flexible, scalable, static, server-side anonymization solution integrated into the NetEye ecosystem. A proof of concept implementation of the presented solution will be provided to support and prove the practical feasibility of the theoretical proposal. Such an implementation helps to discover and address flawed assumptions. In addition, it highlights the practical challenges and limitations of the proposal.

# Chapter 4

# Solution

By addressing the scope of the problem, as laid out at the end of the last chapter, an anonymization solution that brings value to Würth Phoenix should have four main characteristics. Firstly, it should be able to handle large quantities of data and provide scalability in this regard. Secondly, it should be able to work with Elasticsearch indices so that there is no need to export every dataset that a customer decides to anonymize. Thirdly, it should be integrable into NetEye and should not introduce excessive complexity into the currently existing workflows. And finally, referring back to the taxonomy proposed in the previous chapter, the solution should focus on static, one-time anonymization on the server-side and address the use cases relevant for the company.

## 4.1 The first approach

The database technology to use is rather straightforward, given that the company relies on Elasticsearch for the collection of logs. It is a valid assumption that the data source will be stored in Elasticsearch and that the anonymized information should be ingested into another index of the same database. The one decision to make is which of the two industry-standard anonymity models to rely on: $k$-anonymity or differential privacy.

The two models take a fundamentally different approach, thus it makes sense to choose one and focus the proposed solution entirely on that. The model of $k$-anonymity provides syntactic privacy[28]. "The syntactic mechanism revamps the dataset before release to link any tuple/record to more than one sensitive value in the dataset". In case of $k$-anonymity, this is achieved by placing each record in an equivalence class of size $k$. Syntactic privacy guarantees are in general considered weaker than those provided by semantic mechanisms. "The semantic mechanism restricts the impact of individual values on the query/analysis output based on the dataset". Differential privacy applies noise addition to prevent attackers from spotting the deletion or insertion of a record from the dataset.

When making a decision between the two approaches, the well-known trade-off in anonymization must be faced: does the use case at hand attribute more importance to privacy or to utility? Applying differential privacy as an anonymization measure yields a dataset of lower utility due to its manipulating the original attribute values[9]. However, it hides the single individuals better, and thus provides more privacy compared to $k$-anonymity. So, what is the most reasonable trade-off in case of Würth Phoenix and its customers?

There are two main reasons for choosing the $k$-anonymity model with the use cases in mind that the discussions with the colleagues revealed as most realistic and relevant. Firstly, the anonymized datasets are primarily meant for internal use, or sharing it with trusted partner companies. Usage data that has been anonymized instead of deletion for analytical purposes will be processed by the analysts of the company. Sharing an anonymized dataset with an external company also happens with the intention of extracting additional analytical insights. In neither of the cases will the data be available to a wider public. Thus, the risk of an attacker launching a sophisticated de-identification attack against such an anonymous dataset seems rather low. At the same time, data utility is of huge importance for the analytical processes. The second, and even stronger argument for $k$-anonymity is that once the algorithm produces the anonymous dataset, the original one can be completely removed. However, differential privacy requires the retention of the original dataset: it works by carrying out the user query on the original data and modifying the results through noise addition in order to render it anonymous. Considering the use case of retaining anonymous data after reaching the limit of the allowed time frame for storing personal information, the data controller is legally obliged to delete any personal data. Thus, differential privacy is not even an option for some of the use cases.

Once the decision for $k$-anonymity has been made, the only thing left to do is choosing an algorithm that is capable of producing anonymous data of high utility. Mondrian is one of the most common and most referenced algorithms, but there are a series of more modern ones that might offer better performance in terms of privacy, utility and runtime.

## 4.2 A revised solution

The first approach does address the requirement of supporting the anonymization of Elasticsearch indices, and by directly focusing on the Elastic Stack would probably be easily integrated into NetEye, as well. Through the right choice in terms of the algorithm, also scalability can be provided. However, such a solution ends up with the same shortcoming that the other, currently existing anonymization tools are criticised for: inflexibility. By focusing on one specific technology and algorithm, the proposed solution would not be ready to adapt to changing circumstances in technology and follow the new directions in the industry or the strategy of the company, for example. Accordingly, the updated objective becomes to provide a flexible architecture for an anonymization module that allows the replaceability of both the database technology and the algorithm. This is achieved by dividing the architecture of the module into an algorithm- and a database-specific component. They should only communicate through abstract interfaces using standard message formats when exchanging information.

### 4.2.1 $k$-anonymity

It is possible to support multiple anonymity models, as demonstrated by the ARX data anonymization tool. However, proposing an architecture that works with an abstraction of the anonymity model seems overly ambitious, especially considering how differently for example $k$-anonymity and differential privacy approach anonymization. Moreover, this would considerably increase the complexity of the proof of concept implementation. Narrowing the scope of the proposal to the model of $k$-anonymity helps to come up with an architecture that is abstract enough to provide flexibility, but still remains low-level enough to be put to practice.

The model is an industry-standard that the biggest companies, like Google, rely upon when anonymizing personal data. This clearly demonstrates the current relevance of the model. $k$-anonymity is an active field of research and the largest players in the technology industry relying on it suggests that the research efforts around the model will continue in the future with high probability. Since achieving optimal $k$-anonymity is an NP-hard problem, researchers are constantly looking for new ways to approximate optimal $k$-anonymization leveraging more and more sophisticated techniques. To be able to make use of the latest results of research, it should be easy and straightforward to plug in new, more performant algorithms into the anonymization module. This replaceability would provide flexibility in terms of the anonymization logic.

Building a solution around one specific model allows for the extraction of standardised elements. Let's recap what are the steps of producing a $k$-anonymous dataset to be able to point out the algorithm-agnostic parts of the process. It all starts with an analysis of the dataset to be anonymized. As presented in Chapter 2, $k$-anonymity differentiates between three major categories of attributes. Direct identifiers cover any information that allows the direct singling out of individuals. These pieces of information are removed completely from the dataset at the very beginning. Sensitive attributes contain the information that is valuable for the analysts, from which they intend to extract some sort of insights. Accordingly, such attribute values are retained in their original form. Note that sensitive attributes are only necessary for the second phase of the anonymization, when the anonymous dataset is created. The first step focuses entirely on indirect identifiers. It is usually not possible to identify individuals based on one single indirect identifier, but once more and more pieces of such information is revealed about the data subject, their combination can already single out the person behind the data. Equivalence classes, i.e. the "buckets" of size k, are created by generalising the values of these attributes.

So what does a $k$-anonymization algorithm do from a bird's-eye view? As an input, it expects three pieces of information. The size of the equivalence classes, that is the definition of the parameter k. The dataset based on which the equivalence classes should be created. And the categorization of the attributes present in the dataset. Referring back to Chapter 2 again, it is also essential to further split the indirect attributes into categories based on the data type they contain. An indirect attribute can be numerical, categorical or hierarchical, for example. Accordingly, the input of the algorithm should contain any additional information, like the generalisation hierarchy trees of hierarchical attributes. Finally, an algorithm might also need some further parameterization per indirect identifier. Once each piece of the above-mentioned input information is provided, the algorithm is ready to run and construct the equivalence classes, placing the individual records into $k$-sized buckets. The quality of the resulting anonymous dataset highly depends on how sophisticated the anonymization logic of the algorithm is.

#### Standardised elements

All this input information is easily summarised in a configuration file, which can then be passed to the $k$-anonymity algorithm as one single input. The simplest format to define a configuration file is JSON: using key-value pairs, the input parameters are easily described. The parameter k is, of course, a simple number. it is not necessary to explicitly list the entire set of direct identifiers. By not mentioning them, it is an implicit implication that attributes should be entirely suppressed. Since there can be multiple sensitive attributes in

the dataset, these can be listed in a string array. At this point it is only the indirect identifiers that need to be described.

Indirect attributes can be mapped into a set of key-value pairs, with the key being the name of the attribute and the value representing all the information about it necessary for the anonymization. One such mandatory piece of information is the type of the attribute. Then, depending on the attribute type, additional data might be required. For example, the handling of hierarchical attributes necessitates the definition of generalisation hierarchy. Generalisation hierarchies are trees, which are also easily mapped into a recursive JSON structure. Each node in a tree has a value, and a set of children. Accordingly, each JSON object representing a tree node has these two properties, the value being a string and the children stored in an array of objects with the same structure.

Listing 4.1 shows what such a configuration file could look like. It is taken directly from the proof of concept implementation presented in the next chapter of the thesis. The configuration file shows one way of mapping the generalisation hierarchy of marital status, presented in Figure 2.2 of Chapter 2, into a JSON format. In addition to the structure described above, this configuration file also demonstrates how algorithm specific information can be attached to the indirect identifier attributes. In this case, some additional information is added for the Datafly algorithm, the exact semantics of which are presented in the next chapter.

```
{
    "k": 10,
    "sensitive_attributes": ["class"],
    "indirect_identifiers": {
        "age": {
            "type": "numerical",
            "datafly_num_of_buckets": 20
        },
        "marital_status": {
            "type": "hierarchical",
            "datafly_init_level": 2,
            "tree": {
                "value": "*",
                "children": [
                    # ...
                    {
                        "value": "Married",
                        "children": [
                            {
                                "value": "Married-civ-spouse",
                                "children": []
                            },
                            {
                                "value": "Married-AF-spouse",
                                "children": []
                            }
                        ]
                    },
                    # ...
                ]
            }
        }
    }
}
```

Listing 4.1: A snippet from the configuration file for the UCI adult dataset

Based on this configuration file, the algorithm can already carry out the anonymization and produce a series of equivalence classes. These serve as the output of the procedure. Since each $k$-anonymity algorithm eventually creates equivalence classes, the format of the output can also be standardised. The generalised form of the data typically varies per attribute type. Once it is clearly defined which type of data should be transformed to which format, each algorithm can generate output of the exact same standard. Thus, the receiver of the equivalence classes becomes completely independent of the algorithm producing them.

There are, of course, various ways to describe equivalence classes. Numerical attributes generalised as ranges will contain the minimum and the maximum of the corresponding range, separated by a comma or a hyphen. When describing a hierarchical attribute in an equivalence class, the most straightforward approach is to map it into the generalised value of the internal node from the generalisation tree. Alternatively, instead of returning the generalised value of internal nodes, the hierarchical attribute can also be mapped to the leaf values of internal nodes. An example for the potential description of equivalence classes in JSON format is presented in Listing 4.2.

```
[
    {
        "age": "20,30",
        "marital_status": "Married-civ-spouse"
    },
    {
      "age": "45",
      "marital_status": "Married"
    }
]
```

Listing 4.2: A potential description of equivalence classes in JSON format

The configuration file and the equivalence class can thus become two standardised descriptors, independent of the algorithms used. This makes it possible to treat algorithms as black boxes and easily replace one with another, an important step towards the replaceability of the architectural components.

### 4.2.2 The algorithm-specific component

For the functioning of the algorithm-specific component, two pieces of input are required. On the one hand, the algorithm requires the standardised configuration file that defines the general properties essential for working with the $k$-anonymity model along with any further algorithm-specific settings. On the other hand, the component needs the data that it should carry out the anonymization on. Once the algorithm has received all of the information necessary for its initialisation, it can be launched. There are no further interactions required from outside. Thus, the only two methods that the algorithm-specific component needs to make available to the external world is one for its initialization and another one for starting the anonymization process.

With a standard already defined, constructing the configuration file and sharing it with the component is straightforward. However, how should the component get access to the data to work with? The currently existing anonymization tools and libraries usually expect the data to be extracted from their host database in CSV format that the anonymization solution is then capable of understanding and processing. This is how these tools achieve independence of any database technology. The scalability of this approach is questionable. As the number of the data sources and the volume of data increases, the continuous extraction of CSV files and reading them into a program can become inconvenient and time-consuming. Furthermore, the operations that are now carried out from programme code are most probably accessible natively in the database technology itself, implemented in a more efficient and performant way.

To leverage these native database operations and avoid the CSV extraction step, the component follows a query-based approach: every information necessary for the anonymization is obtained by directly querying the data in the database. However, the component should not rely on any one specific database technology. A layer of abstraction must therefore be introduced here. The algorithm-specific component should communicate with an API only. This API must implement all the necessary queries that the algorithm would intend to make to the database.

It is important to note that there are a set of queries that each $k$-anonymity algorithm most probably relies on. Such queries are, for example, obtaining the count of records belonging to an equivalence class, or getting the minimum and maximum value of a subset of the dataset. All these queries can be extracted into a general API. An operation that allows the algorithm to push the generated equivalence class to the database backend should also be part of this general API. Then, each of the different $k$-anonymity algorithms might have a specific set of queries that they need for the anonymization, and accordingly must define their own APIs. The definition of the algorithm-specific API can serve as a contract between the algorithm and the backend. From this point on, instead of working with raw data, the algorithm communicates with an API that is capable of responding to the set of predefined queries. As long as the API returns the expected information, the algorithm can treat it as a black box and completely disregard the database technology serving the requests sent.

During the anonymization process, it is often necessary to formulate some queries about a certain equivalence class. For these queries, the component can rely on the same, standardised equivalence class format that is used for sending the output of the algorithm to the backend. Figure 4.1 presents the architecture of the algorithm-specific component. The algorithm takes a standard configuration file and outputs equivalence classes in a standardised format. The server implements the "universal API" which covers the common queries across the various $k$-anonymity algorithms along with the operation that allows the sending of the generated equivalence classes to the server. If the backend wants to support one specific algorithm, it also implements the algorithm-specific API. Relying on the abstraction of these two interfaces, the algorithm-specific component can treat the backend, which manages the data to anonymize, as a black box.

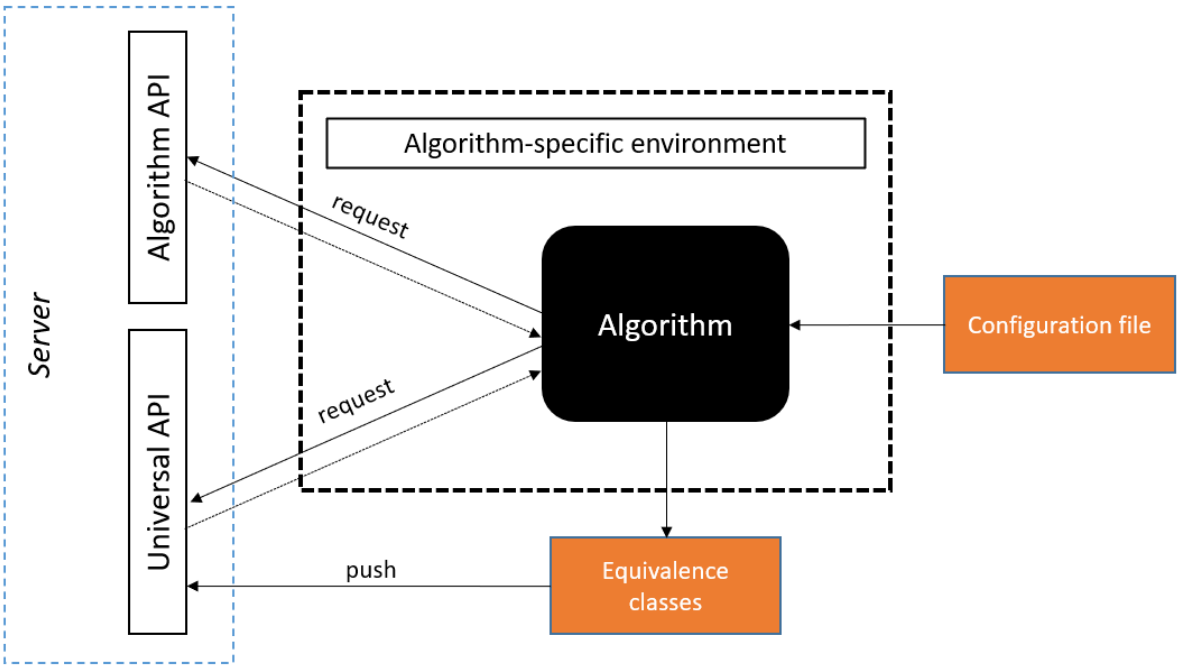


Figure 4.1: The architecture of the algorithm-specific component

### 4.2.3 The database-specific component

There are two main parts that the database-specific component is made up of: the database and the backend. The database stores the dataset to anonymize and provides a set of functionalities that makes it possible to effectively query and work with the data. The backend is the point of entry to the database. It is the backend that makes the API available for any client that intends to interact with the database.

The backend starts with implementing the algorithm-agnostic functionalities: the mapping of the equivalence class descriptor to database queries, the functions like the minimum and maximum, the count operation, and the final digestion of the equivalence classes for producing the anonymous dataset. It then goes on to implement the contracts that have been defined by the various algorithms. Once the backend working with a specific database technology decides to support a specific type of algorithm, it needs to implement the list of operations that are necessary for the functioning of the algorithm.

Not exporting the data, but addressing the queries directly to the database can increase the performance of the anonymization compared to working with an exported CSV file. The backend can make sure to implement the queries in the most efficient way, while the database engine itself performs further optimizations before eventually executing the query.

Figure 4.2 presents the architecture of the database-specific component. The backend implements and makes a set of operations available for external clients. Other than the specification of the methods they need, the backend does not need to know anything about the entities using its public interface. The component receives requests, transforms them into queries, the database answers these queries. Finally, the backend responds to the clients with the query results.

## 4.3 The final architecture

Figure 4.3 presents how the two components eventually interact relying only on the introduced abstractions. For each new algorithm that the backend intends to support, it must implement the operations as defined and expected by the algorithm. Once the algorithm knows where to find the backend, it can call the endpoints that return the relevant information for carrying out the anonymization.

To make the solution technology-agnostic, the backend should make its functionality available through a web API. The component implementing the algorithm should communicate with the backend only via this web interface. The communication thus becomes standardised and also remains independent of one specific technology. From this point on, the algorithm- and the database-specific components can be separately implemented in the preferred languages and frameworks of the development team.

Both components are aware of the standardised JSON format of describing equivalence classes, as both of them must work with them. Feeding in the configuration file to the backend, however, is not necessarily a must,

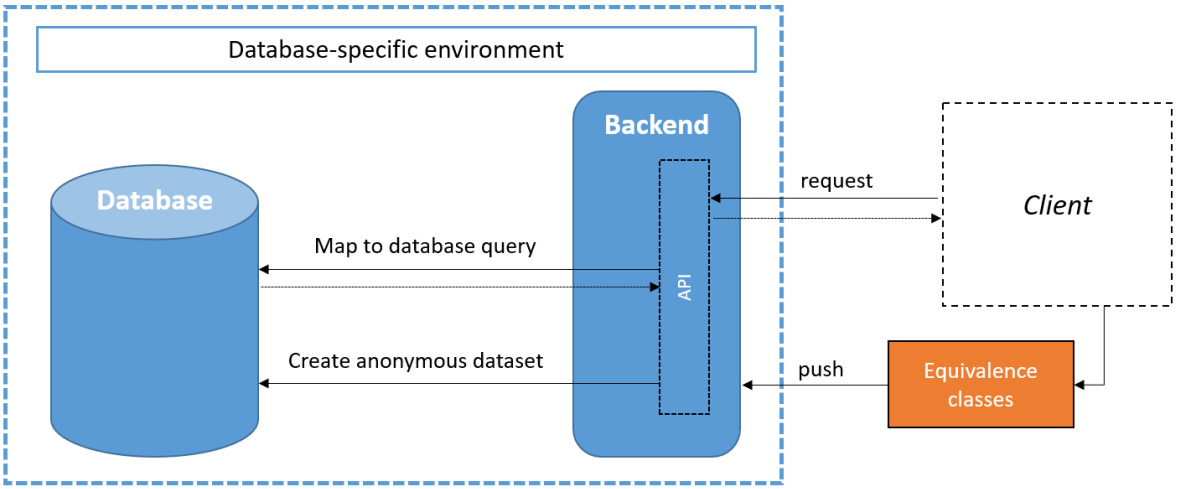


Figure 4.2: The architecture of the database-specific component

if the implementation of the algorithms make sure to send data in a format that the backend can directly use for querying. Let's take generalisation hierarchies as an example. If the equivalence class contains values of internal nodes from the generalisation hierarchies for some of the attributes, and the backend does not know the generalisation hierarchy, it is not able to map these generalised values to the actual ones that are present in the database. Without the configuration file, the backend also does not know which attributes are sensitive ones that should be extracted and combined with the equivalence classes when producing the final, anonymous dataset. Nevertheless, this introduces additional complexity for the algorithms, and necessitates the duplication of the same operations across algorithms. Thus, it is best to share the configuration file with the backend directly, as well.

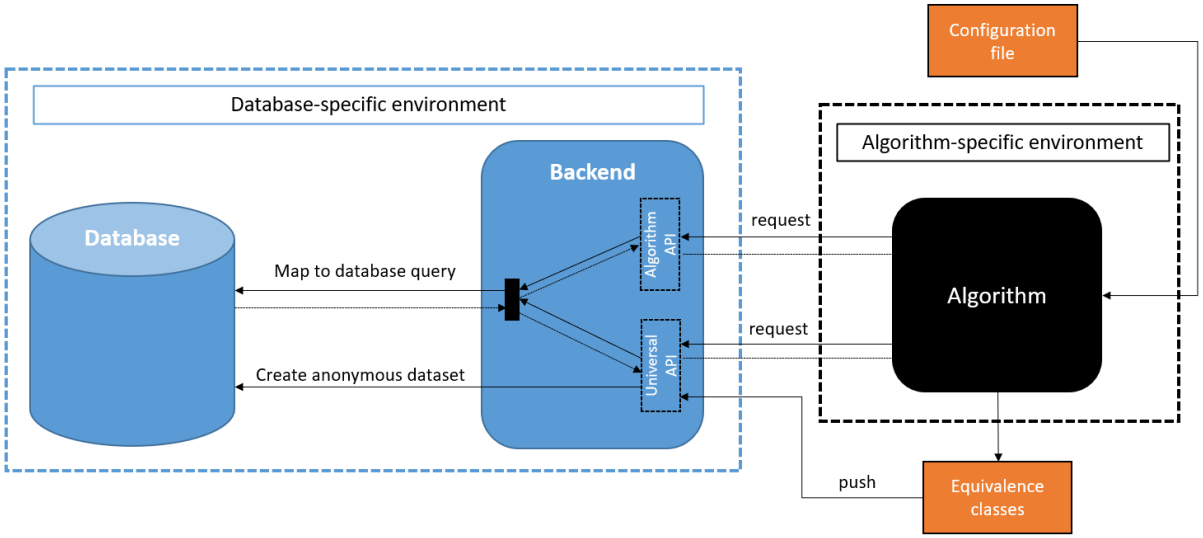


Figure 4.3: The architecture showing the interactions between the two components

## 4.4 Integration into NetEye

The ultimate goal is to integrate the module into the NetEye ecosystem. Thanks to the flexibility and modular architecture of the Icinga user interface, a new item is easily added to the menu of NetEye. Similarly to other optional feature modules, like Log Management and Command Orchestrator, the anonymization module would only be installed on demand.

On setting up the anonymization of a data source, the user needs to define the configuration file and specify the dataset to anonymize. Accordingly, the page of the module should provide a simple user interface for the

easy management of these two options. As a matter of fact, only the definition of the generalisation hierarchies would require a more elaborate design.

Once all the parameters have been specified, the user should be able to either launch the anonymization right away, or schedule it as a recurring task.

# Chapter 5

# Proof of concept

Once the high-level architectural design, as a proposed solution, has been completed, the next step of the thesis project was to provide a proof-of-concept implementation of the anonymization module[35]. A proof of concept is a sort of prototype, the main purpose of which is to demonstrate that the solution introduced on a theoretical level can be realised and has practical value. The ultimate goal of this proof-of-concept implementation of the anonymization module is twofold. On the one hand, it is meant to prove that the proposed architecture provides the promised flexibility. It must demonstrate that the abstractions work, and through them the database- and algorithm-specific components are indeed easily replaceable. On the other hand, by implementing anonymization for an Elasticsearch backend, it presents to the company how anonymization could be integrated into their ecosystem.

The proof of concept is implemented in Python. The language has a number of advantages that make it a particularly good tool for developing a proof of concept. The flexibility of Python, for example, helps with quick prototyping and trying out various ideas. Moreover, there are several anonymization libraries implemented in the language, which were of help when implementing the two $k$-anonymization algorithms. Also, a wide set of easy-to-use database connectors are written for Python.

To reduce the complexity of the implementation, instead of using web interfaces as proposed in the previous chapter, the abstractions are modelled through abstract classes. The proof of concept is constructed in the form of a single Python project. Thus, the technology-agnostic property is not fulfilled. However, the web API is simulated by making the algorithms communicate through interfaces only. This makes it possible to demonstrate the replaceability of the components. Considering that the implementation is meant to be a proof of concept, this simplification is acceptable.

## 5.1 Overview of the implementation

When it comes to the database-specific components, the proof of concept comes with support for two technologies, namely Elasticsearch and MySQL. Given that Würth Phoenix provides its log management services based on the Elastic Stack, currently the Elaticsearch backend is of the greatest importance and interest for the company. It serves as a hands-on demonstration of what an anonymous Elasticsearch index looks like, and can thus contribute to the first discussions about how to best make use of them for data analytics. The rationale behind choosing MySQL is on the one hand its widespread use, and on the other hand the popularity of SQL databases in general. It might be valuable for a number of stakeholders to see how the anonymization module can work using this query language.

There are two algorithms included in the proof-of-concept implementation: Mondrian and Datafly. Mondrian is a frequently-referenced, top-down, greedy $k$-anonymization algorithm. It is capable of producing a good approximation of an optimal $k$-anonymous dataset. Datafly is one of the early $k$-anonymity algorithms. It follows a bottom-up approach, which provides a good contrast to the top-down Mondrian. The implementation relies on a simplified version of the algorithm[41]. Chapter 2 contains a more detailed description of how these two algorithms work.

Configuration files are provided for two datasets. The adult dataset is a subset of the US Census Data from 1994. It is frequently used for trying out new anonymization algorithms or for comparing existing ones. It has attributes of the most common types, namely numerical, categorical and hierarchical ones. The other dataset, shipped by default with Kibana, contains artificially created web requests. Other than the most basic data types, it also has fields containing timestamps and IPs. Given that attributes of these types are often found in machine logs, the Kibana web logs come in handy for demonstrating how the anonymization of such data could be carried out.

Figure 5.1 gives an overview of the parts of the proof-of-concept implementation, using the building blocks,

namely the database- and algorithm-specific components, presented in the previous chapter. The database-specific component is implemented for Elasticsearch and MySQL. They each come with a corresponding database connector, i.e. a backend that initiates the queries and communicates with the clients. The proof of concept includes an implementation for the Mondrian and Datafly algorithms. It is important to note that while the adult dataset is available in both databases, the web logs of Kibana are only reachable in Elasticsearch.

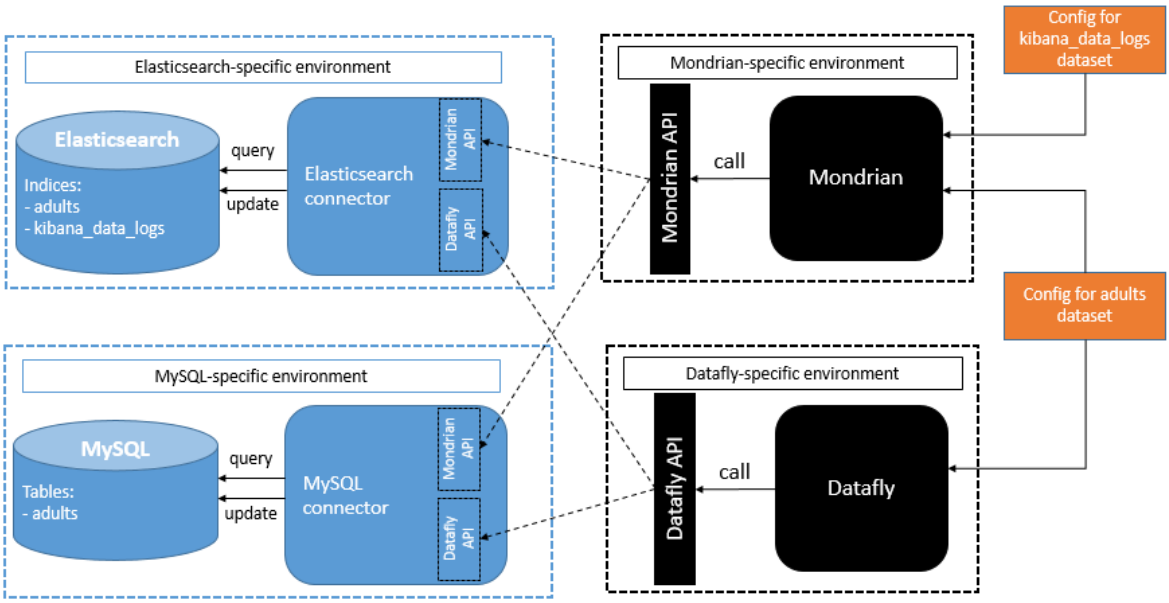


Figure 5.1: Overview of the proof-of-concept implementation with the database- and algorithm-specific components

## 5.2 The timeline

The implementation of the proof of concept started with the getting to know and refactoring of a publicly available Mondrian repository. From its original approach of carrying out the anonymization directly on the data read into memory, the implementation of the algorithm was redesigned to work with database queries. For this, a Mondrian-specific API had been extracted for each operation that was necessary for running the algorithm. The next step was to set up the connector for Elasticsearch and create the queries based on the Mondrian API. Pivoting from working on in-memory data to a query-based approach required the tweaking and adapting of the implementation of the original repository.

With Mondrian using an Elasticsearch backend already capable of anonymization, the next milestone was to come up with an implementation for Datafly. The bottom-up way of data anonymization using database queries required a fundamentally different approach compared to the Datafly implementations available in public repositories. Therefore, the algorithm had to be developed from scratch. The Datafly API could then be extracted, and the corresponding queries were implemented in the Elasticsearch backend. Relying on the experience with the Python library of Elasticsearch, gained during the realisation of the universal and the Mondrian APIs, the development of the Datafly-related operations on the backend went rather smoothly.

The following step was to implement a backend for MySQL with support for both of the algorithms. The change of the query language, of course, required a different approach for implementing the two API definitions. The abstractions worked well: once a solution had been found for answering all of the required queries through SQL, the Elasticsearch backend could indeed be seamlessly replaced in one step with the new MySQL backend, without having to change anything on the side of the algorithms.

Up until this point, the proof of concept focused entirely on the adults dataset. The next objective was to create a new configuration file for the Kibana sample logs and try out the anonymization on another dataset. On the one hand, it shows how the currently supported data types can be extended with new ones. On the other hand, a potential way for the anonymization of timestamps and IP addresses, both recurring fields in machine logs, can be demonstrated with the help of this sample dataset.

## 5.3 Datasets

The proof of concept shows how the anonymization module works using two datasets. The UCI adult income dataset[8] is a subset of the 1994 US Census data that contains some basic demographic information about individuals, along with an indication whether their salary is above or below $50,000 per year. It is often used for the demonstration of how various anonymization algorithms work. The sample web logs included in Kibana of the Elastic Stack presents a series of artificially generated web requests. The fact that it contains timestamps and IP addresses makes it valuable for this proof-of-concept implementation, for it allows to showcase how these two further data types can be rendered anonymous.

### 5.3.1 UCI adult income

The UCI adult dataset, also referenced as Census Income dataset, was extracted from the US Census data of 1994. It was originally meant for training and evaluating machine learning models. The task was binary classification: predict whether a person makes above or below $50,000 yearly based on the 14 available attributes. As a side note it is worth mentioning that the entries in the dataset do not cover specific individuals, but they cover groups of people who are described by the given attributes values. The number of people corresponding to one entry is represented by the final weight (fnlwgt) attributes. When using the data for anonymization, however, the entries are assumed to belong to single individuals.

Among others, the dataset contains attributes like age, level of education, marital status, race and sex, among others. These count as personal information, more precisely they are indirect identifiers. Therefore, besides building machine learning models, this dataset is also a good candidate for testing various anonymization algorithms. The salary attribute, the target variable to predict in the classification exercises, is regarded as a sensitive attribute when anonymizing the dataset.

Typically, only a subset of the 14 attributes are used in anonymization tasks, and this proof of concept follows the same practice. In total eight attributes are taken into account when creating the equivalence classes. Two of them are continuous, numerical attributes, namely the age and educational number fields. The educational number is the equivalent of the education attribute, mapped into a numerical form. The education attribute is a categorical one, however it has a natural ordering: from primary school until a doctorate degree, the levels of education follow each other in clear succession. Categorical attributes with a natural ordering can be mapped into numerical values, and from that point on they can be treated as numerical attributes during the anonymization procedure. This comes in handy for demonstrating the anonymization with multiple numerical attributes, otherwise only the age could be used.

There are three categorical attributes: occupation, sex and race. And the last three attributes, namely marital status, native country and workclass, are hierarchical. The proof of concept takes an open repository with a Mondrian implementation as a starting point. This project treats each of the three above mentioned categorical attributes as hierarchical ones, with a two-level hierarchy tree. The root simply contains an asterisk, meaning that the value is completely suppressed, while the leaves store the original attribute values in the second level. This precisely reflects how categorical attributes, without any means for generalisation, should be treated: they are either kept in their original form or are suppressed completely. However, mapping them into either hierarchical or, in case of categorical attributes with a natural ordering, into numerical attributes makes the implementation simpler, since instead of three only two attribute types need to be handled and the end results remain the same. The proof-of-concept implementation follows the same approach, using the generalisation hierarchies defined in the open Mondrian repository.

The dataset comes in the format of a text file that contains each entry in a new line, with the attribute values separated by commas. After each comma, a space is included that causes problems during parsing. Besides, some entries have missing values, denoted with a question mark, for some of the attributes. For the sake of simplicity, these should be filtered out before launching the anonymization. Therefore, a simple preprocessing of the original file is required. The spaces must be removed, along with any empty lines and lines that contain a question mark. This can easily be done, for example, with the default `grep` and `sed` Linux programmes, or practically with any other word processor.

At this point, the dataset is ready to be ingested into a database and to be anonymized. Figure 5.2 shows a sample of the adults dataset with the attributes used during the anonymization.

### 5.3.2 Kibana web logs

The Kibana web logs is one of the three artificially generated datasets shipped with the tool. They can be ingested into an Elasticsearch index on demand with one click. Once in the database, the data can be explored through the KIbana UI with the help of premade visualisations and dashboards.The web logs contain general information about web requests made by some fictional clients. The fields of the Elasticsearch documents

| age | workclass | education | education number | marital status | occupation | relationship | race | sex | native country | class |
|---|---|---|---|---|---|---|---|---|---|---|
| 39 | State-gov | Bachelors | 13 | Never-married | Adm-clerical | Not-in-family | White | Male | United-States | <=50K |
| 50 | Self-emp-not-inc | Bachelors | 13 | Married-civ-spouse | Exec-managerial | Husband | White | Male | United-States | <=50K |
| 38 | Private | HS-grad | 9 | Divorced | Handlers-cleaners | Not-in-family | White | Male | United-States | <=50K |

Figure 5.2: A sample of the UCI adult dataset

present some details about the machine used, the client IP address, the user agent and geographical data about both the server and the client, among others. Figure 5.3 shows a document of the Kibana web logs.

| | |
|---|---|
| @timestamp | Apr 6, 2023 @ 22:49:29.440 |
| agent | Mozilla/5.0 (X11; Linux x86_64; rv:6.0a1) Gecko/20110421 Firefox/6.0a1 |
| bytes | 2,950 |
| clientip | 99.76.103.49 |
| geo.dest | TR |
| geo.src | US |
| geo.srcdest | US:TR |
| host | artifacts.elastic.co |
| index | kibana_sample_data_logs |
| ip | 99.76.103.49 |
| request | /elasticsearch/elasticsearch-6.3.2.deb |
| response | 200 |
| tags | success, info |
| timestamp | Apr 6, 2023 @ 22:49:29.440 |

Figure 5.3: A document of the Kibana web logs

The main reason for choosing this dataset for anonymization is the presence of timestamps and IP addresses. These are attribute types the anonymization of which could not be demonstrated on the adults dataset. Given that timestamps and IPs are frequently recurring fields in logs and that the company is primarily interested in log anonymization, demonstrating the anonymization of these two data types in the proof of concept is essential. For this reason, the *clientip* and the *timestamp* fields are chosen to be anonymized. Other than these two, the *bytes* field, as a numerical attribute, and the *geo.dest* field, as a hierarchical one, are also included in the anonymization. The former represents the size of the response, while the latter contains a two-letter country code that describes where the server is located. In this dataset, two attributes are considered sensitive: the host and request fields, describing the domain and the path of the requested URL. Semantically, the anonymization of these logs does not make much sense, but it serves well the demonstration of how the two new data types can be rendered anonymous.

For the *geo.dest* field, containing two-letter ISO country codes, a four-level generalisation hierarchy is constructed. Following the asterisk, i.e. the completely suppressed value, in the root node, the internal ones contain continents, their regions, and finally the two-letter country codes stored in the leaf nodes[22]. Listing 5.1 shows a segment of the description of the generalisation hierarchy from the configuration file of the Kibana web logs. It is important to note that some country codes used in the web logs were not part of the standard ISO definition. The small set of documents containing such country codes are ignored during the anonymization.

```
{
    "value": "Europe",
    "children": [
        {
```

```
            "value": "Central Europe",
            "children": [
                {
                    "value": "AT",
                    "children": []
                },
                {
                    "value": "CH",
                    "children": []
                },
                ...
            ]
        },
        ...
    ]
}
```

Listing 5.1: A segment of the description of the generalisation hierarchy from the configuration file of the Kibana web logs

### 5.3.3 The configuration files

For each new dataset, a dedicated configuration file must be constructed. The configuration file contains the $k$ parameter for the minimal number of records each equivalence class must have, the list of sensitive attributes and information about each indirect attribute to be involved in the anonymization. Such information is the attribute type, for hierarchical attributes the generalisation hierarchy and any additional algorithm-specific properties. Listing 4.1 in Chapter 4 shows a segment of the configuration file for the adult dataset. Listing 5.2 below presents the layout of the configuration file for the Kibana web logs. A fragment of the generalisation hierarchy for the *geo.dest* field is included in the previous section. The configuration files in their entirety are to be found in the GitHub repository.

```
{

    "k": 10,
    "sensitive_attributes": ["host", "request"],
    "qids": {
        "timestamp": {
            "type": "timestamp"
        },
        "clientip": {
            "type": "ip"
        },
        "bytes": {
            "type": "numerical",
        },
        "geo.dest": {
            "type": "hierarchical",
            "tree": {
                "value": "*",
                "children": [
                {
                        "value": "Africa",
                        "children": [...]
          },
                ...
                ]
            }
        }
  }
}
```

Listing 5.2: The layout of the configuration file for the Kibana web logs

## 5.4 Mondrian

In the hope of speeding up the development, the proof of concept takes as a starting point one of the several Mondrian implementations readily available in open repositories. The Basic Mondrian implementation of Qiyuan Gong has been chosen[18]. As it is clarified in the description of the repository, Basic Mondrian is an extension

of the original Mondrian algorithm with support for categorical and hierarchical attributes besides the numerical ones.

The Mondrian algorithm is introduced in more detail in Chapter 2. Briefly, the top-down approach of the algorithm can be summarised as follows. The entire, original dataset is taken as the initial equivalence class, or as the Mondrian terminology calls it, the initial partition. This initial partition sets the most generalised value for each of the quasi-identifier attributes. For numerical attributes it is the range that covers all of the values in the dataset, while hierarchical ones take the value of the root node. Mondrian is a greedy algorithm. In every step, based on a predefined heuristic, it chooses an attribute. It assumes that splitting the current partition along this attribute creates more or less two subpartitions of equal size. The splitting of a numerical attribute means the splitting of the range, for example along the mean or the median value of the items in the partition. While these examples produce two subpartitions, there might be different methods for splitting a numerical range into multiple new ones. The splitting of a hierarchical attribute is done by moving one level down in the generalisation hierarchy tree. Depending on the number of child nodes, such a split results in two or more subpartitions.

Once the new subpartitions are successfully created, the algorithm recursively tries to split these further. Mondrian terminates once there are no more valid splits available. A split is valid, if the created subpartitions contain at least k elements. It is important to note that this description of the algorithm relies on the assumption made in the previous section that categorical attributes are mapped into hierarchical ones for the sake of simplicity.

### 5.4.1 The open Mondrian implementation

It is completely up to the implementation to decide on, firstly, what heuristic to use for picking the attribute to split along, and, secondly, on the methods for how exactly to split numerical ranges and the values of the generalisation hierarchy tree. This proof of concept relies entirely on the approach followed by the Basic Mondrian repository of Qiyuan Gong.

The heuristic for choosing the attribute to split uses the normalised width of the attributes. The width of a numerical attribute equals the size of the range it covers, i.e. the difference between its minimum and maximum. The normalised width is calculated by dividing the width of the current attribute value by the width of the attribute value in the initial partition, i.e. by the greatest width the attribute can have. The heuristic always picks the attribute with the greatest normalised width for splitting the current partition.

For example, in the adults dataset the age attribute of the initial partition covers the range from 17 to 93, while the domain of the educational number attribute is between 1 and 16. This yields a width of $93 - 17 = 76$ for the age attribute, and $16 - 1 = 15$ for the educational number. Let's assume a subpartition with a range of (24-30) for its age attribute, and its educational number already having been split to the specific value of 14. In this case, the age attribute of the partition has a normalised width of $(30 - 24)/76 = 0.079$, while the educational number yields $(14 - 14)/15 = 0$. In this case, the heuristic picks the age attribute as the next attribute to split.

The width of a hierarchical attribute is defined as the number of child nodes that belong to the node, which represents the current value of the attribute, in the generalisation hierarchy tree. Let's take the marital status attribute of the adult dataset as an example. Its generalisation hierarchy is presented in Figure 2.2 of Chapter 2. Seven leaf nodes belong to the root value, thus its width equals 7. Let's assume the subpartition, mentioned in the previous paragraph with the range of (24-30) for its age attribute, and 14 for the educational number, contains the internal node value "Married" in the marital status attribute. This "Married" node has two children, thus the width of the subpartition with regards to the marital status is 2. Furthermore, its normalised width is $2/7 = 0,28$. In this case, given that the normalised width of the marital status is greater than that of the age, the heuristic would pick the marital status as the next attribute to split.

Once the attribute has been chosen, it must be defined how to split a partition along this type of attribute. The Basic Mondrian implementation splits each numerical range into two. The split is carried out at the median value of the attribute inside the partition. The implementation allows the creation of overlapping partitions. For example, if one partition with a range of (54-55) for the age attribute has a median of 55, the split creates one partition with the (54-55) range, and another one with the value 55. This detail becomes important when rewriting the implementation to carry out the anonymization based solely on database queries.

Hierarchical attributes are split by stepping one level down in the generalisation hierarchy and creating new partitions using the child nodes of the node representing the current value of the attribute. The implementation only deems a split valid, if each of the created subpartitions have either at least k number of elements or exactly zero elements in which case the subpartition can be discarded. This means that overlapping values in terms of hierarchical attributes cannot be created. Overlapping subpartitions would mean that a dedicated subpartition is created for each child value that covers more than k elements, and the rest of the elements are kept in the original partition that is being split - provided, of course, that the number of such elements exceeds the k number. Let's take again the marital status as an example. On splitting a partition with the root value for this

attribute, it might happen that the child values "Married" and "Singled" can constitute their own subpartitions, but both the "Never married" and "Married left" cover only $k-1$ elements. In this case, the elements belonging to the latter two values could be grouped together under the root node value, and the splitting of the partition would be valid.

A discussion about overlapping attribute values is relevant, because allowing the creation and usage of overlapping attribute values gives more flexibility to the algorithm, and can thus contribute to the generation of an anonymous dataset with higher utility. At the same time, it can considerably increase the complexity of the implementation.

After the termination of the Mondrian algorithm, i.e. having created all of the equivalence classes, the implementation goes on to carry out the anonymization itself in a second step. The anonymized entry is constructed by concatenating the sensitive attribute values of the original entry with the attribute values of the equivalence class that the entry belongs to. The implementation iterates through each final partition, that is each equivalence class, takes the original entries covered by this equivalence class and executes the previously presented concatenating operation. The anonymized dataset along with its corresponding Global Certainty Penalty or GCP score[17] constitutes the output of this Mondrian implementation. GCP measures the information loss incurred during anonymization, thus the aim is to minimize this metric. Converted to percentage as done in this implementation, 0% means that no information has been lost, while 100% suggests that practically all of the attribute values have been suppressed and the anonymous dataset does not contain any information at all.

The implementation relies on two data structures for storing information about numerical and hierarchical attributes, and one for keeping track of the created partitions. These have become fundamental building blocks of the proof of concept, as well, therefore it is worth introducing them briefly. The `NumRange` class represents basic information about numerical attributes, like their range and width, in the original partition. The `GenTree` is a recursive data structure that is capable of describing a generalisation hierarchy tree. Each instance of the class represents one node in the tree. Other than the strictly necessary properties for a tree data structure, it stores additional information that facilitates the operation of splitting and generalisation. The `NumRange` and `GenTree` classes are essentially used for obtaining metadata about the various attributes while running the algorithm. Accordingly, instances of these two classes are filled up at the very beginning, when reading the dataset and constructing the initial partition. The original implementation does not use a configuration file. Instead, the generalisation hierarchies are described in text files. The format is the following: each line describes one path in the tree, going from the root to a leaf value, dividing the value of the nodes along the path with a semicolon. Thus, the descriptor files of the generalisation hierarchies contain one line for each leaf value. The `NumRange` instances are filled up after the reading through of the entire dataset and the extraction of the attribute-specific metadata.

Finally, the `Partition` class describes the partitions produced by the Mondrian splits. Considering that the final partitions generated by the algorithm correspond to the output equivalence classes, the `Partition` class is suitable to be used as the standardised descriptor for equivalence classes. Accordingly, this class is passed around during the communication between the database-specific backend and the algorithm implementation. In case of pivoting to the use of a web API, as proposed in the architecture of Chapter 4, this class can also be easily used after serialisation.

The original repository uses these classes for basic data storing. As the implementation of the proof of concept progressed, the structure of all three of them were adapted and changed. The attribute related information, scattered and stored as multiple dictionaries in the original repository, were extracted into a new, dedicated `Attribute` class. Some helper functions were added to the classes, like that of `GenTree`, for instance, for obtaining all of the nodes from a specific level in the generalisation hierarchy tree. And eventually some functionality from the original codebase was also extracted, mainly as dedicated functions into the `Attribute` class. Such are, for example, the functions for mapping an attribute to a condition in the desired database technology. This is discussed in more detail in Section 5.8.

The data to be anonymized is read from a text file in CSV format. It is read line-by-line and the cleaning of the data, like the removal of spaces and entries with missing values presented in Subsection 5.3.1, is managed entirely from code, with the help of if statements and string operation. Each entry, i.e. each line of the CSV file, is stored as a list of strings. The index of the string value in the list corresponds to the index of the attribute in the original file. Thus, the input dataset is eventually read into memory as a list of lists of strings. The original implementation works directly on this representation of the dataset in memory. While this approach is sufficient for demonstration purposes with small datasets, when it comes to data sizes of higher magnitudes, this type of in-memory anonymization faces scalability issues very soon. The reading of millions of rows of data into memory, even if done using powerful data handling libraries, like *pandas*, instead of simple lists, has its clear limitations.

### 5.4.2 Adapting the codebase

For a better readability of the code, the work started with refactoring the original codebase of the repository. This refactoring considerably contributed to developing a profound understanding of the implementation. This was fundamental, given that the original repository carries out in-memory anonymization from which the proof of concept needed to pivot to a database query-based approach.

The refactoring primarily focused on the renaming of variables, the splitting of large functions and the introduction of Python type hints into the code. Once a deep understanding of the implementation was established and the codebase was easier to navigate, the next step was the extraction of the Mondrian API, i.e. the construction of the contract that describes for future backend "contractors" precisely what operations the algorithm expects them to provide. The three aggregate queries, which are required for the extraction of the attribute specific metadata, and then eventually throughout the anonymization, include the calculation of the minimum, maximum and the count of the desired attribute. As a matter of fact, these three operations will constitute the so-called universal API that any $k$-anonymization algorithm most probably needs to function, and that every database backend must implement accordingly.

There is only one Mondrian-specific query that makes up the algorithm-specific API: obtaining the median and the next value after the median for the specified attribute. The heuristic, as introduced in Subection 5.4.1, assumes that the splitting along the median divides the current partition the most efficiently. The maximum of the first subpartition will be the median, while its minimum is that of the original partition. The value following the median determines what the minimum of the second subpartition should be, and the maximum of the original partition is assigned to the maximum of the second new subpartition.

The universal API can be mapped into an abstract Python class definition. On creating a new algorithm-specific API, this abstract API class can be taken as the parent to inherit from. Once the APIs are defined, the algorithm implementation can take the algorithm-specific API class as a constructor parameter. From that point on, the algorithm can legitimately expect that calling the "endpoints", i.e. the public functions of the object they obtain through their constructor, gives them the correct answers which they can trust and use for carrying out their further operations. Accordingly, the code blocks of the original implementation carrying out these operations, like the calculation of the attribute minimum, maximum and median, can be replaced with calls to the algorithm-specific API.

The abstract parent class, i.e. the universal API, also contains a function for forwarding the generated, final equivalence classes to the database backend. The production of the anonymized dataset is thus delegated to the database-specific component. Consequently, the logic carrying out this step in the code can also be removed and substituted with a simple call to the API object received in the algorithm constructor.

These steps significantly simplified the codebase, and also allowed the further refactoring of the used data structures. The next milestone was the implementation of the Elasticsearch backend, as presented in Section 5.6. The challenges and bugs in the Mondrian implementation explained in the following subsection were only discovered after running the algorithm with the Elasticsearch backend. For the sake of cohesion, however, their discussion is included in this section.

### 5.4.3 Challenges encountered

During the first runs of the query-based proof-of-concept implementation of Mondrian, the execution was stopped due to a "maximum recursion depth exceeded" error. As it turned out, the code got stuck when trying to split the partition with a numerical range of (18-19): it kept creating one subpartition with the same numerical range, i.e. (18-19), and another one with a numerical value of 19. Then, it recursively calls the anonymization function on the first subpartition. However, this was the exact same partition that was originally split. The numerical attribute range was still (18-19), and, of course, the other attributes were not modified at all. As presented in Subsection 5.4.1, the original Mondrian implementation allows the creation of overlapping numerical ranges. And this is precisely what happened here, as well. The code produced the subpartitions the way it is supposed to. So what went wrong here?

Once the cause of the infinite recursion was found, it was not so difficult to understand how this approach was problematic in the query-based approach. If the splitting along the (18-19) numerical range produces a subpartition that looks exactly the same as the original one, the query generated from this partition always covers the same set of records, as shown in Figure 5.4. Thus, despite theoretically having executed a split, the algorithm cannot divide the partition any further.

In contrast, the in-memory anonymization approach keeps track of the records belonging to the various partitions through direct references. After the split, the entries covered by the subpartitions are not obtained through queries: when the split is being carried out, the entries are explicitly placed into the new subpartitions using their references, as Figure 5.5 demonstrates.

The problem of infinite recursion only occurs when the splitting of the partition results in a subpartition of width 0, i.e. when the attribute contains a concrete value instead of a range. As long as splitting results in

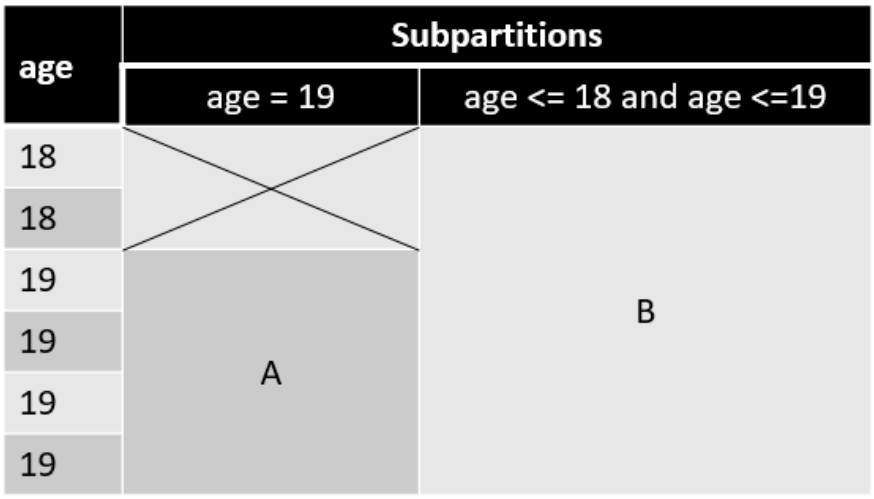


Figure 5.4: The result of querying overlapping ranges

| age | Subpartitions |
|---|---|
| 18 | A |
| 18 | |
| 19 | |
| 19 | B |
| 19 | |
| 19 | |

Figure 5.5: When creating partitions with overlapping ranges, explicitly defining which record goes to which partition

subpartitions with ranges of greater width, the algorithm seems to progress correctly. However, the functioning of the algorithm is only seemingly correct. Splitting the range of, for example, 30-40 into two subpartitions with ranges 30-35 and 35-40, the two subpartitions are overlapping. When formulating a query for these two ranges, records with a value of 35 for the given attribute will be included in both of the subpartitions, which corrupts the functionality of the algorithm already early on.

Undoubtedly, there would be ways to come up with some workaround that allows a query-based Mondrian to support overlapping equivalence classes. For example, by initialising each partition with an additional identifier number, the records owned by a partition could be flagged by its ID. Updating the records this way would, however, come with a significant amount of additional database operations, considerably slowing down the running of the algorithm. There might be other smarter, optimised workarounds that inflict less overhead on Mondrian. However, considering that the implementation provided in this thesis project is meant purely as a proof of concept, the above described problem is addressed through the simplification of avoiding overlapping ranges altogether, instead of crafting a sophisticated workaround. This change solved the issue of the ambiguous query results.

By preventing the generation of overlapping ranges, the recursion depth was not exceeded any more, the algorithm terminated and produced the desired equivalence classes. The simplification, however, led to a reduction in the quality of the produced anonymous dataset: its GCP score increased, i.e. more information was lost during the anonymization procedure compared to the results of the original implementation. This did not come as a surprise, such a compromise naturally results in a decrease in data utility. What was unexpected, however, was a reduction in the number of equivalence classes created. Despite running the algorithm with

$k = 10$, there were final partitions of size greater than 150. Frequently, one of the numerical attributes in these partitions had a width of 1. This could already have raised some suspicion that perhaps a bug had been introduced during the simplification described above. And indeed, that was the case.

The splitting of partitions along a numerical attribute in the original implementation is carried out the following way. First, the records in the partition are sorted according to the numerical attribute. Then the median, i.e. the record at the middle index in the sorted list of records, is stored, along with the record right after the median. The first subpartition is created using the minimum value for the attribute range in the original partition as the lower bound, and the value of the "median record" as the upper bound. The smallest value of the record right after the "median record" is assigned to the minimum value of the second subpartition, and its maximum value is copied from the maximum for the given attribute range in the original partition. To prevent the creation of overlapping ranges, simply a new condition has been introduced: if the median equals the upper bound of the attribute in the original partition, the split is deemed invalid and the partition is left as is. Let's consider what this means for the range of (18-19). The only valid split is creating one subpartition for 18 and another one for 19, for both of which the number of records covered must exceed k. If the median for the attribute in this partition is 18 (and also the subpartition sizes exceed k), the two subpartitions are created. However, if the median is 19, then, using the condition introduced to prevent the creation of overlapping partitions, the split is regarded invalid. Thus, the partitioning is aborted, even though the split could be made. Once this bug was found and fixed, the output of the proof of concept approximated that of the original implementation, both in terms of the partition sizes and the GCP metric.

## 5.5 Datafly

Given the availability of Datafly implementations in open repositories, the most convenient would have been to base the proof-of-concept implementation on one of these, similarly to the approach followed for Mondrian in the previous section. However, the bottom-up nature of Datafly did not allow it to simply reuse the existing repositories due to scalability issues. Therefore, a new solution was required that is capable of executing query-based anonymization. The original proposal of the Datafly algorithm includes various parameters that can be finetuned per attribute. This provides a means to guide the heuristic that chooses the attribute to generalise in each iteration of the algorithm. In one paper, Sweeney presented a simplified algorithm which assumes complete homogeneity of the attributes in terms of their properties to fine-tune[41]. For the sake of the proof-of-concept implementation, this simplified version of Datafly is used.

Datafly relies on the data structures defined and used for the Mondrian implementation. The configuration file needs to be updated to involve an algorithm specific property per attribute, the details of which are presented in the next section. The parser of the configuration file does not need to be changed, as it simply reads all of the properties of each attribute.

A more detailed description of the algorithm is found in Chapter 2. Still, it makes sense to provide a brief reiteration of the steps that the algorithm takes. The bottom-up approach means that the algorithm starts by defining an equivalence class per unique record in the dataset and from there, by means of generalising the attributes and merging equivalence classes, eventually creates and outputs the final set of equivalence classes that already fulfil the $k$-anonymity property. The algorithm keeps track of a frequency list that contains all of the defined equivalence classes up to that point along with the number of records covered by them. Datafly also uses a heuristic for choosing the attribute the values of which are then generalised in the current iteration. The heuristic picks the attribute with the highest number of unique values. The algorithm then generalised the value of this attribute in each of the equivalence classes. This generalisation step merges equivalence classes, until there are more than k number of records that do not belong to an equivalence class of size greater than k. If the number of such records goes below k, they are simply discarded and the algorithm terminates.

The so-called synchronised step of the algorithm, i.e. always generalising the value of the attribute chosen by the heuristic for all of the equivalence classes, makes Datafly rather Inflexible. In addition, the heuristic introduces unnecessary generalisation steps along the way, which leads to an overgeneralization of the data, as Sweeney points out in his paper. These characteristics combined with the simplified version of the algorithm results in an anonymous output with considerably lower utility compared to that produced by Mondrian. However, the Datafly implementation serves as a demonstration of working with a bottom-up algorithm, and also helps to prove that the algorithms are indeed easily replaceable in the anonymization module.

### 5.5.1 Addressing the scalability issue

Following the bottom-up approach of the algorithm, as described in the original proposal, would require an enormous effort at the beginning of the anonymization procedure. To produce the frequency list, the implementation should go through all of the records in the dataset and find each of the unique values among them. This solution does not scale well as the size of the dataset increases. Generating the frequency list directly from the

records generates quite an overhead. Instead of starting from the "very bottom", skipping the initial generalisation steps and starting with higher-level equivalence classes would spare quite some time and computation. And this can be achieved by generating a set of such high-level equivalence classes, i.e. the frequency list, prior to launching the anonymizer.

Although the equivalence classes would not be constructed from individual records, thus they can already contain generalised values for the attributes, the algorithm still follows the bottom-up approach. Thus, the more specific an attribute value is kept, the more equivalence classes will need to be generated to cover the entire set of attributes in the dataset. The more equivalence classes are generated, the more statistics must then be extracted from the database through queries to construct the frequency list. It becomes clear quite fast that scalability is not an intrinsic property of this approach either. To keep the number of generated equivalence classes manageable, the parameters of the generation procedure must be set carefully . Let's see what this implies for the different attribute types.

Numerical attributes are generalised by placing them in wider and wider ranges. The database backend can be assumed to be the most efficient in defining the initial numerical ranges, considering that, through its direct access to the data, it has all the information to construct the ranges based on some statistical properties of the attribute in question. The proof-of-concept implementation relies precisely on this assumption. A corresponding function is extracted into the Datafly API that expects the spreading of a numerical attribute into a specified number of numeric ranges. On receiving the numerical attribute name and the number of ranges to produce, the database backend must return a list of numeric ranges in the form of `NumRange` objects.

When it comes to categorical attributes, only the level of the generalisation hierarchy tree needs to be defined. Based on that, the corresponding node values can be extracted and used in the initially generated equivalence classes.

The information about how to use the attributes for creating the initial equivalence classes, i.e. the width of the range in case of numerical attributes and the level of the hierarchy tree for the hierarchical ones, is the best to include in the configuration file. Thus, the parsing function can read all information about each of the attributes, from which each algorithm implementation can extract the information that is relevant for their operations. Listing 4.1 in Chapter 4 shows one segment of the configuration file for the adults dataset, with the Datafly specific attribute properties already included. If there is no Datafly-specific property defined for one attribute, the implementation assumes that the attribute is expected to be suppressed entirely. Accordingly, for categorical attributes the asterisk, for numerical ones the entire domain of the attribute will be used in the generated equivalence classes.

The initial equivalence classes are constructed in two steps. First, according to the configuration file, the desired number of ranges is generated for each numerical attribute and the node values on the defined levels are extracted for each hierarchical attribute. In the second step, all of the permutations are generated from the list of ranges and node values of the attributes. The total number of the initial equivalence classes can be easily calculated by multiplying the number of generated ranges for the numerical attributes and the number of nodes on the specified levels of the generalisation hierarchies. Considering how quickly the number of the generated equivalence classes increases, making this calculation serves as a quick check and assists in fine-tuning the generation procedure. It is important to note that this approach makes sense as long as the magnitude of the number of the initial equivalence classes stays below that of the number of records in the dataset. For example, in the case of the two datasets used in this project, i.e. the UCI adult and the Kibana web logs dataset, which contain only a small set of entries, the number of the initial equivalence classes quickly goes beyond the size of the entire datasets.

### 5.5.2 The implementation

The proof-of-concept Datafly implementation starts with generating the initial equivalence classes, as described in the previous section. For each equivalence class, the number of records covered by them is queried using the database API. This process corresponds to the creation of the frequency set in the description of the original algorithm. The Datafly-specific API only contains one "endpoint", or function in case of the proof of concept written in Python, namely the one used for calculating the percentiles and creating the numerical ranges for the initial equivalence classes. The rest of the functionality required by the algorithm is already part of the universal API that all database backends are supposed to implement.

With the frequency set, i.e. the initial equivalence classes, already in place, the anonymization algorithm starts. In each iteration the attribute with the greatest number of distinct values in the frequency set is picked for generalisation. Hierarchical attributes are generalised by stepping one up in the generalisation hierarchy tree and replacing the current node value with that of the parent node. Numerical attributes are generalised by merging two adjacent ranges together. For example, in case of the ranges (48-49), (50-55), (56-66) and (67-80), the generalised ones would be (49-55) and (56-80). It is important to keep in mind that Datafly generalises each equivalence class, i.e. each item in the frequency set, at the same time.

The algorithm terminates once there are less than k number of records that do not yet belong to an equivalence class. Each of these records are suppressed completely, while the final equivalence classes are pushed to the database backend.

### 5.5.3 Challenges

During the implementation of the algorithm, one anomaly appeared. Once the equivalence classes for the adults dataset had been created and sent to the database backend, and the anonymization terminated, the anonymous dataset ended up with more records compared to the original dataset. This suggested that some of the equivalence classes might be overlapping. Looking into the generated equivalence classes, this indeed was the case. The issue was caused by the marital status attribute.

Taking a look at Figure 2.2 that depicts the generalisation hierarchy of the marital status attribute, it is easy to notice that the tree itself is unbalanced. And this is precisely what introduced the bug into the implementation. If the configuration file prompts the anonymization module to include second-level node values of the marital status in the initial equivalence classes, it cannot find any such node in the "Never-married" branch of the tree. In case the given branch is more shallow, i.e. does not have any nodes on the specified level, the leaf nodes of the branch are taken instead. Since the generalisation steps are synchronised, the shorter branches will reach the root node faster than the others. If this happens, then one partition is created with the fully generalised root node value, while the other partitions contain internal node values. The one partition with the root value will cover all of the records belonging to the other partitions. And thus, the same records will be anonymized multiple times.

One solution would be, in case of unbalanced trees, to discard all other partitions as soon as another value reaches the root node in the generalisation procedure. This would introduce yet another overgeneralization step. Instead, the implementation, before carrying out the generalisation for a hierarchical attribute, checks whether the attribute values in the partitions are on the same level. If not, only those partitions will be generalised that have the node values of the greatest depth. This way, even if some partitions start from a deeper level in the generalisation hierarchy tree, they can catch up with the other partitions. At some point, the node values across the partitions will be aligned, i.e. each will be a node value from the same level, after which the generalisation steps can be synchronised again for this hierarchical attribute, as well.

## 5.6 Elasticsearch

Having extracted the algorithm-specific API, it was time to set up the Elasticsearch database and the backend that relays the communication between the database and the algorithm. For the proof of concept a NetEye instance was used that comes packaged with Elasticsearch. NetEye currently supports and is shipped with the version 7.17 of Elasticsearch. The proof-of-concept implementation, of course, also works with a pure Elasticsearch instance without a NetEye installation. However, it cannot be guaranteed that the implementation functions out-of-the box with other versions of Elasticsearch. It is tested for version 7.17 only. For example, the dataset of web logs shipped with Kibana by default looks slightly different in the latest version. Similarly, minor changes might have been or may be introduced to the query API of Elasticsearch, which could break the current implementation of the proof of concept backend.

For ingesting the adults dataset, already cleared as described in section 4.2, into Elasticsearch, a Logstash instance is used that also comes preinstalled with NetEye. Logstash provides support for a wide range of data sources. It is capable of parsing CSV files, as well. The configuration for the Logstash pipeline that reads and sends the adults data to Elasticsearch along with some general hints are included in the GitHub repository. Besides, through the Kibana frontend it is also possible to directly upload CSV files. Thus, setting up a Logstash instance can be entirely omitted.

### 5.6.1 Reaching Elasticsearch from Python

The proof of concept relies on the official Elasticsearch Client Python library[36]. It makes the Elasticsearch API easy to reach and work with from code, providing access to the entirety of its functionality. The queries are assembled as strings which is a rather error-prone approach. However, considering that the proof-of-concept implementation operates with queries of moderate complexity, it is not a major problem. For more elaborate queries, it is recommended to use the Elasticsearch-DSL library that serves as a higher level client library. It comes with additional abstractions and data structures that facilitate the writing and manipulation of queries, making the interaction with Elasticsearch through Python code even more convenient.

For the authentication of the Elasticsearch client, the current proof-of-concept implementation uses an API key. An API key for Elasticsearch can be generated through a dedicated API request or directly from the

Kibana UI. The main function reads the API key, along with the address of the Elasticsearch instance and the name of the index containing the data to be anonymized from environment variables.

### 5.6.2 Implementation of the APIs

The operations required by the two algorithms along with the universal API for algorithm backends in general are already defined. Although the corresponding queries in Elasticsearch are not particularly complicated, the usage of the query API is not trivial. Elasticsearch comes with a thorough documentation with plenty of explanations and examples, which helps to get through the initial hurdles relatively easily. The various database operations of Elasticsearch are reachable through a REST API. The Kibana Dev Tools, available in the Kibana user interface, is a helpful tool for preparing, executing and trying out any Elasticsearch API call. Its autocomplete feature also assists the user in constructing requests with the correct syntax.

The universal API contains three simple aggregate queries: counting the number of documents, and calculating the minimum and maximum of an attribute. Each of these operations must be made available for subsets of the dataset which are filtered using the descriptor of the equivalence classes. There is a dedicated endpoint, namely `index_name/_count`, for counting the documents. All other types of queries are available through the `index_name/_search` endpoint. Aggregate queries also return the documents that are used for the aggregation, unless explicitly prompted not to do so. The syntax for filtering among documents is the same for both API endpoints. The construction of the filter condition might be somewhat more challenging, otherwise the count, minimum and maximum operations are, of course, provided out-of-the-box by Elasticsearch.

Both of the two algorithm-specific APIs only define one extra operation. Mondrian requires the splitting into two of a numerical range along its median. The algorithm needs a new upper bound for one subpartition, and a new lower bound for the other one. Accordingly, beside the median value, the next smallest or previous greatest value must also be obtained. Keep in mind that the Mondrian implementation of the proof of concept does not support overlapping numerical ranges. If the median equals the maximum of the range, the numerical attribute of one partition is assigned the median. That of the other partition will be a range with the minimum of the original range as its minimum and the greatest value before the median as its maximum. If the median, on the other hand, is smaller than the maximum of the range being split, one of the subpartitions is assigned a new range with the minimum of the original range as its minimum and the median as its maximum. The smallest value after the median serves as the minimum of the new range of the other subpartition, while its maximum will be that of the original range. Elasticsearch comes with an aggregate operation that returns the specified percentiles for a numerical attribute. Querying the 50th percentile yields precisely the median of the attribute. The greatest value before the median, or the smallest one right after is obtained in a second step through a simple maximum or minimum aggregation filtering for values less or greater than the median, respectively.

The only operation that a database backend must implement to support the Datafly algorithm is the spreading of the domain of a numerical attribute into the specified number of ranges. One way to define numerical ranges is to obtain the minimum and maximum for the attribute in question, and decide on the width of the ranges. For example, for an attribute with a minimum of 10 and a maximum of 100, and the range width of 10 defined would result in the ranges of (10-19), (20-29), (30-39), etc. This approach is rather naive, as it does not consider the ingested data at all, other than considering its minimum and maximum. Instead, the distribution of the data can be taken into account to generate equivalence classes better tailored to the actual data. In this case, the parameter to fine-tune is the number of ranges that is meant to be defined. Combining this with the calculation of percentiles yields a set of ranges more adapted to the underlying data. Thus, percentiles aggregation of Elasticsearch is adequate for responding to this request. The total of 100 percentiles are divided into as many steps as the number of ranges defined in the configuration file. For example, wanting to create four ranges yields the percentiles of 25, 50, 75 and 100. For the given attribute a query must be formulated for each percentile. The result of each query will determine the upper bound of the new range, while the lower bound can be simply set to the upper bound of the previous range.

### 5.6.3 Data mapping

To formulate the filter conditions, the descriptor of the equivalence classes stored in the form of Python objects must be translated into Elasticsearch queries. Besides, once the algorithms terminate and the final equivalence classes are produced, the same descriptors must be mapped into Elasticsearch documents with fields of the right data types.

Elasticsearch supports numerical range queries; this functionality can be leveraged for mapping the generalised range values of equivalence classes directly to queries. Hierarchical attribute values are stored as strings, or to be more precisely as keyword data types in Elasticsearch. It is important to note that the internal node value of the generalisation hierarchy trees cannot be directly used for querying, as the database does not have any means of resolving these values by default. Even though the resolution could be done, for example, through runtime fields, it would require the maintenance of the generalisation hierarchy tree both in the backend and

directly in the database. Instead, it is the backend that resolves internal node values to all of their corresponding leaf values. The queries are then built using only values that correspond to the values stored in the Elasticsearch indices. For fields of the keyword type an array of values can be provided and included in the query to filter for multiple values.

The mapping of the generalised attribute values of an equivalence class works quite similarly. Not only range queries, but also range data types are supported by Elasticsearch. Numerical attributes are mapped to such numerical range fields. Hierarchical attributes are mapped to keyword fields with one or more values, depending on whether the attribute value in the equivalence class corresponds to a leaf or an internal node of the generalisation hierarchy tree. There is another type of attribute that comes into play when producing the anonymized dataset, namely the sensitive attribute.

Due to the $k$-anonymity property each equivalence class contains at least k elements. Accordingly, when constructing the anonymous dataset, there will be k number of entries from which the sensitive attributes must be extracted and inserted into the anonymous index concatenated with the equivalence class attributes. Creating a dedicated anonymous document with all this information for each original document introduces a large overhead due to the attribute values of each equivalence class being stored at least k number of times. In an SQL database the equivalence classes can be placed in a dedicated table, and the sensitive attribute values can be stored separately in another one. Then, the sensitive attribute values are linked to the equivalence classes through their ID number, thus preventing the duplication of data. However, this approach of splitting data into tables, which can then be joined on demand, is not meant for NoSQL databases, like Elasticsearch. So the challenge is to create a $k$-anonymous Elasticsearch index with the least amount of data duplication, while at the same time keeping it easily and efficiently searchable.

The most straightforward solution is to map the original documents one-by-one to the anonymous ones. While from the perspective of data storage it is rather wasteful, working with and querying the resulting index is straightforward. Alternatively, if an equivalence class contains the same set of sensitive attribute values multiple times, multiple anonymous documents can be merged. Such merged documents would contain an additional count field representing the number of times the current sensitive attribute value occurs. This already compresses the anonymous index to a certain extent, while not adding particular complexity to the queries. Another approach is storing each equivalence class once, with an array of the sensitive values that belong to them. Storage-wise this is undoubtedly the most efficient solution. However, during queries working with these arrays needs quite an effort. In addition, this approach is not applicable in case of multiple sensitive attributes: storing each of the sensitive attribute values in their separate, dedicated arrays would sever the link between the sensitive attributes that belong together. To circumvent this, the unique sensitive attribute value combinations could be stored together as objects inside an array per equivalence class. Since these sensitive attribute value combinations are already objects, their count inside the equivalence class can also be easily included, similarly to the second approach presented here. While from the perspective of human readability, this approach provides the best solution, the obstacles inflicted on querying through this document structure might not be worth the trade-off. The proof of concept contains an implementation for all of the four approaches introduced above. Depending on the use case, one or another might prove to be the best to work with.

The most straightforward solution is to map the original documents one-by-one to the anonymous ones. While from the perspective of data storage it is rather wasteful, working with and querying the resulting index is straightforward. An anonymous document created from the Kibana web logs using this approach is presented in Figure 5.6. Remember that there are two sensitive attributes in this dataset, namely the host and the request. The rest of the attributes are indirect ones that are generalised for achieving anonymity.

Alternatively, if an equivalence class contains the same set of sensitive attribute values multiple times, multiple anonymous documents can be merged. Such merged documents would contain an additional count field representing the number of times the current sensitive attribute value occurs. This already compresses the anonymous index to a certain extent, while not adding particular complexity to the queries. Figure 5.7 shows that the anonymous document is simply extended with a count attribute.

Another approach is storing each equivalence class once, with an array of the sensitive values that belong to them. Storage-wise this is undoubtedly the most efficient solution. However, during queries working with these arrays needs quite an effort. In addition, this approach is not applicable in case of multiple sensitive attributes: storing each of the sensitive attribute values in their separate, dedicated arrays would sever the link between the sensitive attributes that belong together. Figure 5.8 presents the outcome of an attempt at using this approach and how the arrays of values in the host and request attributes yield a confusing result.

To circumvent this, the unique sensitive attribute value combinations could be stored together as objects inside an array per equivalence class. Since these sensitive attribute value combinations are already objects, their count inside the equivalence class can also be easily included, similarly to the second approach presented here. While from the perspective of human readability, this approach provides the best solution, the obstacles inflicted on querying through this document structure might not be worth the trade-off. The sensitive attribute of such an anonymous document is presented in Figure 5.9. The JSON format of the document, as it is stored in Elasticsearch, is shown here, because Kibana visualises such an array of objects in a confusing way.

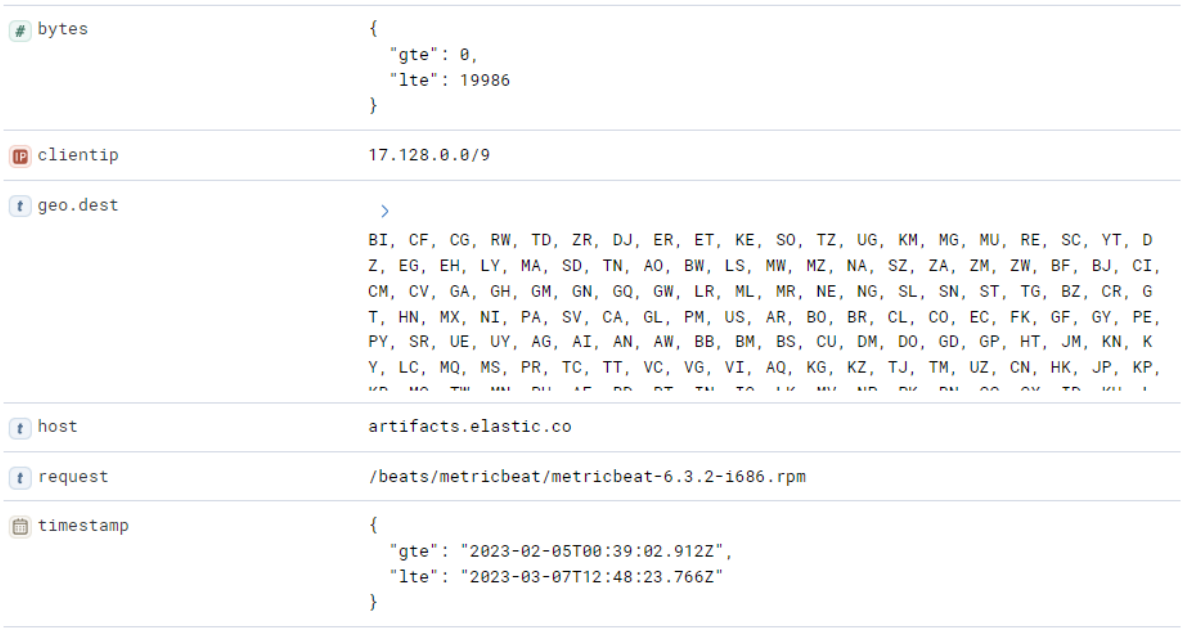


Figure 5.6: An anonymous document created from the Kibana web logs, using the approach of mapping each original documents to an anonymous ones

The proof of concept contains an implementation for all of the four approaches introduced above. Depending on the use case, one or another might prove to be the best to work with.

## 5.7 MySQL

The primary goal of developing a second database backend for the proof of concept is demonstrating the replaceability of the database-specific component in the architecture of the anonymization module. The algorithms are supposed to rely only on the abstract API that they are interacting with, and they should function the same way regardless of the actual backend implementation they are communicating with. SQL databases have not lost their popularity, and both the company and its customers probably store data also in relational databases. Consequently, the demonstration of how anonymization works inside MySQL can be of value for Würth Phoenix.

### 5.7.1 Reaching MySQL from Python

Before loading the data into MySQL, it is necessary to create a dedicated database with two tables, one for the adult data and another one for its anonymized version. CSV files are easily loaded into MySQL with the dedicated `LOAD DATA` command. It is also a good practice to create a database user with limited access rights, with which the anonymization module can authenticate itself to MySQL. The SQL commands for setting up the database and parsing the adults data file can be found in the *README* file of the GitHub repository[35].

The Python library developed by MySQL[31] makes it easy to connect to and work with the database from code. The connector requires the address of the database, the username and the password, and the name of the database that needs to be available by this point. The database backend of the proof of concept, similarly to the Elasticsearch backend, reads all information necessary for the connection from environment variables.

### 5.7.2 Implementation of the APIs

The implementation currently assembles the database queries with the help of simple string concatenation. To prevent SQL injection, the Python library for MySQL provides an easy way to safely parse external data into the queries.

The universal API is rather easy to implement with MySQL, as well. The aggregate operation of COUNT, MIN and MAX are, of course, provided out of the box. However, in MySQL a default operator for calculating percentiles does not exist. There are various solutions proposed by the community to produce the percentiles for a numerical attribute. One of the most simple ways, used also in this proof of concept, is to first sort the data according to the attribute in question in descending order, and then take the nth row where n corresponds

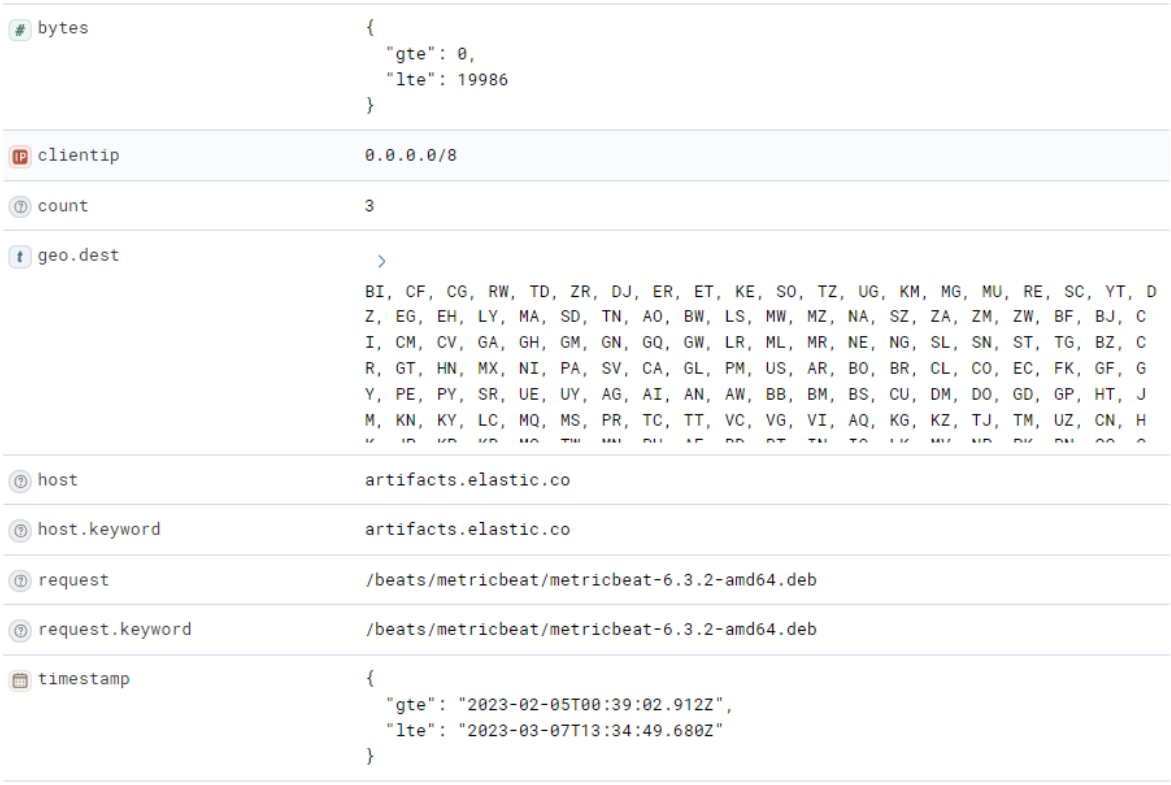


Figure 5.7: An anonymous document created from the Kibana web logs, using the approach of merging anonymous documents in the same equivalence class with the same sensitive attribute value into one document, and adding a count attribute

to the desired percentile. The value of n can be calculated by multiplying the size of the result set with the desired percentile divided by 100.

Once an implementation for calculating percentiles is provided, the algorithm-specific APIs can be implemented the same way it is done in the Elasticsearch backend. Delivering the median and the previous or next value for Mondrian is again carried out in two steps. First, the median is obtained by querying the 50th percentile for the specified attribute. Then, depending on whether the median equals the maximum value of the numerical range of the attribute, the previous greatest value before or the next smallest one after the median is queried using the MIN and MAX SQL operators. Spreading the attribute range into the given number of subranges for Datafly is achieved by executing the percentiles query multiple times.

### 5.7.3 Data mapping

Partitions must first be mapped to conditions for querying with the SQL WHERE statements, and at the end of the anonymization procedure they must be transformed into SQL INSERT statements in order to be added to the anonymous table. The difference between the two database technologies, MySQL and Elasticsearch, becomes evident when observing the data mapping functions.

Similar to the numerical range queries and the possibility to use an array of values when specifying filtering conditions in Elasticsearch, the SQL language supports the same functionalities through the BETWEEN and IN operators. Thus, numerical attributes are trivial to query. When it comes to hierarchical attributes, it is important to note again that the internal node values, i.e. the generalised attribute values cannot be directly queried without explicitly implementing a function for resolving it inside MySQL. This would require the maintenance of the generalisation hierarchies also on the database size: To avoid this, just like for Elasticsearch, the resolution should instead be done on the backend side, and the database should obtain only leaf values inside queries that it can directly work with.

Once the equivalence classes are created, it is more straightforward how to create the anonymous table in MySQL than it is in Elasticsearch. To avoid data duplication the equivalence classes can be stored in a separate table, and the sensitive attributes can be placed in another one, referencing their equivalence classes through IDs. The proof of concept for now, however, stores the anonymous data in one single table.

SQL databases do not support range data types. Therefore, the values of generalised numerical attributes must be mapped into two fields. The proof-of-concept implementation creates an *attribute_from* and an *at-*

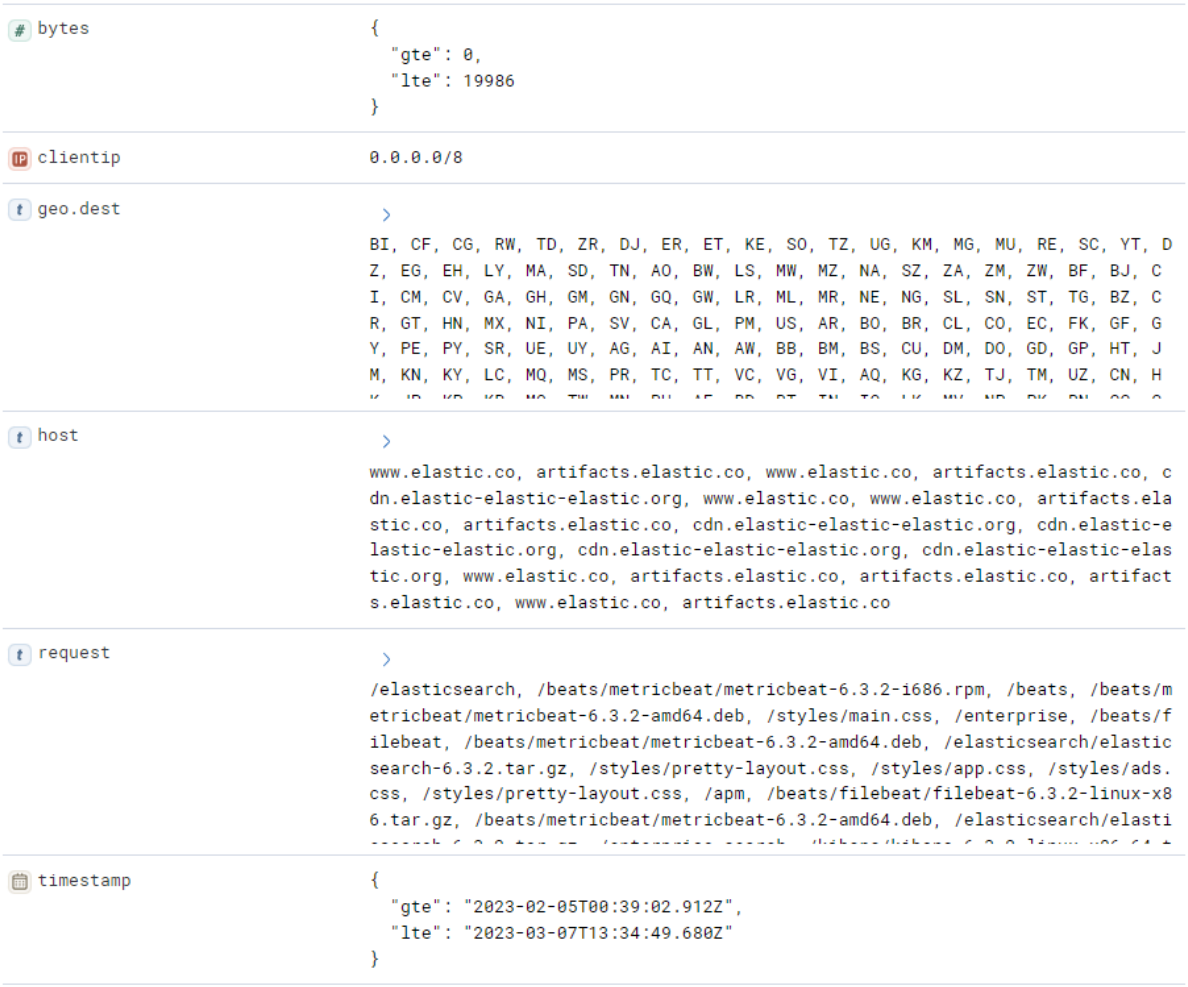


Figure 5.8: An anonymous document created from the Kibana web logs, putting each sensitive value belonging to the equivalence class into a single array

*tribute_to* column for each numerical attribute in the anonymous table. Currently, hierarchical attributes are simply stored as the values that the equivalence class contains, i.e. they are not mapped to the leaf values similar to the approach followed in the Elasticsearch backend. This, however, makes working with the anonymous data rather difficult. Since only the internal node values can be queried directly, some sort of value resolver mechanism should be inserted between the sending of the SQL query and its execution in the database. Storing the leaf values, as in the Elasticsearch backend, instead of the internal node values is problematic due to the fact that, other than PostgreSQL, SQL databases do not support arrays as a native data type. There are workarounds, like using join tables or leveraging the JSON data type, to make an array storage-like functionality available. As always, these have their own advantages and disadvantages that are best considered based on the use case at hand. The current proof of concept relies on the most simple approach.

## 5.8 New data types

The two algorithms with the two database backends were implemented focusing on the UCI adult dataset. Accordingly, only the two most common attribute types, namely the numerical and hierarchical ones, were supported at that point. The next step was to introduce new attribute types into the proof-of-concept implementation. From the side of Würth Phoenix the greatest interest was shown in the anonymization of IP addresses and dates, as these two are frequently recurring data types in logs.

Using logs from a production environment was considered, however extracting and working with them would have required extra caution. For demonstration purposes, however, any sample data containing such attributes is sufficient. Kibana comes with three sample dataset, from which the web logs have been chosen to present the anonymization of IP addresses and dates. It is a convenient choice, as, similarly to the adults dataset, it is accessible for anyone, anytime. The Kibana web logs dataset is presented in more detail in Section 5.3.2. The bytes, geographical location of the request destination, the timestamp and the client IP addresses are assumed

```
"sensitive_attributes": [
  {
    "host": "www.elastic.co",
    "request": "/elasticsearch",
    "count": 1
  },
  {
    "host": "artifacts.elastic.co",
    "request": "/beats/metricbeat/metricbeat-6.3.2-i686.rpm",
    "count": 1
  },
  {
    "host": "www.elastic.co",
    "request": "/beats",
    "count": 1
  },
  {
    "host": "artifacts.elastic.co",
    "request": "/beats/metricbeat/metricbeat-6.3.2-amd64.deb",
    "count": 3
  },
  {
    "host": "cdn.elastic-elastic-elastic.org",
    "request": "/styles/main.css",
    "count": 1
  },
  {
    "host": "www.elastic.co",
```


Figure 5.9: An anonymous document created from the Kibana web logs, storing the unique sensitive attribute value combinations together as objects inside one array in each equivalence class

to be the indirect identifiers used for producing the equivalence classes. The request URL and path are treated as sensitive attributes. Semantically, the anonymization does not make much sense, but it is adequate for demonstration purposes.

In the current proof-of-concept implementation the anonymization of datasets with attributes of the IP address or date types is only available using top-down algorithms, i.e. Mondrian in this case, on data stored in Elasticsearch. The proof of concept only implements the splitting of these attribute types, and it is not capable of the generation of initial equivalence classes and the generalisation of such attributes that is necessary for running Datafly. The dataset is available by default in Elasticsearch. It could, of course, be exported and inserted into MySQL. When it comes to MySQL, the date type is supported, just like in any other SQL database. A dedicated data type for IP addresses does not exist, but IPs can be mapped to and stored as integers. Storing and working with subnet masks in SQL is less trivial. However, for the discussion of dates and IP addresses, this thesis focuses entirely on Elasticsearch.

### 5.8.1 Refactoring

The original Mondrian implementation, used as a starting point for the proof of concept, relied on `if` statements to differentiate between the attribute types. Consequently, the proof-of-concept implementation ended up following the same approach. As long as there were only two data types, this approach was acceptable. However, when attempting to introduce new types, it became evident that this approach cluttered the code too much and took a toll on the maintainability, extensibility and reusability of the implementation. This necessitated a major refactoring of the codebase.

The main focus of the refactoring was on the creation of a class hierarchy for attributes. The original `Attribute` class became an abstract parent class for all of the attribute types that the implementation intends to support. It retained its data storage functionality, i.e. information like the width or the generalised value of the attribute was stored by the abstract parent class. It was then extended with database specific abstract functions, like the mapping of the attribute to a query or into a database attribute. And, most importantly, the responsibility of splitting the attribute was also moved from the codebase of the algorithm to the `Attribute` itself through its abstract `split()` function.

This meant that a great portion of the original algorithm code had got moved to the `Attribute` implementation, which resulted in a cleaner codebase and the responsibilities being moved to where they belong. Besides the abstract `Attribute` parents class, an implementation was needed for the numerical and hierarchical attributes for a start. From this point on when reading the configuration file, each attribute is instantiated according to its type and stored in the `Partition` class as an abstract `Attribute` object. The algorithm only calls the abstract functions available on the `Attribute` class and for every new type a different implementation can be provided for the splitting or mapping operations.

For Mondrian, a dedicated `MondrianPartition` class was also implemented. It extended the original data storing `Partition` class with two functions that were handling the internal state of the partition: checking whether it is splittable and choosing one of its attributes for splitting. The creation of the subpartitions are, however, still part of the Mondrian codebase.

Since the `Attribute` was turned into an abstract class, the code was not backward compatible, and thus the Datafly implementation needed to be adjusted. The only necessary step was to instantiate the right type of attributes when generating the initial equivalence classes and when generalising the attributes. it is important to note, however, that the current implementation of Datafly still treats the `Attribute` subclasses as simple data storage classes. Consequently, it does not make use of the polymorphism of the `Attribute` class in any way.

The refactoring has indeed made the codebase easier to extend by moving the responsibilities to where they better belong. The next step was to introduce the two new data types. Numerical, date and IP addresses have one thing in common: all three of them are rendered anonymous by placing them in ranges. On the level of code this also implies some commonalities in implementation. Accordingly, a `RangeAttribute` abstract class has been created that serves as the parent class for attributes of type integer, date and IP address.

### 5.8.2 Dates

The complexity of the anonymization of dates depends on the resolution, where resolution refers to how detailed the date attribute is. A timestamp with milliseconds also included is of the highest resolution, while providing only the year for a certain event is evidently a lower-resolution date. The anonymization of a year-only date is trivial, it is practically a numerical attribute. However, as the resolution of the date increases, i.e. the months, days, hours, etc. are also included, the generalisation can be carried out on multiple levels. Considering a top-down approach, first the year ranges, like 1900-2000, are split as long as the splitting ends up with a concrete year, for example 1968. At that point, the splitting can continue on the level of months, the first split yielding for instance the ranges of 1968.01-06 and 1968.07-12. Once the month range is reduced to one specific month, e.g. 1968.03, the days of the month can be split, resulting in two more ranges, for instance 1968.03.01-15 and 1968.03.16-31. And this can go on until the desired specificity. The higher the resolution of the date, the more complex the anonymization logic becomes.

Probably the most common date format in logs is that of timestamps, with milliseconds also included. Thus, it makes sense to focus on this type of date in the proof of concept. There is one lucky characteristic of timestamps: they are represented as the number of milliseconds elapsed since 1st January 1970 00:00:00 UTC. This means that if the readable date format of timestamps is converted into the millisecond format, they can be treated as numerical attributes. And this is leveraged in the proof of concept. The `TimestampAttribute` inherits from the `DateAttribute`, which is a direct child of the `RangeAttribute` class. This means that the `TimestampAttribute` cannot inherit the `IntegerAttribute`, even though this class has all the functionality needed for the anonymization of timestamps. To solve this problem, instead of inheriting from `IntegerAttribute`, the `TimestampAttribute` class stores an instance of it and forwards all the operations to this `IntegerAttribute` instance.

As a matter of fact, when querying Elasticsearch, timestamp fields are returned in both numerical and string formats. Similarly, when ingesting a document into an Elasticsearch field of a date type, the millisecond format is automatically interpreted correctly. Thus, the only function the TimestampAttribute must implement is the one specifying that the corresponding field in the anonymous index should be of the date range type.

There is one point to keep in mind. The width of timestamp attribute ranges is the number of milliseconds between the upper and lower bound. This might result in a far greater magnitude in terms of width, and a far lesser one if normalised, for this attribute compared to numerical or hierarchical ones. Consequently, timestamp

attributes might always end up as the last ones being split. If this is a problem, a different width definition should be constructed.

### 5.8.3 IP addresses

When it comes to IP addresses, a natural way for placing them in ranges already exists, namely by using subnet masks. The Kibana web logs contain IPv4, i.e. IP version 4 addresses, thus this discussion and implementation focuses on this type of IPs. A subnet mask defines which portion of the 32-bit IPv4 address determines the addresses of the network and the host machines. Taking the subnet mask number of bits at the end of the address describes the network that the hosts are part of. Accordingly, the other half of the bits in the address provides the addresses of the hosts inside the subnetwork. For example, the address 143.192.89.156 with a subnet of 24, concisely written as 143.192.89.156/24, says that the first octet, in this case 156, determines a host machine inside the subnetwork with the 143.192.89.0 address.

Subnets masks can be leveraged for the anonymization of client IP addresses. Generalisation is achieved by decreasing the subnet mask. And to make an IP range more and more specific, the subnet mask can be increased in every step. Since the proof of concept only supports the anonymization of IPs with Mondrian, the discussion focuses on the top-down approach, i.e. the splitting of generalised IP ranges. The initial Mondrian partition, which must encompass all of the records in the dataset, starts with the 0.0.0.0/0 subnet mask. This covers the entire set of available IPs on the internet. On splitting a range, similarly to numerical and date attributes, two new ranges are created. The subnet mask is incremented by one, thus the number of bits that determine the network address is increased by one. In the position of the new bit, now included in the network address, one of the two ranges is assigned the value 0, while the other one takes 1, yielding the ranges of 0.0.0.0/1 and 128.0.0.0/1. Splitting the latter, again incrementing the subnet mask by one and assigning 0 and 1 to the new bit position, results in 128.0.0.0/2 and 196.0.0.0/2.

Since Mondrian decides on the next attribute to split along based on the width of the attributes, this must be defined for IP addresses, as well. An obvious choice is calculating the number of host addresses that the current subnet mask covers, i.e. $2^{32-subnet\_mask}$. In case of small subnet masks this yields huge numbers, which implies the problem already mentioned for timestamps: due to the magnitude of the width, when normalised this attribute will be one of the last ones to be chosen for splitting.

Elasticsearch, besides numerical and date ranges, comes with support for the IP range type, as well. There are two ways to determine an IP range: either by simply specifying a starting and end point, or by using subnet masks. The implementation stores the generalised value for IP addresses in the 143.192.89.156/24 format that Elasticsearch recognises and parses correctly. Thus, on producing the anonymous documents, the generalised value can be forwarded right away and it will be ingested as the desired IP range. Then, through range queries, it is easy to work with these attributes.

This approach of systematically splitting the IPs in half starting from 0.0.0.0/0 is rather rigid. It does not take into account any of the statistical properties of the attribute, for example. Unfortunately, functions for calculating percentiles, minimum and maximum are not defined for attributes of the IP type in Elasticsearch, thus the methods used for a smarter splitting of numerical and timestamp attributes are not applicable. The logic, however, could undoubtedly be refined. Another issue is overgeneralization. For example, if there are exactly k number of IP addresses in the 0.0.0.0/31 subnetwork, and eventually all of these belong to the 1.2.3.0/24 subnetwork, the anonymization procedure will produce an equivalence class with the generalised value of 0.0.0.0/31, even though it could further refine the attribute value and increase the utility of the data without decreasing the privacy guarantees.

Overall, the approach presented for the anonymization of IP addresses can be used as a starting point. However, in order to maximise the utility of the anonymous dataset, this solution in the proof-of-concept implementation must be made more sophisticated.

## 5.9 The final implementation

The `wire_up()` function in main.py shows how, leveraging the abstractions, the desired algorithm and database backend combinations can be set in the anonymization module. Of course, it must be taken into account which dataset is found in which database and whether the data types in them are supported by the chosen algorithm. However, like in the case of the adult dataset, the components are indeed easily interchangeable depending on which database and $k$-anonymization algorithm is preferred at the moment.

The proof-of-concept implementation currently comes with a connector for MySQL and Elasticsearch. The two algorithms implemented are Datafly and Mondrian. Datafly only supports numerical and hierarchical attributes, while Mondrian is prepared for the anonymization of IP addresses and timestamps, as well. Two configuration files are provided, one for the adults dataset and another one for the Kibana web logs.

Based on the proof of concept, the anonymization module is ready to be extended with new algorithms and database connectors. Refining the code of Mondrian or implementing one of the more modern algorithms opens the way for producing truly high quality anonymous data. Following the example of the attribute class hierarchy, support for new attribute types can also be developed. And if the company intends to store and anonymize data in any database technology other than Elasticsearch and MySQL, the anonymization module can be easily extended with the desired connector.

It is important to keep in mind that the primary goal of the proof of concept is not to provide sound and efficient implementations for the algorithms and the supporting database operations. Instead, it aims to demonstrate how the proposed architecture can be leveraged to develop an anonymization module that is easy to modify and extend on demand as the technology keeps evolving or the company and customers requirements change. Accordingly, the usability of the implemented algorithms and database connectors is limited, and should be revised and at least refined, if not reimplemented, for production systems.

# Chapter 6

# Discussion

Implementing the proposed architecture for the anonymization module has provided valuable feedback and brought to the surface issues that were not anticipated. Taking a look at the output, i.e. the anonymous datasets, in the two different kinds of databases also highlighted some challenges when it comes to ingesting and working with $k$-anonymized data. This chapter briefly recaps these insights and discusses some general questions around anonymization and GDPR.

## 6.1 Takeaways from the implementation

To make the anonymization scalable, the anonymization module carries out query-based anonymization. Instead of directly reading the data into memory and processing it from code, the bulk of the operations are formulated as queries, thus "outsourcing" them to the database. Although for the sample datasets currently used in the proof-of-concept implementation the communication with the database considerably slows down the anonymization procedure, the query-based approach shows its strength as the quantity of the data increases. Once the data does not fit into the memory any more, the usability of the in-memory anonymization becomes questionable. Furthermore, directly accessing the data inside the database, where it is originally located, also spares the extra steps of the extraction and rereading of the datasets. This is convenient, especially if there are a great number of datasets to anonymize.

Of course, the query-based approach does not come without its own challenges. During the implementation two major issues emerged: handling equivalence classes with overlapping attribute ranges and finding a workaround to skip the otherwise non-scalable initial steps of bottom-up algorithms, in this case those of Datafly. On the one hand, with all of the records read into memory, and thus having direct references to them, in-memory anonymization can easily move records around among equivalence classes without the overlapping ranges causing too much problem. However, when keeping track of the records through queries only, it is not possible to trivially differentiate between records belonging to two or more overlapping ranges. The trade-off to consider is whether to incur some additional computational costs or to accept the increased information loss in the data that comes with avoiding overlapping equivalence classes.

On the other hand, the Datafly implementation in the proof of concept serves as a good example how it might not be possible to realise some algorithms in practice using the query-based anonymization. Datafly follows a bottom-up approach, i.e. it creates an initial equivalence class for each record in the dataset, which the algorithm then goes on to merge until all of the merged equivalence classes exceed the k number of records. The workaround used in the proof-of-concept implementation aims to generate higher-level, initial equivalence classes based on the statistical properties of the dataset. The goal is to approximate a state that the algorithm would reach by starting from single records, as proposed in the original pseudocode. Such a scenario might occur with other algorithms too, which necessitates the development of workarounds to adapt some of the algorithm steps to the query-based approach.

It is also worth remembering that, while the proof of concept implements the anonymization module in one single Python project, the original proposal suggests using a web API for intercomponent, i.e. backend-algorithm communication. This aims to avoid locking the module into one technology or framework, and thus making it easily extensible and the components easily replaceable. While relying on web interfaces maintains the flexibility of the module, it of course incurs some additional communication overhead. This constitutes another trade-off to take into account.

The proof-of-concept implementation is only meant to demonstrate the feasibility and usability of the proposed architecture, and highlight some initial pitfalls and trade-offs to keep in mind. It has fulfilled the goal of pointing out a general direction for a future implementation, but the exact trade-offs and fine-tuning of parameters are to be decided upon there and then based on the use case in question.

## 6.2 Takeaways from the anonymous datasets

As presented in the dedicated sections for the MySQL and Elasticsearch backends in the previous chapter, the final appearance of the anonymized data might be significantly different based on the database technology. In MySQL, it is easy to reduce the storage size by creating separate tables for the equivalence classes and the sensitive attributes. However, storing arrays is not supported by default which causes issues when the leaf values belonging to an internal node of a generalisation hierarchy tree should be put into an attribute. In Elasticsearch, storing multiple values in one field is trivial, while a compact storage with easy searchability is not. Therefore, when rendering data anonymous, careful attention must be paid when defining the final database scheme.

It is important to keep in mind that the end goal is not simply anonymization itself. Anonymization only serves as a layer for protecting the personal data of individuals. Ultimately, however, the produced dataset will be subject to thorough data analysis. Whether processed by human analysts or algorithms, the anonymous data set must be both of high utility and easy to work with. Without experience, however, defining precisely what sort of output is easy to work with is not trivial either. A common practice of the industry, when lacking the experience, is "outsourcing" the generation of some initial insights or best practices to the "community" through the organisation of hackathons. While these, of course, require financial and time investment, the company might benefit from them in many ways. The participating teams can come up with creative ways to extract information from such anonymous datasets. They can give feedback on what made their work difficult which might point to some aspects to tweak when producing anonymous data.

## 6.3 Anonymization and GDPR

When it comes to anonymization, a tool in itself is not sufficient to produce GDPR-compliant anonymous datasets. First of all, having experts around with relevant knowledge in the domain of the data along with a clear understanding of statistics and privacy is inevitable for correctly mapping out the environment and setting up the anonymization procedure. The direct, indirect and sensitive attributes must be categorised. The algorithm parameters must be fine-tuned. The potential weaknesses must be taken into account. Besides, other publicly available datasets must also be considered to evaluate whether linking them with the anonymous output can compromise the privacy of any individuals. The anonymization module, while an important component, constitutes only one part of the anonymization procedure, and it must be guided and assisted by the right domain knowledge complemented with the sufficient level of scrutiny to function as a real anonymization solution.

The uncertainty around the practical implications of GDPR further complicates the clear definition of a sound anonymization procedure. Despite the regulation being in force since 2018, the implementation of the various technical and administrative measures is progressing rather slowly. To a large part this is due to the fact that it is difficult to find clear best practices even for fundamental articles of the regulation. As a matter of fact, the regulation is meant to address the legal uncertainty. Recital 9 states that "the objectives and principles of Directive 95/46/EC remain sound, but it has not prevented [...] legal uncertainty"[37]. Considering this, it is paradoxical that there seems to be no attempts at clarifying the practical implications of the regulation, and thus the uncertainty around it is not reduced in any way. For example, since 2014 the opinion of the Working Party 216 on anonymization has not been extended or updated.

When looking at the situation from the perspective of the legislator, however, this uncertainty might be intentional to a certain extent. At the current speed of technological development, it is almost impossible to anticipate the new directions in the coming years. In addition, regulating technology from the legal domain is also quite a challenge. This is why GDPR relies on terminology like "all the means reasonably likely to be used" and "reasonably feasible". Maintaining this degree of "legal uncertainty" eventually becomes an incentive for data controllers to keep an eye out for the latest developments, and to update their security and privacy measures regularly. It incentivises organisations to get involved and invest in the related research fields, for example in anonymization. And these, in turn, generate discussions and raise awareness not only in the academic world, but also among European citizens in general. Thus, GDPR indirectly becomes a driver of research, innovation and public discourse in the domain of security and privacy through maintaining uncertainty. While this undoubtedly causes some headache to data controllers, ultimately it is furthering the cause of the regulation and helps to achieve its final goal.

# Chapter 7

# Limitations

One limitation of the thesis project is its focus on one specific anonymity model. This comes with a certain degree of inflexibility, which limits the number of use cases that can be addressed through the proposed anonymization module. In comparison, the ARX data anonymization tool, for example, offers a wide range of models to choose from, which can then also be combined according to the user's preferences and needs.

Although already mentioned previously, it is important to emphasise again that the proof-of-concept implementation has been developed with several simplifications, with the sole purpose of showcasing the feasibility and the functioning of the proposed architecture. Accordingly, the implementation of the algorithms and the database queries are not optimised and might not be sound. The anonymization of the various data types could also be revised and refined. Furthermore, the proof of concept does not offer a solution for supporting non-overlapping equivalence classes with the query-first approach. Neither does it present the web API-based intercomponent communication, as proposed in the original architecture.

# Chapter 8

# Future work

The next step would be to, potentially taking the proof of concept as a starting point, implement a more robust and performant anonymization module by optimising both the database- and the algorithm-specific components. The web interface should be introduced to make the two components independent from each other. It would be worth looking into more modern $k$-anonymity algorithms that are capable of producing a better approximation of an optimal $k$-anonymization.

Afterwards, the various simplifications can be addressed. More sophisticated ways can be introduced for the anonymization of the various attribute types. A solution needs to be proposed for supporting overlapping equivalence classes. These steps can considerably increase the utility of the anonymized datasets.

To improve the privacy guarantees, the $k$-anonymity model can be updated. In the opinion of the Working Party 216 of the European Data Protection Board the known weaknesses of $k$-anonymous datasets are presented, along with some updated models that address precisely these weaknesses. Looking into $l$-diversity and $t$-closeness, and algorithms capable of producing datasets that fulfil the properties of these models, would result in a more robust anonymization module in terms of privacy. Furthermore, the introduction of other anonymity models, like differential privacy, into the anonymization module can be considered to further extend the capabilities of the solution.

When it comes to addressing use cases, there are two directions for future work. On the one hand, further connectors can be implemented for any of the database technologies that the company or its customers use for storing personal data. On the other hand, other items from the anonymization taxonomy, as proposed in Chapter 3, could be considered. By taking a closer look at these and initiating discussions with various stakeholders around the company or with its customers, potentially new use cases can be identified. Such use cases could, in turn, highlight new directions for the further development and extension of the anonymization module.

# Chapter 9

# Conclusion

The main goal of this thesis project has been to propose a way for Würth Phoenix to include anonymization in their service portfolio. Since the GDPR became applicable in 2018, the company has been actively seeking ways to facilitate the compliance of their customers with the regulation, partly relying on already existing functionality of their product, NetEye, and partly through the development of additional tools. Recital 26 of the GDPR states that the regulation does not apply to anonymized data. And this is precisely why anonymization is of interest for the company, and why the inclusion of data anonymization in their GDPR-related services provides value.

It is primarily log anonymization that the company deems as potentially relevant for their customers, and thus the main interest of Würth Phoenix lies in anonymization inside Elasticsearch. However, considering the recent rapid technological advancement, it seems irrational to focus entirely on one technology. Therefore, the thesis project proposes a flexible architecture for an anonymization module that makes it possible to easily replace the data source and algorithm used. The benefit of such an architecture is threefold. Firstly, even though the logs of Würth Phoenix and its customers are stored in Elasticsearch, a demand might emerge for the anonymization of personal data stored in other technologies. Keeping the anonymization easily adaptable helps to quickly respond to such new requirements. Secondly, the company might decide to pivot to other log storage technologies, like Splunk or any upcoming competitor of Elasticsearch. In this case, again, the flexibility will be beneficial. Thirdly, as the research in anonymization advances, new algorithms will undoubtedly be popping up in the coming years. Therefore, the replaceability of the algorithms is crucial for a robust solution.

For the anonymity model the industry-standard $k$-anonymity has been chosen. The decision was made after proposing an anonymization taxonomy, considering various use cases and taking into account which of these could be of interest for the company. The log anonymization that Würth Phoenix is interested in belongs to the static, one-time, server-side anonymization. And such use cases can be addressed through producing $k$-anonymous datasets.

Based on the proposed architecture, a proof of concept has been developed to demonstrate the feasibility and functioning of the anonymization. The proof-of-concept implementation comes with an Elasticsearch and a MySQL backend. Both of them support the anonymization of numerical and hierarchical attributes, while the Elasticsearch backend is also capable of anonymizing timestamps and IP version 4 addresses. The proof of concept can run two $k$-anonymization algorithms, Datafly and Mondrian. Furthermore, it comes with two configuration files, one for the UCI adult dataset and another one for the Kibana web logs. These two present the general layout for a correct configuration file of the anonymization module. They show the various properties that are expected to be present, along with how the types, the algorithm specific information and the generalisation hierarchies per attribute are to be defined.

While the proof-of-concept implementation, per definition, comes with limitations and simplifications, it can serve as a good starting point for the company to start their discussions about anonymization and to develop their own implementation. During one of the weekly developer meetings at the company, the thesis project was presented in the frame of a one-hour workshop, with the first 20 minutes dedicated to a theoretical introduction of the basic concepts in the field of anonymization along with the proposed anonymization module. The rest of the workshop focused on the demonstration of the anonymization of the two datasets with the help of the proof-of-concept implementation. In addition, the module has been set up on one of the test machines of the company, so that anyone interested in trying out or developing further the implementation has direct access to it.

Referring back to Chapter 8, there are multiple directions for extending the anonymization module. The current work has shown that the flexibility in terms of algorithms and data sources can be achieved relying on the proposed architecture, and that the set of supported data types can be easily extended. With its feasibility proven, the next step is to make the solution more performant and robust.

# Bibliography


[1] *Accountability — European Data Protection Supervisor*. en. URL: https://edps.europa.eu/data-protection/our-work/subjects/accountability_en (visited on 07/10/2023).

[2] *Aircloak - Data Anonymization*. URL: https://aircloak.com/ (visited on 07/11/2023).

[3] Aleem Akhtar. *Popularity Ranking of Database Management Systems*. Jan. 2023. DOI: 10.48550/arXiv.2301.00847.

[4] *Apache Lucene*. URL: https://lucene.apache.org/index.html (visited on 07/08/2023).

[5] *Apple Differential Privacy Technical Overview*. URL: https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf (visited on 07/10/2023).

[6] *Application Performance Monitoring (APM) with Elastic Observability*. URL: https://www.elastic.co/observability/application-performance-monitoring (visited on 07/08/2023).

[7] *ARX - Data Anonymization Tool*. URL: https://arx.deidentifier.org/ (visited on 07/11/2023).

[8] Ronny Kohavi Barry Becker. *UCI adult income dataset*. DOI: 10.24432/C5XW20. URL: http://archive.ics.uci.edu/dataset/2/adult (visited on 07/11/2023).

[9] A. Calle, Yucheng Chen, and Zhiqiang Hao. "Comparing K-Anonymity and Epsilon-Differential Privacy Effectiveness For Social Media". In: URL: https://www.semanticscholar.org/paper/Comparing-K-Anonymity-and-%CE%B5-Differential-Privacy-Calle-Chen/750001e7c7de74480b485718395dfbbcbb1144c8 (visited on 07/11/2023).

[10] Ashish Dandekar, Debabrota Basu, and Stephane Bressan. *Differential Privacy at Risk: Bridging Randomness and Privacy Budget*. 2020. DOI: 10.48550/arXiv.2003.00973.

[11] *Data Anonymization*. URL: https://corporatefinanceinstitute.com/resources/business-intelligence/data-anonymization/ (visited on 07/08/2023).

[12] *Data Storage and Anonymization — Yahoo*. URL: https://legal.yahoo.com/ca/en/yahoo/privacy/topics/datastorage/index.html (visited on 07/10/2023).

[13] Manolis Terrovitis Dimakopoulos Dimitris Tsitsigkos and Nikolaos. *Amnesia Anonymization Tool*. URL: https://amnesia.openaire.eu/ (visited on 07/11/2023).

[14] *Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. Oct. 1995. URL: http://data.europa.eu/eli/dir/1995/46/oj/eng (visited on 07/08/2023).

[15] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography*. Lecture Notes in Computer Science. Springer, 2006, pp. 265–284. ISBN: 978-3-540-32732-5. DOI: 10.1007/11681878_14.

[16] *General Introduction — NetEye User Guide*. URL: https://neteye.guide/4.20/introduction/general.html (visited on 07/08/2023).

[17] Gabriel Ghinita et al. "Fast data anonymization with low information loss". In: *Proceedings of the 33rd international conference on Very large data bases*. Sept. 2007, pp. 758–769. ISBN: 978-1-59593-649-3.

[18] Qiyuan Gong. *Basic Mondrian*. URL: https://github.com/qiyuangong/Basic_Mondrian (visited on 07/11/2023).

[19] Michael Hirsch. *Anonymize-It: The General Purpose Tool for Data Privacy Used by the Elastic Machine Learning Team*. Aug. 2018. URL: https://www.elastic.co/blog/anonymize-it-the-general-purpose-tool-for-data-privacy-used-by-the-elastic-machine-learning-team (visited on 07/10/2023).

[20] *How Google anonymizes data – Privacy & Terms – Google*. URL: https://policies.google.com/technologies/anonymization?hl=en-US (visited on 07/10/2023).

[21] *Icinga*. URL: https://icinga.com/ (visited on 07/08/2023).

[22] *International Country Codes*. URL: https://cloford.com/resources/codes/index.htm (visited on 07/11/2023).

[23] Lynda Kacha, Abdelhafid Zitouni, and Mahieddine Djoudi. "KAB: A new k-anonymity approach based on black hole algorithm". In: *Journal of King Saud University - Computer and Information Sciences* 34.7 (July 2022), pp. 4075–4088. ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2021.04.014.

[24] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan. "Mondrian Multidimensional K-Anonymity". In: *22nd International Conference on Data Engineering (ICDE'06)*. Apr. 2006, pp. 25–25. DOI: 10.1109/ICDE.2006.101.

[25] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity". In: vol. 2. May 2007, pp. 106–115. ISBN: 978-1-4244-0803-0. DOI: 10.1109/ICDE.2007.367856.

[26] Dale McDiarmid Luca Wintergerst Mike Paquette. *Protecting GDPR Personal Data with Pseudonymization*. Mar. 2018. URL: https://www.elastic.co/blog/gdpr-personal-data-pseudonymization-part-1 (visited on 07/10/2023).

[27] A. Machanavajjhala et al. "L-diversity: privacy beyond k-anonymity". In: *22nd International Conference on Data Engineering (ICDE'06)*. Apr. 2006, pp. 24–24. DOI: 10.1109/ICDE.2006.1.

[28] Abdul Majeed and Seong Oun Hwang. "Rectification of Syntactic and Semantic Privacy Mechanisms". In: *IEEE Security & Privacy* (2022), pp. 2–16. ISSN: 1558-4046. DOI: 10.1109/MSEC.2022.3188365.

[29] Kristóf Majkut. "Kliensoldali Anonimizálási Módszer a GDPR Megfelelés Érdekében (Client-side Anonymization Method for GDPR Compliance)". hu-HU. Bachelor's Thesis. Budapest University of Technology and Economics, 2019.

[30] Adam Meyerson and Richard Williams. "On the Complexity of Optimal K-Anonymity." In: vol. 23. June 2004. DOI: 10.1145/1055558.1055591.

[31] *MySQL :: MySQL Connector/Python Developer Guide :: 4 Connector/Python Installation*. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-installation.html (visited on 07/11/2023).

[32] Article 29 Data Protection Working Party. *Opinion 05/2014 on Anonymisation Techniques*. Apr. 2014. URL: https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf (visited on 07/11/2023).

[33] Lorenza Peschiera. *Würth Phoenix porta l'IT dall'Italia nel mondo*. it-IT. Apr. 2022. URL: https://www.datamanager.it/2022/04/wurth-phoenix-porta-lit-dallitalia-nel-mondo/ (visited on 07/08/2023).

[34] Fabian Prasser et al. "Flexible data anonymization using ARX—Current status and challenges ahead". In: *Software: Practice and Experience* 50.7 (2020), pp. 1277–1304. ISSN: 1097-024X. DOI: 10.1002/spe.2812. (Visited on 07/10/2023).

[35] *Proof-of-concept implementation of the anonymization module*. URL: https://github.com/majkutK-unitn/anonymization_module (visited on 07/11/2023).

[36] *Python Elasticsearch Client — Elasticsearch 7.17.7 documentation*. URL: https://elasticsearch-py.readthedocs.io/en/v7.17.7/ (visited on 07/11/2023).

[37] *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, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. 2016. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj (visited on 07/11/2023).

[38] Julián Salas, David Megías, and Vicenç Torra. "SwapMob: Swapping Trajectories for Mobility Anonymization". In: *Privacy in Statistical Databases*. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 331–346. ISBN: 978-3-319-99771-1. DOI: 10.1007/978-3-319-99771-1_22.

[39] Pierangela Samarati and Latanya Sweeney. "Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression". In: (Mar. 1998).

[40] L. Sweeney. "Guaranteeing anonymity when sharing medical data, the Datafly System". In: *Proceedings: a conference of the American Medical Informatics Association. AMIA Fall Symposium* (1997), pp. 51–55. ISSN: 1091-8280.

[41] Latanya Sweeney. "Achieving k-anonymity privacy protection using generalization and suppression". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (Oct. 2002), pp. 571–588. ISSN: 0218-4885. DOI: 10.1142/S021848850200165X.

[42] *The ELK Stack*. URL: https://www.elastic.co/what-is/elk-stack (visited on 07/08/2023).

[43] Joana Tomás, Deolinda Rasteiro, and Jorge Bernardino. "Data Anonymization: An Experimental Evaluation Using Open-Source Tools". In: *Future Internet* 14 (May 2022). DOI: 10.3390/fi14060167.

[44] *Unified monitoring — Würth Phoenix*. URL: https://www.wuerth-phoenix.com/en/solutions/it-system-management/unified-monitoring (visited on 07/08/2023).

[45] *Würth - Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=W%C3%BCrth&oldid=1162038783 (visited on 07/08/2023).

[46] *Würth Group*. URL: https://www.wuerth-phoenix.com/en/about-us/about-wuerth-phoenix/wuerth-group (visited on 07/08/2023).

[47] *Würth Phoenix: ERP—CRM—Service Management—Cyber Security*. URL: https://www.wuerth-phoenix.com/en/?r=1&cHash=79f032f0608474491b2b6a77d4f98dfb (visited on 07/08/2023).

# List of Figures

# List of Tables

# Listings