Monolith to serverless microservice application evolution

Novokmet, Ana

University of Rijeka – Faculty of informatics and digital technologies

Informational and communicational systems

Ana Novokmet

Monolith to serverless microservice application evolution

Master's thesis

Mentor: Assist. Prof. Dr. Vedran Miletić

Rijeka, September 2022.

Rijeka, 28. ožujka 2022.

Zadatak za diplomski rad

Pristupnik: Ana Novokmet

Naziv diplomskog rada: Evolucija monolitne web aplikacije u mikroservisnu besposlužiteljsku aplikaciju

Naziv diplomskog rada na eng. jeziku: Monolith to serverless microservice application evolution

Sadržaj zadatka:

Suvremeni sustavi na webu sastoje se od većeg broja više ili manje neovisnih komponenata. Ukoliko su realizirani kao monolitna web aplikacija, te su komponente dio jedne aplikacije, dok su kod realizacije korištenjem mikroservisa radi o više različitih neovisnih servisa. Mikroservisna arhitektura, iako kompleksnija, olakšava suradnju većeg broja programskih inženjera te nudi mogućnost razvoja svakog od servisa u vlastitom jeziku i okviru, ako se za tim pokaže potreba. Cilj ovog rada je opisati proces transformacije monolitne web aplikacije u mikroservisnu besposlužiteljsku aplikaciju na praktičnom primjeru.

Mentor:
Doc. dr. sc. Vedran Miletić

Voditeljica za diplomske radove:
Prof. dr. sc. Ana Meštrović

Komentor:

Zadatak preuzet: 28. ožujka 2022.

Ana Novokmet

Fakultet informatike i digitalnih tehnologija
Radmile Matejčić 2, 51000 Rijeka, Hrvatska
http://inf.uniri.hr ured@inf.uniri.hr
Tel: +385 (0)51584700 | Fax: +385 (0)51 584 749

Summary

Nowadays, data is considered to be one of the most valuable resources in the world. Some data is viewed as strictly confidential, so it needs to be protected and secured. Therefore, data masking is becoming increasingly popular. Data anonymization is a concern for many global businesses, including the Libelle company, which provides numerous optimal algorithms for data anonymization in its Libelle **Data***Masking* product. Due to the fact that the aforementioned solution is implemented on-premise, the idea is to utilise current algorithms and transfer the masking functionality to the cloud. The main goal here is to enable serverless data anonymization, while the ultimate objective is to develop a simple-to-use solution that allows users to enter certain data, send it, and receive anonymized data back. This was accomplished through the use of a serverless architecture and the Function-as-a-Service (FaaS) model. According to the serverless concept, the architecture is handled by a third party, leaving developers to focus solely on the product's functionality and logic. FaaS is primarily concerned with event-driven computing paradigms in which the entire application code is only run in response to requests – in this case, sending data for anonymization. As a result, the organization's energy and resources are conserved, allowing management and developers to concentrate on the product and profit strategy. There are several technologies for implementing FaaS solutions, with AWS Lambda serving as the primary technology in this ‘serverless-data-masking’ scenario.

Keywords

AWS Lambda, serverless, Function-as-a-Service, cloud, data masking, anonymization

Content

1. Introduction

On-premise software installation and deployment allows the organisation the highest level of control and responsibility over the entire infrastructure. Although being physically close to the stored data, having direct access to the hardware and its upgrades, and having almost fixed costs are all advantages of on-prem computing, there are certain drawbacks. Computing time and costs can be high due to power consumption and cost, more hardware storage space is required, hardware failures are possible, and thus data loss. With all this, it is necessary to have adequate, specialized, and experienced people to manually manage and maintain the entire architecture and infrastructure of the business. With cloud and its services, many software functionalities, business operations and procedures may be enhanced and made easier to use. The ability to access and use a great number of application contents and resources from any location using the Internet only is the core incentive for utilizing the cloud services. Thus, by adopting cloud services, it is feasible to physically relocate a major portion of a company's operations while maintaining a stable and secure business environment. The delivery of various services through the Internet is known as cloud computing. These resources, technologies or services include data centres and storage, servers, networking, databases together with management and development tools, among other tools and applications. It is possible to operate a smaller or larger range of services depending on the needs of the organization, and cloud computing services are typically split into four concepts: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), and Function-as-a-Service (FaaS). The extent of services provided in each of these concepts or models varies. Briefly, IaaS provides high-level application programming interfaces (APIs), whereas PaaS provides a computing platform, which typically includes an operating system. Finally, SaaS allows software and applications to be installed and run on cloud servers, whereas FaaS, also known as serverless computing, is a concept in which programmes and applications do not need a dedicated machine to function and are deployed and executed on servers only when needed. Each of the models listed is appropriate for specific types of software, and it is up to businesses to determine how they will utilise the selected service.

This paper will concentrate on the process of data masking and its use in organisations from various business industries. Generally speaking, data masking, also known as data obfuscation, is the process of modifying sensitive data such that it is of no use to unauthorised intruders but valuable to software or authorised individuals [1]. Many companies provide data masking solutions in a variety of strategies and methods. The great majority of businesses provide on-premise software, but an increasing number of them are beginning to adopt cloud alternatives. The reason for this is the growing popularity of cloud computing, as well as the simplicity of use of such solutions and services.

Libelle AG delivers an on-premise data masking software, Libelle **Data***Masking* (LDM), to anonymize sensitive and personal data and operate with it responsibly [2]. Given the increasing popularity of cloud capabilities, it is reasonable to assume that the current data masking process could be enhanced and made more convenient. Considering that it is now required to install

LDM on client devices and understand the entire underlying infrastructure, there is surely a motive to improve the data anonymization process. With such a logic, the goal of this paper is to study and research market and existing data masking solutions in order to clarify whether developing one's own solution is essential.

To summarize, this paper aims to explain and demonstrate the entire process of researching and implementing the intended product functionality – data masking, using the best approach determined during the research.

The following sections give a brief overview of the study's subject and explain the general motivations for, and techniques used in data masking. Given that the subject is the functionality of a product made by the Libelle group, both the business operations of the company and the data masking product are explained. The current market for data masking solutions and software was researched and briefly described in order to identify the need for developing one's own solution. After determining the optimal method for developing a solution, the technologies used, as well as the reasons for using them, are thoroughly described. Finally, the serverless data masking use cases are listed, along with the specific methods and steps for implementing the solution.

2. Data masking

Data masking is a method of securing sensitive data by replacing the original value with a fictional but realistic equivalent. It is an umbrella word encompassing data anonymization, pseudonymization, scrubbing, redaction, or de-identification. Data masking is also known as data obfuscation [3]. Data masking procedures modify the values of data while keeping the same format. The goal is to create a version that cannot be interpreted or reverse engineered. Changes to the data can be accomplished by character shuffling, word or character substitution, and encryption. There are several forms of data that may be protected via masking, but the following are the most common [4]:

- PII: Personally Identifiable Information
- PHI: Protected Health Information
- PCI-DSS: Payment Card Industry Data Security Standard
- ITAR: Intellectual Property

2.1 Why is data masking important?

Data masking eliminates a number of significant risks, including data loss, data exfiltration, insider threats or account breach, and insecure interactions with third-party systems. It reduces the risks connected with cloud migration in terms of data and allows authorised users, such as testers and developers, to share data without exposing production data. What is more, data is rendered worthless to an attacker while retaining many of its underlying essential characteristics [5].

Table 1 - Common data masking and data security techniques

As illustrated in the graphic above, there are about 9 methods for masking and securing data.

1. The **encryption** method uses mathematical computations and algorithms to scramble data. It is most appropriate for securing data that must be restored to its original state, such as production data or data in motion. A hacker who possesses the proper keys can decrypt sensitive data and restore it to its original condition. There is no master key in data masking, therefore scrambled data cannot be restored to its original values.
2. **Tokenization** is a type of encryption that creates stateful or stateless tokens that can typically be re-identified.
3. **Scrambling** of letters or numbers does not effectively protect sensitive data.
4. **Shuffling** refers to shifting data among rows in the same column, which might be beneficial in some situations but does not ensure data protection.
5. **Null out or deletion** alters data properties and removes any usefulness from data.
6. **Variance** changes the data based on the provided ranges. It can be beneficial in some cases, such as when non-sensitive transactional data has to be secured for aggregation or analysis.

7. During the **substitution**, data is replaced with another value. The amount of complexity to perform might differ greatly, but when done correctly, it is the correct approach to mask the data.
8. The **redaction** method of data masking demands altering all characters to the same character. It's simple, although data loses business value [3].
9. According to the EU General Data Protection Regulation (GDPR), **pseudonymization** is any approach that assures data cannot be used for personal identity. It necessitates the elimination of direct identifiers and, preferably, the avoidance of numerous characteristics that, when combined, can be used to identify a person. Furthermore, encryption keys and other information that can be used to restore original data values should be kept separate and safe [5].

2.2 Types of data masking

Depending on particular use case, one may choose from a variety of data masking types. The most common are static and on-the-fly data masking. The types of data masking are briefly described below [6]:

- **Static data masking (SDM)** is typically applied on a backup of a production database. It alters data to make it appear accurate in order to correctly develop, test, and train - all without exposing the actual data information.
- **Dynamic data masking (DDM)** occurs dynamically at run time and streams data straight from a production system, eliminating the requirement for masked data to be kept in another database. Therefore, DDM is used in read-only situations to prevent masked data from being written back into the production system.
- **Deterministic data masking** implies replacing data in columns with the same value. For example, if one's databases have numerous tables with a first name field, there might be many tables with the first name. The masking will produce the same results every time.
- **On-the-fly data masking** happens when data is transferred from one environment to another, such as test or development. It is suitable for businesses that constantly deploy software or have complex integrations. Because it is difficult to maintain a continuous backup copy of masked data, this method will only deliver a subset of masked data when required.
- **Statistical data obfuscation** methods can mask production data that might contain a variety of statistical information. Differential privacy is a strategy for sharing information about patterns in a data set without exposing information about the data set's real individuals.

2.3 Data masking best practices

The process of determining the project scope, ensuring referential integrity, and securing the data masking algorithms are all aspects of best practises for the data masking process. Companies must understand which information should be secured, who has access to it, which programs consume the data, and where it is stored, both in production and non-production domains, in order to execute data masking efficiently. While this may appear to be a simple procedure on paper, due to the complexity of operations and various lines of business, it may need a significant amount of effort and must be designed as a distinct stage of the project. Referential integrity requires that each "form" of information originating from a business application must be masked using the same algorithm. When dealing with the same type of data, different data masking strategies and tools should be coordinated across the company. This will help to avoid problems later on when data has to be used across multiple business areas. It is vital to think about how to preserve the data-creation algorithms, as well as alternate data sets or dictionaries used to obfuscate the data. Because only authorised individuals should have access to the actual data, these algorithms must be treated with extreme caution. Someone who discovers which recurring masking algorithms are being employed can reverse engineer large blocks of classified information [5].

3. Libelle AG company

Libelle AG is a German company that provides IT solutions in database, SAP Basis, and other environments. The headquarters of Libelle AG are in Stuttgart, and the company has four subsidiaries: Schwelm in Germany, Paris in France, Atlanta in the United States, and Lipik in Croatia. Libelle was founded in 1994 and now operates in more than 15 countries with over 500 satisfied customers. Around 100 experts at the Libelle Group work tirelessly to position customer for automation in areas such as availability and disaster prevention, anonymization, SAP Basis operation, SAP monitoring, and SAP master data. The Libelle Group's technology partners include Amazon Web Services (AWS), Microsoft, SAP, NetApp, IBM, and Oracle, among others. Products offered by the Libelle company [7]:

- Libelle **Busines***Shadow*: a solution for disaster recovery and high availability, user can mirror databases, file systems, and application systems with a time delay
- Libelle **System***Copy*: at any time, non-production systems are automatically provided with fresh production data. It handles all aspects of system copy pre- and post-processing, whether on-premises, hybrid, or in the cloud
- Libelle **Data***Masking*: it has the ability to anonymize personal and sensitive data and work with it responsibly - whether in SAP® or another environment. During anonymization, realistic-looking and GDPR-compliant data replaces data in non-production systems
- Libelle **MasterData***ServiceSuite*: Six tools in one toolbox - this solution integrates everything needed for effective master data management. Simplify the work with master data. Whether for routine maintenance, migration projects, or quality assurance.
- Libelle **EDIMON**: an add-on for monitoring and managing IDocs across entire SAP® landscapes. The IDoc workflow is a critical aspect of incoming and outgoing orders, deliveries, and invoices, and is thus inextricably linked to a company's core business.
- Libelle **SABMON**: allows early detection and documentation of error situations and can initiate concrete measures on demand. This software solution enables automated monitoring based on a number of predefined and customizable checks.

3.1 SAP

Among others, the SAP environment is included in each of Libelle's software solutions. As a *silver partner* of Libelle, SAP is one of the world's leading producers of business process management software, developing solutions that enable effective data processing and information flow across organisations [8]. SAP was founded in 1972 in Walldorf, Germany, and now employs over 105 000 people worldwide. The company is the world's leading provider of enterprise resource planning (ERP) software. In addition to ERP software, the company sells database software and technology (particularly its own brands), cloud engineered systems, and other ERP software products such as human capital management (HCM) software, customer

relationship management (CRM) software (also known as customer experience), enterprise performance management (EPM) software, product lifecycle management (PLM) software, supplier relationship management (SRM) software, supply chain management (SC) software, and supply chain management (SC) software. SAP is the world's third-largest publicly traded software company by revenue, the largest non-American software company by revenue, and the second largest German company by market capitalization [9]. To put it more simply, SAP helps businesses of all sizes and industries run profitably, adapt continuously, and grow sustainably [8].

3.2 Libelle DataMasking – monolithic application

Libelle **Data***Masking* (LDM) is a data masking solution for seamlessly anonymizing production data. LDM complies with many legal frameworks throughout the world, including EU General Data Protection Regulation (GDPR), Gramm Leach Bliley Act (GLBA), Family Educational Rights and Privacy Act (FERPA), Personal Information Protection Law (PIPL), Health Insurance Portability and Accountability Act (HIPAA), and numerous others. Privacy, realistic testing, and simplicity are some of the benefits listed by Libelle. Anonymization protects against unauthorised data access or perhaps even data theft on the non-production systems. Also, LDM generates realistic-looking and logically valid data, allowing tests to be optimised. It is important to note that the product works with all common databases and may be run nightly with user-defined configurations.

LDM can effectively anonymise up to 200,000 data entries per second since it operates at the database level, although tables may also be anonymised parallelly if needed. This software offers over 40 anonymization algorithms along with a reference database with predefined target values. Nevertheless, if one wants to modify the data masking software to its particular needs, it can be done here as well. Libelle **Data***Masking* is a data masking solution that can access any common database and be applied to individual systems or entire system landscapes. Databases from various environments are identified and anonymised across systems. One of the most noticeable features is that LDM converts original production data into realistic-looking and logically valid data. Thus, without a clear personal reference, one may realistically train, test, and assess on non-production systems [2]. Libelle DataMasking is also offered as a so-called Machine Image by two cloud providers, Amazon AWS, and Microsoft Azure. To be more specific, all that is required is the launch of an instance with this image pre-packaged.

3.3 How Libelle DataMasking works

In general, the data masking process consists of four steps: inspection, preparation, anonymization, and review. Libelle **Data***Masking* verifies the IT infrastructure and whether the target system is operational during the check phase. It makes certain that the system is not in

production, which is very essential in SAP environments. This is done completely automatically and is a necessary step in order to properly utilise this data masking software. Errors or unrecognised fields can therefore be readily fixed. The pre-phase begins after the successful check. The supplied reference files are provided here, as are the keys for anonymization. Backup tables can be created here if necessary. This is additional assistance provided by Libelle data masking software prior to one's first anonymization. If this has not been done according to one's expectations, backup tables can be relied on. The third and most interesting phase is anonymization, which utilises both provided and personally configured anonymization algorithms. These algorithms receive data from the non-production target system and anonymize it based on the references provided. The end result is data that is both realistic and GDPR compliant. The fourth step, the post phase, verifies if the data consistency has been preserved, as a final measure to achieve the best possible outcome. Libelle **Data***Masking* provides a comprehensive final report with all relevant information to provide customers a complete overview of the anonymization process.

The technical data are presented in the table below.

EXTRACT OF THE ANONYMISATION ALGORITHMS	SUPPORTED DATABASES AND DATABASE CONNECTORS	SUPPORTED OPERATING SYSTEMS
Addresses	IBM DB2	AIX
Dates and times	LcSAP	HP-UX
E-mail addresses	MariaDB	Linux
Names	Microsoft SQL Server	MacOS
Numbers	MySQL	Solaris
Organisations	ODBC	Windows
Regions	Oracle	
Bank Accounts / Credit Cards SNN (Social Security Number)	PostgreSQL	
	SAP ASE	
	SAP MaxDB	
	SAP HANA	

Table 2 - Libelle DataMasking technical data

In addition to the technical data given above, LDM delivers the following SAP templates: SAP CRM, SAP ERP, SAP FI/CO, SAP HCM, SAP SD, and SAP SRM. Libelle DataMasking also supports CSV, JSON, and XML file formats [2].

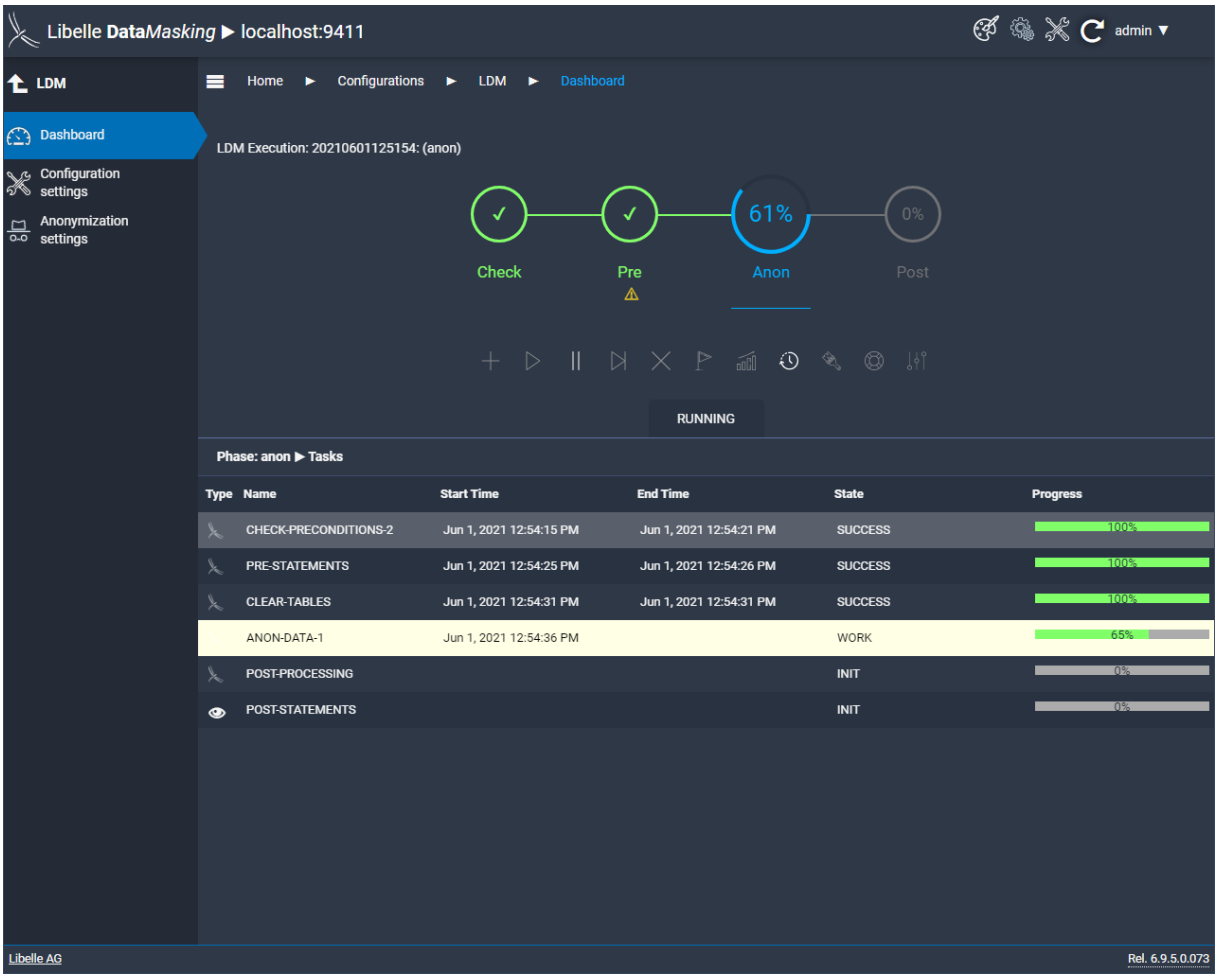

Image 1 - Screenshot of LDM during the anonymization step

Image 2 - Example of single value anonymization (first and last name)

4. Data anonymization solutions – current market state

A thorough market analysis and research is required to determine the purpose for developing one's own data masking software solution. The following paragraphs will analyse and evaluate the present state of the market for data anonymization cloud solutions.

4.1 DataMasque

DataMasque positions itself as the right toolset for meeting one’s compliance requirements through best practise data masking, all while being seamlessly integrated into one’s DevOps workflow - in the cloud or on-premise. Currently, Oracle, Microsoft SQL Server, and PostgreSQL are supported in this solution. DataMasque is compatible with the following deployment methods:

- As a Cohesity App available from the Cohesity marketplace.
- As a collection of Docker containers deployed using Docker Compose on Red Hat Enterprise Linux or Ubuntu.
- As a pre-configured product available from the AWS Marketplace.

DataMasque is accessible through a browser-based web interface that provides a centralised single-pane-of-glass view of your entire data masking environment. DataMasque also offers a robust set of REST API endpoints, making it simple to access its various features via automated processes [10]. Without getting into the precise details of the functionality, the way to use this product is very similar to Libelle **Data***Masking*.

4.2 Maskopy

Maskopy is a solution for copying and running obfuscation on production data across AWS accounts. In a transient instance, any sensitive information in the production data is obfuscated. The final obfuscated/masked snapshot is shared in the environments specified by the user. Maskopy is a self-service solution that allows users to obtain production data without requiring the involvement of multiple teams. It is fully automated and designed to easily integrate with CI/CD pipelines and other automation solutions via Amazon Simple Notification (SNS) or Amazon Simple Queue Service (SQS). Maskopy includes security controls like access management via IAM roles, caller identity authorization, and network access to transient resources managed via security groups. Maskopy is a tool-independent obfuscation solution. Teams can use any encryption tools or obfuscation frameworks that meet their requirements and bake them into a docker container. The container can then be brought to the Maskopy solution to mask the data [11]. Because the architecture of this solution is made up of several

different AWS services, the client must go through a few procedures before using the product. As prerequisites, one must have python3, pip, zip, aws cli, and docker installed for a local environment, while for the AWS environment, at least two AWS accounts are needed; source and staging account. The source account hosts an RDS instance within a Virtual Private Cloud (VPC) and Subnet that must be masked and copied to other accounts. On the other hand, the Lambda and Step functions are deployed and executed from the staging account. In this account, the final snapshot is created. This account must have a Virtual Private Cloud (VPC), Public and Private subnets, Route tables for internet access, and an S3 bucket to stage Lambda code [12].

To begin any data masking process, a number of other steps must be taken. This alone makes implementing the above solution extremely difficult. One of the reasons for this significantly different implementation of data anonymization compared to the Libelle and DataMasque products is the strict connection to AWS.

4.3 Amnesia

The Amnesia anonymization tool is Java and JavaScript software that should be used locally to anonymize personal and sensitive data. Users load a file containing personal data (original data) into Amnesia, and Amnesia transforms it into an anonymous dataset, which can then be stored locally. The transformation is guided by user selections and ensures anonymity for the resulting dataset. Currently, Amnesia supports k-anonymity and km-anonymity guarantees [13]. The entire data masking process consists of five simple steps, which are briefly described below:

1. Importing the original data - the original dataset must be in the form of a simple text file with any delimiter. For instance, a character separating different fields in a record.
2. Creating the generalization hierarchies - Amnesia allows the user to semi-automatically create rules for generalising values, save and reuse them, or import them from other sources. For example, rules for replacing card number or zip code with more abstract values.
3. Starting the anonymization - the user can choose the best method for this problem (for example, k-anonymity or km-anonymity) and link the hierarchies to the records' respective attributes. The anonymization procedure is initiated after that.
4. Choosing the solution user likes - Amnesia depicts several possible solutions, visualises the value distribution, and provides statistics on the data quality in the anonymous dataset. A solution tailored to the user's needs can be inspected and applied with a few simple clicks.
5. Saving anonymised data locally or to Zenodo, a general-purpose open repository.

Amnesia is available as an online demo tool where one can simply upload files and follow those 5 steps stated above and have the anonymised data back [14]. The infrastructure of the said web interface solution is unknown, i.e., the sources do not specify how the backend operates.

There are a plenty of other data masking solutions on the market today. Almost every cloud provider offers some sort of data masking solution, and the majority of them may be constrained by their own resources or technologies. For example, Google Cloud's BigQuery, a serverless, cost-effective and multicloud data warehouse, supports dynamic data masking at a column level. Data masking can be used to selectively obscure column data for groups of users while allowing access to the column [15]. What is more, only Oracle databases can be masked and secured in Oracle Data Safe, which is available in Oracle Cloud [16].

After researching the market for data masking solutions, it is safe to conclude that almost all solutions require a greater number of actions and, in some cases, comprehensive knowledge in order to anonymize data. The findings of the market research, namely the existing data masking solutions, are unsatisfactory for the purposes of this paper. This implies that the market lacks a simple solution in which the user can easily enter the desired data, specify some preferences, and receive anonymized data back.

One of the great examples of how a service can easily do its job is Microsoft Translator. Evidently, the listed product does not anonymize the data, but instead demonstrates the ease of the work, in this case the translation of the entered text, can be completed.

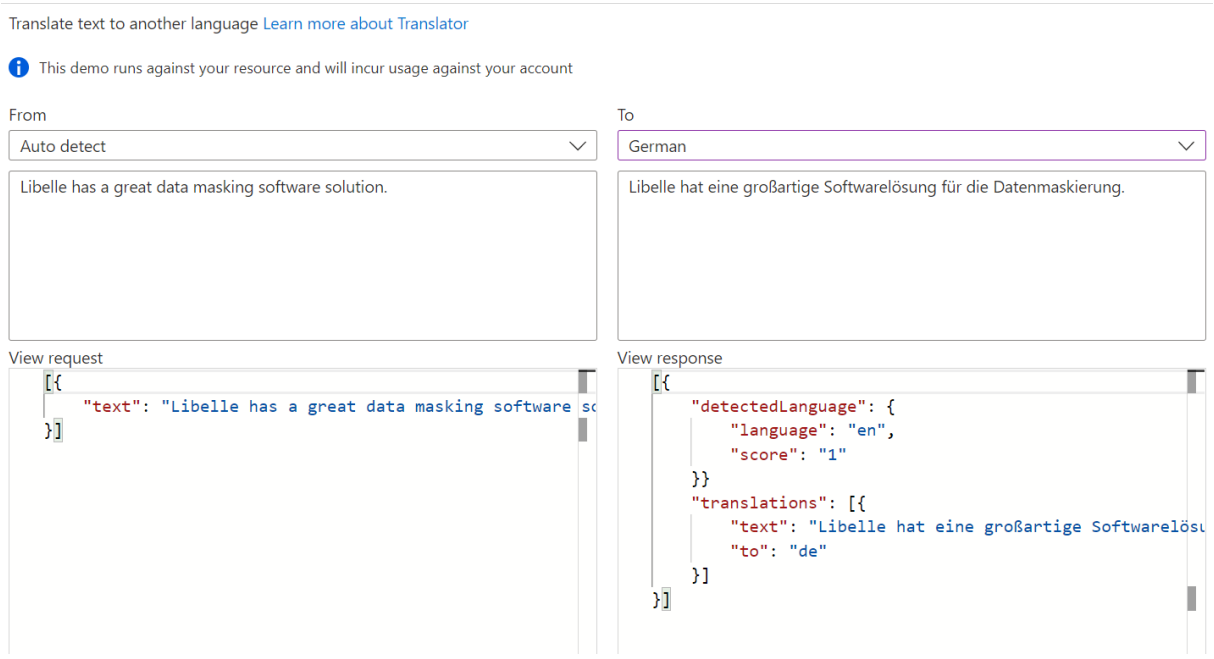

Image 3 - Microsoft Translator service task example

After learning about this solution, the idea of anonymizing data in a similar manner - by entering data and receiving almost instant feedback result - came up.

5. A comparison of monolithic and microservices architecture

Monolithic applications are single-tiered, which means they combine multiple components into a single large application. As a result, they frequently have large codebases that can be difficult to manage over time. Besides that, if one programme component needs to be updated, other elements may need to be rewritten as well, and the entire application must be recompiled and tested. The process can be time-consuming and may limit software development teams' agility and speed. Despite these issues, the method is still used because it has some advantages. Furthermore, because many early applications were created as monolithic software, the approach cannot be completely ignored when those applications are still in use and require updates. Monolithic applications are made up of multiple components that are linked together to form one large application. These elements may include the following characteristics [17]:

- **Authorization** of users for granting them access to the application
- **Presentation** for responding to Hypertext Transfer Protocol (HTTP) requests
- **Logical reasoning** for the underlying business logic, functionality and features of the application
- **The database layer** for connecting data objects to the application's database
- **Integration of applications** for the integration of the application with other data sources
- Some have **notification module** for allowing users to control and send automated emails

Monolithic architecture benefits [18]:

- Simplicity of development process
- Simplicity of testing the application
- Simple application deployment
- Simplicity while scaling horizontally

Monolithic architecture drawbacks [18]:

- Limitation of size and complexity
- Apps can be too large and complex to make fast and correct modifications
- App size can be an issue during the start-up time
- On each update the application must be redeployed
- Difficulty of continuous deployment
- Difficulty with scaling
- Reliability problem (for example, one bug can affect entire application)
- Time and cost expensive to adopt new technologies

Microservices architecture, on the other hand, has the idea of splitting the application into a set of smaller, interconnected services rather than building a single monolithic application. Each microservice is a small application with its own hexagonal architecture that includes business logic as well as various adapters. Some microservices would expose a REST, RPC, or message-based API, while the majority of services would consume APIs provided by other services. Other microservices may include a web interface [18]. There are around 3 principles of microservices: single responsibility, built around business capabilities, and design for failure. The first principle specifies that a single unit, such as a class, method, or microservice, should have only one responsibility. Each microservice must be responsible for a single task and provide a single function. The second principle states that depending on the business requirements, each microservice can use a different technology. According to the third principle, microservices must be designed with failure scenarios in mind. Microservices must take advantage of this architecture, and the failure of one microservice should not affect the entire system; all other functionalities must remain available to the user [19].

Microservices architecture benefits [19]:

- Easy to manage because of relatively small size
- In the case of one microservice update, only that one needs to be redeployed
- Start-up time is faster than with monolithic applications, microservices are self-contained and deployed independently
- Easy for new developer to onboard the project as he will not work on the entire system
- Microservices support horizontal scaling
- Different technologies can be used in each microservice
- If a bug happens, it doesn't affect other microservices, the entire system remains intact

Microservices architecture drawbacks [19]:

- As a distributed system, complexity increases with the number of microservices
- Complexity of deploying independent microservices
- Microservices are expensive in terms of network usage (they must interact with one another; it can cause network latency)
- Because of the inter-services communication over the network, microservices are less secure than monolithic applications
- Debugging can be difficult

Microservices architecture is similar to serverless architecture in some ways. The primary distinction is that microservices are a method of designing applications, whereas serverless is a method of running applications (or a part of an application). The serverless model is, in principle, easier to implement with microservices architecture. If compared, the difference between a microservice and a serverless function is that microservices can be long-term, continuous, and perform more than one function, whereas serverless functions are never continuous, require a trigger event, and cannot include more than one function [20].

6. Determining the best model approach

As previously stated in the paper, there are several cloud computing concepts. Some of the ‘XaaS’ models, an abbreviation for any ‘as-a-service’ concept, will be shortly summarized.

Infrastructure-as-a-Service (IaaS)

IaaS is the most basic form of XaaS. It has a lot of power but requires a lot of setup. IaaS provides a virtual machine that must be maintained. The difference between IaaS and a physical server room is that the user does not need to purchase any physical computers and can have servers in different parts of the world. However, when compared to other XaaS, IaaS is more difficult to maintain and requires a skilled DevOps engineer to configure the virtual machines so that they operate efficiently and securely [21]. IaaS allows users to purchase only the components they require and scale them up or down as needed. IaaS is a very cost-effective option because it has low overhead and no maintenance costs. IaaS could be used as a quick and flexible way to set up and tear down development and testing environments [22].

Public cloud providers, such as AWS, Microsoft Azure, and Google Cloud, are example of IaaS.

Platform-as-a-Service (PaaS)

PaaS is a step beyond full on-premise infrastructure management. It is the process by which a provider hosts the hardware and software on its own infrastructure and then delivers this platform to the user via an internet connection as an integrated solution, solution stack, or service [22]. The main disadvantage is that it is not very flexible because custom system dependencies (e.g., from apt-get) cannot be installed and only one of the available technologies can be used. So, if one develops their own programming language that does not run on any available platforms but is compiled directly to bytecode, they will be unable to use PaaS [21].

PaaS examples include AWS Elastic Beanstalk, Heroku, Red Hat OpenShift, and Google App Engine.

Software-as-a-Service (SaaS)

SaaS is also referred to as "on-demand software" [23] or “cloud application services” and is typically accessed via a thin client through a web browser. It is the most comprehensive type of cloud computing service, delivering an entire application managed by a provider through a web browser. The provider handles software updates, bug fixes, and general software maintenance, and the user connects to the app via a dashboard or API. There is no software installation on individual machines, and group access to the programme is smoother and more reliable. SaaS is a great option for small businesses that don't have the staff or bandwidth to

handle software installation and updates, as well as for applications that don't require a lot of customization or will only be used on a regular basis. What SaaS saves users in terms of time and maintenance may cost them in terms of control, security, and performance, so it's critical to choose a trustworthy provider [22].

Some examples of SaaS include Dropbox, Salesforce, Google Apps, and Red Hat Insights.

Function-as-a-Service (FaaS)

FaaS is even more straightforward than PaaS. It is an event-based architecture, as the name implies, because it is based on functions that can be triggered by a given event. The simplicity is so great that it's referred to as a serverless architecture. The developer only needs to write a function and does not need to think about deployment, server resources, scalability, etc. It's because FaaS is self-scaling. As a result, billing is based on actual consumption rather than declared resource needs. Supported technologies are one of the most significant drawbacks of FaaS. FaaS solutions have even fewer available technologies than PaaS; for example, a limited number of programming languages [23]. With FaaS there is also less system control as a third party manages part of the infrastructure. Together with that, it can be difficult to integrate FaaS code into a local testing environment, making thorough application testing more difficult [24]. Because FaaS can handle HTTP requests, it allows for the implementation of any algorithm (Turing completeness). After that, another HTTP request can be sent, and time and queuing events can be handled. Functions and microservices can be combined to connect FaaS and PaaS, but it is also possible to build the entire backend on FaaS, with some limits on technologies used on the backend side [21].

FaaS examples include AWS Lambda, Google Cloud Functions, and Azure Functions.

There are a few more 'as-a-service' models that are not yet fully established, such as Container-as-a-Service (CaaS), Database-as-a-Service (DBaaS), and Data-as-a-Service (DaaS).

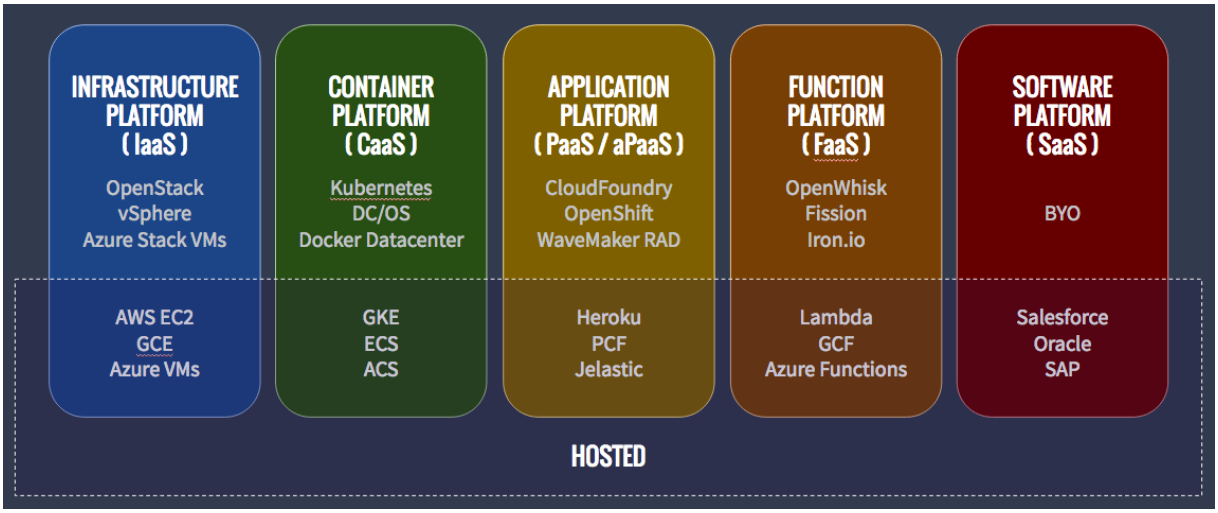

Image 4 - XaaS overview with examples and available services [23]

After this brief summary, the question still arises: which method is best for implementing the simplest possible data masking process?

As stated in the initial concept, the end goal is to have an interface where the user can enter data and have it back anonymized. This means that the user should enter his data somewhere on the web interface and having to send it should trigger the backend to perform the anonymization process.

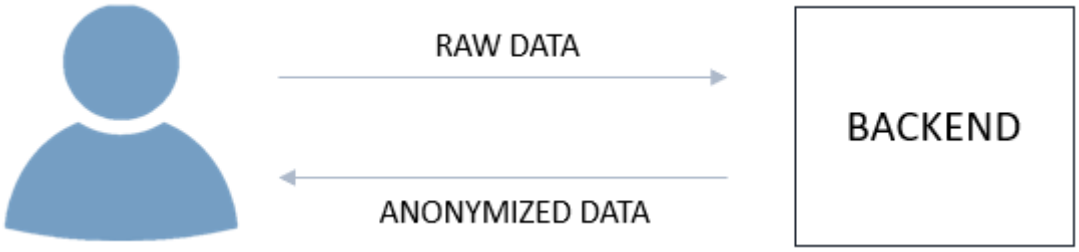

Image 5 - Simple display of the required functionality

Control over almost anything except the functionality itself is not required during the implementation of the data masking process. In this case, when implementing the solution, the focus should be solely on the functionality, i.e., only on the code. Based on previously described concepts, IaaS and SaaS are the first to be ruled out. This ultimately leaves two options: PaaS and FaaS.

Scalability is the primary operational distinction between FaaS and PaaS. Most PaaS applications require consideration of scale, whereas with a FaaS application this is completely transparent. Even if a developer configures the PaaS application to auto-scale, he will not be doing so at the level of individual requests (unless one has a very specifically shaped traffic profile), so a FaaS application is far more cost-effective [25].

With this knowledge, the most efficient and cost-effective solution would be to implement the data masking solution as a FaaS model. This would imply that one function should handle the majority of the work, ensuring that it only executes when requested. More specifically, the data sent by the user should trigger the developed function, and only then should a charge be applied. Event-Driven Architecture, as defined by AWS, uses events to trigger and communicate between decoupled services and is common in modern applications built with microservices. An event is a state change or an update, such as an item being added to a shopping cart on an e-commerce website [26]. In the context of this paper, the event could be the user's data.

To implement the above solution in the desired manner - in a FaaS environment - it is necessary to thoroughly examine the above architecture.

7. Serverless architecture

Cloud computing refers to the use of software or services over an internet connection, with the processing taking place in a separate location yet being supplied via the internet. The word "as-a-Service" is present in each of the cloud computing models, and it refers to the service provided by a third party. This implies that the company won't have to deal with some of the processes involved in on-premise software installation, depending on the adopted cloud computing model. This paper requires a description of the Function-as-a-Service (FaaS) approach, widely recognized as serverless architecture. In this model, clients and developers may focus entirely on the application and business concepts and strategy while the backend services are taken care of by the provider of serverless computing services. Storage, networking, servers, virtualization, operating systems, middleware, runtime, and data are all managed by given backend services. As a result, the company and developers are solely accountable for the application program's logic and code. Another great benefit of serverless computing is that it is charged on a per-use basis, based on activity and computation requests. The columns below represent some general pros and cons of the serverless architecture [27]:

PROS

1. Scaling is managed automatically by third-party software, so developers don't have to worry about it.
2. The development process is faster since less configuration is required.
3. It is triggerable - there are several triggers available in third-party systems. For example, with AWS, it is quite simple to run a Lambda function as a cron job, or to execute a function when a new file is added to S3.
4. There are numerous programming languages provided, and each function may be written in a language appropriate for it.
5. Pay as-you-go model — you only pay for the time consumed during executions.
6. There is no infrastructure or operating system to maintain.

CONS

1. There is no state - Since the organization does not have a server in a serverless architecture, it cannot preserve its global state. This problem may be handled by including an in-memory database into the infrastructure, such as Redis or Memcached, to preserve the global state and allowing each execution to get the state from there.
2. Maximum run time - each available serverless service has a limited time of execution (for example, AWS Lambda has a maximum timeout of 15 minutes).
3. More complexity is required for testing - It might be difficult to integrate FaaS code into a local testing environment, making application testing more time consuming.
4. Less system control - Having a third party administer part of the infrastructure makes understanding the complete system harder and complicates troubleshooting.

7.1 Options for implementing FaaS

Today, the majority of cloud providers offer services for developing serverless architecture applications. The best-known services today are Microsoft Azure Functions, Google Cloud Functions, IBM Cloud Functions, OpenFaaS, and others. Each of the mentioned services operates on a very similar basis, by executing function code according to the trigger event. The table below compares the focus of different FaaS providers [28]:

Do it yourself	Configurability	Ease-of-use	Edge computing
Open-FaaS	AWS Lambda	Vercel Functions	Cloudflare Workers
Fn project	Google Functions	Netlify Functions	Edge Engine
Fission	Azure Functions	Fly.io	Fly.io
Open-Whisk	IBM Cloud Functions		Lambda @Edge
Kubeless	Alibaba Functions		Vercel Functions + Edge Cache
Fly.io	Oracle Functions		

Table 3 - Categorizing FaaS providers based on their primary focus

Since AWS Lambda is one of the most popular FaaS implementation options with a wide range of possibilities, it will be used to implement the data masking solution. Its description will be covered in greater detail.

7.2 AWS Lambda

AWS Lambda is an event-driven computing service that makes it possible to execute code without having to provision or manage servers. Lambda executes its code on a high-availability compute infrastructure and handles all compute resource administration, including server and OS maintenance, capacity provisioning and automated scaling, including logging. One can use Lambda to run code for almost any form of application or backend service. All that has to be done is to provide the code in one of the programming languages supported by Lambda [29]. The code is organized into Lambda functions, where Lambda executes instances of the function to handle events. One may either invoke the function directly using the Lambda Application Programming Interface or setup an AWS service or resource to do so [30]. The procedure of the invoking function will be explained in detail further on in the paper. A developer is exclusively responsible for his code while deploying Lambda. Lambda handles the compute fleet, which provides the code running with a balance of memory, CPU, network, and other resources. Since Lambda maintains these resources, developers are unable to log in to compute instances or modify the operating system on given runtimes. Lambda handles operational and administrative tasks for the organisation, such as controlling capacity, monitoring, and recording Lambda functions [29].

According to Amazon, the features and capabilities listed below assist with the creation of scalable, secure, and easily extensible Lambda applications [29]:

- **Concurrency and scaling controls**: These two parameters allow users to fine-tune the scalability and responsiveness of the production applications.
- **Functions defined as container images**: the desired container image tooling, processes, and dependencies can be used to develop, test, and deploy Lambda functions.
- **Code signing**: Code signing for Lambda offers trust and integrity controls, allowing organizations to ensure that only unmodified code published by authorised developers is deployed in the Lambda services.
- **Lambda extension** can be used to extend Lambda functions. Extensions can be used, for example, to make it easier to integrate Lambda with one's favourite monitoring tools, confidentiality, and other.
- **Function blueprint** contain sample code for demonstration on how to use Lambda in conjunction with other Amazon services or third-party apps.
- **Database access**: A database proxy manages a pool of database connections and forwards function queries. This allows the Lambda function to achieve high concurrency without exhausting connections of the database.
- **File systems access**: A function can be designed to mount an Amazon Elastic File System (Amazon EFS) to a local directory. The function code can safely and concurrently access and modify shared resources using Amazon EFS.

Lambda can be accessed, or more precisely, created, invoked, and managed via several interfaces. One of the most common is the AWS Management Console, which provides a web interface for developers to access their functions. Second option includes providing commands

for a wide range of AWS services via the AWS Command Line Interface (AWS CLI). Other types of access can be granted using AWS Software Development Kits (SDKs), AWS CloudFormation, and AWS Serverless Application Model (AWS SAM). SDKs offer language-specific Application Programming Interfaces and manage multiple connection details, such as error handling or retry of a request. CloudFormation allows developers to create templates that define Lambda applications, whereas AWS SAM provides templates and a command line interface to configure and manage serverless apps [31].

7.2.1 Pricing

Creation of the Lambda functions is not being charged. There are fees for running a function and transferring data between Lambda function and other available Amazon services [32].

AWS Lambda Pricing

Region: Europe (Frankfurt)

Architecture	Duration	Requests
x86 Price		
First 6 Billion GB-seconds / month	$0.0000166667 for every GB-second	$0.20 per 1M requests
Next 9 Billion GB-seconds / month	$0.000015 for every GB-second	$0.20 per 1M requests
Over 15 Billion GB-seconds / month	$0.0000133334 for every GB-second	$0.20 per 1M requests
Arm Price		
First 7.5 Billion GB-seconds / month	$0.0000133334 for every GB-second	$0.20 per 1M requests
Next 11.25 Billion GB-seconds / month	$0.0000120001 for every GB-second	$0.20 per 1M requests
Over 18.75 Billion GB-seconds / month	$0.0000106667 for every GB-second	$0.20 per 1M requests

Image 6 - AWS Lambda pricing example for Europe [33]

The cost of ephemeral storage is determined by the amount of ephemeral storage assigned to the Lambda function and the duration of function execution in milliseconds. Any additional storage space between 512 MB and 10,240 MB can be assigned to the function in 1 MB increments. Ephemeral storage can be configured to run on both the x86 and Arm architectures. Each Lambda function has access to 512 MB of ephemeral storage at no extra cost, while only additional ephemeral storage is being charged.

Provisioned concurrency can be enabled for the Lambda functions to give developers more control over the performance of the serverless application. Provisioned Concurrency keeps Lambda functions initialised and ready to respond in a few milliseconds when enabled. Users are charged for the amount of configured concurrency as well as the length of time a user configures it for. Excessive function execution will be charged if the function exceeds the configured concurrency. Provisioned Concurrency is calculated by rounding up to the nearest five minutes from the time the function is enabled until it is disabled.

Data transferred into and out of AWS Lambda functions from regions other than the one in which the function was executed will be charged at the Amazon EC2 data transfer rates listed in the Lambda documentation's "Data transfer" section. Additional fees will apply when using Amazon Virtual Private Cloud (VPC) or VPC peering with AWS Lambda functions. Extra costs could be charged if the Lambda function uses other Amazon services or transfers data. For instance, if the Lambda function reads and writes data to or from Amazon S3, the company will be charged for the read and write requests as well as the data saved in AWS Simple Storage Service [33].

7.2.2 Lambda foundations

To use Amazon services, including AWS Lambda, the user must have an AWS account. With that, one can begin implementing his own Lambda functions. The list below shows the languages that Lambda supports, as well as the tools and options that can be used with them [34]:

- **Node.js**: Lambda console, Visual Studio with IDE plugin, user's own environment for the authorization
- **Java**: Eclipse together with the Amazon Toolkit for Eclipse, IntelliJ with the Amazon Toolkit for JetBrains, user's own environment for the authorization
- **C#**: Visual Studio with IDE plugin, .Net Core, user's own environment for the authorization
- **Python**: Lambda console, PyCharm with the AWS Toolkit for JetBrains, user's own environment for the authorization
- **Ruby**: Lambda console, user's own environment for the authorization
- **Go**: user's own environment for the authorization
- **PowerShell**: user's own environment for the authorization, PowerShell Core 6.0, .Net Core 3.1 SDK, AWSLambdaPSCore module

To process events, Lambda executes instances of the function. The function can be invoked directly using the Lambda API, or it can be configured to be invoked by an AWS service or resource. A function is a resource that can be utilized to run Lambda code. A function contains code that processes events that are passed into it or that other Amazon services send to it [35].

For a Lambda function to run, there must be a trigger. A resource or configuration that activates a Lambda function is known as a trigger. Amazon services that can be configured to invoke a function and event source mappings are examples of triggers [36]. An event source mapping is a Lambda resource that reads items from a stream or queue and calls a function. A Lambda function processes an event, which is a JSON-formatted document. The event is converted to an object by the runtime and passed to the function code. When a function is called, the user controls the structure and content of the event [37]. Another Lambda concept is the execution environment. An execution environment provides the Lambda function with a secure and isolated runtime environment. An execution environment oversees the processes and resources required to carry out the function. The execution environment supports the function's lifecycle as well as any extensions associated with it [38].

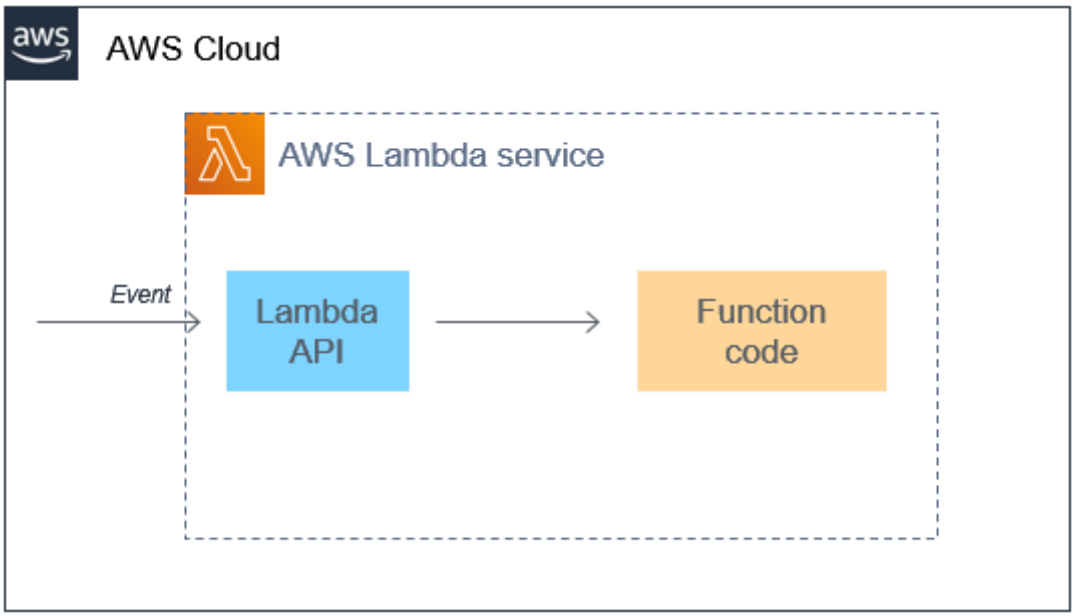

Image 7 - How Lambda fits into the event-driven paradigm [39]

The type of computer processor that Lambda uses to run the function is determined by the instruction set architecture. Lambda supports the following instruction set architectures [40]:

- arm64 – 64-bit ARM architecture
- x86_64 – 64-bit x86 architecture

A deployment package is used to deploy the Lambda function code. Lambda accepts two kinds of deployment packages [41]:

- The function code and its dependencies are contained in a.zip file archive. The function's operating system and runtime are provided by Lambda.
- A container image conforming to the Open Container Initiative (OCI) specification. The image is updated with the user's function code and dependencies. The operating system and a Lambda runtime must also be included by the user.

The runtime provides a language-specific environment that executes in an environment. The runtime communicates between Lambda and the function by relaying invocation events, context information, and responses. One can use the runtimes provided by Lambda or create his or her own. If a user packages his code as a.zip file archive, he must configure the function to use the appropriate runtime for his programming language. When creating a container image, one includes the runtime [42].

One more important concept of the Lambda is a layer. A Lambda layer is a.zip file archive that can include additional code or content. Libraries, a custom runtime, data, or configuration files can all be found in a layer. Layers make it simple to package libraries and other dependencies that users can use with Lambda functions. Using layers reduces the size of uploaded deployment archives and speeds up code deployment. Layers also encourage code sharing and the separation of responsibilities, allowing for faster iteration when writing business logic. Each function can have up to five layers. Layers are included in the standard Lambda deployment size quotas. When a layer is included in a function, its contents are extracted to the execution environment's '/opt' directory. The layers that the user creates are by default private to his AWS account. A layer can be shared with other accounts or made public by the user. If the functions consume a layer published by another account, the user's functions may continue to use the layer version after it has been deleted or his permission to access the layer has been revoked. A deleted layer version, however, cannot be used to create new functions or update existing ones. Layers are not used by functions deployed as a container image. Instead, when building the image, the user should package his preferred runtime, libraries, and other dependencies into the container image [43].

Lambda extensions allow the user to enhance his functions. Users can, for example, use extensions to integrate their functions with their preferred monitoring, observability, security, and governance tools. The user can select from a wide range of tools provided by AWS Lambda Partners, or he can create his own Lambda extensions. Internal extensions run within the runtime process and follow the same lifecycle as the runtime. In the execution environment, an external extension runs as a separate process. The external extension is started before the function is called, runs in parallel with the function's runtime, and continues to run after the function is called [44]. Concurrency is the number of requests served by the function at any given time. When the function is called, Lambda creates an instance of it to handle the event. When the function code has completed its execution, it is ready to handle another request. When the function is called again while a request is being processed, another instance is created, increasing the function's concurrency. Concurrency is subject to AWS Region-level quotas. Individual functions can be configured to limit their concurrency or to allow them to reach a specific level of concurrency [45]. Finally, an AWS resource to which Lambda can send events from an asynchronous invocation is referred to as a destination. The user can specify a destination for events that fail to process. Some services also provide a destination for successfully processed events [46].

7.2.3 Lambda features

Lambda offers a management console as well as an Application Programming Interface for managing and invoking functions. It offers runtimes that support a standard set of features, allowing users to easily switch between languages and frameworks based on their needs. Users can also create versions, aliases, layers, and custom runtimes in addition to functions.

Scaling

Lambda manages the code's infrastructure and scales automatically in response to incoming requests. Lambda scales up by running additional instances when the function is invoked faster than a single instance of the function can process events. Inactive instances are frozen or stopped when traffic stops. The user is only charged for the period that his Lambda function is initialising or processing some specified events. [47].

Concurrency controls

To make sure that production applications are highly responsive and available, concurrency settings should be used. Reserved concurrency is used to prevent a function from using too much concurrency and to reserve a portion of a user's account's available concurrency for a function. The pool of available concurrency is divided into subsets by reserved concurrency. A reserved concurrency function only draws from its own pool of concurrency. Using provisioned concurrency allows functions to scale without experiencing latency fluctuations. Provisioned concurrency allows users to prepare instances of functions for the initialization and keep them running all the time s for functions that take a long time to initialise or that require extremely low latency for all invocations. Lambda integrates with Application Auto Scaling to support provisioned concurrency autoscaling based on utilisation [48].

Function URLs

Amazon Lambda has built-in support for HTTP(S) endpoints via function URLs. Using function URLs, the user can direct the Lambda function to a specific HTTP endpoint. After configuring the function URL, the user can use it to invoke the function using a web browser, curl, Postman, or any HTTP client. A function URL can be added to an existing Lambda function or used to build a different Lambda function [49].

Asynchronous invocation

When a user calls a function, he has the option of doing so synchronously or asynchronously. The user waits for the function to process the event and return a response when using synchronous invocation. With asynchronous invocation, Lambda queues the event for processing and immediately returns a response. Lambda handles retries for asynchronous invocations if the function returns an error or is throttled. The user can customise this behaviour by configuring error handling settings on a function, version, or alias. User can also tell Lambda to send failed events to a dead-letter queue or to send a record of any invocation to a destination [50].

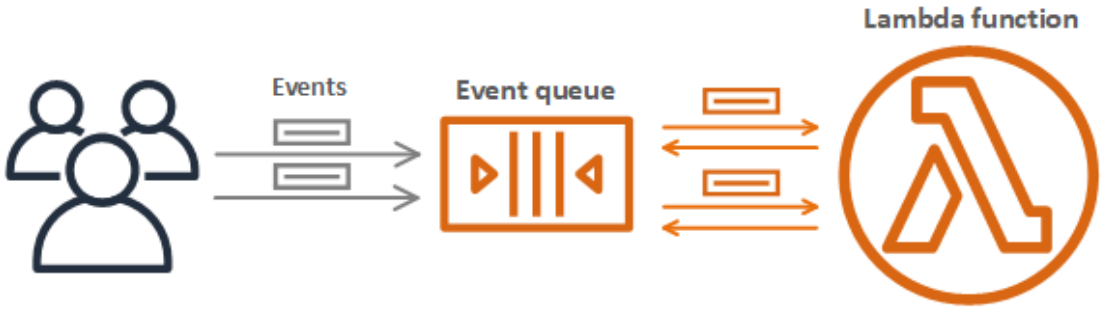

Image 8 - Asynchronous invocation overview [50]

Event source mappings

An event source mapping can be used to process items from a stream or queue. An event source mapping is a Lambda resource that reads items from an Amazon Simple Queue Service queue, an Amazon Kinesis stream, or an Amazon DynamoDB stream and sends them to the function in batches. Each event handled by the user's function may contain hundreds or thousands of items. If the return of the function execution is an error, event source mappings keep a local queue of unprocessed items and handle retries. An event source mapping can be configured to customise batching behaviour and error handling, or to send a record of items that fail processing to a destination [51].

Destinations

A destination is an Amazon resource that receives function invocation records. Users can configure Lambda to send invocation records to a queue, topic, function, or event bus for asynchronous invocation. The user can specify separate destinations for successful invocations and failed processing events. The invocation record contains information about the event, the function's response, and the reason for sending the record. Users can configure Lambda to send a record of batches that failed processing to a queue or topic for event source mappings that read from streams. A failure record for an event source mapping contains batch metadata and points to items in the stream [52].

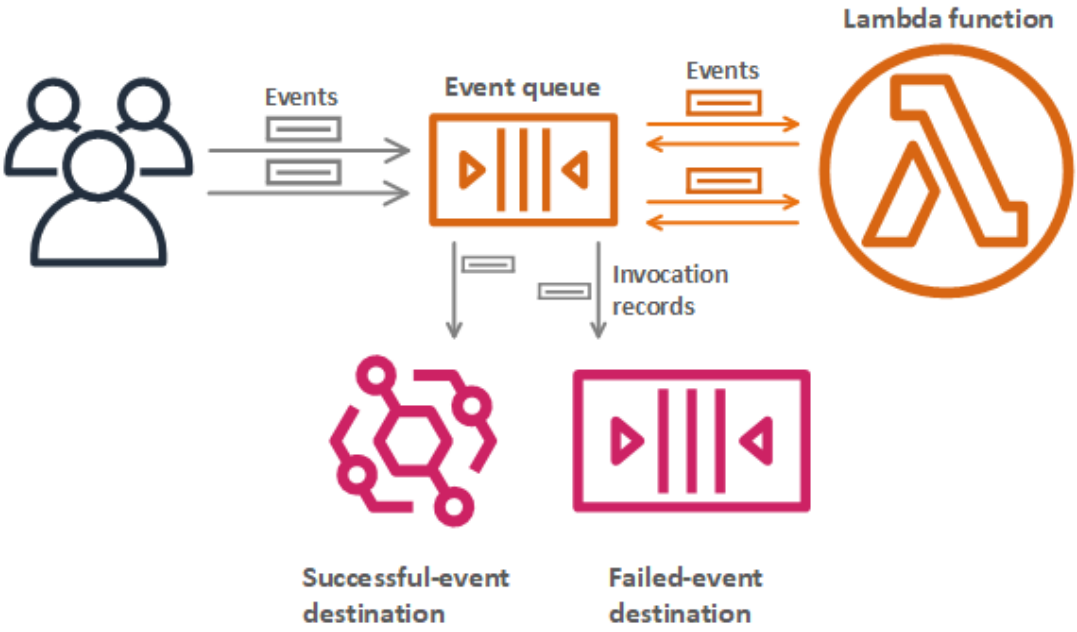

Image 9 - Destinations for Asynchronous invocation [52]

Function blueprints

When a user creates a function in the Lambda console, he has the option of starting from scratch, using a blueprint, deploying an application from the AWS Serverless Application Repository, or using a container image. A blueprint contains sample code that demonstrates how to use Lambda in conjunction with an AWS service or a popular third-party application. Sample code and function configuration presets for the Node.js and Python runtimes are included in blueprints [53].

Testing and deployment tools

Lambda allows users to deploy code as is or as container images. AWS and other tools such as the Docker CLI provide a rich tool ecosystem for handling authorization, build, and deploy process of Lambda functions. The command line tools AWS CLI and SAM CLI are used to manage Lambda application stacks. The AWS CLI supports higher-level commands that simplify tasks like uploading deployment packages and updating templates, in addition to commands for managing application stacks with the AWS CloudFormation API. The AWS SAM CLI adds functionality such as template validation, local testing, and integration with CI/CD systems [54].

Application templates

The Lambda console can be used to create an application with a continuous delivery pipeline. In the Lambda console, application templates include code for one or more functions, an application template defining functions and supporting AWS resources, and an infrastructure template defining an AWS CodePipeline pipeline. The pipeline includes build and deploy stages that are activated whenever user pushes changes to the included Git repository. The application templates are distributed under the MIT No Attribution licence and are only available in the Lambda console [55].

7.2.4 Lambda function handler in Python

The Lambda function handler is the method in the function code responsible for event processing. Lambda executes the handler method when the function is called. When the handler exits or returns a response, it becomes ready to deal with another event [56]. When writing a function handler in Python, the user can use the following general syntax:

```
def handler_name(event, context):
    ...
    return some_value
```

Image 10 - Basic Lambda function syntax [56]

The name of the Lambda function handler specified when the user creates a Lambda function is derived from:

- The name of the file containing the Lambda handler function
- The Python handler function's name

A function handler can have any name; however, the Lambda console's default name is '*lambda_function.lambda_handler'*. The name of this function handler reflects the name of the function *(lambda_handler*) and the file in which the handler code is stored (*lambda_function.py*) [57].

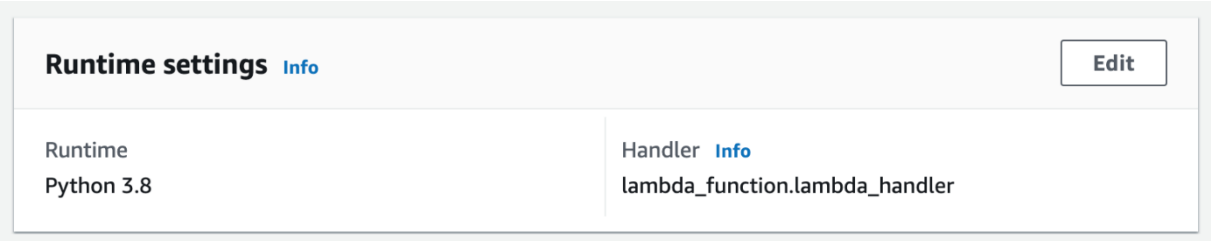

Image 11 - Lambda runtime settings pane [57]

When Lambda calls the function handler, the Lambda runtime passes two arguments to it:

1. **Event**

An event object is the first function argument. A Lambda function processes an event, which is a JSON-formatted document. The event is converted to an object by the Lambda runtime and passed to the function code. It is most commonly of the Python dict type, but it can also be a list, str, int, or float, or NoneType. The event object contains information from the service that was invoked. The structure and contents of an event are determined by the user when she invokes a function. The event structure is defined when an AWS service invokes the function. See Using AWS Lambda with Other Services for more information on events from AWS services.

2. **Context**

The context object is the second argument. Lambda passes a context object to the function at runtime. This object contains methods and properties that provide information about the function, invocation, and runtime environment [58].

A handler can optionally return a value. What happens to the returned value is determined by the type of invocation and the service that called the function. Lambda, for example, can return the result of a Python function call, an error, or null [59].

The official AWS documentation shows the lambda handler function example, which uses the python3.8 Lambda runtime. The function accepts first and last name input from the user and returns a message containing data from the event it received as input [60].

```
def lambda_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'], event['last_name'])
    return {
        'message' : message
    }
```

Image 12 - Function called lambda_handler [60]

```
{
    "first_name": "John",
    "last_name": "Smith"
}
```

Image 13 - Event data to invoke the function [60]

Image 14 - Function response of the event data passed as input [60]

7.2.5 General use cases

AWS Lambda has over 100,000 monthly active users, does billions of monthly executions, and is available in 22 worldwide locations. Many organizations have chosen to use the Function-as-a-Service (FaaS) model to enhance their business operations while saving both time and money. AWS Lambda has a variety of fascinating use cases, and few of them will be briefly covered in this chapter [61].

1. Mass Emailing

Each year, many huge corporations, for example, The New York Times, must send billions of transactional emails, as well as newsletters, and other publications. Mass emailing demands technological skill and is highly expensive, but it has become a vital aspect of any marketing campaign. Simple Email Service (SES) and AWS Lambda can work together to provide a cost-effective email flow, while the email list is safely stored in Simple Storage Service (S3). General steps are:

1. Upload CSV file
2. A CSV file is a trigger for an S3 event
3. That S3 event invokes the Amazon Lambda function
4. Lambda function inserts the CSV file to DynamoDB database
5. The same Lambda function emails the newly added email addresses [61]

2. Real-time notifications

“Amazon Simple Notification Service (Amazon SNS) is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication” [62]. SNS is in charge of topic development and determining which of the subscribers will be notified. The Lambda functions that have been assigned to the message can modify it and publish it [61].

3. Real-time data processing

Another use case for AWS Lambda is analysing data metrics in order to analyse data in real time and respond to it. The data from the website is captured using Amazon Kinesis Stream, while Lambda automatically grows the number of functions to meet the stream. The stream-based architecture may be used to manage a company's data sources throughout its website. These data sources give information to a stream, which runs a Lambda function. Otherwise, it is possible to manually invoke the functions with the 'RequestResponse' type. Kinesis and Lambda may be ideal for a company if it has a website or application that requires real-time processing of massive amounts of data. Other advantages include the ability to test new features and keep track of consumer satisfaction [61].

4. Alexa

Alexa, a well-known virtual assistant, converts human voice into a programming action. A simple function may be written to invoke a Lambda response; for example, Alexa can locate the nearest coffee store. Alexa can perform the following tasks:

- the Skill Handler Function offers the individual theory and reasons that must be run to fulfil the demands of the user
- interactions with third-party services, such as databases, are ensured by third-party functions
- as a result of the voice command, the handler function is triggered by Custom Alexa Skill Set [61]

5. Image recognition engine

Another use case for serverless Lambda is keeping track of a high number of picture uploads. Several workflows can be developed to handle it more effectively, with one workflow executing the next. The procedure is as follows:

1. the data extraction from a picture is the initial function
2. then the item in the image is identified
3. the following function will then create a thumbnail and return to the database

This is only an example; the pattern can be customized to match specific objectives. It is possible, for example, to crop the photos automatically in order to identify the primary subject [61].

6. Building a serverless website

Businesses have been able to focus resources and attention on their key capabilities and duties by avoiding the maintenance of servers, including virtual servers. Developers may focus on the real product instead of dealing with the day-to-day details of server management. AWS Lambda, in conjunction with DynamoDB, Amazon S3, and Amazon Cognito, enables them to create a website or application without having to manage servers or operating systems [61].

1. AWS S3 is used to host website content
2. DynamoDB is used as a noSQL database
3. Amazon Cognito handles user authentication and administrative chores
4. AWS Lambda develops database-accessing functions

Companies can create an e-commerce website that fits almost all of its needs. Payments, carts, and developing an engine are all examples of Amazon Lambda use cases. In preference, two Lambda functions could be used to create a dynamic web page and prevent losing data due to, for example, EC2 termination [61].

7. Authentication

The customisation of the authentication process is handled by Amazon Cognito in conjunction with AWS Lambda. Even in business, we are moving toward more personalized and individual communication. Lambda function can be linked to the following triggering sources:

- signing up and signing in (to guarantee the right series of questions is asked)
- authentication complexity (to keep the signing in process secure)
- customized message (to respond to a specific inquiries and requests quickly), etc.

Amazon Cognito, for example, will check with the Lambda function before sending a customer a confirmation email, allowing organization or developers to customise the phrasing and make it more helpful. Requests for a new password or important contact details, as well as letters of confirmation, are frequent triggers [61].

8. Text-to-speech

Amazon Polly, when used in conjunction with Lambda, enhances Amazon Lambda use cases by replicating speech and assuring faster reaction times, which becomes critical in real-time communication.

1. newly stored file in S3 starts the execution of the Lambda function
2. the Lambda function then starts an MP3 generating process

3. it also keeps a backup of the data in the database
4. the conversion of the text into an audio file is carried out by the second Lambda function [61]

9. Converting document formats

When a corporation provides documents (specifications, manuals, and so on) to its consumers via a website, they may not necessarily be expected to be in a single format. While many people may be OK with an HTML page, some may want to download a PDF or require a more specialized format. Storing static documents takes up a lot of space, and it's not practicable if the documents' content changes regularly or in reaction to user inputs. It is frequently far more convenient to create files on the fly. This is exactly the type of activity that an AWS Lambda application can do quickly and easily: obtaining the relevant content, formatting and converting it, and providing it for display on a webpage or download [63].

10. Rendering predictive pages

AWS Lambda can assist a corporation that uses predictive page rendering to prepare webpages for display based on the likelihood that the user will pick them. It's possible to utilize a Lambda-based application to retrieve documents and multimedia assets that could be used by the next page requested, as well as to execute the first phases of rendering them for display. If multimedia files are delivered by an external source, such as YouTube, the Lambda application can examine availability and seek to use other sources if necessary [63].

11. Automating backups and everyday tasks

AWS accounts benefit from scheduled Lambda events for maintenance. Using the boto3 Python libraries and hosted on AWS Lambda, you can quickly create backups, check for idle resources, generate reports, and do other common activities [63].

12. Backend cleaning

A quick response time is one of the top requirements for any consumer-oriented website. A late response, or simply an apparent delay in response, might result in considerable traffic loss. Backend tasks should not cause frontend queries to take longer to reply to. If you need to parse user input before storing it in a database, or if you need to perform other input-processing tasks that aren't required for rendering the next page, the data could be sent to an AWS Lambda process, which can then clean it up and send it to the database or application that will use it [63].

7.2.6 Best practices

Since serverless computing has its own set of disadvantages, it may not be appropriate for everyone or every use case. When it comes to developing AWS Lambda code, knowing how a Lambda function works may help us identify some of the best practices to follow when utilizing Lambda functions, as well as the value of each best practice. Function code and dependencies, function configuration, logging and monitoring are among the key practices described below.

1. The Lambda handler should be separated from service logic

It is recommended that the business logic be written in a separate service layer rather than in the handler function itself. It's because it makes it much simpler to write more readable and intelligible code, as well as unit-testable code for the business logic. For example, the handler function will be an entry point for a Lambda function and let's say that multiple variables are passed into the mentioned handler function. It may get problematic if each Lambda triggers an event with the first parameter containing all information about the triggering event [64].

2. The execution environment should be reused

Lambda workers will be utilized as long as they are up and operating, even if they terminate after a certain length of time. As a result, ensuring that the most of this reusability is taken may result in significantly reduced execution times. It is crucial not to save any sensitive data in the execution environment when reusing it. If these settings or environments are reused, there is a risk of data leakage, which can cause plenty of issues [64].

3. Environment variables should be used to manage operational parameters

Other operational parameters can be managed using variables whose values are set outside of the program. A name for an S3 bucket, for example, may be an environment variable rather than a hard-coded value. Lambda functions may be deployed into diverse environments without modifying the code by using env variables. SSM Parameter Store is an AWS service that saves variables and makes managing parameters for various environments or serverless architectures easier [64].

4. Common code should be deployed to a Lambda layer

A Lambda layer represents an archive that contains any additional code, such as libraries, dependencies, or custom runtimes. reference Lambda will extract archive's data in the layer to

the execution environment's /opt directory when setting up the exec. environment. Code and dependencies may be readily shared across numerous functions using Lambda layers. The Lambda function package size may also be lowered by adding heavy dependencies like AWS SDK, which will likely be reused in multiple Lambda functions, resulting in faster deployments [64].

5. Pay attention to the package size

The size of Lambda packages is expected to rise in conjunction with the expansion of the company, applications, and their logic. As package size rises, it has an impact on deployment time and, more critically, it can significantly increase the Lambda function's cold start time. The cold start refers to the time it takes to set up the execution environment (extracting or installing dependencies, etc.). In short, the time required rises as the package size grows larger; thus it's critical to keep the package size under control [64].

6. Recursive code should be avoided

When code is written to recursively call the Lambda function until a condition is met, it might result in a huge number of function calls and increased expenses. When compared to an EC2 instance for simple REST API use cases, Lambda functions are far less expensive. However, when faced with extended execution periods, Lambda functions may be very expensive [64].

7. One Lambda should use one IAM role

1:1 mapping should be maintained between Lambda and Identity and Access Management (IAM) roles at all times. When one function is allowed access to an AWS KMS key using IAM roles, any other function that utilizes the same IAM role will have access to the Key Management Systems (KMS) rule as well, possibly opening it up to exploitation. Also, while defining Lambda roles, the least privilege model should be used at all times, and no Lambda should have more permissions than it needs [64].

8. VPC access for Lambda should be controlled

Lambda functions operate in a shared Virtual Private Cloud (VPC) at all times and may send network requests to any address on the internet, including other AWS APIs. Private VPCs are also used to ensure that specific resources are operated in a secure manner and that other services may access these private resources in a restricted way. As a result, it's critical to ensure that only Lambda functions that require access to these resources have VPC access enabled. It's also worth noting that once the function is VPC enabled, any traffic generated by the Lambda function is subject to the VPC's routing rules [64].

9. Concurrency for critical functions should be reserved

The default concurrent execution limit for Lambda functions is 1000 across all functions in a particular region, which is a soft limit and can be adjusted. If this limit is reached, Lambda functions will begin to reject inbound requests. And this can lead to the failure of certain extremely important company functions, resulting in revenue loss. Consequently, reserved concurrency can be a good way to ensure that crucial functions don't fail because of the concurrent limit. To ensure that a set amount of concurrent requests is reserved for a wanted business-critical Lambda function, a specific quantity of concurrent requests may be stored for a specific Lambda function. The reserved limit will not be utilized by other function invocations and will be accessible for the chosen business-critical function even if the concurrent limit is reached [64].

10. The right balance between memory and execution time should be found

The CPU for Lambda is proportionately given to the RAM allocated to the Lambda function. As a result, giving extra memory to a function might generally help it run quicker and save money. However, as memory grows, so do expenses, and it's crucial to notice that at a certain point, the reduction in execution becomes significantly less significant in comparison to cost growth. Therefore, finding the right balance between memory and execution time is essential to ensuring Lambda runs in the highest suitable environment [64].

11. AWS CloudWatch logs could be useful

Although this is arguable when it comes to best practices, application logs are often created and accessed on the application server on which the application is deployed. This is not feasible with Lambda since the application developers do not have access to a server. As a result, AWS has put up a method for Lambda functions to store logs to Amazon CloudWatch, and application developers must ensure that Lambda functions have the required permissions to save logs to CloudWatch. This may be further improved by utilizing CloudWatch triggers to set up alerts for any essential logs, such as production errors [64].

12. AWS Config

AWS Config may or may not immediately improve the performance or security of a Lambda function. However, AWS Config can log changes to a Lambda function's settings, such as runtime environment data, handler name, code size, security groups, IAM roles, and so on [64].

Using the best practices stated above when setting application logic should result in a business application environment that is optimized and safe [64].

8. Data Masking – serverless solution

After comprehending the concepts of data masking and FaaS (serverless) architecture, it is time to clarify the approach for implementing own solution. Once again, the task is to implement a data masking solution in a serverless architecture using Amazon Lambda as a FaaS environment. The mentioned data masking solution will not resemble the current Libelle product interface, but the anonymization functionality will be comparable. The goal of this work is not to transform a monolithic application into a serverless application, but to obtain the simplest possible process of anonymizing data using the existing product's algorithms and features.

8.1 Solution approach

This section will describe the overall approach to accomplishing the task, including technologies, issues, and solutions encountered while developing the product. Since the company provided the core components of the Libelle DataMasking product as well as anonymization algorithms, the main part of the task was to combine the obtained program code with AWS Lambda and potentially other services or technologies.

The Libelle **Data***Masking* (LDM) product currently anonymizes entered data based on:

- Anonymization settings
- References
- Anonymization key

Anonymization settings are configuration files used to adapt and personalise the anonymization process, such as for a specific database or other requirements. The anonymized values come from reference files. Some anonymization algorithms need a reference file that consists of names, addresses, organizations, URLs, e-mail, domains, or other needed values. These reference files can be replaced with one’s own references. Anonymization key is a value used to calculate which reference data is going to be used from reference files. In the serverless solution, a corresponding file structure should be used. Since the LDM python code package was received from colleagues within the company, it must be adapted and utilised into Amazon Lambda. Below is an initial diagram describing the structure of the desired product.

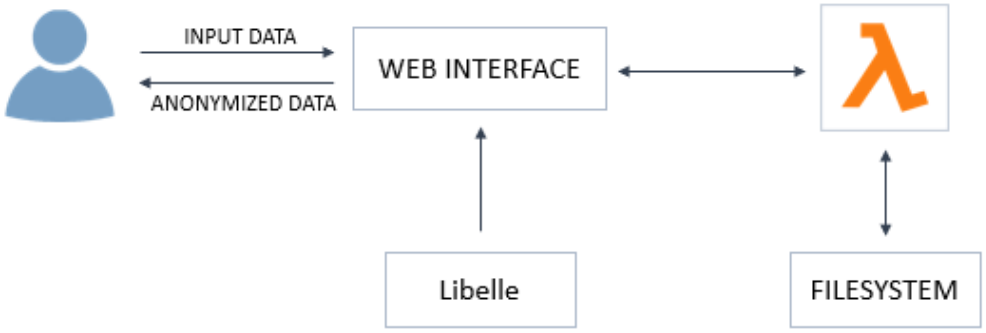

Image 15 - Initial serverless data masking solution diagram

The above diagram illustrates the user's interaction with the Libelle-hosted web interface, which is used to send data to Lambda and receive a response. The diagram also shows the filesystem that is implemented to cooperate with the lambda function.

As previously stated, the user can use his own anonymization settings and references based on his preferences. As a result, the idea is that multiple users can save on the server and reuse certain files considered necessary for anonymization. These files must be optimally organised using the filesystem in order to be properly stored and used. The following paragraphs will be divided into sections based on the above diagram to describe the concepts and the implementation approach in greater detail.

8.1.1 User input and actions

The product's main functionality is, of course, anonymization, but as previously noted, there are mainly three types of files used during anonymization: settings, references, and keys. To add some features to this functionality, the user can select the specified predefined files or use their own that are customized to their requirements. The end goal is for the user to be able to perform several **'actions'** with 4 different '**objects'** by entering input data correctly.

These **Actions** would be:

- **anon**: anonymize entered data
- **list**: list available configuration files (anonymization settings, references, or keys)
- **get**: see the content of available configuration files
- **set**: set own configuration files (store them on the server)
- **delete**: delete previously set configuration files
- **login**: user login (at Libelle-hosted web frontend)

Objects that can be listed, seen, and deleted:

- anon_settings
- reference_settings
- reference_file
- anon_key

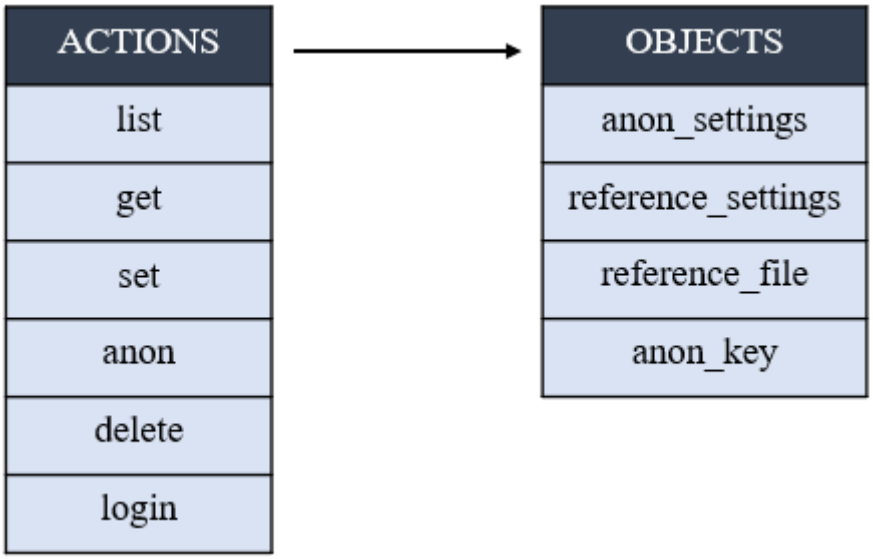

Image 16 - Available actions and objects

The following table contains a list of all **action-object** combinations. To further illustrate the point with an example - it is possible to list, get, set, and delete *anon_settings*, but it is not possible to anonymize them. It is directly analogous to all other actions, i.e., objects.

	list	get	set	anon	delete	login
anon_settings	+	+	+	-	+	
reference_settings	+	+	+	-	+	
reference_file	+	+	+	-	+	
anon_key	+	+	+	-	+	
anon_data	-	-	-	+	-	
encryption	+	+	+	+	+	

Legend
Can execute action with object
Cannot execute action with object
Not yet implemented

Image 17 - All action-object possibilities

User input must be in the JavaScript Object Notation (JSON) format, where action name has to be specified, together with other needed parameters. The image below shows a JSON template form with all possible variants of parameter input, separated by pipe (‘|’) sign.

```
{
    "action": "anon | list | get | set | delete | login",
    "object": "anon_settings | reference_settings | reference_file | anon_key",
    "data"  : [
                {
                "FNAME": "John",
                "LNAME": "Doe",
                "COMPANY": "Libelle",
                "CAR": "Land Rover"
                },
                {
                "FNAME": "Jane",
                "LNAME": "Smith",
                "COMPANY": "datamir",
                "CAR": "Jaguar"
                }
                ]
    ,
    "anon_settings":{
        "type" : "id | data",
        "id"   : "id_of_specific_settings_file",
        "settings": []
    },
    "references": {
        "type"  : "id | data",
        "id":"id_of_specific_reference_file",
        "references":[]
    },
    "anon_key": {
        "type"  : "id | data",
        "id"    : "id_of_specific_key_file",
        "value" : ""
    },
    "encryption" :"base64 | none"
}
```

Image 18 - JSON template for data input

Action ‘list’

The 'list' action displays a list of available objects and IDs for use in the anonymization process. A list of Reference Files, Anonymization Settings, or Anon Keys, for example. After listing and inspecting what is already stored in the filesystem, the user can specify and employ it in the data anonymization process.

To execute ‘list’ action, user must specify which objects he wants to list (for example, object = anon_settings).

Image 19 - Action 'list' example

Action ‘get’

The 'get' action allows the user to retrieve and view the content of various available objects and IDs that can be used in the anonymization process. This action can assist the user in determining which configuration file is best for him.

To execute ‘get’ action, user must specify object and object ID (for example, object = anon_settings, object ID = SETTINGS_1).

Image 20 - Action 'get' example

Action ‘set’

The 'set' action allows the user to send objects to the Lambda function and save them on the server. All four objects can be set, and each user can use objects set by himself or objects delivered by Libelle.

To execute ‘set’ action, user must specify object, object ID, and relating data values (for example, object = anon_settings, object ID = new_settings_id, settings = [relevant_data]).

Image 21 - Action 'set' example

Action ‘anon’

The 'anon' action allows the user to send data and have it anonymized using the anonymization settings, reference file, and anon key provided by Libelle, or objects previously sent via the 'set' action. In the ‘anon’ action, object parameter doesn’t have to be defined.

To use the 'anon' action, the user must enter data for anonymization as well as anonymization settings. Optionally, reference files can also be specified. For example, raw data can be some personal information (name, surname, company, and car), while anonymization settings, references and key can be selected from already existing one (in this case IDs of those files must be specified).

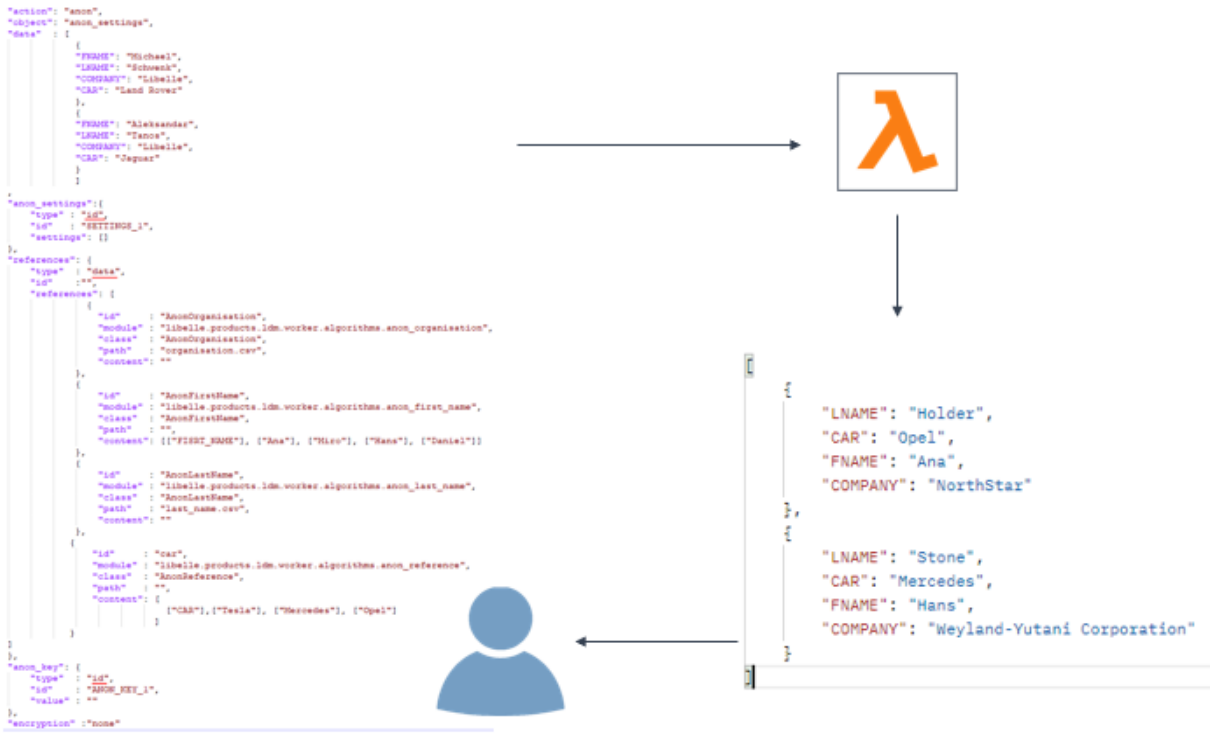

Image 22 - Action 'anon' example

To simplify the above diagram, which can have complex data input, an example of only the 'data' parameter with the given result is provided below.

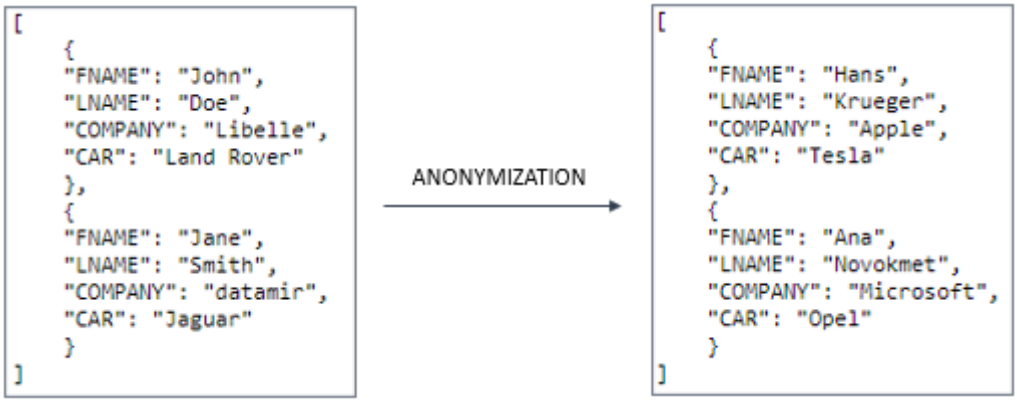

Image 23 - Raw data input with anonymized output example

Action 'delete'

The 'delete' action allows the user to delete all four objects that are not provided by Libelle. More specifically, the user can delete the files he previously saved on the server (via the 'set' action).

To use the 'delete' action, user must specify which object he wants to delete together with specific object ID (for example, object = anon_settings, object ID = new_settings_id).

Image 24 - Action 'delete' example

Action ‘login’

The "login" action is intended to allow the user to log in and potentially use a data masking solution tailored to him. In this case, an individual user can only see and use the files that he has stored on the server, in addition to those previously delivered by Libelle. This functionality is expected to be available in the near future.

8.1.2 Lambda function

Once the form of the input data has become slightly clearer, the Lambda function must be modified to accept it. Considering that the input data will be delivered to Lambda through the web interface, the only logical way to send data is via a REST API call. Application programming interfaces (APIs) allow one programme to communicate with another. API calls are the way in which they interact. An API call, also known as an API request, is a message sent to a server that requests that an API provide a service or information. API requests are sent from a client to an API endpoint. API endpoints are the destinations of API calls, which are typically a web application and a server. For example, a mobile client generates an API call that is routed to the API endpoint, which is a server. The API call is received by the server, processed, the request is executed, and a response is sent [65]. REST is an architectural style for distributed hypermedia systems that stands for REpresentational State Transfer [66]. REST API is an architectural style for developing web services that communicate using the HTTP protocol [67].

There are two approaches to sending API calls to Lambda: Amazon API Gateway and Lambda Function URL. The Amazon API Gateway is an AWS service that enables the development of REST, HTTP, and WebSocket APIs at any scale. When Amazon API Gateway and AWS Lambda are used to build APIs, API Gateway serves as the HTTPS endpoint URL [68]. The API Gateway manages all aspects of accepting and processing hundreds of thousands of concurrent API calls, such as traffic management, CORS support, authorization and access

control, throttling, monitoring, and API version management. There are no minimum fees or startup costs with API Gateway. The user is charged for the API calls he receives as well as the amount of data transferred out. The API Gateway tiered pricing model allows users to reduce their costs as their API usage grows [69]. A function URL, on the other hand, is a dedicated HTTP(S) endpoint for the Lambda function. A function URL can be created and configured using the Lambda console or the Lambda API. When a user creates a function URL, Lambda creates a unique URL endpoint for it which does not change [70]. Function URLs can be added to new or existing Lambda functions [68].

	Lambda Function URLs	***Amazon API Gateway***
Resource	Lambda with Function URL	API Gateway + Lambda
API Type support	HTTP	HTTP, REST, Websocket
AuthType	IAM	API Key, IAM, Cognito, Lambda
Response timeout	15 minutes	29 seconds
CloudWatch metrics	YES	YES
CORS	YES	YES
Validate request and response	NO	YES
Access CloudWatch logs	NO	YES
Manage API Key	NO	YES
Request & response mapping	NO	YES
Caching	NO	YES
Usage plans	NO	YES
Proxy to other AWS services	NO	YES
Websocket	NO	YES
Price	Applied only Lambda price	Lambda price + API Gateway price

Table 4 – Comparison of Lambda URLs and API Gateway

Function URLs can be interpreted as a simpler option, with far fewer extension possibilities than the API Gateway. Given that the serverless data masking solution only needs to safely receive input data, Function URL appears to be the simplest, cheapest, and thus best option. Furthermore, the product is not dependent on AWS, i.e., another Amazon service, which provides a bit more control over the product itself.

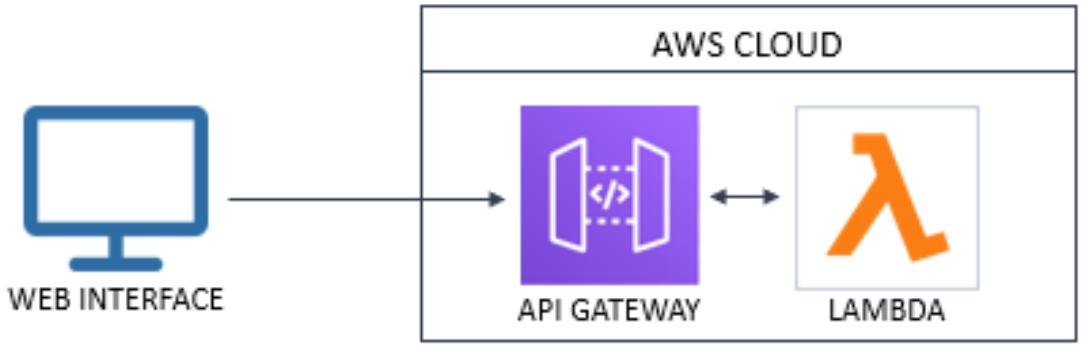

Image 25 - API call diagram using Amazon API Gateway

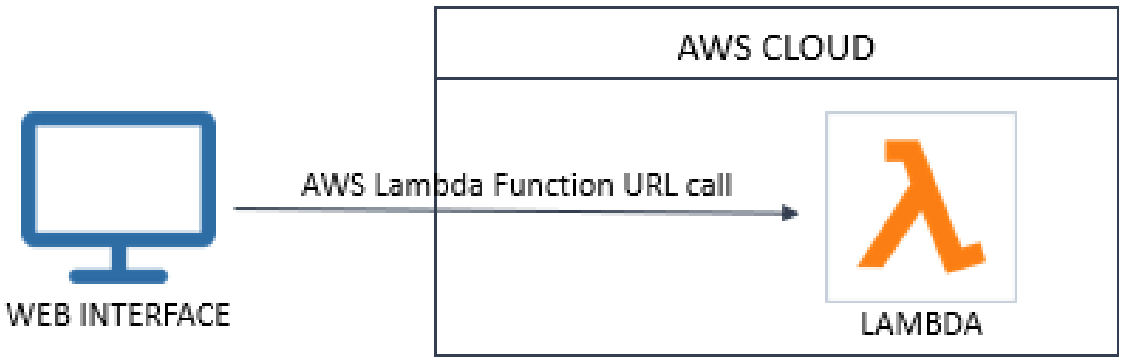

Image 26 - API call diagram using Lambda's Function URL

8.1.3 Filesystem

As mentioned, a filesystem is required to store and organise files such as anonymization settings, references, and keys. These files must be easily accessible and modifiable. Simple Storage Service (S3) is one of Amazon's most popular file storage services. Amazon also provides Elastic Block Storage (EBS) and Elastic File System (EFS) storage services. Amazon EBS is a simple, scalable, high-performance block storage service for Amazon Elastic Compute Cloud (EC2) [71]. EBS is only accessible by a single EC2 instance, making it most similar to a physical hard drive [72]. Therefore, EBS is incompatible with the data masking solution. The second storage option, Amazon S3, is an object storage service that provides industry-leading scalability, data availability, security, and performance. S3 has access to multiple EC2 instances and is ideal for handling large volumes of static data. S3 provides access to multiple EC2 instances and is ideal for storing large amounts of static data [72]. S3 service in general can be good for triggering Lambda functions but isn’t good for network file systems (NFS) as it is not good for reading and writing the data. Because data gets to be stored in buckets, the S3 service in general is good for triggering Lambda functions but not for network file systems (NFS). S3 is not suitable for reading and writing data because files cannot be appended. In the end, the only solution left is the Elastic File System (EFS). Amazon EFS is a simple, serverless, set-and-forget elastic file system that grows and shrinks automatically as files are added or removed. Management and provisioning are not required [73]. EFS' performance can scale in lockstep with its storage, reaching up to 500,000 Input/output operations per second (IOPS) or 10 GB per second for sudden, high-volume file dumps [74]. The main feature of EFS is its scalability, which, combined with its speed and accessibility [72], makes it ideal for the serverless data masking solution.

8.1.4 Interface

Given that the goal of the work is to implement the simplest possible principle of data anonymization, the interface also needs to be user-friendly and not overly complicated. The web frontend consists of a multiline field for user input, a submit button, and a field for the application's response. Libelle company will host this application so that it is always accessible to both registered and other service users.

Below is a complete architecture diagram that summarises the solution construction.

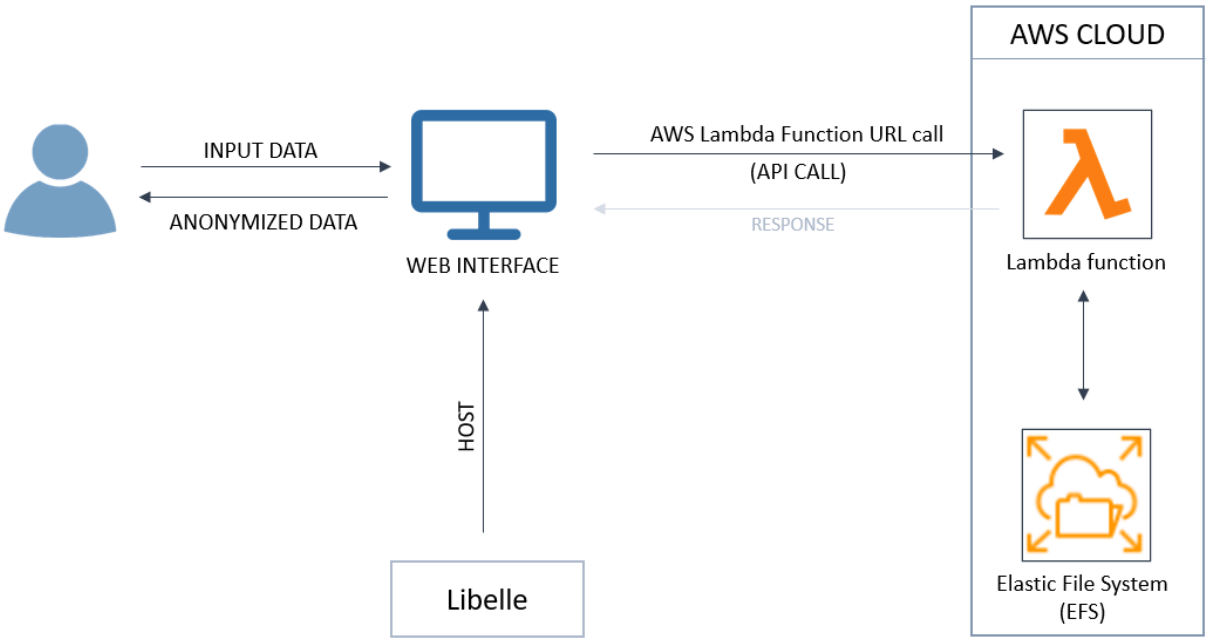

Image 27 - Diagram of a serverless data anonymization solution

8.2 Use cases

The FaaS method of implementing data masking solution has a variety of use cases. They mostly vary depending on how the product is used, or more specifically, according to the "actions" and configuration settings that the user will use. Some of them will be shortly described.

1. **Quick and infrequent data anonymization**

In this scenario, the user occasionally wants to anonymize data and sends all the data necessary for the anonymization process. He doesn't have a lot of data or doesn't use it frequently since he simply wants to test out the functionality. Nothing is saved on the server following this anonymization procedure. A different approach to this use case is to make use of pre-existing configuration settings, where the user only sends the data that he wants to anonymize and is not concerned with the standards that will be used.

Actions used in this use case: anon

2. **Anonymizing after finding the best suitable anonymization settings and references**

In this scenario, the user might want to inspect what is offered and available on the server first. He can begin the anonymization process after determining the configuration settings that are most appropriate for his data.

Actions that can be used in this use case: list, get, anon

3. **Frequent anonymization - setting own anonymization settings and references**

In this scenario, the user can integrate the anonymization procedure into the process for his own environment and landscape. It could be applied frequently to the same target system in an already-existing environment (for example, once every three months). The goal of this use case is to have reusable configuration settings for all upcoming anonymization executions.

Actions that can be used in this use case: set, anon

4. **Frequent anonymization - setting own configuration settings and deleting them after**

In this scenario, the user can integrate the anonymization procedure as a fully automated process, but for some new system that has the identical anonymization settings. The idea behind this use case is that the user can make use of the stored (previously set) settings and then clean them up, preventing them from being stored on the server permanently.

Actions that can be used in this use case: set, anon, delete

9. Implementation

This section will describe how to create a Lambda function and the related configuration, including Function URL, Elastic File System (EFS), and other technologies.

9.1 Initial Lambda function creation

The first thing to do is to create the initial Lambda function. The process of creating an Amazon Lambda function using the Console is displayed in the image below.

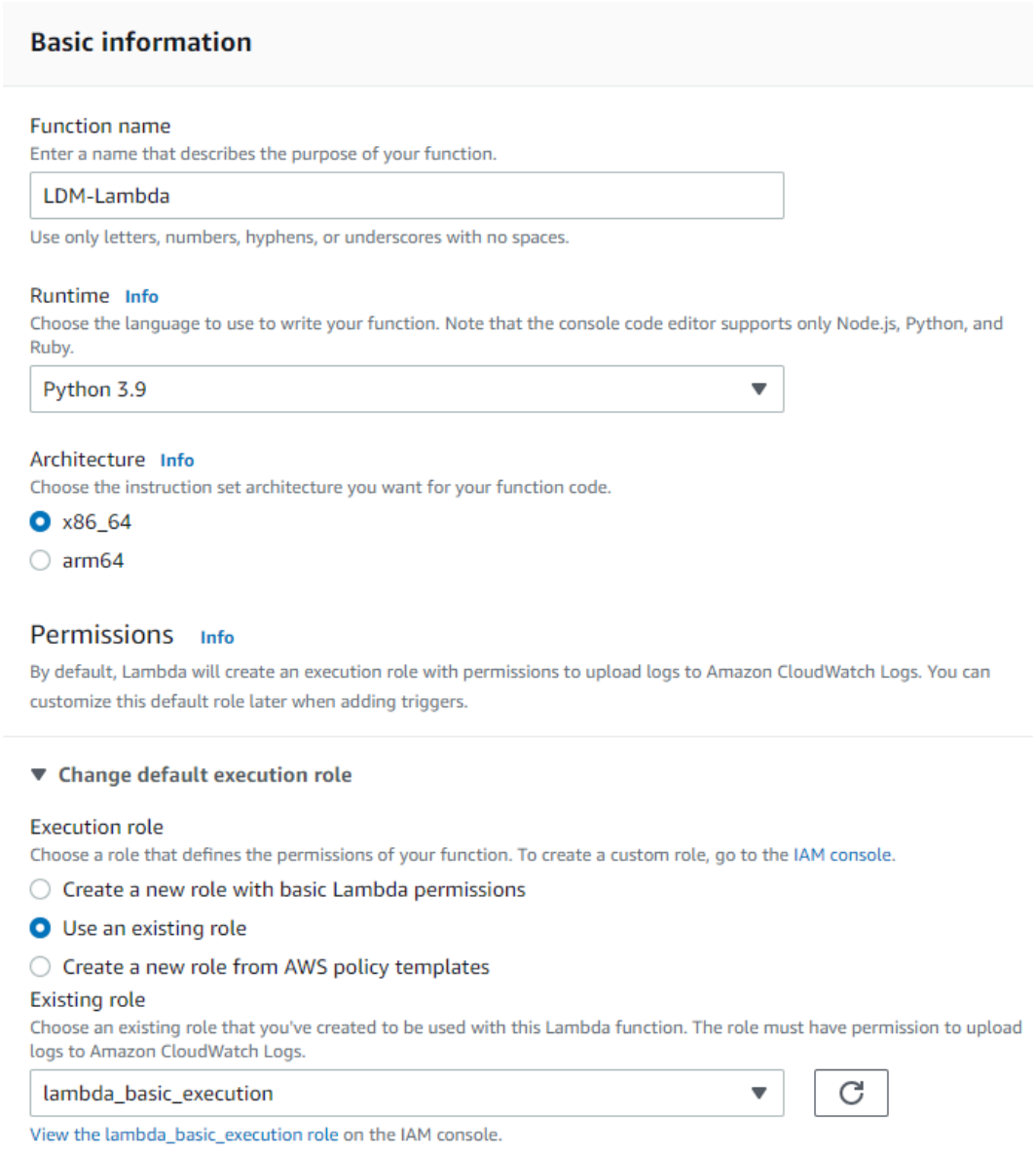

Image 28 - Initial Lambda function creation

On the image, all of the configuration settings are visible. Later on, the execution role will grow to be a significant factor of the function; for that reason, a more thorough explanation will be provided.

A function's overview and its automatically generated source code are displayed after the function has been successfully created.

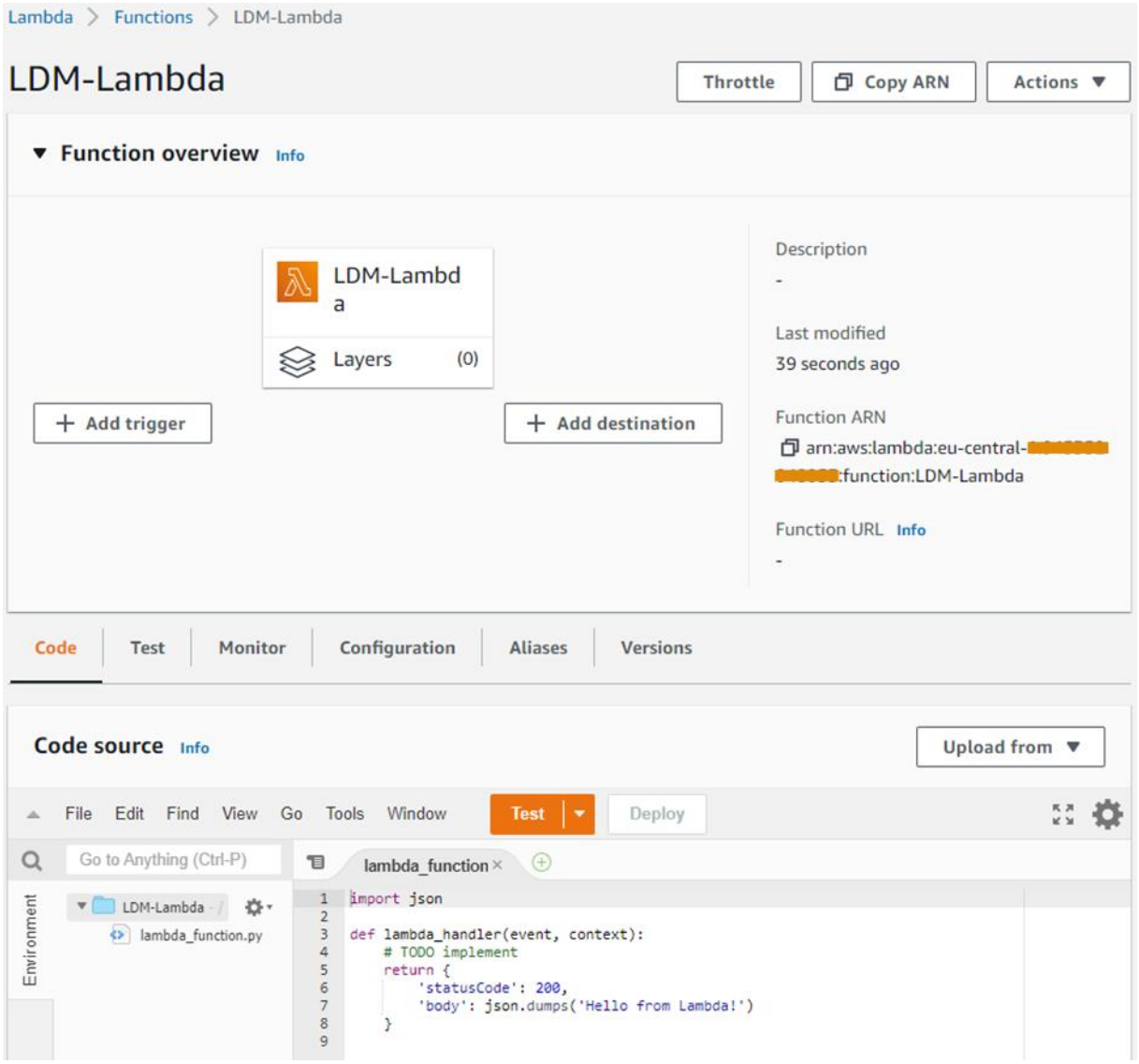

Image 29 - Initial function overview and code source

The code of an AWS Lambda function is made up of scripts, or compiled programs, as well as their dependencies. The function code is deployed to Lambda using the Deployment package. Container images and .zip file archives are the two types of deployment packages supported by Lambda [75]. The Libelle **Data***Masking* (LDM) .zip file archive will be used as a deployment package as the code package is already available. The LDM package is about 50 kilobytes in size, and if it were larger than 10 megabytes, it would have to be uploaded to a Simple Storage Service (S3) bucket from which Lambda would pull all the files.

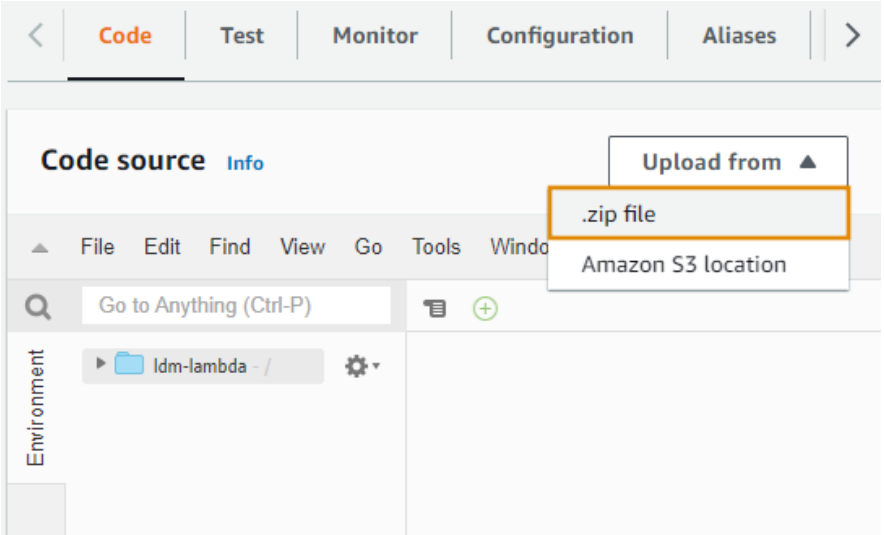

Image 30 - Libelle DataMasking and Lambda deployment package

When new .zip file package is uploaded, the existing code gets overwritten. Since the lambda function is absent from the uploaded deployment package, which only contains LDM code, it must be written and tested. A "test event" must be created in order to test the initial Lambda function, but it can be left with its default generated values.

Image 31 - Configuring Lambda test event

After successfully creating a test, the function must first be deployed (by clicking the "Deploy" button), and only then can Lambda function be started (by clicking the "Test" button).

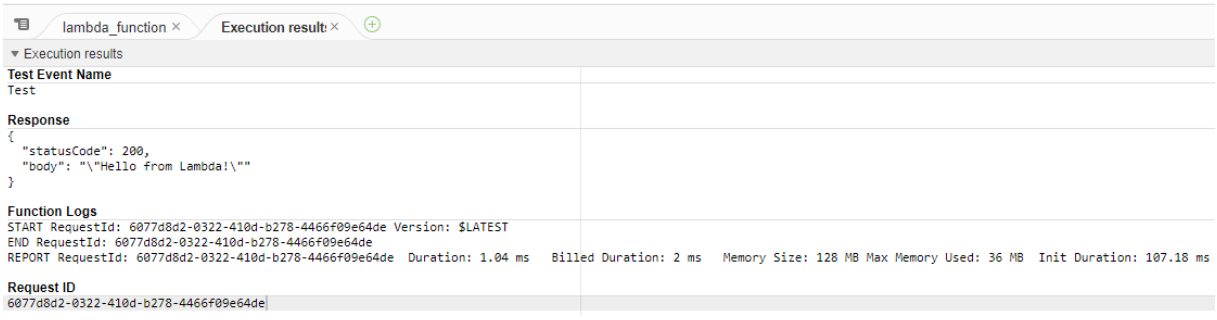

Image 32 - Basic Lambda function execution results

As seen in the above image, the execution results of the first function are:

- Function response, RequestId and version
- Execution duration: 1.04 milliseconds
- Billed duration: 2 milliseconds
- Memory size: 128 megabytes
- Maximum memory used: 36 megabytes
- Initialization duration: 107.18 milliseconds

9.2 Function URL configuration

Now that there exists a working Lambda function that can be run via the AWS Management Console, it is necessary to enable it to run externally, i.e., independent of the AWS account. Therefore, the Function URL endpoint for the current Lambda function needs to be configured. The AWS Management Console has a number of tabs for viewing, configuring, and monitoring function activities under the "Function overview" section. Currently, it is necessary to navigate to the "Function URL" section of the "Configuration" tab.

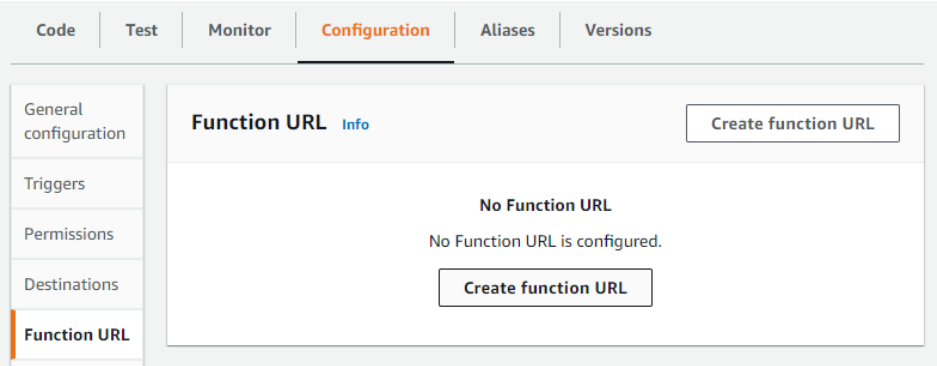

Image 33 - Navigating to the 'Function URL' section under the Configuration tab

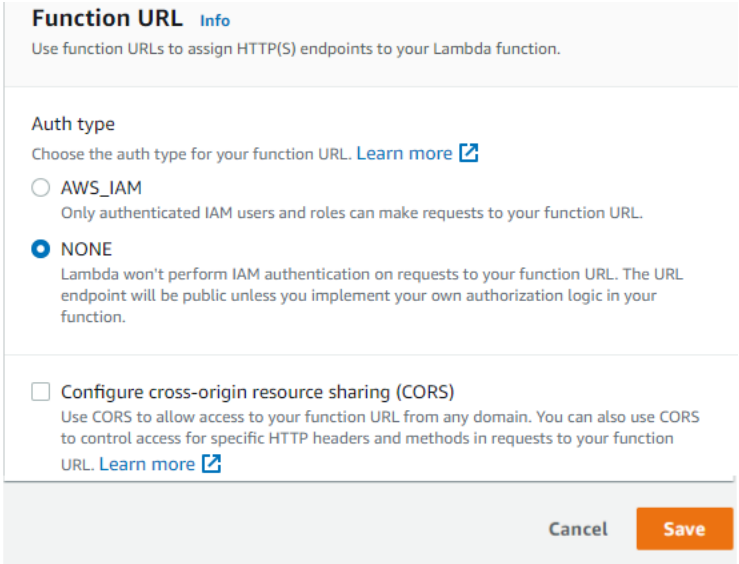

Image 34 - Configuring Function URL

In order for the URL endpoint to be publicly accessible and not restricted by an AWS account or an IAM role, the authority type must be set to "NONE," as seen in the image above. With this choice, Lambda automatically creates the needed resource-based policy and attaches it to the function. Cross-origin resource sharing (CORS) configuration is not required right now, but it will be explained later on when it is considered necessary for the product's functionality. After saving this configuration dialog, the Lambda function receives a dedicated HTTP(S) endpoint that will be used for the function invocation through an HTTP client (for example, browser, Postman, curl, and others). Function URL endpoint is of format:

```
https://<url-id>.lambda-url.<region>.on.aws
```

Image 35 - Function URL endpoint format

The next step would be to invoke the function externally in order to test it using the configured Function URL. For this, Postman, an API platform for creating and utilising APIs [76], will be used.

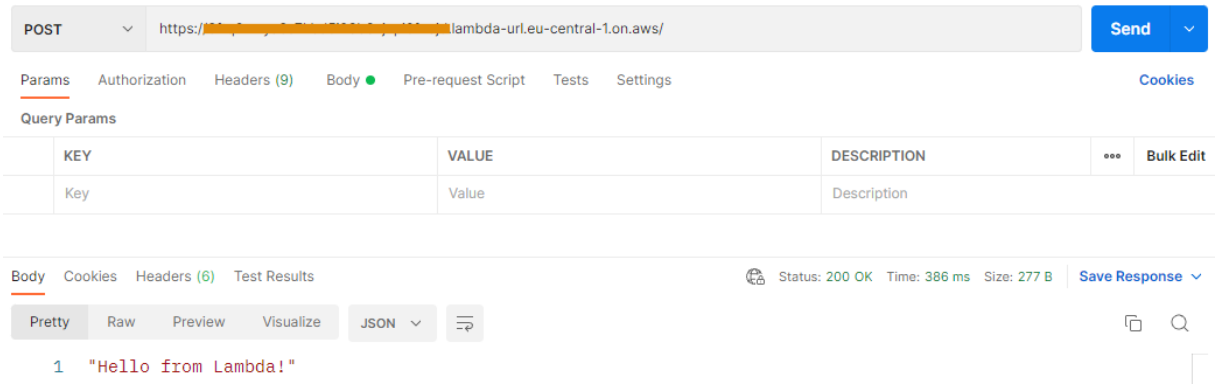

Image 36 – Initial POST API request for the Lambda Function using Postman

This scenario explains the POST API request, which is used to send data to the API server in order to create or update a resource. Function URL endpoint value represents the request URL in Postman. To send a request and invoke the Lambda function, just those two parameters are required. As a result, the same message that appears in the Response body section of the AWS Management Console, "Hello from Lambda," is displayed.

9.3 Adding functionality to Lambda

After the Lambda function invocation from the outside has been enabled, it's time to upgrade it. More specifically, it is necessary to finally include the required functionality, that is, data masking.

The request name and URL endpoint were the only parameters included in the previous section's POST API request. By default, the request has a ‘Body’ parameter, which was in that case empty. The mentioned 'Body' parameter must contain the data that needs to be sent through an API call. The JSON data that was already described in this paper's "Solution approach" section will be the "Body" parameter for this serverless data masking solution. The JSON parameters that the user can send through the API request, i.e., through a web interface are:

Parameter	*Value*
action	list, get, set, anon, delete, login
object	anon_settings, reference_settings, reference_file, anon_key
data	List of data to be anonymized
anon_settings_type	id (to use existing file), data (to send own settings)
anon_settings_id	ID of a settings file to be used/set/deleted
anon_settings	List of anonymization settings to be used
reference_type	id (to use existing file), data (to send own settings)
reference_id	ID of a reference file to be used/set/deleted
references	Reference data to be used
anon_key_type	id (to use existing file), value (to send own settings)
anon_key_id	ID of an anon_key file to be used/set/deleted
anon_key_value	Value of the anon_key to be used
encryption	None (for no encryption), base64 (for sending encrypted data to Lambda)

Table 5 - List of input data parameters

The Lambda function must therefore be able to accept the specified data and process it adequately in order to be able to send the anonymized data back to the user. The Lambda is developed to process the data based on the delivered "action" parameter. Other parameters are considered depending on the desired "action," after which additional actions required for masking are taken.

```
def lambda_handler(event, context):

    # Receive input data parameters

    if action   == 'anon':
        # result = ...
    elif action == 'list':
        # result = ...
    elif action == 'get':
        # result = ...
    elif action == 'set':
        # result = ...
    elif action == 'delete':
        # result = ...
    elif action == 'login':
        # result = ...
    else:
        # result = wrong action input

    return result
```

Image 37 - Pseudocode for Lambda processing the input data

Employees of Libelle Group are responsible for creating the functions that are used in the data anonymization component itself. The functions needed for handling input data and performing the desired actions (list, get, set, anon, delete) were additionally created for the purposes of this master's thesis project. These python functions are imported as modules and are organized like stated below:

- **Anonymizer ()**
  - Class in '**anonymizer'** module with functions provided by Libelle
- **Anonymization ()**
  - Class in '**anon_actions**' module with functions made for handling input data and action executions

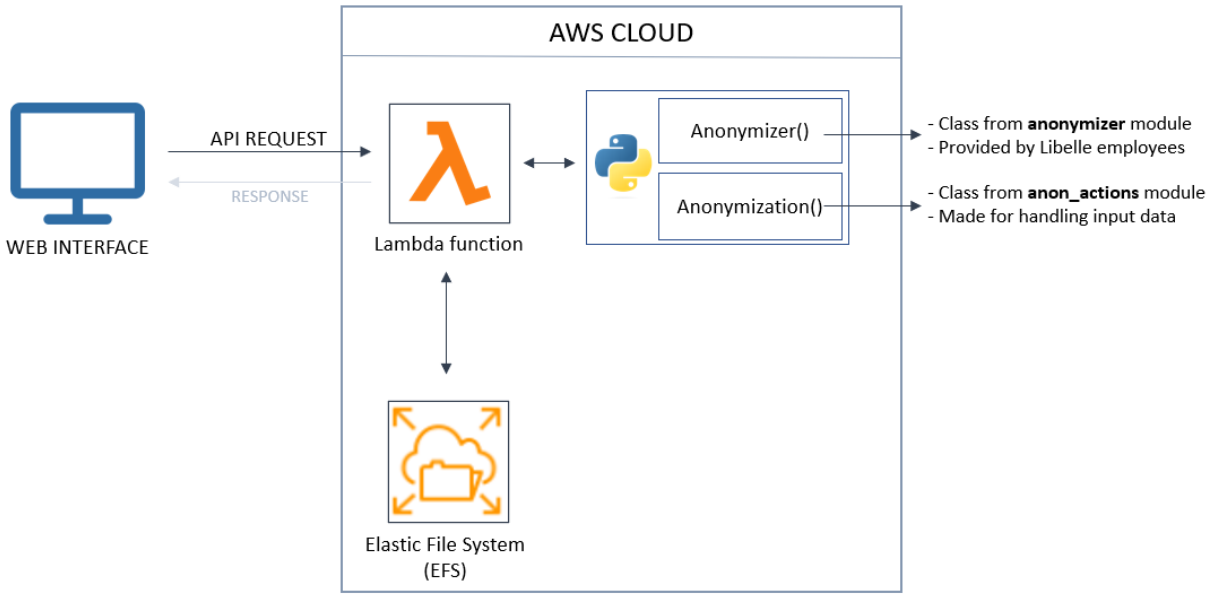

Image 38 - Lambda function and modules architecture

```
from libelle.products.ldm.worker.anonymizer import Anonymizer
from anon_actions import Anonymization

def lambda_handler(event, context):

    myAnonymizer = Anonymizer()
    myAnonAction = Anonymization()

    # ...
```

Image 39 - Importing python modules and classes to the Lambda function

The complete Lambda function code will be provided as an attachment at the end of the "Implementation" section.

9.4 Creating and adding filesystem

Some definitions of "actions" include getting specific file content, placing or storing files on the server, or even removing user-uploaded files. Therefore, files must be saved and organised in a filesystem, in this case AWS Elastic File System (EFS), in order to perform any of these file operations.

For EFS to work with AWS Lambda, some other services must be configured. The next steps describe the complete process of configuring and implementing EFS:

1. Virtual Private Cloud (VPC) configuration
2. Security Group configuration
3. IAM Role configuration
4. EFS configuration
5. Lambda configuration

9.4.1 VPC configuration

On the VPC dashboard, new VPC must be created using settings shown in the image below.

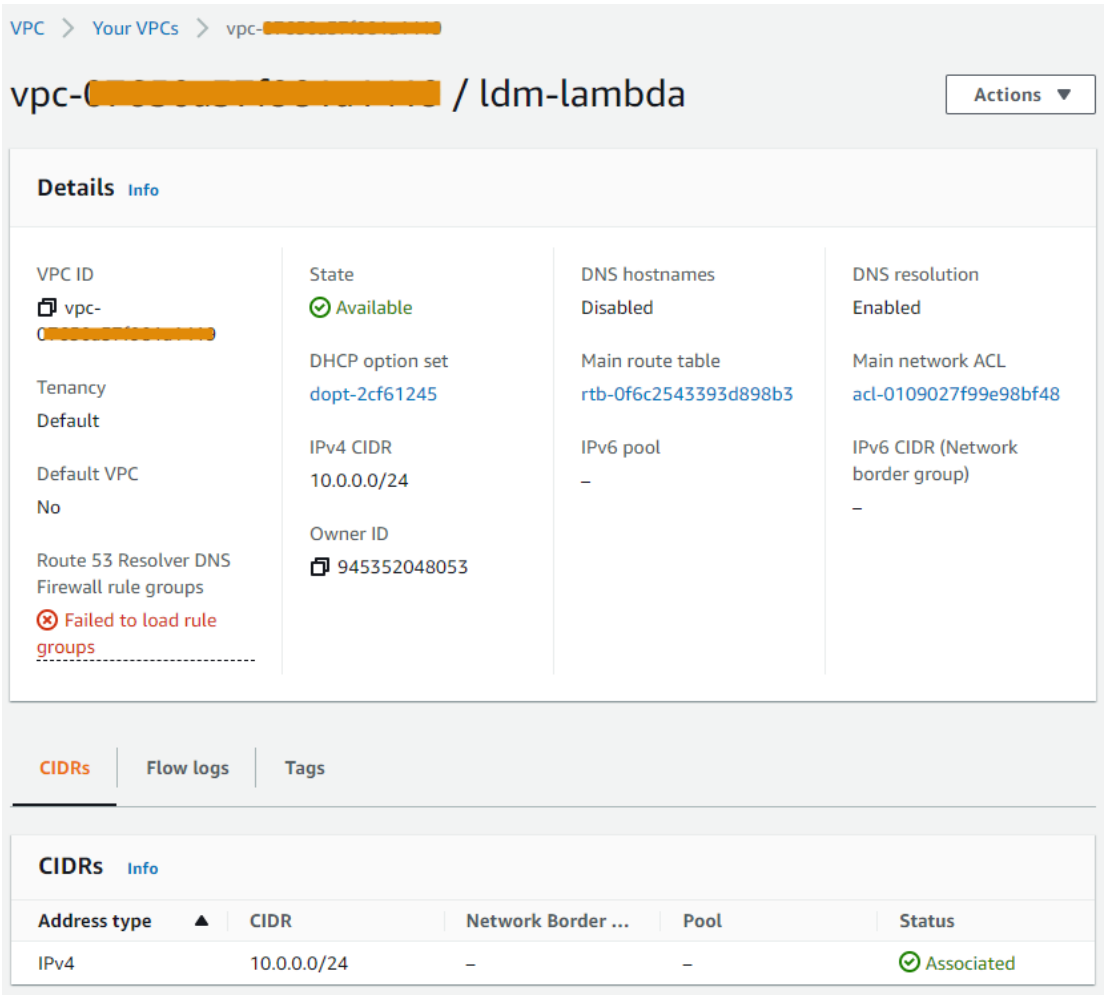

Image 40 - VPC settings

The optimum configuration for a VPC that will be connected to EFS, and Lambda is to have a total of three subnets—one public and two privates—in one network.

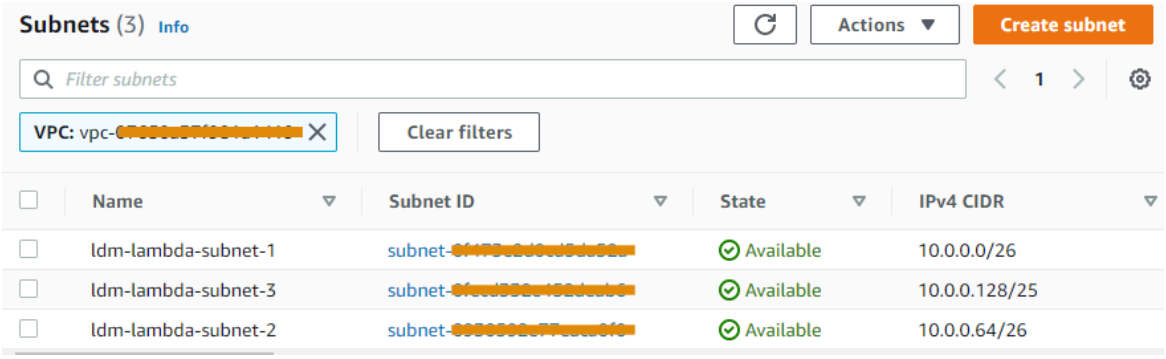

Image 41 - Subnets settings under created VPC

The "Security" tab of the VPC Dashboard enables the implementation of a Security Group under an existing VPC. To give a Lambda function access to the filesystem, inbound rule of type ‘Custom TCP’, port ‘2049’, and source ‘Anywhere’ should be added.

9.4.2 EFS configuration

Everything is set up for the creation of the Elastic File System after the VPC, Subnets, and Security group have been successfully created. The "customise" option needs to be enabled when creating the filesystem on the EFS Dashboard. As can be seen in the image below, there are numerous settings that must be utilized.

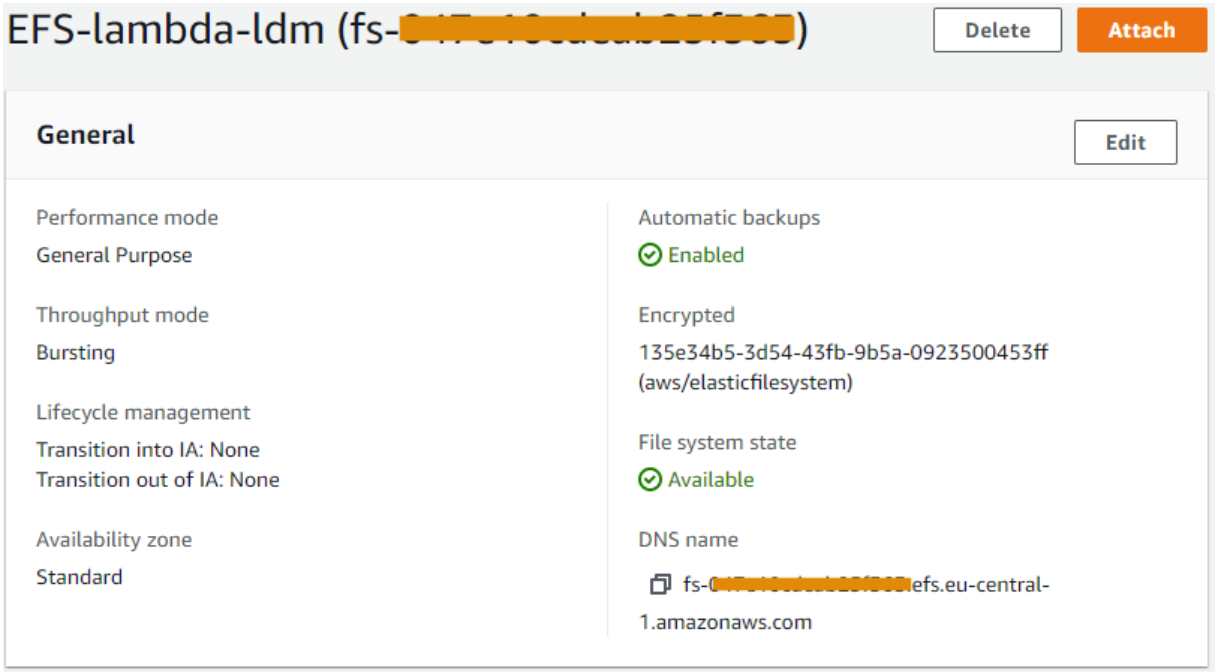

Image 42 - EFS general configuration

The next crucial step is to set up network access for the EFS. Utilizing mount targets, different instances can connect to a filesystem. A mount target should be present in each VPC's Availability Zone (i.e., subnet) so that the EFS can be accessed from various zones. The Security Group that was previously created should then be connected to each VPC's subnet. One can also enforce read-only access, disable root access by default, and require in-transit encryption for all clients when setting up the EFS.

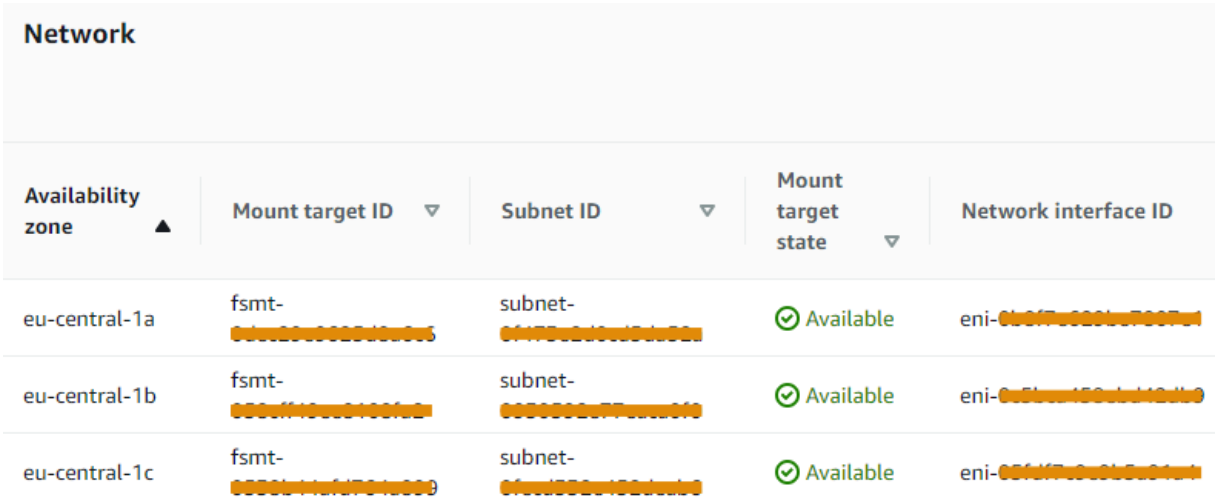

Image 43 - EFS Network configuration

The creation of an access point is the final and most important step in granting applications access to the filesystem. All connections to the newly created access point will also be configured with POSIX identities and root directories in this scenario.

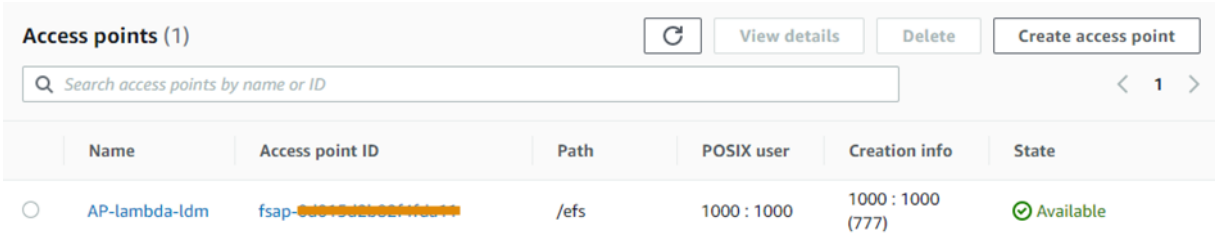

Image 44 - EFS Access point configuration

9.4.3 IAM role configuration

It is necessary to create an IAM role in the IAM console. A Lambda use case must be selected prior to role creation. The following policies must be attached to it:

- AWSLambdaExecute
- AWSLambdaVPCAccessExecutionRole
- AmazonElasticFileSystemClientFullAccess

9.4.4 Attaching VPC and EFS to the Lambda

Returning to the Lambda Dashboard, it is necessary to associate the Lambda function with any previously set up services and settings. The Execution role can be set to the previously created one in the "Permissions" section of the "Configuration" tab. Next, a new VPC must be added in the "VPC" section along with three Subnets and a Security Group.

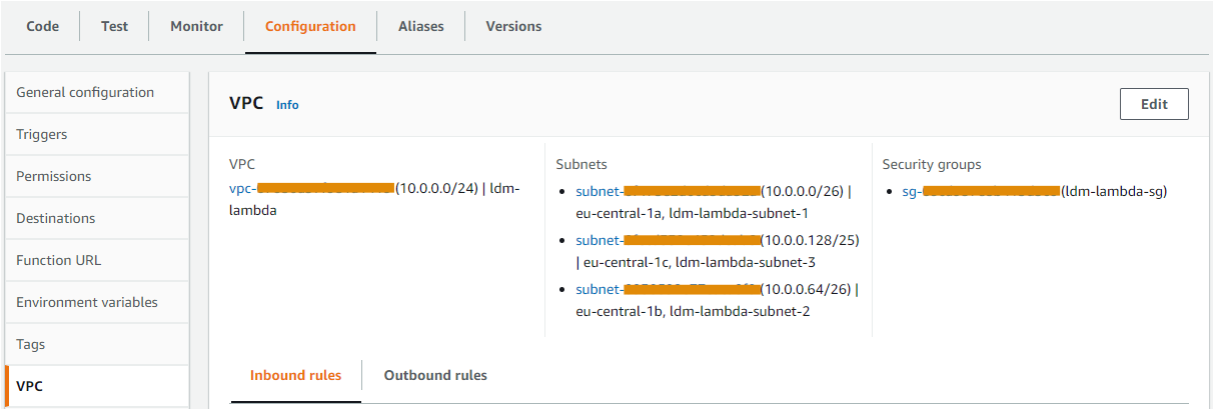

Image 45 - Attaching VPC to the Lambda function

Created filesystem should be added in the section labelled "EFS." Local mount paths must be defined with the source directory '/mnt/' at the start after choosing the EFS and Access point.

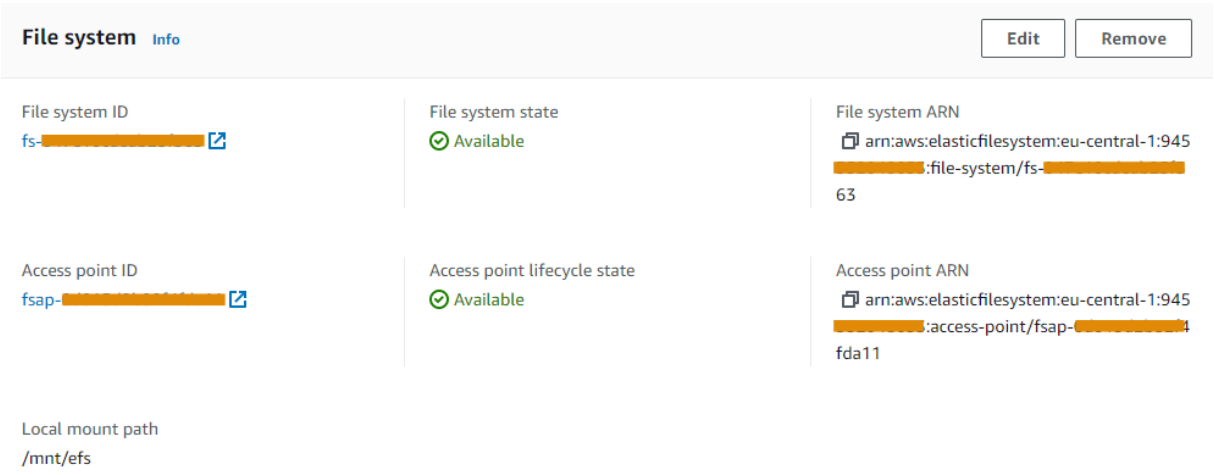

Image 46 - Attaching EFS to the Lambda function

9.5 React application – web interface

After completing all AWS service configurations, the backend component is now complete. The only thing left to do is to build a straightforward React application that can send an API request to the access point (Lambda endpoint) and display the responses. The screenshot below displays a very basic application interface.

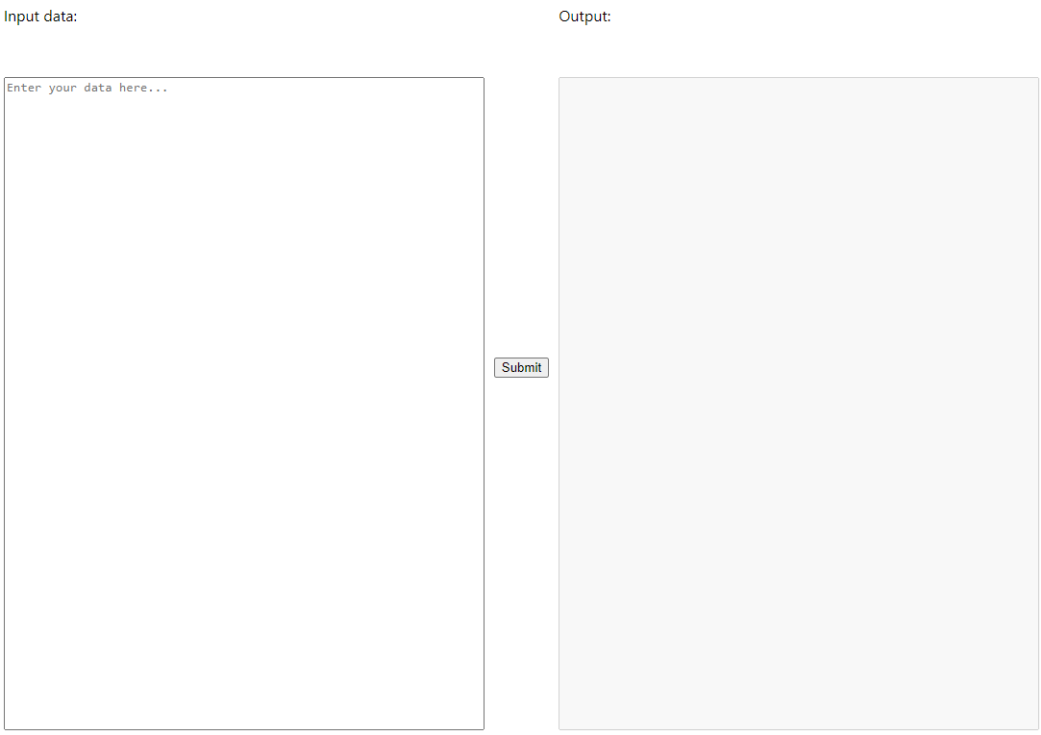

Image 47 - Web interface from the simple React application

Image 48 - Example of performing 'list' action for 'car' reference file

Once the backend and frontend components are finished, the project can be generally considered finished. Evidently, the main focus was on the code and backing services to operate as intended; the web frontend only exists to provide an interface where the data masking functionality can be tested.

9.6 Complete Lambda function code

```
import json, base64, sys, datetime, logging, threading, os, csv, base64
from operator import imod
from time import sleep, time
from importlib import import_module, reload
from os.path import exists
from pathlib import Path
from libelle.products.ldm.worker.anonymizer import Anonymizer
from anon_actions import Anonymization

def lambda_handler(event, context):

    myAnonAction = Anonymization()
    result = ""

    data = json.loads(event["body"])

    action = data['action']
    object = data['object']
    anon_data = data['data']

    anon_settings_type = data['anon_settings']['type']
    anon_settings_id = data['anon_settings']['id']
    anon_settings = data['anon_settings']

    references_type = data['references']['type']
    references_id = data['references']['id']
    references = data['references']['references']
    references_set = data['references']

    anon_key_type = data['anon_key']['type']
    anon_key_id = data['anon_key']['id']
    anon_key_value = data['anon_key']['value']

    encryption = data['encryption']

    if encryption == "base64":
        action, object, anon_data, anon_settings_type, anon_settings_id,
anon_settings, references_type, references_id, references, anon_key_type,
anon_key_id, anon_key_value = myAnonAction.decode(
            action, object, anon_data, anon_settings_type, anon_settings_id,
anon_settings, references_type,
            references_id, references, anon_key_type, anon_key_id, anon_key_value)
    elif encryption == "none":
        pass
    else:
        result = "Wrong encryption type. Please choose: 'base64' or 'none'"

    if action == 'anon':

        if anon_data != "":
            myAnonAction.add_anon_settings(anon_settings_type, anon_settings_id,
anon_settings)
            myAnonAction.add_references(references_type, references_id, references)
            myAnonAction.add_anon_key(anon_key_type, anon_key_id, anon_key_value)
            myAnonAction.anonymize_data(anon_settings_id, anon_data)
            result = anon_data
        else:
            result = 'Data for the anonymization not provided.'
```

```
elif action == 'list':
    if object != '':
        if object == 'anon_settings':
            result = myAnonAction.list_anon_settings(object)
        elif object == 'reference_file':
            result = myAnonAction.list_reference_files(object)
        elif object == 'reference_settings':
            result = myAnonAction.list_reference_settings(object)
        elif object == 'anon_key':
            result = 'Not yet fully functional.'
        else:
            result = 'Wrong object input. '
    else:
        result = 'No object input. '

elif action == 'get':
    if object != '':
        if object == 'anon_settings':
            result = myAnonAction.get_anon_settings(object, anon_settings_id)
        elif object == 'reference_file':
            result = myAnonAction.get_reference_files(object, references_id)
        elif object == 'reference_settings':
            result = myAnonAction.get_reference_settings(object, references_id)
        elif object == 'anon_key':
            result = 'Not yet functional.'
        else:
            result = 'Wrong object input. '
    else:
        result = 'No object input. '

elif action == 'delete':
    if object != '':
        if object == 'anon_settings':
            result = myAnonAction.delete_anon_settings(object, anon_settings_id)
        elif object == 'reference_file':
            result = myAnonAction.delete_reference_file(object, references_id)
        elif object == 'reference_settings':
            result = myAnonAction.delete_reference_settings(object, references_id)
        elif object == 'anon_key':
            print("Not available")
        else:
            result = 'Wrong object name input. '
    else:
        result = 'No object input. '

elif action == 'set':
    if object != '':
        if object == 'anon_settings':
            result = myAnonAction.set_anon_settings(object, anon_settings_id,
anon_settings)
        elif object == 'reference_settings':
            result = myAnonAction.set_reference_settings(object, references_id,
references_set, references)
        elif object == 'reference_file':
            result = myAnonAction.set_reference_file(object, references_id,
references)
        elif object == 'anon_key':
            result = 'Not yet fully functional.'
        else:
            result = 'Wrong object input.'
    else:
        result = 'No object input.'
elif action == 'login':
    print("Not available")
else:
    result = 'Wrong action input. '

return result
```

10. Conclusion

A thorough examination of technologies and available software on the market resulted in the development of an inventive data masking solution. For implementing the data anonymization process, a serverless architecture with Function-as-a-Service approach has emerged as the ideal option. As a result, the product is simple to understand and use, which is a significant feature of today's software. To rephrase, the focus of this paper was to make the core functionality of data marking serverless available. Aside from anonymization, there is still work to be done in the future. For example, implementing token handling for security reasons and supporting multi-user access to the Lambda function. In addition, defining and implementing a business model for the Lambda function, such as pay-per-user and other principles, should be accomplished.

To conclude, the implementation of specific cloud solutions can improve and optimise the operation of a wide range of product areas. The serverless data masking solution is a great example of how cloud services can make developers' lives easier. It is obvious that understanding a small part of the infrastructure, such as the filesystem, some networking, cloud services, and others, was required for the creation of this project, but the majority of the work was done by a third party. As a result, the development process for the company and its developers is simplified, allowing them to concentrate on developing the business model and the product logic that will ultimately determine how the company will make profit.

11. Bibliography

[1] ‘Data masking’, *Wikipedia*. Jul. 27, 2022. Accessed: Aug. 15, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Data_masking&oldid=1100735503

[2] ‘Libelle DataMasking | GDPR-compliant data’, *Libelle AG*. https://www.libelle.com/products/datamasking (accessed Aug. 10, 2022).

[3] Delphix, ‘Data Masking’, *Delphix*. https://www.delphix.com/glossary/data-masking (accessed Aug. 15, 2022).

[4] H. Varshney, ‘What is Data Masking? 5 Key Types and Techniques’, *Learn | Hevo*. https://hevodata.com/learn/data-masking/ (accessed Aug. 15, 2022).

[5] ‘What is Data Masking? | Techniques & Best Practices | Imperva’, *Learning Center*. https://www.imperva.com/learn/data-security/data-masking/ (accessed Aug. 15, 2022).

[6] S. Wickramasinghe, ‘What’s Data Masking? Types, Techniques & Best Practices’, *BMC Blogs*. https://www.bmc.com/blogs/data-masking/ (accessed Aug. 16, 2022).

[7] ‘Libelle IT Group’. https://www.libelle.com (accessed Aug. 22, 2022).

[8] ‘What is SAP? | Definition and Meaning’, *SAP*. https://www.sap.com/croatia/about/company/what-is-sap.html (accessed Aug. 22, 2022).

[9] ‘SAP’, *Wikipedia*. Aug. 13, 2022. Accessed: Aug. 22, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SAP&oldid=1104205660

[10] ‘DataMasque | Solution’. https://datamasque.com/solution/ (accessed Aug. 18, 2022).

[11] ‘Maskopy : DevOps : Open Source : FINRA Technology’. https://finraos.github.io/maskopy/#related-products (accessed Aug. 18, 2022).

[12] ‘Maskopy : DevOps : Open Source : FINRA Technology’. Accessed: Aug. 18, 2022. [Online]. Available: https://finraos.github.io/maskopy/docs.html

[13] M. T. Dimakopoulos Dimitris Tsitsigkos and Nikolaos, ‘Documentation | Amnesia - Data anonymization made easy’. https://amnesia.openaire.eu/about-documentation.html (accessed Aug. 18, 2022).

[14] Amnesia, ‘Amnesia | Home (Demo version)’. https://amnesia.openaire.eu/amnesia/index.html (accessed Aug. 19, 2022).

[15] ‘Introduction to dynamic data masking | BigQuery | Google Cloud’. https://cloud.google.com/bigquery/docs/column-data-masking-intro (accessed Aug. 19, 2022).

[16] ‘Data Masking Overview’, *Oracle Help Center*. https://docs.oracle.com/en/cloud/paas/data-safe/udscs/data-masking-overview.html#GUID-5A68E40F-1193-425C-B1FB-5FEE027EF781 (accessed Aug. 19, 2022).

[17] ‘What is monolithic architecture in software?’, *WhatIs.com*. https://www.techtarget.com/whatis/definition/monolithic-architecture (accessed Sep. 08, 2022).

[18] A. Kharenko, ‘Monolithic vs. Microservices Architecture’, *Medium*, Apr. 10, 2018. https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59 (accessed Sep. 08, 2022).

[19] raman_257, ‘Monolithic vs Microservices architecture’, *GeeksforGeeks*, Mar. 24, 2020. https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/ (accessed Sep. 08, 2022).

[20] R. Krajewski, ‘Serverless vs Microservices Architecture: What Should You Choose For Your Product?’ https://www.ideamotive.co/blog/serverless-vs-microservices-architecture (accessed Sep. 08, 2022).

[21] ‘Cloud: IaaS vs PaaS vs SaaS vs DaaS vs FaaS vs DBaaS’. https://brainhub.eu/library/cloud-architecture-saas-faas-xaas (accessed Aug. 22, 2022).
[22] ‘IaaS vs PaaS vs SaaS’. https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas (accessed Aug. 22, 2022).
[23] N. Suryavanshi, ‘What are Cloud Computing Services [IaaS, CaaS, PaaS, FaaS, SaaS]’, *Medium*, Nov. 09, 2017. https://medium.com/@nnilesh7756/what-are-cloud-computing-services-iaas-caas-paas-faas-saas-ac0f6022d36e (accessed Aug. 22, 2022).
[24] ‘What is Function-as-a-Service (FaaS)?’, *Cloudflare*. https://www.cloudflare.com/learning/serverless/glossary/function-as-a-service-faas/ (accessed Aug. 22, 2022).
[25] ‘FaaS, PaaS, and the Benefits of the Serverless Architecture’, *InfoQ*. https://www.infoq.com/news/2016/06/faas-serverless-architecture/ (accessed Aug. 22, 2022).
[26] ‘Event-Driven Architecture’, *Amazon Web Services, Inc.* https://aws.amazon.com/event-driven-architecture/ (accessed Aug. 22, 2022).
[27] M. Kibenko, ‘Monolith vs Microservices vs Serverless and what to choose for your business needs’, *NI Tech Blog*, Jun. 24, 2020. https://medium.com/ni-tech-talk/monolith-vs-microservices-vs-serverless-and-what-to-choose-for-your-business-needs-49d58b9e91f1 (accessed Aug. 16, 2022).
[28] ‘A Comparison of Serverless Function (FaaS) Providers’, *Fauna*. https://fauna.com/blog/comparison-faas-providers (accessed Aug. 17, 2022).
[29] ‘What is AWS Lambda? - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (accessed Aug. 17, 2022).
[30] ‘Lambda concepts - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-function (accessed Aug. 17, 2022).
[31] ‘What is AWS Lambda? - AWS Lambda - accessing’. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html#accessing (accessed Aug. 19, 2022).
[32] ‘What is AWS Lambda? - AWS Lambda - pricing’. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html#pricing (accessed Aug. 19, 2022).
[33] ‘Serverless Computing – AWS Lambda Pricing – Amazon Web Services’, *Amazon Web Services, Inc.* https://aws.amazon.com/lambda/pricing/ (accessed Aug. 19, 2022).
[34] ‘Prerequisites - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/dg/lambda-settingup.html (accessed Aug. 22, 2022).
[35] ‘Lambda concepts - AWS Lambda - function’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-function (accessed Aug. 23, 2022).
[36] *Lambda concepts - AWS Lambda - trigger*. Accessed: Aug. 23, 2022. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-trigger
[37] ‘Lambda concepts - AWS Lambda - event’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-event (accessed Aug. 23, 2022).
[38] ‘Lambda concepts - AWS Lambda - execution environment’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-ee (accessed Aug. 23, 2022).

[39] ‘How Lambda fits into the event-driven paradigm - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/operatorguide/lambda-event-driven-paradigm.html (accessed Aug. 23, 2022).
[40] ‘Lambda concepts - AWS Lambda - set architecture’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-arch (accessed Aug. 23, 2022).
[41] ‘Lambda concepts - AWS Lambda - deployment package’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-dp (accessed Aug. 23, 2022).
[42] ‘Lambda concepts - AWS Lambda - runtime’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-runtime (accessed Aug. 23, 2022).
[43] ‘Lambda concepts - AWS Lambda - layer’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-layer (accessed Aug. 23, 2022).
[44] ‘Lambda concepts - AWS Lambda - extensions’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-extensions (accessed Aug. 23, 2022).
[45] ‘Lambda concepts - AWS Lambda - concurrency’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-concurrency (accessed Aug. 23, 2022).
[46] ‘Lambda concepts - AWS Lambda - destinations’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-destinations (accessed Aug. 23, 2022).
[47] ‘Lambda features - AWS Lambda - scaling’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-scaling (accessed Aug. 23, 2022).
[48] ‘Lambda features - AWS Lambda - concurrency controls’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-concurrency (accessed Aug. 23, 2022).
[49] ‘Lambda features - AWS Lambda - function URLs’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-urls (accessed Aug. 23, 2022).
[50] ‘Lambda features - AWS Lambda - asynchronous invocation’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-async (accessed Aug. 23, 2022).
[51] ‘Lambda features - AWS Lambda - event source mapping’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-eventsourcemapping (accessed Aug. 23, 2022).
[52] ‘Lambda features - AWS Lambda - destinations’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-destinations (accessed Aug. 23, 2022).
[53] ‘Lambda features - AWS Lambda - blueprints’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-blueprints (accessed Aug. 23, 2022).
[54] ‘Lambda features - AWS Lambda - testing and deployment tools’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-tools (accessed Aug. 23, 2022).

[55] ‘Lambda features - AWS Lambda - application templates’. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html#gettingstarted-features-templates (accessed Aug. 23, 2022).
[56] ‘Lambda function handler in Python - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html (accessed Aug. 23, 2022).
[57] ‘Lambda function handler in Python - AWS Lambda - naming’. https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html#naming (accessed Aug. 23, 2022).
[58] ‘Lambda function handler in Python - AWS Lambda - how it works’. https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html#python-handler-how (accessed Aug. 23, 2022).
[59] ‘Lambda function handler in Python - AWS Lambda - handler return’. https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html#python-handler-return (accessed Aug. 23, 2022).
[60] ‘Lambda function handler in Python - AWS Lambda - python example’. https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html#python-example (accessed Aug. 23, 2022).
[61] ‘14 Uncommon AWS Lambda Use Cases for Serverless Project: TechMagic’, *Blog | TechMagic*, Oct. 01, 2020. https://www.techmagic.co/blog/14-use-cases-for-aws-lambda-how-to-make-the-most-of-it/ (accessed Aug. 15, 2022).
[62] ‘Amazon Simple Notification Service (SNS) | Messaging Service | AWS’, *Amazon Web Services, Inc.* https://aws.amazon.com/sns/ (accessed Aug. 15, 2022).
[63] ‘9 Killer Use Cases for AWS Lambda’, *Contino | Global Transformation Consultancy*. https://www.contino.io/insights/aws-lambda-use-cases (accessed Aug. 15, 2022).
[64] M. Samarasinghe, ‘AWS Lambda Best Practices’, *Medium*, Jul. 12, 2021. https://aws.plainenglish.io/aws-lambda-best-practices-7454da49314d (accessed Aug. 15, 2022).
[65] ‘What is an API call?’, *Cloudflare*. https://www.cloudflare.com/learning/security/api/what-is-api-call/ (accessed Aug. 25, 2022).
[66] ‘What is REST’, *REST API Tutorial*. https://restfulapi.net/ (accessed Aug. 25, 2022).
[67] ‘API vs REST API Simplified: 6 Critical Differences’, *Learn | Hevo*. https://hevodata.com/learn/api-vs-rest-api/ (accessed Aug. 25, 2022).
[68] ‘AWS Lambda Function URLs vs. Amazon API Gateway | Serverless Guru’. https://www.serverlessguru.com/blog/aws-lambda-function-urls-vs-amazon-api-gateway (accessed Aug. 25, 2022).
[69] ‘Amazon API Gateway | API Management | Amazon Web Services’, *Amazon Web Services, Inc.* https://aws.amazon.com/api-gateway/ (accessed Aug. 25, 2022).
[70] ‘Lambda function URLs - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/dg/lambda-urls.html (accessed Aug. 25, 2022).
[71] ‘High-Performance Block Storage – Amazon EBS – Amazon Web Services’, *Amazon Web Services, Inc.* https://aws.amazon.com/ebs/ (accessed Aug. 25, 2022).
[72] ‘EBS vs EFS vs S3 – when to use AWS’ three storage solutions’, *Just After Midnight*. https://www.justaftermidnight247.com/insights/ebs-efs-and-s3-when-to-use-awss-three-storage-solutions/ (accessed Aug. 25, 2022).
[73] ‘Amazon EFS’, *Amazon Web Services, Inc.* https://aws.amazon.com/efs/ (accessed Aug. 25, 2022).
[74] ‘Amazon EFS performance - Amazon Elastic File System’. https://docs.aws.amazon.com/efs/latest/ug/performance.html (accessed Aug. 25, 2022).

[75] ‘Deploy Python Lambda functions with .zip file archives - AWS Lambda’. https://docs.aws.amazon.com/lambda/latest/dg/python-package.html (accessed Aug. 24, 2022).
[76] ‘Postman API Platform | Sign Up for Free’, *Postman*. https://www.postman.com/ (accessed Aug. 26, 2022).

List of tables

List of images