Degree Programme in Computer Engineering
First Cycle 15 credits

# Securing Data in a Cloud Environment: Access Control, Encryption, and Immutability

Säkerhetshantering av data som överförs genom molnbaserade tjänster: åtkomstkontroll, kryptering och omutlighet

Ahmad Al Khateeb
Abdulrazzaq Summaq

STOCKHOLM, SVERIGE 2023

# Securing Data in a Cloud Environment: Access Control, Encryption, and Immutability

Säkerhetshantering av data som överförs genom molnbaserade tjänster: åtkomstkontroll, kryptering och omutlighet

Ahmad Al Khateeb
Abdulrazzaq Summaq

Bachelor Thesis in
Computer Science, 15 hp

Supervisor at KTH: Reine Bergström
Examiner: Ibrahim Orhan
TRITA-CBH-GRU-2023:088

KTH
Royal Institute of Technology
141 52 Huddinge, Sweden

# Sammanfattning

Mängden av data och utvecklingen av banbrytande teknologier som idag används av alla samhällsbärande organisationer ökar drastiskt. I samma takt ökar dataintrång, cyberattacker och dess förödande konsekvenser samt antalet personer och organisationer som utgör potentiella offer för sådana typer av attacker. Detta ställer högre krav på säkerheten när det gäller att skydda data mot cyberattacker, men även att kontrollera åtkomsten till data som autentiserade användare vill komma åt. Rapporten fokuserar på att studera hur data säkras i GitLab-baserade molnsystem. Syftet med detta arbete är att ge svar på frågeställningar som till exempel att lova säker åtkomst och skydd för data från obehörig åtkomst och ändringar. Arbetet bakom detta projekt inkluderade undersökning av tekniker som används inom accesskontroll, datakryptering och data-omutlighet. Studien resulterade i en implementation som möjliggör att hämta signerade ändringar (Commits) från GitLab, verifiera användaridentiteten och åtkomstbehörighet, hantera dataåtkomst samt presentera resultaten. Resultaten av detta examensarbete demonstrerar effektiviteten av den implementerade säkerhetsteknikerna i att skydda data och kontrollera access.

## Nyckelord

Access Control, Authorization, Keycloak, GPG Keys, Encryption, GitLab, Version Control, Neo4j, Data Security.

# Abstract

The amount of data and the development of new technologies used by all society-critical organizations are increasing dramatically. In parallel, data breaches, cyber-attacks, and their devastating consequences are also on the rise, as well as the number of individuals and organizations that are potential targets for such attacks. This places higher demands on security in terms of protecting data against cyber-attacks and controlling access to data that authenticated users want to access. The paper focuses on studying concepts of secure data practices in a GitLab-based cloud environment. The objective is to give answers to questions such as how to ensure the guarantee of secure data and protect it from unauthorized access and changes. The work behind this thesis includes exploring techniques for access control, data encryption, and data immutability. The study is followed by an implementation project that includes fetching code from GitLab verifying user identity and access control, managing data access, and displaying the results. The results of the thesis demonstrate the effectiveness of the implemented security measures in protecting data and controlling access.



## Keywords

Access Control, Authorization, Keycloak, GPG Keys, Encryption, GitLab, Version Control, Neo4j, Data Security.

# Acknowledgements

This is a bachelor's thesis done by two students in the last year of the Bachelor of Science in Computer Engineering at the Royal Institute of Technology KTH. This paper is a result of the thesis project within the field of computer engineering at KTH, the Royal Institute of Technology.

We would like to express our deepest gratitude to our supervisor *Reine Bergström* for his guidance, support, and expertise throughout our thesis project. We are grateful for his insightful feedback and constructive criticism which have consistently pushed us to strive for excellence. His mentorship and encouragement have inspired us to surpass our expectations.

# Table of Contents

# 1 Introduction

With the increasing amount of data being generated and stored by organizations, data security has become a critical concern. Data breaches and cyber-attacks can have significant consequences, including financial losses, reputational damage, and legal liabilities. In addition to these data breaches, in the modern world we live in there are other risks that may be caused by users who have access to a system, but who are not allowed to access data in that system. Thus, it is essential to ensure that data is secured against unauthorized access and modification, especially when large amounts of data are increasingly stored and managed remotely in cloud-based servers. To mitigate these risks, organizations must implement robust data security measures to protect their data from unauthorized access.

The report is organized in the following structure: The *Theory and Background* chapter provides a theoretical foundation for understanding concepts of Access Control, encryption, and data immutability and how they can be applied to secure data. It also provides an overview of technologies used in this project, such as GitLab and a graph database (Neo4j) and their respective functionalities. The *Method and Result* chapter presents best practices and methods for securing data in a GitLab environment, starting from those technologies introduced in the theory and background chapter, and giving an overview of implementing approaches used during the thesis process. The *Analysis and Discussion* chapter evaluates the implementation results and assesses the effectiveness of the implemented security measures and access control mechanisms. Finally, the *Conclusion* chapter addresses the initial questions and problem introduced in the beginning, it also summarizes the findings of this research.

## 1.1 Problem Description

Today, cloud-based solutions like GitLab are increasingly being used in modern enterprises to manage code and software development projects. The adoption of cloud-based platforms, however, prompts questions regarding to what extent the data kept on these platforms are secure. Companies are required to make sure that their data is protected from illegal access and modification.

GitLab is a popular tool for managing software development projects. It offers features such as version control, code review and continuous integration and delivery. However, with the increasing use of GitLab for development projects, data security has become a major concern. The platform requires strict data access control to prevent unauthorized access to sensitive data, such as source code and project files, but also data from external sources that GitLab projects try to access.

The *KTH Royal Institute of Technology* in *Flemingsberg* has an ongoing project of developing its GitLab-based cloud system, which enables uploading, downloading, and executing of software programs. These programs may need, when running, to access sensitive data from a separate data source. This places great demands on the security of the system. There is a need to investigate the strategies and tactics for protecting data in a cloud environment powered by GitLab-based systems.

The main problem addressed in this thesis is how to secure data in a GitLab environment. Specifically, the thesis aims to investigate the best practices for access control, encryption, and immutability to protect data from unauthorized access. The thesis will also examine how to ensure data integrity and immutability so that data cannot be accessed or changed without authorization.

The thesis will address the following research questions:

1. How can access control be implemented to prevent unauthorized access to data?
2. What are the different methods of data encryption, and which is most effective in a GitLab environment?
3. How can data integrity and immutability be ensured in a Cloud environment?
4. How can data be protected during execution and display, and what security measures should be put in place?
5. How fetching data and controlling data access should be handled in terms of using the Neo4j database.

   This project also seeks to address the answer of what techniques of access control are used in GitLab-based cloud environments, where applications developed in GitLab may have access and execute on data fetched from external databases (which e.g. could be graph database Neo4j).

The answers to these research questions will provide insights into the best practices for securing data used in a GitLab environment. The thesis aims to develop a proof of concept that demonstrates how to implement these practices in a real-world environment. The implementation results will provide insights into the effectiveness of the implemented security measures and access control mechanisms. Moreover, the thesis aims to contribute to the field of data security by providing practical and theoretical insights into how to secure data in a GitLab environment.

## 1.2 Goals

The main goal of this thesis is to investigate different technologies used for securing data, such as access control, data encryption, and data immutability. Another goal of this project is to study how such technologies can be implemented in a GitLab-based cloud environment, and how different types of databases, especially graph databases, should be taken into concern when implementing these security approaches. The more specific goals that this study aims to achieve are:

1. Provide an analysis of different methods of protecting data, such as access control, encryption, and data immutability.
2. Identify best practices for securing data in a GitLab environment.
3. Identify controlling data access using Neo4j.
4. Develop a software solution that:
   a. Demonstrates the implementation of these techniques in a GitLab environment.
   b. Enables secure access to the system.
   c. Ensures that data is not modified by unauthorized users.

The research will provide insights into best practices for securing data in a GitLab environment, which will benefit organizations using GitLab for their development project, as well as organizations having data in cloud storage systems. In addition, the study will provide recommendations on how to ensure data protection and access control in a secure manner, which can be used as a guide for other similar projects.

## 1.3 Boundaries and Delimitations

This is a Bachelor's thesis that extends over approximately 10 weeks, where part of the time is spent on gathering relevant scientific facts as well as delving deeper into previous work that has been done in the same area. Considering the period that the implementation work will be taking place, there will not be a space for large-scale work in all mentioned areas. Additionally, some of the techniques that will be used in the implementation are going to be dependent on the choice of other technologies in the same system that other student groups are investigating in parallel with this project.

## 1.4 Method

The method and the approach used in this thesis include a combination of a literature study as well as software development. The literature study includes investigating existing literature about data protection, access control, encryption, and data immutability. This will provide a theoretical framework to understand different methods and best practices used for data security.

The approach of the implementation includes developing software that shows how to secure data and protect it from unauthorized access. The methodology also includes an analysis of the implementation results, by examining how well the implementation meets the security and access control objectives.

## 1.5 The Authors' Contribution to the Thesis

The investigation, analysis, and conclusions contained in this report are the result of the authors' work on this thesis. The authors' research, which was carried out in partnership with the selected faculty at the *KTH Royal Institute of Technology*, will serve as the foundation for the system's design and implementation proposal.

The research method used in this thesis provides a theoretical and practical understanding of how to secure data in a GitLab environment. The literature study will provide a theoretical foundation for understanding the best practices and methods for securing data, while the software development will provide a practical demonstration of how to implement these practices in a real-world environment. The analysis of the implementation results will provide insights into the effectiveness of the implemented security measures and access control mechanisms.

Abdulrazzaq Summaq focused mostly on developing the server application that would represent the cloud solution, that would host all features needed to fulfil the project goals. Ahmad Al Khateeb tested the integrability of the different chosen technologies that were planned to be used in the server application.

# 2 Theory and Background

Data security is an essential aspect of information technology and is becoming increasingly important as more data is generated and stored by organizations. Data security involves protecting data from unauthorized access, modification, and destruction. The study behind this thesis includes important aspects to take into concern when developing a secure computer system, aspects such as Zero Trust, encryption, access control, and immutability. This chapter will cover all three concepts in a manner related to self-managed GitLab environments, including the definition of theoretical concepts related to this thesis as well as previous work in the area. Other technologies such as Neo4j are also introduced in this chapter. Finally in the chapter, some of the security threats that many organizations are facing today are highlighted.

## 2.1 Zero Trust

The expansion of the internet, globalization, and the shift to cloud computing has caused a need for adapting security measures that meet those security threats coming with the new technologies. The traditional perimeter-based security that is based on implicit trust for users in systems has, in many cases, proven limited and vulnerable when it comes to remote work and the use of third-party applications. *Thomas Yacob* [12] introduced the concept of Zero Trust in his paper as an alternative approach, emphasizing the importance of validation rather than blind trust towards the system users.

The author describes Zero Trust as a cybersecurity paradigm shift, from perimeter-based security, and that focuses on securing resources rather than network segments, considering all users and devices as potential threats [12]. The core Zero Trust principles [12] are introduced as the following:

1. Ensure all resources are accessed securely, regardless of location:
   Any data traffic passing through resources must be authorized, inspected, and secure, including encryption for traffic directed through untrusted networks.
2. Adopt the least privilege strategy and strictly enforce access control:
   Access to restricted resources is not granted, reducing vulnerabilities by limiting user access to necessary resources.
3. Inspect and log all traffic:
   Verification is emphasized, restricting user access to necessary resources, and verifying their actions.

By focusing on securing individual resources rather than relying on perimeter-based roles, Zero Trust provides a higher level of protection and its popularity is increasing [12]. It ensures that all resources are accessed securely, regardless of the user's location, and enforces a least privileges strategy, giving access only to "right" resources. On the other hand, there are some drawbacks to using Zero Trust's least privileges strategy, such as the complexity appearing during the migration process from traditional perimeter-based security, where detailed information about assets, actors and processes needs to be carefully planned. Moreover, there is a potential impact on user experience and productivity since users need to go through additional authentication and verification steps.

## 2.2 Encryption

Encryption is the process of converting data into a secure form that can only be accessed with a decrypting key. Encryption is used to protect data during transmission and storage. There are two main types of encryption: symmetric encryption and asymmetric encryption. Symmetric encryption uses the same key for encryption and decryption. Asymmetric encryption uses a public key for encryption and a private key for decryption. In this chapter, cryptography technologies used in Keycloak are presented.

The advancement of technology and networks has made safe communication and protection against attacks a significant concern. In the early days of the Internet, organizations stored their data within the enterprise perimeter, which was highly fortified, making it difficult for attackers to breach [3]. However, technology advancements have made it possible to access data outside the perimeter, leading to the development of the de-parameterization concept by *Jon Measham* and the Jericho Forum Commandments. *Paul Simmonds* later designed the concept of de-parameterization without a tightened perimeter, which led to the renewed Zero Trust model. The Zero Trust model was first introduced by *Forrester Research*' analyst *John Kindervag* [3].

The Zero Trust cybersecurity paradigm, introduced in 2.1, emphasizes protecting resources by not implicitly granting trust and frequently evaluating access [12]. Legacy Privileged Access Management (PAM) systems were designed to work for systems and resources inside an enterprise network, but with the rise of cloud computing [3], PAM faces challenges. An attacker can easily compromise the entire system by gaining access to the network through social engineering, a trusted user's negligence, or innocence. Despite spending an estimated $137 billion on security technologies in 2019, many organizations experience data breaches. Zero Trust aims to protect people, property, and infrastructure from potential threats to enterprise or organization data. This paper [3] provides a detailed summary of Zero Trust, its evolution, its status, and how it reshapes the future trust landscape.

Furthermore, the authors of [3] present a model that assumes that no interface can be trusted and that all network traffic is untrusted. The Zero Trust model uses multiple authentication steps, including multi-factor authentication, to verify a user's identity. Network segmentation, using a network segmentation gateway (SG) and Microcore and Perimeter (MCAP), is important to the Zero Trust model. The Data Acquisition Network (DAN) is used to log all network traffic, and strict access control policies are implemented as well. Additionally, the article [3] defines a theory regarding Zero Trust that recognizes all data sources and computational services as resources, secures communication regardless of network location, grants access to resources only on a session basis, and monitors the integrity and security of the enterprise.

Another important aspect of Zero Trust is that it periodically verifies the authenticity and authority of devices, applications, and users to provide maximum security for protected personal information. The authors in [2] propose a Zero Trust model for cloud computing with practical experimentation, which enforces strict access control and maintains logs of all activities within the network. The architecture in [2] consists of a client, proxy server, authentication and authorization server, access control, and application, all of which are implemented within the Kubernetes container platform. The authentication and authorization server acts as a mediator between the proxy server and the application and continually checks for certificates to ensure no compromised user enters the system. The access control uses a

hybrid of Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) to provide specific access authority to clients within an organization [2].

According to [2], the authors describe the architecture for user authentication and access control, using various tools and services, such as Ubuntu, OpenID Connect, Kubernetes, Docker, Keycloak, React, Nginx, Squid proxy, and Mozilla Firefox. The system uses control groups, Docker CE, XACML, RBAC, OpenID Connect, and SSO for authentication and authorization. It also uses logging and algorithmic flow to ensure Zero Trust [2].

There are different types of cyber-attacks, including infection-based attacks, explosion attacks, probe attacks, cheating attacks, traverse attacks, and concurrency attacks. The text also provides examples of how to defend against each type of attack, such as using Zero Trust to create a secure perimeter and applying health monitoring within Kubernetes to defend against explosion attacks. To defend against traverse attacks, the authentication service tracks every logged-in user activity and suspends any user who changes their IP address or header agent more than twice. The proxy server is equipped to handle concurrency attacks by dropping rapid packets with flood drop thresholds [2].

Furthermore, the authors in [2] discuss securing assets at various layers of the Open System Interconnect (OSI) model. The Application Layer emphasizes multifactor authentication (MFA) with a combination of username, password, and One Time Password (OTP). The Presentation Layer encrypts data using sophisticated encryption algorithms and implements a combination of Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) for access control. The Session Layer maintains secure connections and terminates sessions if there are any discrepancies. The Transport Layer is secured by disabling any open ports and using proxy servers, while the Network Layer uses Calico to define network policies. The Data Link Layer and Physical Layer are secured through redundant power supplies, NIC cards, and Ethernet cables to ensure immediate availability during a failure.

Some of the advantages of implementing a Zero Trust Architecture (ZTA) in an organization include strong user authentication and access policies, data segmentation, reduced vulnerability, data protection, and security orchestration. However, there are some disadvantages as well coming with the ZTA, such as the tedious effort and time-consuming nature of implementing ZTA, versatile management of dynamic users, managing multiple devices, complex application management, and meticulous data security [2]. More about other encryption technologies are introduced in 2.5.1.

Additionally, since many of today's computer systems and platforms are increasingly concerned about protecting data in multiple layers, built-in encryption techniques and having support for them have become a must. Some of these encryption techniques are SSH keys, GPG keys which are, from previous relevant work [12], recommended to be used, and X.509 certificates. All of these cryptographical models are supported by a huge number of platforms such as GitLab, which will be introduced and discussed in 3.4. However, research and development of new encryption techniques will have to go faster, as some of the current encryption techniques can be threatened by smarter devices, such as quantum computers.

### 2.2.1 Quantum Computers and Today's Cryptography

Quantum computers are a new type of computer that uses quantum technology, which eventually can be a risk for today's cryptography [3]. Indeed, running quantum computers has not yet been realized, but researchers have suggested that there will be a reality in the near future. While quantum computers could, in theory, break current public-key cryptography, such as RSA and DSA, other cryptographic systems, such as hash-based cryptography, code-based cryptography, lattice-based cryptography, multivariate-quadratic-equations cryptography, and secret-key cryptography, are believed to resist quantum attacks.

Quantum computers will have a different impact on cryptography depending on the cryptography technologies used to encrypt data. Traditional public-key systems such as RSA rely on mathematical problems that are solvable on classical computers. Quantum computing can break these mathematical problems, rendering traditional public-key systems useless.

However, there are examples of public-key signature and encryption systems that would be difficult to break, even for a cryptanalyst armed with a quantum computer, where i.e. larger key sizes are required, making them less efficient than traditional public-key systems like RSA. On the other hand, there is still a need for post-quantum cryptography [3].

Some of the public-key signature systems used to encrypt and decrypt data are a hash-based public-key signature system and a code-based public-key encryption system [3]. Both of these systems are examples of public-key cryptography, where different keys are used for encryption and decryption, or generation and verification. The encryption or signature generation key is public, while the decryption or verification key is kept secret. In the first system, matrix multiplication is used for encryption. The receiver generates a public key, which is a matrix with coefficients, that can be used to encrypt messages. To decrypt the ciphertext, the receiver uses a "hidden Goppa code" structure to undo the matrix multiplication in a reasonable amount of time [3]. In the second system, multivariate-quadratic polynomials are used for signature generation. The signer generates a public key, which is a sequence of polynomials with coefficients, that can be used to verify signatures. To generate a signature, the signer finds roots of a secret low-degree univariate polynomial over a field. The secret polynomial is chosen to have a certain structure, which makes it possible for the signer to solve equations in a suitable amount of time. Both systems rely on the difficulty of certain mathematical problems, such as syndrome-decoding or solving multivariate quadratic equations, to provide security. In both cases, the security of the system is based on the fact that the problems are believed to be hard for an eventual attacker to solve.

## 2.3 Access Control

Access control involves defining who can access data and what actions a legitimate user is allowed to perform [11]. In other words, access control sets limits on what an authenticated user can do, or what a program that an authenticated user executes in a computer system is allowed to do. With this definition, access control aims to prevent activities carried out by an authorized user in a computer system, which in one way or another could harm the security of the computer system. This is done with a reference monitor that, by using an authorization database, checks if a user has permission to do a specific action, as described in Figure 2.1. The system has also an auditing feature that records important activities done by users or programs.

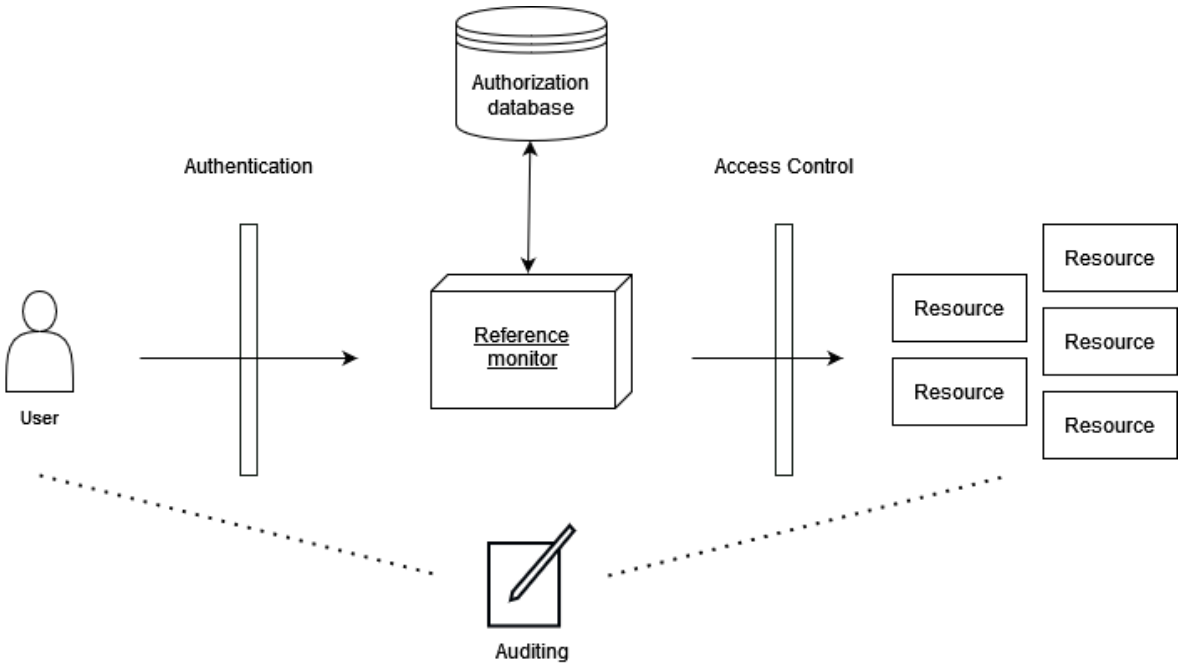


*Figure 2.1. Access Control Architecture*

Figure 2.1 shows a general picture of how access control could be implemented in a computer system. Access control is usually only one of the other important services of the security system, whereas the other services, such as authentication and encryption, together with access control techniques constitute the security unit in the system [11]. As it is shown in Figure 2.1, access control comes after the authentication filter. Moreover, the resource data might be encrypted, and the audition feature logs every single action taken by the user, so that all these services together constitute the security system.

Access control is typically implemented using access control models such as Access Control Lists (ACL), role-based access control (RBAC), and attribute-based access control (ABAC), or other modern techniques such as Keycloak, that will be introduced later in this chapter. But first, 2.3.1 will introduce Access Control List, and then it introduces the RBAC, which is a widely used access control model that assigns roles to users and specifies what actions each role can perform. ABAC is a more flexible access control model that uses attributes to define access policies [11]. Furthermore, the chapter will talk about the idea behind IAM and Keycloak as a service of it.

### 2.3.1 Access Control Technologies

#### *2.3.1.1 Access Control Lists (ACL)*

ACLs are a predetermined set of permissions that can be assigned to a specific user or group for read/write access to a specific resource, file, or repository. The file owner manages file access permissions by the creation of access control lists [19]. An access control list is a table containing usernames and the permissions assigned to each user on a single resource. For instance, a user might own a file on a GitLab environment managed by an access control system, and the user decides that he/she wants other users to access his/her file. The user might, through the ACL-management system, give another user the ability to edit the file, and a large group of users the ability to only read that file. Each of these decisions would require an entry on an access control list.

An example of an ACL management system is NTFS implemented by Windows. Some of the permissions that NTFS allows users to assign to other users on a file or a folder are:

- *Full Control*, which gives users full authority over a resource including all actions that can be performed on the resource.
- *Read permission* allows users to view the contents of the file.
- *Read & execute permission* enabling users to execute applications and executable programs.
- *Write permission* that allows users to create files and modify their contents.
- *Modify permission* that makes it possible for users to, beyond Write permission, delete files (it also includes Read & execute permissions).

This will help in streamlining the process to give administrators a faster way to give a group or user the right permission for their level of authority.

#### *2.3.1.2 Role-based access control*

Role-based access control (RBAC) is a data security model that assigns user permissions based on the roles they have been given. The role is defined as a job function with certain semantics relating to the authority given to the user [1]. RBAC makes security administration less complicated and enables organizations to examine user permissions. The RBAC reference model in ANSI/INCITS 359-2004 defines four model components: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations. The core RBAC is essential when implementing RBAC in any type of system and defines the required elements and relations in the RBAC system.

In the RBAC model, a process is defined as the image of an executable program, while an object represents the data queried by a user or a device connected to the system [1]. RBAC's management of processes and objects is dependent on the specific system in which it is being implemented. For instance, in a file system, RBAC may regulate operations such as read, write, and execute, while in a database management system, it may oversee operations like insert, delete, append, and update.

The core definition of RBAC, according to [1], is the assignment of roles and permissions to each user having a role in the system. As a result, many-to-many relationships between individual users and permissions are made by roles. Furthermore, sessions map a user to a subset of roles that the user has been given. The arrangement of this many-to-many

relationship between role-user and role permission provides flexible assignment of permissions to users and roles. This provides users with only the necessary access to resources, thus preventing unauthorized access and minimizing the privilege of users.

Central to the RBAC model is role hierarchies, which help illustrate how a specific chain of hierarchies with authority is considered in an organization [1]. Hierarchies in RBAC are represented by inheritance relations between roles. For instance, a user with role $r_1$ has all privileges (or permissions) assigned to both $r_1$ and $r_2$ if role $r_1$ inherits $r_2$. Additionally, RBAC includes a separation of duty relations, which aims to make sure that failures inside an organization are solely brought about by individual collaboration and to reduce the risk of collusion by assigning individuals with different skills to separate tasks.

However, RBAC has been criticized for being difficult to set up an initial role structure and for being inflexible to rapidly change domains. Attribute-based access control ABAC may offer a solution by using attributes and rules to replace or supplement RBAC. RBAC simplifies access control by assigning roles with permissions, which can be structured hierarchically to support efficiency. RBAC also supports separation-of-duty requirements. The RBAC standard has been revised several times and a revision is underway to extend its usefulness to more domains, particularly distributed applications [1].

### 2.3.2 Identity and access management (IAM)

Identity and access management (IAM) is a management tool policy that helps manage the users on the system and provides the right access to the environment [7]. This ensures security in the system and provides users with one interface of authentication so they can prove their identity and navigate through the different services in the environment without proving their identity multiple times.

Several IAM tools help in providing a centralized way to manage user permissions across various systems, such tools are Keycloak, Okta and more [7].

#### *2.3.2.1 Keycloak*

Keycloak is a very popular Identity and access management tool that is used to enhance the user experience for all roles of the system. Keycloak uses the access control methods mentioned in 2.1. It is built on a set of administrative UIs and a RESTful API to allow you to create permissions for your protected resources and scopes, associate those permissions with authorization policies, and enforce authorization decisions in your applications and services.

Resource servers (applications or services that serve protected resources) typically rely on information to determine whether access to a protected resource should be granted [6]. That information is typically obtained from a security token, which is typically sent as a bearer token with each request to the server for RESTful-based resource servers. When a web application relies on a session to authenticate users, that information is typically saved in the user's session and retrieved from there for each request.

There are many features to an Identity and access management tool like Keycloak. According to the research of *D.N. Divyabharathi and Nagaraj G. Cholli* [20], there are many features and the ones relevant to security are as below:

1. OpenID Connect Support
2. CORS Support

These features that Keycloak helps us establish a secure system in an IaaS (Infrastructure as a System, a cloud computing service that provides essential computing, storage and networking resources). According to the Keycloak documentation [6], Keycloak allows users to create different types of permissions such as resource-based permissions, scope based-permissions and policies that define conditions that must be met before any access is given to that user.

#### *2.3.2.2 Okta and Other IAM Tools*

There are not as many IAM tools that are as popular as Keycloak but one that comes up is Okta. Okta is a paid identity management tool that has gained popularity with many big companies that offer their software as a service [5].

## 2.4 Immutability

Data immutability is one of the most important things when implementing security in a system. That is because of multiple types of security threats facing organizations today, such as malicious insiders and human errors. For organizations, it’s essential having secure data backups to guarantee data immutability, even if the ideal approach to secure data is to limit users' access to data as much as possible. It is the property of data that prevents it from being changed once it has been created. Immutability ensures that data remains unchanged and can be trusted. It can be implemented using cryptographic hash functions that generate a unique hash for each data object. Any change to the data object will result in a different hash, indicating that the data has been tampered with.

There are different levels of data immutability. For instance, immutable data may include uneditable data that even system administrators do not have permission to edit. It may also include that some data sets can be modified by users with specific privileges via an API. However, even when data is exposed to attacks or tampering, the backups will guarantee that the data is secure. When developing systems that handle sensitive data, it is important to identify data to be immutable and the privileges needed to be able to edit mutable data sets.

GitLab uses PostgresSQL as the DBMS. To guarantee data immutability, it needs to be ensured that the data that PostgresSQL contains should only be accessed via GitLab. This means that even users with administration privileges should not have direct access to the database, but they need to access the data through GitLab. This provides the ability to redefine the permissions given to a specific user.

Postgres provides another permission that can be defined via GitLab configurations. There are two approaches to redefining immutability in GitLab. The first one is reconfiguring the PostgresSQL server included with Omnibus GitLab, and the other one is using an external PostgresSQL server.

## 2.5 Database Access Control

This section will introduce access control mechanisms used in graph databases, such as Neo4j. But first, let us define what a graph database is. Unlike traditional databases such as relational or file-based databases that store data in tables or documents, graph databases consist of nodes representing the data and relationships between these nodes [10]. Nodes, representing entities in the graph databases, are tagged with labels that correspond to table names in relational databases and properties, which are key-value pairs corresponding to a column value belonging to a row key. Moreover, the labels can also contain metadata for specific nodes. The relationship between two nodes represents the relationship between these entities' nodes, where one of them is the start node and the other is the end node. Relationships may also have properties, and the direction that it provides makes it efficient to navigate through the nodes. The most popular graph database used today is Neo4j, which will be introduced in the next section.

### 2.5.1 Neo4j

Neo4j is a graph database that is optimized for storing and querying connected data. It is an open-source, NoSQL, and native graph database, which means that data is structured in a genuine graph model, even at the storage level, and not only providing a graph abstraction. The fact that Neo4j is a native graph database, compared to other traditional none-graph databases, makes it a flexible and efficient database to use for managing relationships between data entities and handling complex connections between data nodes [10]. Neo4j provides a query language called Cypher, which is a declarative query language equivalent to SQL languages in other database systems. Cypher enables developers to query the database using graph patterns and specify patterns of nodes and relationships in the graph.

Utilizing a comprehensive security model, Neo4j provides flexibility and customizable access control handled and maintained within a dedicated database in the system known as the *system database*, where all actions on the administrative level are directed to and performed [14]. The access control technology used in Neo4j’s system database enforces Role-based access control, which was introduced in 2.1.1.2. In addition to RBAC, Neo4j uses privileges to optimize access control, customize already existing roles and create new roles, and define what access rights a user is assigned [13].

Figure 2.2 shows the hierarchical structure between different privileges used in Neo4j. These privileges are assigned to defined roles in the system. However, Neo4j makes it possible to customize each role's privileges and define new roles with needed privileges.

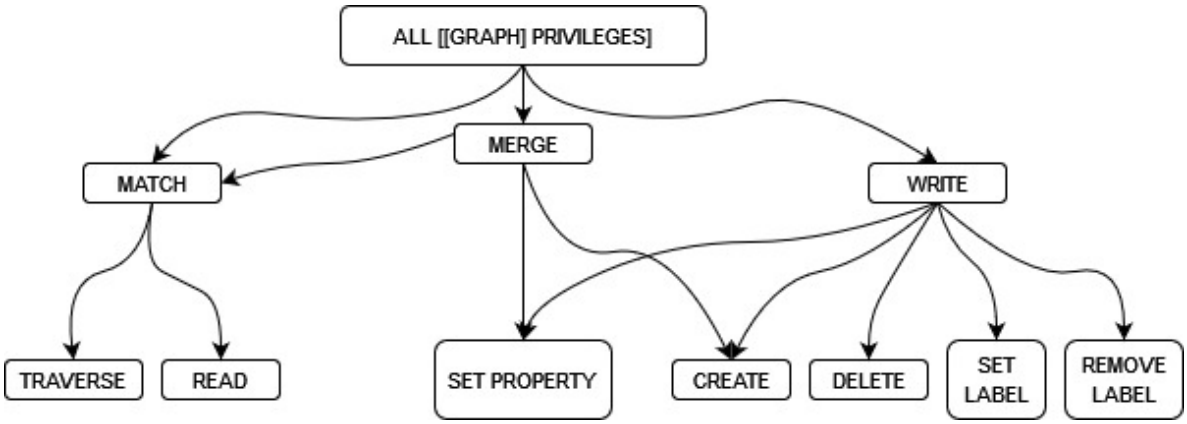


*Figure 2.2. The hierarchy between different graph privileges, inspired by figures in [13]*

Neo4j controls the user's permissions to graph elements by using the allowlist (GRANT) and denylist (DENY) mechanism [13]. These lists are used to determine whether a user has the right to access the data element, and if they have, what privileges they are given (for instance read privilege or write privilege).

Cypher, used in Neo4j, makes it possible to create new roles and assign needed privileges to these roles. Moreover, it is possible to modify the already built-in roles by adding or revoking privileges on these [13]. The built-in roles and privileges that Neo4j provides are:

1. The Public role: users with this role have access to the default database and are allowed to execute functions and other actions. Unlike other roles, the Public role cannot be revoked, but it can be modified though.
2. The reader role: users with this role are only allowed to read graph elements, but the system database (the administration database)
3. The editor role: users with this role are allowed to read and write on graph elements, but the system database. They are not allowed to create new labels or relationship types.
4. The publisher role: users with this role have the same rights as the editor role, but also can create new labels or relationship types.
5. The architect role: users with this role have the same rights as the publisher, but also can manage indexes and constraints.
6. The admin role: users with this role can do any administrative work, with all that it means from database management to set roles and privileges.

## 2.6 Security Threats

Many security threats should be accounted for when developing a complex system that will be used in production even if the system is on a private network. Such threats are already mitigated in the cloud technologies available today.

### 2.6.1 Session Hijacking

Session hijacking is when a logged-in user's session cookie gets infiltrated, and then used in the attacker's browser to impersonate the user that originally had the token. This may lead to catastrophic consequences if not dealt with caution [8].

Sessions can be compromised in several different ways; one would be intercepting network packets which will be inherently solved by enforcing a VPN to access the network, or by malware that will search for active session tokens [8].

To ensure a session token cannot be compromised or taken advantage of once it has been compromised there has to be stricter rules on how long the token lives and the IPs and client info the session token is connected to [8].

### 2.6.2 Remote Code Execution

Remote code execution exploits are one of the most dangerous attacks a system can suffer. (RCE for short) is usually done by finding a way to upload a malicious script to the system and making an operative system user execute that script [9].

The best way to avoid RCE exploit attacks is to keep all software updated, but sometimes that would be hard for infrastructures to do without running into dependency issues. To avoid this, all ways to upload files should be checked and the files uploaded would be run through a security check before being let through to a stage where they could be executable [9].

## 2.7 Related Works

### *2.7.1 Securing Sensitive Data in the Cloud, using Zero Trust Principles*

*Yacob* discusses in his thesis [12] the implementation of a secure system for handling sensitive data in a Zero Trust environment. He criticizes the traditional security systems' structure and addresses the need for minimizing trust and treating all network traffic as a potential security threat, emphasizing the importance of strict access control and authentication. The solution, chosen by Yacob to implement the security system, includes the use of GPG (GNU Privacy Guard) keys for signing and verifying commits in a GitLab repository, along with an identity provider such as Keycloak for user authentication. These key components, according to Yacob [12], are recommended to use for future work when there is a need to protect and control access to data, particularly focusing on applying the Zero Trust approach to the build/execute side of the process. This is done by customizing separate instances of Keycloak and improving the authentication and authorization process.

Alternative methods and providers for implementing Zero Trust principles were examined in the research [12], such as Cloudflare Access and VPN with MFA (Multi-Factor Authentication). Moreover, the self-hosting approach was compared to utilizing a third-party provider of Zero Trust environment from an economic aspect, considering different types of costs and maintenance expenses. Yacob, in his thesis report, takes other sustainable aspects into

concern, highlighting the benefits of Zero Trust security in protecting sensitive information, preventing data breaches, and following conventional ethical regulations [12].

### 2.7.2 Designing and implementing a private cloud for student and faculty

*Le Fevre and Karlsson* [4] did research focusing on the design and implementation of a private cloud hosting system. The authors [4] discuss various aspects of cloud platforms, to understand security in a cloud environment, create a practical and secure infrastructure-as-a-service (IaaS) platform, and find best practices for implementing security measures and approaches. Security is addressed in the research [4] as a critical aspect when developing or working on these kinds of cloud systems.

By choosing CloudStack as a cloud platform to use in their proposed system, the authors [4] highlight the benefits of using the built-in security features provided by CloudStack, such as virtual firewalls, VPNs, and VLAN isolation. These security measures, according to the research [4], contributed to creating a secure implementation of the CloudStack platform. Furthermore, secure communication is recommended to be used to ensure that data is secure and not tampered with, and the use of TLS to implement this type of secure communication [4].

# 3 Method and Results

First, in this chapter, all methods used to complete the project and to fulfil the goals defined in 1.2 are presented in a simplified manner, to make it possible for future developers to achieve the same result by following the steps mentioned below. Furthermore, this chapter introduces the literature study done behind this thesis, including previous work in the same domain.

## 3.1 Method

The choice of the solutions presented in this chapter was based on a thorough review of existing literature as well as related works in the field, that have been reviewed throughout the implementation process.

With all the research that has been done, it was decided to implement a customised control model that controls the access to the data that the applications built by the GitLab users have access to. The research done on Access Control was vital in designing the model for the data access control in the database part of the infrastructure. Some of the other technologies that already were decided by the faculty are GitLab and Neo4j, and section 3.1.1 will introduce these requirements in detail.

The research is done on Identity and Access Management (IAM) tools like Keycloak provides the architecture with a source of identification for the users that are logged in to the system. According to *Yacob* [12], Keycloak in similar systems as the one this thesis aims to develop is used to provide users with an interface to manage their data access and deployments. Furthermore, Keycloak also provides configurations that can be customised to increase the strictness of the system on the authorization. Applying Identity and Access Management (IAM) in the proposed system provides an effective approach to managing the system's users and their roles. Such an approach makes similar systems centralized, solid, and secure [7].

The research done on Identity Access Management providers such as Keycloak and Okta, including the past work done by *Yacob* [12] has been vital to the system developed. For this thesis project Keycloak is used for the popularity, extensive support, and security customization it provides. Keycloak provided an essential part in the system where the Keycloak instance could be used to log into GitLab and provide the cloud service with the identities of the user to show the right interface and allow users to access the system with their privileges. Other IAM tools do not have the same integration with GitLab and thus have a significantly harder way to connect them.

### 3.1.1 Requirements From the Faculty

There was a requirement from the faculty to have a secure database service that is connected to the GitLab Instance. The research done on GPG commit signing by Yacob [12] are vital for the access control methods used in this work. It was decided to apply the model to specific user access control to enable the system to adapt to specific access control needs for each GitLab user depending on their privileges on the repository. The commit verification GitLab offers, allows the system to take the user information from GitLab only when the code changes are authorised, then the system provides credentials from the Neo4J database service to the user to access the data related to the repository these changes were made to.

These security requirements required the system to be built upon a cloud where the users can manage their deployments and the data access the users and their repository contributors have to the application data.

To enforce strict control over data access, Neo4j was chosen, which is a graph database management system (DBMS). Neo4j provides a flexible and powerful platform, that helps manage access control in the GitLab-based cloud environment. It makes it possible for system administrators to define access rules, user roles, and permissions. Neo4j was one of three options to integrate with the system, whereas the other two alternatives were relational database (MySQL) and document database (in the form of files). Since the faculty showed the most interest in the graph database (Neo4j), it was chosen.

## 3.2 System Architecture

Software development-wise, the thesis aims to implement a proof of concept for securing data using GitLab and Neo4j. The implementation followed an agile software development process, with iterative cycles of development, testing, and deployment.
For the implementation, JavaScript programming language was used as well as relevant libraries, such as GitLab API and Neo4j database. The implementation includes the following key components:

1. Fetching code from GitLab: The implementation involves fetching code from a GitLab repository and verifying the user's identity and access control.
2. Managing data access: The implementation will involve managing data access and ensuring that only authorized users can access the data. This will be done by implementing access control mechanisms such as role-based access control.

### 3.2.1 Architecture Overview

Figure 3.1 shows the architecture of the proposed system that was implemented with the security improvements. It is important to provide a secure database for GitLab users to deploy their applications to.

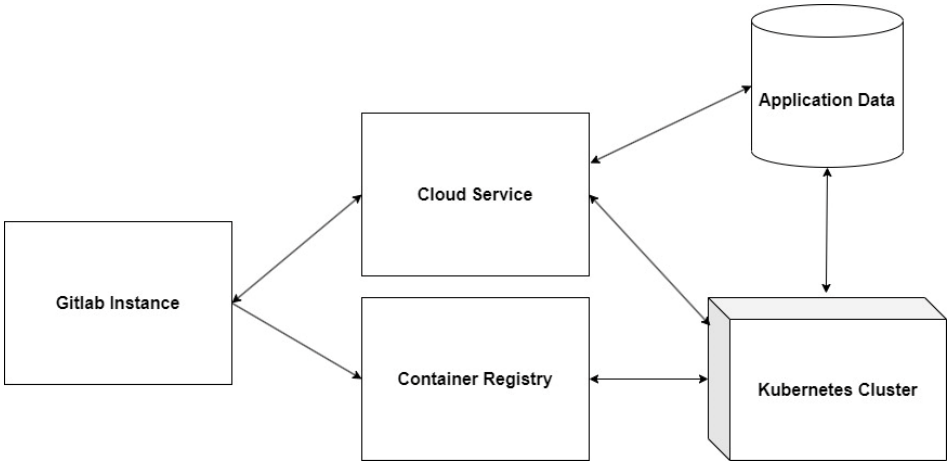


*Figure 3.1 Abstract System Architecture*

#### *3.2.1.1 Fetching Commits' Signatures*

The cloud service fetches new commits from the selected Gitlab repository and verifies that the commit has been signed, if so, it will continue with the rest of the deployment process.

#### *3.2.1.2 Verified Commits*

Verified commits then have their images pushed to the container registry and then the cloud proceeds to connect the signed signatures author to a database user and provide the credentials using temporary environment variables.

## 3.3 GitLab

GitLab is an open-source tool for managing software development projects. It provides features such as version control, code review, and continuous integration and delivery [4]. In this section, the security features provided on GitLab platforms are introduced to build a clear picture of how unauthorized access is handled. In addition, the security tools used in each stage of the feature development workflow in GitLab are described.

GitLab has been one of the revolutionary platforms for CI/CD solutions, Continuous integration, continuous delivery, and continuous deployment, simplifying the process of merging work from different teams into a unified product. CI/CD establishes a centralized work repository as well as ensures automation of integration and continuous testing [16]. The CI/CD pipelines in GitLab provide a chain of actions making the DevOps process and software application delivery more efficient and secure. Some of the stages of the DevOps CI/CD pipeline are source code repository, building an executable iteration of a product (often containerization), testing the repository, validating the caused behaviour, and deploying an executable instance of the application.

Furthermore, applications built in GitLab are examined for potential security vulnerabilities, using different tools in different stages of the application development lifecycle [15]. GitLab offers access control mechanisms that enable organizations to control who can access their data and what actions they can perform [15].

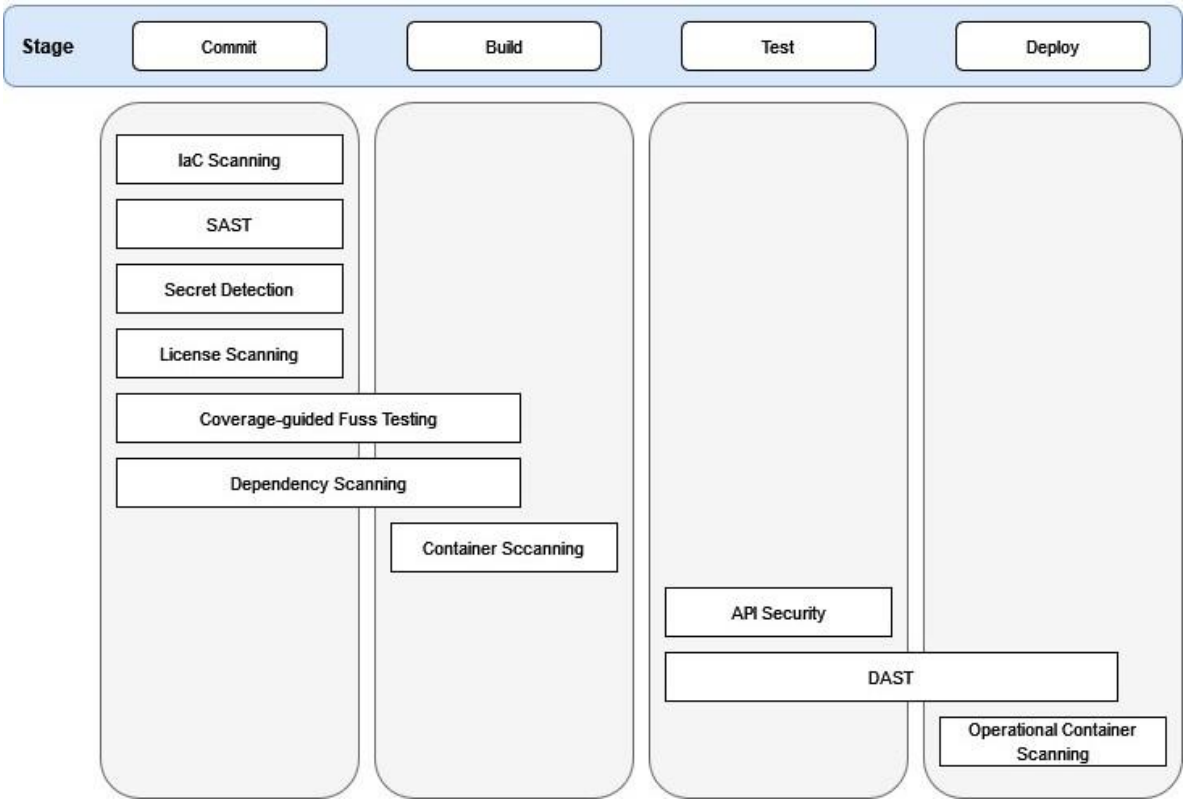


*Figure 3.2. The stage of the feature development workflow, and GitLab application security tools performed on each stage. Inspired by [15].*

Figure 2.3 shows the GitLab application security tools performed at each stage of the feature development workflow. The commit action takes place when code in a repository will be updated, trigging some scanning and testing tools before performing the commit [15]. The scanning and testing tools shown in Figure 2.3 can be customized by enabling or disabling them, making the validation process flexible and customizable.

In addition, GitLab provides a set of REST APIs making it possible for developers to integrate GitLab projects with other platforms [15]. Some of the REST APIs' usages are used to verify a user's authenticity, list projects and their status, get metadata about repositories, and much other information. More about how features in GitLab can be utilized is introduced later in Method Chapter, 3.4.

### 3.3.1 Encryption in GitLab, GPG Keys

The OpenPGP email encryption standard is used by GPG (GNU Privacy Guard) Keys to encrypt and sign messages and other data. The GPG key pairs have a public key and a corresponding private key, where the public key is used for encryption and to verify digital signatures, and the private key is kept secret and used for decrypting messages and signing data [18]. In GitLab, GPG keys play an important role in improving the security of GitLab by enabling users to verify their authenticated commits. GPG keys make it possible for developers to sign their commits on their repository with their private key, adding a digital signature that can be verified by other users. This approach ensures that commits originated from a trusted source, obstructing unauthorized modifications to the source code, and guaranteeing the integrity of the project or the repository. In other words, once a commit is signed with a GPG key, it becomes tamper evident.

GitLab allows users to add new and associate, update, and disable locally generated GPG keys. Moreover, GitLab provides public APIs to verify the authenticity of users' signed commits [18]. It is also possible to get a user's public GPG key that is provided with the user's username. Generating GPG keys and associating them in GitLab is not complicated. However, some obstacles may appear when verifying the signed commits. Table 3.1 shows the common failures appearing during the verification process and how they can be fixed.

*Table 3.1 Fix verification problems with signed commits. Inspired by [18].*

| Value | Description | Possible fixes |
|---|---|---|
| UNVERIFIED | The commit signature is not valid. | Sign the commit with a valid signature. |
| SAME_USR_DIFFERENT_EMAIL | The GPG key used to sign the commit does not contain the committer email but does contain a different valid email for the committer. | Amend the commit to use an email address that matches the GPG key, or update the GPG key to include the email address. |
| OTHER_USER | The signature and GPG key are valid, but the key belongs to a different user than the committer. | Amend the commit to use the correct email address, or amend the commit to use a GPG key associated with your users. |
| UNVERIFIED_KEY | The key associated with the GPG signature has no verified email address associated with the committer. | Add and verify the email to your GitLab profile, update the GPG key to include the email address, or amend the commit to use a different committer email address. |
| UNKNOWN_KEY | The GPG key associated with the GPG signature for this commit is unknown to GitLab. | Add the GPG key to your GitLab profile. |
| MULTIPLE_SIGNATURES | Multiple GPG or x.509 signatures have been found for the commit | Amend the commit to use only one GPG or x.509 signature. |

GitLab provides an API that enables developers to access GitLab functionality programmatically. The GitLab API enables developers to manage repositories, issues, merge requests, and other GitLab features [4].

### 3.3.2 GitLab Instance Setup

GitLab instances can be set up in different ways, but the approach that was followed in this project is pulling a GitLab image from Docker hub containing the GitLab instance. Another approach could be to install a Kubernetes cluster that supports GitLab.

- Pull and run the GitLab instance:
  - docker run --detach --hostname gitlab.example.com --publish 443:443 --publish 80:80 --publish 22:22 --name gitlab --restart always --volume "/$(pwd)/config:/etc/gitlab" --volume "/$(pwd)/logs:/var/log/gitlab" --volume "/$(pwd)/data:/var/opt/gitlab" --shm-size 256m gitlab/gitlab-ee:latest

- Change the default password (unknown) to root:
  1. *docker exec -it containerID bash*
  2. *gitlab-rake "gitlab:password:reset[root]"*
  3. *Insert your password.*

Now, the GitLab instance is up and running, the next step is to log in with the username root and the password that was chosen before.

- Getting the GitLab API Access Token:

  1. Log in with the root user account.
  2. From the avatar icon located on the very right on the top bar, select **Preferences**.
  3. Click on the **Access Tokens** in the left sidebar.
  4. Create the personal access token by inserting the Token name and selecting all scops below.
  5. Copy the generated token and paste it into the text field in Postman (Authorization -> Type: Bearer Token).
  6. Test this URL on the GET HTTP-request method: localhost/api/v4/projects/1 and you will get the repository as a response.

## 3.4 Basic Security

Several important measures must be taken while securing the environment.

### 3.4.1 Security Measures

#### *3.4.1.1 VPN Access*

For a private implementation, it is viable to set a Firewall where a certain VPNs IP address and network are allowed to access the system. This adds another layer of security on top of the system that would.

#### *3.4.1.2 Commit GPG Signatures*

According to the previous work done by *Thomas Yacob* GPG Signatures provide proof of the authenticity of commits that are pushed into the GitLab instance. These signatures can be used by other processes in the infrastructure to know if the commit is authentic and thus use that information from the API to allow for a higher privileged action later in the chain.

#### *3.4.1.3 Two-Factor Authentication (TFA)*

Two-factor authentication is a widely used security measure that requires clients to provide additional form of identification. This is a measure that assures that if one of the factors is compromised that there would still be another requirement to get full access to the account's privileges.

#### *3.4.1.4 Securing The Pipeline*

Gitlab allows for many security methods to add to their DevOps pipeline such that there would be a manual pipeline that administrators must approve for them to continue and other security measures in the Auto DevOps feature that GitLab provides.

### 3.4.2 Implementation

#### *3.4.2.1 Enforcing Two-Factor Authentication on GitLab*

To enforce TFA to all GitLab users just follow these steps through the admin user interface on GitLab:

1. On the top bar, select **Main menu** > **Admin**.
2. On the left sidebar, select **Settings** > **General** (/admin/application settings/general).
3. Expand the Sign-in restrictions section.
4. Enable TFA and set the grace period to zero.

#### *3.4.2.2 GPG Signing Setup*

A previous work [12], done by *Thomas Yacob*, was used in the project to set up the GPG (GNU Privacy Guard) Signing Keys. The GPG keys are used primarily to verify committed changes on a GitLab repository and use the public key to verify commits signed with the corresponding private key. That means that every single user in the system, with commit permissions, requires a won unique GPG (public and private) key pair. Moreover, it is important to know that the GPG keys are only used for the verification purpose, and not to handle logging into GitLab.

Once a user is authenticated and verified, the process of GPG signing is initiated by creating a private and public key pair [12]. To submit a commit on a GitLab repository, the user must sign that commit using their private GPG key. The signature is created by a cryptographic hash function that generates a unique value. After that, the server verifies the signature's authenticity using the user's public GPG key. In case of the signature is valid, it proves that the commit is legitimate and has not been altered. Consequently, GitLab will accept the commit. "The private key signs the commit, the public key verifies the signature" (*Yacob* [12]) resulting in that GPG key signing guarantees that once a specific authenticated user, with the right permissions, make a commit in GitLab the data will never be tampered on.

Integrating GPG key signing with GitLab is done by following the steps below:

1. Install GPG locally.
2. Generate a GPG key pair.
3. Copy the public key.
4. From the avatar icon on the top bar, select **Edit profile**.
5. Click on the GPG Keys on the left bar.
6. Paste the public key in the text field with the Key title and click on Add key.

## 3.5 Architecture Implementation

### 3.5.1 Reserving Environment Variable Names

To allow users to connect to a database provided, there must be environment variable names that are reserved to allow the developer to easily use the database. This is used later in the implementation to inject database credentials into the code.

Reserved Environment Variables:

GLC_NEO4J_URL
- URL of the server where it is hosted.

GLC_NEO4J_PORT
- PORT is used for the server.

GLC_NEO4J_USER
- Database user for the application to access the data.

GLC_NEO4J_PASSWORD
- Database user password.

GLC_ prefix, which stands for GitLab Cloud, is used to be unique and avoid clashing with other reserved environment variables that users may choose to use.

### 3.5.2 Cloud Setup

The cloud backend server is set up with the JavaScript backend framework NodeJS with the addition of Express which is a web framework to simplify sending and receiving data through the web.

The most straightforward way of setting up an express server is to create the repository directly on Gitlab like such:

1. Create a Gitlab account on gitlab.com.
2. Click on New Project.
3. Click on Create from template.
4. Fill in all the fields.
5. Clone the repository to your device.

In addition to this, some important information needs to be established before proceeding that is needed throughout the cloud backend server:

1. The Gitlab instance's root user's access token as described at the end of section 3.4.1.
2. Registry URL.
3. Gitlab instance IP and port.
4. Neo4J IP, port, root credentials.

These should be stored securely in the project's environment variables.

The Neo4J JavaScript driver is used in the cloud to allow queries to be sent to the neo4j server and access user information to make the best decision later.

## 3.6 Securing Application Data

After all the architectural setup is done with the right security precautions it is time to implement the system that will control the access to the Neo4J database that the Gitlab users will have their applications built upon.

To summarise the Access Control system that was made here is an ordered list and then will be followed by detail for all the steps:

1. Validate commit is signed by GPG using GitLab API
2. Check if the commit author has a database user in Neo4J. (If not then add the database user)
3. Check if the User has privileges to access the data in the database connected to the repository. (If not then add the privileges)
4. Check if there is already a temporary active user, if so, then delete it.
5. Create a temporary user with privileges only to the database that the repository is connected to.
6. Send database user credentials with the deployment.
7. Remove user credentials from the docker-compose file.

### 3.6.1 Getting the commit

#### *3.6.1.1 Request*

```
var axios = require('axios');

axios.get(`https://localhost:80/api/v4/projects/${id}/repository/commits`, {
 headers: {
  'PRIVATE-TOKEN': `${token}`
 }
}).// Request Handling
```

Replace *${token}* with the Access Token and *${id}* with the project ID (Retrieved by fetching the projects the user has)

#### *3.6.1.2 Returned Data Example*

```
[{
  "id": "ed899a2f4b50b4370feeea94676502b42383c746",
  "short_id": "ed899a2f4b5",
  "title": "Replace sanitize with escape once",
  "author_name": "Example User",
  "author_email": "user@example.com",
  "authored_date": "2021-09-20T11:50:22.001+00:00",
  "committer_name": "Administrator",
  "committer_email": "admin@example.com",
  "committed_date": "2021-09-20T11:50:22.001+00:00",
  "created_at": "2021-09-20T11:50:22.001+00:00",
  "message": "Replace sanitize with escape once",
  "parent_ids": [
   "6104942438c14ec7bd21c6cd5bd995272b3faff6"
  ],
  "web_url": "https://gitlab.example.com/janedoe/gitlab-foss/-
/commit/ed899a2f4b50b4370feeea94676502b42383c746"
 }]
```

### 3.6.2 Validating Signed Commit

To take advantage of the GPG signing of the commits the Gitlab API is used to assure that the latest commit in the repository is signed and verified.

#### *3.6.2.1 Request Getting GPG signature of the latest commit:*

```
var axios = require('axios');

axios.get(`https://localhost:80/api/v4/projects/${id}/repository/commits/${branch
}/signature`, {
 headers: {
  'PRIVATE-TOKEN': `${token}`
 }
})..// Request Handling
```

Replace *${token}* with the Access Token, *${id}* with the project ID (Retrieved by fetching the projects the user has) and ${branch} with the branch name or tag.

#### *3.6.2.2 Example Successful Response*

```
{
 "signature_type": "PGP",
 "verification_status": "verified",
 "gpg_key_id": 1,
 "gpg_key_primary_keyid": "8254AAB3FBD54AC9",
 "gpg_key_user_name": "John Doe",
 "gpg_key_user_email": "johndoe@example.com",
 "gpg_key_subkey_id": null,
 "commit_source": "gitaly"
}
```

#### *3.6.2.3 Validation*

```
if (data.signature_type === “PGP” && data.verification_status === “verified”) {
            //Valid and continue
} else {
            //Invalid
}
```

### 3.6.3 Database User

Since only signed commits are allowed to be deployed the property “gpg_key_user_email” from 3.6.2.2 is used to create a connection between the Gitlab user to the Neo4J database. The user email is checked against the Neo4J database with the following query to the driver:

```
SHOW USER WHERE user=”<user_email>”
```

and if it doesn’t exist the following query should be done to add that user through the javascript driver:

```
CREATE USER <user_email>
SET PASSWORD <generated password> CHANGE REQUIRED
SET STATUS ACTIVE
SET HOME DATABASE <repository_identifier>
```

The generated password should be sent securely to the user who is forced to change it after logging in for the first time.

### 3.6.4 Graph Privileges

The privileges of the user against the connected repository application data are checked with the following query:

```
SHOW USER PRIVILEGES WHERE user="<user_email>" AND graph="<repository_identifier>"
```

If there are no privileges for that user on the repository graph, then they are added to them with the GRANT keyword.

### 3.6.5 Temporary Database User

Temporary database users in this implementation are essential to creating an isolated environment for applications deployed to the cluster to have their database service.

```
CREATE USER “temp_<user_email>_<repository_identifier>”
SET PASSWORD <generated password> CHANGE NOT REQUIRED
SET STATUS ACTIVE
SET HOME DATABASE <repository_identifier>
```

The generated password is then used to be passed in the environment variables of the deployment. The same permissions are granted that the original user account has to that account.

### 3.6.6 Deployment

For deployment, the temporary user is inserted into the docker-compose file under the YML environment key, and the repository-built image is inserted into the compose file in the following format:

```
image: <url_to_image_on_private_registry>
  environment:
        - GLC_NEO4J_URL=<neo4j_url>
        - GLC_NEO4J_PORT=<_port>
        - GLC_NEO4J_USER=<temp_user>
        - GLC_NEO4J_PASSWORD=<temp_user_password>
```

Directly after deployment as a security precaution, the YAML file is edited to remove sensitive information.

## 3.7 Result

The result of this implementation is a deeply controlled system where the administrator manages the security and data access methods for the users. Deep Security is implemented in all sections of the architecture to ensure maximal control.

GPG signatures ensure that the commits originated from the commit author so that the system can use that proven trust to then provide the deployment with the right data. Communication between all the systems in the architecture is secured by having TLS enabled on every system, this provides a safe tunnel for crucial data not to be exploited.

Temporary users are created with identical privileges to the signed commit author to ensure the right access has been provided to the application. The temporary user does not include access privileges to the other graph databases the author has access to, which is the main reason the author's database user is not used directly. That ensures that any code in the repository that tries to access other databases is denied. After that, the GitLab is ready to create an image of the repository and log it into a container registry. Before going on in the process and building a container, the temporary database user credentials are stored in a YAML file as environment variables, which are then attached to the image. If the container is executing, the program, through the temporary user, can access the data. Once the container is revoked, the new container is built, the temporary database user is removed from the database and a new temporary database user is created.

In this implementation credentials used in the developed application's drivers will receive privilege errors when trying to access restricted data, and also privilege errors when trying to interact with the data in some way for example when trying to do write actions on data that is only accessible with read permissions.

The result has technically been controlling applications' access to data, which means that the system can handle users and assign permissions to them. This was done by integrating different platforms with the cloud service application developed during the project thesis, where users are verified in every single step of the process, which makes the system compatible with the Zero Trust principles.

# 4 Analysis and Discussion

In this chapter, the analysis of the implementation results involves evaluating the effectiveness of the implemented security measures and access control mechanisms. The analysis will also involve measuring the security of the system, evaluating the performance of the system, and assessing the usability of the system.

The implementation of a deeply controlled system discussed in this thesis provides users with comprehensive management of security and data access methods. The goal was to maintain maximum control and implement extensive security measures throughout the architecture. The thesis emphasizes several key aspects that contribute to the achievement of this goal.

To begin, GPG signatures are used to validate the origin of commits, ensuring that they are genuine and come from the designated commit author. The system then uses this trust to provide the necessary data for the deployment. The system can maintain a high level of integrity and trustworthiness in the codebase by implementing this verification mechanism.

Furthermore, Transport Layer Security (TLS) is used to secure communication between all systems in the architecture. Enabling TLS on all systems creates secure tunnels for critical data transmission, preventing potential exploitation and unauthorized access. This safeguarding mechanism ensures that data remains confidential and intact while in transit.

To effectively manage user access privileges, temporary users with the same privileges as the signed commit author are created. This method ensures that the application is granted the necessary access rights. To reduce the risk of unauthorized access, the temporary user is intentionally barred from accessing other graph databases. By enforcing this restriction, any attempts by the repository's code to access other databases are denied, improving the overall security posture of the system.

The implementation process involves creating an image of the repository and logging it in a container registry using GitLab. Before proceeding with container building, the temporary database user credentials are stored in a YAML file as environment variables, which are then attached to the image. This enables the program, through the temporary user, to access the required data while the container is executing. Once the container is revoked, the temporary user is removed from the database, and a new temporary user is created for the next container.

The outcome of this implementation is a system that has gained control over applications' access to data. Through the integration of various platforms with the developed cloud service application, the system can effectively manage users and assign permissions to them. The thesis highlights the verification of users at each step of the process, aligning the system with the principles of Zero Trust. By adhering to these principles, the system establishes a robust and secure foundation, enhancing the overall security and control of the application.

Alternative solutions would have helped achieve the same goals as the current solution has already. GPG commit signatures in GitLab could have been alternated with either SSH commit signing or X.509 certificate. Using X.509 certificates to sign commits requires more resources for the project to implement and adds more complexity to the end user trying to

take advantage of the system. In a more ideal service, all these methods would be available to the end user to allow for freedom of choice.

The model the thesis explored creating temporary users to access the application data. A different model that would achieve the same goals but with less security strictness would be to provide the deployment of the same database user credentials that the owner of the repository has. This creates a less strict environment for the application and thus allows the applications deployed on the infrastructure to access data the owner has access to, even though the application built has no intended connection to them. This kind of model does not allow for future development to provide dynamic credentials to different users.

Including the credentials in the docker-compose file of an application allows the credentials to be injected into the deployment, the project goes further by removing the credentials from the YAML file to avoid any intruders from accessing the data. Removing this step would also suffice but then there must be assurance that there is not any possibility for intruders to have access to the YAML file.

## 4.1 Sustainability Impacts

The Royal Institute of Technology KTH has regulations of sustainability goals within all programmes [17]. Therefore, in this thesis, technologies and implementation approaches have been considered from social, economic, and environmental perspectives. The impacts that the implementation has can be significant, and they arise from the increased protection of data, improved access control mechanisms, and the adoption of encryption techniques.

### 4.1.1 Social Impact

Implementing secure data practices enhances data privacy, protecting sensitive information and personal data. This promotes trust between organizations and their users, which could be patients, clients, or customers, as individuals feel more confident that their data is secure from unauthorized access or breaches. Moreover, by implementing secure data practices, organizations can effectively follow data protection regulations and standards such as GDPR and HIPPA. This ensures that personal or sensitive data is handled suitably and legally, which promotes ethical and responsible data management. Since the resulting project can be used by government agencies or organizations handling information about a huge amount of people, the security requirements do have a huge impact on individuals' lives.

### 4.1.2 Economic Impact

Implementing secure data practices reduces the risk of data breaches and their associated costs. Organizations can avoid financial losses due to potential litigation, reputation damage, and other costs of remediation in the case of a data breach. In addition, showing an engagement in data security can provide organizations with a competitive edge. Both clients and customers more and more prioritize a high level of data protection and organizations that can guarantee robust security measures are more attractive to clients and customers.

### 4.1.3 Environmental Impact

The efficient implementation of secure data practices can contribute to energy savings. By optimizing data storage and access, organizations can reduce energy consumption related to data processing and storage systems, leading to a smaller environmental footprint. Improved security measures promote the transition to a paperless approach, reducing paper waste and the associated environmental impact. Digital storage and secure data practices enable

organizations to minimize reliance on physical documents, contributing to sustainability efforts. This will align with the national and European sustainability goals.

### 4.1.4 Ethical aspect

Data engineers and system developers must always consider the ethical aspect of handling data, especially in larger systems where large amounts of data about many people are stored and processed. Ethics play an important role in secure data management, where privacy rights are respected, and data confidentiality is ensured. By implementing reliable security measures and access controls, organizations prove their engagement to protect individuals' personal information and upholding ethical principles. In domains such as healthcare, finance, and other sectors that deal with sensitive data, it is extra important to take the ethical aspect into concern. The ethical aspect of secure data practices includes having transparency about data handling practices, obtaining consent from users, and following standardized ethical guidelines and regulations.

# 5 Conclusion

This bachelor's thesis has explored and implemented secure data practices in a GitLab-based cloud environment, addressing the problem of data security and access control. By investigating access control, data encryption, and data immutability, this paper has made significant contributions to knowledge development within the area of secure data management. The thesis has also provided practical insights into the implementation of these techniques and their integration with the GitLab platform. The goal of this report was to guarantee secure data and protect it from unauthorized access and changes. This has been achieved by exploring techniques for data encryption and access control in GitLab and studying how to implement them using a Neo4j database. Furthermore, this paper has identified the importance of cryptographic task-role-based access control, as well as the use of built-in encryption techniques in Neo4j.

The implementation of the report's proposed solution includes fetching code from GitLab, verifying identity and access control, managing data access, and displaying results. The implementation is recommended to be implemented in a similar system, this can be done by following the steps described in the *Method and Results* chapter. Furthermore, potential continuous work can be done in this area to address new problem statements, such as the integration of other databases with GitLab than Neo4j, the use of blockchain technology for enhanced data security, the implementation of more advanced access control mechanisms, a closer investigation of how the encryption can be more quantum computer resistant, and the automation process of fetching code from GitLab, verify data access and permissions, execute the code on that data, and view the result of the executing. These areas provide opportunities for future research and development in secure data management. Moreover, the project's cloud service application could be done more user friendly.

## 5.1 Goal Evaluation

Following is a reminder of the goals that were presented before starting the thesis project:

1. Provide an analysis of different methods of protecting data, such as access control, encryption, and data immutability.
2. Identify best practices for securing data in a Cloud environment.
3. Identify controlling data access using Neo4j.
4. Develop a software solution that:
    a. Demonstrates the implementation of these techniques in a GitLab environment.
    b. Enables secure access to the system.
    c. Ensures that data is not modified by unauthorized users.

There were several goals to achieve in this thesis such as investigating the different methods to manage data and the user's access to the data. Many users of Git repository services also use these repositories for their security and Development Operations features, so it was important to provide security for the data that is handled in such a cloud environment. The work behind this thesis has concluded that controlling the database users and having a connection between them and their counterparts on the GitLab instance was vital to have a secure pipeline and provide a purpose for the GPG integration into the environment.

The practices for securing a Gitlab environment were partially identified. Issues occurred where there was an extensive number of configurations that improved the security of the Gitlab instance, but the thesis project touched upon the essentials of that which will most certainly provide confidence in the system.

To control the data access in Neo4J, the research done upon Access Control was used to create a model to enforce strict access to the database with the access users used in the deployments.

## 5.2 Future Work

Most of the thesis' goals were accomplished, but for any work done on this model, it is suggested to create a more unified system for a better experience given to the users of the system. It is also encouraged to create a system where repository owners can define the access for several developers on the same repository and create a way where the application will support test environments with user credentials of lower privileged roles using the cloud.

# Bibliography

[1] Role-Based Access Control [Internet]. American National Standard ANSI INCITS 359-2004. [cited 24/03/2023]. Available from https://profsandhu.com/journals/tissec/ANSI+INCITS+359-2004.pdf

[2] D'Silva D. Ambawade D. Building A Zero Trust Architecture Using Kubernetes [Internet]. IEEE Xplore; [cited 02/05/2023]. Available from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9418203

[3] Bernstein D. Introduction to post-quantum cryptography [Internet]. Department of Computer Science, University of Illinois at Chicago. Springer Link; [cited 03/05/2023]. Available from http://www.pqcrypto.org/www.springer.com/cda/content/document/cda_downloaddocument/9783540887010-c1.pdf

[4] Le Fevre P. Karlsson E. Designing and implementing a private cloud for student and faculty software projects [Internet]. KTH Royal Institute of Technology; [cited 24/03/2023]. Available from https://kth.diva-portal.org/smash/get/diva2:1666025/FULLTEXT01.pdf

[5] Bridgwater A. Okta Insists Identity 'Goes Beyond' Passwords [Internet]. Forbes [cited 24/03/2023]. Available from https://www.forbes.com/sites/adrianbridgwater/2018/05/23/okta-insists-identity-goes-beyond-passwords/?sh=5516d8ce26b7

[6] Authorization Services Guide [Internet]. Keycloak Docs [cited 17/04/2023]. Available from https://www.keycloak.org/docs/latest/authorization_services/

[7] What is Identity and Access Management (IAM)? [Internet]. OneLogin [cited 17/04/2023]. Available from https://www.onelogin.com/learn/iam

[9][8] KEYFACTOR. What is Session Hijacking and How Does it Work? [Internet]. KEYFACTOR; [cited 29/4/2023]. Available from https://www.keyfactor.com/blog/what-is-session-hijacking-and-how-does-it-work/

[10] Remote code execution (RCE) [Internet]. Invicti; [cited 02/05/2023]. Available from https://www.invicti.com/learn/remote-code-execution-rce/[9] invicti. Remote code execution (RCE). https://www.invicti.com/learn/remote-code-execution-rce/

[10] What is a Graph Database? [Internet]. neo4j (Developer); [cited 17/04/2023]. Available from https://neo4j.com/developer/graph-database/

[11] Sandhu R. S. and Samarati P. Access Control: Principles and Practice [Internet]. IEEE Xplore; [cited 23/04/2023]. Available from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=312842

[12] Yacob. T. Securing Sensitive Data in the Cloud: A New Era of Security Through Zero Trust Principles [Internet]. KTH Royal Institute of Technology; [cited 22/03/2023]. Available from https://www.diva-portal.org/smash/record.jsf?aq2=%5B%5B%5D%5D&c=1&af=%5B%5D&searchType=SIMPLE&sortOrder2=title_sort_asc&query=Cloud+security+gitlab&language=sv&pid=diva2%3A1739157&aq=%5B%5B%5D%5D&sf=all&aqe=%5B%5D&sortOrder=author_sort_asc&onlyFullText=false&noOfRows=50&dswid=-5747

[13] Managing privileges [Internet]. Neo4j Docs; [cited 17/04/2023]. Available from https://neo4j.com/docs/cypher-manual/current/administration/access-control/manage-privileges/

[14] Access control [Internet]. Neo4j Docs; [cited 17/04/2023]. Available from https://neo4j.com/docs/cypher-manual/current/administration/access-control/

[15] Application security [Internet]. GitLab Docs; [cited 23/04/2023]. Available from https://docs.gitlab.com/ee/user/application_security/

[16] What is a CI/CD pipeline? GitLab [Internet]. GitLab Docs; [cited 24/04/2023]. Available from https://about.gitlab.com/topics/ci-cd/cicd-pipeline/

[17] KTH:s hållbarhetsmål inom utbildning. KTH The Royal Institute of Technology; [cited 16/05/2023]. Available from https://www.kth.se/om/miljo-hallbar-utveckling/utbildning-miljo-hallbar-utveckling/verktygslada/larande-for-hallbar/outcomes/kth-s-hallbarhetsmal-inom-utbildning-1.612098

[18] Sign commits with GPG [Internet]. GitLab Docs; [cited 04/05/2023]. Available from https://docs.gitlab.com/ee/user/project/repository/gpg_signed_commits/

[19] Qian J. Hinrichs S. Nahrstedt K. ACLA: A Framework for Access Control List (ACL), Analysis and Optimization [Internet]. SpringerLink [cited 26/03/2023]. Available from file:///C:/Users/alkha/Downloads/978-0-387-35413-2_18.pdf

[20] Divyabharathi D.N. Cholli N. G. A Review on Identity and Access Management Server (KeyCloak) [Internet]. MEDEWELL PUBLICATIONS [cited 13/04/2023]. Available from A Review on Identity and Access Management Server (KeyCloak) (docsdrive.com)

TRITA – CBH-GRU-2023:088