Secrets Management in a Multi-Cloud Kubernetes Environment

Master of Science in Technology
Thesis
University of Turku
Department of Computing
Software Engineering
2021
Markus Blomqvist

Supervisors:
Lauri Koivunen
Tuomas Mäkilä

UNIVERSITY OF TURKU
Department of Computing

Markus Blomqvist: Secrets Management in a Multi-Cloud Kubernetes Environment

Master of Science in Technology Thesis, 51 p.
Software Engineering
May 2021

Secrets are anything that can be used to authorize or authenticate to e.g. cloud services, databases, APIs etc. They are something that an organization must protect from being ended up in the wrong hands. As the size of the organization grows, the importance of protecting the business-critical secrets becomes more and more relevant and that is why the organizations also must pay an increasing amount of attention to their secrets management as the organization grows.
The secrets being compromised is a threat that can be prevented with a variety of methods. Configuring all of these prevention methods manually is non-trivial. Secrets management platforms implement these methods by both improving security and automating tasks. The use cases of a secrets management platform might have great variety between organizations based on their requirements. Some organizations might want to fully automate the entire lifecycle of their secrets management and use extensive features of a secrets management platform, whereas many others would only need to store their existing credentials to a centralized and secure location.
A case study is performed on the secrets management of a company called Anders Innovations. Their adoption of a secrets management platform required some further investigation as their end goal was to get a full cloud-agnostic service that can automate their secrets management. The research questions are made with a mindset that they would act as a reference for other organization in plans of adopting a secrets management platform. The first research question is about generalizing the cloud-agnosticism of secrets management. The second research question aims to clarify the automation of secrets management in automated build environments, which are being used in an increasing amount as organizations adopt new DevOps practices. The third research question is about combining the access rights management with an existing system of an organization.

Keywords: secrets management, Hashicorp Vault, DevSecOps

Contents

List of Figures

List of acronyms

AD Active Directory

API Application Programming Interface

CA Certificate Authority

CI/CD Continuous Delivery/Continuous Integration

CLI Command Line Interface

CPU Central Processing Unit

EaaS Encryption as a Service

GCP Google Cloud Platform

HA High Availability

HCL Hashicorp Configuration Language

IaC Infrastucture as Code

IAM Identity & Access Management

JSON JavaScript Object Notation

JWT JSON Web token

LDAP Lightweight Directory Access Protocol

OS Operating System

PAM Privileged Access Management

PKI Public Key Infrastructure

RBAC Role-based Access Control

SSL Secure Sockets Layer

TLS Transport Layer Security

URI Uniform Resource Identifier

VPC Virtual Private Cloud

1 Introduction

As the adoption of distributed cloud infrastructure by companies has become more and more popular and the complexity of the cloud computing technologies is increasing constantly, up-to-date solutions for handing the security aspects of these environments must keep up. The containerized microservice architecture requires the security measures to adapt to the distributed systems and allow networking with these separate entities securely. Secrets are key-value pairs that allow access to certain entities in the company's architecture or external APIs. To protect the business-critical functions and to enable Agile development at the same time, it is crucial to have these secrets stored safely, while maintaining a decent workflow for managing access for users to these secrets.

The case study performed in this thesis is done for Anders Innovations, which is a borderline mid-sized software consultancy company. The company takes care of hosting dozens of customer projects and maintaining the cloud infrastructure to run them. The projects are mostly web applications that are run on multi-cloud environments, focusing on Google Cloud Platform and Microsoft Azure services [1] [2]. This thesis focuses on researching methods for improving the current secrets management methods of the company.

The aim is to improve both the security aspects and to make the workflow for administrators and developers easier and most importantly, automated. Taking the leaps into automating the security operations is a natural step for the company as

it is already leveraging DevOps practices reducing the lead times and to automate mundane tasks. In technical terms, integrating DevOps practices into security operations is often referred as DevSecOps, which is another key area of this thesis. [3]

Secrets are at their simplest form, key-value pairs that contain secret information to access certain resources. It goes without saying that they must be stored in a secure location. Ideally they are wanted to be accessible from various environments such as a local development environment, CI/CD(Continuous Integration/Continuous Delivery) pipeline, a staging or a production environment. Implementing a centralized management for the secrets is capable for automating many of the laborious but important tasks such as creating, rotating and revoking secrets. A centralized solution can not only improve the security level in general, but also enforce the automation of such tasks. Systems like these are referred a secrets management systems. They are part of modern DevSecOps practices and a shift from manual security opration to more automated culture. For this thesis, the tool that we will be focusing on regarding the management of the secrets, will be Hashicorp Vault. [4] [5]

Containers are isolated environments running on their own software requirements. Containers build up the microservice architecture by running different pieces of the distributed system. Kubernetes is a modern solution built and maintained by Google, that allows scalable orchestration of the containerized architecture. With Kubernetes, the individual secrets must be injected to the containers as environment variables. Doing this with an architecture built on multi-cloud, multi-tenant environments is a non-trivial task that requires some further investigation and custom software. [6]

The research questions of the thesis will be the following:

- RQ1: How secrets management is handled in multi-cloud environments?
- RQ2: How secrets can be used in automated build environments?
- RQ3: How can access rights management of a secrets management platform be combined with third party services?

In the 2 chapter, existing literature in the area of secrets management is reviewed to provide a solid background on the motives and use cases on implementation of a secrets management platform like Vault. The literature is used to understand the key areas of secrets management, such as defining secrets and the use cases and features for a secrets management platform.

In the 3 chapter, a deeper dive is taken into the case and to review the methods used to perform the case study. The background for the company, the infrastructure and the problem are also detailed in this chapter to provide a base for the case study. The interview questions will be introduced and the role of the different technologies within the infrastructure will be reviewed.

The 4 chapter is focused on conducting the user interviews as well as demonstrating the concepts via technical examples. The chapter acts as a source of information for the following 5 chapter, that contains the solutions for the case study. The final chapter 6 concludes the thesis by generalizing the observations and the results of the case study.

The 5 chapter aims to provide a general level solution for the research questions as well as to generalize the case study results in a way that they can be used regardless of the infrastructure. The aim is also to prove the benefits of an automated secrets management platform and how it fits in to the DevSecOps paractices in the modern IT world.

Chapter 6, as the last chapter, concludes the thesis by pointing out the key points in the literature review and the key results in the case study and combining them into conclusions that concretely answer to the research questions. The 6 chapter also summarizes the message of modern literature in the area of secrets management and DevSecOps principles related to that.

2 Secrets Management

This chapter is a literature review of the key concepts on secrets management and the process of automating it. Secrets management platforms are systems capable of operating with multi-cloud environments due to their APIs that enable a secure communication regardless of the physical or virtual location of the client applications or users. Extensive features such as EaaS (Encryption as a Service), automatic rotation of secrets or a variety of different secrets engines can make such platform a powerful component of governing the security of the production systems of an organization. [7]

2.1 What Are Secrets?

Secrets in a cloud environment are key-value pairs that grant a specific user or a set of users, either authorization or authentication to APIs and services. Secrets are used in source code and configuration files, that are version controlled. The version controlled code should not reveal the secrets from the source code itself and that is the reason why secrets management solutions exist. [7] Secrets include credentials such as:

- Passwords, e.g. user or database
- API keys
- SSH keys

- Certificates (TLS, SSL)

Secrets are often both generated and encrypted with various cryptographic functions. These kinds of functions are often built into complete secrets management platforms like Vault, and they can be used mostly by EaaS APIs that are described in detail later in this section.

When running applications in containerized environments, the secrets must be injected into the isolated environments as environment variables. This is done at build time, allowing the secrets to be accessible at runtime. Picking a secrets management platform that places no restrictions on injecting the secrets into containers is often a requirement, as the containerized microservice architecture has gained popularity over the recent years. To illustrate this, the trends for Google searches for the "Kubernetes" keyword is shown in figure 2.1. [8]

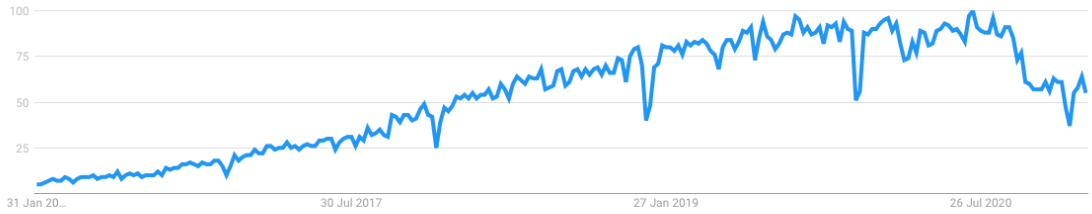

Figure 2.1: Google trends showing the interest towards "Kubernetes" keyword. Source: Google Trends (https://www.google.com/trends).

Secrets management aims to centralize the administration of multiple secrets, possibly spread over multiple projects. Secrets management aims to keep the secrets secure both at rest and while in transit. In addition, secrets management brings the secrets available in all phases of the development lifecycle, including local development, automated builds, staging and production environments. The automation combined to the improved security is a sign of DevSecOps principles being enforced

by such platforms. [7]

2.2 Use Cases of a Secrets Management Platform

The use cases for fully-blown secrets management platform are broad and allow a great amount of extensibility from just storing plain key-value pairs. Generally, the use cases include security improvements, advanced automation in handling secrets, generating and rotating secrets and integrating with third-party services for proxy access.

Security-wise, the most notable improvements, an advanced secrets management provides, are encryption, revocation, rotation and IAM (Identity and Access Management) of the secrets. The rotation and revocation are also a key part of the automation, which is being used in an increasing amount in modern software development processes. Integrating the secrets management with third-party services for proxy access other platforms and sharing secrets across multiple clients adds a use case that can boost the productivity of a development team. [7]

A typical use case of a secrets management platform is to use the platform as a single store for all the secrets needed in the application, e.g. database passwords and API keys. Build automation is another use case of a secrets management platform. Not storing credentials in plaintext to a version control system is a must, thus the credentials must be accessed from the temporary environments caused by automated builds. In a containerized architecture, the build automation phase also includes authentication to the secrets management platform, fetching the secrets and injecting them in the containers. [7]

A secrets management platform can also be used to provision the machine identities. In practice, this means allowing only certain clients to have access to certain secrets and keeping a log of the clients' requests. Due to the logging output, in case of a failure or a breach, the log outputs help to pinpoint the reason why the inci-

dent took place. Secret sharing is yet another upside of using a secrets management platform as it allows a more secure way for multiple users and environment to access the same credentials when compared to manual method of sharing the credentials. [7]

Storing secrets in an encrypted form requires management of the keys to access the keys. In a secrets management platform, external servers can be used to handle this management of the encryption keys. Sharing secrets is a method for securely handling the secrets in a team environment or different physical locations. Having certain secrets shared by a team may expose some vulnerabilities, but combining the secrets with an effective secret rotation system is a method for mitigating these threats. The same applies for sharing secrets with multiple data centers. [7]

2.3 Features of a Secrets Management Platform

The features provided by a secrets management platform can provide significant improvements to the security and automation of a company's operations. A full-featured platform like Vault is capable of providing all functions from secrets backends to EaaS (Encryption as a Service) APIs explained in this chapter. Not only can one use a platform to manually keep the secrets in a secure location but have a complete, managed solution to handle the entire lifecycle of the secrets and their assignments to the requisite parties. [7]

A core feature of such system is obviously the centralized storage for the secrets and their administration. All the features of a secrets management platform aim to improve either security or to make the operations of mundane tasks of administering secrets automated. This is why secrets management platforms suit well for DevSecOps culture. DevSecOps, like regular DevOps, is a method for reducing the lead times by automation. DevSecOps however, is focused on the security operations unlike more general DevOps [3].

2.3.1 Storage & Encryption

All modern and secure secret management platforms use encryption for their storage. The storages are often referred as "vaults". The storage can be a simple key-value storage, or it can be a more scalable relational or non-relational database. The supported formats of the secrets in these kinds of storages are flexible as any text-based secret can be encrypted and stored in them. [9] [7]

Public key infrastructure (PKI), that is explained in subsection 2.3.4, is the most common method for encryption used by secrets management platforms. PKI is an optimal way as it makes dealing with multiple clients easy. Leveraging PKI also adds another layer of security on top of encrypted network communications, adding up on the overall security. [7]

PKI is also a solution for managing security in multi-cloud environments consisting of public, private and hybrid clouds. These three types of clouds are the primary categories for all clouds. Figure 2.3.1 shows a high-level visualization of the coexistence of the different cloud types in a shared infrastructure. [10]

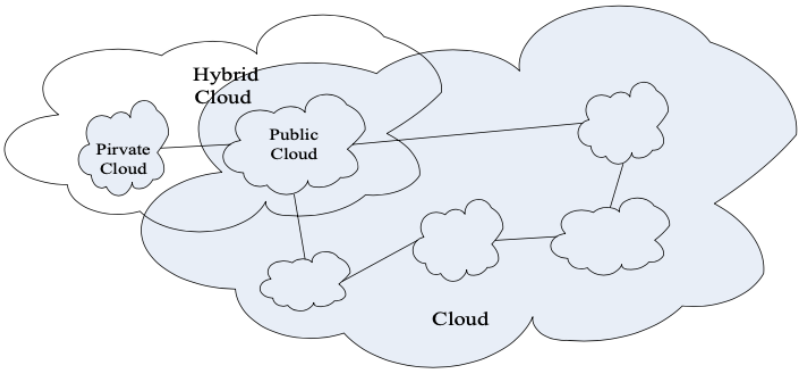

Figure 2.2: A high-level visualization of public, private and hybrid clouds in a shared infrastructure. [10]

2.3.2 Identity & Access Management

All the large cloud providers have their own identity and access management (IAM) solutions. This means, that in order for a secrets management platform to be able to integrate with the identities and access management of a third-party service, the secrets management platform must be able to read external permission from IAM, Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) services. Privileged Access Management (PAM) are high level access management services of cloud platforms, that manages the access for users with elevated permissions in an organization. Some secrets management platforms also integrate with these kinds of systems. For instance in Vault, this takes place via the different plugins for the cloud providers. [11] [12] [7] [4]

Zero-trust network is a term for internal networks, e.g. virtual private clouds that are designed in a way that even the communications that are completely isolated to that network, still require some authentication of different clients. This leads to identity-based security model, that splits up the authentication, clients and secrets to their own entities, even though they're all part of a single secrets management system. Figure 2.3.2 shows how the identity-based security model takes place in Vault infrastructure; the clients communicate with separate authentication backends that deliver the secrets via tokens. In this model, each client must perform the authentication by themselves. [7] [4]

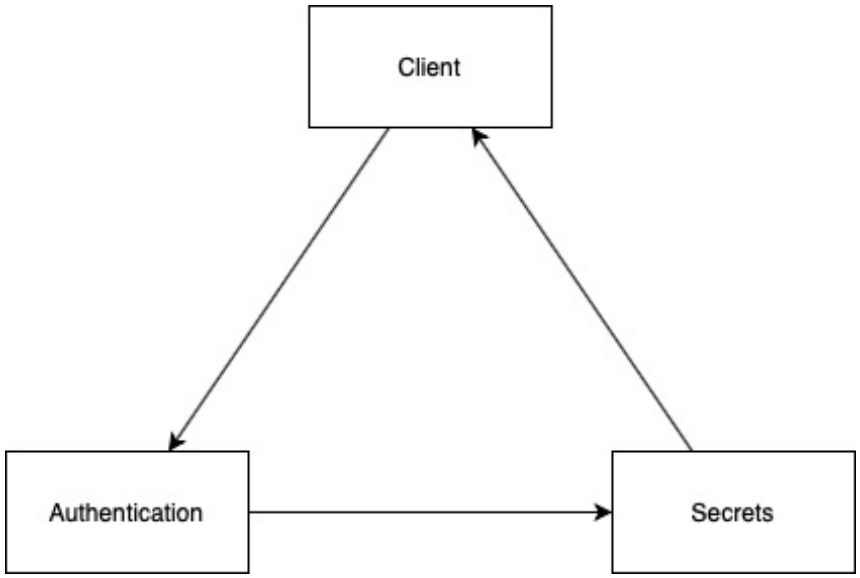

Figure 2.3: Hashicorp Vault's model for Identity-based Security.

2.3.3 APIs

All the secrets management platforms need and interface that they can be consumed through. Interactions with the APIs handle the management of the entire lifecycle of the secrets from creation to revocation. The APIs also need a strong method for authenticating the callers. For instance, authentication tokens can be used to authenticate and consume the APIs from any environment such as a local development environment or an automated build environment. [7]

The authentication parts of the APIs are often referred as API gateways since they lead the authorized client to access the actual API via proxying or otherwise relaying access. Most enterprise platforms, like Vault, use an HTTP API to implement the client-facing endpoints. The APIs are often bundled with client libraries for various programming languages. All secure APIs only allow TLS communication and a secure authentication method, such as authentication tokens. [4]

2.3.4 Shared Secrets

A very essential topic for this thesis if to find out a feasible way to share the secrets using existing roles and permissions. A shared method for authentication is required in such cases, meaning that each member of an authorized group may authenticate to a centralized secrets management server, gaining access to the shared credentials. A strong encryption over the communication must be used, especially when the secrets are sent over the network in cleartext, i.e. in a ready, human-readable form. While some older secrets management systems use secure sockets layer (SSL), which is not recommended by modern security standards, the more secure and modern systems take advantage of transport layer security (TLS). [7] [13]

Public Key Infrastructure is a common method for handling the common authentication for users of a group with shared permissions. A public key can be used to encrypt the secrets sent to the different clients, which then are able to decrypt those using their dedicated private keys. PKI seems overall a really useful solution, because it solves two problems; it does not necessarily need another layer of encryption (such as provided by TLS) and it is a secure way of making sure that no secrets are accessed by unauthorized clients, unless the client's private keys are compromised. [14] [7] [13]

Wrapping is a method for creating on-demand keys for clients and removed when the clients no longer exist. The concept of wrapping is generally used for more short-lived, ephemeral secrets. When sharing the secrets between systems located in containers that share a common host machine, the secrets can be stored in the memory of the host machine. This way the access to the secrets is faster than through network. The secrets are stored in memory in clear text. This means, limiting access to only authorized clients, in this case containers, is mandatory. Having the secrets stored in memory also makes the process of disposing them extremely easy. Injection is a method for providing container with the required

secrets upon launching them. Identity certificates are used for container to identify themselves, thus granting access for containers to access shared secrets. When using Kubernetes, these identity certificates can be issued directly to a pod to have all the containers in the given pod have the access to the secrets. [14] [7]

2.3.5 Administration

Authorization model is a concept that defines, which clients are allowed perform operations on the secrets managed by a platform. The administration of a secrets management platform is often accessed by a dashboard that contains functions not available for regular users of the platform. The administration varies to some degree between different vendors. Enterprise grade tools often have different features for administration, such as being able to delegate the needed roles for administration to other users. [7]

A common method for governing the access of different clients is role-based access control (RBAC). It aims to waive access for all other parties except authorized ones, meaning each client that needs access to certain resources governed by RBAC policies, needs to have a specific role. The clients in this context can be either machines or humans, like developers. [11] [15]

2.3.6 Provisioning Machine Identities

Provisioning the machine identities is a key piece of the security side of secrets management. Having a detailed log of the machines that are given authorization to any given secrets allows issuing e.g. unique access tokens for each machine, improving the overall security of the system and giving more detailed information in case of a breach. Doing this is the recommended way and all secure and modern secrets managements systems do provision the machine identities one way or the other. [7]

Certificate Authorities are needed to issue the certificates for machines in a PKI. The most common certificate used in PKI is X.509, used by e.g. Vault and other systems that have a PKI built-in to them. Managing the secrets in a microservice architecture ran by containerized applications, places extra requirements on provisioning the machine identities due to the dynamic nature of the environment. Also being able to scale up quickly these kinds of containerized architectures means that the provisioning of the machine identities must keep up with the scaling.

According to Hashicorps white paper Protecting Machine Identities: Blueprint for the Cloud Operating Model, "Smart policy enforcement must be automated and embedded into the tools used by application development teams. By shifting machine identity processes left into the pre-production phase and hooking directly into automated DevOps workflows, security teams can regain control over X.509 certificates in fully automated environments." This proves how the secrets management plaforms fit into the DevOps/DevSecOps culture and that automation is the way to go even for provisioning machince identities. [16]

2.3.7 Lifecycle of the secrets

The complete lifecycle of the secrets consists of creating, rotating and revoking them. Nearly all kinds of secrets can be created by a full-featured secrets management platform, including SSL and TLS certificates, passwords, tokens and encryption keys. To automate the lifecycle, the secrets can be created with expiration dates to have them automatically go out of use after a certain period of time. Some secrets management platforms also act as a certificate authority (CA) by issuing digital certificates. [7]

The key players in creating the secrets are the secrets engines. They are responsible for creating, encrypting and storing the secrets. The various clients that interact with the APIs, are requesting the secrets from the secrets engines, with the

APIs acting as middle men in between the client and the secrets engine. [4]

Revocation means the ability of a secrets management platform to invalidate or retire access for a client to a specific secret or a set of secrets. Automatic revocation of the secrets to specific applications or users is a key feature of secrets management platforms, making the process of removing unnecessary access both easier and more secure. [7]

2.3.8 Ephemeral Secrets

Ephemeral secrets are a security-improving feature and a way to create secrets. Short-lived secrets are a way to mitigate the damage caused by compromised secrets. In the container orchestration context, ephemeral secrets can be utilized efficiently, since often containers run in parallel, allowing multiple instances of certain secrets to coexist. Ephemeral secrets require the secrets management system to handle the creation and revocation automatically. The system also needs to keep track of the dynamic, short-lived instances of the secrets that have been issued for instance, a Kubernetes pod. [7]

Ephemeral secrets are often also referred as dynamic secrets. The key features of dynamic secrets are their on-demand availability and their uniqueness to each client. The dynamic secrets make the logging results more significant, because the failure points of the systems can be traced exactly by auditing the logs, which is explained in more detail in subsection 2.3.10. Figure 2.3.8 shows an example of using a dynamic secret containing a database password.

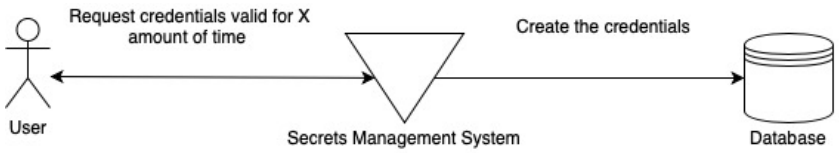

Figure 2.4: An example of a dynamic secret for a database password.

2.3.9 Encryption as a Service

Encryption as a Service is a method to create on-demand secrets for multiple clients. EaaS is also a solution for managing encryption on behalf of encryption libraries. This way, developers can omit the usage of these kinds of libraries completely and rely solely on the encryption of encryption engine provided by the secrets management system. An EaaS model requires an API to respond to the requests of clients for secrets. It is imperative that the request payload and responses use a strong encryption. Let's Encrypt is a popular example of an EaaS service. It has an API that is responsible for issuing free TLS certificates. [7] [17]

2.3.10 Logging

Being able to audit logs when a security breach occurs is a feature that most customers of security software expect to have. Keeping logs of the secrets management platforms actions help locate the point in which the breach happened and the logs keep important information such as which client requested the compromised secrets, the time of the breach or the identifier of an IAM role responsible for the incident. [7]

Configuring operational and performance alerts is another task to be done hand-in-hand when configuring logging for a secrets management system. Operational alerts are more related to security, whereas performance alerts can give real-time heads ups for e.g. throttled CPU or memory usage of the secrets management

server. Operational alerts can be used to prevent various threats. Take for instance an attacker performing a denial-of-service attack on the secrets management server. Operational alerts can be used to recognize these kinds of events. Other use cases for operation alerts would be to e.g. notify the administrators about a suspicious login. The exact operational alert cases vary case-by-case by the basics should be covered for all systems. [18]

2.3.11 Proxy Access

Proxy access of a secrets management system means that the system is able to deliver access to certain secrets by a method such as an access token or a permission authorized by a third-party IAM system, e.g. Cloud Identity and Access Management by Google Cloud Platform. Proxy access is used for authorization, not authentication. Concretely, this means that the secrets management system can give permission for a client to request the secrets while still keeping the encryption layer on to protect those secrets. [7]

2.3.12 Deployment

The deployment of a secrets management system depends on the architecture of the system. Some systems have a monolithic architecture in which a single server handles all the functions such as APIs, storage and key management. A decoupled architecture usually keeps the storage separate from a single or multiple nodes running in parallel that handle other functions of the system. This kind of architecture is more scalable as the containerized nodes can be orchestrated in a more scalable way. However, as noted in chapter 4, most organizations can get away with a single-node deployment for starters and high availability (HA) is something that might become relevant only sometime in the future. [7]

To achieve a high availability for a secrets management platform, the deployment

has to include multiple instances, ideally running in parallel in a cluster. Usually, high availability is achieved gradually, making the instances available on-demand based on automatic scaling rules of the cluster. This kind of pattern is very typical for all cloud-native web applications and thus the deployment of a secrets management platform can also be considered as one.

3 Case Description

This chapter focuses on describing the background for the case. Infrastructure-wise, the case is very typical as an increasing amount of IT organizations run their production systems in multi-cloud environments. Combining secrets management platforms to the CI/CD pipelines is a key area that is not self-explanatory for any infrastructure or organization. For this case however, the research focuses on using the ready-made solutions in case of GitLab, Vault and the other pieces of the given infrastructure. Even though the technology stack enables a relatively low amount of work to implement the automation of the injection of secrets into the containers and the other steps required in order to achieve it, the amount of configuration options for the given environments is an area that needs further investigation. [19] [4]

3.1 Company Description

Anders Innovations client pool consists of Finnish private and public sector customers. The company has over 50 employees, including 30+ developers. Within the company, there are several teams that are responsible for the development of the customer projects. In addition, there is a DevOps/support team that focuses on the support, administration and automation tasks, including the adoption of Vault. Most of the customer projects are hosted by the company in their cloud environments, while some of them are hosted by the customers.

3.2 The Problem Description

The underlying problem in the current setting of the secrets management in the company is that there is no centralized solution for the management of secrets. A role-based access integrated to GitLab permissions is currently not possible as the roles must be configured manually. Revoking access and rotating secrets are not automated currently, which becomes more and more required feature as the amount of employees goes up with time. To solve the issue of secrets sprawl, a centralized, automated system for administration is welcome.

3.3 Research Methods

The research methods of this thesis consists of reviewing existing literature, conducting interviews within the company and demonstrating technical examples. Based on the observations gained by these methods, outcome of the thesis will be a clear definition on the adoption of a secrets management platform like Vault. The literature part will be used as a background to state certain choices made along the way. The literature also works as a reference for providing details on why certain decisions are not made in some other way and justifying the decisions via counter examples. Features that will be demonstrated via technical examples include:

- Configuring Vault into a project by a developer.
- Adding and managing Vault-managed secrets in specific projects.

3.3.1 Interviews

The interviews that will be conducted in chapter 4, will be used as a reference for answering the research questions. The interviewees will be experts who work at the company and are part of the project of implementing Vault to the current

infrastructure. The interviews are aimed to gain a perspective of administrators, whereas the technical examples are more aimed towards developers. The interview questions are the following:

- What are the main problems in the current method of managing secrets in the company projects?
- What kind of custom code has to be written to make Vault work with other pieces of infrastructure?
- How users and applications will authenticate to Vault?
- After authentication, what access is needed within Vault?
- What it takes for developers to adopt Vault into their projects and development?
- How in practice will new developers gain/lose access to using secrets managed by Vault?
- In terms of security, which are the most important features introduced by Vault and will there be any drawbacks?
- Which portion of the company's secrets that are currently managed manually, can be automated with Vault.
- How will Vault affect the administration?
- Which Vault version will be used?
- Where will Vault be deployed?
- How Vault will be provisioned?
- Will Vault be deployed in a cluster?

- Does Vault need to be available in multiple data centers or cloud regions?
- Which storage backend will be used with Vault
- Which secrets engines will be used?
- What performance and operational alerts should be configured?
- On what basis will the Vault audit logs be checked?
- Who will have access to recovery keys?
- How long will the TTL of the secrets be?

3.4 Infrastructure Description

The company's infrastructure has been built mostly on GCP and Azure cloud platforms. The customer projects are run in the Kubernetes cluster that is configured using Helm Charts. GitLab CI is leveraged together with the in-house tool, Kólga, to automate the CI/CD pipeline. The entire infrastucture, including Vault is provisioned using Terraform, which is shown in figure 3.4. [6] [1] [2] [19] [20] [21]

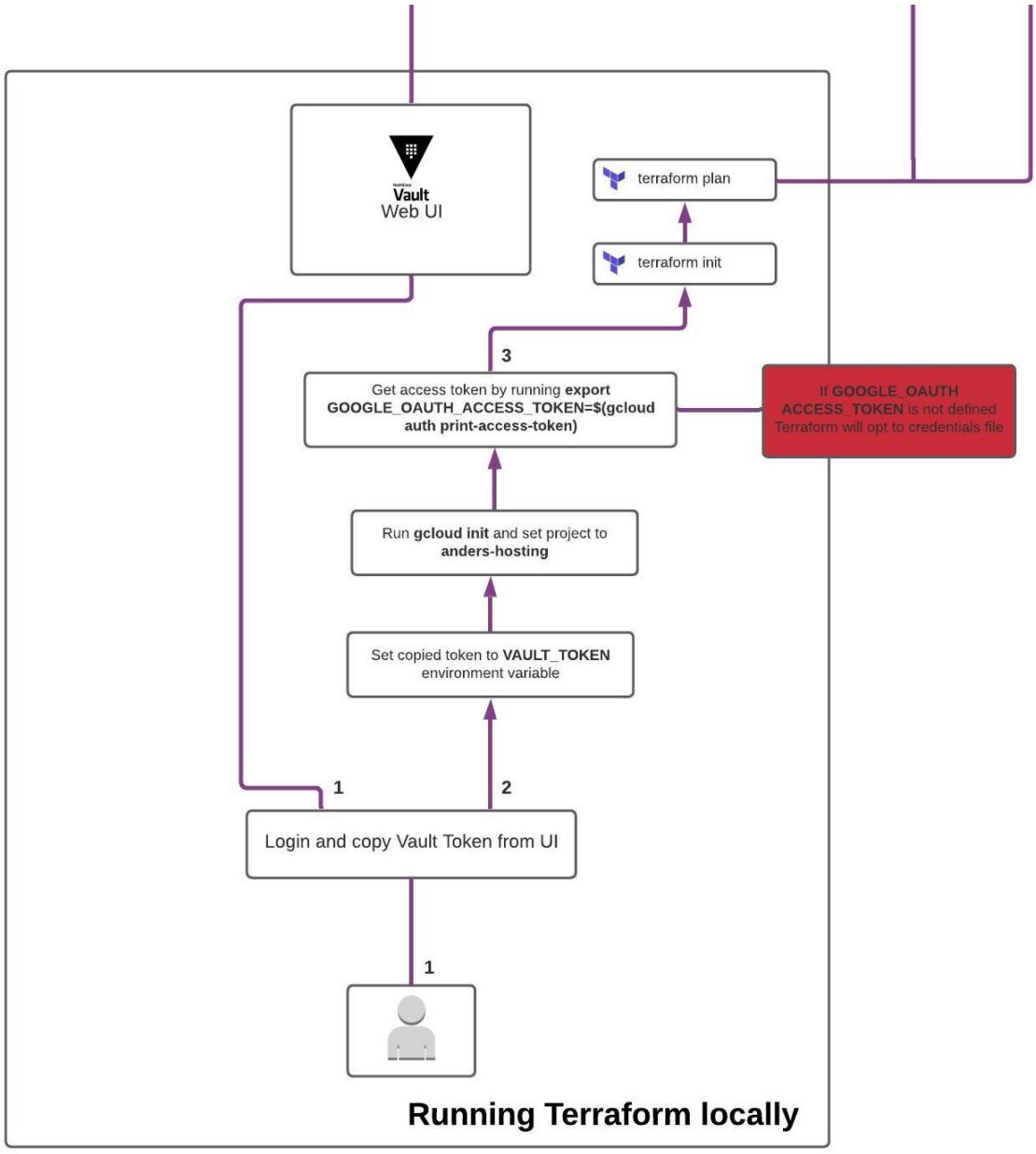

Figure 3.1: Provisioning Vault using Terraform. [22]

Kólga is the in-house DevOps tool, that handles building and deploying the environments in the VPC (Virtual Private Cloud) of the company. For the management of secrets, Kólga is also responsible for fetching the Vault-managed secrets and injecting them to containers as environment variables via its Vault module. The visualization for fetching Vault-managed secrets using Kólga is shown in figure 3.4.

The RBAC is based on GitLab, which will be integrated with Vault, meaning that GitLab users with the sufficient permissions will be able to create, edit or delete Vault-managed secrets. [20] [15]

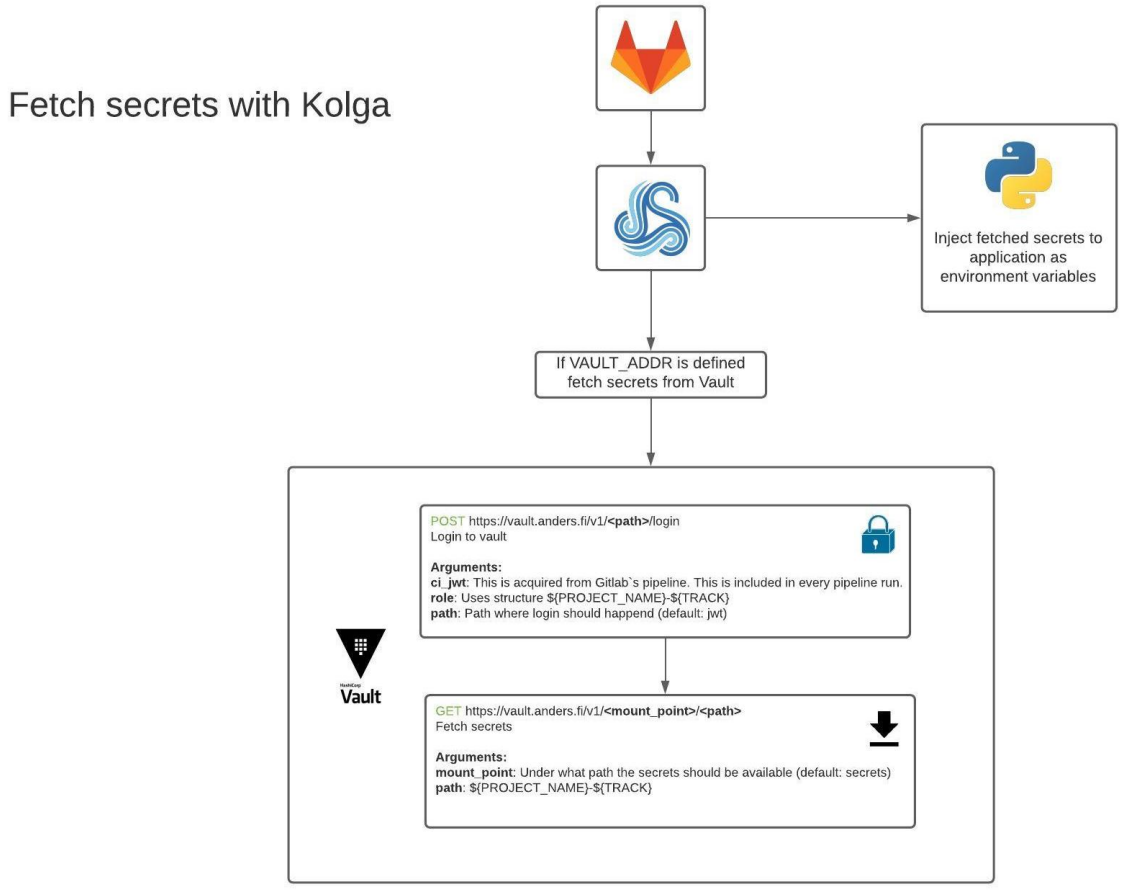

Figure 3.2: Fetching secrets in the CI/CD pipeline with Kólga. [22]

After Kólga has fetched the secrets managed by Vault, it takes care of injecting the secrets to the containers as environment variables using Kubernetes sidecar injection shown in figure 3.4. The "sidecar" in this context refers to a container that is dedicated to injecting the secrets as environment variables. In the example scenario the Kubernetes pod contains one container for running the application and the sidecar container, even though the application could consist of multiple containers.

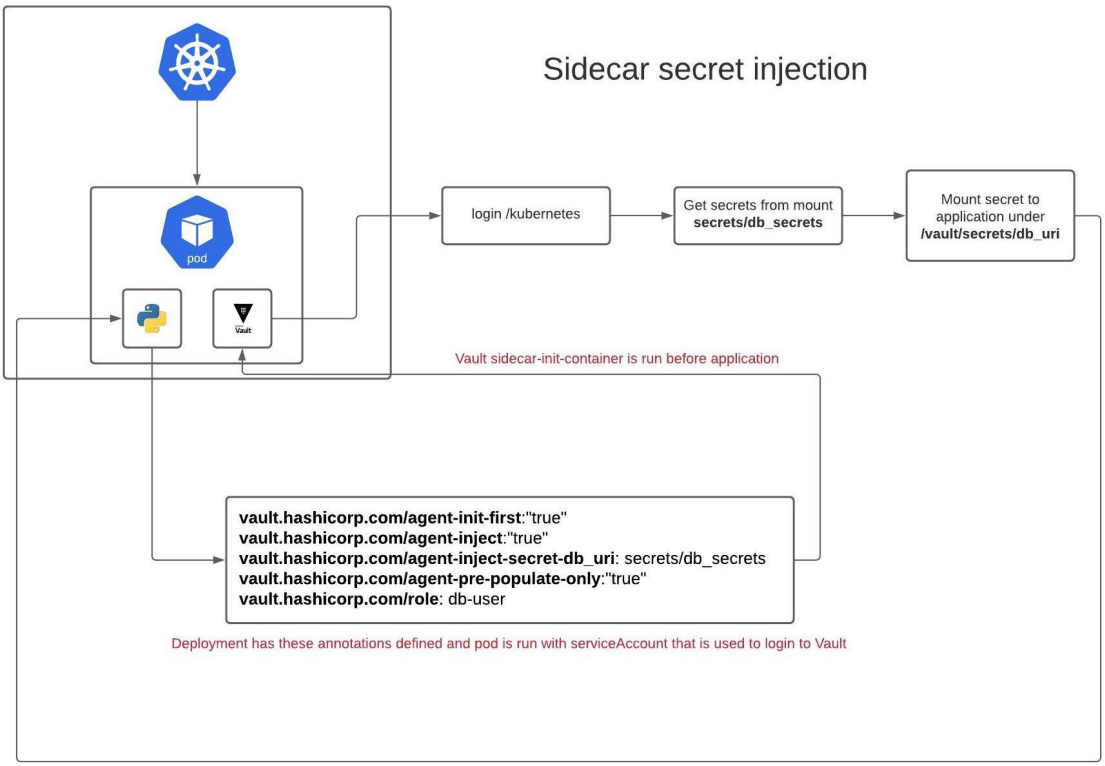

Figure 3.3: Kubernetes sidecar secret injection. [22]

3.4.1 Vault

Vault fits for the use case of this thesis due to its support for multi-cloud purposes, meaning it is capable of integrating with multiple cloud providers and environments. Hashicorp refers to the underlying problem that Vault is trying to solve as "secrets sprawl". It is a state in which all the secrets of a given infrastructure are located in arbitrary pieces of the environment, be it inside the source code or the configuration files. The solution for this is to centralize the management of the secrets [4].

Vault is a secure secret storage as it encrypts the secrets rather than storing them as plaintext. This means that even if an attacker were to have access to the secrets, they would still be of no use without knowing the encryption key. Another key feature of Vault is its dynamic secrets. Dynamic secrets are secrets that can be dynamically created, rotated and revoked for multiple clients [4].

Vault supports creating on-demand secrets for certain services and automate the revocation of those secrets after they are no longer needed. Vault's data encryption feature allows Vault to act as an Encryption as a Service (EaaS) API to create secure credentials. These credentials are not stored anywhere by Vault. Vault's leasing and renewal feature allows clients to subscribe to fixed-term secrets provided by Vault. After the leasing period, the secrets are revoked by Vault. The revocation feature of Vault extends to revocation of secrets by category, e.g. secrets belonging to a specific user or application [4].

For authentication and integration with different services, Vault uses different authentication backends for getting the identity of the caller. In the infrastructure detailed in this thesis, the used authentication backends are OIDC, JWT and Kubernetes shown in figure 3.4.1. This means that Vault-managed secrets can be accessed using e.g. GitLab permissions, JWT tokens and Kubernetes. For auditing and logging the data about the request-response cycle, Vault has its audit logging service. This can be connected to a third-party storage for later use. For storing Vault's internal data, it uses a storage backend. This can also be connected to a third-party storage, such as a relational database. The actual secrets management of vault is handled by its secret backend. In its simplest form it is only a key-value storage. However, Vaults secret backend integrates with plugins that extends its use case to the dynamic secrets management. [4]

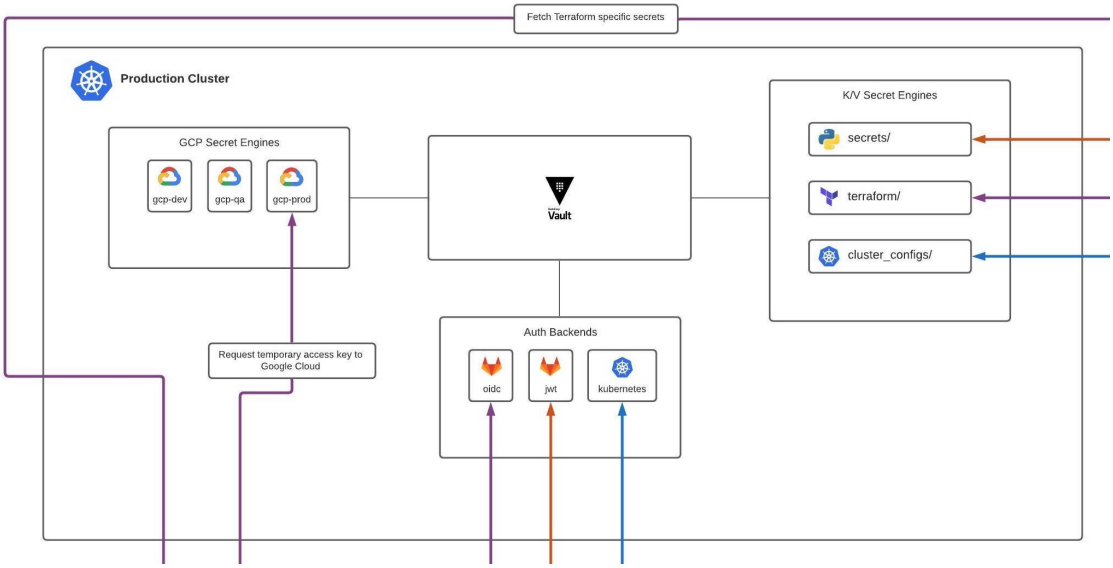

Figure 3.4: Kubernetes-managed cluster for Vault deployment. [22]

To summarize the case, the organizational structure as well as the infrastructure are very typical for an IT provider house, making this case potential for a lot of organizations that are motivated to automate their secrets management. Even though the interview questions are mostly related to the adoption of Vault, the case contains a lot of similarities to organizations that might not have a perfectly matching infrastructure and technology stack.

4 Case Study

This chapter puts the research methods into practice by documenting the interviews as well as by providing the technical examples for the key parts of implementing Vault in an infrastructure similar to one of the case company. The results described in this chapter are not particularly generalized but only for justifying the decisions and solutions made for the given case. More generalized and higher level solutions are presented in chapters 5 and 6. [4]

4.1 Interviews

The interviewees are the following people, who have given consent to use of their names in the results:

- Frank Wicktröm, CTO
- Joonas Venäläinen, DevOps Engineer
- Paul Nylund, CIO

4.1.1 What are the main problems in the current method of managing secrets in the company projects?

According to the whole team behind the implementation of Vault, the lack of a centralized secrets management system is an issue causing the secrets sprawl. The

secrets are managed on a project level and kept close to the applications. Configuring the secrets manually, by accessing servers to modify individual environment variables etc. is too cumbersome and unscalable. Another issue lies within the transparency of the access to secrets - the governance is harder when the people with access to secrets are not explicitly defined, says Wickström. Venäläinen adds that the exact method that is currently used for secretes is GitLab variables, so there is no other way than to set them per project.

4.1.2 What kind of custom code has to be written to make Vault work with other pieces of infrastructure?

The only custom part for full adoption of Vault is the injection of the secrets to different environments. "There's pretty easy ways to pull out the secrets from Vault, but injecting them to the running application has proven to be non-trivial", Wickström states. Custom code has been written to the CI/CD pipeline, that pulls the secrets from Vault and injects them to the applications as environment variables. The functionalities within Vault, however, require no custom implementations. "In our projects we haven't found any use cases that Vault wouldn't cover.", Venäläinen says.

4.1.3 How users and applications will authenticate to Vault?

Currently, GitLab access rights are leveraged for authentication - the groups are automatically used for Vault access. According to Wickström, this is an area that might still experience some changes in the sense that whereas currently, one massive group is used to give access to all people, in the future, projects must be grouped to form structures in a way that the right people gain Vault access.

According to Venäläinen, the authentication happens via GitLab's OIDC provider: "Vault will check in what groups the user belongs in GitLab and grants access to

secrets based on that. Applications can use the `CI_JWT_TOKEN` that is available on every CI (Continuous Integration) run. Token based authentication is also used with Terraform. In Kubernetes when using sidecar to inject secrets service account of running pod will be used to authenticate against Vault."

4.1.4 After authentication, what access is needed within Vault?

The exact rights within Vault are tied to GitLab permissions, so the user can only access things within Vault that the GitLab group of that user has access to. "There are a subset of secrets that are behind a permission wall where it (Vault) checks the group they are in GitLab.", Wickström clarifies.

The answer to this question depends on the use case, according to Venäläinen: "Applications generally only read secrets so read access will be enough. Some users could have `["read", "list", "create", "write", "delete"]` access to allow making necessary changes to secrets. Normally read access should be enough for most users."

4.1.5 What it takes for developers to adopt Vault into their projects?

According to Wickström, two things are needed by developers. Firstly, they need to add their secrets to Vault. They can do it with their GitLab permissions that allows them to access e.g. Vault UI for adding the secrets. Secondly, they need to enable Vault to be used in the CI/CD pipeline. They can do it conveniently by setting the Vault address as an environment variable for the project and after that it will be picked up by Vault. Venäläinen adds that the Kólga integration picks up an environment variable called `VAULT_ADDR`. Kólga will log in to Vault using the `CI_JWT_TOKEN` and fetch all the project secrets.

A Vault project is also needed for each project using Vault secrets. Wickström

states however, that the Vault projects are not for the developers to create, but they will be created by Terraform. So ideally, the Vault project already exists in the point where the developer wants to add Vault secrets into a project.

4.1.6 How in practice will new developers gain/lose access to using secrets managed by Vault?

The rotation of access is fully automated via integration of GitLab permissions. Adding or removing members from GitLab groups automatically reflects to Vault access of the given users, so no loose ends are left after the people come and go. The answer to this is equivalent from all interviewees.

4.1.7 In terms of security, which are the most important features introduced by Vault and will there be any drawbacks?

According to Wickström, the centralized control is the number one security improvement introduced by Vault. Another crucial improvement is the access control for groups. A drawback that Wickström states however, is the fact that currently Vault won't take into account the level of access within GitLab, so a wanted feature would be to include the level of the user's GitLab permissions in the Vault security checks, to allow giving Vault access for e.g. only people with Maintainer-level GitLab access.

4.1.8 How will Vault affect the administration?

According to Venäläinen, the overhead introduced by Vault is minimal at least in the beginning. Al of the interviewees had a similar answer, since only the minimal features of Vault are used in the beginning.

4.1.9 Which Vault version will be used?

The open source version 1.5.4. The requirements set for implementing Vault allow the usage of the open source version, since no features included only in the enterprise version are needed at least in the beginning.

4.1.10 How will Vault be provisioned?

According to Venäläinen, Vault will be provisioned with Helm. All secrets provisioning will be handled through Vault, including Vault itself as well as all the Vault projects, Wickström adds.

4.1.11 Will Vault be deployed in a cluster?

Wickström says that the high availability of Vault is currently not needed. The situation might change in the future however. Venäläinen adds that Vault is deployed in a cluster however, without high availability.

4.1.12 Do we need Vault to be available in multiple data centers or cloud regions?

"Not at this point. When the high availability will be needed, it will be implemented incrementally in the future.", Wickström stated.

4.1.13 Which storage backend will be used with Vault?

Google Cloud Storage was a unanimous answer from all of the interviewees. The decision had been made within the DevOps team.

4.1.14 Which secrets engines will be used with Vault?

KV Secrets Engine - Version 1. Another decision made by the team behind implementing Vault.

4.1.15 What performance and operational alerts should be configured for Vault?

Venäläinen describes the flow as the following: "Vault writes audit log file. This log file should be monitored for anomalies. Vault offers `/metrics` endpoint which allows us to integrate these metrics easily for our monitoring solution which contains set of predefined alerting rules that works for Vault too.". Prometheus will be configured to monitor Vault. Prometheus monitoring will be graphed in Grafana. Logs will be also graphed in Grafana through Grafana Loki in the future, Wickström says. [23]

4.1.16 On what basis will the Vault audit logs be checked?

Venäläinen says that at the time of writing this, it has not yet been decided: "Probably Loki will be configured to automatically check the logs with predefined rules and alert if needed."

4.1.17 Who will have access to recovery keys?

"Not yet decided. Most likely someone from the DevOps team will have the keys. Suggested method would be to share the keys to multiple persons instead one person having them." - Venäläinen

4.1.18 How will the TTL be configured for the secrets?

"As we are only using K/V currently as secret engine TTL is not enabled" - Venäläinen

4.2 Technical Examples

The following technical examples aim to demonstrate, how the secrets management practices handled in the preceding chapters can be implemented in the given infrastructure. The examples focus on working with Vault, Terraform, Kólga, GitLab and Kubernetes. Both developer-facing implementations and administration tasks are covered, showing how the secrets management is done with the given tools. [4] [20] [19] [6]

4.2.1 Configuring Vault to a Project

Kólga contains a Vault-module, that adds support for Vault-managed secrets. The secrets can be injected to all environments automatically, including local, review, staging and production environments. Activating the module is as easy as adding an environment variable specifying the address for the Vault server. With Kólga, the environment variable needed is called `VAULT_ADDR`. Since in this case, a single Vault server takes care of managing multiple projects and their environments and the address will be the same for all projects. Kólga also supports other configurable variables shown in 4.2.1. [20]

Variable	Default	Description
`VAULT_ADDR`		Vault address
`VAULT_TLS_ENABLED`	True	Enable TLS
`VAULT_JWT`		JWT Token used to login to Vault. If using Gitlab will default to CI_JOB_JWT
`VAULT_JWT_AUTH_PATH`	jwt	Path used for authentication
`VAULT_KV_SECRET_MOUNT_POINT`	secrets	k/v mount point where to fetch secrets

Figure 4.1: Configurable environment variables for Kólga's Vault module. [20]

With the help of the `VAULT_ADDR` variable, Kólga will do the authentication via JWT authentication which is at the time of writing this thesis, the only supported method to authenticate to Vault by Kólga. This is due to the built-in support for

GitLab to authenticate to Vault using the JWT method. Each CI job on GitLab CI has a unique job-specific environment variable called `CI_JOB_JWT` that is used to authenticate to Vault. The complete flow of GitLab authentication to Vault is shown in figure 4.2.1.

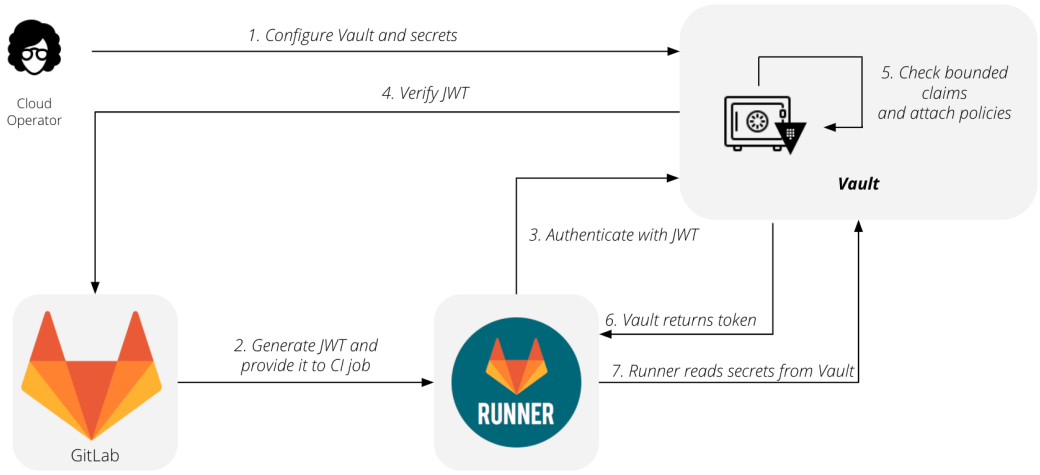

Figure 4.2: Authenticating to Vault using GitLab.
Source: GitLab (https://docs.gitlab.com/ee/ci/secrets/).

4.2.2 Managing Vault Secrets

The Vault secrets can be added by people with the required permissions. OIDC is an authentication protocol that stands for "OpenID Connect". It enables verifying a user's identity before granting access to certain endpoints. It is a further developed version from OAuth and OpenID authentication methods. With OIDC authentication, the Vault permissions will be linked to GitLab. This means that employees with the required GitLab permissions can add the project specific secrets to Vault using either Vault UI or the CLI. They can then add the project specific secrets. [20] [24] [4]

The secrets are stored in Vault under a mount called `secrets/${project_name}-${track}`, where the `project_name` and `track` are dynamic values. The Vault module of Kólga supports secrets for review, staging and production environments (or tracks). The mount paths for those environments go like the following:

- `secrets/$project_name-$review`
- `secrets/$project_name-$staging`
- `secrets/$project_name-$stable`

4.2.3 Vault Administration

Provisioning Vault can be done via Terraform, a popular IaC tool. Terraform contains a provider for Vault, which makes the process of deploying and provisioning Vault convenient. An important consideration when provisioning Vault with Terraform is to note that secrets read or written by Terraform will also be persisted in the Terraform state. This means that the Terraform state that is stored in a `tfstate`-file, must be held behind strong encryption. The snippet below shows and example how Vault can be configured using Terraform in a very minimal form. The language used in the example is called Hashicorp Configuration Language (HCL). [25] [26] [4]

```
provider "vault" {
  address = "https://vault.example.net:8200"
}
```

To enable the OIDC authentication method with GitLab and Vault, the first step is to create an application from the GitLab dashboard. The application ID and secret will be used to authenticate to Vault. After that, the OIDC method must be enabled with Vault like shown below [19]:

```
$ vault auth enable oidc
```

After the OIDC method has been enabled in Vault, the generated application ID and secret must be added to Vault. The OIDC configuration can be written to Vault in the following way, where `your_application_id` and `your_secrets` are placeholders for the credentials of the GitLab application [19]:

```
$ vault write auth/oidc/config \
    oidc_discovery_url="https://gitlab.com" \
    oidc_client_id="your_application_id" \
    oidc_client_secret="your_secret" \
    default_role="demo" \
    bound_issuer="localhost"
```

When the GitLab application credentials have been written to Vault, the same must be done for the OIDC role configuration. It is done in order to provide Vault with the redirect URIs and scopes that were created with the GitLab application. The paths must also match the location of Vault deployment. An example of the OIDC role configuration is shown below, where the `your_vault_instance_redirect_uris` and `your_application_id` are placeholders for the GitLab application credentials and the `yourGroup` and `yourSubgroup` are placeholders for the GitLab groups [19]:

```
vault write auth/oidc/role/demo -<<EOF
{
    "user_claim": "sub",
    "allowed_redirect_uris": "your_vault_instance_redirect_uris",
    "bound_audiences": "your_application_id",
    "oidc_scopes": "openid",
    "role_type": "oidc",
    "policies": "demo",
    "ttl": "1h",
    "bound_claims": { "groups": ["yourGroup/yourSubgroup"] }
}
EOF
```

Completing the previous steps are enough to enable the OIDC authentication method. After that, a user with the required permissions may log in to Vault using either the Vault UI or the CLI. Provided the previous configuration for the OIDC method has been set, a login using the CLI can be done with the following command [19]:

```
$ vault login -method=oidc port=8250 role=demo
```

The "port" command line argument must match with which ever value has been set for the redirect URIs of the GitLab application. The "role" command line argument refers to the configuration that was previously created.

5 Solution

The technical examples shown in 4 prove that any organization can set up a secrets management platform suitable for multi-cloud environments with a fairly low amount of work. The transformation towards the more automated security operations is not only technical but also cultural. Like introducing DevOps into the software delivery process by starting the process from the culture, same same goes for DevSecOps and a subset of it such as secrets management. [5]

5.1 DevSecOps Practices

The fact that the applications are run in containers often times these days, makes the basic principle of injecting the secrets as environment variables a very generalized solution regardless of the platforms in use. Container orchestration tools, such as Kubernetes, places little overhead on the process of injecting the secrets, thanks to a good level of support in integrating DevOps platforms, such as GitLab, to secrets management platforms like Vault. [19] [4]

In large organizations, the use of RBAC is imperative, placing support for one as a technical requirement for a secrets management platform. Without the advantages of RBAC that can be attached to existing IAM, the administration of the secrets management platform ends up being laborious and manual. Having a centralized access management however, is a perfect example of leveraging DevOps practices and in security terms, DevSecOps practices. [15]

Depending on the infrastructure and the amount of secrets sprawl, the adoption of automated secrets management can be implemented with ease or it may require a lot of configuration. Cloud-agnostic container orchestration tools like Kubernetes makes the process easier in the sense that the cloud environments can take place in any of the known providers' platforms, but on the other hand, increase the overhead due to their complexity by a great amount. [27]

The benefits of cloud-agnostic secrets management platforms are imminent for organizations running multi-cloud environments. One of the biggest advantage comes when such platform integrates with the RBAC systems used already in the organization. GitLab is a good choice due to its roles and group permissions. Another commonly used developer platform Github, similarly allows authentication to Vault based on existing roles of an organization. With the integration support of just the two mentioned platforms, most organizations are already covered, setting the requirements on the shared access control schemes between the secrets management platform and the repository manager as accomplished for most organizations. [19] [4] [28]

5.2 Requirements & Deployment

When organizations get started with a secrets management platform like Vault, requirements must be gathered for security and the end-user. The security requirements include e.g. restricting access to Vault nodes, securing the underlying operating system (OS), encryption key protection, checking audit logs, securing Vault with TLS and configuring the TTL. The end-user requirements may include e.g. methods for users and applications to authenticate to the secrets management platform, the required access within the platform, the secrets engines used and the availability. [18] [4]

Configuring operational, security and performance monitoring is another important task to do when taking a secrets management platform into use. Operational monitoring means real-time monitoring on events. An example of an operational monitoring event would be a system failure needing instant response to fix it. In theses kind of cases, the operational monitoring comes into play by notifying the administrators in real time. [18]

Security monitoring is a proactive method to get a heads-up from a possible threat or to minimize the time to take action in an incident response. Security monitoring events might include events such as the use of a root token, security policy modifications, configuring auth methods or even permission denied responses.

Performance monitoring is also important when an organization wants to ensure the availability of a secrets management platform. Example metrics tracked by performance monitoring include e.g. CPU, disk and memory usage, response time and the storage backend transactions. [18]

A dedicated support team is recommended to exist when a secrets management platform like Vault is adopted into the infrastructure of an organization. Such team must be equipped with the required experience. An ideal line-up for such team is to have personnel with automation and application integration experience. An example for the team would consist of:

- Site Reliability Engineer
- DevOps Engineer
- Senior DevOps Engineer

The requirements and the experience level for each position complement each other so that they complement each other and thus form a suitable support team. [18]

Chapter 2 shows that a full-featured secrets management platform like Vault has anything that an organization that currently manages its secrets manually, could use to either improve the security or reduce the lead times via automation. Chapter 4 shows that an organization using a common developer platform like GitLab or GitHub, can get away with a relatively low level of configuration for such platform. [19] [28]

The benefits of secrets management platforms compound with the organization size. A future consideration for the research conducted in this thesis would be to find out the optimal company size in which the benefits of a secretes management platform would outweigh the time lost in configuring one. However, as shown in chapter 4, the amount of work needed from both developers and administrators of such platform remain so small that the benefits could be significant event for small companies.

5.3 Answering the Research Questions

The research questions were aimed for general audience that could leverage the answers regardless of the infrastructure of their organization. However, containerized architecture is a requirement for someone who can benefit from the information as the main research took place around the idea of injecting the secrets into the containers.

5.3.1 How secrets management is handled in multi-cloud environments?

Containers offer a way for an organization to operate scalably in multi-cloud environments. With Kubernetes and many other container orchestration platforms, the principles for secrets are the same; the secrets are injected into the containers as environment variables. A required component however, is a service that handles fetching the secrets from the secrets management platform in use and handling the injection. Luckily for many organizations, the mainstream platforms are capable of this, as noted in previous section 5.2.

5.3.2 How secrets can be used in automated build environments?

The automated build environments might have a great variance among organizations. However, containers provide a unified solution that can be applied for most infrastructures. Using secrets managed by a secrets management platform requires an additional software component for multi-cloud environments as none of the developer platforms covered in this thesis is capable of spinning up such environments out of the box. An organization might have to bring in another DevOps tool at this point, like Kólga that was covered in chapter 4.

5.3.3 How can access rights management of a secrets management platform be combined with third party services?

The answer to this research question is very similar than on the RQ1. There's a good likelyhood that an organization using a mainstream developer platform can combine the access rights with the developer platform. In case of Vault, both GitLab and GitHub are optimal choices in this sense. By combining the access rights

management, an organization can easily restrict access to e.g. employess leaving a specific project or the entire company only in the developer platform that then syncs the permissions with the secrets management platform. Authenticating and authorizing to the secrets management platform can also be done with the same access rights. An example of this is the LDAP protocol used by GitLab and which can also be used when authenticating to Vault, described in 4.

6 Conclusion

This chapter concludes the thesis by reviewing the observations and findings made along the way of completing the literature review and the case study. The goal of this chapter is also the make the outcome more holistic in the sense that the solution could be applied to more organizations.

6.1 Concluding Secrets Management

In chapter 2, literature review was done on secrets management, focusing on the features and use cases of secrets management platforms. The chapter presented the use cases that an organization can benefit from such platform. By not only speeding up the software delivery process, the chapter also showed what kind of improvements a secrets management platform can bring to an incident response process and help to locate the origin of the breach. [7]

To begin with secrets management, one must understand the definition of secrets. To summarize the definition of them in brief, the quote by Armon Dadgar, the Co-Founder and CTO of Hashicorp explains it well: "When we talk about secret management what we're really talking about is managing a set of different credentials. What we mean when we talk about these credentials is anything that might grant you authentication to a system or authorization to a system. Some examples of this might be usernames and passwords, it might be things like database credentials, it might be things like API tokens, or it might be things like TLS Certificates"

[4]

In chapter 2 a good amount of methods that a secrets management platform can help an organization to mitigate security incidents were introduced. The features that a platform can provide regarding sharing secrets and dividing them into multiple instances for different users and applications, provisioning machine identities as well as using ephemeral secrets and aggregating logs into a form that auditing them can be automated, are the most important ones that a platform can provide. [7]

Taking a secrets management platform into use differs for each organization depending on their infrastructure, team structure, expertise and even culture. The role of chapter 3 was to justify the choices made for the case and how they fit for the organization. The underlying problems might also have variety between companies. Their secrets management is done based on what they have been accustomed to. Some organizations may experience a greater problem with secrets sprawl, whereas some others might think that the automation of their mundane tasks are the first priority. [18]

Public-key infrastructure is a key part of a secrets management that allows efficient sharing of the secrets by multiple users or applications. Especially in multi-tenant systems, the separation between the access of the different entities is important for being able to restrict access as well as to improve the security by provisioning the identities of the tenants. Regarding Vault, the method called identity-based security is a good example of a security-improving model that separates the clients, authentication and secrets within a single system, forming a zero-trust network [7] [4].

Depending on the needs of an organization, picking a secrets management solution that supports the sufficient API access must be taken into account. While programmatic access might be enough for some organizations, also a GUI support is useful for organizations that want to make the administration possible for employees with no deep knowledge about the system. For instance, with Vault UI, a developer with the required permissions can manage project-specific secrets relatively easily. [4]

When organizations adopt a new secrets management platform, they want put effort not only in automating the practical workflows around managing the secrets but also making sure that they configure sufficient methods to act accordingly in case of security incidents. This kind of work includes e.g. configuring operational alerts or deciding on the employees that have a access to recovery keys of such platforms. With the right configuration, a secrets management platform can provide a great deal of improvements into the incident response of an organization. [7]

6.2 Concluding the Case

In chapter 4 there were two key sections; the interviews and the technical examples. The interviews were aimed to get an understanding on the technical considerations that and organization has to make when adopting a secrets management platform like Vault. While the questions were all made around Vault, many of them can also be applied on alternative platforms. Likewise, the technical examples were focused around Vault, GitLab and Kólga in the case, but the same basic principles can be applied when injecting secrets into containers using alternative solutions. [4] [19] [20]

As the case description in chapter 3 showed, Kubernetes suits well for injecting secrets managed by a platform like Vault. The sidecar injection allows the separation of the interaction between Kubernetes and the secrets management platform in an

isolated environment from the application containers, while both of them still being located in the same pod. [6]

The case study showed as well that an organization can start with very minimal configuration in the sense that the different steps for adopting a secrets management platform can be done gradually. For instance high availability is a feature that most organizations do not need immediately but it can be considered as something that can be upgraded to when the need appears.

A good practice when provisioning a secrets management platform like Vault is to do it using a tool like Terraform. Along with other benefits of IaC, an increased transparency for the configuration of Vault makes the administration more straight forward especially when multiple people are included in the process. OIDC authentication method allows easier administration as well, waiving the need for additional IAM. As protocols like LDAP are widely used, a lot of organizations are set for configuring these kinds of shared authentication schemes. [29] [12]

The case study covered the key component of the continuous integration, Kólga and the Vault-module of it in not too detail, making the observations not too tied to the given infrastructure. The whole process of fetching the secrets managed by a platform like Vault and injecting them into the containers, is the part of the pipeline that might require the most input from an organization that is in plans of automating their secrets management. [20] [4]

The technical examples of the case study showed that the improvements in the speed of the whole secrets management process are undeniable for the case and all organizations that are running a similar kind of infrastructure. The fact that the company included in the case study, Anders Innovations, has been running their multi-cloud environment including the production environments proves that the migration to a secrets management platform can be done relatively rapidly even for a mid-sized company.

6.3 Future Considerations

This thesis has a high focus on specific technologies such as Vault or GitLab. A suitable continuation for the research done in this thesis would be to compare the other available platforms and to investigate, what kinds of requirements they might have for an organization that could adopt them. For instance, the configuration of Vault covered in 4 chapter might not apply to many of the alternative secrets management platforms. That is also why this thesis does not provide a perfect one-off solution for organizations but it rather covers the theoretical part of secrets management that in general, is very similar in all containerized multi-cloud infrastructures. [4] [19]

In addition to the general theory of secrets management covered in this thesis, the usage of the secrets was limited to injecting them into containers. Although the containerized infrastructure is widely used in this cloud computing era, this research could be extended to non-containerized architectures such as serverless, which means that the applications are not running in containers but are running directly in virtualized runtime environments that execute them in functions. [30]

This thesis assumes that an organization in plans to migrate its manual secrets management to an automated platform has the required expertise to implement it. However, it is imminent that a lot of organizations lack this kind of expertise. This problem could lead to researching methods for these kinds of organizations on what kind of hiress they would have to make. Alternatively, these kinds of organizations could use fully managed solutions that require no expertise in the area of technical implementation and DevSecOps in general. So a similar research to this, could be conducted on the managed solutions to make it clear for such organizations on how they could exactly benefit from a managed secrets management platform and how would they take one into use. A managed secrets management platform would mean that the organization would not have to deploy or provision the platform but they would get a turnkey solution that they could administer without extensive technical

knowhow. [3]

The security impacts of a secrets management platform can be noted effectively in the long term and it would most likely require data from more than just one case study. The amount of security indicents that take place with manual vs. automated secrets management is an area that could also be further investigated.

Another area that would be interesting to further investigate in long term, is the impacts that the automation has in the workflow of both developers and administrators in terms of saved time. For this case, making sure measurements would've been hard as the implementation of Vault took place in the same time of writing this this thesis and gathering such data was not possible. [4]

References

[1] Google, "Google cloud platform", 2021. [Online]. Available: https://cloud.google.com.

[2] Microsoft, "Microsoft azure", 2021. [Online]. Available: https://azure.microsoft.com.

[3] J. A. Morales, T. P. Scanlon, A. Volkmann, J. Yankel, and H. Yasar, "Security impacts of sub-optimal devsecops implementations in a highly regulated environment", in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20, Virtual Event, Ireland: Association for Computing Machinery, 2020, ISBN: 9781450388337. DOI: 10.1145/3407023.3409186. [Online]. Available: https://doi.org/10.1145/3407023.3409186.

[4] Hashicorp, "Vault docs", 2020. [Online]. Available: https://www.vaultproject.io/docs.

[5] S. Jones, J. Noppen, and F. Lettice, "Management challenges for devops adoption within uk smes", in *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*, ser. QUDOS 2016, Saarbrücken, Germany: Association for Computing Machinery, 2016, pp. 7–11, ISBN: 9781450344111. DOI: 10.1145/2945408.2945410. [Online]. Available: https://doi.org/10.1145/2945408.2945410.

[6] T. L. Foundation, "Kubernetes docs", 2020. [Online]. Available: https://kubernetes.io/docs.

[7] Securosis, "Understanding and selecting a secrets management platform", 2018. [Online]. Available: https://cdn.securosis.com/assets/library/reports/Securosis_Secrets_Management_JAN2018_FINAL.pdf.

[8] P. Ghosh, Q. Nguyen, and B. Krishnamachari, "Container orchestration for dispersed computing", in *Proceedings of the 5th International Workshop on Container Technologies and Container Clouds*, ser. WOC '19, Davis, CA, USA: Association for Computing Machinery, 2019, pp. 19–24, ISBN: 9781450370332. DOI: 10.1145/3366615.3368354. [Online]. Available: https://doi.org/10.1145/3366615.3368354.

[9] S. Idreos and M. Callaghan, "Key-value storage engines", in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20, Portland, OR, USA: Association for Computing Machinery, 2020, pp. 2667–2672, ISBN: 9781450367356. DOI: 10.1145/3318464.3383133. [Online]. Available: https://doi.org/10.1145/3318464.3383133.

[10] L. Y. R. Zhao, "Strengthen cloud computing security with federal identity management using hierarchical identity-based cryptography", 2009. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-10665-1_15.

[11] M. Gaedke, J. Meinecke, and M. Nussbaumer, "A modeling approach to federated identity and access management", in *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, ser. WWW '05, Chiba, Japan: Association for Computing Machinery, 2005, pp. 1156–1157, ISBN: 1595930515. DOI: 10.1145/1062745.1062916. [Online]. Available: https://doi.org/10.1145/1062745.1062916.

[12] X. Wang, H. Schulzrinne, D. Kandlur, and D. Verma, "Measurement and analysis of ldap performance", *IEEE/ACM Trans. Netw.*, vol. 16, no. 1, pp. 232–

243, Feb. 2008, ISSN: 1063-6692. DOI: 10.1109/TNET.2007.911335. [Online]. Available: https://doi.org/10.1109/TNET.2007.911335.

[13] M. Morbitzer, M. Huber, and J. Horsch, "Extracting secrets from encrypted virtual machines", in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '19, Richardson, Texas, USA: Association for Computing Machinery, 2019, pp. 221–230, ISBN: 9781450360999. DOI: 10.1145/3292006.3300022. [Online]. Available: https://doi.org/10.1145/3292006.3300022.

[14] S. Halevi and H. Krawczyk, "Public-key cryptography and password protocols", *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 3, pp. 230–268, Aug. 1999, ISSN: 1094-9224. DOI: 10.1145/322510.322514. [Online]. Available: https://doi.org/10.1145/322510.322514.

[15] L. Giuri, "Role-based access control: A natural approach", in *Proceedings of the First ACM Workshop on Role-Based Access Control*, ser. RBAC '95, Gaithersburg, Maryland, USA: Association for Computing Machinery, 1996, 13–es, ISBN: 0897917596. DOI: 10.1145/270152.270176. [Online]. Available: https://doi.org/10.1145/270152.270176.

[16] V. HashiCorp, "Protecting machine identities: Blueprint for the cloud operating model", 2019. DOI: https://www.hashicorp.com/resources/protecting-machine-identities-blueprint-for-the-cloud-operating-model.

[17] L. Encrypt, "Let's encrypt docs", 2020. [Online]. Available: https://letsencrypt.org/docs/.

[18] B. K. Dan McTeer, "Running hashicorp vault in producion", 2020. [Online]. Available: https://www.amazon.com/Running-HashiCorp-Vault-Production-McTeer-ebook/dp/B08JJLGMZ3.

[19] Gitlab, "Gitlab docs", 2020. [Online]. Available: https://docs.gitlab.com.

[20] A. Innovations, "Kolga docs", 2021. [Online]. Available: https://github.com/andersinno/kolga.

[21] C. N. C. Foundation, "Helm docs", 2021. [Online]. Available: https://helm.sh.

[22] Anders Innovations, *Guides and internal documentation.*

[23] G. Labs, "Grafana docs", 2021. [Online]. Available: https://grafana.com/docs.

[24] S. Hammann, R. Sasse, and D. Basin, "Privacy-preserving openid connect", in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 277–289, ISBN: 9781450367509. DOI: 10.1145/3320269.3384724. [Online]. Available: https://doi.org/10.1145/3320269.3384724.

[25] Hashicorp, "Terraform registry", 2021. [Online]. Available: https://registry.terraform.io.

[26] ——, "Terraform docs", 2021. [Online]. Available: https://www.terraform.io/docs.

[27] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson, "Toward cloud-agnostic middlewares", in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '09, Orlando, Florida, USA: Association for Computing Machinery, 2009, pp. 619–626, ISBN: 9781605587684. DOI: 10.1145/1639950.1639957. [Online]. Available: https://doi.org/10.1145/1639950.1639957.

[28] Github, "Github docs", 2021. [Online]. Available: https://docs.github.com/en.

[29] M. Artač, T. Borovšak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "Devops: Introducing infrastructure-as-code", in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17, Buenos Aires, Argentina: IEEE Press, 2017, pp. 497–498, ISBN: 9781538615898. DOI: 10.1109/ICSE-C.2017.162. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.162.

[30] N. Kaviani, D. Kalinin, and M. Maximilien, "Towards serverless as commodity: A case of knative", in *Proceedings of the 5th International Workshop on Serverless Computing*, ser. WOSC '19, Davis, CA, USA: Association for Computing Machinery, 2019, pp. 13–18, ISBN: 9781450370387. DOI: 10.1145/3366623.3368135. [Online]. Available: https://doi.org/10.1145/3366623.3368135.