



Kunnskap for en bedre verden

DEPARTMENT OF INFORMATION SECURITY AND  
COMMUNICATION TECHNOLOGY

IIKG3005 - INFRASTRUCTURE AS CODE

---

# Exploration of how various secret management approaches for Infrastructure as Code can be implemented

---

*Authors:*

Fernandez Marin, Sofia Xiaofan  
Godana, Abdurahman Hadjiawel  
Karpowicz, Michael Angelo  
Zerai, Mehari Berhe

10th November, 2021

---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Secret Management</b>	<b>1</b>
<b>3</b>	<b>Secret Management approaches</b>	<b>2</b>
3.1	Encrypted Secrets . . . . .	2
3.2	Secret less authorization . . . . .	2
3.3	Runtime secret injection . . . . .	4
3.4	Disposable secrets . . . . .	4
<b>4</b>	<b>Secrets Management Technologies</b>	<b>4</b>
4.1	Docker Secrets Management . . . . .	4
4.2	HashiCorp Vault . . . . .	6
4.3	AWS Secrets Management . . . . .	8
4.4	Kubernetes Secret Management . . . . .	9
4.5	Git-crypt . . . . .	10
4.6	Chef Vault . . . . .	11
<b>5</b>	<b>Best technology</b>	<b>13</b>
<b>6</b>	<b>Reflection</b>	<b>14</b>
	<b>Bibliography</b>	<b>15</b>

---

# 1 Introduction

In this digitalized era, it is very important to properly manage the IT infrastructure in an organization. As the IT operation are shifting from iron age to cloud age. The progress to manage IT infrastructure changed from manual to an automate alternative IaC<sup>1</sup>. The main concern of IaC is managing and provisioning computer data into human readable files. There are three resources provided by an infrastructure platform: compute, storage, and networking resources. Those resources are combined and packaged in different ways. For instance, VM machine, container instances, and database instances in our platform w/c combines compute, storage, and networking in primitive ways. Cloud platforms combine all primitive infrastructure into composite resources, such as DBaaS, Load balancing, DNS, Identify management, Secret management. Our focus in this paper is on **secret management** infrastructures which is one type of storage resources.

## 2 Secret Management

The secret management are the tools and methods that helps managing digital authentication credentials. It can be user or auto-generated passwords, SSH keys, APIs or another application keys/credentials, database passwords, TLS or SSL, PGP, RSA, or tokens among others. Secret management are mostly referred to IT in DevOps environments, tools, and processes.

It is important as the passwords and keys are the most common and popular tools to authenticating users and applications and provide them different services. This information must be secret, so it has to be transmitted securely, secret management has to achieve this while theses secrets are in transit and in rest.

There are several risks and considerations related to secret management:

- **Incomplete visibility and awareness:** In one organization can be millions of SSH keys. This becomes a problem because the key starts to be decentralized as different persons of the organization manage their secrets separately.
- **Hardcoded/embedded credentials:** Sometimes there are applications and IoT devices where the default credentials are hardcoded, which the hackers find easy to attack.
- **Privileged credentials and the cloud:** The cloud and each virtual machine that provides superuser privileges to users, comes with its own set of privileges and secrets that need to be managed.
- **DevOps tools:** The DevOps environments are, in this moment, the most affected when managing secrets. This is because all the DevOps environments has a large proliferation and decentralization of secrets. Different static secrets are located in various environments and managed by different administrators and no unified platform is available for the management of these multiple secrets' repositories.
- **Manual secrets management process:** When more manual secrets management process, more likelihood of security gaps.

For this risks, there are some practices we should focus to try to avoid this problems:

- **Discover/identify all types of passwords:** Bring all the keys and secrets under a centralized management.
- **Eliminate hardcoded/embedded secrets:** When using DevOps tools, build scripts, production builds, applications, among others, bring hardcoded credentials under management. Never the less, eliminating hardcoded and default passwords removes dangerous backdoors.

---

<sup>1</sup>Infrastructure as Code

- 
- **Enforce password security best practices:** Not share the secrets and enforce the password making it longer, complex, using rotation etc. If the secret is shared, change it immediately and use one-time passwords and rotation is more sensitive systems .
  - **Apply privileged session monitoring to log, audit, and monitor:** Enable IT teams to highlight suspicious sessions and let them pause, lock or terminate a session.
  - **Threat analytics:** Analyze secrets usage to detect anomalies and potential threats continuously. This is why the more centralized the secrets, the more you will be able to report risks.
  - **DevSecOps:** Having into account that everyone is responsible of the DevOps security. It also has to be build secure since the first step of the lifecycle (design) to the maintenance.

By using this recommendations and adding some other more, you will ensure that users and applications have the right permissions. Restrictions and separation of privileges help condensing the attack surface.

## 3 Secret Management approaches

Any storage resources can be encrypted as if we can store passwords, API keys, database credentials, encryption keys, and other sensitive configuration settings (email, username, debug flags). So that the attackers might not gain privileged access to exploit the file system. A secrets management service adds functionality specifically designed to help manage these kinds of resources. Let us see “Handling secret as parameters” as of the good secret management techniques and infrastructure code. The secrets are not recommended to put in code, so that let us see few approaches how to handle secrets needed by infrastructure code.

### 3.1 Encrypted Secrets

The strategies to handle secret in repositories should be guided by the rule of never put a secret in resource code unless it is encrypted. We have several tools to encrypt secret in repository like, git-crypt, blackbox, sops, and transcript. However, the “key” to decrypt those secret should not be put in the same repository in order to keep away from attackers. We need to have another strategy to handle those keys, it’s mainly recommended to use the following approaches.

### 3.2 Secret less authorization

This is a good approach to authorize an action without using a secret. Now a days some of the cloud platforms have this functionality that can authorize a compute service such as virtual machine or container instance as authorized for privileged actions. We can mention an AWS EC2 instance as of good examples, which can be assigned an IAM profile that gives processes on the instance rights to carry out a set of an API commands. In old days all application runs on an EC2 instance must include AWS credentials in the AWS API requests. Developers store AWS credentials within the EC2 instance and allow applications in that instance to use those credentials directly. However, this would happen under the good control of developers, and they ensure that as they pass credentials securely to each instance and update each EC2 instance every time. But now a day instead we can use an IAM role to manage temporary credentials for applications that run on EC2 instance. Those temporary credentials can then be used in the application’s API calls to access resources and limit access to only resources that the role have specifies. Using role has many advantages one, credentials are temporary and rotated automatically, second you don’t have to manage credentials, and the third you don’t worry about long term security risks.

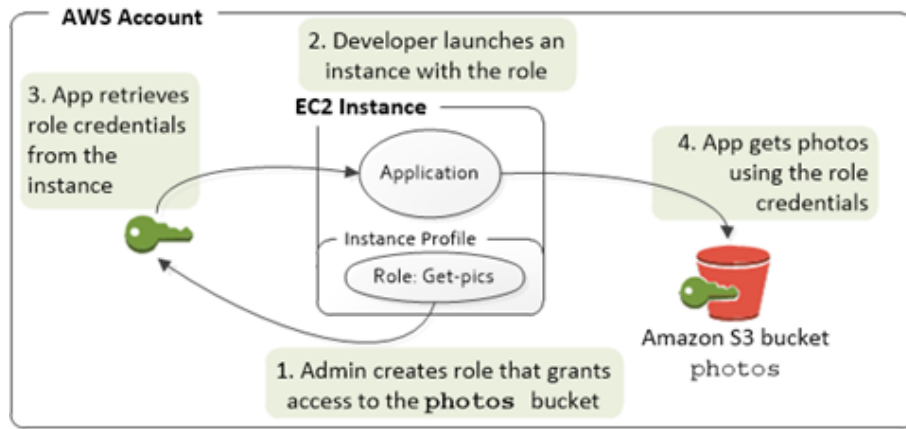


Figure 1: How do roles for EC2 instances work

Source: AWS identity and access management user guide.

For the business owners its recommended that to deploy password less authentication tools on their websites, applications, software, and office devices to strength security and provide a convenient login experience to their customers and employees. Unless hackers can gain an access to the business resources by using brute force attacks or they can buy lists of breached passwords on the dark web.

The strength of password less authentication.

1) It improves user experience: - Users do not need to set passwords and memorize strong passwords. Password less authentication provides users a stress-free experience. Users just need to enter their user ID or phone numbers to receive one-time passwords (OTP), PIN, generated token code, or biometric. 2) No worries for password theft. No more stress for data breaches as a result from password compromises. For public-facing websites, OTPs and “magic links” provide greater security links than many other passwords. 3) protects against brute-force attack: - this is the hacker trial and error methods for guessing the password. However, limited login attempt option been enabled to protect websites from brute force attacks. But still attackers use other methods to bypass the “limit login attempt” features. i) by password spraying: - attackers apply most commonly used passwords for millions of users IDs.by repeating this the process they may find matching set of insecure credentials. ii) Brute-Force attacks via Botnets 4) Password less authentication Strengthens organizations cyber security posture: - If the company invested on the robust password less authentication solutions like PKI client certificates with hardware token device, they would assure that only legitimate employees will access the confidential accounts and hardware resources. By doing this companies will reduce the risk of an employee’s passwords to be leaked which leads to security disaster. 5) Password less security helps to reduce cost in long run. Reduces money that companies invested on password storage, management, and rests. Simultaneously IT departments would not need to constantly put password security policies.

Weakness of Password less authentication.

1) Can’t protect users in event of device theft/SIM Swapping. If someone lost his/her mobile device, an attacker or somebody who er interested can use the device to intercept all OTPs, PIN, and magic links that has been generated on the apps or sent over SMS text messages. In password-based authentication may someone steal the device can’t get access easily to user’s device. Therefore, in this context password less authentication riskier than the password protected-based authentication. For instance, let’s consider a SIM swapping attack, this will take place when someone tricks a mobile service provider into transferring your SIM card to them by pretending to be you, or falsely claiming as you lost your SIM card, then after asking for SIM replacement. If the perpetrator succeeded in this way, then he easily uses SIM card to intercept all SMS messages and access all the apps that relay on SMS-based authentication. 2) Biometric aren’t Fool proof: - hackers can mislead password less security technology by showing images using machine learning to create morphed images of the planed targets, or others like sound recording or videos by voice

---

cloning technology. 3) Users are hesitant about trusting password less technology. Since the concept password was introduced in early 1960s and it is becoming fundamental for authentication and security for every login activity like, e-mail accounts, applications, and websites. However, some people started using password managers to manage multiple passwords, its less familiar than traditional password-based security. Moreover, that password less authentication requires users to put in OTPs, or PIN for every login, but people are more resistant to this change. 4) Implementation cost will be high: - The software/application that enable OTP/magic link facilities on your websites will costs form 25upto1000 per month. On the other hand if employees lose their hardware device like card, or token, companies will cost other expense to replace those tools which is more expensive than resetting passwords. 5) In some cases Password less authentication doesn't protect against certain types of malware: - for example, malwares that is designed for spyware attacks can intercept OTP because this spyware attack can take screen shoots and records from your device screen.

### 3.3 Runtime secret injection

Runtime secret injection is one of the few approaches for handling secrets without having to putting them into code. If you must use a secret for an infrastructure as code, you can inject it at runtime. There are two ways of achieving this:

**Local development:** Keeping the file local and not store it in the version control. In this way the stack can read the file directly. This is useful in the stack configuration files, where you manage different parameters in separate files. You can also work with stack environment variable and setting the secrets using a script.

**Unattended agents:** The server that runs the agent is the one keeping the secret, used in CI testing or CD delivery pipelines, this is because as they are automated, it is easier to keep the secret on the server or container that is running it. You can also provide secrets to the stack command or pull secrets from a storage resource using the stack parameter registry, where the values are stored in a central location instead of in the stack code.

### 3.4 Disposable secrets

The disposable secrets are passwords that are generated automatically. As the example given in the book "Infrastructure as Code" by Kief Morris, in a database, a password is generated by the code that provisions the database and then is passed to the code that provisions the application. In this way, we do not need to know the secret. If needed, it can be applied a reset password if the application server is rebuilt.

## 4 Secrets Management Technologies

In this section we will discuss different technologies that can be used to compose secret management architecture.

### 4.1 Docker Secrets Management

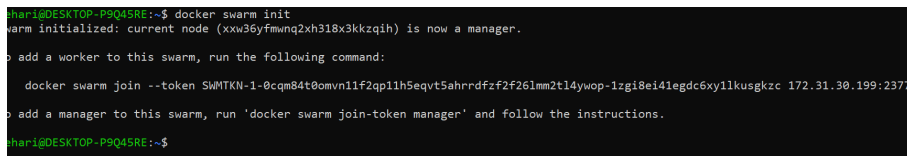
Docker is a technology that helps users to deploy different applications in a cloud or desktops as a separate service in a container. These containers work as an independent of the other container and services in that particular machine or server. Docker uses a special file called Dockerfile, which helps to cock your image with the different specification that you wanted, or we can use the docker daemon to run a container form pre-defined images that are already in the docker hub. But the main point here is how can docker help in secret management of IaC. In docker swarm we can use the "docker secret" command to make our password and keys encrypted and limit container

---

access or securely transmit it to those containers that they need to access to the services by using the encrypted password. This can be done by creating a password file and encrypt them by using the secret command in docker to encrypt the password so that accessing becomes limited. A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

The following figures are a screenshot that we took from our computer to implement a secret in a docker swarm in which we just created in our desktop to demonstrate how to create a secret from a file and use it in a service container.

In the figure 2, creating a docker swarm with only one node the manager.



```
mehari@DESKTOP-P9Q45RE:~$ docker swarm init
Swarm initialized: current node (xxv36yfmwng2xh318x3kzqih) is now a manager.

To add a worker to this swarm, run the following command:

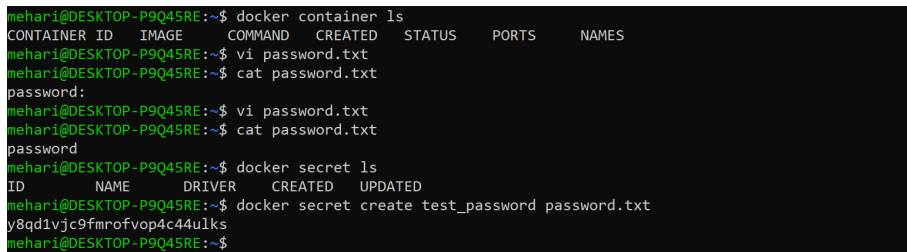
    docker swarm join --token SWMTKN-1-0cqm84t0omvn11f2qp11h5eqvt5ahrndfzf2f26lmm2t14ywop-1zgi8ei41egdc6xy11kusgkzc 172.31.30.199:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

mehari@DESKTOP-P9Q45RE:~$
```

Figure 2: Creating a swarm with only one node(manager)

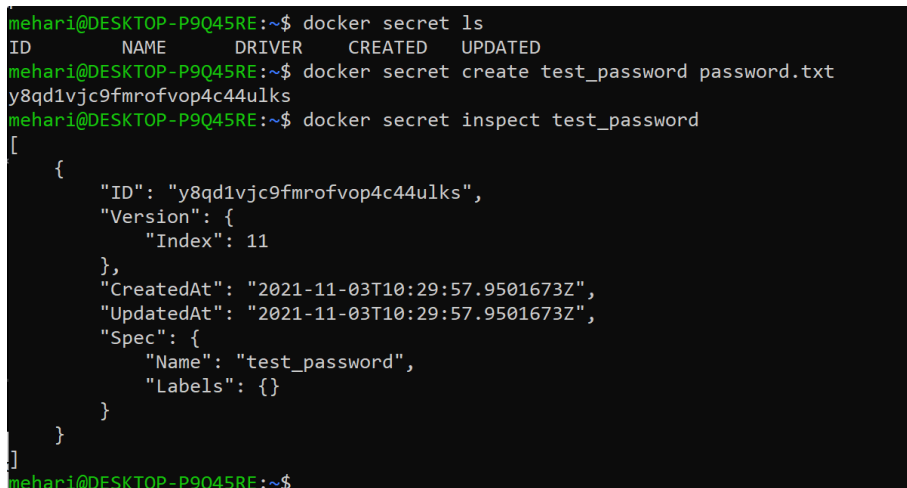
In figure 3, we check that we don't have a container running and creating a file name password and has a "password" as a content. We are going to use this to run a service.



```
mehari@DESKTOP-P9Q45RE:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
mehari@DESKTOP-P9Q45RE:~$ vi password.txt
mehari@DESKTOP-P9Q45RE:~$ cat password.txt
password:
mehari@DESKTOP-P9Q45RE:~$ vi password.txt
mehari@DESKTOP-P9Q45RE:~$ cat password.txt
password
mehari@DESKTOP-P9Q45RE:~$ docker secret ls
ID             NAME          DRIVER      CREATED      UPDATED
mehari@DESKTOP-P9Q45RE:~$ docker secret create test_password password.txt
y8qd1vjc9fmrofvop4c44ulks
mehari@DESKTOP-P9Q45RE:~$
```

Figure 3: Creating a file name password and has a password as a content

In figure 4, we just create a secret for password.txt and uses inspect command to inspect the content.



```
mehari@DESKTOP-P9Q45RE:~$ docker secret ls
ID             NAME          DRIVER      CREATED      UPDATED
mehari@DESKTOP-P9Q45RE:~$ docker secret create test_password password.txt
y8qd1vjc9fmrofvop4c44ulks
mehari@DESKTOP-P9Q45RE:~$ docker secret inspect test_password
[
  {
    "ID": "y8qd1vjc9fmrofvop4c44ulks",
    "Version": {
      "Index": 11
    },
    "CreatedAt": "2021-11-03T10:29:57.9501673Z",
    "UpdatedAt": "2021-11-03T10:29:57.9501673Z",
    "Spec": {
      "Name": "test_password",
      "Labels": {}
    }
  }
]
mehari@DESKTOP-P9Q45RE:~$
```

Figure 4: Creating a secret from the file password.txt and name it test\_password

In figure 5, We created a service from docker hub image postgres with the secret that we have already created.

```

mehari@DESKTOP-P9Q45RE:~$ docker service ls
ID            NAME      MODE     REPLICAS  IMAGE      PORTS
mehari@DESKTOP-P9Q45RE:~$ docker service create --name postgresdb -p 5432:5432 -e POSTGRES_USER=postgres --secret test_password -e POSTGRES_PASSWORD_FILE=/run/secrets/test_password postgres:latest
azcu36syn73qkt2f81830kzs
overall progress: 1 out of 1 tasks
1/1: running
Verify: Service converged
mehari@DESKTOP-P9Q45RE:~$ docker service ls
ID            NAME      MODE     REPLICAS  IMAGE      PORTS
azcu36syn73q  postgresdb replicated 1/1      postgres:latest *:5432->5432/tcp
mehari@DESKTOP-P9Q45RE:~$ docker service ps postgresdb
ID            NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE      ERROR      PORTS
twwmfyhabjdu9 postgresdb.1 postgres:latest DESKTOP-P9Q45RE Running        Running 3 minutes ago
mehari@DESKTOP-P9Q45RE:~$

```

Figure 5: Running a service from postgres image

In figure 6, the container run in bush command line to see the secret file which is located under `/run/secrets/test_password`. Now, since the secret is here we can just delete the file "password.txt" from the node in this case the computer.

```

mehari@DESKTOP-P9Q45RE:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
cd2164d5f112  postgres:latest "docker-entrypoint.s..." 6 minutes ago  Up 6 minutes  5432/tcp
p             postgresdb.1.twwmfyhabjdu90f5yxhykh1oxw
mehari@DESKTOP-P9Q45RE:~$ docker exec -it postgresdb.1.twwmfyhabjdu90f5yxhykh1oxw bin/bash
root@cd2164d5f112:/# cd /run/secrets
root@cd2164d5f112:/run/secrets# ls
test_password
root@cd2164d5f112:/run/secrets# cat test_password
password
root@cd2164d5f112:/run/secrets#

```

Figure 6: Running the container in bin/bash to check the secret

- **Strengths:**

- Managing security data and transmit securely to those containers which have access to it. - In the case, we have different environment like development, test and production environment which usual have a different credentials(for security reasons) but have same secrets name. Then a container need only to know the secret to work in all the environment.

- **Weaknesses:**

- Docker secrets are only available in docker swarm. - we can't use to a standalone container.
- To use docker secrets we need to run our container as a service

## 4.2 HashiCorp Vault

HashiCorp Vault is a tool that is used for storing and accessing secrets. This tool even provides a unified interface without compromising great access control and logging a comprehensive audit log. It can give an API allow access to secrets based on policies, so any user of the API needs to verify themselves and only see the secrets they are allowed to view. Vault encrypts data using 256-bit AES with GCM. It can also accumulate data in various back-ends like Amazon DynamoDB, SQL, and many more. Vault supports logging to a local file for audit services, a Syslog server, or directly to a socket. The vault tool logs all the information about the client that has acted; their clients IP address, the action itself, and at what time it was performed.

Key strengths of this tool are that it encrypts and decrypts data without storing it, can generate secrets on-demand for some operations, allows replication across multiple data centers, has built-in protection for secret revocation, serves as a secret repository with access control details. There are some downsides with this tool is in its complexity where there can be late on be too many elements and features to set up and maintain even in the basic version. It also requires decisions around the backend type to be made up front and requires a team member to be responsible for maintaining and managing the access list and the Vault server itself, but overall is a flexible and great secret management solution.

Most of AWS secrets engine enabled inside at aws/



---

Here are some of the Demo illustration on how HashiCorp Vault integration works on Sensu Go's.  
prerequisite for environments in place

```
.Download and install, https://docs.docker.com/compose/install/
.Download and install the Sensu CLI (sensuctl); see: https://sensu.io/downloads,
.Download and install the Vault CLI (vault); see:
→ https://www.vaultproject.io/docs/install.
```

1. startup demo environment.

```
$ sudo docker-compose up -d
```

2. Configure sensuctl to connect to the local backend

```
$ sensuctl configure -n --url http://127.0.0.1:8080
--username sensu
--password sensu
--namespace default
$ sensuctl cluster health
=== Etcd Cluster ID: 3b0efc7b379f89be
      ID          Name      Error    Healthy
      8927110dc66458af  default      true
```

3. Configure the vault CLI to connect to the Vault server

```
$ export $(cat .env | grep VAULT_DEV_ROOT_TOKEN_ID)
$ echo "${VAULT_DEV_ROOT_TOKEN_ID}" > $HOME/.vault-token
$ export VAULT_ADDR=http://127.0.0.1:8200
```

4. Create some Vault secrets!

```
$ vault kv put secret/sensu/slack @vault/slack-secrets.json
$ vault kv get secret/sensu/slack
```

5. Configure the Sensu Vault Provider and Slack handler.

```
$ sensuctl create -f sensu/vault.yaml
$ sensuctl create -f sensu/slack.yaml
$ sensuctl handler list
$ sensuctl secret list.
```

6. Create an event that should alert in Slack First, create a Sensu API Key using sensuctl:

```
$ sensuctl user create webinar --password $(uuid)
Created
$ sensuctl user add-group webinar cluster-admins
Updated
$ sensuctl api-key grant webinar
Created: /api/core/v2/apikeys/ad3a7c47-6485-48af-a7e1-c9fc4a8a709e
```

Then, use the API key to create an event via the Sensu API

---

```
$ export SENSU_API_KEY=ad3a7c47-6485-48af-a7e1-c9fc4a8a709e
$ curl -i -XPOST -H "Authorization: Key $SENSU_API_KEY" -H "Content-Type:
→ application/json" -d
→ '{"entity":{"metadata":{"name":"i-424242"},"check":{"metadata":{"name":"my-app"},
→ "status":2,"output":"ERROR: failed to connect to
→ database."},"handlers":["slack","pagerduty"]}]}'
→ http://127.0.0.1:8080/api/core/v2/namespaces/default/events
```

7.Resolve the alert

```
$ curl -i -XPOST -H "Authorization: Key $SENSU_API_KEY" -H "Content-Type:
→ application/json" -d
→ '{"entity":{"metadata":{"name":"i-424242"},"check":{"metadata":{"name":"my-app"},
→ "status":0,"output":"Database connection
→ restored."},"handlers":["slack","pagerduty"]}]}'
→ http://127.0.0.1:8080/api/core/v2/namespaces/default/events
```

### 4.3 AWS Secrets Management

AWS Secrets Manager is a tool that is used for storing and managing credentials and other sensitive information. What is interesting about this tool is that it lets us quickly rotate AWS RDS credentials automatically on a schedule, manage, retrieve, and rotate database credentials, API keys, and other passwords. This tool can secure, analyze, and manage secrets needed to access the AWS Cloud capabilities, on third-party services. Secrets Manager allows us to manage access to secrets using permissions.

The key strengths of this tool are that it encrypts secrets at rest using encryption keys, decrypts the secret, and then it transmits securely over TLS, provides code samples that help to call Secret Manager APIs, and has client-side caching libraries to improve the availability and reduce the latency of using your secrets. Downsides using this tool is that it might get cumbersome retrieving secrets compared to HashiCorp Vault as an example. It does not provide much streamlined support for most popular CI/CD tools today.

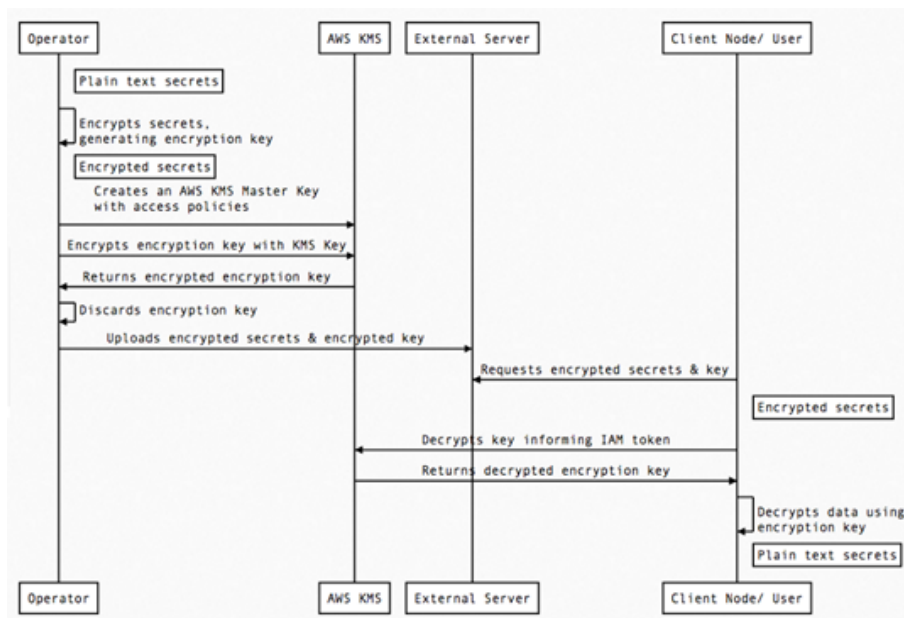


Figure 7: AWS sequence diagram

Source: Ferreira 2017

---

Before creating a secret, you need to have prerequisites already met. If you do not have an AWS account, you will have to register and make an account. After that you need permissions granted by the SecretManagerReadWrite AWS managed policy to create secrets. You need to prepare on the AWS console an admin user, then you can attach or remove an AWS managed policy to users in your account, but you can not modify or delete the whole policy. More info here. After successfully creating an admin user and downloading the credentials as a .csv file you can go to your preferred terminal and configure your AWS default profile there.

Next step is to create a secret in you AWS secrets manager. There are many types of secrets you can store; credentials for RDS databases, documentDB databases, redshift cluster, and others.

The screenshot shows the AWS console interface for creating a secret. The first section, 'Select secret type', has three radio button options: 'Credentials for RDS database', 'Credentials for other database', and 'Other type of secrets (e.g. API key)'. The third option is selected. The second section, 'Specify the key/value pairs to be stored for this secret', has two tabs: 'Secret key/value' (active) and 'Plaintext'. Below the tabs, there are two rows of input fields. The first row has 'username' and 'admin', with a 'Remove' button. The second row has 'password' and 'admin123', also with a 'Remove' button. At the bottom, there is a '+ Add row' link.

Figure 8: AWS Select secret type

For simplicity sake you can choose "Other type of secrets" and type in the secret key and value. On the next page you need to provide a new secret name. Next you may skip the rotation section, and then you will find some code samples in different programming languages, which is useful for to call or retrieve the secret key in your application or API for example. Lastly you just press on store the key and then you have a secret key available and stored in your AWS secret manager.

## 4.4 Kubernetes Secret Management

To begin with "*Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF)*" Authors 2021a. Kubernetes secret management is a secure object in which it stores different sensitive data such as passwords, authentication token, ssh keys(private key and public key), Private encryption keys, One-time passwords for individual devices and other secrets that we don't want to be seen or implemented in our actual code in an encrypted form. Kubernetes have the ability to store this secret information in a pod and manage access of the secrets by the different container in a pod. Secrets in Kubernetes can be either built-in or customized. It automatically creates built-in secrets for service accounts by attaching them to containers at the time of building a container. The customized secrets can be done as a root user in the system. It offers several options to use a secret. It can be mounted as a content of a file, environment variable or in configuration files. Kubernetes uses the kubectl command to create different types of secrets.

---

```
mehari@DESKTOP-P9Q45RE:/etc/kubernetes/manifests$ cd
mehari@DESKTOP-P9Q45RE:~$ kubectl create secret --help
Create a secret using specified subcommand.

Available Commands:
  docker-registry Create a secret for use with a Docker registry
  generic          Create a secret from a local file, directory, or literal value
  tls             Create a TLS secret
```

Figure 9: Kubernetes command for creating secrets screenshot

**Docker-registry:** Creates a dockercfg Secret for use with a Docker registry. Used to authenticate against Docker registries.

**Generic:** Create a Secret from a local file, directory, or literal value.

**TLS(Transport Layer Security):** Create a TLS secret from the given public/private key pair. First we have to create public/private key pair and the public key certificate must be .PEM encoded and match the given private key.

- **Strengths:**

- If we see from point of an administrator, it simplifies the need to update credentials because all the containers that use the specified secret get updates as soon as we update the pod.
- Since secrets are created independently of the pods, it is less likely to be exposed to other pods at the time of creating a new pod or configuring another pod.

- **Weaknesses:**

- It is only used in a clusters
- It is available to all the users that have access to the cluster
- If we load a secret in an environmental variable we must make sure that the logging statement do not contain the variable. If this is so then this will expose the secret in a plain text in the log.

## 4.5 Git-crypt

Git-crypt is a tool used to encrypt and decrypt files transparently in git. It uses GPG keys in the background. The files are encrypted when committed and decrypted when pulled. This tool lets you have a mix of private and public files in the same git repository, so users without the secret key can also commit and checkout the repository.

It uses AES-256 in CTR mode with a synthetic IV (a type of block cipher) delivered from the SHA-1 HMAC (a cryptographic hash function) of the file for encrypting files. This provides a mode of security where whether the encryption is deterministic, it does not leak information, just if two files are the same or not.

One of the limitations of git-crypt is that is not the best tool for encrypting all the files in a repository as the git filters where not designed for encryption.

Git was not designed with encryption in mind, so as said before, it relies on this filters and it is not the best tool to encrypting a whole repository. It does not encrypt file names, commit messages, gitlinks or some other metadata neither hide when a file changes, the length of that file or if they are identical or not, as previously mentioned.

It also does not let revoke access to a repository which was previously granted. If a key was given a key and then revoked the access, they will still could have access to a previous repository. Finally, the files encrypted with git-crypt are not compressible. The smallest change on a file requires to store the whole file.

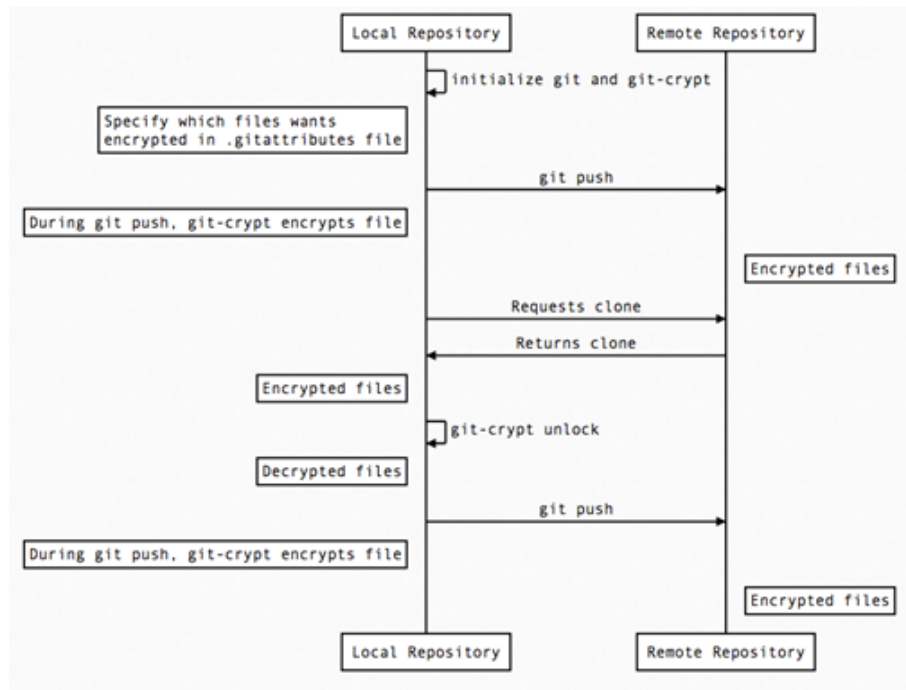


Figure 10: Git-crypt sequence diagram

Source: Ferreira 2017

## Using git-crypt

Configure a repository to use git-crypt:

```
$ cd repo
$ git init
$ git-crypt init
```

After that you create a `.gitattributes` file:

```
# define which file we want to encrypt
FILE_TO_ENCRYPT filter=git-crypt diff=git-crypt
# create new definitions for each file and add some patterns
*.key filter=git-crypt diff=git-crypt
secretdir/** filter=git-crypt diff=git-crypt
# for not encrypt itself
.gitattributes !filter !diff`
```

For sharing the repository with others using GPG:

```
$ git-crypt add-gpg-user USER_ID
```

Finally, you can unlock a repository with the command

```
$ git-crypt unlock
```

## 4.6 Chef Vault

Chef allows to describe what an instance should look like (which users can register, which applications installed, which ports open,...). All of this is stored in "recipes", contained in "cookbooks".

Chef vault does not require the use of one shared key by all the users and pieces of infrastructure. The reason of this is that, it has a set of nodes through a Chef Server and every node managed through that server has an associated client object on the Chef Server, it is aware of and managing the node. When a node is managed through a Chef Server it uses a RSA keypair, the private half retained by the node and the public one by the Chef Server.

Every user has also a RSA keypair, and this is how they can authenticate/authorize a workstation, that will have the private half of the key while the public one is still in the Chef Server. After that, when someone creates a Vault, it will encrypt the Vault item with a random secret string and they will have to specify the users and nodes who will need access to the vault. The Chef Vault will encrypt a copy of the random secret for each node.

Secrets can be required to complete some tasks during the provisioning process, like installing a database server or create new users. Then the Chef Client Node will follow this workflow:

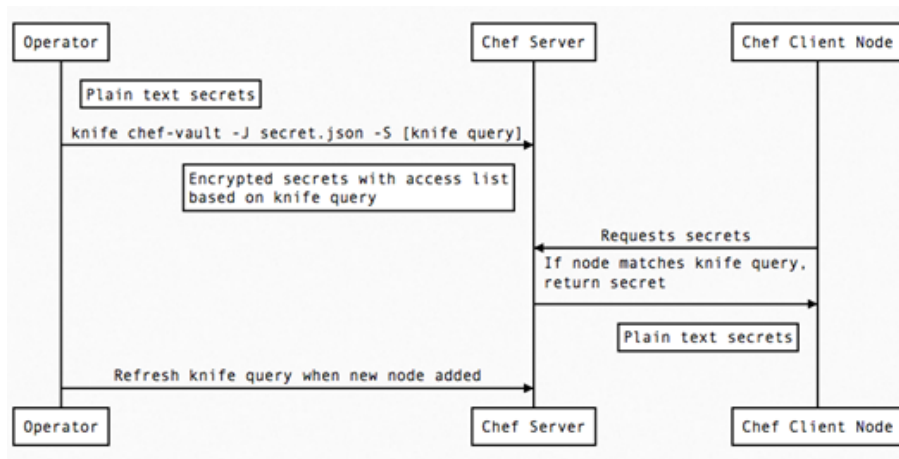


Figure 11: Chef Vault sequence diagram

Source: Ferreira 2017

When the operator pushes secrets to the Chef Server using the knife tool, they specify a query in the parameters that lists which nodes will have access to a set of secrets.

Some weaknesses of Chef Vault is that the access to the lists of secrets is static. This means that when the secrets are uploaded (associated to a query), only the nodes matching the query at that moment will be added to the access list. If new nodes are added after that, they will not have access to this secret. For resolving this problem, the query has to be executed manually each time a node is added.

We are going to create a vault and after that, we will use the Chef Recipe.

### Creating a Vault

After you have installed chef vault with the command

```
$ gem install chef-vault
```

You create a JSON file "database.json" with the password

```
{ "db_password": "password" }
```

Now we define who will be able to decrypt this vault.

```
$ knife vault create VAULT_NAME ITEM_IN_VAULT_TO_ENCRYPT -A USER1, USER2 -M
→ client -S 'name:NODE*' -J ./database.json
```

---

We have the USER1 and USER2 that will be able to decrypt the items NODE1 and NODE2. As we are using chef server, we use "-M client". If we were using Chef Solo it would be "-M solo". As the content is in the file "database.json", we include it.

After this, you will have a vault named VAULT\_NAME and two items: ITEM\_IN\_VAULT\_TO\_ENCRYPT and ITEM\_IN\_VAULT\_TO\_ENCRYPT\_keys

The VAULT\_NAME contains the encrypted content. If we see the item as a not admin user it appears everything encrypted:

```
$ knife vault show VAULT_NAME ITEM_IN_VAULT_TO_ENCRYPT
db_password:
  cipher:      aes-256-cbc
  encrypted_data:
    ↪ dsiBtaHX8SbTs42yKuYBrddDX5pu8bQFJrS2hoE7doy4fR8roF5NzpVHyoaG24yb3
  iv:          +0sPN7zFjH6qtMT7k5JIYw==
  version:     1
id:           VAULT_NAME
```

If we see it with an admin user we can see it decrypted:

```
$ knife vault show VAULT_NAME ITEM_IN_VAULT_TO_ENCRYPT
db_password: password
id:          VAULT_NAME
```

Now, if we take a look at the ITEM\_IN\_VAULT\_TO\_ENCRYPT\_keys we get the following:

```
$ knife data bag show VAULT_NAME ITEM_IN_VAULT_TO_ENCRYPT_keys
admins:
  USER1
  USER2
clients:
  NODE1
  NODE2
id:      VAULT_NAME_test_keys
USER1: KEY
USER2: KEY
NODE1: KEY
NODE2: KEY
```

This has some 4 keys, when some user or node tries to decrypt the item, the private key will decrypt the shared secret and the item.

## 5 Best technology

After researching and conclude with six different technologies for handling secrets, we have come to an end and decide that the best technology is HashiCorp Vault, closely followed by AWS Secrets Management.

The reasons are that HashiCorp provides a unified interface without compromising great access control and logging a comprehensive audit log. It is no static and on-premise, it scales and shrinks depending on servers and applications. Everything is almost in the cloud, so it has various vendors and service providers, and it can be replicated across those data centres that are using it.

HashiCorp will also give an API that any user will need to verify and they will only be able to see the secrets they are authorized to see. As it is Vault, it will store information about the client and

---

their IP address, the action they performed and at what time it was done. It also generates some secrets for some certain operations.

It also encrypts and decrypts data without having to store it and has a built-in protection for revocation of secrets.

## 6 Reflection

The secret management approach and techniques are important in an industry. Industries are using different approaches and technologies to manage their credentials. The secret management approaches have discussed so far are encrypt secrets, the main focus here is to encrypt all secret in repositories with the help of tools like Git-Crypt.

However, it is not guarantee to put decryption key in same repository to minimize vulnerability risks. Secondly, Secret-less authorization:- this approaches permit authorized users to get a service with out using a secret. cloud platforms such as virtual machine, container instance, and AWS EC2 instances assign roles using Identity and access management (IAM), and then permit the application on that instances with predefined roles to access resources.

On the other hand, OTP, magic-links, PIN, generated token code, or bio-metric are some of the password less authentication methods which improves users login experience. The other approaches mentioned are Run time secret injection which gives opportunity not put secret in resource code. It is possible to inject secret in code at run-time, only when it is must and conditional. The last approaches mentioned in this paper are disposable secrets, mainly focused on passwords that would be generated automatically.

We have also seen some technologies that can be useful when trying to approach secrets managements, concluding that the best technology is HashiCorp Vault for the reasons given in part 5.



---

## Bibliography

- Authors, The Kubernetes (2021a). *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/home/> (visited on 3rd Nov. 2021).
- (2021b). *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/home/> (visited on 3rd Nov. 2021).
- Ayer, Andrew (2021). *git-crypt - transparent file encryption in git*. URL: <https://www.agwa.name/projects/git-crypt/> (visited on 22nd Oct. 2021).
- BeyondTrust (2021a). *DevOps Security*. URL: <https://www.beyondtrust.com/resources/glossary/devops-security> (visited on 22nd Oct. 2021).
- (2021b). *Secrets Management*. URL: <https://www.beyondtrust.com/resources/glossary/secrets-management> (visited on 22nd Oct. 2021).
- Consulting, AVM (2021). *Secret Management in Kubernetes*. URL: <https://medium.com/avmconsulting-blog/secrets-management-in-kubernetes-378cbf8171d0> (visited on 2nd Nov. 2021).
- Ferreira, Stenio (2017). *Secret Management Architectures: Finding the balance between security and complexity*. URL: <https://medium.com/slalom-technology/secret-management-architectures-finding-the-balance-between-security-and-complexity-9e56f2078e54> (visited on 22nd Oct. 2021).
- Inc., Docker (2021). *Introducing Docker Secrets Management*. URL: <https://docs.docker.com/engine/swarm/secrets/> (visited on 30th Oct. 2021).
- Li, Ying (2017). *Introducing Docker Secrets Management*. URL: <https://www.docker.com/blog/docker-secrets-management/> (visited on 3rd Nov. 2021).
- Morris, Kief (2020). *Infrastructure as Code, Second Edition. Dynamic Systems for the Cloud Age*. O'Reilly.
- Platform9 (2020). *Introduction to Kubernetes Management*. URL: <https://platform9.com/blog/kubernetes-secrets-management/> (visited on 3rd Nov. 2021).
- Walsh, John (2020). *Best Practices for Kubernetes Secrets Management*. URL: <https://www.conjur.org/blog/kubernetes-security-best-practices-for-kubernetes-secrets-management/> (visited on 2nd Nov. 2021).