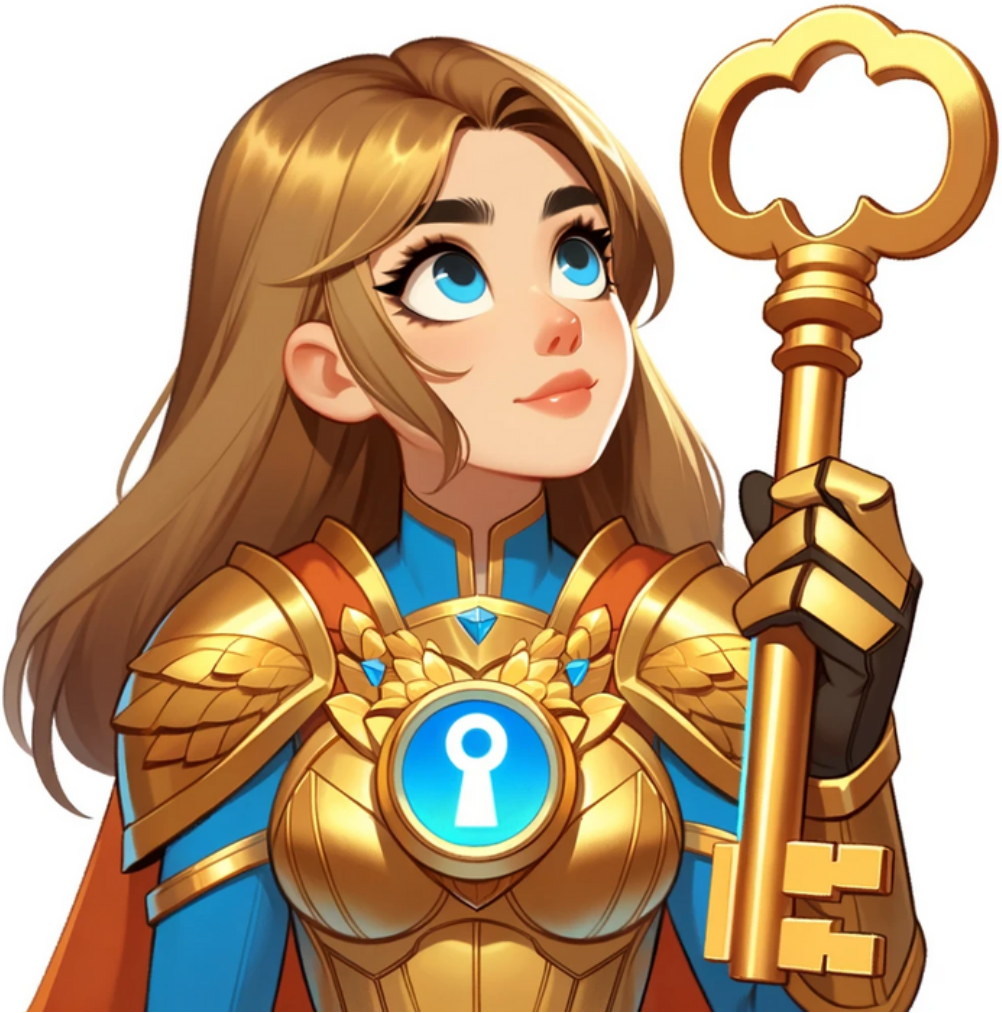
Antony Kervazo-Canut

Vault for Teenagers

SUITABLE FOR ADULTS



Introduction

What is Vault ?

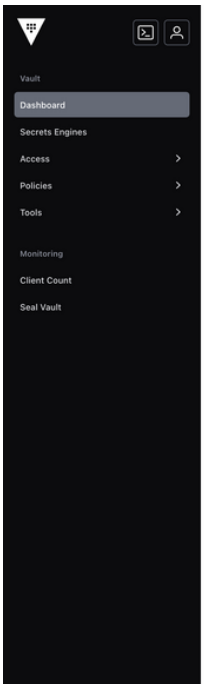
Vault is a secret management tool created by HashiCorp. It enables the secure storage and control access to tokens, passwords, certificates, API keys, and other secrets.

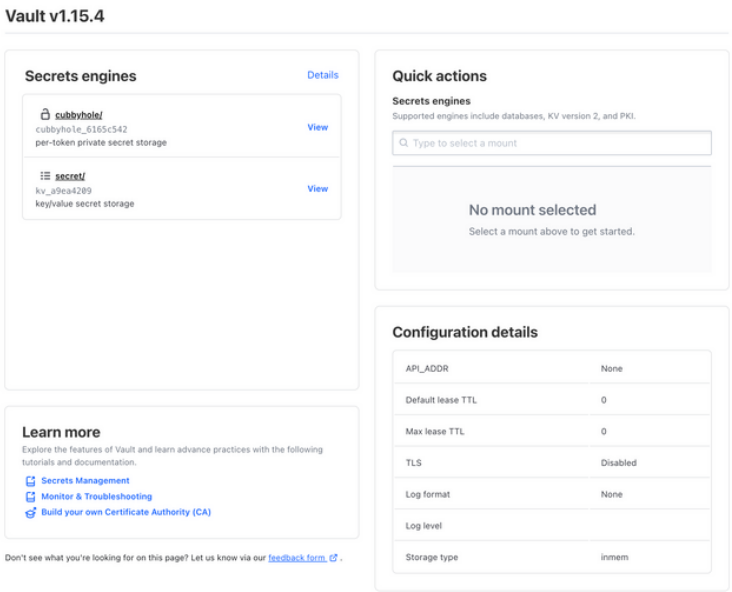
Why use Vault ?

Vault is designed to help manage secrets securely in a modern computing environment, where the proliferation of sensitive data and remote access require robust security.

Overview of the CLI and GUI

Vault offers a command-line interface (CLI) and a graphical user interface (GUI). The CLI is powerful for automation and scripting, while the GUI provides an intuitive and visual user experience.





Installation of Vault

```
# Downloading Vault
# Standard download link:
https://www.vaultproject.io/downloads
# The page offers different binaries depending on the
operating system.

# Installation on a Unix/Linux system
# Unzip the downloaded file and move it to a directory in the
PATH
tar -xzf vault_[version]_[system].zip
sudo mv vault /usr/local/bin/

# Starting Vault in development mode
# This command starts a Vault server in development mode with
minimal configuration
vault server -dev

# You can now test Vault locally
```

In development mode, the user interface is accessible at the address http://127.0.0.1:8200/ui. Use the token displayed in the terminal to log in.

The login token is shown on the console after Vault starts.

Vault Configuration

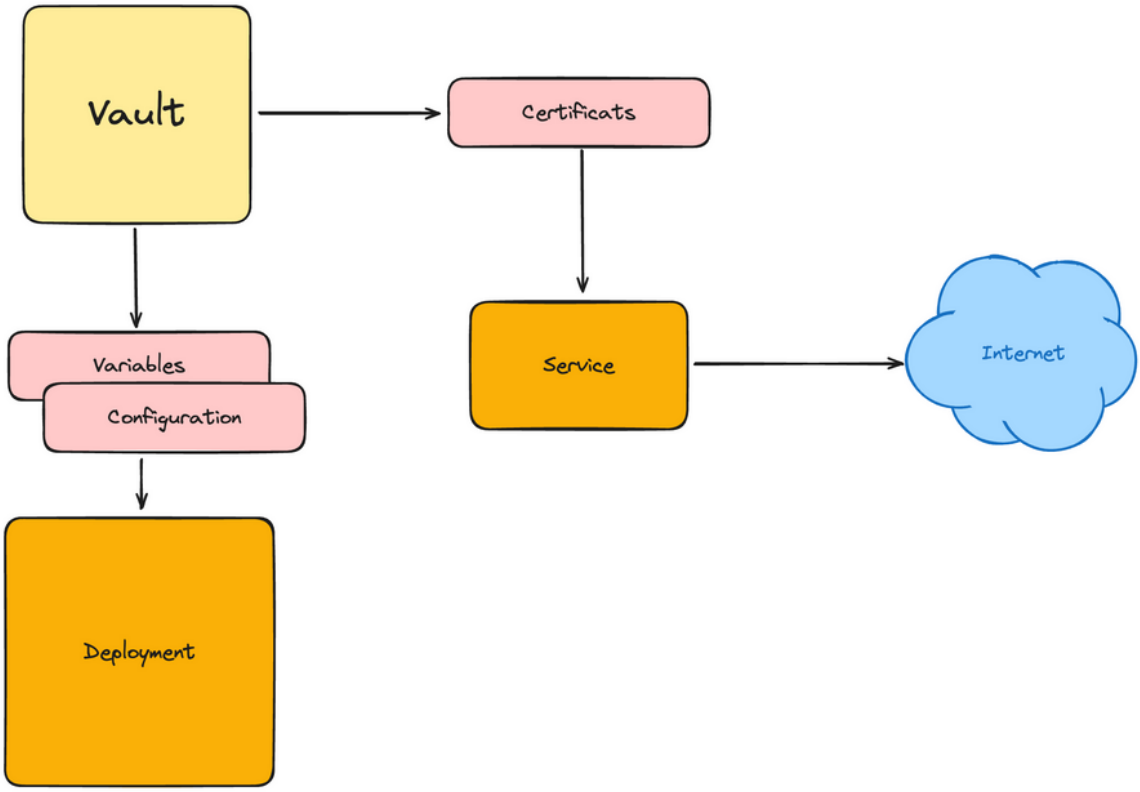
```
# configuration file (config.hcl)
storage "file" {
  path = "/path/to/data"
}

listener "tcp" {
  address     = "127.0.0.1:8200"
  tls_disable = 1
}

ui = true
```

```
# Starting Vault with a configuration file
# Use this command to start the Vault server
vault server -config=[path to configuration file]

# Stopping Vault
# To stop the Vault server, use this command
pkill vault
```

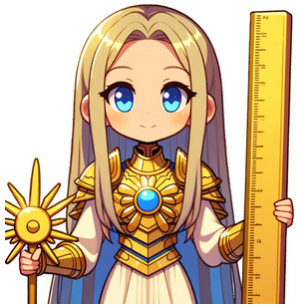
Vault et Kubernetes

The purpose of this document is to introduce Vault by HashiCorp. But also, and more importantly, to envision it within a Kubernetes environment.

In this case, Vault can then provide secrets to Kubernetes, including configuration as well as certificates, etc.



Policies



Policies in Vault are rules written in HCL (HashiCorp Configuration Language) that define the permissions for secrets and authentications.

```
path "secret/data/*" {
  capabilities = ["create", "read", "update", "delete",
"list"]
}
```

```
# Write the policy in a file (e.g., my-policy.hcl) and use
this command to add it to Vault
vault policy write [policy-name] [path-to-policy-file]

# This command assigns a policy to an existing token
vault token create -policy=[policy-name]

# Assign a policy to a user or group (e.g., LDAP)
# Adapt this command according to the authentication method
used
vault write auth/[method]/groups/[group-name] policies=
[policy-name]
```

It is entirely possible to write the policies directly using the GUI.

Authentication

Vault allows the creation of service accounts, which are identifiers for applications or automated services.

It is possible to use many login methods (Github, AWS, LDAP, etc).

```
# Create a role for the service account
vault write auth/token/roles/[role-name] allowed_policies="
[policy-list]"

# Generate a token for the service account
vault token create -role=[role-name]

# Log into Vault with a service token
vault login [service-token]

###

# Enable GitHub authentication
vault auth enable github

# Configure GitHub authentication with an organization
vault write auth/github/config organization=[organization-
name]

# Log into Vault with a GitHub token
vault login -method=github token=[github-token]
```

Authentication Configuration

```
# Enable an authentication method
# This command activates a specific authentication method in
Vault
vault auth enable [auth-method]

# Example to enable username/password authentication
vault auth enable userpass

###

# Configure an authentication method
# This command configures settings specific to the chosen
authentication method
vault write auth/[auth-method]/config [parameters]

# Example to configure username/password authentication
vault write auth/userpass/config password_policy="my-policy"

# Create a user with Userpass
vault write auth/userpass/users/[username] password=
[password] policies="[policy-list]"

# Example of creating a user
vault write auth/userpass/users/john password=examplepassword
policies="default"
```

The Secrets

Secrets in Vault are sensitive data such as passwords, API keys, and certificates. Vault can store, generate, and control access to these secrets. Here is an example with secrets in the form of key-value pairs.

```
# Create a secret
vault kv put [path] [key]=[value]

# Example of creating a secret
vault kv put secret/hello password=world

# Read a secret
vault kv get [path]

# Example of reading a secret
vault kv get secret/hello

# Update a secret
vault kv put [path] [key]=[new-value]

# Example of updating a secret
vault kv put secret/hello password=newpassword

# Delete a secret
vault kv delete [path]

# Example of deleting a secret
vault kv delete secret/hello
```

Vault & Kubernetes

Vault Agent Injector is a tool that automates the injection of Vault secrets into Kubernetes pods. You must first configure it in your Kubernetes cluster.

```
# Install Vault Agent Injector in your Kubernetes cluster
kubectl apply -f
https://raw.githubusercontent.com/hashicorp/vault-
k8s/master/deploy/injector.yaml
```

Create a standard ConfigMap, but with references to Vault secrets.

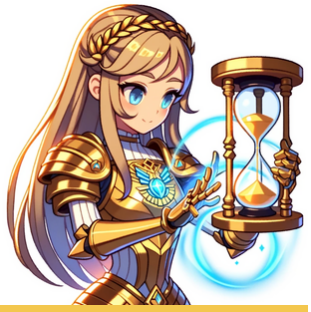
```
# Example of ConfigMap with references to Vault secrets
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  SECRET_PATH: "vault:secret/data/hello#password"
path "secret/data/*" {
  capabilities = ["create", "read", "update", "delete",
"list"]
}
```

Vault & Github Action

Before retrieving secrets, ensure that Vault is configured to allow access via GitHub Actions. This may involve creating a role with the appropriate permissions.

```yaml
# Example GitHub Actions workflow to retrieve a secret from Vault
name: My Simple Pipeline
on: [push]
jobs:
  show-secret:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Retrieve and display the secret from Vault
      env:
        VAULT_ADDR: 'https://[vault-address]' # Replace with your Vault server address
        VAULT_TOKEN: ${{ secrets.VAULT_TOKEN }} # Vault token is stored as a GitHub secret
      run: |
        secret=$(curl -s --header "X-Vault-Token: $VAULT_TOKEN" $VAULT_ADDR/v1/secret/data/hello | jq -r .data.data.password)
        echo "The secret is: $secret"
```

Secrets Rotation



```
# Store database login credentials in Vault
vault kv put secret/db-creds username="root"
password="rootpassword"

# Create a role for rotating secrets
vault write database/roles/my-role \
    db_name=mydb \
    creation_statements="CREATE USER '{{name}}'@'%'
IDENTIFIED BY '{{password}}'; GRANT SELECT ON *.* TO
'{{name}}'@'%';" \
    default_ttl="1h" \
    max_ttl="24h"

# Configure the database connection using Vault's secrets
vault write database/config/mydb \
    plugin_name=mysql-database-plugin \
    connection_url="{{with secret \"secret/db-creds\"}}
{{.Data.data.username}}:{{.Data.data.password}}
{{end}}@tcp(127.0.0.1:3306)/" \
    allowed_roles="my-role"

# Request a new database credential
response=$(vault read -format=json database/creds/my-role)

# Extract the lease ID and credentials
lease_id=$(echo $response | jq -r '.lease_id')
username=$(echo $response | jq -r '.data.username')
password=$(echo $response | jq -r '.data.password')

# Renew the credentials' lease
vault lease renew $lease_id
```

Audit Logging

Audit logging in Vault allows you to keep a detailed record of all interactions with Vault, including requests and responses. This feature is essential for security monitoring and analysis.

```
# Enable audit logging with a file
vault audit enable file file_path=/var/log/vault_audit.log

# Check the status of audit logging
vault audit list
```

Backup and Restoration

Vault stores all its data (including secrets and configurations) in its storage backend. To back up this data, you must back up the storage backend itself.

```
# Example command to backup Vault's storage backend
# This command depends on the type of storage you are using
(e.g., Consul, S3, local file, etc.)
# Here is an example for a file type storage
tar -czvf vault-backup.tar.gz /path/to/vault/data

# Example command to restore Vault's storage backend
# Make sure the Vault service is stopped before restoring the
data
tar -xzvf vault-backup.tar.gz -C /path/to/vault/data
```

Revocation

The revocation of secrets and tokens. This feature is essential for effectively managing the end-of-life of secrets and access.

```
# Revoke a specific token
vault token revoke [token_id]

# Revoke a secret lease
vault lease revoke [lease_id]

# Revoke all tokens or leases associated with a role
vault token revoke -mode=path -path auth/userpass/users/john
```

Transit Engine

The Transit Engine is for encryption as a service. This feature allows Vault to encrypt and decrypt data without storing it, thus providing a secure way to manage encryption.

```
# Enable Transit Engine
vault secrets enable transit

# Create an encryption key
vault write -f transit/keys/my-key

# Encrypt data
cipher_text=$(vault write -format=json transit/encrypt/my-key
plaintext=$(base64 <<< "Hello World") | jq -r
'.data.ciphertext')
echo "Cipher Text: $cipher_text"

# Decrypt data
plain_text=$(vault write -format=json transit/decrypt/my-key
ciphertext=$cipher_text | jq -r '.data.plaintext' | base64 --
decode)
echo "Plain Text: $plain_text"
```

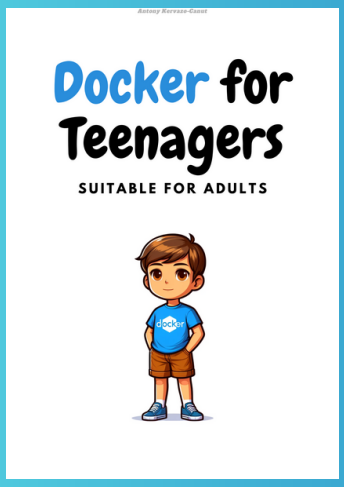
Seal / Unseal

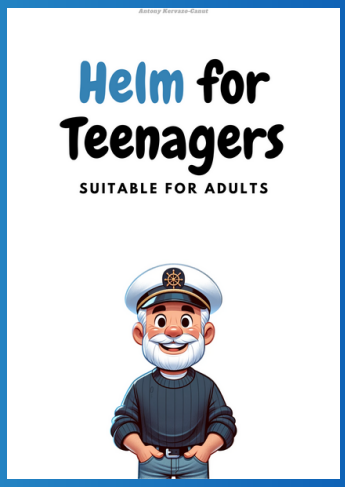
During the initialization of Vault, recovery keys are generated. These keys can be used to unlock Vault in case of an emergency, such as the loss of access by administrators.

Locking Vault without shutting it down is an important operation for maintenance or security management. This action temporarily prevents access to the secrets stored in Vault, without stopping the service itself.

```
# Locking Vault prevents access to keys
vault operator seal

# Use a recovery key to unlock Vault
vault operator unseal [Recovery_Key]

# Renew recovery keys
vault operator rekey -init
# Follow the instructions to complete the renewal process

# Create an emergency token
vault token create -ttl="1h" -policy="root"
```

In the same collection





ANTONYCANUT

ANTONY KERVAZO-CANUT