HashiCorp Vault

Introduction

Name

Total Experience

Background – Development / Infrastructure / Database / Network

Experience on monitoring tools

Your Expectations from this training

Module 1:
What is Vault

What are Secrets

A secret is any sensitive information that must be protected and tightly controlled to prevent unauthorized access. It can be credentials, encryption keys, tokens, or any data that should not be publicly accessible.

- Passwords
- PKI Certificates
- SSH Keys
- Key Value
- API Key
- Encryption Keys (Symmetric & Asymmetric)
- Time-based one-time password (TOTP)
- TLS Certs

Secrets

Purpose of Secrets:

- Secrets control access to systems, applications, and sensitive data.
- Managing and protecting secrets is crucial for maintaining security and preventing breaches.

Challenges with Secrets:

- **Sprawl**: Secrets are often scattered across systems, applications, and environments.
- **Plain Text Storage**: Many secrets are stored in unencrypted formats, increasing vulnerability to attacks.
- **Access Control**: It can be difficult to ensure that only authorized entities have access to certain secrets.

What is Vault?

It is an open source tool by HashiCorp developed in GO language and introduced in 2018

HashiCorp Vault:

- Identity-based secrets and encryption management system.

Purpose:

- Securely store, manage, and protect sensitive data such as tokens, API keys, passwords, encryption keys, and certificates.

Access Control:

- Vault uses authentication and authorization methods to restrict access to secrets.

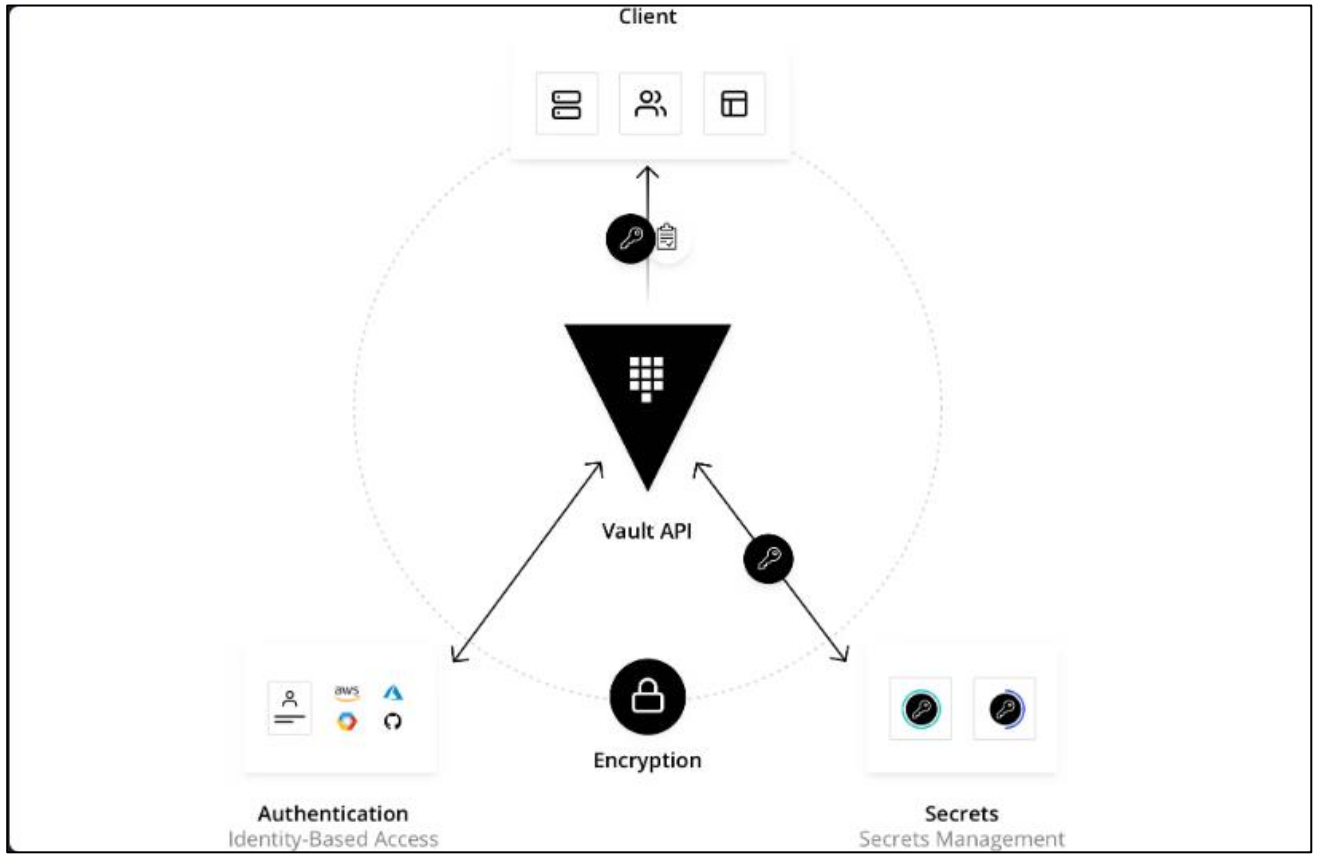
Audit Logs:

- Provides detailed audit logs for all interactions with secrets.

UI, CLI, or HTTP API:

- Multiple interfaces to interact with Vault based on user preference

What is Vault?



Why Use Vault?

Credential Sprawl:

- Enterprises often have credentials (passwords, API keys, etc.) scattered across various systems in plain text—within app source code, config files, etc. This increases security risks and makes it hard to control access.

Centralized Secrets Management:

- Vault addresses this challenge by centralizing the management of credentials, ensuring they are securely stored in one location, reducing the risk of unauthorized access.

Authentication & Authorization:

- Vault enforces strict authentication and authorization policies, ensuring that only verified users, apps, and systems can access sensitive resources.

Audit Trail:

- Vault provides detailed audit logs, recording all actions taken by clients to ensure full visibility and traceability of secret access.

Mitigating Security Risks:

- By removing plain text credentials and limiting access, Vault significantly reduces the threat of both internal and external malicious attacks.

Key Features

1. Secure Secret Storage

- Arbitrary key/value secrets can be stored securely.
- Encryption: Vault encrypts secrets before storing them.
- Persistent Storage: Secrets can be stored in disk, Consul, etc.
- Even if raw storage is accessed, encrypted secrets remain safe.

2. Dynamic Secrets

- On-Demand Secret Generation: Vault can dynamically generate secrets for services like AWS or SQL databases.
- Example: For S3 access, Vault generates an AWS keypair with valid permissions.
- Automatic Revocation: Secrets are revoked after the lease period ends, reducing risk.

3. Data Encryption

- Encryption as a Service: Vault can encrypt and decrypt data without storing it.
- Custom Encryption: Security teams define encryption methods; developers don’t need to design their own.
- Flexible Storage: Encrypted data can be stored in various locations like databases.

Key Features

4. Leasing and Renewal

- Lease Management: Every secret has a lease duration.
- Automatic Revocation: Secrets are revoked when the lease expires.
- Renewal: Clients can renew leases via the renew APIs to extend access.

5. Revocation

- Built-In Secret Revocation: Vault can revoke single secrets or groups of secrets.
- Granular Control: Revoke all secrets accessed by a user or a particular secret type.
- Security Response: Revocation helps in key rolling and system lockdown during security incidents.

Use Cases

General Secret Storage

- Store it and read in plain text

Employee Credential Storage:

- Manage employee credentials
- Create, rollout, manage, revoke etc

API Keys & Tokens:

- Manage access to external service credentials.
- AWS,AZURE,GCP – Access and IAM

Service Communication:

- Secure credentials for service-oriented architecture (SOA).

Platform-Specific Security:

- Unify secrets management across different platforms.

Use Cases

Data Encryption

- Encrypt/decrypt the data
- Worry free encryption

Automated PKI Infrastructure

- Creating, rotating and managing certificates

Data Encryption and Tokenization:

- Data across clouds, applications and systems
- Encrypt and tokenize

Database Credential Rotation

- Each DB credentials for apps, services and users
- Automated shared DB credential rotation

Use Cases

Dynamic Secrets:

- Organizations: avoid long living credentials
- Generate time based access credentials

Key management:

- Centrally manage and automate encryption keys
- Across different clouds and environments

Kubernetes Secrets

- No sharing of creds/token to pods
- Vault to securely inject secrets to stack

How Vault Works:

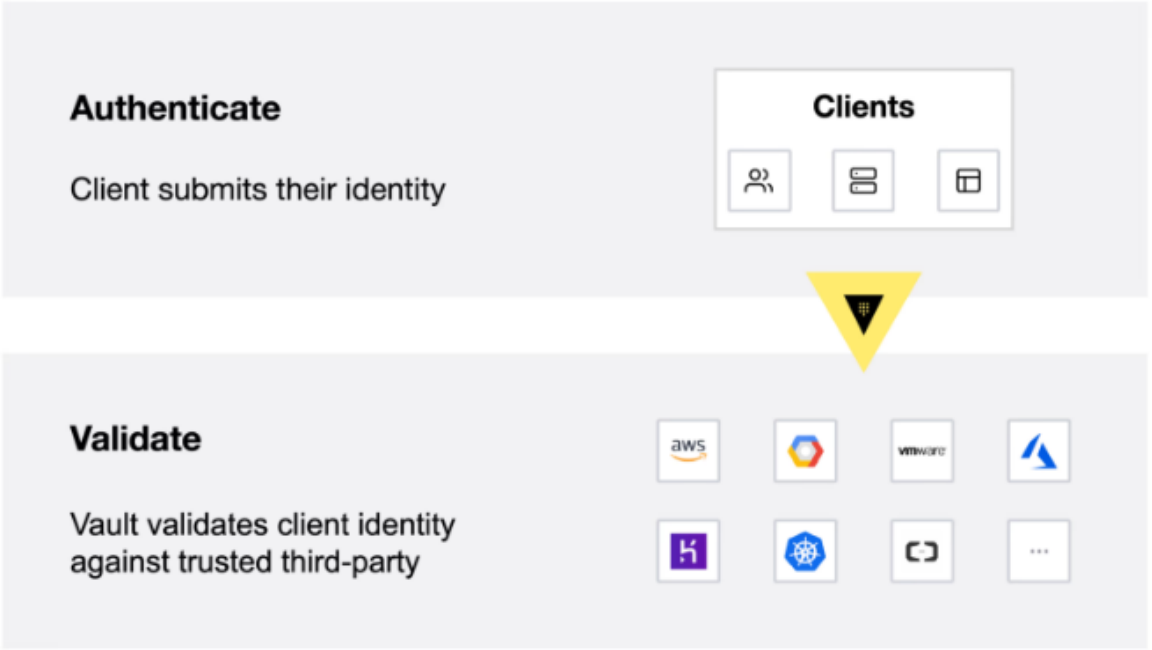
Core Workflow (4 Stages):

Authenticate

- Clients provide authentication credentials to Vault.
- Vault verifies the client's identity using an auth method (e.g., GitHub, LDAP, AppRole).
- A token is generated after authentication, linked to a security policy that controls access.

Validation

- Vault validates the client's identity against trusted third-party sources like GitHub, LDAP, or custom AppRoles.
- This ensures that only authorized entities can authenticate with Vault.



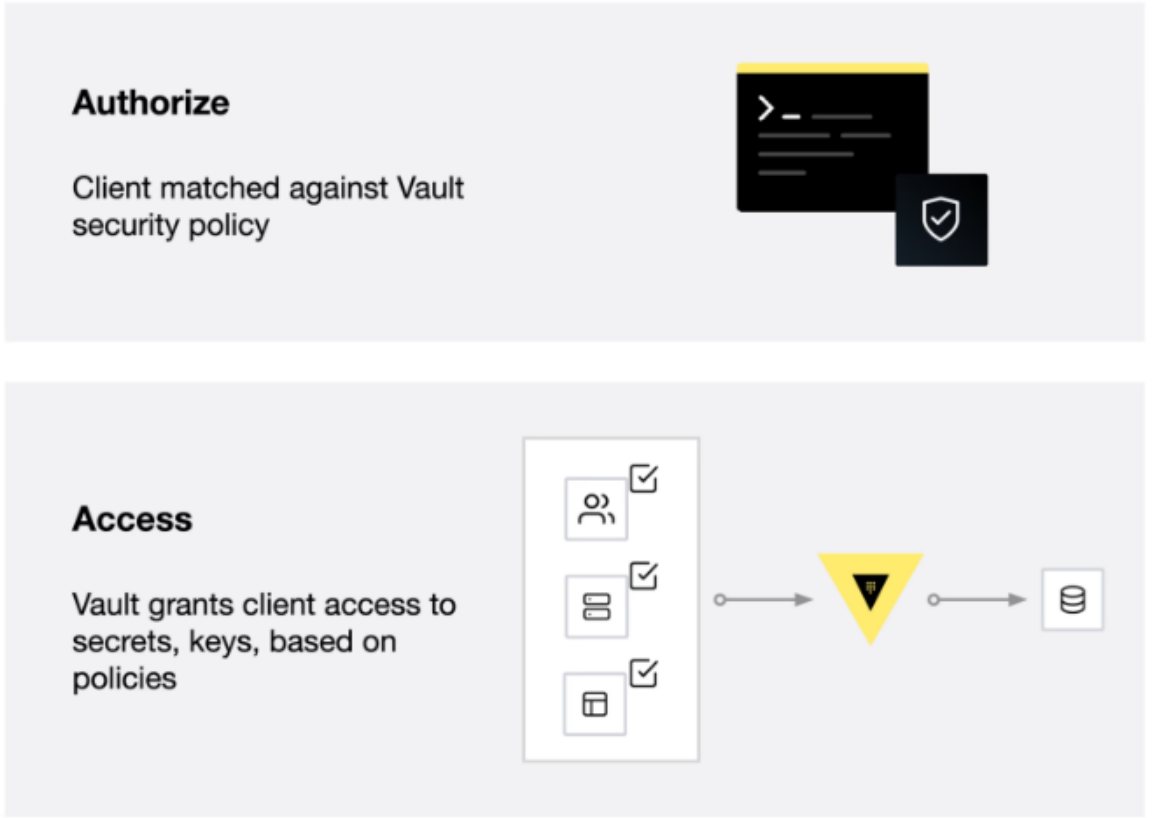
How Vault Works:

Authorize

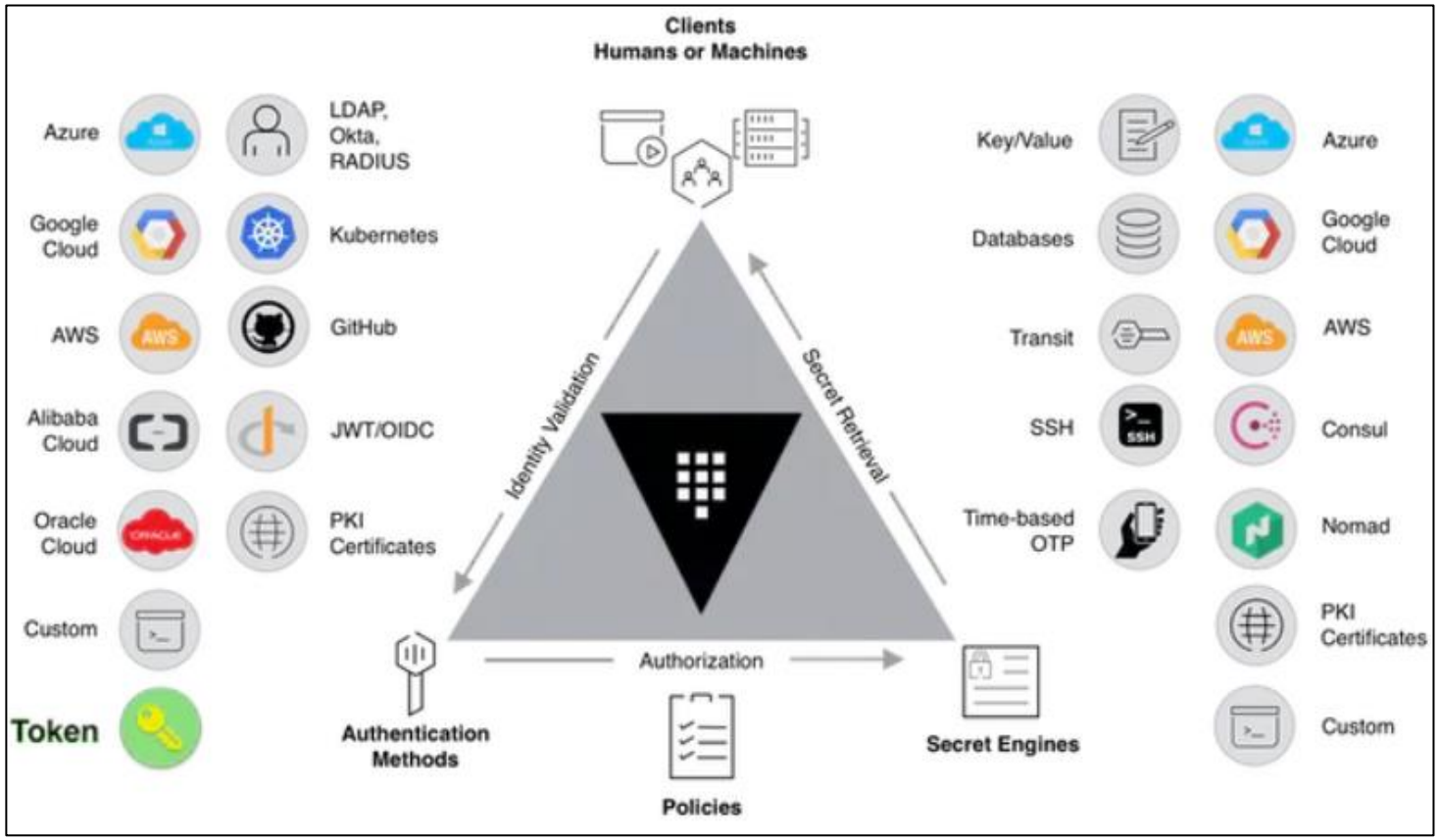
- Vault applies security policies to the authenticated client.
- Policies define which API endpoints, secrets, and operations the client can access.
- The policy is path-based, constraining actions to specific resources or operations.

Access

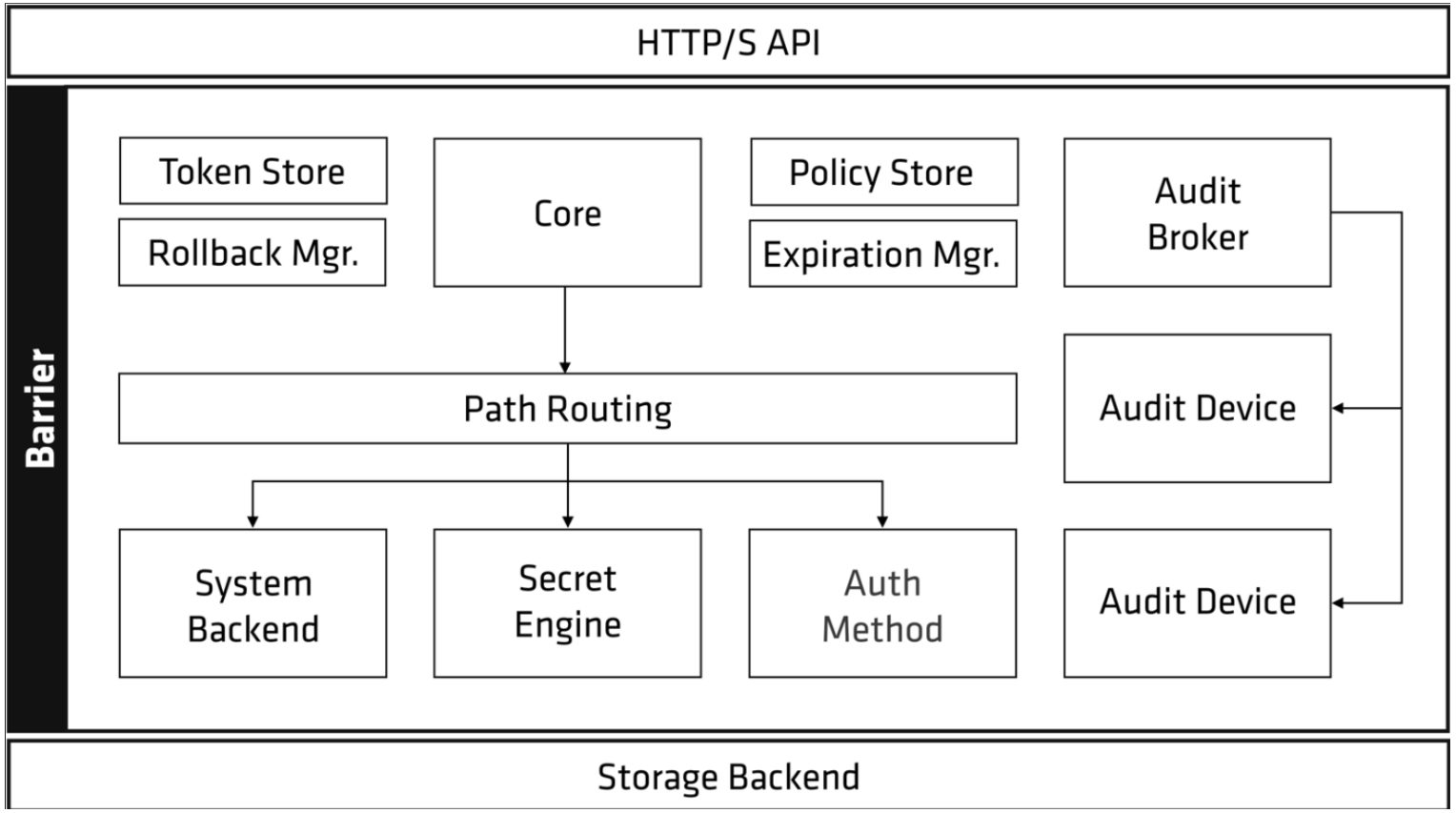
- Upon successful authorization, Vault grants the client access to secrets, keys, and encryption functions.
- The Vault token is issued to the client, which can be used for further interactions with Vault.
- The token is used in future operations until it expires or is revoked.



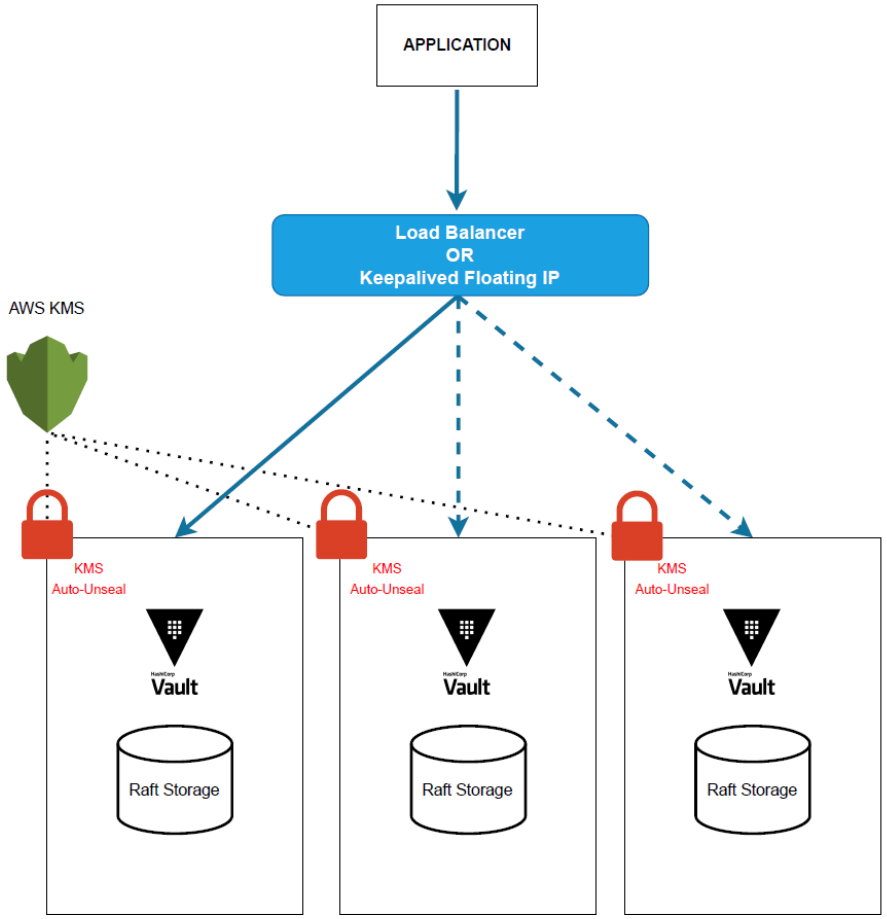
Vault Architecture



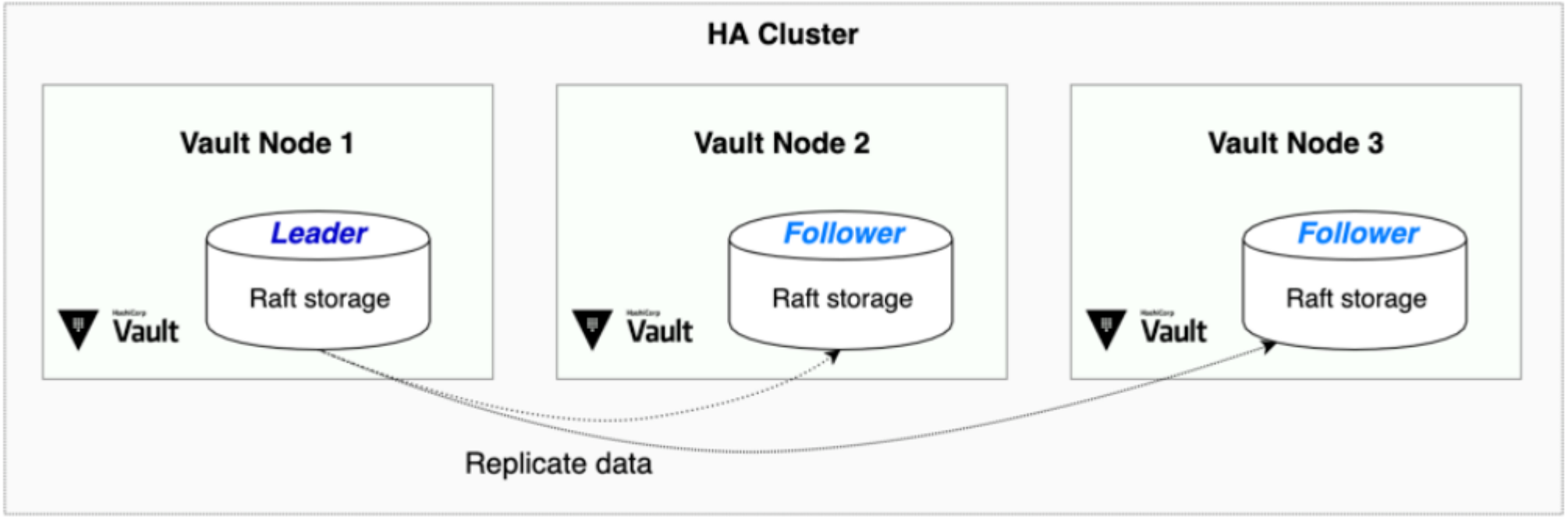
Vault Architecture



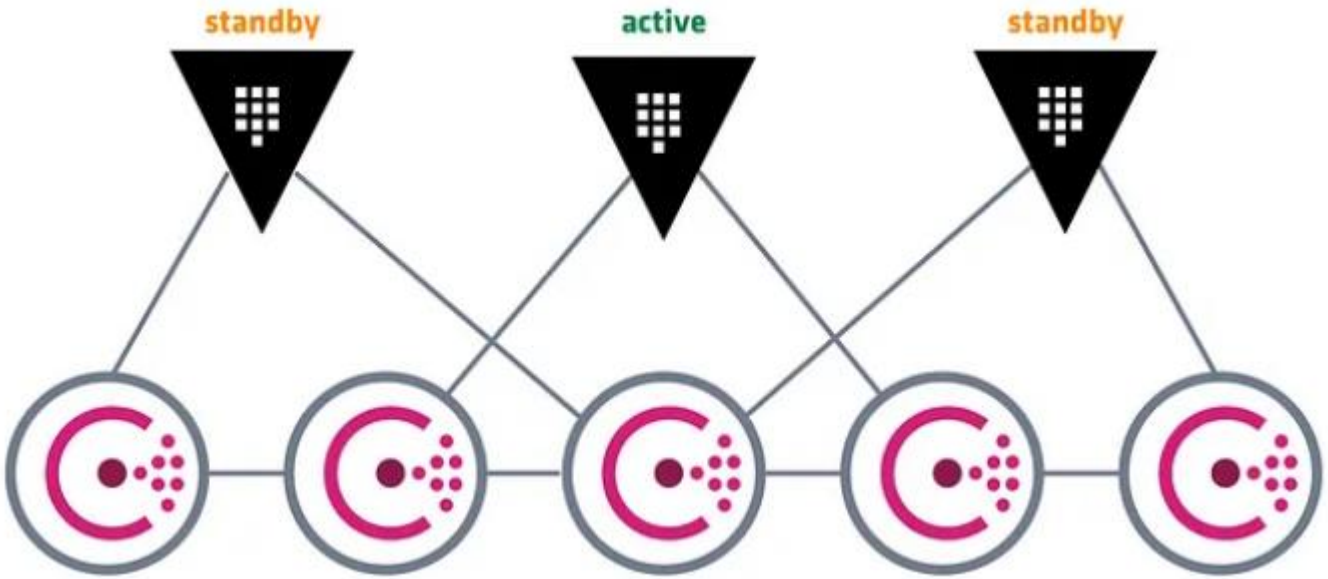
Vault HA Architecture



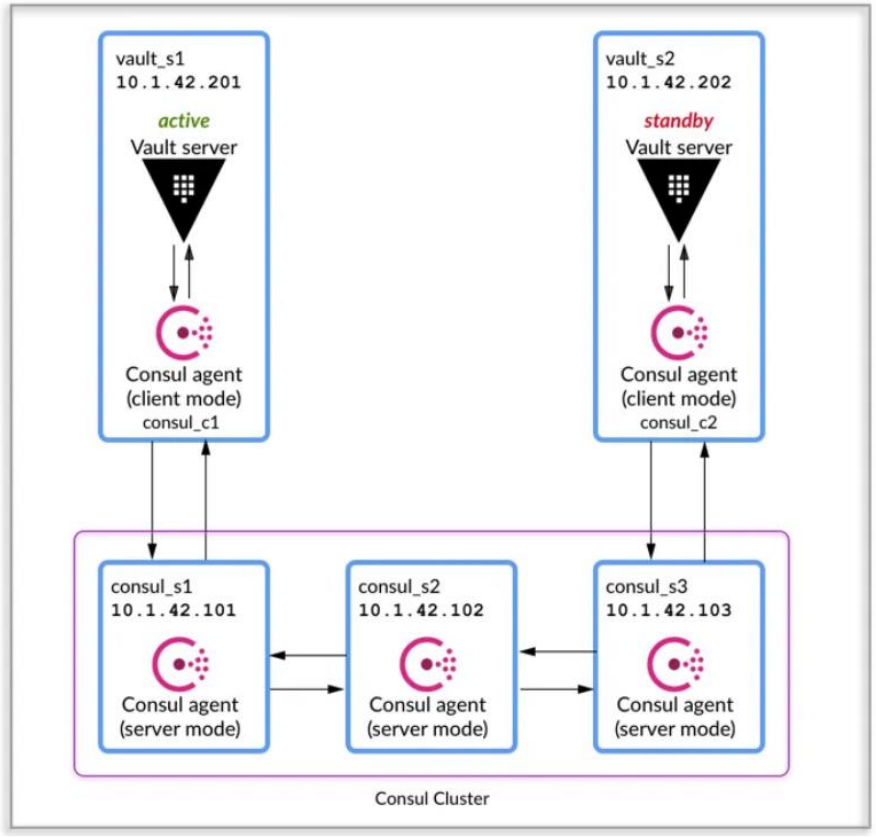
Vault HA Architecture



Vault HA with Concul Architecture



Vault HA with Concul Architecture



How to use Vault

Below are the methods to use it:

- CLI (Vault & API)
- HashiCorp Cloud Platform (HCP)
- UI

Vault Installation Methods

Below are the methods to install it:

- Install using a Linux Package
- Use a precompiled Binary
- Install from the Source

Vault Integration

There are two main types of integrations with Vault:

- Runtime Integrations
- Custom Plugins

Vault Integration : Runtime Integration

1. Runtime Integrations:

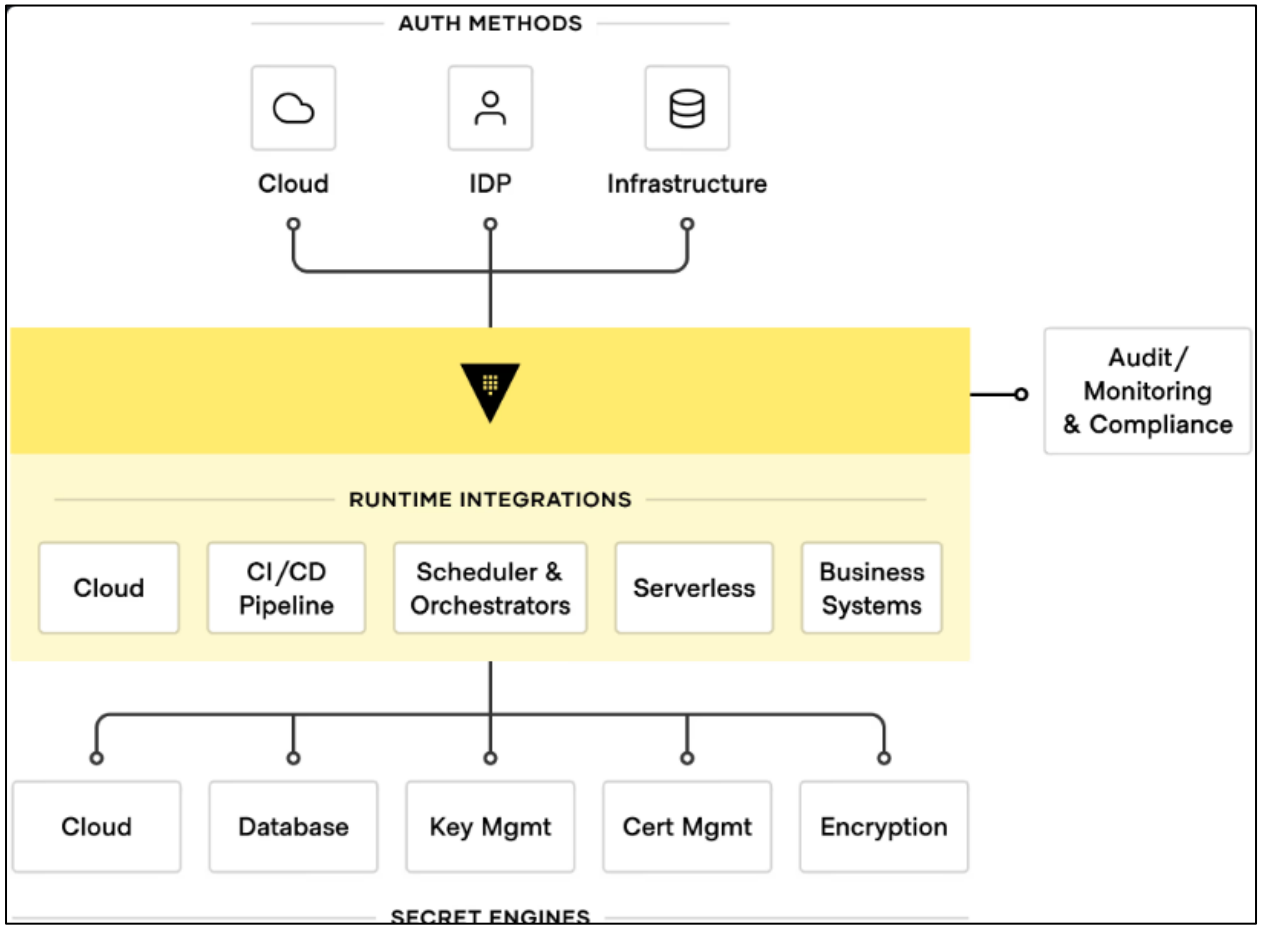
- Purpose: Use Vault as part of a workflow to manage secrets and encryption across applications, systems, and infrastructure.
- Functionality: Vault integrates with various platforms to:
  - Store and retrieve secrets (e.g., API keys, tokens, passwords).
  - Issue and manage PKI certificates.
  - Act as an external key management system (KMS).
- Use Cases: Vault can be integrated with applications to securely manage secrets, automate certificate issuance, and ensure encrypted data communication.

Vault Integration : Custom Plugins

2. Custom Plugins

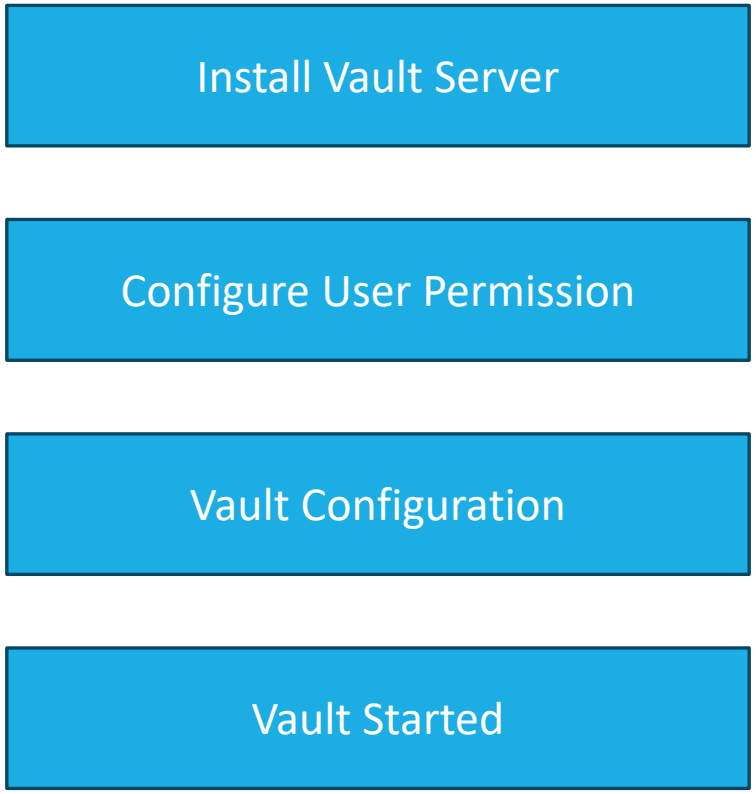
- Purpose: Partners or developers build custom plugins to extend Vault's functionality using its secure plugin architecture.
- Categories of Plugins:
  - Secrets Engines: Plugins for generating, storing, and accessing secrets.
  - Auth Methods: Plugins for managing authentication methods (e.g., LDAP, GitHub, AWS IAM).
- Types:
  - Built-in Plugins: Developed and maintained by HashiCorp; bundled with the Vault binary.
  - External Plugins: Developed by HashiCorp, technology partners, or the open-source community and must be manually registered.
- Curated Plugins Collection: A comprehensive list of both built-in and external plugins is available on the Vault Integrations page.

Vault Integration



Module 2:
Vault Installation

Vault Installation Steps:



Vault Installation: Install Vault Server using Binary

1. **Install Vault Server** (https://developer.hashicorp.com/vault/docs/install/install-binary)

Description	Commands
1. Download the binary	https://developer.hashicorp.com/vault/install#linux
2. Set the VAULT_DATA environment variable to your preferred Vault data directory	export VAULT_DATA=/opt/vault/data
3. Set the VAULT_CONFIG environment variable to your preferred Vault configuration directory.	export VAULT_CONFIG=/etc/vault.d
4. Move the Vault binary to /usr/bin	sudo mv PATH/TO/VAULT/BINARY /usr/bin/
5. Ensure the Vault binary can use mlock() to run as a non-root user	sudo setcap cap_ipc_lock=+ep $(readlink -f $(which vault))
6. Create your Vault data directory	$ sudo mkdir -p ${VAULT_DATA}
7. Create your Vault configuration directory	sudo mkdir -p ${VAULT_CONFIG}

Vault Installation: Install Vault Server using Package Manager

1. **Install Vault Server** (https://developer.hashicorp.com/vault/install#linux)

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install vault
```

Vault Installation: Configure User Permission

2. Configure User Permission

Description	Commands
1. Create a system user called vault to run Vault when your Vault data directory as home and nologin as the shell:	sudo useradd --system --home ${VAULT_DATA} --shell /sbin/nologin vault
2. Change directory ownership of your data directory to the vault user	sudo chown vault:vault ${VAULT_DATA}
3. Grant the vault user full permission on the data directory, search permission for the group, and deny access to others:	sudo chmod -R 750 ${VAULT_DATA}

Vault Installation: Vault Configuration

3. Vault Configuration:

Configuring Vault involves setting up the environment to manage secrets securely, ensure high availability, and define policies and access controls. Here's an overview of key aspects of Vault configuration:

- Configuration File: Vault server is configured via a configuration file, typically written in HCL (HashiCorp Configuration Language) or JSON.
  - Located at **/etc/vault.d/vault.hcl**

Key sections in the configuration file:

- **Storage Backend**: Defines where Vault stores its data. Common backends include:
- **Consul**: Distributed key-value store for high availability.
- **File**: Stores data on disk, typically for development.
- **Integrated Storage**: Vault's built-in storage solution.
- **Listener**: Specifies how Vault listens for client connections (HTTP or HTTPS).
- **Telemetry**: Enables telemetry data collection for monitoring.
- **Seal**: Configures the seal/unseal mechanism, such as AWS KMS or GCP KMS for auto-unsealing Vault.

Vault Installation: Vault Configuration

3. Vault Configuration:

```
ui            = true
cluster_addr  = "https://127.0.0.1:8201"
api_addr      = "https://127.0.0.1:8200"
disable_mlock = true

storage "raft" {
  path = "/path/to/raft/data"
  node_id = "raft_node_id"
}

listener "tcp" {
  address       = "127.0.0.1:8200"
  tls_cert_file = "/path/to/full-chain.pem"
  tls_key_file  = "/path/to/private-key.pem"
}

telemetry {
  statsite_address = "127.0.0.1:8125"
  disable_hostname = true
}

storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault/"
}

seal "awskms" {
  region = "us-east-1"
  kms_key_id = "example-kms-key-id"
}
```

Vault Seal/Unseal

- Vault Seal and Unseal is a critical security mechanism used to protect the contents of HashiCorp Vault.
- The data stored in Vault is encrypted, and the encryption key is stored in memory when Vault is unsealed.
- If Vault is sealed, the encryption key is not accessible, making the data in Vault unusable until it is unsealed again.
- By default the vault server is sealed.

Vault Seal/Unseal

1. Vault Seal

Purpose: To protect Vault's data by encrypting it and making it inaccessible without the master key.

When Sealed: Vault is sealed by default when it starts up or when a manual seal command is issued.

How It Works:

- Vault's data encryption key is encrypted by a master key.
- This master key is not stored on disk but is split into multiple key shares using Shamir’s Secret Sharing algorithm.
- A minimum number of key shares (called the unseal threshold) are required to reconstruct the master key and unseal Vault.

Vault Seal/Unseal

2. Vault Unseal

Purpose: To bring Vault back to a usable state where it can decrypt its secrets.

How It Works:

- Vault needs to be unsealed after startup or any seal action. The unseal process involves providing the key shares to reconstruct the master key.
- A configured number of key holders must each provide their unique key share.
- Once enough shares are entered to meet the unseal threshold, Vault will decrypt its data encryption key and become operational.

Example: If Vault is configured with 5 key shares and an unseal threshold of 3, you would need 3 key holders to provide their keys to unseal Vault.

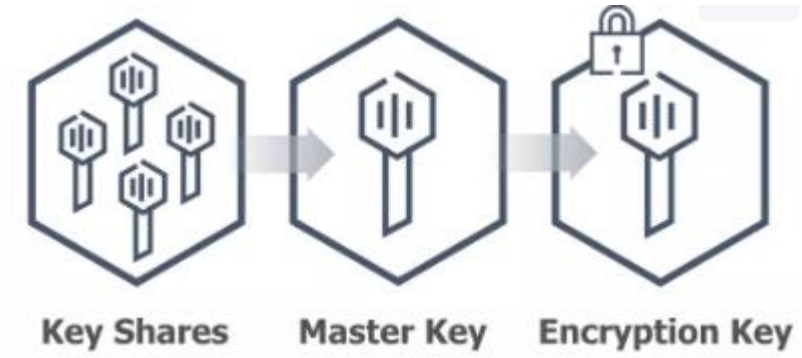
Vault Seal/Unseal

3. Shamir's Secret Sharing

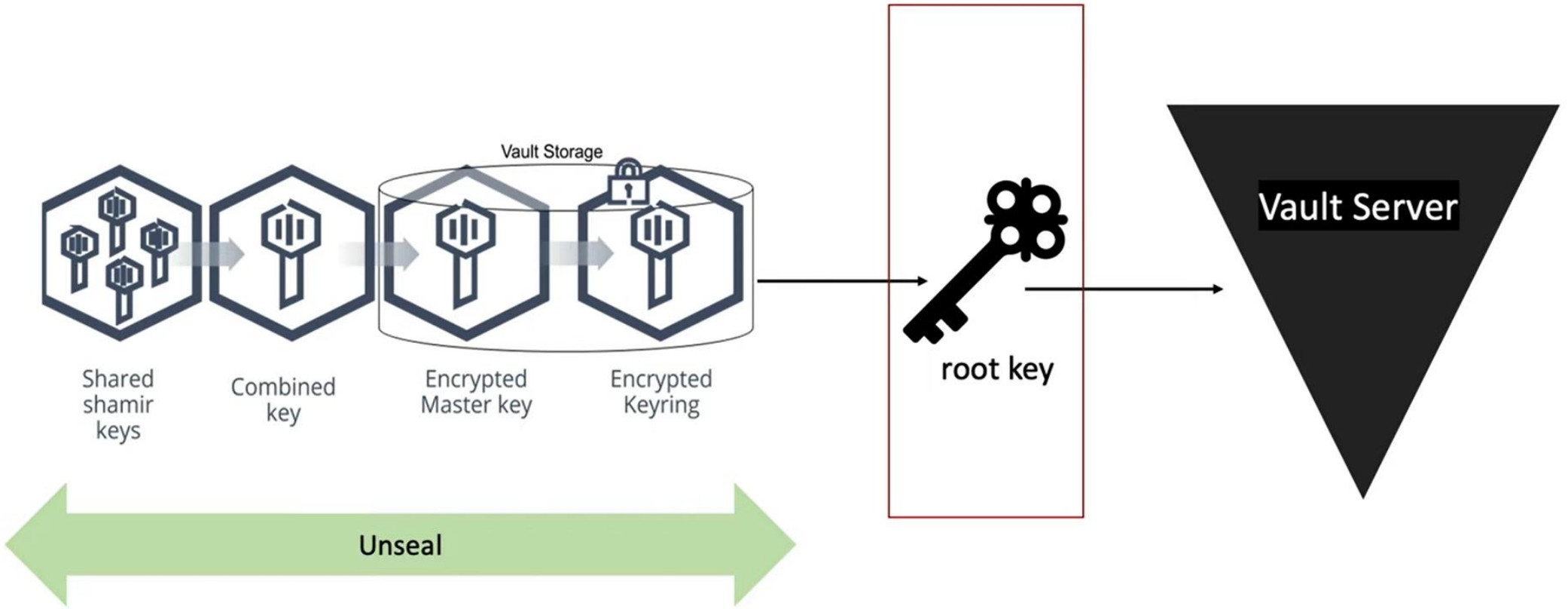
Concept: The master key is split into multiple key shares, distributed among different operators or key holders. Only a portion of these key shares (the unseal threshold) are needed to unseal Vault.

- Split Master keys in to K shares
- Unsealing requires T keys to meet unseal threshold
- Default: K=5 T=3

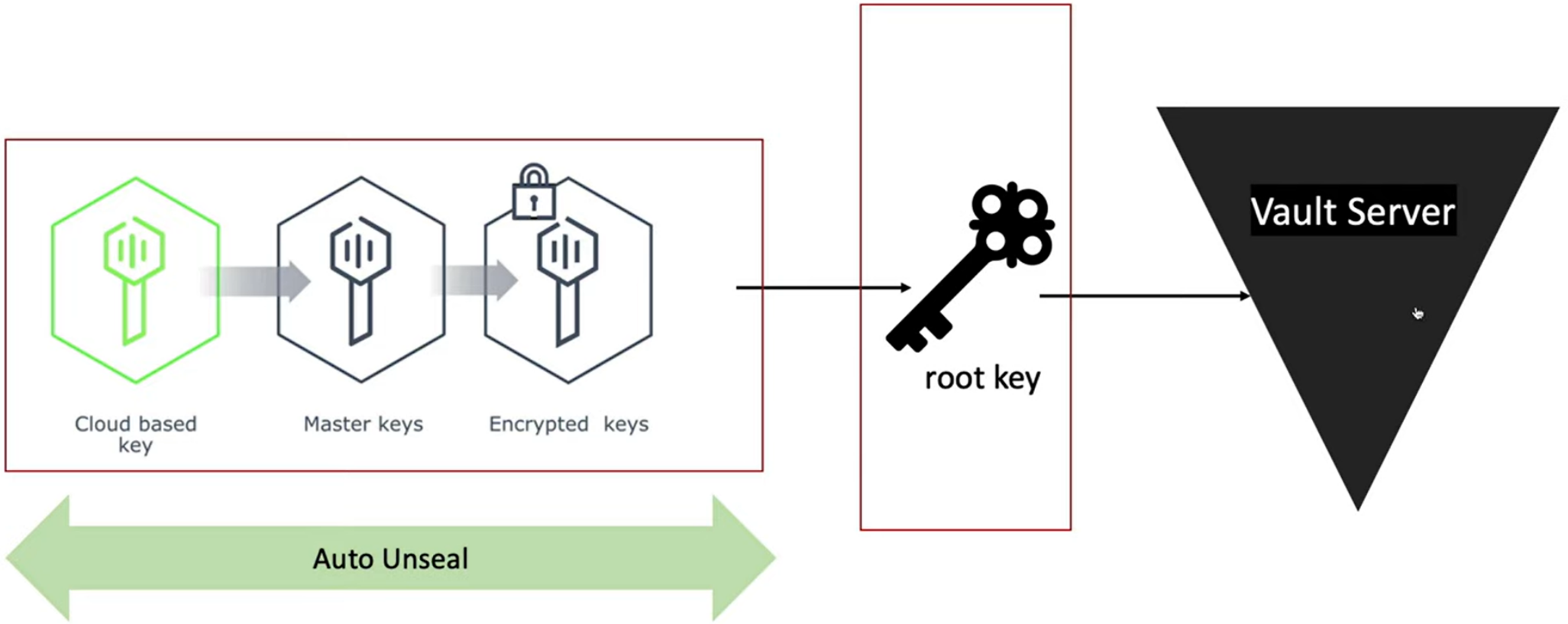
Purpose: Increases security by requiring multiple people to participate in the unsealing process.



Vault Seal/Unseal



Vault Seal/Unseal: Auto unseal using Cloud



Vault Seal/Unseal

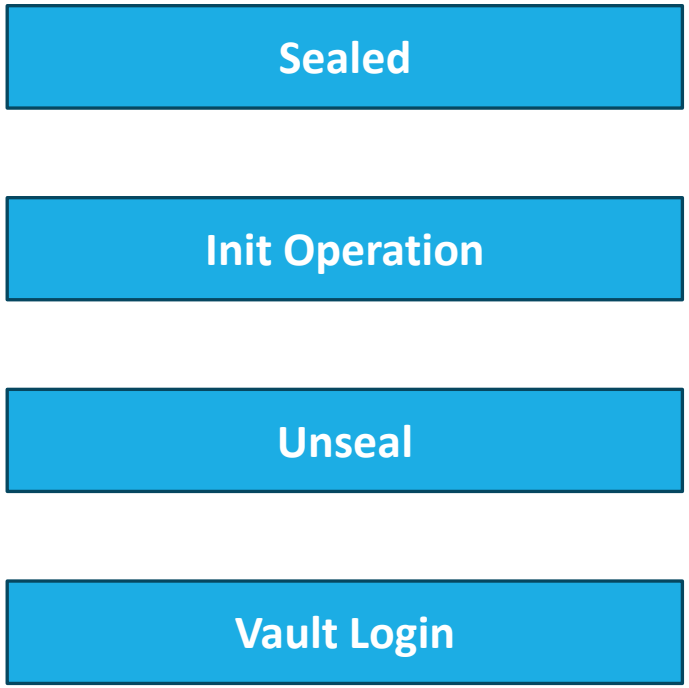
4. Automatic Unseal

Purpose: To remove the need for manually unsealing Vault each time it restarts.

How It Works: Vault integrates with cloud-based Key Management Services (KMS) such as AWS KMS, Azure Key Vault, or Google Cloud KMS to automatically unseal Vault.

Benefit: Simplifies management and reduces downtime, especially in environments where Vault needs to be highly available.

Vault Login Steps



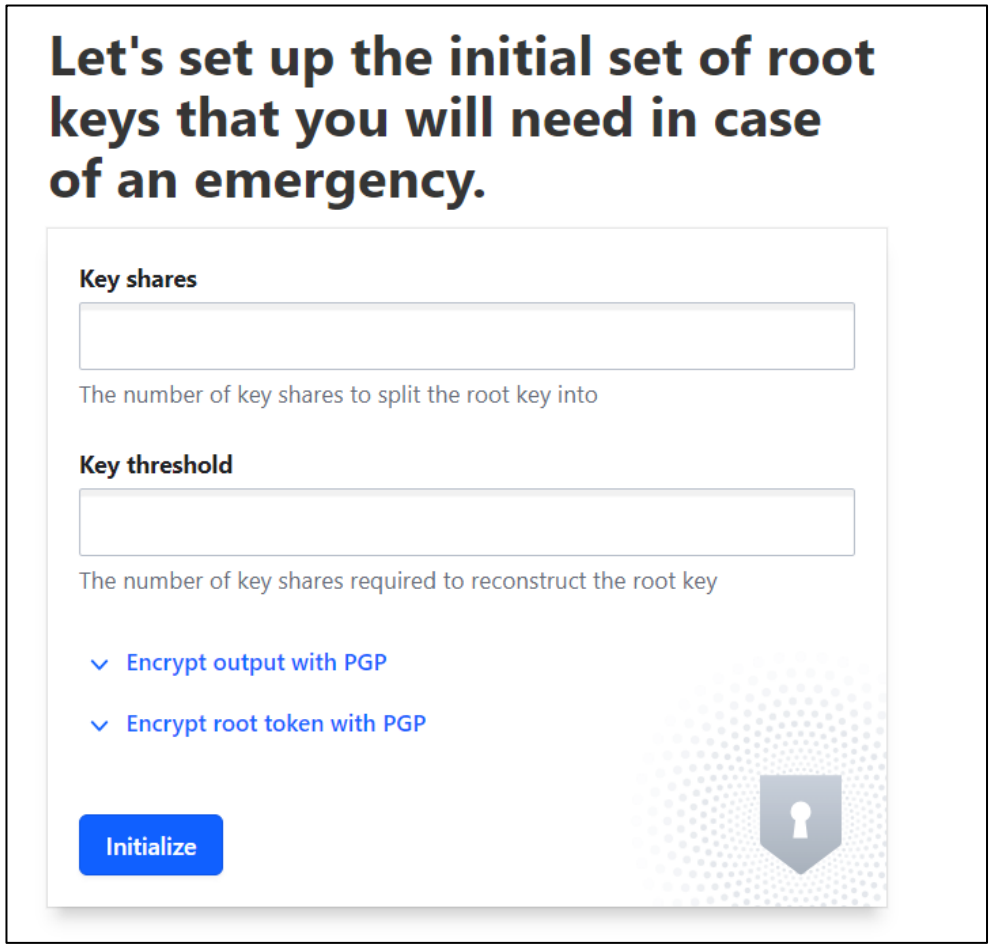
Vault Ports

Port	Protocol	Description
8200	HTTP/HTTPS	Default port for Vault client communication (HTTP or HTTPS)
8201	TCP	Used for HA (High Availability) node-to-node communication
443	TCP	API Requests
8500/8501	TCP	Storage Backend Port (Consul)

Vault Initial Configuration

```
ui = true

#mlock = true
#disable_mlock = true

storage "file" {
  path = "/opt/vault/data"
}

#storage "consul" {
#  address = "127.0.0.1:8500"
#  path    = "vault"
#}

# HTTP listener
listener "tcp" {
  address = "172.31.4.218:8200"
  tls_disable = 1
}

# HTTPS listener
#listener "tcp" {
 # address       = "172.31.4.218:8200"
 # tls_cert_file = "/opt/vault/tls/tls.crt"
 # tls_key_file  = "/opt/vault/tls/tls.key"
#}
```

Vault Init Operator



Vault Init Operator

Vault Initialization Steps:

Storage Backend Preparation:

- Initialization is required to prepare the storage backend (e.g., Consul, S3, etc.) to store Vault data.
- In HA mode, since multiple Vault servers share the same storage backend, you only need to initialize one Vault server. The others will join the initialized cluster.

Root Key Generation:

- Vault generates a root key during initialization, which is stored securely in the backend along with other Vault data.
- This root key is encrypted, meaning it can only be decrypted using unseal keys.

Shamir's Secret Sharing:

- By default, Vault uses Shamir's Secret Sharing to divide the root key into multiple key shards (called unseal keys).
- A certain number of these key shards (called the threshold) are needed to reconstruct the root key and decrypt Vault's encryption key.

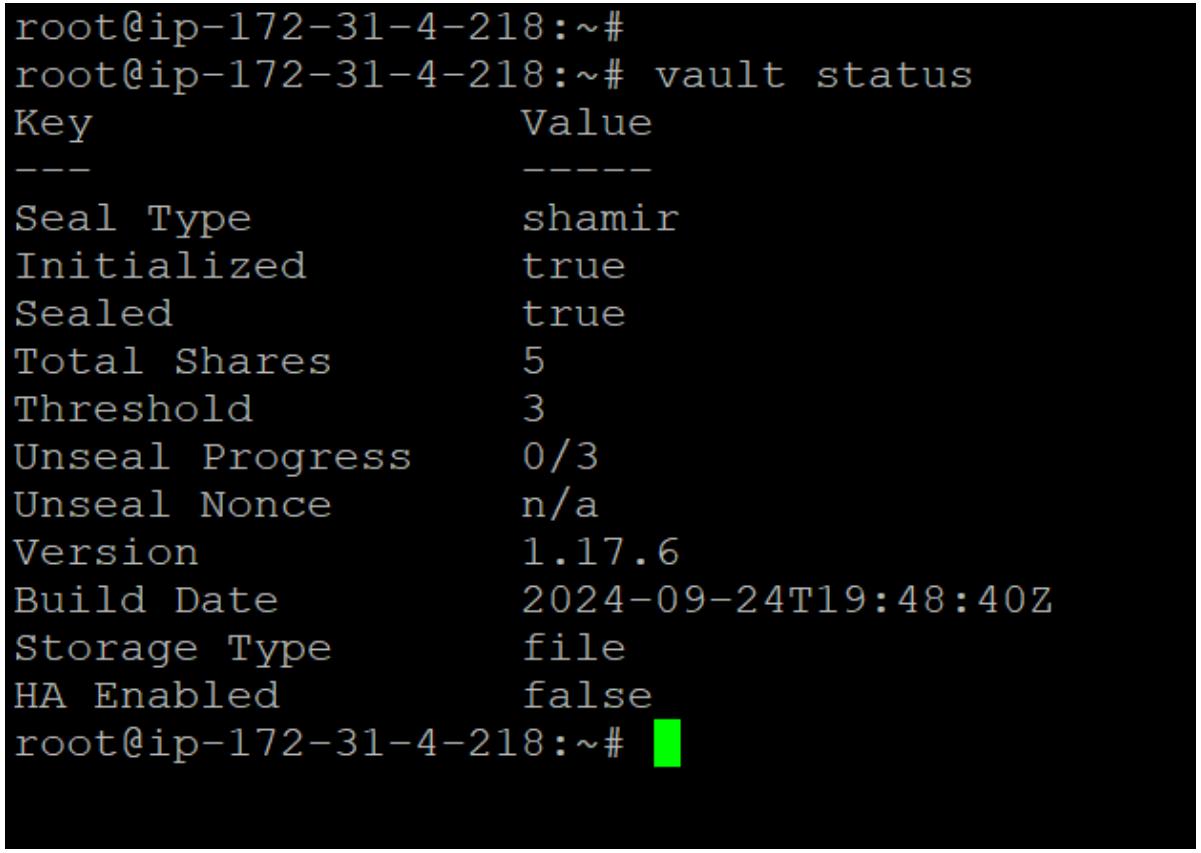
Unsealing Vault:

- Once initialized, Vault is in a sealed state. To unseal it and begin using Vault, you need to provide a certain number of unseal keys (as configured during initialization).

Vault Init Operator

```
root@ip-172-31-4-218:~# export VAULT_ADDR='http://127.0.0.1:8200'
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault operator init
Unseal Key 1: aFSRNdQuPaOCMj1FBHrEA/9iiuZ/M0uxmiIfOapyqJ6O
Unseal Key 2: cRHQDHjLAXL5DxUr/IDbSgTUvvSBRkdeSd2T+UAqvVzC
Unseal Key 3: MNAjc0eNUI3A5WsZ8BpvRwAVxR+lpmP/+D5/DuRHNdYD
Unseal Key 4: /CwTWyEnpZ/jtNoCSo/evpq7EfMyzAz6B/0FPUcRbidy
Unseal Key 5: RAsnJWgPtIIDv0q2ldLCb1j4zmN+ToGzo/L90gVbLDWv

Initial Root Token: hvs.7JqBoMDAyni6zmlWzhyG8uF6

Vault initialized with 5 key shares and a key threshold of 3. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal it
before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to
reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more information.
root@ip-172-31-4-218:~#
```

Vault Init Operator



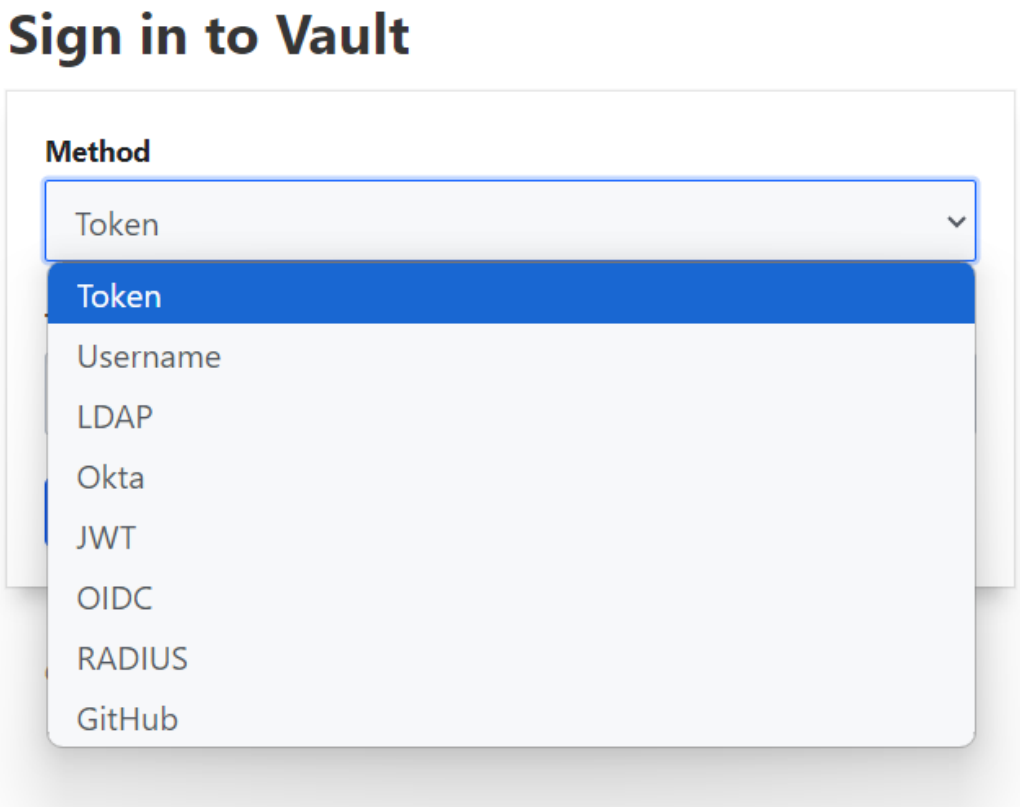
Vault Init Operator

Unseal can be done via UI and command:

vault operator unseal cRHQDHjLAXL5DxUr/IDbSgTUvvSBRkdeSd2T+UAqvVzC

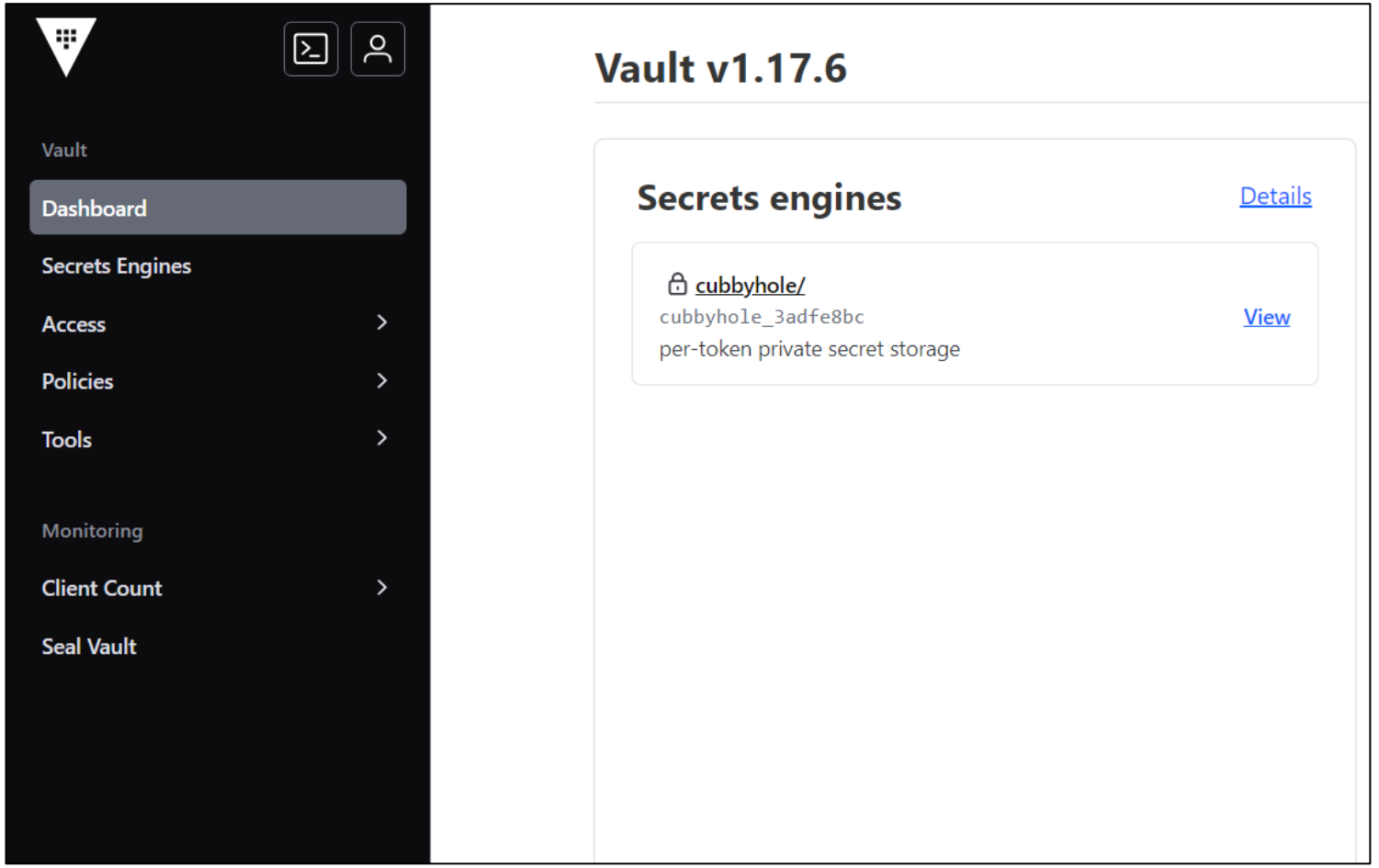
```
root@ip-172-31-4-218:~# vault operator unseal cRHQDHjLAXL5DxUr/IDbSgTUvvSBRkdeSd2T+UAqvVzC
Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             true
Total Shares       5
Threshold          3
Unseal Progress    2/3
Unseal Nonce       77a552f4-b9ef-3c74-b06a-5fb2639ee893
Version            1.17.6
Build Date         2024-09-24T19:48:40Z
Storage Type       file
HA Enabled         false
root@ip-172-31-4-218:~# vault operator unseal MNAjc0eNUI3A5WsZ8BpvRwAVxR+lpmP/+D5/DuRHNdYD
Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    5
Threshold       3
Version         1.17.6
Build Date      2024-09-24T19:48:40Z
Storage Type    file
Cluster Name    vault-cluster-daaa5cf5
Cluster ID      128b1152-4678-2050-8b12-af578db1fe07
HA Enabled      false
root@ip-172-31-4-218:~#
```

Vault Sign-in



Vault Sign-in

Sign in with the root token



Vault Sign-in

Sign in with the root token

```
root@ip-172-31-4-218:~# vault login
Token (will be hidden):
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                hvs.7JqBoMDAyni6zmlWzhyG8uF6
token_accessor       FhztpfwzMl4EG8WDaR4FIQq5
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
root@ip-172-31-4-218:~#
```

Vault Development Mode

Vault Development Mode (vault server -dev)

What is Vault Dev Mode?

- Single Node Setup: Dev mode starts a single Vault server instance.
- No Persistent Storage: All data is stored in-memory, meaning that when the server is stopped, all the data is lost.
- Automatic Unseal: Vault is automatically unsealed and initialized with a single unseal key.
- Root Token: A root token is automatically generated and printed to the console.
- Low Security: Dev mode is highly insecure (for example, it allows access without authentication), so it should only be used for testing.

Reinitializing HashiCorp Vault

Reinitializing HashiCorp Vault means starting from scratch by creating a new instance of the Vault, which will destroy all existing data and configurations in the Vault. This process generates new unseal keys and a root token. Reinitialization should only be done if you are sure that recovery is not possible and you are prepared to lose all current data.

Steps	Commands
1. Stop vault service	systemctl stop vault
2. Clear Old Data	rm -rf /opt/vault/data/
3. Start Vault in Initialization Mode	systemctl start vault
4. Initialize Vault	vault operator init
5. Unseal the vault	vault operator unseal
6. Login as root	vault login

Root Key

The root token in Vault is a special administrative token that provides unrestricted access to all Vault operations, including:

- Managing policies
- Creating or revoking tokens
- Configuring the Vault
- Unsealing Vault (in some configurations)
- Accessing all secrets and configuring secret engines

Impact of losing Root Key/Token

If you lose the root key (root token) while the Vault is operational, the consequences are:

No Loss of Data: The data stored in Vault is not directly impacted. The root key/token is not required to access encrypted data, as long as other tokens or policies provide access to the secrets.

Loss of Administrative Control: Losing the root key means you no longer have the highest level of administrative privileges, which could make it difficult to manage certain aspects of Vault. For instance:

- You cannot create new tokens or policies with unrestricted access.
- You might lose the ability to revoke certain tokens or control access at the highest level.

Cannot Perform Critical Administrative Operations: If you need to perform tasks such as enabling or disabling authentication methods, mounting new secret engines, or managing tokens with high-level permissions, you will not be able to do this without the root key.

Policy Updates Restricted: Policy changes (e.g., granting users more privileges, managing authentication methods) require the root token or a token with similar permissions. Without the root key, these tasks would be more difficult unless another admin token exists with similar privileges.

Mitigation and Recovery option

Generate a New Root Token (Root Token Regeneration):

- If the root token is lost, and you still have unseal keys, you can generate a new root token using the unseal keys.

Create Admin Tokens with Full Access:

- If you regularly need high-level administrative privileges, create tokens with policies that allow administrative access, but not as unrestricted as the root token. This reduces reliance on the root token for daily operations.

Root Token Best Practices

Minimize Root Token Use: Use the root token only for essential configuration tasks (like setting up authentication, policies, etc.), then revoke it or store it securely.

Securely Store the Root Token: If you must keep the root token for recovery purposes, store it in a secure location (e.g., an encrypted vault or password manager) with restricted access.

Use MFA for Root Token: If supported, enable multi-factor authentication (MFA) or additional layers of security for root token access.

Regenerate root token

If you have lost the root token, and you need to generate a new one using the unseal keys.

```
root@ip-172-31-4-218:~# vault operator generate-root -init
A One-Time-Password has been generated for you and is shown in the OTP field.
You will need this value to decode the resulting root token, so keep it safe.
Nonce         7a2ae619-0fff-b874-4094-95f6e92d81a9
Started       true
Progress      0/3
Complete      false
OTP           rwZr6BguvFk3z4Syg5FhedqLcD3V
OTP Length    28
```

Nonce: This is the unique identifier for the root generation process. It is used during subsequent steps when submitting unseal keys.

OTP: This is the One-Time-Password (OTP). You need to keep this OTP safe, as it will be used to decode the resulting root token after the process is complete.

Regenerate root token

If you have lost the root token, and you need to generate a new one using the unseal keys.

Steps	Commands
1. Initialize Root Token Generation	vault operator generate-root -init
2. Generate using nonce which is a unique identifier. Provide the unseal keys further	vault operator generate-root -nonce=7a2ae619-0fff-b874-4094-95f6e92d81a9
3. Decode the root key using the OTP	vault operator generate-root -decode=GgEpXFUWHxgeCjhePmEdNi1TKlFTDUEaUChRJw -otp=rwZr6BguvFk3z4Syg5FhedqLcD3V

Impact of losing Unseal Keys

Losing the unseal keys in HashiCorp Vault can have serious consequences, particularly depending on the type of Vault setup you're using and whether it’s sealed or unsealed.

If Vault is Sealed:

Data Access Impact: Without the unseal keys, you cannot unseal the Vault, which means you won’t be able to access any secrets or data stored in Vault.

Data is Encrypted: While Vault is sealed, all data remains encrypted and inaccessible. Vault encrypts data before writing it to disk, and unsealing is necessary to decrypt the data and make Vault operational again.

Impact of losing Unseal Keys

If Vault is Unsealed:

Data Access Remains Intact:

If Vault is already unsealed, you can continue to access the data as long as it remains unsealed. However, if Vault restarts or is sealed (either automatically or manually), you will not be able to unseal it again without the unseal keys.

Cannot Recover Vault after Restart:

If the server hosting Vault restarts (e.g., for maintenance or a crash), Vault will return to a sealed state. Without the unseal keys, it will be impossible to unseal Vault and access the data.

Impact of losing Unseal Keys

No Data Corruption or Loss (While Sealed):

The loss of the unseal keys does not cause data corruption or direct data loss, as Vault still stores the encrypted data. However, you will lose access to the data if the unseal keys are not recovered.

How to Recover from Lost Unseal Keys:

Unfortunately, if you've completely lost the unseal keys and do not have a backup, there is no way to recover the unseal keys or access the data stored in Vault. The keys are critical for decrypting the Vault’s master key and unsealing Vault.

- Backup the unseal keys
- Auto-Unseal
- Backup with HSM

How to Recover from Lost Unseal Keys:

Unfortunately, if you've completely lost the unseal keys and do not have a backup, there is no way to recover the unseal keys or access the data stored in Vault. The keys are critical for decrypting the Vault’s master key and unsealing Vault.

Precautions:

- Backup the unseal keys
- Auto-Unseal
- Backup with HSM

How to Recover from Lost Unseal Keys:

Precautions:

- **Backup the unseal keys**

- When Vault is initialized, it generates unseal keys. It's crucial to safely back up these keys in a secure, distributed manner (e.g., store them in a secure password manager, distribute among trusted team members).
- Vault's Shamir Secret Sharing method generates multiple unseal keys, and you only need a threshold (e.g., 3 out of 5) to unseal the Vault. Distribute these keys securely among several team members.

How to Recover from Lost Unseal Keys:

Precautions:

- **Use Auto-Unseal (for Production):**

- To avoid the need for manual unsealing, Vault can be configured with an auto-unseal mechanism using cloud-based key management services (e.g., AWS KMS, Azure Key Vault, GCP KMS). This allows Vault to automatically unseal without requiring the manual unseal keys.
- This approach provides both high availability and disaster recovery, as it eliminates the need to store unseal keys manually.

- **Key Backup with HSM (Hardware Security Module):**

- If your organization uses a Hardware Security Module (HSM), you can integrate Vault with it for secure key storage and recovery.

Unseal Rekey

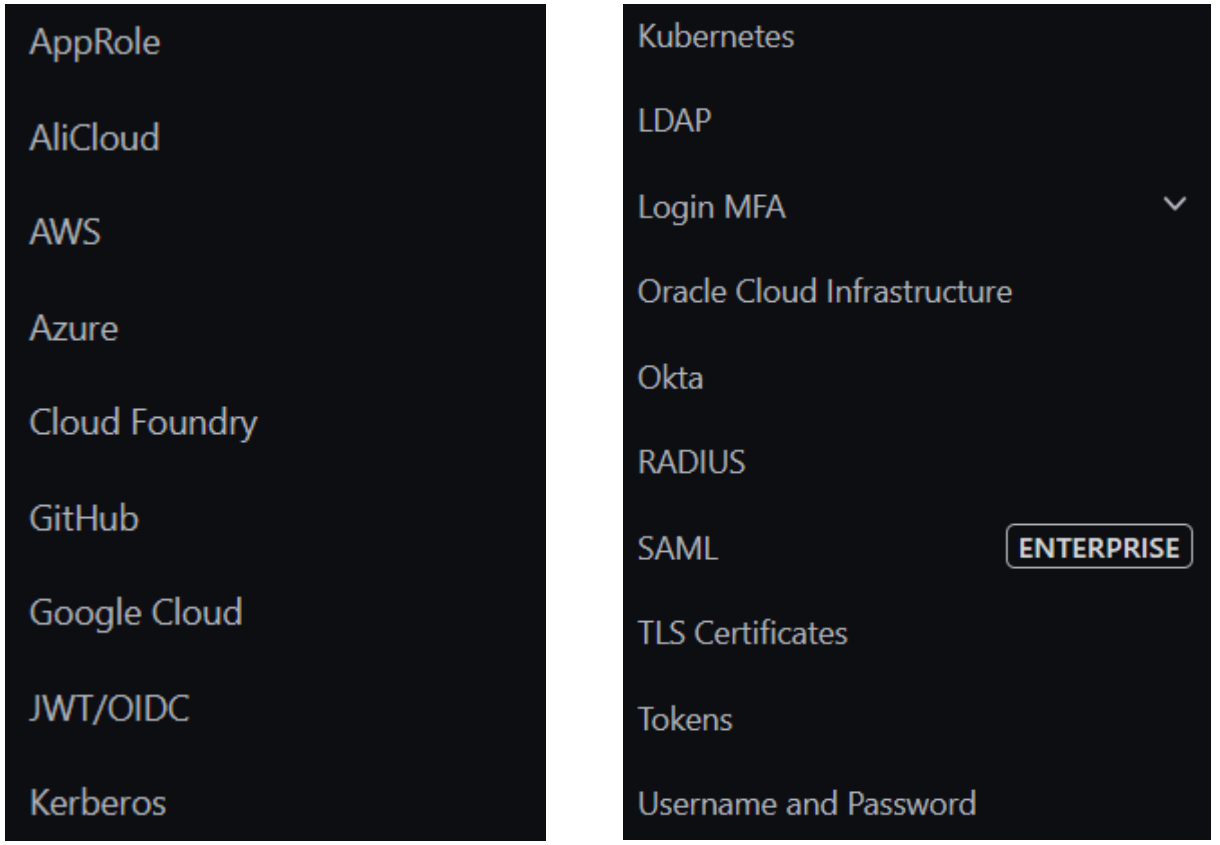
The command vault operator rekey is used to rotate or change the unseal keys (Shamir's Secret Shares) in HashiCorp Vault. This process allows you to change the number of shares or the threshold required to unseal Vault without needing to reinitialize Vault.

Commands:

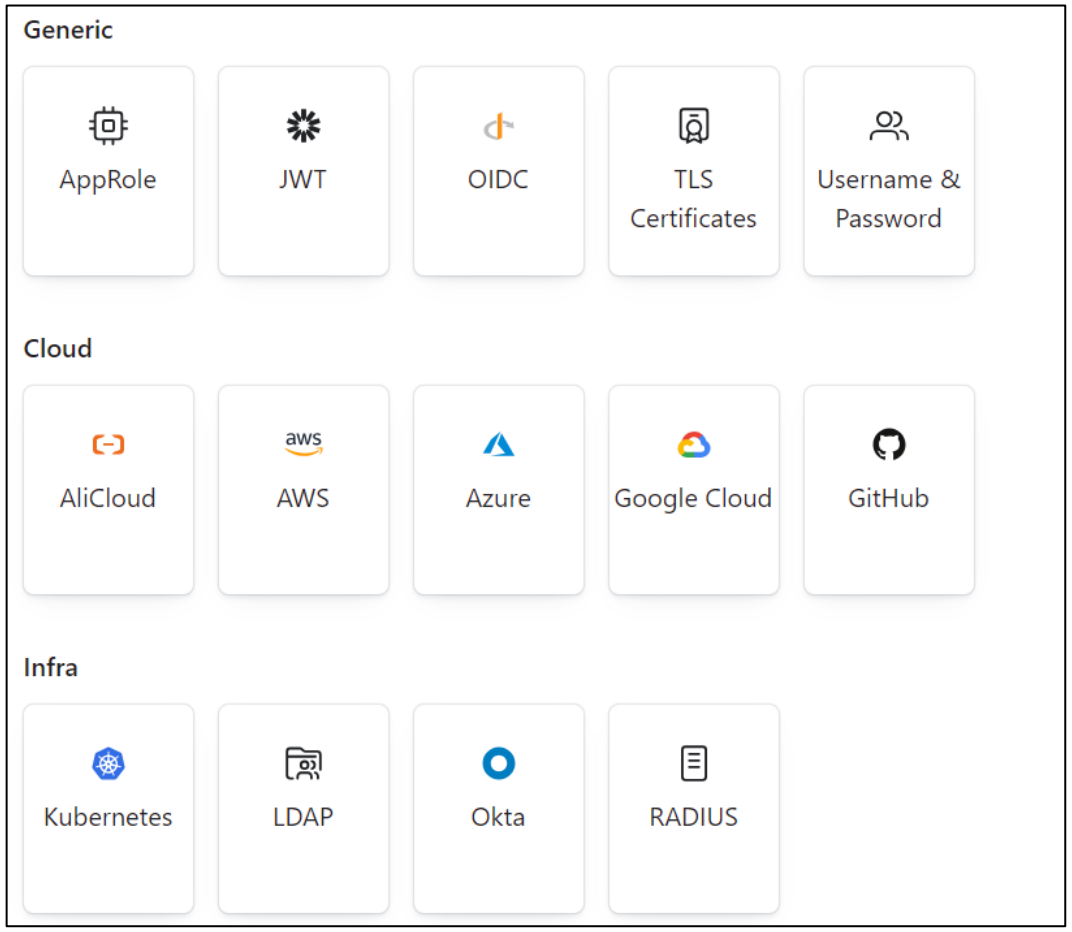
- vault operator rekey -init -key-shares=5 -key-threshold=3
- vault operator rekey -nonce=<nonce>
- vault operator rekey -status
- vault operator rekey -cancel

Module 3:
Authentication Methods

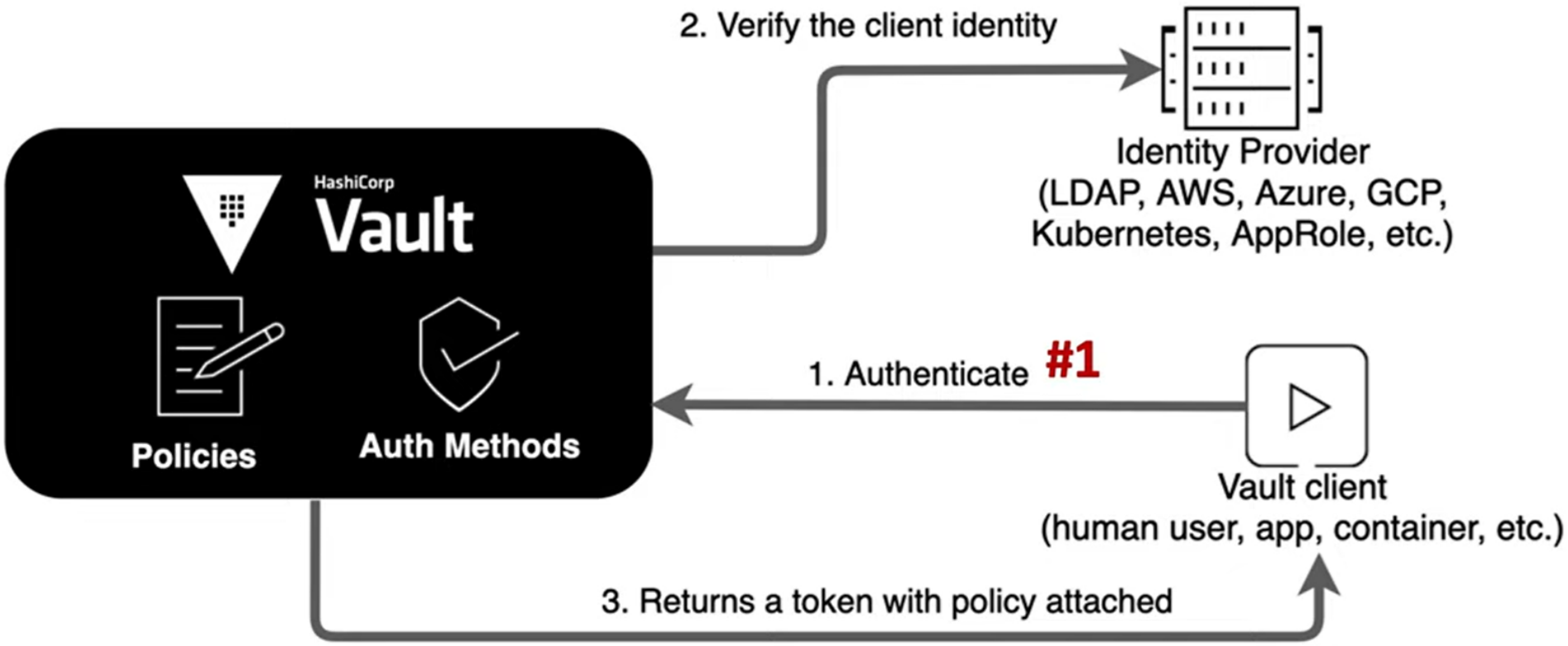
Vault Authentication Methods



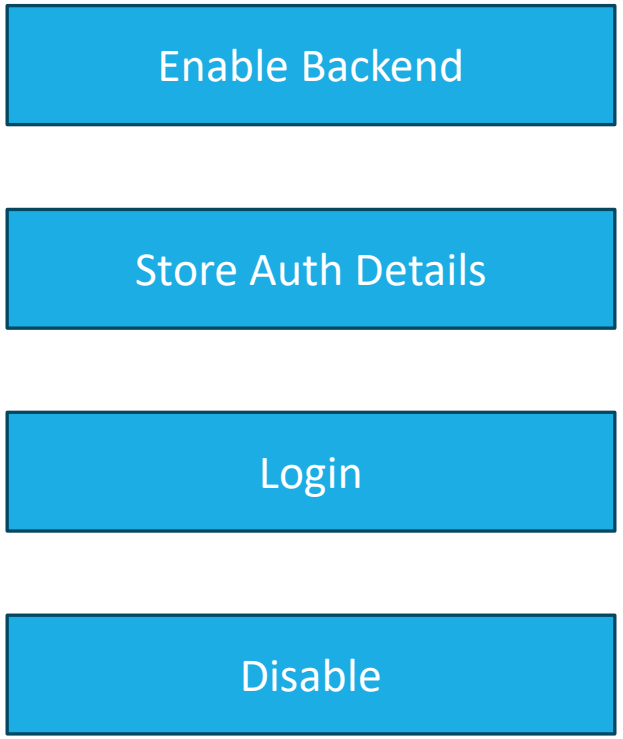
Vault Authentication Methods



Vault Authentication Flow



Vault Authentication Steps



Vault Authentication Token

Token Creation:

- Vault can create tokens directly using the vault token create command. You can create tokens with different capabilities, such as different TTLs (time-to-live), renewable or non-renewable, and different policies.

Token Renewal:

- Tokens can be renewed if they are renewable, extending their TTL to prevent them from expiring. This is useful for long-running processes or users who need continued access.

Token Revocation:

- Tokens can be revoked manually, or they will automatically expire when their TTL ends. Vault also supports revocation of tokens along with their associated secrets.

Token Orphaning:

- You can create orphaned tokens, which are not linked to any parent tokens, making them independent of other tokens in the system.

Token Accessors:

- Vault provides accessors that can be used to manage tokens without having the full token. Accessors can be used to revoke tokens without exposing the token itself.

Vault Authentication Token

Command	Description	Example
vault login <token>	Log in to Vault using a token.	vault login s.XYZ123
vault token create	Creates a new token with default TTL and policies.	vault token create
vault token create -policy=<policy> -ttl=<ttl>	Create a token with specific policy and TTL.	vault token create -policy="my-policy" -ttl="1h"
vault token create -orphan	Create an orphan token (independent of parent token).	vault token create -orphan
vault token renew <token>	Renews a token, extending its TTL if it's renewable.	vault token renew s.XYZ123
vault token revoke <token>	Revokes a token, invalidating it for future use.	vault token revoke s.XYZ123

Vault Authentication Token

Command	Description	Example
vault token revoke -accessor <accessor>	Revoke a token using its accessor, without knowing the full token.	vault token revoke -accessor jkl567
vault token lookup <token>	Lookup details about a specific token, including TTL and policies.	vault token lookup s.XYZ123
vault token lookup –accessor accessor-id	Lookup details about a specific token using accessor, including TTL and policies.	vault token lookup -accessor=byk6qF4fNUcqC2HfdeeVi7sx
vault token lookup	Look up details about the token you are currently authenticated with.	vault token lookup
vault list auth/token/accessors	List all tokens by their accessors.	vault list auth/token/accessors
vault write auth/token/roles/<role-name>	Create or update a token role with specific constraints, such as policies, TTL, and orphan settings.	vault write auth/token/roles/my-role allowed_policies="my-policy" orphan=true renewable=true
vault read auth/token/roles/<role-name>	Read details about a specific token role.	vault read auth/token/roles/my-role

What is an Accessor?

- An accessor is a unique identifier assigned to each token in Vault.
- It is used to retrieve and manage token metadata without exposing the actual token itself.
- This enhances security by allowing operations on tokens without revealing their sensitive values.

Historical Record: The accessors remain in the system for auditing and historical purposes. They provide a record of all tokens that have been created, used, and revoked.

Token Accessor vs. Token ID:

The token ID is the actual string used to authenticate against Vault, while the accessor ID is used internally for token management. You can think of the accessor as a reference to the token that allows you to look up information about the token (like policies, TTL, etc.) without exposing the token itself.

Vault Authentication Token

```
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault auth list
Path      Type     Accessor               Description                Version
----      ----     --------               -----------                -------
token/    token    auth_token_e8591e67    token based credentials    n/a
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault token lookup
Key                  Value
---                  -----
accessor             FhztpfwzMl4EG8WDaR4FIQq5
creation_time        1728149668
creation_ttl         0s
display_name         root
entity_id            n/a
expire_time          <nil>
explicit_max_ttl     0s
id                   hvs.7JqBoMDAyni6zmlWzhyG8uF6
meta                 <nil>
num_uses             0
orphan               true
path                 auth/token/root
policies             [root]
ttl                  0s
type                 service
root@ip-172-31-4-218:~#
```

Vault Auth Mount Paths

- In HashiCorp Vault, the auth mount path refers to the endpoint where authentication methods are enabled.
- Each authentication method (auth method) can have its own mount path, allowing you to customize how clients authenticate with Vault.

Auth Method	Default Mount Path	Description
Token	/auth/token	Allows users to authenticate using tokens.
AppRole	/auth/approle	Allows machines and applications to authenticate using role-based access.
Userpass	/auth/userpass	Allows users to authenticate with a username and password.
LDAP	/auth/ldap	Enables authentication against an LDAP directory.
GitHub	/auth/github	Allows authentication via GitHub accounts.
OIDC	/auth/oidc	Allows authentication using OpenID Connect providers.
AWS	/auth/aws	Allows AWS IAM users and roles to authenticate.
Kubernetes	/auth/kubernetes	Enables authentication via Kubernetes service accounts.
Azure	/auth/azure	Allows Azure AD users to authenticate.
Radius	/auth/radius	Enables authentication using RADIUS servers.

Vault Auth Userpass

The **Userpass** authentication method in HashiCorp Vault allows users to authenticate using a **username** and **password**. This method is particularly useful for applications where users need to log in with credentials that are familiar to them.

- **Mount Path:** /auth/userpass
- **Purpose:** Provides a simple way for users to authenticate against Vault using a username and password.

Vault Auth Userpass

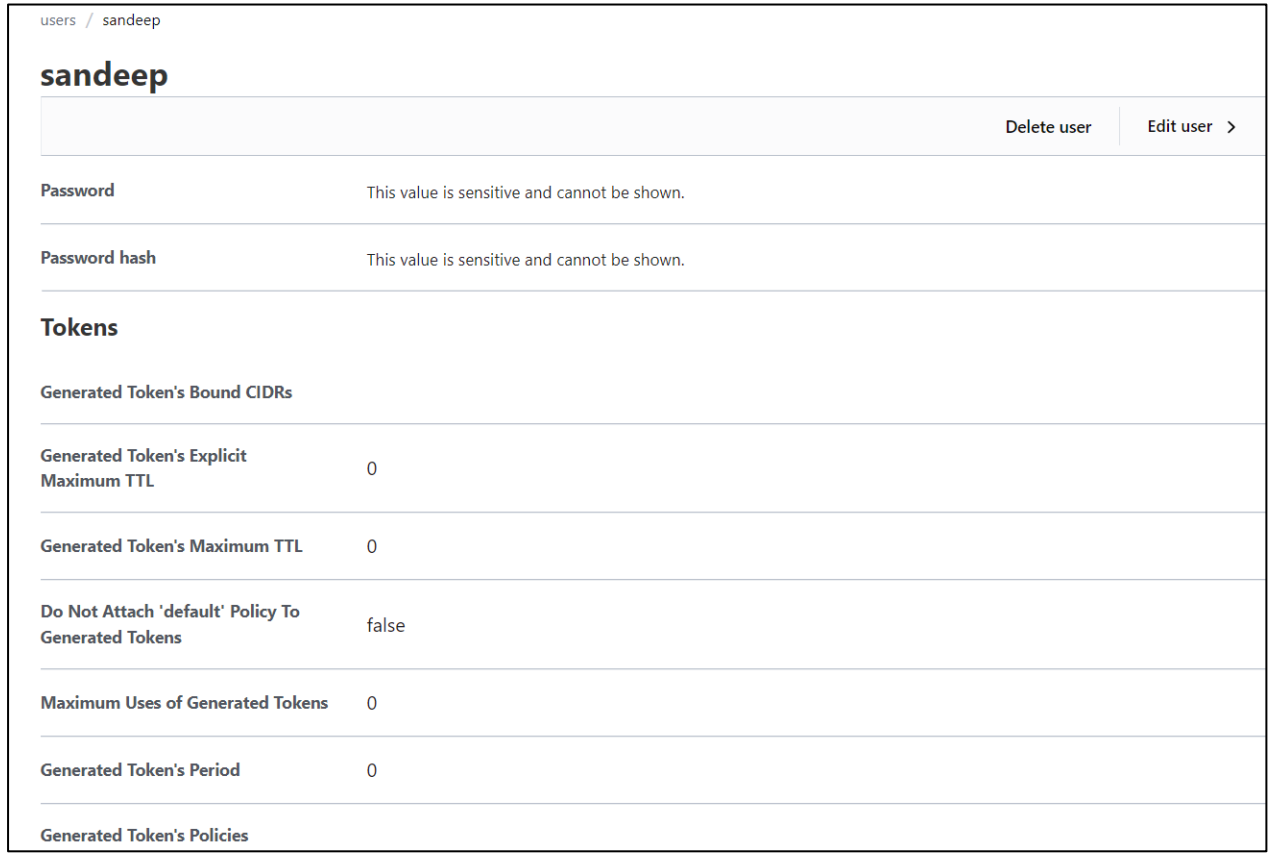
- **Enable userpass**
  - vault auth enable userpass
- **Create a user**
  - vault write auth/userpass/users/sandeep password=password123
- **User Login**
  - vault login -method=userpass username=sandeep password=password123
- **Update User**
  - vault write auth/userpass/users/sandeep password=sandeep123
- **Read the user details**
  - vault read auth/userpass/users/sandeep
- **Delete user**
  - vault delete auth/userpass/users/sandeep
- **Detailed information about the operations available for a specific path in Vault.**
  - vault path-help auth/userpass

Vault Auth Userpass

```
root@ip-172-31-4-218:~# vault write auth/userpass/users/sandeep password=sandeep123
Success! Data written to: auth/userpass/users/sandeep
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault login -method=userpass username=sandeep password=sandeep123
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                    Value
---                    -----
token                  hvs.CAESIB9UvTHgJrVwIOW-XV3ChrDdhZTXYh19zg2nAFv_DFQrGh4KHGh2cy53RWxRRzFQN092a
W16T3JodUlTR0xVblU
token_accessor         MoiDx1u2penoE967RMtni011
token_duration         768h
token_renewable        true
token_policies         ["default"]
identity_policies      []
policies               ["default"]
token_meta_username    sandeep
```

Vault Auth Userpass

Login as a root token



Auth Methods: GitHub

Login as a root in Vault:

Vault login token=token-id

Vault auth list

Vault auth enable github

Vault auth list

export GITHUB_TOKEN=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

Vault write auth/github/config organization=cloudsihmar-org

Vault read auth/github/config

Vault login --method=github token=token-id

Vault path-help auth/github

Vault write auth/github/map/teams/support value=default

Vault write auth/github/map/teams/dev value=dev-policy

Vault read auth/github/map/teams/dev

Auth Methods: AWS

HashiCorp Vault offers several authentication methods (auth methods) to integrate with AWS environments securely.

These methods allow Vault to authenticate applications, machines, or users in AWS using various AWS-specific credentials or metadata. Here's an overview of the main AWS authentication methods in Vault:

- IAM Auth Method
- EC2 Auth Method

Auth Methods: AWS

IAM Auth Method:

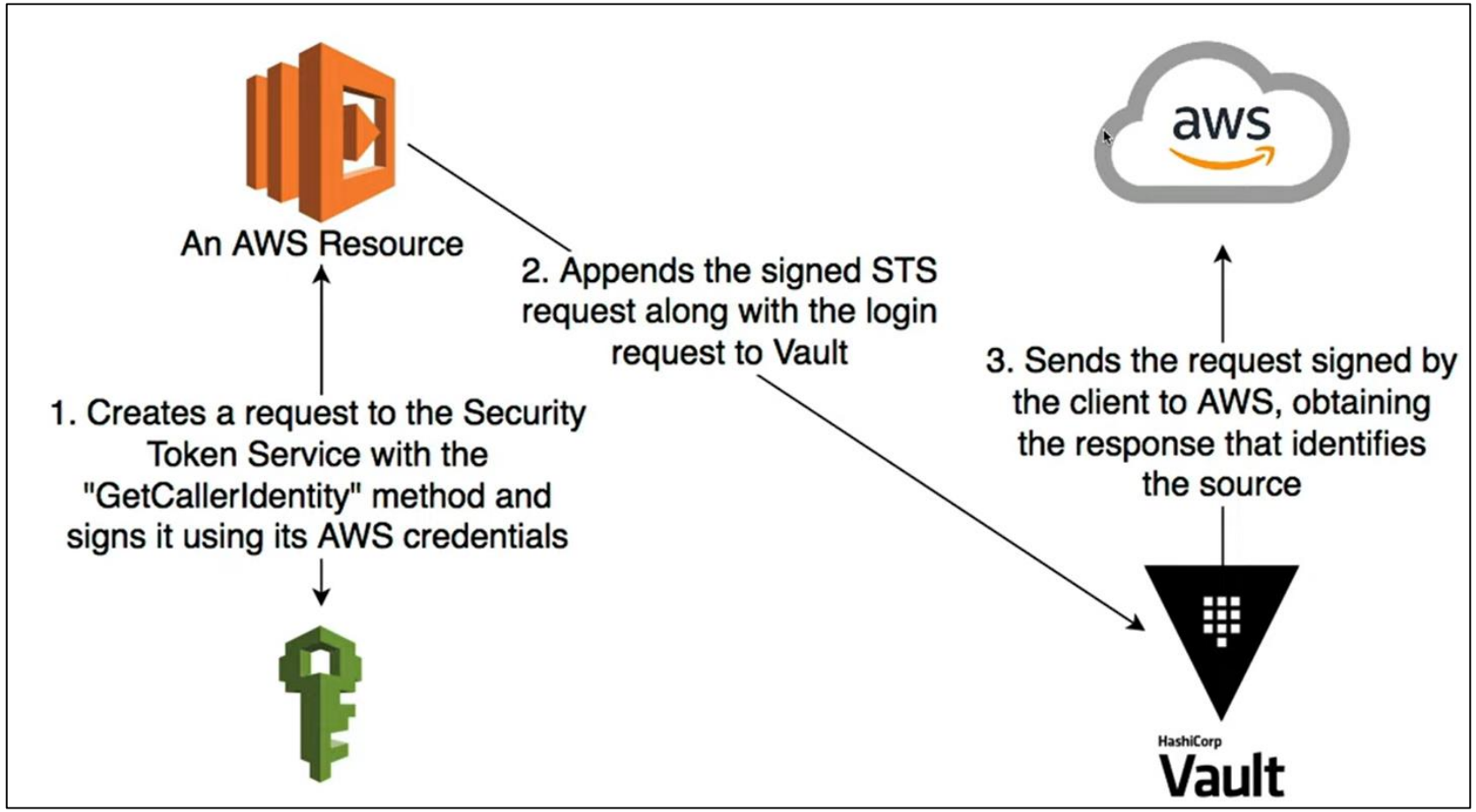
Purpose: This method authenticates AWS EC2 instances, Lambda functions, or any AWS resource that has an associated IAM role.

How it Works: Applications running on AWS instances or services authenticate to Vault by presenting an AWS Identity and Access Management (IAM) role or IAM principal (user or service). Vault verifies the authenticity of the request via the AWS STS (Security Token Service).

Use Case: Useful when you want to allow EC2 instances, Lambda functions, or other AWS resources to authenticate to Vault securely without needing static credentials.

Auth Methods: AWS

IAM Auth Method:



Auth Methods: AWS

IAM Auth Method:

Steps:

- The client retrieves an AWS signed identity document.
- It sends the signed document to Vault.
- Vault verifies the signature using AWS's STS service to validate that the identity is authentic.
- Upon successful verification, Vault returns a Vault token, which the client can use for subsequent interactions with Vault.

Auth Methods: AWS

IAM Auth Method:

Commands
Enable AWS auth method vault auth enable aws # Configure Vault to allow authentication from a specific IAM role vault write auth/aws/config/client secret_key=<AWS_SECRET> access_key=<AWS_ACCESS> # Create a role in Vault that maps to an IAM role vault write auth/aws/role/my-aws-role auth_type=iam bound_iam_principal_arn=arn:aws:iam::<account-id>:role/<role-name> policies=my-policy

Auth Methods: AWS

EC2 Auth Method:

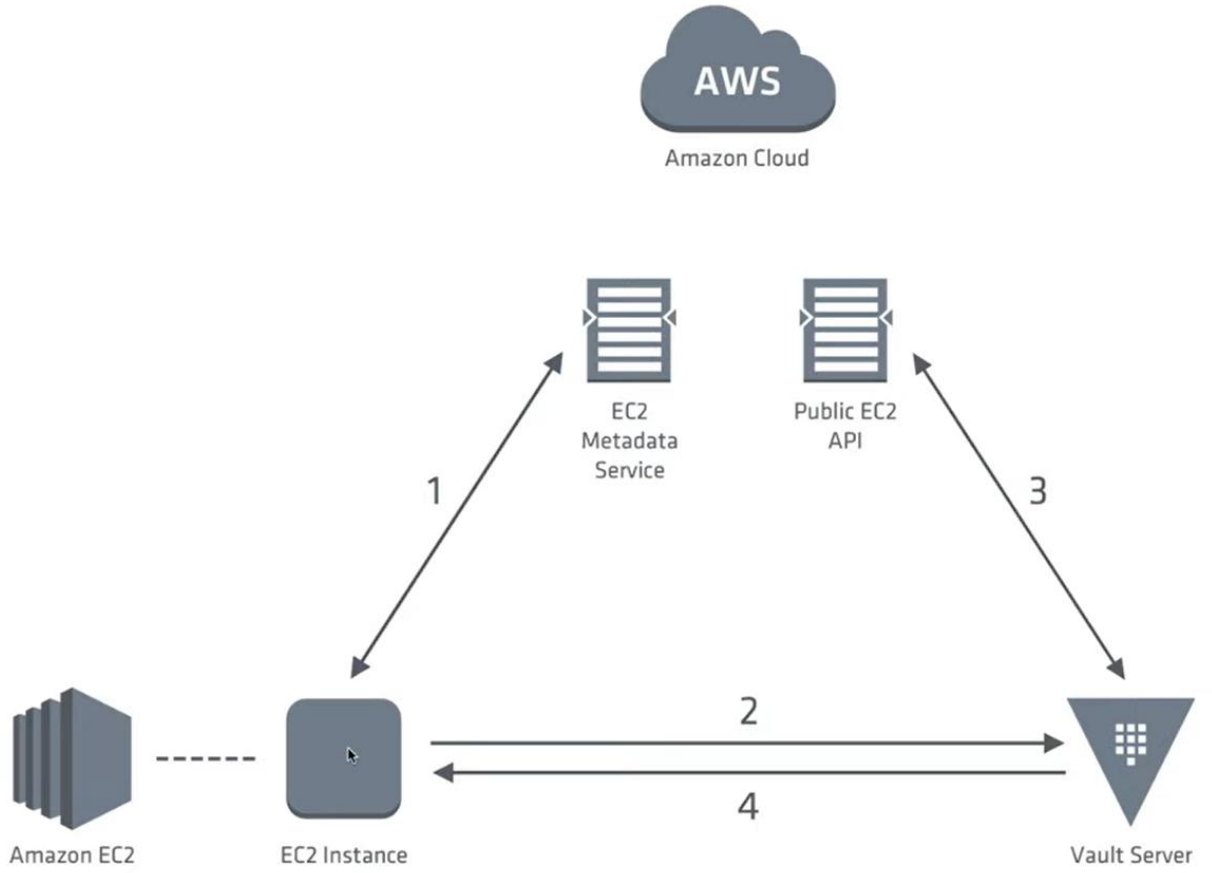
Purpose: This method allows EC2 instances to authenticate to Vault by verifying their identity against AWS metadata.

How it Works: The instance retrieves an identity document and signature from the EC2 instance metadata service. Vault verifies the authenticity of this identity document with AWS.

Use Case: Ideal for EC2 instances that need to authenticate to Vault without requiring static credentials.

Auth Methods: AWS

EC2 Auth Method:



Auth Methods: AWS

EC2 Auth Method:

Steps:

1. The EC2 instance fetches its identity document and signature from the instance metadata service.
2. The instance presents the identity document and signature to Vault.
3. Vault verifies the signature with AWS, ensuring the instance is who it claims to be.
4. Vault grants a token if the instance's metadata matches the Vault configuration.

Auth Methods: AWS

EC2 Auth Method:

Commands
Enable AWS auth method for EC2 instances vault auth enable aws # Configure the AWS auth method for EC2 vault write auth/aws/role/my-ec2-role auth_type=ec2 bound_ami_id=<ami-id> policies=my-policy

Human Authentication Methods

These methods are designed for interactive use by humans who need access to Vault for managing secrets, configurations, or other resources.

Examples of Human Auth Methods in Vault:

Username & Password: A traditional username and password login.

LDAP: Authentication using an LDAP directory like Active Directory, which is often used for employees.

GitHub: Authentication via GitHub OAuth for individuals who are part of an organization.

Okta: Authentication via Okta identity provider.

OIDC (OpenID Connect): Authentication via federated identity systems like Google or Azure AD.

Human Authentication Methods

Use Cases:

- Administrators managing Vault secrets or configuring Vault policies.
- Developers manually accessing secrets during development or troubleshooting.
- Security teams reviewing or auditing configurations.

Human Authentication Methods

Use Cases:

- Administrators managing Vault secrets or configuring Vault policies.
- Developers manually accessing secrets during development or troubleshooting.
- Security teams reviewing or auditing configurations.

System Authentication Methods

These methods are designed for non-human entities such as applications, services, containers, and machines that need access to Vault to retrieve secrets programmatically.

Examples of System Auth Methods in Vault:

AWS IAM or EC2 Auth: Allows AWS resources like EC2 instances, Lambda, or ECS tasks to authenticate using IAM roles or instance metadata.

Kubernetes Auth: Authenticates pods in a Kubernetes cluster based on their service account JWT token.

AppRole: Vault AppRole auth method assigns roles to applications, providing a secure way for apps to authenticate.

TLS Certificates: Authentication using client-side TLS certificates, ideal for mutual TLS (mTLS) setups.

System Authentication Methods

Use Cases:

- Microservices running in Kubernetes accessing secrets through Kubernetes Auth.
- EC2 instances retrieving sensitive API keys via AWS IAM Auth.
- Applications in a CI/CD pipeline using AppRole to retrieve credentials and secrets from Vault.
- Automated backup systems authenticating to Vault to retrieve encryption keys.

App Role

- AppRole is a feature of HashiCorp Vault that provides an authentication method specifically designed for machines or applications, rather than individual users.
- It allows applications to authenticate and gain access to Vault without requiring human interaction.
- This makes AppRole ideal for scenarios where you need machines or microservices to access Vault secrets securely.

App Role

- AppRole is a feature of HashiCorp Vault that provides an authentication method specifically designed for machines or applications, rather than individual users.
- It allows applications to authenticate and gain access to Vault without requiring human interaction.
- This makes AppRole ideal for scenarios where you need machines or microservices to access Vault secrets securely.

When to Use App Role

AppRole is best suited for:

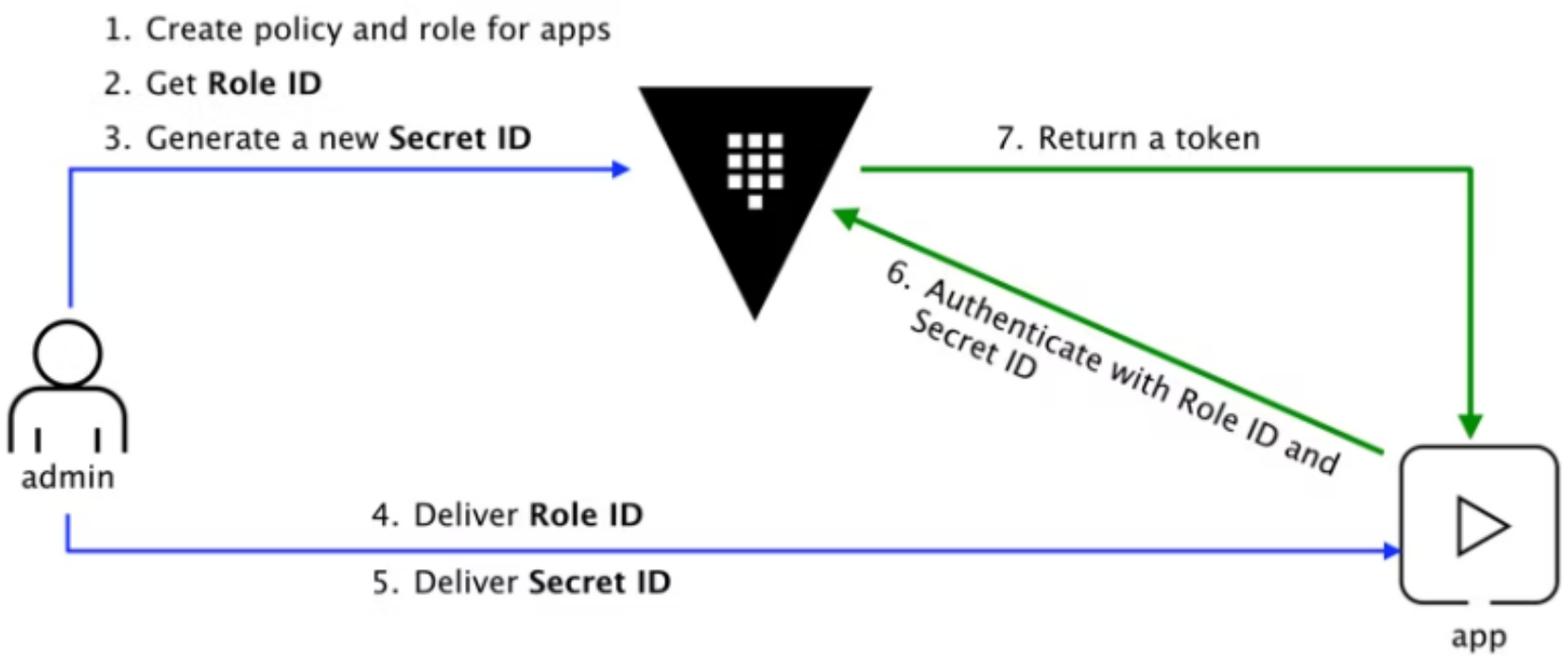
- **Applications:** Where the system needs an automated way to authenticate to Vault without manual intervention.
- **CI/CD Pipelines:** For securely managing secrets in automated build, test, and deploy environments.
- **Microservices:** For containerized applications or microservices needing to fetch secrets from Vault in an automated and secure way.
- **Headless Environments:** Any environment where there's no user interface, and authentication must be done programmatically.

Auth Methods

- Open ID Connect method
- Azure AD with OIDC
- OIDC auth with okta
- AppRole Pull Authentication
- AppRole with Terraform or Chef
- Vault Agent with AWS
- Vault Agent with Kubernetes
- Entities and Group
- Build Own Plugins

Workflow

AppRole is an authentication mechanism within Vault to allow machines or apps to acquire a token to interact with Vault. It uses Role ID and Secret ID for login.



App Role Steps

1. Enable AppRole auth backend
2. Create a role with policy attached
3. Get Role ID and Secret ID
4. Login with Role ID & Secret ID
5. Read secrets using the AppRole token

App Role Steps

1. Enable AppRole auth backend

- vault auth enable approle
- vault auth list

2. Enable some secrets

- vault secrets enable -path=secret/mysql kv
- vault secrets enable -path=secret/postgres kv

3. Create some secrets

- vault kv put secret/mysql/app1 db-name="employee-db" username="admin" password="admin123"
- vault kv put secret/postgres/app1 db-name="product-db" username="admin" password="admin123"

App Role Steps

4. create a policy

vi read-policy.hcl

```
path "secret/mysql/app1"
{
  capabilities = ["read"]
}
```

vault policy write jenkins-policy read-policy.hcl

5. create approle with the policy

vault write auth/approle/role/jenkin-role token_policies="jenkins-policy" token_ttl=1h token_max_ttl=4h secret_id_num_uses=10

vault read auth/approle/role/jenkin-role

vault read auth/approle/role/jenkin-role/role-id

App Role Steps

6. generate the secret id

vault write -f auth/approle/role/jenkin-role/secret-id

vault write auth/approle/login role_id="2c44a6f9-2ad0-27ef-aa65-f425690c9a46" secret_id="3206e085-d7f3-3101-1036-a4b40f4e267a"

7. Login with the above token and test

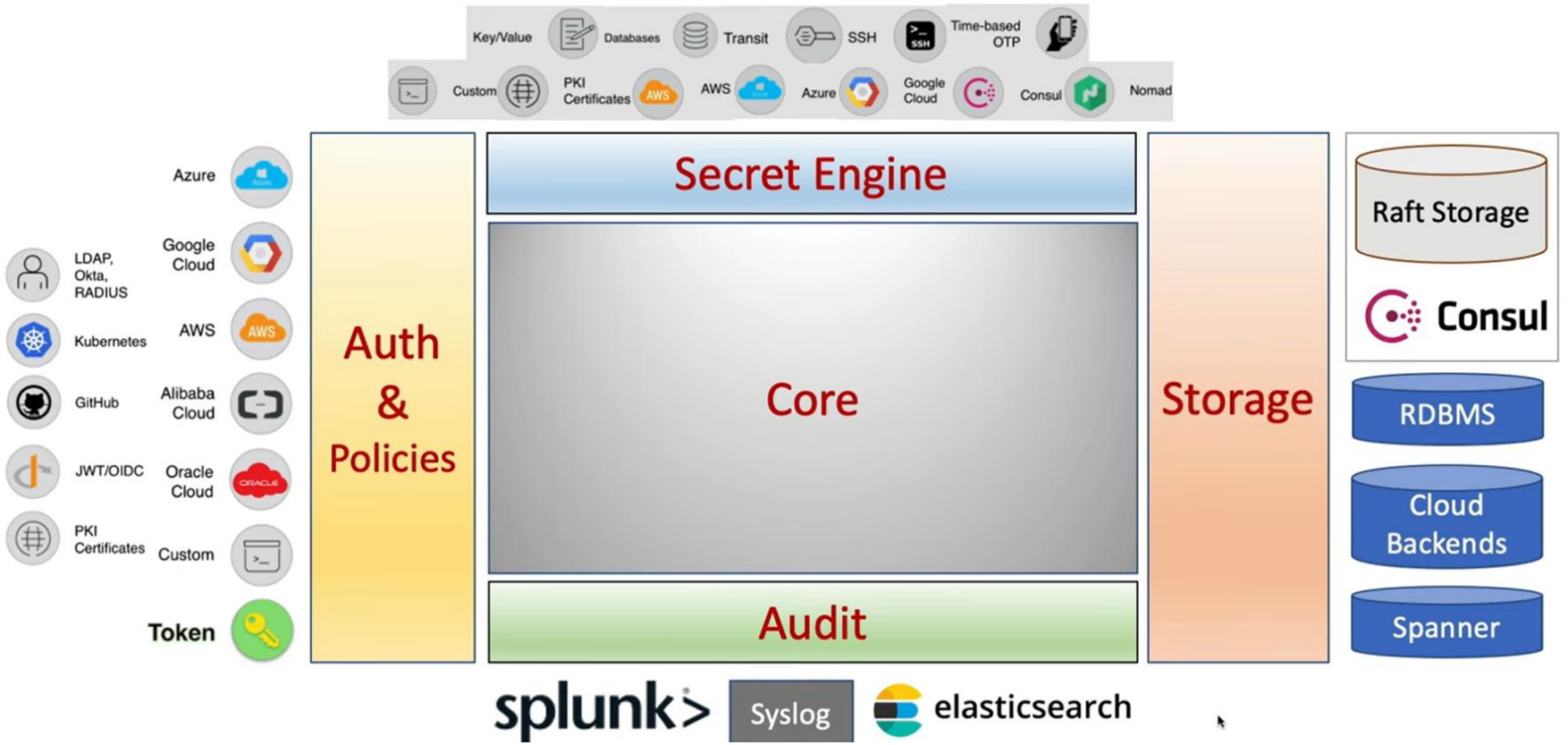
vault login token=token-id

vaule secrets list

vault kv get secret/mysql/app1

Module 4:
Secret Engine

Vault Architecture



Secret Engine

In HashiCorp Vault, a secret engine is a component that is responsible for storing, generating, or managing secrets. Secrets can be anything sensitive, such as API keys, passwords, certificates, and more. Each secret engine is designed to handle a specific type of secret and can be enabled or disabled based on your needs.

- Store, Generate or Encrypt the secrets
- Secrets will be stored in a Path
- Every secret has a configuration
- Secret can be versioned using v2 API
- Every secret has leave TTL
- Secret has metadata
- We can do put, list, get, patch, delete, undelete and destroy

Secret Engine: Features

Key Features of Secret Engines

Types of Secrets: Different secret engines can manage various types of secrets, including static secrets (like passwords) and dynamic secrets (like temporary database credentials).

Secret Management: Secret engines provide functionality for creating, reading, updating, and deleting secrets. They also manage the lifecycle of secrets, including rotation and revocation.

Access Control: You can define policies to control who can access which secrets stored in a secret engine. This enhances security by ensuring that only authorized users or applications can retrieve sensitive data.

Dynamic Secrets: Some secret engines can generate secrets dynamically. For example, the database secret engine can create temporary database credentials that expire after a specified period, reducing the risk of long-lived secrets.

Integration with External Systems: Many secret engines can integrate with external systems, such as databases, cloud providers, and identity providers, to manage secrets more effectively.

Common Types of Secret Engines

KV (Key-Value) Secrets Engine:

- Stores arbitrary secrets as key-value pairs. It's often used for static secrets like configuration settings or API keys.

Database Secrets Engine:

- Dynamically generates database credentials for various database systems (e.g., MySQL, PostgreSQL) that can expire after a defined TTL (Time To Live).

AWS Secrets Engine:

- Manages AWS access keys and can generate temporary IAM credentials for AWS resources.

PKI Secrets Engine:

- Issues and manages X.509 certificates, enabling the creation and signing of certificates for secure communications.

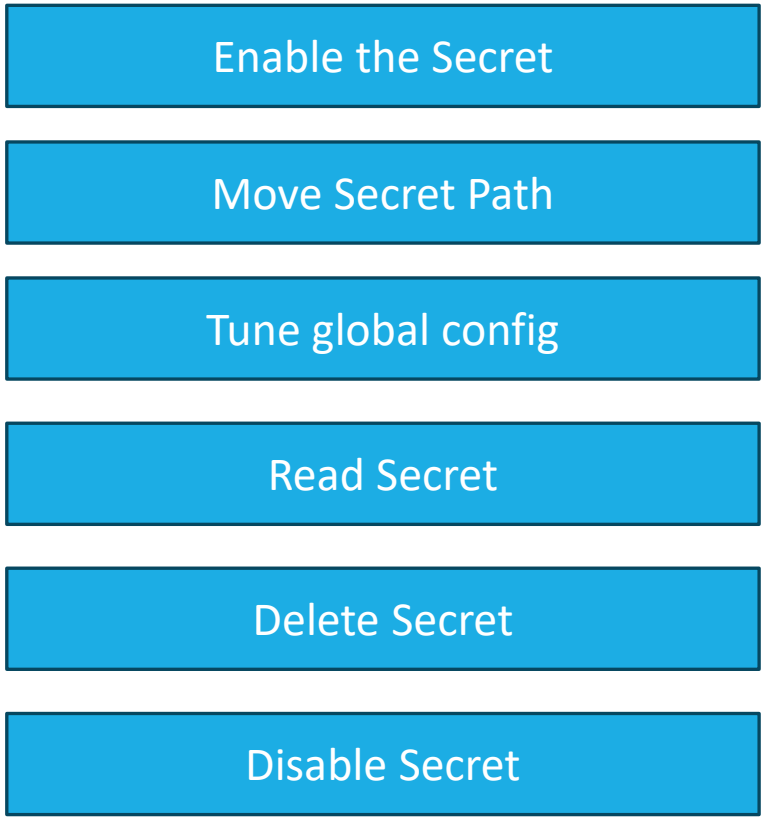
Identity Secrets Engine:

- Manages identity-related information, allowing you to create and manage identities for users and applications.

Transit Secrets Engine:

- Provides encryption and decryption services without storing any data, useful for securing sensitive information before storing it in databases.

Secret Secret lifecycle



List the secrets

Vault secrets list

```
root@ip-172-31-4-218:~# vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_c9023e62    per-token private secret storage
identity/     identity     identity_be8fb72d     identity store
sys/          system       system_95a50e2d       system endpoints used for control, policy and debugging
root@ip-172-31-4-218:~#
```

List the secrets

Cubbyhole Secret Engine:

- The cubbyhole secret engine is primarily used for storing secrets that are tied to a **specific token**. Each token has its own private storage space, and secrets stored here are automatically deleted when the token is revoked.

Identity Secret Engine:

- The identity secret engine manages identities and aliases. It is particularly useful in scenarios where you need to manage users and groups across various authentication methods.

System Secret Engine:

- The system secret engine is used for managing internal operations, such as policies and audit logging.

Cubbyhole Secret

Cubbyhole Secret Engine:

vault token create

vault login <your-token>

vault kv put cubbyhole/my-api-key api_key=abcd1234

vault kv get cubbyhole/my-api-key

vault token revoke <your-token>

vault kv get cubbyhole/my-api-key

Identity Secret Engine

Identity Secret Engine:

	Commands
1. Create an identity	vault write identity/entity name="sandeep"
2. Add an alias	vault write identity/alias name="sandeep_user" mount_accessor=$(vault auth list -format=json \| jq -r '.["userpass/"].accessor') \ canonical_id=$(vault read -field=id identity/entity/name/sandeep)
3. Check the existing identity	vault list identity/entity/id
4. Check the metadata	vault read identity/entity/id/<entity-id>

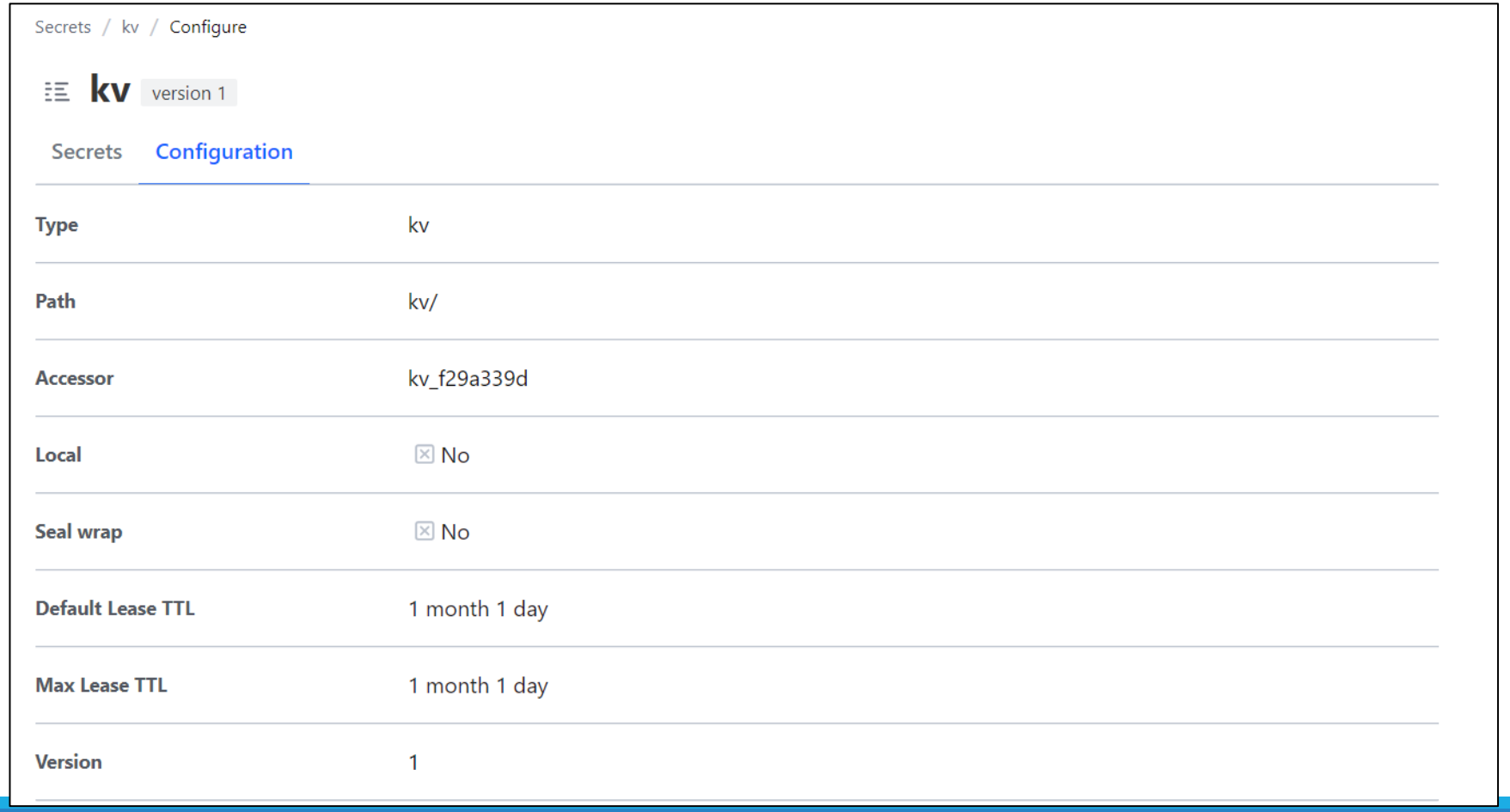
Key Value Secret

Enable and List the KV secret

- vault secrets enable kv
- vault secrets list

```
root@ip-172-31-4-218:~# vault secrets enable kv
Success! Enabled the kv secrets engine at: kv/
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_c9023e62    per-token private secret storage
identity/     identity     identity_be8fb72d     identity store
kv/           kv           kv_f29a339d           n/a
sys/          system       system_95a50e2d       system endpoints used for control, policy and debugging
root@ip-172-31-4-218:~#
```

Key Value Secret



Key Value Secret

Enable a secret with a different path:

vault secrets enable -path=kv-store kv

```
root@ip-172-31-4-218:~# vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_c9023e62    per-token private secret storage
identity/     identity     identity_be8fb72d     identity store
kv-store/     kv           kv_0bcd3760           n/a
kv/           kv           kv_f29a339d           n/a
sys/          system       system_95a50e2d       system endpoints used for control, policy and debugging
root@ip-172-31-4-218:~#
```

Key Value Secret

Create, view and update a secret

vault kv put kv-store/database user=sandeep pass=sandeep@123

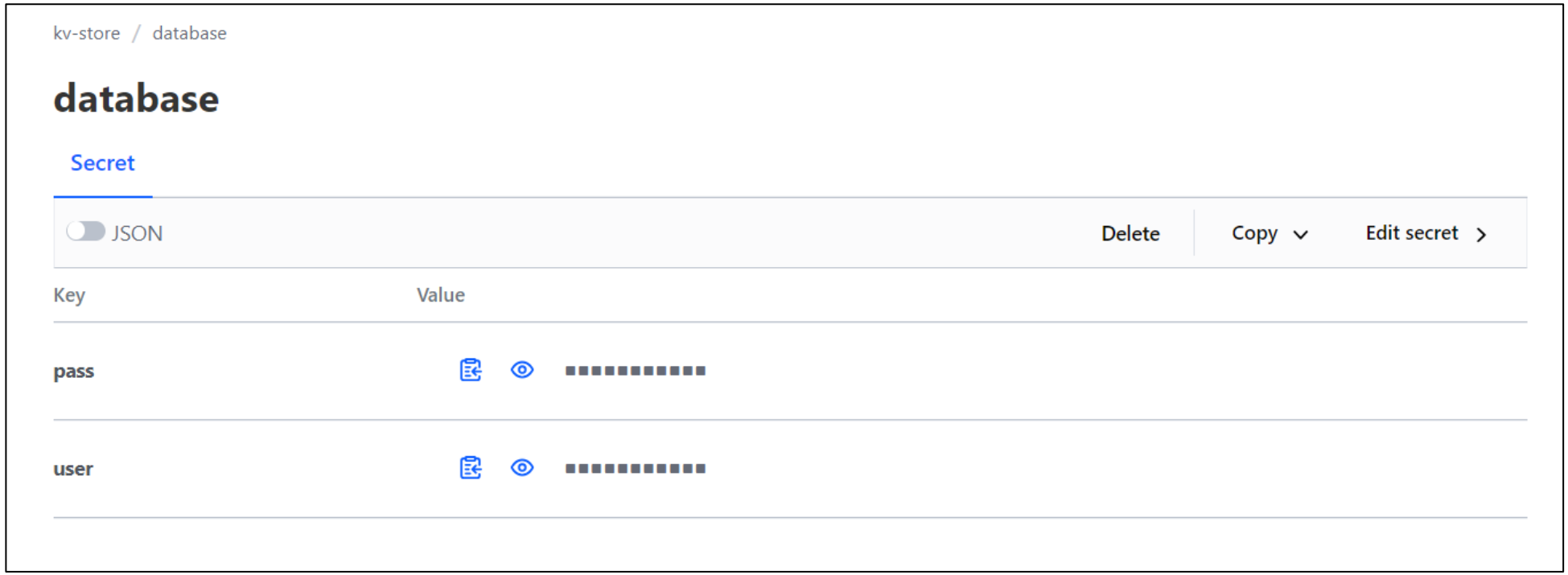
vault kv get kv-store/database

vault kv put kv-store/database user=sandeep pass=sandeep897

```
root@ip-172-31-4-218:~# vault kv put kv-store/database user=sandeep pass=sandeep@123
Success! Data written to: kv-store/database
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault kv get kv-store/database
==== Data ====
Key     Value
---     -----
pass    sandeep@123
user    sandeep
root@ip-172-31-4-218:~# vault kv put kv-store/database user=sandeep pass=sandeep897
Success! Data written to: kv-store/database
root@ip-172-31-4-218:~#
```

Key Value Secret

Create, view and update a secret



Key Value Secret

Delete and disable a secret

vault kv delete kv-store/database

vault secrets disable kv

vault secrets disable kv-store

```
root@ip-172-31-4-218:~# vault kv delete kv-store/database
Success! Data deleted (if it existed) at: kv-store/database
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_c9023e62    per-token private secret storage
identity/     identity     identity_be8fb72d     identity store
kv-store/     kv           kv_0bcd3760           n/a
kv/           kv           kv_f29a339d           n/a
sys/          system       system_95a50e2d       system endpoints used for control, policy and debugging
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault secrets disable kv
Success! Disabled the secrets engine (if it existed) at: kv/
root@ip-172-31-4-218:~# vault secrets disable kv-store
Success! Disabled the secrets engine (if it existed) at: kv-store/
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_c9023e62    per-token private secret storage
identity/     identity     identity_be8fb72d     identity store
sys/          system       system_95a50e2d       system endpoints used for control, policy and debugging
root@ip-172-31-4-218:~#
```

Key Value Secret: Versioning

Versioning the secret

vault secrets enable -version=2
-path=new-kv-store kv

```
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault kv put new-kv-store/users username=admin password=pass123
===== Secret Path =====
new-kv-store/data/users

======= Metadata =======
Key                Value
---                -----
created_time       2024-10-09T09:24:30.11382762Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            1
root@ip-172-31-4-218:~# vault kv put new-kv-store/users username=admin password=pass456
===== Secret Path =====
new-kv-store/data/users

======= Metadata =======
Key                Value
---                -----
created_time       2024-10-09T09:24:34.192671783Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            2
root@ip-172-31-4-218:~# vault kv put new-kv-store/users username=admin password=pass678
===== Secret Path =====
new-kv-store/data/users

======= Metadata =======
Key                Value
---                -----
created_time       2024-10-09T09:24:38.712300356Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            3
```

Key Value Secret: Versioning

Versioning the secret

vault kv get new-kv-store/users

vault kv get -version=2 new-kv-store/users

```
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault kv get new-kv-store/users
===== Secret Path =====
new-kv-store/data/users

======= Metadata =======
Key                Value
---                -----
created_time       2024-10-09T09:24:38.712300356Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            3

====== Data ======
Key         Value
---         -----
password    pass678
username    admin
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault kv get -version=2 new-kv-store/users
===== Secret Path =====
new-kv-store/data/users

======= Metadata =======
Key                Value
---                -----
created_time       2024-10-09T09:24:34.192671783Z
custom_metadata    <nil>
deletion_time      n/a
destroyed          false
version            2

====== Data ======
Key         Value
---         -----
password    pass456
username    admin
```

Key Value Secret: Metadata

Metadata of the secret

vault kv metadata get new-kv-store/users

```
root@ip-172-31-4-218:~# vault kv metadata get new-kv-store/users
====== Metadata Path ======
new-kv-store/metadata/users

========== Metadata ==========
Key                     Value
---                     -----
cas_required            false
created_time            2024-10-09T09:24:30.11382762Z
current_version         3
custom_metadata         <nil>
delete_version_after    0s
max_versions            0
oldest_version          0
updated_time            2024-10-09T09:24:38.712300356Z

====== Version 1 ======
Key              Value
---              -----
created_time     2024-10-09T09:24:30.11382762Z
deletion_time    n/a
destroyed        false

====== Version 2 ======
Key              Value
---              -----
created_time     2024-10-09T09:24:34.192671783Z
deletion_time    n/a
destroyed        false

====== Version 3 ======
Key              Value
---              -----
created_time     2024-10-09T09:24:38.712300356Z
deletion_time    n/a
destroyed        false
```

Key Value Secret: Version Deletion

Version delete and destroy

vault kv delete -versions=1 new-kv-store/users

vault kv destroy -versions=1 new-kv-store/users

```
root@ip-172-31-4-218:~# vault kv destroy -versions=1 new-kv-store/users
Success! Data written to: new-kv-store/destroy/users
root@ip-172-31-4-218:~#
root@ip-172-31-4-218:~# vault kv metadata get new-kv-store/users
====== Metadata Path ======
new-kv-store/metadata/users

========== Metadata ==========
Key                     Value
---                     -----
cas_required            false
created_time            2024-10-09T09:24:30.11382762Z
current_version         3
custom_metadata         <nil>
delete_version_after    0s
max_versions            0
oldest_version          0
updated_time            2024-10-09T09:24:38.712300356Z

====== Version 1 ======
Key              Value
---              -----
created_time     2024-10-09T09:24:30.11382762Z
deletion_time    2024-10-09T09:35:14.048566231Z
destroyed        true

====== Version 2 ======
Key              Value
---              -----
created_time     2024-10-09T09:24:34.192671783Z
deletion_time    n/a
destroyed        false

====== Version 3 ======
Key              Value
---              -----
created_time     2024-10-09T09:24:38.712300356Z
deletion_time    n/a
destroyed        false
```

Key Value Secret: Detailed

Detailed Secrets

vault secrets list -detailed

```
root@ip-172-31-4-218:~# vault secrets list -detailed
Path            Plugin      Accessor            Default TTL    Max TTL    Force No Cache    Replication    Seal Wrap    External Entropy Access    Options          Description
                            UUID                                          Version    Running Version         Running SHA256    Deprecation Status
----            ------      --------            -----------    -------    --------------    -----------    ---------    ----------------------    -------          -----------
                            ----                                          -------    ---------------         --------------    ------------------
cubbyhole/      cubbyhole   cubbyhole_c9023e62  n/a            n/a        false             local          false        false                     map[]            per-token private se
cret storage                469eefab-99cd-1e7b-ba80-e3d2df9871f5  n/a        v1.17.6+builtin.vault    n/a              n/a
identity/       identity    identity_be8fb72d   system         system     false             replicated     false        false                     map[]            identity store
                            f771f68b-a217-a637-8525-e1d73764af3e  n/a        v1.17.6+builtin.vault    n/a              n/a
kv-store/       kv          kv_0a23da36         system         system     false             replicated     false        false                     map[version:2]   n/a
                            d06b3585-1a6e-7c04-597c-b5cf61bfafa2  n/a        v0.19.0+builtin          n/a              supported
kv/             kv          kv_07ff115a         system         system     false             replicated     false        false                     map[]            n/a
                            d0dfaea3-02b7-b821-316a-27c15d82fc8c  n/a        v0.19.0+builtin          n/a              supported
new-kv-store/   kv          kv_af2023cd         system         system     false             replicated     false        false                     map[version:2]   n/a
                            f18847c8-7583-8f63-2699-d20448e845b2  n/a        v0.19.0+builtin          n/a              supported
sys/            system      system_95a50e2d     n/a            n/a        false             replicated     true         false                     map[]            system endpoints use
d for control, policy and debugging    2d5055fa-371f-9bf6-0463-9ac2e087a788  n/a        v1.17.6+builtin.vault    n/a              n/a
root@ip-172-31-4-218:~#
```

Dynamic Secrets

Dynamic secrets in HashiCorp Vault are secrets that are generated on-the-fly when a client requests them, rather than being stored statically in Vault. This approach provides a number of benefits, particularly in terms of security and manageability. Here's an overview of dynamic secrets:

Key Features of Dynamic Secrets

On-Demand Generation: Dynamic secrets are created in real time when they are requested. For example, when a user requests AWS credentials, Vault generates a new set of temporary AWS access keys with a defined lease duration.

Temporary Credentials: These secrets typically have a limited lifespan (or lease duration). Once they expire, they are automatically revoked by Vault. This reduces the risk associated with long-lived static credentials.

Access Control: Because dynamic secrets are generated based on the permissions assigned to the requesting identity, access can be tightly controlled. Users can get unique credentials that only they can use.

Revocation: Dynamic secrets can be easily revoked by Vault at any time, which helps mitigate the risk of leaked credentials. When a dynamic secret is revoked, all clients using that secret lose access.

Auditing: Vault can keep track of which dynamic secrets were issued and when, allowing for better auditing and tracking of secret usage.

Dynamic Secrets

Dynamic Secrets Use Cases:

AWS Secrets:

Vault can dynamically generate AWS IAM credentials for users or applications. When a client requests AWS credentials, Vault creates an IAM user with specific permissions, provides the access key and secret key, and sets a lease duration. When the lease expires, the IAM user is deleted.

Database Credentials:

Similar to AWS, Vault can generate database credentials dynamically for various databases (like MySQL, PostgreSQL, etc.). When an application needs access, Vault creates a user in the database with a limited set of permissions and a temporary password.

SSH Credentials:

Vault can also provide temporary SSH credentials to users, allowing them to access servers without having to manage long-term SSH keys.

AWS Dynamic Secrets

1. Enable and list the secret

- vault secrets enable -path=aws aws

2. Write the configuration

```
vault write aws/config/root \
access_key=AKIAZ7FSO3B5UC7CRGMU \
secret_key=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 \
region=ap-south-1
```

AWS Dynamic Secrets

3. Create a policy

Policy.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1426528957000",
      "Effect": "Allow",
      "Action": [
        "ec2:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

AWS Dynamic Secrets

4. Read

vault read aws/creds/my-ec2-role

vault lease renew aws/creds/my-ec2-role/gIQbrirExUatiECZqahwaKsL

vault lease revoke aws/creds/my-ec2-role/gIQbrirExUatiECZqahwaKsL

vault secret disable aws