O'REILLY®

Hashicorp Certified: Vault Associate Certification Crash Course

Vault Associate 002

About the trainer

bmuschko

bmuschko

bmuschko.com

AUTOMATED ASCENT

automatedascent.com

Certification Exam

Objectives, Curriculum, Prerequisites, Learning Resources

Exam Objectives

“Basic understanding of concepts and skills associated with open source HashiCorp Vault.”

The certification program allows users to demonstrate their competence in a multiple choice test.

https://www.hashicorp.com/certification/vault-associate

The Curriculum

1	**Compare authentication methods**
1a	Describe authentication methods
1b	Choose an authentication method based on use case
1c	Differentiate human vs. system auth methods

2	**Create Vault policies**
2a	Illustrate the value of Vault policy
2b	Describe Vault policy syntax: path
2c	Describe Vault policy syntax: capabilities
2d	Craft a Vault policy based on requirements

3	**Assess Vault tokens**
3a	Describe Vault token
3b	Differentiate between service and batch tokens. Choose one based on use-case
3c	Describe root token uses and lifecycle
3d	Define token accessors
3e	Explain time-to-live
3f	Explain orphaned tokens
3g	Create tokens based on need

4	**Manage Vault leases**
4a	Explain the purpose of a lease ID
4b	Renew leases
4c	Revoke leases

The Curriculum

5	**Compare and configure Vault secrets engines**
5a	Choose a secret method based on use case
5b	Contrast dynamic secrets vs. static secrets and their use cases
5c	Define transit engine
5d	Define secrets engines

6	**Utilize Vault CLI**
6a	Authenticate to Vault
6b	Configure authentication methods
6c	Configure Vault policies
6d	Access Vault secrets
6e	Enable Secret engines
6f	Configure environment variables

7	**Utilize Vault UI**
7a	Authenticate to Vault
7b	Configure authentication methods
7c	Configure Vault policies
7d	Access Vault secrets
7e	Enable Secret engines

8	**Be aware of the Vault API**
8a	Authenticate to Vault via Curl
8b	Access Vault secrets via Curl

The Curriculum

9	**Explain Vault architecture**
9a	Describe the encryption of data stored by Vault
9b	Describe cluster strategy
9c	Describe storage backends
9d	Describe the Vault agent
9e	Describe secrets caching
9f	Be aware of identities and groups
9g	Describe Shamir secret sharing and unsealing
9h	Be aware of replication
9i	Describe seal/unseal
9j	Explain response wrapping
9k	Explain the value of short-lived, dynamically generated secrets

10	**Explain encryption as a service**
10a	Configure transit secret engine
10b	Encrypt and decrypt secrets
10c	Rotate the encryption key

Candidate Prerequisites

Basic terminal skills

Basic understanding of on premises and cloud architecture

Basic level of security understanding

Preparing for the Exam

Documentation is not allowed during the test

- Study Guide: https://developer.hashicorp.com/vault/tutorials/associate-cert/associate-study
- Exam Review: https://developer.hashicorp.com/vault/tutorials/associate-cert/associate-review
- Sample Questions: https://developer.hashicorp.com/vault/tutorials/associate-cert/associate-questions

Interacting with Vault

Introduction, Installation, Vault UI & API

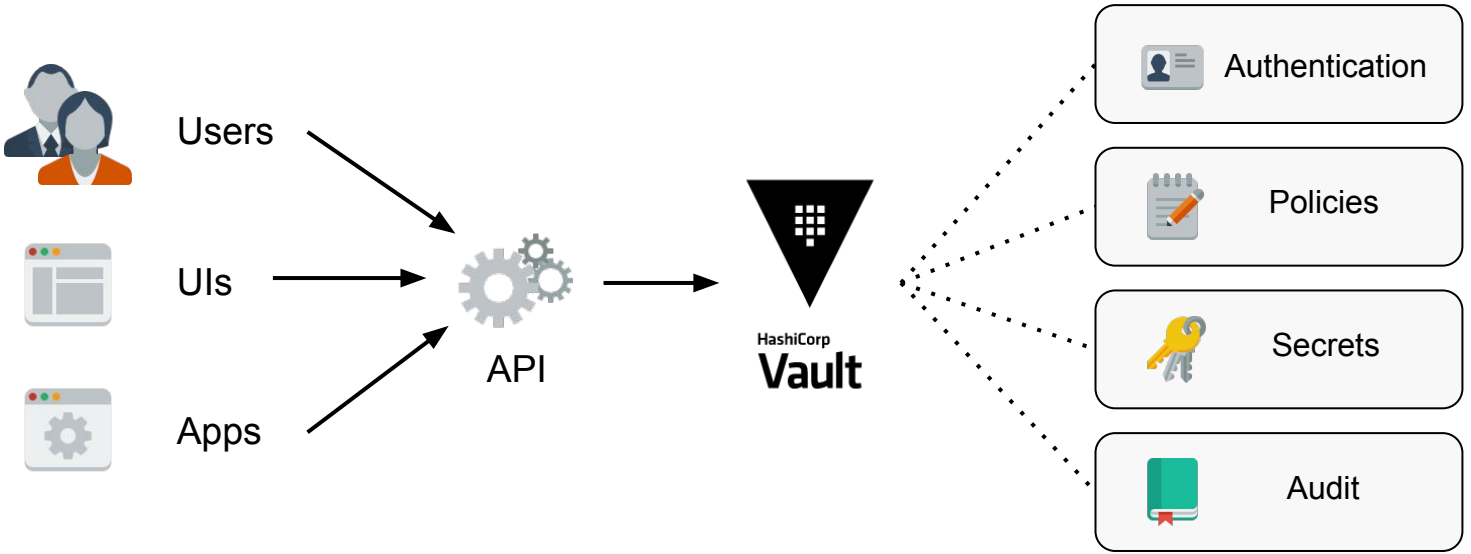
What is Vault?

Centralized secrets manager with a client and server component

- Manages static (a hard-coded value) and dynamic secrets (generated on-the-fly)
- A secret could be an API key, a password, or a SSN
- Exists as open source solution or in the cloud as a commercial offering (Vault Cloud)
- You can interact with Vault with the CLI, the UI, or directly with the API

High-Level Vault Architecture

Common entry point for all communication is the API



Installing Vault

Easy to install on all operating systems

- Manual installation
  - Download ZIP file containing the pre-compiled binary
  - Add binary to `PATH` environment variable

- Using a package manager
  - Available via Homebrew, Chocolatey, and `apt-get`
  - Takes care of adding binary to terminal

Verifying and Using Vault

The `vault` executable is the main entry point

```
$ vault version
Vault v1.12.1 ...
```

```
$ vault -help
Usage: vault <command> [args]

Common commands:
    read        Read data and retrieves secrets
    write       Write data, configuration, and secrets
    delete      Delete secrets and configuration
    list        List data or secrets
    login       Authenticate locally
    agent       Start a Vault agent
    server      Start a Vault server
    status      Print seal and HA status
    unwrap      Unwrap a wrapped secret

...
```

CLI Autocompletion

Opt-in feature for commands and flag

```
$ vault -autocomplete-install

$ vault a<tab>
agent  audit  auth

$ vault auth tune -d<tab>
-default-lease-ttl  -description  -disable-redirects
```

Using Development Mode

Server can be run on local machine for dev purposes

- Runs on localhost without SSL (only allows HTTP access)
- The user interface is enabled by default
- Runs a key-value secrets store in memory (loses data upon restart)
- Starts unsealed with a single unseal key (allows access to decrypted data by clients)

Starting Developing Server

Start up and use root token to log in

```
$ vault server -dev
...
Unseal Key: 6WYdi1rhIv+FsfvoV7fL2grz0wwk1ChAfGXWjguC2Fk=
Root Token: hvs.kVrw0a8LT4MzDj9xxrFmDhPN

$ addr='http://127.0.0.1:8200'
$ token=hvs.kVrw0a8LT4MzDj9xxrFmDhPN

$ vault login -address=$addr $token
```

Vault Environment Variables

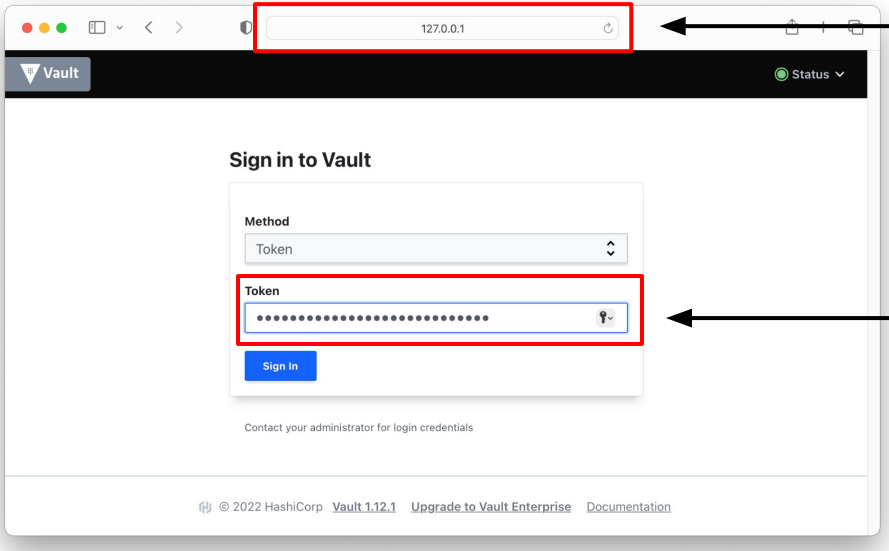
Evaluated automatically by `vault` command

```
$ export VAULT_ADDR='http://127.0.0.1:8200'
$ export VAULT_TOKEN=hvs.kVrw0a8LT4MzDj9xxrFmDhPN
```

Just two of the standard Vault environment variables

Interacting with the UI

Accessing via Vault address + the `/ui` context path



`http://127.0.0.1:8200/ui`

Enter the root token

Interacting with the API

The UI and CLI use the RESTful API

```
$ curl -H "X-Vault-Token: $VAULT_TOKEN" -X GET
  $VAULT_ADDR/v1/sys/host-info

$ curl -H "X-Vault-Token: $VAULT_TOKEN" -X LIST
  $VAULT_ADDR/v1/secret

$ curl -H "X-Vault-Token: $VAULT_TOKEN" -X GET
  $VAULT_ADDR/v1/secret/foo
```

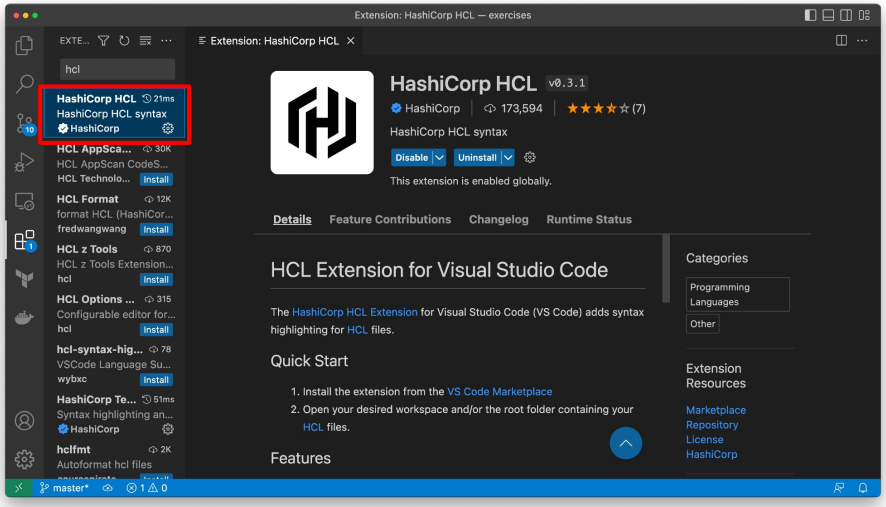
Get host information

List all secrets

Get information about a specific secret

IDE Integration with VSCode

Search for “HashiCorp HCL” in the Extensions



EXERCISE

Installing Vault and
Interacting with it

Q & A

Managing Authentication Methods

Use Cases, Configuration, Human vs. System Authentication Methods

What are Authentication Methods?

Verifies identity when interacting with the Vault API

- Different methods are allowed and can be referenced from internal (Vault native) or external sources (e.g. AWS IAM, LDAP).
- Authentication methods are provided by plug-ins.
- The end goal is to retrieve a *token* stored at `~/.vault-token`.
- Subsequent calls to the Vault API use the token to authenticate.

Choosing a Suitable Method

Ask yourself the following question to determine fitting method

- **Who** is going to access Vault? Is an internal or external human user? Are you planning to use Vault from a machine?
- **How** does the authentication process look like? Interactive or by providing a value.
- **What** kind of authentication method has been established in the past? LDAP vs. username/password vs. cloud provider

Vault Native Methods

Built into Vault and doesn't require reaching out to external source

- **Userpass:** Meant for human users as a way to enter username/password credentials interactively.
- **AppRole:** Meant for applications that provide credentials programmatically. Consists of RoleID and SecretID.
- **Token:** Can be used by human or program. A single key sent with API request in header.

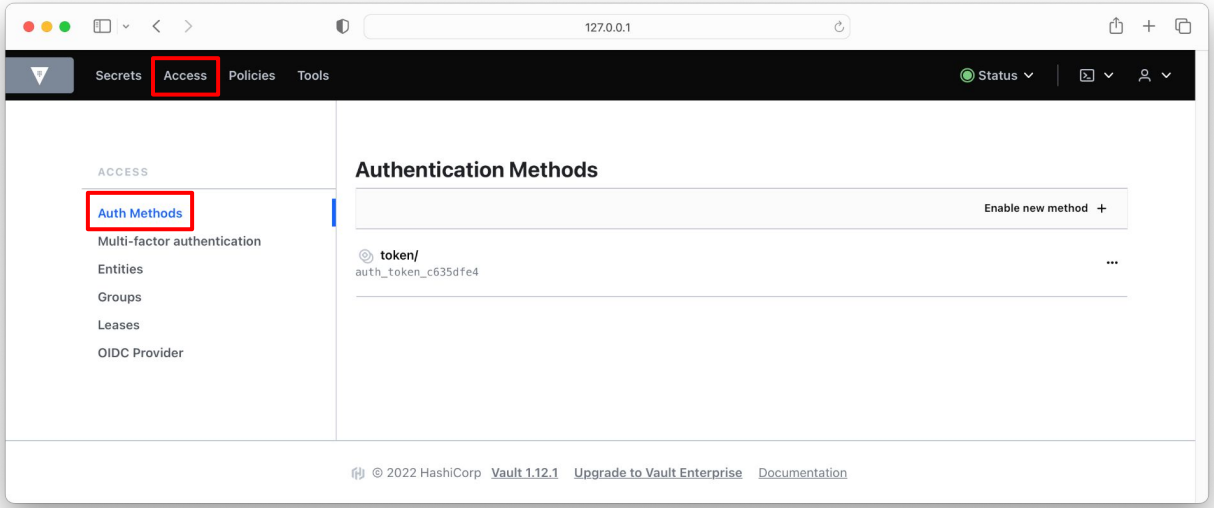
Listing Methods from CLI

By default, only the token auth method is enabled

```
$ vault auth list
Path      Type     Accessor                Description                Version
----      ----     --------                -----------                -------
token/    token    auth_token_c635dfe4     token based credentials    n/a
```

Listing Methods from UI

Access > Auth Methods



Enabling a Method

Providing a custom path is optional (see docs)

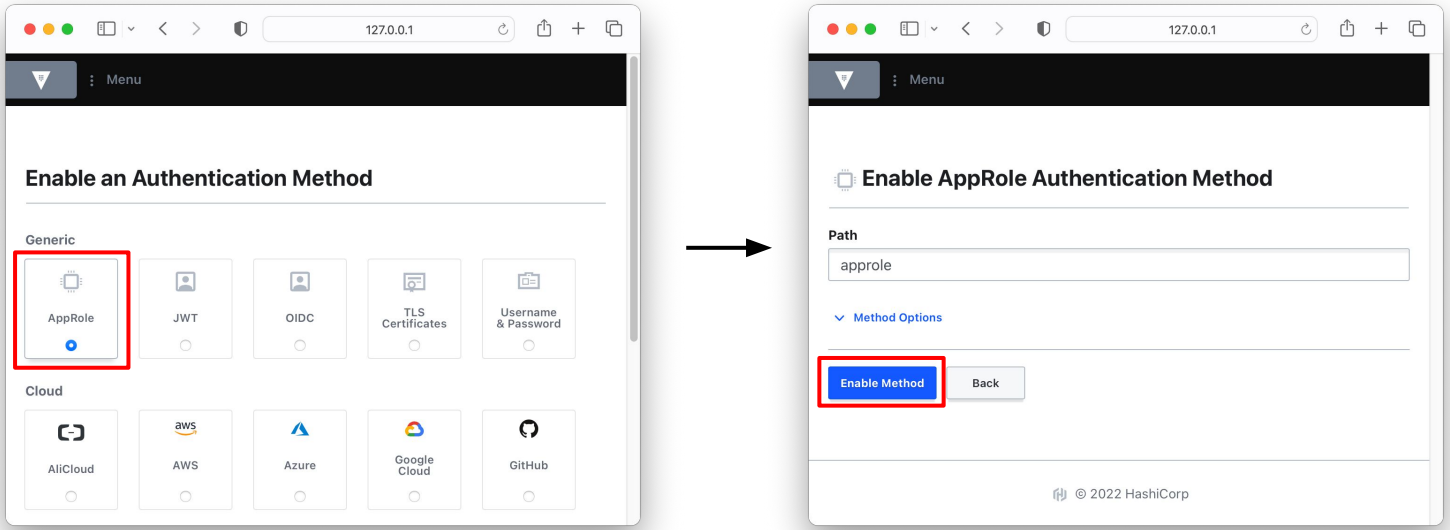
```
$ vault auth enable userpass
Success! Enabled userpass auth method at: userpass/

$ vault auth enable -path=service approle
Success! Enabled approle auth method at: service/
```

Derives path from method type

Uses the custom path provided

Enabling a Method from UI

Access > Auth Methods > Enable new method



Tuning a Method from CLI

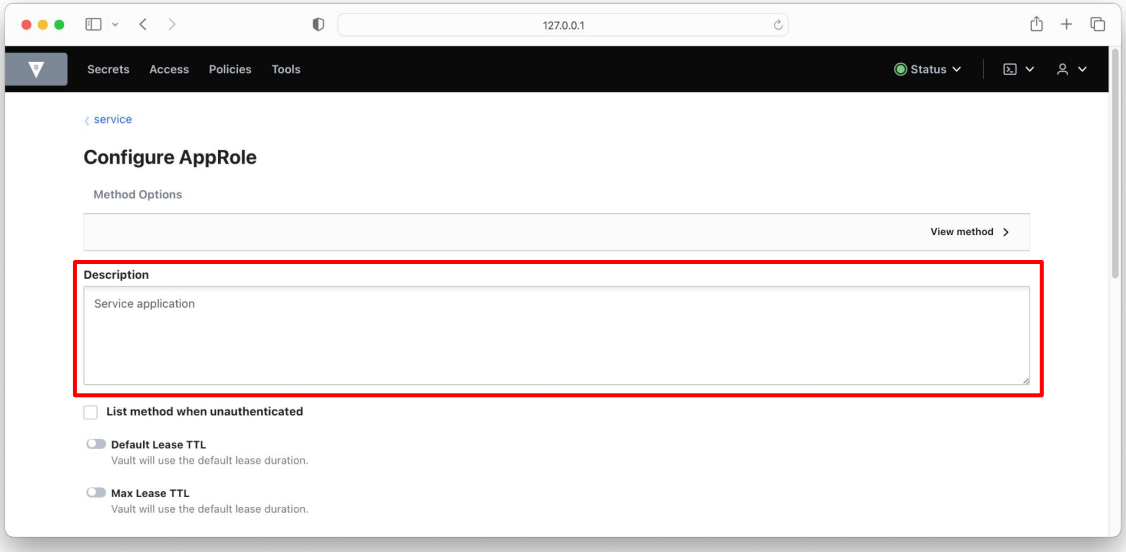
Changes the settings of an existing method

```
$ vault auth tune -description="Service application" service/
Success! Tuned the auth method at: service/
```

Sets the description

Tuning a Method from UI

Access > Auth Methods > View configuration > Configure



Getting Path Information from CLI

Renders supported paths for a backend

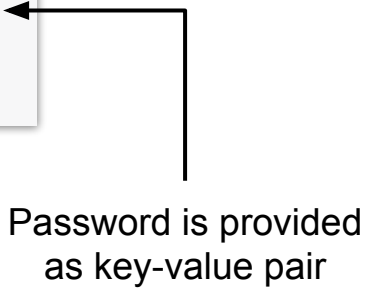
```
$ vault path-help auth/userpass
...
    ^login/(?P<username>\w(([\w-.]+)?\w)?)$
        Log in with a username and password.

    ^users/(?P<username>\w(([\w-.]+)?\w)?)$
        Manage users allowed to authenticate.

    ^users/(?P<username>\w(([\w-.]+)?\w)?)/password$
        Reset user's password.

    ^users/(?P<username>\w(([\w-.]+)?\w)?)/policies$
        Update the policies associated with the username.

    ^users/?$
        Manage users allowed to authenticate.
```

Requires the type of path, in this case `auth`

Creating a User from CLI

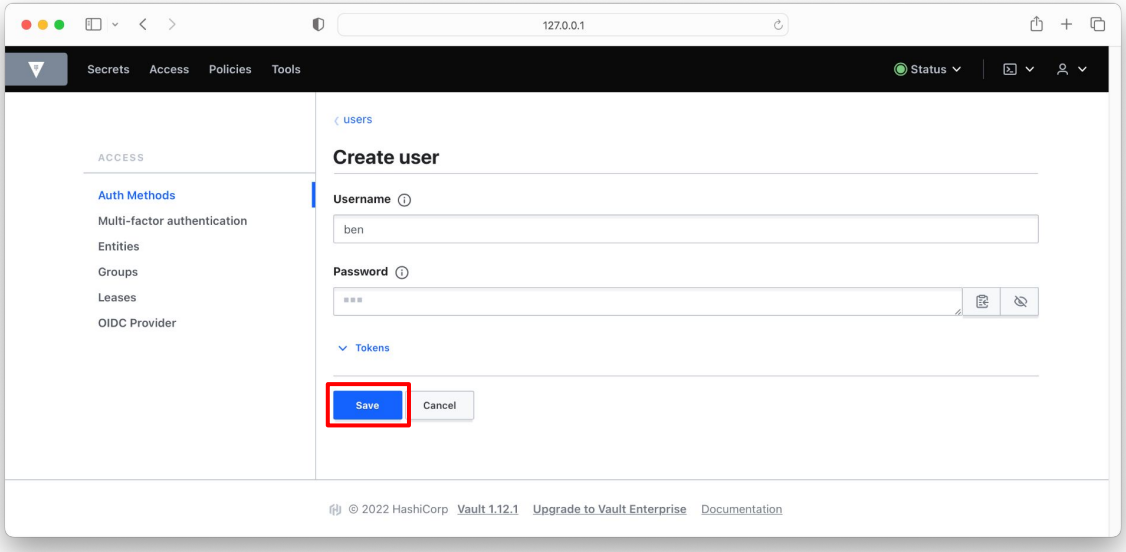
Add username to path & provide password with param

```
$ vault write auth/userpass/users/ben password=123
Success! Data written to: auth/userpass/users/ben
```



Creating a User from UI

Access > Auth Methods > Users > Create user



Creating an AppRole from CLI

Provide the role name as parameter

```
$ vault write auth/approle/role/batch role_name="batch"
Success! Data written to: auth/approle/role/batch

$ vault list auth/approle/role
Keys
----
batch
```

From the Vault UI:

> The Vault UI only supports configuration for this authentication method. For management, the **API or CLI** should be used.

EXERCISE

Managing
Authentication
Methods

Using a User from CLI

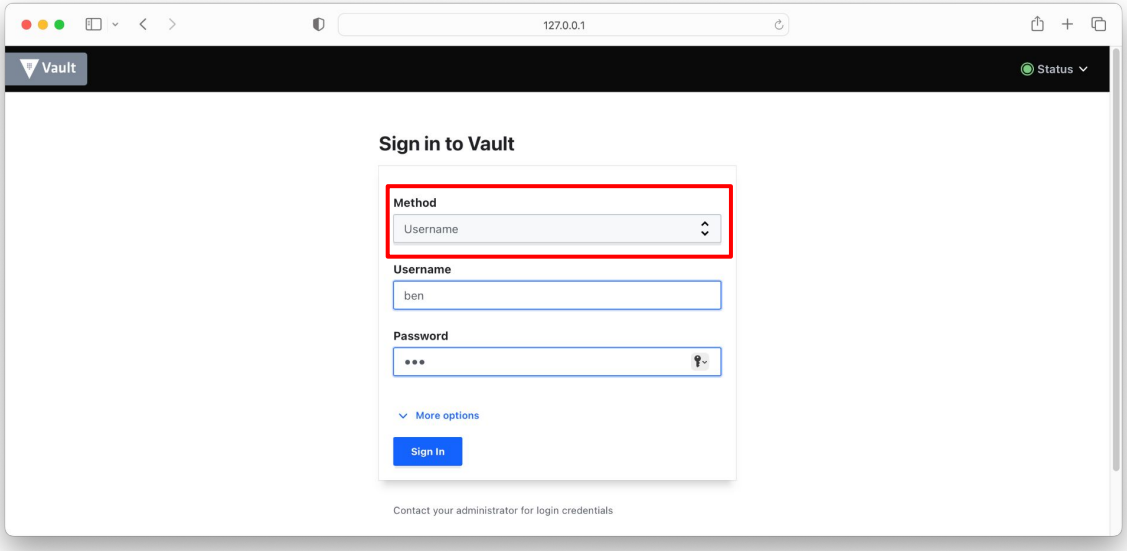
Use the `login` command and provide username param

```
$ vault login -method=userpass username=ben
Password (will be hidden):
...

Key                    Value
---                    -----
token                  hvs.CAESIIZ_1g...
token_accessor
M2SX78GxvpijWpulzXEPXNdM
token_duration         768h
token_renewable        true
token_policies         ["default"]
identity_policies      []
policies               ["default"]
token_meta_username    ben
```

You can use the token for API calls going forward

Using a User from UI

Log out of dashboard and sign back in with username method



Using an AppRole from CLI

Read RoleID, generate SecretID, and write information

```
$ vault read auth/approle/role/batch/role-id
Key        Value
---        -----
role_id    0a437428-f796-f336-8013-2a248f91da7b

$ vault write -force auth/approle/role/batch/secret-id
Key                   Value
---                   -----
secret_id             9962a7fc-391b-a509-6741-0fd5f3c0096d

$ vault write auth/approle/login role_id=0a437428-f796-f336-8013-2a248f91da7b
  secret_id=9962a7fc-391b-a509-6741-0fd5f3c0096d
Key                    Value
---                    -----
token                  hvs.CAESILoTjUkA...
```

Read the role identifier

Generate a secret identifier

You can use the token for API calls going forward

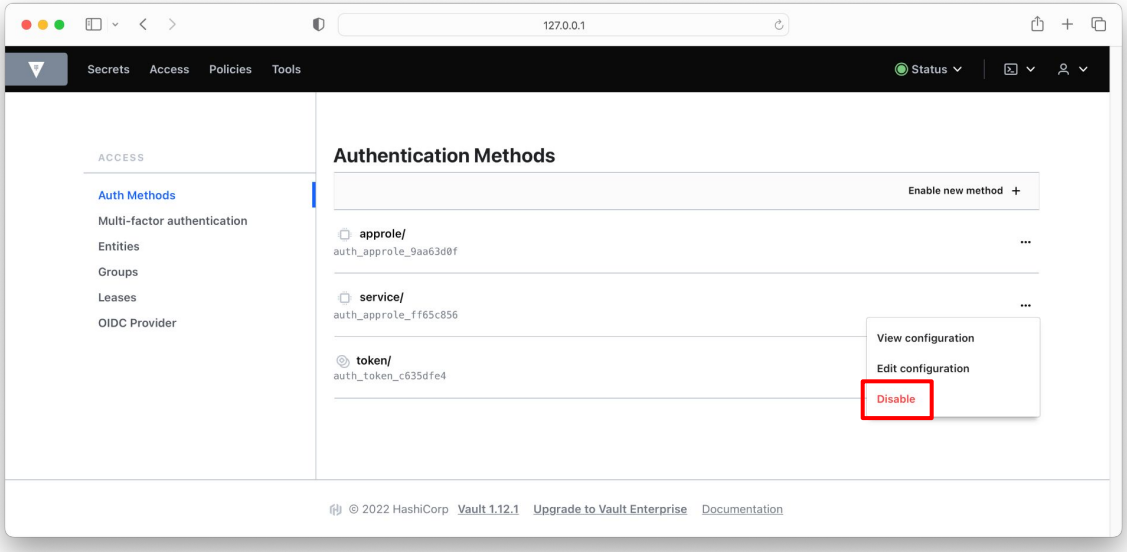
Disabling a Method from CLI

Identified by path, removes authentication method

```
$ vault auth disable service/
Success! Disabled the auth method (if it existed) at: service/
```

Disabling a Method from UI

Access > Auth Methods > Disable



EXERCISE

Using an Authentication Method

Q & A

Managing Policies

Overview, Policy Syntax and Management, Assignment to Authentication Method

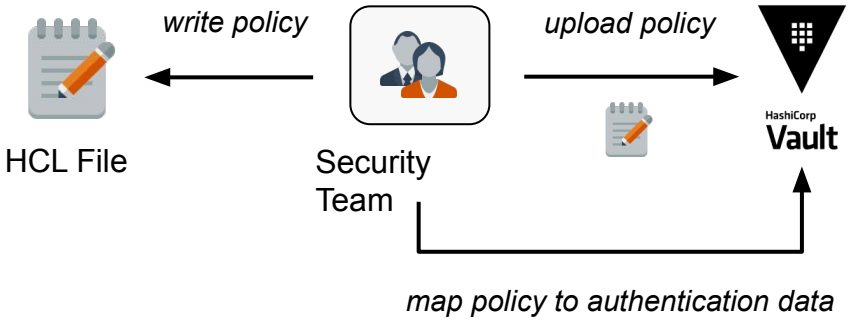
What are Policies?

Define permissions in Vault, also called ACL Policies

- Specifies “What operations am I allowed to perform or denied as the authenticated user or identity?”.
- Built-in policies (`default` and `root`) exist to establish standard permissions.
- Deny by default, so an empty policy grants no permission in the system.

Policy Management Process

Security team is responsible for writing and uploading policies



Policy Syntax

The most common syntax is HCL, consists of two components

- **Path:** Defines the URL context path for a functionality to assign the permissions to.
- **Capability:** The operations are allowed for the path.

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

Read permissions for the path `secret/foo`

Path Definition

Explicit, fully-spelled out path or path containing a glob

```
path "secret/foo" { ... }
path "secret/bar/*" { ... }
path "secret/+/team" { ... }
```

Only this path, *no* path segment below

A*ny* segment path below this path

`secret/bar/web`

`secret/bar/some/other`

This path with a *single* path segment (any number of characters)

`secret/web/team`

`secret/other/team`

Available Capabilities

Permitted or denied operations (see docs)

- **Basic (mapped to HTTP verbs):** `create`, `read`, `update`, `patch`, `delete`, `list`
- **Vault-Specific (not mapped to HTTP verbs):** `sudo` (access to paths that are root-protected), `deny` (disallow access)

Capability HTTP Verbs

Verbs to be used when making a call to

Capability	Associated HTTP verbs
create	POST/PUT
read	GET
update	POST/PUT
delete	DELETE
list	LIST
patch	PATCH
sudo	-
deny	-

Path Precedence

Rules can define more fine-grained capabilities

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

path "secret/bar" {
  capabilities = ["deny"]
}
```

Deny capability takes precedence over more generalized rule for this specific path

Formatting Policies from CLI

Automatically aligns contents based on style conventions

```
$ vault policy fmt admin-policy.hcl
Success! Formatted policy: secrets-list.hcl
```

```
path "secret/bar"
{
    capabilities = ["deny"]
}
```



```
path "secret/bar" {
  capabilities = ["deny"]
}
```

Policy Assignment

Explicit or automatic assignment to authentication method

- Explicitly assigned when token is created.
- Automatic assignment to an authentication method when its token is created. The association between policy and authentication method needs to be defined beforehand.
- Assigned to an entity in the identity secrets engine.

Built-In Policies

Come with Vault and define standard permissions

- **Default Policy:** Attached to all tokens upon creation. Can be modified but not deleted.

- **Root Policy:** A policy that is allowed to perform any operation on any path. Assigned to root token and acts as initial superuser. Cannot be modified or deleted. Exists in a brand new Vault installation but should be revoked in the production environments.

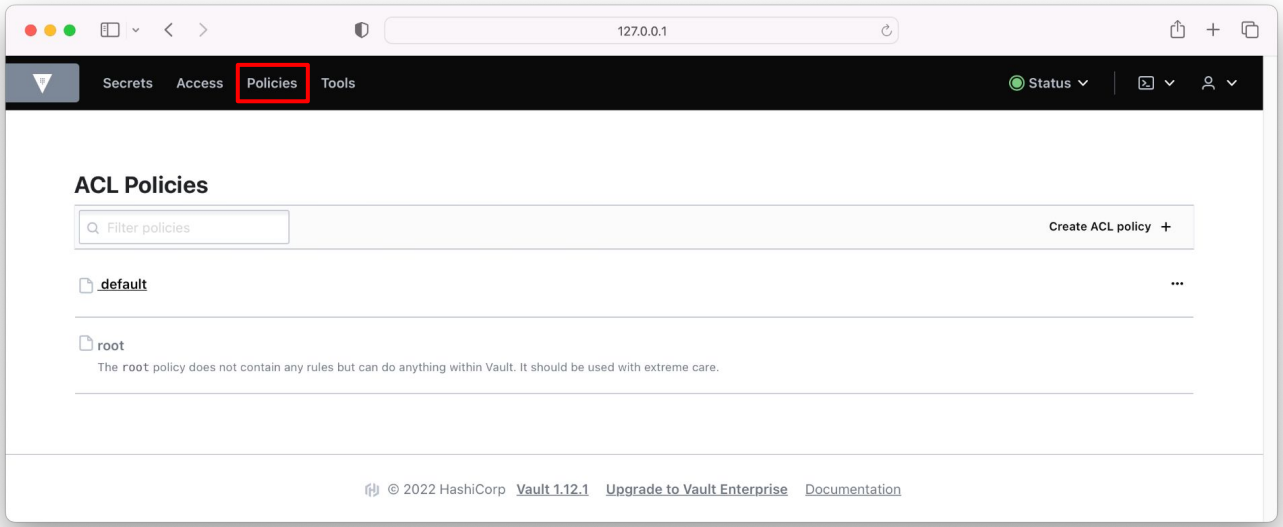
Listing Policies from CLI

Built-in policies will always be rendered

```
$ vault policy list
default
root
```

Listing Policies from UI

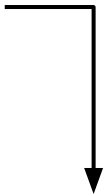
Policies menu option



Creating a Policy from CLI

Point to the HCL policy file

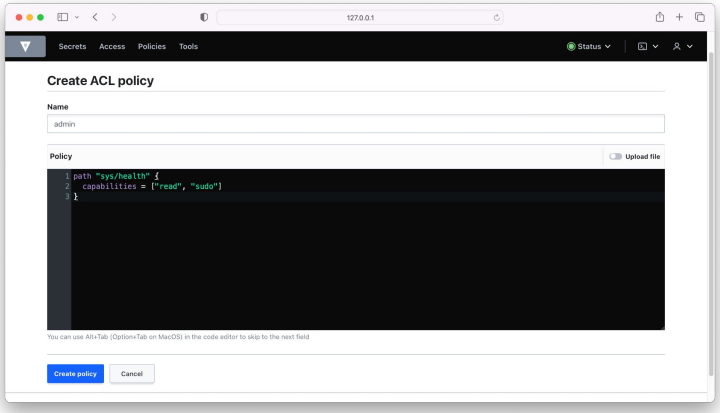
```
$ vault policy write admin admin-policy.hcl
Success! Uploaded policy: admin
```



```
path "sys/health" {
  capabilities = ["read", "sudo"]
}
```

Creating a Policy from UI

Policies > Create ACL policy



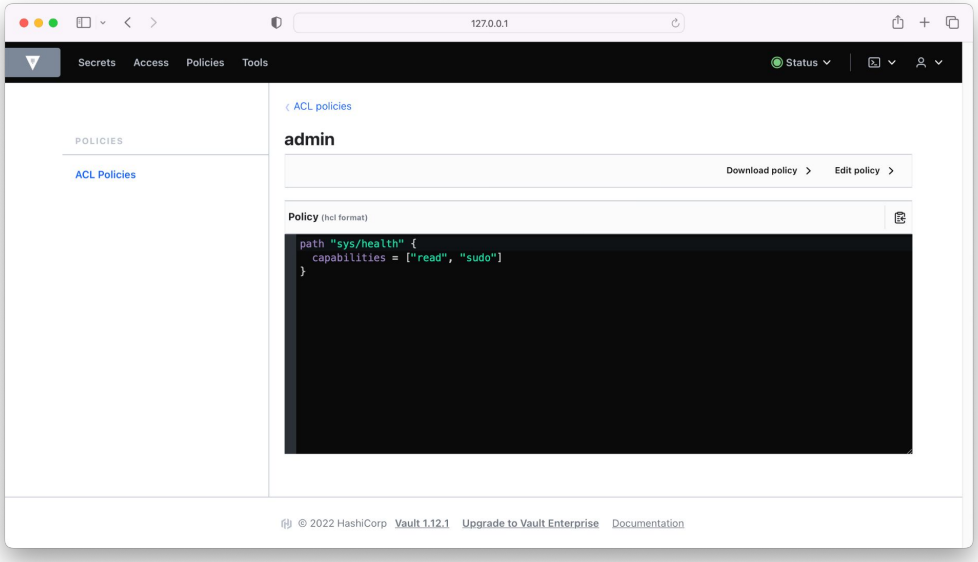
Reading a Policy from CLI

Renders the contents of uploaded policy file

```
$ vault policy read admin
path "sys/health" {
  capabilities = ["read", "sudo"]
}
```

Reading a Policy from UI

Policies > Select policy



Updating a Policy from CLI

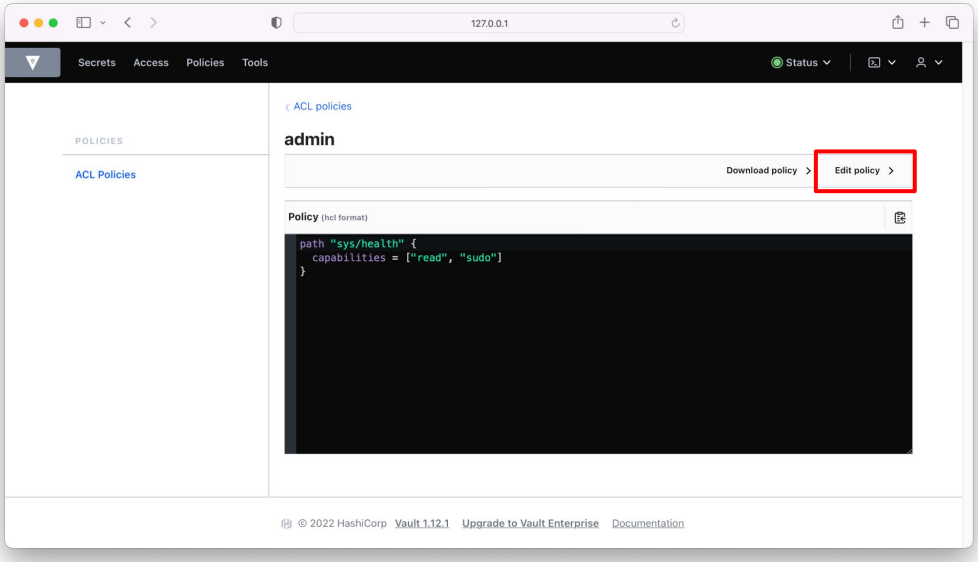
Change contents of policy file and upload again

```
$ vault policy write admin admin-policy.hcl
Success! Uploaded policy: admin
```

Change file contents and run the same command used for creation

Updating a Policy from UI

Policies > Select policy > Edit policy



EXERCISE

Managing Policies

Assigning a Policy from CLI

Assignment to token, userpass, or identity

```
$ vault token create -policy="admin"

$ vault write auth/userpass/users/ben token_policies="admin"

$ vault write identity/entity/name/ben policies="admin"
```

Assignment to a token

Assignment to an identity in secrets engine

Assignment to an authentication method

Policy Effects from CLI

Enforces denied operations

```
$ curl -H "X-Vault-Token: $token" -X GET $VAULT_ADDR/v1/sys/audit
{"errors":["1 error occurred:\n\t* permission denied\n\n"]}

$ curl -H "X-Vault-Token: $token" -X GET $VAULT_ADDR/v1/sys/health
{"initialized":true,"sealed":false,"standby":false,"performance_st
andby":false,"replication_performance_mode":"disabled","replicatio
n_dr_mode":"disabled","server_time_utc":1668210669,"version":"1.12
.1","cluster_name":"vault-cluster-f910f67d","cluster_id":"d6d79d2e
-9ced-21dd-66ec-3c094b051a79"}
```

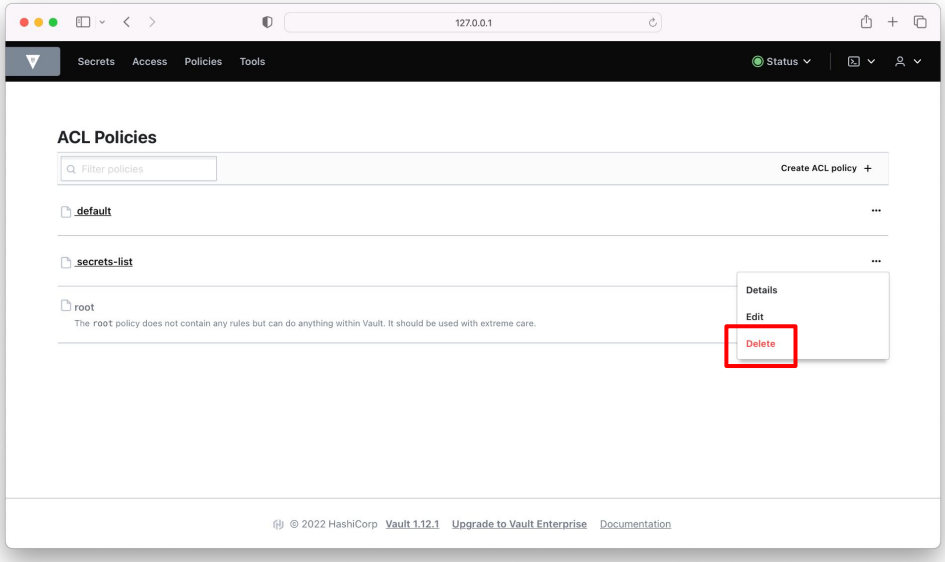
Deleting a Policy from CLI

Detaches policy from assigned authentication data

```
$ vault policy delete admin
Success! Deleted policy: admin
```

Deleting a Policy from UI

Policies > Select policy > Delete



EXERCISE

Assigning a Policy
to an Authentication
Method

Q & A

Using Tokens

Overview, Use Cases, Token Types, Token Lifecycle, TTL

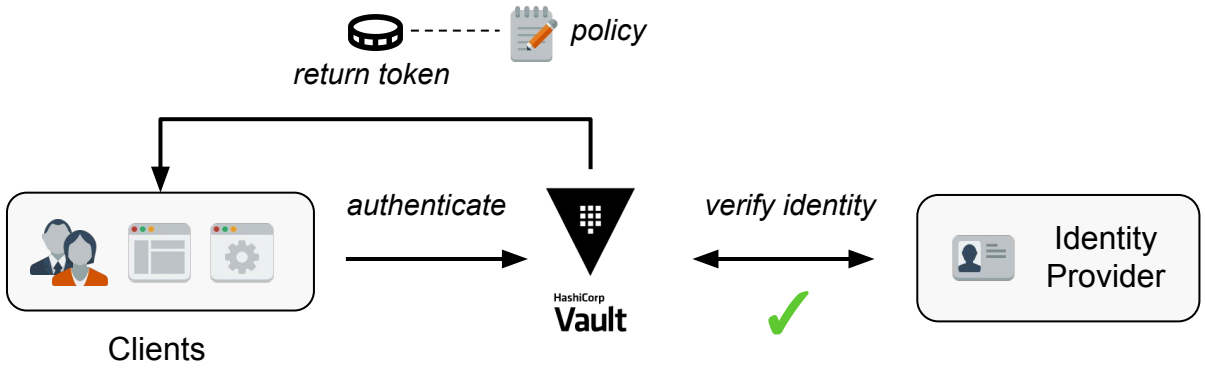
What are Tokens?

Core method for authentication within Vault (see docs)

- Can be provided directly when making a call to the API. For example, you can use the initial root token to initialize your system.
- You can generate a token from an authentication method. For example, for a Userpass authentication method, you can generate a corresponding token.
- A token maps to a set of policies that defines what the token holder is allowed to do.

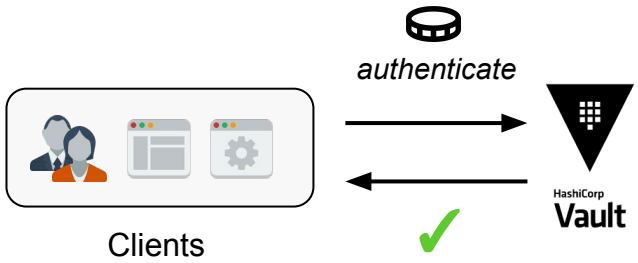
Token Generation

Vault generates token upon successful authentication



Token Usage

Subsequent calls to API provide the token



Root Token

A token with all available permissions

- Vault starts with a root token used to initialize setup.
- Does not expire.
- Should be revoked as soon as possible after setup procedure.
- A root token can create other root tokens. Those tokens should only be used in emergency situations.

Token Properties

Every token has specific characteristics

- **ID:** Value used for logging in or making a call to the API.
- **Accessor:** Value to look up a token without using it.
- **Type:** Is it a `service` or a `batch` token?
- **Policies:** Assigned when issued and evaluated when used.
- **TTL:** When does the token expire?
- **Orphaned:** Does the token have a parent token?

What's the Role of the Accessor?

Reference to a token with limited actions (see docs)

- Created and returned when token is generated. Some token types do not provide an accessor.
- It can only view token properties but not the ID.
- It can view capabilities for a path.
- It can be used to renew or revoke a token.

Token Types

Select the type based on use case (see feature comparison)

- **Service Token**
  - Has all features and is persisted in storage backend.
  - Can be renewed or revoked before reaching their time-to-live (TTL) value.
  - Default if not explicitly set to batch type.

- **Batch Token**
  - Lightweight, scalable, and not written to storage backend.
  - Cannot be listed or manually revoked.
  - Use to reduce the stress of managing a large number of tokens in parallel for performance reasons.

Token Prefixes

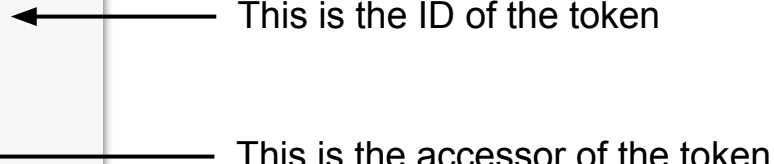
The ID of the token indicates token type

Token Type	Prefix
Service token	`hvs.`
Batch token	`hvb.`
Recovery token	`hvr.`

Creating a Service Token from CLI

Returns token ID and accessor

```
$ vault token create -policy=default -ttl=24h
Key                  Value
---                  -----
token                hvs.CAESIDPJkS-bHn...
token_accessor
YV7q0uqcYqivx0vJ2pfoyFys
token_duration       24h
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```



Service Token Info with ID from CLI

Uses the ID and renders full information

```
$ vault token lookup hvs.CAESIDPJkS-bHn...
Key                  Value
---                  -----
accessor             YV7q0uqcYqivx0vJ2pfoyFys
creation_time        1668629257
creation_ttl         24h
display_name         token
entity_id            n/a
expire_time
2022-11-17T13:07:37.251426-07:00
explicit_max_ttl     0s
id                   hvs.CAESIDPJkS-bHn...
issue_time
2022-11-16T13:07:37.251432-07:00
meta                 <nil>
num_uses             0
orphan               false
path                 auth/token/create
policies             [default]
renewable            true
ttl                  23h51m24s
type                 service
```

Type `service` is the default

Token Info with Access. from CLI

Uses the accessor and renders limited information

```
$ vault token lookup -accessor YV7q0uqcYqivx0vJ2pfoyFys
Key                 Value
---                 -----
accessor            YV7q0uqcYqivx0vJ2pfoyFys
creation_time       1668629257
creation_ttl        24h
display_name        token
entity_id           n/a
expire_time         2022-11-17T13:07:37.251426-07:00
explicit_max_ttl    0s
id                  n/a
issue_time          2022-11-16T13:07:37.251432-07:00
meta                <nil>
num_uses            0
orphan              false
path                auth/token/create
policies            [default]
renewable           true
ttl                 23h50m37s
type                service
```

Token ID cannot be viewed

Creating a Batch Token from CLI

Needs to declare token type explicitly

```
$ vault token create -type=batch -policy=default -ttl=10m
Key                  Value
---                  -----
token                hvb.AAAAAQJz7U...
token_accessor       n/a
token_duration       10m
token_renewable      false
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

Batch Token Info with ID from CLI

Uses the ID and renders full information

```
$ vault token lookup hvb.AAAAAQJz7U...
Key                  Value
---                  -----
accessor             n/a
creation_time        1669162941
creation_ttl         10m
display_name         token
entity_id            n/a
expire_time          2022-11-22T17:32:21-07:00
explicit_max_ttl     0s
id                   hvb.AAAAAQJz7U...
issue_time           2022-11-22T17:22:21-07:00
meta                 <nil>
num_uses             0
orphan               false
path                 auth/token/create
policies             [default]
renewable            false
ttl                  5m57s
type                 batch
```

Doesn’t have an accessor

Cannot be renewed

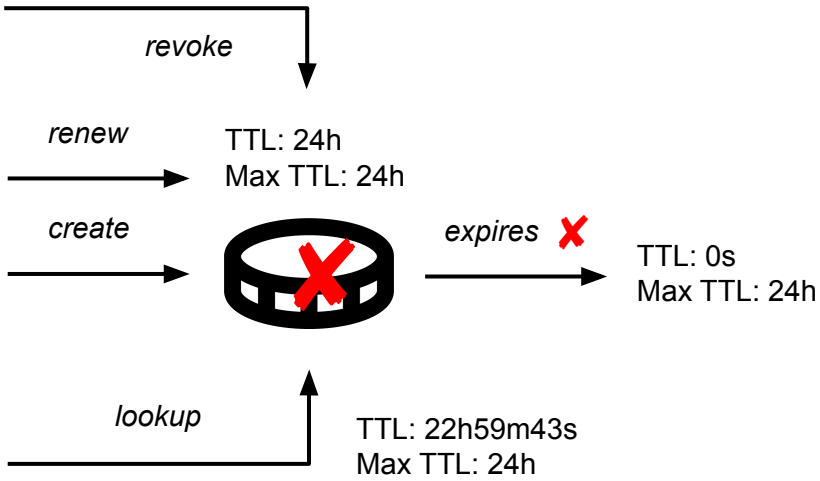
TTL Details

Creation time, elapsed time, and maximum expiration time

TTL Attribute	Description
`creation_time`	The token creation timestamp in Unix format.
`creation_ttl`	TTL set when token was created.
`expire_time`	Projected time when token will expire.
`expire_max_ttl`	Maximum TTL that cannot be exceeded.
`issue_time`	Token creation timestamp in human-readable format.
`ttl`	Computed time left until token expires.

Token Life Cycle

The lifespan of a token can be controlled



Renewing a Token from CLI

Extend TTL by given increment or up to max TTL

```
$ vault token renew -increment=120m hvs.CAESIDPJkS-bHn...
Key                  Value
---                  -----
token                hvs.CAESIDPJkS-bHn...
token_accessor       YV7q0uqcYqivx0vJ2pfoyFys
token_duration       2h
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

Revoking a Token from CLI

Uses the accessor and renders limited information

```
$ vault token revoke -accessor YV7q0uqcYqivx0vJ2pfoyFys
Success! Revoked token (if it existed)
```

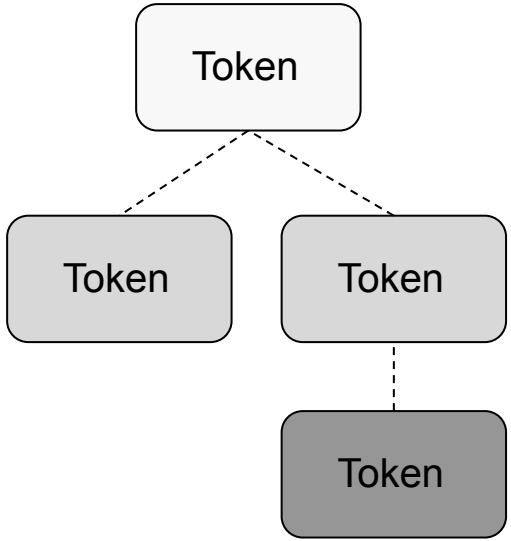
For security reasons, the effective outcome of the operation is not revealed

EXERCISE

Creating and Using
a Service and Batch
Token

Token Hierarchies

A token can create child tokens, usually cascade revokes children



Orphaned Tokens

Token without a parent token

- Can be defined upon creation with `-orphan` option.
- Tokens created by authentication methods will be orphaned by default.
- Revoking a parent token can explicitly make its children orphans.

Creating Orphan Token from CLI

A token without a parent

```
$ vault token create -orphan
Key                  Value
---                  -----
token                hvs.7fM8ne4k08Vq6BbYKKSSMff4
token_accessor       AhG5TfkiEtfY4gp2lzBtf5nQ
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Q & A

Comparing and Configuring Secrets Engines

Overview, Dynamic vs. Static, Managing Engines

What are Secrets Engines?

Plugins in Vault for managing secret data (see docs)

- Can store, generate, or encrypt data depending on the engine type.
  - **Store:** Data is stored outside of Vault but we want to give access to it through Vault.
  - **Generate:** Interacts with external service that Vault generates information for.
  - **Encrypt:** Vault provides encryption for data that lives outside of the system.

- Accessible through the path `/sys/mounts` and provide functions like read, write, and delete.

Examples for Secrets Engines

Wide range of options are available

- **Vault-internal:** Identity, Key-value store, Transit
- **Cloud:** AWS, Azure, Google Cloud
- **Identity:** LDAP, Active Directory
- **Databases:** MongoDB, Redis, PostgreSQL, MySQL
- **Others:** Kubernetes, Terraform Cloud, RabbitMQ

Default Dev Secrets Engines

The dev server starts with some initial secret engine paths

- **/identity:** Keeps track of authenticated clients to Vault. Cannot be disabled.
- **/cubbyhole:** Stores arbitrary secrets within the configured physical storage for Vault namespaced to a token. Cannot be disabled.
- **/secret:** A key-value store used to store arbitrary secrets. Can be disabled. This secrets engine is meant as a playground for starters when using the development server.

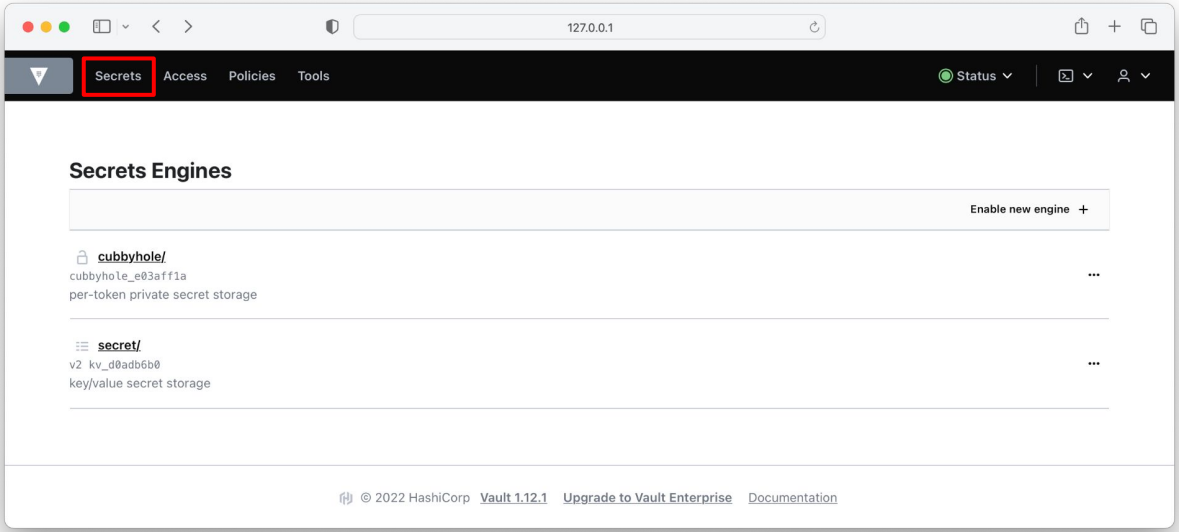
Listing Secrets Engines from CLI

Description field describes the type of secret engine

```
$ vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_e03aff1a    per-token private
secret storage
identity/     identity     identity_3b6c23eb     identity store
secret/       kv           kv_d0adb6b0           key/value secret
storage
sys/          system       system_47c5e1dd       system endpoints used
for control, policy and debugging
```

Listing Secrets Engines from UI

Secrets menu option



Static vs. Dynamic Secrets Engines

As an end user, you will primarily interact with dynamic ones

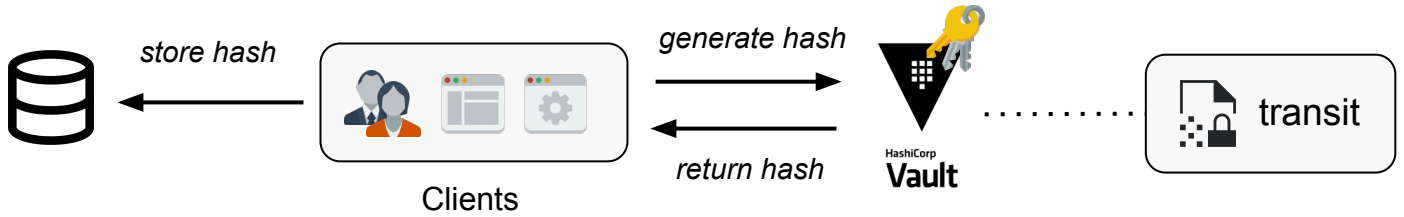
- **Static:** Manual management of secrets in Vault. An example of this is the key-value secret engine.

- **Dynamic:** Generates data upon request. Every secret has a lease which automatically expires the secret after a specific time frame. Most secret engines in Vault are dynamic, e.g. the database secret engines.

Transit Engine

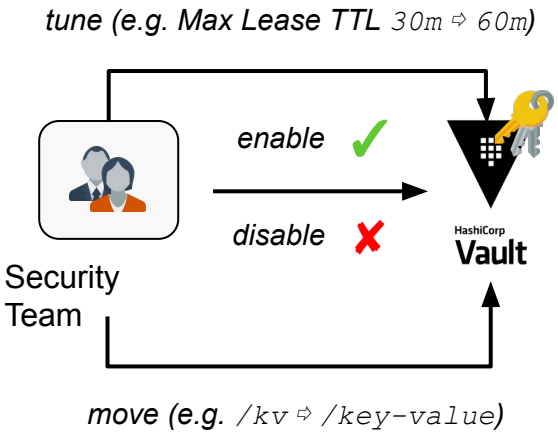
Doesn't store data, but provides cryptographic functions (see docs)

- **Primary use case:** Encrypt data from applications while still storing that encrypted data in some primary data store.
- **Functions:** Sign and verify data; generate hashes and HMACs of data; and act as a source of random bytes.



Secrets Engine Lifecycle

Security team is responsible for writing and uploading policies



Enabling a Secrets Engine from CLI

Enables engine by path

```
$ vault secrets enable -path=kv-v1 kv
Success! Enabled the kv secrets engine at: kv-v1/

$ vault secrets enable -path=kv-v2 -version=2 kv
Success! Enabled the kv-v2 secrets engine at: kv-v2/
```

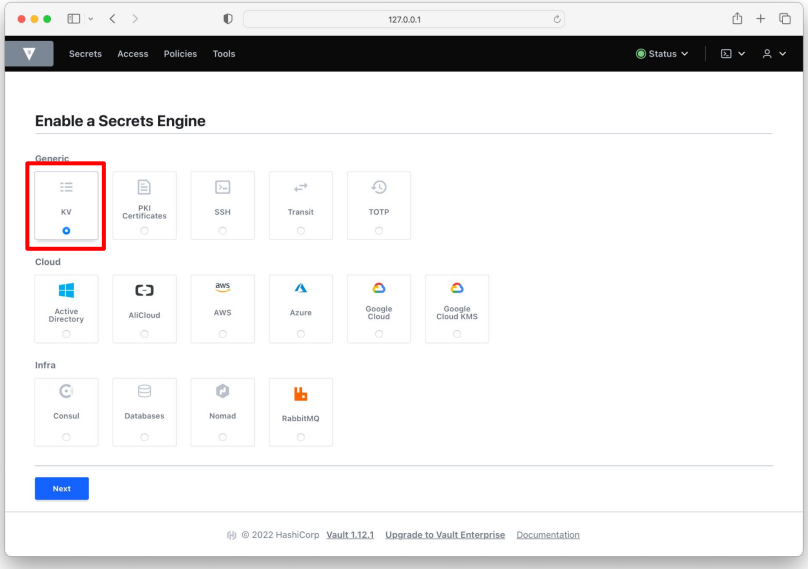
Engine type is “key-value” with version 1

Engine type is “key-value” with version 2

Comparison between kv engine versions

Enabling a Secrets Engine from UI

Secrets > Enable new engine



Tuning a Secrets Engine from CLI

Tweaks configuration of existing engine (see docs)

```
$ vault secrets tune -default-lease-ttl=72h
  -description="My key-value secret" kv
Success! Tuned the secrets engine at: kv/

$ vault write kv/config max_versions=5
Success! Data written to: kv/config
```

Updates TTL and description

Updates # of versions to keep in history

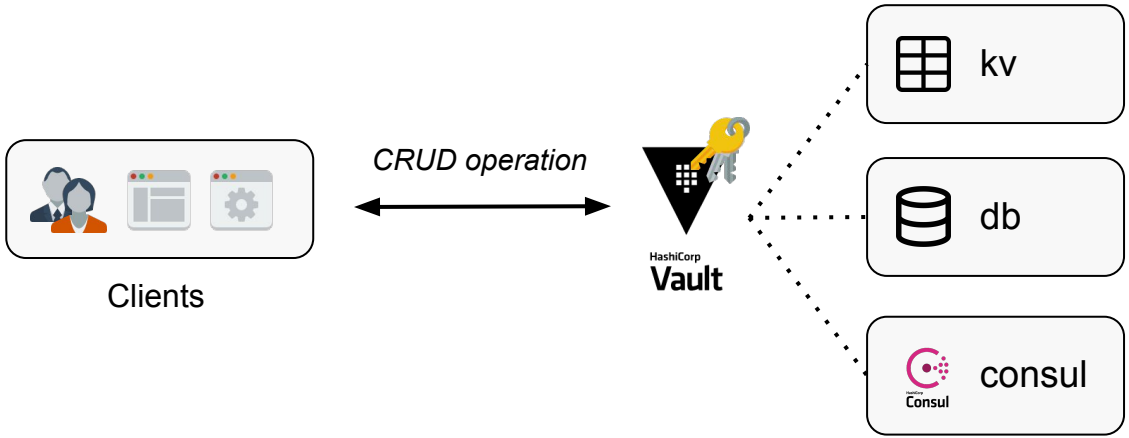
Cannot be changed from the UI!

EXERCISE

Enabling and
Configuring a
Secrets Engine

Secrets Usage

Standard operations are available through CLI, API, and UI



KV Command Interaction from CLI

Only applies to kv secrets engine (see docs)

```
$ vault kv put business-app/db name=therium
...

$ vault kv list business-app
Keys
----
db

$ vault kv get -field=name business-app/db
therium
```

Creates a new version of entries in kv secrets engine

Get all keys

Get the value of a specific key

Secret Interaction from CLI

Use the `read` and `write` command

```
$ vault read business-app/data/db
Key            Value
---            -----
data           map[password:test123 username:ben]
metadata       map[created_time:2022-12-02T20:07:23.356605Z
custom_metadata:<nil> deletion_time: destroyed:false version:3]
```

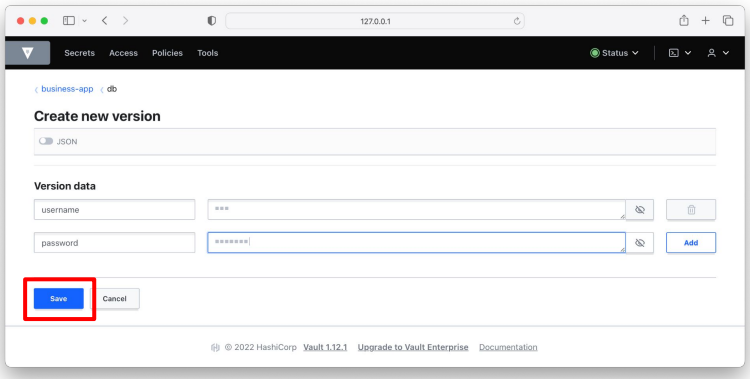
Secret Interaction from API

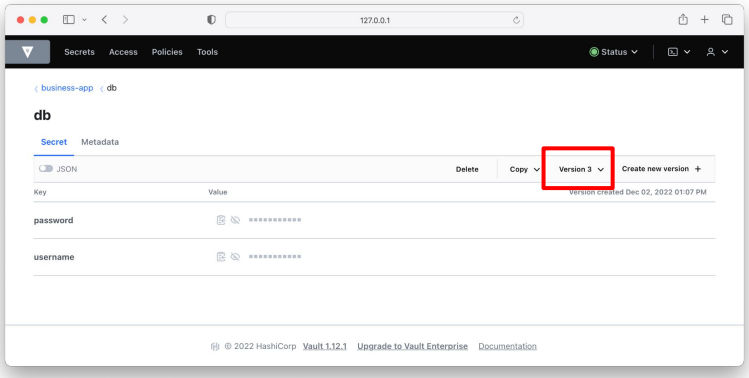
Get the latest or a specific version (see docs)

```
$ curl -H "X-Vault-Token: hvs.1dx0yCjCNZFQQOfUljLyRukq"
  $VAULT_ADDR/v1/business-app/data/db\?version=3
{"request_id":"4096fcec-d6e5-8d75-a18b-e29373474cd8","lease_id":"
","renewable":false,"lease_duration":0,"data":{"data":{"password"
:"test123","username":"ben"},"metadata":{"created_time":"2022-12-
02T20:07:23.356605Z","custom_metadata":null,"deletion_time":"","d
estroyed":false,"version":3}},"wrap_info":null,"warnings":null,"a
uth":null}
```

KV Interaction from UI

Secrets > Select Engine





EXERCISE

Interacting with a
Secrets Engine

Q & A

Managing Leases

Overview, Life Cycle, Operations

What are Leases?

Controls life cycle of a dynamic secret or a service token (see docs)

- A lease is effectively metadata that determine the duration the data will be available at path `/sys/leases`.
- When a lease expires, Vault will automatically invalidate the data and prevents further renewal.
- A client can renew a lease to extend the time span of a secret, or request a replacement secret.

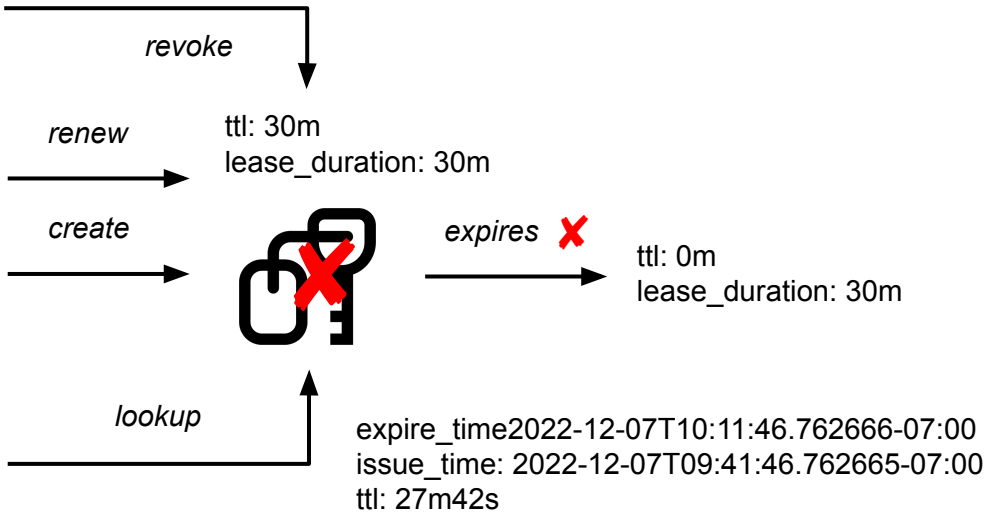
Lease Properties

Similar to token properties

- `lease_id`: The unique identifier attached to a lease.
- `lease_duration`: The amount of time left until the lease expires.
- `lease_renewable`: A boolean value that tells you if the lease can be renewed.

Lease Life Cycle

Very similar to token life cycle but with different terminology



Listing Active Leases from CLI

Access only available through path

```
$ vault list sys/leases/lookup
Keys
----
auth/

$ vault list sys/leases/lookup/auth/userpass/login/jill
Keys
----
h39b78c1c71e0bca8c130c4dfd811837db0f5c006a9760487ad4ab7b79b7fae3a
h432e5a575b0e40f188e795c0a2980b767afdcf0717ce4363bbfbf6d3a7487f63
```

Lease Details from CLI

Renders TTL information

```
$ vault lease lookup auth/userpass/login/jill/h39b78c1c71e0b...
Key              Value
---              -----
expire_time      2022-12-07T10:11:46.762666-07:00
id               auth/userpass/login/jill/h39b78c1c71e0b...
issue_time       2022-12-07T09:41:46.762665-07:00
last_renewal     <nil>
renewable        true
ttl              27m42s
```

Renewing a Lease from CLI

Resets the TTL and adjusts

```
$ vault lease renew database/creds/readonly/27e1b9a1-27b8-...
Key                Value
---                -----
lease_id           database/creds/readonly/27e1b9a1-27b8-...
lease_duration     5m
lease_renewable    true
```

Command only works for a dynamic secret, not for a token

Revoking a Lease from CLI

Deletes the lease

```
$ vault lease revoke auth/userpass/login/jill/h39b78c1c71e0b...
All revocation operations queued successfully!

$ vault list sys/leases/lookup/auth/userpass/login/jill
Keys
----
h432e5a575b0e40f188e795c0a2980b767afdcf0717ce4363bbfbf6d3a7487f63

$ vault lease revoke -prefix auth/userpass/login/jill
All revocation operations queued successfully!

$ vault list sys/leases/lookup/auth/userpass/login/jill
No value found at sys/leases/lookup/auth/userpass/login/jill
```

Revokes a single lease by ID

Revokes all leases by path

EXERCISE

Working With
Leases

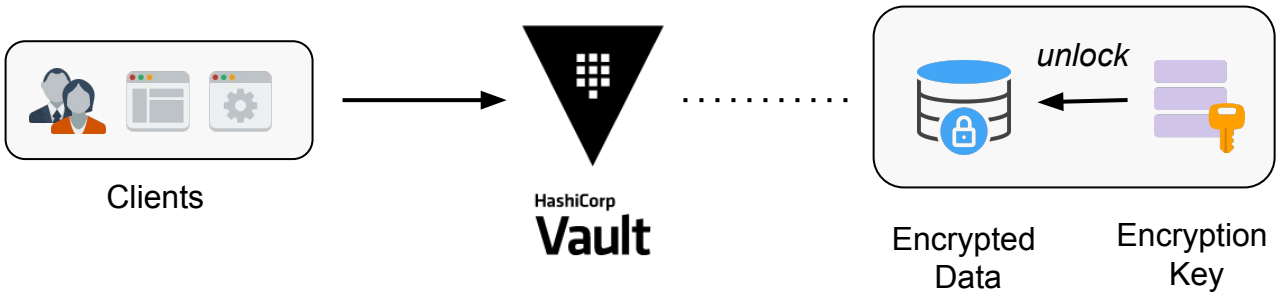
Q & A

Understanding the Vault Architecture

Data Encryption, Storage Backends, High Availability, Identity Management, Vault Agent, Response Wrapping

Encryption of Data in Vault

Data stored in Vault is encrypted with encryption key



Seal and Unseal

Only unsealed servers allow access to decrypted data (see docs)

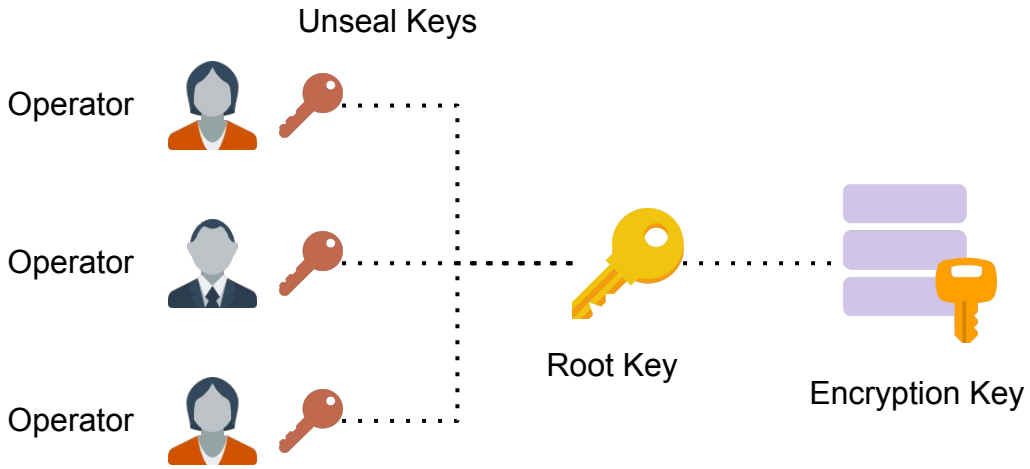
- Vault data is encrypted using the encryption key in the keyring; the keyring is encrypted by the root key; and the root key is encrypted by the unseal key.

- A server starts in *sealed* mode. It knows where and how to access the physical storage, but doesn't know how to decrypt any of it. Almost no operations are possible with Vault.

- *Unsealing* is the process of obtaining the plaintext root key necessary to read the decryption key to decrypt the data.

Shamir Seal

Algorithm that distributes unseal keys used to reconstruct root key



Configuring Vault Server

Meant for production environments

- Runtime parameters can be set with the help of an HCL or JSON file.
- Supports runtime aspects like storage backend, telemetry, and high availability.
- Every initialized Vault server starts in the sealed state. You need to unseal it before data can be accessed.

Server Configuration File

File name can be chosen freely (see docs)

```
storage "raft" {
  path    = "./vault/data"
  node_id = "node1"
}

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

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

Storage backend configuration

How Vault server listens for incoming requests from clients

General configuration
e.g. if the UI is enabled

Starting the Vault Server

Use `server` command and point to HCL config file

```
$ mkdir -p ./vault/data

$ vault server -config=config.hcl
...

==> Vault server configuration:

...

==> Vault server started! Log data will stream in
below:

...
```

Storage directory needs to exist before starting the Vault server

Section that renders server configuration

Section that renders log stream

Initializing the Vault

Generates encryption keys, unseal keys, and the initial root token

```
$ vault operator init
Unseal Key 1:
XNLxjqdc/rB03GzZBjVIHS5JiYv6IbGLU9qR0RmLLbSa
Unseal Key 2:
BminmjWWV73O+qLvNwKVkPRC7WTBq2dVO8YTjdWNgzH0
Unseal Key 3:
aNBfR0p91DZCsCjJPb+UswOkbJZGWTwvuGH/wcGdG032
Unseal Key 4:
SwGrPcLyVHXiB7rGcxUOaHqhhnX8QKyBZRxVZ8NCM6mn
Unseal Key 5:
IKrOA7DT/j5AR4CHoDFILJUmr7zeTPMToCKoMzCIfFjI

Initial Root Token: hvs.ZdvMiI70dPVw4UDqe4haES6V
```

Store unseal keys and root token in safe location. They cannot be retrieved anymore.

Unseal Key and Threshold

# of generated unseal keys, and required threshold can be tweaked

```
$ vault operator init -key-shares=3 -key-threshold=2
```

Generates the given number of unseal keys and renders them in console output

Required number of keys to be entered to unseal server

Unsealing the Vault

Repeat process until “unseal threshold” has been reached

```
$ vault operator unseal
Unseal Key (will be hidden):  <enter unseal key>
Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             true
Total Shares       5
Threshold          3
Unseal Progress    1/3
Unseal Nonce
93c5d490-4dde-9ce1-be62-2af1fe778a9d
Version            1.12.1
Build Date         2022-10-27T12:32:05Z
Storage Type       raft
HA Enabled         true
```

Enter one of the unseal keys here

The amount of times you have to repeat the process for

EXERCISE

Initializing and
Unsealing a Vault
Server

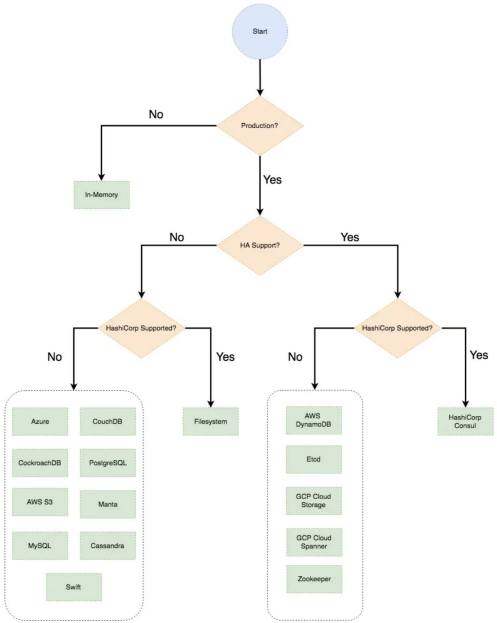
Storage Backend

Persists encrypted data in external or integrated location (see docs)

- Supports *external* solutions like Azure, Consul, and S3, each of which have their own pros, cons, advantages, and trade-offs.
- *Integrated* storage, e.g. Raft, to support HA concerns by replicating the data across multiple nodes.
- HashiCorp recommends using an integrated storage over external storage for easy of management and support for non-functional requirements.

Deciding on a Storage Backend

Factors like High Availability, support, and production-readiness



https://www.ahead.com/resources/hashicorp-vault-storage-backend-decision-tree

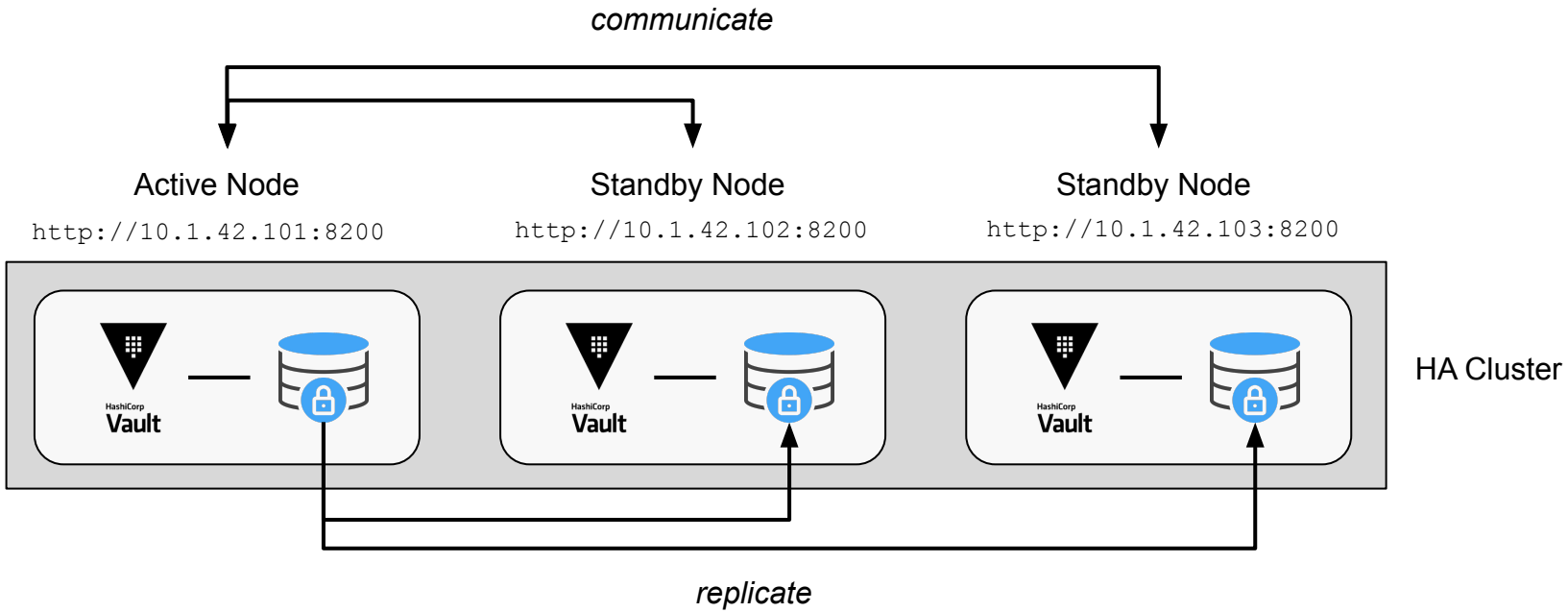
Configuring a Storage Backend

Use the `storage` stanza (see docs)

```
storage "raft" {
  path    = "./vault/raft"
  node_id = "node_2"

  retry_join {
    leader_api_addr = "http://127.0.0.2:8200"
  }
}
```

High Availability Cluster Architecture

Involves node machines, networking, and storage



High Availability Cluster Aspects

Protects against outages by running multiple Vault servers

- The active node puts a lock on the storage, all other nodes become standby nodes.
- Standby nodes forward or redirect requests to the active node if they receive requests.
- HA in Vault is not about scalability. The bottleneck is access to storage.

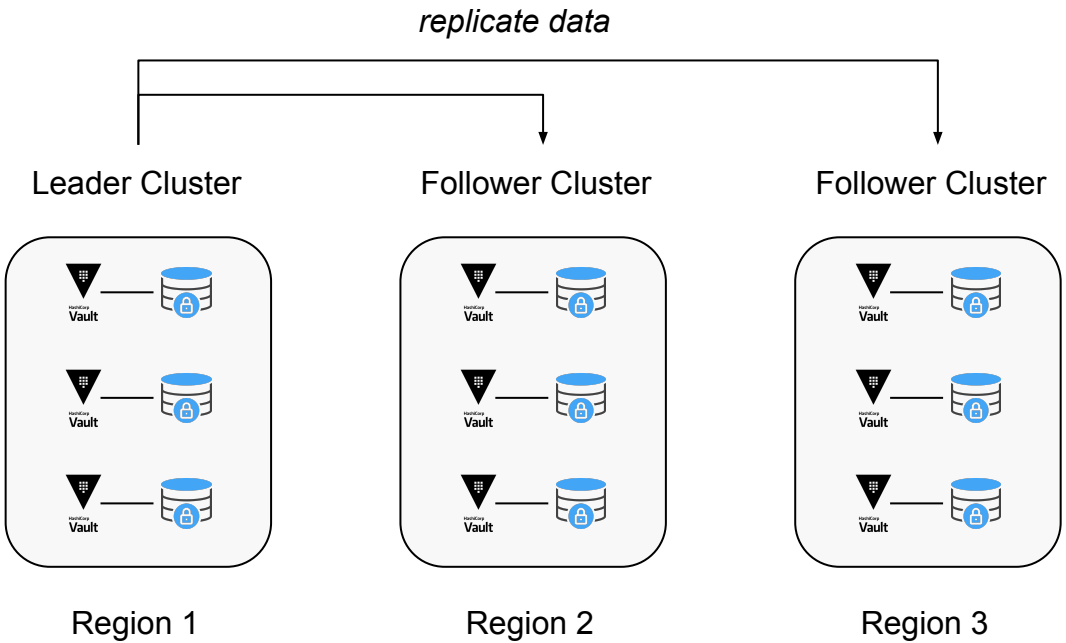
Enterprise Data Replication

Works across Vault clusters in different regions or data centers (see docs)

- Feature is only supported by Vault Enterprise, the commercial product of Vault.
- Operates on a leader/follower model, and works asynchronously.
- Requires a storage backend that supports transactional updates, such as Integrated Storage or Raft or Consul.

Multi-Cluster HA Architecture

Supports aspects like disaster recovery and performance



What is the Identity Engine?

Maintains the clients who are recognized by Vault (see docs)

- Mounted by default. Cannot be disabled or moved.
- Policies can be set on the entities which adds capabilities to the tokens that are tied to entity identifiers.
- Introduces and manages identity objects like *Entity*, *Alias*, and *Group*.

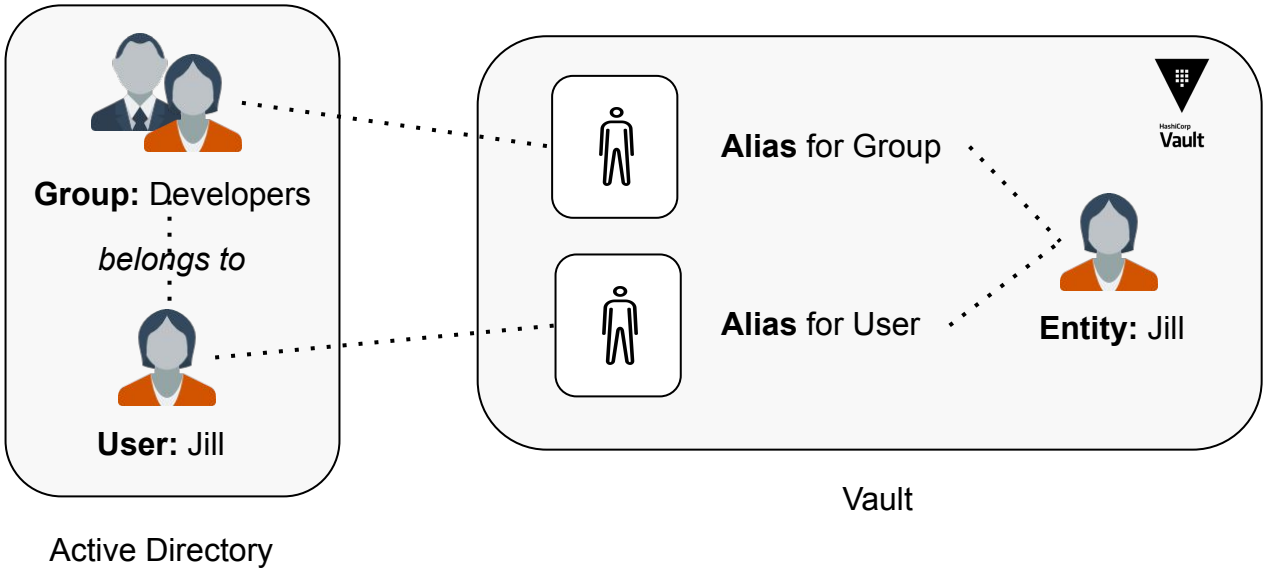
Identity Terms

Terms used by identity engine

- **Entity:** Represents a client, e.g. the `vault` CLI tool, a HTTPS call made via the `curl` command, or a human user, associated with a token.
- **Alias:** Represents an authentication method and is linked to an entity.
- **Group:** Managed internal or externally. Internal groups can contain entities and other groups.

Example: Mapping Entity and Alias

Users and Groups in Active Directory



What is the Vault Agent?

Simplifies integration of an application with Vault (see docs)

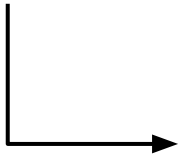
- The agent needs to be started as a process and usually runs in the background. The application interacts with the agent instead of the Vault server directly.

- The agent provides the following features:
  - Auto-authentication and token renewal.
  - Secrets caching and renewal.
  - Template for writing a secret to a file for consumption by the application.

Running the Vault Agent

Configuration lives in HCL file (see docs)

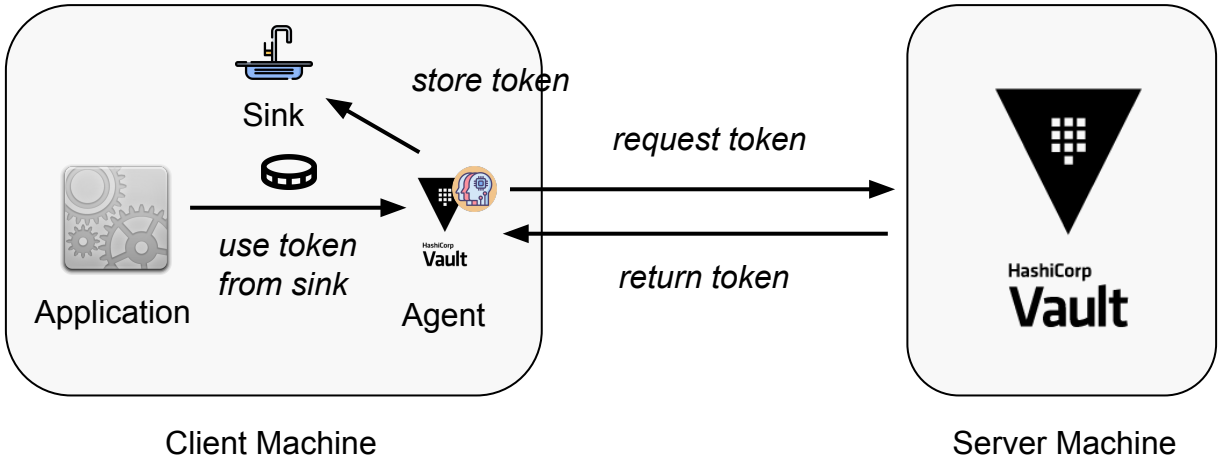
```
$ vault agent -config=/etc/vault/agent-config.hcl
```



```
vault {
  address = "https://vault-fqdn:8200"
  retry {
    num_retries = 5
  }
}

...
```

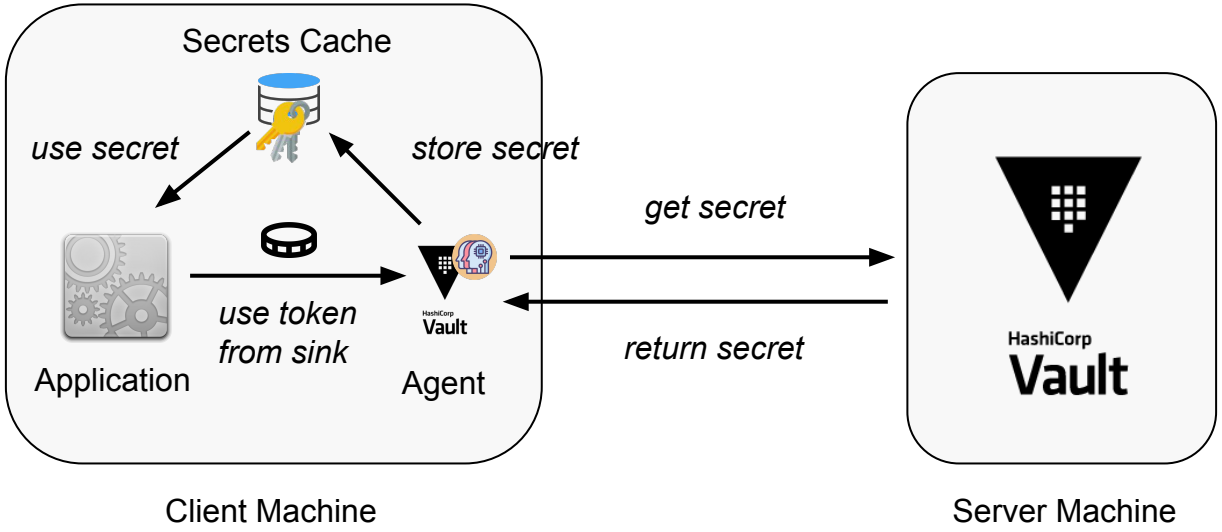
Auto-Authentication Workflow

Abstracts authentication procedure and provides token to application



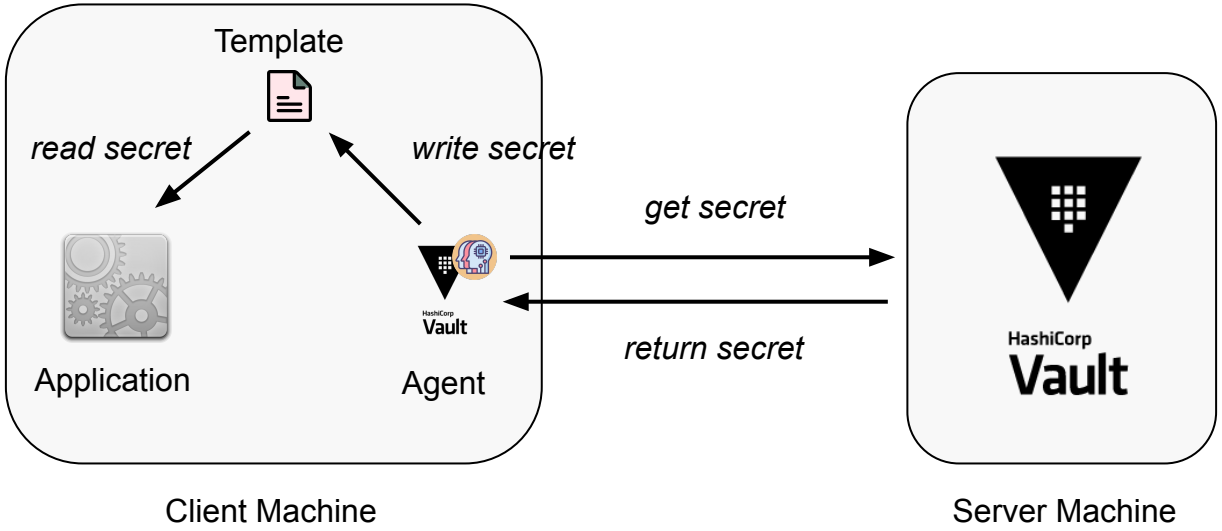
Secrets Caching Workflow

Stores secrets locally and reuses them for subsequent calls



Template Workflow

Application reads plain-text secret from file, doesn't interact with agent



What is the Response Wrapping?

Using a single-use token to access secret (see docs)

- Under certain conditions, it's easier to provide a client with a token instead of the secret directly.
- The original response is wrapped by the token, and retrieving it requires an unwrap operation against this token.
- Instead you can provide the dedicated, single-use token to clients. Clients would then be able to retrieve the secret themselves. The token cannot be renewed and will be invalidated after use.

Response Wrapping Commands

To wrap use `-wrap-ttl` option, to unwrap use `unwrap` cmd

```
$ vault kv get -wrap-ttl=1m secret/dev
Key                 Value
---                 -----
wrapping_token:     hvs.CAESIGu9Ulc0Uhmqa...
...

$ vault unwrap hvs.CAESIGu9Ulc0Uhmqa...
Key         Value
---         -----
data        map[password:my-long-password
username:webapp]
...
```

Creates wrapping token with a TTL

Unwraps secret with wrapping token

EXERCISE

Wrapping and Unwrapping a Secret

Q & A

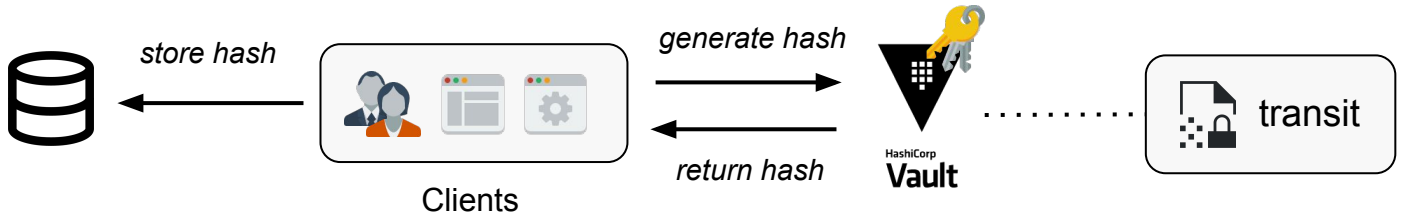
Explain Encryption as a Service

Overview, Transit Engine, Encryption, Decryption, Encryption Key Rotation

What is the Transit Engine?

Provides encryption as a service (see docs)

- Consumer submits unencrypted data, engine return encrypted data.
- Engine doesn't store the data.
- Manages cryptographic keys.



Encryption Keys

Select a key type by use case (see docs)

- AES, ChaCha20, Ed25519, ECDSA, RSA, HMAC with different versions.
- Key type determines supported actions, e.g. AES supports encryption/decryption, ECDSA supports signing and signature verification.
- Periodic rotation of the encryption keys is recommended.

Versioning of Keys

Not all versions are kept in memory for performance reasons

- The transit engine supports versioning of keys. If you rotate a key, it will be added to the *working set* available in memory.
- Versions earlier than `min_decryption_version` or `min_encryption_version` get archived in the *archive set* residing in the Vault storage backend.
- Keys can be deleted permanently by trimming them. The config operation takes care of configuring them.

Enabling the Transit Engine from CLI

Enable by type `transit`

```
$ vault secrets enable transit
Success! Enabled the transit secrets engine at: transit/
```

Creating Encryption Key from CLI

Interact with key using operations like `list`, `read`, `write`

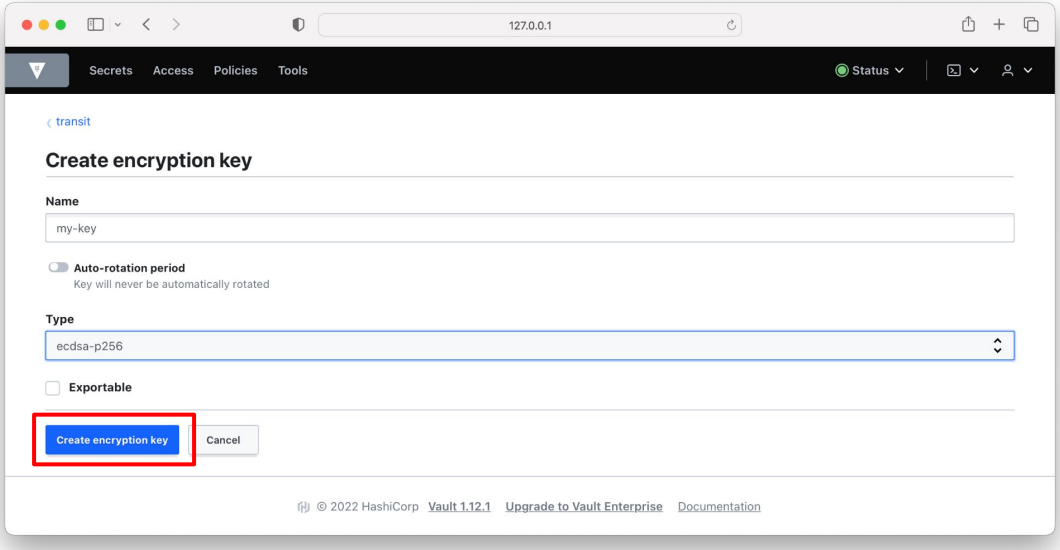
```
$ vault write -f transit/keys/my-key
Success! Data written to: transit/keys/my-key

$ vault write -f transit/keys/my-key type=ecdsa-p256
Success! Data written to: transit/keys/my-key

$ vault read transit/keys/my-key
Key                     Value
---                     -----
...
latest_version          1
```

Default is `aes256-gcm96`

Provides path with `type` parameter

Creating Encryption Key from UI

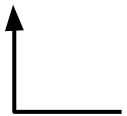
Secrets > Select transit engine > Create encryption key



Rotating Encryption Key from CLI

Determine time when key should be rotated based on formula

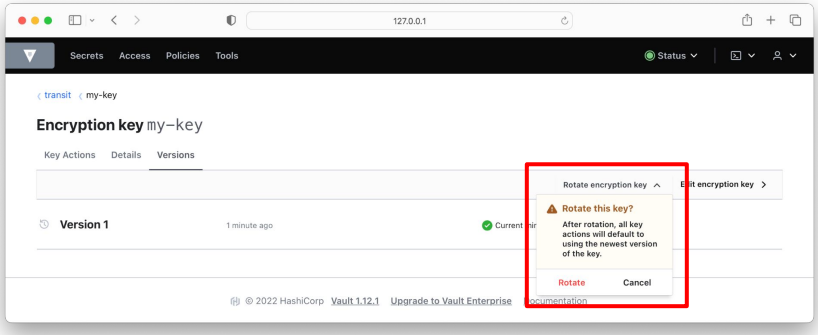
```
$ vault write -f transit/keys/my-key/rotate
Success! Data written to: transit/keys/my-key/rotate

$ vault read transit/keys/my-key
Key                         Value
---                         -----
...
latest_version              2
```

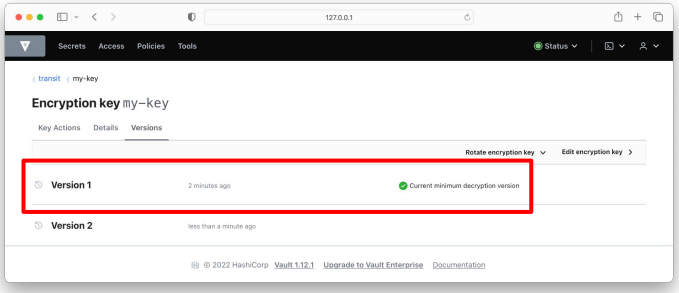


Key version is bumped up

Rotating Encryption Key from UI

Secrets > Select transit engine > Select key > Versions > Rotate





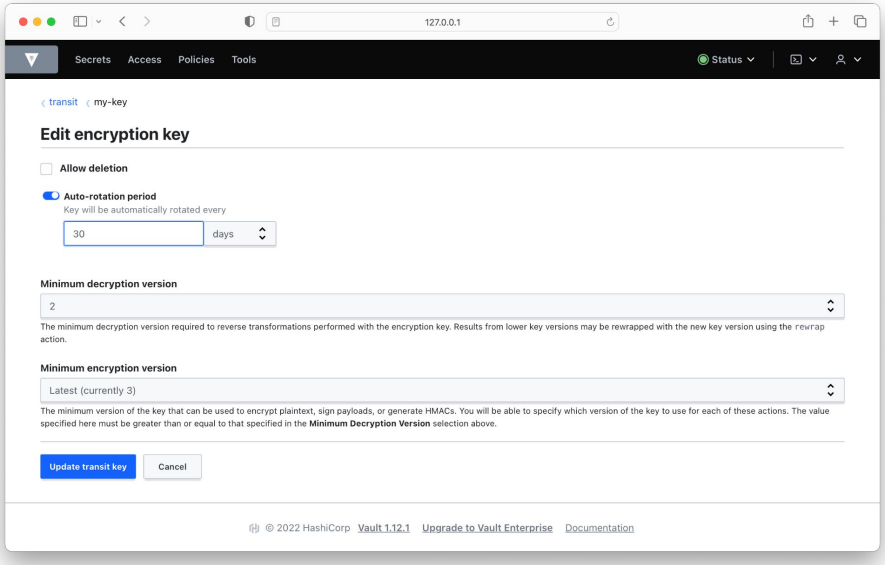
Configuring Encryption Key from CLI

Configures properties of encryption key

```
$ vault write transit/keys/my-key/config min_decryption_version=2
Success! Data written to: transit/keys/my-key/config

$ vault read transit/keys/my-key
Key                         Value
---                         -----
...
min_decryption_version      2
```

Configuring Encryption Key from UI

Secrets > Select transit engine > Select key > Versions > Edit



Trimming Encryption Key from CLI

Deleted key versions cannot be revived

```
$ vault write transit/keys/my-key/trim min_available_version=2
Success! Data written to: transit/keys/my-key/trim
```

Deletes version 1 and 2
(everything before 3)

Encrypting & Decrypting Data

Operations only available to key type that supports them

- **Encryption:** Submitted value has to be base64-encoded. You will receive encrypted value and metadata in response.

- **Decryption:** Returned value is base64-encoded and needs to be decoded. Version of the key is included as metadata. You will need to use exact key version to decrypt value.

Encrypting Data from CLI

Base64-encode value and pass it in with encryption call

```
$ echo "mypassword" | base64
bXlwYXNzd29yZAo=

$ vault write transit/encrypt/my-key plaintext=bXlwYXNzd29yZAo=
Key             Value
---             -----
ciphertext      vault:v3:6QeNuLGvLWnsND...
key_version     3
```

Decrypting Data from CLI

Use ciphertext to decrypted, Base64-decode for plain-text value

```
$ vault write transit/decrypt/my-key ciphertext=6QeNuLGvLWnsND...
Key           Value
---           -----
plaintext     bXlwYXNzd29yZAo=

$ echo "bXlwYXNzd29yZAo=" | base64 -d
mypassword
```

EXERCISE

Using the Transit Engine for Cryptographic Operations

Q & A

Summary & Wrap Up

Last words of advice...

O'REILLY®

Thank you