



Enable and Configure Secrets Engines



Available Secrets Engines



Active Directory

Database

Identity

SSH

AliCloud

Google Cloud

MongoDB Atlas

Terraform Cloud

AWS

Google Cloud KMS

Nomad

TOTP

Azure

Key Management

OpenLDAP

Transform

Consul

KMIP

PKI

Transit

Cubbyhole

KV

RabbitMQ

Venafi



Generic Secrets Engines



- HashiCorp can't expect everybody is proficient on all platforms or cloud providers
- These are the secrets engines you'll most likely be tested on

Cubbyhole

Identity

Database

PKI

KV

Transit



Generic Secrets Engines



- The database secrets engine supports 13+ different database platforms using plugins
- Key/Value has two versions, KV v1 and KV v2
- PKI creates X.509 certificates
- Transit encrypts data with encryption keys. Can also be used for Transit auto-unseal



Enabling Secrets Engines



- Cubbyhole and Identity are enabled by default (can't disable)
- Any other secrets engine must be enabled
 - Can enable using the CLI, API, or UI (most)
- Secrets engines are enabled and isolated at a unique path
 - All interactions with the secrets engine are done using the path
 - Paths do not need to match the secrets engines name or type
 - Make them meaningful for you and your organization



Enabling Secrets Engines

Command Line Interface (CLI)



Use the `vault secrets` command

- disable
- enable
- list
- move
- tune

Terminal

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

Terminal

```
$ vault secrets tune -default-lease-ttl=72h pki/
```



Enabling Secrets Engines

Command Line Interface (CLI)



Helpful flags to use with the vault secrets command

- vault secrets list --detailed
- vault secrets enable -path=developers kv
- vault secrets enable -description="my first kv" kv

Terminal

```
$ vault secrets enable -description="My Secrets" -path="cloud-kv" kv-v2
```



Enabling Secrets Engines

Command Line Interface (CLI)



```
vault secrets enable -path=bryan kv-v2
```



Type of Vault
object you want
to work with



Subcommand



Define a custom path to
enable the secrets
engine on



The type of
secrets engine
you want to
enable



Enabling Secrets Engines

Command Line Interface (CLI)



Use the `vault secrets list` command

Terminal

```
$ vault secrets list
```

Path	Type	Accessor	Description
aws/	aws	aws_dafa7adc	n/a
azure/	aws	aws_1a214ff6	n/a
vault-ops-pro/	kv	kv_28b1ceaa	Earn Your HCVOP Certification
cloud-team-kv/	kv	kv_fa270a3f	n/a
cubbyhole/	cubbyhole	cubbyhole_88c8e2e3	per-token private secret storage
dev-team-kv/	kv	kv_55c319c4	n/a
identity/	identity	identity_e60e93cb	identity store
kv-v2/	kv	kv_eea3206c	n/a
sys/	system	system_66b0d8ee	system endpoints used for local
transit/	transit	transit_7b8038ca	n/a



Enabling Secrets Engines

User Interface (UI)



Secrets Engines

aws aws/
aws_dafa7adc

aws azure/
aws_1a214fff6

☰ bryan/
v2_kv_28b1ceaa

☰ cloud-team-kv/
v2_kv_fa270a3f

Enable Additional Secrets Engines

Enable new engine +

Secrets Engines that are already enabled

The screenshot shows the HashiCorp Vault user interface. At the top, there's a navigation bar with tabs for "Secrets", "Access", "Policies", and "Tools". On the far right of the header, there are "Status" dropdowns and user icons. Below the header, the main content area is titled "Secrets Engines". It lists four entries: "aws aws/" (with subkey "aws_dafa7adc"), "aws azure/" (with subkey "aws_1a214fff6"), "☰ bryan/" (with subkey "v2_kv_28b1ceaa"), and "☰ cloud-team-kv/" (with subkey "v2_kv_fa270a3f"). To the right of the "aws" entry, there's a green curly brace grouping the first three entries. To the right of the "cloud-team-kv" entry, there's another green curly brace. In the top right corner of the main area, there's a red arrow pointing to a button labeled "Enable new engine +". The text "Enable Additional Secrets Engines" is overlaid in red above the arrow. The text "Secrets Engines that are already enabled" is overlaid in green below the first three entries.





Key/Value Secrets Engine



Key/Value Secrets Engine



- Key/Value secrets engine is used to store static secrets
 - There are two versions: v2 (kv-v2) is versioned but v1 (v1) is not
 - Secrets are accessible via UI, CLI, and API – interactive or automated
 - Access to KV paths are enforced via policies (ACLs)
- Like everything else in Vault, secrets written to the KV secrets engine are encrypted using 256-bit AES



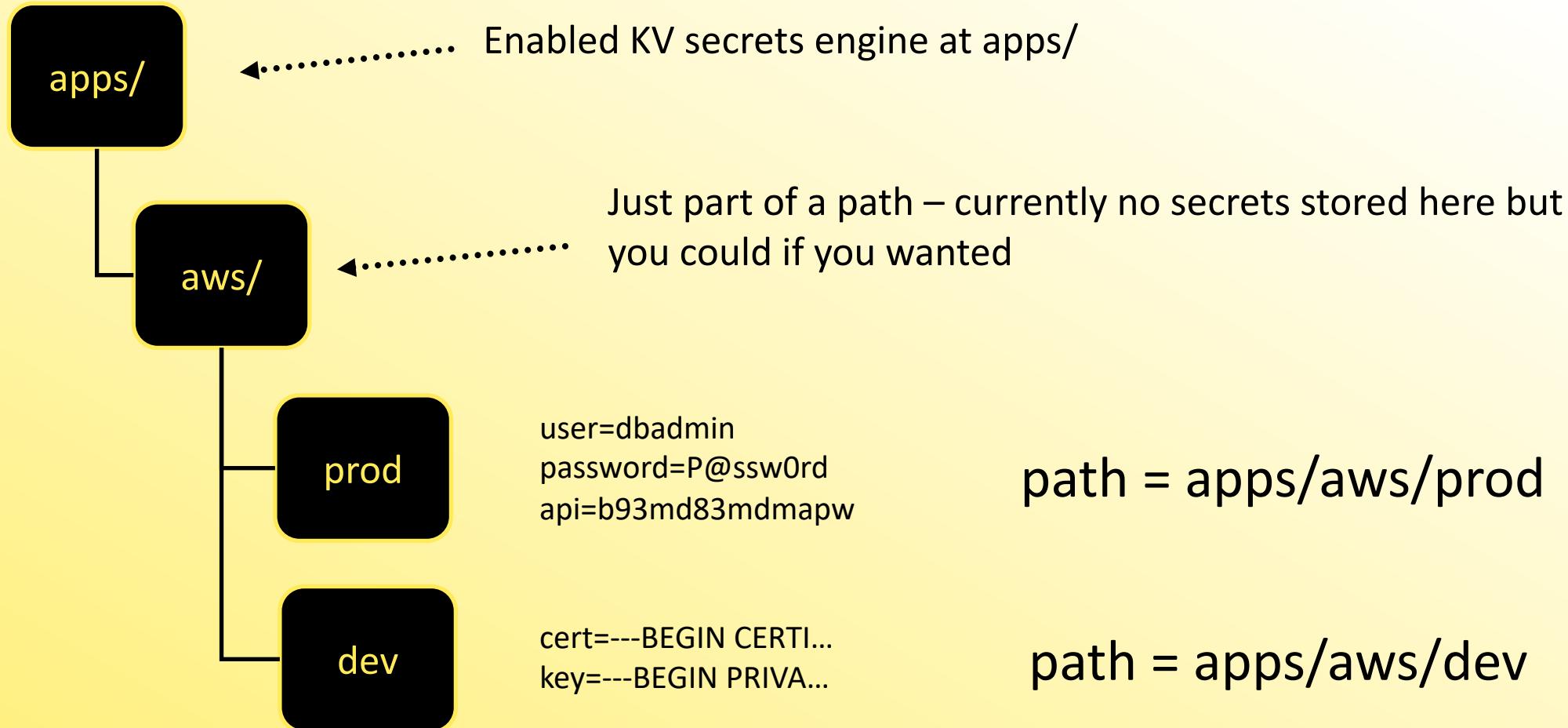
Key/Value Secrets Engine



- Key/Value secrets engine can be enabled at different paths
 - Each key/value secrets engine is isolated and unique
- Secrets are stored as key-value pairs at a defined path – (e.g., secret/applications/web01)
 - Writing a new secret will ***replace the old value***
 - Writing a new secret requires the `create` capability
 - Updating/overwriting a secret to an existing path requires `update` capability



Key/Value Secrets Engine



Key/Value Secrets Engine

Enable Version 1

```
$ vault secrets enable kv      ◀..... Enable at default path
Success! Enabled the kv secrets engine at: kv/

$ vault secrets enable -path=hcvop kv   ◀..... Enable at custom path
Success! Enabled the kv secrets engine at: hcvop/

$ vault secrets list --detailed
Path          Plugin        Accessor        ... Options
----          -----        -----          -----
cubbyhole/    cubbyhole    cubbyhole_ee5ae49  map[]       Empty map means
kv/           kv           kv_e8b99a3       map[]      ◀..... it's a KV v1 secrets
hcvop/        kv           kv_1d5e9cc1       map[]       engine
```



Key/Value Secrets Engine

Enable KV Version 2

```
$ vault secrets enable kv-v2      ◀..... Enable at default path
Success! Enabled the kv-v2 secrets engine at: kv-v2/

$ vault secrets enable -path=training -version=2 kv   ◀..... Another way to
Success! Enabled the kv-v2 secrets engine at: training/                                         enable KV v2

$ vault secrets list --detailed
Path          Plugin          Accessor          ... Options
---          -----
cubbyhole/    cubbyhole     cubbyhole_ee5ae49  map []
kv-v2/        kv            kv_e8b99a3       map [version:2]
training/     kv            kv_1d5e9cc1       map [version:2]
```

Notice the `version:2` in map, indicating a KV v2



Upgrade KV v1 to KV v2



- You can upgrade a KV v1 secrets engine to a KV v2
- You can't undo this upgrade
- ...and no....you can't go from KV v2 to KV v1

```
$ vault kv enable-versioning training/  
Success! Tuned the secrets engine at: training/
```



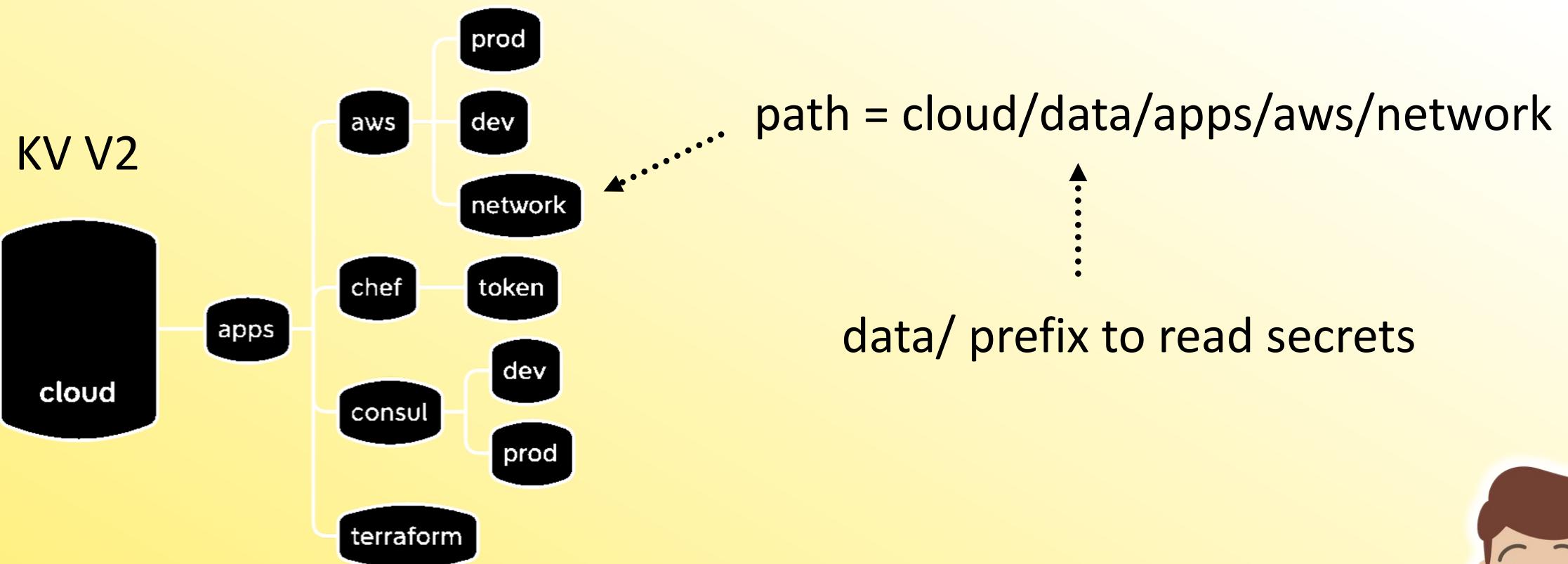
How is KV v2 Different?



- To support versioning, KV V2 adds metadata to our Key Value entries
- Used to determine creation date, the version of the secret, etc.
- Introduces two prefixes that must be accounted for when referencing secrets and/or metadata
 - cloud/data – data is where the actual K/V data is stored
 - cloud/metadata – the metadata prefix stores our metadata about a secret



How is KV v2 Different?



Important Operational Concept about KV V2!

How is KV v2 Different?



- The data/ and metadata/ prefix is required for API and when writing Vault policies
- It does NOT change the way you interact with the KV v2 store when using the CLI





Working with the Key/Value Secrets Engine



Working with KV using the CLI



Use the `vault kv` command

- `put` - write data to the KV
- `get` - read data from the KV
- `delete` - delete data from the KV
- `list` - list data within the KV

- `undelete` - undelete version of secret
- `destroy` - permanently destroy data
- `patch` - add specific key in the KV
- `rollback` - recover old data in the KV



Only available
for KV V2



Working with KV using the CLI

Compare KV Version 1 and Version 2

KV Version 1

```
$ vault kv put kv/app/db pass=123
Success! Data written to: kv/app/db
```

KV Version 2

```
$ vault kv put kv/app/db pass=123
== Secret Path ==
kv/data/app

===== Metadata =====
Key          Value
---          ---
created_time 2022-03-27T15:52:29.361762Z
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       1
```

The CLI command is the same, but we get different output behavior



Writing Data to the KV Store



```
vault kv put kv/app/db pass=123
```

Command Sub-command
when working with the KV Store

Path where you want to store the KV pair

Data to store – entered as KV pairs



Writing Data to the KV Store

What if I have a bunch of key pairs?

```
$ vault kv put kv/app/db pass=123 user=admin api=a8ee4b50cce124
Success! Data written to: kv/app/db

$ vault kv put kv/app/db @secrets.json .....  
.
Success! Data written to: kv/app/db
```

```
{
  "pass": "123",
  "user": "admin",
  "api": "a8ee4b50cce124"
}
```



Writing Data to the KV Store

Critical Things to Remember



Writing a new secret will replace the old value

Terminal

```
$ vault kv get kv/app/db
== Secret Path ==
kv/app/db

===== Metadata =====
Key          Value
---          ---
<output abbreviated>

==== Data ====
Key          Value
---          ---
pass        123
user        admin
api         a8ee4b50cce124
```

Maybe I want to update the value
of api key but nothing else



Writing Data to the KV Store



What will happen if I run the following command?

A blue terminal window icon with red, yellow, and green dots in the top-left corner.

Terminal

```
$ vault kv put kv/app/db api=39cms1204mfi2m
```



Writing Data to the KV Store



Terminal

```
$ vault kv put kv/app/db api=39cms1204mfi2m
== Secret Path ==
kv/data/app

Key          Value
---          -----
created_time 2022-12-21T14:40:26.886255Z
custom_metadata <nil>
deletion_time n/a
destroyed      false
version        2
```



A blue arrow points from the text "version" to the value "2" in the terminal output.



Writing Data to the KV Store



Terminal

```
$ vault kv get kv/app/db
== Secret Path ==
kv/data/app

===== Metadata =====
Key          Value
---          ---
created_time 2022-12-21T14:40:26.886255Z
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       2

==== Data ====
Key      Value
---      ---
api     39cms1204mfi2m
```

A write is NOT a
merge



Writing Data to the KV Store



All you wanted to do was update the api
but now you've lost your data?

What can we do?



Writing Data to the KV Store

Recover Your Data using Rollback



Terminal

```
$ vault kv rollback -version=1 kv/app/db
== Secret Path ==
kv/data/app

Key          Value
---          -----
created_time 2022-12-21T14:49:23.746331Z
custom_metadata <nil>
deletion_time n/a
destroyed      false
version        3 ←
```



Writing Data to the KV Store

Recover Your Data Using Rollback



Terminal

```
$ vault kv get kv/app/db
== Secret Path ==
kv/data/app

===== Metadata =====
Key          Value
---          -----
<output abbreviated>

==== Data ====
Key          Value
---          -----
pass         123
user         admin
api          a8ee4b50cce124
```



Writing Data to the KV Store

Patch Your Data



Terminal

```
$ vault kv patch kv/app/db user=bryan
....
===== Metadata =====
Key          Value
---          -----
created_time 2022-12-22T17:57:35.157363Z
destroyed    false
version      4

$ vault kv get kv/app/db
....
==== Data ====
Key          Value
---          -----
pass         123
user         bryan
api          a8ee4b50cce124
```

Outputs abbreviated



Reading Data from the KV Store

Compare Version 1 and Version 2



KV Version 1

```
$ vault kv get kv/app/db  
=====Data=====
```

Key	Value
---	----
pass	123
user	admin
api	a8ee4b50cce124



KV Version 2

```
$ vault kv get kv/app/db  
== Secret Path ==  
kv/data/app  
  
=====Metadata=====
```

Key	Value
---	----
creation_time	2022-12-15T04:35:56.395821Z
deletion_time	n/a
destroyed	false
version	1


```
=====Data=====
```

Key	Value
---	----
pass	123
user	admin
api	a8ee4b50cce124

The CLI command is the same, but we get different output behavior

Reading Data from the KV Store

Output



```
KV Version 2  
$ vault kv get kv/app/db  
== Secret Path ==  
kv/data/app  
  
=====Metadata=====  
Key          Value  
---          ----  
creation_time 2022-12-15T04:35:56.395821Z  
deletion_time n/a  
destroyed     false  
version       1  
  
=====Data=====  
Key          Value  
---          ----  
pass         123  
user         admin  
api          a8ee4b50ccel24
```

Default output type is table



Reading Secrets from the KV Store

Output



KV Version 2

```
$ vault kv get -format=json kv/app/db
{
  "request_id": "249fca06-a8ce-5617-d598-1c12384d4ac8",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": {
    "data": {
      "pass": "123",
      "user": "admin",
      "api": "a8ee4b50cce124",
    },
    "metadata": {
      "created_time": "2022-12-21T13:59:29.917893Z",
      "custom_metadata": null,
      "deletion_time": "",
      "destroyed": false,
      "version": 1
    }
  }
}
```

Change output
format to json

Output abbreviated

Useful for creating
machine-readable
outputs



Reading Secrets from the KV Store

Important Things to Remember



- A regular read request will return the latest version of the secret
- If the latest version of the secret has been deleted (KV V2), it will return the related metadata but no data (secrets)
- You can read a previous version of a secret (if one exists) by adding the `-version=x` flag to the request

A blue terminal window icon with three colored dots (red, yellow, green) in the top-left corner.

Terminal

```
$ vault kv get -version=3 kv/app/db
```



Deleting Secrets from the KV Store



- A delete on KV V1 is a delete – the data is destroyed
 - You'd have to restore Vault/Consul to retrieve the old data
- A delete on KV V2 is a soft delete – data is not destroyed
 - Data can be restored with a undelete/rollback action
- A destroy (only KV V2) is a permanent action – destroyed on disk
 - Cannot be restored except for a Vault/Consul restore action



Deleting Secrets in the KV Store

KV Version 1 – Read Output after Deleting



```
Terminal

$ vault kv delete secret/app/database
Success! Data deleted (if it existed) at: secret/app/database

$ vault kv get secret/app/database
No value found at secret/app/database
```



No values exist if you delete a secret at a path for KV V1



Deleting Secrets from the KV Store

KV Version 2 – Read Output after Deleting the Latest Version



```
Terminal

$ vault kv delete secret/app/web
Success! Data deleted (if it existed) at: secret/app/web

$ vault kv get secret/app/web
== Secret Path ==
secret/data/web

=====
Metadata =====
Key          Value
---          ---
created_time 2022-12-15T17:41:41.13052Z
custom_metadata <nil>
deletion_time 2022-12-15T17:42:03.369955Z
destroyed    false
version      3
```

Only returned metadata but no data (since it was deleted)



Destroy Secrets from the KV Store

KV Version 2 – Read Output after Destroying the Latest Version



Terminal

```
$ vault kv destroy -versions=3 secret/app/web
Success! Data written to: secret/app/web

$ vault kv get secret/app/web
== Secret Path ==
secret/app/web

===== Metadata =====
Key          Value
---          ---
created_time 2022-12-21T14:49:23.746331Z
custom_metadata <nil>
deletion_time n/a
destroyed      true ←
version        3
```





Database Secrets Engine



Intro to Database Secrets Engine



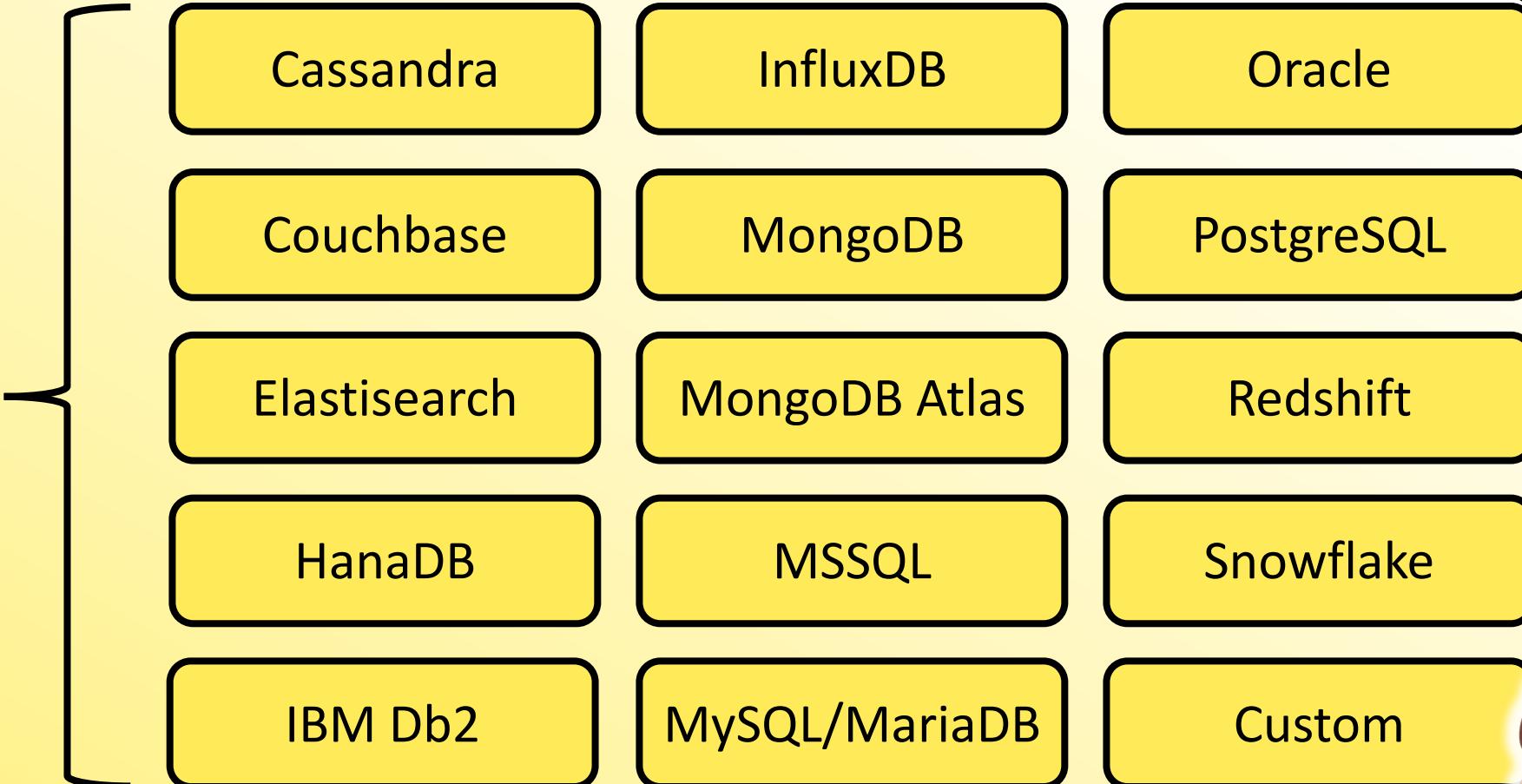
- The database secrets engine generates dynamic credentials against one or more databases – username and password
- Generated credentials are based on roles which provides a one-to-one mapping to a permission set on the targeted database
- Credentials are tied to a lease – credentials are revoked when the lease expires
 - Vault reaches back out to the database and deletes the credentials (no technical debt)



Database Secrets Engine - Plugins



Database
Secrets
Engine



Database Secrets Engine - Configuration



There are generally two steps when configuring a secrets engine that will generate dynamic credentials:

1

Configure Vault with access to the database

2

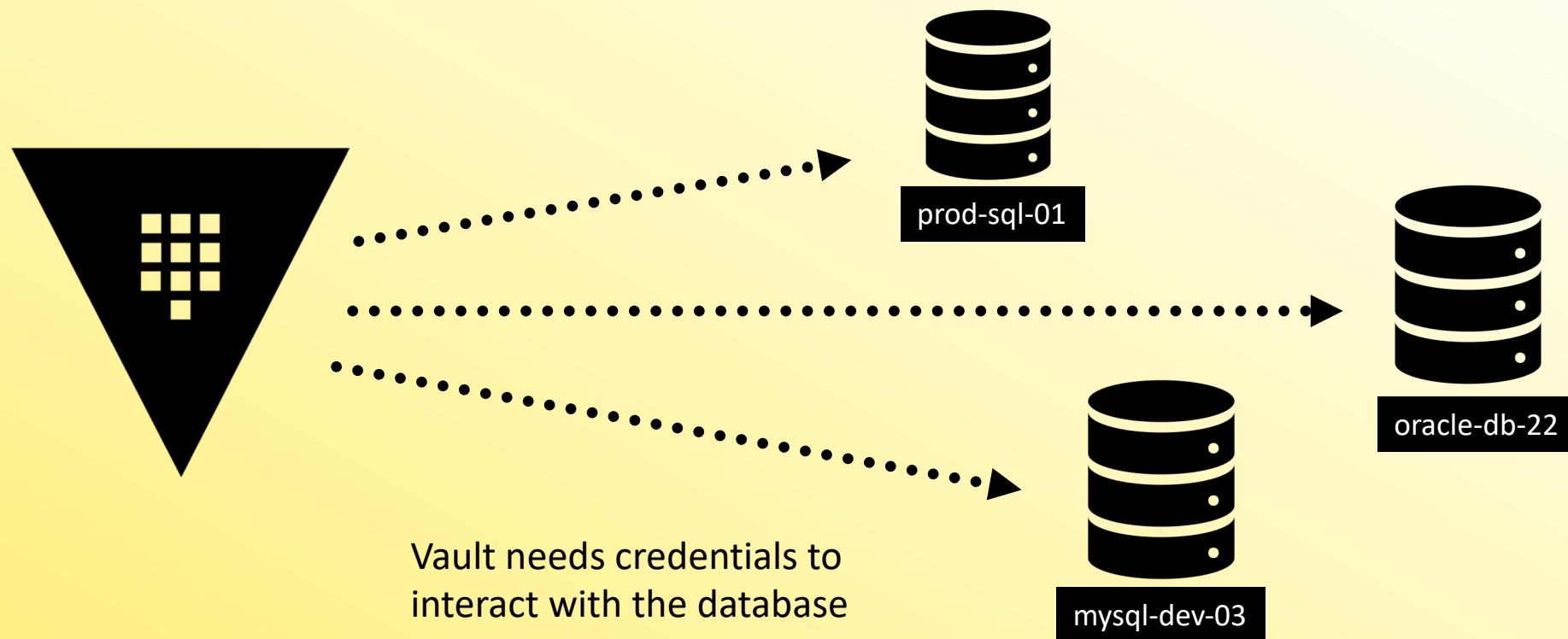
Configure roles based on permissions required



Database Secrets Engine – DB Config



Configure Vault with access to the database



Database Secrets Engine – DB Config



Provide credentials to a secrets engine that gives Vault permission to create, list, and delete credentials on the platform:



TERMINAL

```
$ vault write database/config/prod-database \
  plugin_name=mysql-database-plugin \
  connection_url="{{username}}:{{password}}@tcp(prod.hcvop.com:3306) /" \
  allowed_roles="app-integration, app-hcvop" \
  username="vault-admin" \
  password="vneJ4908fk3084Bmrk39fmslslf#e&349"
```

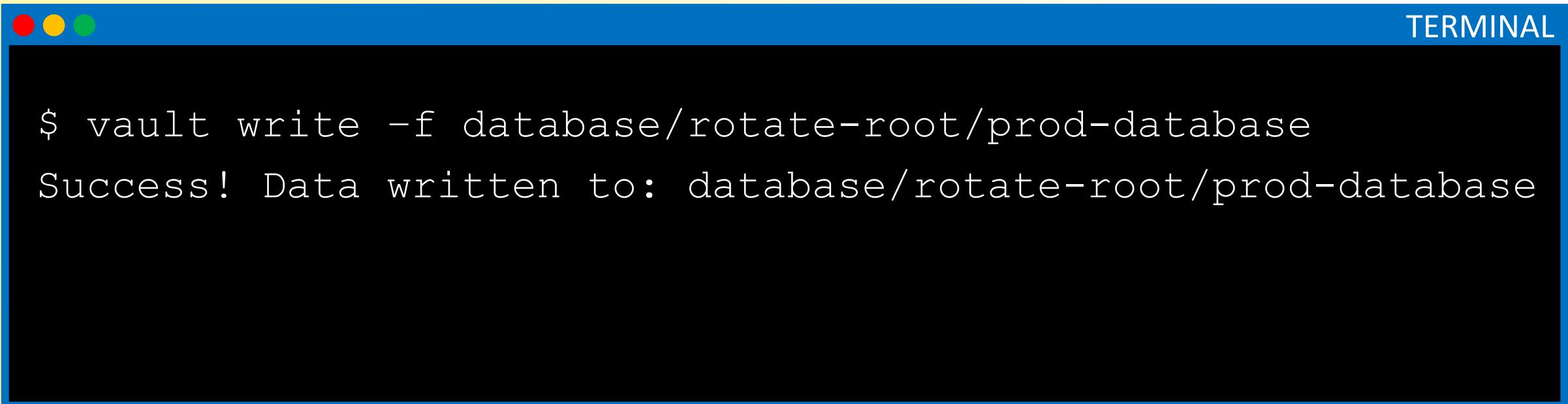
Rotate Root Credentials



- You can easily rotate the root credentials for each database configuration using the rotate-root endpoint
- This allows you to meet any internal policies where credentials should be frequently rotated
- Also ensures that only Vault and the Database server know the root credentials – *no human would know the creds.*



Rotate Root Credentials

A screenshot of a Mac OS X terminal window titled "TERMINAL". The window has its characteristic red, yellow, and green close buttons at the top left. The title bar is blue. The main area contains the following command and its output:

```
$ vault write -f database/rotate-root/prod-database
Success! Data written to: database/rotate-root/prod-database
```



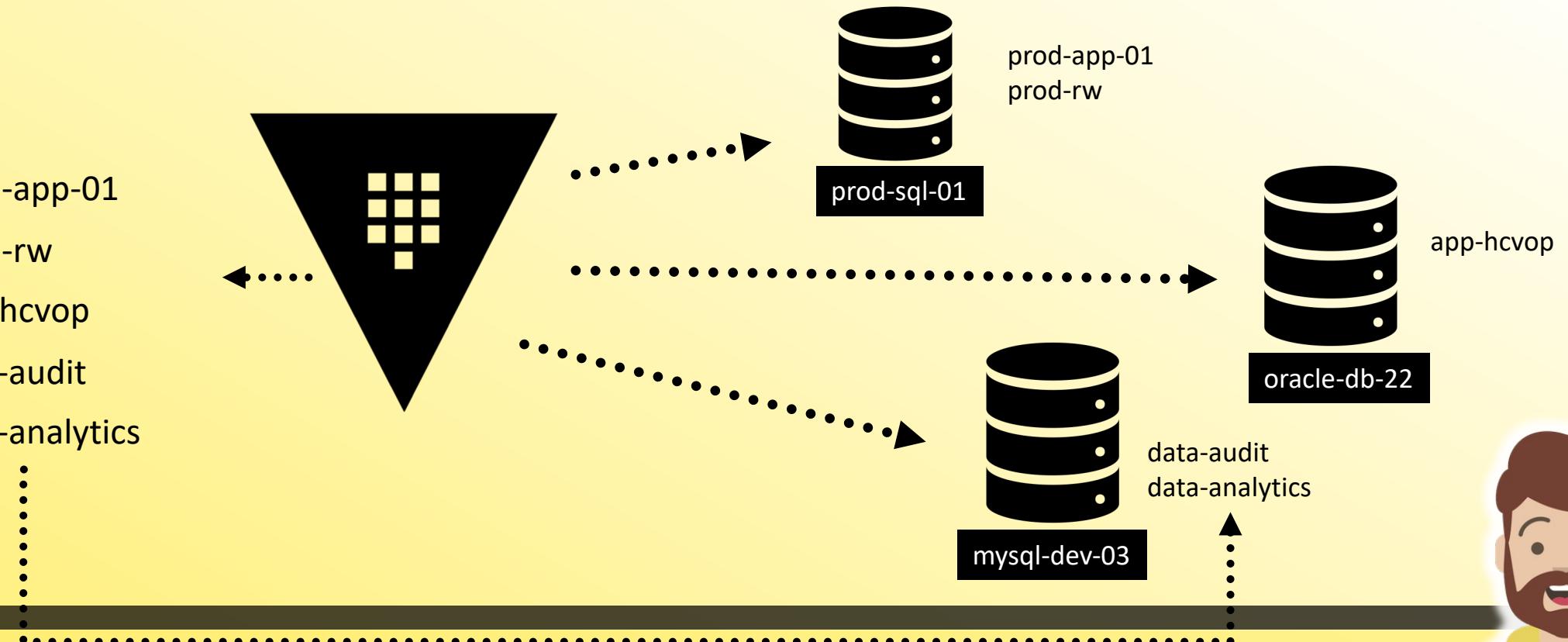
Database Secrets Engine - Roles



Configure roles based on permissions required

Roles

- prod-app-01
- prod-rw
- app-hcvop
- data-audit
- data-analytics



Database Secrets Engine – Create a Role

```
$ vault write database/roles/app-hcvop \
  db_name=prod_database \
  creation_statements="CREATE USER '{name}'@'%' IDENTIFIED
    BY '{password}';GRANT SELECT ON *.* TO '{name}'@'%';" \
  default_ttl="1h" \
  max_ttl="24h"
```



Database Secrets Engine – Create a Role

Breaking Down the Command



vault write database/roles/app-hcvop



Use write
when
creating a
new role



Database secrets
engine mount



Path where
roles are
created/stored



Name of the role to
be created



Database Secrets Engine – Create a Role

Important Parameters for a Role

db_name="prod_database"

Maps to the database config that you want to generate credentials on

creation_statements="CREATE USER '{ {name..}"

The statement to be executed on the database to create the user with the appropriate permissions

default_ttl="1h"

Initial Lease time for the generated dynamic credential

max_ttl="24h"

Make the lease renewable up to a certain timeframe



Static Roles



- Static role is a 1-1 mapping of a role to an existing, static user on the database.
- Vault will rotate the credential based on a configurable time period but the user does not change
- *Example Use Case:* Use this if an application requires a specific username to connect to the database but the password can change.



Password Policies



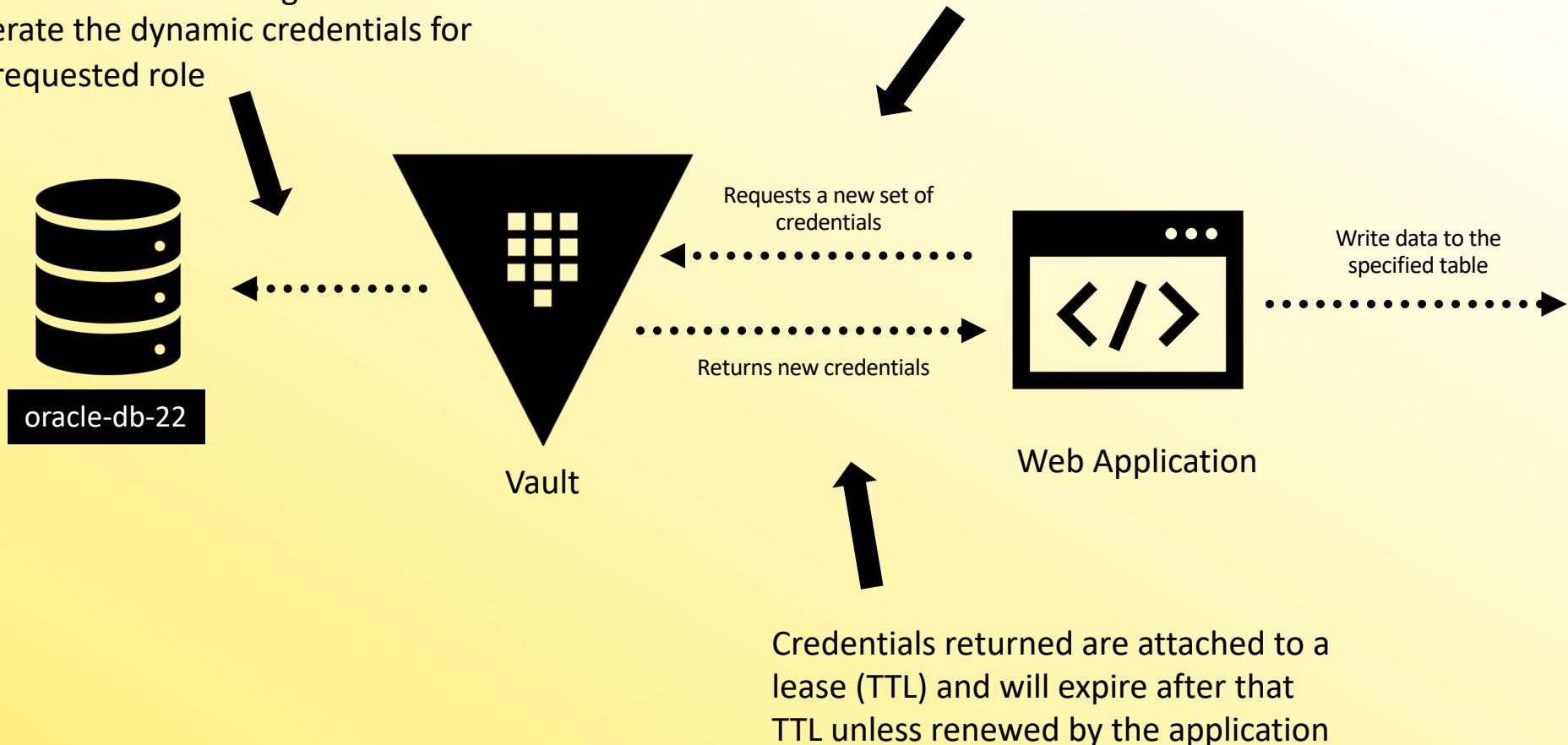
- Can set password policies per database configuration, not per role
- Each database has a default password policy defined as: 20 characters with at least 1 uppercase character, at least 1 lowercase character, at least 1 number, and at least 1 dash character.
- Note that some of the credentials generated by different plugins might be different due to the supported platform



Database Secrets Engine – Generate Creds



Vault uses the credentials we provided the database secrets engine to generate the dynamic credentials for the requested role



Credentials returned are attached to a lease (TTL) and will expire after that TTL unless renewed by the application



Database Secrets Engine – Generate Creds



```
$ vault read database/creds/app-hcvop <..... role we created

Key  Value
---  ---
lease_id          database/creds/app-hcvop/2f14c-4aa4b9-ad944a8d4de6
lease_duration    1h
lease_renewable   true
password          yRUSyd-vPYDg5NkU9kDg
username          V_VAULTUSE_APP_HCVOP_SJJUK3Q_KRAUSEN_8S62_160543009
```



Permissions for Generating Database Creds



- Like anything else in Vault, a Vault client requires a token with a policy that has permission to generate database credentials

```
# Get credentials from the database secrets engine
path "database/creds/app-hcvop" {
    capabilities = ["read"]
}
```





Identity Secrets Engine



Identity Secrets Engine

Identity secrets engine is the internal identity management solution for Vault

- Maintains clients recognized by Vault
- Each client is designated as an entity
- Operators can manage entities in Vault

The Identity secrets engine is mounted by default. It cannot be moved or disabled.



Vault Entities



Vault creates an entity and attaches an alias to it if a corresponding entity doesn't already exist.

- This is done using the Identity secrets engine, which manages internal identities that are recognized by Vault

An entity is a representation of a single person or system that logged into Vault. Each has a unique value. Each entity is made up of zero or more aliases

Alias is a combination of the auth method plus some identification. It is a mapping between an entity and auth method(s)

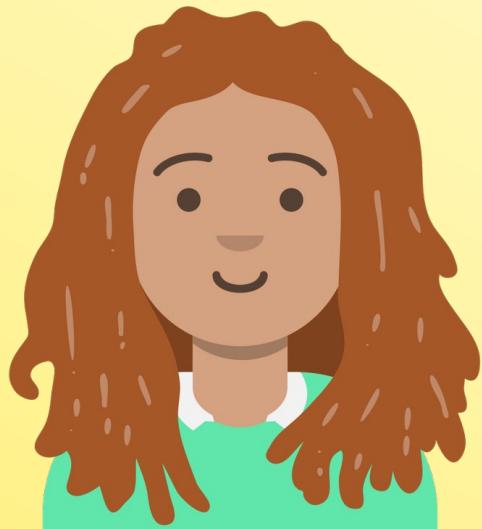


Vault Entities

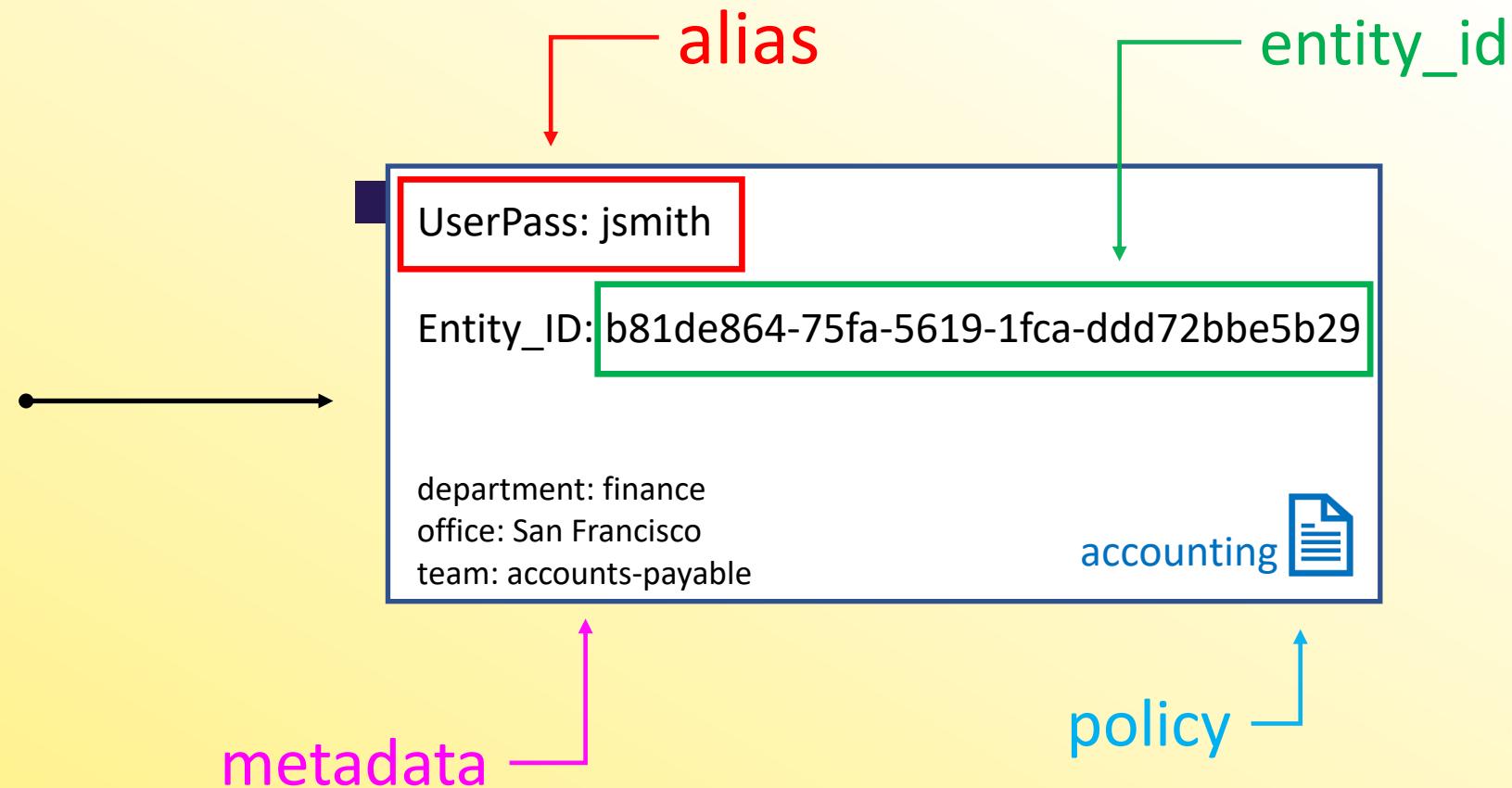


Julie Smith

Finance Specialist



UserPass

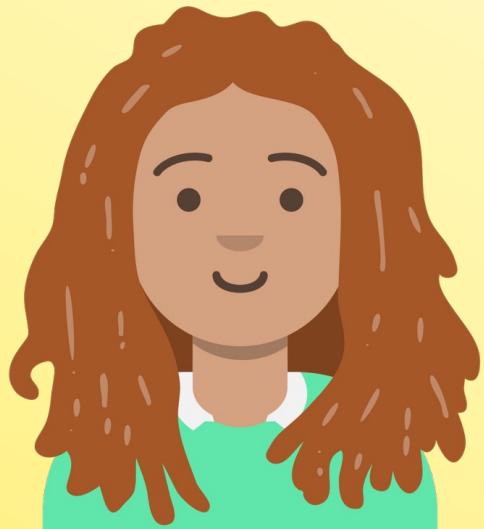


Vault Entities



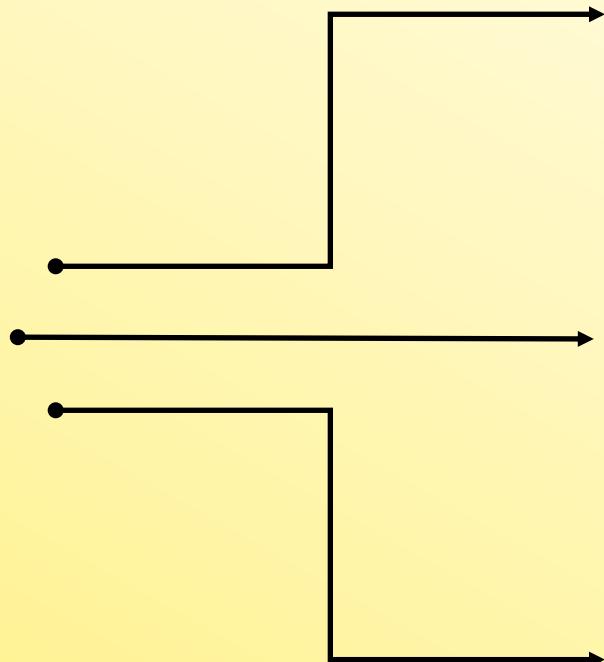
Julie Smith

Finance Specialist



Auth
Options:

UserPass
LDAP
GitHub



UserPass: jsmith
Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

department: accounting
sub-team: accounts-payable

accounting

LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b

department: finance
team: management

finance

GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa

location: us
sales-region: west

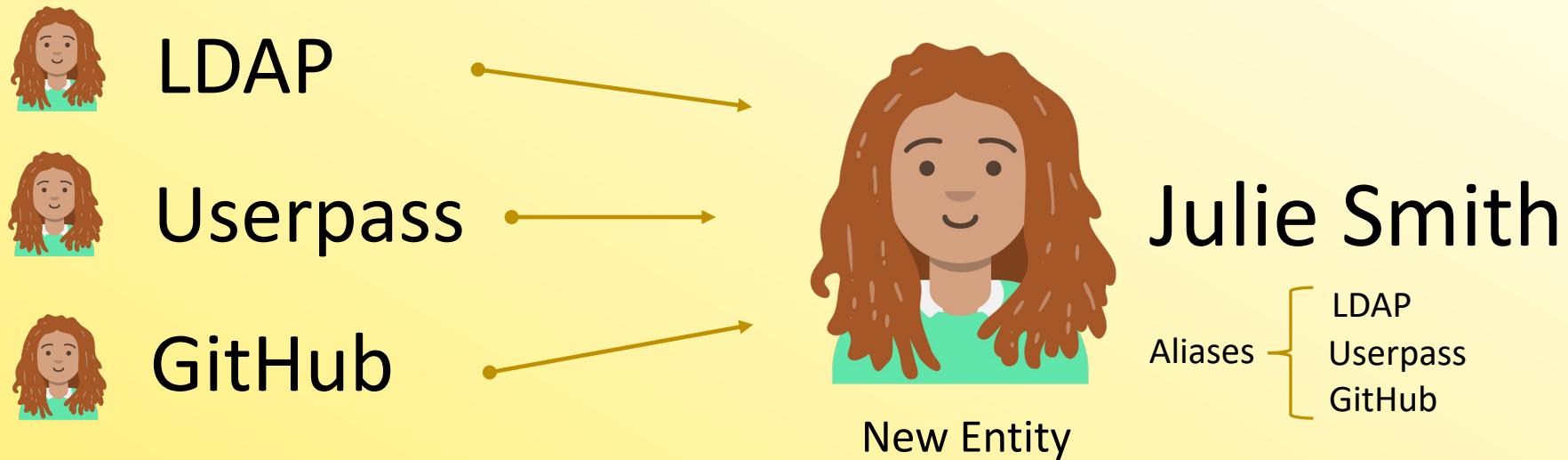
accounts payable



Consolidating Logins Under a Single Entity



- An entity can be manually created to map multiple entities for a single user to provide more efficient authorization management
- Any tokens that are created for the entity inherit the capabilities that are granted by alias(es).



Vault Entities



Entity →



Name: Julie Smith
Entity_ID: e48de234-58fa-0093-5fde-e5b99abe8b33
Policy: *management*

Aliases:



GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *finance*



LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *accounting*

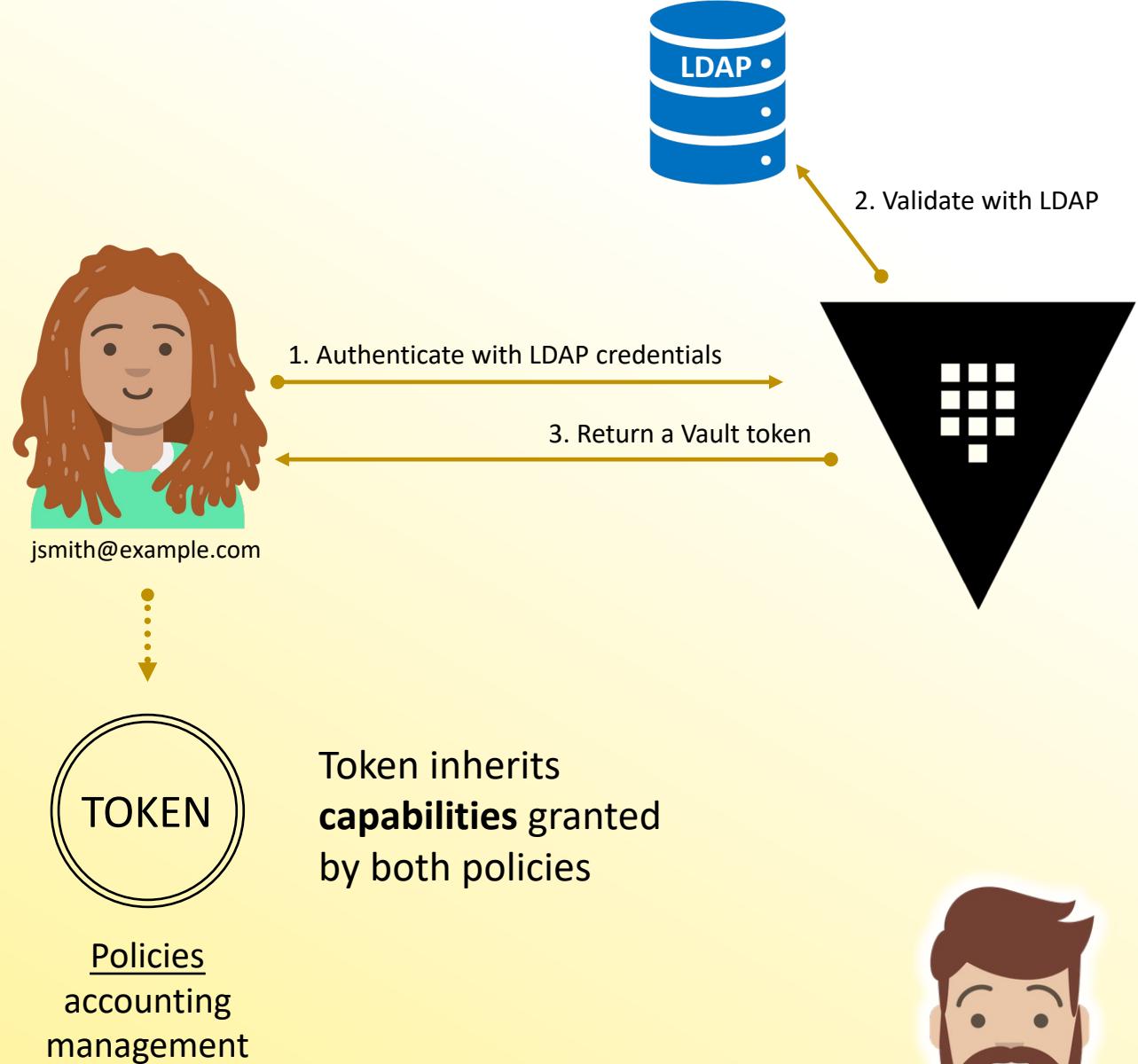
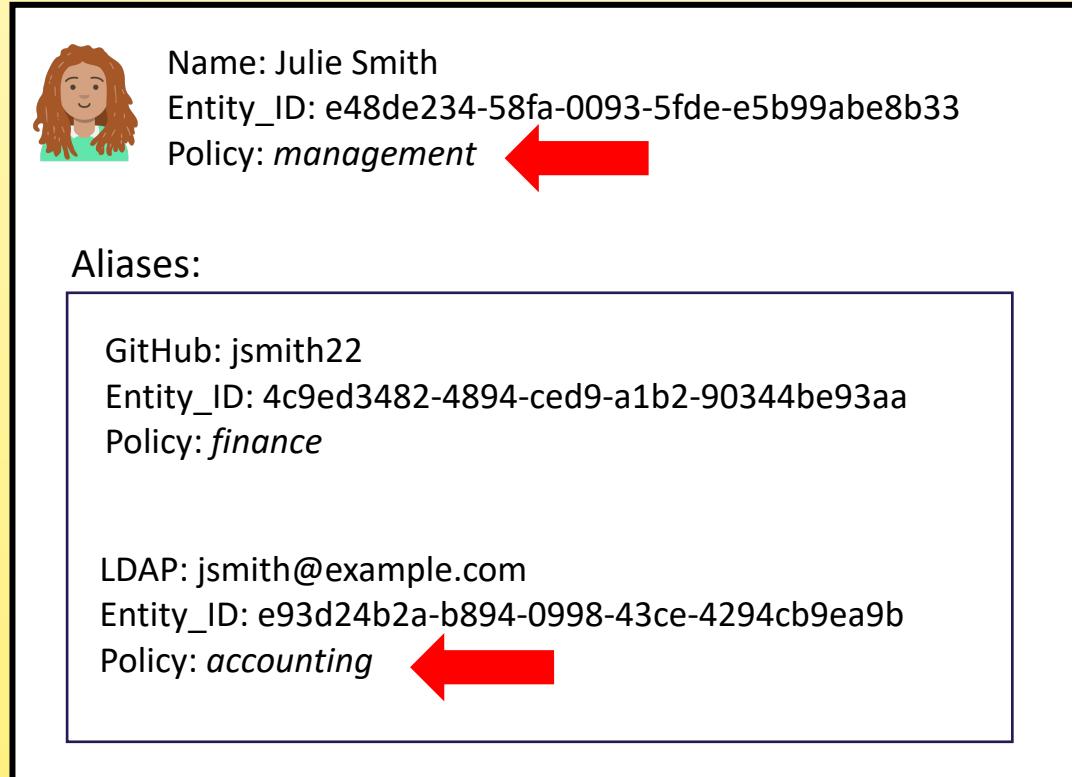


UserPass: jsmith
Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

Aliases



Vault Entities



Vault Groups



- A group can contain multiple entities as its members.
- A group can also have subgroups.
- Policies can be set on the group and the permissions will be granted to all members of the group.



Name: Finance_Team
Policy: *finance*

Members:



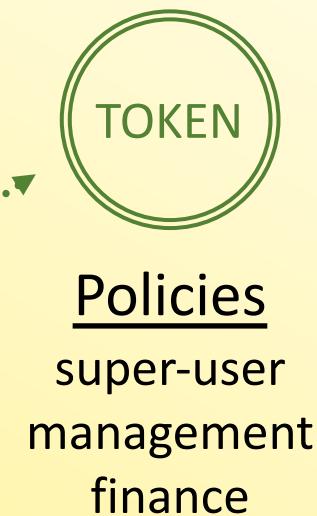
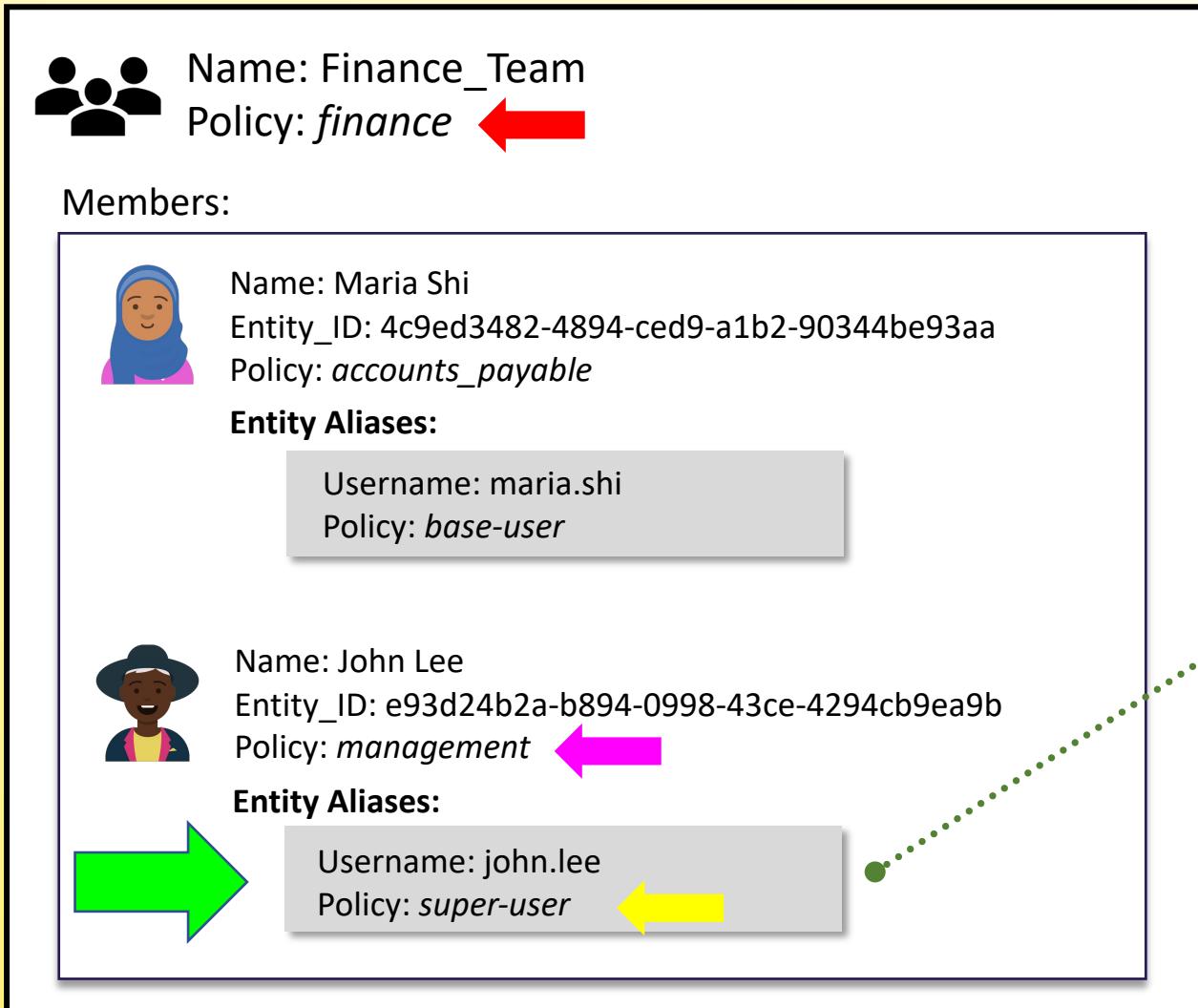
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: accounts_payable



Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: management



Vault Groups



Token inherits **capabilities** granted by alias, entity, and the group



Vault Groups – Internal vs. External



Internal Group

Groups created in Vault to group entities to propagate identical permissions

Created Manually

External Group

Groups which Vault infers and creates based on group associations coming from auth methods

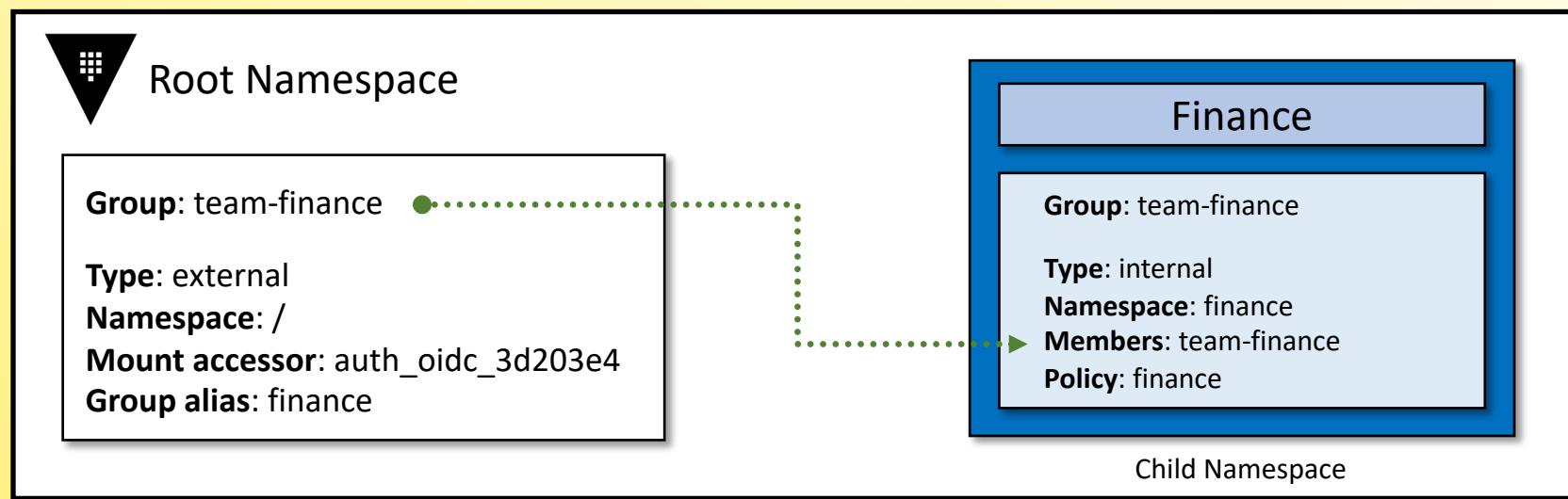
Created Manually or Automatically



Internal Groups

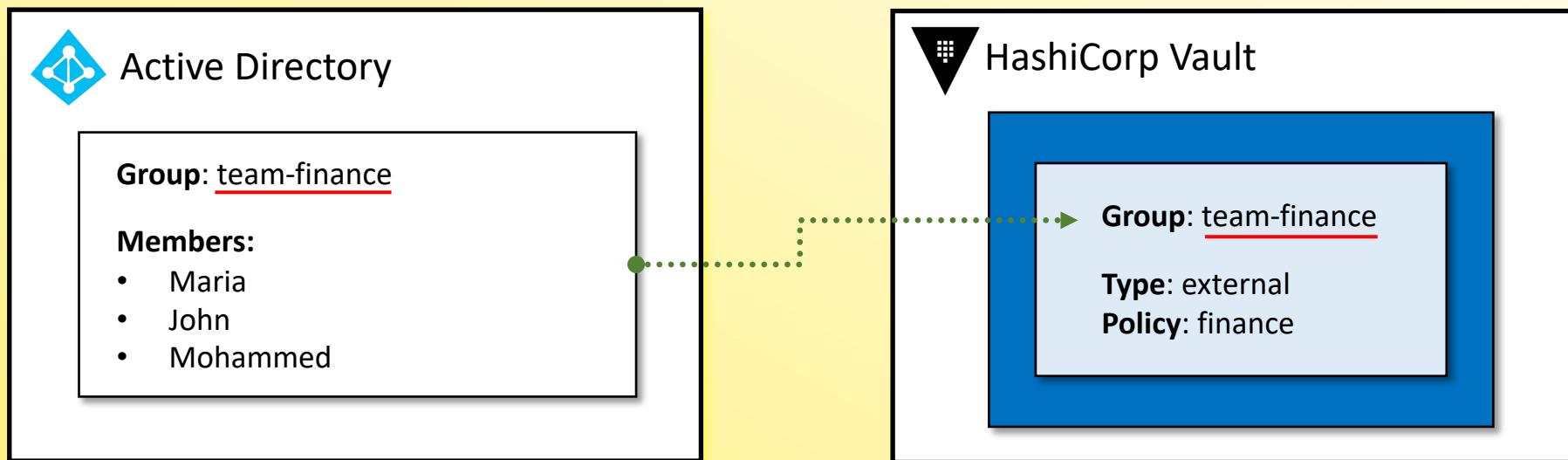


- Internal groups can be used to easily manage permissions for entities
- Frequently used when using Vault Namespaces to propagate permissions down to child namespaces
 - Helpful when you don't want to configure an identical auth method on every single namespace



External Groups

- External groups are used to set permissions based on group membership from an external identity provider, such as LDAP, Okta, or OIDC provider.
- Allows you to set up once in Vault and manage membership in the identity provider.
 - Note that the group name* must match the group name in your identity provider



*OIDC may require different configurations

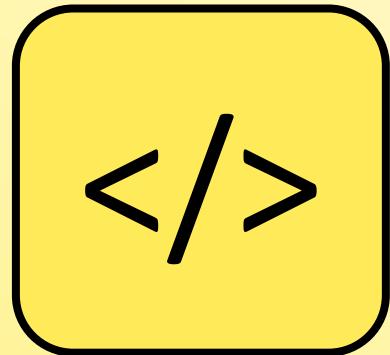




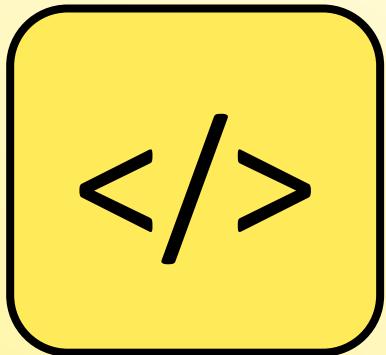
Transit Secrets Engine



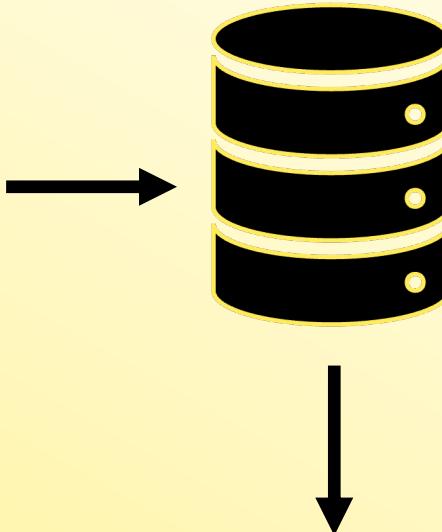
Problems with Encryption in the Enterprise



Web Tier



App Tier



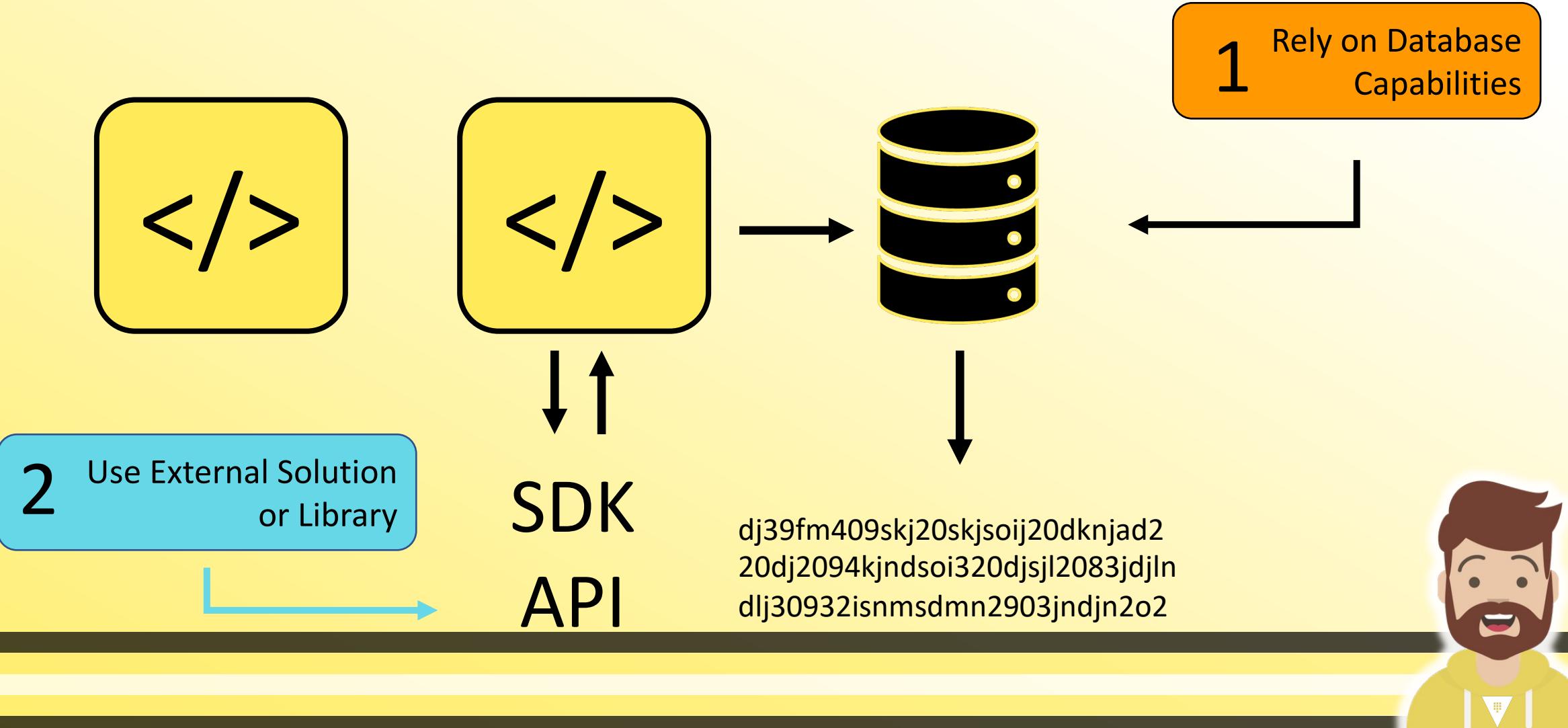
dj39fM40dky205kRAijSDknjad2
20dJ2094kj2345828fj1028B3djln
dljB0932i087sd30n2903jndjn2o2
d vMaP2-284n25-2010ifh3058bnls

Yes!
Encrypted is
Much Better!



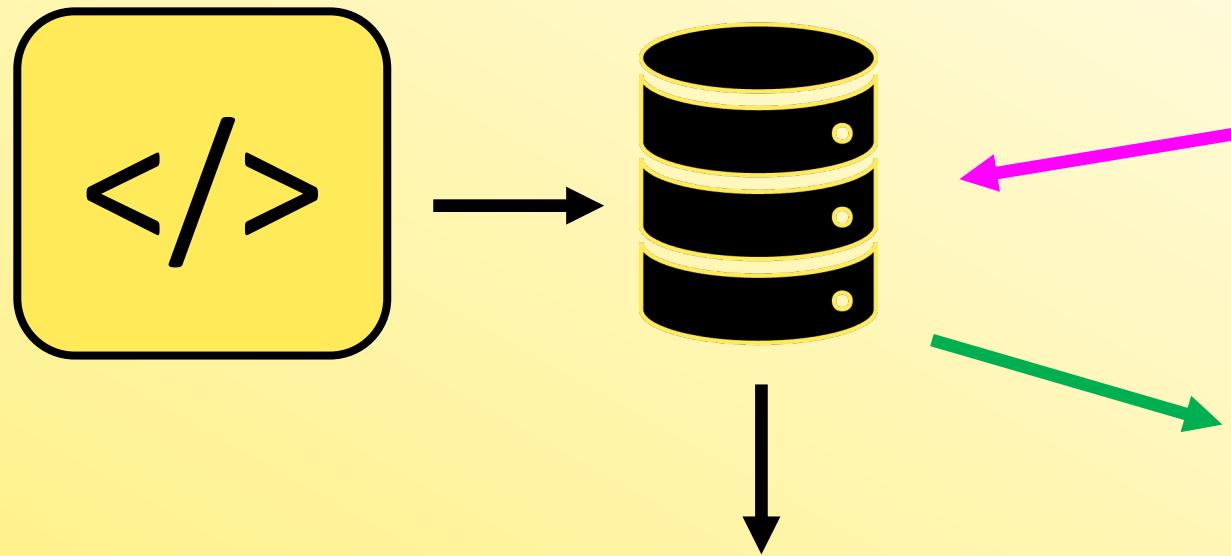
Problems with Encryption in the Enterprise

Options to Encrypt Data



Problems with Encryption in the Enterprise

Choosing a Database Platform for our App



dj39fm409skj20skjslij20dknjad2
20dj2094kjndsoi320djsjl2083jdjln
dlj30932isnmsdmn2903jndjn2o2

Ideal Database: Cassandra

but.....maybe it doesn't support the type/level of encryption we need

Required Database: MSSQL

Only MSSQL features meet the requirements to encrypt our data

NOT IDEAL

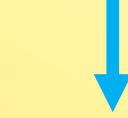
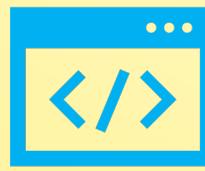


Problems with Encryption in the Enterprise

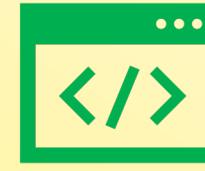
Putting the Responsibility on the Developers



OpenSSL



Golang



.NET



Internally
Developed



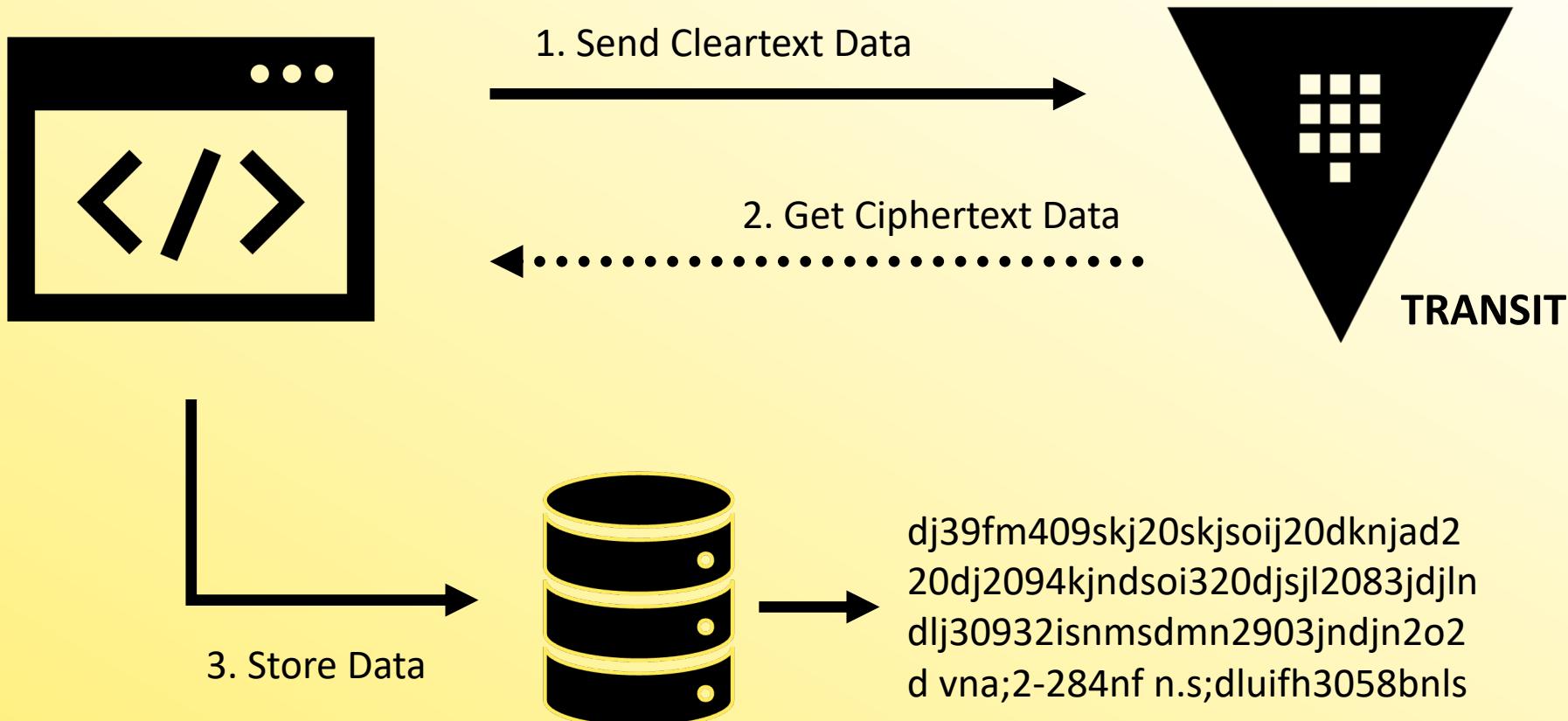
Voltage

Not
Good!



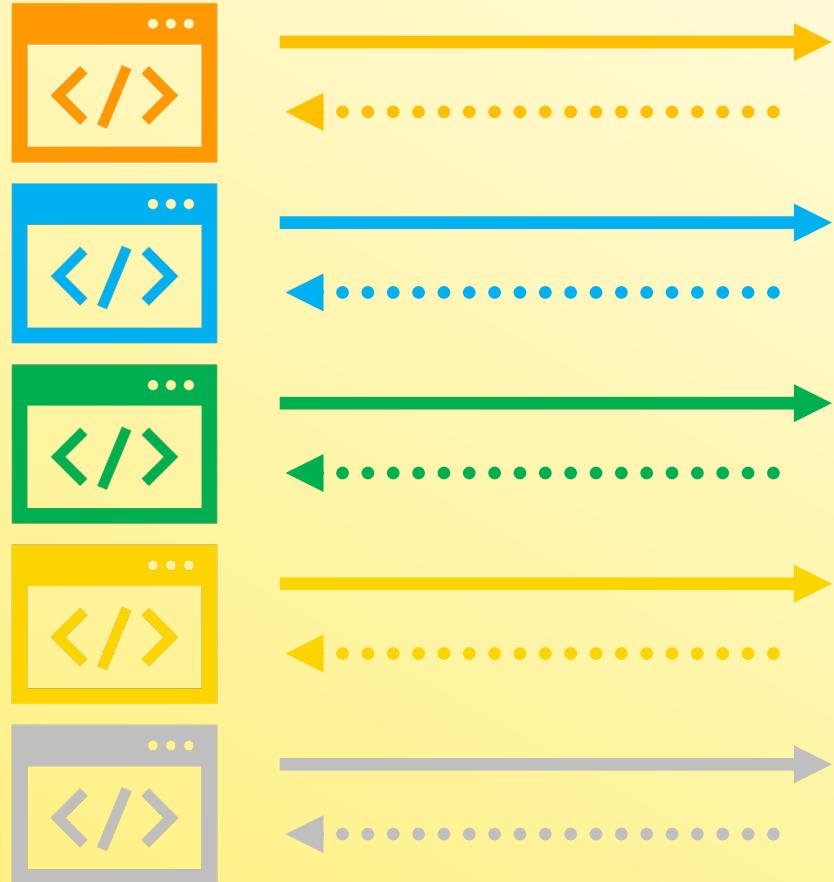
Solution?

Use Vault's Transit Secrets Engine



Solution

Centralize the Organization's Encryption Needs



Intro to Transit Secrets Engine



Transit secrets engine provides functions for encrypting/decrypting data

- Enables organizations to outsource/centralize encryption to Vault

Applications can send cleartext data to Vault for encryption

- Vault encrypts using the specified key and returns ciphertext to the app
- The application NEVER has access to the encryption key (stored in Vault)
- Decouples storage from encryption and access control

Transit can also provide auto unseal capabilities to other Vault clusters

- *More information in the auto unseal section*





Transit secrets engine
DOES NOT STORE the
encrypted data



Intro to Transit Secrets Engine



Encryption keys are created and stored in Vault to process data

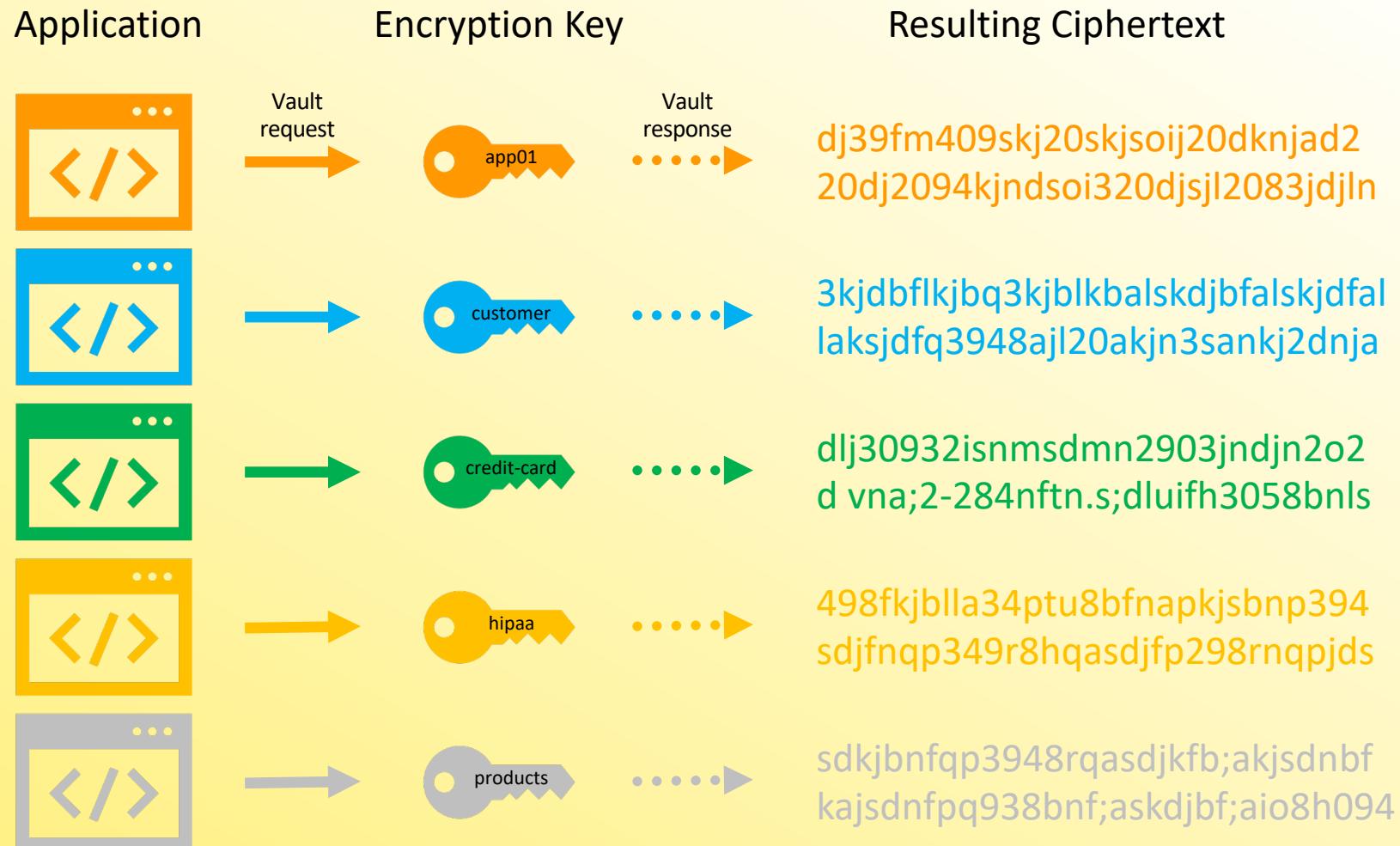
- Each application can have its own encryption key (or more!)
- Apps must have permission to use the key for encryption/decryption operations, which is bound by the policy attached to its token

Keys can be easily rotated as often as needed

- Keys are stored on keyring 
- Can limit what version(s) of keys can be used for decryption
- You can create, rotate, delete, and export a key (need permissions)
- Easily rewrap ciphertext with a newer version of a key



Intro to Transit Secrets Engine



Encryption Key Types



Key Type	Description
aes128-gcm96	AES-GCM with a 128-bit AES key and a 96-bit nonce; supports encryption, decryption, key derivation, and convergent encryption
aes256-gcm96	AES-GCM with a 256-bit AES key and a 96-bit nonce; supports encryption, decryption, key derivation, and convergent encryption (default)
chacha20-poly1305	ChaCha20-Poly1305 with a 256-bit key; supports encryption, decryption, key derivation, and convergent encryption
ed25519	Ed25519; supports signing, signature verification, and key derivation
ecdsa-p256	ECDSA using curve P-256; supports signing and signature verification
ecdsa-p384	ECDSA using curve P-384; supports signing and signature verification
ecdsa-p521	ECDSA using curve P-521; supports signing and signature verification
rsa-2048	2048-bit RSA key; supports encryption, decryption, signing, and signature verification
rsa-3072	3072-bit RSA key; supports encryption, decryption, signing, and signature verification
rsa-4096	4096-bit RSA key; supports encryption, decryption, signing, and signature verification



Intro to Transit Secrets Engine



Vault also supports convergent encryption mode

- Means that every time you encrypt the same data, you'll get the same ciphertext back
- This enables you to have searchable ciphertext

Encrypting Data

- All plaintext data must be base64-encoded
- This is because Vault doesn't require that the plaintext is "text" only. It could be a file such as a PDF or image
- But...please understand that base64-encoding is NOT encryption

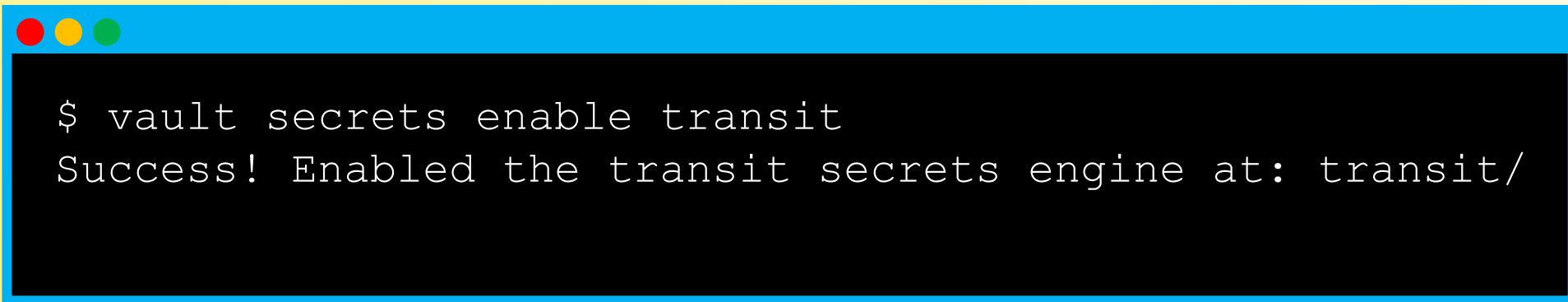


Enable the Transit Secrets Engine



Before you can use the Transit secrets engine to encrypt data, it must first be enabled

- Can use the default path of transit or enable on another path

A screenshot of a terminal window with a blue header bar. The window shows the command "\$ vault secrets enable transit" being run, followed by the output "Success! Enabled the transit secrets engine at: transit/".

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



Create an Encryption Key



The next step is to create one or many encryption keys used to encrypt/decrypt data

A terminal window with a purple title bar showing red, yellow, and green window control buttons. The main area contains the following text:

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

Note: `-f` is shorthand for "force" and is needed on some Vault commands



Create an Encryption Key



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

Custom Key Type

```
$ vault write -f transit/keys/training_rsa type="rsa-4096"  
Success! Data written to: transit/keys/training_rsa
```



Encrypt Data with a Key



Pass the cleartext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/encrypt/training \
  plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")

Key          Value
---          -----
ciphertext   vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEaha15LhDPSoN6HkTOhwn79DCwt0m...
key_version  1
```

*output truncated



Breaking Down the Command for Encryption



```
vault write transit/encrypt/training
```



Sub-command



Path where the
Transit secrets
engine was mounted
(enabled)



The desired
action



The encryption key
you want to use



Breaking Down the Parameter



```
plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")
```

Required
Parameter for
Encryption
operation
(what are we
going to encrypt?)

Base64 encode the string "Getting Started with
HashiCorp Vault"

If your data is already base64 encoded, you
can just pass it as is....you don't need to
encode it again



Breaking Down the Vault Response



Key	Value
---	-----
ciphertext	vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQ.....
key_version	1

This is the data that you would store in a database, file store, or anywhere so your application or organization can read later

The Key Version is built right into the ciphertext

Output abbreviated



Decrypting Data



Pass the ciphertext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/decrypt/training \
  ciphertext="vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEaha15LhDPSoN6HkTO....."

Key          Value
---          -----
plaintext   R2V0dGluZyBTdGFydGVkIHdpdGggSGFzaG1Db3JwIFZhdWx0Cg==
```





Rotating Encryption Keys



Key Rotation



Transit allows for a simplified key rotation process

- keys can be rotated manually or by an automated process
- Vault v1.10 includes the ability to set a rotation period

Vault maintains a versioned keyring

- All versions of the encryption key are stored
- Vault admins can limit the minimum key version allowed to be used for decryption operations (older keys won't work)
- You can `rewrap` encrypted data (ciphertext) to use a newer version of the encryption key



How to Rotate an Encryption Key



Pass the ciphertext data to Vault specifying the action and desired encryption key to use

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



Reading a Key after Rotation



```
$ vault read transit/keys/training
Key          Value
---          -----
allow_plaintext_backup    false
auto_rotate_period        8760h
deletion_allowed          false
derived                  false
exportable                false
keys                      map[1:1647960245 2:1647960257 3:1647961177]
latest_version            3
min_available_version     0
min_decryption_version   1
min_encryption_version   0
name                     training
supports_decryption      true
supports_derivation      true
supports_encryption      true
supports_signing         false
type                     aes256-gcm96
```

A green arrow points to the value '3' under the 'latest_version' key. A purple bracket labeled 'keyring' encloses the three values in the 'keys' map entry.



Encryption Key Configuration



We can limit what version of the key can be used to decrypt data

- Maybe we have old data that we have converted and don't want anybody to be able to decrypt it
- This is configured using the minimum key version configuration
- It can be configured for each encryption key (not key version)



Encryption Key Configuration

Minimum
Key Version = 1



vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjh4b9dbnajksbr3948f
dja;siurn4398ebjkbalks
jdbnfp3948fbhdjkbal4
8rh4938ebaskjbq9384
hbfjdblaw948rhfjasdu



vault:v2:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjNdf230r89qj
vnmndf148fnadjkvN23
8rhqeojdfnvq3948rhq
39ruincsajkvnpq9384h
qpjisnv;aksjdnf9q384h
fjdsnlaisjnfq9384hr938



vault:v3:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfq93u4fbalskjdnB
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf
pq938rfhpqekjfdbvlaks
jdnvq3948fhgfdjabnlsk
rjbnre3pq9uesdkjnrpfi
unaklns394njksalskern



vault:v4:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdoobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl
akjbpaieurhq83urbgl
kjbvpqieurbgpqijfbvlaif
uhpq0w9eurbfpakjdbf
vlakdjfbngqiasdf34dsfa



Encryption Key Configuration



vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f
dja;siurn4398ebjkbalks
jdbnfp3948fbhdjkbal4
8rh4938ebaskjbq9384
hbfjdblaw948rhfjasdu



vault:v2:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23
8rhqeojdfnvq3948rhq
39ruincsajkvnpq9384h
qpjisnv;aksjdnf9q384h
fjdsnlaisjnfq9384hr938



vault:v3:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfqp93u4fbalskjdn
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf
pq938rfhpqekjfdbvlaks
jdnvq3948fhgfdjabnlsk
rjbnre3pq9uesdkjnrpfi
unaklns394njksalskern



Minimum
Key Version = 4



vault:v4:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdocsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl
akjbpaieurhq83urbgl
kjbvpqieurbgpqijfbvlaif
uhpq0w9eurbfpakjdbf
vlakdjfbngqiasdf34dsfa



Setting the Minimum Decryption Version



```
$ vault write transit/keys/training/config \
  min_decryption_version=4

Success! Data written to: transit/keys/training/config
```



Setting the Minimum Encryption Version



```
$ vault read transit/keys/training
Key          Value
---          -----
allow_plaintext_backup    false
auto_rotate_period        8760h
deletion_allowed          false
derived                  false
exportable                false
keys                     map[4:1647962305]
latest_version            4
min_available_version     0
min_decryption_version   4 ←
min_encryption_version   0
name                     training
supports_decryption      true
supports_derivation      true
supports_encryption      true
supports_signing         false
type                     aes256-gcm96
```

Only the key(s) equal or greater than the minimum key version are available

A green arrow points to the value "4" under "min_encryption_version". A pink arrow points to the value "4" under "min_decryption_version".



Permissions Required for Transit



Clients need permission to use certain encryption keys and specific operations related to each key

```
# Permit Encryption Operations
path "transit/encrypt/training" {
    capabilities = ["update"]
}

# Permit Decryption Operations
path "transit/decrypt/training" {
    capabilities = ["update"]
}
```

This same policy can be used for
Transit auto unseal operations
using the training key

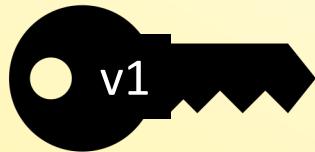




Rewrapping Ciphertext



Rewrapping Ciphertext



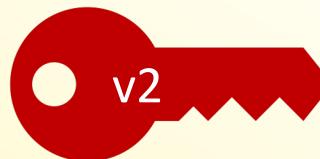
vault:v1:jnaclvjabenpri
ugbdkjbpiiaeurbnlkjcab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f

vault:v1:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmmndf148fnadjkvn23

Data Encrypted in 2019

....but our key has been rotated 3 times since....

Rotated 2020



Rotated 2021



Rotated 2022



How can we upgrade our encrypted data to be
encrypted by the latest version of the key?



Rewrapping Ciphertext



```
$ vault write transit/rewrap/training \
  ciphertext="vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEahal5LhDPSoN6
  HkTOhwn79DCwt0mct1ttLokqikAr0PAopzm2jQAKJg=2/QGPTMnzKPlw4cCPGTbkzE
  PlX5OyPkLIgX+erFWdUXKKUIEb6D2Gm5ZjTaola314LsVkbLF5G1RkBTAActskk="

Key          Value
---          -----
ciphertext   vault:v4:RFzp1kMpjtUIiS+6qxrnjIJEdPqCepFUa2ivr70.....
key_version  4
```

A pink arrow points to the command line, and a blue arrow points to the "key_version" value of 4.

The data was never available in plaintext when rewrapping the data with the latest version of the key





PKI Secrets Engine

PKI Secrets Engine



Generates dynamic X.509 certificates

- Eliminates the manual process for generating a private key and CSR, submitting to the CA, and waiting for verification and signing
- Vault's built-in authentication (auth method) and authorization mechanisms (ACLs) provide the verification to generate certificates
- Can only have one CA certificate per PKI secrets engine
 - If you need to issue certs from multiple CAs, use multiple PKI secrets engines on different paths

PKI secrets engine is use for INTERNAL certificates



PKI Secrets Engine



Allows certs to have short TTLs because certs are now ephemeral

- Virtually eliminates revocations
- Applications can obtain certificate at runtime and simply discard the certificate at shutdown
- Simple to allocate a certificate to each workload
- Stops the certificate sharing, use of wildcard certificates, or using self-signed certificates which are prone to MITM attacks



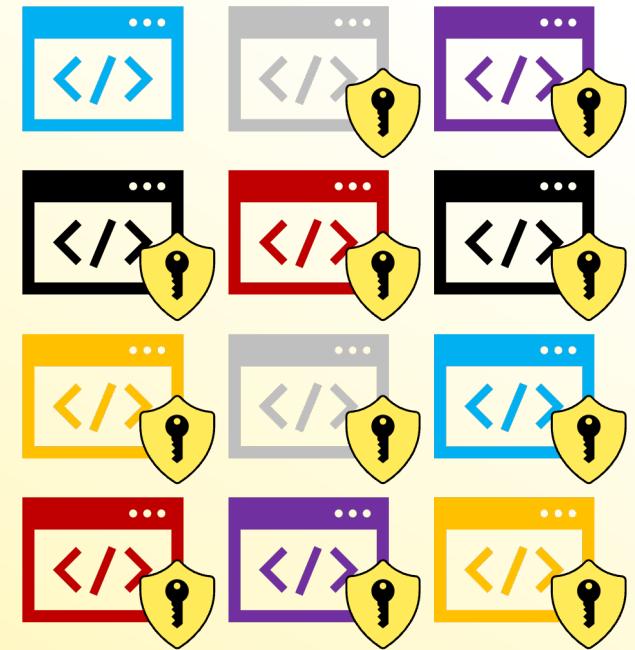
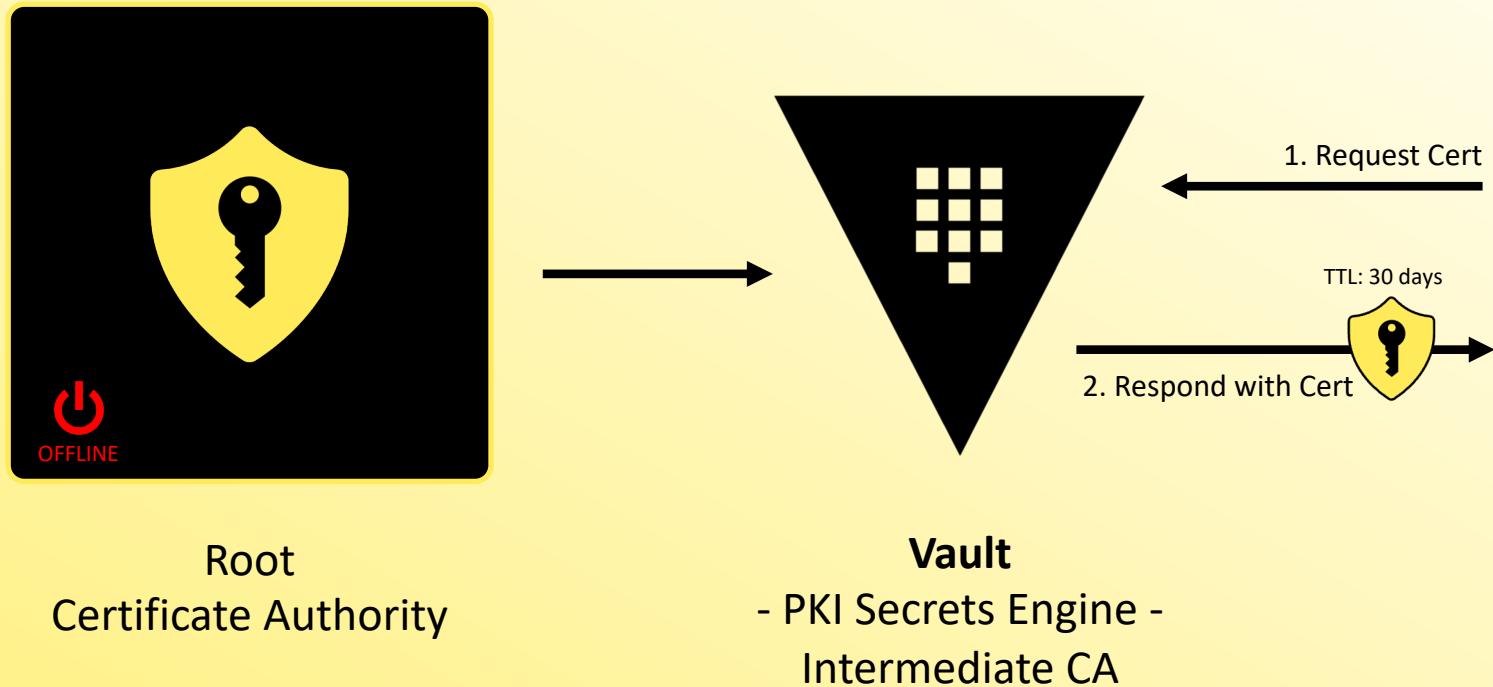
PKI Secrets Engine



- Most likely, you're going to use Vault as an Intermediate as your root CA is generally offline (*industry best practices*)
- Most organizations already have an existing CA structure in place and Vault can plug right into that, if needed.
- Using multiple PKI secrets engines, Vault can perform the root and the intermediate(s) CA functionality from the same cluster



Common Architecture



Enable the PKI Secrets Engine



```
# Enable using default path of pki/
$ vault secrets enable pki
Success! Enabled the pki secrets engine at: pki/

# Enable using custom path of hcvop-int/
$ vault secrets enable -path=hcvop_int pki
Success! Enabled the pki secrets engine at: hcvop_int/
```



Tune the PKI Secrets Engine



```
# Set a maximum TTL for certificates to 30 days
$ vault secrets tune -max-lease-ttl=720h pki
Success! Tuned the secrets engine at: pki/

# Set a maximum TTL for certificates to 1 year
$ vault secrets tune -max-lease-ttl=8760h hcvop_int
Success! Tuned the secrets engine at: pki/
```



Create an Intermediate Certificate for Signing



```
# Generate an Intermediate CSR
$ vault write -format=json \
  hcvop_int/intermediate/generate/internal \
  common_name="hcvop.com Intermediate" \
  | jq -r '.data.csr' > pki_intermediate.csr

# At this point, you need to sign the intermediate with the root CA

# Import signed certificate from the root
$ vault write pki_int/intermediate/set-signed \
  certificate=@intermediate.cert.pem
```



Configure the CA and CRL URLs



```
# These are URLs that certificates will contain, and hosts will use to validate whether the certs are valid or revoked

$ vault write pki/config/urls \
  issuing_certificates="https://vault.hcvop.com:8200/v1/pki/ca" \
  crl_distribution_points="https://vault.hcvop.com:8200/v1/pki/crl"
```



Intro to PKI Roles



A role provides a one-to-one mapping between a policy and a configuration in a secrets engine

- In the PKI secrets engine, a role is configured for each configuration based on parameters that control certificate common names, alternate names, and other parameters
- For each unique certificate type or configuration, you'll need to create a unique role
- **Notable configurations include:**
 - allowed_domains
 - allow_bare_domains
 - allow_subdomains
 - allow_glob_domains



Defining Unique PKI Roles



web_dmz_role

Role Configuration

```
allowed_domains=dmz.hcvop.com  
allow_subdomains=true  
allow_bare_domains=false  
allow_glob_domains=false  
max_ttl=720h
```

server1.dmz.hcvop.com
dmzweb01.dmz.hcvop.com

internal_apps_role

Role Configuration

```
allowed_domains=app.hcvop.com  
allow_subdomains=true  
max_ttl=24h
```

benefits.app.hcvop.com
payroll.app.hcvop.com

k8s_apps_role

Role Configuration

```
allowed_domains=k8s.hcvop.com  
allow_subdomains=true  
allow_glob_domains=false  
max_ttl=4h
```

certs.k8s.hcvop.com
license.k8s.hcvop.com



Create a Role



```
# Create a new web DMZ role

$ vault write pki/roles/web_dmz_role \
    allowed_domains=dmz.hcvop.com \
    allowed_subdomains=true \
    allow_bare_domains=false \
    max_ttl=720h \
    allow_localhost=true \
    organization=hcvop \
    country=us
```



Create a Role



```
vault write pki/roles/web_dmz_role
```

Sub-command

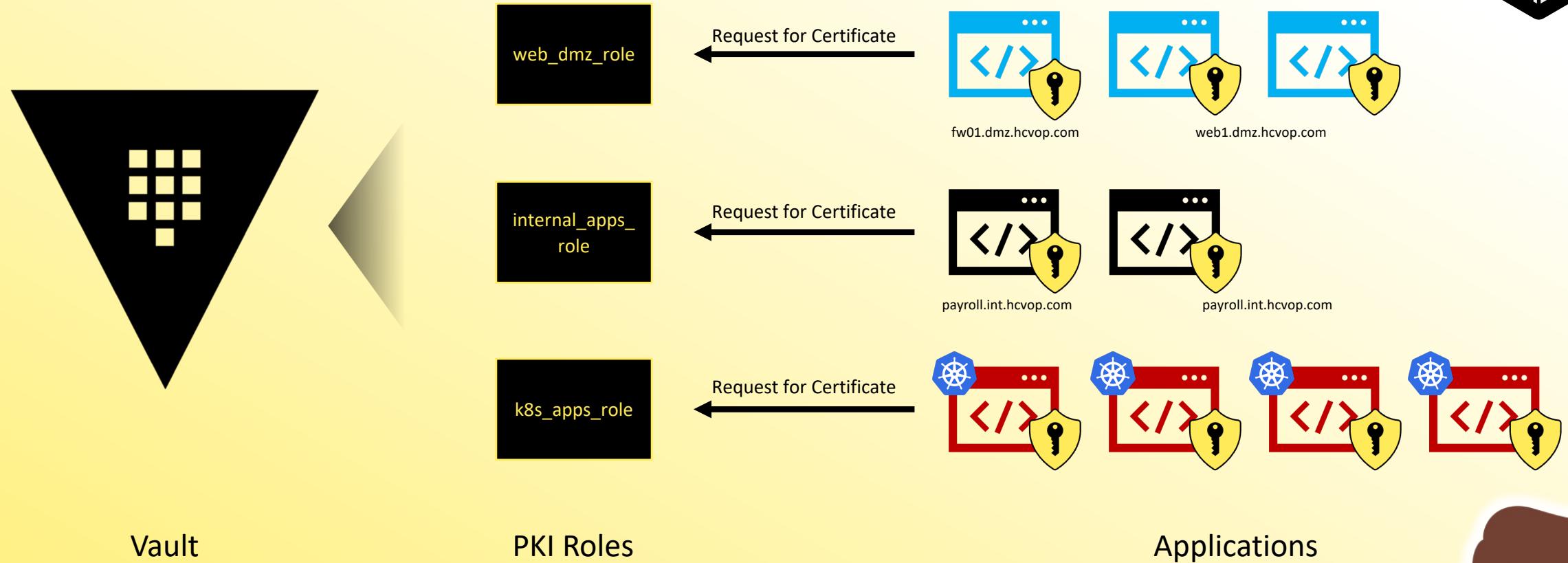
Path where
the PKI
secrets
engine was
mounted
(enabled)

Path where
roles are
created/stored

Name of the role we
want to create or
modify



Create a Role



Generate a Certificate



```
# Generate a new certificate for web_dmz_role

$ vault write pki/issue/web_dmz_role \
    common_name=dmzhcp01.dmz.hcvop.com \
    alt_names=portal.dmz.hcvop.com \
    max_ttl=720h

Key          Value
---          -----
certificate  -----BEGIN CERTIFICATE-----
MIIDwzCCAqgAwIBAgIUTQABMCAsXjG6ExFTX8201xKvh4IwDQYJKoZIhvcNAQELBQA...  
DTE4MDcyNDIxMTMxOVox ...
-----END CERTIFICATE-----

issuing_ca    -----BEGIN CERTIFICATE-----MIIDQTCCAimgAwIBAgIUbMYp39mdj7dKX033Zjk18...  
NAQEL ...
-----END CERTIFICATE-----

private_key    -----BEGIN RSA PRIVATE KEY----MIIEowIBAAKCAQEAtelfqy2Ekj+EFqKV6N5QJ1BgMo/U4IIxwLZI6a87yA  
C/rDhm W58liadXrwjzRgWeqVOoCRr/B5JnRLbyIKBVp6MMFwZVky...  
nyuomSfJkM ...
-----END RSA PRIVATE KEY-----

private_key_type rsa
serial_number  4d:00:01:30:20:2c:5e:31:ba:13:11:53:5f:cd:b4:d7:12:95:1f:82
```

This is the ONLY time you will get the private key so make sure to save it



Generate a Certificate



```
vault write pki/issue/web_dmz_role
```

Sub-command

Path where
the PKI
secrets
engine was
mounted
(enabled)

Path used to
generate
certificates

Name of the role we
want to use to create
our new certificate



Revoke a Certificate



```
# Revoke a single certificate

$ vault write pki/revoke serial_number="4d:00:01:30:20:2c:5e:31:ba:a9:7b"

Key                                Value
---                                -----
revocation_time                    1628908066
revocation_time_rfc3339            2022-12-25T06:11:32.676612Z
```



Remove Revoked and Expired Certificates



```
# Keep the storage backend clean by periodically removing certs  
  
$ vault write pki/tidy tidy_cert_store=true tidy_revoked_certs=true  
WARNING! The following warnings were returned from Vault:  
  
* Tidy operation successfully started. Any information from the operation  
will be printed to Vault's server logs.
```





Cubbyhole Secrets Engine



Where Does 'Cubbyhole' Come From?



Intro to Cubbyhole

Cubbyhole Secrets Engine is used to store arbitrary secrets

- ▶ Enabled by default at the **cubbyhole/** path
- ▶ Its lifetime is linked to the **token** used to write the data
 - No concept of a time-to-live (TTL) or refresh interval for values in cubbyhole
 - Even the **root** token cannot read the data if it wasn't written by the root

Cubbyhole Secrets Engine **cannot** be disabled, moved, or enabled multiple times



Service Tokens Have a Cubbyhole

- Each service token gets its own cubbyhole
- A token cannot access another token's cubbyhole
- Cubbyhole expires when the token does



Viewing Cubbyhole Secrets Engine

```
$ vault secrets list
```

Path	Type	Accessor	Description
---	---	-----	-----
cloud/	kv	kv_dd590f0e	This is an KV secret engine mount
cubbyhole/	cubbyhole	cubbyhole_9c6c2ca2	per-token private secret storage
identity/	identity	identity_e55fbf01	identity store
kv/	kv	kv_ed482380	n/a
kvv2/	kv	kv_0559442e	n/a



Writing and Read Data to Cubbyhole via CLI

```
# Write data to Cubbyhole
$ vault write cubbyhole/training certification=hcvop
Success! Data written to: cubbyhole/training
```

```
# Read data from Cubbyhole
$ vault read cubbyhole/training
Key          Value
---          -----
certification    hcvop
```



Writing and Read Data to Cubbyhole via API

```
# Write data to cubbyhole via API
$ curl --header "X-Vault-Token: hvs.QRx4pz2RIka7RhhrjiVRBNjq" \
  --request POST
  --data '{"certification": "hcvop"}'
https://vault.training.com:8200/v1/cubbyhole/training

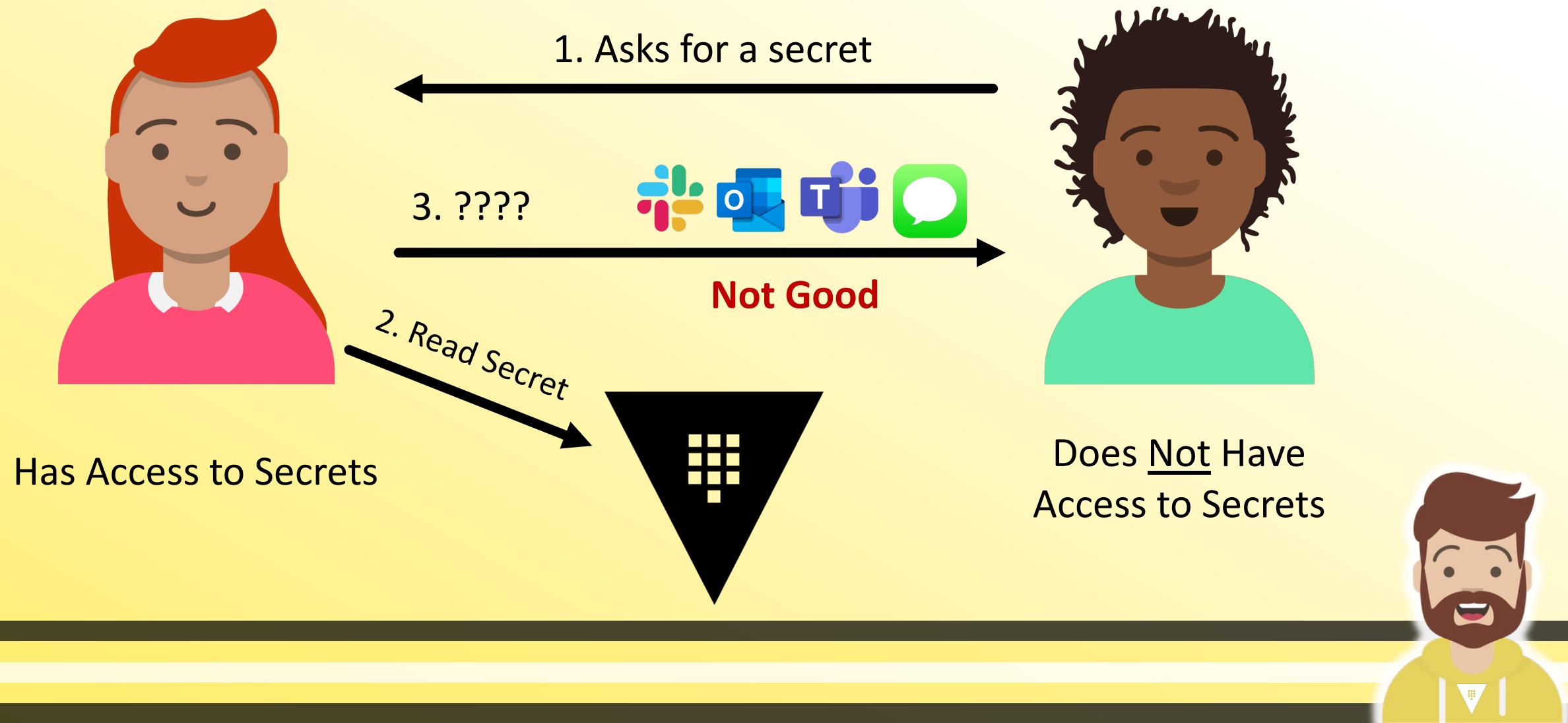
# Read secret from cubbyhole via API
$ curl --header "X-Vault-Token: hvs.QRx4pz2RIka7RhhrjiVRBNjq" \
  https://vault.training.com:8200/v1/cubbyhole/training
```





Cubbyhole Doesn't Seem
Very Useful....Yet....

How Do You Share Secrets?



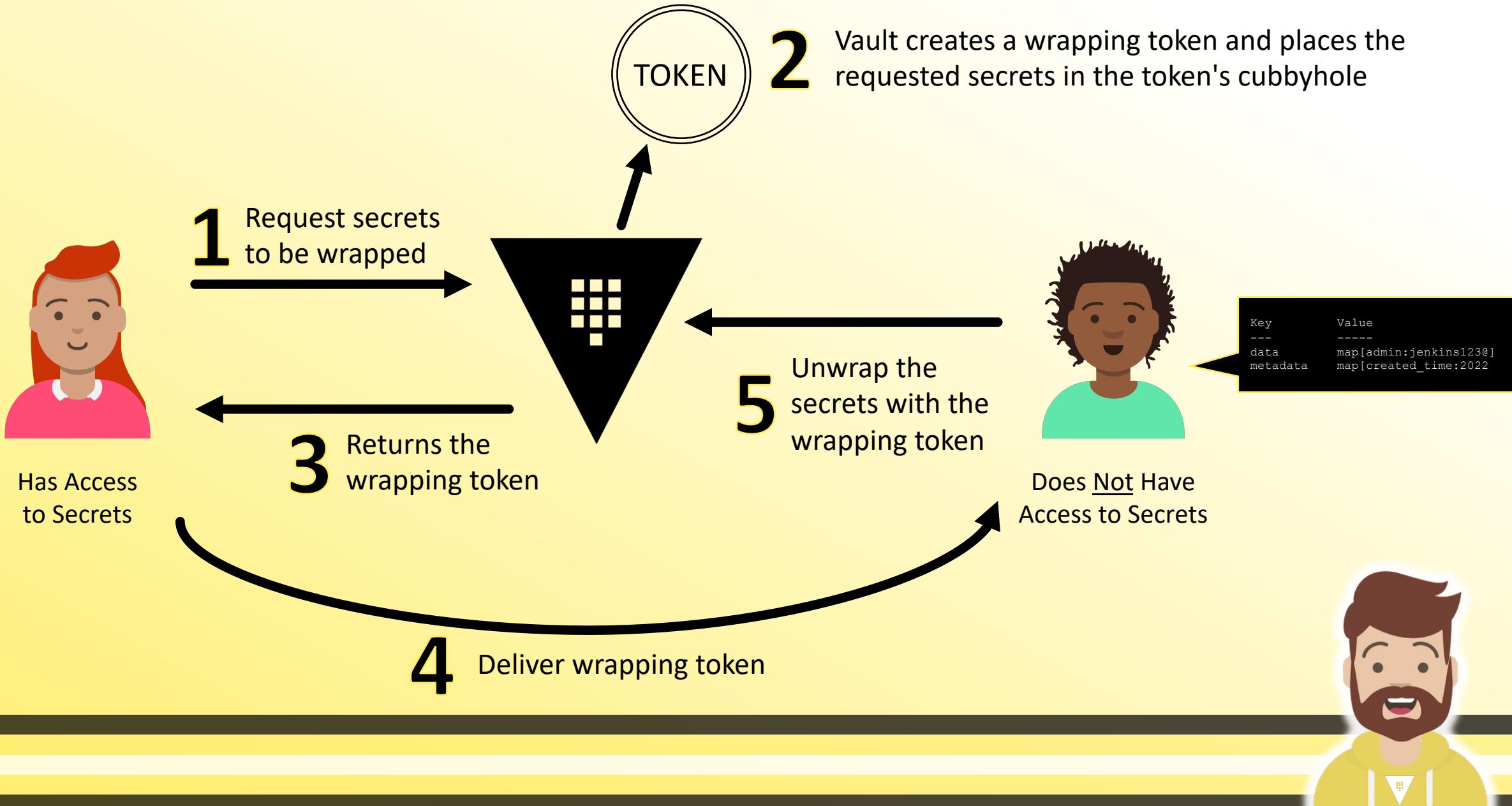
Response Wrapping

Instead of sending secrets over the network, you can use cubbyhole's response wrapping feature

- Enables the retrieval of secrets from a path and Vault will store them inside of another token's cubbyhole
 - This token is called the wrapping token
 - The wrapping token is temporary and limited by a TTL
 - It is also a single-use token
- The wrapping token can be sent across the network and the user can unwrap the token to retrieve the real data (secrets)



Response Wrapping





THE SECRET NEVER LEAVES VAULT

You are sending the reference
(wrapping token) across the network



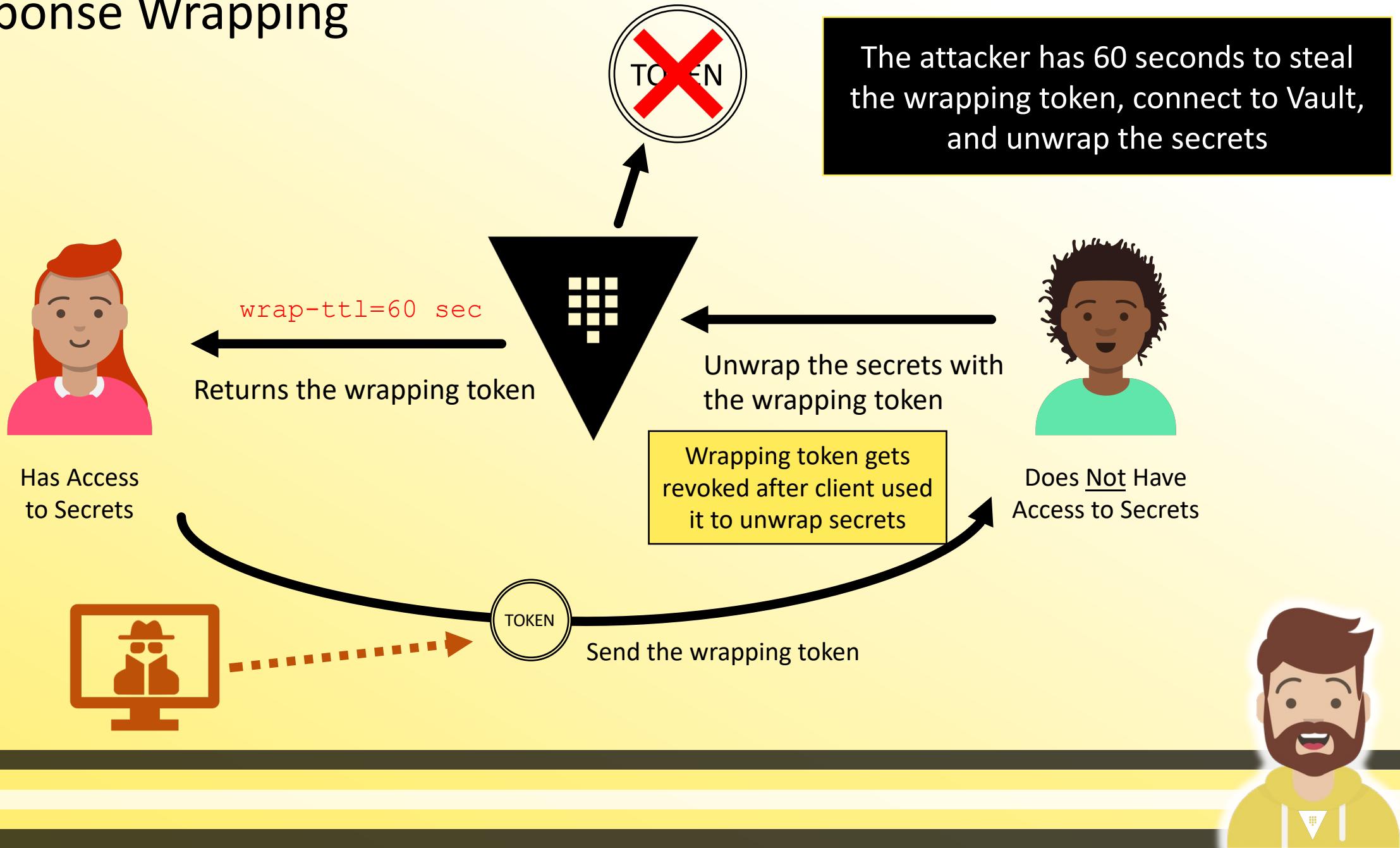
Benefits of Response Wrapping

Response wrapping provides:

- ***privacy*** by ensuring that any secret transmitted across the network is not the actual secret
- ***malfeasance detection*** by ensuring that only a single party can unwrap the token and gain access to the secrets (because it's a single-use token)
- ***limitation of the lifetime*** of the secret exposure because the wrapping token has a defined TTL



Response Wrapping



Wrapping Secrets using the CLI

```
$ vault kv get -wrap-ttl=5m secrets/certification/hcvop
```

Key	Value
---	-----
wrapping_token:	hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW
wrapping_accessor:	O5XSKsRf0c7CwXo996BjKyNi
wrapping_token_ttl:	5m
wrapping_token_creation_time:	2022-12-25 10:36:36.588947 -0400 EDT
wrapping_token_creation_path:	secrets/certification/hcvop

We got back a wrapping token instead of the actual secrets

You can get static or dynamic secrets using response wrapping



Lookup a Wrapping Token

```
$ vault token lookup hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW
```

Key	Value
---	---
accessor	05XSKsRf0c7CwXo996BjKyNi
creation_time	1649172724
creation_ttl	5m
display_name	n/a
entity_id	n/a
expire_time	2022-12-25T10:41:36.588968-04:00
explicit_max_ttl	5m
id	hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW
issue_time	2022-12-25T10:36:36.588947-04:00
meta	<nil>
num_uses	1
orphan	true
path	secrets/certification/hcvop
policies	[response-wrapping]
renewable	false
ttl	4m48s
type	service

Expires in 5 minutes

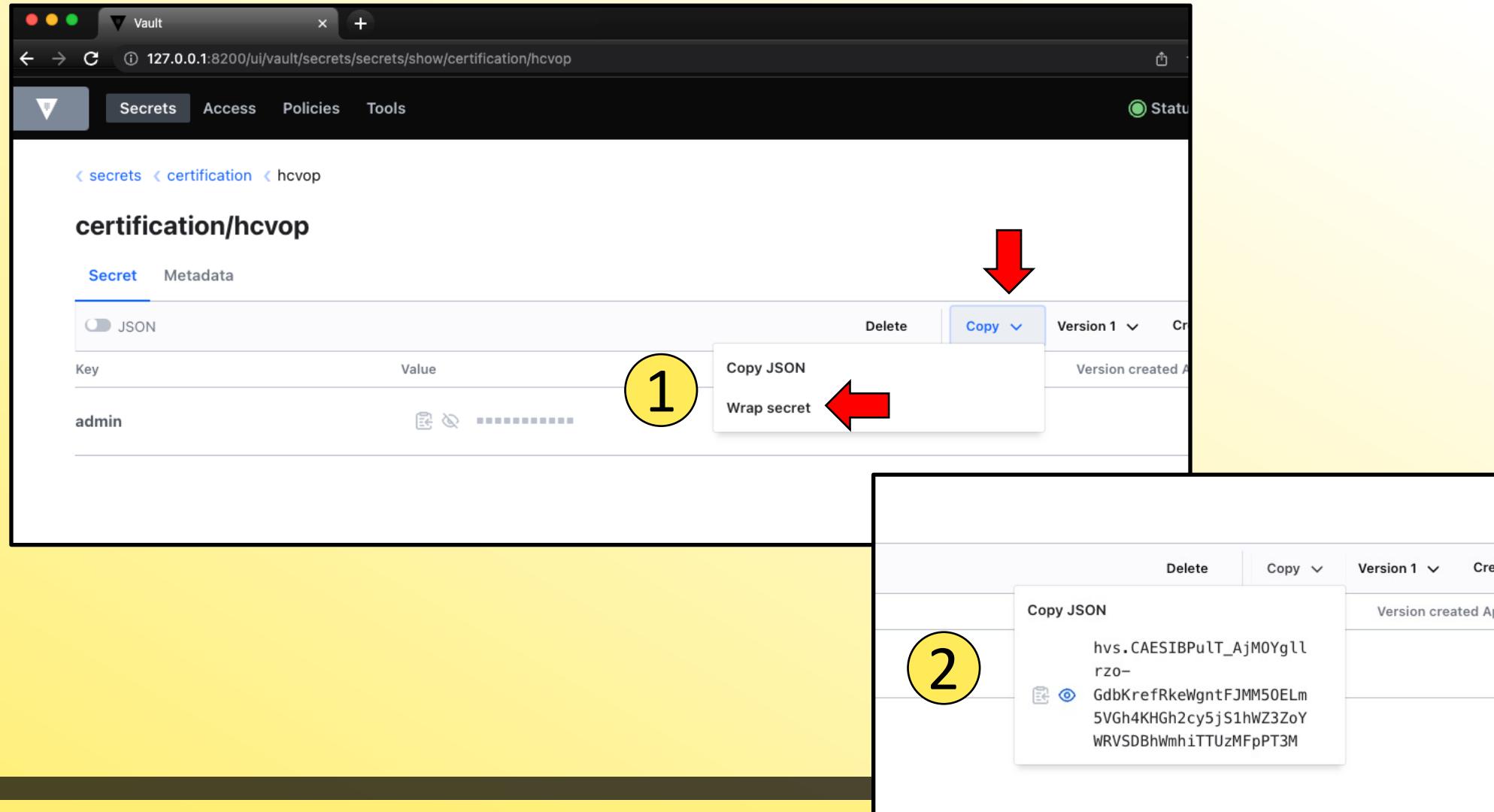
Single-Use Token

Path of secret we requested

Not Renewable



Wrap a Secret using the UI



Unwrap the Secret

Use `vault unwrap` command to retrieve the secrets

```
$ vault unwrap <wrapping-token>  
or  
$ export VAULT_TOKEN=<wrapping-token> vault unwrap  
or  
$ vault login <wrapping-token>  
$ vault unwrap
```



Unwrap the Secret using the CLI

Use `vault unwrap` command to retrieve the secrets

```
$ vault unwrap hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW

Key          Value
---
data         map[admin:jenkins123@]
metadata     map[created_time:2022-12-25T14:33:10.525712Z custom_metadata:<nil> ...]
```



Unwrap the Secret using the UI

The screenshot shows two screenshots of the HashiCorp Vault UI, illustrating the process of unwrapping a secret.

Screenshot 1: The "Unwrap data" page. A red arrow points to the "Tools" menu at the top, and another red arrow points to the "Unwrap" option in the "TOOLS" sidebar. A yellow circle with the number "1" is placed over the "Unwrap data" button.

Screenshot 2: The "Unwrap data" page after the process has completed. The "Data" tab is selected, showing the unwrapped secret. A yellow circle with the number "2" is placed over the JSON output area. The JSON output is as follows:

```
{  
  "data": {  
    "admin": "jenkins123@"  
  },  
  "metadata": {  
    "created_time": "2022-04-05T15:54:09.779Z",  
    "custom_metadata": null,  
    "deletion_time": "",  
    "destroyed": false,  
    "version": 1  
}
```



Practice Production Hardening



Intro to Production Hardening



There are many best practices for a production hardened deployment of Vault.

Practice defense in depth and follow the Vault security model

Product Hardening is broken down into multiple categories:

- General Recommendations
- Operating System Recommendations
- Vault-Specific Recommendations



Note for this Section



This will be a conceptual discussion with no demos

In the exam, you will NOT be expected to know how to perform Operating System-level configurations, like disabling swap or make file permission changes.

However, you may be asked questions (multiple choice) on HOW to make a configuration or deployment MORE secure based on provided examples.





General Topics



Deployment Model



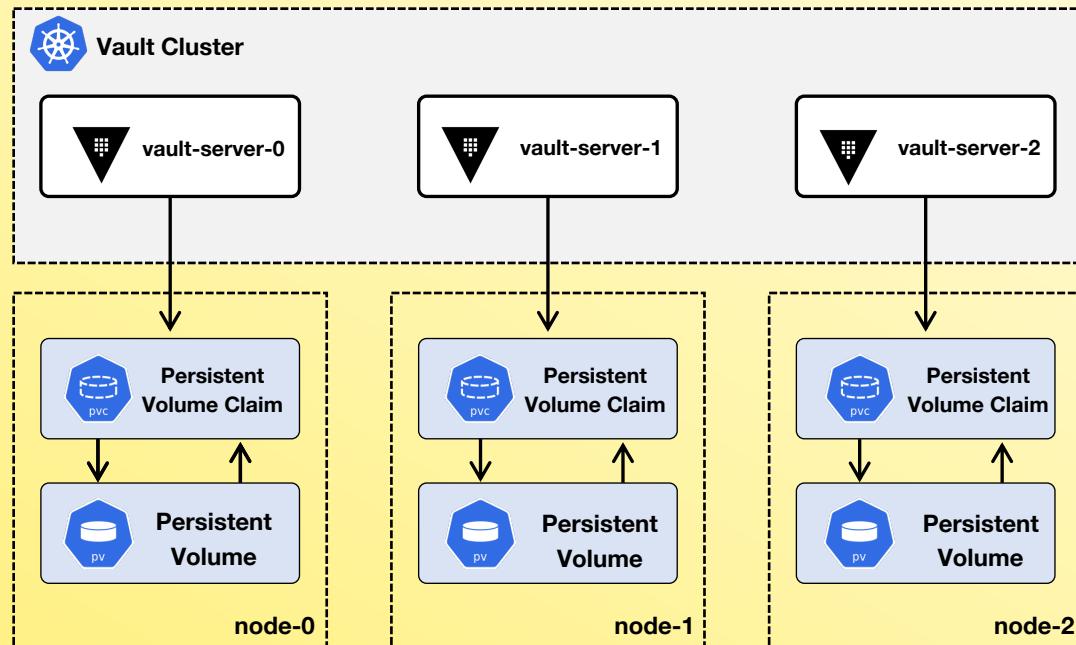
- The fewer shared resources, the better
- Think "single tenancy" where possible
- Secure deployments: Hardware > VMs > Containers
- Ultimately comes down to protecting memory contents
- Many customers will still use virtualization (VMware/Cloud) or containerization (Docker/K8s) but will deploy to dedicated clusters



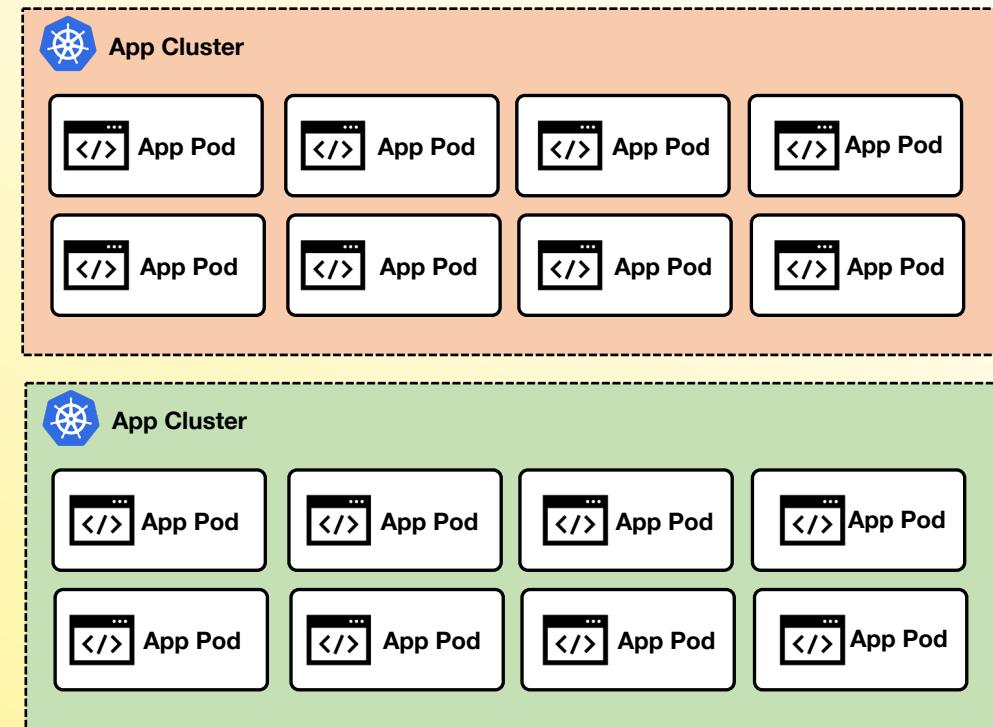
Deployment Model



Dedicated Vault Cluster



Application Clusters



Limit Access to Vault Nodes



- Reduce or eliminate access to Vault nodes
- Includes SSH/RDP and through platform-based access (i.e., AWS SSM, `kubectl exec -it <pod>`, etc)
- Instead, access Vault via API or CLI from your workstation or jump box
- If you REALLY need access, use HashiCorp Boundary to limit/control access



Limit Services Running on Vault Nodes



- Vault nodes should be dedicated to Vault services
- You should not have other services contending for resources
- More services = more firewall requirements
- Don't forget: Encryption keys are stored in memory
- Exception to this rule may include:
 - Telemetry agent
 - Log file agent (Splunk, SumoLogic, DataDog)



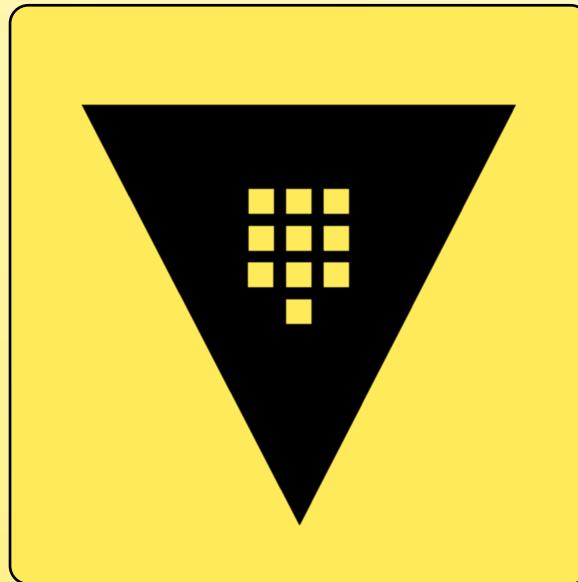
Permit Only Required Ports on Firewall



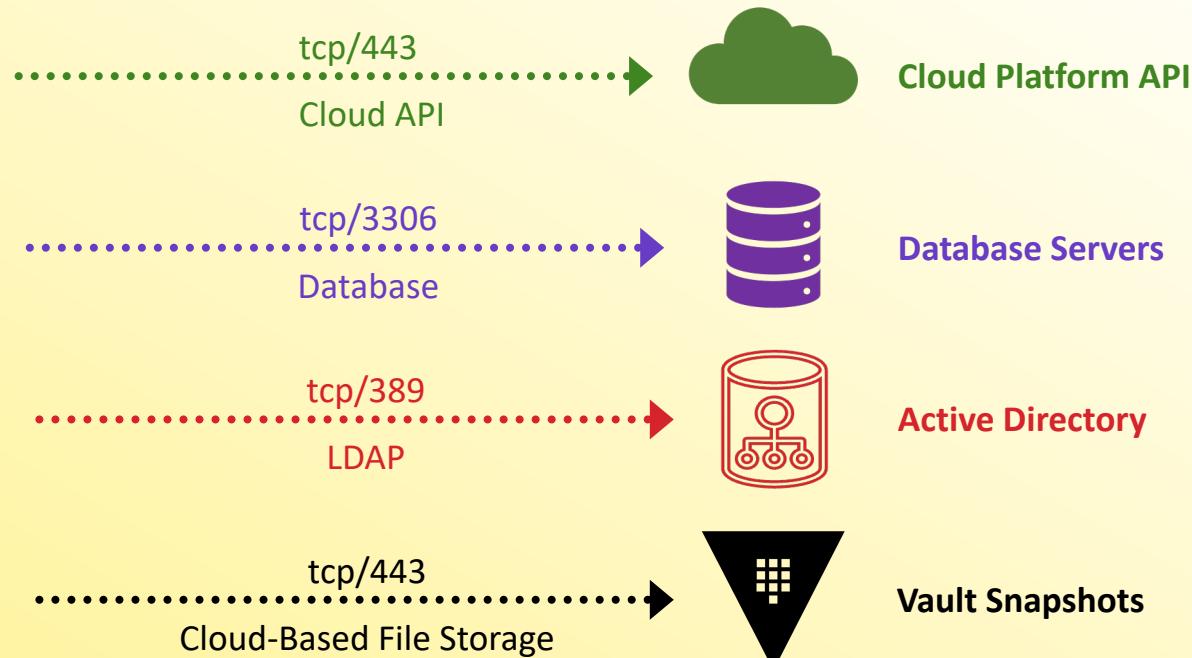
- Vault and Consul use dedicated ports for communication
- Permit only the required ports to reduce attack surface
- Many enterprise Vault deployments don't even allow SSH or UI ports
- Default ports include:
 - Vault: 8200, 8201
 - Consul: 8500, 8201, 8301



More Ports May Be Needed for Your Implementation

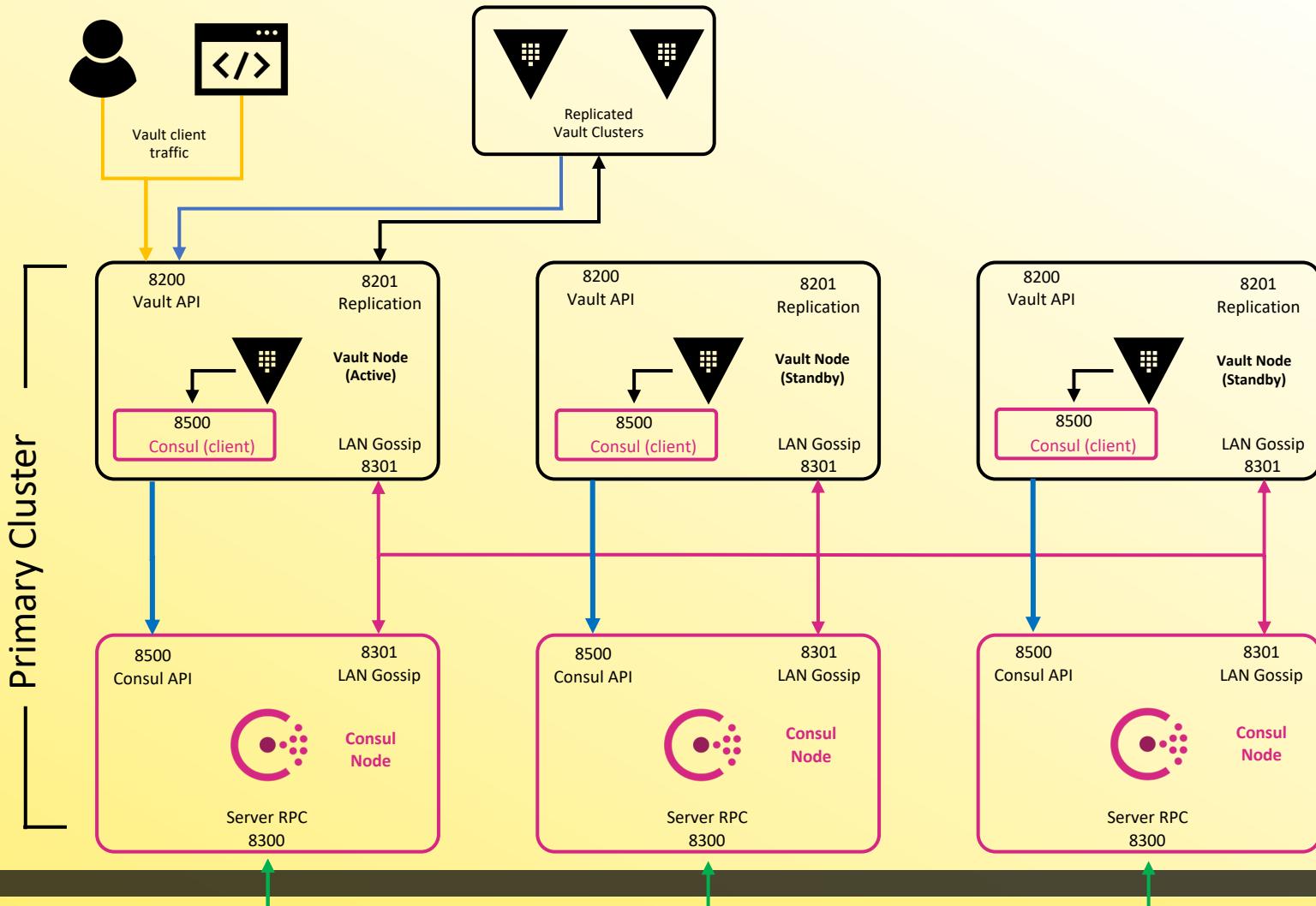


Vault Cluster



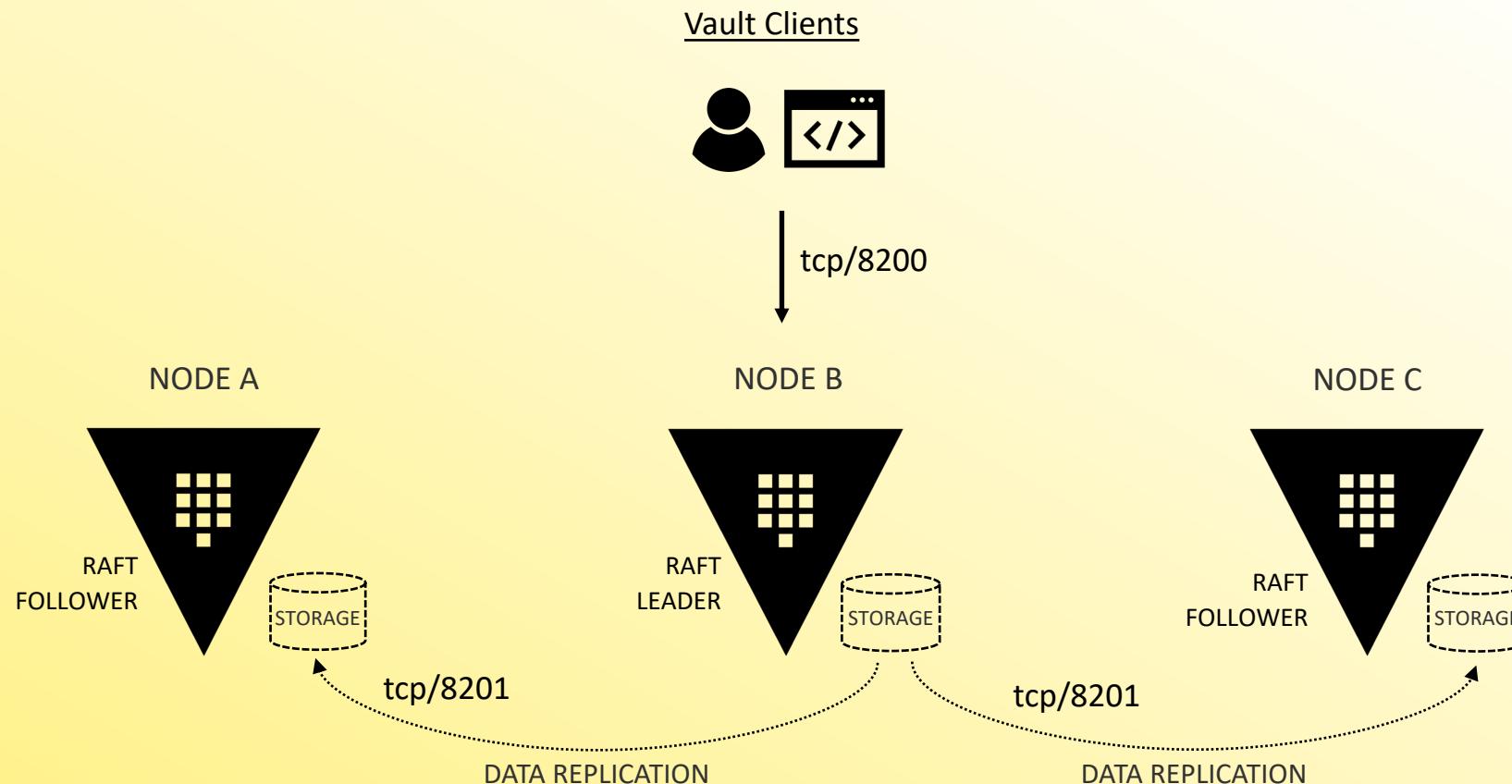
Permit Only Required Ports on Firewall

Consul Example



Permit Only Required Ports on Firewall

Integrated Storage Example



Immutable Upgrades



- Immutable upgrades guarantee a known state because you know the result of your automation configurations
- Easy to bring new nodes online, destroy the old nodes
- Consul and Raft can use AutoPilot to assist with upgrades
- Care must be taken when using Raft because you need to ensure replication has been completed to newly added nodes





Operating System



Run Vault as an Unprivileged User



- Never run Vault as Root
- Running as root can expose Vault's sensitive data
- Limit access to configuration files and folders to the Vault user
- Normally, I create a user named “vault”
- Don't forget that the Vault user will need access to write local files, such as the database, audit logs, and snapshots
 - `chmod -R vault:vault /opt/vault/data`



Secure Files and Directories



- Protect and audit critical Vault directories and files, including directory for snapshots
- Ensure unauthorized changes can't be made
- Includes binaries, config files, plugins files and directory, service configurations, audit device files and directory, etc.

```
# Set permissions on Vault folder  
$ chmod 740 -R /etc/vault.d
```



Protect the Storage Backend



- Vault writes all configuration and data to the storage backend
- No storage backend = No Vault!
- Consul Storage Backend:
 - Use Consul ACLs when running on Consul
 - Limit access to any Consul node
 - Enable `verify_server_hostname` in config file



Disable Shell History



- Disabling history prevents retrieval of commands
- Possible to discover credentials/tokens in history
- Can also disable just 'vault' command in history

```
$ history
1365 vault login hvs.RTMd9YZ5Np9WGjvTfARaqffQ
1366 vault policy list
1367 vault list sys/policies/acl
1368 vault secrets list
1369 vault list pki/roles
```

to prevent this...

```
# Disable a command history system wide
$ echo 'set +o history' >> /etc/profile
```

do this....



Configure SELinux/AppArmor



- Don't disable to make install/management easier
- Provides additional layers of protection for the OS
- Adhere to CIS or DISA to improve posture of the host OS

▼ PRODUCTS & TECHNOLOGY

Hardening HashiCorp Vault with SELinux

We have developed a baseline SELinux policy for securing Vault on Red Hat-based Linux Distributions

hashicorp.com/blog/hardening-hashicorp-vault-with-selinux



Turn Off Core Dumps



- Core dumps could reveal encryption keys
- Different process to disable depending on the OS
- You won't have to do this in the exam, just know it's something that you **SHOULD** do in a production environment



Protect and Audit the vault.service File



- Make sure you know if this file is modified or replaced
- An attacker could point to compromised binaries to leak data
- This assumes you are running systemd, but any "service" file should be monitored and secured



Patch the Operating System Frequently



- Make sure to patch the OS frequently
- Follow your standards or be more stringent for Vault
- Options include Satellite, SpaceWalk, or other solution
- If you use an immutable architecture, then replace the nodes often with known good "patched" images. Use Packer to help simplify this workflow.



Disable Swap



- Vault stores sensitive data in-memory, unencrypted
 - That data should never be written to disk
 - Disabling swap provides an extra layer of protection
-
- Different process for different operating systems but again, you don't have to do this on the exam
 - Example is enabling `mlock` to prevent memory swap







Vault-Specific Configurations



Secure Vault with TLS



- Vault contains sensitive data
- Communications should never occur without TLS in place
- Load Balancers used with Vault can terminate TLS or instead use pass through to the Vault nodes.
- Verify `tls_disable` configuration does not equal `true` or `1`
 - Default is `false` (not disabled)



Secure Consul



- Consul contains your sensitive data
- Communications should never occur without TLS in place
- Issue a certificate from a trusted CA
- Enable Consul ACLs
- Configure gossip encryption (use `consul keygen`)
- In short, follow the Consul Security Model



Enable Auditing



- Use multiple Audit Devices to log all interactions
- Send that data to a collection server
- Archive log data based upon security policies
- Create alerts based on certain actions



Say No to Cleartext Credentials



- Don't put credentials in configuration files
- Use Environment Variables, where supported
- Use cloud-integrated services, such as AWS IAM Roles or Azure Managed Service Identities

```
● ● ●  
1 seal "awskms" {  
2   region = "us-east-1"  
3   kms_key_id = "12345678-abcd-1234-abcd-123456789101",  
4   endpoint = "example.kms.us-east-1.vpce.amazonaws.com"  
5   access_key="AKIAIOSFODNN7EXAMPLE"  
6   secret_key="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"  
7 }
```

← Don't do this....



Upgrade Vault Frequently



- New updates frequently include security fixes
- New cipher suites can be enabled or supported
- New functionality enabled



Stop Using Root Tokens, Seriously



- Root tokens have unrestricted access to Vault **with no TTL**
- Not bound by ACL policies, ERPs, or RGPs
- Get rid of the initial root token after initial setup
 - `vault token revoke hvs.xxxxxxxxxx`



Verify the Integrity of the Vault Binary



- Always get Vault binaries directly from HashiCorp
- Use the HashiCorp checksum to validate
- Modified version of Vault binary could leak data



Disable the UI – if Not in Use



- Vault UI is disabled by default
- Configured in the Vault configuration file
- Do the same for Consul UI, as well

```
● ● ●  
1 ui = false  
2 log_level = "INFO"  
3 license_path = "/opt/vault/vault.hcl"
```



Encrypt the Gossip Protocol (Consul)



- TLS only secures the interfaces, not Consul gossip traffic
- Use the `--encrypt` flag in the Consul configuration file
- Uses a 32-byte key – can use `consul keygen` to generate



Secure the Unseal/Recovery Keys



- Initialize Vault using PGP keys, such as keybase.io
- Distribute to multiple team members, no single person should have all the keys
- Don't store the keys in Vault itself



Minimize the TTLs for Leases and Tokens



- Use the smallest TTL possible for tokens
- Define Max TTLs to prevent renewals beyond reasonable timeframe
- Minimizing TTL also helps reduce burden on the storage backend



Follow the Principle of Least Privilege



- Only give tokens access to paths required for business function
- Separate policies for applications and users
- Limit use of * and + in policies, where possible
- Templatized policies can help with policy creation and maintenance



Perform Regular Backups



- Backup configuration files and directories
 - Automate Vault backup using snapshots or equivalent depending on the storage backend
 - Regularly test backups to ensure functionality
-
- `vault operator raft snapshot save monday.snap`
 - `vault write sys/storage/raft/snapshot-auto/config/daily` (Enterprise)



Integrate with Existing Identity Providers



- Use your existing IdP to provide access to users
- If/when users leave, they immediately lose access to Vault
- The fewer places a user has credentials, the better
- Using locally defined credentials is an administrative burden





Monitoring and Alerting



Vault Security Monitoring



- Use of a root token
- Creation of a new root token
- Vault policy modification
- Enabling a new auth method
- Modification of an auth method role
- Creation of a new auth method role
- Permission denied (403) responses
- Use of Vault by human-related accounts outside of regular business hours
- Vault requests originating from unrecognized subnets
- Transit Minimum Decryption Version Config
- Seal Status of Vault
- Audit Log Failures
- Resource Quota Violations
- Updates to Vault Policies
- Transit Key Deletion
- Cloud-based resource changes

Done via Audit Log and
Log Collection Tool





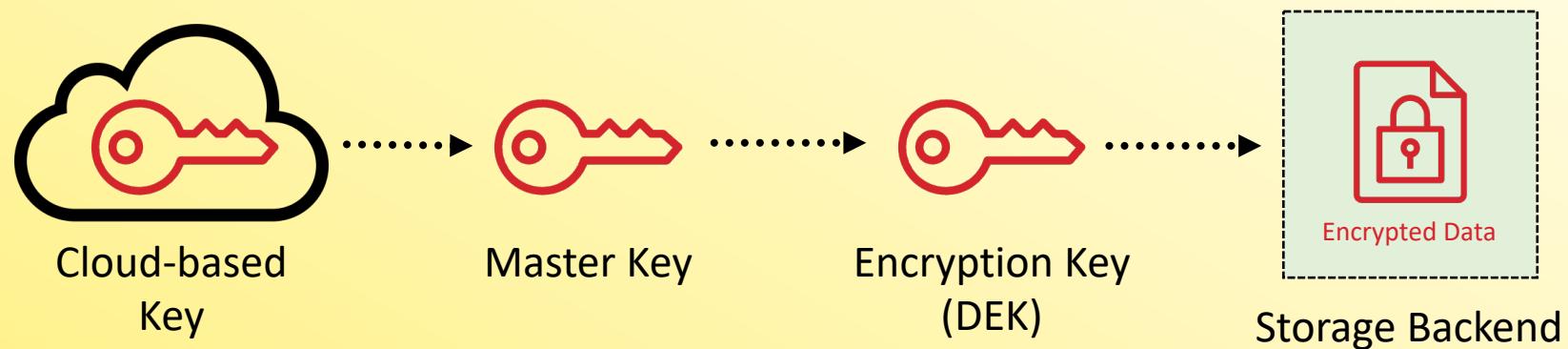
Auto Unseal Vault



What is Auto Unseal?



- Rather than use shared keys (unseal keys), auto unseal can use a trusted cloud-based key to protect the master key



Supported Auto Unseal Mechanisms



- Supported Services include:
 - AWS KMS
 - Azure Key Vault
 - GCP Cloud KMS
 - AliCloud KMS
 - OCI KMS
 - HSM (*Enterprise Only*)
- Transit Secrets Engine from *another* Vault cluster can also be used to unseal



How Auto Unseal Works



- Vault does not write anything to the cloud-based service
- The master key is encrypted with the cloud-based key and stored on the storage backend – [stored at path/core/master](#)
- It uses the cloud-based key to decrypt the master key during the unseal process
- This is different from a Vault cluster not using auto unseal and is using Shamir (unseal keys) - where the master key is never written to persistent storage



Design Considerations



- Vault does support rotation for cloud-based keys
- Remember that some cloud-based KMS services are regional, so if an entire region goes down, the KMS key will not be available to unseal
- During normal operations, the Vault cluster does not communicate with the auto unseal key service for unseal operations
 - However, recent versions of Vault have introduced a "health check" where Vault validates access to the key periodically
- Health Check will test the health of the auto-unseal backend once every 10 minutes.
 - If unhealthy, logs a warning on the condition and begin testing every one minute until healthy again.



Design Considerations



Service downtime is only a problem if your Vault cluster is restarted or sealed and needs to be unsealed during that time

- To avoid this problem, many folks will create a key outside of the cloud-based service and import it to multiple regions for a fail-safe
- Clusters used for Transit auto unseal need to be highly available to prevent this from happening to the cluster it supports
- Remember that *some* cloud-based services now offer multi-region keys to provide built-in regional high-availability



How to Enable Auto Unseal



Configured in the Vault configuration file

```
...
listener "tcp" {
    address = "0.0.0.0:8200"
    cluster_address = "0.0.0.0:8201"
    tls_disable = 0
}
...
}
seal "awskms" {
    region = "us-east-1"
    kms_key_id = "12345678-abcd-1234-abcd-123456789101",
    endpoint = "example.kms.us-east-1.vpce.amazonaws.com"
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = " https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```

*Configuration truncated



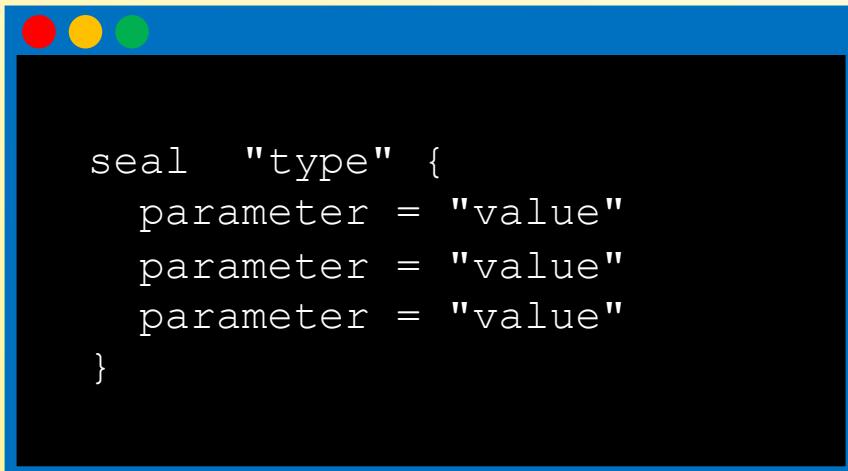
How to Enable Auto Unseal



1. Add a seal stanza to your configuration.
2. Choose what service to use
3. Configure the parameters based on the service

Options include:

- alicloudkms
- awskms
- azurekeyvault
- gcpckms
- ocikms
- pkcs11
- transit

A screenshot of a terminal window with a blue title bar and red, yellow, and green window controls. The terminal itself is black with white text. It displays a single line of configuration code:

```
seal "type" {  
    parameter = "value"  
    parameter = "value"  
    parameter = "value"  
}
```



Auto Unseal with AWS KMS



- Set the seal type to aws kms
- Identify the KMS Key ID (ARN) for the key Vault will use
- Declare the region for the key
- Set the VPC Endpoint (if being used)
- Set the AWS credentials

Be careful with adding credentials to a clear text file:

- If hosting in AWS, use an IAM role
- If hosting on-prem, use environment variables
- AWS role needs kms:Encrypt, kms:Decrypt, and kms:Describe for the KMS key



```
seal "awskms" {
    region = "us-east-1"
    kms_key_id = "arn:aws:kms:us-east-1:12345678:key/abcd"
    endpoint = "example.kms.us-east-1.vpce.amazonaws.com"
    access_key = "AKIAIOSFODNN7EXAMPLE"
    secret_key = "wJalrXUtFEMI/K7Mexample5"
}
```

A screenshot of a terminal window showing a configuration block for a Vault seal. The access and secret keys are highlighted with a red rectangle.

Auto Unseal with Azure Key Vault



- Set the seal type to azurekeyvault
- Identify the Key Vault Name
- Set the Key Name for HashiCorp Vault to use
- Set the Azure credentials

Be careful with adding credentials to a clear text file:

- If hosting in Azure, use Managed Service Identities
- If hosting on-prem, use environment variables



```
seal "azurekeyvault" {
    vault_name = "vault-hashicorp"
    key_name = "hashicorp-vault-key"
    tenant_id = "8427464-8963-6422-example"
    client_id = "03dc33fc-16d9-example"
    client_secret = "DKEMCI8...."
```

Auto Unseal with GCP KMS



- Set the seal type to “gcpckms”
- Identify the GCP Project ID to use
- Set the region where the key ring lives
- Set the key ring to use
- Identify the crypto key that Vault will use

- Set the GCP credentials

Be careful with adding credentials to a clear text file:

- If hosting in GCP, set the instance's service account with Cloud KMS role
- If hosting on-prem, use environment variables



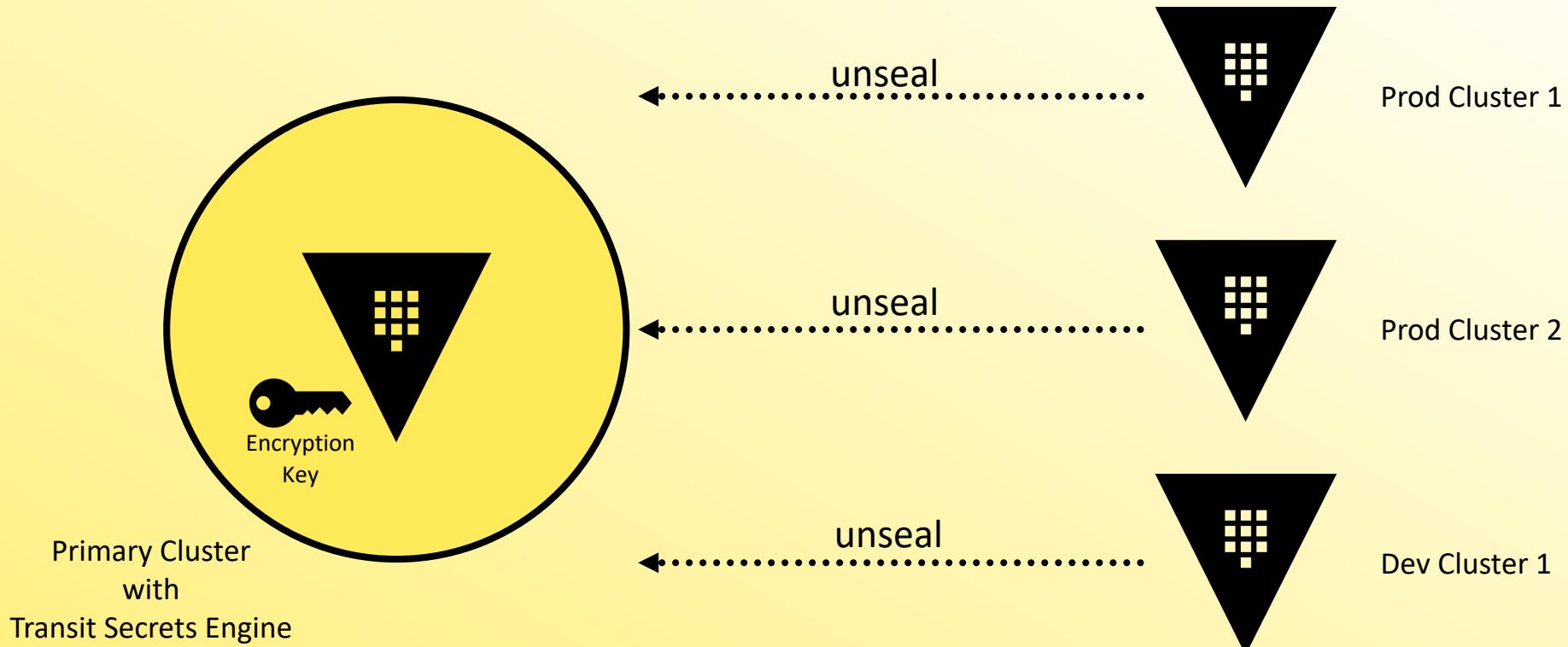
```
seal "gcpckms" {
    project = "vault-project"
    region = "global"
    key_ring = "hashicorp-vault-keyring"
    crypto_key = "hashicorp-vault-key"
    credentials = "/usr/gcp.json"
}
```

A screenshot of a terminal window showing a configuration block for sealing with GCP KMS. The 'credentials' line is highlighted with a red box.

Auto Unseal with Transit



- Using *another* Vault cluster, configure the transit secrets engine and create a key to be used by Vault



Auto Unseal with Transit



- Set the seal type to transit
- Identify the address of the primary cluster
- Set the key name
- Set the mount path and namespace (if applicable)
- Set the credentials (token)

Be careful with adding credentials to a clear text file:

- Use environment variable VAULT_TOKEN
- The token needs "update" capability for *transit/encrypt/<key>* and *transit/decrypt/<key>*



```
seal "transit" {
  address = "https://vault.hcvop.com:8200"
  token = "s.Qf1s5zigZ4OX6akYexample"
  key_name = "auto-unseal-key"
  mount_path = "/transit"
  tls_ca_cert = "/etc/vault/ca.pem"
  tls_client_cert = "/etc/vault/client.pem"
  tls_client_key = "/etc/vault/key.pem"
}
```

Other Options Not Covered Here...



- **HSM**
 - Follows a similar process of using a key stored in a trusted HSM.
 - Vault uses the key to decrypt the master key
 - HSM integration also enables Seal Wrapping as well to provide FIPS 140-2 compliance
- **AliCloud**
- **Oracle Cloud**

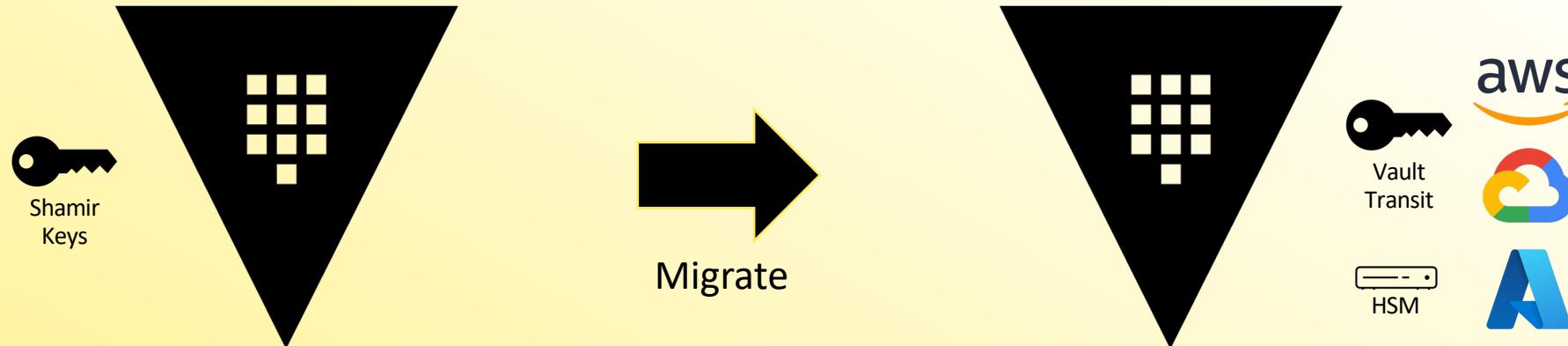




Seal Migration



Seal Migration



Vault Cluster

Shamir Keys

- Manual Unseal
- Risk of Losing Keys
- Not Automated

Vault Cluster

Auto Unseal

- Automatic Unseal
- Protected by trusted KMS
- No Reliance on Humans



Seal Migration



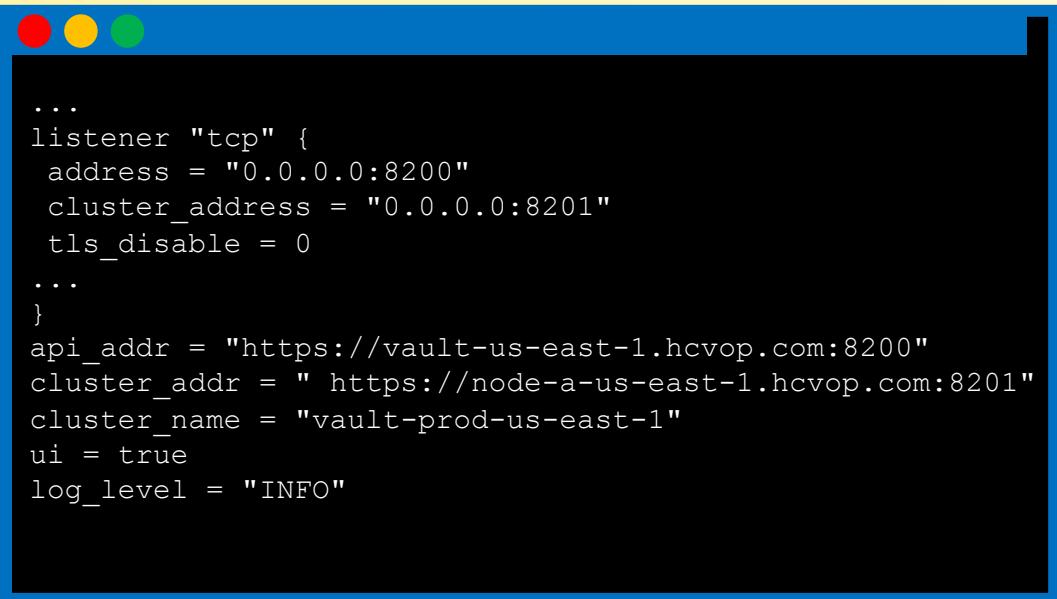
- You can migrate from one seal type to another if needed
- Unfortunately, it requires downtime for Vault since you must restart the service
- This is not a process that you will perform very often, if at all...
- Examples of migrations:
 - Shamir → [AWS KMS](#)
 - [GCP Cloud KMS](#) → [Azure Key Vault](#)
 - [AWS KMS](#) → [AWS KMS](#)
 - [Azure Key Vault](#) → [HSM](#)



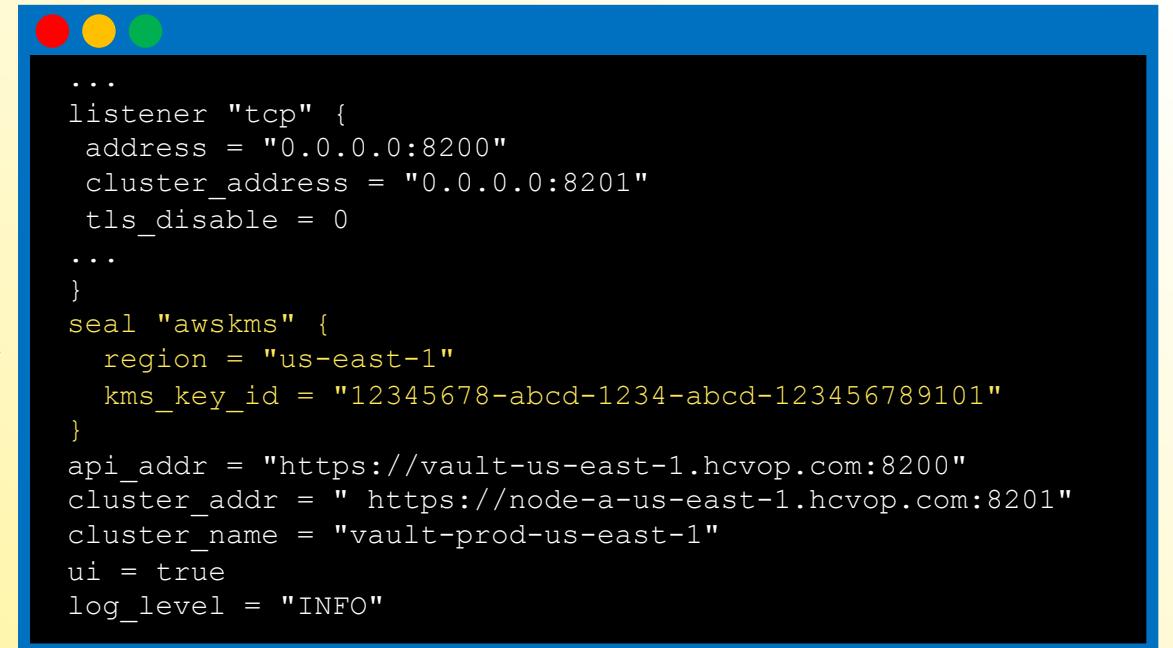
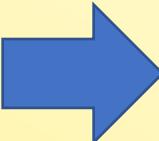
Shamir to Auto Unseal Migration

Shamir clusters don't have a seal configuration because it's the default configuration

...but auto unseal does, so the first step is to add the new configuration to a standby node (or the only node)



```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
  tls_disable = 0
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = " https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```



```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
  tls_disable = 0
}
seal "awskms" {
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101"
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = " https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```



Shamir to Auto Unseal Migration



Restart the Vault service

```
# Restart the Vault service
$ systemctl restart vault.service

# View the log file to see a new log entry

2022-12-25T10:47:09.295-0400 [WARN]  core: entering seal migration mode; Vault will not
automatically unseal even if using an autoseal: from_barrier_type=shamir to_barrier_type=awskms
```



Shamir to Auto Unseal Migration



Unseal Vault using the `-migrate` flag

```
# Unseal Vault using the migrate flag
$ vault operator unseal -migrate
Unseal Key (will be hidden): <enter key here>

Key          Value
---          ---
Recovery Seal Type      shamir
Initialized           true
Sealed                false
Total Recovery Shares   5
Threshold              3
Seal Migration in Progress  true
Version               1.10.0+ent
Storage Type          raft
Cluster Name          vault-prod-us-east-1
Cluster ID             fa1b-4ccb17a1ecac
HA Enabled             true
HA Cluster             n/a
HA Mode                standby
Active Node Address    <none>
Raft Committed Index   65
Raft Applied Index     65
```

Do this multiple times to meet key threshold

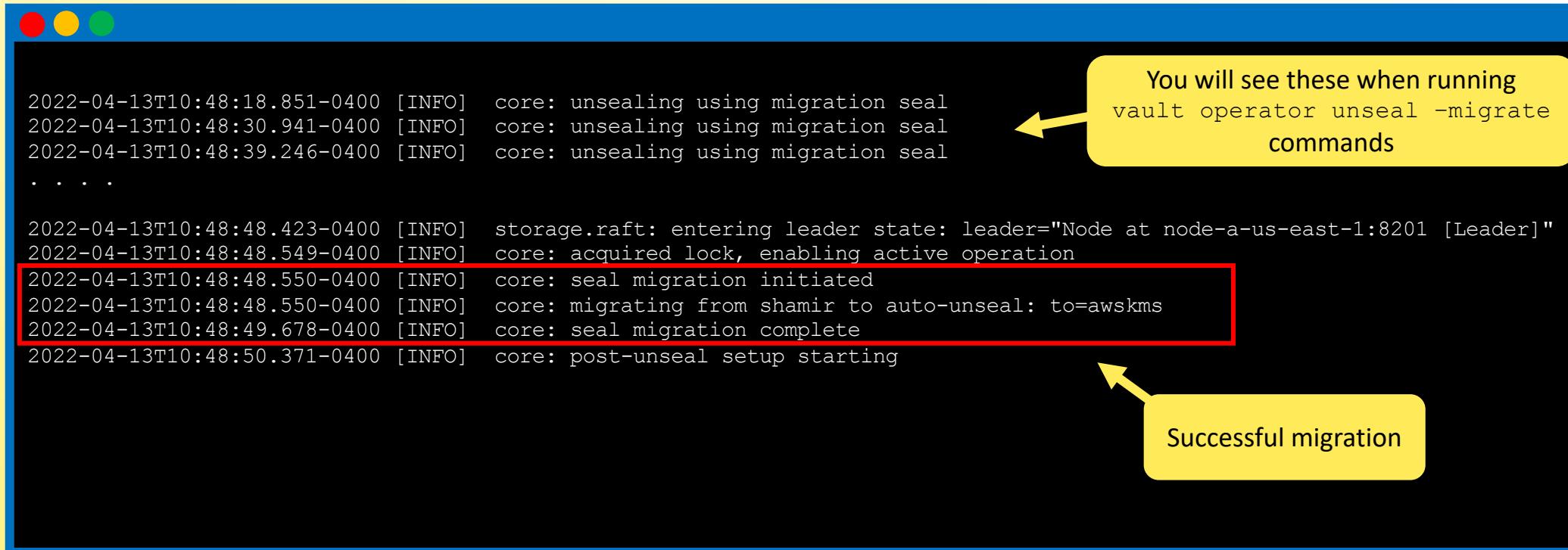
Note: If the key threshold is 3, then you will need to run this command 3 times.

If it is set to 1, *like you'll find in the exam*, then you'll just need to run it 1 time



Shamir to Auto Unseal Migration

Viewing the Logs During Migration Steps



A screenshot of a terminal window showing Vault log output during a migration from Shamir to Auto-Unseal. The logs are timestamped from April 13, 2022, at 10:48:18 to 10:48:50. A red box highlights the migration steps, and yellow callout boxes explain what they mean.

```
2022-04-13T10:48:18.851-0400 [INFO] core: unsealing using migration seal
2022-04-13T10:48:30.941-0400 [INFO] core: unsealing using migration seal
2022-04-13T10:48:39.246-0400 [INFO] core: unsealing using migration seal
...
2022-04-13T10:48:48.423-0400 [INFO] storage.raft: entering leader state: leader="Node at node-a-us-east-1:8201 [Leader]"
2022-04-13T10:48:48.549-0400 [INFO] core: acquired lock, enabling active operation
2022-04-13T10:48:48.550-0400 [INFO] core: seal migration initiated
2022-04-13T10:48:48.550-0400 [INFO] core: migrating from shamir to auto-unseal: to=awskms
2022-04-13T10:48:49.678-0400 [INFO] core: seal migration complete
2022-04-13T10:48:50.371-0400 [INFO] core: post-unseal setup starting
```

You will see these when running vault operator unseal -migrate commands

Successful migration

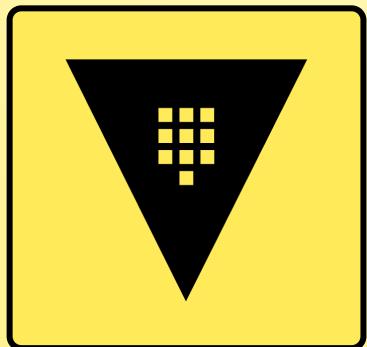


Shamir to Auto Unseal Migration

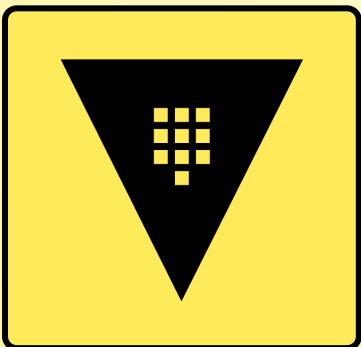
Order of Operations



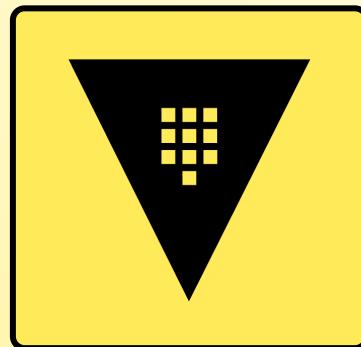
1. Perform unseal migration on standby node 1
2. Perform unseal migration on standby node 2
3. Run `vault operator step-down` on the Leader node
4. Perform unseal migration on the last node (previous leader)



Standby Node



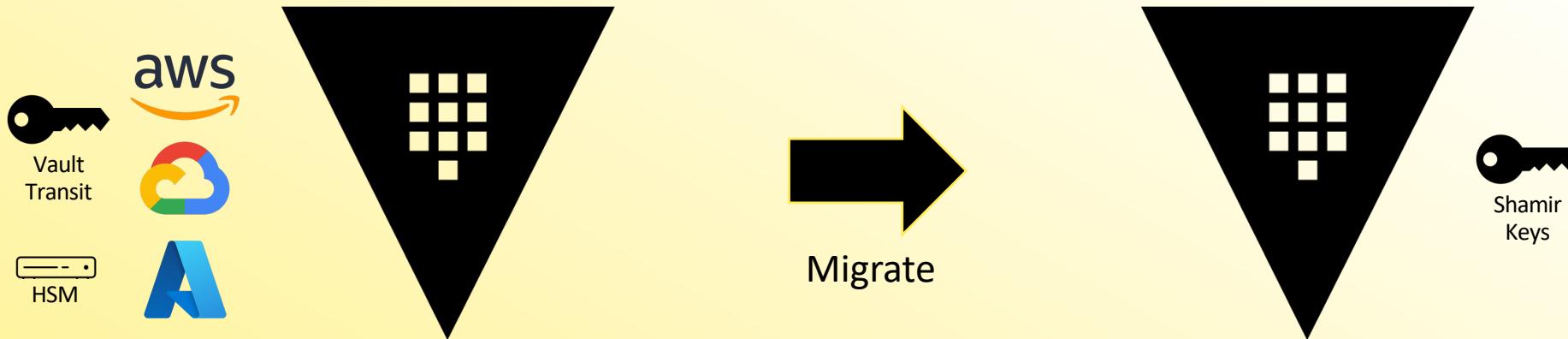
Leader Node



Standby Node



Auto Unseal to Shamir Migration



Vault Cluster

Auto Unseal

- Automatic Unseal
- Protected by trusted KMS
- No Reliance on Humans

Vault Cluster

Shamir Keys

- Manual Unseal
- Risk of Losing Keys
- Not Automated



Auto Unseal to Shamir Migration



- Update the seal stanza to include `disabled = true`
 - This allows Vault to decrypt the key, but it will NOT use it for unseal operations
- Follow the same process as previously discussed:
 - Restart the Vault service
 - Run `vault operator unseal -migrate`
 - Provide RECOVERY keys to perform the migration
 - Recovery keys will be migrated to be used as unseal keys moving forward



Auto Unseal to Shamir Migration

Vault

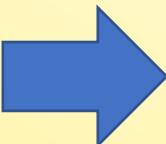
CERTIFIED
OPERATIONS
PROFESSIONAL



Auto Unseal configuration on an auto unsealed cluster needs to be modified...

...add the `disabled = true` configuration in the seal stanza as the first step for migration

```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
  tls_disable = 0
}
...
seal "awskms" {
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101"
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = " https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```



```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
  tls_disable = 0
}
...
seal "awskms" {
  disabled = true
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101"
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = " https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```



Auto Unseal to Auto Unseal Migration

Including Changing Keys when using the Same Auto Unseal



Vault Cluster

Auto Unseal

- Automatic Unseal
- Protected by trusted KMS
- No Reliance on Humans

Vault Cluster

Auto Unseal

- Automatic Unseal
- Protected by trusted KMS
- No Reliance on Humans



Auto Unseal to Auto Unseal



1. Update the **original** seal stanza to include disabled = true
 - This allows Vault to decrypt the key, but it will NOT use it for unseal operations
2. Add the NEW stanza to the configuration
3. Follow the same process as previously discussed:
 - Restart the Vault service
 - Run `vault operator unseal -migrate`
 - Provide RECOVERY keys to perform the migration
 - Recovery keys will be migrated to be used as unseal keys moving forward

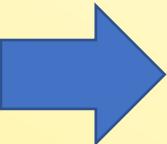


Auto Unseal to Shamir Migration

Auto Unseal configuration on an auto unsealed cluster needs to be modified.....add the disabled =true configuration in the seal stanza as the first step for migration

...add the new seal configuration as a second stanza to the configuration file

```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
  tls_disable = 0
}
...
seal "awskms" {
  disabled = true
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101"
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = "https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```



```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
  tls_disable = 0
}
...
seal "awskms" {
  disabled = true
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101"
}
seal "awskms" {
  region = "us-east-1"
  kms_key_id = "987654321-dcba-4321-dcba-10987654321"
}
api_addr = "https://vault-us-east-1.hcvop.com:8200"
cluster_addr = "https://node-a-us-east-1.hcvop.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```



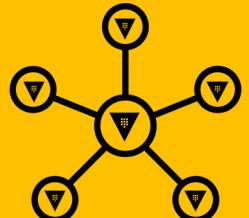


Implementing Integrated Storage



Vault Integrated Storage

Introduced in Vault 1.4, Integrated Storage provides a highly-available, durable storage backend without relying on any external systems



Uses the Raft Protocol

Integrated Storage uses the same underlying consensus protocol as Consul to handle cluster leadership and log management



Locally Stored Data

Vault data is stored locally on each node, and replicated to all other nodes in the cluster for high availability



Introduction to Integrated Storage



- All nodes in a Vault cluster have a replicated copy of Vault's data
- Eliminates the network hop of connecting from Vault to the external storage provider
- Also removes the administrative overhead of managing an external solution
 - When Vault has an outage, you only need to troubleshoot Vault



Introduction to Integrated Storage



- Since Integrated Storage stores all data on a local disk, it is recommended that you use storage optimized, high IOPS volumes wherever you are storing the data



Integrated Storage Features



Replication

Integrated Storage is fully supported for Vault Enterprise environments using Performance and/or DR replication across data centers

Auto Snapshots

A "Set and Forget" DR feature that schedules raft snapshots and copies them to cloud storage (Enterprise feature)

Cloud Autojoin

Stop joining clusters by hostname or IP address and discover other Vault nodes using cloud-based tags attached to your resources

Autopilot

Increase operational efficiency with automated features to help you manage your Vault clusters



Benefits of Integrated Storage Over Other Solutions



Integrated Storage can be used as a Storage Backend to avoid deploying and managing an external solution



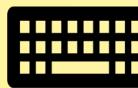
Reduced Complexity

All configuration is done within Vault. No external systems to provision alongside of Vault.



Decreased Costs

Fewer resources required for an enterprise-ready highly-available solution.



Easier to Troubleshoot

No external system to troubleshoot since Integrated Storage is a built-in solution. Storage is not memory-bound like Consul.



Integrated Storage Benefits



Similar Architecture

Same quorum requirements as Consul, so it's familiar



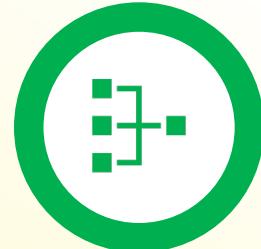
Fewer Networking Requirements

Raft only uses two ports for cluster operations and replication



Not Memory-Bound

Stores data on local disk rather than in memory

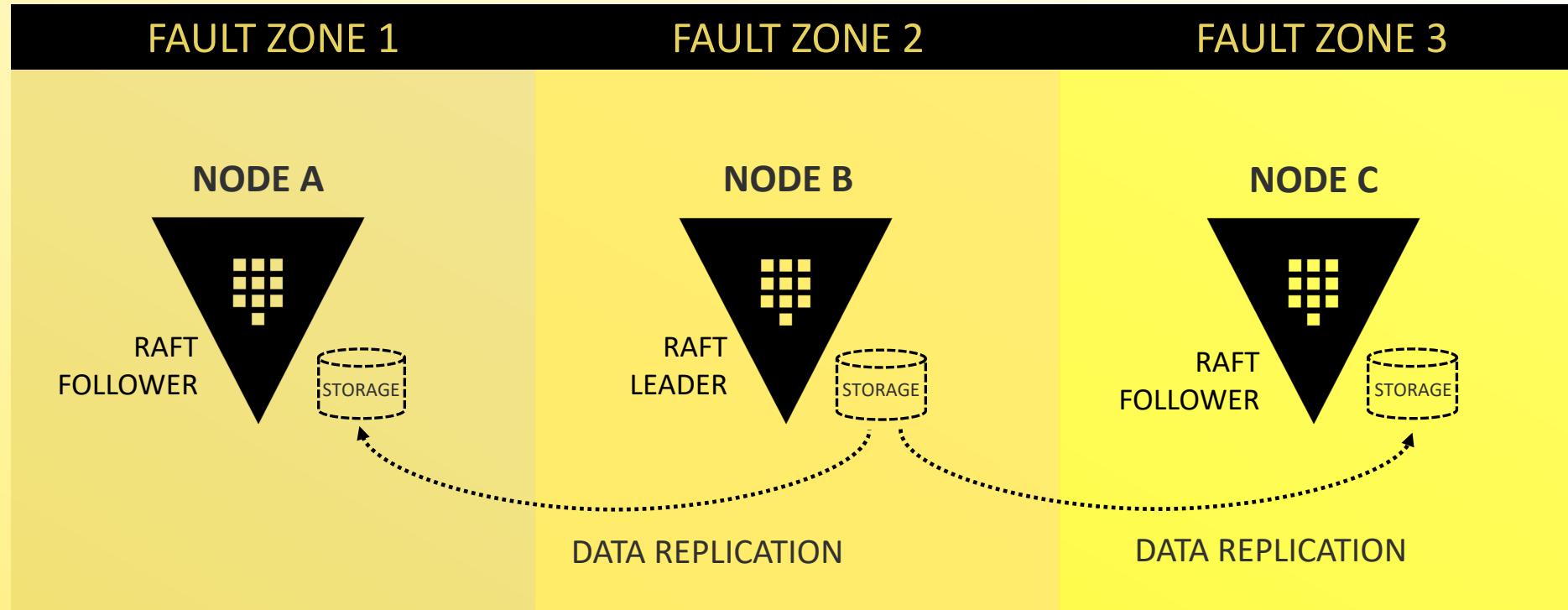


No Network Hops Required

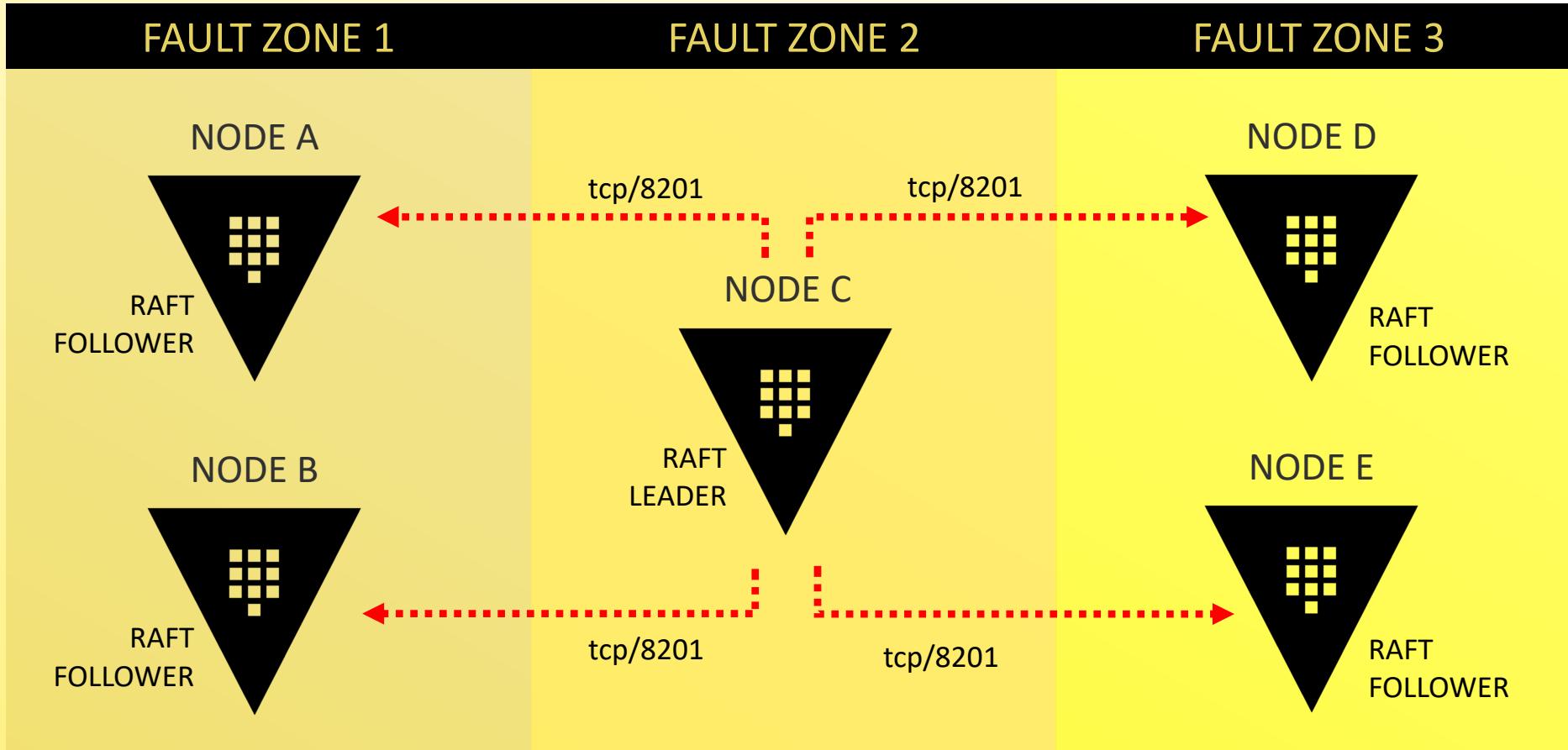
Data is stored locally and doesn't need to be retrieved over the network



Reference Architecture – Development Cluster

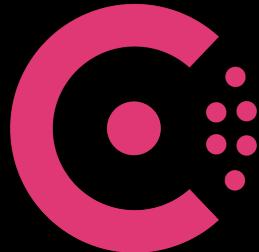


Reference Architecture – Production Cluster



Networking Requirements

CONSUL BACKEND



REQUIRED PORTS

Vault – 8200 – API
Vault – 8201 – RPC/Replication

Consul – 8500 – API
Consul – 8300 – RPC
Consul – 8301 - Serf

INTEGRATED STORAGE



REQUIRED PORTS

Vault – 8200 – API
Vault – 8201 – RPC/Replication

FEWER PORTS FOR COMMUNICATION

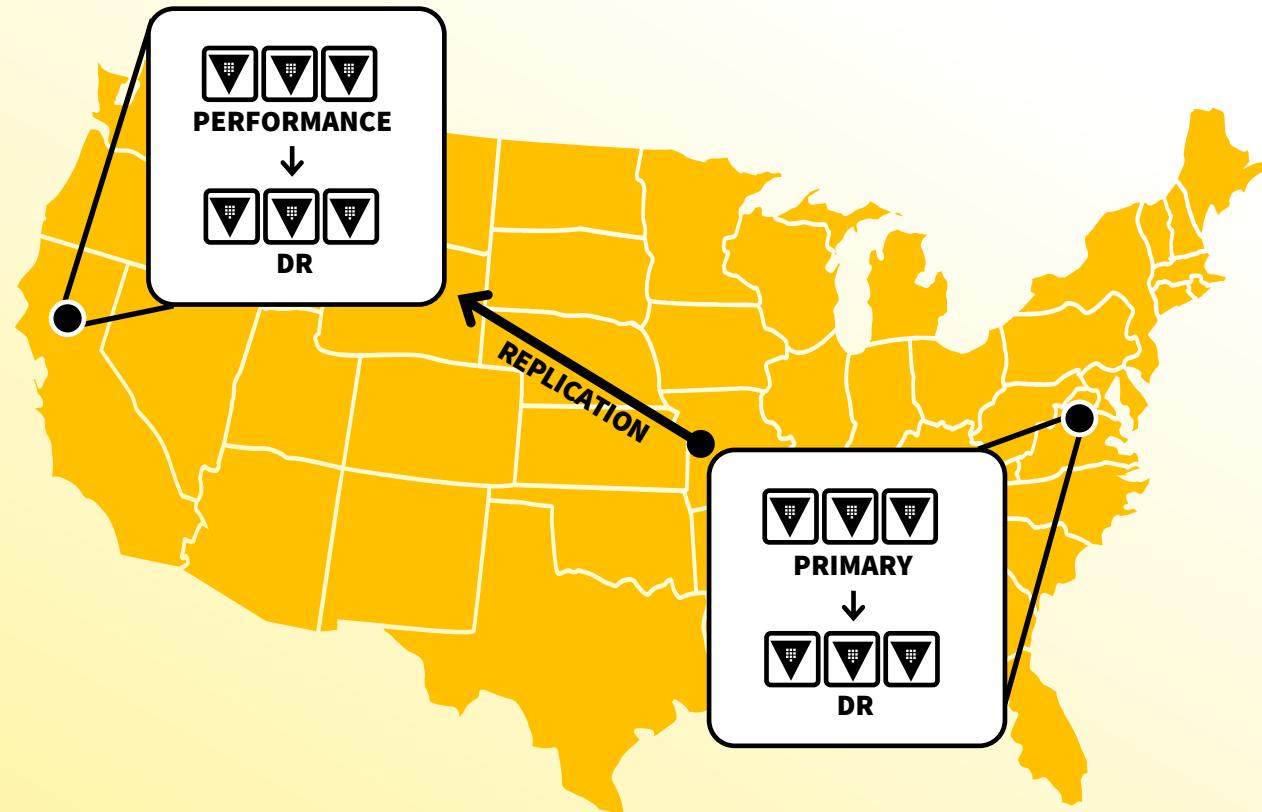
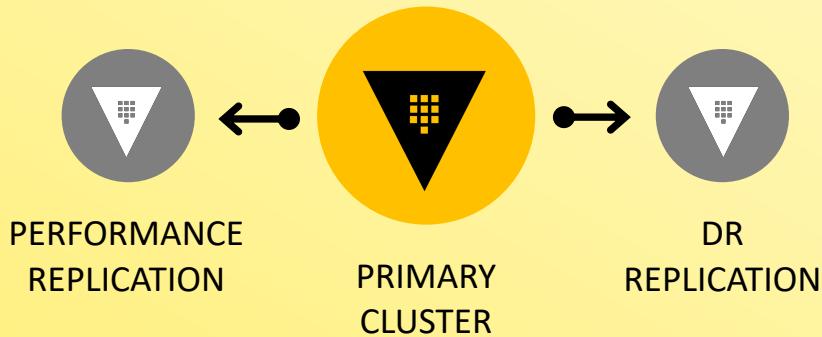
Fewer ports for communication means less attack surface for your Vault infrastructure and fewer ports required for internal firewalls.



Replicated Environment

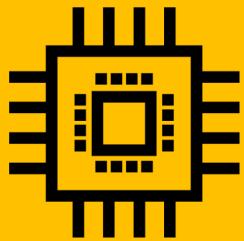
Enterprise-Level Deployments

Organizations relying heavily on Vault will commonly have multiple clusters deployed in multiple data centers/cloud-based regions for high-availability and disaster recovery.



Performance Requirements

CPU & MEMORY



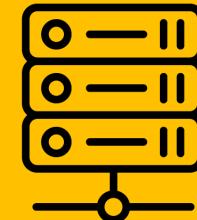
Consolidating both Vault and Consul workloads on the Vault nodes so consider increasing resources

STORAGE



Integrated Storage is disk-bound, so use high-performing disks and sufficient volume sizes

NETWORKING



Similar to Consul, replication requires low latency, high throughput connectivity



Configuring Integrated Storage

Configured in the Vault configuration file



```
...
listener "tcp" {
  address = "0.0.0.0:8200"
  cluster_address = "0.0.0.0:8201"
...
}
storage "raft" {
  path = "/opt/vault/data"
  node_id = "vault-node-a.hcvop.com"
  retry_join {
    auto_join = "provider=aws region=us-east-1 tag_key=vault tag_value=us-east-1"
  }
  performance_multiplier = 1
}
api_addr = "https://vault.hcvop.com:8200"
cluster_addr = " https://vault-node-a.hcvop.com:8201"
```

*Configuration truncated



Configuring Integrated Storage

Common Configurations



- storage "raft" = use integrated storage for the node/cluster
- path = local directory to store data
- node_id = name of the local node – cannot be duplicated within a cluster
- retry_join = Vault nodes to communicate with and join a cluster
- performance_multiplier = configure the time it takes Vault to detect leader failures and to perform leader elections



Configuring Integrated Storage

Retry Join Block



- `leader_api_addr` = address of a potential leader in the cluster (IP or DNS)
or
- `auto_join` = use cloud auto-join config using tags assigned to Vault nodes
- `leader_ca_cert_file` = file path of the CA cert of possible leader node
- `leader_client_cert_file` = file path to the client cert to establish client auth with the possible leader node
- `leader_client_key_file` = file path to the client key to establish client auth with the possible leader node



Configuring Integrated Storage

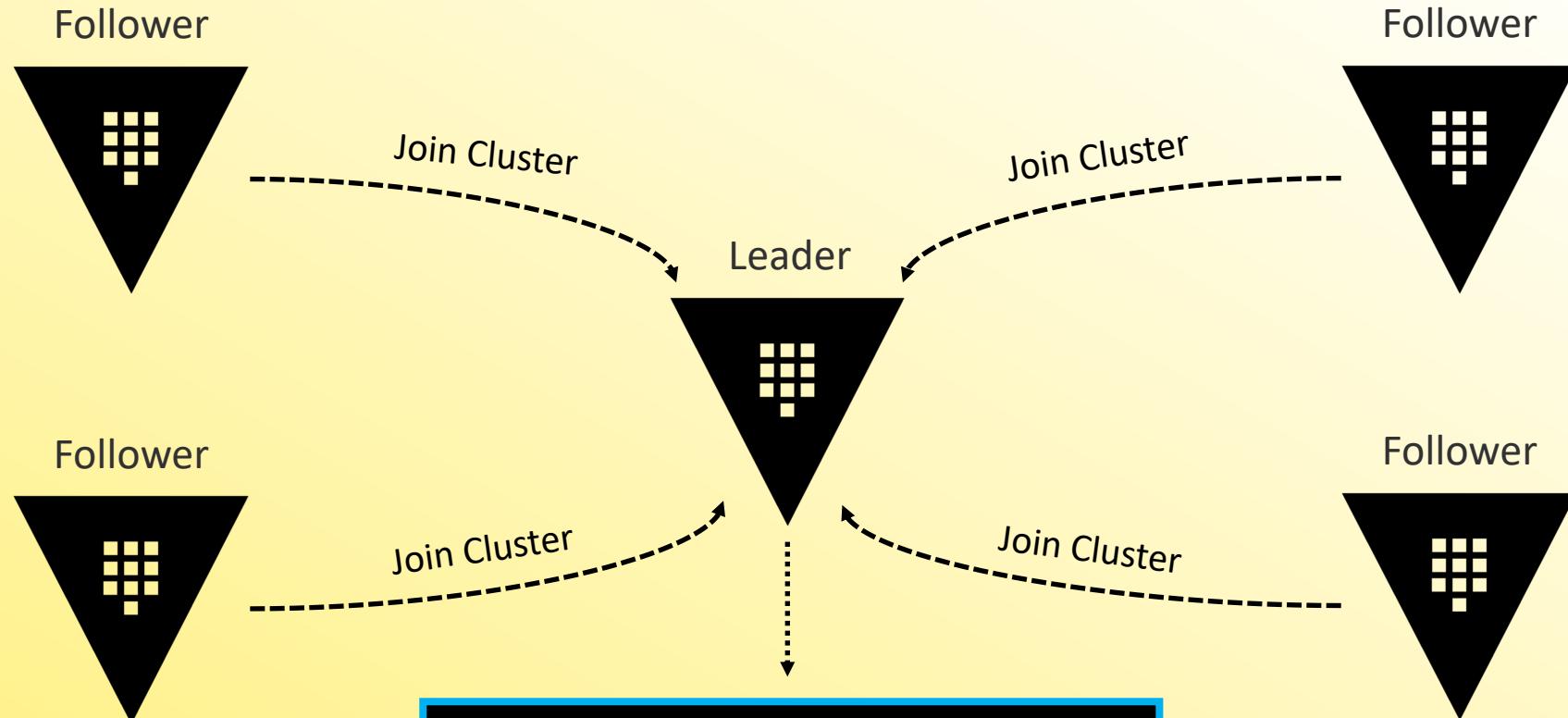
Retry Join Blocks



```
storage "raft" {
  path = "/opt/vault/data"
  node_id = "vault-node-a.hcvop.com"
  retry_join {
    leader_api_addr = https://vault-node-b.hcvop.com:8200
    leader_ca_cert_file = "/opt/vault.d/ca.pem"
    leader_client_cert_file = "/opt/vault.d/cert.pem"
    leader_client_key_file = "/opt/vault.d/pri.key"
  }
  retry_join {
    leader_api_addr = https://vault-node-c.hcvop.com:8200
    leader_ca_cert_file = "/opt/vault.d/ca.pem"
    leader_client_cert_file = "/opt/vault.d/cert.pem"
    leader_client_key_file = "/opt/vault.d/pri.key"
  }
}
```



Cluster Configuration Workflow



```
$ vault operator init  
$ vault operator unseal
```



Managing Integrated Storage



Vault CLI:

```
vault operator raft <subcommand> [options] [arguments]
```

Subcommand	Description
list-peers	Returns the raft cluster member information
join	Joins a node to the cluster
remove-peer	Removes a node from the cluster
snapshot	Save or restore a raft snapshot from the cluster



Raft Operations



```
# Join a Node to an existing (or new) Cluster  
$ vault operator raft join https://vault-0.hcvop.com:8200  
  
# Remove a Node from a cluster  
$ vault operator raft leave vault-4  
Peer removed successfully!
```

Name of the node
to be removed



Raft Operations



```
# List the cluster members
$ vault operator raft list-peers

Node          Address        State      Voter
----          -----        ----      -----
vault-0       vault-0.hcvop:8201   leader    true
vault-1       vault-1.hcvop:8201   follower  true
vault-2       vault-2.hcvop:8201   follower  true
vault-3       vault-3.hcvop:8201   follower  true
vault-4       vault-4.hcvop:8201   follower  true
```



Raft Snapshots



Integrated Storage includes the ability to create a snapshot manually or on a scheduled configuration (Ent only)

Snapshot is a point-in-time backup that includes configuration data and data contained within the KV stores



Create a Raft Snapshot



```
# Save a snapshot
$ vault operator raft snapshot save daily.snap
```

```
# Log Entries after snapshot
2022-04-18T16:41:09.545-0400 [INFO] storage.raft: starting snapshot up to: index=389
2022-04-18T16:41:09.585-0400 [INFO] storage.raft: snapshot complete up to: index=389
```



Restore Raft Snapshot



```
# Restore a snapshot
$ vault operator raft snapshot restore daily.snap

# Log Entries after snapshot
2022-12-18T16:42:26.298-0400 [INFO]  core: applying snapshot
2022-12-18T16:42:26.298-0400 [INFO]  storage.raft.snapshot: creating new snapshot:
path=/opt/vault/data/raft/snapshots/6-422-1654546298.tmp
2022-12-25T16:42:26.466-0400 [INFO]  storage.raft: copied to local snapshot: bytes=54038
2022-12-25T16:42:26.482-0400 [INFO]  storage.raft.fsm: installing snapshot to FSM
2022-12-25T16:42:26.504-0400 [INFO]  storage.raft.fsm: snapshot installed
2022-12-25T16:42:26.586-0400 [INFO]  storage.raft: restored user snapshot: index=1
2022-12-25T16:42:31.708-0400 [INFO]  core: running post snapshot restore invalidations
```





Auth Methods



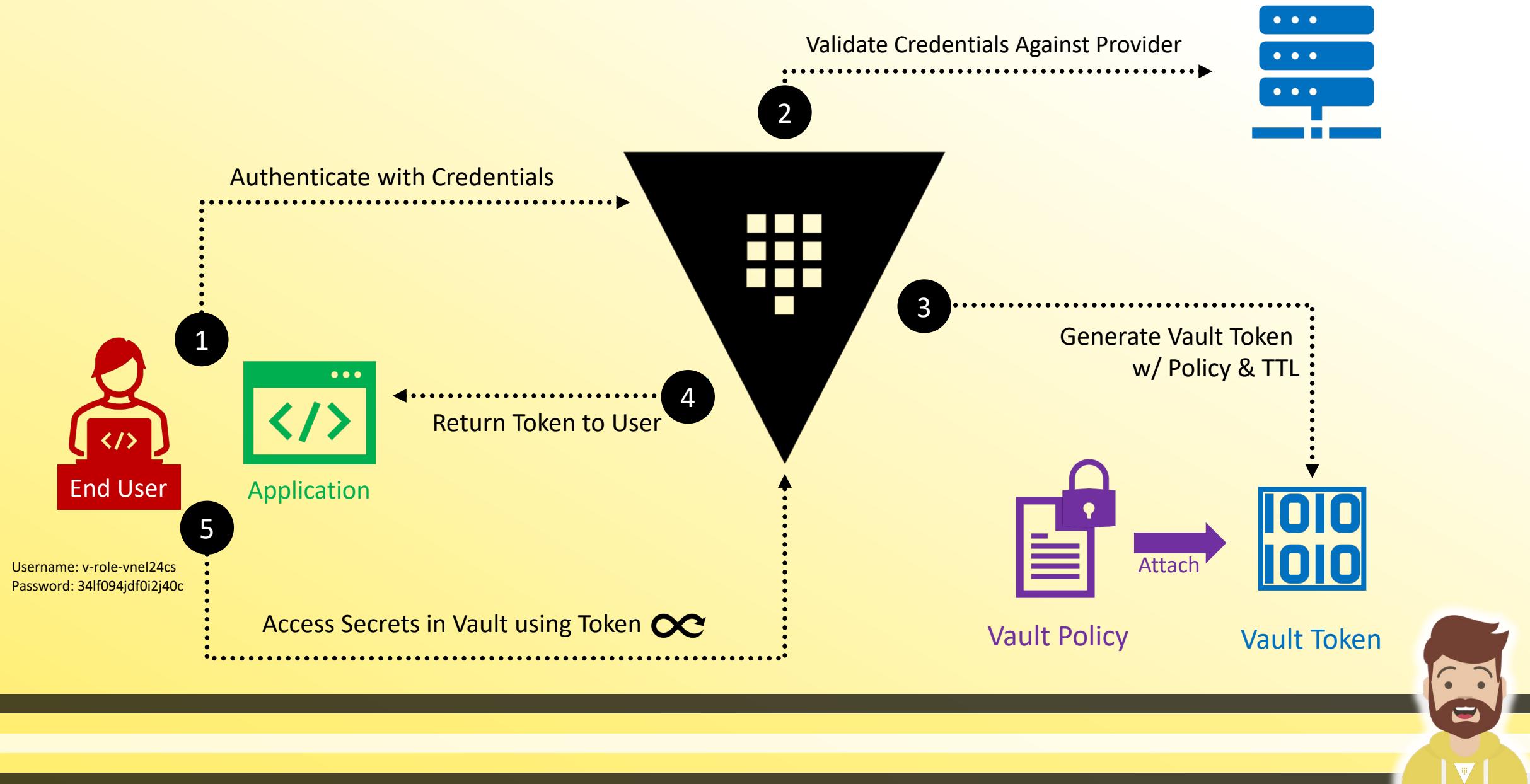
Introduction to Auth Methods



- Vault components that perform authentication and manage identities
- Responsible for assigning identity and policies to a user
- Multiple authentication methods can be enabled depending on your use case
 - Auth methods can be differentiated by human vs. system methods
- Once authenticated, Vault will issue a client token used to make all subsequent Vault requests (read/write)
 - The fundamental goal of all auth methods is to obtain a token
 - Each token has an associated policy (or policies) and a TTL



Auth Methods Workflow



Auth Methods



AppRole

aws



kubernetes



Microsoft Azure



ORACLE
CLOUD



GitHub



TOKENS



CLOUD FOUNDRY



JWT/OIDC



TLS Certificates



Alibaba Cloud

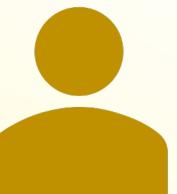
okta



RADIUS



Username/Password



Kerberos



Auth Methods Likely on the Exam



AppRole



Tokens



UserPass



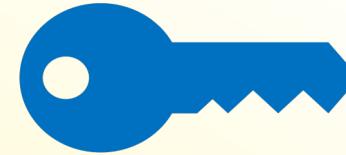
Auth Methods

Human-Based

- Integrates with an Existing Identity Provider
- Requires a Hands-On Approach to Use
- Logging in via Prompt or Pop-up
- Often configured with the Platforms Integrated MFA



JWT/OIDC



Userpass



RADIUS

okta



Auth Methods

System-Based

- Uses non-human friendly methodologies (not easy to remember)
- Usually Integrates with an Existing Platform
- Vault validates credentials with the platform



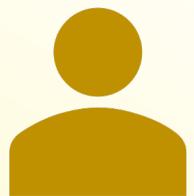
TOKENS



CLOUD FOUNDRY



TLS Certificates



Kerberos



Microsoft Azure



kubernetes



Working with Auth Methods



- Most auth methods must be enabled before you can use them
- One or many auth methods can be used at any given time
 - Generally different auth methods are used for different use cases (app vs. human)
- The token auth method is enabled by default, and you cannot enable another nor disable the tokens auth method
 - New Vault deployment will use a token for authentication
 - Only method of authentication for a new Vault deployment is a root token



Working with Auth Methods



Auth methods can be enabled/disabled and configured using the UI, API, or the CLI

- Note that the UI isn't fully-featured like the CLI and API, so there might be things you can't do in the UI

You must provide a valid token to enable, disable, or modify auth methods in Vault. The token must also have the proper privileges.

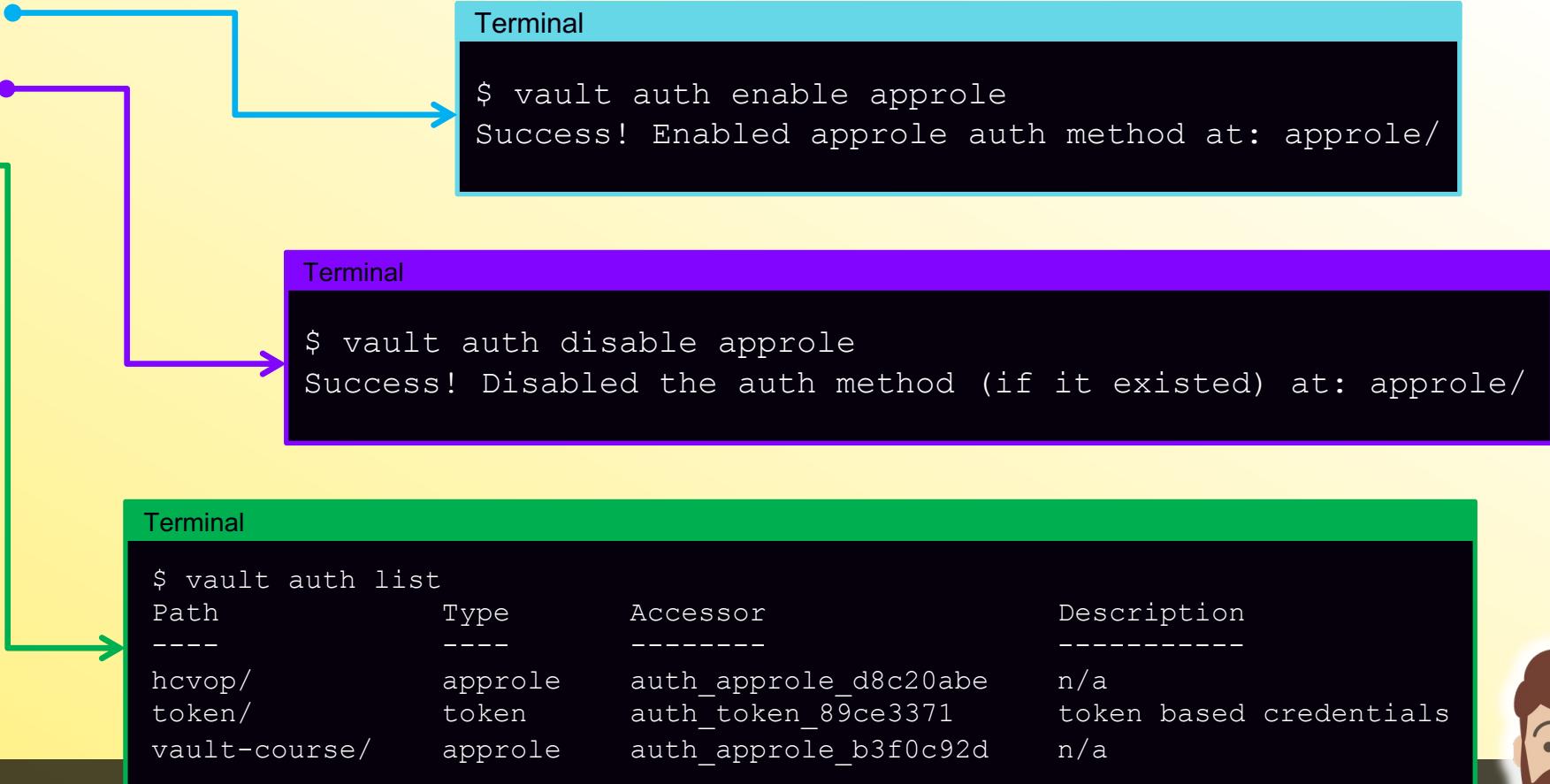


Working with Auth Methods



Use the vault auth command

- enable
- disable
- list
- tune
- help



Working with Auth Methods



Enable an Auth Method at the Default Path

A blue terminal window icon with three colored dots (red, yellow, green) in the top-left corner.

Terminal

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

Enable an Auth Method using a Custom Path

A yellow terminal window icon with three colored dots (red, yellow, green) in the top-left corner.

Terminal

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



Working with Auth Methods

Enable Auth Method Using Default Path via CLI



```
$ vault auth enable approle
```



Type of Vault
object you want
to work with



Subcommand



Type of Auth
Method



Working with Auth Methods

Enable Auth Method with Custom Path via CLI



```
$ vault auth enable -path=apps approle
```



Type of
Vault object
you want to
work with



Subcommand



Customize the Path
Name



Type of Auth
Method



Working with Auth Methods



After the auth method has been enabled, use the auth prefix to configure the auth method:

- Syntax: vault write auth/<path name>/<option>

The diagram shows the terminal command \$ vault write auth/approle/role/hcvop \ secret_id_ttl=10m \ token_num_uses=10 \ token_ttl=20m \ token_max_ttl=30m \ secret_id_num_uses=40. Four arrows point from the text above to specific parts of the command: a red arrow points to the first 'auth/' in 'auth/approle/role/hcvop', a purple dashed arrow points to the second 'auth/' in 'auth/approle/role/hcvop', a green arrow points to the '\', and a blue arrow points to the second '\'. This illustrates how the 'auth' prefix is used to configure specific authentication methods.

```
Terminal
$ vault write auth/approle/role/hcvop \
secret_id_ttl=10m \
token_num_uses=10 \
token_ttl=20m \
token_max_ttl=30m \
secret_id_num_uses=40
```



Working with Auth Methods

Using the API



Enabling an Auth Method:

- Method: POST

Enable
Auth
Method

```
Terminal
$ curl \
  --header "X-Vault-Token: s.v2otcpHygZHWiD7BQ7P5aJjL" \
  --request POST \
  --data '{"type": "approle"}' \
  https://vault.hcvop.com:8200/v1/sys/auth/approle
```

API Endpoint

Don't forget you need
a valid token

Alternatively, you can point
to a file here if you want
--data @data.json





AppRole Auth Method



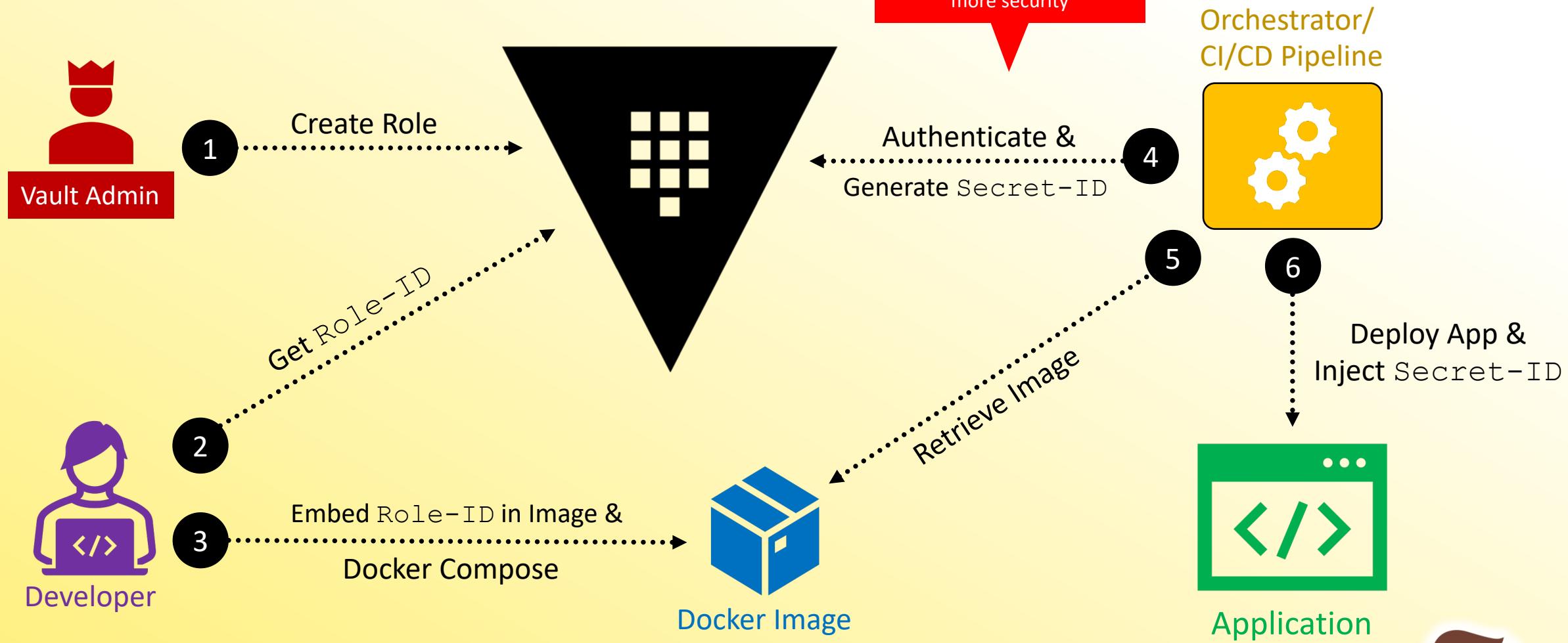
Introduction to AppRole



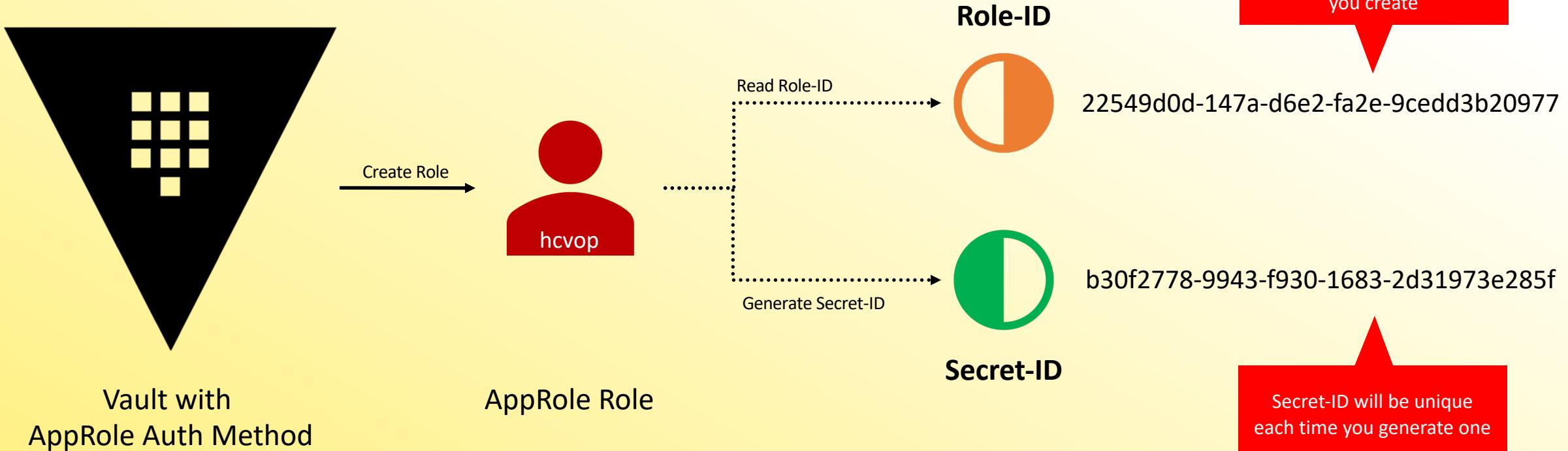
- AppRole auth method enables machines or applications to authenticate to Vault using a pre-defined *role*.
 - A *role* represents a one-to-one mapping between client authentication and the Vault permission requirements
 - Each defined role has a static *role-id* and can have zero or many *secret-ids* that can be generated and used for authentication
 - *Example: A fleet of web servers can all use the same role-id but each have a unique secret-id*
- This auth method is oriented for use by machines/automated services and is not very useful for human clients



AppRole – Auth Workflow



AppRole – Configuration Workflow



AppRole – Configuration Workflow

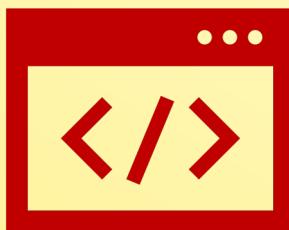


Needed for authentication

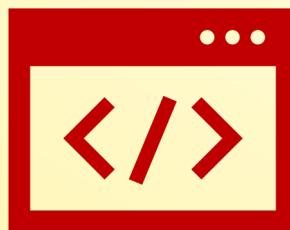
Fleet of Web Servers

Requires Identical Permissions in Vault

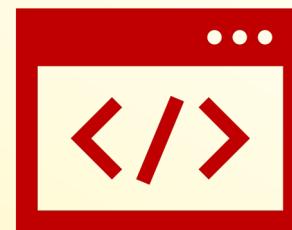
Application



Application



Application



Role-ID 22549d0d-147a-d6e2-fa2e-9cedd3b20977

Secret-ID b30f2778-9943-f930-1683-2d31973e285f

22549d0d-147a-d6e2-fa2e-9cedd3b20977

0514b3b1-e1ce-2741-0b57-ef836c29c7d3

22549d0d-147a-d6e2-fa2e-9cedd3b20977

d2628ca1-4683-a285-f55d-186d9b10e530

Unique Secret-ID for each Workload



Enable AppRole Auth Method



```
# Enable AppRole at the default path
$ vault auth enable approle
Success! Enabled approle auth method at: approle/
```

```
# Enable AppRole at a custom path
$ vault auth enable -path=hcvop approle
Success! Enabled approle auth method at: hcvop/
```



Create a New Role



```
# Create a new role named 'hcvop'

$ vault write auth/approle/role/hcvop \
  token_policies=web-app \
  token_ttl=1h \
  token_max_ttl=24h
Success! Data written to: auth/approle/role/hcvop/
```



Create a New Role

Breaking Down the CLI Command



vault write auth/approle/role/hcvop



Subcommand
used to read,
write, delete,
or list a role

Default path
for auth
methods (not
configurable)

Path where the
AppRole auth
method was
enabled

Path where
roles are
created and
stored (not
configurable)

The name of
the role you
want to
create,
modify, or
delete



Create a New Role

A Few Configuration Tips for AppRole



- You can set the TTL of the resulting token
 - `token_max_ttl=24h`
 - `token_ttl=1h`
- You can set the TTL of the secret-id that you have generated
 - `secret_id_ttl=24h`
- You can configure CIDR restrictions for a role (binds the resulting token as well)
 - `token_bound_cidrs="10.1.16.0/16"`
- You can change the resulting token type to a batch token
 - `token_type=batch`

These are configuration parameters for an AppRole role



Reading the Role-ID



```
# Read the role-id for a particular role  
  
$ vault read auth/approle/role/hcvop/role-id  
Key          Value  
---          ----  
role_id      22549d0d-147a-d6e2-fa2e-9cedd3b20977
```

Every time you run this command for the **same role name**, you will get the same exact value in response



Read a Role Configuration



```
# Read the current configuration of the role named 'hcvop'

$ vault read auth/approle/role/hcvop
Key          Value
---          -----
bind_secret_id      true
local_secret_ids    false
policies           [web-app]
secret_id_bound_cidrs <nil>
secret_id_num_uses   0
secret_id_ttl        0s
token_bound_cidrs    [10.1.16.0/16]
token_explicit_max_ttl 0s
token_max_ttl        24h
token_no_default_policy false
token_num_uses       0
token_period         0s
token_policies       [web-app]
token_ttl            1h
token_type           default
```



Generating a Secret-ID



```
# Generate a secret-id for a particular role

$ vault write -f auth/approle/role/hcvop/secret-id
Key          Value
---
secret_id    0514b3b1-e1ce-2741-0b57-ef836c29c7d3
secret_id_accessor da025e1f-7247-1888-218c-37382d31e98e
secret_id_ttl 24h
```

Every time you run this command for the **same role name**, you will get a **different secret-id**

You will need to include `-f` or `-force` to run this command



Authenticate to Vault using AppRole

Via Command Line (CLI)



```
$ vault write auth/approle/login \
role_id=22549d0d-147a-d6e2-fa2e-9cedd3b20977 \
secret_id=b30f2778-9943-f930-1683-2d31973e285f
```

Key	Value
---	-----
token	hvs.CAESIGjTXNYnz9T4h5y2_5Rt_d5ZNhW3dfgU4gQ
token_accessor	KmRLXSRbOzhXoA746g8g0KJu
token_duration	24h
token_renewable	true
token_policies	["default" "web-app"]
identity_policies	[]
policies	["default" "web-app"]
token_meta_role_name	hcvop



Authenticate to Vault using AppRole

Via API



```
# Format the response using jq
$ curl \
  --request POST \
  --data '{"role_id":"22549d0d-...","secret_id":"b30f2778-..."}' \
  https://vault.hcvop.com:8200/v1/auth/approle/login | jq

# Authenticate and query for the client_token using jq
$ curl \
  --request POST \
  --data '{"role_id":"22549d0d-...","secret_id":"b30f2778-..."}' \
  https://vault.hcvop.com:8200/v1/auth/approle/login | jq -r
  '.auth.client_token'
```



Authenticate to Vault using AppRole

Via API



```
{  
    "request_id": "c3750ef1-9fdf-b2db-0e7b-cfb433bf4120",  
    "lease_id": "",  
    "renewable": false,  
    "lease_duration": 0,  
    "data": null,  
    "wrap_info": null,  
    "warnings": null,  
    "auth": {  
        "client_token": "hvs.CAESIIJCoQiCpcI-U0xiqGr0fG9pP0SHDgfVB96cti7bSI3aGiEKGh2cy5vdWlPd2pEZ2RwVHZaWm95VWZoMnN1bmkQywE",  
        "accessor": "rlbbs2nFlRvr348p9qaALM7O",  
        "policies": [  
            "default",  
            "web-app"  
        ],  
        "token_policies": [  
            "default",  
            "web-app"  
        ],  
        "metadata": {  
            "role_name": "hcvop"  
        },  
        "lease_duration": 2764800,  
        "renewable": true,  
        "entity_id": "58a97ed8-0003-c32f-b061-3ef9817a628e",  
        "token_type": "service",  
        "orphan": true,  
        "mfa_requirement": null,  
        "num_uses": 0  
    }  
}
```

Get the resulting token by parsing the JSON response:

```
jq -r '.auth.client_token'
```





Userpass Auth Method



Introduction to Userpass



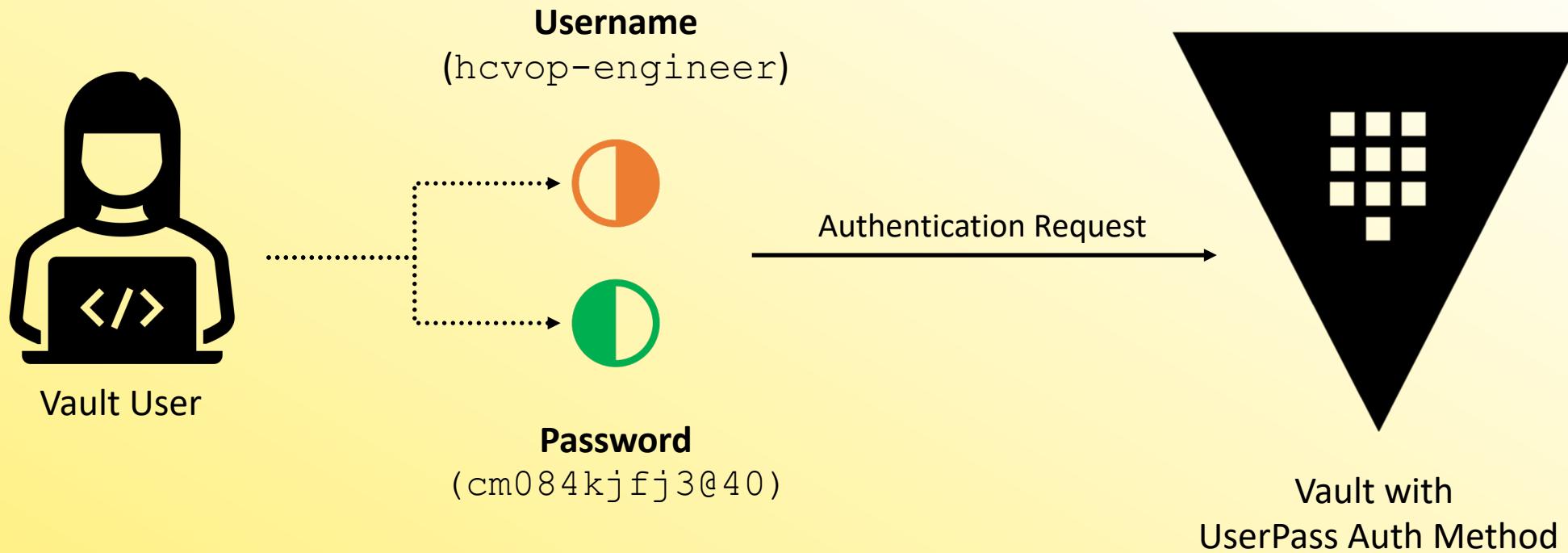
- Userpass auth method enables Vault clients to authenticate using a local username and password
- Userpass does NOT integrate or read credentials from an external identity provider. Everything is local to Vault itself.



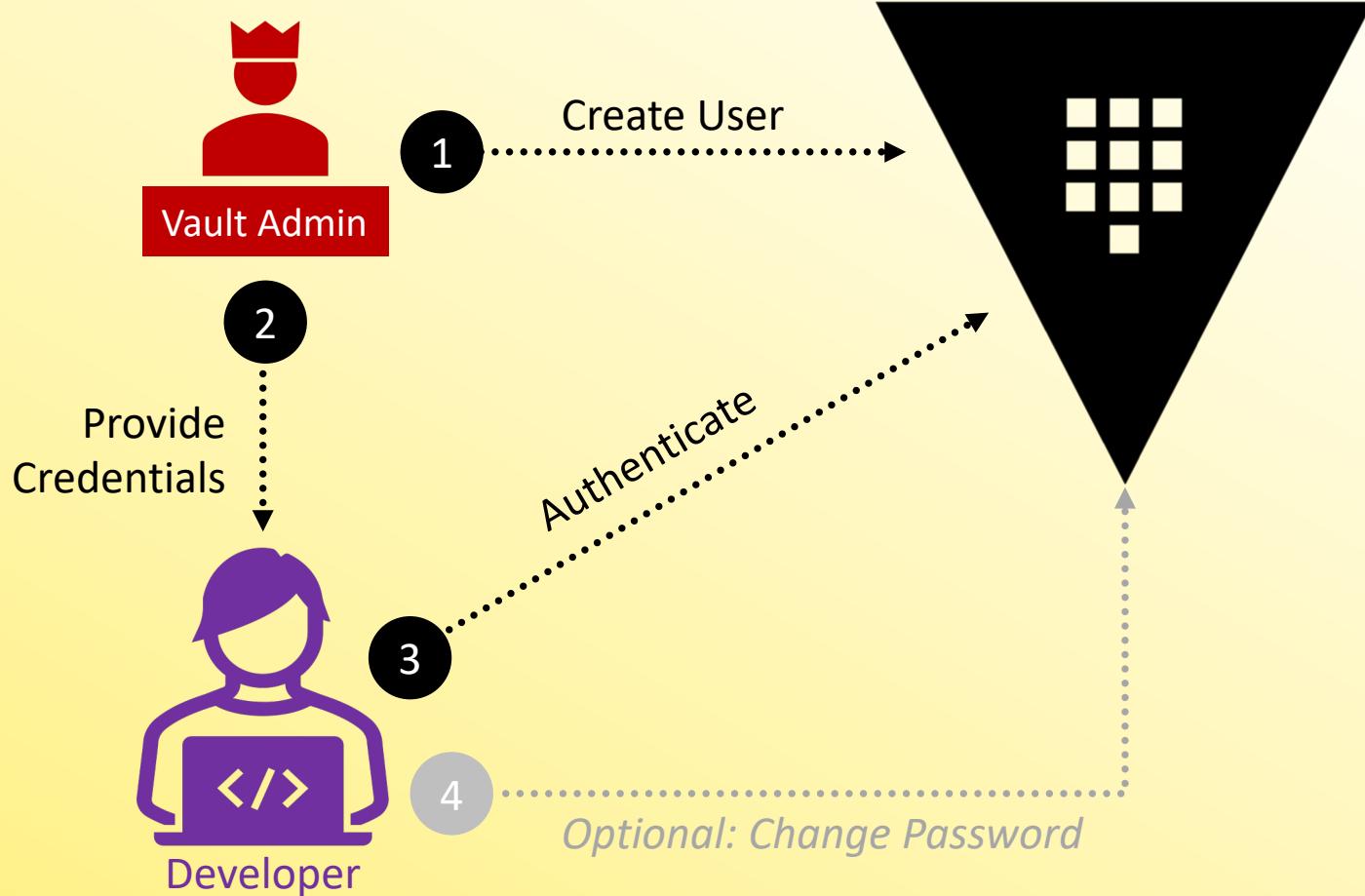
Userpass – Auth Workflow



Needed for authentication



Userpass – Configuration Workflow



Enable Userpass Auth Method



```
# Enable Userpass at the default path
$ vault auth enable userpass
Success! Enabled userpass auth method at: userpass/

# Enable Userpass at a custom path
$ vault auth enable -path=vault-local userpass
Success! Enabled userpass auth method at: vault-local/
```



Create a New User



```
# Create the new user hcvop-engineer and assign a policy\n\n$ vault write auth/userpass/users/hcvop-engineer \  
  password=cm084kjfj3@40 \  
  policies=engineering-policy \  
  token_ttl=15m \  
  token_max_ttl=8h \  
Success! Data written to: auth/userpass/users/hcvop-engineer
```



Create a New User

Breaking Down the CLI Command



```
vault write auth/userpass/users/hcvop-engineer
```



Subcomm
and used
to read,
write,
delete, or
list a role

Default path
for auth
methods (not
configurable)

Path where the
Userpass auth
method was
enabled

Path where
users are
created and
stored (not
configurable)

The name of the user you
want to create, modify, or
delete



Create a New User

A Few Configuration Tips for Userpass



- You can set the TTL of the resulting token:
 - `token_max_ttl=24h`
 - `token_ttl=1h`
- You can change the token type to be a batch token:
 - `token_type=batch`
- You can configure the token to be a use-limited token:
 - `token_num_uses=5`
- You can configure CIDR restrictions for a token:
 - `token_bound_cidrs="10.1.16.0/16"`

These are configuration parameters
for a Userpass user



Read a User Configuration



```
# Read the current configuration of the hcvop-engineer user

$ vault read auth/userpass/users/hcvop-engineer
Key          Value
---          -----
policies      [engineering-policy]
token_bound_cidrs []
token_explicit_max_ttl 0s
token_max_ttl    8h
token_no_default_policy false
token_num_uses   0
token_period     0s
token_policies   [engineering-policy]
token_ttl        15m
token_type       default
```



Authenticate with Userpass Credentials



```
# Authenticate with hcvop-engineer user

$ vault login -method=userpass username=hcvop-engineer
Password (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.CAESIKcXGsfBA8pzKwBgAQ1Xferekt1n9S5PC0mgz36EKHy5zR2RFR
token_accessor 2Pm2ybDoAyRqrrhMEfDi674N
token_duration 15m
token_renewable true
token_policies ["default" "engineering-policy"]
identity_policies []
policies      ["default" "engineering-policy"]
token_meta_username hcvop-engineer
```



Update a Password for a User



```
# Authenticate with hcvop-engineer user  
  
$ vault write auth/userpass/users/hcvop-engineer/password password=xmeij9dk20je  
Success! Data written to: auth/userpass/users/hcvop-engineer/password
```





Token Auth Method



Introduction to Tokens



- Tokens are the core method for authentication
 - Most operations in Vault require an existing token (not all, though)
 - Accessing a login path doesn't, for example
- The token auth method is responsible for creating and storing tokens
 - The token auth method cannot be disabled
 - Tokens can be used directly, or they can be used with another auth method
 - Authenticating with an external identity (e.g. LDAP) dynamically generate tokens
- Tokens have one or more policies attached to control what the token is allowed to perform



Introduction to Tokens



- **service** tokens are the default token type in Vault
 - They are persisted to storage (heavy storage reads/writes)
 - Can be renewed, revoked, and create child tokens
 - Most often, you'll be working with service tokens
- **batch** tokens are encrypted binary large objects (blobs)
 - Designed to be lightweight & scalable
 - They are NOT persisted to storage, but they are not fully-featured
 - Ideal for high-volume operations, such as encryption
 - Can be used for DR Replication cluster promotion as well



Create a Periodic Token



```
$ vault token create -policy=hcvop -period=24h
```

Key	Value
---	-----
token	hvs.CAESINq3yTGLYZofP7iZBStz3zAktvOHfWBiN
token_accessor	fy9Jjse9SRTLIYLUFysE6qP0
token_duration	24h
token_renewable	true
token_policies	["default" "hcvop"]
identity_policies	[]
policies	["default" "hcvop"]



No Max TTL – You can renew a periodic token an infinite number of times



Create a Use-Limited Token



```
$ vault token create -policy="hcvop" -use-limit=2
Key          Value
---          -----
token        hvs.CAESIFu4Z5VF8AWeTgAXO89WCewUqZGVUd18Pj
token_accessor by7TufOOTCoUZRL86PRu0cil
token_duration 768h
token_renewable true
token_policies  ["default" "hcvop"]
identity_policies []
policies      ["default" "hcvop"]
```



Can only use this token **TWO** times and then it will be automatically revoked



Create a Orphan Token



```
$ vault token create -policy="hcvop" -orphan  
Key          Value  
---  
token        hvs.CAESIMQJcAGE...8jFP  
token_accessor wAtiKTSt1PmnL2zhL537FHBa  
token_duration 768h  
token_renewable true  
token_policies ["default" "hcvop"]  
identity_policies []  
policies      ["default" "hcvop"]
```



Token is NOT affected by the TTL/revocation of its parent token



Set the Token Type at the Auth Method



To configure the AppRole auth method to generate batch tokens:

```
TERMINAL
$ vault auth enable approle
$ vault write auth/approle/role/hcvop policies="engineering" \
  token_type="batch" \
  token_ttl="60s"
```

To configure the AppRole auth method to generate periodic tokens:

```
TERMINAL
$ vault write auth/approle/role/hcvop policies="hcvop" \
  period="72h"
```



Authenticate with a Token

A screenshot of a web browser window titled 'Vault' showing the 'Sign in to Vault' page. The URL is 127.0.0.1:8200/ui/vault/auth?with=token. A green annotation on the left side points to the 'Method' dropdown menu, which has 'Token' selected. Below it is a text input field labeled 'Token'. At the bottom is a blue 'Sign In' button. A small note at the bottom of the page says 'Contact your administrator for login credentials'.

Log in directly with a token

Sign in to Vault

Method

Token

Sign In

Contact your administrator for login credentials



Authenticate with a Token

A screenshot of the HashiCorp Vault UI in a web browser. The URL is 127.0.0.1:8200/ui/vault/secrets. The page shows a list of "Secrets Engines": "cubbyhole/" (with sub-item "cubbyhole_621b0b76"), "kv/" (with sub-item "v2_kv_e4e20e9a"), and "secret/" (with sub-item "v2_kv_589b99e1"). A context menu is open over the user icon in the top right corner. The menu items are "ROOT", "Restart guide", "Copy token" (which is highlighted with a red box), and "Sign out".

Copy the Token
You are Using



Authenticate with a Token



Client token must be sent in the X-Vault-Token HTTP header

- Note that Authorization: Bearer <token> is valid as well

Terminal

```
$ curl --header "X-Vault-Token: hvs.cDIPyitdJKSm46ydTXJ0saQR" \
--request POST \
--data '{ \"apikey\": \"3230sc$832d\" }'
https://vault.hcvop.com:8200/v1/secret/apikey/splunk
```

Terminal

```
$ curl --header "Authorization: Bearer hvs.cDIPyitdJKSm46ydTXJ0saQR" \
--request GET \
https://vault.hcvop.com:8200/v1/secret/data/apikey/splunk
```



Authenticate with a Token



```
$ 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.cDIPyitdJKSm46ydTXJ0saQR
token_accessor ggVElRJkK0mWS2uYt9Kdi0SL
token_duration 24h
token_renewable false
token_policies  ["engineering"]
identity_policies []
policies      ["engineering"]
```



Revoke a Token



Tokens can easily be revoked so they can no longer be used for authentication

- **This includes a root token**



A terminal window with a red header bar and three colored dots (red, yellow, green) on the left. The main area contains a command-line session:

```
$ vault token revoke hvs.cDIPyitdJKSm46ydTXJOsaQR
Success! Revoked token (if it existed)
```

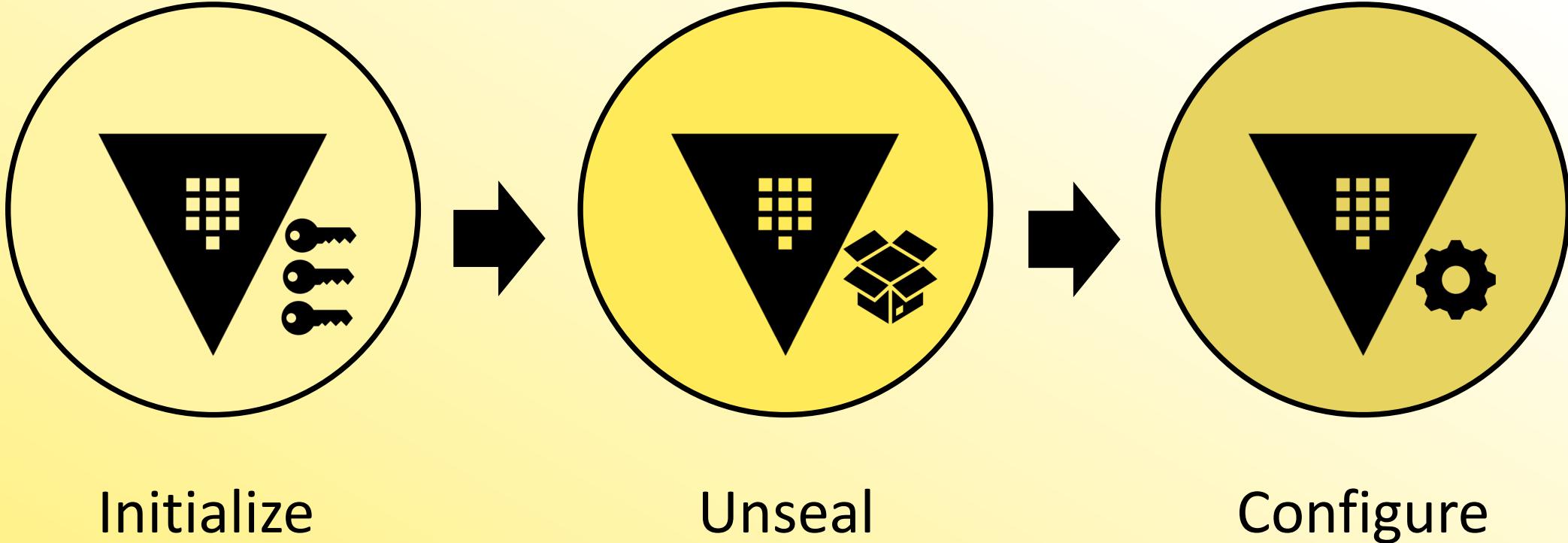




Practice Secure Vault Initialization



Vault Deployment Workflow



Vault Initialization



Initialize

- Vault initialization is when Vault creates the master key and key shares
- Options to define thresholds, key shares, recovery keys, and encryption
- Vault initialization is also where the initial root token is generated and returned to the user



Vault Initialization

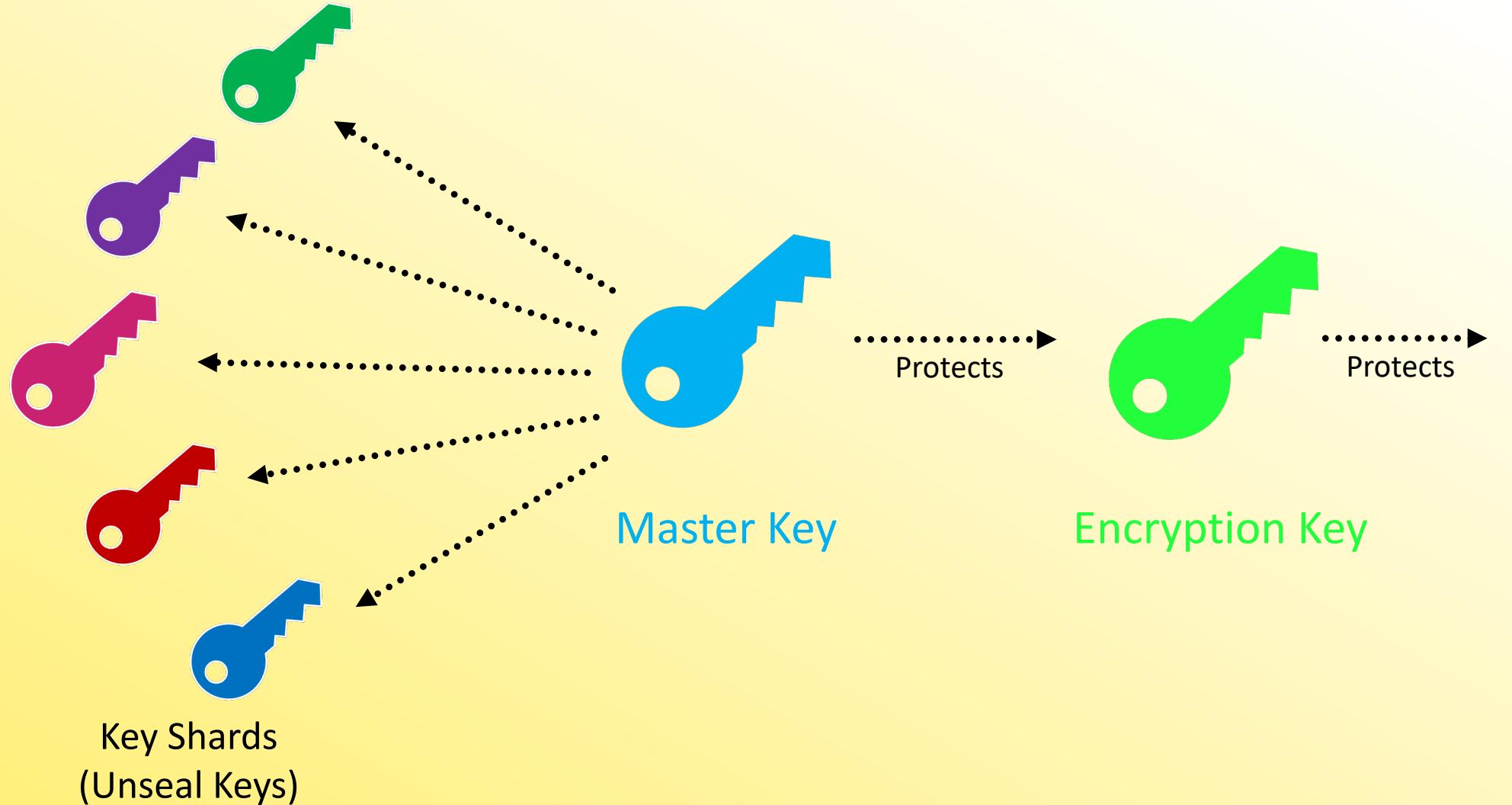


Initialize

How can you protect these keys and root token?



Vault Initialization



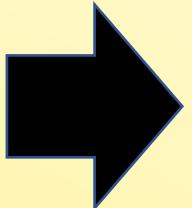
Provide Keys to Trusted Employees



But...Only a Single Person Can Initialize Vault



```
$ vault operator init
```



Wait...he has all the keys? That doesn't seem very secure





How can we better protect our keys
and root token while following the
Vault security model?



Vault Initialization



Common options for: vault operator init

Change the Number of Unseal Keys/Threshold

```
-key-shares=5  
-key-threshold=3
```

Change the Number of Keys/Threshold for Auto Unsealed Clusters

```
-recovery-shares=5  
-recovery-threshold=3
```



Vault Initialization



Common options for: vault operator init

**Encrypt the Unseal Keys
with Provided PGP Keys**

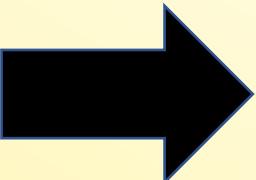
-pgp-keys=<keys>

**Encrypt the Recovery Keys with
the Provided PGP Keys**

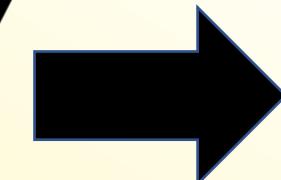
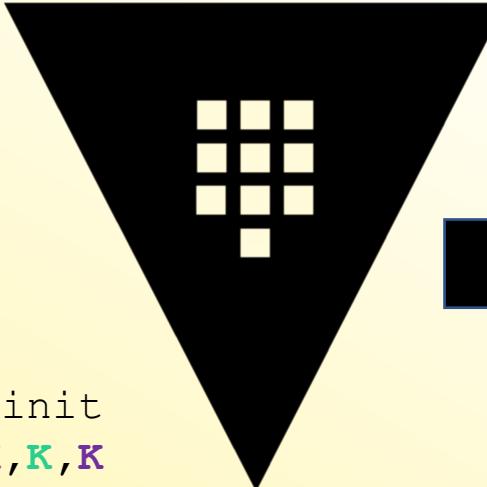
-recovery-pgp-keys=<keys>



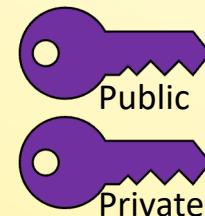
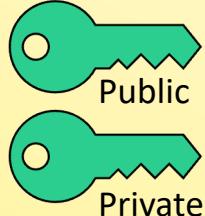
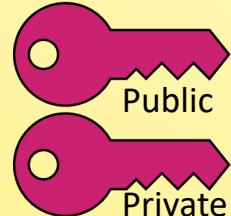
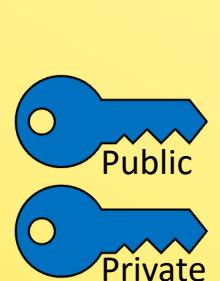
Secure Vault Initialization



vault operator init
-pgp-keys=K,K,K,K,K



Encrypted
Recovery
Keys



Secure Vault Initialization

For Default Unseal Keys



Terminal

```
$ vault operator init \
  -key-shares=5 \
  -key-threshold=3 \
  -pgp-keys="/opt/bob.pub,/opt/steve.pub,
  /opt/stacy.pub,/opt/katie.pub,/opt/dani.pub"
```



Secure Vault Initialization

For Auto Unsealed Clusters



Terminal

```
$ vault operator init \
  -recovery-shares=5 \
  -recovery-threshold=3 \
  -recovery-pgp-keys="/opt/bob.pub,/opt/steve.pub,
  /opt/stacy.pub,/opt/katie.pub,/opt/dani.pub"
```

The number of keys and PGP keys provided must match

Vault will output the encrypted recovery keys using the PGP keys in the order they were provided



Protecting the Root Token



Initialize

Unseal or
Recovery Keys

Initial Root Token



Protecting the Root Token



Option for: vault operator init

Encrypt the Root Token with Provided PGP Key

-root-token-pgp-key=<key>



Secure Vault Initialization

For Default Unseal Keys



Terminal

```
$ vault operator init \  
  -key-shares=5 \  
  -key-threshold=3 \  
  -root-token-pgp-key="/opt/bryan.pub"  
  -pgp-keys="/opt/bob.pub,/opt/steve.pub,  
  /opt/stacy.pub,/opt/katie.pub,/opt/dani.pub"
```





Regenerate a Root Token



About Root Tokens



Root token is a superuser that has unlimited access to Vault

- It does NOT have a TTL – meaning it does not expire
- Attached to the root policy
- Note: Root tokens can create other root tokens that DO have a TTL

Root tokens should NOT be used on a day-to-day basis

- In fact, rarely should a root token even exist
- Once you have used the root token, it should be revoked



My First Root Token!



The initial root token comes from Vault cluster initialization

- Only method of authentication when first deploying Vault
- Used for initial configuration – such as auth methods or audit devices
- Once your new auth method is configured and tested, the root token should be revoked

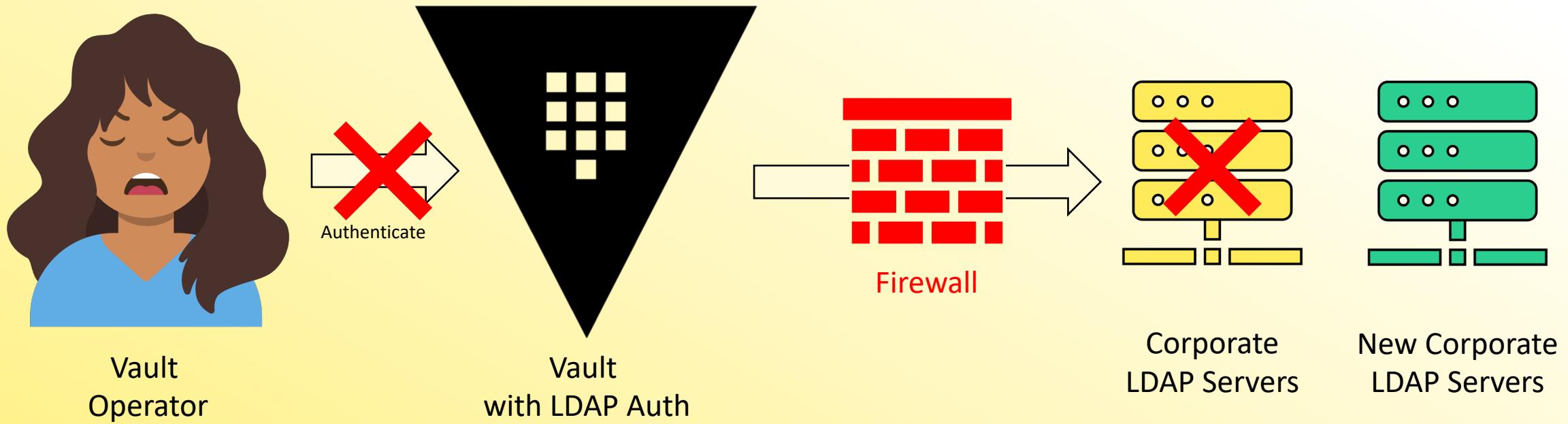


Terminal

```
$ vault token revoke s.dhtIk8VsE3Mj61PuGP3ZffFrg  
Success! Revoked token (if it existed)
```



A Broken Auth Workflow...



What happens if we do not have a working auth method to fix?

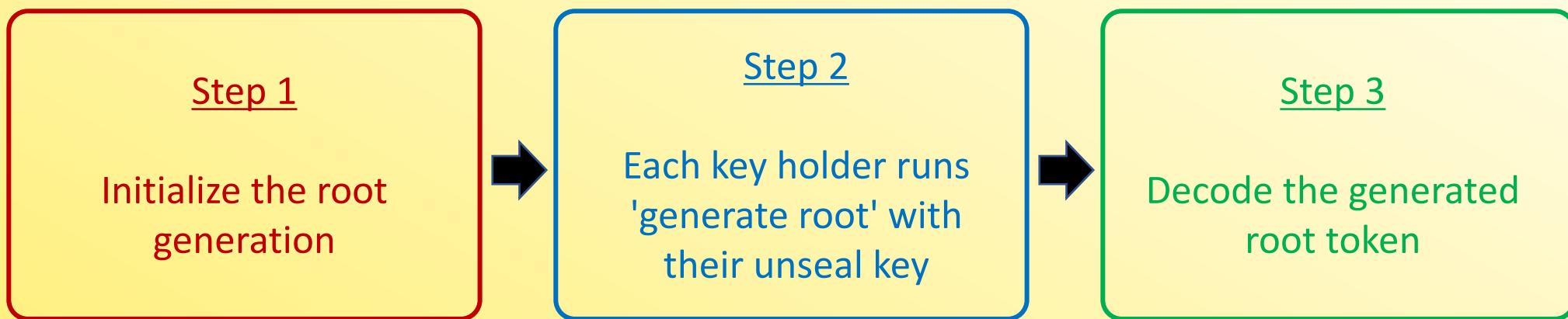


Regenerate a Root Token



Create a root token using unseal/recovery keys

- Helpful if you need to generate a root token in an emergency or a root token is needed for a particular task
- A quorum of key holders can generate a new root token
 - Enforces the "no single person has complete access to Vault"



Vault Initialization



To perform the task, use the `vault operator generate-root` command

Command Options	Description
<code>-generate-otp</code>	Generate and print high-entropy one-time-password
<code>-init</code>	Start a root token generation
<code>-decode=<string></code>	Decode and output the generated root token
<code>-otp=<string></code>	OTP code to use with <code>-decode</code> or <code>-init</code>
<code>-status</code>	Print the status of the current attempt
<code>-cancel</code>	Cancel the current attempt



Generating a Root Token

Step 1 – Initialize the Process



Terminal

```
$ 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	5b6e3831-2a45-4695-7757-5810074d36c8
Started	true
Progress	0/1
Complete	false
OTP	E87jF6ZeJo8NjJwvytl7mvKLEr
OTP Length	26

One-Time-Password (OTP) gets generated



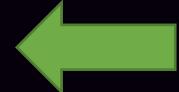
Generating a Root Token

Step 2 – Provide the Keys



Terminal

```
$ vault operator generate-root
Root generation operation nonce: f8579a51-5138-c31...
Unseal Key (will be hidden):
Nonce      f8579a51-5138-c319-445d-2d3640119f87
Started     true
Progress    1/3
Complete   false
```



Key holders each provide their key until you
meet the threshold



Generating a Root Token

Step 3 – Receive Encrypted Token



Terminal

```
$ vault operator generate-root
Root generation operation nonce: f8579a51-5138-c319...
Unseal Key (will be hidden):
Nonce          f8579a51-5138-c319-445d-2d3640119f87
Started        true
Progress       3/3
Complete       true
Encoded Token G2NeKUZgXTsYYxILAC9ZFBguPw9ZXBovFAs
```



Encrypted Root Token



Generating a Root Token

Step 4 – Decode the Newly Generated Root Token



Terminal

```
$ vault operator generate-root \
  -otp="hM9q24nNizfnYIiNvhnGo4UFc3" \
  -decode="G2NeKUZgXTsYYxILAC9ZFBguPw9ZXBoVFA\$"
```

Root token: hvs.gXtT3uq9teYf0ZnFQH6hOiw8 ←

We Got A Root Token!!!





Rekey Vault and Rotate Encryption Keys



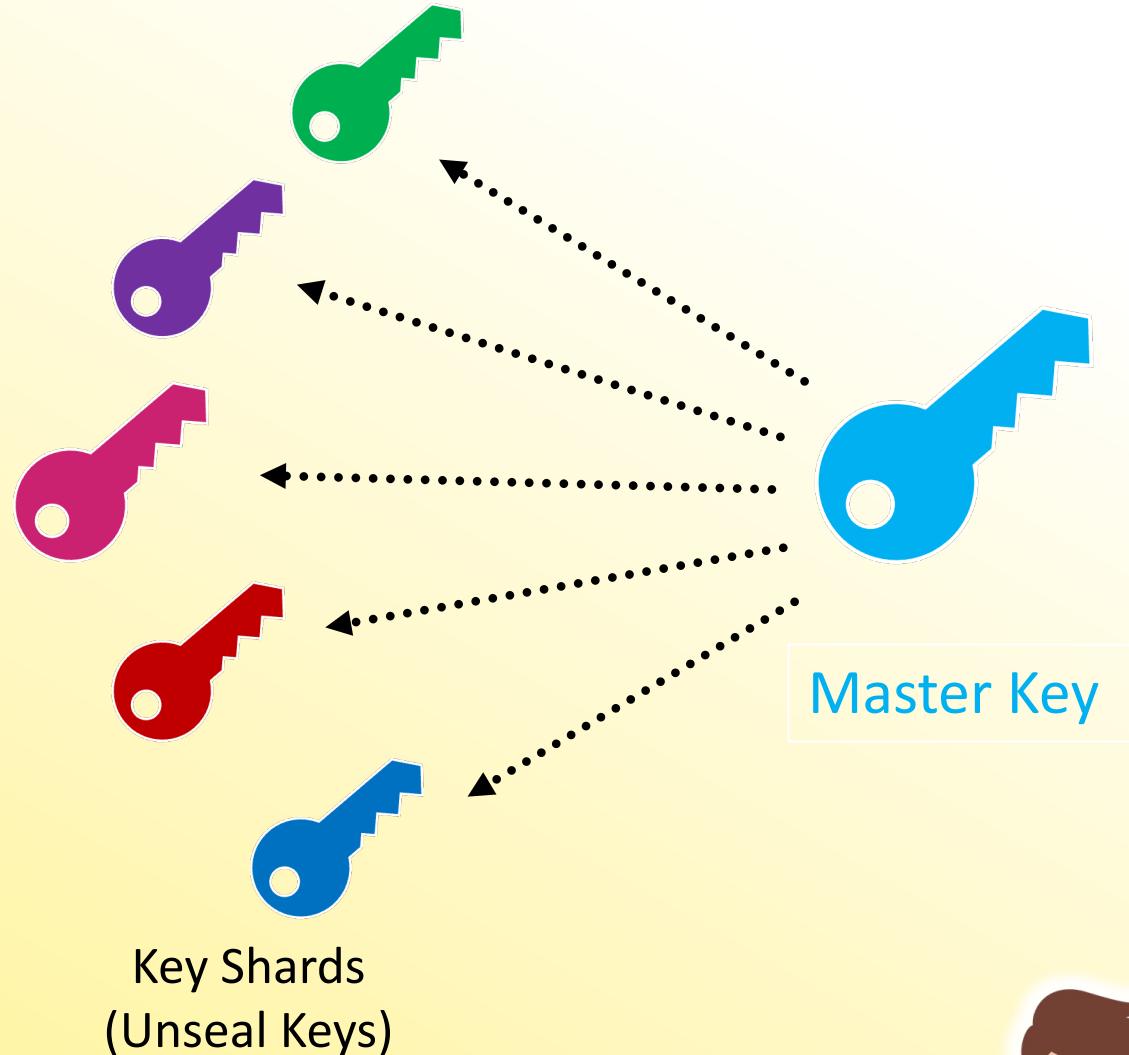
What is Rekey?

- Creates a new set of Vault recovery/unseal keys
- Allows you to specify the number of keys and threshold during the rekey process
- Requires a threshold of keys to successfully rekey (similar to an unseal or root token generation)
- Provides a nonce value to be given to key holders



Why Would I Need to Rekey?

- One or more keys are lost
- Employees leave the organization
- Your organization requires that you rekey after a period of time as a security measurement



Rekey



To rekey Vault, use the `vault operator rekey` command:

Command Options	Description
<code>-init</code>	Initialize the rekey process
<code>-key-shares=<num></code>	Specify the number of key shares
<code>-key-threshold=<num></code>	Specify the threshold of keys needed to reconstruct the master key
<code>-nonce</code>	Pass the nonce value
<code>-pgp-key=<keys></code>	Specify the PGP keys to encrypt the generated keys
<code>-status</code>	Print the status of the current rekey operation
<code>-target=recovery</code>	Specify that you want recovery keys if using HSM or Auto Unseal



Rekeying Vault

Step 1 – Initialize the Process



Terminal

```
$ vault operator rekey -init -target=recovery
```

WARNING! If you lose the keys after they are returned, there is no recovery. Consider canceling this operation and re-initializing with the -pgp-keys flag to protect the returned unseal keys along with -backup to allow recovery of the encrypted keys in case of emergency. You can delete the stored keys later using the -delete flag.

Key	Value
--	---
Nonce	6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c
Started	true
Rekey Progress	0/3
New Shares	5
New Threshold	3
Verification Required	false

Nonce gets generated



Rekeying Vault

Step 2 – Provide the Existing Keys



Terminal

```
$ vault operator rekey -target=recovery
Rekey operation nonce: 6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c
Unseal Key (will be hidden):
Key          Value
---          -----
Nonce        6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c
Started      true
Rekey Progress 1/3
New Shares    5
New Threshold 3
Verification Required false
```

Key holders each provide their key until you
meet the threshold



Rekeying Vault

Step 2 – Provide the Existing Keys



Terminal

```
$ vault operator rekey -target=recovery
Rekey operation nonce: 6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c
Unseal Key (will be hidden):
Key          Value
---          -----
Nonce        6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c
Started      true
Rekey Progress 2/3
New Shares   5
New Threshold 3
Verification Required false
```

Key holders each provide their key until you
meet the threshold



Rekeying Vault

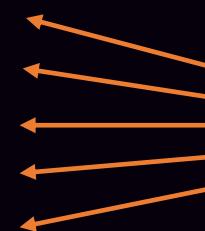
Receive New Recovery Keys



Terminal

```
$ vault operator rekey -target=recovery  
Rekey operation nonce: 6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c  
Unseal Key (will be hidden):
```

```
Key 1: DwCpPnsbvUMqBtXJcAewCHgYr4b+5C56036mWDpx7d7r  
Key 2: roNCdt dtoK+Z7crwZvrPYsraXZm7ZkIzj7lwm6gq8LkP  
Key 3: 5BYFqW/Pt1TXtFmzXft10XwqIt6v/gQjWF8srMbx7Luo  
Key 4: eD6gkKkcdM5TsmnSSk5k0ogI5KksdH2GZvguyBFungPS  
Key 5: HtFsHfcYvSICFEetguouhqr4K9ehXAoJm8ktxdT0EJl
```



New Recovery Keys

```
Operation nonce: 6e2fb7b0-b9f6-12a8-d94c-a36a7b26c67c
```

Vault rekeyed with 5 key shares and a key threshold of 3. Please securely distribute the key shares printed above. When 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.



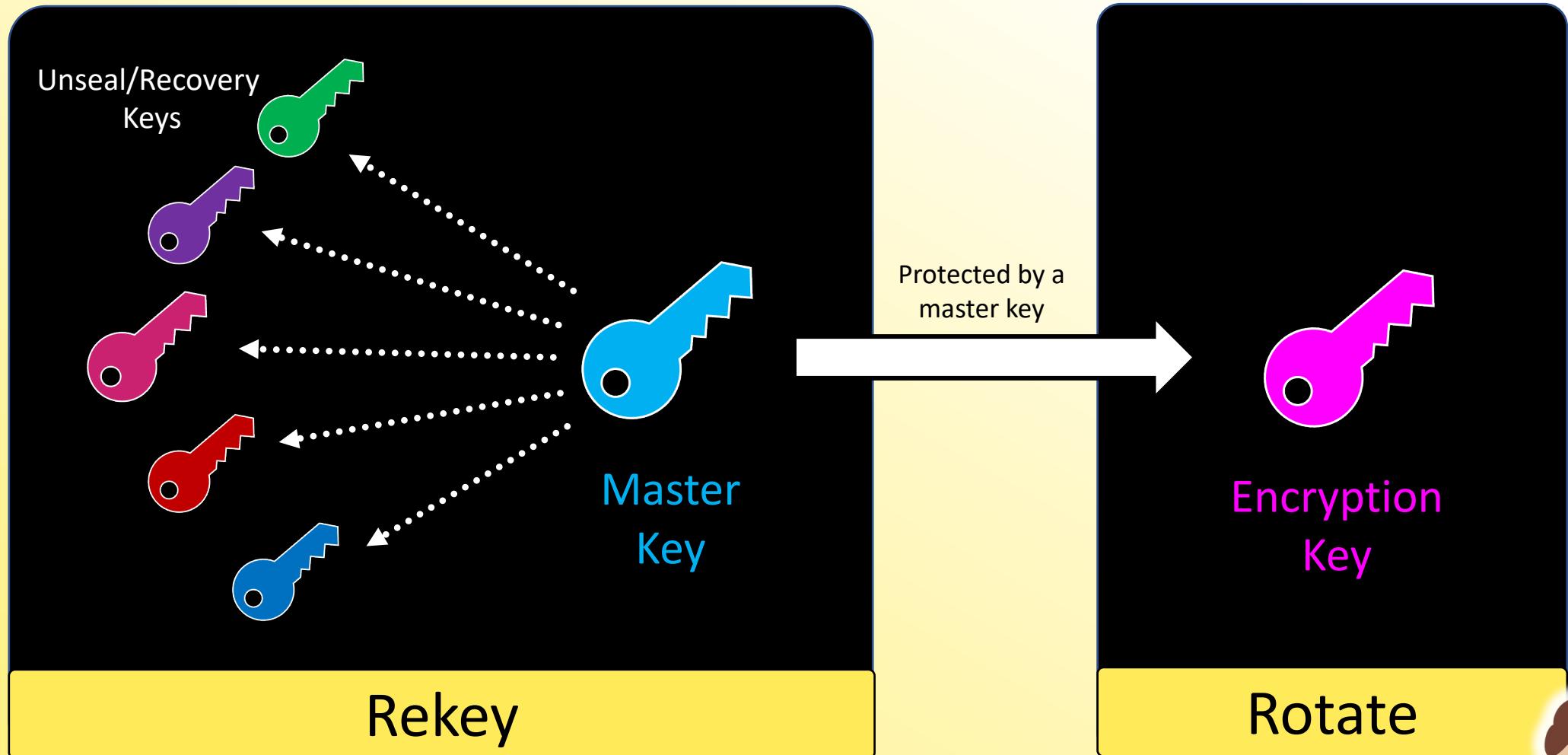
Impact to Production?



- The Rekey operation is an online operation and can be done anytime
- This means no downtime for performing this operation
- Vault continues to service requests during the rekey operation

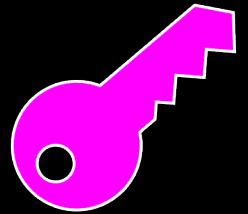


Rekey vs. Key Rotation



What is Key Rotation?

- Rotates the encryption key that is used to protect data on the storage backend
- Since the encryption key is never available to users or operators, it does NOT require a threshold of key holders to rotate
- New keys are added to the keyring - old values can still be decrypted with the old key



Encryption
Key

Rotate



Rotating the Encryption Key



Terminal

```
$ vault operator rotate  
Success! Rotated key
```

Key Term	2
Install Time	25 Dec 22 15:47 UTC
Encryption Count	6



Permissions Needed for Key Rotation

- The `sys/rotate` endpoint is a root protected API endpoint
 - This means that you either need a **root token** or **sudo** privileges on the `sys/rotate` path

Terminal

```
path "sys/rotate" {  
    capabilities = ["update", "sudo"]  
}  
  
path "sys/key-status" {  
    capabilities = ["read"]  
}
```

Needed to read the status
of the key after rotating it,
not to rotate it

