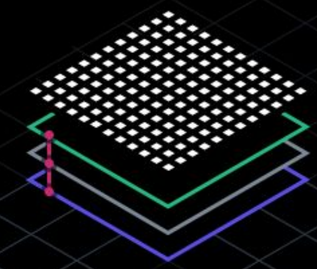
HashiCorp

DEVOPS DELIVERED



HashiCorp

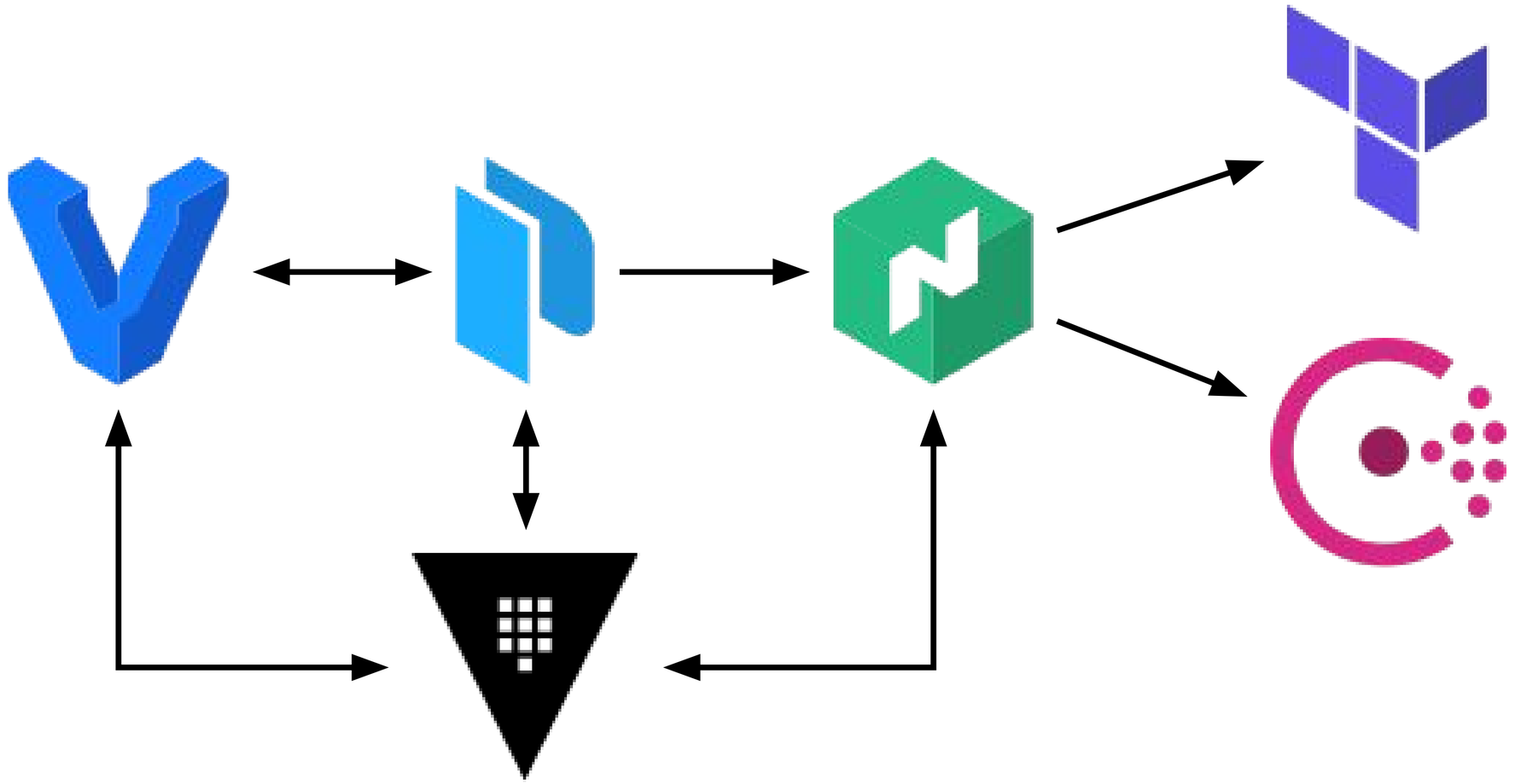
Vault

Agenda

1. Glossary and Architecture
2. Static and Generic Secrets
3. Policies and Policy Workflow
4. Dynamic Secrets
5. Authentication, Auditing, and Lease Model
6. Operationalizing Vault
7. HTTP API
8. Direct Application Integration

HashiCorp

Vault Manages Secure Information



HashiCorp

Pre-Vault World

Secret sprawl

Decentralized keys

Limited visibility

Poorly-defined "break-glass" procedures

Post-Vault World

Single source for secrets

Programatic access (automation)

Operation access (manual)

Practical security

Modern data-center friendly (no hardware reqs.)

Glossary

Glossary

Storage backend

The storage backend is responsible for durable storage of encrypted data. There is only one storage backend per Vault cluster.

Data is encrypted in transit and at rest with 256bit AES.

Examples: *in-mem*, *file*, *consul*, and *postgresql*

Glossary

Barrier

The barrier is a cryptographic seal around the Vault. All data that flows between Vault and the storage backend passes through the barrier.

Glossary

Secret backend

A secret backend is responsible for managing secrets. Some secret backends behave like encrypted key-value stores, while others dynamically generate secrets when queried. There can be multiple secret backends in a Vault cluster.

Examples: *pki, generic, transit, postgresql*

Glossary

Secret backend

Secret backends can perform almost any function, not just return static data or hand out credentials.

PKI – Acts as a full CA, leveraging Vault's auth

Transit – Allows round-tripping data through Vault for "encryption as a service", without ever divulging the key

Glossary

Audit backend

An audit backend is responsible for managing audit logs. There can be multiple audit backends in a Vault cluster. Example audit logs include *file* and *syslog*.

Glossary

Auth backend

An auth backend is a credential-based backend that can be used as a way to authenticate humans or machines against Vault.

Machine-oriented: *approle*, *tls*, *tokens*
Operator-oriented: *github*, *ldap*, *userpass*

Glossary

Vault token

A vault token is a conceptually similar to a session cookie on a website. Once a user authenticates via an auth backend, Vault returns a token which is to be used for future requests.

Example: dc57a797-fc99-05d1-6878-f731206b1717

Glossary

Secret

A secret is anything stored or returned by Vault that contains confidential material.

A secret is anything that, if acquired by an unauthorized party, would cause political, financial, or appearance harm to an organization.

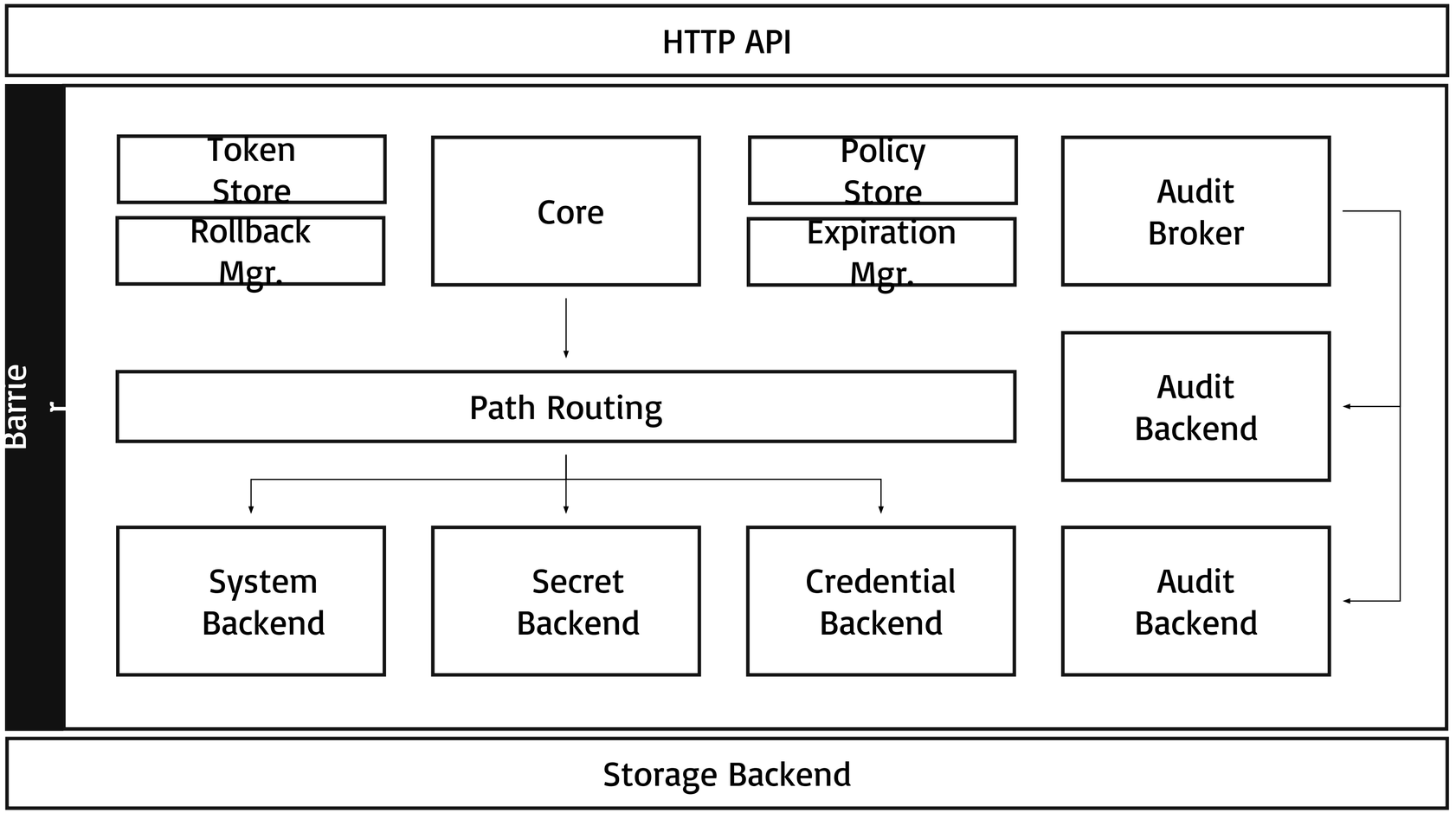
Glossary

Server

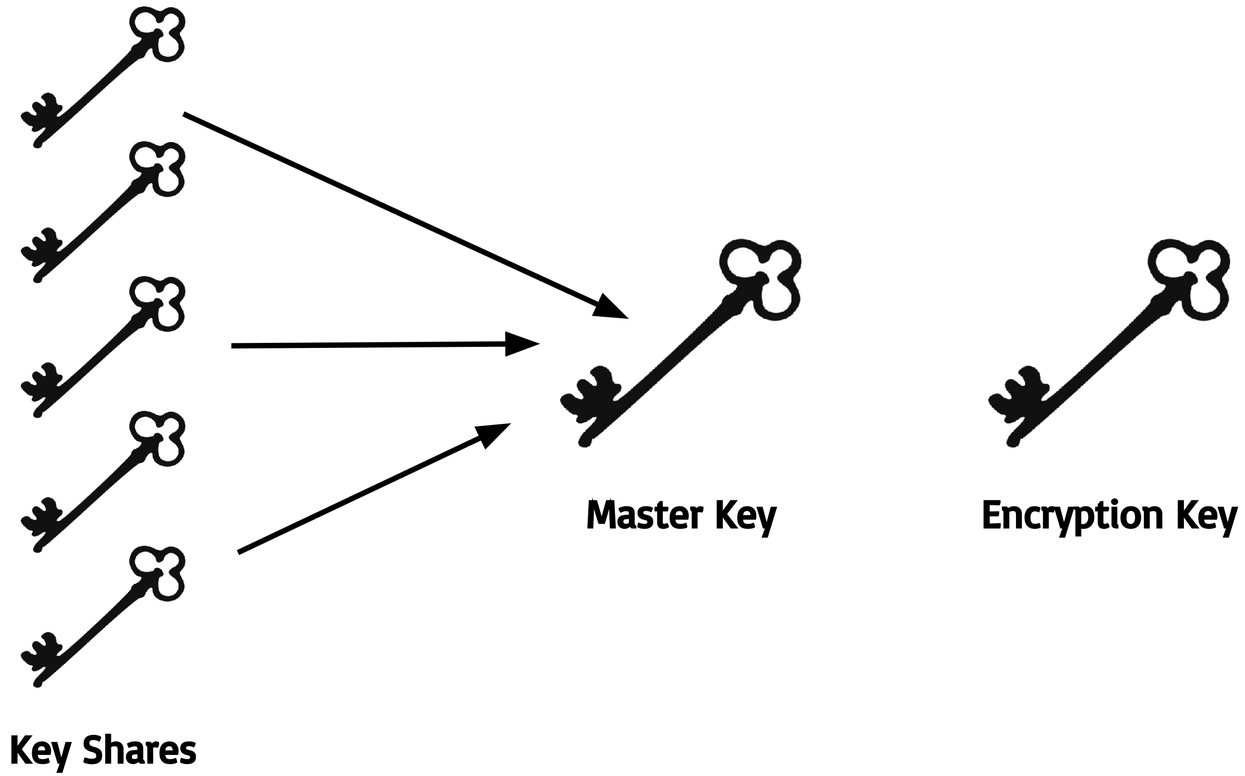
The Vault server provides an HTTP API which clients interact with and manages the interaction between all the backends, ACL enforcement, and secret lease revocation.

Architecture

Architecture



Shamir’s secret sharing



Summary

Solves the "secret sprawl" problem

Protects against external threats (cryptosystem)

Protects against internal threats (ACLs and secret sharing)

Using Generic Secrets

Exercise: Launch Training Environment

Install Prerequisites:

- VirtualBox https://www.virtualbox.org/wiki/Downloads
- Vagrant https://www.vagrantup.com/downloads.html
- Git https://git-scm.com/downloads

Clone vault repository:

```
git clone https://github.com/chrismatteson/vault-101
```

Launch vagrant environment:

```
cd vault-101
vagrant up
```

HashiCorp

Exercise: Launch Training Environment

View Consul UI for Health Checks:

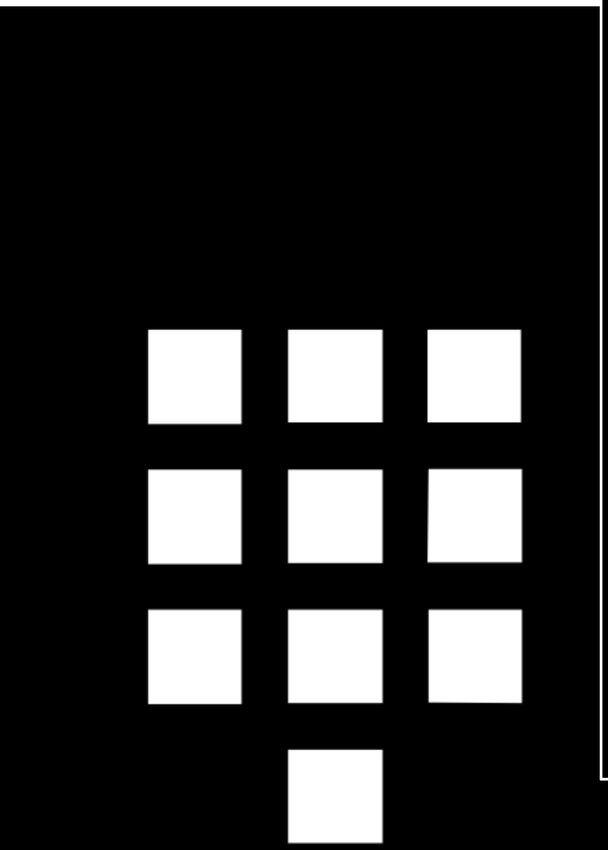
https://192.168.50.151:8500

View Vault UI:

```
https://192.168.50.151:8200
```

Connect to Vault:

```
export VAULT_ADDR=http://192.168.50.151:8200
vault status
```



Terminal

```
$ vault status
Error checking seal status: Error making API request.

URL: GET http://192.168.50.151:8200/v1/sys/seal-status
Code: 400. Errors:

* server is not yet initialized
```

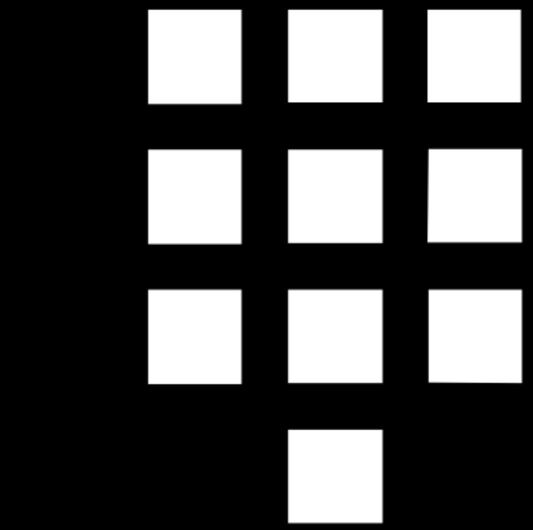
HashiCorp

Exercise: Initialize Vault

Initialize Vault:

```
vault init
```

HashiCorp



Terminal

```
$ vault init
Unseal Key 1: gELVCpVZiVxlq87lI2++Fo/jMzcSzC8pQGMdrMrZWhr9
Unseal Key 2: 39qtnuSLB6UMYrSXNktJqpxXF38TvmjpokBSzPyb338N
Unseal Key 3: r6qaD/4R2fz+WcaYlG7URfaVNTxOzuoKgo2XoyKder7H
Unseal Key 4: gSrtsxFRM5hz5+yH/6Oumw+zjSLqgrRev24yWe3z2NQ+
Unseal Key 5: Rrt8tUam2CrHnjYI3QMIpMN1HizSxN1COq7yvStVpta6
Initial Root Token: 3803f26e-0064-00a1-c0a2-c4e49a37a43c

Vault initialized with 5 keys and a key threshold of 3. Please
securely distribute the above keys. When the vault is re-sealed,
restarted, or stopped, you must provide at least 3 of these keys
to unseal it again.

Vault does not store the master key. Without at least 3 keys,
your vault will remain permanently sealed.
```

HashiCorp

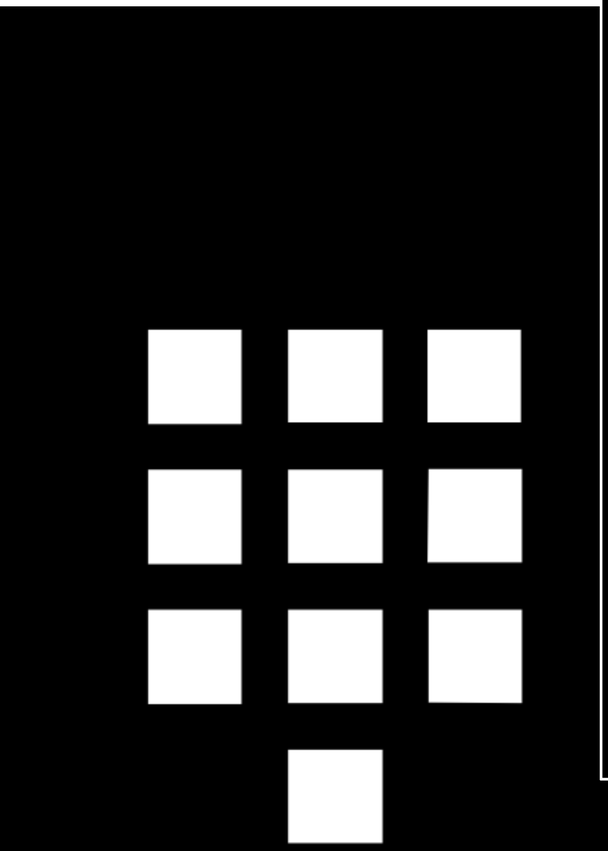
Exercise: Unseal Vault

Unseal:

```
vault unseal
<enter one key>
vault unseal
<enter different key>
vault unseal
<enter different key>
```

View Status:

```
vault status
```



Terminal

```
$ vault status
Type: shamir
Sealed: false
Key Shares: 5
Key Threshold: 3
Unseal Progress: 0
Unseal Nonce:
Version: 0.9.0.1+ent
Cluster Name: vault-cluster-5723c776
Cluster ID: 9b6227c6-77aa-c61d-16f6-8d294e91d050

High-Availability Enabled: true
  Mode: active
  Leader Cluster Address: https://192.168.50.151:8201
```

HashiCorp

Generic Secret Backend

The **generic** secret backend is mounted by default and cannot be disabled.

Behaves like encrypted redis or memcached.

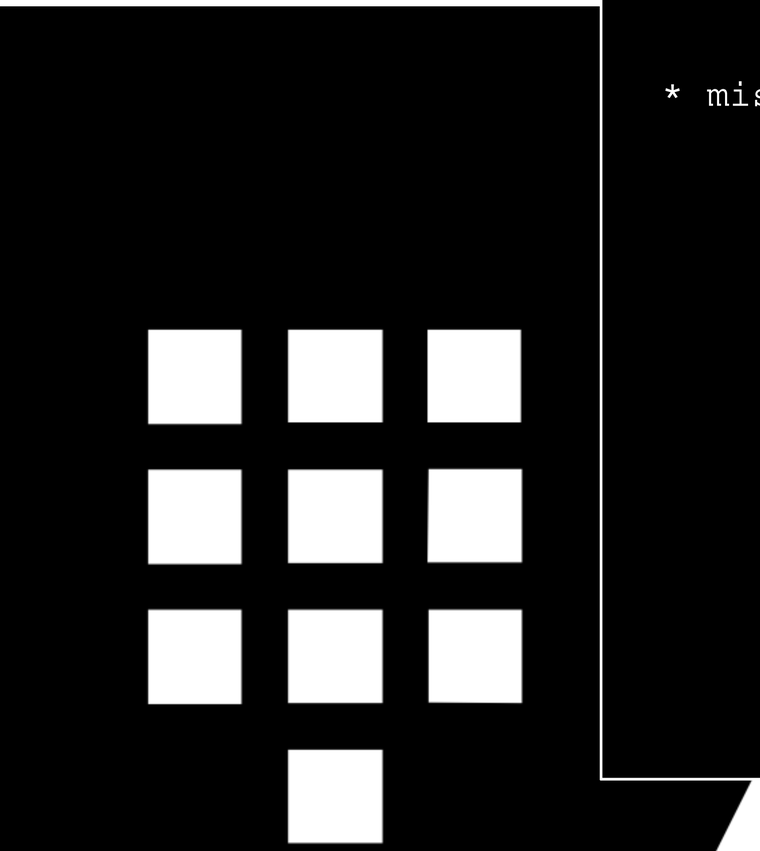
Lives at the `secret/` endpoint.

Exercise: Read Generic Secret

Attempt to read the secret at `secret/training`.

HINT: You can use Vault's help documentation

ANOTHER HINT: You'll get an error



Terminal

```
$ vault read secret/training
Error reading secret/training: Error making API request.

URL: GET http://192.168.50.151:8200/v1/secret/training
Code: 400. Errors:

* missing client token
```

HashiCorp

Authentication

Most interactions with Vault require a token.

Tokens are generated via authentication.

Authentication is covered in more detail in a later section.

Information is persisted by the local client (you do not need to re-authenticate before each command).

Authenticating as Root

Authenticating as the root user is bad practice.

For the purpose of training, we will start slightly insecure and move to a more secure workflow.

The root token is usually used to setup policy and initial set of users, but then is discarded.

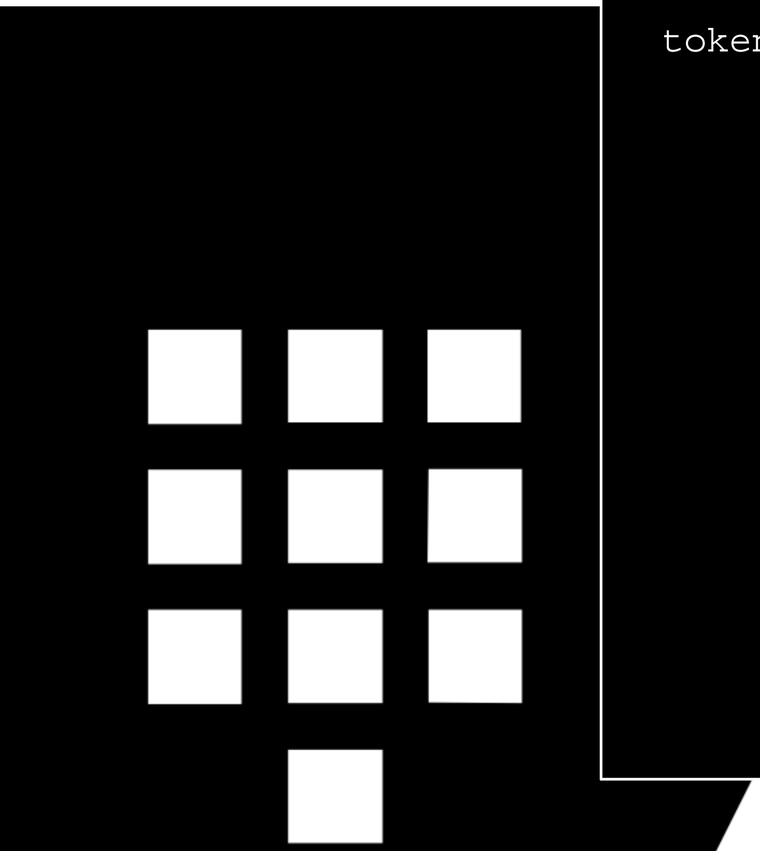
Authenticate as root to continue.

Exercise: Authenticate Vault

Authenticate:

```
vault auth
<enter root authentication token>
```

HashiCorp



Terminal

```
$ vault auth
Token (will be hidden):
Successfully authenticated! You are now logged in.
token: 3803f26e-0064-00a1-c0a2-c4e49a37a43c
token_duration: 0
token_policies: [root]
```

HashiCorp

Vault Enterprise Licensing

Vault Enterprise uses a special binary and license file which is written to sys/license. If a license is not installed, vault will stop operating after 30 minutes. **If that happens** before the license is installed, connect into the vm and restart the service:

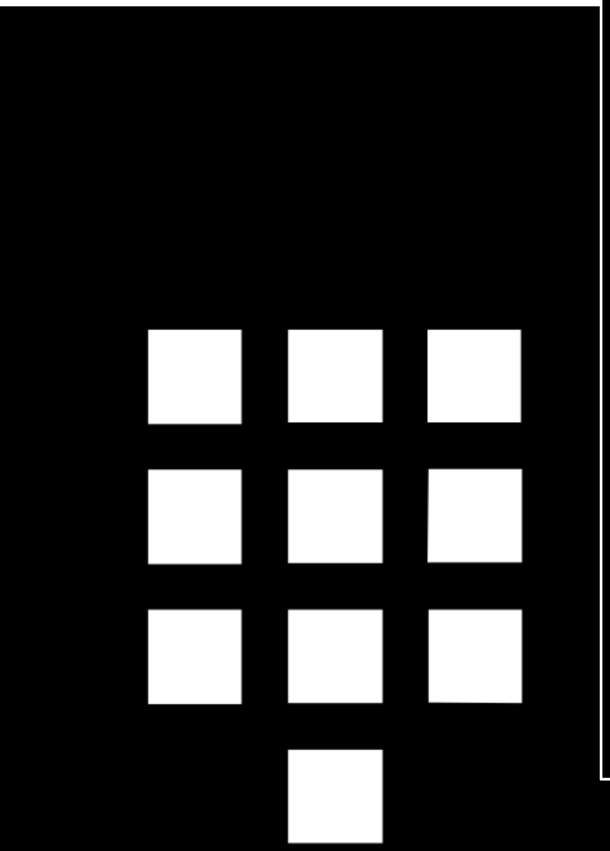
```
vagrant ssh vault1
sudo service vault restart
exit
```

Exercise: Install License

Authenticate:

```
vault write sys/license text=`cat <license file>`
```

HashiCorp



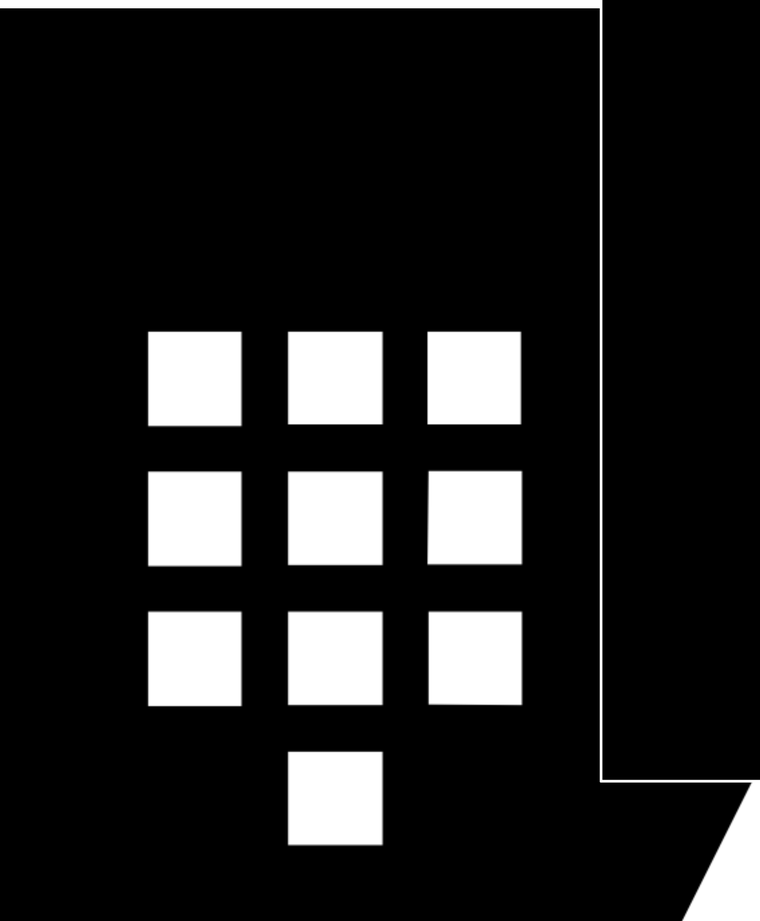
Terminal

```
$ vault write sys/license text=`cat
57b3e2a3-e85f-7a6b-ae95-639fa81269a2.hclic`
Success! Data written to: sys/license
$ vault read sys/license
Key            Value
---            -----
expiration_time2018-12-19T07:59:59.999Z
features       [UI HSM Performance Replication DR Replication MFA Sentinel AWS
KMS Autounseal GCP CKMS Autounseal Seal Wrapping Control Groups]
license_id     57b3e2a3-e85f-7a6b-ae95-639fa81269a2
start_time     2018-01-10T08:00:00Z
```

HashiCorp

Exercise: Read Generic Secret (again)

Now that we have authenticated, attempt to read the secret at `secret/training` again.



Terminal

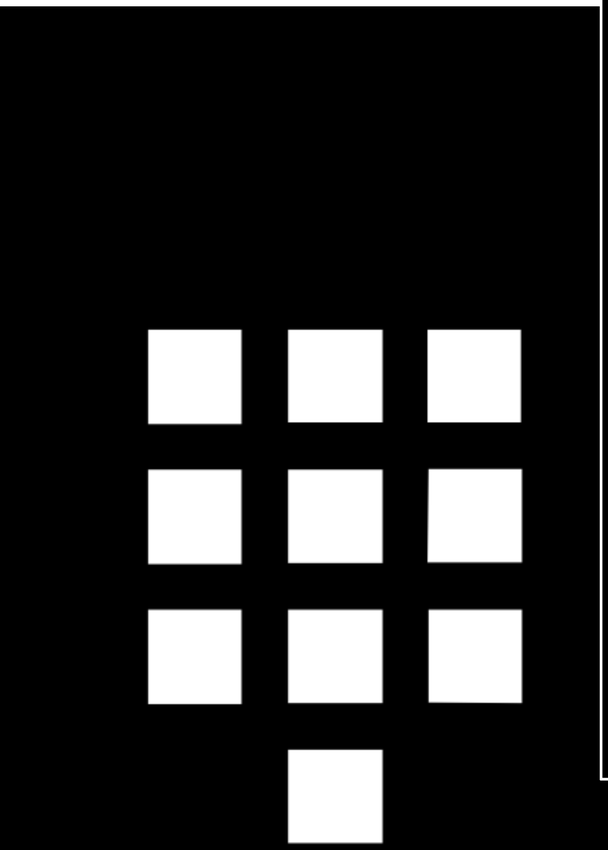
```
$ vault read secret/training
No value found at secret/training
```

HashiCorp

Exercise: Write Generic Secret

Write a value into `secret/training`.

HINT: Data is expressed as key=value pairs on the CLI



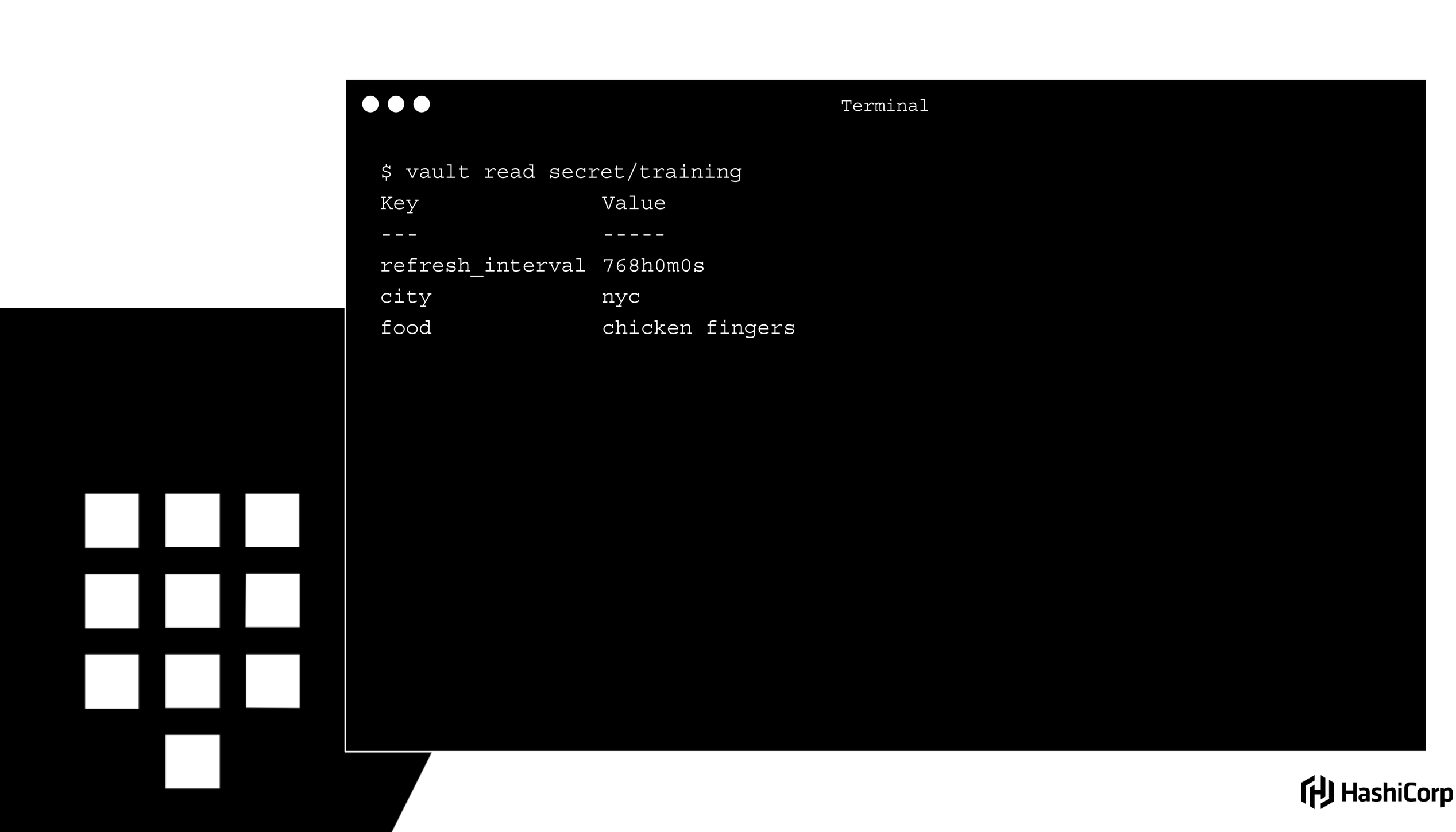
Terminal

```
$ vault write secret/training city=nyc food="chicken fingers"
Success! Data written to: secret/training
```

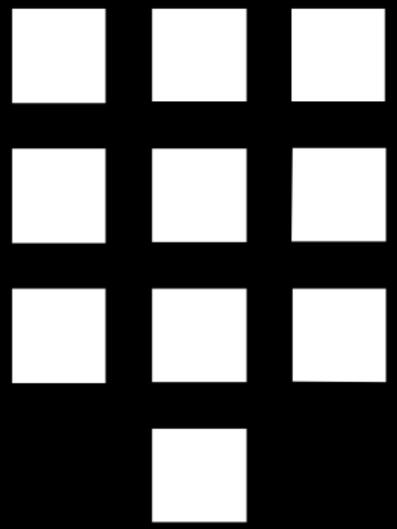
HashiCorp

Exercise: Retrieve Secret

Read the value of the secret you just stored in `secret/training`.


Terminal
$ vault read secret/training
Key Value
--- -----
refresh_interval 768h0m0s
city nyc
food chicken fingers

HashiCorp

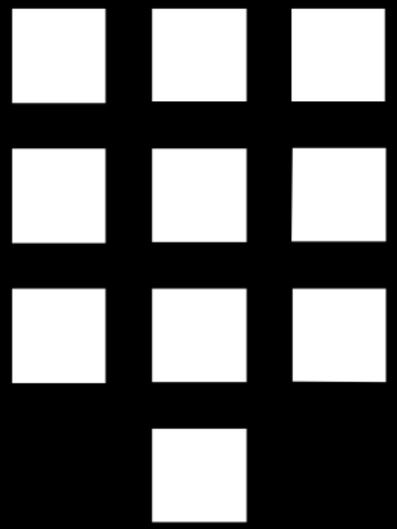


Terminal

```
$ vault read -field=city secret/training
nyc%
```

HashiCorp

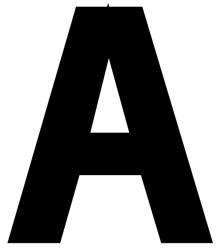
Q

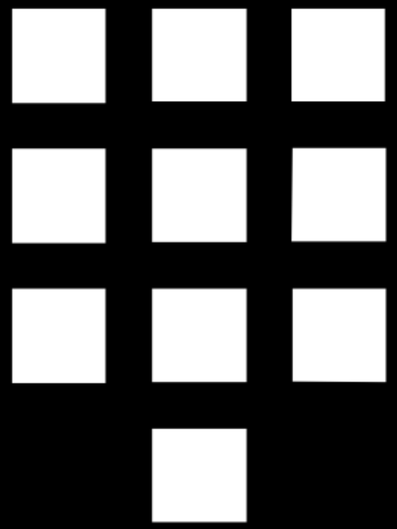


Terminal

```
$ vault write secret/training food=pizza
Success! Data written to: secret/training
```

HashiCorp

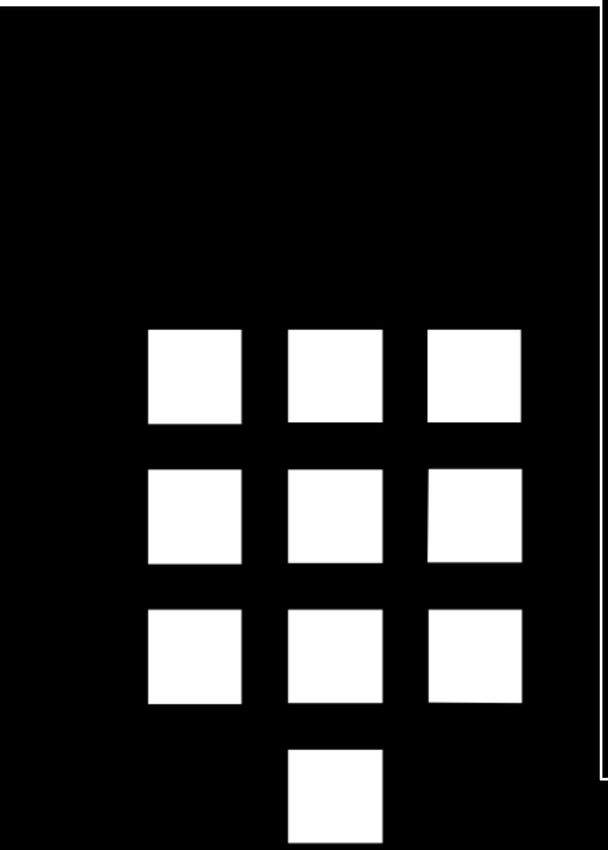




Terminal

```
$ vault read secret/training
Key                     Value
---                     -----
refresh_interval        720h0m0s
food                    pizza
```

HashiCorp



Terminal

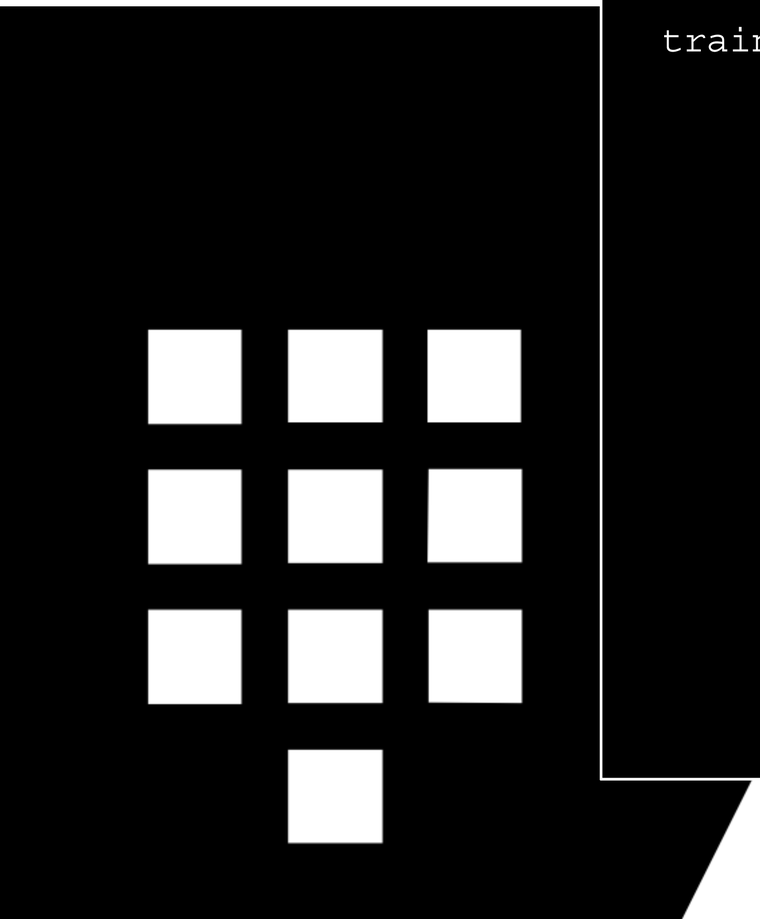
```
$ vault write secret/foo a=b

# You can also read values from a file using the "@" symbol.
$ vault write secret/foo a=@file.txt
```

HashiCorp

Exercise: List Secrets

List all the secret keys stored in the generic secret backend.

HINT: Just the keys, not the values.



Terminal

```
$ vault list secret
Keys
----
bar
foo
training
```

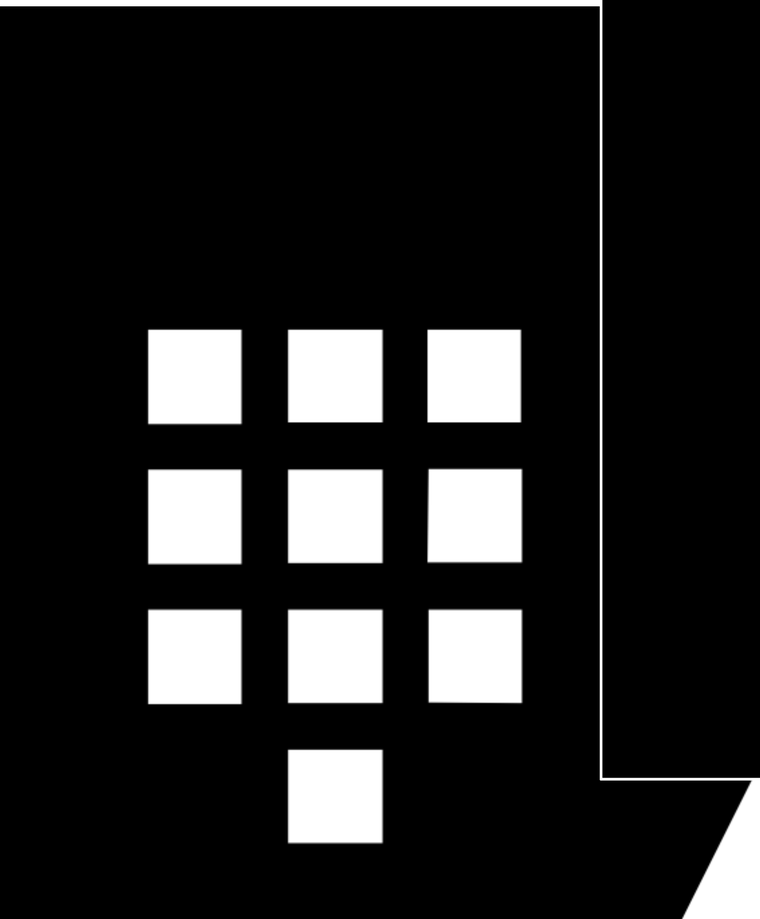
HashiCorp

Exercise: Delete Secret

Delete one of the secrets you just created.

Do **NOT** delete the training key.

HashiCorp



Terminal

```
$ vault delete secret/foo
Success! Deleted 'secret/foo' if it existed.
```

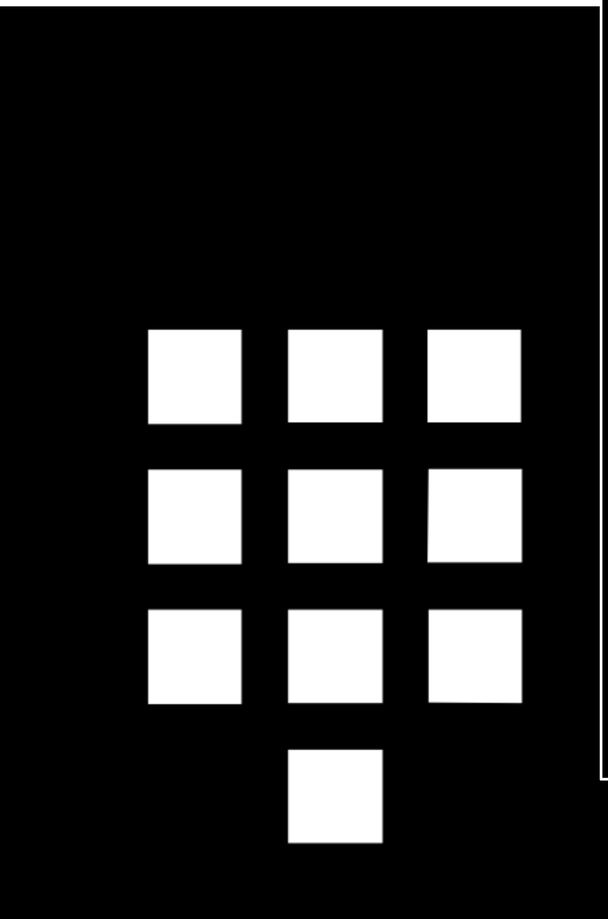
HashiCorp

Getting Help

Getting Help

There are two primary ways to get help in Vault:

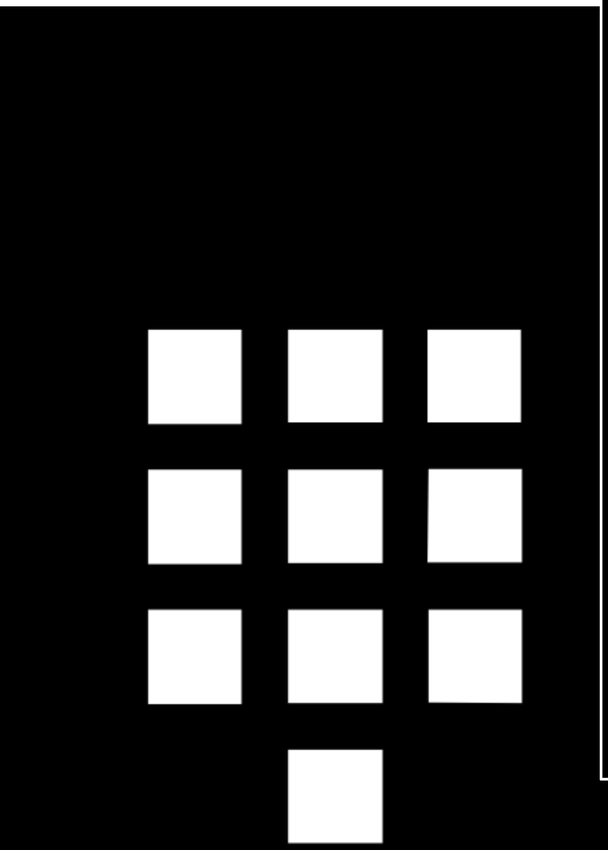
- CLI help (vault -h)
- API help (vault path-help)



Terminal

```
$ vault help # CLI help (aka "-h")

$ vault path-help # API help
```

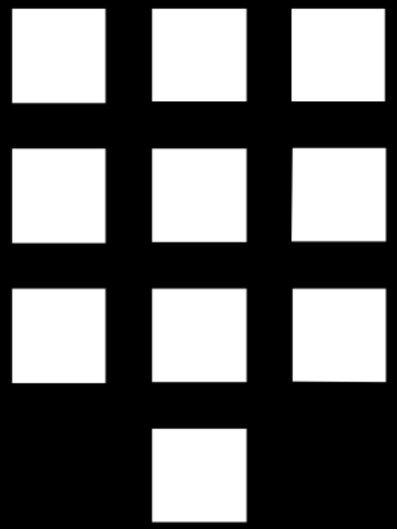
HashiCorp



Terminal

```
$ vault read -h
Usage: vault read [options] path

  Read data from Vault.

  Reads data at the given path from Vault. This can be used to read
  secrets and configuration as well as generate dynamic values from
  materialized backends. Please reference the documentation for the
  backends in use to determine key structure.
```

HashiCorp



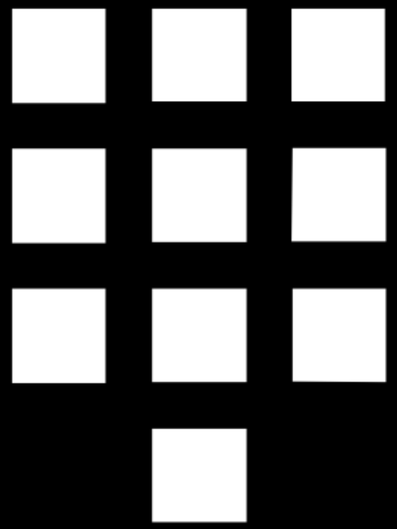
Terminal

```
$ vault path-help secret/
## DESCRIPTION

The generic backend reads and writes arbitrary secrets to the backend.

...

## PATHS

The following paths are supported by this backend. To view help for
any of the paths below, use the help command with any route matching
the path pattern. Note that depending on the policy of your auth token,
you may or may not be able to access certain paths.

    ^.*$
        Pass-through secret storage to the storage backend, allowing you to
        read/write arbitrary data into secret storage.
```

HashiCorp

Exercise: Using Help

List help information for the HTTP API cubbyhole backend

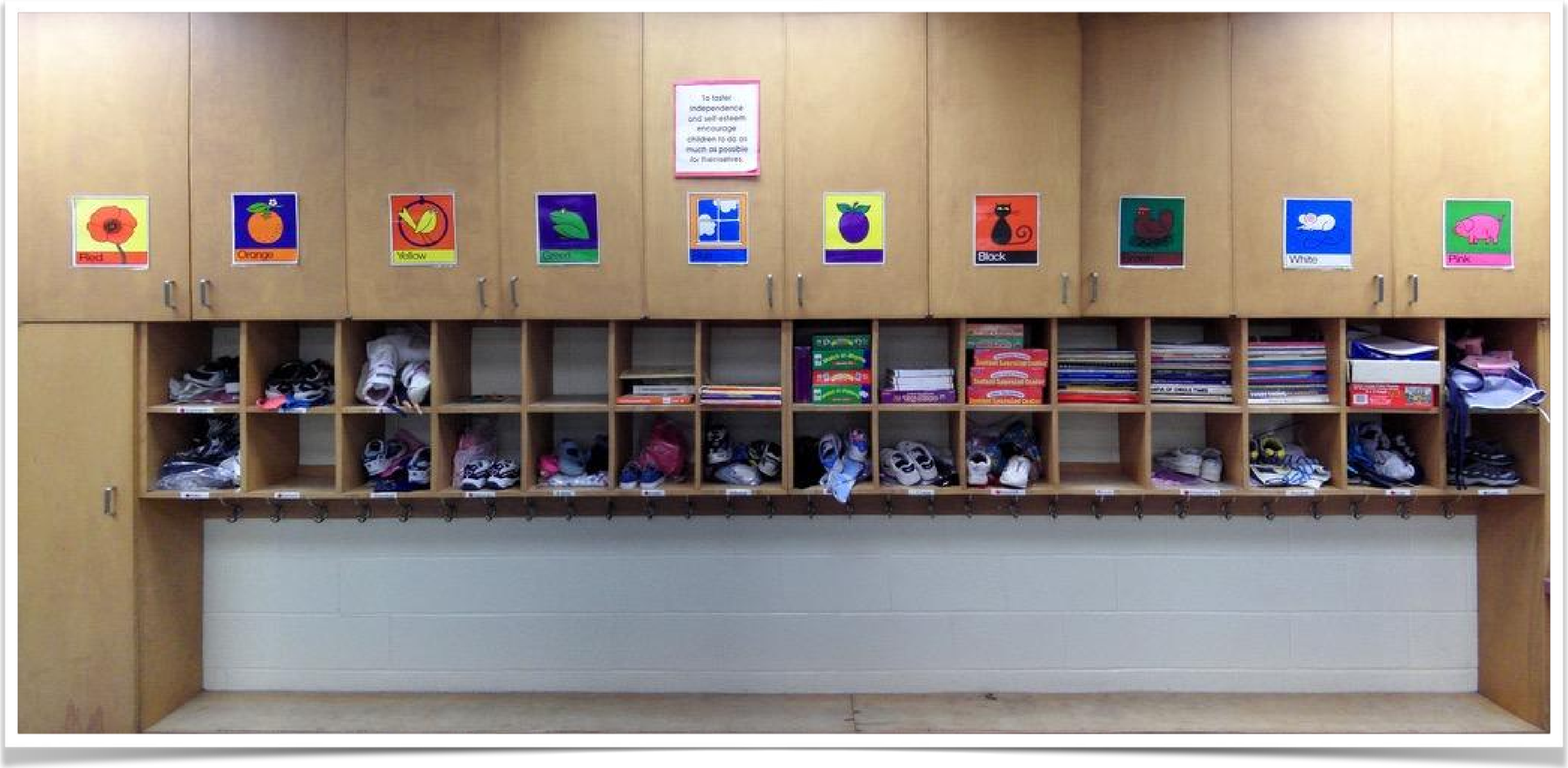


Terminal

```
$ vault path-help cubbyhole/
## DESCRIPTION

The cubbyhole backend reads and writes arbitrary secrets to the backend.
The secrets are encrypted/decrypted by Vault: they are never stored
unencrypted in the backend and the backend never has an opportunity to
see the unencrypted value.

This backend differs from the 'generic' backend in that it is namespaced
per-token. Tokens can only read and write their own values, with no
sharing possible (per-token cubbyholes). This can be useful for implementing
certain authentication workflows, as well as "scratch" areas for individual
clients. When the token is revoked, the entire set of stored values for that
token is also removed.

## PATHS

The following paths are supported by this backend. To view help for
```

HashiCorp

About: Cubbyhole



HashiCorp

Setting Policy

HashiCorp

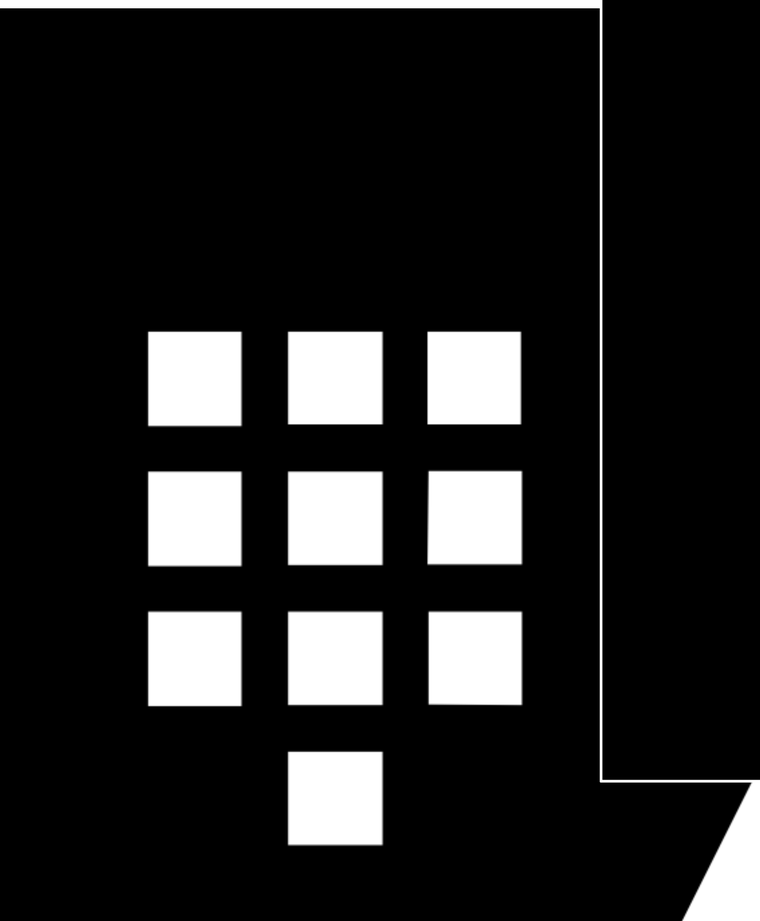
Access Control Policies (ACLs)

"root" policy is created by default – superuser with all permissions.

"default" policy is created by default - common permissions.

Policies are written in HashiCorp Configuration Language (HCL), which is a human-friendly config format.

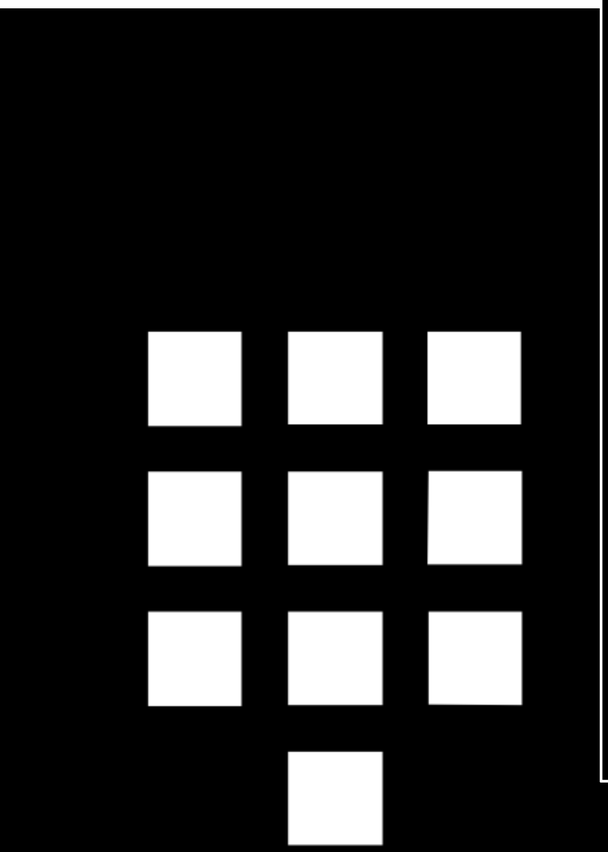
Deny by default (no policy = no authorization).



Terminal

```
$ vault policies
```

HashiCorp



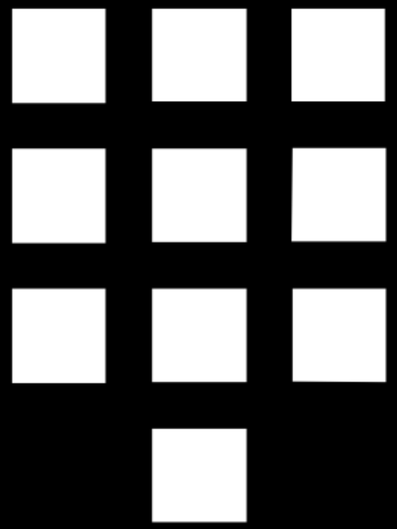
Terminal

```
$ vault policies
default
root
```

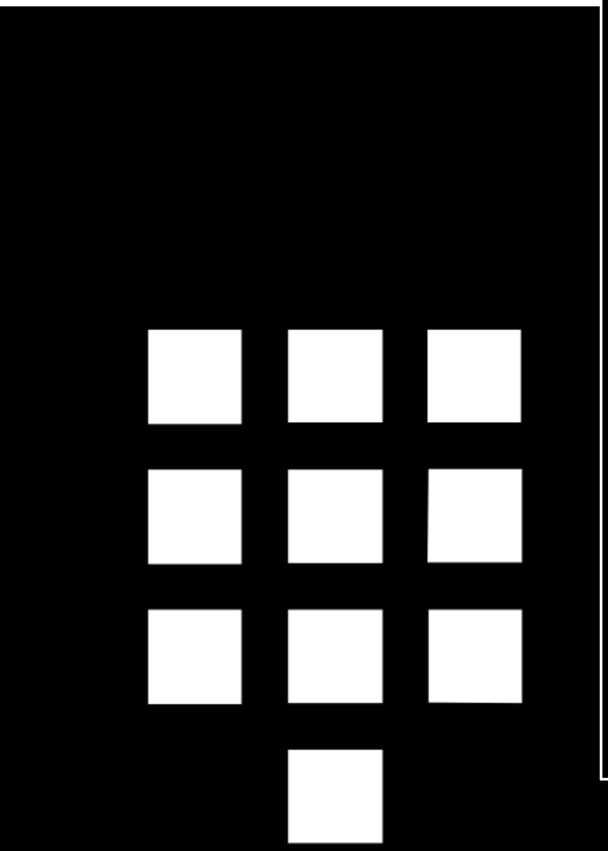
HashiCorp

base.hcl

```
path "secret/training_*" {
  capabilities = ["create", "read"]
}
```



HashiCorp

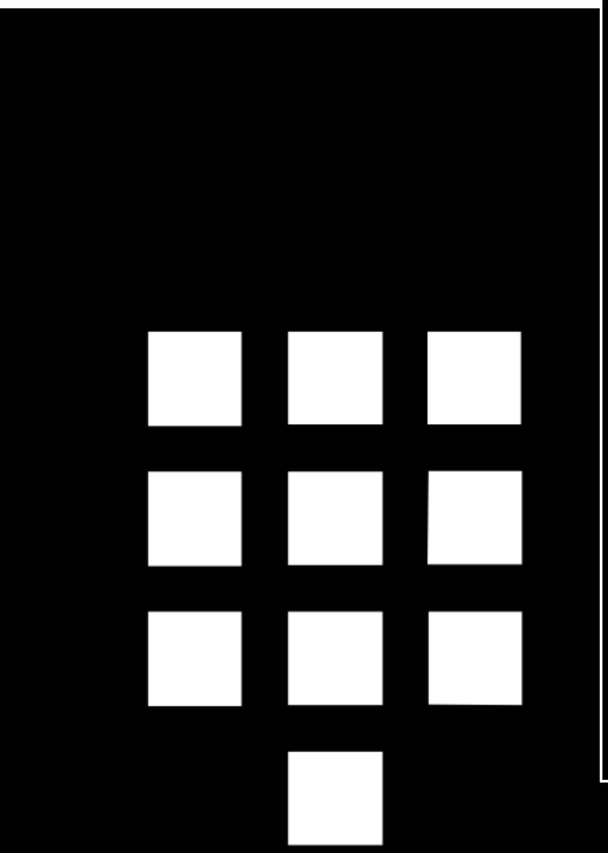


Terminal

```
$ vault policy-write base ./base.hcl
```

```
$ vault write sys/policy/base rules=@base.hcl
```

HashiCorp

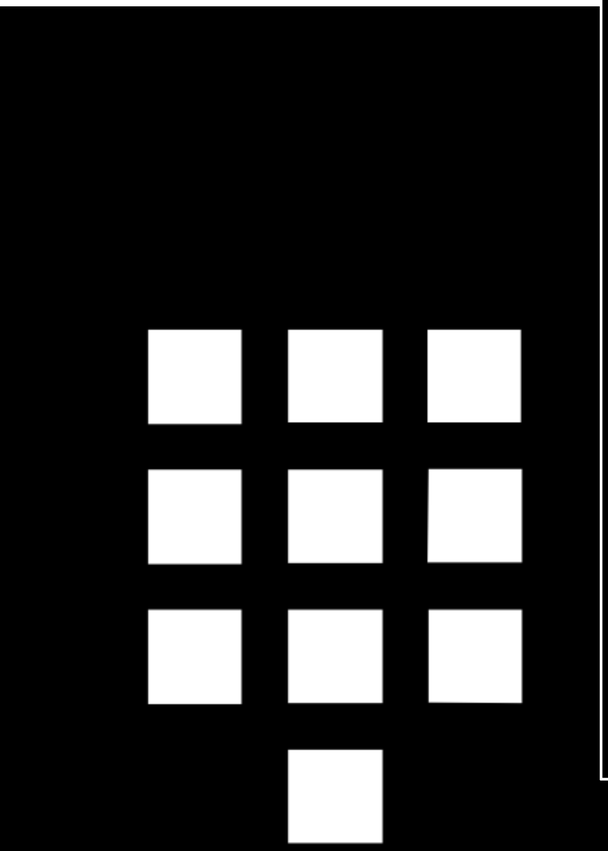


Terminal

```
$ vault policy-write base ./base.hcl
Policy 'base' written.
```

```
$ vault write sys/policy/base rules=@base.hcl
Success! Data written to: sys/policy/base
```

HashiCorp

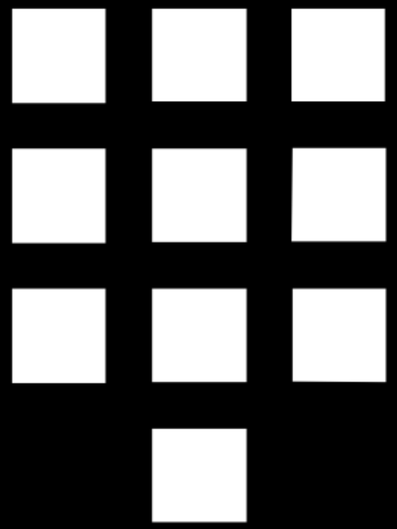


Terminal

```
$ vault policies
base
default
root
```

```
$ vault read sys/policy
Key      Value
---      -----
keys     [base default root]
policies [base default root]
```

HashiCorp

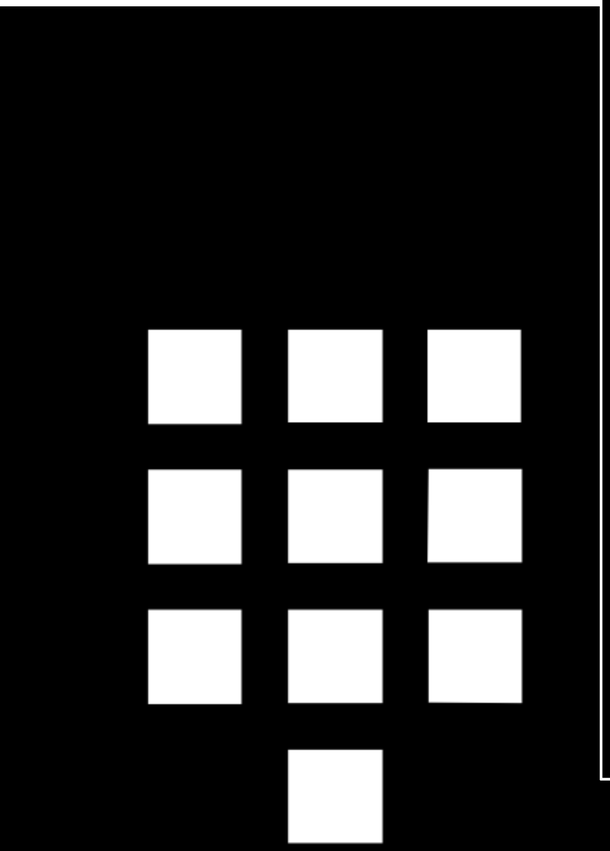


Terminal

```
$ vault policies base
path "secret/training_*" {
  capabilities = ["create", "read"]
}
```

```
$ vault read sys/policy/base
Key    Value
---    -----
name   base
rules  path "secret/training_*" {
  capabilities = ["create", "read"]
}
```

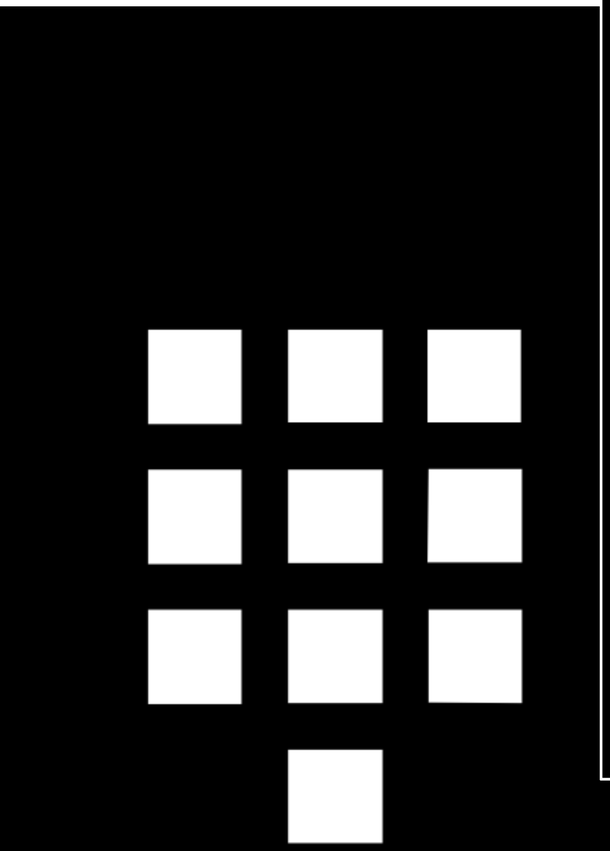
HashiCorp



Terminal

```
$ vault token-create -policy=base
Key            Value
---            -----
token          ce3bd491-2533-7a32-9526-f0ea83c6a68a
token_accessor bf772963-95b1-1776-1b4c-9f214dab071a
token_duration 768h0m0s
token_renewabletrue
token_policies [base default]


$ vault write auth/token/create policies=base
Key            Value
---            -----
token          ce3bd491-2533-7a32-9526-f0ea83c6a68a
token_accessor bf772963-95b1-1776-1b4c-9f214dab071a
token_duration 768h0m0s
token_renewabletrue
token_policies [base default]
```

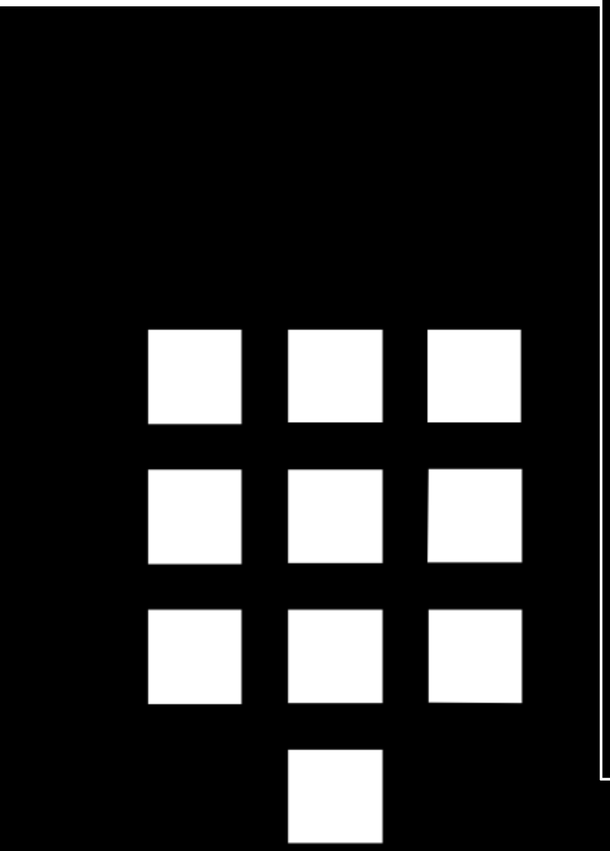
HashiCorp



Terminal

```
$ vault token-create -policy=base
Key            Value
---            -----
token          60a3c690-7120-9edd-e4ed-75eb305c99b6
token_accessor 626c4876-1c6a-01f2-5d2e-d8cc9ec27ae5
token_duration 720h0m0s
token_renewabletrue
token_policies [base default]


$ vault write auth/token/create policies=base
Key            Value
---            -----
token          60a3c690-7120-9edd-e4ed-75eb305c99b6
token_accessor 626c4876-1c6a-01f2-5d2e-d8cc9ec27ae5
token_duration 720h0m0s
token_renewabletrue
token_policies [base default]
```

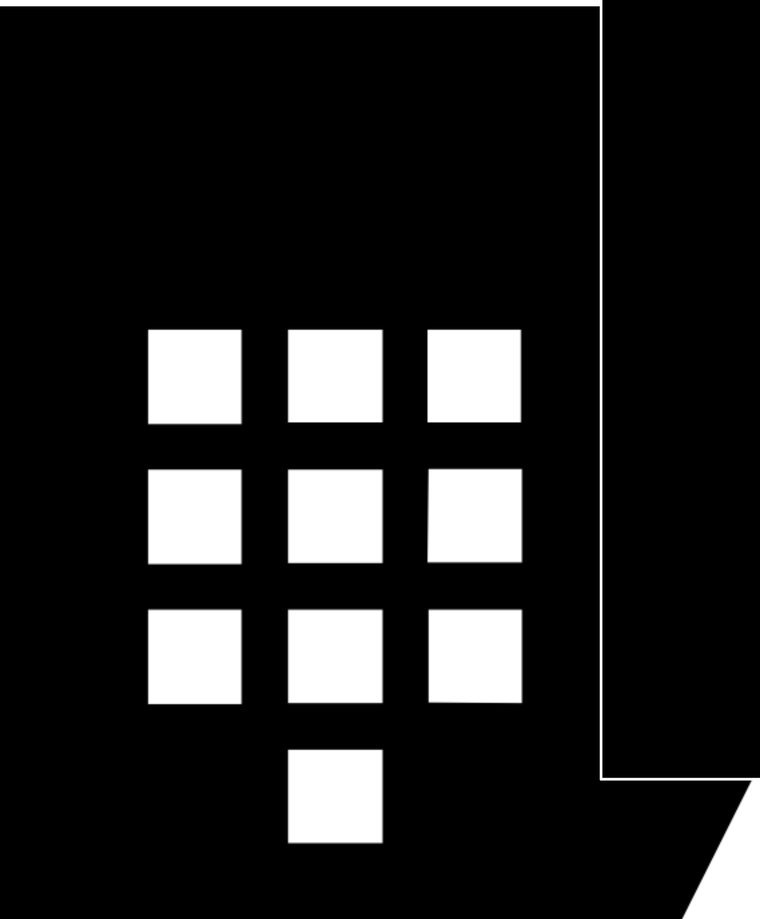
HashiCorp



Terminal

```
$ vault token-create -policy=base
Key            Value
---            -----
token          60a3c690-7120-9edd-e4ed-75eb305c99b6
token_accessor 626c4876-1c6a-01f2-5d2e-d8cc9ec27ae5
token_duration 720h0m0s
token_renewabletrue
token_policies [base default]


$ vault write auth/token/create policies=base
Key            Value
---            -----
token          60a3c690-7120-9edd-e4ed-75eb305c99b6
token_accessor 626c4876-1c6a-01f2-5d2e-d8cc9ec27ae5
token_duration 720h0m0s
token_renewabletrue
token_policies [base default]
```

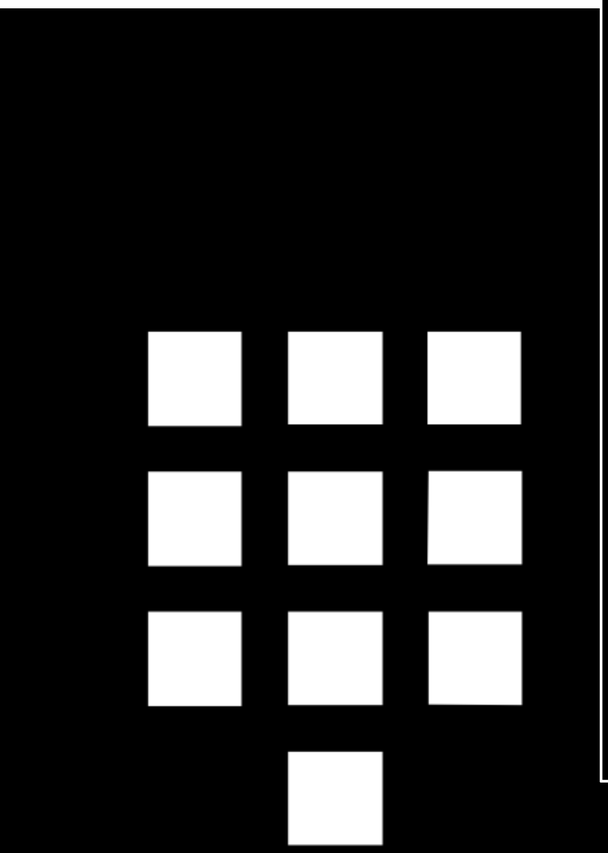
HashiCorp



Terminal

```
$ vault auth 062d33e2-52e8-e60b-2f43-9f09277b0716
Successfully authenticated! You are now logged in.
token: 062d33e2-52e8-e60b-2f43-9f09277b0716
token_duration: 2591973
token_policies: [base, default]
```

HashiCorp

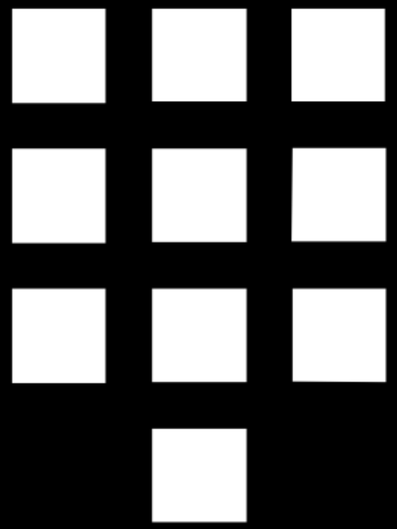


Terminal

```
$ vault policies
```

```
$ vault read sys/policy
```

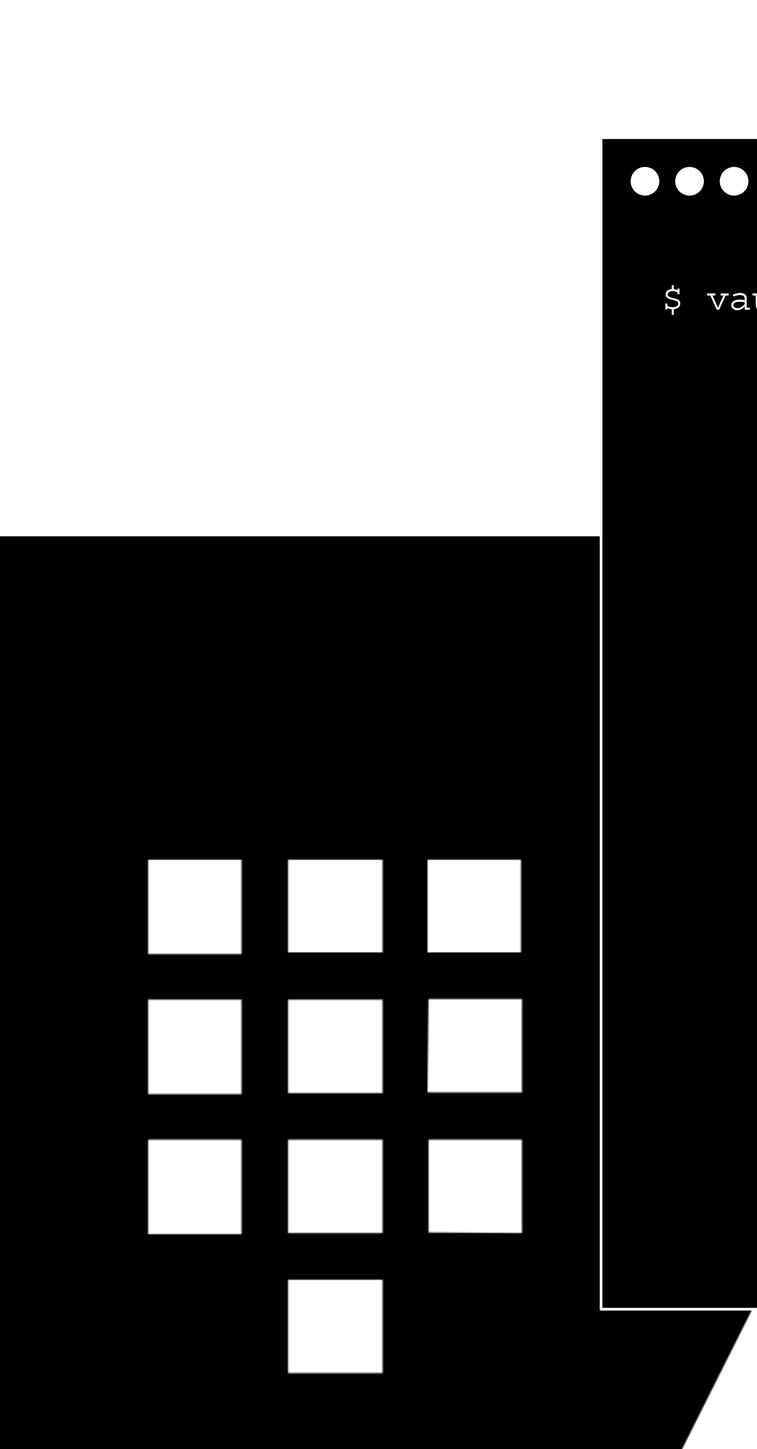
HashiCorp



Terminal

```
$ vault policies
Error: Error making API request.

URL: GET http://127.0.0.1:8200/v1/sys/policy
Code: 403. Errors:

* permission denied



$ vault read sys/policy
Error reading sys/policy: Error making API request.

URL: GET http://127.0.0.1:8200/v1/sys/policy
Code: 403. Errors:

* permission denied
```

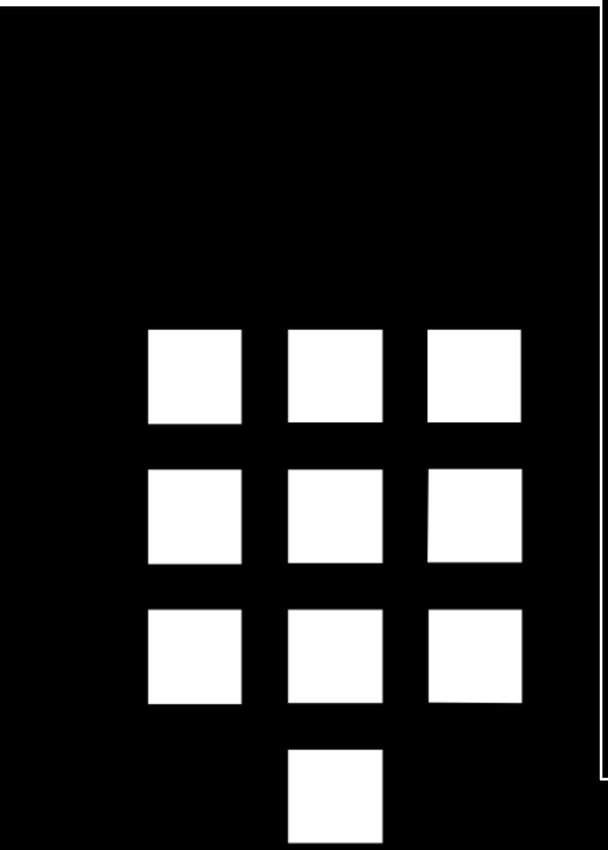
HashiCorp



Terminal

```
$ vault write secret/foo bar=1
```

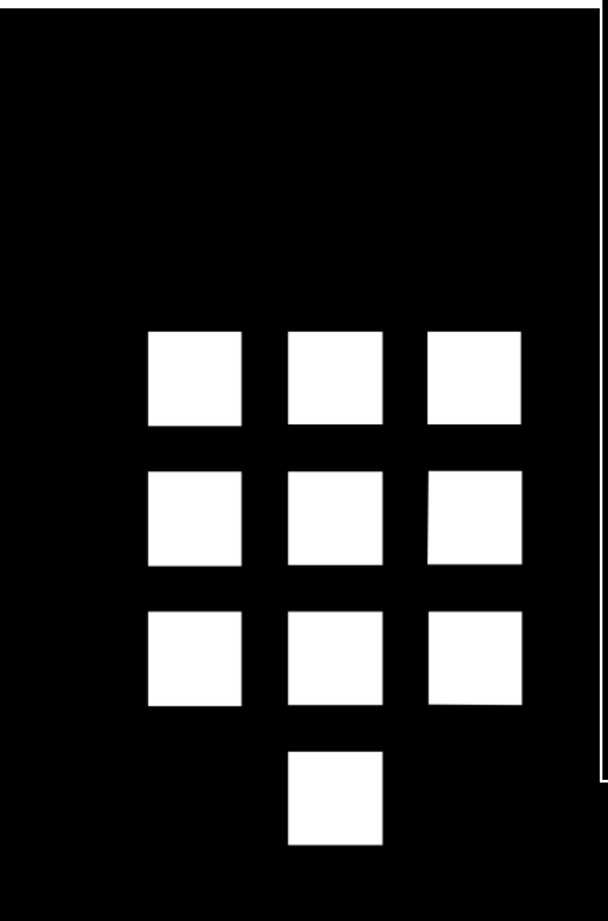
HashiCorp



Terminal

```
$ vault write secret/foo bar=1
Error writing data to secret/foo: Error making API request.

URL: PUT http://192.168.50.151:8200/v1/secret/foo
Code: 403. Errors:

* permission denied
```

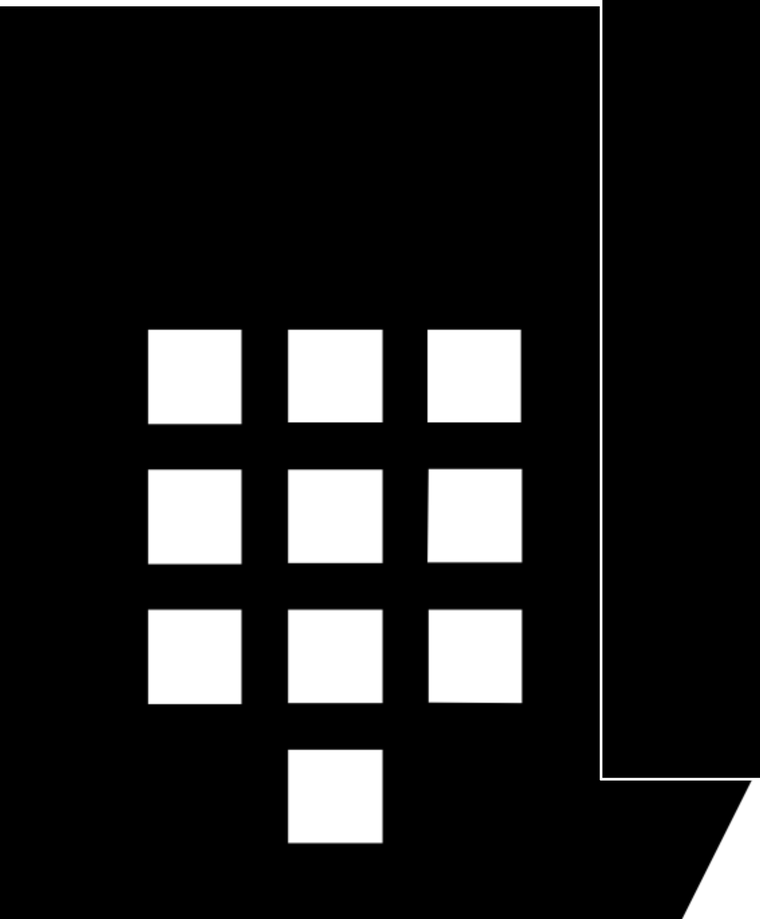
HashiCorp



Terminal

```
$ vault write secret/training_foo bar=1
```

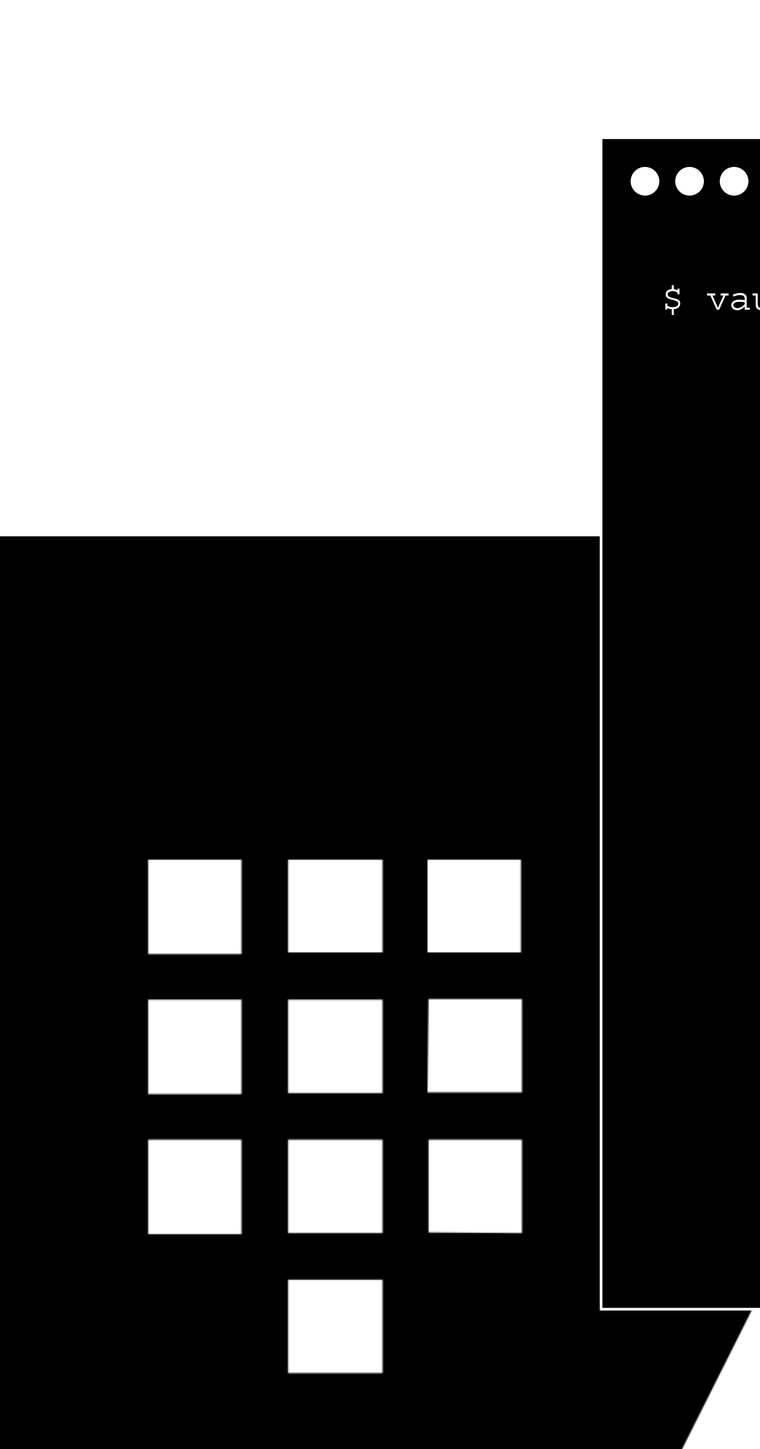
HashiCorp



Terminal

```
$ vault write secret/training_foo bar=1
Success! Data written to: secret/training_foo
```

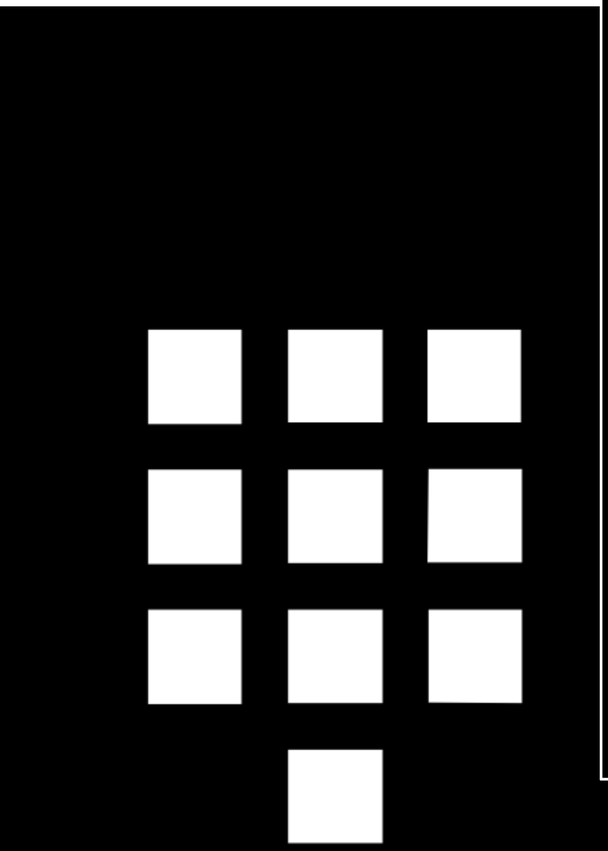
HashiCorp



Terminal

```
$ vault write secret/training_foo bar=2
```

HashiCorp

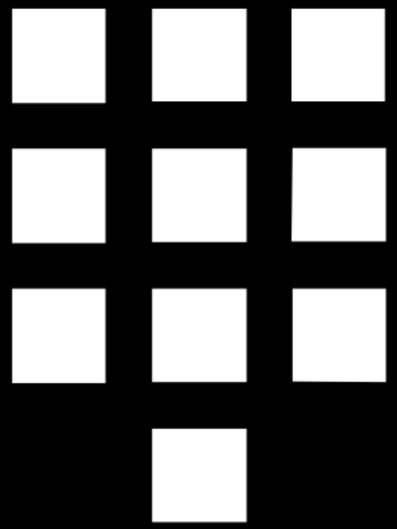


Terminal

```
$ vault write secret/training_foo bar=2
Error writing data to secret/training_foo: Error making API request.

URL: PUT http://192.168.50.151:8200/v1/secret/training_foo
Code: 403. Errors:

* permission denied
```

HashiCorp

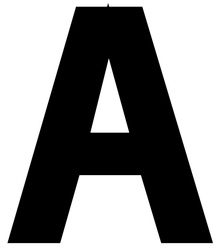
Q

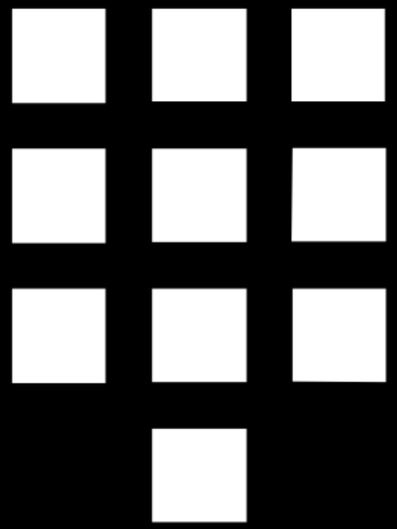


Terminal

```
$ vault write secret/training_ bar=1
```

HashiCorp





Terminal

```
$ vault write secret/training_ bar=1
Success! Data written to: secret/training_
```

HashiCorp

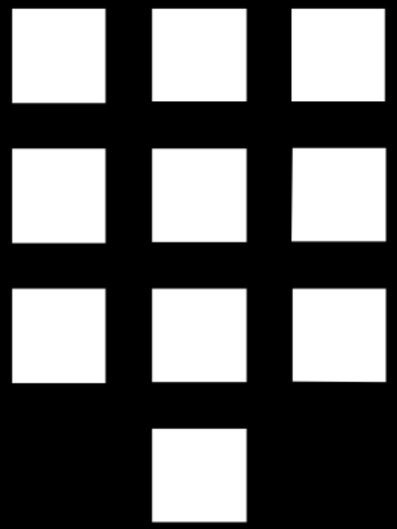
Exercise: Create a Policy

Write a policy named "exercise" that permits listing and deleting anything in the generic secret backend, but forbids creating, reading, or updating a secret. **Do not upload the policy.**

```
$ vault list secret/       # ok
$ vault delete secret/foo # ok
$ vault read secret/foo   # 403
$ vault write secret/foo  # 403
```

exercise.hcl

```
path "secret/*" {
  capabilities = ["delete", "list"]
}
```

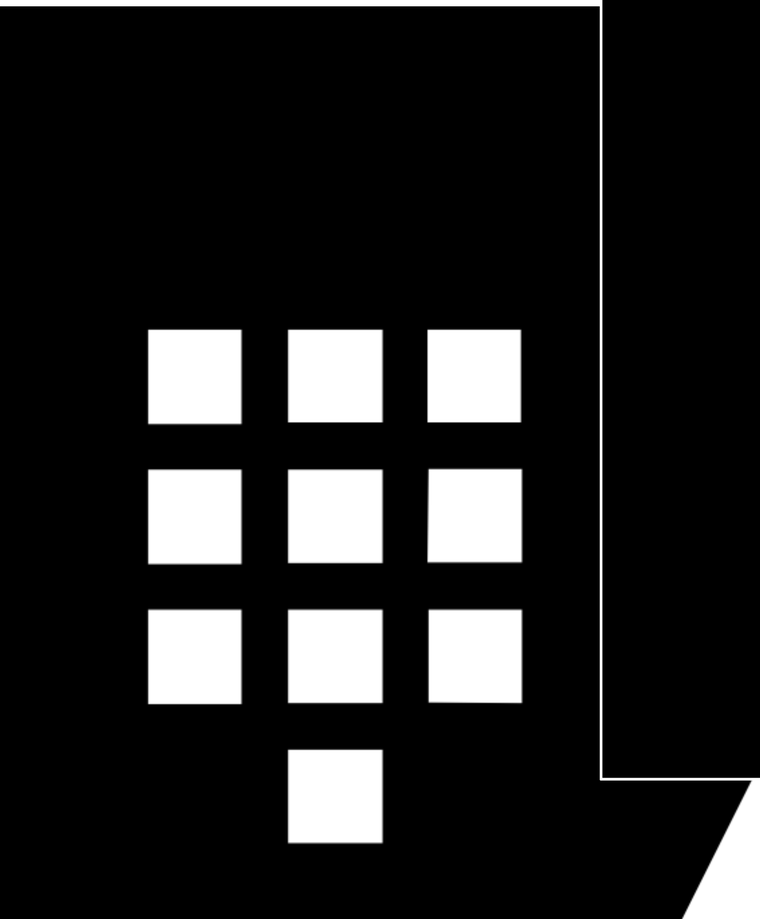


HashiCorp

Exercise: Re-auth as root

Re-authenticate as root

(Our current user does not have enough permission)



Terminal

```
$ vault auth root
Successfully authenticated! You are now logged in.
token: root
token_duration: 0
token_policies: [root]
```

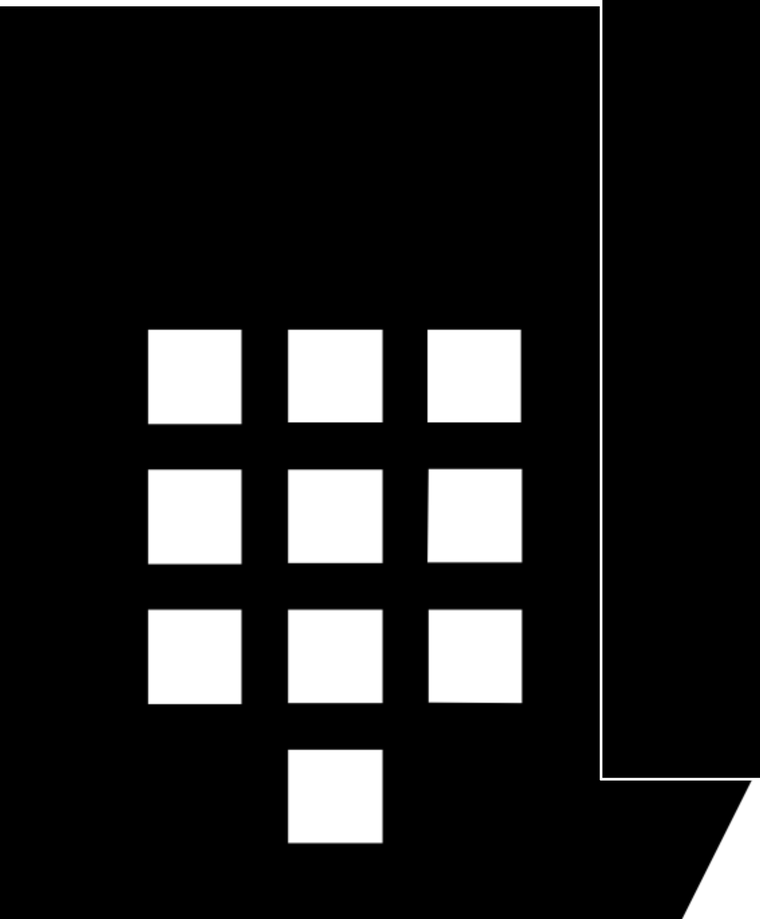
HashiCorp

Dynamic Secrets

Secret Backends

Most secret backends must be mounted before use.

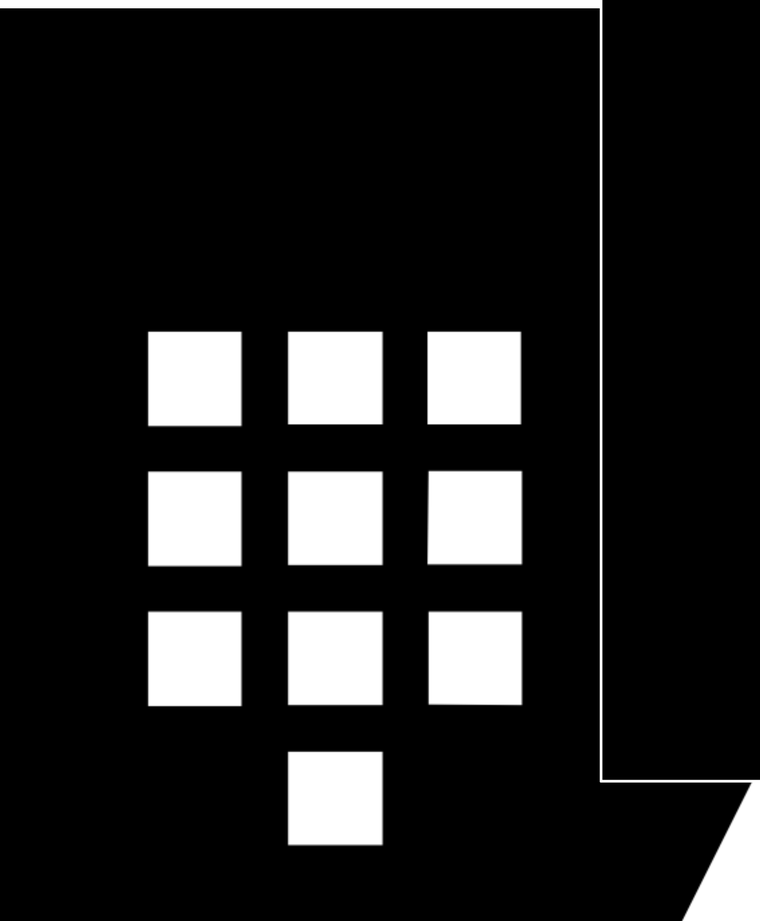
Many secret backends require additional configuration before use.



Terminal

```
$ vault mount database
Successfully mounted 'database' at 'database'!
```

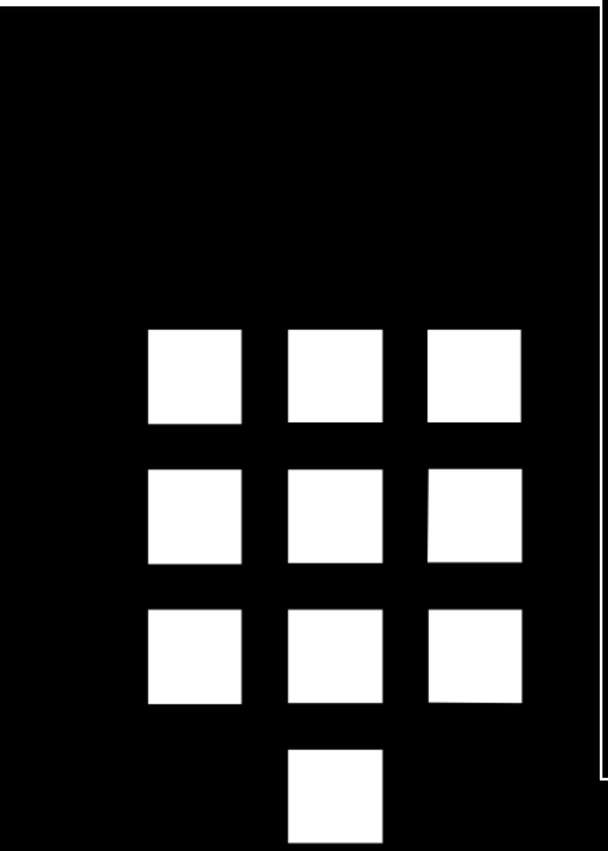
HashiCorp



Terminal

```
$ vault path-help database/
```

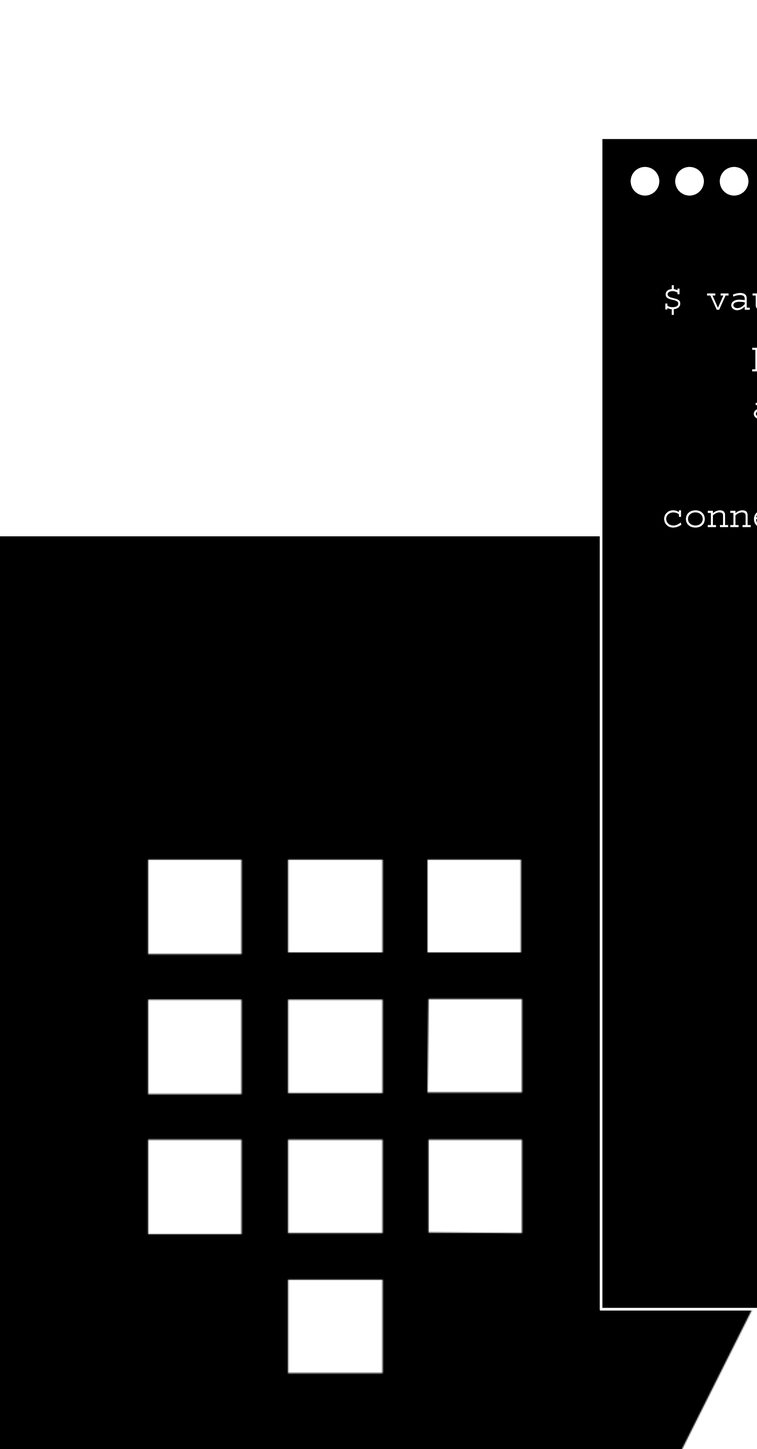
HashiCorp



Terminal

```
$ vault path-help database/
The following paths are supported by this backend. To view help for
any of the paths below, use the help command with any route matching
the path pattern. Note that depending on the policy of your auth token,
you may or may not be able to access certain paths.

    ^config/(?P<name>\w(([\w-.]+)?\w)?)$
        Configure connection details to a database plugin.

    ^config/?$
        Configure connection details to a database plugin.

    ^creds/(?P<name>\w(([\w-.]+)?\w)?)$
        Request database credentials for a certain role.

    ^reset/(?P<name>\w(([\w-.]+)?\w)?)$
        Resets a database plugin.

    ^roles/(?P<name>\w(([\w-.]+)?\w)?)$
```

HashiCorp



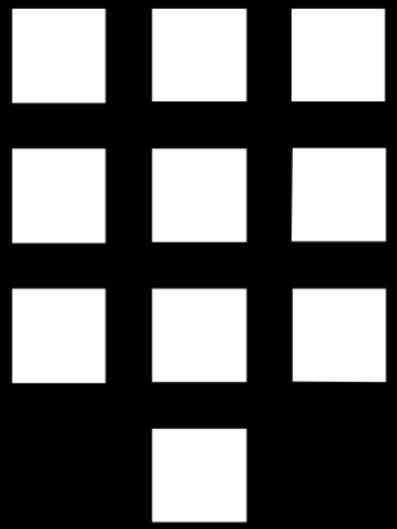
Terminal

```
$ vault write database/config/postgresql \
    plugin_name=postgresql-database-plugin \
    allowed_roles=readonly \

connection_url=postgresql://postgres@192.168.50.154/postgres?sslmode=disable
```

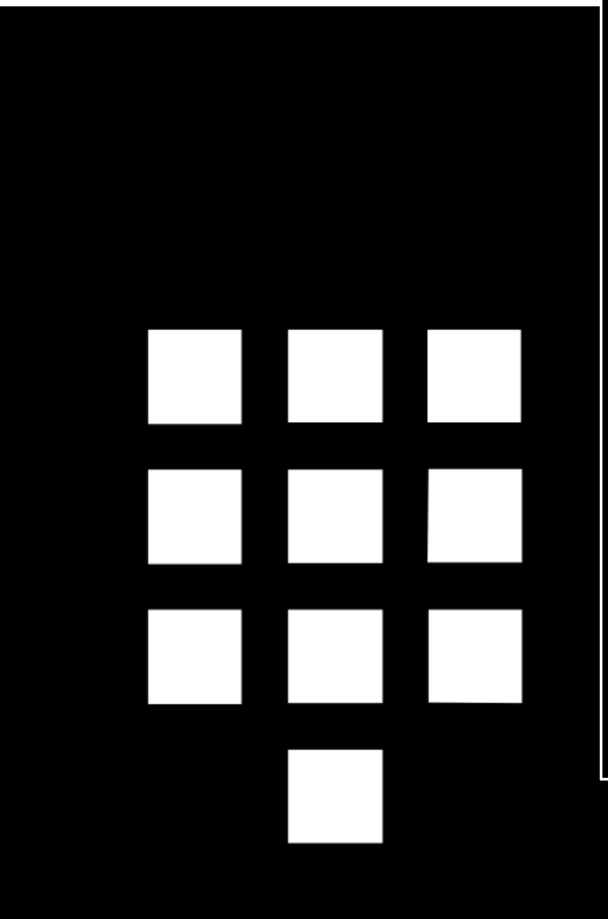
HashiCorp

Terminal

```
$ vault write database/config/postgresql \
    plugin_name=postgresql-database-plugin \
    allowed_roles=readonly \

connection_url=postgresql://postgres@192.168.50.154/postgres?sslmode=disable

The following warnings were returned from the Vault server:
* Read access to this endpoint should be controlled via ACLs as it will return
the connection string or URL as it is, including passwords, if any.
```



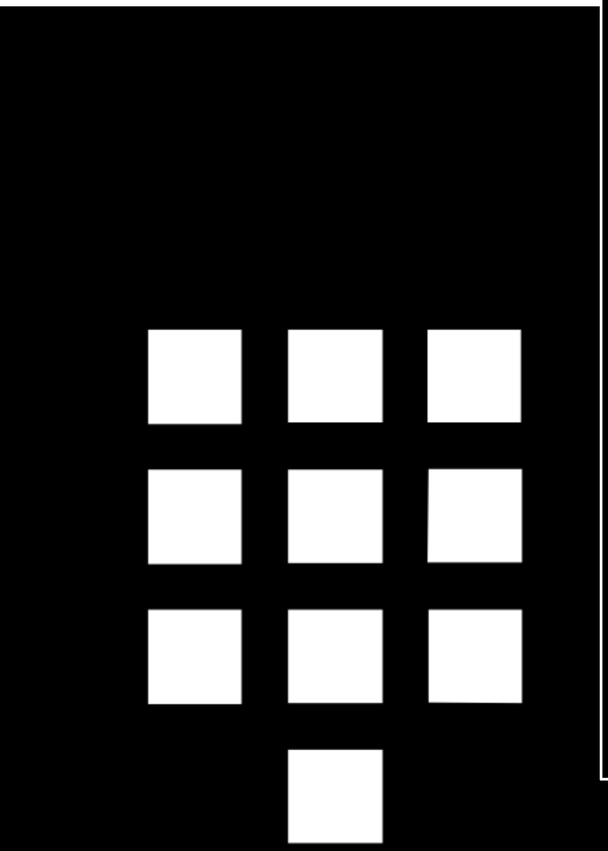
HashiCorp



Terminal

```
$ vault write database/roles/readonly \
    db_name=postgresql \
    creation_statements=@readonly.sql \
    default_ttl=1h \
    max_ttl=24h
Success! Data written to: database/roles/readonly
```

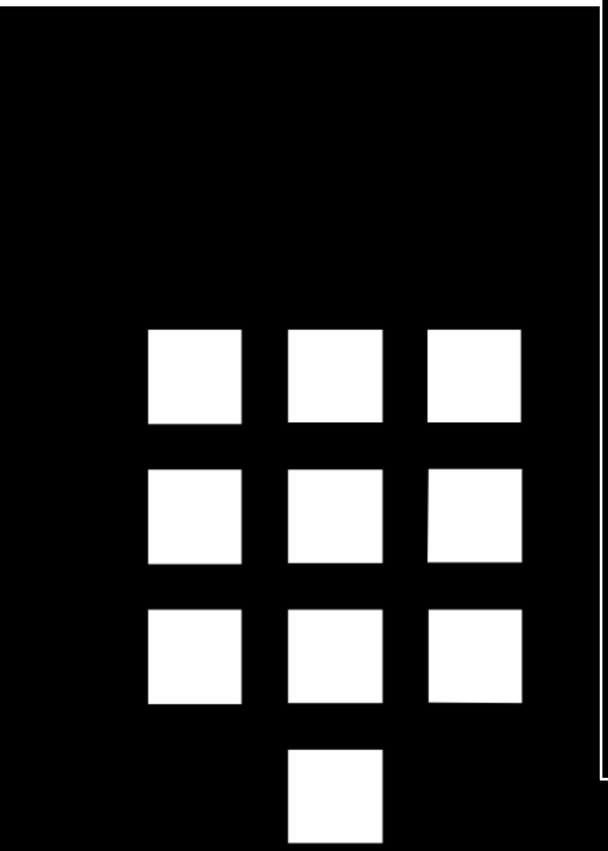
HashiCorp



Terminal

```
$ cat readonly.sql
CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL
'{{expiration}}';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO "{{name}}";
```

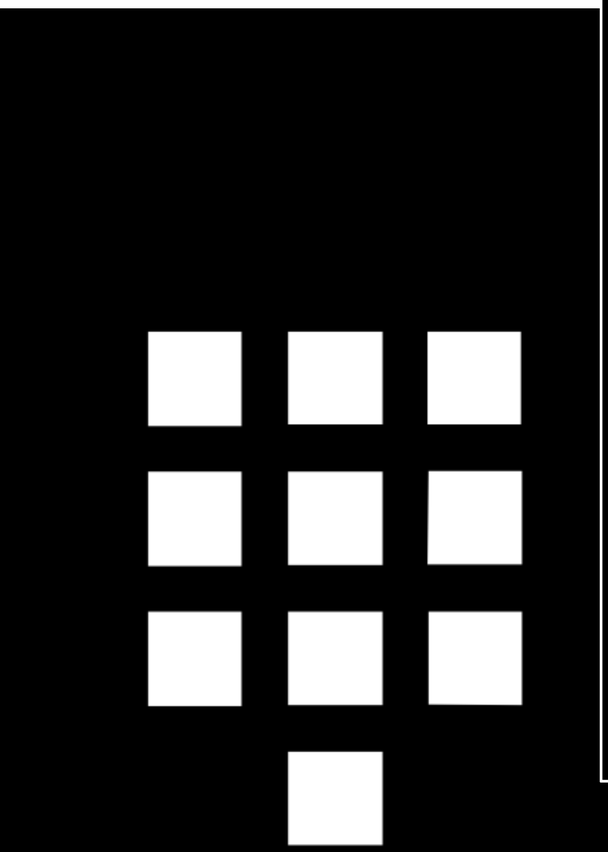
HashiCorp



Terminal

```
$ cat readonly.sql
CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL
'{{expiration}}';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO "{{name}}";
```

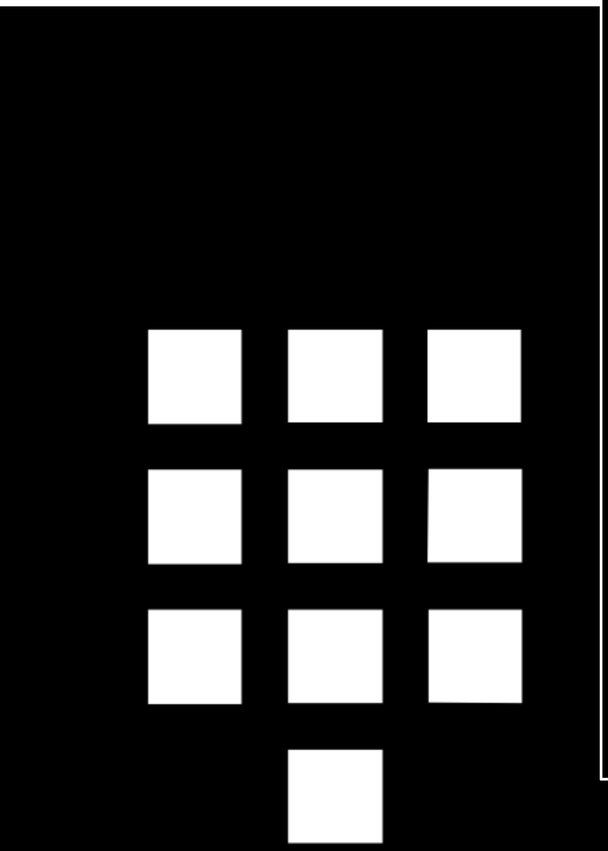
HashiCorp



Terminal

```
$ vault read database/creds/readonly
```

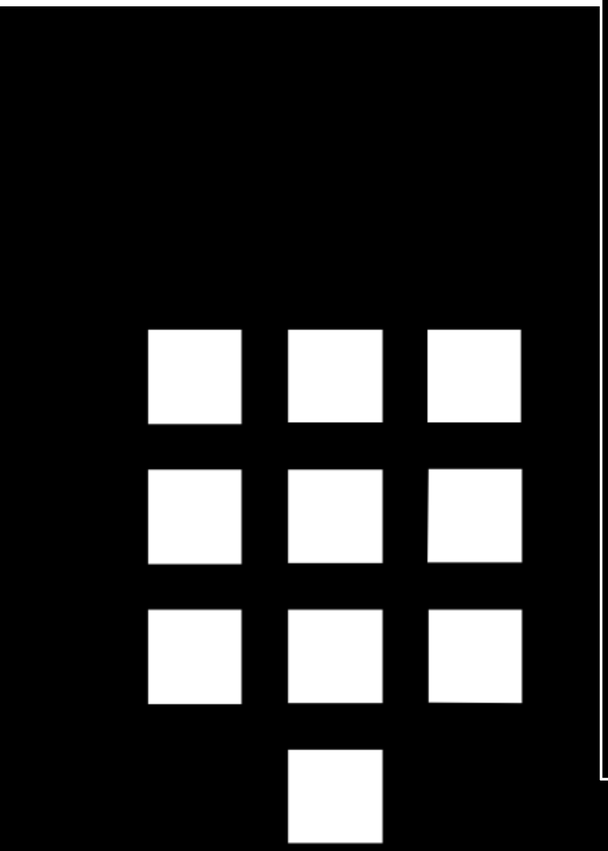
HashiCorp



Terminal

```
$ vault read database/creds/readonly
Key            Value
---            -----
lease_id       database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b
lease_duration 1h0m0s
lease_renewabletrue
password       A1a-s02p77zvq75yyyq6
username       v-token-readonly-71q3zrty1pu4z7p508pp-1505187364
```

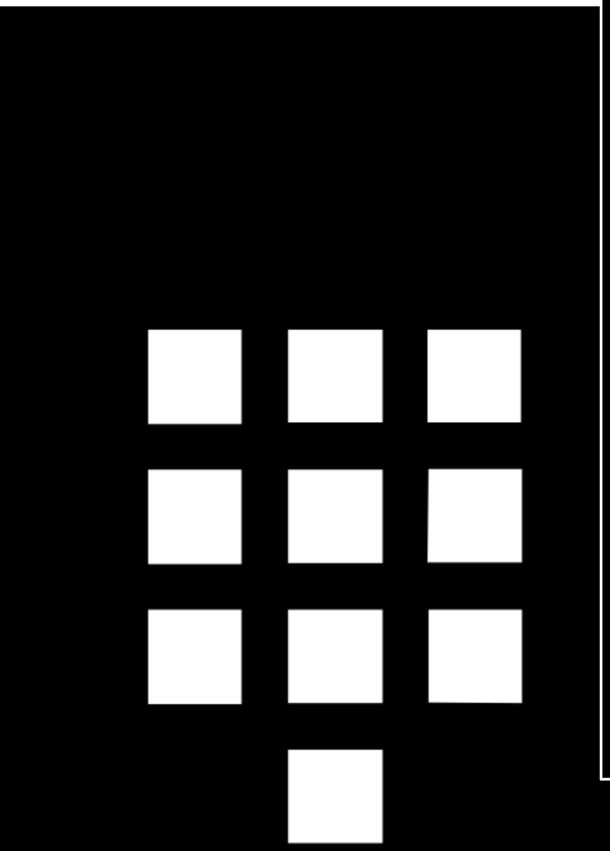
HashiCorp



Terminal

```
$ vault read database/creds/readonly
Key            Value
---            -----
lease_id       database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b
lease_duration 1h0m0s
lease_renewabletrue
password       A1a-s02p77zvq75yyyq6
username       v-token-readonly-71q3zrty1pu4z7p508pp-1505187364
```

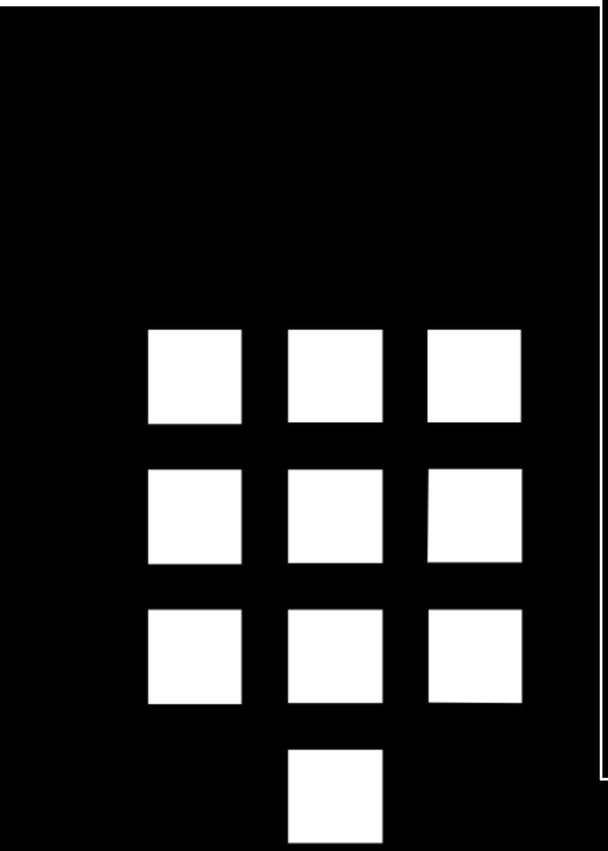
HashiCorp



Terminal

```
$ psql -U postgres

postgres=# \du

                   Role name                    |        Attributes         |
Mem
------------------------------------------------+---------------------------|
postgres                                        | Superuser, Create ...
|v-token-readonly-71q3zrty1pu4z7p508pp-1505187364 | Password valid until ... |

postgres=# \q
```

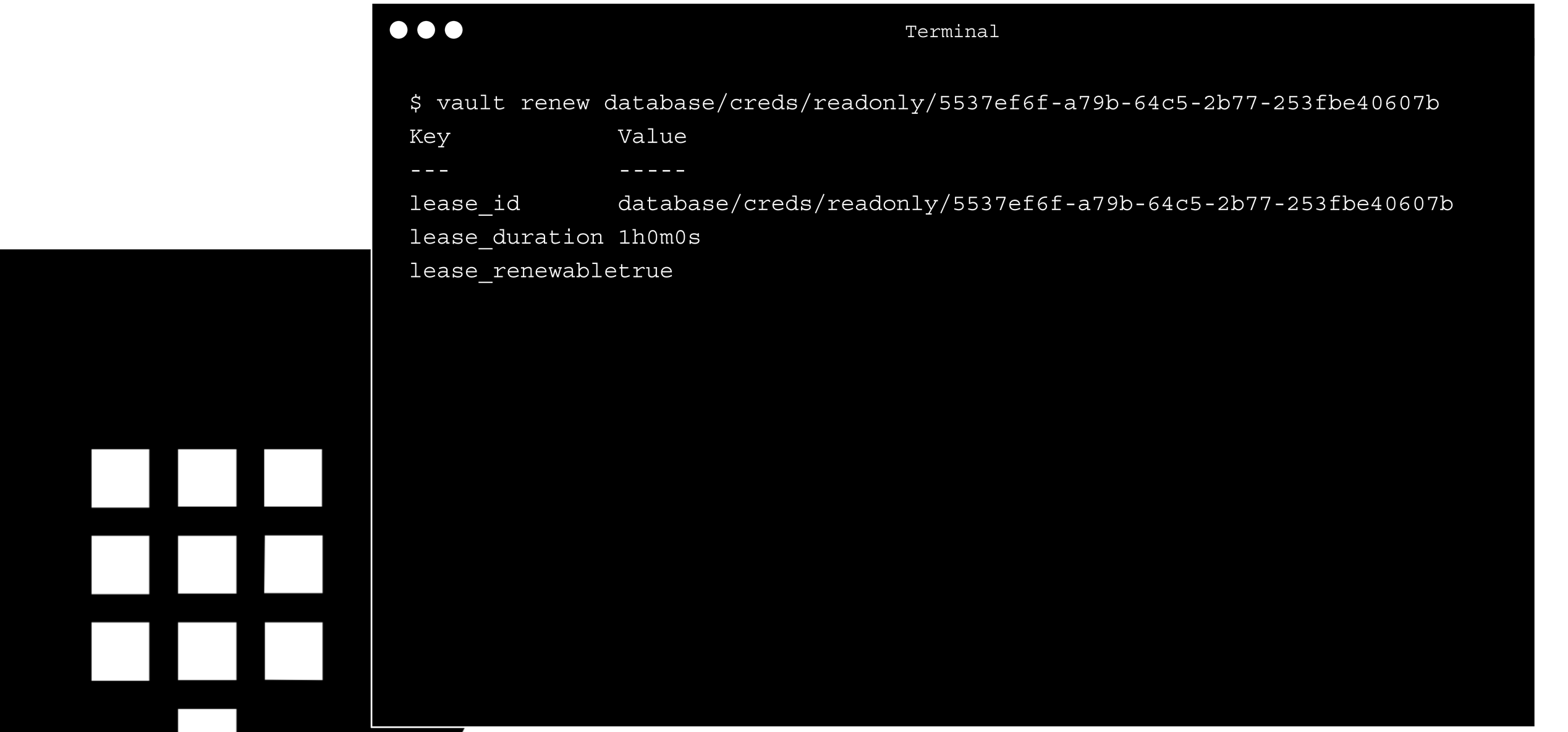
HashiCorp



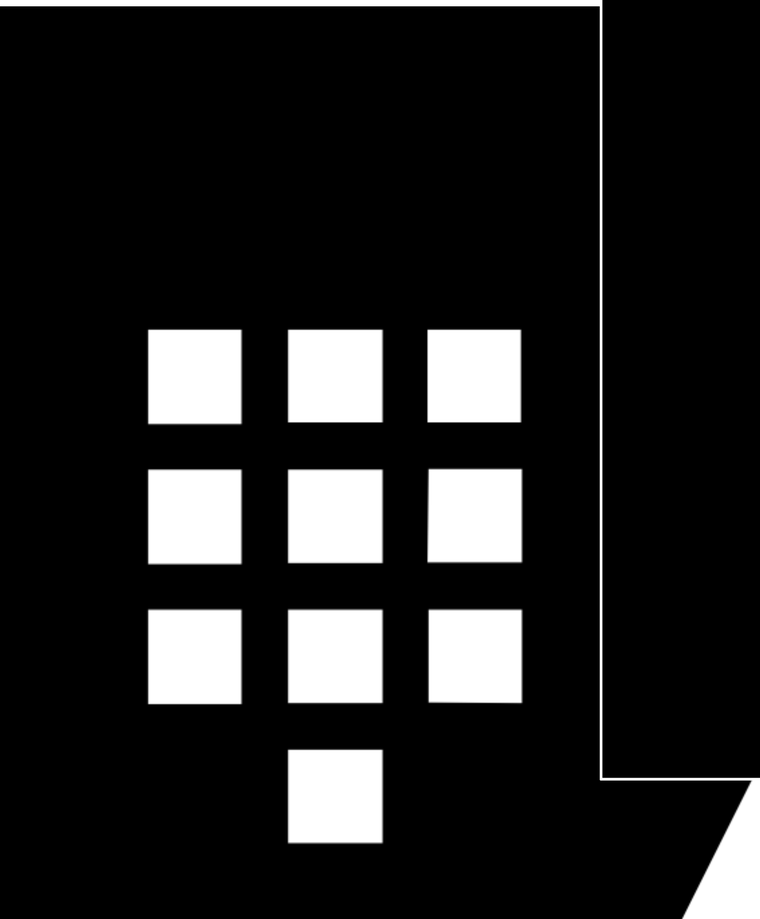
Terminal

```
$ vault renew database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b
```

HashiCorp


Terminal
$ vault renew database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b
Key Value
--- -----
lease_id database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b
lease_duration 1h0m0s
lease_renewabletrue

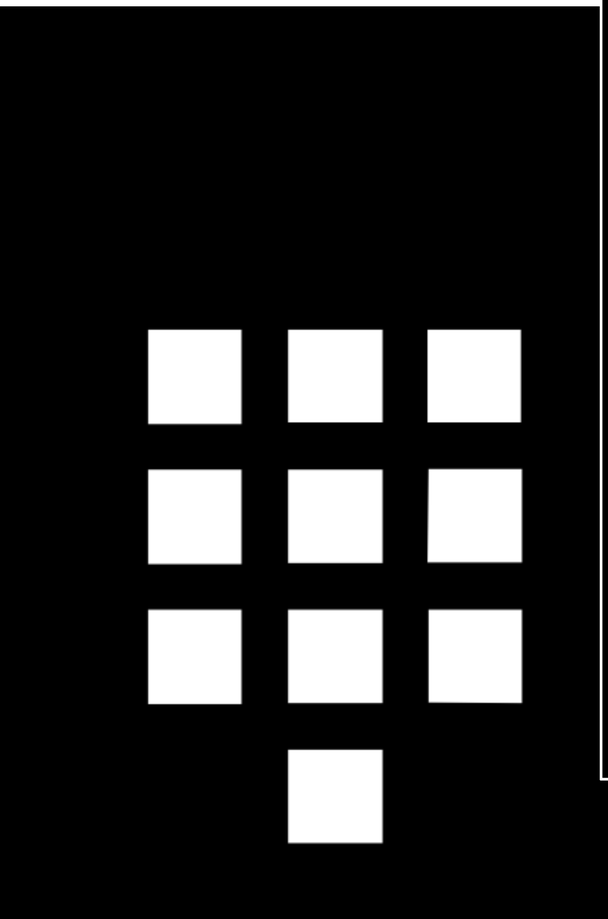
HashiCorp



Terminal

```
$ vault revoke database/creds/readonly/e78e8b77-8738-674d-c03b-642f28ae322a
```

HashiCorp



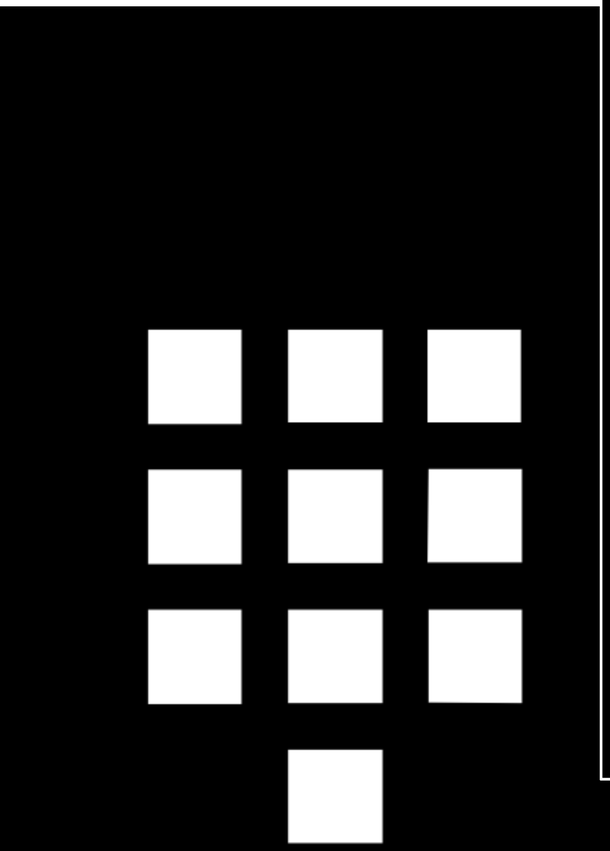
Terminal

```
$ vault revoke database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b
Success! Revoked the secret with ID
'database/creds/readonly/5537ef6f-a79b-64c5-2b77-253fbe40607b', if it existed.
```

HashiCorp


Terminal
$ psql -U postgres
postgres=# \du
Role name | Attributes | Mem
postgres | Superuser, Create ... | {}
postgres=# \q

HashiCorp



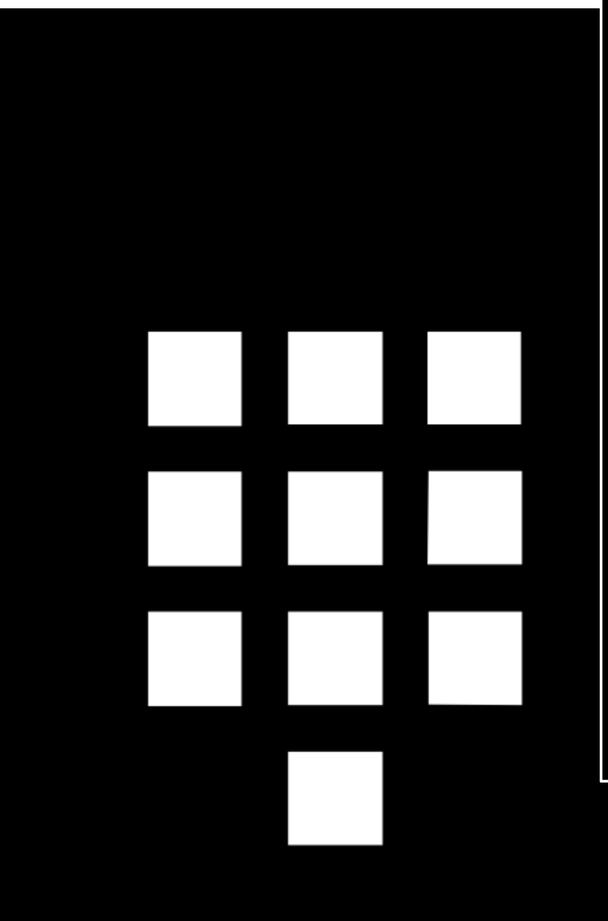
Terminal

```
$ vault read database/creds/readonly
password       A1a-5v4vu5w1xu2sr0vy
username       v-token-readonly-x98pq0vrw1qzt99z06xy-1505187883

$ vault read database/creds/readonly
password       A1a-z7v93ssy1z4r4r6p
username       v-token-readonly-67tu0wz1ys95q7vxv0r2-1505187890

$ vault read database/creds/readonly
password       A1a-85ypstvpy329tv5r
username       v-token-readonly-uu8xr0972z195s304324-1505187898

# ...
```

HashiCorp

Terminal

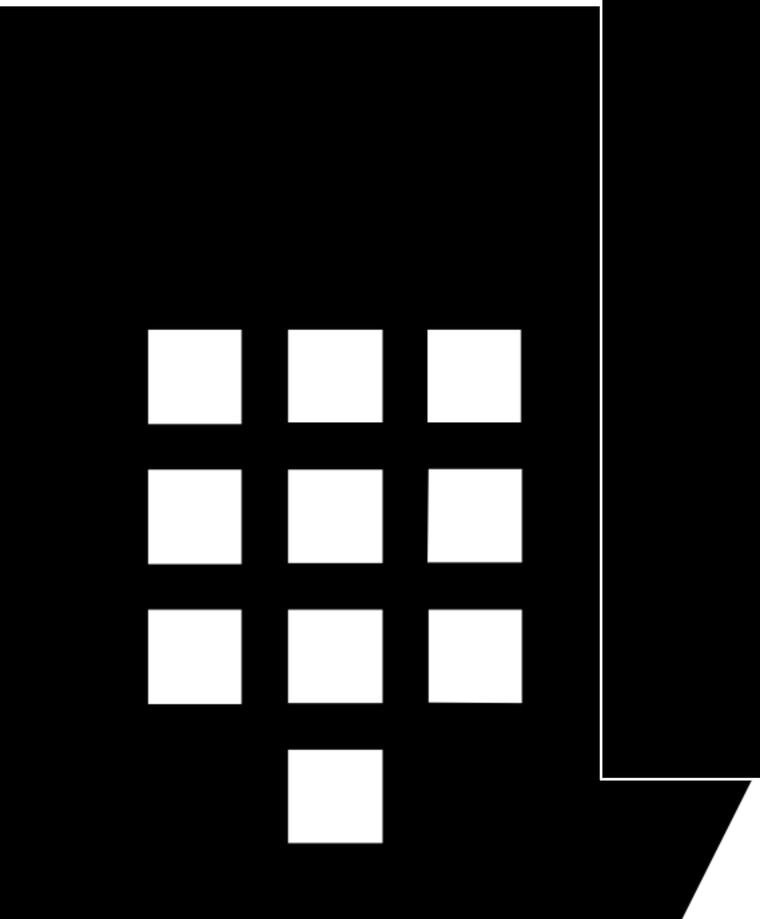
HashiCorp



Terminal

```
$ vault revoke -prefix database/creds
```

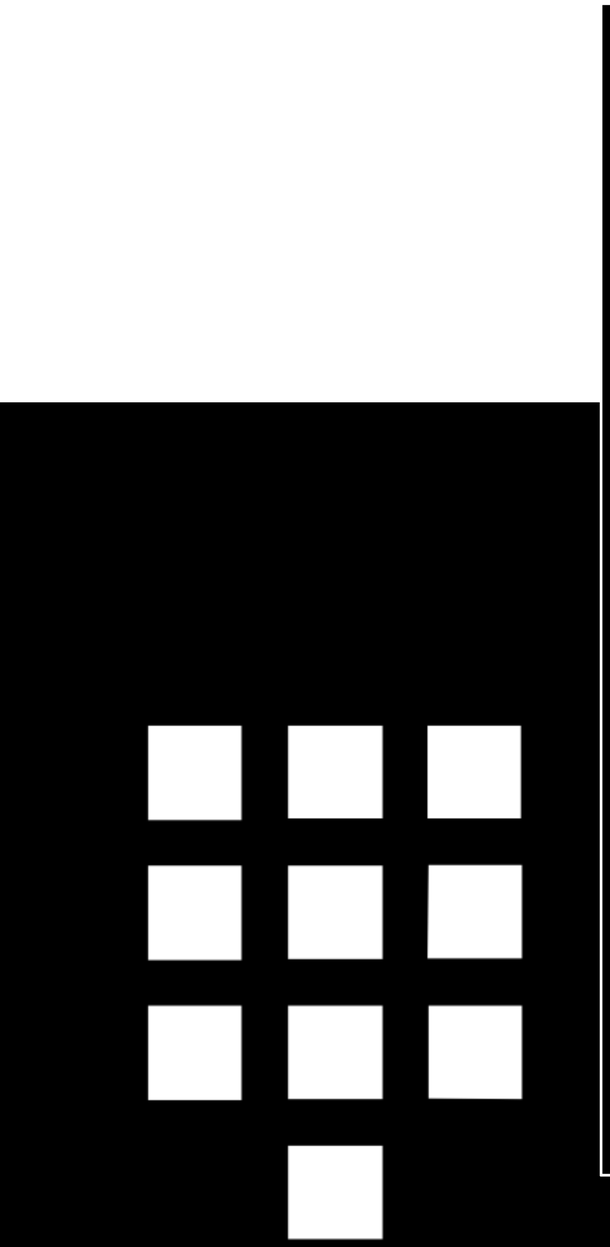
HashiCorp



Terminal

```
$ vault revoke -prefix database/creds
Success! Revoked the secret with ID 'database/creds', if it existed.
```

HashiCorp



Terminal

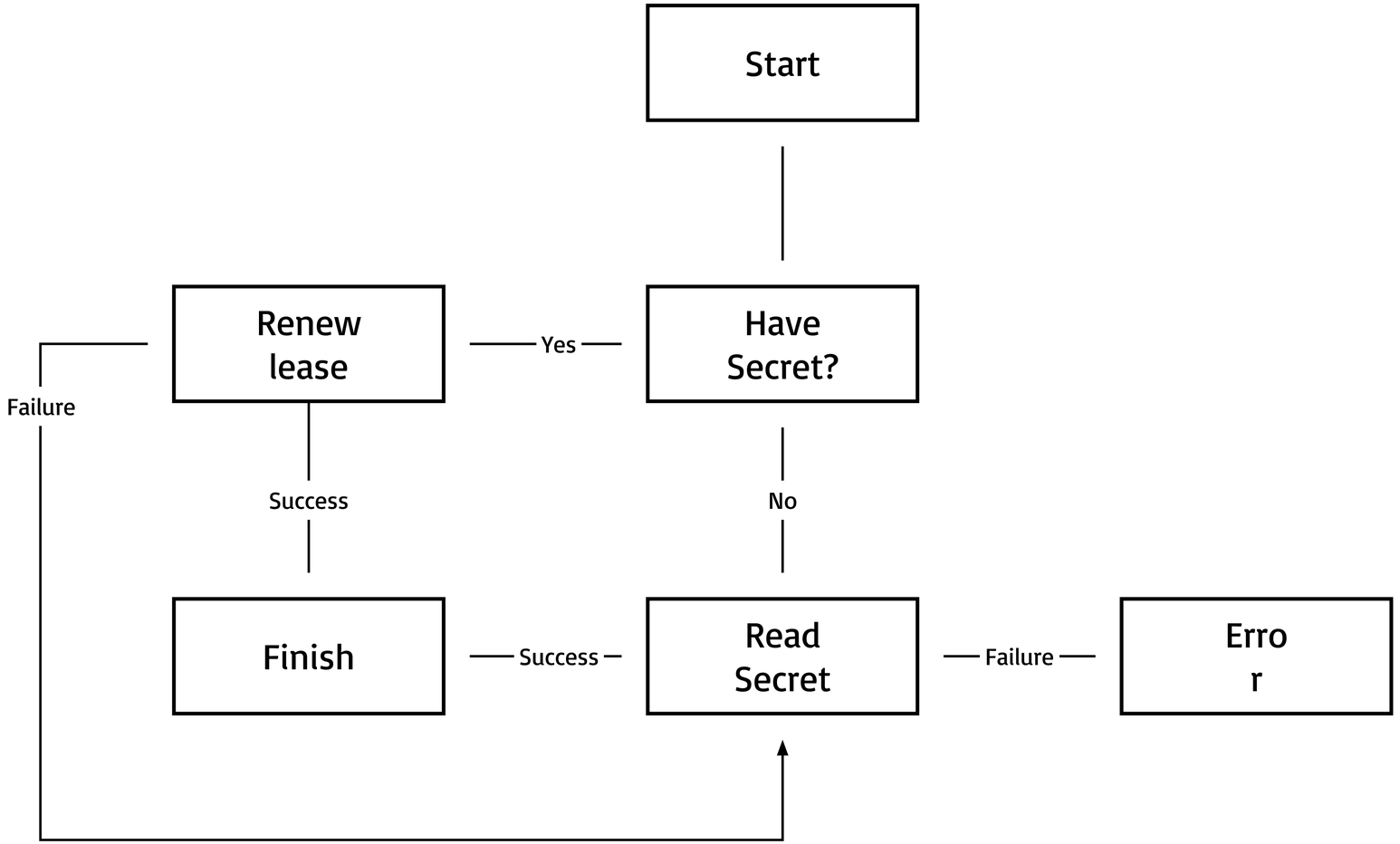
```
$ psql -U postgres

postgres=# \du

                   Role name                   |         Attributes         | Mem
-----------------------------------------------+----------------------------+-------
postgres                                       | Superuser, Create ...      | {}

postgres=# \q
```

HashiCorp

Working with Leases

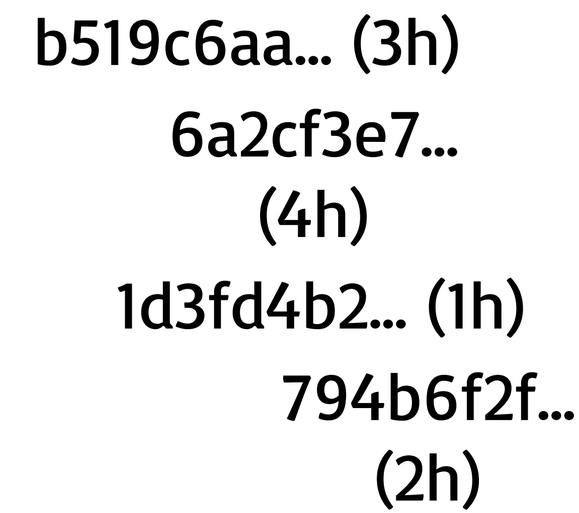
HashiCorp

Leasing, renewal, and revocation



HashiCorp

Lease Hierarchy and Revocations



Exercise: Predicting Behavior

List the order in which the leases would expire.

b519c6aa... (3h)

6a2cf3e7... (4h)

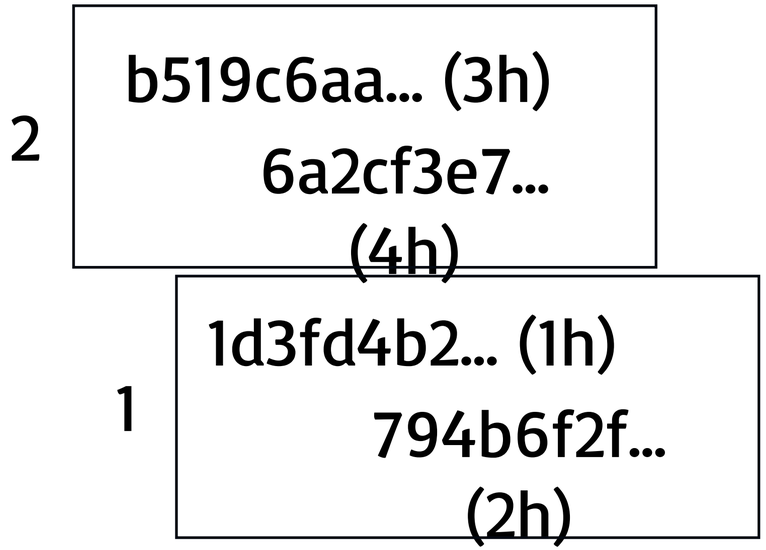
1d3fd4b2... (1h)

794b6f2f... (2h)

HashiCorp

Exercise: Predicting Behavior

List the order in which the leases would expire.



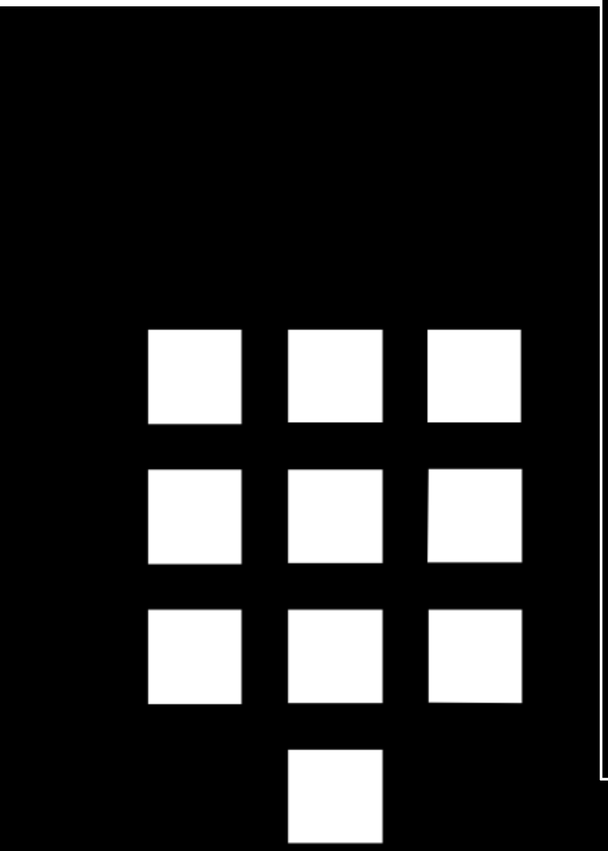
Token and Lease Renewals

If a token or secret with a lease is not renewed before the lease expires, it and all children will be revoked by the Vault server.

A child is a token, secret, or authentication created by a parent. A parent is almost always a token.

Exercise: Understanding Revocations

1. Create a new token with a 30s lease (hint: use help output)
2. Auth as this token
3. After 30s, try to read a value using token



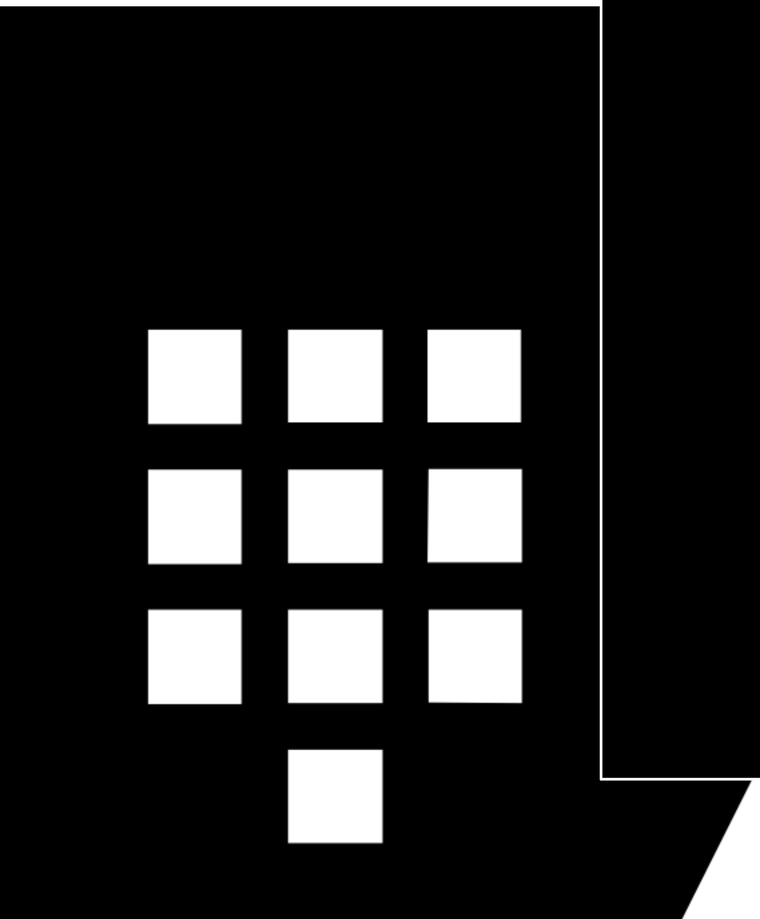
Terminal

```
$ vault token-create -ttl=30s
token          5cf4e7b5-7f88-0769-e406-686e2a90c471
token_duration 30

$ vault auth 5cf4e7b5-7f88-0769-e406-686e2a90c471
Successfully authenticated!

$ vault read secret/training
...

$ vault read secret/training
Error!
```

HashiCorp

Exercise: Re-auth as root

Re-authenticate as root



Terminal

```
$ vault auth root
Successfully authenticated! You are now logged in.
token: root
token_duration: 0
token_policies: [root]
```

HashiCorp

Lease Best Practices

Renew leases at half the lease duration value – e.g. 10m lease should renew every 5m.

Attempt a re-read if renewal fails (generates new credentials).

Notable Exception: Orphan Token

Root/sudo users have the ability to generate "orphan" tokens.

Orphan tokens are not children of their parent, therefore do not expire when their parent does.

Orphan tokens still expire when their own Max TTL is reached.

Notable Exception: Periodic Token

Root/sudo users have the ability to generate "periodic" tokens.

Periodic tokens have a TTL, but no max TTL.

Periodic tokens may live for an infinite amount of time, so long as they are renewed within their TTL.

This is useful for long-running services that cannot handle regenerating a token.

Notable Exception: Use Limits

In addition to TTL and Max TTL, tokens may be limited to a number of uses.

Use limit tokens expire at the end of their last use, regardless of their remaining TTLS.

Use limit tokens expire at the end of their TTLs, regardless of remaining uses.

Authentication

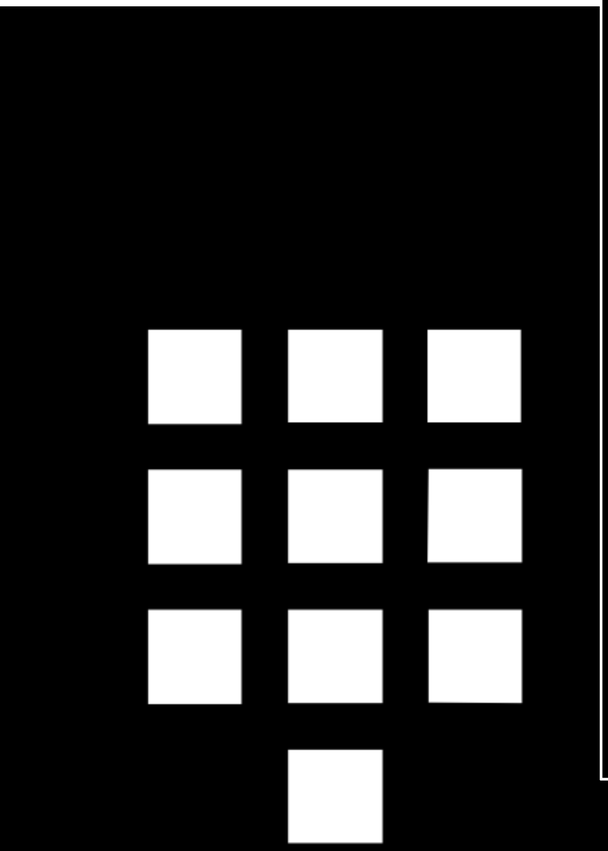
Understanding Authentication

Authentication is a process in Vault by which user or machine-supplied information is verified to create a token with pre-configured policy.

Future requests are made using the token.

Authentication Setup

1. Activate the authentication using the auth-enable command
2. Configure the authentication (varies)
3. Map the authentication to a set of policies

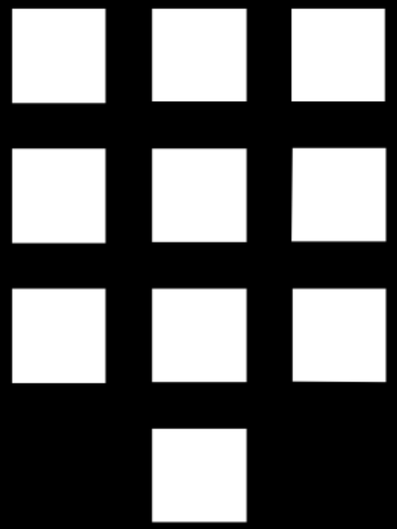


Terminal

```
$ vault auth -methods
```

```
$ vault read sys/auth
```

HashiCorp

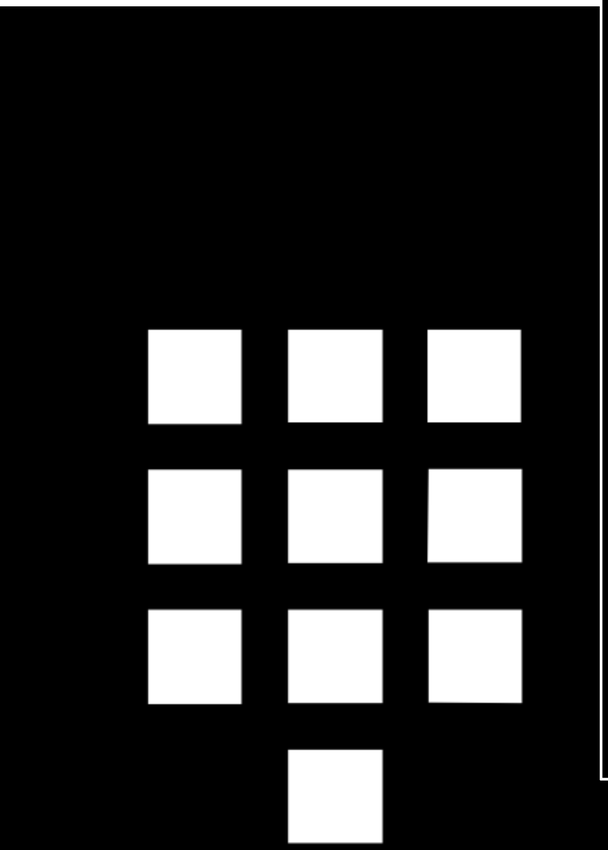


Terminal

```
$ vault auth -methods
Path    Type   Default TTL  Max TTL  Replication Behavior  Description
token/  token  system       system   replicated            token based
```

```
$ vault read sys/auth
Key    Value
---    -----
token/ map[description:token based credentials local:false type:token #...]
```

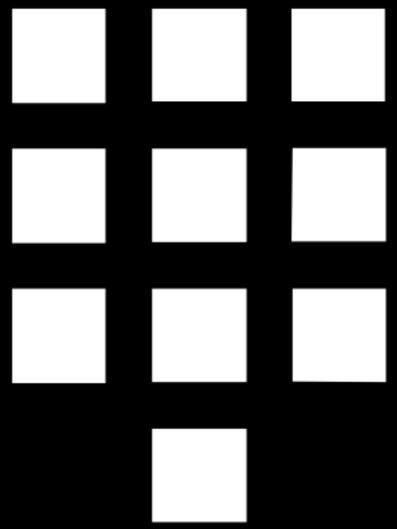
HashiCorp



Terminal

```
$ vault auth-enable userpass

$ vault write sys/auth/userpass type=userpass
```

HashiCorp

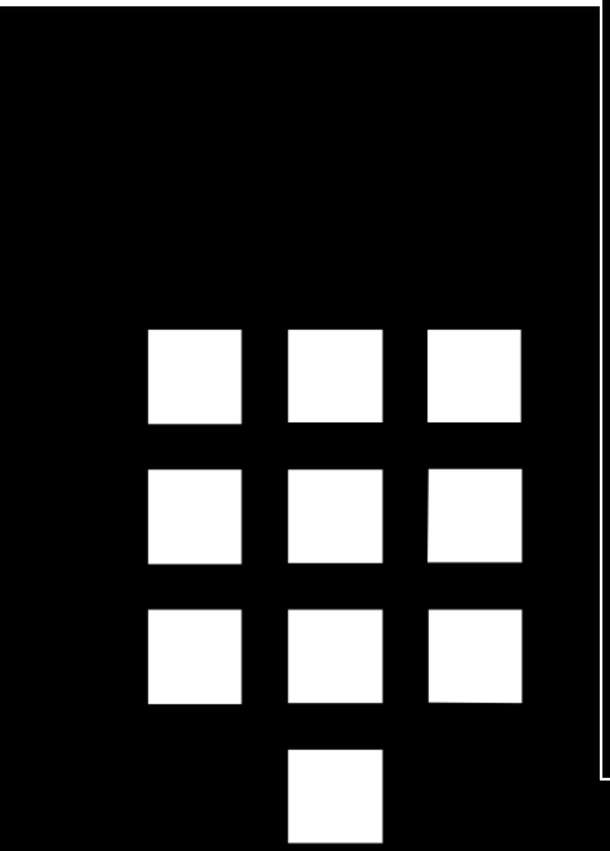


Terminal

```
$ vault auth-enable userpass
Successfully enabled 'userpass' at 'userpass'!
```

```
$ vault write sys/auth/userpass type=userpass
Success! Data written to: sys/auth/userpass
```

HashiCorp



Terminal

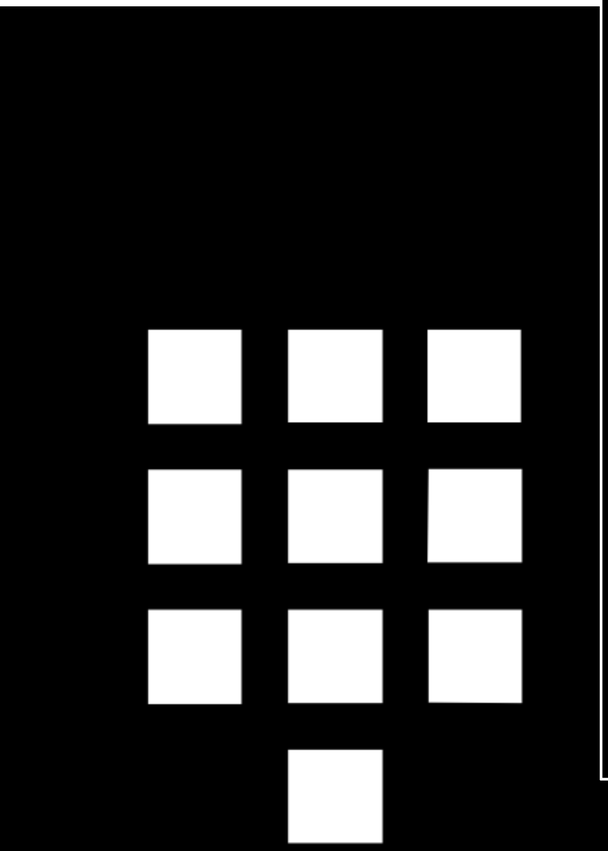
```
$ vault auth -methods
Path        Type      Default TTL  Max TTL  Replication Behavior  Description
token/      token     system       system   replicated            token based
userpass/   userpass  system       system   replicated
```

```
$ vault read sys/auth
Key         Value
---         -----
token/      map[local:false type:token config:map[default_lease_ttl:0 # ...]
userpass/   map[config:map[default_lease_ttl:0 max_lease_ttl:0] # ...]
```

HashiCorp

Exercise: Mount at Path

Mount the userpass backend at the path "training-userpass".

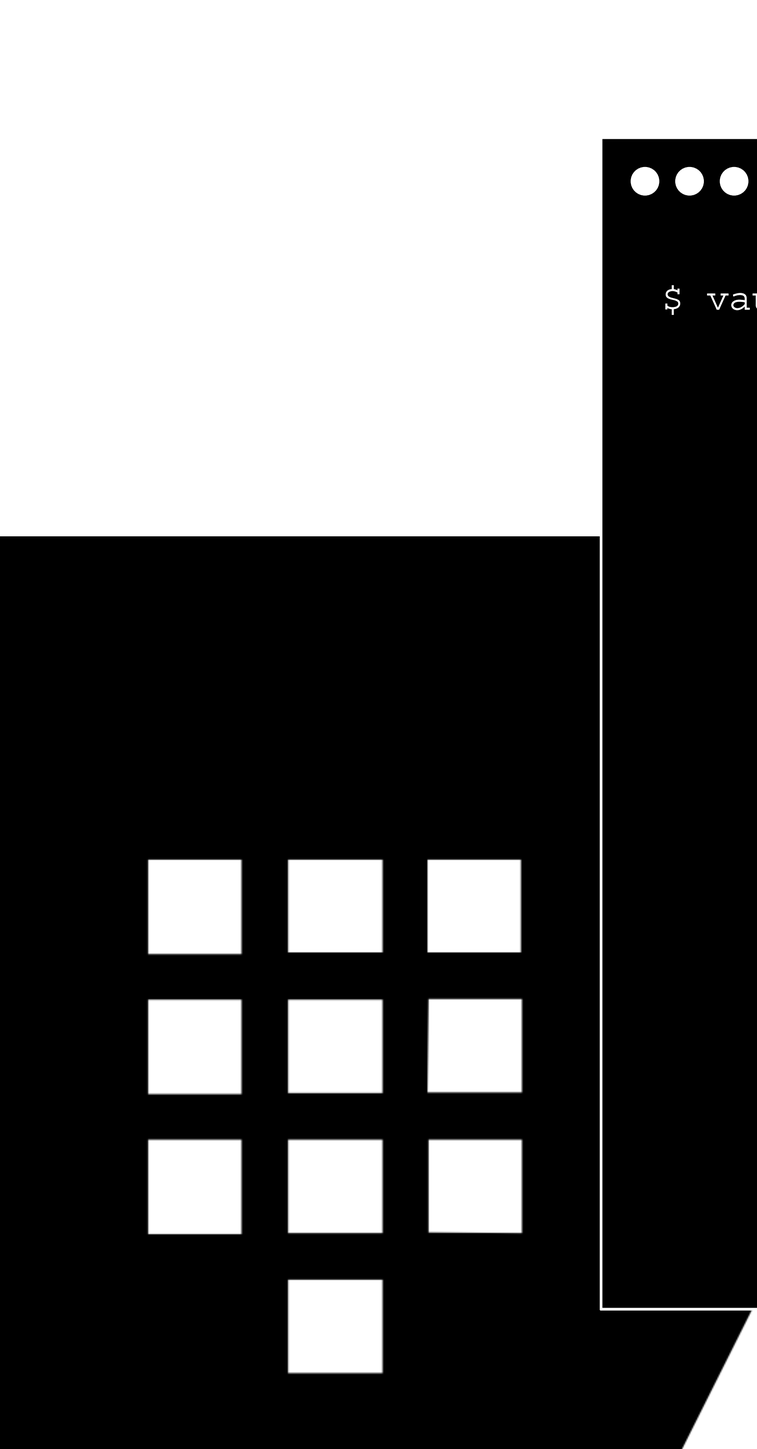


Terminal

```
$ vault auth-enable -path=training-userpass userpass
Successfully enabled 'userpass' at 'training-userpass'!
```

```
$ vault write sys/auth/training-userpass type=userpass
Success! Data written to: sys/auth/training-userpass
```

HashiCorp



Terminal

```
$ vault write auth/userpass/users/sethvargo password=training policies=base
```

HashiCorp


Terminal
$ vault write auth/userpass/users/sethvargo password=training policies=base
Success! Data written to: auth/userpass/users/sethvargo

Terminal

```
$ vault read auth/userpass/users/sethvargo
Key            Value
---            -----
max_ttl        0
policies       base,default
ttl            0
```

HashiCorp

Exercise: Create Auth with Custom Policy

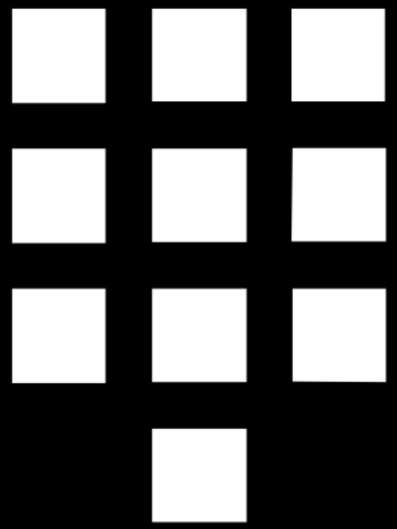
Create a new policy named "contractor" that grants only the ability to generate readonly credentials from the database backend.

Create a new userpass authentication that attaches the above policy. Use the username "sandy" and the password "training".

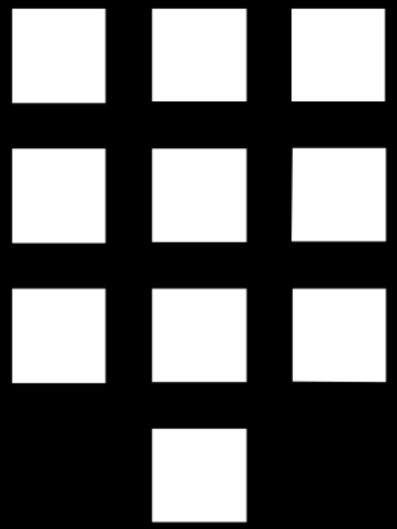
Authenticate as this user and generate a postgresql credential (HINT: vault auth -h)

contractor.hcl

```
path "database/creds/readonly" {
  capabilities = ["read"]
}
```



HashiCorp

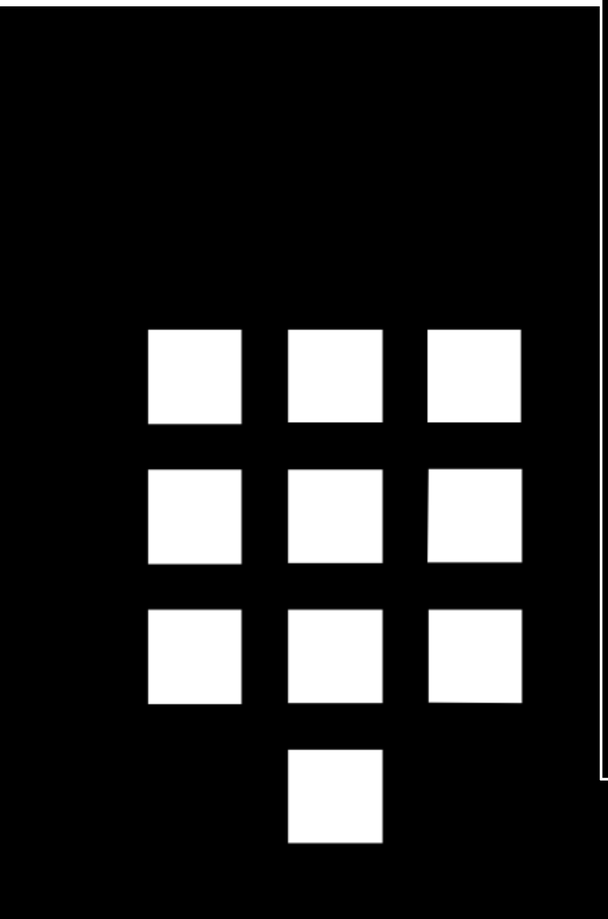


Terminal

```
$ vault policy-write contractor ./contractor.hcl
Policy 'contractor' written.
```

```
$ vault write sys/policy/contractor rules=@contractor.hcl
Success! Data written to: sys/policy/contractor
```

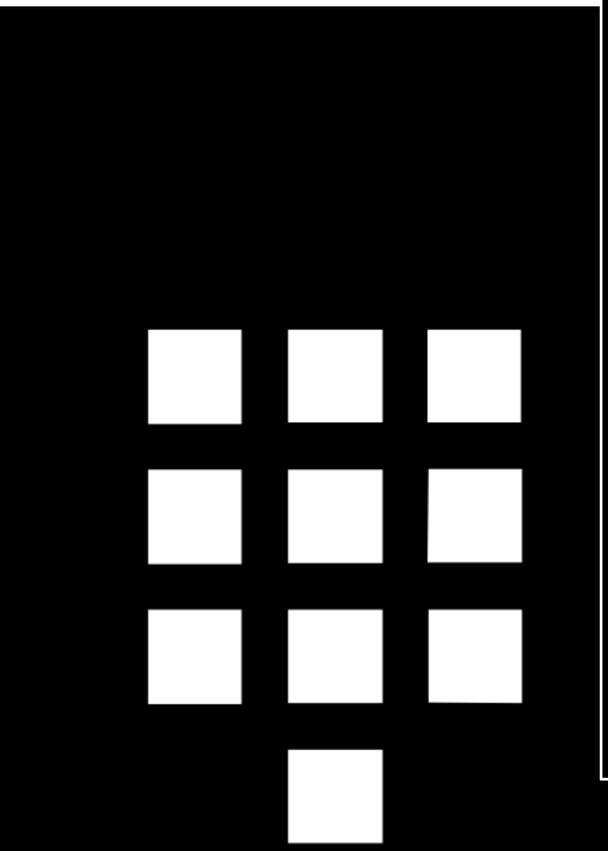
HashiCorp



Terminal

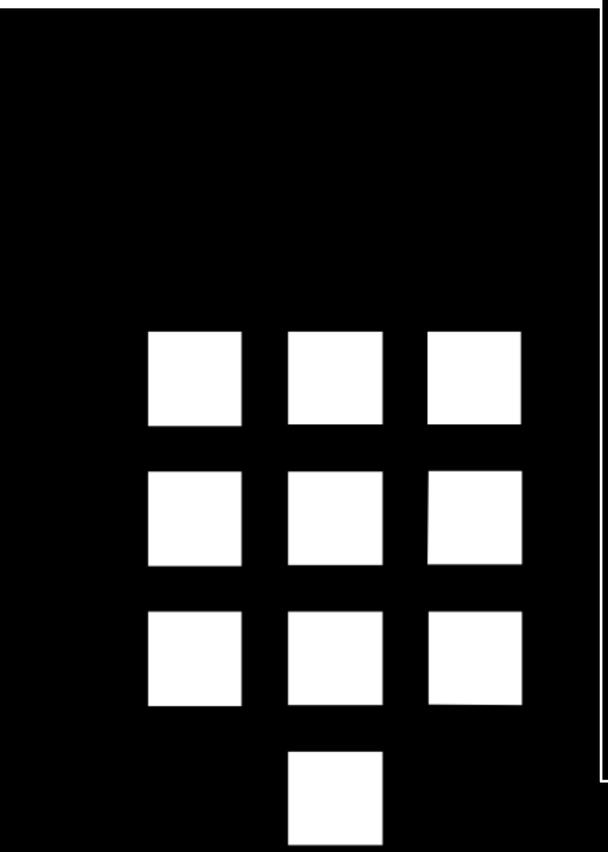
```
$ vault write auth/userpass/users/sandy password=training policies=contractor
Success! Data written to: auth/userpass/users/sandy
```

HashiCorp



Terminal

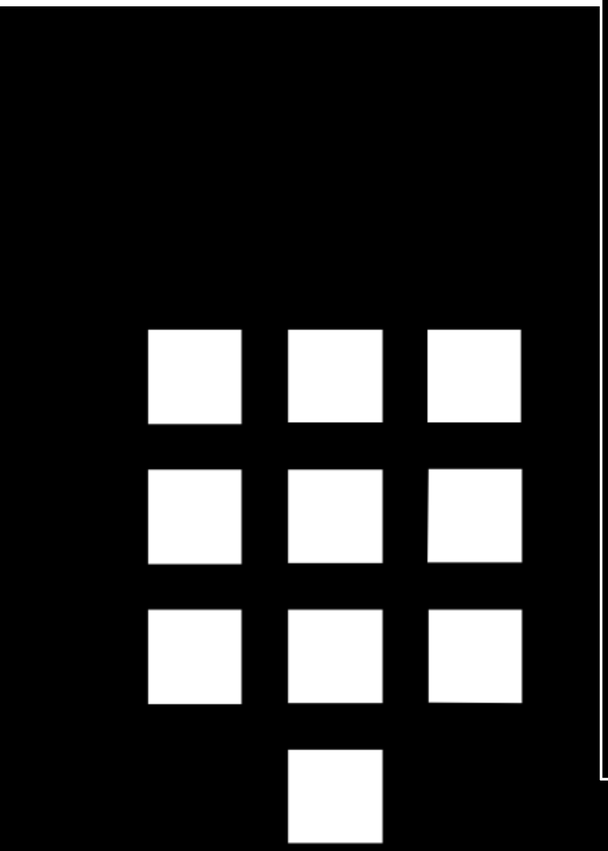
```
$ vault auth -method=userpass username=sandy password=training
Successfully authenticated! You are now logged in.
The token below is already saved in the session. You do not
need to "vault auth" again with the token.
token: ca7999c5-841c-ae7d-6e37-d279d35ecaa2
token_duration: 2591999
token_policies: [contractor default]


$ vault write auth/userpass/login/sandy password=training
Key                  Value
---                  -----
token                fb0522fa-b990-a8c9-1087-2c17ea2b2682
token_accessor       1a8a7e6f-6af8-c7a1-2d03-3d521f5bb3b3
token_duration       768h0m0s
token_renewable      true
token_policies       [contractor default]
token_meta_username "sandy"
```



Terminal

```
$ vault read database/creds/readonly
```

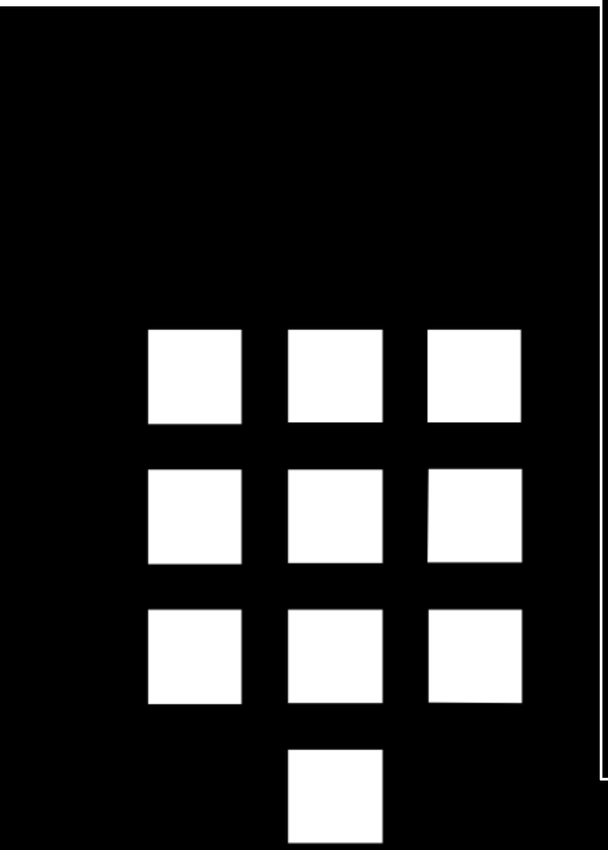
HashiCorp



Terminal

```
$ vault read database/creds/readonly
Key            Value
---            -----
lease_id       database/creds/readonly/1128dff9-ec27-16a6-719c-aababa3b2b7a
lease_duration 1h0m0s
lease_renewabletrue
password       A1a-4zv198vqs4yzqs1p
username       v-userpass-readonly-qw3r12turq2t0v56u1r8-1505188144
```

HashiCorp



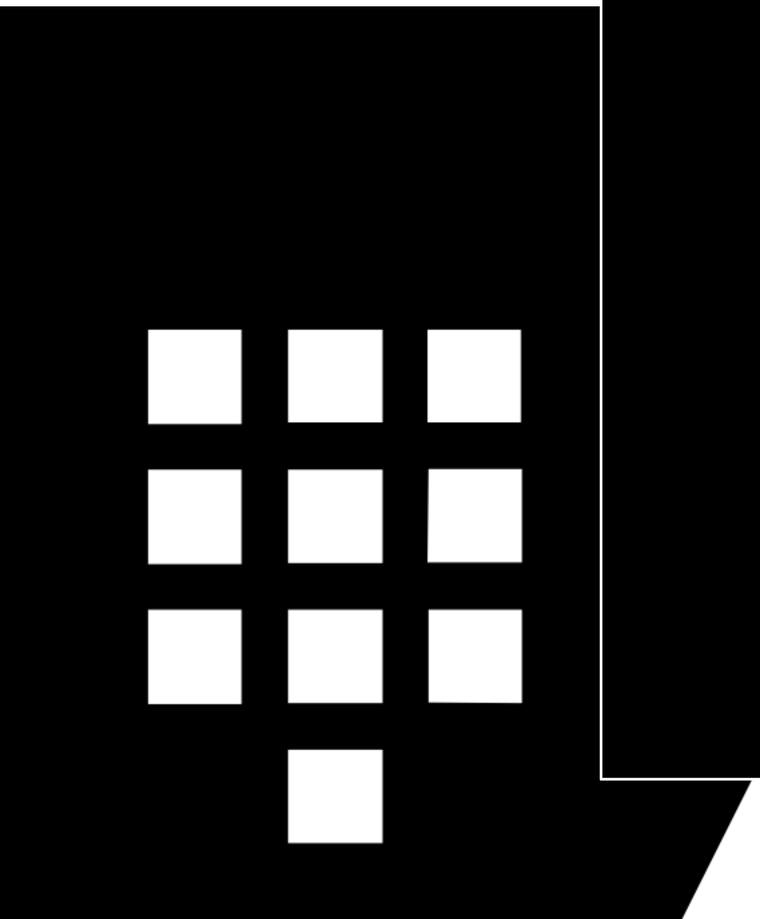
Terminal

```
$ vault write secret/foo bar=1
Error writing data to secret/foo: Error making API request.

URL: PUT http://192.168.50.151:8200/v1/secret/foo
Code: 403. Errors:

* permission denied
```

HashiCorp

Exercise: Auth as yourself

Authenticate as root token.



Terminal

```
$ vault auth root
Successfully authenticated! You are now logged in.
token: root
token_duration: 0
token_policies: [root]
```

HashiCorp

Auditing

About Audit Backends

Audit backends keep a detailed log of all requests and responses to Vault.

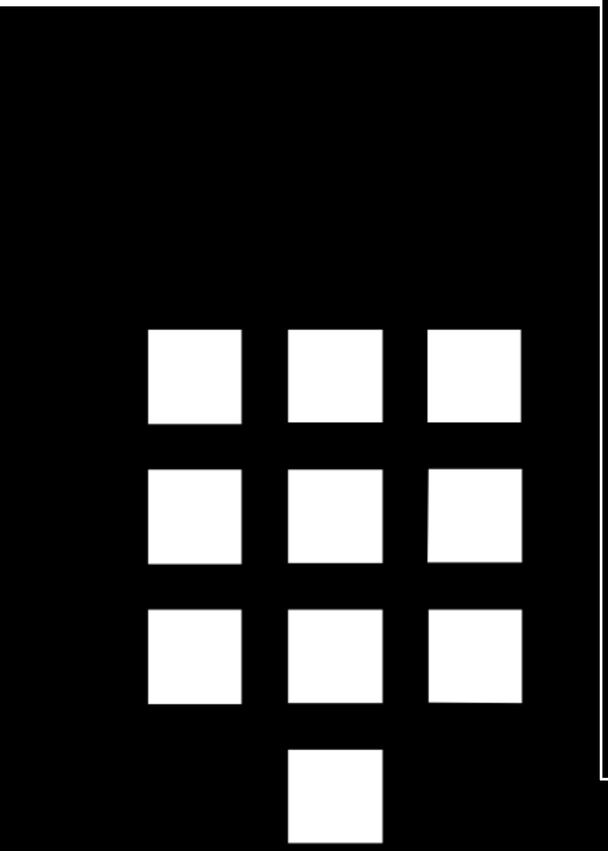
Sensitive information is obfuscated by default (HMAC).

Prioritizes safety over availability.

Exercise: Enable Audit Backend

Enable the "file" audit backend to write to the path `/workstation/vault/audit.log`

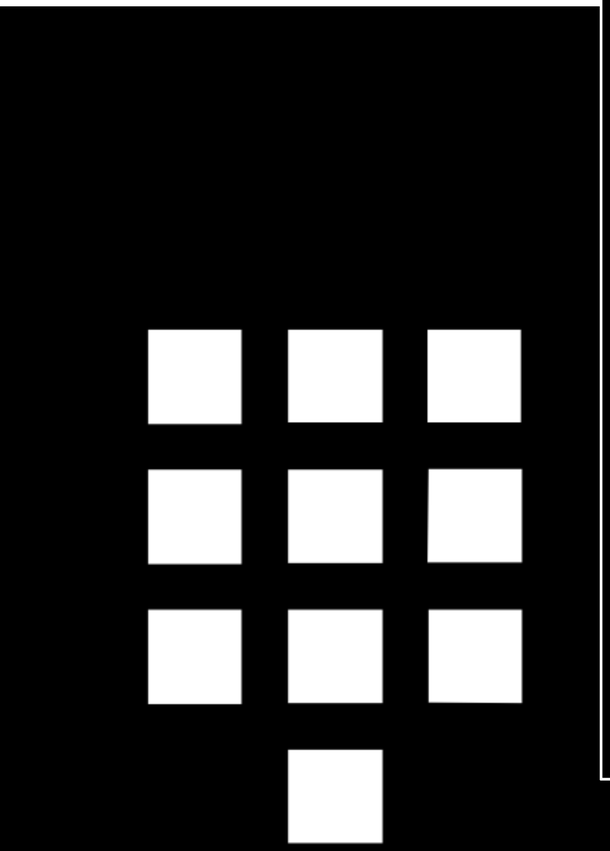
HINT: there are two "paths" - the URL path and the path on disk



Terminal

```
$ vault audit-enable file file_path=/tmp/vaultaudit.log
Successfully enabled audit backend 'file' with path 'file'!
```

HashiCorp



Terminal

```
$ sudo cat vaultaudit.log | jq .
{
  "time": "2017-09-12T03:50:13Z",
  "type": "response",
  "auth": {
    "client_token":
"hmac-sha256:91225458750478b6673a4471d9341ba8d30b2cc28ad1181740f6ada558ecc72c"
,
    "accessor":
"hmac-sha256:90a7fc1e478b88c906aa8864e59bed07ec44ebab37c930c183e61579142cf9b2"
,
    "display_name": "token",
    "policies": [
      "root"
    ],
    "metadata": null
  },
  "request": {
    "id": "154a84d0-c1cd-9f4d-f4ac-2fb60d03cbb5",
# .
```

HashiCorp

Auditing Additional Fields

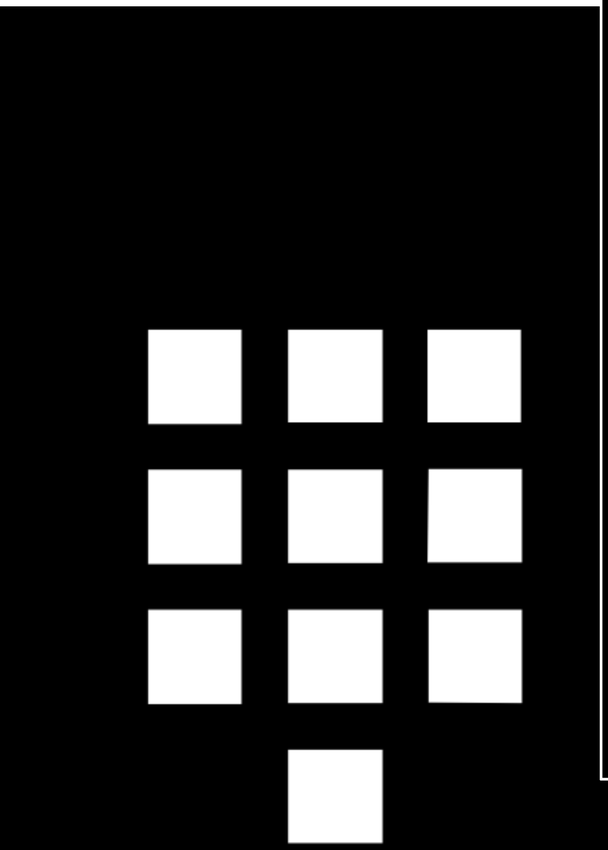
In addition to the standard fields, Vault can optionally audit user-defined headers

Useful for logging things like `X-Forwarded-For`

Exercise: Audit X-Forwarded-For

Configure Vault to audit the `X-Forwarded-For` header.

HINT: API docs for `sys/config`

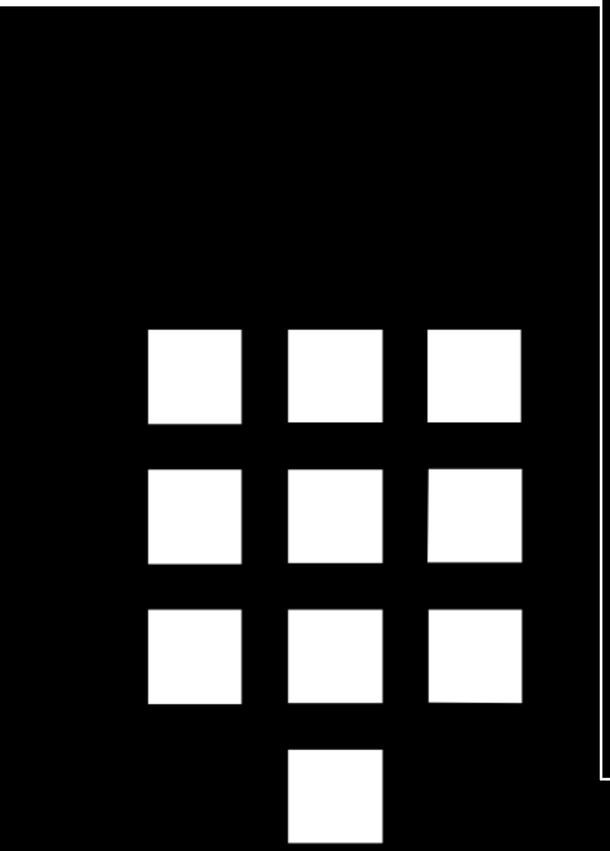


Terminal

```
$ vault write sys/config/auditing/request-headers/X-Forwarded-For hmac=false
Success! Data written to: sys/config/auditing/request-headers/X-Forwarded-For
```

```
$ vault write -f sys/config/auditing/request-headers/X-Forwarded-For
Success! Data written to: sys/config/auditing/request-headers/X-Forwarded-For
```

HashiCorp



Terminal

```
$ curl -H "X-Forwarded-For: hello-world"
http://192.168.50.151:8200/v1/secret/training



$ sudo cat audit.log
{
  "request": {
    "headers": {
      "x-forwarded-for": [
        "hello-world"
      ]
    },

# ...
```

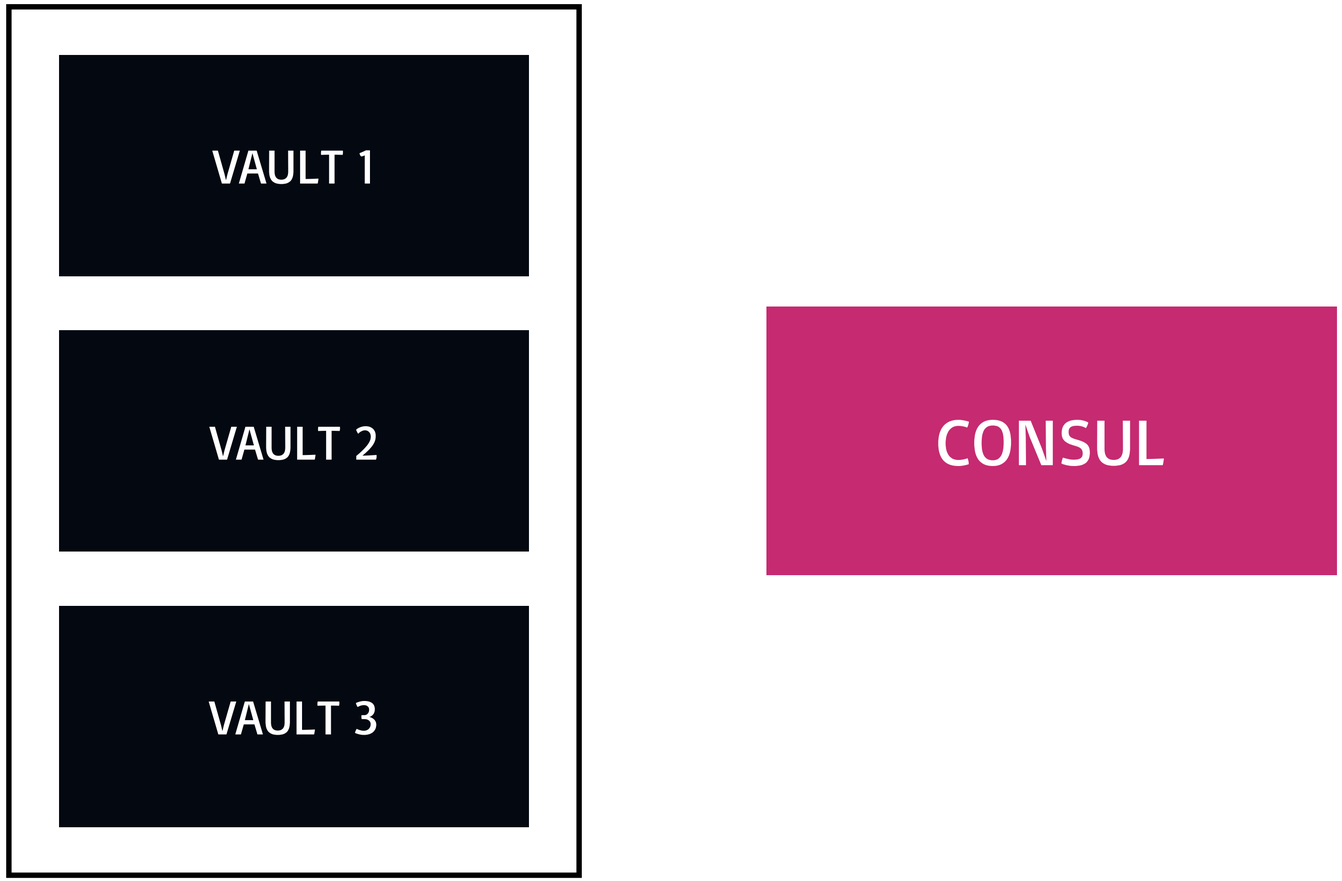
HashiCorp

High Availability

Deploying Vault HA

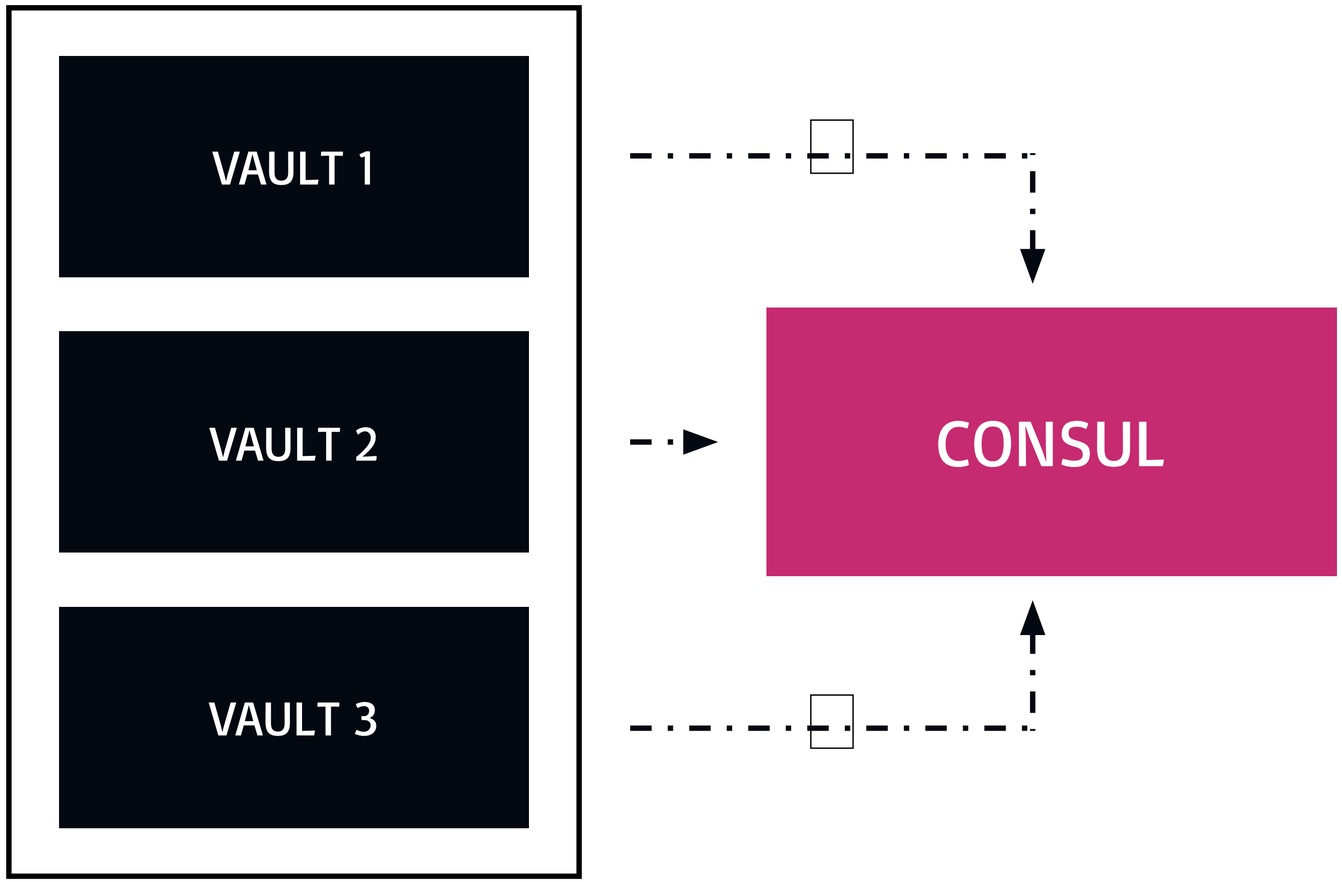
1. Deploy one Vault with an HA storage backend configured
2. Run *vault init* to generate unseal keys and token on first Vault
3. Unseal the Vault
4. Repeat the above steps on the second Vault, except `init`

HA Vault

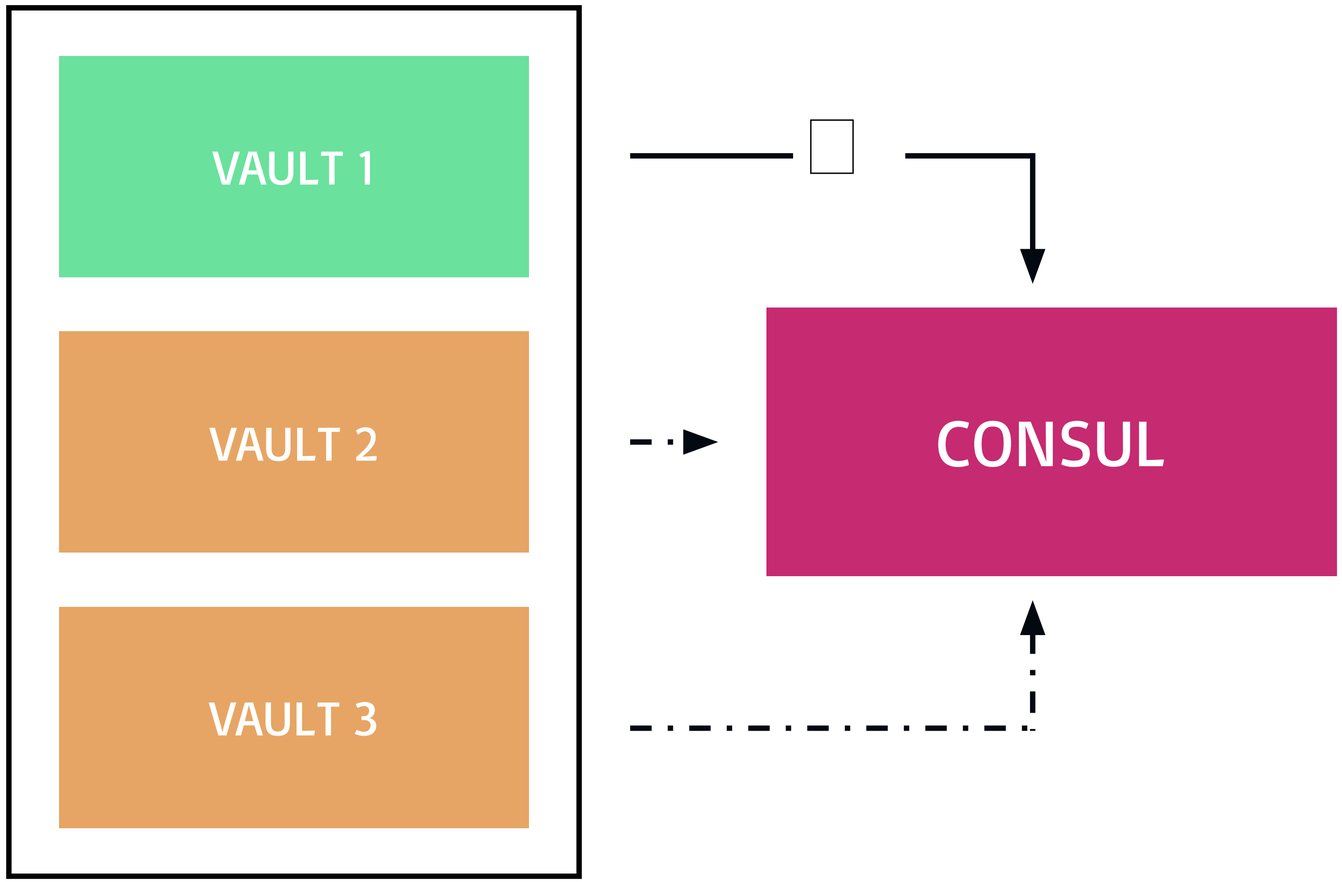


HashiCorp

HA Vault

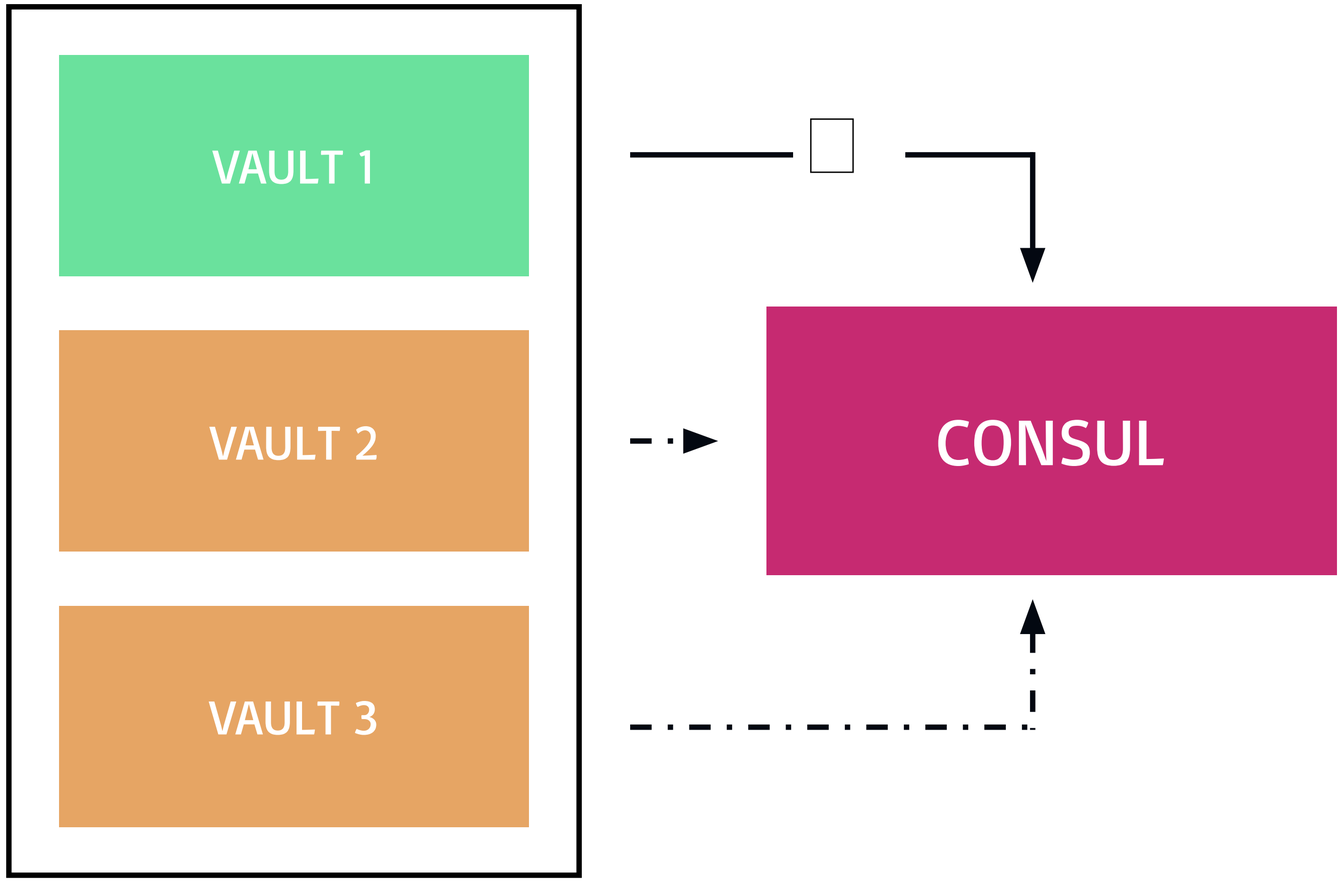


HA Vault



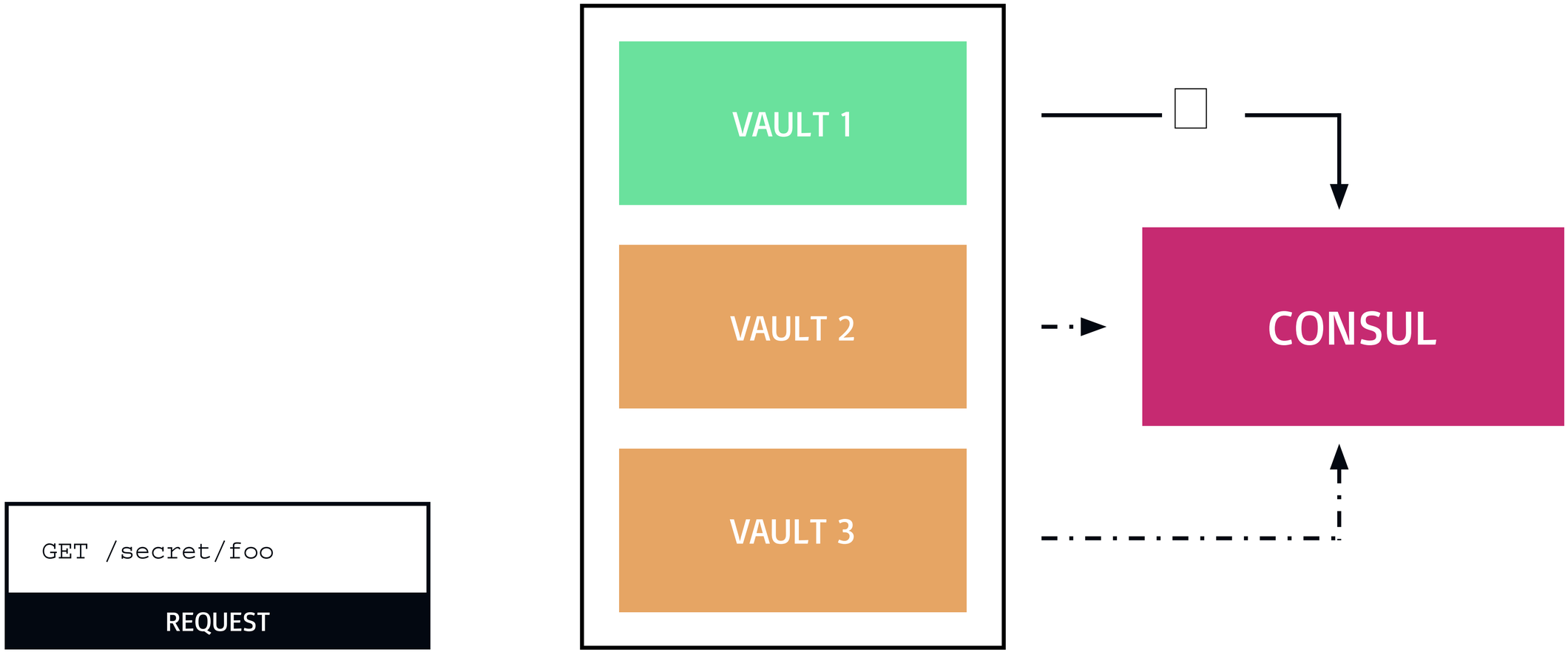
HA Vault

LEADER ELECTION

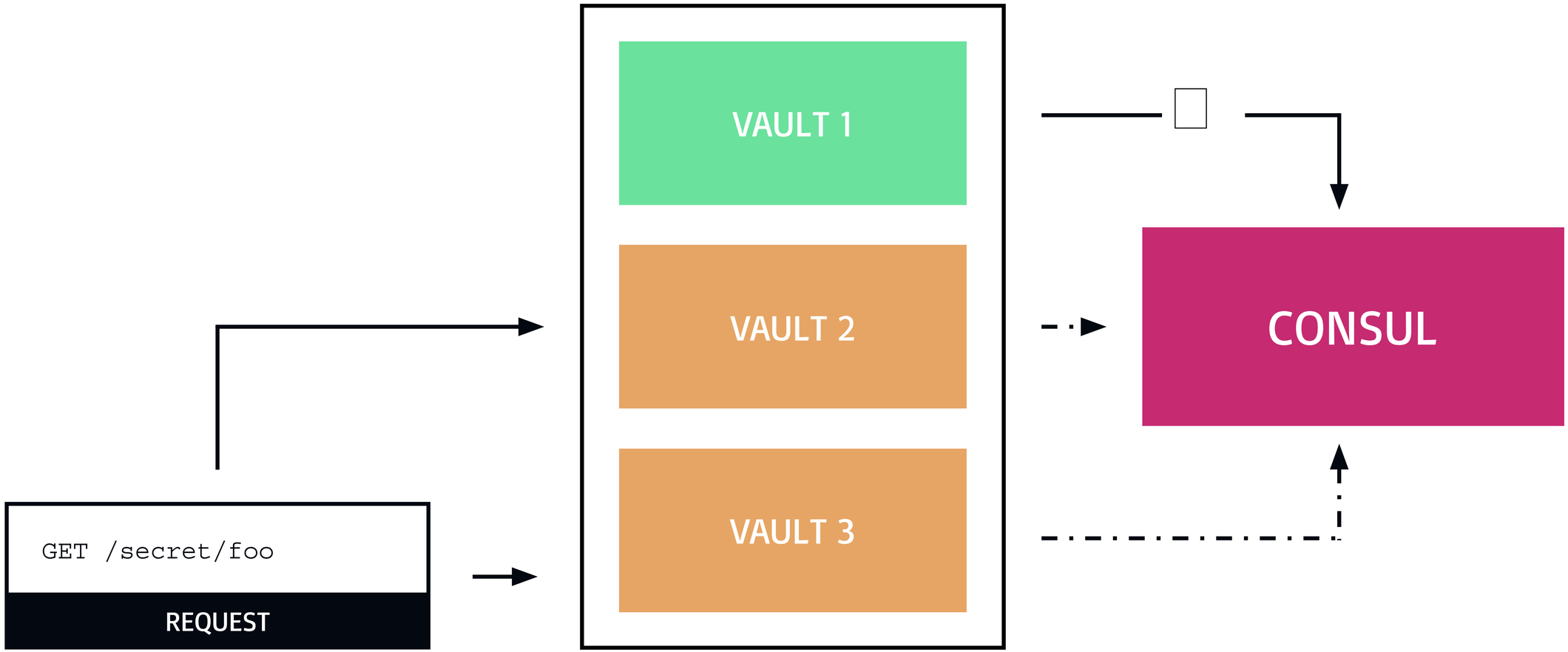


HashiCorp

HA Vault

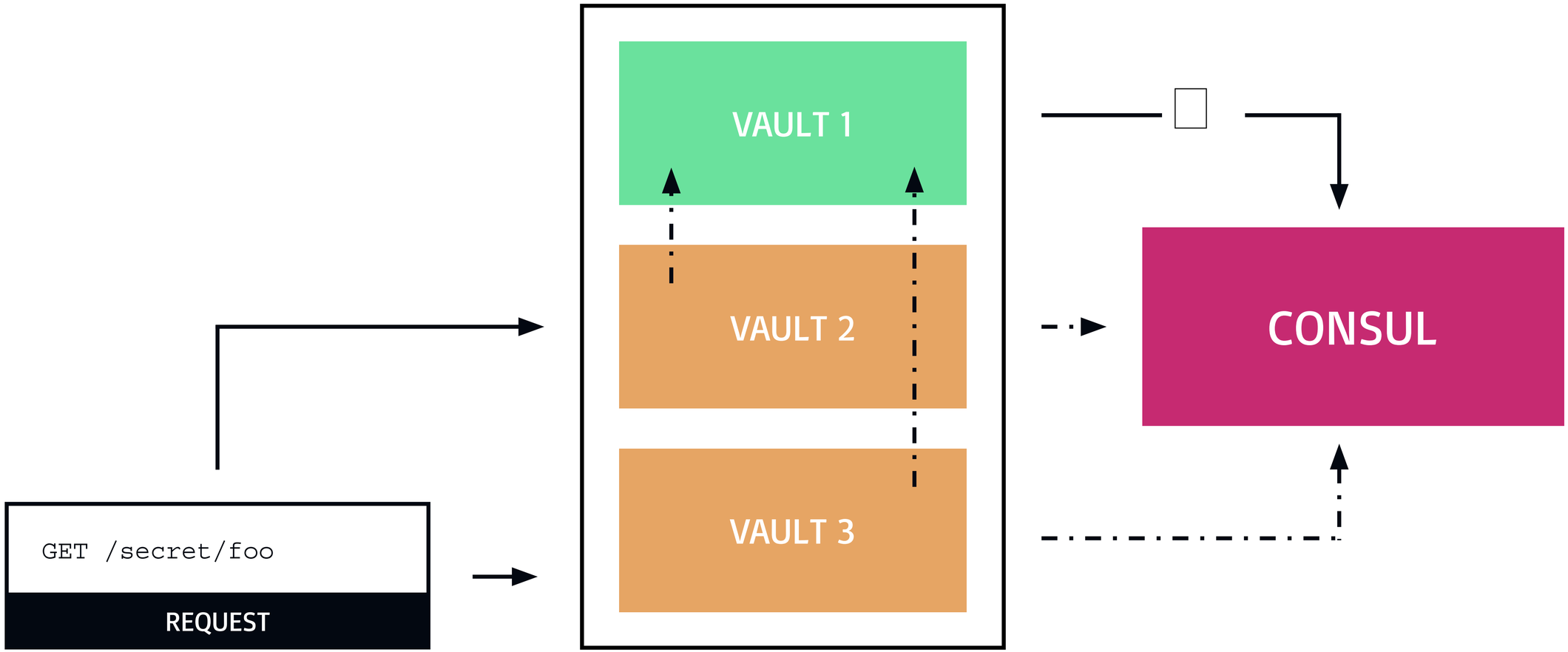


HA Vault

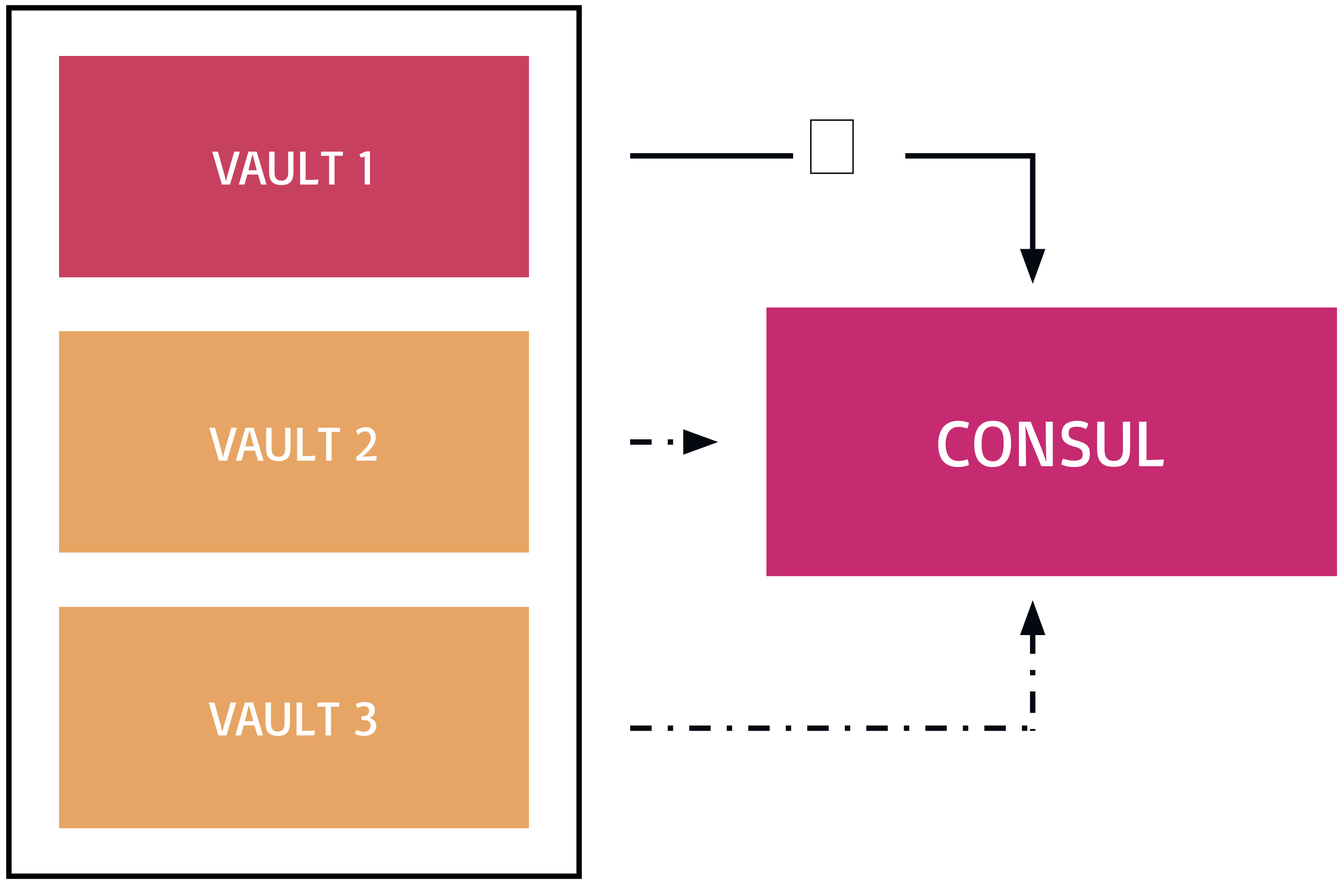


HashiCorp

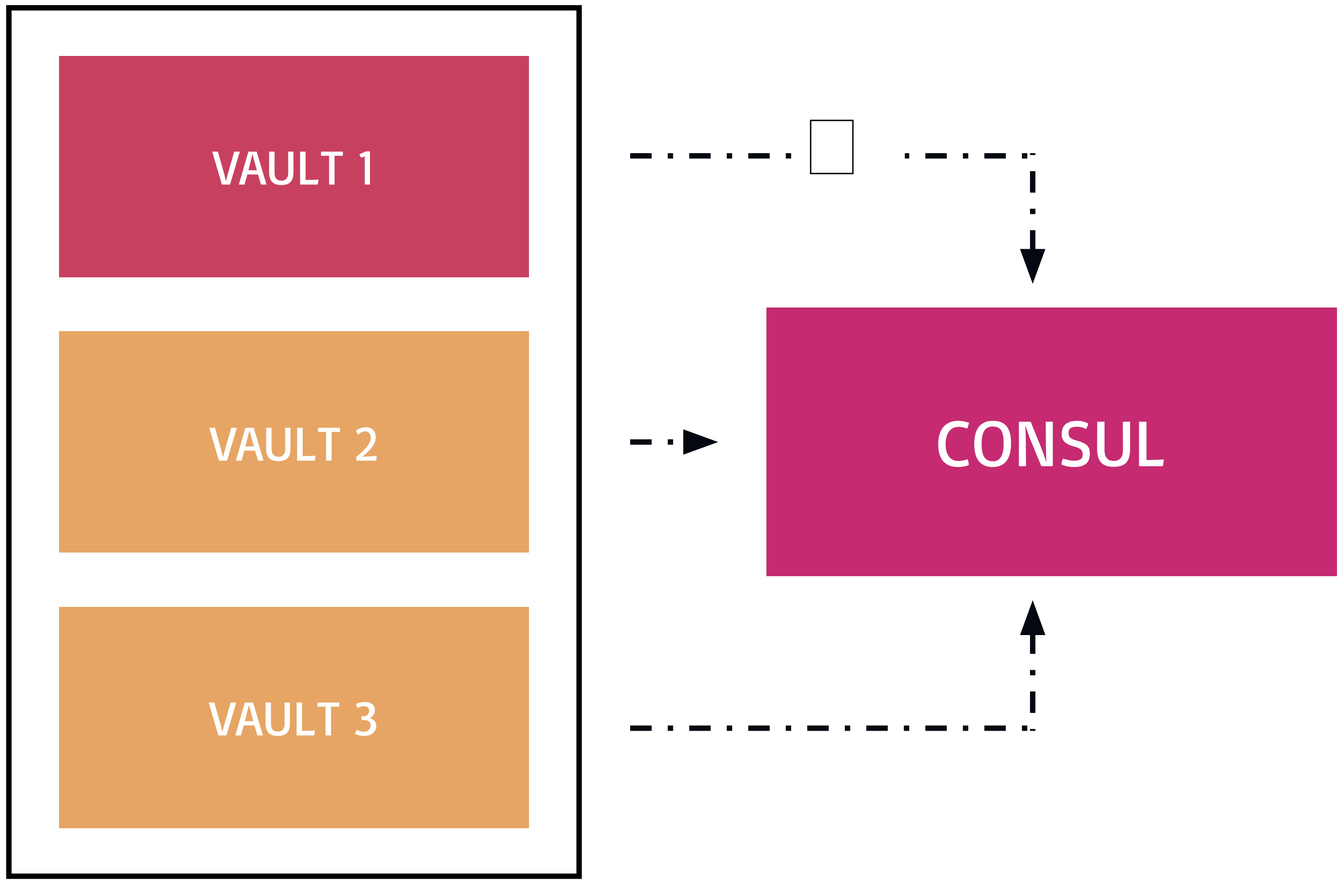
HA Vault



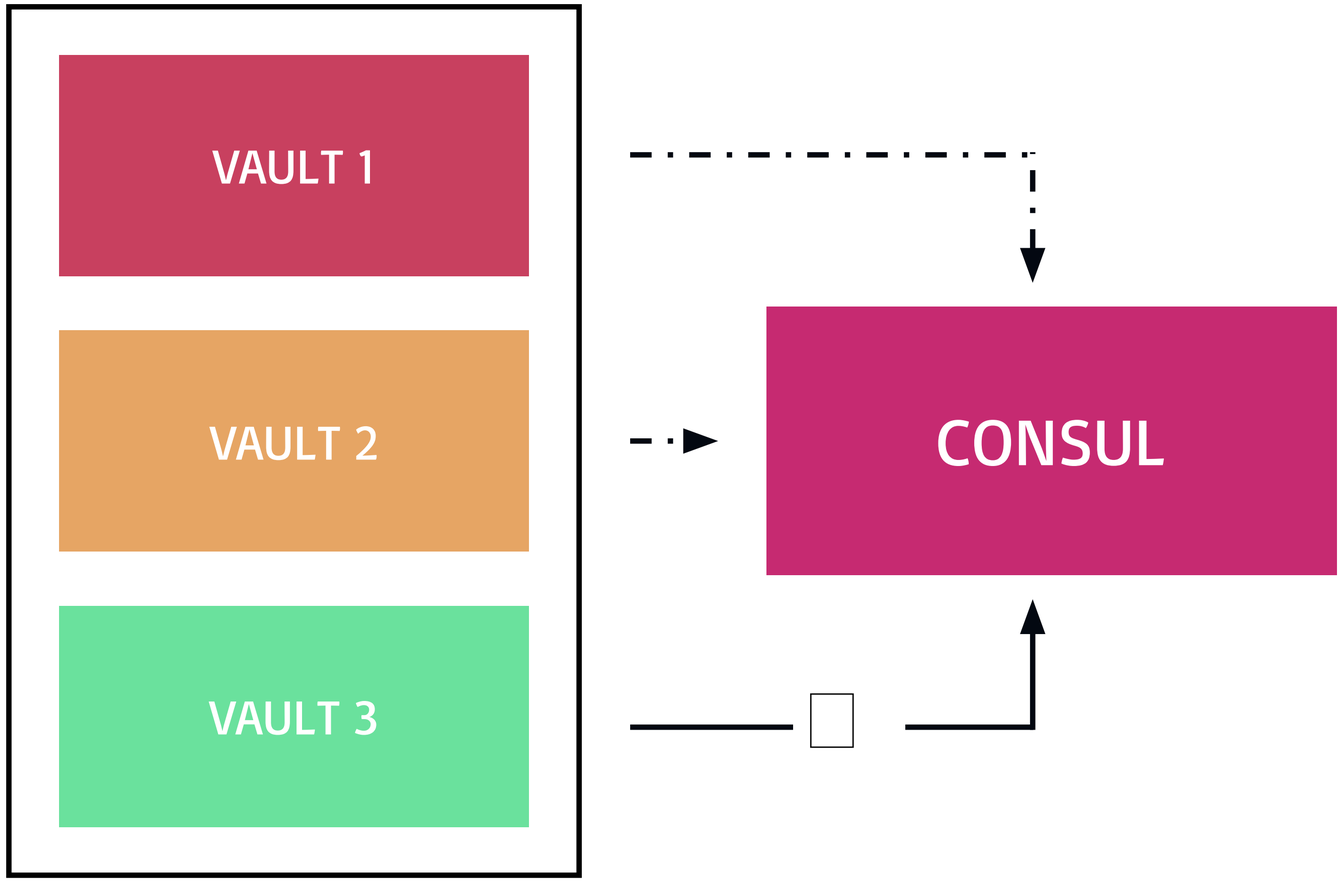
HA Vault



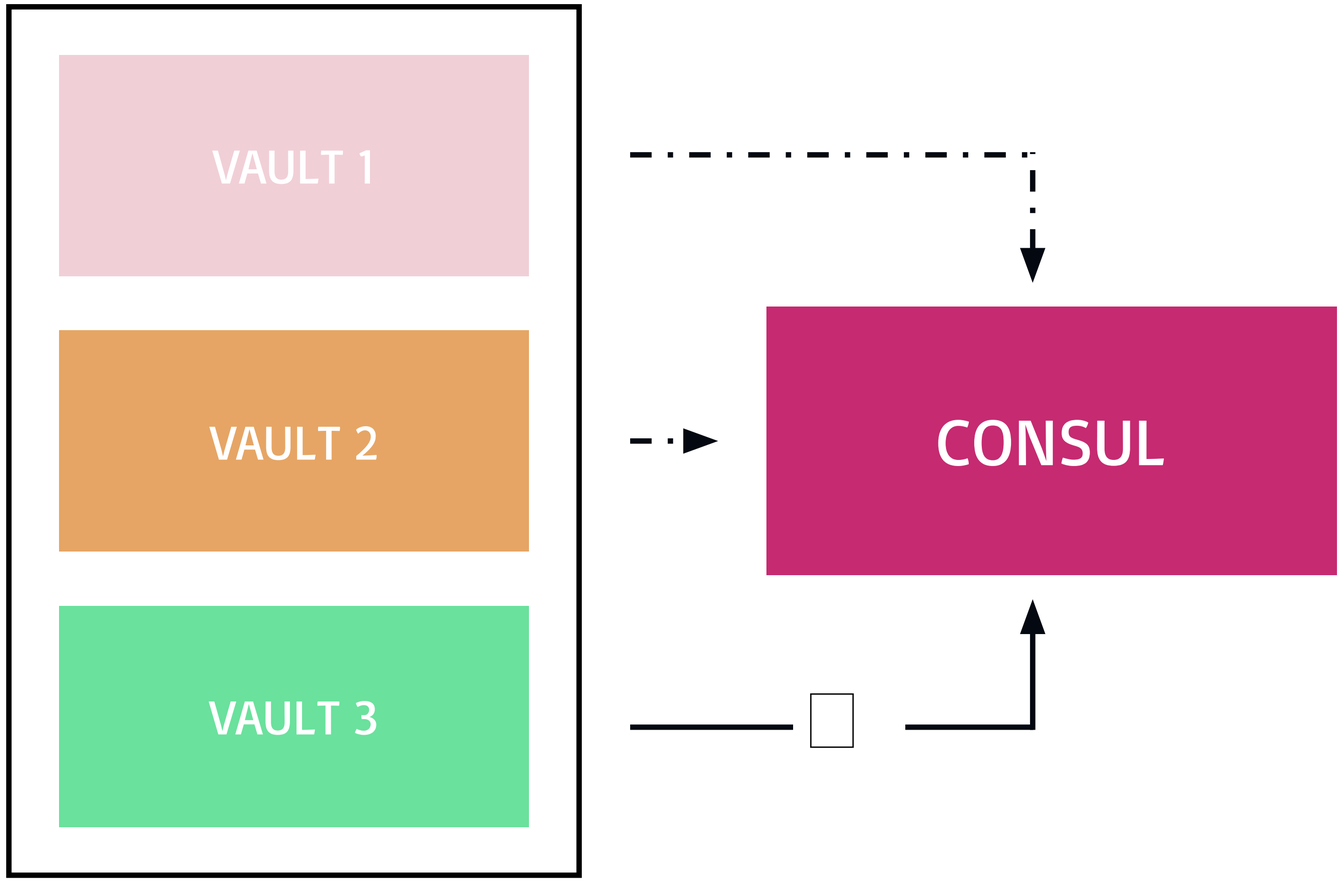
HA Vault



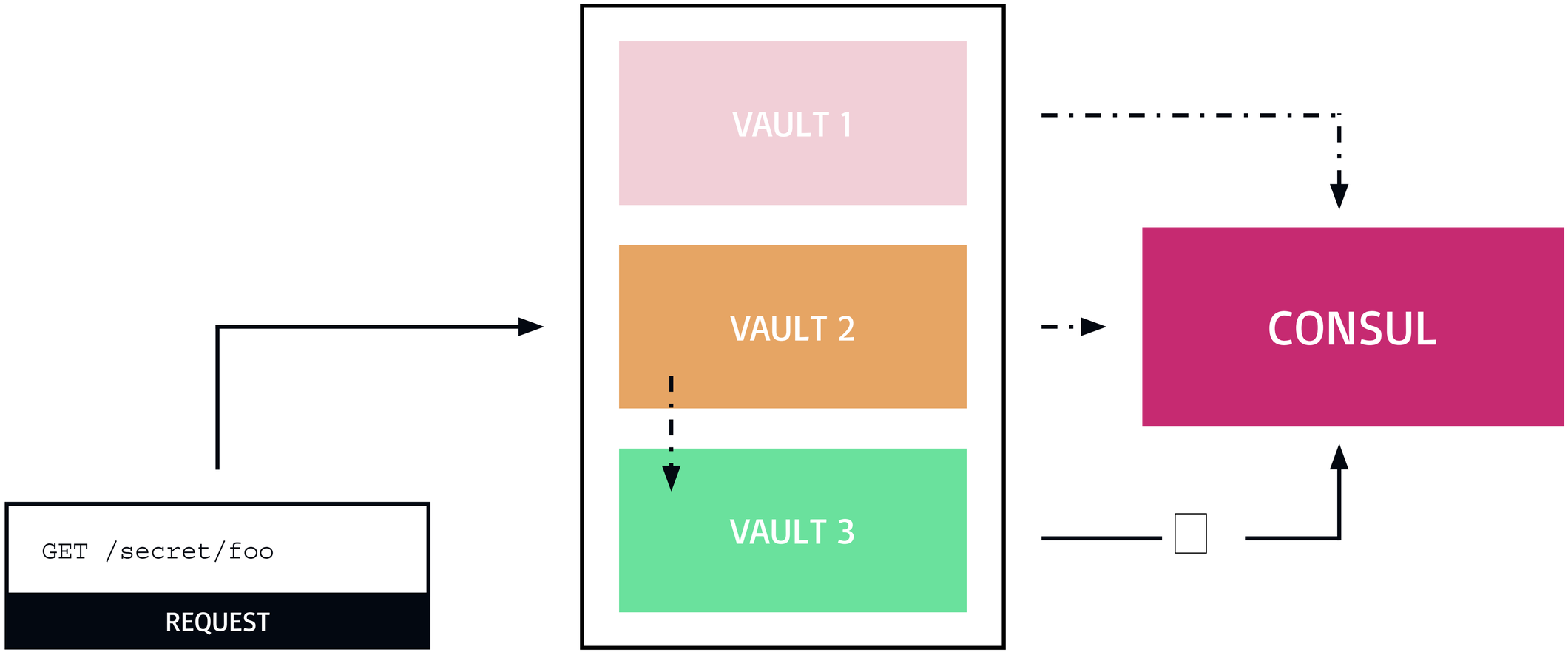
HA Vault



HA Vault



HA Vault



Exercise: Enable HA

Unseal vault on vault2:

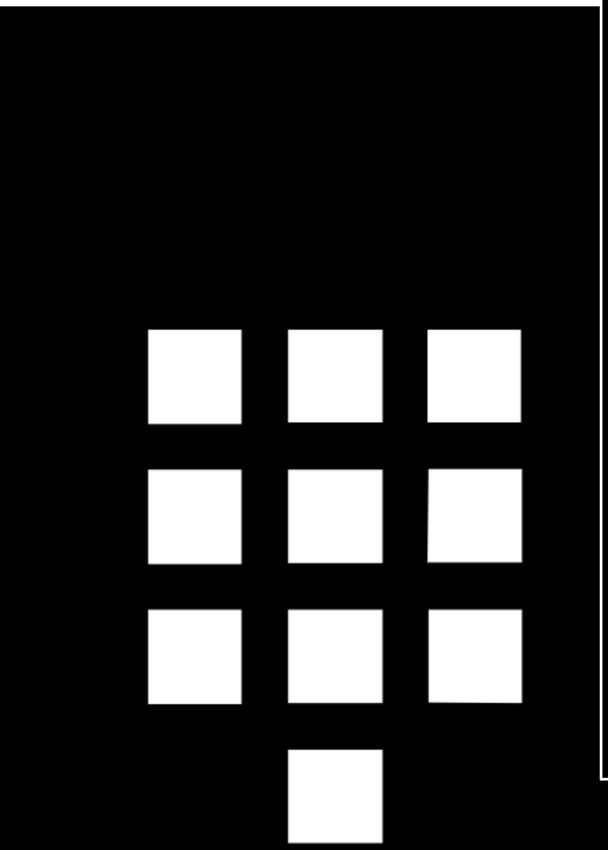
```
export VAULT_ADDR=http://192.168.50.152:8200
vault unseal
```

Unseal vault on vault3:

```
export VAULT_ADDR=http://192.168.50.153:8200
vault unseal
```

View Health Checks on Consul:

```
http://192.168.50.151:8500
```



Terminal

```
$ vault write sys/config/auditing/request-headers/X-Forwarded-For hmac=false
Success! Data written to: sys/config/auditing/request-headers/X-Forwarded-For
```

```
$ vault write -f sys/config/auditing/request-headers/X-Forwarded-For
Success! Data written to: sys/config/auditing/request-headers/X-Forwarded-For
```

HashiCorp

(Re)generating Root

Regenerating the Root Token

In a production Vault installation, the initial root token should only be used for initial configuration.

After a subset of administrators have `sudo` access, almost all operations can be performed.

But for some system-critical operations, a root token may still be required.

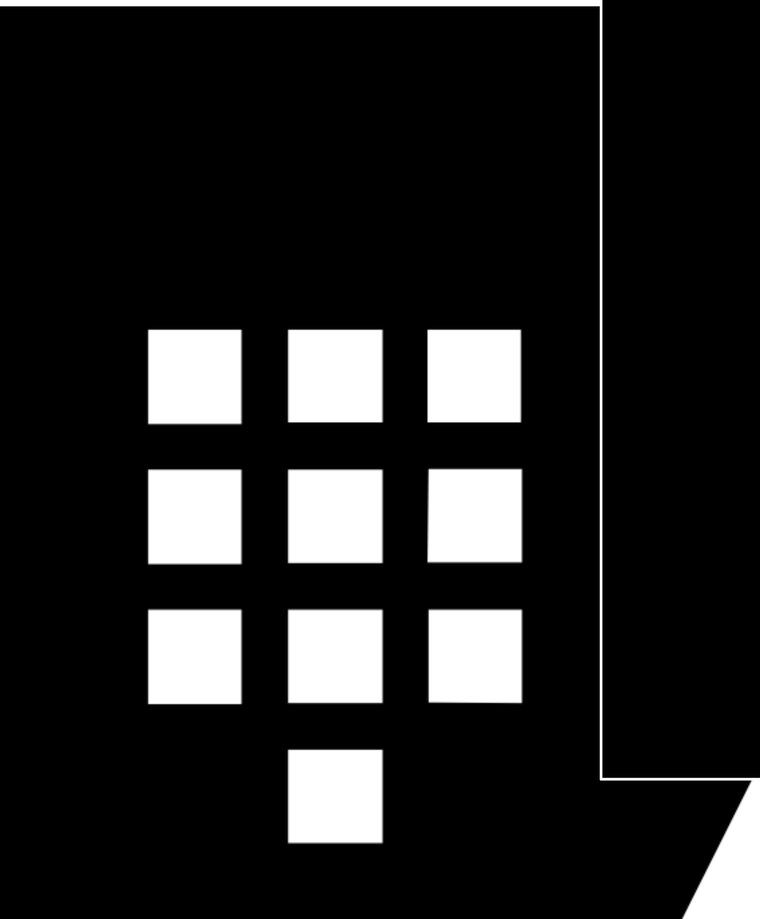
Regenerating the Root Token

A quorum of unseal key holders can generate a new root token.

Enforces the "no one person has complete access to the system".

Steps to Regenerate Root

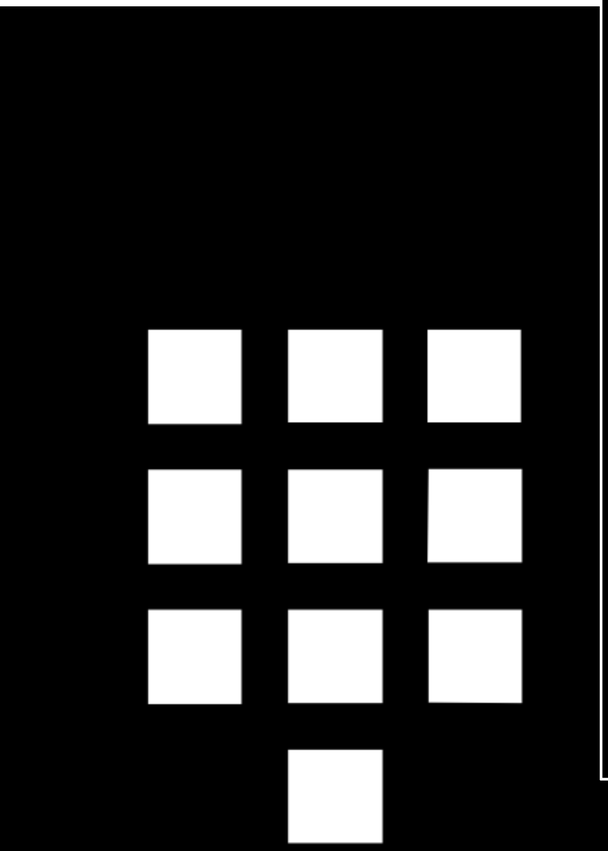
1. Make sure the Vault is unsealed
2. Generate a one-time-password to share
3. Each key-holder runs `generate-root` with the OTP
4. Decode the root token



Terminal

```
$ vault unseal ...
(this is already done)
```

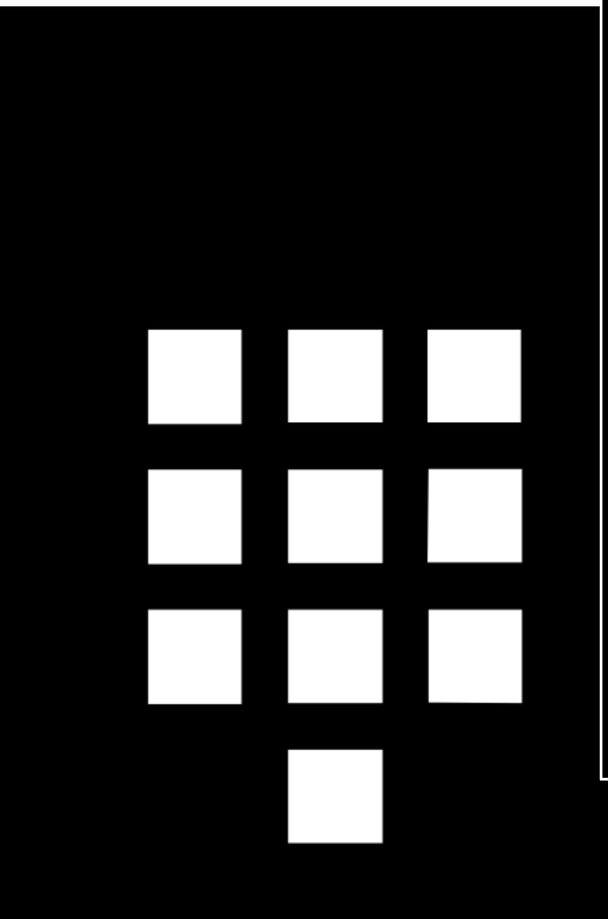
HashiCorp



Terminal

```
$ vault generate-root -genotp
OTP: PddHKv/HqfP9OGddpiY69Q==
```

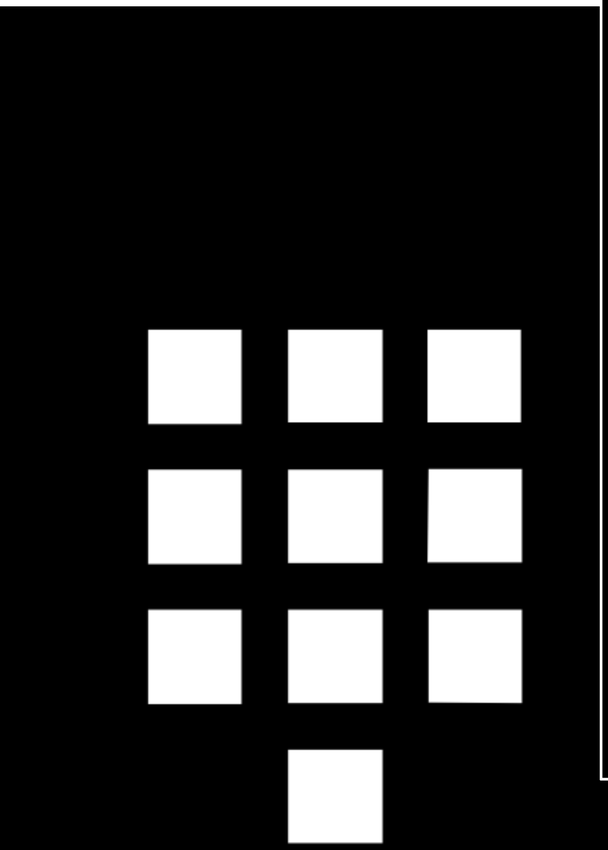
HashiCorp



Terminal

```
$ vault generate-root -otp="PddHKv/HqfP9OGddpiY69Q=="
Root generation operation nonce: 3edbf635-0581-56ca-5f5c-143402252b7b
Key (will be hidden):
```

HashiCorp



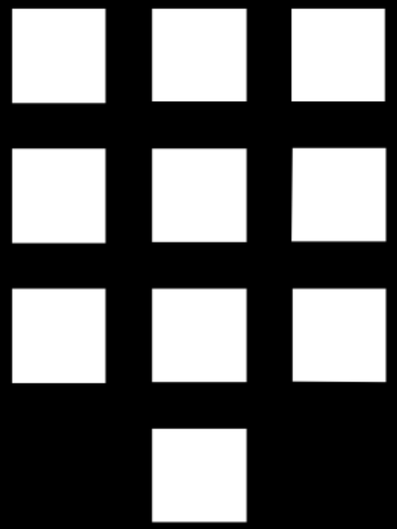
Terminal

```
$ vault generate-root -otp="PddHKv/HqfP9OGddpiY69Q=="
Root generation operation nonce: 3edbf635-0581-56ca-5f5c-143402252b7b
Key (will be hidden):
Nonce: 3edbf635-0581-56ca-5f5c-143402252b7b
Started: true
Generate Root Progress: 1
Required Keys: 1
Complete: true

Encoded root token: B6pCxMFVOhQ28BWmgwHNhQ==
```

HashiCorp

Terminal

```
$ vault generate-root -otp="PddHKv/HqfP9OGddpiY69Q==" \
    -decode="B6pCxMFVOhQ28BWmgwHNhQ=="

Root token: 3a7d05ee-3e92-93e7-cbc8-72fb2527f770
```



HashiCorp

Exercise: Generate New Root Token

Generate a new root token for the initial Vault server (not vault-2).

HINT: Find the unseal key in `sudo journalctl -u vault-2`

HTTP API

About the HTTP API

All interactions with Vault happen via the HTTP API

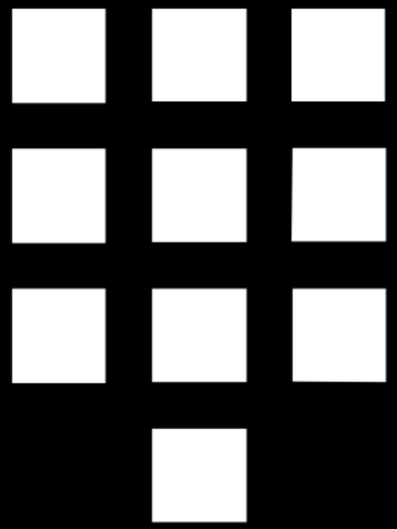
Even the CLI uses the HTTP API – there is nothing special

Auth is passed via the X-Vault-Token header unless authing

Multiple client libraries exist (Go, Ruby, Python, Node, etc)

HTTP API Status Codes

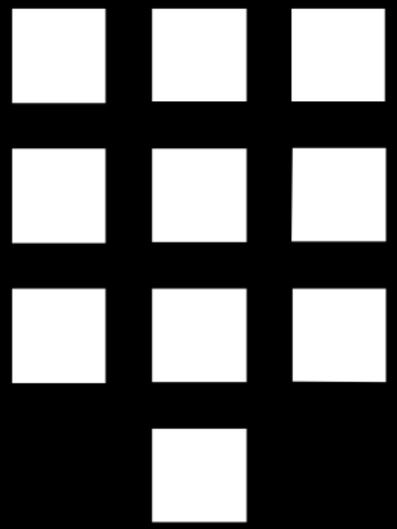
- 200/204 - Success (no data)
- 400 - Invalid request
- 403 - Forbidden
- 404 - Invalid path
- 429 - Rate limit exceeded
- 500 - Internal server error
- 503 - Vault is sealed or in maintenance



Terminal

```
$ vault read secret/training
# ...

$ curl $VAULT_ADDR/v1/secret/training \
  --request GET \
  --header "X-Vault-Token: d9213f90-f569-adae-663f-eb6668403aed"
{
  "auth": null,
  "warnings": null,
  "data": {
    "name": "seth",
    "food": "chicken fingers"
  },
  "lease_duration": 2592000,
  "renewable": false,
  "lease_id": ""
}
```

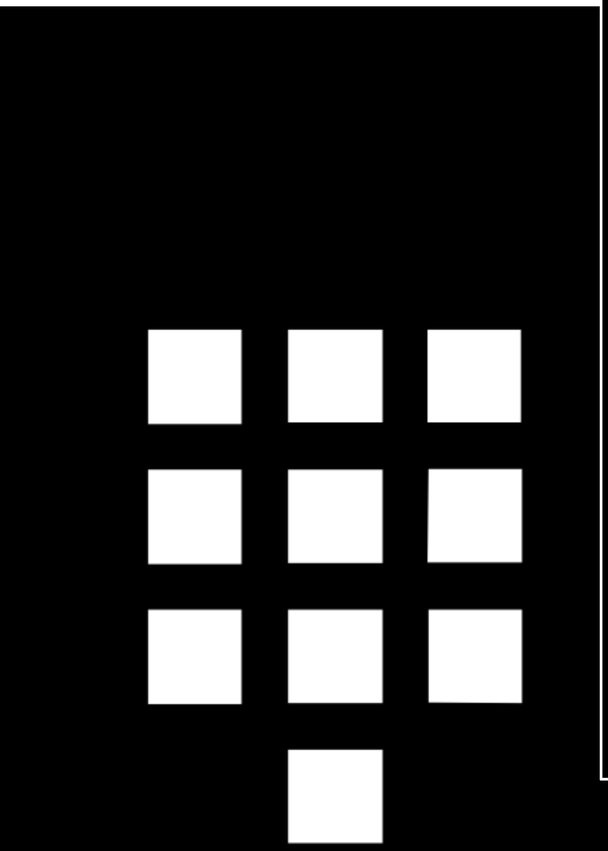
HashiCorp



Terminal

```
$ vault list secret/
# ...

$ curl $VAULT_ADDR/v1/secret \
  --request LIST \
  --header "X-Vault-Token: d9213f90-f569-adae-663f-eb6668403aed"
{
  "auth": null,
  "warnings": null,
  "data": {
    "keys": [
      "foo",
      "training"
    ]
  },
  "lease_duration": 0,
  "renewable": false,
  "lease_id": ""
```

HashiCorp



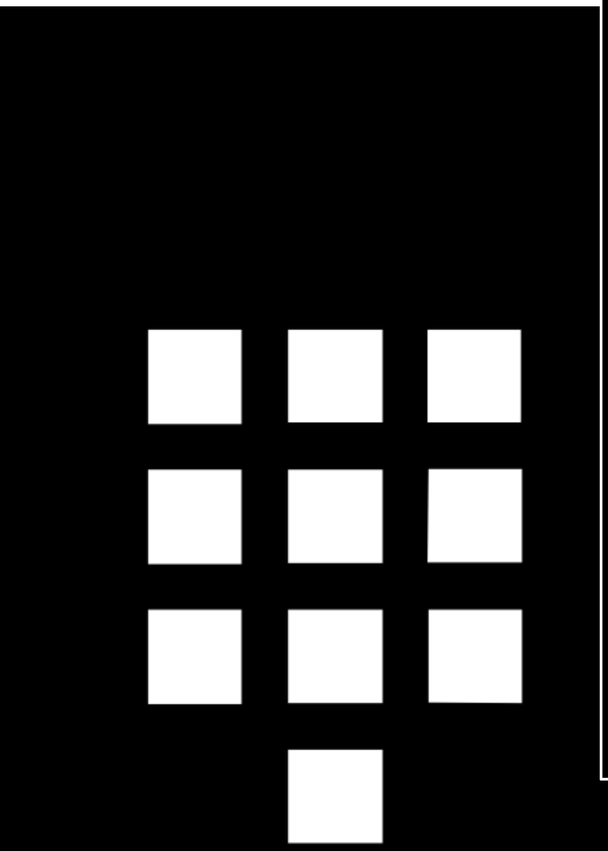
Terminal

```
$ vault write secret/foo bar=1
# ...

$ curl $VAULT_ADDR/v1/secret/foo \
  --request POST \
  --header "X-Vault-Token: d9213f90-f569-adae-663f-eb6668403aed" \
  --data '{"bar":"1"}'
```

HashiCorp

Exercise: Use HTTP API

Retrieve a new set of readonly credentials from the postgres backend using the HTTP API.



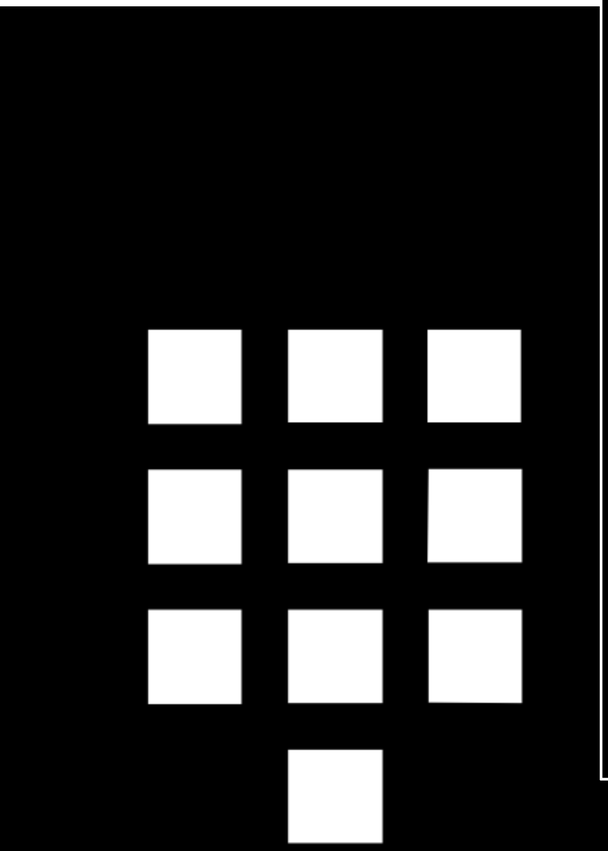
Terminal

```
$ curl $VAULT_ADDR/v1/postgresql/creds/readonly \
  --request GET \
  --header "X-Vault-Token: root"
{
  "auth": null,
  "warnings": null,
  "wrap_info": null,
  "data": {
    "username": "token-eb0376e4-c6e0-2de4-0692-21fb7f93334d",
    "password": "ec139929-4b0f-51ac-6bf1-25fc8ff7a7a9"
  },
  "lease_duration": 3600,
  "renewable": true,
  "lease_id":
"postgresql/creds/readonly/6d0b6607-a472-c2a7-e933-878815a451d8",
  "request_id": "87964c9d-b636-91c2-3d7e-2b4984cca14b"
}
```

HashiCorp

Exercise: Use HTTP API

Renew the lease you just created, using the HTTP API (HINT: `sys`).



Terminal

```
$ curl
$VAULT_ADDR/v1/sys/renew/postgresql/creds/readonly/6d0b6607-a472-c2a7-e933-878
815a451d8 \
  --request POST \
  --header "X-Vault-Token: root"
{
  "auth": null,
  "warnings": null,
  "wrap_info": null,
  "data": null,
  "lease_duration": 3600,
  "renewable": true,
  "lease_id":
"postgresql/creds/readonly/6d0b6607-a472-c2a7-e933-878815a451d8",
  "request_id": "fd394f9d-991a-1f78-6e07-44e97507b9ef"
}
```

HashiCorp

Exercise: Authenticate Using the HTTP API

Authenticate as a contractor with userpass using the HTTP API.

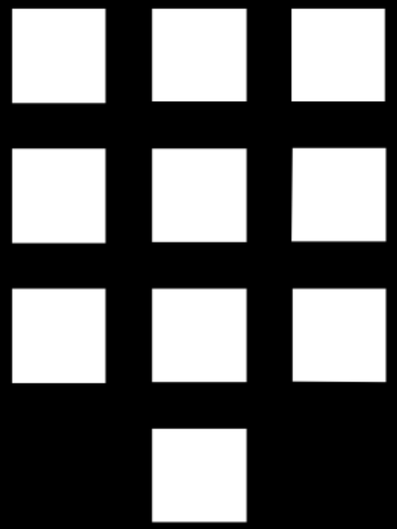
Recall that the credentials are:

Username: `sandy`
Password: `training`

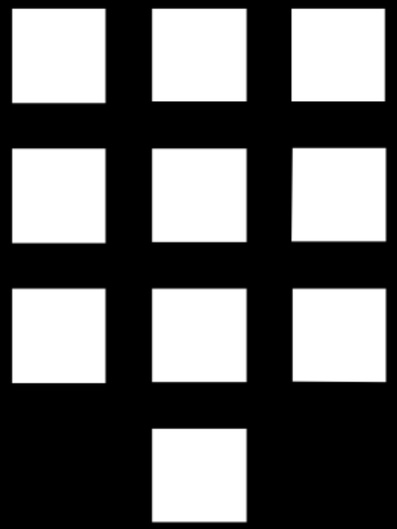
HINT: You may need to look at the API documentation

Terminal

```
$ curl $VAULT_ADDR/v1/auth/userpass/login/sandy \
  --request POST \
  --data '{"password":"training"}'
```



HashiCorp



Terminal

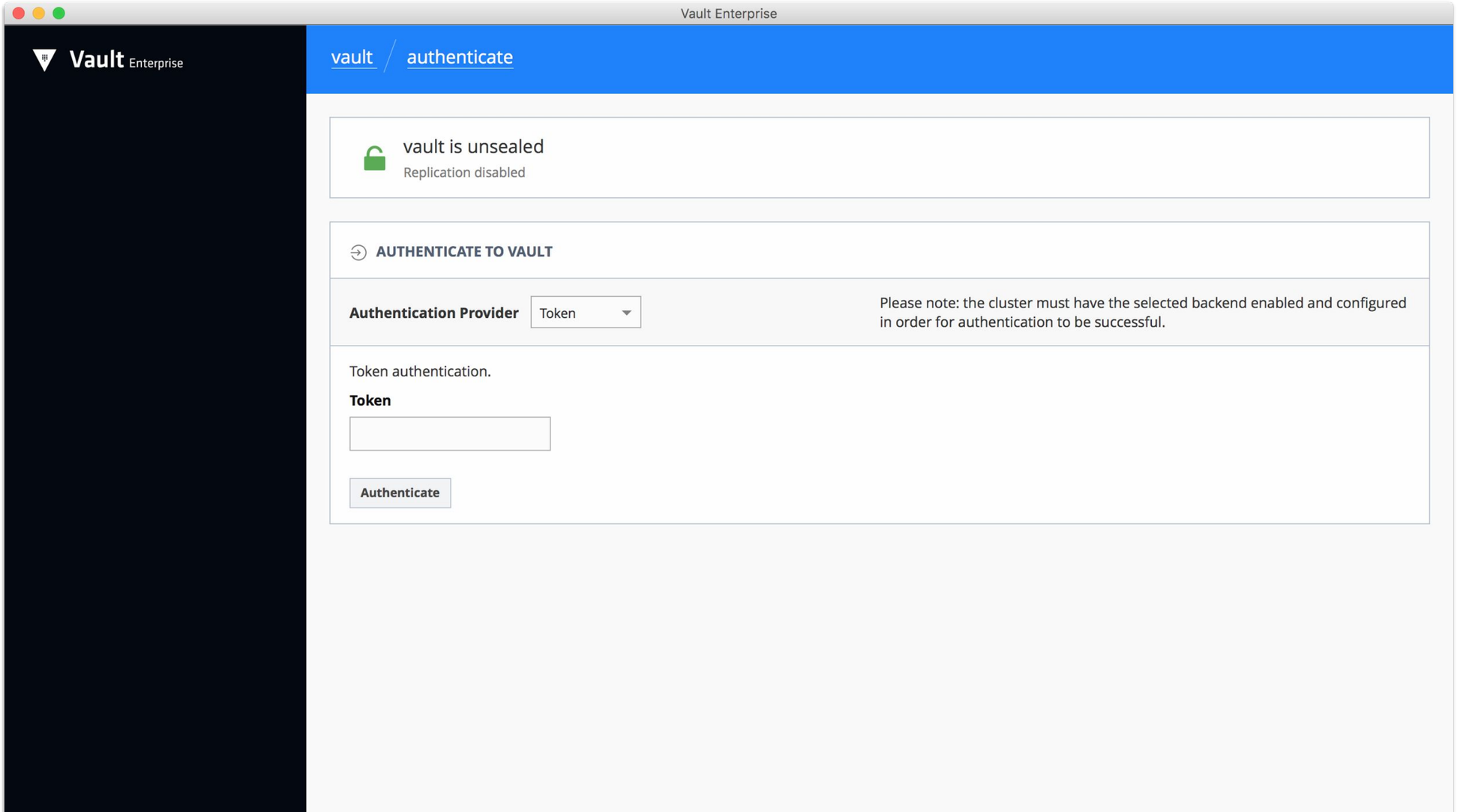

```
{
  "auth": {
    "renewable": true,
    "lease_duration": 2592000,
    "metadata": {
      "username": "sandy"
    },
    "policies": [
      "contractor",
      "default"
    ],
    "accessor": "7abdc853-f43f-57ba-628a-e0f31436f6ab",
    "client_token": "e6187881-c0ff-f772-c4cd-6ccbac161e33"
  },
  "warnings": null,
  "wrap_info": null,
  "data": null,
```


HashiCorp

Vault

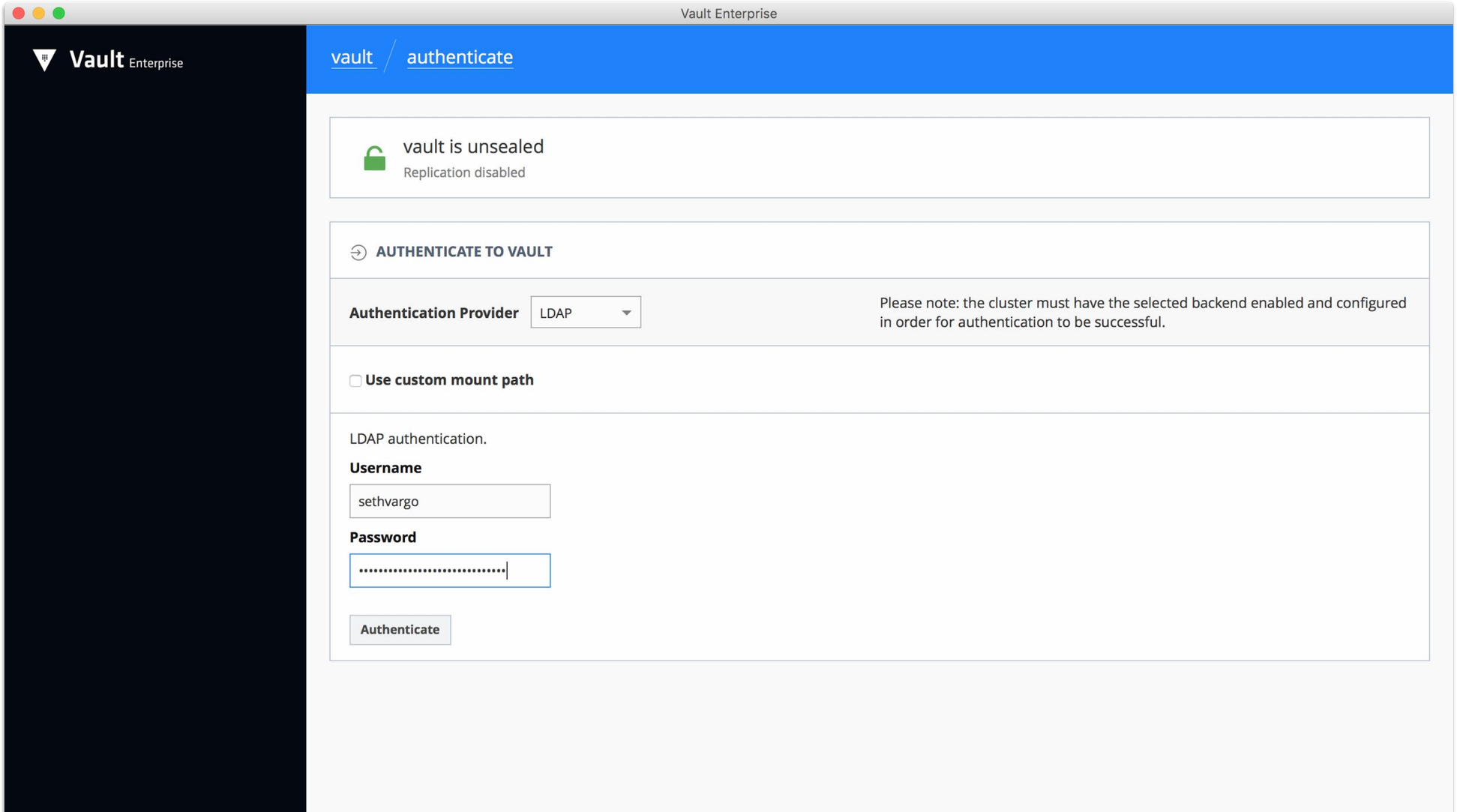
ENTERPRISE

Vault Enterprise



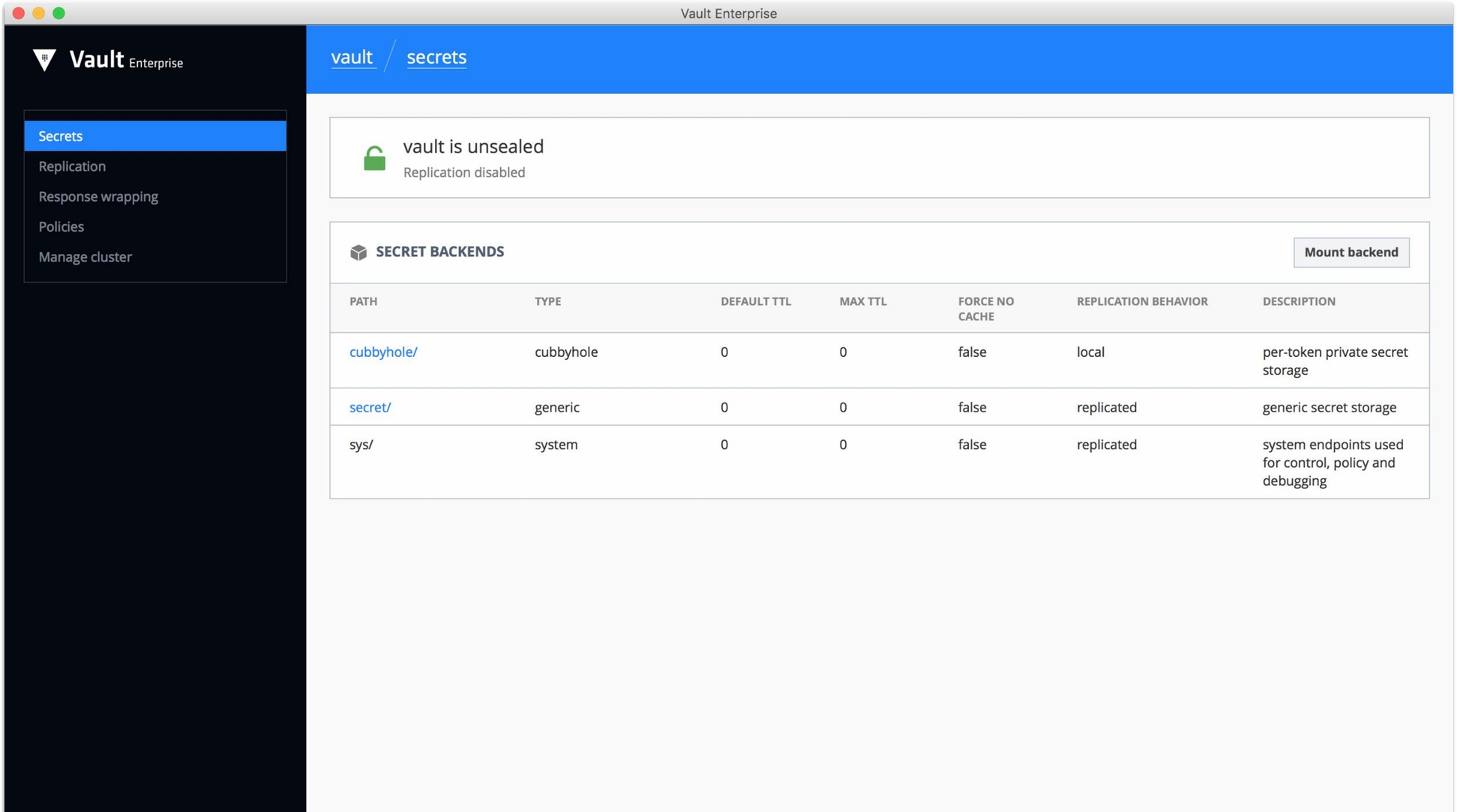
HashiCorp

Vault Enterprise



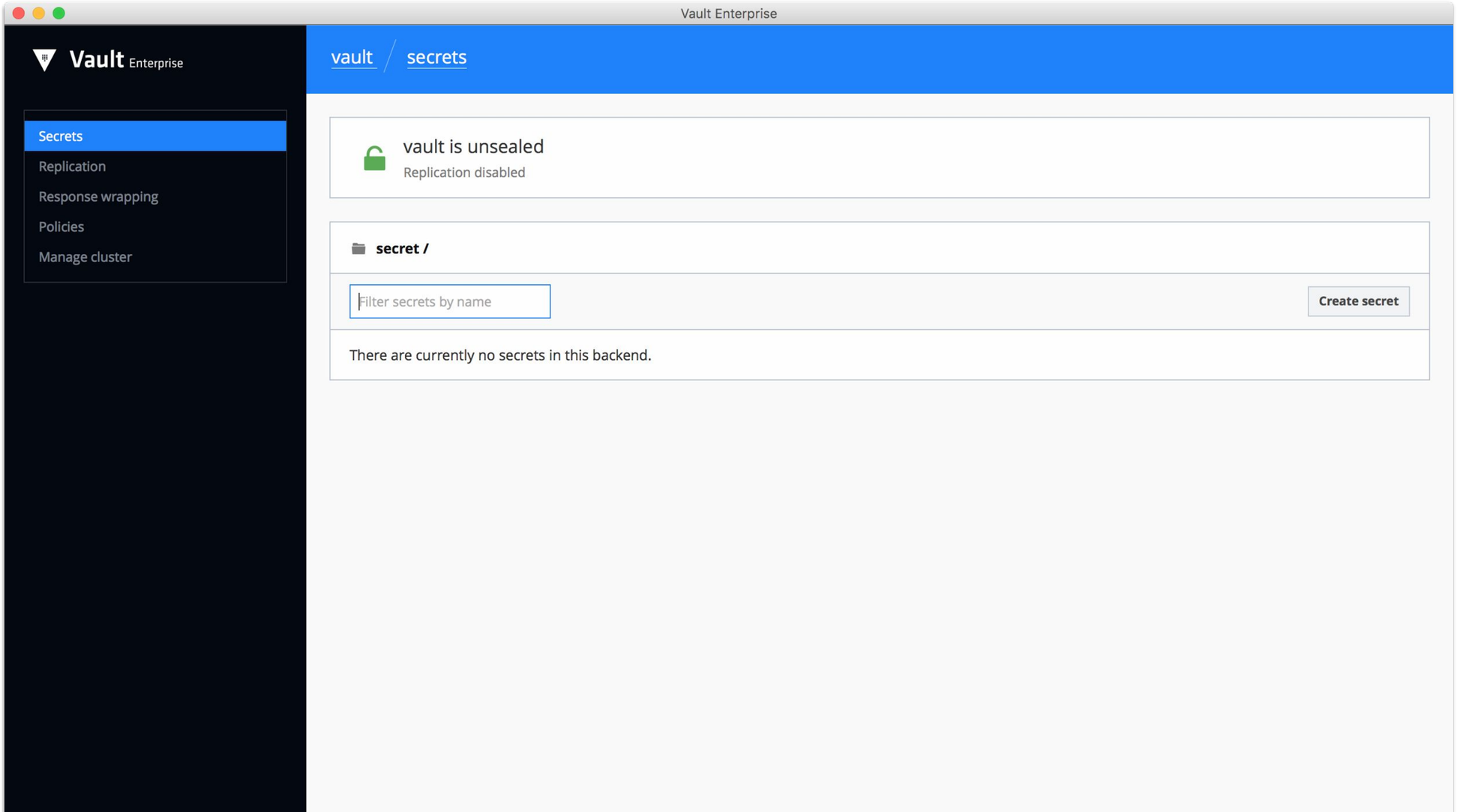
HashiCorp

Vault Enterprise



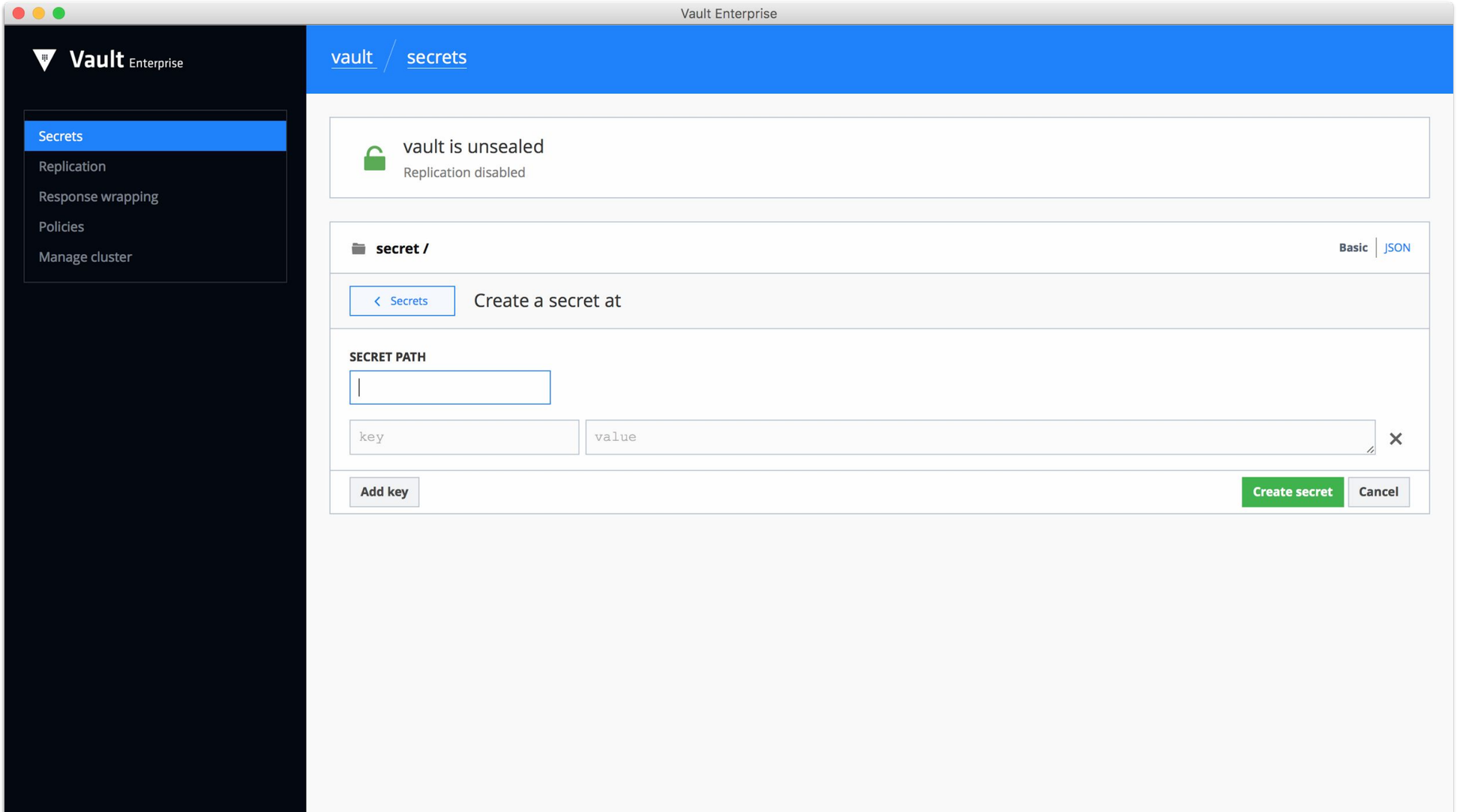
HashiCorp

Vault Enterprise



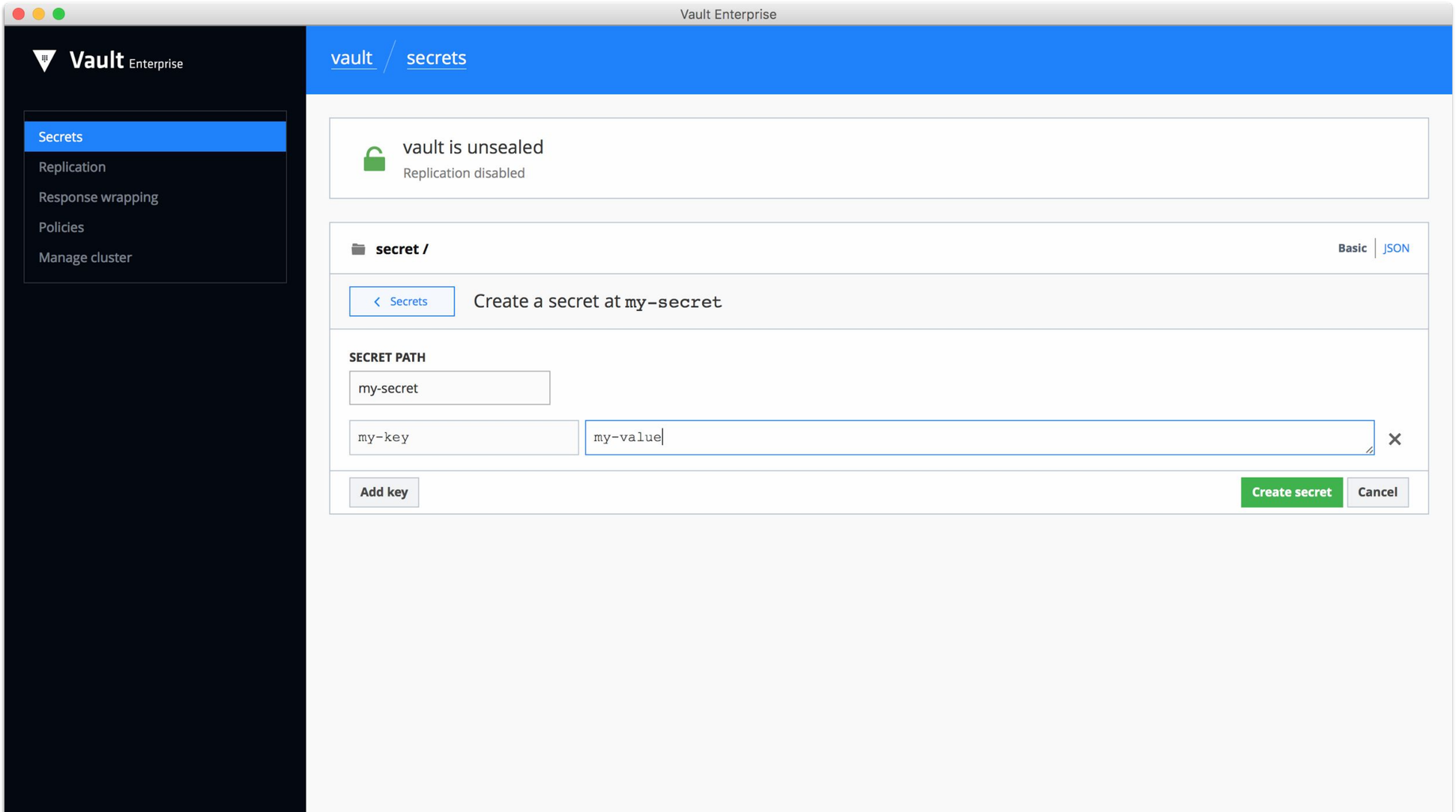
HashiCorp

Vault Enterprise



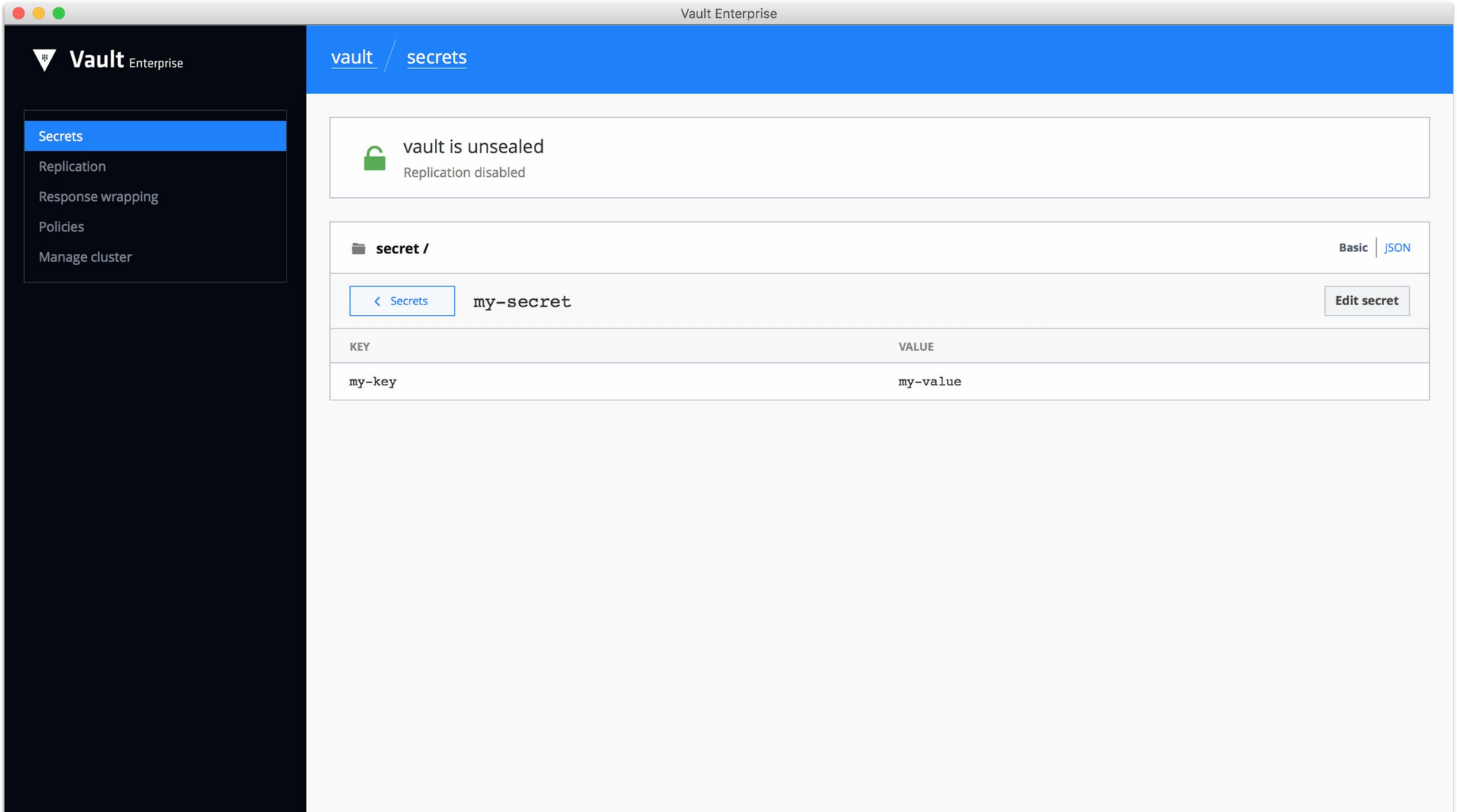
HashiCorp

Vault Enterprise



HashiCorp

Vault Enterprise



HashiCorp

Further Reading

Further Reading

CLI

vaultproject.io/docs/commands

HTTP API

vaultproject.io/docs/http

Internals

vaultproject.io/docs/internals

HashiCorp