HashiCorp Vault Secrets management

START HERE

`vault server -dev`
Throwaway local server

`export VAULT_ADDR=<url>`
Point the CLI at a server

`vault login`
Authenticate, token is cached

`vault status`
Sealed? Initialized?

`vault kv put -mount=secret <path> <key>=<value>`
Store a secret

`vault kv get -mount=secret <path>`
Read a secret

`vault kv list -mount=secret`
List secret paths

`vault policy write <name> <file>`
Define access rules

Auth method userpass, token, oidc... → **issues** → **Token** carries policies → **checked by** → **Policy** path capabilities → **allows** → **Secrets engine** kv, transit, database...

CONCEPTS

The CLI is a thin wrapper over the HTTP API: everything lives at a path, and read/write/list/delete work on any of them.

The server starts sealed. Unseal it with enough key shares before any request works; dev mode skips this.

Dev mode is in-memory and insecure. Never use it for real secrets.

CONNECT

`export VAULT_ADDR=<url>`
Set the server address
Default is https://127.0.0.1:8200

`export VAULT_TOKEN=<token>`
Authenticate via environment
Takes effect without a login; beats the cached token

`vault status`
Show seal and HA state
Exit code 0 unsealed, 2 sealed, 1 error

`-format=json`
Machine-readable output (also yaml)
Or set VAULT_FORMAT

`-namespace=<ns>`
Run inside a namespace (Enterprise)
Or set VAULT_NAMESPACE

`-ca-cert=<file>`
Trust a private CA
Or set VAULT_CACERT

DEV SERVER

`vault server -dev`
Run in-memory, unsealed

`vault server -dev -dev-root-token-id=<id>`
Dev server with a known root token

`vault server -config=<file>`
Run with a config file

LOGIN AND TOKENS

`vault login`
Log in with a token (prompts)

`vault login -method=<type> <key>=<value>`
Log in with another auth method
For example -method=userpass username=alice

`vault token lookup`
Show the current token

LOGIN AND TOKENS (cont.)

`vault token create -policy=<name>`
Create a token with a policy

`vault token create -ttl=<dur> -use-limit=<n>`
Short-lived, limited-use token

`vault token renew`
Extend the current token

`vault token revoke <token>`
Revoke a token and its children

KV SECRETS

`vault kv put -mount=secret <path> <key>=<value>`
Write a secret
On KV v2 this replaces all keys with a new version

`vault kv get -mount=secret <path>`
Read a secret

`vault kv get -mount=secret -field=<key> <path>`
Print one field only
No trailing newline, good for $(...)

`vault kv list -mount=secret`
List paths under a mount

`vault kv get -mount=secret -version=<n> <path>`
Read an older version

`vault kv patch -mount=secret <path> <key>=<value>`
Change keys, keep the rest

`vault kv put -mount=secret <path> @<file>.json`
Write from a JSON file

`vault kv put -mount=secret <path> <key>=-`
Read a value from stdin
Keeps it out of shell history

`vault kv delete -mount=secret <path>`
Soft-delete the latest version

`vault kv undelete -mount=secret -versions=<n> <path>`
Restore a deleted version

`vault kv destroy -mount=secret -versions=<n> <path>`
Permanently remove versions

`vault kv metadata get -mount=secret <path>`
Show versions and settings

RAW API PATHS

`vault read <path>`
GET any path

RAW API PATHS (cont.)

`vault read -field=<key> <path>`
Print one field of a read

`vault write <path> <key>=<value>`
POST/PUT data to a path

`vault write -force <path>`
Write with no data

`vault write <path> <key>=@<file>`
Take a value from a file

`vault list <path>`
List keys under a path

`vault delete <path>`
DELETE a path

`vault path-help <path>`
Show what a path accepts

ENGINES AND AUTH METHODS

`vault secrets list`
Show enabled secrets engines

`vault secrets enable -path=<path> <type>`
Enable an engine
For example kv-v2, transit, database

`vault secrets disable <path>`
Disable and wipe an engine

`vault auth list`
Show enabled auth methods

`vault auth enable <type>`
Enable an auth method
For example userpass, approle, oidc

`vault auth enable -path=<path> <type>`
Enable at a custom path

`vault auth disable <path>`
Disable an auth method

`vault write auth/userpass/users/<name> password=<pw>`
Add a userpass user
Add policies=<name> to grant access

POLICIES

`vault policy list`
List all policies

`vault policy read <name>`
Show a policy's rules

`vault policy write <name> <file>.hcl`
Upload a policy from a file

`vault policy write <name> -`
Upload a policy from stdin

`vault policy delete <name>`
Remove a policy

POLICIES (cont.)

`-output-policy`
Print the minimal policy a command needs

Read-only policy

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

KV v2 policies need /data/ (and /metadata/) in the path, even though -mount hides it in the CLI.

OPERATE AND SEAL

`vault operator init`
Initialize a new server
Prints unseal keys and root token once; store them safely

`vault operator init -key-shares=<n> -key-threshold=<n>`
Choose key split

`vault operator unseal`
Enter one unseal key (prompts)
Repeat until the threshold is reached

`vault operator seal`
Seal the server

`vault lease revoke <lease-id>`
Revoke a lease now

RECIPES

Try Vault locally

```
vault server -dev -dev-root-
token-id=root &
export VAULT_ADDR=http://127.0.
0.1:8200
export VAULT_TOKEN=root
vault kv put -mount=secret app/
db password=s3cret
vault kv get -mount=secret app/
db
```

Policy and scoped token

```
vault policy write app-ro app-
ro.hcl
vault token create -policy=app-
ro -ttl=1h
# hand out the printed token,
not the root one
```

Use a secret in a script

```
DB_PASS=$(vault kv get -
mount=secret -field=password
app/db)
vault kv get -mount=secret -
format=json app/db | jq -
r .data.data.password
```

Bring up a real server

```
vault server -config=config.hcl
vault operator init -key-
shares=5 -key-threshold=3
vault operator unseal   #
repeat with 3 different keys
vault login
```