Vault

🙃

- Passwords, API keys, DB credentials, certs scattered across config, .env, etc.
- Static, long-lived → dynamic, short-lived creds.

Components

Server

Handle requests, apply policy, mng secrets.

Cluster Setup: Active (Leader): 1; Standby: rest

Storage Backend

Save config, e.g. Encryption key, encrypted secrets, ACL, Tokens, LDAP group policy mapping, e.g. Integrated (Raft) Storage, Consul.

Secret Engine (SE)

Store, gen, encrypt data, e.g. KV, PKI, Transit, Cubbyhole, TOTP, Database.

`vault secrets` sub-cmd: `tune, list, move, enable, disable`

KV SE

`vault kv metadata delete` permanently deletes secret of *all ver*; `vault kv destroy` permanently deletes secret of *cur ver*.

ACL rules

```
path "secret/data/dev/team-1/*" {
  capabilities = ["create", "read", "update"]
}
```

Delete policy requires spec setup

- Latest ver

-
  ```
  path "secret/data/dev/team-1/*" {
    capabilities = ["delete"]
  }
  ```
- Any ver

-
  ```
  path "secret/delete/dev/team-1/*" {
    capabilities = ["update"]
  }
  ```
- Undelete permission

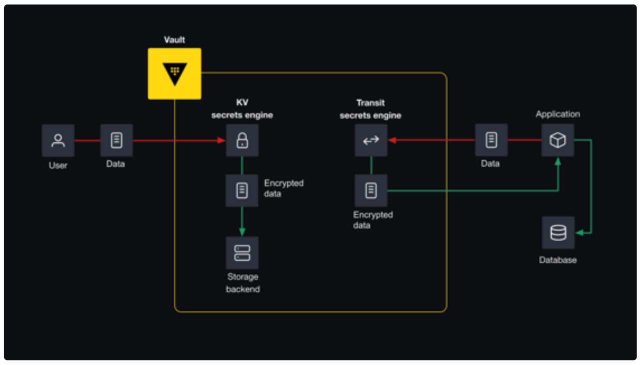
-
  ```
  path "secret/undelete/dev/team-1/*" {
    capabilities = ["update"]
  }
  ```

KV-v2 SE

- Version control: retain configurable #secret ver. This enables data retrieval from older secret ver, in case of undesired data deletion / update.

Transit SE

- so-called EaaS.
- encrypt app DB data, e.g. name, address, ID. Transit key is stored in vault, just like other secrets.
- `vault:v[1,2,3]` #rotation of keys, not ver of encrypted data per se.



Actual workflow

1. Plaintext → Base64 → Ciphertext

```
POST $VAULT_ADDR/v1/<mount-path>/encrypt/<key>
{
  "plaintext": <base64-encoded text>
}

echo "{ \"plaintext\": \"$(echo -n 'first-plaintext' | base64)\" }" | \
POST -H "X-Vault-Token: $VAULT_TOKEN" \
     -c "application/json" \
     $VAULT_ADDR/v1/transit-1/encrypt/crypto
```

HTTP body in `echo`

2. UPSERT email w/ encrypted email

```
INSERT INTO users (username, email)
VALUES ('bob', 'vault:v1:8n3A...9s==');

UPDATE users
SET email = 'vault:v1:8n3A...9s=='
WHERE id = 50;
```

Config

- TTL

Unsealed / Sealed

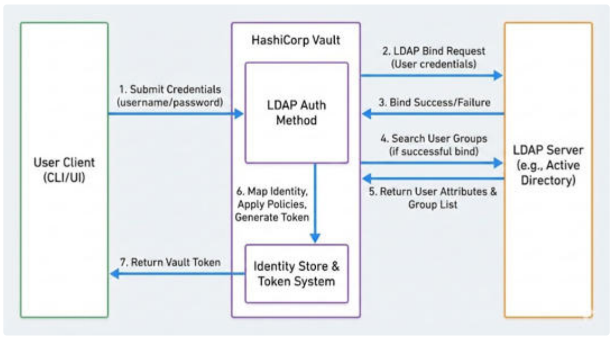
- Shared keys (collect min. requirement) → Master / Root key → Encryption key → Secrets
- For Shamir's method, shared keys distributed among Security, Infra, Senior roles.
- Keyhole, i.e. encryption algo. for encryption key is known to everyone, e.g. AES-256, but reconstruction is almost impossible.

Methods

- Shamir's Secret Sharing: Gen share keys to decrypt Master / Root key

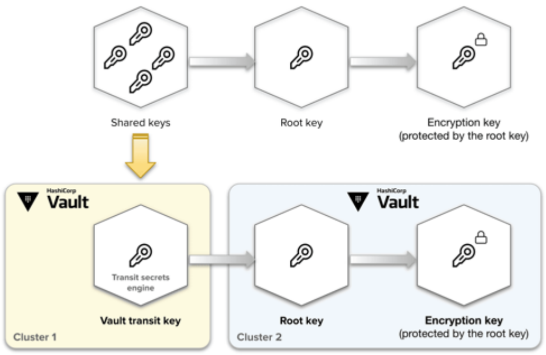
  Auth

  User → Vault [delegate / bind] → LDAP


HashiCorp Vault
User Client (CLI/UI)
1. Submit Credentials (username/password)
LDAP Auth Method
2. LDAP Bind Request (User credentials)
3. Bind Success/Failure
4. Search User Groups (if successful bind)
5. Return User Attributes & Group List
LDAP Server (e.g., Active Directory)
6. Map Identity, Apply Policies, Generate Token
7. Return Vault Token
Identity Store & Token System

-
  ```
  vault operator init -key-shares=7 -key-threshold=4
  ```

  (N, K) = (7, 4), N: #shared_keys, K: min. requirement to reconstruct / glue up a master key.
- HSM (Hardware Sec. Mod.) Auto Unseal: Master key → encryption key in HSM → Wrapped key
- Cloud Auto Unseal: Master key → customer-mng key → Wrapped key
  - Customer-mng key is in Cloud provider KMS (Key Mgmt Sys)
- Transit SE



Scenario

Cluster : port mapping : 1 : `8200` , 2 : `8100`

1. Config auto-unseal key provider (Cluster 1)

   ```
   vault token create -orphan -policy="autounseal" -wrap-ttl=120 -period=24
   ```

2. Config auto-unseal (Cluster 2)

   Unwrap secret from Cluster 1 & set `VAULT_TOKEN` env var

2.
   ```
   vault unwrap -field=token $(cat wrapping-token.txt)
   export VAULT_TOKEN="hvs.my_token"
   ```

   `hvs` HashiCorp Vault Service, vide Types.

2. `config-autounseal.hcl`

3.
   ```
   seal "transit" {
     address = "$VAULT_ADDR"
     disable_renewal = "false"
     key_name = "autounseal"
     mount_path = "transit/"
     tls_skip_verify = "true"
   }
   ```

   ```
   vault server -config=config-autounseal.hcl
   ```

4. Terminate

```
pgrep -f vault | xargs kill
unset VAULT_TOKEN VAULT_ADDR
rm config-autounseal.hcl vault-1.log audit.log
rm -r vault/wrapping-token.txt
```

`xargs` : like grep, read items from stdin, then build & exec cmds.

Master key mgmt for Cluster 1? Quis custodiet ipsos custodes? Who will watch the watchmen?

Sol:

- Cluster 1: internal, Cluster 2: external, separated by *Firewall*.
- Or ultimately require Shamir's Secret Sharing, HSM, Cloud auto unseal.

Transit SE auto-unseal

State

- Sealed: Planned maintenance, Power/OS crash, emergency.
  - Master key: DNE, requires shared keys reconstruction / glue up
  - Encryption key: disk storage
- Unsealed: 99% of the time.
  - Master key & Encryption key @server mem

Replication & Disaster Recovery (DR)

2 Vault Cluster Types

- Primary:
  - Automate cluster data snapshots to local / cloud storage
  - Replicate data to Secondary, incl. config, policy, auth methods, SE, tk, audit logs.
- Secondary: cluster maintenance.
  - Performance Secondary: Speed, scaling, locality. Does NOT contain: tk & leases.
  - DR Secondary: backup

- GDPR personally identifiable data not be physically transferred to locations outside EU unless the region / country has an equal rigor of data protection regulation as EU.

  Replication → Performance → Secondaries → Paths filter: Deny: EU_GDPR_data/, office_FR/

  or `vault secrets enable -local -path=<path/> kv-v2`

Primary & Secondary comm. via Repilcation Boostrap Bundle (Secondary actv tk) *qua* secret handshake. Secondary identify Primary by the Bundle and mTLS (mutual TLS) protocol.

P: Performance Cluster qui replicates to a DR Secondary cluster.

D: DR Primary, the cluster being backed-up. Standalone cluster, i.e. w/o Secondary Perf is also D.

$\forall x \in P \implies D, P \subset D$

Performance scaling

- Active node: handle req qui mutate storage.

- Standby node: handle req qui do not mutate storage, e.g. Transit SE (EaaS)

Identity

- Entity: an identity, e.g. a person.
- Group: an identity w/ members, i.e. entities.
- Aliases: entity's accounts w/ auth, e.g. LDAP, GitHub. Must be on different auth mount, otherwise unable to map entity.

Policy

Capabilities: CRUD, list, patch, sudo, deny

```
path "auth/userpass/*" {
  capabilities = [ "create", "read", "update", "delete" ]
}
```

1. Define ACL policy w/ HCL (JSON)

```
vault policy write <policy-name> <filename>.hcl

vault policy write developer-vault-policy - << EOF
path "dev-secrets/+/creds" {
   capabilities = ["create", "list", "read", "update"]
}
EOF
```

- `+` denote any #char.
- Permit CRUD the "creds" path *under any top-lvl path* under dev-secrets/

```
path "secret/apps/confidential" {} # confidential per se
path "secret/apps/confidential/*" {} # children of confidential
```

> Apply policy: `vault token create -policy=<policy-name>`

2. Enable new SE on a path

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

3. Get Policy

```
vault kv get -output-policy dev-secrets/creds
```

Set user policy

```
vault write /auth/userpass/users/danielle-vault-user \
    password='training' \
    policies=developer-vault-policy
```

Secret engine

```
vault secrets list
```

Put a secret in kv SE

```
vault kv put /dev-secrets/creds api-key=E6BED968-0FE3-411E-9B9B-C45812E4
```

View secret

```
vault kv get /dev-secrets/creds
```

Login

```
vault login -method=userpass username=danielle-vault-user
```

- Will ask for password, then show tk-policy

Fine-grained ACL param policy

- Param: `required_parameters, allowed_parameters, denied_parameters`
- Use `allowed` over `denied`, positive setup

```
path "auth/userpass/users/*" {
  capabilities = ["update"]
  allowed_parameters = {
    "password" = []
  }
}
```

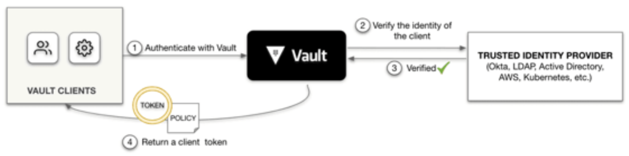
- Allows user update password param val.
- However, cannot update other param val.

Deny: block acc.

```
path "dev-secrets/+/root" {
  capabilities = ["deny"]
}
```

- Dev team *root* acc. not permitted.

Token & Policy: Client (user) auth → Tk-Policy pair


VAULT CLIENTS
1 Authenticate with Vault
Vault
2 Verify the identity of the client
3 Verified
TRUSTED IDENTITY PROVIDER (Okta, LDAP, Active Directory, AWS, Kubernetes, etc.)
TOKEN
POLICY
4 Return a client token

While update is poss, il y a no way to change policies asso. w/ a tk, must *revoke*.

Use Identity to spec user in path

```
path "dev-secrets/+/creds/{{identity.entity.name}}" {
  capabilities = ["create", "list", "read", "update"]
}
```

Token

Orphan token

- tk holder creates new tk, new tk will be children of orig. tk
- When parent is revoked, all children revoked as well.
- But orphan can still happen, e.g.
  a. `write auth/token/create-orphan` endpoint w/ `sudo` permission
  b. sudo / root acc. `auth/token/create` & set `no_parent=true`
  c. Tk store roles
  d. Login w/ non-tk auth method

Token accessor

Tk ref ID, use it to revoke or renew tk.

Turn OFF obfuscation for tk accessor in audit log:

- Pro: Revoke tk immediately, tk accessor plaintext is on the log.
- Con: Hacker got audit logs, and revoke tks by tk accessors in the logs → DoS attack.

P.S. ON: tk accessor is hashed, e.g. `hmac-sha256:7d9s...`

TTL

Set `explicit_max_ttl`

If period/Max TTL not set, TTL will be det. by

1. Sys Max TTL: 32 days.
2. Max TTL on *mount*, override sys Max TTL, i.e. higher priority.
3. Suggested by auth method, config-ed per role, group, user. *SuggestionMaxTTL < MountSysMaxTTL*

Mount

`"config": { "default_lease_ttl": "1s", "max_lease_ttl": "5m" }`

`"lease_ttl": 0` : use default, i.e. mount or sys

Periodic token

Created by

1. sudo / root tk @ `auth/token/create` endpoint.
2. Tk store roles.
3. auth method qui support it, e.g. AppRple.

Only way for an infinte lifetime tk, aside from root tk.

Types

- Service: normal
- Batch: require no storage, limited functionality,
  - ✅ used across Performance Replication clusters, creation scales w/ Performance Standby Node Count
  - ❌ be root tk, create child, renew, manual revoke, be periodic, MaxTTL, have accessors.
- Recovery: TBD

Prefix for type def.

`hvs.string`	Service
`hvb.string`	Batch
`hvr.string`	Recovery

Lease

Time related metadata of Tk & dynamic secrets.

TTL vs. Max TTL

TTL: duration a tk can exist.

Max TTL: how frequently app must check-in.

```
vault lease renew/revoke <lease_id>
```

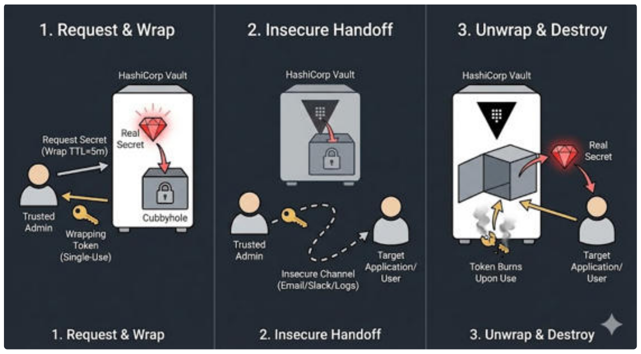
```
# Check all tk
vault list auth/token/accessors

# Check tk id
vault token lookup -accessor <tk_accessor>
```

Response Wrapping

- Cubbyhole: secret box, a type of SE, each tk has its own storage, cf. kv
- Wrapping Tk: single use for cubbyhole
- Wrap TTL: lifetime of wrapping Tk

Wrapping Tk is intercepted → Real receiver cannot unwrap → Trigger alert



App Role

- RoleID: identifier for a role
- SecretID: dynamic cred, auth-ed for a client to use a RoleID.

Auth w/ Vault-defined roles

```
vault auth enable approle
```

Create a role

```
vault write auth/approle/role/my-role \
    token_type=batch \
    secret_id_ttl=10m \
    token_ttl=20m \
    token_max_ttl=30m \
    secret_id_num_uses=40
```

If tk issued by approle requires to create *child tk*, set `token_num_uses=0`.

Vault Secrets Operator (VSO)

a K8s operator, sync secrets from Vault → K8s, e.g. DB → Vault → K8s VSO ns → other ns

CLI

CLI Doc: Env var

Cheat sheet

```
# Login & Logout
export VAULT_TOKEN=<tk>
unset VAULT_TOKEN

# Server off
pgrep -f vault | xargs kill

# List engines / auth method / policies
vault <secrets / auth / policy> list

# List content
vault list <path>
```

Init

Dev server: in-mem

```
vault server -dev -dev-root-token-id root -dev-tls
export VAULT_ADDR=<addr>
export VAULT_CACERT='/var/folders/qr/zgztx0sj6n1dxy86sl36ntnw0000gn/T/va
```

Prod server

```
vault operator init
```

- This will create root tk and shared keys.

Check server status

```
vault status -format=json
```

Path / endpoint info

Path Semantics	`<Mount point>/<Internal API route>/<Resoruce name>`
isVar	Yes/No/Yes

```
vault path-help /auth/userpass
```

Common used path

```
secret/data/<name>
secret/<metadata>

transit/keys/<key-name>
transit/encrypt/<key-name>
transit/decrypt/<key-name>
transit/sign/<key-name>

# Require payload @ HTTP body for auth
auth/userpass/login/<username>
auth/approle/login
auth/kubernetes/login
auth/token/create
```

```

# Dynamic secrets
database/creds/<role-name>
aws/creds/<role-name>
```

Enable Auth method

```
vault auth enable userpass
vault auth list
```

View policies

```
vault policy list

vault read sys/policy # raw API acc
```

Token renew & revoke

```
vault token renew -accessor <tk-accessor>
vault token revoke <tk or tk-accessor>
```

DB SE: Dynamic DB cred mgmt

0. Create role 'class' and spec acc. in DB

```
CREATE ROLE \"ro\" NOINHERIT;
"GRANT SELECT ON ALL TABLES IN SCHEMA public TO \"ro\";"
```

1. Set env var for PostgreSQL adr.

SE → DB via plugin interface, in casu `postgresql-database-plugin`

```
export POSTGRES_URL=$TF_VAR_POSTGRES_URL
```

or API Gateway (Kong, ngrok) FW adr., do not incl. `tcp://`

2. Enable DB SE

```
vault secrets enable database
```

or spec path

```
vault secrets enable -path=<mount-path> database
```

`database` : engine type

3. Conn. to DB, config SE

```
vault write database/config/postgres \
   plugin_name=postgresql-database-plugin \
   connection_url="postgresql://{{username}}:{{password}}@$POSTGRES_URL/
   allowed_roles=dev \
   username="root" password="rootpassword"
```

Create dynamic role

1. SQL def. for role instance bind to role class

`dev-role.sql`

```
CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{
GRANT ro TO "{{name}}";
```

`tee` : read stdin, write it to stdout & files. Overwrite file vs. `echo "..." >> filename.txt` : append

2. Create role instance (dynamic role)

```
vault write database/roles/dev \
      db_name=postgres \
      creation_statements=@dev-role.sql \
      default_ttl=1h \
      max_ttl=24h
```

3. App / Dev side: req dynamic cred.

```
vault read database/creds/dev
```

Chk DB users

```
SELECT username, valuntil FROM pg_user;
```

N.B. create pseudo-root user for Vault to utilize rather than using the actual root.

Transit SE

Create encryption keyring `orders`

```
vault write -f transit/keys/orders
```

Keyring: store known encryption keys

Encrypt

```
vault write -format=json transit/encrypt/orders \
      plaintext=$(printf thisIsPlaintext | base64) \
      | jq -r ".data.ciphertext" > cipher.txt
```

Plaintext must be base64-encoded.

Decrypt

```
vault write -field=plaintext transit/decrypt/orders \
ciphertext=$(cat cipher.txt) \
| base64 -d ; echo
```

KV SE

Write

```
vault kv put/patch -mount=secret <secret-name> <k1>=<v1> <k2>=<v2>
```

Mount to `secret/data/my-secret`, could be other path, it is spec in

```
vault secrets enable -path=finance-data kv-v2
```

vide Enable new SE on a path, Enable DB SE.

`PATCH`: update keys; `PUT`: will overwrite all keys

List secrets

```
vault kv list -mount=<mount>
```

Read: use `get` for kv engine

```
vault kv get -mount=<mount> <secret-name>
vault read <mount>/data/<secret-name>
```

Undelete

```
vault kv undelete -mount=secret -versions=2 my-secret
```

Permanently Delete

```
vault kv destroy -mount=secret -versions=2 my-secret
```

Write Metadata

```
vault kv metadata put -custom-metadata=Membership="Platinum" secret/cust
```

Retain #ver

- Global Config

```
vault write secret/config max_versions=4
```

- Local Config: overwrite global

```
vault kv metadata put -max-versions=4 secret/customer/acme
```

Password Policy

e.g. 6-digit OTP

```
length = 6
rule "charset" {
  charset = "0123456789"
  min-chars = 6
}
```

```
length = 12
rule "charset" {
  charset = "abcdefghijklmnopqrstuvwxyz"
  min-chars = 1
}
rule "charset" {
  charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  min-chars = 1
}
rule "charset" {
  charset = "0123456789"
  min-chars = 1
}
rule "charset" {
  charset = "!@#$%^&*()_+-="
  min-chars = 1
}
```

```
vault write sys/policies/password/policy-1 policy=my-pwd-policy.hcl

vault kv put -mount=secret my-secret \
    password=$(vault read -field password sys/policies/password/policy-1
```

HA Cluster for Integrated (Raft) Storage

`vault_[1,2,3,4]:` [Transit auto-unseal, Leader, Follower], Follower `config-vault_2.hcl`

```
storage "raft" {
   path    = "<path_to_local>/raft-vault_2/"
   node_id = "vault_2"
}

# Vault_2 cluster
export VAULT_ADDR="http://127.0.0.2:8200"

vault operator raft list-peers
```

Join nodes

```
export VAULT_ADDR="http://127.0.0.3:8200" # adr. for vault_3

vault operator raft join http://127.0.0.2:8200

export VAULT_TOKEN=$(cat root_token-vault_2)
```

Retry join: auto join cluster, if node connection details are known `config-vault_4.hcl`

```
storage "raft" {
   path    = "<path_to_local>/raft-vault_4/"
   node_id = "vault_4"
   retry_join {
      leader_api_addr = "http://127.0.0.2:8200"
   }
   retry_join {
      leader_api_addr = "http://127.0.0.3:8200"
   }
}
```

Resign from Leader (acitve duty)

```
vault operator step-down
```

Remove cluster member

```
vault operator raft remove-peer <node-name>
```

Take & Restore SNAPSHOT

```
vault operator raft snapshot save <filename>.snapshot
vault operator raft snapshot restore
```

Agent / Proxy

Agent > Proxy

Proxy: For static secret caching

`agent.hcl`

```
vault {
  address = "https://127.0.0.1:8200"
  tls_skip_verify = true
}

# Auth: Read & mng an existing key
auto_auth {
  method {
    type = "token_file"
    config {
      token_file_path = "/opt/vault/agent-token.txt"
    }
  }

  # Sink: Write renewed tk back to the same file.
  sink "file" {
    config = {
      path = "/opt/vault/agent-token.txt"
      mode = 0600
    }
  }
}

# min template to satisfy Agent's validation check.
# e.g., IRL: DB login
template {
  source      = "/etc/vault.d/app.ctmpl"
  destination = "/app/.env"
  perms = 0600
}
```

`/etc/vault.d/app/ctmpl`

```
# This section queries Vault once for the secret
{{ with secret "secret/data/webapp/db" }}

# .Data.data acc the inner JSON of the KV v2 secret
DB_HOST="{{ .Data.data.host }}"
DB_PORT="5432"
DB_USER="{{ .Data.data.username }}"
DB_PASS="{{ .Data.data.password }}"

# Add logic. If no 'debug' key exists, default to false.
DEBUG_MODE="{{ if .Data.data.debug }}{{ .Data.data.debug }}{{ else }}fa
{{ end }}
```

```
vault agent -config=agnet.hcl
```

KV & Identity (Entity & Group)

```
vault secrets enable -mount=test-kv -version=2 kv

vault kv put -mount=test-kv first-secret a=12345 b=23456

vault kv patch -mount=test-kv first-secret a=aaaaa

vault kv get -mount=test-kv -version=2 first-secret

vault kv delete/undelete -mount=test-kv -versions=3,5 first-secret

# delete/undelete w/ -versions flag, spec by 1,2,3 (w/o space)


# ------ Password gen ------
vault write sys/policies/password/policy-1 policy=@/etc/vault.d/6-dgt-o

vault kv put -mount=test-kv second-secret \
password=$(vault read -field password sys/policies/password/policy-1/ge



# ------ Write identity policy ------
# Obj: Entity ID, userpass accessor ID, Group ID

vault auth enable userpass

vault write auth/userpass/users/john password=password policies=default

vault write identity/entity name="entity-john"
# --- Return: John entity ID, e.g. 218b49ce-4271-aee0-c0-57762df2fbb6

vault auth list -format=json | jq -r '."userpass/".accessor'

# --- Return: userpass accessor ID, e.g. auth_userpass_5903a8c7

vault write identity/entity-alias name="john" \
canonical_id=<entity_id> \
mount_accessor=<userpass_accessor_id>


vault policy write dev /etc/vault.d/dev.hcl

vault write identity/group \
name="managers" \
policies=<policy_name> \
member_entity_ids=<entity_id>

```

```
# --- Return: Group ID

vault login -method=userpass username=john
```

File Structure in Different Environments

Machine	Cluster
`/etc/vault.d`: config	`/vault`
`/opt/vault/`: TLS certs	