Vault HTTP API with Postman — Quickstart

Compiled for local demos and learning

November 1, 2025

Contents

1 Overview **2**

2 Prerequisites **2**

3 KV Engine Version (v1 vs v2) **2**

4 Workflow Steps **3**

4.1 1) Status: uninitialized vs sealed . . . 3
4.2 2) Initialize Vault . . . 3
4.3 3) Unseal (repeat threshold times) . . . 4
4.4 4) Write a KV secret (v2) . . . 4
4.5 5) Update the secret (overwrite) . . . 5
4.6 6) Read the secret . . . 5
4.7 7) List mounts . . . 5
4.8 8) Enable `userpass` auth method . . . 5
4.9 9) Create a user . . . 6
4.10 10) Login as that user . . . 6

5 Troubleshooting **6**

6 Optional: Postman Environment Template **7**

7 Appendix: End-to-end cURL Script (for reference) **7**

Security heads-up: Postman is great for demos, but it can sync environments and request history. For anything beyond local learning, prefer CLI or code. If you do use Postman, disable sync for the environment, avoid storing production tokens, and clear history when done.

1 Overview

You will exercise core Vault flows through HTTP using Postman:

1. Status check
2. Initialize (Shamir shares)
3. Unseal (repeat threshold times)
4. Write, update, and read a KV secret
5. List mounts
6. Enable `userpass`
7. Create a user and log in to get a client token

2 Prerequisites

- A Vault server reachable at `http://localhost:8200`
- Postman installed
- A Postman *Environment* with these variables:
  - `BASE_URL` = `http://localhost:8200`
  - `ROOT_TOKEN` (populate after init)
  - `UNSEAL_1`, `UNSEAL_2`, `UNSEAL_3` (during init)
- For safety: turn *Sync Off* for this environment and uncheck "Save responses" in Postman preferences.

3 KV Engine Version (v1 vs v2)

The v2 engine mounts at `/secret/` but uses the `/data` and `/metadata` subpaths and nests fields under `"data"`. Confirm your mount and version:

Request (curl)

```
curl -s \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  ${BASE_URL}/v1/sys/mounts | jq .
```

Expected shape (excerpt)

```
{
  "secret/": {
    "type": "kv",
    "options": { "version": "2" },
    "config": { "default_lease_ttl": 0, "max_lease_ttl": 0 }
  },
  "...": { }
}
```

If `"options":"version":"2"` is present, use the v2 paths shown below. If you are on v1, adjust paths accordingly (noted where relevant).

4 Workflow Steps

4.1 1) Status: uninitialized vs sealed

Method: GET **URL:** `{{BASE_URL}}/v1/sys/seal-status`
Headers: none

curl

```
curl -i {{BASE_URL}}/v1/sys/seal-status
```

Notes

On a fresh server, you may get `400 Bad Request` with `"initialized": false`.

4.2 2) Initialize Vault

Method: PUT **URL:** `{{BASE_URL}}/v1/sys/init`

Body (JSON)

```
{
  "secret_shares": 5,
  "secret_threshold": 3
}
```

curl

```
curl -s -X PUT {{BASE_URL}}/v1/sys/init \
  -d '{"secret_shares":5,"secret_threshold":3}'
```

Result

You receive `unseal_keys_b64` (5 values) and a `root_token`. Store three keys into `UNSEAL_1..3` and keep `ROOT_TOKEN` for admin calls.

4.3 3) Unseal (repeat threshold times)

Method: PUT **URL:** `{{BASE_URL}}/v1/sys/unseal`

Body (JSON)

```
{ "key": "{{UNSEAL_1}}" }
```

curl

```
curl -s -X PUT {{BASE_URL}}/v1/sys/unseal \
  -d "{\"key\":\"${UNSEAL_1}\"}"
curl -s -X PUT {{BASE_URL}}/v1/sys/unseal \
  -d "{\"key\":\"${UNSEAL_2}\"}"
curl -s -X PUT {{BASE_URL}}/v1/sys/unseal \
  -d "{\"key\":\"${UNSEAL_3}\"}"
```

Result

After the final call, `"sealed": false`.

4.4 4) Write a KV secret (v2)

Method: POST **URL:** `{{BASE_URL}}/v1/secret/data/mySecret`
Headers: `X-Vault-Token: {{ROOT_TOKEN}}`

Body (JSON)

```
{ "data": { "dbUser": "dbUser1" } }
```

curl

```
curl -i -X POST {{BASE_URL}}/v1/secret/data/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"data":{"dbUser":"dbUser1"}}'
```

Notes

Depending on Vault version, you may see `204 No Content` or `200 OK`.

KV v1 path (if applicable):

```
# v1 uses /v1/secret/mySecret and raw fields, no "data" envelope
curl -i -X POST {{BASE_URL}}/v1/secret/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"dbUser":"dbUser1"}'
```

4.5 5) Update the secret (overwrite)

Method: PUT **URL:** {{BASE_URL}}/v1/secret/data/mySecret

Body (JSON)

```
{ "data": { "dbUser": "dbUser2" } }
```

curl

```
curl -i -X PUT {{BASE_URL}}/v1/secret/data/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"data":{"dbUser":"dbUser2"}}'
```

4.6 6) Read the secret

Method: GET **URL:** {{BASE_URL}}/v1/secret/data/mySecret

curl

```
curl -s {{BASE_URL}}/v1/secret/data/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" | jq .
```

Expected shape

```
{
  "data": {
    "data": { "dbUser": "dbUser2" },
    "metadata": { "version": 2, "...": "..." }
  }
}
```

4.7 7) List mounts

Method: GET **URL:** {{BASE_URL}}/v1/sys/mounts

curl

```
curl -s {{BASE_URL}}/v1/sys/mounts \
  -H "X-Vault-Token: ${ROOT_TOKEN}" | jq .
```

4.8 8) Enable userpass auth method

Method: POST **URL:** {{BASE_URL}}/v1/sys/auth/userpass

Body (JSON)

```
{ "type": "userpass" }
```

curl

```
curl -i -X POST {{BASE_URL}}/v1/sys/auth/userpass \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"type":"userpass"}'
```

4.9 9) Create a user

Method: POST **URL:** `{{BASE_URL}}/v1/auth/userpass/users/vaultuser`

Body (JSON)

```
{ "password": "s3cret-P@ss!", "policies": "default" }
```

curl

```
curl -i -X POST {{BASE_URL}}/v1/auth/userpass/users/vaultuser \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"password":"s3cret-P@ss!","policies":"default"}'
```

4.10 10) Login as that user

Method: POST **URL:** `{{BASE_URL}}/v1/auth/userpass/login/vaultuser`

Body (JSON)

```
{ "password": "s3cret-P@ss!" }
```

curl

```
curl -s -X POST {{BASE_URL}}/v1/auth/userpass/login/vaultuser \
  -H "Content-Type: application/json" \
  -d '{"password":"s3cret-P@ss!"}' | jq .
```

Result

The response contains `auth.client_token`. Use this token for subsequent, user-scoped requests.

5 Troubleshooting

- **400 on `/sys/seal-status`**: Vault not initialized yet. Run init (Section 2).
- **503 Service Unavailable**: Vault is sealed. Unseal until `"sealed": false`.
- **403 permission denied**: Missing or invalid token. Ensure `X-Vault-Token` header is set.
- **KV v1 vs v2 mismatch**: v2 uses `/secret/data/<path>` and nests under `"data"`. v1 writes to `/secret/<path>` with raw fields.
- **Postman history leakage**: Disable sync for the environment and clear history after the session.

6 Optional: Postman Environment Template

You can import a minimal environment JSON and then populate values interactively.

```
{
  "id": "vault-local-demo",
  "name": "Vault Local Demo",
  "values": [
    { "key": "BASE_URL", "value": "http://localhost:8200", "enabled": true },
    { "key": "ROOT_TOKEN", "value": "", "enabled": true },
    { "key": "UNSEAL_1", "value": "", "enabled": true },
    { "key": "UNSEAL_2", "value": "", "enabled": true },
    { "key": "UNSEAL_3", "value": "", "enabled": true }
  ],
  "_postman_variable_scope": "environment",
  "_postman_exported_at": "2025-11-01T00:00:00.000Z",
  "_postman_exported_using": "Postman/CI"
}
```

7 Appendix: End-to-end cURL Script (for reference)

This script roughly mirrors the Postman flow for a local dev server.

```
#!/usr/bin/env bash
set -euo pipefail

BASE_URL="${BASE_URL:-http://localhost:8200}"

echo "==> Seal status"
curl -s -i ${BASE_URL}/v1/sys/seal-status || true

echo "==> Initialize (5 shares, threshold 3)"
INIT=$(curl -s -X PUT ${BASE_URL}/v1/sys/init \
  -d '{"secret_shares":5,"secret_threshold":3}')
ROOT_TOKEN=$(echo "$INIT" | jq -r '.root_token')
UNSEAL_KEYS=($(echo "$INIT" | jq -r '.unseal_keys_b64[]'))
echo "ROOT_TOKEN=${ROOT_TOKEN}"

echo "==> Unseal x3"
for i in 0 1 2; do
  curl -s -X PUT ${BASE_URL}/v1/sys/unseal \
    -d "{\"key\":\"${UNSEAL_KEYS[$i]}\"}" >/dev/null
done

echo "==> Write KV v2 secret"
curl -s -X POST ${BASE_URL}/v1/secret/data/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"data":{"dbUser":"dbUser1"}}' >/dev/null

echo "==> Update secret"
curl -s -X PUT ${BASE_URL}/v1/secret/data/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"data":{"dbUser":"dbUser2"}}' >/dev/null

echo "==> Read secret"
curl -s ${BASE_URL}/v1/secret/data/mySecret \
  -H "X-Vault-Token: ${ROOT_TOKEN}" | jq .

echo "==> Enable userpass"
curl -s -X POST ${BASE_URL}/v1/sys/auth/userpass \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
  -H "Content-Type: application/json" \
  -d '{"type":"userpass"}' >/dev/null

echo "==> Create user vaultuser"
curl -s -X POST ${BASE_URL}/v1/auth/userpass/users/vaultuser \
  -H "X-Vault-Token: ${ROOT_TOKEN}" \
```

```
  -H "Content-Type: application/json" \
  -d '{"password":"s3cret-P@ss!","policies":"default"}' >/dev/null

echo "==> Login as vaultuser"
curl -s -X POST ${BASE_URL}/v1/auth/userpass/login/vaultuser \
  -H "Content-Type: application/json" \
  -d '{"password":"s3cret-P@ss!"}' | jq '.auth.client_token'
```