

Demo HashiCorp Vault

Group: BT

Team Members:

Member	Student ID
Pham Quang Binh	2270663
Nguyen Duc Thuy	2012158

Setup infrastructures

In this tutorial, we will spawn up 2 Docker containers to simulate our system. The containers are:

1. Hashicorp Vault server (as ours IAM system)
2. PostgreSQL (as the database)

Inside the repository, we include a demo `docker-compose.yaml` file for this instance.

```
docker compose up
```

Docker Compose will run and logs from our database and Vault will be displayed in the terminal.

Check for the log similar to below:

```
demo-havc-vault-1 | You may need to set the following environment variables:
demo-havc-vault-1 |
demo-havc-vault-1 | $ export VAULT_ADDR='http://0.0.0.0:8200'
demo-havc-vault-1 |
demo-havc-vault-1 | The unseal key and root token are displayed below in case you want to
demo-havc-vault-1 | seal/unseal the Vault or re-authenticate.
demo-havc-vault-1 |
demo-havc-vault-1 | Unseal Key: /cHKDK3ZU/da2t017mnJZp7yd0CzyYwjncMFBjKB6Ek=
demo-havc-vault-1 | Root Token: toor
demo-havc-vault-1 |
```

Create Vault secrets

Login to Vault

We first need to login to Vault before we can do other operations:

```
vault login
```

If error, try to run the below script **in the same terminal** to point Vault CLI to the correct server location:

```
export VAULT_ADDR='http://localhost:8200'
```

Create Secret Engine and add PostgreSQL credentials

Let us first create a specific **Secret Engine** in our Vault. Using that, we add 2 pairs of key-value secrets, storing the username and the password of the PostgreSQL database.

```
vault secrets enable -version=2 -path="backend-secret" -description="Test K/V v2" kv
```

```
vault kv put -mount="backend-secret" postgres user=postgres  
password=password
```

We are currently using Secret engine of Key-Vault - version 2. For comparison with version 1, visit the documentation: [KV secrets engine](#).

To check whether the secret has been added successfully, try getting back the secret

```
vault kv get -mount="backend-secret" postgres
```

```
> vault kv get -mount="kv-v2" postgres
=== Secret Path ===
kv-v2/data/postgres

===== Metadata =====
Key                               Value
---                               -
created_time                     2023-11-12T14:37:30.107251751Z
custom_metadata                  <nil>
deletion_time                    n/a
destroyed                       false
version                         1

===== Data =====
Key                               Value
---                               -
password                        password
user                           postgres
```

Setting up Vault for Backend Role

Another mode for working with HashiCorp Vault is to create multiple Authentication Methods. This can help centralize our Identity & Access Management (IAM).

AppRole is used mainly for automatic system. However, in scope of this demo, we will use it to demonstrate Vault authentication and authorization capability.

Create AppRole Authentication

```
# Create a new role for our Backend
vault write -f auth/approle/role/backend-api

# By default, Vault generate a Role ID as one of the credentials.
vault read auth/approle/role/backend-api/role-id

# Sample output
#
# Key          Value
# ---          -
# role_id      00e45e77-547c-5b16-6f2a-f491f8401edc

# Make Vault generate a new Secret ID. This is the last part of this
```

```
authentication mode requirements.  
vault write -f auth/approle/role/backend-api/secret-id
```

```
# Sample output  
# Key          Value  
# ---          -----  
# secret_id    4255f32e-1fea-d70e-bb74-53d60bda6b2a  
# secret_id_accessor 5227cdef-30d0-5eb2-81d3-424add5d2613  
# secret_id_num_uses 0  
# secret_id_ttl    0s
```

Authorization

First, create a **Policy** to access our newly created secret. The file content of `backend-policy.hcl` would be:

```
path "backend-secret/data/*" {  
    capabilities = ["read"]  
}
```

Then to create a new policy based on this, we use

```
vault policy write backend-policy ./backend-policy.hcl
```

This will create a new Policy called `backend-policy`. Assigning this to our Backend role:

```
# A comma seperated string of policies  
vault write auth/approle/role/backend-api token_policies='backend-policy'  
  
vault read auth/approle/role/backend-api/policies  
# Sample output  
# Key          Value  
# ---          -----  
# token_policies ['backend-policy']
```

Getting credentials from Backend API

Inside this project `note_api`, our team includes a small Python FastAPI to demonstrate the login of external libraries to HashiCorp Vault.

Inside `app.py`, here is the code that fetch the `postgres` secret:

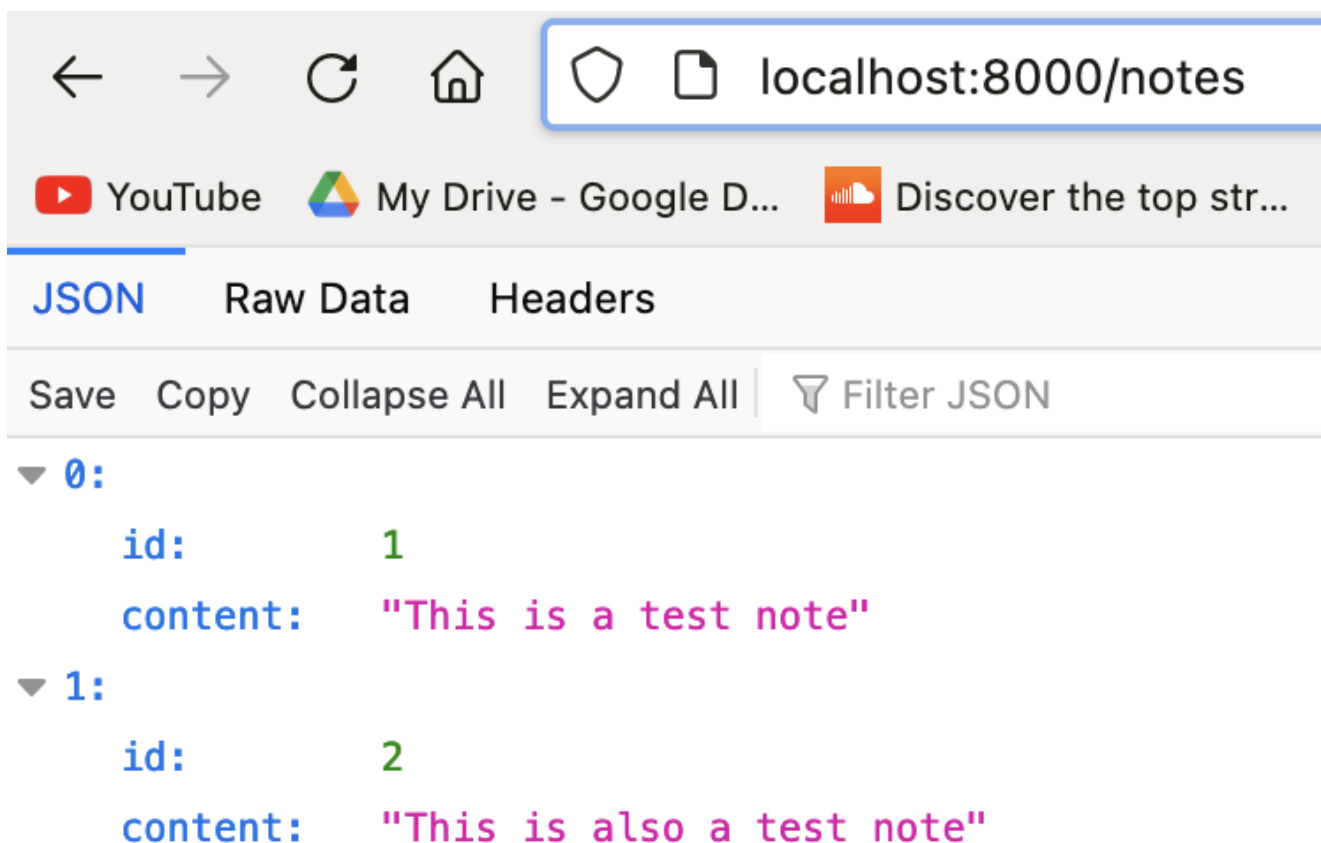
```
# app.py  
  
...  
# This value should be stored in $env
```

```
# Login using predefined AppRole
vault_client = hvac.Client(url="http://localhost:8200")
vault_client.auth.approle.login(
    role_id="00e45e77-547c-5b16-6f2a-f491f8401edc",
    secret_id="4255f32e-1fea-d70e-bb74-53d60bda6b2a",
)
...

# Reading the secret
secret_read_response = vault_client.read('/backend-secret/data/postgres')
if not secret_read_response:
    raise Exception("Cannot read secret from Vault")

username = secret_read_response['data']['data']['username']
password = secret_read_response['data']['data']['password']
```

If setup correctly, running the server that `http://localhost:8000/notes`, we can read the PostgreSQL table:



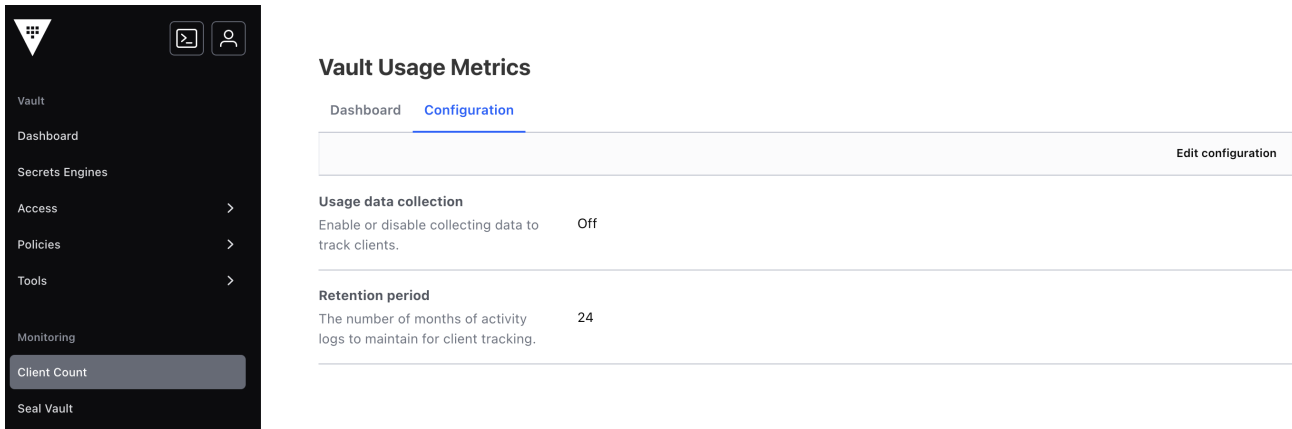
The screenshot shows a web browser window with the address bar displaying `localhost:8000/notes`. Below the address bar, there are navigation icons and a search bar. The main content area shows a JSON response with two entries:

```
0:
  id: 1
  content: "This is a test note"
1:
  id: 2
  content: "This is also a test note"
```

View access count

Using HashiCorp Vault, we can also monitor our client access for security purposes.

From the Vault's webserver, go to **Client Count > Configuration** and enabled **Usage data collection**.



Vault Usage Metrics

Dashboard **Configuration** [Edit configuration](#)

Usage data collection
Enable or disable collecting data to track clients. Off

Retention period
The number of months of activity logs to maintain for client tracking. 24

Then re-run the Backend API, this access will be recorded to the log:

Vault Usage Metrics

[Dashboard](#) Configuration

This dashboard will surface Vault client usage over time. Clients represent a user or service that has authenticated to Vault. Documentation is available [here](#). Date queries are sent in UTC.

Client counting start date

October 2023 [Edit](#)

This date is when client counting starts. Without this starting point, the data shown is not reliable.

FILTERS

Oct 2023 - Nov 2023 ▾

Vault client counts

A client is any user or service that interacts with Vault. They are made up of entity clients and non-entity clients. The total client count number is an important consideration for Vault billing.

Running client total

The number of clients which interacted with Vault during this date range.

Entity clients

1

Non-entity clients

0



Access can also be export for further investigation:

clients_by_namespace_October-2023-November-2023.csv

```
Namespace path,"Authentication method
*namespace totals, inclusive of auth method clients",Total clients,Entity clients,Non-entity
clients
root,*,1,1,0
root,auth/approle/,1,1,0
```