**HashiCorp Vault**

Step 1: Start the EC2 Instance 🡪 Connect 🡪 Copy SSH and paste in CMD.

Step 2: Start the Vault

**Code:** sudo systemctl start vault

(If you want to know the Status of the Vault

**Code:** sudo systemctl status vault )

Step 3: Unseal the vault three times:

**Enter the code:**

sudo systemctl start vault

vault operator unseal <unseal key 1>

vault operator unseal <unseal key 2>

vault operator unseal <unseal key 3>

Step 4: go to your browser:

**Enter the url to host the Vault UI:** http://<ec2 public ip>:8200

Step 5: enter the initial root token:

<generated root token>

**Entered Vault !**

**2. Github CICD**

**To create a role**

vault write auth/approle/role/github-role policies="github-policy"

**get role id**

vault read auth/approle/role/github-role/role-id

**get secret id:**

vault write -f auth/approle/role/github-role/secret-id

**Github token** (From github site)

<generated token>

**Why is GITHUB\_TOKEN hidden (\*\*\* masked)?**

* **Security reason**: GitHub **automatically detects anything** that looks like a sensitive secret — **Vault secrets, tokens, passwords**, etc.
* Even if you manually echo a secret, GitHub **redacts** (hides) the value as \*\*\* in the console output.
* **This is to prevent accidental secret leaks** if someone looks at the Actions logs.

✅ It's normal. It means **GitHub is protecting your secrets properly**.

**📌 Store These in GitHub Repository Settings → Secrets:**

| **GitHub Secret Name** | **Value** |
| --- | --- |
| VAULT\_ADDR | http://<ip>:8200 |
| VAULT\_ROLE\_ID | Role ID from above |
| VAULT\_SECRET\_ID | Secret ID from above |

**3. DYNAMODB X Vault**

**Geneartae credentials**

**Use in simple progeam(Example)**

!pip install boto3

import boto3

# Vault Generated Temporary Credentials

aws\_access\_key\_id = ""

aws\_secret\_access\_key = ""

region\_name = "ap-south-1"  # Or your AWS region

# Connect to DynamoDB

dynamodb = boto3.resource(

    'dynamodb',

    aws\_access\_key\_id=aws\_access\_key\_id,

    aws\_secret\_access\_key=aws\_secret\_access\_key,

    region\_name=region\_name

)

# Specify the DynamoDB Table

table = dynamodb.Table('Vault-Demo-Table')

# Member Details

members = [

    {"id": "1", "Name": "Alice", "Role": "Developer"},

    {"id": "2", "Name": "Bob", "Role": "Tester"},

    {"id": "3", "Name": "Charlie", "Role": "Manager"},

    {"id": "4", "Name": "Diana", "Role": "DevOps"},

    {"id": "5", "Name": "Eve", "Role": "Designer"}

]

# Insert each member into the table

for member in members:

    table.put\_item(Item=member)

print("✅ Successfully added 5 members into DynamoDB!")

**4. Cloud-Native Application Secrets Management:**

(Retieveing Passwords and using the token in the Application to input the creds)

**Code: (in terminal)**

vault kv get -mount="app-secrets" "myapp/config"(To display the secret)

**📚 What is the use of "Cloud-Native Application Secrets Management with HashiCorp Vault"?**

It means:

👉 **Securely storing and managing** sensitive information (like API keys, passwords, database credentials) that your cloud-native application needs — without hardcoding them inside the app.

**🛡️ Why it's useful:**

| **Problem** | **How Vault Helps** |
| --- | --- |
| Developers often hardcode passwords, API keys inside the app code (very risky). | Vault **stores secrets securely** in a central place. |
| Secrets need to change sometimes (rotation). | Vault can **update/rotate secrets** automatically. |
| Different apps/users should have **limited access** to secrets. | Vault uses **policies** to control who can access what. |
| Secrets leak is a big security threat. | Vault **encrypts secrets** and provides **audit logs**. |
| In cloud-native apps (like on AWS, Azure, GCP), you often have **dynamic, scalable** apps that need safe access to secrets. | Vault provides **API-based secure access** for secrets retrieval. |

**🏆 In short:**

* No secrets inside your app code.
* Rotate/update secrets without touching apps.
* Fine-grained access control over secrets.
* Full auditability (who accessed what and when).
* Highly secure and scalable for cloud-native environments.

**🔥 Real-life example:**

Imagine a cloud app hosted on AWS needs a **database password**.

* Without Vault: You manually insert the password inside your app code.
* With Vault: Your app **requests the password from Vault at runtime** — and you can change the password anytime in Vault without changing the app!

**5. Secure Access Management for Microservices with HashiCorp Vault**

Step 1: Set up role

**write auth/approle/role/micro-approle policies="microservice-a-policy"**

Step 2: Get the Role ID

**vault read auth/approle/role/micro-approle/role-id**

**Step 3: Get Seret ID**

**vault write -f auth/approle/role/micro-approle/secret-id**

**Step 4: Get Token**

**vault write auth/approle/login \**

**role\_id="" \**

**secret\_id=""**

This will give you a **Vault Token** like:

{

"auth": {

"client\_token": "s.XYZAbcTokenHere",

"accessor": "abcd1234",

"policies": ["default", "microservice-a-policy"],

...

}

}

The client\_token (s.XYZAbcTokenHere) is **what your microservice will use** to read secrets!

the microservice first authenticates to Vault using a token or AppRole method. Once authenticated, the service sends an API request to Vault to fetch required secrets like database usernames and passwords. These secrets are then used inside the microservice for secure connections without hardcoding credentials. Ideally, instead of manually handling tokens, a Vault Agent or dynamic authentication method should be used to automatically manage tokens securely in production environments.