HOW-TO: Setup GitHub actions to access Vault secrets

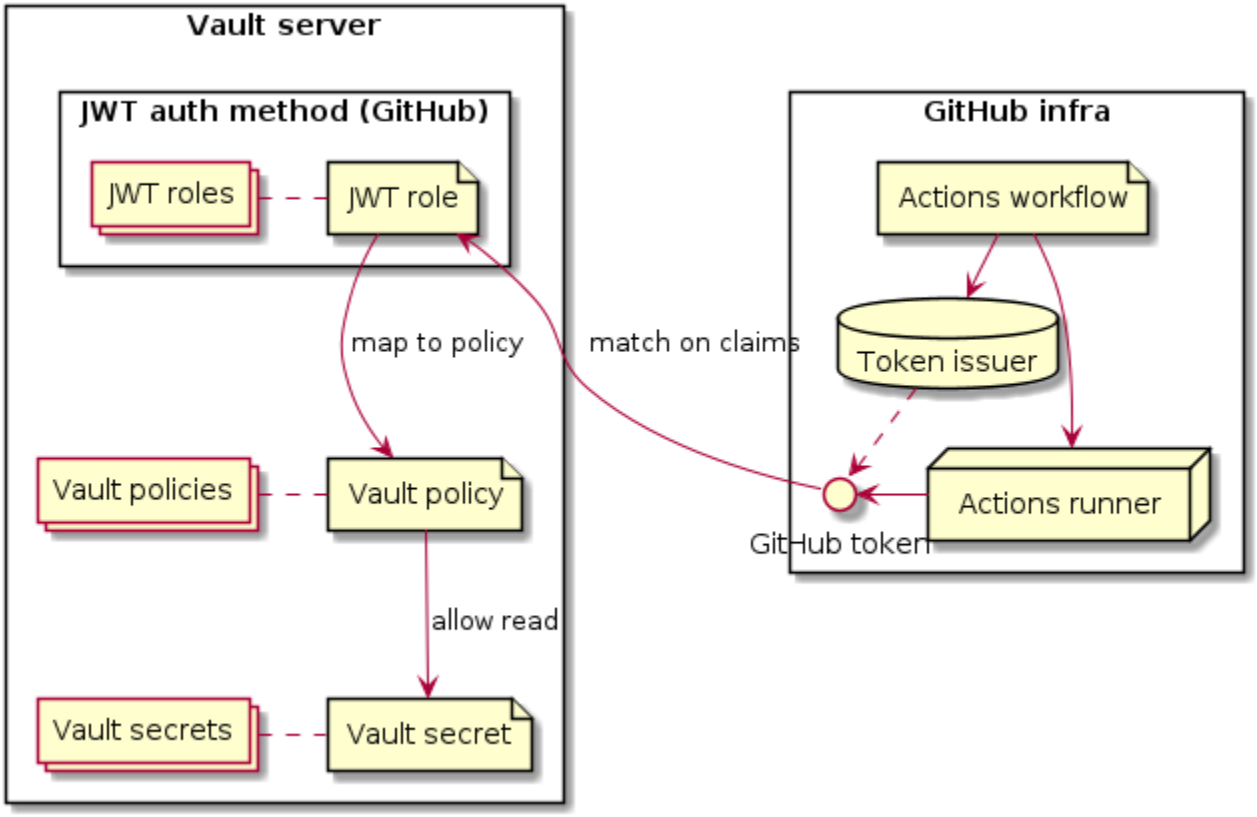
Background

- I have a Vault server and want GitHub actions to access Vault secrets.

Objectives

1. Setup the Vault server for JWT authentication method to trust the GitHub token issuer.
2. Create the Vault policy for the secret.
3. Create the JWT role to map the GitHub token to the Vault policy.
4. Create the GitHub actions workflow to access the Vault secret.

Interaction of GitHub and Vault in accessing Vault secrets



Procedure

Setting up the Vault server for JWT authentication

Enabling the JWT authentication method

	Steps
1	Login to the Vault UI with permissions to add new authentication method.
2	Navigate to **Access Auth methods Enable a new Auth method**.
3	Select the **JWT** option and click Next. ⚠ Keep the default **`jwt`** path, otherwise the Hashicorp Vault Secrets action has trouble logging in.
4	Set the **OIDC discovery URL** to: https://token.actions.githubusercontent.com
5	Expand the **JWT Options** section. Set the **Bound issuer** to: https://token.actions.githubusercontent.com
6	Click Save.

Creating the policy for the secret access

	Steps
1	Login to the Vault UI with permissions to add new policies.
2	Navigate to **Policies Create ACL policy**.
3	Provide a name to the policy. This will be referenced in the JWT role created in a later step.

4	Provide the policy block, which references the secrets and their capabilities.
	
	Example policy: read_secret_mysecret
	
	`path "secret/data/mysecret"`
	`{`
	`  capabilities = ["read"]`
	`}`

5	Click Save.

Create the JWT role

These steps can only be done through the Vault CLI tool.

	Steps
1	Login to Vault CLI with permissions to add JWT roles. Run `vault login -method=github` Provide your GitHub personal access token at the prompt.

2	Prepare the Vault role. See Vault documentation on all the fields.
	
	Example Vault role for test-vault repo
	
	`{`
	`  "role_type": "jwt",`
	`  "policies": ["read_secret_mysecret" ],`
	`  "bound_audiences": "https://github.com/tenzin-io",`
	`  "user_claim": "repository",`
	`  "bound_claims_type": "string",`
	`  "bound_claims":{`
	`  "repository": "tenzin-io/test-vault"`
	`  }`
	`}`
	
	See the GitHub documentation for all the possible fields to use in `bound_claims` to perform the role match.

3	Write the Vault role to the authentication method. **Example of vault write command** `vault write auth/jwt/role/test-vault - < test-vault.json` The role name in this example is `test-vault` and the role payload is in the JSON file. This role name needs to be referenced in the GitHub actions workflow file.

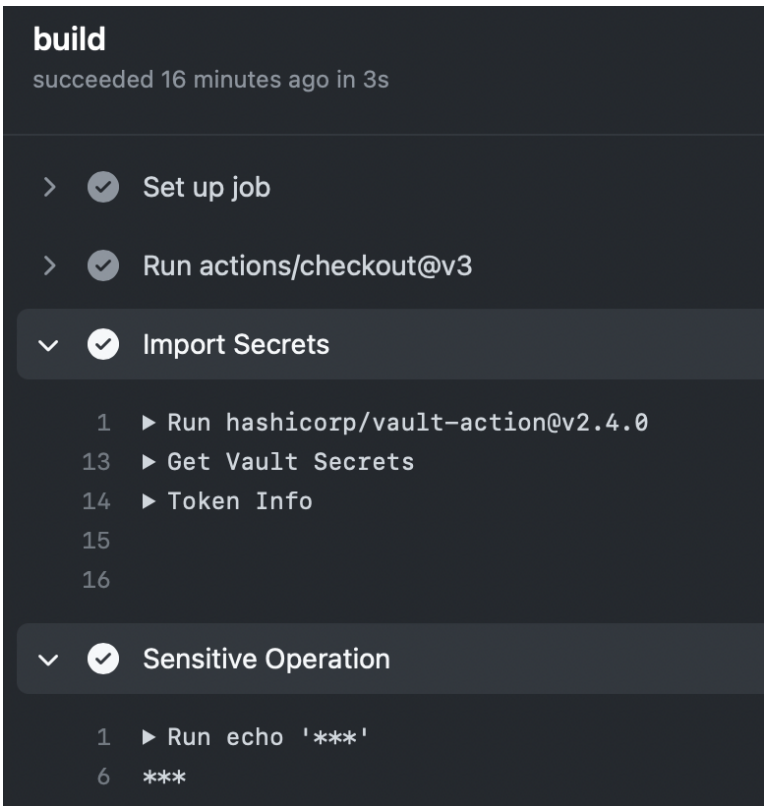
Setting up GitHub workflow to access Vault secrets

	Steps
1	Create the workflows YAML file in the GitHub repository. In the above example the repository is: tenzin-io/test-vault

2	The example YAML file below:
	
	GitHub workflow: test-vault.yaml
	
	`name: Testing vault`
	
	`on:`
	`  # Triggers the workflow on push or pull request events but only for the "main" branch`
	`  push:`
	`  branches: [ "main" ]`
	`  pull_request:`
	`  branches: [ "main" ]`
	`  # Allows you to run this workflow manually from the Actions tab`
	`  workflow_dispatch:`
	
	`jobs:`
	`  build:`
	`  # The type of runner that the job will run on`
	`  runs-on: [ self-hosted, Linux, ARM64 ]`
	`  permissions:`
	`  contents: read`
	`  id-token: write`
	`  steps:`
	`  - uses: actions/checkout@v3`
	
	`  - name: Import Secrets`
	`  uses: hashicorp/vault-action@v2.4.0`
	`  id: secrets`
	`  with:`
	`  url: https://vault.tenzin.io`
	`  method: jwt`
	`  role: test-vault`
	`  secrets: |`
	`  secret/data/mysecret secret_one | SECRET_ONE ;`
	
	`  - name: Sensitive Operation`
	`  run: "echo '${{ steps.secrets.outputs.SECRET_ONE }}'"`
	
	Source: https://github.com/tenzin-io/test-vault/tree/main/.github/workflows

3	The `permissions` block modifies the default permissions on the `GITHUB_TOKEN`.
	
	Modifying the GITHUB_TOKEN permissions
	
	`jobs:`
	`  build:`
	`  runs-on: [ self-hosted, Linux, ARM64 ]`
	`  permissions:`
	`  contents: read`
	`  id-token: write`

4	Add the step that uses the `hashicorp/vault-action` GitHub action. See more details on the action documentation
	
	Vault action
	
	`- name: Import Secrets`
	`  uses: hashicorp/vault-action@v2.4.0`
	`  id: secrets`
	`  with:`
	`  url: https://vault.tenzin.io`
	`  method: jwt`
	`  role: test-vault`
	`  secrets: |`
	`  secret/data/mysecret secret_one | SECRET_ONE ;`
	
	The `role` name should correspond to the Vault role name that was created on the JWT auth path.
	
	See the `vault write auth/jwt/role/...` command from an earlier step.

5	Perform the commit, which should automatically dispatch the workflow to a GitHub actions runner. Example workflow output: 

Appendix

Vault troubleshooting

- There were times where it was hard to decipher exactly why I was getting HTTP `Bad Request` or `Unauthorized` error message in the GitHub actions log.
- I found the below debug commands helpful.

Helpful debugging commands

```
# this enables the audit logging
## option log_raw=true is very important, this decodes the request and response data field,
## otherwise the error messages are in some unhelpful encrypted form
vault audit enable file file_path=/vault/logs/audit.log log_raw=true

# the log file will be found on the vault server itself,
# not on the machine from which you ran the vault command.
```