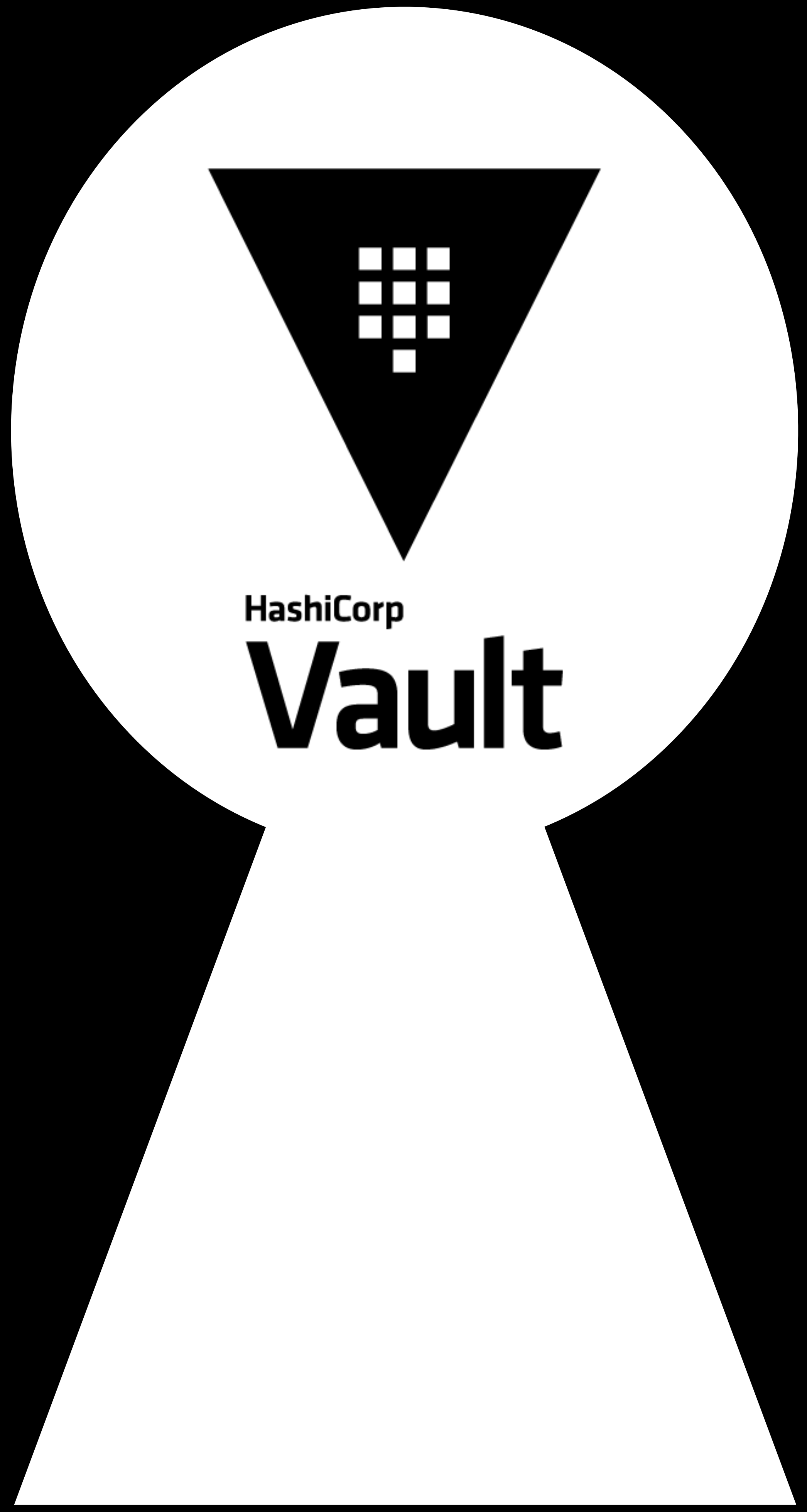
HashiCorp
Vault

Daniel Hinojosa
@dhinojosa

Agenda

What is in this half day

- Description
- Storage of Confidential Material
- Sealing & Unsealing
- Secrets & Dynamic Secrets
- Using the UI
- Policies
- Authentication Methods
- AWS & Kubernetes



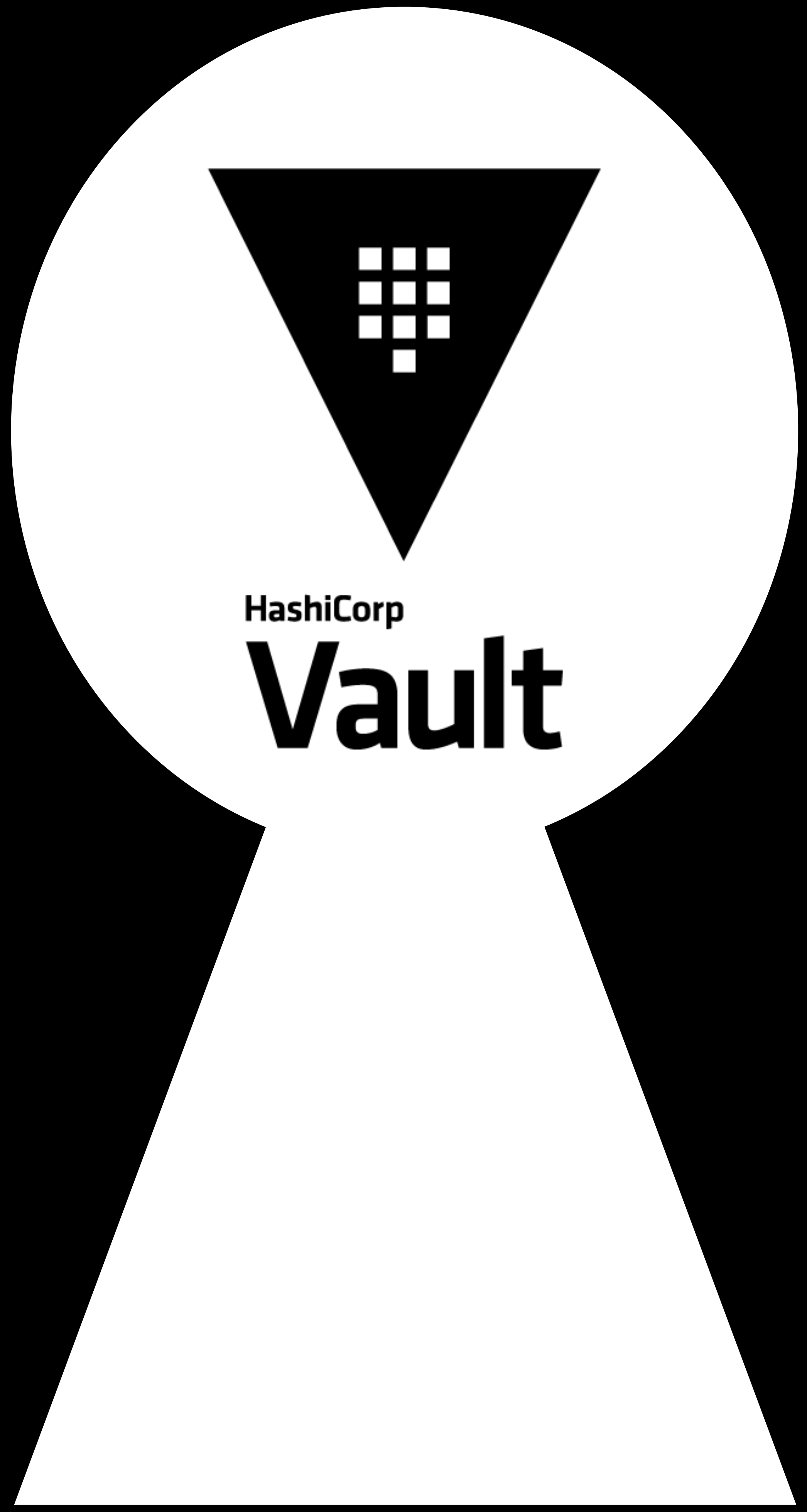
Introduction

What is Vault?

- Integrate with external systems using pluggable components, called:
    - Secrets engines
    - Authentication methods
- Purpose of those components is to manage and protect your secrets in dynamic infrastructure
    - (e.g. database credentials, passwords, API keys)

Vault Architecture

- Vault operates as server/client
- Server is the only one that interacts with backend storage
- Client interacts with the Server through TLS (Transport Layer Security)



Installing Vault

Setting up Vault

- Install Ubuntu, Debian, Fedora, RHEL, Amazon Linux, and other distributions using GPG key and repositories

- Find and install from the Hashicorp Download site, https://www.vaultproject.io/downloads

- On MacOSX

```
brew install vault
```

- On Windows

```
choco install vault
```

Verify The Installation

This is how it shows from your local installation

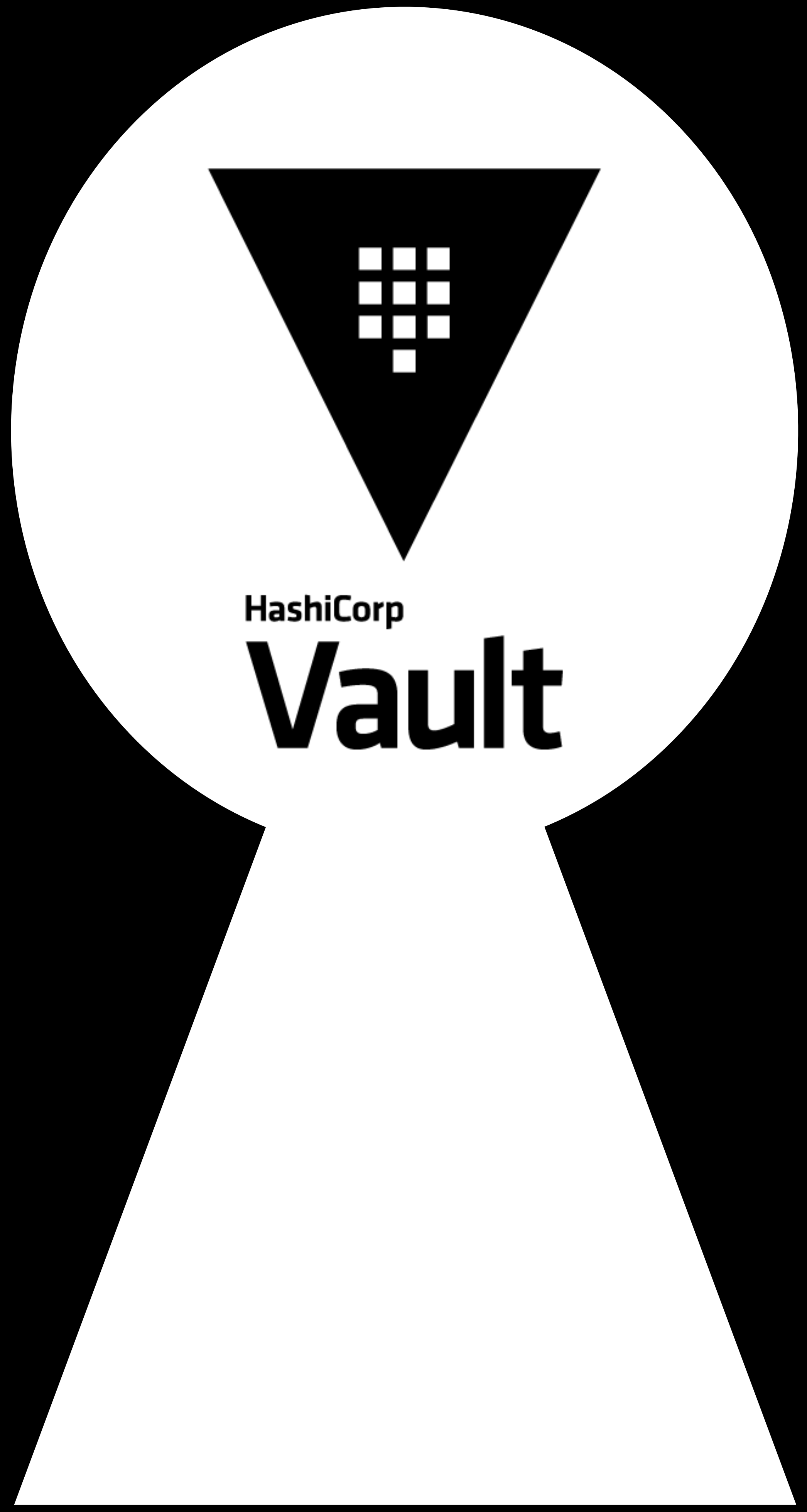
```
$ vault

Usage: vault <command> [args]

Common commands:
    read        Read data and retrieves secrets
    write       Write data, configuration, and secrets
    delete      Delete secrets and configuration
    list        List data or secrets
    login       Authenticate locally
    server      Start a Vault server
    status      Print seal and HA status
    unwrap      Unwrap a wrapped secret

Other commands:
    audit          Interact with audit devices
    auth           Interact with auth methods
    lease          Interact with leases
    operator       Perform operator-specific tasks
    path-help      Retrieve API help for paths
    policy         Interact with policies
    secrets        Interact with secrets engines
    ssh            Initiate an SSH session
    token          Interact with tokens
```

Setting up Auto-Completion

- Autocompletion can be applied if you have bash, fish, or zsh as your shell
- An error will occur if you do not have any of these shells

```
$ vault -autocomplete-install
```

- Once applied you can type: **`vault <TAB>`** and be offered all possible commands



Dev Server

Running a Development Server

- The Dev server is a built-in, pre-configured server
- Not very secure, no TLS (Transport Layer Security) over the wire
- Useful for playing with Vault locally

```
$ vault server -dev
```

Root Token & Unseal Key

- After Running the Server, you will be provided with two items:
    - Root Token
        - An initial token used to create tokens, authentications, policies, etc.
    - Unseal Key
        - A simple key used to seal and unseal the vault.
        - A production instance is more complicated, this is just used to seal up the vault to prevent any use

After Starting the Dev Server

Example of what you see after starting the server

```
The unseal key and root token are displayed below in case you want to
seal/unseal the Vault or re-authenticate.

Unseal Key: 1+yv+v5mz+aSCK67X6slL3ECxb4UDL8ujWZU/ONBpn0=
Root Token: s.XmpNPoi9sRhYtdKHaQhkHP6x
```

The Dev Server is not meant for production

Mapping the Token and Address

- When you run the Dev Server it displays the Vault Address

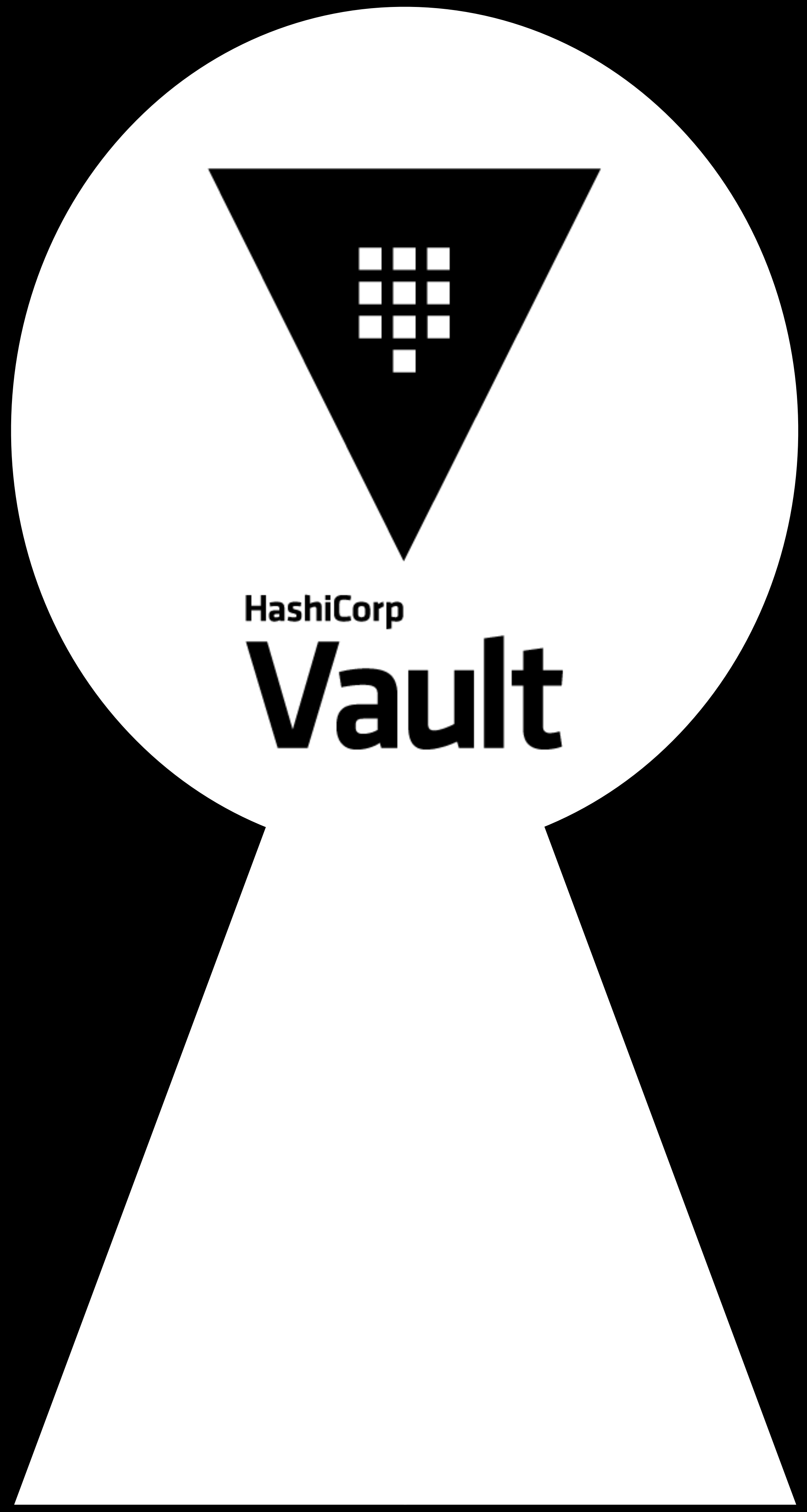
```
You may need to set the following environment variable:

    $ export VAULT_ADDR='http://127.0.0.1:8200'
```

- Do what it says, export the address so that it can be referred to by the `vault` client

```
$ export VAULT_ADDR='http://127.0.0.1:8200'
```

Verify that Everything Runs

```
$ vault status

Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.5.0
Cluster Name    vault-cluster-4d862b44
Cluster ID      92143a5a-0566-be89-f229-5a9f9c47fb1a
HA Enabled      false
```



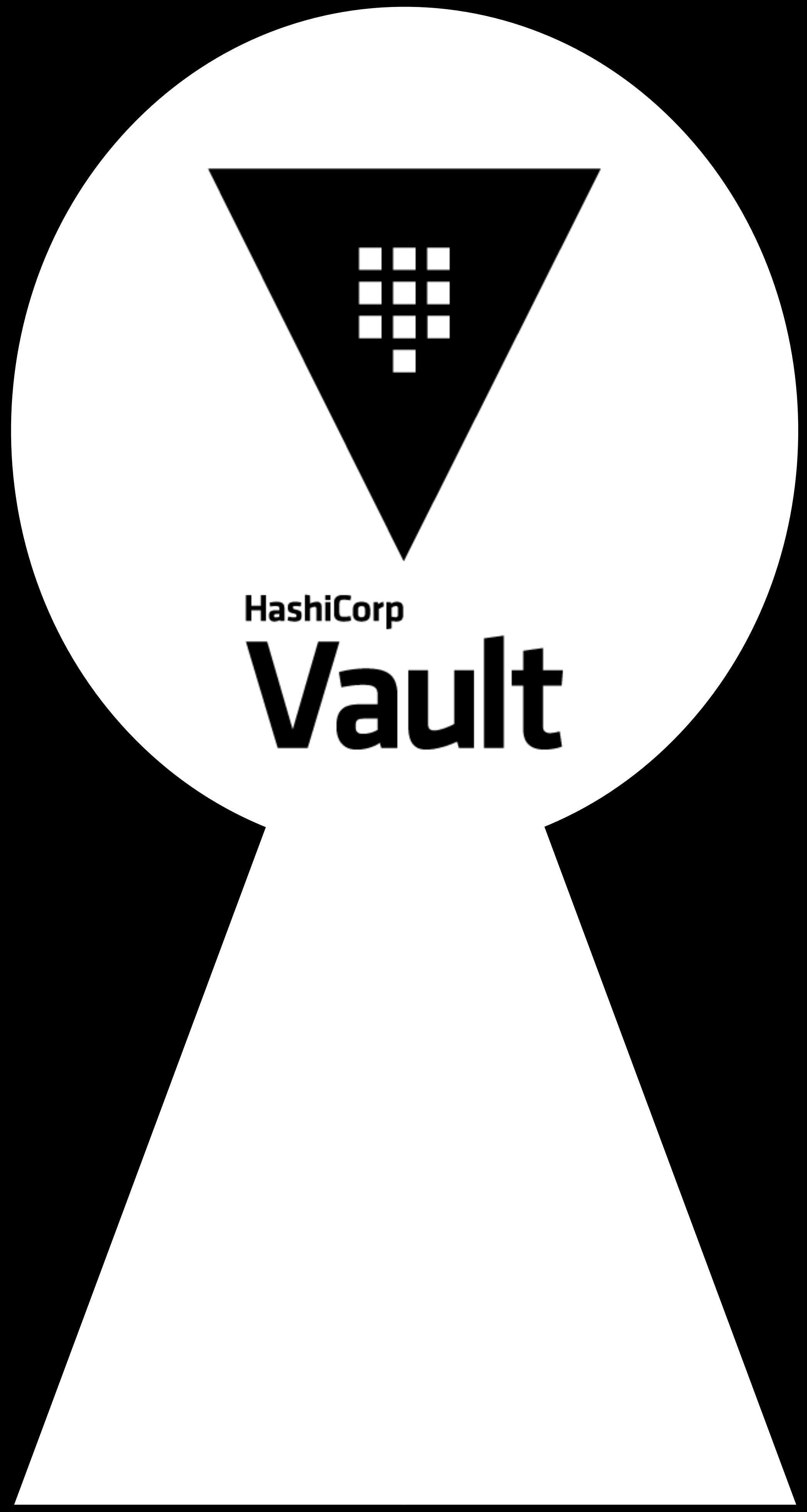
Storage of Confidential Material

Storage

Securing of confidential material

- Tokens
- Passwords
- Certificates
- API keys
- SSH Keys

- Dynamic Secrets
  - AWS Access IDs
  - Databases
  - Message Queues



Storing Secrets

Writing a Secret

- kv is a storage type, here we are putting a foo and world in the path secret and the identifier hello

```
vault kv put secret/hello foo=world
```

- You can also write multiple pieces of data

```
vault kv put secret/hello foo=world excited=yes
```

Reading a Secret

- `kv` is a storage type, here we are getting the secret key values that are stored in the `secret/hello` path

```
vault kv get secret/hello
```

- You can effectively "fish" out the specific field in the path

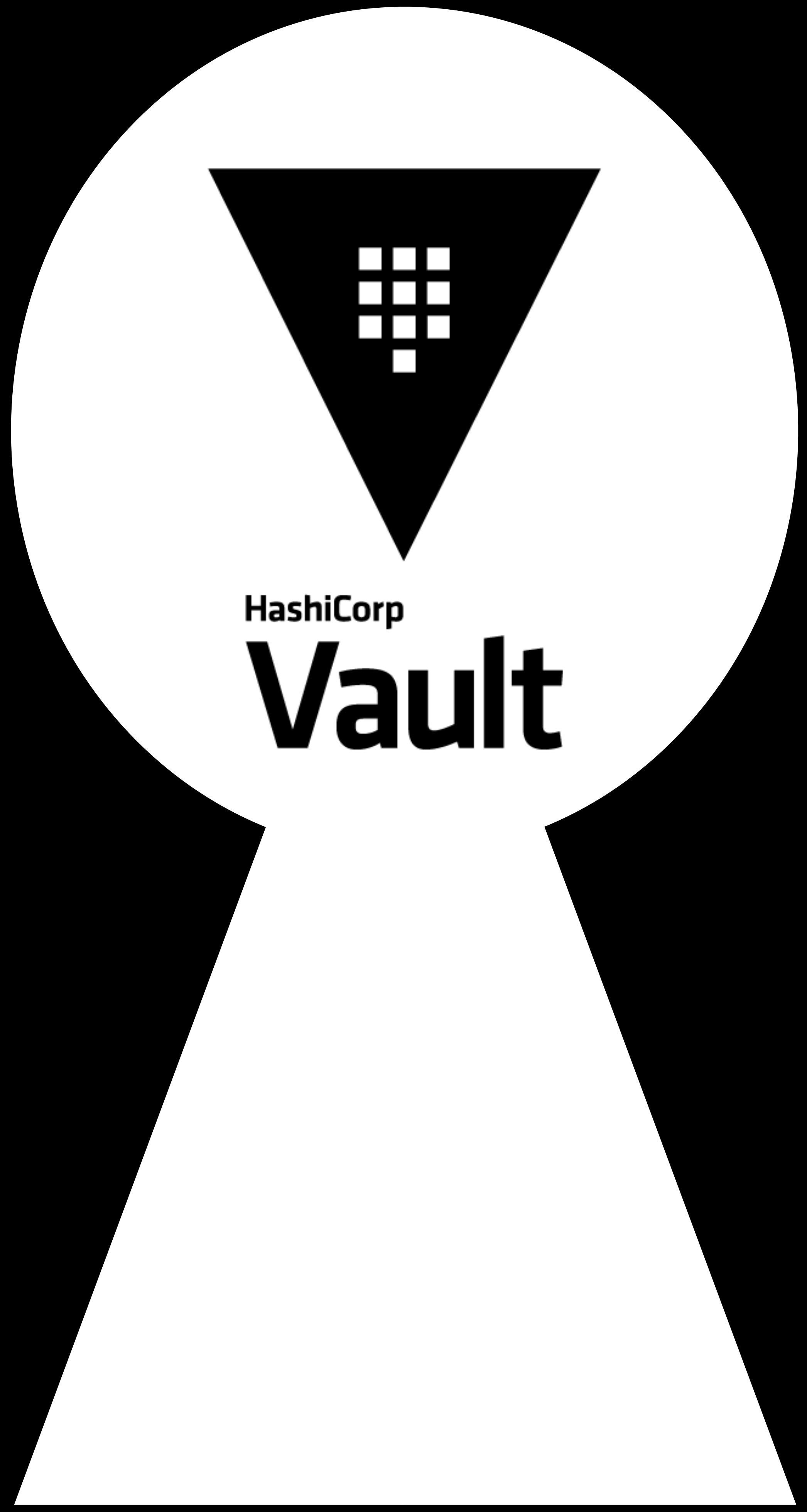
```
vault kv get -field=excited secret/hello
```

Deleting a Secret

- `kv` is a storage type, here we are deleting the `secret/hello` path

```
vault kv delete secret/hello
```

Lab 1: Putting Secrets into Vault



Dynamic Secrets

Dynamic Secrets

- Secrets can be automatically created and used for short during of time
- Vault has built-in revocation mechanisms, dynamic secrets can be revoked immediately after use.
- This minimizes the amount of time the secret existed
- While we will be doing AWS, there are secret engines for Oracle, MySQL, etc. You can refresh access constantly

Enabling AWS

- We would need to enable a different engine, this time the `aws` secrets engine

```
$ vault secrets enable -path=aws aws
```

- Specify the AWS Access Key and Secret Key that will procure other keys

```
$ vault write aws/config/root \
    access_key=AKIAI4SGLQPBX6CSENIQ \
    secret_key=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 \
    region=us-east-1
```

Specifying a Role

- Every new access key probably should ***not*** have rights to everything
- In the following example, this has EC2 access only, we can make a role out of this

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1426528957000",
      "Effect": "Allow",
      "Action": ["ec2:*"],
      "Resource": ["*"]
    }
  ]
}
```

Write the Role in Vault

- We are then bind this role to a path so we can refer to it when we request a new Access Key
- We write to `aws/roles/:name` where `:name` is your unique name that describes the role

```
vault write aws/roles/all-ec2-actions \
        credential_type=iam_user \
        policy_document=-<<EOF
   {
      "Version": "2012-10-17",
      "Statement": [
        {...}
      ]
   }
EOF
```

Generating a Secret

- We can now create a new access key and secret key based on the role

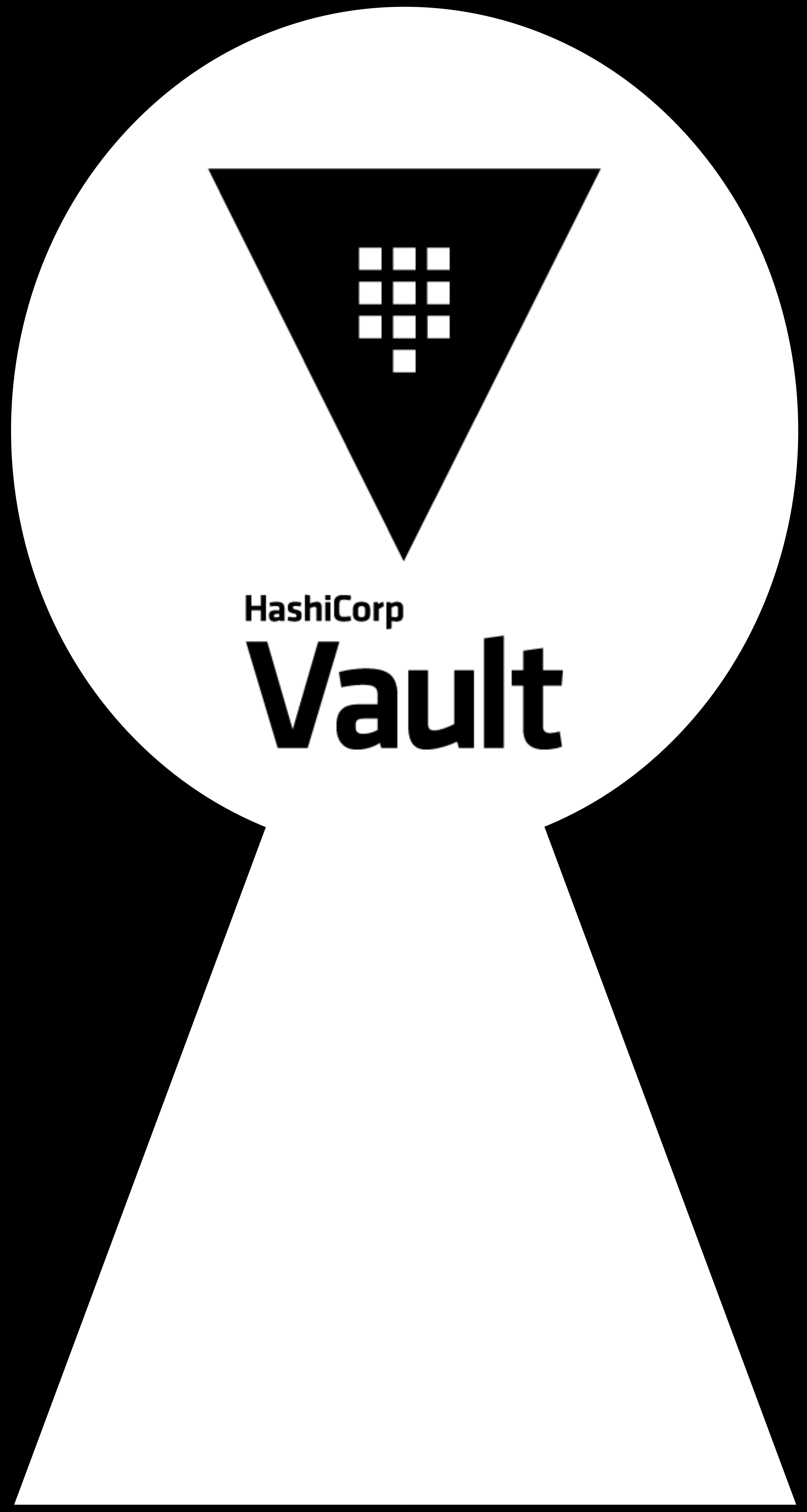
```
vault read aws/creds/all-ec2-actions
Key                Value
---                -----
lease_id           aws/creds/all-ec2-actions/0bce0782-32aa-25ec-f61d-
c026ff22106e
lease_duration     768h
lease_renewable    true
access_key         AKIAJELUDIANQGRXCTZQ
secret_key         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
security_token     <nil>
```

Revoking a Secret

- At some point you may want to remove access for that key
- You can do that with lease revoke

```
$ vault lease revoke aws/creds/my-role/0bce0782-32aa-25ec-f61d-
c026ff22106
Success! Revoked lease: aws/creds/my-role/0bce0782-32aa-25ec-f61d-
c026ff22106e
```

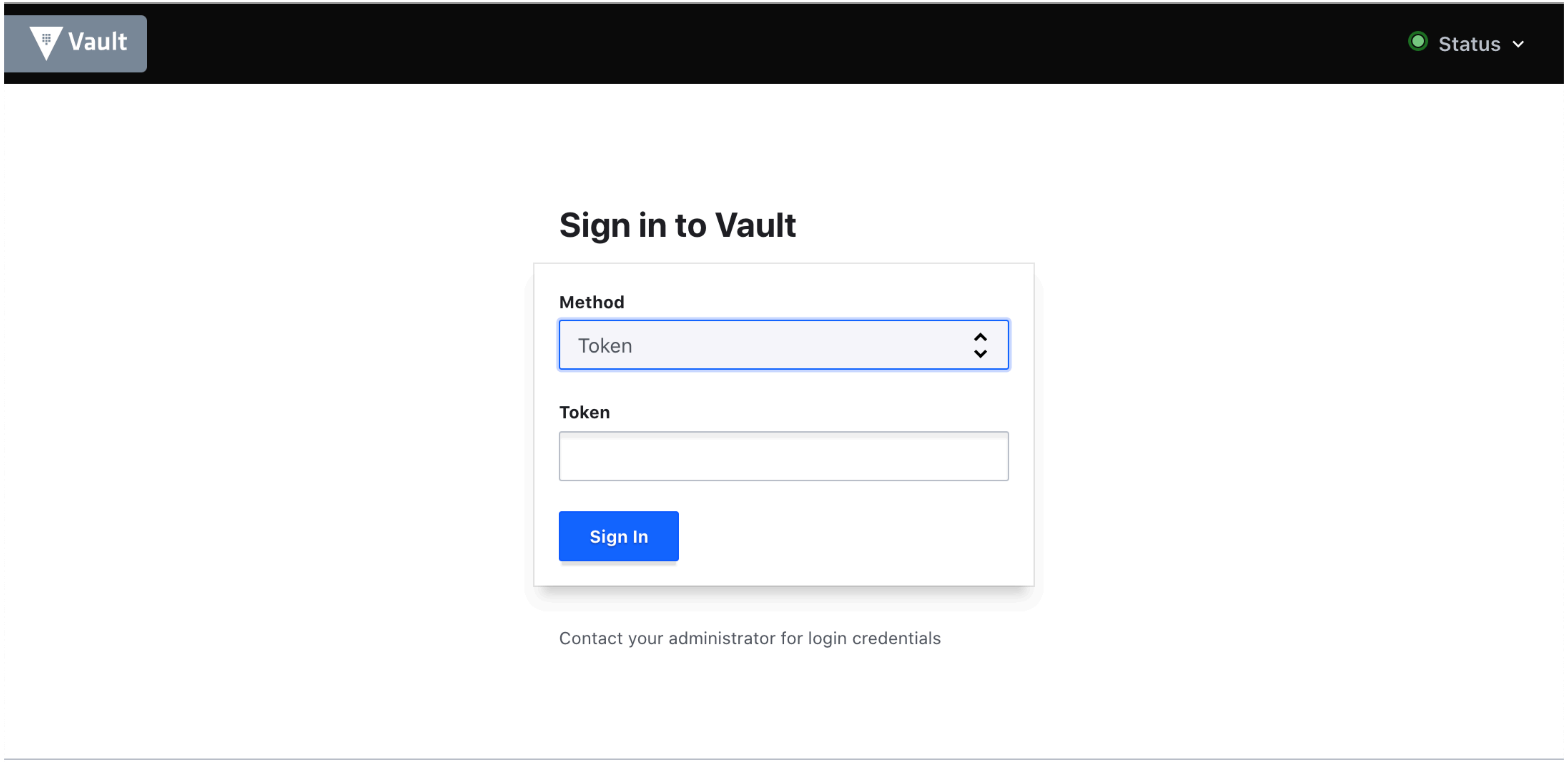
Lab 2: Create an AWS Dynamic Secret



Using the Web UI

Web UI

- Pleasant Web Interface
- Easily create, read, update, and delete secrets, authenticate, unseal, and more with the Vault UI.
- `http://127.0.0.1:8200/ui`
- Log in with any of authentication that you set up like using a standard token, username, ldap, okta, oidc, or GitHub

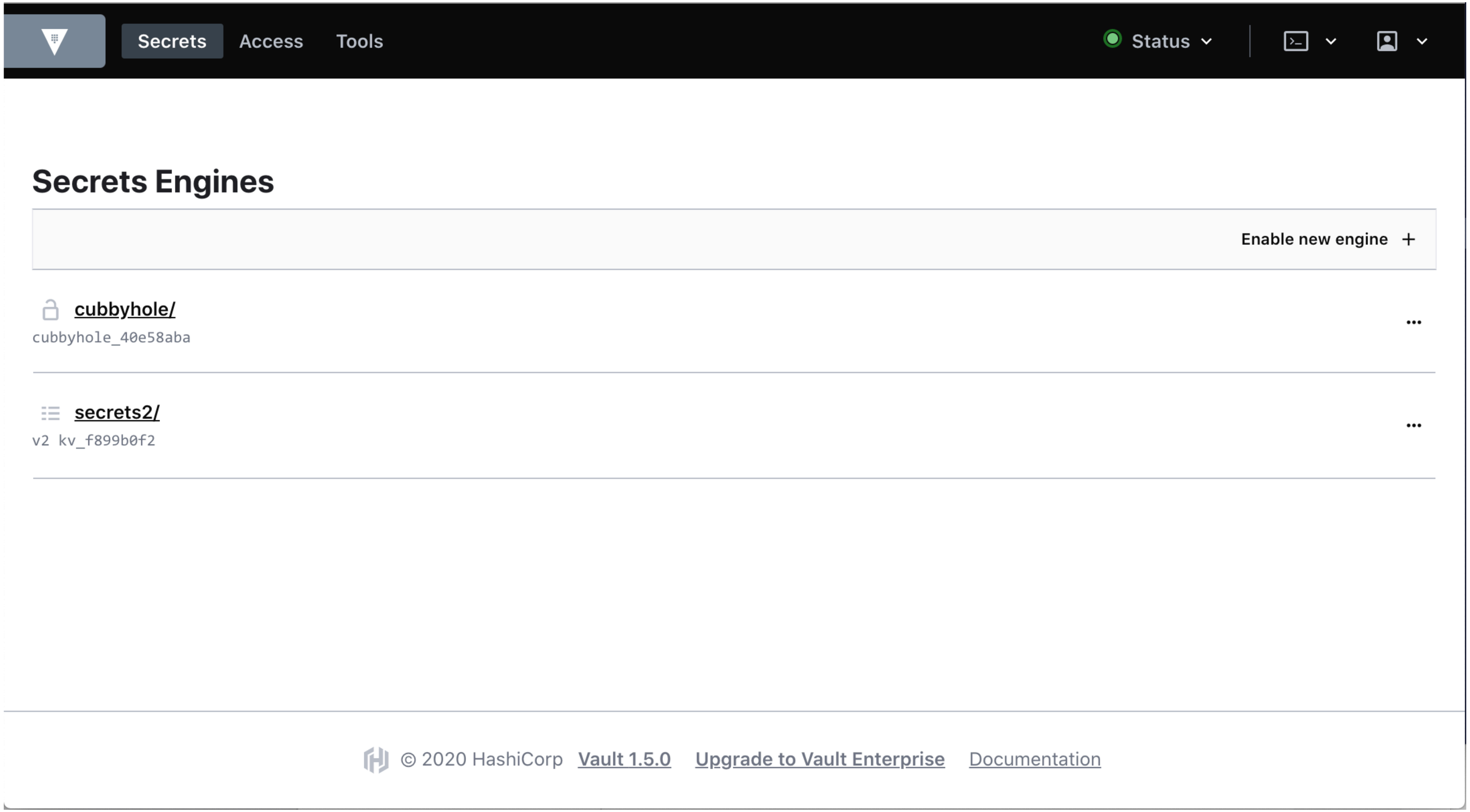


Sign in to Vault

Method: Token

Token

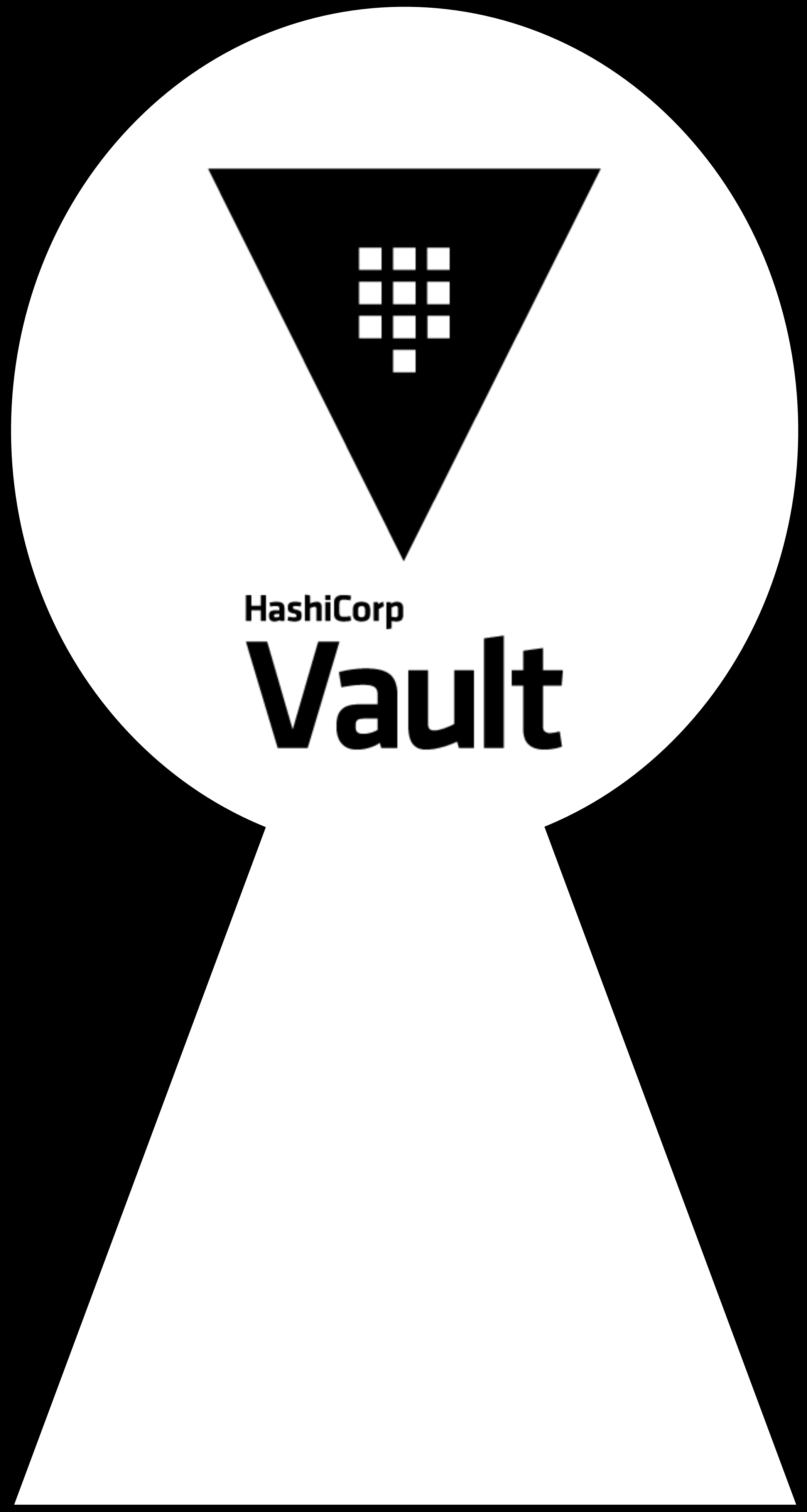
Contact your administrator for login credentials



Secrets Engines

cubbyhole/
cubbyhole_40e58aba

secrets2/
v2 kv_f899b0f2



Authentication

Authentication Concepts

- Verification from:
  - Inner (token) system
  - External system
- Various Authentication Methods
  - Github, LDAP, AppRole, etc.

Authentication using Tokens

- Token authentication is automatically enabled
- When you start the dev server, the output displayed a root token
- The Vault CLI read the root token from the `$VAULT_TOKEN` environment variable
- The `root` token is given a root policy which can do anything within vault
- Root tokens are nonces, used only once in production

Authentication using Tokens

- This token is a child of the *root token*
- by default, it inherits the policies from its parent.

```
$ vault token create
Key                  Value
---                  -----
token                s.iyNUhq8Ov4hIAx6snw5mB2nL
token_accessor       maMfHsZfwLB6fi18Zenj3qh6
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Logging In using Tokens

- Once a token is created it can be used to login
- Once logged in, it will automatically use this request

```
$ $ vault login s.iyNUhq8Ov4hIAx6snw5mB2nL

Success! You are now authenticated. The token information displayed
below is already stored in the token helper. You do NOT need to run
"vault login" again. Future Vault requests will automatically use this
token.
```

Custom Authentication is just Tokens

- You may authenticate using something like GitHub, Vault generates a unique access token for you to use for future requests.
- The CLI automatically attaches this token to requests, but if you're using the API you'll have to do this manually.

Creating Userpass Authentication

- Most Authentication must be enabled before use.
- Note: Paths with `sys`, as a system call which can only be allowed by root or a root token

```
vault auth enable userpass
```

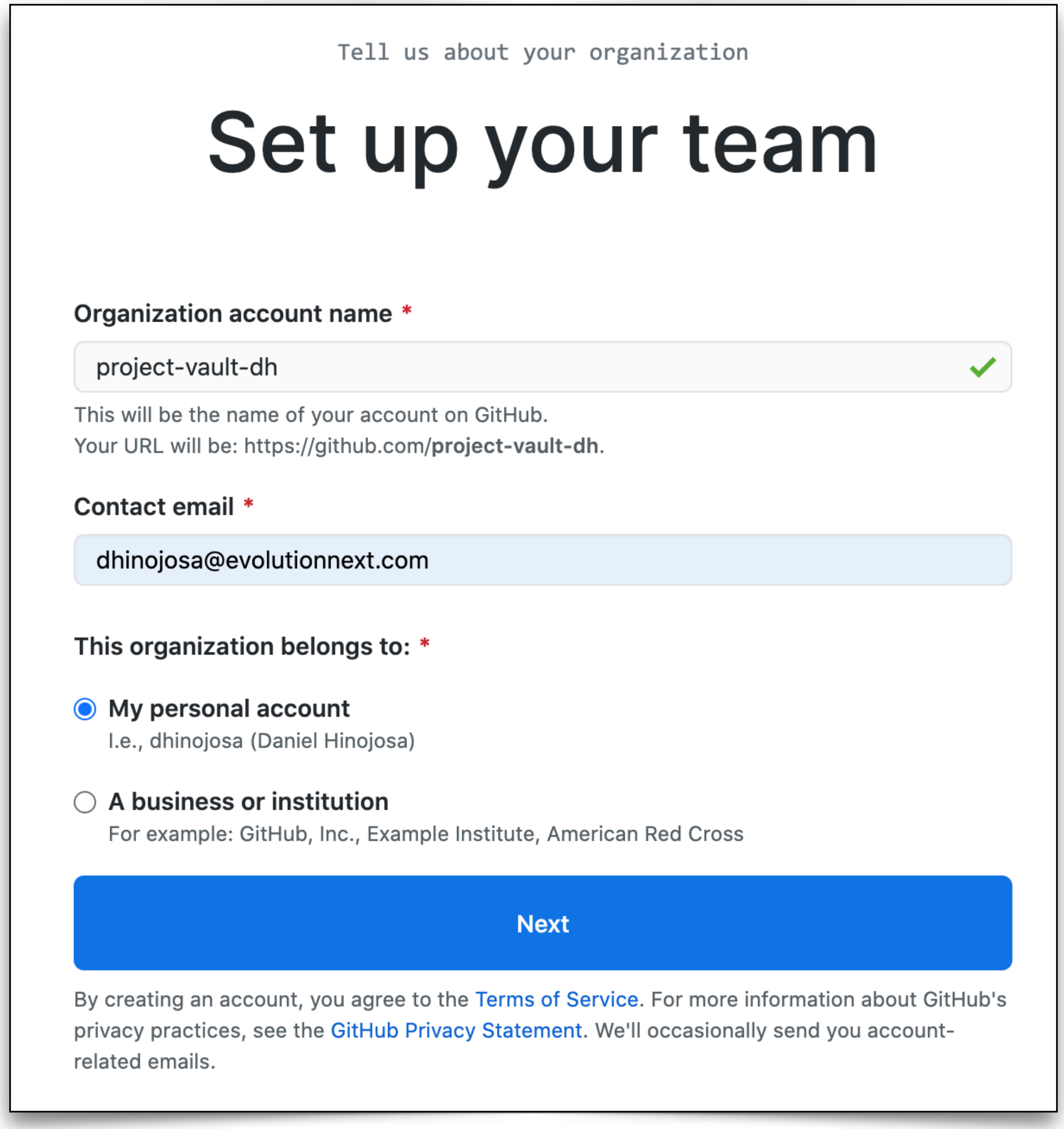
```
vault write auth/userpass/users/priva \
    password=foo \
    policies=admins
```

- To get help about the path and what it entails, one can use `path-help`

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

Creating Github Organization

- You can login as a Github user and create an organization
  - https://github.com/settings/organizations
  - Select your level or organization
  - Add members to your organization



Tell us about your organization

Set up your team

Organization account name *

project-vault-dh ✓

This will be the name of your account on GitHub.
Your URL will be: https://github.com/**project-vault-dh**.

Contact email *

dhinojosa@evolutionnext.com

This organization belongs to: *

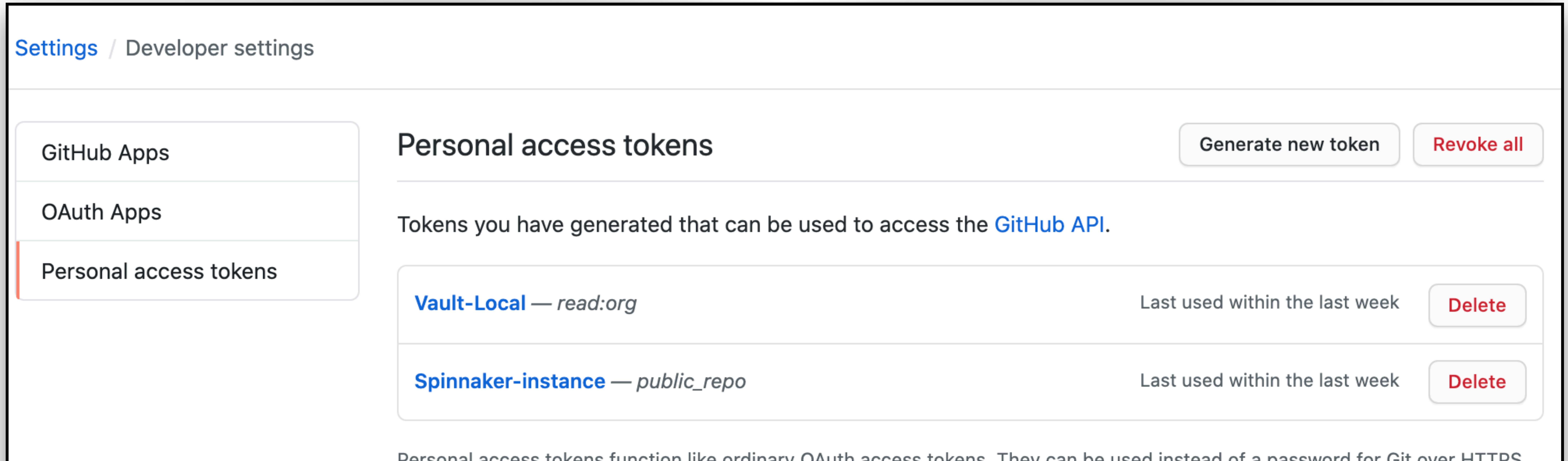
● My personal account
I.e., dhinojosa (Daniel Hinojosa)

○ A business or institution
For example: GitHub, Inc., Example Institute, American Red Cross

By creating an account, you agree to the Terms of Service. For more information about GitHub's privacy practices, see the GitHub Privacy Statement. We'll occasionally send you account-related emails.

Creating Github Personal Access Token

- You can login as a Github user and create a personal access token
  - https://github.com/settings/tokens
  - Generate and Copy Token



Personal access tokens

Tokens you have generated that can be used to access the GitHub API.

Vault-Local — *read:org* Last used within the last week

Spinnaker-instance — *public_repo* Last used within the last week

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS

Establishing Github Authentication

- Enable Github

```
vault auth enable github
```

- Once an organization is created, tell Vault about it

```
vault write auth/github/config organization=project-vault-dh
```

- Make a login with your Personal Access Token you created

- Reminder: This is not the Vault Token, a Vault Token gets generated after the initial login

```
vault login -method=github
token="19ca41bd6018ff94801d6150256268ea87dbfc94"
```

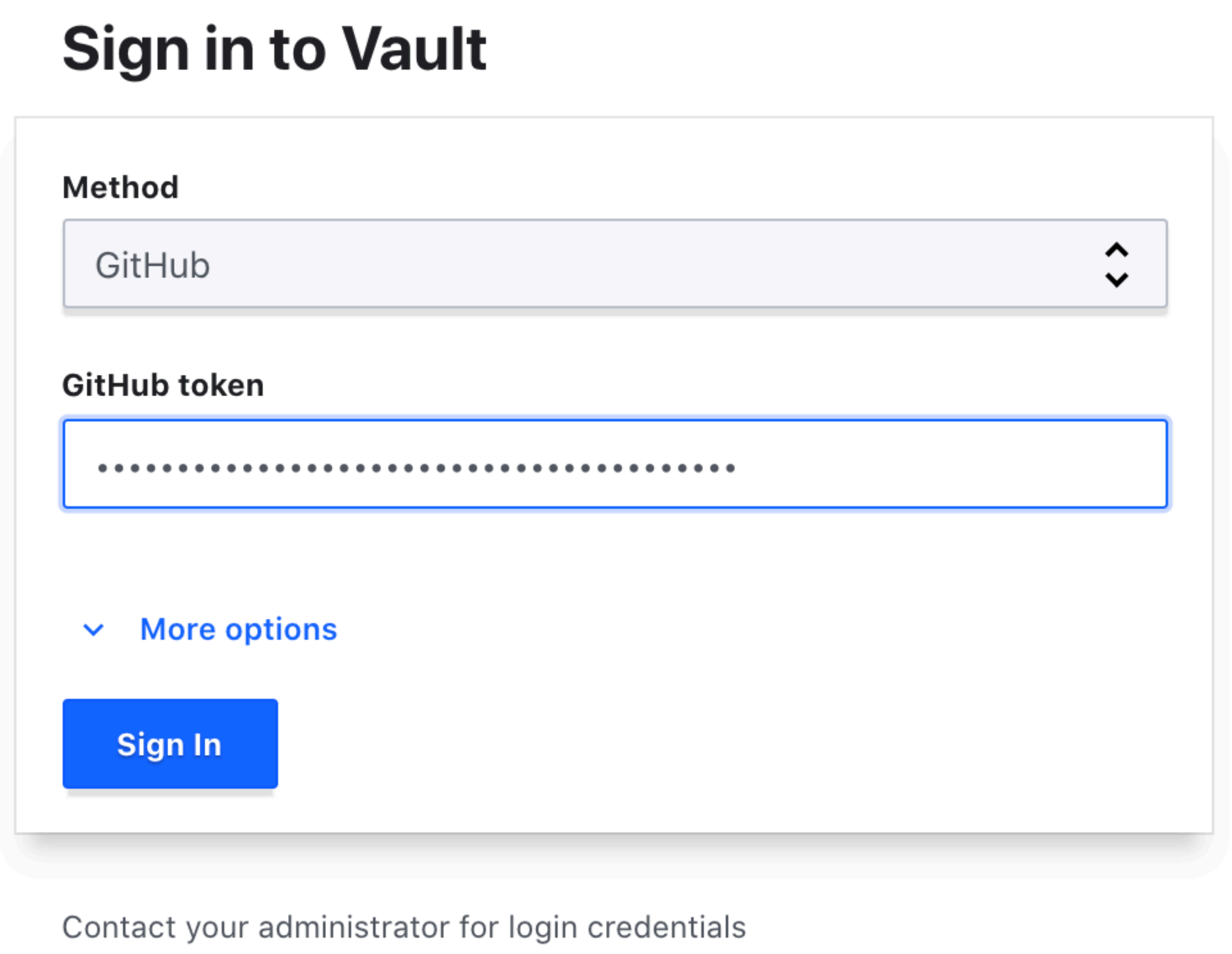
Receiving the Vault Token

- You can use either the token, or login, using github!

```
Success! You are now authenticated. The token information displayed
below is already stored in the token helper. You do NOT need to run
"vault login" again. Future Vault requests will automatically use this
token.

Key                     Value
---                     -----
token                   s.2YEWaAdnmCJd8Ymr6A9n9tnE
token_accessor          Nt4psOUbpzSWzuBw4pseSwws
token_duration          768h
token_renewable         true
token_policies          ["default"]
identity_policies       []
policies                ["default"]
```

Logging into the UI

Using your Github Token

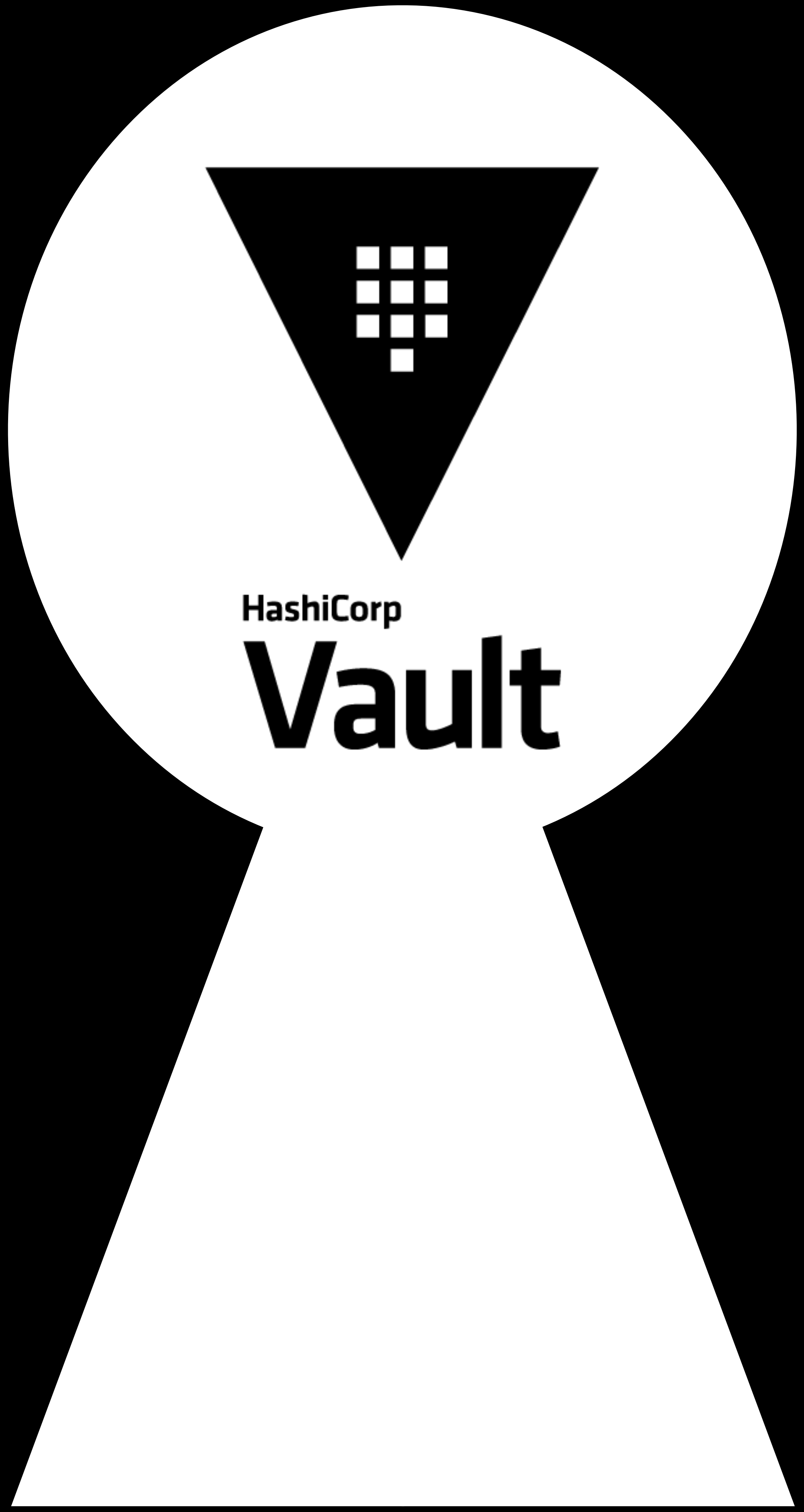


Sign in to Vault

Method: GitHub

GitHub token: ••

Contact your administrator for login credentials



Using the Backend API

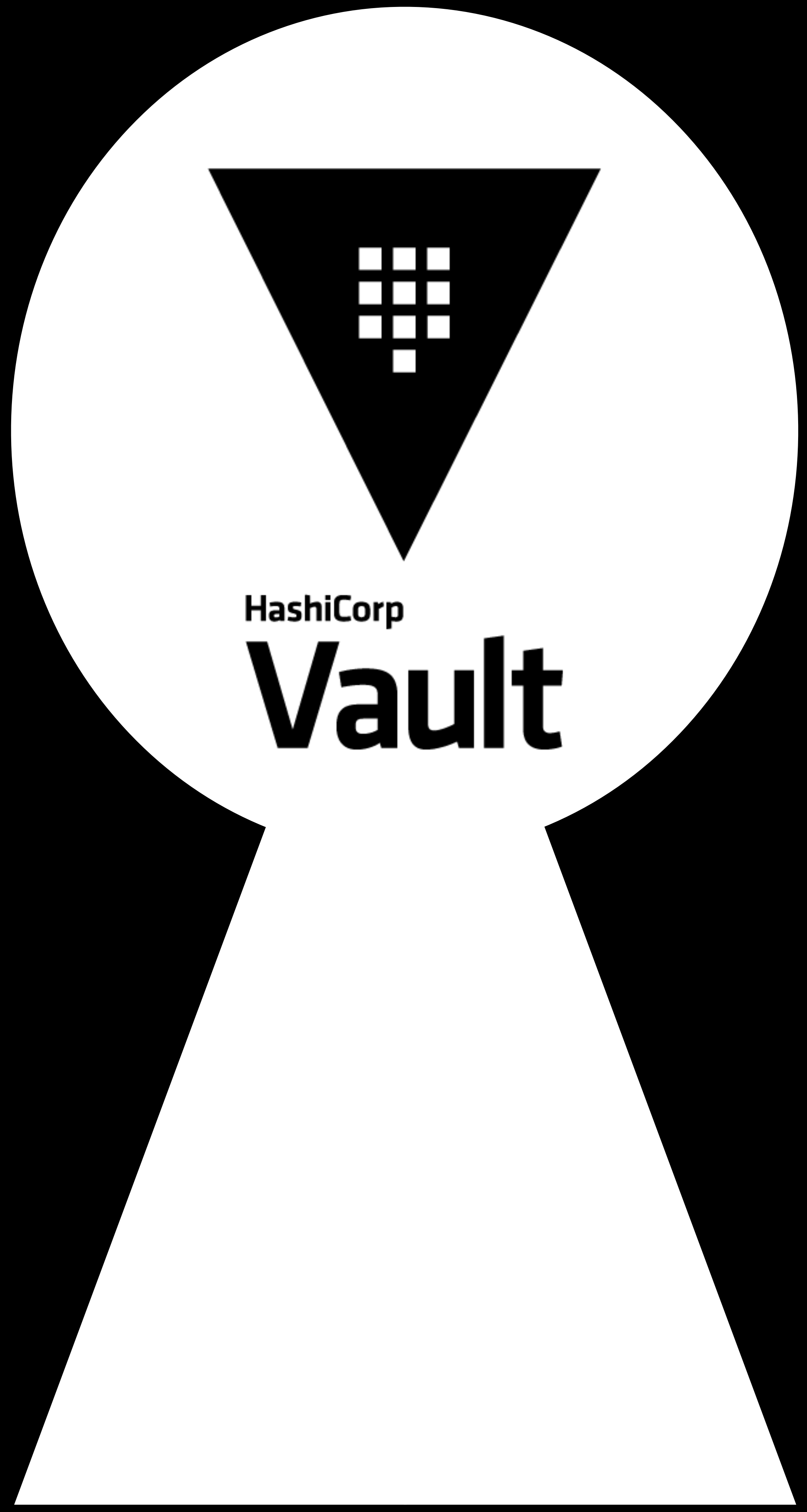
HTTP API

- The Vault HTTP API gives you full access to Vault via HTTP
- Every aspect of Vault can be controlled via this API
- The Vault CLI uses the HTTP API to access Vault.
- All API routes are prefixed with `/v1/`
- The API is expected to be accessed over a TLS connection at all times
- Consult specific actions at https://www.vaultproject.io/api-docs

Authentication with Tokens

- A user may have a client token sent to them.
- The client token must be sent as either the `X-Vault-Token` header
- HTTP Header or as `Authorization` HTTP Header using the `Bearer <token> scheme.`

```
$ curl \
    -H "X-Vault-Token: f3b09679-3001-009d-2b80-9c306ab81aa6" \
    -X LIST \
    http://127.0.0.1:8200/v1/secret/
```



Policies

Policies

- Policies to govern the behavior of clients
- Instrument Role-Based Access Control (RBAC)
- Specifying access privileges (*authorization*) by linking roles to tokens or an authentication

Initial Policies

- Two Policies get created initially:
  - `root` - Superuser to everything in Vault
  - `default` - Applied to everyone and provides common permissions

HCL

- Hashicorp Configuration Language
- Also known as ACL Policies
- JSON Compatible

```
path "<PATH>" {
  capabilities = [ "<LIST_OF_CAPABILITIES>" ]
}
```

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

HCL Wildcards

- HCL has the ability to use wildcards
- Asterisk
  - `*` at the end can represent any string in its place
  - `secret/training_*` grants permissions on any path starting with "`secret/training_`" (e.g. `secret/training_vault`)
- Plus Sign
  - To allow wildcard matching for a single directory, use `"+"`
  - `secret/app/+/stage` would match `secret/app/release_1.0/stage`

Capabilities and HTTP Verbs

Capability	HTTP Verbs
create	POST/PUT
read	GET
update	POST/PUT
delete	DELETE
list	LIST

Special Capabilities

- `sudo` allows access to paths that are root-protected
  - This includes some `auth/*`, `pki/*`, `sys/*`
  - For a complete reference: https://learn.hashicorp.com/tutorials/vault/policies?in=vault/operations#root-protected-api-endpoints
- `deny` disallows access

Creating a Policy

- Create a set of paths and save to a file 'secretsonly.hcl'

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

- Create a set of paths

```
$ vault policy write <POLICY_NAME> <POLICY_FILE>
```

```
$ vault policy write secretsonly secretsonly.hcl
```

Viewing Policies

- You can always view all the policies that are saved in Vault

```
$ vault policy list
```

- Reading a policy

```
$ vault policy read <POLICY_NAME>
```

```
$ vault policy read secretsonly
```

Analyzing Token Capabilities

- If you wish to see the capabilities of a given token

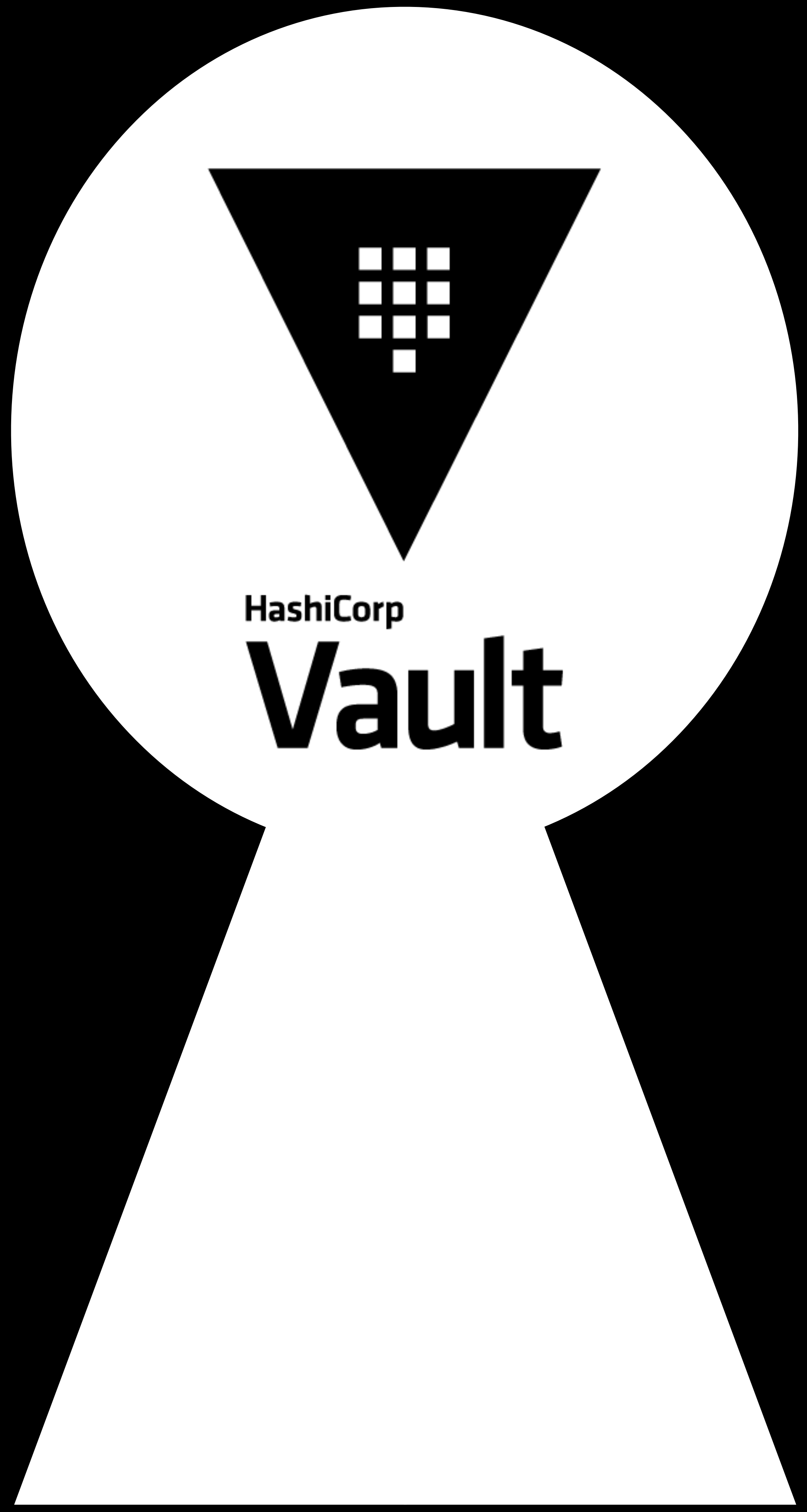
```
$ vault token capabilities <TOKEN> <PATH>
```

- For example, after creating a token with a given policy

```
$ vault token create -policy="secretsonly"
```

```
$ vault token capabilities 3LsGdNENj36ZpqQ2ZOve20ws secrets/foo
create, delete, read, sudo, update
```

Lab 3: Authentication and Policies



Sealing the Vault

Sealing

- When a Vault Server is started it is in a *sealed state*
- While Vault has access to the physical storage it doesn't know how to decrypt it
- Only possible operations are to unseal the Vault and check the status of the unseal

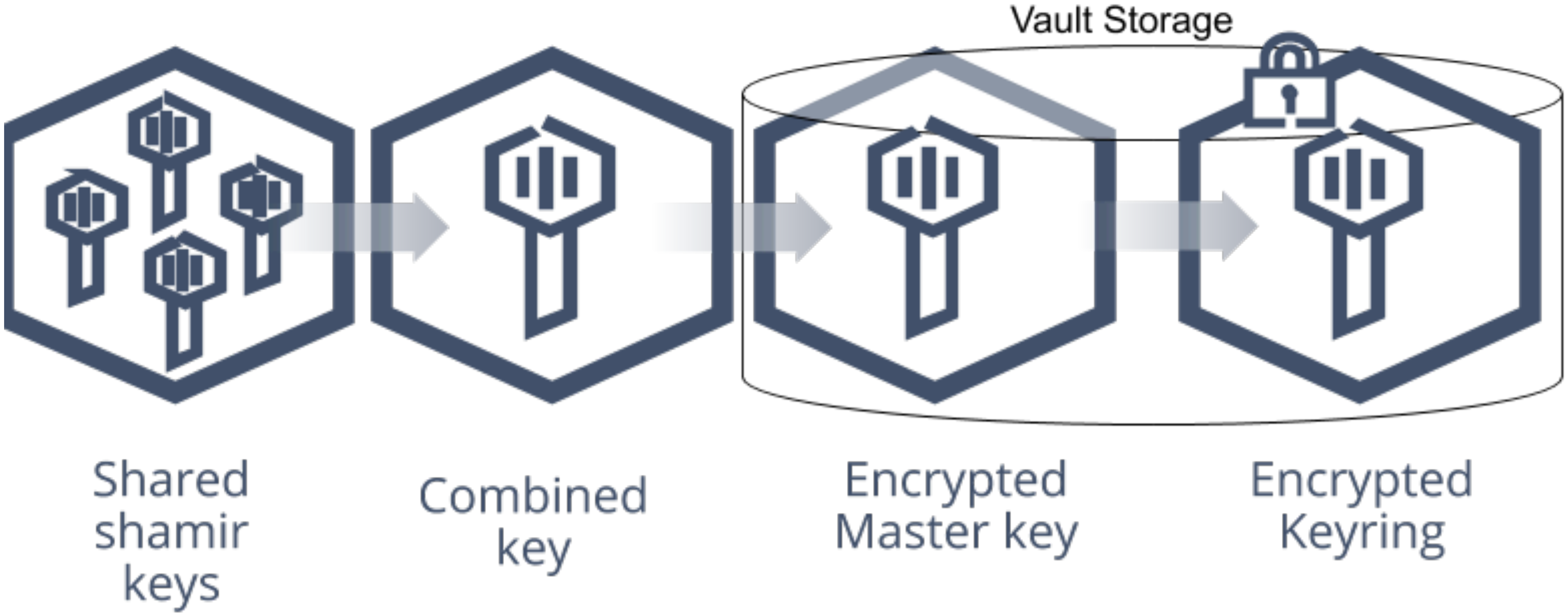
Encrypted Storage

Encryption

- Vault Requires the Encryption Key to decrypt the data
- Encryption Key is Also Stored with the Data in a *keyring*
- Data is encrypted with another encryption key known as the *master key*

Decryption

- Vault must decrypt the encryption key which requires the master key
- Unsealing is the process of getting access to this master key
- The master key is stored alongside all other Vault data, but is encrypted by yet another mechanism: the unseal key.



https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

Shamir

- Default Vault config uses a Shamir seal.

- Vault uses an algorithm known as Shamir's Secret Sharing to split the key into shards

- A certain threshold of shards is required to reconstruct the unseal key

- Shards are added one at a time (in any order) until enough shards are present to reconstruct the key and decrypt the master key.

Unsealing

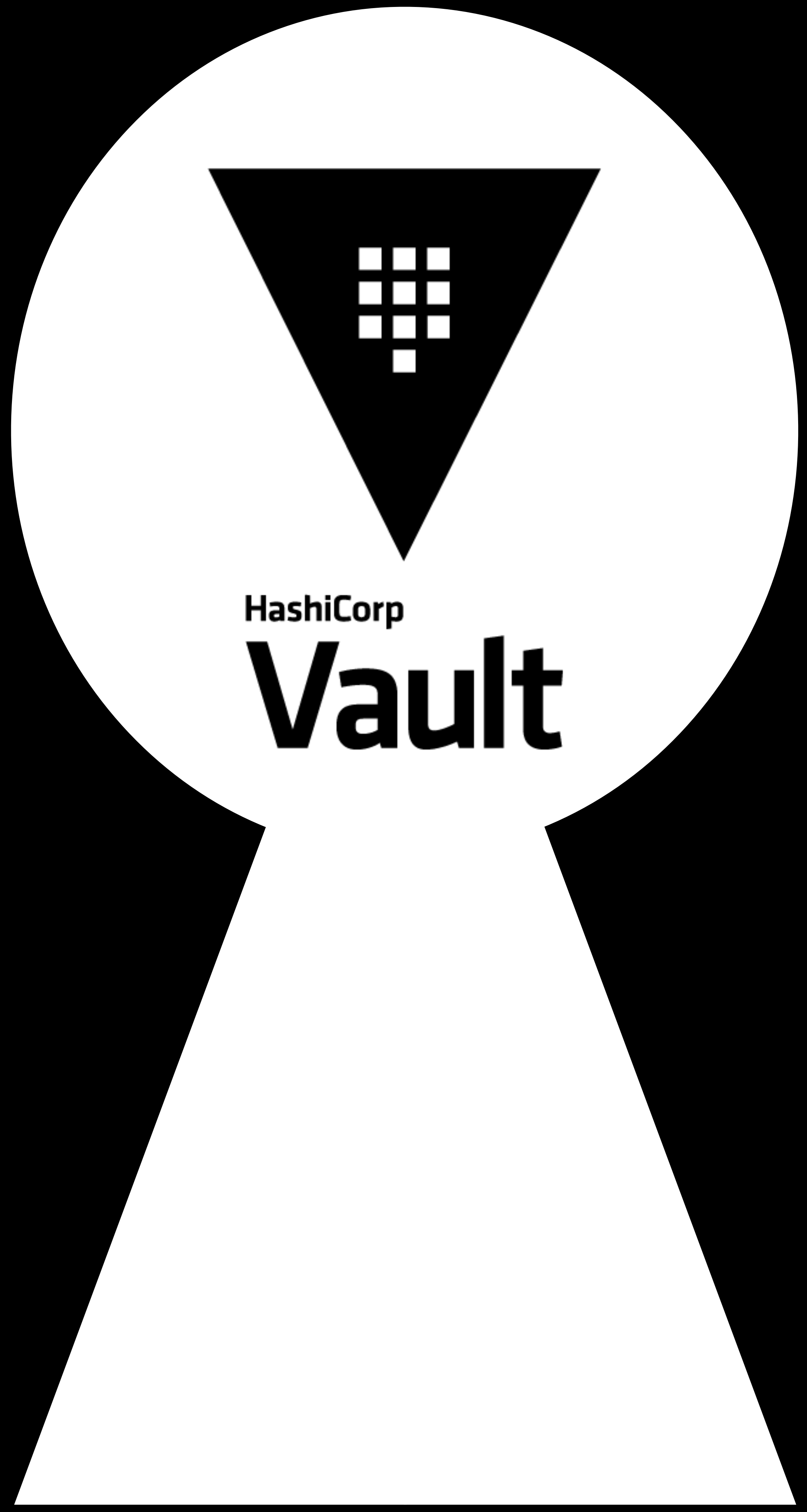
- To Unseal

```
$ vault operator unseal
```

- Process is Stateful
- Each key can be entered on multiple machines
- Can be done by CLI or API

When it gets Sealed Again

- The Vault Remains Unsealed Until:
  - It is resealed via the API
  - The server is restarted.
  - Vault's storage layer encounters an unrecoverable error
- When a Reseal Happens:
  - Master Key Disposed
  - Requires another Reseal Process



Starting a Production Server

Configuring a Server

```
storage "raft" {
  path    = "./vault/data"
  node_id = "node1"
}

listener "tcp" {
  address     = "127.0.0.1:8200"
  tls_disable = 1
}

api_addr = "http://127.0.0.1:8200"
cluster_addr = "https://127.0.0.1:8201"
ui = true
```

- Raft is a methodology of how to store data.
  - Multinode
  - https://raft.github.io/
- We typically want to deploy via TLS, in this case it is turned off for lab exercise

Starting a Server

- Start the server with your config.hcl

```
vault server -config=config.hcl

==> Vault server configuration:

             Api Address: http://127.0.0.1:8200
                     Cgo: disabled
         Cluster Address: https://127.0.0.1:8201
              Go Version: go1.14.4
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster address:
"127.0.0.1:8201", max_request_duration: "1m30s", max_request_size:
"33554432", tls: "disabled")
               Log Level: info
                   Mlock: supported: true, enabled: true
           Recovery Mode: false
                 Storage: raft (HA available)
```

Initializing the Server

- Just because it started does not mean it is ready
  - It is in a sealed state
  - Unseal Keys have not been disbursed
- We will need to initialize the server

```
$ vault operator init
```

Initialization Response

```
$ vault operator init

Unseal Key 1: 4jYbl2CBIv6SpkKj6Hos9iD32k5RfGkLzlosrrq/JgOm
Unseal Key 2: B05G1DRtfYckFV5BbdBvXq0wkK5HFqB9g2jcDmNfTQiS
Unseal Key 3: Arig0N9rN9ezkTRo7qTB7gsIZDaonOcc53EHo83F5chA
Unseal Key 4: 0cZE0C/gEk3YHaKjIWxhyyfs8REhqkRW/CSXTnmTilv+
Unseal Key 5: fYhZOseRgzxmJCmIqUdxEm9C3jB5Q27AowER9w4FC2Ck

Initial Root Token: s.KkNJYWF5g0pomcCLEmDdOVCW

Vault initialized with 5 key shares and a key threshold of 3. Please
securely distribute the key shares printed above. When the Vault is re-
sealed, restarted, or stopped, you must supply at least 3 of these keys
to unseal it before it can start servicing requests.

Vault does not store the generated master key. Without at least 3 key to
reconstruct the master key, Vault will remain permanently sealed!
```

Unseal With The First, Second, Third Key

Unseal Key #1

```
$ vault operator unseal FCKAxgmM9pmoitFAqGJBzp4LRe1wfgDyFPqR7Yj/d8Kv
```

Unseal Key #2

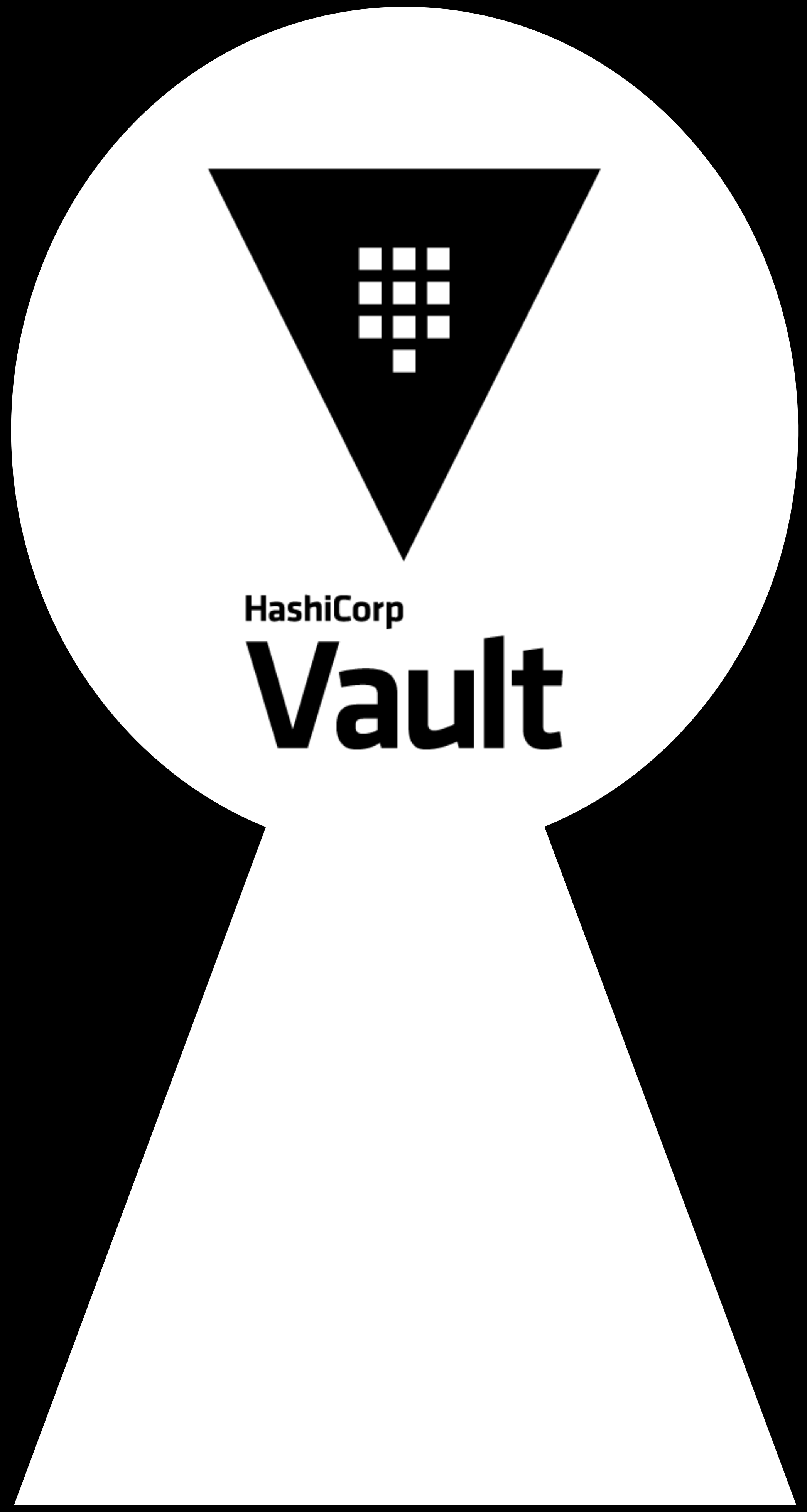
```
$ vault operator unseal VCgIXC/4zqjVauFmML04mvsMId7T2XYi3GLoO051q9w9
```

Unseal Key #3

```
$ vault operator unseal 6JETbawxIRgkVaHwWFw+j8z6diet6bUP2/+cy8VMjipM
```

Thresholds can be customized during `init`

Lab 4: Production and Sealing

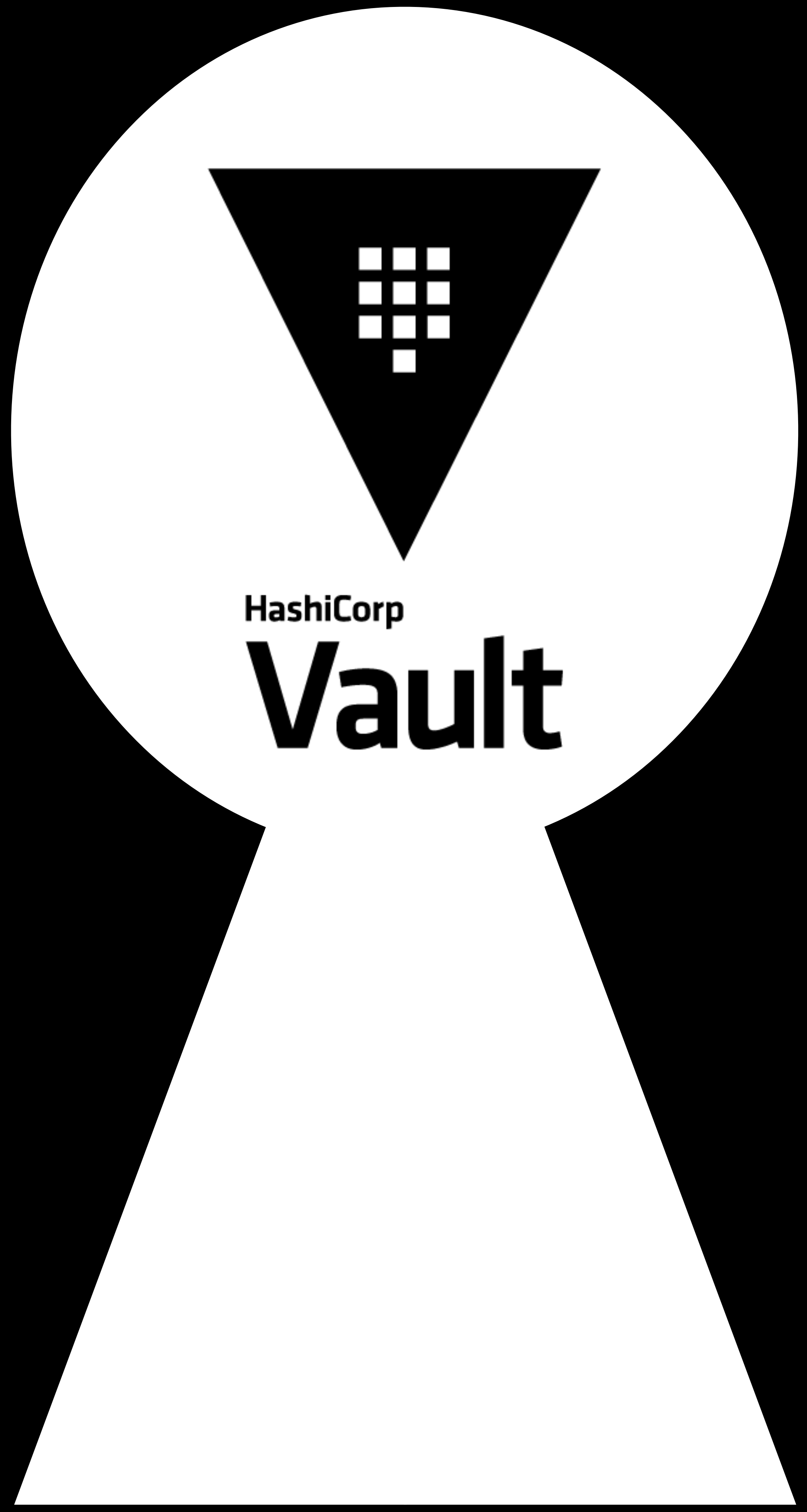


Agent

Vault Agent

- Client Daemon - Client Side Helper to interact with the Vault Server and runs as a daemon
    - Auto-Auth - **Automatically** authenticate to Vault and manage the token renewal process for locally-retrieved dynamic secrets.
    - Caching - Allows client-side caching of responses containing newly created tokens and responses containing leased secrets generated off of these newly created tokens.
    - Templating - Allows rendering of user supplied templates by Vault Agent, using the token generated by the Auto-Auth step. https://www.vaultproject.io/docs/agent/template

```
$ vault agent -h
```



Docker Container

Running Development Vault on Docker

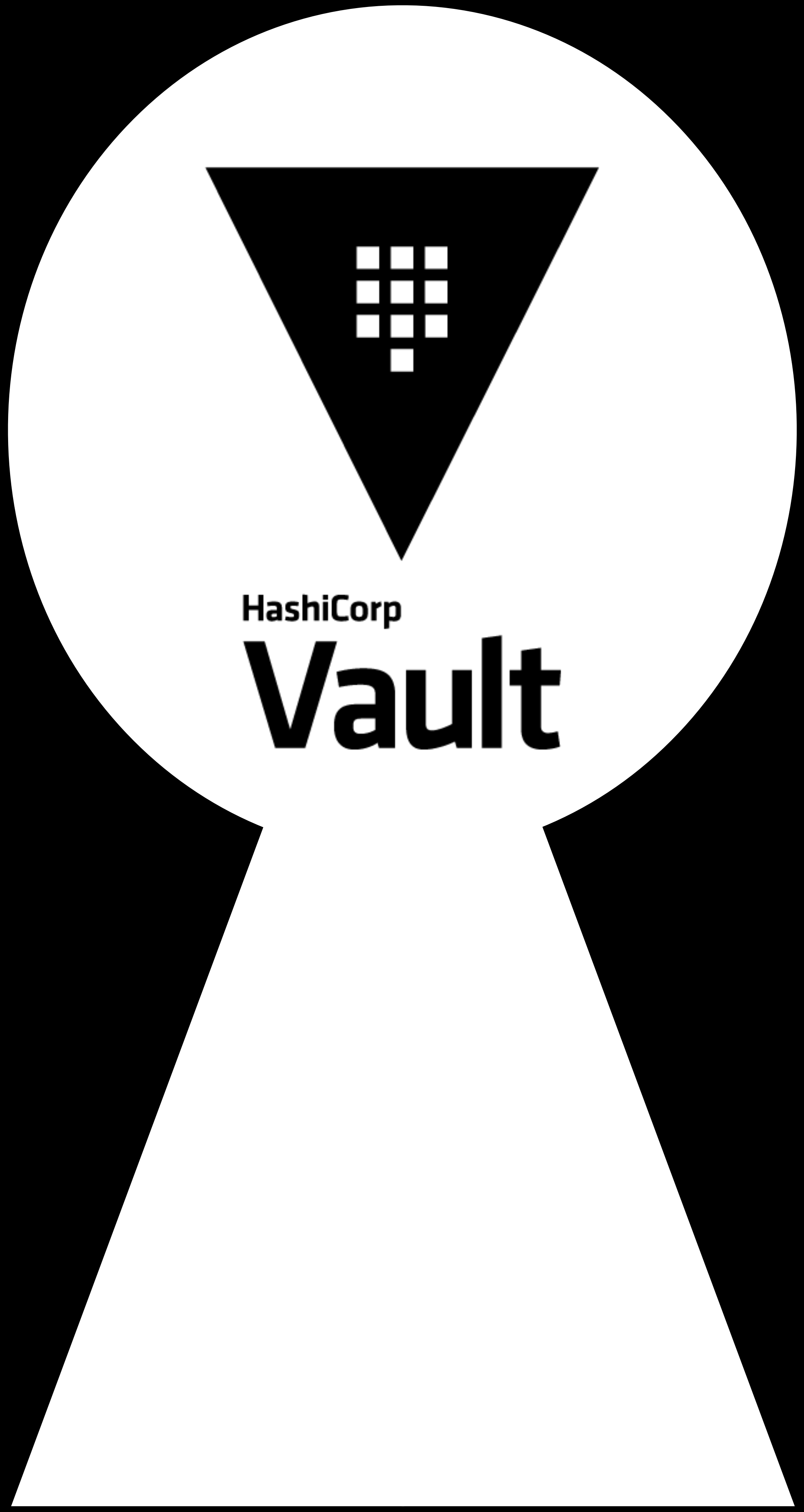
- https://registry.hub.docker.com/_/vault/
- `--cap-add=IPC_LOCK` is used lock memory to prevent sensitive values from being swapped to disk

```
$ docker run --cap-add=IPC_LOCK -e 'VAULT_DEV_ROOT_TOKEN_ID=myroot' -e
'VAULT_DEV_LISTEN_ADDRESS=0.0.0.0:1234' vault
```

Running Production Vault on Docker

- https://registry.hub.docker.com/_/vault/
- `--cap-add=IPC_LOCK` is used lock memory to prevent sensitive values from being swapped to disk
- `VAULT_LOCAL_CONFIG` specifies any setting that you wish to bootstrap before running, including path, lease time.

```
docker run --cap-add=IPC_LOCK -e 'VAULT_LOCAL_CONFIG={"backend":
{"file": {"path": "/vault/file"}}, "default_lease_ttl": "168h",
"max_lease_ttl": "720h"}' vault server
```



Kubernetes

Running Production Vault on K8S

- The Vault Helm chart is the recommended way to install and configure Vault on Kubernetes.
- Helm chart is the primary method for installing and configuring Vault to integrate with other services such as Consul for High Availability (HA) deployments
- While it automates a lot, you will be responsible to initialize, monitor, backup, upgrade the Vault cluster.

Setup a Helm Repo

- Add the repository for helm, which will include the vault chart

```
$ helm repo add hashicorp https://helm.releases.hashicorp.com
```

- When you now do a search, you should find the helm chart for Vault

```
$ helm search repo hashicorp/vault
```

- When you now do a search, you should find the helm chart for Vault

```
$ helm install vault hashicorp/vault --namespace vault --dry-run
```

Deploy as HA

- You can deploy in High Availability Mode using the following command

```
$ helm install vault hashicorp/vault \
    --namespace vault \
    --set "server.ha.enabled=true" \
    --set "server.ha.replicas=5" \
```

Lab 5: Kubernetes

HashiCorp
Vault

Thank You

Daniel Hinojosa
@dhinojosa