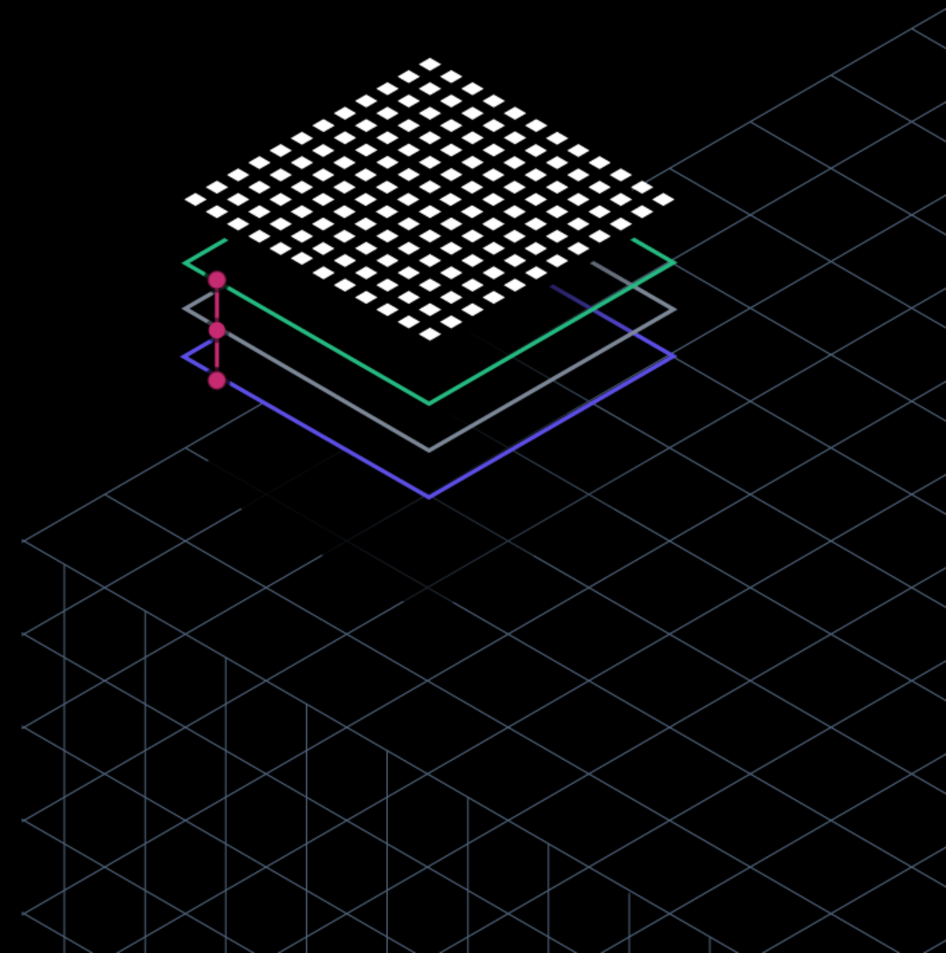
HashiCorp

Vault 102: Introduction to Vault Operations



Courseware Download

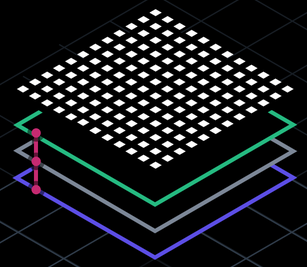
Download the course materials from the following links:

Lab book (HTML): **http://bit.ly/VLT102-Lab-html**

Lecture slides: **http://bit.ly/VLT102-Slides**

Your Name

Your Role, Your Company
Interesting Fact



HashiCorp

Agenda

Time	Topic	Duration
9:00 - 9:15 AM	Welcome	15 min.
9:15 - 9:50 AM	Module 1: Vault Overview	35 min.
9:50 - 10:05 AM	Lab 1: Lab Setup	15 min.
10:05 - 10:35 AM	Module 2: Vault Installation	30 min.
10:35 - 10:50 AM	Break	15 min.
10:50 - 11:15 AM	Lab 2: Vault Server Configuration	25 min.
11:15 - 12:00 PM	Module 3: Auto-Unseal	45 min.
12:00 - 1:00 PM	Lunch break	60 min.
1:00 - 1:30 PM	Lab 3: Auto-Unseal	30 min.
1:30 - 2:00 PM	Module 4: Vault Cluster Deployment	30 min.
2:00 - 2:20 PM	Module 5: Vault Operations	20 min.
2:20 - 2:50 PM	Lab 5: Vault Operations	30 min.
2:50 - 3:25 PM	Module 6: Vault Policies	35 min.
3:25 - 3:40 PM	Break	15 min.
3:40 - 4:05 PM	Lab 6: Working with Policies	25 min.
4:05 - 4:25 PM	Module 7: Secure Introduction	20 min.
4:25 - 4:45 PM	Lab 7: Vault Agent	20 min.
4:45 - 5:00 PM	Wrap-up	15 min.

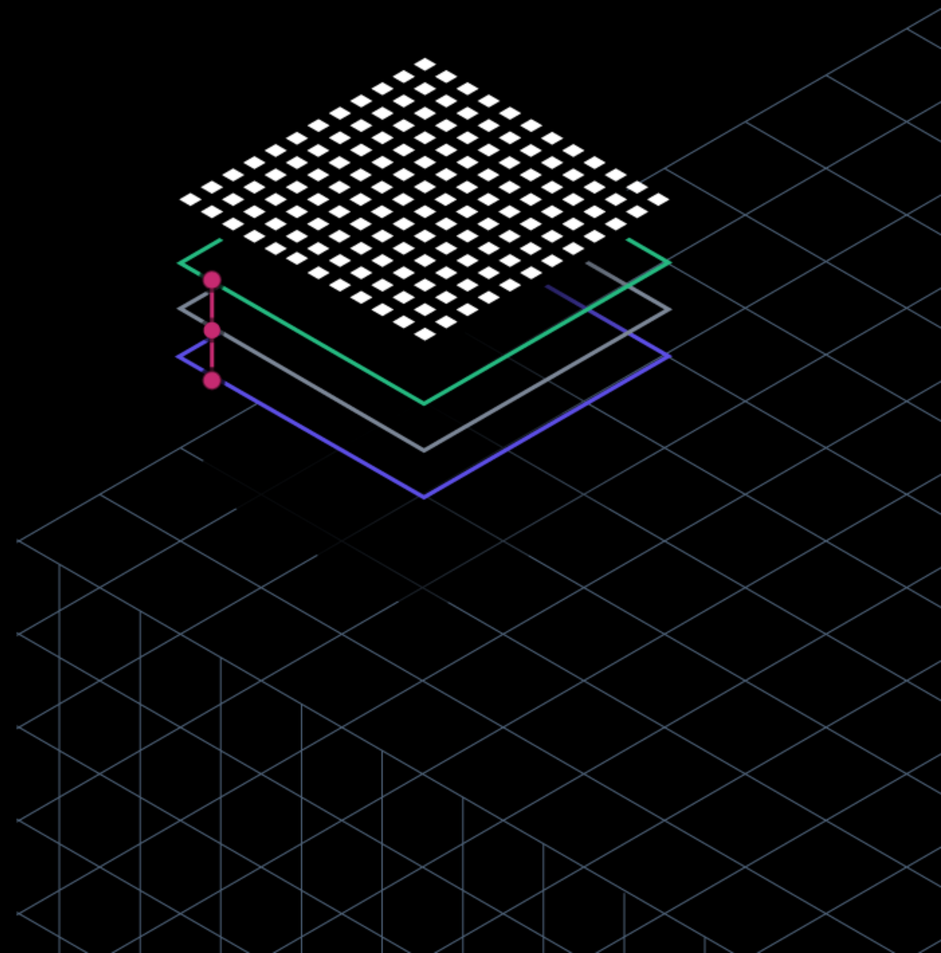
Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp

Vault Overview

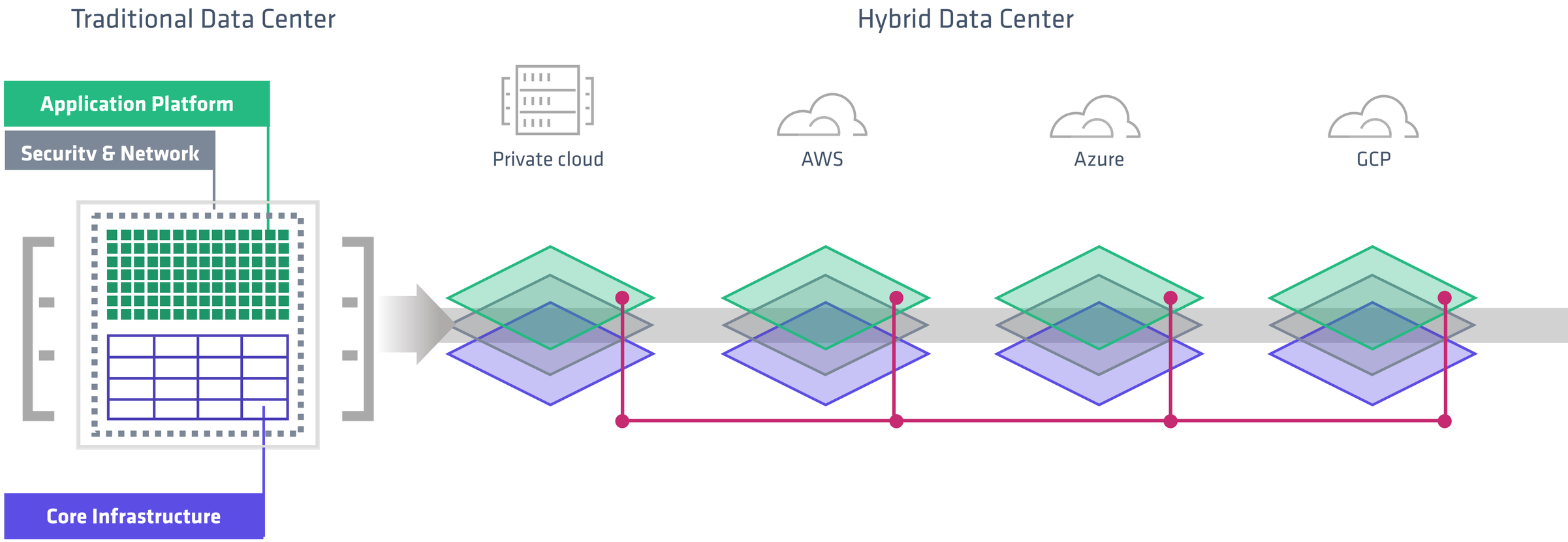


Agenda

- Vault Introduction
  - Security in a dynamic world
  - Managing secrets in distributed infrastructure
  - Vault use cases
- Vault Architecture
  - Introducing Vault components
- Vault Audit Devices

Vault Introduction

Shifting from Static to Dynamic Infrastructure



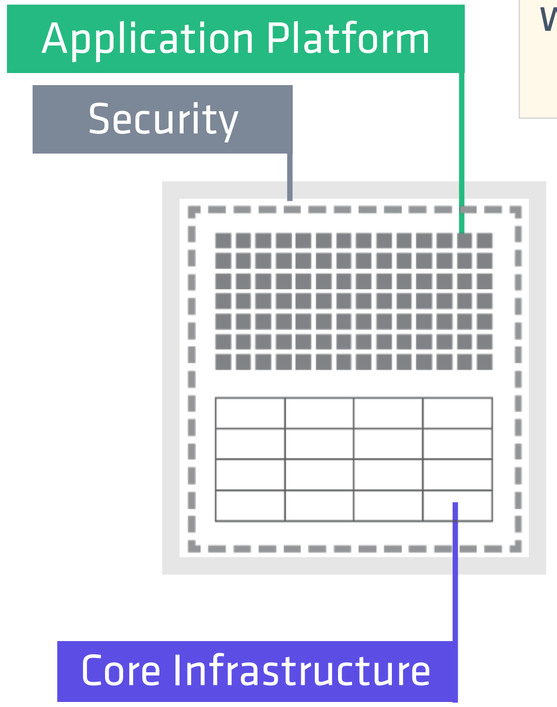
Homogenous, Static, Consolidated

Heterogeneous, Dynamic, Distributed

Security in a Dynamic World

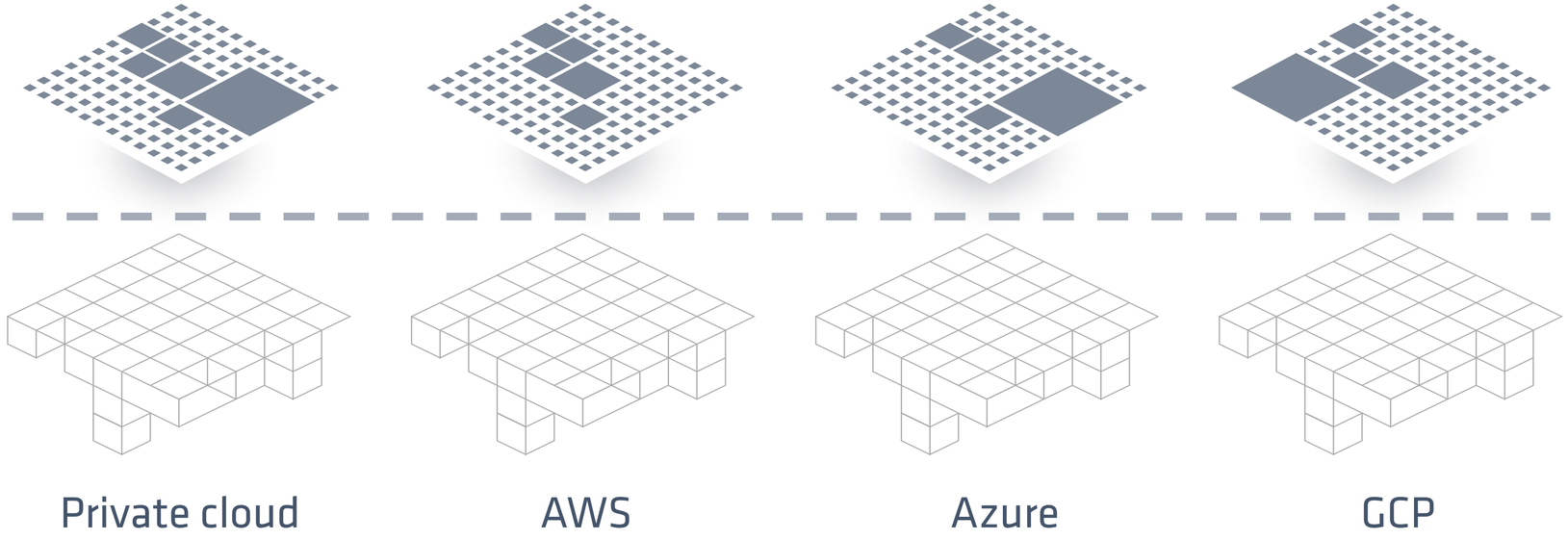
Static and Consolidated

Perimeter-based security was sufficient for private datacenters



Hybrid, Dynamic and Distributed

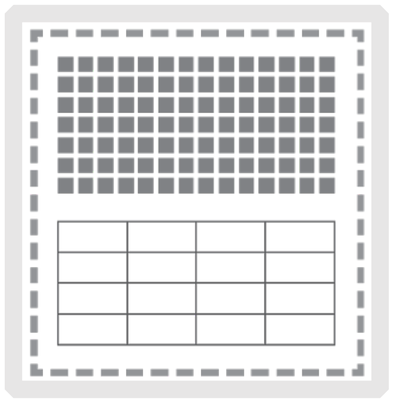
How does security extend across multiple data centers/clouds?



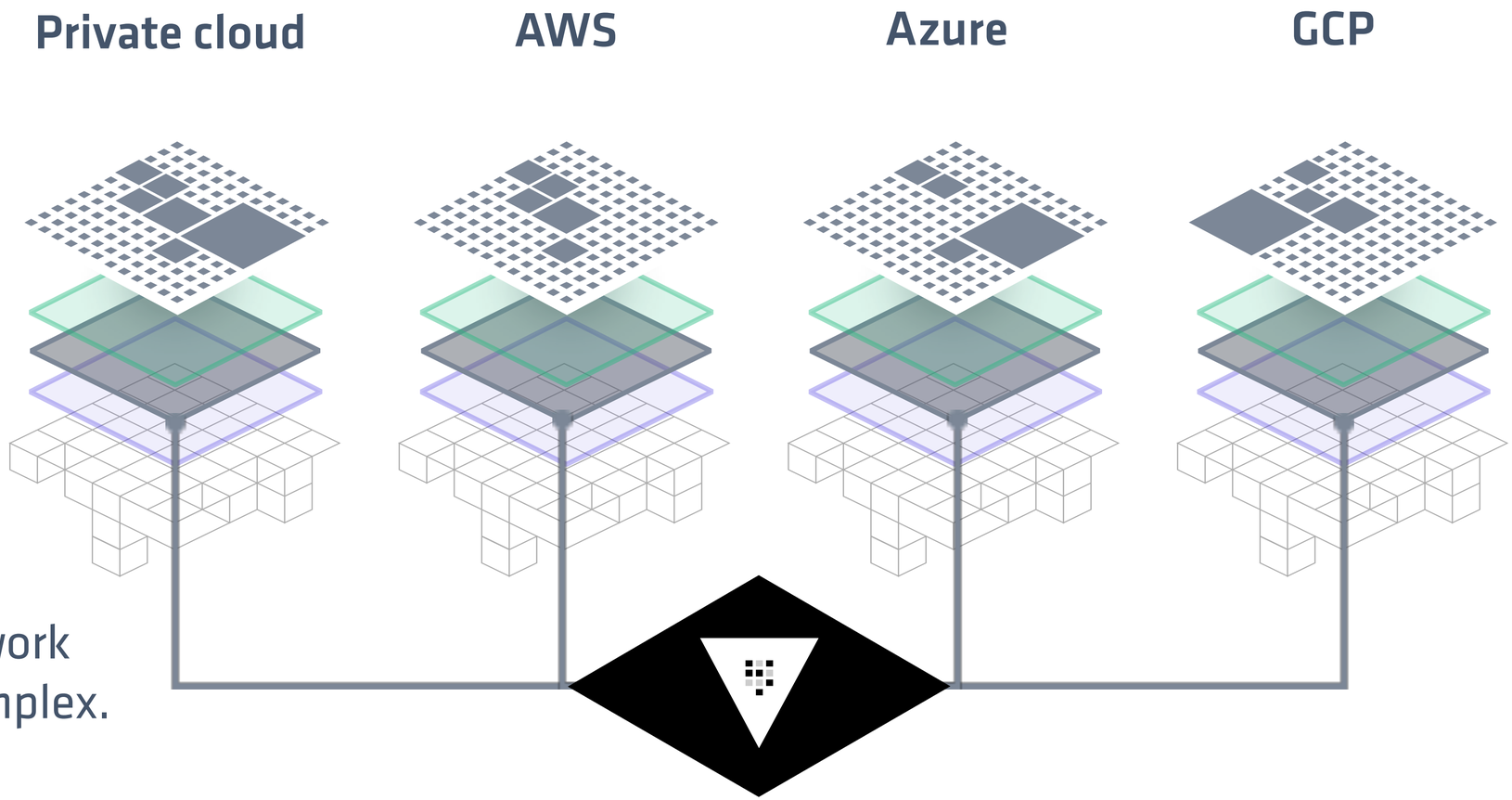
Managing Secrets in Distributed Infrastructure

- How can we **centralize** secrets?
- How can we manage the **lifecycle** of a secret?
- How can we **audit** secret access?
- How can we manage **access** to secrets?
- How can we securely distribute secrets **across hybrid environments**?
- How can we **mitigate** a compromised secret?

Secure Infrastructure using Vault

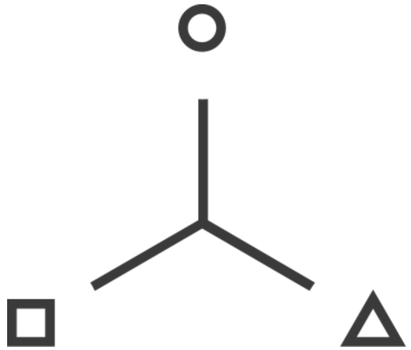


With each new cloud, network topologies become more complex.



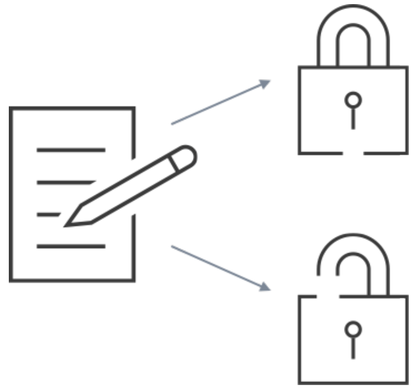
Vault Objectives

- Provide **single source of secrets** for humans and machines
- **Scale** to meet security needs of largest organizations
- Allow for complete **secret lifecycle management**



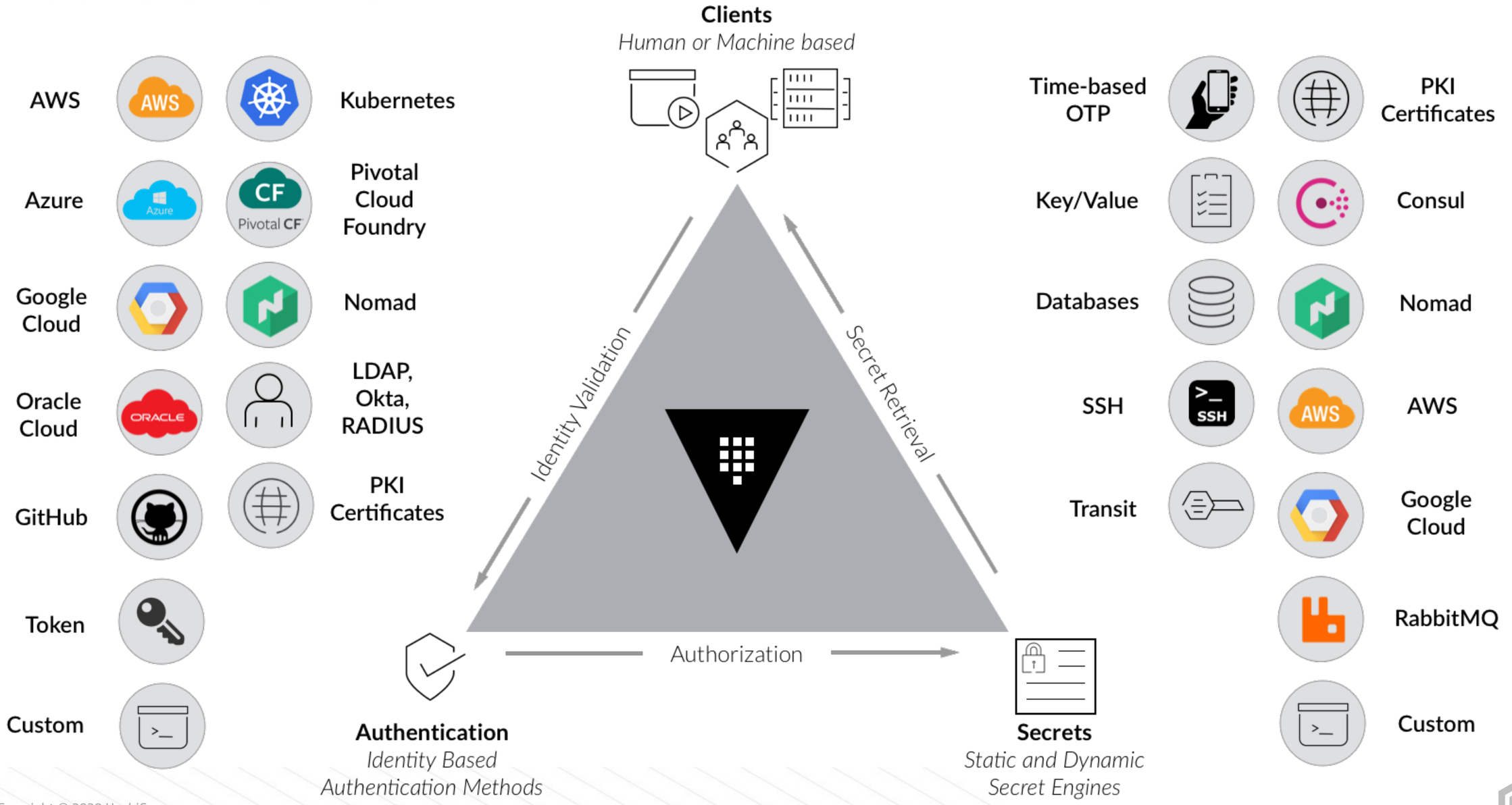
Eliminate Secret Sprawl

Securely Store Any Secret



Provide Secret Governance

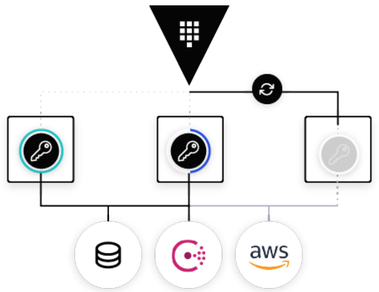
Vault Overview



Use Cases

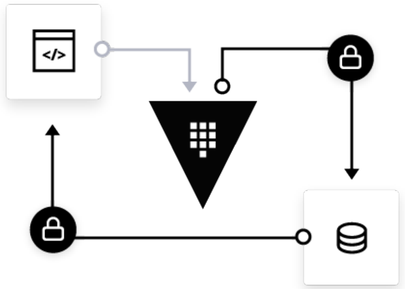
Secrets Management

Centrally store, access, and deploy secrets across applications, systems, and infrastructure



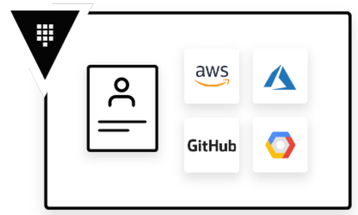
Encrypting Application Data

Keep secrets and application data secure with one centralized workflow to encrypt data in flight and at rest



Identity-based Access

Authenticate and access different clouds, systems, and endpoints using trusted identities

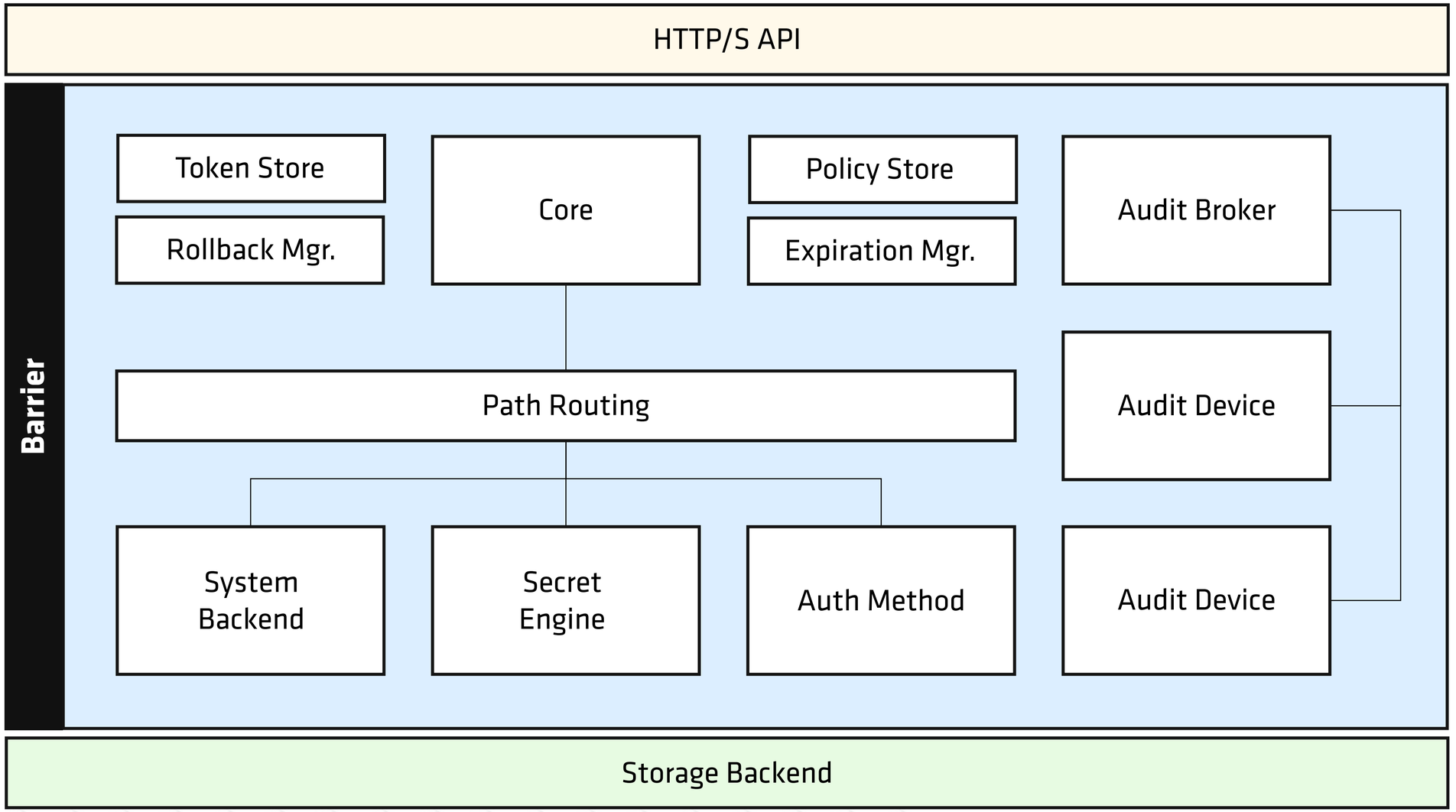


Vault Architecture & Terminology

Server

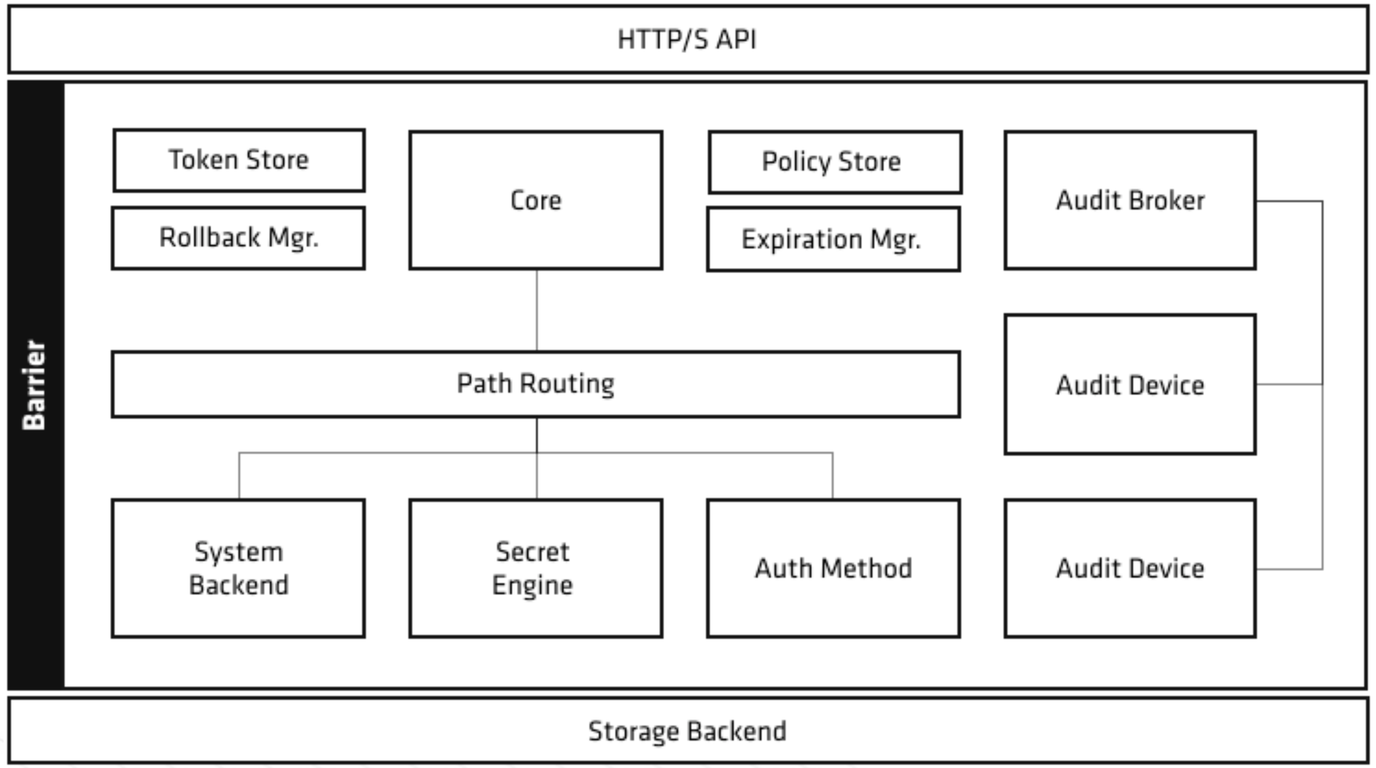
- Vault server provides an HTTP API which clients interact with and manages the interaction between all the **backends**, **ACL enforcement**, and **secret lease revocation**
  - Vault server requires a **storage backend** so that data is available across restarts
  - Vault server starts the **HTTP API** on start so that clients can interact

Architecture



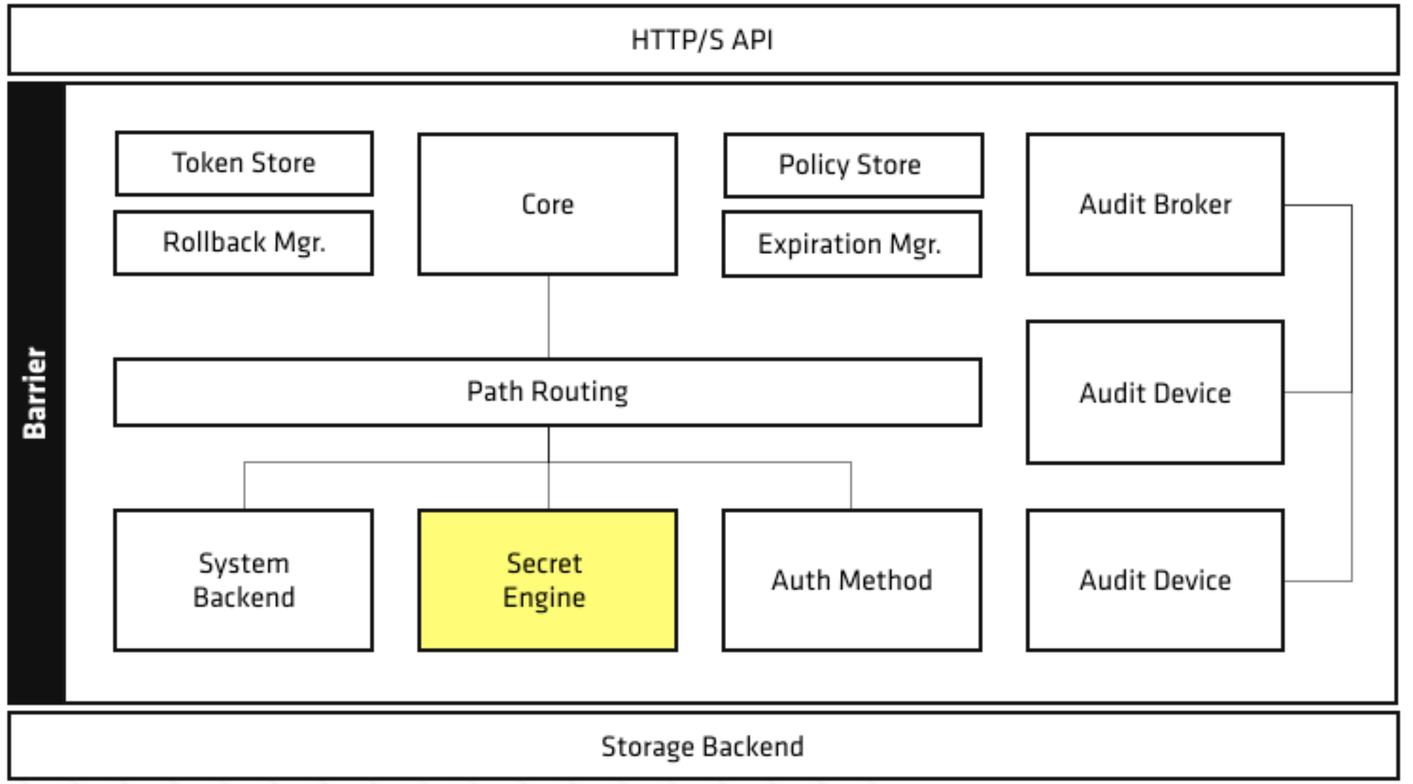
Barrier

- The **barrier** is a cryptographic seal around the Vault
- All data that flows between **Vault** and the **storage backend** passes through the barrier



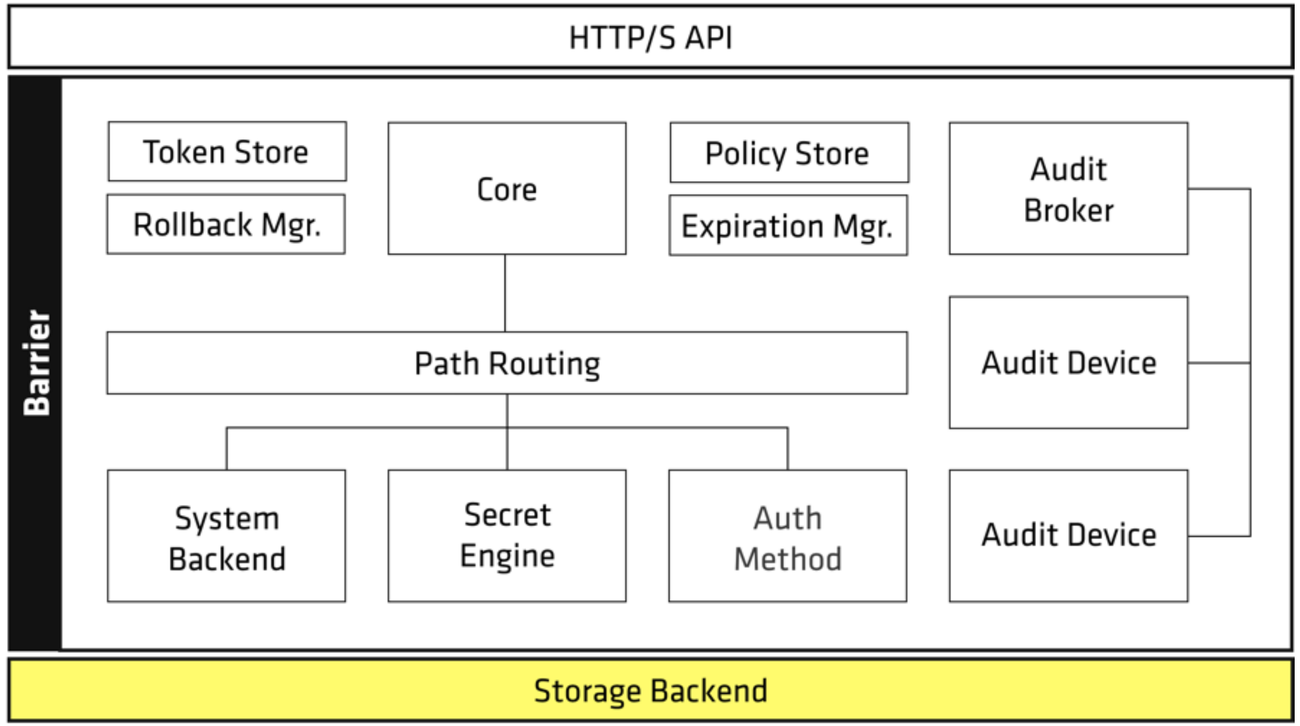
Secret Engine

- A **secret engine** stores, generates, or encrypt data - responsible for managing secrets
  - Some secret engines behave like encrypted key-value stores while others dynamically generate secrets when queried
  - A Vault cluster can have **multiple secret engines**



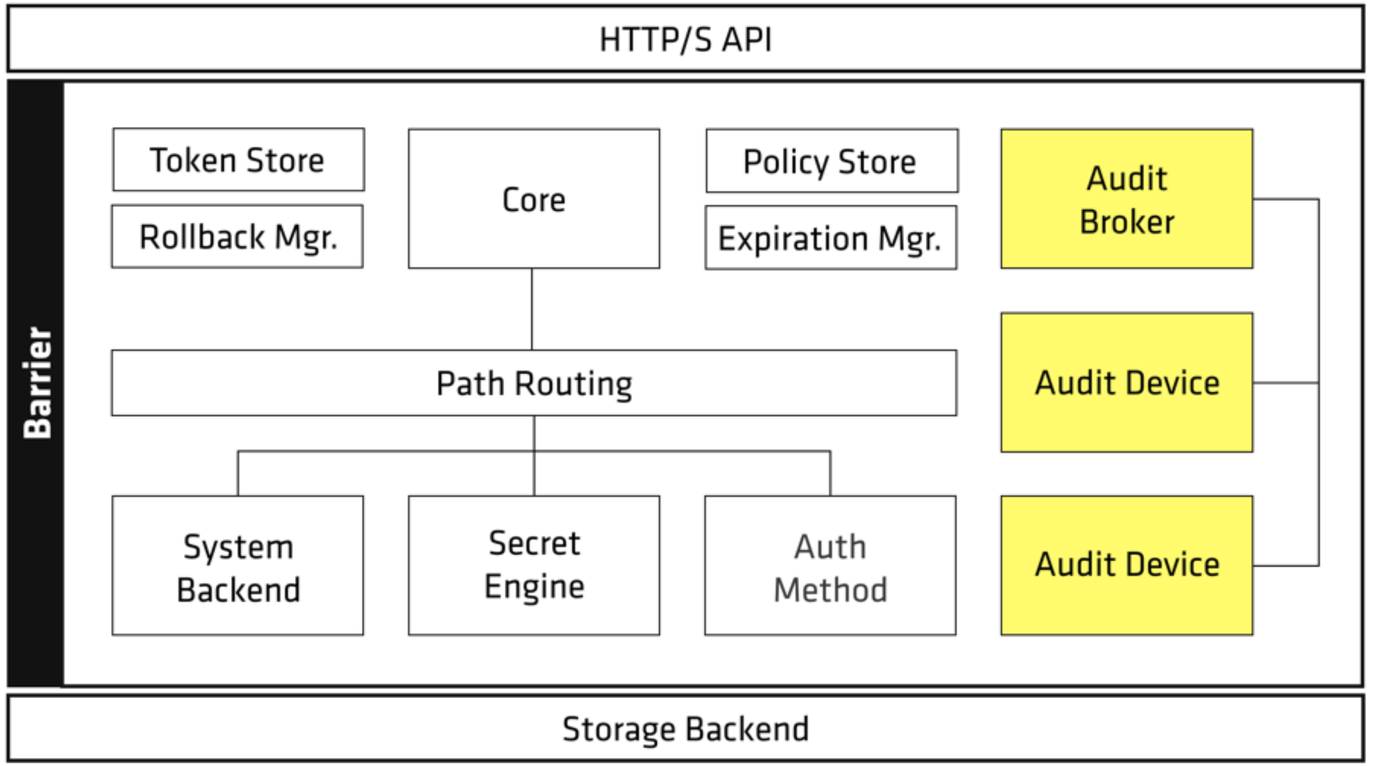
Storage Backend

- The storage backend is responsible for durable storage of encrypted data
  - There is only one storage backend per Vault cluster
- Data is encrypted **in-transit** and **at-rest** with **256 bit AES**
- Storage backend examples: *Consul*, *Amazon S3*, Cassandra, *MySQL*, *in-memory*



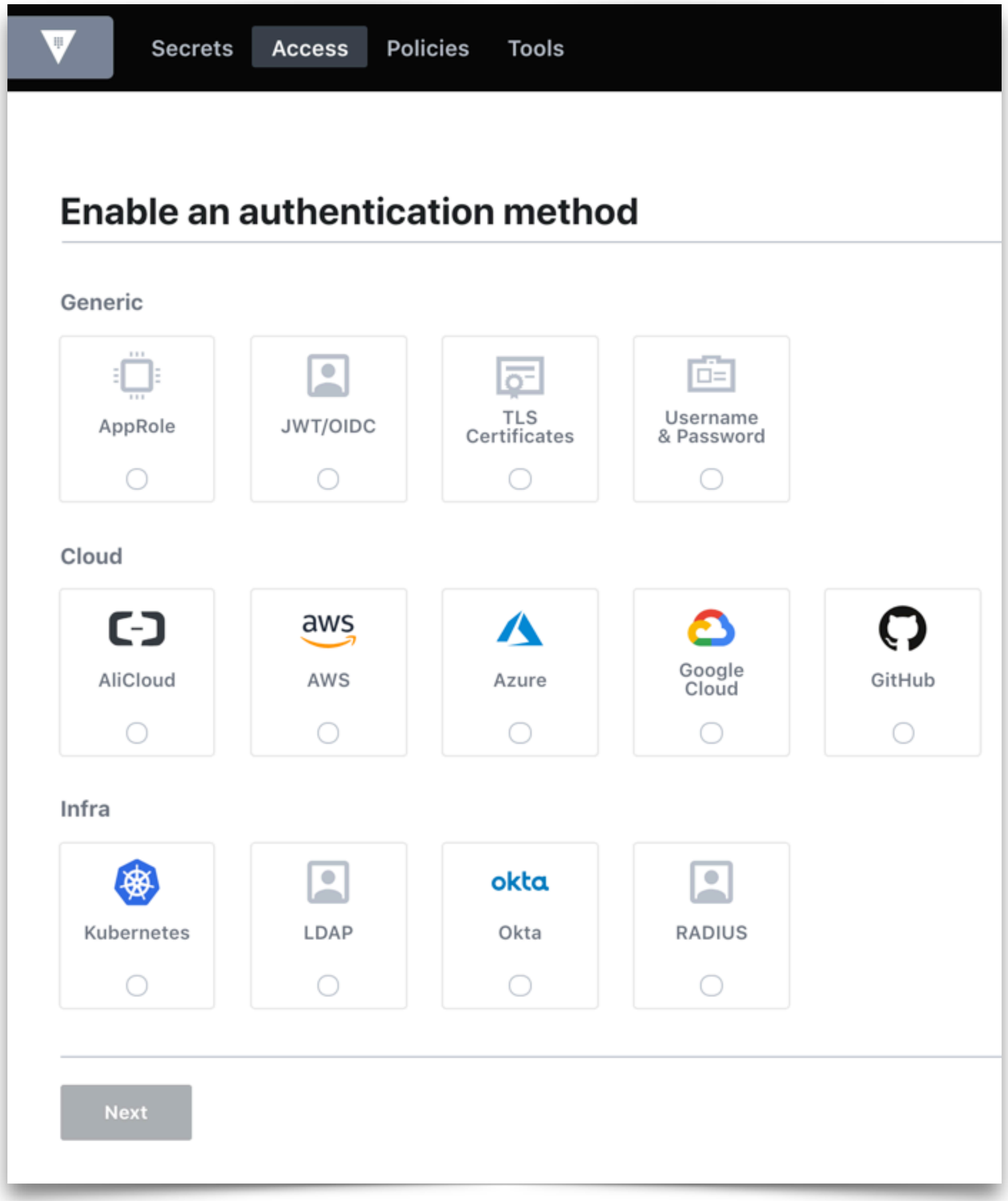
Audit Device

- An **audit device** is responsible for managing audit logs
- There can be multiple audit devices in a Vault cluster
- Audit devices:
  - File
  - Syslog
  - Socket



Auth Method

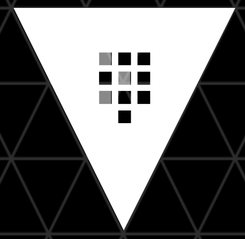
- An **auth method** is a credential-based backend that can be used as a way to authenticate humans or machines against Vault
  - **Machine-oriented:** AppRole, TLS, tokens, etc.
  - **Platform specific:** AliCloud, AWS, Azure, GCP, Kubernetes, etc.
  - **Operator-oriented:** Github, LDAP, username & password, etc.



Auditing

About Audit Devices

- Audit devices keep a detailed log of all **Vault requests** and **responses**
  - Every interaction with Vault including errors
- Multiple audit devices can be enabled
  - Let's you have redundant copy
  - A second copy in case the file was tampered with
- Sensitive information is obfuscated by default (HMAC-SHA256)
- Prioritizes safety over availability



Enabling Audit Logging

Terminal

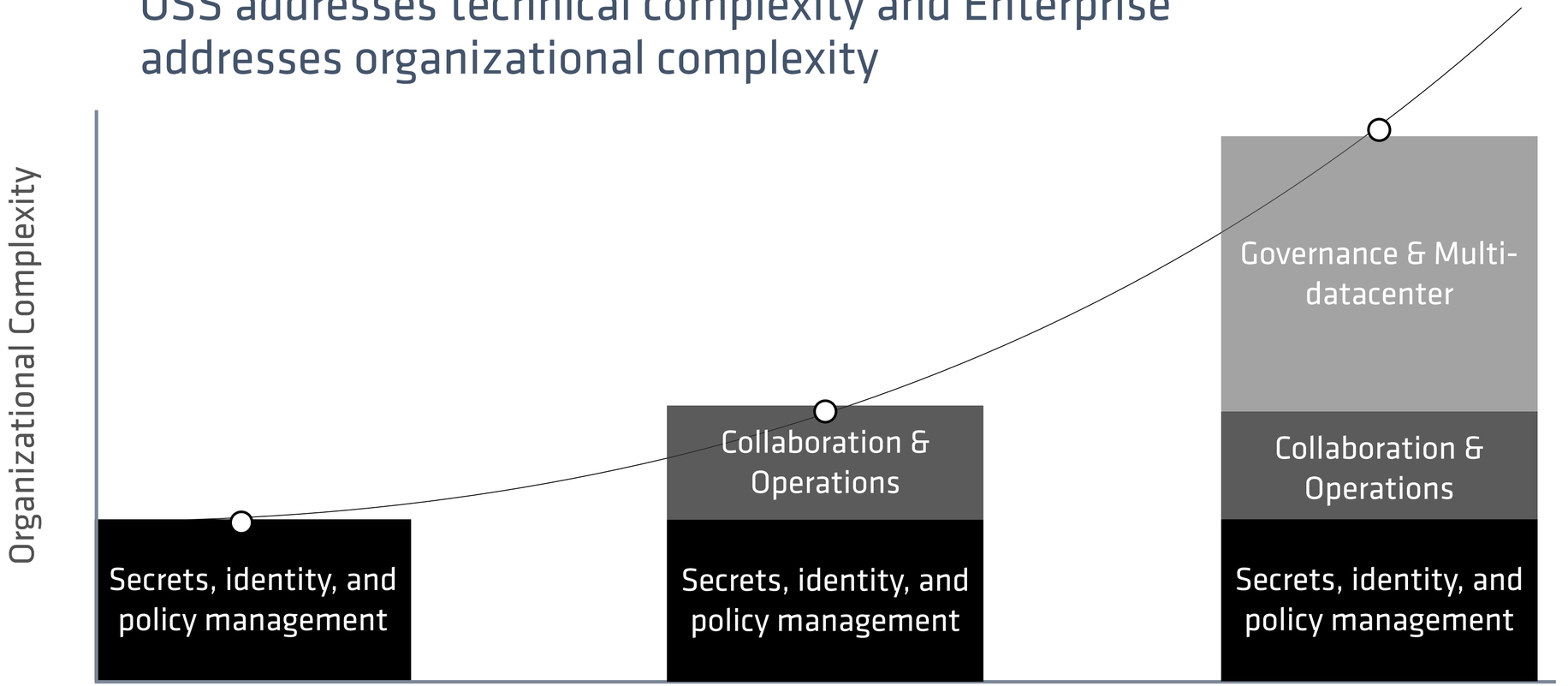
```
$ vault audit enable file file_path=/logs/vault_audit.log
Success! Enabled the file audit device at: file/

$ tail -f /logs/vault_audit.log | jq
{
   "request": {
    "id": "36cc380f-52a5-1825-e507-1a7acaab62a0",
    "operation": "read",
    "client_token": "hmac-sha256:487a19bc11e06e6c0e1003d...",
    "client_token_accessor": "hmac-sha256:34a36a8d835538359ee4fdd...",
    "path": "secret/training",
    "data": null,
    "policy_override": false,
    "remote_address": "127.0.0.1",
    "wrap_ttl": 0,
    "headers": {}
  },
  "error": ""
   ...
```

The audit log is in JSON format. Works well with Elasticsearch, LogStash, and Kibana (a.k.a ELK stack).

OSS vs. Enterprise

Packaging

OSS addresses technical complexity and Enterprise addresses organizational complexity



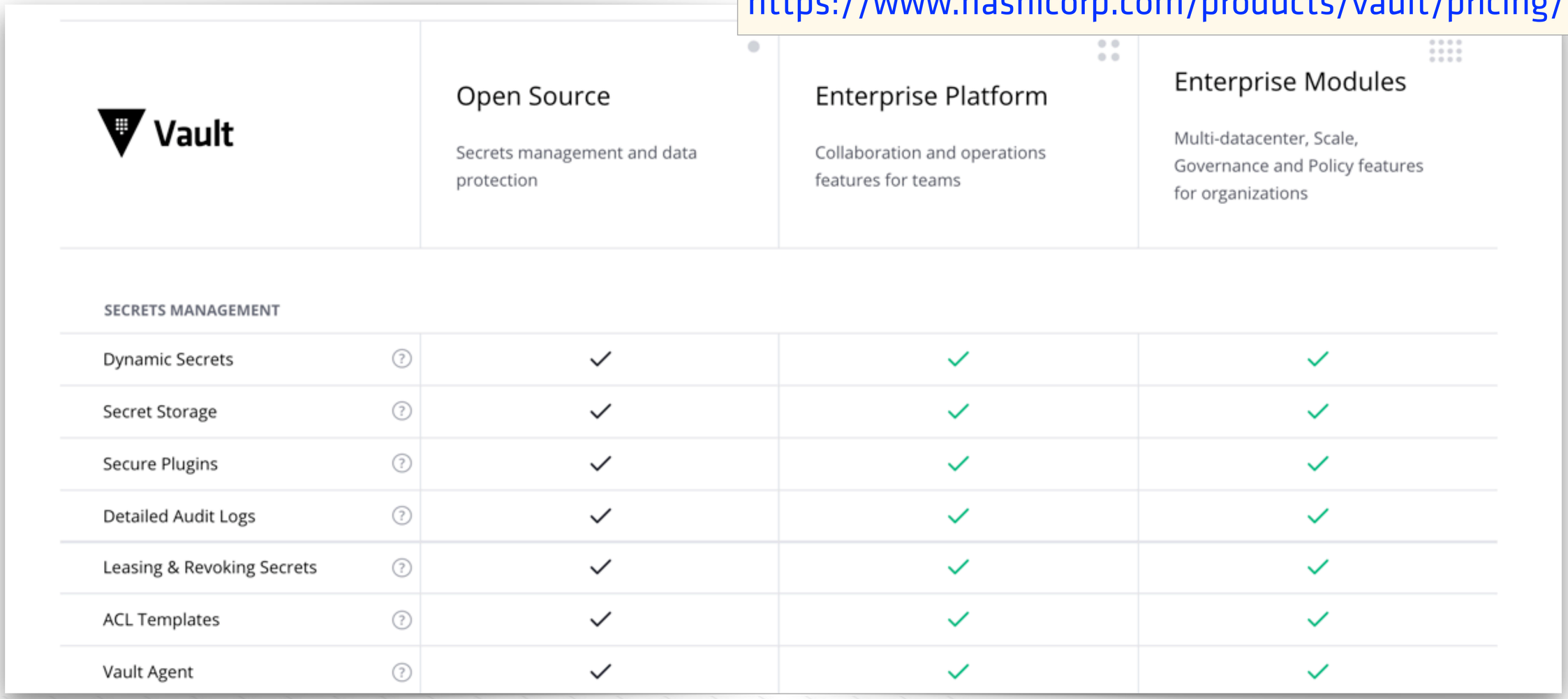
Individuals use Vault **Open Source**

Teams use Vault Enterprise **Platform**

Organizations use Vault Enterprise **Modules**

Vault Packages Comparison

Find the full list at
https://www.hashicorp.com/products/vault/pricing/



Vault	Open Source Secrets management and data protection	Enterprise Platform Collaboration and operations features for teams	Enterprise Modules Multi-datacenter, Scale, Governance and Policy features for organizations
SECRETS MANAGEMENT			
Dynamic Secrets	✓	✓	✓
Secret Storage	✓	✓	✓
Secure Plugins	✓	✓	✓
Detailed Audit Logs	✓	✓	✓
Leasing & Revoking Secrets	✓	✓	✓
ACL Templates	✓	✓	✓
Vault Agent	✓	✓	✓

What's NOT Included in OSS

Feature	Enterprise Platform	Enterprise Modules
Namespaces	✓	✓
Disaster Recovery	✓	✓
Replication		✓
Path Filters		✓
Read Replicas		✓
Control Groups		✓
HSM Integration		✓
Multi-factor Authentication		✓
Sentinel Integration		✓
KMIP		✓
Transform		✓

Multi-Datacenter & Scale

Governance & Policy

Advanced Data Protection

Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which one of the statements is true about Vault:
   - A. Data is encrypted in-transit as well as at-rest
   - B. Manage secret lifecycle
   - C. Revoke credentials in case of under attack
   - D. Provide detailed audit log for traceability
   - ✓ E. All of the above

2. What is secret? Any sensitive, confidential data.

3. True or False: Your encrypted secrets are persisted in Vault itself.

Lab 1: Lab Setup

15 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Connect to the Student Workstation
  - Task 2: Getting Help
  - Task 3: Enable Audit Logging
  - Task 4: Access Vault UI

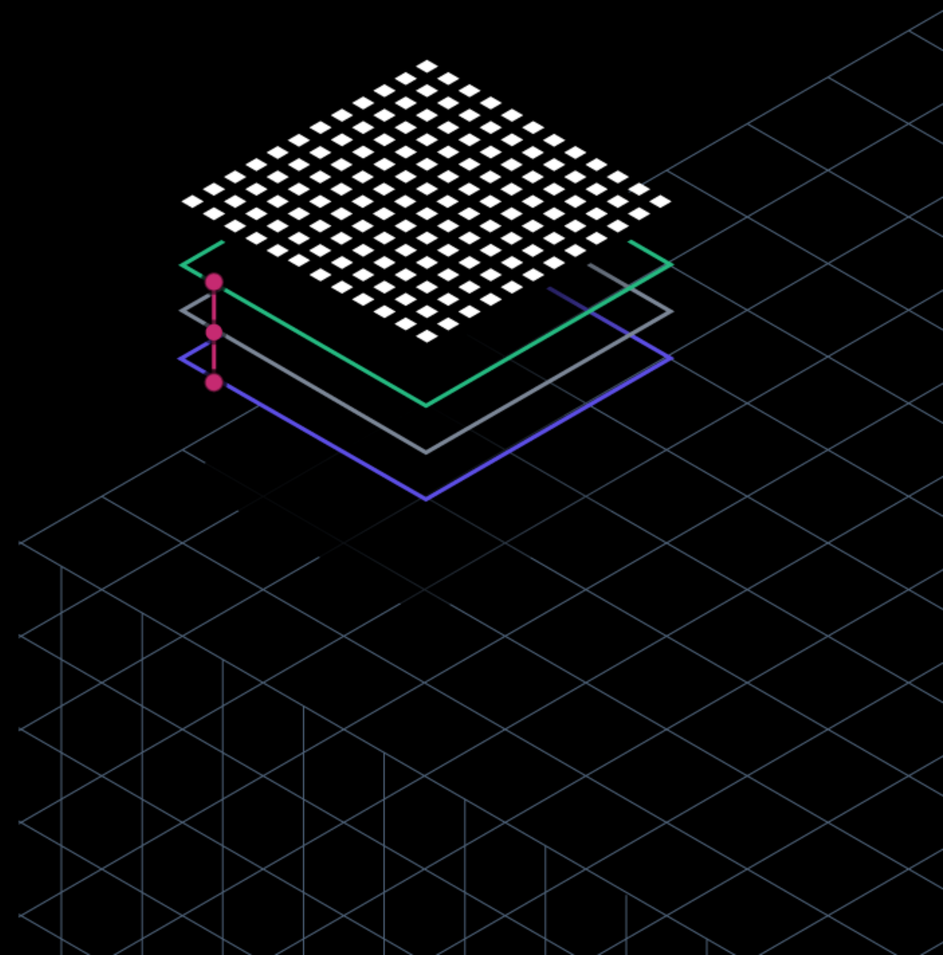
Check your email for your student workstation information!

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp



Vault Installation

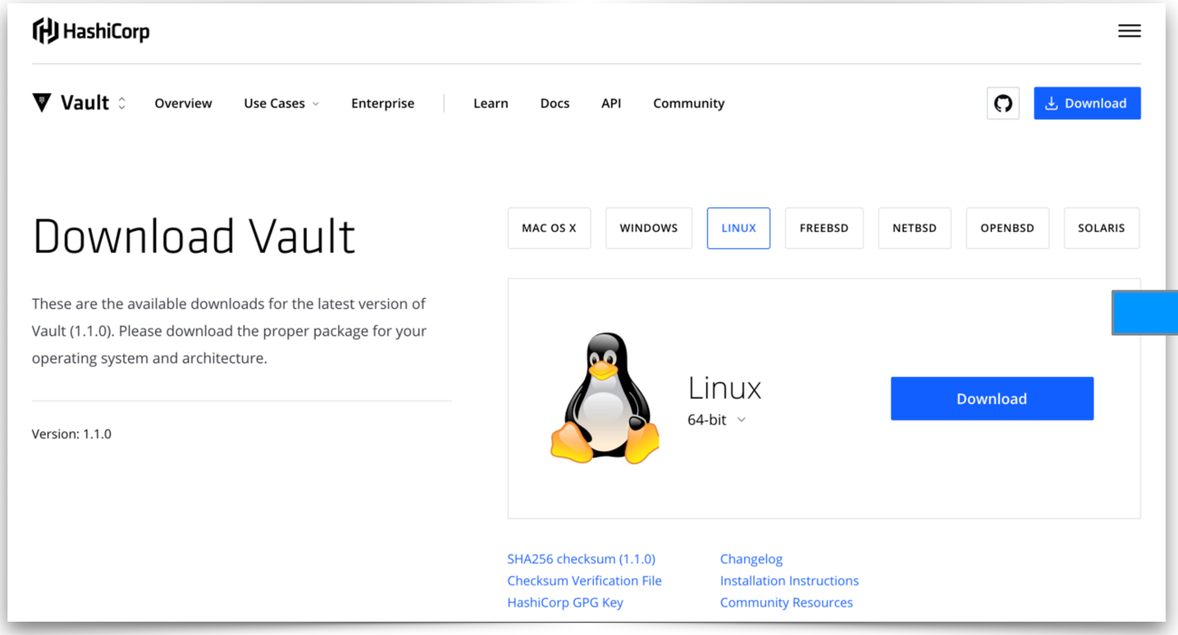
Agenda

- Vault Installation & Configuration
  - Vault binary installation
  - Vault server configuration
  - Initializing Vault
  - Seal / Unseal
- Run Vault in "dev" mode
- Interacting with Vault

Installation & Configuration

Vault Installation

1 Download the Vault binary package for your platform

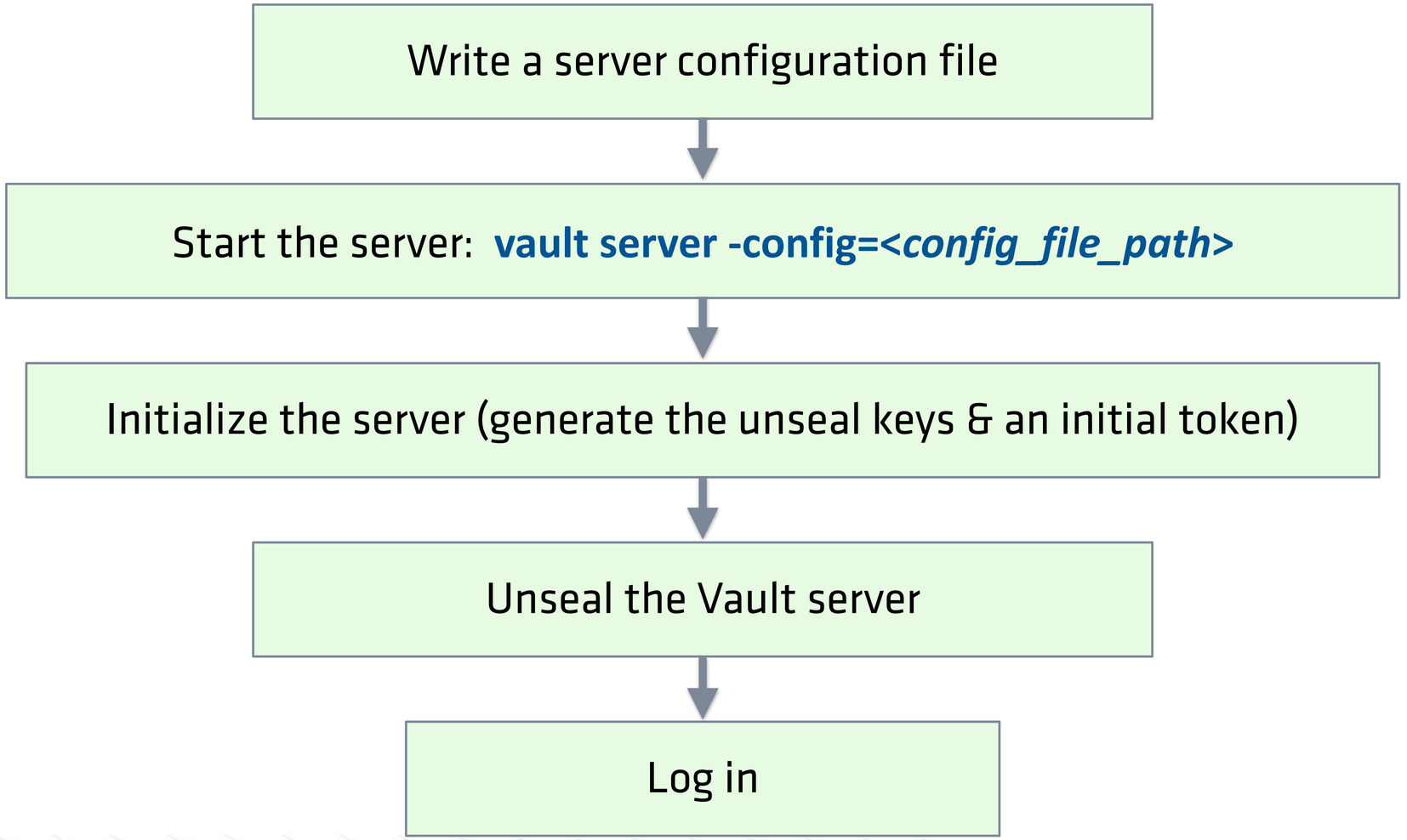


2 Unzip the package and add the **vault** binary to be on the PATH

Terminal

```
$ unzip vault_${VAULT_VERSION}_linux_amd64.zip
$ install -c -m 0755 vault /usr/bin
$ vault -h
```

Vault Server Setup Workflow



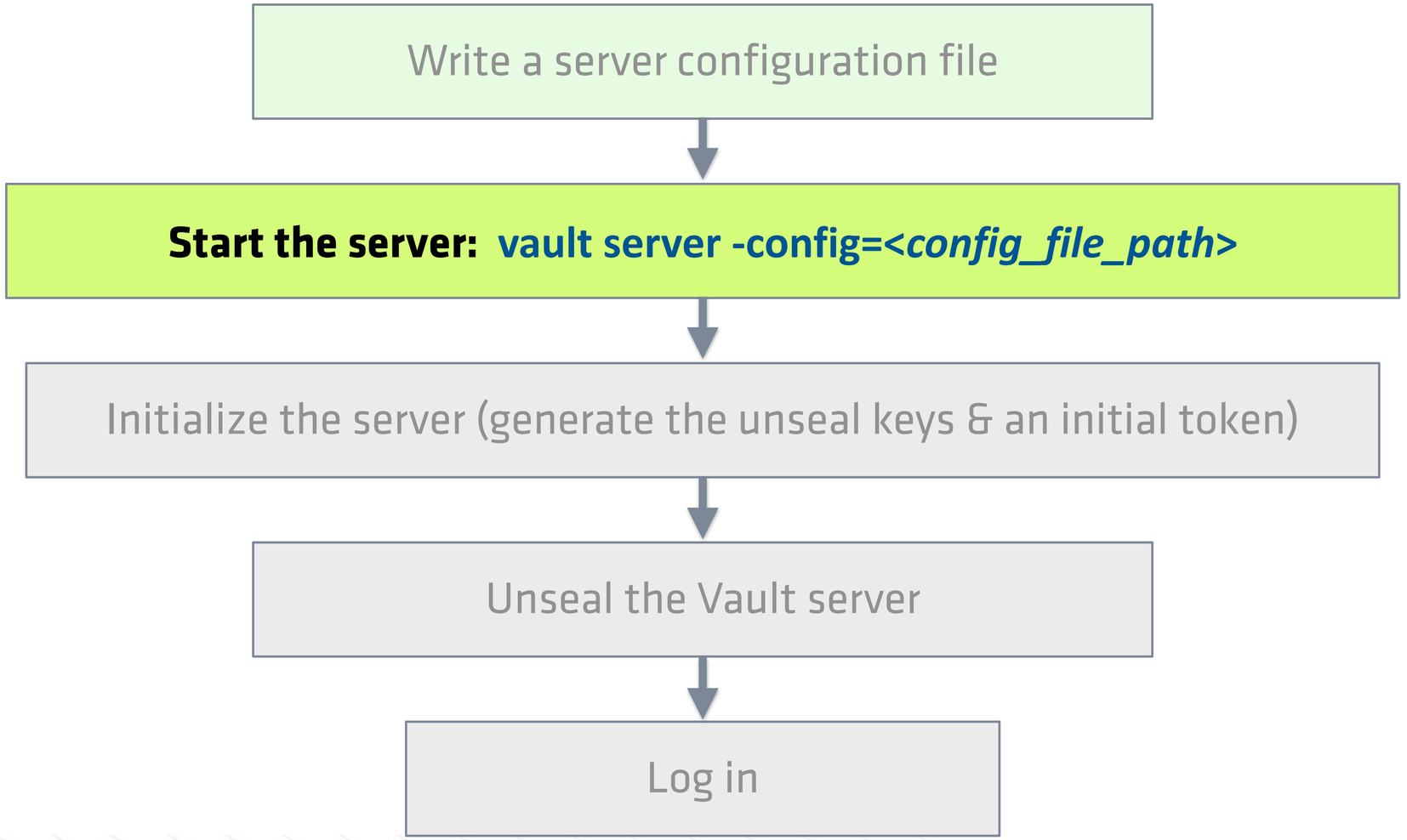
Server Configuration File Example

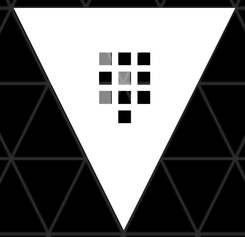
- Vault servers are configured using a file
  - **HCL** or **JSON** format
- Configures:
  - **storage**
  - **listener**
  - seal
  - telemetry
  - ui

config.hcl

```
# Use Consul as its storage backend
storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault"
}

# Configure how Vault listens for API requests
listener "tcp" {
  address     = "127.0.0.1:8200"
  tls_disable = 1
}

# Specifies the telemetry reporting system
telemetry {
  statsite_address = "127.0.0.1:8125"
  disable_hostname = true
}
ui = true
```

Getting Started

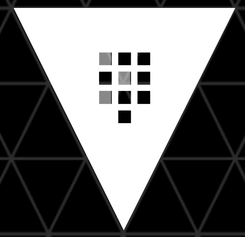




Start Vault Server

Terminal

```
$ vault server -config=config.hcl
...
==> Vault server configuration:

                     Cgo: disabled
              Listener 1: tcp (addr: "0.0.0.0:8200", cluster
address: "0.0.0.0:8201", max_request_duration: "1m30s",
max_request_size: "33554432", tls: "disabled")
                Log Level: info
                    Mlock: supported: false, enabled: false
                  Storage: file
                  Version: Vault v1.1.0+ent
              Version Sha: ecfa8e384947dbfe1aa091c2a866969...

==> Vault server started! Log data will stream in below:
```

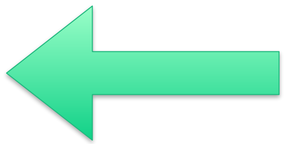


Set the Vault server address as an environment variable so that subsequent CLI commands will be sent to a correct target

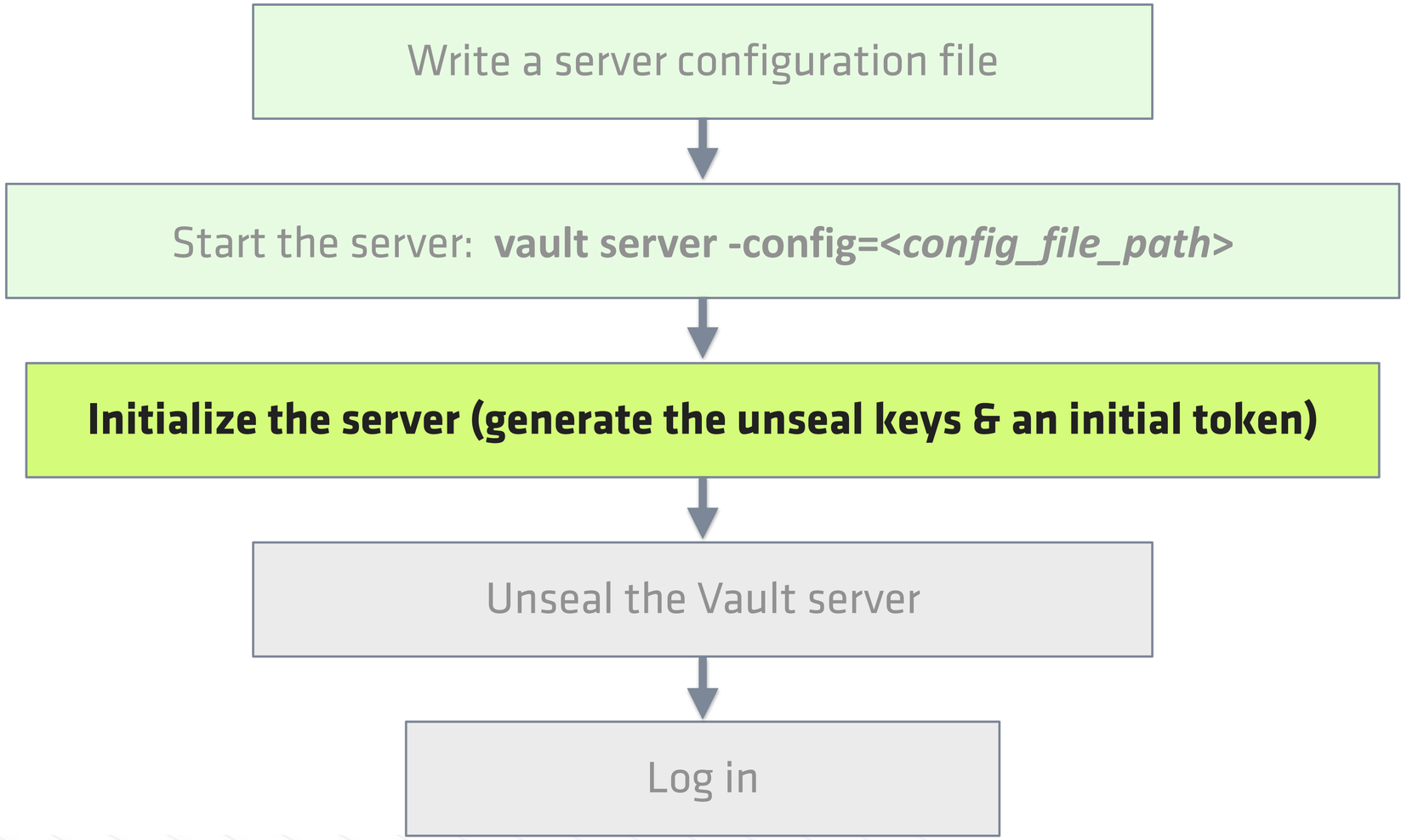
Server Status

Terminal

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

$ vault status

Key                     Value
---                     -----
Seal Type               shamir
Initialized             false
Sealed                  true
Total Shares            0
Threshold               0
Unseal Progress         0/0
Unseal Nonce            n/a
Version                 n/a
HA Enabled              false
```



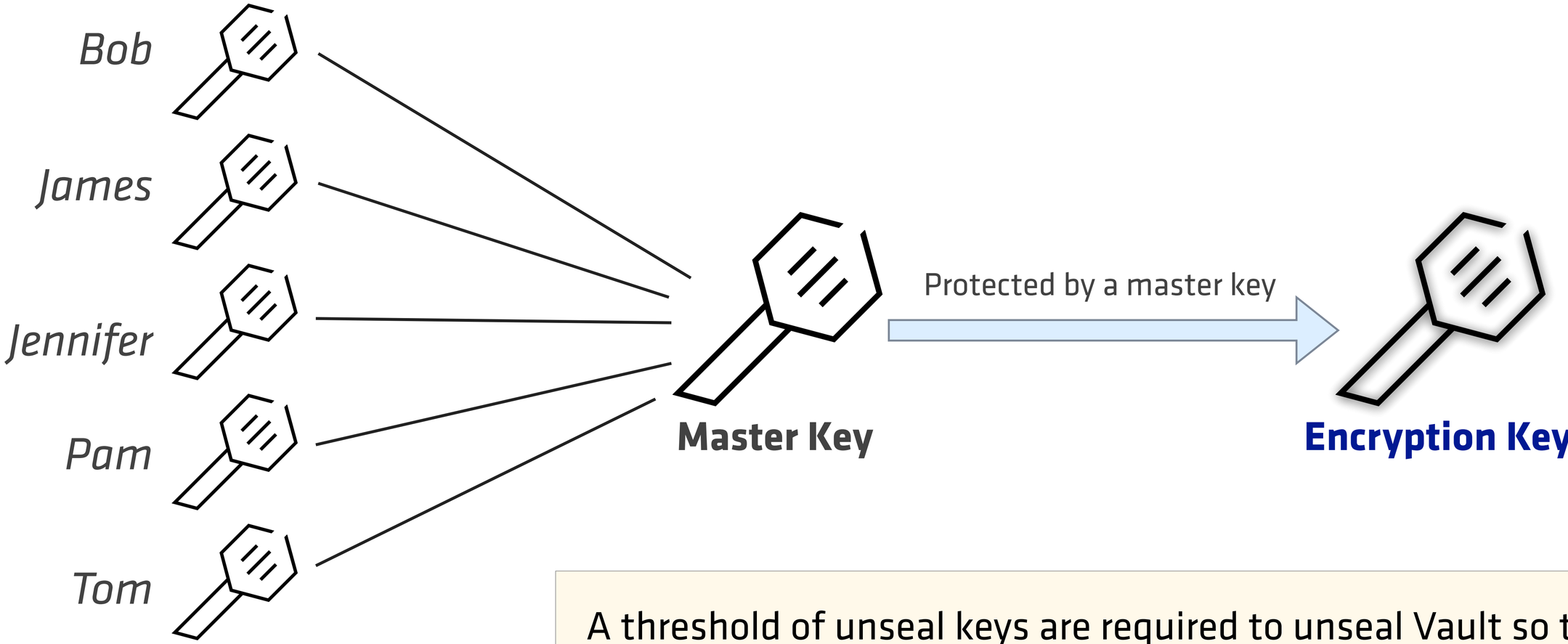
Starting the Vault Server



Vault Server Initialization

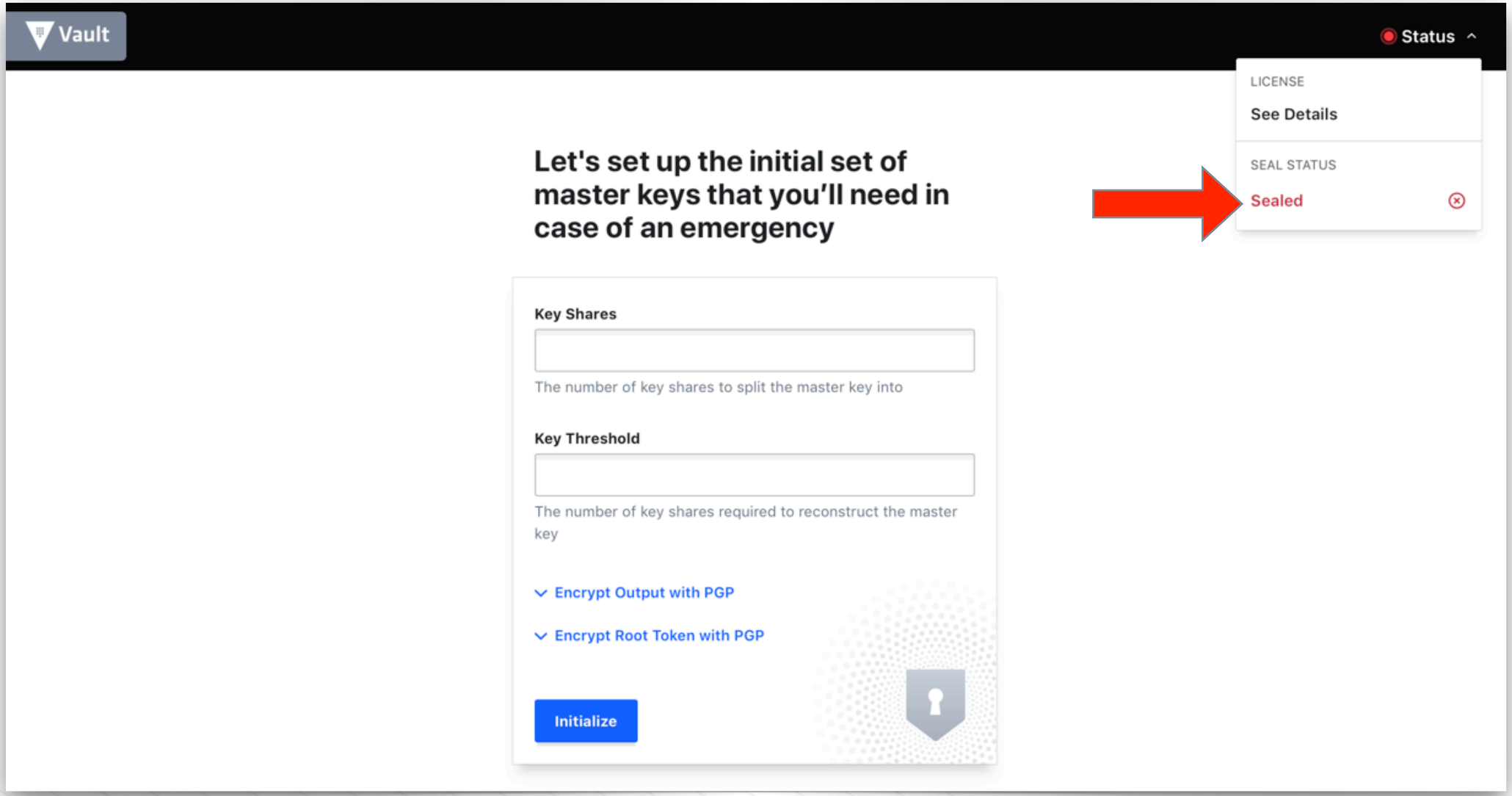
- **Initialization** is the process of configuring the Vault:
  - **Encryption key** gets generated
  - **Unseal keys** are created
  - **Initial root token** is setup

Shamir's Secret Sharing

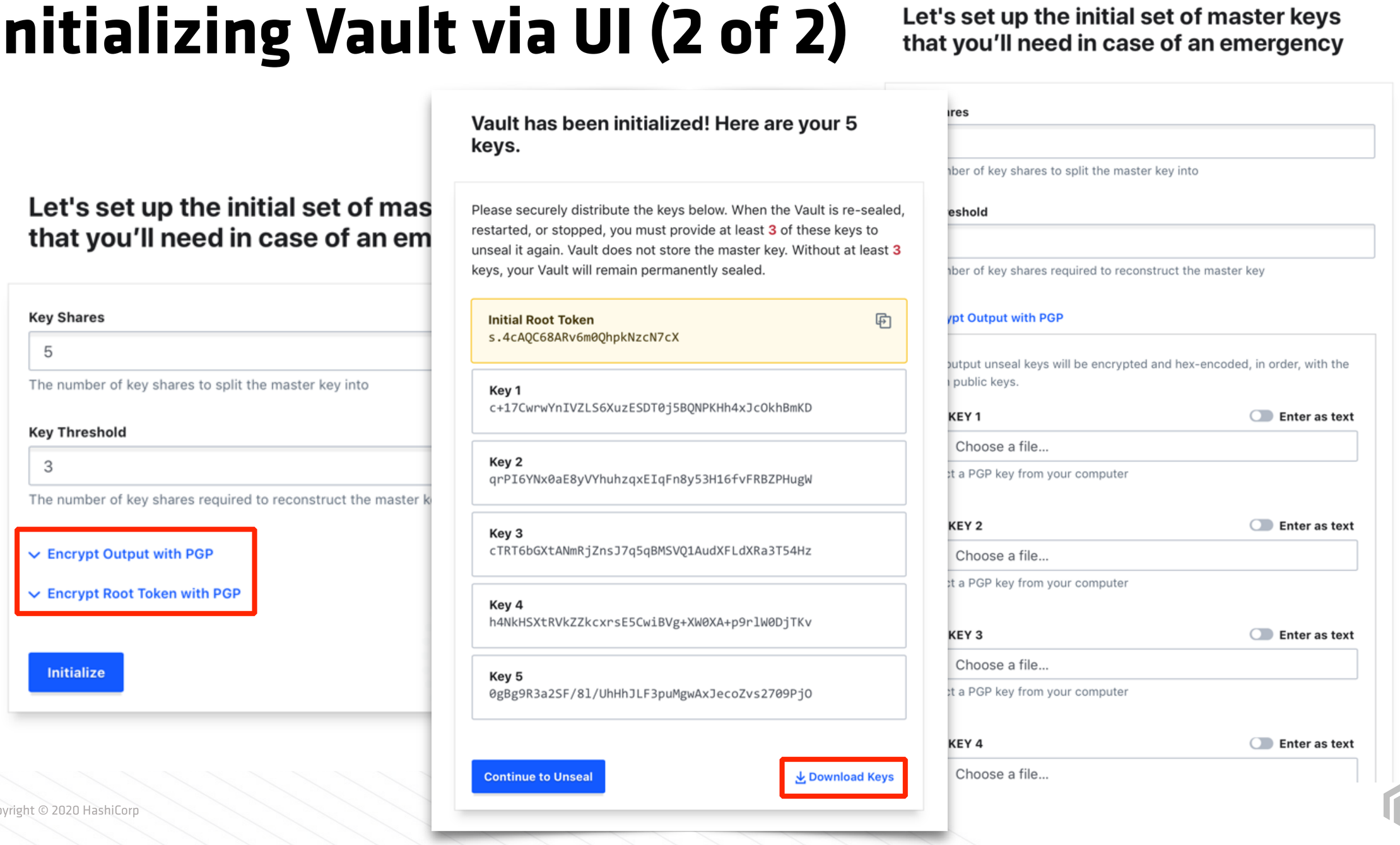


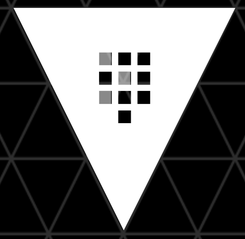
A threshold of unseal keys are required to unseal Vault so that the *key to the kingdom* won't fall into one person's hand!

Initializing Vault via UI (1 of 2)



Initializing Vault via UI (2 of 2)



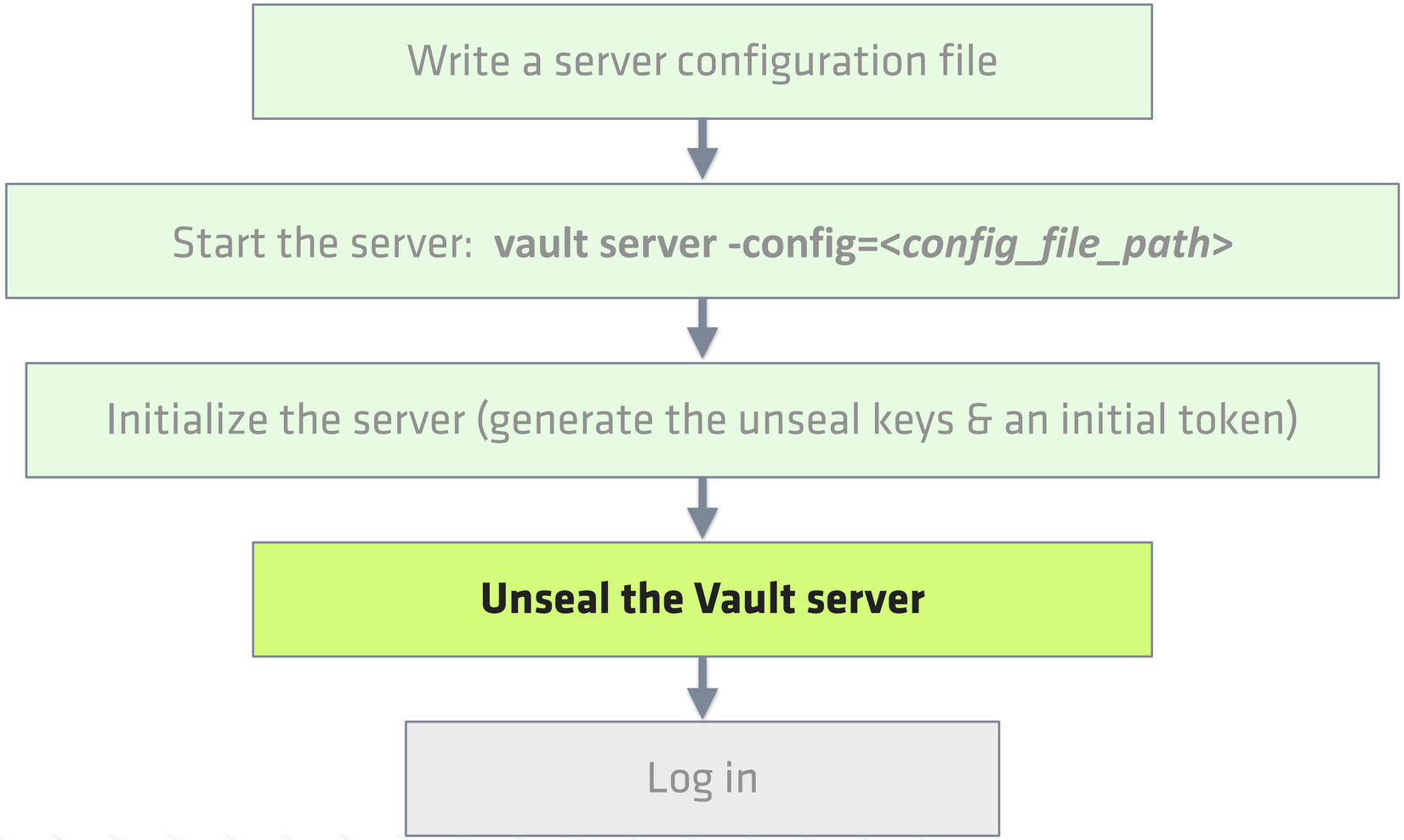


Initialize a Vault Server via CLI

Terminal

```
$ vault operator init
Unseal Key 1: oL8fJP4KreJPbZWIgui340j5bNclip9zGVcYIzElsoF1
Unseal Key 2: Ke9VZlGzuVaf4HJB8c9KQR2j8rFTBALV1fD3hjE5pHoY
Unseal Key 3: 4X6Ja/RpMwNabYzklZKxxXVznLQFGgSiVW7Wx8LWOkQn
Unseal Key 4: dhI04g8dIQSXI11BIC6Gtwy/QaJWhVYoFYwKF9UI6axO
Unseal Key 5: IQ2Ls630Sjd/oEQyTmwwpuFEUTiJP4FX2UI3uZMZoa+x

Initial Root Token: s.arHAbYvyeZQH8StLc5OHtbt4

Vault initialized with 5 keys and a key threshold of 3. Please
securely distribute the above keys. When the Vault is re-sealed,
restarted, or stopped, you must provide at least 3 of these keys
to unseal it again.

...
```

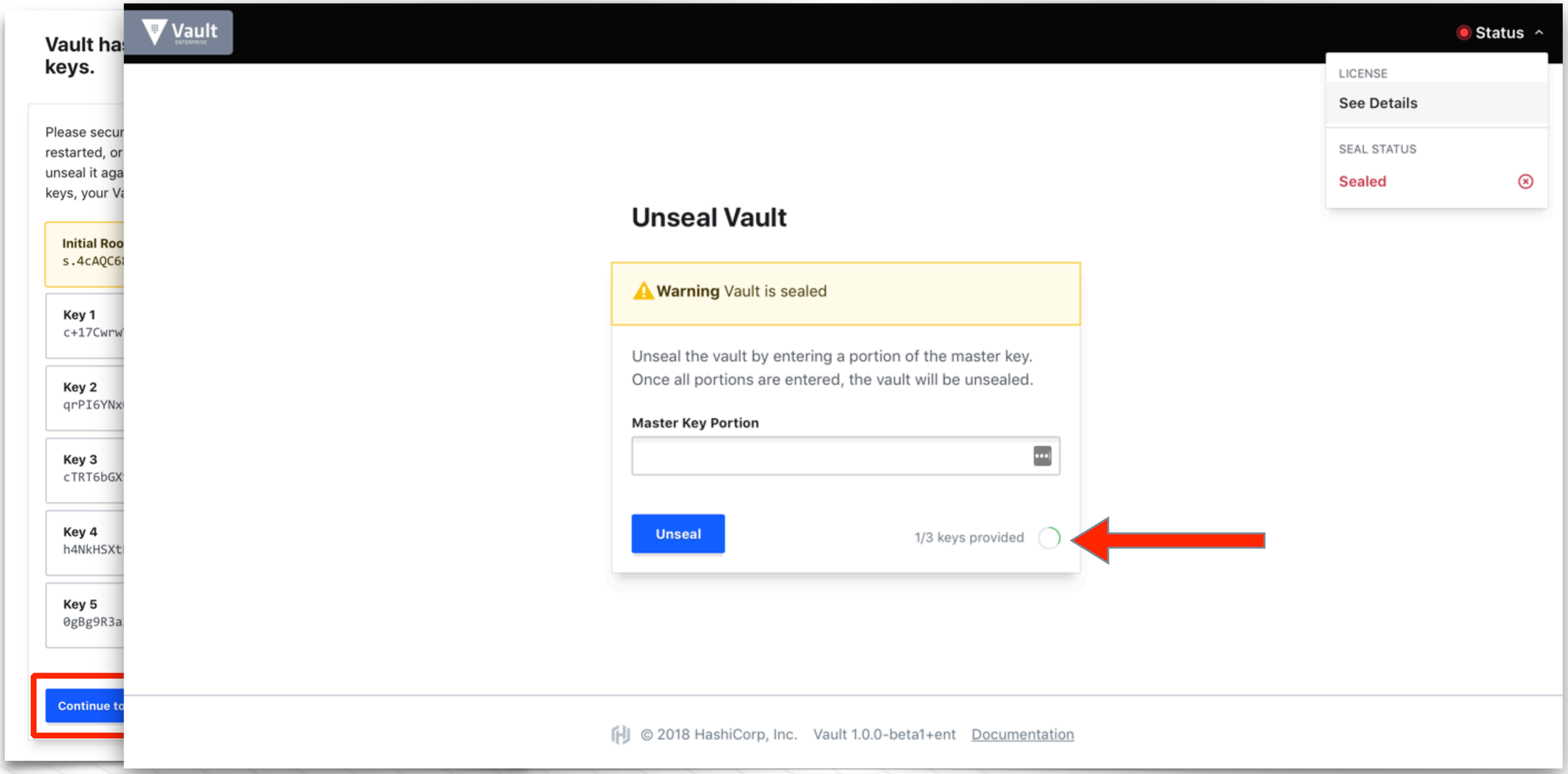
Starting the Vault Server

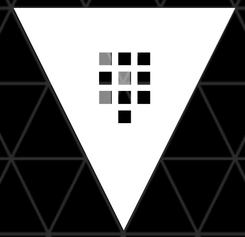


Seal / Unseal

- When a Vault server is started, it starts in ***sealed*** - doesn't know how to decrypt the data
- ***Unsealing*** is the process of constructing the ***master key*** necessary to read the decryption key to decrypt data
- Why?
  - The data stored by Vault is encrypted with encryption key
  - The encryption key is encrypted with master key
  - The master key is encrypted with unseal keys

Unsealing Vault via UI



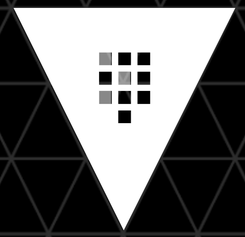


Unsealing via CLI

(1 of 3)

Terminal

```
$ vault operator unseal
Unseal Key (will be hidden):

Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             true
Total Shares       5
Threshold          3
Unseal Progress    1/3
Unseal Nonce       3c0b2c85-6d22-54e4-87ce-249061dd9d1c
Version            1.1.0
HA Enabled         false
```



Unsealing via CLI

(2 of 3)

Terminal

```
$ vault operator unseal
Unseal Key (will be hidden):

Key                    Value
---                    -----
Seal Type              shamir
Initialized            true
Sealed                 true
Total Shares           5
Threshold              3
Unseal Progress        2/3
Unseal Nonce           3c0b2c85-6d22-54e4-87ce-249061dd9d1c
Version                1.1.0
HA Enabled             false
```

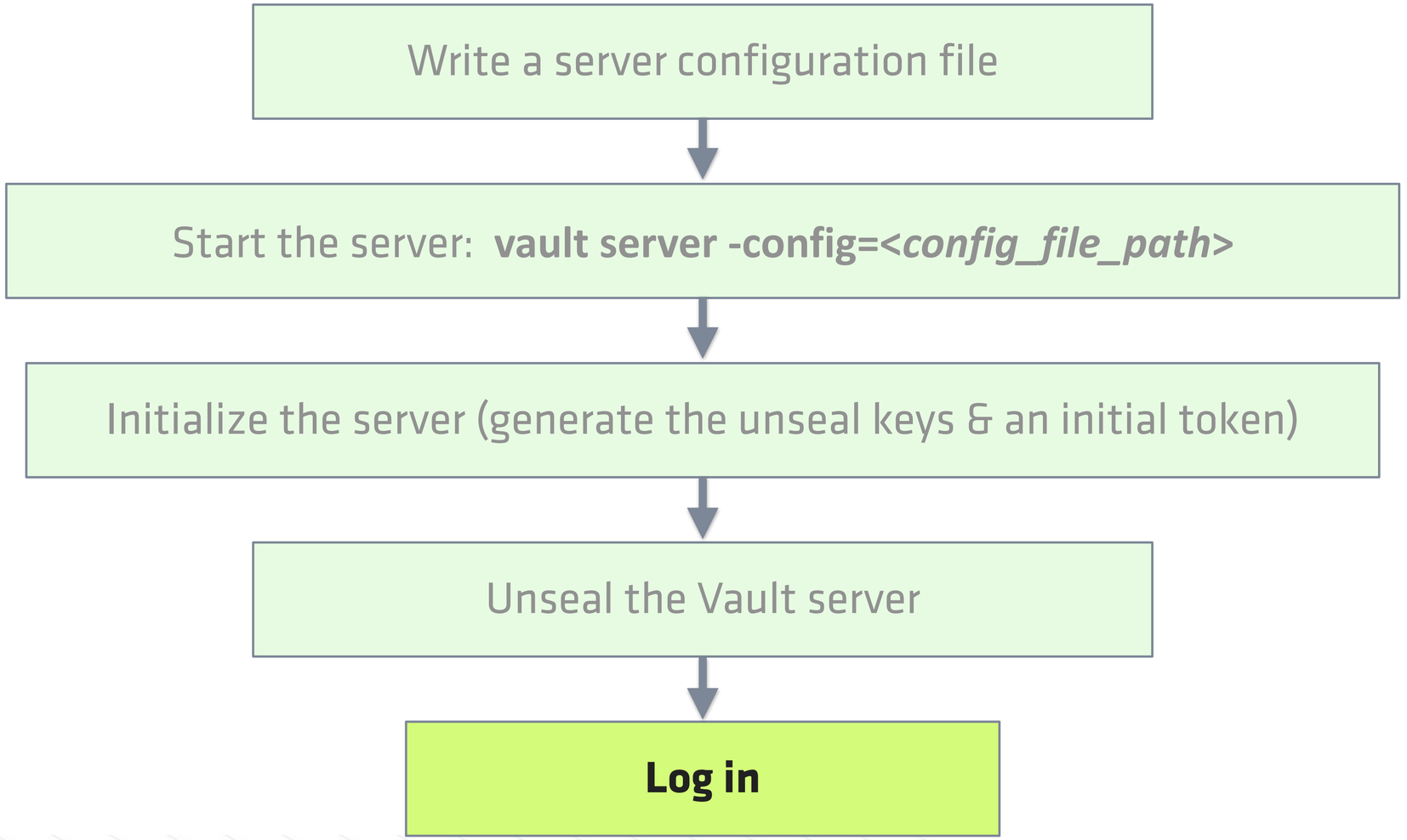
Unsealing via CLI

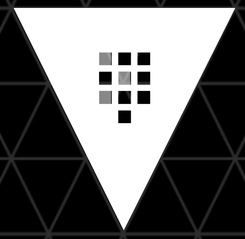
(3 of 3)

Terminal

```
$ vault operator unseal
Unseal Key (will be hidden):

Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             false
Total Shares       5
Threshold          3
Version            1.1.0
Cluster Name       vault-cluster-ad3f168d
Cluster ID         9fcbb3bc-6d9b-98f5-3f2e-a0cf1040a260
HA Enabled         false
```

Starting the Vault Server





Login

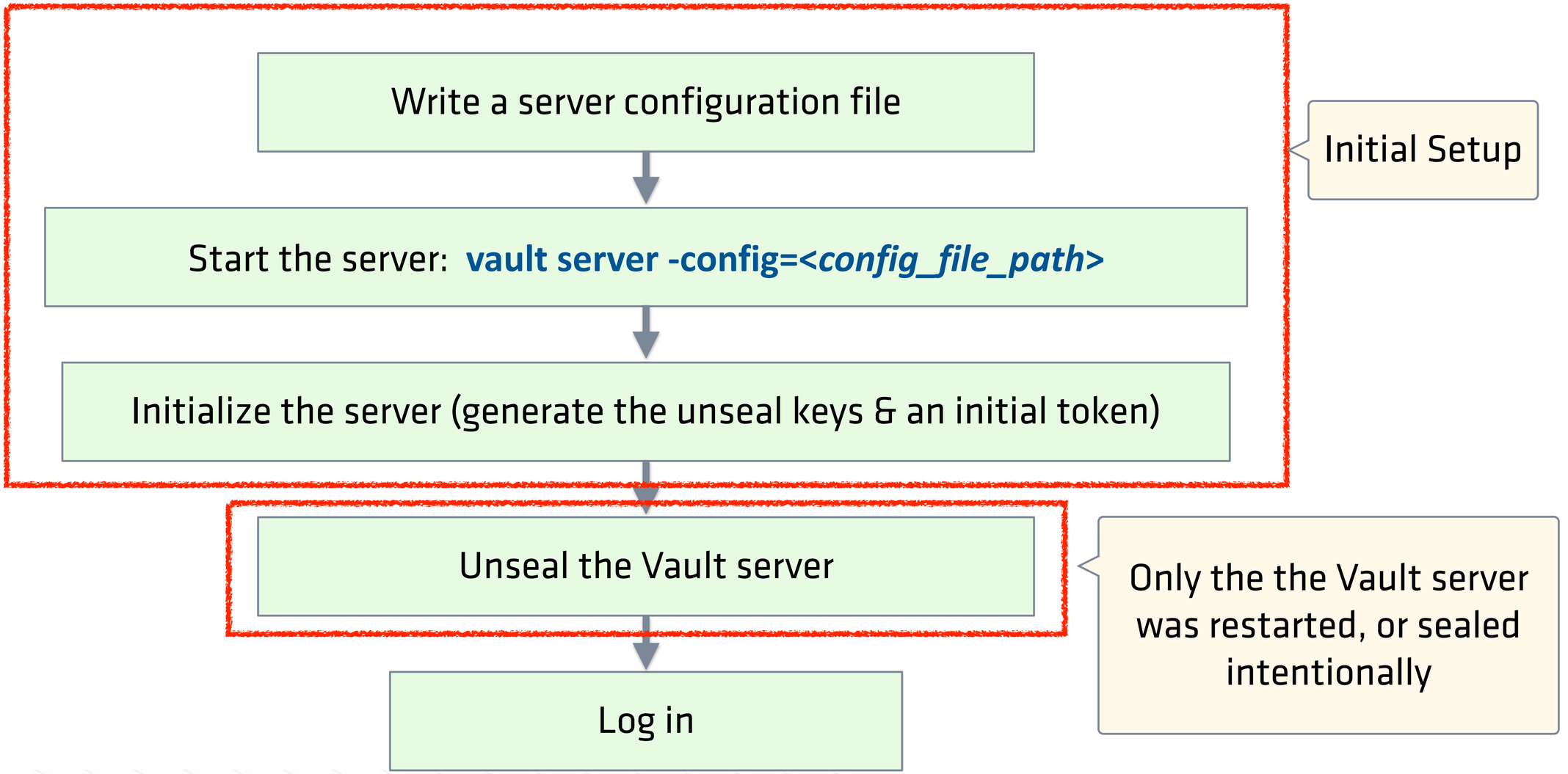
Terminal

```
$ vault login <initial_root_token>
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.arHAbYvyeZQH8StLc5OHtbt4
token_accessor       exovZk0TtbPOhRzkzQOIiZQd
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Initial Root Token

- Authenticating with **root** token should be limited
- Use the root token to:
  - Setup the initial policies
  - Enable and configure initial secret engines and auth methods
  - Create namespaces (Vault Enterprise)
  - Handle emergencies where root privilege is needed

Vault Server Setup Workflow Recap



Question

Why or when would anyone want to ***seal*** their Vault?

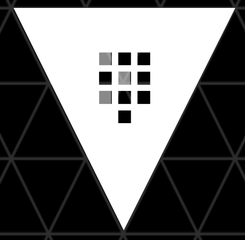
```
$ vault operator seal
```

A: Seal the Vault as an ultimate ***break glass*** procedure. For example, you suspect that someone has *unauthorized* access to Vault with wrong intention.

Dev mode

"Dev" Mode

- The "dev" server is a built-in, pre-configured server
  - Useful for local **development**, **testing**, and **exploration**
  - Not very secure
  - Everything is stored **in-memory**
  - Vault is **automatically unsealed**
  - Can optionally set the initial root token



"Dev" mode

Terminal

```
$ vault server -dev
==> Vault server configuration:

                     Api Address: http://127.0.0.1:8200
                     Cgo: disabled
          Cluster Address: https://127.0.0.1:8201
               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: (not set)
                    Mlock: supported: false, enabled: false
                  Storage: inmem
                  Version: Vault v1.0.0+ent
              Version Sha: 019967f8bcf35d0a882ce83b6b66e820d362855b

WARNING! dev mode is enabled! In this mode, Vault runs entirely in-memory
and starts unsealed with a single unseal key. The root token is already
authenticated to the CLI, so you can immediately begin using Vault.

You may need to set the following environment variable:

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

Interacting with Vault

Interacting with Vault (1 of 2)

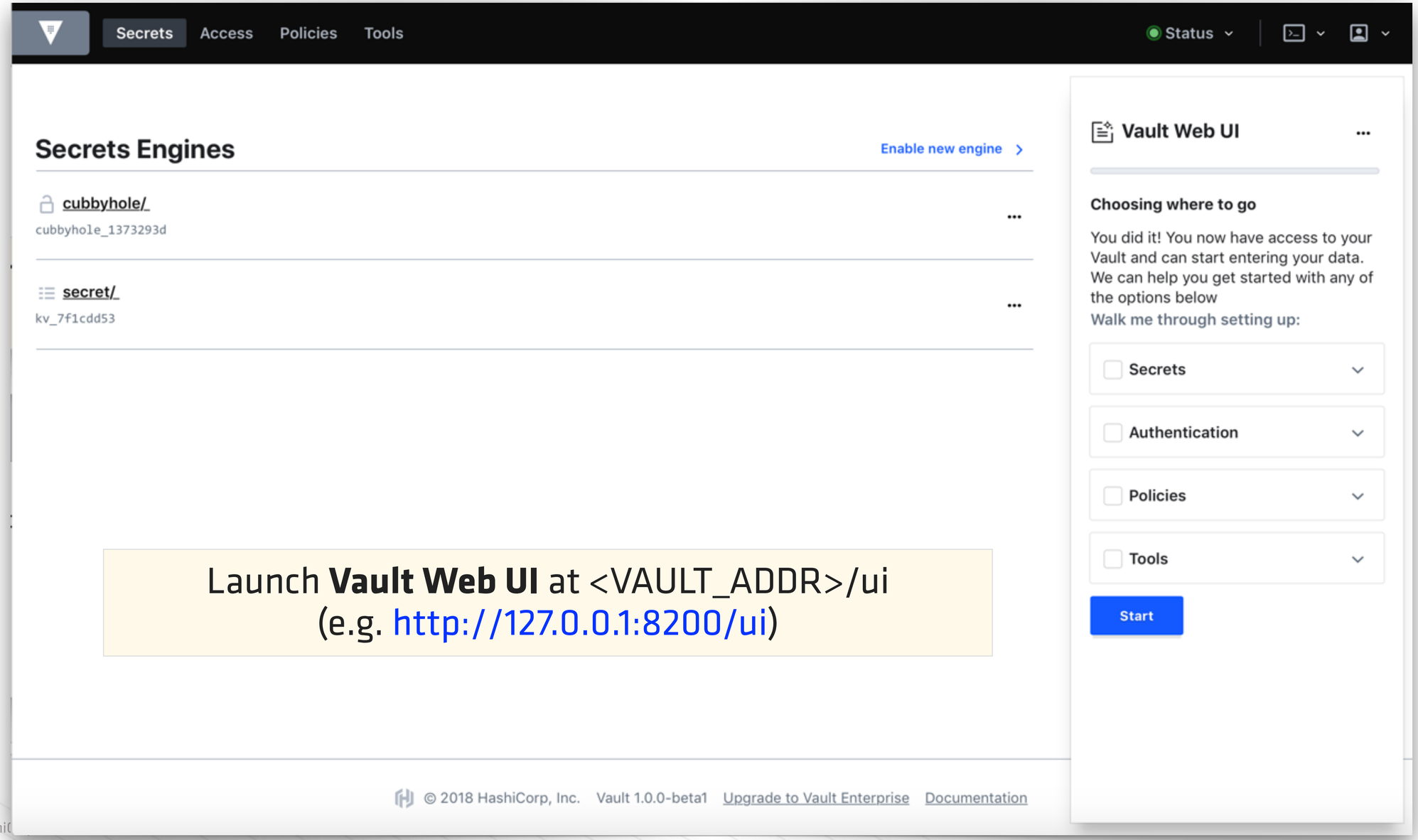
- Vault **HTTP API**
  - Full access to Vault via HTTP
  - Every aspect of Vault can be controlled via API

```
# Getting help
$ vault path-help <path>
```

- Vault **CLI**
  - Uses the HTTP API to access Vault
  - Thin wrapper around the HTTP API (wraps common functionality)
  - Outputs are formatted

```
# Getting help
$ vault <command> -h
```

Interacting with Vault (2 of 2)



Launch **Vault Web UI** at <VAULT_ADDR>/ui
(e.g. http://127.0.0.1:8200/ui)

Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which of the following statements **are** true about Vault **initialization**:
   - ✓ A. Encryption key gets generated
   - ✓ B. Unseal keys are created
   - ✓ C. Initial root token is setup
   - D. Audit log is enabled

2. **Unsealing** is the process of decrypting the master key.

Lab 2: Vault Server Configuration

25 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Configure and setup Vault
  - Task 2: Connect to Vault UI
  - Task 3: Explorer K/V Secrets Engine

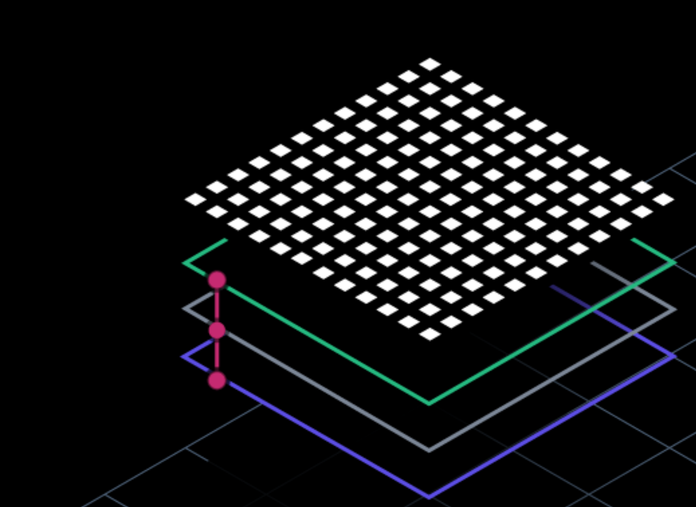
Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp

Auto-Unseal



Agenda

- Auto-unseal with cloud provider keys
- Vault integration with HSM (Vault Enterprise)
  - Seal Wrap

Auto-unseal with Cloud Provider Keys

Challenge

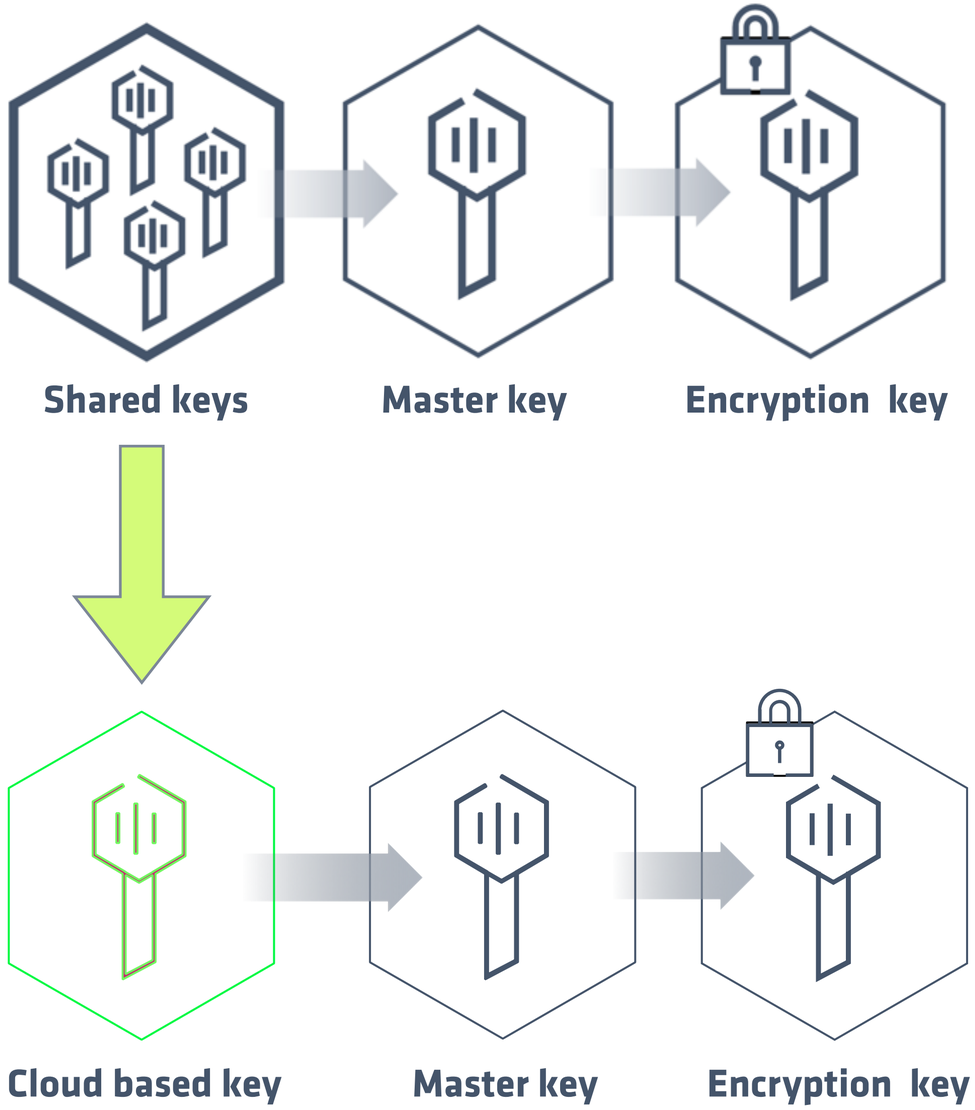
Unsealing process requires a threshold of unseal keys

No single person holds the key to the Vault kingdom!

Unsealing is a **manual** process and become **painful** when you have multiple Vault clusters

Auto-Unseal Vault

- Instead of using shared keys based on Shamir's Secret Sharing algorithm, use the trusted **cloud-based encryption key** to protect the master key
- Supported cloud services today:
  - AliCloud KMS
  - AWS KMS
  - Azure Key Vault
  - GCP Cloud KMS
  - OCI KMS
- Use *Transit* secrets engine



Enabling Auto-Unseal

- To enable auto-unseal, your Vault configuration file must declare the **seal** stanza
- Use one of the following:
  - alicloudkms
  - awskms
  - azurekeyvault
  - gcpckms
  - ocikms
  - transit

Example Configuration File (config.hcl)

```
ui = true

storage "consul" {
  address = "127.0.0.1:8500"
  path = "vault/"
}

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

seal "[TYPE]" {

  Seal type specific configuration parameters

}
```


AliCloud KMS Integration

- Add **seal** stanza to your server configuration and set it to **alicloudkms**
- Set the credentials as environment variables:
  - ALICLOUD_REGION
  - ALICLOUD_ACCESS_KEY
  - ALICLOUD_SECRET_KEY

config.hcl

```
...
seal "alicloudkms" {
  kms_key_id = "08c33a6f-4e0a-4a1b-a3fa-7ddfa1d4fb73"
  region     = "us-east-1"
  access_key = "0wNEpMMlzy7szvai"
  secret_key = "PupkTg8jaaaabbbXxYacgE736PJj4cA"
}
...
```

If Vault is hosted on AliCloud, use Resource Access Management (**RAM**) role instead

AWS KMS Integration

- Add **seal** stanza to your server configuration and set it to **awskms**
- Set the credentials as environment variables:

  AWS_REGION
  AWS_ACCESS_KEY_ID
  AWS_SECRET_ACCESS_KEY

config.hcl

```
...
seal "awskms" {
    kms_key_id = "19ec80b0-dfdd-4d97-example"
    region     = "us-east-1"
    access_key = "AKIAIOSFODNN7EXAMPLE"
    secret_key = "wJalrXUtnFEMI/K7Mexample5"
}
...
```

If Vault is hosted on EC2 or ECS, use **IAM role** with KMS permissions instead

Benefit: AWS KMS is a secure and resilient service that uses FIPS 140-2 validated hardware security modules to protect your keys.

Azure Key Vault Integration

- Add **seal** stanza to your server configuration and set it to **azurekeyvault**
- Set the credentials as environment variables:

  AZURE_TENANT_ID
  AZURE_CLIENT_ID
  AZURE_CLIENT_SECRET
  AZURE_ENVIRONMENT

config.hcl

```
...
seal "azurekeyvault" {
    tenant_id      = "46646709-b63e-4example"
    client_id      = "03dc33fc-16d9-example"
    client_secret  = "DUJDS3..."
    vault_name     = "hc-vault"
    key_name       = "vault_key"
}
...
```

If Vault is hosted on Azure, use **Managed Service Identities** (MSI) instead

Benefit: Azure Key Vault processes your keys in FIPS 140-2 Level 2 validated HSMs (hardware and firmware).

GCP Cloud KMS Integration

- Add **seal** stanza to your server configuration and set it to **gcpckms**
- Set the credentials as environment variables:

  GOOGLE_CREDENTIALS
  GOOGLE_PROJECT
  GOOGLE_REGION

config.hcl

```
...
seal "gcpckms" {
  credentials = "/usr/vault-project-user-creds.json"
  project     = "vault-project"
  region      = "global"
  key_ring    = "vault-keyring"
  crypto_key  = "vault-key"
}
...
```

If Vault is hosted on GCE, set the instance's service account with Cloud KMS role instead of providing credentials here

Benefit: GCP Cloud KMS allows you to easily generate, use, rotate, and destroy cryptographic keys.

OCI KMS Integration

- Add **seal** stanza to your server configuration and set it to **ocikms**
- Set the credentials as environment variables:

  VAULT_OCIKMS_CRYPTO_ENDPOINT

  VAULT_OCIKMS_MANAGEMENT_ENDPOINT

  VAULT_OCIKMS_SEAL_KEY_ID

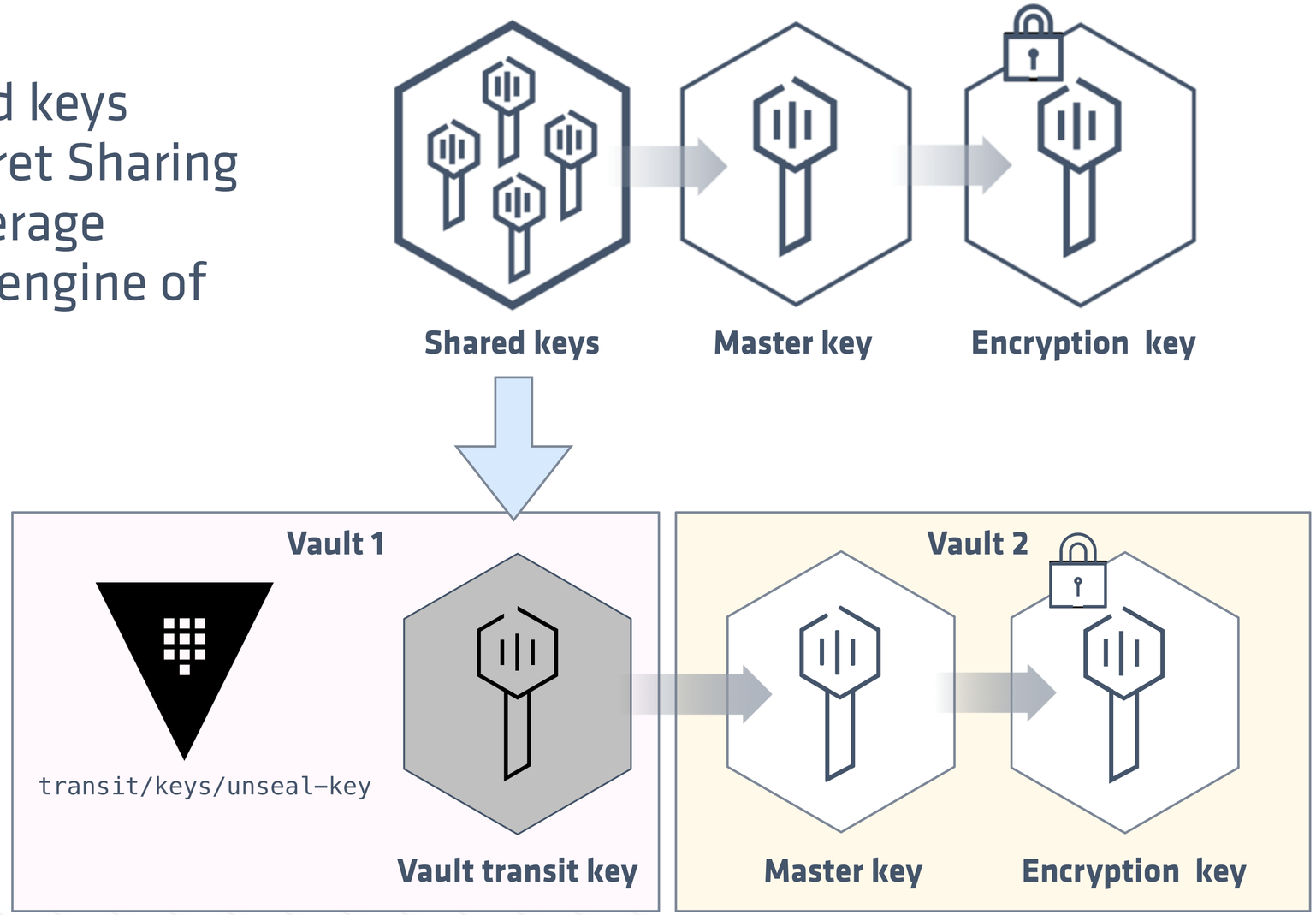
config.hcl

```
...
seal "ocikms" {
   auth_type_api_key    = "true"
   key_id               = "ocid1.key.oc1.iad.afnx..."
   crypto_endpoint      = "https://my-crypto.kms.oraclecloud.com"
   management_endpoint  = "https://my-mgmt.kms.us-1.oraclecloud.com"
}
...
```

If Vault is hosted on OCI, you can grant permission for the compute instance to use OCI KMS service (write policies for KMS access)

Auto-unseal with Transit Secrets Engine (1 of 3)

- Instead of using shared keys based on Shamir's Secret Sharing algorithm, you can leverage Vault's ***transit*** secrets engine of ***another*** Vault server



Auto-unseal with Transit Secrets Engine (2 of 3)

- Add **seal** stanza to your server configuration and set it to **transit**
- Use environment variables:
  - VAULT_TOKEN
  - VAULT_CA_CERT
  - VAULT_CLIENT_CERT
  - VAULT_CLIENT_KEY
  - etc.

Requires Vault 1.1 or later

config.hcl

```
...
seal "transit" {
   token            = "s.m518biFFaJ1QE0zYk0czeogs"
   address          = "https://10.0.101.123:8200"
   disable_renewal  = "false"
   key_name         = "auto-unseal"
   mount_path       = "transit/"
   namespace        = "ns1/"
   tls_skip_verify  = "true"
}
...
```

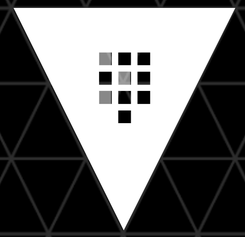
This token needs "update" capability on the transit key. Highly recommended to set this as an environment variable, **VAULT_TOKEN**

Auto-unseal with Transit Secrets Engine (3 of 3)

- The client token used to authenticate with the key provider Vault server must allow appropriate permissions to encrypt and decrypt the master key

Auto-unseal Client's Policy

```
path "transit/encrypt/autounseal-key" {
     capabilities = [ "update" ]
}

path "transit/decrypt/autounseal-key" {
     capabilities = [ "update" ]
}
```

Start & Initialize

Terminal

```
# Start the Vault server
$ vault server -config=/etc/vault.d/config.hcl


# Still need to initialize for the very first time to obtain
# initial root token & recovery keys
$ vault operator init -recovery-shares=3 \
      -recovery-threshold=2
```

With auto-unseal, use the **recovery keys** to regenerate root token, key rotation, etc.

HSM Support (Vault Enterprise)

HSM Support

- *Vault Enterprise* integrates with Hardware Security Module (HSM) to take advantage of HSMs to provide three pieces of special functionality:
  - **Master Key Wrapping**: Vault protects its master key by transiting it through the HSM for encryption rather than splitting into key shares
  - **Auto Unsealing**: Vault stores its HSM-wrapped master key in storage, allowing for automatic unsealing
  - **Seal Wrapping** to provide FIPS KeyStorage-conforming functionality for Critical Security Parameters

Master Key Wrapping and Auto-unseal

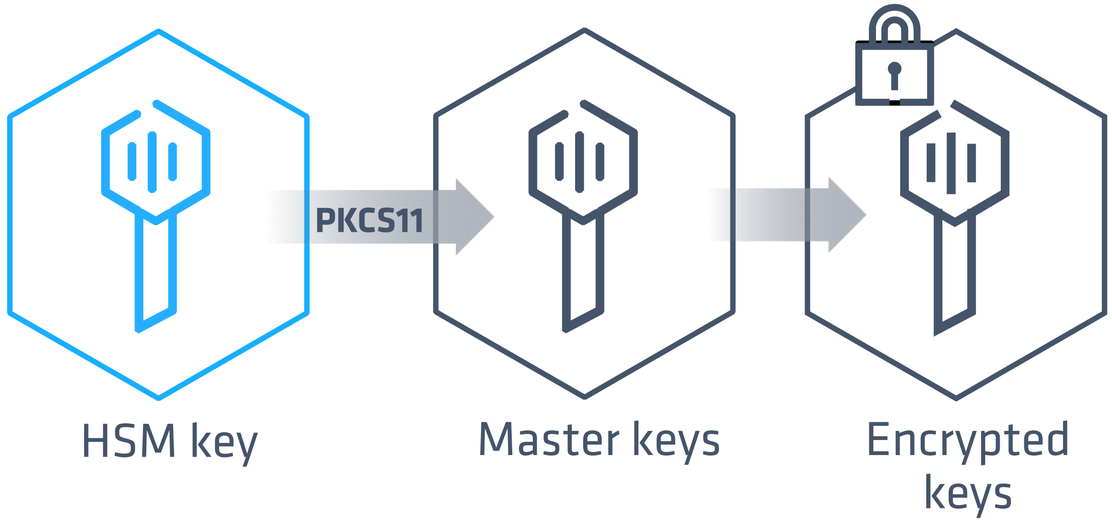
- Protect **encryption key** with master key
- HSM encryption key protects master key in place of Shamir's Secret Sharing
- Communication with HSM via PKCS #11 API to decrypt the master key

SafeNet

THALES

AWS CloudHSM

utimaco®



Vault Configuration with HSM

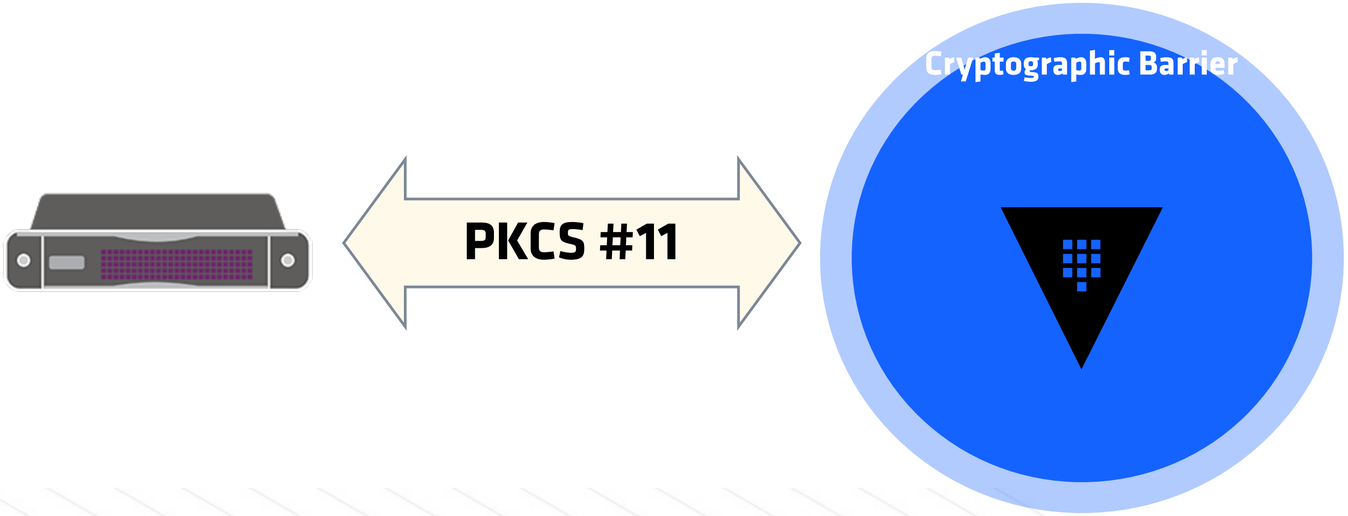
- Vault Enterprise's HSM PKCS #11 support is activated by one of the following:
  - The presence of a seal "**pkcs11**" block in Vault's configuration file
  - The presence of the environment variable **VAULT_HSM_LIB** set to the library's path as well as **VAULT_HSM_TYPE** set to **pkcs11**

```
seal "pkcs11" {
  lib             = "/usr/vault/lib/libCryptoki2_64.so"
  slot            = "0"
  pin             = "AAAA-BBBB-CCCC-DDDD"
  key_label       = "vault-hsm-key"
  hmac_key_label = "vault-hsm-hmac-key"
}
```

Seal Wrap

Seal Wrap

- **Double wrap** the cryptography within Vault using an HSM's crypto modules and random number generator
- Seal Wrap is *always enabled* for:
  - **Cryptographic Barrier**: In-flight and at-rest sealing encryption for Vault
  - **PKI Backend**: Keys generated and used by Vault as a CA



Enabling Seal Wrap (1 of 2)

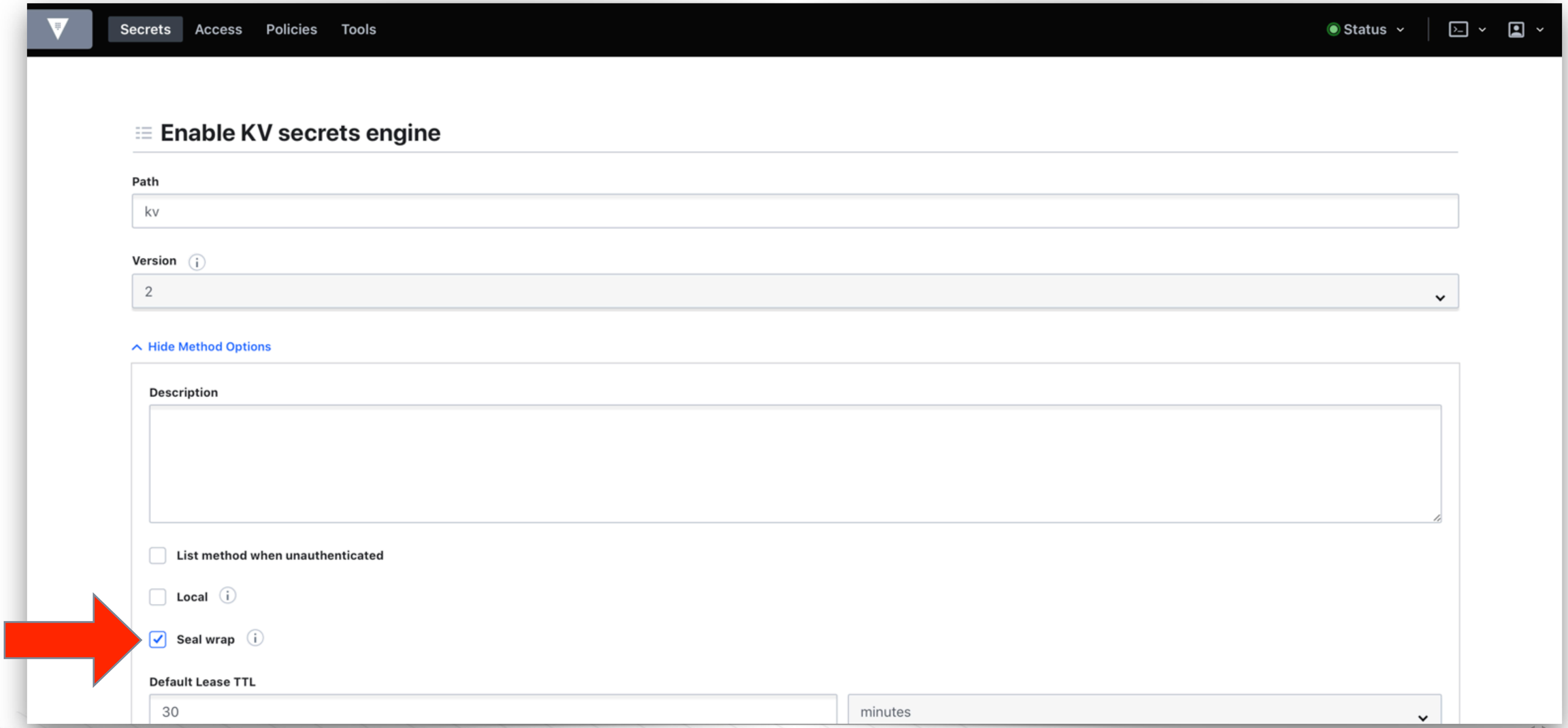
- CLI:

```
$ vault secrets enable -seal-wrap kv
```

- API:

```
$ tee payload.json <<EOF
{
  "type": "kv",
  "options": { "version": "2" },
  "seal_wrap": true
}
EOF

$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data @payload.json \
       http://127.0.0.1:8200/v1/sys/mounts/secret2
```

Enabling Seal Wrap (2 of 2)



Benefits of the Vault Seal Wrap

- General compliance of FIPS 140-2 as certified by Leidos
- Supports FIPS level of security equal to HSM:
  - For example, if you use Level 3 Hardware Encryption on an HSM, Vault will be using FIPS 140-2 Level 3 cryptography
    - This would make Vault the most "*provably secure*" cryptography product on the market
- Allows Vault to be deployed in high security governance, risk management, and compliance (GRC) environments
  - e.g. Federal, some extreme interpretations of PCI-DSS and HIPAA

Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which statements are true about integrating Vault with a cloud-based encryption key to auto-unseal?

✔ A. If there are multiple Vault environments, auto-unseal can significantly reduce the operational cost

✔ B. Some cloud-based keys are backed by FIPS 140-2 HSM which is desirable for protecting the master key

C. It's cheaper solution than shared keys!

✔ D. No more shared keys, so operators can go on a vacation at the same time and don't have to worry about entering their unseal key!

Lab 3: Auto-Unseal

30 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Configure Auto-unseal Key Provider
  - Task 2: Configure Auto-unseal
  - Task 3: Audit the incoming request

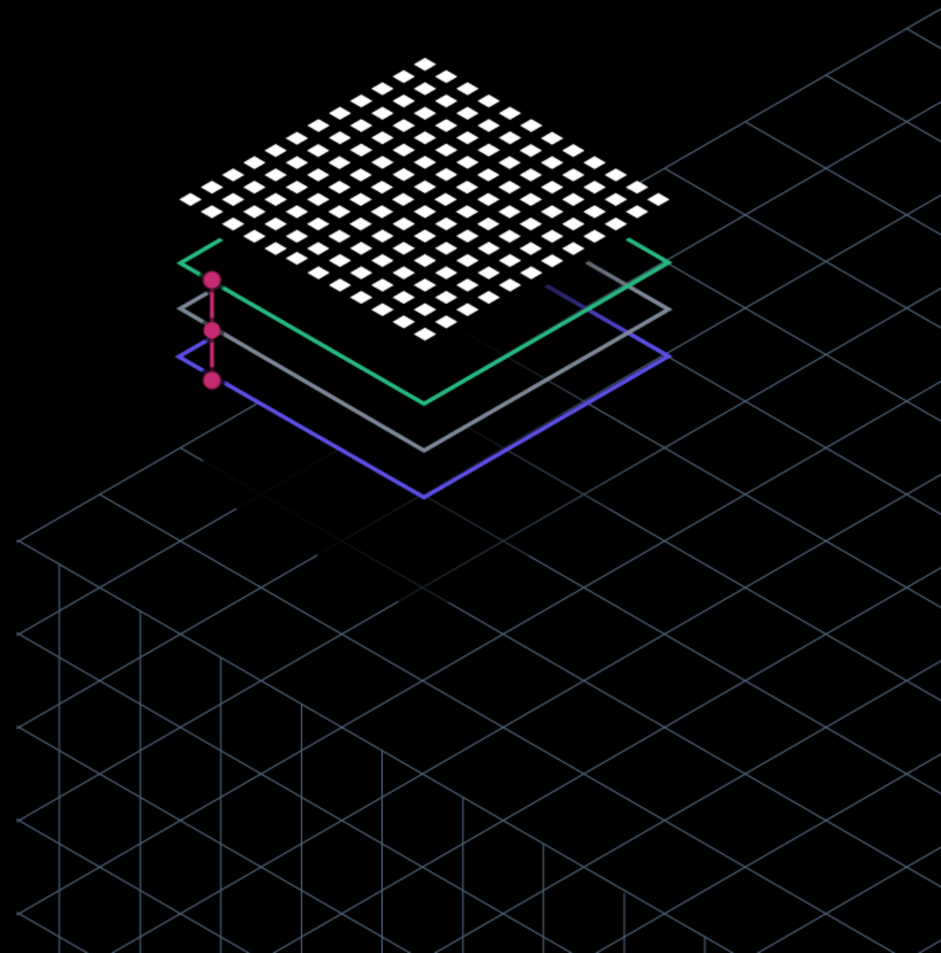
Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp

Vault Cluster Deployment



Agenda

- Vault Cluster Reference Architecture
  - Vault HA Cluster with Consul
  - Vault HA Cluster with Integrated Storage
  - Consul vs. Integrated Storage
  - Mechanics of Vault HA
    - Performance Standby Nodes

Reference Architecture

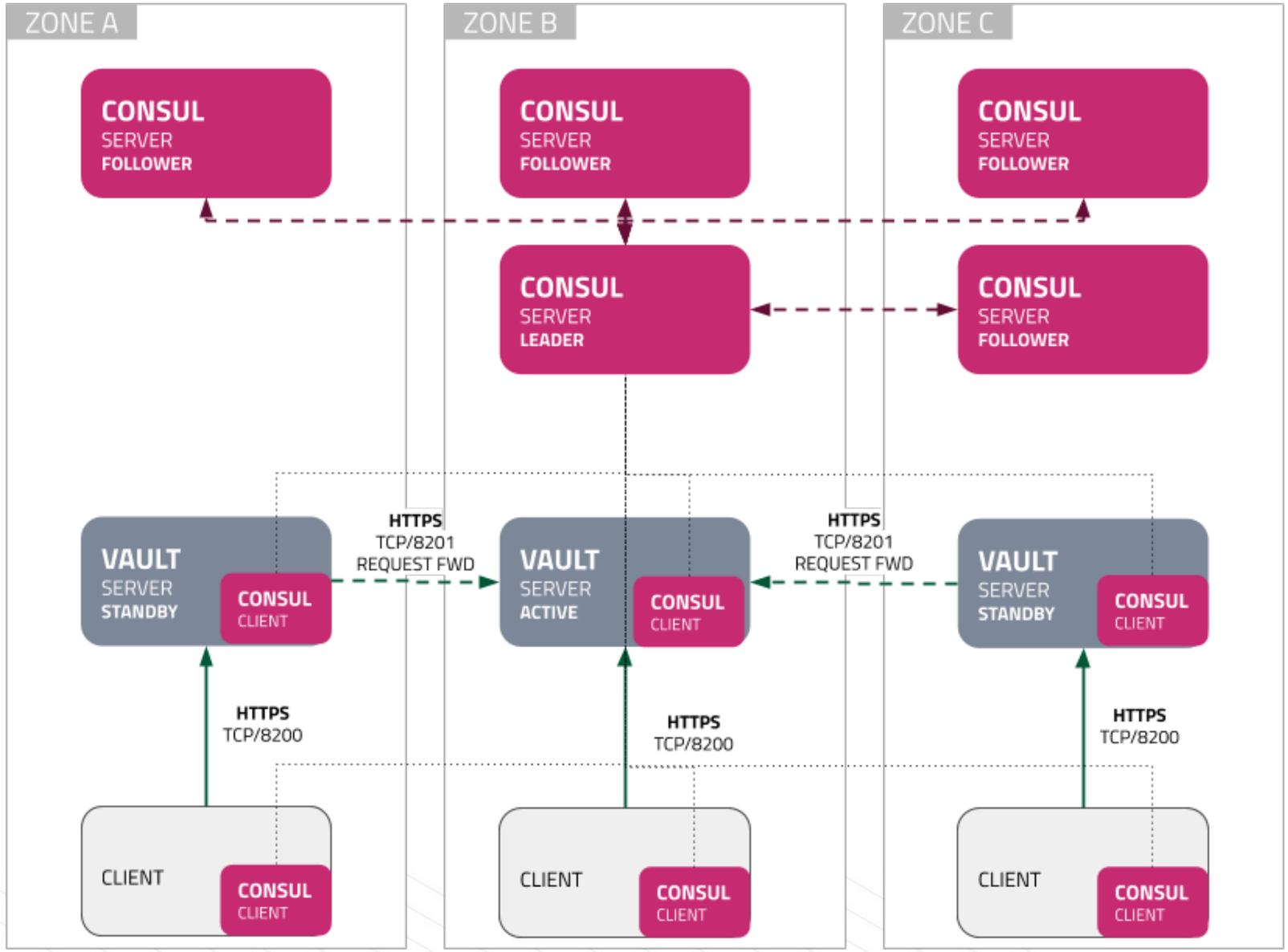
Production Cluster in a Datacenter

- Cluster design considerations:
  - Failure tolerance
    - **Three or more Vault server nodes** in a cluster for **High Availability** (HA)
    - Spread the Vault server nodes into separate fault domains (e.g. availability zones, hosts, racks, blades, etc.)
    - Vault should be the only main process running on a machine

Vault HA Cluster with Consul

External Storage

Reference Architecture with Consul



System Requirements (1 of 2)

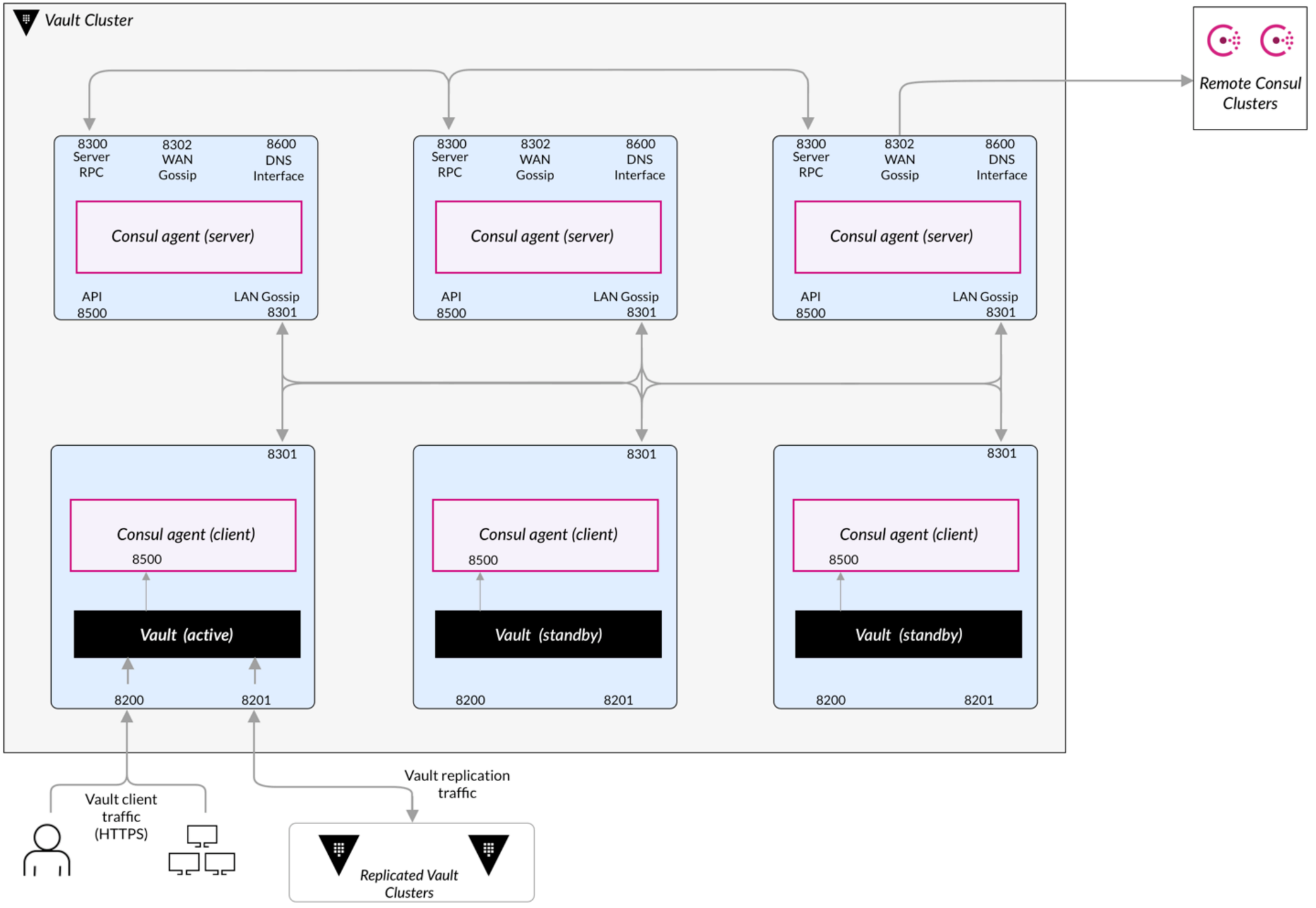
- Vault servers:

Size	CPU	Memory	Disk	Typical Instance Types
Small	2 core	4-8 GB RAM	25 GB	AWS: m5.large Azure: Standard_D2_v3 GCE: n1-standard-4, n1-standard-8
Large	4-8 core	16-32 GB RAM	50 GB	AWS: m5.xlarge, m5.2xlarge Azure: Standard_D4_v3, Standard_D8_v3 GCE: n1-standard-16, n1-standard-32

System Requirements (2 of 2)

- Consul servers:

Size	CPU	Memory	Disk	Typical Instance Types
Small	2 core	8-16 GB RAM	50 GB	AWS: m5.large, m5.xlarge Azure: Standard_D2_v3, Standard_D4_v3 GCE: n1-standard-8, n1-standard-16
Large	4-8 core	32-64+ GB RAM	100 GB	AWS: m5.2xlarge, m5.4xlarge Azure: Standard_D4_v3, Standard_D5_v3 GCE: n1-standard-32, n1-standard-64


Vault Cluster
Remote Consul Clusters
8300 Server RPC
8302 WAN Gossip
8600 DNS Interface
Consul agent (server)
API 8500
LAN Gossip 8301
8301
Consul agent (client)
8500
Vault (active)
Vault (standby)
8200
8201
Vault client traffic (HTTPS)
Vault replication traffic
Replicated Vault Clusters

Vault Server Configuration Parameters

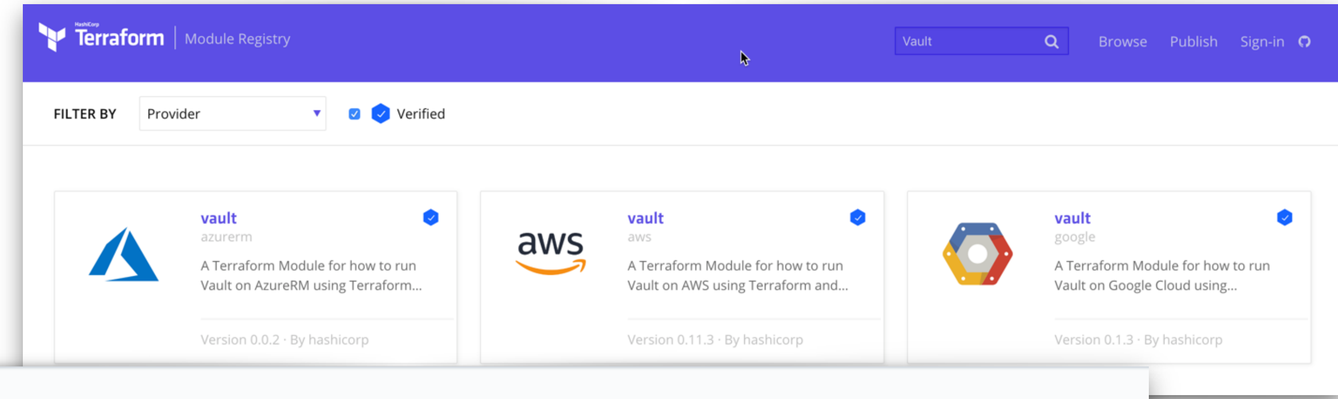
- Listener parameters:
  - **`address`**: For listening
  - **`cluster_address`**: Server-to-server requests within the cluster
- HA parameters:
  - **`api_addr`**: To advertise other Vault servers in cluster for client redirection
  - **`cluster_addr`**: To advertise other Vault servers in cluster for request forwarding

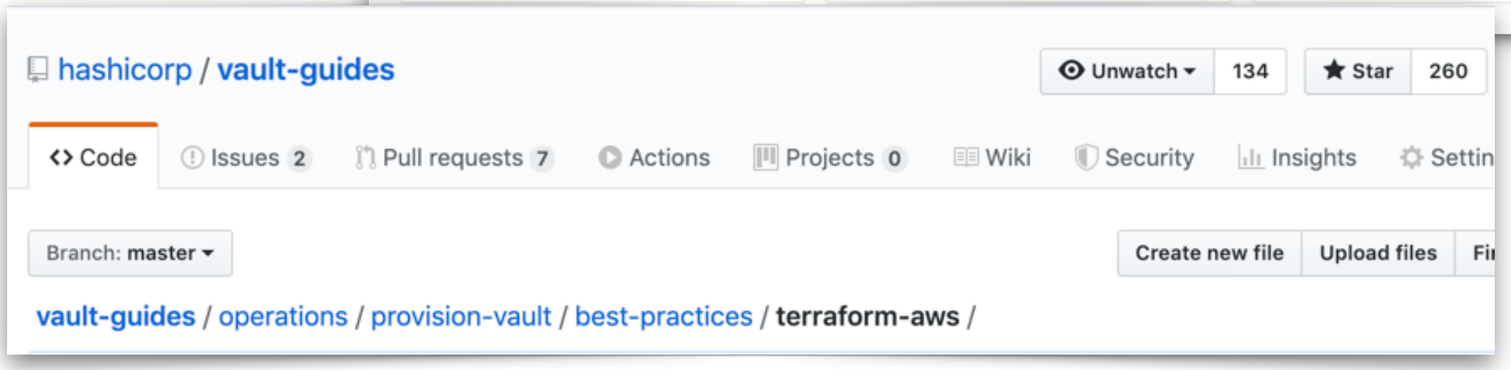
config.hcl

```
listener "tcp" {
  address          = "0.0.0.0:8200"
  cluster_address  = "0.0.0.0:8201"
  tls_cert_file = "/path/to/fullchain.pem"
  tls_key_file  = "/path/to/privkey.pem"
}

storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault/"
}
ui = true
api_addr = "https://13.57.14.206:8200"
cluster_addr = "https://10.1.42.201:8201"
```


Provision Cloud Resources

- Provision your cloud resources
  - Leverage a tool such as **Terraform** to provision
    - VMs for Vault
    - VMs for Consul
    - Load Balancer
    - Networks
    - Security Groups
    - etc.





  - Download and install **Vault** binary onto each Vault node
  - Download and install **Consul** binary onto each Consul node

Vault HA Cluster with Integrated Storage

Embedded Storage

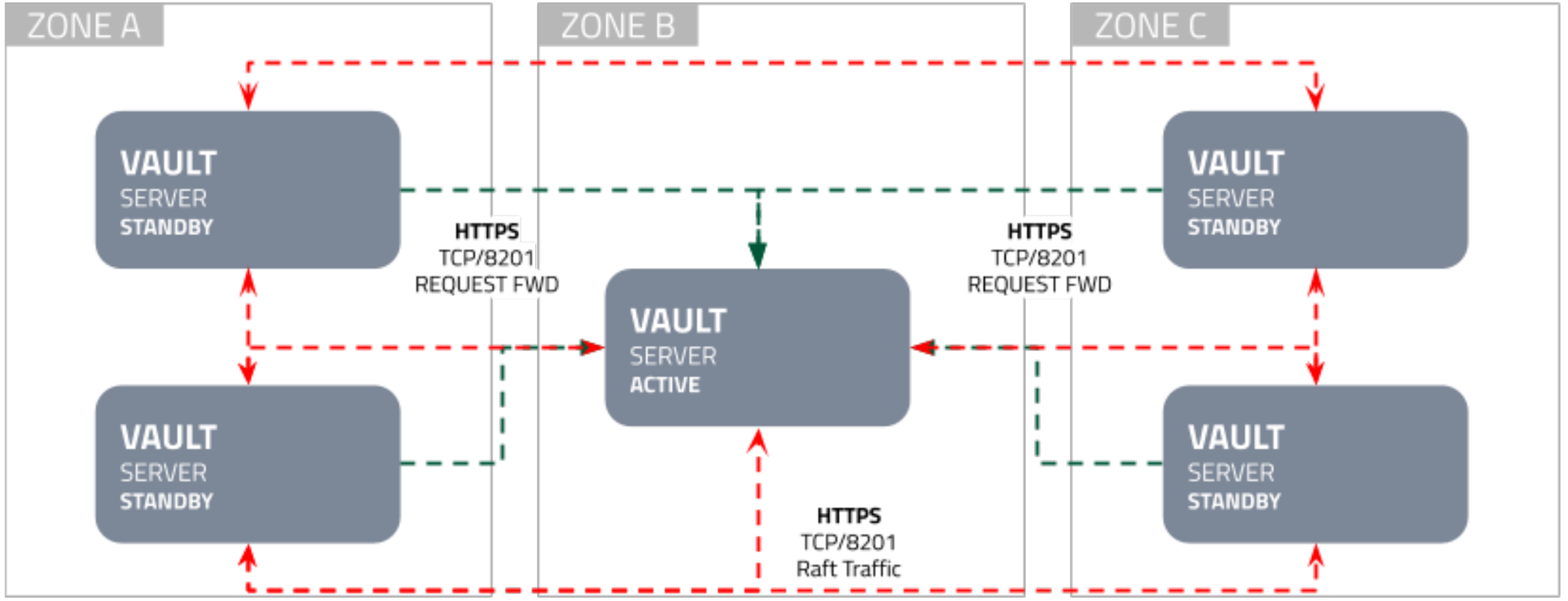
Challenge with External Storage

- It often requires admin-level knowledge of the storage backend that your Vault cluster is using
  - When Vault has an outage, the problem may have been caused by the storage backend, so you have **two systems** to investigate
  - Not to forget the **network hop** of connecting from Vault to the storage

Vault Integrated Storage

- **Integrated Storage** exists as a purely Vault ***internal*** storage option which leverages the **Raft** consensus protocol
  - All nodes in a Vault cluster will have a replicated copy of Vault's data
  - Eliminates the network hop of connecting from Vault to the external storage provider
  - When Vault has an outage, you only need to troubleshoot Vault
  - Requires **Vault 1.4** or later

Reference Architecture with Integrated Storage



The recommended number of Vault instances is **5** in a cluster.

System Requirements

- If many secrets are being generated, the data will need to be flushed to the disk often. The infrastructure should have a **high-performance hard disk system** when using the integrated storage.

Size	CPU	Memory	Disk	Typical Instance Types
Small	2 core	8-16 GB RAM	100 GB	AWS: m5.large, m5.xlarge Azure: Standard_D2_v3, Standard_D4_v3 GCE: n2-standard-2, n2-standard-4
Large	4-8 core	32-64 GB RAM	200 GB	AWS: m5.2xlarge, m5.4xlarge Azure: Standard_D8_v3, Standard_D16_v3 GCE: n2-standard-8, n2-standard-16

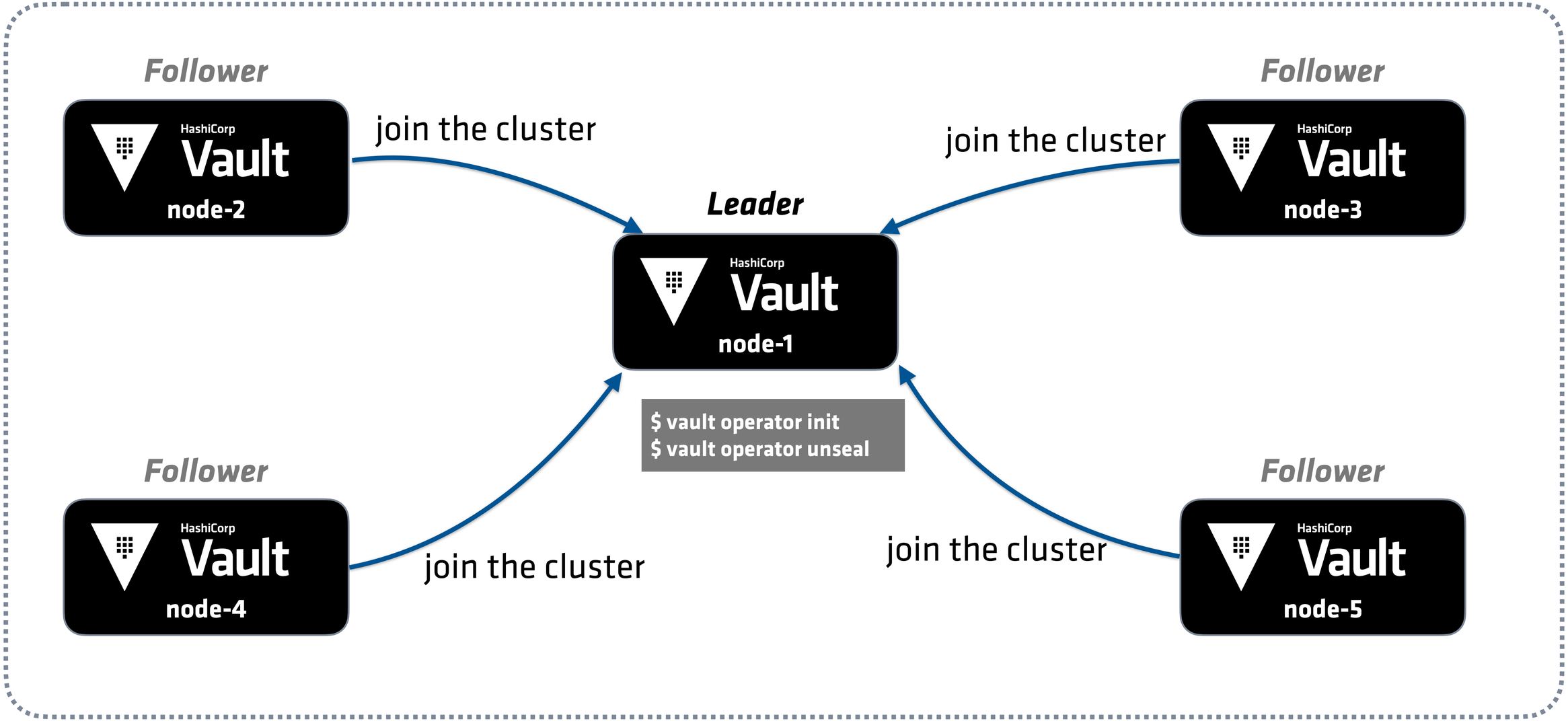
Vault Server Configuration

- Set the storage stanza to use "**`raft`**"
  - **`path`**: The filesystem path where all the Vault data gets stored
  - **`node_id`**: The identifier for the node in the cluster

config.hcl

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

 listener "tcp" {
   address = "0.0.0.0:8200"
   cluster_address = "0.0.0.0:8201"
   tls_disable = true
 }
 ....
 api_addr = "http://192.0.2.53:8200"
 cluster_addr = "http://10.0.0.2:8201"
```

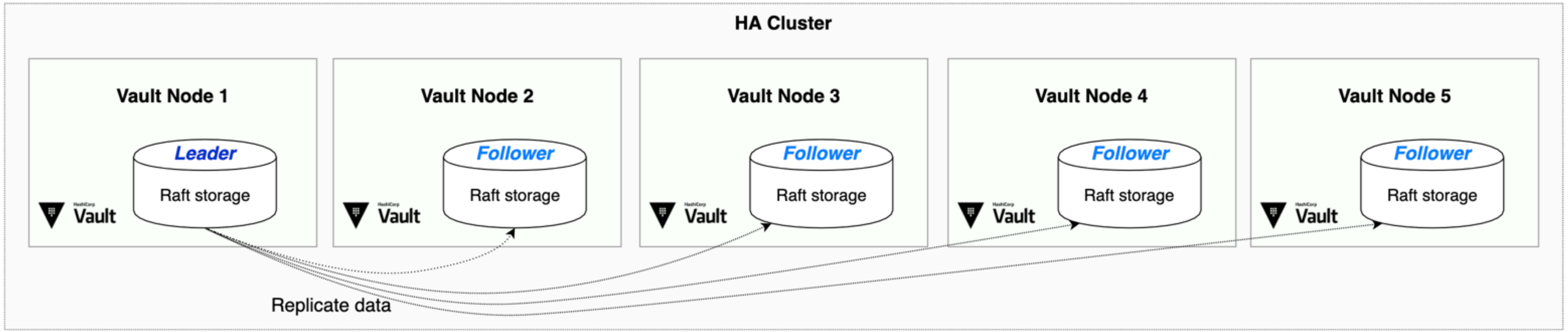

Cluster Configuration Workflow

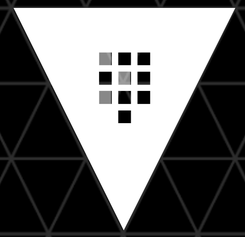


Adding Nodes

- Example: A new standby node to join the cluster

```
$ vault operator raft join https://active.vault-server.address:8200
```

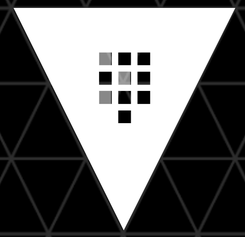




List the cluster members

Terminal

```
$ vault operator raft list-peers

Node       Address                State       Voter
----       -------                -----       -----
vault_1    10.0.101.22:8201       leader      true
vault_2    10.0.101.23:8201       follower    true
vault_3    10.0.101.24:8201       follower    true
vault_4    10.0.101.25:8201       follower    true
vault_5    10.0.101.26:8201       follower    true
```



Remove a node from the cluster

Terminal

Name of the node to be removed

```
$ vault operator raft remove-peers vault_5

Peer removed successfully!


$ vault operator raft list-peers

Node       Address              State       Voter
----       -------              -----       -----
vault_1    10.0.101.22:8201     leader      true
vault_2    10.0.101.23:8201     follower    true
vault_3    10.0.101.24:8201     follower    true
vault_4    10.0.101.25:8201     follower    true
```

Data Backup

- Back up the data by taking a snapshot

```
$ vault operator raft snapshot save <file-name.snap>
```

- Restore the data from a snapshot

```
$ vault operator raft snapshot restore <file-name.snap>
```

Managing Integrated Storage using CLI

- Usage: **vault operator raft <subcommand> [options] [args]**

Subcommand	Description
list-peers	Returns the raft cluster member information
join	Joins a node to the cluster
remove-peer	Removes a node from the cluster
snapshot	Restores and saves snapshots from the cluster

Consul vs. Integrated Storage

Consul vs. Integrated Storage

Consul

Pros:

- All data is in memory

Cons:

- Operationally more complex and harder to debug
- Extra network hop between Vault and Consul nodes
- Incremental snapshots must write all data
- **Memory-bound** (vulnerable to out of memory error)

Integrated Storage

Pros:

- Operationally simpler
- One less network hop
- Incremental snapshots are fast since the data is already on disk

Cons:

- Data is on disk (1 extra disk write compare to Consul)
- **Disk-bound**; therefore, the host machine should not allow burstable IOPS

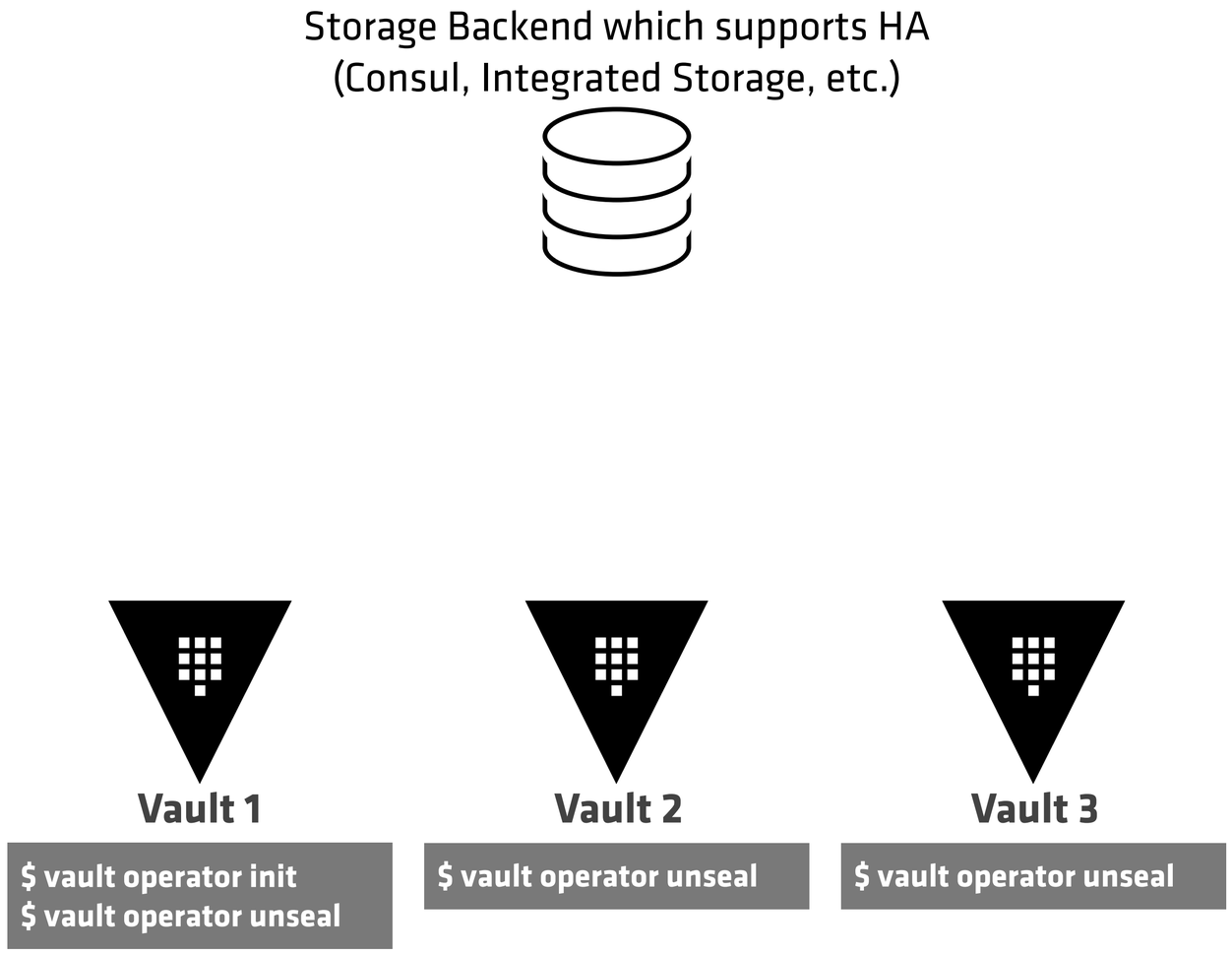
Comparison Summary

Consideration	Consul as storage backend	Vault Integrated Storage
System requirement	Memory optimized machine	Storage optimized high IOPS machine
Data snapshot	Frequent snapshots	Normal data backup strategy
Snapshot automation	Snapshot agent (Consul Enterprise only)	No out-of-the-box automation
Data inspection	Use **`consul kv`** command or UI	Limited API endpoints
Autopilot	Supported	Currently not available

Mechanics of Vault HA

Vault HA: How It Works (1 of 4)

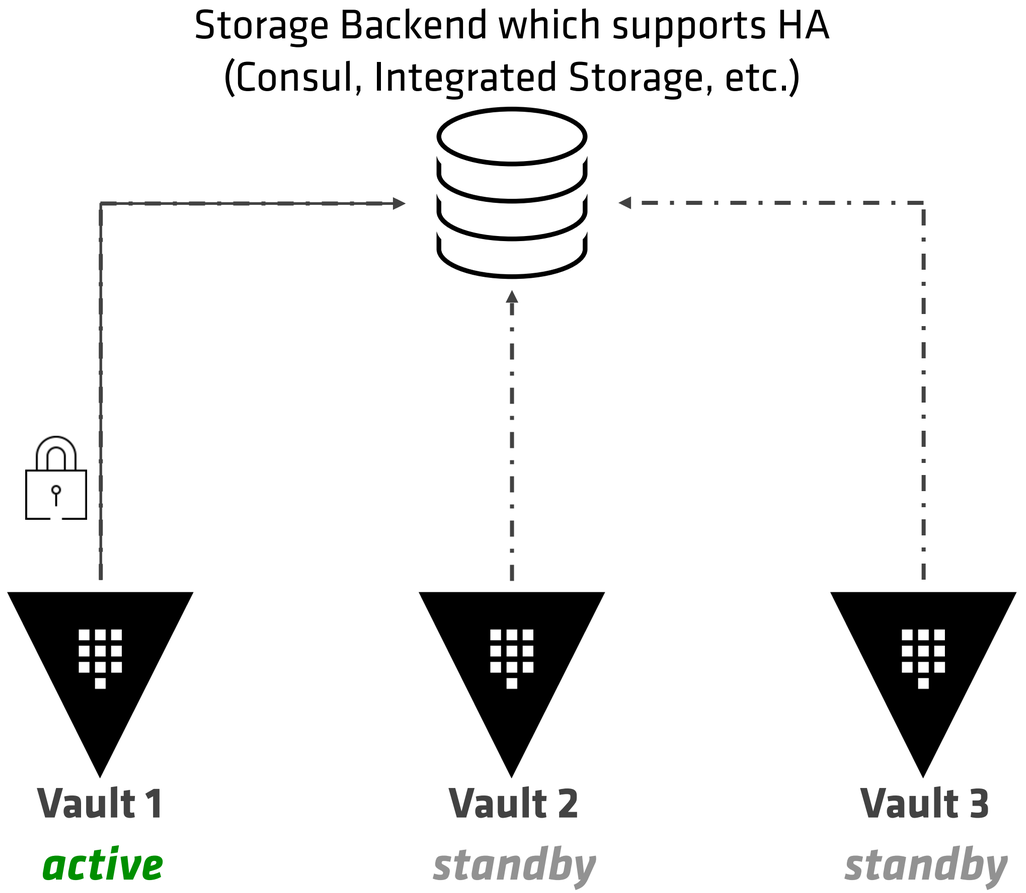
1. Deploy a Vault server with an HA storage backend configured
2. Run ***vault operator init*** to generate unseal keys and token on the first Vault
3. **Unseal** the Vault
4. Unseal the second and third Vault servers with the same unseal keys



Vault HA: How It Works (2 of 4)

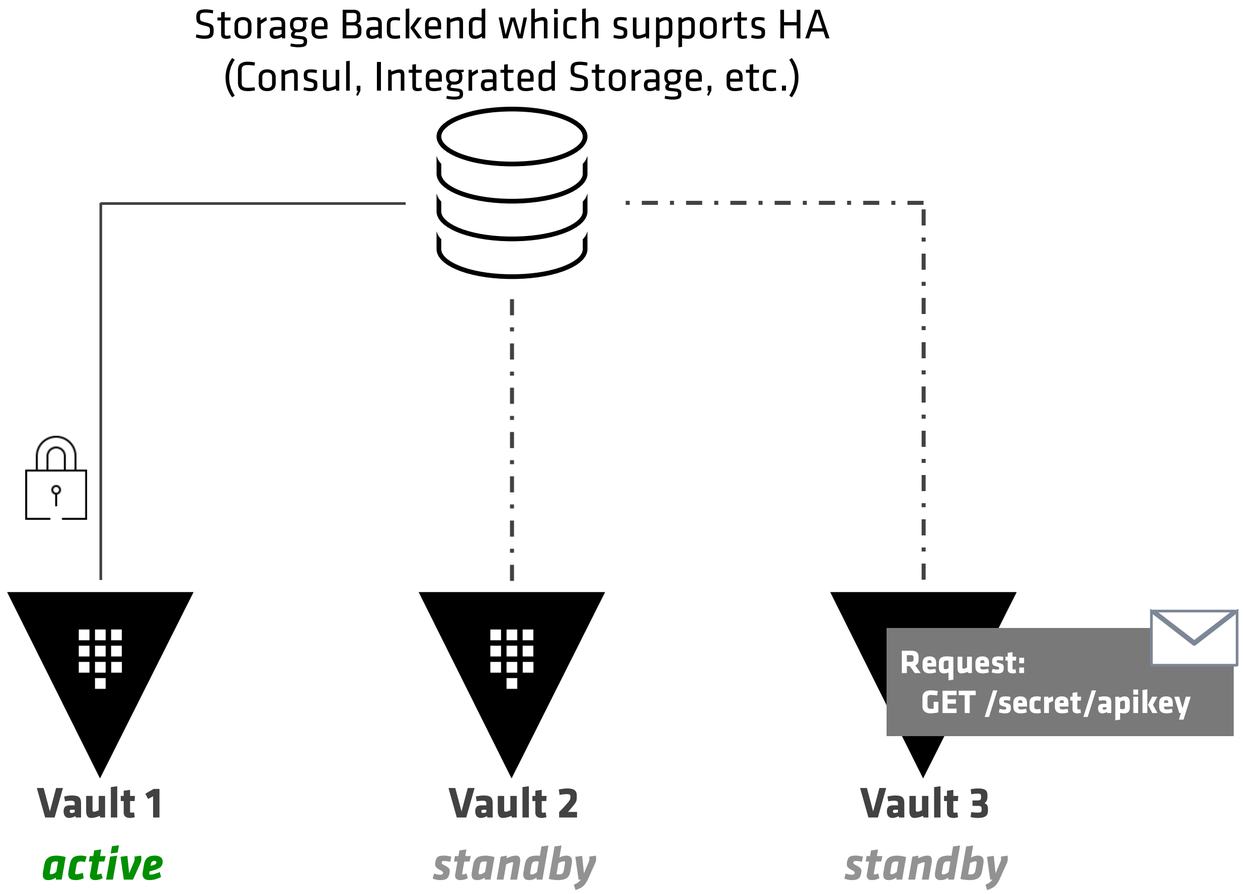
- At startup, they all try to acquire a "**lock**" on a shared key in the storage backend
- One of them succeeds, and becomes ***active*** while others go into a ***standby*** mode

Only the unsealed Vault servers act as standby



Vault HA: How It Works (3 of 4)

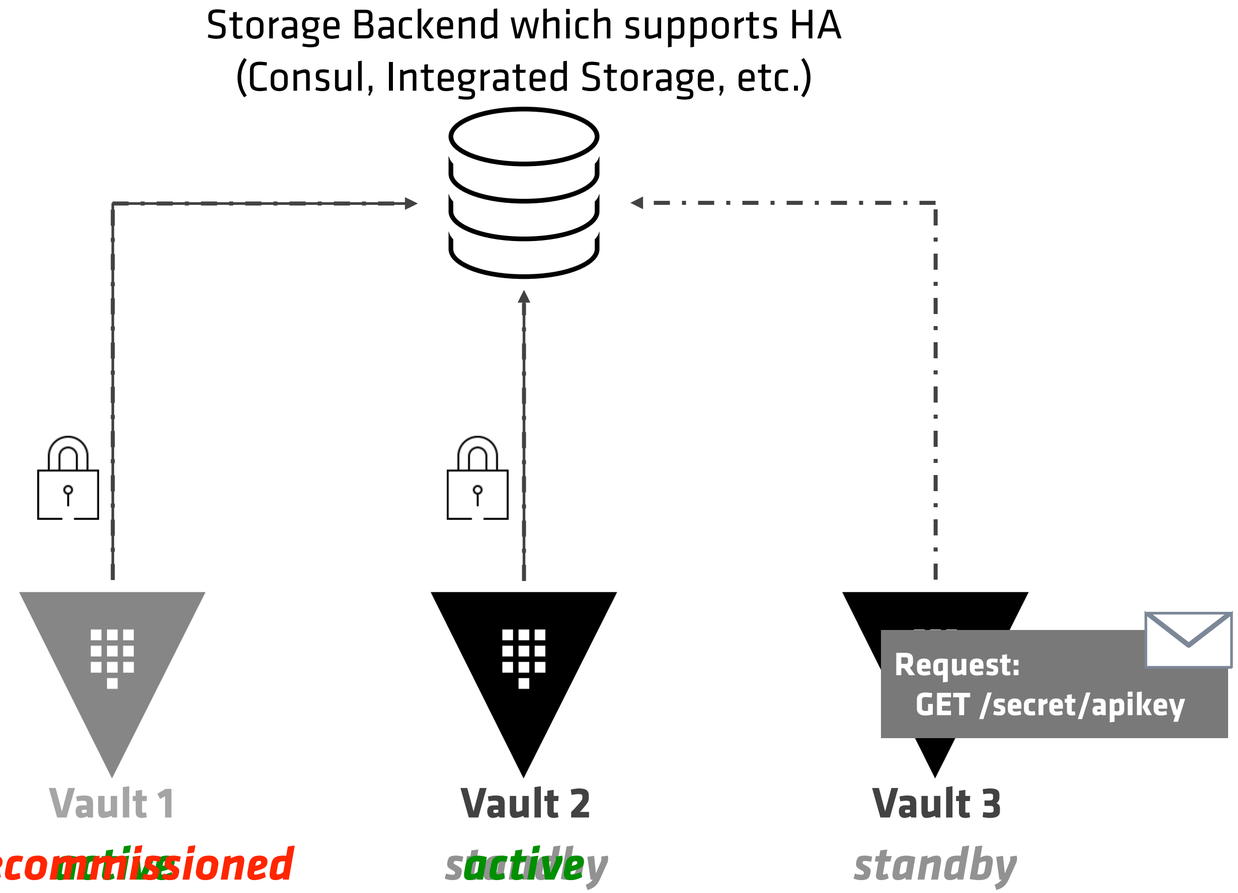
- If the *standby* nodes receive a request...
- One of the following happens based on the current configuration:
  - Forward the request based on the **`cluster_addr`**
  - Redirect the client based on the **`api_addr`**



HA does not enable increased scalability. In general, the bottleneck of Vault is the data store.

Vault HA: How It Works (4 of 4)

- When the ***active*** node goes down...
  - It loses it's lock either gracefully or after the minimal TTL
  - Meanwhile one of the standby nodes becomes ***active***
  - The rest of the behavior is the same
  - Except one less server...



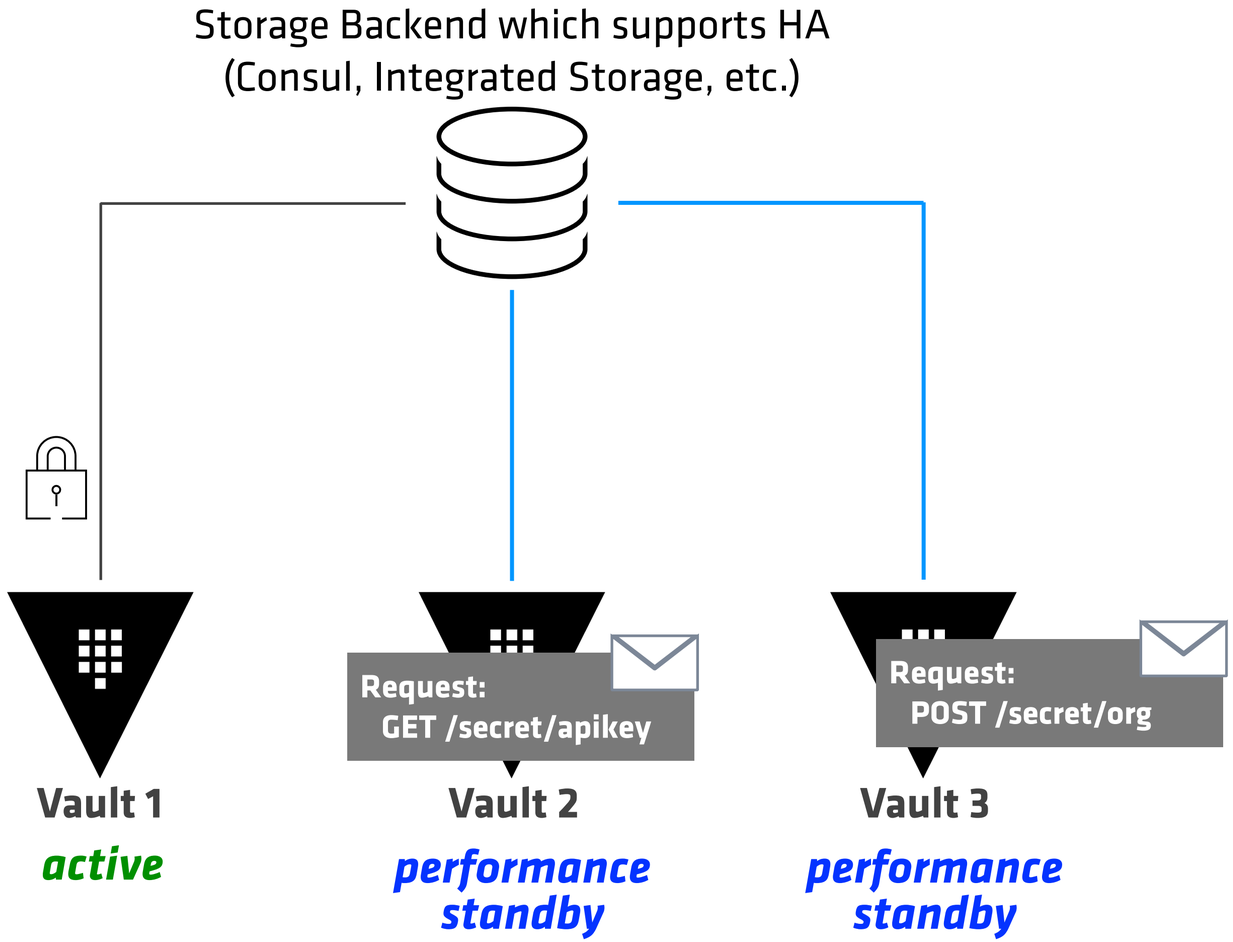
Performance Standby

Vault Enterprise Feature

Vault HA: Performance Standby Nodes (1 of 2)

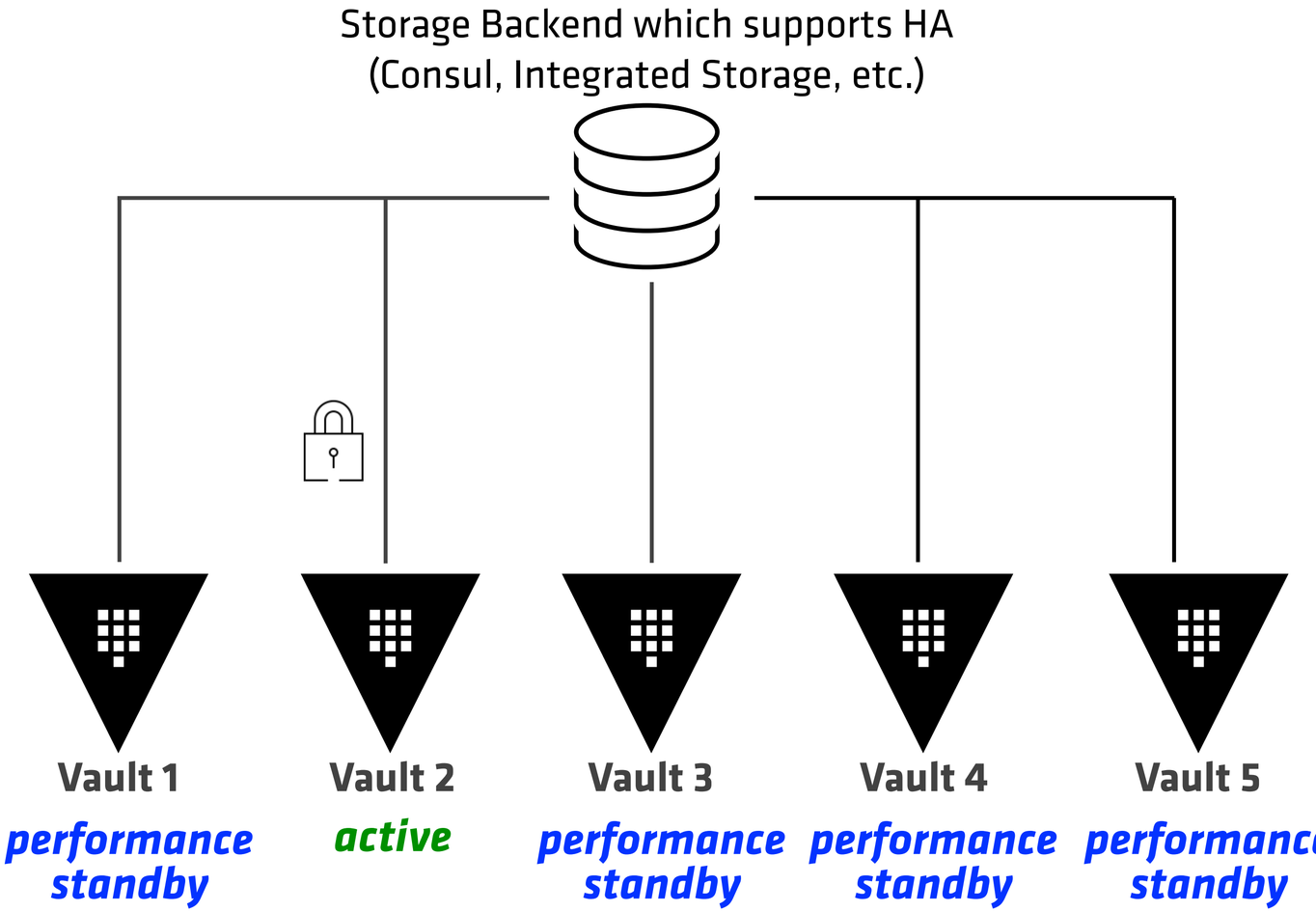
- If you are running **Vault Enterprise**, the standby nodes can handle **read-only** requests. This is called *performance standby nodes*

This is particularly useful for processing high volume ***Encryption as a Service*** (transit secrets engine) requests



Vault HA: Performance Standby Nodes (2 of 2)

- **Example:**
  - 5 servers in a cluster
  - Cluster scales for **read** operations as you add more performance standby nodes



Knowledge Check & Lab Overview

Knowledge Check Questions

fault tolerance

1. True or False: To increase the ~~***scalability***~~, a Vault cluster should have 3 or more Vault servers.

2. In a HA cluster, what happens when a standby node receives a request from a client?

   Either ***forward*** the request or ***redirect*** the client to the *active* node

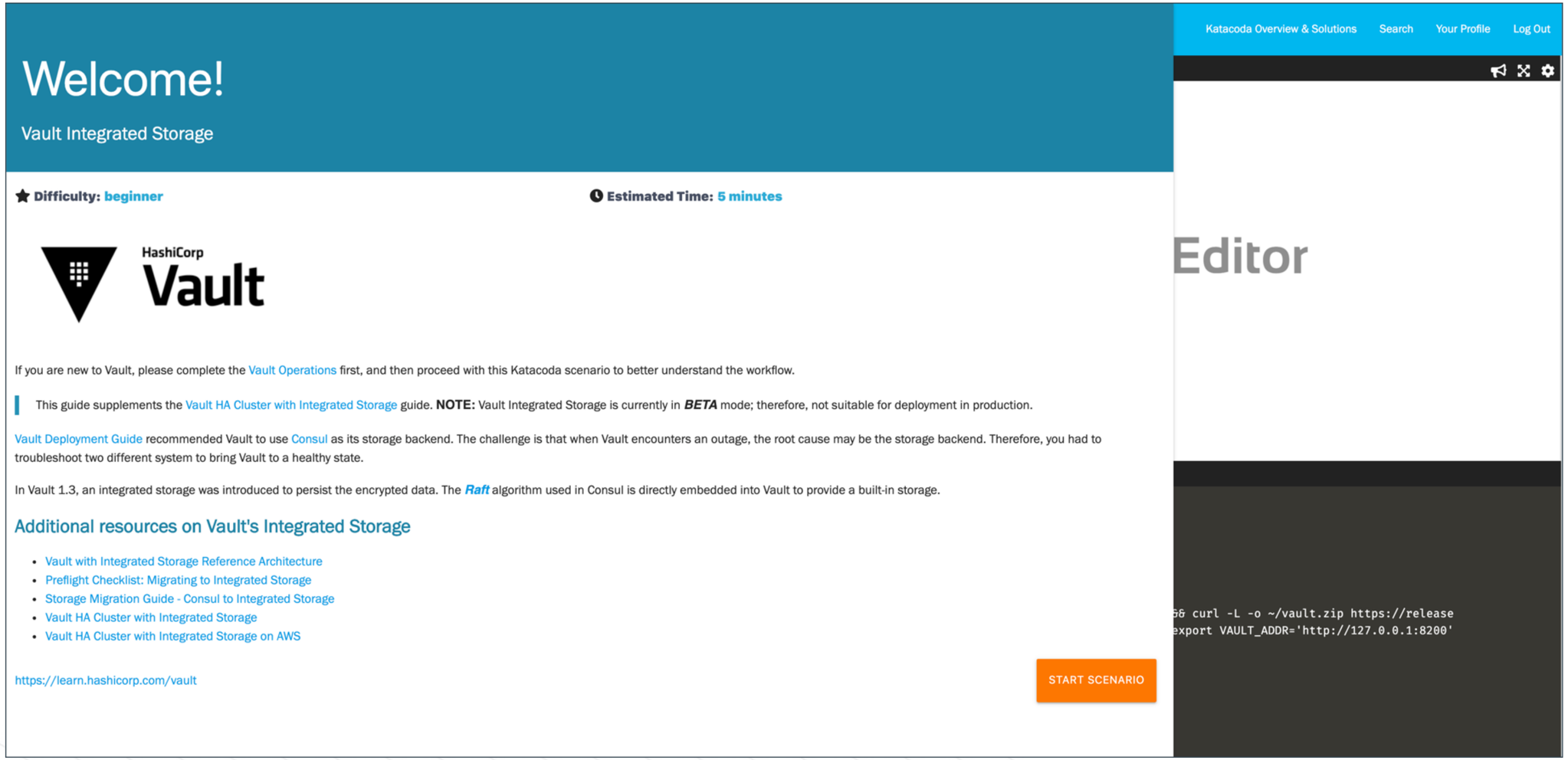
3. Can you explain the difference between the ***standby*** node and ***performance standby*** node?

   Performance standby can handle ***read-only*** requests and it's a part of VE

4. True or False: You should set the HA parameters (**api_addr** and **cluster_addr**) explicitly in the Vault configuration especially when you have a load balancer.

Optional Lab: Katacoda Interactive Tutorial

- URL: https://www.katacoda.com/hashicorp/scenarios/vault-raft

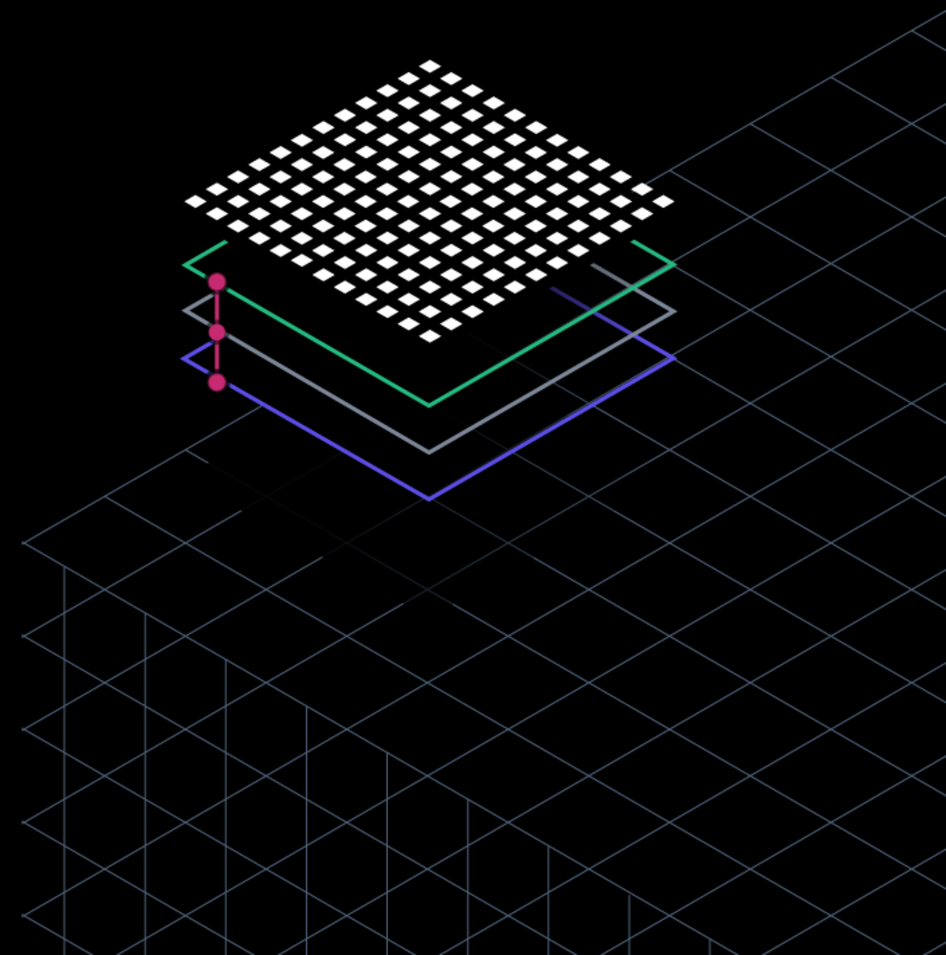


Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp



Vault Operations

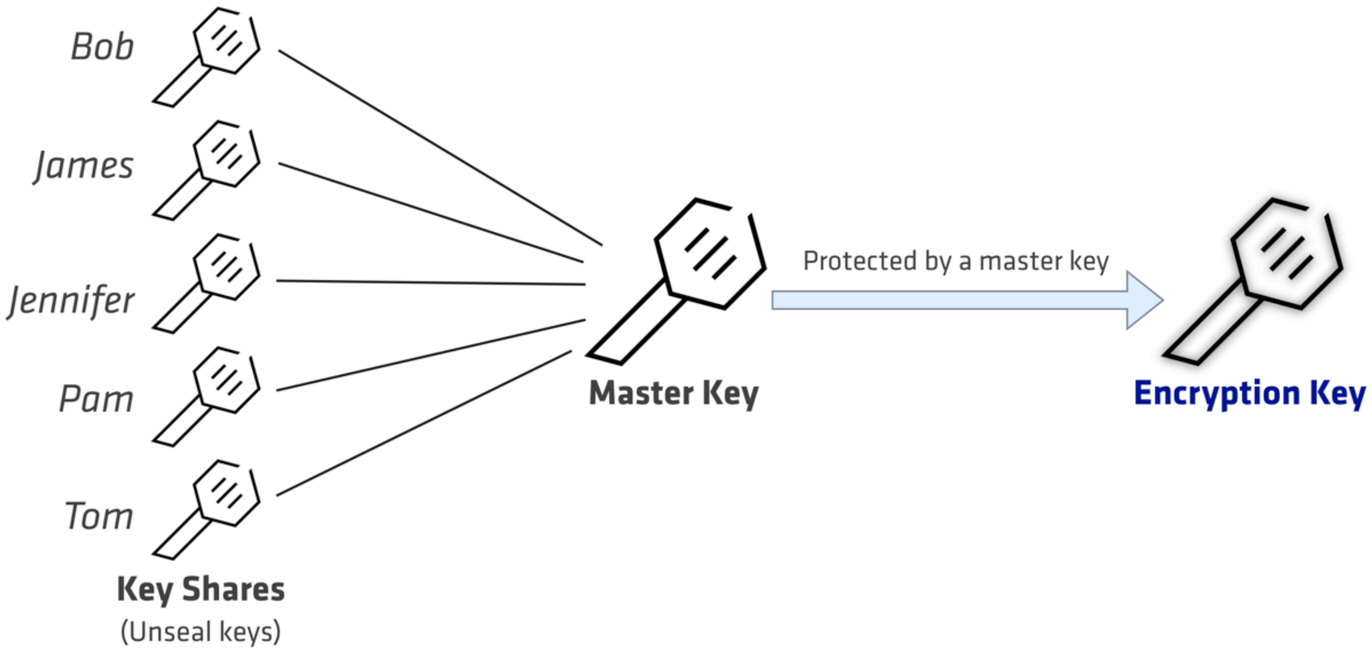
Agenda

- Vault Key Rotation: Rekey and Rotate
  - What is Rekey?
  - Shamir's keys vs. Recovery keys
  - What is Key Rotation?
- Generating a Root Token

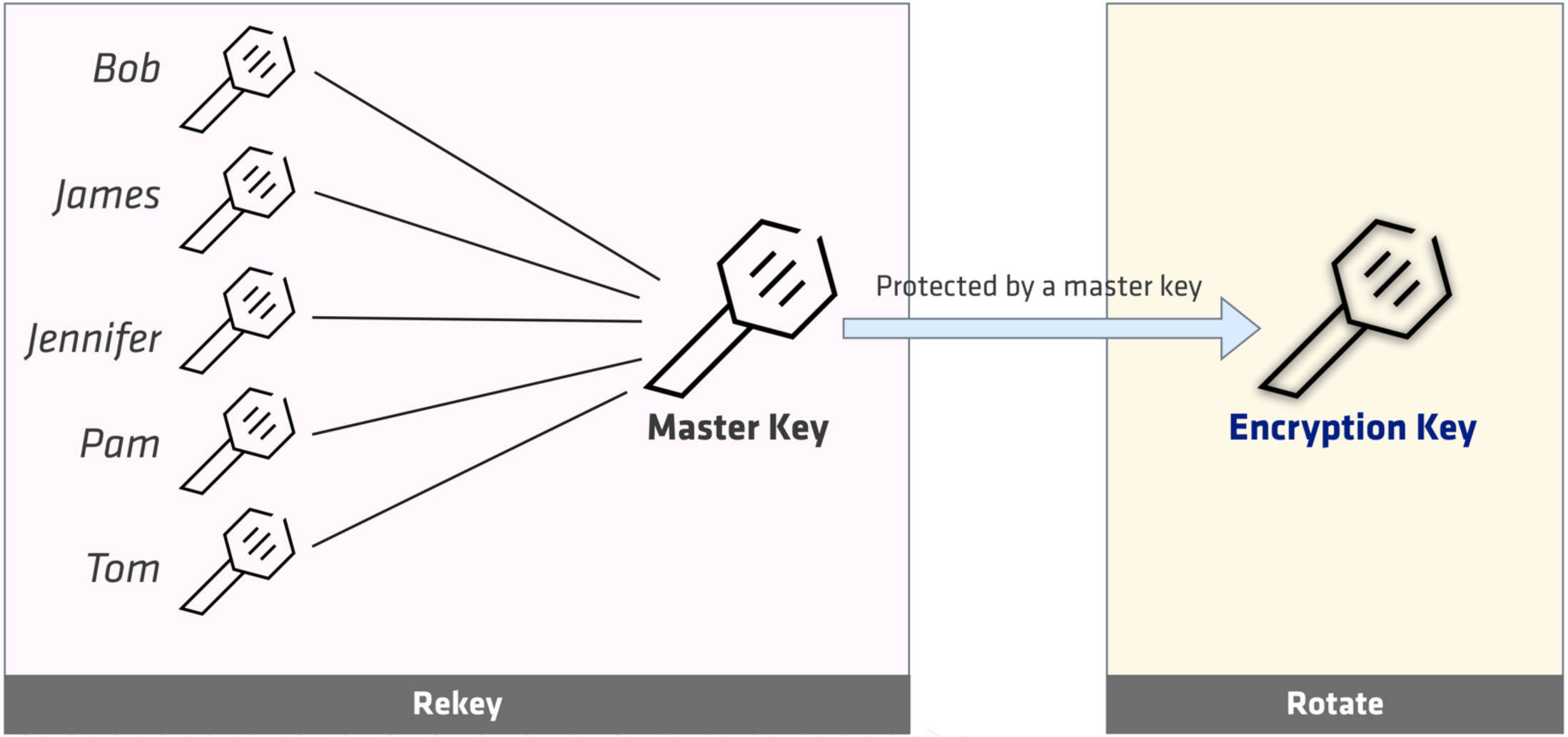
Vault Key Rotation

Consider the Following Scenarios

- Some situations may require rekey and/or key rotation:
- For example:
  - One of the unseal key holder leaves the organization
  - Your organization requires to rekey after a period of time as a security measurement



Key Rotation: Rekey and Rotate



Question

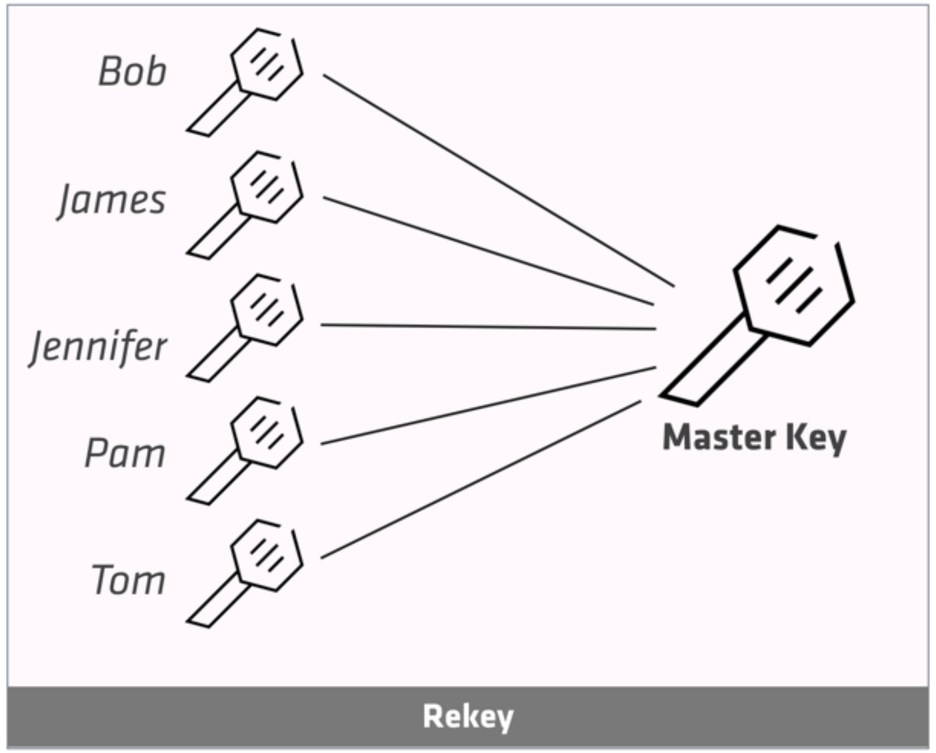
What about those Vault servers with auto-unseal enabled?

A: The Shamir's keys are NOT used to unseal the Vault server; however, they still exists --> *Recovery Keys*

The cloud provider's key is used to protect the **master key**. You still need Shamir's keys (**recovery keys**) for other operations: regenerate a root token, or unseal a Vault which was sealed via CLI command: **`vault server seal`**

Rekey (1 of 2)

- Allows changing the number of shares and threshold of shares
- Requires a threshold of unseal keys (like an unseal operation)
- Provides a nonce value which must be given to all unseal keys (similar to `generate-root`)



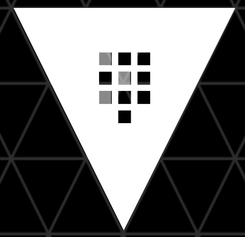
Rekey (2 of 2)

- To perform the task, use the **`rekey`** command

  Usage: **`vault operator rekey [options] [key]`**

Common Command Options:

Option	Description
-init	Initialize the rekeying operation
-key-shares=<int>	Number of key shares to split the generated master key into
-key-threshold=<int>	Number of key shares required to reconstruct the master key
-nonce	Pass the nonce value provided at initialization
-pgp-key=<keybase:user1, ...>	Public PGP keys to encrypt the generated unseal keys
-status	Print the status of the current attempt



Initialize the Rekey Process

Step 1

Terminal

```
# With auto-unsealed, set -target flag
$ vault operator rekey -init -target="recovery" ...
$ vault operator rekey -init -key-shares=5 -key-threshold=3

WARNING! If you lose the keys after they are returned, there is no
recovery. Consider canceling this operation and re-initializing with
the   -pgp-keys flag to protect the returned unseal keys along with
-backup to allow recovery of the encrypted keys in case of
emergency. You can delete the stored keys later using the -delete
flag.

Key                 Value
---                 -----
Nonce               ad1c6c50-5296-d61d-5c2d-24f017b0aa97
Started             true
Rekey Progress      0/3
New Shares          5
New Threshold       3
```

Distribute this nonce to the unseal key holders since they will need this value to rekey

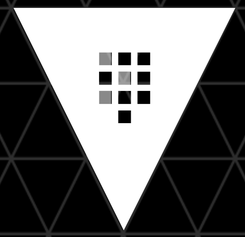
Rekey (1 of 3)

Step 2

Terminal

```
# Pass the nonce generated from initialization (-init)
$ vault operator rekey -nonce=ad1c6c50-5296-d61d-5c...

Rekey operation nonce: ad1c6c50-5296-d61d-5c...
Unseal Key (will be hidden):

Key                  Value
---                  -----
Nonce                ad1c6c50-5296-d61d-5c2d-24f017b0aa97
Started              true
Rekey Progress       1/3
New Shares           5
New Threshold        3
```

Each unseal key holder must run this command providing the nonce and their unseal key

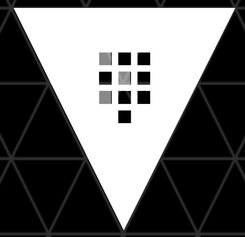


Rekey (2 of 3)

Step 2

Terminal

```
# Another unseal key-holder performs the same
$ vault operator rekey -nonce=ad1c6c50-5296-d61d-5c...

Rekey operation nonce: ad1c6c50-5296-d61d-5c...
Unseal Key (will be hidden):

Key                   Value
---                   -----
Nonce                 ad1c6c50-5296-d61d-5c2d-24f017b0aa97
Started               true
Rekey Progress        2/3
New Shares            5
New Threshold         3
```



Rekey (3 of 3)

Step 2

Terminal

```
# The last unseal key-holder performs the rekey process
$ vault operator rekey -nonce=ad1c6c50-5296-d61d-5c...

Rekey operation nonce: ad1c6c50-5296-d61d-5c2d-24f017b0aa97
Unseal Key (will be hidden):

Key 1: RUhMZW6S17E3s2Mj/pc+s+wR/onlCM7xK/zUvCVD468y
Key 2: Nh7m9IvpY5DQf/OPV8pSHg/EqwcaY0lypc99m1Lby8Rb
Key 3: i8ku69SlIxpp4+cmIzuDzI+2CLMplpoq3mhVoblNV1S0
Key 4: f96mVNGS2HwsQeiO6qr6+lUWcr+FuycWUtidQ4lc/z4D
Key 5: Lt//U4VF12tWiiUzbe/GRy40aW9Gw8OpL8cd4UMyuohl

Operation nonce: ad1c6c50-5296-d61d-5c2d-24f017b0aa97
```

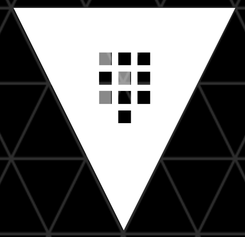
Question

Is the rekey operation an "online" (no downtime) operation?

A: Yes. Vault continues to service requests while the rekey operation is taking place.

Rotate

- Changes the **encryption key** that protects data in the storage backend
- The **encryption key** is ***never available*** to operators and thus does NOT require a threshold of key holders like rekey
- New keys are added to a key ring so that old values can still be decrypted and are updated on-the-fly



Rotate

Terminal

```
$ vault operator rotate

Success! Rotated key

Key Term          2
Install Time      22 Dec 19 22:52 UTC
```

Yes, this simple!

Question

When to rotate the encryption key?

A: Vault used AES-GCM to encrypt keys. NIST recommends no more than 2^32 operations per key. Evaluate your usage (writes, not reads) and rotate regularly.

Note: 2^32 = 4,294,967,296

(Re)generating Root

Root Token

- In a production Vault installation, the initial root token should **only be used for initial configuration**
  - After a subset of administrators have `sudo` access, almost all operations can be performed
- Best practice is **NOT** persist the root tokens
- For some system-critical operations, a root token may still be required

Question

Q: What if the initial root token was lost and we really need a root token?!

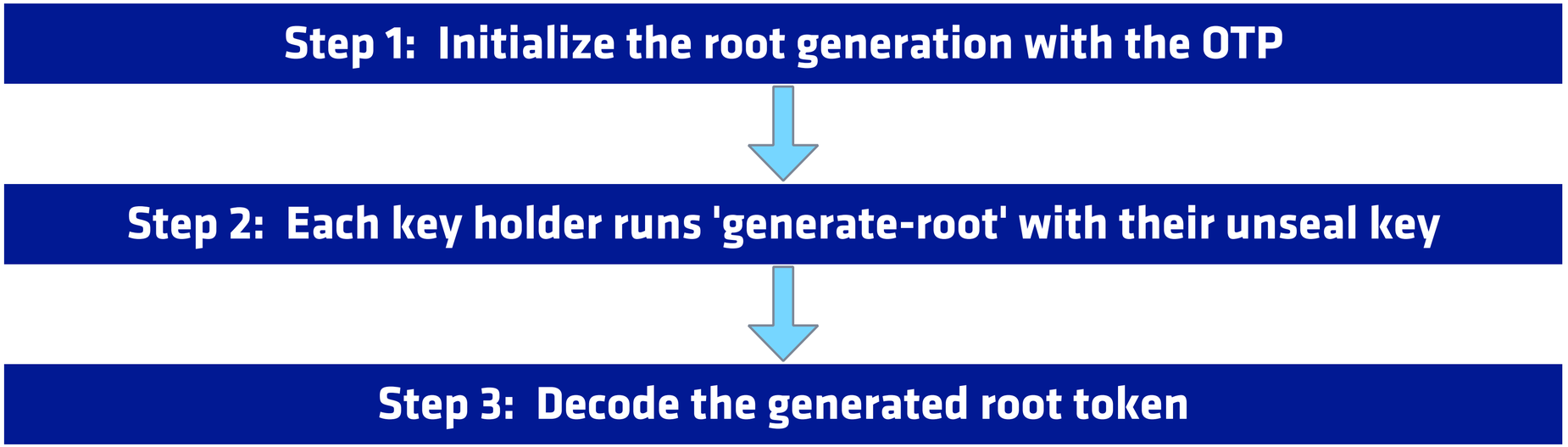
A: Regenerate a root token.

Remember, generate a root token only when absolutely necessary

Regenerate a Root Token (1 of 2)

NOTE: Vault server must be unsealed

- A quorum of **unseal key holders** can generate a new root token
  - Enforces the "*no one person has complete access to the system*"



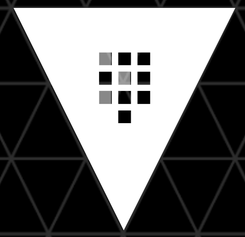
Regenerate a Root Token (2 of 2)

- To perform the task, use the **`generate-root`** command

  Usage: **`vault operator generate-root [options] [key]`**

Common Command Options:

Option	Description
-generate-otp	Generate and print a high-entropy one-time-password
-init	Start a root token generation
-decode=<string>	Decode and output the generated root token
-otp=<string>	OTP code to use with "**-decode**" or "**-init**"
-pgp-key=<keybase:user>	Path to a file on disk containing a binary or base64-encoded public PGP key
-status	Print the status of the current attempt



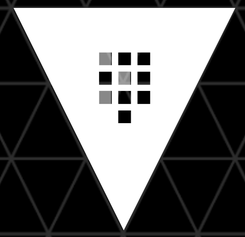
Initialize a Root Token Generation with the OTP

Step 1

Terminal

```
$ vault operator generate-root -init

A One-Time-Password has been generated for you and is shown in
the OTP field.
You will need this value to decode the resulting root token,
so keep it safe.

Nonce            5b6e3831-2a45-4695-7757-5810074d36c8
Started          true
Progress         0/1
Complete         false
OTP              E87jF6ZeJo8NjJwvytl7mvKLEr
OTP Length       26
```

One-time password (OTP) gets generated

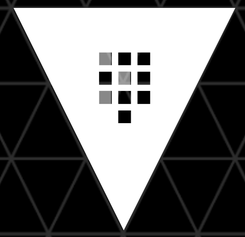


Root Generation (1 of 2)

Step 2

Terminal

```
$ vault operator generate-root
Root generation operation nonce: f8579a51-5138-c31...
Unseal Key (will be hidden):
Nonce       f8579a51-5138-c319-445d-2d3640119f87
Started     true
Progress    1/3
Complete    fals
```



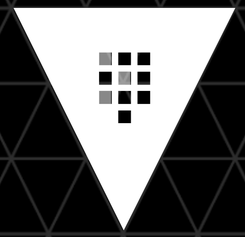
Root Generation (2 of 2)

Step 2

Terminal

```
$ vault operator generate-root

Root generation operation nonce: f8579a51-5138-c319...
Unseal Key (will be hidden):

Nonce                f8579a51-5138-c319-445d-2d3640119f87
Started              true
Progress             3/3
Complete             true
Encoded Token        G2NeKUZgXTsYYxILAC9ZFBguPw9ZXBovFAs
```

Encoded root token



Decode the Root Token

Step 3

Terminal

```
$ vault operator generate-root \
      -otp="hM9q24nNiZfnYIiNvhnGo4UFc3" \
      -decode="G2NeKUZgXTsYYxILAC9ZFBguPw9ZXBovFAs"

Root token: s.gXtT3uq9teYf0ZnFQH6hOiw8
```

Knowledge Check & Lab Overview

Knowledge Check Questions

1. True or False: If the initial root token was lost, a threshold of unseal keys are required to regenerate a new root token.

2. Explain the ***rekey*** process and ***key rotation*** process. How different?

   Rekey is the process of getting a fresh set of unseal keys, and key rotation is the process of rotating the Vault encryption key

3. True or False: Vault **encryption** key is ***never available*** to operators to begin with; therefore, key rotation does NOT require a threshold of key holders like rekey

Lab 5: Vault Operations

30 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Generate a root token
  - Task 2: Rekeying Vault
  - Task 3: Rotate the encryption key

Read the instructions carefully!

At **Step 5.2.1**, if your Vault server is currently auto-unsealed, execute the command with **-target="recovery"** flag as instructed.

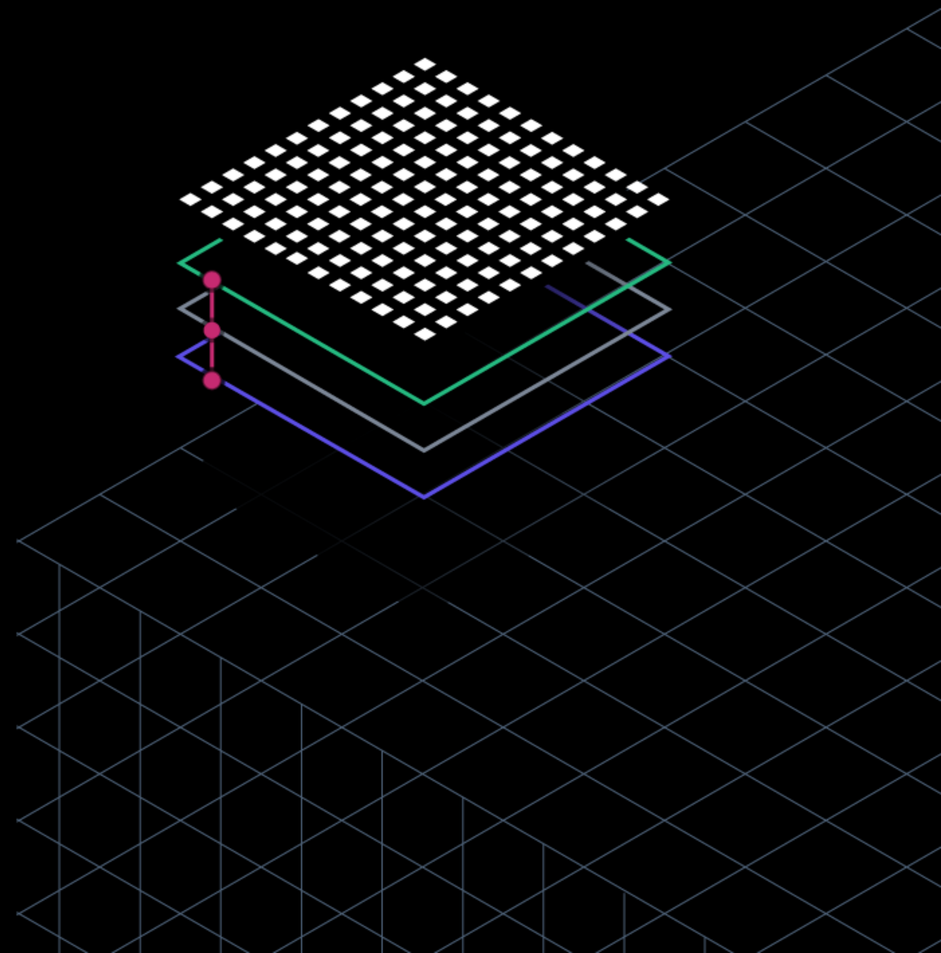
Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp

Vault Policies



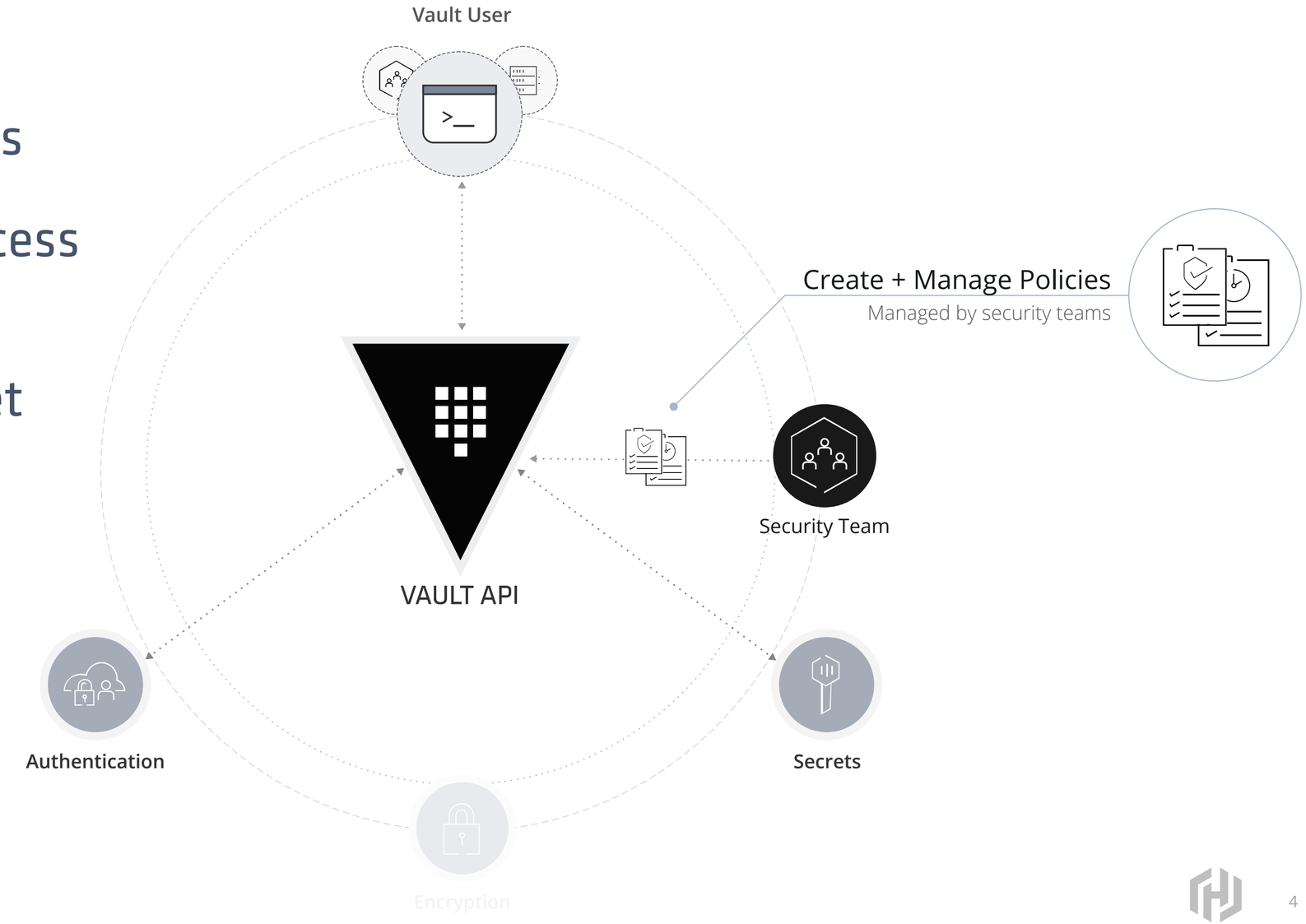
Agenda

- Access Control Policies
- Policy Authoring Workflow
- Authoring Policies
- Policy Considerations
- Managing Policies
- Associating Policies
- Testing Policies

ACL Policies

Policies

- Use policies to govern the behavior of the Vault clients
- Instrument Role-Based Access Control (**RBAC**)
- Safeguard access and secret distribution to apps



Out-Of-the-Box Policies

- **root** policy is created by default – superuser with all permissions
- **default** policy is created by default - common permissions
- Policies are written in ***HashiCorp Configuration Language*** (**HCL**), which is a human-friendly config format

Deny by default (**no policy** = **no authorization**)

Authenticating as Root

- Most interactions with Vault require a **token**
  - Tokens have policies associate with it
- Authenticating with **root** token should be limited
- Use the root token *only to*:
  - Setup the initial policies and configurations
  - Handle emergencies where root privilege is needed

Policy Authoring Workflow

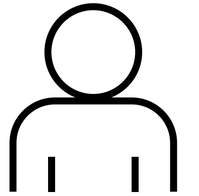
Policy Authoring Workflow

Gather Policy Requirements → Write Policies → Test the Policies

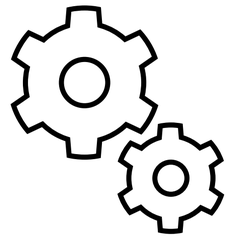
- Who needs to access what?
  - Which path(s) to access
  - What operations must be allowed
    - Role-based policies
  - Any restriction on parameters and parameter values
  - etc.
- Author policies to fulfill the requirements
  - Practice **least privileged**
  - Fine-grained policies
  - *Empty policy means no authorization*
- Test to make sure that the policies do not grant too much or too little permissions

Policy Requirement Examples

- **admin** users must be able to perform the following:
  - Configure and manage **auth methods** broadly across Vault
  - Configure and manage **secret engines** broadly across Vault
  - Create and manage **policies** broadly across Vault
  - Read system **health check**



- **web-app** clients must be able to perform the following:
  - Request an **AWS credential** granting to read from an Amazon S3 bucket
  - **Read** secrets from `secret/apikey/Google`



Authoring Policies

Anatomy of ACL Policies

- Everything in Vault is **path** based
  - Path corresponds to the Vault API endpoints
- Write policies to grant or forbid access to certain paths and operations

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

- Empty policy grants **no permission** in the system

Paths (1 of 2)

- ACL policy path supports wildcard ("*") at the end of the path

```
path "secret/team-*" { ... }
```

This matches any path starting with "**secret/team-**":

`secret/team-`**`security`**

`secret/team-`**`A/project`**

- There are paths used by the system & paths mounted by the users:

Path Mount Point	Description
`auth/`	Endpoint for auth method configuration
`cubbyhole/`	Endpoint used by the Cubbyhole secrets engine
`identity/`	Endpoint for configuring Vault identity (entities and groups)
`secret/`	Endpoint used by Key/Value v2 secrets engine ***if running in dev mode***
`sys/`	System endpoint for configuring Vault

Paths (2 of 2)

- ACL policy path supports wildcard matching for a **single directory** in the path definition. ("**+**")

```
path "secret/app/+/dev" { ... }
```

This matches:

```
secret/app/release_1.0/dev
secret/app/properties/dev
etc.
```

- Combine with "*" and "**+**":

```
path "secret/app/+/*" { ... }
path "secret/app/+/team-*" { ... }
```

Capabilities (1 of 2)

- Capabilities are specified as a list of strings even if there is only one capability

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

  - A capability similarly matches to an HTTP verb, but the mapping is **not** strictly 1:1
    - Look up the HTTP API documentation for the paths and HTTP verbs

Capabilities (2 of 2)

- There are some capabilities do **not** map to HTTP verbs

Capability	Description
sudo	Allows access to paths that are ***root-protected***
deny	Disallows access regardless of any other defined capabilities

NOTE: "**deny**" always takes precedence regardless of any other defined capabilities.

Policy File

- Save the policy file with **`.hcl`** extension (e.g. **`admin-policy.hcl`**)

admin-policy.hcl

```
# Manage auth backends broadly across Vault
path "auth/*"
{
  capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}

# List, create, update, and delete auth backends
path "sys/auth/*"
{
  capabilities = ["create", "read", "update", "delete", "sudo"]
}

# Manage secret engines broadly across Vault
path "sys/mounts/*"
{
  capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
...
```

Root-protected Endpoint Examples

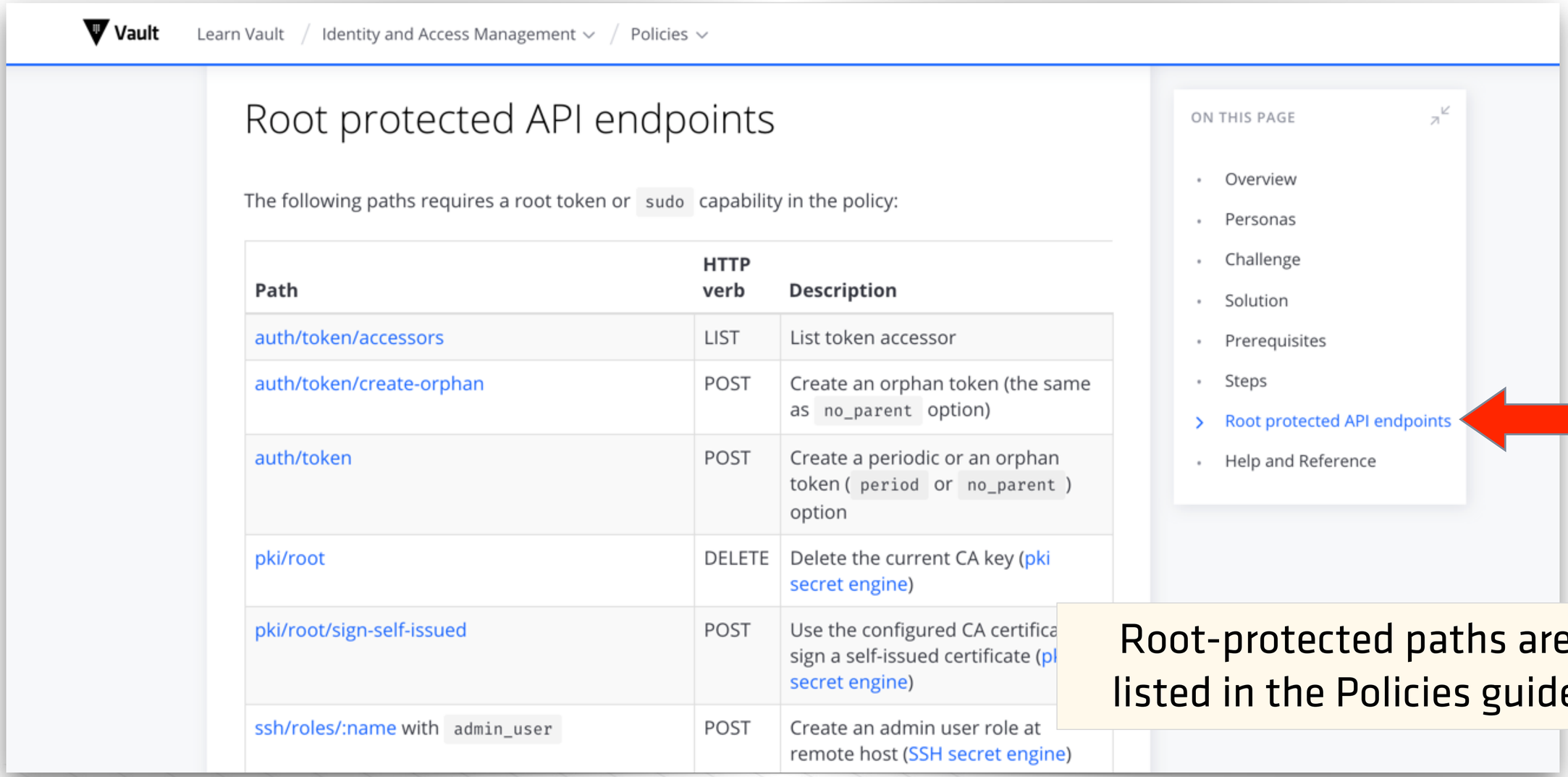
Terminal

```
$ vault operator rotate

Error rotating key: Error making API request.

URL: POST http://127.0.0.1:8200/v1/sys/rotate
Code: 403. Errors:

* 1 error occurred:
    * permission denied
```

To rotate the encryption key, the following policies must be granted:

```
path "sys/rotate" {
  capabilities = [ "update", "sudo" ]
}

path "sys/key-status" {
  capabilities = [ "read" ]
}
```

List of Root-protected Endpoints



Root-protected paths are listed in the Policies guide

Policy Considerations

Policy Consideration: Path Hierarchy

- The "*" in path allows any value in its place. This includes:
  - secret/data/training/projA
  - secret/data/training/test
  - secret/data/training/api
  - ..etc
- However, this does **NOT** include its parent path, "secret/data/training"
- Remember, no policy means no permission!

training-policy.hcl

```
# Any path starting with secret/training
path "secret/data/training/*" {
    capabilities = ["create", "read", "update", "delete",
                    "list", "sudo"]
}

# Add a policy rule if required
path "secret/data/training" {
    capabilities = ["read", "list"]
}
```

Policy Consideration: Excluding a Path

- Think of a scenario where you want to grant all permission on paths starting with "secret/data/training"
  - secret/data/training/projA
  - secret/data/training/test
  - secret/data/training/api
  - ..etc
- **One exception to add:** No permission granted on "secret/data/training/**team-admin**" path

training-policy.hcl

```
# Any path starting with secret/training
path "secret/data/training/*" {
    capabilities = ["create", "read", "update", "delete",
                    "list", "sudo"]
}

# Deny any operation on secret/training/team-admin path
path "secret/data/training/team-admin" {
    capabilities = ["deny"]
}
```

Or, you can have an empty list.
Vault uses **deny-by-default** model.

Managing Policies

Policy CLI Commands

- Usage: `vault policy <sub-command> [options] [args]`

Sub-command	Arguments	Description
list	none	Lists the names of the policies that are installed
write	<policy_name> <file_path>	Add or update a policy name <policy_name>, and its definition written in <file_path>
read	<policy_name>	Prints the contents and metadata of a specific policy
delete	<policy_name>	Delete a specific policy
fmt	<file_path>	Formats a local policy file to the policy specification.

Policy CLI Commands

```
# Display the help message for a policy command
$ vault policy <sub-command> -h

# List all existing policy names
$ vault policy list

# Create a new policy
$ vault policy write admin ./admin-policy.hcl

# Delete a policy named "admin"
$ vault policy delete admin
```

- **NOTE:** The "default" and "root" policies are built-in policies that the cannot be deleted.

Policy API Endpoints

HTTP verb	Endpoint	Description
GET	/sys/policies/acl	Lists the names of the policies that are installed
GET	/sys/policies/acl/<policy_name>	Prints the contents and metadata of a specific policy
PUT	/sys/policies/acl/<policy_name>	Add or update a policy named <policy_name>. Pass the policy file in the request payload
DELETE	/sys/policies/acl/<policy_name>	Delete a specific policy

NOTE: There are **`sys/policies/rgp`** and **`sys/policies/egp`** endpoints for Sentinel policies.

Policy for Policies

- To allow users to be able to manage policies, you need to write policies for that

admin.hcl

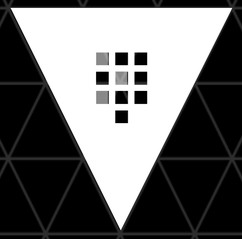
```
...

# Create and manage ACL policies via CLI
path "sys/policies/acl/*"
{
  capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}

# Create and manage ACL policies via API
path "sys/policies/acl"
{
  capabilities = ["list"]
}

...
```

Associating Policies

Associating Policies

- During the configuration of users, roles, entities, and groups, a list of associated polices can be specified
- Create new token with specific policies attached

```
# LDAP group, "sre" has "dev" & "ops" policies attached
$ vault write auth/ldap/groups/sre policies="dev, ops"


# Create a new token with "admin" policy attached
$ vault token create -policy="admin"
```



Associating Policies

Terminal

```
# Create a new token with "web-app" policy attached
$ vault token create -policy="web-app"

Key                  Value
---                  -----
token                s.7uBlZwXSxOg31uGXIUetEdXD
token_accessor       18r88muoe3x1xEqVqXdlTMwJ
token_duration       768h
token_renewable      true
token_policies       ["default" "web-app"]
identity_policies    []
token_policies       [default web-app]
```

Every token get the **default** policy automatically

Test Policies

Testing Policies

- Test to make sure that the policy fulfills the requirements
- Example requirements:
  - **web-app** clients must be able to perform the following:
    - Request an **AWS credential** granting to read from Amazon S3 bucket
    - **Read** secrets from `secret/apikey/Google`

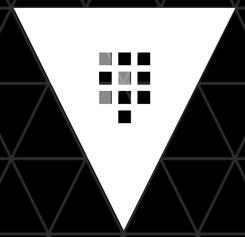
Terminal

```
# Create a new token with "web-app" policy attached
$ vault token create -policy="web-app"

# Authenticate with the newly generated token
$ vault login <token>

# Make sure that the token can read
$ vault read secret/apikey/Google

# This should fail
$ vault write secret/apikey/Google key="ABCDE12345"

# Request a new AWS credentials
$ vault read auth/aws/creds/s3-readonly
```

Print Token Capabilities

- Usage: **`vault token capabilities [options] [token] path`**

```
# List granted permission on secret/training path
$ vault token capabilities s.b30a25c9-fe8f... secret/training
create, delete, read, update

# In the absence of token, check against the current token
$ vault token capabilities secret/training
root
```



Read an Existing Policy

Terminal

```
# vault policy read <policy_name> to read
$ vault policy read admin
path "auth/*"
{
  capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}

path "sys/auth/*"
{
  capabilities = ["create", "read", "update", "delete", "sudo"]
}

path "sys/policies/acl"
{
  capabilities = ["list"]
}
...
```

Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which operations are granted by the following policy:

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

✔A. vault kv put secret/web/eng/team-X apikey="AW549IRUW23923I"

B. vault kv delete secret/web/eng/team-X

✔C. vault kv get secret/web/eng/team/team-X

D. vault kv list secret/web/eng

Lab 6: Working with Policies

25 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Create a new policy
  - Task 2: Test the policy
  - *Challenge*: Create a new policy based on a given set of requirements

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

Fine-Grained Control

Key/Value version 1

Not supported with kv-v2

Fine-Grained Control: required_parameters

- Optional, finer-grained control options are available:
  - **required_parameters** - A list of parameters that must be specified

```
path "secret/tests/phase1" {
    capabilities = [ "create", "update" ]
    required_parameters = [ "date", "owner" ]
}
```

- This requires the user to create "**secret/tests/phase1**" with a parameter named "**date**" and "**owner**"

Fine-Grained Control: allowed_parameters

- **allowed_parameters** - Whitelists a list of parameter and values that are permitted on the given path

```
path "secret/tests/phase2" {
    capabilities = [ "create", "update" ]
    allowed_parameters = {
       "state" = [ "stopped", "started", "starting", "stopping" ]
       "*" = []
    }
}
```

"state" parameter can only have the listed values

Any other parameters may be created with any value

Fine-Grained Control: denied_parameters

- **denied_parameters** - Blacklists a list of parameter and values
  - Takes precedence over **allowed_parameters**

```
path "secret/tests/phase3" {
    capabilities = [ "create", "update" ]
    denied_parameters = {
       "owner" = [ "anyone", "everyone", "all" ]
    }
}
```

"owner" parameter can NOT have any of the listed values

- To deny any parameter to be created:

```
denied_parameters = {
      "*" = []
  }
```

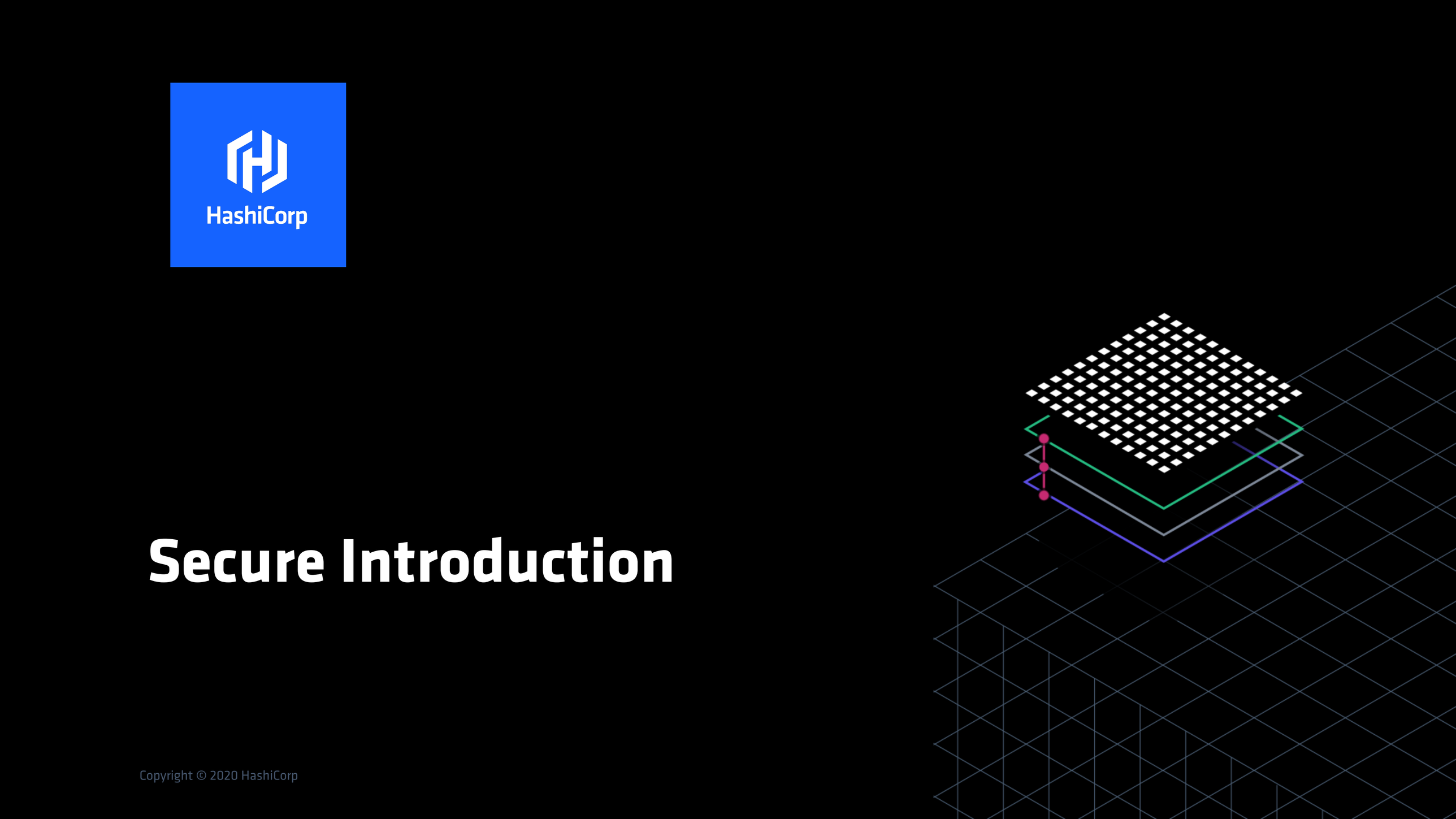
Fine-Grained Control: wildcard

- Parameter values also support prefix/suffix globbing

```
path "secret/tests/phase4" {
    capabilities = [ "create", "update" ]
    allowed_parameters = {
       "username" = [ "test-*" ]
       "*" = []
    }
}
```

username parameter is allowed for as long as its value starts with "**test-**"

Wildcard parameter ("*") cannot have parameter value list (has to be [ ]).



Secure Introduction

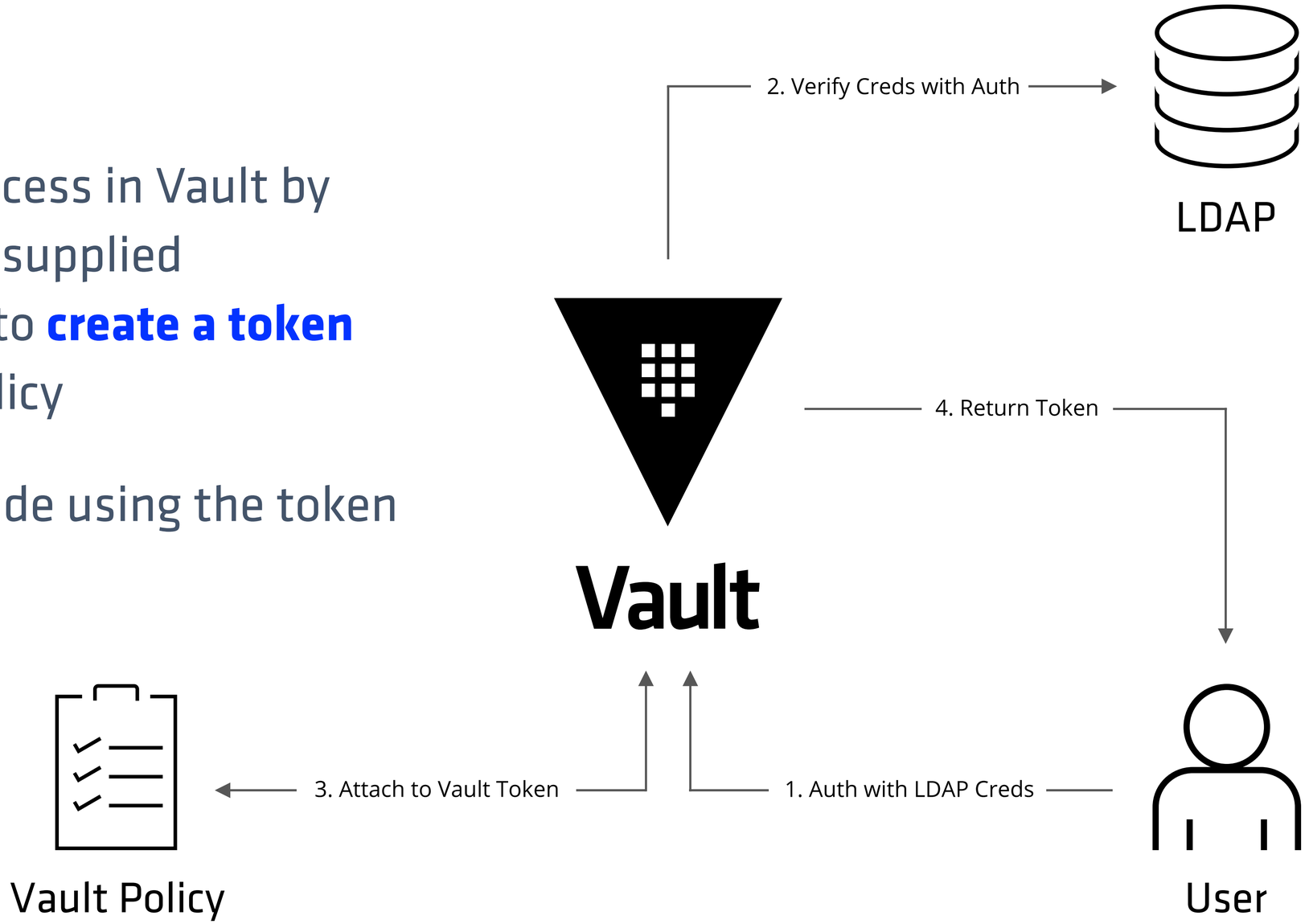
Agenda

- Authentication
- Vault Secure Introduction using Vault Agent
  - Vault Agent Auto-Auth
  - Vault Agent Caching
- Cubbyhole Response Wrapping

Authentication

Authentication

- **Authentication** is a process in Vault by which user or machine-supplied information is verified to **create a token** with pre-configured policy
- Future requests are made using the token



Tokens

- Tokens are the ***core*** method for authentication
  - Most operations require existing authenticated tokens
- The **token auth method** is responsible for creating and storing tokens
  - token auth method cannot be disabled
  - Authenticating with external identity (e.g. LDAP) dynamically generates tokens
- Tokens have one or more policies attached to control what the token is allowed to perform

Token Types

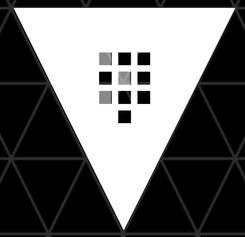
- There are two types of tokens:
  - **Service tokens** are persisted
    - Tokens have time-to-live (TTL) and max TTL
    - Tokens can be **renewed** or **revoked** before reaching its TTL until it reaches its max TTL
  - **Batch tokens** are NOT persisted (ephemeral)
    - Tokens have TTL (No max TTL)
    - Tokens **cannot** be renewed or revoked before reaching its TTL
    - Tokens are **encrypted binary large objects** (blobs)
    - Carries just enough information

Auth Methods (1 of 2)

Method	Description
AppRole	Allows machines or apps to authenticate with Vault.
AliCloud	Retrieves a Vault token for AliCloud entities
AWS	Retrieves a Vault token for AWS EC2 instances and IAM principals
Azure	Authenticate against Vault using Azure Active Directory credentials
Cloud Foundry	Retrieves a Vault token for Cloud Foundry instances.
Google Cloud	Allows authentication against Vault using Google credentials
GitHub	Authenticate with Vault using a GitHub personal access token
JWT / OIDC	Authenticate with Vault using a JSON Web Token (JWT) or using OIDC

Auth Methods (2 of 2)

Method	Description
Kerberos	Retrieve a Vault token for Kerberos entities
Kubernetes	Authenticate with Vault using a Kubernetes Service Account
LDAP	Allows authentication using an existing LDAP server
Okta	Allows authenticating using Okta and user/password credentials
RADIUS	Authenticate with Vault using an existing RADIUS server
TLS Certificates	Authenticate using SSL/TLS client certificates
Tokens	**Built-in, automatically available authentication method**
Username & Password	Allows users to authenticate using a username and password

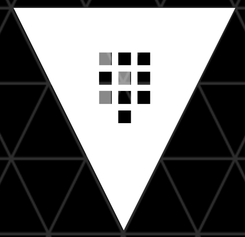


Configure Auth Method

Command Example

Terminal

```
# vault auth enable <auth_method>
$ vault auth enable ldap
Success! Enabled ldap auth method at: ldap/

# Configure the auth method
$ vault write auth/ldap/config \
        url="ldap://ldap.example.com" \
        userdn="ou=Users,dc=example,dc=com" \
        groupdn="ou=Groups,dc=example,dc=com" \
        ...

# Map policies
$ vault write auth/ldap/groups/engineers policies=eng
```



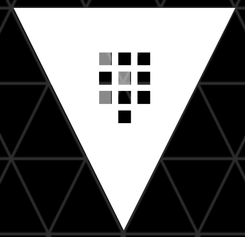
Login with GitHub auth method

Command Example

Terminal

```
$ vault login -method=ldap username=james@example.com

Password (will be hidden):

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.Cy1cZb3SxPt0AyrItnfsYKOt
token_accessor
token_duration
token_renewable      true
token_policies       [default eng]
...
```

Service tokens are prefixed with "**s.**"



Set Token Type

Command Example

Terminal

```
$ vault auth enable approle
$ vault write auth/approle/role/billing policies="billing" \
        token_type="batch" token_ttl="60s"

$ vault write auth/approle/login role_id="..." secret_id="..."
Key                     Value
---                     -----
token                   b.AAAAAQJRhQ1NPS3mCi_eZNToL587ZWuCWPzX...
token_accessor
token_duration
token_renewable         false
token_policies          ["default" "shipping"]
...
```

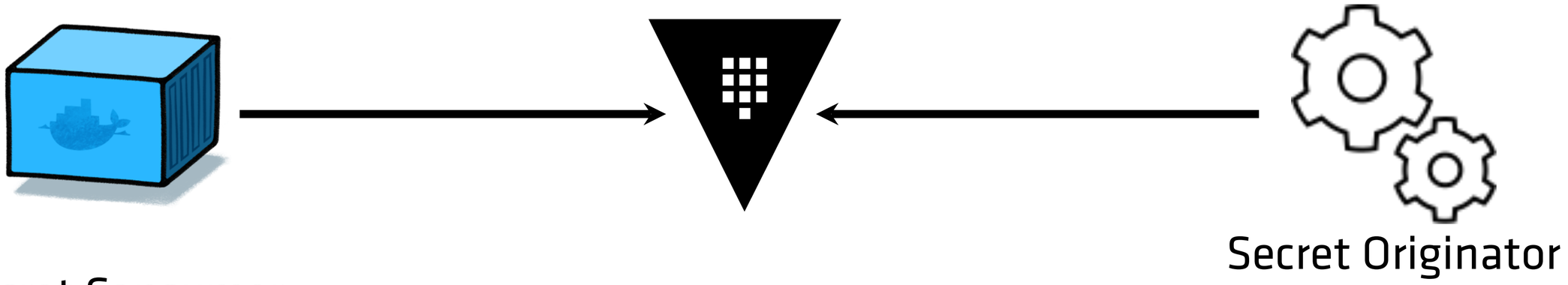
Batch tokens are prefixed with "**b.**" and much longer than service tokens

Secure Introduction

Secret zero problem

Secret Originator and Consumer

If you can securely get the first secret from originator to a consumer, all subsequent secrets transmitted between them can be authenticated with the trust established by the successful distribution of the first secret.



Challenge

- **Tokens** are the *core* method for authentication within Vault
  - Every secret consumer (client) must acquire a valid token

- How does a secret consumer (an application or machine) prove that it is the legitimate recipient for a secret so that it can acquire a token?

- How do I securely distribute the initial token to a machine or application?

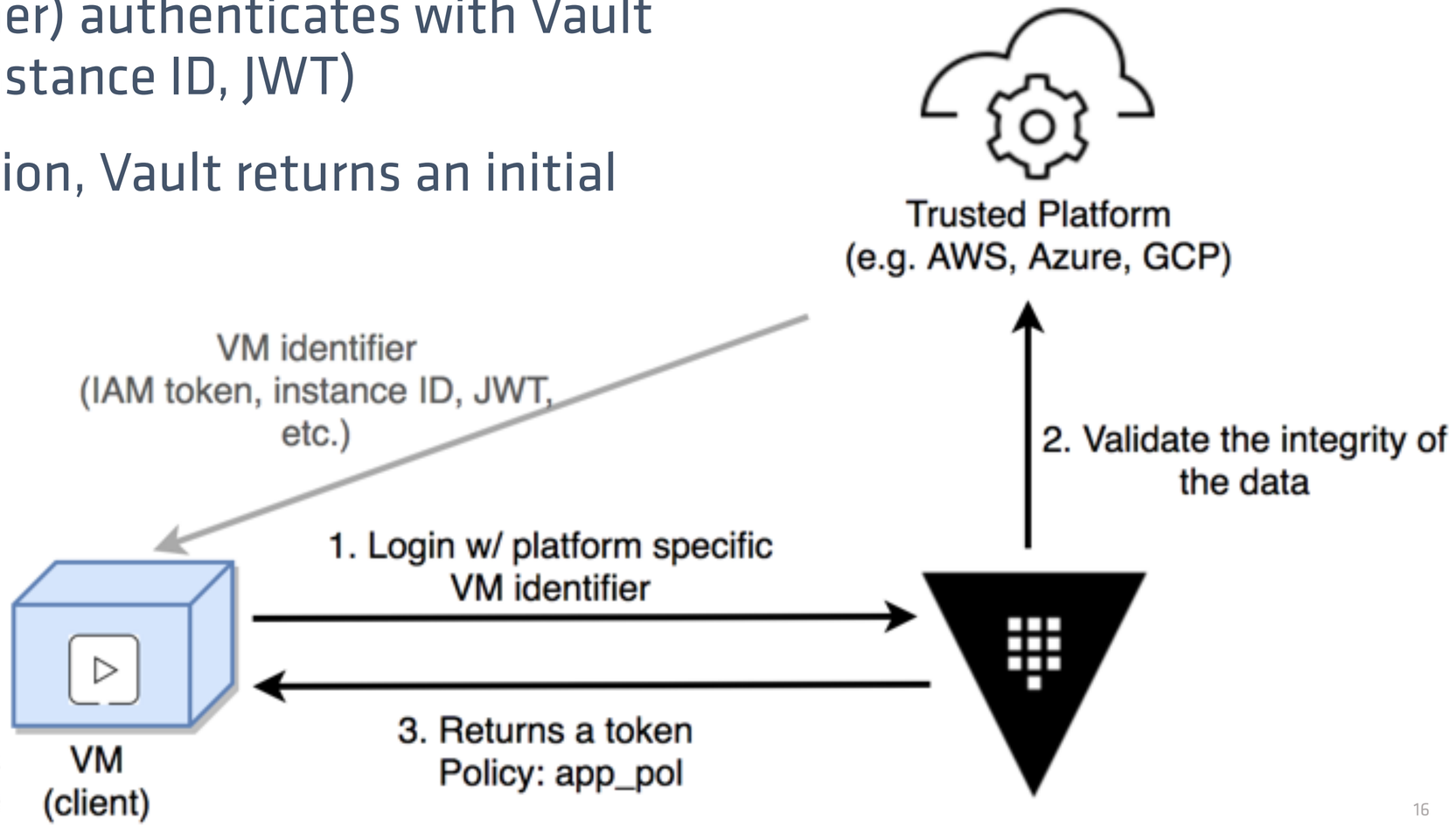
Secret zero problem!

Secure Introduction Approaches

- There are three basic approaches to secure introduction of Vault clients
  - Platform Integration
  - Trusted Orchestrator
  - Vault Agent

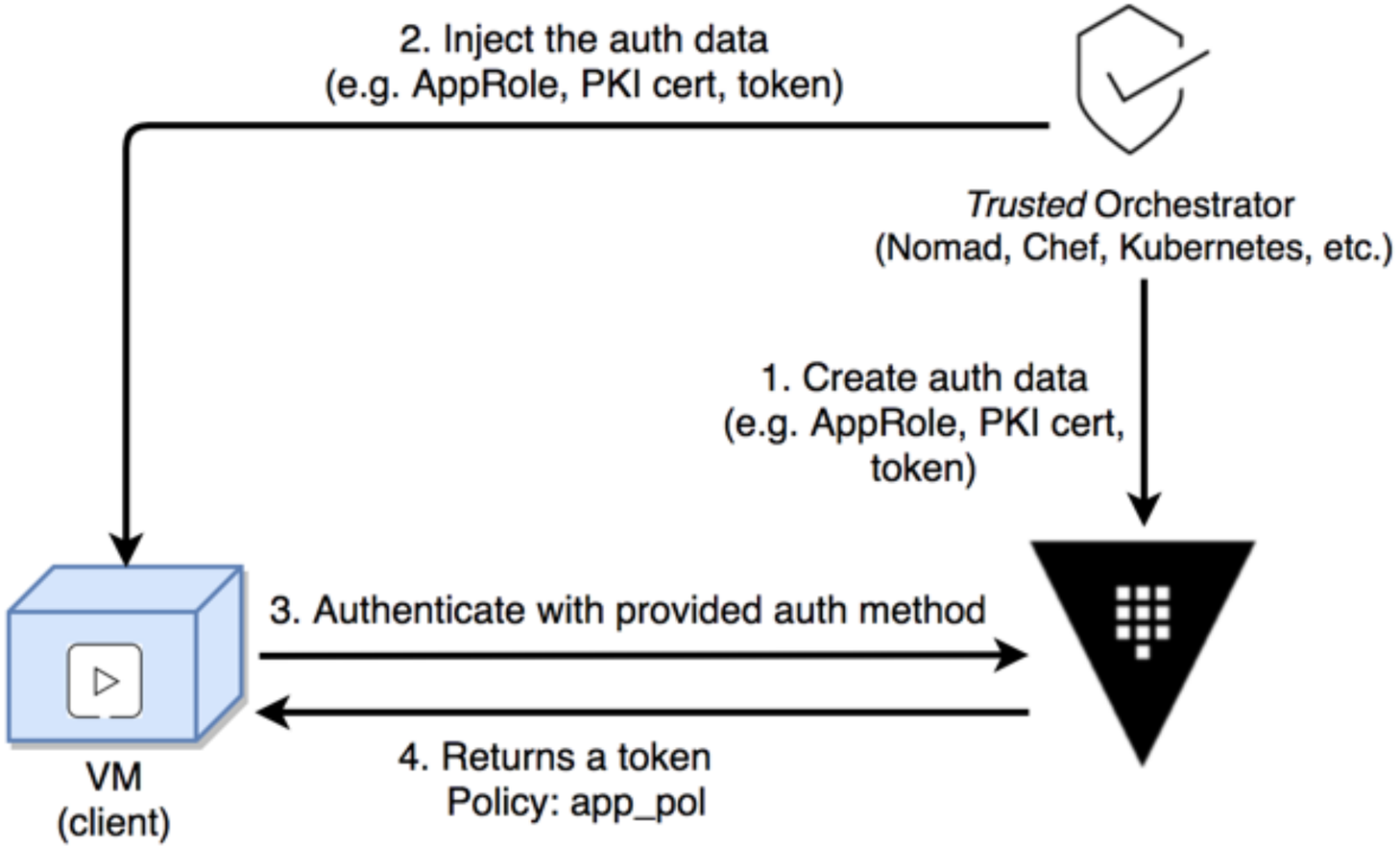
Secure Introduction: Platform Integration

- Vault trusts the underlying platform (e.g. AWS)
- The client (secret consumer) authenticates with Vault using its identifier (e.g. instance ID, JWT)
- Upon a successful validation, Vault returns an initial token



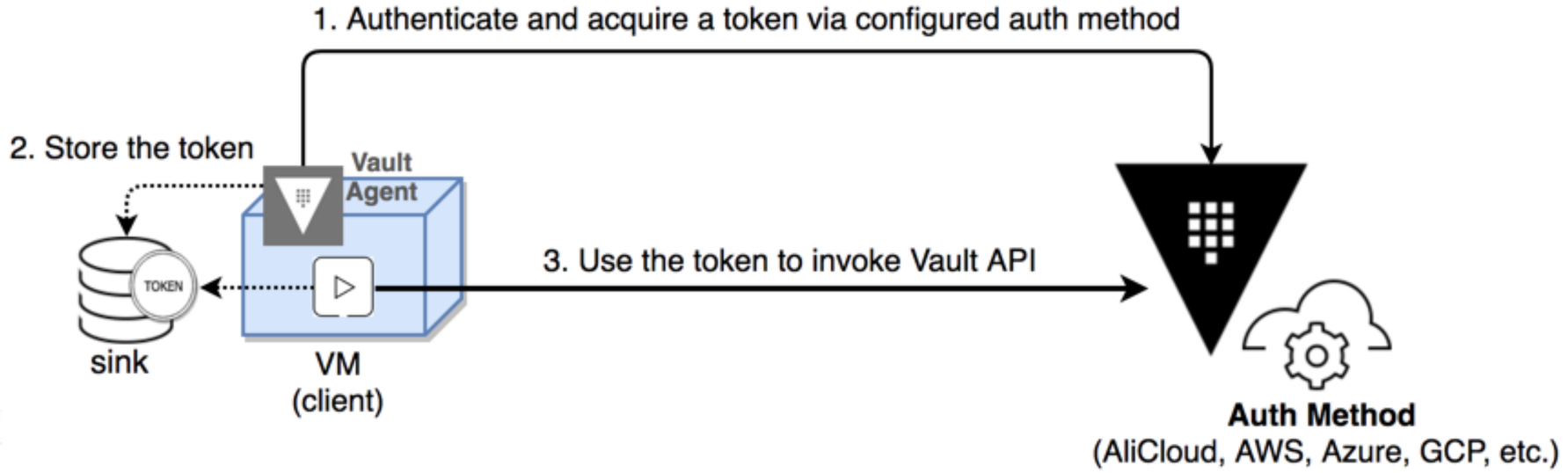
Secure Introduction: Trusted Orchestrator

- Orchestrator is already authenticated with Vault with privileged permissions
  - Launches new applications
  - Orchestrator injects information necessary for the client to authenticate with Vault
- The client proves that it is the legitimate recipient of the secret using the information



Secure Introduction: Vault Agent

- Vault Agent is a **client daemon** which automates the client authentication workflow:
  - Authenticate and acquire a Vault token using the configured auth method
    - Tokens can be response-wrapped or encrypted
  - Renew the token until renewal is no longer allowed



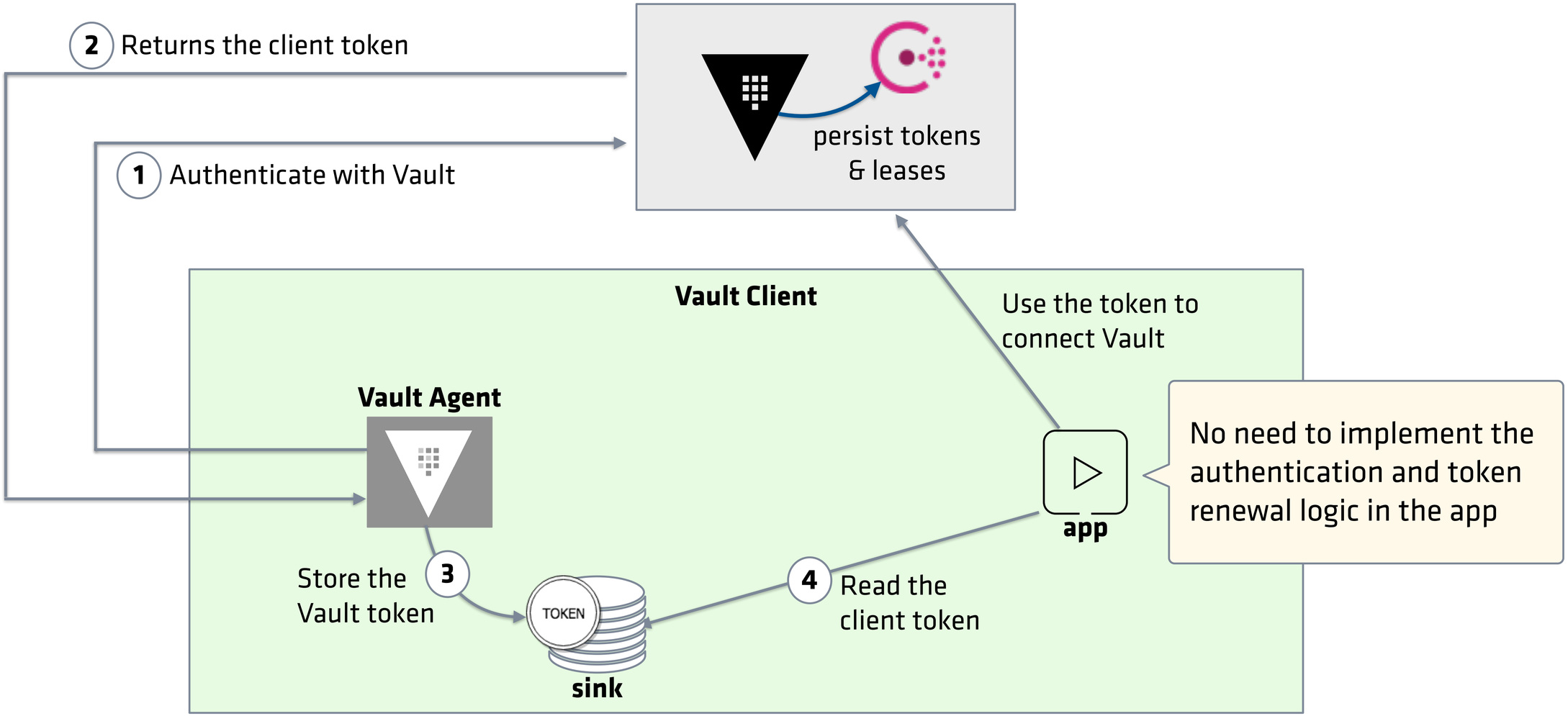
Vault Agent

Vault Agent

- Vault Agent approach can be used with either ***platform integration*** or ***trusted orchestration*** approach

- Vault Agent's **Auto-Auth** allows for easy authentication to Vault:
  - Automatically **authenticates** to Vault using supported auth methods
  - Keeps the client token **renewed** until the renewal is no longer allowed
  - Client tokens can be **response wrapped**

Auto-Auth Basic Workflow



Auto-Auth Configuration

- Auto-auth consists of two parts:
  - A **method** - auth method to use
  - **Sink**(s) - locations where the agent writes the token
- **pid_file** is where the agent's process ID (PID) should be stored
- Vault server address can be set via **VAULT_ADDR** env var as well

Example: agent-config.hcl

```
pid_file = "/home/vault/pidfile"

auto_auth {
    method "kubernetes" {
        mount_path = "auth/kubernetes"
        config = {
            role = "example"
        }
    }

    sink "file" {
        config = {
            path = "/home/vault/.vault-token"
        }
    }
}

vault {
    address = "http://vault.server.address.com:8200"
}
```

Uses Kubernetes auth method enabled at **auth/kubernetes** path on the Vault server. Authenticate as "example" role

Vault Agent writes the acquired client token at this location

Running Vault Agent

1. Download and install the Vault binary on the client host

2. Start Vault in agent mode

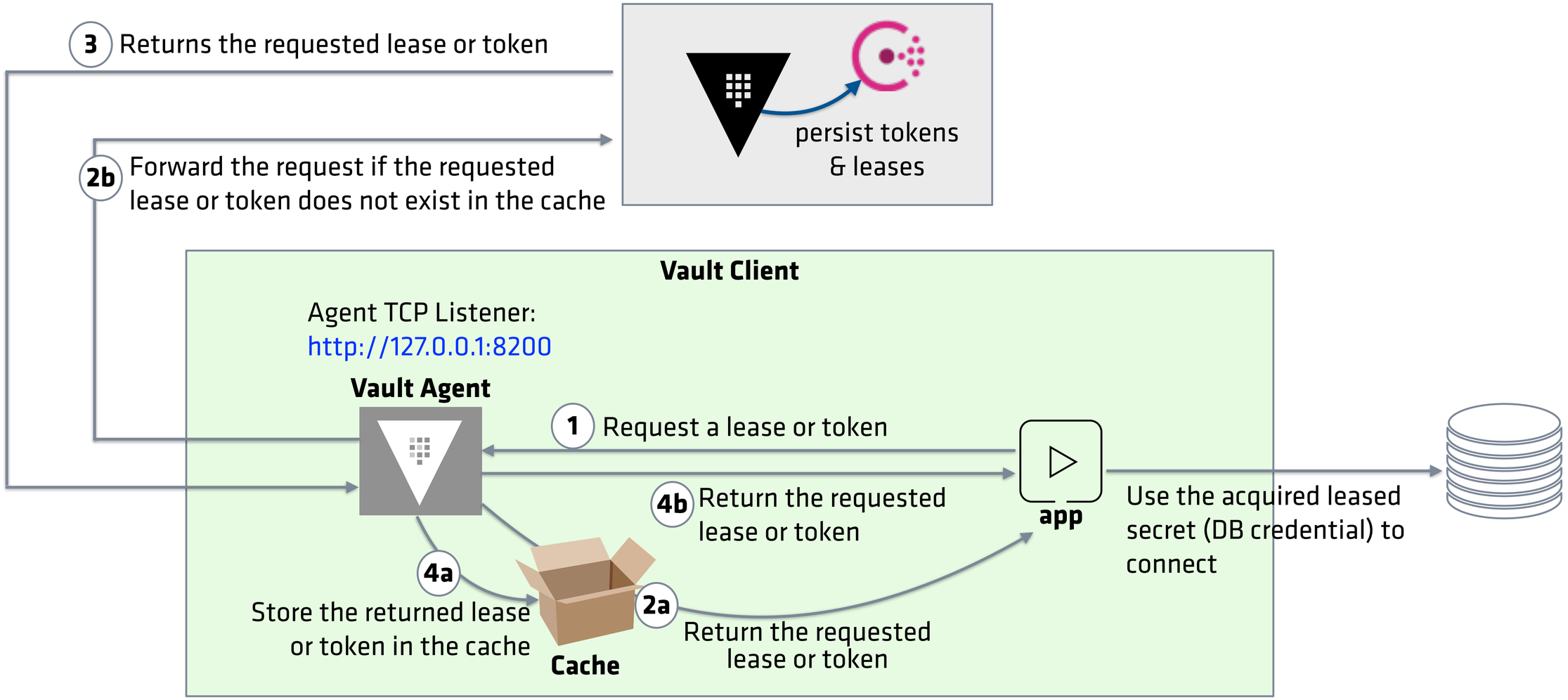
```
# To get help
$ vault agent -h

# start the Vault Agent
$ vault agent -config=/etc/vault/agent-config.hcl
```

Vault Agent Caching

Vault Agent Caching

- Vault Agent **Caching** allows caching of tokens and leases on the **client** side
  - Easier access to Vault secrets for edge applications
  - Reduces I/O burden for basic secret access
  - Secure local access to leases and tokens
- **Tokens** and **leases** are cached if:
  - Token creation requests were sent via agent
  - Login request sent via agent
  - Leased secret creation requests were made via agent using tokens that are managed by the agent (this includes **auto-auth** token)

Vault Agent Caching Basic Workflow



Caching Configuration

- Cache block contains:
  - **use_auto_auth_token**
- **listener** block defines:
  - type ("**tcp**" or "**unix**")
  - address
  - tls_disable
  - tls_key_file
  - tls_cert_file

Example: agent-config.hcl

```
auto_auth {
    ...
}

cache {
   use_auto_auth_token = true
}

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

vault {
   address = "http://vault.server.address.com:8200"
}
```


Cache Eviction

- The eviction of cached entries occur when:
  - Vault agent fails to renew the lease or token (e.g. it reached its max TTL)
  - Token or lease was revoked via agent
- When a token or lease was revoked outside of the agent's knowledge, stale entries exist in the cache
  - Programmatically clear the stale cache using **`/agent/v1/cache-clear`** endpoint

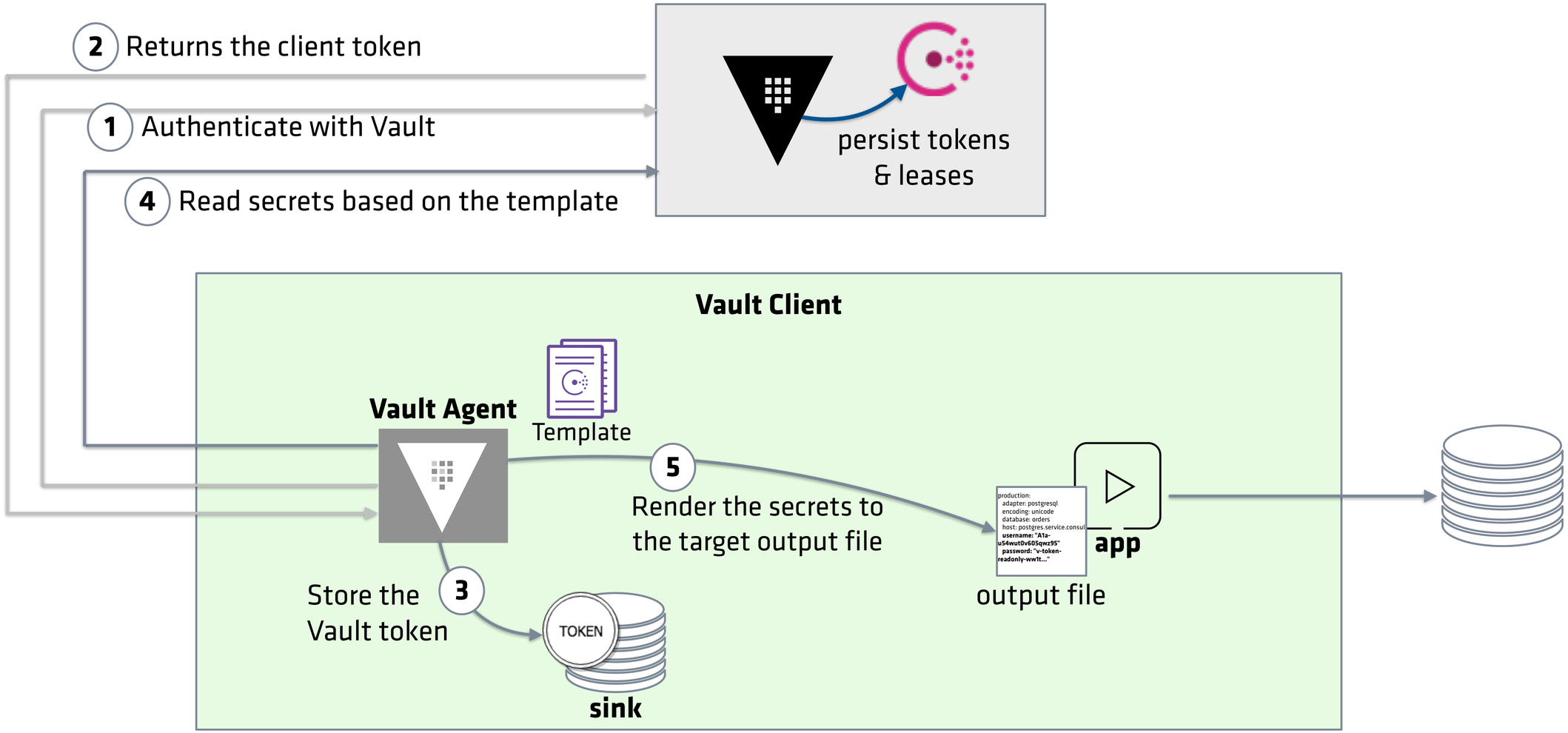
```
$ curl --request POST \
       --data '{"type": "lease", "value": "aws/creds/readonly"}' \
       http://127.0.0.1:8200/agent/v1/cache-clear
```


Vault Agent Templating

Vault Agent Templates

- **Vault Agent Templates** was introduced in **Vault 1.3**
  - A subset of the Consul Template functionality is directly embedded into the Vault Agent
    - No need to install the Consul Template binary
  - Vault secrets can be rendered to the destination files using the Consul Template markup language
  - Uses the client token acquired by the auto-auth
    - No need to implement a logic between the agent and Consul Template

Templates Basic Workflow



Templates Configuration

- Template block contains:
  - **source**
  - **destination**
  - Refer to the doc for the full list of available parameters: https://www.vaultproject.io/docs/agent/template/index.html

Example: agent-config.hcl

```
auto_auth {
    ...
}

vault {
   address = "http://10.1.0.34:8200"
}

template {
   source = "/path/to/config.yml.tmpl"
   destination = "/path/to/config.yml"
}
```


Additional Resources

- Vault Agent with AWS guide: https://learn.hashicorp.com/vault/identity-access-management/vault-agent-aws
- Vault Agent with Kubernetes guide: https://learn.hashicorp.com/vault/identity-access-management/vault-agent-k8s
- Vault Agent Caching guide: https://learn.hashicorp.com/vault/identity-access-management/agent-caching
- Vault Agent Templates guide: https://learn.hashicorp.com/vault/identity-access-management/agent-templates
- Injecting Secrets into Kubernetes Pods via Vault Helm Sidecar: https://learn.hashicorp.com/vault/getting-started-k8s/sidecar

Knowledge Check & Lab Overview

Knowledge Check Questions

1. True or False: Batch tokens are persisted in the storage backend until it expires.
2. True or False: Vault Agent can have only one sink to store the client token.
3. True or False: When a cached token is revoked, Vault Agent automatically evicts the cached token.
4. True or False: Response wrapping token TTL should be set to a short period so that you can limit the secrets exposure time.

Lab 7: Vault Agent

20 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Run Vault Agent
  - Task 2: Test Vault Agent Caching
  - Task 3: Evict Cached Leases

Vault Agent Auto-Auth with Response Wrapping

Security Concern

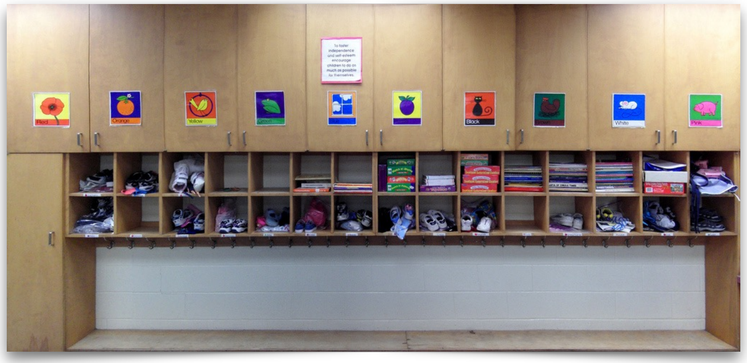
Security Engineer

I generated a token for a trusted entity with appropriate policy attached so that it can interact with Vault.

I'm little concerned about sending the token over the network. Can I protect it against a potential man-in-the-middle (MITM) attack?

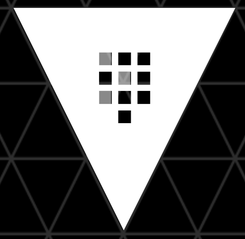
Solution: Cubbyhole Response Wrapping

- Use Vault's cubbyhole response wrapping
  - Stores the initial token inside a temporary (restricted) token's cubbyhole with a short TTL
  - Allows only the expecting client (trusted entity) to unwrap this secret
    - Wrapping token to unwrap the secret is a ***single-use*** token
- Any secret can be distributed using the response wrapping



Benefits of Response Wrapping

- It provides ***cover*** by ensuring that the value being transmitted across the wire is not the actual secret
- It provides ***malfeasance detection*** by ensuring that only a single party can ever unwrap the token and see what's inside
- It ***limits the lifetime*** of the secret exposure
  - The TTL of the response-wrapping token is separate from the wrapped secret's lease TTL



Wrap Secrets

(Privileged User Task)

Terminal

Expires after 60 seconds

```
$ vault token create -policy=jenkins -wrap-ttl=60

Key                                 Value
---                                 -----
wrapping_token:                     s.3KRbJu4S8dkdTPqQtIt9kUYA
wrapping_accessor:                  2gtVQC08OirVCrr0ckxzHMIb
wrapping_token_ttl:                 1m
wrapping_token_creation_time:       2018-12-13 18:24:36.518517...
wrapping_token_creation_path:       auth/token/create
wrapped_accessor:                   59IKQOhanQicPGK4fsuUI11Q
```

Response Wrapping is triggered by providing the desired TTL (**`-wrap-ttl`**)

Unwrap the Secret

- Trusted entities execute *unwrap* command to retrieve the secret
- Usage: **`vault unwrap [options] <wrapping_token>`**

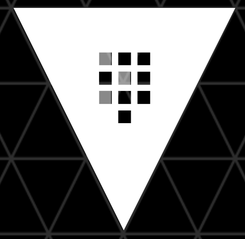
```
$ vault unwrap <wrapping_token>
```

or

```
$ VAULT_TOKEN=<wrapping_token> vault unwrap
```

or

```
$ vault login <wrapping_token>
$ vault unwrap
```



Unwrap Secrets

(Expecting Client Task)

Terminal

```
$ vault unwrap 9ad6708d-ed3b-f764-7286-f200...

Key                  Value
---                  -----
token                s.1mbNceYW8HW3oMQDxkW0X1Ol
token_accessor       39VOu1BEFBYqTgbDYDHErtnZ
token_duration       768h
token_renewable      true
token_policies       ["default" "jenkins"]
identity_policies    []
policies             ["default" "jenkins"]
```

Vault Agent Auto-Auth with Response-Wrapping (1 of 2)

- The client token can be response-wrapped:
  - By the **auth method**
    - **Pros:** Prevents man-in-the-middle (MITM) attack (more secure)
    - **Cons:** Vault agent cannot renew the token

  - By any of the **sinks**
    - **Pros:** Allow Vault agent to renew the token and re-authenticate when the token expires
    - **Cons:** The token gets wrapped *after* it's fetched; therefore, vulnerable to MITM attack

Vault Agent Auto-Auth with Response-Wrapping (2 of 2)

Response wrapped by the method

```
pid_file = "/home/vault/pidfile"

auto_auth {
    method "kubernetes" {
        wrap_ttl = "5m"
        mount_path = "auth/kubernetes"
        config = {
            role = "example"
        }
    }
  ...
}
```

Response wrapped by the sink

```
pid_file = "/home/vault/pidfile"

auto_auth {
    ...

    sink "file" {
        wrap_ttl = "5m"
        config = {
            path = "/home/vault/.vault-token"
        }
    }
}
```


Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com