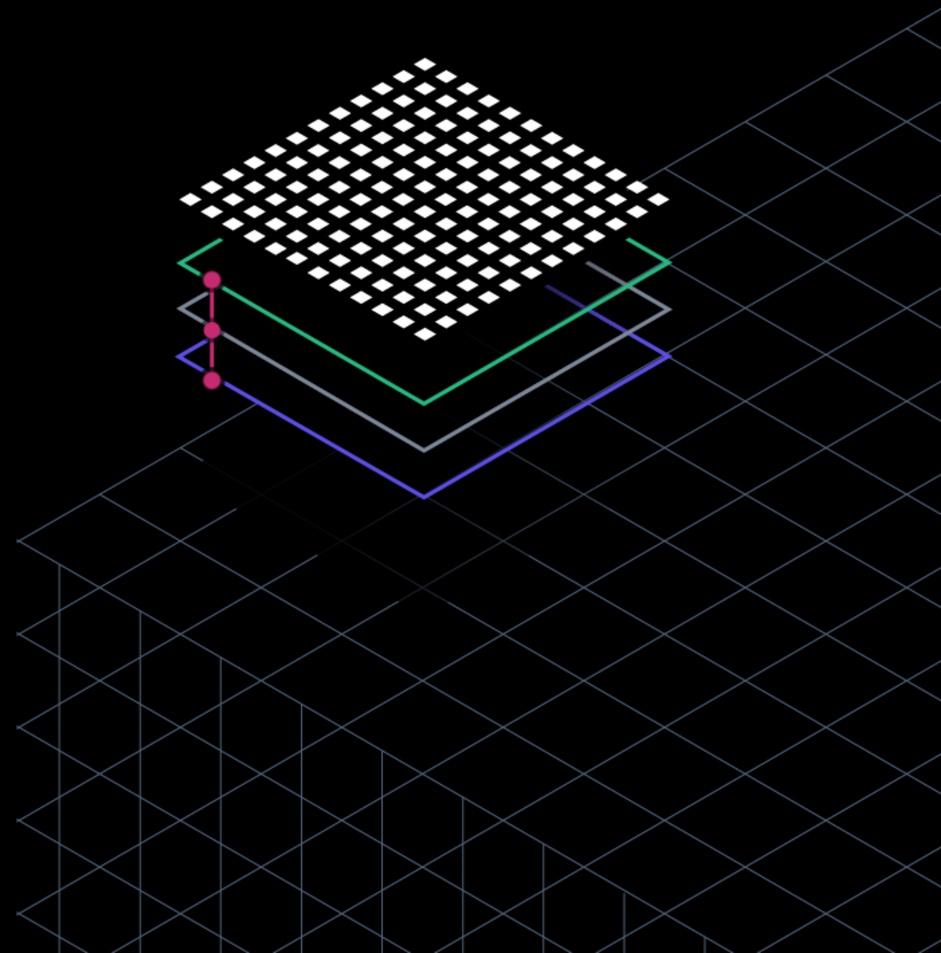
HashiCorp

Vault 101: Introduction to Vault for Developers



Courseware Download

Download the course materials from the following links:

Lecture slides: **http://bit.ly/VLT101-book**

Lab book: **http://bit.ly/VLT101-Lab-html**

Agenda

Time	Topic	Duration
9:00 - 9:15 AM	Welcome	15 min.
9:15 - 10:00 AM	Module 1: Vault Overview	45 min.
10:00 - 10:10 AM	Lab 1: Lab Setup	10 min.
10:10 - 10:50 AM	Module 2: Secret Engines - Static Secrets	40 min.
10:50 - 11:05 AM	Break	15 min.
11:05 - 11:25 AM	Lab 2: Writing Static Secrets	20 min.
11:20 - 11:40 AM	Module 3: Secret Engines - Cubbyhole Response Wrapping	20 min.
11:40 - 12:00 PM	Lab 3: Cubbyhole Secret Engine	20 min
12:00 - 1:00 PM	Lunch Break	60 min.
1:00 - 1:25 PM	Module 4: Secret Engines - Dynamic Secrets	25 min.
1:25 - 1:50 PM	Lab 4: Dynamic Secrets	25 min.
1:50 - 2:10 PM	Module 5: Encryption as a Service - Transit Secrets Engine	20 min.
2:10 - 2:35 PM	Lab 5: Encryption as a Service - Transit Secrets Engine	25 min.
2:35 - 2:50 PM	Break	15 min.
2:50 - 3:30 PM	Module 6: Authentication	40 min.
3:30 - 3:50 PM	Lab 6: Authentication and Tokens	20 min.
3:50 - 4:10 PM	Module 7: Application Integration	20 min.
4:10 - 4:45 PM	Lab 7: Application Integration	35 min.
4:45 - 5:00PM	Course Wrap-up	15 min.

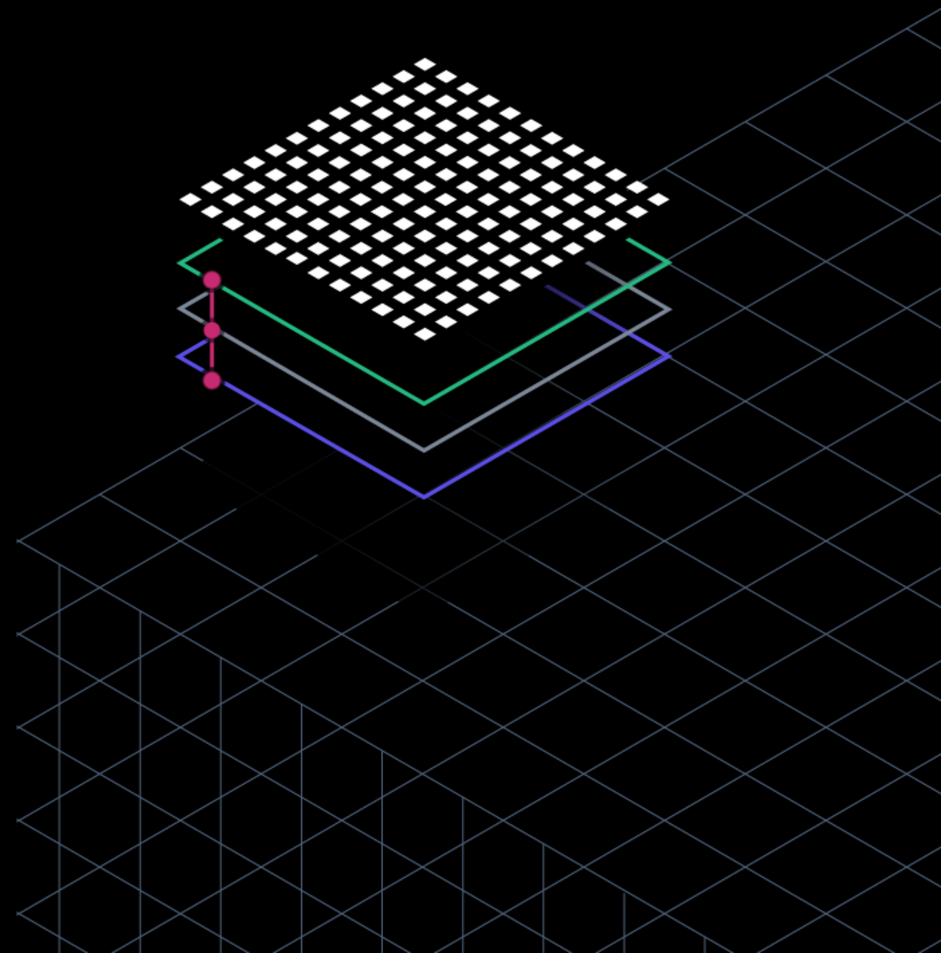
Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp

Vault Overview

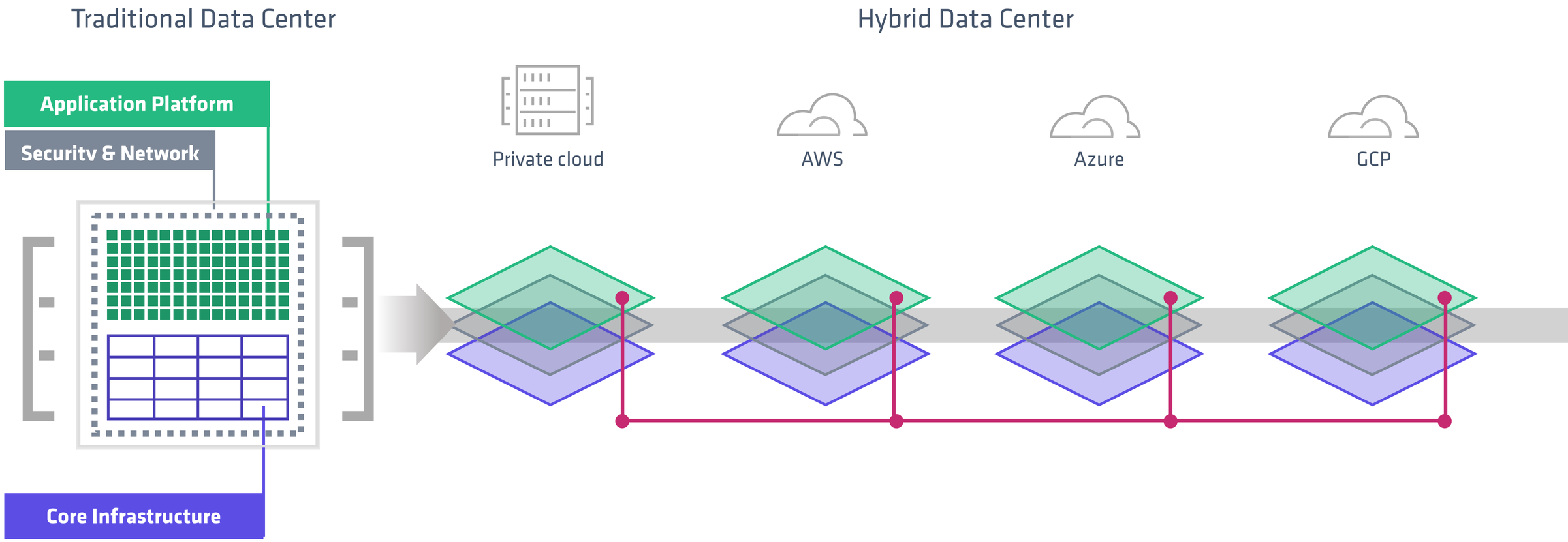


Agenda

- Vault Introduction
  - Security in a dynamic world
  - Managing secrets in distributed infrastructure
  - Vault use cases
- Vault Architecture
  - Introducing Vault components
- Vault Server
  - Sealing and unsealing a Vault server
  - Shamir's secret sharing
- How Vault Works

Vault Introduction

Shifting from Static to Dynamic Infrastructure



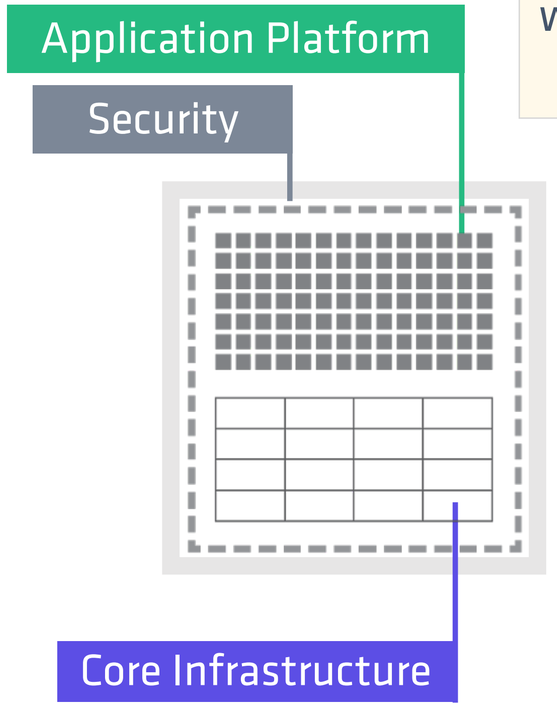
Homogenous, Static, Consolidated

Heterogeneous, Dynamic, Distributed

Security in a Dynamic World

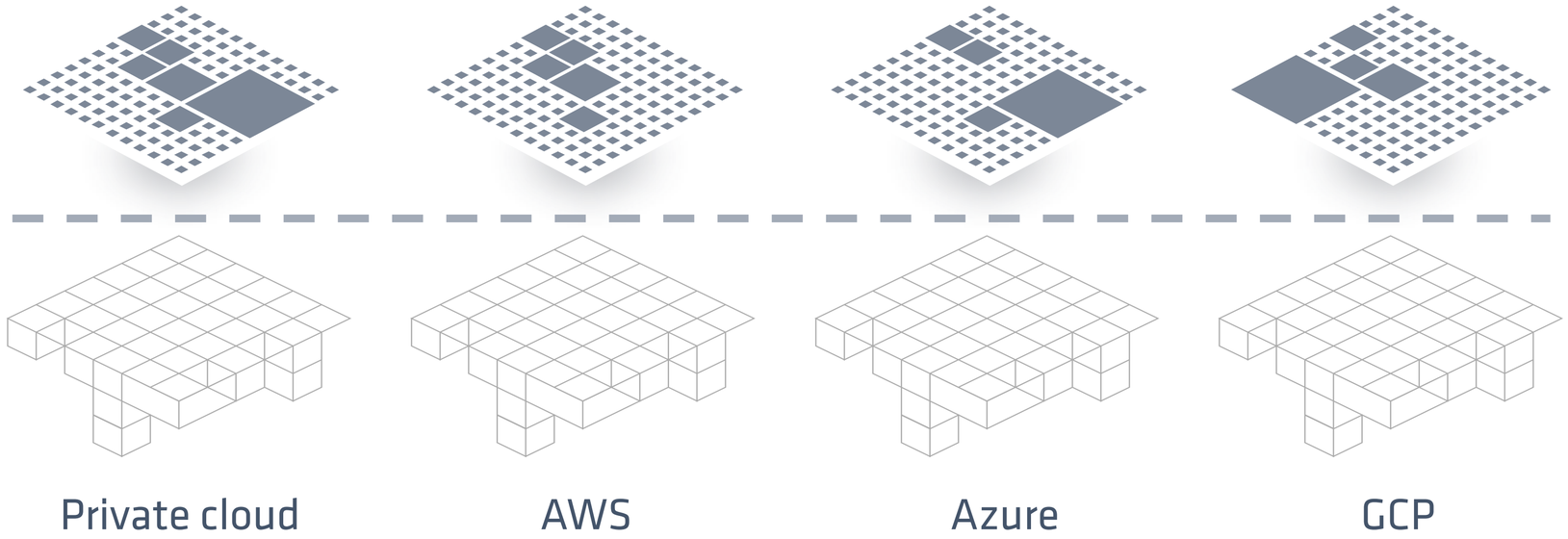
Static and Consolidated

Perimeter-based security was sufficient for private datacenters



Hybrid, Dynamic and Distributed

How does security extend across multiple data centers/clouds?

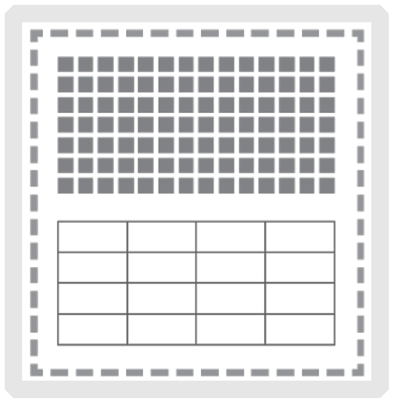


Managing Secrets in Distributed Infrastructure

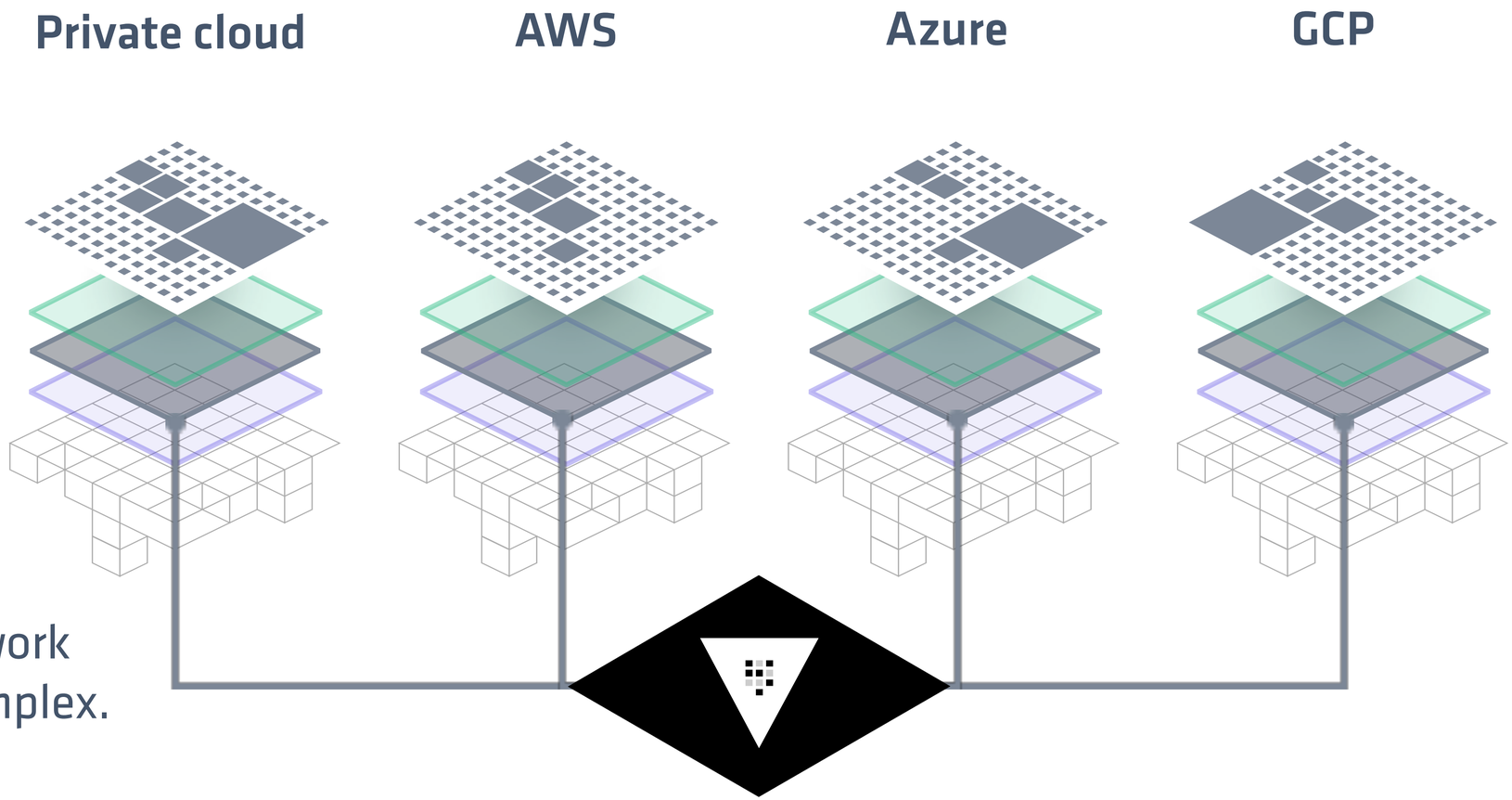
- How can we **centralize** secrets?
- How can we manage the **lifecycle** of a secret?
- How can we **audit** secret access?
- How can we manage **access** to secrets?
- How can we securely distribute secrets **across hybrid environments**?
- How can we **mitigate** a compromised secret?

- A **secret** is:
  - Any *sensitive* information that if acquired by unauthorized party would cause harm to an organization
  - Anything stored or returned by Vault that contains ***confidential*** or ***cryptographic*** material

Secure Infrastructure using Vault

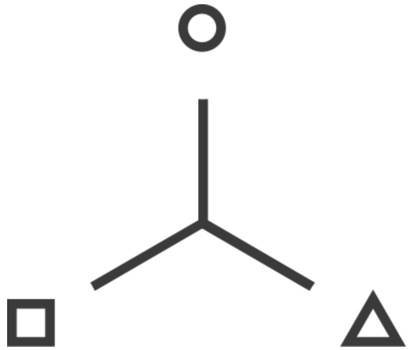


With each new cloud, network topologies become more complex.



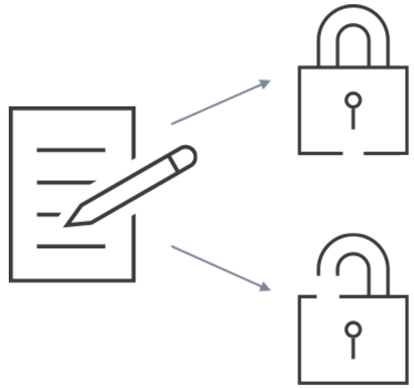
Vault Objectives

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



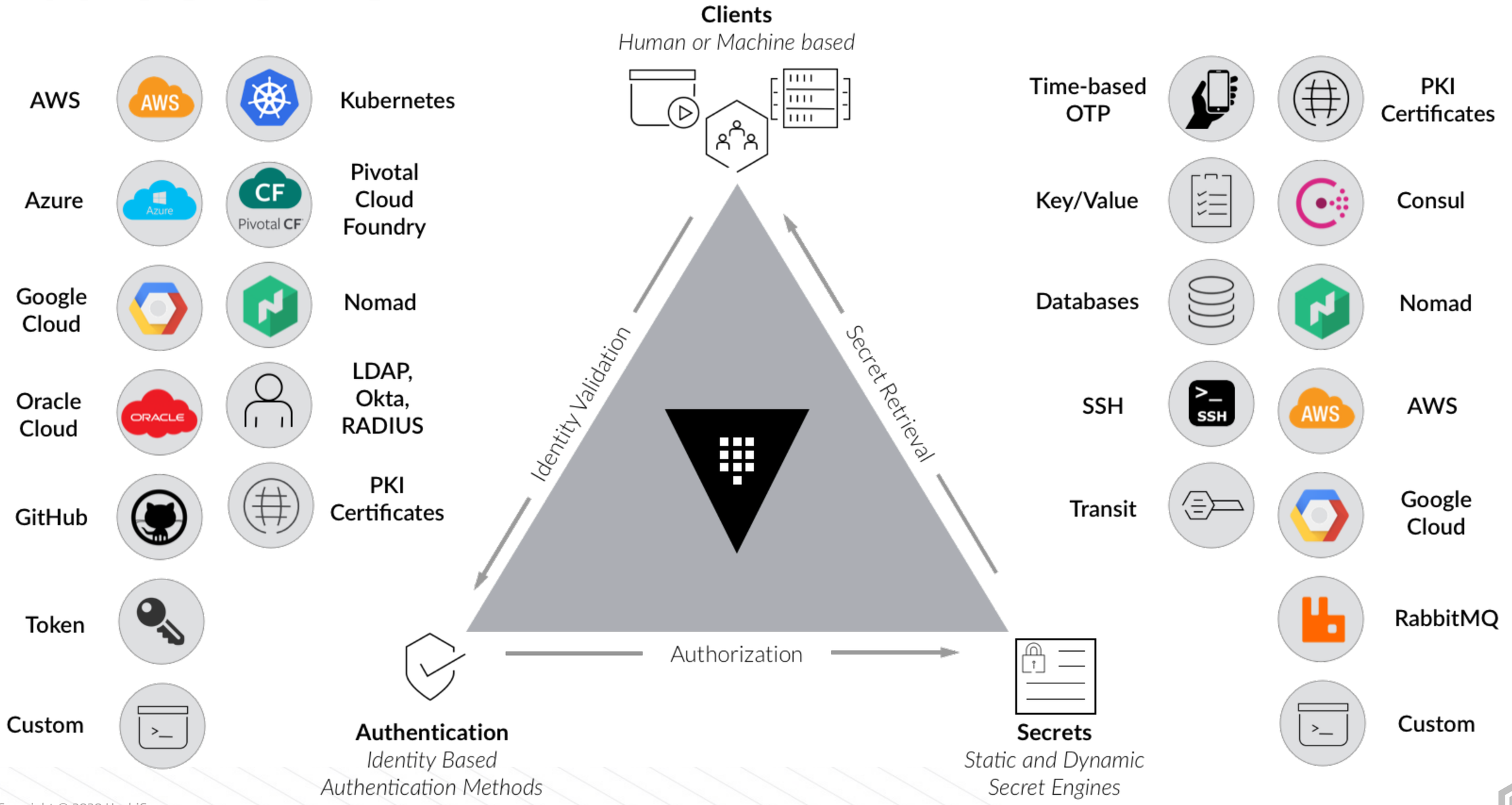
Eliminate Secret Sprawl

Securely Store Any Secret



Provide Secret Governance

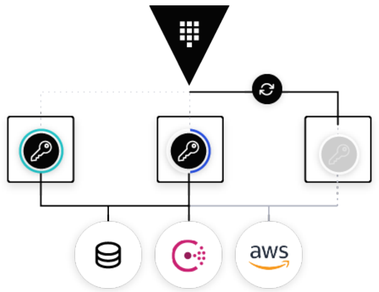
Vault Overview



Use Cases

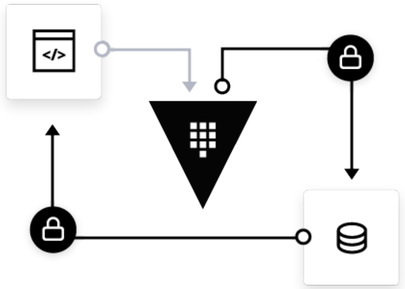
Secrets Management

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



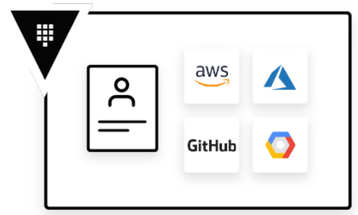
Encrypting Application Data

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



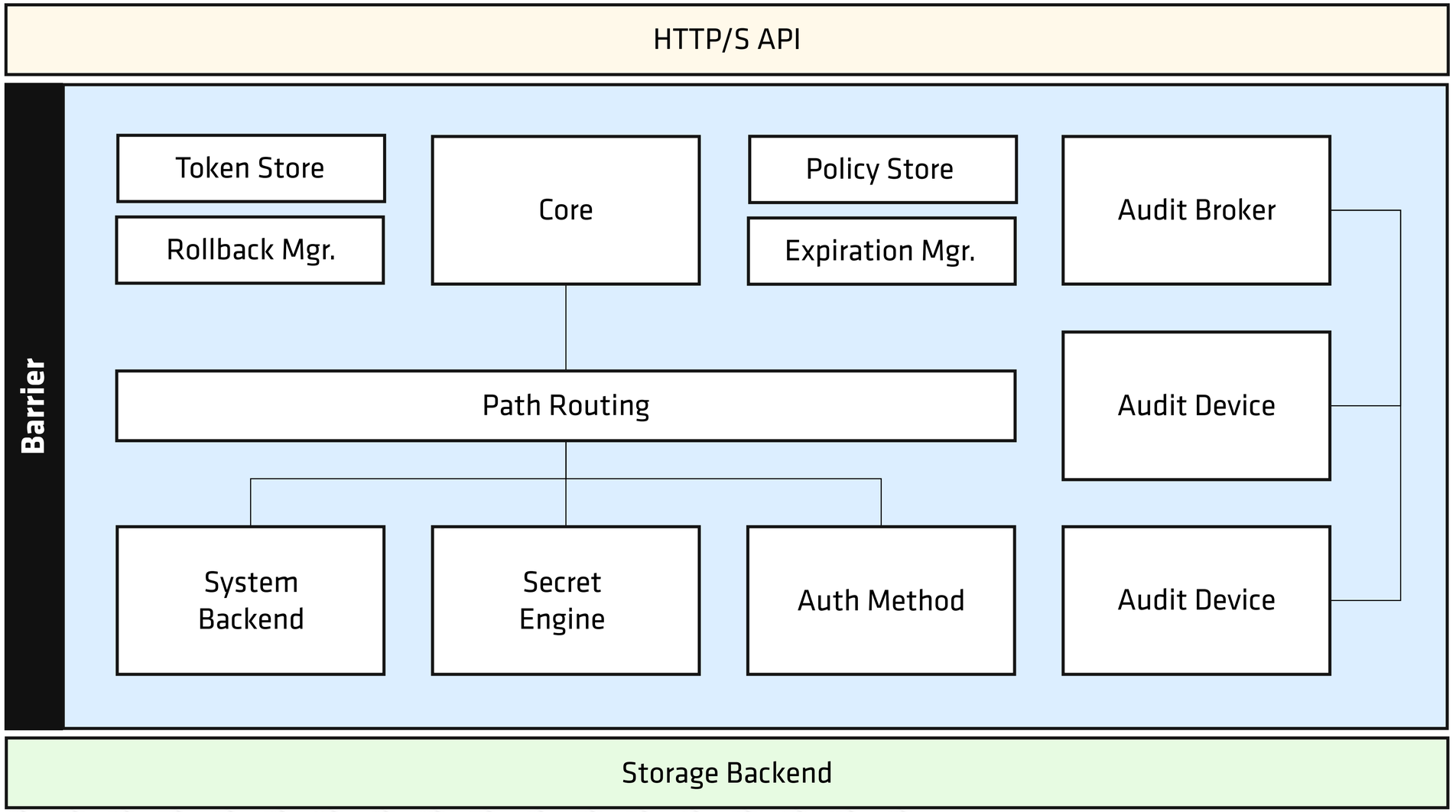
Identity-based Access

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



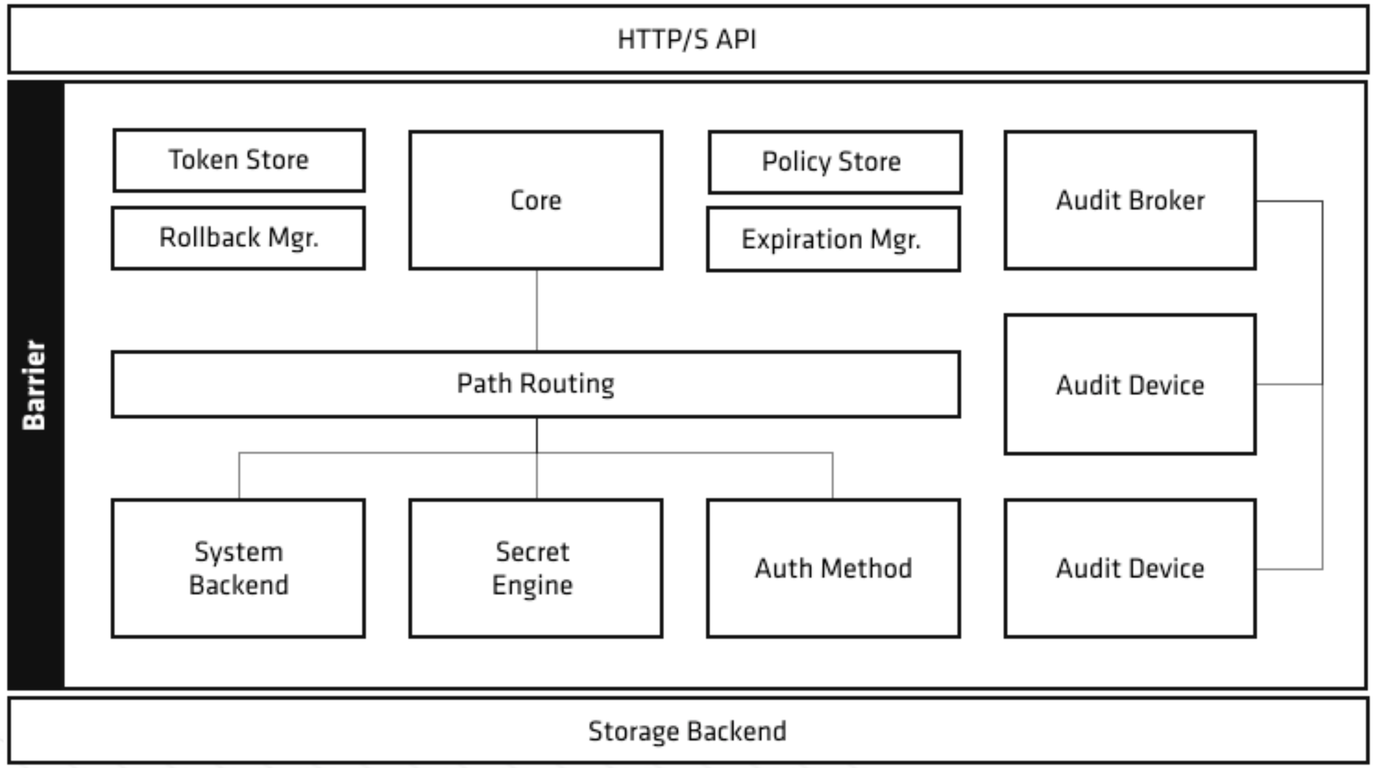
Vault Architecture & Terminology

Architecture



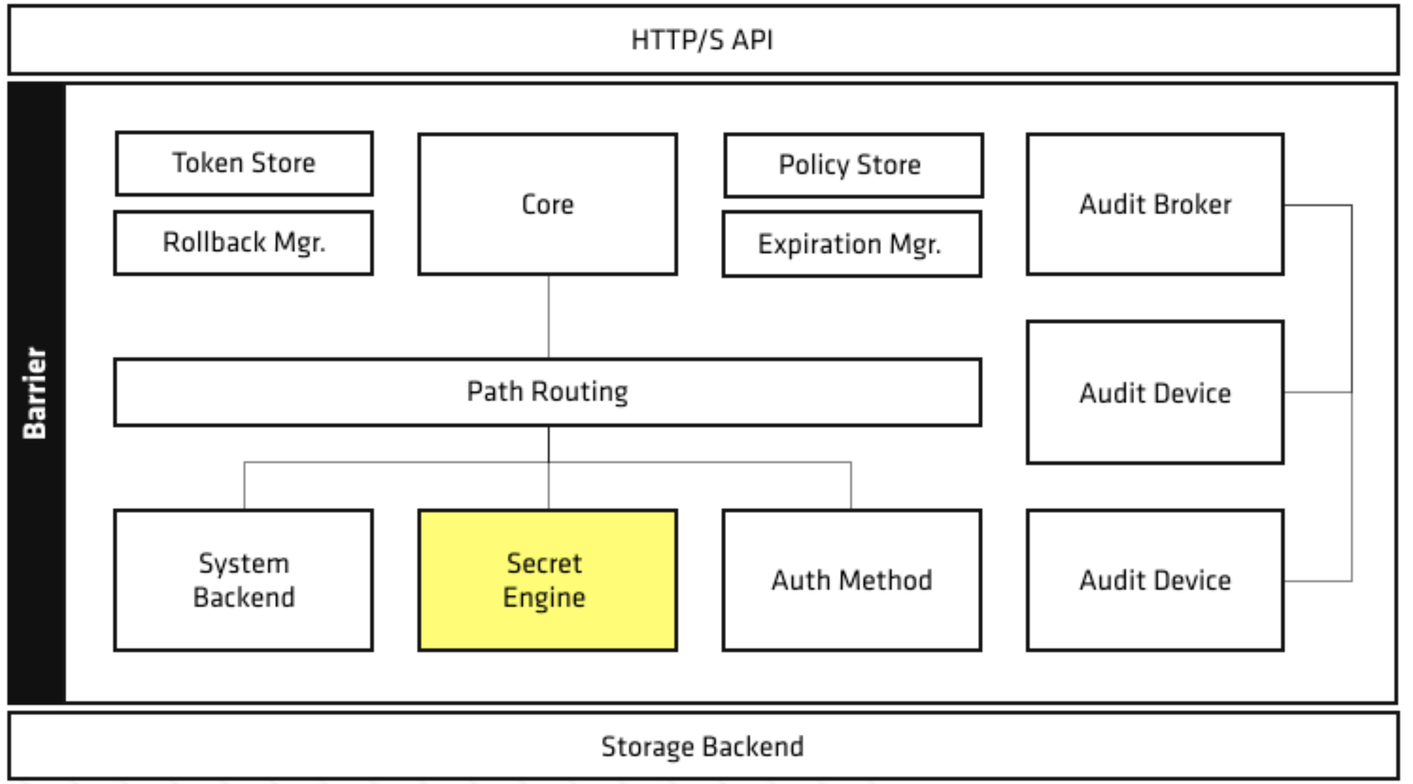
Barrier

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



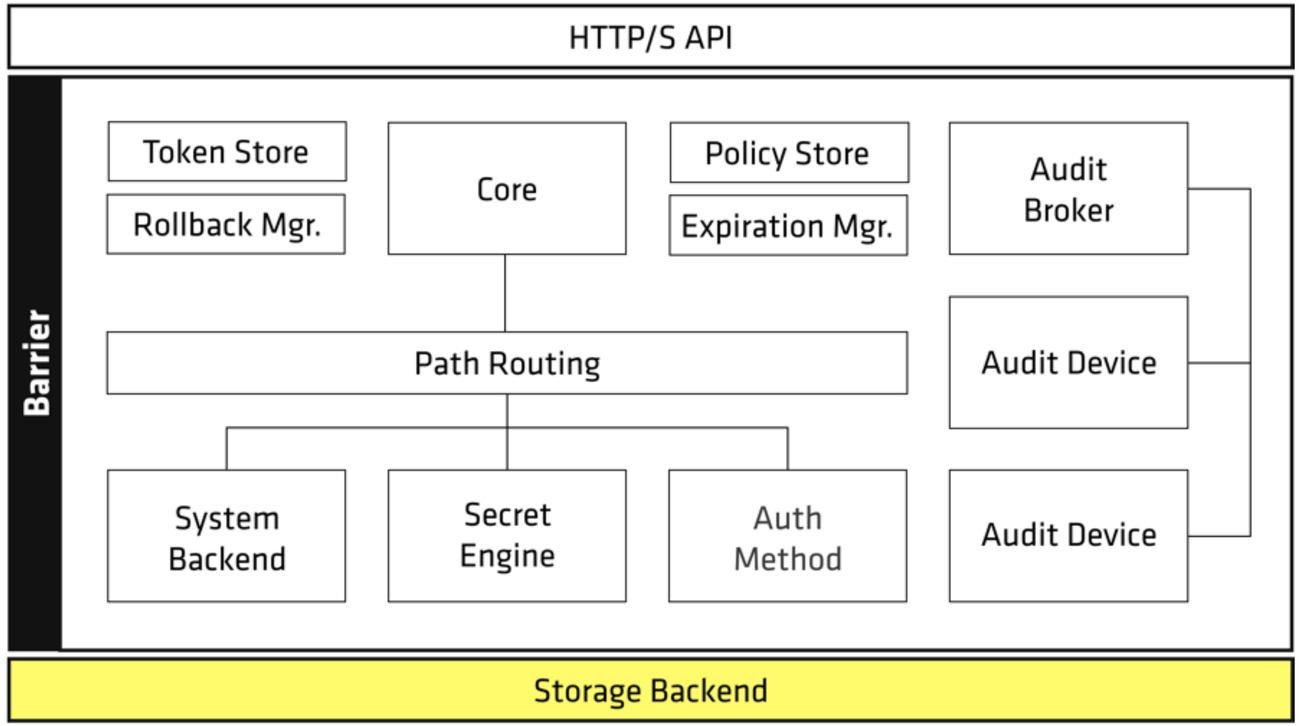
Secret Engine

- 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**



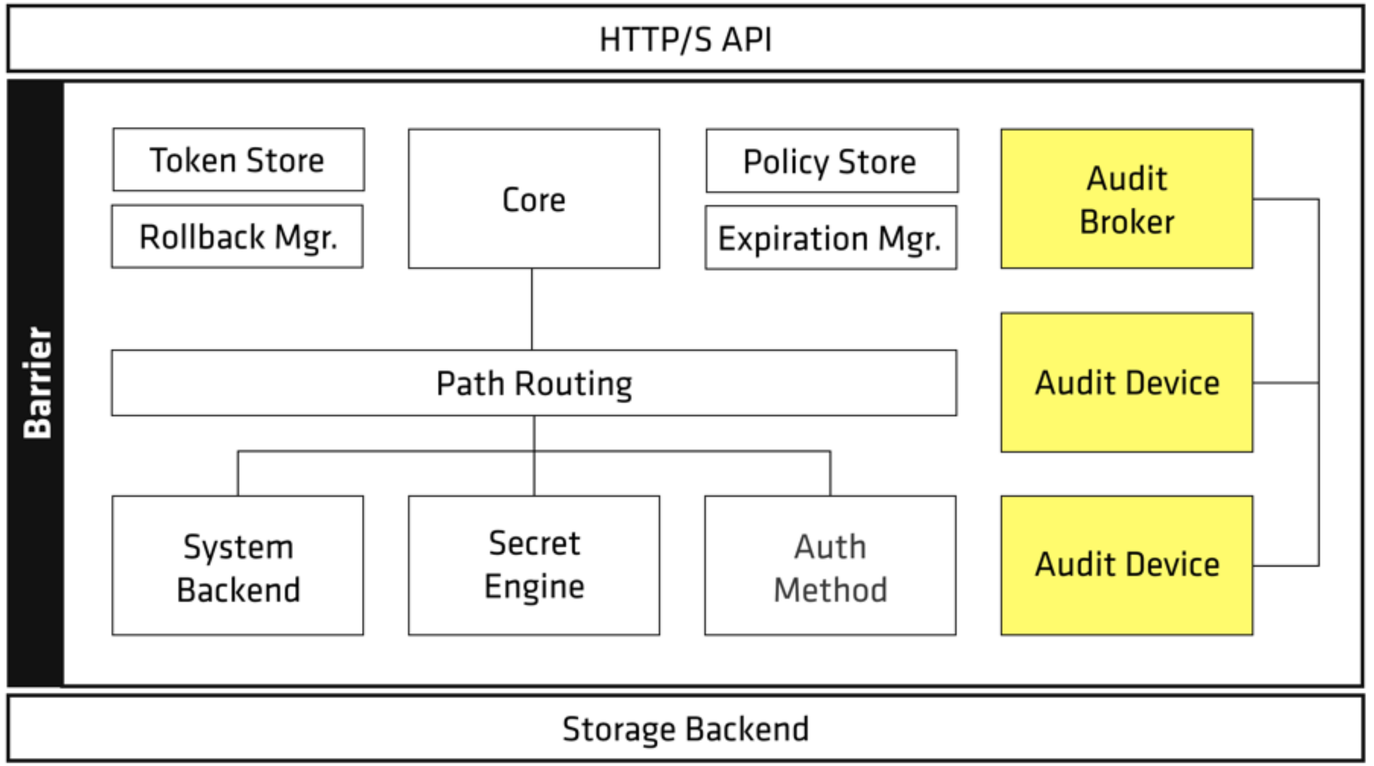
Storage Backend

- 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*, *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, etc.
  - **Operator-oriented:** Github, LDAP, username & password, etc.

Vault Server

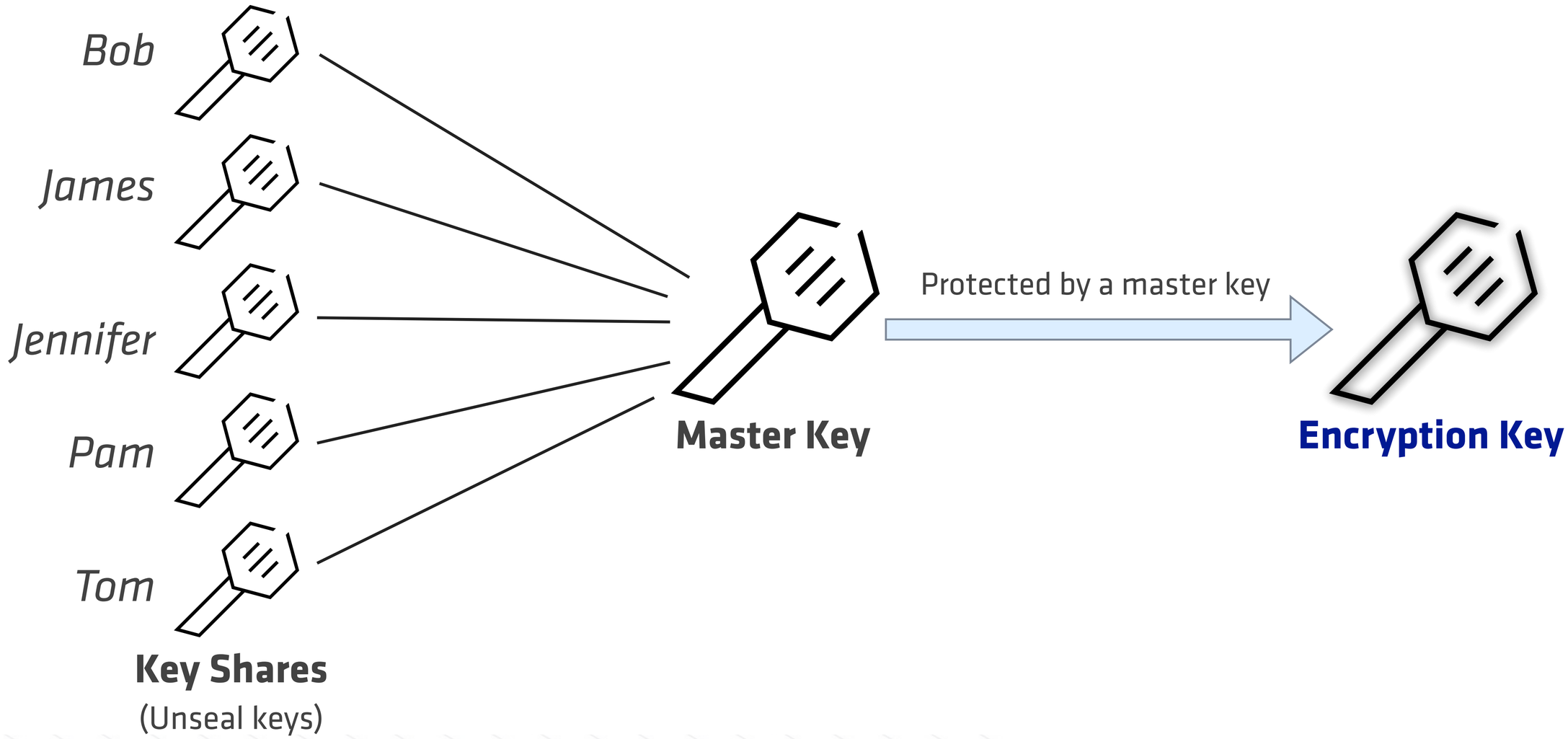
Server

- 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

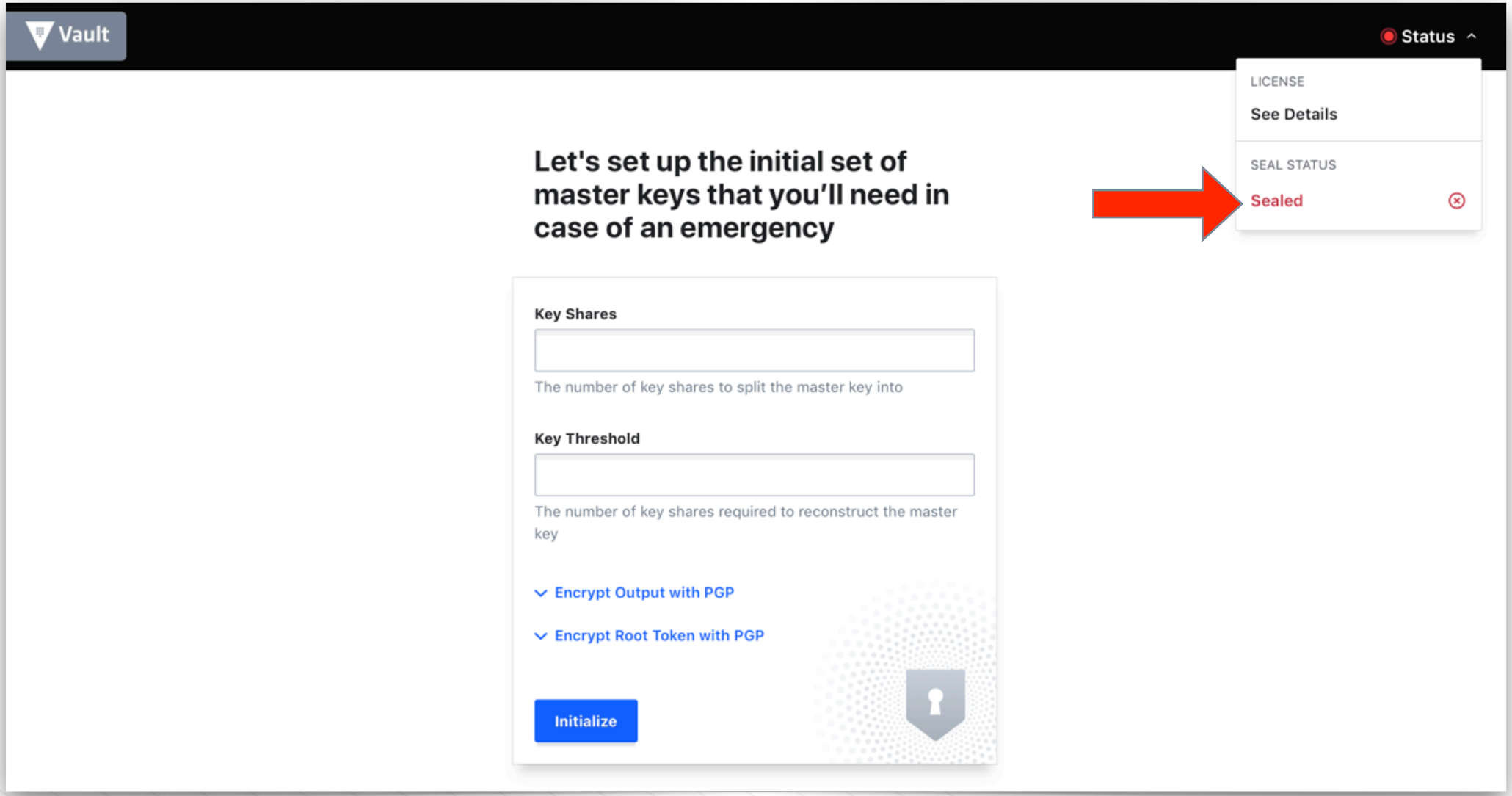
Seal / Unseal

- 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 NOT stored anywhere

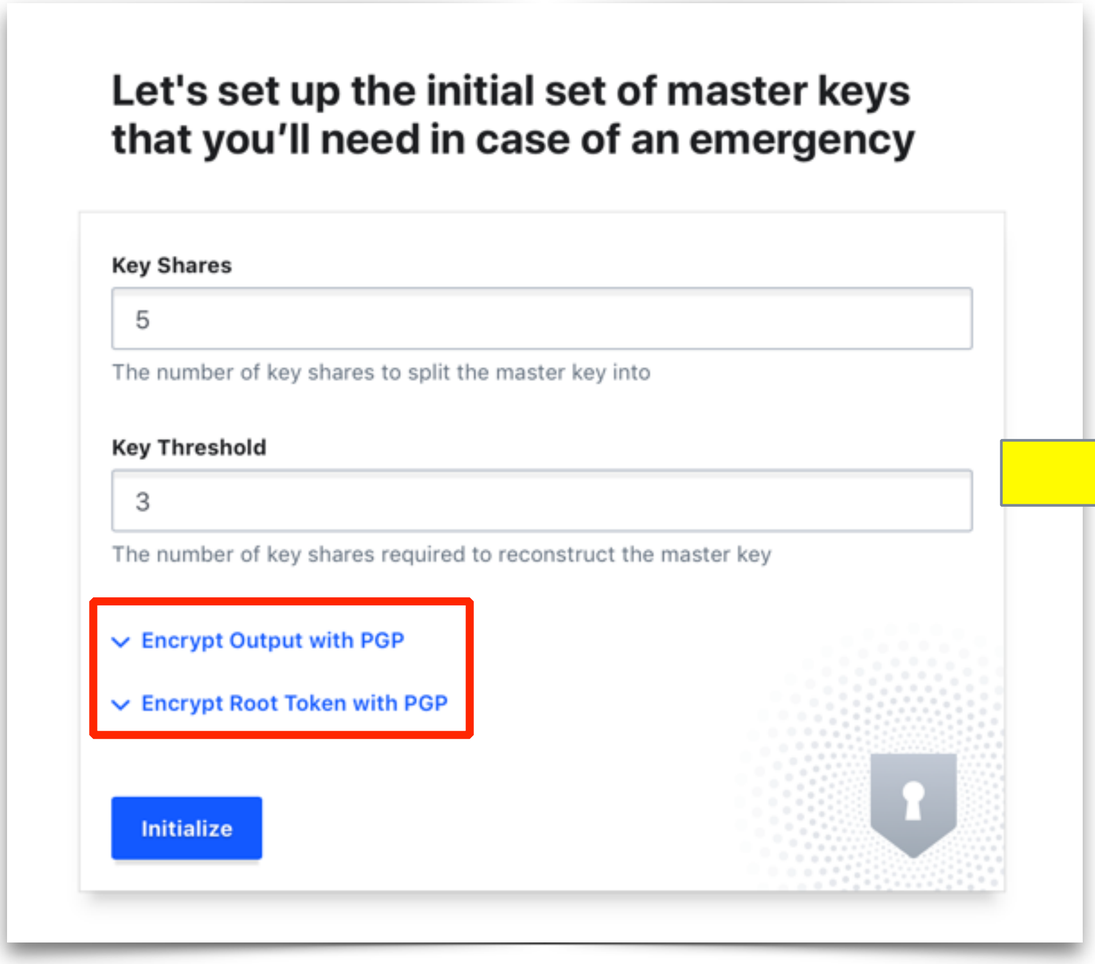
Shamir's Secret Sharing

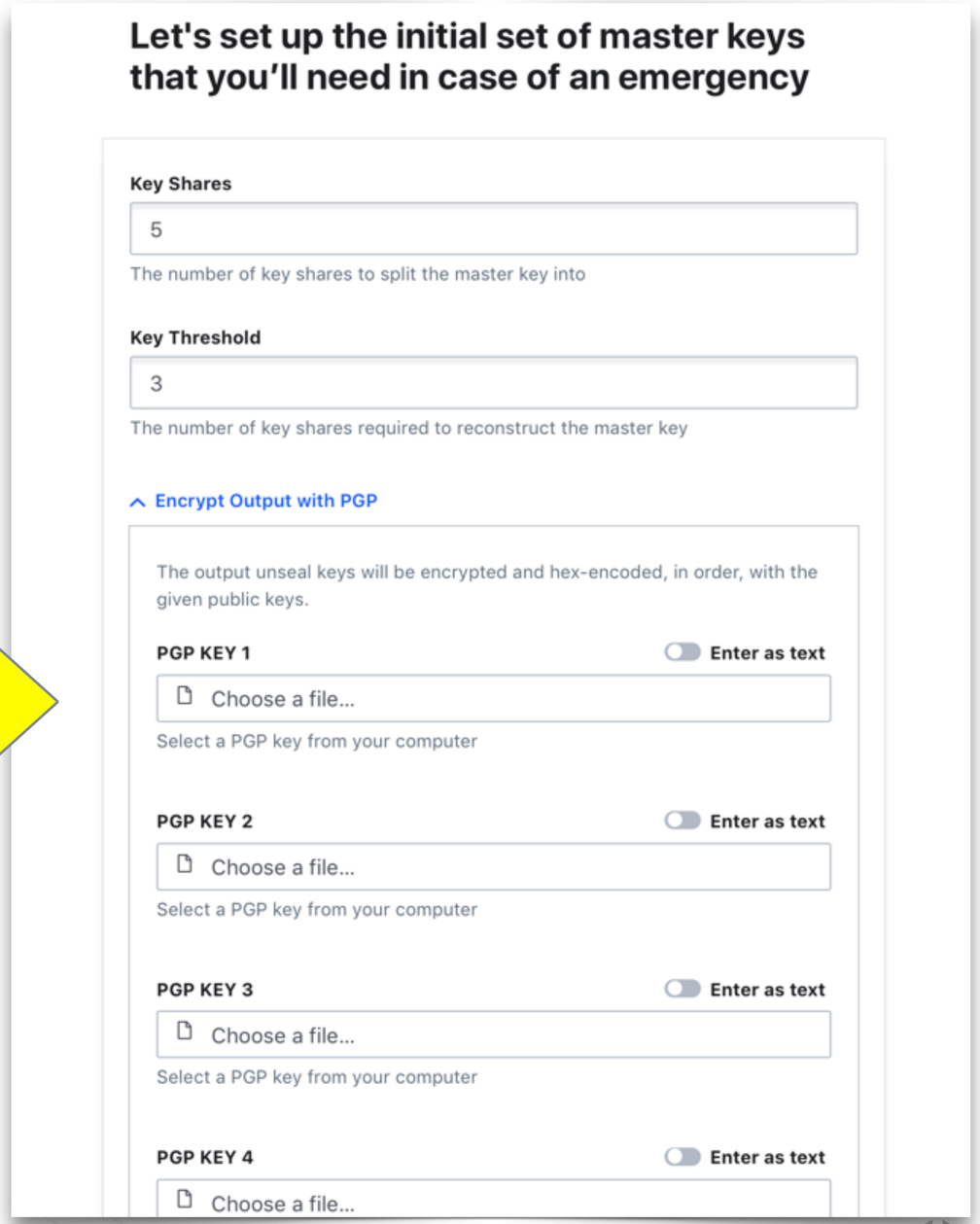


Initializing Vault (1 of 2)

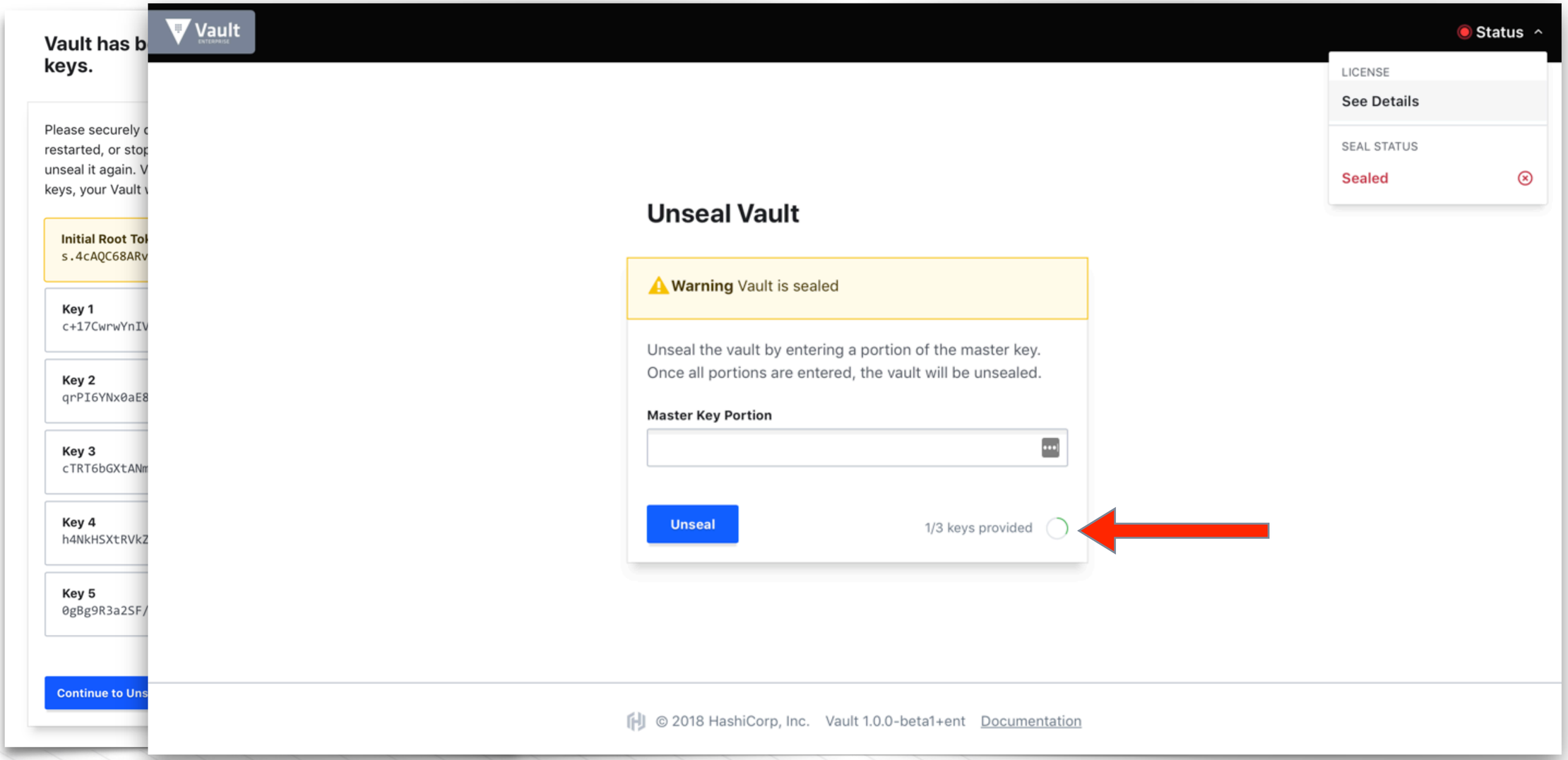


Initializing Vault (2 of 2)



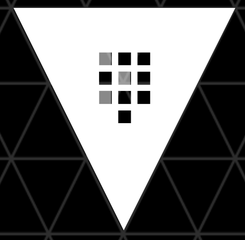


Unsealing Vault



"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

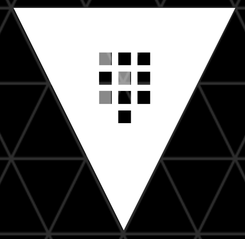
Terminal

```
$ 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'
```

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": "6b6e015f-4a29-4d68-3ca3-e5de75550db4",
    "operation": "update",
    "client_token": "hmac-sha256:4704888f637a42b2b3299d946d0e67c6746...",
    "client_token_accessor": "hmac-sha256:4dcbd61db61a0bdcbd44fe5c3f...",
    "namespace": {
      "id": "root",
      "path": ""
    },
    "path": "secret/data/employee/james",
    "data": {
      "data": {
        "employeeID": "hmac-sha256:22539b17be8d3c944f5303dbbf1d8..."
      },
     ...
```

Request and response messages are logged including API endpoint, request data, etc.

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

Interacting with Vault

Interacting with Vault (1 of 2)

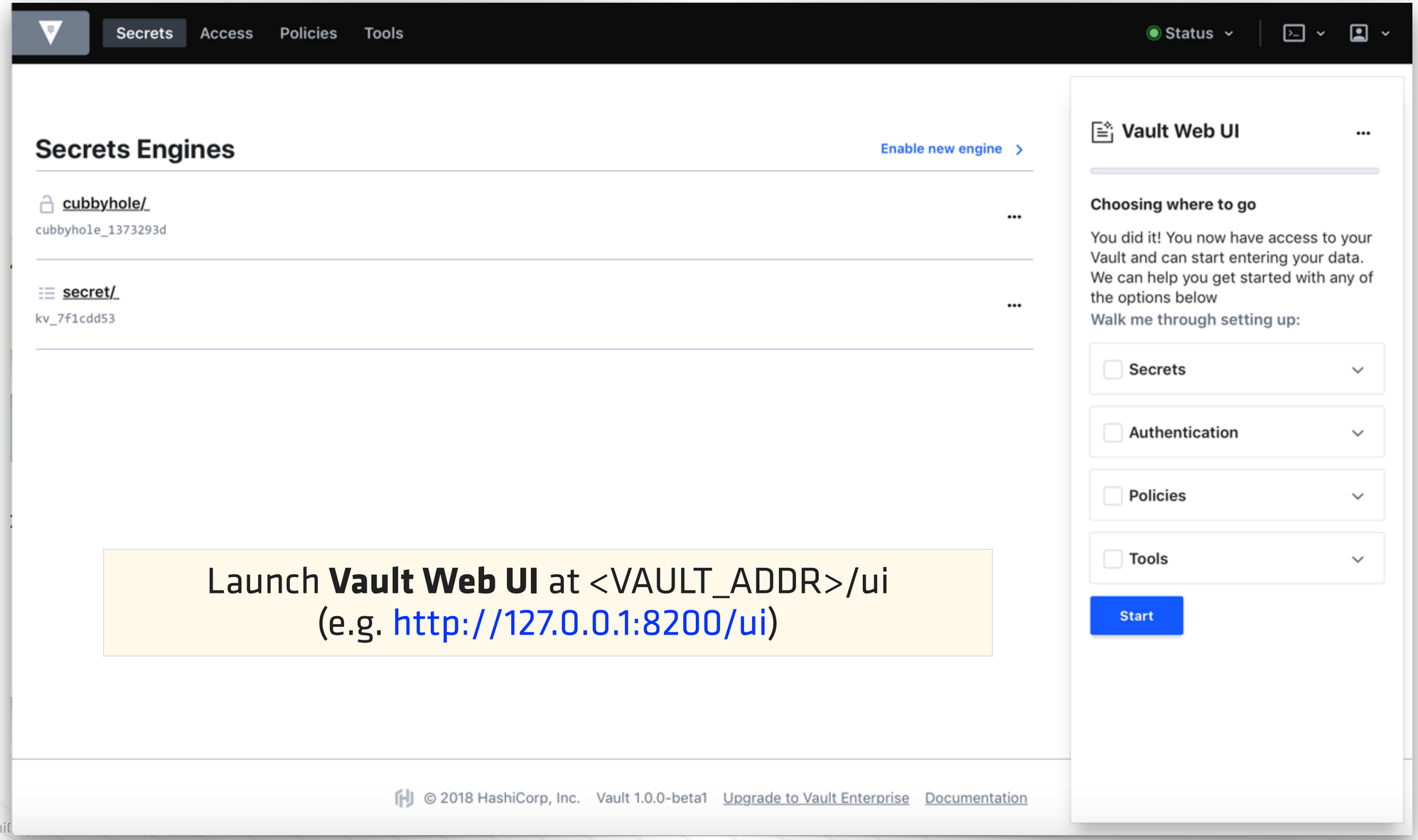
- Vault **HTTP API**
  - Full access to Vault via HTTP
  - Every aspect of Vault can be controlled via API

```
# 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)

Vault CLI

- Vault CLI provides convenient wrapper around Vault API
  - Every CLI command maps to the API
  - Each CLI command is represented as a **command** or **subcommand**

```
$ vault <command> [options] [path] [args]
```

- Get help with "**`-h`**" or "**`help`**" flag

```
$ vault -h
$ vault <command> -h
```

Learning Vault API

- Easier to learn the CLI first; however, apps typically invoke Vault API
  - Get help for an API endpoint with "**`path-help`**" command

```
$ vault path-help <endpoint>
```

  - Use "vault" to directly trigger an API endpoint
    - **`read`**, **`write`**, and **`delete`** maps to **GET**, **POST**, **PUT**, **DELETE** HTTP verbs

```
$ vault write <endpoint> [args]
```

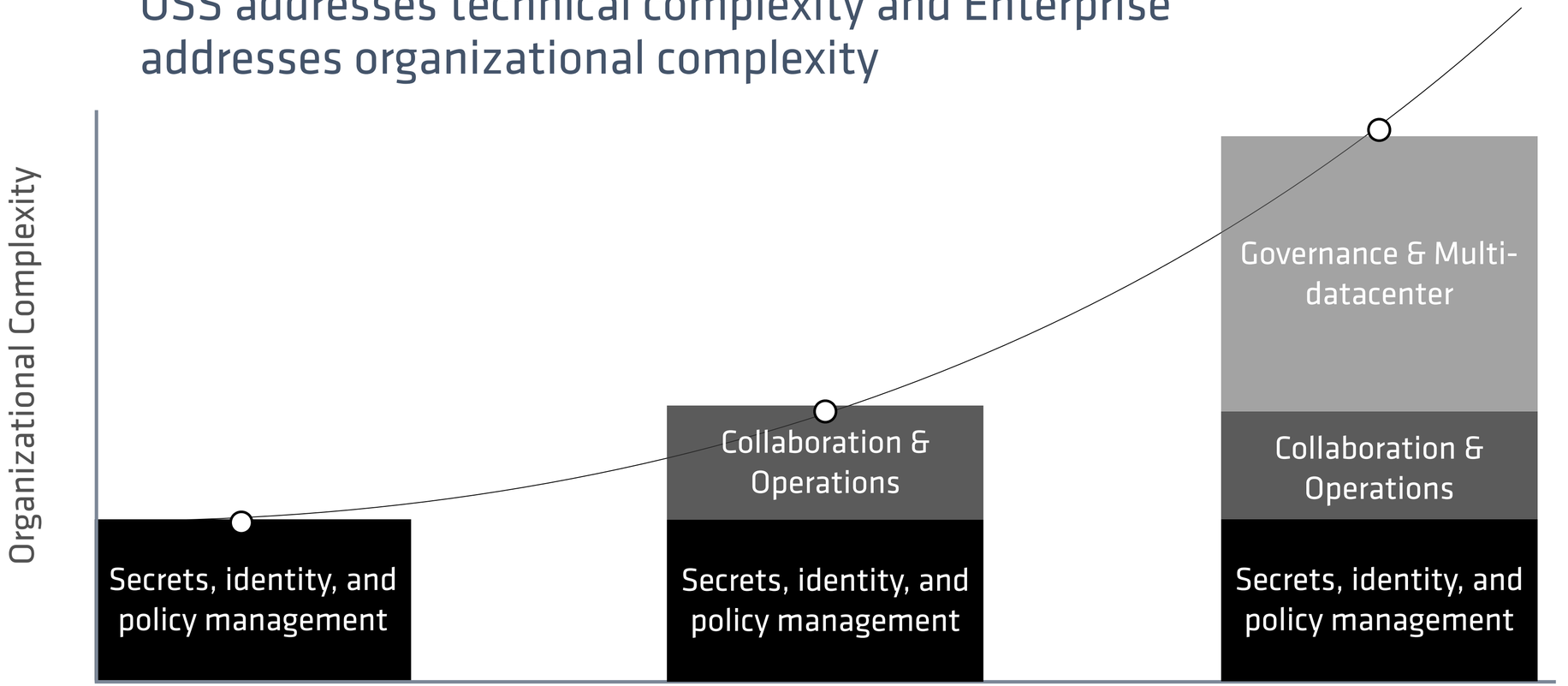
  - Use **`-output-curl-string`** flag to print equivalent API call for a CLI command

```
$ vault kv get -output-curl-string secret/apikey/google

  curl -X GET -H "X-Vault-Token: $(vault print token)"
     http://127.0.0.1:8200/v1/secret/data/apikey/google
```

OSS vs. Enterprise

Packaging

OSS addresses technical complexity and Enterprise addresses organizational complexity



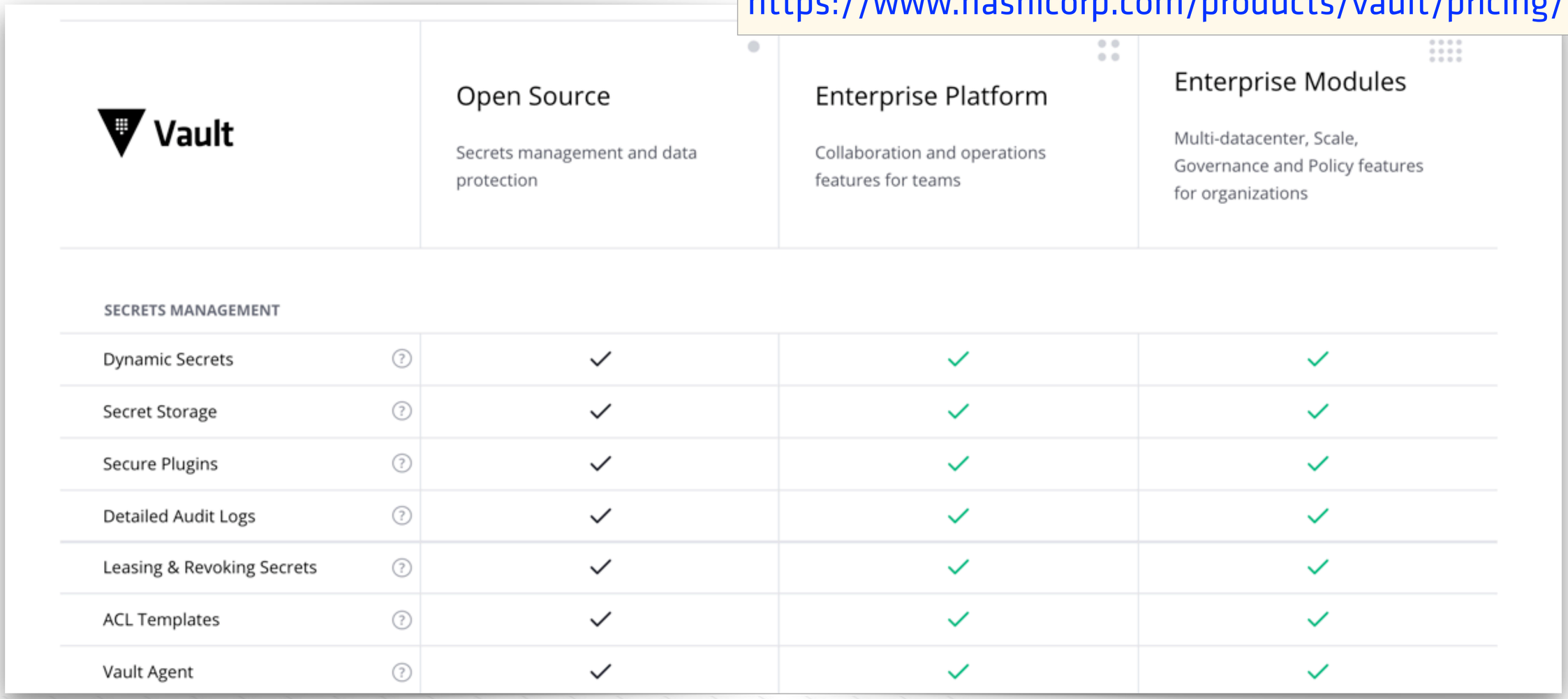
Individuals use Vault **Open Source**

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

Knowledge Check Questions

1. Where does Vault store secrets? Storage Backend (e.g. Consul, file, MySQL, etc.)

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

3. What's the benefit of Shamir's Secret Sharing algorithm?

   No single person holds the key to the Vault (your secrets)!

Lab 1: Lab Setup

10 minutes

- Open the Lab Book PDF

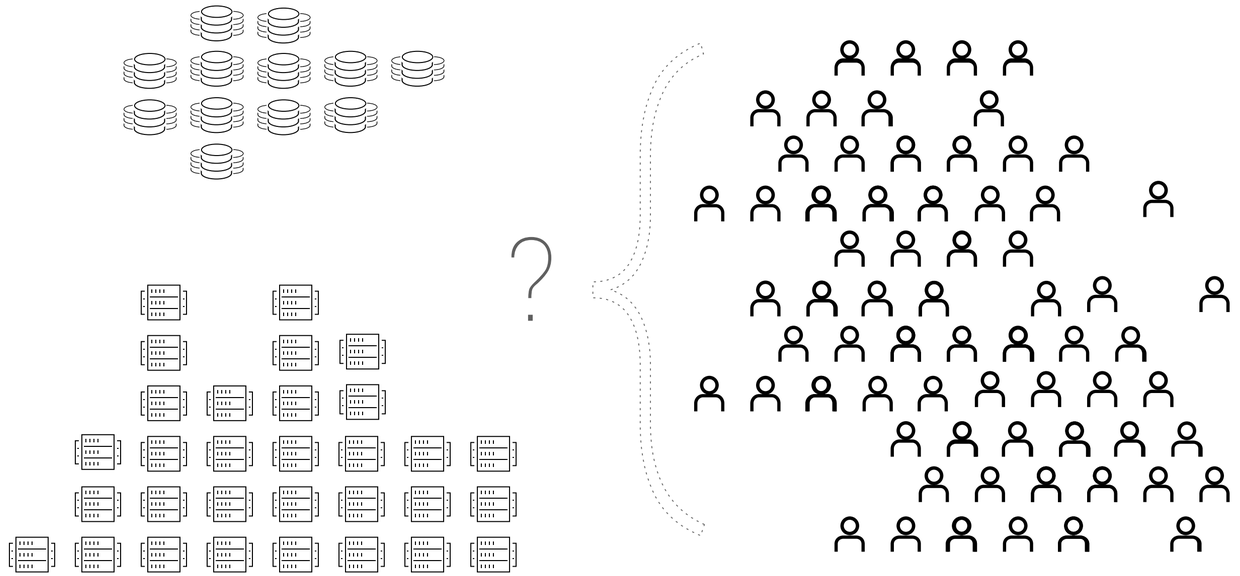
- In this lab, you are going to:
  - Task 1: Connect to the Student Workstation
  - Task 2: Getting Help
  - Task 3: Enable Audit Logging
  - Task 4: Connecting to Vault UI

Check your email for your student workstation information!

Supplemental - Vault Use Case Scenarios

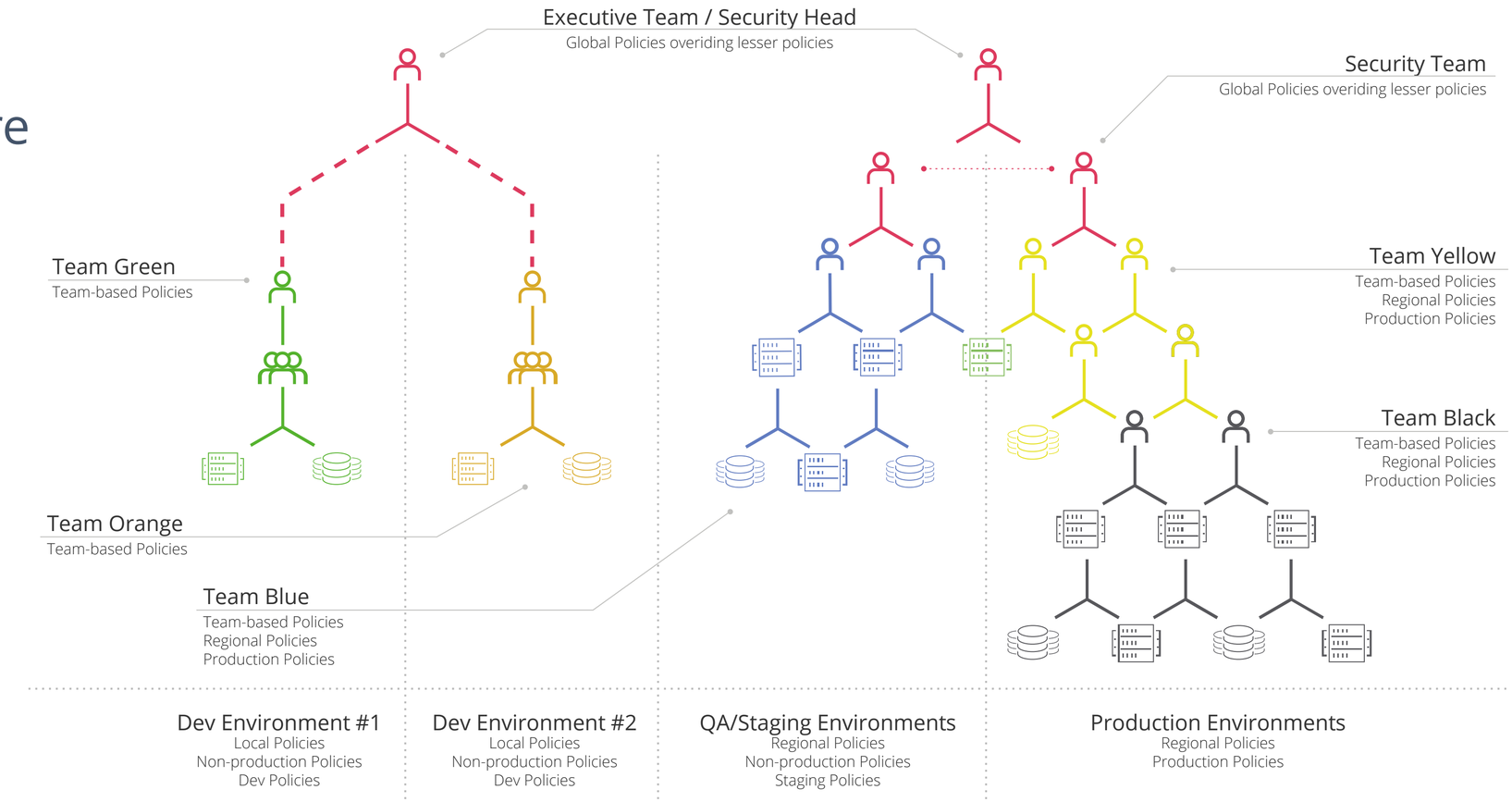
Use Case — Secret Exposure

ACME, Inc. has several different divisions of their company, with separate needs to isolate secrets and information by team, but they're struggling to keep pace with growth and the company's sensitive information keeps spreading due to everyone sharing credentials or credentials not having a clear separation of roles and policies.



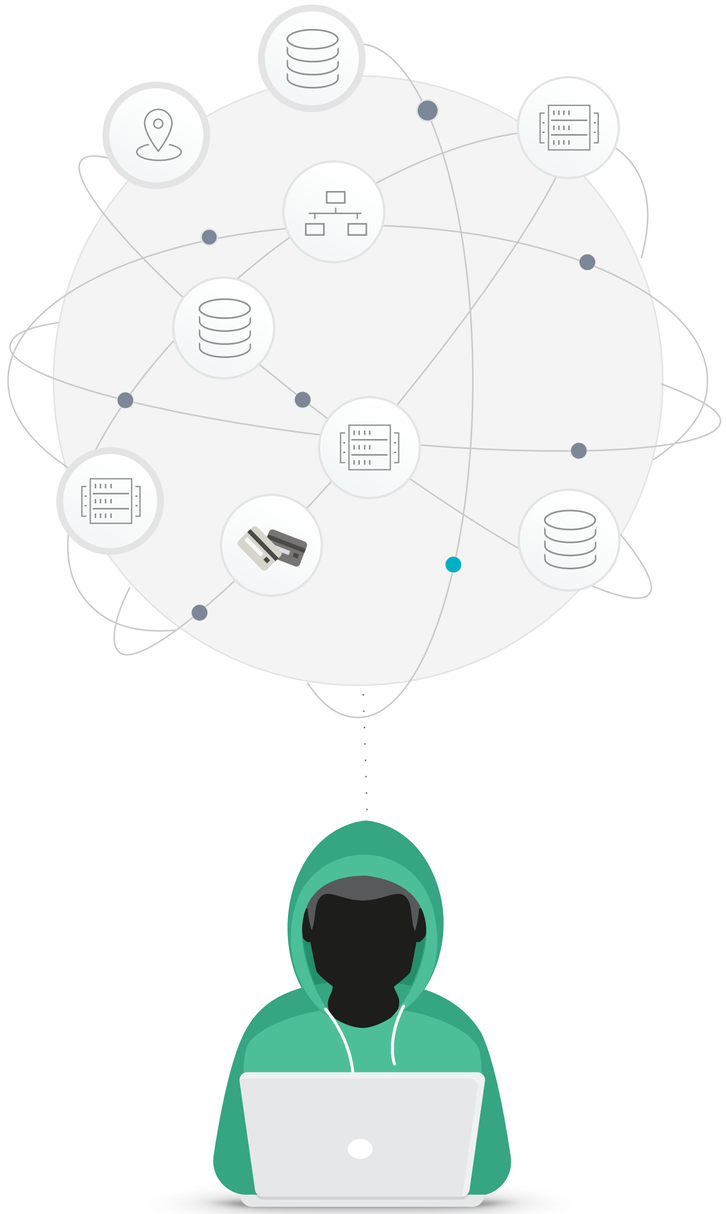
Use Case — Secret Exposure (Solved with Vault)

With Vault, the CISO, executive teams and the security teams can ensure that teams can use the same credentials across their infrastructure, but use secrets isolated to their organization, team, etc., via Identity Groups and ACL Policies



Use Case — Access Exposure

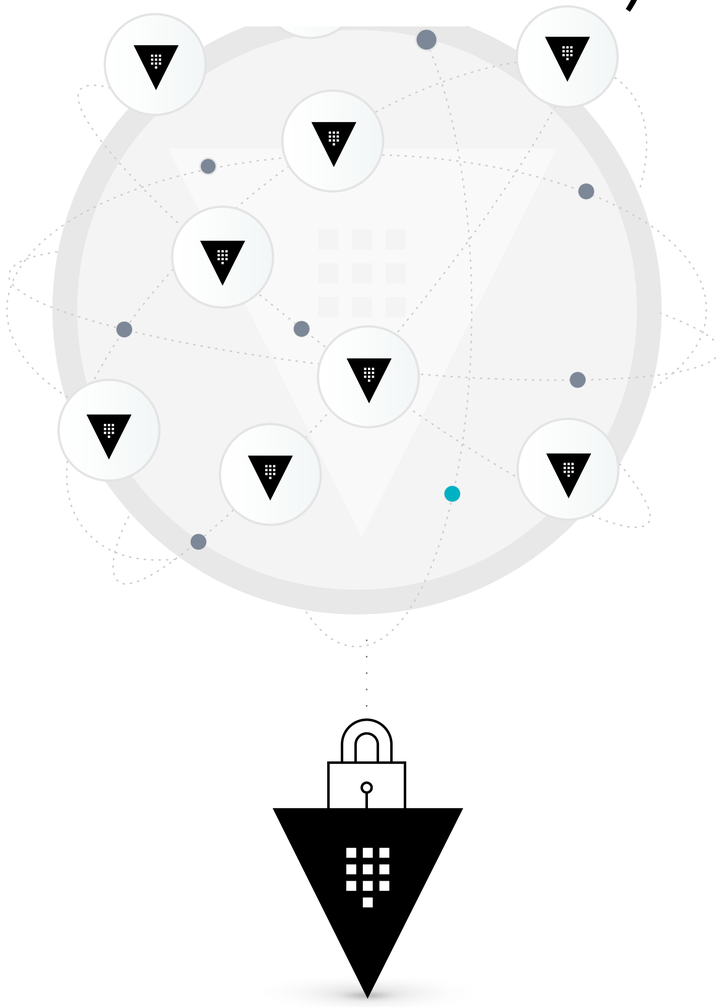
An ACME, Inc. employee with admin privileges has their company laptop stolen from a local coffee shop. The thief uses the laptop and steals tokens and stored authorizations to access company systems. ACME has to go through all different systems and end points to manually remove the employee's authorizations and access, during which the thief steals valuable company information.



Use Case — Access Exposure (Solved with Vault)

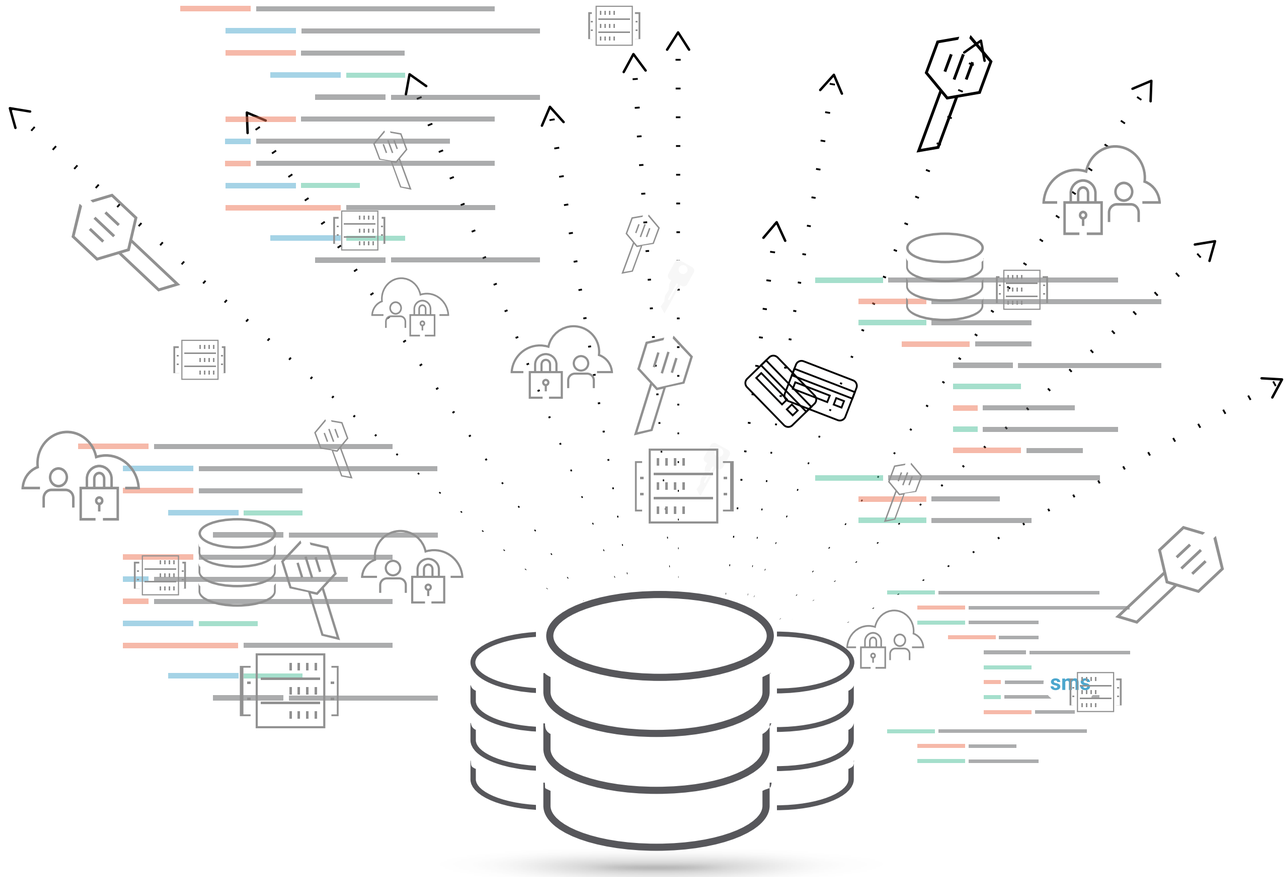
Whether monitoring audit logs to see if systems are being accessed inappropriately or breaches are reported, Vault can be sealed immediately to revoke all access and invalidate credentials across an organization. Similarly, tokens can be cancelled and/or rerolled.

Once the account(s) have been scrubbed, secrets, tokens, and access can be turned back on, rerolled, and redistributed minimizing access and the damage done from outside influencers.



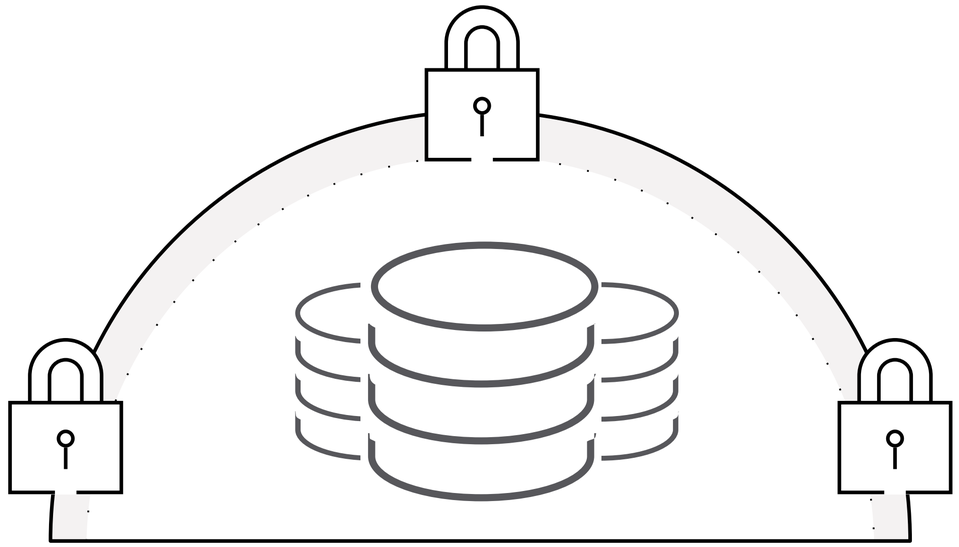
Use Case — Unencrypted Data

ACME, Inc. recently made headlines for a massive data breach which exposed millions of their users payment card accounts online. When they tracked down the problem they found that a new HVAC system with management software had been put into their data centers and had created vulnerabilities in their networks and exposed ports and IPs to the databases publicly.



Use Case — Unencrypted Data (Solved with Vault)

Intrusions happen. They're more common than ever with the move to the cloud. Vault provides Encryption as a Service (EaaS) to enable security teams to fortify data during transit and at rest. So even if an intrusion occurs, your data is encrypted with AES 256-bit GCM (TLS in transit). Even if an attacker were able to access the raw data, they would only have encrypted bits. This means attackers would need to compromise multiple systems before exfiltrating data.

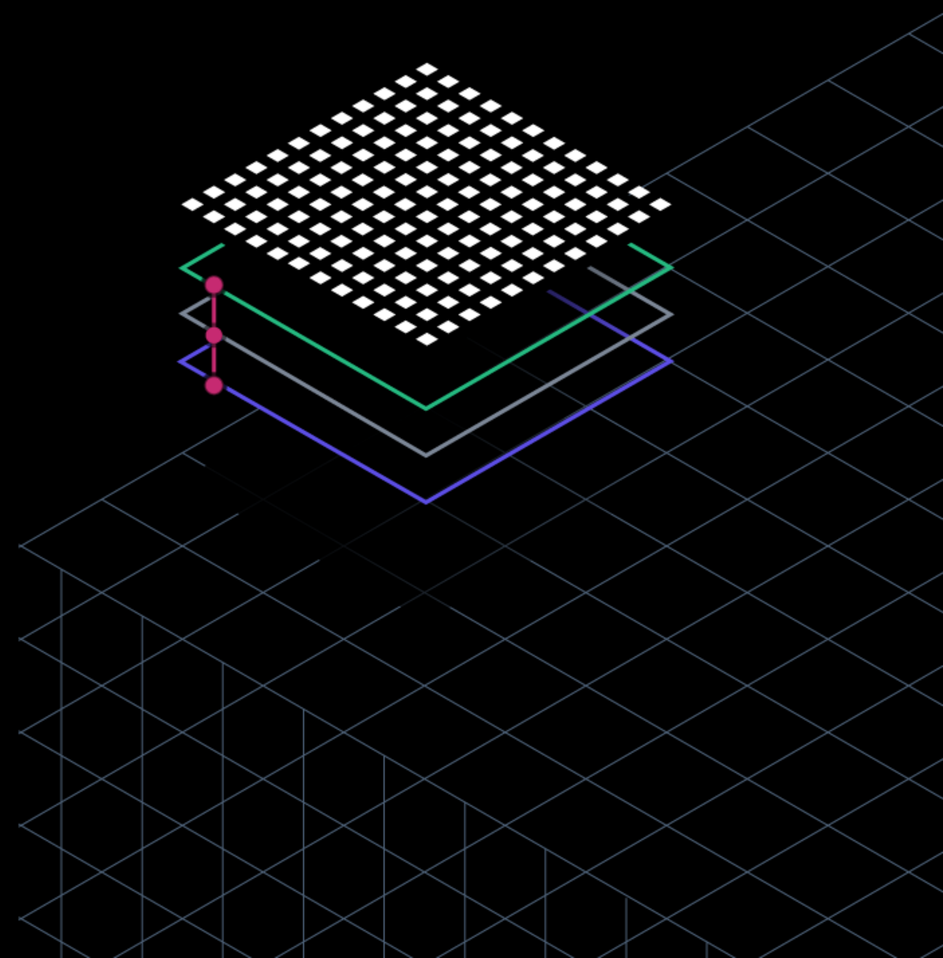


Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp



Secrets Engines

Static Secrets

Agenda

- Secrets Engines
  - What are secrets engines?
  - Secrets engines supported out-of-the-box
  - Secrets engine lifecycle
- Managing Secrets Engines
- Key/Value Secrets Engines
  - Storing sensitive data

Secrets Engines

What are Secrets Engines?

- Secrets engines are components responsible for managing **secrets**
  - **Secrets** are pieces of sensitive information that can be used to access infrastructure, resources, data, etc.
  - Some secrets engines simply **store** and **read** data
    - Like encrypted Redis/Memcached
  - Some connect to other services and **generate dynamic credentials** on-demand
  - Others provide encryption as a service (EaaS), TOTP generation, certificates, etc.

Secrets Engines (1 of 3)

Secrets Engine	Description
Cubbyhole	Stores arbitrary secrets within the configured physical storage for Vault namespaces to a **token**. Paths are scoped per token.
Key/Value	Stores arbitrary secrets within the configured physical storage for Vault (a.k.a. generic secret)
Active Directory	Rotates AD password dynamically (designed for a high-load environment)
AliCloud	Generates AliCloud access tokens based on RAM policies, or AliCloud STS credentials based on RAM roles
AWS	Generates AWS access credentials dynamically based on IAM policies
Azure	Generates Azure service principals and role assignments
Consul	Generates Consul API tokens dynamically based on Consul ACL policies

Secrets Engines (2 of 3)

Secrets Engine	Description
Databases	Generates database credentials dynamically based on configured roles Database specific plugins: Cassandra, Elasticsearch, HanaDB, InfluxD, MongoDB, MSSQL, MySQL/MariaDB, PostgreSQL, Oracle, Redshift and custom.
Google Cloud	Generates Google Cloud service account keys and OAuth tokens based on IAM policies
GCP Cloud KMS	Provides encryption and key management via Google Cloud KMS
KMIP	[Vault Enterprise] Vault to behave as a KMIP server provider
Identity	Identity management solution for Vault. Internally maintains the clients that are recognized by Vault
Nomad	Generates Nomad API tokens dynamically based on pre-existing Nomad ACL policies
PKI	Generates dynamic X.509 certificates

Secrets Engines (3 of 3)

Secrets Engine	Description
OpenLDAP	Manage LDAP entry passwords
RabbitMQ	Generates user credentials dynamically based on configured permissions and virtual hosts
SSH	Provides secure authentication and authorization for access to machines via the SSH protocol
TOTP	Generates time-based credentials according to the Time-based One-time Password (TOTP) standard
Transform	[Vault Enterprise] Handles secure data transformation and tokenization against provided input
Transit	Handles cryptographic functions on data in-transit
Venafi	Provides SSL/TLS certificates using Venafi Trust Protection Platform or Venafi Cloud

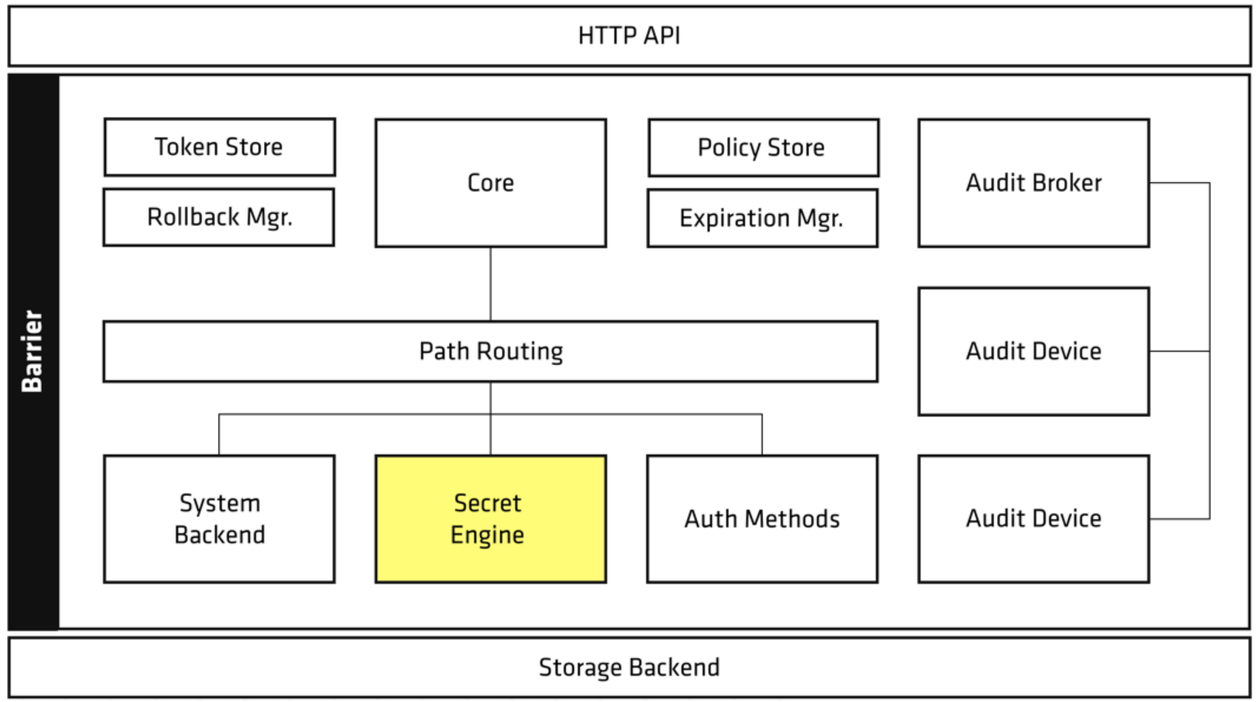
Secrets Engine Lifecycle

- Secrets engines must be enabled at a **path** so that the request can be routed
  - Each secrets engine defines its own paths and properties
  - **Cubbyhole** secrets engine is enabled by default at **`cubbyhole/`**
  - Key/Value v2 secrets engine is enabled at **`secret/`** *if running in **`dev`** mode*

Operation	Description
Enable	Enables a secrets engine at a given path (e.g. "`aws/`", "`database/`")
Disable	Disables an existing secrets engine
Move	Moves the path for an existing secrets engine
Tune	Tunes global configuration for the secrets engine such as its TTLs

Barrier View

- When a secrets engine is enabled, a **random UUID** is generated
- Whenever the engine writes to the physical storage layer, it is prefixed with that UUID folder
- Vault storage does NOT support relative access (e.g. ../)
  - Makes it impossible for other enabled secrets engines to access other folders/data



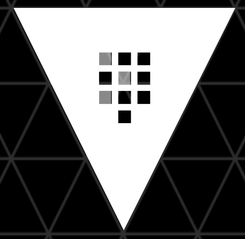
Managing Secrets Engines

Secrets Engine Commands

- Usage: **`vault secrets <subcommand> [options] [args]`**
  - Display detailed help: **`vault secrets -h`**

Subcommand	Description
disable	Disable a secrets engine
enable	Enable a secrets engine
list	List enabled secrets engines
move	Move an already enabled secrets engine to a new path
tune	Tune a secrets engine configuration (e.g. TTLs)

- Most secrets engines need to be ***enabled*** first to interact



CLI Commands

```
vault secrets enable <type>
                    or
vault secrets enable -path=<path> <type>
```

```
# Enables database engine at database/ path
$ vault secrets enable database

# Enables database engine at postgresql/ path
$ vault secrets enable -path=postgresql database

$ vault secrets list

Path            Type          Description
----            ----          -----------
cubbyhole/      cubbyhole     per-token private secret storage
database/       database      n/a
identity/       identity      identity store
postgresql/     database      n/a
secret/         kv            key/value secret storage
...
```

System Reserved Paths

- Secrets engines can be enabled at any path with "**`-path`**" flag; however, there are paths **reserved** by the system:

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

Key/Value Secrets Engine

Challenge: Sensitive Data

- **Challenge 1:** *ACME Inc.* has a platform which accepts and manages payments. This means that the security team must mitigate fraud and secure their **customer's payment data**.

- **Challenge 2:** Operations team is responsible for provisioning environments for Dev, QA, Staging and Production. Today, the cluster configuration information is stored in **plain-text**.

Solution

All these different secrets can be stored in **key/value** secrets engine, and accessed through CLI or programmatically via API.

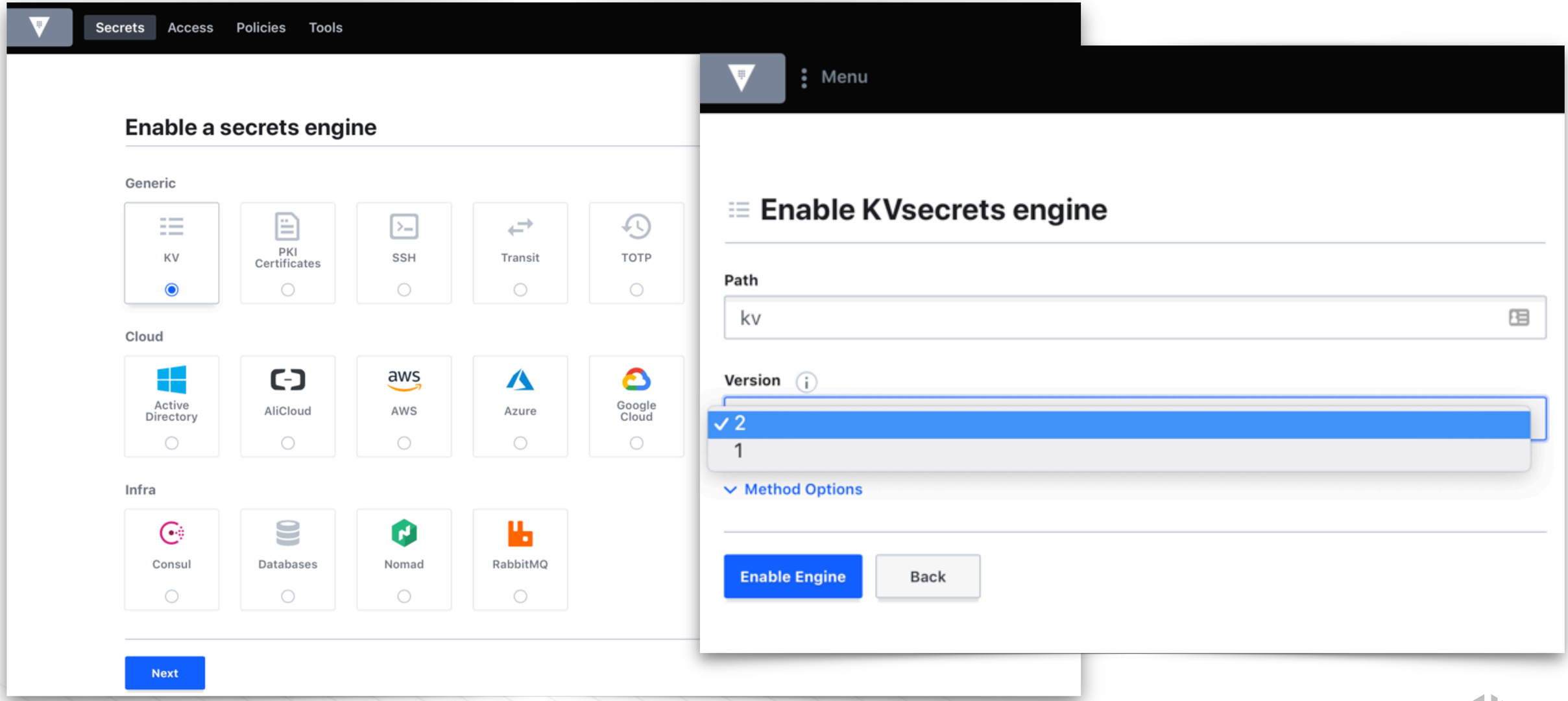
Key/Value Secrets Engine (1 of 2)

- **Key/Value** secrets engine is used to store arbitrary secrets
  - There are two versions: v2 (`kv-v2`) is ***versioned*** but v1 (`kv`) is not
  - Secrets are accessible via interactive or automated means
  - Enforced access control via policies
  - Fully audited access
- The secrets are **encrypted** using 256-bits AES in GCM mode with a randomly generated nonce prior to writing them to its storage backend
  - Anything that leaves Vault is encrypted

Key/Value Secret Engine (2 of 2)

- Key/Value v2 secret engine is **enabled by default** at the **`secret/`** path ***if*** the server is running in ***dev*** mode
  - The path prefix (**`secret/`**) tells Vault to route traffic to key/value v2 secret engine
  - Key/Value secret engine **can be enabled at different paths**
    - Each key/value secret engine is *isolated* and *unique*
- Secrets are stored as key-value pairs
  - Writing to a key in the key/value secret will **replace the old value**
    - Sub-fields are not merged together

Enabling Key/Value Secrets Engine via UI



Enabling K/V v1 Secret Engine

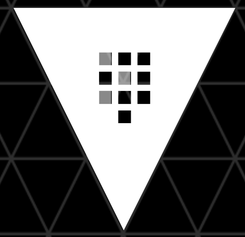
Terminal

```
# Enable k/v v1 secrets engine
$ vault secrets enable kv

# Enable k/v v1 secrets engine at 'secret'
$ vault secrets enable -path=secret/ kv


$ vault secrets list -detailed
Path          Plugin       Accessor              ...   Options
----          ------       --------                    -------
cubbyhole/    cubbyhole    cubbyhole_2d9698bb    ...   map[]
identity/     identity     identity_814b9eeb     ...   map[]
kv/           kv           kv_ee7b19a9           ...   map[]
secret/       kv           kv_1d957541           ...   map[]
sys/          system       system_54ec50d4       ...   map[]
```

Enabling K/V v2 Secret Engine

Terminal

```
# Enable the versioned k/v secrets engine
$ vault secrets enable kv-v2

    Or

# Enable the versioned k/v secrets engine at 'v-secret'
$ vault secrets enable -path=secret/ -version=2 kv

$ vault secrets list -detailed

Path          Plugin       Accessor              ...   Options
----          ------       --------                    -------
cubbyhole/    cubbyhole    cubbyhole_2d9698bb    ...   map[]
identity/     identity     identity_814b9eeb     ...   map[]
kv-v2/        kv           kv_ee7b19a9           ...   map[version:2]
secret/       kv           kv_1d957541           ...   map[version:2]
sys/          system       system_54ec50d4       ...   map[]
```



Upgrade K/V v1 to K/V v2

Terminal

```
# Upgrade an existing non-version k/v to versioned k/v
$ vault kv enable-versioning secret/

Success! Tuned the secrets engine at: secret/
```

Working with Key/Value Secret Engine via CLI

Operation	CLI command for K/V v1	Endpoint for K/V v2
Write	vault kv put <path> key=data	vault kv put <path> key=data
Read	vault kv get <path>	vault kv get <path> or vault kv get -version=<ver> <path>
Delete	vault kv delete <path>	vault kv delete <path> or vault kv delete -versions=<ver> <path>
List	vault kv list <path>	vault kv list <path>
Undelete	N/A	vault kv undelete -versions=<ver> <path>
Destroy	N/A	vault kv destroy -versions=<ver> <path>
Patch	N/A	vault kv patch <path> key=data
Rollback	N/A	vault kv rollback -version=<ver> <path>

Write Secrets: Compare v1 and v2

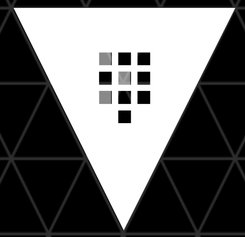
Terminal

```
# Write secrets in non-versioned k/v (v1)
$ vault kv put secret/apikey/splunk  \
       apikey="3230sc$832d"
Success! Data written to: secret/apikey/splunk
```

Terminal

```
# Write secrets in versioned k/v (v2)
$ vault kv put secret/apikey/splunk  \
       apikey="3230sc$832d"
Key              Value
---              -----
created_time     2019-08-14T00:35:43.733151Z
deletion_time    n/a
destroyed        false
version          1
```

The **CLI command is the same** but different output behavior



Write Key/Value Secret via CLI

Terminal

```
# Any value begins with "@" is loaded from a file
$ vault kv put secret/customer/acme @acme.json

$ cat acme.json
{
  "organization": "ACME Inc.",
  "region": "US-West",
  "zip_code": "94105",
  "type": "premium",
  "contact_email": "james@acme.com"
}
```

This is much easier than executing:

```
vault kv put secret/customer/acme organization="ACME Inc."  \
      region="US-West" zip_code="94105" \
      typ="premium" contact_email="james@acme.com"
```

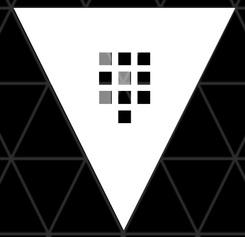
Read Secrets: Compare v1 and v2

Terminal

```
# Write secrets in non-versioned k/v (v1)
$ vault kv get secret/customer/acme

====== Data ======
Key                 Value
---                 -----
contact_email       james@acme.com
organization        ACME Inc.
region              US-West
type                premium
zip_code            94105
```

Terminal

```
# Write secrets in versioned k/v (v2)
$ vault kv get secret/customer/acme

====== Metadata ======
Key              Value
---              -----
created_time     2019-08-14T00:18:14.316494Z
deletion_time    n/a
destroyed        false
version          1

======== Data ========
Key              Value
---              -----
contact_email    james@acme.com
organization     ACME Inc.
region           US-West
type             premium
zip_code         94105
```



Read a specific Version of the Data

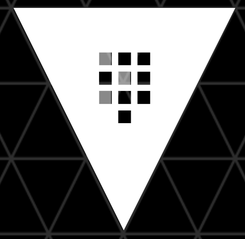
Versioned K/V (`kv-v2`)

Terminal

```
# View a different version of the data
$ vault kv get -version=1 secret/customer/acme

====== Metadata ======
Key              Value
---              -----
created_time     2018-04-12T18:14:48.17956817Z
deletion_time    n/a
destroyed        false
version          1

======== Data ========
Key              Value
---              -----
contact_email    james@acme.com
organization     ACME Inc.
region           US-West
type             premium
zip_code         94105
```

Updating Data

Versioned vs. Non-versioned K/V

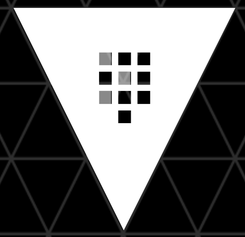


Updating Secrets

Versioned K/V (`kv-v2`)

Terminal

```
# Current data
$ vault kv get secret/customer/acme

====== Metadata ======
  ...
======== Data ========
Key                Value
---                -----
contact_email      james@acme.com
organization       ACME Inc.
region             US-West
type               premium
zip_code           94105


# What happens when you execute the following command???
$ vault kv put secret/customer/acme  \
      contact_email="jennifer@acme.com"
```



Updating Secrets

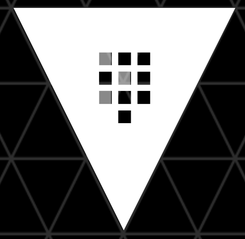
Versioned K/V (`kv-v2`)

Terminal

```
$ vault kv put secret/customer/acme  \
      contact_email="jennifer@acme.com"
Key              Value
---              -----
created_time     2018-04-12T18:55:25.2481343Z
deletion_time    n/a
destroyed        false
version          2

$ vault kv get secret/customer/acme
====== Metadata ======
Key              Value
---              -----
created_time     2018-04-12T18:55:25.2481343Z
deletion_time    n/a
destroyed        false
version          2

======== Data ========
Key              Value
---              -----
contact_email    jennifer@acme.com
```

What if all you wanted was to update the contact email from james@acme.com to jennifer@acme.com?

Now you lost rest of the data...



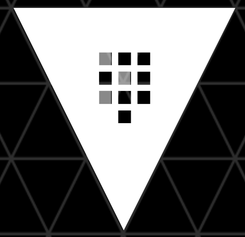
Recover the data

Versioned K/V (`kv-v2`)

Terminal

```
$ vault kv rollback -version=1 secret/customer/acme

Key                 Value
---                 -----
created_time        2019-08-14T00:18:33.037337Z
deletion_time       n/a
destroyed           false
version             3

$ vault kv get secret/customer/acme

====== Metadata ======
...

======== Data ========
Key                 Value
---                 -----
contact_email       james@acme.com
organization        ACME Inc.
region              US-West
type                premium
zip_code            94105
```

Restore the original data (version 1) using the **`vault kv rollback`** command.



Updating Secrets

Versioned K/V (`kv-v2`)

Terminal

```
$ vault kv patch secret/customer/acme  \
        contact_email="jennifer@acme.com"

Key              Value
---              -----
created_time     2018-04-12T18:55:25.2481343Z
deletion_time    n/a
destroyed        false
version          4

$ vault kv get secret/customer/acme

<metadata>
======== Data ========
Key              Value
---              -----
contact_email    jennifer@acme.com
organization     ACME Inc.
region           US-West
type             premium
zip_code         94105
```

The **`vault kv patch`** command updates the data without overwriting the existing data.

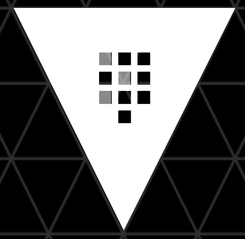
Check-And-Set

Being able to rollback and restore the original data is great.

But, how can I **prevent** my secrets from been overwritten by mistake?

Check-And-Set (kv-v2)

- Versioned K/V (**`kv-v2`**) provides a *check-and-set* (cas) operation to prevent unintentional secret overwrite
- When **cas_required** is set to **true**:
  - Write operation *requires* the **-cas** flag
  - If **-cas=0**, a write will be permitted if the key does not exist
  - If the **-cas** is non-zero, the index value must match the current secret version
- By default, **cas_required=false**
  - You can still pass the **-cas** flag, but *not* required



Check-and-Set

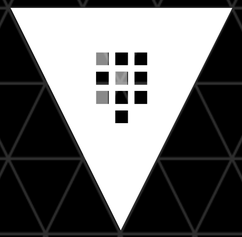
Versioned K/V (`kv-v2`)

Terminal

```
# Enable cas_required on the secret engine mounted at secret/
$ vault write secret/config cas_required=true


# Enable cas_required only on the secret/partner path
$ vault kv metadata put -cas-required=true \
        secret/partner


# When check-and-set is enabled, every write operation requires
# cas value
$ vault kv put -cas=0 secret/apikey \
        admin-key="109384718"
```

Set to **`cas`** to **`0`** if the write should be allowed only if the key does not exist. Otherwise, set it to the **current version** number.

Deleting Data



Delete Secret via CLI

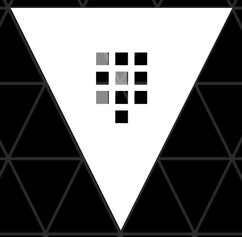
Versioned K/V (`kv-v2`)

Terminal

```
$ vault kv delete secret/apikey/splunk

Success! Data deleted (if it existed) at: secret/apikey/splunk
```

Deletes the ***current*** version of the secret. You can delete a specific version of the data by providing "**`-versions`**" flag

```
$ vault kv get secret/apikey/splunk
====== Metadata ======
Key                Value
---                -----
created_time       2019-08-14T00:35:43.733151Z
deletion_time      2019-08-14T00:36:33.507732Z
destroyed          false
version            1
```

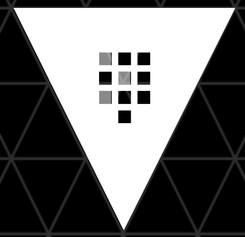


Recover Deleted Secret

Versioned K/V (`kv-v2`)

Terminal

```
$ vault kv undelete -versions=1 secret/apikey/splunk

Success! Data written to: secret/undelete/apikey/splunk



$ vault kv get secret/apikey/splunk
====== Metadata ======
Key              Value
---              -----
created_time     2019-08-14T00:35:43.733151Z
deletion_time    n/a
destroyed        false
version          1

===== Data =====
Key       Value
---       -----
apikey    TEST-239482398
```



Permanently Delete Secret

Versioned K/V (`kv-v2`)

Terminal

```
$ vault kv destroy -versions=1 secret/apikey/splunk



$ vault kv get secret/apikey/splunk
====== Metadata ======
Key                Value
---                -----
created_time       2019-08-14T00:35:43.733151Z
deletion_time      n/a
destroyed          true
version            1


# Delete all versions and metadata
$ vault kv metadata delete secret/apikey/splunk
```

K/V Secret Engine API

Working with Key/Value Secret Engine via API

- All API routes are prefixed with version: **/v1/**
- API is accessed over a **TLS** connection at all times
  - Client token must be sent in the **X-Vault-Token** HTTP header

HTTP verb	Endpoint for K/V v1	Endpoint for K/V v2	Response
GET	secret/<path>	secret/**data**/<path>	200 application/json
POST / PUT	secret/<path>	secret/**data**/<path>	204 (empty body)
LIST	secret/<path>	secret/**data**/<path>	200 application/json
DELETE	secret/<path>	secret/**data**/<path>	204 (empty body)
POST		secret/**undelete**/<path>	204 (empty body)
POST		secret/**destroy**/<path>	204 (empty body)
GET		secret/**metadata**/<path>	200 application/json

Working with API: Compare v1 and v2

Key/Value v1

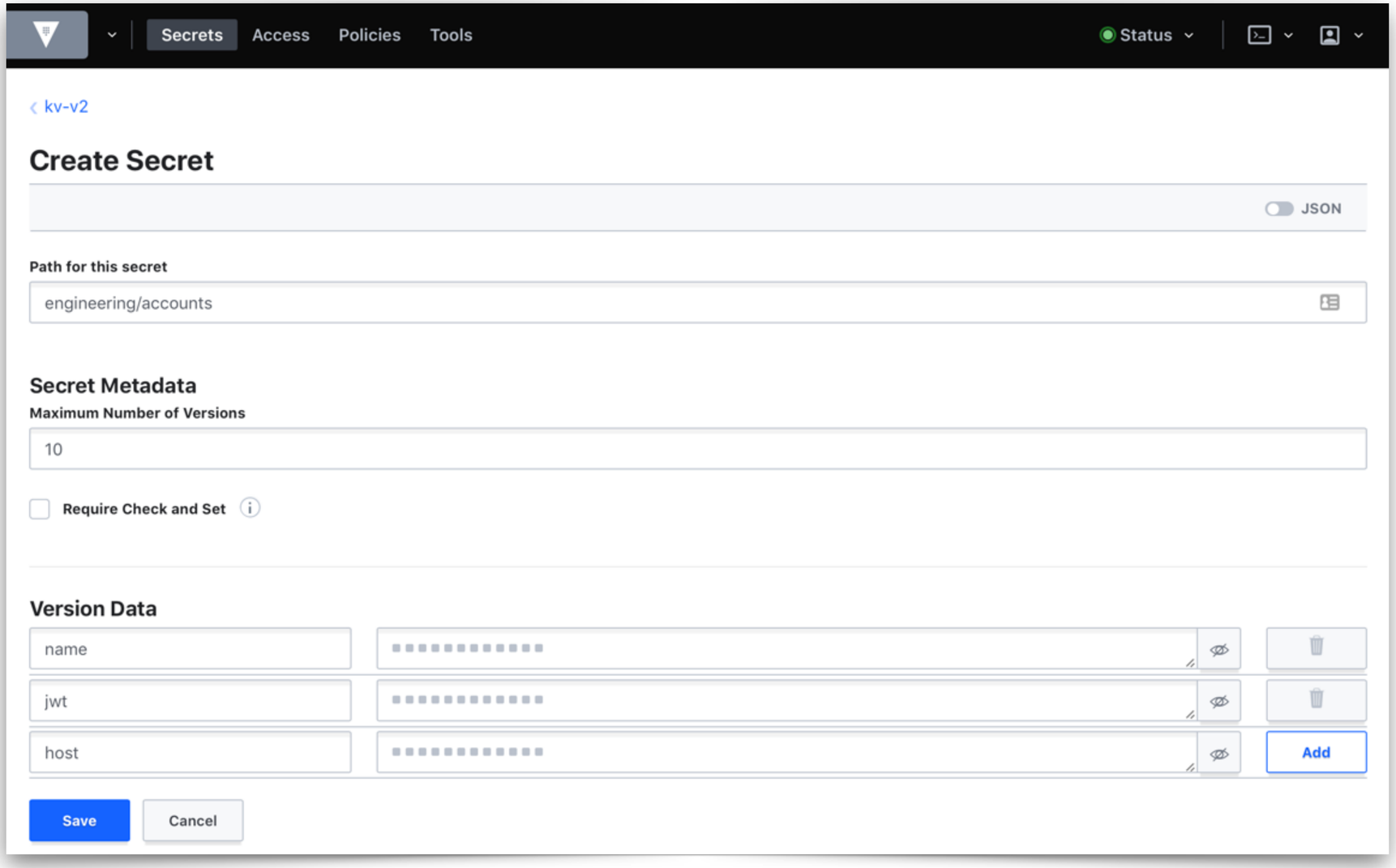
```
# Write secret to secret/apikey/splunk path
$ curl --header "X-Vault-Token:<token>" \
       --request POST \
       --data '{ "apikey": "3230sc$832d" }'
       https://127.0.0.1:8200/v1/secret/apikey/splunk


# Read secrets in secret/apikey/splunk path
$ curl --header "X-Vault-Token:<token>" \
       --request GET \
       https://127.0.0.1:8200/v1/secret/apikey/splunk


# Delete secrets in secret/apikey/splunk path
$ curl --header "X-Vault-Token:<token>" \
       --request DELETE \
       https://127.0.0.1:8200/v1/secret/apikey/splunk
```

Key/Value v2

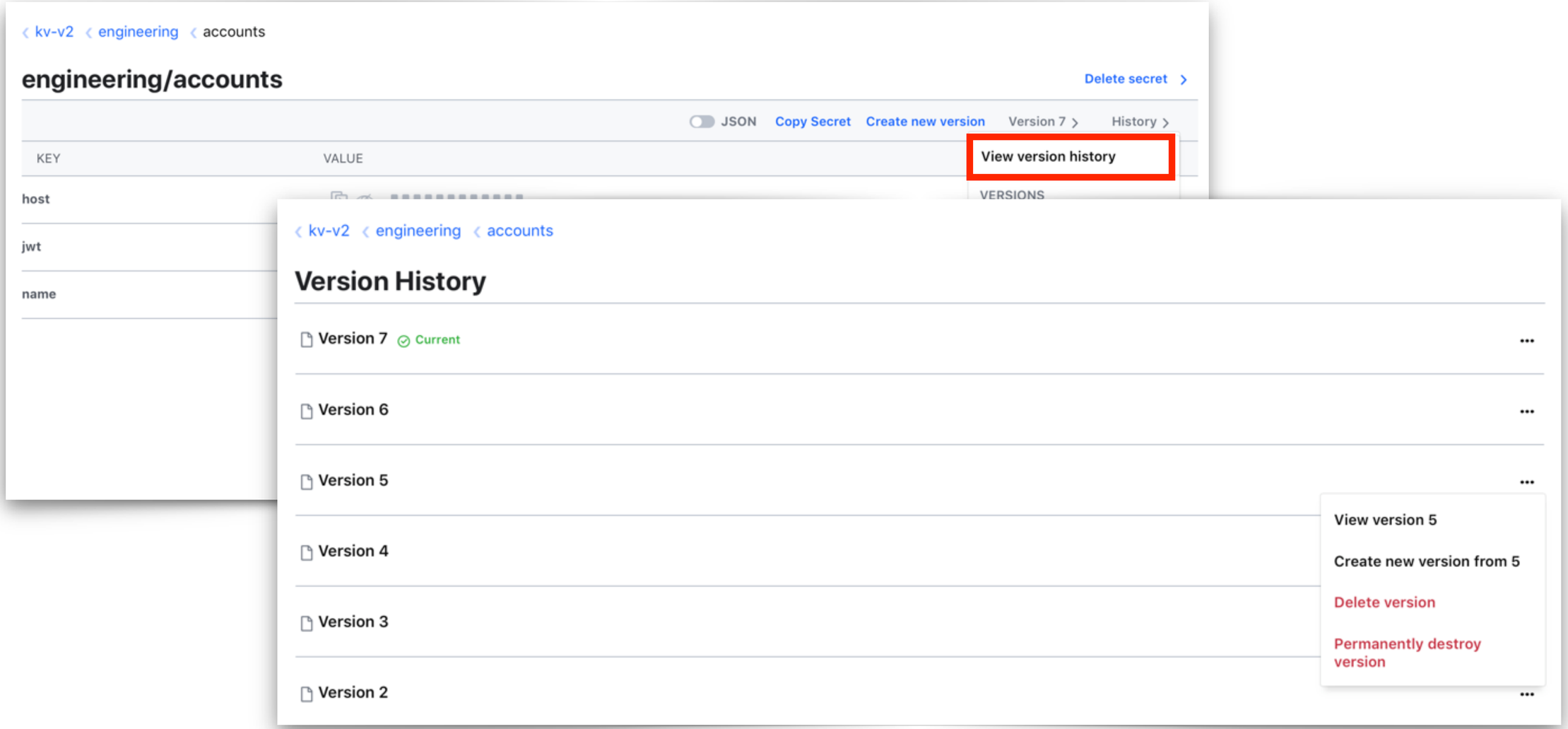
```
# Write secret to secret/apikey/splunk path
$ curl --header "X-Vault-Token:<token>" \
       --request POST \
       --data '{ "apikey": "3230sc$832d" }'
       https://127.0.0.1:8200/v1/secret/data/apikey/splunk


# Read the latest version of the secret
$ curl --header "X-Vault-Token:<token>" \
       --request GET \
       https://127.0.0.1:8200/v1/secret/data/apikey/splunk


# Delete the version 1 of the seret
$ curl --header "X-Vault-Token:<token>" \
       --request DELETE \
       --data '{ "versions": [1] }'
       https://127.0.0.1:8200/v1/secret/data/apikey/splunk
```

Web UI

Web UI (1 of 2)



Web UI (2 of 2)



Knowledge Check & Lab Overview

Knowledge Check Questions

1. How does Vault know which secret engine to route requests to?

2. What does the following command do?

```
$ vault kv get -field=company_size secret/membership/premium
```

3. Which of the following statements are true about **key/value** secret engine?

   ✔ A. Enabled at **secret/** path by default in dev server

   B. Can NOT be enabled at another path

   ✔ C. Writing to a key will replace the old value (write is *NOT* a merge)

   ✔ D. Check-and-Set feature can be a protection against unintentional secret overwrite

Lab 2: Static Secrets

20 minutes

- Open the Lab Book PDF

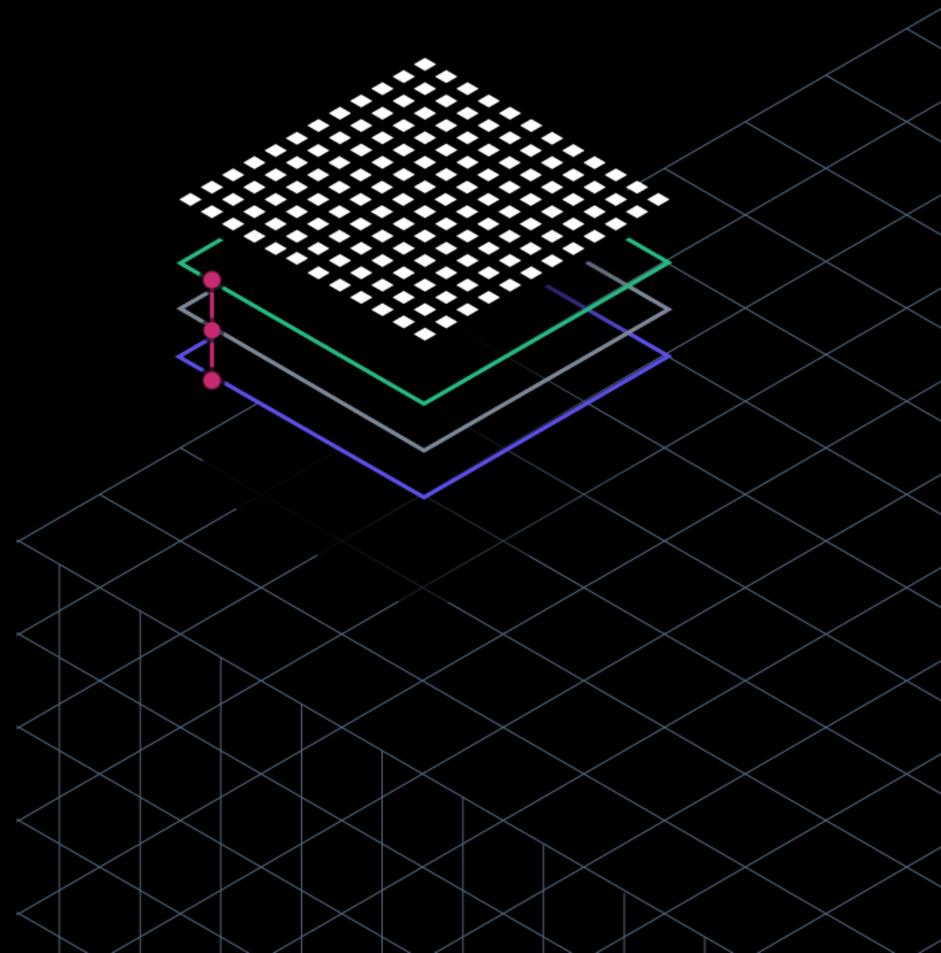
- In this lab, you are going to perform the following tasks:
  - Task 1: Write Key/Value Secrets using CLI
  - Task 2: List Secret Keys using CLI
  - Task 3: Delete Secrets using CLI
  - Task 4: Working with Key/Value Secret Engine using API
  - Task 5: Explorer UI only features
  - Challenge: Protect secrets from unintentional overwrite

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp



Secrets Engines

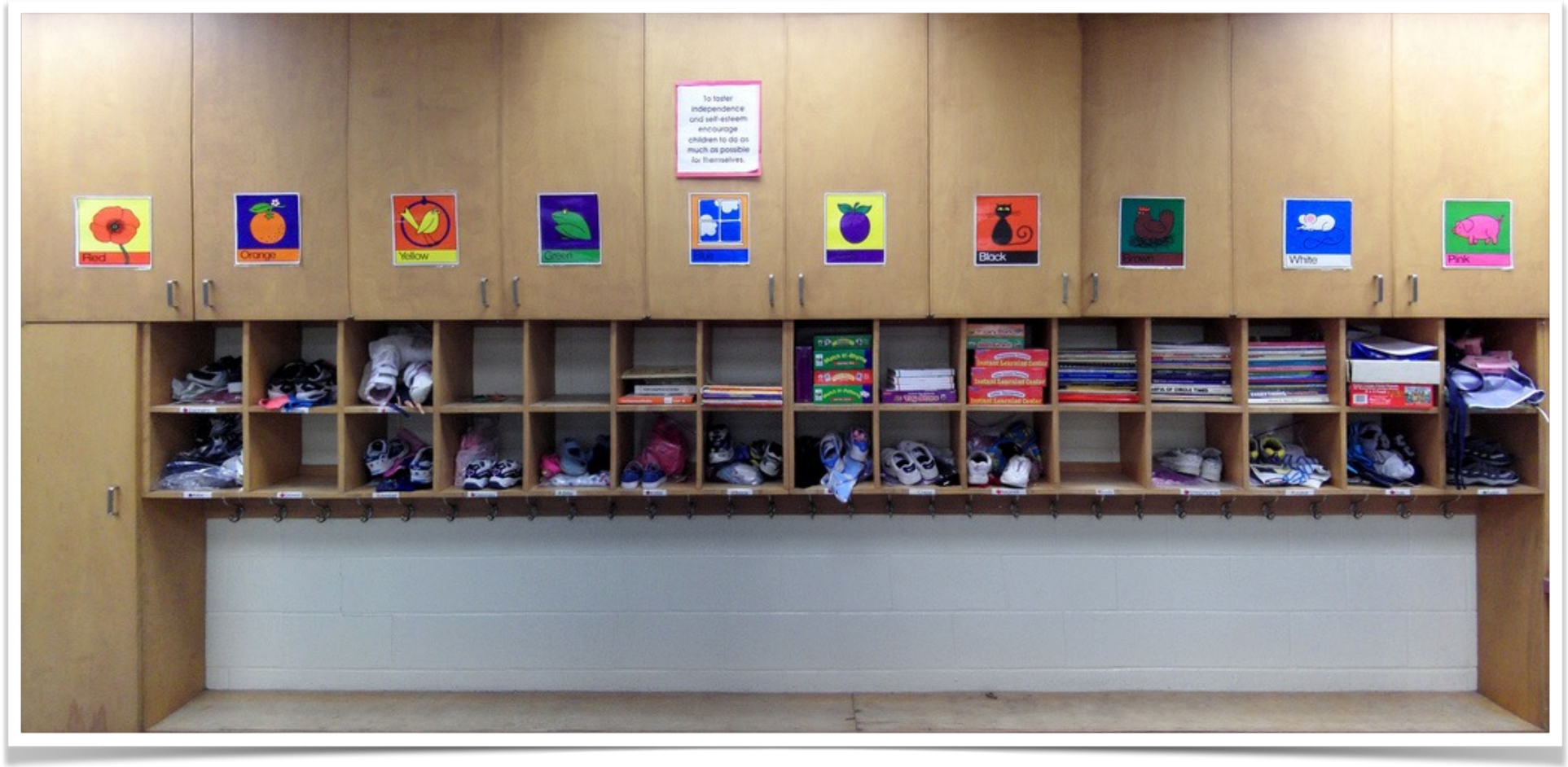
Cubbyhole Secrets Engine

Agenda

- What is the Cubbyhole Secrets Engine?
- Sharing secrets with response wrapping
  - The problems that accompany with sharing a secret
  - The benefits of using response wrapping
  - Examples of wrapping and unwrapping secrets

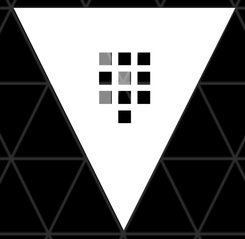
Cubbyhole Secrets Engine

About: Cubbyhole



Cubbyhole Secrets Engine

- **Cubbyhole** Secrets Engine is used to store arbitrary secrets
  - Enabled by default at the `cubbyhole/` path
  - Its lifetime is linked to the **token** used to write the data
    - No concept of a time-to-live (TTL) or refresh interval for values in cubbyhole
    - Even the **root** token cannot read the data if it wasn't written by the root
- Cubbyhole Secrets Engine **cannot** be disabled, moved or enabled multiple times



Write & read secrets in cubbyhole

Terminal

```
$ vault write cubbyhole/github_token token="123456789XYTNE"
Success! Data written to: cubbyhole/github_token


$ vault read cubbyhole/github_token
Key      Value
---      -----
token    123456789XYTNE


# Write secret to cubbyhole/github_token path via API
$ curl --header "X-Vault-Token:<token>" \
       --request POST \
       --data '{ "token": "123456789XYTNE" }'
       https://vault.rocks/v1/cubbyhole/github_token


# Read secrets in cubbyhole/github_token path via API
$ curl --header "X-Vault-Token:<token>" \
       https://vault.rocks/v1/cubbyhole/github_token
```

Response Wrapping

Team Admin

A team member, James needs the secrets stored at `secret/customer` path. However, he does not have the read permission.

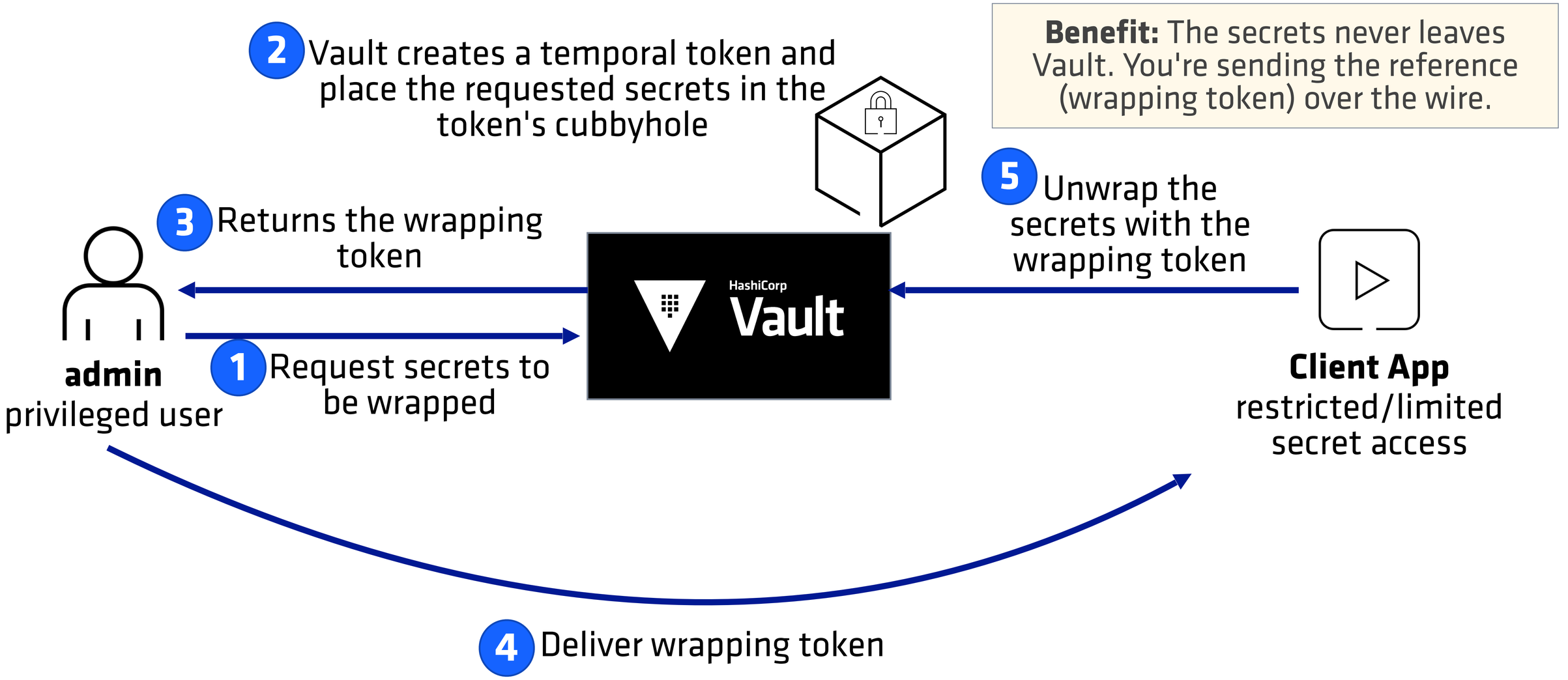
I can retrieve the secrets from Vault for James; however, how can I ***securely distribute*** this secrets to him over the network?

Solution: Cubbyhole Response Wrapping (1 of 2)

- Retrieve the secrets from its path and store it inside **another token's cubbyhole** while the token is:
  - Restricted to only write and read from its own cubby hole
  - Temporary, defined by a Time-to-live (TTL)
  - Limited to single-use

This is called **wrapping token**

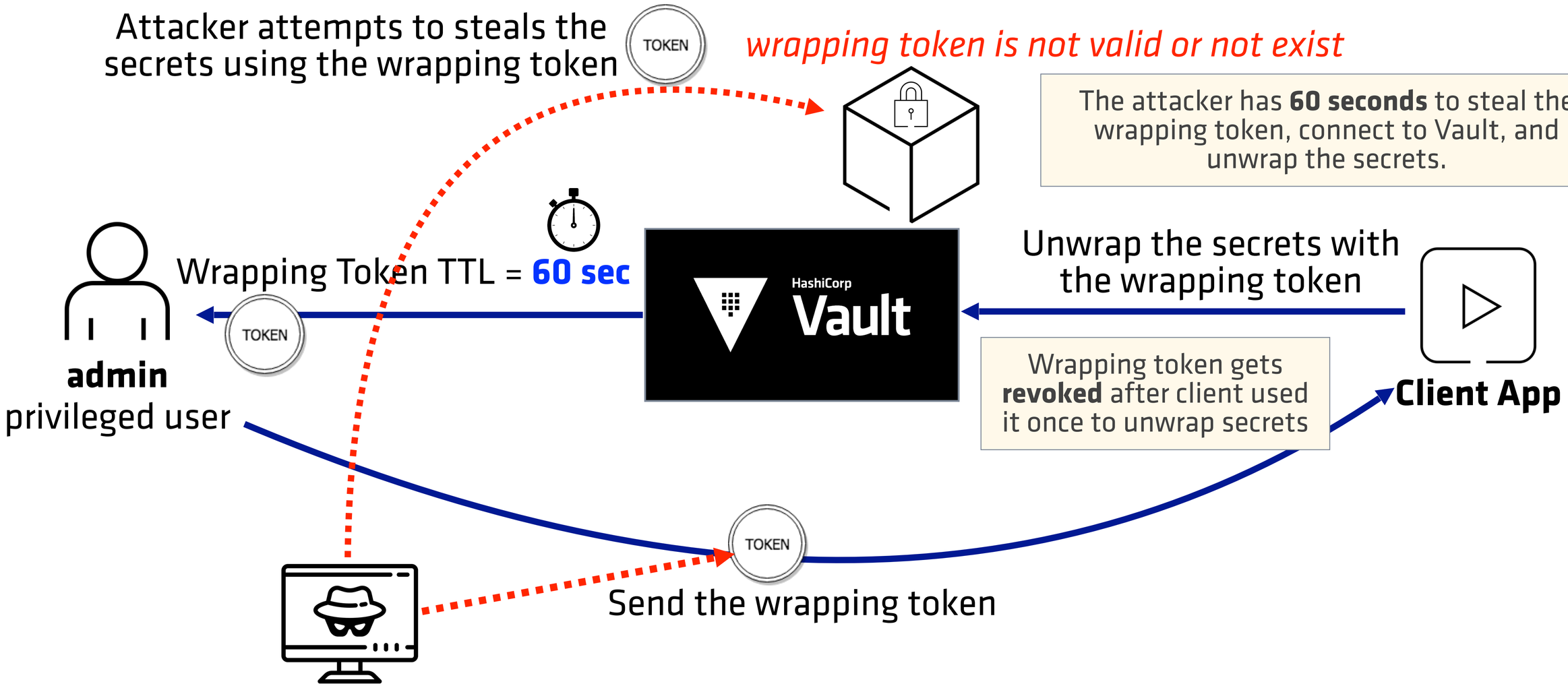
Solution: Cubbyhole Response Wrapping (2 of 2)



Benefits of Response Wrapping (1 of 2)

- It provides:
  - ***cover*** by ensuring that any secret being transmitted across the wire is not the actual secret
  - ***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 short-lived TTL of the response-wrapping token is separate from the token it may wrap.

Benefits of Response Wrapping (2 of 2)



Wrap Secrets

(Privileged User Task)

Expires after 60 seconds

Terminal

```
$ vault kv get -wrap-ttl=60 secret/customer/acme
Key                                 Value
---                                 -----
wrapping_token:                     s.pxJDd9P6DU7D2sHoGhhnRyE0
wrapping_accessor:                  Xy48fOOpdenvmKJC4fV41BoH
wrapping_token_ttl:                 1m
wrapping_token_creation_time:       2019-11-18 16:43:35.393756 -0800
PST
wrapping_token_creation_path:       secret/data/customer/acme
```

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>`**

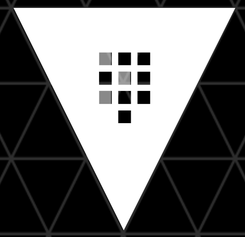
```
$ vault unwrap <wrapping_token>
```

or

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

or

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



Unwrap Secrets

(Expecting Client Task)

Terminal

```
$ vault unwrap s.pxJDd9P6DU7D2sHoGhhnRyE0

Key          Value
---          -----
data         map[contact_email:james@acme.com organization:ACME
             Inc. region:US-West type:premium zip_code:94105]
metadata     map[created_time:2019-11-19T00:42:11.63714Z
             deletion_time: destroyed:false version:2]
```

Response Wrapping API

- Response wrapping is per-request and triggered by providing a desired TTL
  - To wrap a secret, set desired TTL using **X-Vault-Wrap-TTL** in the request header

```
$ curl --header "X-Vault-Wrap-TTL: <TTL>" \
       --header "X-Vault-Token: <token>" \
       --request <HTTP_verb>  --data '<parameters>' \
       <VAULT_ADDRESS>/v1/<request_endpoint>
```

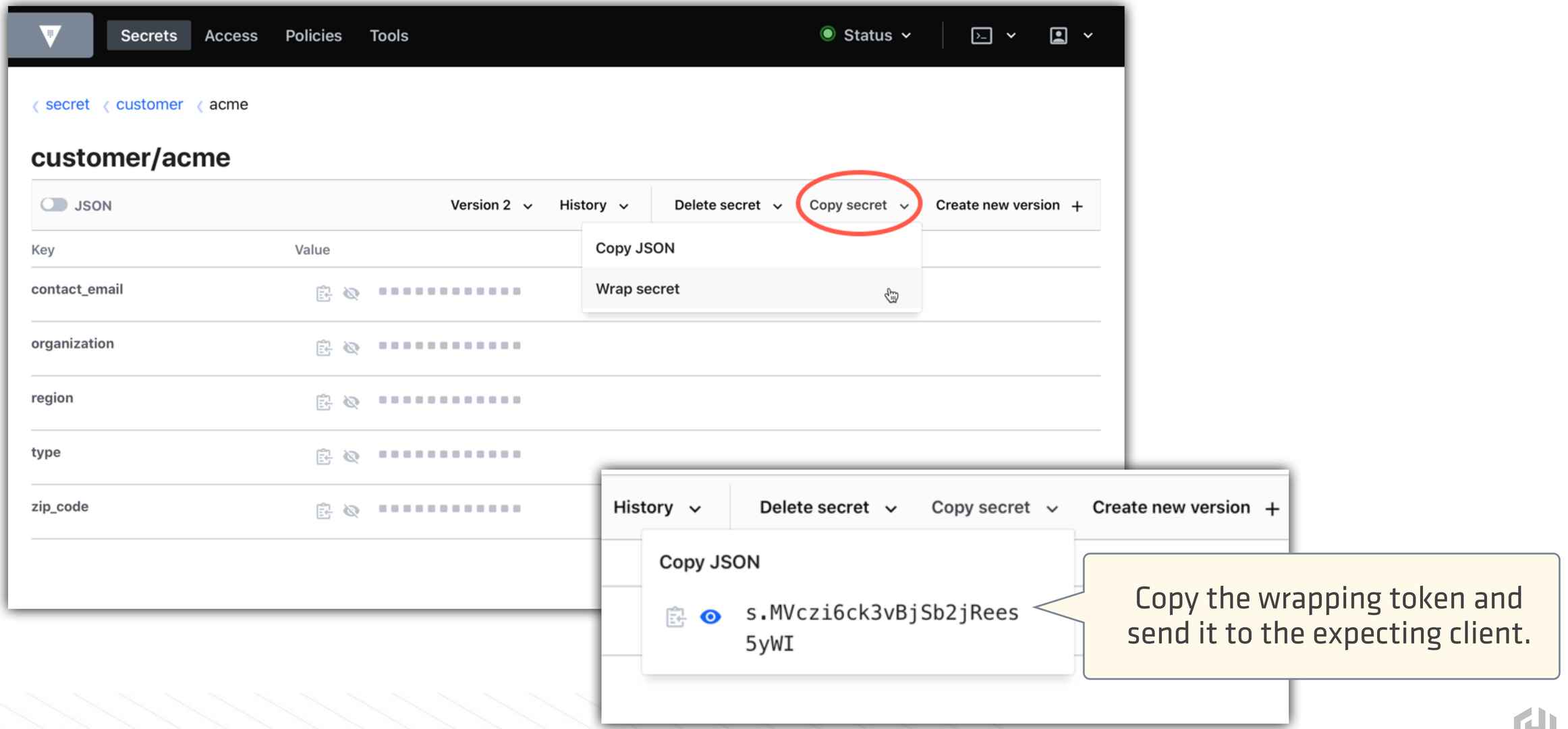
  - To unwrap a secret:

```
$ curl --header "X-Vault-Token: <wrapping_token>" \
       --request POST  \
       <VAULT_ADDRESS>/v1/sys/wrapping/unwrap
```

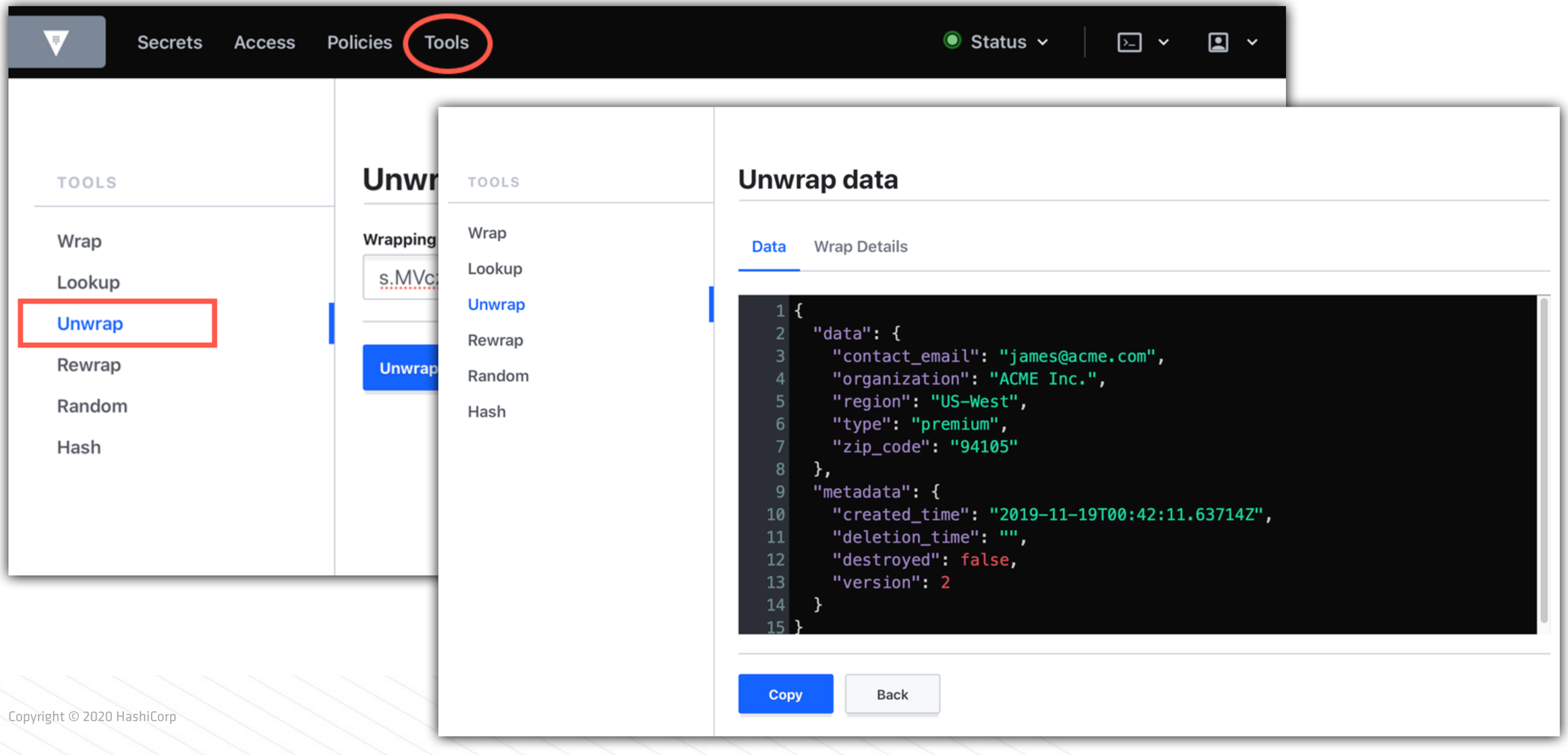
What if James does not have access to Vault? (He cannot log in with Vault.)

```
$ curl --header "X-Vault-Token: <wrapping_token>" \
       --request POST  \
       <VAULT_ADDRESS>/v1/sys/wrapping/unwrap
```

Web UI Support: Wrap Secret



Web UI Support: Unwrap Secret



Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which of the following statements **are** true about Cubbyhole Secrets Engine?
   - A. You can enable it at multiple paths
   - ✓ B. Secrets in cubbyhole is tied to its tokens
   - ✓ C. There is no concept of TTL or refresh interval with Cubbyhole
   - D. The *root* token has privileged permission to read secrets in cubbyhole

2. Which of the following statements **are** true about response wrapping?
   - ✓ A. Only the expecting client can unwrap the secret
   - B. Wrapping token inherits the wrapped secret's TTL
   - ✓ C. Wrapping token is a single-use token
   - D. Wrapping token should have a long TTL

Lab 3: Cubbyhole Secrets Engine

20 minutes

- Open the Lab Book PDF

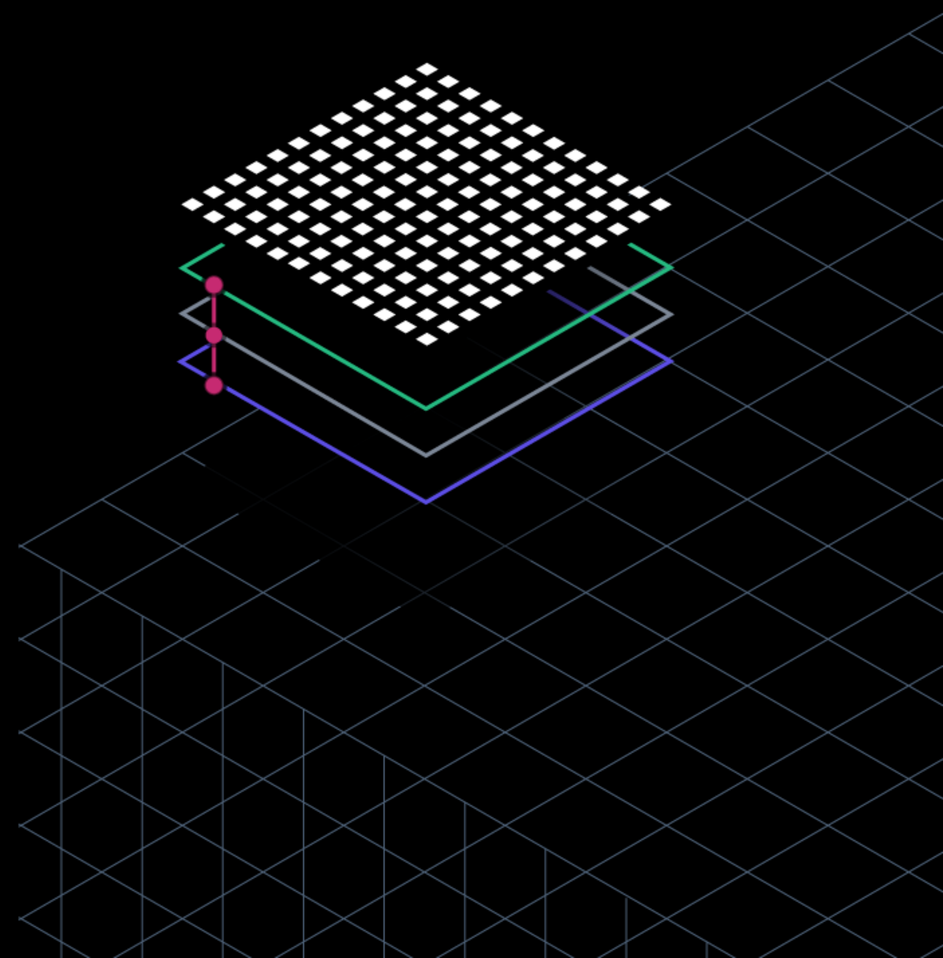
- In this lab, you are going to perform the following tasks:
  - Task 1: Test the cubbyhole Secrets Engine using CLI
  - Task 2: Trigger response wrapping
  - Task 3: Unwrap the wrapped secret
  - Task 4: Response wrapping via the web UI

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp



Secret as a Service

Dynamic Secrets

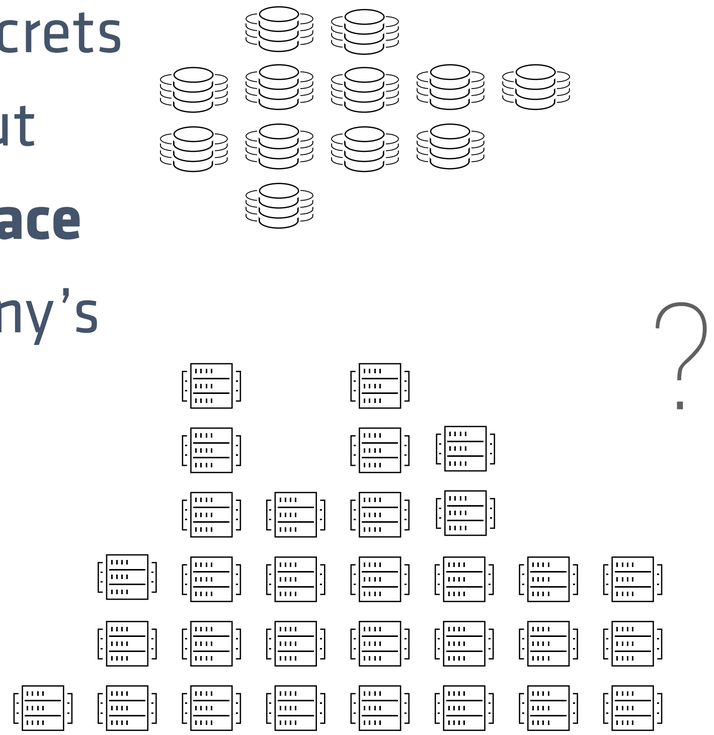
Agenda

- Setting Up a Secret Engine
  - Enabling and Configuring a Secret Engine
  - Creating Roles
  - Creating Policies
- Acquiring Credentials
- Secret Lease Lifecycle
  - Lease Hierarchy and Revocation

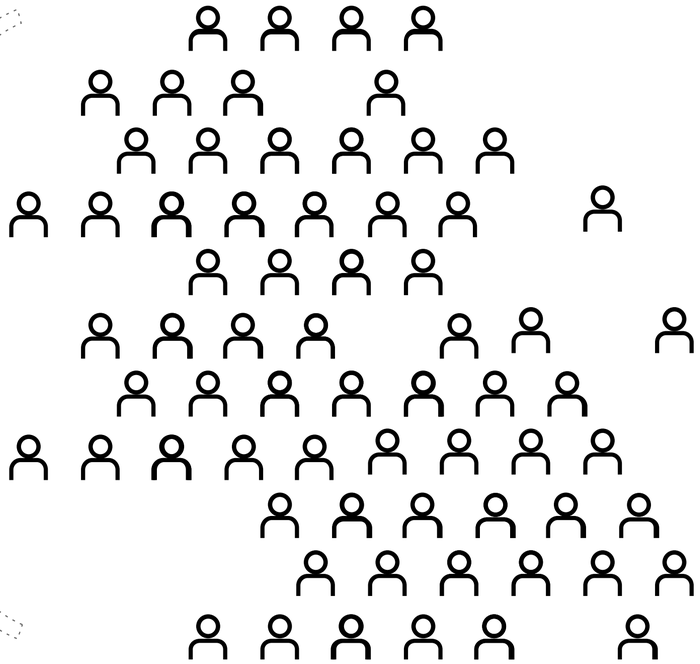
Dynamic Secrets Introduction

Challenge: Secret Exposure

ACME, Inc. has several different divisions of their company, with separate needs to isolate secrets and information by team, but they're **struggling to keep pace with growth** and the company's sensitive information keeps spreading due to **everyone sharing credentials**.

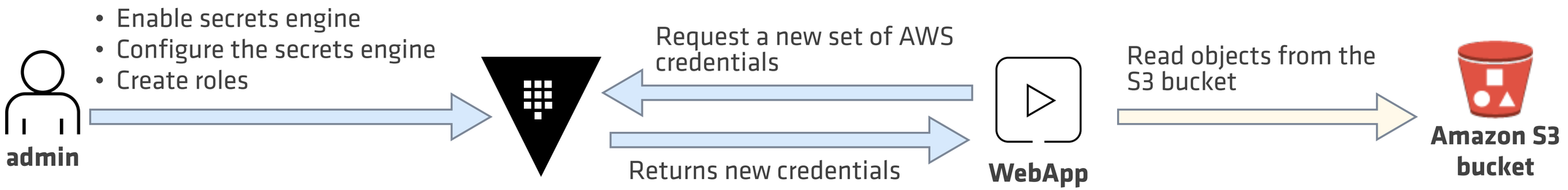


What if these are **apps** or **micro-services**?

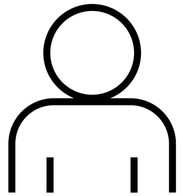


Solution: Secrets as a Service

- Use Vault to generate and manage the lifecycle of credentials **on-demand**:
  - No more sharing credentials
    - Credentials get **revoked automatically** at the end of its life
    - Audit trail can identify point of compromise
  - Use policy to control the access based on the client's role

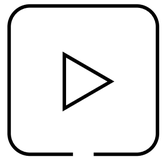


Secret Engine Workflow



privileged users (admin, security engineer, etc.)

- Enable the secrets engine
- Configure the connection to the backend (AWS, PostgreSQL, Consul, etc.)
- Create roles defining permissions to the backend system
- Create policies granting permission to read from the secret engine



clients (users, apps, services, machines, etc.)

- Read a set of credentials from secret engine
- Renew the lease before its expiration if needed or permitted

Setting Up a Secret Engine

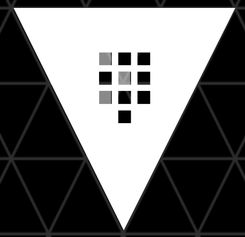
Privileged User Tasks

Secret Engine Commands

- Usage: **`vault secrets <subcommand> [options] [args]`**
  - Display detailed help: **`vault secrets -h`**

Subcommand	Description
disable	Disable a secret engine
enable	Enable a secret engine
list	List enabled secret engines
move	Move an already enabled secret engine to a new path
tune	Tune a secret engine configuration (e.g. TTLs)

- Most secret engines need to be ***enabled*** first to interact

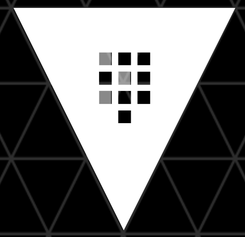


Example: Enabling AWS Secret Engine

(1 of 3)

Terminal

```
# Enable aws secret engine at a defaulted path
$ vault secrets enable aws
Success! Enabled the aws secrets engine at: aws/
```



Example: Configure the AWS Secret Engine

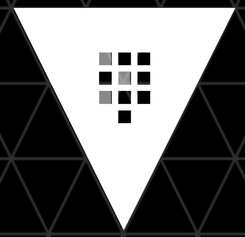
(2 of 3)

Configuration of the secret engine depends on the backend that it is connecting to.

Terminal

```
# Configure aws secret engine to connect to AWS
$ vault write aws/config/root \
        access_key=<AWS_ACCESS_KEY_ID>  \
        secret_key=<AWS_SECRET_KEY>

Success! Data written to: aws/config/root
```

It is common to give Vault a highly privileged credentials and let Vault manage the auditing and lifecycle credentials.



Example: Configure the AWS Secret Engine

(3 of 3)

Terminal

```
# You can specify the lifecycle of generated credentials
$ vault write aws/config/lease \
        lease="1h" lease_max="24h"

Success! Data written to: aws/config/lease
```

Limiting the lease duration to 1 hour. Maximum is 24 hours.

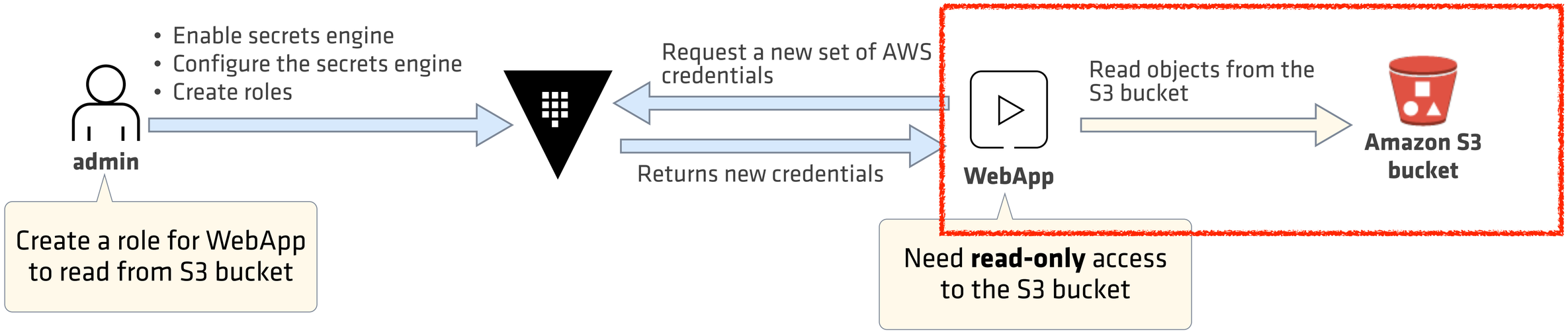
If you don't specify, the system default gets applied (768 hours = 32 days).

Creating Roles

Privileged User Tasks

Creating Roles

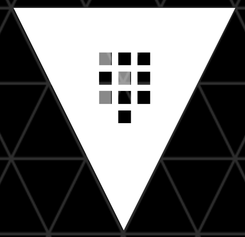
- Vault does **not** know what Amazon permissions, groups and policies you want to attached to the credentials
  - Each **role** maps to a set of permissions to the connecting system



AWS IAM Policy Example

Example: IAM policy (**`webapp-policy.json`**)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::<bucket_name>"
    },
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::<bucket_name>/*"
    }
  ]
}
```



Example: Create a Role

Terminal

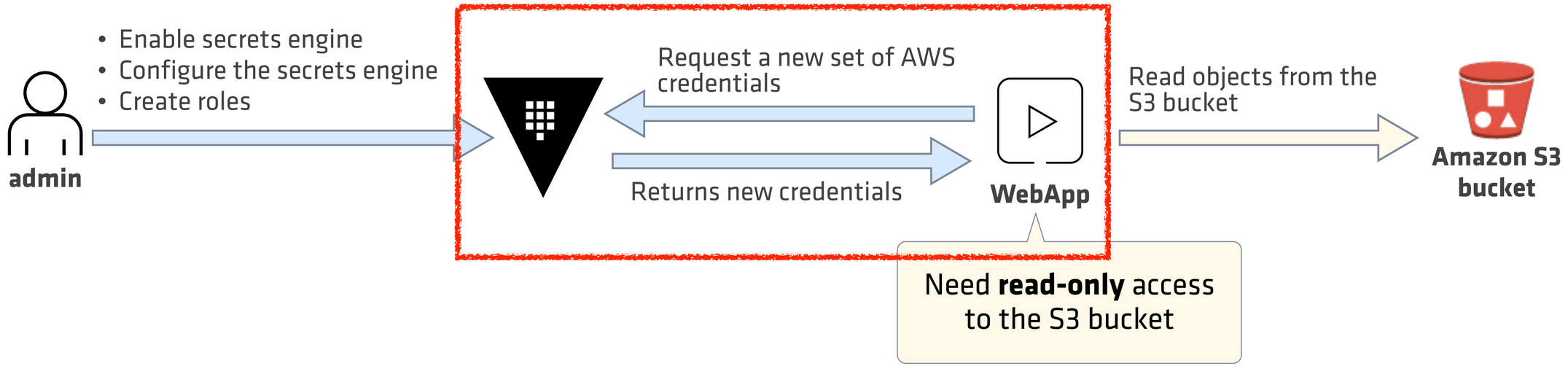
```
# Create a role named 'webapp'
$ vault write aws/roles/webapp \
        credential_type=iam_user \
        policy_document=@webapp-policy.json



# Alternatively, you can use existing IAM policies
$ vault write aws/roles/readonly \
   credential_type=iam_user \
   policy_arns=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Creating Vault Policies

Privileged User Tasks

Creating Vault Policies for WebApp

- At bare minimum, Vault clients need to have a permission to **read** generated credentials from Vault
  - To allow an expiring lease to be **renewed** by the client, policy must permit that



Example Policy for WebApp

- ACL policy for the WebApp client:

```
# Read credentials for webapp role
path "aws/creds/webapp" {
   capabilities = [ "read" ]
}

# To renew the lease
path "sys/lease/renew" {
   capabilities = [ "update" ]
}
```

- To create policies:

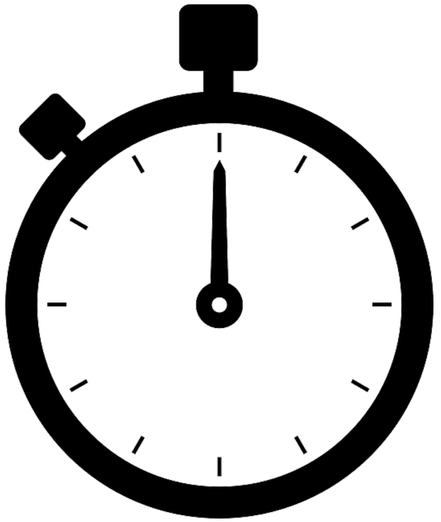
```
vault policy write <POLICY_NAME> <POLICY_PATH>
```

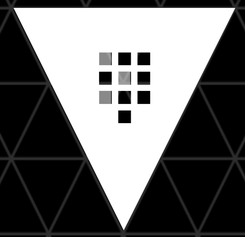
Acquiring and Managing Secret Lease

Client's Activities

Dynamic Secrets Lease

- Every dynamic secret (and token) has a ***lease***
- Secrets are revoked at the end of the lease unless ***renewed***
- Secrets may be ***revoked*** early by operators
  - "*Break Glass*" procedure
  - Enforceable leases

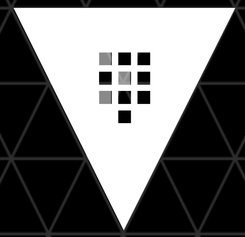




Get a new set of AWS credentials

Terminal

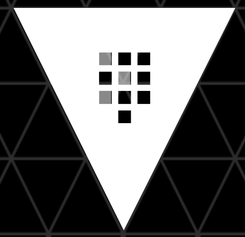
```
$ vault read aws/creds/webapp

Key                Value
---                -----
lease_id           aws/creds/webapp/9c326a6e-64d3-5b82-5692-2cbcc561f1cf
lease_duration     1h
lease_renewable    true
access_key         AKIAIVETWJ7L5B2CJ6CA
secret_key         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
security_token     <nil>
```



Renew the AWS secret lease

Terminal

```
# Renew a lease before it expires
$ vault lease renew <lease_id>
Key                Value
---                -----
lease_id           aws/creds/webapp/7293d215-74ed-94f4-ba47-ced293e0c563
lease_duration     1h
lease_renewable    true

# Renew a lease and increment its TTL
$ vault lease renew -increment="2h" <lease_id>
Key                Value
---                -----
lease_id           aws/creds/webapp/7293d215-74ed-94f4-ba47-ced293e0c563
lease_duration     2h
lease_renewable    true
```



Revoke the AWS secret lease

Terminal

```
# Revoke any secret lease before its expiration
$ vault lease revoke <lease_id>

All revocation operations queued successfully!

# Revoke all secret leases for 'webapp' role
$ vault lease revoke -prefix aws/creds/webapp

All revocation operations queued successfully!
```

Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which of the following statements are true about dynamic secrets?
   A. Leases are renewed automatically if lease_renewable is set to true
   ✓B. Leases can be revoked before reaching its expiration
   C. If you don't specify the lease's TTL, it has no expiration by default
   ✓D. When a lease reaches its max TTL, no longer renewable

2. True or False: Vault knows how to generate a set of new credentials, but does not know what type of credentials to generate. So, you need to specify the rules using the backend system specific language (e.g. AWS IAM policies, SQL).

Lab 4: Database Secrets Engine

25 minutes

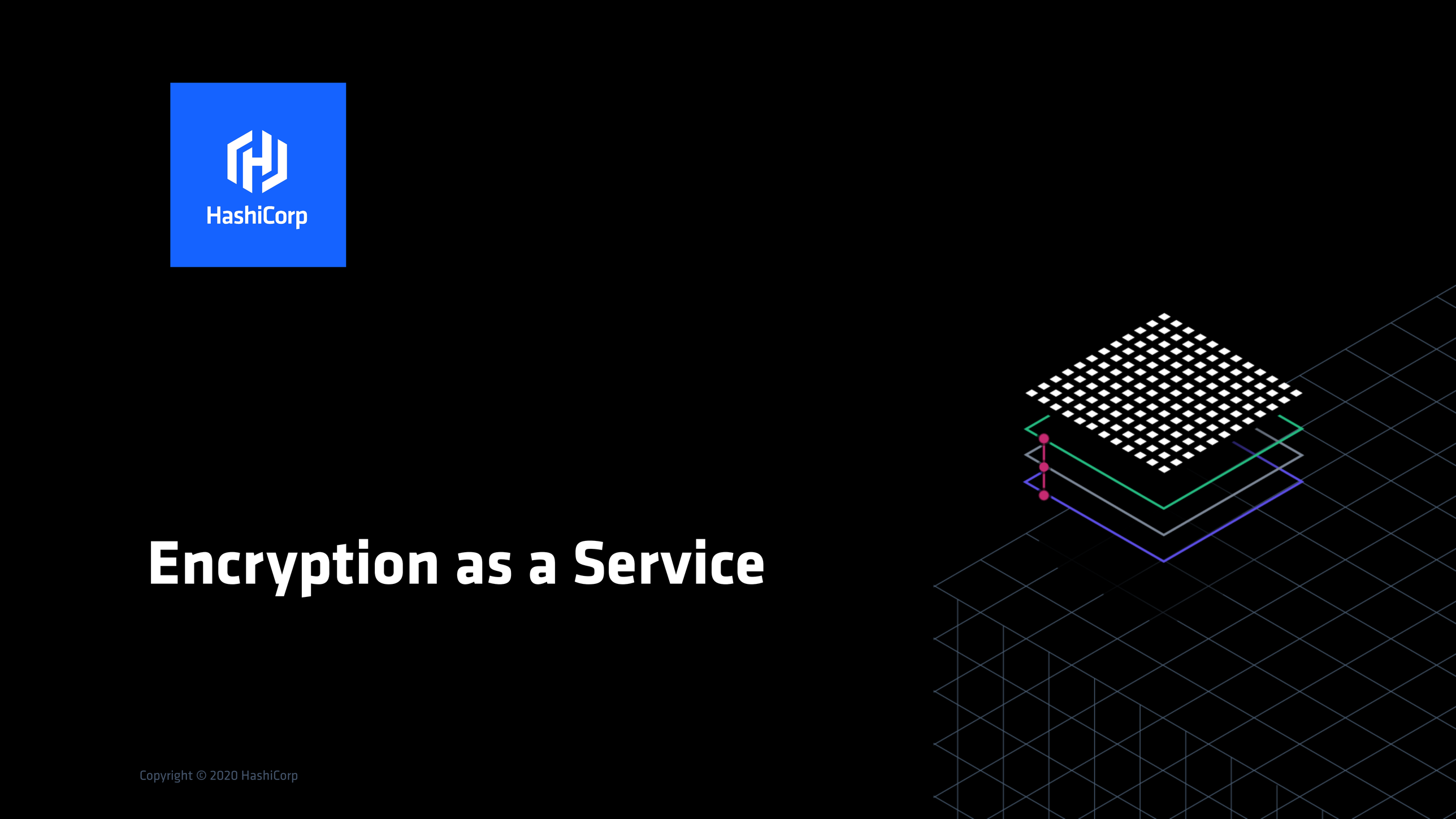
- Open the Lab Book PDF

- In this lab, you are going to perform the following tasks:
  - Task 1: Enable and configure a database secret engine
  - Task 2: Create and manage a static role
  - Task 3: Generate dynamic readonly credentials
  - Task 4: Revoke leases
  - Challenge: Setup database secret engine via API

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com



Encryption as a Service

Agenda

- Data encryption challenge
- Encryption as a Service with Transit secrets engine
  - Supported key types
  - Transit secrets workflow
- Encryption key rotation
  - Key configuration
  - Re-wrapping data

Transit Secrets Engine

Security Engineer

Encrypting data requires collaborative effort across the organization (data producer, data consumer, engineering team, etc.)

Applications must implement encryption and decryption of data

How do we manage the encryption key?

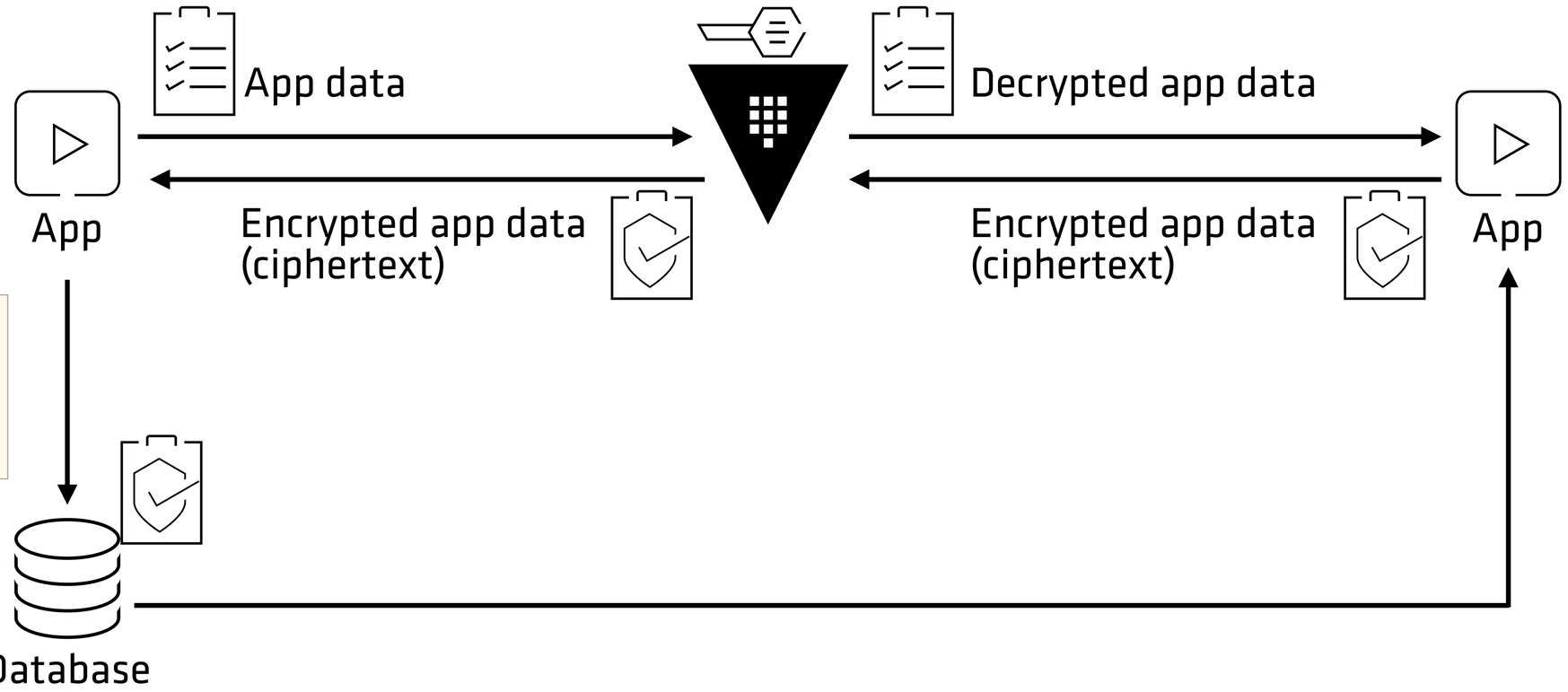
Encryption as a Service: Transit Secrets Engine

- **Transit** secrets engine handles cryptographic functions on data in-transit
  - Allow applications to *outsource* secrets management and encryption
    - Applications **never have access** to the underlying encryption keys
    - Decouples storage from encryption and access control
  - Convergent encryption mode is supported to allow *searchable ciphertext*
  - Keys can be **rotated** and rewrap the ciphertext with new key
    - Configurable **required minimum key version** for decryption
    - Prevents old or lost ciphertext from being decrypted

Encryption as a Service

App sends data to Vault and have it encrypted

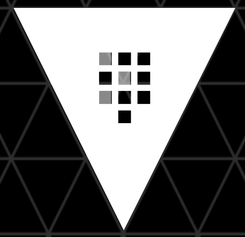
Vault decrypts the ciphertext for the app to use



Ciphertext can be stored in the primary data store

Transit Secrets Engine: Key Types

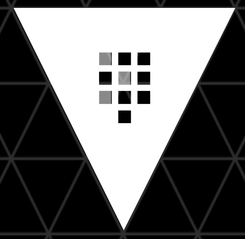
Key Type	Description
aes256-gcm96	AES-GCM with a 256-bit AES key and a 96-bit nonce (encryption, decryption, key derivation, and convergent encryption)
chacha20-poly1305	ChaCha20-Poly1305 with a 256-bit key (encryption, decryption, key derivation, and convergent encryption)
rsa-2048	2048-bit RSA key (encryption, decryption, signing, and signature verification)
rsa-4096	4096-bit RSA key (encryption, decryption, signing, and signature verification)
ed25519	Ed25519 (signing, signature verification, and key derivation)
ecdsa-p256	ECDSA using the P256 elliptic curve (signing and signature verification)



Setup transit secrets engine

Terminal

```
# Enable transit secrets engine
$ vault secrets enable transit

# Create a key to encrypt 'phone_number'
$ vault write -f transit/keys/phone_number

# Specify the key type (default would be used if not specified)
$ vault write transit/keys/ssn \
        type="chacha20-poly1305"
```



Encrypt / Decrypt Data

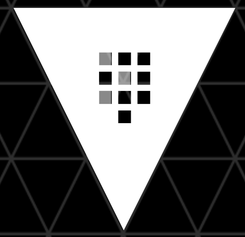
Terminal

```
# Encrypt data with 'phone_number' key
$ vault write transit/encrypt/phone_number \
        plaintext=$(base64 <<< "1-(415)123-4567")

# Decrypt data with 'phone_number' key
$ vault write transit/decrypt/phone_number \
        ciphertext="vault:v1:tgx2vsxtlQRfyLSKvem..."
 Key           Value
 ---           -----
 plaintext     MS0oNDE1KTEyMy00NTY3Cg==

# Decode
$ base64 --decode << "MS0oNDE1KTEyMy00NTY3Cg=="
```

Key Rotation

Key Rotation

- Transit secrets engine allows **easy key rotation**
  - Keys can be rotated manually or by an automated process
- Vault maintains the versioned keyring
  - Admins can decide the **minimum key version allowed** for decryption operations
  - You can upgrade already encrypted data to a new key



Key Rotation

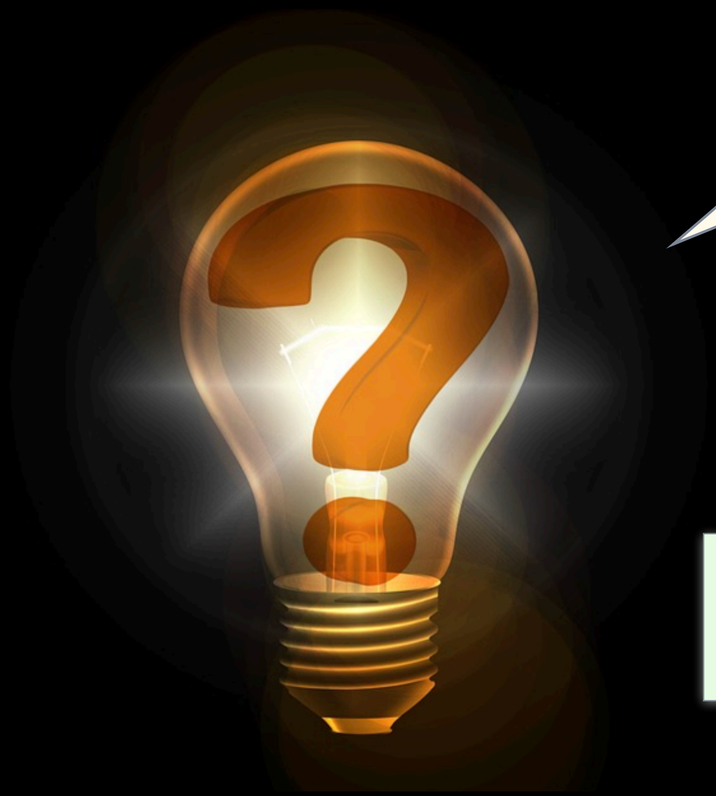
Terminal

```
# Rotate the 'phone_number' key
$ vault write -f transit/keys/phone_number/rotate

# Read 'phone_number' key
$ vault read transit/keys/phone_number
Key                          Value
---                          -----
...
keys                         map[1:1549341798 2:1549346826
3:1549347105 4:1549347108 5:1549347109 6:1549347110]
latest_version               6
min_available_version        0
min_decryption_version       1
min_encryption_version       0
...
```

Key Configuration

Terminal

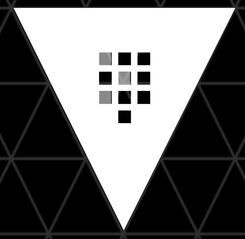
```
# Update key configuration
$ vault write transit/keys/phone_number/config \
        min_decryption_version=5

$ vault read transit/keys/phone_number
 Key                         Value
 ---                         -----
 ...
 keys                        map[5:1549347109 6:1549347110]
 latest_version              6
 min_available_version       0
 min_decryption_version      5
 min_encryption_version      0
 ...
```



What happen to those ciphertext that were encrypted with v1, v2, v3, or v4 of the key?

A: Vault would refuse to decrypt the data as the key used is less than the minimum key version allowed.

This prevents old copies of ciphertext from being decrypted, should they fall into the wrong hands.



Re-wrap Data

Terminal

```
# Upgrade already encrypted data to a new key
$ vault write transit/rewrap/phone_number \
        ciphertext="vault:v1:tgx2vsxtlQRfyLSKvem..."

 Key           Value
 ---           -----
 ciphertext    vault:v6:Xa1f9FIJtn13em/Wb7QCsXsU/kCOn7...
```

Knowledge Check & Lab Overview

Knowledge Check Questions

1. True or False: The transit secrets engine handles cryptographic functions on data in-transit. Vault does NOT store the data.

2. True or False: The client app can access the underlying encryption key when it first sends data to be encrypted.

3. True or False: Once the key was rotated, those already encrypted data get re-wrapped with the new key automatically.

4. True or False: To prevent old copies of ciphertext from being decrypted, you should specify the minimum key version allowed for decryption.

Lab 5: Encryption as a Service - Transit Secrets Engine

25 minutes

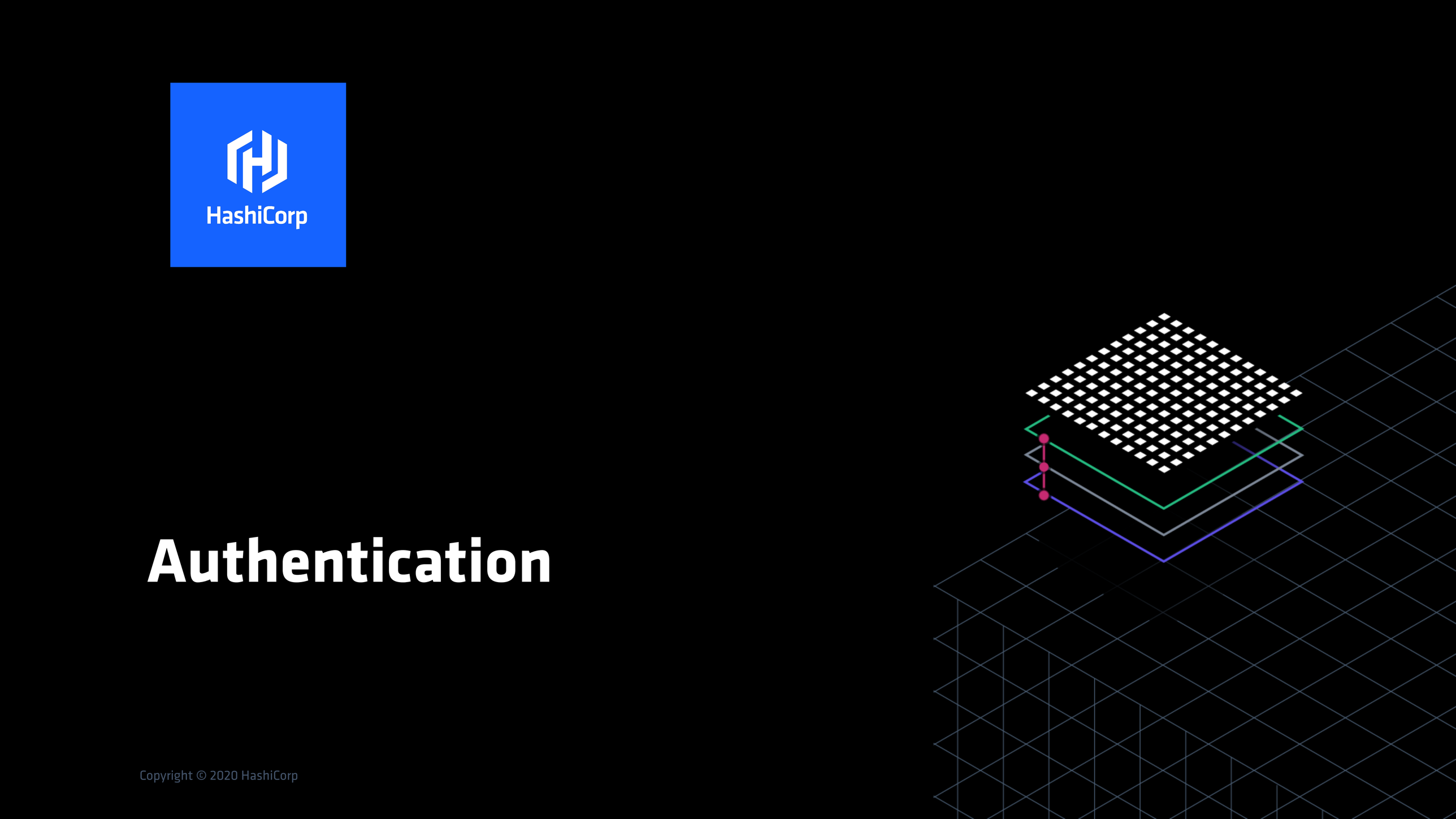
- Open the Lab Book PDF

- In this lab, you are going to perform the following tasks:
  - Task 1: Configure transit secrets engine
  - Task 2: Encrypt secrets
  - Task 3: Decrypt a cipher-text
  - Task 4: Rotate the encryption key
  - Task 5: Update the key configuration
  - Task 6: Encrypt data via UI
  - Challenge: Sign and validate data

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com



Authentication

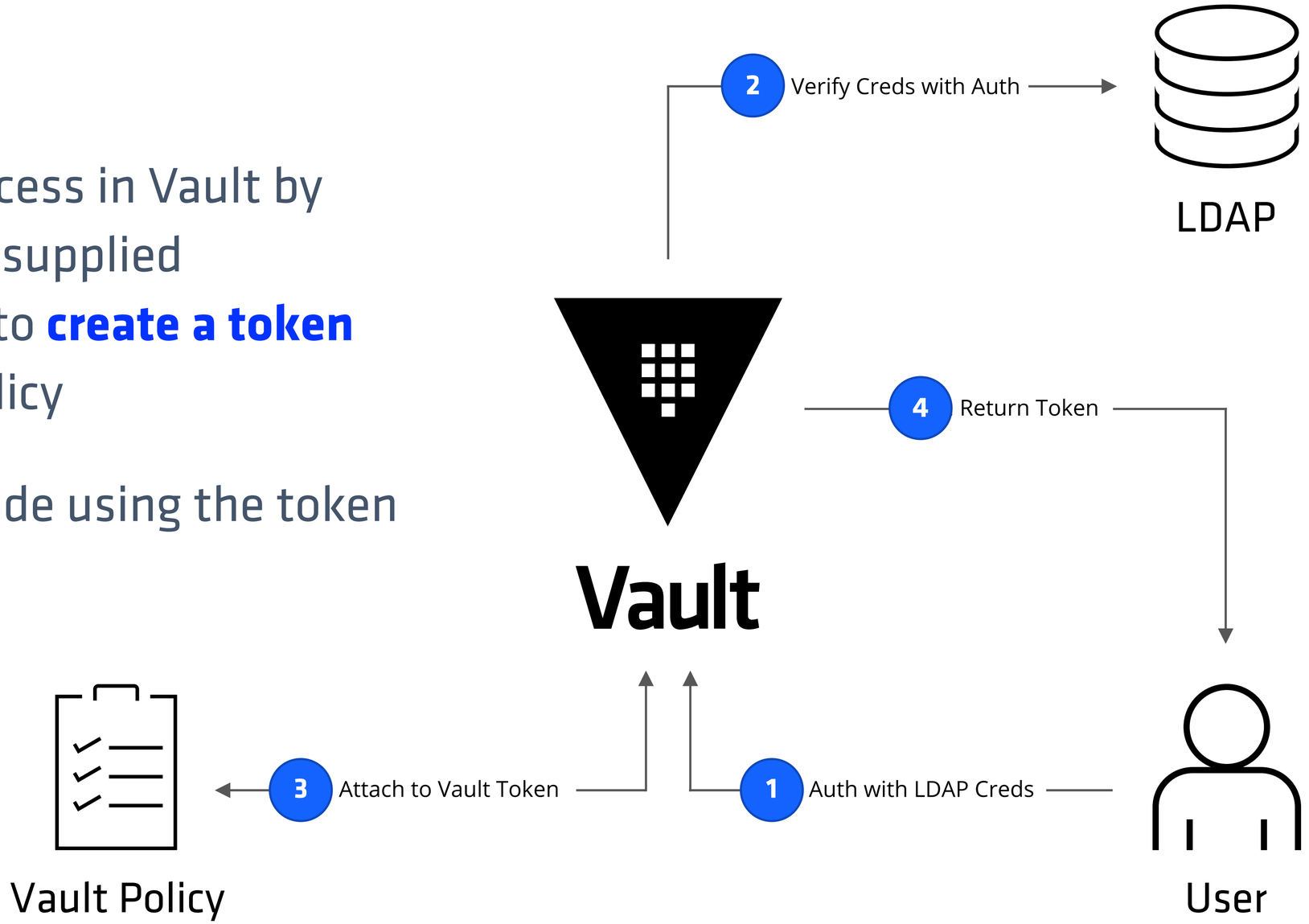
Agenda

- Authentication
  - Authentication Methods in Vault
  - Authentication Setup Workflow
  - Admin task commands
  - User commands
- Tokens
  - Controlling the Token Lifecycle
  - Periodic Tokens
  - Orphan Tokens
  - Tokens with Use Limit

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



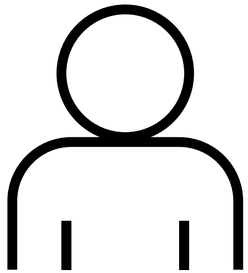
Auth Methods (1 of 2)

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
Google Cloud	Allows authentication against Vault using Google credentials
JWT / OIDC	Authenticate with Vault using a JSON Web Token (JWT) or using OIDC
Kubernetes	Authenticate with Vault using a Kubernetes Service Account

Auth Methods (2 of 2)

Method	Description
GitHub	Authenticate with Vault using a GitHub personal access token
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

Authentication Setup Workflow

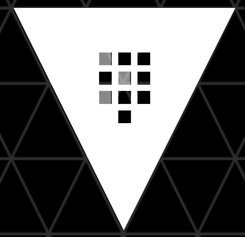


Privileged Users
(Security Engineers, admin, etc.)

Activate the authentication method

Configure the authentication method (varies by the type)

Create and map policies to the authentication method

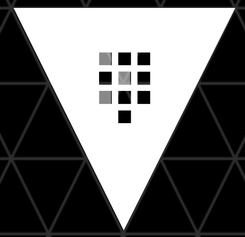


Enable an Auth Method

Command Example

Terminal

```
# CLI command
$ vault auth enable github
Success! Enabled github auth method at: github/
```



Configure Auth Method

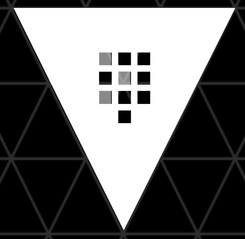
Command Example

Terminal

```
$ vault write auth/github/config organization=hashicorp
Success! Data written to: auth/github/config
```

In addition, you can specify keys like:

- **max_ttl**
- **ttl**



Map Policies

Command Example

Terminal

```
$ vault write auth/github/map/teams/training \
        value="base,training"
Success! Data written to: auth/github/map/teams/training
```

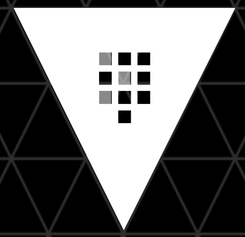
team name within the *organization* (slugified name)

The "**value**" is the comma-separated string of policies to attach to the team.

Authenticate to Vault

- To find out which auth methods are available:

```
$ vault auth list
Path          Type         Accessor                    Description
----          ----         --------                    -----------
approle/      approle      auth_approle_d38fc48c       For application to authenticate
github/       github       auth_github_0d71180c        Use GitHub access token
ldap/         ldap         auth_ldap_78862e3a          Authenticate with LDAP account
token/        token        auth_token_722287e9         token based credentials
userpass/     userpass     auth_userpass_ba4e7771      Username & password
```



Login with GitHub auth method

Command Example

Terminal

```
$ vault login -method=github

GitHub Personal Access Token (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       1X7mbCZAGiAjdX9fsHCGimVA
token_duration       768h
token_renewable      true
token_policies       [base default training]
...
```

Tokens

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 generate 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 can be renewed, revoked, and create children
  - **Batch tokens** are NOT persisted (ephemeral)
    - Tokens are **encrypted binary large objects** (blobs)
    - Carries just enough information

Compare Token Types

Characteristic	Service Tokens	Batch Tokens
Can be root tokens	Yes	No
Can create child tokens	Yes	No
Renewable	Yes	No
Can be periodic	Yes	No
Can have explicit Max TTL	Yes	No (always uses a fixed TTL)
Has accessors	Yes	No
Has Cubbyhole	Yes	No
Revoked with parent (if not orphan)	Yes	Stops Working
Dynamic secrets lease assignment	Self	Parent (if not orphan)
Can be used across Performance Replication clusters	No	Yes
Creation scales with performance standby node count	No	Yes
Performance cost	Heavyweight; multiple storage writes per token creation	**Lightweight**; no storage cost for token creation

Create Tokens

Basic Command for Creating Tokens

- Usage: **`vault token create [options]`**

```
$ vault token create -policy="base" -ttl="24h"
```

  - Attach "base" policy
  - Initial TTL of 24 hours

- API endpoint: **`auth/token/create`**

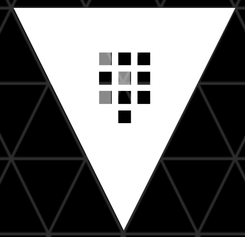
```
$ curl --header "X-Vault-Token:<token>" --request POST \
       --data <payload> \
       <VAULT_ADDRESS>/v1/auth/token/create
```

Token creation

Terminal

```
$ vault token create -policy="test"
Key                  Value
---                  -----
token                s.Cy1cZb3SxPt0AyrItnfsYKOt
token_accessor       1X7mbCZAGiAjdX9fsHCGimVA
token_duration       768h
token_renewable      true
...

$ vault token create -policy="test" -type="batch"
Key                  Value
---                  -----
token                b.AAAAAgKIRy8QpgoGB1S2jW-nqlHz4CWHXMtYwFSoxE8AI...
token_accessor       n/a
token_duration       768h
token_renewable      false
...
```

The default token type is **service** token.



Batch Token

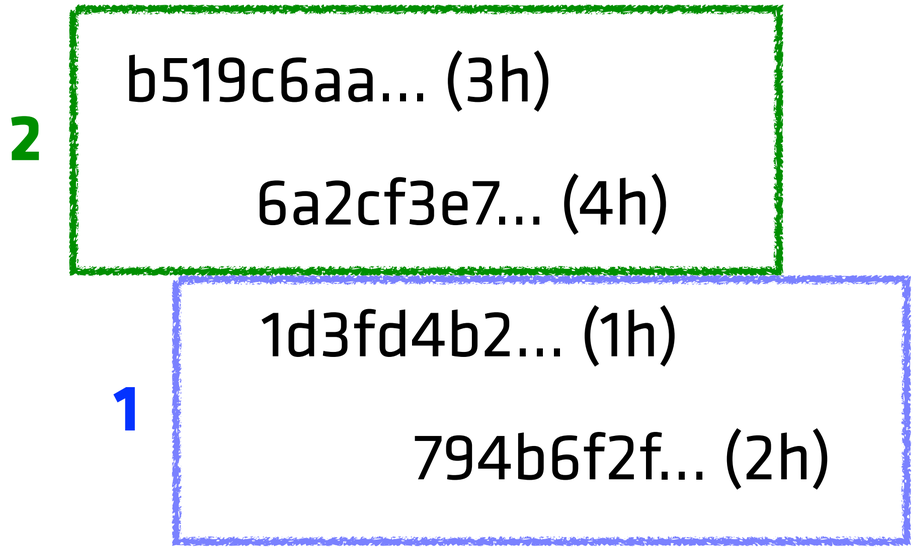
Terminal

```
$ vault token renew <batch_token>

Error renewing token: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/auth/token/renew
Code: 400. Errors:

* batch tokens cannot be renewed

$ vault token revoke <batch_token>

Error renewing token: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/auth/token/renew
Code: 400. Errors:

* batch tokens cannot be revoked
```

Service Tokens

Service Token Lifecycle

- Each **token** has a time-to-live (TTL)
  - Exception: ***root*** token has no TTL
- Tokens are **revoked** once reached its TTL unless **renewed**
  - Once a token reaches its **max TTL**, it gets revoked
  - May be revoked early by "*Break Glass*" procedure
  - When a parent token is revoked, all of its **children are revoked** as well



Controlling Token Lifecycle

Challenge	Solution
You have a *long-running app* which can not handle a regeneration of a token or secret	Periodic Service Tokens
You want to create a token which gets revoked automatically after one use (single-use token)	Service Tokens with Use Limit
You have a case where the lease's expiration is influenced by its *parent* is not desirable	Orphan Tokens

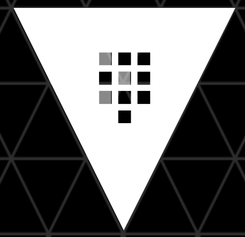
Periodic Service Token

- When having a token be revoked would be problematic:
  - **Root** or **sudo** users have the ability to generate ***periodic*** tokens

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

  - Periodic tokens have a TTL, but **no max TTL**
  - Periodic tokens may live for an infinite amount of time, so long as they are renewed within their TTL

This is useful for long-running services that cannot handle regenerating a token.



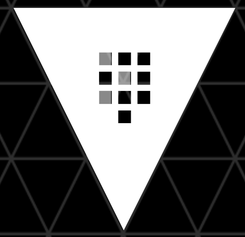
Creating a Periodic Token

Terminal

```
$ vault token create -policy="exercise" -period="24h"

Key                  Value
---                  -----
token                s.2kjqZ12ofDr3efPdtMJ1z5dZ
token_accessor       73rjN1kmnzwT71pMw9H7p6P9
token_duration       24h
token_renewable      true
token_policies       ["default" "exercise"]
identity_policies    []
policies             ["default" "exercise"]
```

Service Token with Use Limits

- When you wish to limit the number of requests coming to Vault from a token:
  - Limit the token's number of uses in addition to TTL and Max TTL
  - Use limit tokens **expire at the end of their last use**, regardless of their remaining TTLs
  - Use limit tokens **expire at the end of their TTLs**, regardless of remaining uses



Creating a Use Limit Token

Terminal

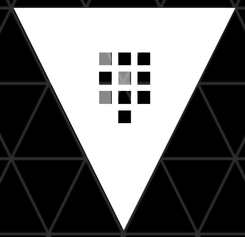
```
$ vault token create -policy="exercise" -use-limit=2

Key                  Value
---                  -----
token                s.516LO9Ssk1CQzvKo8ny1G0eu
...

$ vault token lookup s.516LO9Ssk1CQzvKo8ny1G0eu

Key                  Value
---                  -----
...
id                   s.516LO9Ssk1CQzvKo8ny1G0eu
issue_time           2018-12-13T18:35:08.004652-08:00
meta                 <nil>
num_uses             2
```

Orphan Token

- When the token hierarchy behavior is ***not*** desirable:
  - **Root** or **sudo** users have the ability to generate **orphan** tokens

```
path "auth/token/create-orphan" {
    capabilities = [ "create", "read", "update", "delete", "sudo" ]
}
```

  - Orphan tokens are **not** children of their parent; therefore, do not expire when their parent does
  - Orphan tokens still expire when their own Max TTL is reached



Creating an Orphan Token

Terminal

```
$ vault token create -policy="exercise" -orphan

Key                  Value
---                  -----
token                s.3rPJCQbGWD9O6uybtTuojjFs
...

$ vault token lookup s.3rPJCQbGWD9O6uybtTuojjFs

Key                  Value
---                  -----
...
id                   s.3rPJCQbGWD9O6uybtTuojjFs
issue_time           2018-12-13T18:35:41.02532-08:00
meta                 <nil>
num_uses             0
orphan               true
...
```

Set Token Type

Set Token Type

- To configure the AppRole auth method to generate **batch** tokens:

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

- To configure the AppRole auth method to generate **periodic** tokens:

```
$ vault write auth/approle/role/jenkins policies="jenkins" \
        period="72h"
```

Knowledge Check & Lab Overview

Knowledge Check Questions

1. Which of the following statements are true about the following code?

```
vault token create -use-limit=5 -ttl=3600
```

   A. Throws an error. Missing policy
   ✓ B. Created token has the same policy as its parent token
   C. Throws an error. Can not specify both use limit and TTL
   ✓ D. Revoked automatically when its parent is revoked
   ✓ E. Revoked when its use limit is reached regardless of the TTL

2. True or False: Only the root or sudo users can create an orphan token.

Lab 6: Authentication and Tokens

20 minutes

- Open the Lab Book PDF

- In this lab, you are going to perform the following tasks:
  - Task 1: Create Short-Lived Tokens
  - Task 2: Token Renewal
  - Task 3: Create Tokens with Use Limit
  - Task 4: Create a Token Role and Periodic Token
  - Task 5: Create an Orphan Token
  - Task 6: Enable Username & Password Auth Method
  - Challenge: Generate batch tokens

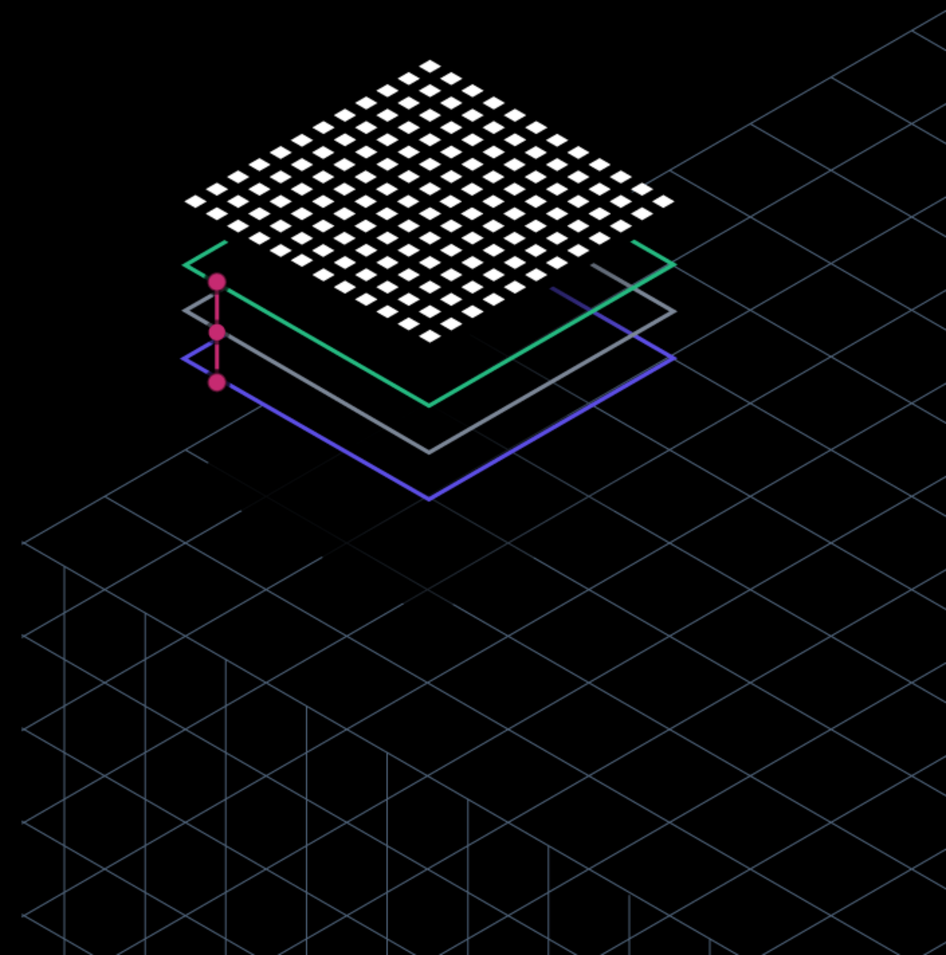
Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

HashiCorp

Application Integration



Agenda

- Vault Agent
  - Vault Agent Auto-Auth
  - Vault Agent Caching
- Direct Application Integration
  - Envconsul
  - Consul Template
  - Vault Agent Templates

Now Vault became the single source of secrets!

What's the next step to complete the Vault integration?

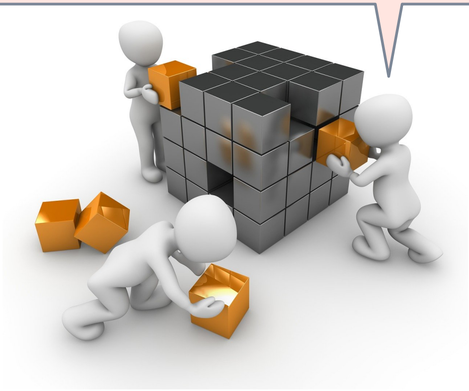
Vault Adoption Journey

- An application acting as a secret consumer must:
    1. Authenticate and acquire a client token
    2. Manage the lifecycle of the token
    3. Retrieve secrets from Vault
    4. Manage the leases of any dynamic secrets

How is this implemented?

Implement/Import a client of the Vault REST API

Does this mean we have to update all of our applications?



Vault Agent

Vault Agent

Vault Agent is client daemon that provides the following features:

- **Auto-Auth** automatically authenticate and manage the token renewal process for locally-retrieved dynamic secrets.

- **Caching** Allows client-side caching of responses containing newly created tokens and responses containing leased secrets generated off of these newly created tokens.

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 Configuration (1 of 2)

- **pid_file**
  is where the process ID (PID) should be stored

- **vault**
  specifies the **address** and other connection information for the target Vault

Example: agent-config.hcl

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

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

 // TODO: Authentication
```

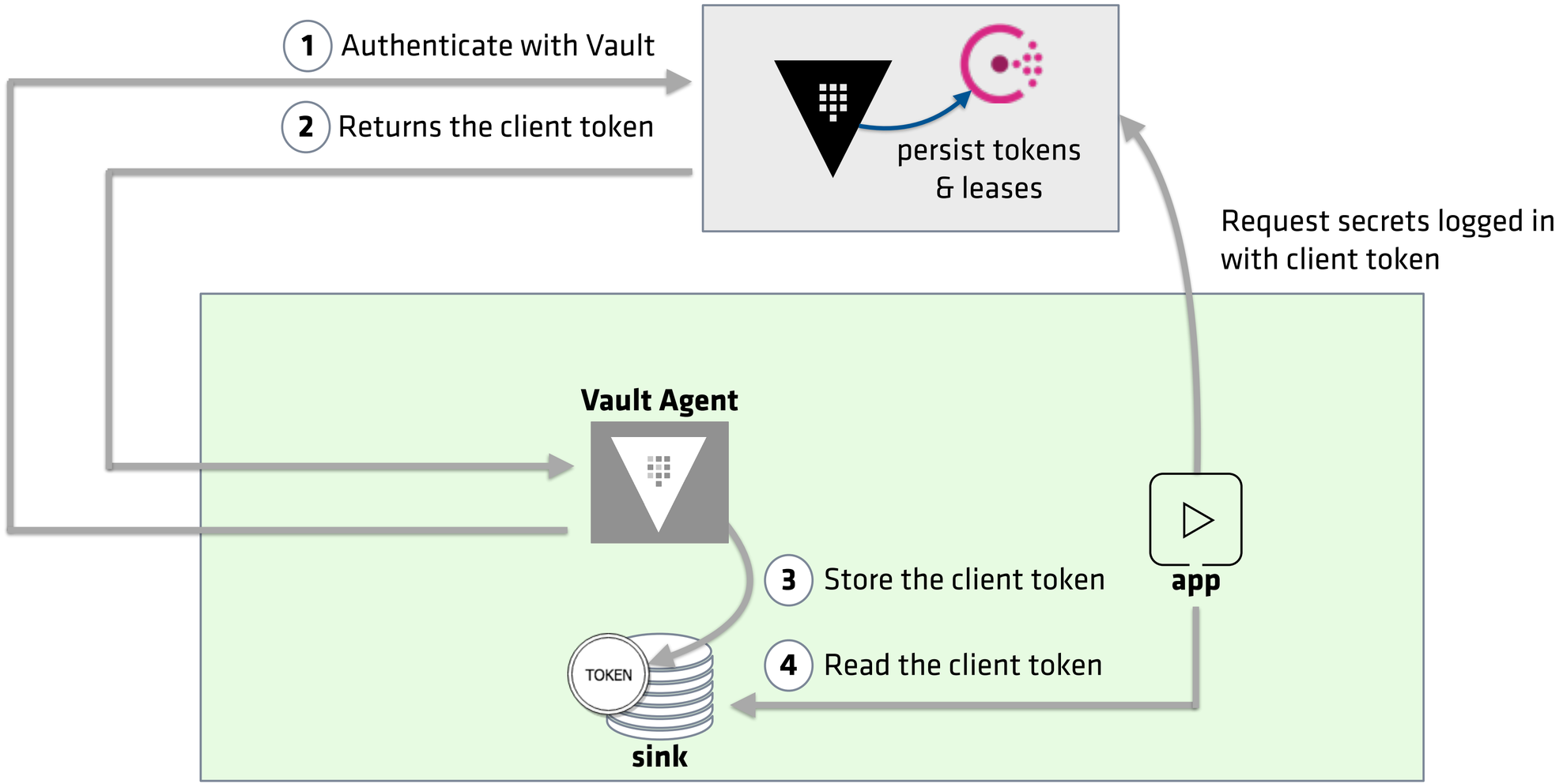
Auto-Auth

automatically authenticate and manage the token renewal process for locally-retrieved dynamic secrets.

Auto-Auth

- Vault Agent's **auto-auth** allows for easy authentication to Vault:
  - Automatically **authenticates** using supported auth methods
  - Keeps the client token **renewed** until the token is revoked
  - Client tokens can be **response wrapped**

Auto-Auth Basic Workflow



Vault-Agent Configuration

- **auto_auth** requires:
  - **method**
    the authentication method
  - **sink**
    the location(s) where the agent writes the token

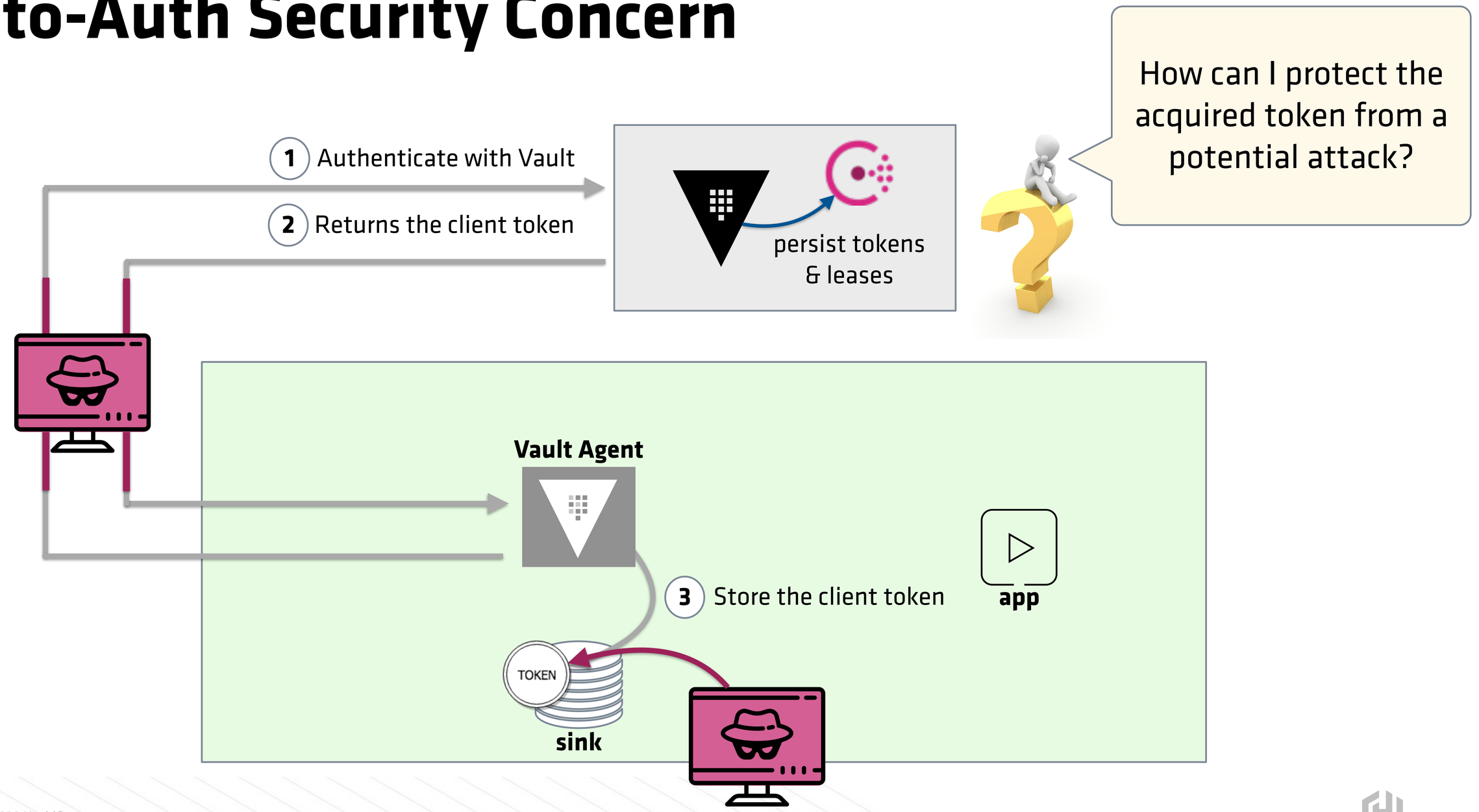
Example: agent-config.hcl

```
...

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

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

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

The acquired client token is written to this location

Auto-Auth Security Concern



Response-Wrapping Tokens (1 of 2)

Response wrapped by the **method**

- **Pros:** Prevents man-in-the-middle (MITM) attack (more secure)
- **Cons:** Vault agent cannot renew the token

Response wrapped by the **sink**

- **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

Response-Wrapping Tokens (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"
        }
    }
}
```

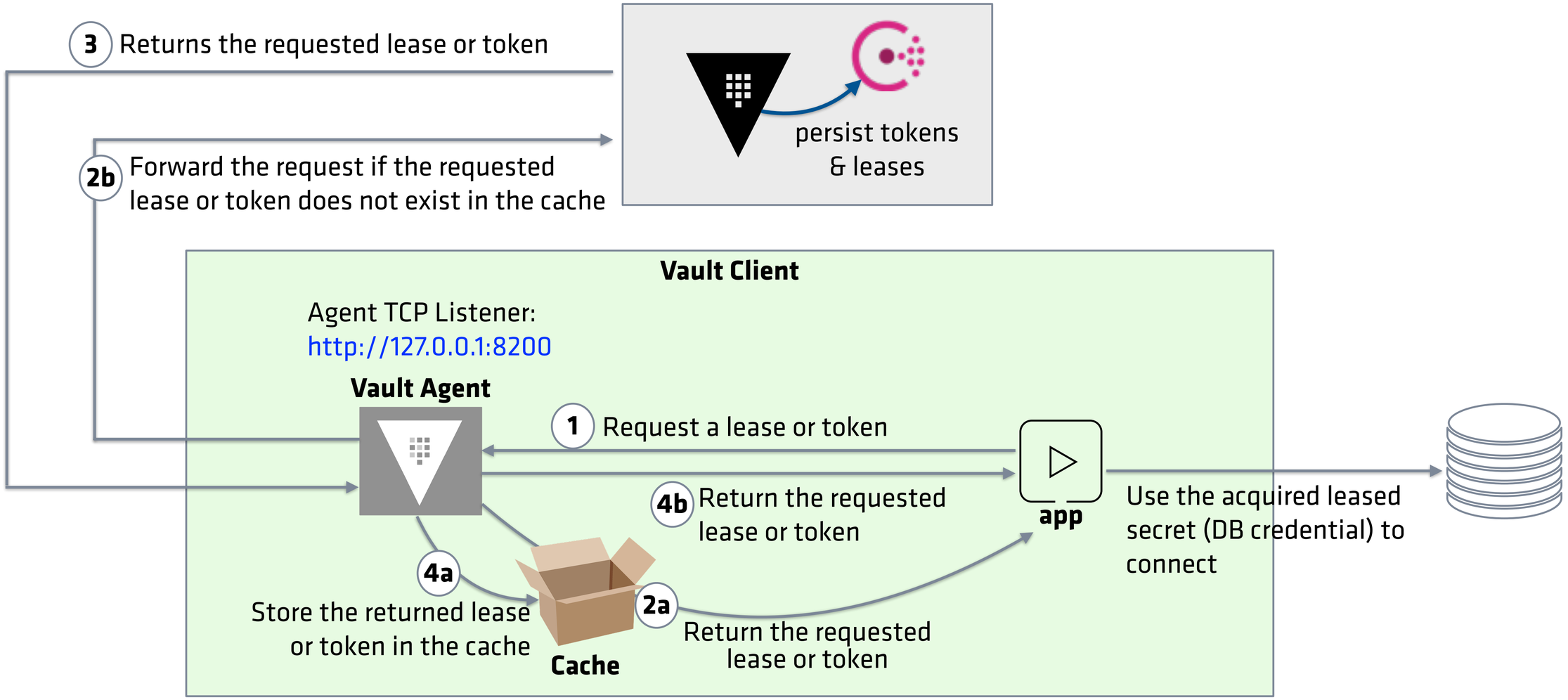

Caching

Allows client-side caching of responses containing newly created tokens and their leased secrets.

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

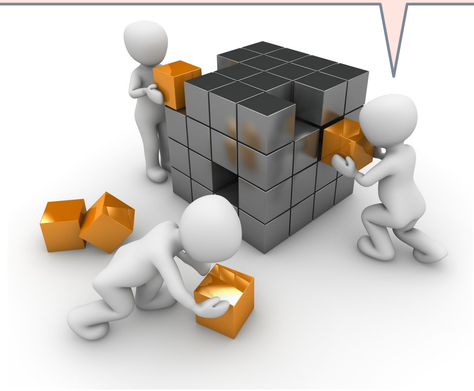
```
auto_auth {
    ...
}

cache {
   use_auto_auth_token = true
}

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

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


Integrating your Apps

Vault Adoption Journey

- An application acting as a secret consumer must:
  - ✅ Authenticate and acquire a client token
  - ✅ Manage the lifecycle of the token
  - ⬜ Retrieve secrets from Vault
  - ✅ Manage the leases of any dynamic secrets

Even if Vault Agent manages tokens and leases this application still requires code changes to request those secrets!



Example Application Configuration

```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.service.consul
  username: san-francisco
  password: p@ssw0rd
```

When you have a file or code containing secrets, it makes better sense to manage them in Vault.

Create Vault Unaware Applications

• **Envconsul**	Requests secrets and creates environment variables.
• **Consul Template**	Requests secrets and renders those secrets to a templated file once or continuously as a daemon.
• **Vault-Agent Templates**	A subset of Consul Template features embedded into Vault-Agent.

Envconsul

Requests secrets and creates environment variables

Envconsul

- Envconsul launches a subprocess with **environment variables** populated from Consul and Vault
  - https://github.com/hashicorp/envconsul
  - Environment variables are dynamically populated
    - Applications read those environment variables
  - Envconsul does **not** require a Consul cluster to operate
- Envconsul enables flexibility and portability for applications across systems

Envconsul Usage (1 of 2)

- Your application should read secrets from environment variables

Example template file: **`config.yml`**

```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.service.consul
  username: ${DATABASE_CREDS_READONLY_USERNAME}
  password: ${DATABASE_CREDS_READONLY_PASSWORD}
```

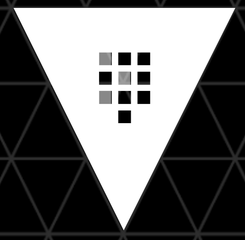
Envconsul Usage (2 of 2)

- Run Envconsul
  - Render from secret **`path`** to **`file`**

```
$ envconsul -secret <path> <file>
```

  - To communicate with Vault, you need to pass a valid Vault token with appropriate permissions:

```
$ VAULT_TOKEN="..." envconsul -secret <path> <file>
```



Envconsul Demo

config.cfg

```
$ VAULT_TOKEN="s.8a7d2115-c065..." \
    envconsul -upcase \
    -secret database/creds/readonly env | grep DATA


DATABASE_CREDS_READONLY_PASSWORD=A1a-2klp36AHFf...
DATABASE_CREDS_READONLY_USERNAME=v-token-readonly...
```

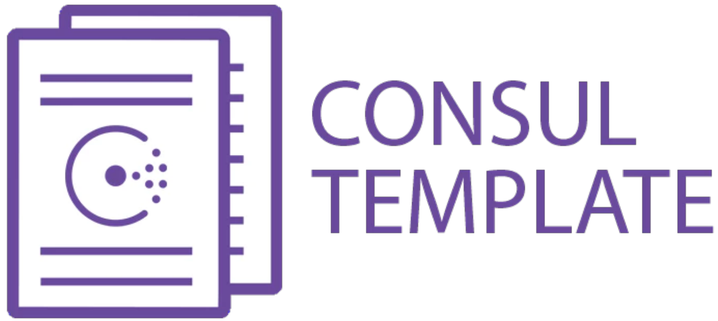
Consul Template

Requests secrets and renders those secrets to a templated file once or continuously as a daemon.

Consul Template

- A stand-alone application that **renders data** from Consul and Vault onto the target file system
  - https://github.com/hashicorp/consul-template
  - Despite its name, Consul Template does **not** require a Consul cluster to operate

- Consul Template retrieves secrets from Vault
  - Manages the acquisition and renewal lifecycle
  - **Requires a valid Vault token to operate**



Consul Template Usage (1 of 2)

- Create a **templated** file

Example template file: **config.yml.tpl**

```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.service.consul
  {{ with secret "database/creds/readonly" }}
  username: "{{ .Data.username }}"
  password: "{{ .Data.password }}"
  {{ end }}
```

The secrets to be read from Vault

Consul Template Usage (2 of 2)

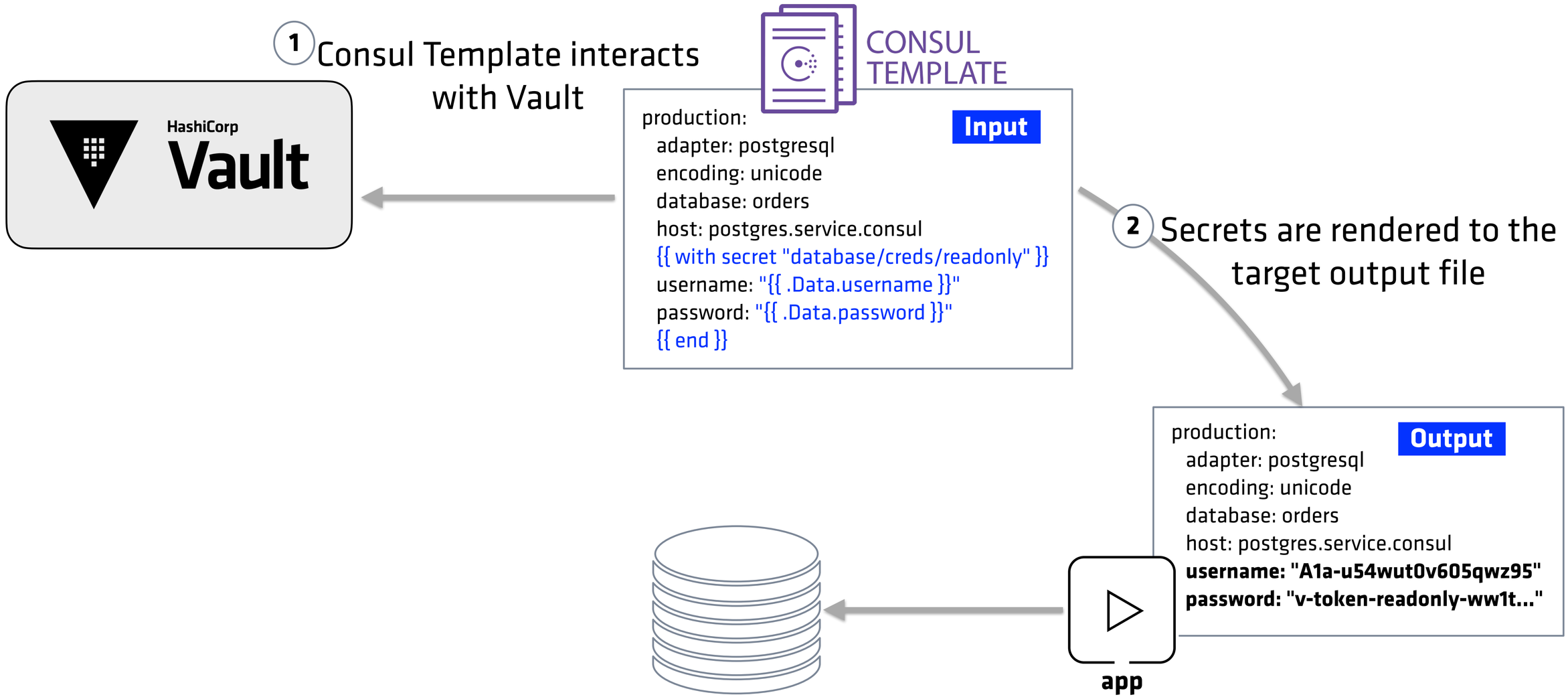
- To run the Consul Template:
  - Render from **input_file** to **output_file**

```
$ consul-template -template "<input_file>:<output_file>"
```

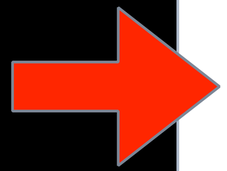
  - To communicate with Vault, you need to pass a valid Vault token with appropriate permissions:

```
$ VAULT_TOKEN="..." consul-template -template "in.tpl:out.file"
```

Envconsul Basic Workflow



Run Consul Template

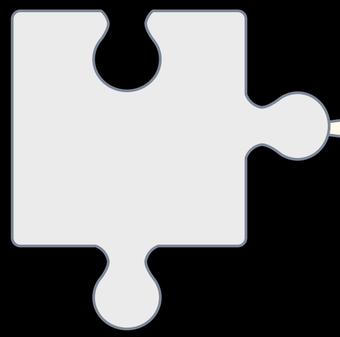


Terminal

```
$ VAULT_TOKEN="s.8a7d2115-c065..." \
    consul-template -template="config.yml.tmpl:config.yml" \
    -once

$ cat config.yml
 production:
   adapter: postgresql
   encoding: unicode
   database: orders
   host: postgres.service.consul
   password: A1a-u54wut0v605qwz95
   username: v-token-readonly-ww1tq33s7z5uprpxxy68-1527631219
```

Input

Output



Consul Template allows my applications to be "*Vault-unaware*" which is nice!

Vault Agent Auto-Auth can acquire a client token which Consul Template can use!

But wait! Do I still need to worry about scripting Consul Template to read the token from the agent's sink?

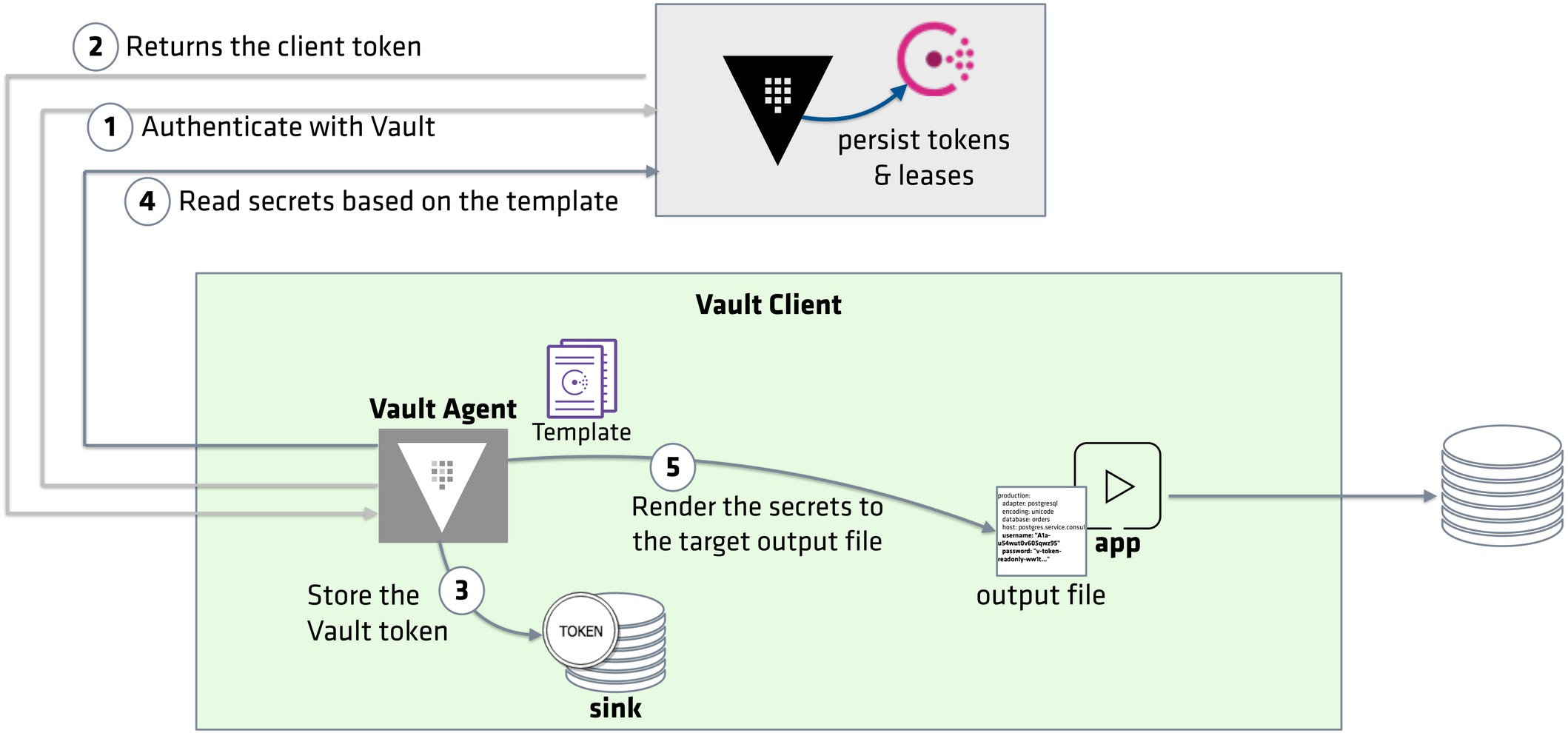
Vault Agent Templates

A subset of Consul Template features embedded into Vault-Agent.

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

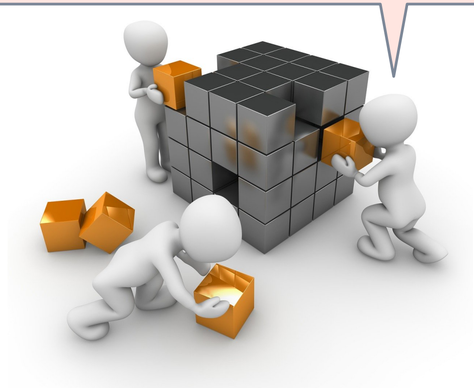
Example: agent-config.hcl

```
auto_auth {
    ...
}

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

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


Vault Adoption Journey

- An application acting as a secret consumer must:
  - ✅ Authenticate and acquire a client token
  - ✅ Manage the lifecycle of the token
  - ✅ Retrieve secrets from Vault
  - ✅ Manage the leases of any dynamic secrets

Vault Agent with **auto-auth** and the suite of **templating** tools means we can leave this application as-is!



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

Knowledge Check & Lab Overview

Knowledge Check Questions

1. True or False: Vault Agent can have only one sink to store the client token.

2. True or False: If you are concerned about MITM attack, you would need to implement a solution to protect the acquired token because Vault Agent does not provide that feature.

3. Which statements are true about *Consul Template* and *Envconsul*?

   A. Your Vault server must use Consul as its storage backend to leverage either of the tools

   ✓ B. Envconsul requires minimum to no code change if your app is designed to read environment variables

   C. You need to give those tools a highly privileged token to work

   ✓ D. Using these tools eliminate the need to invoke Vault API in your code to retrieve secrets

Lab 7: Direct Application Integration

35 minutes

- Open the Lab Book PDF

- In this lab, you are going to:
  - Task 1: Run Vault Agent
  - Task 2: Use Envconsul to populate DB credentials
  - Task 3: Use Consul Template to populate DB credentials
  - Task 4: Use Vault Agent Templates

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com