Introduction to Vault

What is Vault?

HashiCorp Terraform

HashiCorp Vault

HashiCorp Consul

HashiCorp Nomad

HashiCorp Boundary

HashiCorp Packer

HashiCorp Vagrant

HashiCorp Waypoint

What is Vault?

- Manage Secrets and Protect Sensitive Data
- Provides a Single Source of Secrets for both Humans and Machines
- Provides Complete Lifecycle Management for Secrets
  - Eliminates secret sprawl
  - Securely store any secret
  - Provide governance for access to secrets

- What is a Secret?
  - Anything your organization deems sensitive:
    - Usernames and passwords
    - API keys
    - Certificates
    - Encryption Keys

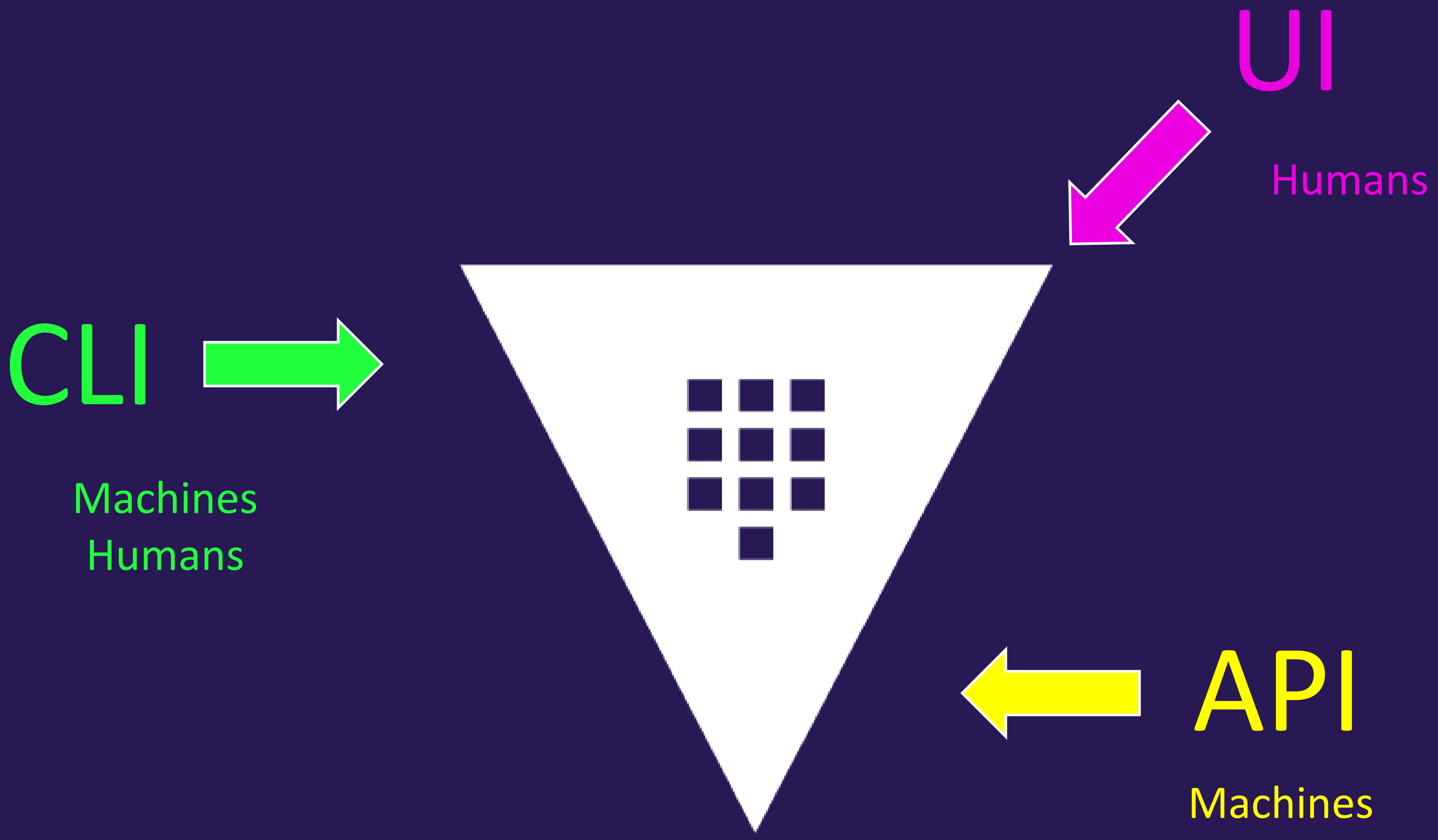
HashiCorp Vault

How Vault Works

Hotel Atlantis Paradise Island – The Bahamas

How Vault Works

Vault Interfaces



UI

Humans

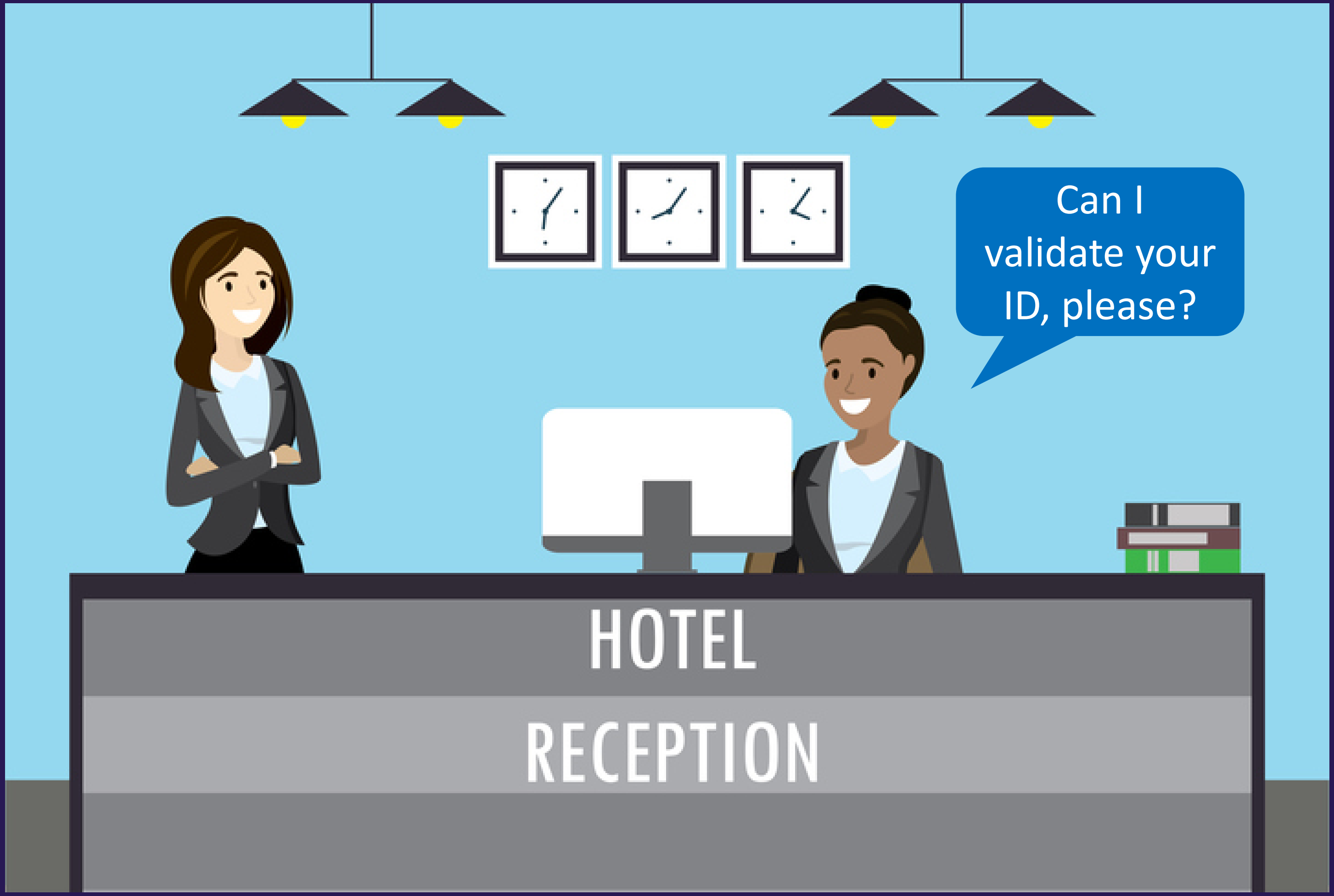
CLI

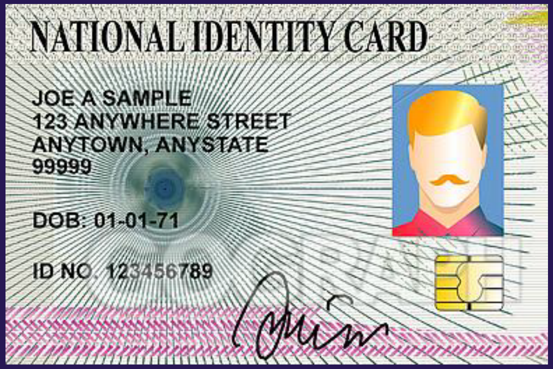
Machines
Humans

API

Machines

Vault Authentication

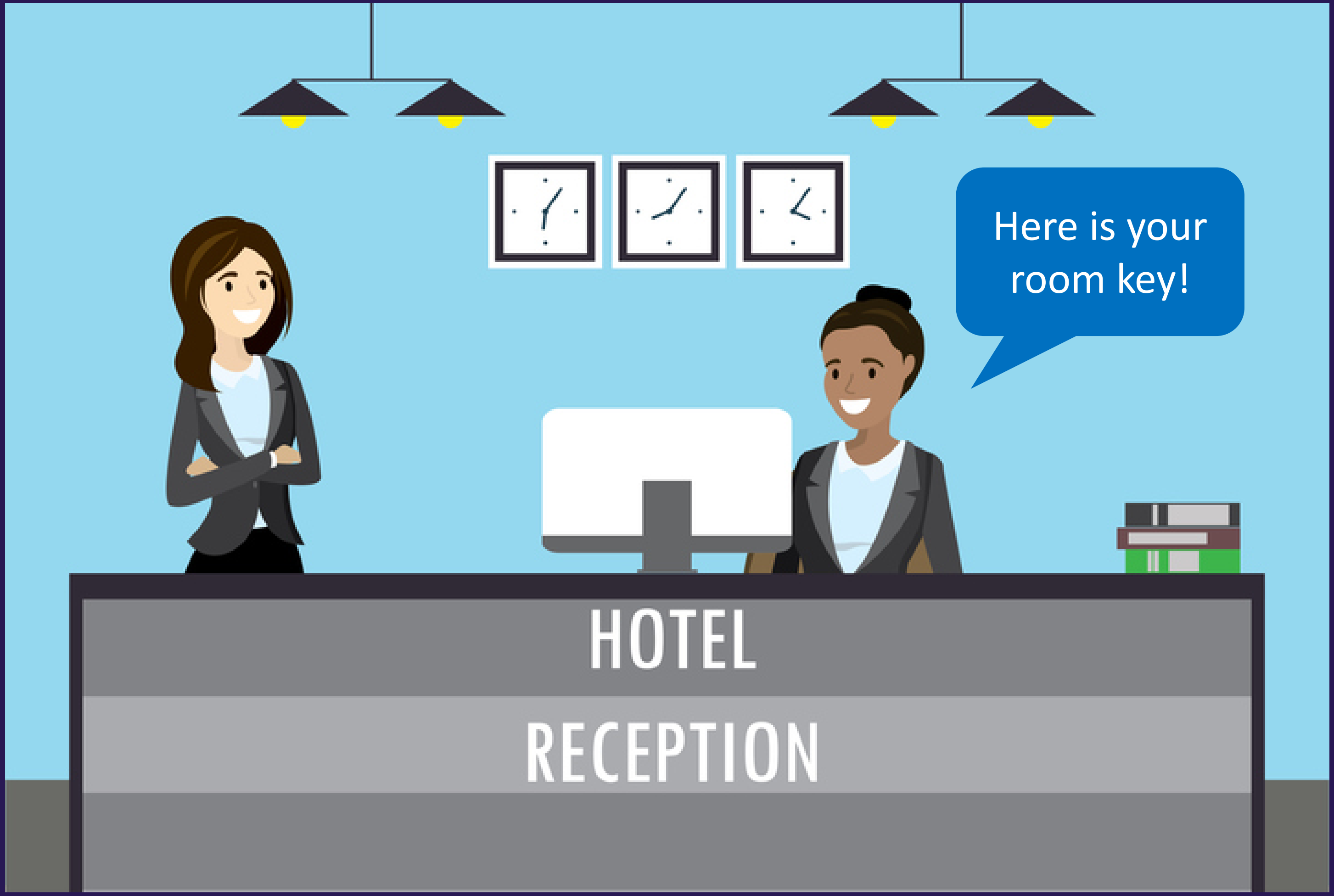


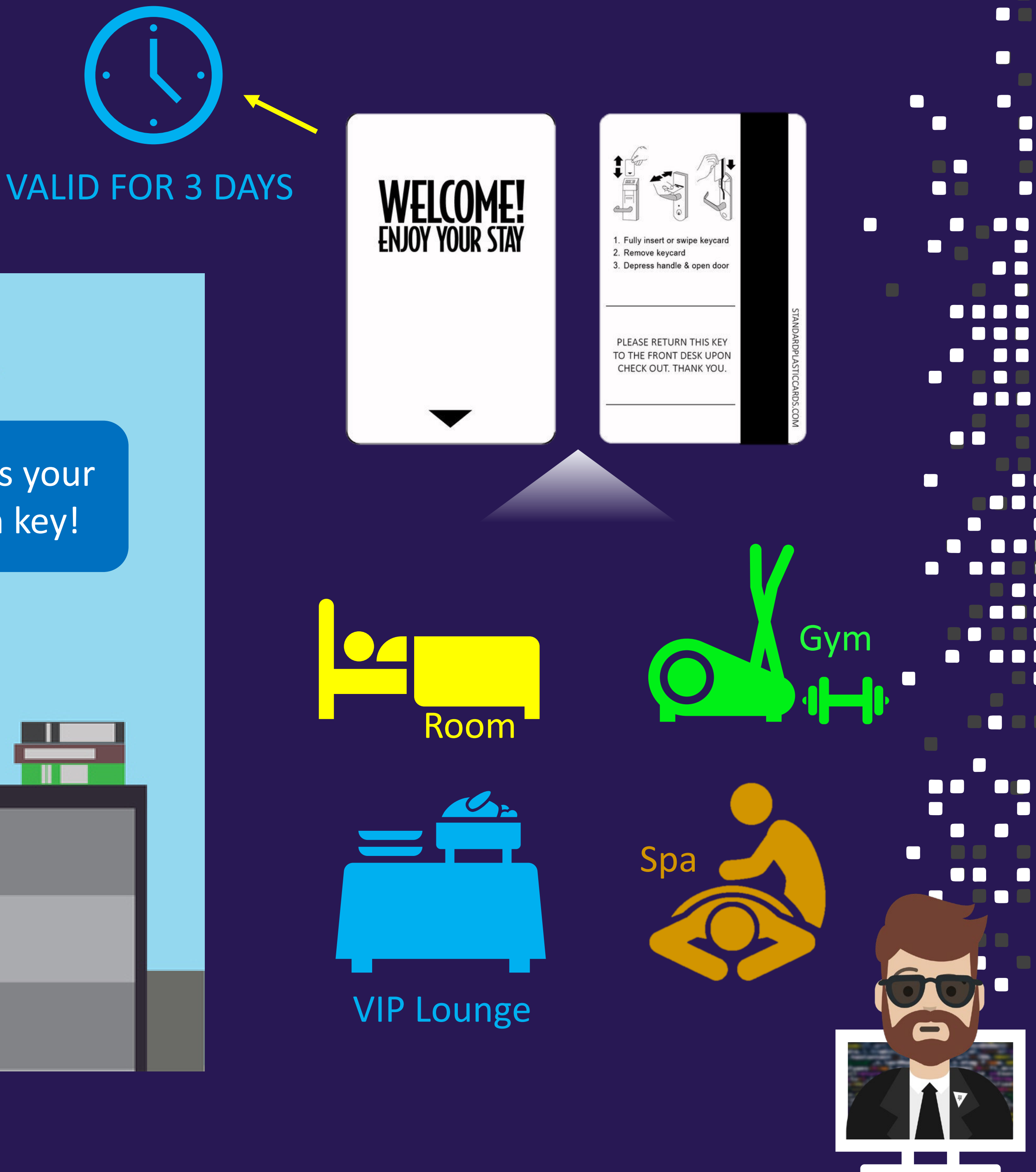


Vault Interfaces

Authentication

Vault Interfaces




VALID FOR 3 DAYS
WELCOME!
ENJOY YOUR STAY
1. Fully insert or swipe keycard
2. Remove keycard
3. Depress handle & open door
PLEASE RETURN THIS KEY TO THE FRONT DESK UPON CHECK OUT. THANK YOU.
STANDARDPLASTICCARDS.COM
Room
Gym
VIP Lounge
Spa

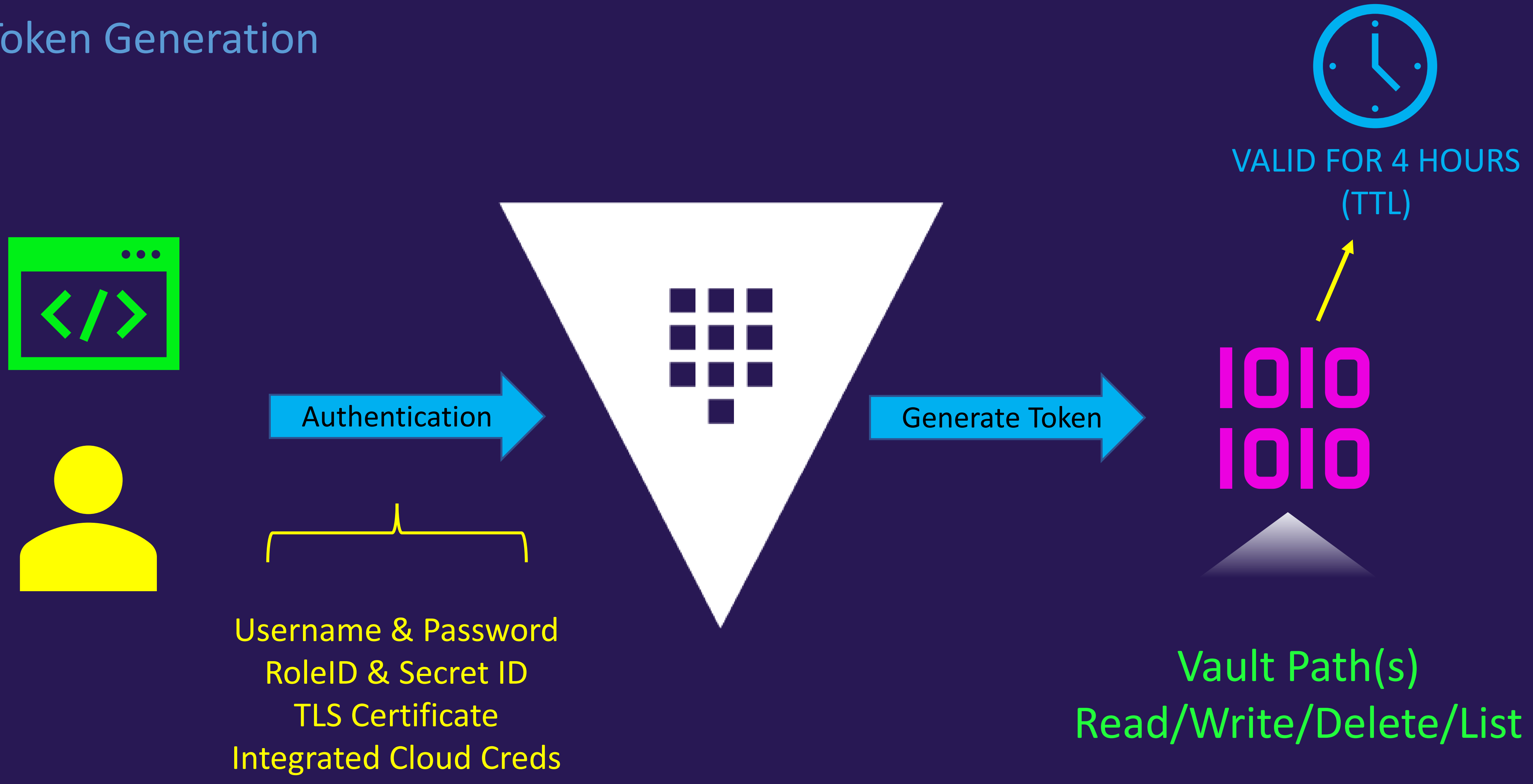
Vault Interfaces

WELCOME!
ENJOY YOUR STAY
1. Fully insert or swipe keycard
2. Remove keycard
3. Depress handle & open door
PLEASE RETURN THIS KEY TO THE FRONT DESK UPON CHECK OUT. THANK YOU.
STANDARDPLASTICCARDS.COM

We present our key.
We don't authenticate again

Vault Interfaces

Token Generation



Authentication

Username & Password
RoleID & Secret ID
TLS Certificate
Integrated Cloud Creds

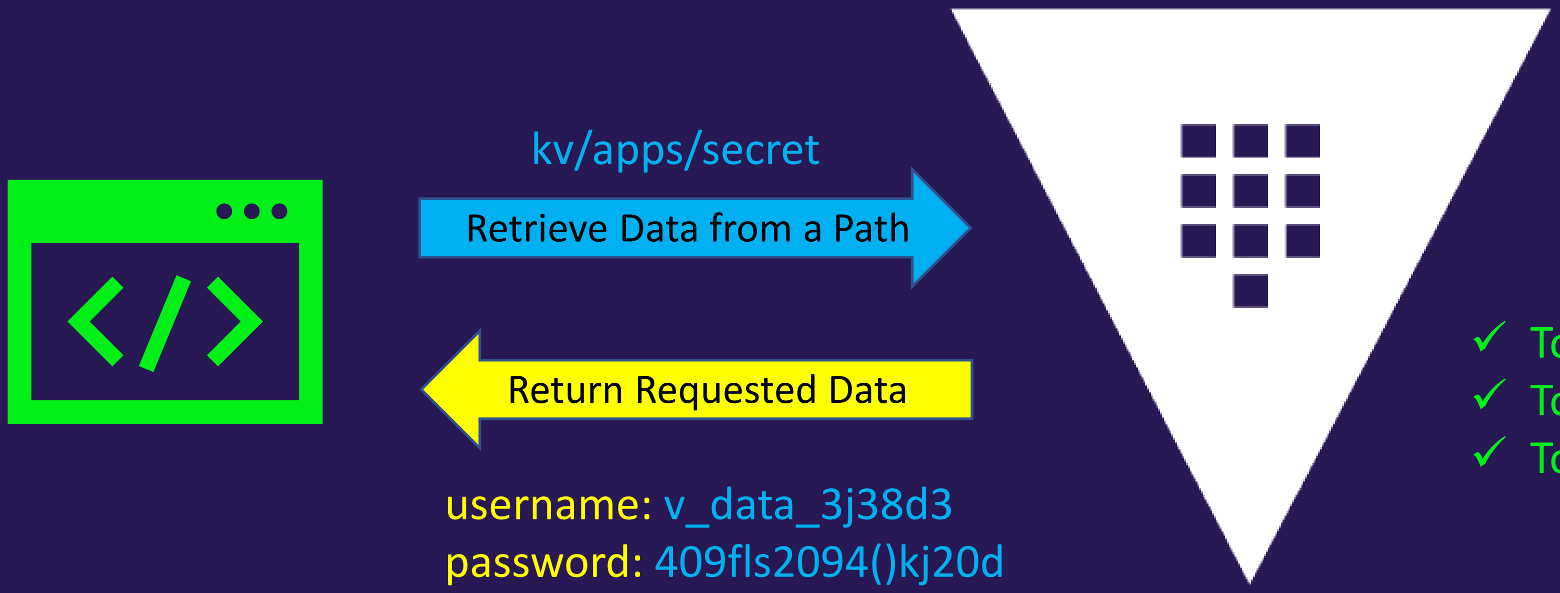
Generate Token

VALID FOR 4 HOURS
(TTL)

Vault Path(s)
Read/Write/Delete/List

Vault Interfaces

Token Usage



kv/apps/secret

Retrieve Data from a Path

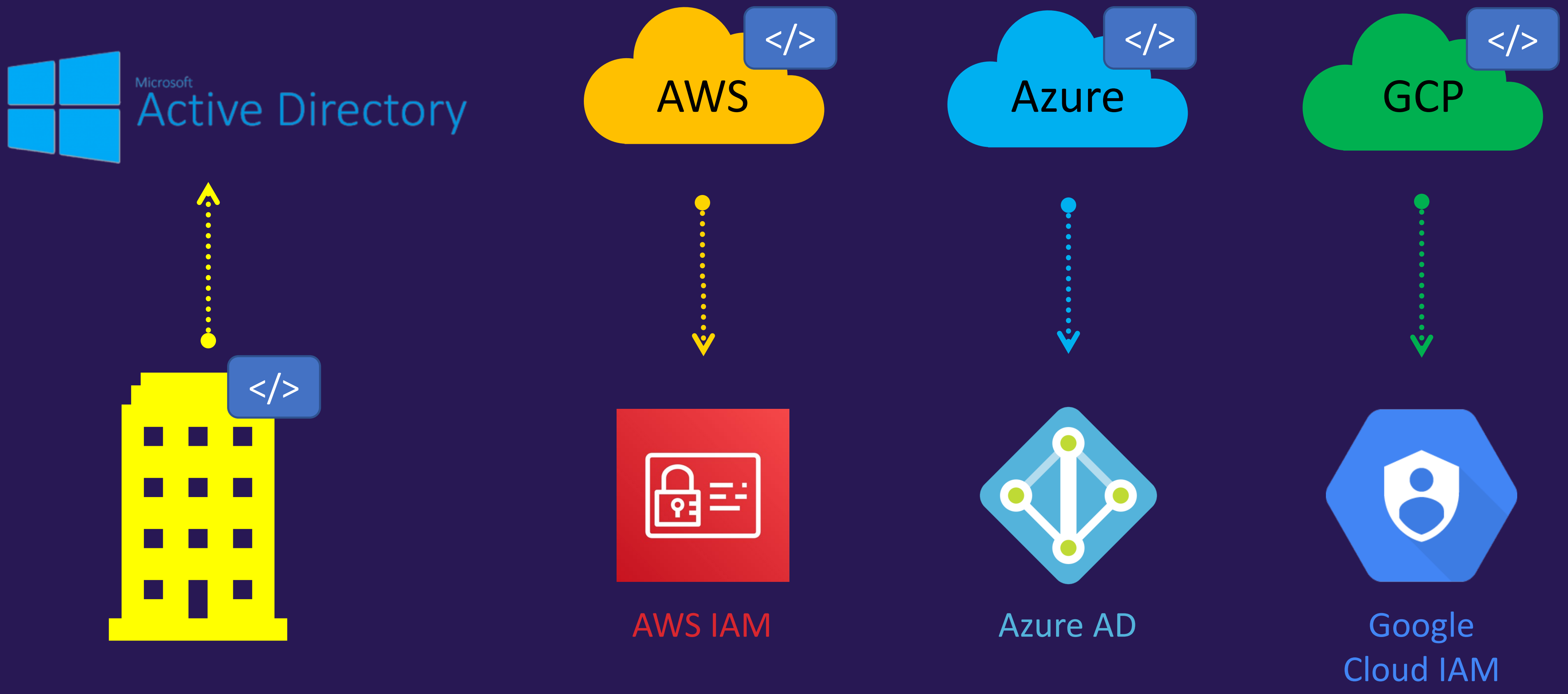
Return Requested Data

username: v_data_3j38d3
password: 409fls2094()kj20d

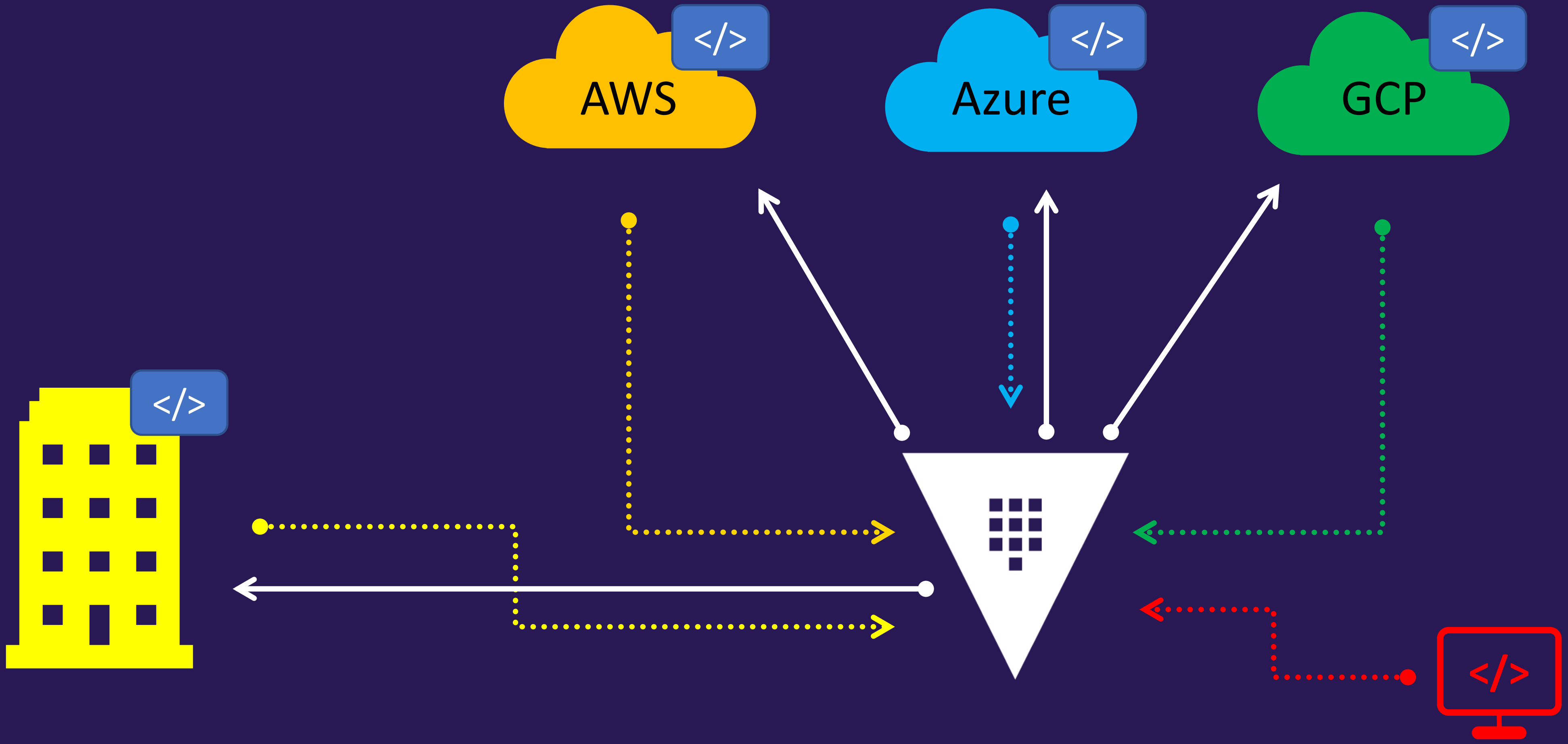
- ✓ Token is Valid
- ✓ Token is not Expired
- ✓ Token has Permission

We present our token.
We don't authenticate again

Why Organizations Choose Vault



Why Organizations Choose Vault



Benefits of HashiCorp Vault

Store Long-Lived, Static Secrets

Dynamically Generate Secrets, upon Request

Fully-Featured API

Identity-based Access Across different Clouds and Systems

1010
1010

Provide Encryption as a Service

Act as a Root or Intermediate Certificate Authority

Use Cases

Secure Data with a centralized workflow for Encryption Operations

Migrate to Dynamically Generated Secrets

Automate the Generation of X.509 Certificates

Centralize The Storage Of Secrets

Migrate to Identity-Based Access

Use Case – Storage of Secrets

CHEF
Databags

Jenkins
Credentials

AWS Secrets Manager

Azure Key Vault

Centralize the storage of secrets across the organization into a consolidated platform

KEY/VALUE

Use Case – Migrate to Dynamic Credentials

👎

Static Credential

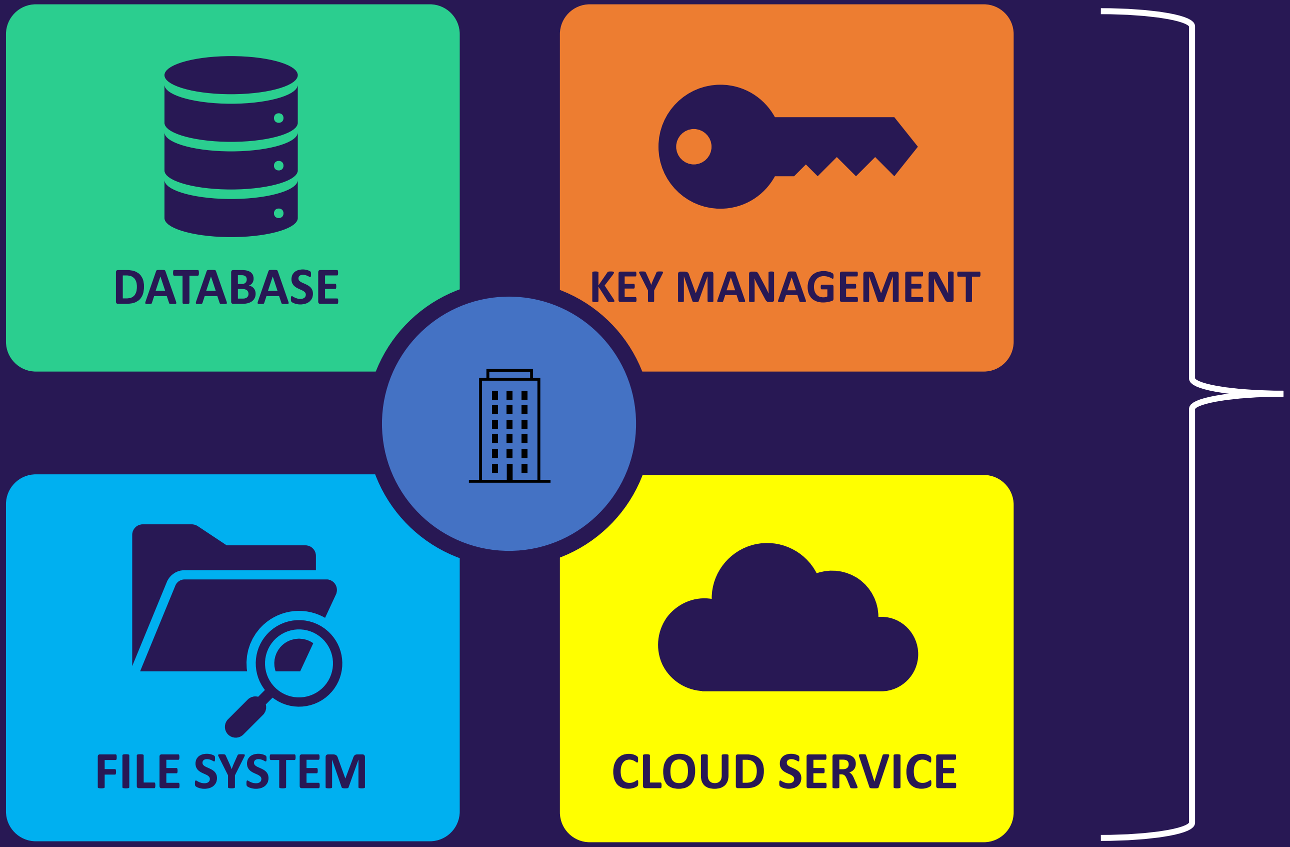
- Validate 24/7/365
- Long-Lived
- Manual Password Rotation
- Frequently Shared Across the Team
- Reused Across Systems
- Susceptible to Being Added to Code/Repo
- Often Highly Privileged
- Manually Created by Human

👍

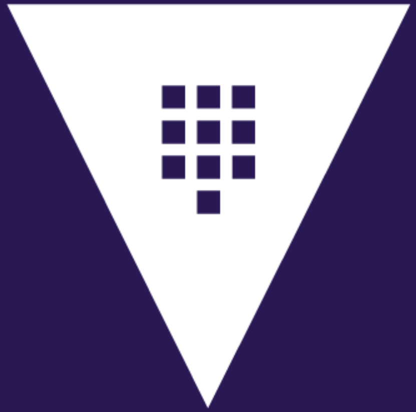
Dynamic Credential

- Short-Lived
- Follows Principal of Least Privilege
- Automatically Revocated (Based on Lease)
- Each System Can Retrieve Unique Credentials
- Programmatically Retrieved
- No Human Interaction

Use Case – Encrypt Data



Secure Data with a centralized workflow for Encryption Options



Secrets Engines

- Transit
- Key Mgmt
- KMIP
- Transform

Use Case – Automate X.509 Certificates

BEFORE VAULT

GENERATE A CSR

ENTER TICKET FOR CERT CREATION

SUBMIT CSR TO SIGNING CA

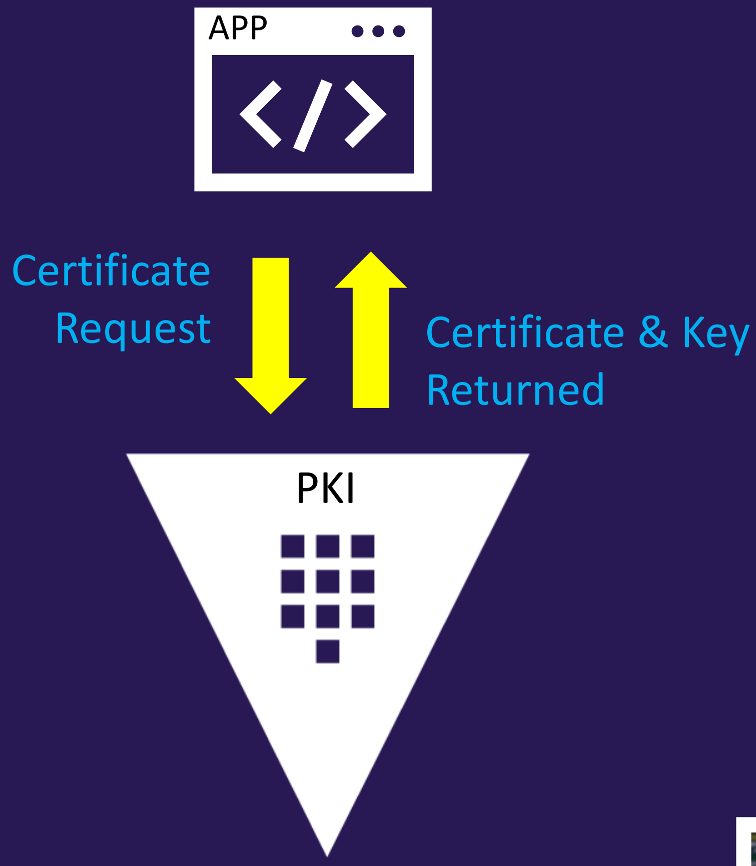
RETRIEVE THE CERTIFICATE & KEY

CERTIFICATE IS RETURNED (TICKET CLOSED)

ENGINEER UPLOADS CERTIFICATE AND PRIVATE KEY

RENEWAL FOLLOWS THE SAME PROCESS

USING VAULT



APP

</>

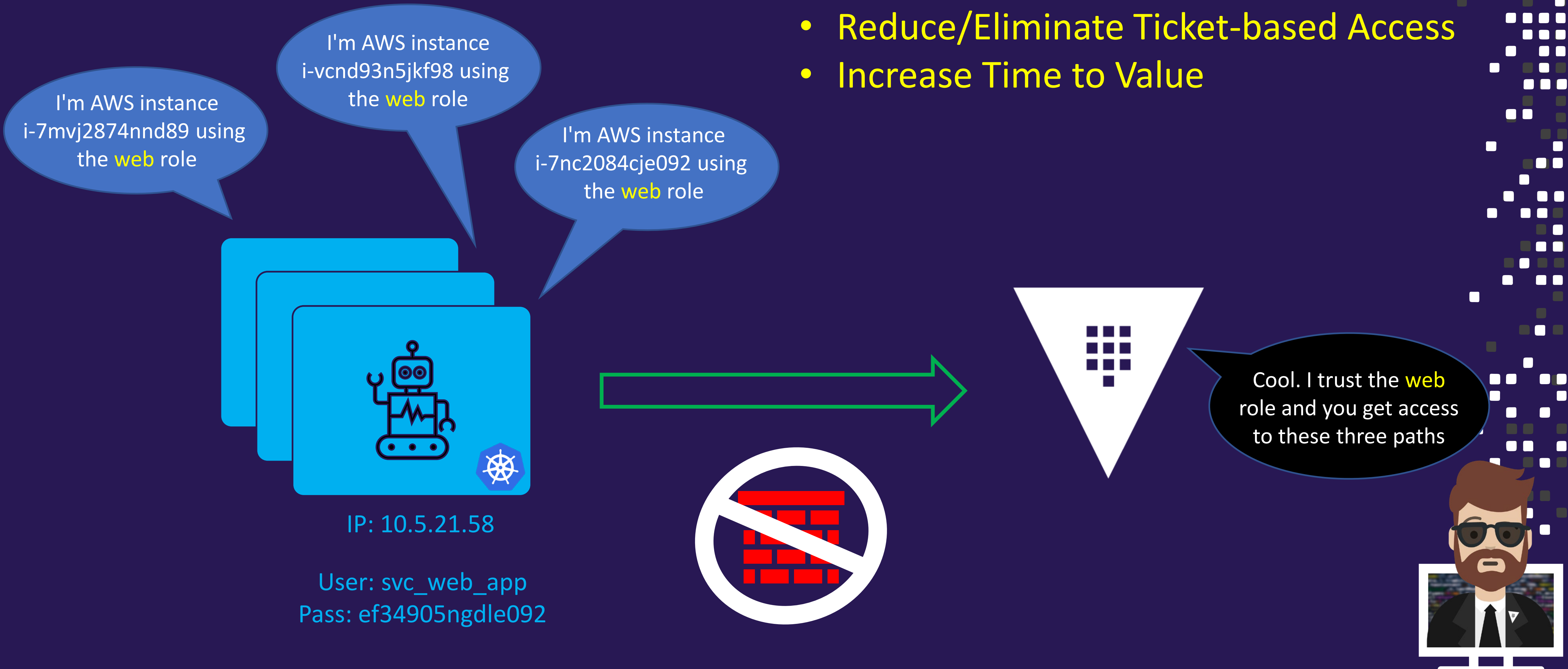
Certificate Request

Certificate & Key Returned

PKI

Use Case – Migrate to Identity-Based Access

- Quickly Scale Up and Down
- Reduce/Eliminate Ticket-based Access
- Increase Time to Value



I'm AWS instance i-7mvj2874nnd89 using the web role

I'm AWS instance i-vcnd93n5jkf98 using the web role

I'm AWS instance i-7nc2084cje092 using the web role

IP: 10.5.21.58

User: svc_web_app
Pass: ef34905ngdle092

Cool. I trust the web role and you get access to these three paths

Vault – Compare Versions

FREE Open Source	Enterprise	Vault on HCP
Dynamic Secrets	Disaster Recovery	Hosted by HashiCorp
ACL Templates	Namespaces	Fully Managed Solution
Init & Unseal Workflow	Replication	Reduce Admin Burden
Vault Agent	Read Replicas	Scalable
Key Rolling	HSM Auto-Unseal	Push Button Deployment
Access Control Policies	MFA	Pay by the Hour
Encryption as a Service	Sentinel	All Enterprise Features
AWS, Azure, & GCP Auto Unseal	FIPS 140-2 & Seal Wrap	Dev or Prod Options

Self-Hosted and Managed

HashiCorp Hosted & Managed

Vault – Open Source

Includes:

- Incredible number of features and integrations
- Local high-availability by way of clustering
- Almost all secrets engines and auth methods
- Can easily integrate with any application using fully-featured API

Does Not Include:

- No Replication capabilities = single datacenter/cloud deployment
- Does not include access to Enterprise integrations (MFA, HSM, Automated Backups)
- Limited Scalability

Vault – Enterprise

Includes:

- Access to all* features and functions Vault offers
- Replication capabilities to other Vault clusters across datacenters/clouds
- All secrets engines and auth methods
- Can easily integrate with any application using fully-featured API
- Namespaces for multi-tenancy solution
- Policy as Code using Sentinel
- Easily scale local reads using Performance Standbys
- Access to the Raft/Consul snapshot agent for automated disaster recovery solution

Does Not Include:

- Self-Managed - Not hosted or managed by HashiCorp

Vault – Enterprise

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

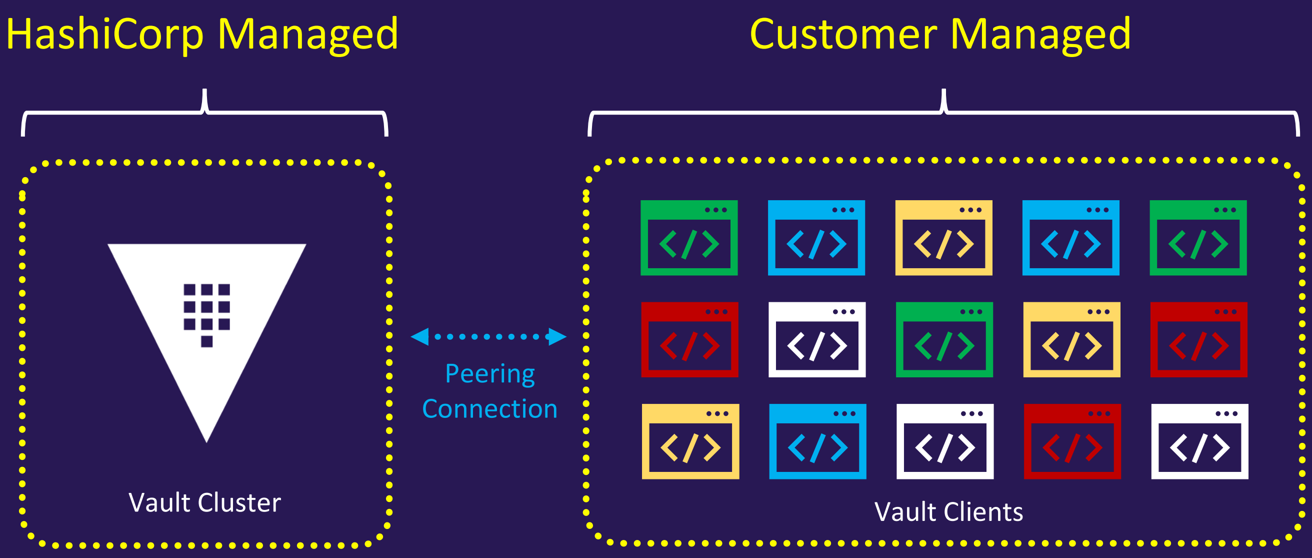
*Based on Current Version - Subject to Change

hashicorp.com/products/vault/pricing

Vault on HashiCorp Cloud Platform (HCP)

Includes:

- All features of Vault Enterprise
- Fully managed solution
- Click button deployment
- HashiCorp team of Vault experts manages and upgrades your cluster(s)



cloud.hashicorp.com

Vault Components

Storage Backends

Secrets Engines

Authentication Methods

Audit Devices

Storage Backends

- Configures location for storage of Vault data
- Storage is defined in the main Vault configuration file with desired parameters
- All data is encrypted in transit (TLS) and at-rest using AES256
- Not all storage backends are created equal:
  - Some support high availability
  - Others have better tools for management & data protection

- There is only one storage backend per Vault cluster!

More details later in this section

Secrets Engines

- Vault components that are responsible for managing secrets for your organization
- Secrets Engines can store, generate, or encrypt data
- Many secrets engines connect to other services to generate dynamic credentials on-demand
- Many secrets engines can be enabled and used as needed
  - Even multiple secrets engines of the same type
- Secret engines are enabled and isolated at a "path"
  - All interactions are done directly with the "path" itself

More details in Objective 5

More details in Objective 1

Auth Methods

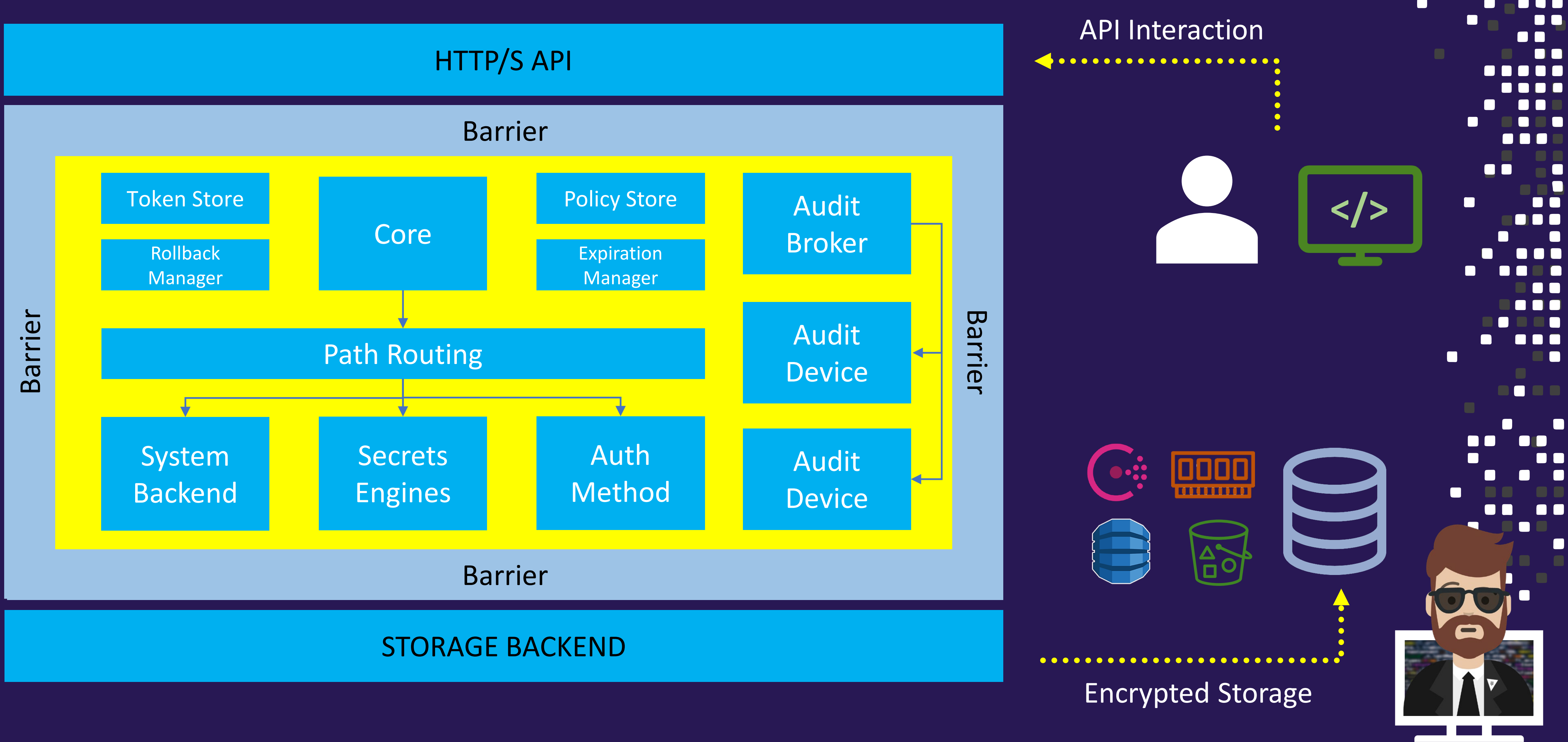
- Vault components that perform authentication and manage identities
- Responsible for assigning identity and policies to a user
- Multiple authentication methods can be enabled depending on your use case
  - Auth methods can be differentiated by human vs. system methods
- Once authenticated, Vault will issue a client token used to make all subsequent Vault requests (read/write)
  - The fundamental goal of all auth methods is to obtain a token
  - Each token has an associated policy (or policies) and a TTL
- Default authentication method for a new Vault deployment = tokens

Audit Devices

- Keeps detailed log of all requests and responses to Vault
- Audit log is formatted using JSON
- Sensitive information is hashed before logging
- Can (and should) have more than one audit device enabled
  - Vault requires at least one audit device to write the log before completing the Vault request – if enabled
  - Prioritizes safety over availability

More details later in this section

Vault Architecture



Encrypted Storage

Vault Paths

- Everything in Vault is path-based
- The path prefix tells Vault which component a request should be routed
- Secret engines, auth methods, and audit devices are “mounted” at a specified path
  - Often referred to as a 'mount'
- Paths available are dependent on the features enabled in Vault, such as Auth Methods and Secrets Engines
- System backend is a default backend in Vault which is mounted at the /sys endpoint.

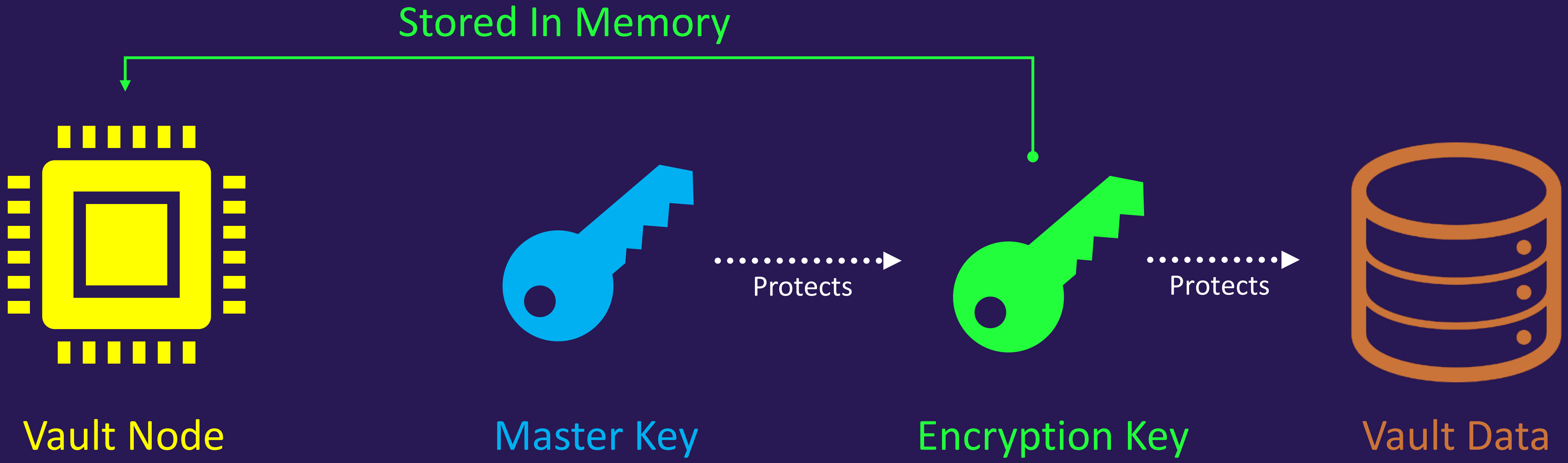
Vault Paths

- Vault components can be enabled at ANY path you'd like using the –path flag
  - Each component does have a default path you can use as well

- Vault has a few System Reserved Path which you cannot use or remove:

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

How Does Vault Protect My Data?



How Does Vault Protect My Data?

Master Key – used to decrypt the master key

- Created during Vault initialization or during a rekey operation
- Never written to storage when using traditional unseal mechanism
- Written to core/master (storage backend) when using Auto Unseal

Encryption Key – used to encrypt/decrypt data written to storage backend

- Encrypted by the Master Key
- Stored alongside the data in a keyring on the storage backend
- Can be easily rotated (manual operation)

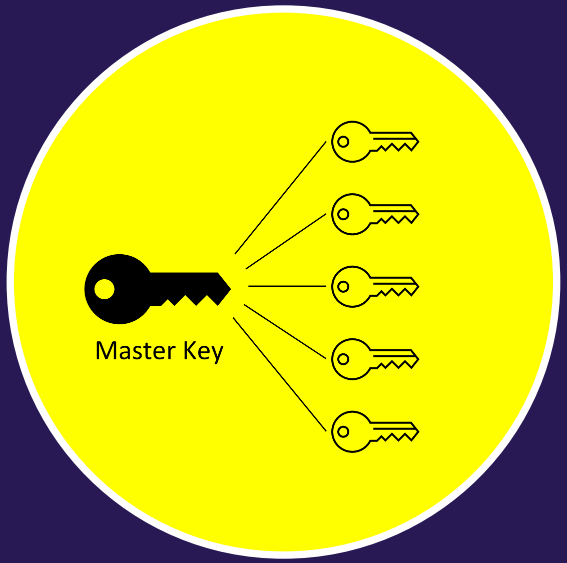
Seal and Unseal

- Vault starts in a sealed state, meaning it knows where to access the data, and how, but can’t decrypt it
- Almost no operations are possible when Vault is in a sealed state (only status check and unsealing are possible)
- Unsealing Vault means that a node can reconstruct the master key in order to decrypt the encryption key, and ultimately and read the data
- After unsealing, the encryption key is stored in memory

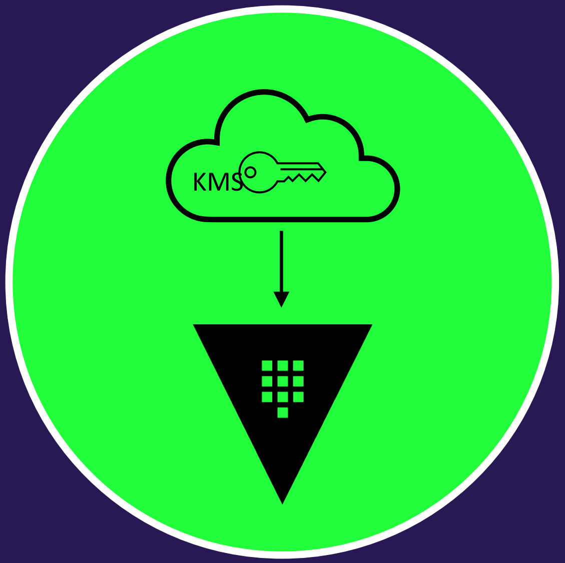
Seal and Unseal

- Sealing Vault means Vault “throws away” the encryption key and requires another unseal to perform any further operations
- Vault will start in a sealed state – you can also manually seal it via UI, CLI, or API
- When would I seal Vault?
  - Key shards are inadvertently exposed
  - Detection of a compromise or network intrusion
  - Spyware/malware on the Vault nodes

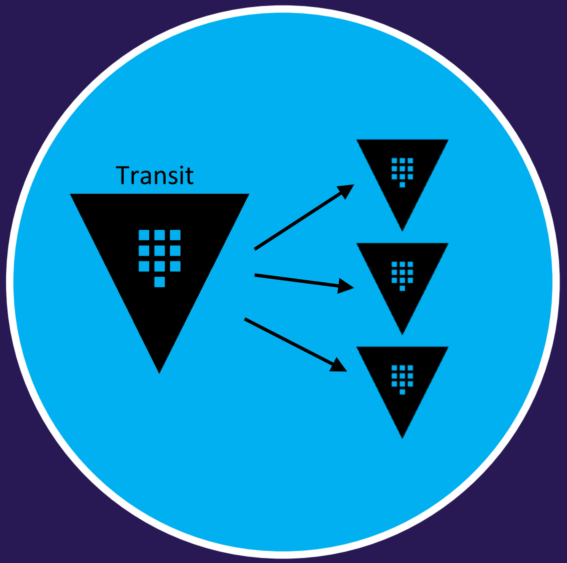
Seal and Unseal - Options



Key Sharding
(Sharmir)

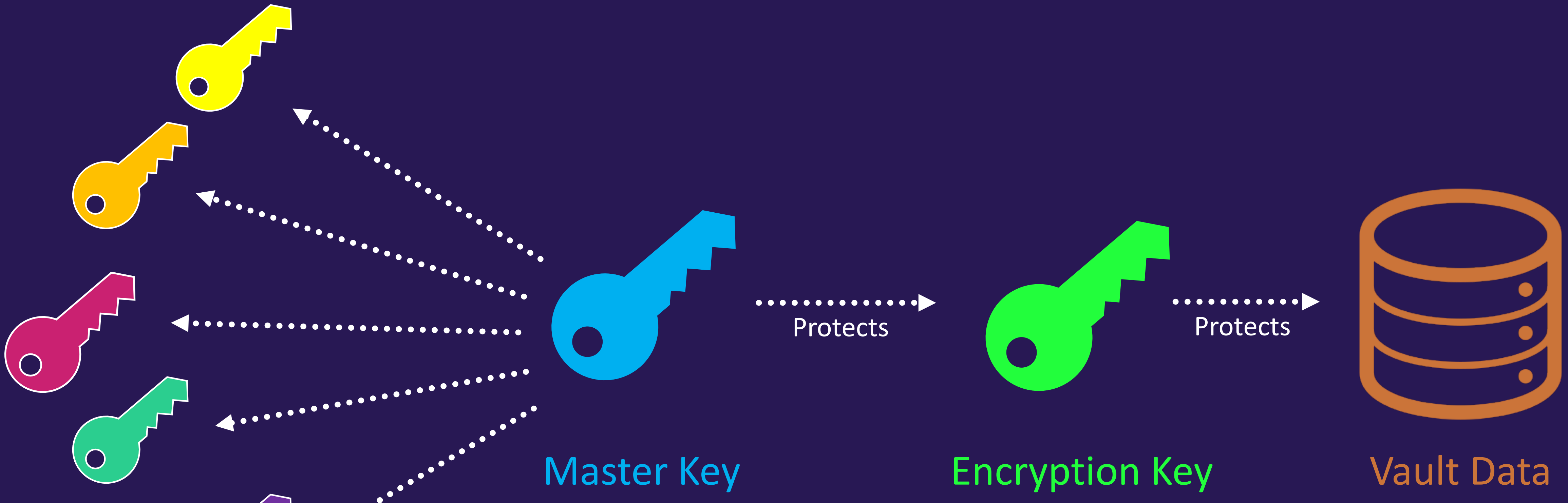


Cloud
Auto Unseal



Transit
Auto Unseal

Unsealing with Key Shards



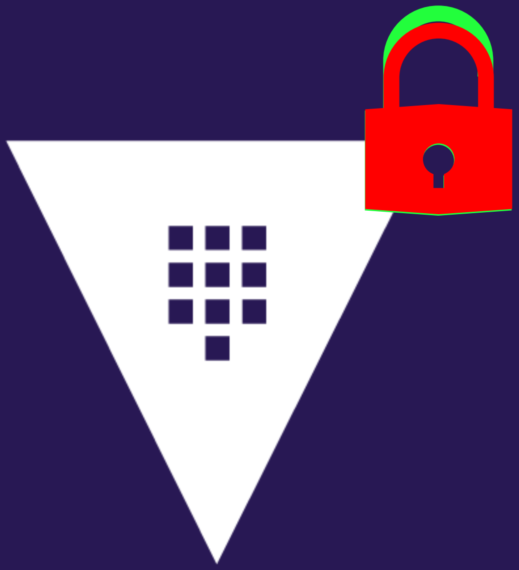
Key Shards
(Unseal Keys)

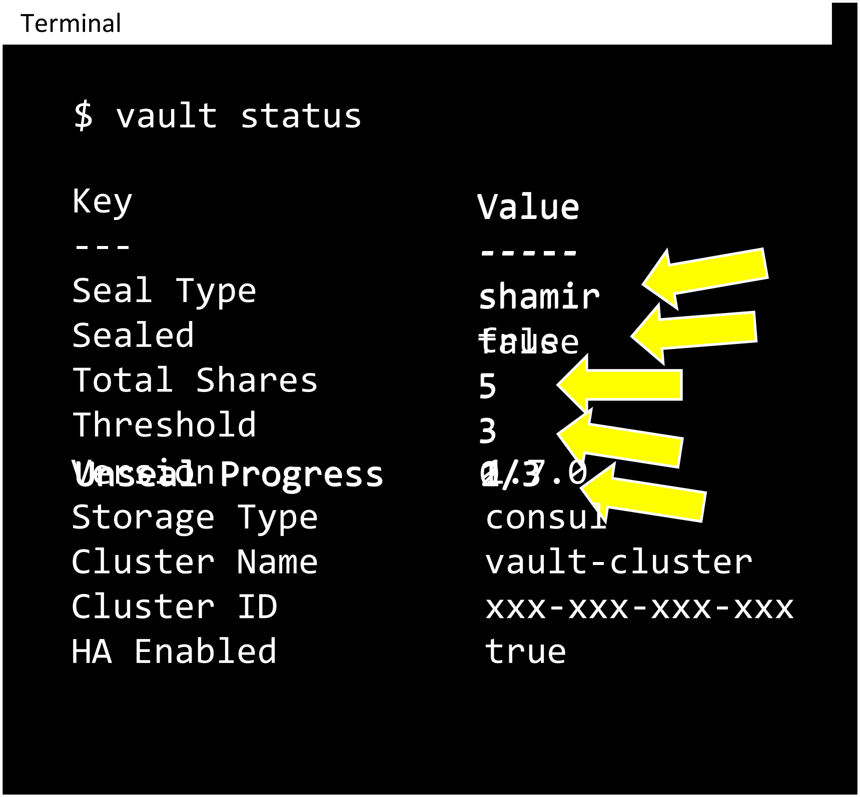
Shamir's Secret Sharing Algorithm

Unsealing with Key Shards

Trusted Employees

Unsealing with Key Shards





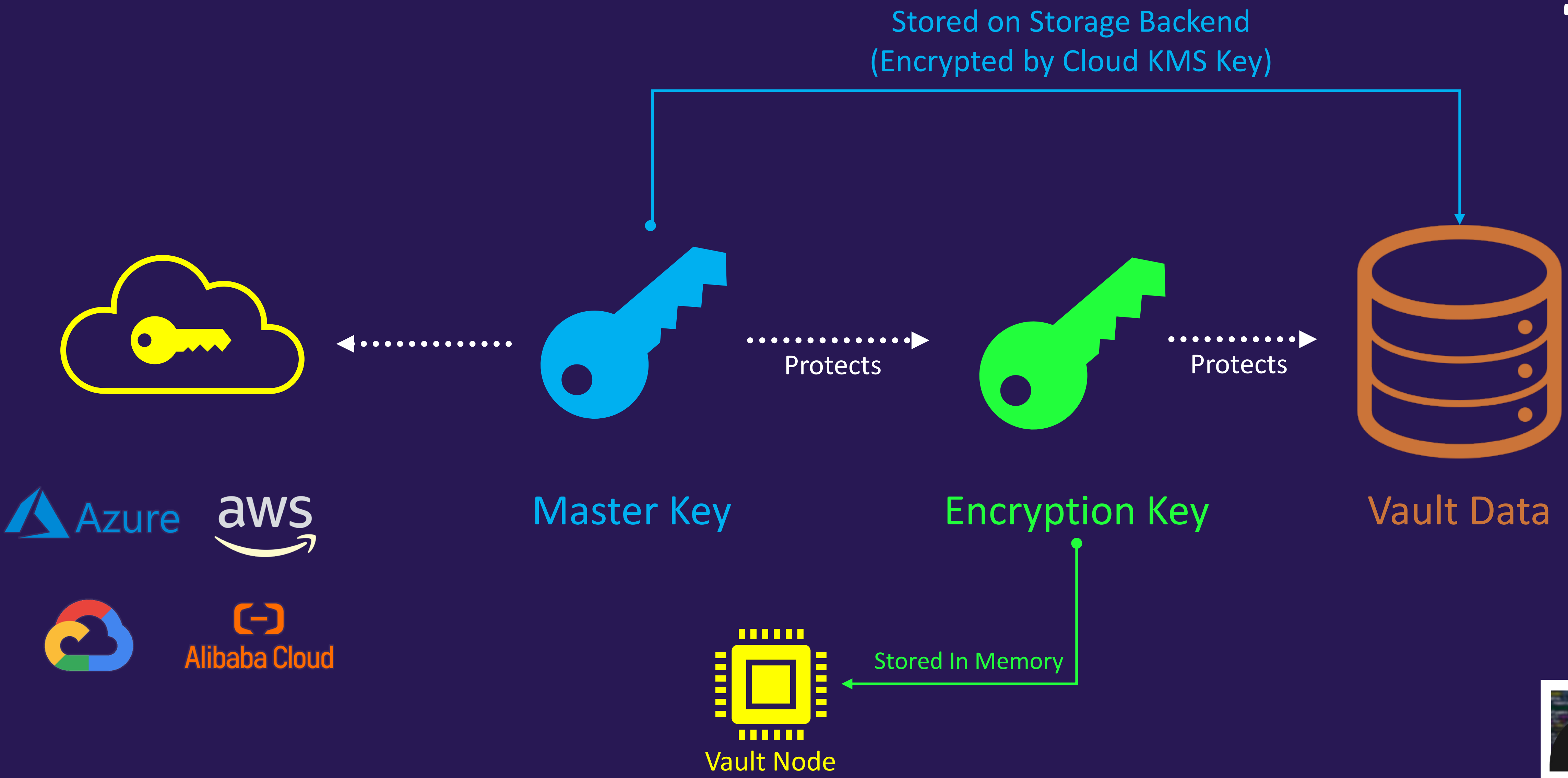
```
$ vault status

Key                 Value
---                 -----
Seal Type           shamir
Sealed              true
                    false
Total Shares        5
Threshold           3
Version             0.7.0
Unseal Progress     1/3
Storage Type        consul
Cluster Name        vault-cluster
Cluster ID          xxx-xxx-xxx-xxx
HA Enabled          true
```

Unsealing with Key Shards

- Default option for unsealing – no configuration needed
- No single person should have access to all key shards
- Ideally, each key shard should be stored by a different employee
- When initializing Vault, you can request the individual shards to be encrypted with different PGP keys
- When unsealing Vault, you will need an equal number of employees to provide their key which is equal to the threshold
- Key shards should not be stored online and should be highly protected – ideally stored encrypted

Unsealing with Auto Unseal



Unsealing with Auto Unseal

- Auto Unseal uses a cloud or on-premises HSM to decrypt the Master key
- Vault configuration file identifies the particular key to use for decryption
- Cloud Auto Unseal automatically unseals Vault upon service or node restart without additional intervention
- Available in both open source and Enterprise editions
- Formally an Enterprise-only feature until Vault 1.0

Unsealing with Auto Unseal

```
storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault/"
}
listener "tcp" {
 address = "0.0.0.0:8200"
 cluster_address = "0.0.0.0:8201"
}
seal "awskms" {
  region = "REGION"
  kms_key_id = "KMSKEY"
}
api_addr = "https://IPADDRESS:8200"
ui = true
```

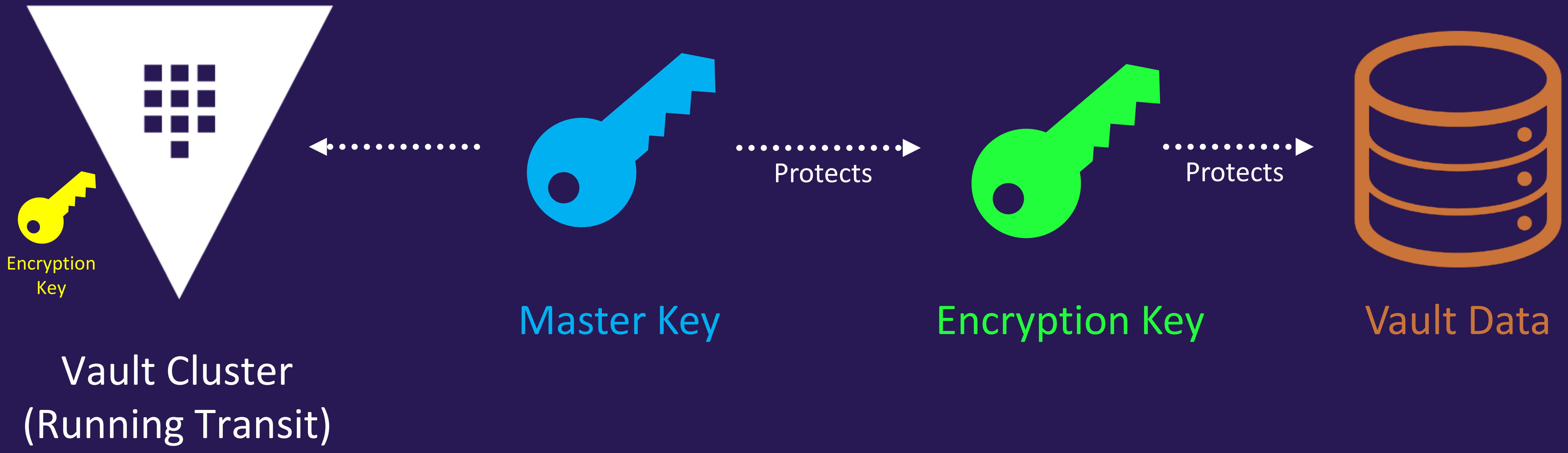
seal “awskms” – identifies the type of seal mechanism for the cluster

region = “REGION” – identifies the region where the KMS key resides

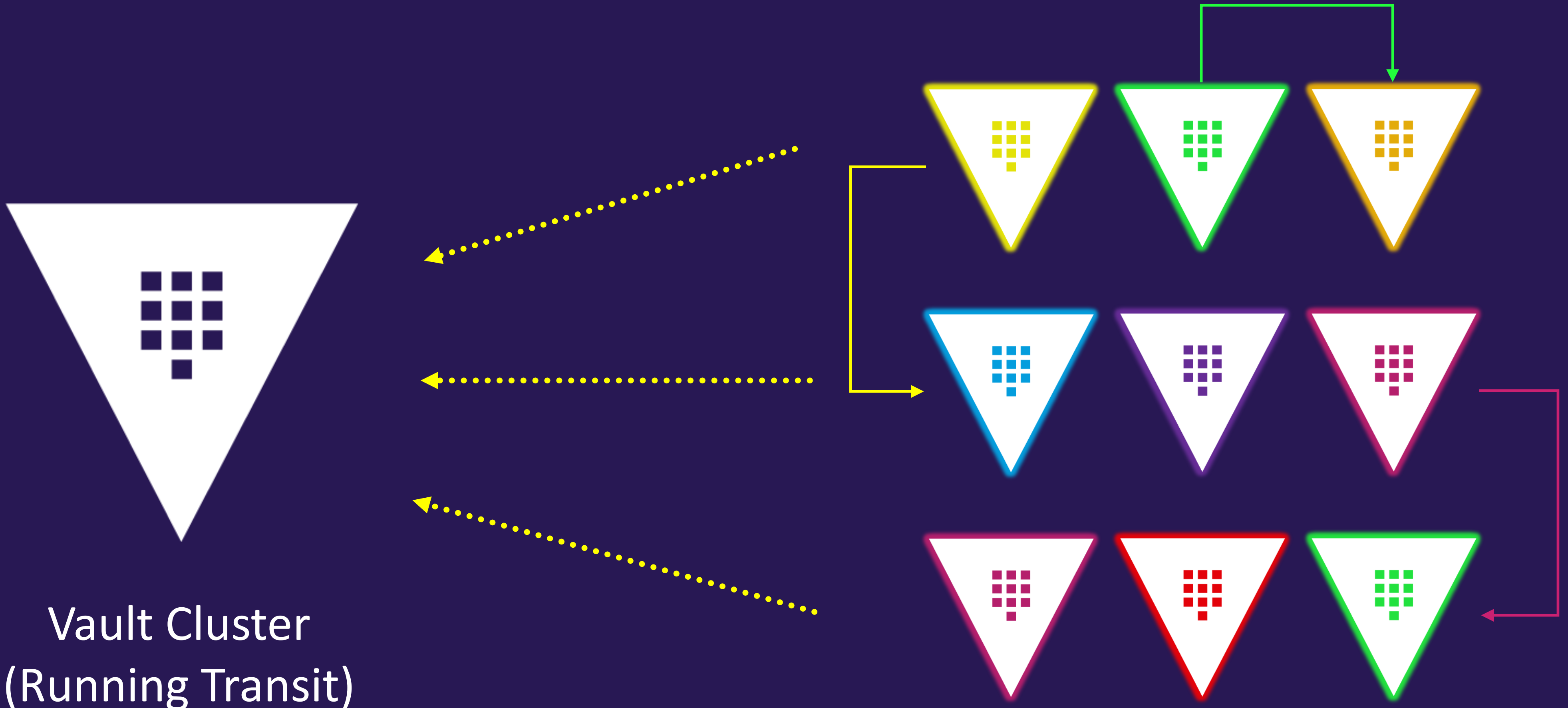
kms_key_id = “KMSKEY” – identifies the actual KMS key in AWS

Deep dive included in my
HashiCorp Vault:
The Advanced Course

Unsealing with Transit Auto Unseal



Unsealing with Transit Auto Unseal



Vault Cluster
(Running Transit)

Other Vault Clusters
In the Organization

Unsealing with Transit Auto Unseal

- Uses the Transit Secret Engine of a different Vault cluster
- The Transit Secret Engine may be configured in a Namespace
- The Transit Unseal supports key rotation
- Available in open source and Enterprise

- The core Vault cluster must be highly-available

Unsealing with Transit Auto Unseal

```
seal "transit" {
  address            = "https://vault.example.com:8200"
  token              = "s.Qf1s5zigZ4OX6akYjQXJC1jY"
  disable_renewal    = "false"

  // Key configuration
  key_name           = "transit_key_name"
  mount_path         = "transit/"
  namespace          = "ns1/"

  // TLS Configuration
  tls_ca_cert        = "/etc/vault/ca_cert.pem"
  tls_client_cert    = "/etc/vault/client_cert.pem"
  tls_client_key     = "/etc/vault/ca_cert.pem"
  tls_server_name    = "vault"
  tls_skip_verify    = "false"
}
```

address = Vault cluster running Transit

token = ACL token to use if enabled

key_name = transit key used for encryption/decryption

mount_path = mount path to the transit secret engine

namespace = namespace path to the transit secret engine

Pros and Cons of Unseal Options

Keys Shards

- ✓ Simplest form of unsealing
- ✓ Works on any platform
- ✓ Configuration options make it flexible

Auto Unseal

- ✓ Automatic unsealing of Vault
- ✓ Set and forget
- ✓ Integration benefits for running on same platform

Transit Unseal

- ✓ Automatic unsealing of Vault
- ✓ Set and forget
- ✓ Platform agnostic
- ✓ Useful when running many Vault clusters across clouds/data centers

Pros and Cons of Unseal Options

Keys Shards

X Introduces risk for storing keys

X Requires manual intervention for unsealing

X Keys can be inadvertently shared and require rotation

Auto Unseal

X Regional requirements for cloud HSMs

X Cloud/vendor lock-in

Transit Unseal

X Requires a centralized Vault cluster

X Centralized Vault cluster needs the highest level of uptime

Vault Initialization

- Initializing Vault prepares the backend storage to receive data
- Only need to initialize a Vault cluster one time via a single node
- Vault initialization is when Vault creates the master key and key shares
- Options to define thresholds, key shares, recovery keys, and encryption
- Vault initialization is also where the initial root token is generated and returned to the user

- Vault can be initialized via CLI, API, or UI

```
$ vault operator init <options>
```

Configuration File

- Vault servers are configured using a file
    - Written in HCL or JSON
- The configuration file includes different stanzas and parameters to define a variety of configuration options
- Configuration file is specified when starting Vault using the – config flag

```
$ vault server -config <location>
```

- Usually stored somewhere in /etc (doesn’t have to be)
    - I store mine at /etc/vault.d/vault.hcl

```
$ vault server -config /etc/vault.d/vault.hcl
```

Configuration File

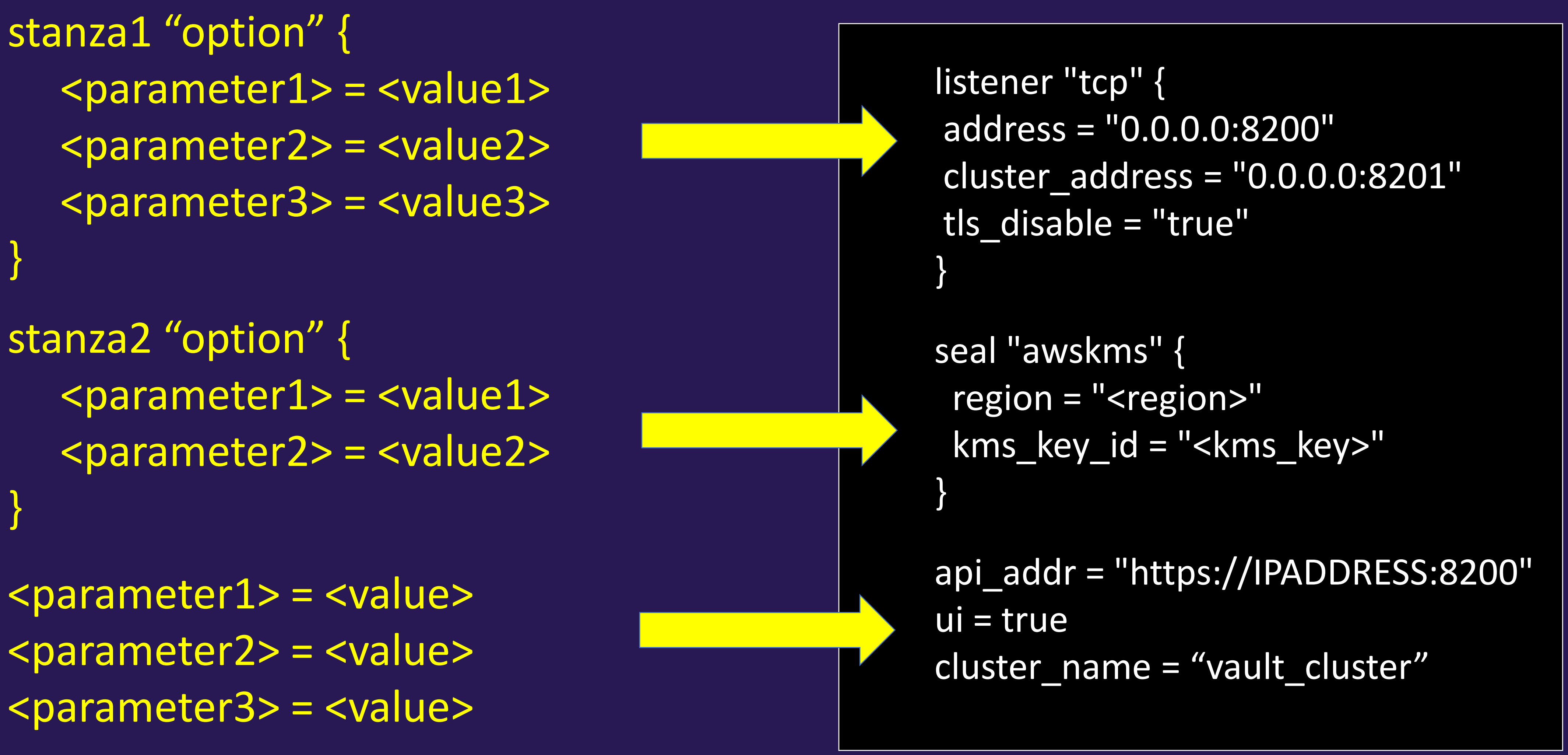
What’s Configured in the File?

- Storage Backend
- Listener(s) and Port
- TLS certificate
- Seal Type
- Cluster Name
- Log Level
- UI
- Cluster IP and Port

What’s *Not*?

- Secrets Engines
- Authentication Methods
- Audit Devices
- Policies
- Entities & Groups

Configuration File



```
stanza1 “option” {
    <parameter1> = <value1>
    <parameter2> = <value2>
    <parameter3> = <value3>
}

stanza2 “option” {
    <parameter1> = <value1>
    <parameter2> = <value2>
}

<parameter1> = <value>
<parameter2> = <value>
<parameter3> = <value>
```

```
listener "tcp" {
 address = "0.0.0.0:8200"
 cluster_address = "0.0.0.0:8201"
 tls_disable = "true"
}

seal "awskms" {
  region = "<region>"
  kms_key_id = "<kms_key>"
}

api_addr = "https://IPADDRESS:8200"
ui = true
cluster_name = “vault_cluster”
```

Configuration File

Available Stanzas:

– seal – seal type

– listener – addresses/ports for Vault

– storage – storage backend

– telemetry – where to publish metrics to upstream systems

Example of Parameters:

– cluster_name – identifier for the cluster – Vault will auto-generate name if omitted

– log_level – specifies the log level to use – Trace, Debug, Error, Warn, Info

– ui – enables the built-in web UI

– api_addr – address to advertise to other Vault servers for client redirection

– cluster_addr – address to advertise to other Vault servers for request forwarding

Configuration File - Example

```
storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault/"
  token   = "1a2b3c4d-1234-abdc-1234-1a2b3c4d5e6a"
}
listener "tcp" {
 address = "0.0.0.0:8200"
 cluster_address = "0.0.0.0:8201"
 tls_disable = 0
 tls_cert_file = "/etc/vault.d/client.pem"
 tls_key_file = "/etc/vault.d/cert.key"
 tls_disable_client_certs = "true"
}
seal "awskms" {
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101",
  endpoint = "example.kms.us-east-1.vpce.amazonaws.com"
}
api_addr = "https://vault-us-east-1.example.com:8200"
cluster_addr = " https://node-a-us-east-1.example.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```

Storage Stanza

Listener Stanza

Seal Stanza

Additional Parameters

https://github.com/btkrausen/hashicorp/blob/master/vault/config_files/vault.hcl

Storage Backend

- Configures location for storage of Vault data

- Open-source users can choose a storage backend based on their preferences (for the most part)

- Enterprise Vault Clusters should use HashiCorp Consul or Integrated Storage
    - Everything else is “community supported” and can be used for open-source

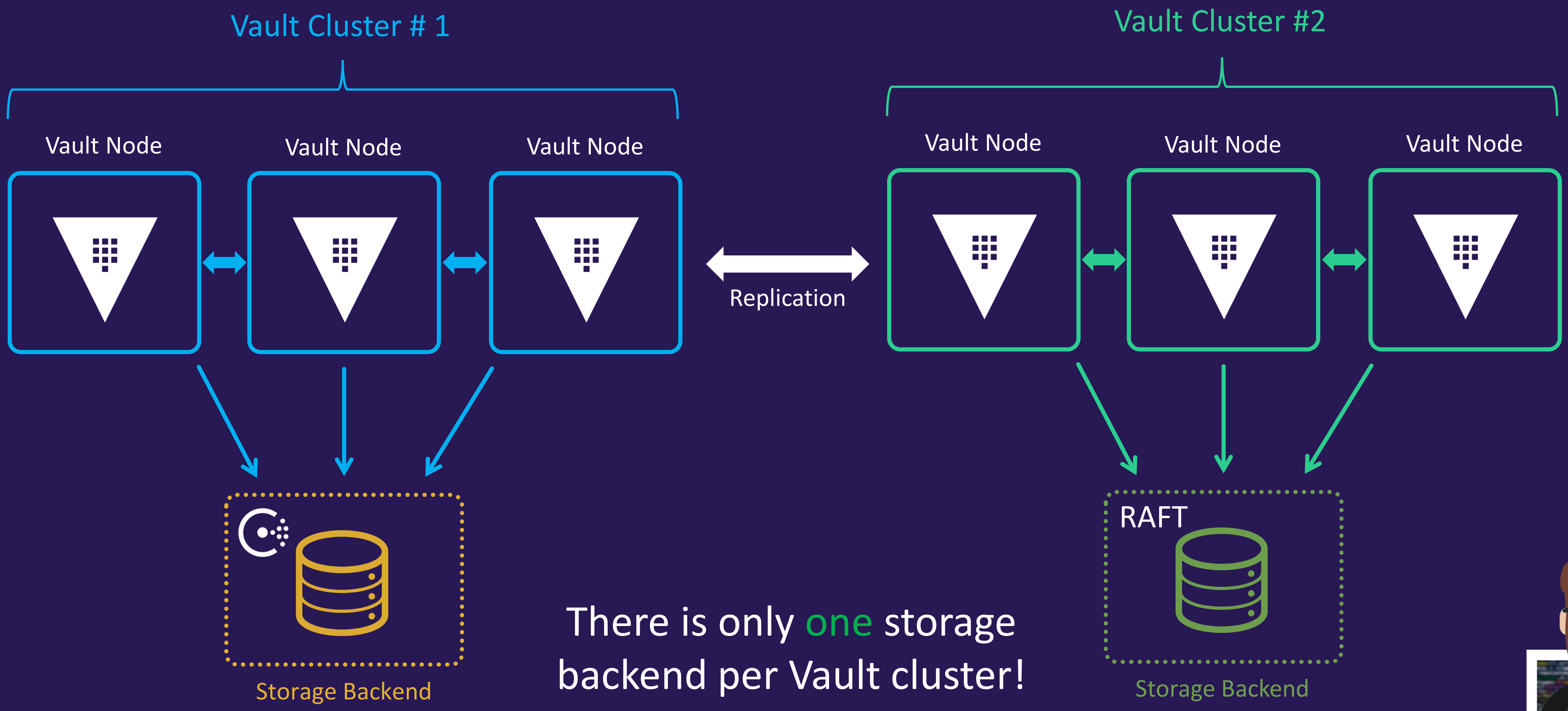
Storage Backend

- Aerospike
- Azure
- Cassandra
- CockroachDB
- Consul
- CouchDB
- Etcd
- Filesystem
- FoundationDB
- Google Cloud Spanner
- Google Cloud Storage

- In-Memory
- Manta
- MSSQL
- MySQL
- OCI Object Storage
- PostgreSQL
- Integrated Storage (Raft)
- Amazon S3
- Swift
- Zookeeper

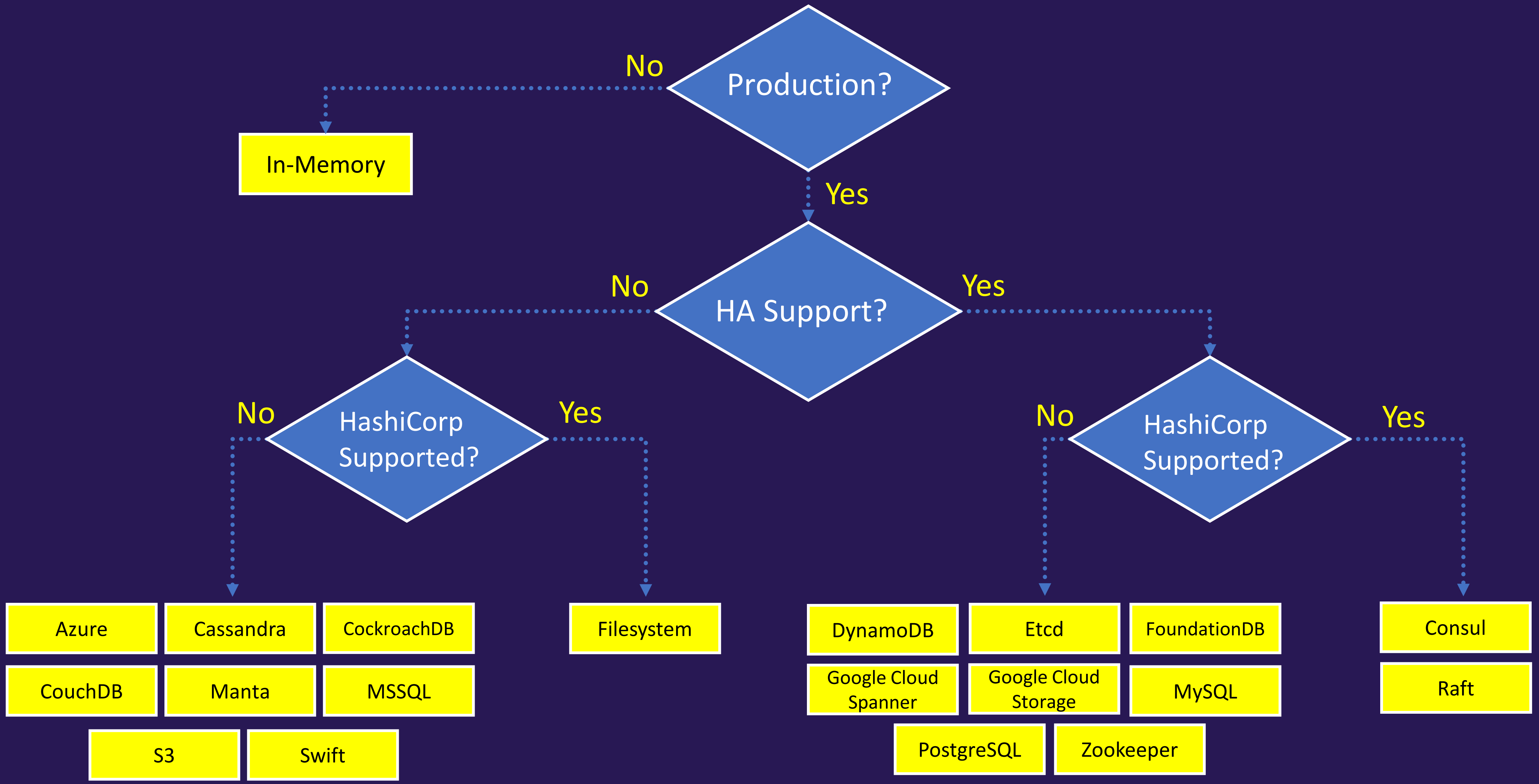
*Updated based on Vault 1.7

Storage Backend



There is only one storage backend per Vault cluster!

Choosing a Storage Backend



Production?

No

In-Memory

Yes

HA Support?

No

HashiCorp Supported?

No

Azure

Cassandra

CockroachDB

CouchDB

Manta

MSSQL

S3

Swift

Yes

Filesystem

Yes

HashiCorp Supported?

No

DynamoDB

Etcd

FoundationDB

Google Cloud Spanner

Google Cloud Storage

MySQL

PostgreSQL

Zookeeper

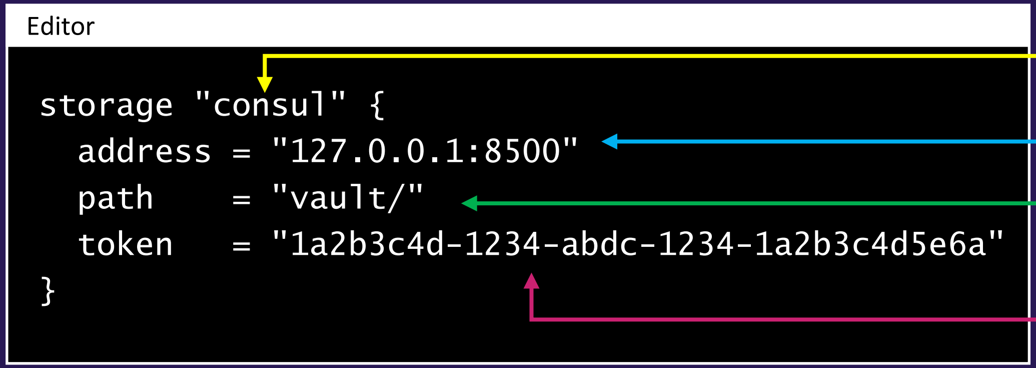
Yes

Consul

Raft

Credit: @GreenReedTech

Storage Backend - Configuration



```
storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault/"
  token   = "1a2b3c4d-1234-abdc-1234-1a2b3c4d5e6a"
}
```

Type of Storage Backend

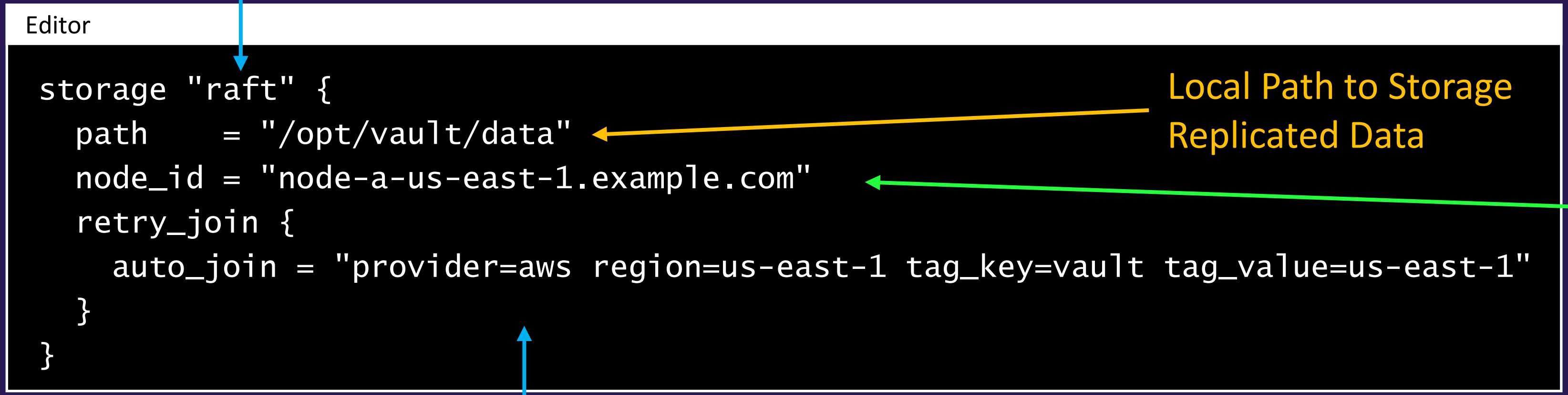
IP/Port of Consul Agent

Path in Consul K/V to store Vault Data

Consul ACL Token

Integrated Storage

Type of Storage Backend



```
storage "raft" {
  path    = "/opt/vault/data"
  node_id = "node-a-us-east-1.example.com"
  retry_join {
    auto_join = "provider=aws region=us-east-1 tag_key=vault tag_value=us-east-1"
  }
}
```

Local Path to Storage Replicated Data

Name/ID of Node

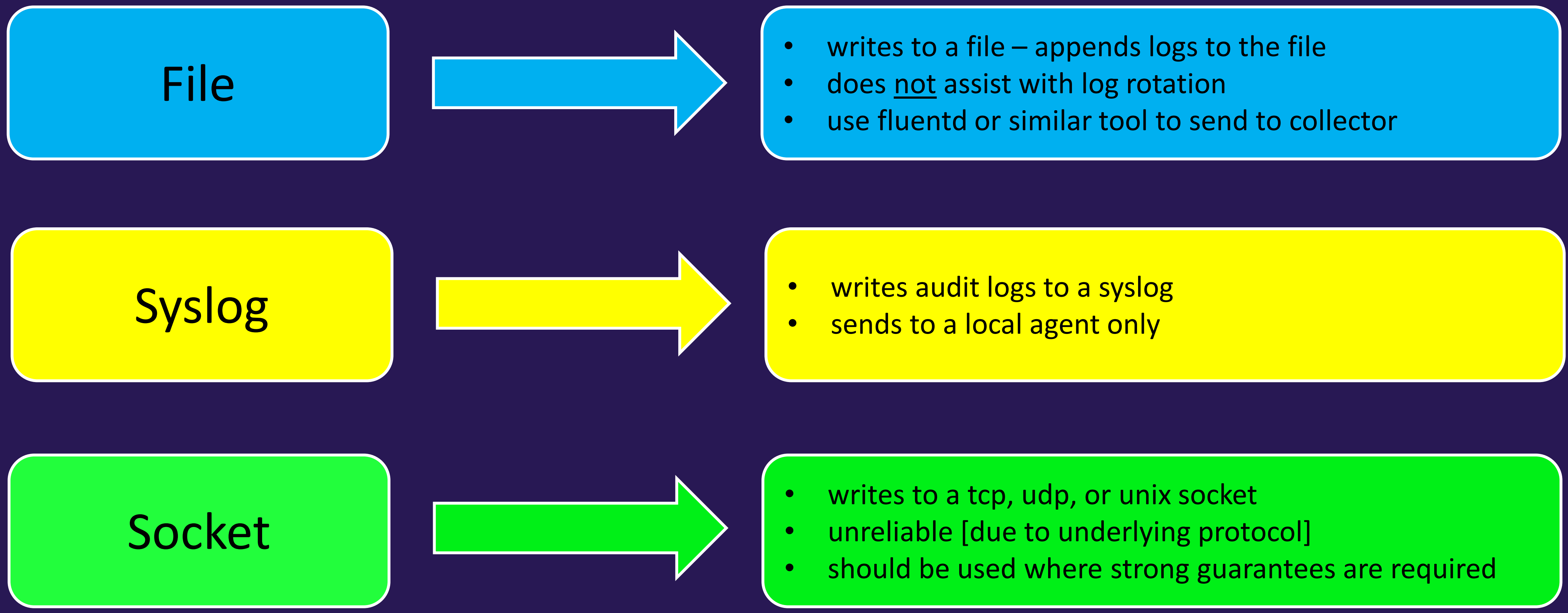
Cluster Join options

Audit Device

- Keep a detailed log of all authenticated requests and responses to Vault
- Audit log is formatted using JSON
- Sensitive information is hashed with a salt using HMAC-SHA256 to ensure secrets and tokens aren’t ever in plain text
- Log files should be protected as a user with permission can still check the value of those secrets via the /sts/audit-hash API and compare to the log file

```
$ vault audit enable file file_path=/var/log/vault_audit_log.log
```

Audit Device



File

- writes to a file – appends logs to the file
- does not assist with log rotation
- use fluentd or similar tool to send to collector

Syslog

- writes audit logs to a syslog
- sends to a local agent only

Socket

- writes to a tcp, udp, or unix socket
- unreliable [due to underlying protocol]
- should be used where strong guarantees are required

Audit Device

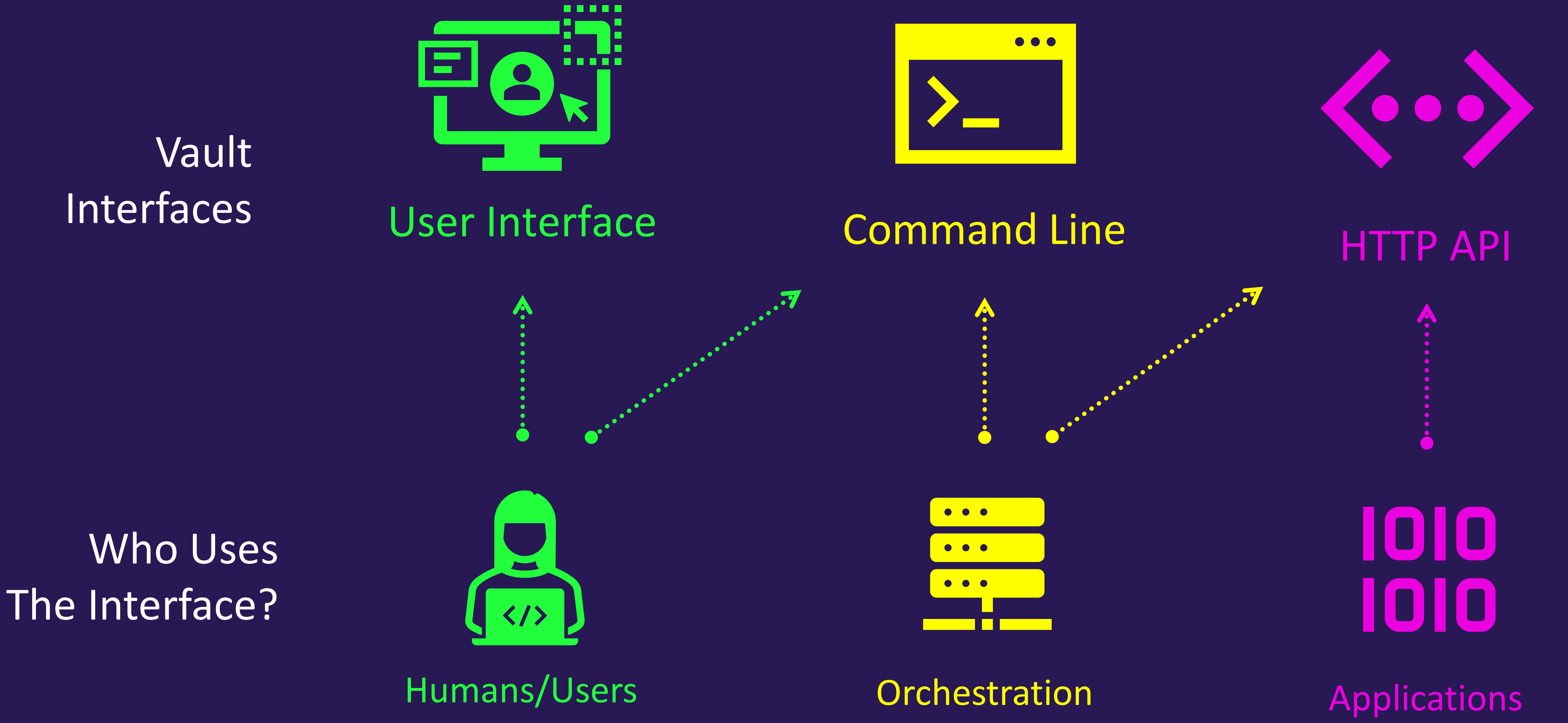
- Can and should have more than one audit device enabled
- If there are any audit devices enabled, Vault requires that it can write to the log before completing the client request.
  - Prioritizes safety over availability
- If Vault cannot write to a persistent log, it will stop responding to client requests – which means Vault is down!

⚠ Vault requires at least one audit device to write the log before completing the Vault request – if enabled

Vault Interfaces

- Three interfaces to interact with Vault: UI, CLI, and HTTP API
- Not all Vault features are available via UI and CLI but all features can be accessed using the HTTP API
- Calls from the CLI and UI invoke the HTTP API. CLI is just a thin wrapper on the HTTP API
- UI must be enabled via configuration file
- Authentication required to access any of the interfaces

Vault Interfaces



Want to Learn More?

Command Line

Objective 6

Utilize Vault CLI

User Interface

Objective 7

Utilize Vault UI

HTTP API

Objective 8

Be Aware of the Vault API

Installing Vault

- Vault is platform agnostic….meaning it can be run on many different underlying platforms
  - Kubernetes
  - Cloud-based Machines (AWS Instances, Azure Virtual Machines)
  - VMware Virtual Machines
  - Physical Servers
  - A Laptop

Installing Vault

- Vault is also available for many operating systems…
  - ✓ macOS
  - ✓ Windows
  - ✓ Linux
  - ✓ FreeBSD
  - ✓ NetBSD
  - ✓ OpenBSD
  - ✓ Solaris

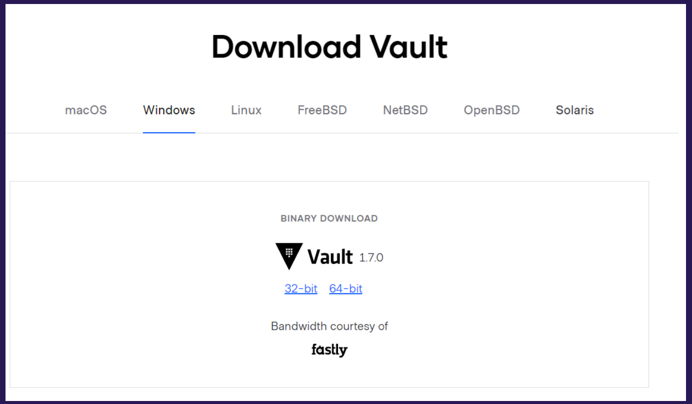
Installing Vault

Order of Operations

1. Install Vault
2. Create Configuration File
3. Initialize Vault
4. Unseal Vault

Installing Vault

- So where do I download Vault?
  - vaultproject.io
  - releases.hashicorp.com/vault



Download Vault

BINARY DOWNLOAD

Vault 1.7.0

Bandwidth courtesy of

fastly

- You can also download/install Vault using your preferred package manager as well (apt, yum, even homebrew(community supported))

Terminal

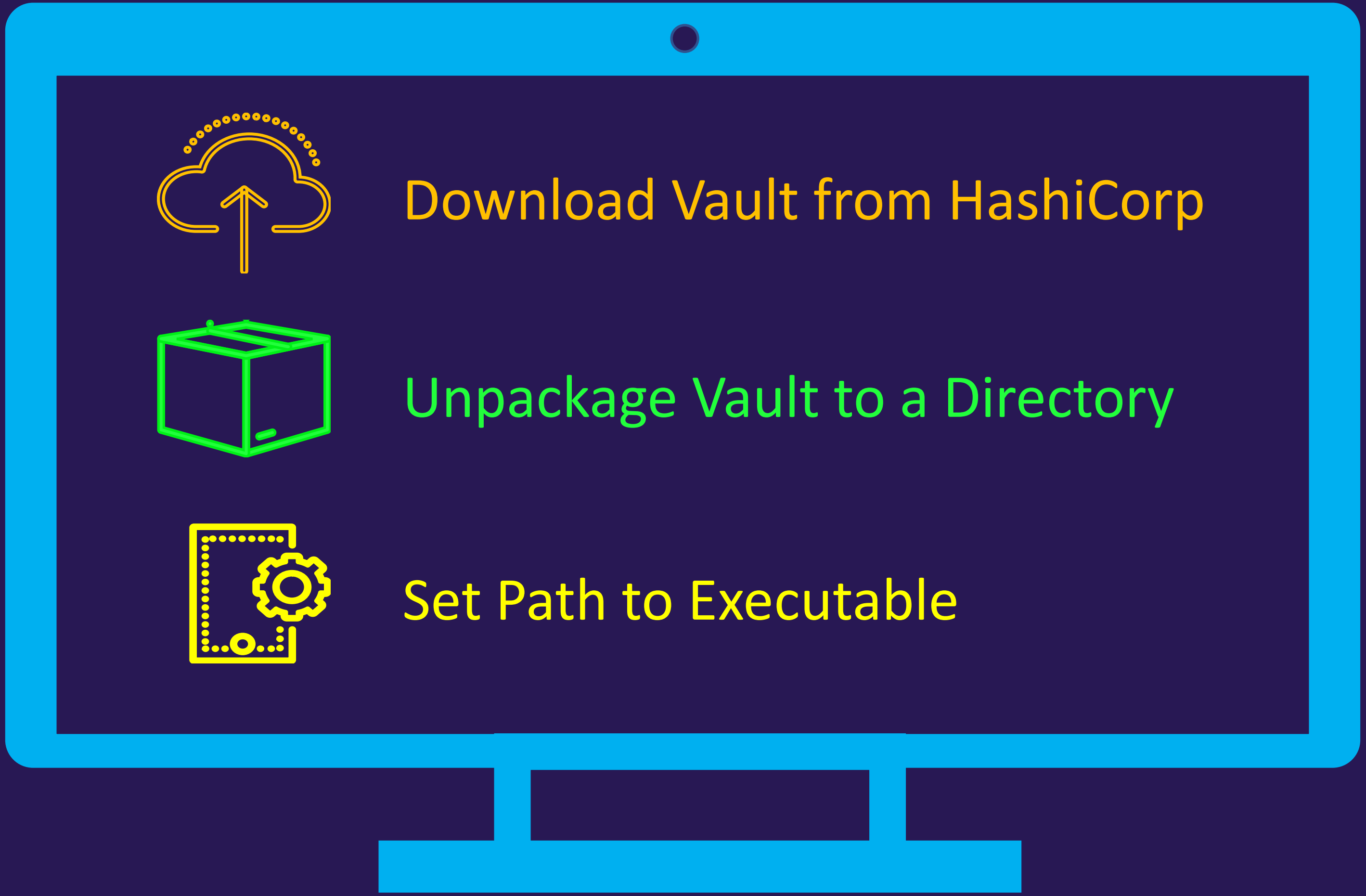
```
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
$ sudo apt-get update && sudo apt-get install vault
```

- Use the Vault Helm Chart to install/configure Vault on Kubernetes

Terminal

```
$ helm install vault hashicorp/vault
```

Installing Vault



Download Vault from HashiCorp

Unpackage Vault to a Directory

Set Path to Executable

Running Vault Dev Server

Quickly run Vault without configuration

Automatically initialized and unsealed

Enables the UI – available at localhost

Provides an Unseal Key

Automatically logs in as root

Non-Persistent – Runs in memory

Insecure – doesn’t use TLS

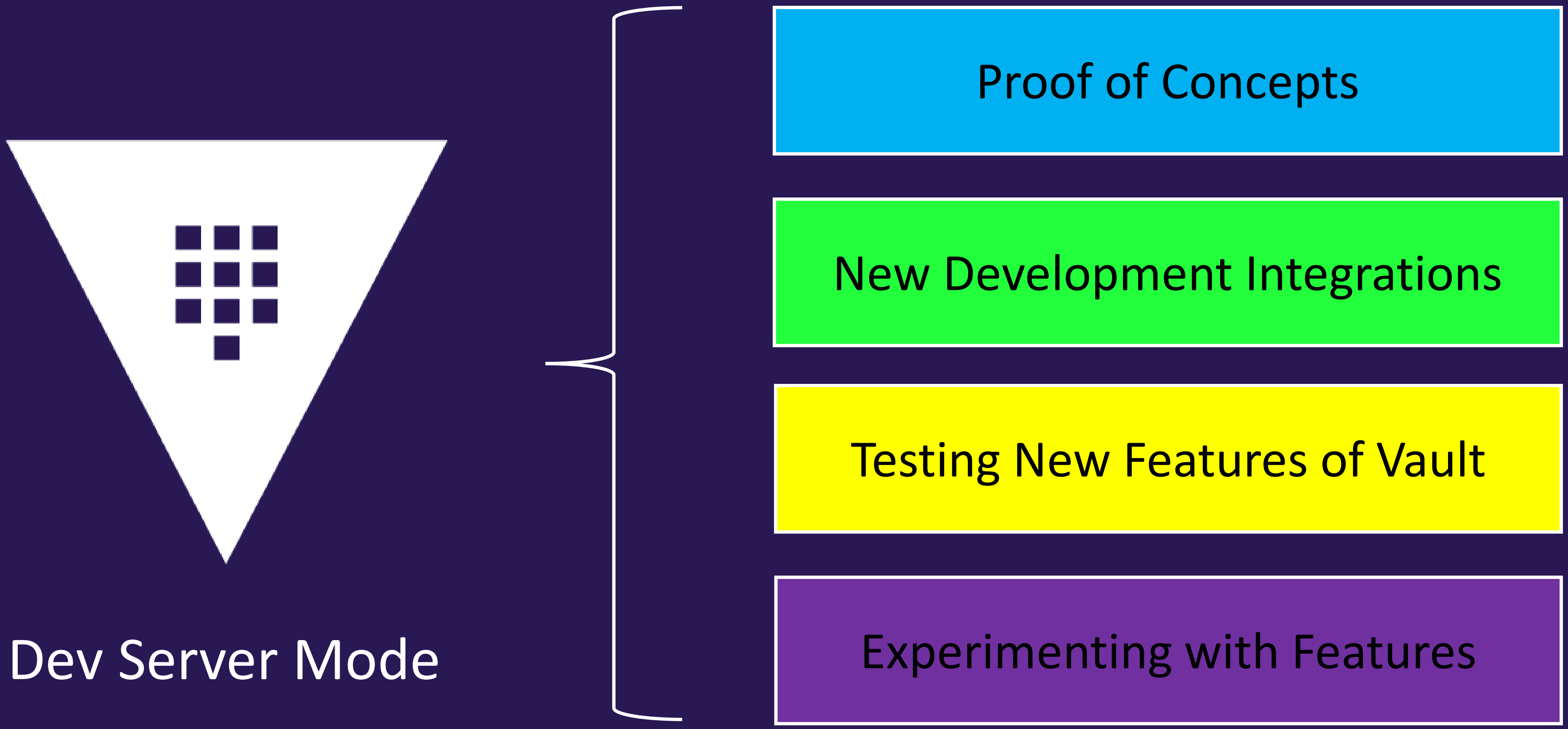
Sets the listener to 127.0.0.1:8200

Mounts a K/V v2 Secret Engine

Provides a root token

NEVER USE DEV SERVER MODE IN PRODUCTION!

Where Would I Use Dev Server?



Dev Server Mode

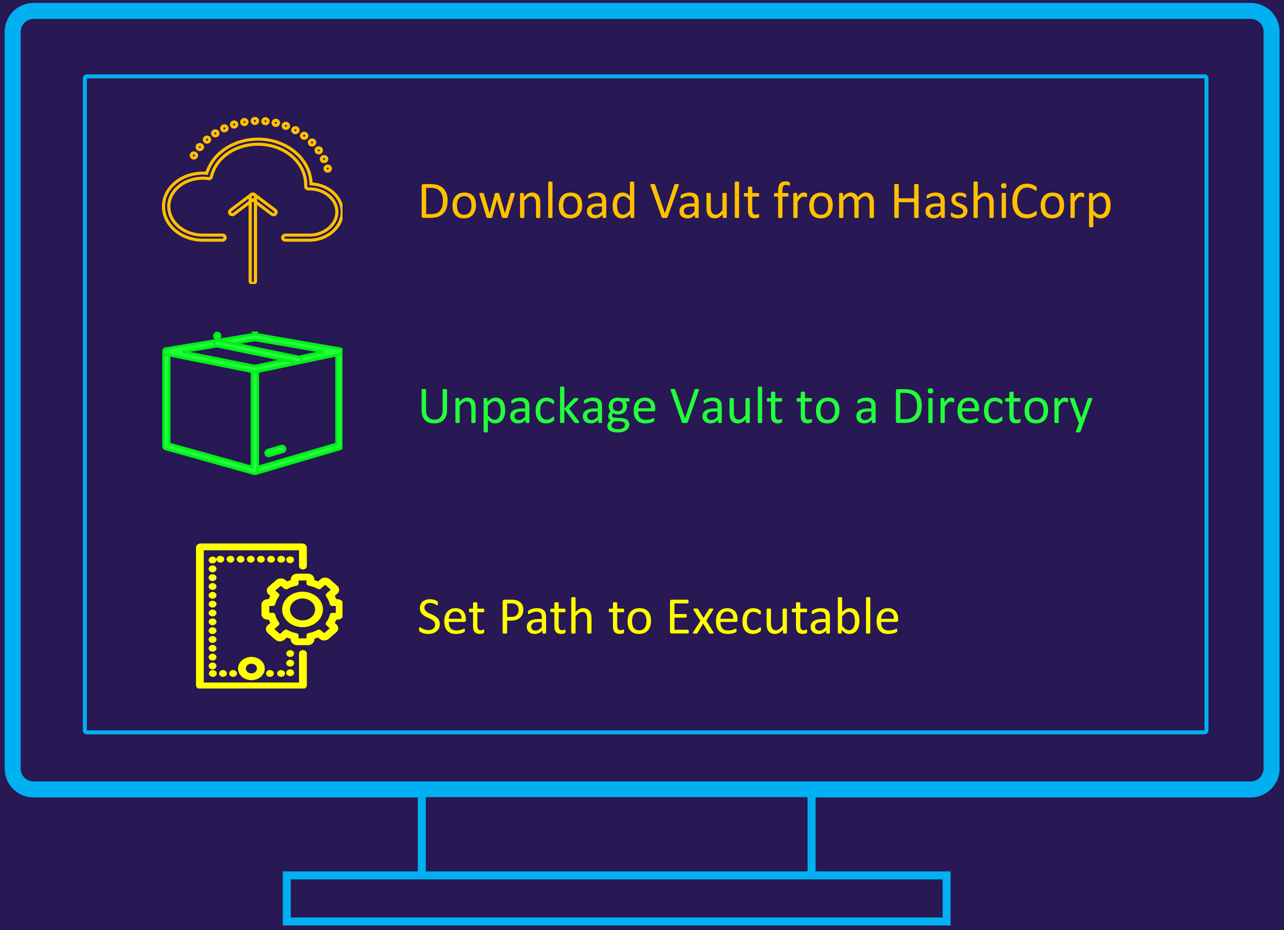
Proof of Concepts

New Development Integrations

Testing New Features of Vault

Experimenting with Features

Installing Vault

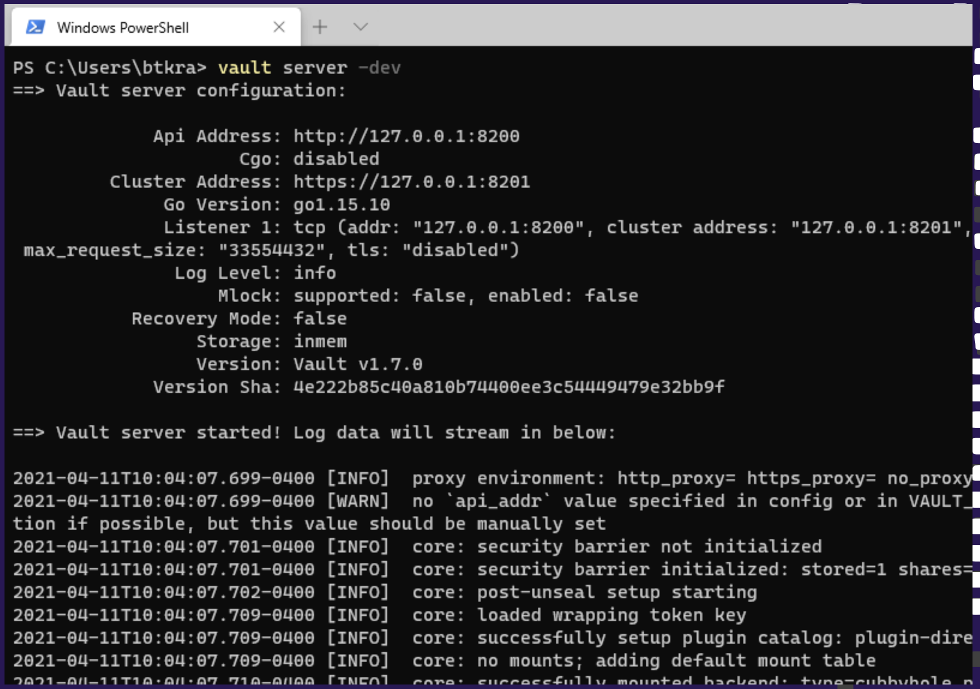


Download Vault from HashiCorp

Unpackage Vault to a Directory

Set Path to Executable

```
$ vault server -dev
```

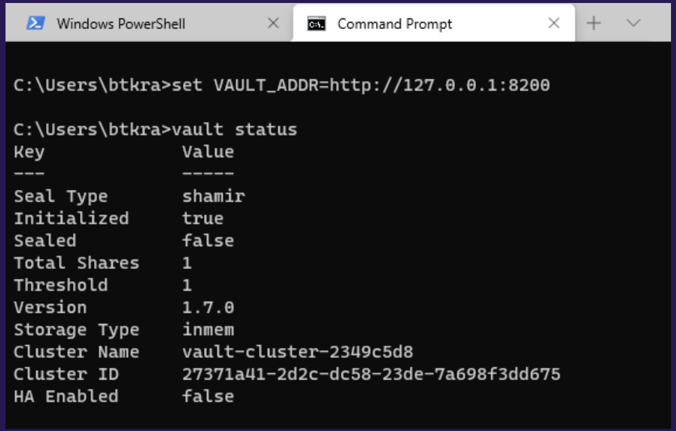


```
PS C:\Users\btkra> vault server -dev
==> Vault server configuration:

             Api Address: http://127.0.0.1:8200
                     Cgo: disabled
         Cluster Address: https://127.0.0.1:8201
              Go Version: go1.15.10
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201",
 max_request_size: "33554432", tls: "disabled")
               Log Level: info
                   Mlock: supported: false, enabled: false
           Recovery Mode: false
                 Storage: inmem
                 Version: Vault v1.7.0
             Version Sha: 4e222b85c40a810b74400ee3c54449479e32bb9f

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

2021-04-11T10:04:07.699-0400 [INFO]  proxy environment: http_proxy= https_proxy= no_proxy
2021-04-11T10:04:07.699-0400 [WARN]  no `api_addr` value specified in config or in VAULT_
tion if possible, but this value should be manually set
2021-04-11T10:04:07.701-0400 [INFO]  core: security barrier not initialized
2021-04-11T10:04:07.701-0400 [INFO]  core: security barrier initialized: stored=1 shares=
2021-04-11T10:04:07.702-0400 [INFO]  core: post-unseal setup starting
2021-04-11T10:04:07.709-0400 [INFO]  core: loaded wrapping token key
2021-04-11T10:04:07.709-0400 [INFO]  core: successfully setup plugin catalog: plugin-dire
2021-04-11T10:04:07.709-0400 [INFO]  core: no mounts; adding default mount table
2021-04-11T10:04:07.710-0400 [INFO]  core: successfully mounted backend: type=cubbyhole p
```



```
C:\Users\btkra>set VAULT_ADDR=http://127.0.0.1:8200

C:\Users\btkra>vault status
Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.7.0
Storage Type    inmem
Cluster Name    vault-cluster-2349c5d8
Cluster ID      27371a41-2d2c-dc58-23de-7a698f3dd675
HA Enabled      false
```

Running Vault Server in Production

- Deploy one or more persistent nodes via configuration file
- Use a storage backend that meets the requirements
- Multiple Vault nodes will be configured as a cluster
- Deploy close to your applications
- Most likely, you’ll automate the provisioning of Vault

Running Vault Server in Production

- To start Vault, run the `vault server -config=<file>` command
- In a production environment, you'll have a service manager executing and managing the Vault service (systemctl, Windows Service Manager, etc.)

- For Linux, you also need a systemd file to manage the service for Vault (and Consul if you're running Consul)

Running Vault Server in Production

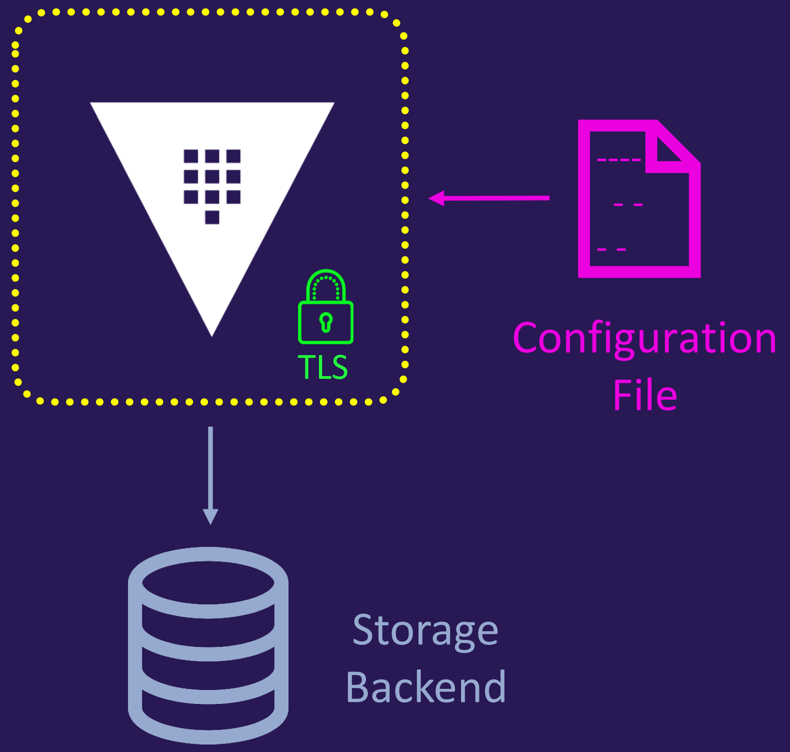
- Systemd for a Vault service:
  - https://github.com/btkrausen/hashicorp/blob/master/vault/config_files/vault.service

- Systemd file for a Consul Server:
  - https://github.com/btkrausen/hashicorp/blob/master/consul/consul.service

- Systemd for a Consul client (that would run on the Vault node):
  - https://github.com/btkrausen/hashicorp/blob/master/vault/config_files/consul-client.json

Running Vault Server in Production

Single Node

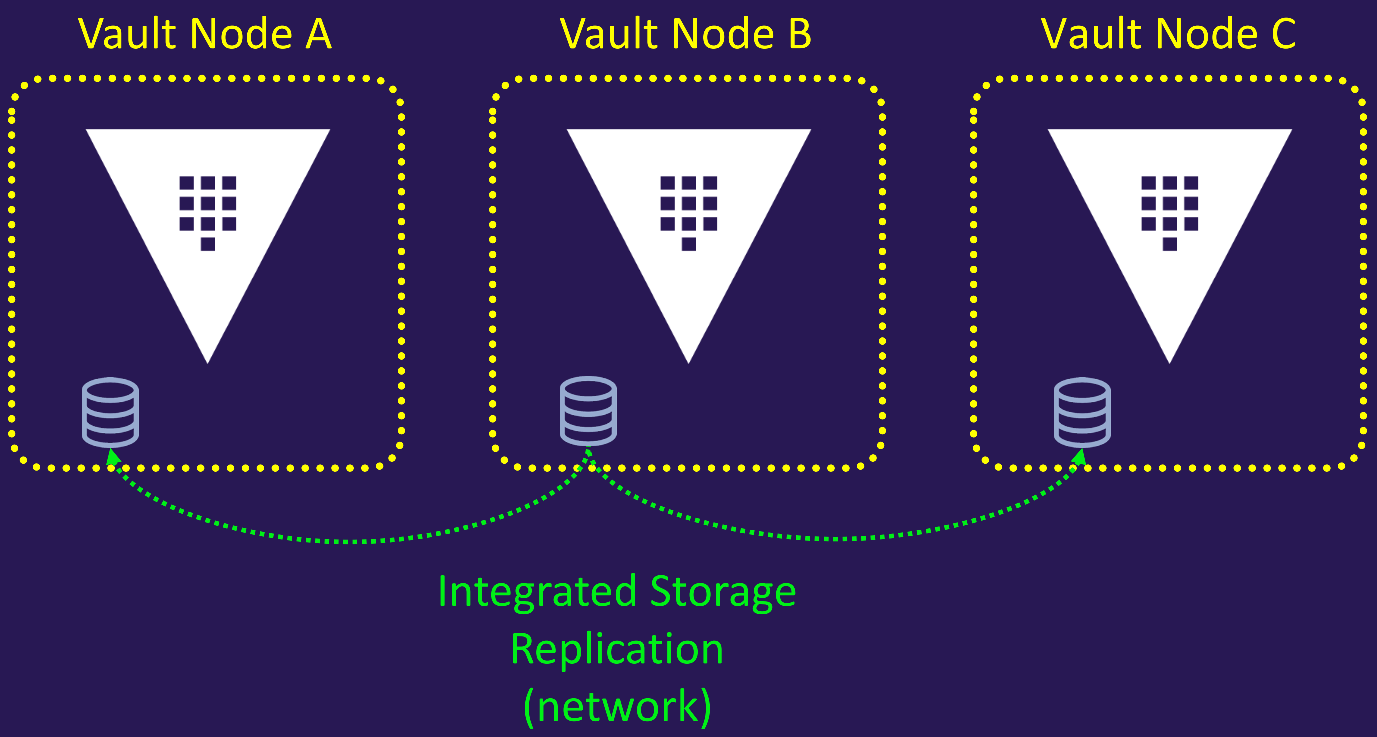


Not a Recommended Architecture

- No Redundancy
- No Scalability

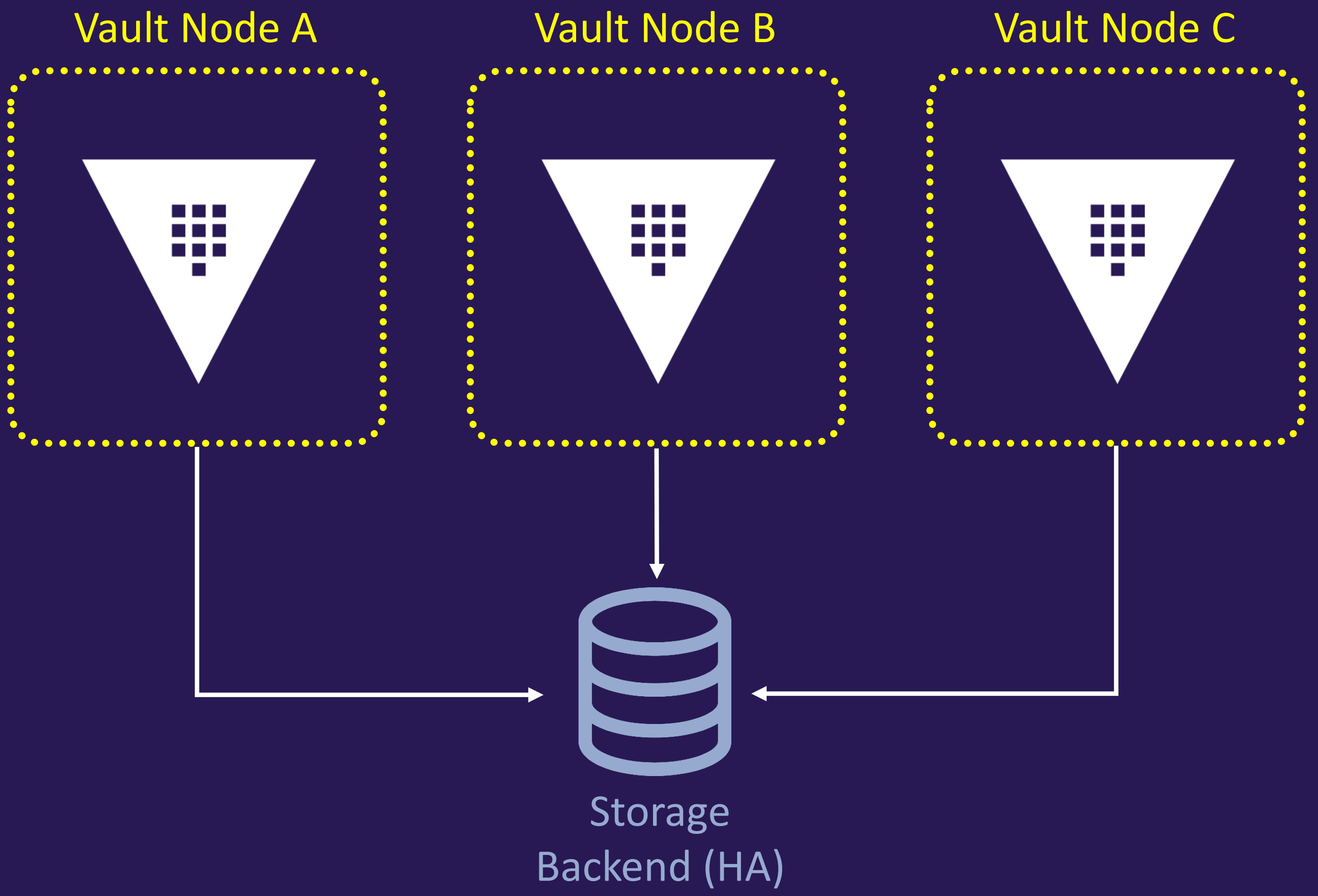
Running Vault Server in Production

Multi-Node Vault Cluster (with Integrated Storage)



Running Vault Server in Production

Multi-Node Vault Cluster (with external storage backend)



Running Vault Server in Production

Step-by-Step Manual Install

1. Download Vault from HashiCorp
2. Unpackage Vault to a Directory
3. Set Path to Executable
4. Add Configuration File & Customize
5. Create Systemd Service File
6. Download Consul from HashiCorp
7. Configure and Join Consul Cluster
8. Launch Vault Service

Deploying the Consul Storage Backend

Provides Durable K/V Storage For Vault

Can Independently Scale Backend

Easy To Automate

Built-in Integration Between Consul/Vault

Supports High Availability

Distributed System

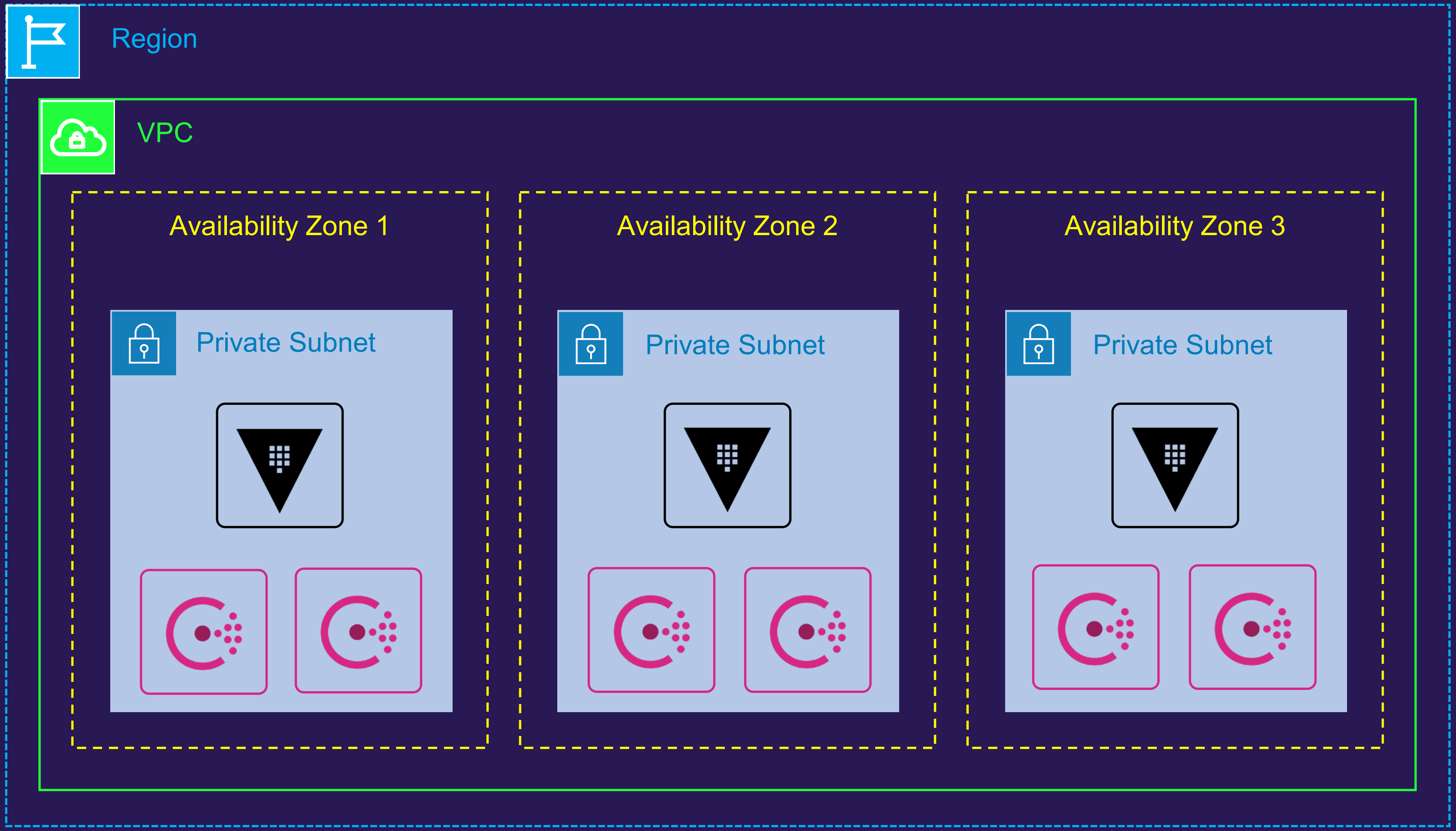
Built-in Snapshots For Data Retention

HashiCorp Supported

Deploying the Consul Storage Backend

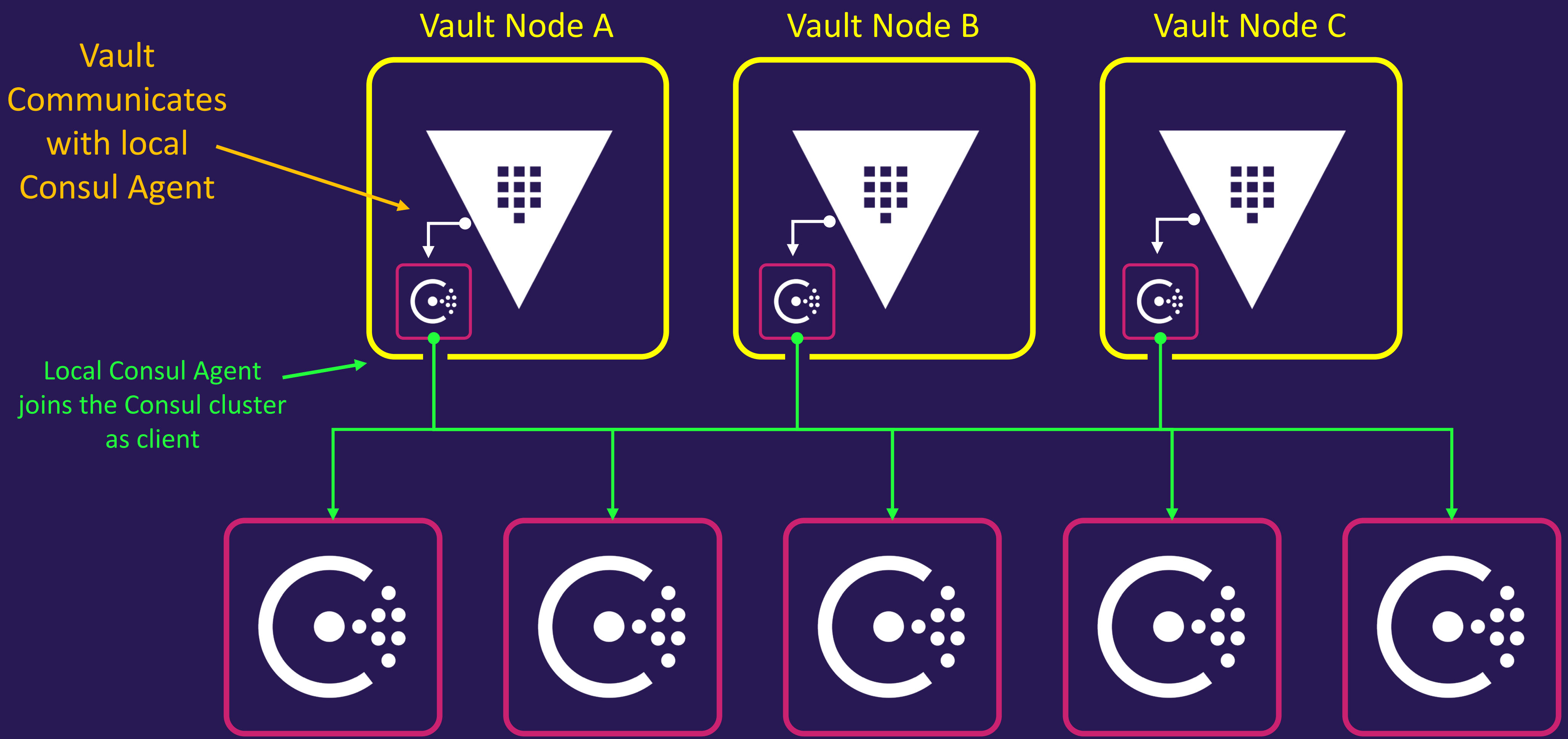
- Consul is deployed using multiple nodes and configured as a cluster
- Clusters are deployed in odd numbers (for voting members)
- All data is replicated among all nodes in the cluster
- A leader election promotes a single Consul node as the leader
- The leader accepts new logs entries and replicates to all other nodes
- Consul cluster for Vault storage backend shouldn’t be used for Consul functions in a production setting

Deploying the Consul Storage Backend



Special Install of Consul using Redundancy Zones

Deploying the Consul Storage Backend


Vault Node A
Vault Node B
Vault Node C
Vault Communicates with local Consul Agent
Local Consul Agent joins the Consul cluster as client

Deploying the Consul Storage Backend

Example Consul Server Configuration File

```
 storage "consul" {
  address = "127.0.0.1:8500"
  path    = "vault/"
  token   = "1a2b3c4d-1234-abdc-1234-1a2b3c4d5e6a"
}
listener "tcp" {
 address = "0.0.0.0:8200"
 cluster_address = "0.0.0.0:8201"
 tls_disable = 0
 tls_cert_file = "/etc/vault.d/client.pem"
 tls_key_file = "/etc/vault.d/cert.key"
 tls_disable_client_certs = "true"
}
seal "awskms" {
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101",
  endpoint = "example.kms.us-east-1.vpce.amazonaws.com"
}
api_addr = "https://vault-us-east-1.example.com:8200"
cluster_addr = " https://node-a-us-east-1.example.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```

https://github.com/btkrausen/hashicorp/blob/master/vault/config_files/vault.hcl

Deploying the Consul Storage Backend

Example Consul Server Configuration File

```
{
 "log_level": "INFO",
 "server": true,
 "key_file": "/etc/consul.d/cert.key",
 "cert_file": "/etc/consul.d/client.pem",
 "ca_file": "/etc/consul.d/chain.pem",
 "verify_incoming": true,
 "verify_outgoing": true,
 "verify_server_hostname": true,
 "ui": true,
 "encrypt": "xxxxxxxxxxxxxx",
 "leave_on_terminate": true,
 "data_dir": "/opt/consul/data",
 "datacenter": "us-east-1",
 "client_addr": "0.0.0.0",
 "bind_addr": "10.11.11.11",
 "advertise_addr": "10.11.11.11",
 "bootstrap_expect": 5,
 "retry_join": ["provider=aws tag_key=Environment-Name tag_value=consul-cluster region=us-east-1"],
 "enable_syslog": true,
 "acl": {
    "enabled": true,
    "default_policy": "deny",
    "down_policy": "extend-cache",
    "tokens": {
      "agent": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  }
},
  "performance": {
   "raft_multiplier": 1
 }
}
```

https://github.com/btkrausen/hashicorp/blob/master/consul/config.hcl

Looking for More on Consul?

For a deeper dive on Consul, check out my dedicated course on Consul:

Getting Started with HashiCorp Consul

Coupons Available on github.com/btkrausen/hashicorp

Deploying the Integrated Storage Backend

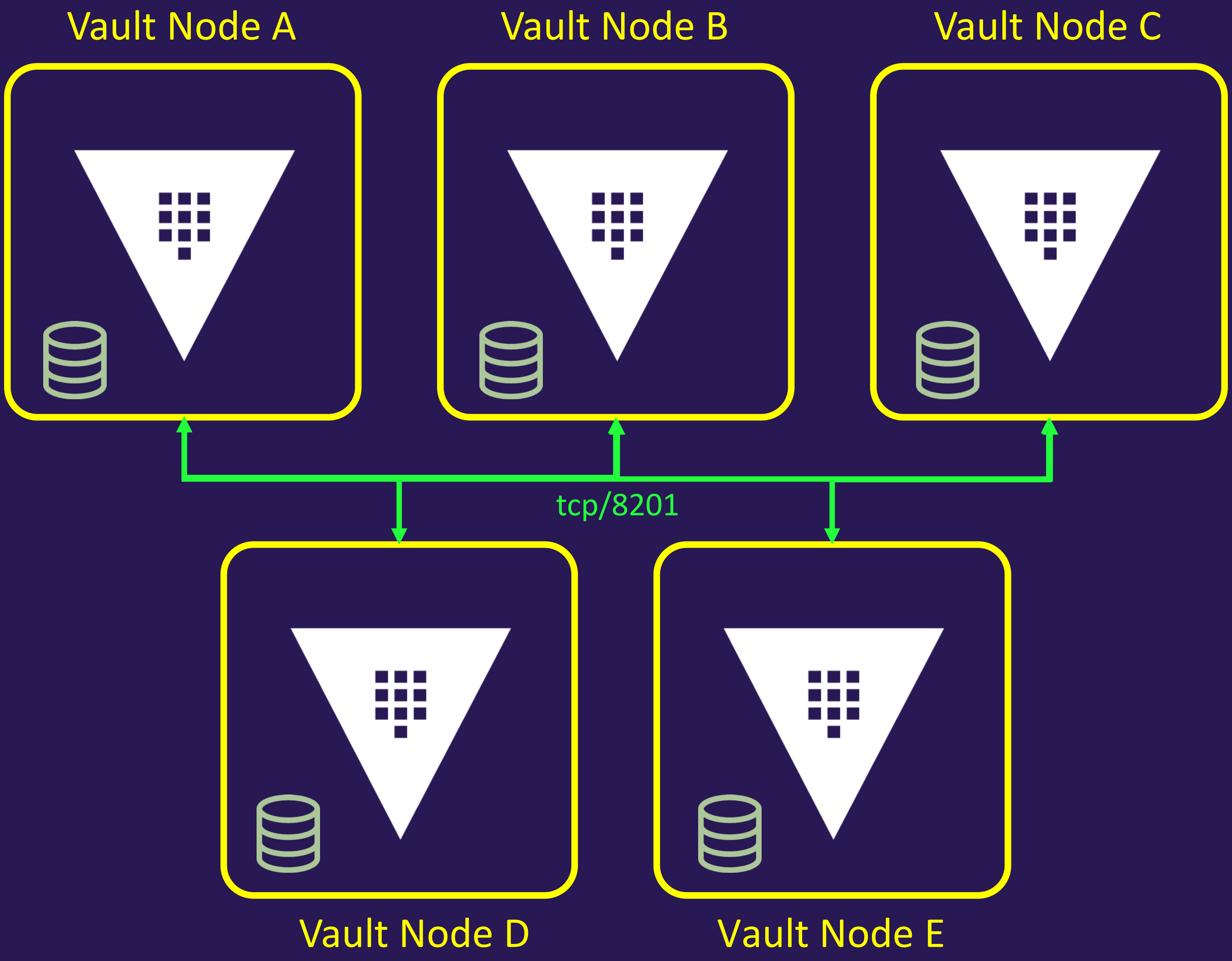
- Vault Internal Storage Option
- Leverages Raft Consensus Protocol
- All Vault nodes have copy of Vault's Data
- Eliminates Network Hop to Consul

- Supports High Availability
- Only need to troubleshoot Vault
- Built-in Snapshots For Data Retention
- HashiCorp Supported

Deploying the Integrated Storage Backend

- Integrated Storage (aka Raft) allows Vault nodes to provide its own replicated storage across the Vault nodes within a cluster
- Define a local path to store replicated data
- All data is replicated among all nodes in the cluster
- Eliminates the need to also run a Consul cluster and manage it

Deploying the Integrated Storage Backend



Deploying the Integrated Storage Backend

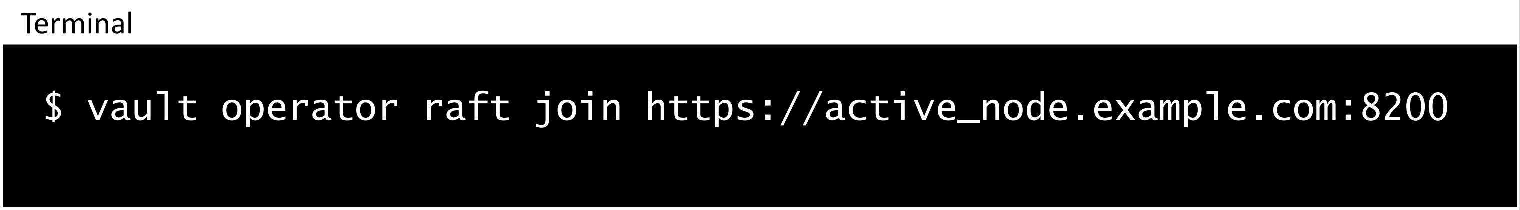
Example Vault Server Configuration File

```
storage "raft" {
  path    = "/opt/vault/data"
  node_id = "node-a-us-east-1.example.com"
  retry_join {
    auto_join = "provider=aws region=us-east-1 tag_key=vault tag_value=us-east-1"
  }
}
listener "tcp" {
 address = "0.0.0.0:8200"
 cluster_address = "0.0.0.0:8201"
 tls_disable = 0
 tls_cert_file = "/etc/vault.d/client.pem"
 tls_key_file = "/etc/vault.d/cert.key"
 tls_disable_client_certs = "true"
}
seal "awskms" {
  region = "us-east-1"
  kms_key_id = "12345678-abcd-1234-abcd-123456789101",
  endpoint = "example.kms.us-east-1.vpce.amazonaws.com"
}
api_addr = "https://vault-us-east-1.example.com:8200"
cluster_addr = " https://node-a-us-east-1.example.com:8201"
cluster_name = "vault-prod-us-east-1"
ui = true
log_level = "INFO"
```

https://github.com/btkrausen/hashicorp/blob/master/vault/config_files/vault_int_storage.hcl

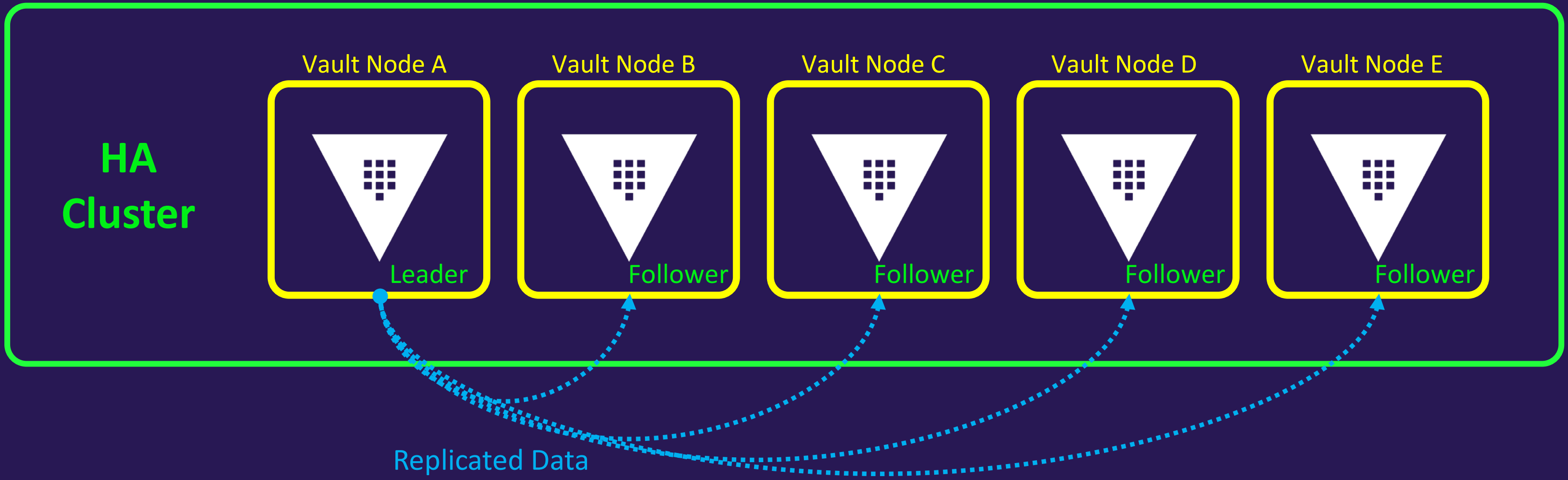
Deploying the Integrated Storage Backend

- Manually join standby nodes to the cluster using the CLI:



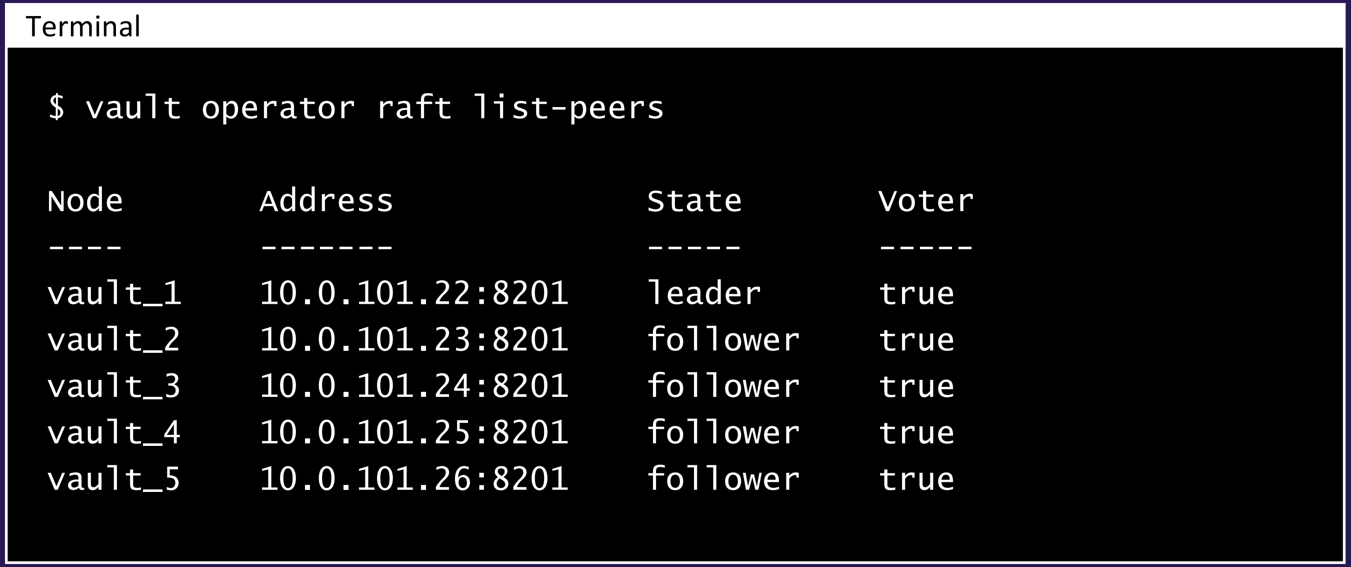
Terminal

```
$ vault operator raft join https://active_node.example.com:8200
```



Deploying the Integrated Storage Backend

- List the cluster members



```
$ vault operator raft list-peers

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

END OF SECTION