Compare and Configure Secrets Engines

Objective 5 – Compare and Configure Secrets Engines

Objective 5a – Choose a secret method based on use case

Objective 5b – Contrast Dynamic Secrets vs. Static Secrets and their use cases

Objective 5c – Define transit secrets engine

Objective 5d – Define secrets engines

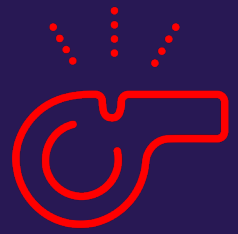
Problems with Managing Static Secrets

Expiration

- Secrets never expire
- Required by legacy apps

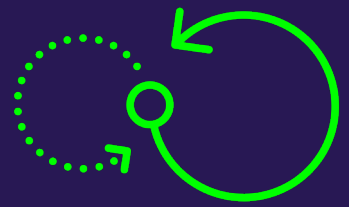
Secrets Aren't Secret

- Secrets are often shared among team members
- No accountability



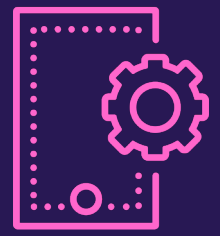
Validity

- Secrets are valid 24/7
- Frequently the target for penetrators



Rotation

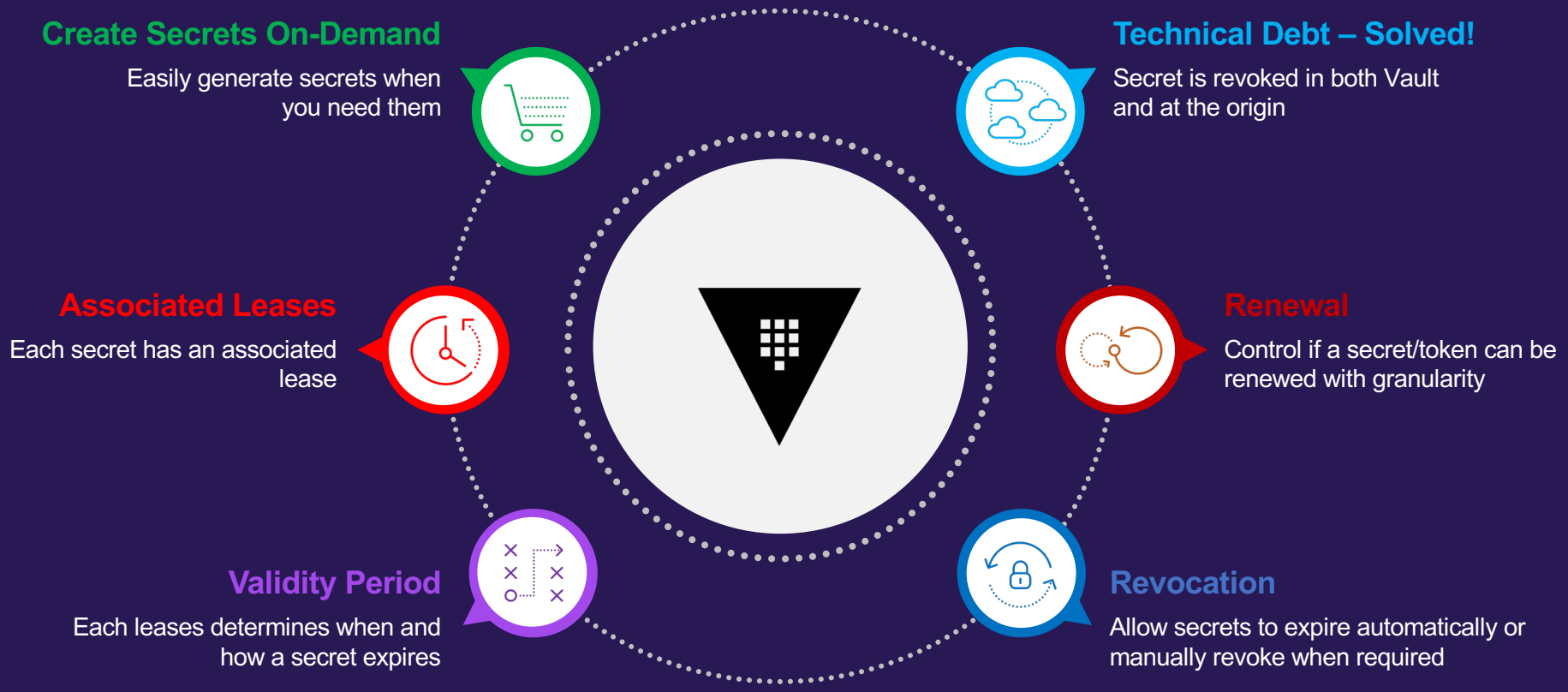
- Secrets are rarely, if ever, rotated
- Manual process



Long-Lived

- Secrets tend to live perpetually
- Result of technical debt, employee turnover, etc

Why You Should Use Dynamic Secrets



Create Secrets On-Demand

Easily generate secrets when you need them

Technical Debt – Solved!

Secret is revoked in both Vault and at the origin

Associated Leases

Each secret has an associated lease

Renewal

Control if a secret/token can be renewed with granularity

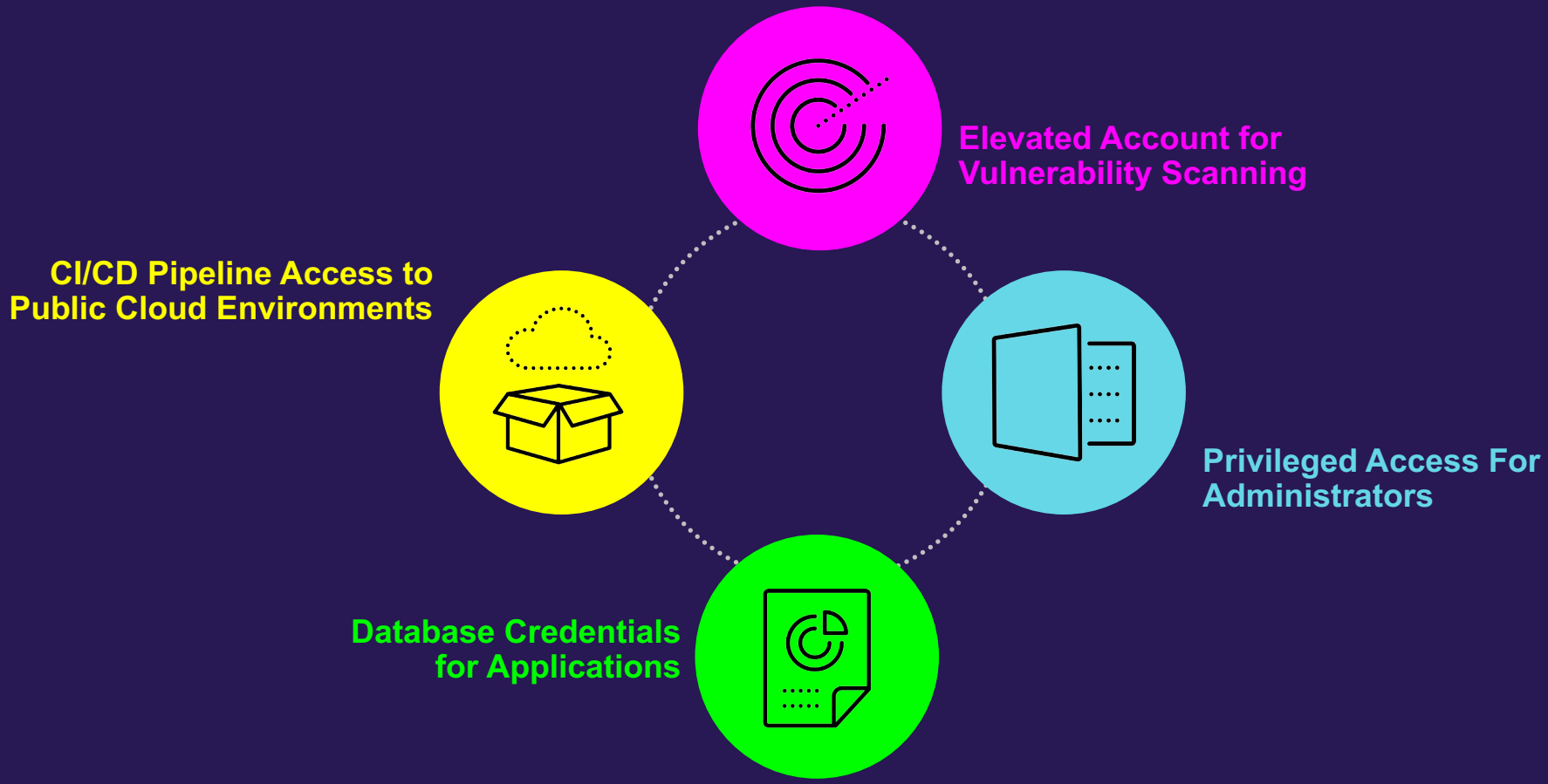
Validity Period

Each leases determines when and how a secret expires

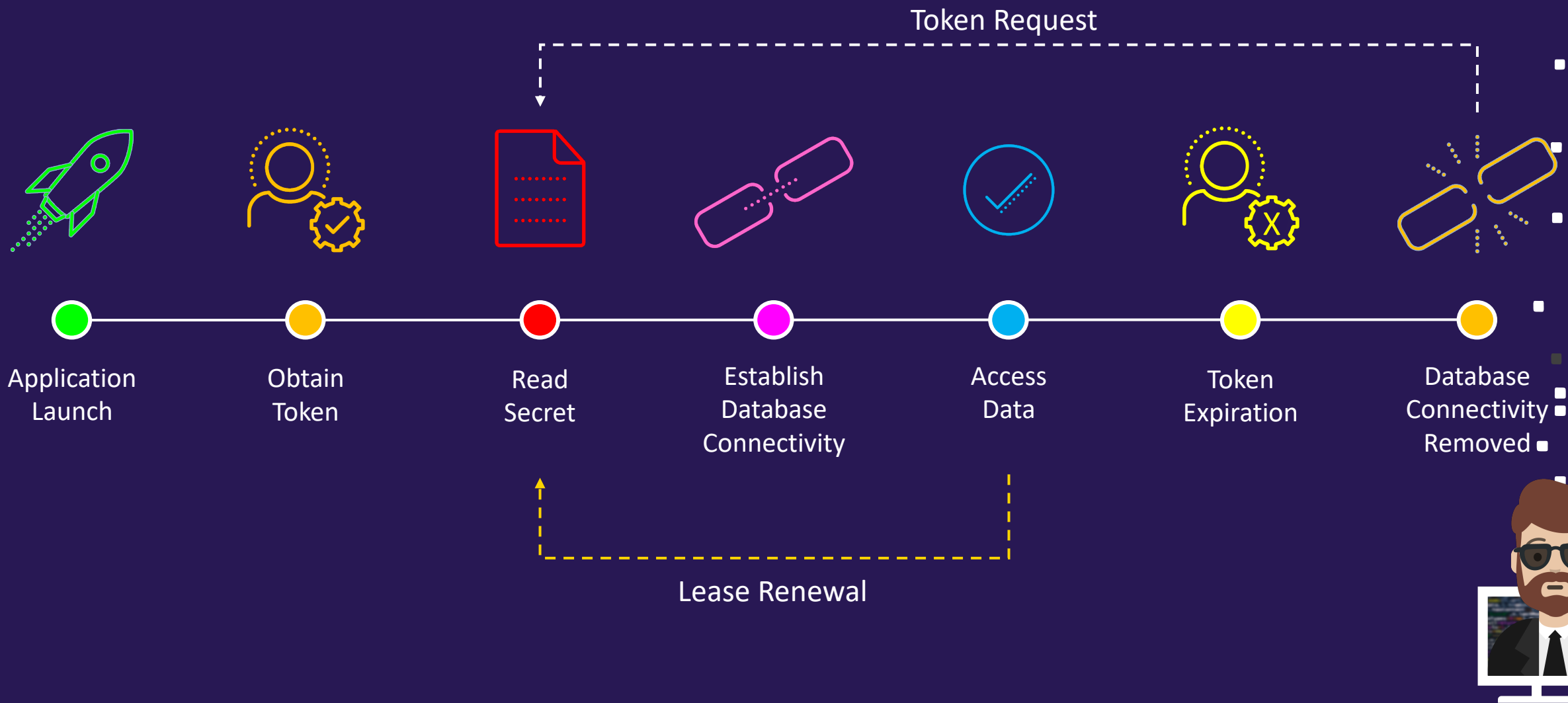
Revocation

Allow secrets to expire automatically or manually revoke when required

Why You Should Use Dynamic Secrets



Example – Application Using Vault



Example – Pipeline Using Vault

Code Development

Commit to Repo

Test & Build

Obtain Secret

Provision Resources

Application Launch

Revoke Secret

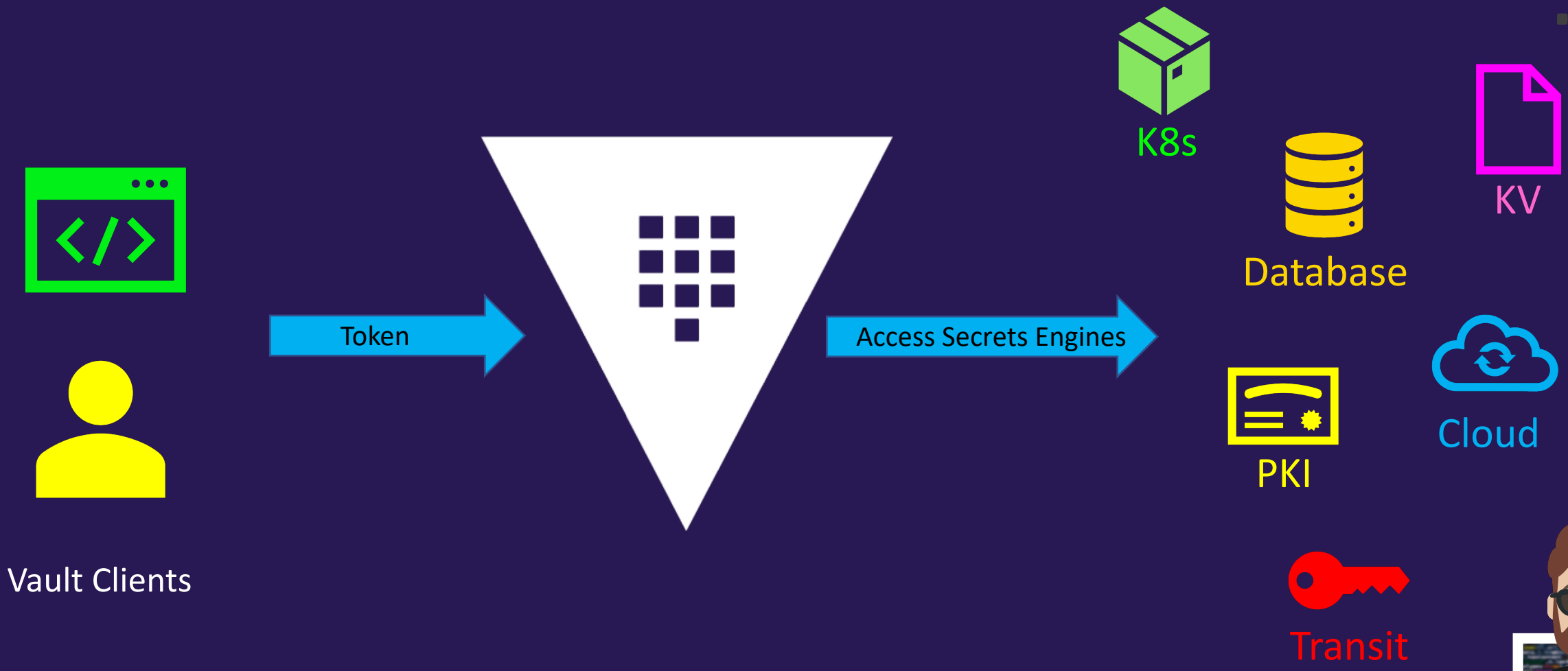
Vault Interfaces

VALID FOR 3 DAYS

Vault Interfaces

We present our key.
We don't authenticate again

Vault Secrets Engines



Secrets Engines are THE Reason you Deploy Vault

Secrets Engines

- Secrets engines are components that can store, generate, or encrypt data
  - Many secrets engines can be enabled in Vault
  - You can even enable multiple instances of the same secrets engine
  - Secrets engines are plugins that extend the functionality of Vault

- Secrets engines are enabled and isolated at a path
  - All interactions with the secrets engine are done using the path
  - Path must be unique

Secrets Engines

- Secrets engines are components that can store, generate, or encrypt data
  - Many secrets engines can be enabled in Vault
  - You can even enable multiple instances of the same secrets engine
  - Secrets engines are plugins that extend the functionality of Vault

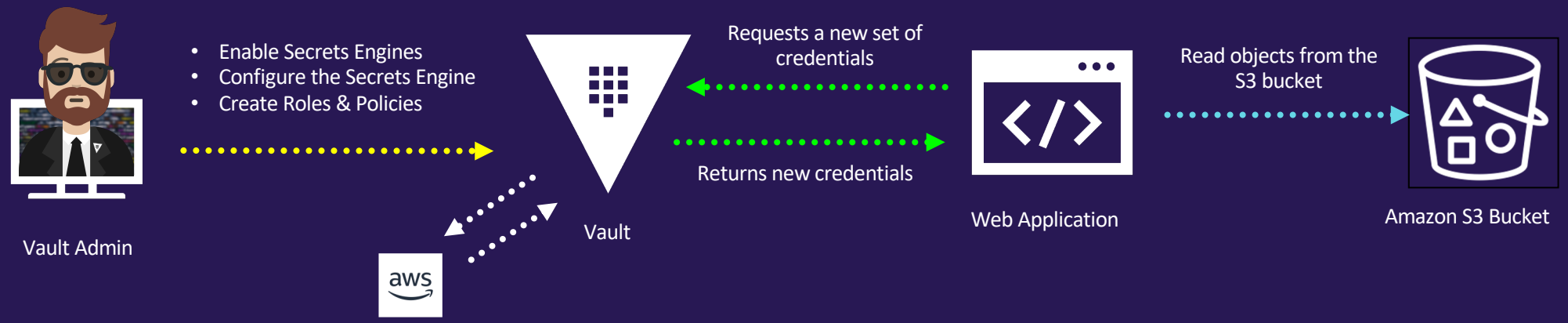
- Secrets engines are enabled and isolated at a path
  - All interactions with the secrets engine are done using the path
  - Path must be unique

What Is a Secret?

- A secret is anything an organization deems sensitive within their organization
  - Username & Password
  - TLS Certificate – private cert & cert
  - API Key
  - Database Credentials
  - Application Data
  - Anything else you don't want stored in plaintext

Secrets as a Service

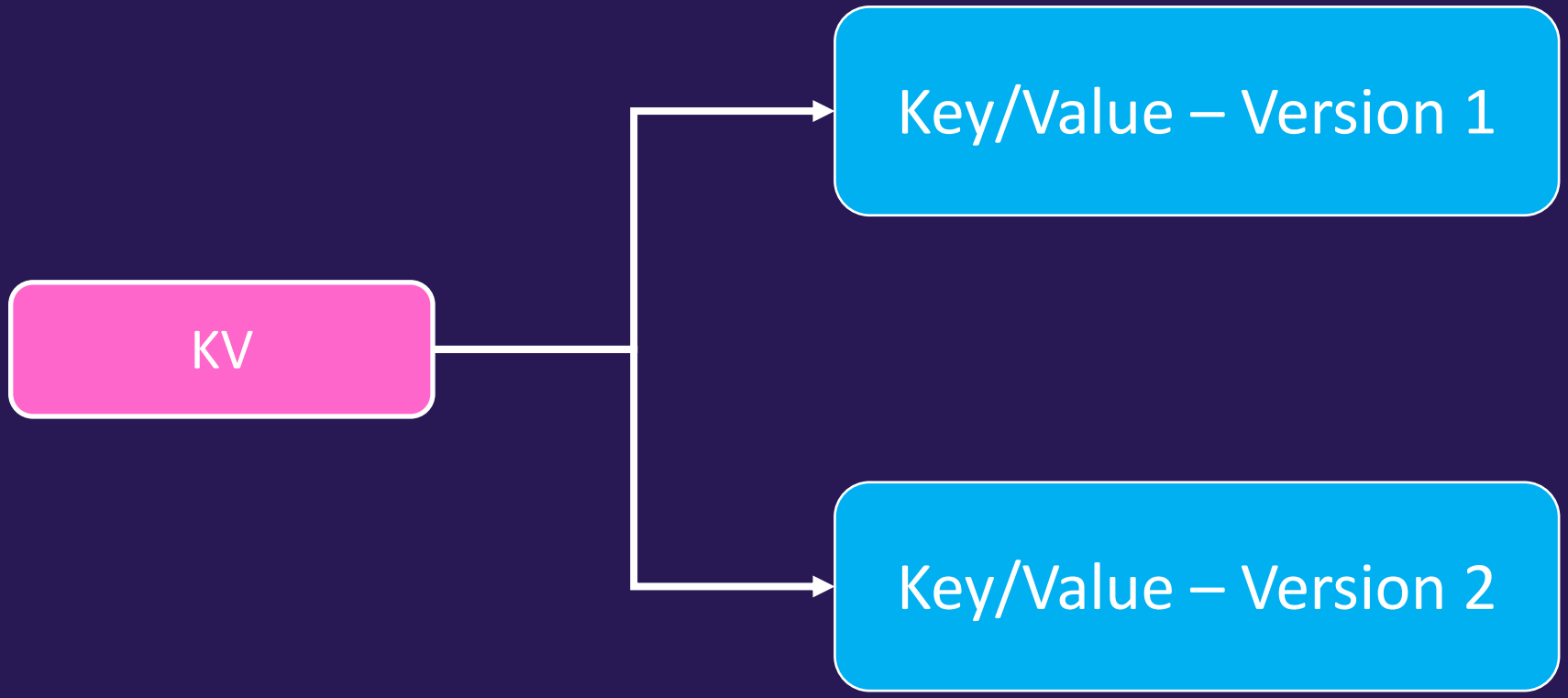
- Use Vault to generate and manage the lifecycle of credentials on-demand
- No more sharing credentials
  - Credentials get revoked automatically at the end of its lease
  - Audit trail can identify points of compromise
- Use policies to control the access based on the client's role



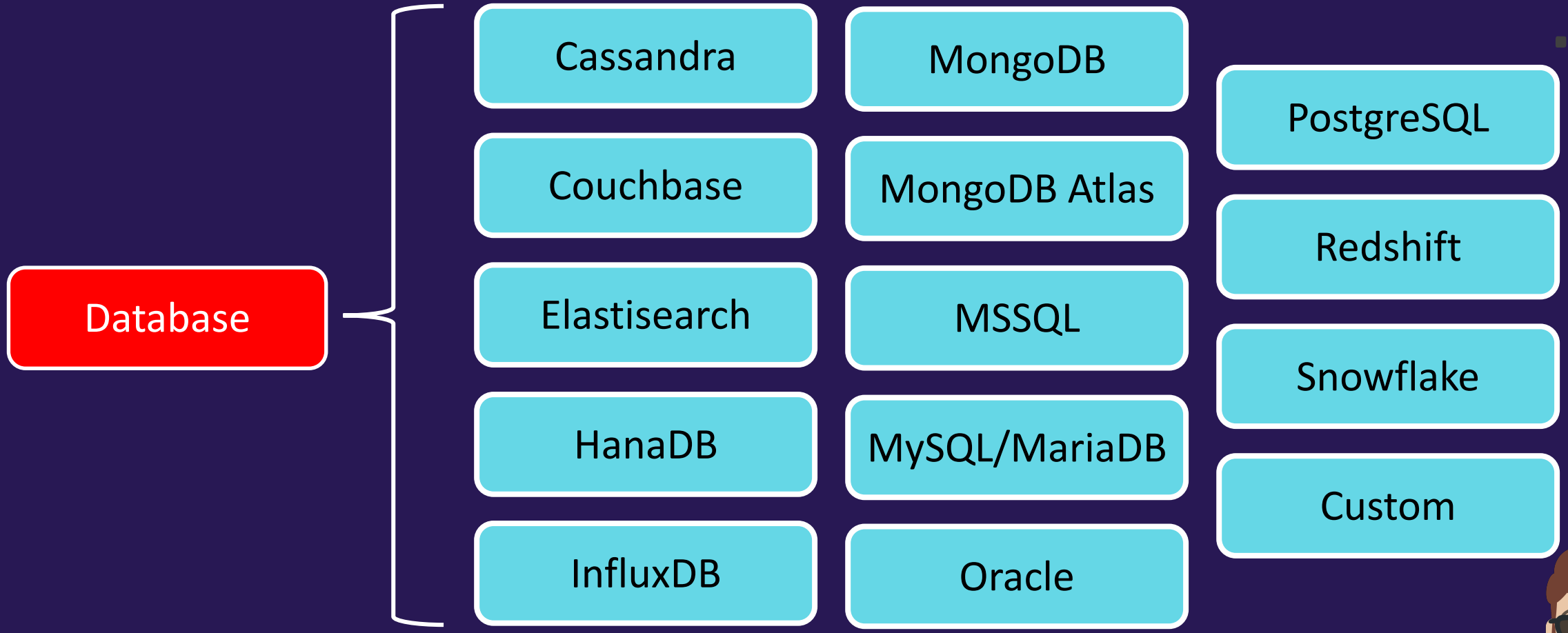
Secrets Engines

Active Directory	Database	Identity	SSH
AliCloud	Google Cloud	MongoDB Atlas	Terraform Cloud
AWS	Google Cloud KMS	Nomad	TOTP
Azure	Key Management	OpenLDAP	Transform
Consul	KMIP	PKI	Transit
Cubbyhole	KV	RabbitMQ	Venafi

Secrets Engines



Secrets Engines



Secrets Engines

Key Management

- Azure Key Vault
- AWS KMS
- GCP Cloud KMS

Secrets Engines

Grouped by Function

Amazon Web Services
Microsoft Azure
Google Cloud Platform
Google Cloud KMS
Alibaba Cloud

Apache Cassandra
InfluxDB
MongoDB
Microsoft SQL
MySQL/MariaDB
PostgreSQL
Oracle
SAP HANA
Snowflake

Active Directory
Consul
Cubbyhole
Key/Value
Key Management
Identity
RabbitMQ
Nomad
SSH
TOTP
Terraform Cloud

KMIP
PKI
Transform
Transit
Venafi

Secrets Engines

What Does Each Secrets Engine Do?

Active Directory	Rotate credentials for an Active Directory account – the account must already exist in Active Directory and Vault uses the existing credentials to rotate the password based on the threshold
AliCloud	generates AliCloud access tokens based on RAM policies, or AliCloud STS credentials based on RAM roles.
AWS	generates AWS credentials based on an IAM user, STS credentials, or a federation token.
Azure	generates Azure service principals along with role and group assignments
Consul	generates Consul API tokens dynamically based on Consul ACL policies.

Secrets Engines

What Does Each Secrets Engine Do?

Cubbyhole	KV store used to store sensitive data – tied directory to a token and all data expires when the token itself expires – no other token can read the data stored inside of another token's cubbyhole
Database	generates dynamic credentials on the targeted database server – the credentials provided are based on the statement sent to the database during credential generation
Google Cloud	generates Google Cloud service account keys and OAuth tokens based on IAM policies
Google Cloud KMS	provides encryption and key management via Google Cloud KMS
Key Management (ENTERPRISE ONLY)	Provides distribution and lifecycle management of cryptographic keys in various key management service (KMS) providers

Secrets Engines

What Does Each Secrets Engine Do?

KMIP (ENTERPRISE ONLY)	allows Vault to act as a Key Management Interoperability Protocol (KMIP) server provider and handle the lifecycle of its KMIP managed objects.
KV	Enables Vault to store key/value objects in path – has a non-versioned (KV) and a versioned (KVv2) option
Identity	identity management solution for Vault- enabled by default – cannot enable again nor can you remove – essential for Vault
MongoDB Atlas	create a Programmatic API key for each lease that provide appropriate access to the defined MongoDB Atlas project or organization with appropriate role(s)
Nomad	generates Nomad ACL tokens dynamically based on pre-existing Nomad ACL policies

Secrets Engines

What Does Each Secrets Engine Do?

OpenLDAP	allows management of LDAP entry passwords as well as dynamic creation of credentials
PKI	Generates dynamic x.509 certificates – turns Vault into a certificate authority – Vault can be a root CA or an intermediate CA
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
Terraform Cloud	generates Terraform Cloud API tokens dynamically for Organizations, Teams, and Users

Secrets Engines

What Does Each Secrets Engine Do?

TOTP	generates time-based credentials according to the TOTP standard
Transform (ENTERPRISE ONLY)	handles secure data transformation and tokenization against provided input value
Transit	handles cryptographic functions on data in-transit – provides encryption-as-a-service – centralize encryption functionality across the organization
Venafi	provides applications with the ability to dynamically generate SSL/TLS certificates that serve as machine identities using Venafi Trust Protection Platform or Venafi Cloud

Enabling a Secrets Engine

- Cubbyhole and Identity are enabled by default (can't disable)
- Any other secrets engine must be enabled
  - Can enable using the CLI, API, or UI (most)

- Secrets engines are enabled and isolated at a path
  - All interactions with the secrets engine are done using the path
  - Path must be unique
  - Paths do not need to match the secrets engines name or type
    - Make them meaningful for you and your organization

Secrets Engine - Responsibilities

Privileged User
(Vault Admin, Security Team)

1. Enable the Secrets Engine
2. Configure the connection to the backend platform (AWS, Database, etc.)
3. Create roles that define permissions to the backend platform
4. Create policies that grant permission to read from the secrets engine

Vault Clients
(users, apps, services, machines, etc.)

1. Read a set of credentials using token and associated policy
2. Renew the lease before its expiration if needed (or permitted)
3. Renew the token if needed (or permitted)

Working with a Secrets Engine

Command Line Interface (CLI)

Use the `vault secrets` command

- `disable`
- `enable`
- `list`
- `move`
- `tune`

Terminal

```
$ vault secrets enable aws
Success! Enabled the aws secrets engine at: aws/
```

Terminal

```
$ vault secrets tune -default-lease-ttl=72h pki/
```

Working with a Secrets Engine

Command Line Interface (CLI)

Helpful flags to use with the `vault secrets` command

- `vault secrets list -detailed`
- `vault secrets enable -path=developers kv`
- `vault secrets enable -description="my first kv" kv`

Terminal

```
$ vault secrets enable -description="Static Secrets" -path="cloud-kv" kv-v2
```

Working with a Secrets Engine

Command Line Interface (CLI)

```
vault secrets enable path=bryan kv-v2
```

Type of Vault object you want to work with

Subcommand

Define a custom path to enable the secrets engine on

The type of secrets engine you want to enable

Working with a Secrets Engine

Command Line Interface (CLI)

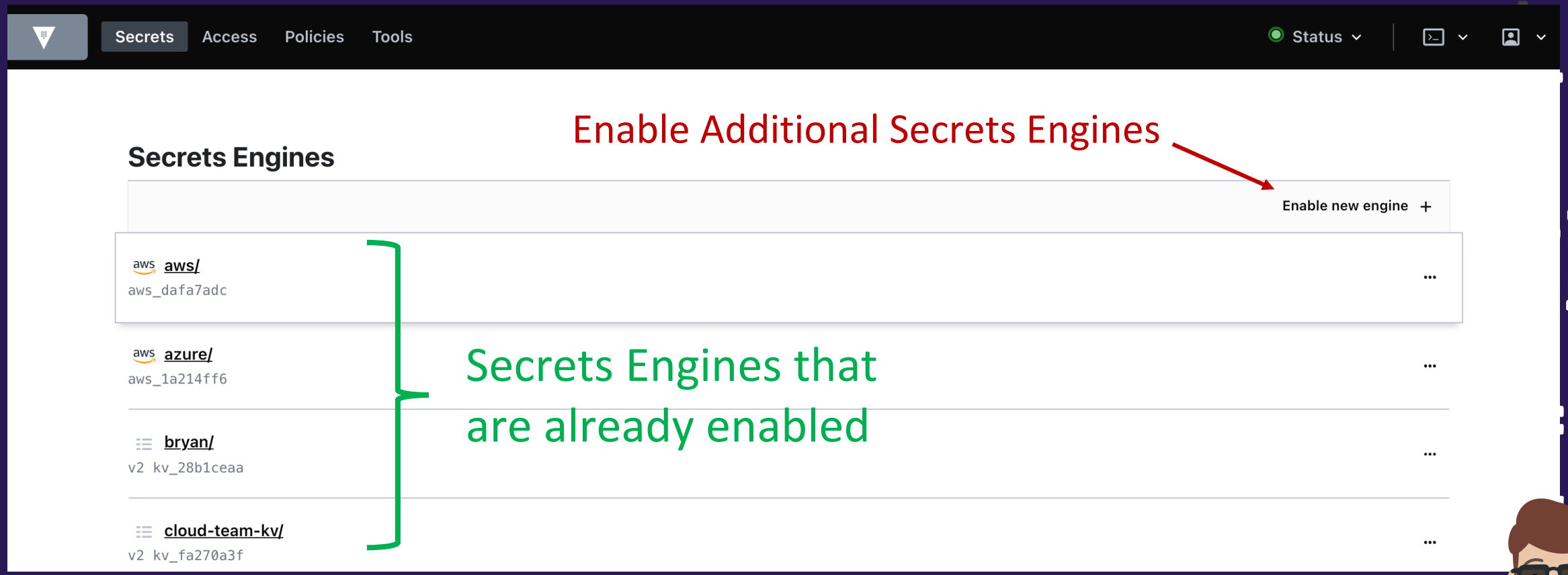
Use the `vault secrets list` command

Terminal

```
$ vault secrets list

Path                Type         Accessor                Description
----                ----         --------                -----------
aws/                aws          aws_dafa7adc            n/a
azure/              aws          aws_1a214ff6            n/a
bryan/              kv           kv_28b1ceaa             n/a
cloud-team-kv/      kv           kv_fa270a3f             n/a
cubbyhole/          cubbyhole    cubbyhole_88c8e2e3      per-token private secret storage
dev-team-kv/        kv           kv_55c319c4             n/a
identity/           identity     identity_e60e93cb       identity store
kv-v2/              kv           kv_eea3206c             n/a
sys/                system       system_66b0d8ee         system endpoints used for
transit/            transit      transit_7b8038ca        n/a
```

Working with a Secrets Engine

User Interface (UI)

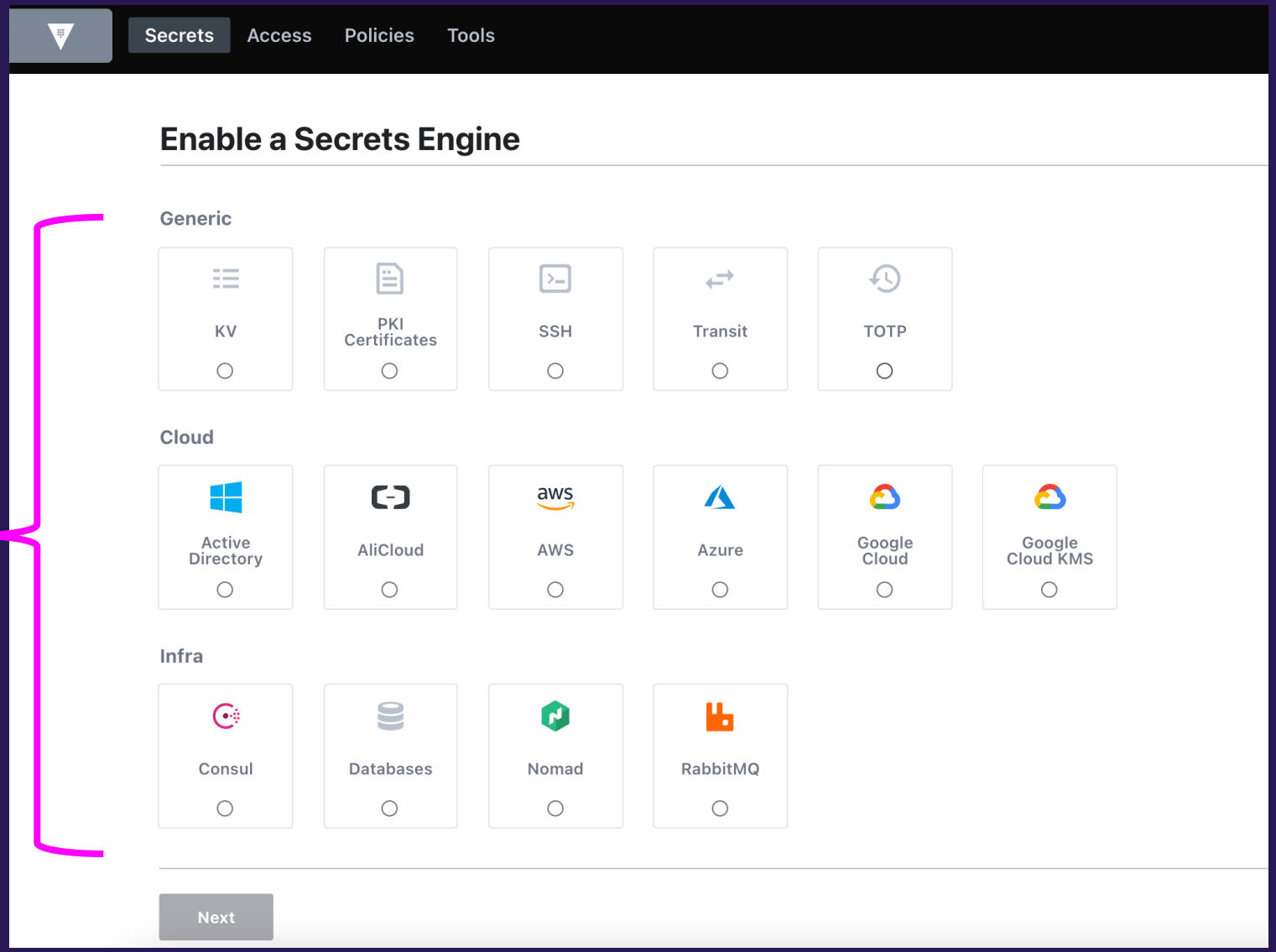


Working with a Secrets Engine

User Interface (UI)

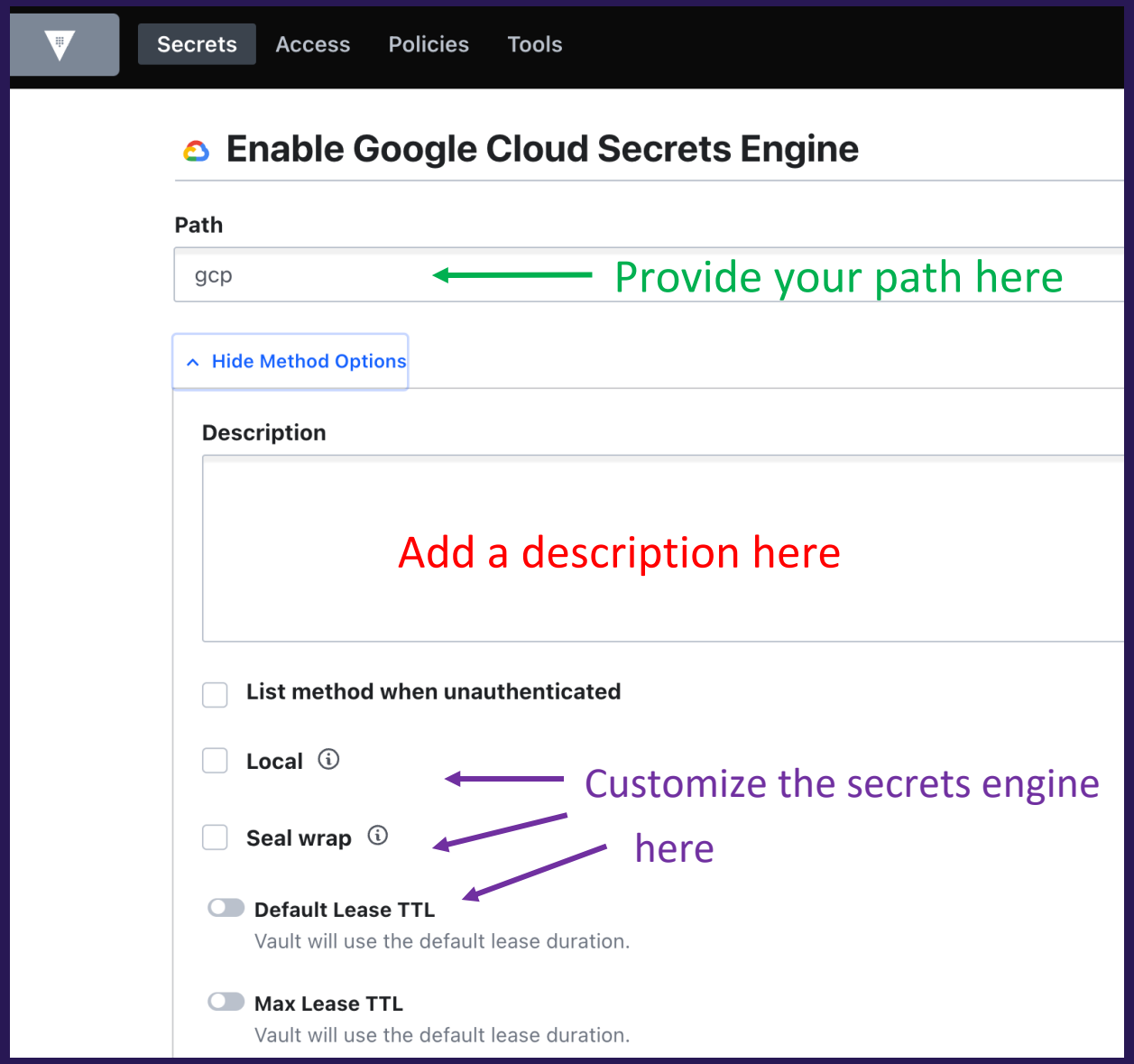
Choose Your Weapon

(I mean, Secrets Engine)



Working with a Secrets Engine

User Interface (UI)



Configuring a Secrets Engine

There are generally two steps when configuring a secrets engine that will generate dynamic credentials:

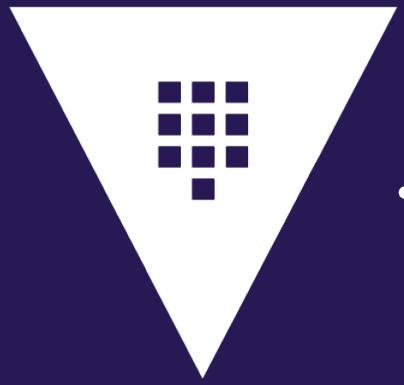
Step 1: Configure Vault with access to the platform

Step 2: Configure Roles based on permissions needed

Configuring a Secrets Engine

AWS Example

Step 1: Configure Vault with access to the platform:



Vault needs credentials to interact with the platform

Credentials can be provided using IAM credentials, an EC2 instance role, or setting environment variables

Configuring a Secrets Engine

AWS Example

Provide credentials to a secrets engine that gives Vault permission to create, list, and delete credentials on the platform:

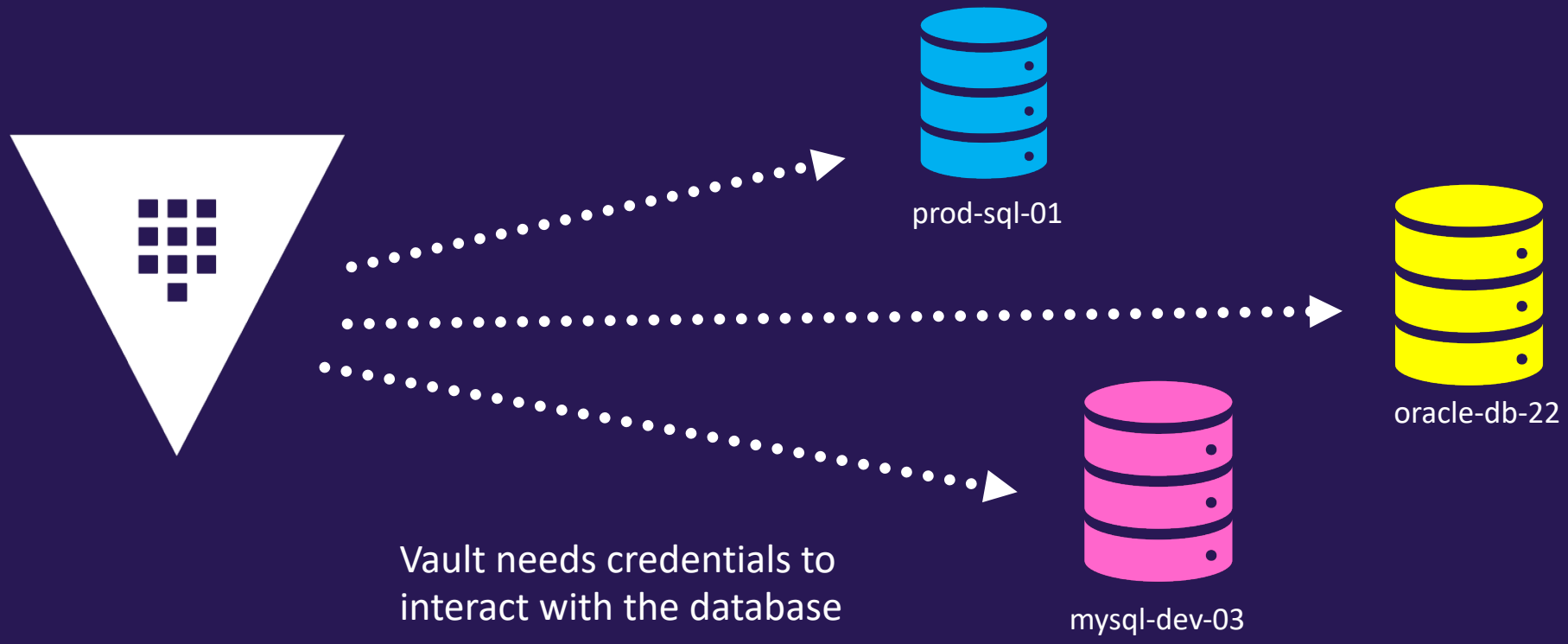
Terminal

```
$ vault write aws/config/root \
    access_key=AKIAIOSFODNN7EXAMPLE \
    secret_key= wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY \
    region=us-east-1

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

Configuring a Secrets Engine

Database Example

Step 1: Configure Vault with access to the platform:



Configuring a Secrets Engine

Database Example

Provide credentials to a secrets engine that gives Vault permission to create, list, and delete credentials on the platform:

Terminal

```
$ vault write database/config/prod-database \
    plugin_name=mysql-aurora-database-plugin \
    connection_url="{{username}}:{{password}}@tcp(prod.cluster.us-east-1.rds.amazonaws.com:3306)/" \
    allowed_roles="app-integration, app-lambda" \
    username="vault-admin" \
    password="vneJ4908fkd3084Bmrk39fmslslf#e&349"
```

Configuring a Secrets Engine - Roles

Vault does ***not*** know what permissions, groups, and policies you want to attach to generated credentials. Each ***role*** maps to a set of permissions on the targeted platform.

Step 2: Configure Roles based on permissions needed

Configuring a Secrets Engine - Roles

AWS Example

Step 2: Configure Roles based on permissions needed

Organizations Usually have Multiple AWS accounts

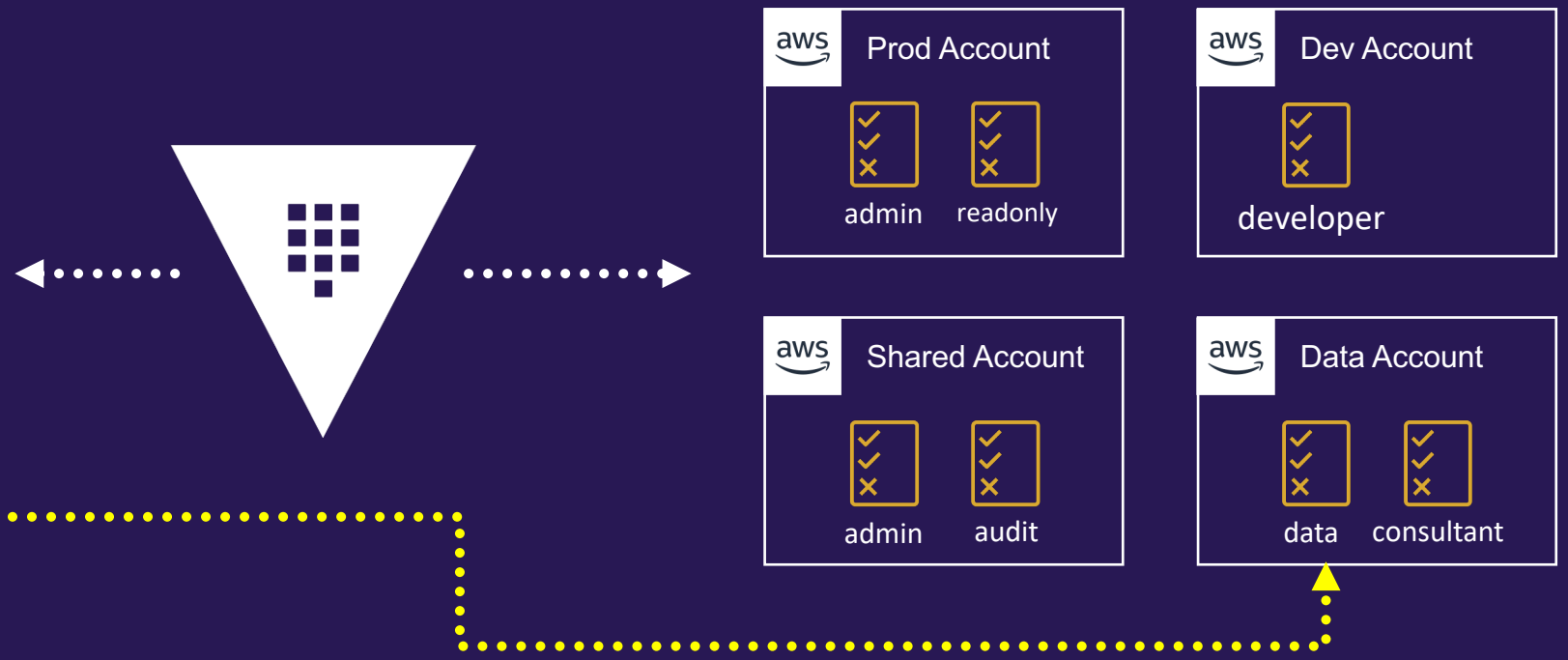
Configuring a Secrets Engine - Roles

AWS Example

Step 2: Configure Roles based on permissions needed

Roles

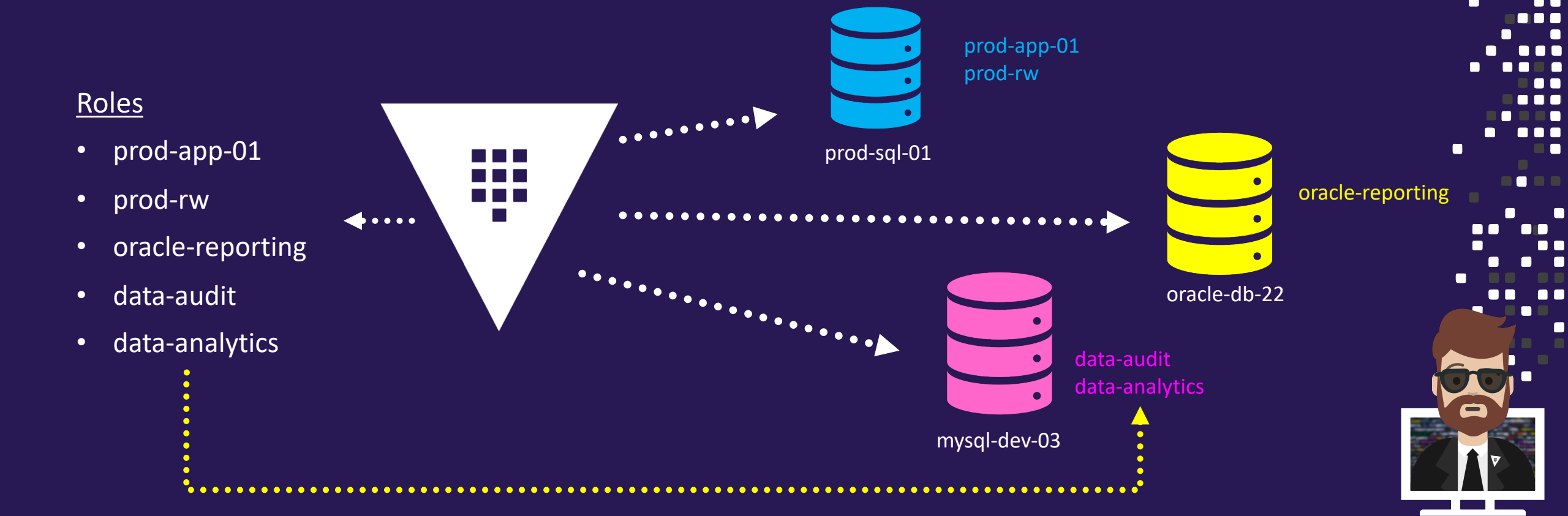
- prod-admin
- prod-readonly
- dev-developer
- shared-admin
- shared-audit
- data-datascientist
- data-consultant



Configuring a Secrets Engine - Roles

Database Example

Step 2: Configure Roles based on permissions needed

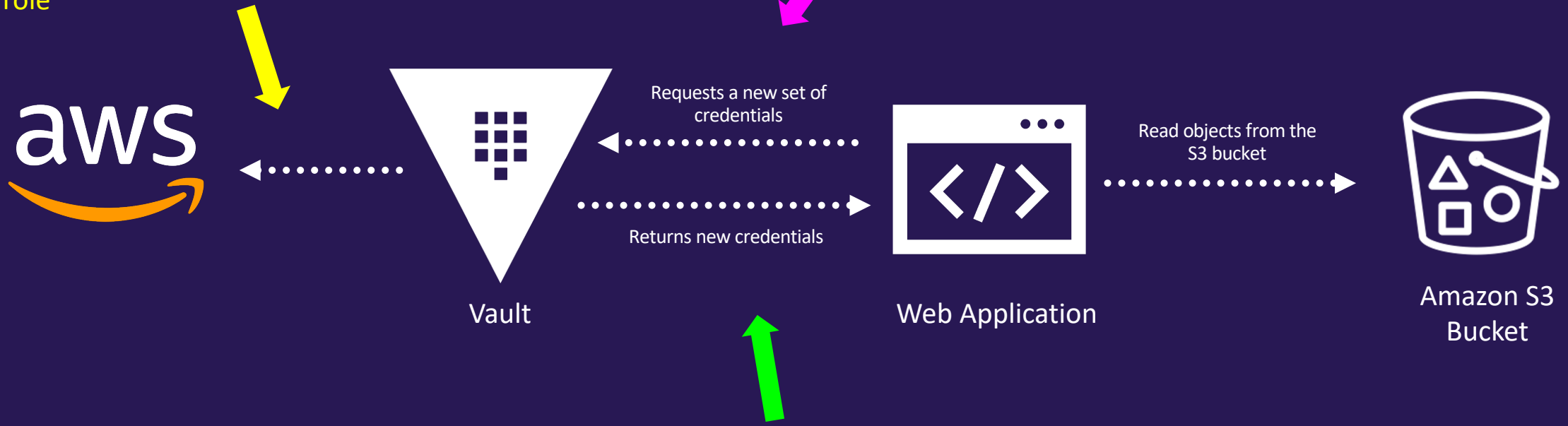


Generate Credentials

AWS Example

Vault uses the credentials we provided the secrets engine to generate the dynamic credentials for the requested role

Vault client references the Vault role when requesting dynamic credentials



Credentials returned are attached to a lease (TTL) and will expire after that TTL unless renewed by the application

Generate Credentials

AWS Example

Terminal

```
$ vault read aws/creds/data-consultant

Key                Value
---                -----
Lease              aws/creds/data-consultant/349dm20s4xp2
lease_duration     24h
lease_renewable    true
access_key         AKIAIOSFODNN7EXAMPLE
secret_key         wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
security_token     <nil>
```

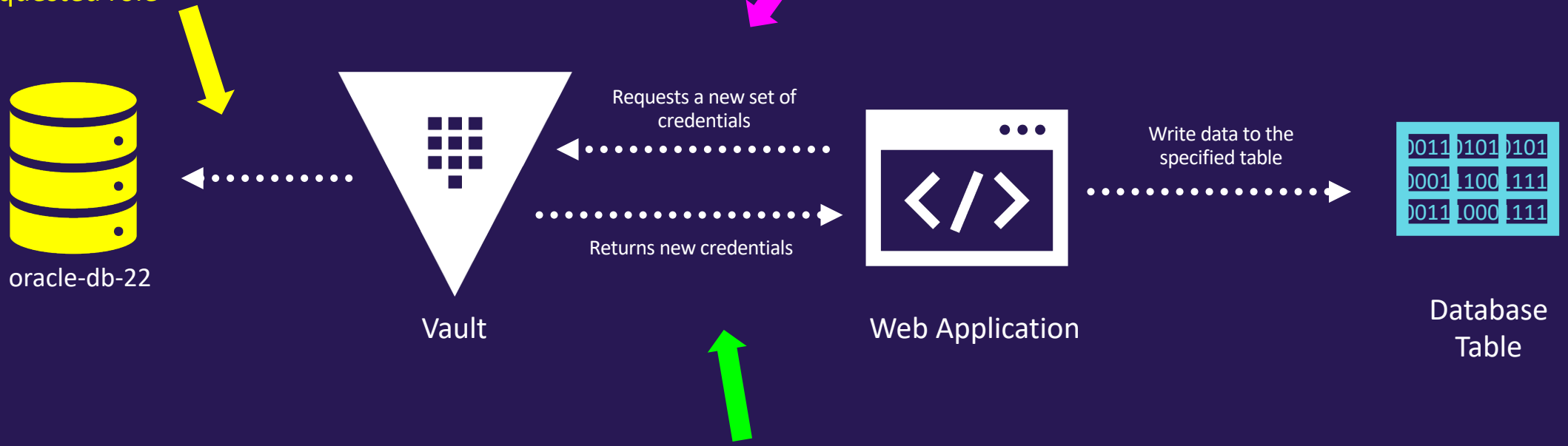
role we created

Generate Credentials

Database Example

Vault uses the credentials we provided the database secrets engine to generate the dynamic credentials for the requested role

Vault client references the Vault role when requesting dynamic credentials



Credentials returned are attached to a lease (TTL) and will expire after that TTL unless renewed by the application

Generate Credentials

Database Example

Terminal

```
$ vault read database/creds/oracle-reporting

Key Value
--- -----
lease_id          database/creds/my-role/2f14c-4aa224b9-ad944a8d4de6
lease_duration    1h
lease_renewable   true
password          yRUSyd-vPYDg5NkU9kDg
username          V_VAULTUSE_MY_ROLE_SJJUK3Q8W3BKAYAN8S62_1602543009
```

role we created

Summary

There are generally two steps when configuring a secrets engine that will generate dynamic credentials:

Step 1: Configure Vault with access to the platform

Step 2: Configure Roles based on permissions needed

Don't forget that the Vault client must be authenticated before it can request dynamic credentials

Key/Value Secrets Engine

- Key/Value secrets engine is used to store static secrets
  - There are two versions: v2 (kv-v2) is versioned but v1 (v1) is not
  - Secrets are accessible via UI, CLI, and API – interactive or automated
  - Access to KV paths are enforced via policies (ACLs)

- Like everything else in Vault, secrets written to the KV secrets engine are encrypted using 256-bit AES

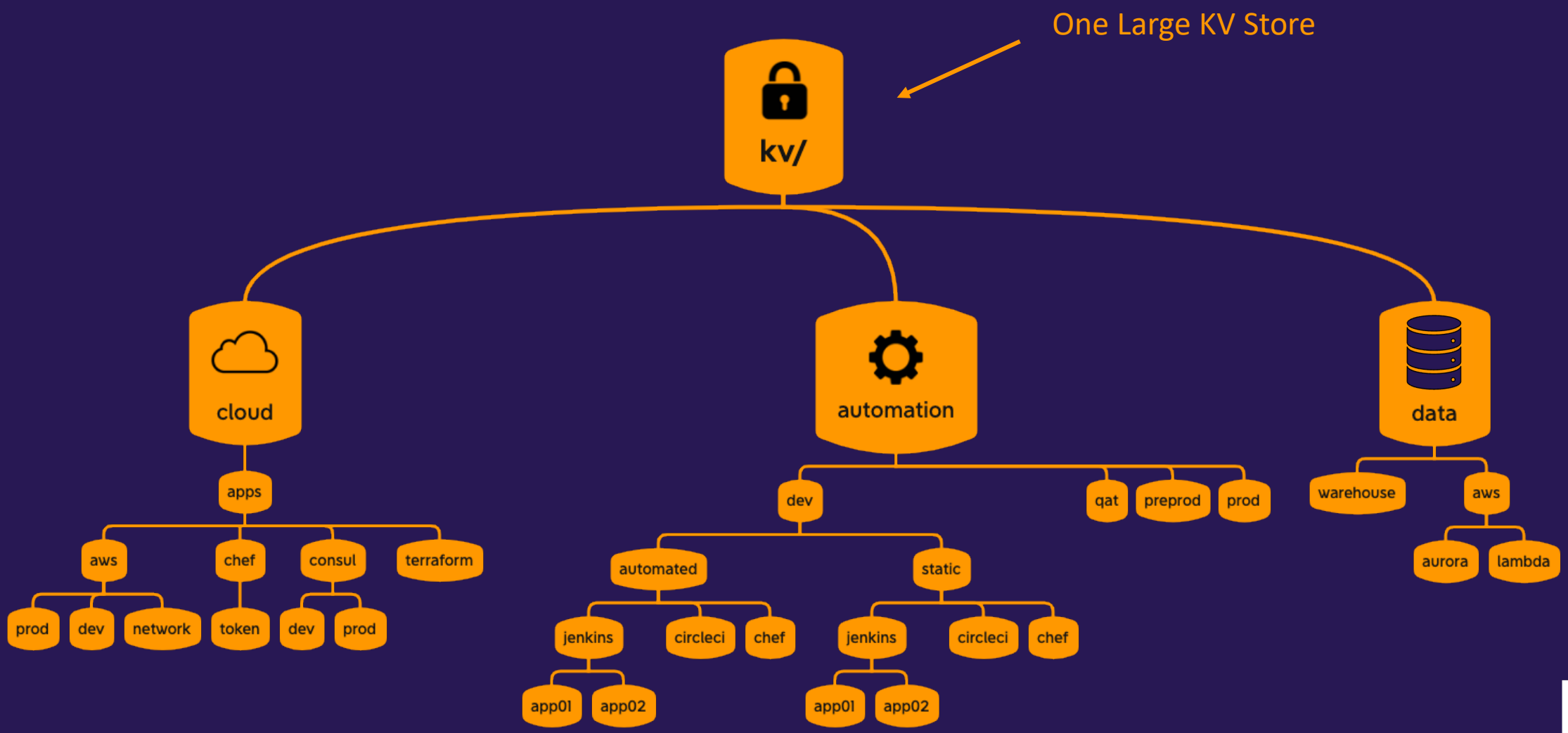
The most frequently used secrets engine in Vault

Key/Value Secrets Engine

- Key/Value secrets engine can be enabled at different paths
  - Each key/value secrets engine is isolated and unique
- Secrets are stored as key-value pairs at a defined path – (e.g., secret/applications/web01)
  - Writing a new secret will ***replace the old value***
  - Writing a new secret requires the `create` capability
  - Updating/overwriting a secret to an existing path requires `update` capability
- When you run Vault in –dev server mode, Vault enables a KV v2 secrets engine at the `secret/` path, by default

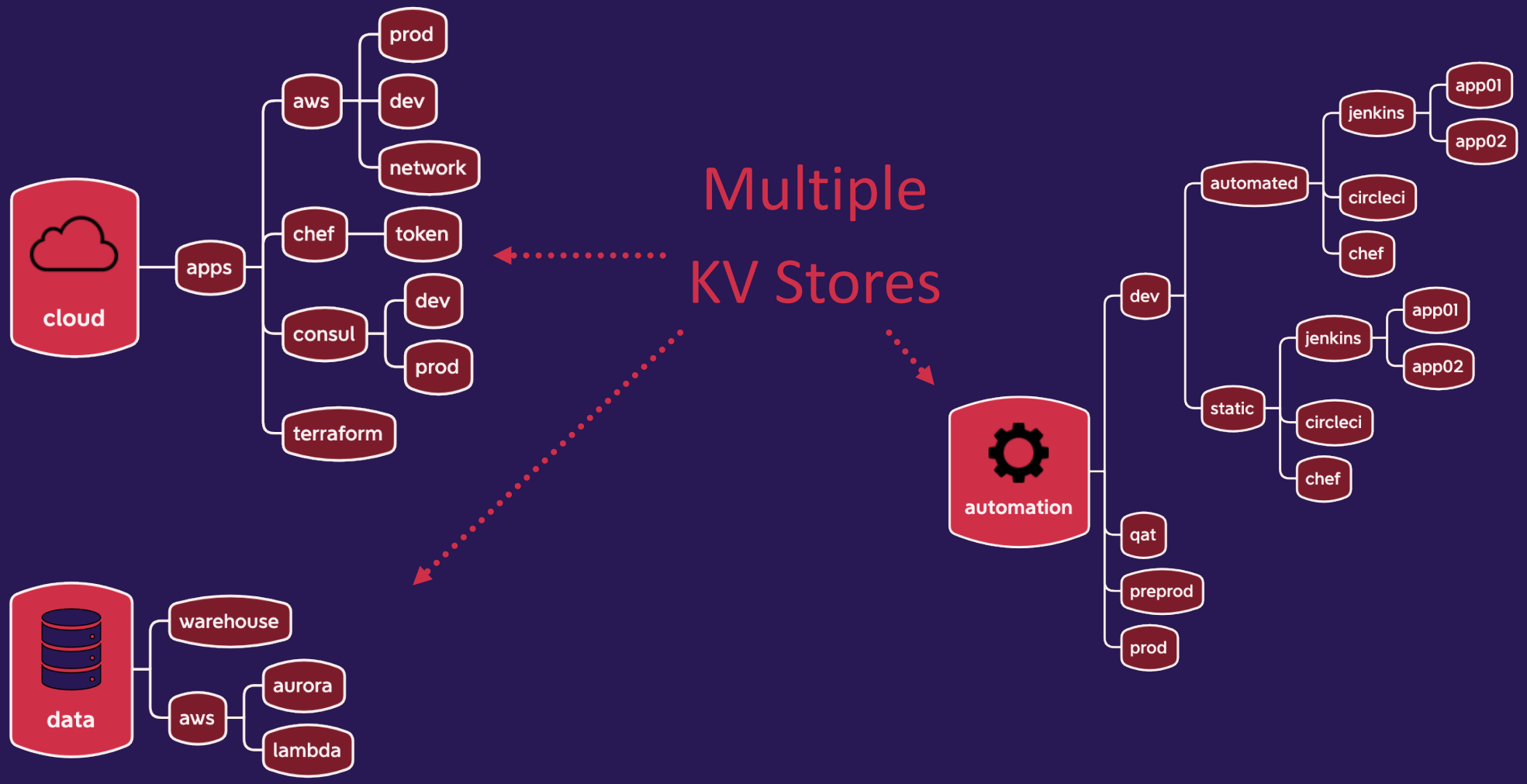
Key/Value Secrets Engine

Organize Data However It Makes Sense to Your Organization



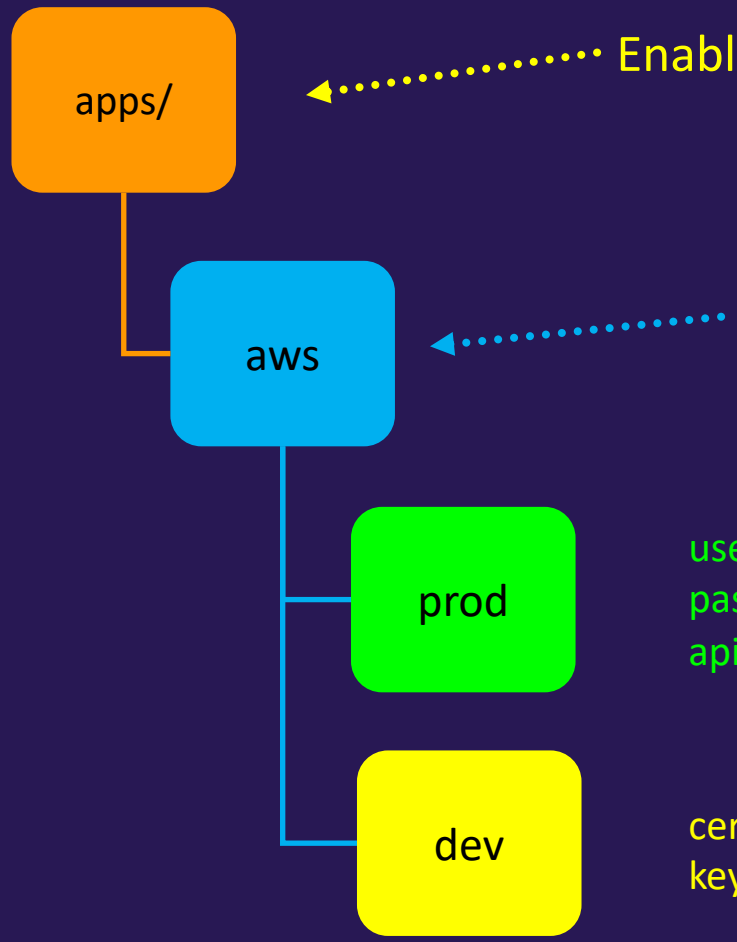
Key/Value Secrets Engine

Organize Data However It Makes Sense to Your Organization



Key/Value Secrets Engine

Let's Take a Look at Paths in the KV Store



Enabled a KV secrets engine at apps/

Just part of a path – currently no secrets stored here but you could if you wanted

user=dbadmin
password=P@ssw0rd
api=b93md83mdmapw

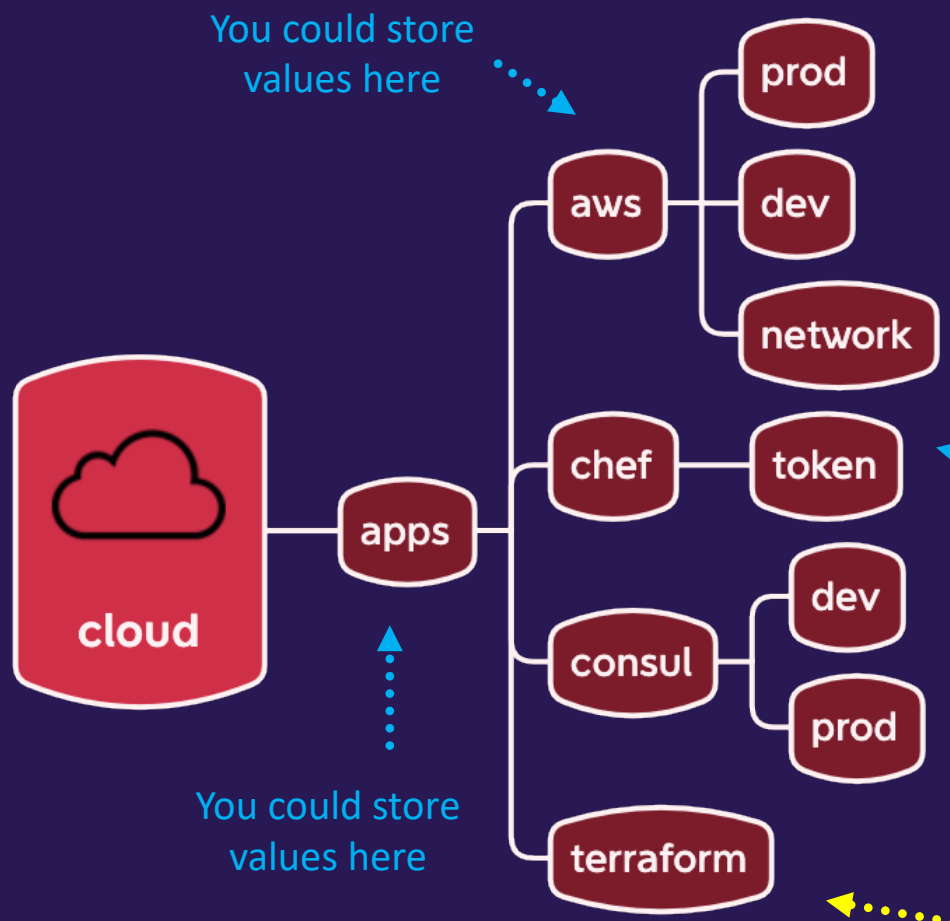
path = apps/aws/prod

cert=---BEGIN CERTI…
key=---BEGIN PRIVA…

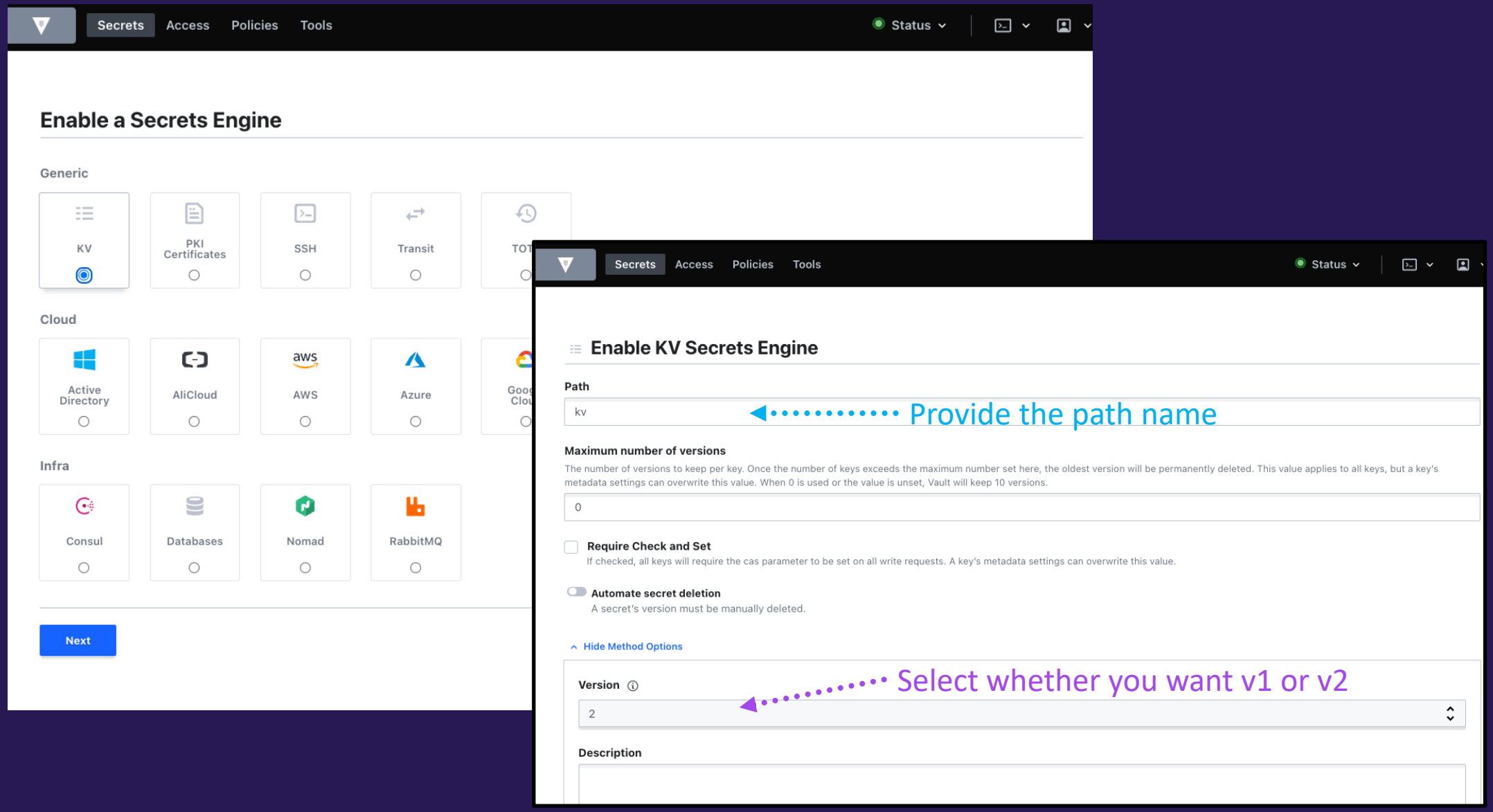
path = apps/aws/dev

Key/Value Secrets Engine

Store Data Wherever It Makes Sense to You



Enable Key/Value Secrets Engine



Enable Key/Value Secrets Engine

Enabling KV v1 Secrets Engine

Terminal

```
$ vault secrets enable kv
Success! Enabled the kv secrets engine at: kv/


$ vault secrets enable -path=training kv
Success! Enabled the kv secrets engine at: training/


$ vault secrets list -detailed
Path          Plugin       Accessor             … Options
----          ------       --------               -------
cubbyhole/    cubbyhole    cubbyhole_ee5ae49    map[]
kv/           kv           kv_e8b99a3           map[]
training/     kv           kv_1d5e9cc1          map[]
```

Enable at default path

Enable at custom path

Empty map means it's a KV v1 secrets engine

Enable Key/Value Secrets Engine

Enabling KV v2 Secrets Engine

Terminal

```
$ vault secrets enable kv-v2
Success! Enabled the kv-v2 secrets engine at: kv-v2/


$ vault secrets enable -path=training -version=2 kv
Success! Enabled the kv-v2 secrets engine at: training/


$ vault secrets list -detailed
Path          Plugin       Accessor             … Options
----          ------       --------               -------
cubbyhole/    cubbyhole    cubbyhole_ee5ae49      map[]
kv-v2/        kv           kv_e8b99a3             map[version:2]
training/     kv           kv_1d5e9cc1            map[version:2]
```

Enable at default path

Another way to enable KV v2

Notice the `version:2` in map, indicating a KV v2

Upgrade KV v1 to KV v2

- You can upgrade a KV v1 secrets engine to a KV v2
- You can't undo this upgrade
- …and no….you can't go from KV v2 to KV v2

```
$ vault kv enable-versioning training/
Success! Tuned the secrets engine at: training/
```

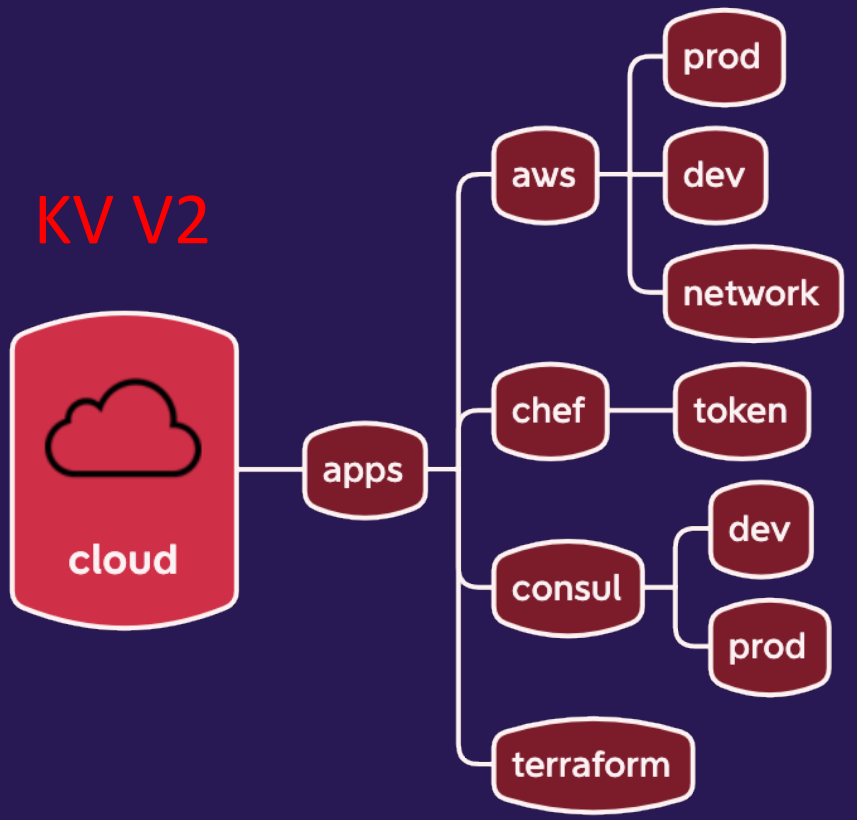
How is KV V2 Different?

- To support versioning, KV V2 adds metadata to our Key Value entries
- Used to determine creation date, the version of the secret, etc.

- Introduces two prefixes that must be accounted for when referencing secrets and/or metadata

  - `cloud/data` – data is where the actual K/V data is stored
  - `cloud/metadata` – the metadata prefix stores our metadata about a secret

How is KV V2 Different?



path = cloud/data/apps/aws/network

data/ prefix to read secrets

How is KV V2 Different?

Important Operational Concept about KV V2!

- The `data/` and `metadata/` prefix is required for API and when writing Vault policies

- It does NOT change the way you interact with the KV store when using the CLI

Don't worry! Examples in the next section!

How is KV V2 Different?

Versioning

Write Secret

pass=123
api=3jd93kd
user=admin

v1

Update Secret

pass=456
api=mfid02s
user=admin

v2

Delete Secret

v2

Rollback Secret

pass=456
api=mfid02s
user=admin

v3

How is KV V2 Different?

Versioning…Continued

Get Secret

pass=456
api=mfid02s
user=admin

v3

Delete Secret

v3

Undelete Secret

pass=456
api=mfid02s
user=admin

v3

Update Secret

pass=789
api=ckj983md
user=admin

v4

How is KV V2 Different?

Versioning…Continued

Get Secret

pass=789
api=ckj983md
user=admin

v4

Destroy Version 4

v4

Update Secret

course=vault
type=training

v5

Update Secret

pass=789
api=ckj983md
user=admin

v6

Working with KV using the CLI

Use the `vault kv` command

- `put` - write data to the KV
- `get` - read data from the KV
- `delete` - delete data from the KV
- `list` - list data within the KV (paths)

- `undelete` - undelete version of secret
- `destroy` - permanently destroy data
- `patch` - add specific key in the KV
- `rollback` - recover old data in the KV

Only available for KV V2

Writing Data to the KV Store

Compare KV Version 1 and Version 2

KV Version 1

```
$ vault kv put kv/app/db pass=123
Success! Data written to: kv/app/db
```

KV Version 2

```
$ vault kv put kv/app/db pass=123

Key                 Value
---                 -----
creation_time       2022-12-15T04:35:56.395821Z
deletion_time       n/a
destroyed           false
version             1
```

The CLI command is the same, but we get different output behavior

Writing Data to the KV Store

Command Line Interface (CLI)

```
vault kv put kv/app/db pass=123
```

- `kv` — Command when working with the KV Store
- `put` — Sub-command
- `kv/app/db` — Path where you want to store the KV pair
- `pass=123` — Data to store – entered as KV pairs

Writing Data to the KV Store

What if I have a bunch of key/value pairs?

```
$ vault kv put kv/app/db pass=123 user=admin api=a8ee4b50cce124
Success! Data written to: kv/app/db

$ vault kv put kv/app/db @secrets.json
Success! Data written to: kv/app/db
```

```
{
   "pass": "123",
   "user": "admin",
   "api": "a8ee4b50cce124"
}
```

Writing Secrets to the KV Store

Important Things to Remember

- Writing a new secret will replace the old value

Terminal

```
$ vault kv get kv/app/db
======= Metadata =======
Key                 Value
---                 -----
created_time        2022-02-21T14:03:02.062236Z
version             3
<output abbreviated>

==== Data ====
Key      Value
---      -----
pass     123
user     admin
api      a8ee4b50cce124
```

Maybe I want to update the value of api key but nothing else

Writing Secrets to the KV Store

Important Things to Remember

What will happen if I run the following command?

Terminal

```
$ vault kv put kv/app/db api=39cms1204mfi2m
```

Writing Secrets to the KV Store

Important Things to Remember

Terminal

```
$ vault kv put kv/app/db api=39cms1204mfi2m

Key                 Value
---                 -----
created_time        2022-12-21T14:40:26.886255Z
custom_metadata     <nil>
deletion_time       n/a
destroyed           false
version             2
```

Writing Secrets to the KV Store

Important Things to Remember

Terminal

```
$ vault kv get kv/app/db

======= Metadata =======
Key                 Value
---                 -----
created_time        2022-12-21T14:40:26.886255Z
custom_metadata     <nil>
deletion_time       n/a
destroyed           false
version             2

=== Data ===
Key    Value
---    -----
api    39cms1204mfi2m
```

A write is NOT a merge

Writing Secrets to the KV Store

All you wanted to do was update the `api`
but now you've lost your data?

What can we do?

Writing Secrets to the KV Store

Recover Your Data using Rollback

Terminal

```
$ vault kv rollback -version=1 kv/app/db

Key                 Value
---                 -----
created_time        2022-12-21T14:49:23.746331Z
custom_metadata     <nil>
deletion_time       n/a
destroyed           false
version             3
```

Writing Secrets to the KV Store

Recover Your Data using Rollback

Terminal

```
$ vault kv get kv/app/db
======= Metadata =======
Key                 Value
---                 -----
created_time        2022-02-21T14:03:02.062236Z
<output abbreviated>

==== Data ====
Key      Value
---      -----
pass     123
user     admin
api      a8ee4b50cce124
```

Writing Secrets to the KV Store

Patch Your Data

Terminal

```
$ vault kv patch kv/app/db user=bryan
======= Metadata =======
Key                 Value
---                 -----
created_time        2022-12-22T17:57:35.157363Z
destroyed           false
version             4

$ vault kv get kv/app/db
....
==== Data ====
Key       Value
---       -----
pass      123
user      bryan
api       a8ee4b50cce124
```

Outputs abbreviated

Reading Secrets from the KV Store

Compare KV Version 1 and Version 2

KV Version 1

```
$ vault kv get kv/app/db
========Data==========
Key         Value
---         -----
pass        123
user        admin
api         a8ee4b50cce124
```

KV Version 2

```
$ vault kv get kv/app/db
=========Metadata===========
Key                 Value
---                 -----
creation_time       2022-12-15T04:35:56.395821Z
deletion_time       n/a
destroyed           false
version             1

===========Data=============
Key         Value
---         -----
pass        123
user        admin
api         a8ee4b50cce124
```

The CLI command is the same, but we get different output behavior

Reading Secrets from the KV Store

Output

KV Version 2

```
$ vault kv get kv/app/db
=========Metadata==========
Key                 Value
---                 -----
creation_time       2022-12-15T04:35:56.395821Z
deletion_time       n/a
destroyed           false
version             1

==========Data=============
Key         Value
---         -----
pass        123
user        admin
api         a8ee4b50cce124
```

Default output type is `table`

Reading Secrets from the KV Store

Output

KV Version 2

```
$ vault kv get -format=json kv/app/db
{
  "request_id": "249fca06-a8ce-5617-d598-1c12384d4ac8",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": {
    "data": {
      "pass": "123",
      "user": "admin",
      "api": " a8ee4b50cce124",
    },
    "metadata": {
      "created_time": "2022-12-21T13:59:29.917893Z",
      "custom_metadata": null,
      "deletion_time": "",
      "destroyed": false,
      "version": 1
```

Output abbreviated

Change output format to `json`

Useful for creating machine-readable outputs

Reading Secrets from the KV Store

Important Things to Remember

- A regular `read` request will return the latest version of the secret
- If the latest version of the secret has been deleted (KV V2), it will return the related metadata but no data (secrets)
- You can read a previous version of a secret (if one exists) by adding the `-version=x` flag to the request

Terminal

```
$ vault kv get -version=3 kv/app/db
```

Deleting Secrets from the KV Store

- A `delete` on KV V1 is a delete – the data is destroyed
  - You'd have to restore Vault/Consul to retrieve the old data
- A `delete` on KV V2 is a soft delete – data is not destroyed
  - Data can be restored with a `undelete/rollback` action
- A `destroy` (only KV V2) is a permanent action – destroyed on disk
  - Cannot be restored except for a Vault/Consul restore action

Deleting Secrets from the KV Store

KV V1 - Read Output after Deleting the Latest Version

Terminal

```
$ vault kv delete secret/app/database
Success! Data deleted (if it existed) at: secret/app/database

$ vault kv get secret/app/database
No value found at secret/app/database
```

No values exist if you delete a secret at a path for KV V1

Deleting Secrets from the KV Store

KV V2 - Read Output after Deleting the Latest Version

Terminal

```
$ vault kv delete secret/app/web
Success! Data deleted (if it existed) at: secret/app/web

$ vault kv get secret/app/web
======= Metadata =======
Key                Value
---                -----
created_time       2022-12-15T17:41:41.13052Z
custom_metadata    <nil>
deletion_time      2022-12-15T17:42:03.369955Z
destroyed          false
version            3
```

Only returned metadata but no data (since it was deleted)

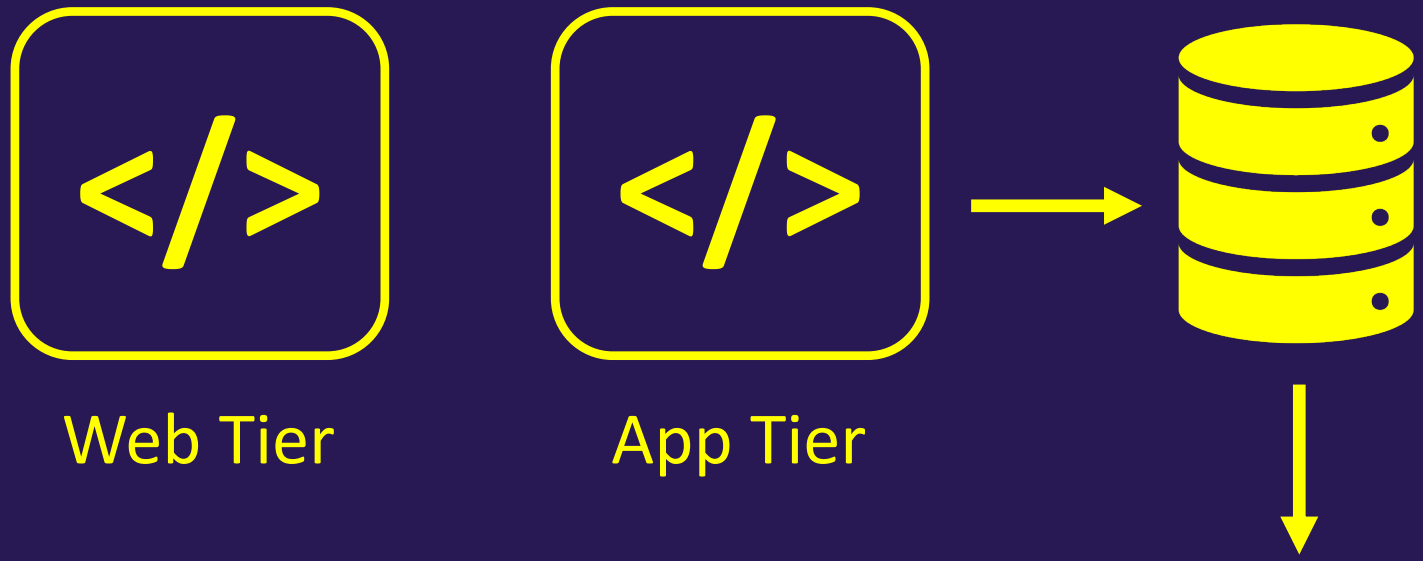
Destroy Secrets from the KV Store

KV V2 - Read Output after Destroying the Latest Version

Terminal

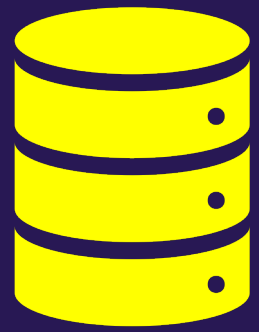
```
$ vault kv destroy -versions=3 secret/app/web
Success! Success! Data written to: secret/app/web


$ vault kv get secret/app/web
======= Metadata =======
Key                 Value
---                 -----
created_time        2022-12-21T14:49:23.746331Z
custom_metadata     <nil>
deletion_time       n/a
destroyed           true
version             3
```

Problems with Encryption in the Enterprise



Problems with Encryption in the Enterprise

Options to Encrypt our Data



1 Rely on Database Capabilities

2 Use External Solution or Library

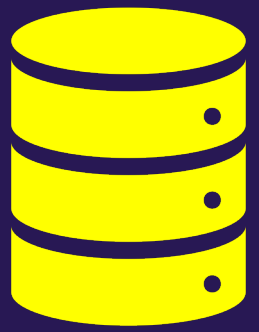
SDK

API

dj39fm409skj20skjsoij20dknjad2
20dj2094kjndsoi320djsjl2083jdjln
dlj30932isnmsdmn2903jndjn2o2
d vna;2-284nf n.s;dluifh3058bnls

Problems with Encryption in the Enterprise

Choosing a Database Platform for Our App



dj39fm409skj20skjsoij20dknjad2
20dj2094kjndsoi320djsjl2083jdjln
dlj30932isnmsdmn2903jndjn2o2
d vna;2-284nf n.s;dluifh3058bnls

Ideal Database: Cassandra

but.....maybe it doesn't support the type/level of encryption we need

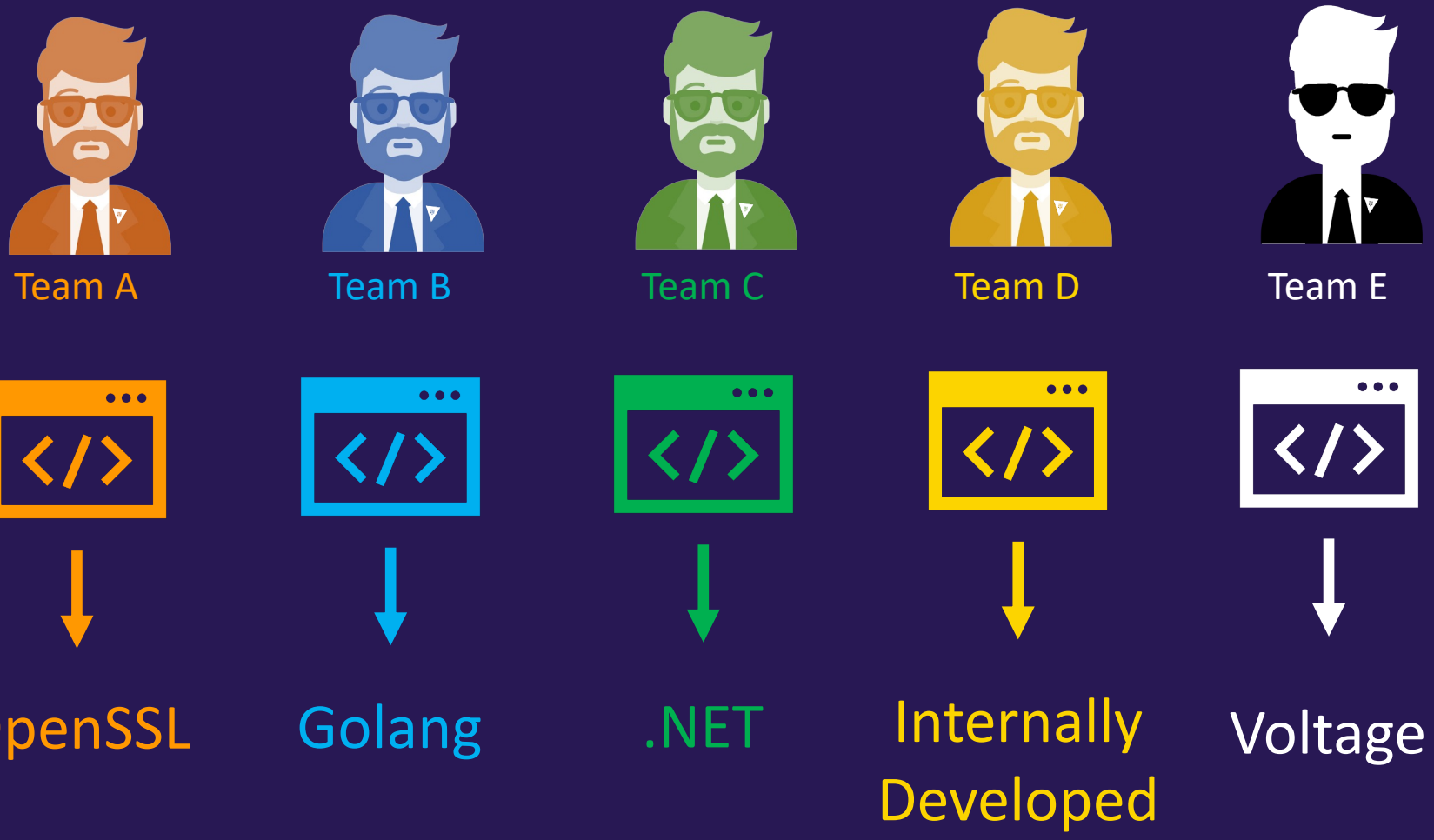
Required Database: MSSQL

Only MSSQL features meet the requirements to encrypt our data

NOT IDEAL

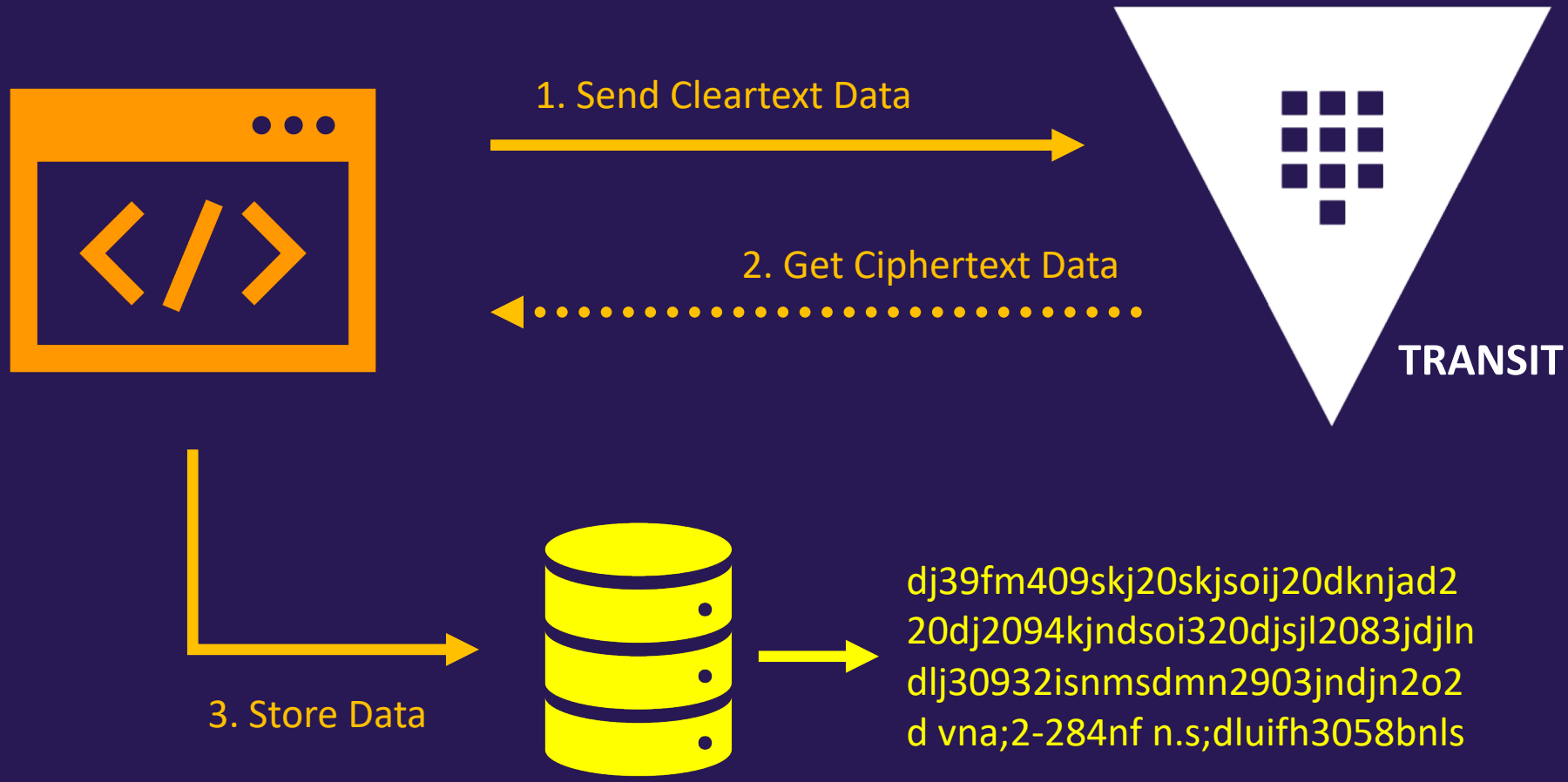
Problems with Encryption in the Enterprise

Putting the Responsibility on the Developer



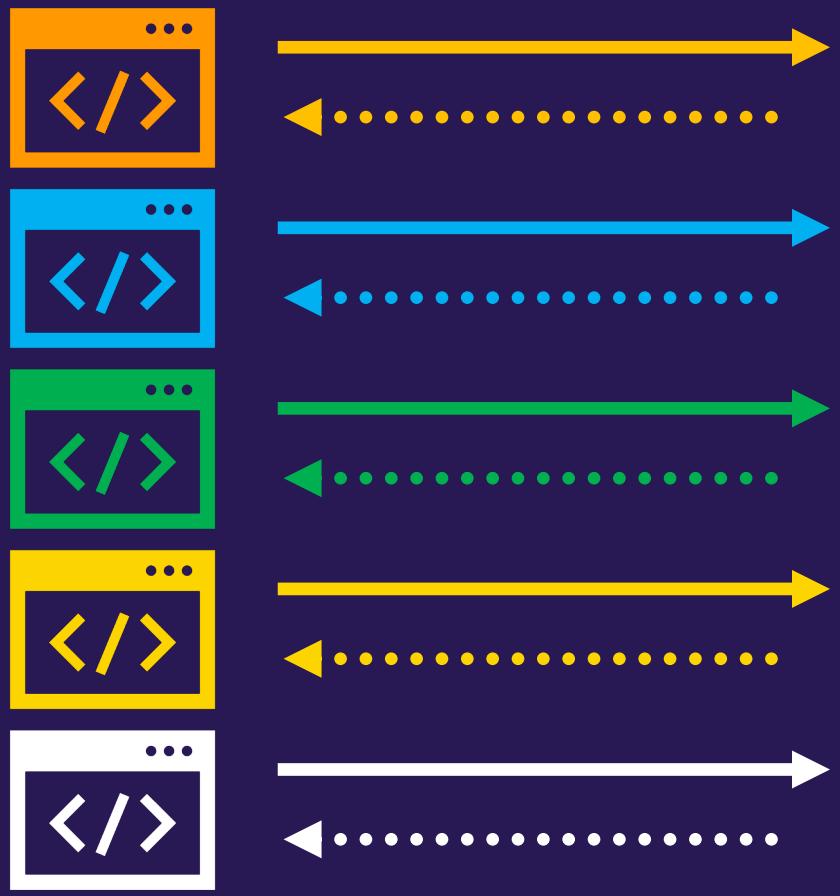
Solution?

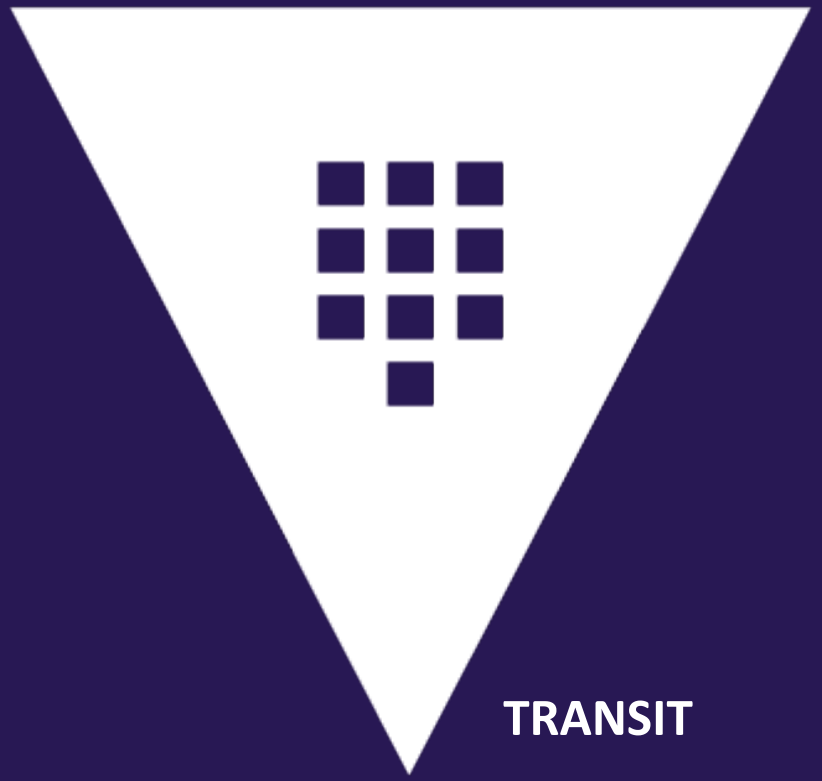
Use Vault's Transit Secrets Engine



Solution?

Centralize the Organization's Encryption Needs





Intro to Transit Secrets Engine

Transit secrets engine provides functions for encrypting/decrypting data

- Enables organizations to outsource/centralize encryption to Vault

Applications can send cleartext data to Vault for encryption

- Vault encrypts using the specified key and returns ciphertext to the app
- The application NEVER has access to the encryption key (stored in Vault)
- Decouples storage from encryption and access control

Transit secrets engine DOES NOT STORE the encrypted data

Intro to Transit Secrets Engine

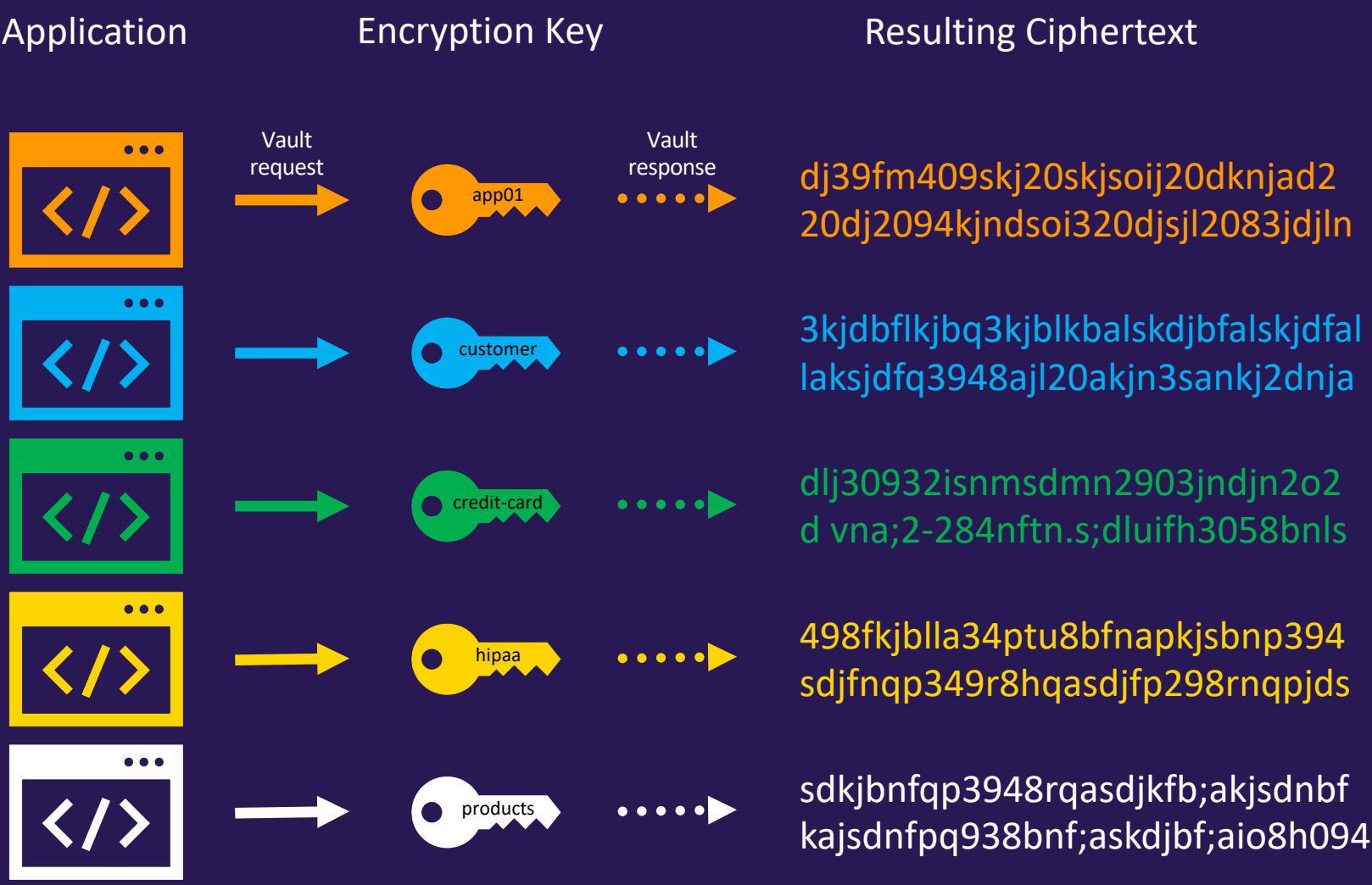
Encryption keys are created and stored in Vault to process data

- Each application can have its own encryption key (or more!)
- Apps must have permission to use the key for encryption/decryption operations, which is bound by the policy attached to its token

Keys can be easily rotated as often as needed

- Keys are stored on keyring 🔑
- Can limit what version(s) of keys can be used for decryption
- You can create, rotate, delete, and export a key (need permissions)
- Easily rewrap ciphertext with a newer version of a key

Intro to Transit Secrets Engine



Encryption Key Types

Key Type	Description
aes128-gcm96	AES-GCM with a 128-bit AES key and a 96-bit nonce; supports encryption, decryption, key derivation, and convergent encryption
aes256-gcm96 Default	AES-GCM with a 256-bit AES key and a 96-bit nonce; supports encryption, decryption, key derivation, and convergent encryption (default)
chacha20-poly1305	ChaCha20-Poly1305 with a 256-bit key; supports encryption, decryption, key derivation, and convergent encryption
ed25519	Ed25519; supports signing, signature verification, and key derivation
ecdsa-p256	ECDSA using curve P-256; supports signing and signature verification
ecdsa-p384	ECDSA using curve P-384; supports signing and signature verification
ecdsa-p521	ECDSA using curve P-521; supports signing and signature verification
rsa-2048	2048-bit RSA key; supports encryption, decryption, signing, and signature verification
rsa-3072	3072-bit RSA key; supports encryption, decryption, signing, and signature verification
rsa-4096	4096-bit RSA key; supports encryption, decryption, signing, and signature verification

Intro to Transit Secrets Engine

Vault also supports convergent encryption mode

- Means that every time you encrypt the same data, you'll get the same ciphertext back
- This enables you to have searchable ciphertext

Encrypting Data

- All plaintext data must be base64-encoded
- This is because Vault doesn't require that the plaintext is "text" only. It could be a file such as a PDF or image
- But…please understand that base64-encoding is NOT encryption

Working with the Transit Secrets Engine

Enable the Secrets Engine

Before you can use the Transit secrets engine to encrypt data, it must first be enabled

- Can use the default path of `transit` or enable on another path

```
$ vault secrets enable transit
Success! Enabled the transit secrets engine at: transit/
```

Working with the Transit Secrets Engine

Create an Encryption Key

The next step is to create one or many encryption keys used to encrypt/decrypt data

```
$ vault write -f transit/keys/vault_training
Success! Data written to: transit/keys/vault_training
```

Note: `-f` is shorthand for "`force`" and is needed on some Vault commands

Working with the Transit Secrets Engine

Encrypt Data

```
$ vault write -f transit/keys/training
Success! Data written to: transit/keys/training


$ vault write -f transit/keys/training_rsa type="rsa-4096"
Success! Data written to: transit/keys/training_rsa
```

Custom Key Type

Working with the Transit Secrets Engine

Encrypt Data

Pass the cleartext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/encrypt/training \
    plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")

Key            Value
---            -----
ciphertext     vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEahal5LhDPSoN6HkTOhwn79DCwt0mct1ttLokqikAr0PAopzm2jQAKJg=
key_version    1
```

Working with the Transit Secrets Engine

Encrypt Data

Pass the cleartext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/encrypt/training \
    plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")

Key            Value
---            -----
ciphertext     vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEahal5LhDPSoN6HkTOhwn79DCwt0mct1ttLokqikAr0PAopzm2jQAKJg=
key_version    1
```

Working with the Transit Secrets Engine

Break Down The Command Used For Encryption

```
vault write transit/encrypt/training
```

Sub-command

Path where the Transit secrets engine was mounted (enabled)

The desired action

The encryption key you want to use

Working with the Transit Secrets Engine

Break Down The Parameter Used For Encryption

```
plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")
```

Required Parameter for Encryption operation (what are we going to encrypt?)

Base64 encode the string "Getting Started with HashiCorp Vault"

If your data is already base64 encoded, you can just pass it as is….you don't need to encode it again

Working with the Transit Secrets Engine

Let's Look at the Output After Encrypting Data

```
Key             Value
---             -----
ciphertext      vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQ.....
key_version     1
```

This is the data that you would store in a database, file store, or anywhere so your application or organization can read later

Output abbreviated

The Key Version is built right into the ciphertext

Working with the Transit Secrets Engine

Decrypt Data

Pass the ciphertext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/decrypt/training \
   ciphertext="vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEahal5LhDPSoN6HkTO……."

Key             Value
---             -----
plaintext       R2V0dGluZyBTdGFydGVkIHdpdGggSGFzaGlDb3JwIFZhdWx0Cg==
```

Rotating Encryption Keys

Key Rotation

Transit allows for a simplified key rotation process

- keys can be rotated manually or by an <external> automated process

Vault maintains a versioned keyring

- All versions of the encryption key are stored
- Vault admins can limit the `minimum key version` allowed to be used for decryption operations (older keys won't work)
- You can `rewrap` encrypted data (ciphertext) to use a newer version of the encryption key

Working with Encryption Keys

Rotate an Encryption Key

Pass the ciphertext data to Vault – specifying the action and desired encryption key to use

```
$ vault write -f transit/keys/training/rotate
Success! Data written to: transit/keys/training/rotate
```

Working with Encryption Keys

Rotate an Encryption Key

```
$ vault read transit/keys/training
Key                       Value
---                       -----
allow_plaintext_backup    false
deletion_allowed          false
derived                   false
exportable                false
keys                      map[1:1647960245 2:1647960257 3:1647961177]
latest_version            3
min_available_version     0
min_decryption_version    1
min_encryption_version    0
name                      training
supports_decryption       true
supports_derivation       true
supports_encryption       true
supports_signing          false
type                      aes256-gcm96
```

keyring

Working with Encryption Keys

Key Configuration

- We can limit what version of the key can be used to decrypt data
  - Maybe we have old data that we have converted and don't want anybody to be able to decrypt it
  - This is configured using the `minimum key version` configuration
  - It can be configured for each encryption key (not key version)

Working with Encryption Keys

Key Configuration

Minimum
Key Version = 1

v1

vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f
dja;siurn4398ebjkbalks
jdbnfp3948fbhdjkbali4
8rh4938ebaskjbq9384
hbfjdblaw948rhfjasdu

v2

vault:v2:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23
8rhqeojdfnvq3948rhq
39ruincsajkvnpq9384h
qpjisnv;aksjdnf9q384h
fjdsnlaisjnfq9384hr938

v3

vault:v3:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfpq93u4fbalskjdnb
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf
pq938rfhpqekjfdbvlaks
jdnvq3948fhgfdjabnlsk
rjbnre3pq9uesdkjnrpfi
unaklns394njksalskern

v4

vault:v4:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl
akjbpaieurhq83urbgla
kjbvpqieurbgpqijfbvlaif
uhpq0w9eurbfpakjdbf
vlakdjfbngqiasdf34dsfa

Working with Encryption Keys

Key Configuration

Minimum
Key Version = 4

v1

vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f
dja;siurn4398ebjkbalks
jdbnfp3948fbhdjkbali4
8rh4938ebaskjbq9384
hbfjdblaw948rhfjasdu

v2

vault:v2:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23
8rhqeojdfnvq3948rhq
39ruincsajkvnpq9384h
qpjisnv;aksjdnf9q384h
fjdsnlaisjnfq9384hr938

v3

vault:v3:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfpq93u4fbalskjdnb
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf
pq938rfhpqekjfdbvlaks
jdnvq3948fhgfdjabnlsk
rjbnre3pq9uesdkjnrpfi
unaklns394njksalskern

v4

vault:v4:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl
akjbpaieurhq83urbgla
kjbvpqieurbgpqijfbvlaif
uhpq0w9eurbfpakjdbf
vlakdjfbngqiasdf34dsfa

Working with Encryption Keys

Setting the Minimum Decryption Version

```
$ vault write transit/keys/training/config \
    min_decryption_version=4

Success! Data written to: transit/keys/training/config
```

Working with Encryption Keys

Rotate an Encryption Key

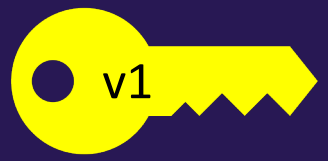
```
$ vault read transit/keys/training
Key                       Value
---                       -----
allow_plaintext_backup    false
deletion_allowed          false
derived                   false
exportable                false
keys                      map[4:1647962305]
latest_version            4
min_available_version     0
min_decryption_version    4
min_encryption_version    0
name                      training
supports_decryption       true
supports_derivation       true
supports_encryption       true
supports_signing          false
type                      aes256-gcm96
```

Only the key(s) equal or greater than the minimum key version are available

Rewrapping Ciphertext

Rewrapping Ciphertext

Rewrapping Data



vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f

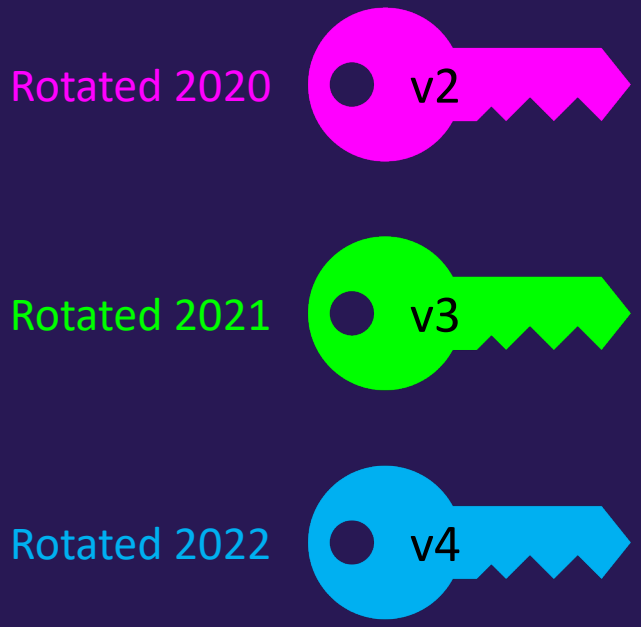
vault:v1:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfpq93u4fbalskjdnb
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf

vault:v1:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23

vault:v1:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl

Data Encrypted in 2019

....but our key has been rotated 3 times since....



How can we upgrade our encrypted data to be encrypted by the latest version of the key?

Rewrapping Ciphertext

Rewrapping Data

```
$ vault write transit/rewrap/training \
    ciphertext="vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEahal5LhDPSoN6
    HkTOhwn79DCwt0mct1ttLokqikAr0PAopzm2jQAKJg=2/QGPTMnzKPlw4cCPGTbkzE
    PlX5OyPkLIgX+erFWdUXKkKUIEbb6D2Gm5ZjTaola314LsVkbLF5G1RkBTACtskk="

Key             Value
---             -----
ciphertext      vault:v4:RFzp1kMpjtUIiS+6qxrNjIJEdPqCepFUa2ivr70……
key_version     4
```

The data was never available in plaintext when rewrapping the data with the latest version of the key

END OF SECTION