Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Interpret Vault Identity Entities and Groups

Vault Entities

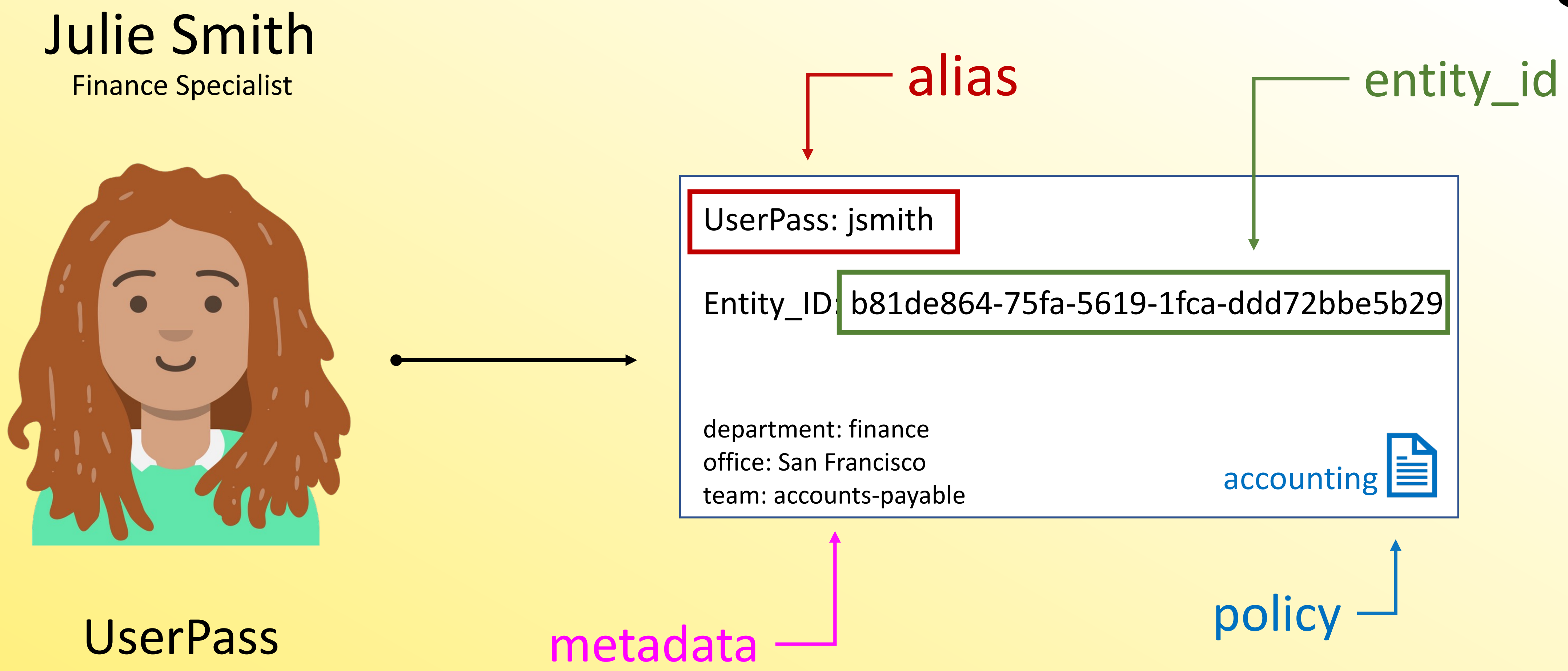
Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

- Vault creates an entity and attaches an alias to it if a corresponding entity doesn't already exist.
  - This is done using the Identity secrets engine, which manages internal identities that are recognized by Vault

- An entity is a representation of a single person or system used to log into Vault. Each has a unique value. Each entity is made up of zero or more aliases

- Alias is a combination of the auth method plus some identification. It is a mapping between an entity and auth method(s)

Vault Entities



Julie Smith

Finance Specialist

UserPass

alias

entity_id

UserPass: jsmith

Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

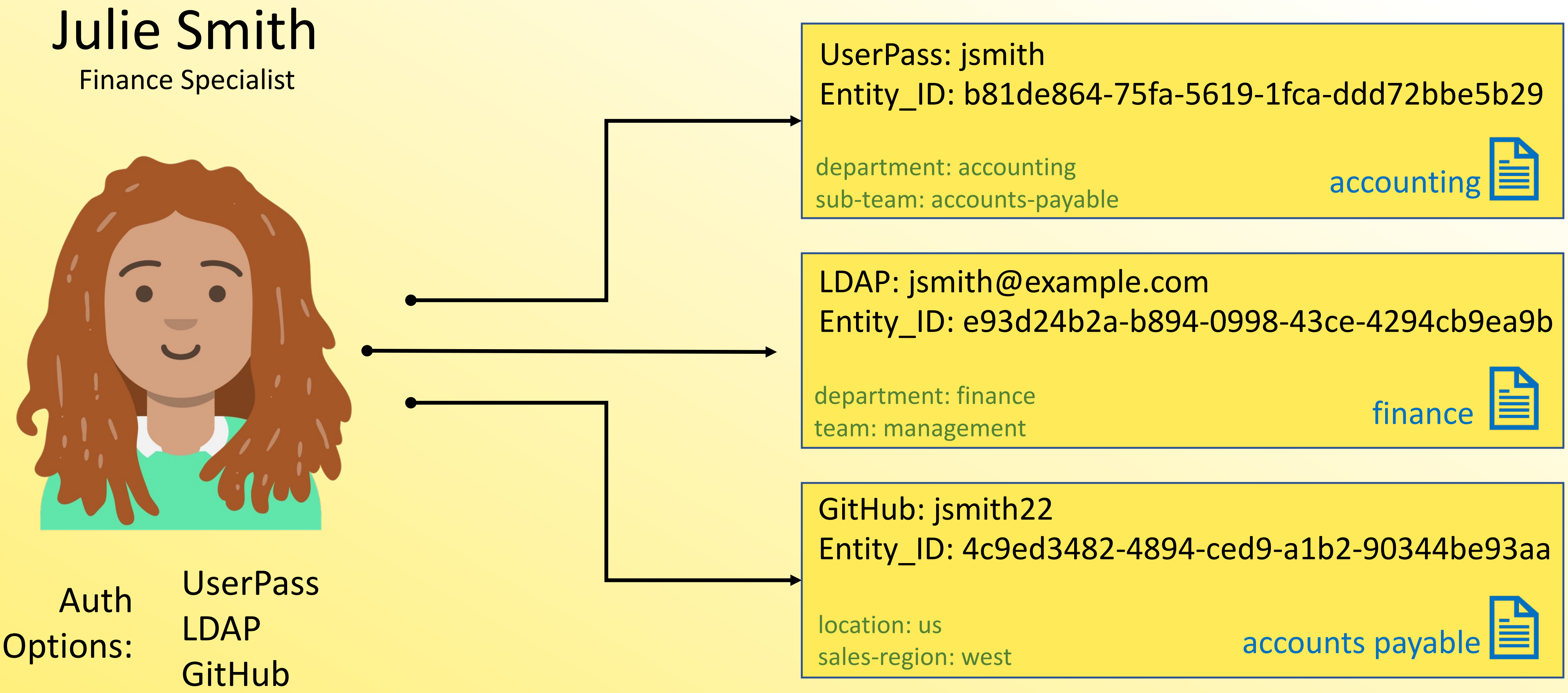
department: finance
office: San Francisco
team: accounts-payable

accounting

metadata

policy

Vault Entities



Julie Smith

Finance Specialist

Auth Options: UserPass
LDAP
GitHub

UserPass: jsmith
Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

department: accounting
sub-team: accounts-payable

accounting

LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b

department: finance
team: management

finance

GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa

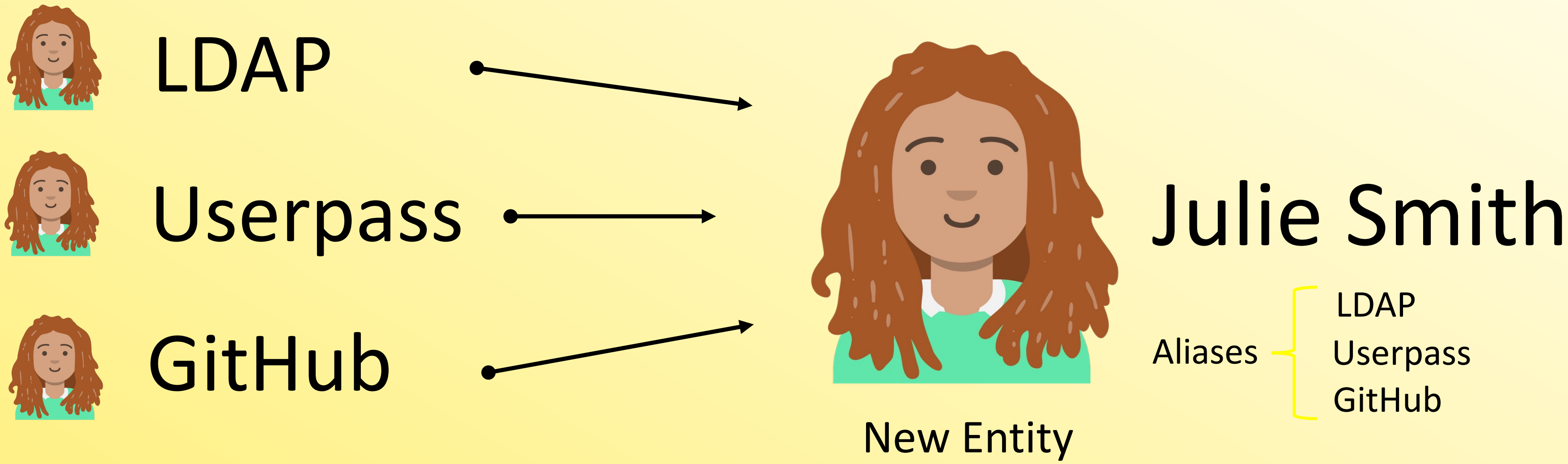
location: us
sales-region: west

accounts payable

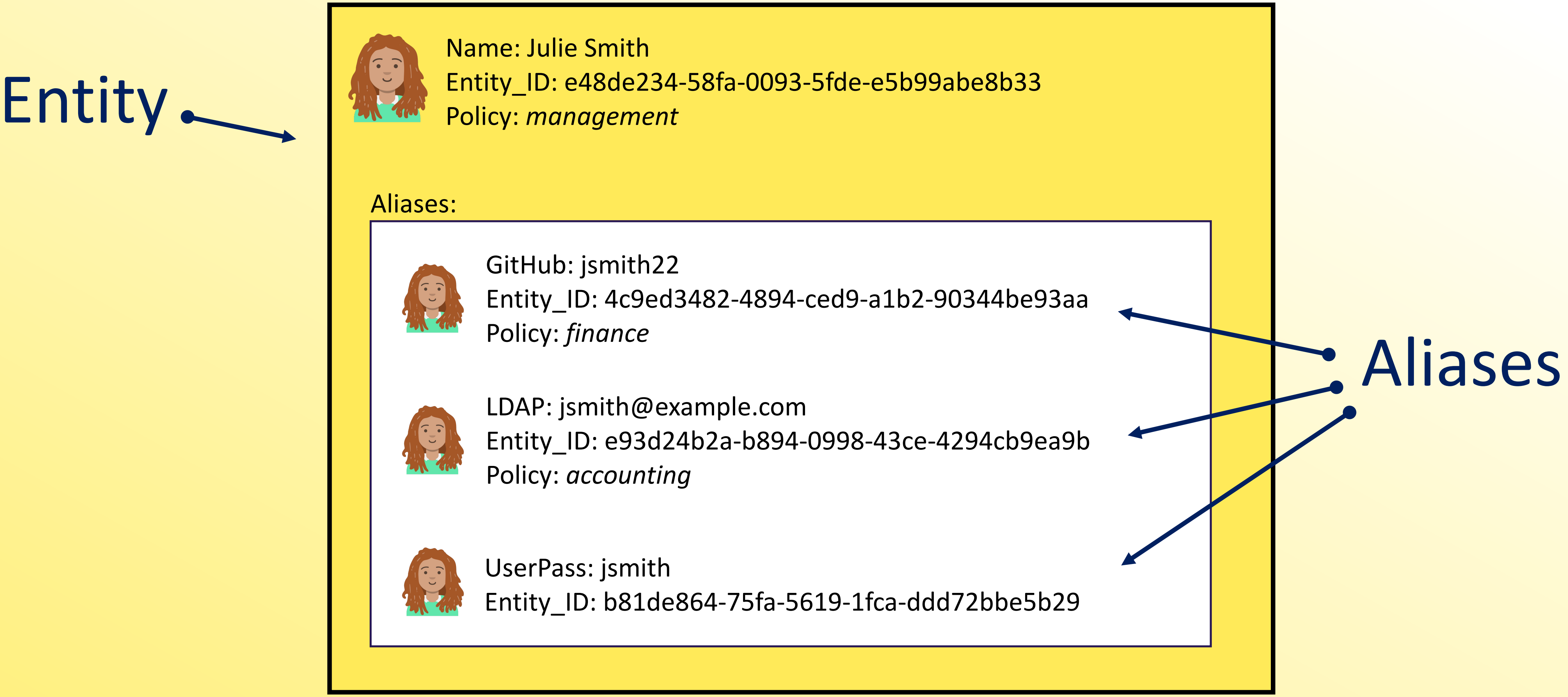
Vault Entities

- An entity can be manually created to map multiple entities for a single user to provide more efficient authorization management

- Any tokens that are created for the entity inherit the capabilities that are granted by alias(es).



Vault Entities



Entity

Name: Julie Smith
Entity_ID: e48de234-58fa-0093-5fde-e5b99abe8b33
Policy: *management*

Aliases:

GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *finance*

LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *accounting*

UserPass: jsmith
Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

Aliases

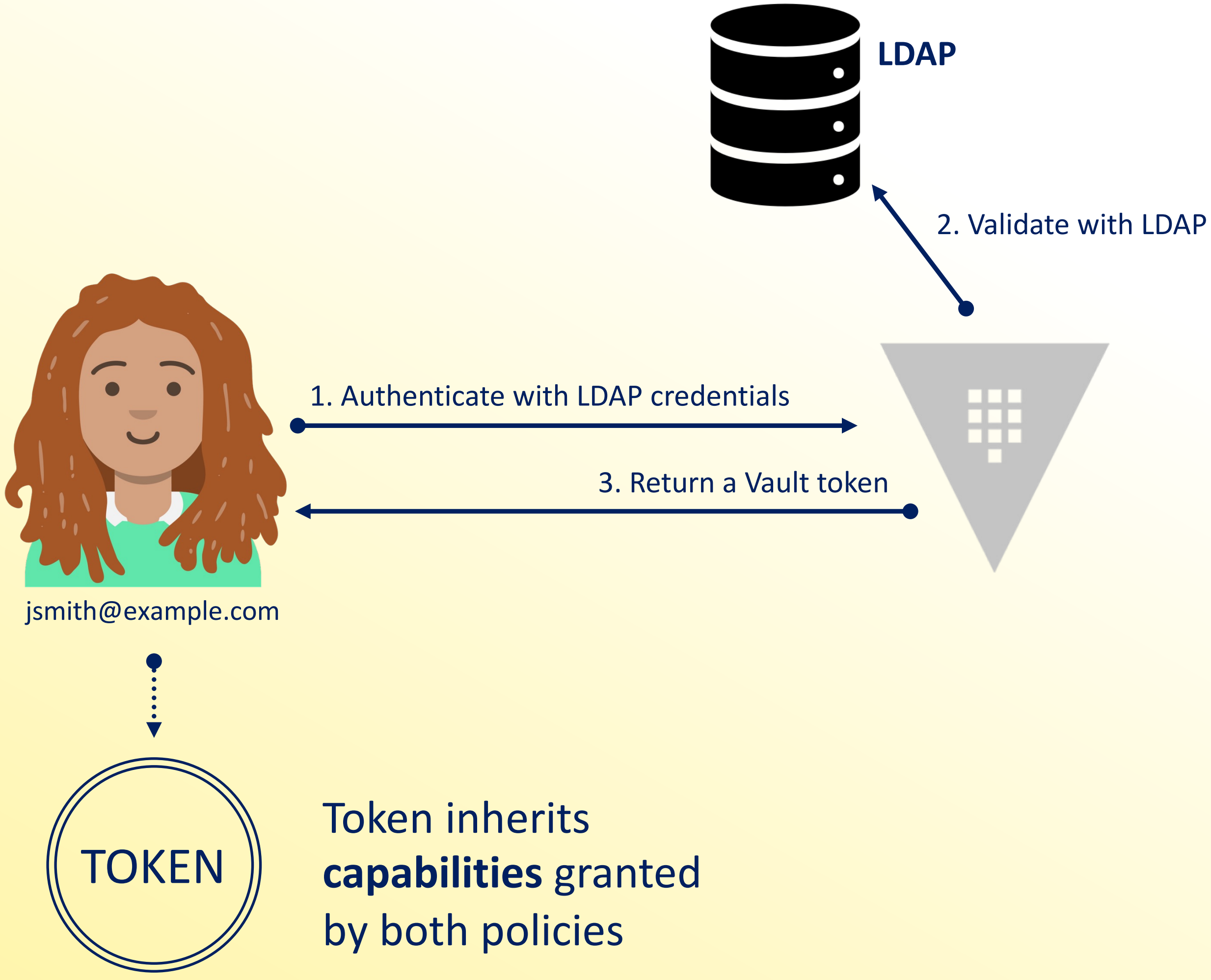
Vault Entities

Name: Julie Smith
Entity_ID: e48de234-58fa-0093-5fde-e5b99abe8b33
Policy: *management*

Aliases:

GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *finance*

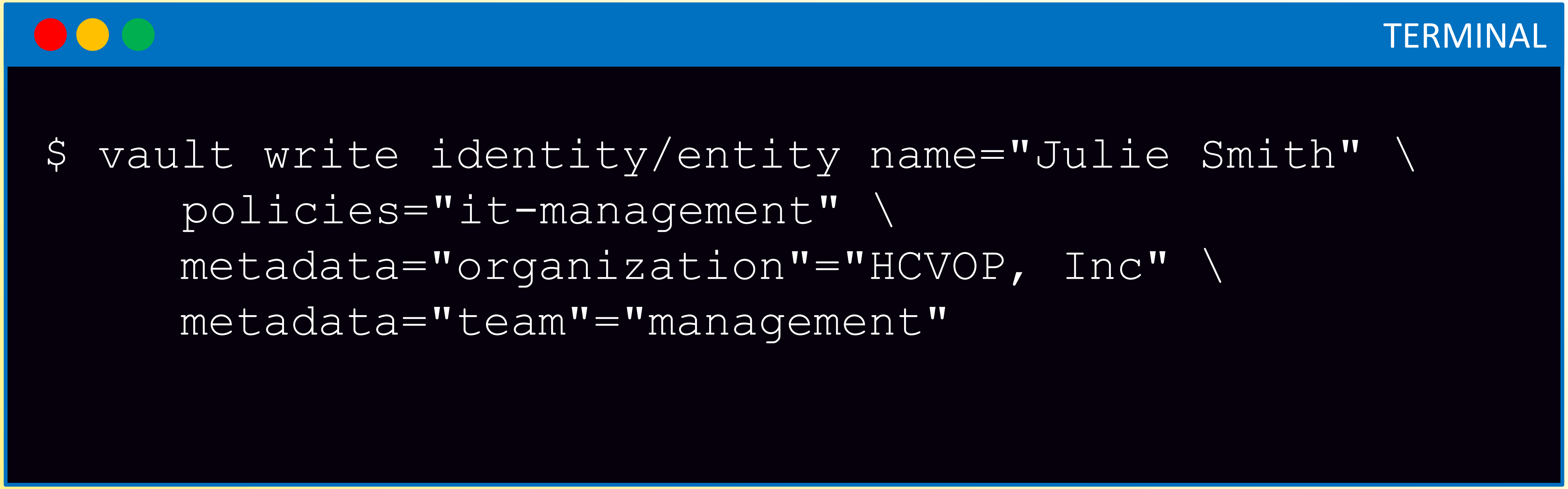
LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *accounting*



Token inherits **capabilities** granted by both policies

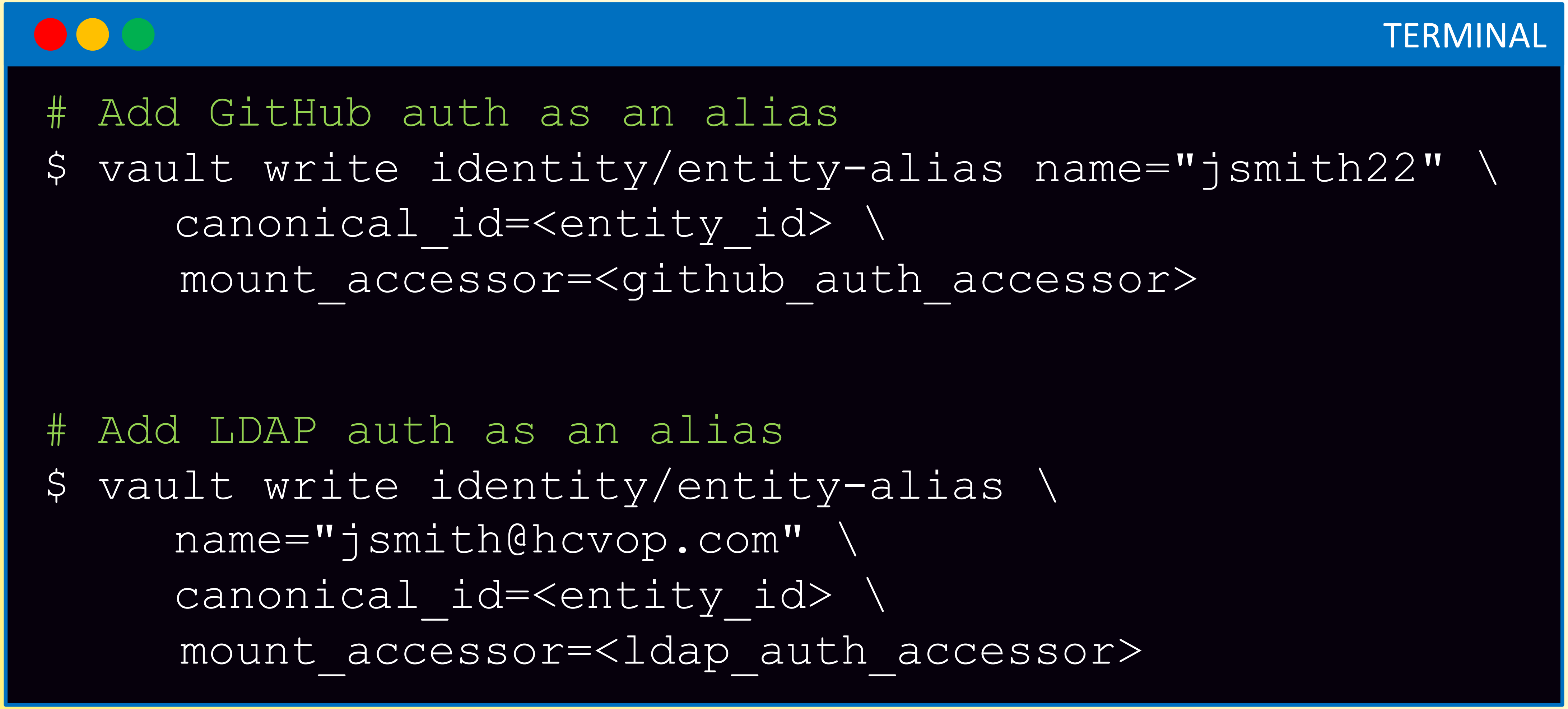
Policies
accounting
management

Create an Entity



```
$ vault write identity/entity name="Julie Smith" \
     policies="it-management" \
     metadata="organization"="HCVOP, Inc" \
     metadata="team"="management"
```

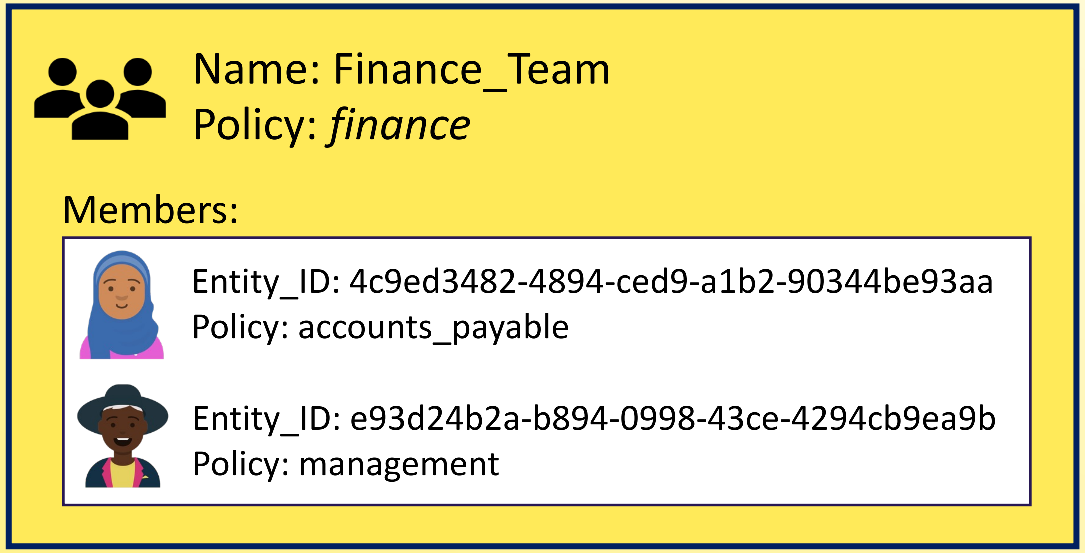
Add an Alias to an Entity



```
# Add GitHub auth as an alias
$ vault write identity/entity-alias name="jsmith22" \
     canonical_id=<entity_id> \
     mount_accessor=<github_auth_accessor>


# Add LDAP auth as an alias
$ vault write identity/entity-alias \
     name="jsmith@hcvop.com" \
     canonical_id=<entity_id> \
     mount_accessor=<ldap_auth_accessor>
```

Vault Groups

- A group can contain multiple entities as its members.
- A group can also have subgroups.
- Policies can be set on the group and the permissions will be granted to all members of the group.



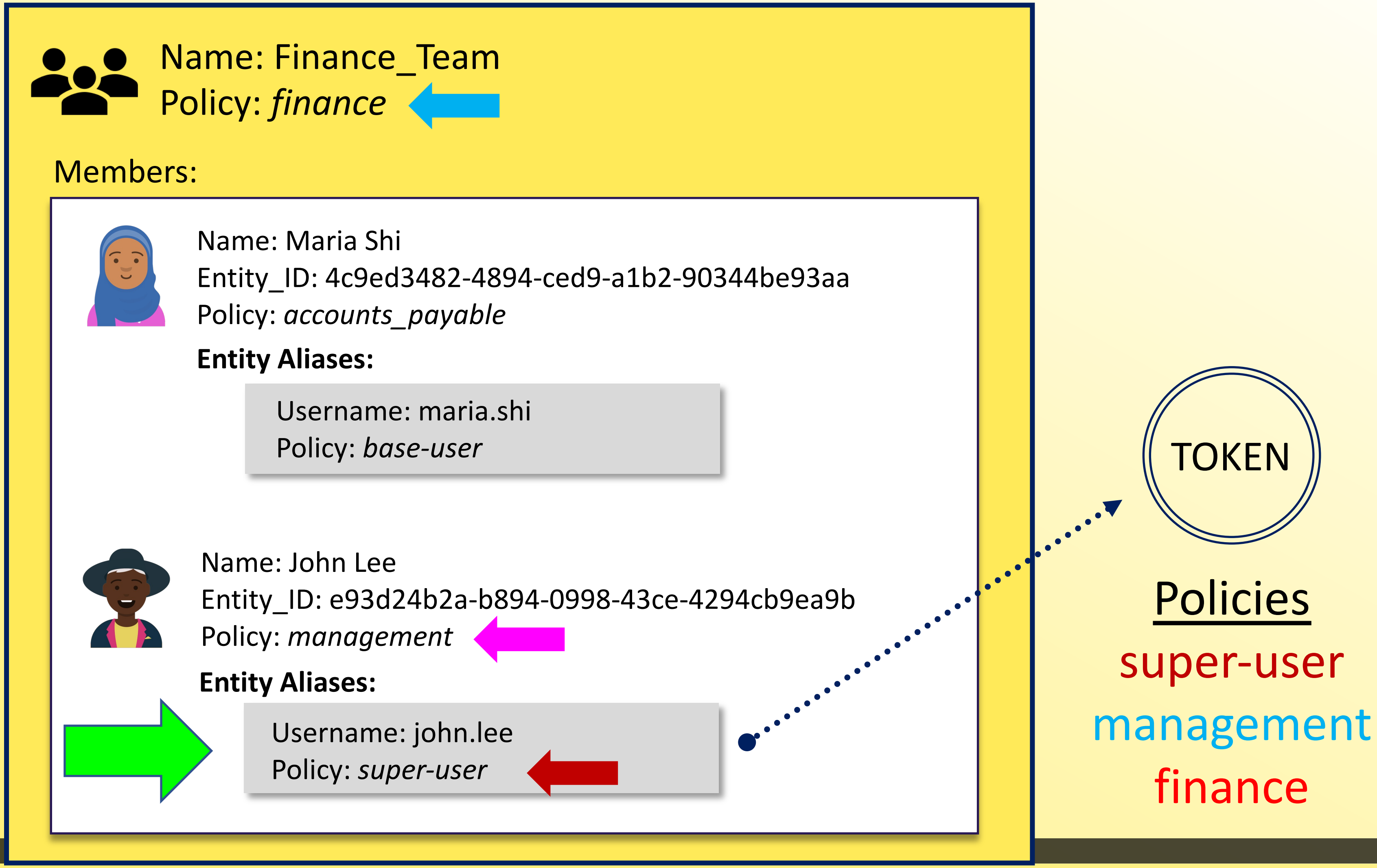
Name: Finance_Team
Policy: *finance*

Members:

Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: accounts_payable

Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: management

Vault Groups



Name: Finance_Team
Policy: *finance*

Members:

Name: Maria Shi
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *accounts_payable*

Entity Aliases:

Username: maria.shi
Policy: *base-user*

Name: John Lee
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *management*

Entity Aliases:

Username: john.lee
Policy: *super-user*

TOKEN

Policies
super-user
management
finance

Token inherits **capabilities** granted by alias, entity, and the group

Vault Groups

Internal Group

Groups created in Vault to group entities to propagate identical permissions

Created Manually

External Group

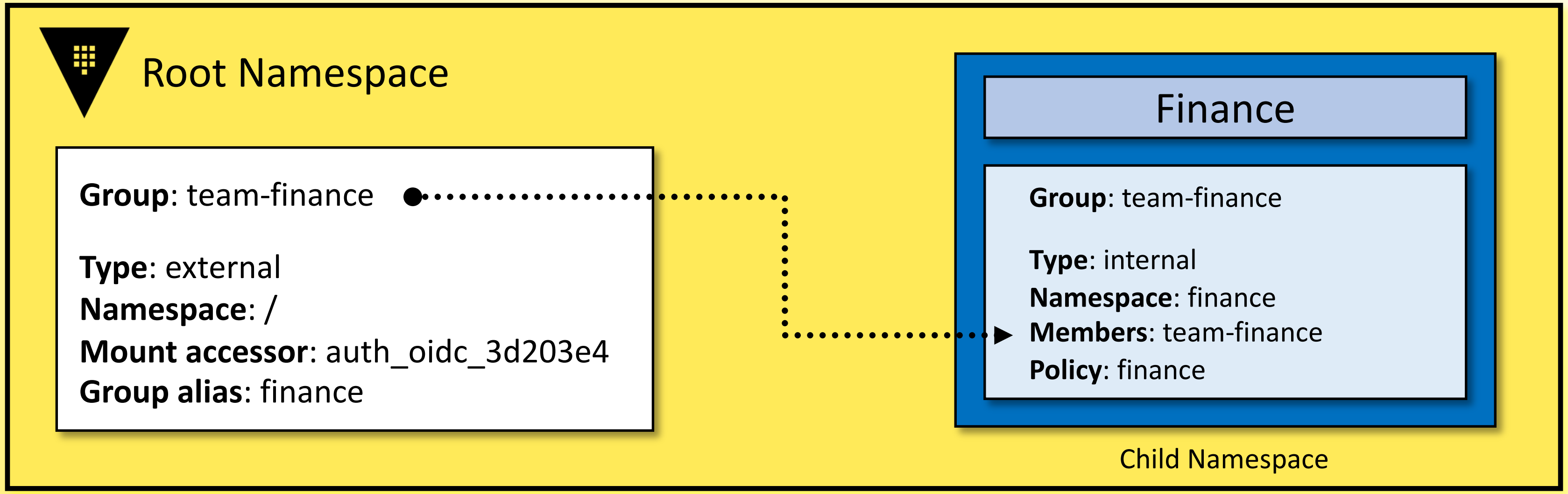
Groups which Vault infers and creates based on group associations coming from auth methods

Created Manually or Automatically

Vault Groups

Internal Groups

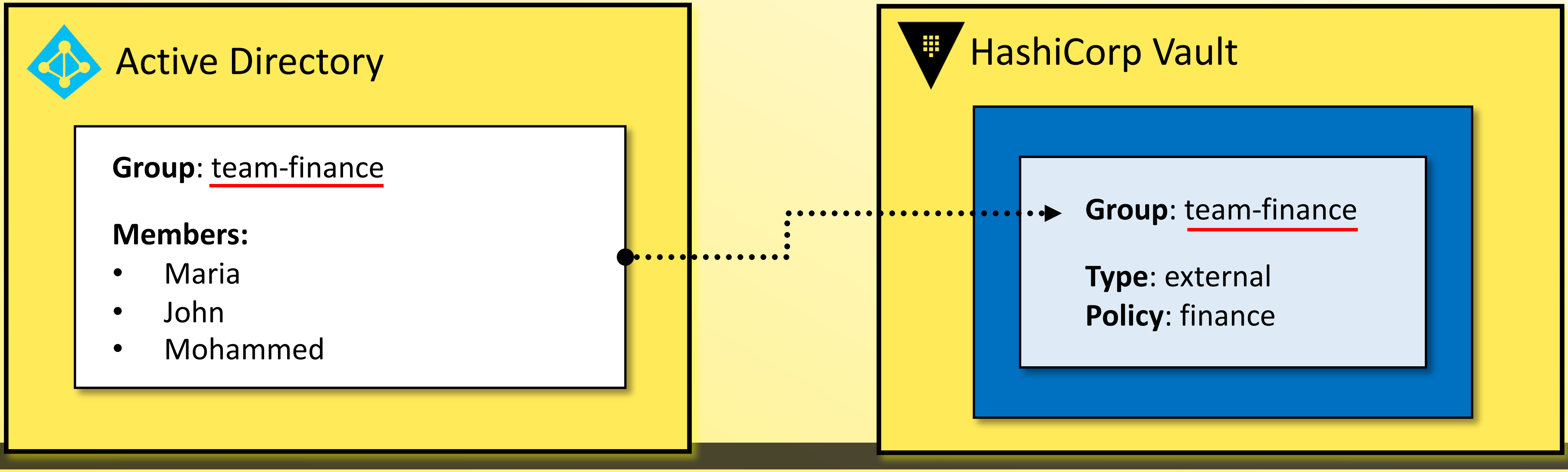
- Internal groups can be used to easily manage permissions for entities
- Frequently used when using Vault Namespaces to propagate permissions down to child namespaces
  - Helpful when you don't want to configure an identical auth method on every single namespace



Vault Groups

External Groups

- External groups are used to set permissions based on group membership from an external identity provider, such as LDAP, Okta, or OIDC provider.
- Allows you to set up once in Vault and continue manage permissions in the identity provider.
  - Note that the group name must match the group name in your identity provider



Active Directory

Group: team-finance

Members:

- Maria
- John
- Mohammed

HashiCorp Vault

Group: team-finance

Type: external
Policy: finance

END OF SECTION

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

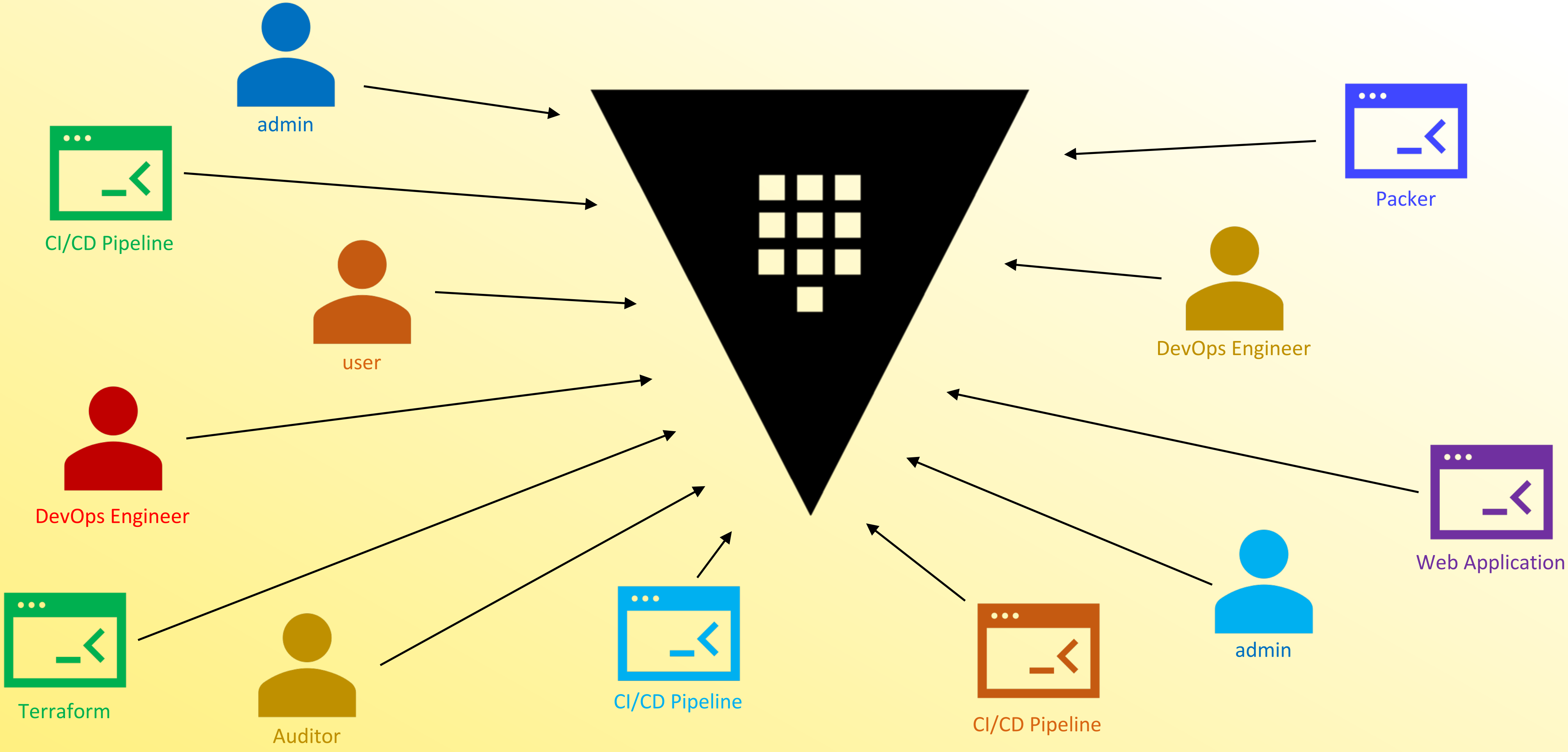
Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Vault Policies

How Do We Determine Who Should Access Secrets

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL



Vault Policies

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

- Vault policies provide operators a way to permit or deny access to certain paths or actions within Vault (RBAC)
  - Gives us the ability to provide granular control over who gets access to secrets
- Policies are written in declarative statements and can be written using JSON or HCL
- When writing policies, always follow the principal of least privilege
  - In other words, give users/applications only the permissions they need

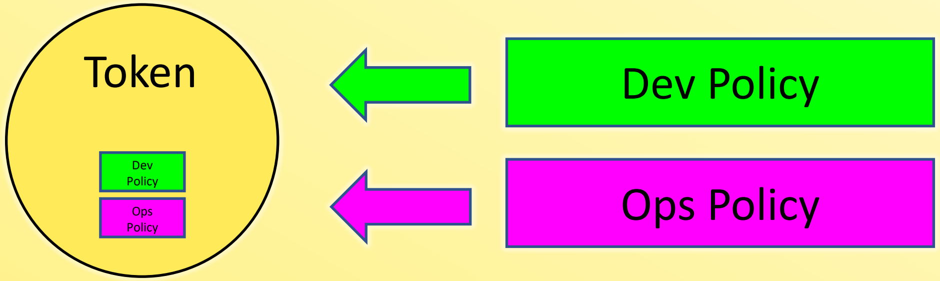
Vault Policies

- Policies are Deny by Default (implicit deny) - therefore you must explicitly grant to paths and related capabilities to Vault clients

  No policy = no authorization

  - Policies support an explicit DENY that takes precedence over any other permission

- Policies are attached to a token. A token can have multiple policies
  - Policies are cumulative and capabilities are additive

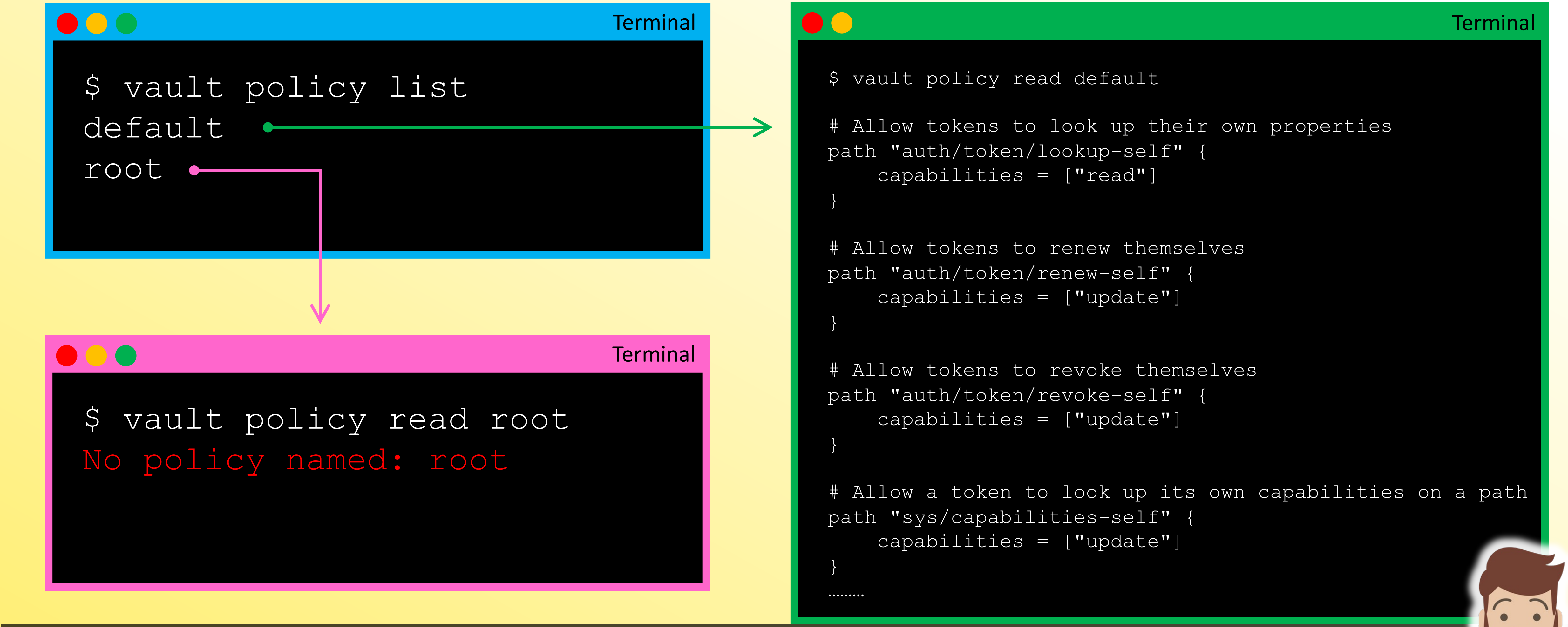


Out of the Box Policies

- `root` policy is created by default – superuser with all permissions
  - You cannot change nor delete this policy
  - Attached to all root tokens

- `default` policy is created by default – provides common permissions
  - You can change this policy, but it cannot be deleted
  - Attached to all non-root tokens by default (can be removed if needed)

Out of the Box Policies



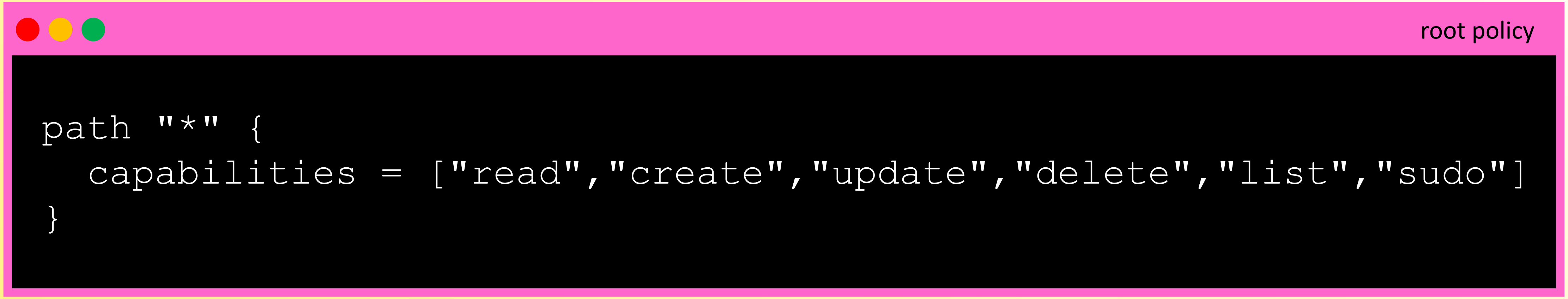
```
$ vault policy list
default
root
```

```
$ vault policy read root
No policy named: root
```

```
$ vault policy read default

# Allow tokens to look up their own properties
path "auth/token/lookup-self" {
    capabilities = ["read"]
}

# Allow tokens to renew themselves
path "auth/token/renew-self" {
    capabilities = ["update"]
}

# Allow tokens to revoke themselves
path "auth/token/revoke-self" {
    capabilities = ["update"]
}

# Allow a token to look up its own capabilities on a path
path "sys/capabilities-self" {
    capabilities = ["update"]
}
………
```

Out of the Box Policies

The root policy <u>does not contain any rules</u> but can do anything within Vault. It should be used with extreme care.



root policy

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

If it *did* have rules, it would probably look something like this….

Managing Policies Using the CLI

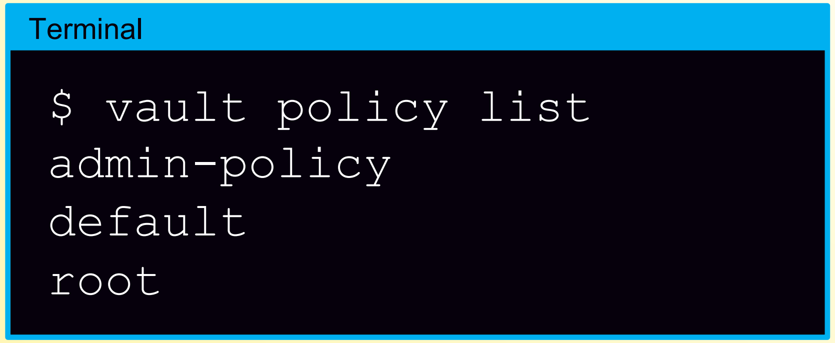
Managing Policies in Vault

Command Line Interface (CLI)

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

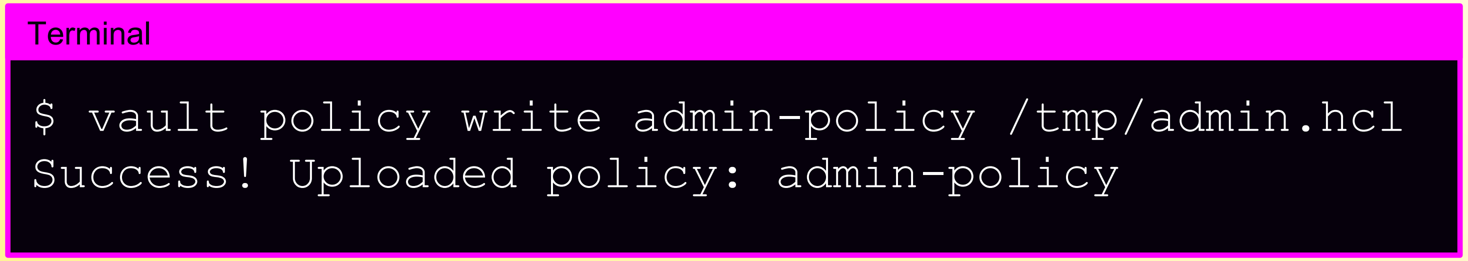
Use the `vault policy` command

- `delete`
- `fmt`
- `list`
- `read`
- `write`



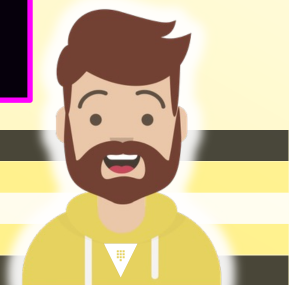
Terminal

```
$ vault policy list
admin-policy
default
root
```



Terminal

```
$ vault policy write admin-policy /tmp/admin.hcl
Success! Uploaded policy: admin-policy
```



Managing Policies in Vault

Command Line Interface (CLI)

Vault CERTIFIED OPERATIONS PROFESSIONAL

```
vault policy write webapp /tmp/webapp.hcl
```

policy: Type of Vault object you want to work with

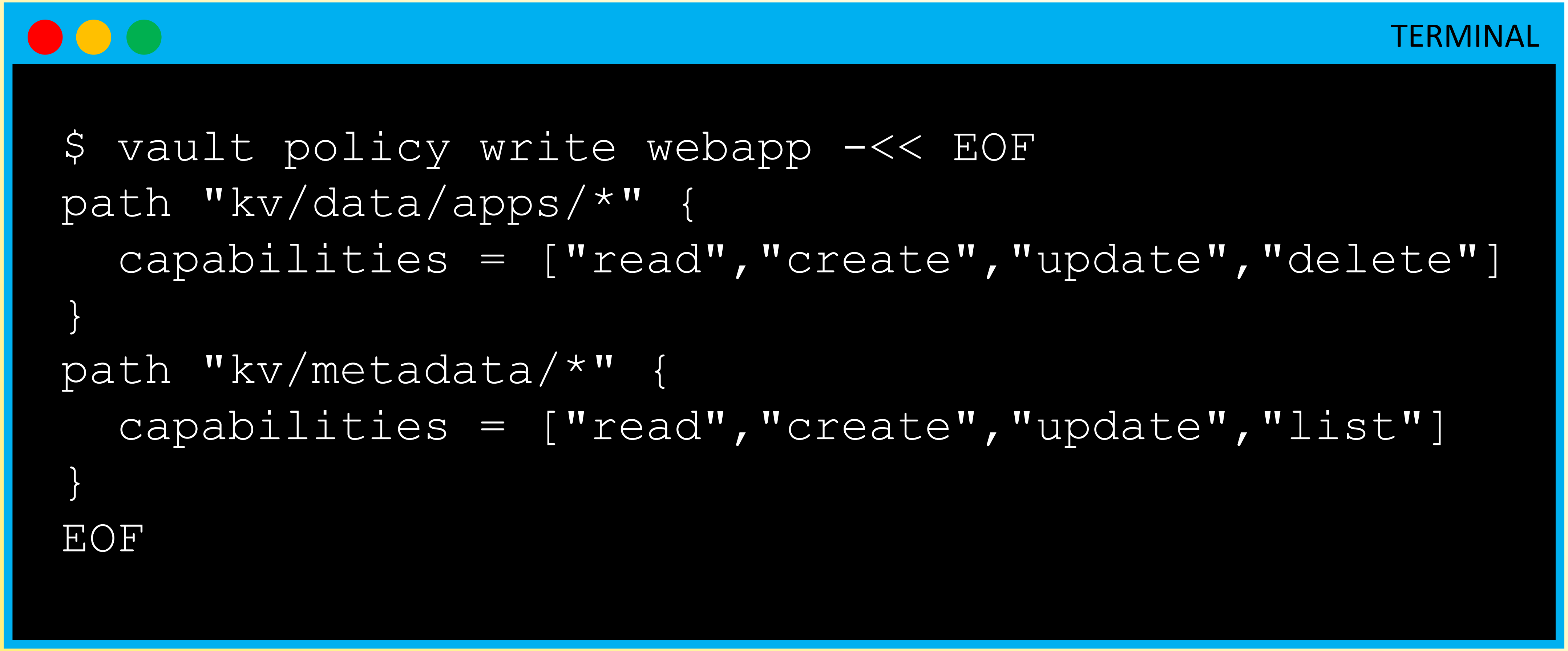
write: Subcommand

webapp: Define the name of the policy you want to create

/tmp/webapp.hcl: The location of the file containing the pre-written policy

Managing Policies in Vault

Command Line Interface (CLI)



```
$ vault policy write webapp -<< EOF
path "kv/data/apps/*" {
  capabilities = ["read","create","update","delete"]
}
path "kv/metadata/*" {
  capabilities = ["read","create","update","list"]
}
EOF
```

Vault

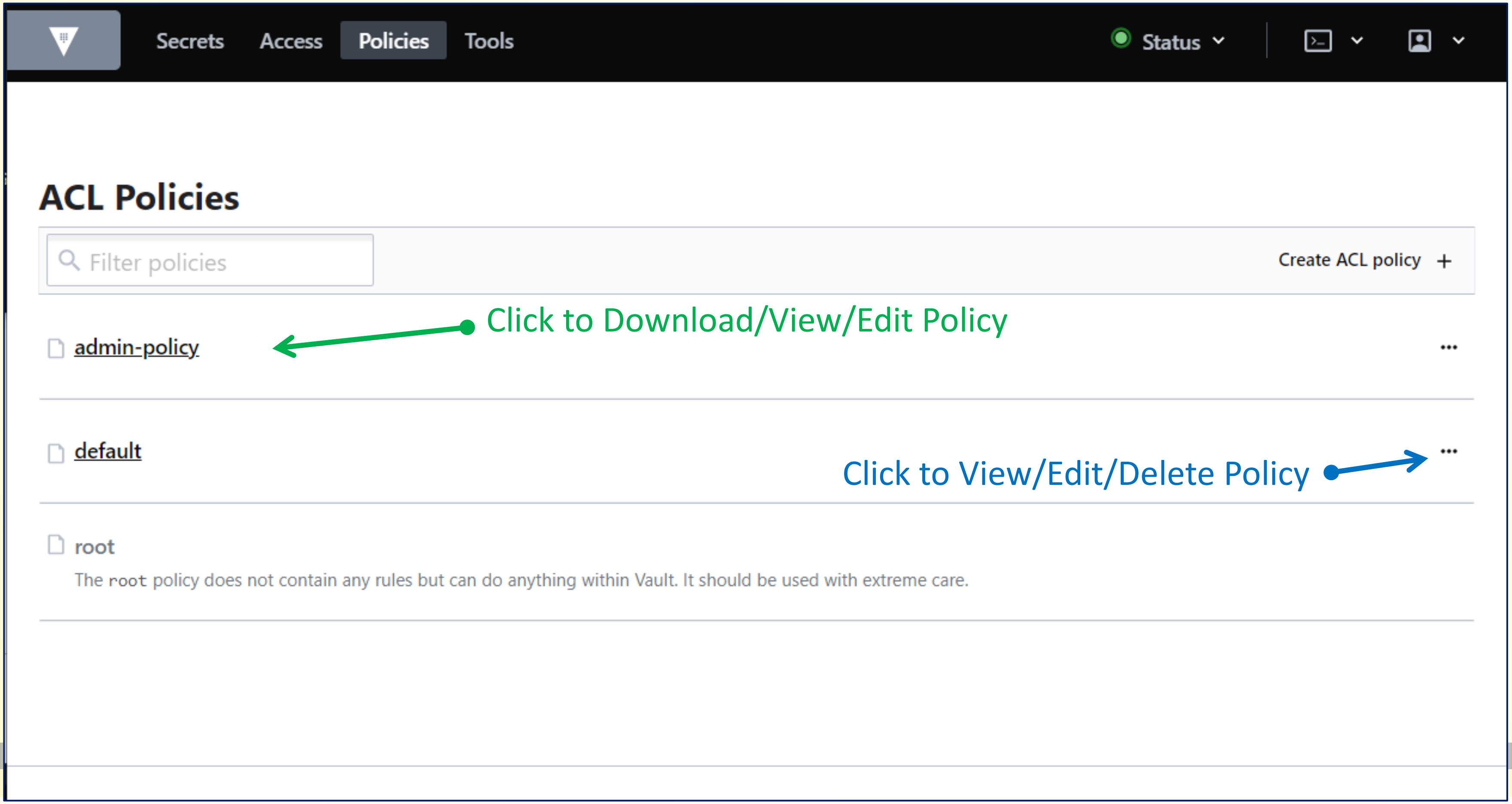
CERTIFIED
OPERATIONS
PROFESSIONAL

Managing Policies Using the UI

Managing Policies in Vault

User Interface (UI)

Create a New Policy



ACL Policies

admin-policy

default

root

The root policy does not contain any rules but can do anything within Vault. It should be used with extreme care.

Click to Download/View/Edit Policy

Click to View/Edit/Delete Policy

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Managing Policies Using the API

Managing Policies in Vault

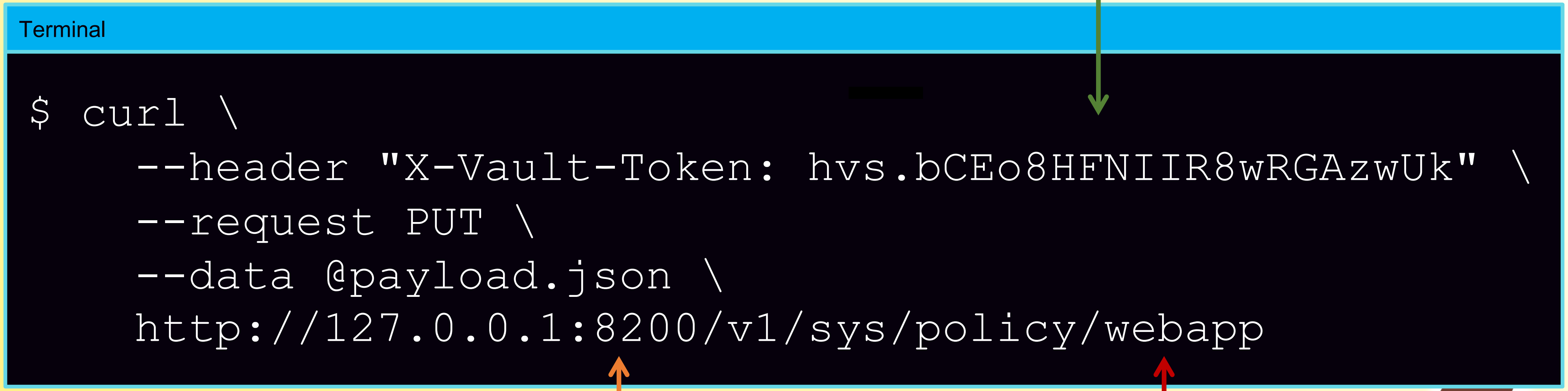
HTTP API

Creating a new Vault policy

- Method: POST

Create Vault Policy

Don't forget you need a valid token



Terminal

```
$ curl \
    --header "X-Vault-Token: hvs.bCEo8HFNIIR8wRGAzwUk" \
    --request PUT \
    --data @payload.json \
    http://127.0.0.1:8200/v1/sys/policy/webapp
```

API Endpoint

Name of the new policy

Managing Policies in Vault

HTTP API

Payload File:

payload.json

```
{
   "policy": "path \"kv/apps/webapp\" { capabilities… "
}
```

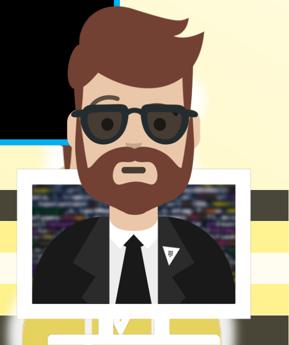
Anatomy of a Vault Policy

Anatomy of a Vault Policy

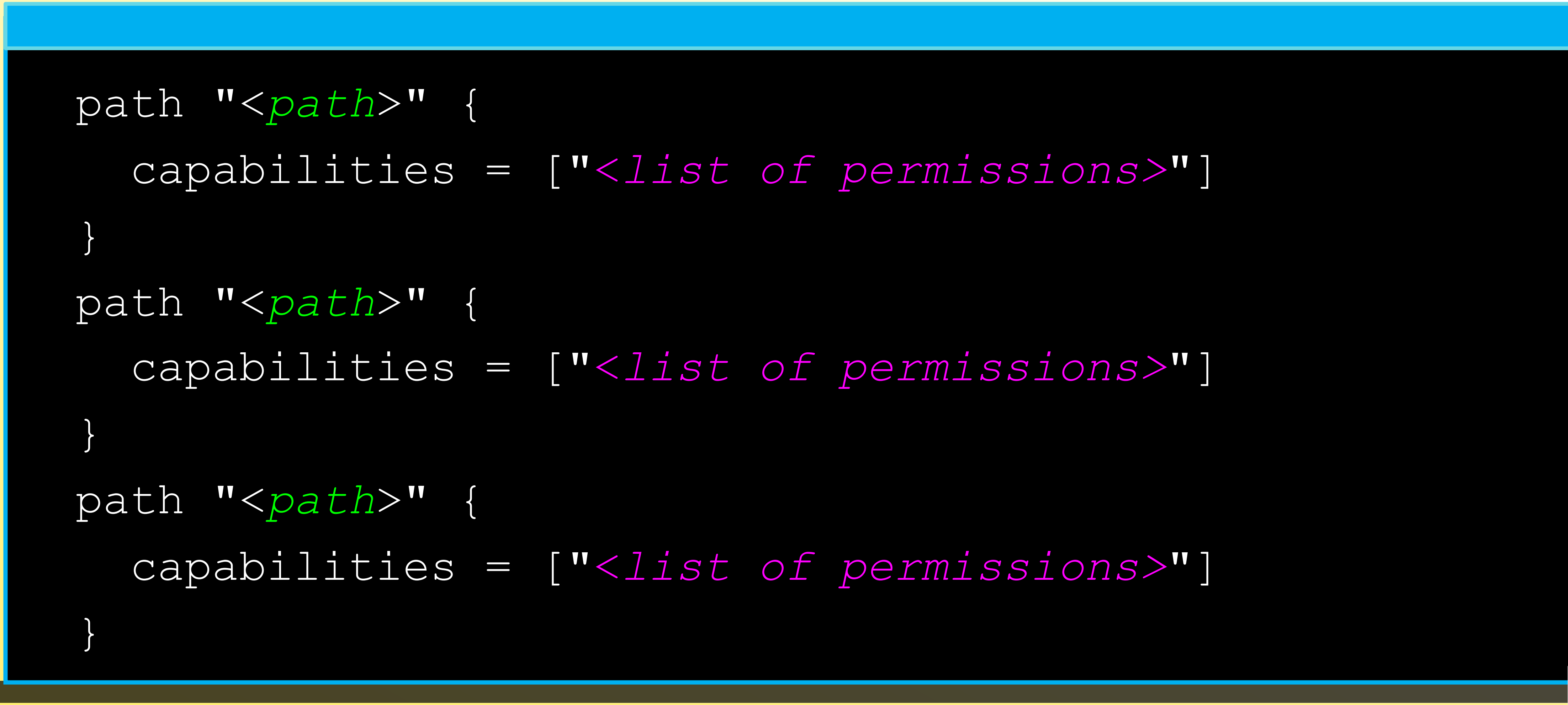
- Remember: Everything in Vault is path based
  - Policies grant or forbid access to those paths and operations

Two key parts to a Vault policy:

```
path "<path>" {
  capabilities = ["<list of permissions>"]
}
```



Anatomy of a Vault Policy



```
path "<path>" {
   capabilities = ["<list of permissions>"]
}
path "<path>" {
   capabilities = ["<list of permissions>"]
}
path "<path>" {
   capabilities = ["<list of permissions>"]
}
```

Anatomy of a Vault Policy

```
path "kv\data\apps\jenkins" {
  capabilities = ["read","update","delete"]
}
path "sys/policies/*" {
  capabilities = ["create","update","list","delete"]
}
path "aws/creds/web-app" {
  capabilities = ["read"]
}
```

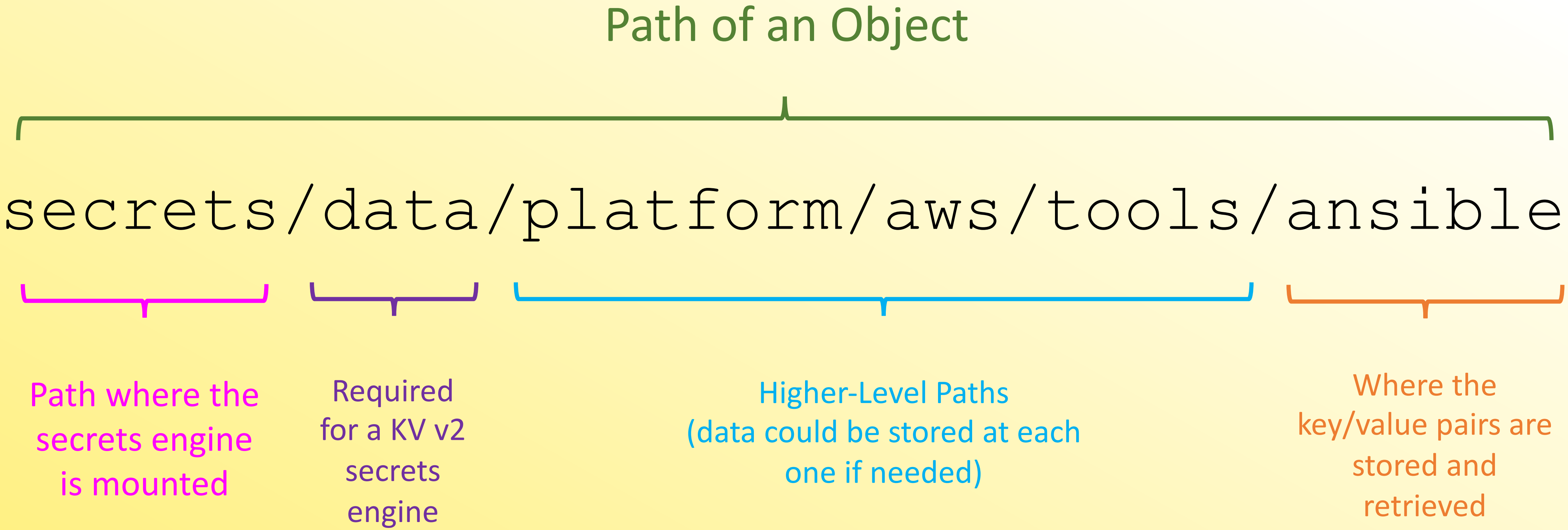
Vault Polices - Path

Vault Policies - Path

- Path: we already know what a path is
  - see Vault Architecture and Pathing Structure in Section 1 for a review
- Examples of paths:
  - sys/policy/vault-admin
  - kv/apps/app01/web
  - auth/ldap/group/developers
  - database/creds/prod-db
  - secrets/data/platform/aws/tools/ansible/app01
  - sys/rekey

The Details are in the Path

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL



Path of an Object

```
secrets/data/platform/aws/tools/ansible
```

Path where the secrets engine is mounted

Required for a KV v2 secrets engine

Higher-Level Paths (data could be stored at each one if needed)

Where the key/value pairs are stored and retrieved

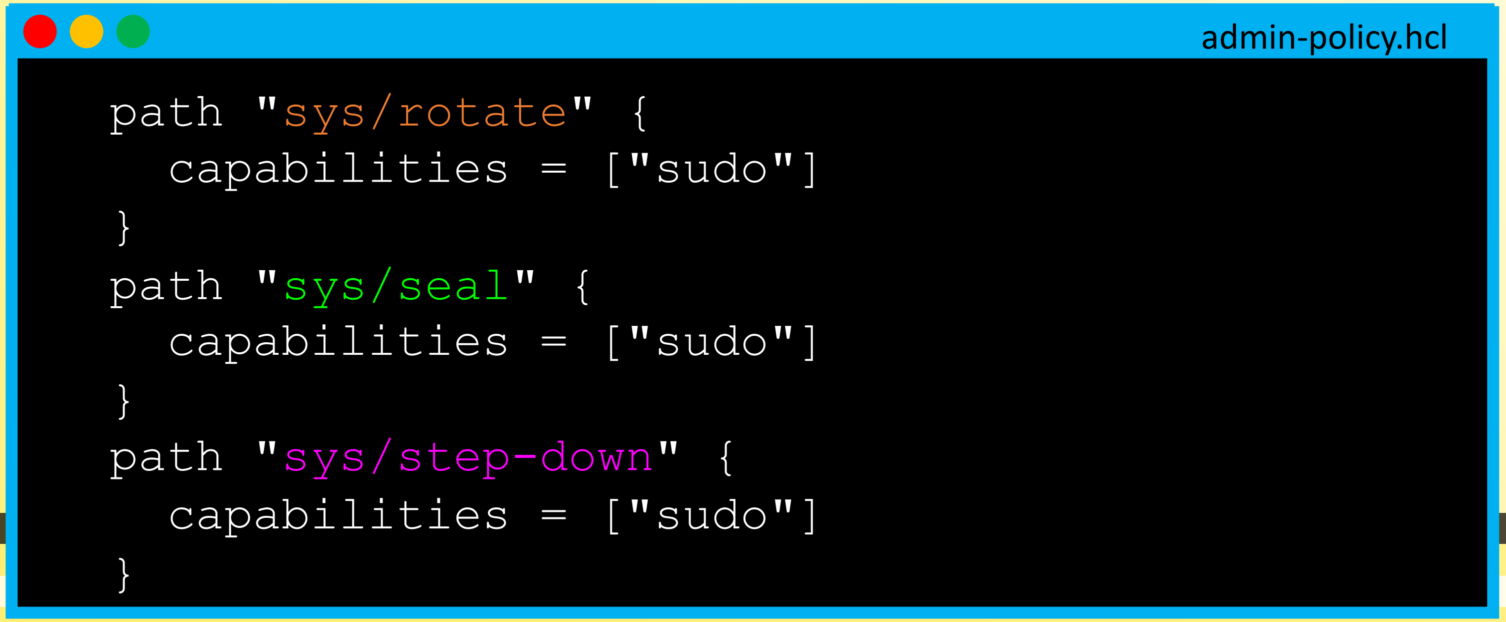
Vault Policies - Path

- Root-Protected Paths
  - Many paths in Vault require a root token or sudo capability to use
  - These paths focus on important/critical paths for Vault or plugins
- Examples of root-protected paths:
  - auth/token/create-orphan (create an orphan token)
  - pki/root/sign-self-issued (sign a self-issued certificate)
  - sys/rotate (rotate the encryption key)
  - sys/seal (manually seal Vault)
  - sys/step-down (force the leader to give up active status)

https://learn.hashicorp.com/tutorials/vault/policies#root-protected-api-endpoints

Vault Policies - Path

- Examples of root-protected paths:
  - sys/rotate (rotate the encryption key)
  - sys/seal (manually seal Vault)
  - sys/step-down (force the leader to give up active status)



admin-policy.hcl

```
path "sys/rotate" {
  capabilities = ["sudo"]
}
path "sys/seal" {
  capabilities = ["sudo"]
}
path "sys/step-down" {
  capabilities = ["sudo"]
}
```

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Vault Polices - Capabilities

Vault Policies - Capabilities

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

- Capabilities define what can we do?
  - Capabilities are specified as a list of strings (yes, even if there's just one)

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

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

`create` = if the key does not yet exist
`update` = if the key exists and you want to replace/update it

Vault Policies - Capabilities

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

- **Create** – create a new entry
- **Read** – read credentials, configurations, etc
- **Update** – overwrite the existing value of a secret or configuration
- **Delete** – delete something
- **List** – view what's there (doesn't allow you to read)
- **Sudo** – used for root-protected paths
- **Deny** – deny access – always takes precedence over any other capability

Note: Write is not a valid capability

Vault Policy - Example

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

Requirement:

- Access to generate database credentials at database/creds/db01
- Create, Update, Read, and Delete secrets stored at kv/apps/dev-app01

```
path "database/creds/dev-db01" {
   capabilities = ["read"]
}
path "kv/apps/dev-app01" {
   capabilities = ["create", "read", "update", "delete"]
}
```

One Policy
With
Mulitple Rules

Vault Policy - Example

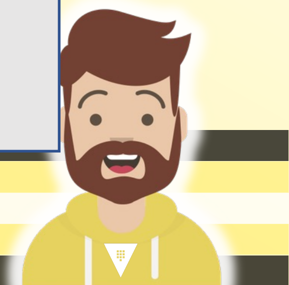
Requirements:

- Access to read credentials after the path kv/apps/webapp
- Deny access to kv/apps/webapp/super-secret

```
path "kv/apps/webapp/*" {
  capabilities = ["read"]
}
path "kv/apps/webapp/super_secret" {
  capabilities = ["deny"]
}
```

Tree

```
--kv
   |--apps
        |--webapp
               |--super_secret ☒
               |--api_token ☑
               |--host_name ☑
        |--mid-tier
        |--database
   |--cloud
        |--aws
             |--prod
        |--gcp
   |--dev
```

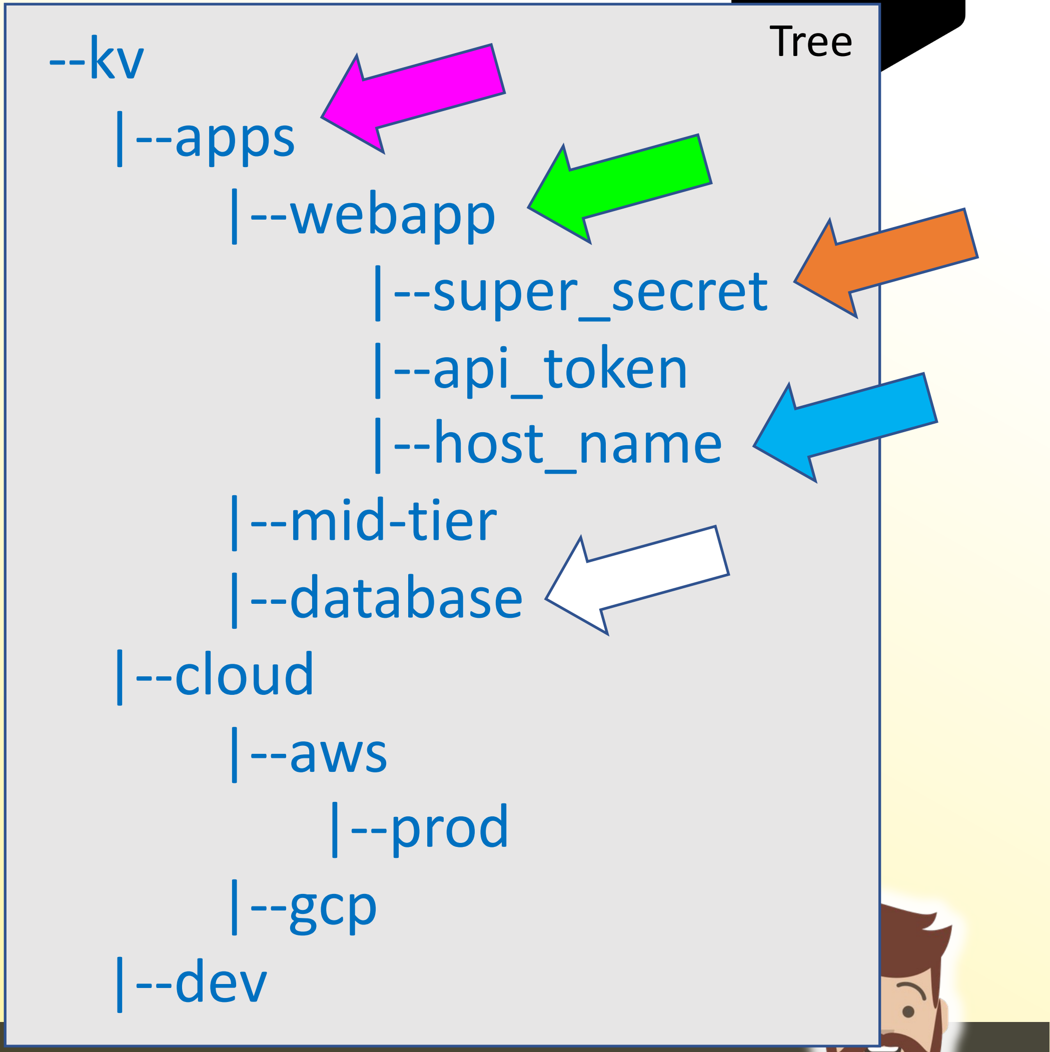


Pop Quiz

Q: Does this policy permit access to kv/apps/webapp?

A: No, because the policy only permits access to secrets AFTER kv/apps/webapp

```
path "kv/apps/webapp/*" {
  capabilities = ["read"]
}
path "kv/apps/webapp/super_secret" {
  capabilities = ["deny"]
}
```



Tree

```
--kv
    |--apps
        |--webapp
            |--super_secret
            |--api_token
            |--host_name
        |--mid-tier
        |--database
    |--cloud
        |--aws
            |--prod
        |--gcp
    |--dev
```

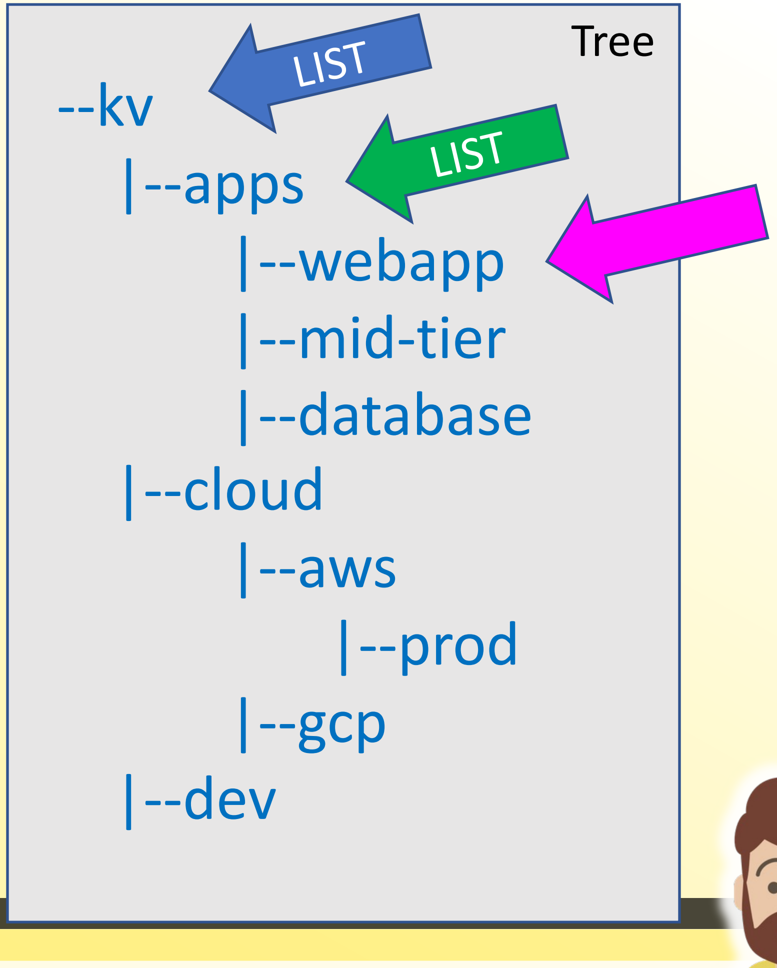
Pop Quiz

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

Q: Does this policy permit you to browse to kv/apps/webapp in the UI?

A: No, because the policy only permits list at the listed path, not the paths leading up to the desired path

```
path "kv/apps/webapp/*" {
  capabilities = ["read", "list"]
}
```



```
--kv
  |--apps
      |--webapp
      |--mid-tier
      |--database
  |--cloud
      |--aws
          |--prod
      |--gcp
  |--dev
```

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

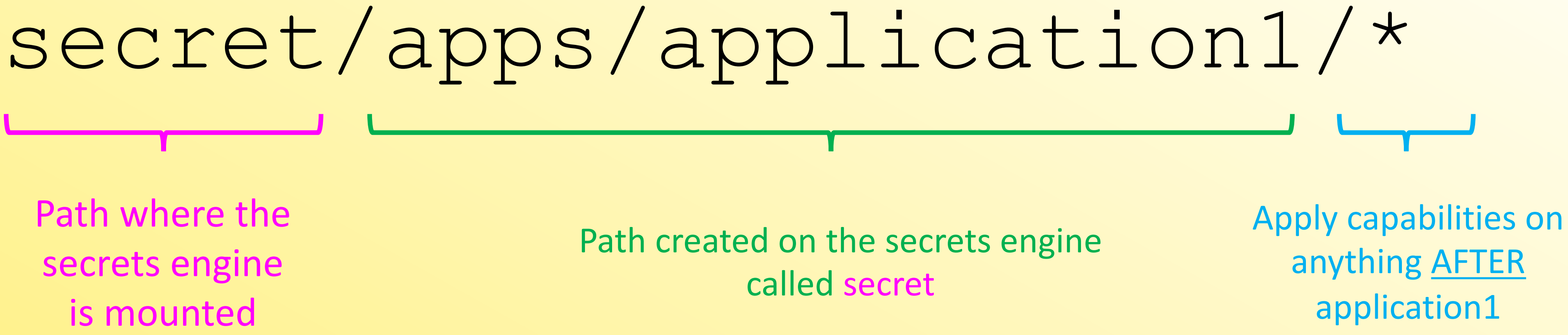
Customizing the Path

Using the * to Customize the Path

- The glob (*) is a wildcard and can only be used at the end of a path
- Can be used to signify anything "after" a path or as part of a pattern

- Examples:
  - secret/apps/application1/* - allows any path after application1
  - kv/platform/db-* - would match kv/platform/db-2 but not kv/platform/db2

The Details are in the Path



```
secret/apps/application1/*
```

Path where the secrets engine is mounted

Path created on the secrets engine called secret

Apply capabilities on anything AFTER application1

Does it Match?

```
secret/apps/application1/*
```

Path must start with this – nothing else

Must ALSO include something beyond application1

Paths that Match

- ✓ `secret/apps/application1/db`
- ✓ `secret/apps/application1/data/production`
- ✓ `secret/apps/application1/web-app`
- ✓ `secret/apps/application1/keys/api_key`

Paths that Do Not Match

- X `secret/apps/database`
- X `secret/apps/application2`
- X `secret/data/front-end`
- X `kv/secret/app/application`

Pop Quiz

Given the policy:

```
path "secret/apps/application1/*" {
   capabilities = ["read"]
}
```

required

Can I read from the following path?

`secret/apps/application1`

Answer: No, because the policy only permits read access for anything AFTER application1, not the path `secret/apps/application1` itself

Pop Quiz

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

If we wanted to ALSO read from `secret/apps/application1`, the policy would look like this:

```
path "secret/apps/application1/*" {
  capabilities = ["read"]
}

path "secret/apps/application1" {
  capabilities = ["read"]
}
```

NEW

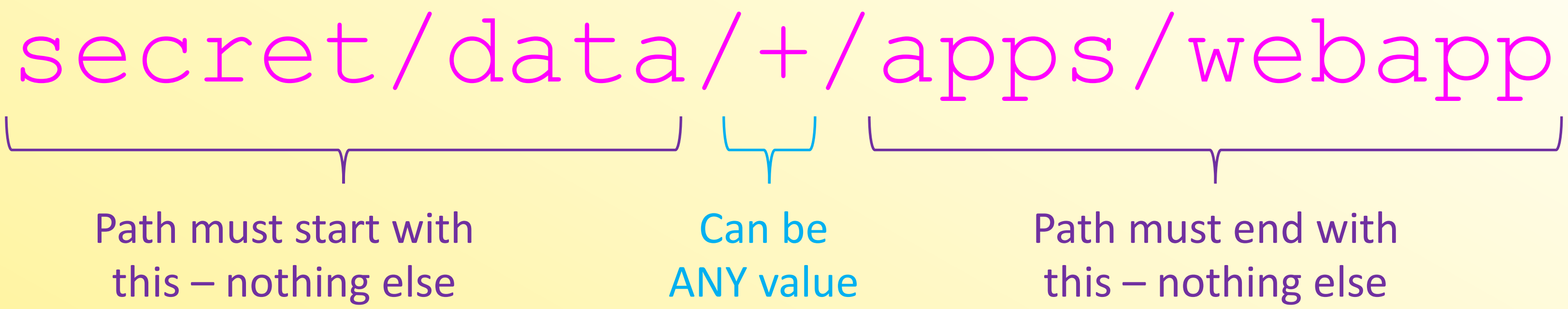
Using the + to Customize the Path

- The plus (+) supports wildcard matching for a single directory in the path
- Can be used in multiple path segments (i.e., secret/+/+/db)
- Examples:
  - secret/+/db - matches secret/db2/db or secret/app/db
  - kv/data/apps/+/webapp – matches the following:
    - kv/data/apps/dev/webapp
    - kv/data/apps/qa/webapp
    - kv/data/apps/prod/webapp

The Details are in the Path

$$\underbrace{\texttt{secret}}_{\text{Path where the secrets engine is mounted}}\texttt{/}\underbrace{\texttt{data}}_{\text{Used for KV V2 Secrets Engine}}\texttt{/}\underbrace{\texttt{+}}_{\text{Can be ANY value}}\texttt{/}\underbrace{\texttt{apps/webapp}}_{\text{Remaining path}}$$

Does it Match?



`secret/data/+/apps/webapp`

Path must start with this – nothing else

Can be ANY value

Path must end with this – nothing else

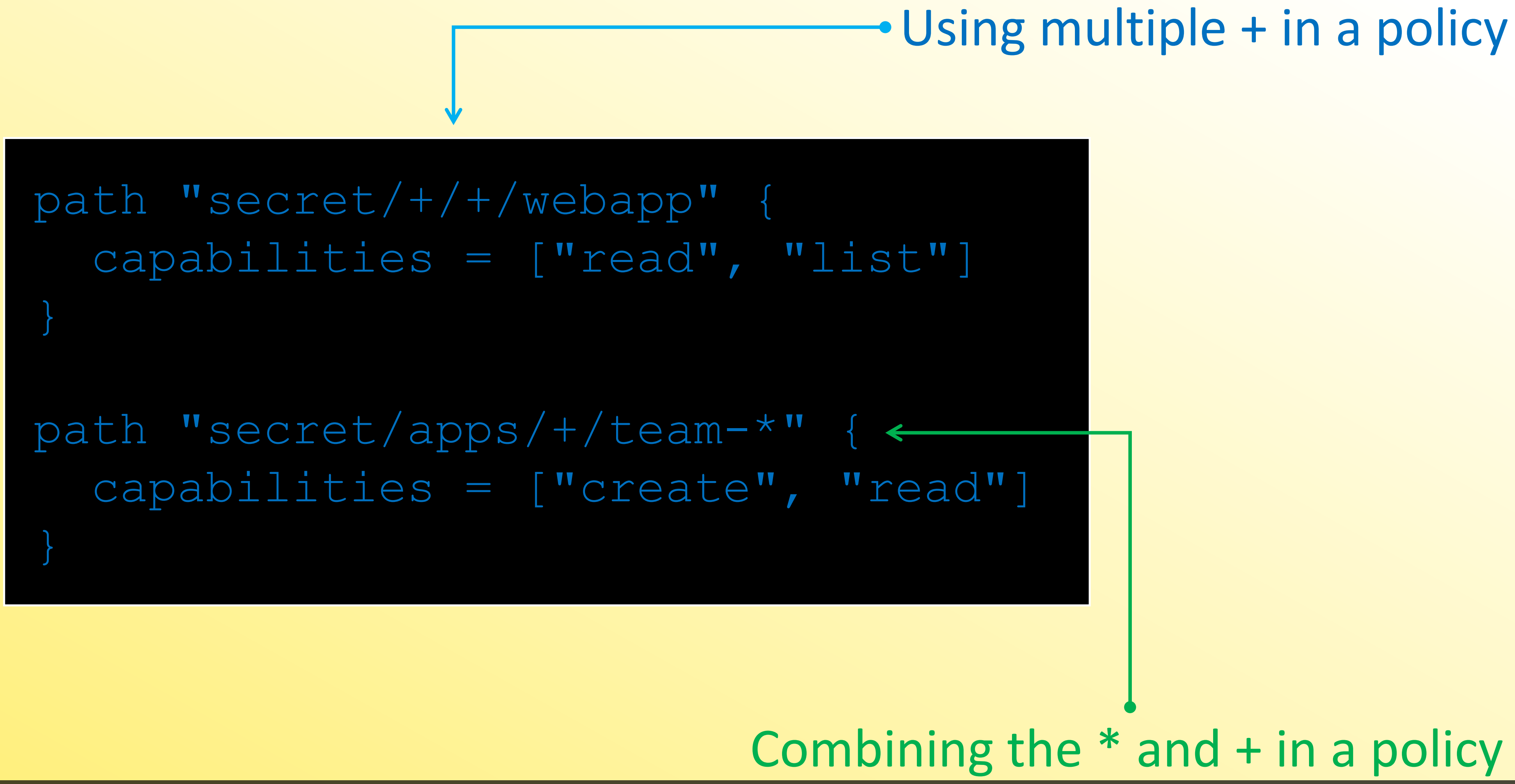
Paths that Match

- ✓ `secret/data/production/apps/webapp`
- ✓ `secret/data/dev1/apps/webapp`
- ✓ `secret/data/team-abc/apps/webapp`
- ✓ `secret/data/456/apps/webapp`

Paths that Do Not Match

- X `secret/data/apps/webapp`
- X `secret/app123/dev`
- X `secret/data/front-end/apps`
- X `secret/dev/apps/webapp`

Example Policy



Using multiple + in a policy

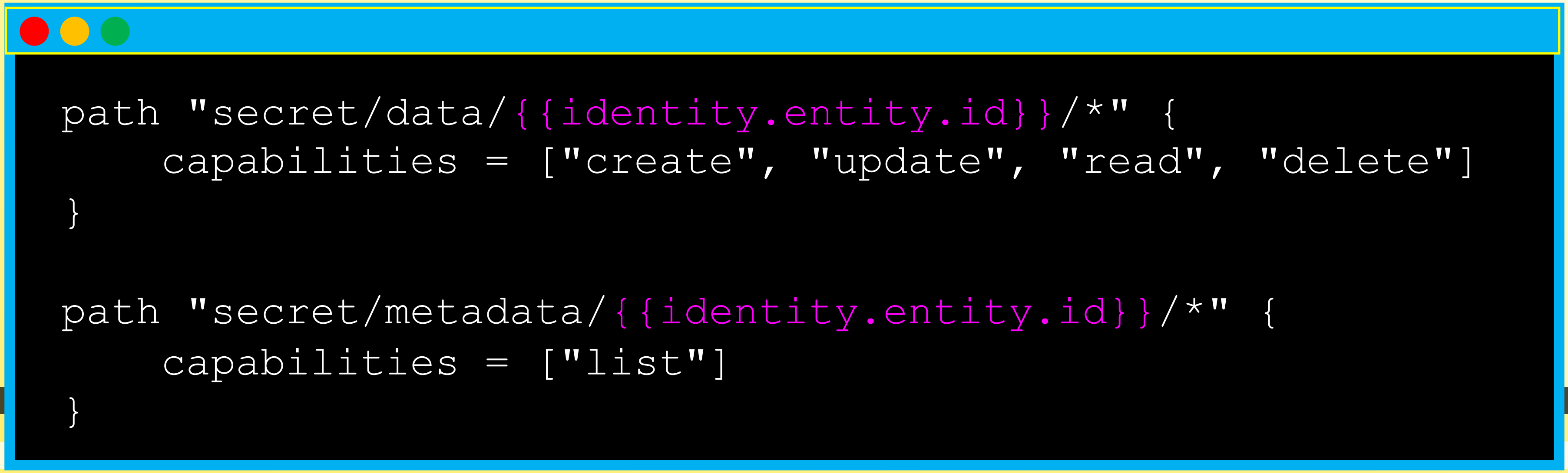
```
path "secret/+/+/webapp" {
  capabilities = ["read", "list"]
}

path "secret/apps/+/team-*" {
  capabilities = ["create", "read"]
}
```

Combining the * and + in a policy

ACL Templating

- Use variable replacement in some policy strings with values available to the token
- Define policy paths containing double curly braces: {{<parameter>}}

Example: Creates a section of the key/value v2 secret engine to a specific user



```
path "secret/data/{{identity.entity.id}}/*" {
    capabilities = ["create", "update", "read", "delete"]
}

path "secret/metadata/{{identity.entity.id}}/*" {
    capabilities = ["list"]
}
```

ACL Templating

Parameter	Description
identity.entity.id	The entity's ID
identity.entity.name	The entity's name
identity.entity.metadata.<<metadata key>>	Metadata associated with the entity for the given key
identity.entity.aliases.<<mount accessor>>.id	Entity alias ID for the given mount
identity.entity.aliases.<<mount accessor>>.name	Entity alias name for the given mount
identity.entity.aliases.<<mount accessor>>.metadata.<<metadata key>>	Metadata associated with the alias for the given mount and metadata key
identity.groups.ids.<<group id>>.name	The group name for the given group ID
identity.groups.names.<<group name>>.id	The group ID for the given group name
identity.groups.names.<<group id>>.metadata.<<metadata key>>	Metadata associated with the group for the given key
identity.groups.names.<<group name>>.metadata.<<metadata key>>	Metadata associated with the group for the given key

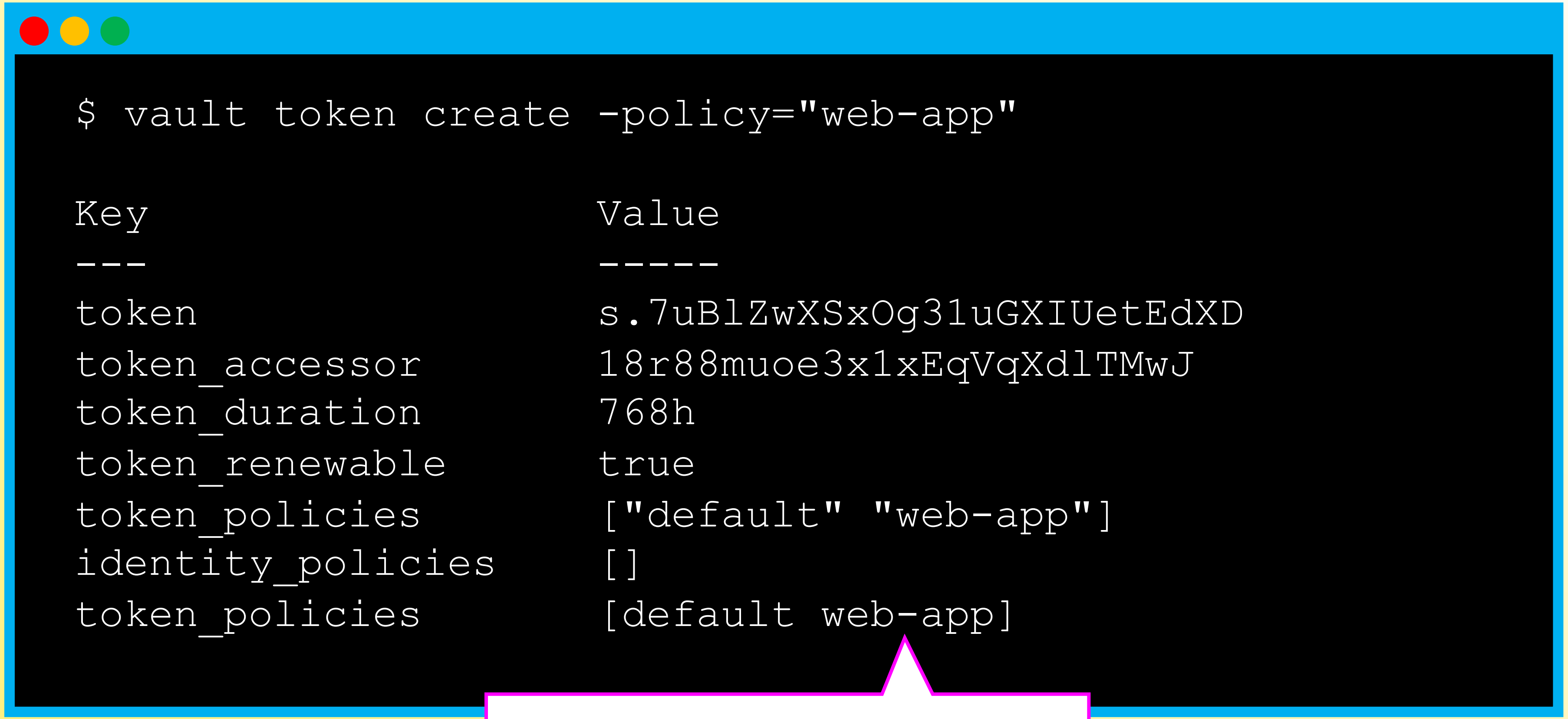
https://learn.hashicorp.com/tutorials/vault/policy-templating?in=vault/policies

Working with Policies

What Policies are Attached?

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

Create a new token with "web-app" policy attached:



```
$ vault token create -policy="web-app"

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

Every token gets the **default** policy plus the assigned policy or policies

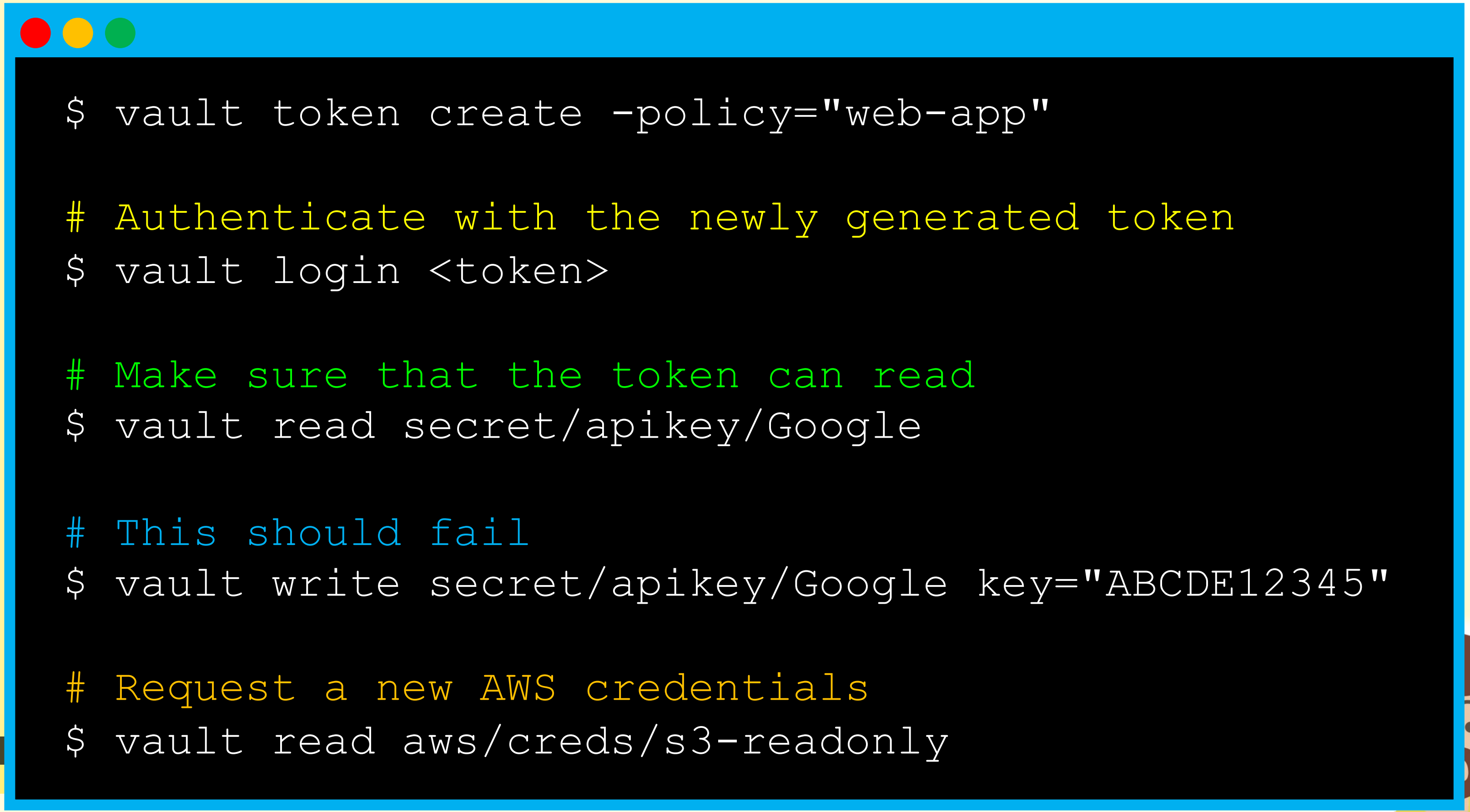
Testing Policies

Test to make sure the policy fulfills the requirements

Example Requirements:

- Clients must be able to request AWS credential granting read access to a S3 bucket
- Read secrets from secret/apikey/Google

TERMINAL



```
$ vault token create -policy="web-app"

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

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

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

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

Administrative Policies

- Permissions for Vault backend functions live at the sys/ path
- Users/admins will need policies that define what they can do within Vault to administer Vault itself
  - Unsealing
  - Changing policies
  - Adding secret backends
  - Configuring database configurations

Administrative Policies

- Licensing
- Setup New Vault Cluster
- Configure UI
- Rotate Keys
- Seal Vault

```
# Configure License
path "sys/license" {
  capabilities = ["read", "list", "create", "update", "delete"]
}
# Initialize Vault
path "sys/init" {
  capabilities = ["read", "update", "create"]
}
# Configure UI in Vault
path "sys/config/ui" {
  capabilities = ["read", "list", "update", "delete", "sudo"]
}
# Allow rekey of unseal keys for Vault
path "sys/rekey/*" {
  capabilities = ["read", "list", "update", "delete"]
}
# Allows rotation of master key
path "sys/rotate" {
  capabilities = ["update", "sudo"]
}
# Allows Vault seal
path "sys/seal" {
  capabilities = ["sudo"]
}
```

END OF SECTION

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Understand Sentinel Policies

What is Sentinel?

Vault CERTIFIED OPERATIONS PROFESSIONAL

Sentinel is an embeddable **policy as code** framework to enable *fine-grained*, *logic-based* policy decisions that can be *extended* to source external information to make decisions.

Policy as Code

Treat policy like an application — version control, pull review, and automate tests. Use programming constructs to determine policy decisions beyond the limited constraints of typical ACL systems.

Fine Grained, Conditioned-Based

Treat policy like an application — version control, pull review, and automate tests. Use programming constructs to determine policy decisions beyond the limited constraints of typical ACL systems.

Embedded

Sentinel is embedded to enable policy enforcement in the data path to actively reject violating behavior instead of passively detecting.

Enforcement Levels

Advisory, soft-mandatory, and hard-mandatory levels allow policy writers to warn on or reject offending behavior.

External Information

Sentinel can permit or deny actions based upon external information available to the token, such as time, IP address, requested path, etc.

Multi-Cloud Compatible

Ensure infrastructure changes are within business and regulatory policy on every infrastructure provider.

Multi-Platform

Sentinel is NOT just a Vault feature.

It is available in the Enterprise versions of other HashiCorp Products.

HashiCorp **Terraform** Enterprise

HashiCorp **Nomad** Enterprise

HashiCorp **Vault** Enterprise

HashiCorp **Consul** Enterprise

Types of Sentinel Policies

Role Governing Policies (RGPs)

- Sentinel policies that are tied to **tokens**, **identity entities**, or **identity groups**
- Access to rich set of controls across various aspects of Vault

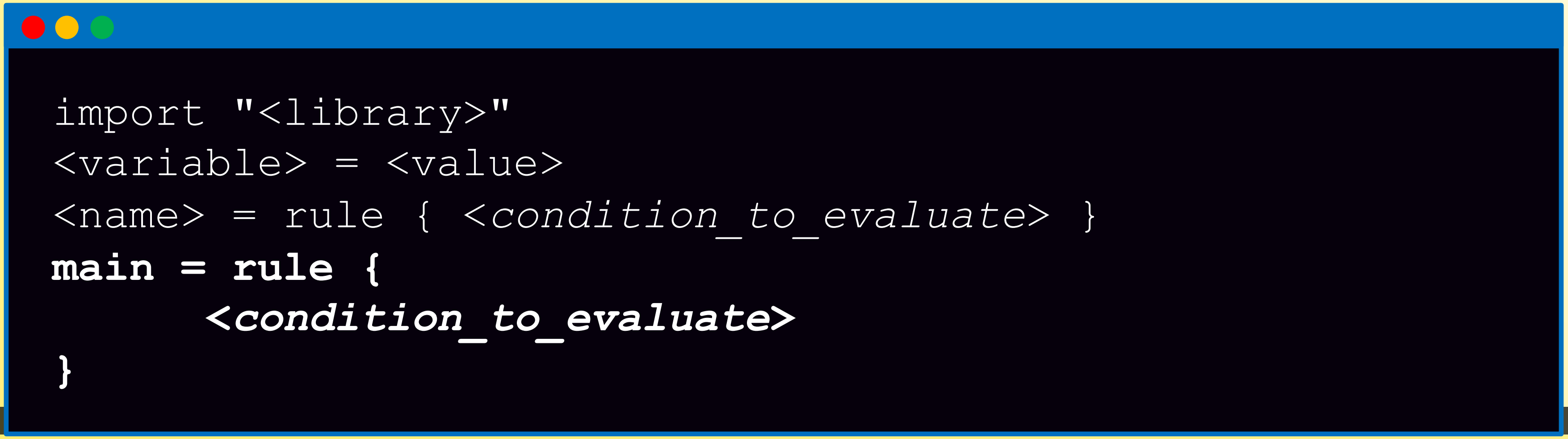
Endpoint Governing Policies (EGPs)

- Sentinel policies that are tied to **paths** instead of tokens
- Access to as much request information as possible
  - Can take an effect even on unauthenticated paths (e.g., login paths)

Anatomy of a Sentinel Policy

Vault CERTIFIED OPERATIONS PROFESSIONAL

- **Import** – access to reusable libraries to import information or use features
- **Main** – (required) the main rule to be evaluated
- **Rule** – describes a set of conditions resulting in either true or false
- **Variables** – optional, dynamically typed variable



```
import "<library>"
<variable> = <value>
<name> = rule { <condition_to_evaluate> }
main = rule {
      <condition_to_evaluate>
}
```

Imports

Example of **Imports** that can be used with Sentinel:

- **`base64`** – encode & decode Base64 values
- **`decimal`** – provides functions for operating on numbers as decimals
- **`http`** – enables the use of HTTP-accessible data outside of the runtime in Sentinel rules
- **`json`** – parse and access a JSON document
- **`runtime`** – contains various information about Sentinel runtime
- **`sockaddr`** – enables working with IP addresses
- **`strings`** – enables common string operations
- **`time`** – provides access to execution time and time functions
- **`types`** – ability to parse an object's type
- **`units`** – provides access to quick calculations for various byte units
- **`version`** – used to parse versions and version constraints

These allow fine-grained controls over your Vault environment

https://docs.hashicorp.com/sentinel/imports

Sentinel Policy Example - RGP

Only allow a specific entity or groups

```
main = rule {
    identity.entity.name is "jeff" or
      identity.entity.id is "fe2a5bfd-c483-9263-b0d4-f9d345efdf9f" or
      "sysops" in identity.groups.names or
      "14c0940a-5c07-4b97-81ec-0d423accb8e0" in keys(identity.groups.by-id)
}
```

If the user "Jeff" is deleted and recreated, the match will fail because we're also enforcing the entity ID

Sentinel Policy Example - EGP

Disallow all previously-generated tokens based on date:

- You could apply this EGP to the "*" endpoint

```
import "time"

main = rule when not request.unauthenticated {
    time.load(token.creation_time).unix >
       time.load("2022-12-25T00:00:01Z").unix
}
```

Could be used as a "break-glass" scenario where previous tokens were compromised

Sentinel Policy Example - EGP

```
import "sockaddr"
import "mfa"
import "strings"

# We expect logins to come only from a specific private IP range
cidrcheck = rule {
    sockaddr.is_contained(request.connection.remote_addr, "10.0.23.0/16")
}

# Require Ping MFA validation to succeed
ping_valid = rule {
    mfa.methods.ping.valid
}

main = rule when request.path is "auth/ldap/login" {
    ping_valid and cidrcheck
}
```

Sets the scope of policy

Must also pass both rules

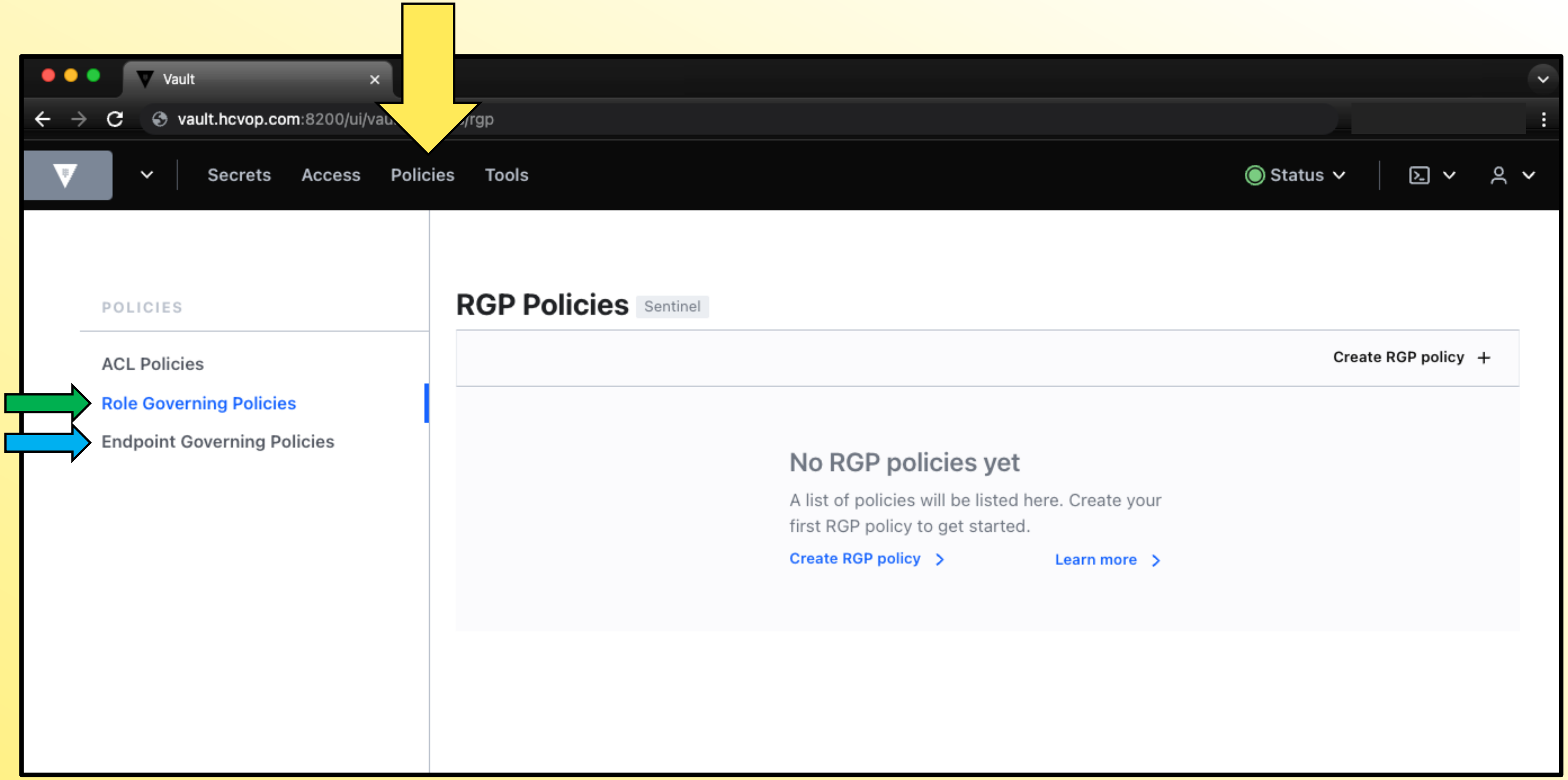
Enforcement Levels

Sentinel offers three different enforcement levels that can be set per Sentinel policy:

Enforcement Level	Description
Advisory	The policy is allowed to fail
Soft Mandatory	The policy must pass unless an override is specified
Hard Mandatory	The policy muss pass no matter what

To override a Sentinel policy (soft mandatory), use the –policy-override flag when executing the Vault command

Deploy Sentinel Policies via UI



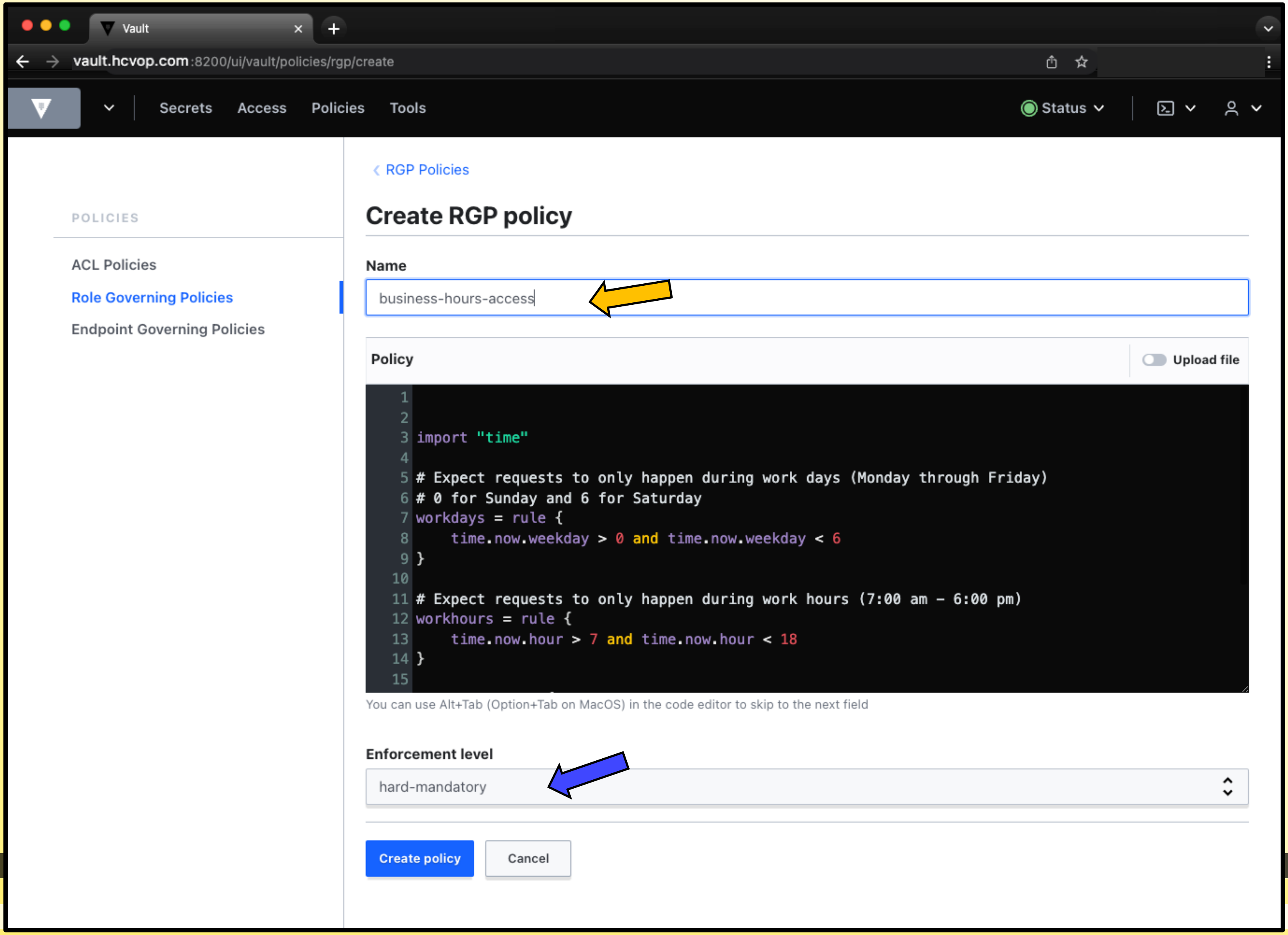
RGP Policies Sentinel

No RGP policies yet

A list of policies will be listed here. Create your first RGP policy to get started.

Deploy RGP Sentinel Policy via UI

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL



Create RGP policy

Name

business-hours-access

Policy

Upload file

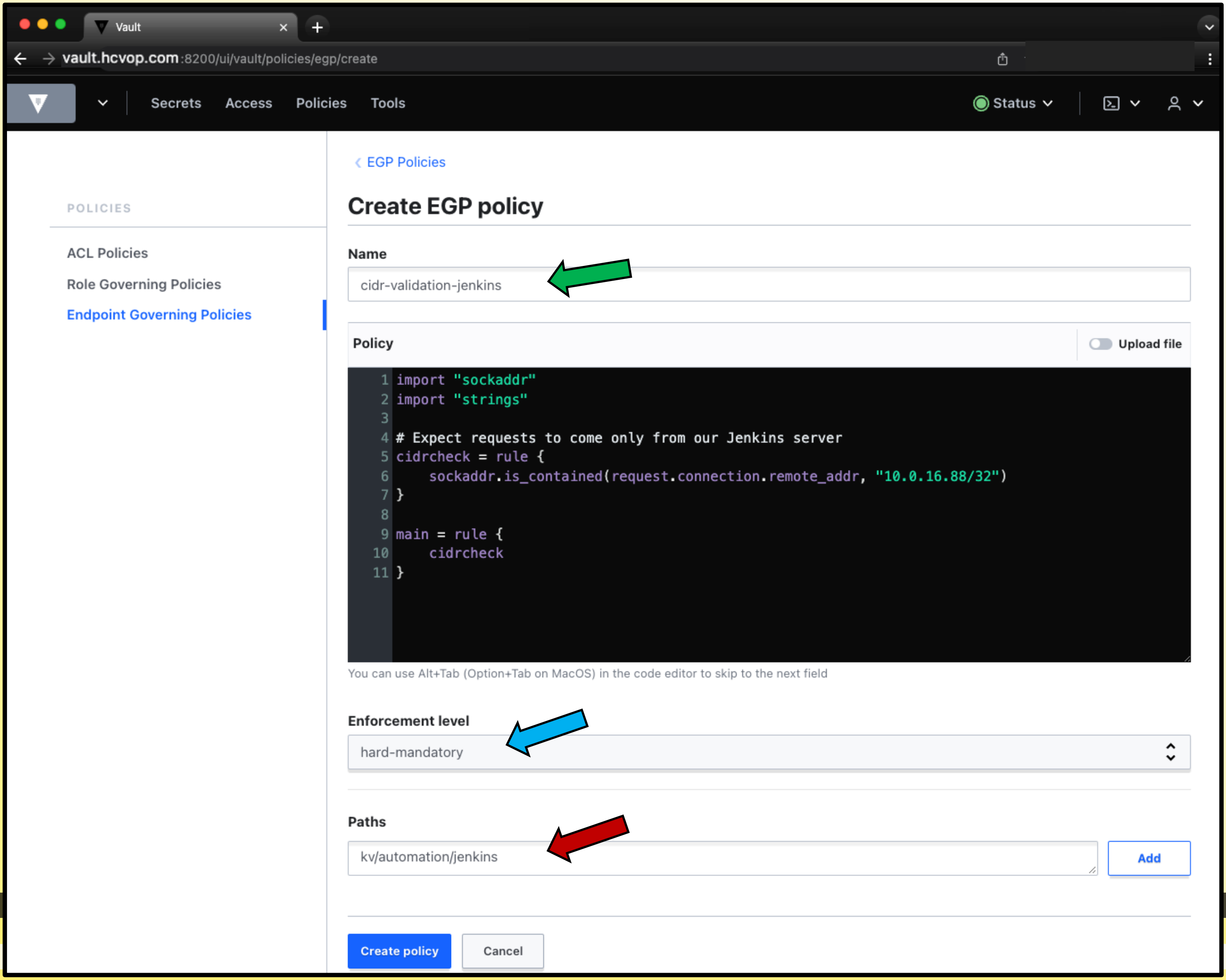
```


import "time"

# Expect requests to only happen during work days (Monday through Friday)
# 0 for Sunday and 6 for Saturday
workdays = rule {
    time.now.weekday > 0 and time.now.weekday < 6
}

# Expect requests to only happen during work hours (7:00 am - 6:00 pm)
workhours = rule {
    time.now.hour > 7 and time.now.hour < 18
}

```

You can use Alt+Tab (Option+Tab on MacOS) in the code editor to skip to the next field

Enforcement level

hard-mandatory

Deploy EGP Sentinel Policy via UI



Create EGP policy

Name: cidr-validation-jenkins

Policy

```
import "sockaddr"
import "strings"

# Expect requests to come only from our Jenkins server
cidrcheck = rule {
    sockaddr.is_contained(request.connection.remote_addr, "10.0.16.88/32")
}

main = rule {
    cidrcheck
}
```

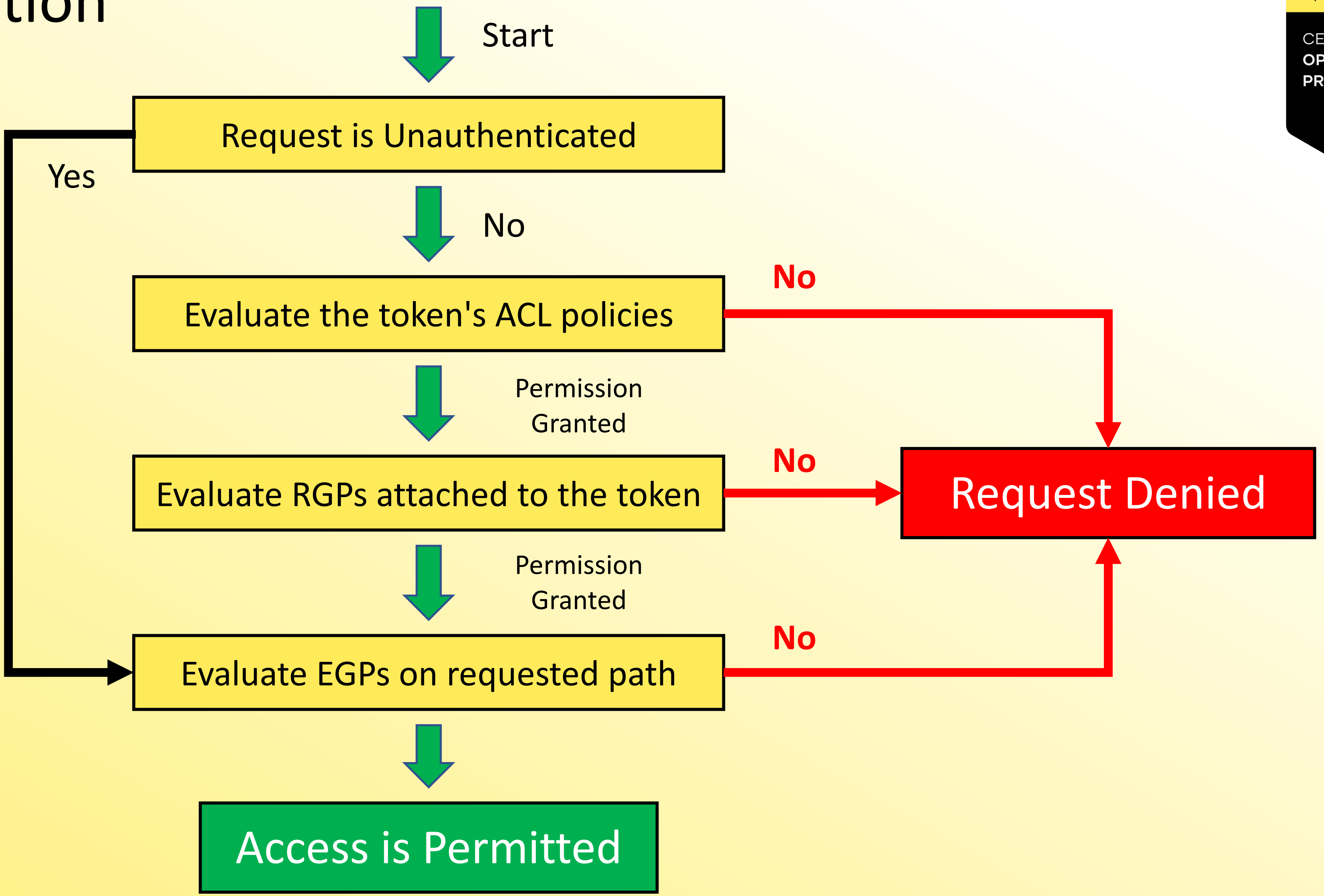
You can use Alt+Tab (Option+Tab on MacOS) in the code editor to skip to the next field

Enforcement level: hard-mandatory

Paths: kv/automation/jenkins

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

Policy Evaluation


Start
Request is Unauthenticated
Yes
No
Evaluate the token's ACL policies
No
Permission Granted
Evaluate RGPs attached to the token
No
Request Denied
Permission Granted
Evaluate EGPs on requested path
No
Access is Permitted

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

END OF SECTION

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

Define Control Groups and Describe their Basic Workflow

Control Groups

- Control groups add an additional authorization requirement on configured paths

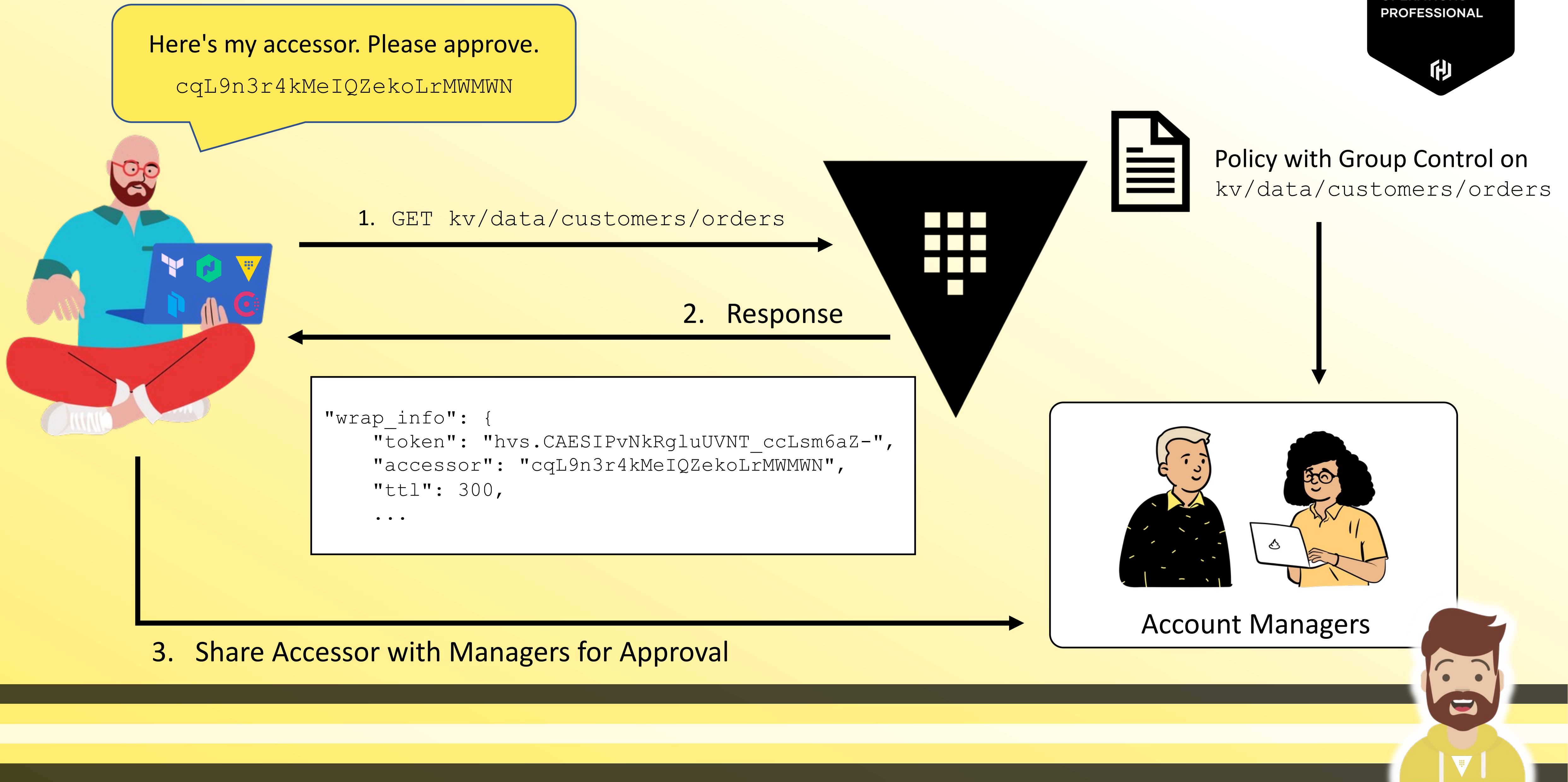
- When a control group is created, the following will occur:
  1. The client makes a request to a path as normal
  2. Vault returns a wrapping token – rather than the requested secrets
  3. The authorizers defined in the control group policy must approve the request
  4. Once all authorizations are satified, the client can unwrap the secrets

Control Group Factors

- Control Group requirements can be specified in either **ACL policies** or within a **Sentinel policy**

- Currently, the only supported Control Group **factor** is an Identity Group
  - An authorizer must belong to a specific identity group
  - The policy will define the group, or groups, who are approvers for the requested path

Control Group Workflow



Here's my accessor. Please approve.

`cqL9n3r4kMeIQZekoLrMWMWN`

1. `GET kv/data/customers/orders`

2. Response

```
"wrap_info": {
    "token": "hvs.CAESIPvNkRgluUVNT_ccLsm6aZ-",
    "accessor": "cqL9n3r4kMeIQZekoLrMWMWN",
    "ttl": 300,
    ...
```

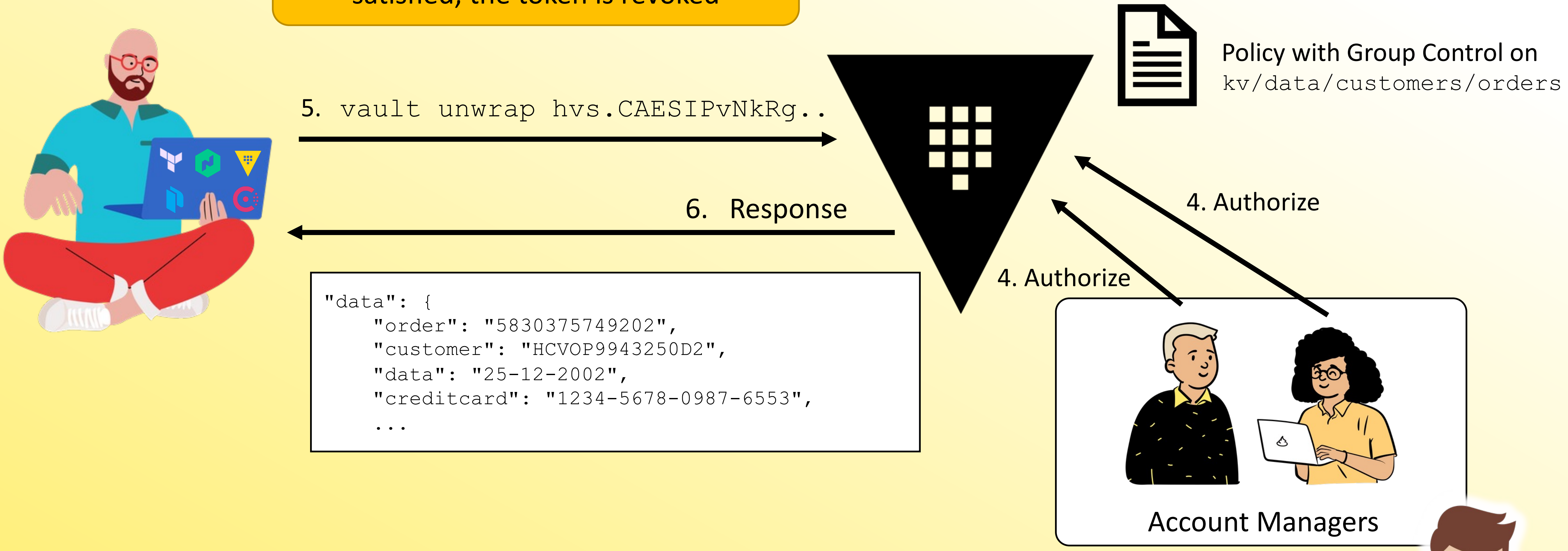
Policy with Group Control on
`kv/data/customers/orders`

Account Managers

3. Share Accessor with Managers for Approval

Control Group Workflow

Note: If the authorization can not be satisfied, the token is revoked



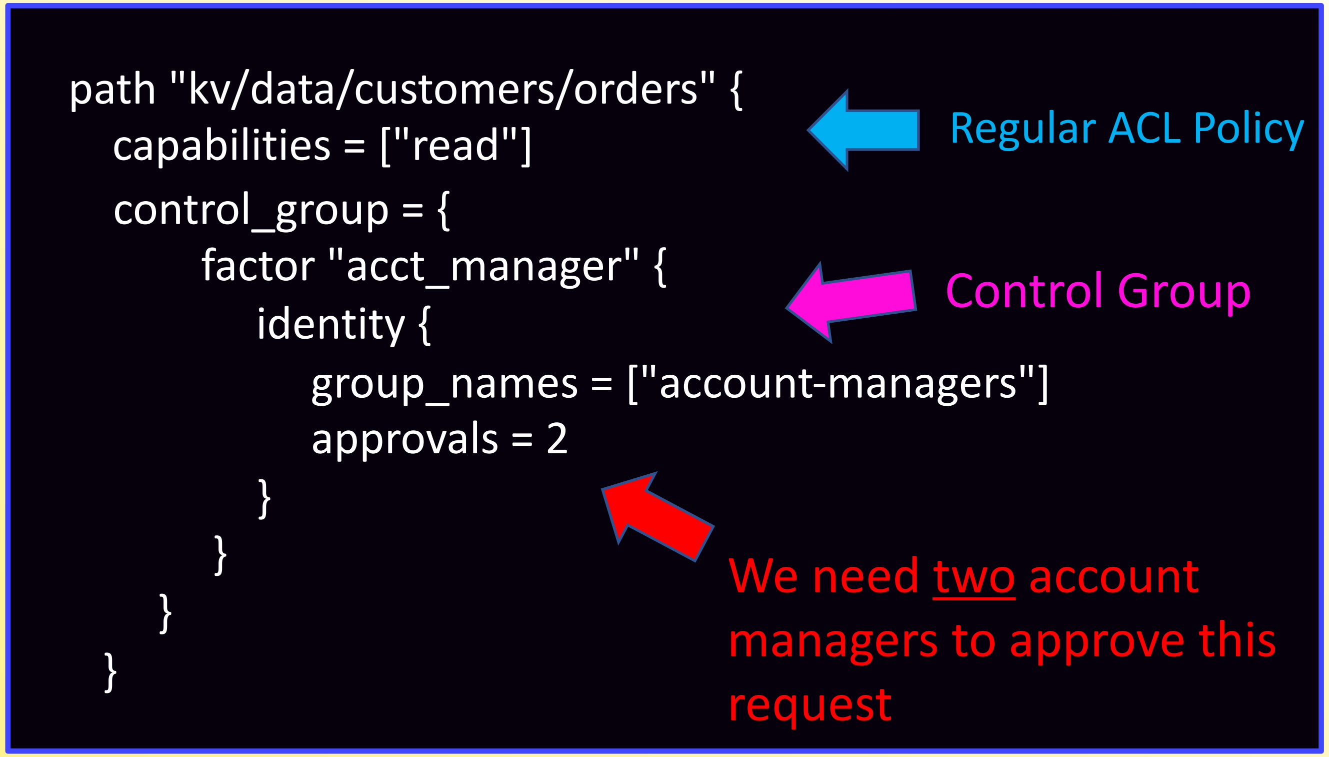
Policy with Group Control on
`kv/data/customers/orders`

5. `vault unwrap hvs.CAESIPvNkRg..`

6. Response

```
"data": {
    "order": "58303757492O2",
    "customer": "HCVOP9943250D2",
    "data": "25-12-2002",
    "creditcard": "1234-5678-0987-6553",
    ...
```

Control Groups in Vault Policies



```
path "kv/data/customers/orders" {
   capabilities = ["read"]
   control_group = {
         factor "acct_manager" {
             identity {
                 group_names = ["account-managers"]
                 approvals = 2
             }
         }
     }
 }
```

Regular ACL Policy

Control Group

We need two account managers to approve this request

Control Groups in Sentinel Policies (EGP)

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

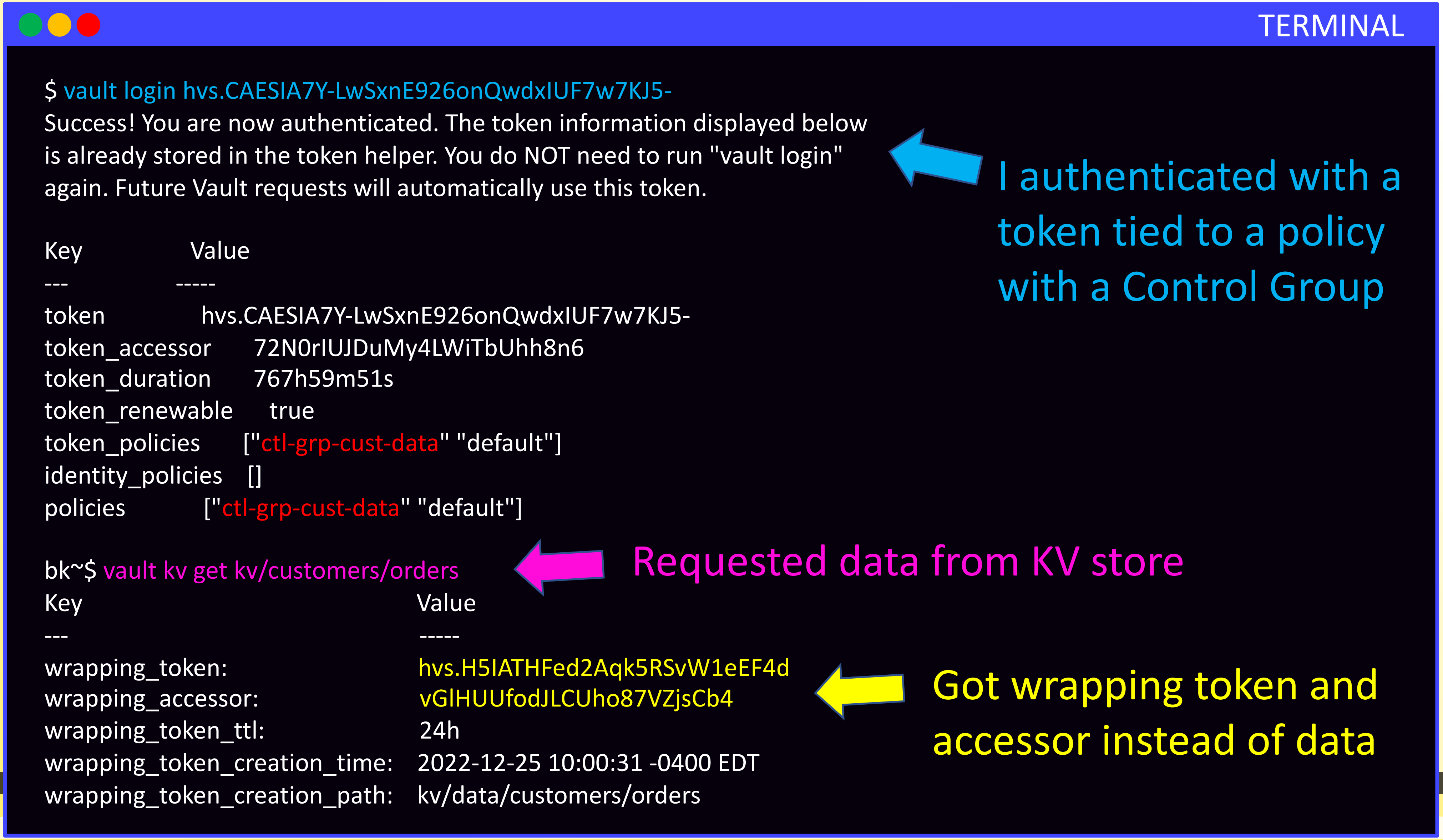
Deploy this EGP against kv/data/customers/orders

```
import "controlgroup"

control_group = func() {
   numAuthzs = 0
   for controlgroup.authorizations as authz {
       if "account-managers" in authz.groups.by_name {
          numAuthzs = numAuthzs + 1
       }
   }
   if numAuthzs >= 2 {
       return true
   }
   return false
}


main = rule {
    control_group()
}
```

We need two account managers to approve this request

Control Groups in Action (CLI)

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL



TERMINAL

```
$ vault login hvs.CAESIA7Y-LwSxnE926onQwdxIUF7w7KJ5-
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key              Value
---              -----
token              hvs.CAESIA7Y-LwSxnE926onQwdxIUF7w7KJ5-
token_accessor      72N0rIUJDuMy4LWiTbUhh8n6
token_duration      767h59m51s
token_renewable     true
token_policies      ["ctl-grp-cust-data" "default"]
identity_policies   []
policies            ["ctl-grp-cust-data" "default"]

bk~$ vault kv get kv/customers/orders
Key                              Value
---                              -----
wrapping_token:                  hvs.H5IATHFed2Aqk5RSvW1eEF4d
wrapping_accessor:               vGlHUUfodJLCUho87VZjsCb4
wrapping_token_ttl:              24h
wrapping_token_creation_time:    2022-12-25 10:00:31 -0400 EDT
wrapping_token_creation_path:    kv/data/customers/orders
```

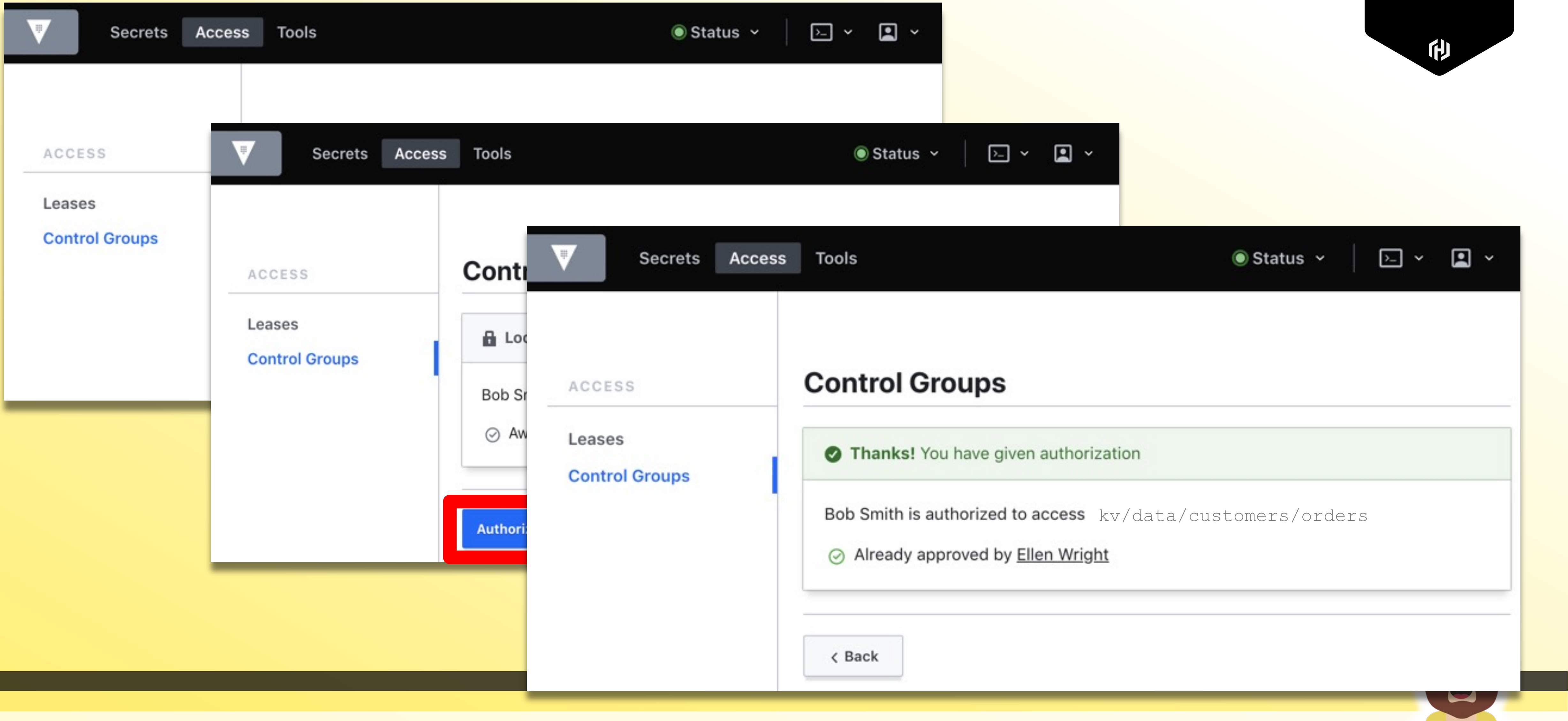
I authenticated with a token tied to a policy with a Control Group

Requested data from KV store

Got wrapping token and accessor instead of data

Authorizer Actions (Account Manager)

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

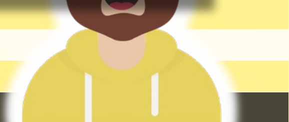


Control Groups

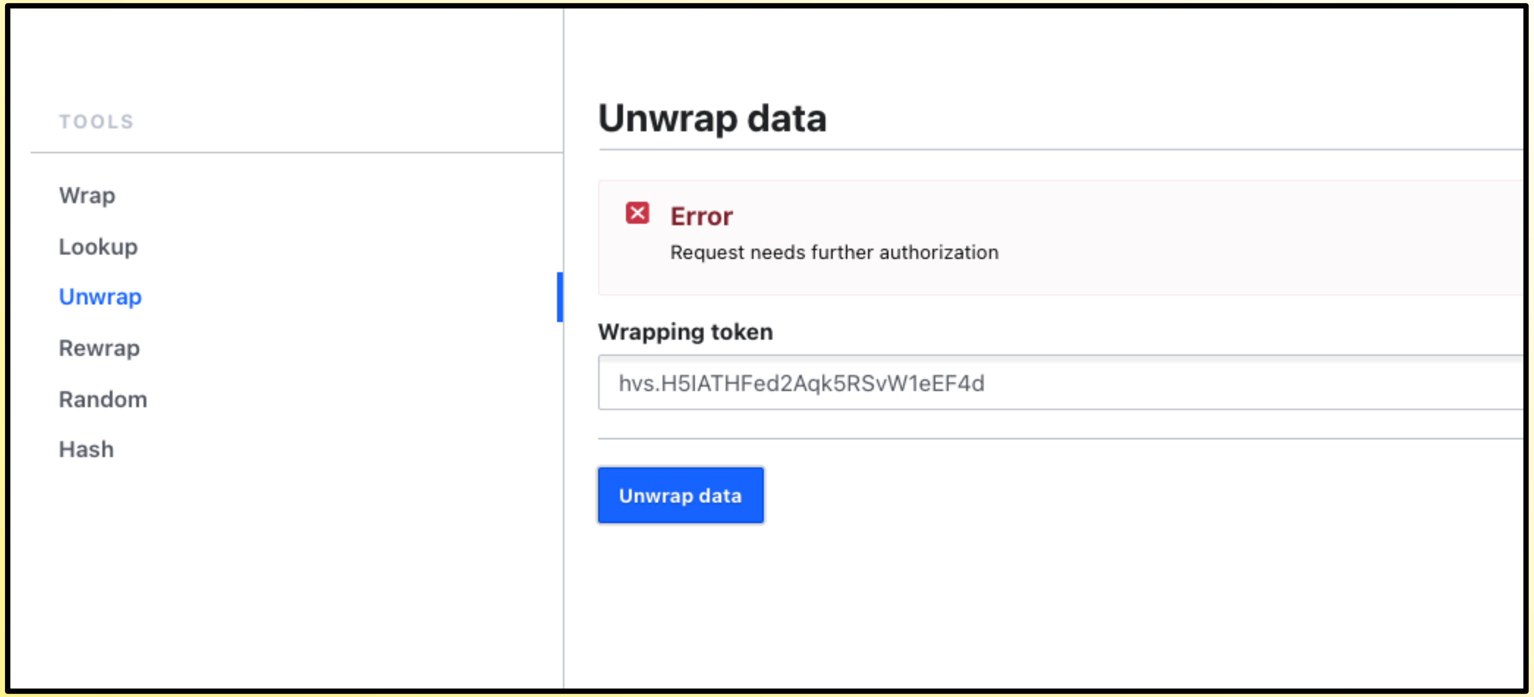
Thanks! You have given authorization

Bob Smith is authorized to access `kv/data/customers/orders`

Already approved by Ellen Wright



Not yet Authorized

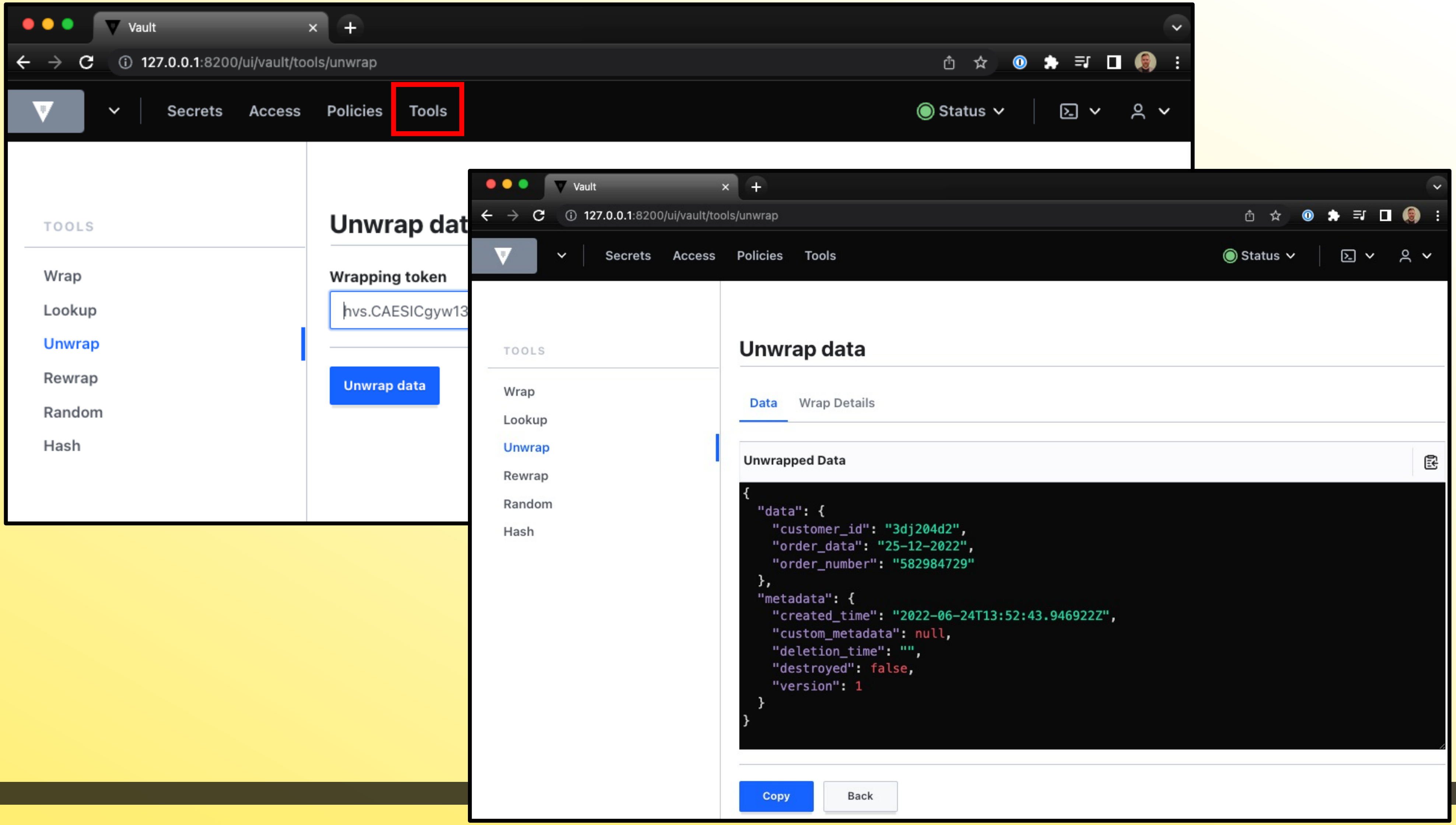


☒ Error
Request needs further authorization

Wrapping token: hvs.H5IATHFed2Aqk5RSvW1eEF4d

Unwrap the Secrets After Approvals

Vault
CERTIFIED OPERATIONS PROFESSIONAL



```
{
  "data": {
    "customer_id": "3dj204d2",
    "order_data": "25-12-2022",
    "order_number": "582984729"
  },
  "metadata": {
    "created_time": "2022-06-24T13:52:43.946922Z",
    "custom_metadata": null,
    "deletion_time": "",
    "destroyed": false,
    "version": 1
  }
}
```

Describe and Interpret Multi-Tenancy with Namespaces

What are Namespaces?

Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

- Allows organizations to provide “Vault as a Service”
  - Provides isolated environments on single Vault environment
  - Multi-tenant but centralized management
  - Allows delegation of Vault of responsibilities

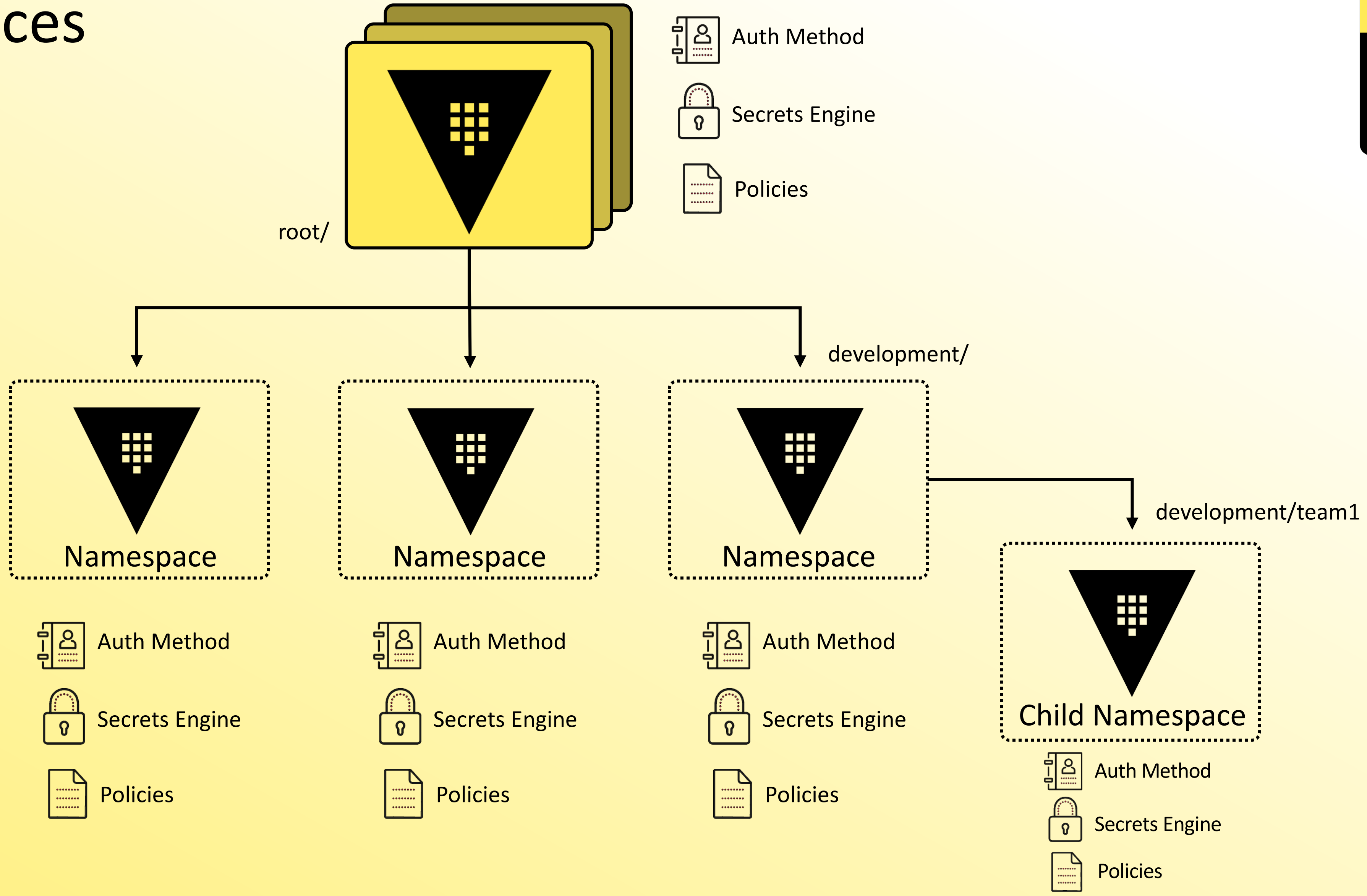
- Available in all versions of Vault Enterprise

- Each namespace can have its own:
  - Policies
  - Auth Methods
  - Secrets Engines
  - Tokens
  - Identities

What are Namespaces?

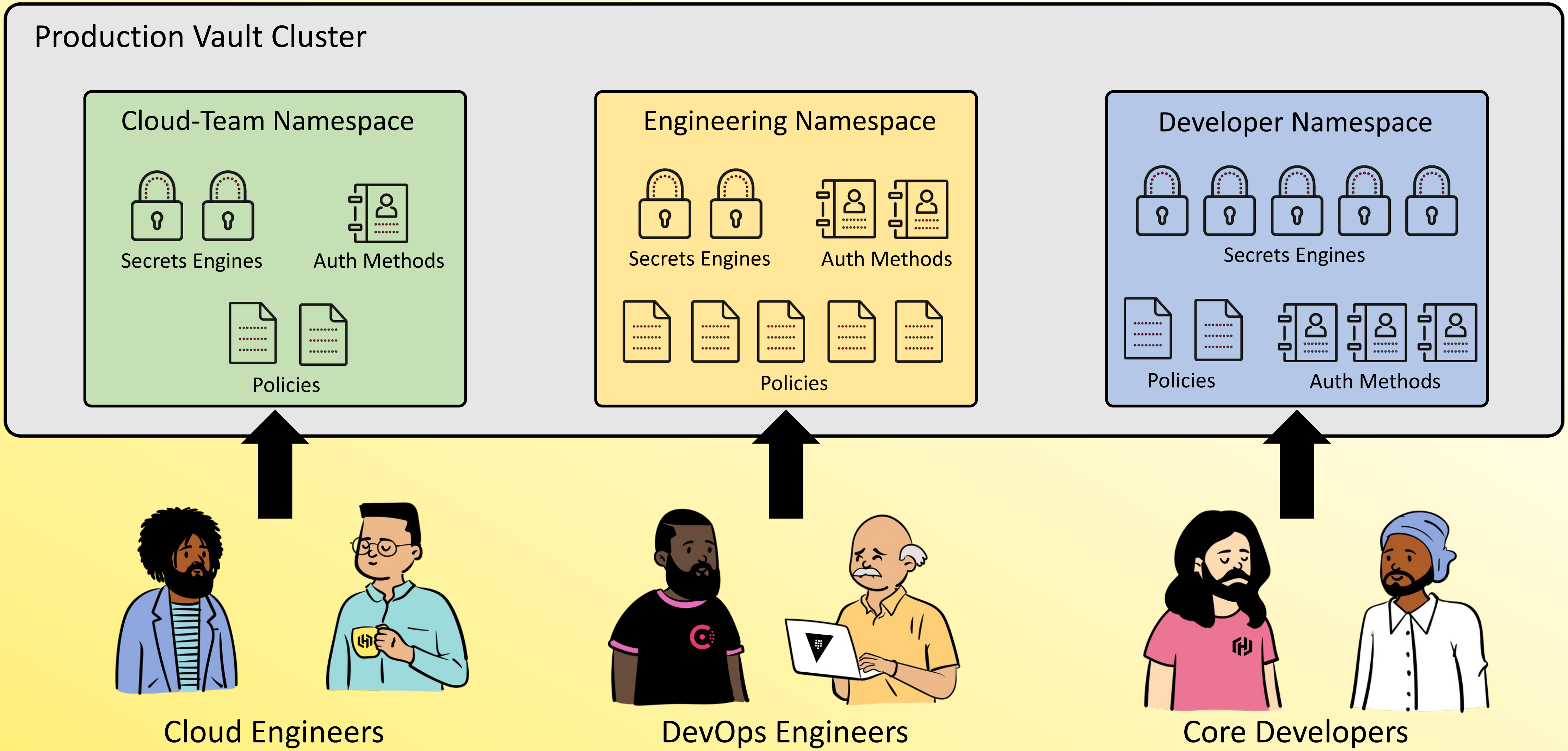
- The default namespace is ‘root’
- Namespaces are created in a hierarchical fashion
- Just like root, paths and ACLs are relative to the namespace, making easier to re-use commands and policies across multiple namespaces
- Tokens are only valid in a single namespace, but you can create an entity who has access to other namespaces

Namespaces

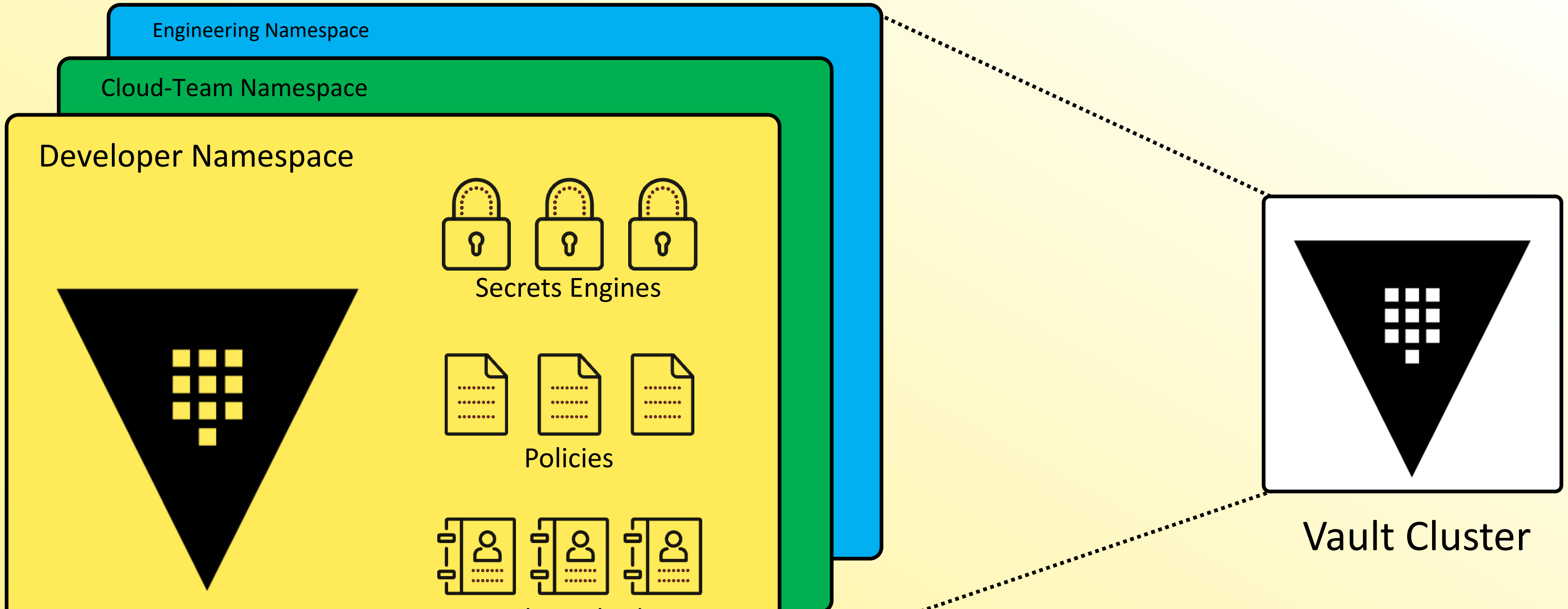


Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

Assigning Namespaces



Administrative Delegation

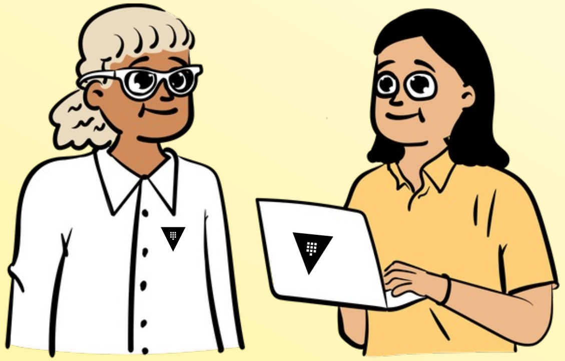


Developer Namespace Admin

Responsible for Dev:

- Secrets Engines
- Policies
- Auth Methods

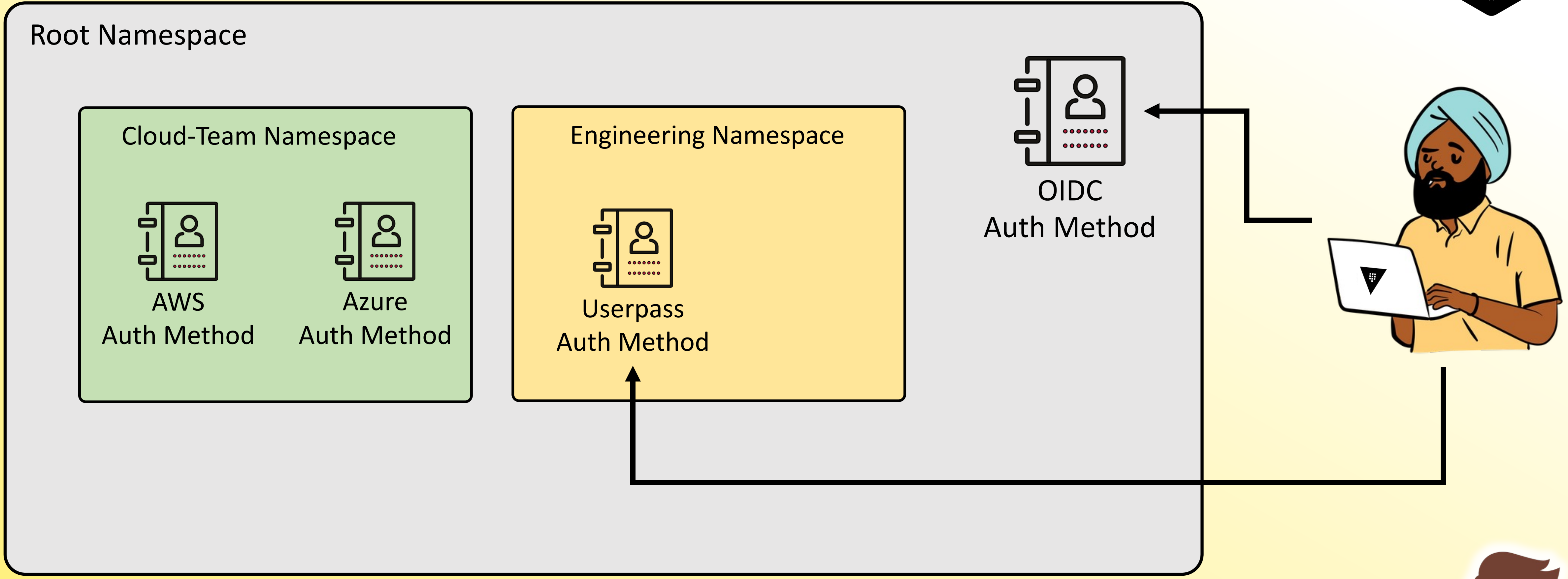
Vault Engineers



Responsible for:

- Cluster Nodes
- Audit Devices
- Root Namespace
- Storage Backend
- Vault Upgrades

Authenticating to Namespaces



Common Namespace Commands

Create Namespace

```
$   vault namespace create <namespace>
```

List Namespaces

```
$   vault namespace list
```

Delete a Namespace

```
$   vault namespace delete <namespace>
```

Using Namespaces on the CLI

Set Namespace Environment Variable – then run commands as normal

```
$  export VAULT_NAMESPACE=<namespace>
```

Reference a Namespace on the CLI when running a command

```
$  vault kv get -namespace=<namespace> kv/data/sql/prod
```

Referencing Namespaces in the API

Add the API Header = `X-Vault-Namespace`

```
curl \
 -header "X-Vault-Token: "hvs.a83b50ed2aa548212" \
 -header "X-Vault-Namespace: "development/" \
 -request GET \
 https://vault.hcvop.com:8200/v1/kv/data/sql/prod
```

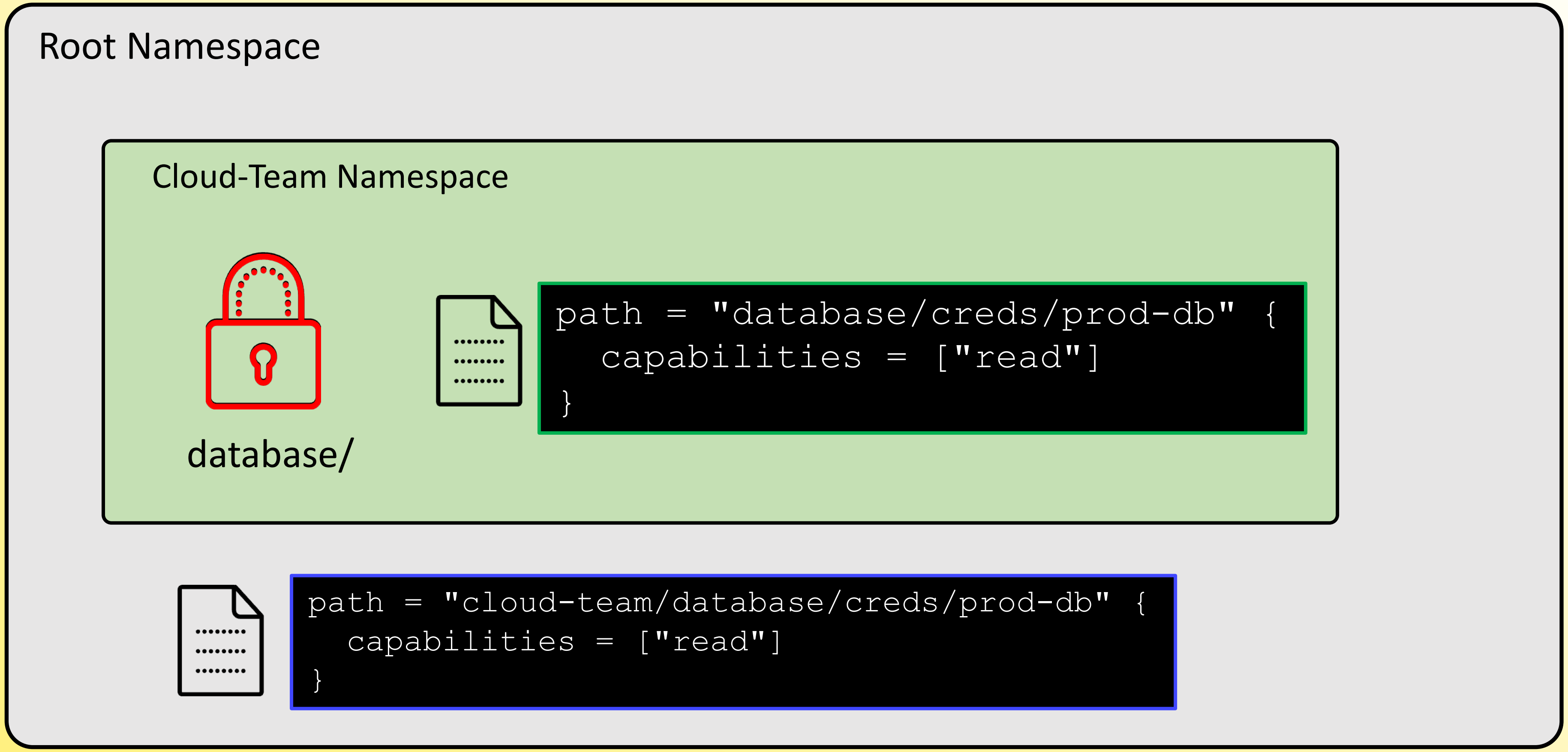
Referencing Namespaces in the API

Add the Namespace to the API Endpoint

```
curl \
 -header "X-Vault-Token: "hvs.CAESIA7Y-LwSxnE926onQwdxIUF7" \
 -request GET \
 https://vault.hcvop.com:8200/v1/development/kv/data/sql/prod
```

Writing Policies for Namespaces

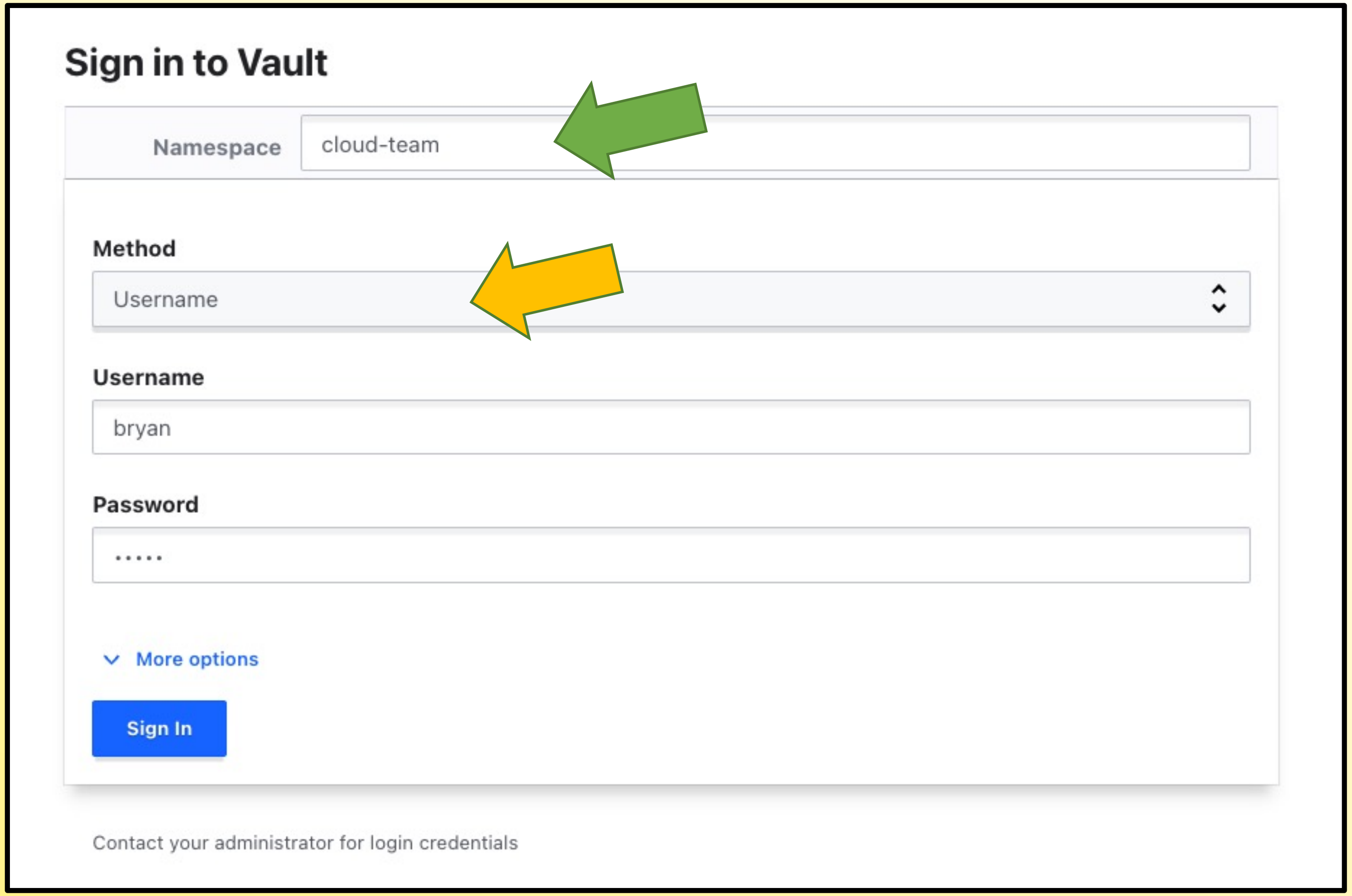
The path is relative to the Namespace



```
path = "database/creds/prod-db" {
  capabilities = ["read"]
}
```

```
path = "cloud-team/database/creds/prod-db" {
  capabilities = ["read"]
}
```

Authenticating to a Namespace via UI



Sign in to Vault

Namespace: cloud-team

Method: Username

Username: bryan

Password: ·····

Contact your administrator for login credentials

Authenticating to a Namespace via CLI

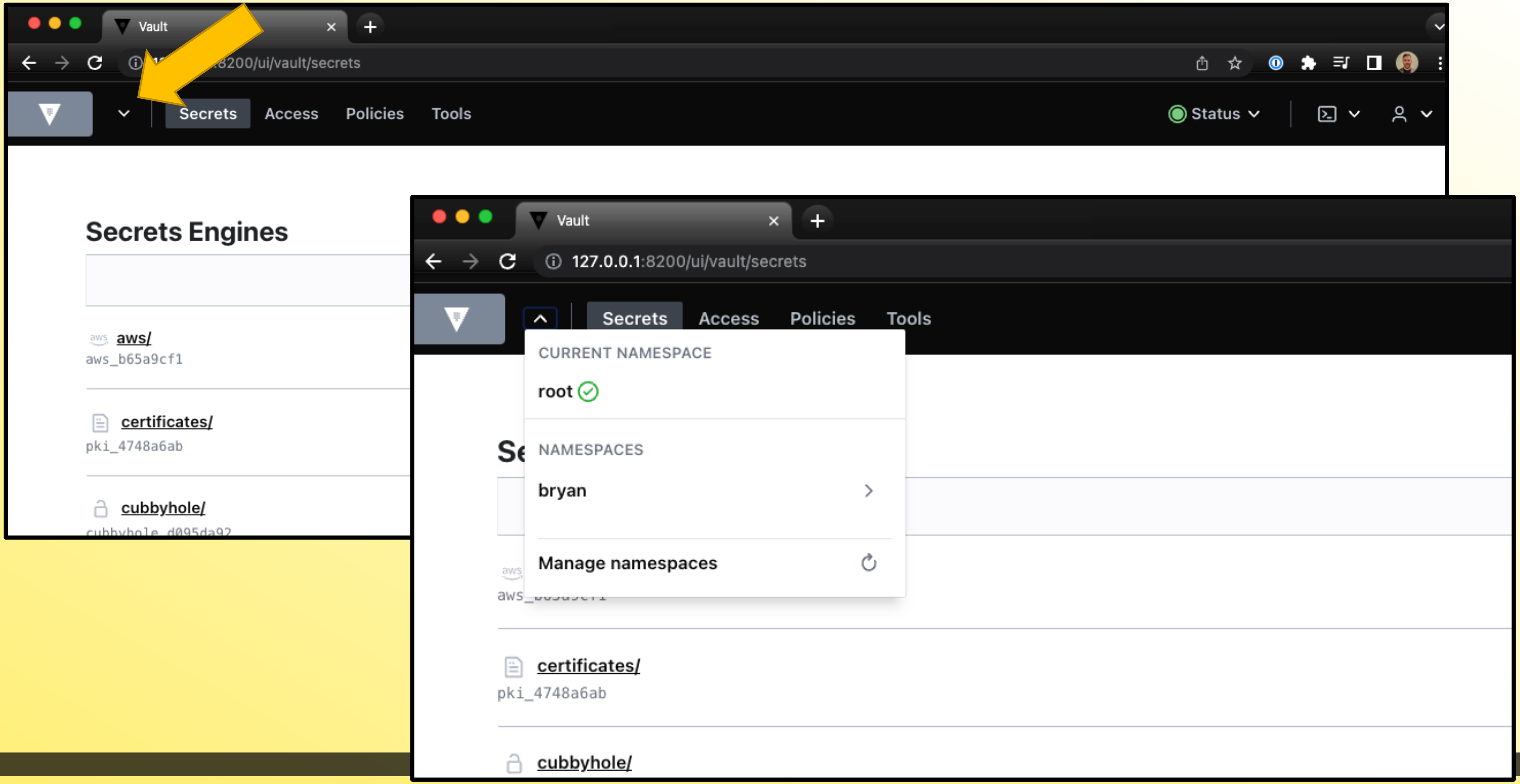
```
$ vault login -namespace=cloud-team -method=userpass username=bryan
Password (will be hidden):

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

Key                    Value
---                    -----
token
hvs.CAESIM5RikdMODs5nZrFrsecgqUKggrnXgSOZrkvXMtUXnwKGicKImh2cy5oOXlrNWFQRHNQM1Y4M
G5xZkF0VFB6dVcubjU3eTYQwAM
token_accessor         rOH7HYtHmZ6fDX4z0RCJVxbF.n57y6
token_duration         768h
token_renewable        true
token_policies         ["default"]
identity_policies      []
policies               ["default"]
token_meta_username    bryan
```

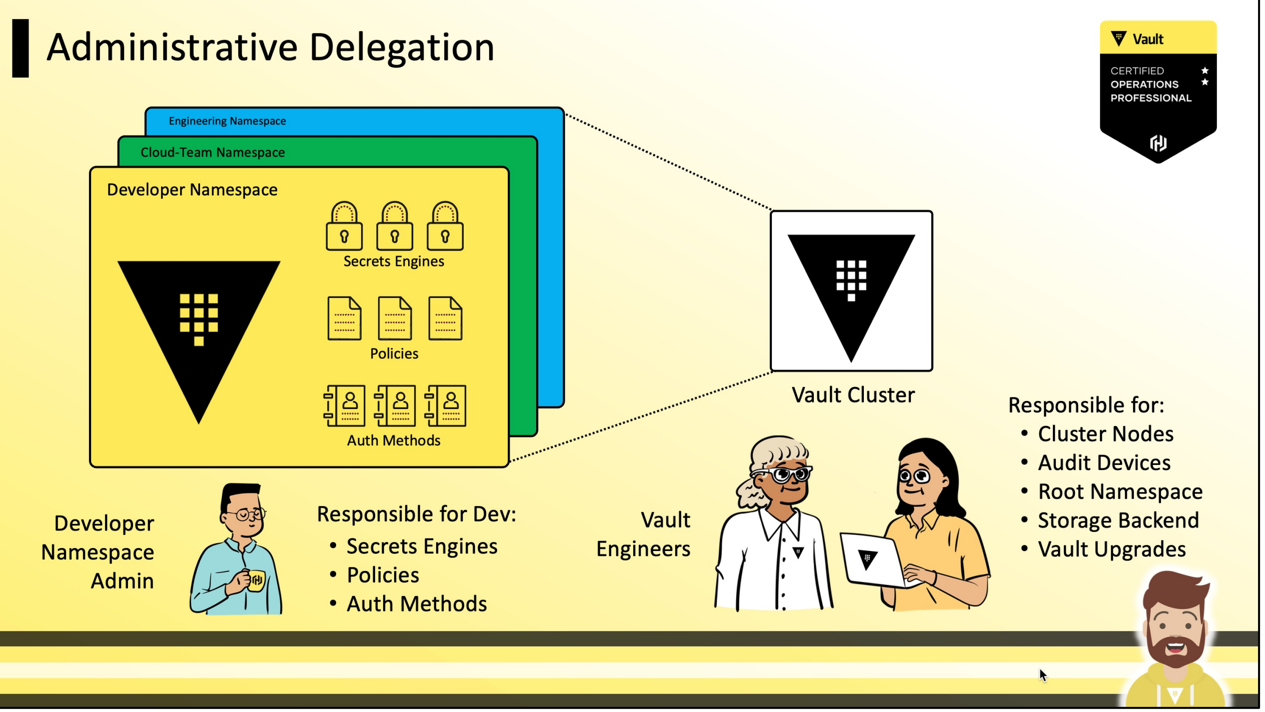
Enabling an Auth Method In a Namespace

```
$ vault namespace create cloud-team
Key       Value
---       -----
id        n57y6
path      cloud-team/

# Enable userpass auth method using the namespace flag
$ vault auth enable -namespace=cloud-team userpass
Success! Enabled userpass auth method at: userpass


# Enable aws auth method using environment variable
$ export VAULT_NAMESPACE=cloud-team
$ vault auth enable aws
```

Working with Namespaces in the UI

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL





Administrative Delegation

Vault CERTIFIED OPERATIONS PROFESSIONAL

Engineering Namespace

Cloud-Team Namespace

Developer Namespace

Secrets Engines

Policies

Auth Methods

Vault Cluster

Responsible for:

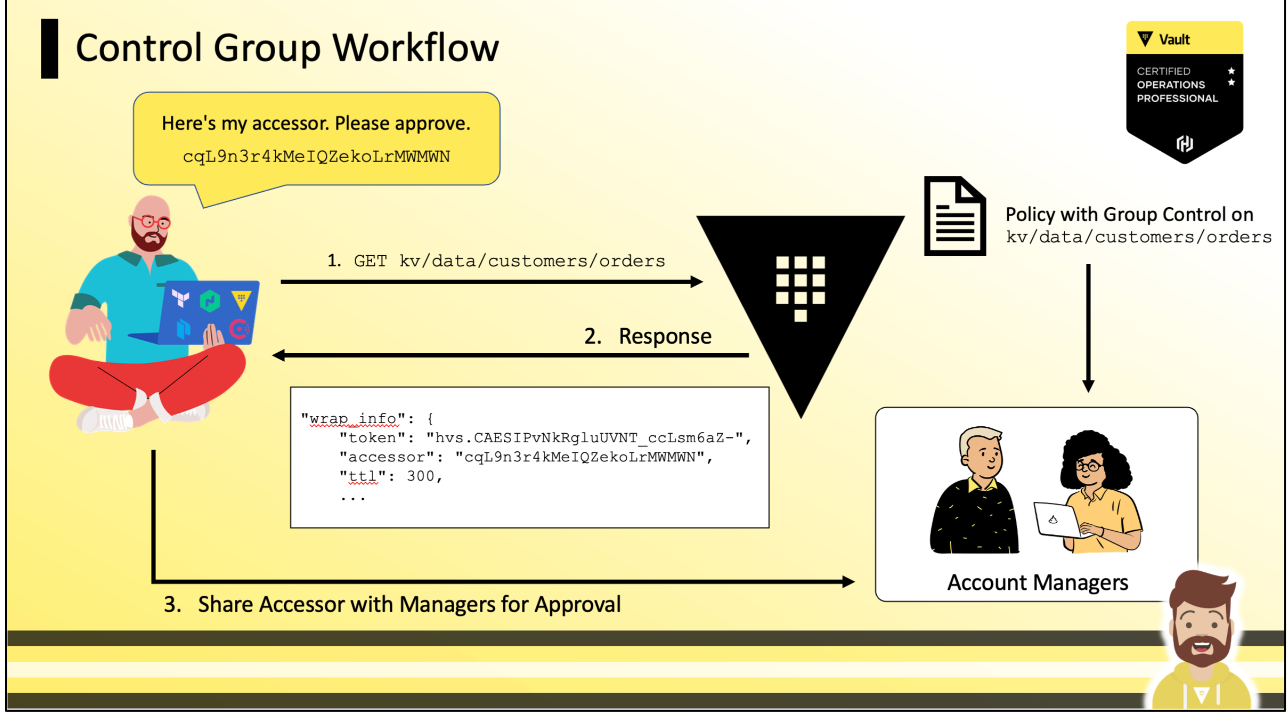
- Cluster Nodes
- Audit Devices
- Root Namespace
- Storage Backend
- Vault Upgrades

Developer Namespace Admin

Responsible for Dev:

- Secrets Engines
- Policies
- Auth Methods

Vault Engineers



Control Group Workflow

Vault CERTIFIED OPERATIONS PROFESSIONAL

1. GET kv/data/customers/orders

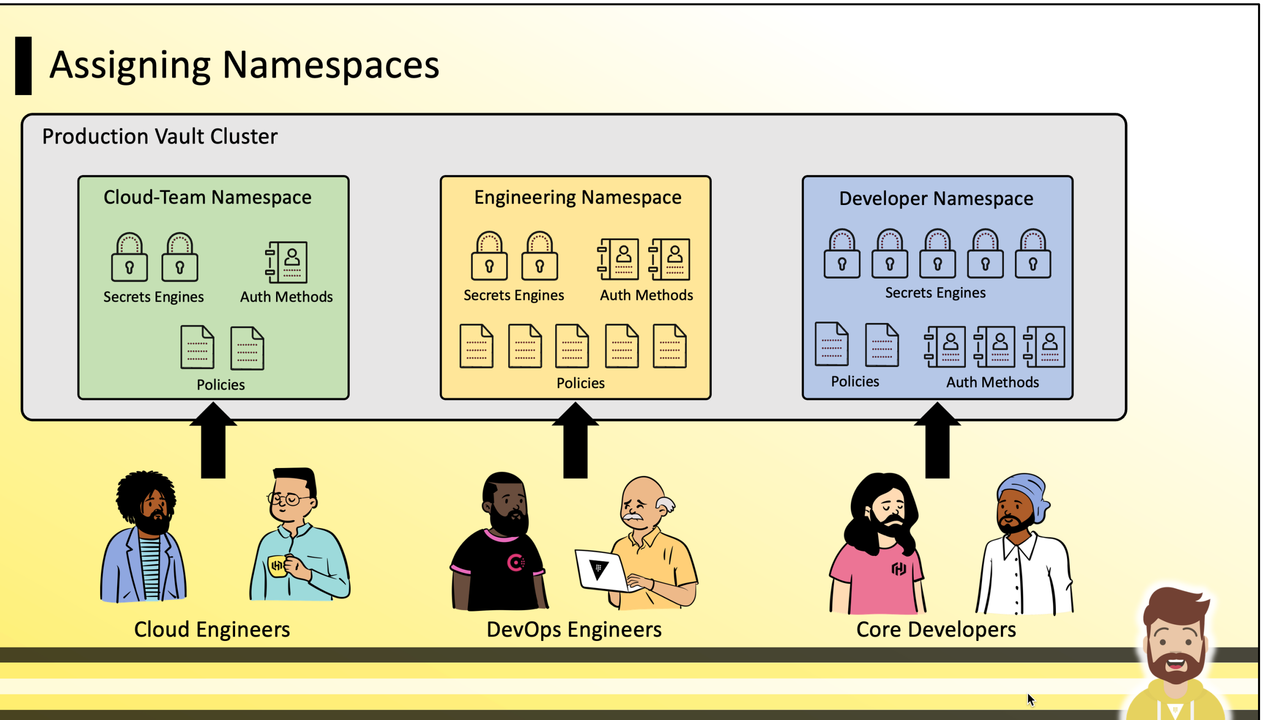
2. Response

```
"wrap_info": {
    "token": "hvs.CAESIPvNkRgluUVNT_ccLsm6aZ-",
    "accessor": "cqL9n3r4kMeIQZekoLrMWMWN",
    "ttl": 300,
    ...
```

Policy with Group Control on kv/data/customers/orders

Account Managers

3. Share Accessor with Managers for Approval



Assigning Namespaces

Production Vault Cluster

Cloud-Team Namespace: Secrets Engines, Auth Methods, Policies

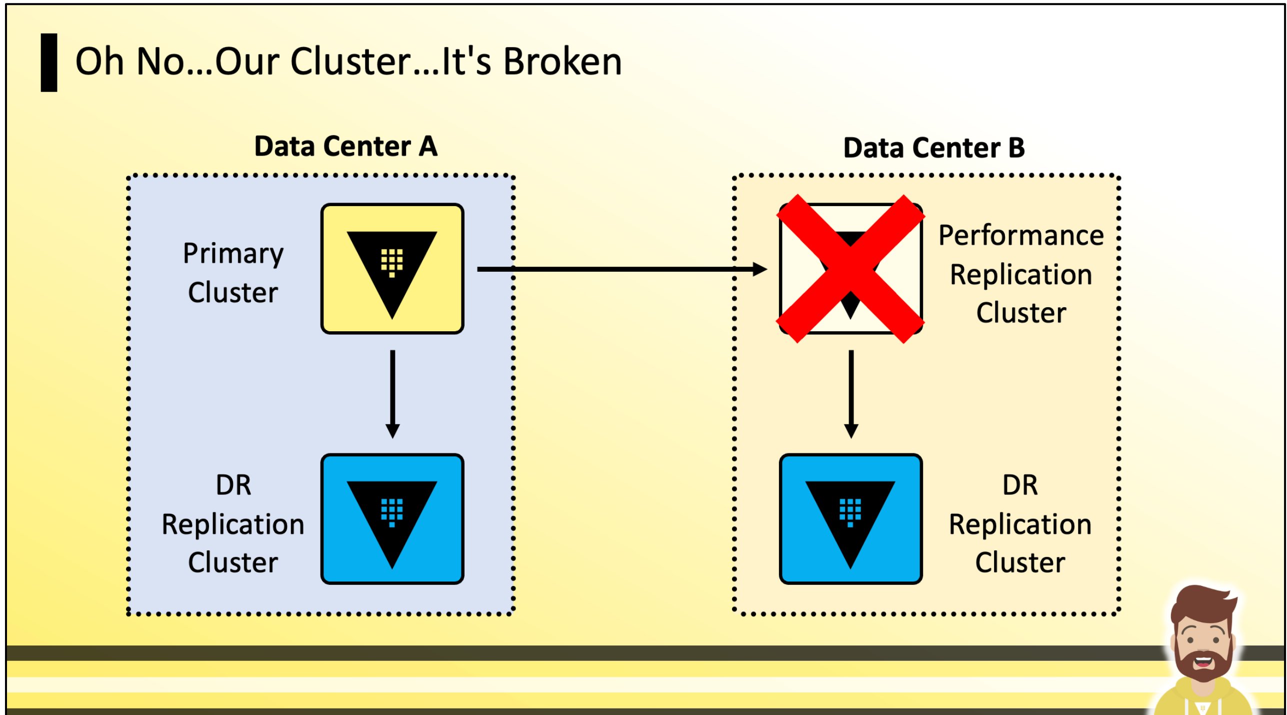
Engineering Namespace: Secrets Engines, Auth Methods, Policies

Developer Namespace: Secrets Engines, Policies, Auth Methods

Cloud Engineers

DevOps Engineers

Core Developers



Oh No...Our Cluster...It's Broken

Data Center A

Primary Cluster

DR Replication Cluster

Data Center B

Performance Replication Cluster

DR Replication Cluster