HashiCorp Vault

Deep Dive into Vault Policies

Pratik Khasnabis

HashiTalk ANZ 2022

August 11, 2022

Pratik Khasnabis

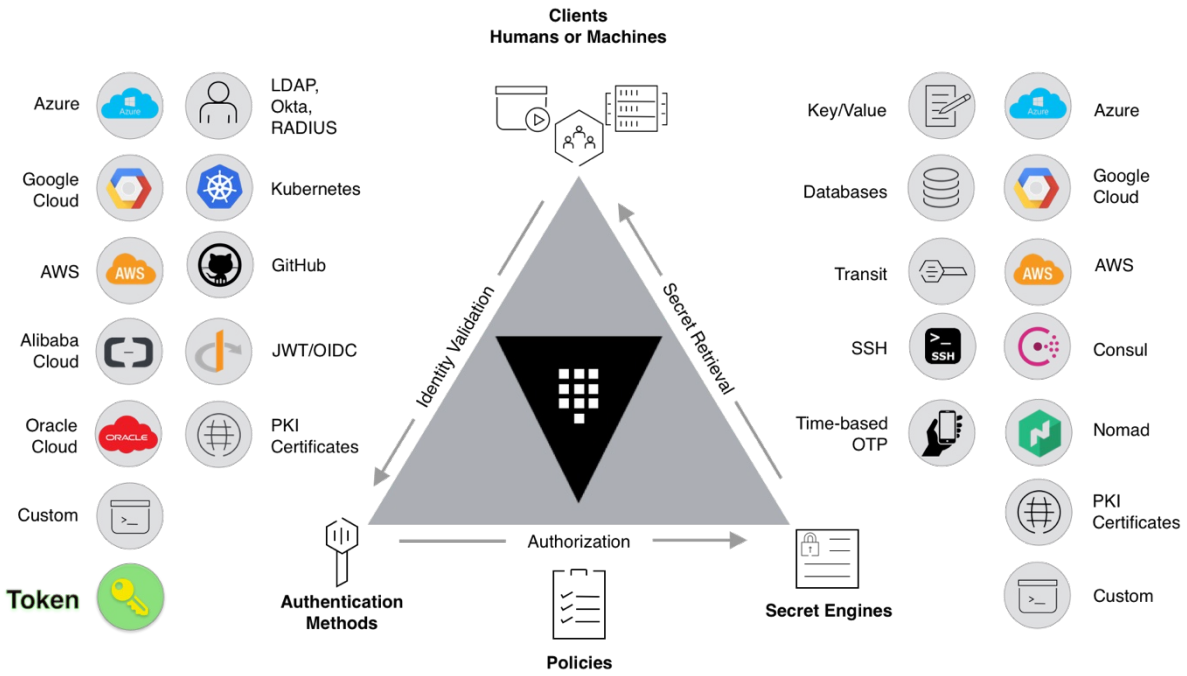
Senior Customer Success Architect
he/him

@softveda

Agenda

- Concept of polices and how it is tied to Authentication and Tokens in Vault.
- Rules to build a policy in Vault.
- Patterns to use Policies together with Identities.
- Finally, some advanced features like fine-grained policies.

Vault Overview



Vault is an **Identity-based** secret and encryption management system.

Identity vs Authn vs Authz

Identity

The credentials that uniquely identifies a user or an application. E.g. password or certificate.

Authentication

Is the user or application granted access to the system based on their verified credentials.

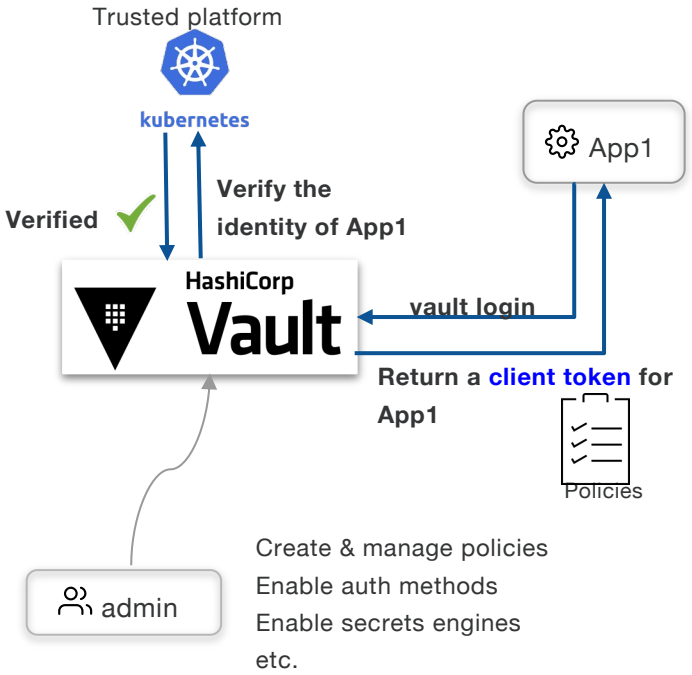
Authorization

What can an authenticated user or application do based on permissions.

Vault Policies and Client Tokens

How it all fits together

- Every Vault client must authenticate with Vault to acquire a **client token**
- The client token has **policies attached**
- Use the client token to invoke further Vault operations (e.g. read/update secrets or encrypt data)



Language of policies

- Policies are written in **HashiCorp Configuration Language** (HCL)
- Everything is **path**-based and corresponds to Vault API endpoints
  - Policies grant or deny access to certain **paths** and operations
- Empty policy grants **no permission**

Vault is **deny by default**

No policy = No authorization

Example policy

TERMINAL

```
path "secret/data/user/ravi" {
  capabilities = ["read"]
}
```

Read Secret Version

This endpoint retrieves the secret at the specified location.

Method	Path
GET	/secret/data/:path?version=:version-number

user/ravi

read capability

default engine path

Policies

Capability

capabilities
create
read
update
patch
delete
list
sudo
deny

HTTP Verbs
POST/PUT
GET
POST/PUT
PUT
DELETE
LIST

Policy HCL Syntax

Glob (*) & Wildcard (+)

- * Any character after the string. Only end of the path
- + Any character within a path segment
- Templates

Capabilities

- list, read
- create, update, patch, delete
- sudo, deny

Parameter Constraints

- allowed_parameters
- required_parameters
- denied_parameters

Response Wrapping

- min_wrapping_ttl
- max_wrapping_ttl

Policy Requirements

- All users should have access to their own secrets at path **user/<user>/***
- Users can authenticate using **userpass** and **ldap** auth methods and should have same policies regardless of their auth method.
- All users should have access to the their team's secret at path **team/<team>/***
- The team for the user is based on the LDAP group the user is in.
- Restrict encryption operation options and parameters for users.

Built-in policies

root

root user

Cannot be modified or removed. Any user associated with this policy becomes a root user. A root user can do *anything* within Vault. As such, it is **highly recommended** that you revoke any root tokens before running Vault in production.

default

Attached to all tokens

Cannot be removed. By default, it is attached to all tokens, but may be explicitly excluded at token creation time by supporting authentication methods. The policy contains basic functionality such as the ability for the token to look up data about itself and to use its cubbyhole data

Vault CLI

How to list, read and write policies

CODE EDITOR

```
$ vault policy --help
```

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

list all

```
$ vault policy read default
...
```

show

```
$ vault policy write apps-policy apps-policy.hcl
```

create or update policy

Vault CLI

Useful commands

CODE EDITOR

```
$ vault token lookup –format=json | jq
.data.policies
[
  "root"
]
```

```
Vault kv get –output-policy secret/user/ravi

path "secret/data/user/ravi" {
  capabilities = ["read"]
}
```

Example of path duplication

TERMINAL

```
path "secret/data/users/ravi/db" {
  capabilities = [ "read" ]
}

path "secret/data/users/ravi/aws" {
  capabilities = [ "read" ]
}
```

Paths Hierarchy

The "*" in path allows any **descending paths** but it does NOT include the top-level

CODE EDITOR

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

# secrets at secret/data/ravi
path "secret/data/ravi" {
   capabilities = ["create", "read", "update", "delete",
                   "list"]
}
```

Glob

*

operator

TERMINAL

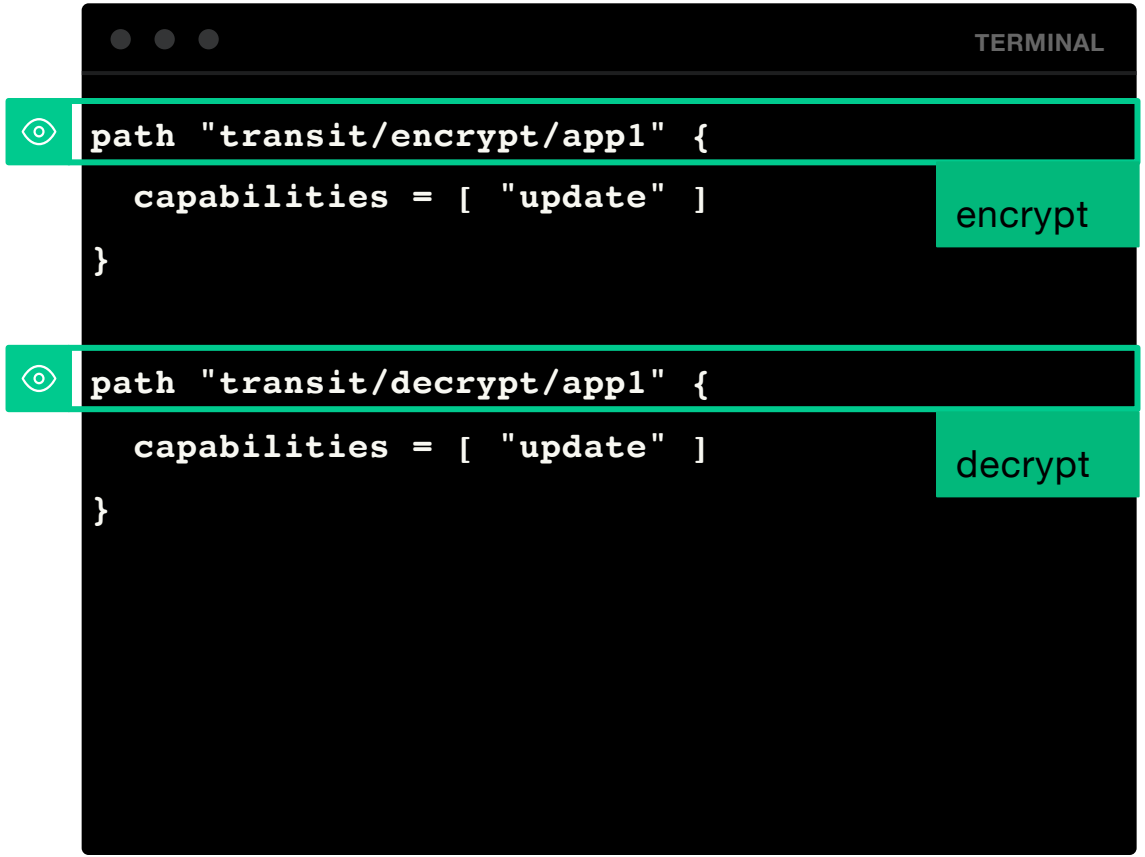
```
path "secret/data/users/ravi/db" {
  capabilities = [ "read" ]
}

path "secret/data/users/ravi/aws" {
  capabilities = [ "read" ]
}

path "secret/data/users/ravi/*" {
  capabilities = [ "read" ]
}
```

The * must be at the **end of the path**. It can be `secret/data/users/ravi/db-*`

Example of path duplication



Wildcard + operator

TERMINAL

```
path "transit/encrypt/app1" {
  capabilities = [ "update" ]
}

path "transit/decrypt/app1" {
  capabilities = [ "update" ]
}

path "transit/+/app1" {
  capabilities = [ "update" ]
}
```

The + can be used to denote any number of characters within a **single path segment**.

Example of using both + and *

A policy for DBAs to read every user's DB secrets

TERMINAL

```
path "secret/data/users/+/db-" {
  capabilities = [ "read" ]
}
```

Deny

The "**deny**" capability takes the precedence regardless of any other capabilities

CODE EDITOR

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

# Deny any operation on sensitive secrets
path "secret/data/users/engineering/sensitive" {
    capabilities = ["deny"]
}
```

Policy Priorities

Paths with wildcard

- **Rule:** The most-specific (granular) policy rules takes the priority
- What about the following?

```
path "secret/data/+/webapp/*"
{
  ...
}
```

Higher priority

vs.

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

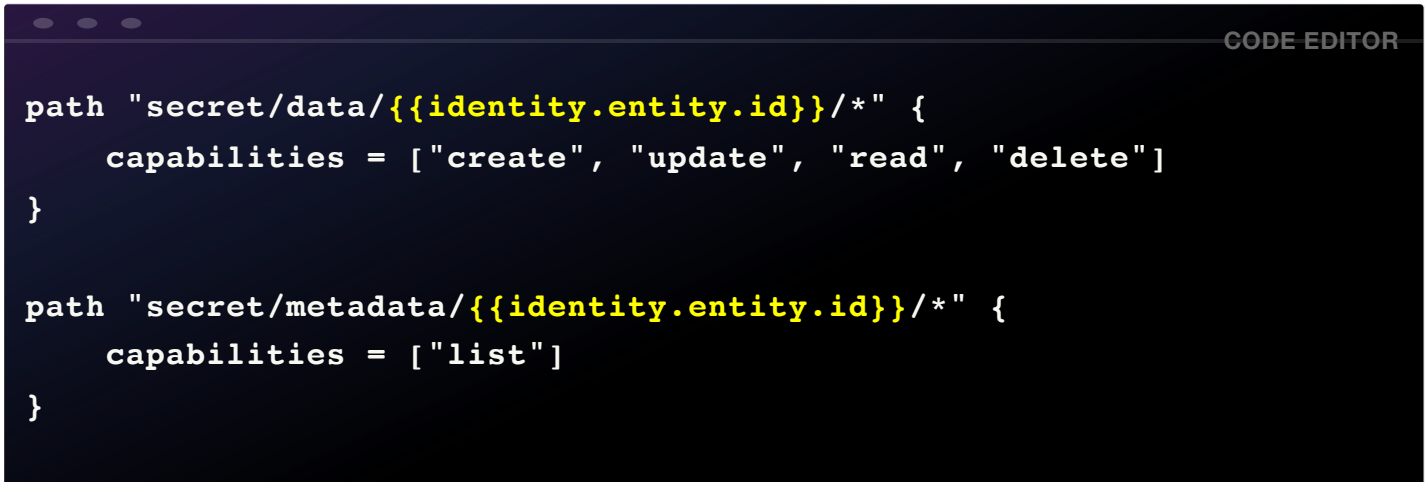
```
path "secret/+/+/webapp/*" {
  ...
}
```

vs.

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

Higher priority

ACL Templating



- Use variable replacement in policy path with values available to the token
- Define templates in double curly braces: **{{<parameter>}}**

ACL Templating with Identity Entity Names

CODE EDITOR

```
path "transit/encrypt/{{identity.entity.name}}" {
   capabilities = [ "update" ]
}

path "transit/decrypt/{{identity.entity.name}}" {
   capabilities = [ "update" ]
}
```

If the app name and key name do not match, you can store the key name as a metadata → {{identity.entity.**metadata.key_name**}}

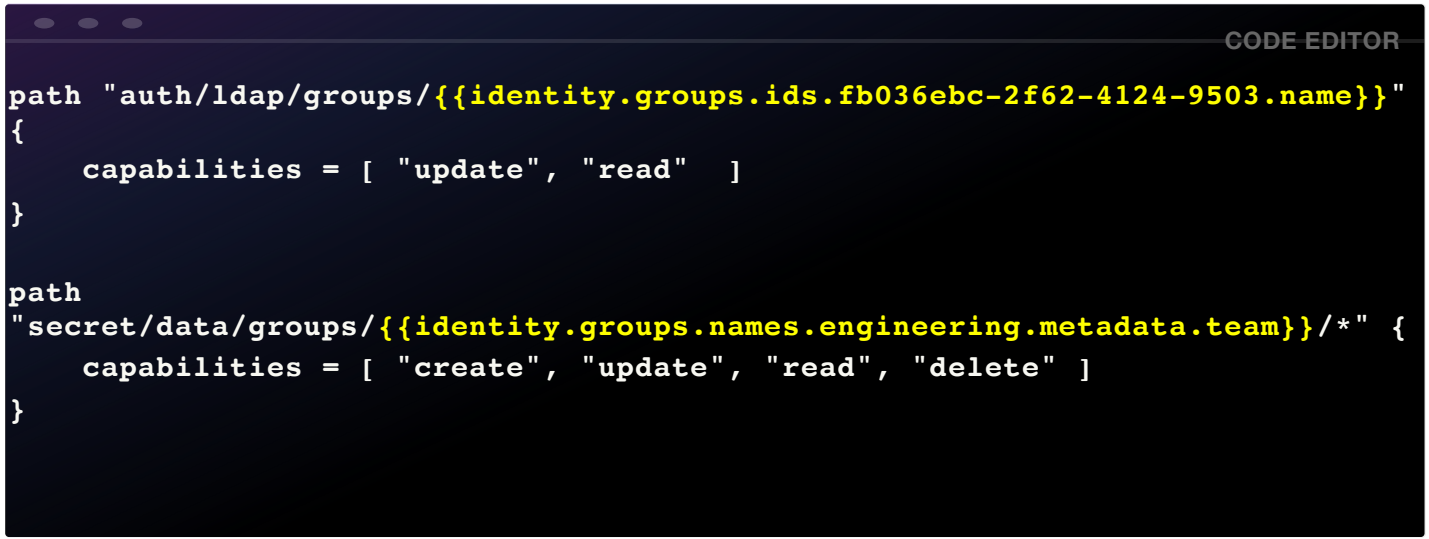
Available Templating Parameters (1 of 2)

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

Available Templating Parameters (2 of 2)

PARAMETER	DESCRIPTION
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

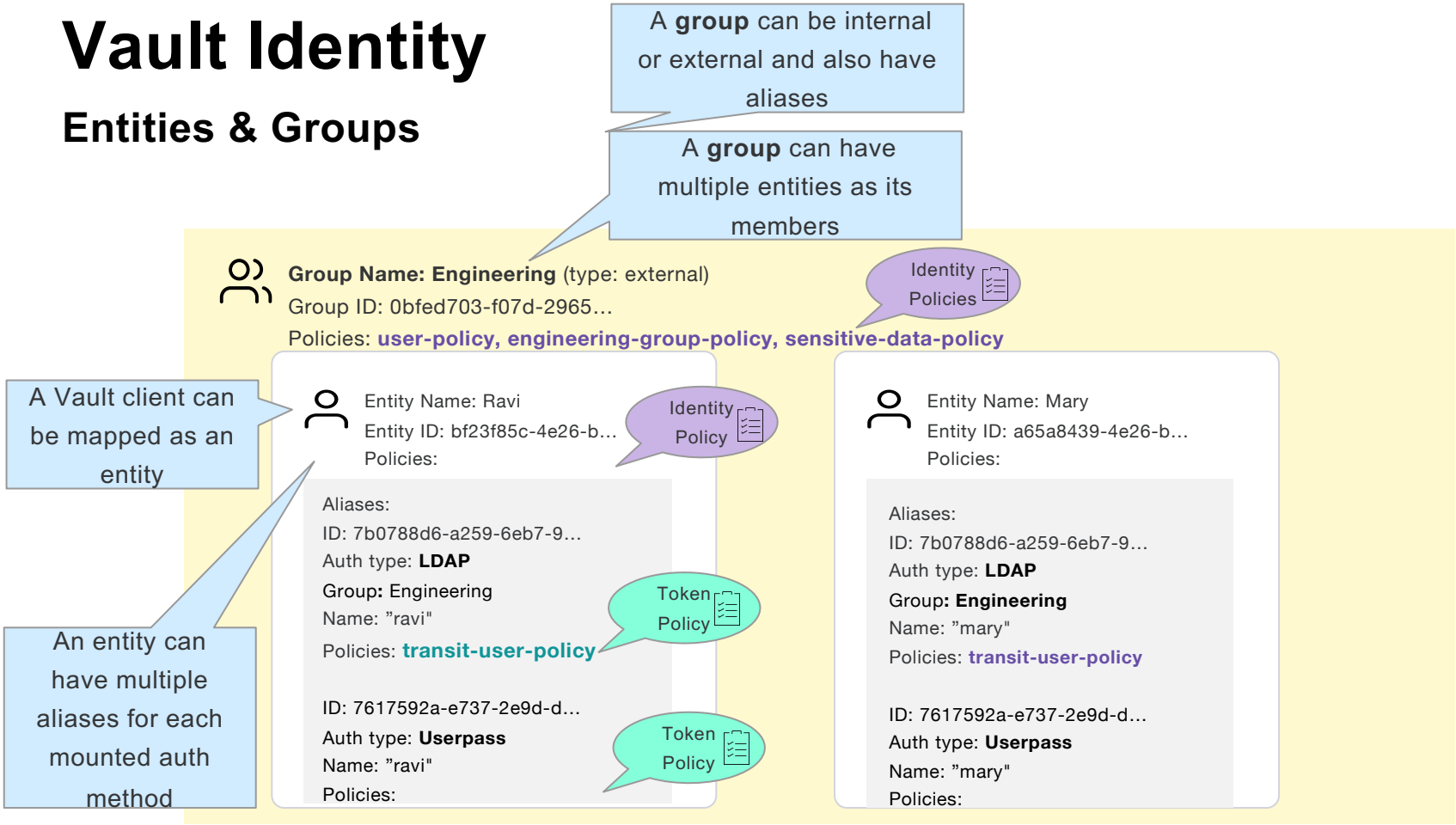
ACL Templating with Identity Groups



- Identity **groups** are not directly attached to a token and an **entity** can be associated with multiple groups
- To reference a group, the **group ID** or **group name** must be provided

Vault Identity

Entities & Groups



Key Takeaways:

Use Policies for access control
Create entities and groups
Use Policy paths templates
Assign policies to entities and groups

Thank You

hello@hashicorp.com
www.hashicorp.com