



Deep Dive into Vault Policies

Pratik Khasnabis

HashiTalk ANZ 2022

August 11, 2022

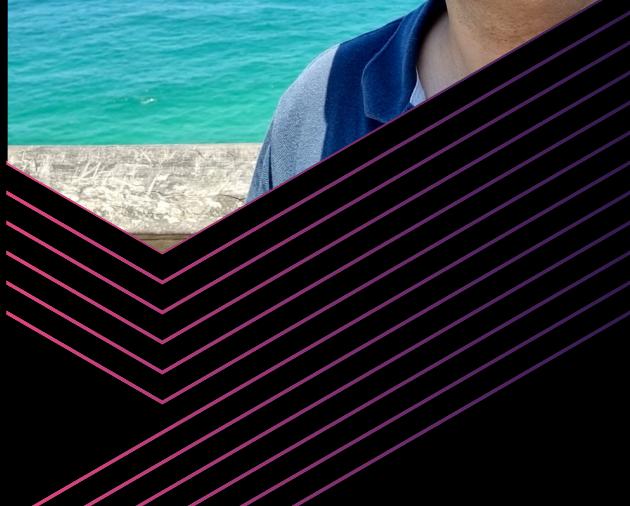
Copyright © 2021 HashiCorp



Pratik Khasnabis

Senior Customer Success Architect
he/him

@softveda

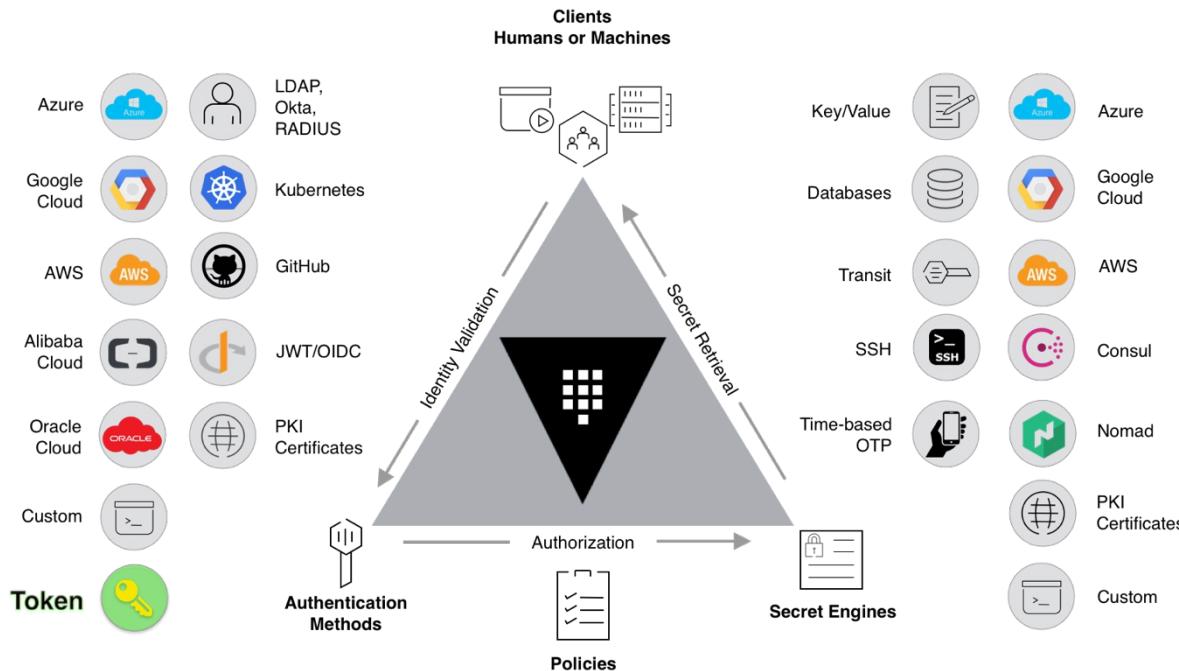


Agenda



- Concept of policies 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

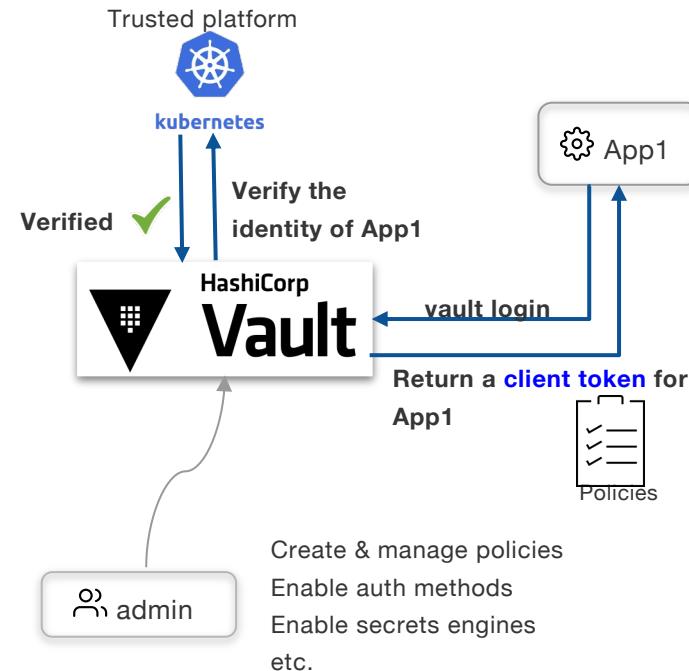
What can an authenticated user or application do based on permissions.



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

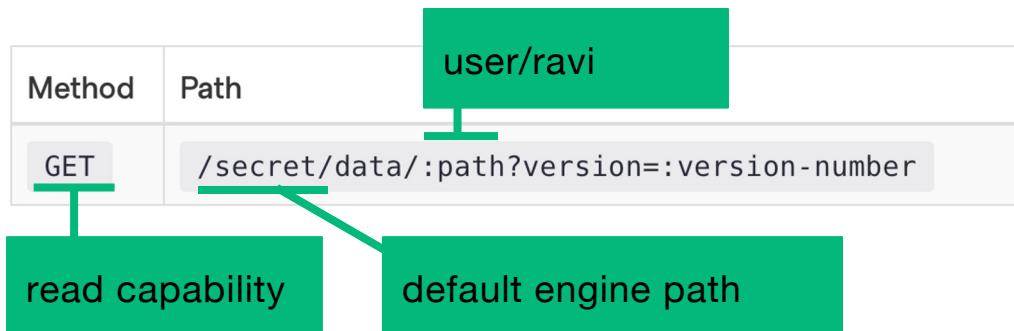


Example policy

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

Read Secret Version

This endpoint retrieves the secret at the specified location.





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

Parameter Constraints

- allowed_parameters
- required_parameters
- denied_parameters

Capabilities

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

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 **Idap** 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

The image shows a terminal window with two examples of Vault CLI commands. The first example uses the `vault token lookup` command to retrieve policies from the token's metadata. The second example uses the `Vault kv get` command with an output policy to set capabilities for a specific path.

```
$ vault token lookup --format=json | jq .data.policies
```

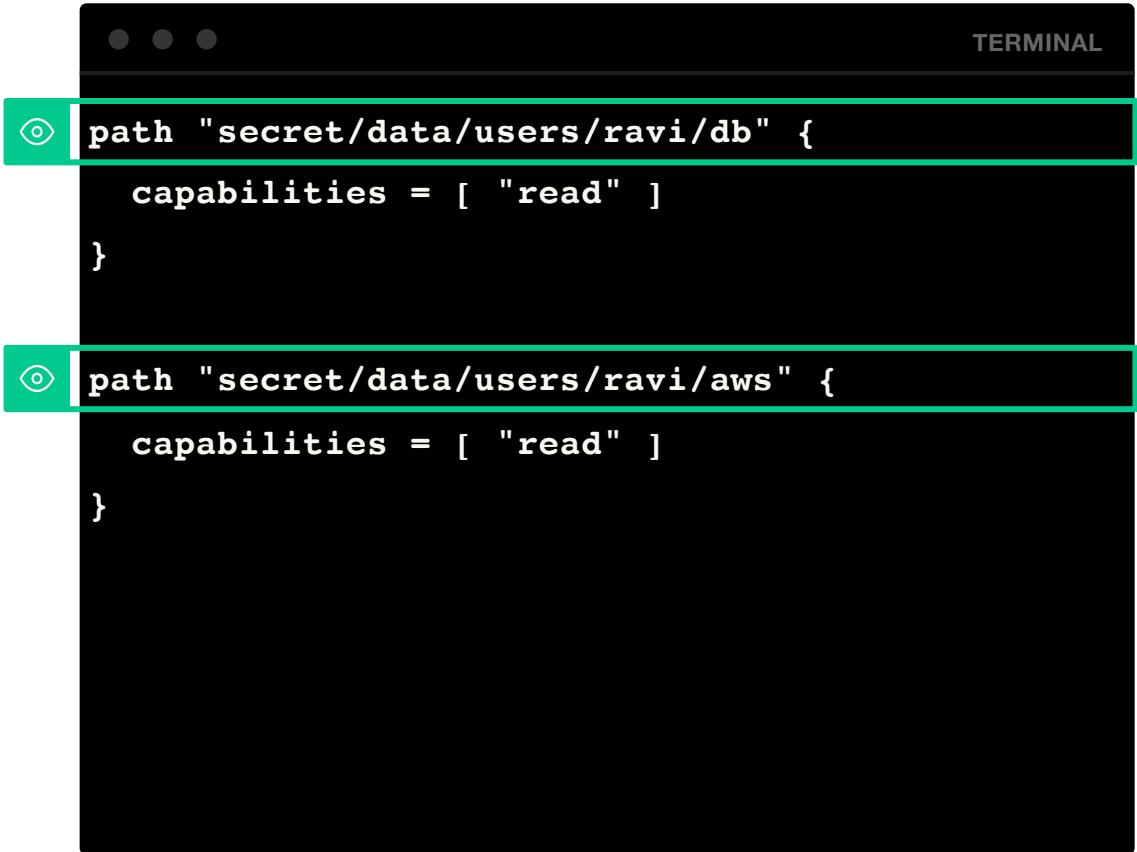
```
[  
  "root"  
]
```

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

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



Example of path duplication



The image shows a terminal window with two configuration snippets. The top snippet defines a path for a database:

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

The bottom snippet defines a path for AWS credentials:

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

The terminal window has a dark background with light-colored text. The code blocks are highlighted with green borders. The word "TERMINAL" is visible in the top right corner.



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
① path "secret/data/ravi/*" {
    capabilities = ["create", "read", "update", "delete",
                     "list"]
}

# secrets at secret/data/ravi
② path "secret/data/ravi" {
    capabilities = ["create", "read", "update", "delete",
                     "list"]
}
```



Glob

*

operator

The * must be at the **end of the path**. It can be

secret/data/users/ravi/db-*

TERMINAL

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

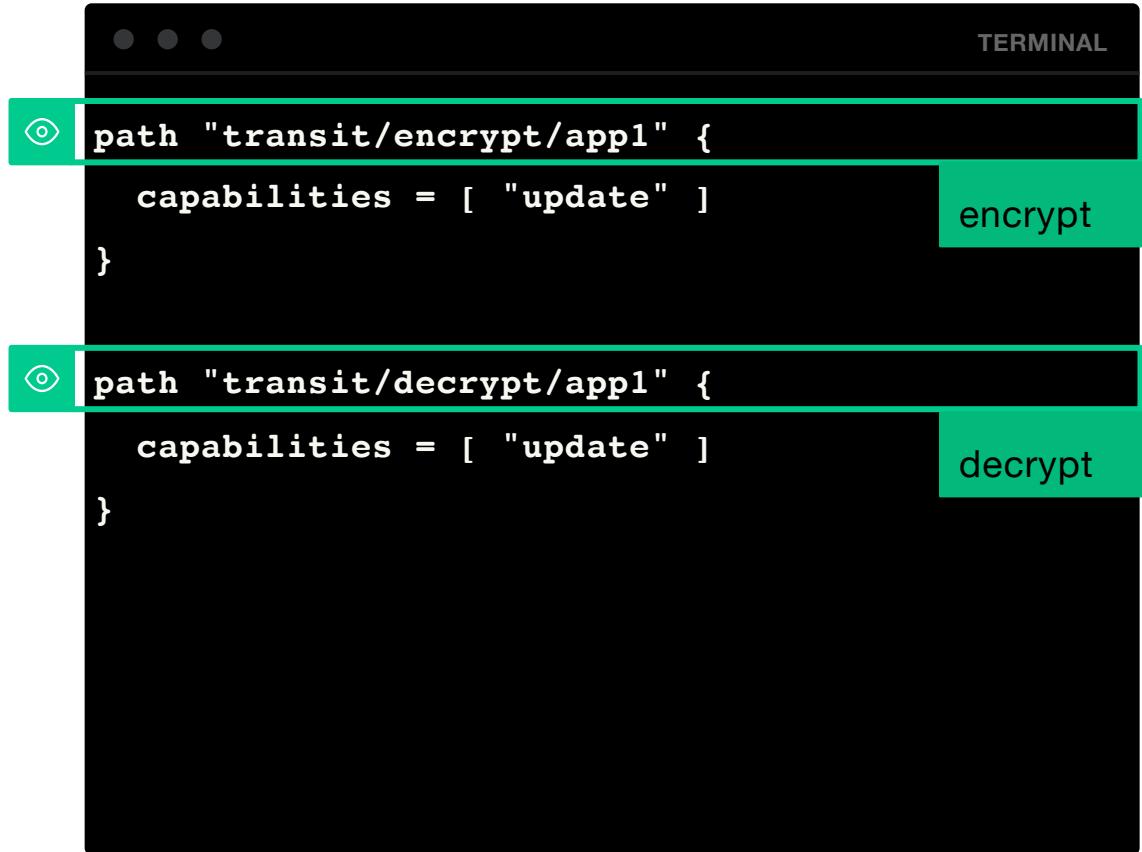
```
path "secret/data/users/ravi/aws" {  
    capabilities = [ "read" ]  
}
```

⑥

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



Example of path duplication



The image shows a terminal window with two configuration snippets. The top snippet is for the 'encrypt' operation, and the bottom snippet is for the 'decrypt' operation. Both snippets define a path 'transit/encrypt/app1' or 'transit/decrypt/app1' with a single capability 'update'.

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

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

The word 'encrypt' is highlighted in a green box next to the second snippet, and the word 'decrypt' is highlighted in a green box next to the fourth snippet.



Wildcard operator

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

```
path "transit/encrypt/app1" {  
    capabilities = [ "update" ]  
}  
  
path "transit/decrypt/app1" {  
    capabilities = [ "update" ]  
}  
  
⑥ path "transit/+/app1" {  
    capabilities = [ "update" ]  
}
```

TERMINAL



Example of using both + and *

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

The terminal window shows a JSON configuration snippet:

```
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?

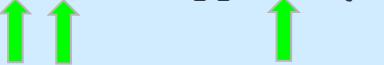
```
path "secret/data/+/*" {  
    ...  
}
```



vs.

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

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



vs.

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



ACL Templating



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

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

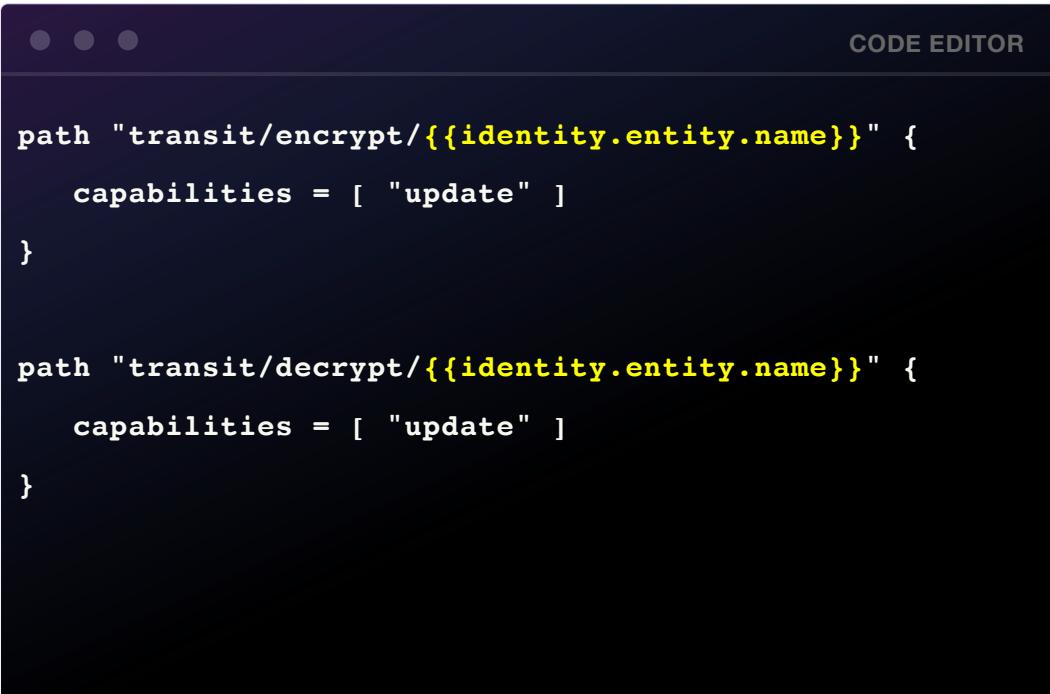
- Use variable replacement in policy path with values available to the token
- Define templates in double curly braces: **{{<parameter>}}**



ACL

Templating with Identity Entity Names

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



The image shows a dark-themed code editor window titled "CODE EDITOR". It displays two lines of configuration code. Both lines contain the path "transit/encrypt/{{identity.entity.name}}". The first line has "update" listed under "capabilities". The second line also has "update" listed under "capabilities". The "identity.entity.name" part is highlighted in yellow, indicating it is a template variable.

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

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

Available Templating Parameters (1 of 2)



PARAMETER	DESCRIPTION
<code>IDENTITY.ENTITY.ID</code>	THE ENTITY'S ID
<code>IDENTITY.ENTITY.NAME</code>	THE ENTITY'S NAME
<code>IDENTITY.ENTITY.METADATA.<<METADATA KEY>></code>	METADATA ASSOCIATED WITH THE ENTITY FOR THE GIVEN KEY
<code>IDENTITY.ENTITY ALIASES.<<MOUNT ACCESSOR>>.ID</code>	ENTITY ALIAS ID FOR THE GIVEN MOUNT
<code>IDENTITY.ENTITY ALIASES.<<MOUNT ACCESSOR>>.NAME</code>	ENTITY ALIAS NAME FOR THE GIVEN MOUNT
<code>IDENTITY.ENTITY ALIASES.<<MOUNT ACCESSOR>>.METADATA.<<METADATA KEY>></code>	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



```
path "auth/ldap/groups/{{identity.groups.ids.fb036ebc-2f62-4124-9503.name}}"
{
    capabilities = [ "update", "read"  ]
}

path
"secret/data/groups/{{identity.groups.names.engineering.metadata.team}}/*" {
    capabilities = [ "create", "update", "read", "delete" ]
}
```

- 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

A Vault client can be mapped as an entity

An entity can have multiple aliases for each mounted auth method

A group can be internal or external and also have aliases

A group can have multiple entities as its members

Group Name: Engineering (type: external)
Group ID: 0bfd703-f07d-2965...
Policies: **user-policy, engineering-group-policy, sensitive-data-policy**

Entity Name: Ravi
Entity ID: bf23f85c-4e26-b...
Policies:
Aliases:
ID: 7b0788d6-a259-6eb7-9...
Auth type: **LDAP**
Group: Engineering
Name: "ravi"
Policies: **transit-user-policy**
ID: 7617592a-e737-2e9d-d...
Auth type: **Userpass**
Name: "ravi"
Policies:

Identity Policies

Identity Policy

Token Policy

Token Policy

Entity Name: Mary
Entity ID: a65a8439-4e26-b...
Policies:
Aliases:
ID: 7b0788d6-a259-6eb7-9...
Auth type: **LDAP**
Group: **Engineering**
Name: "mary"
Policies: **transit-user-policy**
ID: 7617592a-e737-2e9d-d...
Auth type: **Userpass**
Name: "mary"
Policies:



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