# Module 1: Overview of Vault

## 1.1 Understanding Authentication

Authentication is the process of recognizing a user's identity

Authentication can be based on various approaches like username/password, tokens, and others.

## 1.2 Authentication Methods in Vault

There are multiple ways in which we can authenticate in Vault.

Before a client can interact with the Vault, it must authenticate against an auth method.


.

# Module 2: Vault Policies

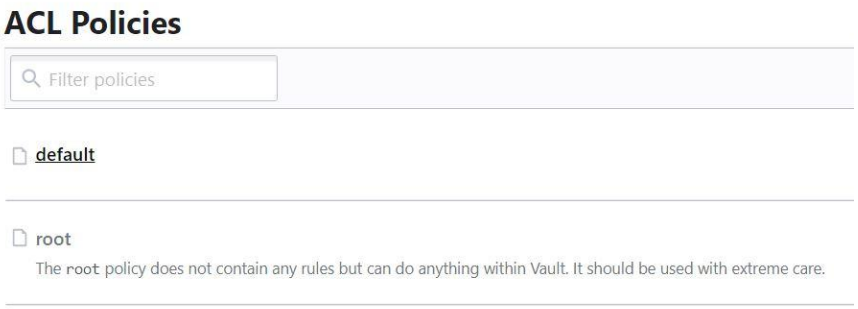
Vault Policies determine the level of access a user maintains.

| Users | Policy |
|---|---|
| Alice | Read from secret/ |
| John | Read and Write from secret/ |
| Matthew | Read, Write, Update Delete from secret/<br>Should be able to work with auth methods. |

When you first initialize Vault, the root policy gets created by default. The root policy is a special policy that gives superuser access to everything in Vault.

This allows the superuser to set up the initial policies, auth methods, etc.

In addition, another built-in policy, default, is created. The default policy is attached to all tokens and provides common permissions.




## 2.2 Basics of Policy Writing

Policies written in HCL format are often referred to as ACL Policies.

Everything in Vault is path-based, and admins write policies to grant or forbid access to certain paths and operations in Vault.

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

## 2.3 Basic Format

An empty policy grants no permission in the system.

Therefore ACL policies are defined for each path.

```
path "<PATH>" {
  capabilities = [ "<LIST_OF_CAPABILITIES>" ]
}
```

## 2.4 Primary Capabilities

Here are some of the important capabilities:

- Create
- Read
- Update
- Delete
- List
- Sudo
- Deny

## 2.5 Root Protected API Endpoints

Some of the paths are more restrictive and requires a root token or sudo capability in the policy.

Some of these paths are as follows:

- auth/token/accessors
- auth/token
- sys/audit
- sys/rotate
- sys/seal

# Module 3: Vault Policies - Part 2

## 3.1 ACL Rules Format - KV V2

The version 2 kv store uses a prefixed API, which is different from the version 1 API.

Writing and reading versions are prefixed with the data/ path

| Path | KV Version 1 | KV Version 2 |
|---|---|---|
| /secret/first | path "secret/first" {<br>capabilities = ["create"]<br>} | path "secret/**data**/first" {<br>capabilities = ["create"]<br>} |

## 3.2 Listing Secrets

- The metadata/ endpoint returns a list of key names at the specified location.
- The input must be a folder.
- The values themselves are not accessible via this command.

| Path | KV Version 2 |
|---|---|
| /secret/ | path "secret/**metadata**/" {<br>capabilities = ["list"]<br>} |

## 3.3 Reading Secret Metadata

This endpoint retrieves the metadata and versions for the secret at the specified path.

| Path | KV Version 2 |
|---|---|
| /secret/firstsecret | path "secret/metadata/**firstsecret**" {<br>capabilities = ["read"]<br>} |

### 3.4 Summarizing the Path

| Description | Path |
|---|---|
| Writing and Reading Versions | data/ |
| Listing Keys | metadata/ [list] |
| Reading Versions | metadata [read] |
| Destroy Versions of Secret | destroy/ [update] |
| Destroy all version of metadata for key | metadata/ [delete] |

# Module 4: AppRole Auth Method

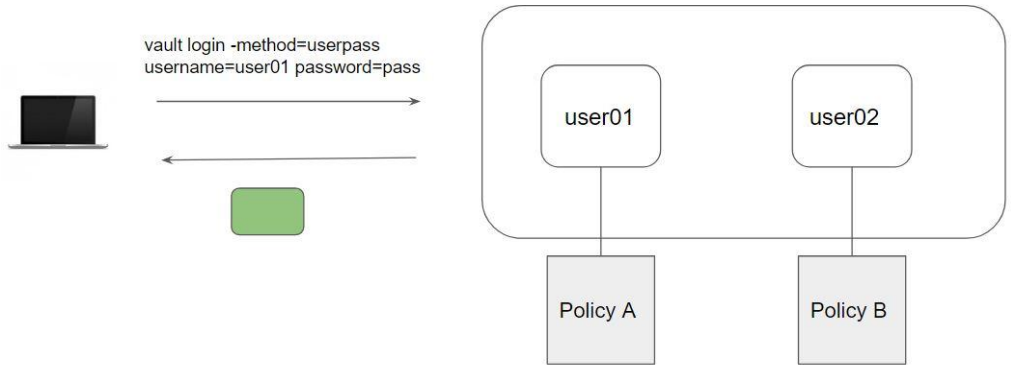
### 4.1 Overview of the Basics

Before a client can interact with Vault, it must authenticate against an auth method.

Some of the supported auth methods are targeted towards human users while others are targeted toward machines or apps.

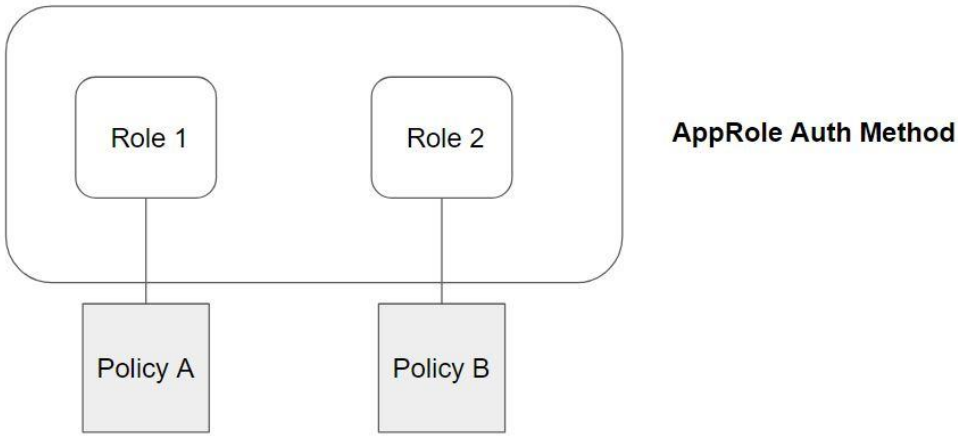
Upon authentication, a token is generated

The token may have attached policy, which is mapped at authentication time.

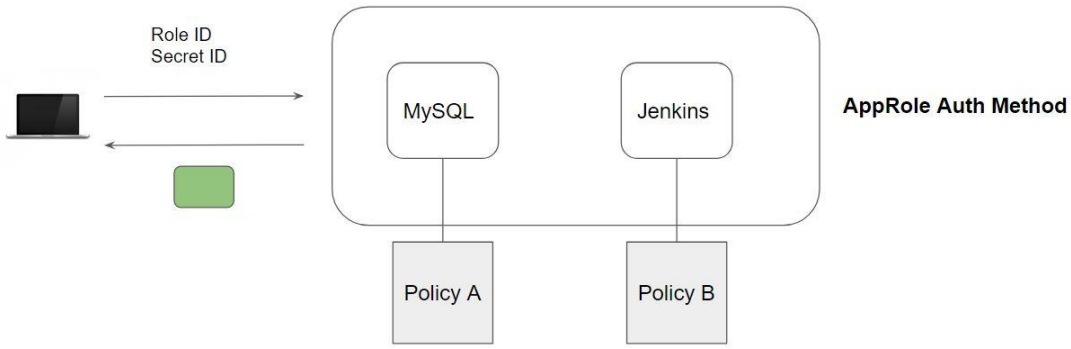


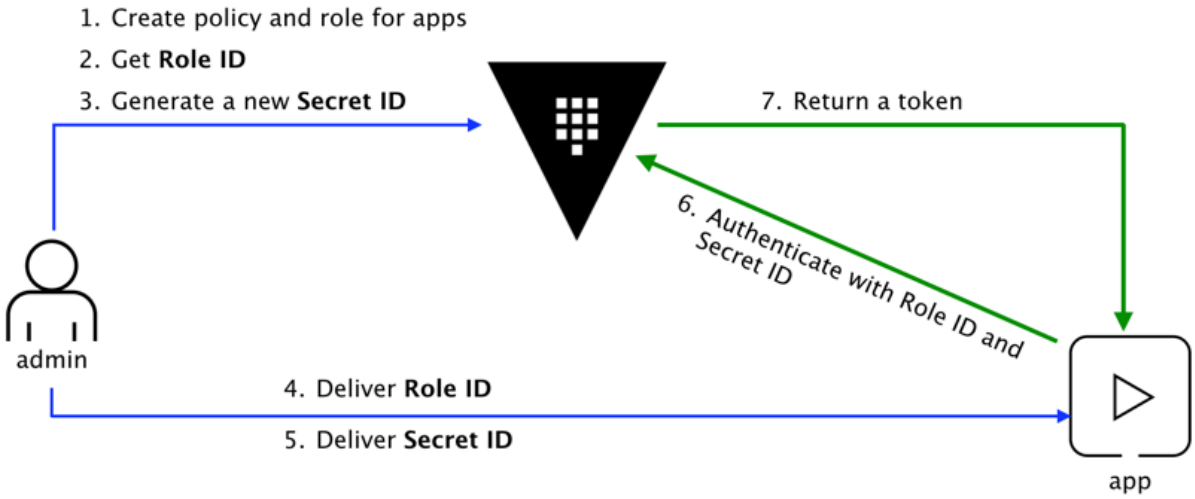
## 4.2 Overview of AppRole Method

The AppRole auth method allows multiple “roles” to be defined corresponding to different applications, each with different levels of access.




The following diagram represents the high-level overview of the working of the AppRole method.






## 4.3 Important Pointers

The AppRole auth method was specifically designed to be used by machines and applications but uses a similar authentication method that a human might use.

You can look at Role ID as a “username” and the Secret ID as a “password” allowing machines to authenticate to Vault.

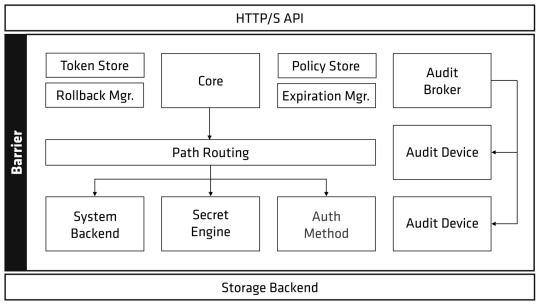
# Module 5: HTTP API with Vault

## 5.1 Overview of HTTP APIs

All of Vault's capabilities are accessible via the HTTP API

Most calls from the CLI actually invoke the HTTP API.

Some of the  Vault features are not available via the CLI and can only be accessed via the HTTP API.



## 5.2 Authenticating with CURL

With the help of curl, we can send a request to the Vault and can get the appropriate response.

To perform an operation, you need to make use of the client token. The client token can be set with the X-Vault-Token HTTP header within the request.

```
C:\Users\Zeal Vora>curl -H "X-Vault-Token: s.D4mEpn77EhyLvDOrAy2egr8g" -X GET http://127.0.0.1:8200/v1/secret/data/fir
stsecret?version=1
{"request_id":"40bada7b-d676-cc12-a3a9-36ca8d6ed59e","lease_id":"","renewable":false,"lease_duration":0,"data":{"data"
:{"user":"password"},"metadata":{"created_time":"2020-05-27T06:14:18.5399269Z","deletion_time":"","destroyed":false,"v
ersion":1}},"wrap_info":null,"warnings":null,"auth":null}
```

## 5.3 HTTP APIs Endpoints

All API routes are prefixed with /v1/.

With few documented exceptions, all request body data and response data from Vault is via JSON.

/v1/secret/foo

This maps to secret/foo where foo is the key in the secret/ mount

```
$ curl \
    -H "X-Vault-Token: f3b09679-3001-009d-2b80-9c306ab81aa6" \
    -X GET \
    http://127.0.0.1:8200/v1/secret/foo
```

## 5.4 HTTP Verbs to Capability Mapping

| Capability | Associated HTTP Verb |
|---|---|
| create | POST/PUT |
| list | GET |
| update | POST/PUT |
| delete | DELETE |
| list | LIST |

## 5.5 List Operations

For listing related aspects, like vault secrets list, you can either issue a GET with the query parameter list=true, or you can use the LIST HTTP verb.

```
$ curl \
    -H "X-Vault-Token: f3b09679-3001-009d-2b80-9c306ab81aa6" \
    -X LIST \
    http://127.0.0.1:8200/v1/secret/
```

# Module 6: Token Capabilities

The token capabilities command fetches the capabilities of a token for a given path.

```
C:\Users\Zeal Vora>vault token capabilities sys/
deny
```

```
C:\Users\Zeal Vora>vault token capabilities cubbyhole/
create, delete, list, read, update
```

If a TOKEN is provided as an argument, the "/sys/capabilities" endpoint and permission is used.

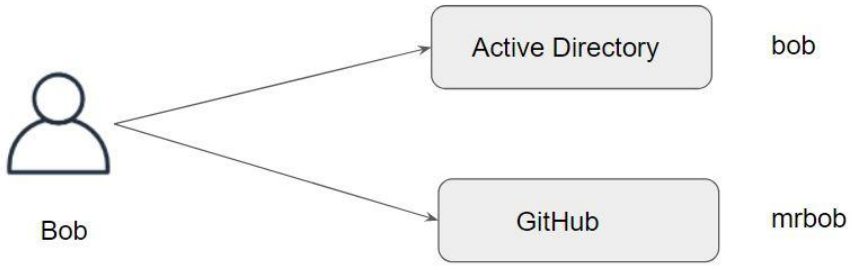
If no TOKEN is provided, the "/sys/capabilities-self" endpoint and permission is used with the locally authenticated token.

# Module 7: Entities and Aliases

## 7.1 Authentication for Multiple Users

Vault supports multiple authentication methods and also allows enabling the same type of authentication method on different mount paths.

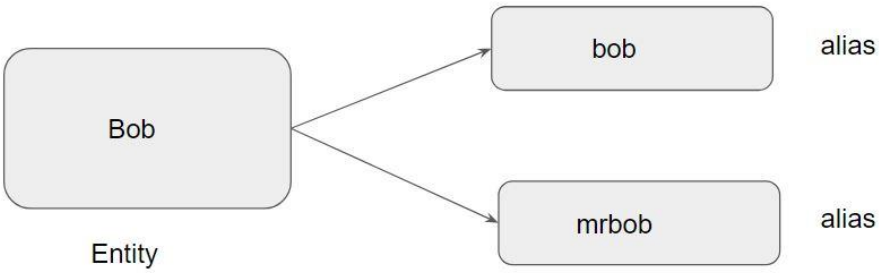
Each Vault client may have multiple accounts with various identity providers that are enabled on the Vault server.

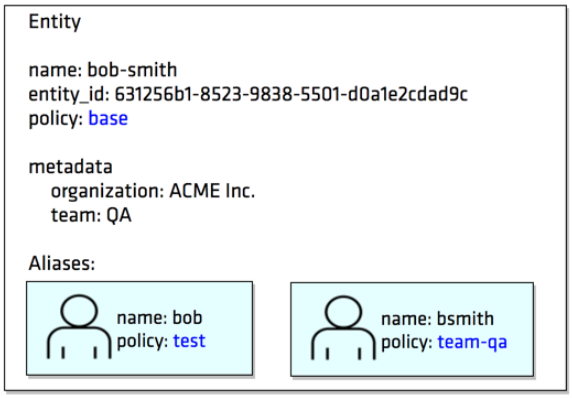


## 7.2 Entity and Alias

An entity can be created with the name of Bob.

Create an alias that represents each of the accounts and associate them as an entity member.

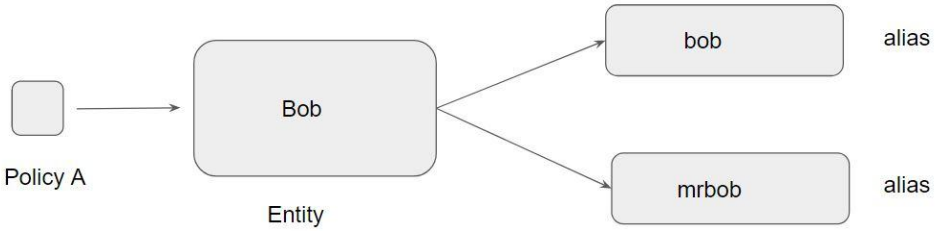



## 7.3 Overview of Policies at Entities

We can set policies on the entity level and both the accounts can inherit.

Create an alias that represents each of the accounts and associate them as an entity member.



7.4 Identity Secret Engine

The Identity secrets engine maintains the clients who are recognized by Vault.

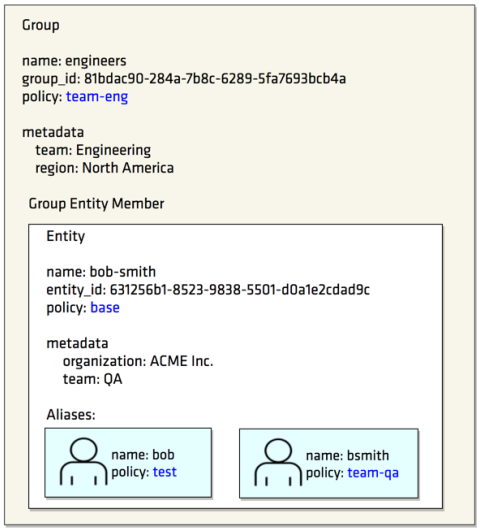
Each client is internally termed as an Entity. An entity can have multiple Aliases

This secret engine will be mounted by default. This secret engine cannot be disabled or moved.

# Module 8: Identity Groups

A group can contain multiple entities as its members.

Policies set on the group is granted to all members of the group.




By default, Vault creates an internal group.

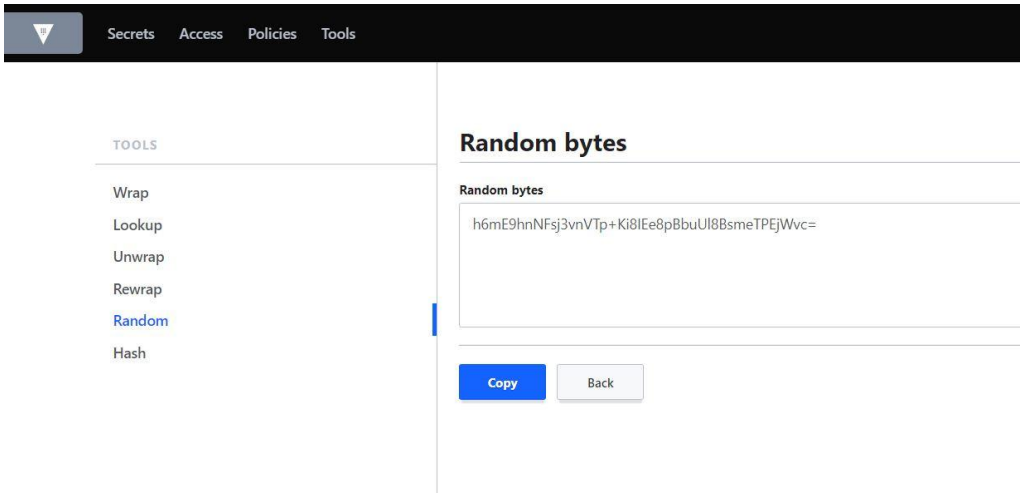
Many organizations already have groups defined within their external identity providers like Active Directory.

External Groups allows to link Vault with the external identity provider (auth provider) and attach appropriate policies to the group.

# Module 9: Tools in Vault

Vault contains a certain set of tools that will allow us to achieve a specific function.

These are also available in the /sys/tools endpoint




| Endpoints | Description |
|---|---|
| /sys/tools/random/164 | Generate 164 bytes of random value. |
| /sys/tools/hash/sha2-512 | Hash input data based on SHA2 |

# Module 10: Vault Auto Complete

Vault Auto-Complete feature allows automatic completion for flags, subcommands, and arguments.

| Manual Command | With Auto-Complete |
|---|---|
| vault operator seal | vault op [TAB] se [TAB] → vault operator seal |

Be sure to restart your shell after installing autocompletion!

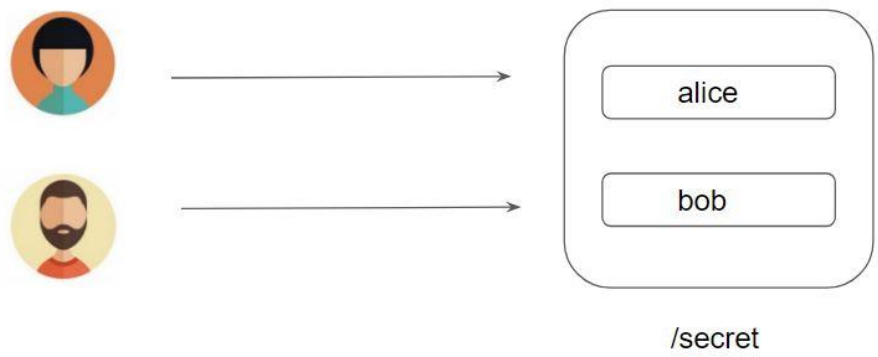
# Module 11: Path Templating

## 11.1 Understanding the Challenge

Let's assume that /secret has two prefix:

- /secret/alice
- /secret/bob

Each user should only be able to access their own data.

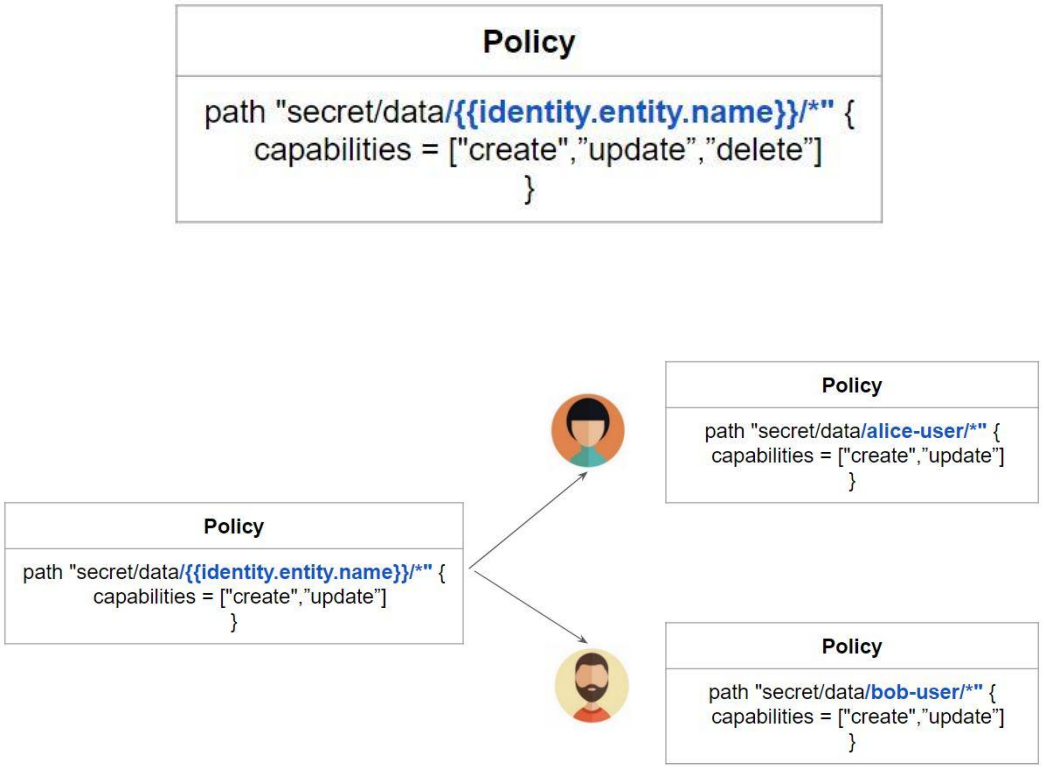



Each user will need to have a different policy.

| Users | Users Policies |
|---|---|
| Alice | path "secret/data/**alice**/" {<br>capabilities = ["create","update","delete"]<br>} |
| Bob | path "secret/data/**bob**/" {<br>capabilities = ["create","update","delete"]<br>} |

## 11.2 Overview of Path Templating

Path Templating allows variable replacement based on information of the entity.




## 11.3 Supported Parameters

| Name | 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.ids.<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 |

# Module 12: Vault Policies - Transit Engine

For the vault clients to be able to perform the encrypt and decrypt operations, there are two important policy rules that need to be added.

```
path "transit/encrypt/<key_name>" {
   capabilities = [ "update" ]
}

path "transit/decrypt/<key_name>" {
   capabilities = [ "update" ]
}
```