



# Vault Namespaces, Authentication, & ACL Policies



# Agenda

Namespaces 01

---

Authentication 02

---

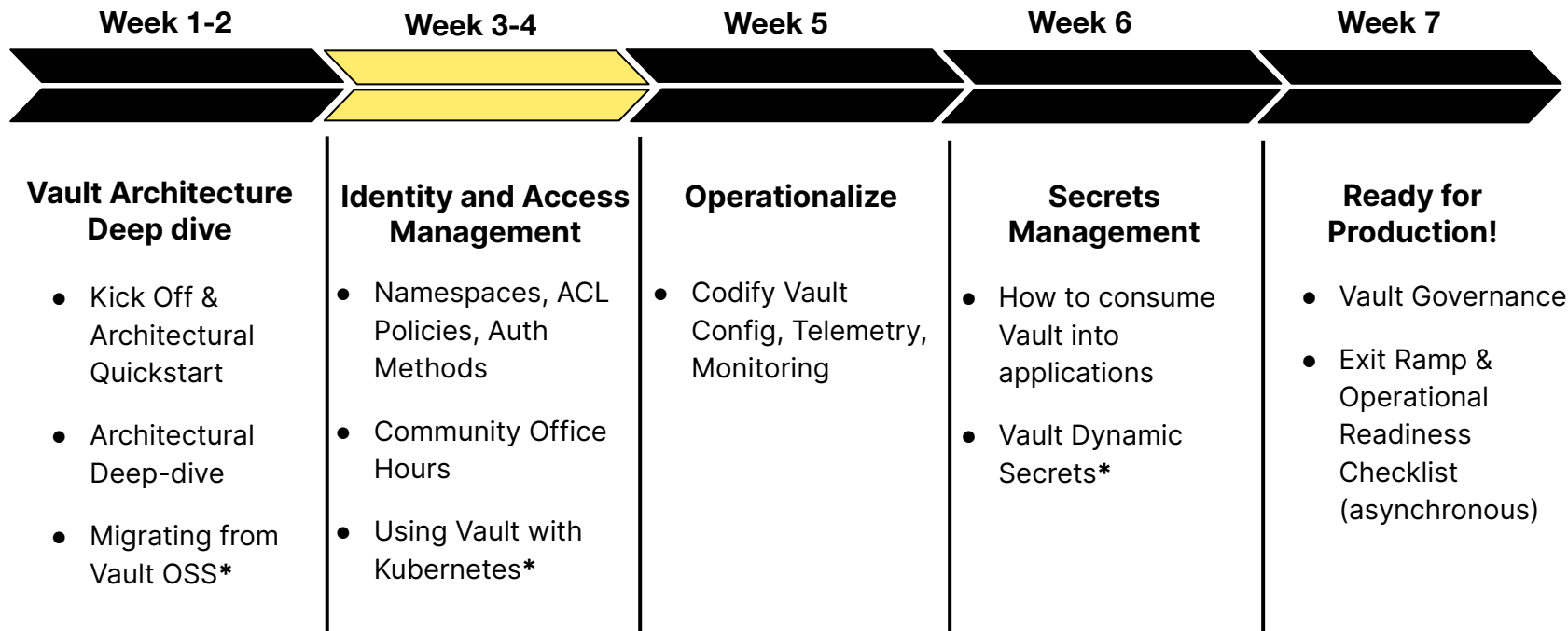
Policies 03

---

# Vault Onboarding Program

A 7 week guided community environment

Assisting customers with onboarding and adoption

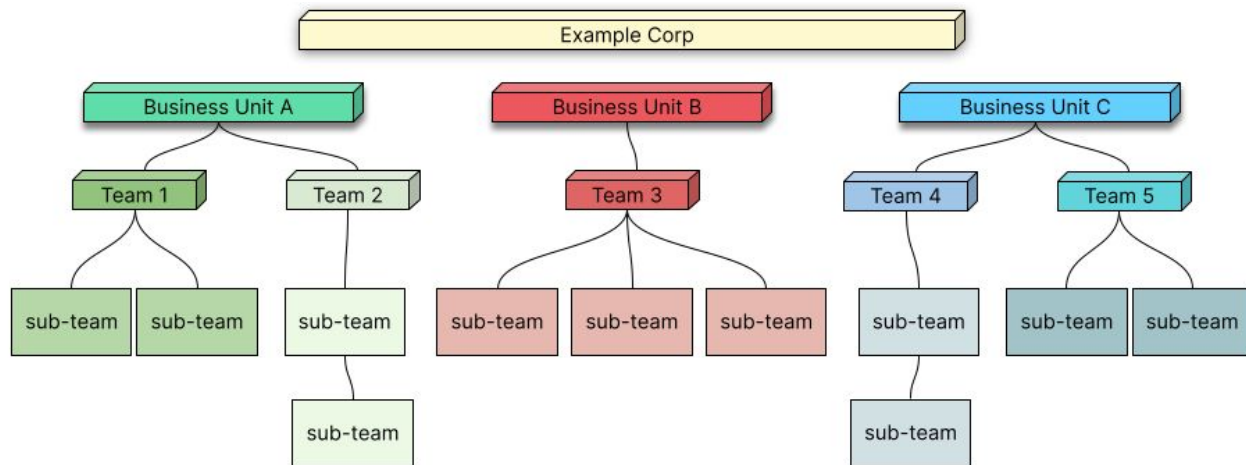


01

# Namespaces

# Namespaces

- Namespaces create “Vaults within a Vault” which enables segmentation for teams & services across an organization
- Each namespace maintains its own path structure
- Namespace segmentation facilitates delegation of administration & helps control blast radius by isolating policies & secrets management



# Namespace Contents

## Unique to each namespace

- Policies
- Secrets Engines
- Authentication Methods
- Tokens
- Identity Entities and Groups

# Considerations

Requirement	What to Consider
<b>Organizational Structure</b>	What is your organizational structure?
	What is the level of granularity across lines of businesses (LOBs), divisions, teams, services, apps that needs to be reflected in Vault's end-state design?
<b>Self-Service Requirements</b>	Given your organizational structure, what is the desired level of self-service required?
	How will Vault policies be managed?
	Will teams need to directly manage policies for their own scope of responsibility?
	Will they be interacting with Vault via some abstraction layer where policies and patterns will be templated? For example, configuration by code, Git flows, the Terraform Vault provider, custom onboarding layers, or some combination of these.

# Considerations

## Requirement

## What to Consider

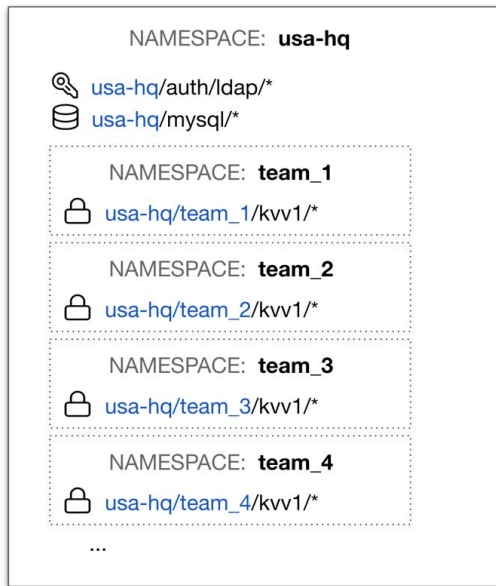
<b>Audit Requirements</b>	What are the requirements around auditing usage of Vault within your organization?
	Is there a need to regularly certify access to secrets?
	Is there a need to review and/or decommission stale secrets or auth roles?
	Is there a need to determine chargeback amounts to internal customers?
<b>Secrets Engine Requirements</b>	What types of secrets engines will you use (KV, database, AD, PKI, etc.)?



# Using Namespaces

- Namespaces should be leveraged sparingly and primarily to delineate administrative boundaries
- Often many unnecessary namespaces get created by trying to replicate organizational structure

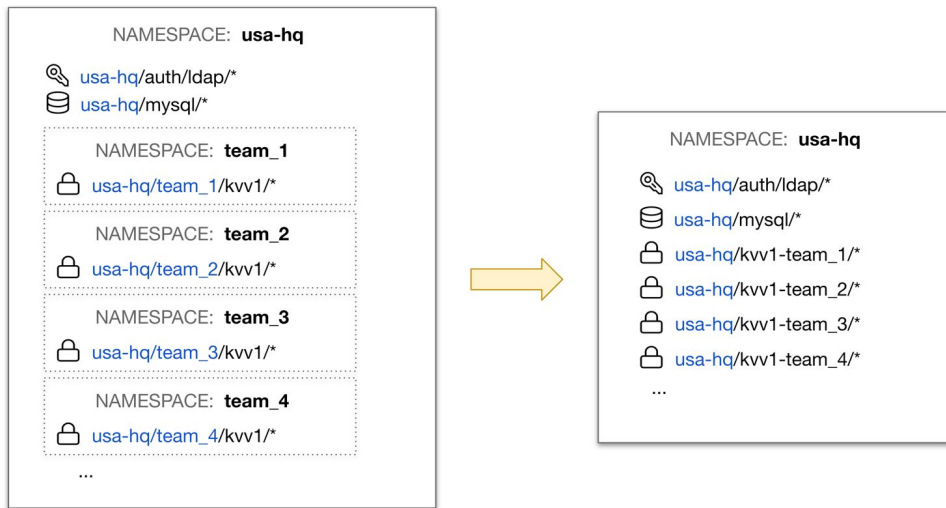
## Anti-Pattern



# Using Namespaces

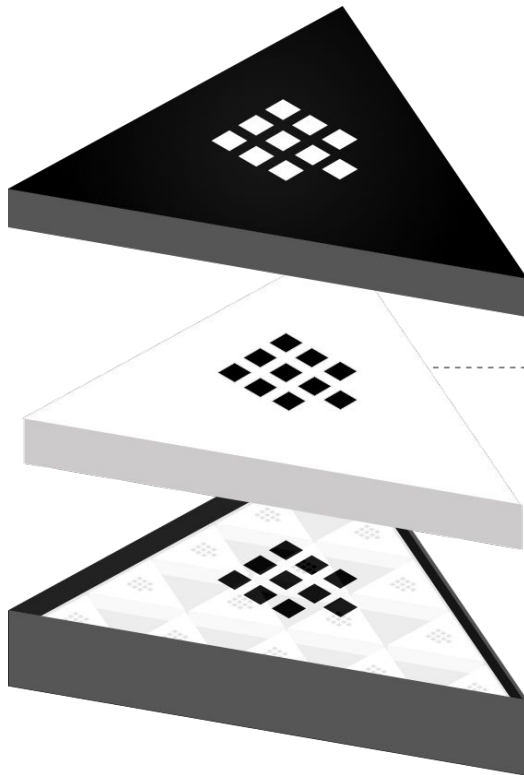
- Instead of providing self-service by implementing many namespaces we recommend implementing an onboarding layer
- Shifting the administrative boundary from teams to the onboarding layer reduces the number of namespaces while enforcing a standard naming convention, secrets path structure, and templated policies

## Best Practice





# Root Namespace



- **Root (Namespace)**

**Members:**  
Security Team

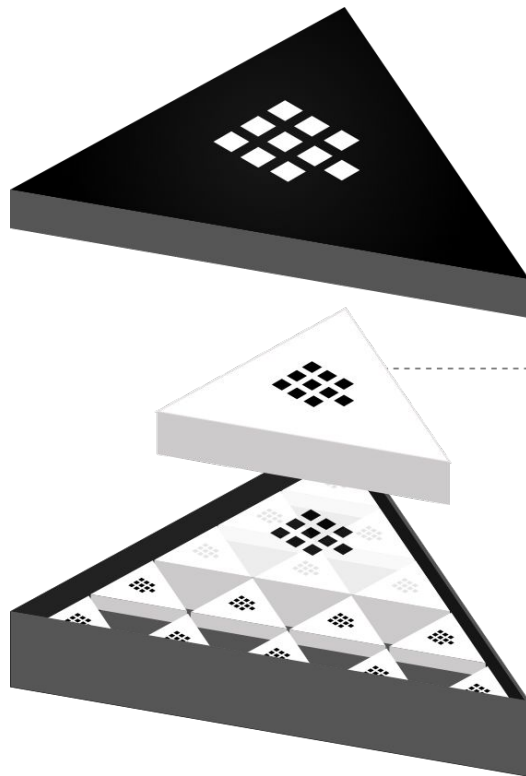
**Namespace Specific Configuration:**  
Defined global member access  
Defined global authentication mounts  
Defined global secrets engines

*Note: Vault supports namespaces within namespaces. By default there can always be a globally managed namespace that has rights to sub-namespaces, such as the Teams, and smaller namespaces*





# Namespaces for Teams and Groups



- **Engineering Org (Namespace)**

**Members:**

Security Team, Operations Teams, Engineering Manager

**Namespace Specific Configuration:**

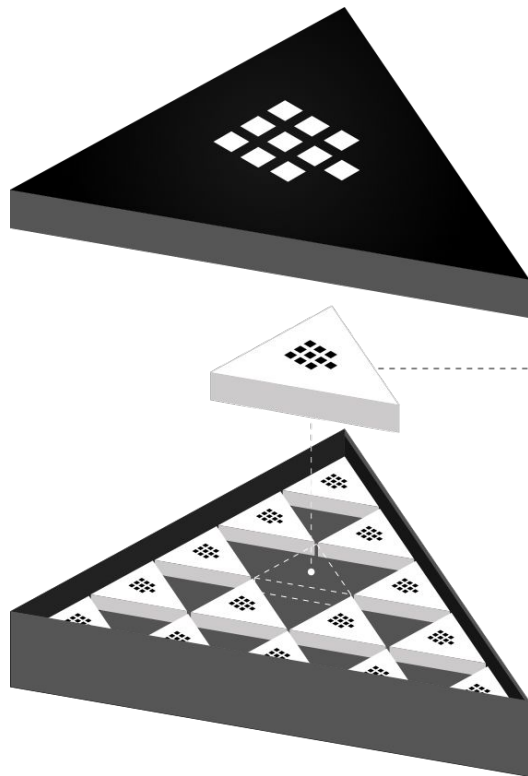
Defined engineering member access  
Defined engineering authentication mounts  
Defined engineering secrets engines

*Note: Vault supports namespaces within namespaces. By default there can always be a parent managed namespace that has rights to sub-namespaces, such as the Applications/User namespaces*





# Namespace per each Application



- **Application (Namespace)**

**Members:**

Alex Smith, Jennifer Johnson, Steve Stevens

**Namespace Specific Configuration:**

Defined member access

Defined authentication mounts for AWS, Azure, and GCP systems

Defined custom secrets engine



# Getting Started with Namespaces

CLI

```
# Create namespace
$ vault namespace create usa-hq

# Create child namespaces
$ vault namespace create -namespace=usa-hq training

# List namespaces from within root namespace
$ vault namespace list
Keys
----
production/
eu-hq/
usa-hq/

# List child namespaces for usa-hq namespace
$ vault namespace list -namespace=usa-hq
Keys
----
dev-test/
prod/

# Instead of a CLI flag, environment variables can be used
$ export VAULT_NAMESPACE="usa-hq"
$ vault namespace create sales
```

# Getting Started with Namespaces

API

```
# Create namespace
```

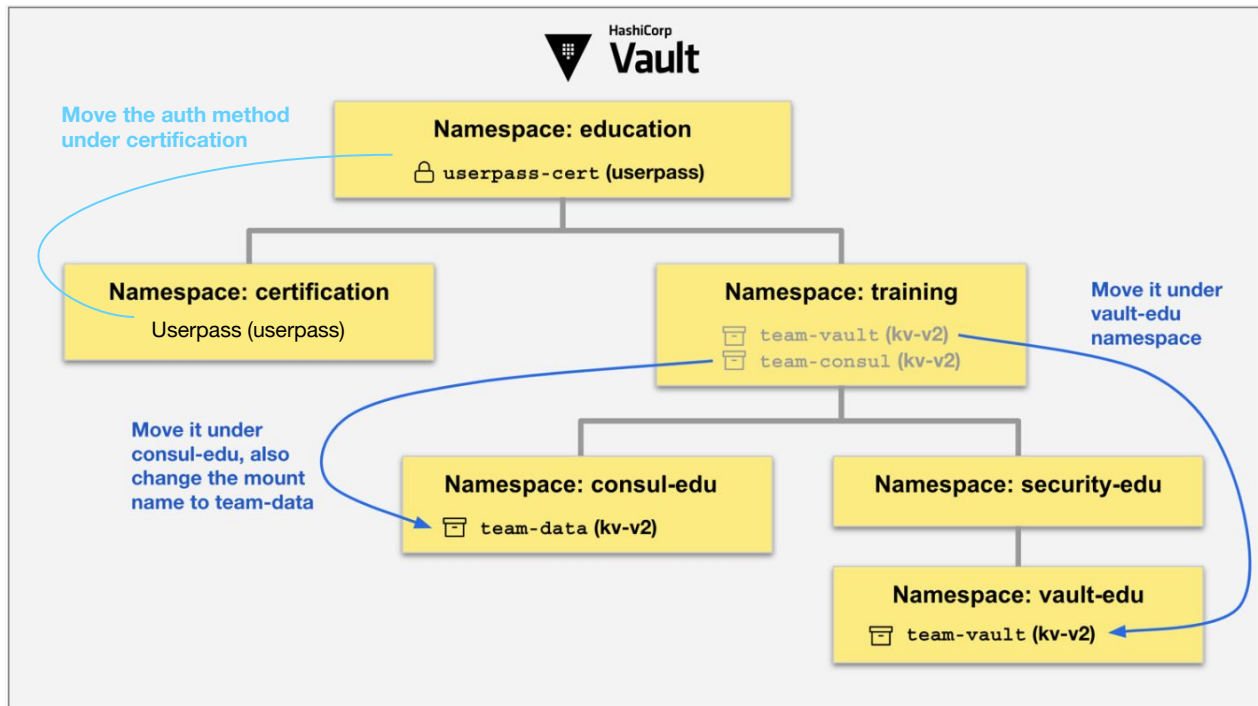
```
$ curl --header "X-Vault-Token: $VAULT_TOKEN" .\  
  --header "X-Vault-Namespace: admin" \  
  --request POST \  
$ VAULT_ADDR/v1/sys/namespaces/usa-hq | jq -r ".data"
```

```
# Create child namespace of admin/usa-hq
```

```
$ curl --header "X-Vault-Token: $VAULT_TOKEN" \  
  --request POST \  
$ VAULT_ADDR/v1/admin/usa-hq/sys/namespaces/training | jq -r  
".data"
```

# Mount Move Command

- Requires Vault 1.10.0+
- API endpoint to move secret engines & auth methods
- Works across mounts within a namespace or across namespaces





# Migrate a Secret Engine

CLI

```
...  
# List the enabled mounts in the education/training namespace  
$ vault_namespace "education/training" vault secrets list  
# Migrate the data  
$ vault secrets move <original namespace path> \ <new namespace path>  
# Confirm that the data was successfully migrated  
$ vault_namespace=  
"education/training/security-edu/vault-edu" vault kv get team-vault  
# Verify the team-vault mount path no longer exists  
$ vault_namespace="education/training" vault secrets list
```

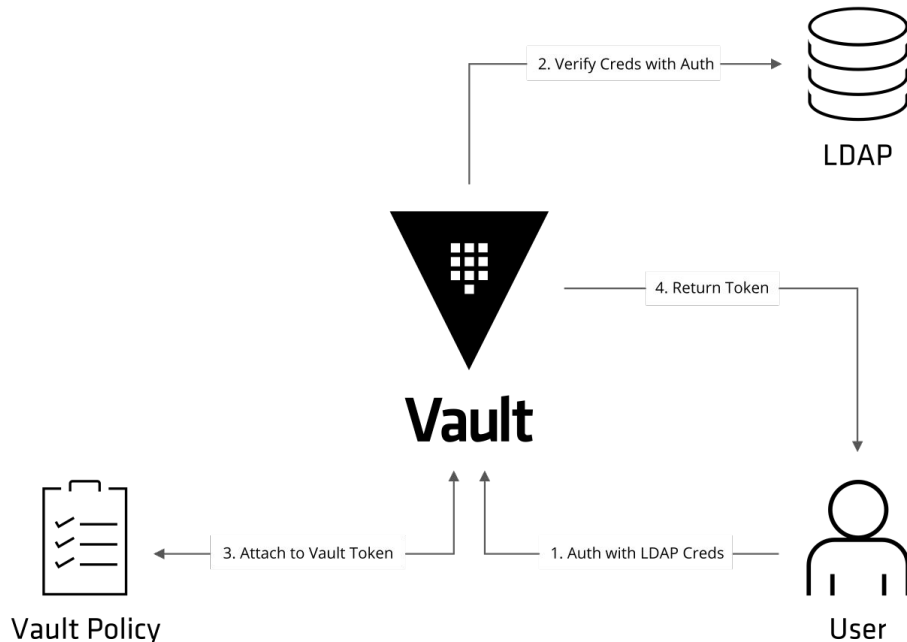
02

# Authentication

# Authentication in Vault

Vault supports integrating with trusted identity providers to validate user or machine supplied information to create a token tied to a pre-configured policy

- If Vault is able to successfully validate the credentials, a Vault token will be returned that can then be used to access Vault
- The token Vault returns is associated to a Vault policy that defines what access and capabilities the token can perform



# Human vs. Machine Authentication

Vault provides authorization and not authentication of users so you will need to integrate with a trusted Identity provider to authenticate and verify the client before access to Vault is granted

- Multiple authentication methods can and should be used
- Human users should authenticate using a method that leverages an external identity provider
- Machine users should authenticate using AppRole or auth type that uses instance metadata to authenticate the machine such as AWS or GCP

Human Auth	Machine Auth
GitHub	AppRole
LDAP/AD	AWS
OIDC	Azure
Okta	Google Cloud
Cloud IAM	JWT
Username & Password	Kubernetes
	RADIUS
	TLS Certificates
	Cloud Foundry

# Vault Identity Recap

## Entities & Groups



**Group Name: Accounting**

Group ID: 0bfed703-f07d-2965...

Policies: **accounting**

A **group** can have multiple entities as its members

Also, a **group** can have subgroups

A Vault client can be mapped as an entity



**Entity Name: Bob Smith**

Entity ID:  
bf23f85c-4e26-b...

Policies: **test**

Aliases:

ID:  
7b0788d6-a259-6eb7-9...  
Auth type: **LDAP**  
Name: "bsmith"  
Policies: test-admin, devops

An entity can have multiple aliases

ID: 7617592a-e737-2e9d-d...  
Auth type: **Userpass**  
Name: "bob"  
Policies: base



**Entity Name: HCP Billing**

Entity ID:  
lw23p85c-2e9-b...

Policies: **billing**

Aliases:

ID: 6713592a-e737-2e9d-d...  
Auth type: **AWS**  
Role Name: "hcp-billing"  
Policies: app



**Group Name: Payment**

Group ID:  
l273k85c-2e9-b...

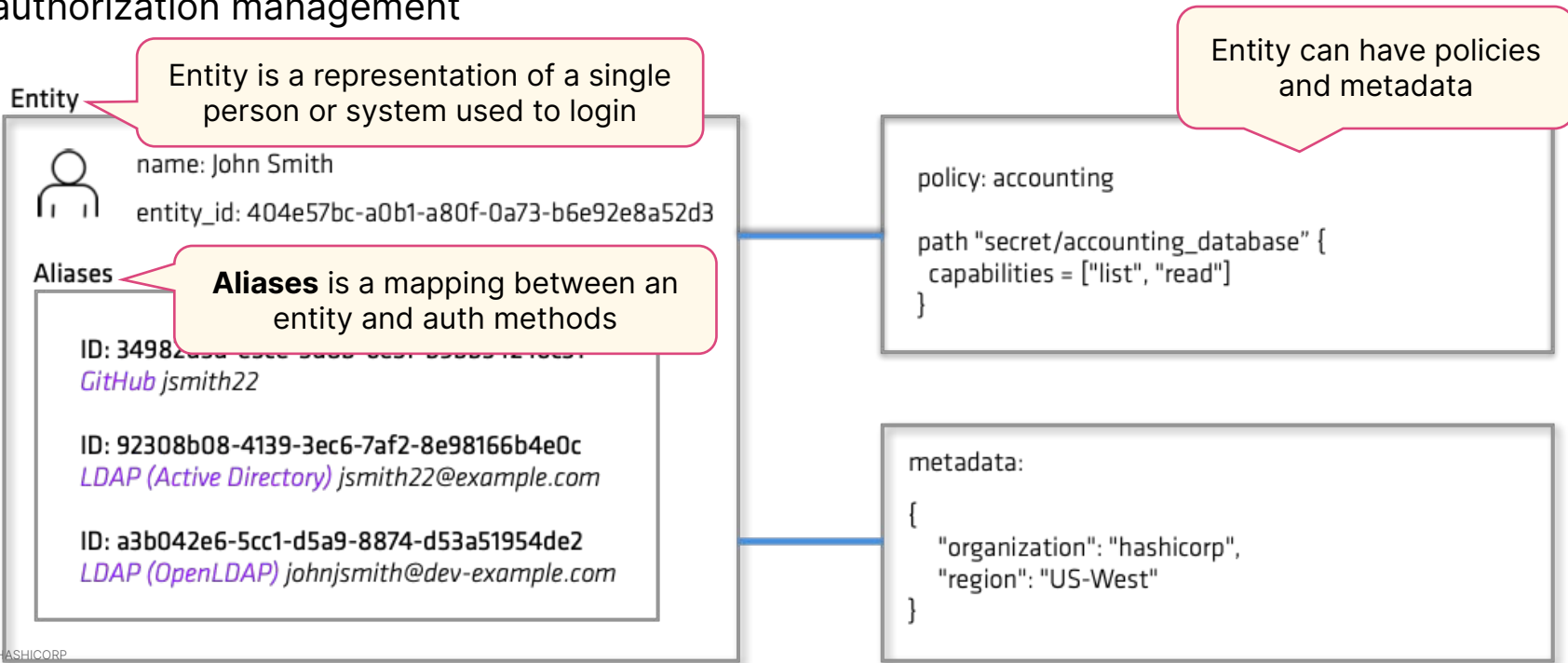
Policies: **payment**

Aliases:

ID: 6713592a-e737-2e9d-d...  
Auth type: **LDAP**  
Name: "payment"  
Policies: payment

# Entities and Aliases

Map multiple user authentication schemes to a single entity to provide for more efficient authorization management



# Token and Policies

Policies can be assigned to entities which will grant **additional** permissions

## Entity



name: John Smith  
entity\_id: 404e57bc-a0b1-a80f-0a73-b6e92e8a52d3  
**policy: acct-manager**

## Aliases

ID: 34982d3d-e3ce-5d8b-6e5f-b9bb34246c31  
GitHub *jsmith22*  
Policy: *acct-test*

ID: 92308b08-4139-3ec6-7af2-8e98166b4e0c  
LDAP (Active Directory) ***jsmith22@example.com***  
**Policy: expense**

ID: a3b042e6-5cc1-d5a9-8874-d53a51954de2  
LDAP (OpenLDAP) *johnjsmith@dev-example.com*  
Policy: *accounting*

Token inherits  
**capabilities** granted  
by both policies



expense

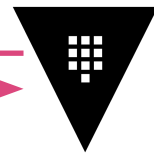


acct-manager



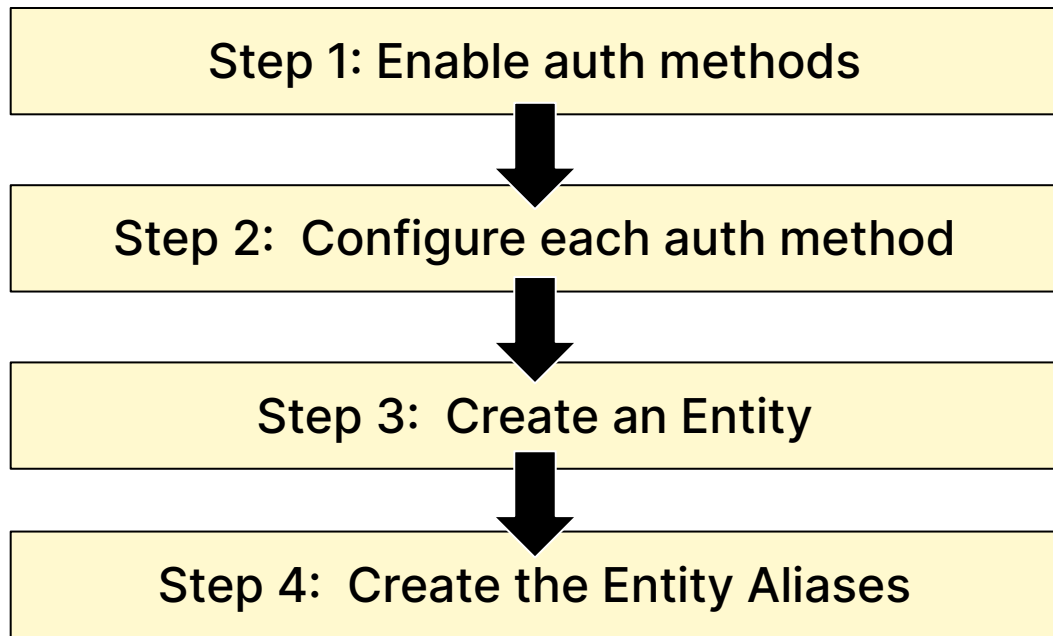
Member of  
this entity

3. Returns a token  
1. Authenticate with LDAP account



2. Verify  
with LDAP

# Basic Workflow

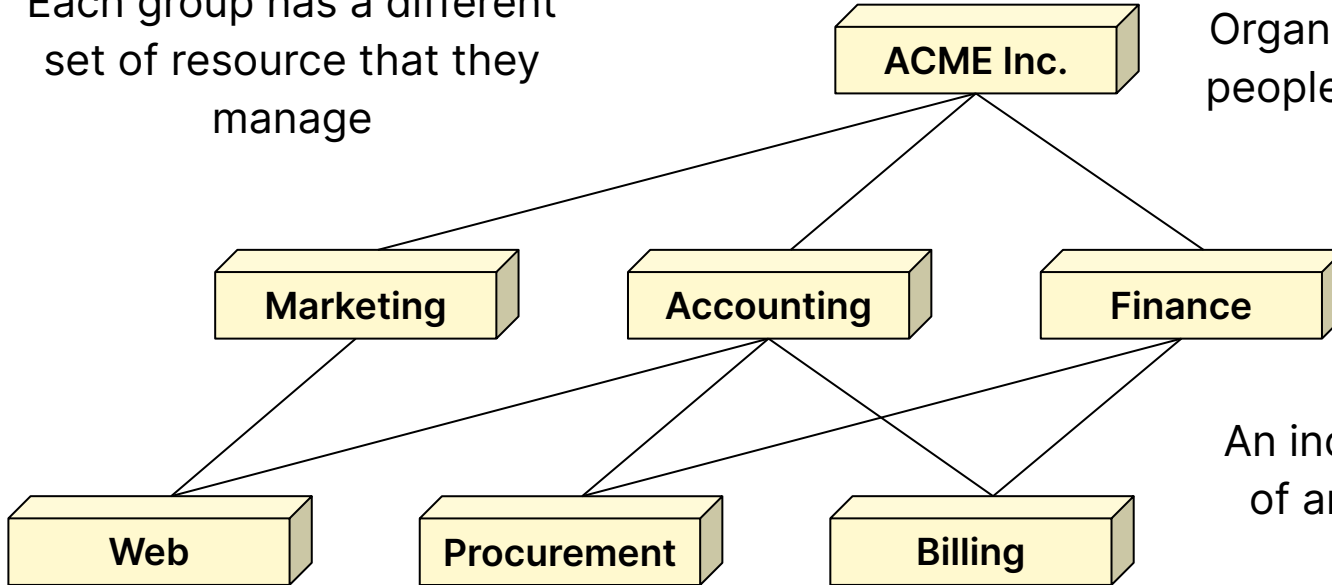




# Organizational Structure

Each group has a different set of resource that they manage

Organization is composed of people who belong in groups

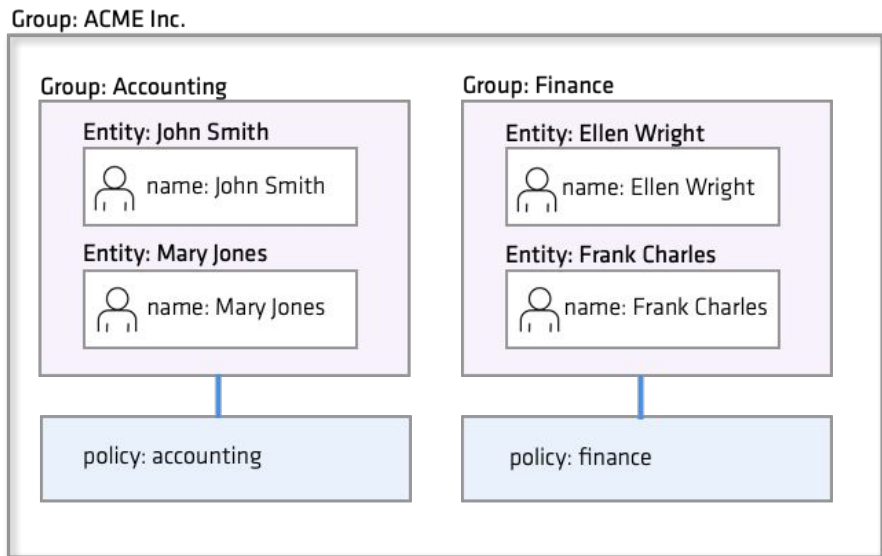


An individual may be a part of any number of groups

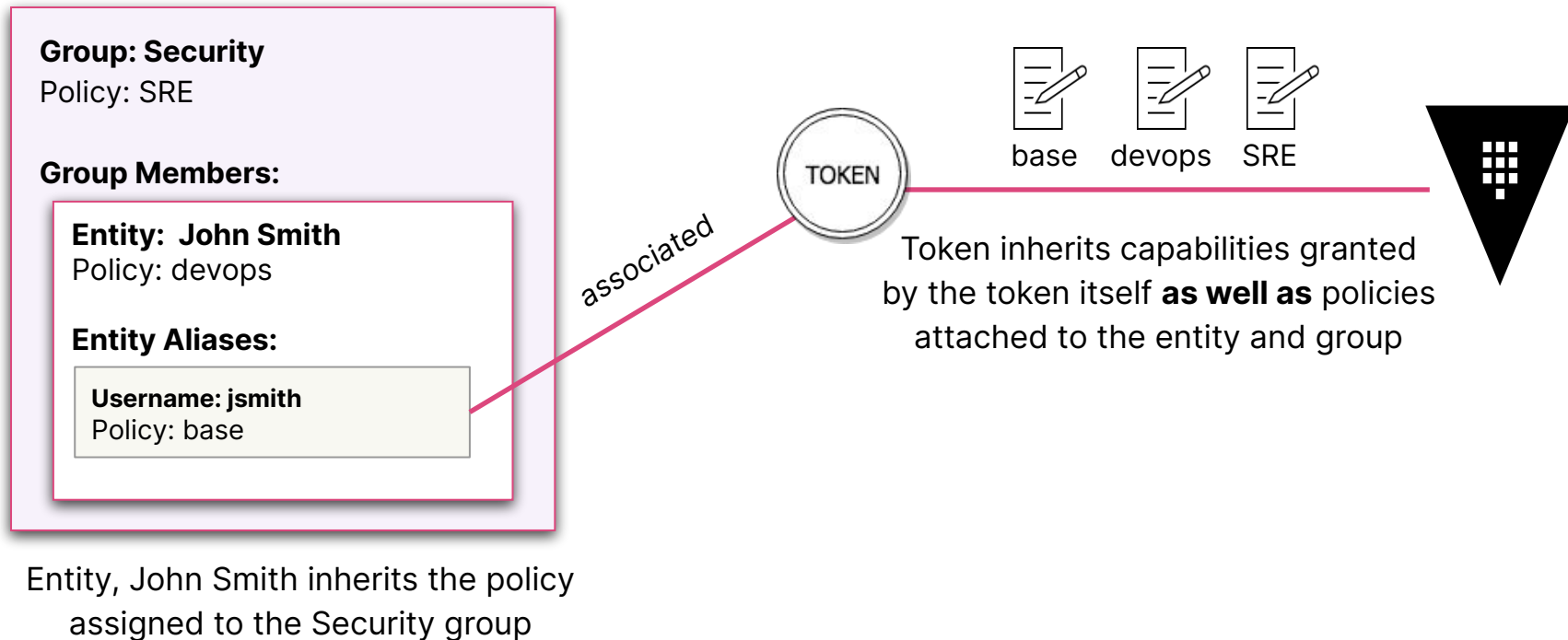
# Identity Groups

Identity group maps multiple user entities to a group for authorization management at scale

- Identity groups can have multiple entities as its members as well as subgroups
- Entities can be direct member of groups
- Inherit the policies of the groups they belong to
- Entities can be indirect member of groups
- Groups can have a set of policies and metadata inherited from the member entity or subgroups



# Group Hierarchical Permissions



# Identity Groups Aliases

- **Internal groups** are those groups manually created by the operators via API
- **External groups** are the groups which Vault infers and creates based on the group associations coming from the auth methods
- Identity **group alias** is a mapping between identity groups and groups in an third party authentication provider
  - If a user is a part of an external group (LDAP group), automatically adds the user to the identity group inheriting the policies and metadata



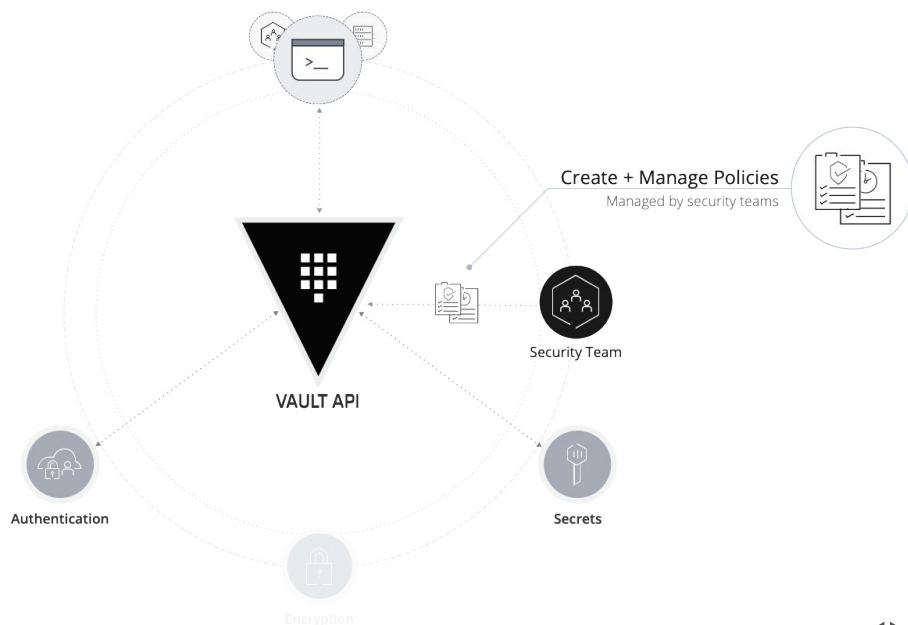
03

# Policies

# Vault Policies

## Role-Based Access Control

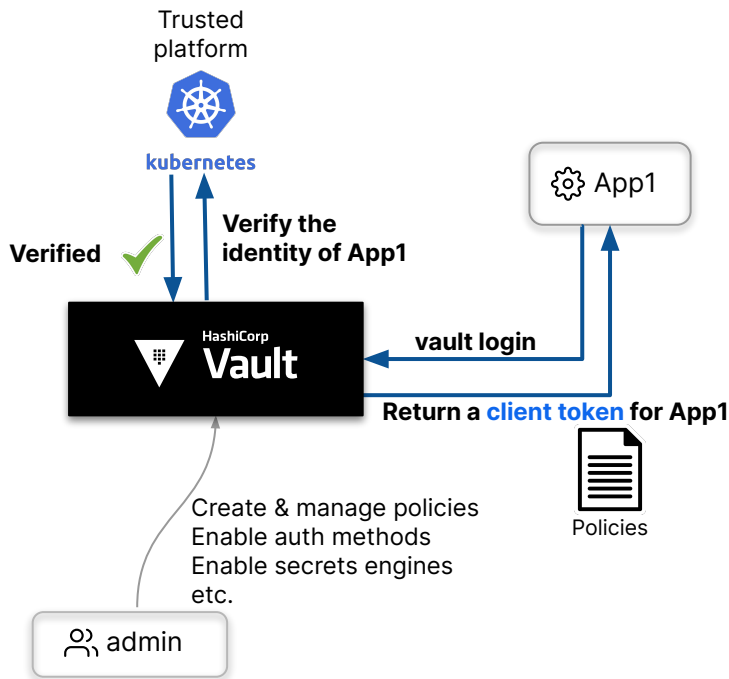
- Use policies to govern the behavior of the Vault clients
- Instrument Role-Based Access Control (RBAC)
- **Safeguard access** and secret distribution to apps



# 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 Vault operations (e.g. read secrets)



# 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



# Policies

path



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

Example path

```
http://VAULT_ADDR:8200/v1/auth/userpass/users/apps
```



# Policies

path

```
path "<PATH>" {
```



```
capabilities = [ <LIST> ]
```

```
}
```

## Capabilities

create  
read  
update  
delete  
list  
sudo  
deny

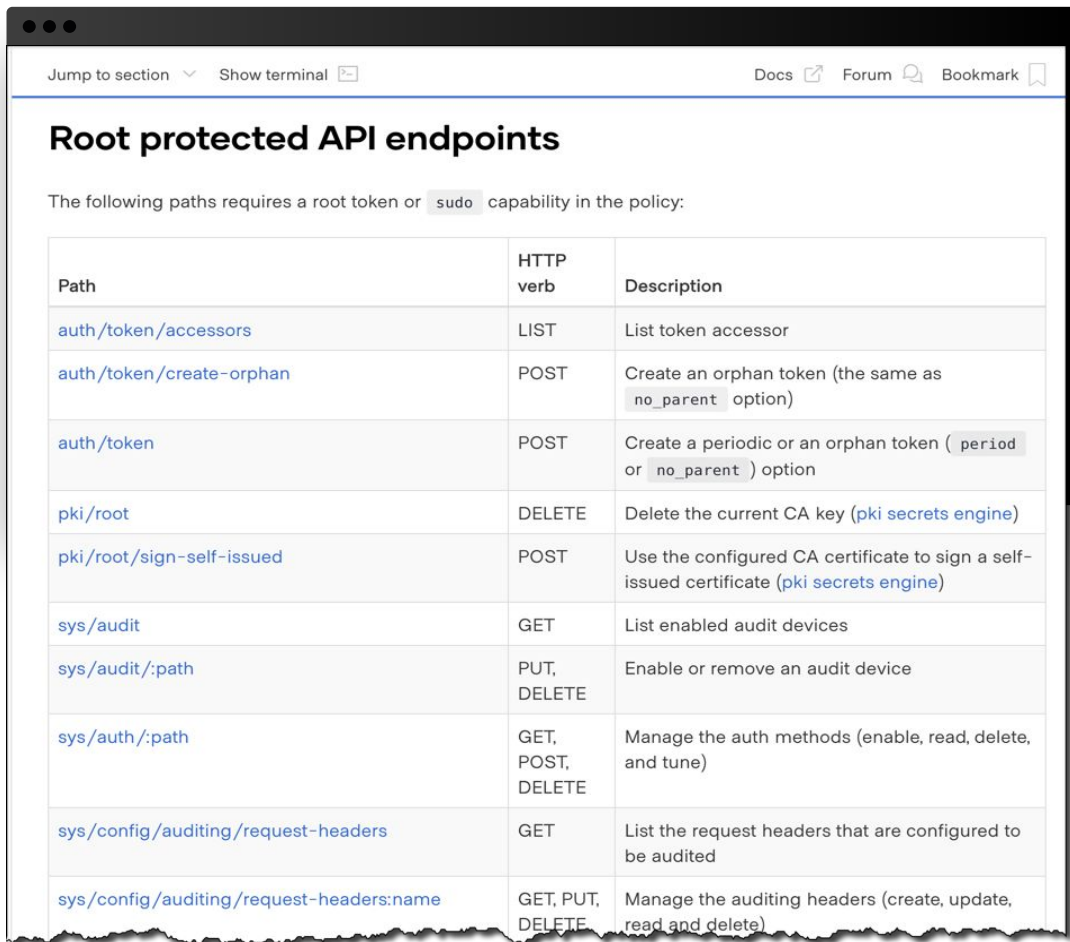
## HTTP Verbs

POST/PUT  
GET  
POST/PUT  
DELETE  
LIST

# Root protected paths

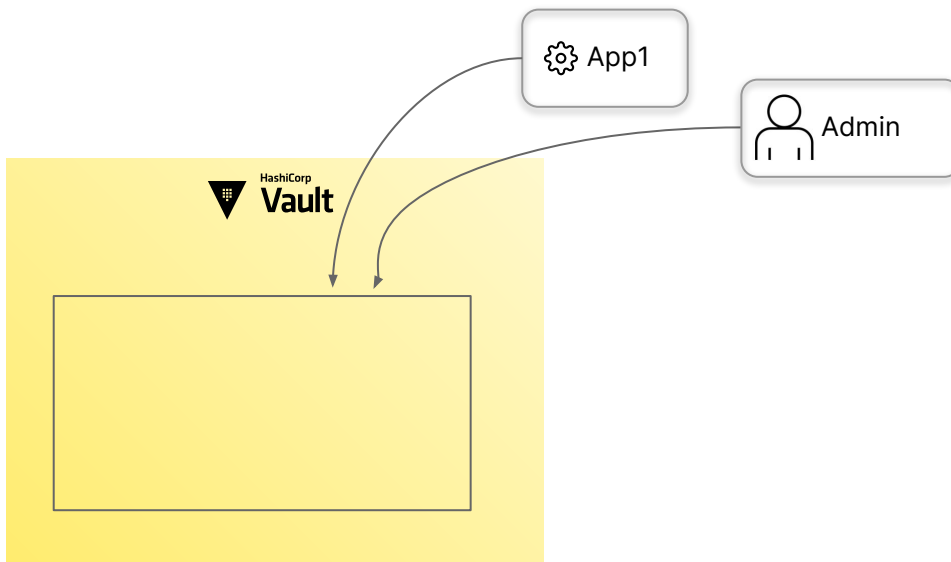
The **sudo** capability must be provided for those root protected paths

Refer to the [Tutorial](#)



Path	HTTP verb	Description
<a href="#">auth/token/accessors</a>	LIST	List token accessor
<a href="#">auth/token/create-orphan</a>	POST	Create an orphan token (the same as <code>no_parent</code> option)
<a href="#">auth/token</a>	POST	Create a periodic or an orphan token ( <code>period</code> or <code>no_parent</code> ) option
<a href="#">pki/root</a>	DELETE	Delete the current CA key ( <a href="#">pki secrets engine</a> )
<a href="#">pki/root/sign-self-issued</a>	POST	Use the configured CA certificate to sign a self-issued certificate ( <a href="#">pki secrets engine</a> )
<a href="#">sys/audit</a>	GET	List enabled audit devices
<a href="#">sys/audit/:path</a>	PUT, DELETE	Enable or remove an audit device
<a href="#">sys/auth/:path</a>	GET, POST, DELETE	Manage the auth methods (enable, read, delete, and tune)
<a href="#">sys/config/auditing/request-headers</a>	GET	List the request headers that are configured to be audited
<a href="#">sys/config/auditing/request-headers:name</a>	GET, PUT, DELETE	Manage the auditing headers (create, update, read and delete)

# Policy Authoring Workflow



Discover the policy:

1. Gather secret requirements
2. Perform operations with Vault
3. Discover the paths & capabilities required
4. Define a policy
5. Test the policy

# Three discovery techniques

- API documentation
- -output-curl-string
- Vault's audit logs

# API docs

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

## Encrypt Data

This endpoint encrypts the provided plaintext using the named key.

This path supports the `create` and `update` policy capabilities as



context parameter is empty or not). If the user only has `update` capability and the key does not exist, an error will be returned.

Method	Path
POST	/transit/encrypt/:name

app1

update capability

# CLI command flag

-output-curl-string

```
$ vault policy read -output-curl-string test
```

default HTTP verb is GET



```
curl -H "X-Vault-Request: true" -H "X-Vault-Token:  
$(vault print token)"
```



```
http://127.0.0.1:8200/v1/sys/policies/acl/test
```

path

```
path "sys/policies/acl/test" {  
  capabilities = [ "read" ]  
}
```

# Audit Log

A detailed log of every authenticated interaction.

- Time
- Requestor
- Request
- Response

```
$ cat log/vault_audit.log | jq -s "[-1]"
```

```
$ cat log/vault_audit.log | jq -s "[-1].request"
{
```



```
"id": "70419a8b-d904-542b-fe48-61d8f869a0b7",
```

```
"operation": "update",
```

operation maps to capability



```
"mount_type": "transit",
```

```
...
```

```
"path": "transit/keys/app-auth",
```

```
"remote_address": "127.0"
```

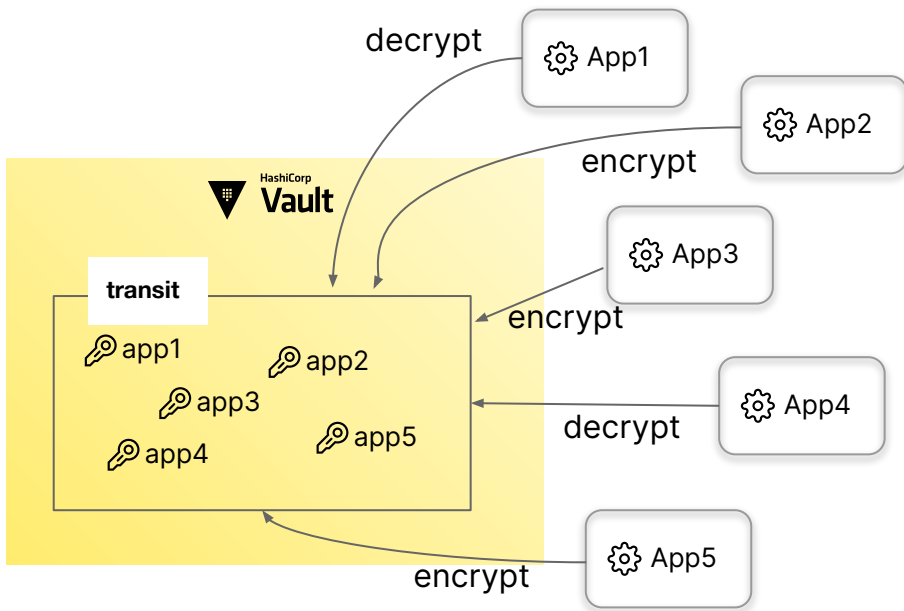
path

```
path "transit/keys/app-auth" {
  capabilities = [ "update" ]
}
```



# Example Scenario

Using transit secrets engine for data encryption/decryption



- Each application has its own encryption key
- Challenge:
  - The number of applications will grow
  - You cannot foresee the name of future applications to be developed

# Policy Solutions

Using transit secrets engine for data encryption/decryption

Solution 1

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

Is it good enough?

- Trouble points:

The policy is **too open**. App1 can access app4 encryption key.

Solution 2

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

Is this easy to scale and maintain?

- Trouble points:

You have to write policies for each app.

# ACL Templating

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

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

# Available Templating Parameters (1 of 2)

Parameter	Description
<b>identity.entity.id</b>	The entity's ID
<b>identity.entity.name</b>	The entity's name
<b>identity.entity.metadata.&lt;&lt;metadata key&gt;&gt;</b>	Metadata associated with the entity for the given key
<b>identity.entity.aliases.&lt;&lt;mount accessor&gt;&gt;.id</b>	Entity alias ID for the given mount
<b>identity.entity.aliases.&lt;&lt;mount accessor&gt;&gt;.name</b>	Entity alias name for the given mount
<b>identity.entity.aliases.&lt;&lt;mount accessor&gt;&gt;.metadata.&lt;&lt;metadata key&gt;&gt;</b>	Metadata associated with the alias for the given mount and metadata key

# Available Templating Parameters (2 of 2)

Parameter	Description
<b>identity.groups.ids.&lt;&lt;group id&gt;&gt;.name</b>	The group name for the given group ID
<b>identity.groups.names.&lt;&lt;group name&gt;&gt;.id</b>	The group ID for the given group name
<b>identity.groups.names.&lt;&lt;group id&gt;&gt;.metadata.&lt;&lt;metadata key&gt;&gt;</b>	Metadata associated with the group for the given key
<b>identity.groups.names.&lt;&lt;group name&gt;&gt;.metadata.&lt;&lt;metadata key&gt;&gt;</b>	Metadata associated with the group for the given key

# Token policies & Identity Policies

```
$ vault token lookup
```

Key	Value
accessor	yOMHJzMZ5Krz7BSr0tF2ZzC2
creation_time	1622087787
creation_ttl	768h
display_name	userpass-bob
entity_id	bf3ea189-61a1-d7...snip...
expire_time	2021-06-28T<time_stamp>
explicit_max_ttl	0s
external_namespace_policies	map[ ]
id	s.UYkAjU6ak70qwQ40CmLP3uyT
identity_policies	[base]
issue_time	2021-05-27T<time_stamp>
meta	map[username:bob]
num_uses	0
orphan	true
path	auth/userpass/login/bob
policies	[default test]
...snip...	

# ACL Templating with Identity Entity Names

```
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}}`

# 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

```
path
"auth/ldap/groups/{{identity.groups.ids.fb036ebc-2f62-4124-9
503.name}}" {

    capabilities = [ "update", "read" ]

}
```

```
path
"secret/data/groups/{{identity.groups.names.education.metada
ta.product}}/*" {

    capabilities = [ "create", "update", "read", "delete" ]

}
```



# CLI

vault

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



```
$ vault policy list
```

```
default
```

```
root
```

list all policies



```
$ vault policy read default
```

```
...
```

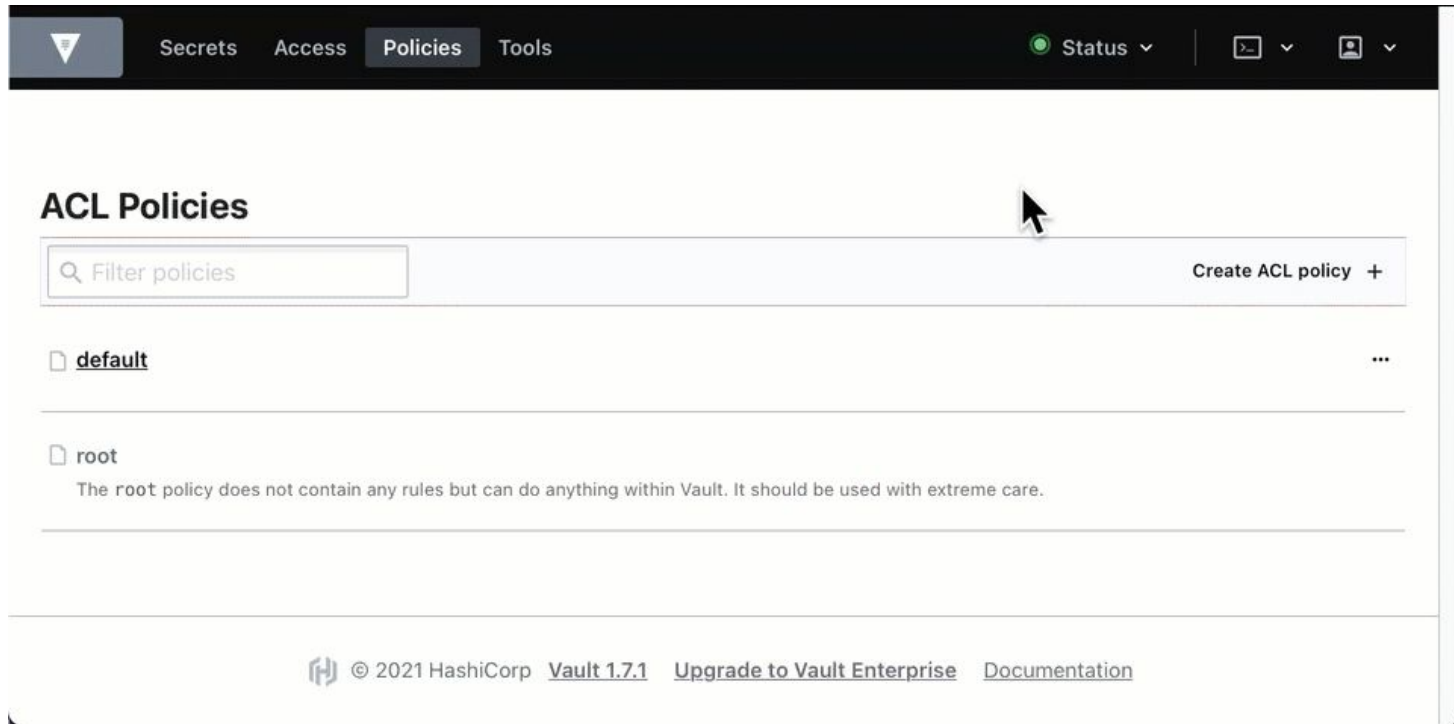
show policy



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

create or  
update policy

# Vault UI



# root and default policies

Vault Enterprise has 2 “out of the box” policies

- **root** policy grants access to everything, a root token (generated when Vault was unsealed or using recovery keys) is the only token able to grant this policy to other tokens
- **default** policy grants a lot of the bare minimum operations

# Associate policies

Upon successful authentication, the generated token will have the policies attached

```
# LDAP group, "sre" has "dev" & "ops" policies attached  
$ vault write auth/ldap/groups/sre policies="dev, ops" 
```

# Next Steps

# Tutorials

Step-by-step guides to accelerate deployment of Vault

The screenshot shows the HashiCorp Vault Developer Tutorials page. The header includes the Vault logo, navigation links (Home, Documentation, Tutorials, Install, Try Cloud), a search bar, and a user profile icon. The left sidebar contains a navigation menu with sections: Vault Home, Tutorials (Overview, Get Started, CLI Quick Start, HCP Vault Quick Start, UI Quick Start), Use Cases (ADP, Database Credentials, Data Encryption, Key Management, Secrets Management), Certification Prep (Associate, Operations Pro), and Production (Day One Preparation, Enterprise, HCP Vault Monitoring, Monitor & Troubleshoot, Recommended Patterns, Standard Procedures). The main content area is titled 'Developer / Vault / Tutorials' and features a yellow banner with the text 'Centrally store, access and deploy secrets' and a link 'Explore HCP Vault →'. Below this are three 'Get Started' tutorial cards: 'Getting Started' (13 tutorials), 'Getting Started with Vault UI' (8 tutorials), and 'Getting Started with HCP Vault' (9 tutorials). The 'Featured Tutorials' section includes 'Vault 1.12 Release Highlights' (7 tutorials), 'HCP Vault Operations' (12 tutorials), and 'Security' (5 tutorials). The 'New Tutorials' section lists 'Vault Installation to Minikube via Helm with TLS enabled' (13min) and 'Audit Device Logs and Incident Response with Elasticsearch' (27min).

Developer / Vault / Tutorials

Centrally store, access and deploy secrets  
Explore HCP Vault →

**Get Started**

- 13 tutorials  
**Getting Started**  
Vault secures, stores, and tightly controls access to tokens, passwords, certificates, API keys,...
- 8 tutorials  
**Getting Started with Vault UI**  
Manage Vault environment as well as your secrets using Vault UI.
- 9 tutorials  
**Getting Started with HCP Vault**  
Quickly get hands-on with HashiCorp Cloud Platform (HCP) Vault using the HCP portal and setup your manage...

**Featured Tutorials**  
Extend your knowledge of Vault features and use cases.

- 7 tutorials  
**Vault 1.12 Release Highlights**  
The listed tutorials were updated to showcase the new enhancements introduced in Vault 1.12.
- 12 tutorials  
**HCP Vault Operations**  
Learn how to provision and connect to HCP Vault clusters.
- 5 tutorials  
**Security**  
Implement security best practices to secure your applications, protect your network, manage sensitive data...

**New Tutorials**  
Here are the most recently published tutorials.

- 13min  
**Vault Installation to Minikube via Helm with TLS enabled**  
Deploy Vault on Kubernetes locally with TLS using Minikube and the official Helm chart.
- 27min  
**Audit Device Logs and Incident Response with Elasticsearch**  
Learn how to use a Vault audit device with Elasticsearch for auditing specific critical Vault events as part of your incident response workflows.

<https://developer.hashicorp.com/vault/tutorials>

# Resources

- [Vault Namespace and Mount Structuring Guide](#)
- [Mount Move Tutorial](#)
- [Vault Authentication Tutorial](#)
- [Vault OIDC with Okta](#)
- [Vault OIDC with Azure AD](#)
- [Vault Policy Tutorial](#)
- [Templated Policies](#)

# Need Additional Help?

## Customer Success

Contact our Customer Success Management team with any questions. We will help coordinate the right resources for you to get your questions answered  
[customer.success@hashicorp.com](mailto:customer.success@hashicorp.com)

## Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at  
[support.hashicorp.com](https://support.hashicorp.com)

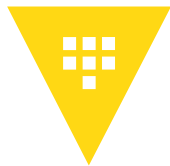
## Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers  
[discuss.hashicorp.com](https://discuss.hashicorp.com)



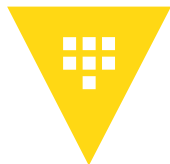


# Upcoming Webinars



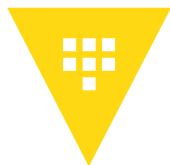
## Office Hours

An open forum with Vault Subject Matter Experts to answer questions that have arisen during the program and your deployment



## Vault Operations Basics & Best Practices

Learn best practices for deploying Telemetry & Monitoring, Disaster Recovery, and Runbooks



## Consuming Vault with Applications

Learn about secret distribution patterns and best practices for integrating Vault secrets with applications including Vault Agent and client libraries

# Action Items

- Share to [customer.success@hashicorp.com](mailto:customer.success@hashicorp.com)
  - Authorized technical contacts for support
  - Stakeholders contact information (name and email addresses)
- Begin designing the namespace structure for your organization's Vault instance(s)
- Configure an authentication method and policy for your Vault Administrators

# Q&A



# Thank you

[customer.success@hashicorp.com](mailto:customer.success@hashicorp.com)

[www.hashicorp.com/customer-success](http://www.hashicorp.com/customer-success)