



Vault Namespaces, Authentication, & ACL Policies

February 2023

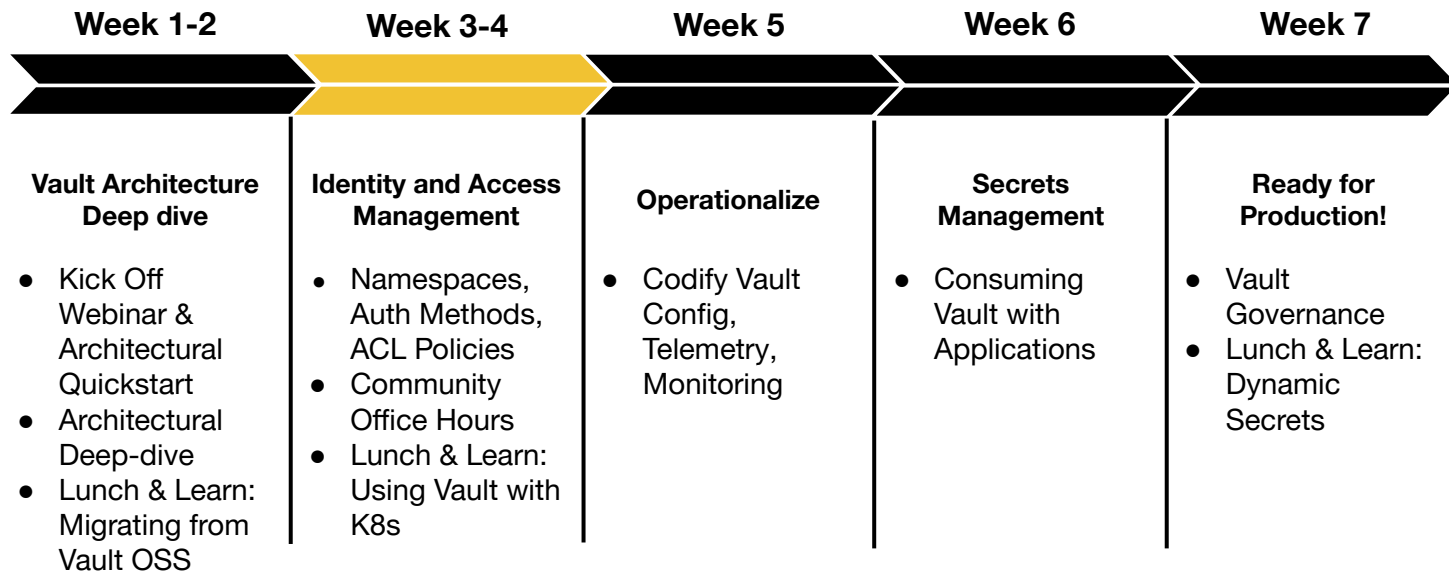
Copyright © 2021 HashiCorp



Agenda

1. Namespaces
2. Authentication
3. Policies
4. Next Steps

Vault Enterprise Path to Production



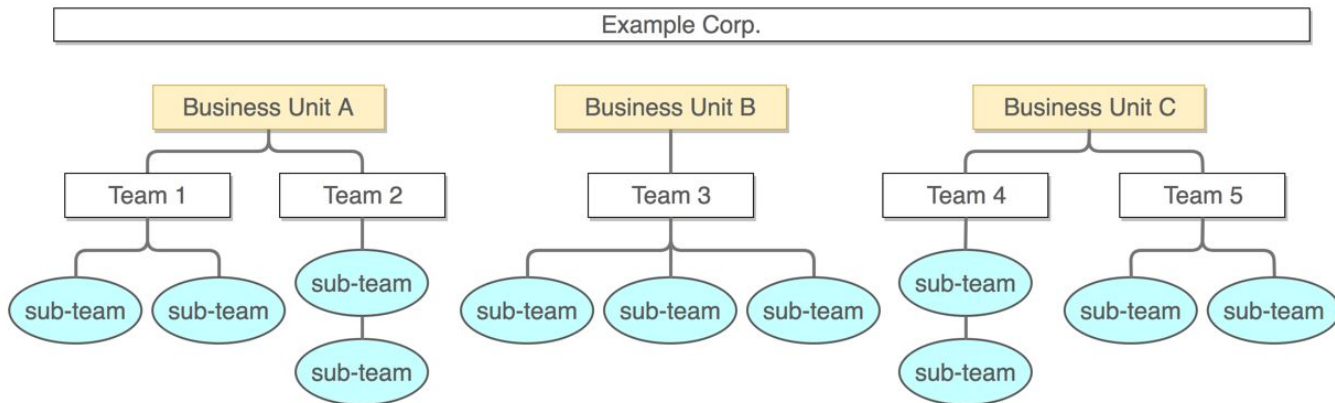
— 01

Namespaces

Namespaces



- Namespaces create isolated “Vaults within a Vault” so that Vault can be provided as a service in an organization
- Each namespace will maintain its own path structure which allows for the delegated administration of policies and secrets management to teams while controlling the blast radius by isolating control within their namespace



Namespace Contents



Unique to each namespace

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

Namespace Considerations



Requirement

What to Consider

Organizational Structure	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?
Self-Service Requirements	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.

Namespace Considerations

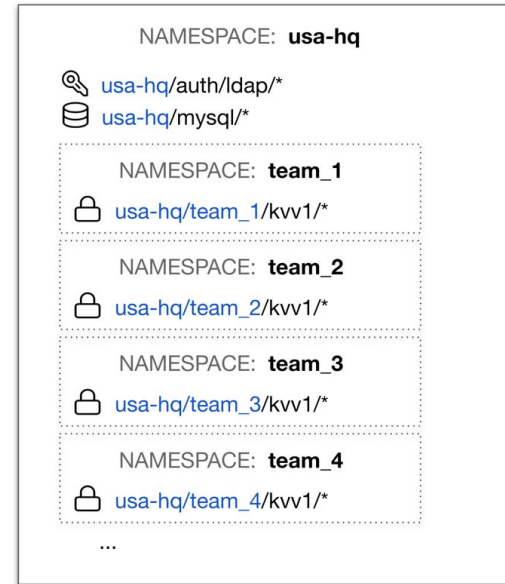


Requirement	What to Consider
Audit Requirements	What are the requirements around auditing usage of Vault within your organization?
	Is there a need to regularly certify access to secrets?
	Is there a need to review and/or decommission stale secrets or auth roles?
	Is there a need to determine chargeback amounts to internal customers?
Secrets Engine Requirements	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

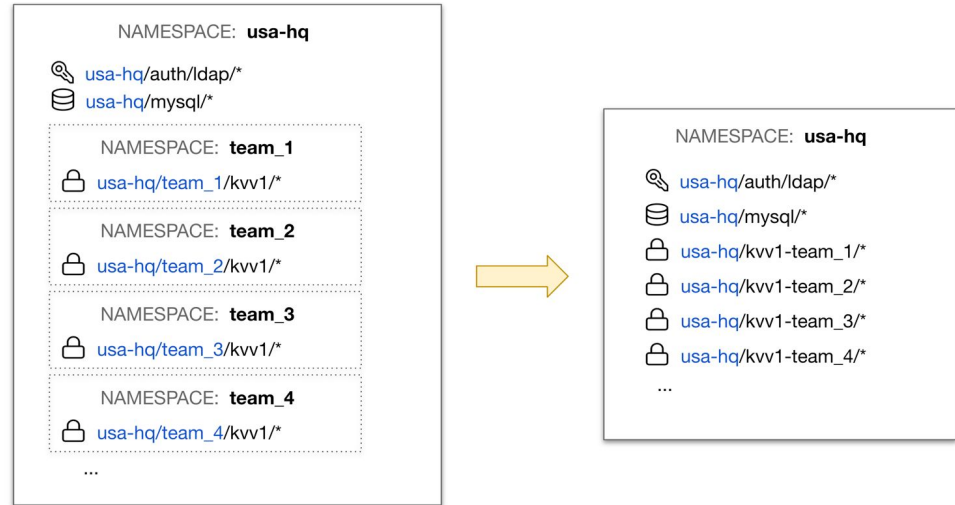


Anti-pattern

Using Namespaces



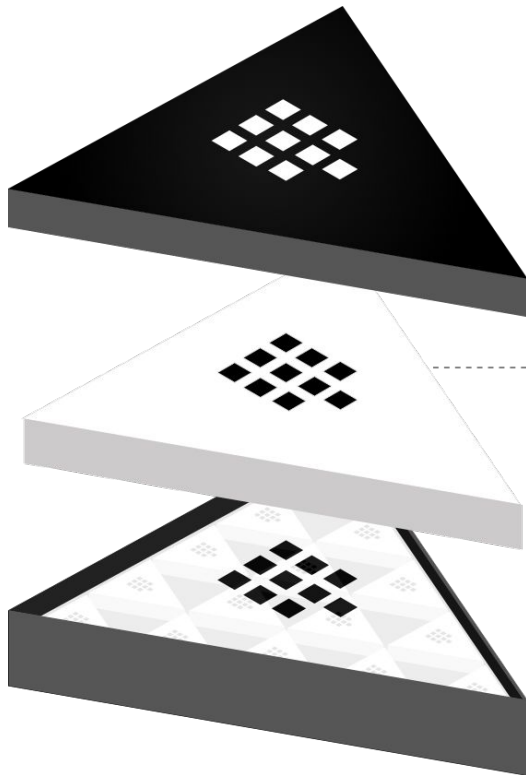
- Instead of providing self-service by implementing many namespaces we recommend implementing an onboarding layer
- Shifting the administrative boundary from teams to the onboarding layer reduces the number of namespaces while enforcing a standard naming convention, secrets path structure, and templated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



Best Practice



Root Namespace



- **Root (Namespace)**

Members:
Security Team

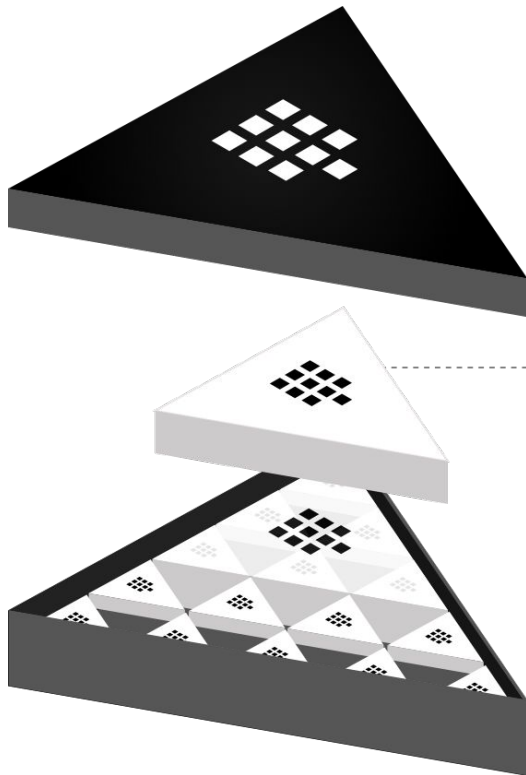
Namespace Specific Configuration:
Defined global member access
Defined global authentication mounts
Defined global secrets engines



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:

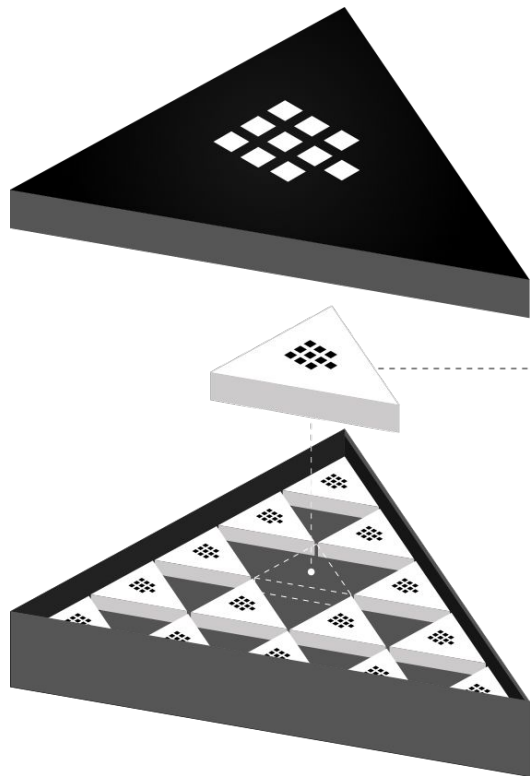
Defined engineering member access
Defined engineering authentication mounts
Defined engineering secrets engines



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 sales

# List namespaces from within root namespace
> vault namespace list

# List child namespaces for usa-hq namespace
> vault namespace list -namespace=usa-hq

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



Getting Started with Namespaces

API

```

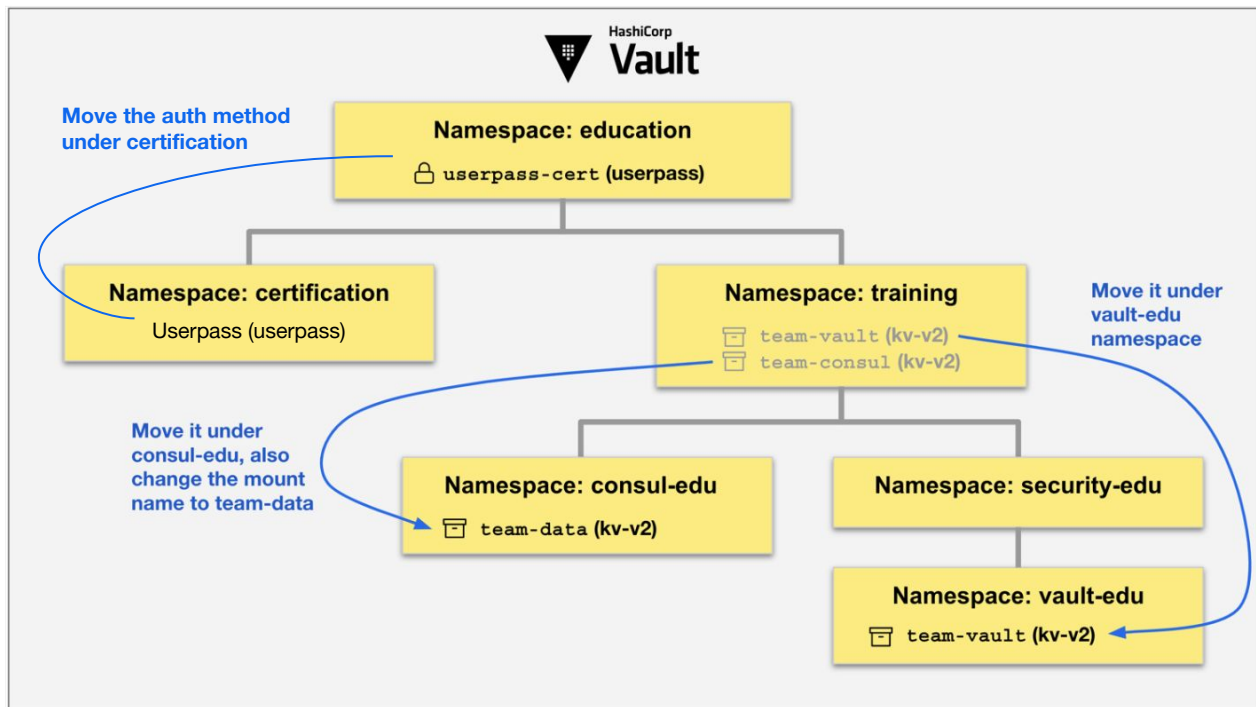
# Create namespace
> curl --header "X-Vault-Token: <TOKEN>" \
--request POST \
https://<vault_addr>/v1/sys/namespaces/usa-hq

# Create child namespaces
> curl --header "X-Vault-Token: <TOKEN>" \
--header "X-Vault-Namespace: usa-hq" --request POST \
https://<vault_addr>/v1/sys/namespaces/sales
```

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

TERMINAL

```
# 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

> vault namespace create sales
```

— 02

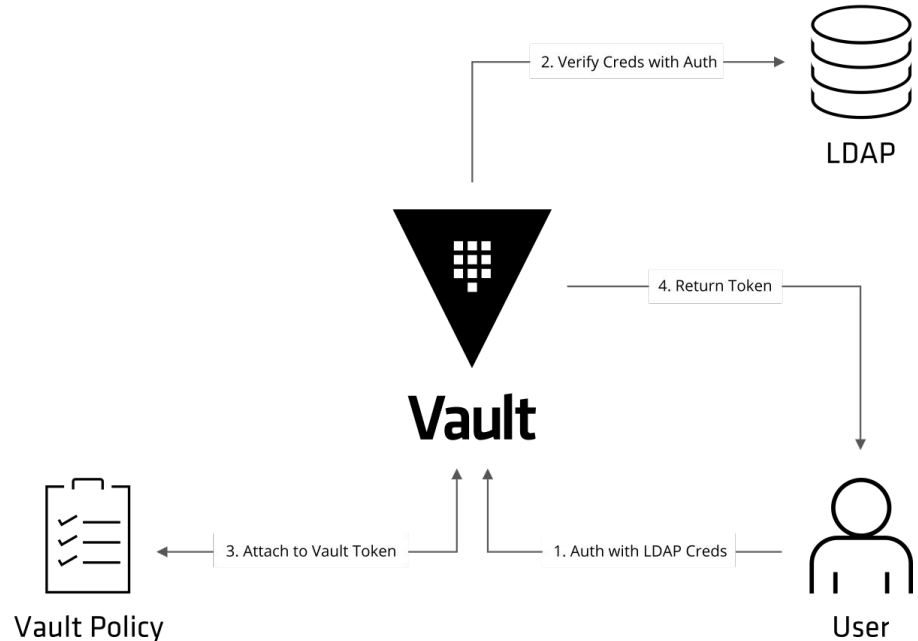
Authentication

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



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

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

A Vault client can be mapped as an entity

An entity can have multiple aliases



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**

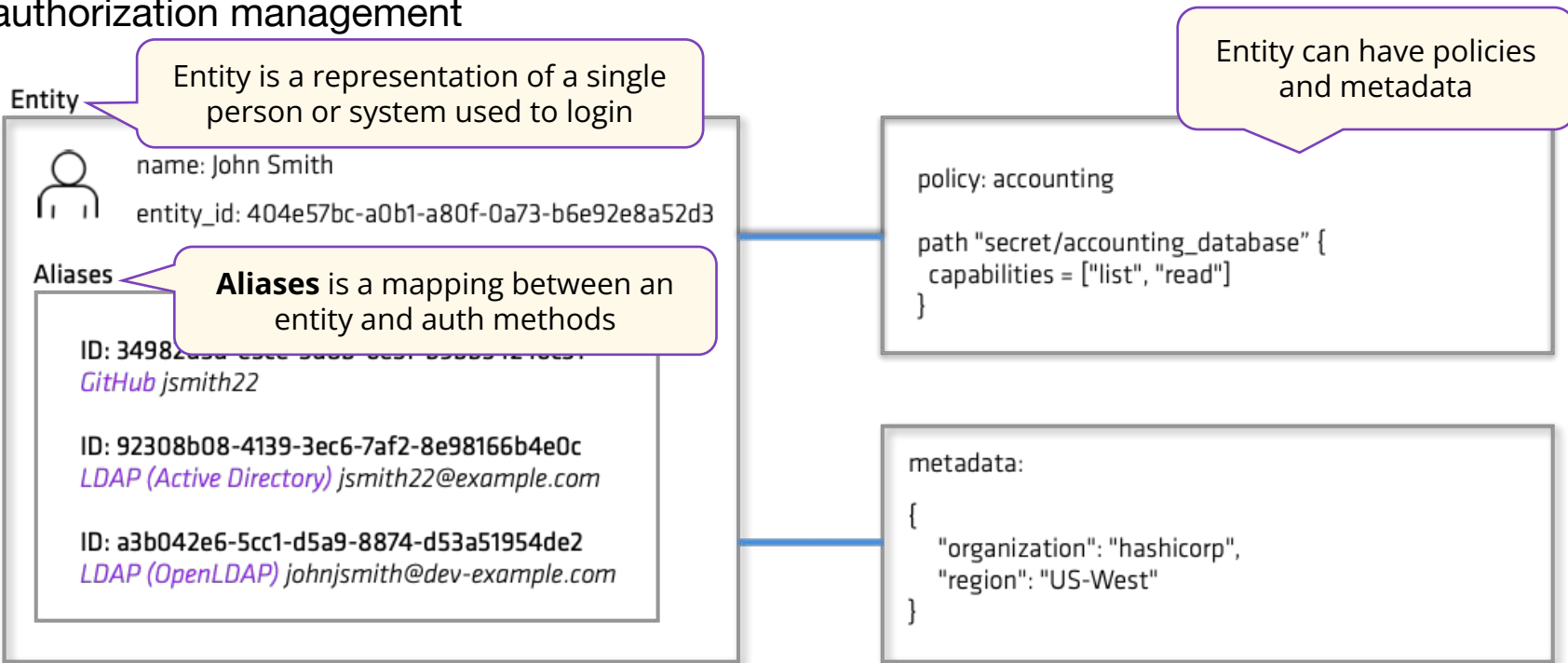
Aliases:

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

Entities and Aliases



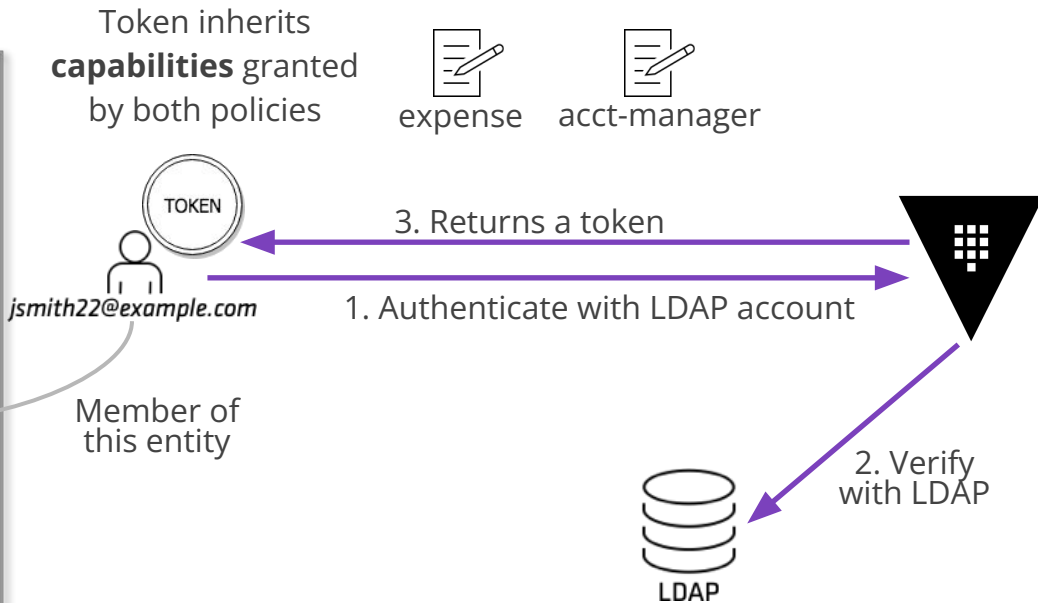
Map multiple user authentication schemes to a single entity to provide for more efficient authorization management



Token and Policies



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



Basic Workflow



Step 1: Enable auth methods



Step 2: Configure each auth method



Step 3: Create an Entity



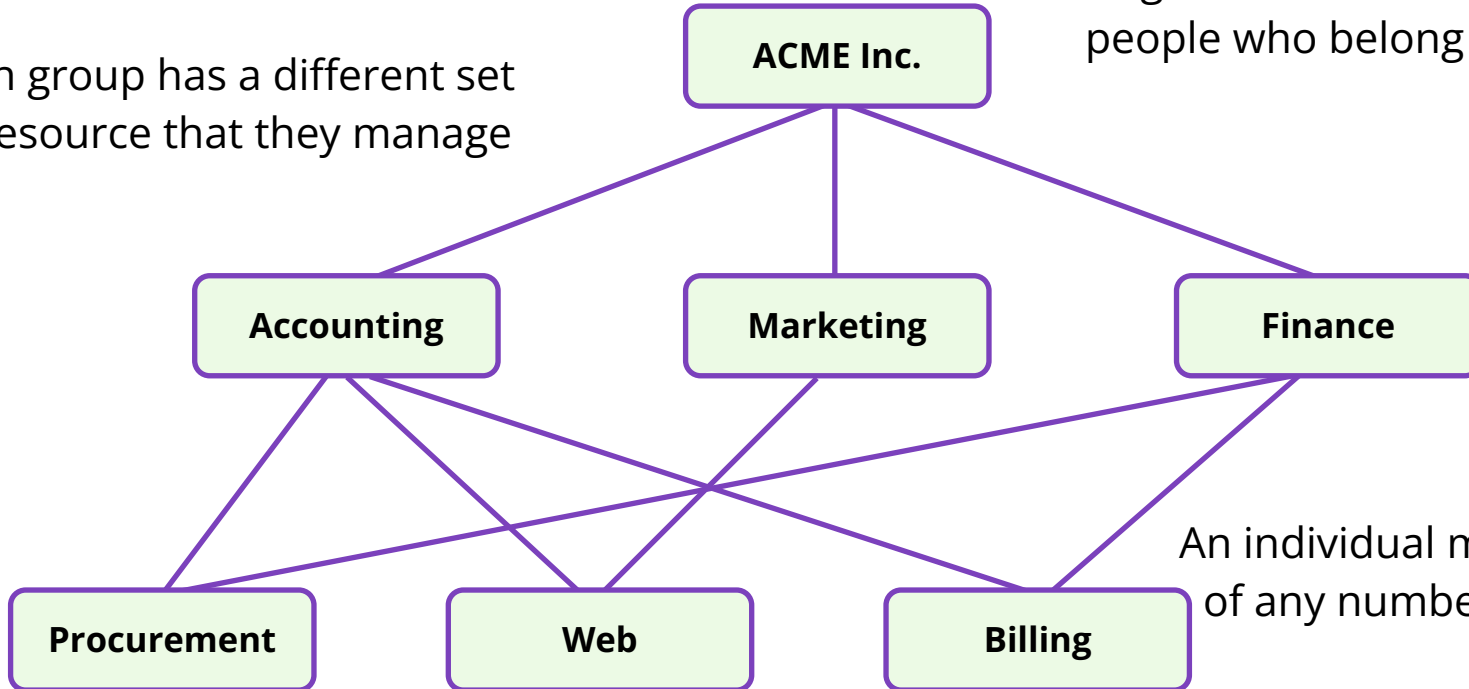
Step 4: Create the Entity Aliases

Organizational Structure



Each group has a different set of resource that they manage

Organization is composed of people who belong in groups



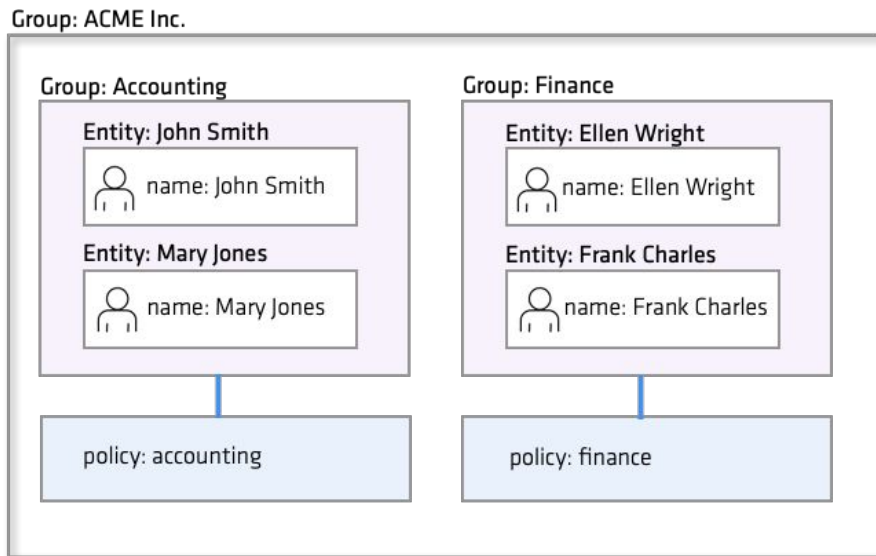
An individual may be a part of any number of groups

Identity Groups

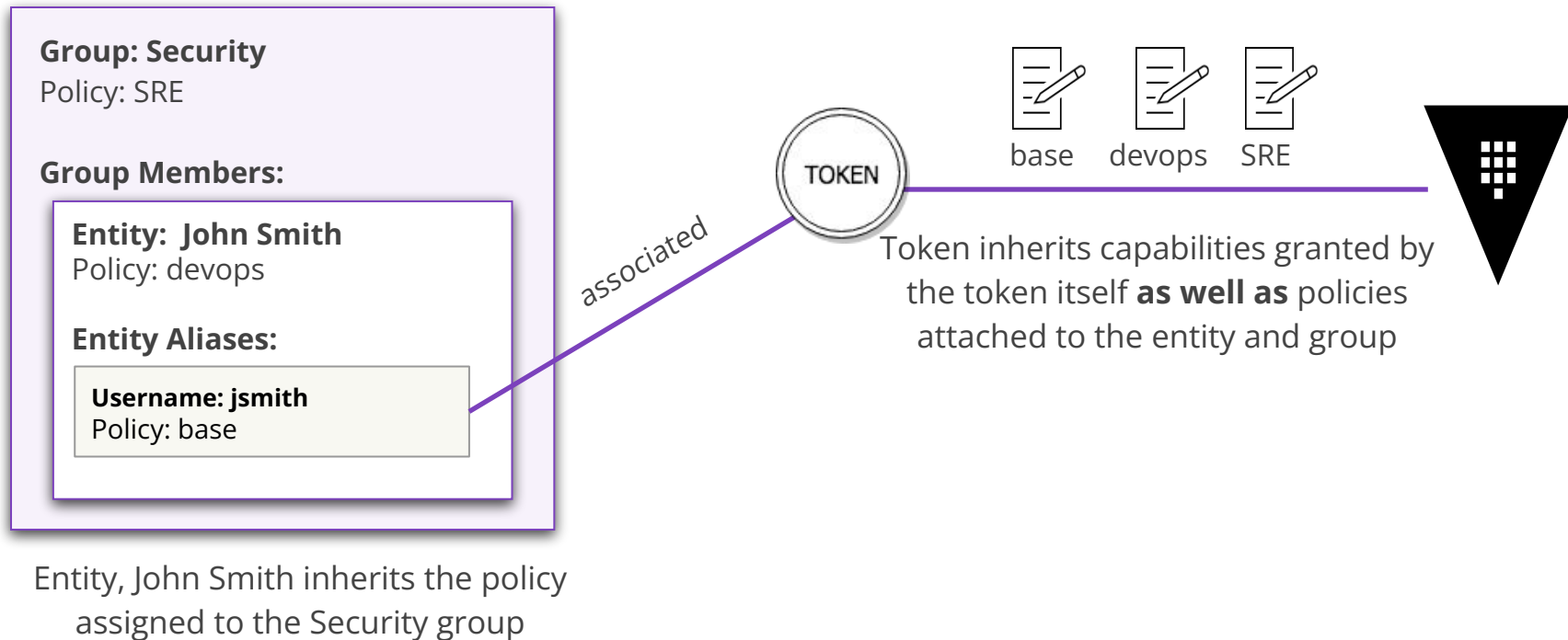


Identity group maps multiple user entities to a group for authorization management at scale.

- 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



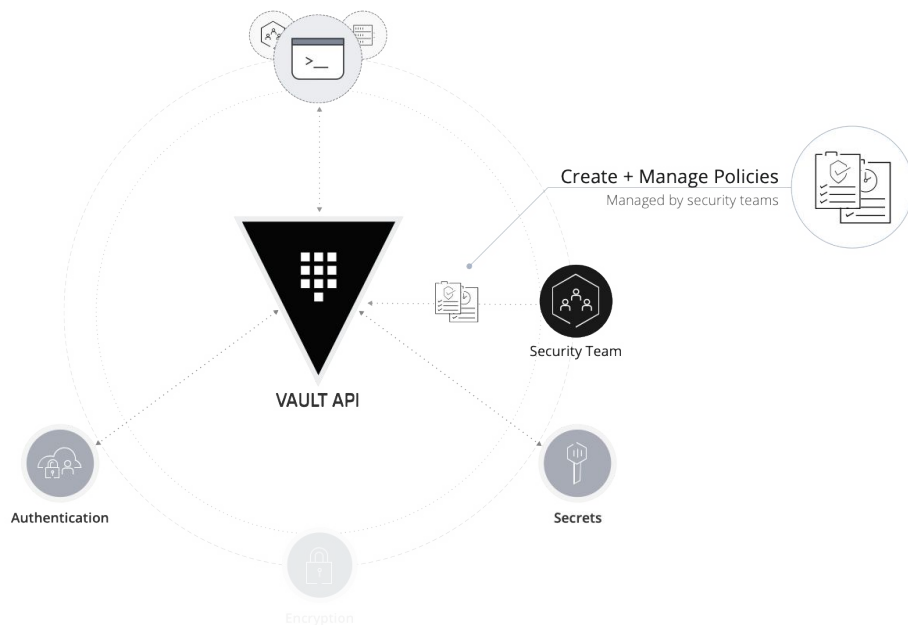
— 03

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

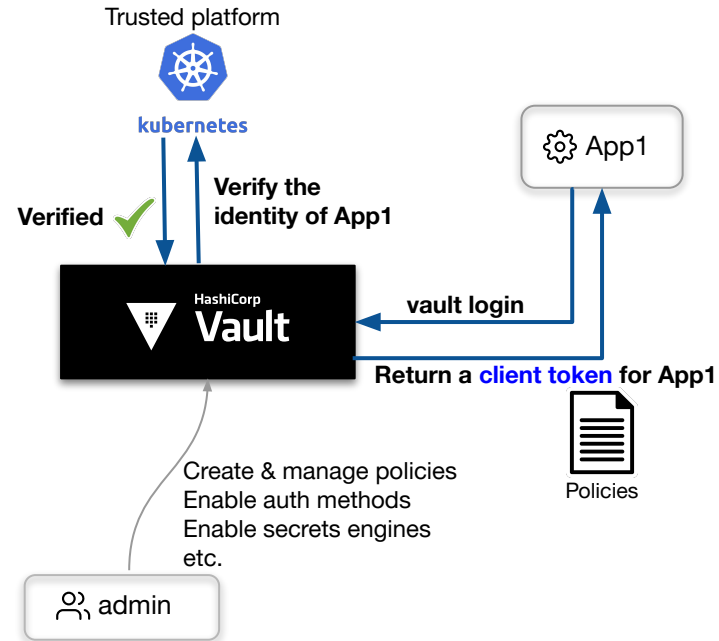


Vault Policies and Client Tokens



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

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

Example path

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



Policies

path

capabilities

```
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](#)

Jump to section ▾ Show terminal

Docs Forum Bookmark

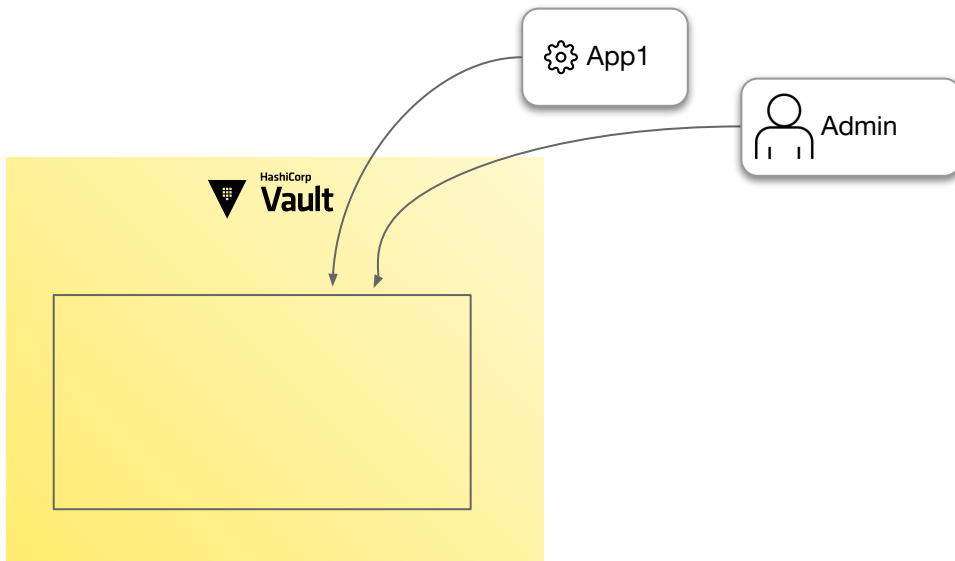
Root protected API endpoints

The following paths requires a root token or `sudo` capability in the policy:

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

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

TERMINAL

```
$ 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



Audit Log

A detailed log of every authenticated interaction

- Time
- Requestor
- Request
- Response

```
TERMINAL

$ 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",
  "mount_type": "transit"
  ...
  "path": "transit/keys/app-auth",
  "remote_address": "127.0.0.1"
}
```

operation maps to capability

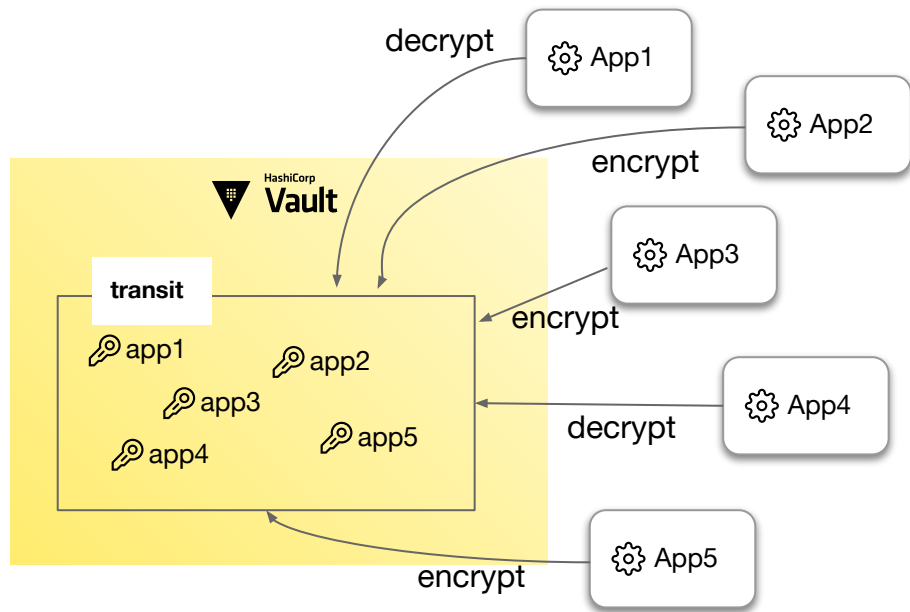
path

```
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



```
CODE EDITOR

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

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

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

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
<code>identity.groups.ids.<<group id>>.name</code>	The group name for the given group ID
<code>identity.groups.names.<<group name>>.id</code>	The group ID for the given group name
<code>identity.groups.names.<<group id>>.metadata.<<metadata key>></code>	Metadata associated with the group for the given key
<code>identity.groups.names.<<group name>>.metadata.<<metadata key>></code>	Metadata associated with the group for the given key



Token Policies & Identity Policies

```
$ vault token lookup
```

Key	Value
accessor	yOMHJzMZ5Krz7BSrOtF2ZzC2
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.UYkAjU6ak70qwQ4OCmLP3uyT
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

```
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}}`

ACL Templating with Identity Groups



```
CODE EDITOR

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

path
"secret/data/groups/{{identity.groups.names.education.metadata.product}}/*" {
    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



CLI

vault

```
TERMINAL

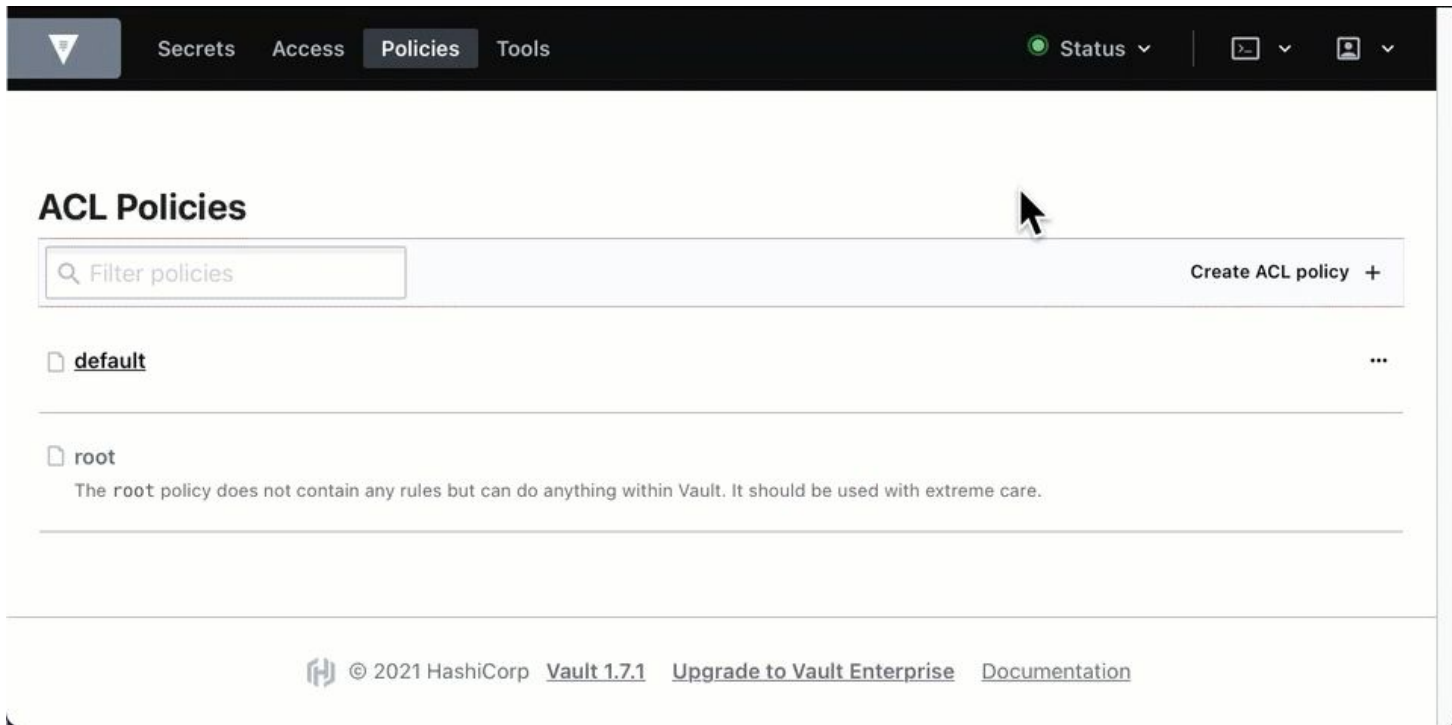
$ vault policy --help

$ 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

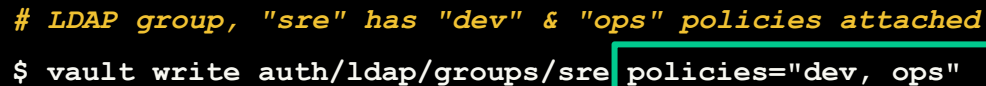
There are two out of the box policies:

- **root** policy grants access to everything. A root token was generated when Vault was unsealed and is the only token that is 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



TERMINAL

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

The terminal window has a dark background. The title bar shows three window control buttons (red, yellow, green) on the left and the word "TERMINAL" on the right. The command prompt is a yellow dollar sign. The command is in white text. The output is in yellow text. A green box highlights the command, and a green eye icon is on the right side of the box.

— 04

Next Steps

Tutorials

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



Step-by-step guides to accelerate deployment of Vault

The screenshot shows the HashiCorp Vault Developer Tutorials page. The page has a dark header with the HashiCorp logo, navigation links (Vault, Install, Tutorials, Documentation, API, Integrations, Try Cloud), a search bar, and a user profile icon. The left sidebar contains a 'Vault Home' link, a 'Tutorials' section with a dropdown arrow, and a list of links: 'Get Started', 'CLI Quick Start', 'HCP Vault Quick Start', 'UI Quick Start', 'Use Cases' (with sub-links: 'ADP', 'Database Credentials', 'Data Encryption', 'Key Management', 'Secrets Management'), and 'Certification Prep' (with sub-link: 'Associate'). The main content area has a breadcrumb 'Developer / Vault / Tutorials' and a large yellow banner with the text 'Centrally store, access and deploy secrets' and a link 'Explore HCP Vault →'. Below this is a 'Get Started' section with three cards: 'Getting Started' (13 tutorials), 'Getting Started with Vault UI' (8 tutorials), and 'Getting Started with HCP Vault' (9 tutorials). The 'New Tutorials' section follows, stating 'Here are the most recently published tutorials.' and showing two cards: 'Disaster Recovery Replication Failover and Failback' (54min) and 'Vault Installation to Minikube via Helm with TLS enabled' (13min).

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...

New Tutorials

Here are the most recently published tutorials.

- 54min
Disaster Recovery Replication Failover and Failback
Learn how to failover in a Disaster Recovery Replication environment and then failback to original operating state.
- 13min
Vault Installation to Minikube via Helm with TLS enabled
Deploy Vault on Kubernetes locally with TLS using Minikube and the official Helm chart.

On this page

- Get Started
- New Tutorials
- HashiCorp Well-Architected ...
- Popular Topics
- All 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

Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at support.hashicorp.com.

Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers

discuss.hashicorp.com

Upcoming Webinars



Vault

Using Vault with Kubernetes

This Lunch & Learn (separate link) covers the best practices for integrating Vault Enterprise with Kubernetes and

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

Action Items



Vault 

- Share to customer.success@hashicorp.com
 - Authorized technical contacts for support
 - Stakeholders contact information (name and email addresses)
- Determine your implementation plan and pattern(s) for Vault namespaces
- Begin developing foundational Vault policy, and an Admin policy so the root token can be revoked



Thank You

customer.success@hashicorp.com

www.hashicorp.com