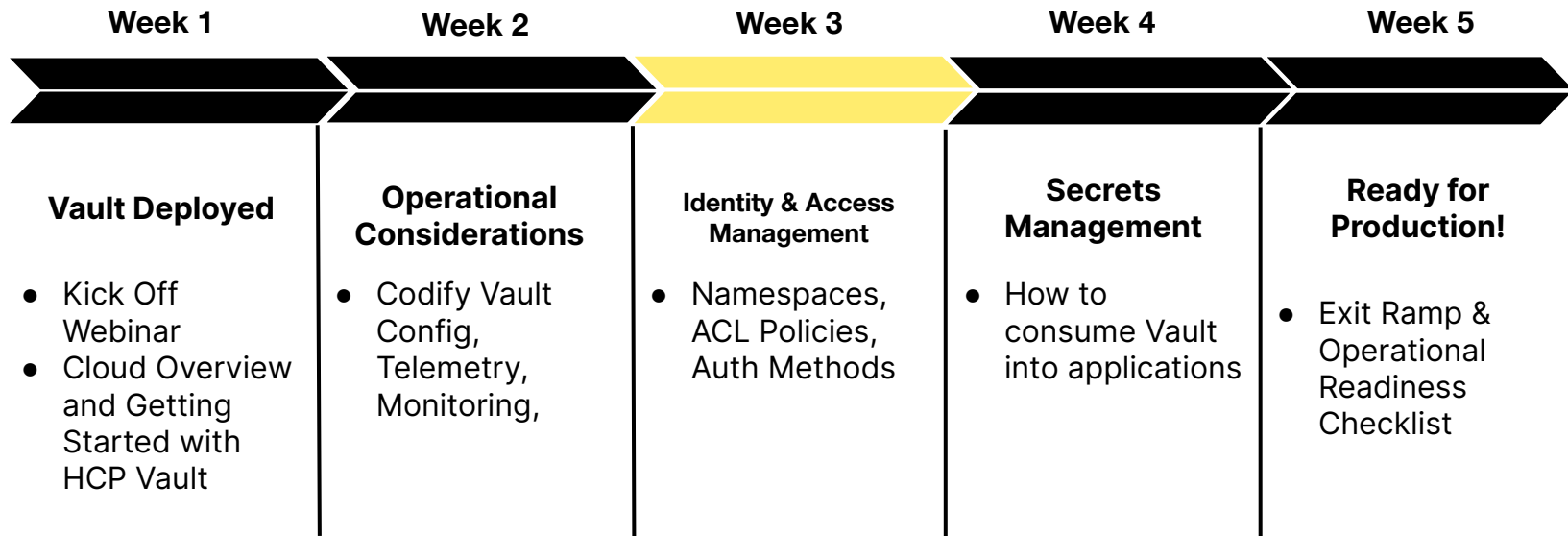# HCP Vault Namespaces, Authentication, & Policies

Agenda

HCP Vault Onboarding Program

A 5 week guided community environment
Assisting customers with onboarding and adoption

| Week 1	Week 2	Week 3	Week 4	Week 5
Vault Deployed	**Operational Considerations**	**Identity & Access Management**	**Secrets Management**	**Ready for Production!**
• Kick Off Webinar • Cloud Overview and Getting Started with HCP Vault	• Codify Vault Config, Telemetry, Monitoring,	• Namespaces, ACL Policies, Auth Methods	• How to consume Vault into applications	• Exit Ramp & Operational Readiness Checklist

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
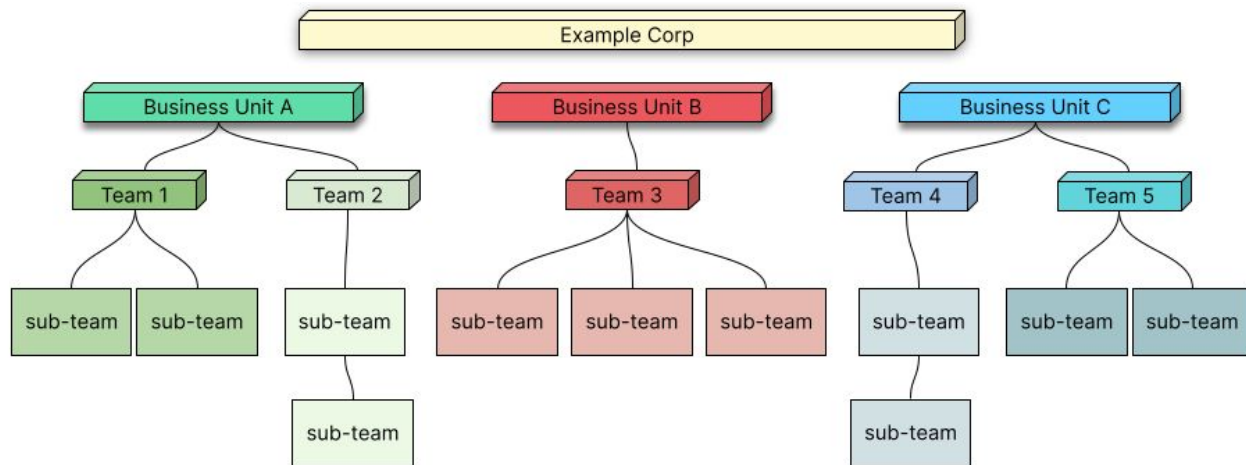
Namespace Contents

Unique to each namespace

- Policies

- Secrets Engines

- Authentication Methods

- Tokens

- Identity Entities and Groups

Considerations

Requirement	What to Consider
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 templatized? For example, configuration by code, Git flows, the Terraform Vault provider, custom onboarding layers, or some combination of these.

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

NAMESPACE: **usa-hq**

usa-hq/auth/ldap/*
usa-hq/mysql/*

NAMESPACE: **team_1**
usa-hq/team_1/kvv1/*

NAMESPACE: **team_2**
usa-hq/team_2/kvv1/*

NAMESPACE: **team_3**
usa-hq/team_3/kvv1/*

NAMESPACE: **team_4**
usa-hq/team_4/kvv1/*

...

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
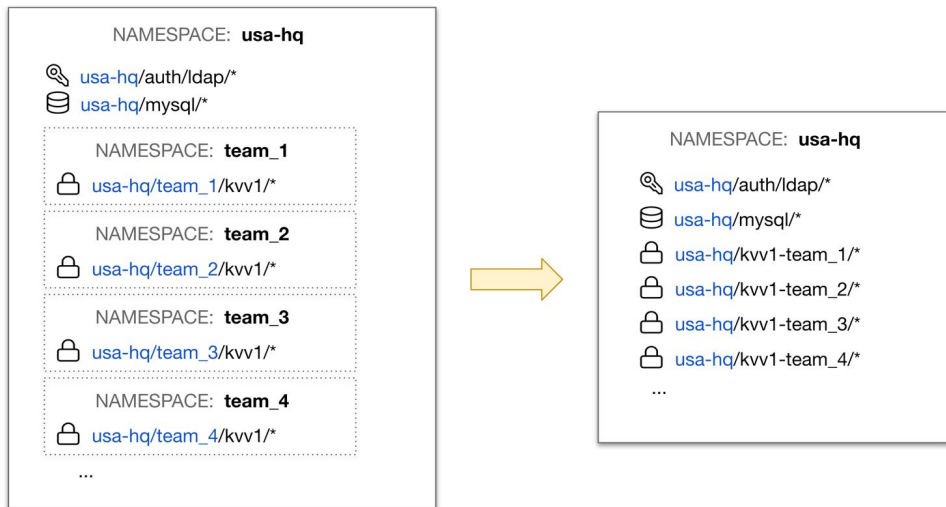
Namespaces for Teams and Groups

HCP Vault Production Cluster

Engineering Team Namespace

Secrets Engines   Auth Methods

Policies

Cloud Team Namespace

Secrets Engines   Auth Methods

Policies
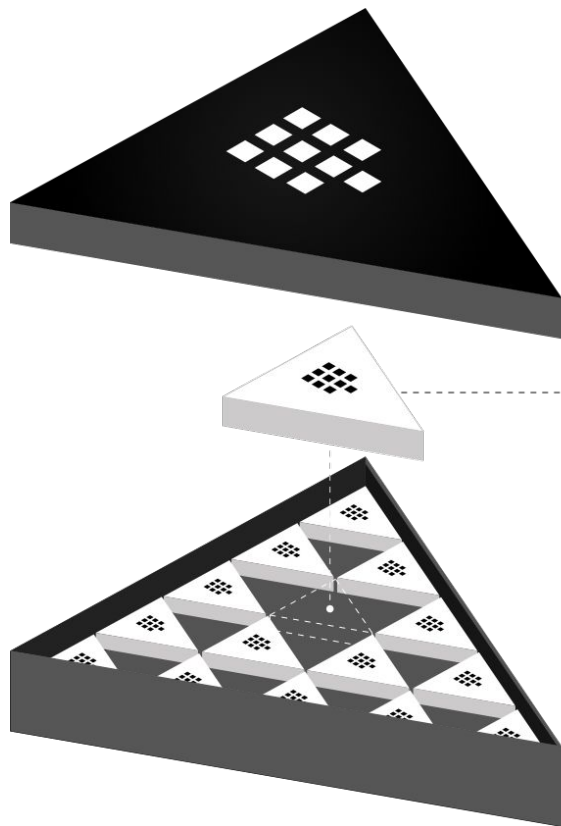
Developer Team

Secrets Engines

Policies

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

```
# Set VAULT_NAMESPACE environment variable to admin
$ export VAULT_NAMESPACE=admin

# Create namespace
$ vault namespace create usa-hq

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

# List namespaces from within admin namespace
$ vault namespace list -namespace=admin
  Keys
  ----
  education/
  eu-hq/
  usa-hq/

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

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"
```

Getting Started with Namespaces

```
/admin/

/admin/education/
    •   Secrets
    •   Policy
    •   Auth
    •   Tokens

/admin/eu-hq/
    •   Secrets
    •   Policy
    •   Auth
    •   Tokens

/admin/eu-hq/team-a/
    •   Secrets
    •   Policy
    •   Auth
    •   Tokens

/admin/usa-hq
```
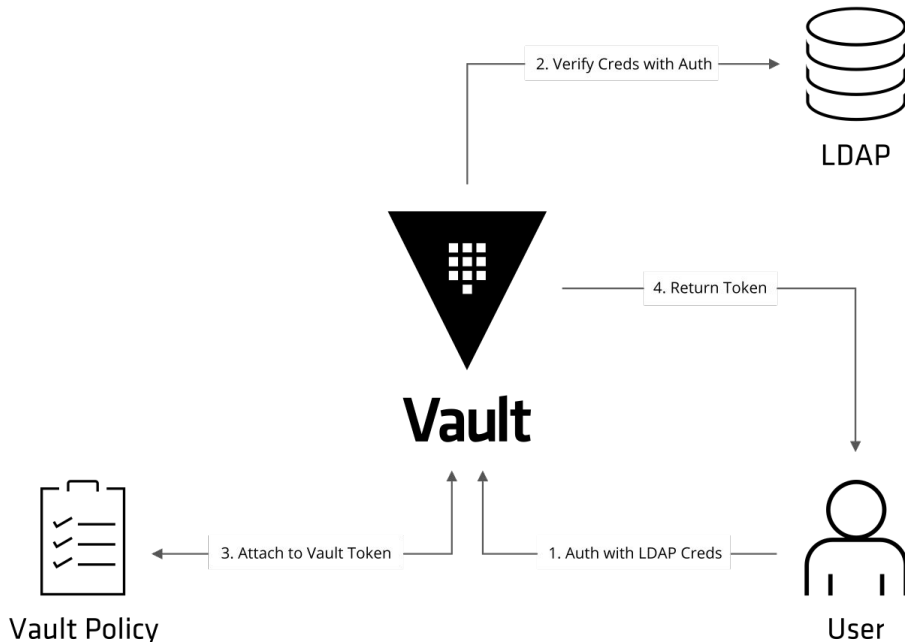
Authentication

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

2. Verify Creds with Auth

LDAP

4. Return Token

Vault

3. Attach to Vault Token

1. Auth with LDAP Creds

Vault Policy

User

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

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

Also, a **group** can have subgroups

Group Name: Accounting
Group ID: 0bfed703-f07d-2965...
Policies: **accounting**

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

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

An entity can have multiple aliases

Entity Name: HCP Billing
Entity ID:
Iw23p85c-2e9-b...
Policies: **billing**

Aliases:

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

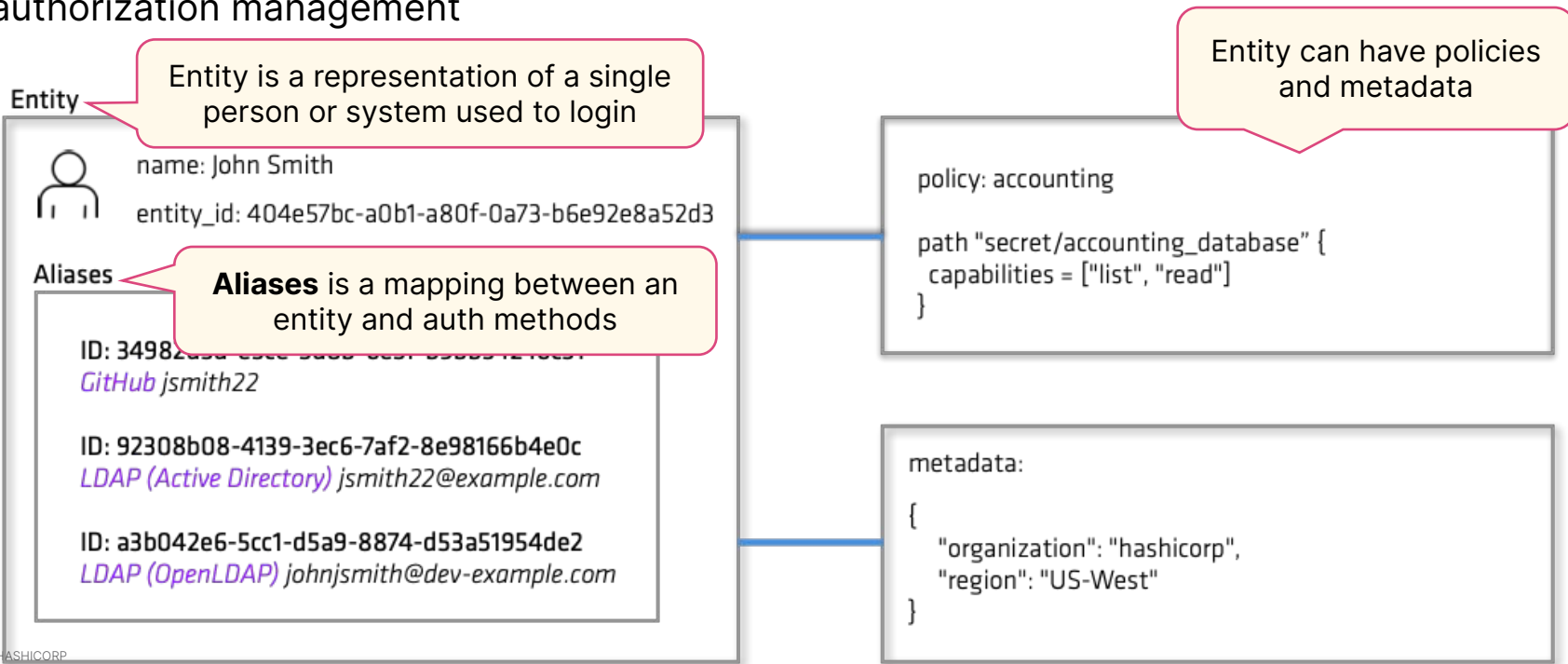Group Name: Payment
Group ID:
I273k85c-2e9-b...
Policies: **payment**

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

Entity is a representation of a single person or system used to login

Entity can have policies and metadata

Entity

name: John Smith

entity_id: 404e57bc-a0b1-a80f-0a73-b6e92e8a52d3

Aliases

Aliases is a mapping between an entity and auth methods

ID: 34982d9d-e9e9-9d6b-9e9f-b99b9f21f8f9
GitHub jsmith22

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

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

```
policy: accounting

path "secret/accounting_database" {
  capabilities = ["list", "read"]
}
```

```
metadata:

{
    "organization": "hashicorp",
    "region": "US-West"
}
```

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

TOKEN

jsmith22@example.com

Member of this entity

3. Returns a token

1. Authenticate with LDAP account

2. Verify with LDAP

LDAP

Basic Workflow

Step 1: Enable auth methods

⬇

Step 2: Configure each auth method

⬇

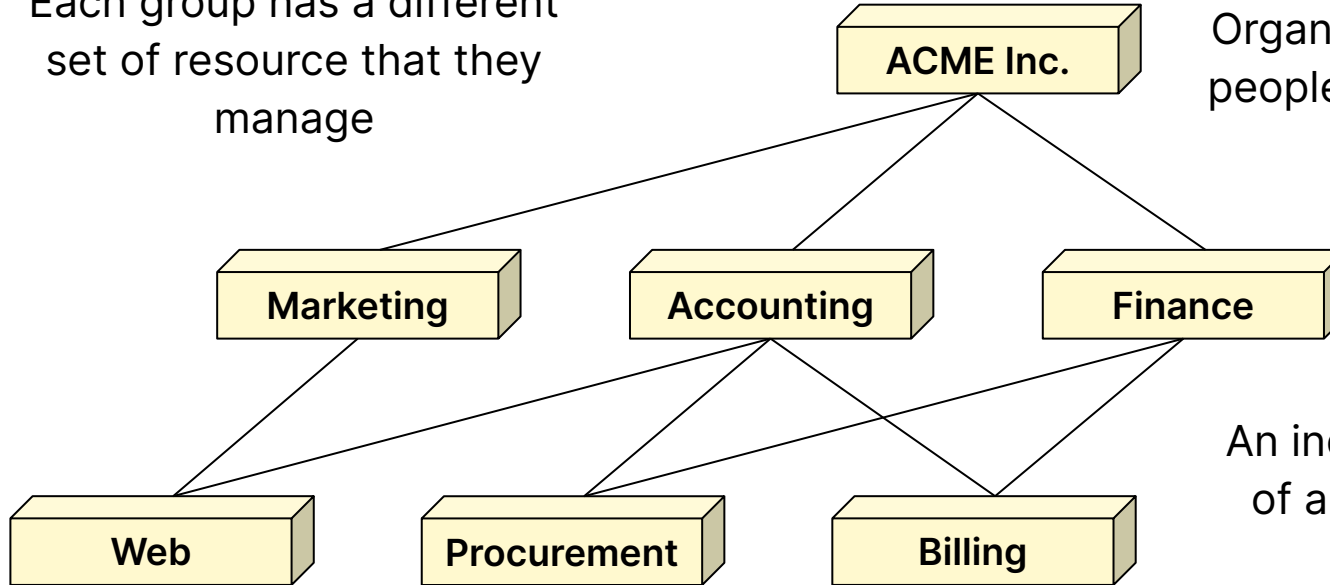| Step 3: Create an Entity |
| :---: |

⬇

Step 4: Create the Entity Aliases
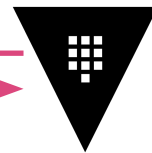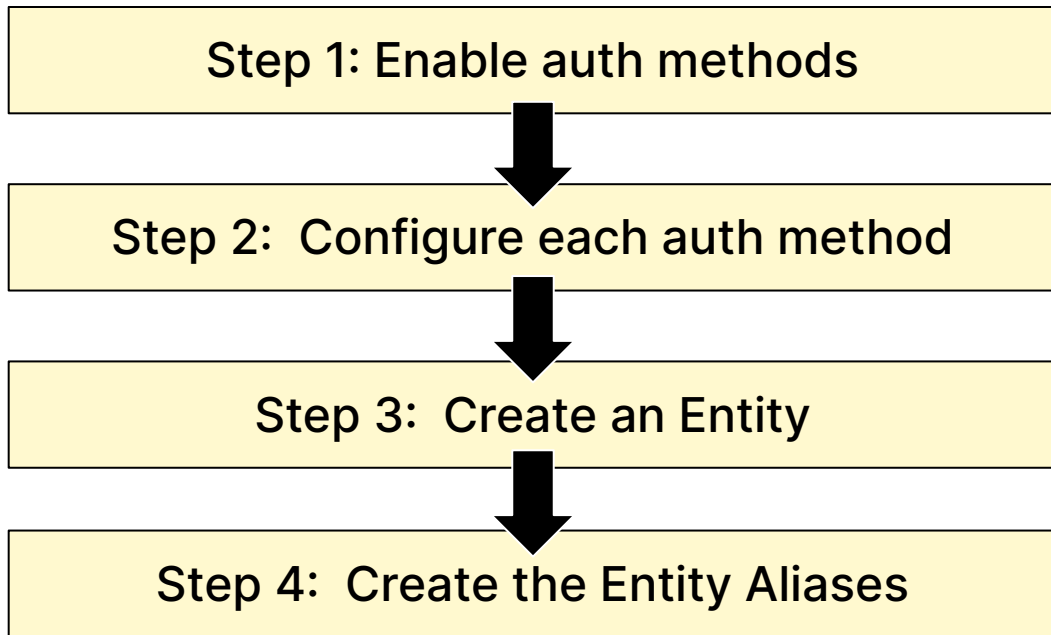
Organizational Structure

Each group has a different set of resource that they manage

ACME Inc.
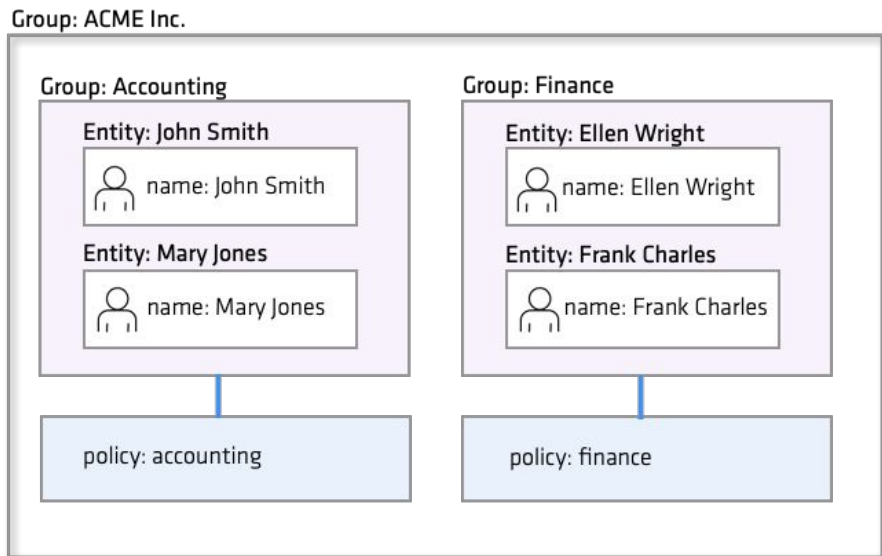
Organization is composed of people who belong in groups

Marketing

Accounting

Finance

An individual may be a part of any number of groups

Web

Procurement

Billing

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

Group: Security
Policy: SRE

Group Members:

Entity: John Smith
Policy: devops

Entity Aliases:

Username: jsmith
Policy: base

associated

TOKEN

base   devops   SRE

Token inherits capabilities granted
by the token itself **as well as** policies
attached to the entity and group
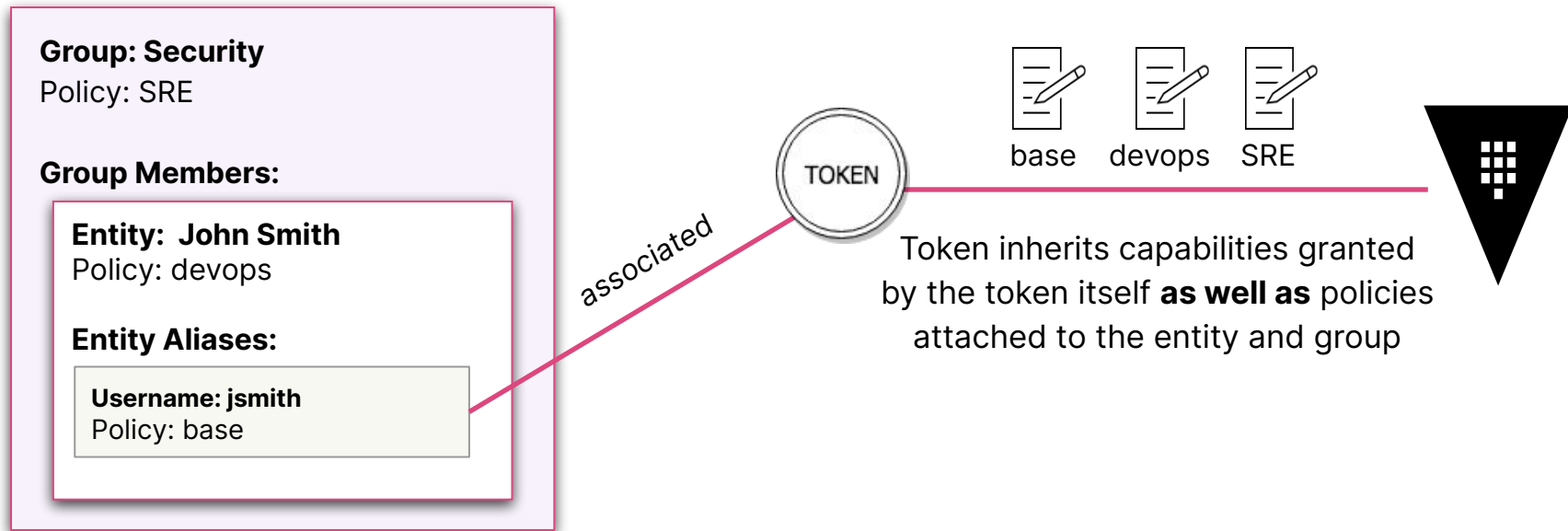
Entity, John Smith inherits the policy
assigned to the Security group

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

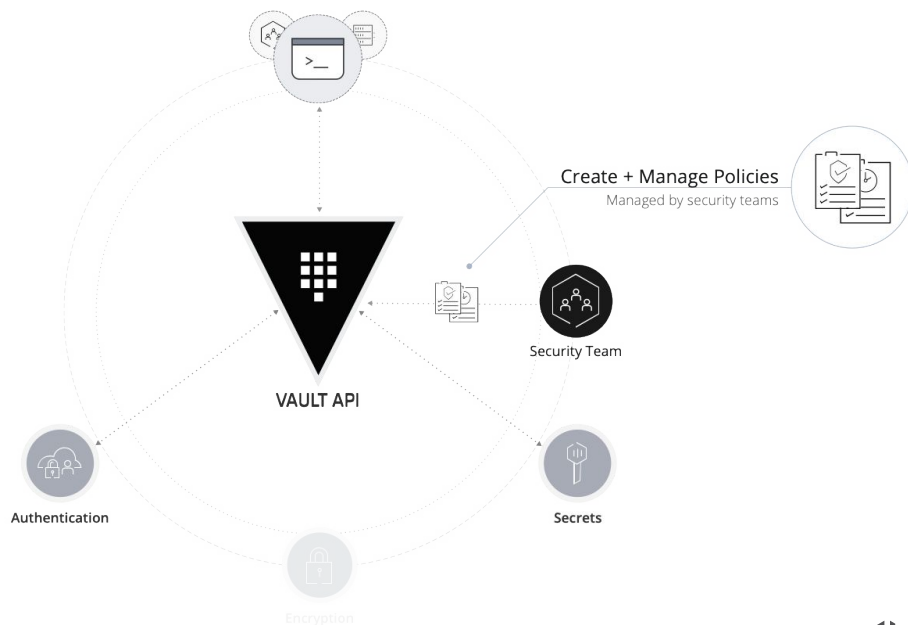 - If a user is a part of an external group (LDAP group), automatically adds the user to the identity group inheriting the policies and metadata

LDAP

bob@example.com
belong to **qa_eng** group

Group

group: testers

id: 61764951-db46-c153-d58c-208ab3e3bcc8

Aliases

LDAP group name: "qa_eng"
mount_accessor: auth_ldap_c48295fc

policy: qa_engineers

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

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

Create + Manage Policies
Managed by security teams

Security Team

VAULT API

Authentication

Secrets

Encryption

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)

Trusted platform

kubernetes

Verified ✓

Verify the identity of App1

⚙ App1

HashiCorp
Vault

vault login

Return a **client token** **for App1**

Policies

Create & manage policies
Enable auth methods
Enable secrets engines
etc.

admin
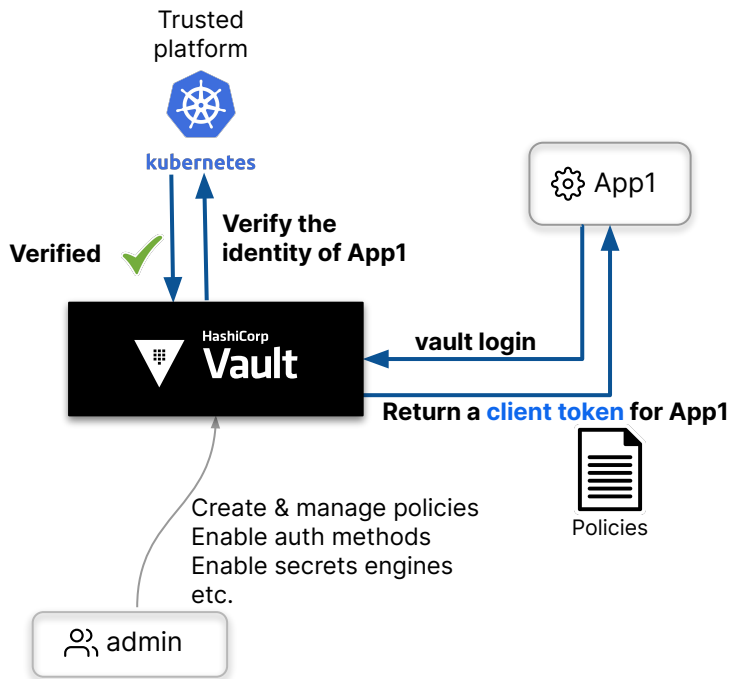
Language of policies

- 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   HTTP Verbs
update   POST/PUT
delete   GET
list   POST/PUT
sudo   DELETE
deny   LIST

Root protected paths

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

Refer to the Tutorial

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 `no_parent` option)
auth/token	POST	Create a periodic or an orphan token (`period` or `no_parent`) 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
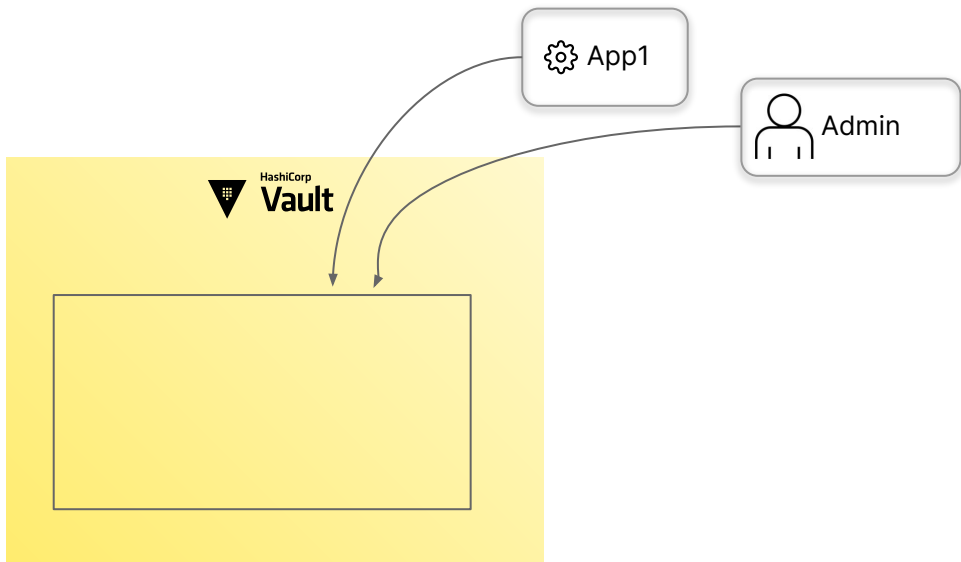
Discover the policy:

- gather secret requirements
- perform operations with Vault
- discovering the paths and the capabilities required
- define a policy
- 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

© HASHICORP

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
```

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

Example Scenario

Using transit secrets engine for data encryption/decryption
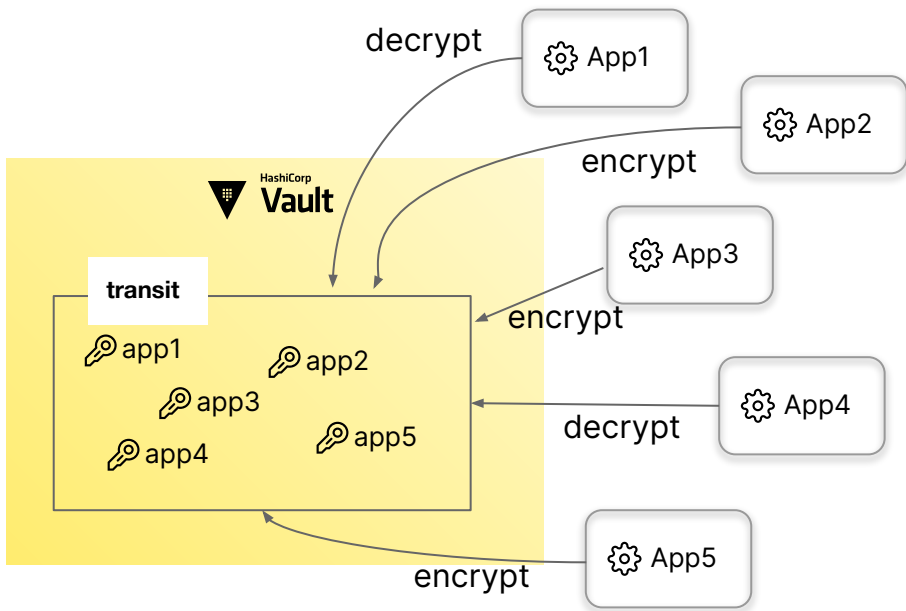
- Each application has its own encryption key

- 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?

Solution 2

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

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

Is this easy to scale and maintain?

- Trouble points:

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

- 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
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

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

```
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
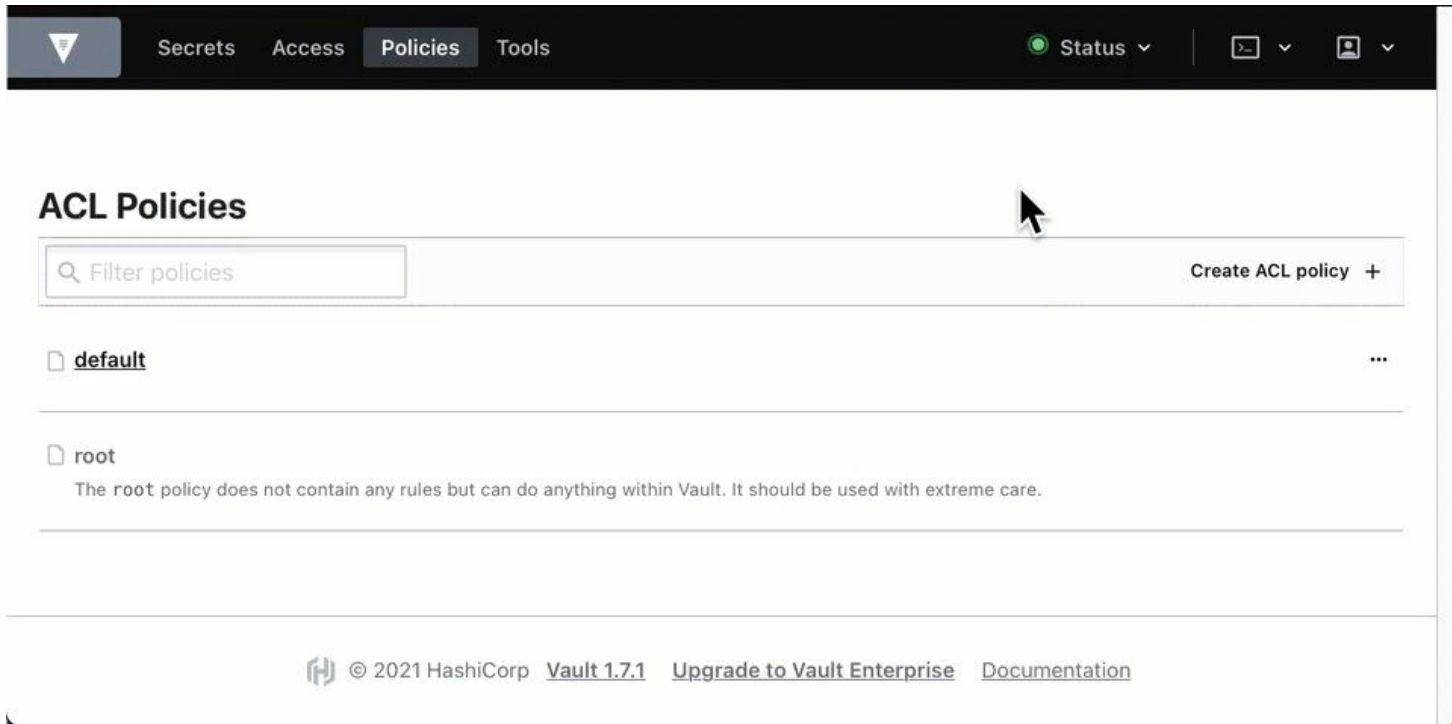
hcp-root policies

- **hcp-root** policy is the policy used for the admin token generated in the HCP control panel and used to access admin namespace.

- Admin tokens with the **hcp-root** policy are not the same as root tokens with the **root** policy

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"
```
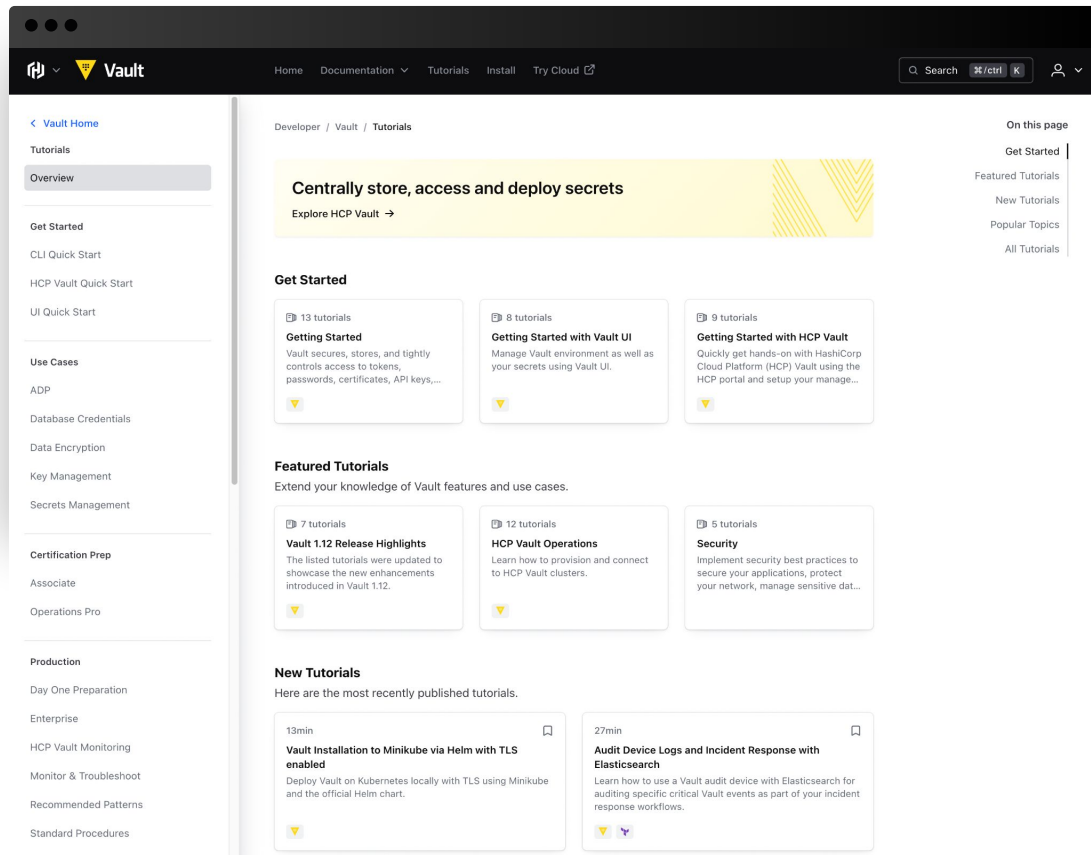
Next Steps

Tutorials

Step-by-step guides to accelerate deployment of Vault

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

Resources

- [Vault Namespace and Mount Structuring Guide](#)

- [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

Consuming Secrets from HCP Vault

The webinar covers how Vault works with trusted platforms to manage Identity and best practices and patterns for leveraging Vault for secrets management

Program Closing

Asynchronous content sent to your inbox that includes some useful resources to validate production readiness and ensure operational best practices for your HCP Vault clusters

Action Items

- Share to 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 HCP Vault instance(s)

- Configure an authentication method and policy for your Vault Administrators

Q&A

Thank you

customer.success@hashicorp.com

www.hashicorp.com/customer-success