Your Name

Your Role, Your Company

Interesting Fact
1. Glossary and Architecture
2. Static and Generic Secrets
3. Policies and Policy Workflow
4. Dynamic Secrets
5. Authentication, Auditing, and Lease Model
6. Operationalizing Vault
7. HTTP API
8. Direct Application Integration
Vault Manages Secure Information
Pre-Vault World

Secret sprawl

Decentralized keys

Limited visibility

Poorly-defined "break-glass" procedures
Post-Vault World

Single source for secrets

Programatic access (automation)

Operation access (manual)

Practical security

Modern data-center friendly (no hardware reqs.)
Glossary
Storage backend

The storage backend is responsible for durable storage of encrypted data. There is only one storage backend per Vault cluster.

Data is encrypted in transit and at rest with 256-bit AES.

Examples: *in-mem*, *file*, *consul*, and *postgresql*
**Barrier**

The barrier is a cryptographic seal around the Vault. All data that flows between Vault and the storage backend passes through the barrier.
Secret backend

A secret backend is responsible for managing secrets. Some secret backends behave like encrypted key-value stores, while others dynamically generate secrets when queried. There can be multiple secret backends in a Vault cluster.

Examples: *pki*, *generic*, *transit*, *postgresql*
Glossary

**Secret backend**

Secret backends can perform almost any function, not just return static data or hand out credentials.

**PKI** – Acts as a full CA, leveraging Vault’s auth

**Transit** – Allows round-tripping data through Vault for "encryption as a service", without ever divulging the key
Audit backend

An audit backend is responsible for managing audit logs. There can be multiple audit backends in a Vault cluster. Example audit logs include *file* and *syslog*. 
**Auth backend**

An auth backend is a credential-based backend that can be used as a way to authenticate humans or machines against Vault.

Machine-oriented: `approle, tls, tokens`
Operator-oriented: `github, ldap, userpass`
Vault token

A vault token is a conceptually similar to a session cookie on a website. Once a user authenticates via an auth backend, Vault returns a token which is to be used for future requests.

Example: dc57a797-fc99-05d1-6878-f731206b1717
Glossary

**Secret**

A secret is anything stored or returned by Vault that contains confidential material.

A secret is anything that, if acquired by an unauthorized party, would cause political, financial, or appearance harm to an organization.
Glossary

Server

The Vault server provides an HTTP API which clients interact with and manages the interaction between all the backends, ACL enforcement, and secret lease revocation.
Architecture
Shamir’s secret sharing

Key Shares → Master Key → Encryption Key
Summary

Solves the "secret sprawl" problem

Protects against external threats (cryptosystem)

Protects against internal threats (ACLs and secret sharing)
Using Generic Secrets
**Exercise: Connect to Workstation**

SSH into your workstation using the provided credentials.

```
ssh <username>@<your.ip.address>
pASSWORD: <password>
```

Change directory into `/workstation/vault`.

There is already a Vault server configured and running locally. Run the `vault status` command to check its status.
$ vault status
Sealed: false
Key Shares: 1
Key Threshold: 1
Unseal Progress: 0
Unseal Nonce: 
Version: 0.7.0
Cluster Name: vault-cluster-9ba50d8b
Cluster ID: 539e6087-78cc-a1ee-f215-2d61db200b80

High-Availability Enabled: false
This Vault server is running in "dev" mode, which is most useful for local development, testing, and exploration.

Everything is stored in-memory.

Vault is automatically unsealed.

Can optionally set the initial root token (which we did).
Generic Secret Backend

The generic secret backend is mounted by default and cannot be disabled.

Behaves like encrypted redis or memcached.

Lives at the secret/ endpoint.
Exercise: Read Generic Secret

Attempt to read the secret at secret/training.

HINT: You can use Vault's help documentation

ANOTHER HINT: You'll get an error
$ vault read secret/training
Error reading secret/training: Error making API request.

URL: GET http://127.0.0.1:8200/v1/secret/training
Code: 400. Errors:

* missing client token
Most interactions with Vault require a token.

Tokens are generated via authentication.

Authentication is covered in more detail in a later section.

Information is persisted by the local client (you do not need to re-authenticate before each command).
Authenticating as the root user is bad practice.

For the purpose of training, we will start slightly insecure and move to a more secure workflow.

The root token is usually used to setup policy and initial set of users, but then is discarded.

Authenticate as root to continue.
$ vault auth root
Exercise: Read Generic Secret (again)

Now that we have authenticated, attempt to read the secret at secret/training again.
Terminal

$ vault read secret/training
No value found at secret/training
Exercise: Write Generic Secret

Write a value into `secret/training`.

HINT: Data is expressed as key=value pairs on the CLI
Terminal

$ vault write secret/training city=nyc food="chicken fingers"
Success! Data written to: secret/training
Exercise: Retrieve Secret

Read the value of the secret you just stored in secret/training.
$ vault read secret/training

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_interval</td>
<td>768h0m0s</td>
</tr>
<tr>
<td>city</td>
<td>nyc</td>
</tr>
<tr>
<td>food</td>
<td>chicken fingers</td>
</tr>
</tbody>
</table>
$ vault read -field=city secret/training
nyc%
$ vault write secret/training food=pizza
Success! Data written to: secret/training
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>refresh_interval</td>
<td>720h0m0s</td>
</tr>
<tr>
<td>food</td>
<td>pizza</td>
</tr>
</tbody>
</table>
Terminal

$ vault write secret/foo a=b

# You can also read values from a file using the "@" symbol.
$ vault write secret/foo a=@file.txt
Exercise: List Secrets

List all the secret keys stored in the generic secret backend.

HINT: Just the keys, not the values.
$ vault list secret
Keys
----
bar
foo
training
Exercise: Delete Secret

Delete one of the secrets you just created.

Do **NOT** delete the training key.
$ vault delete secret/foo
Success! Deleted 'secret/foo' if it existed.
Getting Help
Getting Help

There are two primary ways to get help in Vault:

- CLI help (vault -h)
- API help (vault path-help)
$ vault help # CLI help (aka "-h")

$ vault path-help # API help
$ vault read -h
Usage: vault read [options] path

Read data from Vault.

Reads data at the given path from Vault. This can be used to read secrets and configuration as well as generate dynamic values from materialized backends. Please reference the documentation for the backends in use to determine key structure.
$ vault path-help secret/

## DESCRIPTION

The generic backend reads and writes arbitrary secrets to the backend.

...

## PATHS

The following paths are supported by this backend. To view help for any of the paths below, use the help command with any route matching the path pattern. Note that depending on the policy of your auth token, you may or may not be able to access certain paths.

^.*$

Pass-through secret storage to the storage backend, allowing you to read/write arbitrary data into secret storage.
List help information for the HTTP API cubbyhole backend
$ vault path-help cubbyhole/

## DESCRIPTION

The cubbyhole backend reads and writes arbitrary secrets to the backend. The secrets are encrypted/decrypted by Vault: they are never stored unencrypted in the backend and the backend never has an opportunity to see the unencrypted value.

This backend differs from the 'generic' backend in that it is namespaced per-token. Tokens can only read and write their own values, with no sharing possible (per-token cubbyholes). This can be useful for implementing certain authentication workflows, as well as "scratch" areas for individual clients. When the token is revoked, the entire set of stored values for that token is also removed.

## PATHS

The following paths are supported by this backend. To view help for
About: Cubbyhole
Setting Policy
Access Control Policies (ACLs)

"root" policy is created by default – superuser with all permissions.

"default" policy is created by default - common permissions.

Policies are written in HashiCorp Configuration Language (HCL), which is a human-friendly config format.

Deny by default (no policy = no authorization).
$ vault policies
Terminal

$ vault policies
default
root
path "secret/training_*" {
    capabilities = ["create", "read"]
}
$ vault policy-write base ./base.hcl

$ vault write sys/policy/base rules=@base.hcl
Terminal

$ vault policy-write base ./base.hcl
Policy 'base' written.

$ vault write sys/policy/base rules=@base.hcl
Success! Data written to: sys/policy/base
$ vault policies
base
default
root

$ vault read sys/policy
Key     Value
---     ----- 
keys    [base default root]
policies [base default root]
Terminal

$ vault policies base
path "secret/training_*" {
    capabilities = ["create", "read"]
}

$ vault read sys/policy/base
Key     Value
---     ----- 
name    base
rules   path "secret/training_*" {
    capabilities = ["create", "read"]
}
$ vault token-create -policy=base

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>ce3bd491-2533-7a32-9526-f0ea83c6a68a</td>
</tr>
<tr>
<td>token_accessor</td>
<td>bf772963-95b1-1776-1b4c-9f214dab071a</td>
</tr>
<tr>
<td>token_duration</td>
<td>768h0m0s</td>
</tr>
<tr>
<td>token_renewable</td>
<td>true</td>
</tr>
<tr>
<td>token_policies</td>
<td>[base default]</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>token</td>
<td>60a3c690-7120-9edd-e4ed-75eb305c99b6</td>
</tr>
<tr>
<td>token_accessor</td>
<td>626c4876-1c6a-01f2-5d2e-d8cc9ec27ae5</td>
</tr>
<tr>
<td>token_duration</td>
<td>720h0m0s</td>
</tr>
<tr>
<td>token_renewable</td>
<td>true</td>
</tr>
<tr>
<td>token_policies</td>
<td>[base default]</td>
</tr>
</tbody>
</table>
```bash
$ vault token-create -policy=base

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>60a3c690-7120-9edd-e4ed-75eb305c99b6</td>
</tr>
<tr>
<td>token_accessor</td>
<td>626c4876-1c6a-01f2-5d2e-d8cc9ec27ae5</td>
</tr>
<tr>
<td>token_duration</td>
<td>720h0m0s</td>
</tr>
<tr>
<td>token_renewable</td>
<td>true</td>
</tr>
<tr>
<td>token_policies</td>
<td>[base default]</td>
</tr>
</tbody>
</table>
```
$ vault auth 062d33e2-52e8-e60b-2f43-9f09277b0716
Successfully authenticated! You are now logged in.
token: 062d33e2-52e8-e60b-2f43-9f09277b0716
token_duration: 2591973
token_policies: [base, default]
Terminal

$ vault policies

$ vault read sys/policy
$ vault policies
Error: Error making API request.

URL: GET http://127.0.0.1:8200/v1/sys/policy
Code: 403. Errors:

* permission denied

$ vault read sys/policy
Error reading sys/policy: Error making API request.

URL: GET http://127.0.0.1:8200/v1/sys/policy
Code: 403. Errors:

* permission denied
$ vault write secret/foo bar=1
$ vault write secret/foo bar=1
Error writing data to secret/foo: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/secret/foo
Code: 403. Errors:

* permission denied
$ vault write secret/training_foo bar=1
Terminal

$ vault write secret/training_foo bar=1
Success! Data written to: secret/training_foo
$ vault write secret/training_foo bar=2
$ vault write secret/training_foo bar=2
Error writing data to secret/training_foo: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/secret/training_foo
Code: 403. Errors:

* permission denied
$ vault write secret/training_ bar=1
$ vault write secret/training_ bar=1
Success! Data written to: secret/training_
Write a policy named "exercise" that permits listing and deleting anything in the generic secret backend, but forbids creating, reading, or updating a secret. **Do not upload the policy.**

```
$ vault list secret/      # ok
$ vault delete secret/foo # ok
$ vault read secret/foo   # 403
$ vault write secret/foo  # 403
```
path "secret/*" {
    capabilities = ["delete", "list"]
}


Exercise: Re-auth as root

Re-authenticate as root

(Our current user does not have enough permission)
$ vault auth root
Successfully authenticated! You are now logged in.
token: root
token_duration: 0
token_policies: [root]
Dynamic Secrets
Secret Backends

Most secret backends must be mounted before use.

Many secret backends require additional configuration before use.
$ vault mount postgresql
Successfully mounted 'postgresql' at 'postgresql'!
$ vault path-help postgresql/
$ vault path-help postgresql/
The following paths are supported by this backend. To view help for any of the paths below, use the help command with any route matching the path pattern. Note that depending on the policy of your auth token, you may or may not be able to access certain paths.

^config/connection$
   Configure the connection string to talk to PostgreSQL.

^config/lease$
   Configure the default lease information for generated credentials.

^creds/(?P<name>\w[\w-.]+\w)$
   Request database credentials for a certain role.

^roles/(?P<name>\w[\w-]+\w)$
   Manage the roles that can be created with this backend.
$ vault path-help postgresql/config/connection

## PARAMETERS

  connection_url (string)  
  DB connection string

# ...

The URL looks like:
"postgresql://user:pass@host:port/dbname"
$ vault write postgresql/config/connection \
    connection_url="postgresql://postgres@localhost/myapp"

The following warnings were returned from the Vault server:
* Read access to this endpoint should be controlled via ACLs as it will return
the connection string or URL as it is, including passwords, if any.
Terminal

$ vault write postgresql/config/lease \
  lease=1h \
  lease_max=24h
Success! Data written to: postgresql/config/lease
Terminal

$ vault write postgresql/roles/readonly \
  sql="..."
Success! Data written to: postgresql/roles/readonly
$ cat readonly.sql
CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{password}' VALID UNTIL '{expiration}';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO "{{name}}";
$ cat readonly.sql
CREATE ROLE "{name}" WITH LOGIN PASSWORD '{password}' VALID UNTIL '{expiration}';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO "{name}";

$ vault write postgresql/roles/readonly \
    sql=@readonly.sql
Success! Data written to: postgresql/roles/readonly
$ vault read postgresql/creds/readonly
```
$ vault read postgresql/creds/readonly

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lease_id</td>
<td>postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-...</td>
</tr>
<tr>
<td>lease_duration</td>
<td>1h0m0s</td>
</tr>
<tr>
<td>lease_renewable</td>
<td>true</td>
</tr>
<tr>
<td>password</td>
<td>c8572361-5e45-24cc-378a-f9cbe5667bcd</td>
</tr>
<tr>
<td>username</td>
<td>token-38bf7cf0-b494-a218-152d-f450151f8d01</td>
</tr>
</tbody>
</table>
```
$ vault read postgresql/creds/readonly
Key            Value
---            -----
lease_id       postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-...
lease_duration 1h0m0s
lease_renewable true
password       c8572361-5e45-24cc-378a-f9c8a5667bcd
username       token-38bf7cf0-b494-a218-152d-f450151f8d01
$ psql -U postgres

postgres=# \du

<table>
<thead>
<tr>
<th>Role name</th>
<th>Attributes</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgres</td>
<td>Superuser, Create ...</td>
<td>{}</td>
</tr>
<tr>
<td>token-38bf7cf0-b494-a218-152d-f450151f8d01</td>
<td>Password valid until ...</td>
<td>{}</td>
</tr>
</tbody>
</table>

postgres=# \q
$ vault renew postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-c5a5779cf15a
Terminal

$ vault renew postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-c5a5779cf15a

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lease_id</td>
<td>postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-c5a5779cf15a</td>
</tr>
<tr>
<td>lease_duration</td>
<td>1h0m0s</td>
</tr>
<tr>
<td>lease_renewable</td>
<td>true</td>
</tr>
</tbody>
</table>

Terminal

$ vault revoke postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-c5a5779cf15a
Terminal

$ vault revoke postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-c5a5779cf15a
Success! Revoked the secret with ID 'postgresql/creds/readonly/c0a34ca8-9fba-05af-2874-c5a5779cf15a', if it existed.
Terminal

$ psql -U postgres

postgres=# \du

<table>
<thead>
<tr>
<th>Role name</th>
<th>Attributes</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgres</td>
<td>Superuser, Create ...</td>
<td>{}</td>
</tr>
</tbody>
</table>

postgres=# \q
Terminal

$ vault read postgresql/creds/readonly
password       149237b3-09dd-1906-59f4-df696d02e9b1
username       root-3fe9df95-c5a6-ef05-2a18-000208e29205

$ vault read postgresql/creds/readonly
password       f9de9932-af02-cb22-bfed-3b8fb3b3f03a
username       root-9a554127-2c0e-db2e-1936-8187b7aa5782

$ vault read postgresql/creds/readonly
password       aca3d19b-e653-27e7-5956-29ba20db2739
username       root-a95b255c-cf4b-abe4-a56b-eb3af16fcd7a

# ...
$ vault revoke -prefix postgresql/creds
Terminal

$ vault revoke -prefix postgresql/creds
Success! Revoked the secret with ID 'postgresql/creds', if it existed.
Terminal

$ psql -U postgres

postgres=# \du

<table>
<thead>
<tr>
<th>Role name</th>
<th>Attributes</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgres</td>
<td>Superuser, Create ...</td>
<td>{}</td>
</tr>
</tbody>
</table>

postgres=# \q
Working with Leases
Leasing, renewal, and revocation

Start

Have Secret?

Renew lease

Read Secret

Finish

Error

Yes

No

Success

Failure

Success

Failure
Lease Hierarchy and Revocations

b519c6aa... (3h)
6a2cf3e7... (4h)
1d3fd4b2... (1h)
794b6f2f... (2h)
List the order in which the leases would expire.

b519c6aa... (3h)
6a2cf3e7... (4h)
1d3fd4b2... (1h)
794b6f2f... (2h)
Exercise: Predicting Behavior

List the order in which the leases would expire.

1. 794b6f2... (2h)
2. 1d3fd4b2... (1h)
3. 6a2cf3e7... (4h)
4. b519c6aa... (3h)
If a token or secret with a lease is not renewed before the lease expires, it and all children will be revoked by the Vault server.

A child is a token, secret, or authentication created by a parent. A parent is almost always a token.
Exercise: Understanding Revocations

1. Create a new token with a 30s lease (hint: use help output)
2. Auth as this token
3. After 30s, try to read a value using token
$ vault token-create -ttl=30s

  token          5cf4e7b5-7f88-0769-e406-686e2a90c471
  token_duration 30

$ vault auth 5cf4e7b5-7f88-0769-e406-686e2a90c471
Successfully authenticated!

$ vault read secret/training
...

$ vault read secret/training
Error!
Exercise: Re-auth as root

Re-authenticate as root
$ vault auth root
Successfully authenticated! You are now logged in.
token: root
token_duration: 0
token_policies: [root]
Lease Best Practices

Renew leases at half the lease duration value – e.g. 10m lease should renew every 5m.

Attempt a re-read if renewal fails (generates new credentials).
Notable Exception: Orphan Token

Root/sudo users have the ability to generate "orphan" tokens. Orphan tokens are not children of their parent, therefore do not expire when their parent does. Orphan tokens still expire when their own Max TTL is reached.
Root/sudo users have the ability to generate "periodic" tokens.

Periodic tokens have a TTL, but no max TTL.

Periodic tokens may live for an infinite amount of time, so long as they are renewed within their TTL.

This is useful for long-running services that cannot handle regenerating a token.
Notable Exception: Use Limits

In addition to TTL and Max TTL, tokens may be limited to a number of uses.

Use limit tokens expire at the end of their last use, regardless of their remaining TTLS.

Use limit tokens expire at the end of their TTLs, regardless of remaining uses.
Authentication
Understanding Authentication

Authentication is a process in Vault by which user or machine-supplied information is verified to create a token with pre-configured policy.

Future requests are made using the token.
Authentication Setup

1. Activate the authentication using the auth-enable command
2. Configure the authentication (varies)
3. Map the authentication to a set of policies
$ vault auth -methods

$ vault read sys/auth
$ vault auth -methods

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token/</td>
<td>token</td>
<td>token based credentials</td>
</tr>
</tbody>
</table>

$ vault read sys/auth

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>token/</td>
<td>map[description:token based credentials local:false type:token #...]</td>
</tr>
</tbody>
</table>
Terminal

$ vault auth-enable userpass

$ vault write sys/auth/userpass type=userpass
Terminal

$ vault auth-enable userpass
Successfully enabled 'userpass' at 'userpass'!

$ vault write sys/auth/userpass type=userpass
Success! Data written to: sys/auth/userpass
$ vault auth -methods

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
<th>Default TTL</th>
<th>Max TTL</th>
<th>Replication Behavior</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token/</td>
<td>token</td>
<td>system</td>
<td>system</td>
<td>replicated</td>
<td>token based</td>
</tr>
<tr>
<td>credentials</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>userpass/</td>
<td>userpass</td>
<td>system</td>
<td>system</td>
<td>replicated</td>
<td></td>
</tr>
</tbody>
</table>

$ vault read sys/auth

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>token/</td>
<td>map[local:false type:token config:map[default_lease_ttl:0 # ...]</td>
</tr>
<tr>
<td>userpass/</td>
<td>map[config:map[default_lease_ttl:0 max_lease_ttl:0] # ...]</td>
</tr>
</tbody>
</table>
Mount the userpass backend at the path "training-userpass".
Terminal

$ vault auth-enable -path=training-userpass userpass
Successfully enabled 'userpass' at 'training-userpass'!

$ vault write sys/auth/training-userpass type=userpass
Success! Data written to: sys/auth/training-userpass
Terminal

$ vault write auth/userpass/users/sethvargo password=training policies=base
Terminal

$ vault write auth/userpass/users/sethvargo password=training policies=base
Success! Data written to: auth/userpass/users/sethvargo
Terminal

```
$ vault read auth/userpass/users/sethvargo
Key            Value
---            -----  
max_ttl        0
policies       base,default
ttl            0
```
Exercise: Create Auth with Custom Policy

Create a new policy named "contractor" that grants only the ability to generate readonly credentials from the postgresql backend.

Create a new userpass authentication that attaches the above policy. Use the username "sandy" and the password "training".

Authenticate as this user and generate a postgresql credential (HINT: vault auth -h)
contractor.hcl

path "postgresql/creds/readonly" {
  capabilities = ["read"]
}

$ vault policy-write contractor ./contractor.hcl
Policy 'contractor' written.

$ vault write sys/policy/contractor rules=@contractor.hcl
Success! Data written to: sys/policy/contractor
$ vault write auth/userpass/users/sandy password=training policies=contractor
Success! Data written to: auth/userpass/users/sandy
$ vault auth -method=userpass username=sandy password=training
Successfully authenticated! You are now logged in.
The token below is already saved in the session. You do not need to "vault auth" again with the token.
token: ca7999c5-841c-ae7d-6e37-d279d35ecaa2
token_duration: 2591999

token_policies: [contractor default]

$ vault write auth/userpass/login/sandy password=training

Key                Value
---                -----  
token              fb0522fa-b990-a8c9-1087-2c17ea2b2682
token_accessor     1a8a7e6f-6af8-c7a1-2d03-3d521f5bb3b3
token_duration     768h0m0s
token_renewable    true
token_policies     [contractor default]
token_meta_username "sandy"
$ vault read postgresql/creds/readonly
$ vault read postgresql/creds/readonly

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lease_id</td>
<td>postgresql/creds/readonly/dfe53c55-0b53-3ce6-91a1-16e6496bd7ef</td>
</tr>
<tr>
<td>lease_duration</td>
<td>1h0m0s</td>
</tr>
<tr>
<td>lease_renewable</td>
<td>true</td>
</tr>
<tr>
<td>password</td>
<td>aa475a37-94ac-58b7-b8aa-4fcbd4883581</td>
</tr>
<tr>
<td>username</td>
<td>userpass-sandy-855201cc-12d3-7a39-ed9a-a7353470da60</td>
</tr>
</tbody>
</table>
$ vault write secret/foo bar=1
Error writing data to secret/foo: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/secret/foo
Code: 403. Errors:

* permission denied
Exercise: Auth as yourself

Authenticate as root token.
$ vault auth root
Successfully authenticated! You are now logged in.
token: root
token_duration: 0
token_policies: [root]
Auditing
Audit backends keep a detailed log of all requests and responses to Vault.

Sensitive information is obfuscated by default (HMAC).

Prioritizes safety over availability.
Exercise: Enable Audit Backend

Enable the "file" audit backend to write to the path
/workstation/vault/audit.log

HINT: there are two "paths" - the URL path and the path on disk
$ vault audit-enable file file_path=/workstation/vault/audit.log
Successfully enabled audit backend 'file' with path 'file'!
$ sudo cat audit.log | jq .
{
  "response": {
    "redirect": "",
    "data": null,
    "secret": null
  },
  "request": {
    "wrap_ttl": 0,
    "remote_address": "127.0.0.1",
    "data": {
      "type": "hmac-sha256:c5d7b87...",
      "options": {
        "path": "hmac-sha256:1230e0b...",
      },
      "description": "hmac-sha256:355af1f..."
    },
    # ...
  }
}
Auditing Additional Fields

In addition to the standard fields, Vault can optionally audit user-defined headers

Useful for logging things like **X-Forwarded-For**
Exercise: Audit X-Forwarded-For

Configure Vault to audit the `X-Forwarded-For` header.

HINT: API docs for `sys/config`
$ vault write sys/config/auditing/request-headers/X-Forwarded-For hmac=false
Success! Data written to: sys/config/auditing/request-headers/X-Forwarded-For

$ vault write -f sys/config/auditing/request-headers/X-Forwarded-For
Success! Data written to: sys/config/auditing/request-headers/X-Forwarded-For

$ sudo cat audit.log
{
  "request": {
    "headers": {
      "x-forwarded-for": [
        "hello-world"
      ]
    }
  },
  # ...
}
Operating Vault
Non-Dev Configuration

Vault is configured with one or more configuration files

Configuration defines 1 storage backend and 1+ listeners

Vault is run via a supervisor (upstart, systemd) or a scheduler (nomad, k8s, etc)

Initialization is required before use
# Use the file backend - this will write encrypted data to disk.
storage "file" {
    path = "/workstation/vault/data"
}

# Listen on a different port (8201), which will allow us to run multiple
# Vault's simultaneously.
listener "tcp" {
    address = "127.0.0.1:8201"
    tls_disable = 1
    }
$ sudo service vault-2 start
vault-2 start/running, process 1490
$ vault status -address=http://127.0.0.1:8201
Error checking seal status: Error making API request.

URL: GET http://127.0.0.1:8201/v1/sys/seal-status
Code: 400. Errors:

* server is not yet initialized
$ vault status -address=http://127.0.0.1:8201
Error checking seal status: Error making API request.

URL: GET http://127.0.0.1:8201/v1/sys/seal-status
Code: 400. Errors:

* server is not yet initialized
$ vault init -address=http://127.0.0.1:8201
$ vault init -address=http://127.0.0.1:8201

Unseal Key 1: U9qBRp7cKzvP6CRq5ZLmMAM0JtOCdve0H66qq0dkNHYB
Unseal Key 2: l0Y59WCxSYXWDQXHOweUYkcNciX40qSwIBPd138+N+gC
Unseal Key 3: EYKspLFe36NGdegtFTDA+H39RD2lDtoH4qjHFJ5epoD
Unseal Key 4: c3g+039vadztTCCS8M3m3pjKjenfTGIiw0s//WdjmOcE
Unseal Key 5: thyrgq6MWye2WPLr56xvz4ZrtzRCv3Q/NJBNkok1ZUF
Initial Root Token: 534b5f0c-a498-5e33-9428-099374755214

Vault initialized with 5 keys and a key threshold of 3. Please securely distribute the above keys. When the Vault is re-sealed, restarted, or stopped, you must provide at least 3 of these keys to unseal it again.

Vault does not store the master key. Without at least 3 keys, your Vault will remain permanently sealed.
Terminal

$ vault init -address=http://127.0.0.1:8201

Unseal Key 1: U9qBRp7cKzvP6CRq5ZLmAM0JtOCdveOH66qq0dkNHYB
Unseal Key 2: 1OY59WCxSYXMDQXHOweUYkcNciX40qSwIBPd138+N+gC
Unseal Key 3: EYKspLFSe36NGdegjFTDA+H39RD21DtoH4qjHFJ5epoD
Unseal Key 4: c3g+039vadztTCCS8M3m3pjKenfTGIIw0s//WdjmOcE
Unseal Key 5: thyrgq6MWye2WPL1R56xvz4ZrtzRCv3Q/NJBNkok1ZUF
Initial Root Token: 534b5f0c-a498-5e33-9428-099374755214

Vault initialized with 5 keys and a key threshold of 3. Please securely distribute the above keys. When the Vault is re-sealed, restarted, or stopped, you must provide at least 3 of these keys to unseal it again.

Vault does not store the master key. Without at least 3 keys, your Vault will remain permanently sealed.
Terminal

$ vault init -address=http://127.0.0.1:8201
Unseal Key 1: U9qBRp7cKzvP6CRq5ZLmMAM0JtOcdve0H66qq0dkNHYB
Unseal Key 2: 1OY59WCxSYXWDQXHOweUYkcNiX40gSwIPd138+N+gC
Unseal Key 3: EYKspLFSe36NGdegjFTDA+H39RD21DtoH4qjHFJ5epoD
Unseal Key 4: c3g+039vadztTCCS8M3m3pjjKenfTGIiw0s//WdjmOcE
Unseal Key 5: thyrgg6MWye2WPl1R56xvxzRrtzRCv3Q/NJBNkok1ZU6F
Initial Root Token: 534b5f0c-a498-5e33-9428-099374755214

Vault initialized with 5 keys and a key threshold of 3. Please securely distribute the above keys. When the Vault is re-sealed, restarted, or stopped, you must provide at least 3 of these keys to unseal it again.

Vault does not store the master key. Without at least 3 keys, your Vault will remain permanently sealed.
$ vault unseal -address=http://127.0.0.1:8201

Key (will be hidden):
Sealed: true
Key Shares: 5
Key Threshold: 3
Unseal Progress: 1
$ vault unseal -address=http://127.0.0.1:8201
Key (will be hidden):
Sealed: true
Key Shares: 5
Key Threshold: 3
Unseal Progress: 2
$ vault unseal -address=http://127.0.0.1:8201
Key (will be hidden):
Sealed: false
Key Shares: 5
Key Threshold: 3
Unseal Progress: 0
$ vault init \
  -key-shares=3 \ 
  -key-threshold=2 \ 
  -pgp-keys="keybase:sethvargo,keybase:mitchellh,keybase:armon"

Unseal Key 1: wcFMA0RVkJtoqzRjAmlBoP4m7mgN131nW/G2gasFuX8QqynL...
Unseal Key 2: wcFMA0fJU2fv1nFpARAAUq9vH0I3Cmt+JrglCctXjhIqTDrRKZ7x...
Unseal Key 3: wcFMA+27Y9Ny76cXARAAPR/hEBYFbJPAlu2880PoF+5LqUjL0nFq...
Initial Root Token: 4f0214bd-8489-0d13-bb4d-9f63f7b609be

Vault initialized with 3 keys and a key threshold of 2. Please securely distribute the above keys. When the Vault is re-sealed, restarted, or stopped, you must provide at least 2 of these keys to unseal it again.

Vault does not store the master key. Without at least 2 keys, your Vault will remain permanently sealed.
Deploying Vault HA

1. Deploy one Vault with an HA storage backend configured
2. Run `vault init` to generate unseal keys and token on first Vault
3. Unseal the Vault
4. Repeat the above steps on the second Vault, except `init`
HA Vault

VAULT 1

VAULT 2

VAULT 3

CONSUL

HashiCorp
HA Vault

- Vault 1
- Vault 2
- Vault 3

CONSUL
LEADER ELECTION

VAULT 1

VAULT 2

VAULT 3

CONSUL
HA Vault

GET /secret/foo

REQUEST

VAULT 1

VAULT 2

VAULT 3

CONSUL
HA Vault

GET /secret/foo

REQUEST

VAULT 1

VAULT 2

VAULT 3

CONSUL
HA Vault

GET /secret/foo

REQUEST
HA Vault

- Vault 1
- Vault 2
- Vault 3

Consul
HA Vault

VAULT 1

VAULT 2

VAULT 3

CONSUL
HA Vault

- Vault 1
- Vault 2
- Vault 3

Consul

Security lockdown
GET /secret/foo
REQUEST

VAULT 1

VAULT 2

VAULT 3

CONSUL

HashiCorp
(Re)generating Root
In a production Vault installation, the initial root token should only be used for initial configuration.

After a subset of administrators have `sudo` access, almost all operations can be performed.

But for some system-critical operations, a root token may still be required.
Regenerating the Root Token

A quorum of unseal key holders can generate a new root token.

Enforces the "no one person has complete access to the system".
Steps to Regenerate Root

1. Make sure the Vault is unsealed
2. Generate a one-time-password to share
3. Each key-holder runs `generate-root` with the OTP
4. Decode the root token
Terminal

$ vault unseal ...
(this is already done)
$ vault generate-root -genotp
OTP: PddHKv/HqfP9OGddpiY69Q==
$ vault generate-root -otp="PddHKv/HqfP9OGddpiY69Q=="
Root generation operation nonce: 3edbf635-0581-56ca-5f5c-143402252b7b
Key (will be hidden):
$ vault generate-root -otp="PddHKv/HqfP9OGddpiY69Q=="

Root generation operation nonce: 3edbf635-0581-56ca-5f5c-143402252b7b
Key (will be hidden):
Nonce: 3edbf635-0581-56ca-5f5c-143402252b7b
Started: true
Generate Root Progress: 1
Required Keys: 1
Complete: true

Encoded root token: B6pCxMFVOhQ28BwmgwHnhQ==
$ vault generate-root -otp="PddHKv/HqfP9OGddpiY69Q==" \
-decode="B6pCxMFVOhQ28BWmgwHNhQ=="

Root token: 3a7d05ee-3e92-93e7-cbc8-72fb2527f770
Exercise: Generate New Root Token

Generate a new root token for the initial Vault server (not vault-2).

HINT: Find the unseal key in /var/log/upstart/vault.log
HTTP API
About the HTTP API

All interactions with Vault happen via the HTTP API

Even the CLI uses the HTTP API – there is nothing special

Auth is passed via the X-Vault-Token header unless authing

Multiple client libraries exist (Go, Ruby, Python, Node, etc)
HTTP API Status Codes

- 200/204 - Success (no data)
- 400 - Invalid request
- 403 - Forbidden
- 404 - Invalid path
- 429 - Rate limit exceeded
- 500 - Internal server error
- 503 - Vault is sealed or in maintenance
$ vault read secret/training
# ...

$ curl $VAULT_ADDR/v1/secret/training \
   --request GET \
   --header "X-Vault-Token: d9213f90-f569-adae-663f-eb6668403aed"
{
   "auth": null,
   "warnings": null,
   "data": {
      "name": "seth",
      "food": "chicken fingers"
   },
   "lease_duration": 2592000,
   "renewable": false,
   "lease_id": ""
}
Terminal

```
$ vault list secret/
  # ...

$ curl $VAULT_ADDR/v1/secret \
   --request LIST \
   --header "X-Vault-Token: d9213f90-f569-adae-663f-eb6668403aed"
{
  "auth": null,
  "warnings": null,
  "data": {
    "keys": [
      "foo",
      "training"
    ]
  },
  "lease_duration": 0,
  "renewable": false,
  "lease_id": ""
```
$ vault write secret/foo bar=1
  # ...

$ curl $VAULT_ADDR/v1/secret/foo \
  --request POST \
  --header "X-Vault-Token: d9213f90-f569-adae-663f-eb6668403aed" \
  --data '{"bar":"1"}''
Exercise: Use HTTP API

Retrieve a new set of readonly credentials from the postgres backend using the HTTP API.
$ curl $VAULT_ADDR/v1/postgresql/creds/readonly \
  --request GET \
  --header "X-Vault-Token: root"
{
  "auth": null,
  "warnings": null,
  "wrap_info": null,
  "data": {
    "username": "token-eb0376e4-c6e0-2de4-0692-21fb7f93334d",
    "password": "ec139929-4b0f-51ac-6bf1-25fc8ff7a7a9"
  },
  "lease_duration": 3600,
  "renewable": true,
  "lease_id": "postgresql/creds/readonly/6d0b6607-a472-c2a7-e933-878815a451d8",
  "request_id": "87964c9d-b636-91c2-3d7e-2b4984cca14b"
Exercise: Use HTTP API

Renew the lease you just created, using the HTTP API (HINT: sys).
$ curl $VAULT_ADDR/v1/sys/renew/postgresql/creds/readonly/6d0b6607-a472-c2a7-e933-878815a451d8 \
  --request POST \
  --header "X-Vault-Token: root"
{
  "auth": null,
  "warnings": null,
  "wrap_info": null,
  "data": null,
  "lease_duration": 3600,
  "renewable": true,
  "lease_id": "postgresql/creds/readonly/6d0b6607-a472-c2a7-e933-878815a451d8",
  "request_id": "fd394f9d-991a-1f78-6e07-44e97507b9ef"}

Exercise: Authenticate Using the HTTP API

Authenticate as a contractor with userpass using the HTTP API.

Recall that the credentials are:

Username: sandy
Password: training

HINT: You may need to look at the API documentation
$ curl $VAULT_ADDR/v1/auth/userpass/login/sandy \
   --request POST \
   --data '{"password":"training"}'}
Terminal

{
  "auth": {
    "renewable": true,
    "lease_duration": 2592000,
    "metadata": {
      "username": "sandy"
    },
    "policies": [
      "contractor",
      "default"
    ],
    "accessor": "7abdc853-f43f-57ba-628a-e0f31436f6ab",
    "client_token": "e6187881-c0ff-f772-c4cd-6ccbac161e33"
  },
  "warnings": null,
  "wrap_info": null,
  "data": null,
}
Consul Template
About Consul Template

Despite its name, Consul Template does not require a Consul cluster to operate.

Retrieves secrets from Vault and manages the acquisition and renewal lifecycle.

Requires a token (VAULT_TOKEN) to operate.
$ consul-template \\n    -template "in.ctmpl:out.txt:command"
Consul Template Architecture

{{ secret "secret/foo" }}

CONSUL TEMPLATE

VAULT
✓ Authenticated
✓ Authorized
✓ Result

training

FILE
Exercise: Create Token

Create a new token for Consul Template that uses the "contractor" policy we previously defined. Save this token to your clipboard.
```bash
$ vault token-create -policy=contractor
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>token</td>
<td>9f28c949-37bf-8da3-08a9-29755228e6fa</td>
</tr>
<tr>
<td>token_accessor</td>
<td>588bdd5f-dd63-f38d-1264-f16c20cb3325</td>
</tr>
<tr>
<td>token_duration</td>
<td>2592000</td>
</tr>
<tr>
<td>token_renewable</td>
<td>true</td>
</tr>
<tr>
<td>token_policies</td>
<td>[contractor default]</td>
</tr>
</tbody>
</table>
Terminal

$ cat config.yml.tpl
---
{{- with secret "postgresql/creds/readonly" }}
username: "{{ .Data.username }}"
password: "{{ .Data.password }}"
database: "myapp"
{{- end }}
$ VAULT_TOKEN="..." consul-template -template="config.yml.tpl:config.yml"
$ VAULT_TOKEN="..." consul-template -template="config.yml.tpl:config.yml"

<CTRL+C>
Received interrupt, cleaning up...
$ cat config.yml
$ cat config.yml
---
username: "token-6f40b57e-f0e4-4b42-ac13-d84a309183e3"
password: "23f418d0-9f88-6e7d-d094-4e2575325078"
database: "myapp"
Exercise: Using Consul Template

Create a new policy which permits reading from the generic secret backend (secret/).

Assign this new policy to a token.

Write a CT template that queries the "secret/training" and iterates over all values at that path, using the token you just created to authenticate with Vault.
path "secret/*" {
   capabilities = ["read"]
}

$ vault policy-write readonly-generic ./readonly-generic.hcl
Policy 'readonly-generic' written.

$ vault write sys/policy/readonly-generic rules=@readonly-generic.hcl
Success! Data written to: sys/policy/readonly-generic
$ vault token-create -policy=readonly-generic

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>f763b8f0-cd21-f775-7ef0-df8ad4cbba0a</td>
</tr>
<tr>
<td>token_accessor</td>
<td>f6fbb9df-15a3-b62c-652e-4f272fba4df2</td>
</tr>
<tr>
<td>token_duration</td>
<td>2592000</td>
</tr>
<tr>
<td>token_renewable</td>
<td>true</td>
</tr>
<tr>
<td>token_policies</td>
<td>[default readonly-generic]</td>
</tr>
</tbody>
</table>
{{ with secret "secret/training" }}
{{ range $k, $v := .Data }}
{{ $k }}: {{ $v }}
{{ end }}
{{ end }}
{{ end }}
$ VAULT_TOKEN="..." consul-template -template=template.tpl:out.txt
$ VAULT_TOKEN="..." consul-template -template=template.tpl:out.txt

<CTRL+C>
Received interrupt, cleaning up...
$ cat out.txt
food: pizza
About Envconsul

Despite its name, Envconsul does not require a Consul cluster to operate.

Retrieves secrets from Vault and manages the acquisition and renewal lifecycle.

Requires a token (VAULT_TOKEN) to operate.
Terminal

$ envconsul \
  -secret secret/training \
  -secret secret/foo \
  env

secret_training_food=chicken fingers
secret_training_name=seth
secret_foo_bar=adf1
Envconsul Architecture

- `envconsul -secret=secret/training`
- `ruby my-app.rb`
- `VAULT`
  - Authenticated
  - Authorized
  - Result
Exercise: Create Token

Create a new token for Envconsul that uses the "contractor" policy we previously defined. Save this token to your clipboard.
$ vault token-create -policy=contractor

Key            Value
---            ----- 
token          8a6c5003-5501-5d56-6926-7777ae22ca4c
token_accessor 76f5bc13-2d93-2ee0-7bc1-690461efed02
token_duration 2592000
token_renewable true
token_policies [contractor default]
```ruby
$ cat app.rb
puts <<-STRING.strip
My connection info is:

  username: #{ENV['POSTGRESQL_CREDS_READONLY_USERNAME']}
  password: #{ENV['POSTGRESQL_CREDS_READONLY_PASSWORD']}
  database: my-app"
STRING
```
$ VAULT_TOKEN="..." envconsul -upcase -secret postgresql/creds/readonly ruby app.rb
$ VAULT_TOKEN="..." envconsul -upcase -secret postgresql/creds/readonly ruby app.rb
My connection info is:

username: token-f3d4d6da-d546-3614-3bd6-ac38ff51f21e
password: 6a81a7db-aab8-52b6-ea32-65202b0540ce
database: my-app
Vault Enterprise

Vault Enterprise

Vault is unsealed
Replication disabled

**SECRET BACKENDS**

<table>
<thead>
<tr>
<th>PATH</th>
<th>TYPE</th>
<th>DEFAULT TTL</th>
<th>MAX TTL</th>
<th>FORCE NO CACHE</th>
<th>REPLICATION BEHAVIOR</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>cubbyhole/</td>
<td>cubbyhole</td>
<td>0</td>
<td>0</td>
<td>false</td>
<td>local</td>
<td>per-token private secret storage</td>
</tr>
<tr>
<td>secret/</td>
<td>generic</td>
<td>0</td>
<td>0</td>
<td>false</td>
<td>replicated</td>
<td>generic secret storage</td>
</tr>
<tr>
<td>sys/</td>
<td>system</td>
<td>0</td>
<td>0</td>
<td>false</td>
<td>replicated</td>
<td>system endpoints used for control, policy and debugging</td>
</tr>
</tbody>
</table>
Vault Enterprise

Vault Enterprise

vault is unsealed

Replication disabled

secret /

Filter secrets by name

Create secret

There are currently no secrets in this backend.
Vault Enterprise

Vault Enterprise

vault is unsealed
Replication disabled

secret /
Create a secret at my-secret

SECRET PATH

my-secret

my-key my-value

Add key Create secret Cancel
# Vault Enterprise

Vault Enterprise is a tool for managing secrets and credentials in a secure environment. The image shows the interface for managing secrets, with a focus on a specific key-value pair named `my-secret`.

### Key-Value Pair

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-key</td>
<td>my-values</td>
</tr>
</tbody>
</table>
Further Reading
Further Reading

**CLI**
vaultproject.io/docs/commands

**HTTP API**
vaultproject.io/docs/http

**Internals**
vaultproject.io/docs/internals