Secret Management with Hashicorp's Vault

Daniel Bornkessel
Secret Management with Hashicorp's Vault
<table>
<thead>
<tr>
<th>Fun</th>
<th>I learned a lot</th>
<th>Great speaker</th>
<th>Too technical</th>
<th>Very interesting</th>
<th>Too theoretical</th>
<th>Need more demo</th>
<th>Not enough demo</th>
<th>Too complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 stars</td>
<td>6 stars</td>
<td>7 stars</td>
<td>2 stars</td>
<td>5 stars</td>
<td>4 stars</td>
<td>5 stars</td>
<td>3 stars</td>
<td>3 stars</td>
</tr>
</tbody>
</table>
Secret Management with Hashicorp's Vault

Daniel Bornkessel
Focus of this talk

• what is secret management
• why do you need it
• what is Vault and how can it help you with secret management
• some Vault internals
Goal of this talk

- think about best practices with secrets that your company could improve on
- go and play with Vault
Why focus on Vault

• unmatched (afaik) feature set
• not vendor or framework specific
• open source (mostly ... some closed sourced enterprise features)
Other solutions*

- KeyWiz from Square: not as many features, no dynamic secrets, HSM in open source version
- Cloud Foundry CredHub: tailored and specific to Cloud Foundry
- AWS Secrets Manager: AWS specific, promising, dynamic’esque secrets for certain AWS services, automatic rotation (for supported services + extendable via Lambda functions)
- self made: a lot of complexity and work

* I have not personally used those solutions
Secret Management

INNOQ / Secret Management with Hashicorp’s Vault
Secrets

• sensitive data != secrets ... but: secrets == sensitive data
• tokens
• passwords
• certificates
• API keys
• etc.
Secret Management

• part of your security concept
• one focus: on internal threads like
  • rogue employees
  • unauthorized access to secrets
  • long living secrets
• audit log: who requested credentials for which system at what point of time
• high automation for changing / revoking / rolling secrets
• high entropy passwords
todo: extreme example
Secret Management with Hashicorp’s Vault
Secret Management: current situation

• best practices are widely known
• is usually seen as (very) important
• implementation is hard
• solutions are rare
• apps and frameworks not ready for modern secret management
• high automation still an exception (as opposed to external thread mitigation measures)
• often neglected in favour of business critical features
Question

Who here has production credentials on their laptop at this very moment (e.g. AWS credentials file, DB credentials, passwordless ssh private keys to access machines or git repos, API-keys, etc.)?

Who thinks this is a good idea?
Why am I talking about secret management
About me

Daniel Bornkessel / @kesselborn

- Senior Consultant at INNOQ (part time)
- Focus on DevOps & Continuous Delivery

INNOQ

- Consulting, reviews and development
- https://www.innoq.com/de/culture/working-at-innoq/
Typical project

• Monolith -> Micro Services / Self Contained Systems
• Language: set (mostly Java)
• Framework: set (often Spring Boot)
• Data center: set (mostly AWS or on premise)
• Container Management: set (mostly Kubernetes)
• CI: set (whatever they used before ...please for god’s sake: use Gitlab CI)
• Logging / Monitoring: set (ELK & prometheus)
• Secret Management: sure ... eh ... wat?
Typical project: Secret Management

- we pass secrets in via env vars
- we read the values from Kubernetes secrets
- we have role based access control all figured out
- changing and updating passwords is a manual process for now
- yeah: audit log is something we are looking into
- no, we can not confidently say who has the password for DB xy
- no, we do not change all passwords if an employee leaves the company
- revoking credentials is not something we currently support
Introducing Vault
Vault — executive summary
Vault — executive summary

“A Tool for Managing Secrets”
Vault — executive summary

• not comparable to password managers like 1Password, LastPass, etc.
  • Vault is designed for the system side of things — password managers “just” encrypt your static secrets and provide a nice way use them
Vault — executive summary

- secures, stores and tightly controls
  - tokens
  - passwords
  - certificates
  - API keys
  - and other secrets
Vault — executive summary

- handles
  - leasing
  - key revocation
  - key rolling
  - auditing
- provides an API for all operations
- is not meant as a service or token provider which gets embedded in your request / response cycle
Vault

auth-n + auth-z

Vault

secrets
Vault auth backends

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key /Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit
Vault secret backends

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key /Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit
Vault — secret backends
Vault secret backends — static secrets
Vault secret backends — static secrets

1. `vault write secret/app1/api-key 1234-foo-bar`

2. `vault read secret/app1/api-key`

3. `1234-foo-bar`
Vault secret backends — dynamic secrets
Vault secret backends — dynamic secrets

What they are

• on-the-fly created credentials (hence dynamic) for each instance of an app / user who wants a secret
• usually short to medium long ttl
• fully audited
Vault secret backends — dynamic secrets

How they work (in a Nutshell)

1. provide Vault credentials for a user that has rights to create users or tokens in a remote system (e.g. db)
2. configure Vault with settings on how to create credentials
3. configure Vault with settings on how to invalidate credentials in the remote system
Vault secret backends — Databases
Vault secret backends — Databases

- Idea: get access to databases
- Vault gets configured with credentials for a database user that has necessary permissions on the database
- Vault gets a policy that maps users and roles to users with configured permissions in the database
- when user requests credentials, Vault creates a new database user on the fly
- when configured (usually the case), all created users have a ttl assigned — when the ttl is reached, Vault deletes the user from the database
Secret Management with Hashicorp's Vault

### Vault secret backends — Databases

```
├── aws
│   └── creds
│       ├── admin
│       └── developer
├── database
│   └── creds
│       ├── clients
│       └── contracts
├── pki
│   └── issue
│       └── example.com
├── secret
│   └── team
│       ├── app1
│       │   └── api-keys
│       │       ├── google-analytics
│       │       └── paypal
│       └── app2
│           └── foo
├── ssh
│   └── creds
│       ├── erika
│       └── erna
└── transit
    ├── decrypt
    └── encrypt
```

0 db admin config
Vault secret backends — Databases

```
vault secrets enable -path=db database
vault write db/config/clients \
  plugin_name=mysql-database-plugin \
  connection_url="admin:pw@tcp(db.example.com)/" \
  allowed_roles="clients-ro,clients-rw"

vault write database/roles/clients-ro \
  db_name=clients \
  creation_statements="" \
  CREATE USER '{{name}}'@'%' IDENTIFIED BY \
  '{{password}}'; \
  GRANT SELECT ON clients.* TO '{{name}}'@'%';" \
  default_ttl="1h" \
  max_ttl="240h"
```
Vault secret backends — Databases

```
vault secrets enable -path=db database

vault write db/config/clients 
    plugin_name=mysql-database-plugin 
    connection_url="admin:pw@tcp(db.example.com)/" 
    allowed_roles="clients-ro, clients-rw"

vault write database/roles/clients-ro 
    db_name=clients 
    creation_statements="\n    CREATE USER '{{name}}'@'%' IDENTIFIED BY \n    '{password}'; \n    GRANT SELECT ON clients.* TO '{{name}}'@'%';" 
    default_ttl="1h" 
    max_ttl="240h"
```
Vault secret backends — Databases

1. Read `db/creds/clients-ro` role
2. Create user ...
3. OK
4. DB login / DB password
5. Delete user ...
Vault secret backends — Databases

Available Plugins:

• Cassandra
• HanaDB
• MongoDB
• MSSQL
• MySQL/MariaDB
• PostgreSQL
• Oracle
Vault secret backends — Google Cloud
Vault secret backends — Google Cloud

credentials.json
...or service account

define rolesets to generate oauth2 access tokens (preferred) or Service Accounts
Vault secret backends — Google Cloud

1. **App**
   - read `gcp/token/dev`
   - configured roleset

2. **Vault**
   - `create token / service account`

3. **GCP Api**
   - `oauth2 token / service account`

4. **oauth2 token / service account key**
Vault secret backends — AWS
Vault secret backends — AWS

- Idea: get access to AWS resources
- Vault gets configured with an AWS user that has necessary permissions
- Vault gets a policy that maps users or roles to AWS roles
- when user requests credentials, Vault creates STS tokens, assume role tokens or dynamic IAM users
- when configured (usually the case), all created secrets have a ttl assigned
Vault secret backends — AWS

aws_access_key
aws_secret_key
Vault secret backends — AWS

0. aws_access_key
aws_secret_key

1. configured role
   read aws/creds/dev

2. create STS / IAM user

3. TTL’ed Token / credentials

4. TTL’ed Token / credentials

Vault

App

AWS API
Vault secret backends — PKI
Vault secret backends — PKI

- Idea: issue client certificates on the fly
- Vault gets configured a CA Certificate and a private key
- Vault gets a configuration about how certificates for this CA should be issues (ttl, subject, etc.)
- when user requests credentials, Vault issues a certificate on the fly
- when configured (usually the case), all created certificates have a ttl assigned
Vault secret backends — PKI

0

add ca.cert & ca.key

Vault
Vault secret backends — PKI

1. Write `pki/issue/dev` configured role
2. Create client certificate
3. Get TTL'ed client certificate
Vault secret backends — SSH
Vault dynamic secret backends — ssh

One-Time SSH Passwords

- Idea: get ssh access to machines
- every host in the system has a small Vault-helper process running
- user fetches a one time password from Vault
- when authenticating via ssh, the Vault-helper checks, whether the one time password is valid and deletes it
Vault secret backends — SSH

1. `vault write ssh/creds/dev ip=1.1.2.2`
2. `username / otp`
3. `ssh username@1.1.2.2`
4. `validate otp`
5. `grant access`
Vault dynamic secret backends — ssh

Signed SSH Certificates

- Idea: get ssh access to machines
- user configures Vault-ssh with a CA, a private and a public key
- the public key gets distributed to all system hosts
- the user asks Vault to sign one of his public ssh keys with the provided CA and gets a new, signed public key as a response
- the user can use this new, signed key to login to machines
Vault dynamic secret backends — Transit
Vault dynamic secret backends — Transit

- Idea: de- and encrypt data without handling private keys
- User creates a new transit path in Vault
- Users can encrypt data by writing the data to this transit path (e.g. `transit/encrypt/my-keys/foo`)
- Users with sufficient permissions can decrypt data by writing to the respective transit path (e.g. `transit/decrypt/my-keys/foo`)
- the private key never leaves Vault
- the data is **not** stored on Vault (hence the name transit)
Vault secret backends — Transit

1. `vault write transit/encrypt/app/app1`  
   ![Diagram of vault write encrypt]

2. `vault write transit/describe/app/app1`  
   ![Diagram of vault write describe]

3. `vault write transit/decrypt/app/app1`  
   ![Diagram of vault write decrypt]

4. `foo`  
   ![Diagram of vault write decrypt]

INNOQ / Secret Management with Hashicorp’s Vault
Vault secret backends

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key /Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit
Vault — auth backends
Vault auth backends

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

Vault

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key /Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit
Vault auth backends — tokens
token auth

- created by Vault
- only way to authorize (auth-z) against Vault
- returned when authenticated (auth-n) successfully
- comparable to a session-id on a website
- has permissions / policies assigned to it
token auth

$ vault token create -ttl=5m -policy=admin

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>d9640590-63c8-b3a6-50ac-1403c8180948</td>
</tr>
<tr>
<td>token_accessor</td>
<td>5a362982-f34c-3706-143a-26ada278b6cf</td>
</tr>
<tr>
<td>token_duration</td>
<td>5m</td>
</tr>
<tr>
<td>token_renewable</td>
<td>true</td>
</tr>
<tr>
<td>token_policies</td>
<td>[admin default]</td>
</tr>
</tbody>
</table>
Vault auth backends — userpass
userpass auth

- statically created by users and stored in Vault

$ vault auth enable userpass

$ vault write auth/userpass/users/kesselborn \
   password=foo policies=admin

$ vault login -method=userpass username=kesselborn

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td>d9640590-63c8-b3a6-50ac-1403c8180948</td>
</tr>
<tr>
<td>token_duration</td>
<td>5m</td>
</tr>
<tr>
<td>token_policies</td>
<td>[admin default]</td>
</tr>
</tbody>
</table>
Vault auth backends — userpass

• setup username / password

1. `vault write /auth/userpass/users/foo password 123`
Vault auth backends — userpass

- authenticate with a username & password

1. foo / password123
2. token
Vault auth backends — TLS certificates
Vault auth backends — TLS certificates

- setup TLS certificate authentication

```bash
vault write auth/cert/certs/web \
  ... \
  certificate=@web-cert.pem
```

 Vault auth backends — TLS certificates

- setup TLS certificate authentication

```bash
vault write auth/cert/certs/web \
  ... \
  certificate=@web-cert.pem
```
Vault auth backends — TLS certificates

- authenticate with a TLS client certificate
Vault auth backends — external identity providers
Vault auth backends — LDAP / Radius / Okta auth

- $SERVICE is used as an identity provider (using LDAP here)
LDAP auth

$ vault write auth/ldap/config \
  url="ldaps://ldap.example.com" \ 
  userattr="uid" \ 
  userdn="ou=People,dc=innoq,dc=com" \ 
  binddn="cn=vaultuser,dc=example,dc=com" \ 
  bindpass="3cK{hrh7hi/Hj" \ 
  groupdn="ou=Group,dc=example,dc=com" \ 
  starttls=true

$ vault write auth/ldap/groups/employee policies=employee

$ vault write auth/ldap/users/kesselborn policies=admin
Github auth

- Github is used as an identity provider
Vault auth backends — Kubernetes auth
Vault auth backends — Kubernetes auth

1. K8s container
2. auth against k8s API
3. service_account_name: app1
   service_account_namespace: default
4. token
Vault auth backends — Kubernetes auth

1. K8s token
2. auth against K8s API
3. success / failure
4. token
5. token
Vault auth backends — Kubernetes auth

$ vault auth enable kubernetes

$ vault write auth/kubernetes/config
  kubernetes_host="https://api.k8s.example.com" 
  kubernetes_ca_cert="@ca.crt"

$ vault write auth/kubernetes/role/demo 
  bound_service_account_names=vault-auth 
  bound_service_account_namespaces=default 
  policies=default 
  ttl=1h
Vault auth backends — GCE auth
Vault auth backends — GCE auth

1. **JWT** generated by **Vault**
2. **JWT** sent by **GCE Instance**
3. **JWT** verified by **Vault**
4. **JWT** signature verified against **OAuth2 API**
5. **Token** returned by **Vault**

**GCE Instance**

```
curl -H "Metadata-Flavor: Google" \
'http://metadata/computeMetadata/v1/instance/service-accounts/default/identity?audience=[AUDIENCE]&format=[FORMAT]'`
Vault auth backends — AWS auth
Vault auth backends — AWS auth

- Vault checks passed in data was encrypted with a AWS private key
- can be limited to instances which have a specific instance role applied
- can be limited (and usually is) to allow one authentication per ec2 instance only
- after authentication, roles and policies are mapped as usual
Vault auth backends — AWS auth

1. EC2 MetaData Services
2. EC2 Instance
3. Vault
4. verify PKCS#7 signature against AWS public keys
5. verify instance
6. instance_id: i-a832f734
   ami_id: ami-f083709d
   ...
7. (optionally) set instance on blacklist to avoid double authentication
Vault auth backends — AWS auth

$ vault write auth/aws/role/dev-role \
  auth_type=ec2 \
  bound_ami_id=ami-fce3c696 \
  policies=prod,dev max_ttl=500h

$ vault write auth/aws/role/dev-role-iam \
  auth_type=iam \
  bound_iam_instance_profile_arn=... \
  policies=prod,dev max_ttl=500h
Vault auth backends — AWS auth

- alternatively: IAM auth method
- client signs a GetCallerIdentity query using the AWS Signature v4 algorithm and submits 4 pieces of information to the Vault server to recreate a valid signed request
- https://www.vaultproject.io/docs/auth/aws.html#iam-auth-method
Vault auth backends — AppRole
Vault auth backends — AppRole

- a generic approach to authenticate machines or applications
- an AppRole can be created for a particular machine, a particular user on that machine, or a service spread across machines
- for authenticating, two values are needed
  - RoleID: static, can live with an app or on a machine
  - SecretID: gets created on the fly before authenticating
Vault auth backends — AppRole

- mainly used for machines or apps to authenticate against Vault

1. request SecretID
2. SecretID
3. SecretID
4. RoleID + SecretID
5. token
Vault auth backends — AppRole

• mainly used for machines or apps to authenticate against Vault
Vault auth backends

- **Tokens**
- **LDAP**
- **AWS**
- **Kubernetes**
- **Google Cloud**
- **Username & Password**
- **AppRole**
- **GitHub**
- **MFA**
- **Okta**
- **RADIUS**
- **TLS Certificates**

- **AWS**
- **Consul**
- **Cubbyhole**
- **Databases**
- **Identity**
- **Static secrets (Key /Value)**
- **Nomad**
- **PKI (Certificates)**
- **RabbitMQ**
- **SSH**
- **TOTP**
- **Transit**
Use whatever the auth you want
Vault

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key/Value)
- Nomad
- PKI -> Kubernetes access
- RabbitMQ
- SSH
- TOTP
- Transit
Vault

- Tokens
- LDAP
- AWS
- **Kubernetes**
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

• AWS
• Consul
• Cubbyhole
• **Databases**
• Identity
• Static secrets (Key /Value)
• Nomad
• PKI (Certificates)
• RabbitMQ
• SSH
• TOTP
• Transit
Vault — policies
Vault — secret representation

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

---

Vault — secret representation

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key/Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit
Vault — secret representation
Vault — policies

- applied to “files” or “directories”
- support filesystem wildcards
- control what a user can access
- get assigned after authentication
- policies of a token can’t be changed
Vault — policies

- create (c)
- read (r)
- update (u)
- delete (d)
- list (l)
- deny (d)
- sudo (s)
Vault — policies

$ cat app1-policy.hcl

path "secret/team/app1/*" {
capabilities = ["read", "list"]
}

path "pki/issue/broker" {
capabilities = ["write"]
}

path "database/creds/clients-ro" {
capabilities = ["read"]
}
Vault — policies

$ cat app1-erna-policy.hcl

```hcl
path "secret/team/app1/*" {
  capabilities = ["read", "list", "create", "update", "delete"]
}
path "pki/role/*" {
  capabilities = ["create", "update", "delete"]
}
path "pki/config/*" {
  capabilities = ["create", "update", "delete"]
}
path "database/config/clients" {
  capabilities = ["create", "update", "delete"]
}
path "database/role/clients-ro" {
  capabilities = ["create", "update", "delete"]
}
path "ssh/creds/developers" {
  capabilities = ["read"]
}
```
Vault — Audit log
Vault internals — Audit log

- off by default
- supported backend
  - file
  - syslog
  - socket
- if audit log can not be written, Vault does not reply to requests
Vault internals — Audit log

- every operation creates a log entry with
  - what was done
  - when was it executed
  - by who was it requested
  - request payload
  - response payload
- sensitive data is hashed with a salt using HMAC-SHA256
Vault internals — Audit log

- {"time": "2018-10-10T10:59:53.557231528Z", "type": "response", "auth": 
  {"client_token": "hmac-sha256:41f2474f04f6277eb43cc8eae700dbc8534c5369d9185991eed4c4f70b1a5840", "accessor": "hmac-sha256:27e400da69c94fce2378f5738cbf950531d7a9513215274abfbbdaa4927e00ba", "display_name": "ldap-daniel.bornkessel@innoq.com", "policies": ["default"], "token_policies": ["default"], "metadata": 
  {"username": "daniel.bornkessel@innoq.com"}, "entity_id": "8950f5f7-fad8-3ecb-4e62-e5841815df60"}, "request": 
  {"id": "9f2b6dfa-5c18-af6a-1f66-2c78b25a875f", "operation": "list", "client_token": "hmac-sha256:41f2474f04f6277eb43cc8eae700dbc8534c5369d9185991eed4c4f70b1a5840", "client_token_accessor": "hmac-sha256:27e400da69c94fce2378f5738cbf950531d7a9513215274abfbbdaa4927e00ba", "path": "secret/", "data": null, "policy_override": false, "remote_address": "100.96.0.76", "wrap_ttl": 0, "headers": {}}, "response": "error": "hmac-sha256:d9d7a78363fd091f1b4c12629b7c9b5d7a7ffbf904ef5d29d002d5265d5bbf33"}, "error": "1 error occurred:

* permission denied"}
Secret Management with Hashicorp's Vault

Vault

- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates

- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key/Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit

policies

audit logs
Vault internals
Vault internals — storage
Vault internals — storage

- several storage backends available: Consul, Etcd, Azure, Cassandra, CockroachDB, CouchDB, DynamoDB, Filesystem, FoundationDB, Google Cloud Spanner, Google Cloud Storage, In-Memory, Manta, MySQL, PostgreSQL, S3, Swift, Zookeeper
- data encrypted at rest with a symmetric key
- symmetric key is encrypted by “master key” and stored on storage backend
- master key is encrypted with “Shamir’s Secret Sharing”
Vault internals — storage

Shamir’s Secret Sharing

• 1 ... N keys are needed in order to decrypt the data
• you can provide the decryption keys in any order
• N ... N+M keys can be created and distributed to different parties
Vault internals — storage

Shamir’s Secret Sharing

- by default, Vault creates 5 keys on initialization (which is a once per storage backend operation)
- 3 of the 5 keys are needed in order to unseal a Vault instance
- this is configurable (e.g. 10/8, 15/5, etc.)
Vault internals — storage

Shamir's Secret Sharing

• HA of key holders
• one key alone is worthless
• key holder != admins: designers, ops, devs, etc.
• new unsealing keys can be created when provided enough unsealing keys (e.g. when employees leave the company)
• every time a Vault instance is started, the master key has to be decrypted
Vault internals — HA
Vault internals — HA

• some backends support Vault HA mode (currently: Consul, Etcd, DynamoDB, Foundation DB, Google Cloud Spanner, Google Cloud Storage, MySQL, Zookeeper)

• Active-Passive mode:
  • only the active Vault instance replies to requests
  • all other Vault instances reply with a HTTP 302 to the active Vault instance (i.e. LB in front of HA Vaults does not make sense)
• leader election done in storage backend
Vault usage
Vault usage — integration
Vault usage — integration

• some frameworks have integration for Vault
• when home made solution
  • create config files with a helper app to avoid development pain
  • prepare your app for ttl'ed credentials: react accordingly if the (e.g.) DB password is not valid anymore:
    • re-read config file with new credentials
    • make sure, helper app gets new credentials in time
    • re-try DB request
    • when in a container managed system, exit if appropriate
Vault usage — getting started
Vault — getting started (1 minute invest)

https://www.vaultproject.io/#/demo/0
Vault — getting started

• interactive tutorial

• download it locally and start it with ‘--dev’ parameter
  (investment: 20 min - a few hours)

• there is a steep learning curve
  • different backends use the same words with different meanings
    (ttl, tokens, etc.)
  • hard to quickly test something as you need the backend
    systems in place: AWS auth to get MySQL passwords?
  • most tutorials only run in dev mode
Vault — recap

You authenticate somehow, get a token with some policy attached to it, which again allows you to read some secrets.
Thank you and auf Wiedersehen

We are hiring in Hamburg, Berlin, Munich, Frankfurt, Monheim (between Düsseldorf and Cologne), and remote

https://www.innoq.com/en/culture/working-at-innoq/
https://www.innoq.com/de/culture/working-at-innoq/