



# **Secrets Management with HashiCorp's Vault**

**Daniel Bornkessel**

**INNOQ**

# Focus of this talk

- what is secrets management
- why do you need it
- what is Vault and how can it help you with secret management
- Vault / Kubernetes / MySQL demo

# Why focus on Vault

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

# Secrets Managemant

# Secrets

- sensitive data != secrets ... but: secrets == sensitive data
- tokens
- passwords
- certificates
- API keys
- etc.

# Secrets Management

- part of your security concept
- one focus: on internal threats 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

# Secrets 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 threat mitigation measures)
- often neglected in favour of business critical features

# Vault alternatives\*

- **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



# 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

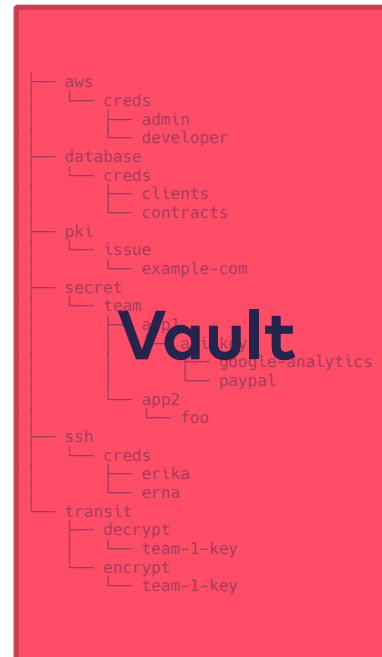
- **secures, stores and tightly controls**
  - **tokens**
  - **passwords**
  - **certificates**
  - **API keys**
  - **and other secrets**

# 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

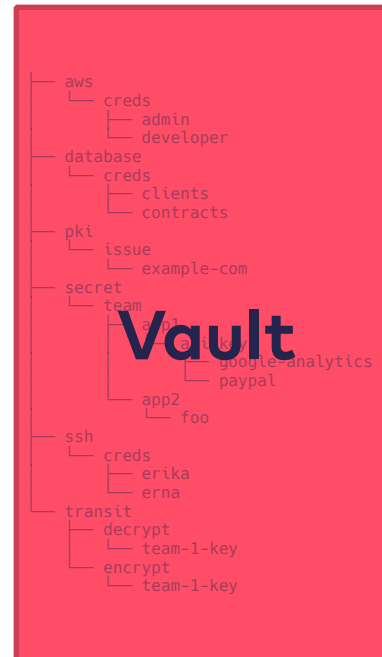
auth-n + auth-z



secrets

# Vault auth backends

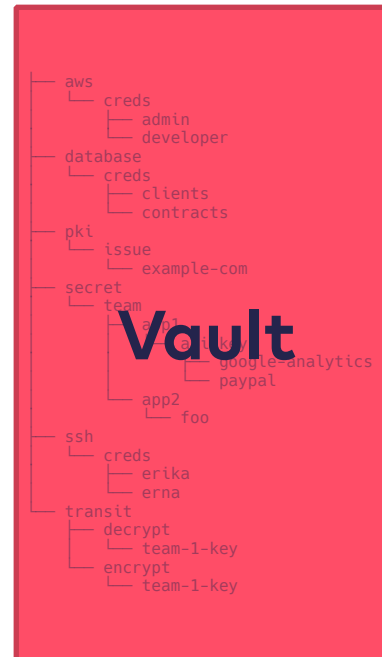
- Tokens
- LDAP
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates



- 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



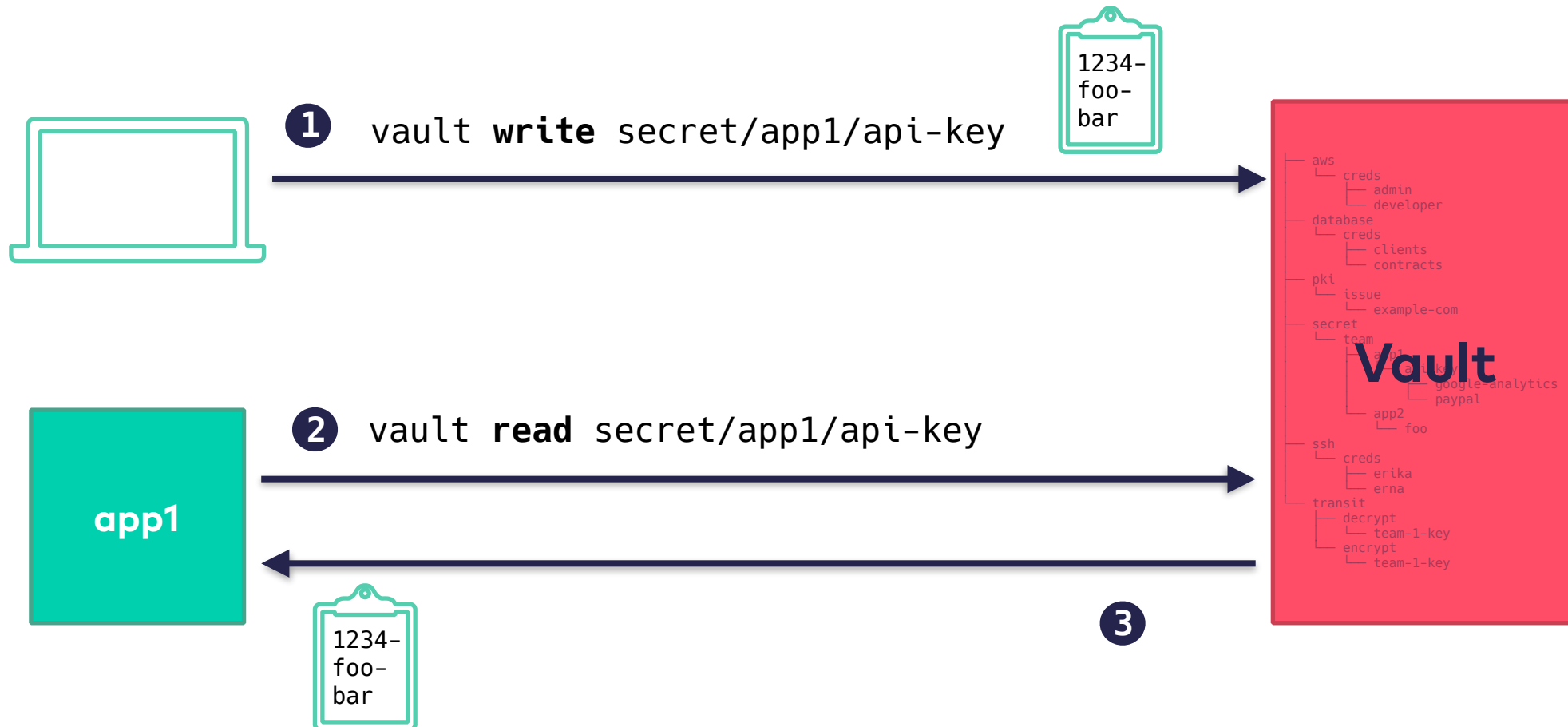
- **AWS**
- **Consul**
- **Cubbyhole**
- **Databases**
- **Identity**
- **Static secrets (Key /Value)**
- **Nomad**
- **PKI (Certificates)**
- **RabbitMQ**
- **SSH**
- **TOTP**
- **Transit**

# Vault — secret backends

# Vault secret backends — static secrets



# Vault secret backends — static secrets



# 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

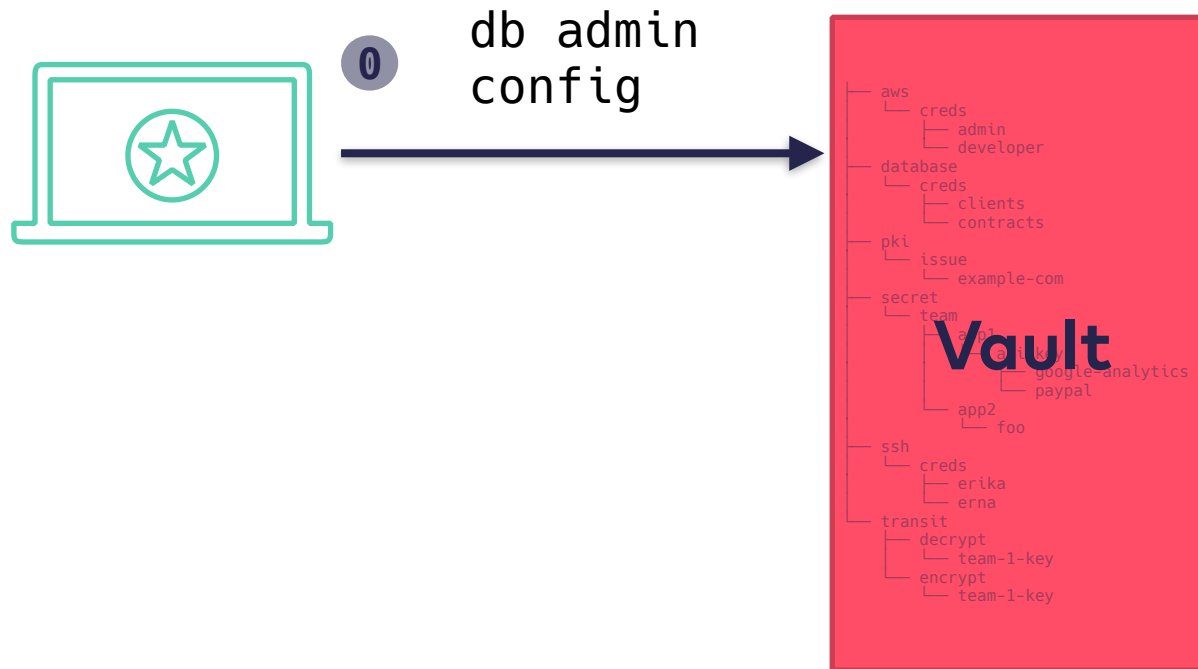
# **Vault secret backends — dynamic secrets**

## **How they work (in a Nutshell)**

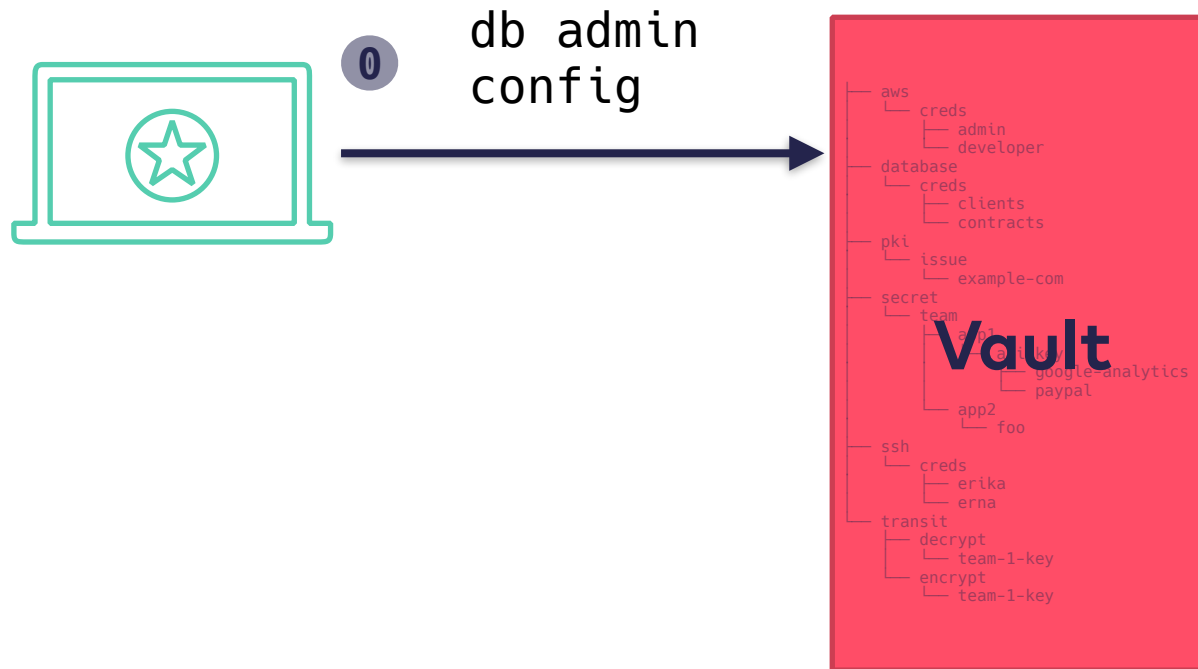
- 1. provide Vault credentials for a user that has rights to create users or tokens in a remote system (e.g. db)**
- 2. configure Vault with settings on how to create credentials**
- 3. configure Vault with settings on how to invalidate credentials in the remote system**

# Vault secret backends — Databases

# Vault secret backends — Databases



# Vault secret backends — Databases

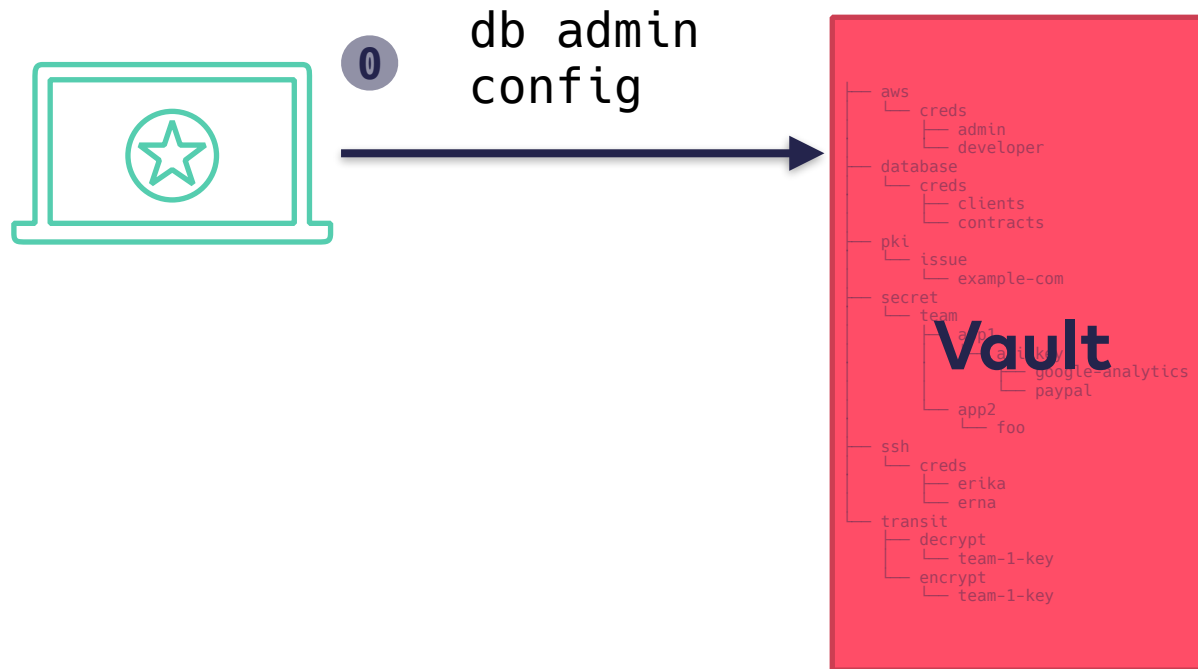


```
vault secrets enable -path=db database
```

```
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



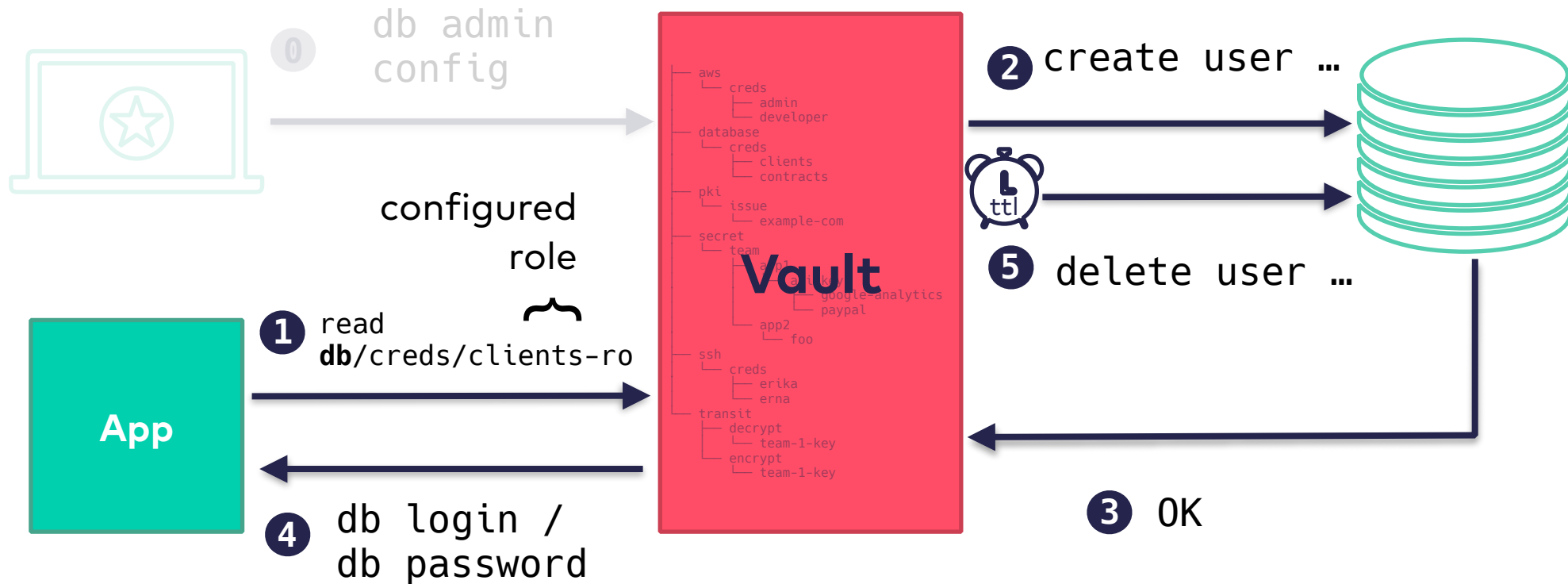
```
vault secrets enable -path=db database
```

```
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="\
    CREATE USER '{{name}}'@'%' IDENTIFIED BY \
    '{{password}}'; \
    GRANT SELECT ON clients.* TO '{{name}}'@'%';" \
  default_ttl="1h" \
  max_ttl="240h"
```



# Vault secret backends — Databases



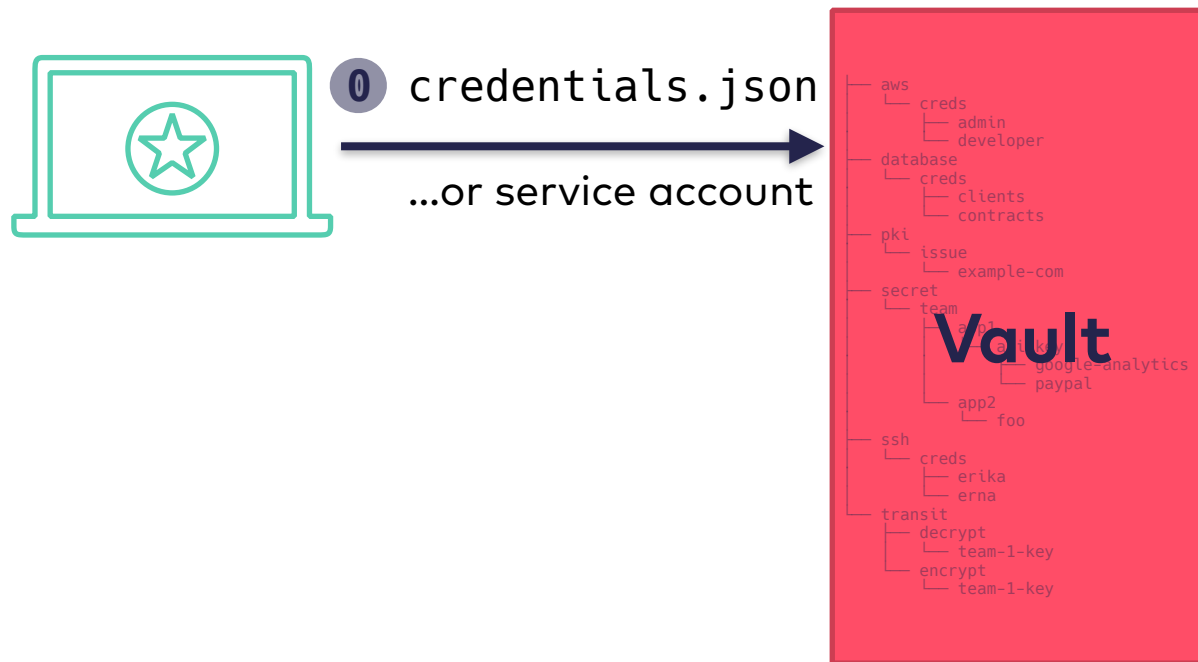
# Vault secret backends — Databases

## Available Plugins:

- Cassandra
- HanaDB
- MongoDB
- MSSQL
- MySQL/MariaDB
- PostgreSQL
- Oracle

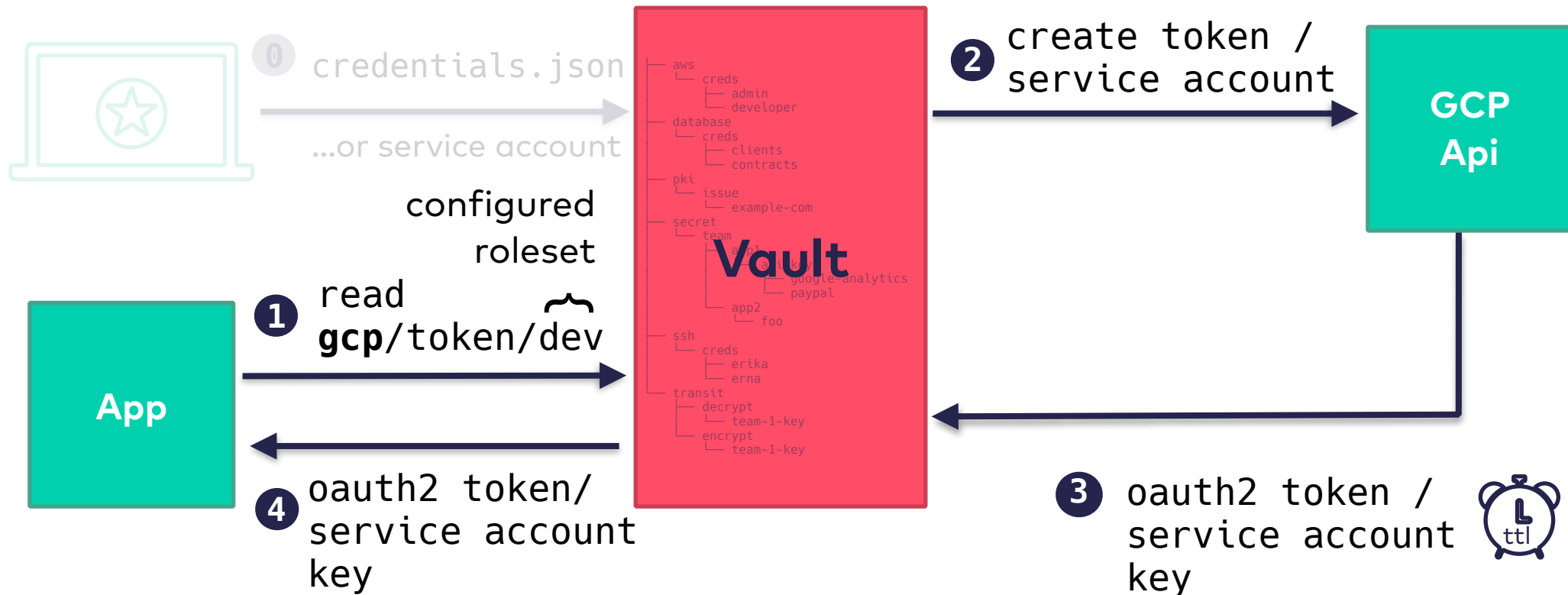
# Vault secret backends — Google Cloud

# Vault secret backends — Google Cloud



define rolesets to generate  
oauth2 access tokens (preferred)  
or Service Accounts

# Vault secret backends — Google Cloud

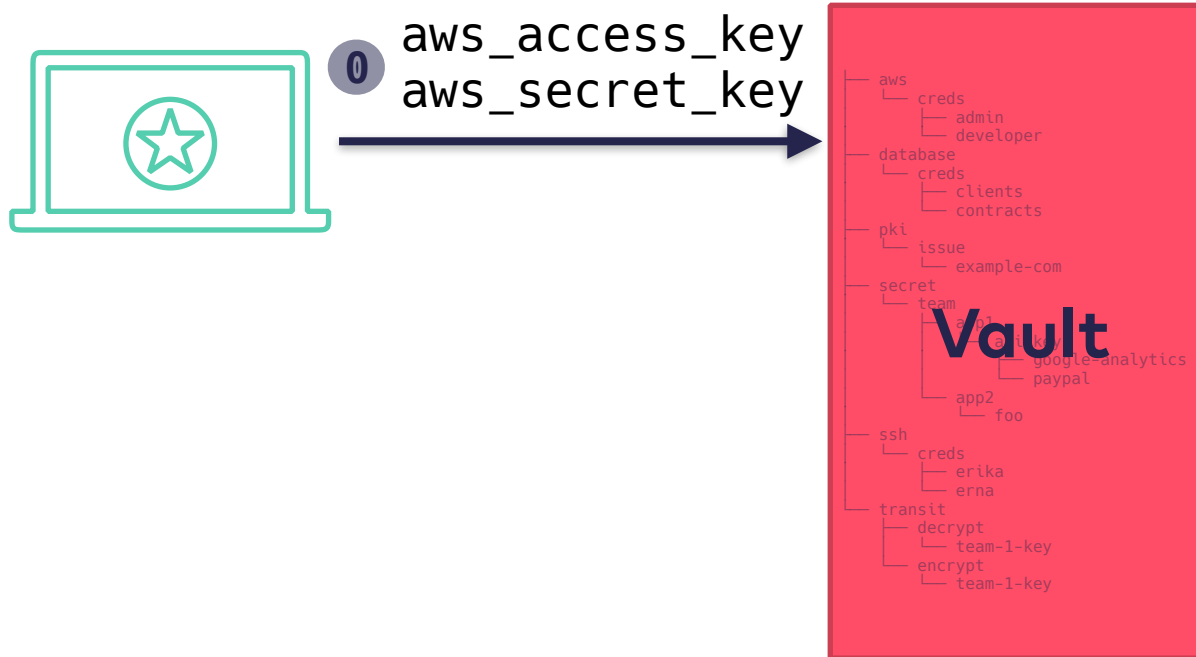


# Vault secret backends — AWS

# Vault secret backends — AWS

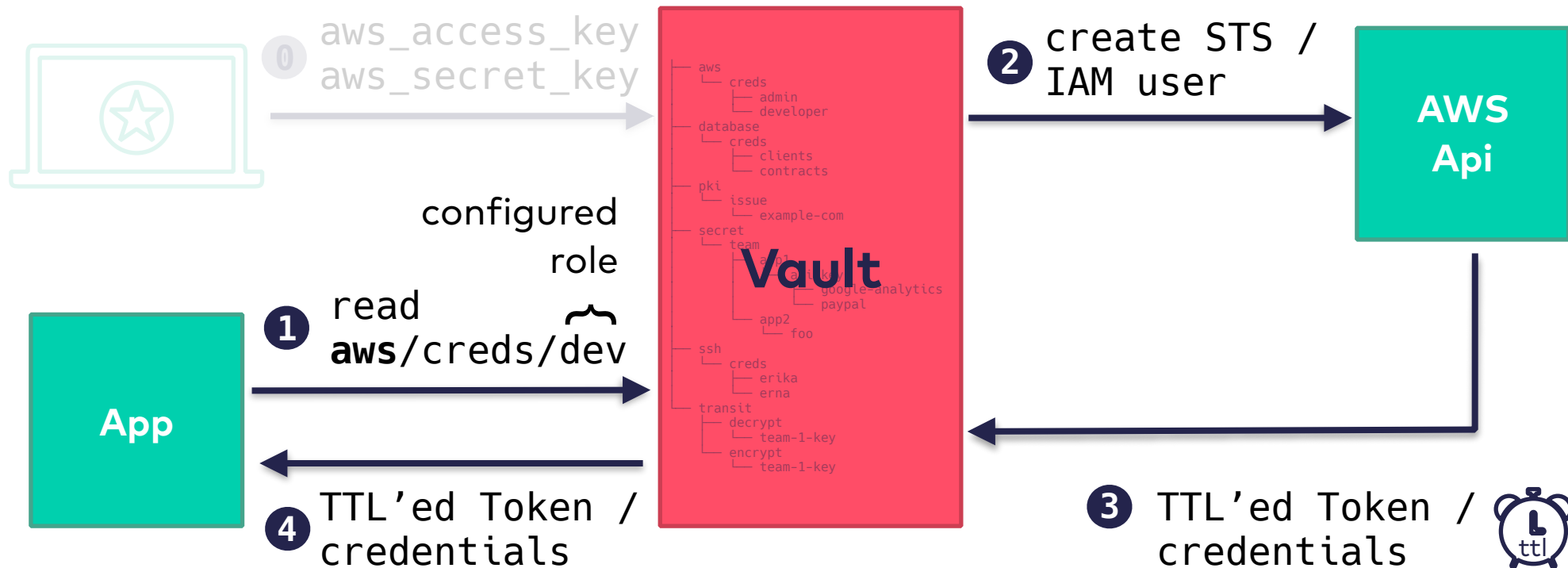
- **Idea: get access to AWS resources**
- **Vault gets configured with an AWS user that has necessary permissions**
- **Vault gets a policy that maps users or roles to AWS roles**
- **when user requests credentials, Vault creates STS tokens, assume role tokens or dynamic IAM users**
- **when configured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

# Vault secret backends — AWS



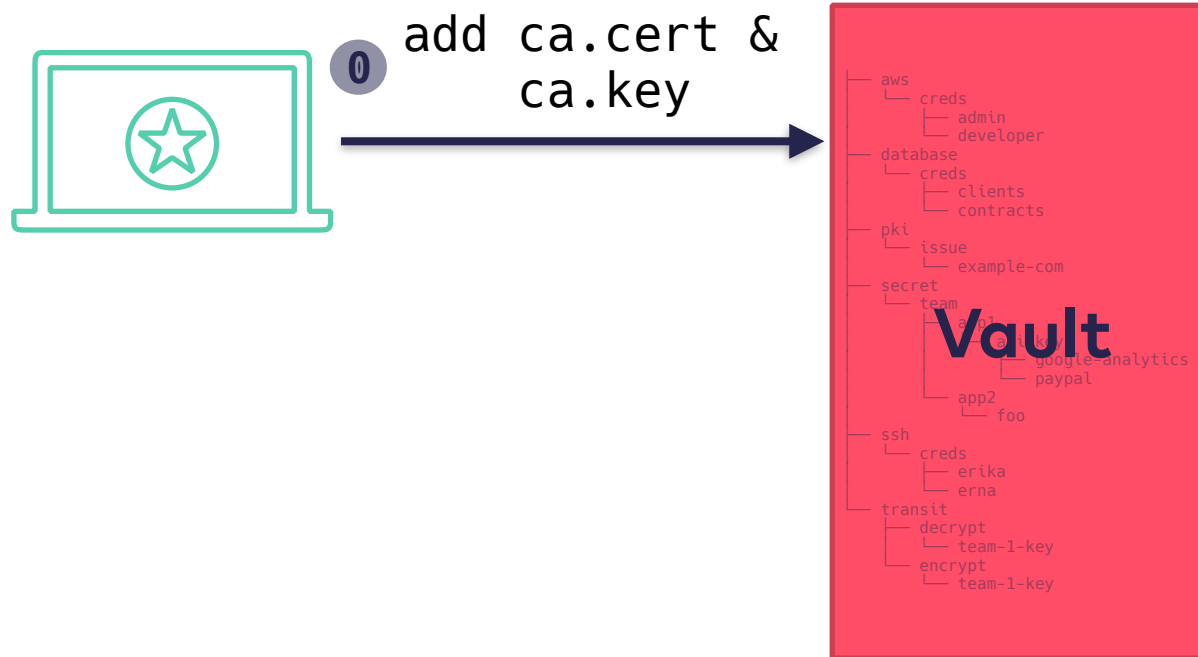


# Vault secret backends – AWS

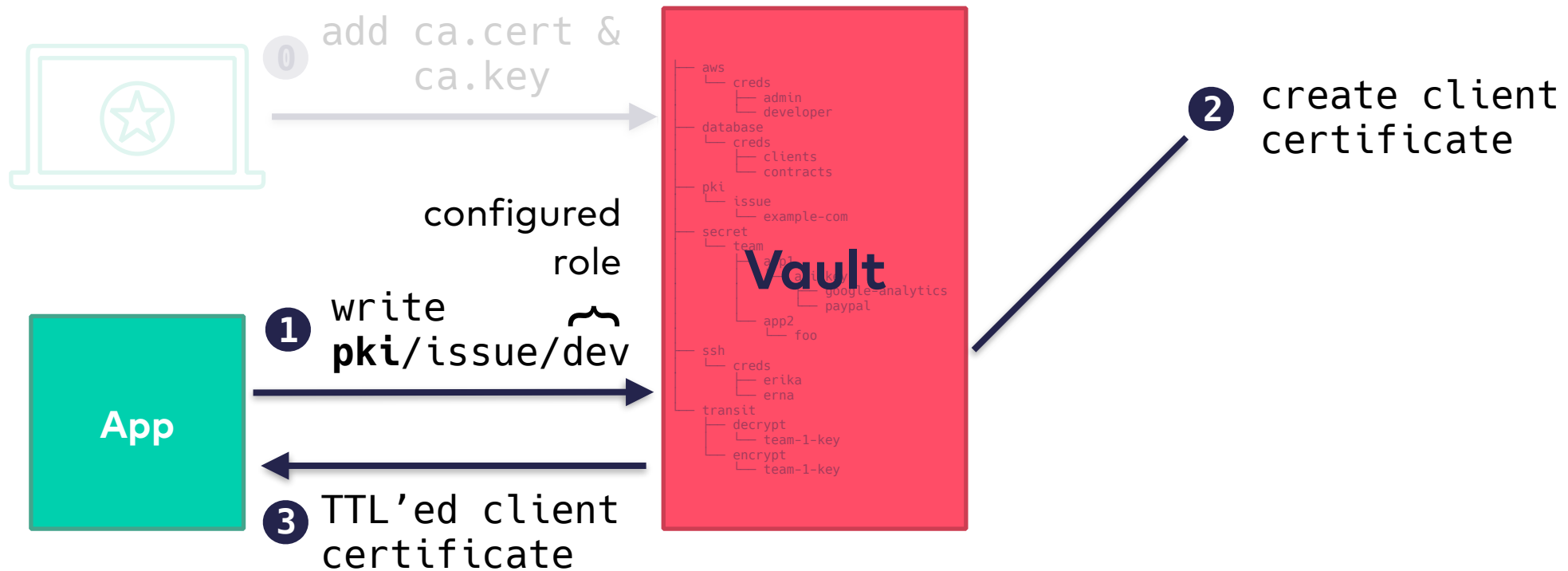


# Vault secret backends — PKI

# Vault secret backends — PKI



# Vault secret backends — PKI



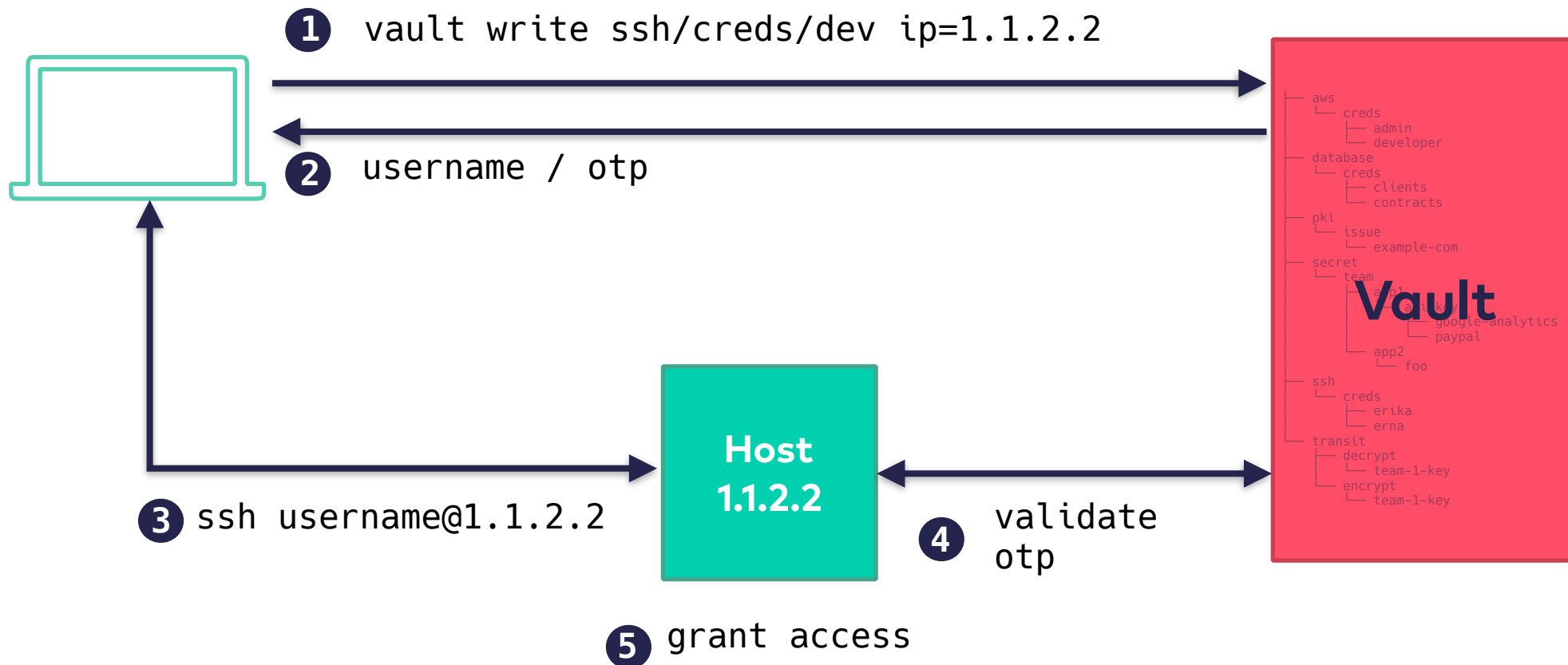
# Vault secret backends — SSH

# 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



# 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

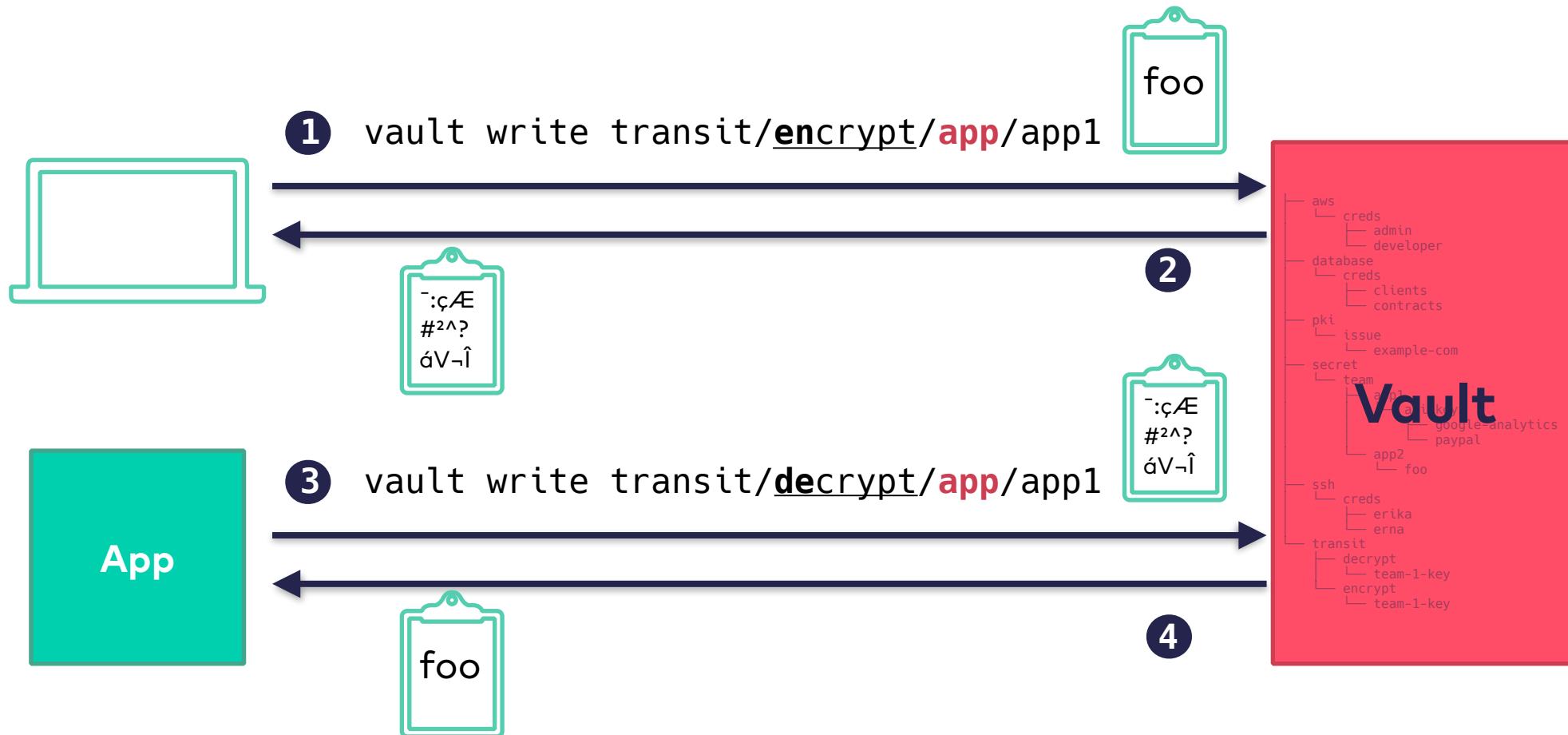


# Vault dynamic secret backends — Transit

# Vault dynamic secret backends — Transit

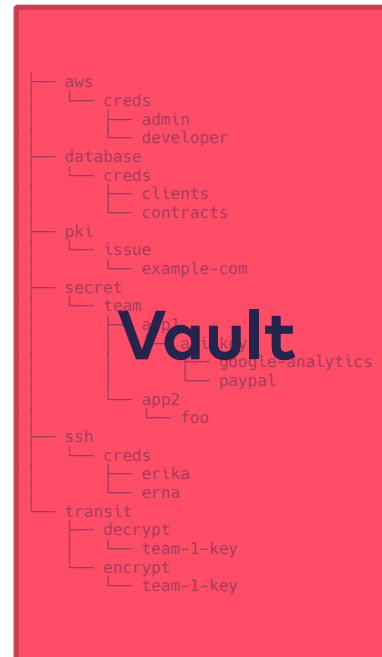
- Idea: de- and encrypt data without handling private keys
- User creates a new transit path in Vault
- Users can encrypt data by writing the data to this transit path  
(e.g. `transit/encrypt/my-keys/foo`)
- Users with sufficient permissions can decrypt data by writing to the respective transit path  
(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



# Vault secret backends

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

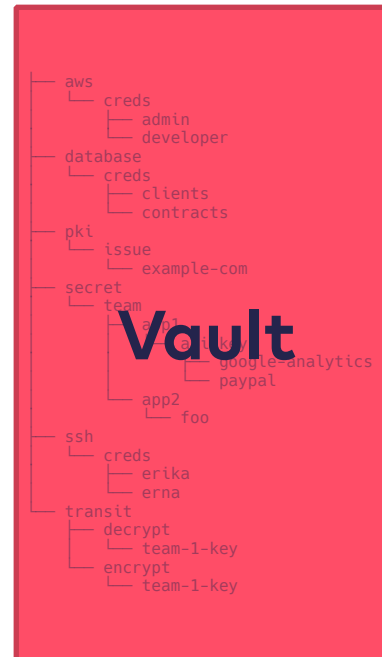


- **AWS, GCloud, Azure,**
- **AliCloud**
- **Cubbyhole**
- **Databases**
- **Identity (OIDC ID tokens (JWT), JWKS endpoint)**
- **Static secrets (Key/Value)**
- **Consul, Nomad**
- **PKI (Certificates)**
- **RabbitMQ**
- **SSH**
- **TOTP**
- **Transit**

# Vault — auth backends

# Vault auth backends

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



- 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
```

Key	Value
---	-----
token	d9640590-63c8-b3a6-50ac-1403c8180948
token_accessor	5a362982-f34c-3706-143a-26ada278b6cf
token_duration	5m
token_renewable	true
token_policies	[admin default]

# Vault auth backends — userpass

# Vault auth backends — TLS certificates

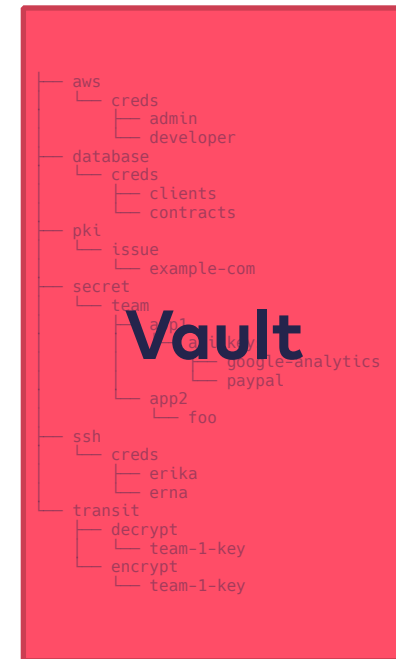
# Vault auth backends — TLS certificates

- setup TLS certificate authentication



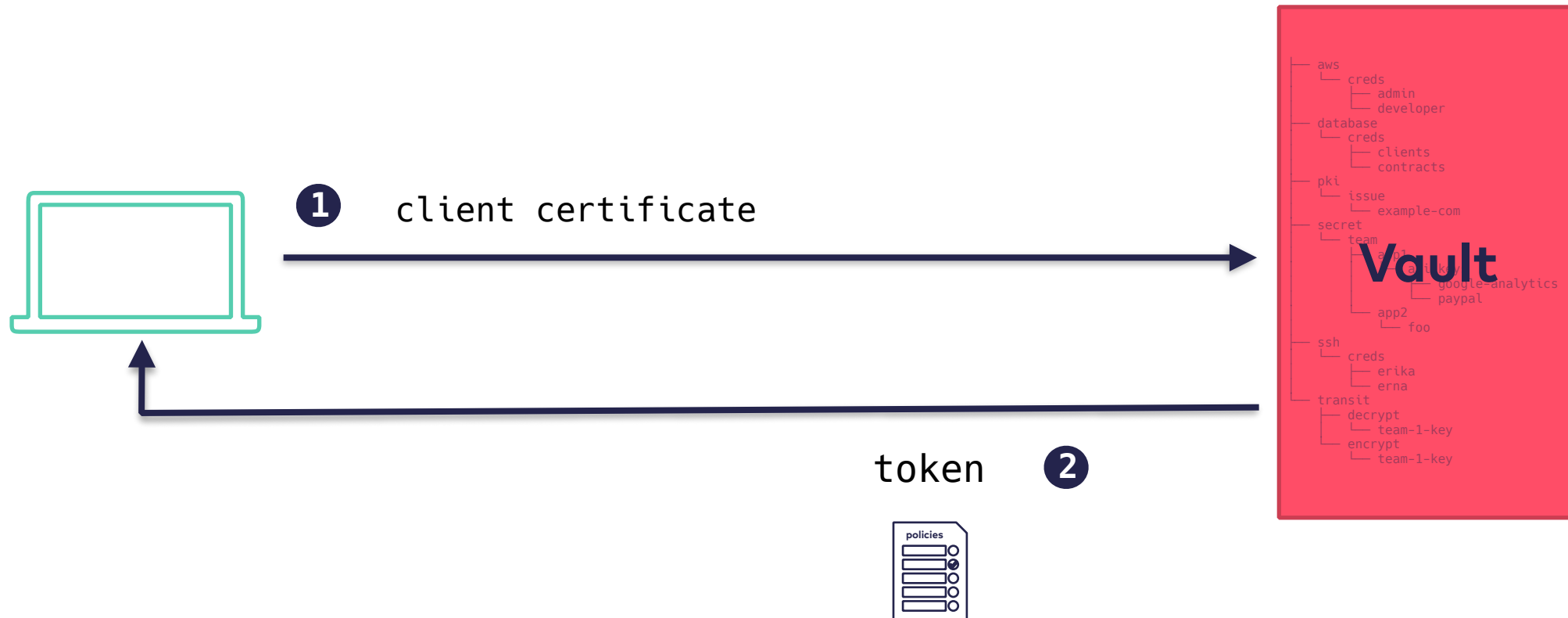
1

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



# Vault auth backends — TLS certificates

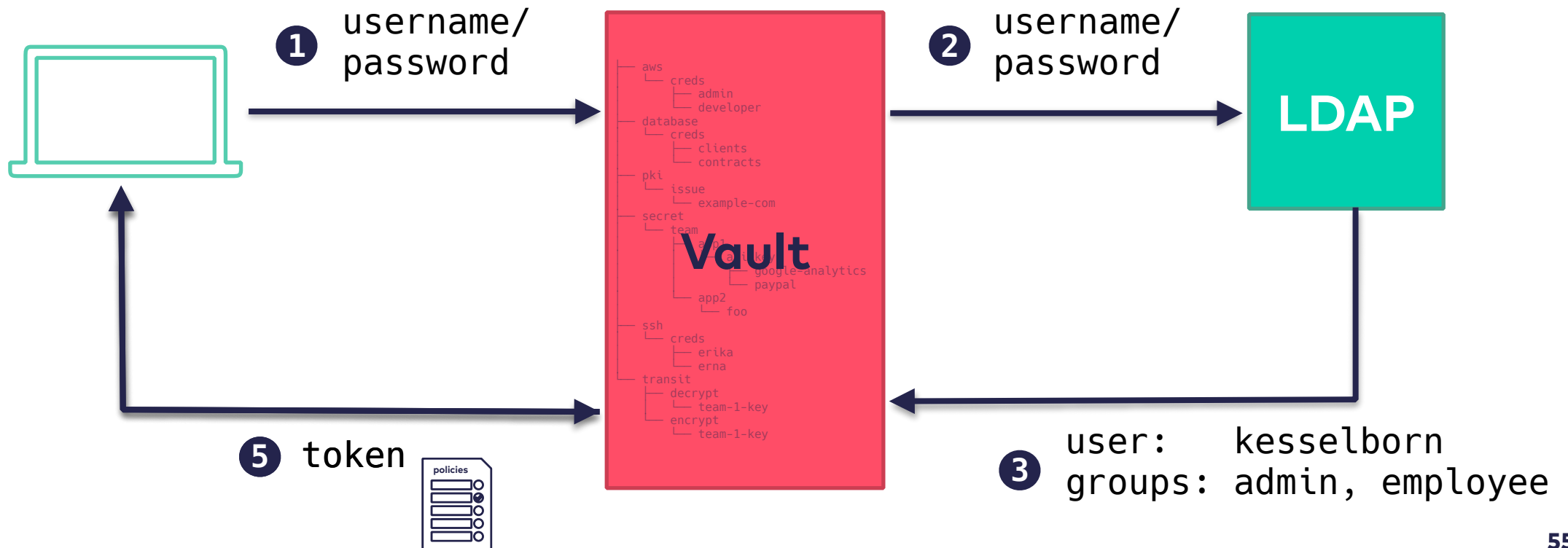
- authenticate with a TLS client certificate



# Vault auth backends — external identity providers

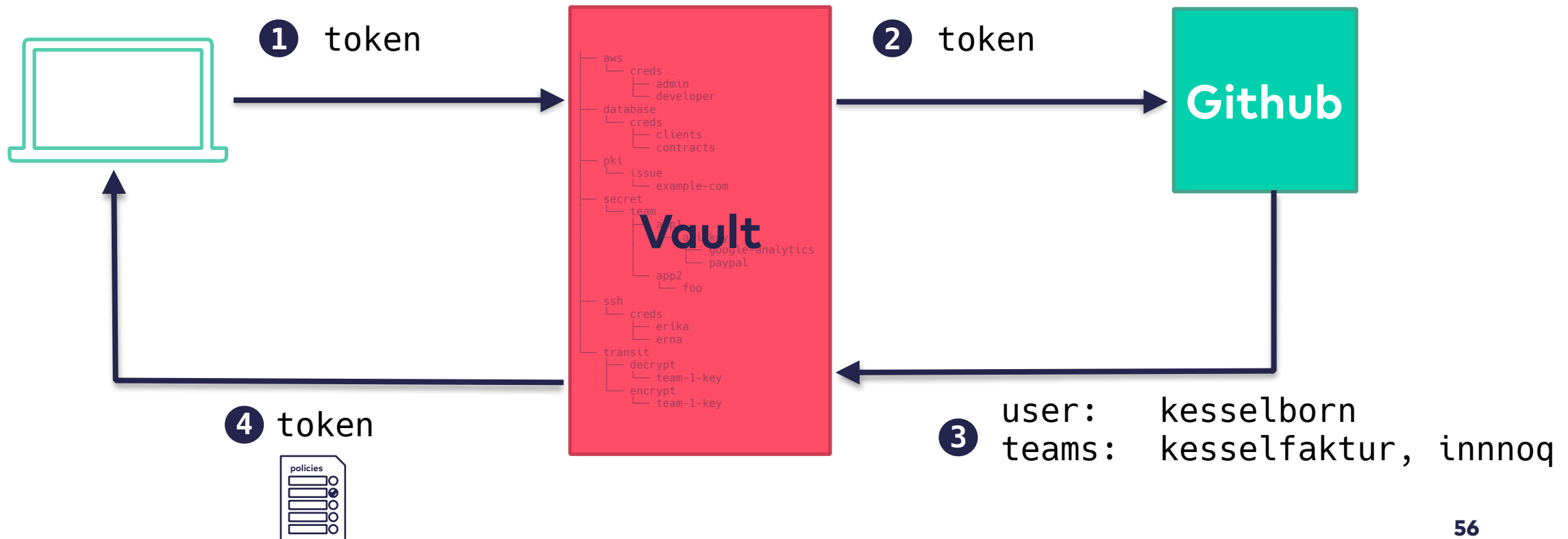
# Vault auth backends – LDAP / Radius / Okta auth

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



# Github auth

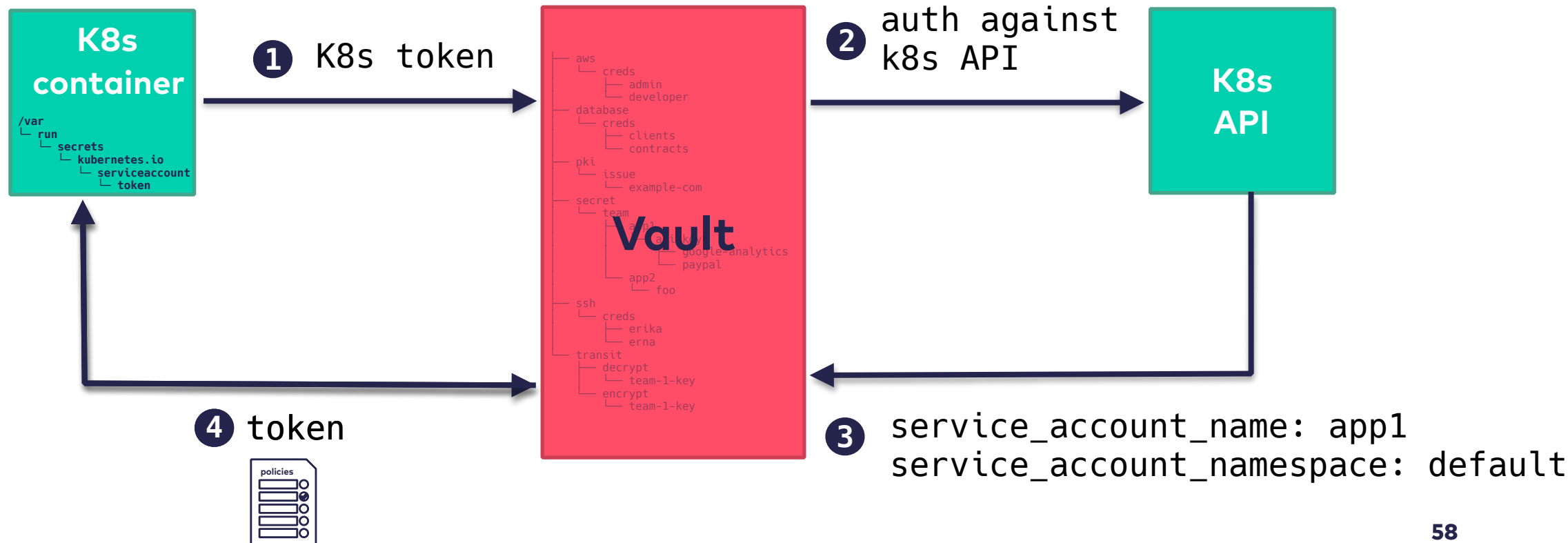
- Github is used as an identity provider



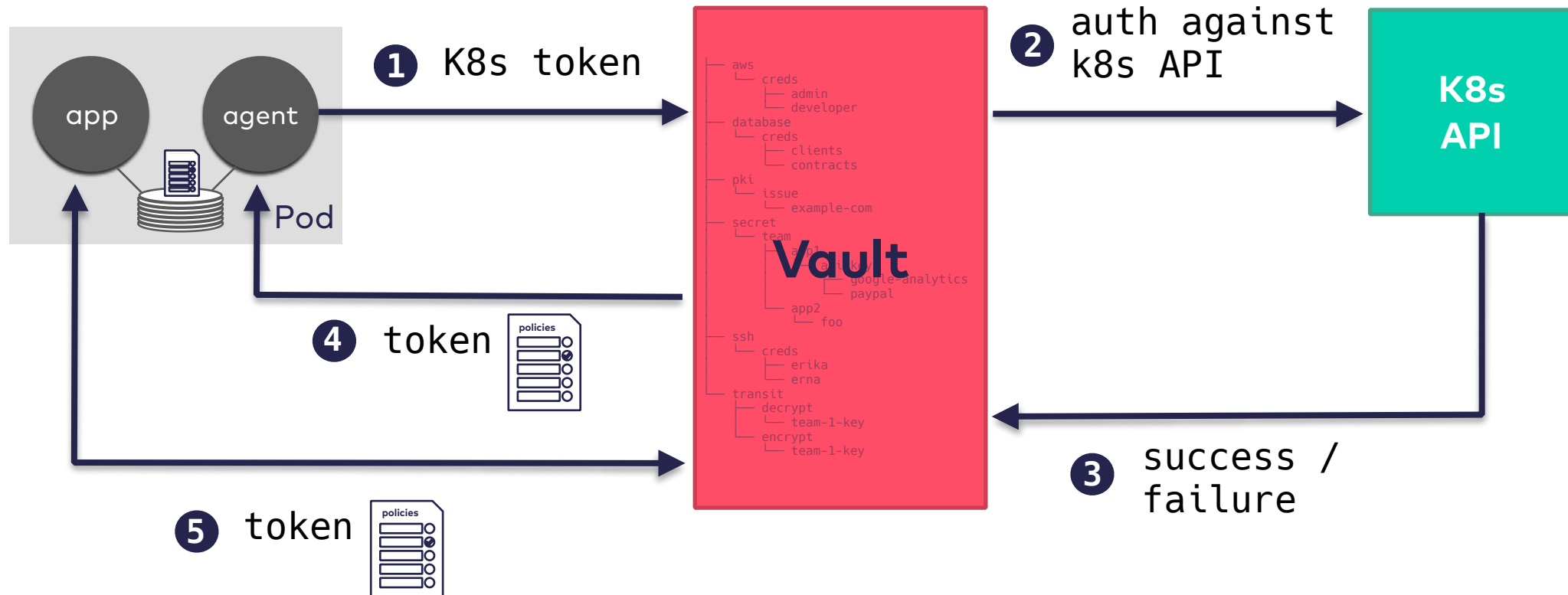


# Vault auth backends — Kubernetes auth

# Vault auth backends — Kubernetes auth



# Vault auth backends — Kubernetes auth



# Vault auth backends — Kubernetes auth

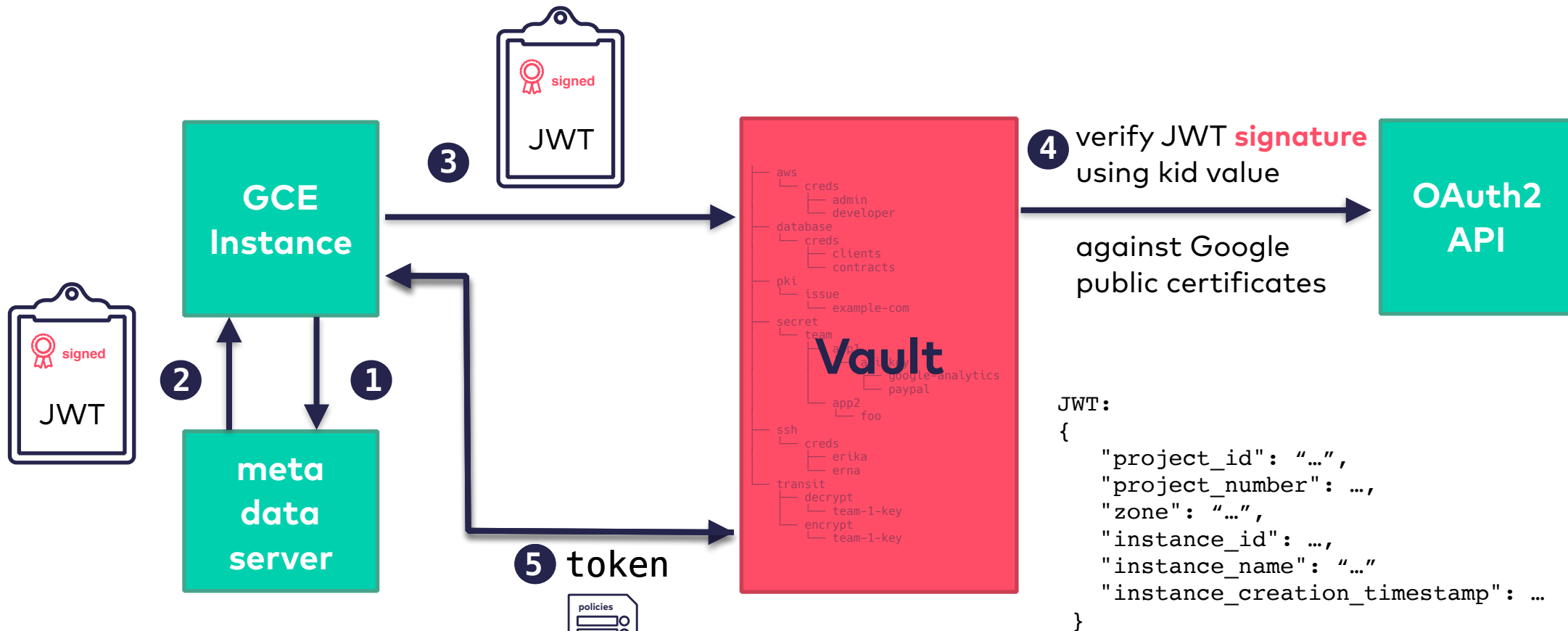
```
$ vault auth enable kubernetes
```

```
$ 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



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

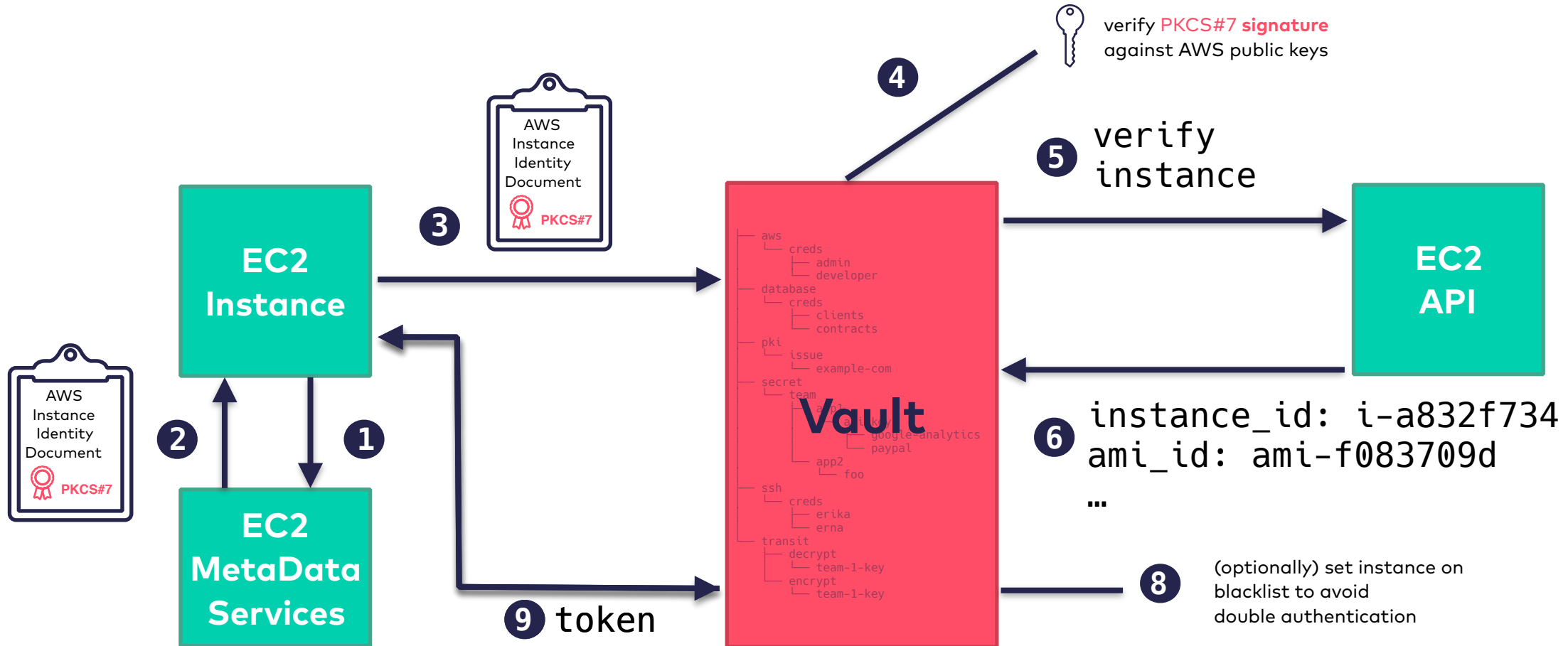
# Vault auth backends — AWS auth

# 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



```
curl -s http://169.254.169.254/latest/dynamic/instance-identity/pkcs7
```

# 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>

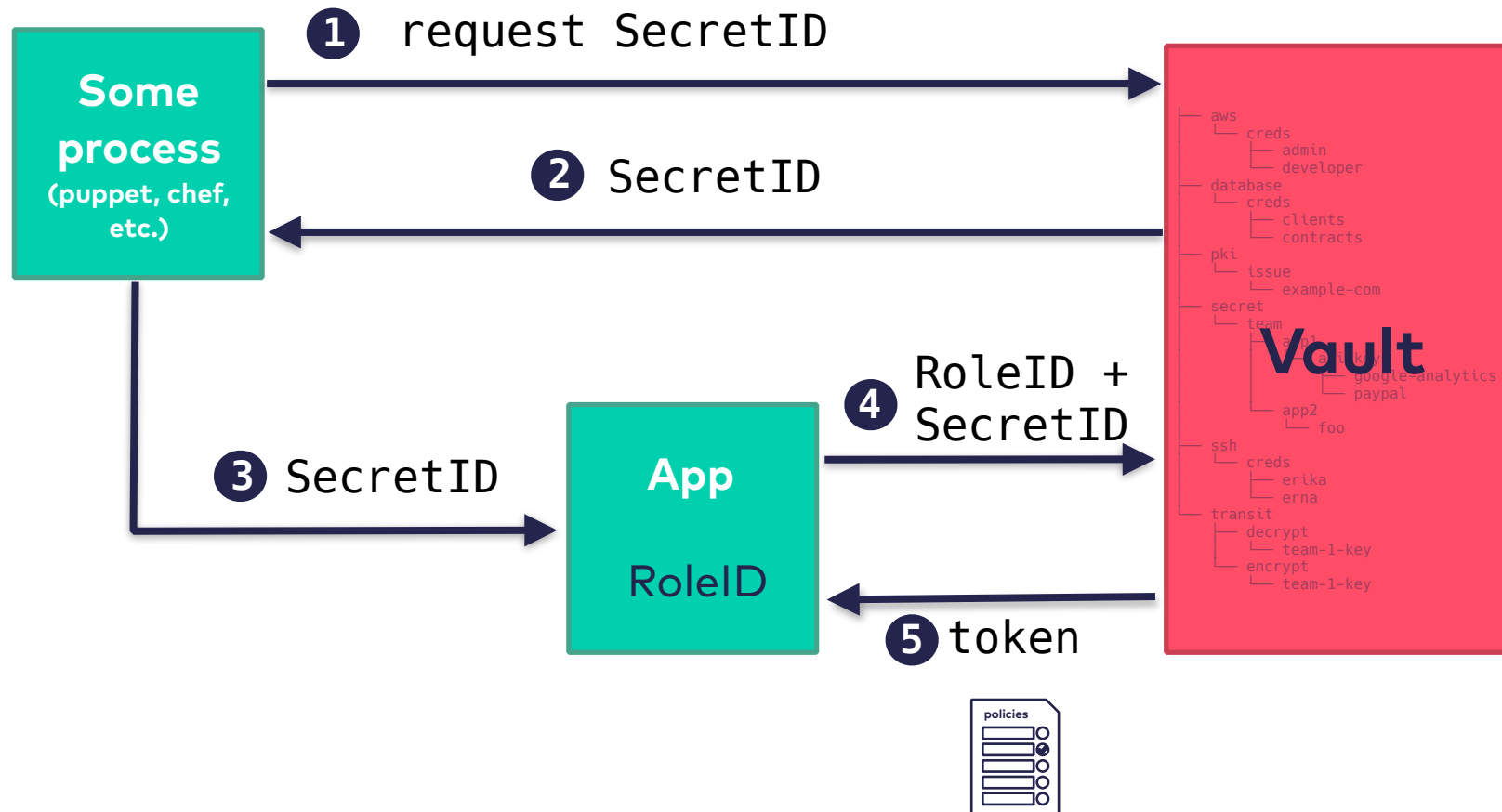
# Vault auth backends — AppRole

# 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

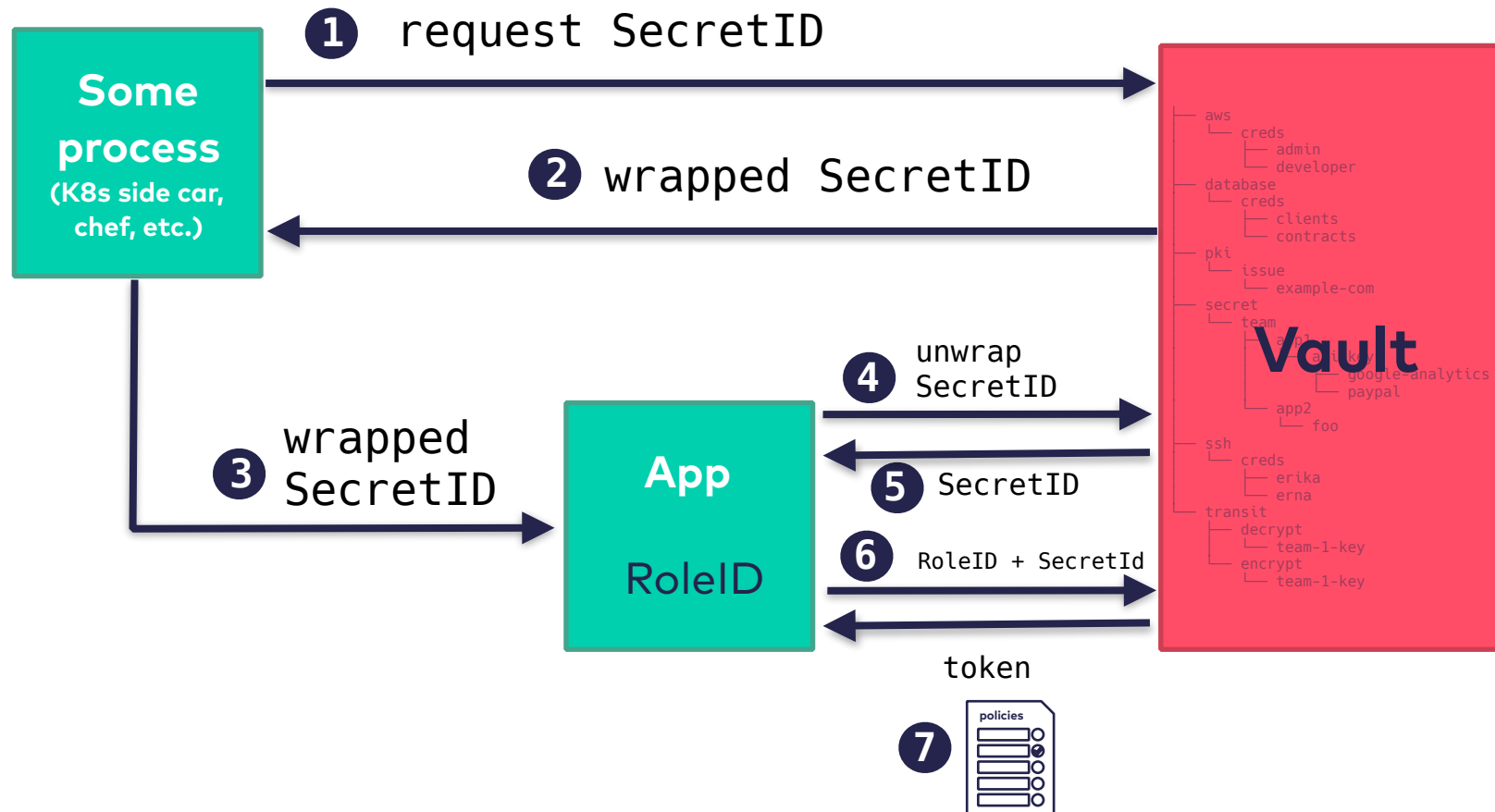
# Vault auth backends — AppRole

- mainly used for machines or apps to authenticate against Vault



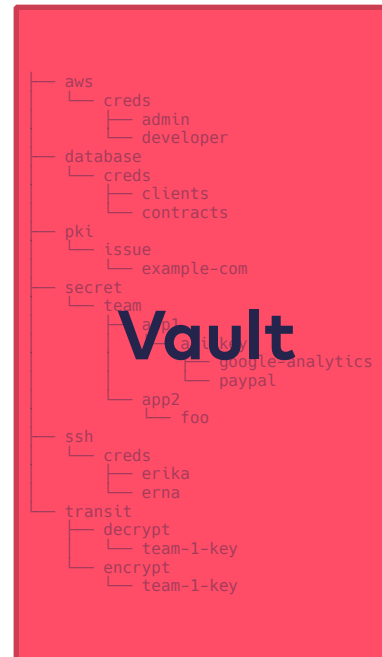
# Vault auth backends — AppRole

- 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



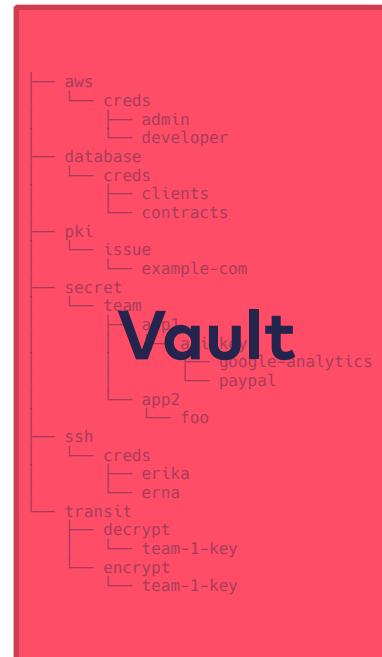
- AWS
- Consul
- Cubbyhole
- Databases
- Identity
- Static secrets (Key /Value)
- Nomad
- PKI (Certificates)
- RabbitMQ
- SSH
- TOTP
- Transit



# Use whatever the auth you want

# Vault

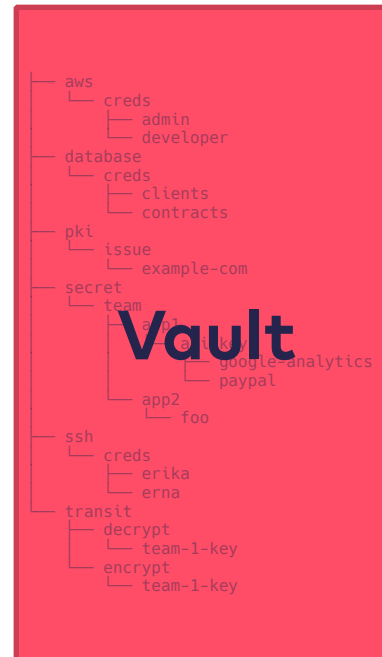
- Tokens
- **LDAP**
- AWS
- Kubernetes
- Google Cloud
- Username & Password
- AppRole
- GitHub
- MFA
- Okta
- RADIUS
- TLS Certificates



- **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

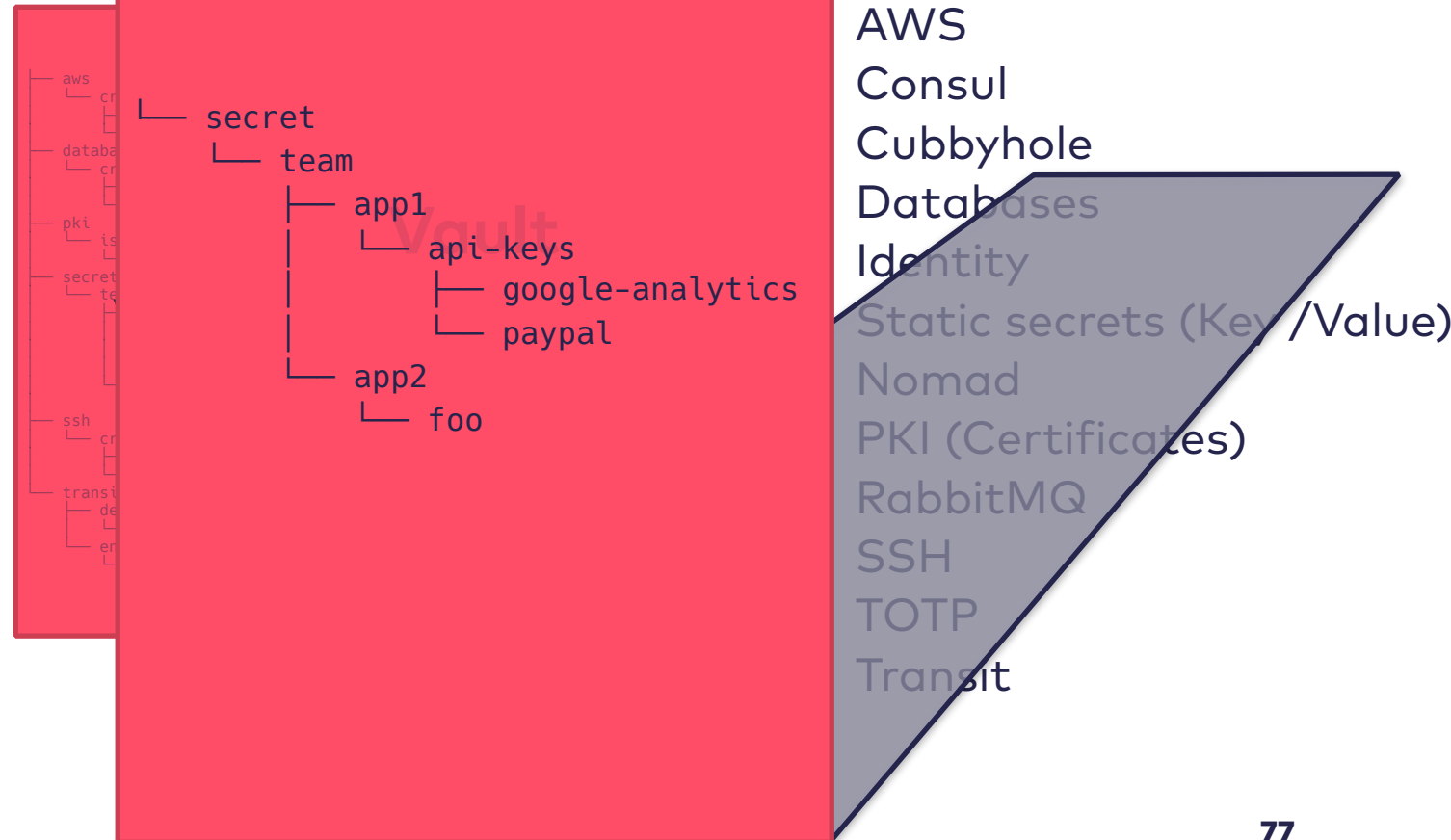


- AWS
- Consul
- Cubbyhole
- **Databases**
- Identity
- **Static secrets (Key /Value)**
- Nomad
- **PKI (Certificates)**
- RabbitMQ
- SSH
- TOTP
- **Transit**

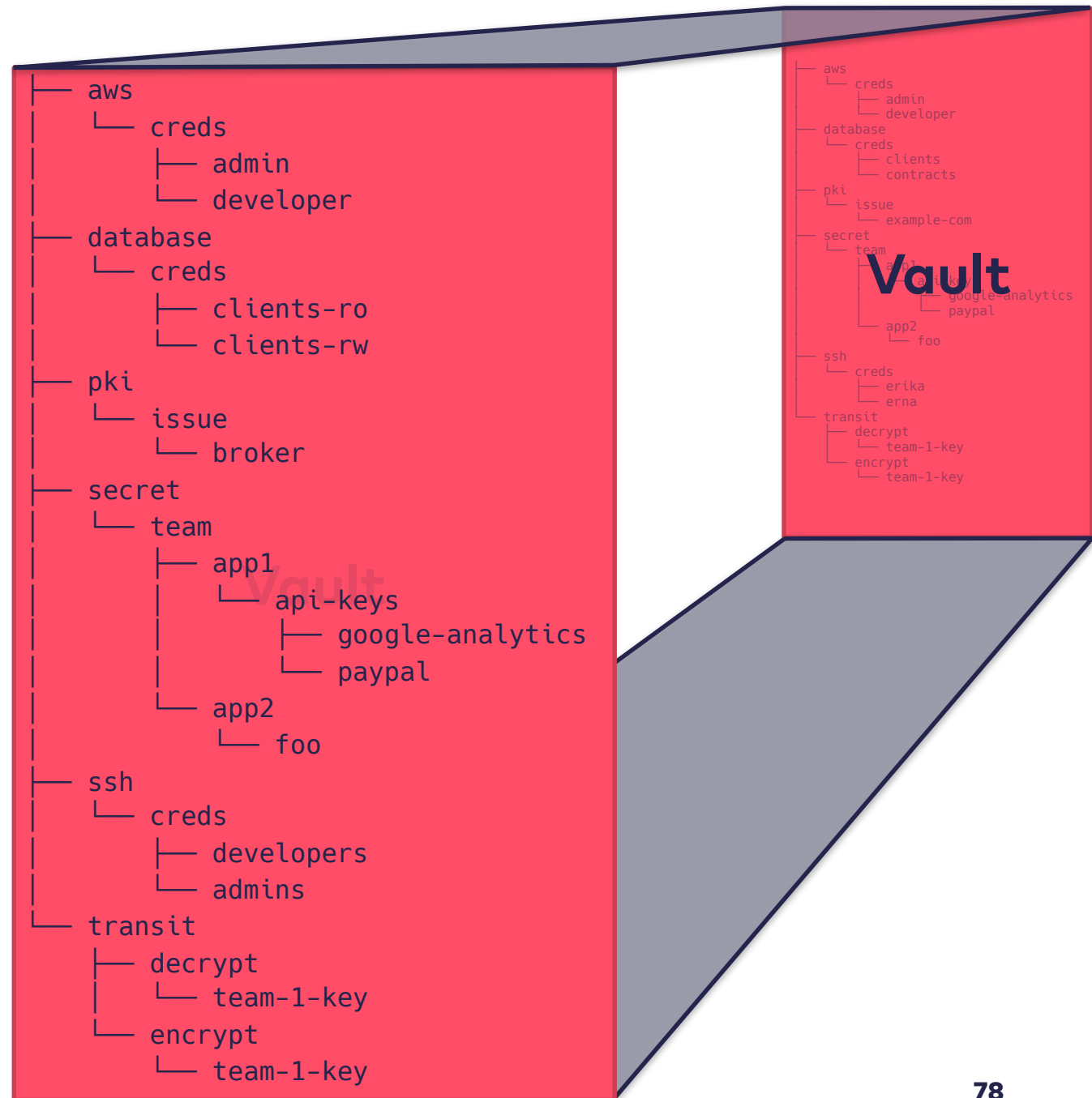
# Vault — policies

# Vault — secret representation

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

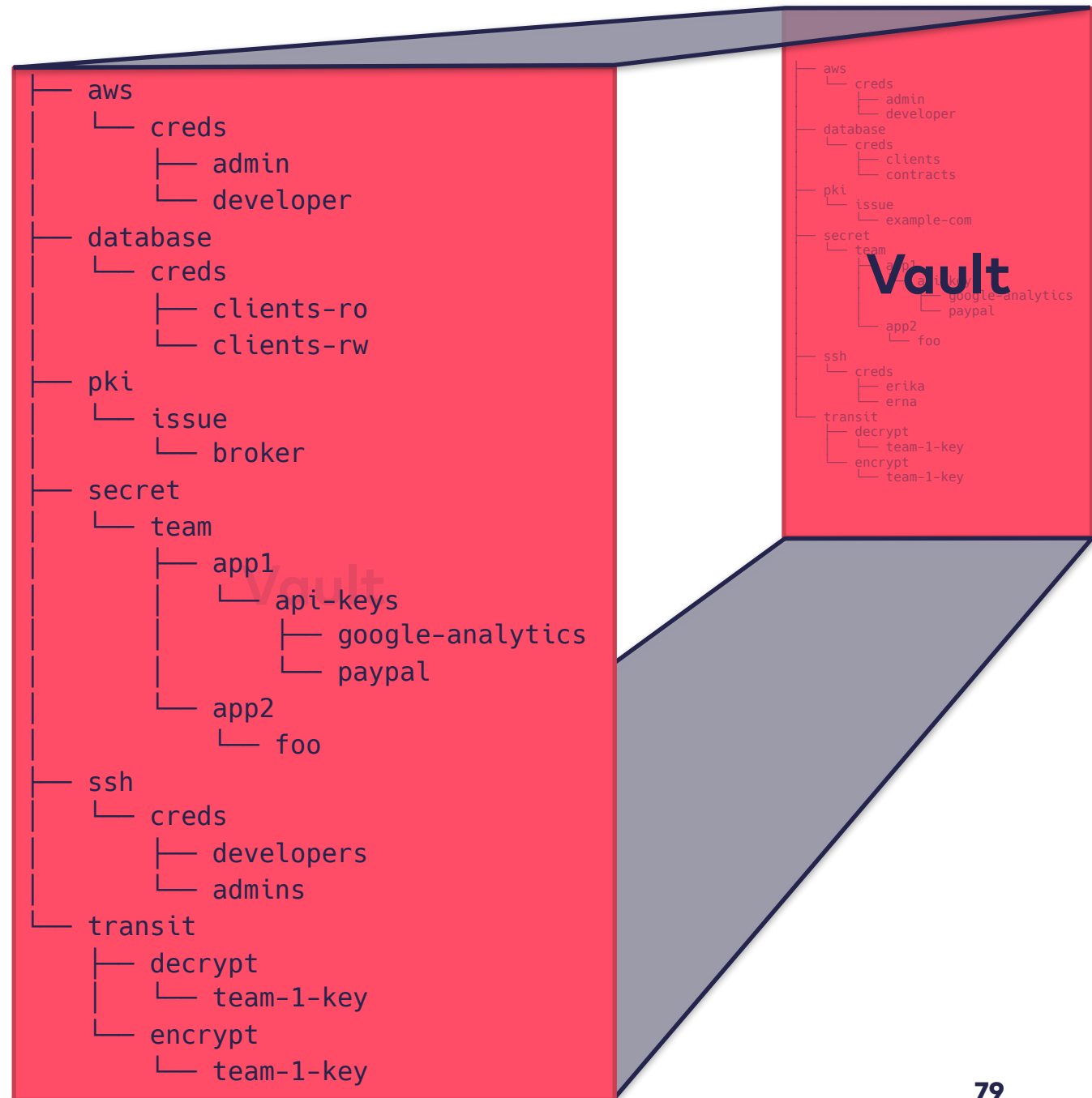


# Vault — secret representation



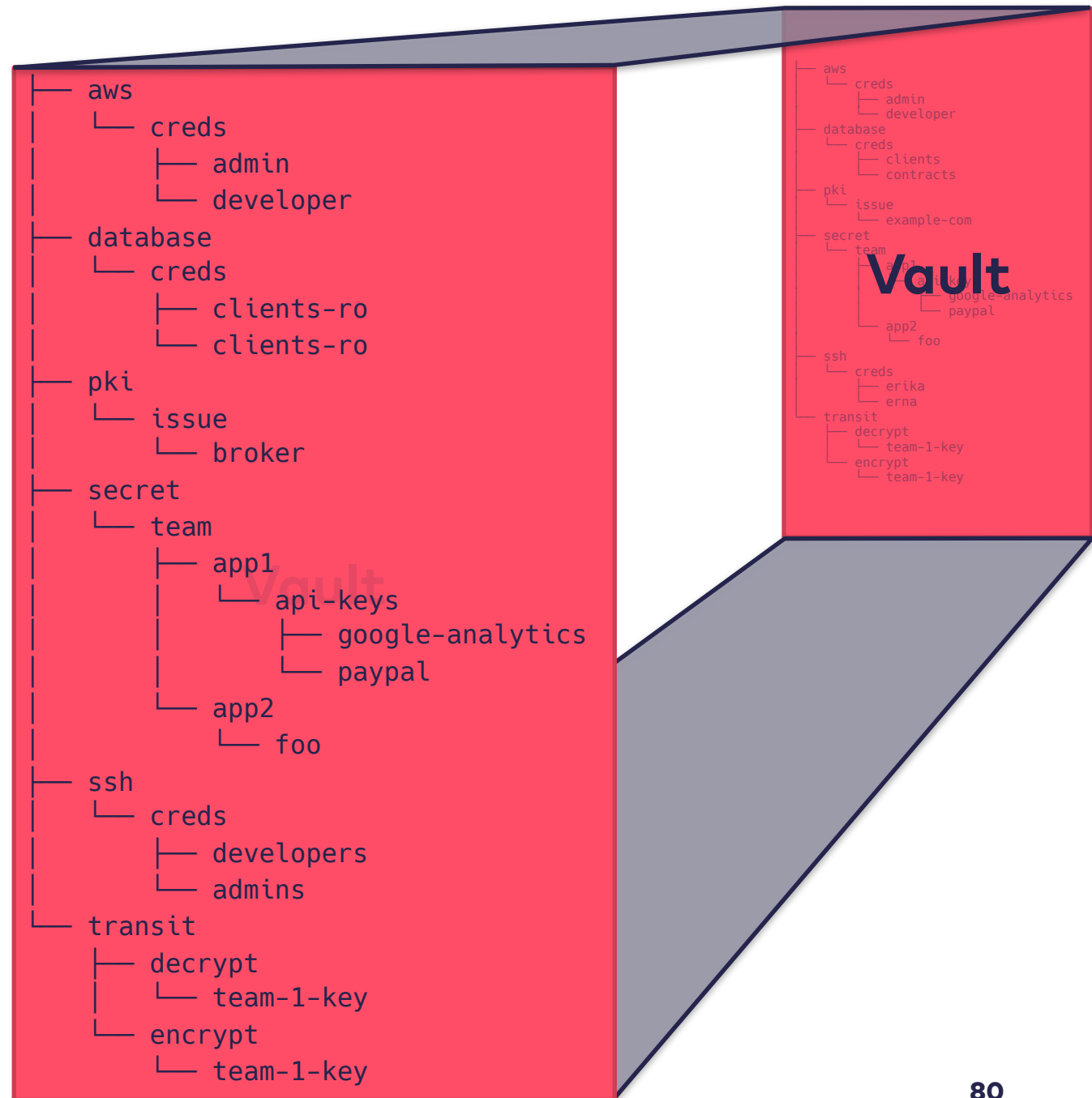
# Vault — policies

- 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





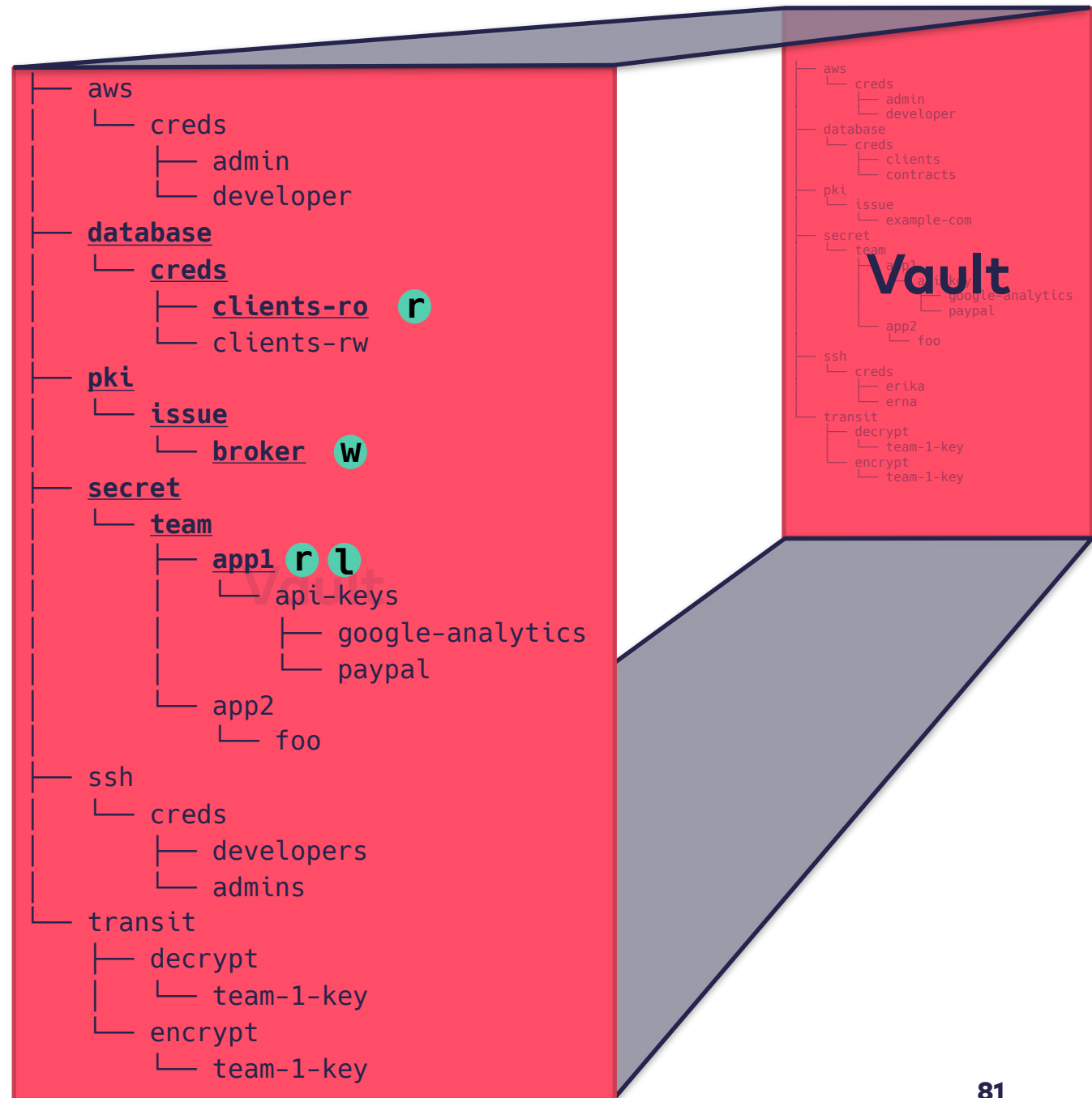
# Vault — policies

```
$ cat app1-policy.hcl
```

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

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

```
path "database/creds/clients-ro" {
  capabilities = ["read"]
}
```

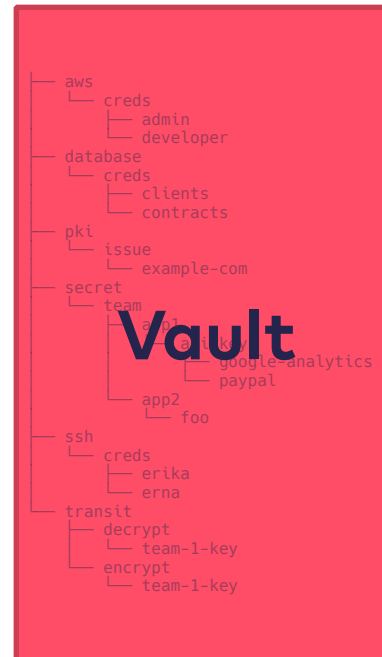


# Vault



policies

- 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 internals — storage

# Vault internals — storage

- several storage backends available
- data encrypted at rest with a symmetric key
- symmetric key is stored on storage backend and encrypted by "master key"
- master key is encrypted with "Shamir's Secret Sharing" algorithm

# 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

# Demo



# 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_na
me":"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_tok
en_accessor":"hmac-
sha256:27e400da69c94fce2378f5738cbf950531d7a9513215274abfbbdaa4927e00ba","path":"/sec
ret/","data":null,"policy_override":false,"remote_address":"100.96.0.76","wrap_ttl":
0,"headers":{},"response":{"data":{"error":"hmac-
sha256:d9d7a78363fd091f1b4c12629b7c9b5d7a7ffbf904ef5d29d002d5265d5bbf33"}}},"error":
"1 error occurred:\n\n* permission denied"}

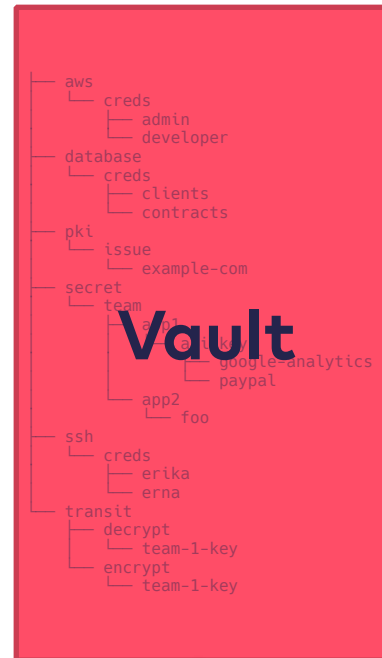
```

# Vault



policies

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



audit logs



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

# Vault internals

# 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

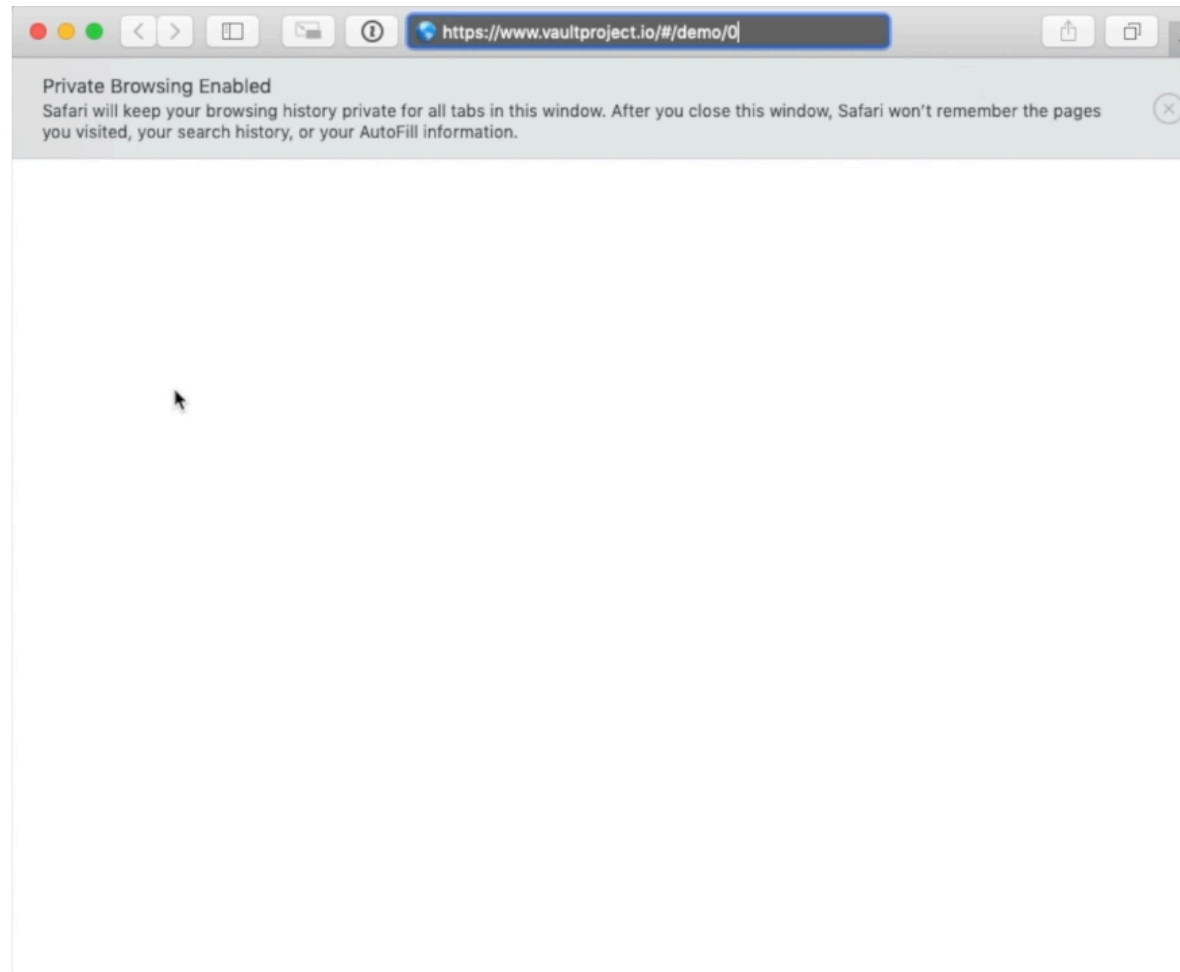
# Vault usage — integration

- some frameworks have integration for Vault
- when home made solution
  - create config files with "vault agent" 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>

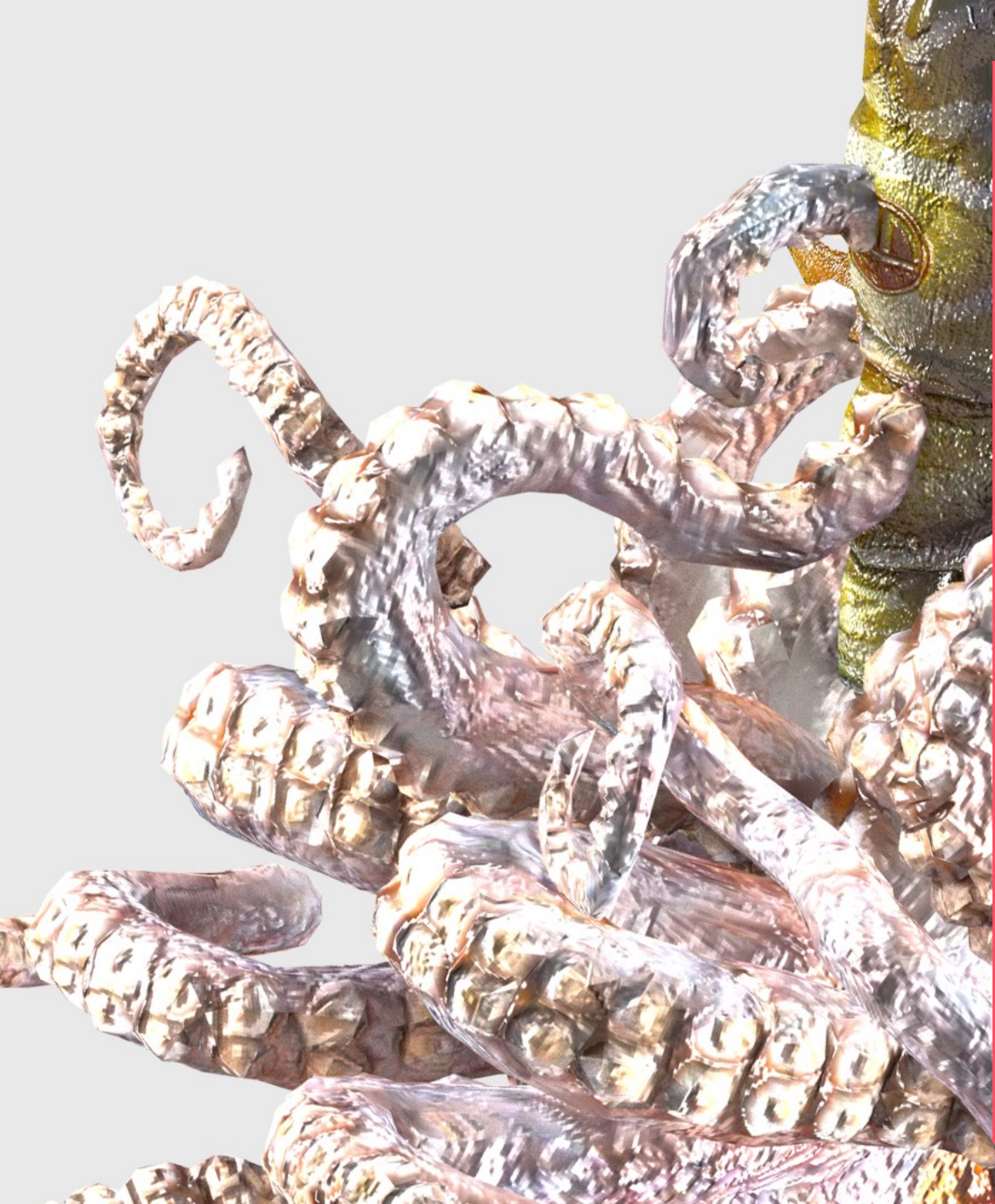


# Vault — getting started

- 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**

**Fragen?**

**INNOQ**