Vault in Action

Java User Group Hessen
27.02.2020

Dominik Schadow
bridgingIT

Secret handling in a typical project

Secrets stored in code or as environment variables

as Docker or Kubernetes secrets

No support for auditing secret access

rotating secrets (they live forever)

revoking secrets

No control about who has access to a secret and uses it

Secrets management ~~wishes~~ requirements

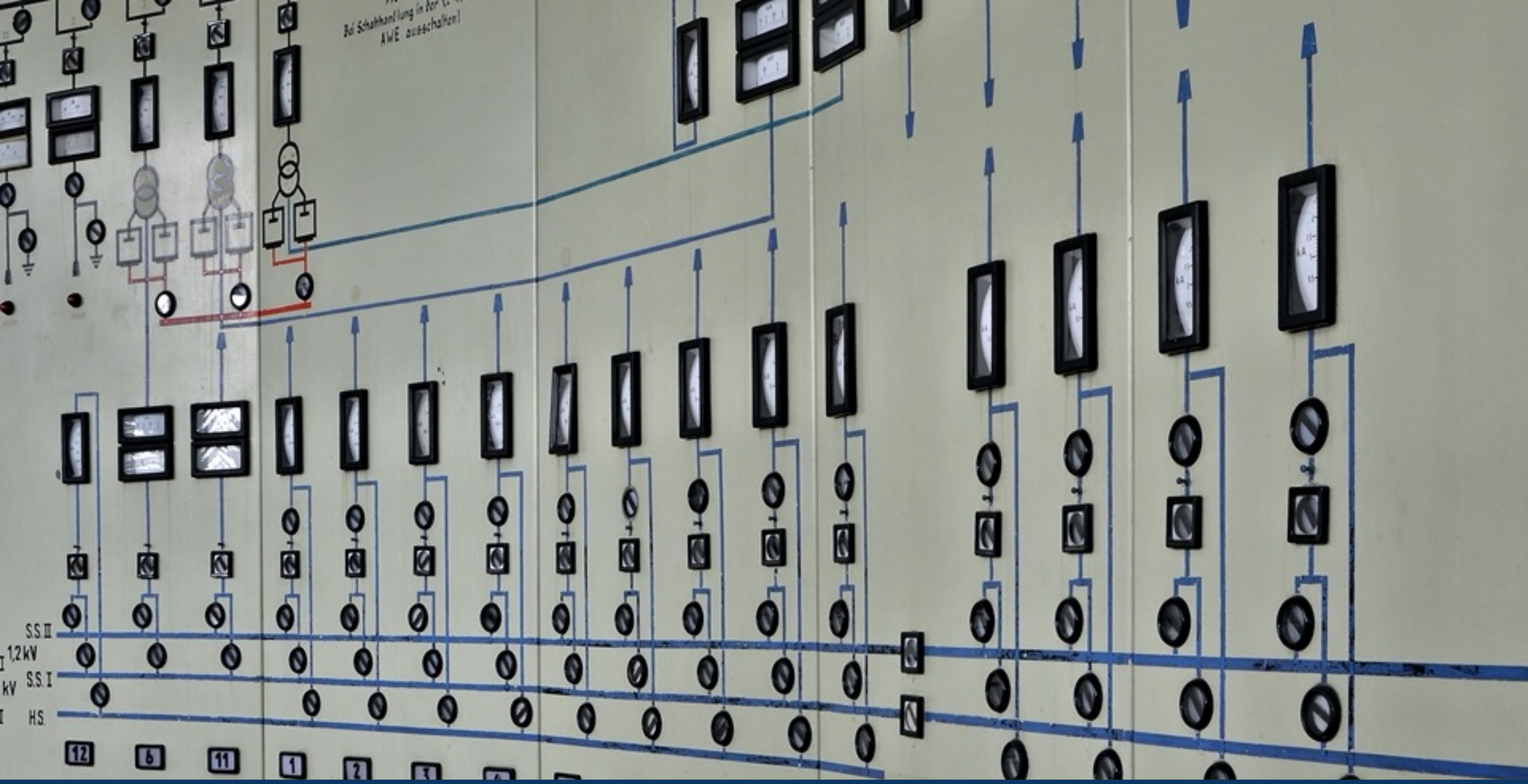
- Single central service
- All secrets are securely stored (encrypted)
- Apps and users can only access their secrets
- Secrets have a time-to-live (TTL)
- All interactions can be logged in an audit log

VAULT

A tool for managing secrets.

Who knows Vault?

Who is using Vault?



Vault is extensive - we'll focus on the dev parts

We assume secure operations, patches, HTTPS, high availability, and all other operational stuff…

Agenda

1. **Vault Basics** you need to know as a developer
2. **Authentication and Authorization** to access only your secrets
3. **Dynamic Secrets** for applications and users

Vault Basics

Vault in a nutshell

Open Source and **Enterprise editions**

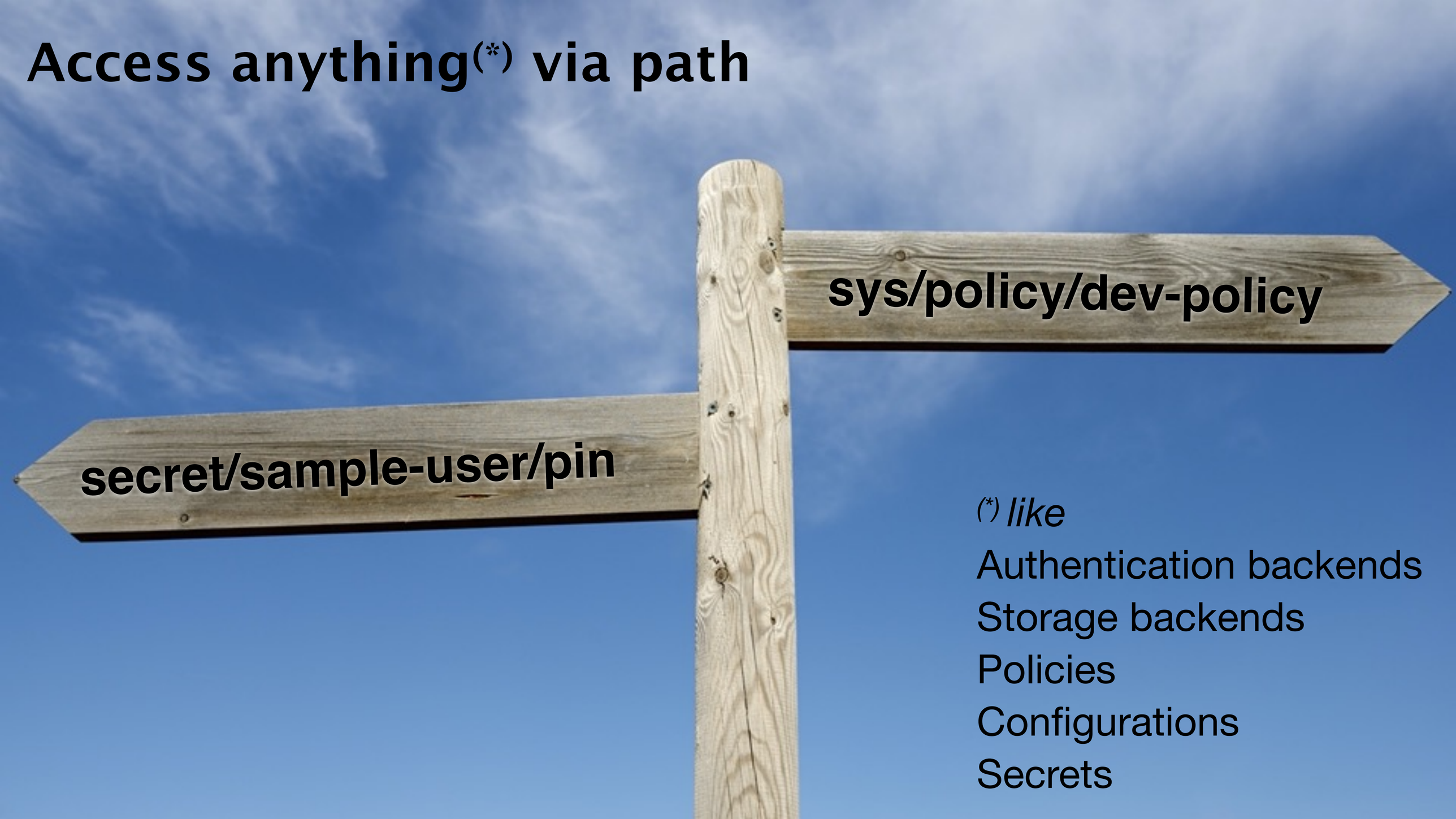
Central store for sensitive data and secrets

Generation, **storage**, and **distribution** of secrets

Detailed **audit logs for all secret interactions**

Provides **HTTP API and CLI**

Access anything(*) via path



(*) *like*

Authentication backends

Storage backends

Policies

Configurations

Secrets

Storage backends

Multiple storage backends Azure, Cassandra, CockroachDB, Consul, CouchDB, DynamoDB, Etcd, Filesystem, FoundationDB, Google Cloud Spanner, Google Cloud Storage, In-Memory, Manta, MSSQL, MySQL, OCI Object Storage, PostgreSQL, Raft, S3, Swift, Zookeeper

Support **static and dynamic secrets**

Data encrypted at rest with symmetric key

Vault is always started in sealed state

Based on **Shamir's Secret Sharing** algorithm

Unsealing (opening) requires **n keys** (persons)

Unsealing **provides master key** (splitted into shards) to Vault, **reconstructs encryption key**

Auto unseal with trusted device or service

Delegates responsibility of securing the master key

Multiple providers AliCloud KMS, AWS KMS, Azure Key Vault, GCP Cloud KMS, OCI KMS, Vault Transit

Some operations require manual unseal (like generating a root token)

Enough with operations,
lets move on to
development

Authentication and Authorization

Authentication

Before a client or user can interact with Vault, it **must authenticate against an activated authentication backend**

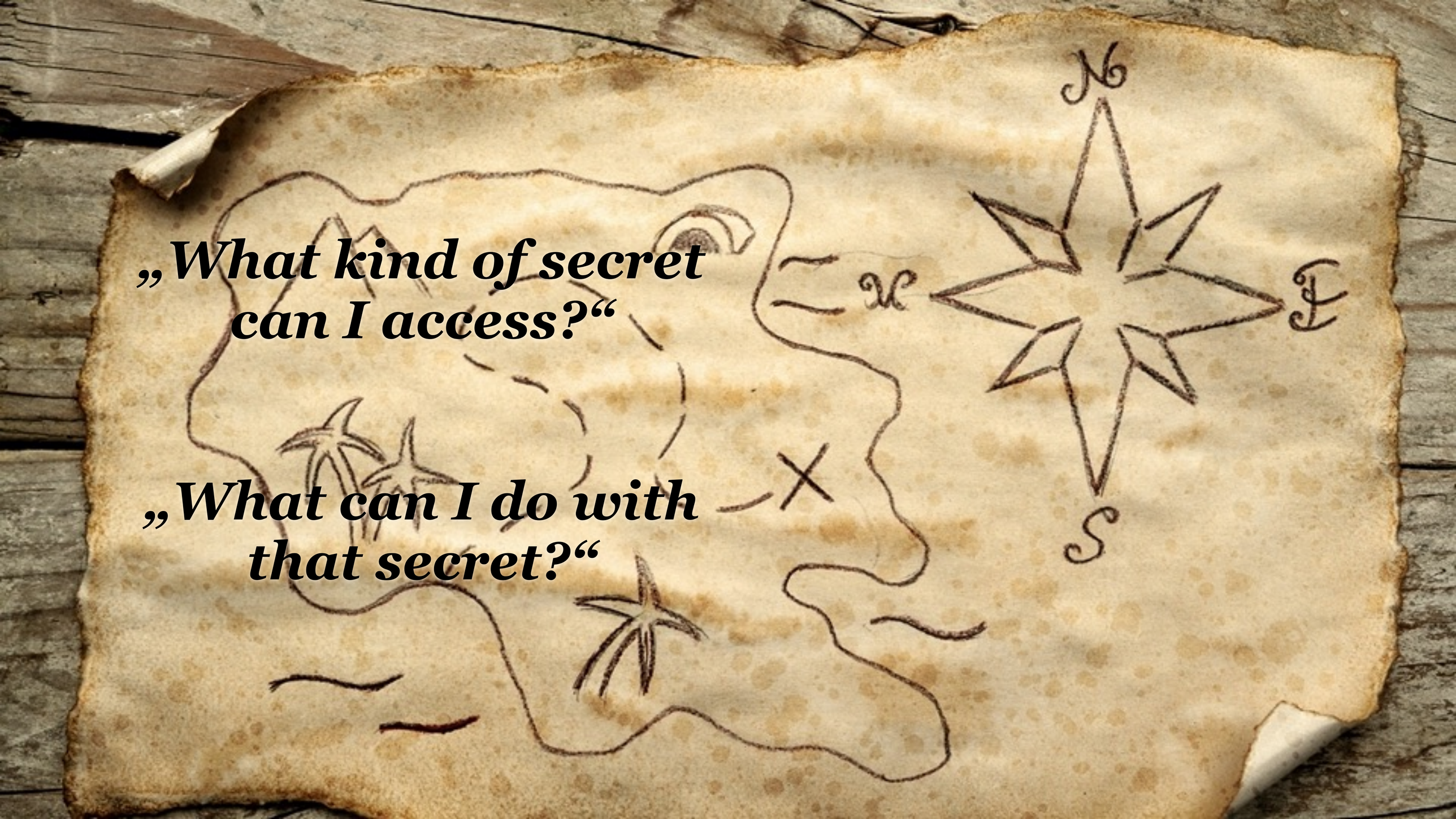
After successful authentication, **a token is returned to the client or user**

Authentication methods

Authentication is provided by **pluggable backends**

Various auth methods for apps and users AppRole, AliCloud, AWS, Azure, Cloud Foundry, Google Cloud, JWT/OIDC, Kubernetes, GitHub, LDAP, Oracle Cloud Infrastructure, Okta, RADIUS, TLS Certificates, Tokens, Username & Password

Only authorized applications, policies restrict access


„What kind of secret can I access?“
„What can I do with that secret?“

Policies

Vault uses policies **to manage and safeguard access**

Declarative way to deny (default) or grant access to operations and paths

create, read, update, delete, list, sudo, deny

Usually written in **HashiCorp Configuration Language (HCL)**

```
# config-server-policy.hcl
path "kv-v2/*" {
  capabilities = ["read"]
}

path "kv-v2/my-secrets/*" {
  capabilities = ["create", "read",
"update", "delete", "list"]
}

path "transit/*" {
  capabilities = ["read", "update"]
}
```

```
|-- database
|-- kv-v2
    |-- config-client-vault
    |-- my-secrets
|-- transit
    |-- config-client-vault-key
    |-- another-key
```

```
# create/ upload config-server-policy
vault policy write config-server-policy \
  config-server-policy.hcl


# create/ upload config-client-policy
vault policy write config-client-policy \
  config-client-policy.hcl
```

Token authentication

Default authentication method (and easiest one)

With **disclosed token**, everybody can **gain access, so never expose a root token in production**

```
# create a token for config-client-vault
vault token create -policy=config-client-policy
```

```
# create a token for config-client-server
vault token create -policy=config-server-policy
```

```
spring.cloud.vault:
   authentication: TOKEN
   token: s.39SL8SdQsr5Hq7nqLc6Mb76d
```

AppRole authentication

Intended for **machine or apps authentication**

Two hard to guess (secret) tokens

RoleId

SecretId (optional, required by default) - treat as password
With secret_id_num_uses, secret id can be forced to be regenerated after a number of uses
Typically provided as **environment variable** SPRING_CLOUD_VAULT_APP_ROLE_SECRET_ID

```
# create config-server role
vault write auth/approle/role/config-server \
    token_ttl=1h \
    token_max_ttl=4h \
    token_policies=config-server-policy


# create config-client role
vault write auth/approle/role/config-client \
    token_ttl=1h \
    token_max_ttl=4h \
    token_policies=config-client-policy
```

```
# update config with returned role-id
vault read auth/approle/role/config-server/role-id
```

```
# update config with returned secret-id
vault write -f auth/approle/role/config-server/secret-id
```

```
spring.cloud.vault:
   authentication: APPROLE
   app-role:
     role-id: 07c2ca09-1c50-a0f2-c1ef-8a9e1130089c
     secret-id: 713f1d28-5f06-f0f6-85de-ee2dbea0a63d
```

Demo

Dynamic Secrets

Classical static secrets

Created once (and often forever)

Static for all applications (each application)

Securely stored in Vault, loaded in application during start up

Dynamic secrets

Created just in time (on-demand)

Unique per client (app or user)

Auto expiring (time-to-live)

Database roles

Role configuration controls the tables a use has access and the lifecycle of the credentials

Different roles per connection can exist (e.g. read-only, write, …)

Vault **runs the given SQL statement to create** the role

When TTL expires, Vault **runs the given SQL statement to revoke** the role

```
# enable dynamic database secrets
vault secrets enable database


# create an all privileges role
vault write database/roles/config-client-vault-write \
  db_name=config-client-vault \
  creation_statements="CREATE ROLE \"{{name}}\" \
    WITH LOGIN PASSWORD '{{password}}' VALID UNTIL \
    '{{expiration}}'; \
    GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public \
    TO \"{{name}}\";" \
  revocation_statements="ALTER ROLE \"{{name}}\" NOLOGIN;"\
  default_ttl="1h" \
  max_ttl="24h"
```

Database connections

A connection **manages the root access for a database**

Connection in Vault is the configuration to **connect to and authenticate with each database**

Parameter

plugin_name configures which database plugin to use

allowed_roles defines which roles can use this connection

connection_url is a standard connection string to access the database

Initial root password

Connection string uses template variables to enable Vault's **root credential rotation** feature (Vault automatically rotates the root credentials for the database)

Vault saves the password but you cannot retrieve it

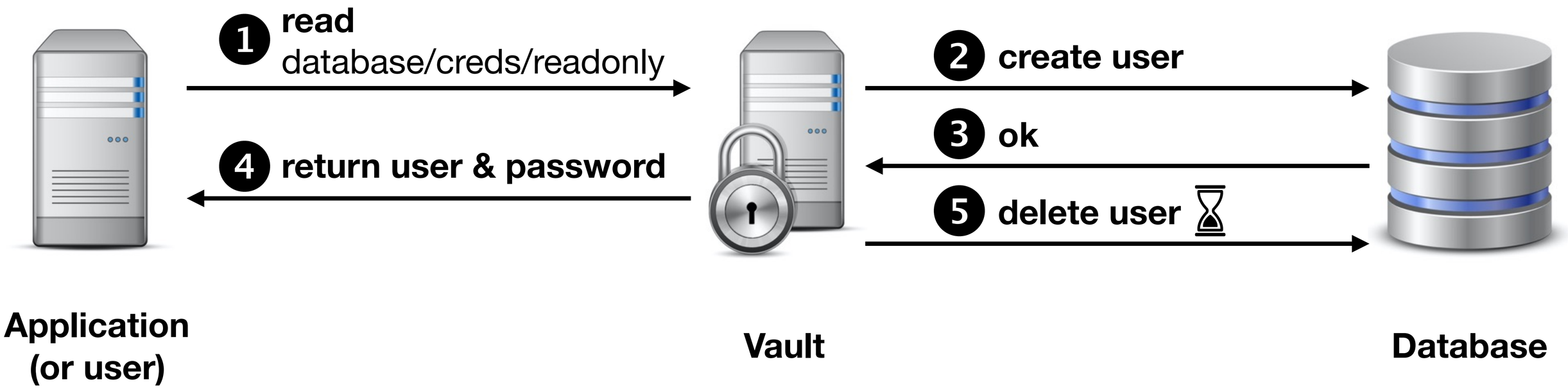
```
# create the database connection
vault write database/config/config-client-vault \
  plugin_name=postgresql-database-plugin \
  allowed_roles="*" \
  connection_url="postgresql://{{username}}:{{password}} \
  @postgres:5432/config-client-vault?sslmode=disable" \
  username="postgres" \
  password="password"
```

command only works with running PostgreSQL and existing database

```
# force rotation for root user
vault write --force /database/rotate-root/config-client-vault
```

careful, make sure you have another role with root permissions left

```
# create new credentials
vault read database/creds/config-client-vault-write
```


1 read
database/creds/readonly
2 create user
3 ok
4 return user & password
5 delete user
Application
(or user)
Vault
Database

Vault managed credentials

Unique credentials - easy auditing

Vault manages the lifecycle of credentials (rotating and revoking as required)

Vault requires **root credentials for the database** to create credentials on demand

Supports various databases (Spring supports them all)
Cassandra, Elasticsearch, Influxdb, HanaDB, MongoDB, MSSQL, MySQL/ MariaDB PostgreSQL, Oracle

```
# extend the policy to retrieve dynamic credentials
path "database/creds/config-client-vault-write" {
  capabilities = ["read"]
}
```

```
# enable dynamic database credentials in bootstrap.yml
spring.cloud.vault:
  database:
    enabled: true
    role: config-client-vault-write
```

```
# credentials are obtained via Vault
spring:
 datasource:
  url: jdbc:postgresql://localhost:5432/config-client-vault
```

```
# add the spring-cloud-vault-config-databases dependency
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-vault-config-databases</artifactId>
  <version>2.2.1.RELEASE</version>
</dependency>
```

Spring Cloud Vault does not support getting new credentials and configuring your `DataSource` with them when the maximum lease time has been reached. That is, if `max_ttl` of the Database role in Vault is set to `24h` that means that 24 hours after your application has started it can no longer authenticate with the database.

https://bit.ly/2vctdBE

Basic solutions

Configure the max time-to-live - possible when frequently redeploying the application
Vault: The system max TTL, which is 32 days but can be changed in Vault's configuration file **- be careful, not for production**

Use a LeaseListener to restart the application when credentials are rotated

https://bit.ly/2PuvIpB

Solution for relational databases

Renew the database credentials at runtime - supports only relational databases and requires Spring Boot with a HikariCP

1. Detect when database credentials are expiring
2. Get new dynamic database credentials from Vault
3. Refresh database connection to use new credentials

https://bit.ly/2Vq8yVf

Demo

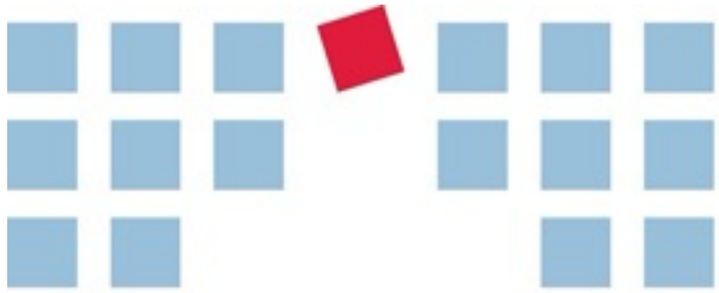
Vault is extensive, we have just touched the surface

Summary

Vault provides tons of features for **secret management**

Get rid of (static) secrets in application code and move on to **(dynamic) secrets in Vault**

Remember Vault security, it’s the **central location with all your secrets**



bridging IT

Marienstr. 17
70178 Stuttgart

dominik.schadow@bridging-it.de
www.bridging-it.de

Blog blog.dominikschadow.de
Twitter @dschadow

Demo Project
https://github.com/dschadow/CloudSecurity
Spring Cloud Vault Reference
https://cloud.spring.io/spring-cloud-vault/reference/html/
Vault
https://www.vaultproject.io

Pictures
https://www.dreamstime.com