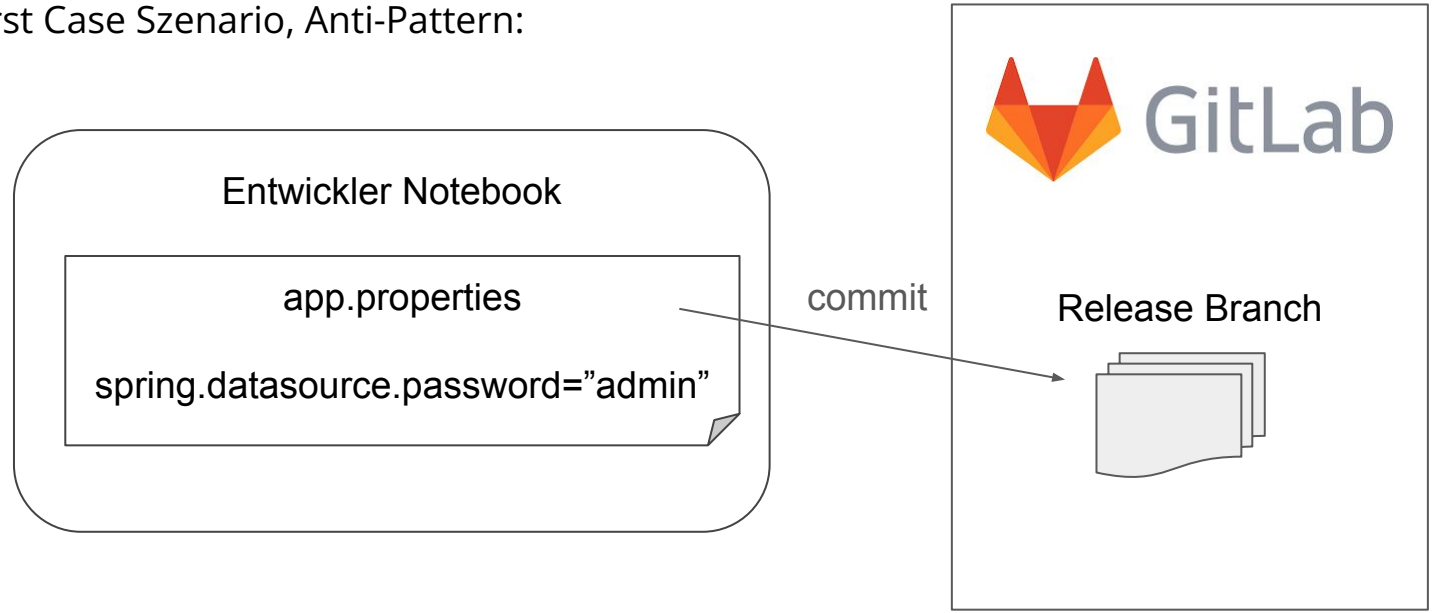
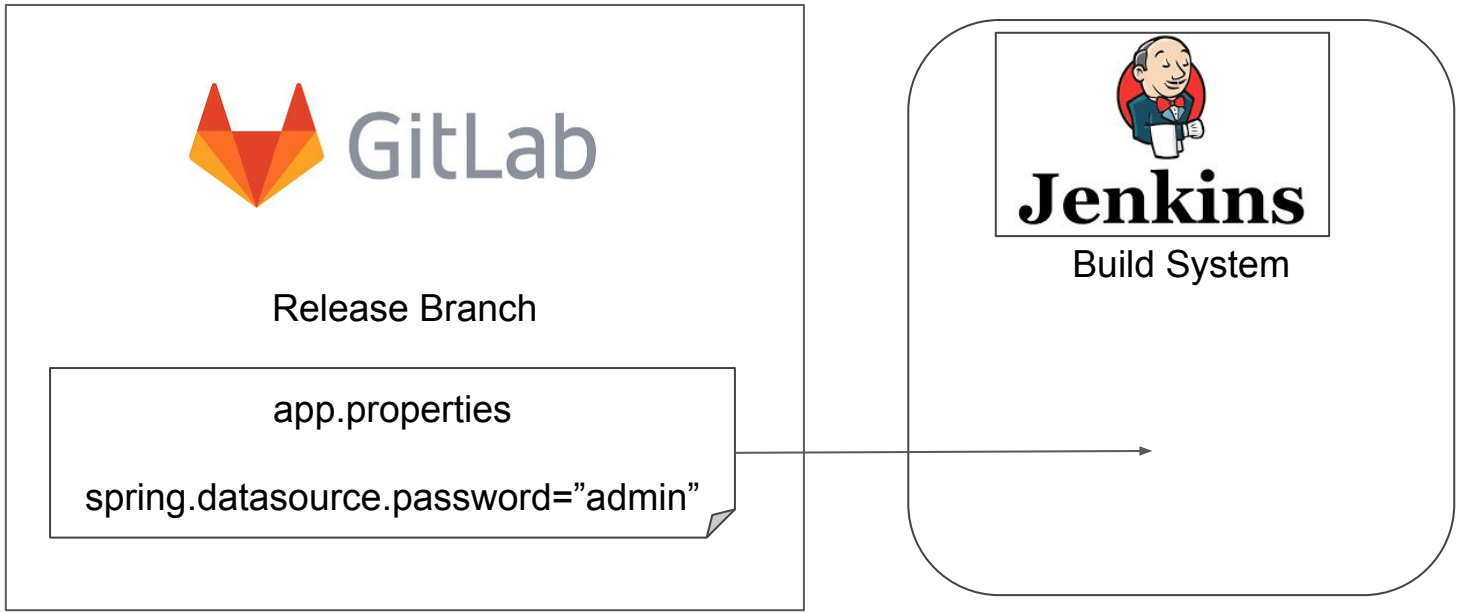
Credentials im Deployment Prozess

Worst Case Szenario, Anti-Pattern:



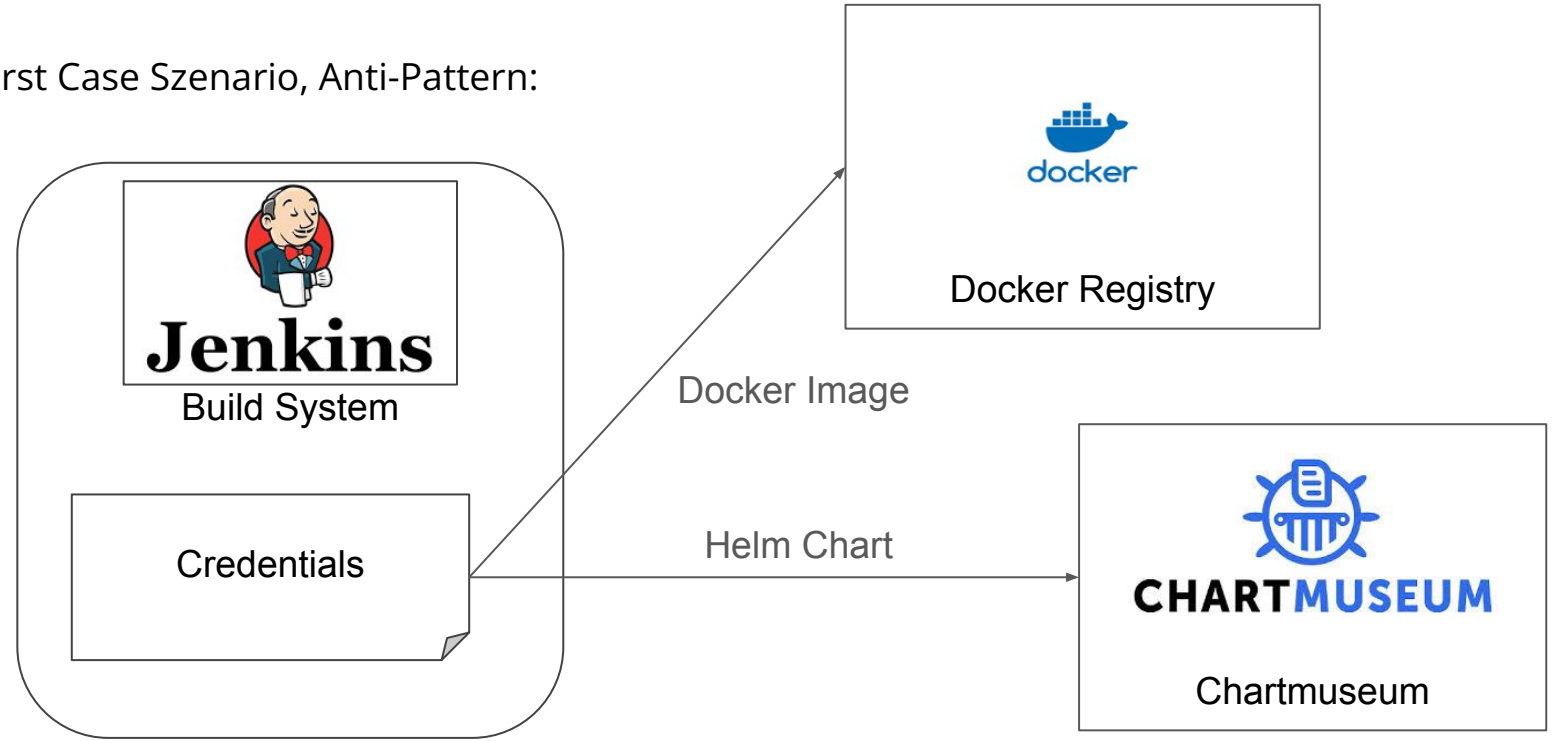
Credentials im Deployment Prozess

Worst Case Szenario, Anti-Pattern:



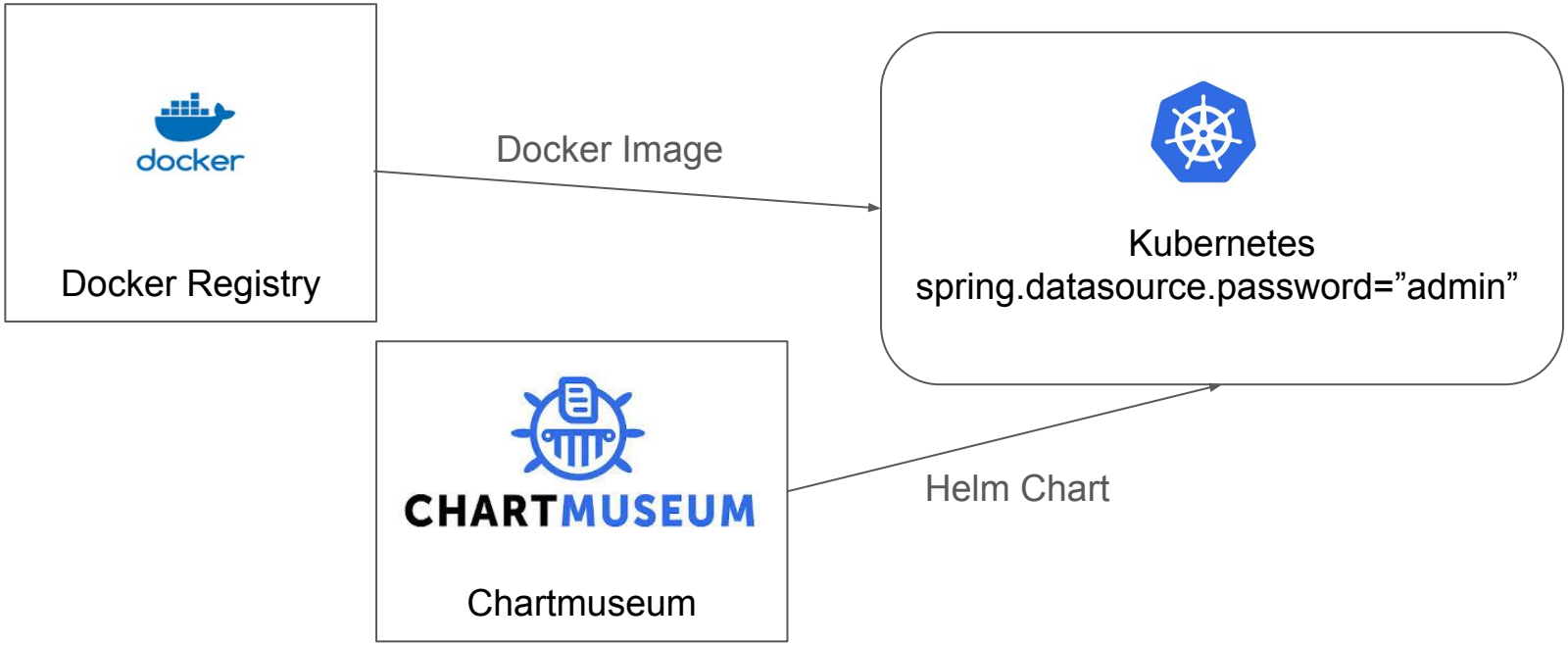
Credentials im Deployment Prozess

Worst Case Szenario, Anti-Pattern:



Credentials im Deployment Prozess

Worst Case Szenario, Anti-Pattern:



Credentials im Deployment Prozess

Worst Case Szenario, wo liegen meine Credentials:

GIT Repository

Buildsystem

Registry

Chartmuseum

Kubernetes

Cache / Build Cache

Backup System

Entwickler Notebook

Kubernetes Secrets

- Mit Kubernetes Secrets werden Credentials vom Deployment Prozess getrennt
- Kubernetes Secrets dienen zum Speichern von:
  - API Keys
  - Passwörter
  - Token
  - SSH Keys
  - alle Daten, die einer Zugriffsbeschränkung unterliegen

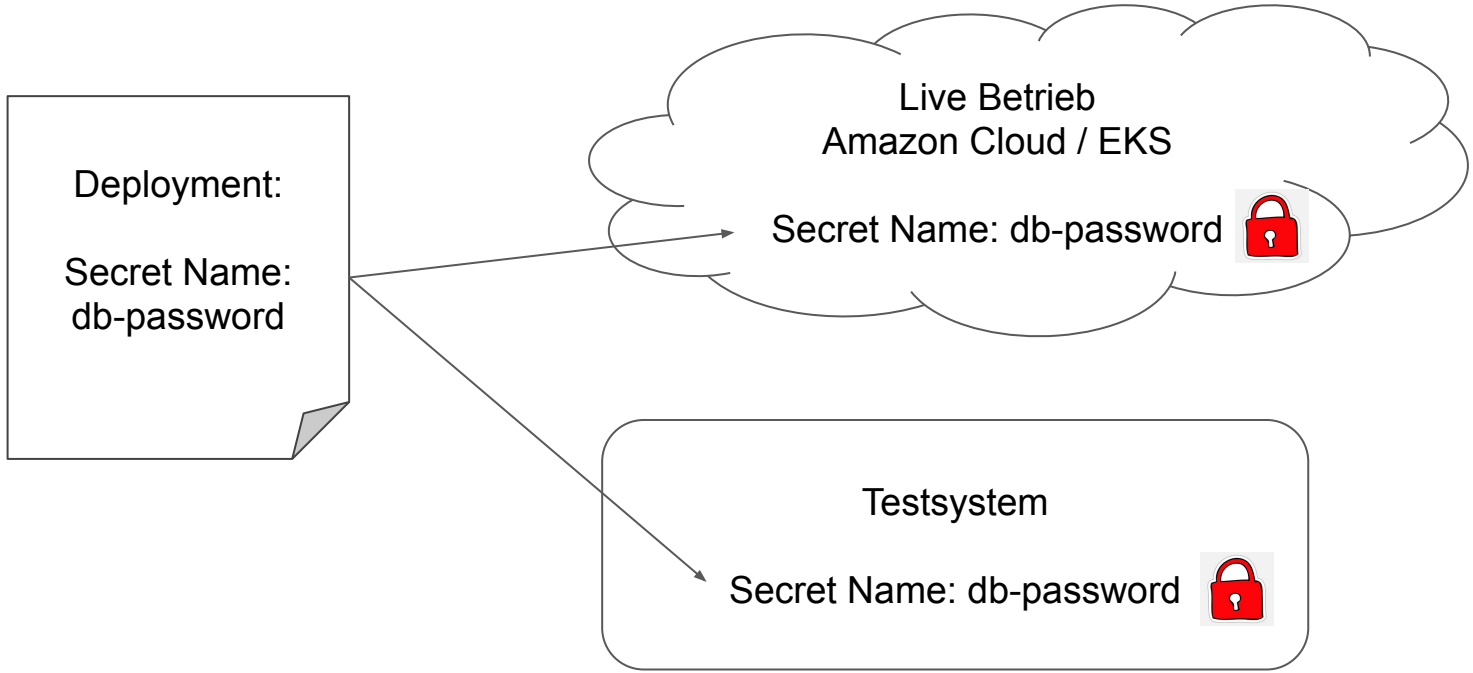
Kubernetes Secrets

Kubernetes Secrets werden über ihren Namen referenziert

- Credentials sind im Deployment nicht mehr sichtbar
- Credentials werden auf der Zielumgebung verwaltet
- Credentials können dynamisch generiert werden

Kubernetes Secrets

Credentials aus dem Deployment herauslösen



Kubernetes Secrets

Wie werden Kubernetes Secrets erstellt ?

- über eine Resource YAML-Datei, secrets.yaml

```
apiVersion: v1
kind: Secret
  name: my-release-mariadb
  namespace: default
data:
  mariadb-password: TmRVOGNLV2pYUA== *
type: Opaque
```

* BASE64 encoded

Kubernetes Secrets

Wie werden Kubernetes Secrets erstellt ?

- aus der Datei wird die Resource im Kubernetes Cluster erstellt.

```
# kubectl apply -f secret.yaml
```

Kubernetes Secrets

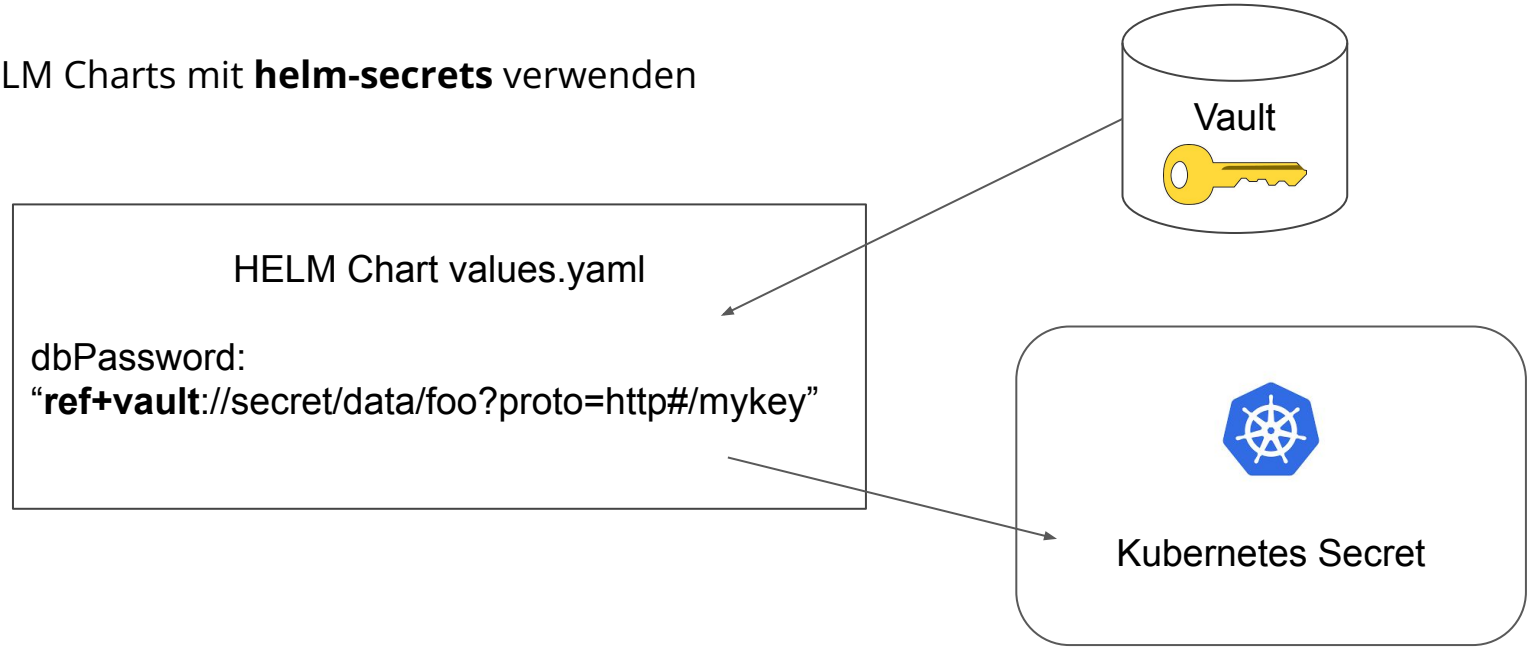
Wie werden Kubernetes Secrets erstellt ?

- Secrets können auch über einen HELM Chart deployed werden
    - mit helm-secrets:
        - Values werden im Helm Chart verschlüsselt abgelegt
        - externe Secret Manager können verwendet werden
            - AWS SecretManager
            - Azure KeyVault
            - HashiCorp Vault
    - Secret im Helm Template dynamisch erzeugen

Kubernetes Secrets

Wie werden Kubernetes Secrets erstellt ?

HELM Charts mit **helm-secrets** verwenden



Kubernetes Secrets

Wie werden Kubernetes Secrets erstellt ?

Parameter im HELM Template secrets.yaml dynamisch erzeugen:

```
…
{{- $password = randAlphaNum 10 }}
…
data:
  mariadb-root-password: {{ print $rootPassword | b64enc | quote }}
```

Kubernetes Secrets

Wie werden Kubernetes Secrets gespeichert ?

- Secrets werden im Kubernetes Key-Value Store gespeichert (etcd)
- per default im Klartext, also **nicht** verschlüsselt !
- Zusätzliche Verschlüssellung erforderlich
  - Kubernetes Encrypt Secret Data at Rest

Kubernetes Secrets

Wer hat Zugriff auf die Secrets ?

- Jeder Benutzer mit API Zugriff
- Innerhalb eines Namespace:
  - Jeder Benutzer mit der Berechtigung Deployments zu starten

Kubernetes Secrets

Wie werden Kubernetes Secrets im Pod (Container) verwendet ?

- Secrets können als **Environment Variable** im Pod zur Verfügung gestellt werden

```
containers:
  env:
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-password
          key: db-password
```

Kubernetes Secrets

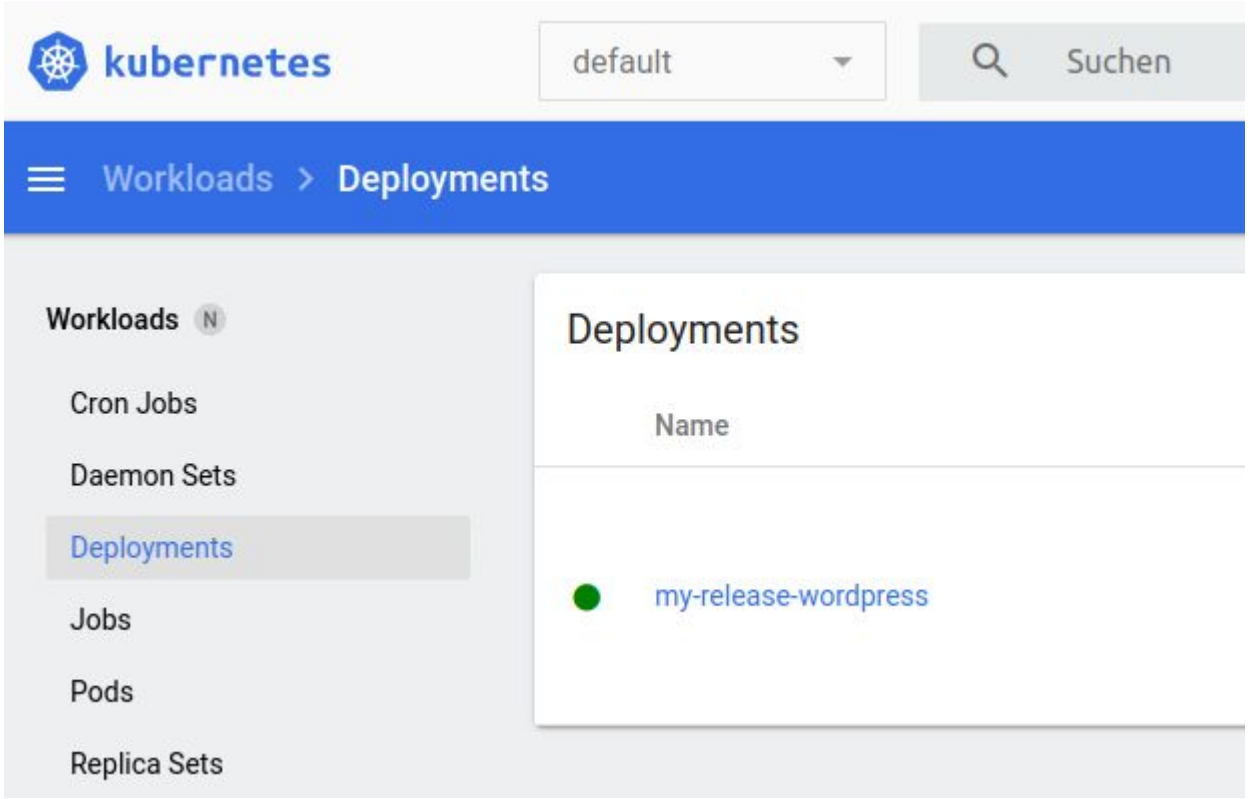
Wie werden Kubernetes Secrets im Pod (Container) verwendet ?

- Secrets können auch als **Volume gemounted** werden

```
volumes:
  - name: tls-secret-volume-ui
    secret:
      secretName: tls-secret-ui

containers:
  volumeMounts:
    - name: tls-secret-volume-ui
      readOnly: true
      mountPath: /cert-manager/certs
```

Kubernetes Secrets



Kubernetes Secrets

● wordpress

Image
docker.io/bitnami/wordpress:6.5.2-debian-12-r7

Status

Ready	Started	Started At
true	true	2024-05-04T14:04:45Z

Environment Variables

ALLOW_EMPTY_PASSWORD	APACHE_HTTP_PORT_NUMBER	APACHE_HTTPS_PORT_NUMBER	B
yes	8080	8443	f

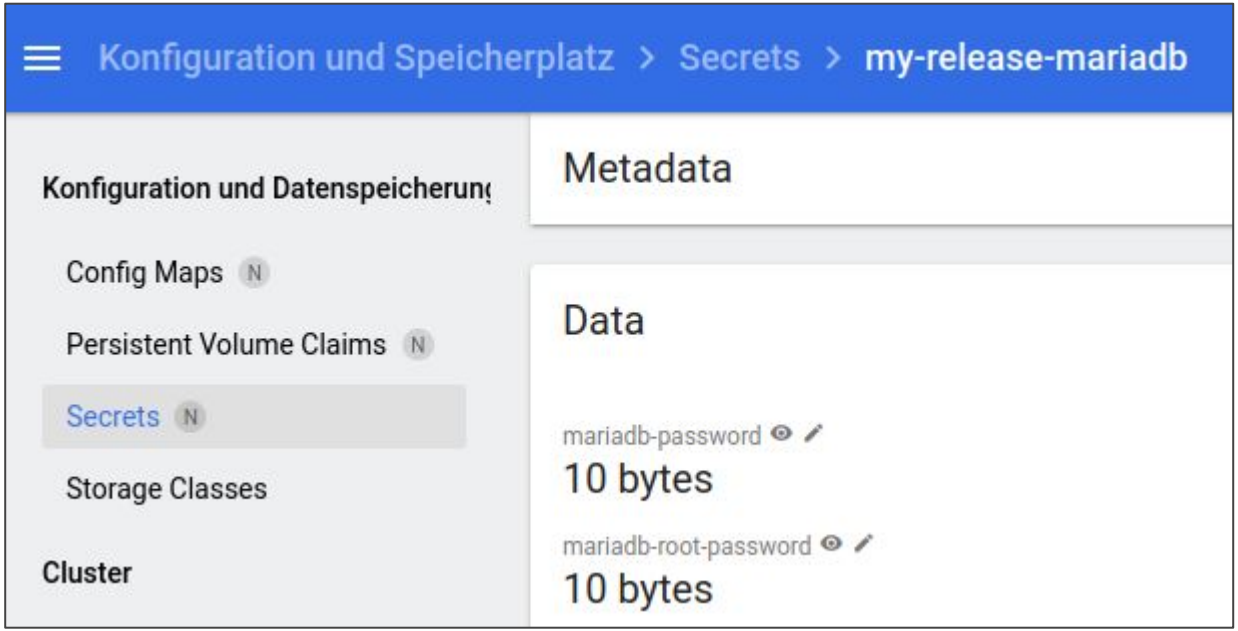
WORDPRESS_BLOG_NAME	WORDPRESS_DATABASE_NAME	WORDPRESS_DATABASE_PASSWORD
User's Blog!	bitnami_wordpress	10 Bytes

Kubernetes Secrets

Wordpress Deployment File:

```
containers:
  name: wordpress
  env:
    - name: WORDPRESS_DATABASE_PASSWORD
      valueFrom:
        secretKeyRef:
          name: my-release-mariadb
          key: mariadb-password
```

Kubernetes Secrets



Kubernetes Secrets

Secret aus dem etcd Key-Value-Store auslesen:

```
ETCDCTL_API=3 etcdctl

--cacert=/etc/kubernetes/pki/etcd/ca.crt
--cert=/etc/kubernetes/pki/etcd/server.crt
--key=/etc/kubernetes/pki/etcd/server.key

get /registry/secrets/default/my-release-mariadb | hexdump -C
```

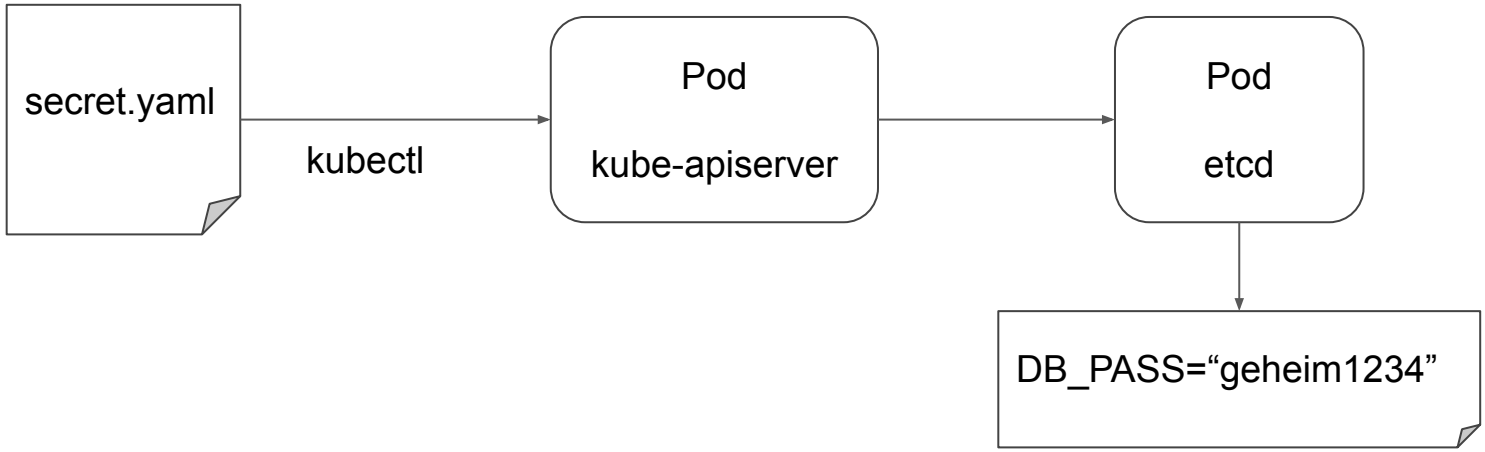
Kubernetes Secrets

Secret aus dem etcd Key-Value-Store auslesen:

```
00000000  2f 72 65 67 69 73 74 72  79 2f 73 65 63 72 65 74  |/registry/secret|
00000010  73 2f 64 65 66 61 75 6c  74 2f 6d 79 2d 72 65 6c  |s/default/my-rel|
00000020  65 61 73 65 2d 6d 61 72  69 61 64 62 0a 6b 38 73  |ease-mariadb.k8s|
00000030  00 0a 0c 0a 02 76 31 12  06 53 65 63 72 65 74 12  |.....v1..Secret.|
00000040  d5 06 0a 85 06 0a 12 6d  79 2d 72 65 6c 65 61 73  |.......my-releas|
00000050  65 2d 6d 61 72 69 61 64  62 12 00 1a 07 64 65 66  |e-mariadb....def|
```

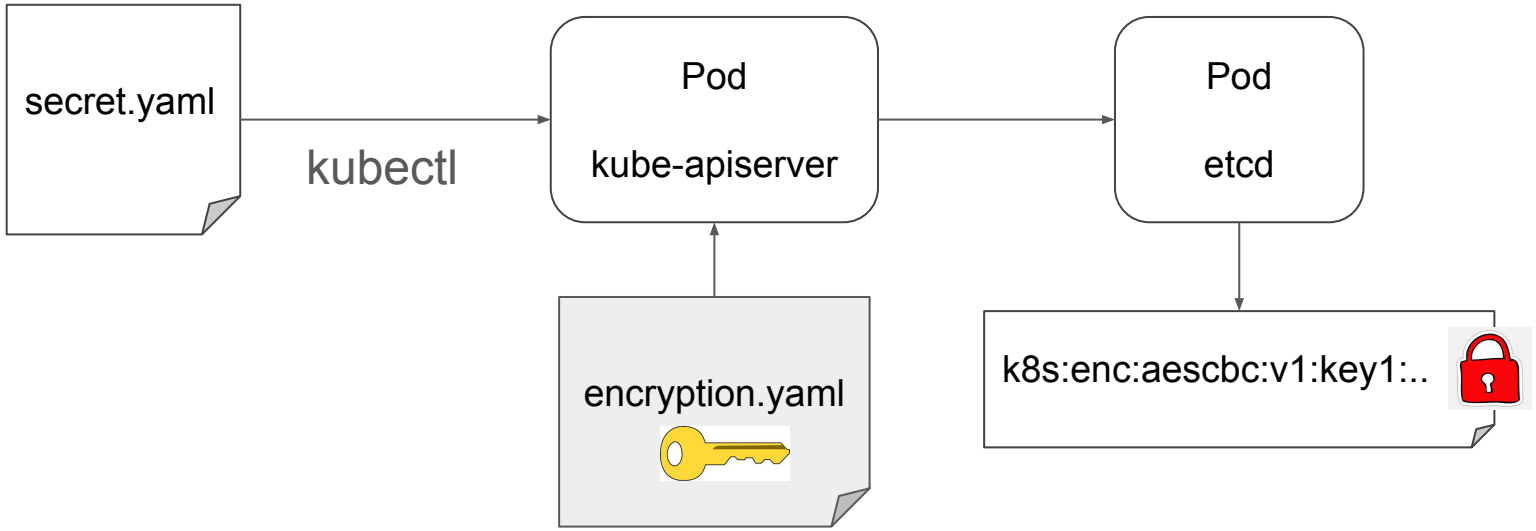
Kubernetes Secrets

Kubernetes Encrypting Confidential Data at Rest



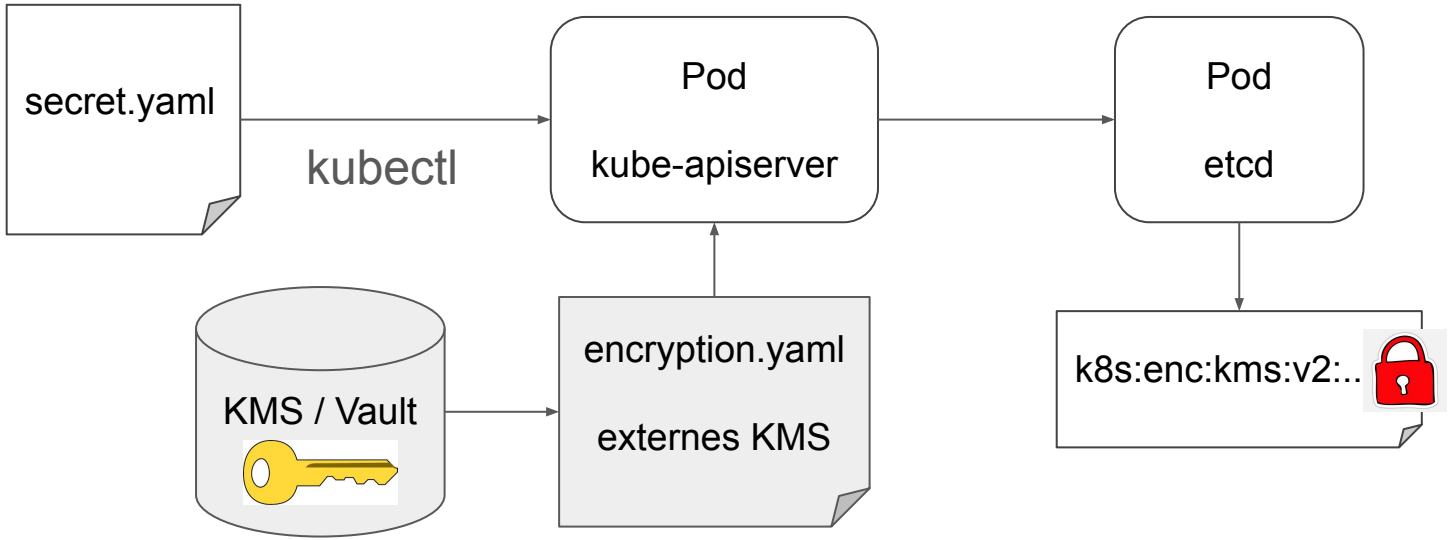
Kubernetes Secrets

Kubernetes Encrypting Confidential Data at Rest



Kubernetes Secrets

Kubernetes Encrypting Confidential Data at Rest



Kubernetes Secrets

Secret Encryption Konfiguration im Kubernetes Cluster mit:

- AES Verschlüssellung
- Key in der Konfigurationsdatei enthalten

Kubernetes Secrets

Secret Encryption Konfiguration

Encryption Key erzeugen:

```
bash# head -c 32 /dev/urandom | base64
E4Zhhq1GFgJlNtbcJ52RUzua6XVIXQ+Fxn1Rsqu8WCU=
```

Kubernetes Secrets

Secret Encryption Konfiguration

EncryptionConfiguration.yaml erstellen:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: E4Zhhq1GFgJlNtbcJ52RUzua6XVIXQ+Fxn1Rsqu8WCU=
      - identity: {}
```

Kubernetes Secrets

Secret Encryption Konfiguration

In der EncryptionConfiguration.yaml Datei wird unter “resources” festgelegt welche Objekte verschlüsselt werden sollen:

```
- resources:
    - secrets
    - configmaps
```

Es können auch Wildcards verwendet werden. Mit *.* wird alles verschlüsselt.

Kubernetes Secrets

Secret Encryption Konfiguration

Wir legen die Datei auf allen Control-Plane Nodes ab.

```
bash# scp EncryptionConfiguration.yaml \
      node01:/etc/kubernetes/enc/enc.yaml
```

Kubernetes Secrets

Secret Encryption Konfiguration

Jetzt konfigurieren wir den kube-apiserver Pod und binden die Encryption Datei ein:

```
bash# cd /etc/kubernetes/manifests/
bash# vi kube-apiserver.yaml
```

Vorsicht:

Die Manifest Datei steuert den Pod. Je nach Konfiguration des Kubernetes Cluster wird ein Neustart des Pod ausgelöst, sobald die Datei gespeichert wird.

Kubernetes Secrets

Secret Encryption Konfiguration

```
    command:
    - --encryption-provider-config=/etc/kubernetes/enc/enc.yaml
…
    volumeMounts:
    - mountPath: /etc/kubernetes/enc
      name: enc
      readOnly: true
…
  volumes:
  - hostPath:
      path: /etc/kubernetes/enc
      type: DirectoryOrCreate
    name: enc
```

Kubernetes Secrets

Secret Encryption Konfiguration

Neustart des kube-apiserver Pod:

```
bash# mv /etc/kubernetes/manifests/kube-apiserver.yaml /root/
bash# sleep 30
bash# mv /root/kube-apiserver.yaml /etc/kubernetes/manifests/
```

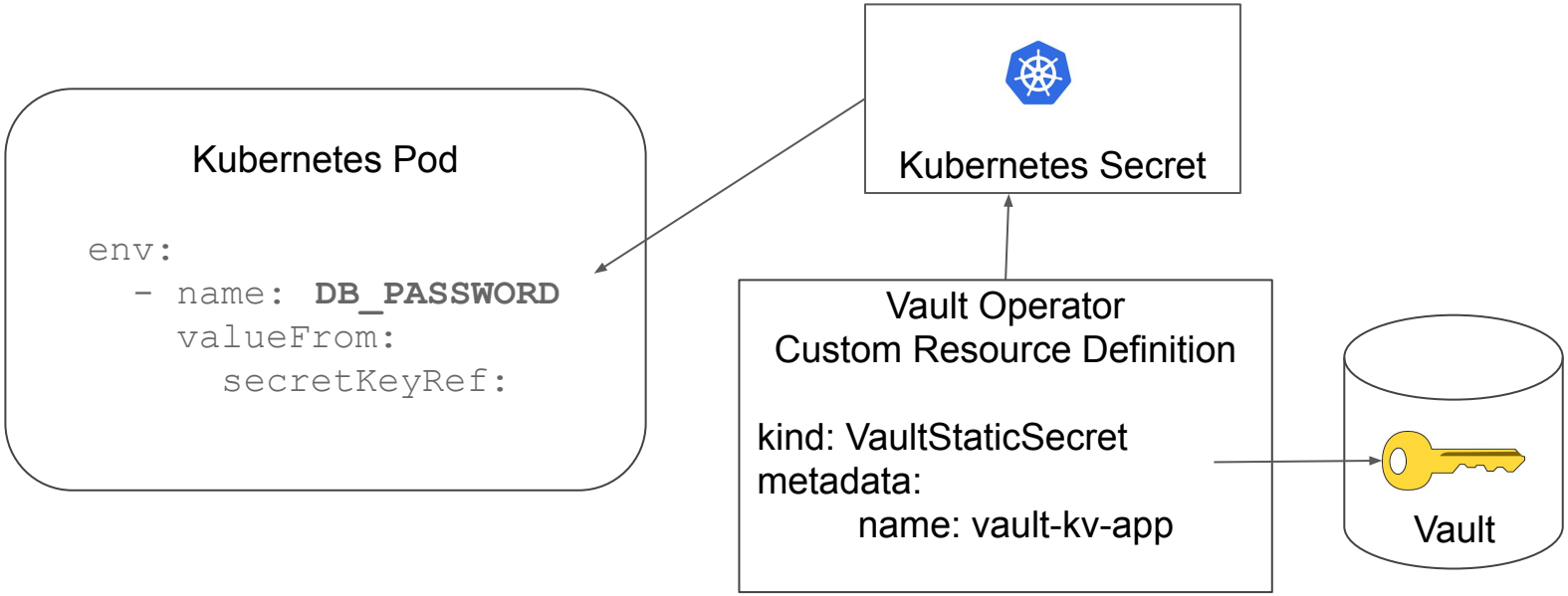
Kubernetes Secrets

Verwaltung der Kubernetes Secrets über Hashicorp Vault Operator

- Vault Operator wird über HELM Chart oder Kustomize installiert
- Secrets werden über Vault verwaltet und mit Kubernetes Secrets synchronisiert
- die Anwendung im Pod sieht nur das Kubernetes Secret

Kubernetes Secrets

Verwaltung der Kubernetes Secrets über Hashicorp Vault Operator



Docker Swarm Secrets

Docker Swarm Secrets (keep it simple stupid)

- werden als Volume im Container gemounted
- sichtbar unter /run/secrets/<secret_name>
- werden intern verschlüsselt abgespeichert
- keine Schnittstelle für externe Verwaltung über Vault oder ähnliches

Docker Swarm Secrets

Docker Swarm Secrets erstellen:

- über die Kommandozeile:

```
# echo “geheim1234” | docker secret create db-password -
```

Docker Swarm Secrets

Docker Swarm Secrets erstellen:

- über das Docker Compose File mit Referenz auf eine Datei

```
services:
  rabbitmq:
    secrets:
      - rabbitmq_default_pass

secrets:
  rabbitmq_default_pass:
    file: rabbitmq_default_pass.txt
```

Docker Swarm Secrets

Referenz zum Docker Swarm Secret im Docker Compose File

In vielen Standard Images kann über Environment Variable
direkt auf das Secret zugegriffen werden

```
services:
  wordpress:
      image: wordpress
      environment:
        - WORDPRESS_DB_PASSWORD_FILE=/run/secrets/mysql-root
```

Docker Swarm Secrets

Referenz zum Docker Swarm Secret im Docker Compose File

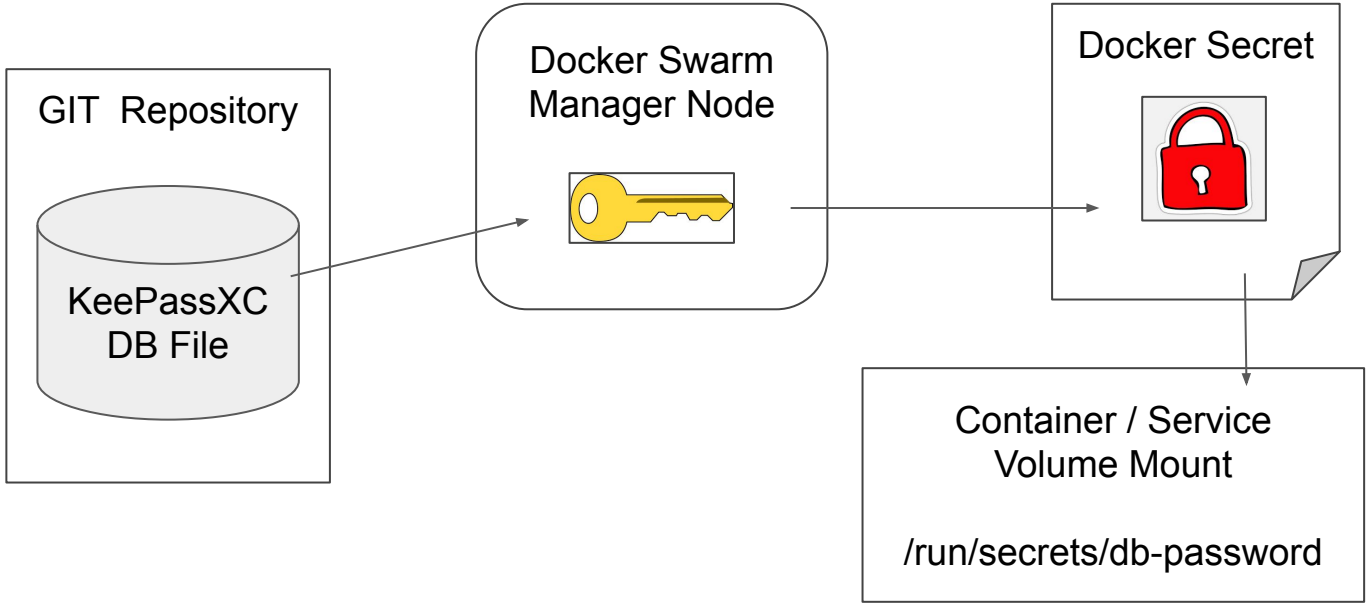
Falls das nicht vorgesehen ist, kann man sich z.B. mit einem Shell-Befehl im entrypoint des Containers behelfen:

```
#!/bin/bash

export WORDPRESS_DB_PASSWORD=$( cat /run/secrets/mysql-root )
```

Docker Swarm Secrets

Beispiel eines Deployment mit verschlüsselter Übertragung der Credentials über einen Passwort Manager



zum Schluss

Umgang mit Passwörter und anderen Credentials im Deployment Prozess

- nicht im GIT Repo ablegen
- nicht in Build Artefakte einbauen (Build Caches, Docker Images)
- nur dort zur Verfügung stellen, wo sie gebraucht werden
- Tools zur Verschlüssellung verwenden
- Credentials nur verschlüsselt transportieren
- Credentials wenn möglich dynamisch erzeugen

zum Schluss

Links zum Thema

Kubernetes Secrets:
https://kubernetes.io/docs/concepts/configuration/secret/

Kubernetes Encrypt Data at Rest
https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

HELM Secrets:
https://github.com/jkroepke/helm-secrets
https://github.com/helmfile/vals

Kubernetes Vault Operator für Kubernetes Secrets verwenden:
https://developer.hashicorp.com/vault/tutorials/kubernetes/vault-secrets-operator