11
09/2025

Leon Krass, Felix Kerker, Lisa Vossen

Where to Hide Your Dirty (Technical) Secrets in Kubernetes

Lisa Vossen

System Engineer
Managed Kubernetes

/ About me

Introduction

Loves IT-infrastructure and automating it! Passionate about self-services, APIs and Kubernetes

Ansible, Terraform, GO, Python, Kubernetes, CI/CD

Felix Kerker

System Engineer

/ About me

Introduction

Passionate about uncovering patterns across tech stacks and bending rules for creative solutions. Enjoys hacking on all levels embedded hardware to cloud-native infrastructure with:

Vault, Kubernetes, Terraform, and Ansible.
💙 coding in Python and Go.

Leon Krass

Technical Leader
HashiCorp Vault

/ About me

Introduction

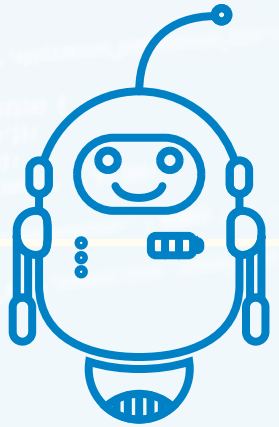
Passionate about secrets management and (sometimes over-) engineering IT automation with a touch of Kubernetes, Terraform and/or Ansible magic.

HashiCorp Vault, Kubernetes, Ansible, Terraform

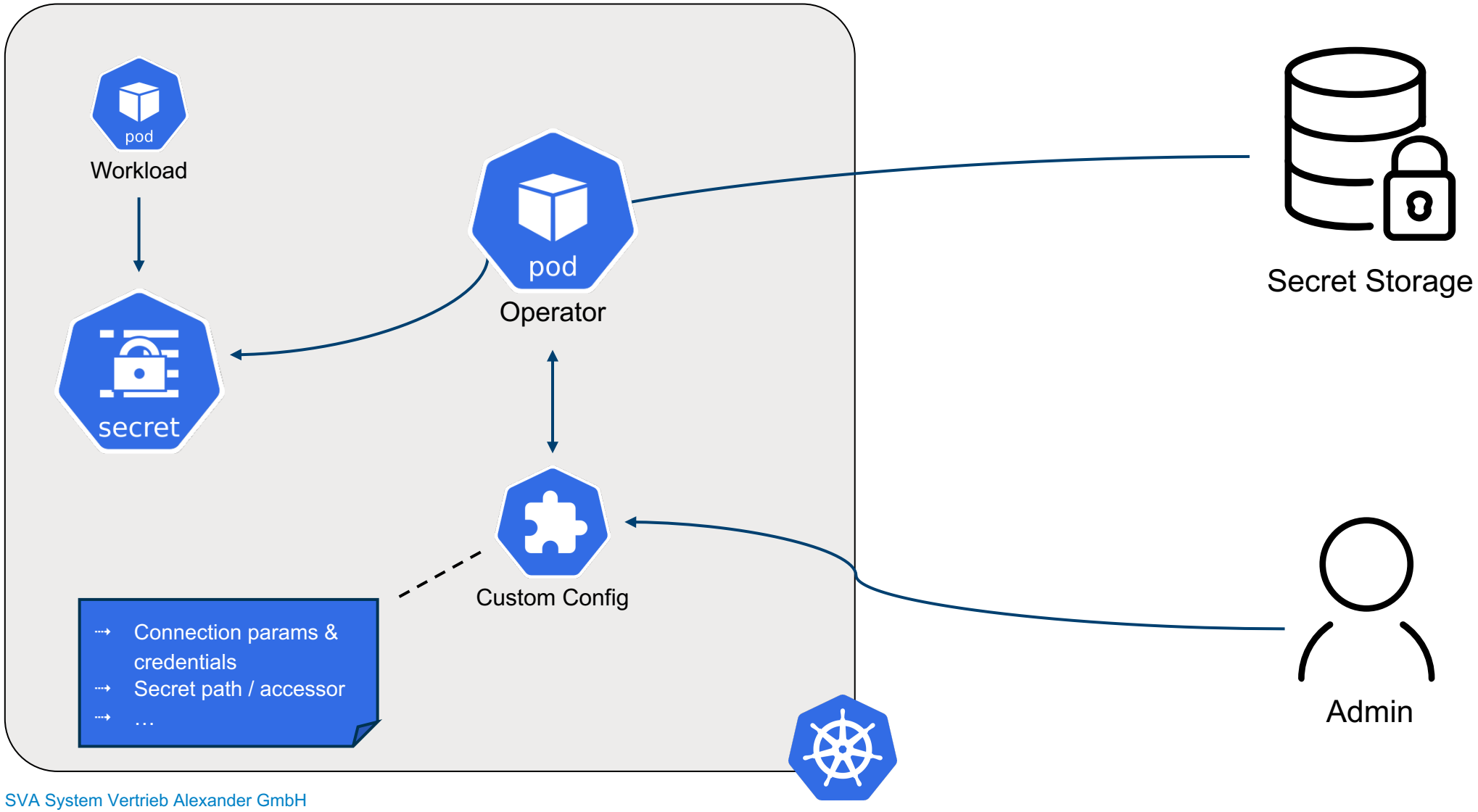
Agenda

1. **Operator & CSI based solutions**
2. **Sidecar & mutating webhook based solutions**
3. **Bridging Worlds: Secrets as Code**
4. **Wrap-up**

Operator & CSI based solutions



How it works


pod
Workload
secret
pod
Operator
Custom Config
Connection params & credentials
Secret path / accessor
…
Secret Storage
Admin

(Dis-) Advantages

- Easy and intuitive usage through native K8s Secrets
- Integration with existing Helm charts
- Loose coupling (operator is easy to exchange)

- Native K8s are stored in etcd and can potentially get extracted
- Operator as additional workload (maintenance, load, attack surface, …)

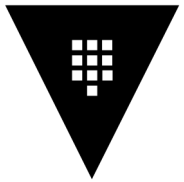
Examples

External Secrets Operator

- Supports multiple secret backends (e. g. HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secrets Manager, Akeyless, …)
- Backend unspecific custom resource definitions

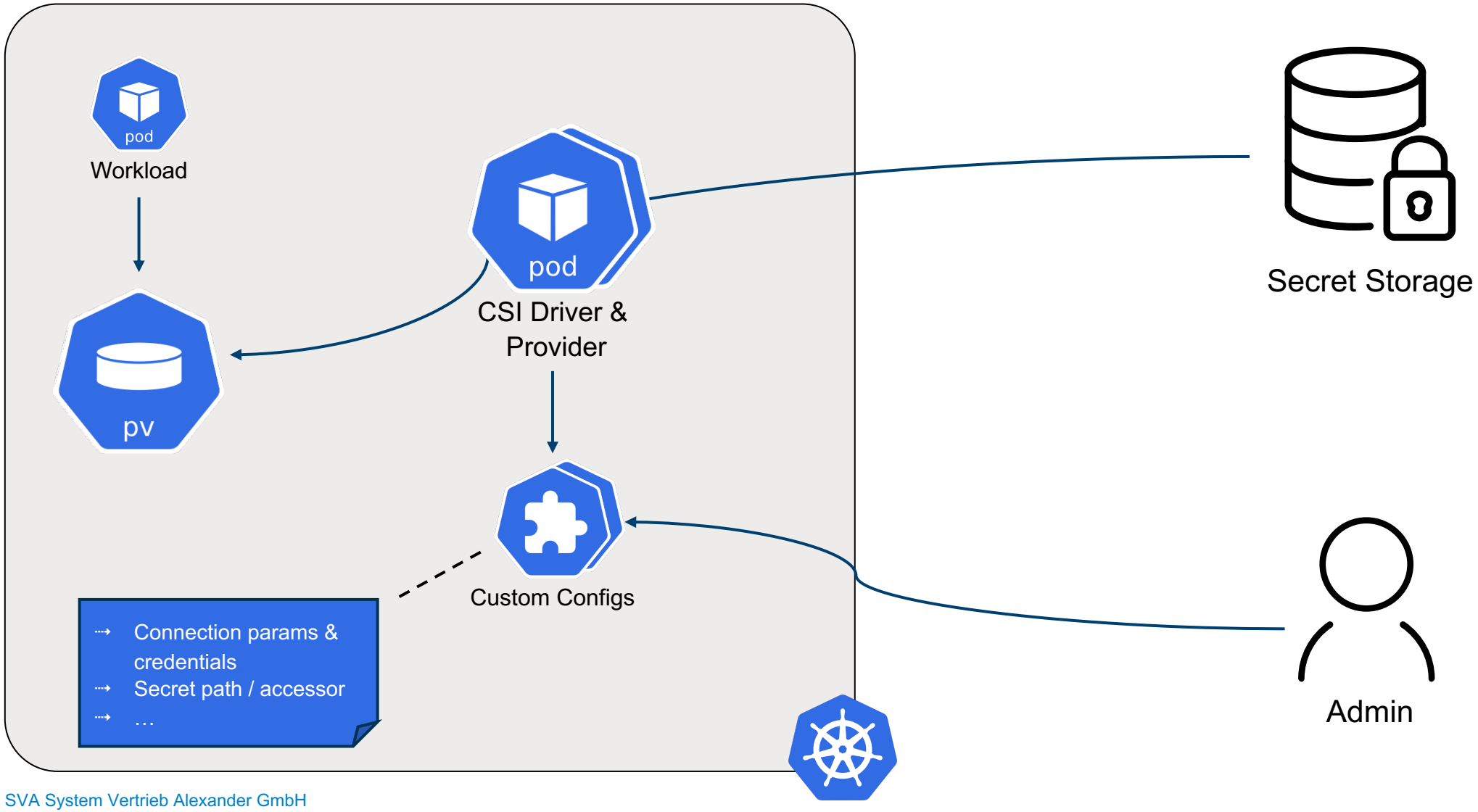
Vault Secrets Operator

- Only supports HashiCorp Vault
- Vault specific custom resource definitions
- Supported by HashiCorp with Vault Enterprise



Demo

How it works


pod
Workload
pv
pod
CSI Driver &
Provider
Custom Configs
Connection params &
credentials
Secret path / accessor
…
Secret Storage
Admin

(Dis-) Advantages

- PV contents (= secrets) are not stored in etcd, can only be read within Pods
- Loose coupling (PVs act as interface, separate CSI provider per backend)

- Integration with existing Helm charts
- Backwards compatibility with existing configuration
- Operator as additional workload (maintenance, load, attack surface, …)

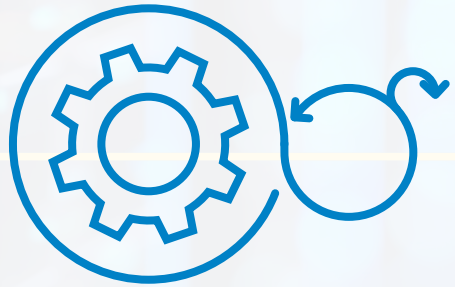
Examples

Secrets Store CSI Driver

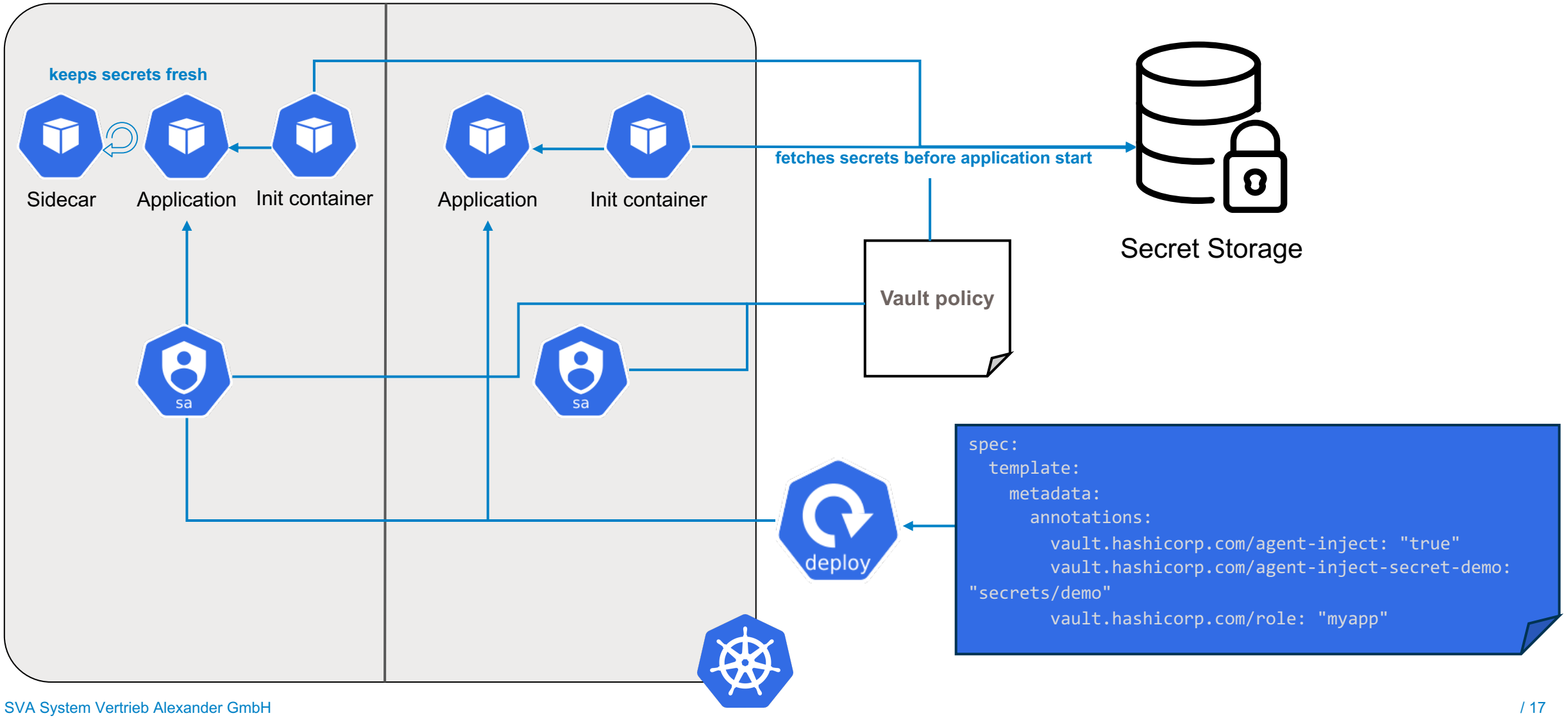
- Supports multiple secret backends (e. g. HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secrets Manager, Akeyless, …)
- Backends are installed as separate CSI provider workloads
- Subproject of K8s' SIG Auth

Demo

Sidecar & webhook based solutions



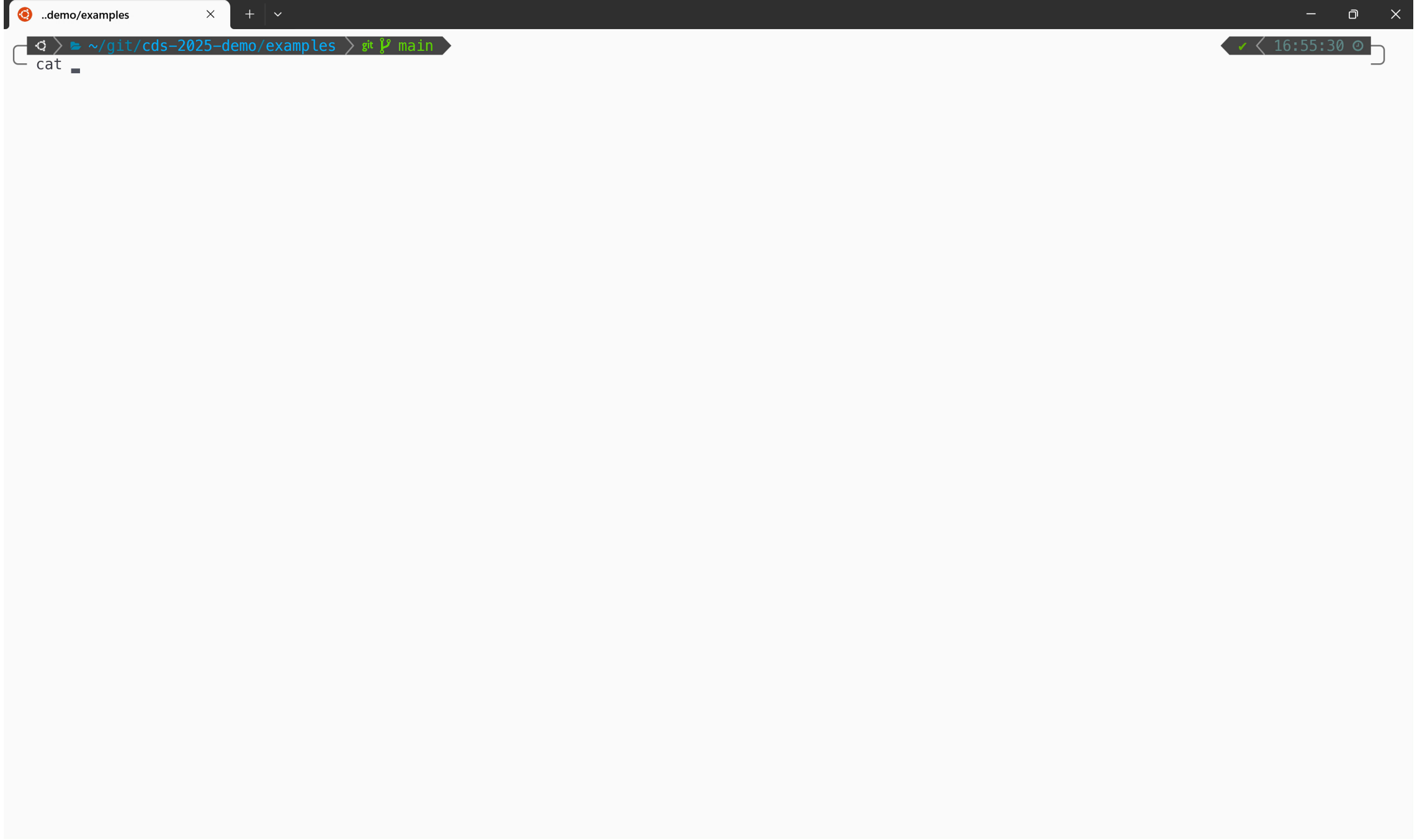
How it works

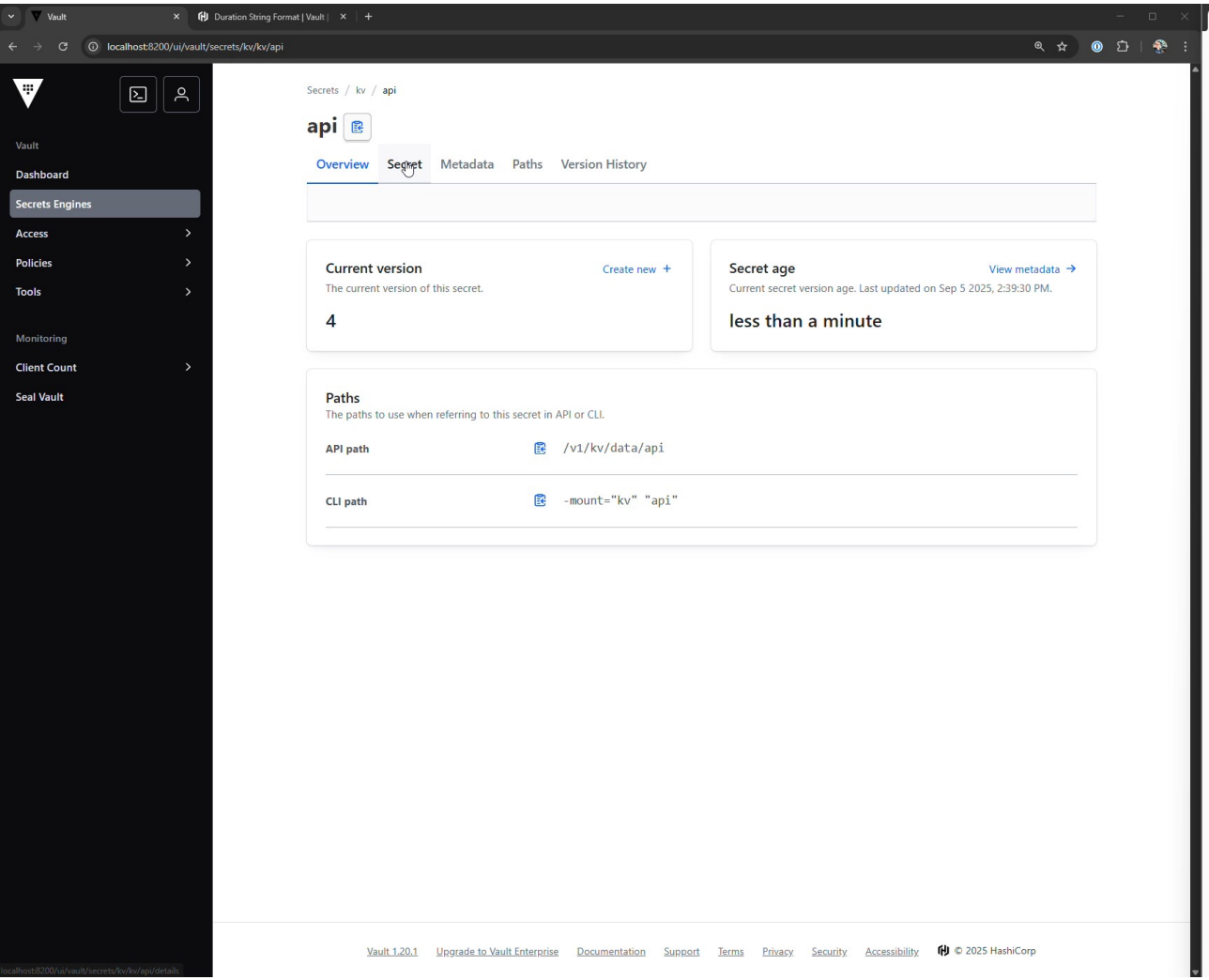

keeps secrets fresh
Sidecar
Application
Init container
Application
Init container
fetches secrets before application start
Secret Storage
Vault policy
sa
sa
deploy
spec:
template:
metadata:
annotations:
vault.hashicorp.com/agent-inject: "true"
vault.hashicorp.com/agent-inject-secret-demo:
"secrets/demo"
vault.hashicorp.com/role: "myapp"

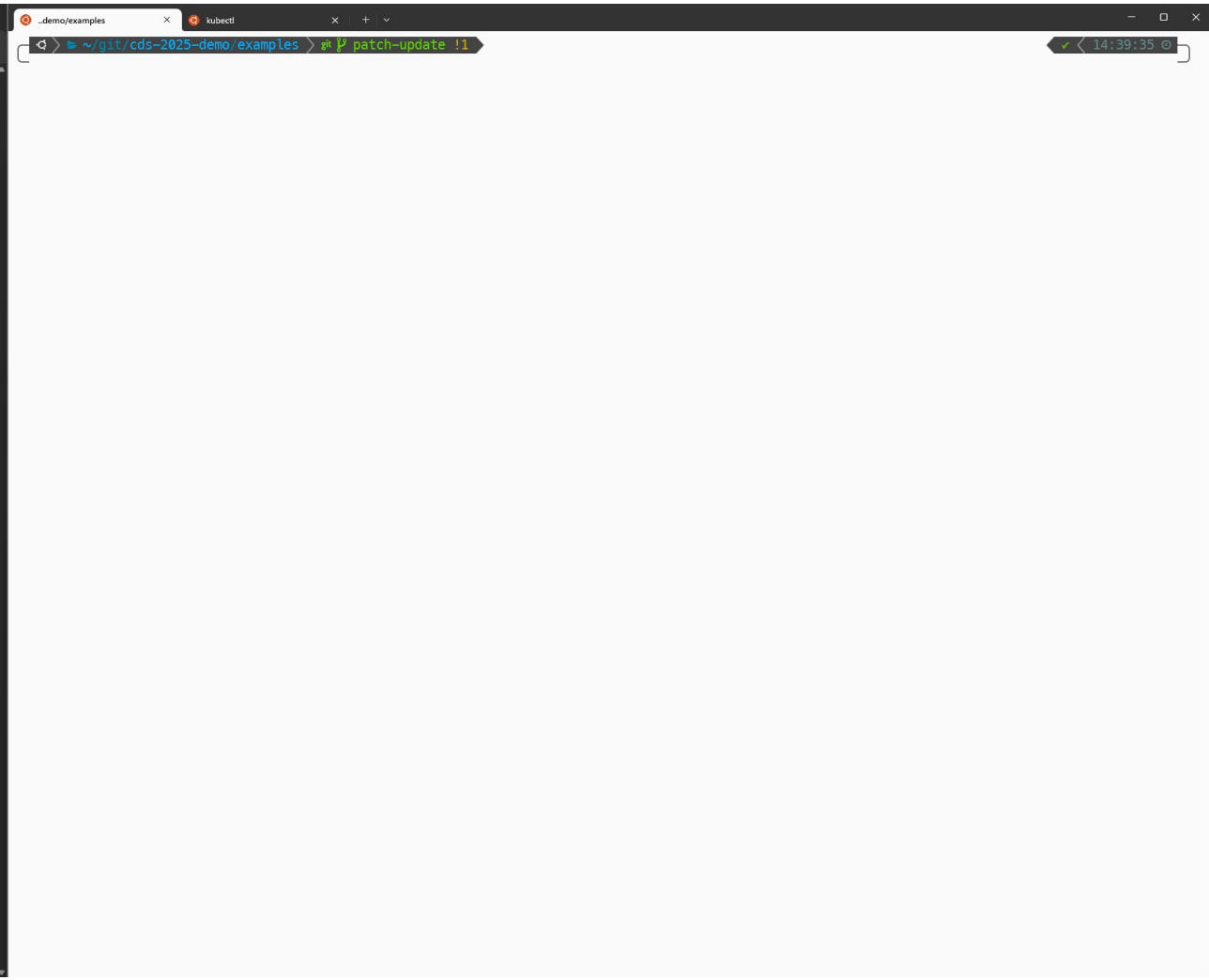
(Dis-) Advantages

- Easy sidecar approach keeps secrets up-to-date
- Secrets are not stored in etcd
- Easy usage via annotations

- More resource consumption due to additional workloads
- Version of secret in use not visible without accessing the container


..demo/examples
~/git/cds-2025-demo/examples
git main
16:55:30
cat


Vault
Duration String Format | Vault
localhost:8200/ui/vault/secrets/kv/api
Vault
Dashboard
Secrets Engines
Access
Policies
Tools
Monitoring
Client Count
Seal Vault
Secrets / kv / api
api
Overview
Secret
Metadata
Paths
Version History
Current version
Create new +
The current version of this secret.
4
Secret age
View metadata →
Current secret version age. Last updated on Sep 5 2025, 2:39:30 PM.
less than a minute
Paths
The paths to use when referring to this secret in API or CLI.
API path
/v1/kv/data/api
CLI path
-mount="kv" "api"
Vault 1.20.1
Upgrade to Vault Enterprise
Documentation
Support
Terms
Privacy
Security
Accessibility
© 2025 HashiCorp


..demo/examples
kubectl
~/git/cds-2025-demo/examples
patch-update !1
14:39:35

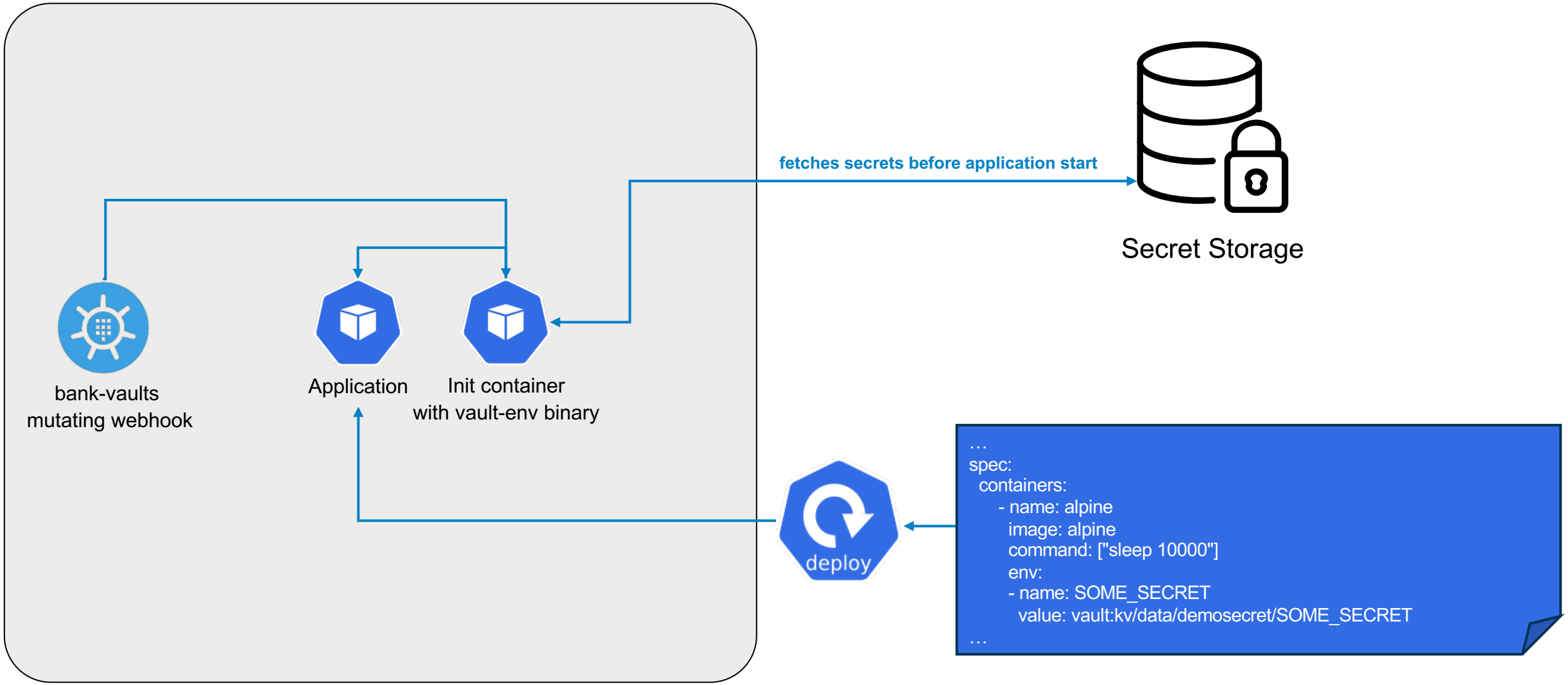
Examples

vault-k8s (Vault Agent Injector)

⇢ Great blog post: https://www.hashicorp.com/de/blog/injecting-vault-secrets-into-kubernetes-pods-via-a-sidecar

⇢ https://github.com/hashicorp/vault-k8s (⭐820)

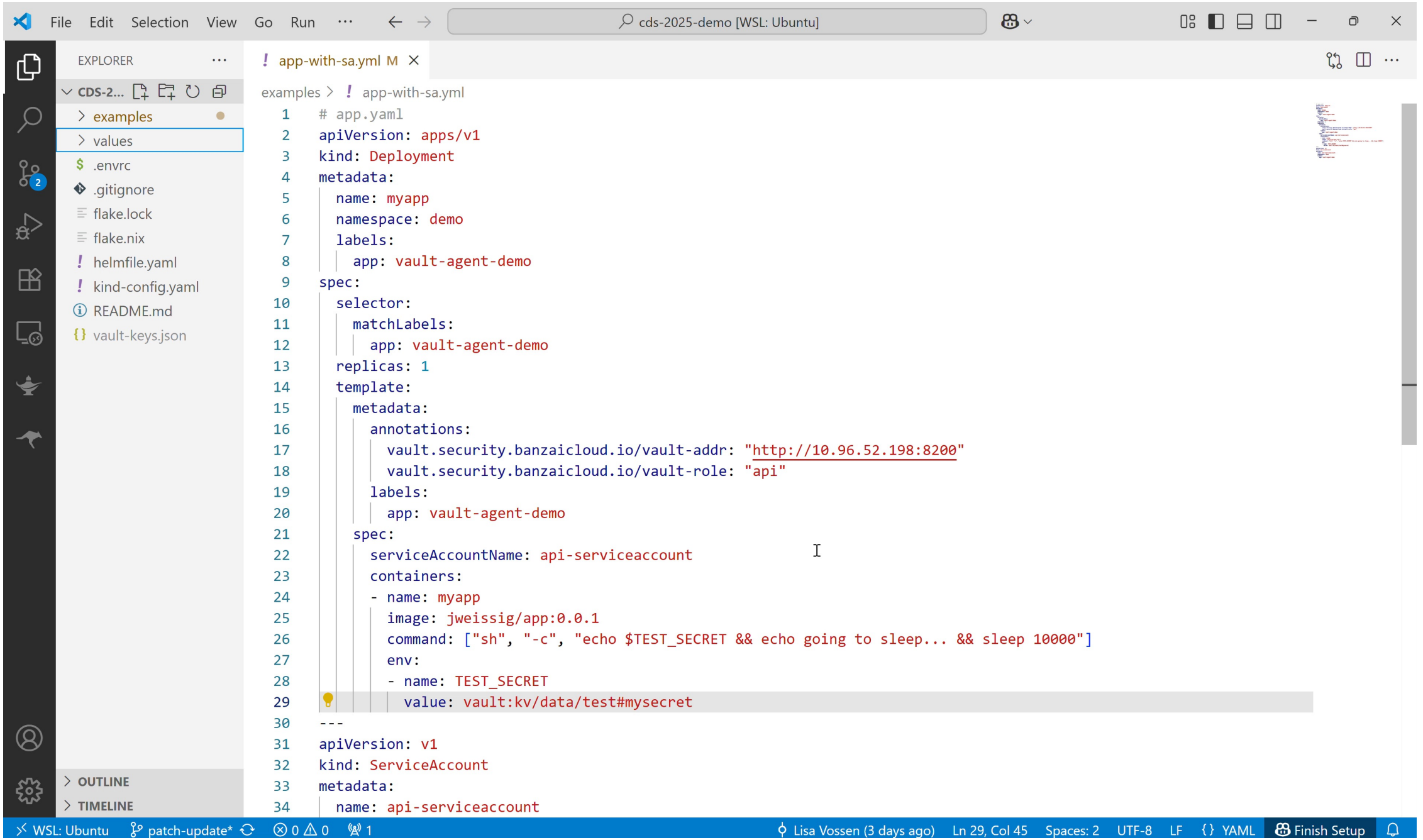
How it works


fetches secrets before application start
Secret Storage
bank-vaults
mutating webhook
Application
Init container
with vault-env binary
deploy
…
spec:
containers:
- name: alpine
image: alpine
command: ["sleep 10000"]
env:
- name: SOME_SECRET
value: vault:kv/data/demosecret/SOME_SECRET
…

(Dis-) Advantages

- Secrets are only stored in memory, not persisted on etcd
- Lightweight solution, no sidecar
- Option to make it even more robust by deactivating `kubectl exec` for containers

- No updates of secrets when changes at Vault happen; only with Pod restarts
- Environment variable exposure possible (if shell access is gained)


File Edit Selection View Go Run
cds-2025-demo [WSL: Ubuntu]
EXPLORER
app-with-sa.yml M
examples > app-with-sa.yml
CDS-2...
examples
values
.envrc
.gitignore
flake.lock
flake.nix
helmfile.yaml
kind-config.yaml
README.md
vault-keys.json
OUTLINE
TIMELINE
1 # app.yaml
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5 name: myapp
6 namespace: demo
7 labels:
8 app: vault-agent-demo
9 spec:
10 selector:
11 matchLabels:
12 app: vault-agent-demo
13 replicas: 1
14 template:
15 metadata:
16 annotations:
17 vault.security.banzaicloud.io/vault-addr: "http://10.96.52.198:8200"
18 vault.security.banzaicloud.io/vault-role: "api"
19 labels:
20 app: vault-agent-demo
21 spec:
22 serviceAccountName: api-serviceaccount
23 containers:
24 - name: myapp
25 image: jweissig/app:0.0.1
26 command: ["sh", "-c", "echo $TEST_SECRET && echo going to sleep... && sleep 10000"]
27 env:
28 - name: TEST_SECRET
29 value: vault:kv/data/test#mysecret
30 ---
31 apiVersion: v1
32 kind: ServiceAccount
33 metadata:
34 name: api-serviceaccount
WSL: Ubuntu
patch-update*
0 0 1
Lisa Vossen (3 days ago)
Ln 29, Col 45
Spaces: 2
UTF-8
LF
YAML
Finish Setup

Examples

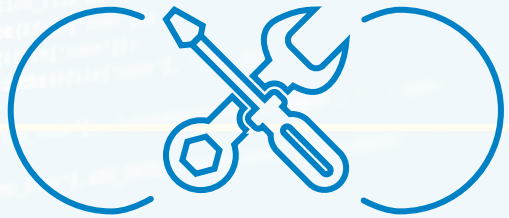
bank-vaults Mutating Webhook

⇢ Great blog post: https://bank-vaults.dev/docs/mutating-webhook

⇢ https://github.com/bank-vaults/vault-secrets-webhook (⭐ 70)

⇢ https://github.com/bank-vaults/vault-env (⭐33)

Bridging Worlds: Secrets as Code



What is the challenge?

- Integrate external secrets into existing Helm charts
- Provision Vault resources (Policies, Roles, …) in addition to K8s CRDs
- Keep your Configuration DRY
- ⭐ Don't create any secret data by hand
- \+ Don't expose it in Terraform tfstate either

The Idea in 4 Building Blocks

⇢ **External Secrets Operator**
→ Handles data flow between Kubernetes and Vault

⇢ **Kubernetes + Vault JWT Authentication**
→ One of the available auth methods

⇢ **Terraform Ephemeral Resources**
→ Secret generation without state

⇢ **GitOps Bridge Pattern**
→ Connects Terraform ↔ ArgoCD

Vault JWT Authentication

- One of the available integration variants
  - An alternative with different variations is the Kubernetes Authentication Engine
- Requires Vault to get access to the JWKs / openid-configuration
- To control secret access, policies and roles must be created in Vault
- 🍀 Vault has an excellent Terraform provider


```
{
  "aud": [
    "https://kubernetes.default.svc.cluster.local",
    "k3s"
  ],
  "exp": 1757452981,
  "iat": 1757449381,
  "iss": "https://kubernetes.default.svc.cluster.local",
  "jti": "b4ee3c17-24e0-42a7-8a1d-e0e87c3bf4cf",
  "kubernetes.io": {
    "namespace": "guestbook",
    "serviceaccount": {
      "name": "vault-secret-store",
      "uid": "7eda5673-12a5-4ed2-91dd-ace6a88021cc"
    }
  },
  "nbf": 1757449381,
  "sub": "system:serviceaccount:guestbook:vault-secret-store"
}
```

Vault JWT Authentication

```hcl
data "vault_policy_document" "this" {
  rule {
    path         = "${var.kv_path}/data/${var.name}/*"
    capabilities = ["read"]
    description  = "Allow reading secrets in ${var.kv_path}/${var.name}/"
  }
}

resource "vault_policy" "this" {
  name   = "app_${var.name}"
  policy = data.vault_policy_document.this.hcl
}

resource "vault_jwt_auth_backend_role" "this" {
  backend        = var.jwt_auth_backend_path
  role_name      = var.name
  token_policies = [vault_policy.this.name]

  bound_audiences = var.bound_audiences
  bound_claims = {
    "/kubernetes.io/namespace"           = var.namespace
    "/kubernetes.io/serviceaccount/name" = var.service_account_name
  }
  user_claim = "sub"
  role_type  = "jwt"
}
```

Terraform Ephemeral Resources

⇢ **Temporary by Design**

→ Ephemeral resources are defined in an `ephemeral` block and have a unique lifecycle. Terraform does **not** store them in state or plan files.

⇢ **Key Elements**

→ Ephemeral Resource Blocks

→ Write-Only arguments

Terraform Ephemeral Resources

```
ephemeral "random_password" "this" {
  for_each         = var.secrets
  length           = 20
  special          = true
  override_special = "!#$%&*()-_=+[]{}<>:?"
}

resource "vault_kv_secret_v2" "this" {
  for_each             = var.secrets
  mount                = var.kv_path
  name                 = "${var.name}/${each.key}"
  data_json_wo_version = each.value.version
  data_json_wo = jsonencode(
    {
      "${each.value.key}" = ephemeral.random_password.this[each.key].result
    }
  )
}
```

GitOps Bridge Pattern

- Presented by Carlos Santana (AWS) on ArgoCon 24
  - https://github.com/gitops-bridge-dev/gitops-bridge

- Referenced in some articles and talks since then, there may be some variations

- I will stress it a little …
  - → … this is the hacky part.

GitOps Bridge Pattern

⇢ *Keep away Kubernetes resources from Terraform state as much as possible*

→ Discussed in many forms, just search for "Terraform vs Argo"

⇢ The *”bridge"* consists of:

→ Argo Cluster Secret

→ Argo Application Set

→ Argo Cluster Generator

```
$ terraform state list module.gitops_bridge_bootstrap

module.gitops_bridge_bootstrap.helm_release.argocd[0]
module.gitops_bridge_bootstrap.helm_release.bootstrap["addons"]
module.gitops_bridge_bootstrap.helm_release.bootstrap["workloads"]
module.gitops_bridge_bootstrap.kubernetes_secret_v1.cluster[0]
```

GitOps Bridge Pattern

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: workloads
  namespace: argocd
spec:
  syncPolicy:
    preserveResourcesOnDeletion: false
  generators:
    - clusters: {}
  template:
    metadata:
      name: workloads
      finalizers:
        - resources-finalizer.argocd.argoproj.io
    spec:
      project: default
      source:
        repoURL: '{{metadata.annotations.workload_repo_url}}'
        path: '{{metadata.annotations.workload_repo_basepath}}{{metadata.annotations.workload_repo_path}}'
        targetRevision: '{{metadata.annotations.workload_repo_revision}}'
        helm:
          valuesObject:
            spec:
              destination:
                server: '{{server}}'
            workloadsB64: '{{metadata.annotations.workload_config}}'
      destination:
        server: '{{server}}'
```

GitOps Bridge Pattern

(Do you see where this is going?)

```
      workload_config = base64encode(jsonencode(merge(
        { applications = local.workloads },
        {
          vault = {
            address       = "https://sl-029840.labda.sva.de:8200"
            kv_path       = "secret"
            caBundle      = "LS0tLS1C..."
            k8s_auth_path = "k8s"
            jwt_audiences = ["https://sl-029840.labda.sva.de"]
          },
        }
      )))
    }
  )
```

```
$ kubectl get secrets -n argocd in-cluster -o jsonpath={.metadata.annotations.workload_config}

eyJnZW5lcmFsIjp7InZhdWx0Ijp7ImFkZHJlc3MiOiJodHRwczovL3NsLTAyOTg0MC5sYWJkYS5zdmEuZGU6ODIwMCIsImNhQnVu
```

GitOps Bridge Pattern – Apps of Apps Chart

- Following an AppsOfApps pattern we generate a chart that takes the base64 encoded values as input
- It must be referenced in the ApplicationSet source
- It creates the workload downstream ArgoCD applications

```
{{/*
Get workloads configuration from either base64 or structured format
*/}}
{{- define "workloads.config" -}}
{{- if and .Values.workloadsB64 (ne .Values.workloadsB64 "") -}}
  {{/* Use base64 configuration - structure must match values.yaml exactly */}}
  {{- .Values.workloadsB64 | b64dec | fromJson | toJson -}}
{{- else -}}
  {{/* Use structured configuration from values.yaml */}}
  {{- dict "global" .Values.global "vault" .Values.vault "applications"
.Values.applications "sources" .Values.sources | toJson -}}
```

GitOps Bridge Pattern – Apps of Apps Chart

```
{{- $guestbook := include "workloads.guestbook" . | fromJson -}}
{{- $vault := include "workloads.vault" . | fromJson -}}
{{- $global := include "workloads.global" . | fromJson -}}
{{- $sources := include "workloads.sources" . | fromJson -}}

{{/*
Only create the application if guestbook is enabled
*/}}
{{- if ne ($guestbook.enabled | toString) "false" -}}
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: {{ $guestbook.name }}
  namespace: {{ $global.argocd.namespace }}
  labels:
    {{- include "workloads.labels" . | nindent 4 }}
    app.kubernetes.io/component: guestbook
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  destination:
    namespace: {{ $guestbook.namespace }}
    server: {{ $global.argocd.destinationServer | default
"https://kubernetes.default.svc" }}
```

```
  sources:
    # Main application source
    - repoURL: {{ $sources.application.repoURL }}
      path: {{ $sources.application.path }}
      targetRevision: HEAD

    # Vault secrets source
    - repoURL: {{ $sources.vaultSecrets.repoURL }}
      path: {{ $sources.vaultSecrets.path }}
      targetRevision: HEAD
      helm:
        valuesObject:
          vault:
            address: {{ $vault.address }}
            kv_path: {{ $vault.kvPath }}
            caBundle: {{ $vault.caBundle }}
          auth:
            jwt:
              path: {{ $vault.k8sAuthPath }}
              role: {{ $guestbook.name }}
            serviceAccount:
              audiences: {{ $vault.jwtAudiences | toYaml | nindent 16
          workload: {{ $guestbook | toYaml | nindent 12 }}

  syncPolicy: {{ $global.syncPolicy | toYaml | nindent 4 }}
{{- end -}}
```

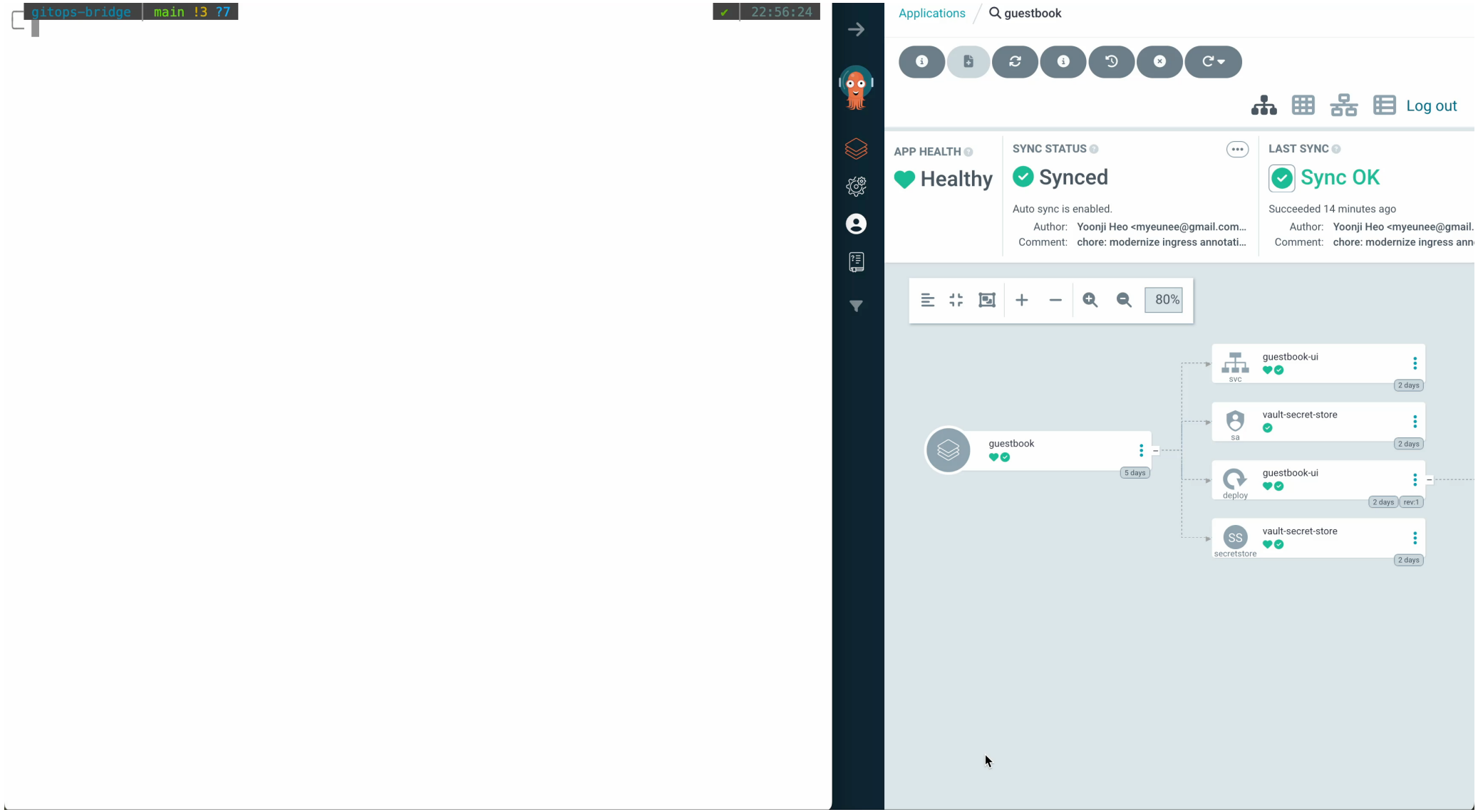
GitOps Bridge Pattern – Secret Helper Chart

```
{{- range $name, $config := .Values.workload.secrets }}
---
apiVersion: external-secrets.io/v1
kind: ExternalSecret
metadata:
  name: vault-{{ $name }}
spec:
  refreshInterval: "15s"
  secretStoreRef:
    name: {{ $.Values.general.secretStoreName }}
    kind: SecretStore
  target:
    name: {{ $name }}
  data:
  - secretKey: {{ $config.key }}
    remoteRef:
      key: {{ $.Values.workload.name }}/{{ $name }}
      property: {{ $config.key }}
{{- end }}
```

GitOps Bridge Pattern – Terraform Secret Definition

```
locals {
  workloads = {
    guestbook = {
      name      = "guestbook"
      namespace = "guestbook"
      secrets = {
        random-secret = {
          key = "foo"
          version = 1
        }
      }
    }
  }
```

Let the magic happen



What happens next?

⇢ The project must run anywhere so typically we would configure:
- → Gitlab CI or GitHub Actions
- → Configure JWT based least privilege access to Vault
- → Short-lived K8s credentials using the Vault Authentication Engine

⇢ This way you could achieve a setup where:
- → No static secrets are held in environment or CI variables
- → In a hardened scenario not even your team has ever seen the secrets

⇢ When combined with a mutating webhook solution like bank-vaults/vault-secrets-webhook ...
- → This could even be extended to a solution without K8s secrets
- → … but I must admit this would be harder to integrate with existing charts

What would be alternative Solutions?

⇢ Keep Vault config and Kubernetes objects separate and create random strings manually

⇢ Using an "only GitOps solution" e.g., with Crossplane

⇢ Extending ESO to implement "create if not exists” (= secret generation)

⇢ Depending on the kind of secret you could directly use dynamic secrets from Vault

(Dis-) Advantages

- Combining Terraform and ArgoCD using the best of both worlds
- Compact codebase, in this case one repo
- Potential for a zero-trust solution, if K8s RBAC and Vault Policies are configured strict enough

- Relies on a rather unusual pattern by passing Helm values as base64 through a secret annotation

Wrap-Up

/ Get your own copy

Slides available at:

/ Get your own copy

Code available at:

/ Want to get in touch?

Contact

Feel free to contact to us if you…

- … still have questions
- ... have feedback/suggestions
- … just want to have a chat ☺

Lisa Vossen
System Engineer – Managed Kubernetes

+49 151 53882519
lisa.vossen@sva.de
www.sva.de

Location Köln
Maarweg 165
50825 Köln

Felix Kerker
System Engineer

+49 151 180 530 68
felix.kerker@sva.de
www.sva.de

Location Hannover
Alemannenhof 1
30855 Langenhagen

Leon Krass
Technical Leader HashiCorp Vault

+49 151 53882677
leon.krass@sva.de
www.sva.de

Location Hamburg
Große Elbstraße 273
22767 Hamburg

11
09/2025

Leon Krass, Felix Kerker, Lisa Vossen

Where to Hide Your Dirty (Technical) Secrets in Kubernetes