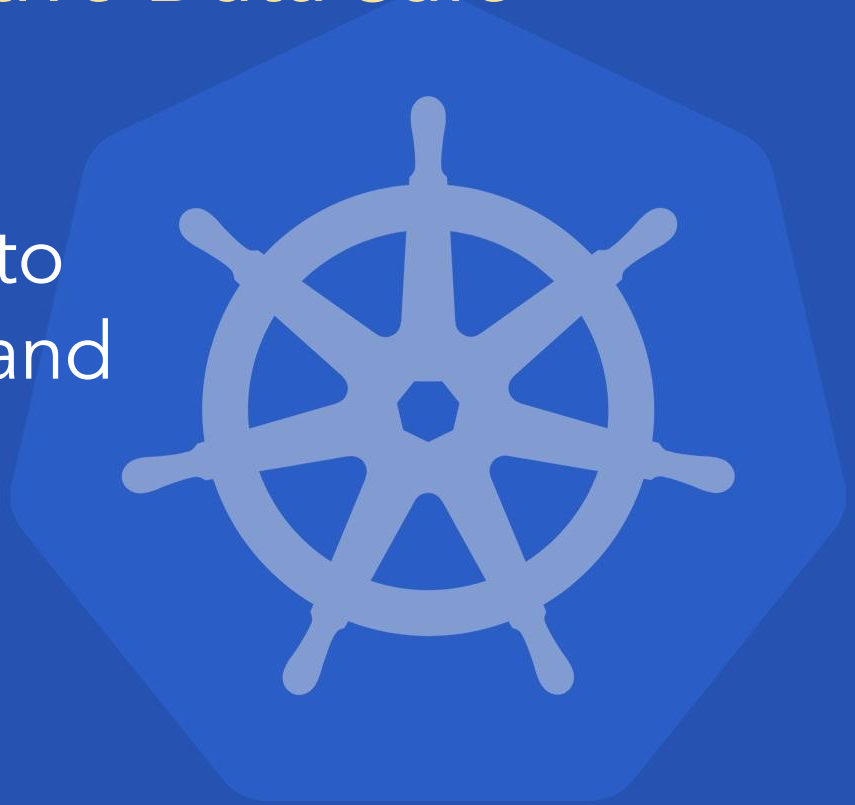
Keeping Your Cloud Native Data Safe



A Common-Sense Guide to Kubernetes, ClickHouse, and Security

Robert Hodges
Altinity Engineering

A brief message from our sponsor…

Robert Hodges

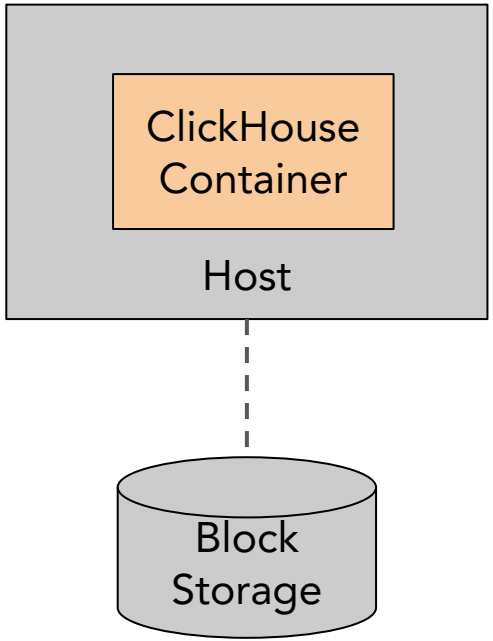
Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Altinity Engineering

Database geeks with centuries of experience in DBMS and applications

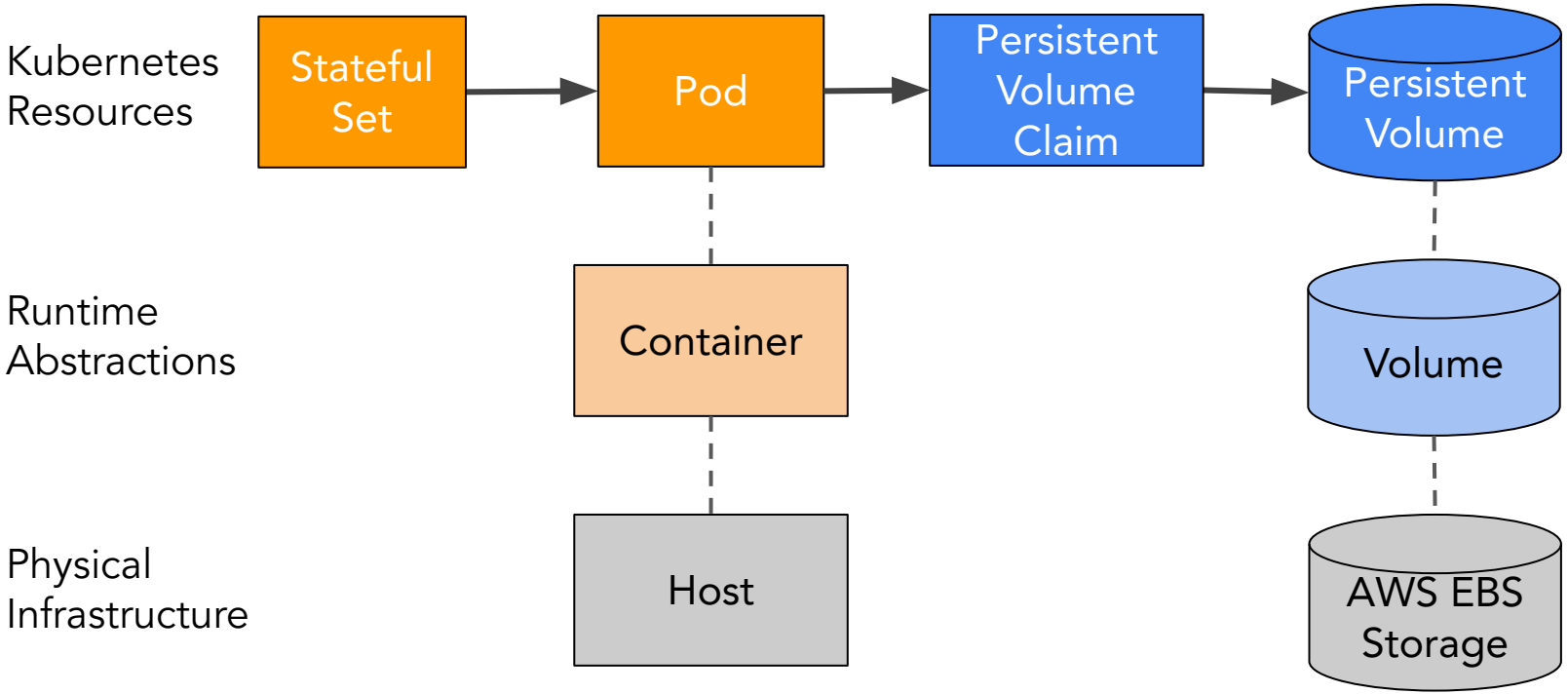
ClickHouse software and services: Altinity.Cloud and Altinity Stable Builds
Authors of Altinity Kubernetes Operator for ClickHouse

What is Kubernetes?

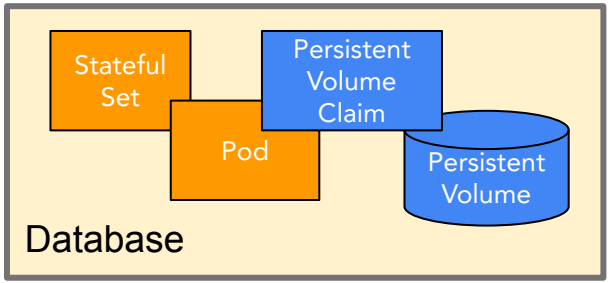


An orchestrator for container-based applications

Kubernetes maps resources to infrastructure

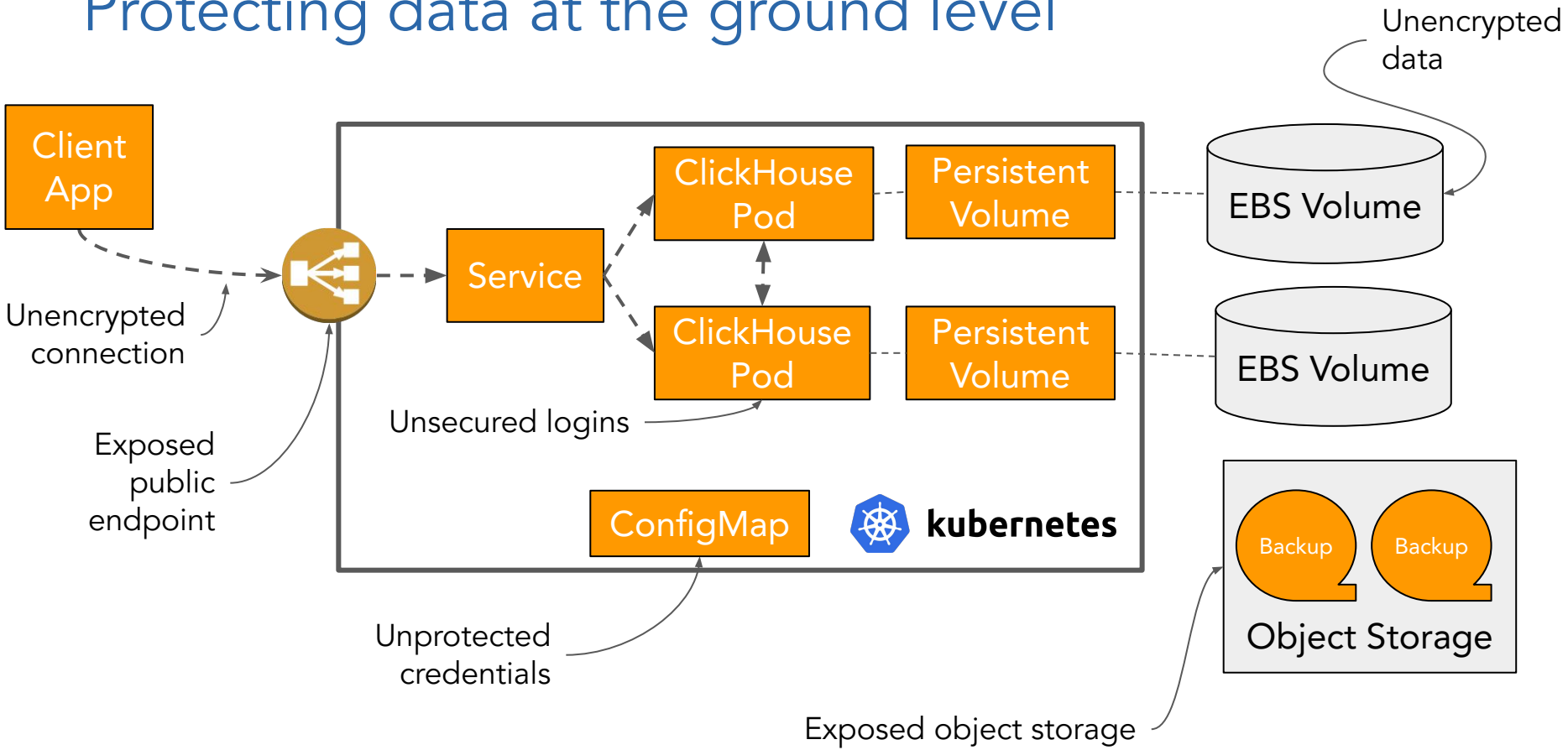


Kubernetes is a great platform for databases!

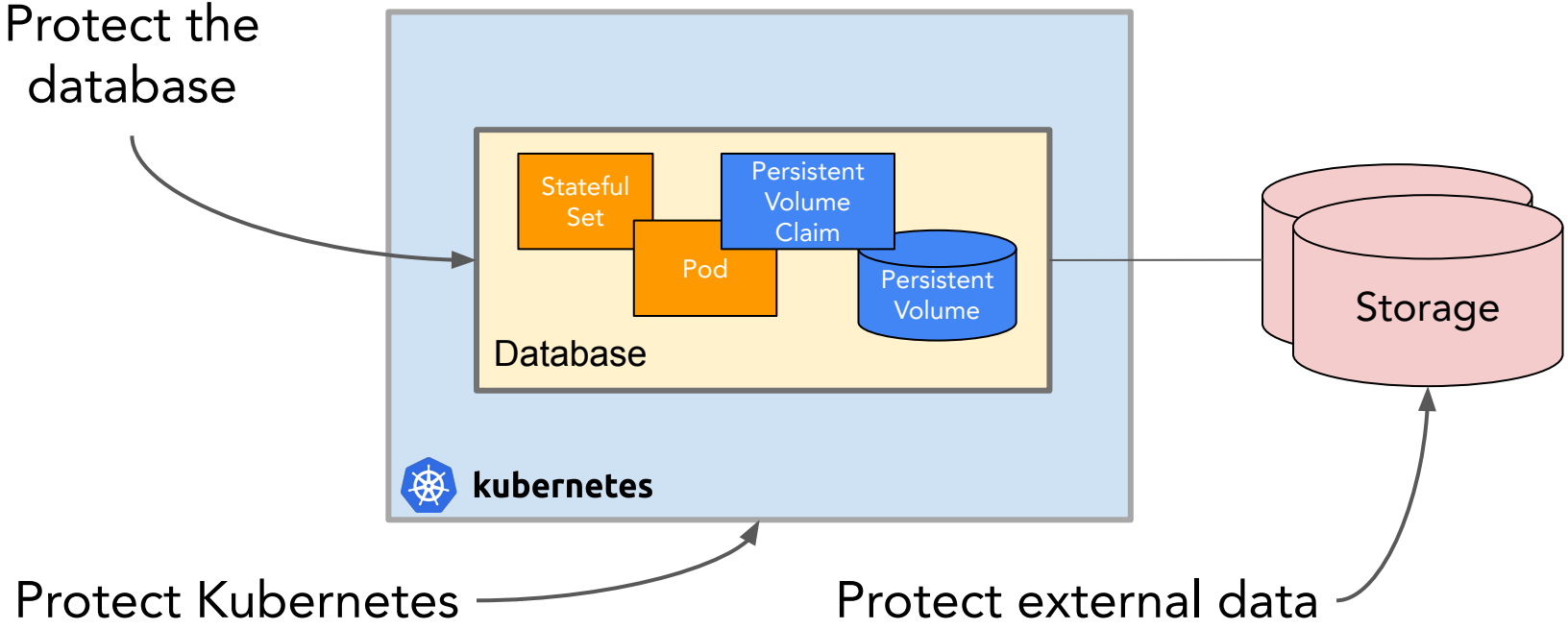


But every silver lining has a cloud…

Protecting data at the ground level

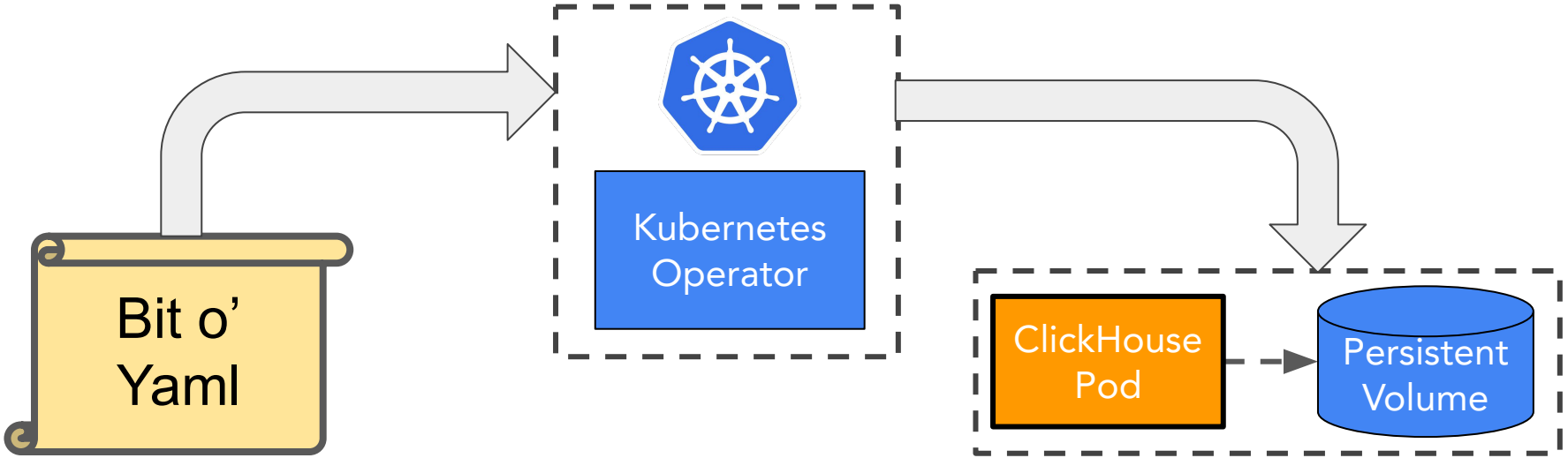


We can split database protection into three parts



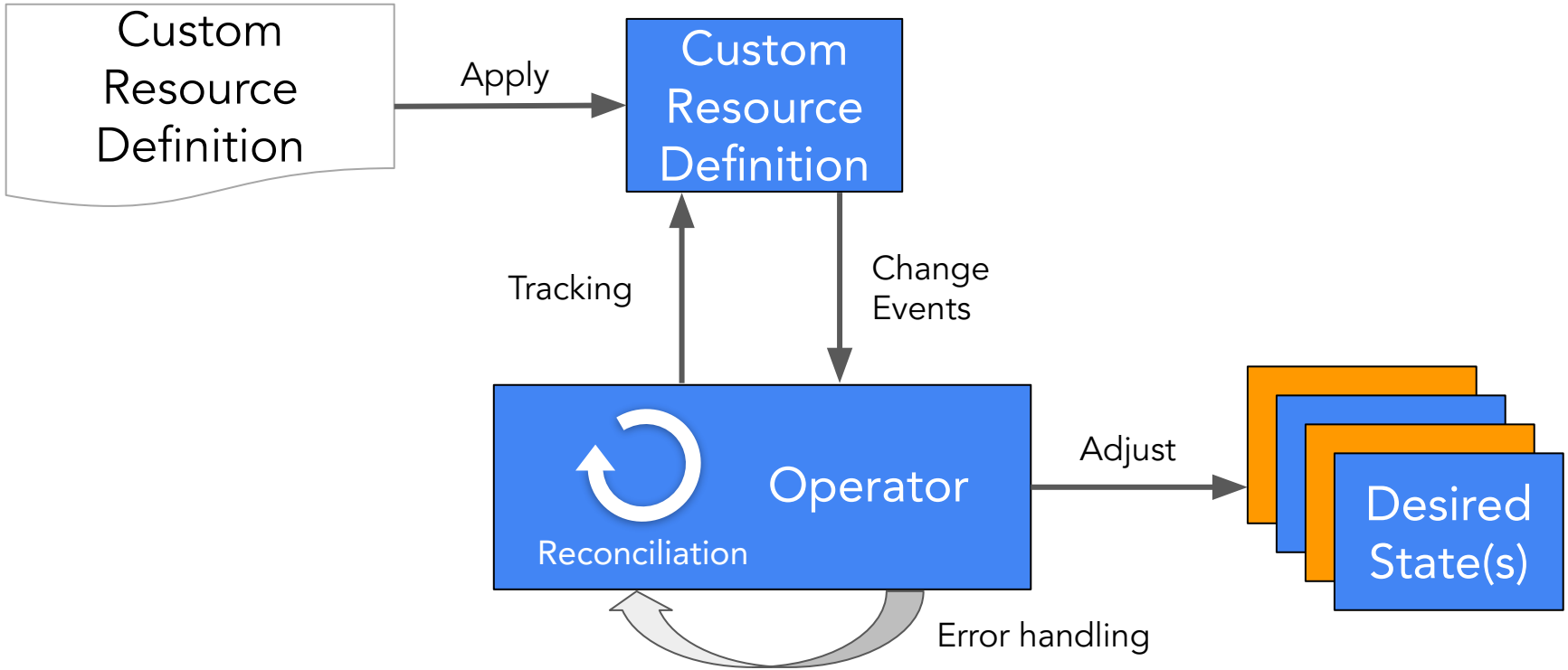
Yikes!
ClickHouse is complicated

Operators reduce databases to a single resource

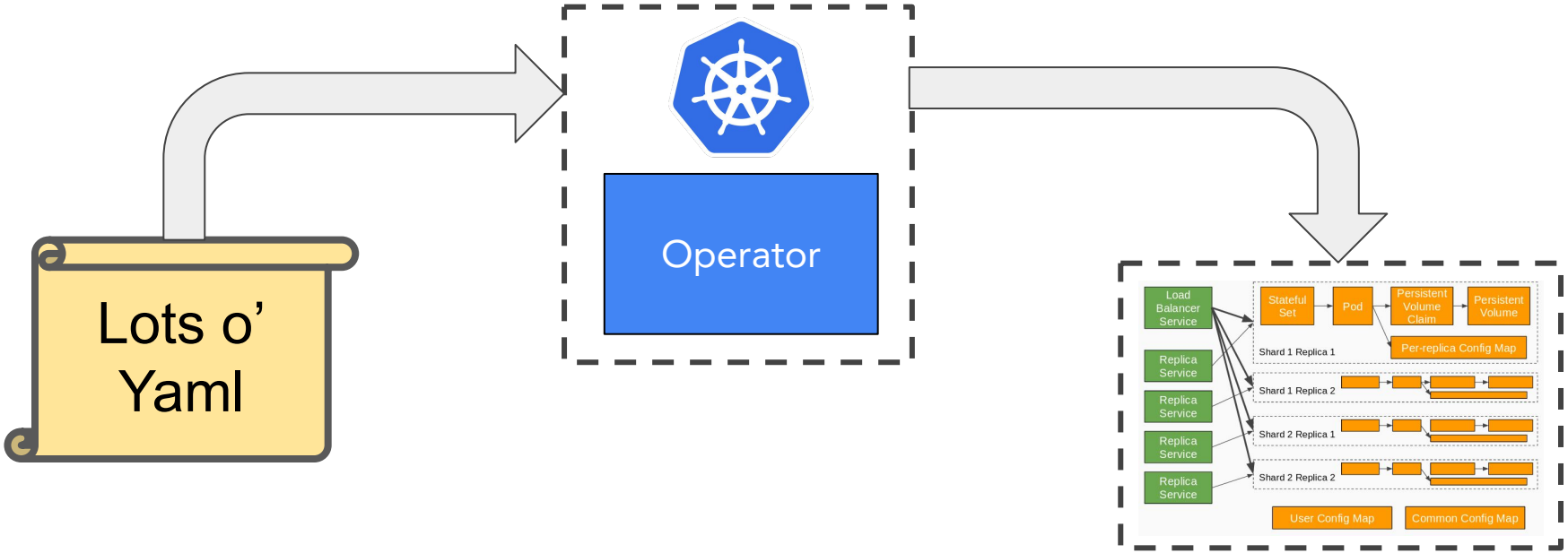


Custom Resource Definition
Aka "CRD"

Operators translate CRDs to a best practice deployment



That's a big win for humans when databases get complex



The operator comes with built-in security features

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"metadata:
  name: "prod"
spec:
  configuration:
    users:
      default/password_sha256_hex: 716b...e448
    clusters:
      - name: "default"
        secret:
          valueFrom:
            secretKeyRef:
              name: "secure-inter-cluster-communications"
              key: "secret"
```

Eliminate empty password for default user ✔

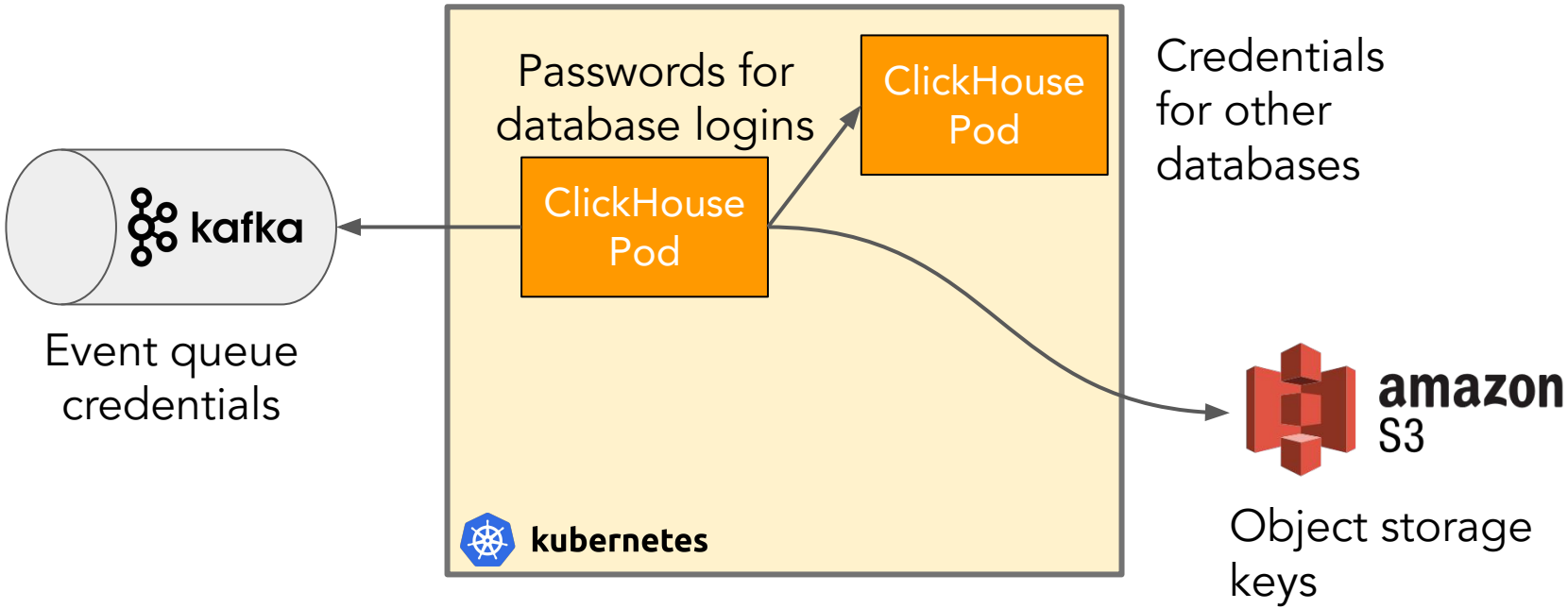
Restrict default user to localhost and cluster IPs ✔

Secure comms with other databases ✔

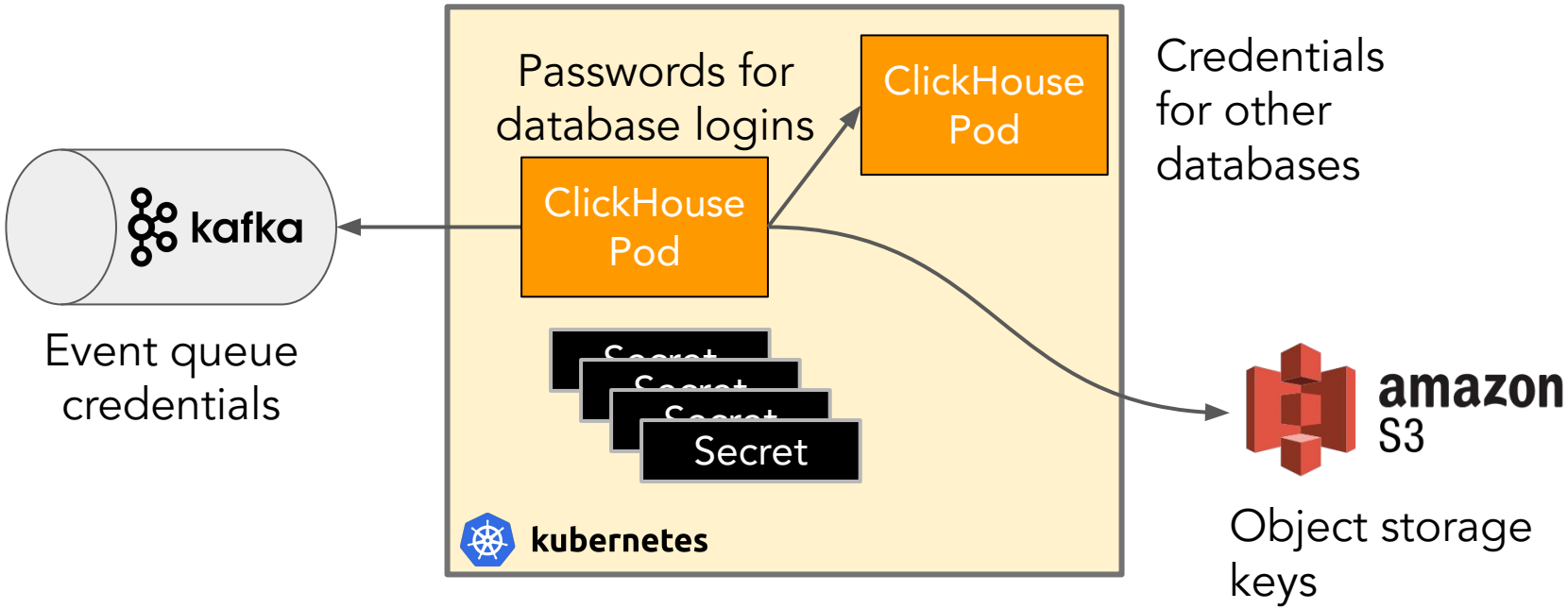
Wait.
What about credentials?

Operators

Credentials are everywhere!!



Secrets transfer safely credentials to pods



The Altinity operator understands Kubernetes secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  root_login: NTgt...
```

You can deliver password to default using special syntax

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"metadata:
  name: "prod"
spec:
  configuration:
    users:
      default/password_sha256_hex: db-passwords/root_login
    clusters:
      - name: "default"
        secret:
          valueFrom:
            secretKeyRef:
              . . .
```

Pod-specific way to set password securely

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  root_login: NTgt...
```

Kubernetes also has general ways to apply secret values

```
spec:
  containers:
  - name: clickhouse
    image: altinity/clickhouse-server:23.3.8.22.altinitystable
    env:
    - name: AWS_ACCESS_KEY_ID
      valueFrom:
        secretKeyRef:
          name: s3-credentials
          key: AWS_ACCESS_KEY_ID
    - name: AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: s3-credentials
          key: AWS_SECRET_ACCESS_KEY
           . . .
```

Simple way to set S3 credentials using secrets

Assign to environmental variable

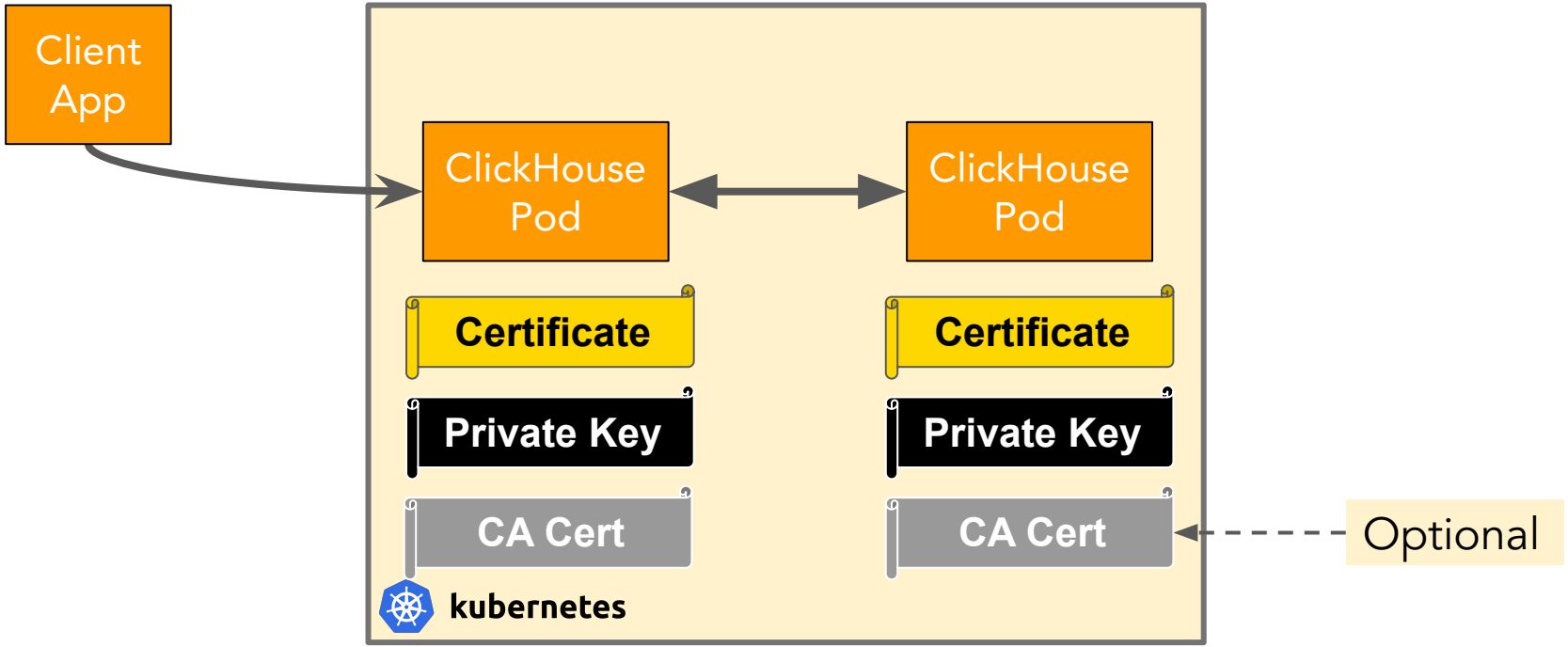
```
apiVersion: v1
kind: Secret
metadata:
  name: s3-credentials
type: Opaque
data:
  AWS_SECRET_ACCESS_KEY: QUtJ...
  AWS_ACCESS_KEY_ID: b00r...
```

Let’s protect connections to the database

Operators

Secrets

TLS encrypts data on connections to/from databases



Configure ports and TLS

```
spec:
  configuration:
    clusters:
      - name: "ch"
        secure: "yes"
        secret:
          auto: "yes"
        . . .
    settings:
      tcp_port: 9000 # keep for localhost
      tcp_port_secure: 9440
      https_port: 8443
    files:
      openssl.xml: |
          <clickhouse>
            <openSSL>
              <server>
                . . .
```

Use TLS-encrypted ports ✔

Specify ports to use ✔

Supply openSSL settings ✔

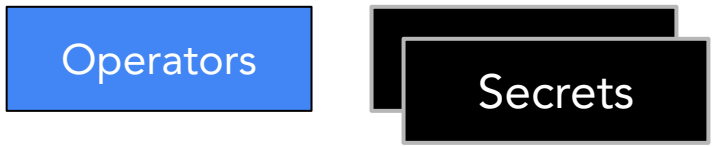
Use secrets to convey the server cert and key

```
spec:
  containers:
  - name: clickhouse
    image: altinity/clickhouse-server:23.3.8.22.altinitystable
    volumeMounts:
    - name: server-crt-volume
      mountPath: "/opt/certs/server.crt"
      subPath: server.crt
    - name: server-crt-volume
      mountPath: "/opt/certs/server.key"
      subPath: server.key
      . . .
  volumes:
    - name: server-crt-volume
      secret:
        secretName: server-certs
          . . .
```

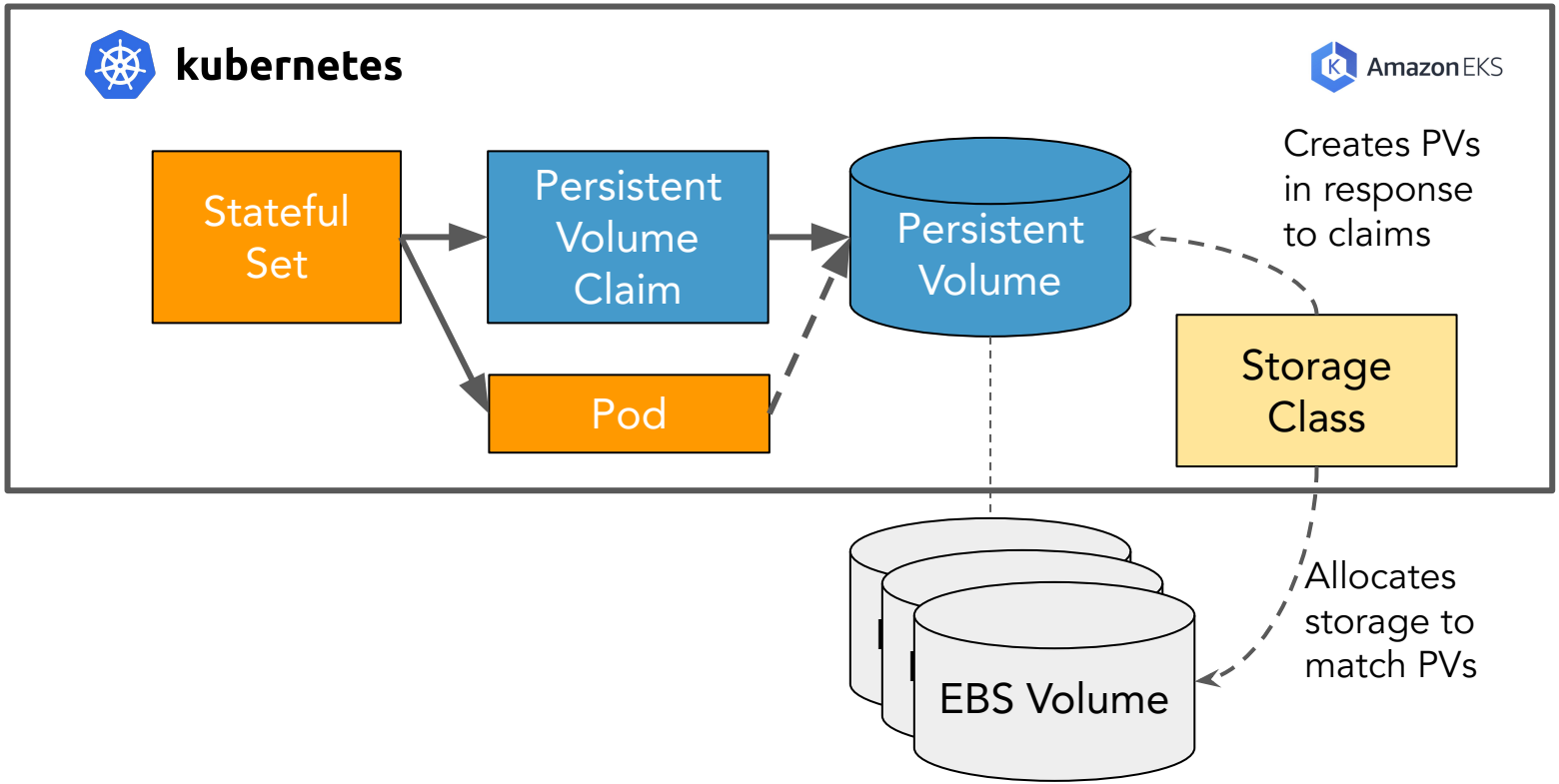
Files mounted automatically ✔

```
apiVersion: v1
kind: Secret
metadata:
  name: server-certs
stringData:
  server.crt: |-
    -----BEGIN CERTIFICATE-----
    ...
  server.key: |-
    -----BEGIN PRIVATE KEy-----
    ...
```

Let's protect stored data, too



How does Kubernetes "make" storage

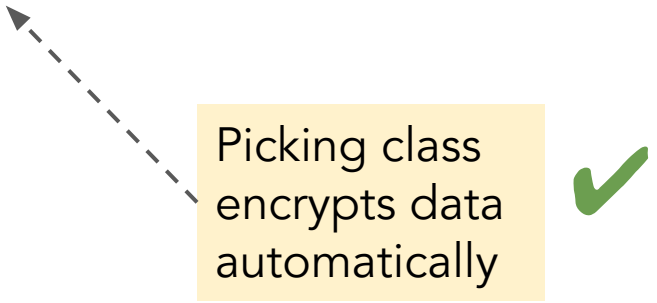


We make a custom storage class that encrypts data

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: encrypted-gp3
provisioner: ebs.csi.aws.com
parameters:
  type: gp3
  fsType: ext4
  encrypted: "true"
allowVolumeExpansion: true
```

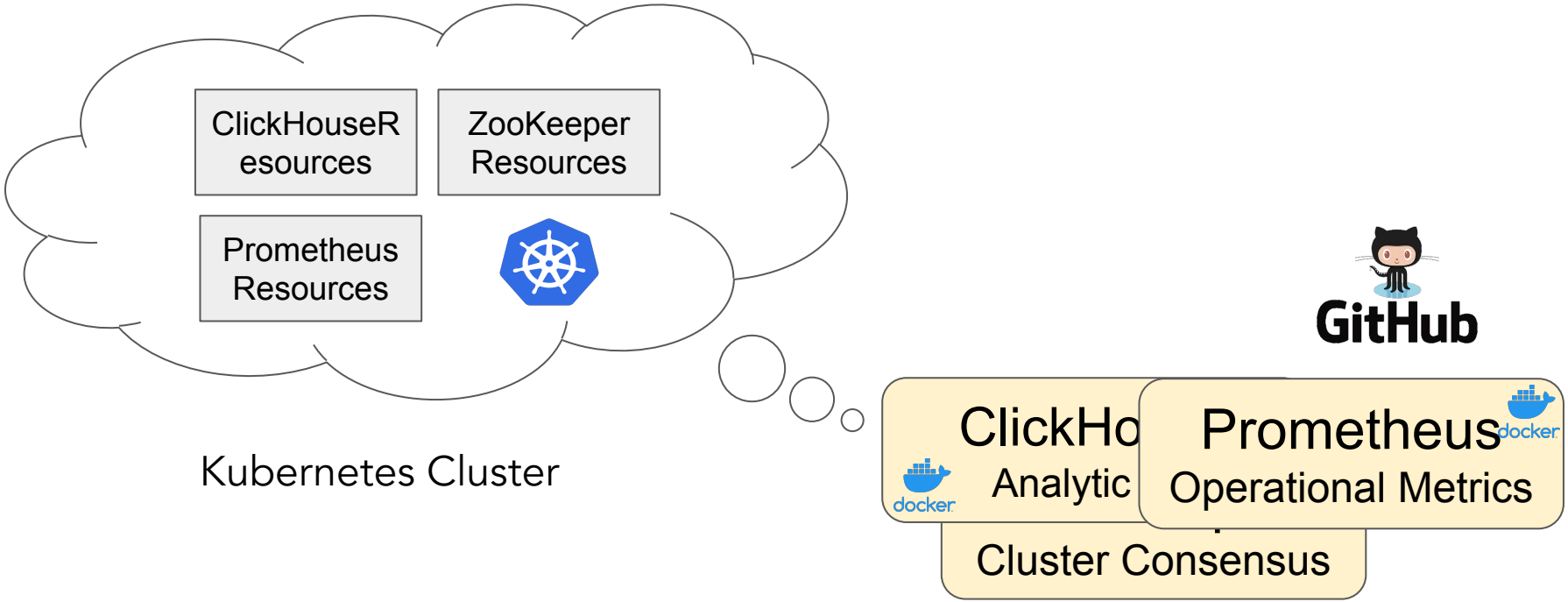
Change your volume claim to use the new class

```
volumeClaimTemplates:
    - name: storage
      # Do not delete PVC if installation is dropped.
      reclaimPolicy: Retain
      spec:
        storageClassName: encrypted-gp3
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 50Gi
```

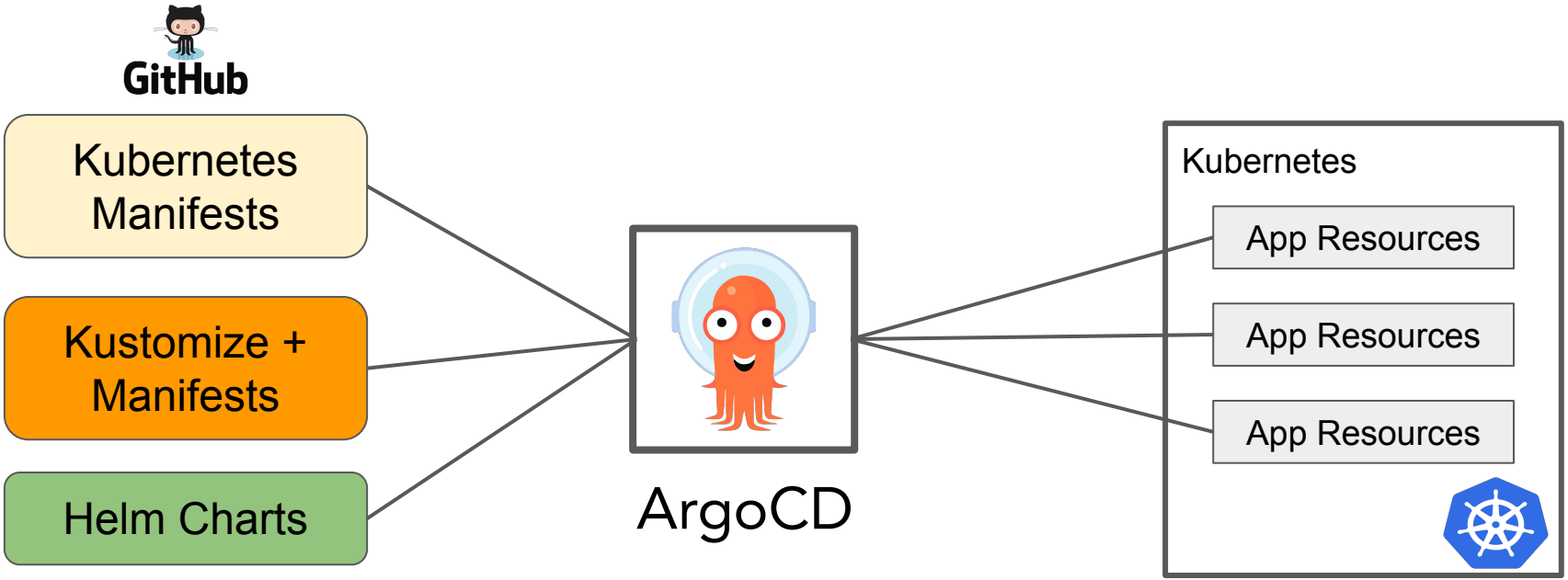


How do we make the process repeatable?

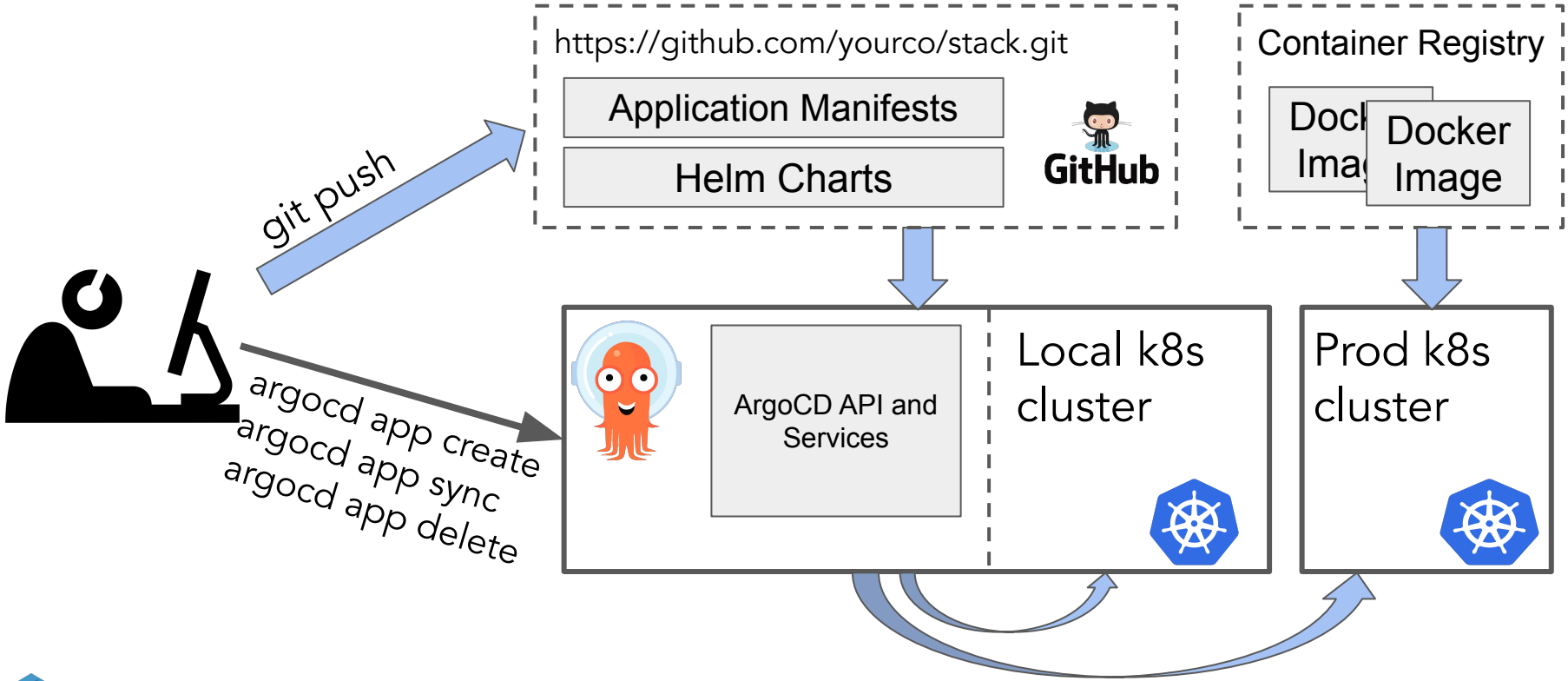
We need a consistent way to deploy databases



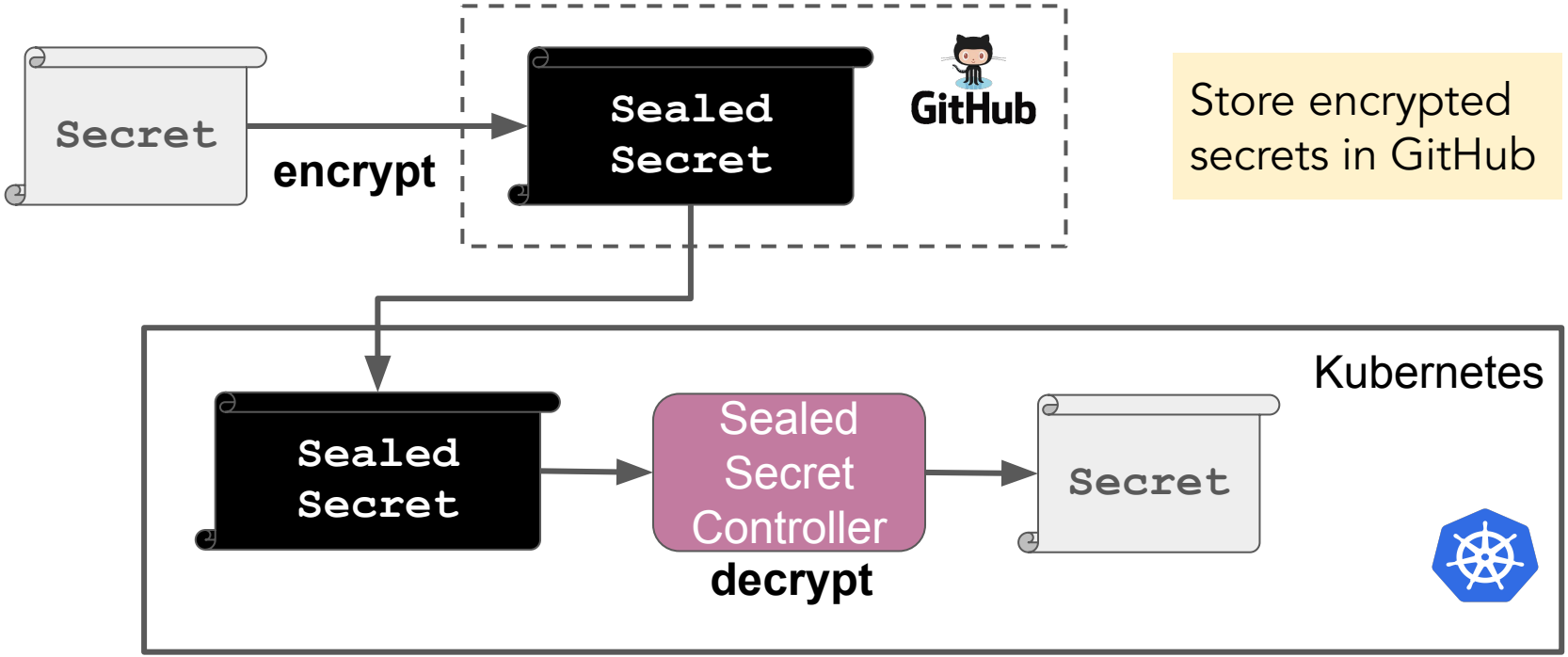
ArgoCD maps deployments from Git[Hub] to K8s



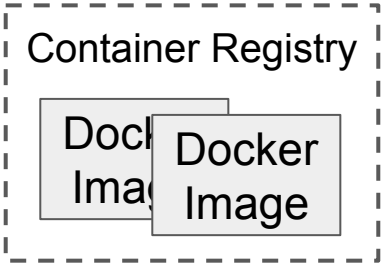
Full GitOps using GitHub, ArgoCD, and Kubernetes



One of many ways to handle credentials



Easy ways to scan containers for security problems



Quick CVE Checking ✔

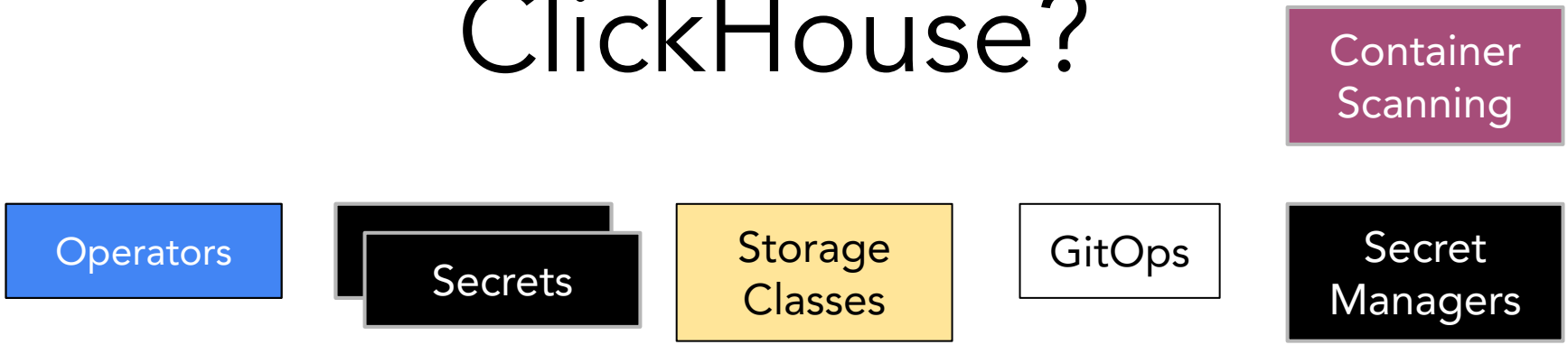
Trivy

Container scanning from command line

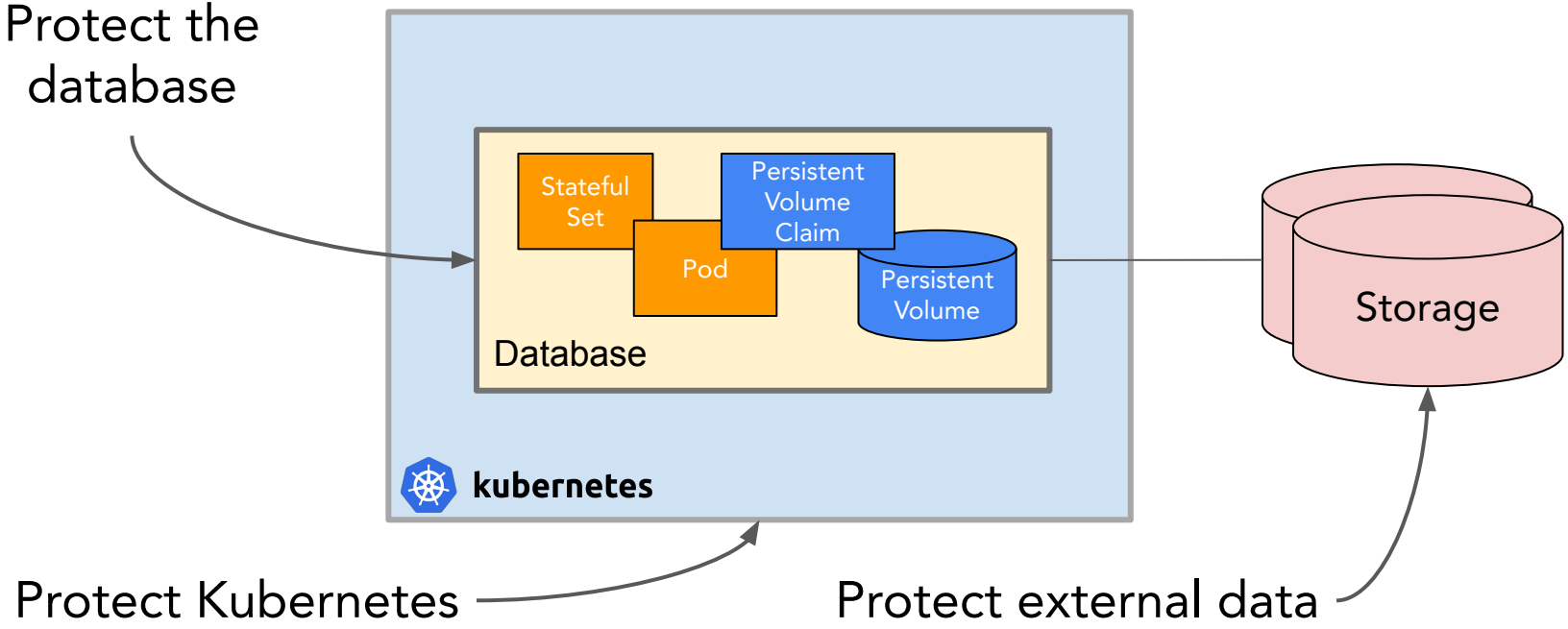
Docker Scout

Scanning in Docker repo(s)

What about “outside” ClickHouse?



Here's our threat model again



Protecting Kubernetes clusters in 2 words or less

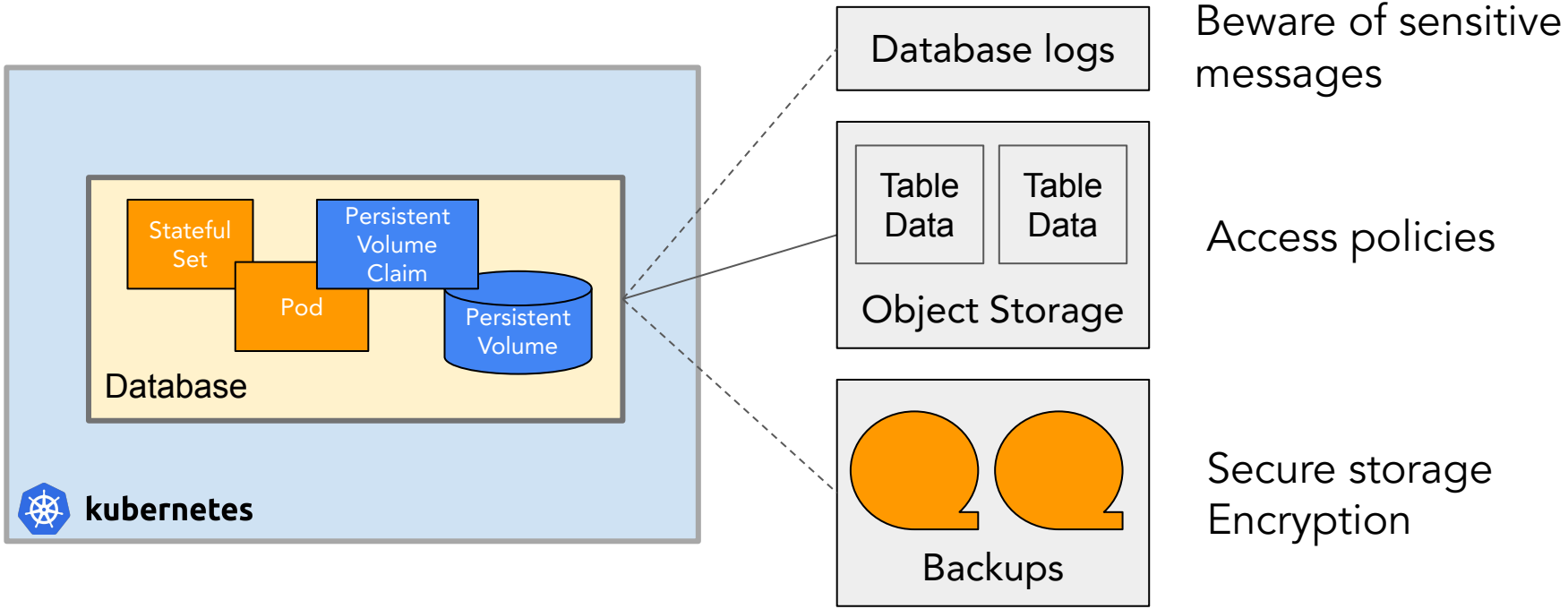
Managed Kubernetes

EKS, GKE, AKS, …

Making your managed Kubernetes safe

- Protect Kubernetes credentials
- Shut off ssh to Kubernetes workers
- Guard network access
  - Private subnets
  - VPC peering / endpoints
- Use vendor scanning tools
  - AWS GuardDuty, AWS Inspector, Google Command Center

External data? Use common sense



Beware of sensitive messages

Access policies

Secure storage
Encryption

How to sleep well with cloud native ClickHouse

Operators

Common Sense

Managed Kubernetes

Secrets

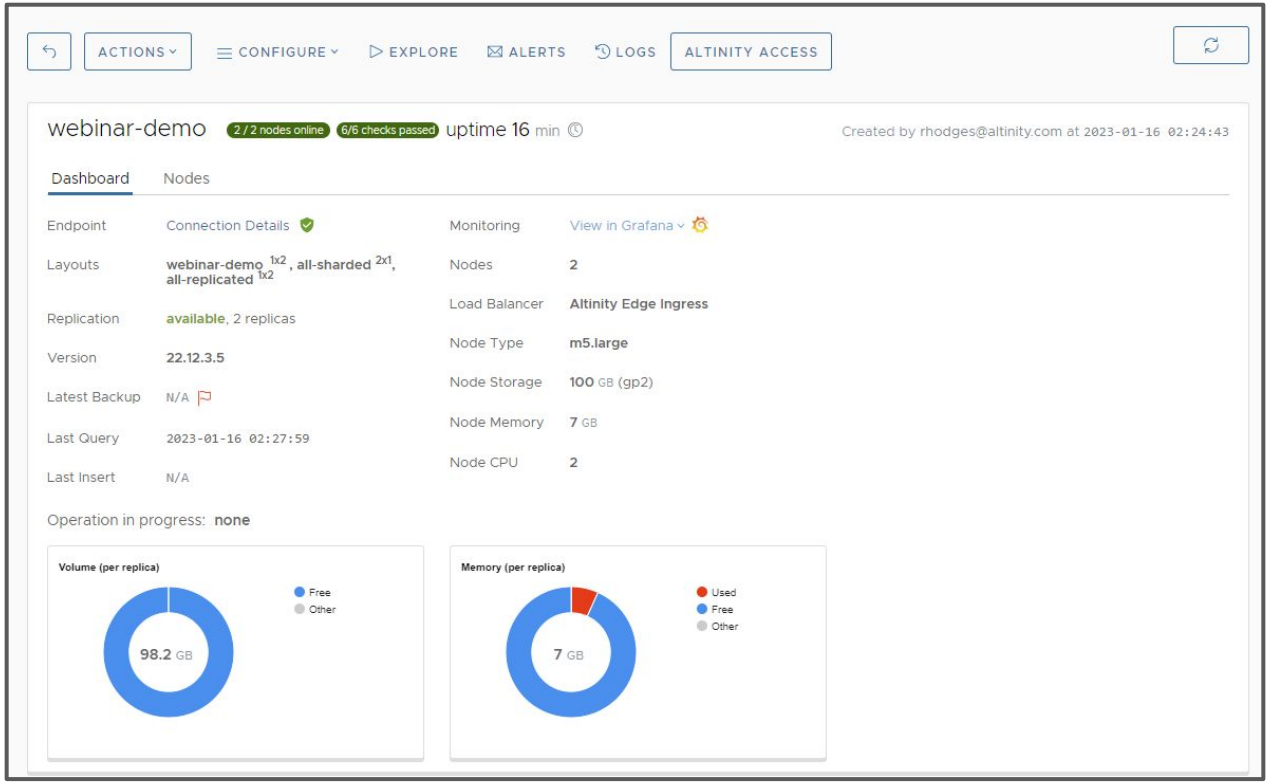
Container Scanning

Storage Classes

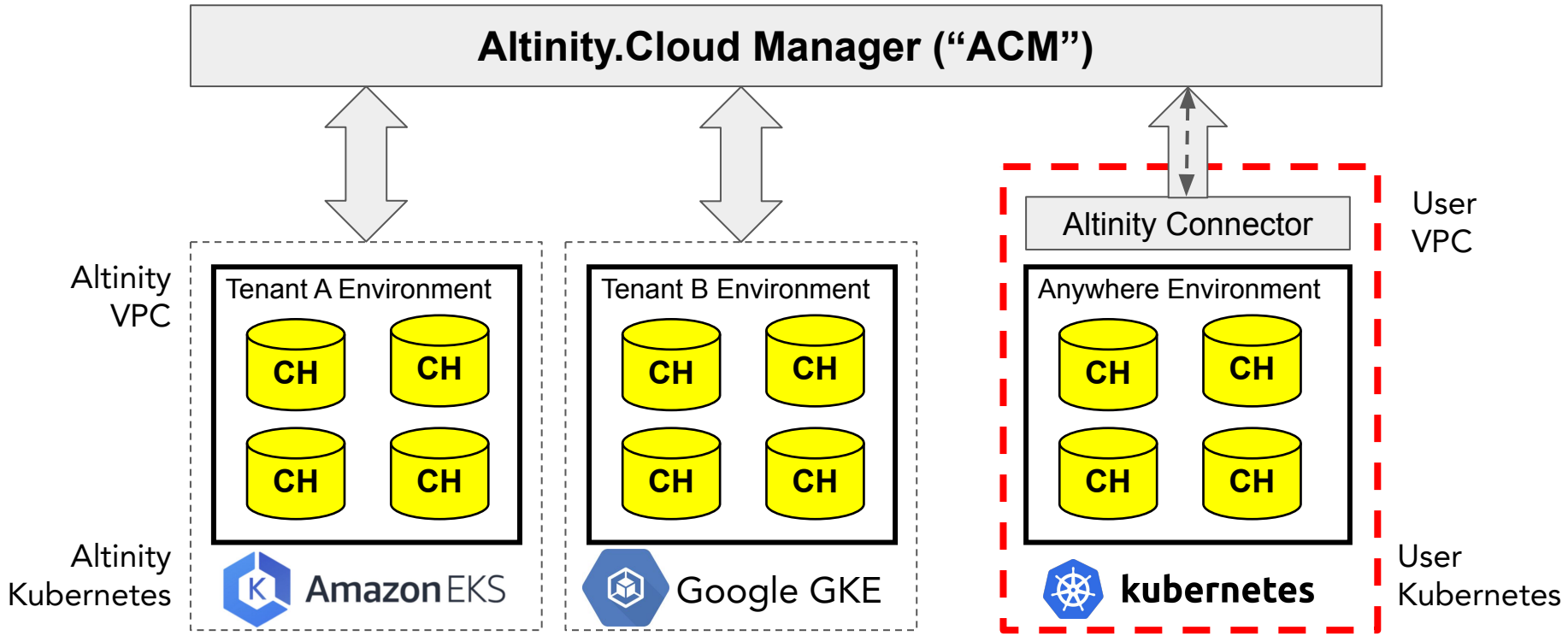
GitOps

Secret Managers

You could save some effort by using Altinity.Cloud!



Altinity.Cloud can manage ClickHouse anywhere!



More information!

- Altinity Kubernetes Operator for ClickHouse on GitHub
  - https://github.com/Altinity/clickhouse-operator
  - Look at clickhouse-operator/docs/security_hardening.md
- Altinity SQL examples project
  - https://github.com/Altinity/clickhouse-sql-examples
- Altinity ArgoCD project
  - https://github.com/Altinity/argocd-examples-clickhouse
- Altinity documention (https://docs.altinity.com)
- Kubernetes docs (https://kubernetes.io/docs/home/)

Thank you and good luck!

Any questions?

Robert Hodges - Altinity
https://altinity.com

Altinity.Cloud
Altinity Stable Builds for ClickHouse
Altinity Kubernetes Operator for ClickHouse