ondat™

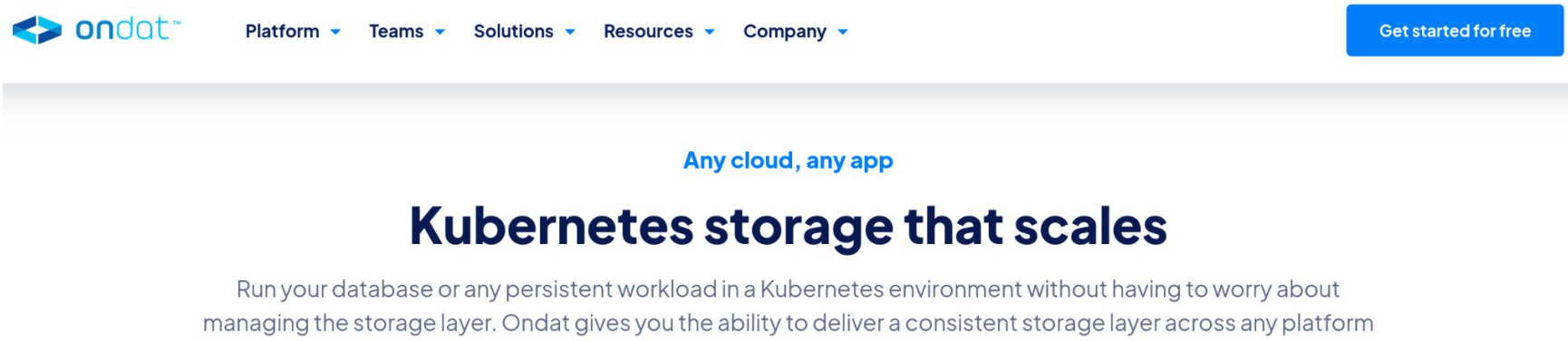
Kubernetes envelope encryption on steroids, with Trousseau

Richa’rd Kova’cs

Boring slide

- At work
  - Staff Kubernetes Engineer
  - @Ondat (former StorageOS)
  - Operator, Scheduler, Controller and Automation
- At IT space
  - Many years of DevOps
  - Who counts years of Go, Java, and so on
    - (22, 9)
  - OSS devotee
  - Known as @mhmxs

Ondat


ondat™
Platform
Teams
Solutions
Resources
Company
Get started for free
Any cloud, any app
Kubernetes storage that scales
Run your database or any persistent workload in a Kubernetes environment without having to worry about managing the storage layer. Ondat gives you the ability to deliver a consistent storage layer across any platform

Secrets are not secrets!

Encoding != Encryption

Solutions

- Leave as it is - done
  - Are you sure?
- Inject by sidecar
  - Features are unique
  - Migration == pain
  - Not transparent, manifests should be different across environments
  - Each pod has its own sidecar
  - Key-rotation usually solved
  - Secret change usually solved
  - Usually both environment variable and file are supported
  - Token is usually a Kubernetes object
- External secret / Injector
  - Features are unique
  - Extra Custom Resource maps secrets
  - Migration == should be pain (some supports multiple providers)
  - Not transparent, manifests should be different across environments
  - Key-rotation usually solved
  - Secret change not solved
  - Secret injection implementation dependent
  - Token is usually a Kubernetes object

Encryption At Rest

Envelope Encryption

Encryption At Rest

- Built-in solution to store encrypted data
- EncryptionConfiguration parsed by Kube API Server
- Any change requires API server restart
- Tokens are located on host filesystem
- Transparent
- Supports several resource types
- Rotation of keys isn't included
- Migration needs manual action
- Encryption done by first provider
- Access Control via RBAC
- **Secrets are still visible inside Kubernetes**

```yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - identity: {}
      - aesgcm:
          keys:
            - name: key1
              secret: c2VjcmV0IGlzIHNlY3VyZQ==
            - name: key2
              secret: dGhpcyBpcyBwYXNzd29yZA==
      - aescbc:
          keys:
            - name: key1
              secret: c2VjcmV0IGlzIHNlY3VyZQ==
            - name: key2
              secret: dGhpcyBpcyBwYXNzd29yZA==
      - secretbox:
          keys:
            - name: key1
              secret: YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY=
```

Encryption At Rest

Providers:

Name	Encryption	Strength	Speed	Key Length	Other Considerations
`identity`	None	N/A	N/A	N/A	Resources written as-is without encryption. When set as the first provider, the resource will be decrypted as new values are written.
`secretbox`	XSalsa20 and Poly1305	Strong	Faster	32-byte	A newer standard and may not be considered acceptable in environments that require high levels of review.
`aesgcm`	AES-GCM with random nonce	Must be rotated every 200k writes	Fastest	16, 24, or 32-byte	Is not recommended for use except when an automated key rotation scheme is implemented.
`aescbc`	AES-CBC with PKCS#7 padding	Weak	Fast	32-byte	Not recommended due to CBC's vulnerability to padding oracle attacks.
`kms`	Uses envelope encryption scheme: Data is encrypted by data encryption keys (DEKs) using AES-CBC with PKCS#7 padding (prior to v1.25), using AES-GCM starting from v1.25, DEKs are encrypted by key encryption keys (KEKs) according to configuration in Key Management Service (KMS)	Strongest	Fast	32-bytes	The recommended choice for using a third party tool for key management. Simplifies key rotation, with a new DEK generated for each encryption, and KEK rotation controlled by the user. Configure the KMS provider

Trousseau

Store and Access your Secrets the Kubernetes native way with any external KMS

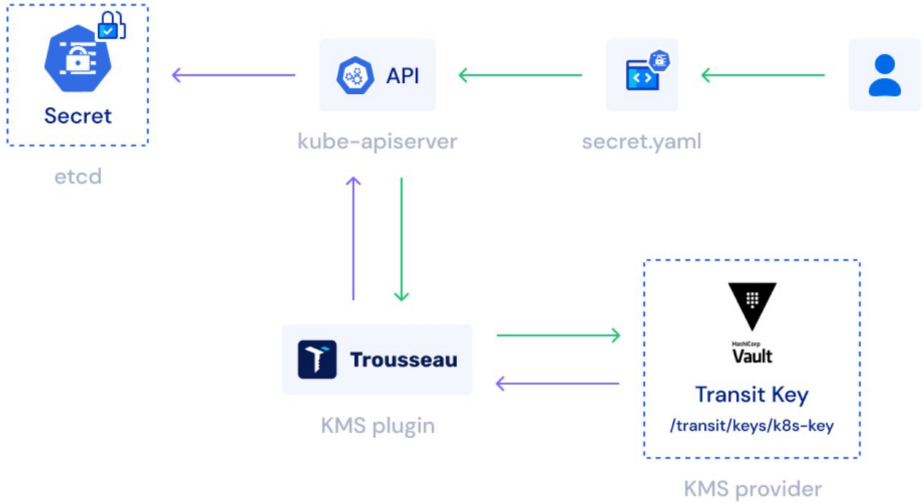
An open-source project based on Kubernetes KMS provider design

Trousseau is an open-source project, based on Kubernetes KMS provider design. It is designed to be a framework for any KMS provider (see release notes).

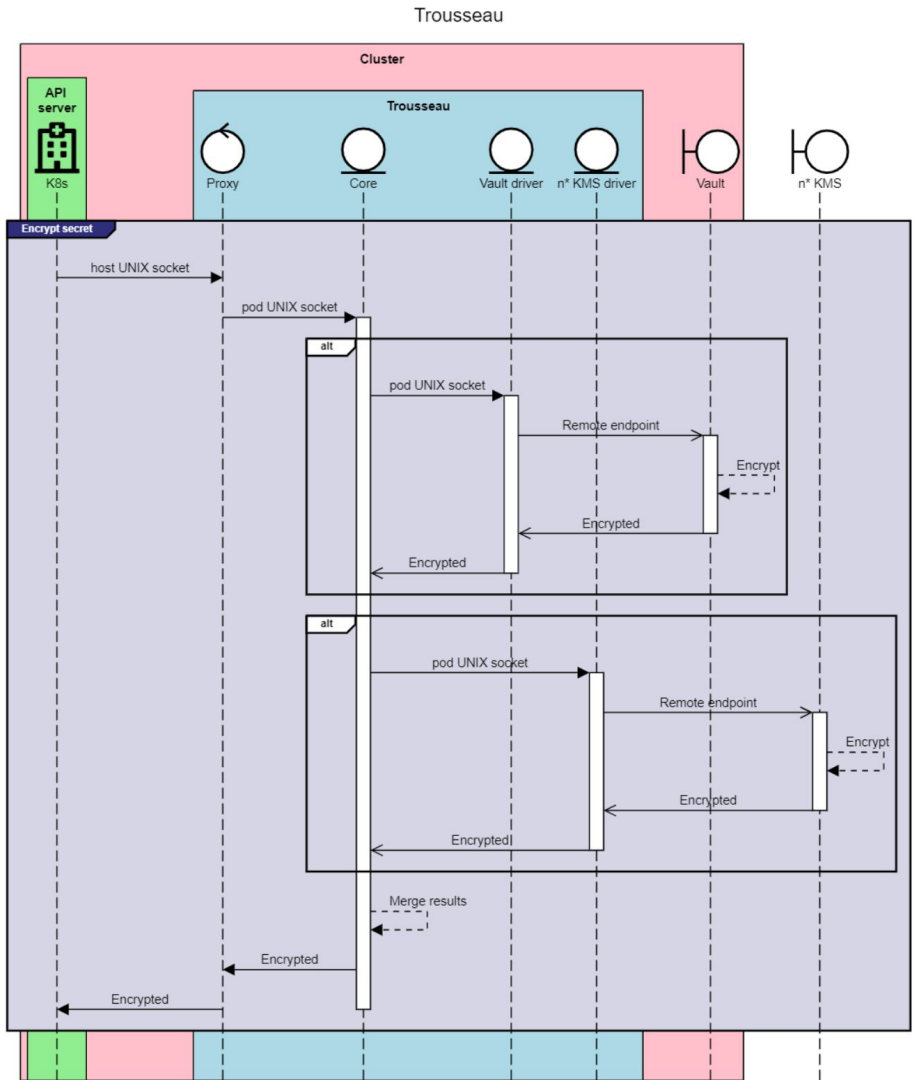
Trousseau

Trousseau Workflow Overview

1. Create a secret
2. kube-api calls trousseau
3. trousseau sends the encryption request to the KMS provider
4. The KMS provider returns the encrypted data to trousseau
5. trousseau sends the encrypted data back to kube-api
6. kube-api stores the encrypted resource in etcd



Trousseau


Trousseau
Cluster
API server
K8s
Trousseau
Proxy
Core
Vault driver
n* KMS driver
Vault
n* KMS
Encrypt secret
host UNIX socket
pod UNIX socket
alt
pod UNIX socket
Remote endpoint
Encrypt
Encrypted
Encrypted
alt
pod UNIX socket
Remote endpoint
Encrypt
Encrypted
Encrypted
Merge results
Encrypted
Encrypted

Trousseau – Benefits

- Solves Unix socket based problems by design
- Configuration change doesn't need restart
- Admins can easily plug and unplug providers (migration still needed)
- It stores every secret in every provider - Easy "HA", falls back if missing
- Supports different types of providers in the same time
- Bring Your Own Provider
- Multiple decryption methods: round-robin, fastest
- Key-rotation is simple and transparent
- **More fault tolerant**

Encryption At Rest – KMS V2

- **Performance**:
  - When starting a cluster, all resources are serially fetched and decrypted to fill the kube-apiserver cache
  - Support for KMS plugins that use a key hierarchy to reduce network requests made to the remote vault
- **Key Rotation**: Extra metadata is now tracked to allow a KMS plugin to communicate what key it is currently using with the kube-apiserver, allowing for rotation without API server restart
- **Health Check & Status**: A dedicated status API is used to communicate the health of the KMS plugin with the API server, eliminate proxy
- **Observability**: To improve observability, a new UID field is included in EncryptRequest and DecryptRequest of the v2 API
- **v2alpha1 introduced at Kubernetes 1.25**

Thank you