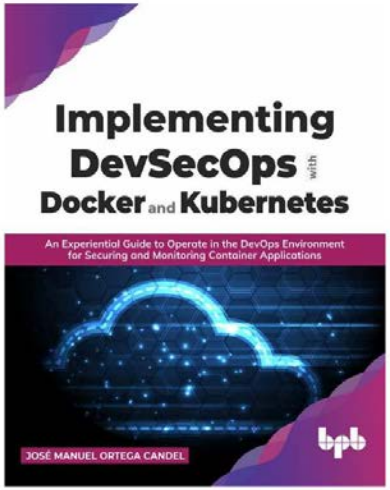
# Sharing secret keys in Docker containers and K8s

José Manuel Ortega
Security researcher

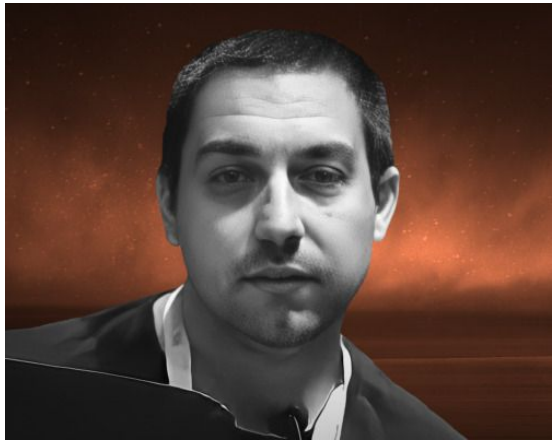



1. Getting Started with DevSecOps
2. Container Platforms
3. Managing Containers and Docker Images
4. Getting Started with Docker Security
5. Docker Host Security
6. Docker Images Security
7. Auditing and Analyzing Vulnerabilities in Docker Containers
8. Managing Docker Secrets and Networking
9. Docker Container Monitoring
10. Docker Container Administration
11. Kubernetes Architecture
12. Kubernetes Security
13. Auditing and Analyzing Vulnerabilities in Kubernetes
14. Observability and Monitoring in Kubernetes

Grafana
RANCHER
docker
podman
clair
Falco
kube-hunter
aqua kube-bench
kubernetes
trivy

SCAN ME

CERT MANAGER



**Jose Manuel Ortega**
Software engineer,
*Freelance*

1. Challenges of security and secret keys in containers
2. Best practices for saving and securing distribution of secrets in Docker Containers
3. Managing secrets in Kubernetes using volumes and sealed-secrets
4. Other tools for distributing secrets in containers

# Challenges of security and secret keys in containers

## The Twelve-Factor App

### Introduction

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

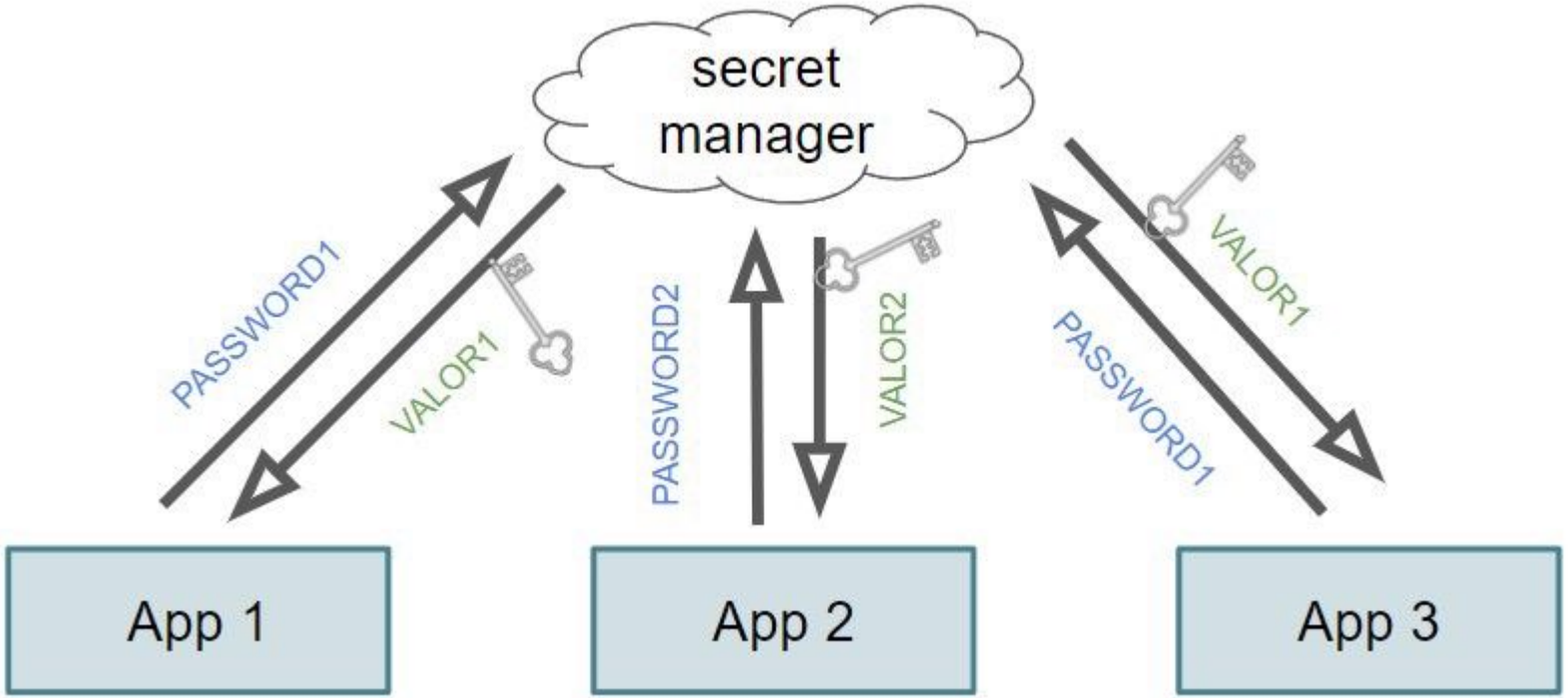
- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

# Challenges of security and secret keys in containers

- Secrets play a critical role in storing sensitive data separately from application code. This includes data such as passwords, hostnames, SSH keys, and more.
- Our application requires a database connection. To do this, it needs a hostname, username, and password. Furthermore, there's a different database server for development, testing, and production.
- With secrets, each environment can provide its own database information to the applications.

# Challenges of security and secret keys in containers

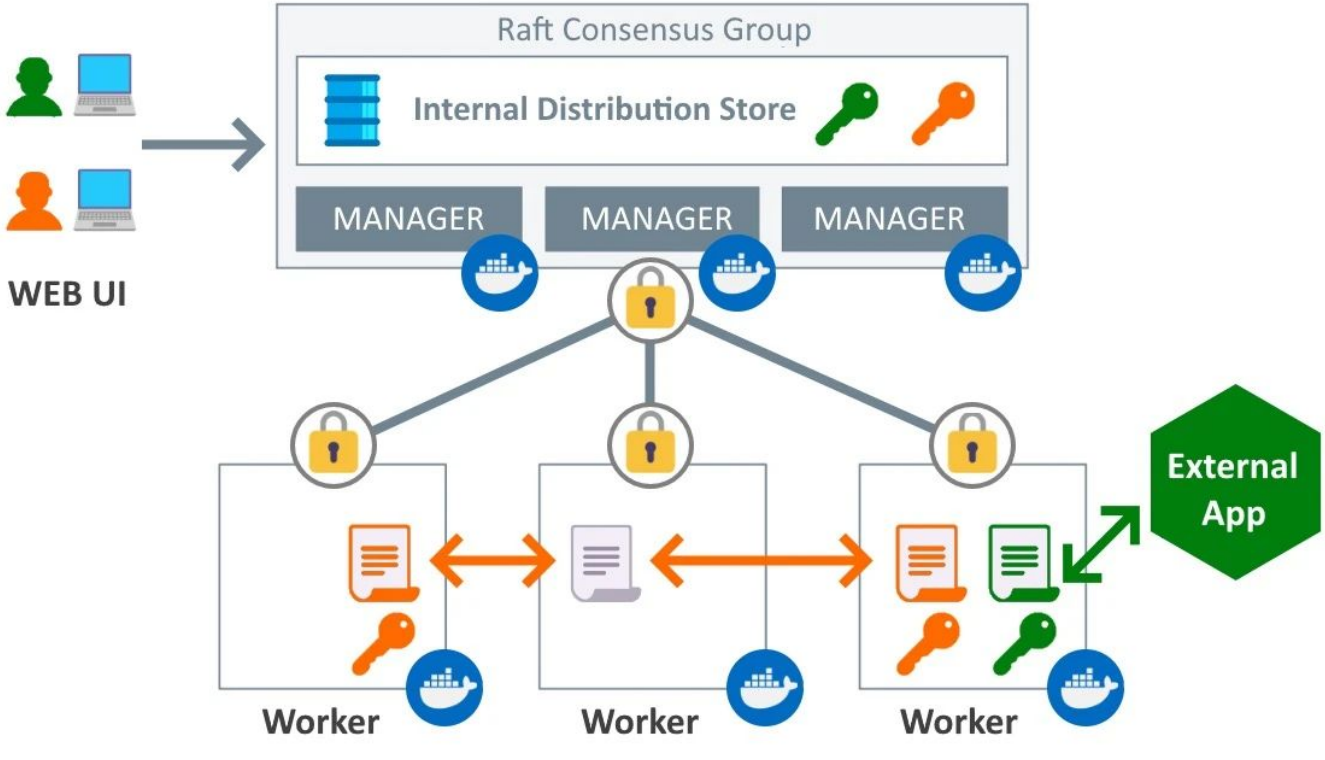


## How Docker manages secrets

Docker's implementation of secrets uses the following features:

- Secrets are created and managed separately from applications.
- Follows principles of least privileged and need-to-know access.
- Flexibility to store a variety of different data types.

# How Docker manages secrets



## How Docker manages secrets

```
$ docker swarm init --advertise-addr
<MANAGER-IP>
```

```
$ docker secret create my_secret
/path/to/secret/file
```

- **/run/secrets/<secret_name>**

# How Docker manages secrets

```
[manager1] (local) root@192.168.0.28 ~
$ docker secret ls
ID        NAME      DRIVER    CREATED   UPDATED
[manager1] (local) root@192.168.0.28 ~
$ printf "THis is a secret" | docker secret create my_secret_data -
crbbfdbwqrsz2e9r4x5fg2tjx
[manager1] (local) root@192.168.0.28 ~
$ docker secret ls
ID                          NAME             DRIVER    CREATED         UPDATED
crbbfdbwqrsz2e9r4x5fg2tjx   my_secret_data             7 seconds ago   7 seconds ago
[manager1] (local) root@192.168.0.28 ~
$ docker service create --name redis --secret my_secret_data redis:alpine
xh8gcscx76v7mn0m4cr9hqbr1
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

# How Docker manages secrets

```
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS      NAMES
ec67a589d92e   redis:alpine   "docker-entrypoint.s…"   21 minutes ago   Up 21 minutes   6379/tcp   redis.1.2bbxr
a2ya8t0vu11a18kzcywz
[worker1] (local) root@192.168.0.27 ~
$ docker container exec $(docker ps --filter name=redis -q) ls -l /run/secrets
total 4
-r--r--r--    1 root     root            16 Sep 26 17:21 my_secret_data
[worker1] (local) root@192.168.0.27 ~
$ docker container exec $(docker ps --filter name=redis -q) cat /run/secrets/my_secret_data
THis is a secret[worker1] (local) root@192.168.0.27 ~
```

# How Docker manages secrets

```
$ docker secret rm my_secret_data
Error response from daemon: rpc error: code = InvalidArgument desc = secret 'my_secret_data' is in use by the following service: redis
[manager1] (local) root@192.168.0.28 ~
$ docker service update
"docker service update" requires exactly 1 argument.
See 'docker service update --help'.

Usage:  docker service update [OPTIONS] SERVICE

Update a service
[manager1] (local) root@192.168.0.28 ~
$docker service update --secret-rm my_secret_data redis
redis
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

# How Docker manages secrets

```
$ docker service ps redis
ID             NAME          IMAGE          NODE       DESIRED STATE   CURRENT STATE            ERROR     PORTS
ycdjxiscr8w5   redis.1       redis:alpine   manager1   Running         Running 3 minutes ago
2bbxra2ya8t0    \_ redis.1   redis:alpine   worker1    Shutdown        Shutdown 3 minutes ago
[manager1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS      NAMES
a05b863c117a   redis:alpine   "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes   6379/tcp   redis.1.ycdjxis
cr8w5d4p513mhiadwf
[manager1] (local) root@192.168.0.28 ~
$ docker container exec -it $(docker ps --filter  name=redis -q) cat /run/secrets/my_secret_data
cat: can't open '/run/secrets/my_secret_data': No such file or directory
[manager1] (local) root@192.168.0.28 ~
```

# Best practices for saving and securing distribution of secrets in Docker Containers

```
$ docker service rm redis
redis
[manager1] (local) root@192.168.0.28 ~
$ docker secret rm my_secret_data
my_secret_data
[manager1] (local) root@192.168.0.28 ~
$ docker secret ls
ID          NAME      DRIVER    CREATED    UPDATED
[manager1] (local) root@192.168.0.28 ~
```

$ docker secret rm my_secret


**Best practices for saving and securing distribution of secrets in Docker Containers**

```
$ docker service create
  --name my_app
  --secret
source=my_secret,target=/different/path/to/secret/file,mode
=0400
```

# Best practices for saving and securing distribution of secrets in Docker Containers

```
version: '3.1'
services:
  my_app:
    image: my_app:latest
    secrets:
     - my_external_secret
     - my_file_secret
secrets:
  my_external_secret:
    external: true
  my_file_secret:
    file: /path/to/secret/file.txt
```


**Best practices for saving and securing distribution of secrets in Docker Containers**

```
$ docker stack deploy -c docker-compose.yml secrets1
Creating service secrets1_viewer

$ docker logs $(docker ps -aqn1 -f status=exited)
my_secret
```

# Managing secrets in Kubernetes

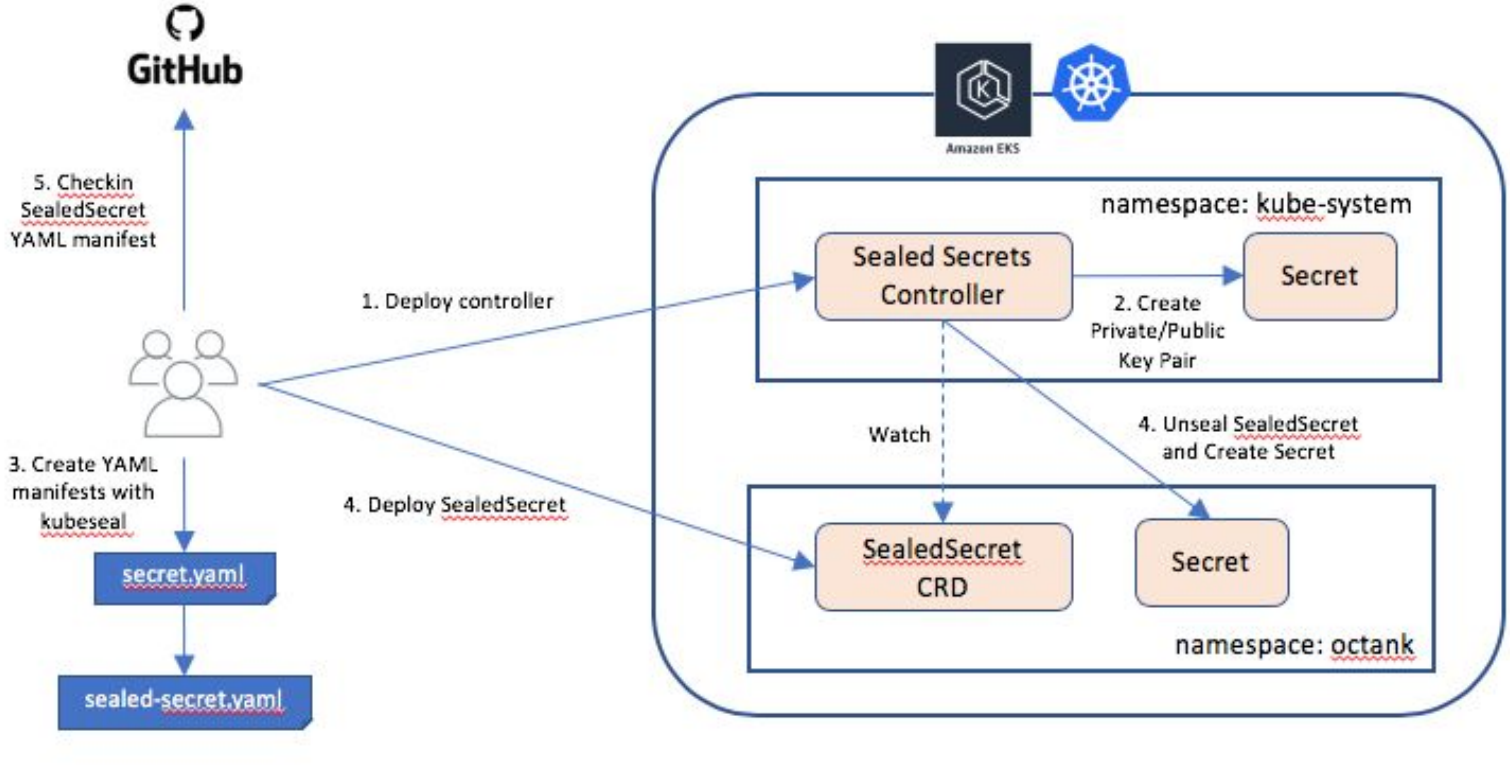
```
→ raw-manifests git:(master) ✗ cat secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: anNtaXRo
  password: bXlzZWNyZXRwYXNzd29yZA==%
→ raw-manifests git:(master) ✗ kubectl create -f secret.yaml
```

```
→ raw-manifests git:(master) ✗ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-hsvnc   kubernetes.io/service-account-token   3
my-secret             Opaque                                2
→ raw-manifests git:(master) ✗
```

# Managing secrets in Kubernetes using volumes

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-pod
spec:
  containers:
  - name: express-test
    image: lukondefmwila/express-test:latest
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/config/secret
  volumes:
  - name: secret-volume
    secret:
      secretName: my-secret
```

# Managing secrets in Kubernetes using sealed-secrets



# Managing secrets in Kubernetes using sealed-secrets

```
apiVersion: v1

kind: Secret

metadata:

  name: my-secret

type: Opaque

data:

  username: dXNlcg==

  password: cGFzc3dvcmQ=
```

# Managing secrets in Kubernetes using sealed-secrets

```
kubeseal --cert=public-key-cert.pem --format=yaml < secret.yaml > sealed-secret.yaml
```

- https://github.com/bitnami-labs/sealed-secrets/releases

| | | |
|---|---|---|
| kubeseal-0.18.5-darwin-amd64.tar.gz | 17.8 MB | 8 days ago |
| kubeseal-0.18.5-darwin-amd64.tar.gz.sig | 96 Bytes | 8 days ago |
| kubeseal-0.18.5-darwin-arm64.tar.gz | 17.1 MB | 8 days ago |
| kubeseal-0.18.5-darwin-arm64.tar.gz.sig | 96 Bytes | 8 days ago |
| kubeseal-0.18.5-linux-amd64.tar.gz | 17.9 MB | 8 days ago |

# Managing secrets in Kubernetes using sealed-secrets

```
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  creationTimestamp: null
  name: my-secret
  namespace: default
spec:
  encryptedData:
    password: AgBvA5WMunlZ5rF9...
    username: AgCCo8eSORsCbeJSoRs/...
```

# Managing secrets in Kubernetes using sealed-secrets

## $ kubectl apply -f sealed-secret.yaml

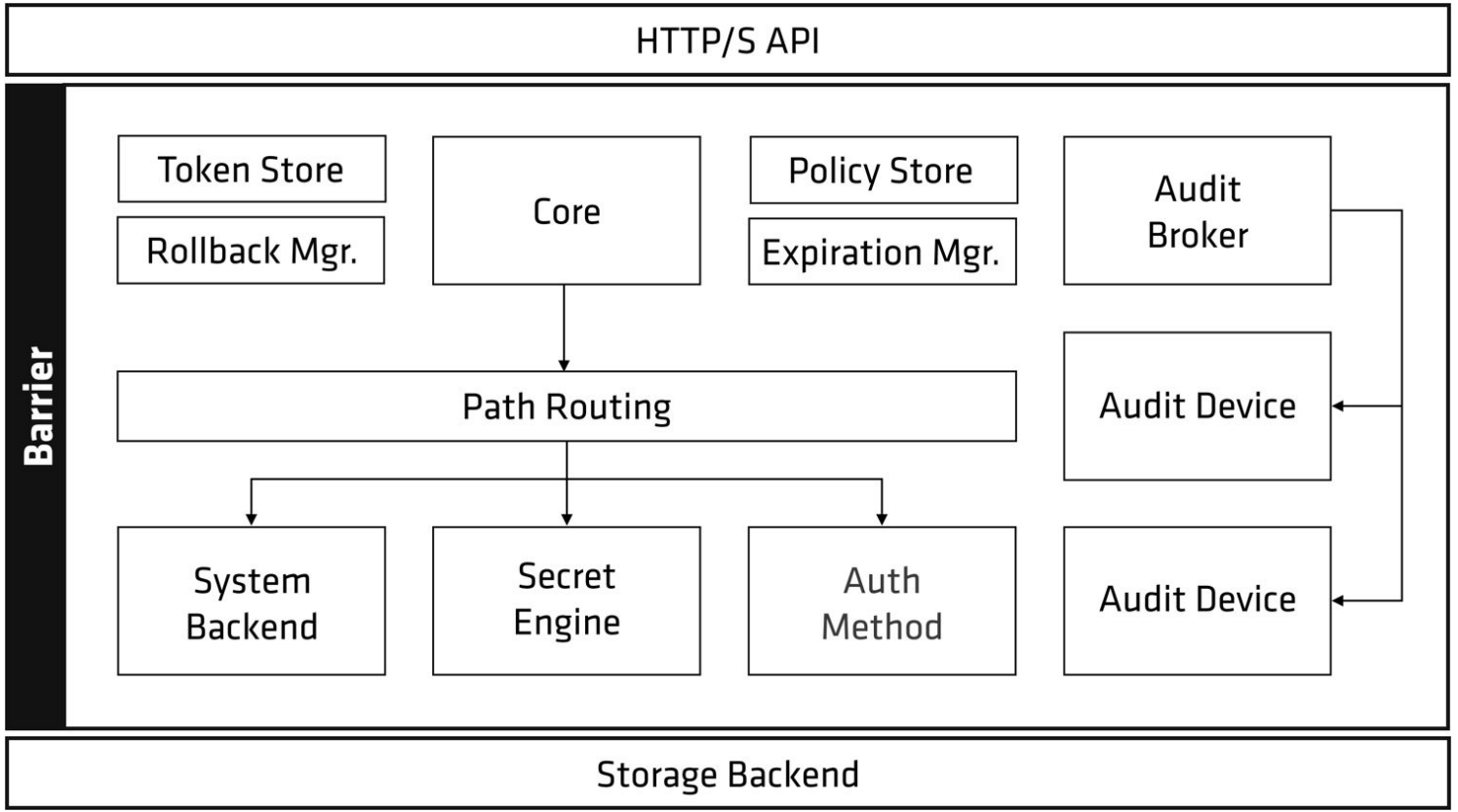
```
Describe(default/my-secret)
Name:         my-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  8 bytes
username:  4 bytes
```

<secret> <describe>

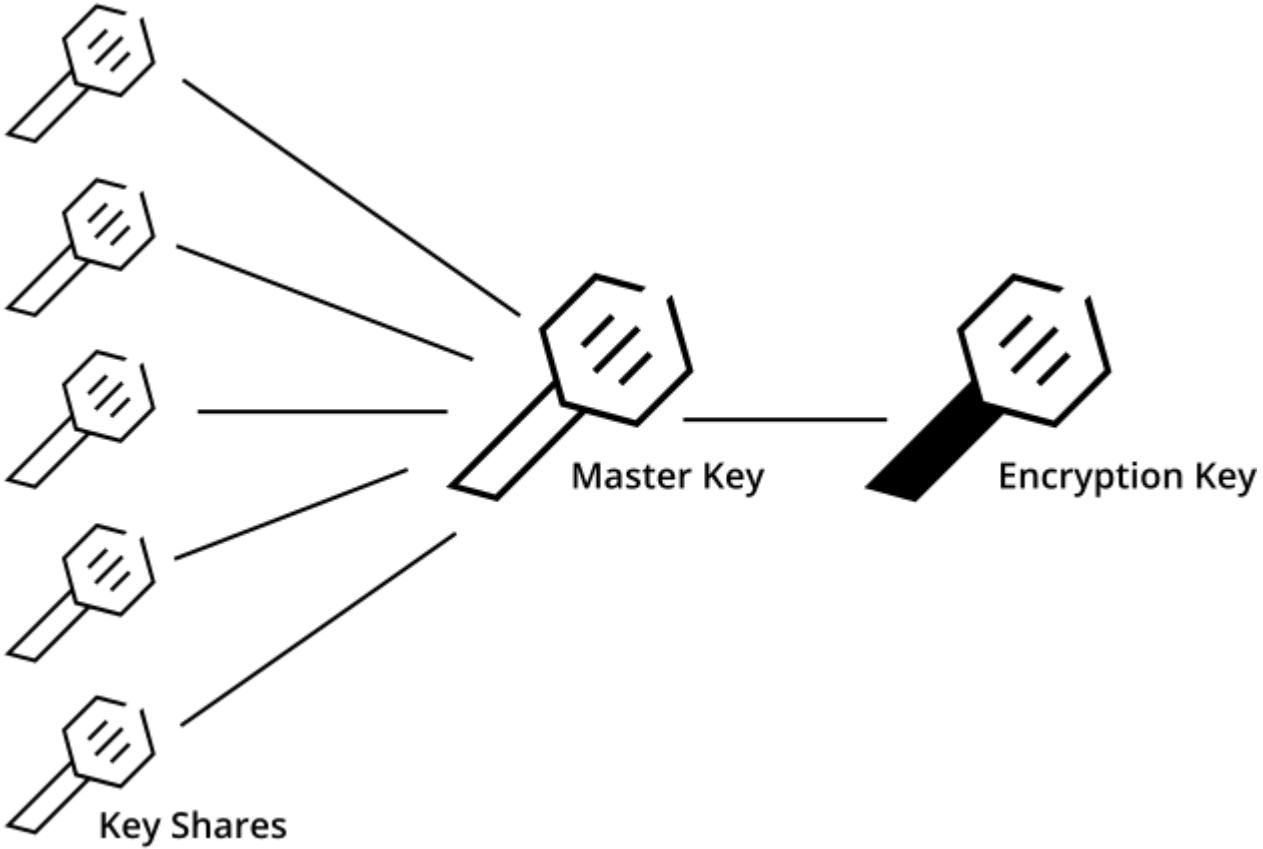
## Other tools for distributing secrets in containers

- Hashicorp Vault
- Keywhiz
- Akeyless Vault
- Cloud Provider solutions (AWS Secrets Manager, GCP Secret Manager)

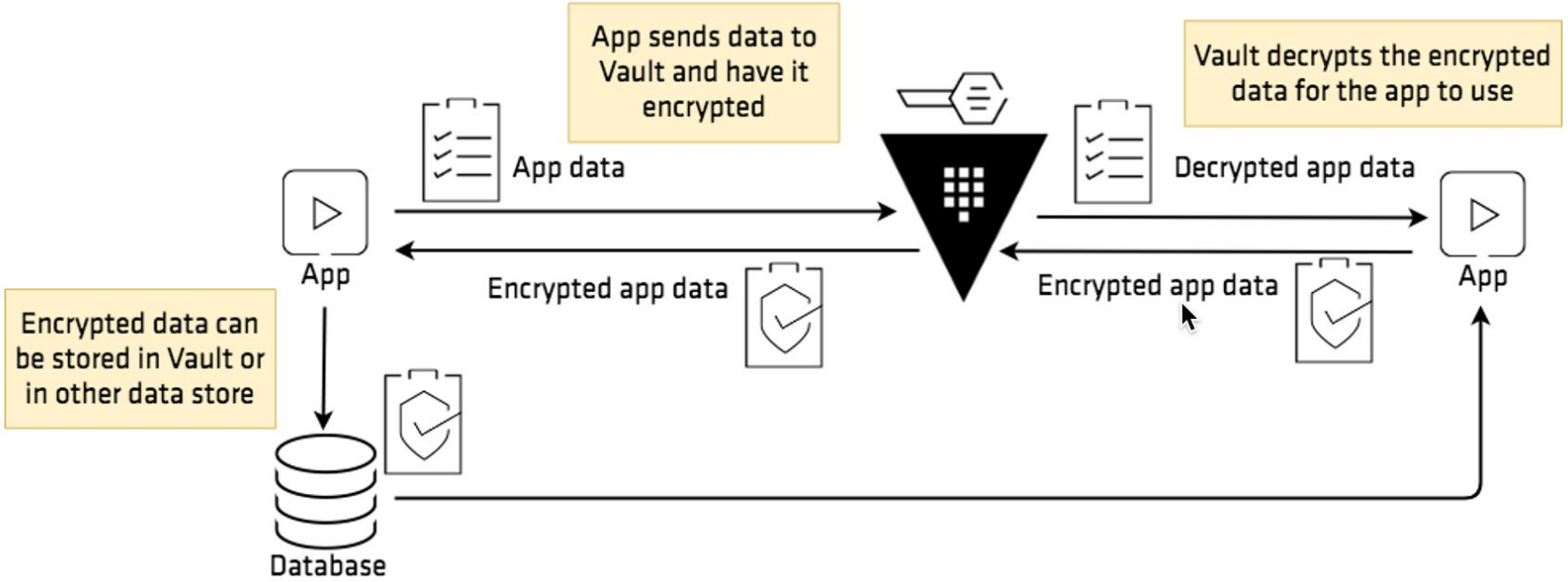

# Hashicorp Vault



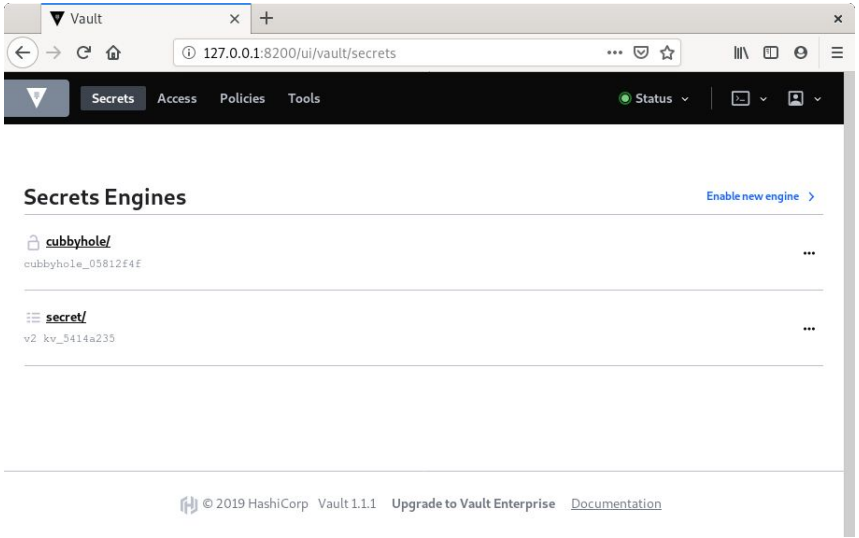
# Hashicorp Vault

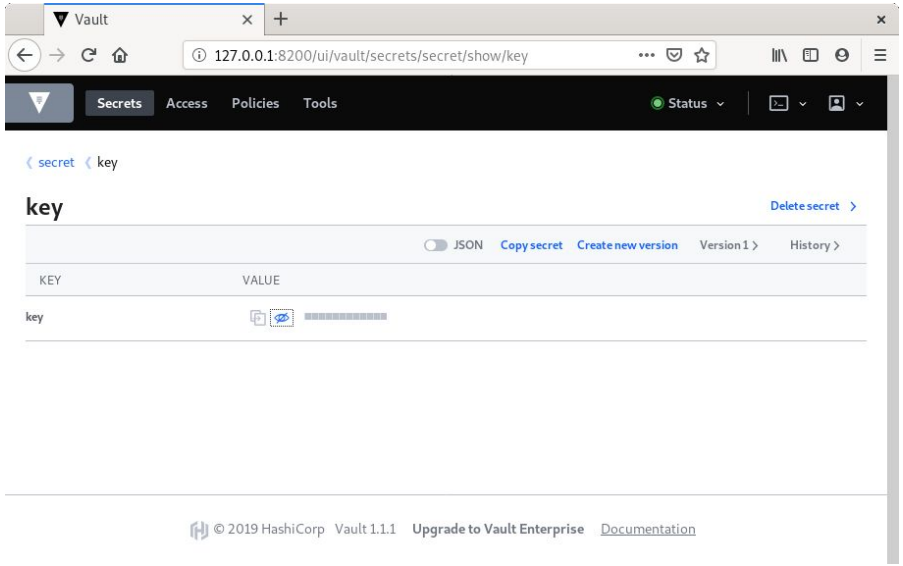


# Hashicorp Vault



# Hashicorp Vault



## Hashicorp Vault

The key features of the Vault are:

- It encrypts and decrypts data without storing it.
- Vault can generate secrets on-demand for some operations, such as AWS or SQL databases.
- Allows replication across multiple data centers.
- Vault has built-in protection for secret revocation.
- Serves as a secret repository with access control details.

# Keywhiz

- Keywhiz helps with infrastructure secrets, GPG keyrings, and database credentials, including **TLS** certificates and keys, symmetric keys, API tokens, and SSH keys for external services.
  - **Keywhiz Server**
  - **Keysync**
  - **Keywhiz CLI**
  - **Keywhiz automation API**

# Keywhiz

ADDING A SECRET

### Using Keywhiz CLI

```
$ keywhiz.cli --devTrustStore --user keywhizAdmin login
$ keywhiz.cli add secret --name mySecretName < mySecretFile
```

### Using Keywhiz automation API

The automation API requires a client certificate and `automationAllowed=true` in the clients DB table. For development purpose, you can use the pre-generated `client.p12` keystore:

```
$ cat request.json
{
  "name":"example.keytab",
  "description":"example kerberos keytab",
  "content":"a2V5dGFiIGNvbnRlbnQ=",
  "metadata":{"owner":"root","group":"root","mode":"0400"}
}
$ curl --cert ./server/src/test/resources/clients/client.p12:ponies -H "Content-Type:application/json" -d @request.json https://localhost:4444/automation/secrets/
```

## Keywhiz

The key features of Keywhiz are:

- Helps with infrastructure secrets, GPG keyrings, and database credentials, including TLS certificates and keys, symmetric keys, API tokens, and SSH keys for external services.
- Keywhiz Server provides JSON APIs for collecting and managing secrets.
- **It stores all secrets in memory only.**

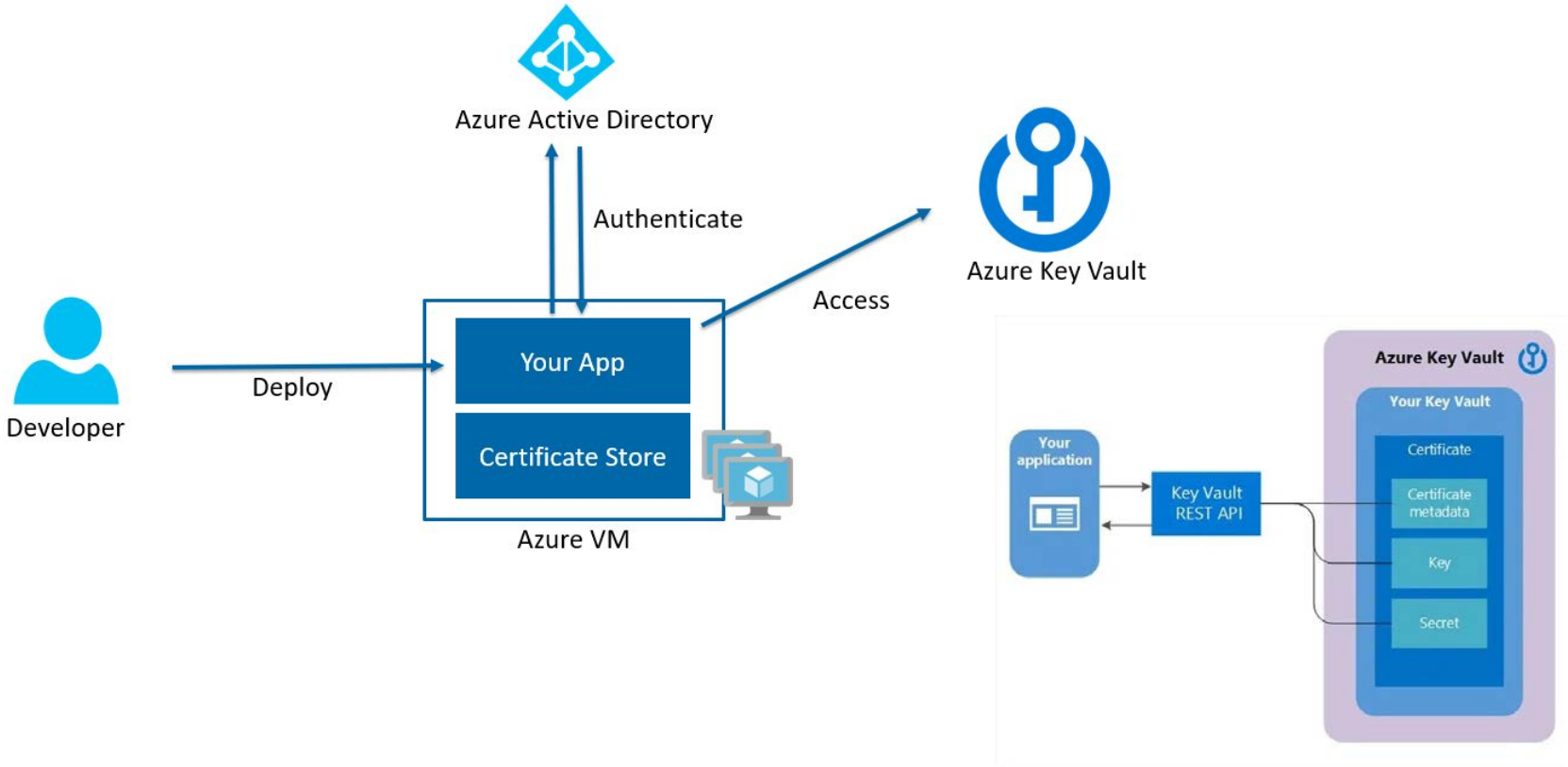
# AWS Secrets Manager

## AWS Secrets Manager

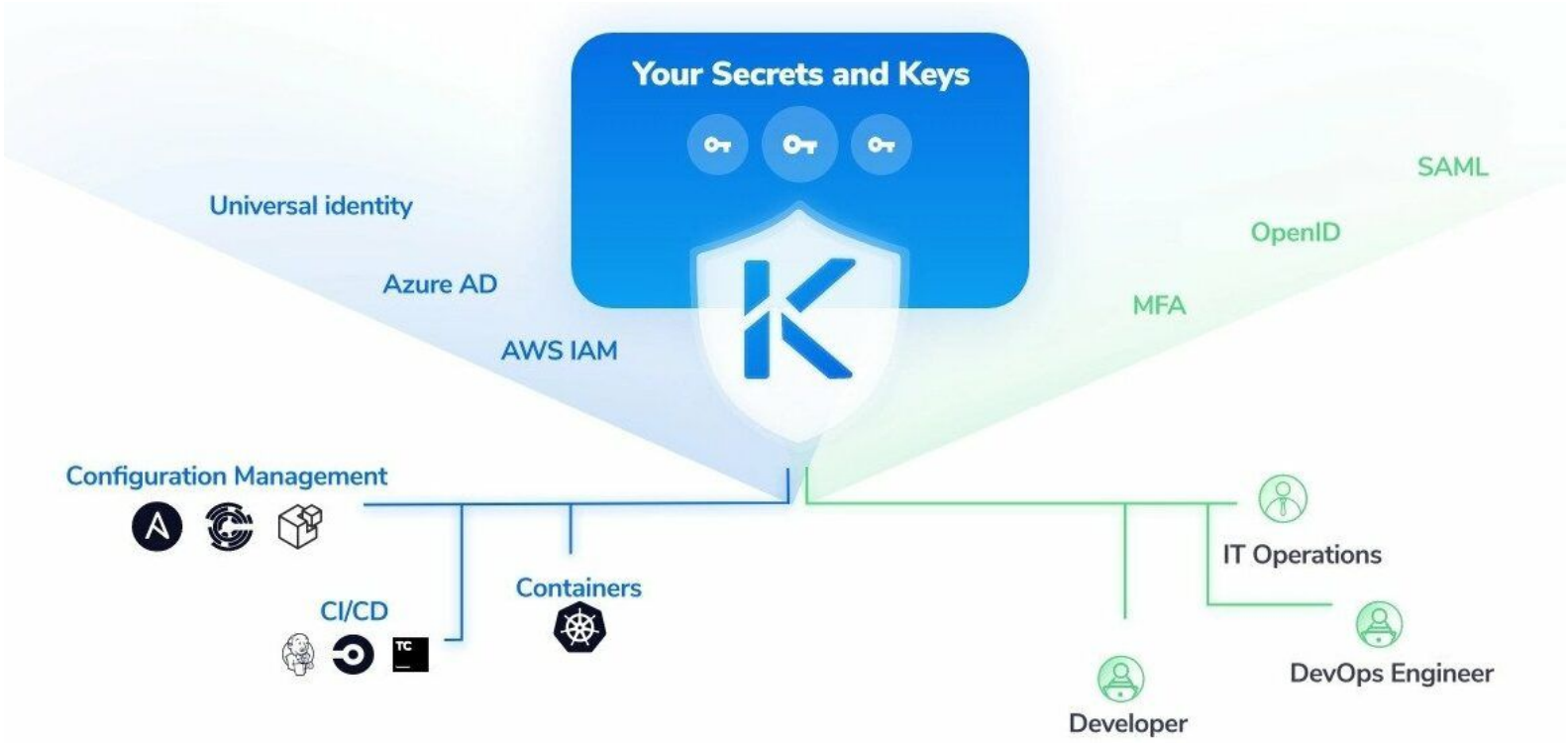
The key features of AWS Secrets Manager are:

- Encrypts and decrypts secrets, transmiting securely over TLS.
- Provides **client-side caching libraries** to improve the availability and reduce the latency of using your secrets.
- You can configure Amazon VPC (Virtual Private Cloud) endpoints to keep traffic within the AWS network.

# Azure Key Vault



# Akeyless Vault



## Akeyless Vault


The platform supports two more pillars:

- **Zero-Trust Application Access** by providing unified authentication and just-in-time access credentials, allowing you to secure the perimeter of applications and infrastructure.
- **Encryption as-a-Service**, allows customers to protect sensitive personal & business data by applying FIPS 140-2 certified app-level encryption.

## Conclusions

- Secrets are an important tool for any container-based architecture because they help us achieve the goal of keeping code and configuration separate.
- **Manage secrets in secure storage**