Configuration as Data - Environment Variables, Secrets, and ConfigMaps

Anthony E. Nocentino
ENTERPRISE ARCHITECT @ CENTINO SYSTEMS
@nocentino www.centinosystems.com

Course Overview

Configuring and Managing Storage in Kubernetes

Configuration as Data - Environment Variables, Secrets and ConfigMaps

Managing and Controlling the Kubernetes Scheduler

Overview

Configuring Pods with Environment Variables

Working with Sensitive Data Using Secrets

Managing Application Configuration with ConfigMaps

Why Do We Need Configuration as Data?

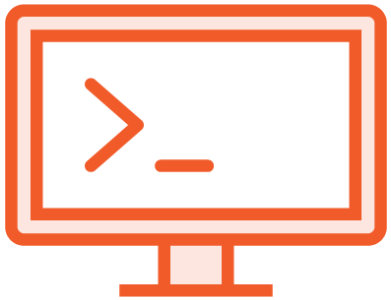
Abstraction

Container Images are Immutable

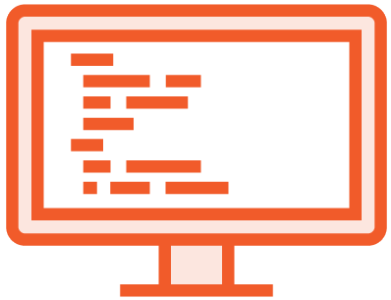
Service Discovery

Sensitive Information

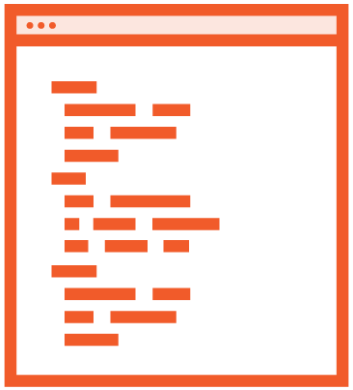
Configuring Applications in Pods



Command Line Arguments



Environment Variables



ConfigMaps

Environment Variables Inside Pods

User defined

- **Pod Spec for each container**
- **Defined inside the container image**
- **Defined in `name`/`value` or `valueFrom`**

System defined

- **Names of all Services available at the time the Pod was created**

Defined at container startup

Cannot be updated once the Pod is created

Defining Environment Variables

```
spec:
  containers:
  - name: hello-world
    image: gcr.io/google-samples/hello-app:1.0
    env:
    - name: DATABASE_SERVERNAME
      value: "sql.example.local"
    - name: BACKEND_SERVERNAME
      value: “be.example.local"
```

Demo

Passing configuration into containers using Environment Variables

Secrets

Store sensitive information as Objects

Retrieve for later use

Passwords, API tokens, keys and certificates

Safer and more flexible configurations (Pod Specs and Images)

Properties of Secrets

- base64 encoded
- Encryption can be configured
- Stored in etcd
- Namespaced and can only be referenced by Pods in the same Namespace
- Unavailable Secrets will prevent a Pods from starting up

https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

Creating Secrets

```
kubectl create secret generic app1 \
 --from-literal=USERNAME=app1login \
 --from-literal=PASSWORD='S0methingS@Str0ng!'
```

Using Secrets in Pods

Environment Variables

Volumes or Files

Updatable

Can be marked Immutable

Secret must be accessible to the Pod at startup

Using Secrets in Environment Variables

```
spec:
  containers:
  - name: hello-world
    ...
    env:
    - name: app1username
      valueFrom:
        secretKeyRef:
          name: app1
          key: USERNAME
    - name: app1password
      valueFrom:
        secretKeyRef:
          name: app1
          key: PASSWORD
```

```
spec:
  containers:
  - name: hello-world
    ...
    envFrom:
    - secretRef:
        name: app1
```

Using Secrets as Files

```
spec:
  volumes:
    - name: appconfig
      secret:                                   /etc/appconfig/USERNAME
        secretName: app1                        /etc/appconfig/PASSWORD
  containers:
    ...
    volumeMounts:
      - name: appconfig
        mountPath: "/etc/appconfig"
```

Demo

Creating and accessing Secrets

Accessing Secrets inside a Pod

- Environment variables
- Files and Volumes

Accessing a Private Container Registry

Secrets for application configuration

Use Secrets to access a private container registry

Want to access registries over the Internet

- **Docker Hub**
- **Cloud based container registries**

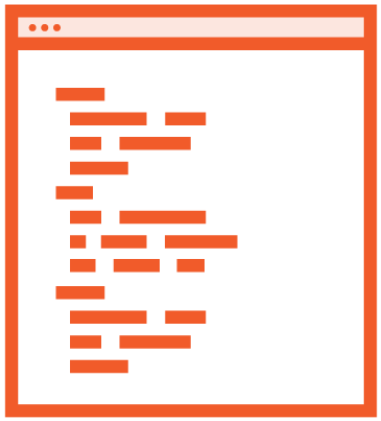
Create a Secret of type `docker-registry`

Enabling Kubernetes (kubelet) to pull the images from the private registry

Demo

Pulling a container image from a private container repository

ConfigMaps



Key value pairs exposed into a Pod used application configuration settings

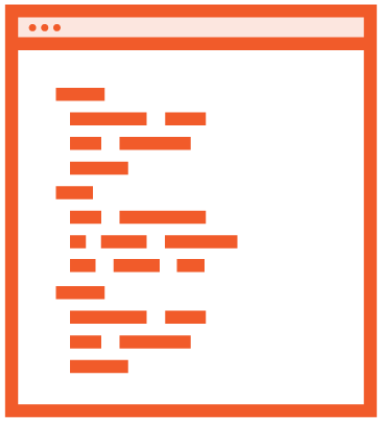
Defining application or environment specific settings

Decouple application and Pod configurations

Maximizing our container image's portability

Environment Variables or Files

Using ConfigMaps in Pods



Environment variables

`valueFrom` **and** `envFrom`

Volumes and Files

- **Volume mounted inside a container**
- **Single file or directory**
- **Many files or directories**
- **Volume ConfigMaps can be updated**
- **Marked Immutable**

Defining ConfigMaps

```
kubectl create configmap appconfigprod \
 --from-literal=DATABASE_SERVERNAME=sql.example.local \
 --from-literal=BACKEND_SERVERNAME=be.example.local

kubectl create configmap appconfigqa \
 --from-file=appconfigqa
```

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: appconfigprod
data:
  BACKEND_SERVERNAME: be.example.local
  DATABASE_SERVERNAME: sql.example.local
```

Using ConfigMaps in Environment Variables

```
spec:
  containers:
  - name: hello-world
    ...
    env:
    - name: DATABASE_SERVERNAME
      valueFrom:
        configMapKeyRef:
          name: appconfigprod
          key: DATABASE_SERVERNAME
    - name: BACKEND_SERVERNAME
      valueFrom:
        configMapKeyRef:
          name: appconfigprod
          key: BACKEND_SERVERNAME
```

```
containers:
- name: hello-world
  ...
  envFrom:
    - configMapRef:
        name: appconfigprod
```

Using ConfigMaps as Files

```
spec:
  volumes:
    - name: appconfig
      configMap:
        name: appconfigqa
  containers:
  - name: hello-world
    ...
    volumeMounts:
      - name: appconfig
        mountPath: "/etc/appconfig"
```

Demo

Creating ConfigMaps

Using ConfigMaps in Pods

Review

Configuring Pods with Environment Variables

Working with Sensitive Data Using Secrets

Managing Application Configuration with ConfigMaps

What's Next!

Managing and Controlling the Kubernetes Scheduler