[Hands-on] 12. Kubernetes ConfigMaps & Secrets

ConfigMap은 Key:Value 형식으로 저장이 됩니다.

문자로 생성하는 방식과 파일로 생성하는 방식이 있습니다.

ConfigMap from Literal

문자로 생성하는 방법은 kubectl create configmap 명령을 사용하여 다음과 같이 생성할 수 있습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl create configmap literal-config --from-literal=company=Samsung
configmap/literal-config created
```

> **명령어** : `kubectl create configmap literal-config --from-literal=company=Samsung`
> key는 company, value는 samsung 으로 생성

원하는대로 잘 생성되었는지는 describe를 통해 확인 가능합니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      2d2h
literal-config     1      3s

ubuntu@ip-10-0-1-161:~$ kubectl describe configmaps literal-config
Name:         literal-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
company:
----
Samsung

BinaryData
====

Events:  <none>
```

> **명령어 :** `kubectl get configmap` , `kubectl describe configmaps literal-config`

ConfigMap from File

파일을 통해 생성하는 방법은 다음과 같습니다.

--from-file 을 사용하면, key 는 파일명이 되고 value는 파일의 내용 자체가 됩니다.
--from-env-file 을 사용하면, 파일내에 key=value로 선언되어 있는 것들이 각각의 data로 configmap에 저장이 됩니다.

먼저 Properties 파일을 하나 만들어 봅니다.

```
database.url=192.168.0.88
database.port=5432
database.db=employee
database.user=hojoon
database.password=elqlvotmdnjem
```

> 파일명은 app.properties 로 합니다.

이제 이 파일을 통해 configmap을 만들어 보겠습니다.
먼저 --from-file 을 사용하여 file-config 라는 이름의 configmap을 만듭니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl create configmap file-config --from-file=./app.properties
configmap/file-config created
```

> **명령어** : `kubectl create configmap file-config --from-file=./app.properties`

다음으로 --from-env-file 을 사용하여 file-env-config 라는 이름의 configmap을 만듭니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl create configmap file-env-config --from-env-file=./app.properties
configmap/file-env-config created
```

> **명령어 :** `kubectl create configmap file-env-config --from-env-file=./app.properties`

두개의 configmap 을 만들었고, 앞에서와 동일하게 각각의 configmap을 describe를 통해 확인해 보겠습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl describe configmaps file-config
Name:         file-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
app.properties:
----
database.url=192.168.0.88
database.port=5432
database.db=employee
database.user=hojoon
database.password=elqlvotmdnjem

BinaryData
====

Events:  <none>
```

> **명령어 :** `kubectl describe configmaps file-config`

```
ubuntu@ip-10-0-1-161:~$ kubectl describe configmaps file-env-config
Name:         file-env-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
database.url:
----
192.168.0.88
database.user:
----
hojoon
database.db:
----
employee
database.password:
----
elqlvotmdnjem
database.port:
----
5432

BinaryData
====

Events:  <none>
```

> **명령어** : `kubectl describe configmaps file-env-config`

Data 부분에 Key와 Value가 각각 어떻게 저장되었는지 확인할 수 있습니다.

ConfigMap from Yaml

yaml 파일로도 생성할 수 있으며, key:value를 여러쌍 포함시킬 수도 있습니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: yaml-config
data:
  location: Jamsil
  business: ITService
```

> 파일명은 yaml-config.yaml 로 합니다.

작성된 yaml을 적용하겠습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl apply -f yaml-config.yaml
configmap/yaml-config created
```

> **명령어 :** `kubectl apply -f yaml-config.yaml`

동일하게 describe로 확인해 봅니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl describe configmaps yaml-config
Name:         yaml-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
business:
----
ITService
location:
----
Jamsil

BinaryData
====

Events:  <none>
```

> **명령어** : `kubectl describe configmaps yaml-config`

yaml파일에 작성한 key-value 쌍이 data로 생성되어 있습니다.

이제 만들어진 ConfigMap들을 사용해보도록 하겠습니다.
먼저 Pod를 생성할 파일을 하나 작성합니다.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
    - name: configmap-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "while true; do echo hi; sleep 10; done" ]
      env:
        - name: COMPANY
          valueFrom:
            configMapKeyRef:
              name: literal-config
              key: company
        - name: LOCATION
          valueFrom:
            configMapKeyRef:
              name: yaml-config
              key: location
        - name: BUSINESS
          valueFrom:
            configMapKeyRef:
              name: yaml-config
              key: business
      envFrom:
        - configMapRef:
            name: file-env-config
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: file-config
  restartPolicy: Never
```

> 파일명은 configmappod.yaml 로 합니다.

Manifest를 보면, 아주 가벼운 busybox shell 만 포함하고 있는 이미지를 사용합니다.

- literal-config에서 company키에 해당하는 value를 COMPANY 환경 변수에 담아주고,
- yaml을 통해 생성했던, yaml-config에서 location과 business key에 해당하는 value를 LOCATION과 BUSINESS 환경 변수에 담아주도록 하였습니다.
- 그리고, 파일로 부터 생성한 file-env-config의 모든 Key와 Value를 환경 변수에 담아 주고
- 마지막으로 file-config는 Volume으로 정의한 후 /etc/config 경로에 app.properties 파일로 Mount 시켰습니다.

이제 작성한 Manifest를 통해 Pod을 생성합니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl apply -f configmappod.yaml
pod/configmap-pod created
```

> **명령어** : `kubectl apply -f configmappod.yaml`

해당 pod의 환경변수에 어떤 값들이 들어갔는지 확인해 보겠습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl exec -it configmap-pod -- env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=configmap-pod
TERM=xterm
BUSINESS=ITService
database.password=elqlvotmdnjem
database.port=5432
database.url=192.168.0.88
database.user=hojoon
database.db=employee
COMPANY=Samsung
LOCATION=Jamsil
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
HOME=/root
```

> **명령어 :** `kubectl exec -it configmap-pod -- env`

마지막으로, Volume으로 Mount된 파일의 내용도 확인해 보겠습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl exec -it configmap-pod -- cat /etc/config/app.properties
database.url=192.168.0.88
database.port=5432
database.db=employee
database.user=hojoon
database.password=elqlvotmdnjem
```

> **명령어 :** `kubectl exec -it configmap-pod -- cat /etc/config/app.properties`

kubectl exec -it configmap-pod -- cat /etc/config/app.properties 를 실행해 봅니다.

이제 **Secret**을 작성하고 사용하는 방법을 알아보겠습니다.

Secret은 **ConfigMap**과 동일하게 **Key:Value** 형식으로 저장이 됩니다.

차이점은, 저장될 때 base64 encoding이 되어서 저장된다는 점입니다. 사실 암호화되어 저장되는 것도 아니고 단순히 base64 encoding만 되기 때문에 안전하다고 할 수는 없으나 공격자(?)에게는 혼란을 줄 수 있습니다.

Secret from Yaml

Secret도 ConfigMap과 동일하게 `--from-literal` 이나 `--from-file`, `--from-env-file`을 사용할 수 있습니다.
이번 실습에는 YAML파일을 사용하는 방법만 사용해보겠습니다.

먼저 Secret을 위한 yaml파일을 하나 작성합니다.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: yaml-secret
data:
  location: SmFtc2ls
  business: SVRTZXJ2aWNl
```

> 파일명은 yaml-secret.yaml 로 합니다.

configmap과 달리 secret을 생성할 때는, value를 base64 encoding한 값으로 작성해야만 합니다.
location의 value는 Jamsil 을 base64 encoding 한 값이며, business의 value는 ITService를 base64 encoding 한 값입니다.

아래를 참고하세요.

```
ubuntu@ip-10-0-1-161:~$ echo -n 'Jamsil' | base64
SmFtc2ls
ubuntu@ip-10-0-1-161:~$ echo -n 'ITService' | base64
SVRTZXJ2aWNl
```

> **명령어 :** `echo -n 'Jamsil' | base64`, `echo -n 'ITService' | base64`

작성된 yaml을 적용하겠습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl apply -f yaml-secret.yaml
secret/yaml-secret created
```

> **명령어 :** `kubectl apply -f yaml-secret.yaml`

동일하게 describe로 확인해 봅니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl describe secret yaml-secret
Name:         yaml-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
business:  9 bytes
location:  6 bytes
```

> **명령어 :** `kubectl describe secret yaml-secret`

이제 앞에서 만든 Secret을 사용할 Pod를 준비하겠습니다.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
    - name: secret-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "while true; do echo hi; sleep 10; done" ]
      env:
        - name: LOCATION
          valueFrom:
            secretKeyRef:
              name: yaml-secret
              key: location
        - name: BUSINESS
          valueFrom:
            secretKeyRef:
              name: yaml-secret
              key: business
  restartPolicy: Never
```

> 파일명은 secretpod.yaml 로 합니다.

Manifest를 보면, 아주 가벼운 busybox shell 만 포함하고 있는 이미지를 사용합니다.

- yaml-secret에서 location과 business key에 해당하는 value를 LOCATION과 BUSINESS 환경 변수에 담아주도록 하였습니다.

이제 Pod을 생성합니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl apply -f secretpod.yaml
pod/secret-pod created
```

> **명령어** : `kubectl apply -f secretpod.yaml`

해당 pod의 환경변수에 어떤 값들이 들어갔는지 확인해 보겠습니다.

```
ubuntu@ip-10-0-1-161:~$ kubectl exec -it secret-pod -- env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=secret-pod
TERM=xterm
LOCATION=Jamsil
BUSINESS=ITService
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
HOME=/root
```

> **명령어** : `kubectl exec -it secret-pod -- env`

이번 실습은 여기까지 입니다. _ ✍(。ˊ‿ˋ。)