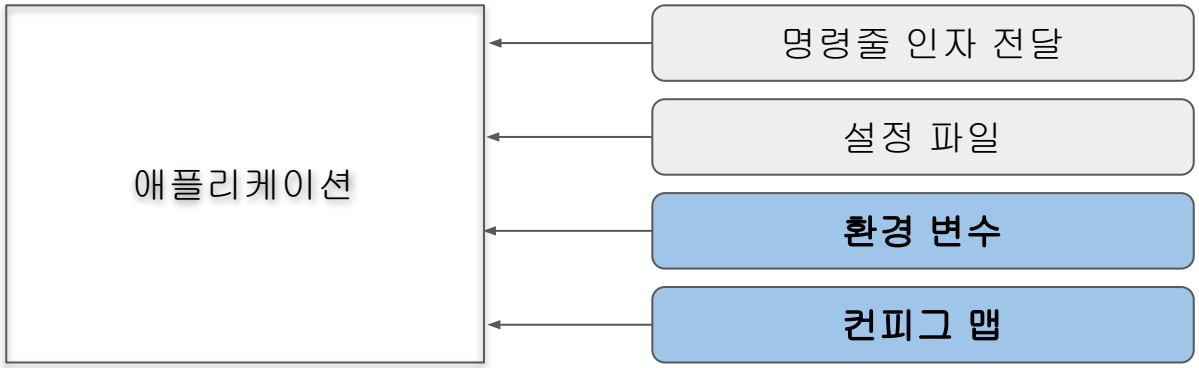
Kubernetes In Action

Ch7. 컨피그맵과 시크릿:애플리케이션 설정

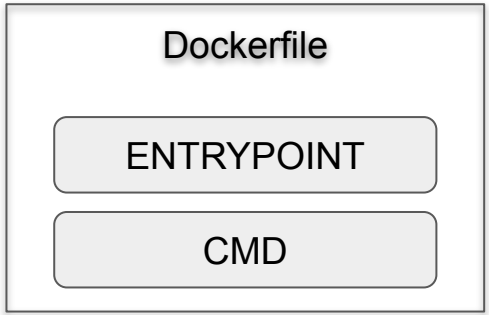
2020.08.29
written by whatwant

configMap
&
secret

애플리케이션 설정



Docker: 명령줄 인자 전달 #1/2



컨테이너가 시작될 때 호출될 명령어 정의

ENTRYPOINT에 전달되는 인자 정의

ENTRYPOINT 없이
명령어들을 넣어서
사용할 수도 있다.

Dockerfile

```
FROM ubuntu:latest
RUN apt-get update ; apt-get -y install fortune
ADD fortuneloop.sh /bin/fortuneloop.sh
ENTRYPOINT [“/bin/fortuneloop.sh”]
CMD [“10”]
```

exec 형태의
ENTRYPOINT 명령

shell 형태의
ENTRYPOINT 명령

```
ENTRYPOINT /bin/fortuneloop.sh
```

Docker: 명령줄 인자 전달 #2/2

```
$ docker build -t docker.io/luksa/fortune:args .
$ docker push docker.io/luksa/fortune:args
```

dockerfile CMD 값

```
$ docker run -it docker.io/luksa/fortune:args
Configured to generate new fortune every 10 seconds
Fri May 19 10:39:44 UTC 2017 Writing fortune to /var/htdocs/index.html
```

전달되는 15 값 사용

```
$ docker run -it docker.io/luksa/fortune:args 15
Configured to generate new fortune every 15 seconds
```

Kubernetes: 명령줄 인자 전달 #1/3

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune2s
spec:
  containers:
  - image: luksa/fortune:args
    command: ["/bin/command"]
    args: ["2"]
    name: html-generator
...
```

▼ 표 7.1 도커와 쿠버네티스의 실행파일과 인자를 지정하는 방법 비교

도커	쿠버네티스	설명
ENTRYPOINT	command	컨테이너 안에서 실행되는 실행파일
CMD	args	실행파일에 전달되는 인자

```
args: ["2", "foo", "bar" ]
```

```
args:
  - "2"
  - "foo"
  - "bar"
```

값이 복수개이면,
이렇게 2가지 방법으로 가능

다만, 이렇게 pod를
생성하면
인자값이 고정된다

Kubernetes: 환경변수 #2/3

pod.yaml

```
kind: Pod
spec:
  containers:
  - image: luksa/fortune:env
    env:
    - name: INTERVAL
      value: “30”
...
```

pod 정의할 때
이렇게 환경 변수를 선언해주고

Container에서
환경 변수 값을 가져다 사용하면

OK ~ !!

Container (luksa/fortune:env)

bash

```
$INTERVAL
```

Java

```
System.getenv(“INTERVAL”)
```

NodeJS

```
process.env.INTERVAL
```

Python

```
os.environ[‘INTERVAL’]
```

Kubernetes: 환경변수 #3/3

pod.yaml

```
env:
- name: FIRST_VAR
  value: “foo”
- name: SECOND_VAR
  value: $(FIRST_VAR)bar”
...
```

환경변수 값에서
다른 환경변수도 참조할 수 있다

이렇게 Pod 정의에서
환경 변수를 선언하는 것은...

여러 환경에서
동일한 Pod 정의를 사용하려면 ...

비효율 !!!

컨피그맵 !!

ConfigMap: create #1/2

이름 값

$ **kubectl create configmap** fortune-config **--from-literal=**sleep-interval=25

$ **kubectl get configmap** fortune-config **-o yaml**

$ **kubectl create -f** fortune-config.yaml

```
apiVersion: v1
data:
    sleep-interval: “25”
kind: ConfigMap
metadata:
    creationTimestamp: 2016-08-11T...
    name: fortune-config
    namespace: default
...
```

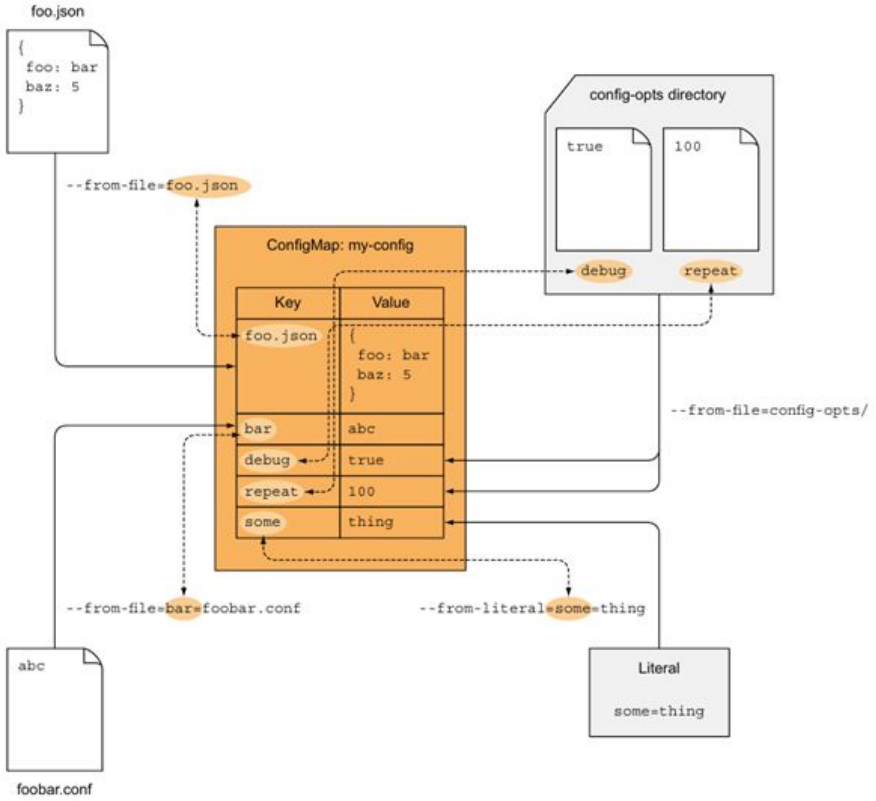
$ **kubectl create configmap** my-config **--from-file=**config-file.conf

$ **kubectl create configmap** my-config **--from-file=**customkey=config-file.conf customkey라는 키값으로 저장

$ **kubectl create configmap** my-config **--from-file=**/path/to/dir 디렉토리 안에 있는 모든 파일

→ 위의 옵션을 모두 결합해서 사용하는 것도 가능

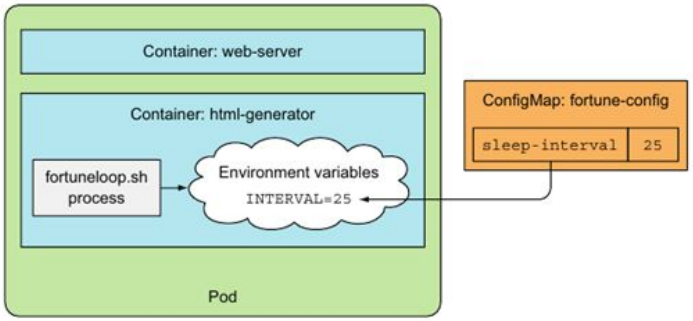
ConfigMap: create #2/2



ConfigMap: 환경변수 #1/2

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune-env-from-configmap
spec:
  containers:
  - image: luksa/fortune:env
    env:
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name:fortune-config
          key: sleep-interval
...
```



만약, 파드에 존재하지 않는 컨피그맵을 참조하게 되면

컨테이너는 시작 실패 → 대기

컨피그맵이 생성되면 컨테이너는 자동 시작

옵션을 통해 컨피그맵이 존재하지 않아도

컨테이너는 실행되도록 설정 가능

optional: true

ConfigMap: 환경변수 #2/2

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune-env-from-configmap
spec:
  containers:
  - image: luksa/fortune:env
    envFrom:
    - prefix: CONFIG_
      configMapKeyRef:
        name:my-config-map
...
```

env 대신 envFrom

모든 환경변수에 CONFIG_ 접두사가 붙음 (생략 가능)

Kubernetes v1.6 이후부터 지원하는 기능

Foo-BAR 처럼 환경변수에 사용될 수 없는 변수명일 경우 Skip !!

ConfigMap: 명령줄 인자 전달

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune-args-from-configmap
spec:
  containers:
  - image: luksa/fortune:args
    env:
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name:fortune-config
          key: sleep-interval
    args: ["$(INTERVAL)"]
...
```

첫 번째 인자에서 INTERVAL 값을 가져오는 이미지 사용

인자에

앞에서 정의한 컨피그맵으로부터 정의한 환경변수 값을 지정

ConfigMap: 설정 파일 #1/4

my-nginx-config.conf

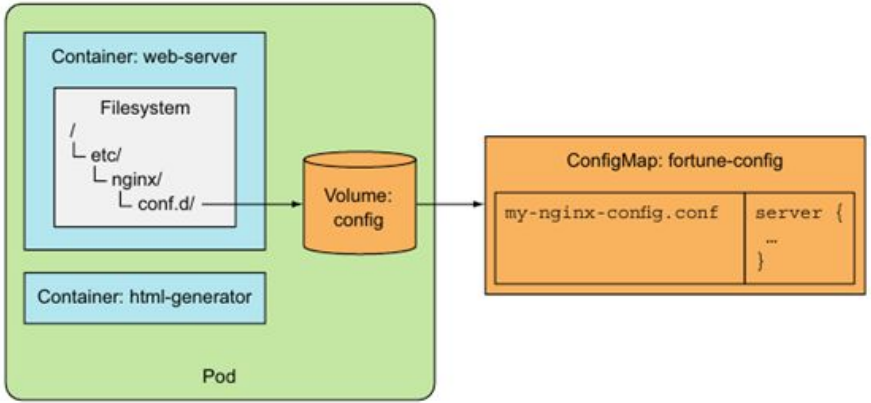
```
server {
  listen              80;
  server_name   www.kubia-example.com;

  gzip on;
  gzip_types  text/plain  application/xml;

  location / {
    root    /usr/share/nginx/html;
    index  index.html  index.htm;
  }
}
```

이 config 파일을 컨피그맵으로 관리 해보고자 한다.

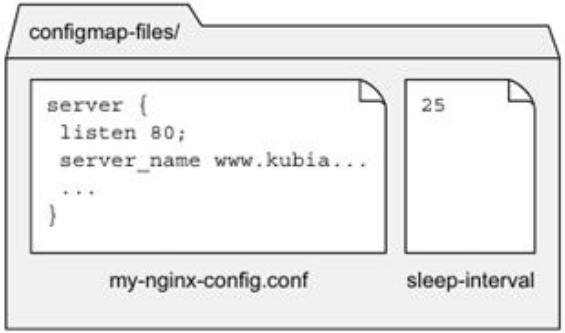
ConfigMap: 설정 파일 #2/4

① 기존 컨피그맵 삭제

```
$ kubectl delete configmap fortune-config
```



② 디렉토리 작성



- 디렉토리 하나 만들고
- nginx config 파일
- sleep-interval 환경변수 값
  (텍스트로 값을 지정)

③ 컨피그맵 생성

```
$ kubectl create configmap fortune-config --from-file=configmap-files
```

```
$ kubectl get configmap fortune-config -o yaml
```

yaml로 확인해보고 싶으면

ConfigMap: 설정 파일 #3/4

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune-configmap-volume
spec:
  containers:
  - image: nginx:alpine
    name: web-server
    volumeMounts:
    ...
    - name: config
      mountPath: /etc/nginx/conf.d
      readOnly: true
    ...
```

```
volumes:
…
- name: config
  configMap:
    name: fortune-config
...
```

④ 컨피그맵의 내용을 가진 볼륨 생성 & 마운트

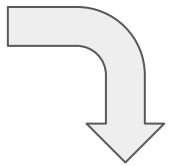
⑤ 마운트 된 컨피그맵 볼륨 내용 살펴보기

```
$ kubectl exec fortune-configmap-volume -c web-server ls /etc/nginx/conf.d
```

ConfigMap: 설정 파일 #4/4

nginx.conf 파일 외에

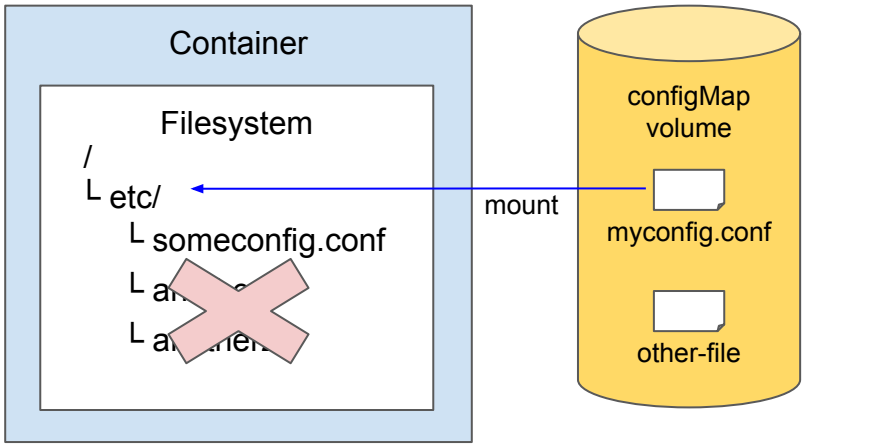
sleep-interval 파일도 있다!!!



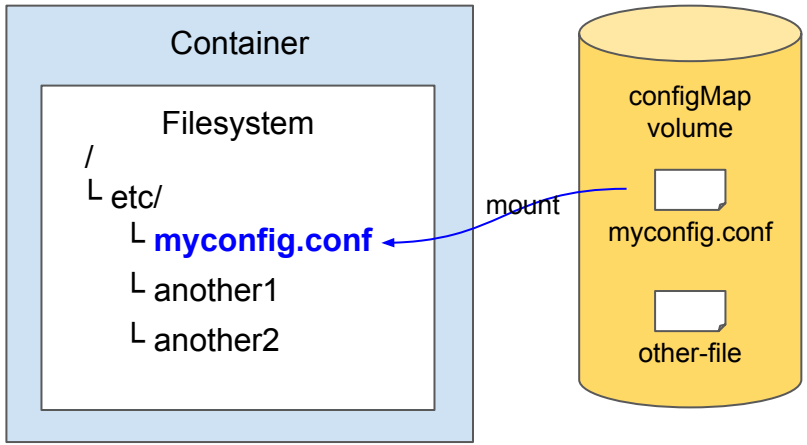
```
volumes:
…
- name: config
  configMap:
    name: fortune-config
    items:
    - key: my-nginx-config.conf
      path: gzip.conf
…
```

볼륨에 포함할 항목을 지정하고,
어떤 파일명으로 할 것인지도 지정

ConfigMap: 마운트 이슈

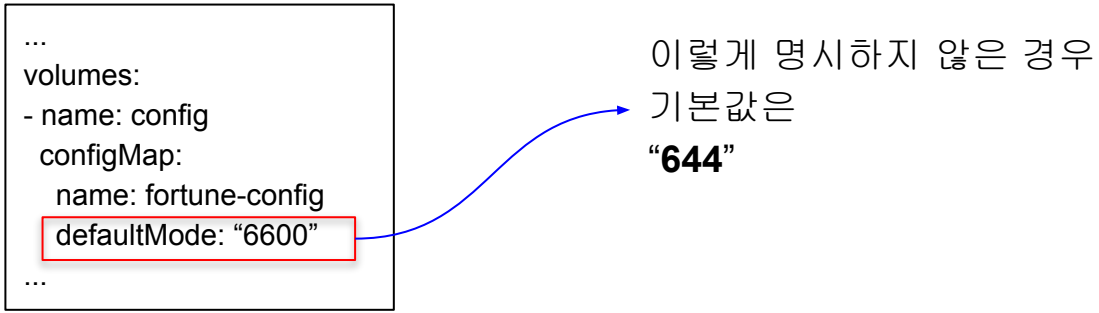


이미 파일이 있는 디렉토리에
마운트를 하는 경우
기존 파일에는 접근할 수가 없다.




```
...
volumeMounts:
- name: myvolume
  mountPath: /etc/someconfig.conf
  subPath: myconfig.conf
...
```

ConfigMap: 퍼미션



ConfigMap: 재시작 없이 설정 업데이트

① 기존 컨피그맵 편집

```
$ kubectl edit configmap fortune-config
```

② 업데이트 내역 확인

```
$ kubectl exec fortune-configmap-volume -c web-server cat /etc/nginx/conf.d/my-nginx-config.conf
```

③ nginx 설정 리로드

```
$ kubectl exec fortune-configmap-volume -c web-server -- nginx -s reload
```

ConfigMap: 업데이트 #1/2

```
total 4
drwxr-xr-x  ... 12:15 ..4984_09_04_12_15_06.865837643
lrwxrwxrwx  ... 12:15 ..data -> ..4984_09_04_12_15_06.865837643
lrwxrwxrwx  ... 12:15 my-nginx-config.conf -> ..data/my-nginx-config.conf
lrwxrwxrwx  ... 12:15 sleep-interval -> ..data/sleep-interval
```

마운트된 컨피그맵 볼륨안의 파일은
심볼릭 링크다 !!

↓

업데이트 하면 새로운 디렉토리로 작성
그리고 심볼릭 링크 교체
→ 한 번에 업데이트

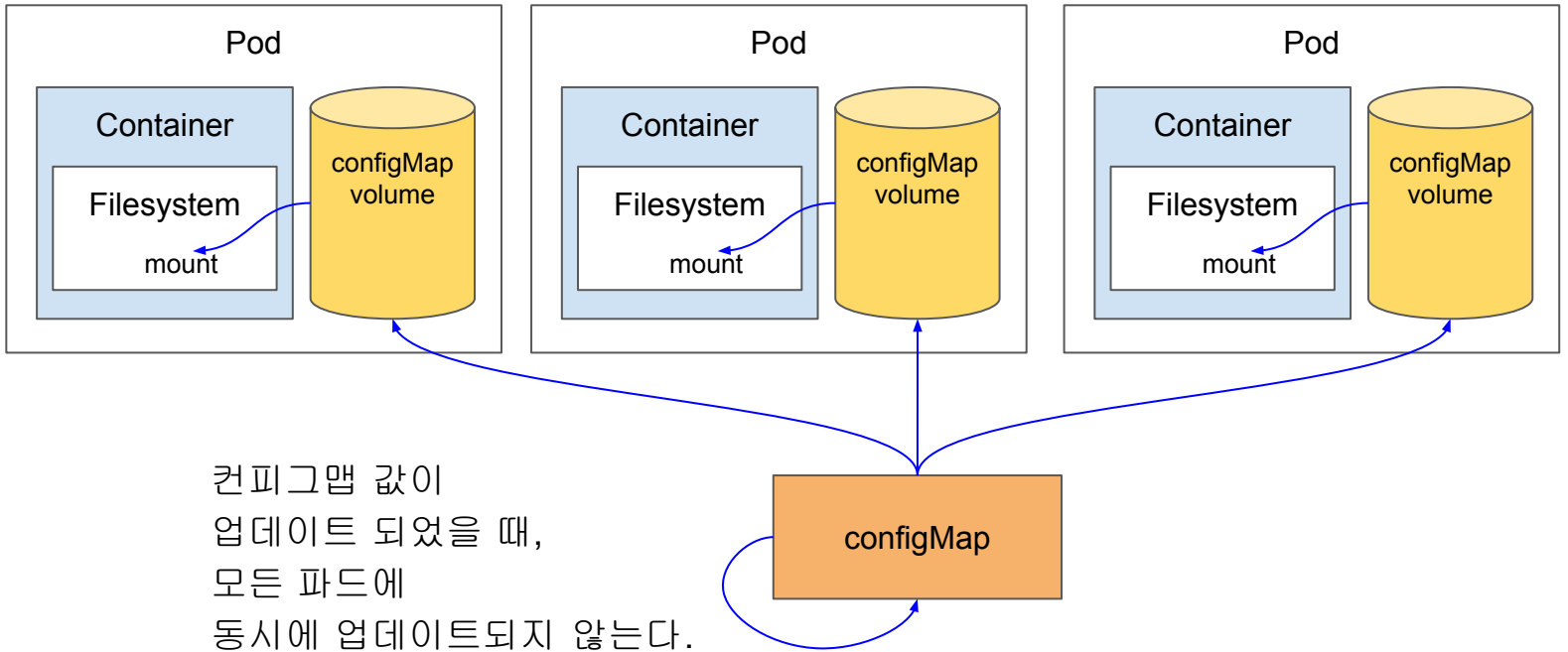
But,

파일로 마운트된 파일은 업데이트 되지 않는다 !!!

workaround

다른 디렉토리에 마운트 한 뒤에
심볼릭 링크로 파일 지정

ConfigMap: 업데이트 #2/2



컨피그맵 값이
업데이트 되었을 때,
모든 파드에
동시에 업데이트되지 않는다.
최대 1분까지
기존 값을 갖고 있을 수 있다.

시크릿

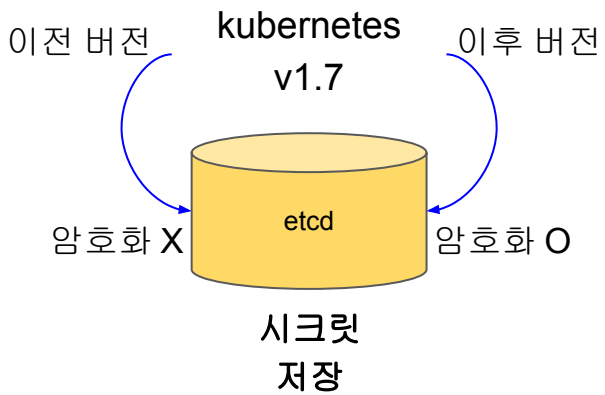
컨피그맵과 비슷한

시크릿

자격 증명이나 개인 암호화 키와 같은
보안이 유지되어야 하는
민감한 정보는 ?

보안을 위해

시크릿에 접근해야하는 파드가 실행되는 노드에만 배포	메모리에만 저장

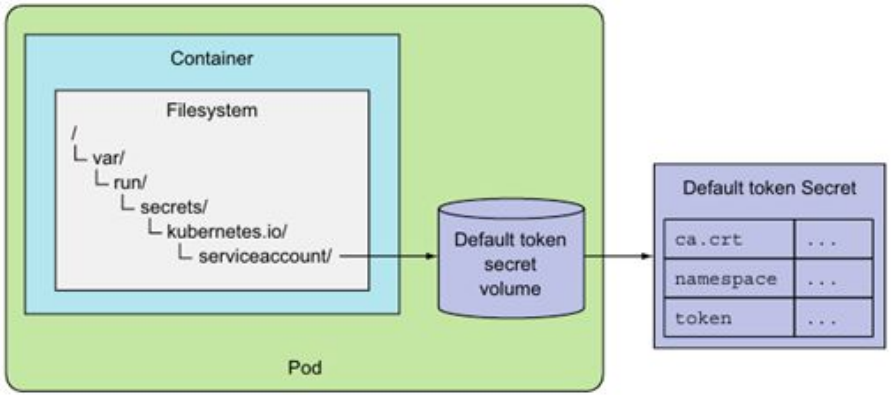


시크릿: secrets & default-token

```
$ kubectl describe secrets
Name:         default-token-cfee9
Namespace:    default
Labels:       <none>
Annotations:  kubernetes.io/service-account.name=default
              kubernetes.io/service-account.uid=cc04bb39-b53f-42010af00237
Type:         kubernetes.io/service-account-token

Data
====
ca.crt:       1139 bytes
namespace:    7 bytes
token:        eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

모든 파드에 기본적으로 연결되어 있는 시크릿

kubernetes API 서버와 통신할 때 필요한 것들



시크릿: 생성 #1/2

시크릿 생성

- 도커 레지스트리 사용: docker-registry
- TLS 통신: tls, generic

시크릿의 3가지 유형

```
$ kubectl create secret generic fortune-https --from-file=https.key --from-file=https.cert --from-file=foo
```

디렉토리를 지정할 수도 있다

```
$ kubectl get secret fortune-https -o yaml
```

secret.yaml

```
apiVersion: v1
data:
  foo: YmFyCg==
  https.cert: :S0tLS1CRU….
  https.key: LS0tLS1CRU….
kind: Secret
...
```

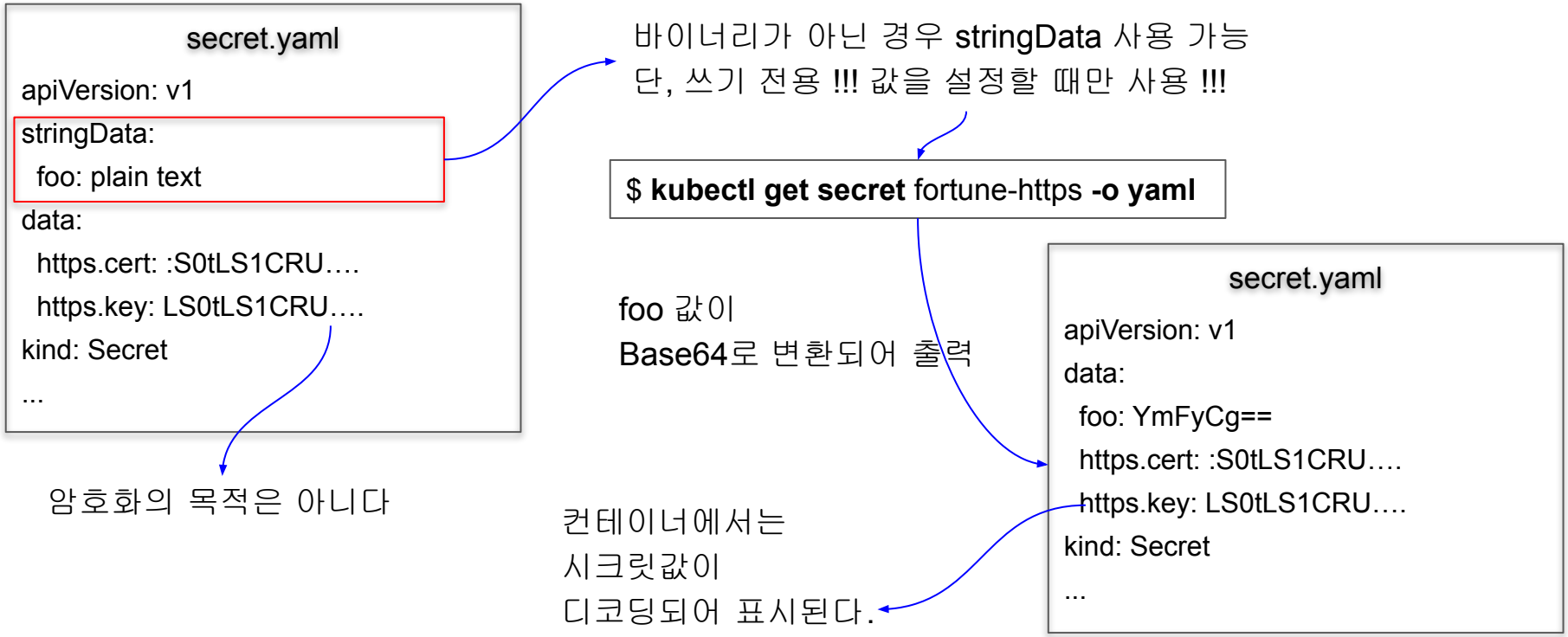
Base64 인코딩된 값으로 되어있음

Why?

바이너리 데이터 시크릿도 사용되기 때문에…

Base64 인코딩을 하면 바이너리도 담을 수 있음

시크릿: 생성 #2/2


secret.yaml
apiVersion: v1
stringData:
foo: plain text
data:
https.cert: :S0tLS1CRU….
https.key: LS0tLS1CRU….
kind: Secret
...
암호화의 목적은 아니다
바이너리가 아닌 경우 stringData 사용 가능
단, 쓰기 전용 !!! 값을 설정할 때만 사용 !!!
$ kubectl get secret fortune-https -o yaml
foo 값이
Base64로 변환되어 출력
secret.yaml
apiVersion: v1
data:
foo: YmFyCg==
https.cert: :S0tLS1CRU….
https.key: LS0tLS1CRU….
kind: Secret
...
컨테이너에서는
시크릿값이
디코딩되어 표시된다.

시크릿: 기본 사용

pod.yaml

```
apiVersion: v1
kind: Pod
…
   volumeMounts:
     …
    - name: certs
      mountPath: /etc/nginx/certs/
      readOnly: true
 …
 volumes:
   …
  - name: certs
    secret:
      secretName: fortune-https
```

일반적인 마운트와
동일한 형식이다.

configMap 대신
secret을 사용하는 것 외에
차이가 없다.

configMapKeyRef 대신
secretKeyRef를 사용

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune-env-from-secretKey
spec:
  containers:
  - image: luksa/fortune:env
    env:
    - name: FOO_SECRET
      valueFrom:
        secretKeyRef:
          name:fortune-https
          key: foo
...
```

시크릿: docker-registry

```
$ kubectl create secret docker-registry dockerhub-secret
    --docker-username=myname --docker-password=mypasswd --docker-email=myemail@mail.net
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: private-pod
spec:
  imagePullSecrets:
    - name: dockerhub-secret
  containers:
    - image: username/private:tag
      name: main
```

프라이빗 이미지 레지스트리 사용 !!!

The End