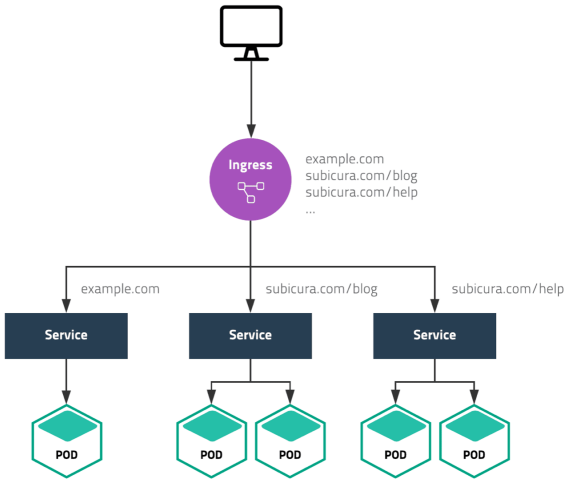
week9

▼ 섹션 3. 쿠버네티스 기본 실습

▼ 🐥 Ingress

- 도메인을 이용하여 **서로 다른 서비스에 접근**하는 방법을 알아봅니다.
- **하나의 클러스터에서 여러 가지 서비스를 운영**한다면 외부 연결을 어떻게 할까요? NodePort를 이용하면 서비스 개수만큼 포트를 오픈하고 사용자에게 어떤 포트인지 알려줘야 합니다. 그럴순 없죠!



위 샘플은 `example.com`, `subicura.com/blog`, `subicura.com/help` 주소로 서로 다른 서비스에 접근하는 모습입니다. `80(http)` 또는 `443(https)` 포트로 **여러 개의 서비스를 연결해야 하는데 이럴 때 `Ingress`를 사용**합니다.

▼ 🎀 Ingress 만들기

echo 웹 애플리케이션을 버전별로 도메인을 다르게 만들어 보겠습니다.

`minikube ip`로 테스트 클러스터의 노드 IP를 구하고 도메인 주소로 사용합니다. 결과 IP가 `192.168.64.5`라면 사용할 도메인은 다음과 같습니다.

- **v1.echo.192.168.64.5.sslip.io**
- **v2.echo.192.168.64.5.sslip.io**

> 💡 **TIP**
>
> 도메인을 테스트하려면 여러가지 설정이 필요합니다. 여기서는 별도의 설정없이 IP주소를 도메인에 넣어 바로 사용할 수 있는 **sslip.io**서비스를 이용합니다.

▼ 👻 minikube에 Ingress 활성화하기

Ingress는 Pod, ReplicaSet, Deployment, Service와 달리 **별도의 컨트롤러를 설치해야 합니다**. 여러 가지 컨트롤러 중에 입맛에 맞게 고를 수 있는데 여기서는 **nginx ingress controller**를 사용합니다.

> 💡 **Ingress Controller**
>
> nginx를 제외한 대표적인 컨트롤러로 haproxy, traefik, alb등이 있습니다.

```
minikube addons enable ingress

# ingress 컨트롤러 확인
kubectl -n ingress-nginx get pod
```

실행 결과

```
C:\Users\user\Downloads\cmder
λ kubectl -n ingress-nginx get pod
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-wj6w2        0/1     Completed   0          3m8s
ingress-nginx-admission-patch-zx8t8         0/1     Completed   0          3m8s
ingress-nginx-controller-64d5884ccb-5fd9g   1/1     Running     0          3m8s
```

잘 설정 되었는지 확인합니다.

`curl -I http://192.168.59.100/healthz # minikube ip를 입력`

실행 결과

```
C:\Users\user\Downloads\cmder
λ curl -I http://192.168.59.100/healthz
HTTP/1.1 200 OK
Date: Wed, 30 Aug 2023 16:36:53 GMT
Content-Type: text/html
Content-Length: 0
Connection: keep-alive
```

▼ 👻 echo 웹 애플리케이션 배포

Nginx Ingress Controller 설치가 완료되면 echo 웹 애플리케이션을 배포합니다. v1, v2 2가지를 배포합니다.

Ingress Spec중에 `rules.host` 부분을 `minikube ip` 로 변경해야 합니다.

▼ guide/ingress/echo-v1.yml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: echo-v1
spec:
  rules:
    - host: v1.echo.192.168.64.5.sslip.io
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: echo-v1
                port:
                  number: 3000

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-v1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: echo
      tier: app
      version: v1
  template:
```

```
    metadata:
      labels:
        app: echo
        tier: app
        version: v1
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v1
          livenessProbe:
            httpGet:
              path: /
              port: 3000

---
apiVersion: v1
kind: Service
metadata:
  name: echo-v1
spec:
  ports:
    - port: 3000
      protocol: TCP
  selector:
    app: echo
    tier: app
    version: v1
```

▼ **guide/ingress/echo-v2.yml**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: echo-v2
spec:
  rules:
    - host: v2.echo.192.168.64.5.sslip.io
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: echo-v2
                port:
                  number: 3000

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-v2
spec:
  replicas: 3
  selector:
    matchLabels:
      app: echo
      tier: app
      version: v2
  template:
    metadata:
      labels:
        app: echo
        tier: app
        version: v2
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v2
          livenessProbe:
            httpGet:
              path: /
              port: 3000

---
apiVersion: v1
kind: Service
metadata:
  name: echo-v2
spec:
```

```
  ports:
    - port: 3000
      protocol: TCP
  selector:
    app: echo
    tier: app
    version: v2
```

💡 **v1.18 이하 버전**

쿠버네티스 v1.19부터 Ingress Spec 중 `rules.http.paths.backend`가 변경되었습니다. v1.18 이하라면 `pathType`을 제거하고 `backend`를 다음과 같이 변경해주세요.

```
backend:
  serviceName: echo-v1
  servicePort: 3000
```

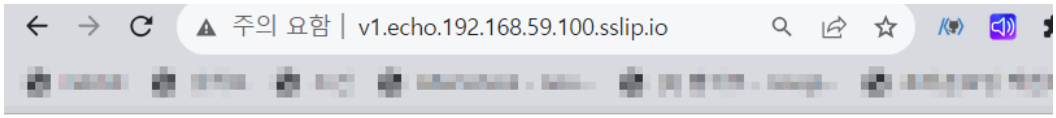
Deployment, Service, Ingress를 생성합니다.

```
kubectl apply -f echo-v1.yml,echo-v2.yml

# Ingress 상태 확인
kubectl get ingress
kubectl get ing
```

실행 결과

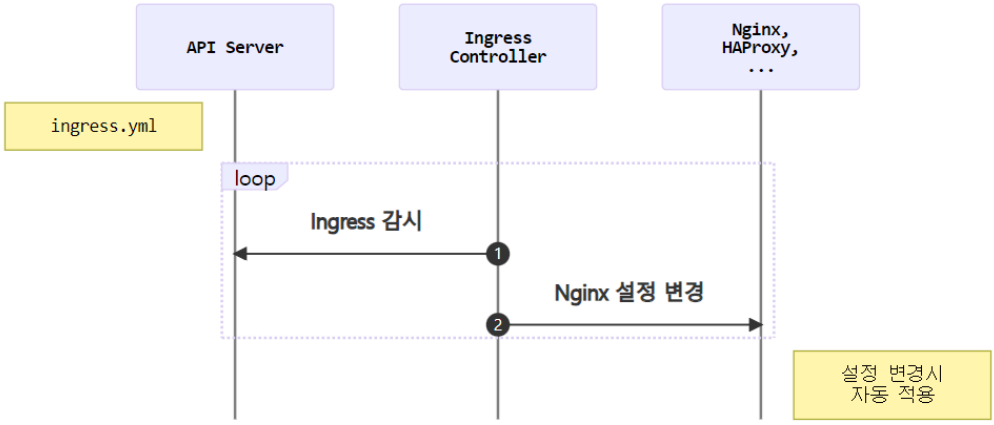
```
λkubectl get ingress
NAME      CLASS   HOSTS                                ADDRESS          PORTS   AGE
echo-v1   nginx   v1.echo.192.168.59.100.sslip.io   192.168.59.100   80      2m45s
echo-v2   nginx   v2.echo.192.168.59.100.sslip.io   192.168.59.100   80      2m44s
```

`v1.echo.192.168.64.5.sslip.io`과 `v2.echo.192.168.64.5.sslip.io`로 접속 테스트합니다.




```
version: v1
hostname: echo-v1-7495d856f9-hmbwj
headers:
  host: v1.echo.192.168.59.100.sslip.io
  x-request-id: 16ff33db5bc64e535c0faadb6cd27033
  x-real-ip: 192.168.59.1
  x-forwarded-for: 192.168.59.1
  x-forwarded-host: v1.echo.192.168.59.100.sslip.io
  x-forwarded-port: '80'
  x-forwarded-proto: http
  x-forwarded-scheme: http
  x-scheme: http
  upgrade-insecure-requests: '1'
  user-agent: >-
    Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/116.0.0.0 Safari/537.36
  accept: >-

text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,app
exchange;v=b3;q=0.7
  accept-encoding: gzip, deflate
  accept-language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
  cookie: _ga=GA1.2.492583870.1693413014; _gid=GA1.2.179609119.1693413014
query: {}
```

▼ 🎀 **Ingress 생성 흐름**



1. `Ingress Controller`는 `Ingress` **변화를 체크**
2. `Ingress Controller`는 **변경된 내용을** `Nginx` **에 설정하고 프로세스 재시작**

동작방식을 보면 **YAML로 만든 Ingress 설정을 단순히 nginx 설정으로 바꾸는 걸 알 수 있습니다.** 이러한 과정을 수동으로 하지 않고 Ingress Controller가 하는 것 뿐입니다.

```
❯ kubectl -n kube-system get po
NAME                                        READY   STATUS      RESTARTS   AGE
coredns-74ff55c5b-7t8v2                     1/1     Running     1          159m
etcd-minikube                               1/1     Running     1          160m
ingress-nginx-admission-create-5hfbg        0/1     Completed   0          8m43s
ingress-nginx-admission-patch-jgd5p         0/1     Completed   0          8m43s
ingress-nginx-controller-558664778f-5gzlf   1/1     Running     0          8m43s
kube-apiserver-minikube                     1/1     Running     1          160m
kube-controller-manager-minikube            1/1     Running     1          160m
kube-proxy-wgw8x                            1/1     Running     1          159m
kube-scheduler-minikube                     1/1     Running     1          160m
storage-provisioner                         1/1     Running     2          160m

 ~/Workspace/k8s/guide/ingress ····· at ⎈ minikube
❯ kubectl exec -it ingress-nginx-controller-558664778f-5gzlf -- sh
Error from server (NotFound): pods "ingress-nginx-controller-558664778f-5gzlf" not fo
und

 ~/Workspace/k8s/guide/ingress ····· at ⎈ minikube
❯ kubectl -n kube-system exec -it ingress-nginx-controller-558664778f-5gzlf -- sh
/etc/nginx $ ls
fastcgi.conf            mime.types              scgi_params
fastcgi.conf.default    mime.types.default      scgi_params.default
fastcgi_params          modsecurity             template
fastcgi_params.default  modules                 uwsgi_params
geoip                   nginx.conf              uwsgi_params.default
koi-utf                 nginx.conf.default      win-utf
koi-win                 opentracing.json
lua                     owasp-modsecurity-crs
/etc/nginx $ cd /etc/^C
/etc/nginx $ cat nginx.conf
```

```
## end server v1.echo.192.168.64.9.sslip.io

## start server v2.echo.192.168.64.9.sslip.io
server {
        server_name v2.echo.192.168.64.9.sslip.io ;

        listen 80 ;
        listen 443  ssl http2 ;
```

해당 주소의 80번 포트로 들어왔을 때

```
    set $balancer_ewma_score -1;
    set $proxy_upstream_name "default-echo-v2-3000";
    set $proxy_host          $proxy_upstream_name;
    set $pass_access_scheme  $scheme;

    set $pass_server_port    $server_port;

    set $best_http_host      $http_host;
    set $pass_port           $pass_server_port;

    set $proxy_alternative_upstream_name "";
```

Ingress는 도메인, 경로만 연동하는 것이 아니라 요청 timeout, 요청 max size 등 다양한 프록시 서버 설정을 할 수 있습니다.

Ingress를 사용하면 YAML 설정만으로 도메인, 경로 설정을 손쉽게 할 수 있습니다. 기존에 도메인을 연결하려면 담당자에게 요청하고 설정 파일을 변경한 다음 프로세스 재시작까지 수동으로 작업했는데, 더 이상 그런 과정을 거치지 않아도 됩니다.

▼ 🐣 **Volume (local)**

⚡ **목표**

Pod 안의 컨테이너 간 디렉토리를 공유하는 방법과 컨테이너의 특정 디렉토리를 호스트 디렉토리와 연결하는 방법을 알아봅니다.

지금까지 만들었던 컨테이너는 **Pod을 제거하면 컨테이너 내부에 저장했던 데이터도 모두 사라집니다.** MySQL과 같은 데이터베이스는 데이터가 유실되지 않도록 반드시 별도의 저장소에 데이터를 저장하고 컨테이너를 새로 만들 때 이전 데이터를 가져와야 합니다.

쿠버네티스는 Volume을 이용하여 컨테이너의 디렉토리를 외부 저장소와 연결하고 다양한 플러그인을 지원하여 흔히 사용하는 대부분의 스토리지를 별도 설정없이 사용할 수 있습니다.

실전에서는 awsElasticBlockStore(aws), azureDisk(azure), gcePersistentDisk(google cloud)와 같은 volume을 사용하지만 이를 테스트하기 위해서는 실제 클라우드를 사용해야 하므로 이번엔 간단하게 로컬 저장소를 사용하는 법만 알아봅니다.

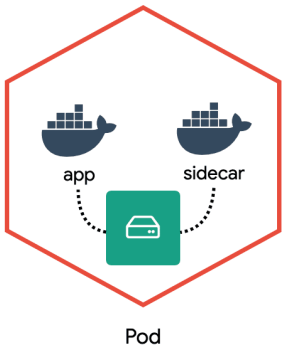
> 💡 **PV/PVC**
>
> 데이터 저장이 필요한 경우에 흔히 Persistent Volume(PV), Persistent Volume Claim(PVC)를 사용합니다. 이 내용은 실제 클라우드 설정 후 테스트할 예정입니다.

▼ 🎀 **Volume 만들기**

▼ 👻 **empty-dir**

Pod 안에 속한 **컨테이너 간 디렉토리를 공유**하는 방법을 알아봅니다.

보통 **사이드카sidecar라는 패턴**에서 사용합니다. 예를 들면, **특정 컨테이너에서 생성되는 로그 파일을 별도의 컨테이너(사이드카)가 수집** 할 수 있습니다.



`app` 컨테이너는 `/var/log/example.log`에 로그 파일을 만들고 `sidecar` 컨테이너는 해당 로그 파일을 처리하도록 합니다.

app에서 나오는 로그를 특정 디렉토리에 저장하고 sidecar가 공유함 → sidecar가 로그파일을 읽어서 저장/가공 가능

▼ **guide/local-volume/empty-dir.yml**

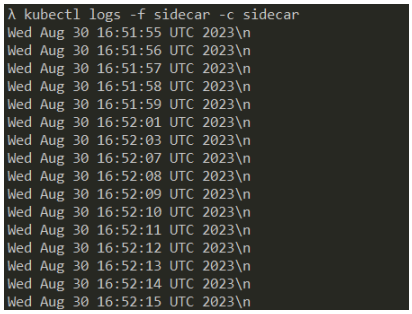
```
apiVersion: v1
kind: Pod
metadata:
  name: sidecar
```

```
spec:
  containers:
    - name: app #로그 생성
      image: busybox
      args:
        - /bin/sh
        - -c
        - >
          while true;
          do
            echo "$(date)\n" >> /var/log/example.log;
            sleep 1;
          done
      volumeMounts: #마운트
        - name: varlog
          mountPath: /var/log
    - name: sidecar #로그를 읽음
      image: busybox
      args: [/bin/sh, -c, "tail -f /var/log/example.log"]
      volumeMounts:
        - name: varlog
          mountPath: /var/log
  volumes: #하나의 pod안에 가상의 디렉토리 생기고 두 pod안에서 서로 공유됨
    - name: varlog
      emptyDir: {}
```

배포 후 `sidecar`의 로그를 확인합니다.

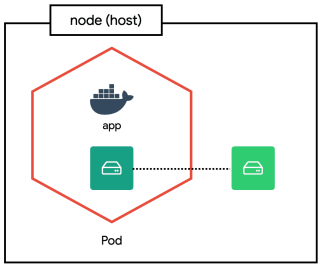
```
kubectl apply -f empty-dir.yml

# sidecar 로그 확인
kubectl logs -f sidecar -c sidecar
```

`app` 컨테이너에서 생성한 로그파일을 `sidecar` 컨테이너에서 처리하는 모습을 볼 수 있습니다.



▼ 👻 hostpath

호스트 디렉토리를 컨테이너 디렉토리에 연결하는 방법을 알아봅니다. 여기서는 **호스트의 `/var/log` 디렉토리를 연결**하여 내용을 확인해 보겠습니다.



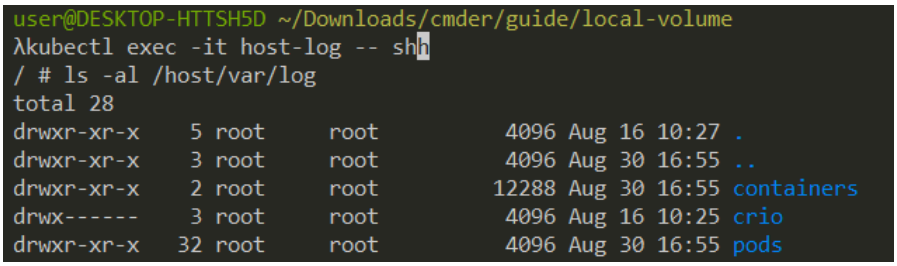
호스트의 `/var/log`를 컨테이너의 `/host/var/log` 디렉토리로 마운트합니다.

▼ guide/local-volume/hostpath.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: host-log
spec:
  containers:
    - name: log
      image: busybox
      args: ["/bin/sh", "-c", "sleep infinity"]
      volumeMounts:
        - name: varlog
          mountPath: /host/var/log
  volumes:
    - name: varlog
      hostPath:
        path: /var/log
```

컨테이너에서 마운트 된 디렉토리를 확인합니다.

```
kubectl apply -f hostpath.yml

# 컨테이너 접속 후 /host/var/log 디렉토리를 확인
kubectl exec -it host-log -- sh
ls -al /host/var/log
```



pod에서 만든 로그가 아닌 host(node)에서 만든 로그

▼ 🐣 ConfigMap

⚡ 목표

쿠버네티스에서 설정파일과 환경변수를 관리하는 방법을 알아봅니다.

컨테이너에서 **설정 파일을 관리**하는 방법은 이미지를 빌드할 때 복사하거나, 컨테이너를 실행할 때 외부 파일을 연결하는 방법이 있습니다. **쿠버네티스는** `ConfigMap` **으로 설정을 관리**합니다.

▼ 🎀 ConfigMap 만들기

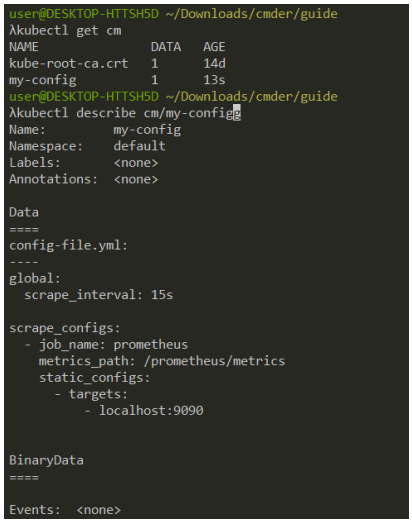
파일을 통째로 ConfigMap으로 만든 다음 컨테이너에서 사용하는 방법을 알아봅니다.

▼ guide/configmap/config-file.yml

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: prometheus
    metrics_path: /prometheus/metrics
    static_configs:
      - targets:
          - localhost:9090
```

먼저, ConfigMap을 만듭니다. `--from-file` **옵션을 이용하여 file을 설정**으로 만듭니다.

```
# ConfitMap 생성 configmap -> cm
kubectl create cm my-config --from-file=config-file.yml

# ConfitMap 조회
kubectl get cm

# ConfigMap 내용 상세 조회
kubectl describe cm/my-config
```



생성한 ConfigMap을 `/etc/config` **디렉토리에 연결**합니다.

▼ **guide/configmap/alpine.yml**

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
    - name: alpine
      image: alpine
      command: ["sleep"]
      args: ["100000"]
      volumeMounts:
        - name: config-vol
          mountPath: /etc/config
  volumes:
    - name: config-vol
      configMap: #cm에 있는 파일 내용을 마운트
        name: my-config
```

volume을 연결하여 배포하고 확인합니다.

```
kubectl apply -f alpine.yml

# 접속 후 설정 확인
kubectl exec -it alpine -- ls /etc/config
kubectl exec -it alpine -- cat /etc/config/config-file.yml

# or
kubectl exec -it alpine -- sh
cd /etc/config
ls -al
cat config-file.yml
```

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/configmap
λkubectl exec -it alpine -- ls /etc/config
ls: C:/Users/user/Downloads/cmder/vendor/git-for-windows/etc/config: No such file or di
rectory
command terminated with exit code 1
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/configmap
λ kubectl exec -it alpine -- sh
/ # cd /etc/config
/etc/config # ls -al
total 12
drwxrwxrwx    3 root     root          4096 Aug 30 17:51 .
drwxr-xr-x    1 root     root          4096 Aug 30 17:51 ..
drwxr-xr-x    2 root     root          4096 Aug 30 17:51 ..2023_08_30_17_51_01.39903896
9
lrwxrwxrwx    1 root     root            31 Aug 30 17:51 ..data -> ..2023_08_30_17_51_0
1.399038969
lrwxrwxrwx    1 root     root            22 Aug 30 17:51 config-file.yml -> ..data/conf
ig-file.yml
/etc/config # cat config-file.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: prometheus
    metrics_path: /prometheus/metrics
    static_configs:
      - targets:
          - localhost:9090
```

▼ 🎀 **env 파일로 만들기**

env 포멧을 그대로 사용합니다.

▼ **guide/configmap/config-env.yml**

```
hello=world
haha=hoho
```

`env-config`로 만듭니다.

```
# env 포멧으로 생성
kubectl create cm env-config --from-env-file=config-env.yml

# env-config 조회
kubectl describe cm/env-config
```

▼ 🎀 **YAML 선언하기**

ConfigMap을 YAML파일로 정의합니다.

▼ **guide/configmap/config-map.yml**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  hello: world
  kuber: netes
  multiline: |-
    first
    second
    third
```

`config-map.yml` 적용 후 마운트 된 내용을 확인합니다.

```
# 기존 configmap 삭제
kubectl delete cm/my-config

# configmap 생성
kubectl apply -f config-map.yml

# alpine 적용
kubectl apply -f alpine.yml

# 적용내용 확인
kubectl exec -it alpine -- cat /etc/config/multiline
```

▼ 🎀 **ConfigMap을 환경변수로 사용하기**

ConfigMap을 volume이 아닌 환경변수로 설정합니다.

▼ **guide/configmap/alpine-env.yml**

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine-env
spec:
  containers:
    - name: alpine
      image: alpine
      command: ["sleep"]
      args: ["100000"]
      env:
        - name: hello
          valueFrom:
            configMapKeyRef:
              name: my-config
              key: hello
```

환경변수를 확인합니다.

```
kubectl apply -f alpine-env.yml

# env 확인
kubectl exec -it alpine-env -- env
```

```
λ kubectl exec -it alpine-env -- env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=alpine-env
TERM=xterm
hello=world
ECHO_V1_PORT=tcp://10.105.46.76:3000
ECHO_V1_PORT_3000_TCP=tcp://10.105.46.76:3000
ECHO_V1_PORT_3000_TCP_ADDR=10.105.46.76
REDIS_PORT_6379_TCP_PORT=6379
KUBERNETES_SERVICE_PORT_HTTPS=443
ECHO_V1_PORT_3000_TCP_PORT=3000
REDIS_PORT_6379_TCP_PROTO=tcp
COUNTER_NP_PORT_3000_TCP=tcp://10.99.116.11:3000
```

▼ 🐣 **Secret**

⚡ **목표**

쿠버네티스에서 비밀번호, SSH 인증, TLS Secret과 같은 보안 정보를 관리하는 방법을 알아봅니다.

쿠버네티스는 `ConfigMap`과 유사하지만, **보안 정보를 관리하기 위해** `Secret`**을 별도로 제공**합니다. ConfigMap과 차이점은 **데이터가** `base64`**로 저장된다는 점** 말고는 거의 없습니다.

> 💡 **Secret은 암호화되지 않음**
>
> Secret은 보안 정보를 다루기 때문에 당연히 암호화될 거라고 생각할 수 있지만, **실제로는 그대로 저장**됩니다. 따라서, **etcd에 접근이 가능하다면 누구나 저장된 Secret을 확인**할 수 있습니다. **vault**와 같은 외부 솔루션을 이용하여 보안을 강화할 수 있습니다.

▼ 🎀 Secret 만들기

아이디와 패스워드를 Secret으로 저장하고 컨테이너에서 환경변수로 사용하는 방법을 알아봅니다.

- **guide/secret/username.txt**

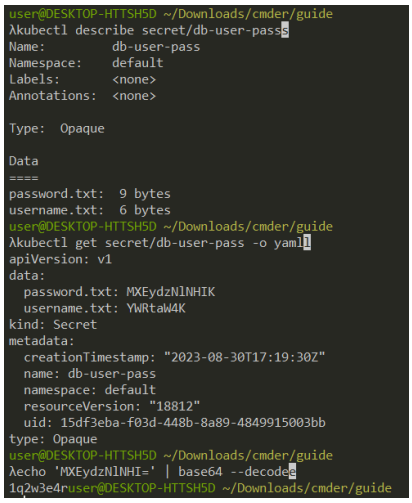
  `admin`

- **guide/secret/password.txt**

  `1q2w3e4r`

Secret을 만들고 확인해봅니다.

```
# secret 생성
kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt

# secret 상세 조회
kubectl describe secret/db-user-pass

# -o yaml로 상세 조회
kubectl get secret/db-user-pass -o yaml

# 저장된 데이터 base64 decode -> 인코딩 된 글자 디코딩 (암호화 되지 않는다...)
echo 'MXEydzNlNHI=' | base64 --decode
```



설정한 Secret을 환경변수로 연결합니다.

▼ **guide/secret/alpine-env.yml**

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine-env
spec:
  containers:
    - name: alpine
      image: alpine
      command: ["sleep"]
      args: ["100000"]
```

```
    env:
      - name: DB_USERNAME
        valueFrom:
          secretKeyRef:
            name: db-user-pass
            key: username.txt
      - name: DB_PASSWORD
        valueFrom:
          secretKeyRef:
            name: db-user-pass
            key: password.txt
```

환경변수를 확인합니다.

```
kubectl apply -f alpine-env.yml

# env 확인
kubectl exec -it alpine-env -- env
```