8. 애플리케이션에서 포드 메타데이터와 그 외의 리소스에 접근하기

류경윤

8장에서 다루는 내용

1. Downward API 사용 방법
2. 쿠버네티스 REST API 탐색
3. kubeclt proxy 사용법
4. 컨테이너 내에서 API 서버에 접근하는 방법
5. 앰배서더 컨테이너 패턴
6. 쿠버네티스의 클라이언트 라이브러리

특정 포드및 컨테이너 메타 데이터를 컨테이너에 전달하는 방법

Downward API를 통한 메타데이터 전달

- ConfigMap 그리고 Secret Volume은 사용자가 직접 설정하고 포드가 노드로 예약되어 실행되기 전에 이미 알고 있는 데이터에 적합
- 실행되기 전까지 알 수 없는 데이터는 어떻게 ? ex) Pod IP, Host Node Name, Pod Name
- Solution: **Downward API**
  - 환경 변수
  - 파일(API 볼륨)을 통해 포드 및 환경에 대한 메타 데이터를 전달

사용 가능한 메타 데이터

- 포드
  - 이름
  - IP 주소
  - 네임스페이스
  - (실행되고 있는) 노드 이름
  - (실행되고 있는) 서비스 계정 이름
  - 라벨
  - 주석
- 컨테이너
  - CPU 및 메모리 요청
  - CPU 및 메모리 한계

환경 변수를 통한 메타 데이터 노출

Listing 8.1 Downward API used in environment variables: downward-api-env.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: downward
spec:
  containers:
  - name: main
    image: busybox
    command: ["sleep", "9999999"]
    resources:
      requests:
        cpu: 15m
        memory: 100Ki
      limits:
        cpu: 100m
        memory: 4Mi
    env:
    - name: POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: SERVICE_ACCOUNT
      valueFrom:
        fieldRef:
          fieldPath: spec.serviceAccountName
    - name: CONTAINER_CPU_REQUEST_MILLICORES
      valueFrom:
        resourceFieldRef:
          resource: requests.cpu
          divisor: 1m
    - name: CONTAINER_MEMORY_LIMIT_KIBIBYTES
      valueFrom:
        resourceFieldRef:
          resource: limits.memory
          divisor: 1Ki
```

Instead of specifying an absolute value, you're referencing the metadata.name field from the pod manifest.

A container's CPU and memory requests and limits are referenced by using resourceFieldRef instead of fieldRef.

For resource fields, you define a divisor to get the value in the unit you need.

환경 변수를 통한 메타 데이터 노출 결과 확인:

Listing 8.2 Environment variables in the downward pod

```
$ kubectl exec downward env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=downward
CONTAINER_MEMORY_LIMIT_KIBIBYTES=4096
POD_NAME=downward
POD_NAMESPACE=default
POD_IP=10.0.0.10
NODE_NAME=gke-kubia-default-pool-32a2cac8-sgl7
SERVICE_ACCOUNT=default
CONTAINER_CPU_REQUEST_MILLICORES=15
KUBERNETES_SERVICE_HOST=10.3.240.1
KUBERNETES_SERVICE_PORT=443
...
```

Downward API 볼륨 내의 파일을 통한 메타 데이터 전달

Listing 8.3 Pod with a `downwardAPI` volume: downward-api-volume.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: downward
  labels:
    foo: bar
  annotations:
    key1: value1
    key2: |
      multi
      line
      value
spec:
  containers:
  - name: main
    image: busybox
    command: ["sleep", "9999999"]
    resources:
      requests:
        cpu: 15m
        memory: 100Ki
      limits:
        cpu: 100m
        memory: 4Mi
    volumeMounts:
    - name: downward
      mountPath: /etc/downward
  volumes:
  - name: downward
    downwardAPI:
      items:
      - path: "podName"
        fieldRef:
          fieldPath: metadata.name
      - path: "podNamespace"
        fieldRef:
          fieldPath: metadata.namespace
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "annotations"
        fieldRef:
          fieldPath: metadata.annotations
      - path: "containerCpuRequestMilliCores"
        resourceFieldRef:
          containerName: main
          resource: requests.cpu
          divisor: 1m
      - path: "containerMemoryLimitBytes"
        resourceFieldRef:
          containerName: main
          resource: limits.memory
          divisor: 1
```

These labels and annotations will be exposed through the downwardAPI volume.

You're mounting the downward volume under /etc/downward.

You're defining a downwardAPI volume with the name downward.

The pod's name (from the metadata.name field in the manifest) will be written to the podName file.

The pod's labels will be written to the /etc/downward/labels file.

The pod's annotations will be written to the /etc/downward/annotations file.

Downward API 볼륨 내의 파일을 통한 메타 데이터 전달 결과 확인:

Listing 8.4 Files in the downwardAPI volume

```
$ kubectl exec downward ls -lL /etc/downward
-rw-r--r--    1 root    root     134 May 25 10:23 annotations
-rw-r--r--    1 root    root       2 May 25 10:23 containerCpuRequestMilliCores
-rw-r--r--    1 root    root       7 May 25 10:23 containerMemoryLimitBytes
$ kubectl exec downward cat /etc/downward/labels
foo="bar"

$ kubectl exec downward cat /etc/downward/annotations
key1="value1"
key2="multi\nline\nvalue\n"
kubernetes.io/config.seen="2016-11-28T14:27:45.664924282Z"
kubernetes.io/config.source="api"
```

라벨과 주석 업데이트

- 쿠버네티스는 라벨과 주석이 업데이트 되면, 파일을 업데이트하여 포드가 항상 최선 데이터를 볼 수 있도록 한다.
- 환경 변수 값은 업데이트 될 수 없으므로 포드의 라벨과 주석을 환경 변수를 통해 노출하면 수정된후 최신 데이터로 업데이트 되지 않는다.

볼륨 스펙에서 컨테이너-라벨 메타 데이터 참조

Listing 8.6 Referring to container-level metadata in a `downwardAPI` volume

```
spec:
  volumes:
  - name: downward
    downwardAPI:
      items:
      - path: "containerCpuRequestMilliCores"
        resourceFieldRef:
          containerName: main
          resource: requests.cpu
          divisor: 1m
```

Container name must be specified

Downward API 사용시 이해해야할 점

- 애플리케이션을 다시 작성하거나 쉘 스크립트로 붙여 넣지 않고도 애플리케이션에 데이터를 노출시킬 수 있다.
- 포드가 실행 된후 수집된 데이터를 환경 변수를 통해 표시 할 수 있다.
- 단점:
  - 사용 가능한 메타데이터가 상당히 제한적
  - Solution: 쿠버네티스 API서버에서 직접 데이터를 구해야 함

쿠버네티스 API 서버와 통신

- Downward API가 제공하는 방법은 포드 자체의 메타 데이터와 모든 포드 데이터의 하위 집합만 노출
- 애플리케이션이 다른 포드와 클러스터에 정의된 리소스를 알아야 한다면 ?
  -

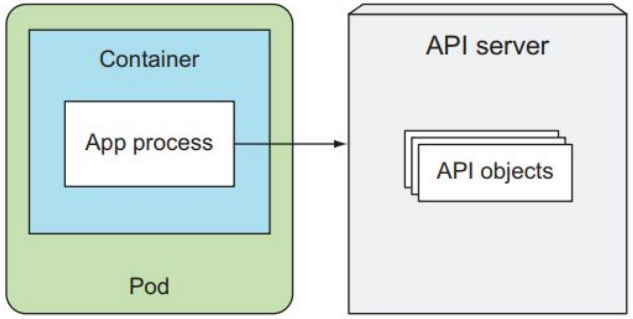

Figure 8.4 Talking to the API server from inside a pod to get information about other API objects

쿠버네티스 REST API 탐색

API 서버 URL:

```
$ kubectl cluster-info
Kubernetes master is running at https://192.168.99.100:8443
```

서버는 HTTPS 사용하고 인증이 필요함.

요청 결과:

```
$ curl https://192.168.99.100:8443 -k
Unauthorized
```

KUBECTL PROXY를 통한 API서버 접근

인증을 직접 처리하는 대신 **kubectl proxy** 명령을 실행해 프록시를 통해 서버와 통신 할 수 있음

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

요청 결과:

```
$ curl localhost:8001
{
  "paths": [
    "/api",
    "/api/v1",
    ...
```

API 탐험(예제 같이 보기)

Ex)

- kubectl proxy를 통한 쿠버네티스 API 탐험
- 배치 API 그룹의 REST 엔드포인트 탐험
- 클러스터에 있는 모든 잡 인스턴스 목록
- 이름으로 특정한 잡 인스턴스 가져오기

포드 내에서 API 서버와 통신

- API서버와 통신 하려면
  - API 서버 주소 찾기
  - 서버의 신원 검증
  - API 서버로 인증

API 서버와의 통신을 시도해서 포드 실행하기

Shell 사용할 Pod 만들기(curl 설치된 컨테이너)

Listing 8.12 A pod for trying out communication with the API server: curl.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: curl
spec:
  containers:
  - name: main
    image: tutum/curl
    command: ["sleep", "9999999"]
```

Using the tutum/curl image, because you need curl available in the container

You're running the sleep command with a long delay to keep your container running.

Shell 실행:

```
$ kubectl exec -it curl bash
root@curl:/#
```

API 서버와의 통신을 시도해서 포드 실행하기

Shell 사용할 Pod 만들기(curl 설치된 컨테이너)

Listing 8.12 A pod for trying out communication with the API server: curl.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: curl
spec:
  containers:
  - name: main
    image: tutum/curl
    command: ["sleep", "9999999"]
```

Using the tutum/curl image, because you need curl available in the container

You're running the sleep command with a long delay to keep your container running.

Shell 실행:

```
$ kubectl exec -it curl bash
root@curl:/#
```

API 서버 주소 찾기

kubectl get svc 사용해서 찾기

```
$ kubectl get svc
NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1     <none>        443/TCP   46d
```

환경변수 사용해서 찾기

- p210 5장
- 쿠버네티스는 포드가 시작되면 그 순간 존재하는 각 서비스를 가리키는 환경 변수 세트를 초기화 한다

```
root@curl:/# env | grep KUBERNETES_SERVICE
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_HOST=10.0.0.1
KUBERNETES_SERVICE_PORT_HTTPS=443
```

DNS 사용해서 찾기

```
root@curl:/# curl https://kubernetes
curl: (60) SSL certificate problem: unable to get local issuer certificate
...
If you'd like to turn off curl's verification of the certificate, use
  the -k (or --insecure) option.
```

서버의 신원 검증

- 시크릿을 설명하면서 default-token-xyz라는 자동 생성 시크릿을 살펴 봤다.
- default-token-xyz 시크릿은 각 컨테이너의 /var/run/secrets/kubernetes.io/serviceaccount에 마운트 된다

```
root@curl:/#ls/var/run/secrets/kubernetes.io/serviceaccount/
ca.crt     namespace     token
```

-

서버의 신원 검증 (CA)

- API서버와 통신중인지 확인하려면 서버의 인증서가 CA에 의해 서명되었는지 확인해야 한다.
  - curl --cacert 옵션 사용해서 CA인증서 지정 or CURL_CA_BUNDLE 환경변수를 설정

```
root@curl:/# curl --cacert /var/run/secrets/kubernetes.io/serviceaccount
             ➥ /ca.crt https://kubernetes
Unauthorized
```

```
root@curl:/# export CURL_CA_BUNDLE=/var/run/secrets/kubernetes.io/
             ➥ serviceaccount/ca.crt
```

```
root@curl:/# curl https://kubernetes
Unauthorized
```

클라이언트가 서버를 신뢰 하지만, 아직 서버는 클라이언트를 구분할 토큰이 없어서 권한을 주지 않음

API서버로 인증 (Token)

- 서버에서 인증해야 클라이언트는 클러스터에 배포된 API객체를 사용 할 수 있다
- 인증을 위해선 토큰(token)이 필요하다
- 토큰 파일은 비밀 볼륨에 저장되어 있다

API서버로 인증 (Token)

환경 변수에 토큰 설정

```
root@curl:/# TOKEN=$(cat /var/run/secrets/kubernetes.io/
             ➥ serviceaccount/token)
```

Authorization HTTP 헤더 안에 토큰을 전달

Listing 8.13 Getting a proper response from the API server

```
root@curl:/# curl -H "Authorization: Bearer $TOKEN" https://kubernetes
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1beta1",
    "/apis/authorization.k8s.io",
    ...
    "/ui/",
    "/version"
  ]
}
```

실행중인 포드의 네임스페이스 가져오기

- 시크릿 볼륨에 네임스페이스 파일
- 포드가 실행중인 네임스페이스가 포함되어 있으므로 환경 변수를 통해 전달할 필요 없음

앰배서더 컨테이너와 API 서버통신 간소화

- kubectl proxy를 사용하여 인증, 암호화 및 서버확인을 대시 처리 하도록함
- 포드 내부에서도 사용 가능

= 엠배서더 컨테이너 패턴

앰배서더 컨테이너 패턴 소개

- API서버와 직접 통신하는 대신 메인 컨테이너 옆에 앰배서더 컨테이너에서 kubectl-proxy를 실행하고 kubectl-proxy를 사용하여 API서버와 통신함
- 주 컨테이너의 Application은 HTTP를 사용하여 앰배서더와 연결
- 앰배서더 프록시가 API서버와 HTTPS로 연결하여 처리
- 포드의 모든 컨테이너는 동일한 루프백 네트워크 인터페이스를 공유하므로 애플리케이션은 localhost 의 포트를 통해 프록시에 엑세스 할 수 있다
-

앰배서더 컨테이너 패턴 소개

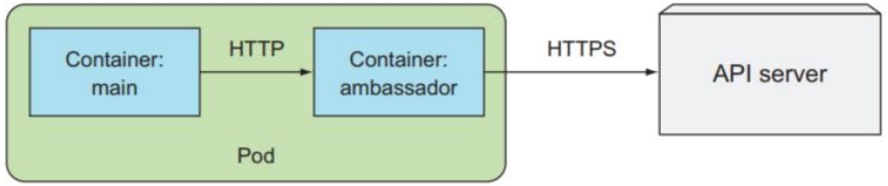

Figure 8.6 Using an ambassador to connect to the API server

Listing 8.15 A pod with an ambassador container: curl-with-ambassador.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: curl-with-ambassador
spec:
  containers:
  - name: main
    image: tutum/curl
    command: ["sleep", "9999999"]
  - name: ambassador
    image: luksa/kubectl-proxy:1.6.2
```

The ambassador container, running the kubectl-proxy image

Listing 8.16 Accessing the API server through the ambassador container

```
root@curl-with-ambassador:/# curl localhost:8001
{
  "paths": [
    "/api",
    ...
  ]
}
```

기본적으로 kubectl proxy 포트 8001에 바인딩되며 루프백을 포함한 모든 네트워크 인터페이스를 공유함 따라서 아래와 같이 실행 가능

앰배서더 컨테이너 패턴 장점 / 단점

장점:

- 외부 서비스에 연결하는 복잡성을 숨김
- 주 컨테이너에서 실행되는 애플리케이션을 단순화 함
- 애플리케이션의 언어와 관계없이 재사용 가능

단점:

- 추가 프로세스가 실행되고 추가 리소스가 필요함

클라이언트 라이브러리를 사용해 API서버와 통신

- 간단한 API요청 이상을 수행하기 위해 기존 쿠버네티스 API클라이언트 라이브러리중 하나 사용 가능
- ex)
  - 고랭 클라이언트
  - 파이썬
  - etc
- 대부분 라이브러리는 https를 지원하고 인증을 처리함
  - 앰배서더 컨테이너 필요 없음

스웨거와 OpenAPI를 사용해 자신만의 라이브러리 구축

https://swagger.io

감사합니다