chap 08. 애플리케이션에서 파드 메타데이터와 그외의 리소스에 액세스하기

DevOps 풀잎스쿨
조준구

8.0 다루는 내용

8.0 다루는 내용

- 컨테이너에 정보를 전달하기 위해 Downward API 사용
- 쿠버네티스 REST API 살펴보기
- 인증과 서버 검증을 kubectl proxy에 맡기기
- 컨테이너 내에서 API 서버에 접근하기
- 앰배서더 컨테이너 패턴의 이해
- 쿠버네티스 클라이언트 라이브러리 사용

8.0 다루는 내용

- 8장에서는
  - 특정 파드와 컨테이너 메타데이터를 컨테이너로 전달하는 방법과
  - 컨테이너 내에서 실행 중인 애플리케이션이 쿠버네티스 API 서버와 통신해
    - 클러스터에 배포된 리소스의 정보를 얻는 것이 얼마나 쉬운지를
  - 더 나아가 이런 리소스를 생성하거나 수정하는 방법을 다룸

8.1 Downward API로 메타데이터 전달

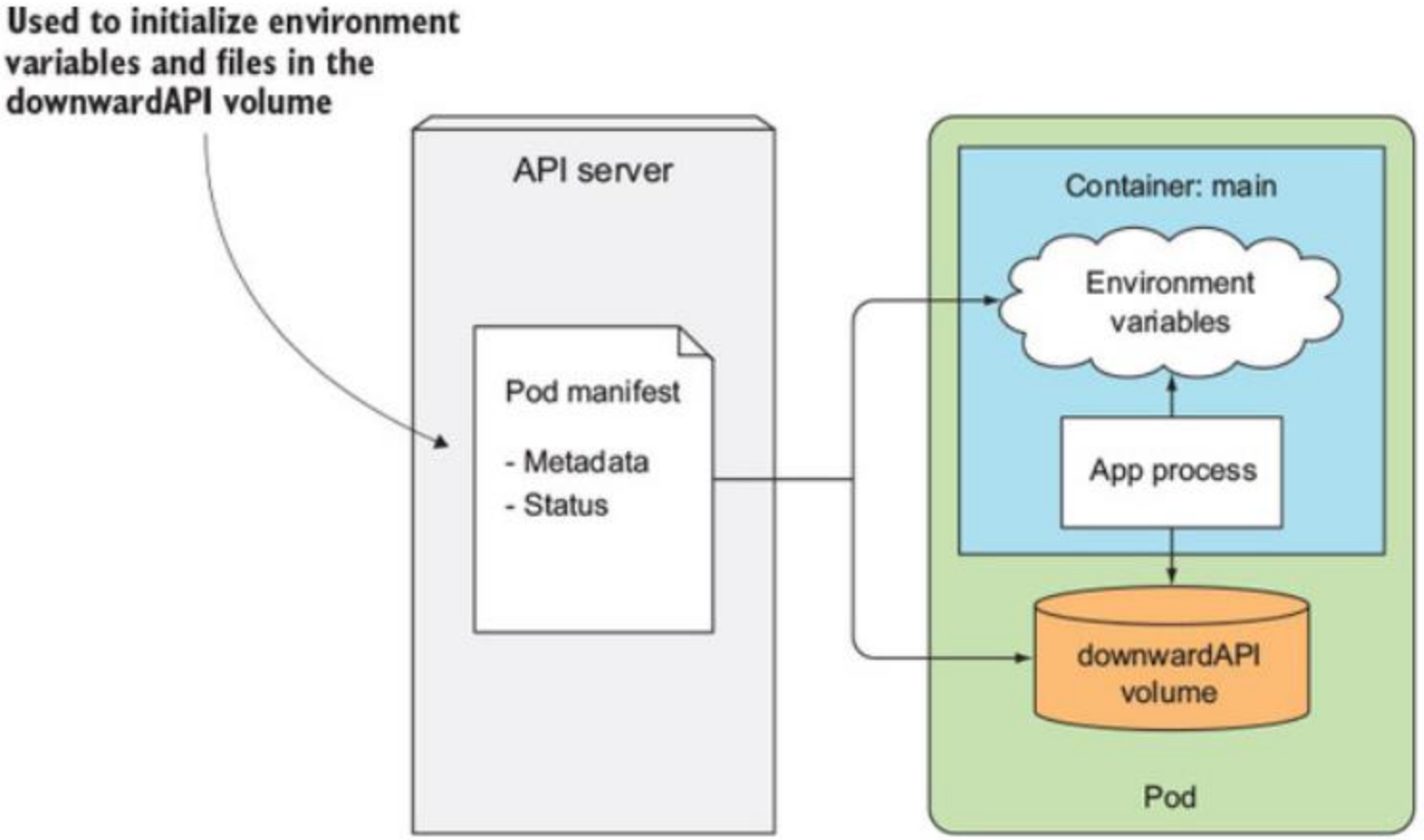
- 메타데이터
  - 파드의 IP
  - 호스트 노드 이름
  - 파드 자체의 이름
  - 파드의 레이블
  - 어노테이션 등

- 환경변수 또는
- (downward API 볼륨 내에 있는)파일로

- 파드와 해당 환경의 메타데이터를 전달(노출)할 수 있음

8.1 Downward API로 메타데이터 전달

Figure 8.1. The Downward API exposes pod metadata through environment variables or files.



8.1 Downward API로 메타데이터 전달

- 컨테이너에 전달 가능한 정보들
  - 파드의 이름
  - 파드의 IP 주소
  - 파드가 속한 네임스페이스
  - 파드가 실행 중인 노드의 이름
  - 파드가 실행 중인 서비스 어카운트 이름
    - 서비스 어카운트는 12장에서 다룸

      → (대략적으로) 파드가 API 서버와 통실할 때 인증하는 계정
  - 각 컨테이너의 CPU와 메모리 요청
  - 각 컨테이너의 CPU와 메모리 제한
    - CPU/메모리 요청 및 제한은 14장에서 다룸

      → 컨테이너에 보장되는 CPU와 메모리의 양과 컨테이너가 얻을 수 있는 최대 양
  - 파드의 레이블
  - 파드의 어노테이션

레이블과 어노테이션은 볼륨으로만 노출

8.1 Downward API로 메타데이터 전달

8.1.2 환경변수로 메타데이터 노출하기

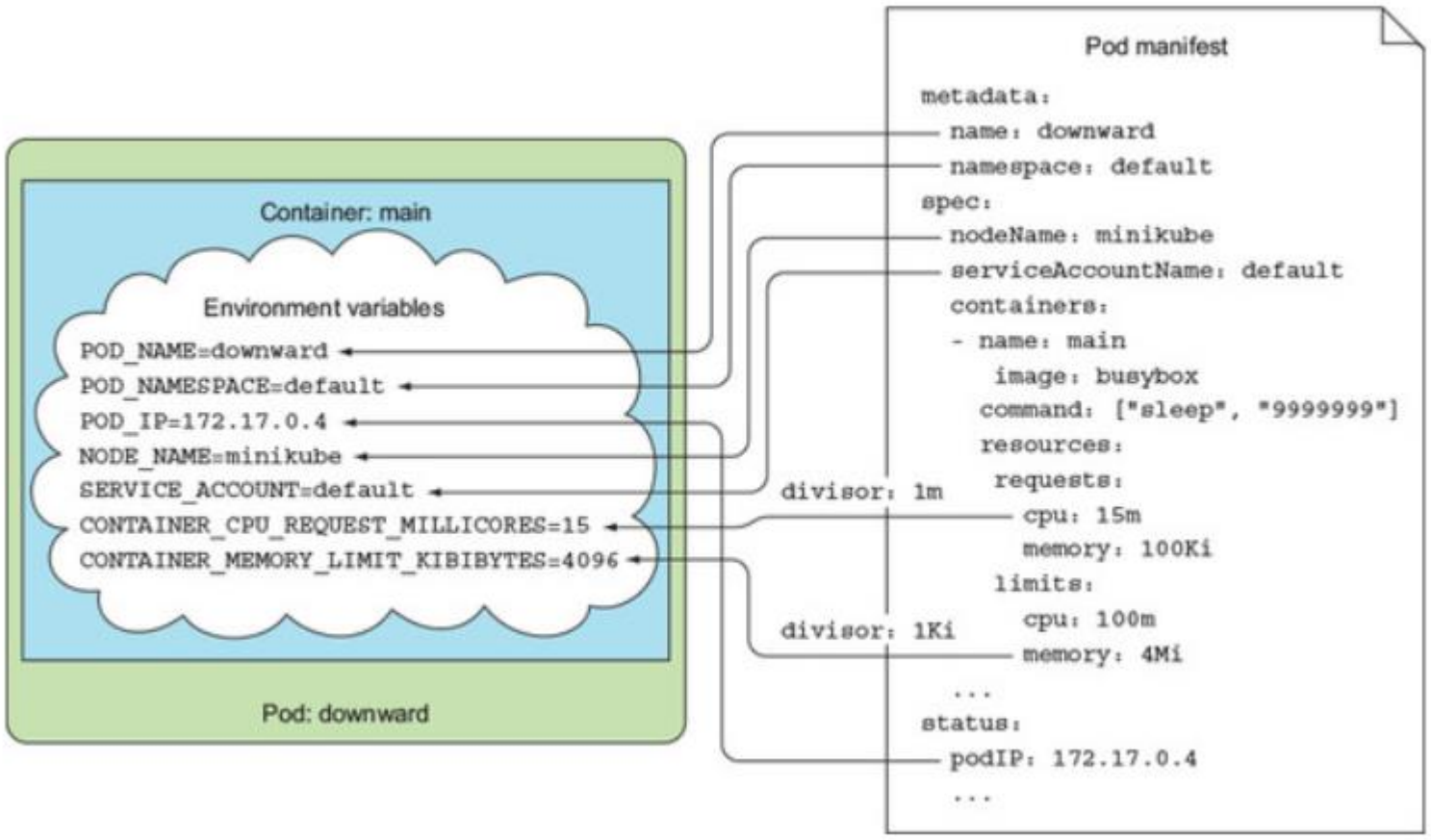
- 환경변수로 파드와 컨테이너의 메타데이터를
  - 컨테이너에 전달하는 방법을 살펴봄

```yaml
apiVersion: v1
kind: POd
metadata:
  name: downward
spec:
  container:
  - name: main
    image: busybox
    command: ["sleep", "9999999"]
    resources:
      requests:
        cpu: 15m
        memory: 100Ki
      limits:
        cpu: 100m
        memory: 4Mi
```

```yaml
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
```

8.1 Downward API로 메타데이터 전달

Figure 8.2. Pod metadata and attributes can be exposed to the pod through environment variables.



8.1 Downward API로 메타데이터 전달

8.1.3 downward API 볼륨에 파일로 메타데이터 전달

- 환경변수 대신 파일로 메타데이터를 노출하려는 경우
  - downward API 볼륨을 정의해 컨테이너에 마운트 할 수 있다

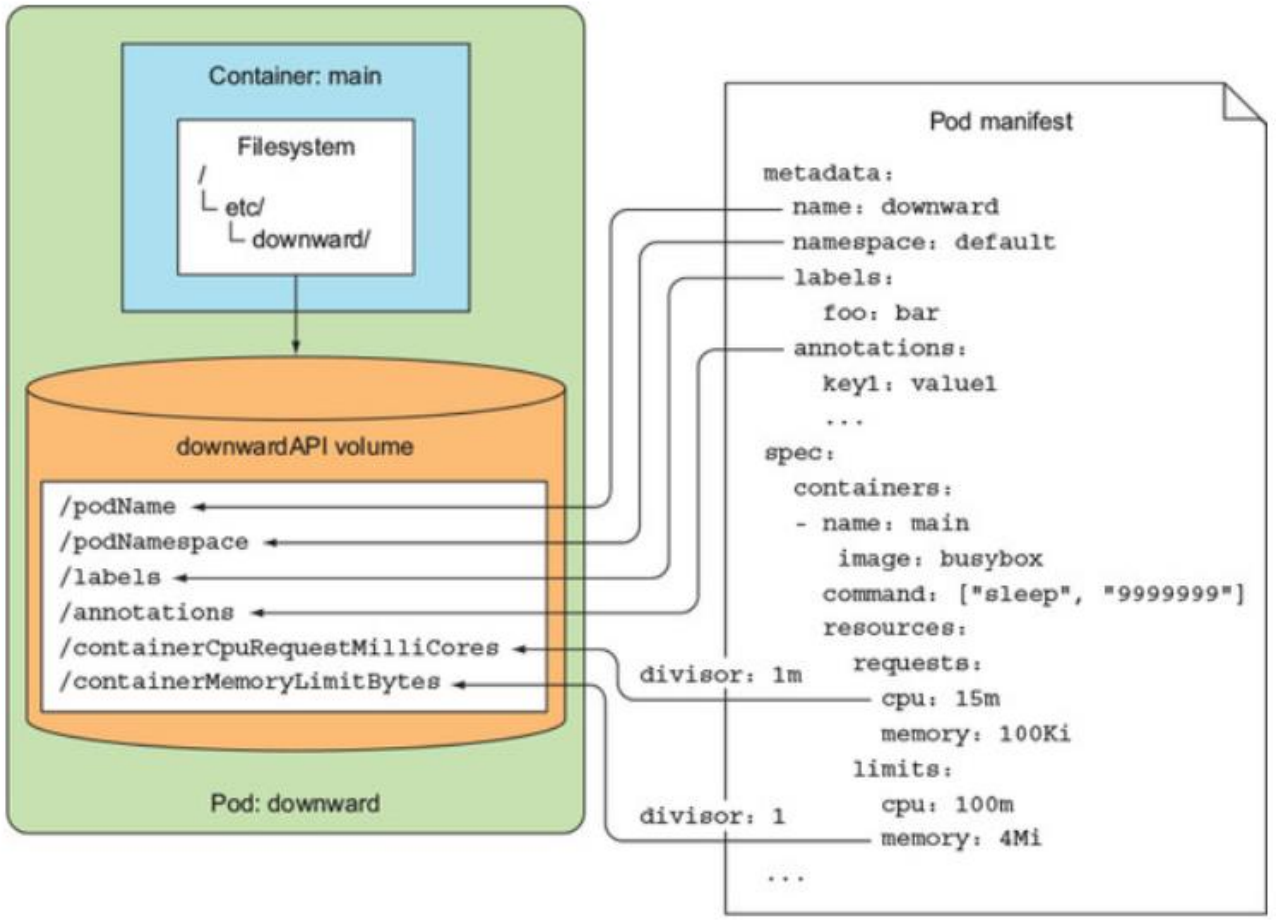
- 환경변수로 메타데이터를 전달하는 대신
  - downward라는 볼륨을 정의하고
  - 컨테이너의 /etc/downward 아래에 마운트한다.
  - 이 볼륨에 포함된 파일들은
    - 볼륨 스펙의 downwardAPI.items 속성 아래에 설정된다.

```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
```

```yaml
  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
```

8.1 Downward API로 메타데이터 전달

Figure 8.3. Using a `downwardAPI` volume to pass metadata to the container



8.1 Downward API로 메타데이터 전달

- 새로 파드를 생성한 뒤, /etc/downward 아래의 파일을 나열해보자

```
kubectl exec downward ls -lL /etc/downward
```

JavaScript

- result

```
-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
-rw-r--r--   1 root   root     9 May 25 10:23 labels
-rw-r--r--   1 root   root     8 May 25 10:23 podName
-rw-r--r--   1 root   root     7 May 25 10:23 podNamespace
```

8.1 Downward API로 메타데이터 전달

- 컨테이너에 전달 가능한 정보들
  - 파드의 이름
  - 파드의 IP 주소
  - 파드가 속한 네임스페이스
  - 파드가 실행 중인 노드의 이름
  - 파드가 실행 중인 서비스 어카운트 이름
    - 서비스 어카운트는 12장에서 다룸

      → (대략적으로) 파드가 API 서버와 통실할 때 인증하는 계정
  - 각 컨테이너의 CPU와 메모리 요청
  - 각 컨테이너의 CPU와 메모리 제한
    - CPU/메모리 요청 및 제한은 14장에서 다룸

      → 컨테이너에 보장되는 CPU와 메모리의 양과 컨테이너가 얻을 수 있는 최대 양
  - 파드의 레이블
  - 파드의 어노테이션

레이블과 어노테이션은 볼륨으로만 노출

8.1 Downward API로 메타데이터 전달

- 컨테이너에 전달 가능한 정보들
  - 파드의 이름
  - 파드의 IP 주소
  - 파드가 속한 네임스페이스
  - 파드가 실행 중인 노드의 이름
  - 파드가 실행 중인 서비스 어카운트 이름
    - 서비스 어카운트는 12장에서 다룸

      → (대략적으로) 파드가 API 서버와 통실할 때 인증하는 계정
  - 각 컨테이너의 CPU와 메모리 요청
  - 각 컨테이너의 CPU와 메모리 제한
    - CPU/메모리 요청 및 제한은 14장에서 다룸

      → 컨테이너에 보장되는 CPU와 메모리의 양과 컨테이너가 얻을 수 있는 최대 양
  - 파드의 레이블
  - 파드의 어노테이션

레이블과 어노테이션은 볼륨으로만 노출

- 레이블과 어노테이션 업데이트
  - 파드 실행 중
    - 레이블과 어노테이션 수정 가능
      - 변경될 때, 쿠버네티스는 파일을 업데이트
      - 파드는 최신 데이터를 보게 된다

8.1 Downward API로 메타데이터 전달

- Downward API 사용 시기 이해
  - 애플리케이션을 쿠버네티스에 독립적으로 유지할 수 있게 한다.
    - 환경변수의 특정 데이터를 활용하는 경우, 특히 유용
  - 그러나,
    - Downward API로 사용 가능한 메타데이터는 상당히 제한적

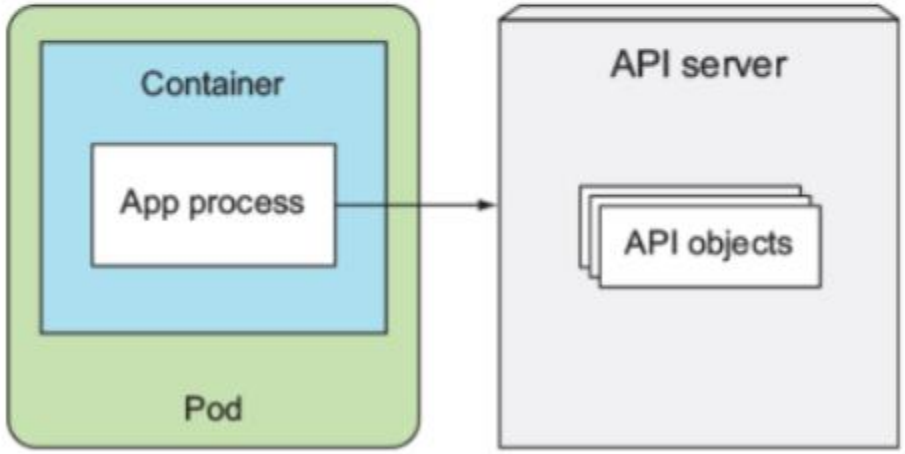
      → 더 많은 정보가 필요한 경우 쿠버네티스 API 서버에서 직접 가져오기

8.2 쿠버네티스 API 서버와 통신하기

- Downward API는
  - 단지 파드 자체의 메타데이터와 모든 파드의 데이터 중 일부만 노출
- 클러스터에 정의된 다른 파드나 리소스에 관한 더 많은 정보가 필요할 땐?

  → API 서버와 직접 통신

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



8.2 쿠버네티스 API 서버와 통신하기

8.2.1 쿠버네티스 REST API 살펴보기

- API 서버에 접속하기 위해 URL 보기

```
kubectl cluster-info
```

JavaScript

- 서버는 HTTPS를 사용하고, 인증이 필요하기 때문에
  - 직접 통신하는 것은 간단하지 않다.

    → kubectl proxy 명령을 실행해 프록시로 서버와 통신할 수 있다.

8.2 쿠버네티스 API 서버와 통신하기

- kubectl proxy로 API 서버에 액세스하기
  - kubectl proxy 명령은
    - 프록시 서버를 실행해
    - 로컬 컴퓨터에서 HTTP 연결을 수신하고,
    - 이 연결을 인증 관리하면서 API 서버로 전달하기 때문에
    - 요청할 때마다 인증 토근을 전달할 필요가 없다
  - 프록시 실행

```
kubectl proxy
```

JavaScript

→ kubectl은 필요한 모든 것(API 서버 URL, 인증 토큰 등)을 이미 알고 있음

→ 요청을 proxy로 보내면

→ 요청을 API 서버로 보낸 다음

→ proxy는 서버가 반환하는 모든 것을 반환한다.

8.2 쿠버네티스 API 서버와 통신하기

- kubectl proxy로 API 서버에 액세스하기
  - kubectl proxy 명령은
    - 프록시 서버를 실행해
    - 로컬 컴퓨터에서 HTTP 연결을 수신하고,
    - 이 연결을 인증 관리하면서 API 서버로 전달하기 때문에
    - 요청할 때마다 인증 토근을 전달할 필요가 없다
  - 프록시 실행

```
kubectl proxy
```

JavaScript

→ kubectl은 필요한 모든 것(API 서버 URL, 인증 토큰 등)을 이미 알고 있음

→ 요청을 proxy로 보내면

→ 요청을 API 서버로 보낸 다음

→ proxy는 서버가 반환하는 모든 것을 반환한다.

8.2 쿠버네티스 API 서버와 통신하기

- kubectl proxy로 쿠버네티스 API 살펴보기
  - 서버 반환 내용

```
curl http://localhost:8001
```

JavaScript

  - res

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1beta1",
    ...
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v2alpha1",
    ...
```

JavaScript

8.2 쿠버네티스 API 서버와 통신하기

8.2.2 파드 내에서 API 서버와 통신

- kubectl이 없는 파드 내에서 통신하는 방법
- 파드 내부에서 API 서버와 통신하려면 다음 세 가지를 처리해야 한다
  - API 서버의 위치를 찾아야 한다(1)
  - API 서버와 통신하고 있는지 확인해야 한다(2)
  - API 서버로 인증해야 한다(3)
    - 인증하지 않으면, 볼 수도 없고 아무것도 할 수 없다.

8.2 쿠버네티스 API 서버와 통신하기

- API 서버와의 통신을 시도하기 위해 파드 실행
  - 필요한 것
    - API 서버와 통신할 파드
      - 아무것도 하지 않는 파드에
      - kubectl exec으로 실행(접속)하여
      - curl을 사용해 API 서버에 엑세스
    - curl.yaml

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

  - 파드를 만든 후, exec

```
kubectl exec -it curl bash
```

→ 통신을 수행할 터미널 실행

8.2 쿠버네티스 API 서버와 통신하기

- API 서버 주소 찾기(1)
  - 쿠버네티스 API 서버의 IP와 포트를 찾아야 한다.
    - kubernetes라는 서비스
      - default 네임스페이스에 자동으로 노출되고
      - API 서버를 가리키도록 구성되어 있음
    - 확인

```
kubectl get svc
```

JavaScript

      - res

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

JavaScript

  - 서비스에 관한 환경변수를 통해 얻기

```
env | grep KURERNETES_SERVICE
```

JavaScript

    - res

```
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_HOST=10.0.0.1
KUBERNETES_SERVICE_PORT_HTTPS=443
```

- 서비스마다 있는 DNS 엔트리를 활용할 수도 있다.

```
curl https://kubernetes
```

JavaScript

  - res

```
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.
```

JavaScript

→ 간단한 해결은 -k 옵션 사용

8.2 쿠버네티스 API 서버와 통신하기

- 서버의 아이덴티티 검증(2)
  - 각 컨테이너의 /var/run/secrets/kubernetes.io/serviceaccount에 마운트되는 (자동으로 생성된) 시크릿 조회하기

```
ls /var/run/secrets/kubernetes.io/serviceaccount
```

JavaScript

    - res

```
ca.crt    namespace    token
```

JavaScript

      - 세 개 항목 중 ca.crt에 집중
  - API 서버와 통신 중인지 확인하기 위해
    - 서버의 인증서가 CA로 서명됐는지 확인

```
curl --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt https:/
```

JavaScript

      - res

```
Unauthorized
```

JavaScript

        → 인증처리 필요

- 쉽게 환경변수를 이용해 설정하는 방법

Copy to clipboard

```
export CURL_CA_BUNDLE=/var/run/secrets/kubernetes.io/serviceaccount/ca.cr

curl https://kubernetes
```

JavaScript

  - res

```
Unauthorized
```

JavaScript

8.2 쿠버네티스 API 서버와 통신하기

- API 서버로 인증(3)
  - 인증 토큰 필요

    → default-token 시크릿으로 제공

    → 시크릿 볼륨의 token 파일
  - 환경변수 설정

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
```

JavaScript

  - API 서버로 요청

```
curl -H "Authorization: Bearer $TOKEN" https://kubernetes
```

JavaScript

    - res

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1beta1",
    "/apis/authorization.k8s.io",
    ...
    "/ui/",
    "/version"
  ]
```

- 역할 기반 엑세스 제어(RBAC) 비활성화
  - RBAC가 활성화된 쿠버네티스 클러스터를 사용하는 경우
    - 서비스 어카운트가 API 서버에 액세스할 권한이 없을 수 있다.
    - 간단한 방법으로 우선 우회하도록 하자

```
kubectl create clusterrolebinding permissive-binding \
--clusterrole=cluster-admin \
--group=system:serviceaccounts
```

JavaScript

- 파드가 실행 중인 네임스페이스 얻기
  - 시크릿 볼륨에 네임스페이스라는 파일이 포함되있었다.
    - 파일에는 파드가 실행 중인 네임스페이스가 포함되있다.
  - 환경변수로 파일 내용을 로드 한 뒤, 파드 나열(조회)

```
NS=$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace)

curl -H "Authorization: Bearer $TOKEN" https://kubernetes/api/v1/namespac
```

JavaScript

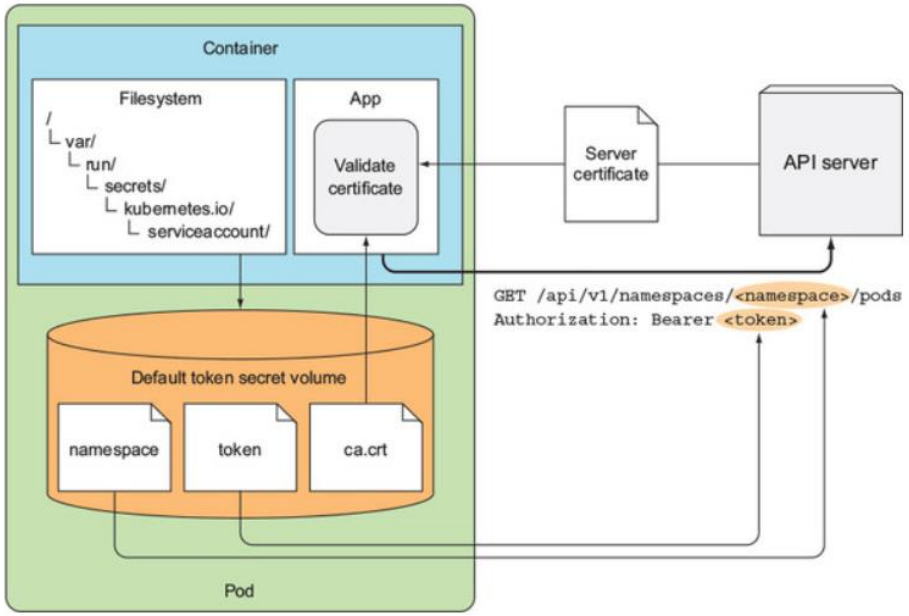
    - res

```
{
  "kind": "PodList",
  "apiVersion": "v1",
```

8.2 쿠버네티스 API 서버와 통신하기

- 파드가 쿠버네티스와 통신하는 방법 정리
  - 애플리케이션은
    - API 서버의 인증서가 인증 기관으로부터 서명됐는지 검증해야 하며
    - 인증 기관의 인증서는 ca.cart 파일에 있다
  - 애플리케이션은 token 파일의 내용을
    - Authorization HTTP 헤더에 Bearer 토큰으로 넣어
    - 전송해서 자신을 인증해야 한다.
  - namespace 파일은
    - 파드의 네임스페이스 안에 있는 API 오브젝트의 CRUD 작업을 수행할 때
    - 네임스페이스를 API 서버로 전달하는데 사용해야 한다.

Figure 8.5. Using the files from the default-token Secret to talk to the API server



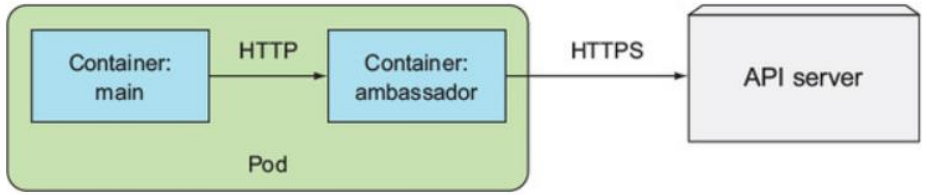
8.2 쿠버네티스 API 서버와 통신하기

8.2.3 앰배서더 컨테이너를 이용한 API 서버 통신 간소화

- 보안을 유지하면서 통신을 훨씬 간단하게 만들수 있는 방법
  - API 서버로 직접 요청하는 대신
  - 프록시로 요청을 보내 인증, 암호화 및 서버 검증을 처리하게 한다.
    - kubectl proxy처럼 파드 내에서도 동일한 방법 사용
- 앰배서더 컨테이너 패턴 소개
  - 메인 컨테이너 옆의 앰배서더 컨테이너에서
    - kubectl proxy를 실행하고,
    - 이를 통해 API 서버와 통신할 수 있다.
  - API 서버와 직접 통신하는 대신,
    - 메인 컨테이너의 애플리케이션은 HTTP로 앰배서더에 연결하고
    - 앰배서더 프록시가 API 서버에 대한 HTTPS 연결을 처리하도록해
    - 보안을 투명하게 관리 할 수 있다.

    → 시크릿 볼륨에 있는 default-token 파일을 사용해 수행

Figure 8.6. Using an ambassador to connect to the API server



8.2 쿠버네티스 API 서버와 통신하기

- 추가적인 앰배서더 컨테이너를 사용한 curl 파드 실행
  - 파드에 메인 컨테이너와 더불어 앰배서더 컨테이너를 실행
  - 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
```

JavaScript

→ 지금 앰배서더로 사용할 컨테이너 이미지는 도커 허브에 배포된 것

→ 직접 빌드하기 위해서는 Dockerfile을 살펴보자

- Dockerfile

Copy to clipboard

```
FROM alpine
RUN apk update && \
    apk add curl && \
    curl -L -O https://dl.k8s.io/v1.8.0/kubernetes-client-linux-amd64.
    tar zvxf kubernetes-client-linux-amd64.tar.gz kubernetes/client/bi
    mv kubernetes/client/bin/kubectl / && \
    rm -rf kubernetes && \
    rm -f kubernetes-client-linux-amd64.tar.gz
ADD kubectl-proxy.sh /kubectl-proxy.sh
ENTRYPOINT /kubectl-proxy.sh
```

JavaScript

8.2 쿠버네티스 API 서버와 통신하기

- 파드를 실행한 다음 main 컨테이너로 접속

```
kubectl exec -it curl-with-ambassador -c main bash
```

JavaScript

→ 파드에 컨테이너가 복수개 있기 때문에,

→-c 옵션으로 접속하려는 main 컨테이너를 지정해줘야 한다.

- 앰배서더를 통한 API 서버와의 통신(main 컨테이너에서)

```
curl localhost:8001
```

JavaScript

- res

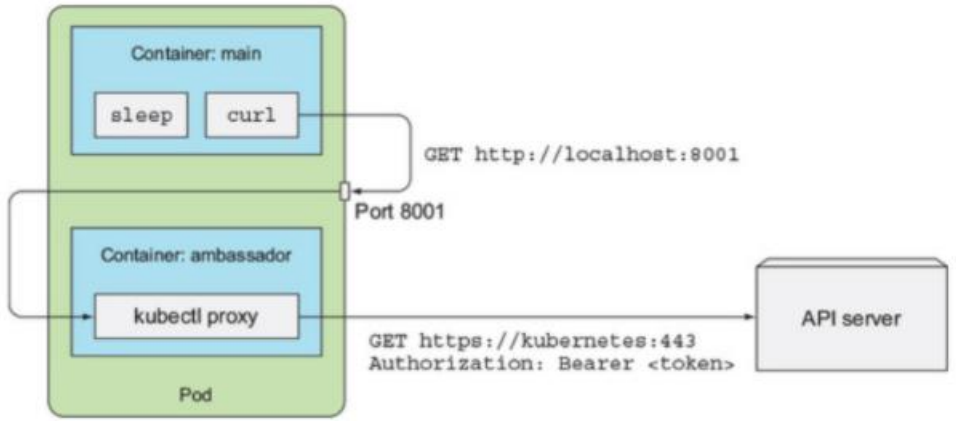
```
{
  "paths": [
    "/api",
    ...
  ]
}
```

JavaScript

→ 앰배서더의 kubectl proxy가 API 서버 URL, 인증 등을 처리하기 때문에

따로 할 것이 없다.

Figure 8.7. Offloading encryption, authentication, and server verification to `kubectl proxy` in an ambassador container



- main 컨테이너는 일반 http 요청을 앰배서더 컨테이너의 프록시로 전송
- 프록시는 HTTPS 요청을 API 서버로 전송,
  - 토큰을 전송해 클라이언트 인증을 처리하고
  - 서버의 인증서를 검증해 서버의 신원을 확인한다
- 앰배서더 컨테이너의 실행으로 추가 리소스 소비한다는 것이 단점

8.2 쿠버네티스 API 서버와 통신하기

8.2.4 클라이언트 라이브러리를 사용해 API 서버와 통신

- 단순한 API 요청 이상을 수행하려면,
  - 쿠버네티스 API 클라이언트 라이브러리 중 하나를 사용하는 것이 좋다.

- 클라이언트 라이브러리 사용
  - API Machinery SIG(Special Interest Group)에서 지원하는 Kubernetes API 라이브러리는 두 가지
    - Golang 클라이언트

      kubernetes/client-go

      Go clients for talking to a kubernetes cluster. We recommend using the v0.x.y tags for Kubernetes releases

      https://github.com/kubernetes/client-go

    - python

      kubernetes-client/python

      Python client for the kubernetes API. From source: git clone --recursive https://github.com/kubernetes-client/python.git

      https://github.com/kubernetes-client/python

- 그 외에도 다양한 언어에 사용자 제공 클라이언트 라이브러리들이 있다.

  Java client by Fabric8—https://github.com/fabric8io/kubernetes-client
  Java client by Amdatu—https://bitbucket.org/amdatulabs/amdatu-kubernetes
  Node.js client by tenxcloud—https://github.com/tenxcloud/node-kubernetes-client
  Node.js client by GoDaddy—https://github.com/godaddy/kubernetes-client
  PHP—https://github.com/devstub/kubernetes-api-php-client
  Another PHP client—https://github.com/maclof/kubernetes-client
  Ruby—https://github.com/Ch00k/kubr
  Another Ruby client—https://github.com/abonas/kubeclient
  Clojure—https://github.com/yanatan16/clj-kubernetes-api
  Scala—https://github.com/doriordan/skuber
  Perl—https://metacpan.org/pod/Net::Kubernetes

  → 일반적으로 HTTPS를 지원하고 인증을 관리하므로 앰배서더 컨테이너 필요X

- 예시
  - Fabric8 Java Client를 사용한 쿠버네티스와의 상호작용 예시
  - 스웨거와 OpenAPI를 사용해 자신의 라이브러리 구축
  - 스웨거 UI로 API 살펴보기