쿠버네티스 교과서 2장

Pod : 컨테이너 하나 이상을 관리하는 단위

파드는 다른 리소스가 관리한다

계층적?으로 고수준의 리소스가 컨테이너의 세부사항을 추상화시킴

Deployment : 파드의 관리를 담당하는 리소스

- 파드는 노드 중 하나에서 실행된다
- 파드는 어플리케이션의 구성요소를 가상화하는 컨테이너를 가상화하는 리소스이다 => 가상화의 가상화
- 파드는 쿠버네티스에서 관리되는 가상 IP 주소를 가짐
- 이 주소로 같은 쿠버네티스 그룹 내에서 통신 가능
- 파드에 포함되는 모든 컨테이너는 이 ip주소를 공유한다

쿠버네티스가 컨테이너를 직접 실행하지 않는다.

컨테이너 런타임에 생성할 책임을 맡김

1. 도커
2. 다른 것들

파드로 감싼 이유??
도커 외에도 다른 시스템이 가상화한 컨테이너를 파드로 묶어서 관리하기 위해

명령어들

```
// 도커허브에서 제공하는 이미지에서 podname으로 pod 생성
kubectl run {pod_name} --image=~~~

// 파드가 준비될 때까지 기다림
kubectl wait --for=condition=Ready pod {pod_name}
```

```
// pods 목록 확인
kubectl get pods


// 파드 확인 추가 옵션 1
// 네트워크 상세 정보 중 특정한 항목을 따로 지정해서 출력
kubectl get pod hello-kiamol --output custom-
columns=NAME:metadata.name,NODE_IP:status.hostIP,POD_IP:statu
s.podIP

// JSONPath로 복잡한 출력을 구성
kubectl get pod {pod_name} -o
jsonpath='{.status.containerStatues[0].containerID}'
```

노드

파드는 생성될 때 하나의 노드에 배정된다
노드는 파드를 관리하고 파드안의 컨테이너를 실행한다

이 과정은 Container Runtime Interface라는 공통 API를 이용해서 컨테이너 런타임과 연동되는 형태로 진행

CRI 기능?

- 컨테이너 생성
- 컨테이너 삭제
- 컨테이너 정보확인 등

```
// 파드에 포함된 컨테이너 찾기
docker container ls -q --filter
label=io.kubernetes.container.name={pod_name}

// 해당 컨테이너 삭제하기
docker container rm -f $(docker container ls -q --filter
label=io.kubernetes.container.name={pod_name})
```

쿠버네티스에서 외부 트래픽에 대한 처리를 허용해야 내부 파드 컨테이너에 접근할 수 있음.

외부에서 8080포트로 들어오면 우리 파드의 80번 포트로 전달하자
`kubectl port-forward pod/hello-kiamol 8080:80`

=> 이 어플리케이션의 정식 설정이 아니라 임시로 설정한거임
그래서 종료하면 포트포워딩이 닫힘

결론 : 우리가 파드를 직접 실행하는 것은 아니다. 이를 관리할 객체로 파드에 접근한다

Deployment

파드 위로 얹히는 또 다른 추상화?
컨트롤러 객체 중 하나

노드가 고장나서 파드가 유실되면 deployment가 대체 파드를 다른 노드에 실행한다

```
// deployment 객체 생성
kubectl create deployment {pod_name} --image=???
```

deployment가 생성한 pod는 이름 뒤에 무작위 문자열이 추가됨

delpoyment 만듦과 동시에 파드가 만들어지는데,
왜??

=> deployment 생성할 때 필요한 파드가 어떤 것인지 정의했음

=> 우리가 원하는 상태를 정의했음
디플로이먼트가 만들어 지고 원하는 상태와 현재상태를 체크해서
차이가 있으니 알아서 만들어 준 것.

모든 쿠버네티스 리소스는 key-value 레이블을 가진다.
이 레이블을 통해 우리가 원하는 데이터를 담을 수 있다

배포 버전 같은 것을 담을 수 있음

리소스는 생명주기를 가진다.
삭제 수정 생성등을 통해 레이블을 부여받음

컨트롤러 객체는 레이블 셀렉터를 통해 자신이 관리하는 리소스인지 확인

아무튼 레이블을 통해 컨트롤러 객체는 파드를 관리한다

=> 그래서 레이블이 임의로 수정되어서 바뀌면 컨트롤러 객체가 인지하지 못함

=> 파드유실로 인지해서 원하는 상태와 벗어나므로 파드를 하나 만듦

=> 이를 이용해서 디버깅에 활용하기도 함.

1. 레이블을 수정해서 컨트롤러 관리 객체에서 분리시키고
2. 별도로 접속해서 뭐가 문제인지 확인하면 됨

---

yaml 파일 정의

```
# 쿠버네티스 API 버전과 정의하려는 리소스 유형을 명시
apiVersion : v1
kind : Pod

# 리소스 메타데이터 정의
# 이름 : 필수요소
# 레이블 : 선택
metadata :
        name : hello-kiamol-3

# spec은 리소스의 실제 정의 내용
# 현재는 파드니깐 컨테이너를 정의
# 컨테이너의 이름과 이미지 명시
spec :
        container :
                - name : web
                image : kiamol/ch02-hello-kiamol
```

메니페스트 파일을 클러스터에 적용하면 이 파일이 API에 전달되어 정의된 사항을 클러스터에 반영한다

주의 apply 명령할 때 해당 파일이 있는 디렉토리 위치에서 명령해야 함

`kubectl apply -f pod.yaml`
-f옵션은 파일 경로를 지정하여 해당 파일에 정의된 리소스를 쿠버네티스 클러스터에 적용시킴

메니페스트 파일은 공유공간에서 가져와도 된다,
로컬공간에서 안해도 됨.

- 메니페스트 파일 내용이 같으면 변경되지 않았다고 뜸

```
apiVersion : v1
kind : Deployment
```

```
metadata:
        name : hello-kiamol-4

# 디플로이가 자신의 관리대상을 결정하는 레이블 셀렉터가 정의
# app 레이블을 사용, 이 때 레이블은 key-value 쌍
spec :
        selector :
                matchLabels :
                        app : hello-kiamol-4


# 파드의 정의

spec :
        containers :
                - name : web
                image : img
```

yaml 파일에는 다양한 정의 가능

- 복잡해지면 스케일링을 위해 복제본을 얼마나 만들지
- cpu와 메모리의 상한선
- 어플리케이션의 상태 체크 방법
- 어플리케이션에 사용할 설정값은 어디서 읽고 어디다 저장할지

웬만한 설정을 yaml파일에 저장한다고 보면 됨

원래는 파드안의 컨테이너에 직접 접근할 수는 없지만 kubectl을 사용하면 접근할 방법이 있다.

생성한 파드와 터미널 세션 연결

```
// 파드 내부와 연결할 대화형 셸 실행
// -it 옵션이 컨테이너에서 실행한 셸과 현재 터미널 세션을 연결하라는 말
kubectl exec -it {pod_Name} sh

// 파드안에서 주소 확인
hostname -i

=> 컨테이너 주소 == 파드의 주소

// 웹 어플리케이션의 동작 확인
wget -0 - http://localhost | head -n 4

// 세션 종료
exit
```

이렇게 파드와 터미널을 연결하면 파드 속 상황을 파악하기 좋음

- 가상 네트워크로 API 엔드포인트에 ping을 날릴 수도 있음
- 로그 열람하는 kubectl명령도 있긴 함

```
1. 쿠버네티스에서 로그 확인하기
// 파드 이름을 직접 알고 있을 때 사용하는 방법
kubectl logs --tail=2 {pod_name}


2. 도커를 통해 로그 확인하기

docker container logs --tail=2 $(docker container ls -q --filter label=io.kubernetes.container.name={pod_name})


// 파드 이름을 몰라도 디플로이에서 어플리케이션 호출
```

```
kubectl exec deploy/hello-kiamol-4 -- sh -c 'wget -o -
http://localhost > /dev/null'

// 해당 파드의 로그 열람
// 레이블 셀렉터를 활용해서 파드를 직접 명시하지 않아도 출력할 수
있음
kubectl logs --tail=1 -l app=hello-kiamol-4
```

쿠버네티스는 컨테이너 런타임을 경유해서 로그를 불러온다
컨테이너를 만든것은 도커 컨테이너이기 때문에!

- exec 명령은 다양한 리소스를 대상으로 사용할 수 있다.
- 파드 속 컨테이너 안에서 명령을 실행하고 그 결과를 출력한다

파드 속 파일 시스템에 접근하는 방법

```
// 로컬에 디렉토리 생성
mkdir -p /tmp/kiamol/ch02

// 파드의 파일을 로컬로 복사하기
kubectl cp hello-kiamol:usr/share/nginx/html/index.html
/tmp/kiamol/ch02/index.html

// 로컬 컴퓨터에서 파일 확인
cat /tml/kiamol/ch02/index.html
```

- 쿠버네티스 위치가 원격이든 아니든 복사할 수 있음. 양방향으로

파드 삭제 방법

`kubectl delete pods --all`

삭제를 해도 파드가 남아있음
왜?
-> deployment로 상태를 정의했기 때문에 파드가 사라져도 원하는 상태로 deployment가 파드를 다시 추가했음

그래서 deployment도 삭제 해줘야 함

컨트롤러 객체가 관리하는 리소스를 삭제하려면 컨트롤러 객체를 삭제시켜야 한다.

Deployment 삭제 방법

```
kubectl delete deployment --all
```

추가 정리

클러스터

- 쿠버네티스를 배포하면 클러스터를 얻는다
- 쿠버네티스 실행 중? == 하나 이상의 클러스터 실행 중
- 클러스터는 노드의 집합
- 모든 클러스터는 하나이상의 마스터 노드와 워커 노드를 가진다

클러스터의 구성 요소

Control Plane : 제어 영역

- Master Node
- 클러스터 상태를 관리하고 명령어를 처리하는 역할
- etcd, cotnroller-manager, scheduler, kube api server 등 컴포넌트를 가짐
- *kubectl은 kube api server와 통신하는 용도로 사용됨*
-

Node : 실행 영역

- Worker Node
- 컨테이너를 실행하는 역할

실습문제

80번 포트를 주시하는 웹사이트 실행하는 컨테이너를 담는 디플로이먼트를 정의하라

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: my-test

spec:

  selector:

    matchLabels:

      app: my-test

  template:

    metadata:

      labels:
```

```
        app: my-test

    spec:

      containers:

        - name: web

          image: kiamol/ch02-hello-kiamol

          ports:

            - containerPort: 80
```

이렇게 하고

`kubectl apply my-test-1.yaml` 하면 되는거 아닌감?

그렇게 하고 localhost에 접속해보니 안됨

```
1. kubectl apply -f my-test-1.yaml

2. 포트포워딩
kubectl port-forward deploy/my-test-1 8080:80

라벨 앞에 deploy를 붙여줌으로써 deploy 리소스라는 것을 명시함
pod로 할거라면

kubectl port-forward pod/my-test-1 8080:80

라고 하면 됨
```

근데 지금은 터미널 세션을 닫으면 포트포워딩이 종료돼서 외부에서의 접근이 항상 유지되지 않음

이걸 유지하려면 Service라는 객체를 사용하면 됨

기존 yaml에서 추가되는 내용

```
---

apiVersion: v1

kind: Service

metadata:

  name: my-test-1

spec:

  selector:

    app: my-test-1

  ports:

    - protocol: TCP

      port: 80

      targetPort: 80

  type: LoadBalancer # 또는 NodePort, ClusterIP 등 필요에 따라 변경 가능
```