4.1 파드를 안정적으로 유지하기

생성일	@2021년 3월 25일 오후 11:21
태그	

> 쿠버네티스에 컨테이너 목록을 제공하면 해당 컨테이너를 클러스터 어딘가에서 계속 실행되도록 할 수 있음

만약 이 컨테이너 중 하나가 죽거나 파드 안에 있는 모든 컨테이너가 죽는다면? → kubelet이 해결해줌

[kubelet : 파드가 노드에 스케줄링되는 즉시, 해당 노드의 kubelet은 파드의 컨테이너를 실행 **]**

- 컨테이너의 주 프로세스에 크래시가 발생하면 kubelet이 파드를 다시 시작함
- 쿠버네티스에서 애플리케이션을 실행하는 것만으로도 자동으로 재시작할 수 있음

앱이 프로세스의 크래시 없이 작동이 중단되는 경우
→ 외부에서 앱이 제대로 동작하지 않는다는 신호를 쿠버네티스에 보내서, 쿠버네티스가 앱을 다시 시작하게 만드는 방법을 알아보자..

1. 라이브니스 프로브 소개

Liveness probe : 컨테이너가 살아 있는지 확인 가능

- 설정방법 : 파드의 specification에 각 컨테이너의 라이브니스 프로브를 지정
- 쿠버네티스가 주기적으로 프로브를 실행하고, 이것이 실패할 경우 컨테이너를 다시 시작한다
- ↔ Rediness Probe (레디니스 프로브) 와는 쓰임새가 다름

프로브 실행 매커니즘

1. HTTP GET 프로브는 지정한 IP 주소, 포트, 경로에 HTTP GET 요청을 수행

- 프로브가 수신한 HTTP 응답 코드가 정상(2,300대)일 경우 → 프로브 성공으로 간주
- 비정상 응답코드거나 응답이 없을 경우 → 프로브 실패로 간주 → 컨테이너 재시작

2. TCP 소켓 프로브 : 컨테이너의 지정된 포트에 TCP 연결을 시도
   - 연결 성공 → 프로브 성공
   - 그렇지 않은 경우 → 프로브 실패 → 컨테이너 재시작
3. Exec 프로브 : 컨테이너 내의 임의의 명령 실행
   - 명령 종료 상태코드 = 0 인 경우 → 프로브 성공
   - 다른 모든 코드 → 프로브 실패 → 재시작

2. HTTP 기반 라이브니스 프로브 생성

기존 node.js 코드에 인위적으로 실패하는 로직을 넣는다

: 5번째 이후의 요청부터 500 internal Server Error HTTP 상태코드를 반환하도록 함

파드에 라이브니스 프로브를 `vi`로 추가 - `kubia-liveness-probe.yaml`

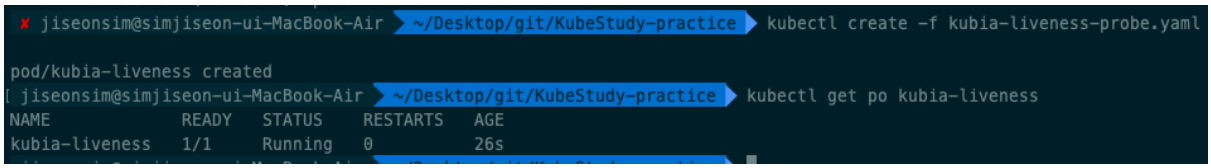
```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-liveness
spec:
  containers:
  - image: luksa/kubia-unhealthy # 약간 문제가 있는 애플리케이션을 포함한 이미지
    name: kubia
    livenessProbe: # HTTP GET을 수행하는 라이브니스 프로브
      httpGet:
        path: / # HTTP 요청 경로
        port: 8080 #프로브가 연결해야하는 네트워크 포트
```

- 해당 요청은 컨테이너가 실행되는 즉시 시작된다
- 다섯번의 요청 후 앱은 HTTP 상태코드 500을 반환하기 시작 → 프로브 실패로 간주 : 컨테이너 재시작

3. 동작중인 라이브니스 프로브 확인

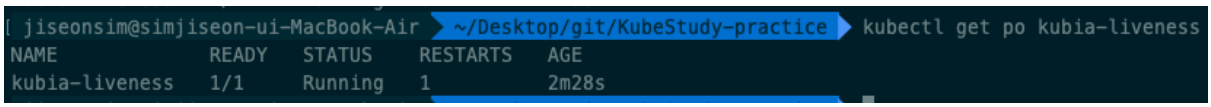
라이브니스 프로브의 기능을 보기 위해 파드 생성

```
kubectl create -f kubia-liveness-probe.yaml
kubectl get po kubia-liveness
```



(아직 리스타트를 안했나보다. 좀만 더 기다려보자)

우와! RESTARTS에 1이 생겼다. → 즉, 파드의 컨테이너가 한 번 다시 시작했다는 것이다



자, 그럼 컨테이너 재시작의 이유를 araboja.

1. 크래시된 컨테이너의 애플리케이션 로그 얻기

   ```
   kubectl logs kubia-liveness --previous
   ```

   `--previous` 로 현재 컨테이너가 아닌, 이전 컨테이너의 로그가 종료된 이유를 파악할 수 있다

```
[ jiseonsim@simjiseon-ui-MacBook-Air ~/Desktop/git/KubeStudy-practice kubectl logs kubia-liveness --previous
Kubia server starting...
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
Received request from ::ffff:10.76.2.1
```

2. 컨테이너가 다시 시작된 후의 파드 디스크립션으로 확인

```
kubectl describe po kubia-liveness
```

```
Containers:
  kubia:
    Container ID:   docker://9f43f395b11facf5374f672b08c6cdc33e4c7f4b0f356a593c627269b4e968a7
    Image:          luksa/kubia-unhealthy
    Image ID:       docker-pullable://luksa/kubia-unhealthy@sha256:5c746a42612be61209417d913030d97555cff0b8225092908c57634ad7c235f7
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 26 Mar 2021 01:52:55 +0900
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      Fri, 26 Mar 2021 01:51:05 +0900
      Finished:     Fri, 26 Mar 2021 01:52:55 +0900
    Ready:          True
    Restart Count:  5
    Liveness:       http-get http://:8080/ delay=0s timeout=1s period=10s #success=1 #failure=3
    Environment:    <none>
```

`State : Running` ← 컨테이너가 현재 실행중임

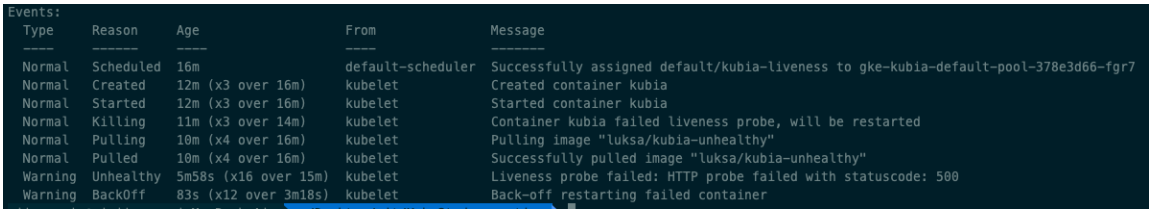
`Last State: Terminated` ← 이전 스테이트가 중지되었고,

`Reason: Error` ← 에러로 인해

`Exit Code: 137` ← 137코드를 반환하고 terminate되었다는 것을 알 수 있다

137은 128+x 두 숫자를 합한 값
x는 프로세스에 전송된 시그널 번호로 SIGKILL이 9로 강종된것을 알 수 있다.

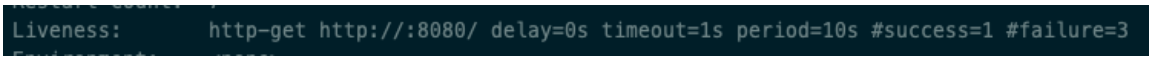
`Restart Count : 5` ← 컨테이너가 5번 재시작되었다..



이벤트 로그를 보면, Liveness probe가 http 500리턴받아서 실패한것으로 간주한다. 그래서 Kubelet이 컨테이너를 kill하고, restart한것을 볼 수 있다.

4. 라이브니스 프로브의 추가 속성 설정

위의 `kubectl describe`는 라이브니스 프로브에 관한 추가적인 정보도 표시해준다



`delays=0s` : 컨테이너가 시작된 후 지연 없이 바로 프로브가 시작된다는 것을 나타냄

`timeout=1s` : 제한시간이 1초로 설정되어있음 → 컨테이너가 1초안에 응답해야한다 (안그럼 프로브가 실패로 간주)

`period=10s` : 컨테이너는 10초마다(기간) 프로브를 수행

`#failure=3` : 프로브가 3번 연속 실패하면 컨테이너가 다시 시작된다

라이브니스 프로브에 추가적인 매개변수 설정방법: initialDelaySeconds 속성 추가

`kubia-liveness-probe-initial-delay.yaml`을 vi로 생성해주자

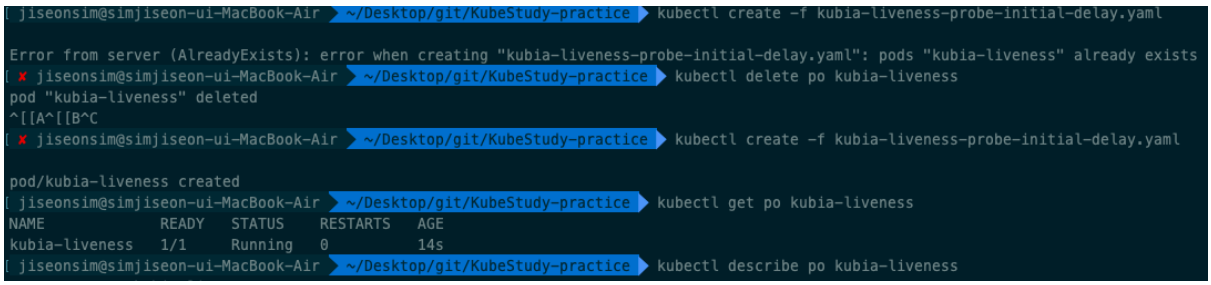
```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-liveness
spec:
  containers:
  - image: luksa/kubia-unhealthy
    name: kubia
    livenessProbe: #여기에 속성 추가
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 15 # 첫번째 프로브 실행까지 15초 대기
```

초기지연을 15초로 설정했는데, 이를 설정하지 않으면 프로브는 컨테이너가 시작되자마자 프로브를 시작

- 이 경우 대부분 앱이 요청받을 준비가 돼 있지 않기 때문에 프로브가 실패함
- 실패 횟수가 실패 임곗값을 초과하면 요청을 올바르게 응답하기 전에 컨테이너가 재시작됨

→ 꽤 빈번히 일어나는 경우이므로 파드 시작시 이 문제를 피하고 싶다면
`initialDelaySeconds`를 설정하자

위를 테스트 하기 위해서 `kubia-liveness` 파드를 삭제후 재생성해주자..



`kubectl describe po kubia-liveness`를 해보면

설정이 잘 되어있는 것을 볼 수 있당..

5. 효과적인 라이브니스 프로브 생성

운영 환경에서 실행 중인 파드는 반드시 라이브니스 프로브를 정의해야함!

- 정의 하지 않으면 쿠버네티스는 앱이 죽었는지 살아있는지 알 방법이 없음

라이브니스 프로브가 확인해야할 사항

위에서 만든 간단한 라이브니스 프로브는 단순히 서버가 응답하는지만 검사

추가적으로, ***특정 URL 경로***에 요청하도록 프로브를 구성해서 앱 내 실행중인 주요 모듈의 응답을 확인 가능

라이브니스 프로브는 앱 내부만 체크. (외부 요인의 영향을 받지 않도록 해야함)
ex) 프론트엔드 웹서버의 라이브니스 프로브 : 프론트 웹서버만 체크
백엔드 데이터 베이스에 연결할 수 없을 때 실패를 반환해서는 안된다..

프로브를 가볍게 유지하기

- 라이브니스 프로브는 너무 많은 연산 리소스를 사용해서 안됨
- 1초 내에 완료되어야 함
- 컨테이너의 속도를 상당히 느려지게 만들기 때문
    - 자바 앱을 실행하는 컨테이너의 경우 exec 프로브로 전체 jvm 재가동 보다는 http get 프로브 이용하자

프로브에 재시도 루프 구현하지 마라..

프로브 실패 임곗값이 1이어도 쿠버네티스는 실패를 한번 했다고 간주하기 전에 프로브 여러번 재시도함

재시도 루프는 헛수고 라는 뜻

애플리케이션이 다른 노드에서 재시작되게 하려면..?

라이브니스 프로브가 실패한 경우, 노드(파드를 호스팅하는)의 Kubelet에서 수행

> 노드 자체에 크래시가 발생하는 경우, 대체 파드를 컨트롤 플레인이 생성해야함
> Kubelet은 노드에서 실행되기 때문에 노드 자체가 고장나면 암것도 못함

앱이 다른 정상 노드에서 시작되게 하려면 레플리케이션 컨트롤러 등에서 파드 관리해야함..