


[k8s in action] 17장 애플리케이션 개발 모범 사례

Created By

 용호 최

Last Edited

4월 14, 2019 11:04 오후

Tags

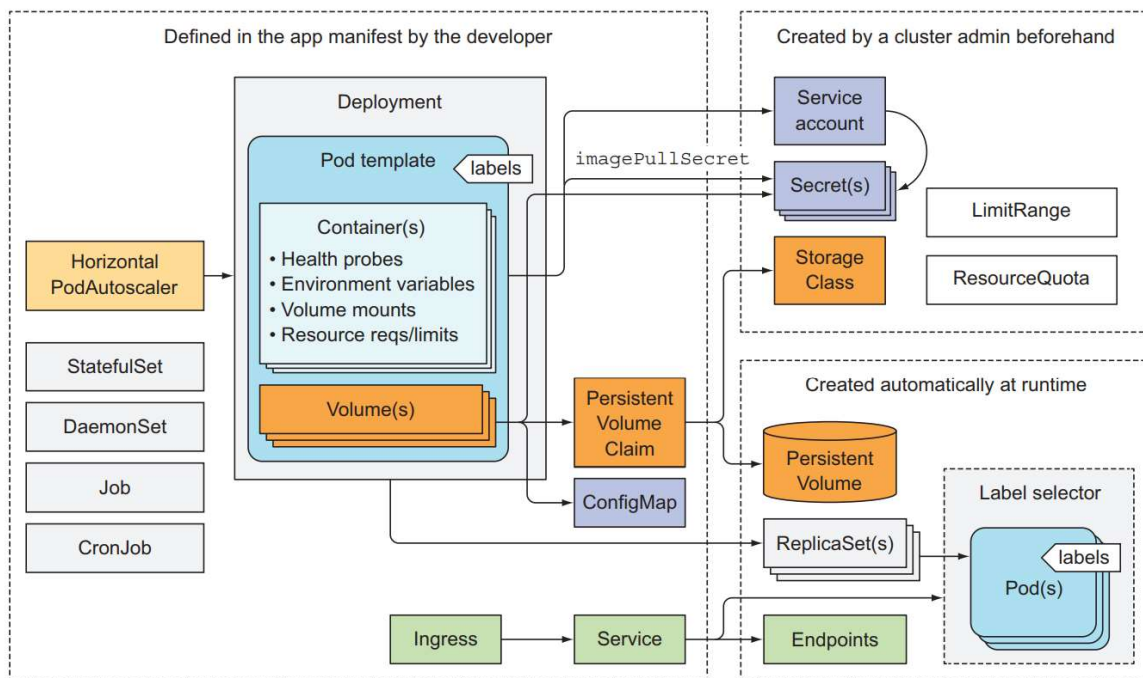
kubernetes 쿠버네티스 kubernetes in action

Add a Property



Add a comment...

17.1 모든 것을 함께 가져오기



- 일반적인 애플리케이션 매니페스트에는 하나 이상의 디플로이먼트 또는 스테이트풀셋 객체가 포함되어 있다.
 - 애플리케이션 매니페스트는? 애플리케이션을 구성하는 정보를 의미하고 여기서는 쿠버네티스에서 애플리케이션에 해당하는 Pod를 구성하기 위해 디플로이먼트를 제작하고, 경우에 따라 State가 필요한 경우에는 스테이트풀셋을 사용하여 애플리케이션을 구성한다는 의미로 해석

- 디플로이먼트와 스테이트풀셋 객체를 구성할 때 포드 템플릿이 포함되는데 여기에 하나 이상의 컨테이너와 livenessProbe나 readinessProbe 등을 구성할 수 있다.
 - 포드 템플릿은? 레플리케이션 컨트롤러, 잡, 데몬셋과 같은 다른 객체를 포함하는 포드의 명세서라고 생각하면 된다. 컨트롤러가 포드 템플릿을 사용하여 실제 포드를 생성한다.
 - 참고로 포드 템플릿을 통해 포드가 생성되고 나면 포드 템플릿이 변경되어도 이미 생성된 포드에는 영향이 가지 않는다.
 - Probe란? kubelet에 의해 주기적으로 수행되는 진단(diagnostic)이며, kubelet이 컨테이너에 의해 구현된 핸들러를 호출하여 진단을 수행한다. 진단 방법은 다음과 같다.
 - ExecAction : 컨테이너 내에서 지정된 명령어를 실행
 - TCPSocketAction : 컨테이너의 IP 주소에 포트가 활성화 되어 있는지 검사
 - HTTPGetAction : 컨테이너의 IP 주소에 HTTP Get 요청을 수행하여 200 상태 코드를 반환하는지 검사
 - kubelet이 컨테이너가 정상 동작 중임을 판단하기 위해 실행하는 두가지 진단 방법이 바로 livenessProbe와 readinessProbe 이다.
 - livenessProbe : 컨테이너가 동작 중인지 여부
 - 컨테이너 동작 체크에 실패하면 kubelet이 컨테이너를 죽이고 해당 컨테이너는 재시작 정책(restartPolicy)의 대상이 된다.

```
livenessProbe: exec: command: - cat - /tmp/healthy
initialDelaySeconds: 5 periodSeconds: 5
```

JavaScript

- 위 예제에서는 컨테이너의 정상 여부를 판단하기 위해 cat 명령을 사용하여 /tmp/healthy 파일을 읽는다. (exit code가 0이면 정상)
- 컨테이너가 구동되고 준비되는 시간이 필요할 수 있기 때문에 initialDelaySeconds 설정에 의해 5초 이후 부터 Probe를 실행한다.
- periodSeconds 설정에 의해 5초 간격으로 Probe를 실행한다.
- readinessProbe : 컨테이너가 요청을 처리할 준비가 되었는지 여부
 - 준비가 되지 않은 경우 엔드포인트 컨트롤러는 포드에 연관된 모든 서비스들의 엔드포인트에서 포드의 IP주소를 제거한다.
 - 예를들어 애플리케이션 구동 시 대용량 파일을 로드해야하는 경우

로드가 되기 전까지는 애플리케이션으로 트래픽이 전달되면 안될 수도 있다. 이런 경우에는 readinessProbe를 사용하여 준비가 되기 전까지는 엔드포인트에서 제외시켰다가 진단에 의해 준비가 되었다고 판단되면 그 때 엔드포인트에 연결하는 것과 같은 처리에 사용될 수 있다.

- readinessProbe 설정방법은 livenessProbe와 동일하다. (위 예제 참고)
- 애플리케이션을 사용자가 이용할 수 있으려면 포드는 하나 이상의 서비스를 통해 외부로 노출하여 클러스터 밖에서도 접속할 수 있도록 해야한다. 외부로 노출하는 방법은 다음과 같다.
 - LoadBalancer
 - NodePort
 - Ingress Resource
- 포드 템플릿은 일반적으로 Private Image Registry에서 컨테이너 이미지를 가져오는데 필요한 시크릿과 포드 내부에서 실행하는 프로세스에서 직접 사용할 시크릿을 참조한다.

- Private Image Registry 접근을 위한 시크릿 등록 방법

1. docker login 명령으로 로그인을 하면 ~/.docker/config.json 파일이 생성된다.
2. 다음 명령으로 regcred라는 시크릿을 생성한다.

```
kubectl create secret generic regcred \ --from-  
file=.dockerconfigjson=$HOME/.docker/config.json \ --  
type=kubernetes.io/dockerconfigjson
```

JavaScript

3. Pod 생성 시 ImagePullSecrets를 지정한다.

```
apiVersion: v1 kind: Pod metadata: name: private-reg spec:  
containers: - name: private-reg-container image: <your-private-  
image> imagePullSecrets: - name: regcred
```

JavaScript

- 시크릿은 애플리케이션 매니페스트의 일부가 아니기 때문에 ServiceAccount에 할당된다.

- 애플리케이션에서 사용할 환경 변수를 지정하거나 설정을 위한 ConfigMap을 포함할 수 있다.

- configMap을 포드의 볼륨에 마운트할 수도 있다. 아래 configMap을 마운트하면 마운트한 경로에 configMap data의 key가 파일명이 되고 value가 파일의 내용으로 입력된다.

```
apiVersion: v1 kind: ConfigMap metadata: name: special-config
namespace: default data: SPECIAL_LEVEL: very SPECIAL_TYPE: charm
```

JavaScript

- 위 예제에서는 마운트 위치에 SPECIAL_LEVEL 파일이 생성되고 안에 내용은 very이다.
- 특정 포드는 emptyDir이나 gitRepo 등 다양한 볼륨을 사용할 수 있고, 영구 저장을 필요로하는 경우에는 persistentVolumeClaim 볼륨을 사용한다.
 - emptyDir : 포드가 생성될 때 빈 볼륨으로 생성되었다가 포드가 제거되면 영구 삭제된다.
 - 사용 시 장점은 포드 내 컨테이너들끼리 데이터를 공유할 수 있고, 포드 내 컨테이너가 크래쉬 나더라도 이 emptyDir은 포드의 생명주기를 따르기 때문에 emptyDir이 삭제되지 않아서 컨테이너가 복구되면 데이터를 그대로 유지할 수 있다.
 - 기본적으로 노드의 디스크를 사용하지만 설정에 따라(emptyDir.medium: Memory) 메모리에 저장할 수도 있다.
 - gitRepo : 포드가 생성될 때 emptyDir처럼 빈 디렉토리가 마운트되고, 볼륨 설정 시 지정된 git repository에서 clone 받는다.
 - 현재는 지원이 중단되었고, emptyDir을 사용하며 initContainer를 통해 해당 기능을 수행할 것을 권장
 - PersistentVolume : 애플리케이션이 인프라 세부 사항에 대해 알 필요 없이 쿠버네티스 클러스터에 등록된 스토리지를 사용할 수 있도록 한다. 이를 위해 PersistentVolume과 PersistentVolumeClaim이라는 리소스를 사용한다.
 - 관리자가 NFS나 AWS의 EBS와 같은 물리적인 스토리지를 생성한 후에 쿠버네티스 API를 통해 클러스터에 PV를 생성한다. 그리고 나서 쿠버네티스 클러스터를 사용하는 사용자가 PVC를 통해 PV를 요청하면 쿠버네티스 클러스터는 적절한 스토리지 크기의 PV를 찾은 후 PVC를 PV에 바인딩한다.

- 애플리케이션에서 잡 또는 CronJob을 사용해야 하는 경우 데몬셋을 사용한다.
- HorizontalPodAutoscalers를 개발자가 매니페스트에 포함하거나 나중에 운영 팀에서 시스템에 추가하게 되면, 클러스터 관리자는 개별 포드 및 모든 포드의 컴퓨팅 리소스 사용을 제어할 수 있도록 LimitRange 및 ResourceQuota 객체를 생성한다.
- 애플리케이션이 배포되고 나면 쿠버네티스 컨트롤러에 의해 다양한 객체가 추가적으로 자동 생성된다.
 - 엔드포인트 컨트롤러로 생성한 서비스 엔드포인트 객체
 - 디플로이 컨트롤러로 생성한 레플리카셋
 - 레플리카셋(또는 Job, CronJob, 스테이트풀셋, 데몬셋)으로 생성한 포드
- 리소스들을 조직적으로 관리하기 위해 하나 이상의 라벨을 사용해 분류한다.
 - 라벨 외에도 해당 리소스를 담당하는 사람이나 팀의 연락처, 기타 도구에 대한 추가 메타데이터 등과 같이 리소스를 설명하는 주석(annotation)이 존재
- 이 모든 것의 중심에는 포드가 있으며 쿠버네티스 리소스들 중 가장 중요하다고 할 수 있다.

17.2 포드의 라이프사이클

17.2.1 애플리케이션은 종료하고 재배포해야 한다

- 쿠버네티스를 사용하지 않고 머신에 직접 애플리케이션을 구성하는 경우에는 애플리케이션을 다른 머신으로 이동하는 경우는 거의 없다.
 - 이동하는 경우 새로운 머신에 애플리케이션을 다시 구성하고 정상 작동하는지 수동으로 확인해야한다.
- 쿠버네티스를 사용하면 애플리케이션을 훨씬 자주 그리고 자동으로 이동할 수 있고, 별도의 확인 없이도 정상 동작을 보장한다.
 - 애플리케이션이 이동하더라도 정상 동작이 가능한 상태(의존성이 없는 상태)로 애플리케이션을 만들어야한다.

변경될 로컬 IP 및 호스트 이름 예상하기

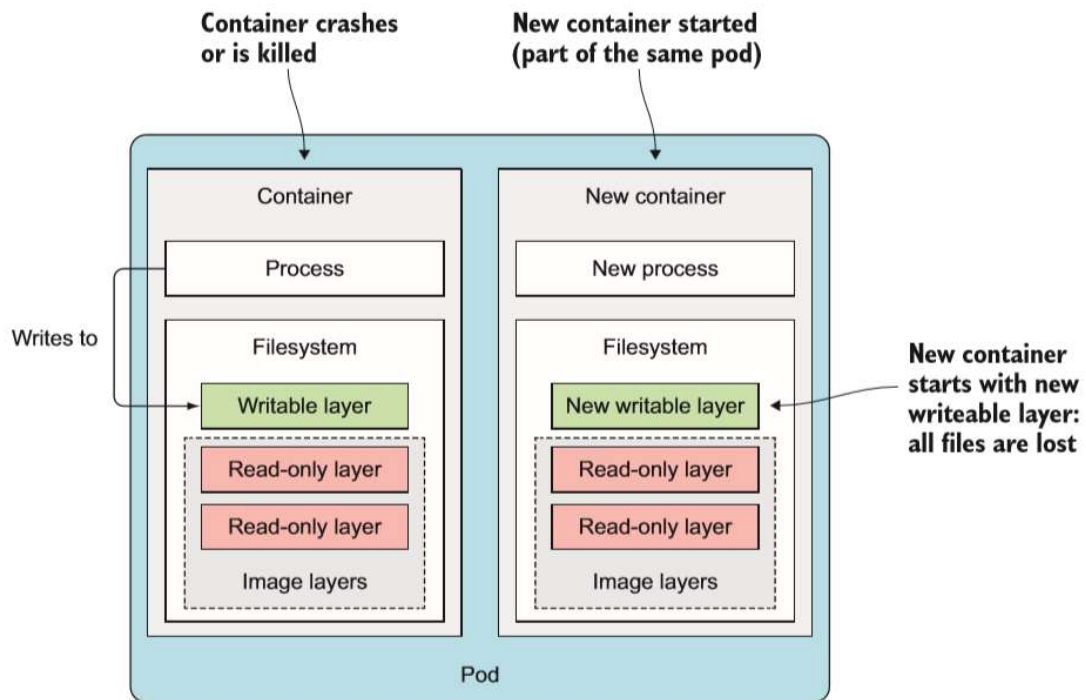
- 포드가 종료되고 다른 곳으로 옮겨지면 새로운 IP 주소, 새로운 호스트 이름이 부여된다.
- 상태 비 저장 애플리케이션의 경우에는 보통 부작용 없이 처리할 수 있다.

- 상태 저장이 필요한 경우에는 스테이트풀셋을 사용해야한다.
 - 스케줄러를 통해 새 노드에서 포드가 생성될 때 동일한 호스트 이름과 영구 상태를 계속 유지할 수 있도록 한다.
 - 이 경우에도 포드의 IP 주소는 변경된다.
 - 따라서 쿠버네티스 클러스터 내에서 IP 주소를 기반으로 애플리케이션들 간의 의존성을 갖도록 하면 안되고, 의존성을 갖더라도 호스트 이름을 기반으로 하는 스테이트풀셋을 사용해야한다.

디스크에 기록된 데이터가 사라지는 경우 예상하기

- 영구 저장 장치를 마운트하지 않은 경우 오토스케일링을 통해 새롭게 생성된 포드는 기존에 디스크에 저장했던 데이터를 사용할 수 없다.

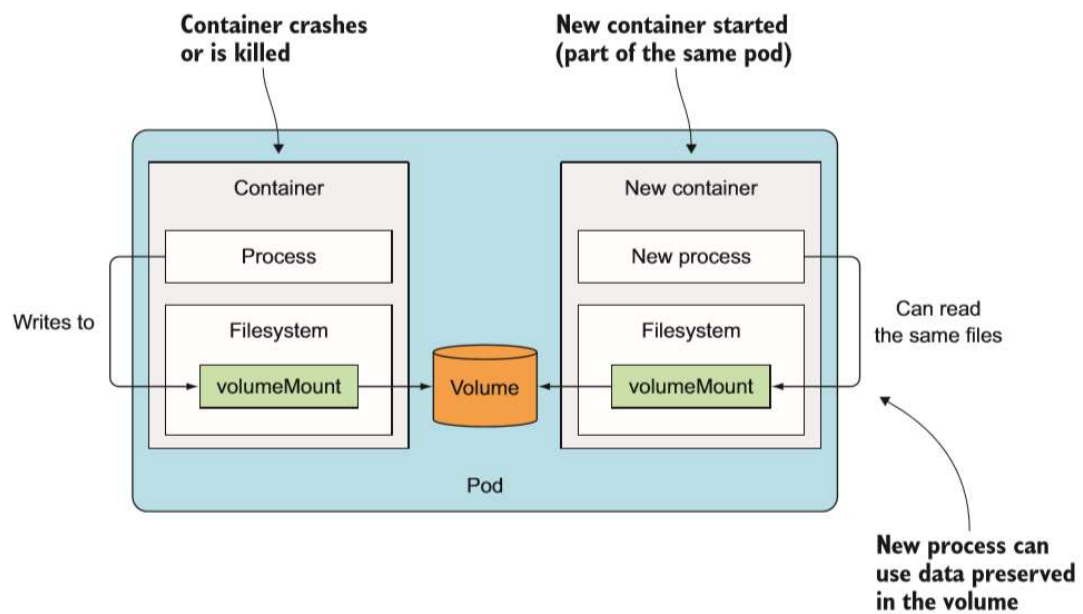
- 포드의 라이프 사이클 동안에는 데이터가 유지 될 것이라 믿을 수 있지만 애플리케이션에 의해 디스크에 기록된 파일이 사라질 수도 있다.



- 예를 들어 애플리케이션 시작 시 계산이 오래걸리는 절차를 포함하는 경우 다음 구동 시에는 빨리 구동될 수 있도록 계산한 결과를 디스크에 기록한다고 가정해보자. 애플리케이션은 기본적으로 컨테이너에서 실행되기 때문에 이 계산 결과 파일은 컨테이너의 파일 시스템에 기록된다. 위 그림과 같이 컨테이너가 다시 시작되면 새로운 컨테이너는 완전히 새로운 writable 레이어로 시작되기 때문에 모든 데이터가 손실된다.
- 동작 중인 컨테이너에 오류가 발생하거나 노드의 메모리 부족으로 OOMKiller에 의해 종료되거나, 프로세스가 충돌하는 등의 여러 이유로 컨테이너는 다시 시작될 수 있다. 이 경우 포드는 동일하지만 컨테이너는 완전히 새 것이다.
 - kubelet이 같은 컨테이너를 stop/start 하는 경우는 없다. 항상 새로운 컨테이너를 생성한다.

컨테이너를 다시 시작하더라도 데이터의 보존을 위해 볼륨 사용

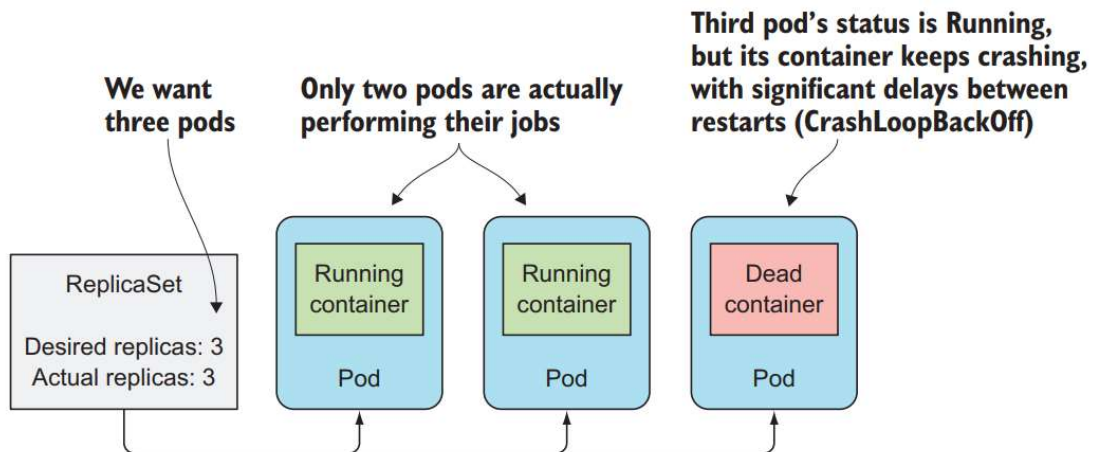
- 데이터가 유실되지 않도록 하려면 최소한 포드 범위의 볼륨을 사용해야 한다.



- EmptyDir은 포드와 라이프 사이클을 함께 하기 때문에 컨테이너 재시작으로 인한 데이터 유실을 방지 할 수 있다.
- 볼륨을 사용해서 컨테이너 재시작 시 파일을 보존하는 것이 항상 좋은 아이디어는 아니다.
 - 데이터가 손상돼 새로 생성된 프로세스에서 다시 크래시가 발생할 경우 크래시가 반복적으로 발생하게 된다.
 - CrashLoopBackOff 상태로 표시
 - 볼륨을 사용하지 않았다면 크래시가 발생하더라도 컨테이너가 재시작되면 크래시가 다시 발생하지 않을 가능성이 크다.
 - 볼륨을 사용하는 것은 양날의 검이기 때문에 사용 여부를 신중하게 생각해야 한다.

17.2.2 죽은 포드 또는 부분적으로 죽은 포드의 재스케줄링

- 포드의 컨테이너에서 크래시가 발생하면 kubelet은 무한정 재시작을 수행한다.
 - 이 때 restartPolicy에 의해 5분으로 제한된 지수 백-오프 지연(10초,20초,40초...)을 기준으로 재시작하고, 지수 값이 5분(320초)이 되었을 때 재설정된다. (10+20+ ... 320 = 10분정도 됨)
 - 즉, 크래시 발생하고 재시도 실행 텀이 점진적으로 증가하다가 10분이 되면 다시 10초부터 점진적으로 재시도 실행 텀이 증가한다.
 - Note : 레퍼런스에는 재설정된다고 되어 있는데 다시 10초부터 시작인지, 320초가 되면 이후부터는 320초 주기로 재시작하는지 확인 필요
- 레플리카셋과 같은 컨트롤러를 통해 포드가 관리되고 있는 경우 크래시가 발생하더라도 제거되지 않고 이로 인해 포드가 재스케줄되지 않는다.



- 레플리카셋 컨트롤러는 포드가 죽었는지는 상관하지 않고 포드의 수가 원하는 복제 카운트와 일치하는 것만 체크한다. 크래시가 나더라도 포드의 복제 카운트는 유지되기 때문에 재스케줄 되지 않는다.
- 쿠버네티스 클러스터에서 크래시 난 포드를 자동으로 재생성 하더라도 해당 포드는 그대로 크래시가 발생할 확률이 높기 때문에 쿠버네티스에서는 지속적인 재시도를 통해 크래시의 근본 원인이 해결되길 바라는 것으로 보인다.

17.2.3 원하는 순서로 포드 시작하기

포드의 시작 방식

- 쿠버네티스 API 서버는 YAML/JSON 파일에 나열된 순서대로 객체를 처리하고 이는 나열된 순서대로 etcd에 기록된다는 것을 의미한다.

- 포드는 순서대로 시작된다는 보장이 없다.
 - 포드에 initContainer를 포함시켜 전제 조건이 충족될 때까지 포드의 주 컨테이너가 시작되는 것을 방지할 수는 있다.

initContainer 소개

- 포드를 초기화하는 데 사용할 수 있는 컨테이너이다.
- 동일한 포드에서 실행되기 때문에 주 컨테이너에 마운트된 포드의 볼륨에 데이터를 쓸 수 있다.
- 하나의 포드에 여러개의 initContainer가 존재할 수 있다.
 - 정의한 순서대로 실행되며 마지막 initContainer가 완료된 후에 주 컨테이너가 시작된다.
 - 이를 이용하여 특정 전제 조건이 충족될 때까지 포드의 주 컨테이너 시작을 지연시킬 수 있다.

```
spec:
  initContainers:
    - name: init
      image: busybox
      command:
        - sh
        - -c
        - 'while true; do echo "Waiting for fortune service to come up...";
        ↪ wget http://fortune -q -T 1 -O /dev/null >/dev/null 2>/dev/null
        ↪ && break; sleep 1; done; echo "Service is up! Starting main
        ↪ container."'
```

← You're defining an init container, not a regular container.

The init container runs a loop that runs until the fortune Service is up.

포드 간 의존도를 다루는 모범 사례

- initContainer를 사용하여 주 컨테이너의 시작을 지연시키는 것 보다는 애초에 의존하고 있는 모든 서비스의 준비를 기다릴 필요가 없는 애플리케이션을 제작하는 편이 훨씬 낫다.
- 그렇지 못한 경우 의존성이 있는 서비스들의 준비 상태 체크를 내부적으로 처리할 수 있어야 한다.
 - 즉, readinessProbe를 사용하라는 것이다.
 - readinessProbe를 사용하면 서비스가 준비될 때까지 해당 애플리케이션을 서비스 엔드포인트에 추가하지 않을 뿐만아니라 롤링 업데이트를 수행할 때도 롤아웃을 방지할 수 있다.

17.2.4 라이프사이클 훅 추가

- `initContainer`를 사용하여 포드 시작 시 전처리를 할 수 있지만 포드의 라이프사이클 단계에서 두가지 훅을 정의할 수도 있다.
 - `post-start` 훅
 - `pre-stop` 훅
- `initContainer`는 전체 컨테이너 시작 전에 실행되는 반면 라이프사이클 훅은 각 컨테이너별로 실행한다.
- 라이프사이클 훅 실행 방법은 `livenessProbe`와 `readinessProbe` 동작 방식과 유사하다.
 - 컨테이너 내부에서 명령 실행
 - URL에 HTTP GET 요청 수행

post-start 컨테이너 라이프사이클 훅 사용

- 컨테이너의 주 프로세스가 시작되고 난 다음 바로 실행된다.
 - 애플리케이션 시작 시 추가 조작을 위해 사용
 - 애플리케이션 소스 코드 내에 시작시 수행할 작업을 작성할 수 있지만 `post-start` 훅을 사용하면 소스코드 수정 없이 추가 명령을 실행할 수 있다.
- 컨테이너의 주 프로세스와 병렬로 실행될 수도 있다.
 - 주 프로세스가 완전히 실행되었음을 `kubelet`이 인지할 수 없기 때문에 주 프로세스가 실행완료될 때까지 기다릴 수가 없다.
- 훅이 완료될 때까지 컨테이너는 `ContainerCreating`인 채로 `Waiting` 상태를 유지한다.
 - 이로 인해 포드의 상태는 `Running` 대신 `Pending` 상태가 된다.
- 훅이 정의되었는데 실행되지 않거나 `exit code`가 0이 아닌 값을 반환하게 되면 주 컨테이너가 종료된다.

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-with-poststart-hook
spec:
  containers:
    - image: luksa/kubia
      name: kubia
      lifecycle:
        postStart:
          exec:
            command:
              - sh
              - -c
              - "echo 'hook will fail with exit code 15'; sleep 5; exit 15"

```

It executes the `postStart.sh` script in the `/bin` directory inside the container.

The hook is executed as the container starts.

- 위 예제를 실행하면 컨테이너가 생성되는 즉시 컨테이너의 주 프로세스와 함께 `postStart` 후크가 실행된다.
- 라이프사이클 후크의 실행 결과는 쿠버네티스에 로깅되지 않고 표준출력으로 로깅되기 때문에 실패할 경우 `kubectl describe pod` 명령으로 확인해보면 `FailedPostStartHook` 경고만 표시된다.
 - 위 예제에서는 `exit code`를 15로 반환했기 때문에 실패하여 아래와 같은 오류가 발생한다.

```

FailedSync Error syncing pod, skipping: failed to "StartContainer" for
"kubia" with PostStart handler: command 'sh -c echo 'hook
will fail with exit code 15'; sleep 5 ; exit 15' exited
with 15: : "PostStart Hook Failed"

```

- 명령 실행 결과를 확인해보고자 하는 경우에는 실행 결과를 파일로 기록하고 `kubectl exec` 명령을 사용하여 확인해볼 수 있다.
 - `post-start` 후크의 실행이 실패하면 컨테이너가 종료되지는 않지만 어떠한 이유에서든 컨테이너는 종료될 수 있기 때문에 로깅 파일을 유지하려면 `emptyDir` 볼륨을 사용해야한다.

pre-stop 컨테이너 라이프사이클 후크 사용

- 컨테이너가 종료되기 직전에 실행된다.
 - `pre-stop` 후크를 실행 한 후에는 프로세스로 `SIGTERM`을 전송한다.
- `pre-stop` 후크는 `post-start` 후크와 달리 후크의 결과에 관계없이 컨테이너는 종료된다.
- `pre-stop` 후크가 실패하면 `FailedPreStopHook` 경고 이벤트가 표시되지만 컨테이너가 바로 종료되기 때문에 보지 못하고 지나칠 수도 있다.
 - `pre-stop` 후크의 성공적인 완료가 중요한 애플리케이션의 경우 전체 시스템이 올바르게 실행 되고 있는지 확인이 필요하다. 필자 경험상 인지하지 못하고 지나친 상황을 목격한적이 있다고 함

애플리케이션이 SIGTERM 신호를 수신하지 않는 이유로 pre-stop 훅 사용

- kubelet이 컨테이너 종료 시 애플리케이션으로 SIGTERM 신호를 보내는 것을 확인하지 못한 많은 개발자들이 pre-stop 훅을 사용하여 SIGTERM 신호를 보내려고 한다.
 - 쿠버네티스가 SIGTERM 시그널을 애플리케이션에 보내지 않은 것이 아니라 시그널이 컨테이너 내부의 애플리케이션 프로세스로 전달되지 않은 것이다.
 - 컨테이너 구동 시 ENTRYPOINT 또는 CMD를 사용하여 쉘 명령으로 애플리케이션을 구동하는 경우 애플리케이션이 쉘 명령의 자식 프로세스로 구동되는데 SIGTERM 시그널을 쉘 명령이 받을 수도 있다.
 - 애플리케이션에 직접 보내려면 pre-stop 훅에 SIGTERM 시그널을 추가하는 것이 아니라 쉘이 시그널을 애플리케이션에 전달하는지를 확인해야 한다.
 - 쉘을 사용하여 애플리케이션을 구동하지 말고 컨테이너 구동 시 애플리케이션 바이너리를 직접 실행하도록 하면 된다.
 - Docker 파일에 ENTRYPOINT 나 CMD를 사용하여 ENTRYPOINT ["/mybinary"] 또는 ENTRYPOINT /mybinary 와 같이 작성한다.

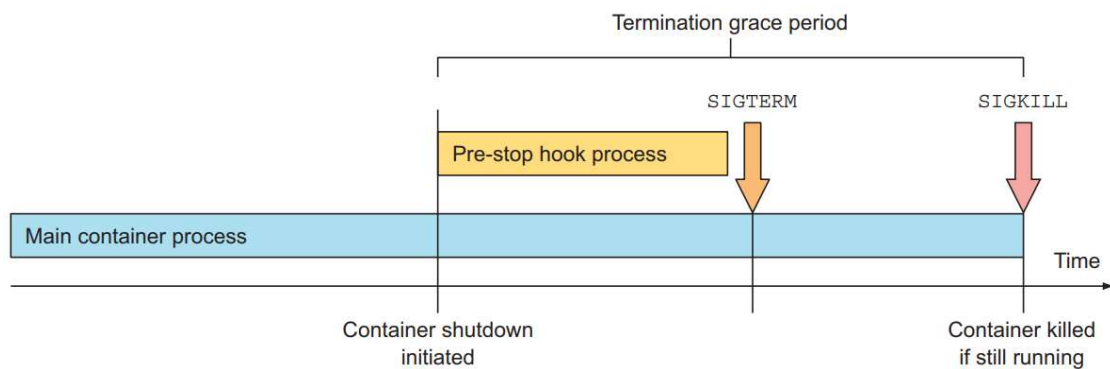
포드가 아닌 컨테이너를 타겟으로 하는 라이프사이클 훅

- 라이프사이클 훅은 포드를 위한 것이 아닌 컨테이너를 위한 것임에 주의해야 한다.
- 포드가 종료될 때 수행해야 하는 작업을 실행하기 위해 pre-stop 훅을 사용해서는 안 된다.
- 컨테이너 단위의 명령이기 때문에 포드의 라이프사이클 내에서 여러번 발생할 수 있다.
 - 포드 종료시 필요한 작업을 놓게 되면 반복 실행으로 인한 문제가 발생할 수 있다.

17.2.5 포드 셋 다운

- 포드 종료는 API 서버의 포드 객체 삭제에 의해 트리거된다.
- HTTP DELETE 요청을 받으면 API 서버는 객체를 바로 삭제하지는 않고 삭제를 위한 deletionTimestamp 필드를 설정한다.

- kubelet이 포드를 종료해야 한다는 사실을 인지하게 되면 포드 내 컨테이너들을 종료하기 시작한다.
 - 이 때 각 컨테이너가 정상적으로 종료될 수 있도록 제한된 시간(the termination grace period)이 주어진다.
 - 포드 단위로 설정할 수 있다.
- 종료 프로세스가 시작되자마자 타이머가 시작되며 다음 작업들이 수행된다.



1. pre-stop 훅이 구성된 경우 훅을 실행하고, 완료될 때까지 대기한다.
2. SIGTERM 신호를 컨테이너의 메인 프로세스로 보낸다.
3. 컨테이너가 완전히 셧다운 되거나 종료 유예 기간이 만료될 때까지 대기한다.
4. 정상적으로 종료되지 않는 경우 SIGKILL로 프로세스를 강제 종료한다.

종료 유예기간 설정

- 포드 스펙의 terminationGracePeriodSeconds 필드에서 설정할 수 있다.
 - 필드 값은 초 단위이며 기본 값은 30초
 - 프로세스가 정리를 끝마칠 수 있는 충분한 시간을 설정해야한다.
- 포드 삭제 명령의 옵션으로 종료 유예기간을 지정할 수도 있다.

```
kubectl delete po mypod --grace-period=5
```

Shell

- 모든 포드의 컨테이너가 종료되면 kubelet이 API 서버에 알리고 포드 리소스가 최종적으로 삭제된다.

- 종료 유예기간을 주지 않고 강제 종료를 하려면 다음 명령을 수행한다.

```
kubect1 delete po mypod --grace-period=0 --force
```

Shell

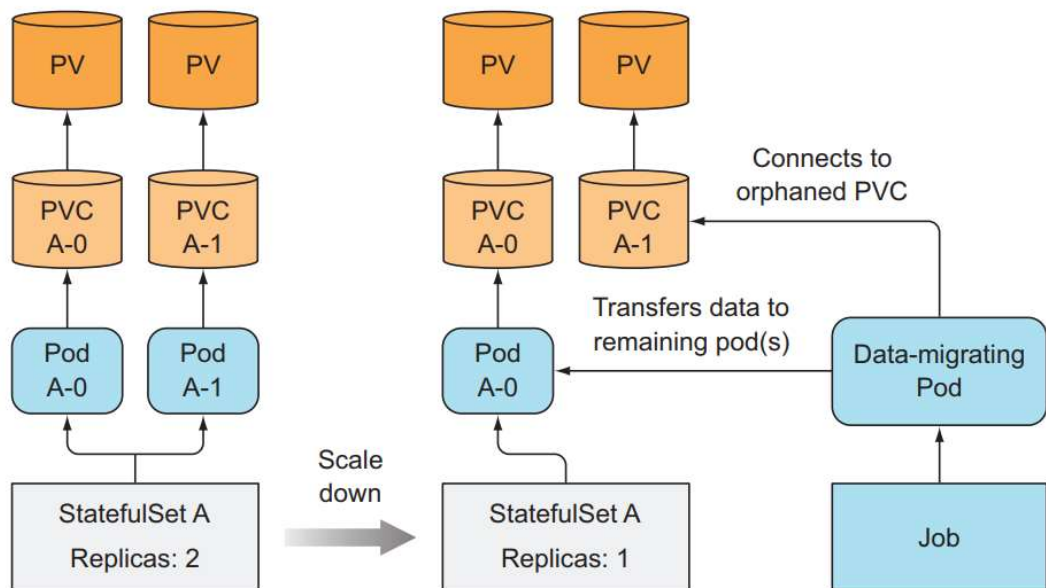
- 스테이트풀셋의 포드인 경우 이 옵션의 사용을 주의해야한다.
 - p707 내용 이해가 안됨

애플리케이션에 적절한 섯다운 핸들러 구현

- 애플리케이션이 완전히 종료되는데까지 걸리는 시간을 예측할 수 없다면 어떻게 될까?
 - 애플리케이션이 분산 데이터 스토리지라고 가정할 때 스케일을 축소하면 포드 인스턴스 중 하나가 삭제되어 종료된다.
 - 포드 인스턴스가 삭제되면 데이터가 유실되기 때문에 종료 절차에서 포드는 모든 데이터가 유실되지 않도록 나머지 포드로 마이그레이션할 필요가 있다.
 - 하지만 다음 두가지 이유로 마이그레이션 하는 것을 권장하지 않는다.
 - 컨테이너의 종료가 반드시 전체 포드의 종료를 의미하지는 않는다.(?)
 - 프로세스가 종료되기 전에 종료 절차가 끝날 것이라는 보장이 없다.
 - 종료 유예기간이 만료되거나 종료 진행 중에 노드가 실패하는 경우
 - 노드가 실패할 경우 노드가 복구 되더라도 kubelet은 종료 절차를 다시 시작하지는 않는다.

전용 섯다운 프로시저 포드를 사용해 중요한 섯다운 프로시저를 대체

- 위에서 분산 데이터 스토리지 예제에서와 같이 완료까지 절대적으로 실행해야 하는 중요한 섯다운 프로시저의 경우 완료를 어떻게 보장받을 수 있을까?
 - 종료 신호를 받으면 애플리케이션을 사용해 삭제된 포드의 데이터를 나머지 포드로 마이그레이션하는 새로운 작업을 수행하는 새로운 포드를 위한 새 잡 리소스를 만드는 것이다.
 - 하지만 애플리케이션이 잡 객체를 생성하는 시점에 노드가 실패하게 되는 경우가 발생할 수 있으므로 매번 잡 객체가 생성될 것이라는 보장이 없다.
 - 해결책은 끊임없이 실행되는 전용 포드를 사용해 분리된 데이터의 존재를 계속 확인하는 것이다.
 - 끊임없이 실행되는 전용 포드 대신 CronJob 리소스 사용 가능
 - 스테이트풀셋은 적합하지 않다.
 - 스테이트풀셋을 축소하면 PersistentVolumeClaim이 고아가 되고 PersistentVolume에 저장된 데이터는 그대로 남는다.
 - 스테이트풀셋이 다시 확장되면 해당 PersistentVolume에 다시 연결된다.



- 애플리케이션을 업그레이드 하는 도중 마이그레이션이 발생하지 않도록 데이터 마이그레이션 포드를 구성한다.(?)
- 데이터 마이그레이션 포드는 마이그레이션을 수행하기 전에 다시 실행할 수 있도록 stateful 포드에게 시간을 준다. (?)

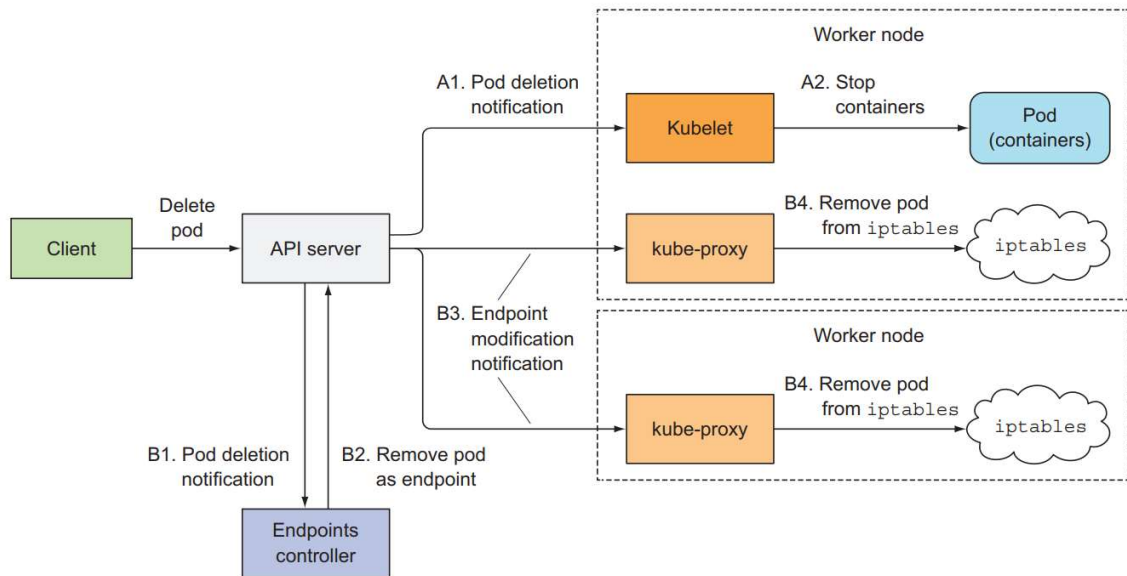
17.3 모든 클라이언트 요청이 올바르게 처리되도록 보장하기

17.3.1 포드가 시작할 때 클라이언트의 연결 끊어짐 방지

- 포드가 시작되면 포드의 라벨과 일치하는 라벨 셀렉터가 있는 모든 서비스에 엔드포인트가 추가된다.
 - 포드 시작 시 준비가 완료 되었음을 쿠버네티스에 알리기 전까지는 서비스 엔드포인트가 추가되지 않기 때문에 클라이언트의 어떠한 요청도 받지 못한다.
- 포드 스펙에 readinessProbe를 지정하지 않을 경우 포드는 항상 준비 상태인 것으로 간주한다.
 - kube-proxy가 노드에 iptables 규칙을 업데이트하고나면 거의 즉시 요청을 받을 수 있다.
 - 이 때까지 애플리케이션이 연결을 수락할 준비가 되지 않았다면 클라이언트는 "connection refused"와 같은 오류를 보게 될 것이다.
 - readinessProbe를 설정하면 이와 같은 오류를 방지할 수 있다.

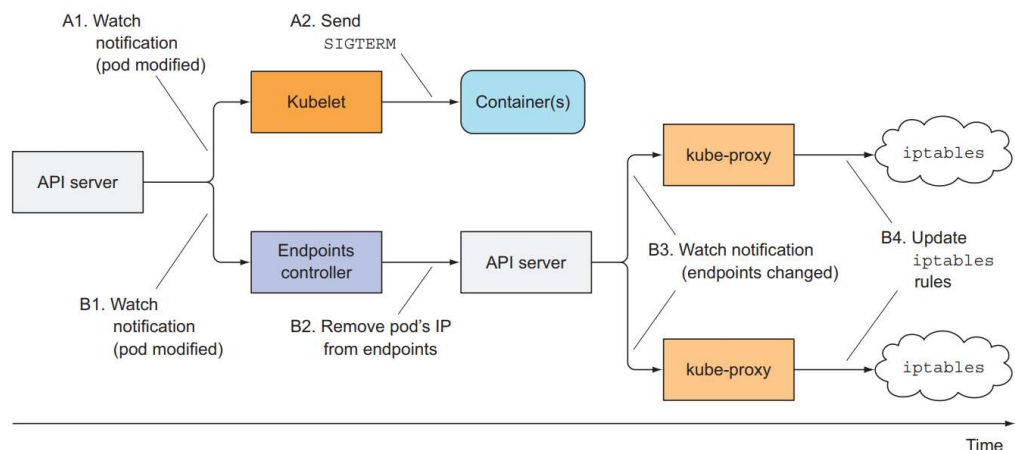
17.3.2 포드 셧다운 동안 연결 끊어짐 방지

포드 삭제 시 발생하는 이벤트의 순서 이해



- 포드 삭제 요청을 API 서버가 수신하면 먼저 etcd에서 상태를 수정한 다음 해당 삭제의 관찰자에게 알린다.
 - 관찰자 중에는 Kubelet과 Endpoints 컨트롤러가 있다.
- 위 예제에 병렬로 발생하는 두 개의 이벤트 시퀀스 A와 B가 있다.
- A 이벤트
 - Kubelet이 포드를 종료해야 한다는 알림을 받으면 첫다운 시퀀스를 시작한다.
 - pre-stop 후 실행 → SIGTERM 시그널 전달 → 종료 유예기간을 초과하면 강제 종료
 - 애플리케이션이 SIGTERM 시그널을 수신하고 나면 연결을 요청한 클라이언트는 Connection Refused 오류를 수신한다.
- B 이벤트
 - 쿠버네티스 컨트롤 플레인의 컨트롤러 매니저에서 실행되는 엔드포인트 컨트롤러가 포드의 삭제 알림을 받으면 포드가 포함된 모든 서비스의 엔드포인트에서 해당 포드를 제거한다.
 - API 서버는 엔드포인트 객체를 모니터링하고 있는 모든 클라이언트들에게 통지한다.

- 각 워커 노드에서 실행되는 모든 kube-proxy가 모니터링 하고 있다.
- 통지를 받은 각 kube-proxy들은 각자의 노드에서 iptables 규칙을 업데이트하여 새로운 연결이 종료 중인 포드로 전달되지 않도록 한다.
 - iptables 규칙을 제거하더라도 기존에 연결되어 있는 connection에는 영향을 주지 않는다.
- 대부분의 경우 포드에서 애플리케이션의 프로세스를 종료하는 데 걸리는 시간이 iptables 규칙을 업데이트하는 데 필요한 시간보다 약간 짧다.
 - 이벤트가 엔드포인트 컨트롤러에 도달하고 API 서버를 통해 kube-proxy에 전달되어 iptables를 수정하기까지의 시간이 소요되기 때문이다.
 - 즉, iptables 규칙이 모든 노드에서 업데이트 되기 전에 SIGTERM 신호가 전송될 확률이 높다는 의미이고, 이는 종료 신호를 보낸 후에도 iptables가 갱신되기 전까지는 계속 클라이언트의 요청을 수신할 수 있다는 것이다.

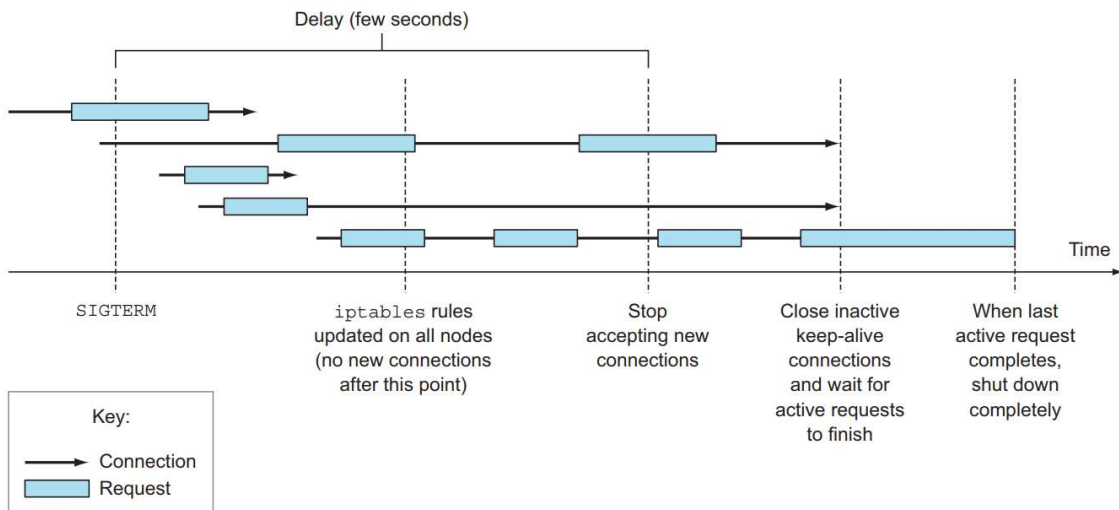


문제 해결

- readinessProbe를 사용할 경우 포드가 SIGTERM을 수신하자마자 readinessProbe의 진단이 실패하여 포드가 서비스의 엔드포인트에서 제거되어야 한다.
 - 하지만 readinessProbe를 통해 엔드포인트에서 제외되려면 readinessProbe 검사에 몇 번 연속 실패가 발생해야 한다.
- 문제를 완벽하게 해결할 수 있는 방법은 없다. 모든 프록시가 자신의 업무를 수행할 때까지 충분히 기다리는 수 밖에 없다.
- 5~10초 정도 지연 시간을 추가하는 것만으로도 사용자 경험(user experience)이 상당히 개선된다.

정리

- 애플리케이션을 적절히 종료하려면?
 - 새 연결을 수락하기까지 몇 초간 기다려라
 - 요청 중간에 있지 않은 모든 keep-alive 연결을 닫아라.
 - 모든 활성 요청이 완료될 때까지 기다리자.
 - 이 후에 완전히 셧다운 한다.



- 종료 신호를 수신하자마자 프로세스를 종료하는 것만큼 간단하지는 않다.
- 이 모든 절차를 수행할지는 관리자가 결정할 일이다.
- 다음 예제와 같이 몇 초 정도 기다리는 명령을 pre-stop 후에 추가하는 것이 가장 좋다.

```
lifecycle: preStop: exec: command: - sh - -c - "sleep 5"
```

Shell

- 애플리케이션 코드를 전혀 수정할 필요 없이 이미 진행 중인 요청이 완전히 처리될 때까지 기다릴 수 있다.

17.4 쿠버네티스에서 애플리케이션을 쉽게 실행하고 관리할 수 있게 만들기

17.4.1 관리 가능한 컨테이너 이미지 만들기

- 애플리케이션을 이미지로 패키징 할 때 바이너리와 함께 추가 라이브러리들을 포함하거나 전체 OS 파일 시스템을 함께 패키징 할 수도 있다.
 - 예를들어 nginx 이미지를 만들 때 실행에 필요한 경량 OS인 debian 기반에서 nginx만 설치할 수도 있지만 ubuntu 이미지를 기반으로 nginx를 설치할 수도 있다.
- 새로운 포드를 배포하고 스케일 조절이 빠를 수 있도록 작은 크기의 이미지를 사용하는 것을 권장한다.
- 작은 크기의 이미지도 무조건 좋지만은 않다.
 - 크기가 작은 만큼 디버깅에 필요한 도구들(ping, dig, curl 등) 또한 포함되어 있지 않기 때문에 디버깅이 어려워질 수 있다.
 - 작은 크기의 이미지를 사용하면서 필요한 도구들을 추가 설치하여 이미지를 패키징하는 등의 자신만의 방식을 찾아야한다.

17.4.2 이미지에 적절히 태그하고 imagePullPolicy를 현명하게 사용

- 이미지 태그로 latest를 지정하는 경우 새로운 이미지를 패키징해서 다시 latest로 푸시를 하게되면 Scale out 시 새로 생성된 포드는 새로운 이미지를 사용하겠지만 기존 포드들은 기존 이미지를 유지하게 된다.
 - 이 경우 롤백을 하려고 해도 이전 버전의 이미지가 생성되어 있지 않기 때문에 이전 버전을 다시 latest로 패키징해서 푸시해야한다.
- latest가 아닌 적절한 버전 지정자를 포함하는 태그를 사용하는 것이 필수적이다.
- 동일한 태그에 변경사항을 업데이트하고 포드에 반영할 수 있으려면 imagePullPolicy 필드를 Always로 설정해야 한다.
 - Always 설정 시 새로운 포드를 배포할 때마다 레지스트리에 연결되어 변경 여부를 체크하기 때문에 포드의 시작이 약간 느려질 수 있음을 주의해야 한다.
 - 또한 레지스트리에 연결이 불가능한 경우 포드 실행이 불가능해지는 문제가 발생한다.

17.4.3 단일 차원 라벨 대신 다차원 라벨 사용

- 포드 뿐만 아니라 모든 리소스에 라벨을 지정할 수 있다는 것을 잊으면 안된다.

- 각 리소스에 여러 개의 라벨을 추가해야 리소스들을 세세하게 선택할 수 있다.
 - 리소스의 수가 증가할 수록 라벨을 여러개 추가한 것에 감사하게 될 것이다.
- 라벨에 포함 될만한 것들은 다음과 같다.
 - 리소스가 속한 애플리케이션의 이름
 - 프론트엔드/백엔드와 같은 애플리케이션 계층
 - 개발, QA, 스테이징, 프로덕션과 같은 환경
 - 버전
 - 릴리즈 유형 (블루/그린, 카나리아, 롤링 등)
 - 테넌트 (namespace 대신 테넌트 별로 별도의 포드를 실행하는 경우)
 - 샷드

17.4.4 annotation을 통해 각 리소스 설명하기

- 리소스에 정보를 추가하고자 할 때 annotation을 사용하는데 최소한 리소스에는 리소스를 설명하는 annotation 작성자의 연락처가 존재해야 한다.
- 포드가 사용하는 다른 서비스의 이름을 나열하는 annotation을 포함할 수 있다.
 - 포드 사이의 종속성을 나타낼 수 있다.
- 도구 및 GUI에 사용되는 빌드나 버전 정보, 메타데이터와 같은 정보를 포함할 수 있다.
- 애플리케이션이 충돌할 때 이유를 확인하기 위해 annotation과 라벨 정보가 많은 것이 좋다.

17.4.5 프로세스가 종료된 원인 제공

- 컨테이너가 종료되는 최악의 순간에 왜 컨테이너가 종료되었는지 원인을 알 수 없는 것보다 좌절스러운 것은 없다.
 - 로그파일에 필요한 모든 디버깅 정보를 포함시키자.

- 쿠버네티스 기능을 사용하여 컨테이너가 종료된 이유를 표시할 수 있다.
 - 프로세스가 컨테이너의 파일 시스템에 있는 특정 파일에 종료 메시지를 작성함으로써 쿠버네티스에서 인지하도록 할 수 있다.
 - `kubectl describe pod`의 출력에 표시된다.
 - 컨테이너 종료시 작성되는 로그파일의 컨테이너와 쿠버네티스 간 약속된 기본 파일 위치는 `/dev/termination-log` 이지만 `terminationMessagePath` 필드를 설정해서 변경할 수 있다.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-termination-message
spec:
  containers:
  - image: busybox
    name: main
    terminationMessagePath: /var/termination-reason
    command:
    - sh
    - -c
    - 'echo "I've had enough" > /var/termination-reason ; exit 1'
```

You're overriding the default path of the termination message file.

The container will write the message to the file just before exiting.

- 위 예제에서 `exit code`를 1로 지정했기 때문에 컨테이너는 오류를 반환하고 종료될 것이다.
- 이 후 포드 상태가 `CrashLoopBackOff`로 표시되고, `kubectl describe` 명령을 사용하여 확인해보면 다음과 같이 메시지를 확인할 수 있다.

```
$ kubectl describe po
Name: pod-with-termination-message
...
Containers:
...
  State: Waiting
    Reason: CrashLoopBackOff
  Last State: Terminated
    Reason: Error
    Message: I've had enough
    Exit Code: 1
    Started: Tue, 21 Feb 2017 21:38:31 +0100
    Finished: Tue, 21 Feb 2017 21:38:31 +0100
```

You can see the reason why the container died without having to inspect its logs.

- 위 예제에서는 /var/termination-reason과 같이 terminationMessagePath에 지정된 경로에 로그를 기록하였는데 더 간단한 방법으로 terminationMessagePolicy 필드를 FallbackToLogsOnError로 설정할 경우 컨테이너가 성공적으로 종료되지 않는 경우 컨테이너 로그의 마지막 몇 줄이 종료 메시지로 사용된다.

17.4.6 애플리케이션 로그 핸들링

- 애플리케이션이 파일 대신 표준 출력으로 로그를 기록할 경우 `kubectl logs` 명령을 사용하여 로그를 볼 수 있다.
 - 컨테이너가 크래시되고 새 컨테이너로 교체되면 새 컨테이너의 로그만 표시된다.
 - 이전 컨테이너의 로그를 보려면 `--previous` 옵션을 사용한다.
- 로그가 파일에 기록되는 경우 `kubectl exec <pod> cat <logfile>` 명령으로 로그를 확인할 수 있다.

컨테이너 내 로그 파일 복사

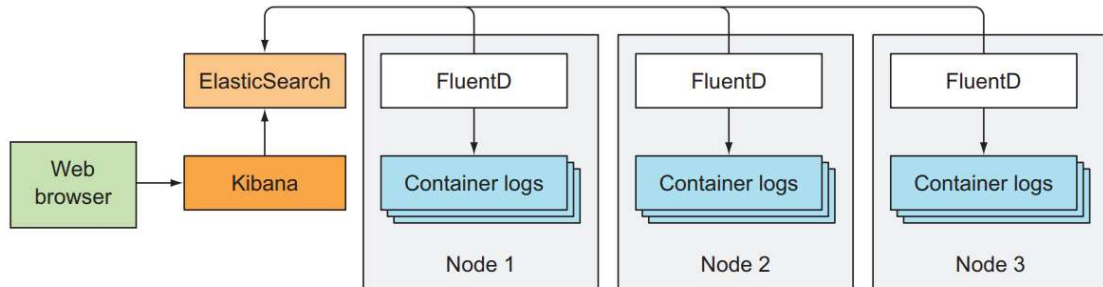
- `kubectl cp` 명령을 사용하여 컨테이너 내의 로그 파일을 로컬 시스템에 복사할 수 있다.
- 로컬 시스템 파일을 컨테이너로 복사도 가능하다.

```
# 컨테이너의 로그 파일을 로컬 시스템으로 복사 kubectl cp foo-
pod:/var/log/foo.log foo.log # 로컬 시스템 파일을 컨테이너로 복사 kubectl
cp localfile foo-pod:/etc/remotefile
```

Shell

중앙 집중식 로깅 사용

- 쿠버네티스는 중앙 집중식 로깅을 지원하지 않는다.
- 중앙 집중식 저장 및 분석을 위한 특정 포드를 구성하는 방식을 사용한다.
- EFK(Elasticsearch + logstash + fluentd) 사용



- 쿠버네티스 클러스터의 모든 노드는 fluentd 에이전트를 데몬셋으로 실행한다.
- 이를 통해 컨테이너에서 로그를 수집하고 포드의 특정 정보로 태그를 지정하여 엘라스틱서치로 전달한다.
- 엘라스틱서치는 쿠버네티스 클러스터 어딘가에 포드로 배포된다.
- 키바나를 통해 웹 브라우저에서 로그를 보고 분석할 수 있다.

여러 줄로 구성된 로그 문구 다루기

- fluentd 에이전트는 로그 파일의 각 행을 엔트리 단위로 엘라스틱서치에 저장한다.
- 자바의 예외 스택 트레이스와 같이 여러 행에 걸치는 로그들은 각각의 행이 로깅 시스템에서 각각 별도의 엔트리로 다루어진다.
 - 이를 해결하기 위해 일반 텍스트 대신 JSON을 출력하도록 설정할 수 있다.
- JSON 사용 시 kubectl logs 명령을 통해 로그를 확인하면 사람이 읽기 어려운 문제가 발생한다.
 - 사람이 읽을 수 있는 로그를 표준 출력으로 계속 출력하면서 동시에 JSON 로그를 파일에 쓰고 플루언트디에서 처리되도록 하면 된다.
 - 이를 위해 노드 수준의 fluentd 에이전트를 적절히 구성하거나 모든 포드에 포드 사이드카 컨테이너를 추가해야한다.

17.5 개발과 테스트의 모범 사례

17.5.1 개발 과정에서 쿠버네티스 외부에서 애플리케이션 실행

- 최종적으로 프로덕션에서 애플리케이션을 실행할 때는 쿠버네티스에서 실행하겠지만 개발 중에도 쿠버네티스 클러스터에서 해야 할까?
 - 개발 후 테스트를 위해 이미지를 빌드하고 쿠버네티스 클러스터에 적용하는 것은 힘든 일이다. 굳이 쿠버네티스 내 컨테이너에서 애플리케이션을 실행할 필요는 없다.
 - IDE에서 개발 및 테스트를 진행한다.

로컬에서 이미지를 빌드하고 미니큐브 VM으로 이미지를 직접 복사

- 로컬 컴퓨터에서 이미지를 빌드하는 경우 다음 명령으로 이미지를 미니 큐브 VM에 복사할 수 있다.

```
docker save <image> | (eval $(minikube docker-env) && docker load)
```

Shell

- imagePullPolicy가 Always로 설정된 경우 로컬 이미지를 사용하지 않고 레지스트리에서 이미지를 다시 가져오기 때문에 복사한 변경사항을 잃어버릴 수 있다.

적절한 쿠버네티스 클러스터가 있는 미니큐브 결합

- 애플리케이션을 미니큐브로 개발할 때 제약이 거의 없다.
- 쿠버네티스 클러스터에 미니큐브 클러스터를 조합할 수도 있다.
 - 로컬 미니큐브에서 개발하다가 수 킬로미터 떨어진 멀티 노드 쿠버네티스 클러스터에 배포되어 있는 다른 워크로드와 연결될 수도 있다.
- 개발이 완료되면 수정 없이 내 로컬 워크로드에서 원격 클러스터로 이동할 수 있다.
 - 쿠버네티스가 하부의 인프라를 추상화하고 있는 덕분

17.5.3 버전 관리 및 리소스 매니페스트 자동 배포

- 쿠버네티스는 선언적 모델을 사용한다.
 - 쿠버네티스에게 원하는 상태를 말하면 원하는 상태로 조정하는데 필요한 모든 조치가 취해진다.
 - 리소스 매니페스트 컬렉션을 버전관리시스템(이하 VCS)에 저장하여 코드 검토를 수행하고 감사 추적을함으로써 필요 시 롤백을 하도록 구성할 수도 있다.
 - `kubectl apply` 명령을 실행하여 변경 사항을 배포된 리소스에 반영한다.
 - 새로운 커밋이 감지되면 VCS에서 매니페스트를 체크아웃 한 후 `kubectl apply` 명령을 실행하는 방식으로 변경사항을 실행 중인 애플리케이션에 반영한다.
 - kube-applier라는 툴을 사용하면 쿠버네티스 API 서버와 통신하지 않아도 VCS를 사용하여 자동으로 변경사항을 반영할 수 있다.

17.5.4 YAML/JSON 매니페스트를 작성하는 대안으로 Ksonnet 소개

- YAML을 사용하는 것이 문제는 아니다.
 - `kubectl explain` 명령을 통해 사용 가능한 옵션을 볼 수도 있다.
- Ksonnet은 JSON 데이터 구조를 빌드하기 위한 데이터 템플릿 언어인 Jsonnet을 기반으로 구현된 라이브러리다.
 - 쿠버네티스 리소스 매니페스트의 설정들이 모듈화 되어 있기 때문에 재사용 가능하고 훨씬 적은 코드로 작성 가능하며 JSON 매니페스트를 신속하게 빌드할 수 있다.

```
local k = import "../ksonnet-lib/ksonnet.beta.1/k.libsonnet";
```

```
local container = k.core.v1.container;
```

```
local deployment = k.apps.v1beta1.deployment;
```

```
local kubiaContainer =
  container.default("kubia", "luksa/kubia:v1") +
  container.helpers.namedPort("http", 8080);
```

This defines a container called **kubia**, which uses the **luksa/kubia:v1** image and includes a port called **http**.

```
deployment.default("kubia", kubiaContainer) +
deployment.mixin.spec.replicas(3)
```

This will be expanded into a full **Deployment** resource. The **kubiaContainer** defined here will be included in the **Deployment's** pod template.

- 위 예제의 파일명이 `kubia.ksonnet`이라고 가정하면 `jsonnet kubia.ksonnet` 명령을 수행 시 JSON 매니페스트로 변환된다.

17.5.5 CI와 CD

- Fabric8 프로젝트는 쿠버네티스의 통합 개발 플랫폼이다.
- 젠킨스와 같은 CI/CD 파이프라인을 제공하기 위한 다양한 도구가 포함돼 있다.