Chapter 10

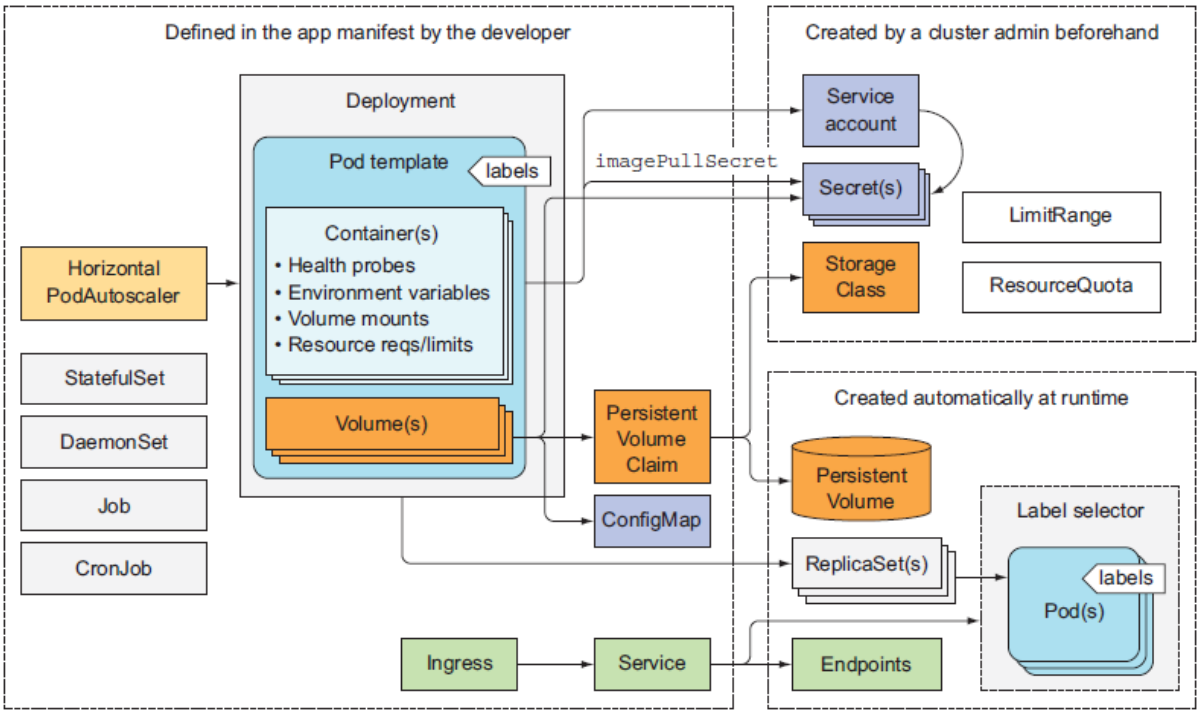
Best practices for developing apps

일반적인 애플리케이션 리소스

- **애플리케이션에서 사용하는 쿠버네티스 구성요소**
  - 개발자에 의한 애플리케이션 매니페스트 정의
  - 클러스터 어드민이 미리 생성
  - 런타임에 자동으로 생성됨

일반적인 애플리케이션 리소스

실제 애플리케이션이 어떻게 구성되는지 살펴보자.



일반적인 애플리케이션 매니페스트에는 하나 이상의 디플로이먼트 또는 스테이트풀셋 객체가 포함되어 있다. 여기에는 컨테이너가 하나 이상 포함된 포드 템플릿이 있고 각각에 liveness probe 가 있고 컨테이너가 제공하는 서비스에 대한 readiness probe 가 있다.

다른 사용자에게 서비스를 제공하는 포드는 하나 이상의 service 를 통해 노출된다. 클러스터 밖에서 연결할 수 있어야할 때 서비스는 LoadBalancer 또는 NodePort 유형 서비스로 구성되거나 ingress 리소스를 통해 노출된다.

Pod 템플릿은 일반적으로 사설(private) 이미지 레지스트리에서 컨테이너 이미지를 가져오는데 사용되는 secret 포드 내부에서 실행하는 프로세스에서 직접 사용되는 secret, 두 가지 유형의 시크릿을 참조한다. 시크릿은 일반적으로 애플리케이션 개발자가 구성하지 않고 운영 팀에서 구성하기 때문에 애플리케이션 매니페스트의 일부가 아니다. 일반적으로 시크릿은 개별 포드에 할당된 ServiceAccount 에 할당된다.

애플리케이션에는 환경 변수를 초기화하는 데 사용되거나 포드에서 configMap 볼륨으로 마운트하는데 쓰이는 하나 이상의 ConfigMap 을 포함하고 있다.

특정 포드는 emptyDir 또는 gitRepo 볼륨과 같은 추가적 볼륨을 사용하지만 영구 저장을 필요로 하는 포드는 persistentVolumeClaim 볼륨을 사용한다. persistentVolumeClaim 은 애플리케이션 메니페스트의 일부이기도 하지만, 시스템 관리자가 참조하는 StorageClass 는 시스템 관리자가 먼저 작성한다.

경우에 따라서는 애플리케이션에서 잡 또는 CronJob 을 사용해야한다. 데몬 셋은 일반적으로 애플리케이션 배포의 일부는 아니지만 일반적으로 시스템의 모든 노드 또는 하위 노드에서 시스템 서비스를 실행하기 위해 생성된다.

HorizontalPodAutoscalers 는 개발자가 메니페스트에 포함하거나 나중에 운영 팀에서 시스템에 추가한다. 또한 클러스터 관리자는 개별 포드 및 모든 포드(전체)의 컴퓨팅 리소스 사용을 제어할 수 있도록 LimitRange 및 ResourceQuota 객체를 생성한다.

애플리케이션이 배포된 후 다양한 쿠버네티스 컨트롤러에 의해 추가 객체가 자동으로 생성된다. 여기에는 Endpoint 컨트롤러로 만든 service Endpoint 객체, deployment 컨트롤러로 만든 replica 및 레플리카셋(Job, cronjob, statfulset, daemonset,)컨트롤러로 만든 실제 포드가 포함된다.

리소스는 종종 조직적으로 관리하기 위해 하나 이상의 Label 을 사용하여 분류한다. 이것은 포드에만 적용되는 것이 아니라 모든 리소스에 적용된다. 라벨 외에도 대부분의 리소스에는 각 리소스를 설명하는 annotation 이 있으며 해당 리소스를 담당하는 사람이나 팀의 연락처나 정보가 나열되거나 관리 및 기타 도구에 대한 추가 메타 데이터가 제공된다.

이 모든 것의 중심에는 Pod 가 있으며, 가장 중요한 쿠버네티스 리소스라고 할 수 있다. 결국 각 애플리케이션은 내부에서 실행된다.

Pod의 라이프사이클

- **애플리케이션을 종료하고 재배치해야한다.**
  - 변경될 로컬 IP 및 호스트 이름 예상하기
  - 디스크에 기록된 데이터가 사라는 경우 예상하기
  - 컨테이너를 다시 시작하더라도 데이터의 보존을 위해 볼륨사용

Pod의 라이프사이클

포드는 단일 애플리케이션만 실행하는 VM 과 비교할 수 있다. 포드 내부에서 실행되는 애플리케이션은 VM 에서 실행중인 애플리케이션과 다르지 않지만 중요한 차이가 있다.

한 가지 예는 쿠버네티스가 어떤 이유로 포드를 다른 노드로 재배치해야 할 필요가 있거나 노드의 스케일다운 요청 등이 있을 때 실행중인 애플리케이션을 언제든지 종료할 수 있다는 것이다.

애플리케이션을 종료하고 재배치해야한다.

쿠버네티스 밖에서는 VM 에서 실행되는 애플리케이션이 한 컴퓨터에서 다른 컴퓨터로 거의 이동하지 않는다. 운영자가 애플리케이션을 이동한다면 애플리케이션을 다시 구성하고 새 위치에서 애플리케이션이 정상적으로 작동하는지 수동으로 확인해야한다. 쿠버네티스를 사용하면 애플리케이션을 훨씬 자주 그리고 자동으로 이동할 수 있다. 작업자가 재구성하지 않아도 이동 후에도 제대로 실행된다. 즉, 애플리케이션 개발자는 애플리케이션이 자주 이동이 가능하도록 상대적으로 애플리케이션을 만들어야하다.

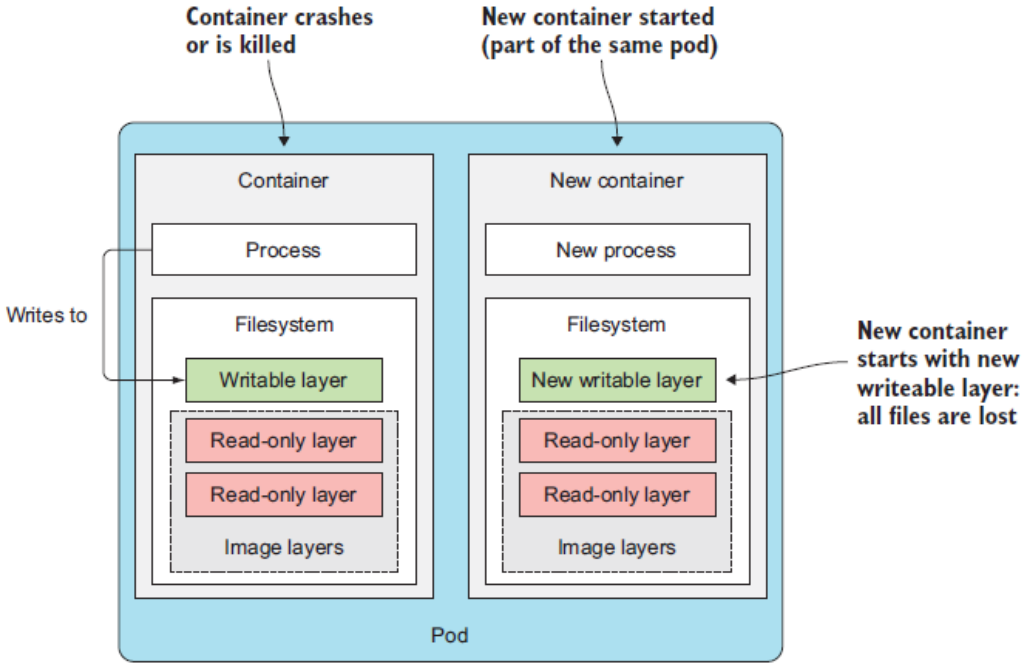
- 변경될 로컬 IP 및 호스트 이름 예상하기

포드가 종료되고 다른 곳으로 옮겨지면 새로운 IP 주소뿐만 아니라 새로운 이름과 호스트 이름도 갖게 된다. 대부분의 상태 비저장 애플리케이션은 보통 부작용 없이 이것을 처리할 수 있지만 상태저장 애플리케이션은 일반적으로 그렇게 할 수 없다. 스테이트풀셋을 통해 상태 저장 애플리케이션을 실행할 수 있다. 애플리케이션이 새 노드에서 시작되어도 이전과 동일한 호스트이름과 여구 상태를 계속 유지하게끔 한다.

그럼에도 불구하고 포드의 IP 는 변경될 것이다. 애플리케이션은 이런 일이 발생하는 경우에 준비돼 있어야한다. 따라서 애플리케이션 개발자는 구성원의 IP 주소에서 클러스터 된 애플리케이션의 구성원을 기반으로 해서는 안 되며 호스트이름을 기반으로 하는 경우 항상 스테이프풀셋을 사용해야한다.

- 디스크에 기록된 데이터가 사라는 경우 예상하기

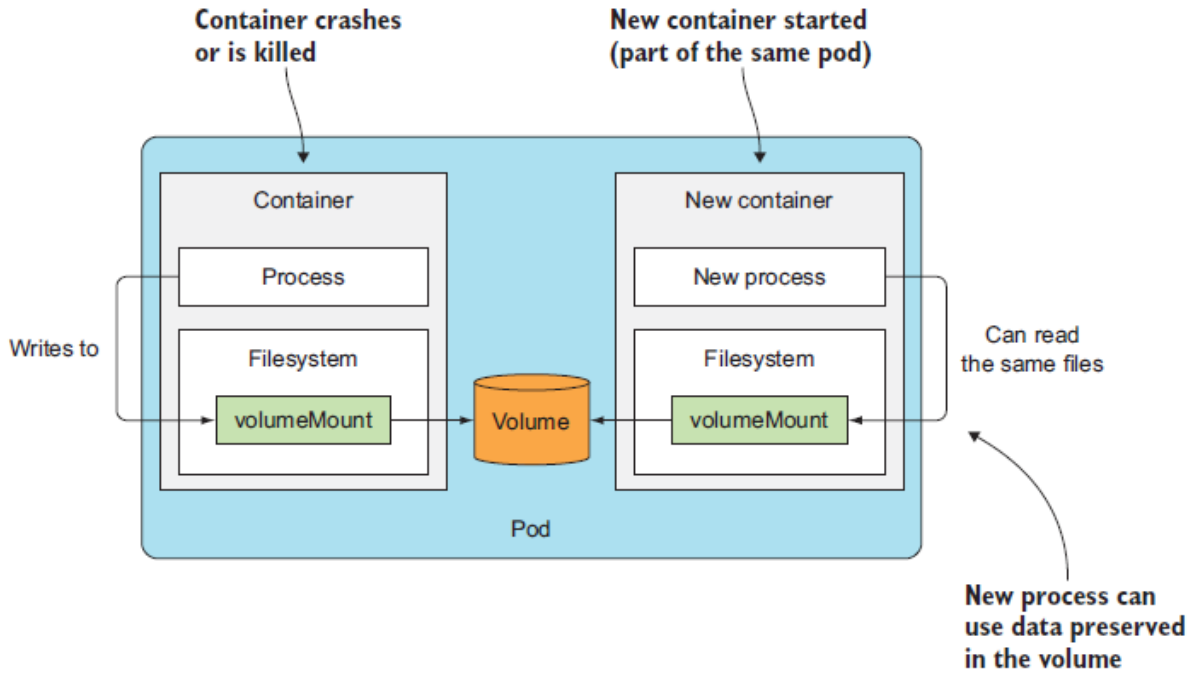
애플리케이션이 데이터를 디스크에 기록할 때 애플리케이션이 작성중인 위치에 영구 저장 장치를 마운트 하지 않았다면 새로운 포드 내부에서 애플리케이션을 시작한 후에 디스크에 저장된 데이터를 사용할 수 없게 된다.



활성 컨테이너가 오류를 반환했거나 노드가 메모리 부족으로 시작돼 프로세스가 OOMKiller 에 의해 종료됐기 때문에 프로세스가 충돌하거나 여러 가지 이유로 개별 컨테이너가 다시 시작될 수 있다는 점을 잊지 말아야한다. 이 경우 포드는 동일하지만 컨테이너 자체는 완전히 새것이다. Kubelet 은 같은 컨테이너를 다시 실행하지 않는다. 항상 새로운 컨테이너를 만든다.

- 컨테이너를 다시 시작하더라도 데이터의 보존을 위해 볼륨사용

컨테이너가 다시 시작해도 데이터가 유실되지 않도록 하려면 적어도 포드 범위의 볼륨을 사용해야한다. 볼륨은 포드가 함께 살고 함께 죽기 때문에 새 컨테이너는 이전 컨테이너가 볼륨에 쓴 데이터를 재사용할 수 있다.



볼륨을 사용해 컨테이너 재시작 시 파일이 보존되는 것이 항상 좋은 것은 아니다. 만약, 데이터가 손상되어 새로 생성된 프로세스에서 다시 크래시가 발생하면? 이때는 크래시가 반복으로 발생하고 포드는 CrashLoopBackOff 상태가 된다. 영구볼륨을 사용하지 않았다면 새 컨테이너는 처음부터 시작되어 크래시가 다시 발생하지 않을 가능성이 크다. 볼륨을 사용하여 컨테이너 재시작시 파일을 보존하는 방법은 양날의 검이다. 사용 여부를 신중하게 생각해야한다.

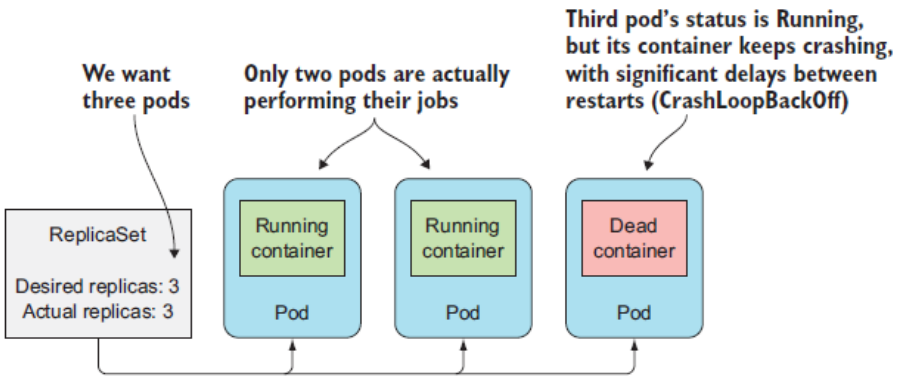
죽은 포드 또는 부분적으로 죽은 포드의 재스케줄링

- **포드의 컨테이너에서 계속 크래시가 생기면**
  - 쿠버네티스는 무한정 다시 시작한다.
  - 재시작 시간은 5분이 될 때까지 증가한다.
  - 재시작하는 컨테이너는 사실상 서비스가 불가능한 상태이다.
  - 레플리카셋은 포드이 수만 count한다.

죽은 포드 또는 부분적으로 죽은 포드의 재스케줄링

포드의 컨테이너에서 계속 crash 가 생기면 Kubelet 은 무한정 다시 시작하게 된다. 재시작 시간은 5 분에 도달할 때가지 기하급수적으로 증가한다. 5 분 간격동안 컨테이너의 프로세스가 실행되고 있지 않기 때문에 포드는 본질적으로 죽었다고 볼 수 있다. 실제로는 다중 컨테이너 포드라면 특정 컨테이너가 정상적으로 작동할 수 있으므로 포드는 일부만 죽은 것이라 볼 수 있다. 그러나 포드에 단일 컨테이너만 들어 있는 경우 프로세스가 더 이상 실행되지 않으므로 사실상 죽은 것이고 쓸모가 없어진다.

레플리카셋 또는 유사한 컨트롤러의 일부인 경우에도 이런 포드가 자동으로 제거되고 재 스케쥴되지 않는다. replicas 수가 3 인 포드를 실행중 하나의 컨테이너에서 크래시가 발생하면 쿠버네티스는 포드를 삭제하고 교체하지 않는다. 포드하나가 계속 CrashBackOff 되어 있으면 결국 2 개의 포드로 서비스를 하고 있는 상태가 된다. 레플리카셋 컨트롤러는 포드가 죽었는지 상관하지 않는다. 관심 있는 모든 것은 포드 수가 원하는 replicas 수와 일치하는 가 이다. crashbackoff 상태이어도 replicas 수는 일치한다. 실습을 통해 알아보자.



원하는 순서로 포드 시작하기

- **포드는 실행되는 순서가 보장되지 않는다.**
- **초기화 컨테이너를 활용**
- **포드 간 의존도를 설정**
  - readiness probe

원하는 순서로 포드 시작하기

포드에서 실행되는 애플리케이션과 수동으로 관리되는 애플리케이션의 다른 점 중 하나는 해당 애플리케이션을 배포하는 운영자가 애플리케이션 간 의존성을 알고 있다는 것이다. 이것은 운영자가 원하는 순서로 애플리케이션을 시작할 수 있다는 의미이다.

포드의 시작방식

쿠버네티스를 이용해 다수의 포드 애플리케이션을 실행할 때 쿠버네티스가 먼저 특정 포트들 실행하고 나머지 포드는 이미 동작 중이며 준비가 완료되었을 때만 알려주는 기본 제공 방법은 없다. 물론 두 번째 메니페스트를 게시하기 전에 첫 번째 포드가 준비될 때까지 기다릴 수는 있지만 전체 시스템은 대개 여러 포드, 서비스 및 다른 오브젝트가 포함된 단일 YAML 또는 jSON 에 정의된다. 쿠버네티스는 나열된 순서대로 YAML/JSON 객체를 처리하지만, 이는 순서대로 etcd 에 기록된다는 것을 의미한다. 포드가 그 순서대로 시작된다는 보장이 없다.

그러나 전제 조건이 충족될 때까지 포드의 주 컨테이너가 시작되는 것을 방지할 수 있다. 이것은 포드에 init container 를 포함시켜서 수행한다.

애플리케이션을 시작하기 전에 의존하고 있는 모든 서비스의 준비를 요구하지 않는 애플리케이션을 작성하는 것이 좋다. 애플리케이션은 의존성이 준비되지 않았을 가능성을 내부적으로 처리할 수 있어야한다. readiness probe 를 통해 준비가 완료되었는지 확인하고, 준비가 되지 않은 경우 서비스 엔드 포인트를 추가하지 않거나, 롤링업데이트 컨트롤러에서 앱 준비 상태로 잘못된 버전의 롤아웃이 방지해야한다.

init container

- **앱 컨테이너들이 실행되기 전에 실행되는 특수한 컨테이너**
- **항상 완료를 목표로 실행**
- **다음 초기화 컨테이너가 시작되기 전에 성공적으로 완료되어야 함**
  - 여러 개의 컨테이너가 있을 경우 하나가 실행이 완료된 후 다음 컨테이너 실행
  - yaml 파일에 명시된 순서로 실행됨

init container

초기화 컨테이너는 앱 컨테이너들이 실행되기 전에 실행되는 특수한 컨테이너이며, 앱 이미지에는 없는 유틸리티 또는 설정 스크립트 등을 포함할 수 있다.

파드는 앱들을 실행하는 다수의 컨테이너를 포함할 수 있다. 또한, 파드는 앱 컨테이너 실행 전에 동작되는 하나 이상의 초기화 컨테이너도 포함할 수 있다.

다음의 경우를 제외하면, 초기화 컨테이너는 일반적인 컨테이너와 매우 유사하다.

- 초기화 컨테이너는 항상 완료를 목표로 실행된다.
- 각 초기화 컨테이너는 다음 초기화 컨테이너가 시작되기 전에 성공적으로 완료되어야 한다.

초기화 컨테이너가 여러 개 있는 경우에는 하나가 실행이 완료된 후에 다음 컨테이너가 실행되는 방식으로 순서대로 실행된다. 초기화 컨테이너가 실행되다가 실패하면, 성공할 때까지 재실행을 하게 된다.

초기화 컨테이너는 앱 컨테이너와 대부분 비슷하게 동작하지만 몇 가지 다른 점이 있다. 초기화 컨테이너는 포드가 모두 준비되기 전에 실행된 후 종료하는 컨테이너이기 때문에 readinessProbe 가 없다. 포드 명세에 초기화 컨테이너를 명시하면 된다. 그렇게 초기화 컨테이너가 모두 실행되고 나면 앱 컨테이너가 시작된다.

Lifecycle hook

- **컨테이너가 시작될 때 실행하고 중지되기 전에 실행**
- **post-start hook**
  - 주컨테이너가 시작한 다음 바로 실행
  - 애플리케이션 시작 시 추가 조작을 수행하기 위해 사용
- **pre-stoop hook**
  - 주 컨테이너가 종료되기 직전에 실행한다.
  - 애플리케이션 종료 직전에 임의의 작업을 수행하는데 사용

Lifecycle hook

앞서 살펴본 init 컨테이너를 이용해 주 컨테이너 실행 전에 특정 작업을 수행하듯이 lifecycle 를 이용하면 주 컨테이너 시작 시점과 종료 전에 원하는 작업을 수행할 수 있다.

컨테이너는 훅의 핸들러를 구현하고 등록함으로써 해당 훅에 접근할 수 있다. 구현될 수 있는 컨테이너의 훅 핸들러에는 두 가지 유형이 있다.

- Exec - 컨테이너의 cgroups와 네임스페이스 안에서, pre-stop.sh와 같은, 특정 커맨드를 실행. 커맨드에 의해 소비된 리소스는 해당 컨테이너에 대해 계산된다.
- HTTP - 컨테이너의 특정 엔드포인트에 대해서 HTTP 요청을 실행

post-start lifecycle hook

post-start hook 은 주 프로세스가 시작하고 난 다음 바로 실행된다. 애플리케이션이 시작될 때 추가 조작을 수행하기 위해 이를 사용한다. 이를 통해 애플리케이션이 시작되고 있는 외부 리스너에게 시그널을 보내거나 애플리케이션을 초기화 하는 작업을 시작할 수 있다. 단, hook 이 주 컨테이너와 병렬로 실행됨을 주의해야한다. 주 컨테이너가 완전히 시작되기 전에 hook 이 실행되기 때문에 오해할 수 있다.

hook 이 완료될 때까지 주 컨테이너는 ContainingCreating 인 채로 Waiting 상태로 유지된다. 이 때문에 포드의 상태는 Running 대신 Pending 상태가 된다. 훅이 실행되지 않거나 0 이 아닌 종료 코드를 반환하면 주 컨테이너가 종료된다.

pre-stop lifecycle hook

pre-stop hook 은 주 컨테이너가 종료되기 직전에 즉시 실행된다. 컨테이너가 종료될 kubelet 은 pre-stop hook 을 실행한 다음 컨테이너 프로세스로 SIGTERM 을 전송한다. 때문에 컨테이너 프로세스가 정상으로 종료하지 않는 경우 pre-stop hook 을 사용해 컨테이너를 정상적으로 종료할 수 있다. 또한 컨테이너가 멈추기 전 상태를 저장하는 의미 있는 hook 을 운영할 수도 있다.

사용자는 hook 핸들러를 가능한 한 가볍게 만들어야 한다.

다음은 Lifecycle hook 을 사용한 메니페스트이다.

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh", "-c", "echo Hello from the postStart handler > /usr/share/message"]
      preStop:
        exec:
          command: ["/bin/sh","-c","nginx -s quit; while killall -0 nginx; do sleep 1; done"]
```

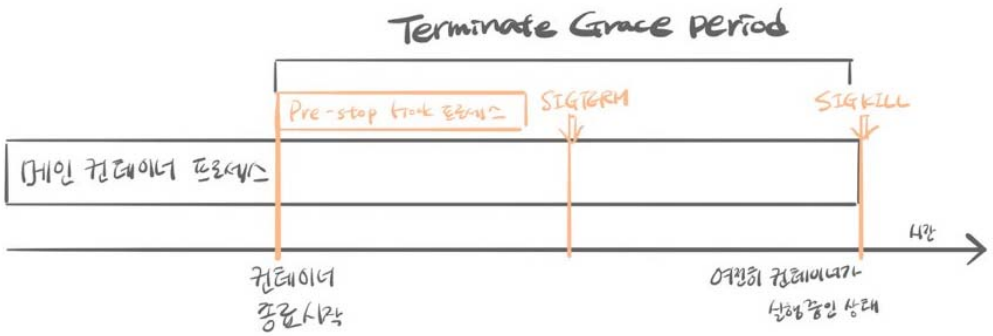
Pod Shutdown 과정

- **API의 요청에 따라 kubelet이 포드의 컨테이너를 삭제한다.**
  - pre-stop hook 이 구성된 경우 이를 실행하고 완료될 때 까지 기다림
  - SIGSTERM 신호를 컨테이너의 메인 프로세스로 보낸다.
  - 컨테이너가 완전히 셧다운 될 때까지 또는 종료 유예 기간이 만료될 때까지 기다린다.
  - 정상적으로 종료도지 않은 경우 SIGKILL 로 프로세스를 강제 종료한다.
- **잘 종료하는 애플리케이션**
  - 포드에서 실행중인 애플리케이션을 안정적으로 종료

Pod Shutdown

포드의 종료는 API 서버의 포드 객체 삭제에 의해 트리거된다. API 서버는 kubelet 에게 포드 삭제를 전달하고, kubelet 컨테이너를 종료하기 시작한다. 각 컨테이너에 정상적으로 종료할 시간이 주어지지만 시간이 제한된다. 그 시간을 종료 유예 기간(termination grace period)라고 하고 포드당 구성할 수 있다. 종료 프로세스가 시작 되면 타이머가 시작되고 다음 아래와 같은 종료 작업이 진행된다.

1. pre-stop hook 이 구성된 경우 이를 실행하고 완료될 때 까지 기다린다.
2. SIGSTERM 신호를 컨테이너의 메인 프로세스로 보낸다.
3. 컨테이너가 완전히 셧다운 될 때까지 또는 종료 유예 기간이 만료될 때까지 기다린다.
4. 정상적으로 종료도지 않은 경우 SIGKILL 로 프로세스를 강제 종료한다.



종료유예기간(TerminateGracePeriod)은 컨테이너가 강제로 종료되기 전에 정상 종료할 수 있도록 지원하는 시간으로 기본값은 30 이다. 이는 포드 스펙에서 terminationGracePeriodSecond 필드에서 설정할 수 있다.

```
# kubectl delete pod lifecycle-demo --grace-period=1
```

종료유예기간(TerminateGracePeriod) 설정 시 강제로 즉시로 삭제하고자 할 때는 다음과 같이 —force 옵션을 추가로 할당한다. 단, 이 옵션을 적용 시 statefulset 리소스에는 적용하지 않도록 한다. 포드를 강제로 삭제하는 과정에서 컨테이너가 gracefull shutdown 을 하지 못하면 상태 저장 클러스터가 오작동할 수 있다. 이는 PersistentVolume 에 문제를 일으킬 수 있으니 주의해야 한다. 포드가 더 이상 실행하지 않거나, 클러스터의 다른 포드와 통신하지 않는다는 확신을 가질 때 statefulset 포드를 삭제한다.

잘 종료되는 애플리케이션

포드에서 동작하고 있는 애플리케이션을 말끔히 종료되는데 고정된 시간이 필요하다. 이 시간을 통해 분산 데이터 스토리지에 데이터를 전송하거나 서비스를 안전하게 종료한다. 애플리케이션이 정상적으로 잘 종료되는 것을 보장해주는 방법을 생각해보자.

한 가지 솔루션은 종료 신호를 받으면 애플리케이션을 사용해 삭제된 포드의 데이터를 나머지 포드로 마이그레이션 하는 작업을 수행하는 새로운 포드를 위한 잡(cron job)리소스를 만들어 운영할 수 있다. 이 때 pre-stop hook 을 이용할 수 있다.

모든 클라이언트 요청이 올바르게 처리되도록 보장하기

- **포드가 시작할 때 클라이언트 연결 끊어짐 방지**
  - readiness probe를 적용
- **포드 셧다운 동안 연결 끊어짐 방지**
  - 애플리케이션이 몇 초 기다린 후에 종료
  - pre-stop hook

모든 클라이언트 요청이 올바르게 처리되도록 보장하기

포드의 클라이언트 관점에서 포드의 라이프사이클을 살펴본다. 포드의 스케일업과 스케일다운 할 때 클라이언트가 문제를 일으키지 않게 하려면 이 부분을 정확히 아는 것이 중요하다.

• 포드가 시작할 때 클라이언트 연결 끊어짐 방지

포드 spec 에 readiness probe 를 설정하지 않으면 포드는 항상 준비 상태로 간주된다. 이 때 애플리케이션이 연결을 수락하지 않으면 클라이언트는 "connection refused" 오류를 만나게 된다. 해결방법은 readiness probe 를 이용해 애플리케이션이 요청을 제대로 받을 준비가 되었을 때 Endpoint 를 전달하여 연결하도록 하면 된다.

• 포드 셧다운 동안 연결 끊어짐 방지

포드 삭제 요청을 API 서버가 수신하면, 먼저 etcd 에서 상태를 수정한 다음 해당 삭제 작업을 endpoint 와 kubelet 에게 병렬로 전달된다. kubelet 이 Pod 를 제거하는 과정에서 kube proxy 가 연결을 해제하는 시간보다 애플리케이션이 SIGTERM 을 먼저 받는다. 이로 인해 클라이언트 사용자는 커넥션이 연결되고 "connection refused" 응답을 받게된다. 해결방법은 kubeproxy 가 연결을 끊을 때까지 애플리케이션이 몇 초 기다린 후에 종료되게 하면 된다. 이대 pre-stop hook 을 사용할 수 있다.

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

애플리케이션 쉽게 실행하고 관리하기

- **관리 가능한 컨테이너 이미지 만들기**
- **이미지에 적절히 태그하고 imagePullPolicy를 현명하게 사용**
- **단일 차원 라벨 대신 다차원 라벨 사용**
- **주석을 통해 각 리소스 설명하기**
- **프로세스가 종료된 원인 제공**
- **애플리케이션 로그 핸들링**

애플리케이션 쉽게 실행하고 관리하기

쿠버네티스에서 더 쉽게 관리할 수 있는 애플리케이션 제작하는 방법을 알아본다.

- 관리 가능한 컨테이너 이미지 만들기
  컨테이너 이미지를 제작해서 작은 이미지로, 꼭필요한 바이너리만 넣어 만든다.

- 이미지에 적절히 태그하고 imagePullPolicy 를 현명하게 사용
  latest 이미지가 아닌 적절한 버전 지정자를 포함하는 태그를 사용하며,
  이미지 풀 정책을 Always 로 설정하여 새 포드를 배포할 때마다 컨테이너 런타임이 이미지 레지스트리에 연결되도록 한다.

- 단일 차원 라벨 대신 다차원 라벨 사용
  모든 리소스에 여러 개의 라벨을 붙여 다차원으로 관리한다.

- annotation 을 통해 각 리소스 설명하기
  리소스에 정보를 추가 할 때는 annotation 을 사용하여 리소스의 설명이나
  주석 책임자의 연락처나 정보를 포함시킨다.

- 프로세스가 종료된 원인 제공
  컨테이너가 왜 종료되었는지를 알기위해 로그파일에 모든 필요한 정보를 포함하고,
  컨테이너 파일시스템에 있는 특정 파일에 종료 메시지를 기록하게 한다.
  terminationMessagePath: /var/termination-reason

- 애플리케이션 로그 핸들링
  애플리케이션 로그를 표준출력으로 작성하고, kubectl logs 명령을 통해 쉽게 볼 수 있게 한다.