4. 레플리케이션과 그 밖의 컨트롤러: 관리되는 파드 배포

210320 김보배 (tree9295@gmail.com)

실환경에서 배포한 애플리케이션이 자동으로 수행되고 수동적인 개입 없이도 안정적인 상태로 유지되길 원할 것

- 레플리케이션컨트롤러 또는 디플로이먼트와 같은 유형의 리소스를 생성해 실제 파드를 생성하고 관리

4.1 파드를 안정적으로 유지하기

쿠버네티스를 사용하는 이점 → 쿠버네티스에 컨테이너 목록을 제공하면 해당 컨테이너를 클러스터 어딘가에서 계속 실행되도록 할 수 있다는 것.

파드가 노드에 스케줄링되는 즉시 해당 노드의 kubelet은 파드의 컨테이너를 실행하고 파드가 존재하는 한 컨테이너는 계속 실행될 것.

- 컨테이너의 주 프로세스에 crash가 일어나면 kubelet이 컨테이너를 다시 시작한다 → 자동 치유
- 하지만 모든 오류에 해당하는 것은 아니다. 예로 Java의 OutofMemoryErrors가 발생하면 JVM 프로세스 자체는 계속 실행된다. 하지만 애플리케이션은 정상적으로 동작하지는 않는다.

그럼 어떻게?

- 더 이상 제대로 동작하지 않는다는 신호를 쿠버네티스에 보내, 쿠버네티스가 관리하도록 하진 못할까?
- 애플리케이션 내부의 기능에 의존하지말고 외부에서 애플리케이션의 상태를 체크하진 못할까?

라이브니스 프로브 (Liveness probe)

쿠버네티스는 liveness probe를 통해 컨테이너가 살아 있는지 확인할 수 있다.

- 쿠버네티스는 주기적으로 프로브를 실행하고, 프로브가 실패할 경우 컨테이너를 다시 시작한다.

프로브 실행에는 3가지 메커니즘이 존재한다.

- HTTP GET probe: 지정한 IP주소, 포트, 경로에 HTTP GET 요청을 수행
  - 프로브가 응답을 수신하고, 응답 코드가 오류를 나타내지 않는 경우, 프로브가 성공했다고 간주
- TCP Socket probe: 컨테이너의 지정된 포트에 TCP 연결을 시도
  - 연결에 성공하면 프로브 성공 간주
- Exec probe: 컨테이너 내의 임의의 명령을 실행 → 명령의 종료 상태 코드를 확인
  - 상태코드가 0이면 프로브 성공 간주

HTTP 기반 라이브니스 프로브 생성

라이브니스 프로브 정의한 파드 ymal 디스크럽터

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-liveness
spec:
  containers:
  - image: luksa/kubia-unhealthy
```

```
    name: kubia
    livenessProbe:  # liveness probe 정의
      httpGet:
        path: /
        port: 8080
```

"/" 경로의 8080 포트에 HTTP GET을 요청해 컨테이너가 정상동작하는지 확인하도록 정의한 것

동작 중인 라이브니스 프로브 확인

```
kubectl get po kubia-liveness # running 확인

kubectl describe po kubia-liveness # 만약 오류가 생겼다면 여기서 에러코드와 이벤트 이유를 볼 수 있다.

kubectl logs pod_name --prevous # 크래시된 컨테이너 애플리케이션 로그 보기
```

라이브니스 프로브의 추가 속성 설정

명시적으로 지정한 라이브니스 프로브 옵션 외에도 지연(delay), 제한시간(timeout), 기간(period) 등과 같은 속성도 존재

describe에서 볼 수 있는 데이터

```
Liveness: http-get http://:8080/ delay=0s timeout=1s period=10s #success=1 #failure=3
```

- delay=0s
  - 컨테이너 시작 후 바로 liveness probe 작동
  - 컨테이너 시작 후 바로 프로브를 작동시키면, 대부분의 애플리케이션이 요청을 받을 준비가 되어있지 않아 프로브가 실패한다. 그래서, 초기지연을 어느정도 설정하는 것이 좋다.
  - yaml에서 `initialDelaySeconds`로 설정이 가능함.
- timeout=1s
  - 프로브 보냈으면 1초안에 응답해야함
- period=10s
  - 10초마다 프로브 작동
- #failure=3
  - 프로브가 연속 3번 실패하면 컨테이너 다시시작

효과적인 라이브니스 프로브 생성

운영환경에서의 파드는 반드시 라이브니스 프로브를 정의해야 한다. 정의하지 않으면 쿠버네티스가 애플리케이션이 살아 있는지를 알 수 있는 방법이 없다. 프로세스가 실행되는 한 쿠버네티스는 컨테이너가 정상적이라고 간주한다.

위의 예제는 단순히 서버가 응답하는지 확인했다. 이걸로도 문제가 있음을 확인하고, 컨테이너를 재실행 할 수 있다.

- 라이브니스 프로브를 위해 특정 URL 경로에 요청하도록 프로브를 구성해 애플리케이션 내에서 실행 중인 모든 주요 구성 요소가 살아 있는지 또는 응답이 없는지 확인하도록 구성할 수 있다.
- Spring Boot actuator 같이 생각하면 융합하거나 하면 모듈별로도 측정이 가능하지 않을까?

라이브니스 프로브 요약

크래시 등으로 인해 라이브니스 프로브가 실패한 경우, 쿠버네티스가 컨테이너를 재시작해 컨테이너가 계속 실행하도록 한다는 것

이 작업은 파드를 호스팅하는 노드의 kubelet에서 수행한다. 마스터 노드에서 작동하는 쿠버네티스 컨트롤 플레인 구성 요소는 이 프로세스에 관여하지 않는다.

- 노드 자체에 크래시가 나서 문제가 생기면 내부 모든 파드가 문제가 생기게 되는데, 이런 경우엔 노드와 파드를 다시 생성해야하는 것은 컨트롤 플레인의 몫이다.
- 하지만 파드자체의 크래시는 kubelet에서만 관리된다.

애플리케이션이 다른 노드에서 다시 시작되도록 하려면 어떻게 해야할까?

- 레플리케이션 컨트롤러!

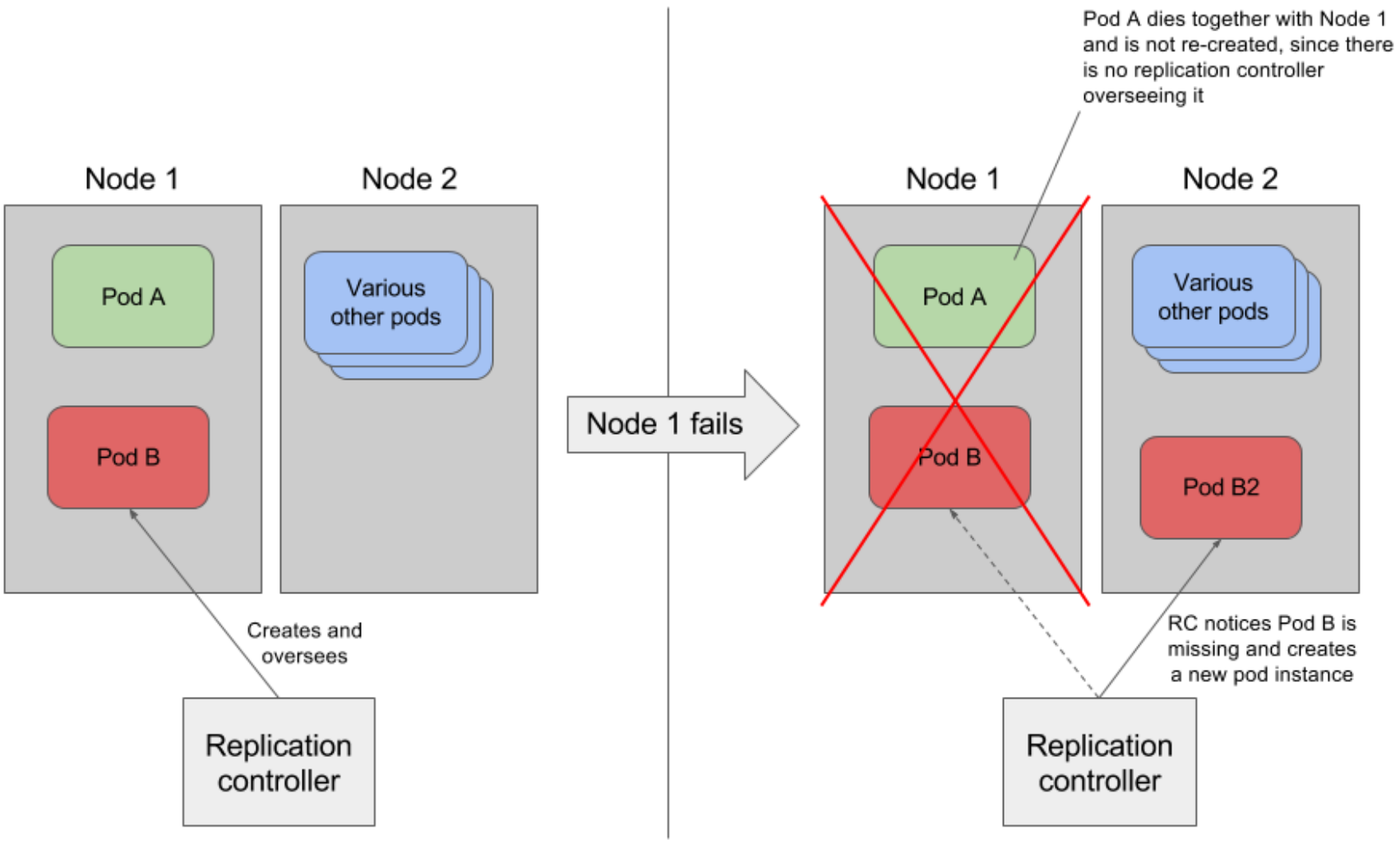
4.2 레플리케이션 컨트롤러

쿠버네티스 리소스로서 파드가 항상 실행하도록 보장한다.

어떤 이유에서든 파드가 사라진다고 하자.

- 클러스터에서 노드가 사라지거나
- 노드에서 파드가 제거된 경우

두 가지 경우를 들 수 있다. 그럼, 레플리케이션 컨트롤러는 사라진 파드를 감지해 교체 파드를 생성한다.



위 그림으로 레플리케이션 컨트롤러의 역할을 확인할 수 있다.

- 파드 A는 관리가 되지 않아 유실이 되면 **그대로 없어진다.**
- 파드 B는 **레플리케이션 컨트롤러에 의해 관리**되기 때문에 노드의 문제로 유실되어도 **다른 노드에 새로 파드를 생성**하게 된다.

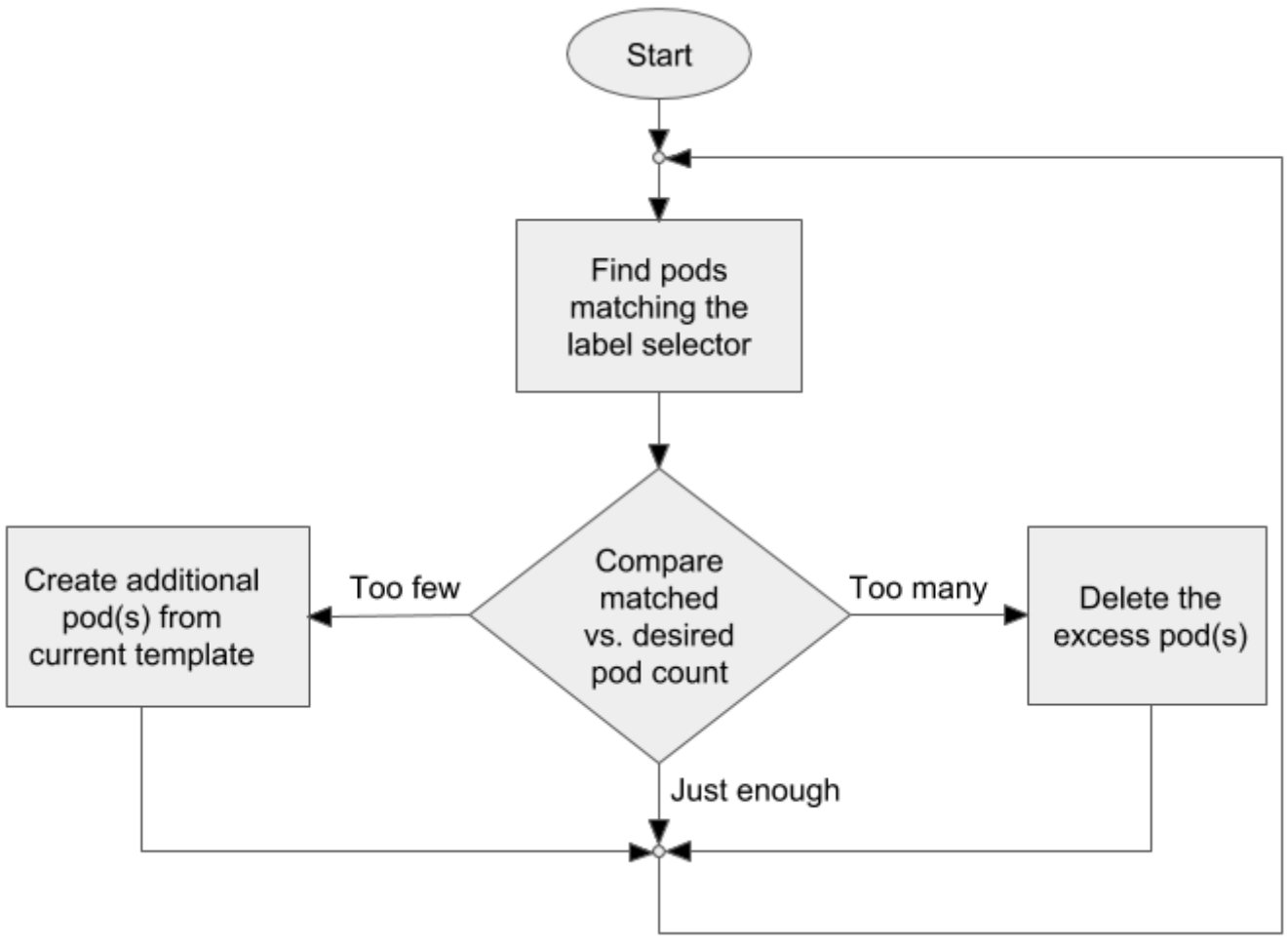
레플리케이션의 동작

실행 중인 파드 목록을 지속적으로 모니터링

- 특정 유형(레이블 셀렉터)의 실제 파드 수가 의도하는 수와 일치하는지 항상 확인

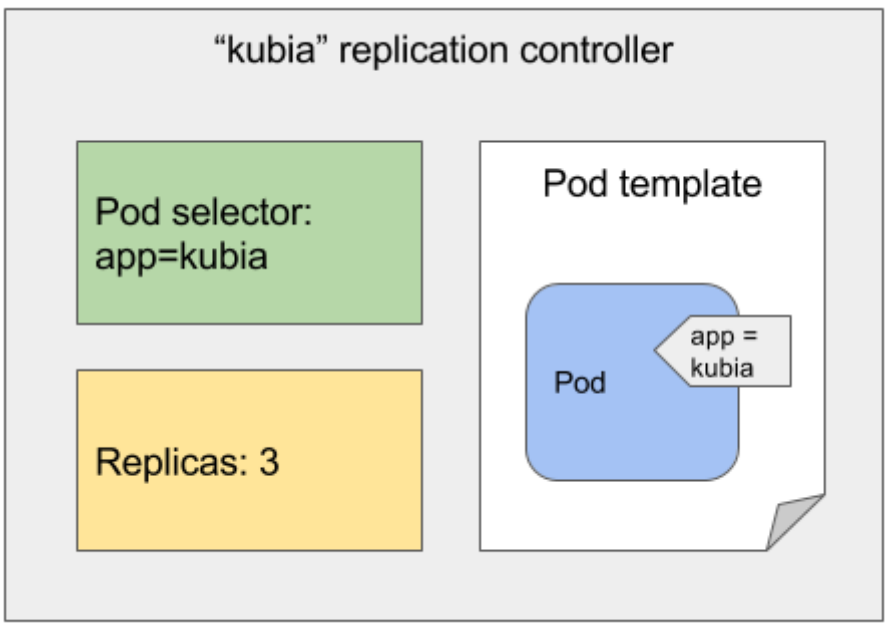
만약 파드의 수가 의도하는 수보다 작으면 새로운 복제본(파드)을 만들고, 의도하는 수보다 많으면 초과 복제본을 제거한다.

- 엥? 의도하는 수보다 많을 수가 있어?
    1. 누군가 같은 유형의 파드를 수동으로 만든 경우
    2. 누군가 기존 파드의 유형(레이블)을 변경한 경우
    3. 누군가 의도하는 파드 수를 줄인 경우



레플리케이션 컨트롤러의 알고리즘을 간단히 나타내면 위 사진과 같다.

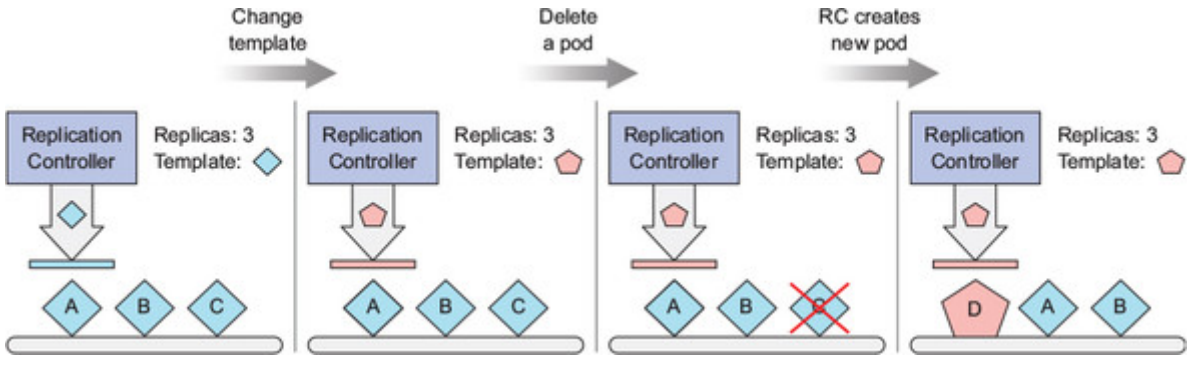
- 레이블 셀렉터와 매치되는 파드를 찾는다.
- 매치되는 파드 수와 의도하는 파드 수를 비교한다.
    - 너무 많다? 삭제
    - 너무 적다? 생성



레플리케이션 컨트롤러는 대표적으로 3가지 요소가 있다.

- 레이블 셀렉터: 레플리케이션 컨트롤러의 범위(디스크럽터에 명시된)에 있는 파드 결정

- 레플리카 수: 실행할 파드의 의도하는 수 지정
- 파드 템플릿: 새로운 파드 레플리카 생성시 사용 → 파드 템플릿 형태로 파드를 생성



책에서 중복되게 설명하는 것이 이러한 '레이블 셀렉터'와 '파드 템플릿'을 변경해도 기존 파드에는 영향이 가지 않는다는 것.

- 즉, 파드 셀렉터의 내용을 A 이미지가 아닌 B 이미지로 생성하도록 수정했다. 그렇다고해서 기존에 만들어진 해당 레플리케이션 컨트롤러에서 관리하는 파드들이 파드 템플릿에 맞게 수정되는 것이 아니다. 그냥 그대로 있고, 이후에 새로 생성되는 파드들만 해당 파드 템플릿에 맞게 생성된다는 것이다. 레이블 셀렉터도 마찬가지로 이후 생성하는 것만 새로 정의된 것에 맞게 작동한다는 것.

레플리케이션 컨트롤러 사용 시 이점

- 기존 파드가 사라지면 새 파드를 시작해 파드가 항상 실행되도록 한다.
- 클러스터 노드에 장애가 발생하면 장애가 발생한 노드에서 실행 중인 모드 파드에 관한 교체 복제본이 생성된다. (레플리케이션 컨토를러 제어 하에 있는 파드들만)
- 수동 또는 자동으로 파드를 쉽게 수평으로 확장할 수 있게 한다.

레플리케이션 컨트롤러 생성

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: kubia        # rc name
spec:
  replicas: 3        # replica 개수
  selector:          # 지정한 레이블 셀렉터: app-kubia인 파드만 선택
    app: kubia
  template:          # pod template: 새로 생성시 이에 해당하는 파드가 생성됨
    metadata:
      labels:
        app: kubia
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
        ports:
        - containerPort: 8080
```

쿠버네티스는 레이블 셀렉터 app=kubia와 일치하는 파드 인스턴스가 세 개를 유지하도록 하는 kubia라는 이름의 새로운 레플케이션 컨트롤러를 생성

레플리케이션 컨트롤러 작동

잘 작동하고 있는지 확인해보자.

```
kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
kubia-c9bc9   1/1     Running   0          70s
kubia-n47bq   1/1     Running   0          70s
kubia-z9rmp   1/1     Running   0          70s
```

그럼 우리가 의도한대로도 잘 작동할까? 이 중 하나의 파드를 삭제해보자.

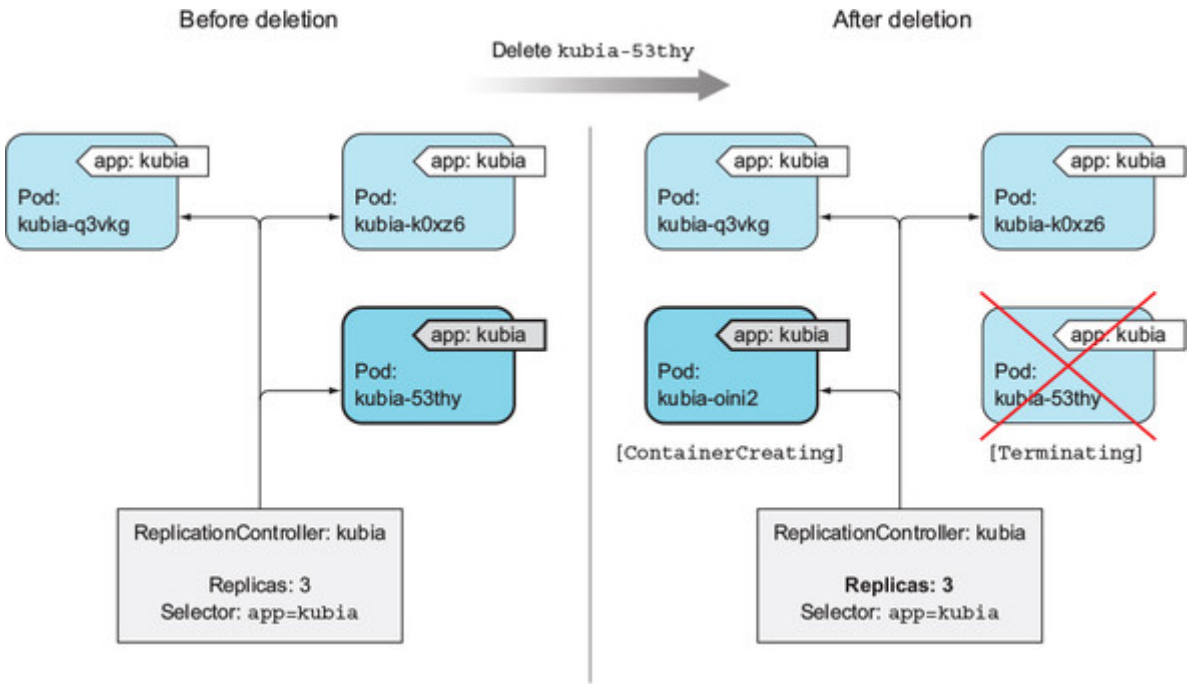
```
> kubectl delete pod kubia-c9bc9
pod "kubia-c9bc9" deleted
```

파드의 삭제에 따라 레플리케이션 컨트롤러는 의도한 수만큼의 파드보다 적게 운용하게 되었다. 그럼 이를 바로 알아채고, 의도한 수에 맞게 맞추려 하나의 파드를 생성한다.

```
> kubectl get pods
NAME          READY   STATUS        RESTARTS   AGE
kubia-c9bc9   1/1     Terminating   0          102s
kubia-f6gxq   1/1     Running       0          6s
kubia-n47bq   1/1     Running       0          102s
kubia-z9rmp   1/1     Running       0          102s
```

- c9bc9 파드 삭제에 따라 새로운 파드인 f6gxq가 생성된 것을 볼 수 있다.

이러한 상황은 파드가 단순히 종료되는 것 이외에도 네트워크가 제대로 동작하지 않을 때 등 파드에 이상이 감지되면 자동으로 새로운 파드로 대체해 작동하게 된다.



위 상황을 그림으로 나타내보면 위 그림처럼 나타낼 수 있다.

레플리케이션 컨트롤러 정보 얻기

`kubectl get rc` : 의도한 파드 수, 실제 파드 수, 준비된 파드 수를 표시해준다. 중요하면서도 간략한 정보만을 제공

`kubectl describe rc replicationControllerName` : 더 상세한 정보들을 볼 수 있다.

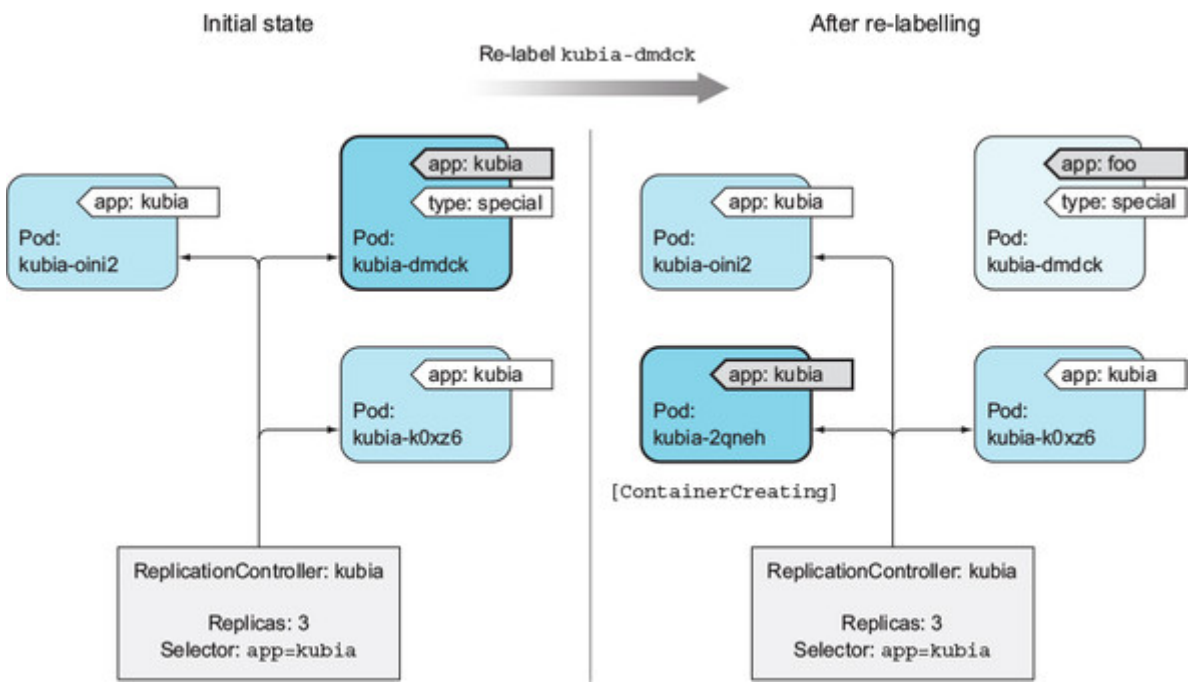
- 실제 의도하는 파드 인스턴스 수
- 파드의 상태별 파드 인스턴스 수
- 레플리케이션 컨트롤러 관련 이벤트

관리하는 파드 레이블 변경

레플리케이션 컨트롤러가 관리하는 파드의 레이블을 변경하면 어떻게 될까?

- 레플리케이션 컨트롤러의 레이블 셀렉터와 일치하지 않게되며, 해당 파드는 수동으로 만든 파드가 된다. (또는 다른 레플리케이션 컨트롤러에서 관리하게 되는 파드가 될 수 있다)

- 파드의 레이블을 변경해 파드가 하나 사라진 것을 레플리케이션 컨트롤러가 감지하고 사라진 파드를 대체하기 위해 새로운 파드를 가동하게 된다.
- `kubectl label pod pod-name app=foo --overwrite` : app 레이블의 값을 foo로 수정한다.



파드의 레이블을 변경할 사례가 있을까?

- 레플리케이션 컨트롤러가 관리하는 특정 파드에 어떠한 버그가 있다고 하자. 원인 파악 및 디버깅하기 위해서 해당 파드에 실험하는 것이 필요하다면, 레플리케이션 컨트롤러에서 관리하지 않도록 단순히 레이블만 수정하면 된다. 그러면 파드는 단순히 수동으로 만든 파드가 될 것이고, 어차피 레플리케이션 컨트롤러가 의도한 수대로 다시 하나의 파드를 생성해 서비스할 것이기 때문에 문제가 발생하지 않을 것이다.

만약 파드의 기존 레이블을 변경하는 것이 아니라 새로운 레이블을 추가하게 되면 어떻게 될까?

- 레플리케이션 컨트롤러는 추가 레이블에 대해서는 상관하지 않기 때문에 별도의 레이블을 추가하는 것은 아무런 효과가 없다. 단지 레이블 셀렉터에서 참조하는 모든 레이블을 갖고 있는지만 고려한다.

레플리케이션 컨트롤러 레이블 셀렉터 및 파드 템플릿 변경

디스크럽터에셔 명시했던 레플리케이션의 레이블 셀렉터를 변경하면 어떻게 될까?

- 모든 파드가 레플리케이션 컨트롤러의 범위를 벗어나게 되기 때문에 3개의 새로운 파드를 생성하게 될 것이다.

근데 보통 이렇게 할 일은 없다. 대신 파드 템플릿은 변경하는 경우가 있다.

- 새롭게 생길 파드들에 대해 새로운 특징을 부가하는 등 기존 생성하던 파드들과 정보가 달라져야한다면 파드 템플릿을 수정하면 된다.
  - `kubectl edit rc rc-name` : yaml 디스크럽터를 열어 수정할 수 있다.
- 앞서 말했듯, 기존 파드들엔 전혀 영향이 가지 않는다. 단순히 새롭게 생길 파드들에서만 변경된 파드 템플릿을 참조해 파드를 생성하게 될 뿐이다.

수평 파드 스케일링

레플리케이션 컨트롤러를 사용하면 파드의 복제본 수를 유지하는 것이 매우 쉽다는 것을 알 수 있었다.

- 그 말은즉슨, 스케일링 하기도 매우 쉽다는 것을 의미한다.
- 단순히 replicas 필드 값만 변경하면 된다.

변경하는 2가지 방법 (replica 수를 10개로 수정하고 싶다)

- `kubectl scale rc rc-name --replicas=10`
- `kubectl edit rc rc-name` : 직접 yaml 디스크럽터 들어가서 수정

다시 스케일 다운하고 싶다면, 위 방식으로 replicas 수를 줄이면 된다.

레플리케이션 컨트롤러 삭제

`kubectl delete rc rc-name` 를 통해 레플리케이션 컨트롤러를 삭제할 수 있다.

- 레플리케이션 컨트롤러를 삭제하면, 관리하는 파드들이 다같이 함께 삭제되게 된다.
- 만약 오로지 레플리케이션 컨트롤러만을 삭제하고 싶다면 (파드는 유지하고 싶다면), `--cascade=false` 조건을 사용하자.
  - `kubectl delete rc rc-name --cascade=false`
  - 그러면 관리하던 파드들은 어디에도 속해있지 않게 된다. 하지만 또 다시 관리하고 싶다면 적절한 레이블 셀렉터를 작성해 레플리케이션 컨트롤러를 생성하면 된다.

4.3 레플리카 셋(ReplicaSet) 사용하기

차세대 레플리케이션 컨트롤러 → 레플리케이션 컨트롤러를 완전히 대체할 것

레플리케이션 컨트롤러와의 비교

도대체 뭐가 다른데?

- 똑같이 동작하지만 레플리카 셋이 더 풍부한 표현식을 가지는 파드 셀렉터를 사용한다.
  1. 특정 레이블이 없는 경우
     - 특정 레이블이 없는 파드들만 관리할 수 있다. (레플리케이션 컨트롤러에선 불가)
  2. 특정 레이블이 있는 경우
     - 특정 레이블이 있는 파드들만 관리할 수 있다. 예를 들어, app이라는 label을 갖기만 하면 관리하는 파드로 선택할 수 있다. 해당 레이블이 어떤 값을 가지든 상관없다.
  3. OR 조건으로 여러 레이블 조건을 매칭 시키려는 경우
     - 2개 이상의 레이블 조건 중 하나만을 만족하는 파드만을 선택해 관리할 수 있다. 예를 들어, app=foo, app=var 두 개의 조건을 두고 이 중 만족만 하는 파드를 선택해 관리할 수 있는 것.
     - 레플리케이션 컨트롤러에서는 하나의 조건으로만 선택이 가능했다.

레플리카셋 정의하기

```
apiVersion: apps/v1beta2
kind: ReplicaSet               # ReplicaSet 정의
metadata:
  name: kubia
spec:
  replicas: 3                  # replicas 수 정의
  selector:
    matchLabels:               # replication controller와 다르게 matchLabels 하위에서 레이블 셀렉터 정의
      app: kubia
  template:
    metadata:
      labels:
        app: kubia
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
```

`kubectl create -f replicaSet-filename.yaml` : 레플리카 셋 생성

`kubectl get rs` : 상태 확인

`kubectl describe rs` : 상세 정보 확인

풍부한 레이블 셀렉터

앞서 이야기 했듯 레플리카셋은 더욱 풍부한 레이블 셀렉터를 지원한다.

```
# ...

spec:
  replicas: 3
  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
         - kubia

# ...
```

- operator
  - `In`: 레이블 값이 지정된 값 중 하나와 일치해야 한다.
  - `NotIn`: 레이블의 값이 지정된 값과 일치하지 않아야 한다.
  - `Exists`: 파드는 지정된 키를 가진 레이블이 포함돼야 한다. (값 중요X, 값 지정X)
  - `DoesNotExist`: 파드에 지정된 키를 가진 레이블이 포함돼 있지 않아야 한다.

파드와 매칭되기 위해서는 모든 표현식이 true로 평가돼야 한다.

레플리카셋 정리

`kubectl delete rs rs-name`

- 레플리케이션 컨트롤러와 마찬가지로 관리하는 모든 파드들도 함께 삭제된다.
- `--cascade=false` 조건을 사용하면 파드 삭제는 진행하지 않을 수 있다.

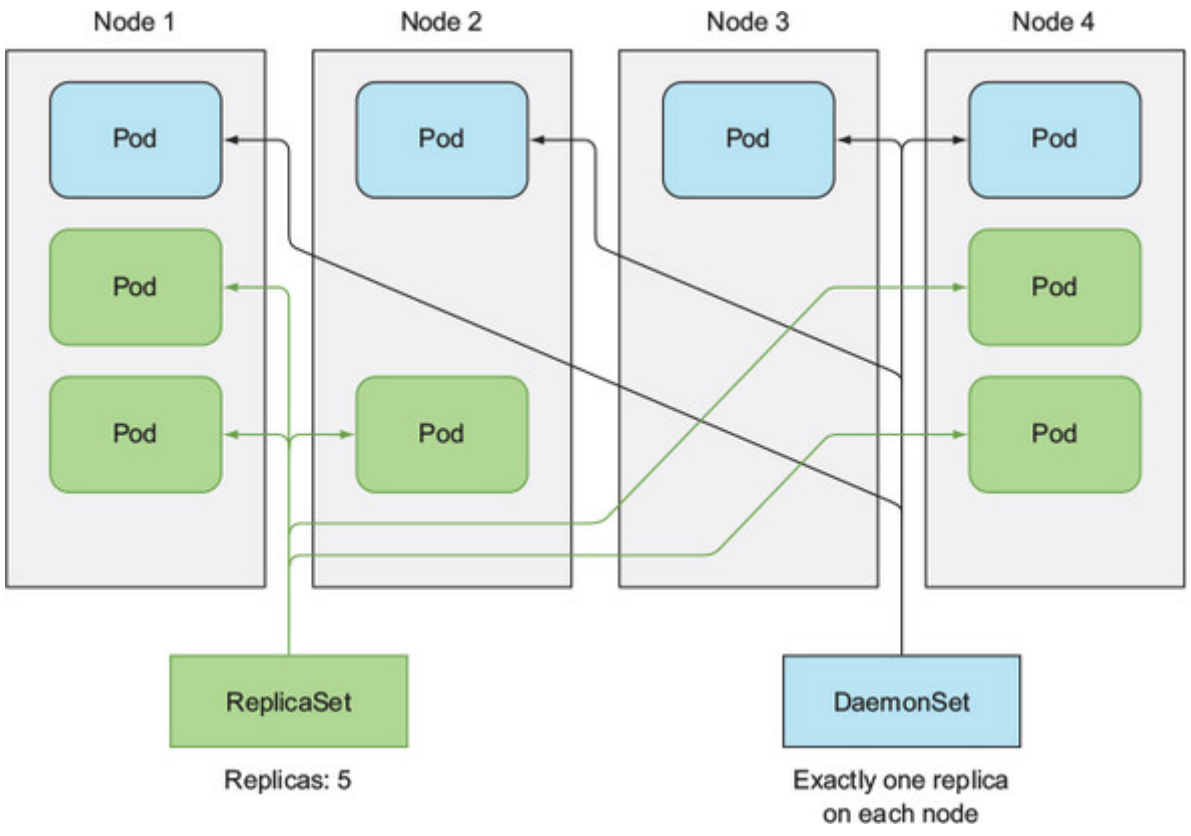
4.4 데몬 셋(DaemonSet): 각 노드에서 정확히 한 개의 파드 실행

레플리케이션 컨트롤러와 레플리카셋은 클러스터 내 어딘가에서 지정된 수 만큼의 파드를 실행하는데 사용된다.

하지만 만약 모든 노드에 있어, 노드당 하나의 파드만 실행되길 원하는 경우가 있을 수 있다. 예를 들어, 모든 노드에서 로그 수집하는 파드가 있어야한다던가, 리소스 모니터를 실행해야 하는 경우가 있을 수 있다.

이런 경우에 데몬 셋을 사용한다.

데몬 셋 작동



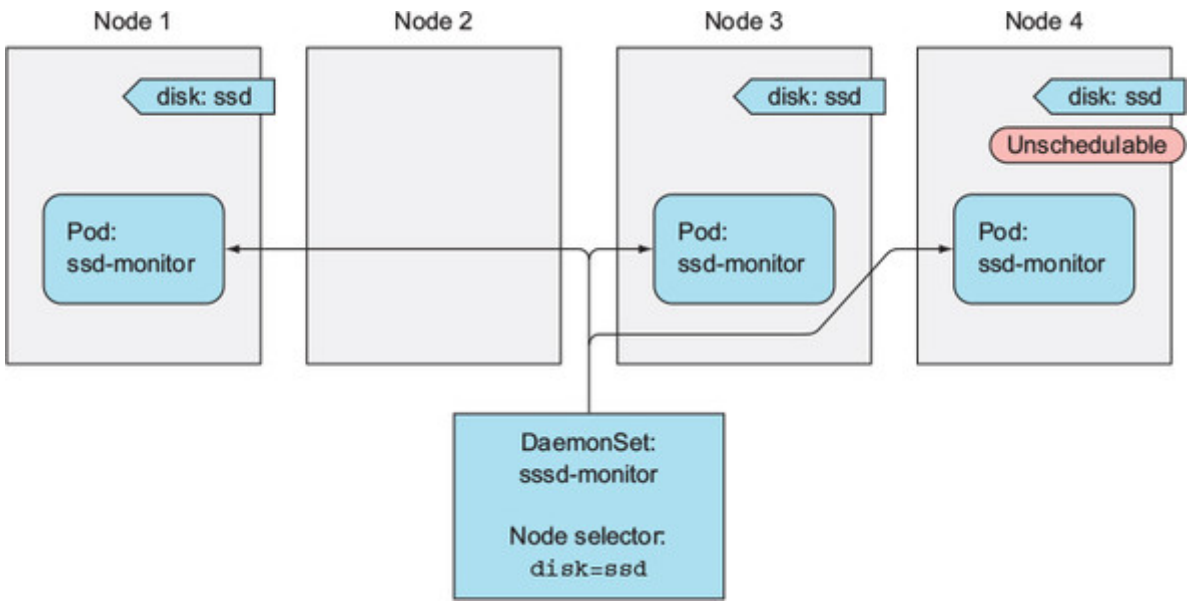
- 데몬 셋은 레플리케이션 컨트롤러, 레플리카 셋과 매우 유사하나 무작위로 흩어져 배포되지 않는다. 그림처럼 노드 수만큼 파드를 만들고, 각 노드에 배포된다.
- 데몬 셋은 복제본 수를 확인하는 절차는 없지만, 파드 셀렉터와 일치하는 파드 하나가 각 노드에서 실행하는 중인지 확인한다.
- 노드가 다운된다고 데몬셋은 새 파드를 생성하진 않는다. 대신 새로운 노드가 만들어지면 새 파드 인스턴스를 만들어 새로운 노드에 배포한다. 마찬가지로 누군가 데몬셋이 관리하는 파드를 삭제한 경우, 데몬셋이 새로운 파드 인스턴스를 만들어 해당 노드에 배포한다.

특정 노드에서만 파드 실행

그러면 데몬셋은 무조건 모든 노드에 하나의 파드를 배포하는 것인가?

- 만약 따로 지정하지 않으면 말그대로 클러스터 모든 노드에 파드를 배포하게 된다.
- 특정 노드에만 배포하고 싶다면 파드 셀렉터 정의에 `node-Selector` 속성을 사용하면 된다.

SSD 예제 → `disk=ssd` 레이블을 갖는 노드에만 배포하고 싶은 상황




```
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
  name: ssd-monitor
spec:
  selector:
    matchLabels:
      app: ssd-monitor
  template:
    metadata:
      labels:
        app: ssd-monitor
    spec:
```

```
      nodeSelector:   # Node Selector 지정
        disk: ssd
      containers:
      - name: main
        image: luksa/ssd-monitor
```

데몬 셋 명령어

데몬 셋의 명령어는 대부분 레플리케이션 컨트롤러, 레플리카 셋과 유사하다. 동작방식에만 차이가 있다고 보면 된다.

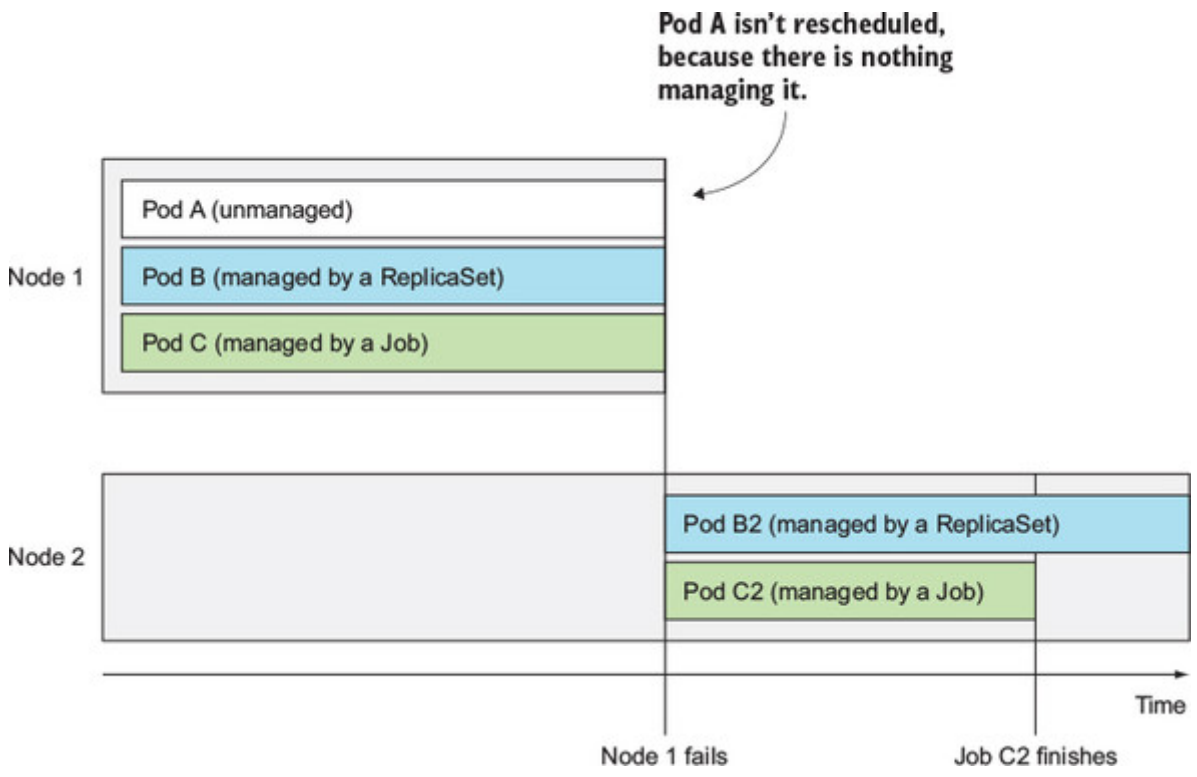
- 노드에 새로운 레이블을 추가하면, 데몬 셋이 이를 감지하고 파드 하나를 새로운 노드에 배포한다.
  - `kubectl label node node-name disk=ssd`
- 노드의 레이블을 변경해 데몬 셋이 더이상 관리하지 않게 되면, 배포했던 파드를 삭제한다.
  - `kubectl label node node-name disk=hdd --overwrite`

4.5 완료 가능한 단일 태스크 수행하는 파드

지금까지 배운 레플리케이션 컨트롤러, 레플리카 셋, 데몬 셋은 '완료'의 개념이 없는 중단 없는 파드. 즉, 지속적인 업무를 수행하는 파드였다.
하지만 **단일 태스크를 수행하고 종료되는 파드의 개념은 없는 것일까?**

잡 리소스

Job이 위 말했던 역할을 수행한다.



실행 중인 프로세스가 성공적으로 완료되면 컨테이너를 다시 시작하지 않는 파드.

- 작동 중 프로세스에 장애가 발생한 경우, 컨테이너를 다시 시작할지 설정이 가능하다.
- 제대로 완료되는 것이 중요한 임시작업에 유용하다.

잡 리소스 정의

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-batch-job
spec:
  completions: 5
  parallelism: 2
  template:
    metadata:
```

```
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
      - name: main
        image: luksa/batch-job
```

`restartPolicy`: 프로세스가 종료될 때, 쿠버네티스가 수행할 작업을 지정할 수 있다.

- 일단 파드의 기본값은 `Always` 인데, 잡은 무한정 실행하지 않으므로 이것으로 지정이 불가능하다.
- `OnFailure`, `Never` 속성을 사용해 에러가 났을 경우에만 재시작하거나, 재시작하지 않는 속성으로 설정해주어야 한다.

`completions`: 순차적으로 몇개의 파드를 실행할 것인지 지정할 수 있다.

- 순차적으로 여러 잡을 계속해서 실행해야 하는 경우 사용하면 된다.
- 현재 5로 설정되어있어 잡 파드가 순차적으로 5번 수행된다. 즉, 하나의 잡이 완료되면 새로운 잡이 생기고 이러한 과정이 5번 반복되어 수행된다.

`parallelism`: 병렬로 잡 파드 실행할 수 있다.

- 단순히 하나씩 순차적으로 수행하는 것이 아니라 여러개의 잡 파드를 병렬적으로 수행할 수 있다.
- 2로 설정해두었으면, 2개의 잡 파드가 한번에 실행되어 프로세스를 수행하게 된다.
- 이 부분은 일반 파드의 replicas와 의미가 비슷하다. (동시에 몇개 수행해야한다는 뜻이므로)
  - 정의 후에 만약 parallelism을 수정하고 싶다면 `kubectl scale job job-name --replicas 3` 처럼 scale 명령어를 통해 수정이 가능하다.

4.6 크론 잡 실행하기

cron은 보통 리눅스를 사용할 때 유용하게 사용된다. 주기적으로 파일을 실행하거나 특정 명령어를 수행해야할 때 crontab에 등록해놓으면 된다. 잡도 이러한 역할과 비슷하게 크론을 통해 주기적으로 파드를 실행하는 기능이 있다.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  startingDeadlineSeconds: 15
  jobTemplate:
    spec:
# ...
```

- cron은 분/ 시/ 일/ 월/ 요일 로 띄어쓰기로 나뉘어 작성된다.
  - 즉 위 예제는 매 15분마다 job이 실행되도록 정의되어 있는 것이다.
- 지정된 시간 내에 작동하지 않으면 작동해서는 안되는 경우가 있을 수 있다.
  - 그런 경우에 `startingDeadlineSeconds` 조건을 두어서 특정 시간 안에 작동하지 않으면 작업을 수행하지 않도록 지정할 수 있다.
  - 여기서는 매 15분 마다 작동하도록 정의되었으나, 정해진 시간의 15초 이내 잡을 실시하지 않으면 수행하지 않도록 정의해두었다.