4. Replication and other controllers(4.1liveness probes~4.2ReplicationController)

이번 장에서
4.1. KEEPING PODS HEALTHY
- 4.1.1. Introducing liveness probes
- 4.1.2. Creating an HTTP-based liveness probe
- 4.1.3. 활동중인 liveness probe 보기
- 4.1.4. Configuring additional properties of the liveness probe
- 4.1.5. 효과적인 liveness probes 만들기

4.2. INTRODUCING REPLICATIONCONTROLLERS
- 4.2.1. ReplicationController의 작동
- 4.2.2. ReplicationController 만들기
- 4.2.3.작동중인 ReplicationController 보기
- 4.2.4. Moving pods in and out of the scope of a ReplicationController
- 4.2.5. pod template변경
- 4.2.6. pods 스케일 확장
- 4.2.7. ReplicationController 삭제

궁금한 것

이번 장에서

- Keeping pods healthy
- Running multiple instances of the same pod
- Automatically rescheduling pods after a node fails
- Scaling pods horizontally
- Running system-level pods on each cluster node
- Running batch jobs
- Scheduling jobs to run periodically or once in the future

- pod는 배포 가능한 기본 단위, 수동으로 생성/감독/관리
  (실제 사용 사례는 배포가 자동으로 작동 및 실행되고 수동 개입이 없는 정상 원함)
- ReplicationController 또는 Deployments와 같이 다른 유형의 리소스를 만든 다음 실제 pods를 만들고 관리한다.
- Kubernetes컨테이너는 자동으로 다시 시작하나, 전체 노드에 대해 실패하면 노드의 pods가 손실되고 관리되지않는 한 새 포드로 교체되지않는다.
- 이번 장에서는 무기한으로 실행되는 pods와 작업 수행을 한 다음 중지하는 pods, 관리되는 pods의 실행방법을 배운다.

4.1. KEEPING PODS HEALTHY

- Kubernetes의 사용 이점 : 컨테이너 목록을 제공, 해당 컨테이너가 클러스터의 어딘가에서 계속 실행
- pod가 노드에 예약 → Kubelet은 컨테이너를 실행 한 다음 포드가 존재하는 한 계속 실행
- 컨테이너의 기본 프로세스가 충돌하면 Kubelet이 컨테이너를 다시 시작(**자체 치유 기능**)
- 앱이 프로세스 충돌없이 멈추는 경우, 앱이 작동하지 않는다는 신호를 Kubernetes에 알리고 Kubernetes가 다시 시작하도록하는 방법
- 충돌이 있는 컨테이너는 자동 재시작되므로 앱에서 오류를 포착하면 재시작하나, 모든 문제를 해결하지는 못함.

4.1.1. Introducing liveness probes

리베네스 프로브?/활성상태 프로브

- liveness probes : 컨테이너가 아직 살아있는지 확인가능
- 주기적으로 probes를 실행하고 probes가 실패하면 재시작

```
! liveness probes =/= readiness probes
# 2 개 헷갈리지 말기~!
```

Kubernetes의 컨테이너 프로브 세 가지 메커니즘(1 택)

- ***HTTP GET***

  컨테이너의 IP 주소에 프로브 수행하는 HTTP GET 요청. 사용자가 지정한 포트와 경로.
  프로브가 응답을 수신하고 응답 코드가 오류를 나타내지 않는 경우 (즉, HTTP 응답 코드가 2xx 또는 3xx 인 경우) 프로브는 성공한 것으로 간주됩니다. 서버가 오류 응답 코드를 반환하거나 전혀 응답하지 않는 경우 프로브는 실패로 간주되고 결과적으로 컨테이너가 다시 시작됩니다.

- ***TCP 소켓***

  프로브는 컨테이너의 지정된 포트에 TCP 연결을 열려고합니다.
  연결이 성공적으로 설정되면 프로브가 성공한 것입니다. 그렇지 않으면 컨테이너가 다시 시작됩니다.

- ***간부***

  프로브는 용기 검사 명령의 종료 상태 코드 안에 임의의 명령을 실행한다. **상태 코드가 0**이면 프로브가 성공한 것입니다. 다른 모든 코드는 실패로 간주됩니다.

4.1.2. Creating an HTTP-based liveness probe

- HTTP 기반 liveness probe 만들기

HTTP GET liveness probe 를 포함하는 새 포드를 생성

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-liveness
spec:
  containers:
  - image: luksa/kubia-unhealthy       ❶
    name: kubia
    livenessProbe:                     ❷
      httpGet:                         ❷
        path: /                        ❸
        port: 8080                     ❹
```

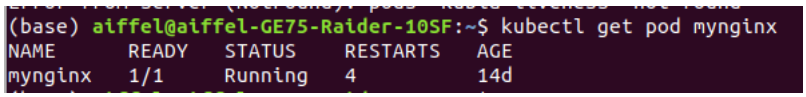
- ❶ **(다소) 깨진 앱이 포함 된 이미지입니다.**
- ❷ **HTTP GET을 수행 할 활성 프로브**
- ❸ **HTTP 요청에서 요청할 경로**
- ❹ **probe가 연결되어야하는 네트워크 포트**

- **pod descriptor** : 컨테이너가 httpGet정상인지 확인하기 위해, HTTP GET 요청을 주기적으로 수행하도록 지시 하는 활성 프로브를 정의

4.1.3. 활동중인 liveness probe 보기

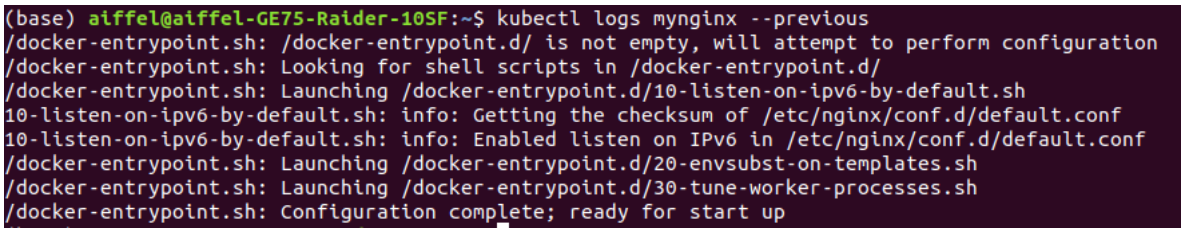
컨테이너가 다시 시작된 후 포드의 설명

```
$ kubectl get pod mynginx
```



충돌 한 컨테이너의 응용 프로그램 로그 얻기

```
$ kubectl logs mynginx --previous
```



- 순차적으로 실행됐다.

컨테이너가 다시 시작된 후 포드의 설명

```
$ kubectl describe po kubia-liveness
Name:           kubia-liveness
...
Containers:
  kubia:
    Container ID:       docker://480986f8
    Image:              luksa/kubia-unhealthy
    Image ID:           docker://sha256:2b208508
    Port:
    State:              Running                                     ❶
      Started:          Sun, 14 May 2017 11:41:40 +0200             ❶
    Last State:         Terminated                                  ❷
      Reason:           Error                                       ❷
      Exit Code:        137                                         ❷
      Started:          Mon, 01 Jan 0001 00:00:00 +0000             ❷
      Finished:         Sun, 14 May 2017 11:41:38 +0200             ❷
    Ready:              True
    Restart Count:      1                                           ❸
    Liveness:           http-get http://:8080/ delay=0s timeout=1s
                        period=10s #success=1 #failure=3
    ...
Events:
... Killing container with id docker://95246981:pod "kubia-liveness ..."
    container "kubia" is unhealthy, it will be killed and re-created.
```

- ❶컨테이너가 현재 작동 중입니다.
- **❷ 이전 컨테이너가 오류로 종료되고 코드 137로 종료되었습니다.**
- **❸ 컨테이너가 한 번 다시 시작되었습니다.**

- 종료 코드는 137 : 128+x, x는 프로세스를 종료하게 만든 신호 번호

```
❗ 컨테이너가 종료되면 완전히 새로운 컨테이너가 생성
(클라우드 상의 엔드포인트같은게 바뀌려나...?)
```

- 컨테이너를 어떤 옵션으로 실행하는지에 따라 다르다. →

4.1.4. Configuring additional properties of the liveness probe

- 활성 상태 프로브의 추가 속성 구성

- liveness probe를 더 자세하게 표기할 수 있다

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

`delay=0s`: 컨테이너가 시작된 직후

`failure=3:` 프로브가 연속으로 3 번 실패시 컨테이너 재시작

초기 지연이 있는 경우

```
livenessProbe:
     httpGet:
       path: /
       port: 8080
     initialDelaySeconds: 15             ❶
```

❶ Kubernetes will wait 15 seconds before executing the first probe.(첫 번째 프로브를 실행하기 전에 15초 대기)

```
💡
Exit code 137 signals that the process was killed by an external signal (exit code is 128 + 9 (SIGKILL). Likewise, exit code 143 corre
```

종료 코드는 137프로세스에 의해 종료되었다. 종료코드 128+9, 종료코드 143은 128+ 15

4.1.5. 효과적인 liveness probes 만들기

- 프로덕션에서 실행되는 pod는 항상 liveness probe를 정의 해야한다.
- 없으면 앱이 살아있는지 모르고, 프로세스가 실행되면 컨테이너 정상으로 간주

*** 활성 프로브가 확인해야하는 사항***

단순한 liveness probe는 단순히 서버가 응답하는지 확인합니다.

> 💡 팁
> Make sure the /health HTTP endpoint doesn’t require authentication; otherwise the probe will always fail, causing your container to be

/health HTTP endpoint의 인증이 필요하지 않은지 확인, 안그러면 프로브 항상 실패하여 컨테이너 무기한 다시시작.

- 외부 요인 확인하기
  Ex. 프론트엔드 웹서버 liveness probe는 서버가 백엔드 DB에 연결할 수 없을 때, 오류를 반환하지 않음. 원인이 DB에 있는 경우, 문제 해결이 안되고 계속 컨테이너가 반복적으로 실행됨

프로브를 가볍게 유지

liveness probes는 너무 많은 계산 리소스를 사용하지 않아야하며 완료하는 데 너무 오래 걸리지 않아야합니다.

- 프로브는 비교적 자주 실행되며 완료되는데 1초만 허용
- 무거운 liveness probe를 사용하면 CPU시간이 줄어든다.

> 💡 팁
> 컨테이너에서 Java 앱을 실행하는 경우 Exec 프로브 대신 HTTP GET liveness probe를 사용해야합니다.

- 추측 : 비동기적으로 되야하니까..!!

***프로브에서 재시도 loops 구현에 시간쓰지말라!**

- failure threshold를 구성할 수 있다.
- failure threshold를 1로 설정하더라도 쿠버네티스는 여러 전 프로브를 재시도 한다.
- 그러니까 자체 재시도 루프를 구현하는 것은 낭비

⇒failure 이런 옵션도 있는데 구태여 구현을 해야하냐(카더라)

***Liveness 프로브 마무리**

쿠버네티스의 충돌, liveness probe가 실패하는 경우 컨테이너를 다시 시작. 컨테이너를 계속 실행

위 작업은 pod를 호스팅하는 노드의 **Kubelet**에 의해 수행

- 노드 자체가 충돌하면 노드와 함께 작동 중지된 모든 pod에 대한 교체를 생성하는 것은 Control Plane
- Kubelet은 노드 자체에서 실행되기때문에 **노드가 실패하면 아무 것도 할 수 없다.**

앱이 다른 노드에서 다시 시작되도록하는 부분은 다음장에서

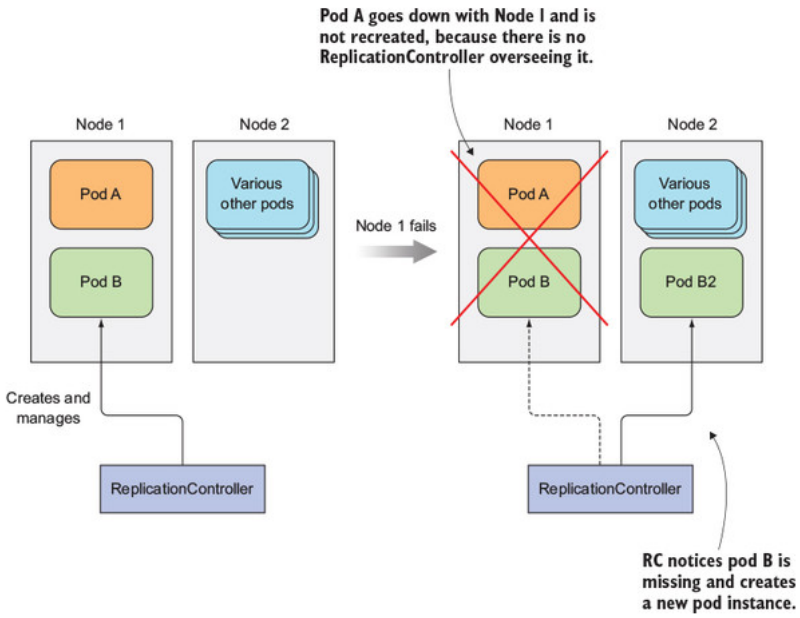
4.2. INTRODUCING REPLICATIONCONTROLLERS

- pod가 계속 실행되도록 하는 리소스

노드가 다운되고 2개의 포드를 가져갈 때 어떤 일이 발생하는지 보여주는 그림

- A = 그냥
- B = ReplicationController에 의해 관리되는 포드

Figure 4.1. When a node fails, only pods backed by a ReplicationController are recreated.



`pod A` 는 손실되고 `pod B2` 가 생성된다.

- 그림처럼 단일 포드가 아닌 Replication-Controller는 포드의 여러 복사본 (복제본)을 만들고 관리하기위한 것

4.2.1. ReplicationController의 작동

- ReplicationController는 실제 포드 수가 항상 원하는 수와 일치하는지 확인
- 실행중인 포드가 적으면 포드 템플릿에서 새 복제본 생성, 너무 많이 실행되면 초과 복제본 제거

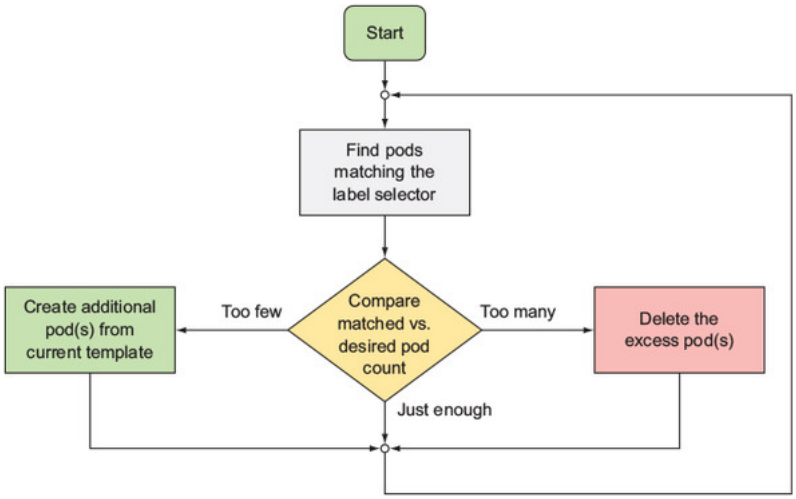
초과 복제본이 발생하는 이유

- 누군가가 동일한 유형의 포드를 수동으로 생성합니다.
- 누군가 기존 포드의 "유형"을 변경합니다.
- 누군가가 원하는 포드 수를 줄이는 식입니다.

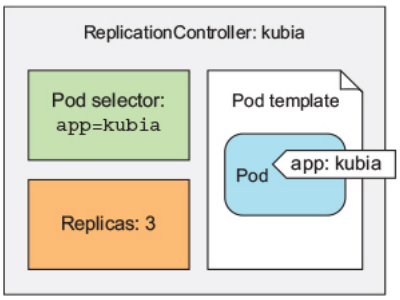
*** controller's reconciliation loop**

포드 수 = 항상 label selector와 일치하는지 확인

실제 값을 원하는 수로 조정하기 위한 조치 그림



***ReplicationController의 주요 부분 세 가지**



- *label selector*
  ReplicationController의 범위에 있는 포드 결정
- *replica count*
  실행해하는 포드 수를 지정
- *pod template*

  새로운 복제본을 만들 때 사용

- 레이블 선택기 및 창 템플릿에 대한 변경 사항은 기존 창에 영향을주지 않는다.

ReplicationController 사용의 이점 이해

- 기존 포드가 누락 된 경우 새 포드를 시작하여 포드가 항상 실행되고 있는지 확인
- 클러스터 노드가 실패하면 실패한 노드에서 실행 중이던 모든 포드에 대한 대체 복제본을 생성
- 이를 통해 수동 및 자동으로 포드의 수평 확장을 쉽게 할 수 있다

4.2.2. ReplicationController 만들기

```
apiVersion: v1
kind: ReplicationController        ❶
metadata:
  name: kubia                      ❷
spec:
  replicas: 3                      ❸
  selector:                        ❹
```

```
    app: kubia                    ❹
  template:                       ❺
    metadata:                     ❺
      labels:                     ❺
        app: kubia                ❺
    spec:                         ❺
      containers:                 ❺
      - name: kubia               ❺
        image: luksa/kubia        ❺
        ports:                    ❺
        - containerPort: 8080     ❺
```

- ❶이 매니페스트는 ReplicationController (RC)를 정의합니다.
- **❷ 이 ReplicationController의 이름**
- **❸ 원하는 포드 인스턴스 수**
- **❹ RC가 작동중인 포드를 결정하는 포드 선택기**
- **❺ 새 포드 생성을위한 포드 템플릿**

```
팁
ReplicationController를 정의 할 때 포드 선택기를 지정하지 마세요
포드 템플릿에서 추출하도록하면 YAML이 더 간단하다.
```

```
$ kubectl create -f kubia-rc.yaml
replicationcontroller "kubia"생성됨
```

4.2.3.작동중인 ReplicationController 보기

```
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
kubia-53thy 0/1 ContainerCreating 0 2s
kubia-k0xz6 0/1 ContainerCreating 0 2s
kubia-q3vkg 0/1 ContainerCreating 0 2s
```

포드 하나 수동 삭제하여 ReplicationController가 포드 수를 다시 3개로 돌리는 방법

```
# 삭제된 포드 확인
$ kubectl delete pod kubia-53thy
pod "kubia-53thy"삭제됨
```

삭제한 포드가 종료되고 새 포드 생성

```
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
kubia-53thy 1/1 Terminating 0 3m
kubia-oini2 0/1 ContainerCreating 0 2s
kubia-k0xz6 1/1 Running 0 3m
kubia-q3vkg 1/1 Running 0 3m
```

ReplicationController에 대한 정보를 얻는 방법

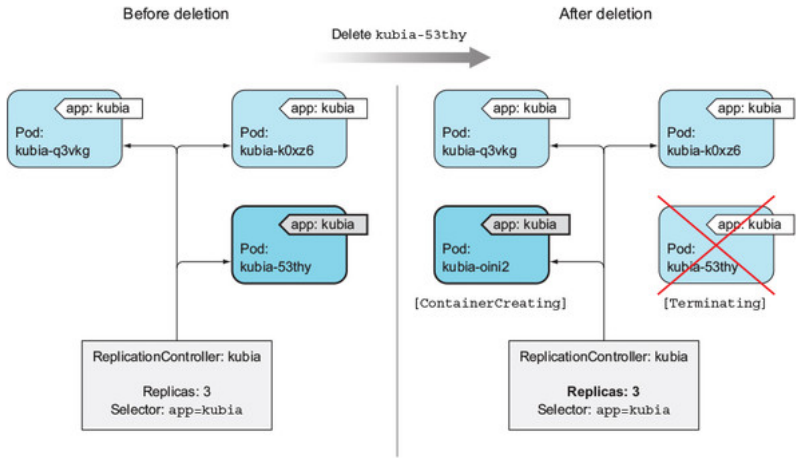
```
$ kubectl get rc
```

ReplicationController의 세부 정보 표시

```
$ kubectl describe rc kubia
Name:           kubia
Namespace:      default
Selector:       app=kubia
Labels:         app=kubia
Annotations:    <none>
Replicas:       3 current / 3 desired                                      ❶
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed             ❷
Pod Template:
  Labels:       app=kubia
  Containers:   ...
  Volumes:      <none>
Events:                                                                    ❸
From                    Type      Reason           Message
----                    -------   ------           -------
replication-controller  Normal    SuccessfulCreate  Created pod: kubia-53thy
replication-controller  Normal    SuccessfulCreate  Created pod: kubia-k0xz6
replication-controller  Normal    SuccessfulCreate  Created pod: kubia-q3vkg
replication-controller  Normal    SuccessfulCreate  Created pod: kubia-oini2
```

- **❶ 실제 대 원하는 수의 포드 인스턴스**
- **❷ 포드 상태 당 포드 인스턴스 수**
- **❸ 이 ReplicationController와 관련된 이벤트**

포드가 사라지면 ReplicationController는 너무 적은 포드를보고 새 교체 포드를 만듭니다.



회색 → 컨트롤러 가 있고 새 교체 포드를 만드는 그림

4.2.4. Moving pods in and out of the scope of a ReplicationController

ReplicationController에서 관리하는 포드에 레이블 추가

```
$ kubectl label pod kubia-dmdck type=special
pod "kubia-dmdck" labeled

$ kubectl get pods --show-labels
```

```
NAME          READY   STATUS    RESTARTS   AGE   LABELS
kubia-oini2   1/1     Running   0          11m   app=kubia
kubia-k0xz6   1/1     Running   0          11m   app=kubia
kubia-dmdck   1/1     Running   0          1m    app=kubia,type=special
```

관리 형 포드의 레이블 변경

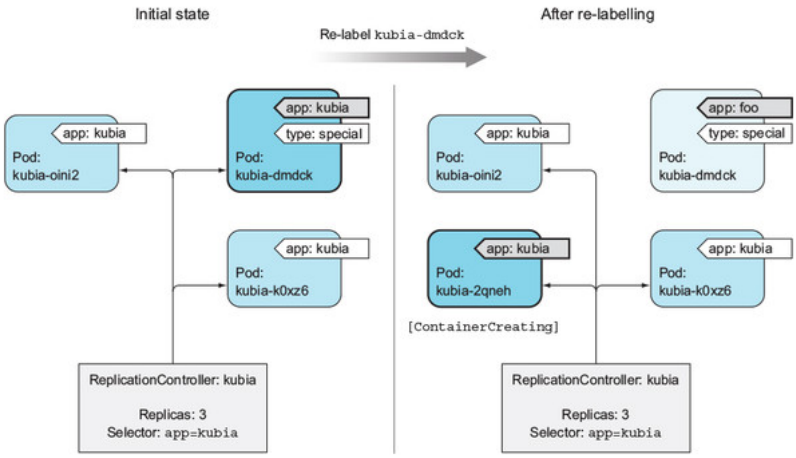
```
$ kubectl label pod kubia-dmdck app = foo --overwrite
pod "kubia-dmdck"labeled
```

모든 포드 다시 나열 : 4개의 포드 표시

```
$ kubectl get pods -L app
NAME          READY  STATUS              RESTARTS  AGE  APP
kubia-2qneh   0/1    ContainerCreating   0         2s   kubia        ❶
kubia-oini2   1/1    Running             0         20m  kubia
kubia-k0xz6   1/1    Running             0         20m  kubia
kubia-dmdck   1/1    Running             0         10m  foo          ❷
```

- **❶ ReplicationController의 범위에서 제거한 pod를 대체하는 새로 생성 된 pod**
- **❷ 더 이상 ReplicationController에서 관리하지 않는 포드**

레이블을 변경하여 ReplicationController의 범위에서 포드 제거

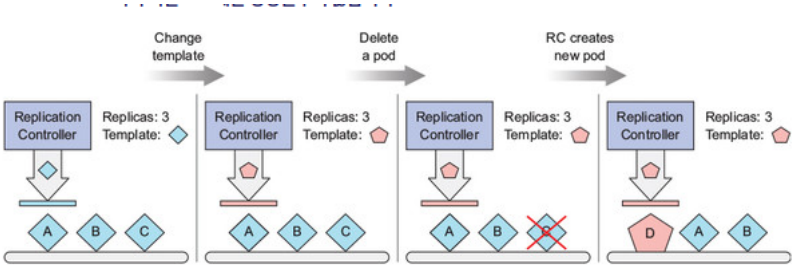


- 레이블로 관리할 포드를 결정하는 조건,
- 대상을 명확하게 선택

4.2.5. pod template변경

ReplicationController의 포드 템플릿은 언제든지 수정할 수 있습니다.

- 쿠키 커터를 교체하는 것과 같다. 쿠키에만 영향
- 이전 포드를 수정하려면 해당 포드를 삭제하고 새 템플릿 기반으로 새 포드로 교체



- ReplicationController를 편집 명령어

```
$ kubectl edit rc kubia
```

변경사항을 저장하면 나타나는 메세지

```
replicationcontroller "kubia" edited
```

⇒ 이 방법보다 더 좋은 방법은 9장에서

- 다른 텍스트 편집기를 이용해서 kubectl edit를 이용할 수 있다.
  - 환경 변수 kubectl를 설정

```
export KUBE_EDITOR = "/ usr / bin / nano"
# 나노로 텍스트 편집기 사용
```

4.2.6. pods 스케일 확장

- 복제본 수 변경에는 스케일 확장이 있다.
- pod수를 늘리고 줄이는 것은 쉽다.

1. **ReplicationController 확장**

숫자를 10까지 확대

```
$ kubectl scale rc kubia --replicas=10
```

2. **ReplicationController의 editing을 편집하여 확장**

```
$ kubectl edit rc kudkkubia
```

텍스트 편집기가 열리면

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving
# this file will be reopened with the relevant failures.
apiVersion: v1
kind: ReplicationController
metadata:
  ...
```

```
spec:
  replicas: 3           ❶
  selector:
    app: kubia
  ...
```

❶ 3을 10으로 변경

하고 저장하면 포드수 10개!

```
$ kubectl get rc
NAME      DESIRED   CURRENT   READY     AGE
kubia     10        10        4         21m
```

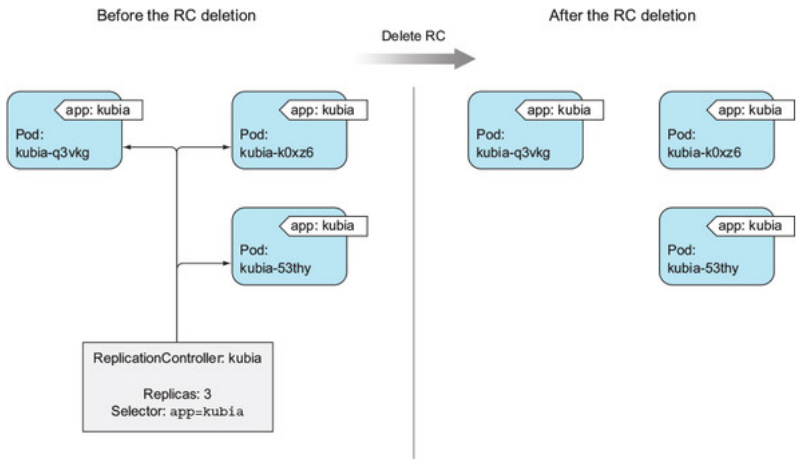
줄이는 방법

```
$ kubectl scale rc kubia --replicas=3
```

수동으로 확인하는 방법

⇒ 위 방법처럼 간단하게 확인 후 숫자를 변경하는 것이 더 쉬울 수 있다.

4.2.7. ReplicationController 삭제



- ReplicationController를 ReplicaSet으로 대체하기로 결정할 때 유용 할 수 있다.

```
$ kubectl delete rc kubia --cascade=false
replicationcontroller "kubia" deleted
```

포드가 자체적으로 유지되도록 ReplicationController를 삭제를 했으나, *replica count*로 언제든 다시 만들 수 있다.

궁금한 것

- 규리님 : 파드가 손상되면 자동으로 되는데.... liveness probes가 있는지
  - 실제로 서버는 돌아가는데 확인은 안되는
  - 켜저는 있고 동작은 하는 것같은데 응답은 없고....
  - 체크업을 위해서...!!

- 상우님 : 개별 노드로 ?????
  - 외부 유저가 ?????

  → 어플리케이션 로드 밸런서 : 설치 자동으로 ~ 설정할 수 있는게 있다.
  - ALB(로드 밸런서)