4장 레플리케이션과 그 밖의 컨트롤러 : 포드 배포 관리

1. 발표순서
   방효균(149p~166p)
   김병렬(166p~181p)
   이재원(181p~197p)

2. 내용정리

- 수동으로 생성한 포드는 관리되지 않아 포드나 노드에 문제가 발생할 경우 해당 포드는 교체되지 않아 장애가 발생한다.

- 쿠버네티스는 포드를 안정적으로 유지하기 위해 컨테이너에 문제가 생길 경우 스케줄에 따라 Kubelet이 자동으로 컨테이너를 다시 실행한다. 컨테이너에 크래시가 발생하여 중단될 경우 별다른 작업없이 쿠버네티스는 자동으로 복구할 수 있다. 하지만 Application 상 문제가 발생하면 크래시 없이 서비스에만 문제가 발생하므로 Liveness Probe를 설정해야 한다.

- Liveness Probe에는 3가지 매커니즘이 있다.
  1) HTTP GET Probe : IP주소, 포트, 경로에 HTTP GET을 요청하여 2xx 또는 3xx가 아닐 경우 실패로 간주한다
  2) TCP Socket Probe : TCP 연결을 시도하여 연결되지 않으면 실패로 간주한다.
  3) Exec Probe : 컨테이너 내부에 임의의 명령을 실행하고 명령의 종료 상태코드가 0이 아니면 실패로 간주한다.

- Liveness Probe는 추가옵션으로 Delay, Timeout, Period, Failure 설정이 가능

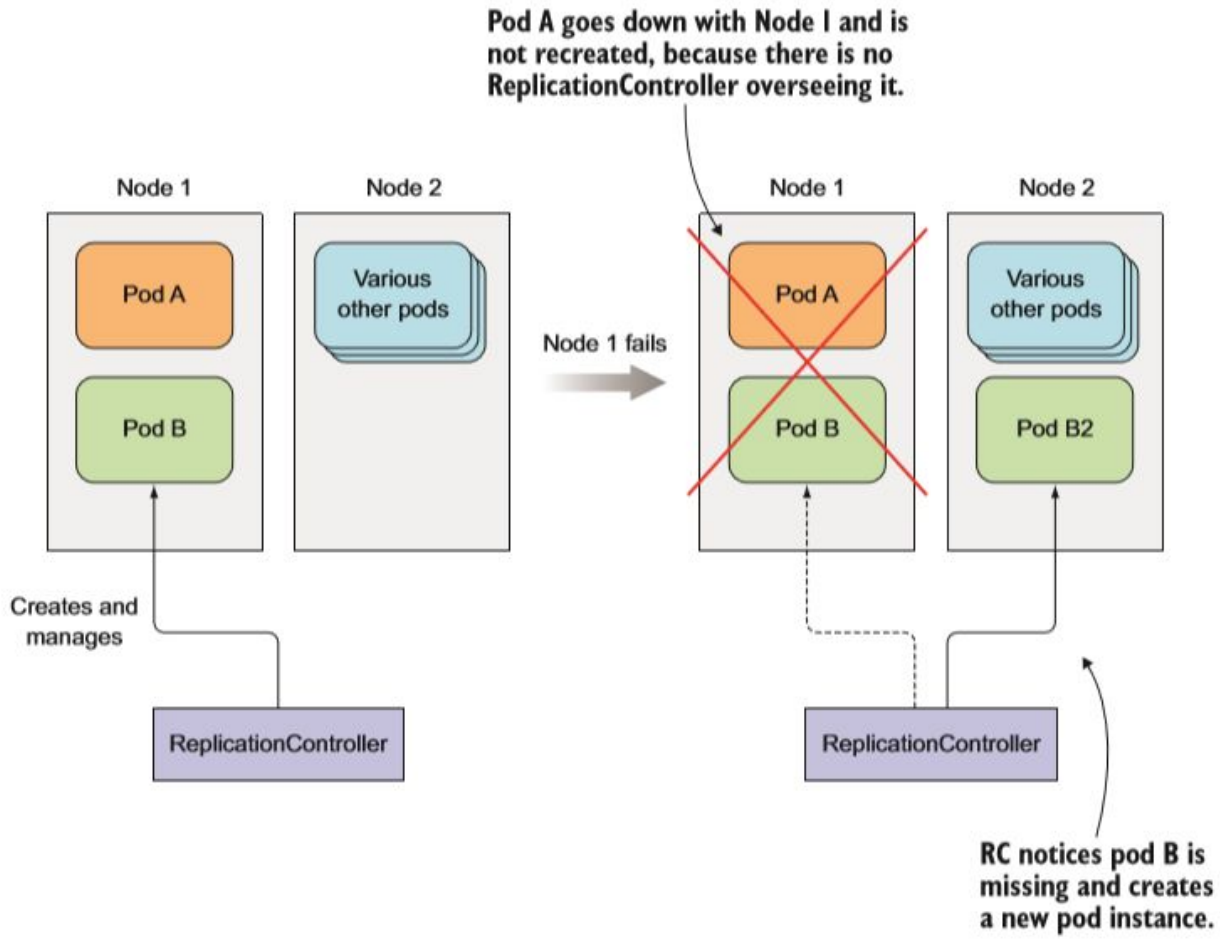
- Liveness Probe Endpoint는 /health 같은 별도 페이지로 구성하는 것이 좋다. 너무 많은 계산 리소스를 사용하면 안되고 완료하는데 너무 오래 걸리지 않아야 한다.
  예) 자사에서는 최소한의 간단한 로직이 포함된 페이지를 구성하여 Health Check 세팅

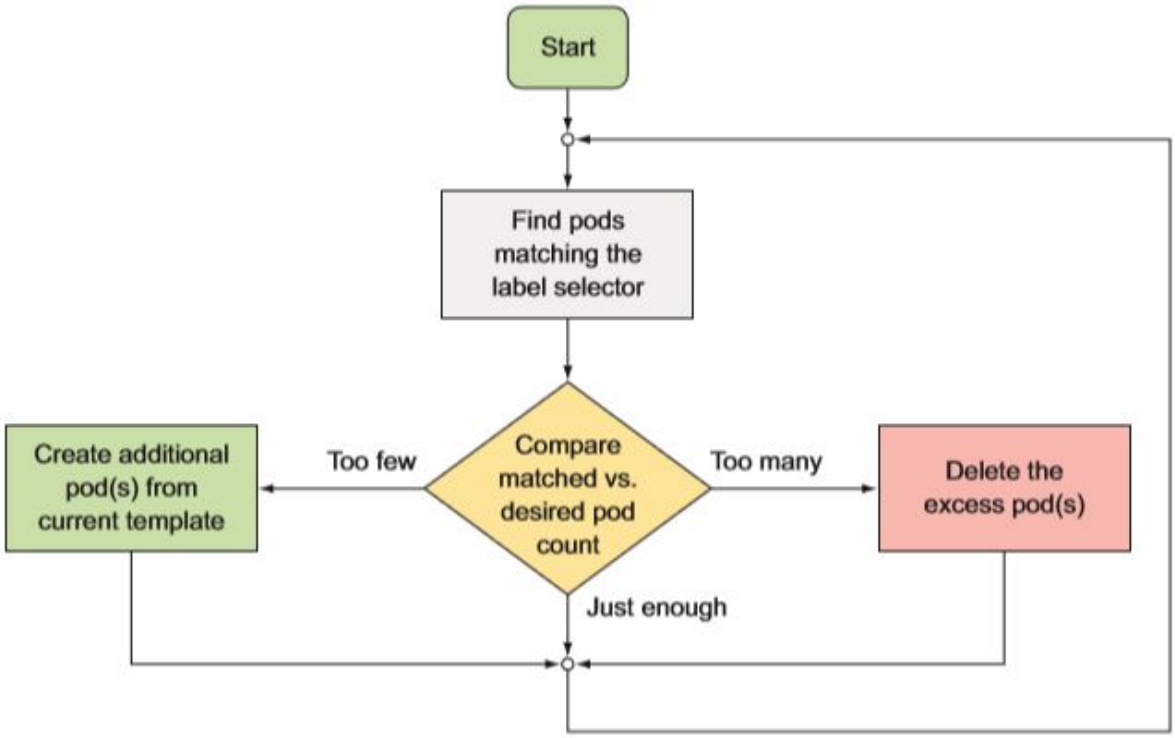
- Kubectl describe 종료코드
  137 : SIGKILL(9)
  144 : SIGTERM(15)

  <참조>
  0: Success
  125: Docker run itself fails
  126: Contained command cannot be invoked
  127: Containerd command cannot be found
  128 + n: Fatal error signal n:
  130: (128+2) Container terminated by Control-C
  137: (128+9) Container received a SIGKILL

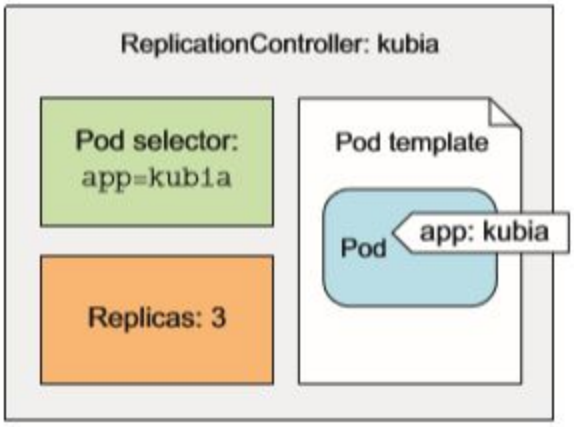
143: (128+15) Container received a SIGTERM
255: Exit status out of range(-1)

- ReplicationController는 포드가 항상 실행되도록 유지하는 리소스. 누락된 포드를 감지하고 대체 포드를 만든다.





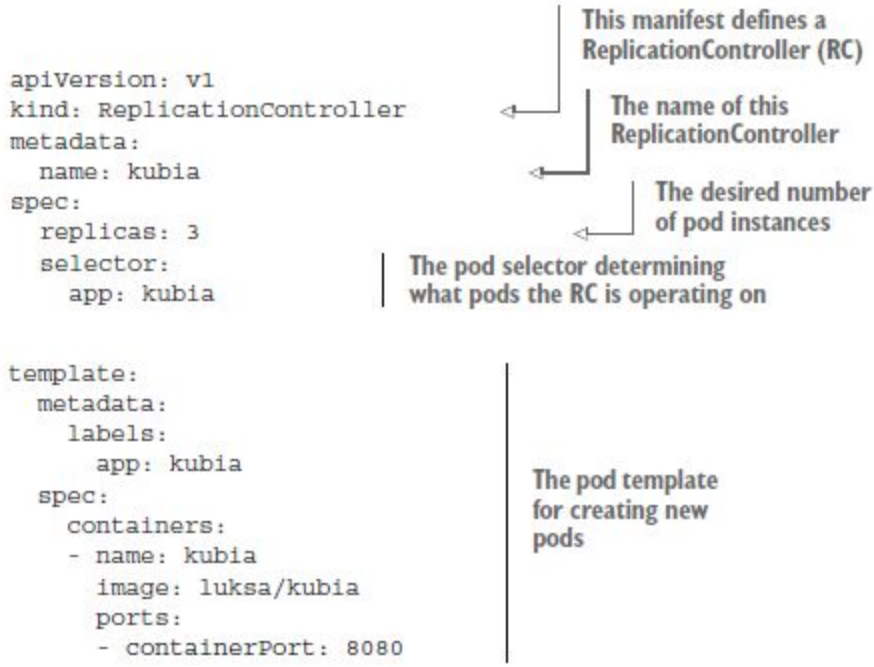
- ReplicationController의 역할은 정확한 수의 포드가 항상 라벨 셀렉터와 일치하는지 확인하는 것이다.



- 세가지 요소
  - ReplicationController 범위에 있는 포드를 결정하는 라벨 셀렉터
  - 실행해야 하는 노드의 원하는 수를 지정하는 복제본 수 - 변경시 기존 포드에 영향을 미친다.
  - 새로운 포드 복제본을 만들 때 사용되는 포드 템플릿

- ReplicationController 이점
  - 포드가 항상 실행되도록 한다.
  - 노드에 장애가 발생하면 실행 중인 모든 포드의 대체 복제본을 만든다.
  - 수평 스케일링 할 수 있다.

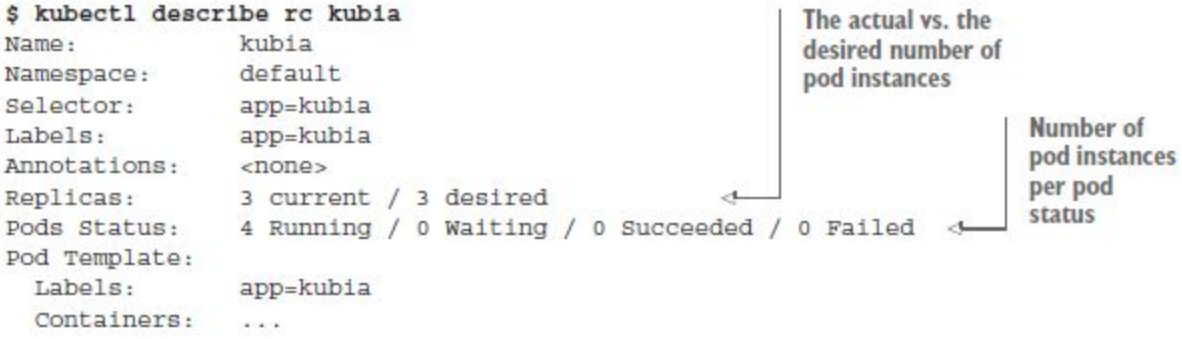
- ReplicationController YAML




```
apiVersion: v1
kind: ReplicationController
metadata:
  name: kubia
spec:
  replicas: 3
  selector:
    app: kubia

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

- ReplicationController에서 Pod 하나를 삭제한 후 상태 확인




```
$ kubectl describe rc kubia
Name:           kubia
Namespace:      default
Selector:       app=kubia
Labels:         app=kubia
Annotations:    <none>
Replicas:       3 current / 3 desired
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:       app=kubia
  Containers:   ...
```

```
  Volumes:      <none>
Events:
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
```

The events related to this ReplicationController

컨트롤러가 새로운 포드를 생성할 때 야기 되는 것

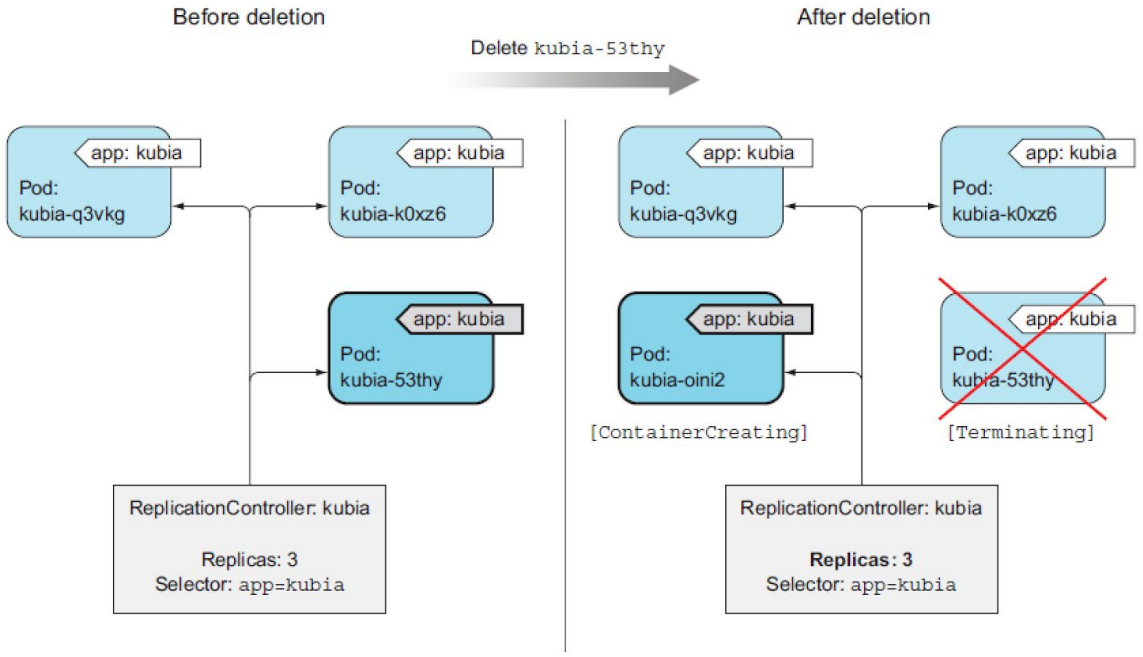


Figure 4.4 If a pod disappears, the ReplicationController sees too few pods and creates a new replacement pod.

API 서버 -> 리소스 및 리소스 목록 변경 사항 모니터링 -> 컨트롤러 트리거: 실제 포드 수 확인 -> 포드 수 적음 -> 새 대체 포드를 생성

노드 장애에 대한 응답

Listing 4.6 Simulating a node failure by shutting down its network interface

```
$ gcloud compute ssh gke-kubia-default-pool-b46381f1-zwko
Enter passphrase for key '/home/luksa/.ssh/google_compute_engine':

Welcome to Kubernetes v1.6.4!
...

luksa@gke-kubia-default-pool-b46381f1-zwko ~ $ sudo ifconfig eth0 down
```

K8s가 노드의 다운 감지: 1~2분 소요

```
$ kubectl get node
NAME                                     STATUS     AGE
gke-kubia-default-pool-b46381f1-opc5     Ready      5h
gke-kubia-default-pool-b46381f1-s8gj     Ready      5h
gke-kubia-default-pool-b46381f1-zwko     NotReady   5h
```

Node isn't ready, because it's disconnected from the network

노드에 수분 간 도달 할 수 없는 경우

```
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
kubia-oini2   1/1     Running   0          10m
kubia-k0xz6   1/1     Running   0          10m
kubia-q3vkg   1/1     Unknown   0          10m
kubia-dmdck   1/1     Running   0          5s
```

This pod's status is unknown, because its node is unreachable.

This pod was created five seconds ago.

노드 리셋 -> 노드 상태 Ready -> Unknown pod 삭제

```
$ gcloud compute instances reset gke-kubia-default-pool-b46381f1-zwko
```

4.2.4 ReplicationController 범위 밖으로 포드 이용

라벨 추가 => 이전과 동일(3개)

```
$ 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
```

관리되는 포드의 label 변경 => 하나는 관리 하지 않고, 나머지 3개 관리 => 포드 새로 생성

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

Newly created pod that replaces the pod you removed from the scope of the ReplicationController

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

Pod no longer managed by the ReplicationController

포드 kubia-zqneh 를 스핀업 => replica 수를 3으로 유지

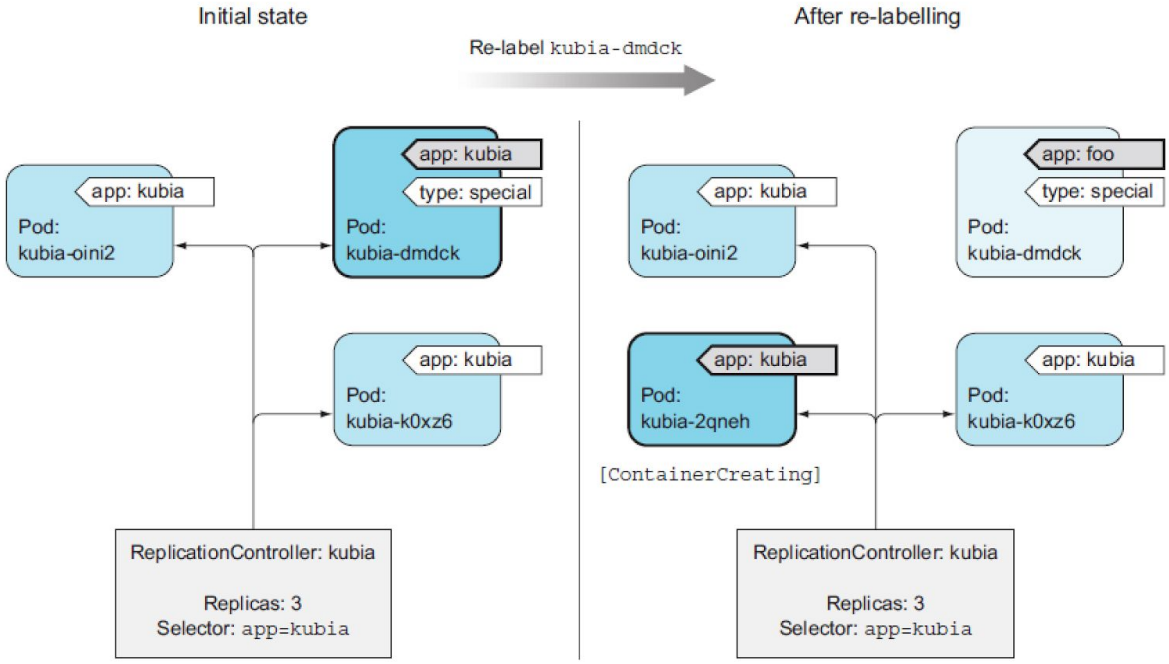


Figure 4.5 Removing a pod from the scope of a ReplicationController by changing its labels

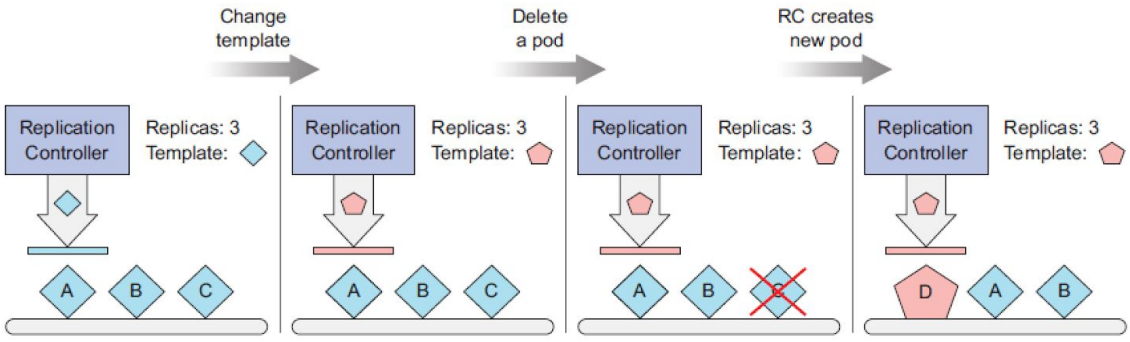


Figure 4.6 Changing a ReplicationController's pod template only affects pods created afterward and has no effect on existing pods.

포드 템플릿 변경 => 이후에 생성된 포드에만 영향을 미치고 기존 포드에는 영향이 없음.

포트 템플릿 변경

```
$ kubectl edit rc kubia
```

```
replicationcontroller "kubia" edited
```

다른 텍스트 편집기 사용

```
export KUBE_EDITOR="/usr/bin/nano"
```

수평 포드 스케일링

Scale up

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

ReplicationController의 정의 편집

```
$ kubectl edit rc kubia
```

Listing 4.7 Editing the RC in a text editor by running `kubectl edit`

```
# 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
  selector:
    app: kubia
  ...
```

Change the number 3 to number 10 in this line.

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

Scaling down

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

스케일링을 위한 선언적 접근법
- 원하는 상태만 지정
- K8s 클러스터와 상호 작용 쉽게 함
- 15장에서 HPA : K8s가 스스로 수행

리플리케이션 컨트롤러 삭제: 해당 포드에 영향을 주지 않고 리플리케이션컨트롤러만 삭제 가능함

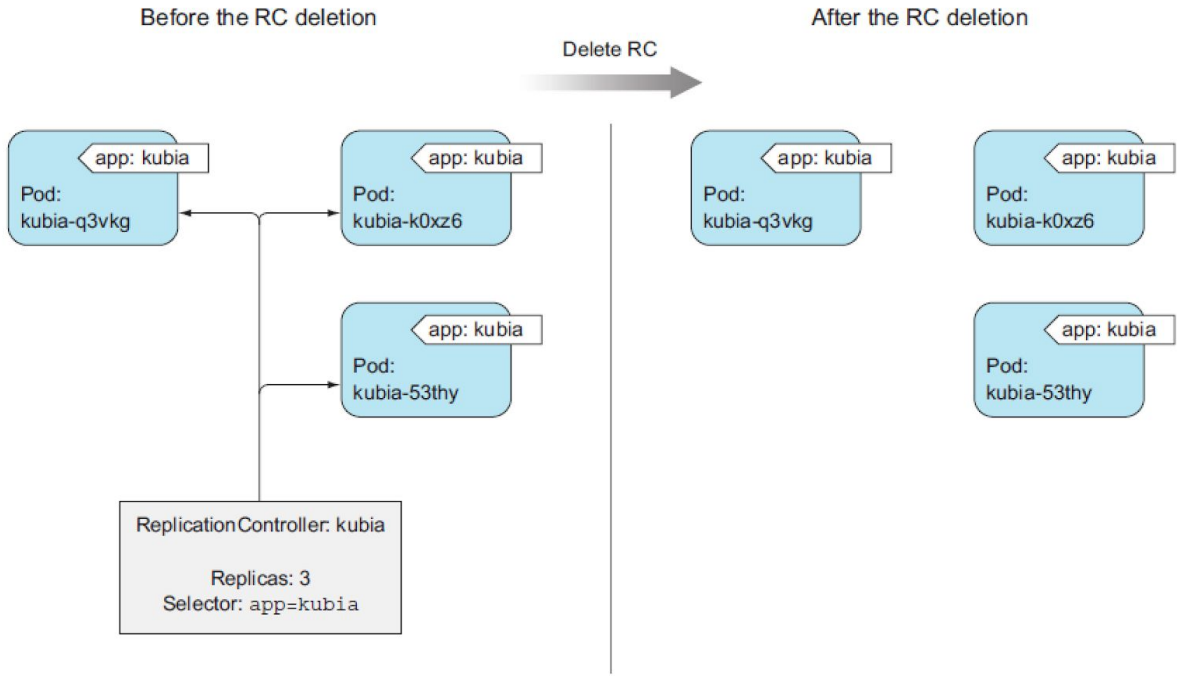


Figure 4.7 Deleting a replication controller with `--cascade=false` leaves pods unmanaged.

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

ReplicationController 대신 ReplicaSet 사용

단일 ReplicationController는
포드를 env = production 이고 env = devel 인 레이블은 동시에 일치시킬 수 없습니다.
: env = procudtion 레이블이 있는 포드 또는
env = devel 레이블이 포드 중 하나만 일치

단일 ReplicaSet은 두 포드 세트를 일치시켜 단일 그룹으로 처리 할 수 있음

- env=* 도 가능

Listing 4.8 A YAML definition of a ReplicaSet: kubia-replicaset.yaml

```
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: kubia
spec:
  replicas: 3
  selector:
    matchLabels:
      app: kubia
  template:
    metadata:
      labels:
        app: kubia
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
```

ReplicaSets aren't part of the v1 API, but belong to the apps API group and version v1beta2.

You're using the simpler matchLabels selector here, which is much like a ReplicationController's selector.

The template is the same as in the ReplicationController.

```
$ kubectl get rs
NAME      DESIRED   CURRENT   READY     AGE
kubia     3         3         3         3s
```

```
$ kubectl describe rs
Name:           kubia
Namespace:      default
Selector:       app=kubia
Labels:         app=kubia
Annotations:    <none>
Replicas:       3 current / 3 desired
Pods Status:    3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:       app=kubia

  Containers:   ...
  Volumes:      <none>
Events:         <none>
```

```
$ kubectl delete rs kubia
replicaset "kubia" deleted
```

Listing 4.9 A `matchExpressions` selector: kubia-replicaset-matchexpressions.yaml

```
selector:
  matchExpressions:
    - key: app
      operator: In
      values:
        - kubia
```

This selector requires the pod to contain a label with the "app" key.

The label's value must be "kubia".

- `In`–Label's value must match one of the specified `values`.
- `NotIn`–Label's value must not match any of the specified `values`.
- `Exists`–Pod must include a label with the specified key (the value isn't important). When using this operator, you shouldn't specify the `values` field.
- `DoesNotExist`–Pod must not include a label with the specified key. The `values` property must not be specified.

4.4 *Running exactly one pod on each node with DaemonSets*

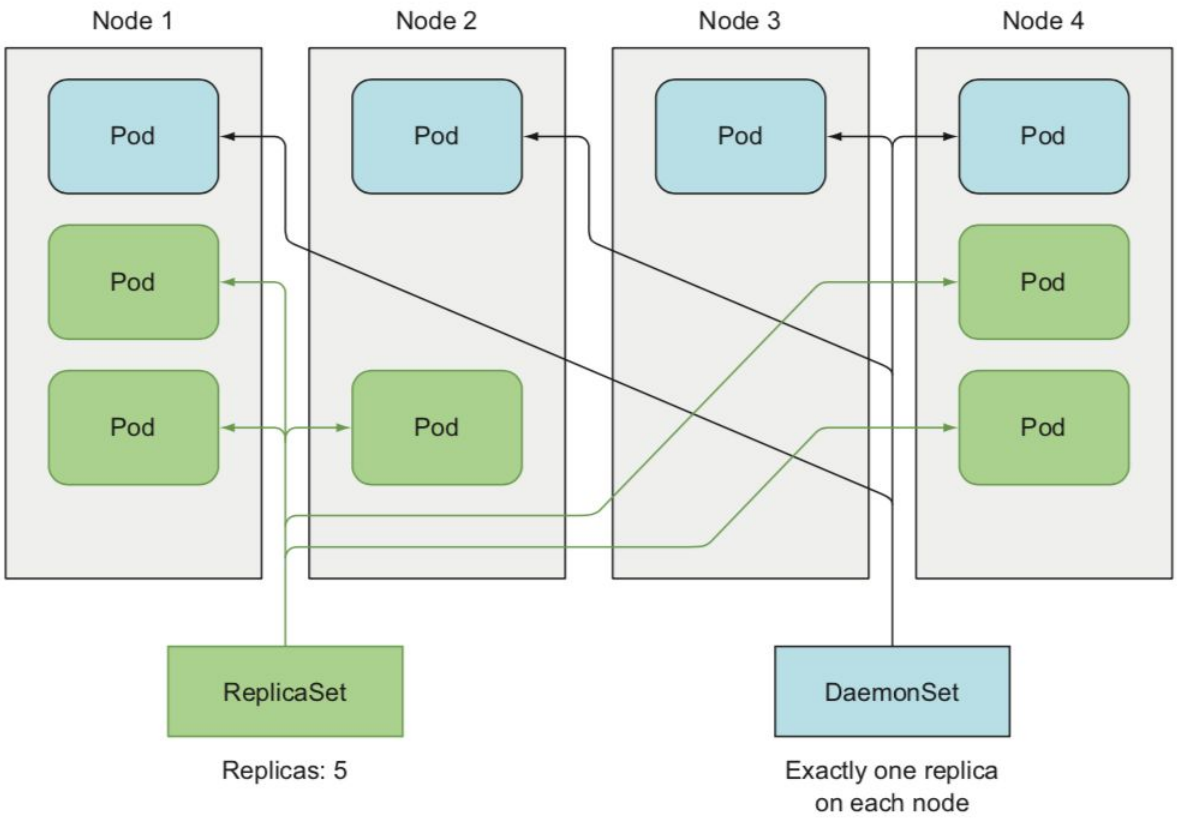


Figure 4.8 DaemonSets run only a single pod replica on each node, whereas ReplicaSets scatter them around the whole cluster randomly.

- 클러스터 각 노드에 pod를 실행해야 하는 경우
- 인프라 관련 pod
    - ex) Log Collector, Resource Monitor, **kube-proxy**

4.4.1 *Using a DaemonSet to run a pod on every node*

- 데몬셋에 의해 만들어진 pod는 이미 대상이되는 노드가 지정돼 있고 Kubrenetes Schduler는 건너뛴다.
- 노드 수 만큼 pod 생성하고 노드에 하나씩 배포된다.
- 데몬셋에는 replica count의 개념이 없다.
- 노드 다운 시, 새로운 노드에 배포되지 않는다 (추가 배포되지 않는다)
- 새로운 노드 추가시 새 pod가 배포됨. 만약 삭제되면 새 pod가 배포됨.
- ReplicaSet 과 마찬가지로 pod template을 사용하여 pod를 생성함.

4.4.1 *Using a DaemonSet to run pods only on certain nodes*

- 기본적으로 모든 노드에 pod를 배포함.
- node-Selector 속성을 사용해 데몬셋이 배포해야할 노드를 지정 할 수 있음.

예제를 통한 데몬셋 설명

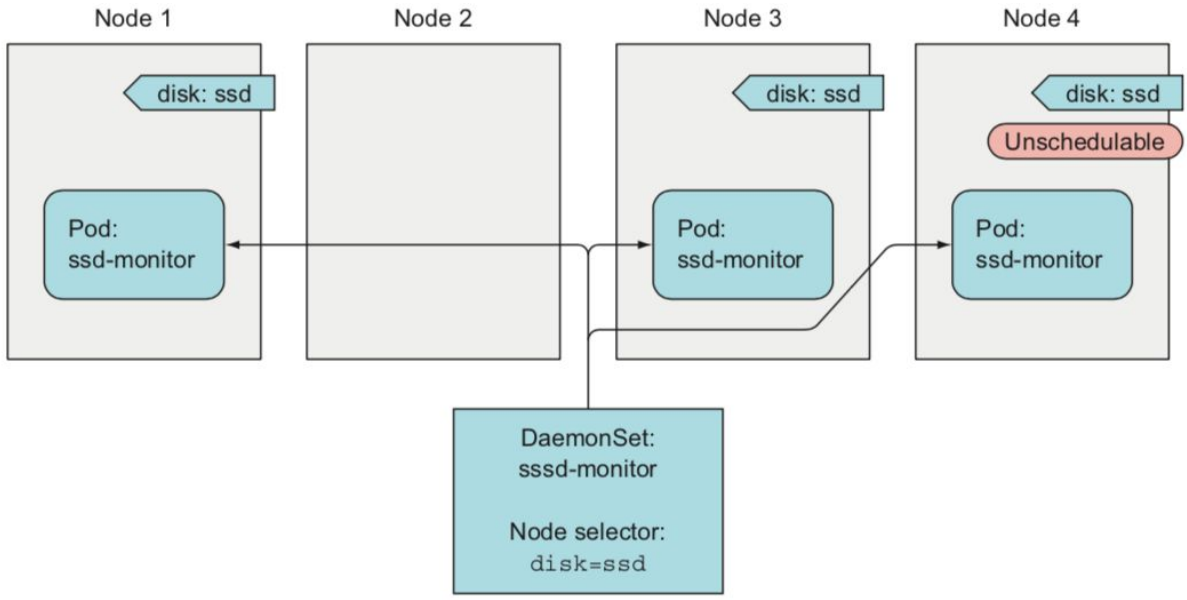


Figure 4.9 Using a DaemonSet with a node selector to deploy system pods only on certain nodes

데몬셋 YAML 정의 만들기

Listing 4.10 A YAML for a DaemonSet: ssd-monitor-daemonset.yaml

```
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
  name: ssd-monitor
spec:
  selector:
    matchLabels:
      app: ssd-monitor
  template:
    metadata:
      labels:
        app: ssd-monitor
    spec:
      nodeSelector:
        disk: ssd
      containers:
      - name: main
        image: luksa/ssd-monitor
```

DaemonSets are in the apps API group, version v1beta2.

The pod template includes a node selector, which selects nodes with the disk=ssd label.

- disk=ssd 라벨이 있는 각 노드에 생성된다.

데몬셋 만들기

```
$ kubectl create -f ssd-monitor-daemonset.yaml
daemonset "ssd-monitor" created
```

Let's see the created DaemonSet:

```
$ kubectl get ds
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE-SELECTOR
ssd-monitor   0         0         0       0            0           disk=ssd
```

Those zeroes look strange. Didn't the DaemonSet deploy any pods? List the pods:

```
$ kubectl get po
No resources found.
```

- 데몬셋을 만들었지만 노드에 disk=ssd 라벨을 지정하지 않았기 때문에 배포된 pod가 없다.
- 데몬셋은 노드의 라벨이 변경되는 것을 감지한다. 매칭되는 라벨이 생기면 배포한다.

필요한 라벨을 노드에 추가

```
$ kubectl get node
NAME       STATUS    AGE       VERSION
minikube   Ready     4d        v1.6.0
```

Now, add the `disk=ssd` label to one of your nodes like this:

```
$ kubectl label node minikube disk=ssd
node "minikube" labeled
```

NOTE Replace `minikube` with the name of one of your nodes if you're not using Minikube.

The DaemonSet should have created one pod now. Let's see:

```
$ kubectl get po
NAME                READY     STATUS    RESTARTS   AGE
ssd-monitor-hgxwq   1/1       Running   0          35s
```

- 노드 중 하나의 라벨을 변경하면, 데몬셋의 pod template 에 설정된 pod가 생성된다.

노드에서 특정한 라벨을 제거

```
$ kubectl label node minikube disk=hdd --overwrite
node "minikube" labeled
```

```
$ kubectl get po
NAME                READY     STATUS        RESTARTS   AGE
ssd-monitor-hgxwq   1/1       Terminating   0          4m
```

- ‘’--overwrite”라는 속성을 이용해 라벨을 변경하면, 데몬셋으로 생성된 pod는 삭제된다.
- 데몬셋을 삭제하면 해당 pod도 삭제 된다.

4.5 *Running pods that perform a single completable task*

- ReplicationControllers, ReplicaSets, DaemonSets 으로 실행된 pod는 프로세스가 종료되면 다시 시작됨.
- Completable task 는 프로세스가 종료된 후 다시 실행되지 않는다.

4.5.1 잡 리소스 소개

- 노드 장애가 발생하면 잡이 관리되는 해당 노드의 pod는 replicaset의 pod와 같은 방식으로 다른 노드로 재스케쥴된다.

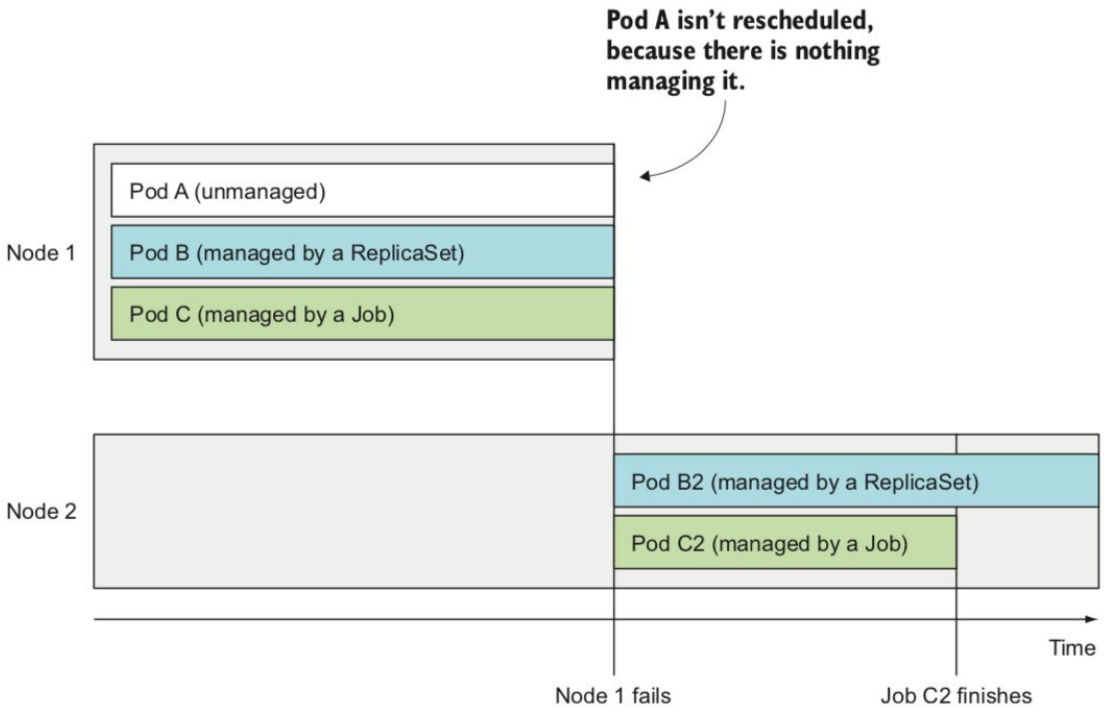


Figure 4.10 Pods managed by Jobs are rescheduled until they finish successfully.

- 처음에 스케줄된 노드가 실패했을 때 잡에 의해 생성된 포드를 새 노드에 다시 스케줄한다.
- 관리되지 않는 pod를 통해 실행된 task는 노드가 제거 되는 경우 task를 수동으로 재생성해야된다.

4.5.2 잡 리소스의 정의

Listing 4.11 A YAML definition of a Job: exporter.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: batch-job
spec:
  template:
    metadata:
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
      - name: main
        image: luksa/batch-job
```

Jobs are in the batch API group, version v1.

You're not specifying a pod selector (it will be created based on the labels in the pod template).

Jobs can't use the default restart policy, which is Always.

- 포드의 스펙에서 컨테이너에서 실행 중인 프로세서가 종료될 때 쿠버네티스가 수행해야하는 작업을 지정 할 수 있다.
  - restartPolicy: Always (# default) / OnFailure / Never
  - pod는 잡 리소스에서 관리되는 것이 아니다.

4.5.3 포드를 실행하는 잡 보기

```
$ kubectl get jobs
NAME        DESIRED   SUCCESSFUL   AGE
batch-job   1         0            2s

$ kubectl get po
NAME              READY     STATUS    RESTARTS   AGE
batch-job-28qf4   1/1       Running   0          4s


$ kubectl get po -a
NAME              READY     STATUS      RESTARTS   AGE
batch-job-28qf4   0/1       Completed   0          2m
```

- --show-all (-a) 파라메터를 사용하지 않으면 기본적으로 완료된 pod는 표시되지 않는다.

```
$ kubectl logs batch-job-28qf4
Fri Apr 29 09:58:22 UTC 2016 Batch job starting
Fri Apr 29 10:00:22 UTC 2016 Finished succesfully
```

- 완료될 때 삭제되지 않는 이유는 로그를 확인 할 수 있도록 하기 위함이다

4.5.4 잡에서 다수의 포드 인스턴스 실행

- 잡은 두 개 이상의 pod 인스턴스를 만들고 병렬 또는 순차적으로 실행하도록 구성할 수 있다.
- completions과 parallelism 속성으로 설정 가능하다.

Listing 4.12 A Job requiring multiple completions: multi-completion-batch-job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-batch-job
spec:
  completions: 5
  template:
    <template is the same as in listing 4.11>
```

Setting completions to 5 makes this Job run five pods sequentially.

- 5개의 pod를 차례로 실행한다.

병렬로 잡 포드 실행

Listing 4.13 Running Job pods in parallel: multi-completion-parallel-batch-job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-batch-job
spec:
  completions: 5
  parallelism: 2
  template:
    <same as in listing 4.11>
```

This job must ensure five pods complete successfully.

Up to two pods can run in parallel.

By setting `parallelism` to 2, the Job creates two pods and runs them in parallel:

```
$ kubectl get po
NAME                                READY   STATUS      RESTARTS   AGE
multi-completion-batch-job-lmmnk    1/1     Running     0          21s
multi-completion-batch-job-qx4nq    1/1     Running     0          21s
```

- 5개의 잡을 실행하고 2개의 pod를 병렬처리 한다.

잡 스케일링

```
$ kubectl scale job multi-completion-batch-job --replicas 3
job "multi-completion-batch-job" scaled
```

- 병렬 처리가 2 -> 3으로 증가함. 다른 pod이 스핀업 되었다.

4.5.5 잡 포드를 완료하는데 허용되는 시간 제한

- pod의 실행 시간은 activeDeadlineSeconds 속성으로 설정한다.
- pod가 그 보다 오래 실행되면 pod를 종료하려고 시도하고 잡을 실패한것으로 표시한다.

> 참고 : 잡 매니패스트의 spec.backoffLimit 필드로 재시도할 횟수를 설정 가능하다. 기본은 6회이다

4.6 Scheduling Jobs to run periodically or once in the future

- 리눅스의 CronJob과 거의 비슷한 내용을 구현 가능하다.

4.6.1 CronJob 만들기

Listing 4.14 YAML for a CronJob resource: cronjob.yaml

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: periodic-batch-job
        spec:
          restartPolicy: OnFailure
          containers:
          - name: main
            image: luksa/batch-job
```

API group is batch, version is v1beta1

This job should run at the 0, 15, 30 and 45 minutes of every hour, every day.

The template for the Job resources that will be created by this CronJob

- Spec.schedule 에 따라 잡 실행
- 잡 리소스의 템플릿도 지정 가능하다.

4.6.2 스케줄 된 잡의 실행 방법

- startingDeadlineSeconds 필드로 스케줄 된 시간을 넘어 잡을 실행 할 수 없다는 요구사항 설정 가능 하다.

Listing 4.15 Specifying a `startingDeadlineSeconds` for a CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
spec:
  schedule: "0,15,30,45 * * * *"
  startingDeadlineSeconds: 15
  ...
```

At the latest, the pod must start running at 15 seconds past the scheduled time.

- 위 내용을 15초 안에 반드시 실행되어야 한다. 그렇지 않으면 실패로 표시된다.

아래의 고려 사항이 필요

- 동시에 두개의 잡이 실행 될 경우,
  - 멱등성 고려 (Idompedent)
- 하나도 실행되지 않는 경우,
  - 이전 (실패한) 실행된 잡이 다음 잡에 영향을 미치는 확인 필요