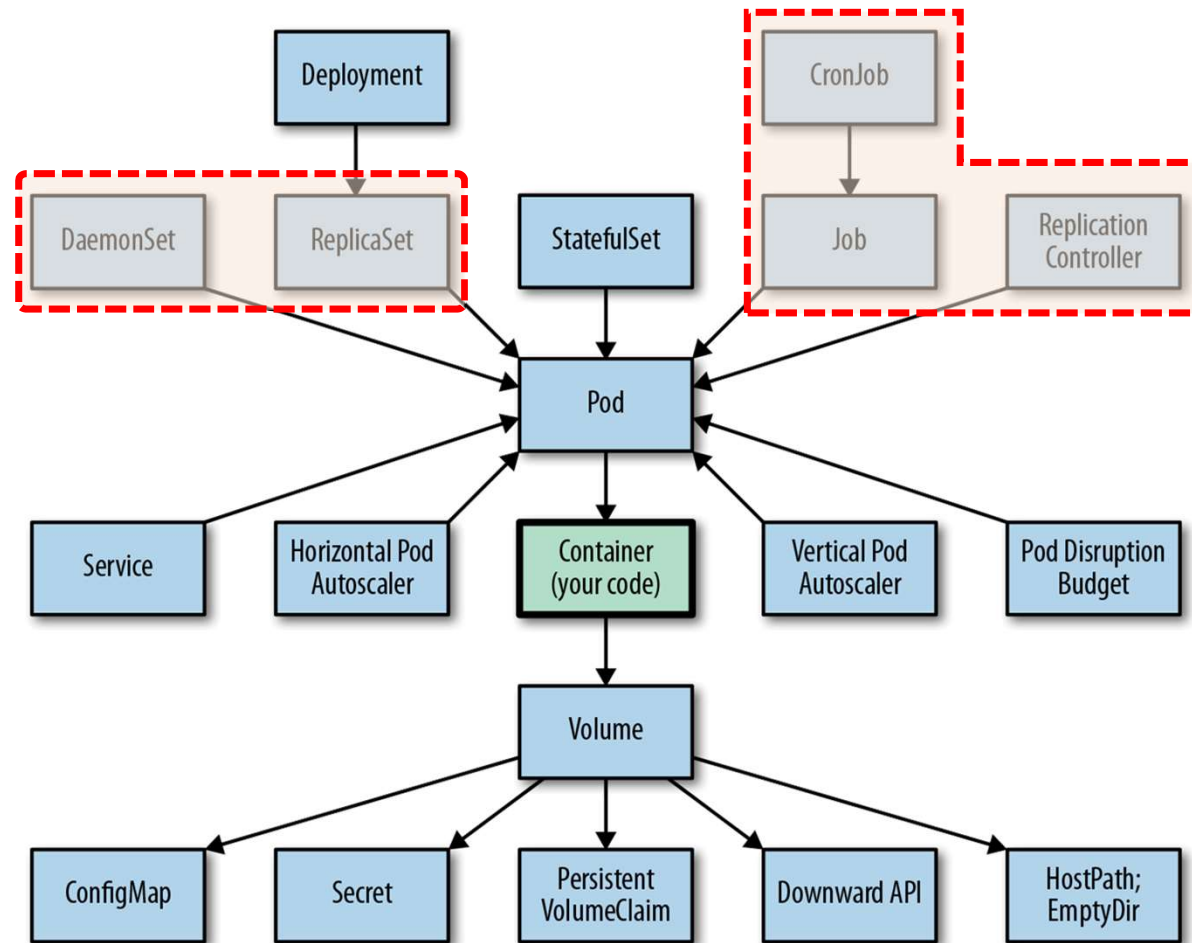


3rd
Week

Kubernetes concepts for developers



※ 참고 : <https://www.oreilly.com/library/view/kubernetes-patterns/9781492050278/ch01.html>

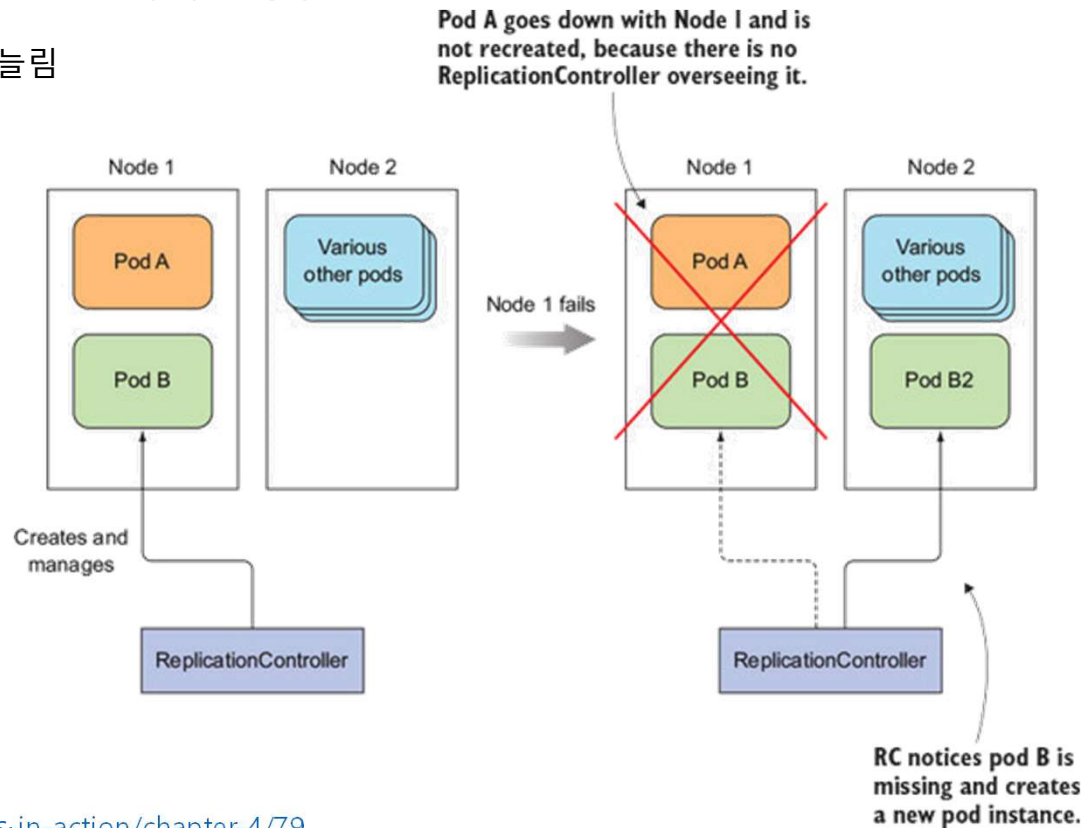


Kubernetes

ReplicaSet

ReplicationController - 1/2

- Pod가 항상 실행되도록 보장하는 리소스
 - . 어떤 이유로든 Pod가 사라지면 사라진 Pod를 감지해 교체 Pod를 생성
- 지속적으로 실행 중인 Pod 목록을 모니터링하고 선언된 수와 일치시킴
 - . 너무 많은 Pod가 실행 중이면 줄이고, 적으면 늘림

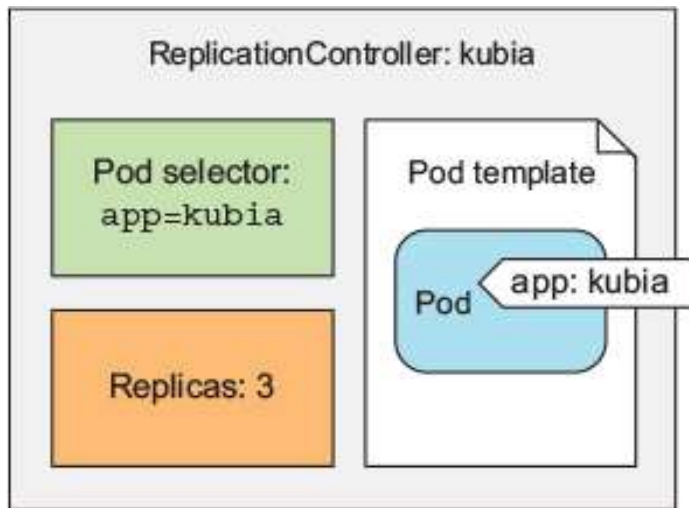


※ 참고 : <https://livebook.manning.com/book/kubernetes-in-action/chapter-4/79>

ReplicationController - 2/2

- ReplicationController 세 가지 필수 요소

- . **replicas**: Pod가 실행되어야 하는 수
- . **selector**: ReplicationController 범위에 있는 Pod 결정
- . **template**: 새로운 Pod replica를 만들 때 사용



```
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
```

※ 참고 : <https://livebook.manning.com/book/kubernetes-in-action/chapter-4/92>

ReplicaSet

- ReplicationController와 똑같이 동작
 - . 여기에 더해 selector에서 풍부한 표현식을 사용할 수 있음
 - . 특정 키가 있는 label을 갖는 Pod를 매칭
 - : **matchLabel** / **matchExpressions**
 - . label 조건을 Or/And로 정의
- 일반적으로 직접 사용하지는 않고,
Deployment 리소스를 생성할 때 자동으로 생성됨

```
apiVersion: apps/v1
kind: ReplicaSet

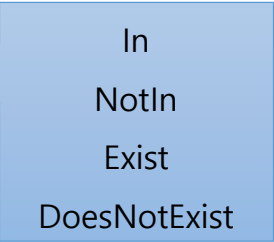
metadata:
  name: kubia

spec:
  replicas: 3

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

  template:
    metadata:
      labels:
        app: kubia

    spec:
      containers:
        - name: kubia
          image: luksa/kubia
          ports:
            - containerPort: 8080
```





K8s : ReplicaSet Hands-On

ReplicaSet Create

rs-web.yaml

```
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: rs-web

spec:
  replicas: 3

  selector:
    matchLabels:
      app: web

  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: node-web
          image: whatwant/node-web:1.0
          ports:
            - containerPort: 8080
```

```
remote > cd kubernetes/03-RS-DS-JOB-CJ/hands-on
```

```
remote > kubectl create -f rs-web.yaml
```

```
replicaset.apps/rs-labels created
```

```
remote > kubectl get replicaset -o wide
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
rs-web	3	3	3	113s	node-web	whatwant/node-web:1.0	app=web

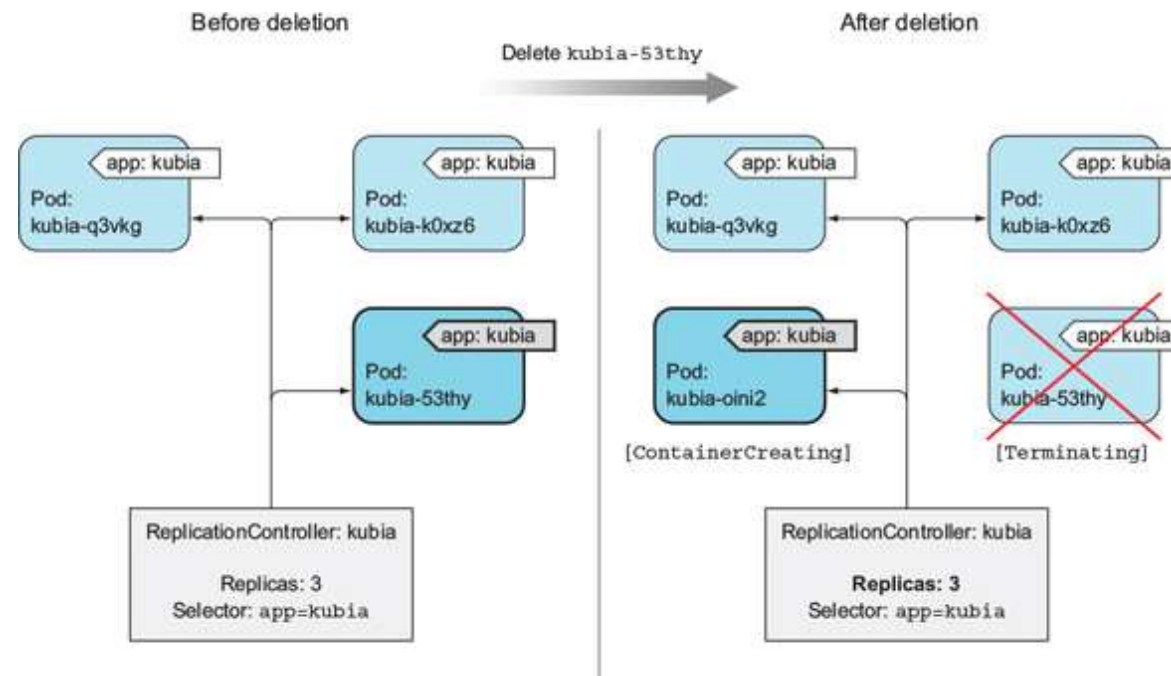
```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-8fsc7	1/1	Running	0	2m1s	10.233.110.4	worker1	<none>	<none>
rs-web-kxbnh	1/1	Running	0	2m1s	10.233.103.6	worker2	<none>	<none>
rs-web-zd98q	1/1	Running	0	2m1s	10.233.103.5	worker2	<none>	<none>

Pod Delete in ReplicaSet - 1/2

Pod가 삭제 되면 어떻게 될까?

삭제가 되는 등의 사유로 `selector`로 매핑이 되는 Pod의 숫자가 `Replicas`와 맞지 않으면, `template` 참조하여 Pod를 생성한다.
. 기존 Pod를 복제하는 것이 아니라 현재 시점의 template 정보를 참조하여 신규로 생성 → Update 기법으로 사용 가능



※ 참고 : <https://livebook.manning.com/book/kubernetes-in-action/chapter-4/141>

Pod Delete in ReplicaSet - 2/2

"kubectl get pods -o wide -w" 실행을 먼저 해놓고, "kubectl delete"를 진행하면 과정을 지켜볼 수 있다.

```
remote > kubectl delete pod rs-web-kxbnh
```

```
pod "rs-web-kxbnh" deleted
```

```
remote > kubectl get pods -o wide -w
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-8fsct	1/1	Running	0	15m	10.233.110.4	worker1	<none>	<none>
rs-web-kxbnh	1/1	Running	0	15m	10.233.103.6	worker2	<none>	<none>
rs-web-zd98q	1/1	Running	0	15m	10.233.103.5	worker2	<none>	<none>
rs-web-kxbnh	1/1	Terminating	0	16m	10.233.103.6	worker2	<none>	<none>
rs-web-d2jr2	0/1	Pending	0	0s	<none>	<none>	<none>	<none>
rs-web-d2jr2	0/1	Pending	0	0s	<none>	worker2	<none>	<none>
rs-web-d2jr2	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
rs-web-d2jr2	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
rs-web-d2jr2	1/1	Running	0	0s	10.233.103.7	worker2	<none>	<none>
rs-web-kxbnh	1/1	Terminating	0	16m	10.233.103.6	worker2	<none>	<none>
rs-web-kxbnh	0/1	Terminating	0	16m	10.233.103.6	worker2	<none>	<none>
rs-web-kxbnh	0/1	Terminating	0	16m	10.233.103.6	worker2	<none>	<none>
rs-web-kxbnh	0/1	Terminating	0	16m	10.233.103.6	worker2	<none>	<none>

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-8fsct	1/1	Running	0	21m	10.233.110.4	worker1	<none>	<none>
rs-web-d2jr2	1/1	Running	0	5m13s	10.233.103.7	worker2	<none>	<none>
rs-web-zd98q	1/1	Running	0	21m	10.233.103.5	worker2	<none>	<none>

Replicas Increase/Decrease (+ change Editor) - 1/2

replicas 숫자를 변경하고 싶으면 어떻게 해야 할까?

resource 내역을 수정하면 되는데, 기본 editor로 vi가 설정되어 있다.
개인적인 취향으로 기본 editor를 `nano`로 변경하고 시작해보겠다.

```
remote > export KUBE_EDITOR=nano
```

이제 설정을 변경해보자!

```
remote > kubectl get replicaset
```

NAME	DESIRED	CURRENT	READY	AGE
rs-web	3	3	3	25m

```
remote > kubectl edit replicaset rs-web
```

```
# 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: apps/v1
kind: ReplicaSet
metadata:
  creationTimestamp: "2023-05-20T16:28:11Z"
  generation: 1
  name: rs-web
  namespace: default
  resourceVersion: "70279"
  uid: 9d8c5422-9fcd-4e53-875b-5d4a8be8a05b
spec:
  replicas: 5
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      . . .
```

Replicas Increase/Decrease (+ change Editor) - 2/2

replicas 숫자를 변경하고 싶으면 어떻게 해야 할까?

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-8fsct	1/1	Running	0	32m	10.233.110.4	worker1	<none>	<none>
rs-web-d2jr2	1/1	Running	0	16m	10.233.103.7	worker2	<none>	<none>
rs-web-pdz7f	1/1	Running	0	5s	10.233.103.8	worker2	<none>	<none>
rs-web-qx9wb	1/1	Running	0	5s	10.233.110.5	worker1	<none>	<none>
rs-web-zd98q	1/1	Running	0	32m	10.233.103.5	worker2	<none>	<none>



Quiz

**ReplicaSet을 삭제해도
Pod는 유지가 될까?**

ReplicaSet Delete

```
remote > kubectl get replicaset -o wide
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
rs-web	5	5	5	36m	node-web	whatwant/node-web:1.0	app=web

```
remote > kubectl delete replicaset rs-web
```

```
replicaset.apps "rs-web" deleted
```

ReplicaSets에 의해 생성된 모든 Pods가 삭제되는 것을 볼 수 있다.

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-8fsct	1/1	Terminating	0	36m	10.233.110.4	worker1	<none>	<none>
rs-web-d2jr2	1/1	Terminating	0	20m	10.233.103.7	worker2	<none>	<none>
rs-web-pdz7f	1/1	Terminating	0	4m55s	10.233.103.8	worker2	<none>	<none>
rs-web-qx9wb	1/1	Terminating	0	4m55s	10.233.110.5	worker1	<none>	<none>
rs-web-zd98q	1/1	Terminating	0	36m	10.233.103.5	worker2	<none>	<none>

ReplicaSet Delete (--cascade=orphan)

ReplicaSet은 삭제하지만, 이미 생성된 Pod는 유지하고 싶다면?

ReplicaSet 다시 생성하고 삭제를 진행해보자.

```
remote > kubectl create -f rs-web.yaml
replicaset.apps/rs-web created
```

```
remote > kubectl get replicaset -o wide
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
rs-web	3	3	3	6s	node-web	whatwant/node-web:1.0	app=web

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-chqwm	1/1	Running	0	14s	10.233.103.9	worker2	<none>	<none>
rs-web-jgc9s	1/1	Running	0	14s	10.233.110.6	worker1	<none>	<none>
rs-web-txlnk	1/1	Running	0	14s	10.233.103.10	worker2	<none>	<none>

```
remote > kubectl delete replicaset rs-web --cascade=orphan
replicaset.apps "rs-web" deleted
```

```
remote > kubectl get replicaset -o wide
```

No resources found in default namespace.

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rs-web-chqwm	1/1	Running	0	49s	10.233.103.9	worker2	<none>	<none>
rs-web-jgc9s	1/1	Running	0	49s	10.233.110.6	worker1	<none>	<none>
rs-web-txlnk	1/1	Running	0	49s	10.233.103.10	worker2	<none>	<none>



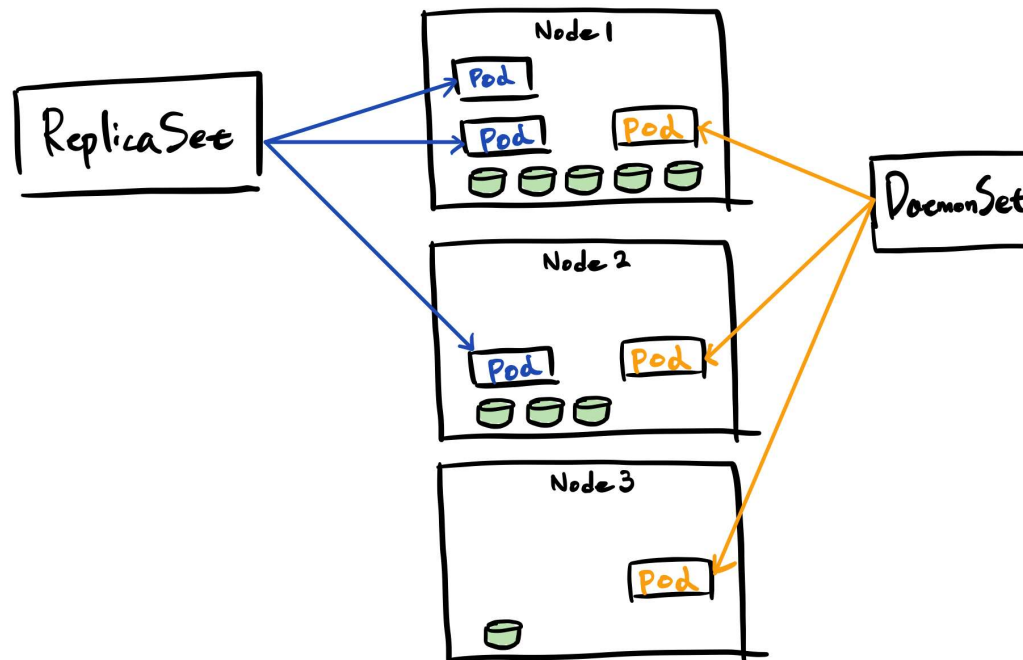
Kubernetes

DaemonSet

DaemonSet is ...

- DaemonSet을 활용하면 cluster의 모든 node에, **node당 하나의 Pod**를 배포할 수 있음
 - . 시스템 수준의 작업을 수행하는 인프라 관련 Pod (로깅, 모니터링), Kube-proxy도 DaemonSet의 일종
- 특정 node에만 Pod를 배포하려면 nodeSelector 속성 지정

▷ DaemonSet



※ 참고 : <https://zunoxi.github.io/devops/2020/11/07/devops-k8s-daemonset/>



K8s : DaemonSet Hands-On

Daemon Create

YAML 파일 작성해서 DaemonSet 생성도 하고 삭제도 해보자 !

ds-web.yaml

```
apiVersion: apps/v1
kind: DaemonSet

metadata:
  name: ds-web

spec:
  selector:
    matchLabels:
      app: ds-web

  template:
    metadata:
      labels:
        app: ds-web

    spec:
      containers:
        - name: node-web
          image: whatwant/node-web:1.0
          ports:
            - containerPort: 8080
```

```
remote > cd kubernetes/03-RS-DS-JOB-CJ/hands-on
```

```
remote > kubectl create -f ./ds-web.yaml
```

```
daemonset.apps/ds-web created
```

```
> kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
ds-web-8rgbq	1/1	Running	0	70s	10.233.110.9	worker1	<none>	<none>
ds-web-kg84c	1/1	Running	0	70s	10.233.103.13	worker2	<none>	<none>

```
remote > kubectl get daemonsets -o wide
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE	CONTAINERS	IMAGES	SELECTOR
ds-web	2	2	2	2	2	<none>	2m10s	node-web	whatwant/node-web:1.0	app=ds-web

Daemon Delete

지우는 방식은 계속 유사하다 ...

```
remote > kubectl get daemonsets -o wide
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE	CONTAINERS	IMAGES	SELECTOR
ds-web	2	2	2	2	2	<none>	2m10s	node-web	whatwant/node-web:1.0	app=ds-web

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
ds-web-8rgbq	1/1	Running	0	3m12s	10.233.110.9	worker1	<none>	<none>
ds-web-kg84c	1/1	Running	0	3m12s	10.233.103.13	worker2	<none>	<none>

```
remote > kubectl delete daemonset ds-web
```

```
daemonset.apps "ds-web" deleted
```

```
remote > kubectl get daemonsets -o wide
```

```
No resources found in default namespace.
```

```
remote > kubectl get pods -o wide
```

```
No resources found in default namespace.
```

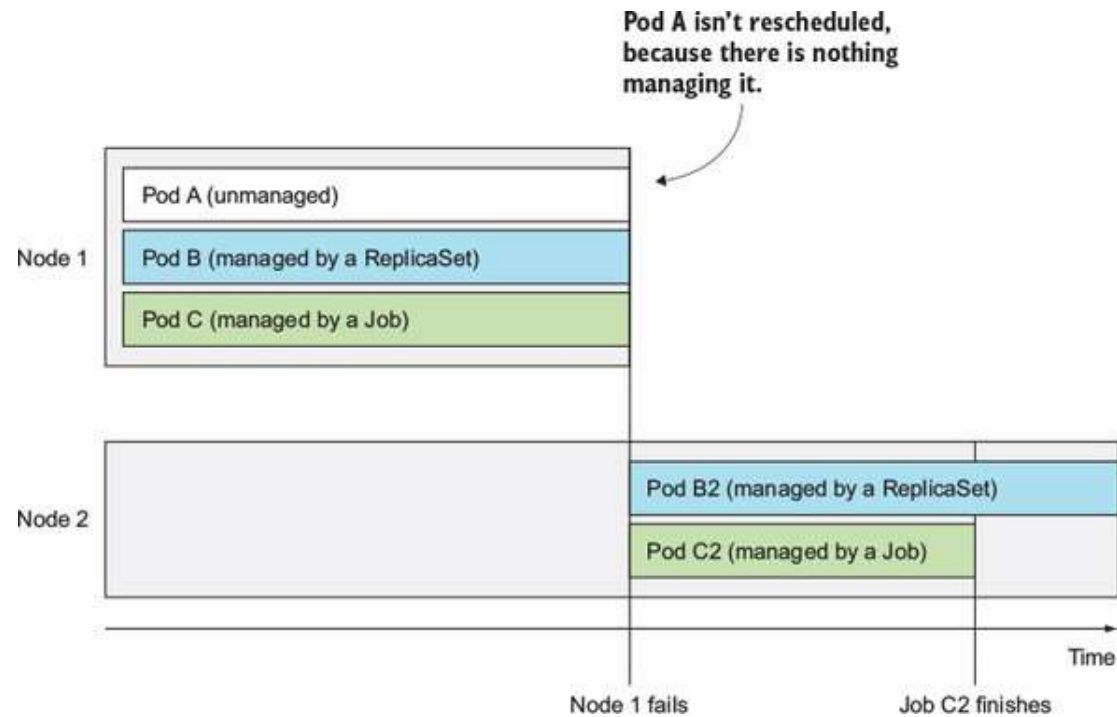


Kubernetes

Job

Job is ...

- 완료 가능한 단일 태스크를 수행하는 Pod를 실행
 - . 프로세스 종료 이후에도 다시 실행되지 않음
 - . node 장애 발생 시 다시 실행할지 말지 여부를 결정할 수 있음



※ 참고 : <https://livebook.manning.com/book/kubernetes-in-action/chapter-4/321>

Job is ... (advanced)

- 고정적(fixed)인 완료 횟수 를 가진 병렬 Job

.spec.completions 에 0이 아닌 양수 값을 지정

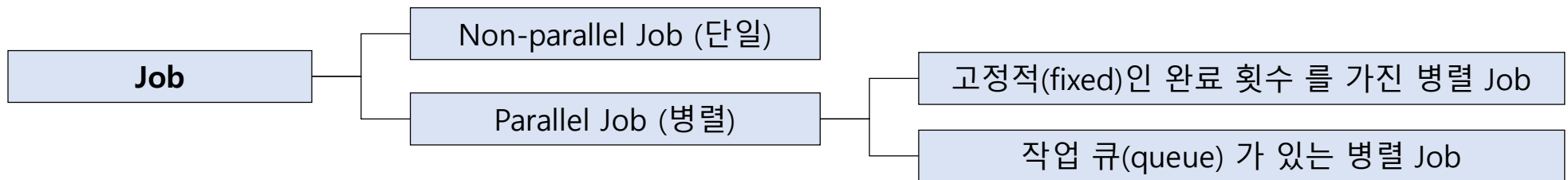
. Job은 전체 작업을 나타내며 1에서 .spec.completions 까지의 범위의 각 값에 대해 한 개씩 성공한 Pod가 있으면 완료

- 작업 큐(queue) 가 있는 병렬 Job

.spec.completions 를 지정하지 않고, .spec.parallelism 에 양수 값 설정

. Job의 모든 Pod가 성공적으로 종료되면, 새로운 Pod는 생성되지 않음

. 하나 이상의 Pod가 성공적으로 종료되고, 모든 Pod가 종료되면 Job은 성공적으로 완료



※ 참고 : <https://kubernetes.io/ko/docs/concepts/workloads/controllers/job/>



K8s : Job Hands-On

Job YAML

Container 에서 command 입력하는 방법과, Job YAML 문법에 대해서 먼저 살펴보자

job-pi.yaml

```
apiVersion: batch/v1
kind: Job

metadata:
  name: job-pi

spec:
  template:
    spec:
      containers:
        - name: pi          ※ 최신 버전에서는 에러 발생
          image: perl:5.34.1
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]

          restartPolicy: Never

          backoffLimit: 4
```

→ 파이 값을 계산하는 명령어이며, 평균 10초 정도 소요

※ 참고 : <https://kubernetes.io/ko/docs/concepts/workloads/controllers/job/>

※ 참고 : <https://kubernetes.io/ko/docs/concepts/workloads/pods/pod-lifecycle/>

※ 참고 : <https://kubernetes.io/ko/docs/tasks/inject-data-application/define-command-argument-container/>

▷ 파드 안에서 동작하는 컨테이너를 위한 **커맨드**와 **인자** 사용 방법

설명	도커 필드 이름	쿠버네티스 필드 이름
컨테이너에서 실행되는 커맨드	Entrypoint	command
커맨드에 전달되는 인자들	Cmd	arg

▷ **restartPolicy** (Pod Lifecycle)

- Always : 항상 재시작
- OnFailure : 비정상 종료 발생 시 재시작
- Never : 재시작 하지 않음

▷ **BackoffLimit** (백오프 정책)

- 실패할 경우 다시 실행시킬 재시도 횟수 지정
- default 값 = 6
- 실패 후 10초, 20초, 40초 간격으로 재시도

Job Create

Job은 앞에서 살펴본 다른 리소스와 차이가 있다.

```
remote > cd kubernetes/03-RS-DS-JOB-CJ/hands-on
```

```
remote > kubectl create -f job-pi.yaml
```

```
job.batch/pi created
```

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
job-pi-5dh5h	0/1	Completed	0	24s	10.233.103.19	worker2	<none>		<none>	

```
remote > kubectl get jobs -o wide
```

NAME	COMPLETIONS	DURATION	AGE	CONTAINERS	IMAGES	SELECTOR
job-pi	1/1	15s	111s	pi	perl:5.34.1	controller-uid=ab042fa9-cdc0-473f-8925-6713b7231674

```
remote > kubectl logs job-pi-5dh5h
```

```
3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128
48111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024...
```

일단 STATUS가 `Completed`이고, 그렇기에 READY 값이 `0/1`로 나오게 된다. 즉, 프로세스가 종료된 것이다.

또한, Jobs의 정보의 형태도 앞에서의 다른 리소스와는 다른 모습을 보인다.

Job Delete

지우는 방식은 계속 유사하다 ...

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
job-pi-5dh5h	0/1	Completed	0	24s	10.233.103.19	worker2	<none>	<none>

```
remote > kubectl get jobs -o wide
```

NAME	COMPLETIONS	DURATION	AGE	CONTAINERS	IMAGES	SELECTOR
job-pi	1/1	15s	111s	pi	perl:5.34.1	controller-uid=ab042fa9-cdc0-473f-8925-6713b7231674

```
remote > kubectl delete jobs job-pi
```

```
job.batch "job-pi" deleted
```

```
remote > kubectl get jobs -o wide
```

```
No resources found in default namespace.
```

```
remote > kubectl get pods -o wide
```

```
No resources found in default namespace.
```



Problem

**특정 작업이 3번 성공
되어야 할 때**

throw-dice - 1/2

success / failure 결과가 나오는 container를 하나 골라봤다.

job-throw-dice.yaml

```
apiVersion: batch/v1
kind: Job

metadata:
  name: job-throw-dice

spec:
  completions: 3
  parallelism: 3
  backoffLimit: 25

  template:
    spec:
      containers:
        - name: throw-dice
          image: kodekloud/throw-dice

      restartPolicy: Never
```

- ▷ **completions** : 몇 번의 Completed가 나올 때까지 (성공 발생 횟수)
- ▷ **parallelism** : 한 번에 몇 개까지 실행할 것인지
- ▷ **backoffLimit** : 최대 실행 횟수 (에러 발생 횟수)

‘6’이 나올 때만 **success**, 나머지는 **failure**

throw-dice - 2/2

앞에서 설명한 옵션들이 어떻게 작용하는지 잘 살펴보자

```
remote > cd kubernetes/03-RS-DS-JOB-CJ/hands-on
```

```
remote > kubectl create -f job-throw-dice.yaml
```

```
job.batch/job-throw-dice created
```

```
remote > kubectl get jobs -o wide -w
```

NAME	COMPLETIONS	DURATION	AGE	CONTAINERS	IMAGES	SELECTOR
job-throw-dice	0/3		0s	throw-dice	kodekloud/throw-dice	controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
job-throw-dice	0/3	0s	0s	throw-dice	kodekloud/throw-dice	controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
job-throw-dice	0/3	5s	5s	throw-dice	kodekloud/throw-dice	controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
job-throw-dice	1/3	5s	5s	throw-dice	kodekloud/throw-dice	controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
job-throw-dice	1/3	7s	7s	throw-dice	kodekloud/throw-dice	controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
job-throw-dice	1/3	15s	15s	throw-dice	kodekloud/throw-dice	controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
...						

```
remote > kubectl get pods -o wide -w
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
throw-dice-job-4ffd2	0/1	Pending	0	0s	<none>	<none>	<none>		<none>	
throw-dice-job-4ffd2	0/1	Pending	0	0s	<none>	worker2	<none>		<none>	
throw-dice-job-5k7gj	0/1	Pending	0	0s	<none>	<none>	<none>		<none>	
throw-dice-job-vmbst	0/1	Pending	0	0s	<none>	<none>	<none>		<none>	
throw-dice-job-5k7gj	0/1	Pending	0	0s	<none>	worker2	<none>		<none>	
throw-dice-job-vmbst	0/1	Pending	0	0s	<none>	worker1	<none>		<none>	
throw-dice-job-vmbst	0/1	ContainerCreating	0	0s	<none>	worker1	<none>		<none>	
throw-dice-job-4ffd2	0/1	ContainerCreating	0	0s	<none>	worker2	<none>		<none>	
throw-dice-job-5k7gj	0/1	ContainerCreating	0	0s	<none>	worker2	<none>		<none>	
throw-dice-job-vmbst	0/1	ContainerCreating	0	1s	<none>	worker1	<none>		<none>	



Kubernetes

CronJob

CronJob is ...

- 특정 시간 또는 지정된 간격으로 Job을 반복 실행

CRONJOB

- An extension of the Job Controller, it provides a method of executing jobs on a cron-like schedule.
- CronJobs within Kubernetes use **UTC ONLY**.



※ 참고 : <https://www.slideshare.net/RonnyTrommer/devjam-2019-introduction-to-kubernetes>



K8s : CronJob Hands-On

CronJob YAML

Job을 주기적으로 실행하기 위한 리소스 이다. 한 번 해보자!

cj-pi.yaml

```
apiVersion: batch/v1
kind: CronJob

metadata:
  name: cj-every-fifteen-minutes

spec:
  schedule: "0,15,30,45 * * * *"

  jobTemplate:
    spec:
      backoffLimit: 3

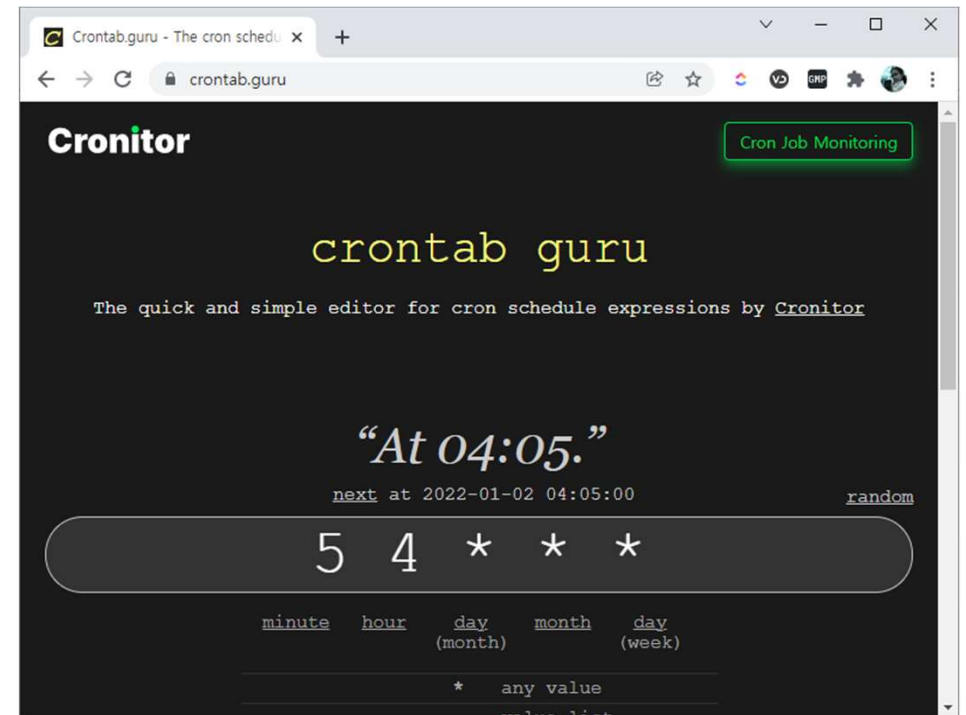
      template:
        metadata:
          labels:
            app: periodic-batch-job

        spec:
          containers:
            - name: pi
              image: perl:5.34.1
              command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]

          restartPolicy: Never
```

※ 참고 : <https://kubernetes.io/ko/docs/tasks/job/automated-tasks-with-cron-jobs/>

▷ schedule 설정은 아래 사이트에서 편하게 확인해볼 수 있다
- <https://crontab.guru/>



CronJob Create

CronJob은 앞에서 살펴본 다른 리소스와 결과 확인이 조금 색다르다.

```
remote > cd kubernetes/03-RS-DS-JOB-CJ/hands-on
```

```
remote > kubectl create -f cj-pi.yaml
```

```
cronjob.batch/batch-job-every-fifteen-minutes created
```

```
remote > kubectl get cronjobs -o wide
```

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE	CONTAINERS	IMAGES	SELECTOR
batch-job-every-fifteen-minutes	0,15,30,45 * * * *	False	0	<none>	9s	pi	perl	<none>

```
remote > kubectl get pods -o wide
```

```
No resources found in default namespace.
```

CronJob Watch

CronJob은 시간이 되면 ACTIVE가 되었다가, 끝나면 다시 inACTIVE가 된다.

```
remote > kubectl get cronjobs -o wide -w
```

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE	CONTAINERS	IMAGES	SELECTOR
cj-every-fifteen-minutes	0,15,30,45 * * * *	False	0	<none>	0s	pi	perl:5.34.1	<none>
cj-every-fifteen-minutes	0,15,30,45 * * * *	False	1	0s	14m	pi	perl:5.34.1	<none>
cj-every-fifteen-minutes	0,15,30,45 * * * *	False	0	7s	14m	pi	perl:5.34.1	<none>

^C%

CronJob은 시간이 안되었을 때엔, Pod가 아예 보이지 않다가, 때가 되면 생성되고, 정해진 일을 수행한다.

```
remote > kubectl get pods -o wide -w
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
cj-every-fifteen-minutes-27652320-xjhwh	0/1	Pending	0	0s	<none>	<none>	<none>	<none>
cj-every-fifteen-minutes-27652320-xjhwh	0/1	Pending	0	0s	<none>	worker2	<none>	<none>
cj-every-fifteen-minutes-27652320-xjhwh	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-every-fifteen-minutes-27652320-xjhwh	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-every-fifteen-minutes-27652320-xjhwh	1/1	Running	0	1s	10.233.103.46	worker2	<none>	<none>
cj-every-fifteen-minutes-27652320-xjhwh	0/1	Completed	0	5s	10.233.103.46	worker2	<none>	<none>
cj-every-fifteen-minutes-27652320-xjhwh	0/1	Completed	0	6s	10.233.103.46	worker2	<none>	<none>

^C%

일을 마친 Pod는 'Completed'인 상태로 누적되어 보인다.

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
cj-every-fifteen-minutes-27652320-xjhwh	0/1	Completed	0	4m30s	10.233.103.46	worker2	<none>	<none>

CronJob Delete

지우는 방식은 계속 유사하다 ...

```
remote > kubectl get cronjobs -o wide
```

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE	CONTAINERS	IMAGES	SELECTOR
cj-every-fifteen-minutes	0,15,30,45 * * * *	False	0	5m3s	19m	pi	perl:5.34.1	<none>

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
cj-every-fifteen-minutes-27652320-xjhwh	0/1	Completed	0	5m20s	10.233.103.46	worker2	<none>	<none>

```
remote > kubectl delete cronjob cj-every-fifteen-minutes
```

```
cronjob.batch "cj-every-fifteen-minutes" deleted
```

```
remote > kubectl get cronjobs -o wide
```

```
No resources found in default namespace.
```

```
remote > kubectl get pods -o wide
```

```
No resources found in default namespace.
```

CronJob History - 1/4

cj-history-3.yaml

```
apiVersion: batch/v1
kind: CronJob

metadata:
  name: cj-history-3

spec:
  schedule: "*/1 * * * *"

  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1

  jobTemplate:

    spec:
      backoffLimit: 3
      template:
        metadata:
          labels:
            app: history-3

        spec:
          containers:
            - name: pi
              image: perl:5.34.1
              command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
              restartPolicy: Never
```

cj-history-0.yaml

```
apiVersion: batch/v1
kind: CronJob

metadata:
  name: cj-history-0

spec:
  schedule: "*/1 * * * *"

  successfulJobsHistoryLimit: 0
  failedJobsHistoryLimit: 0

  jobTemplate:

    spec:
      backoffLimit: 3
      template:
        metadata:
          labels:
            app: history-0

        spec:
          containers:
            - name: pi
              image: perl:5.34.1
              command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
              restartPolicy: Never
```


CronJob History - 2/4

2개의 CronJob을 생성해보자 !

```
remote > cd kubernetes/03-RS-DS-JOB-CJ/hands-on
```

```
remote > kubectl create -f cj-history-3.yaml
```

```
cronjob.batch/cj-history-3 created
```

```
remote > kubectl create -f cj-history-0.yaml
```

```
cronjob.batch/cj-history-0 created
```

1분마다 `0-history` & `3-history` cronjob이 실행되었다가 종료된다

```
remote > kubectl get cronjobs -o wide -w
```

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE	CONTAINERS	IMAGES	SELECTOR
cj-history-0	*/1 * * * *	False	0	<none>	12s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	0	<none>	15s	pi	perl:5.34.1	<none>
cj-history-0	*/1 * * * *	False	1	0s	34s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	1	0s	37s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	0	7s	44s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	0	7s	44s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	1	0s	97s	pi	perl:5.34.1	<none>
cj-history-0	*/1 * * * *	False	2	0s	94s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	0	11s	108s	pi	perl:5.34.1	<none>
cj-history-3	*/1 * * * *	False	0	11s	108s	pi	perl:5.34.1	<none>
cj-history-0	*/1 * * * *	False	1	11s	105s	pi	perl:5.34.1	<none>

^C%

CronJob History - 3/4

Pod들이 생성되고 완료되고, 삭제(Terminating) 되는 모습을 살펴볼 수 있다.

```
remote > kubectl get pods -o wide -w
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
cj-history-3-27652331-f5qjb	0/1	Pending	0	0s	<none>	<none>	<none>	<none>
cj-history-0-27652331-cz7wr	0/1	Pending	0	0s	<none>	<none>	<none>	<none>
cj-history-3-27652331-f5qjb	0/1	Pending	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652331-cz7wr	0/1	Pending	0	0s	<none>	worker1	<none>	<none>
cj-history-3-27652331-f5qjb	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652331-cz7wr	0/1	ContainerCreating	0	0s	<none>	worker1	<none>	<none>
cj-history-3-27652331-f5qjb	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652331-cz7wr	0/1	ContainerCreating	0	0s	<none>	worker1	<none>	<none>
cj-history-3-27652331-f5qjb	1/1	Running	0	1s	10.233.103.47	worker2	<none>	<none>
cj-history-3-27652331-f5qjb	0/1	Completed	0	5s	10.233.103.47	worker2	<none>	<none>
cj-history-3-27652331-f5qjb	0/1	Completed	0	6s	10.233.103.47	worker2	<none>	<none>
cj-history-3-27652331-f5qjb	0/1	Completed	0	7s	10.233.103.47	worker2	<none>	<none>
cj-history-3-27652332-4gsnj	0/1	Pending	0	0s	<none>	<none>	<none>	<none>
cj-history-3-27652332-4gsnj	0/1	Pending	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652332-q66bj	0/1	Pending	0	0s	<none>	<none>	<none>	<none>
cj-history-3-27652332-4gsnj	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652332-q66bj	0/1	Pending	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652332-q66bj	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-history-3-27652332-4gsnj	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-history-0-27652332-q66bj	0/1	ContainerCreating	0	0s	<none>	worker2	<none>	<none>
cj-history-3-27652332-4gsnj	1/1	Running	0	1s	10.233.103.48	worker2	<none>	<none>
cj-history-0-27652332-q66bj	1/1	Running	0	1s	10.233.103.49	worker2	<none>	<none>
cj-history-3-27652332-4gsnj	0/1	Completed	0	9s	10.233.103.48	worker2	<none>	<none>
cj-history-0-27652332-q66bj	0/1	Completed	0	9s	10.233.103.49	worker2	<none>	<none>
cj-history-3-27652332-4gsnj	0/1	Completed	0	10s	10.233.103.48	worker2	<none>	<none>

```
^C%
```

CronJob History - 4/4

pods 현황을 계속 찍어보면, history를 몇 개씩 유지하고 있는지 살펴볼 수 있다.

```
remote > kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
cronjob-history-0-1641034260-z8s2h	0/1	ContainerCreating	0	0s	<none>	worker2	<none>		<none>	
cronjob-history-3-1641034140-nmdfp	0/1	Completed	0	2m	10.233.103.96	worker2	<none>		<none>	
cronjob-history-3-1641034200-m6df8	0/1	Completed	0	60s	10.233.103.98	worker2	<none>		<none>	
cronjob-history-3-1641034260-l5hht	0/1	ContainerCreating	0	0s	<none>	worker2	<none>		<none>	

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
cronjob-history-0-1641034260-z8s2h	1/1	Running	0	8s	10.233.103.100	worker2	<none>		<none>	
cronjob-history-3-1641034140-nmdfp	0/1	Completed	0	2m8s	10.233.103.96	worker2	<none>		<none>	
cronjob-history-3-1641034200-m6df8	0/1	Completed	0	68s	10.233.103.98	worker2	<none>		<none>	
cronjob-history-3-1641034260-l5hht	0/1	Completed	0	8s	10.233.103.99	worker2	<none>		<none>	

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
cronjob-history-0-1641034260-z8s2h	0/1	Completed	0	18s	10.233.103.100	worker2	<none>		<none>	
cronjob-history-3-1641034140-nmdfp	0/1	Completed	0	2m18s	10.233.103.96	worker2	<none>		<none>	
cronjob-history-3-1641034200-m6df8	0/1	Completed	0	78s	10.233.103.98	worker2	<none>		<none>	
cronjob-history-3-1641034260-l5hht	0/1	Completed	0	18s	10.233.103.99	worker2	<none>		<none>	

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
cronjob-history-3-1641034140-nmdfp	0/1	Completed	0	2m54s	10.233.103.96	worker2	<none>		<none>	
cronjob-history-3-1641034200-m6df8	0/1	Completed	0	114s	10.233.103.98	worker2	<none>		<none>	
cronjob-history-3-1641034260-l5hht	0/1	Completed	0	54s	10.233.103.99	worker2	<none>		<none>	

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
cronjob-history-3-1641034200-m6df8	0/1	Completed	0	2m34s	10.233.103.98	worker2	<none>		<none>	
cronjob-history-3-1641034260-l5hht	0/1	Completed	0	94s	10.233.103.99	worker2	<none>		<none>	
cronjob-history-3-1641034320-v4rkn	0/1	Completed	0	34s	10.233.103.102	worker2	<none>		<none>	

