3rd Week

세번째 뵙겠습니다 ?!

▷ 출석 체크도 한 번 해보시면 어떠세요?!

- https://modulabs.co.kr/
- 모두연 홈페이지 → 로그인 → 마이페이지 → 참여한 랩·풀잎 → 자세히 보기 → 내 풀잎스쿨 출석 확인하기

▷ Ground Rule

- 가급적 지각/결석 하지 않기
- 가급적 Camera 켜 놓고 수업 참여하기
- 가급적 적극적으로 참여하기
- 3시간이 넘더라도 배고프다고 화내지 않기
- Slack 읽지 않기
- 꼭 끝까지 함께하기

잡담 &
지난 수업 관련 이야기

///

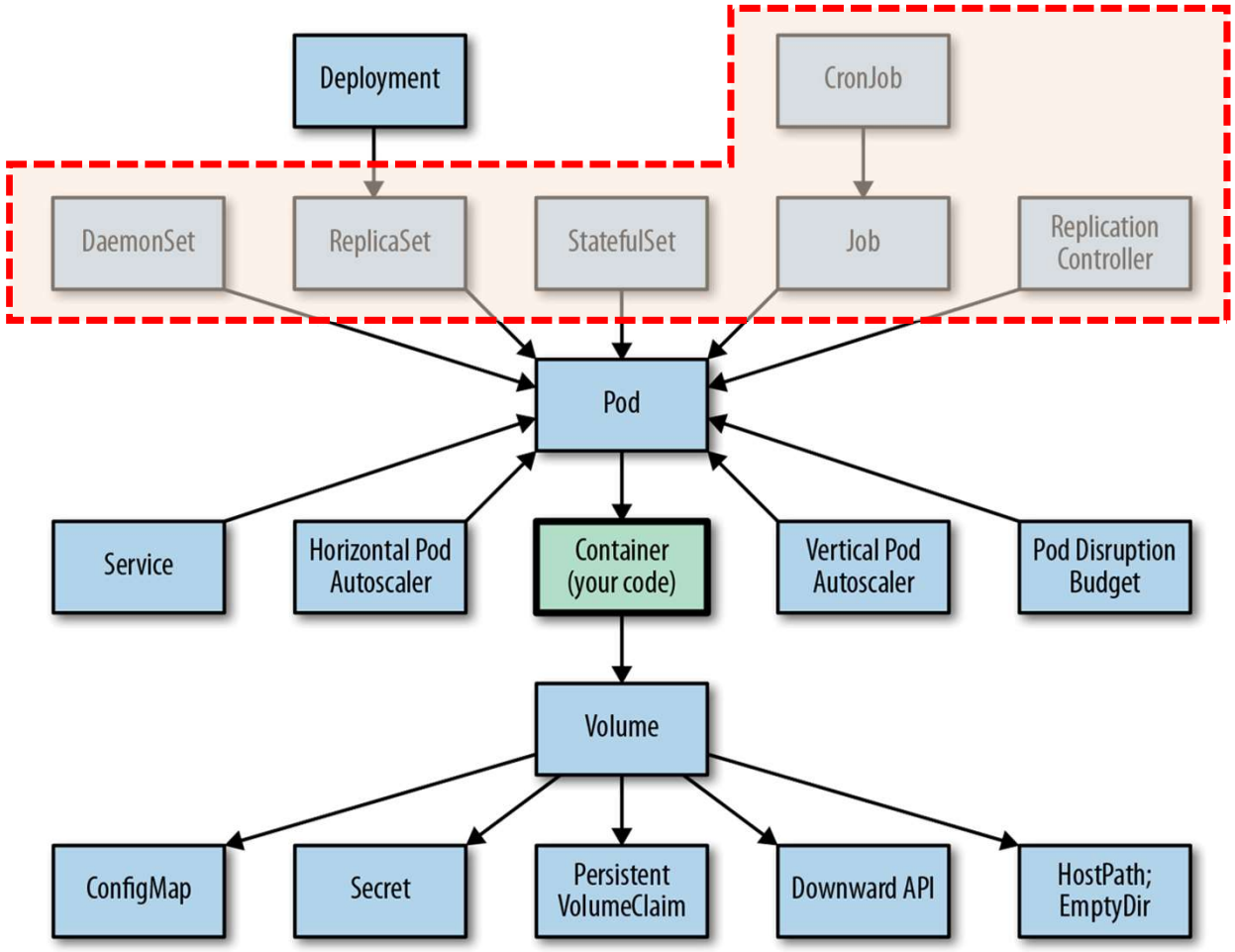
Agenda

- ▷ [10m] make-friendship
- ▷ [30m] Flip - ReplicaSet/DaemonSet/Job/CronJob
- ▷ [30m] Wrap-Up & Hands-On
- ▷ [10m] Break-Time

- ▷ [30m] Flip - Service (ClusterIP/NodePort/ExternalName)
- ▷ [20m] Wrap-Up & Hands-On
- ▷ [10m] Break-Time

- ▷ [20m] Quiz
- ▷ [10m] Q&A

Kubernetes concepts for developers



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

///

Flip Learning

(ReplicaSet/DaemonSet/Job/CronJob)

정현찬님

///

Kubernetes

ReplicaSet

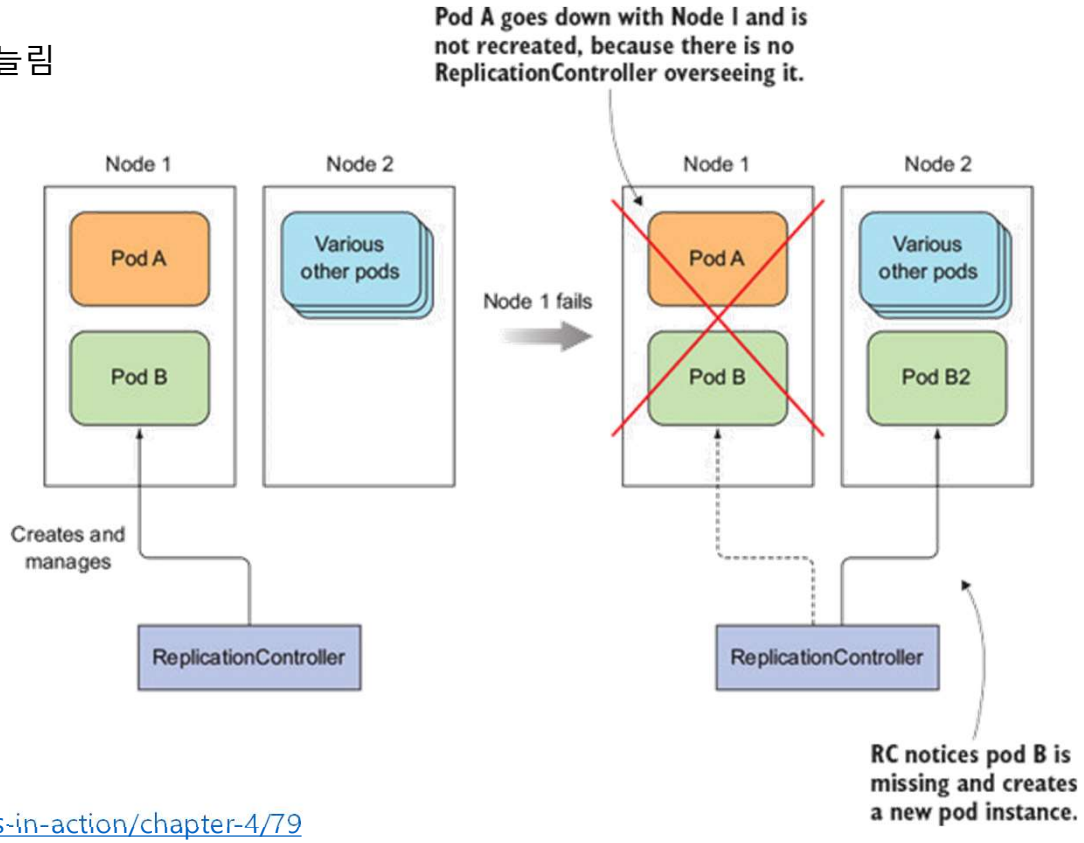
ReplicationController - 1/2

- **Pod**가 항상 **실행**되도록 **보장**하는 리소스

  . 어떤 이유로든 Pod가 사라지면 사라진 Pod를 감지해 교체 Pod를 생성

- 지속적으로 실행 중인 **Pod** 목록을 **모니터링**하고 선언된 수와 **일치**시킴

  . 너무 많은 Pod가 실행 중이면 줄이고, 적으면 늘림



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

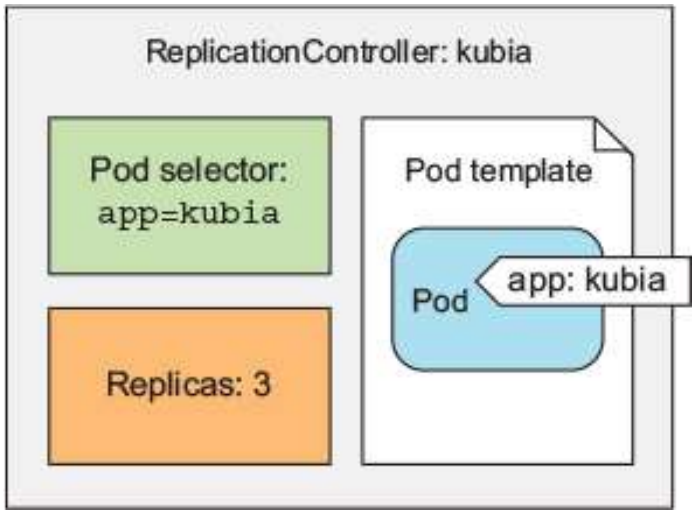
ReplicationController - 2/2

- 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
```

In

NotIn

Exist

DoesNotExist

///

K8s：ReplicaSet Hands-On

ReplicaSet Create

YAML 파일 작성해서 ReplicaSet 실행해 보자!

rs-node-web.yaml

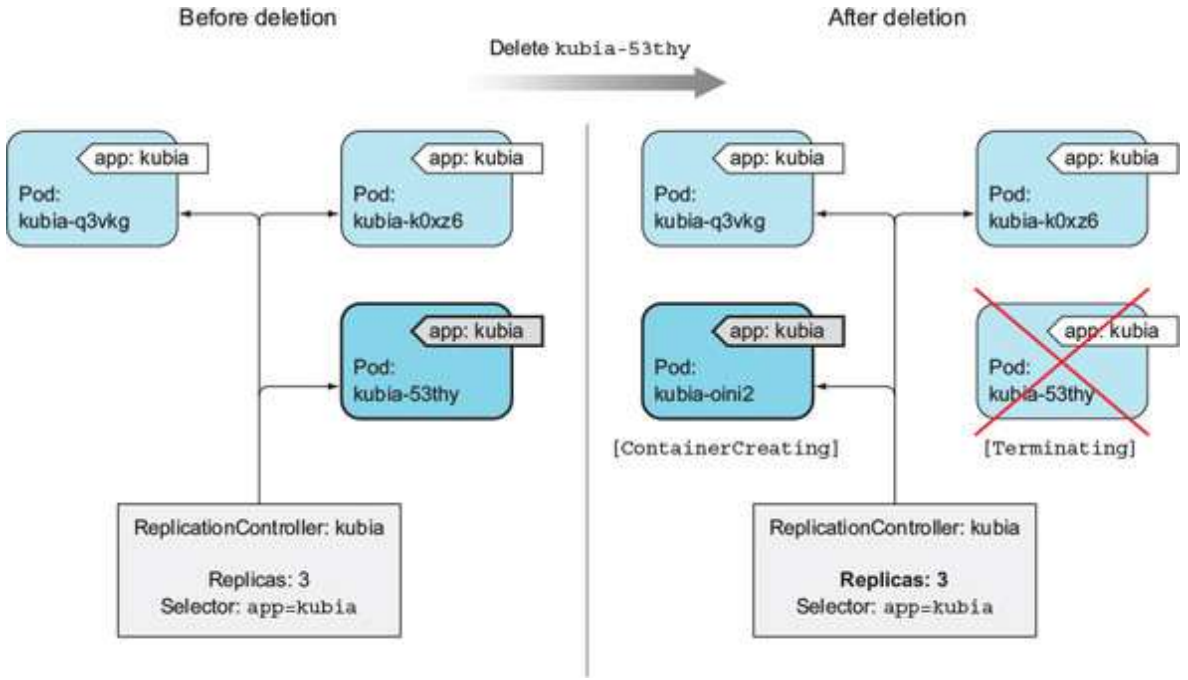
```yaml
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: rs-labels

spec:
  replicas: 3

  selector:
    matchLabels:
      app: node-web

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

```
remote ❯ cd advanced-kubernetes/03-week/ReplicaSet

remote ❯ kubectl create -f ./rs-node-web.yaml

replicaset.apps/rs-labels created


remote ❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-8677p   1/1     Running   0          7s    10.233.110.61   worker1   <none>           <none>
rs-labels-gdj7n   1/1     Running   0          7s    10.233.103.77   worker2   <none>           <none>
rs-labels-z4frf   1/1     Running   0          7s    10.233.103.76   worker2   <none>           <none>


remote ❯ kubectl get replicasets -o wide

NAME        DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES                   SELECTOR
rs-labels   3         3         3       5m6s   node-web     whatwant/node-web:1.0   app=node-web
```

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

```
remote ❯ kubectl delete pod rs-labels-8677p

pod "rs-labels-8677p" deleted
```

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

NAME              READY   STATUS              RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-8677p   1/1     Running             0          68m   10.233.110.61   worker1   <none>           <none>
rs-labels-gdj7n   1/1     Running             0          68m   10.233.103.77   worker2   <none>           <none>
rs-labels-z4frf   1/1     Running             0          68m   10.233.103.76   worker2   <none>           <none>
rs-labels-8677p   1/1     Terminating         0          68m   10.233.110.61   worker1   <none>           <none>
rs-labels-phbs7   0/1     Pending             0          0s    <none>          <none>    <none>           <none>
rs-labels-phbs7   0/1     Pending             0          0s    <none>          worker1   <none>           <none>
rs-labels-phbs7   0/1     ContainerCreating   0          0s    <none>          worker1   <none>           <none>
rs-labels-phbs7   0/1     ContainerCreating   0          1s    <none>          worker1   <none>           <none>
rs-labels-phbs7   1/1     Running             0          1s    10.233.110.62   worker1   <none>           <none>
rs-labels-8677p   1/1     Terminating         0          68m   10.233.110.61   worker1   <none>           <none>
rs-labels-8677p   0/1     Terminating         0          68m   <none>          worker1   <none>           <none>
rs-labels-8677p   0/1     Terminating         0          68m   <none>          worker1   <none>           <none>
rs-labels-8677p   0/1     Terminating         0          68m   <none>          worker1   <none>           <none>
^C%
```

```
remote ❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE     IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-gdj7n   1/1     Running   0          70m     10.233.103.77   worker2   <none>           <none>
rs-labels-phbs7   1/1     Running   0          2m16s   10.233.110.62   worker1   <none>           <none>
rs-labels-z4frf   1/1     Running   0          70m     10.233.103.76   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 edit replicaset rs-labels
```

```
# 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: "2021-12-31T12:51:16Z"
  generation: 1
  name: rs-labels
  namespace: default
  resourceVersion: "1472985"
  uid: 6c6ce66f-e894-4177-b2d2-64f3d1f6bd96
spec:
  replicas: 5
  selector:
    matchLabels:
      app: node-web
...
```

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

replicas 숫자를 변경하고 싶으면 어떻게 해야 할까?

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

NAME              READY   STATUS              RESTARTS   AGE    IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-gdj7n   1/1     Running             0          160m   10.233.103.77   worker2   <none>           <none>
rs-labels-phbs7   1/1     Running             0          92m    10.233.110.62   worker1   <none>           <none>
rs-labels-z4frf   1/1     Running             0          160m   10.233.103.76   worker2   <none>           <none>
rs-labels-dxjvz   0/1     Pending             0          0s     <none>          <none>    <none>           <none>
rs-labels-zq25x   0/1     Pending             0          0s     <none>          <none>    <none>           <none>
rs-labels-dxjvz   0/1     Pending             0          0s     <none>          worker1   <none>           <none>
rs-labels-zq25x   0/1     Pending             0          0s     <none>          worker2   <none>           <none>
rs-labels-zq25x   0/1     ContainerCreating   0          0s     <none>          worker2   <none>           <none>
rs-labels-dxjvz   0/1     ContainerCreating   0          0s     <none>          worker1   <none>           <none>
rs-labels-zq25x   0/1     ContainerCreating   0          0s     <none>          worker2   <none>           <none>
rs-labels-dxjvz   0/1     ContainerCreating   0          0s     <none>          worker1   <none>           <none>
rs-labels-zq25x   1/1     Running             0          1s     10.233.103.78   worker2   <none>           <none>
rs-labels-dxjvz   1/1     Running             0          1s     10.233.110.63   worker1   <none>           <none>
^C%
```

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-labels-dxjvz   1/1     Running   0          2m48s  10.233.110.63   worker1   <none>           <none>
rs-labels-gdj7n   1/1     Running   0          163m   10.233.103.77   worker2   <none>           <none>
rs-labels-phbs7   1/1     Running   0          94m    10.233.110.62   worker1   <none>           <none>
rs-labels-z4frf   1/1     Running   0          163m   10.233.103.76   worker2   <none>           <none>
rs-labels-zq25x   1/1     Running   0          2m48s  10.233.103.78   worker2   <none>           <none>
```

Quiz

ReplicaSet을 삭제해도 Pod는 유지가 될까?

ReplicaSet Delete

```
remote ❯ kubectl get replicasets -o wide

NAME        DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES                   SELECTOR
rs-labels   5         5         5       27s   node-web     whatwant/node-web:1.0    app=node-web


remote ❯ kubectl delete replicaset rs-labels

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

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

NAME              READY   STATUS        RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-68jpd   1/1     Running       0          24s   10.233.103.81   worker2   <none>           <none>
rs-labels-9gbf7   1/1     Running       0          24s   10.233.103.80   worker2   <none>           <none>
rs-labels-mq65n   1/1     Running       0          39s   10.233.110.65   worker1   <none>           <none>
rs-labels-vl8gw   1/1     Running       0          39s   10.233.110.64   worker1   <none>           <none>
rs-labels-zs9h2   1/1     Running       0          39s   10.233.103.79   worker2   <none>           <none>
rs-labels-zs9h2   1/1     Terminating   0          66s   10.233.103.79   worker2   <none>           <none>
rs-labels-9gbf7   1/1     Terminating   0          51s   10.233.103.80   worker2   <none>           <none>
rs-labels-vl8gw   1/1     Terminating   0          66s   10.233.110.64   worker1   <none>           <none>
rs-labels-mq65n   1/1     Terminating   0          66s   10.233.110.65   worker1   <none>           <none>
rs-labels-68jpd   1/1     Terminating   0          51s   10.233.103.81   worker2   <none>           <none>
^C%
```

ReplicaSet Delete (--cascade=orphan)

ReplicaSet은 삭제하지만, 이미 생성된 Pod는 유지하고 싶다면?

ReplicaSet 다시 생성하고 삭제를 진행해보자.

```
remote ❯ kubectl create -f ./rs-node-web.yaml
replicaset.apps/rs-labels created

remote ❯ kubectl get replicasets -o wide
NAME        DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES                   SELECTOR
rs-labels   3         3         3       9s    node-web     whatwant/node-web:1.0    app=node-web

remote ❯ kubectl get pods -o wide
NAME              READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-29r2b   1/1     Running   0          12s   10.233.103.85   worker2   <none>           <none>
rs-labels-747l6   1/1     Running   0          12s   10.233.103.84   worker2   <none>           <none>
rs-labels-xtjbz   1/1     Running   0          12s   10.233.110.67   worker1   <none>           <none>

remote ❯ kubectl delete replicaset rs-labels --cascade=orphan
replicaset.apps "rs-labels" deleted

remote ❯ kubectl get replicasets -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-labels-29r2b   1/1     Running   0          94s   10.233.103.85   worker2   <none>           <none>
rs-labels-747l6   1/1     Running   0          94s   10.233.103.84   worker2   <none>           <none>
rs-labels-xtjbz   1/1     Running   0          94s   10.233.110.67   worker1   <none>           <none>
```

///

Kubernetes

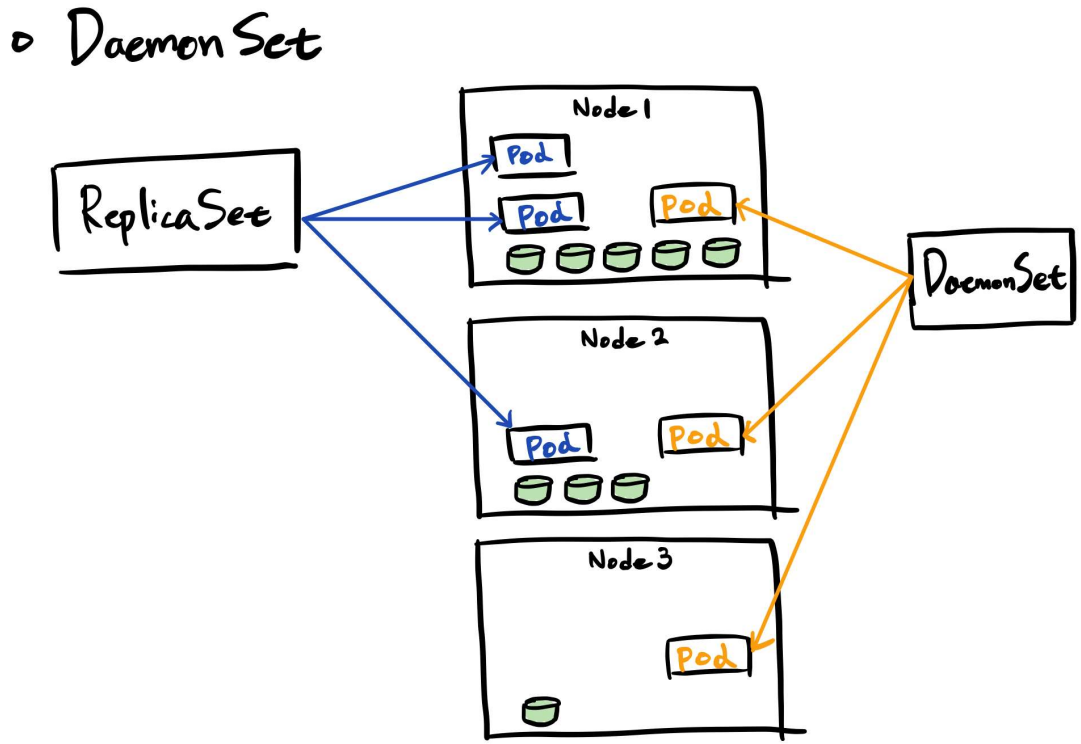
DaemonSet

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



※ 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-node-web.yaml

```yaml
apiVersion: apps/v1
kind: DaemonSet

metadata:
  name: ds-labels

spec:
  selector:
    matchLabels:
      app: node-web

  template:
    metadata:
      labels:
        app: node-web

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

```
remote ❯ cd advanced-kubernetes/03-week/DaemonSet

remote ❯ kubectl create -f ./ds-node-web.yaml

daemonset.apps/ds-labels created


❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
ds-labels-4gtfx   1/1     Running   0          8s    10.233.103.86   worker2   <none>           <none>
ds-labels-kljvl   1/1     Running   0          8s    10.233.110.68   worker1   <none>           <none>
```

```
remote ❯ kubectl get daemonsets -o wide

NAME        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE   CONTAINERS   IMAGES                   SELECTOR
ds-labels   2         2         2       2            2           <none>          11s   node-web     whatwant/node-web:1.0    app=node-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-labels   2         2         2       2            2           <none>          11s   node-web     whatwant/node-web:1.0    app=node-web


remote ❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
ds-labels-f8dld   1/1     Running   0          46s   10.233.110.69   worker1   <none>           <none>
ds-labels-mspkq   1/1     Running   0          46s   10.233.103.87   worker2   <none>           <none>


remote ❯ kubectl delete daemonset ds-labels

daemonset.apps "ds-labels" 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

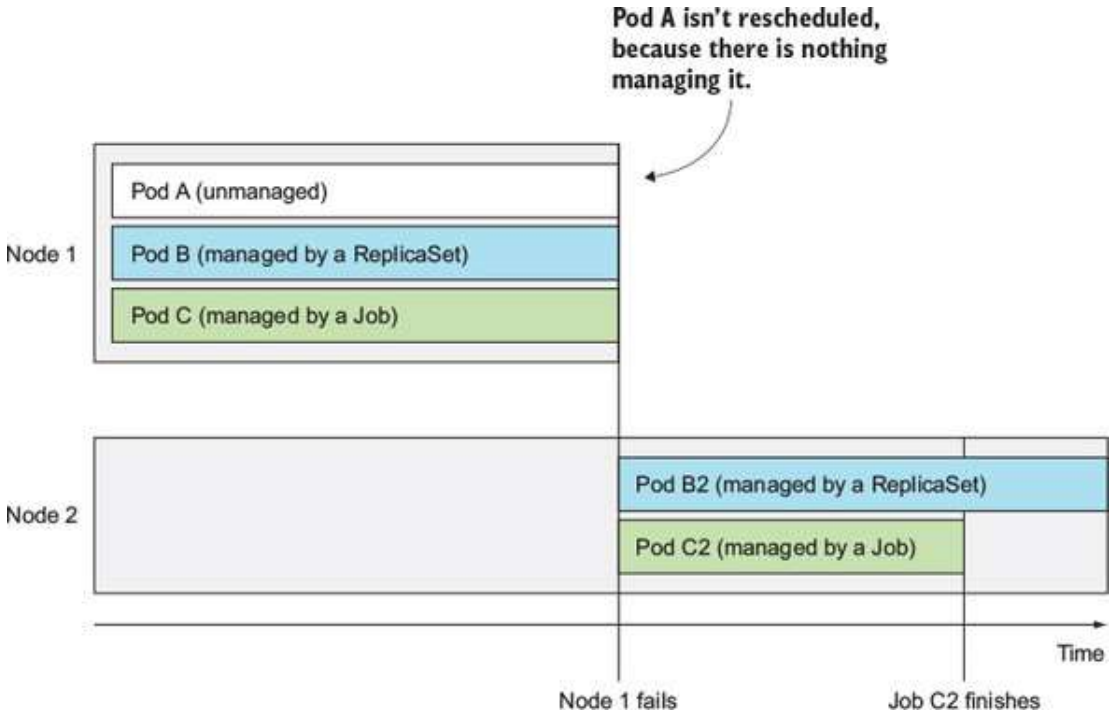
Job

Job is ...

- **완료 가능한 단일 태스크**를 수행하는 Pod를 실행

  . 프로세스 종료 이후에도 다시 실행되지 않음

  . node 장애 발생 時 다시 실행할지 말지 여부를 결정할 수 있음



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

Job is ... (detailed)

- 고정적(fixed)인 완료 횟수 를 가진 병렬 Job

  . .spec.completions 에 0이 아닌 양수 값을 지정

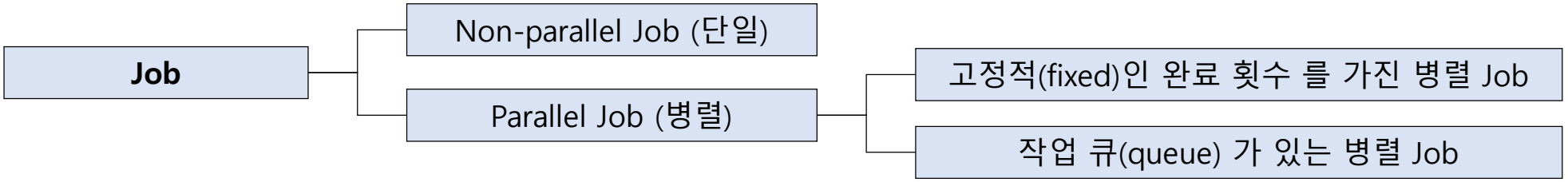
  . Job은 전체 작업을 나타내며 1에서 .spec.completions 까지의 범위의 각 값에 대해 한 개씩 성공한 Pod가 있으면 완료

- 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

```yaml
apiVersion: batch/v1
kind: Job

metadata:
  name: pi

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

      restartPolicy: Never

  backoffLimit: 4
```

→ 파이 값을 계산하는 명령어 이며, 평균 10초 정도 소요

▷ 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 時 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

※ 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/

Job Create

Job은 앞에서 살펴본 다른 리소스와 차이가 있다.

```
remote ❯ cd advanced-kubernetes/03-week/Job

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
pi-x94l9   0/1     Completed   0          33s   10.233.103.88   worker2   <none>           <none>


remote ❯ kubectl get jobs -o wide

NAME   COMPLETIONS   DURATION   AGE   CONTAINERS   IMAGES   SELECTOR
pi     1/1           24s        47s   pi           perl     controller-uid=c4cbbbfc-894b-4a16-97f3-178ed0eab10a
```

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 jobs -o wide

NAME   COMPLETIONS   DURATION   AGE    CONTAINERS   IMAGES   SELECTOR
pi     1/1           24s        7m8s   pi           perl     controller-uid=c4cbbbfc-894b-4a16-97f3-178ed0eab10a


remote ❯ kubectl get pods -o wide

NAME       READY   STATUS      RESTARTS   AGE     IP              NODE      NOMINATED NODE   READINESS GATES
pi-x94l9   0/1     Completed   0          7m10s   10.233.103.88   worker2   <none>           <none>


remote ❯ kubectl delete job pi

job.batch "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

```yaml
apiVersion: batch/v1
kind: Job

metadata:
  name: throw-dice-job

spec:
  completions: 3
  parallelism: 3
  backoffLimit: 25

  template:
    spec:
      containers:
        - name: math-add
          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

※ 참고 : https://github.com/kodekloudhub/throw-dice

throw-dice - 2/2

앞에서 설명한 옵션들이 어떻게 작용하는지 잘 살펴보자

```
remote ❯ cd advanced-kubernetes/03-week/Job

remote ❯ kubectl create -f ./job-throw-dice.yaml

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

```
remote ❯ kubectl get jobs -o wide -w
NAME             COMPLETIONS   DURATION   AGE   CONTAINERS   IMAGES                 SELECTOR
throw-dice-job   0/3                      0s    math-add     kodekloud/throw-dice   controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
throw-dice-job   0/3           0s         0s    math-add     kodekloud/throw-dice   controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
throw-dice-job   0/3           5s         5s    math-add     kodekloud/throw-dice   controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
throw-dice-job   1/3           5s         5s    math-add     kodekloud/throw-dice   controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
throw-dice-job   1/3           7s         7s    math-add     kodekloud/throw-dice   controller-uid=8bca3fe3-d325-4d33-8f21-be14799fd364
throw-dice-job   1/3           15s        15s   math-add     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 간격**으로 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.**

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      completions: 4
      parallelism: 2
      template:
        <pod template>
```

The number of successful jobs to retain

The cron schedule for the job

The number of failed jobs to retain

※ 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!

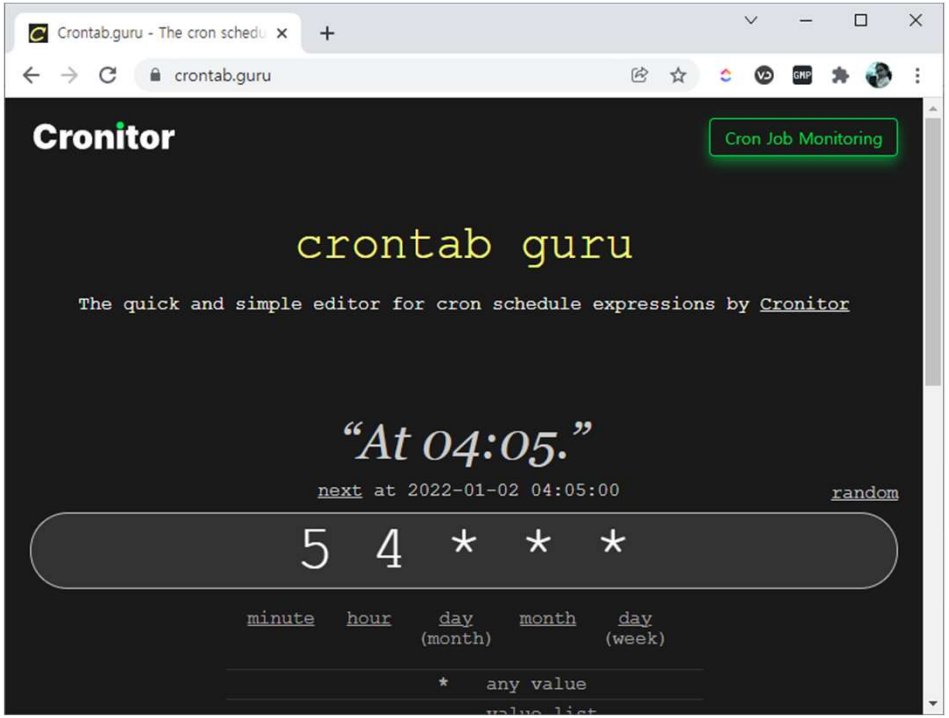
cj-pi.yaml

```yaml
apiVersion: batch/v1beta1
kind: CronJob

metadata:
  name: batch-job-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
            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 advanced-kubernetes/03-week/CronJob

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
batch-job-every-fifteen-minutes   0,15,30,45 * * * *   False     0        <none>          102s    pi           perl     <none>
batch-job-every-fifteen-minutes   0,15,30,45 * * * *   False     1        1s              5m5s    pi           perl     <none>
batch-job-every-fifteen-minutes   0,15,30,45 * * * *   False     0        11s             5m15s   pi           perl     <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
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     Pending             0          0s    <none>          <none>    <none>           <none>
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     Pending             0          0s    <none>          worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     ContainerCreating   0          1s    <none>          worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641028500-7jhvv   1/1     Running             0          5s    10.233.103.90   worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     Completed           0          8s    10.233.103.90   worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     Completed           0          8s    10.233.103.90   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
batch-job-every-fifteen-minutes-1641028500-7jhvv   0/1     Completed   0          15m   10.233.103.90   worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641029400-qp58w   0/1     Completed   0          22s   10.233.103.91   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
batch-job-every-fifteen-minutes   0,15,30,45 * * * *   False     0        14m             79m   pi           perl     <none>


remote ❯ kubectl get pods -o wide

NAME                                          READY   STATUS      RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
batch-job-every-fifteen-minutes-1641030300-pp2jj   0/1     Completed   0          44m   10.233.103.92   worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641031200-ljgw5   0/1     Completed   0          29m   10.233.103.93   worker2   <none>           <none>
batch-job-every-fifteen-minutes-1641032100-glr6d   0/1     Completed   0          14m   10.233.103.94   worker2   <none>           <none>


remote ❯ kubectl delete cronjob batch-job-every-fifteen-minutes

cronjob.batch "batch-job-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-pi-1min-3history.yaml

```yaml
apiVersion: batch/v1beta1
kind: CronJob

metadata:
  name: cronjob-3-history

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

  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1

  jobTemplate:

    spec:
      backoffLimit: 3
      template:
        metadata:
          labels:
            app: periodic-batch-job

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

cj-pi-1min-0history.yaml

```yaml
apiVersion: batch/v1beta1
kind: CronJob

metadata:
  name: cronjob-0-history

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

  successfulJobsHistoryLimit: 0
  failedJobsHistoryLimit: 0

  jobTemplate:

    spec:
      backoffLimit: 3
      template:
        metadata:
          labels:
            app: periodic-batch-job

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

CronJob History - 2/4

2개의 CronJob을 생성해보자 !

```
remote ❯ cd advanced-kubernetes/03-week/CronJob

remote ❯ kubectl create -f ./cj-pi-1min-3history.yaml
cronjob.batch/cronjob-3-history created

remote ❯ kubectl create -f ./cj-pi-1min-0history.yaml
cronjob.batch/cronjob-0-history 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
cronjob-3-history   */1 * * * *   False     0        <none>          0s    pi           perl     <none>
cronjob-0-history   */1 * * * *   False     0        <none>          0s    pi           perl     <none>
cronjob-0-history   */1 * * * *   False     1        0s              1s    pi           perl     <none>
cronjob-3-history   */1 * * * *   False     1        0s              9s    pi           perl     <none>
cronjob-3-history   */1 * * * *   False     0        10s             19s   pi           perl     <none>
cronjob-0-history   */1 * * * *   False     0        20s             21s   pi           perl     <none>
cronjob-0-history   */1 * * * *   False     1        0s              61s   pi           perl     <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
cronjob-0-history-1641034140-28xxg   0/1     Pending             0          0s    <none>          <none>    <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     Pending             0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp   0/1     Pending             0          0s    <none>          <none>    <none>           <none>
cronjob-3-history-1641034140-nmdfp   0/1     Pending             0          0s    <none>          worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp   1/1     Running             0          5s    10.233.103.96   worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp   0/1     Completed           0          8s    10.233.103.96   worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   1/1     Running             0          8s    10.233.103.95   worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp   0/1     Completed           0          8s    10.233.103.96   worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     Completed           0          12s   10.233.103.95   worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     Completed           0          12s   10.233.103.95   worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     Terminating         0          20s   10.233.103.95   worker2   <none>           <none>
cronjob-0-history-1641034140-28xxg   0/1     Terminating         0          20s   10.233.103.95   worker2   <none>           <none>
cronjob-0-history-1641034200-zh56f   0/1     Pending             0          0s    <none>          <none>    <none>           <none>
cronjob-0-history-1641034200-zh56f   0/1     Pending             0          0s    <none>          worker2   <none>           <none>
cronjob-0-history-1641034200-zh56f   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8   0/1     Pending             0          0s    <none>          <none>    <none>           <none>
cronjob-3-history-1641034200-m6df8   0/1     Pending             0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8   0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-0-history-1641034200-zh56f   0/1     ContainerCreating   0          1s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8   0/1     ContainerCreating   0          1s    <none>          worker2   <none>           <none>
cronjob-0-history-1641034200-zh56f   1/1     Running             0          5s    10.233.103.97   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-0-history-1641034260-z8s2h  0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp  0/1     Completed           0          2m    10.233.103.96   worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8  0/1     Completed           0          60s   10.233.103.98   worker2   <none>           <none>
cronjob-3-history-1641034260-l5hht  0/1     ContainerCreating   0          0s    <none>          worker2   <none>           <none>

NAME                                READY   STATUS      RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
cronjob-0-history-1641034260-z8s2h  1/1     Running     0          8s    10.233.103.100  worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp  0/1     Completed   0          2m8s  10.233.103.96   worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8  0/1     Completed   0          68s   10.233.103.98   worker2   <none>           <none>
cronjob-3-history-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-0-history-1641034260-z8s2h  0/1     Completed   0          18s   10.233.103.100  worker2   <none>           <none>
cronjob-3-history-1641034140-nmdfp  0/1     Completed   0          2m18s 10.233.103.96   worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8  0/1     Completed   0          78s   10.233.103.98   worker2   <none>           <none>
cronjob-3-history-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-3-history-1641034140-nmdfp  0/1     Completed   0          2m54s 10.233.103.96   worker2   <none>           <none>
cronjob-3-history-1641034200-m6df8  0/1     Completed   0          114s  10.233.103.98   worker2   <none>           <none>
cronjob-3-history-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-3-history-1641034200-m6df8  0/1     Completed   0          2m34s 10.233.103.98   worker2   <none>           <none>
cronjob-3-history-1641034260-l5hht  0/1     Completed   0          94s   10.233.103.99   worker2   <none>           <none>
cronjob-3-history-1641034320-v4rkn  0/1     Completed   0          34s   10.233.103.102  worker2   <none>           <none>
```

///

Break

///

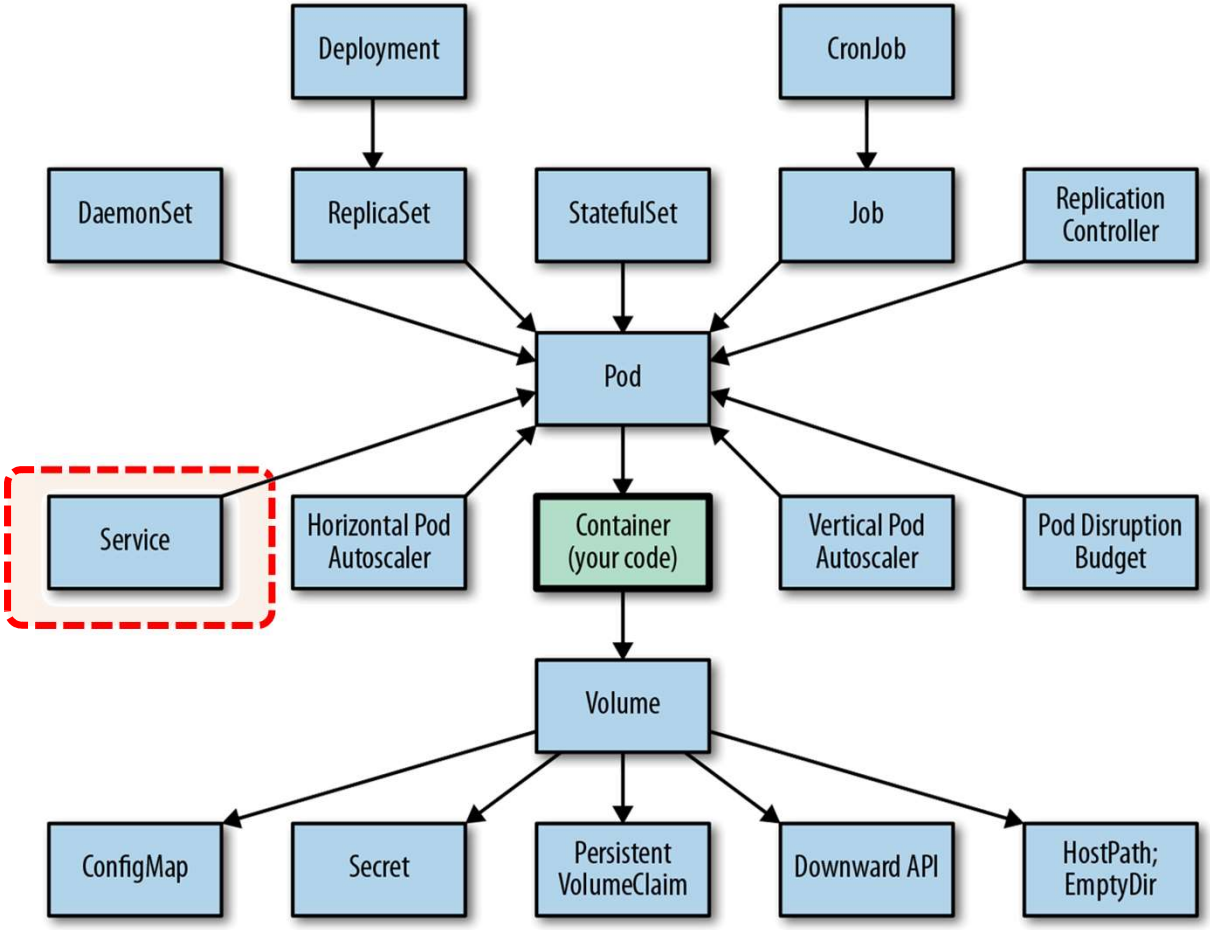
Flip Learning

(Service - ClusterIP/NodePort/ExternalName)

이혜정님

///

Service



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

Kubernetes

Service

너무나 정리가 잘되어 있는 자료

▷ **Kubernetes 네트워크 이해하기 (1) : 컨테이너 네트워크부터 CNI까지**

- https://hyojun.me/~k8s-network-1

▷ **Kubernetes 네트워크 이해하기 (2) : 서비스 개념과 동작 원리**

- https://hyojun.me/~k8s-network-2

※ 참고 : https://www.facebook.com/groups/k8skr/permalink/3011677939113859/

IT 인프라 엔지니어 그룹 밋업

1st IT Infra Engineer Group Meet Up

- https://festa.io/events/592

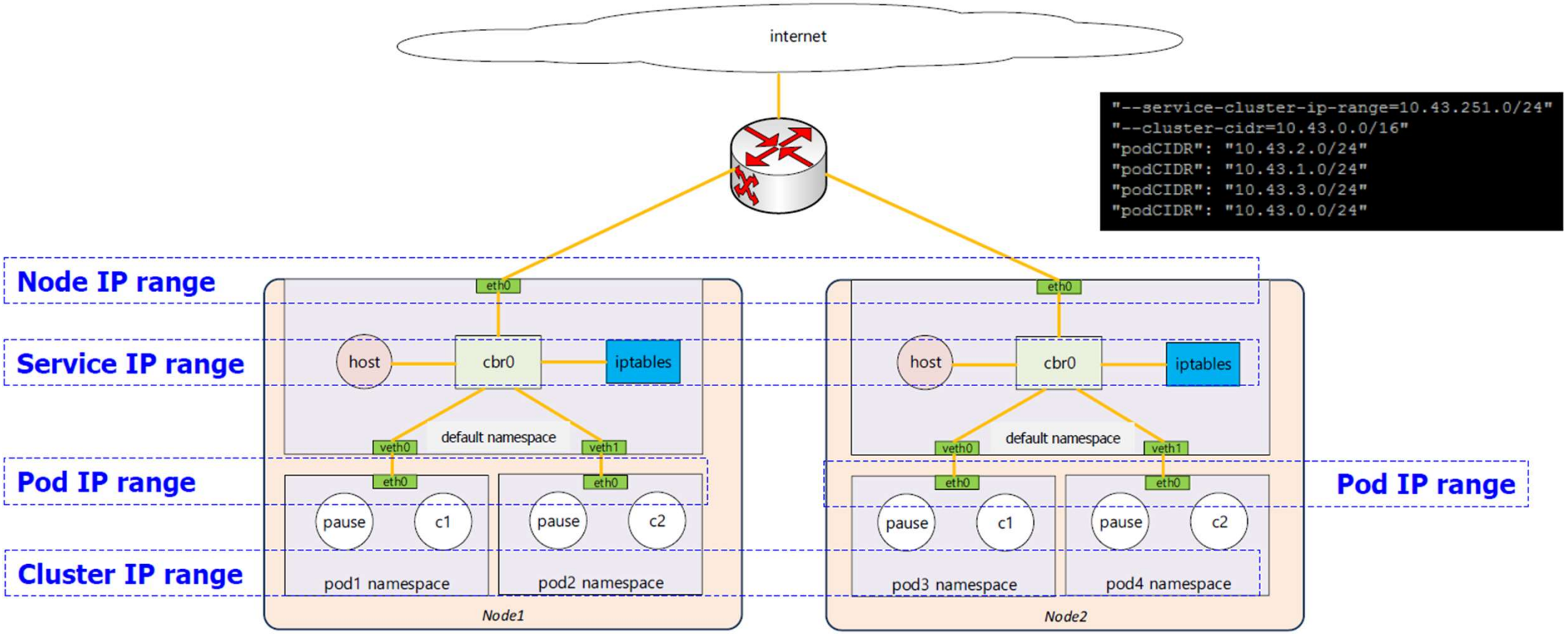
Location: 삼성SDS타워 EASTCAMPUS 5층 X-Room
(서울 송파구 올림픽로35길123)
Date: 2019.10.24 PM 7:00

When	Agenda	Speaker
19:00-19:45	Session1: 자동화를 왜 해야하나요? " Why do I have to do the automation?" (자동화가 구성하는게 더 시간이 걸리지 않나요? 불편하지 않나요? 등에 대하여 유의미한 경험을 공유하고자 합니다.)	조훈(우아하게 앤서블 저자)
20:00-20:45	Session2: "오픈소스를 활용한 netflow/sflow 수집/시각화"	김용대 (홀스홀리커)
21:00-21:45	Session3: "쿠버네티스 네트워킹(Paket Flow in K8S) (쿠버네티스의 패킷흐름을 네트워크 엔지니어의 관점에서 살펴보기)	임창식 (오리멍이)

[발표자료] : https://www.slideshare.net/InfraEngineer/meetup2nd-v12

K8S Network IP Ranges

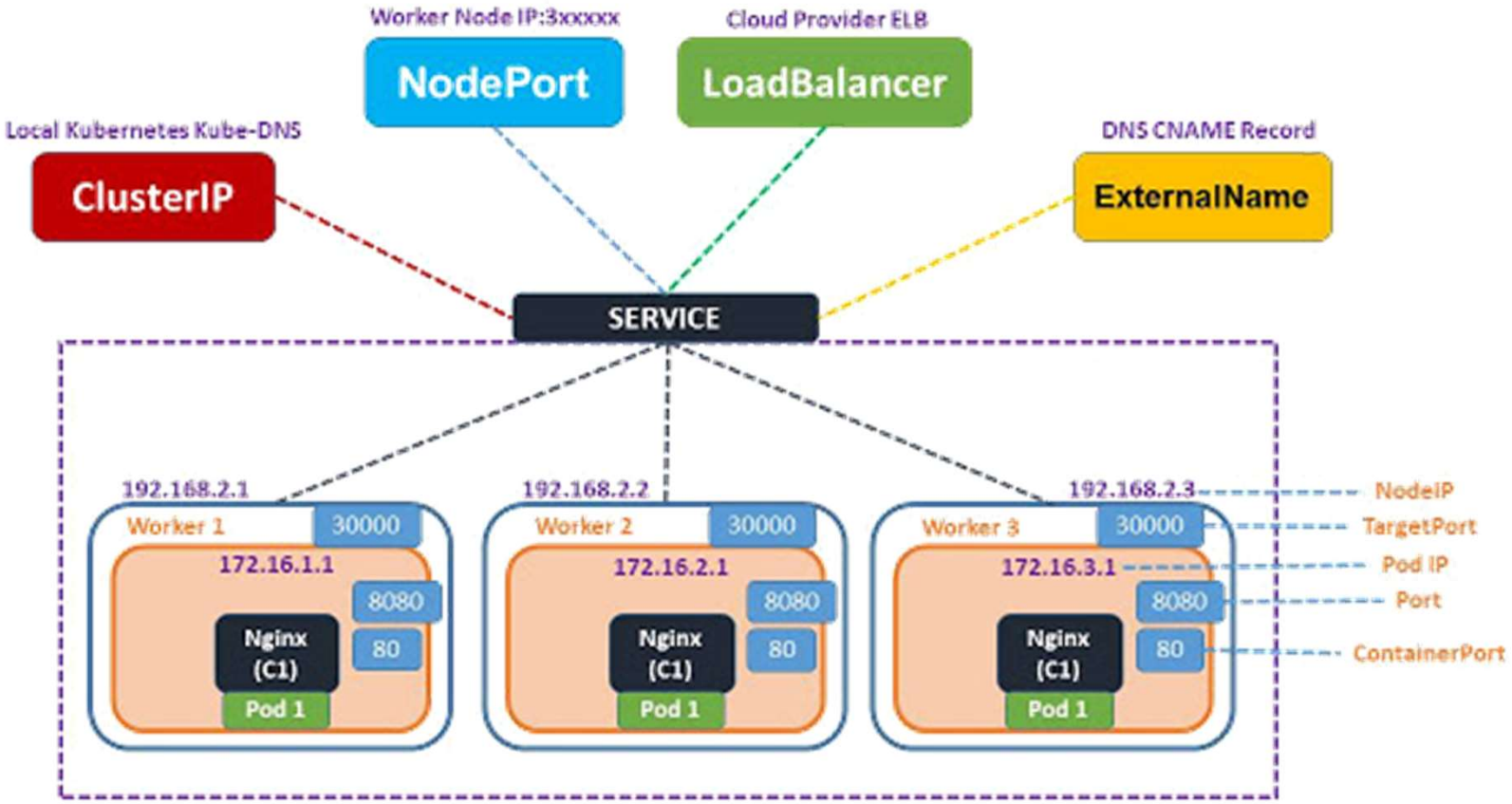
Kubernetes 안에서는 여러 계층의 Network Layer가 존재한다.



※ 참고 : https://www.slideshare.net/InfraEngineer/ss-186475759

Service is ...

- 동일한 서비스를 제공하는 Pod 그룹에 지속적인 **단일 접점**을 만들려고 할 때 생성하는 리소스



※ 참고 : https://www.learnitguide.net/2020/05/kubernetes-services-explained-examples.html

Keywords

▷ ClusterIP

- 디폴트 설정으로, 서비스에 클러스터 IP (내부 IP)를 할당한다.
- 쿠버네티스 클러스터 내에서는 이 서비스에 접근이 가능하지만, 클러스터 외부에서는 외부 IP 를 할당 받지 못했기 때문에 접근이 불가능하다.

▷ NodePort

- 클러스터 IP로만 접근이 가능한 것이 아니라, 모든 노드의 IP와 포트를 통해서도 접근이 가능하게 된다.
- 예를 들어 hello-node-svc 라는 서비스를 NodePort 타입으로 선언을 하고 nodePort를 30036으로 설정하면,
- 설정에 따라 클러스터 IP의 80포트로도 접근이 가능하지만, 모든 노드의 30036 포트로도 서비스를 접근할 수 있다.

▷ Load Balancer

- 보통 클라우드 벤더에서 제공하는 설정 방식으로, 외부 IP 를 가지고 있는 LoadBalancer를 할당한다.
- 외부 IP를 가지고 있기 때문에, 클러스터 외부에서 접근이 가능하다.

▷ ExternalName

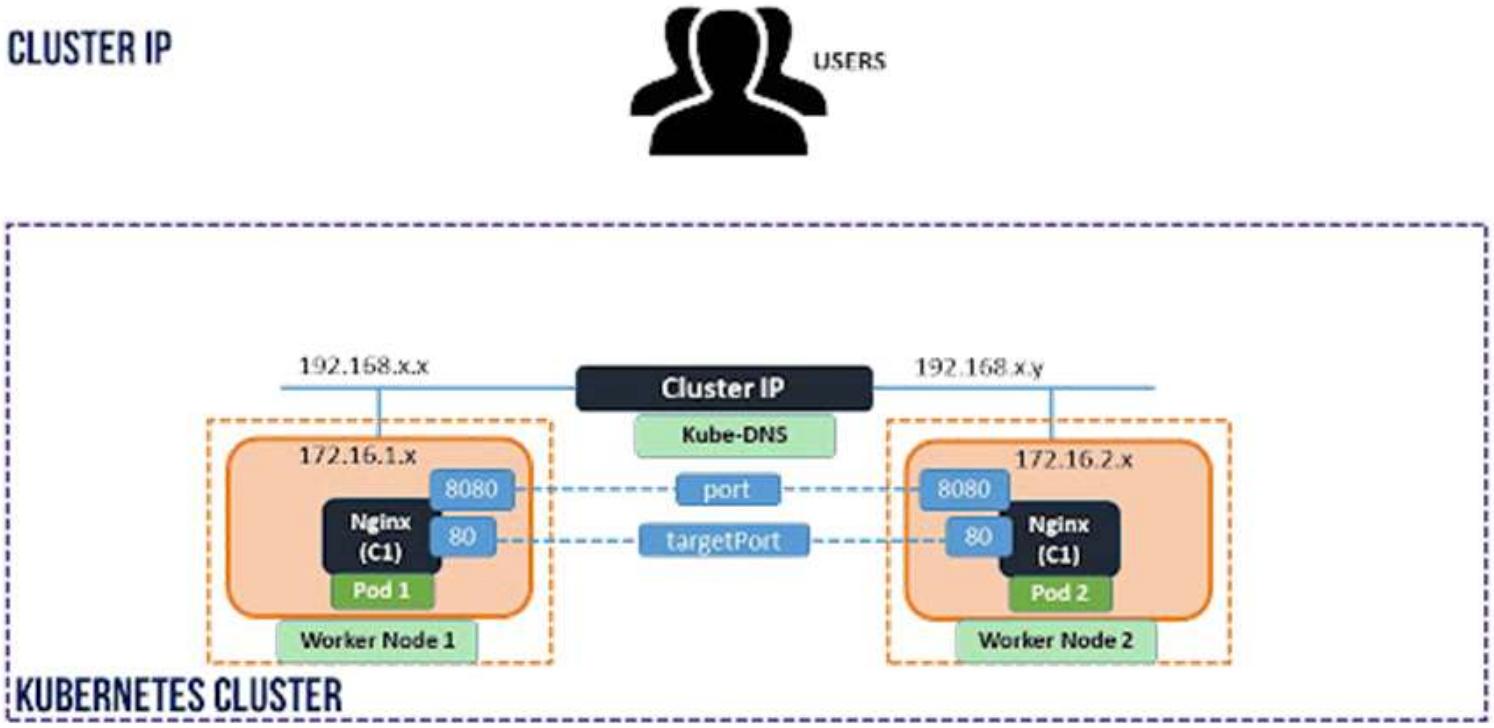
- 외부 서비스를 쿠버네티스 내부에서 호출하고자 할 때 사용할 수 있다.
- Pod들은 클러스터 IP를 가지고 있기 때문에 클러스터 IP 대역 밖의 서비스를 호출하고자 하면, NAT 설정 等 복잡한 설정이 필요하다. 이 때 사용하면 된다.

※ 참고 : https://bcho.tistory.com/1262

ClusterIP

- 디폴트 설정으로, 서비스에 클러스터 IP (내부 IP)를 할당한다.

- 클러스터 내에서는 이 서비스에 접근이 가능하지만, 클러스터 외부에서는 외부 IP 를 할당 받지 못했기 때문에 접근이 불가능하다.



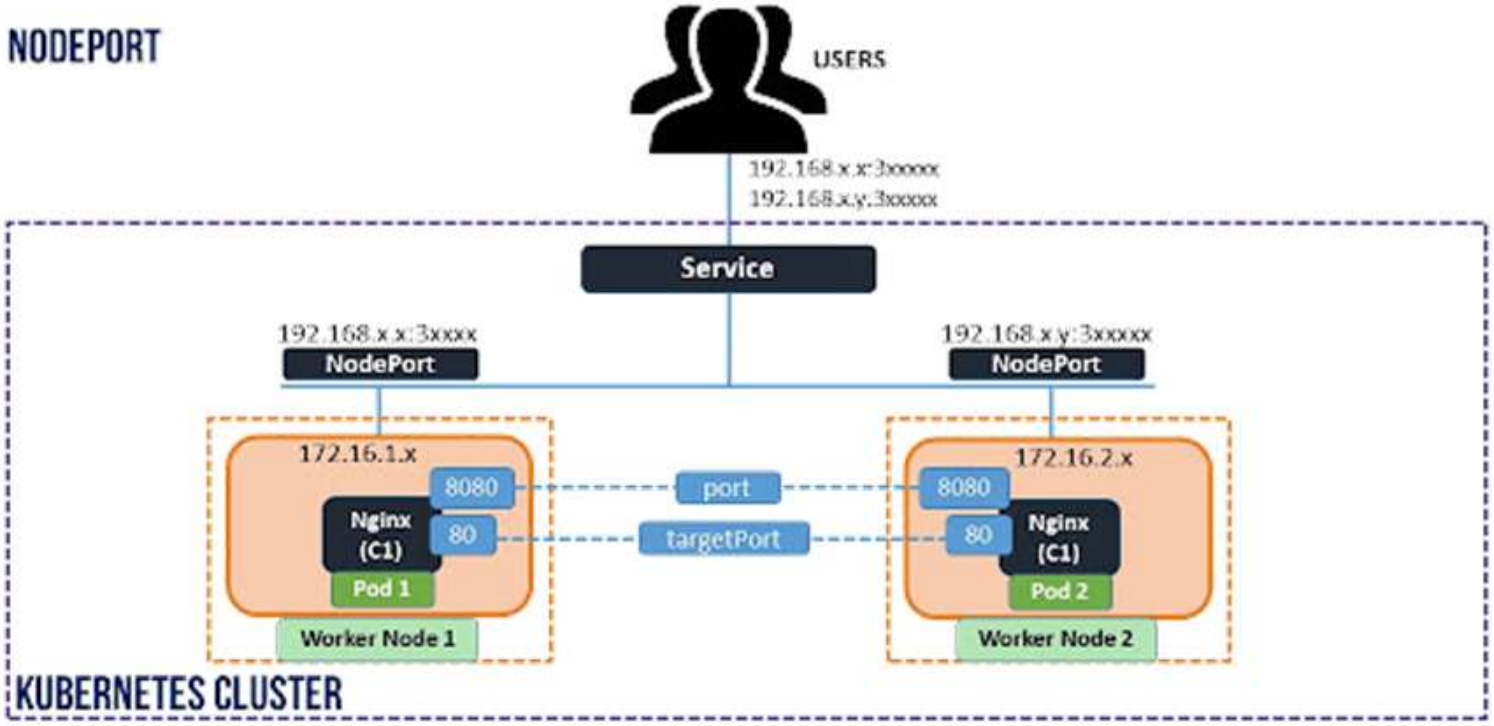
※ 참고 : https://www.learnitguide.net/2020/05/kubernetes-services-explained-examples.html

NodePort

- 클러스터 IP로만 접근이 가능한 것이 아니라, 모든 노드의 IP와 포트를 통해서도 접근이 가능하게 된다.

- 예를 들어 hello-node-svc 라는 서비스를 NodePort 타입으로 선언을 하고 nodePort를 30036으로 설정하면,

- 설정에 따라 클러스터 IP의 80포트로도 접근이 가능하지만, 모든 노드의 30036 포트로도 서비스를 접근할 수 있다.

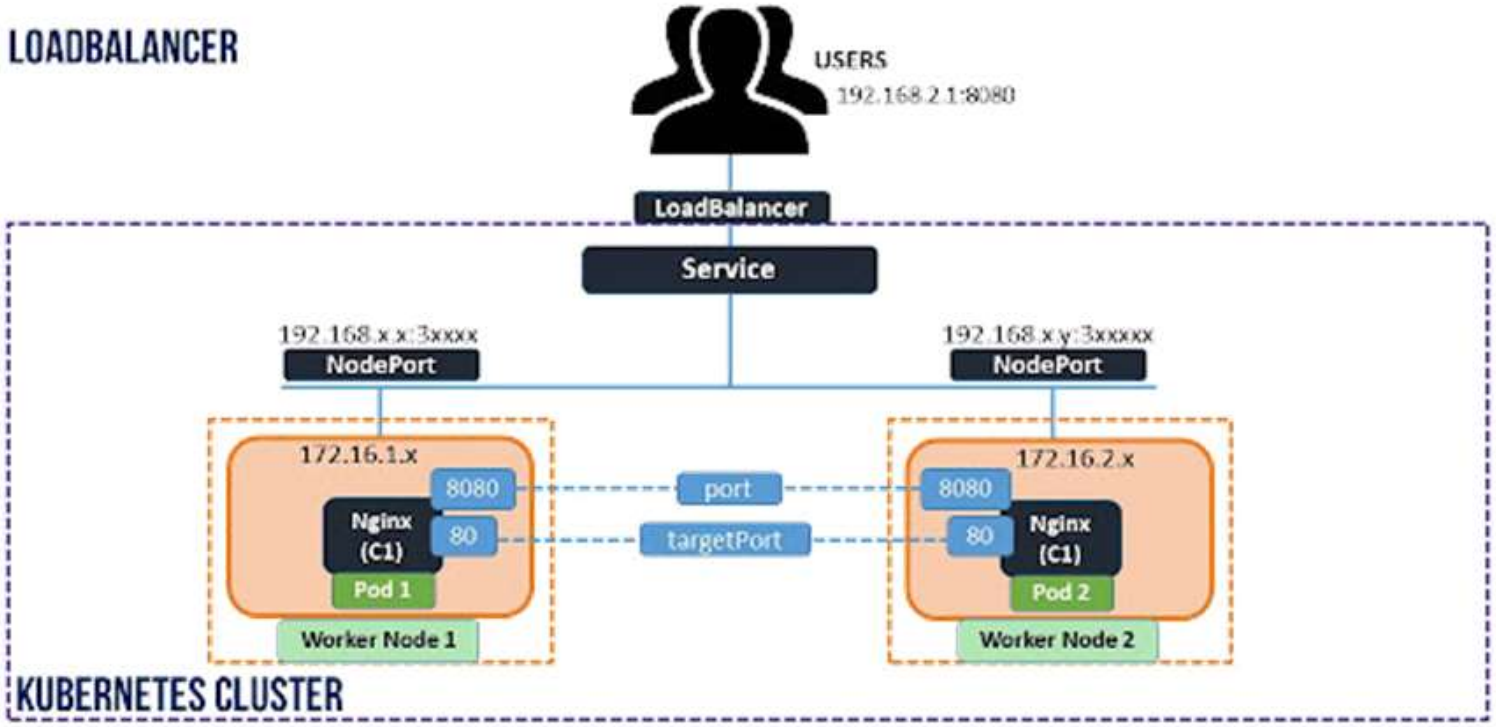


※ 참고 : https://www.learnitguide.net/2020/05/kubernetes-services-explained-examples.html

LoadBalancer

- 보통 클라우드 벤더에서 제공하는 설정 방식으로, 외부 IP 를 가지고 있는 LoadBalancer를 할당한다.

- 외부 IP를 가지고 있기 때문에, 클러스터 외부에서 접근이 가능하다.

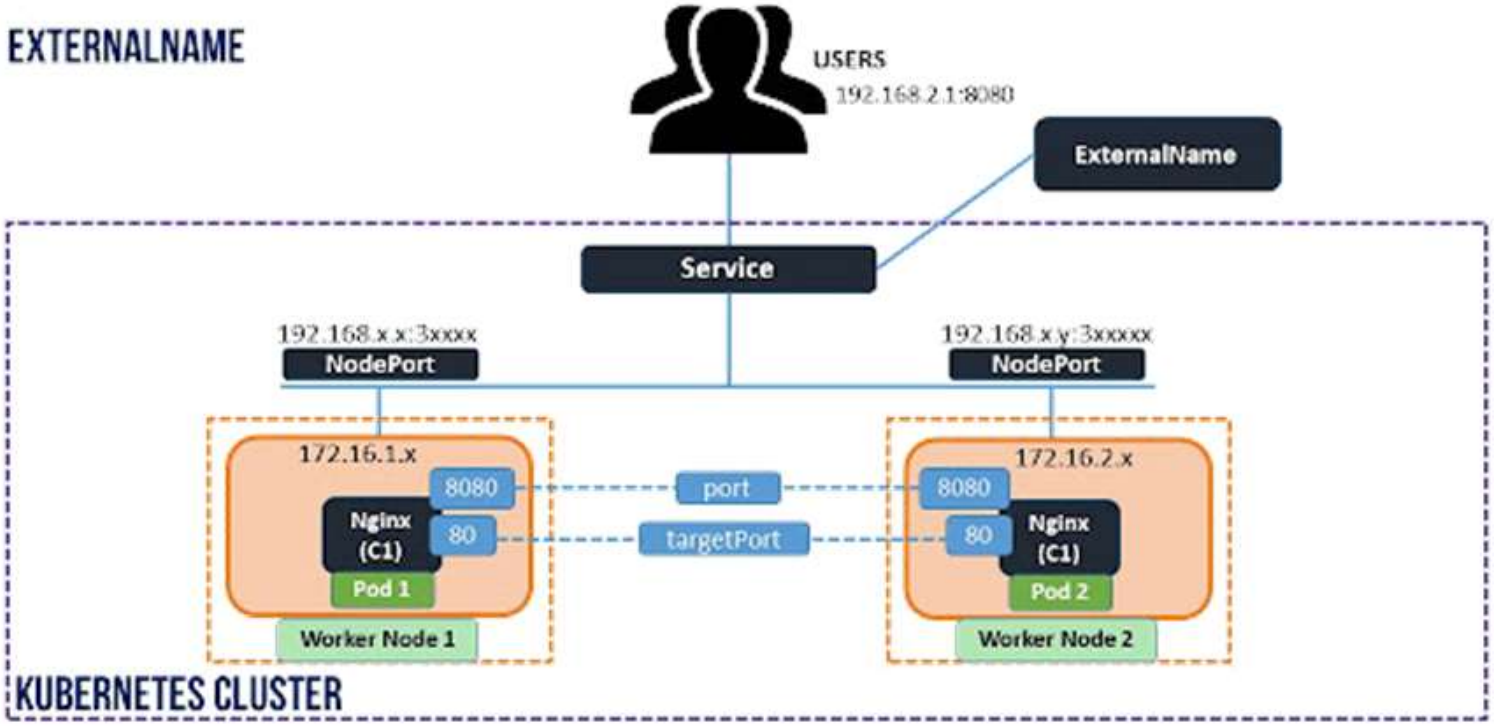


※ 참고 : https://www.learnitguide.net/2020/05/kubernetes-services-explained-examples.html

ExternalName

- 외부 서비스를 쿠버네티스 내부에서 호출하고자 할 때 사용할 수 있다.

- Pod들은 클러스터 IP를 가지고 있기 때문에 클러스터 IP 대역 밖의 서비스를 호출하고자 할 때 사용하면 된다.



※ 참고 : https://www.learnitguide.net/2020/05/kubernetes-services-explained-examples.html

///

K8s : Service
(ClusterIP / NodePort / ExternalName)
Hands-On

ReplicaSet Create

앞에서 공부한 ReplicaSet을 이용해서 Kubernetes Network에 대해서 알아보겠다.

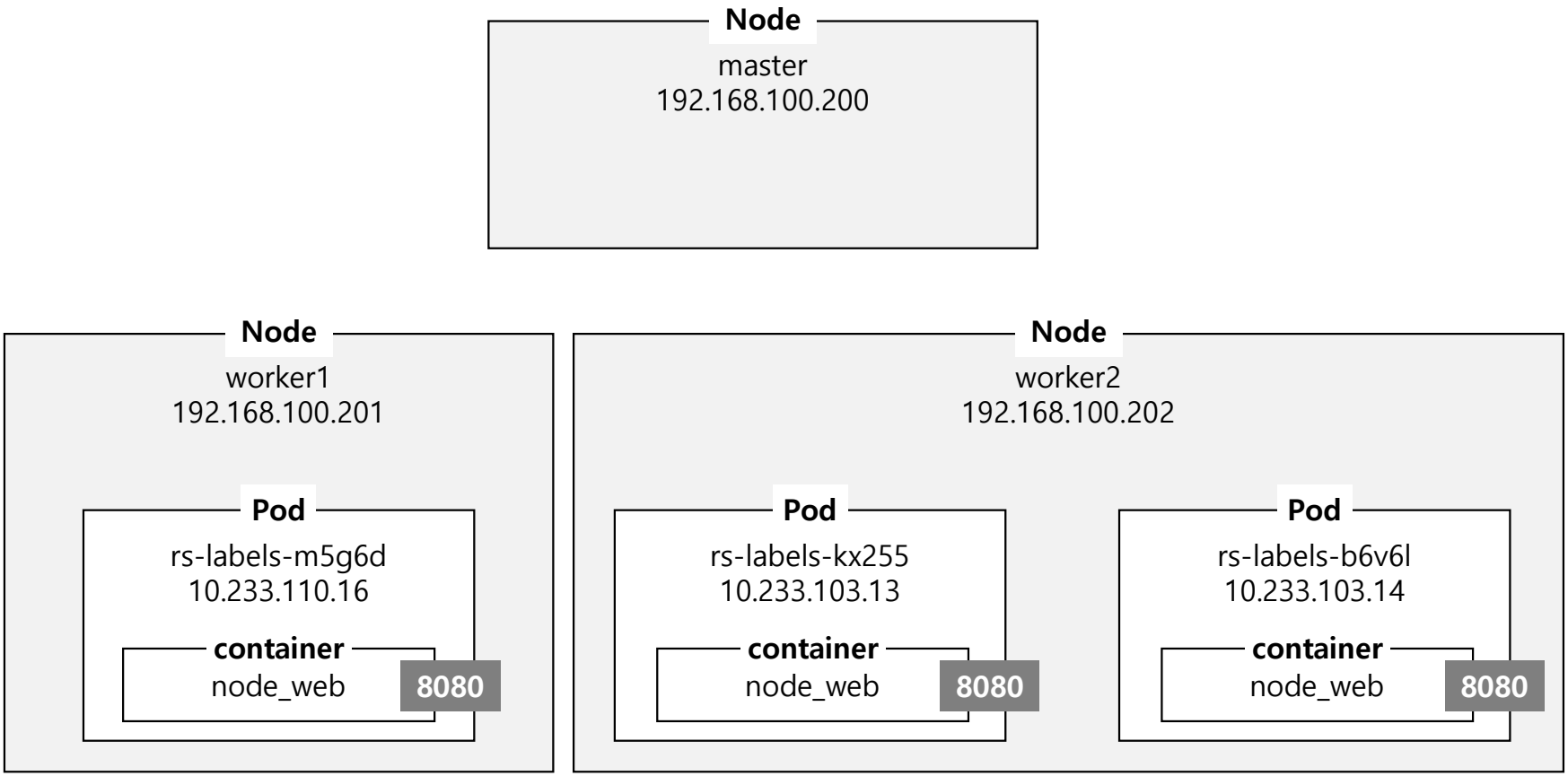
rs-node-web.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: rs-labels

spec:
  replicas: 3

  selector:
    matchLabels:
      app: node-web

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

```
remote ❯ cd advanced-kubernetes/03-week/Service

remote ❯ kubectl create -f ./rs-node-web.yaml

replicaset.apps/rs-labels created


remote ❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-b6v6l   1/1     Running   0          85s   10.233.103.14   worker2   <none>           <none>
rs-labels-kx255   1/1     Running   0          85s   10.233.103.13   worker2   <none>           <none>
rs-labels-m5g6d   1/1     Running   0          85s   10.233.110.16   worker1   <none>           <none>


remote ❯ kubectl get nodes -o wide

NAME      STATUS   ROLES                  AGE   VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION     CONTAINER-RUNTIME
master    Ready    control-plane,master   113d  v1.20.7   192.168.100.200  <none>        Ubuntu 20.04.3 LTS   5.4.0-84-generic   docker://19.3.15
worker1   Ready    <none>                 113d  v1.20.7   192.168.100.201  <none>        Ubuntu 20.04.3 LTS   5.4.0-84-generic   docker://19.3.15
worker2   Ready    <none>                 113d  v1.20.7   192.168.100.202  <none>        Ubuntu 20.04.3 LTS   5.4.0-84-generic   docker://19.3.15
```

Node와 Pod의 IP 정보를 살펴보자.

Now Status


Node
master
192.168.100.200
Node
worker1
192.168.100.201
Pod
rs-labels-m5g6d
10.233.110.16
container
node_web
8080
Node
worker2
192.168.100.202
Pod
rs-labels-kx255
10.233.103.13
container
node_web
8080
Pod
rs-labels-b6v6l
10.233.103.14
container
node_web
8080

Connect Pods

master node에서 Pod를 접근할 수 있을까?

```
remote ❯ ssh vagrant@192.168.100.200

master ❯ kubectl get pods -o wide

NAME             READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-b6v6l   1/1     Running   0          85s   10.233.103.14   worker2   <none>           <none>
rs-labels-kx255   1/1     Running   0          85s   10.233.103.13   worker2   <none>           <none>
rs-labels-m5g6d   1/1     Running   0          85s   10.233.110.16   worker1   <none>           <none>


master ❯ curl http://10.233.110.16:8080

You've hit rs-labels-m5g6d
```

K8s Node가 아닌 외부에서는 Pod를 접근할 수 있을까?

```
master ❯ exit

remote ❯ curl http://10.233.110.16:8080

^C
```

당연히 외부에서는 접근할 수가 없다.

Service Create

우리가 원하는 것은 하나의 접점으로 Pod들 중 하나에 접근할 수 있도록 하는 것이다.

svc-node-web.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-node

spec:
  ports:
  - port: 80
    targetPort: 8080

  selector:
    app: node-web
```

rs-node-web.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet

...

  template:
    metadata:
      labels:
        app: node-web

...
```

```
remote ❯ cd advanced-kubernetes/03-week/Service

remote ❯ kubectl create -f ./svc-node-web.yaml

service/svc-node created


remote ❯ kubectl get services -o wide

NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE    SELECTOR
kubernetes   ClusterIP   10.233.0.1      <none>        443/TCP   113d   <none>
svc-node     ClusterIP   10.233.24.252   <none>        80/TCP    12s    app=node-web


remote ❯ ssh vagrant@192.168.100.200


master ❯ curl http://10.233.24.252

You've hit rs-labels-kx255


master ❯ curl http://10.233.24.252

You've hit rs-labels-b6v6l
```

Restart Pods

ReplicaSet과 Service에 대해서 가장 잘 설명할 방법은, Pods를 죽여버리는 것이다

```
remote ❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-b6v6l   1/1     Running   0          85m   10.233.103.14   worker2   <none>           <none>
rs-labels-kx255   1/1     Running   0          85m   10.233.103.13   worker2   <none>           <none>
rs-labels-m5g6d   1/1     Running   0          85m   10.233.110.16   worker1   <none>           <none>

master ❯ kubectl delete pods rs-labels-b6v6l
master ❯ kubectl delete pods rs-labels-kx255
master ❯ kubectl delete pods rs-labels-m5g6d

master ❯ kubectl get pods -o wide

NAME              READY   STATUS    RESTARTS   AGE     IP              NODE      NOMINATED NODE   READINESS GATES
rs-labels-98pvb   1/1     Running   0          5m38s   10.233.103.15   worker2   <none>           <none>
rs-labels-r52ch   1/1     Running   0          6m18s   10.233.110.18   worker1   <none>           <none>
rs-labels-tws2l   1/1     Running   0          7m8s    10.233.110.17   worker1   <none>           <none>

master ❯ ssh vagrant@192.168.100.200

remote ❯ curl http://10.233.24.252

You've hit rs-labels-tws2l

remote ❯ curl http://10.233.24.252

You've hit rs-labels-98pvb
```

Service Type - ClusterIP

우리가 앞에서 작성한 Service에 type을 지정하지 않았었다. 그러면 기본값은?

svc-node-web.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-node

spec:
  ports:
  - port: 80
    targetPort: 8080

  selector:
    app: node-web
```

```
remote ❯ kubectl get services -o wide

NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE     SELECTOR
kubernetes   ClusterIP   10.233.0.1      <none>        443/TCP   114d    <none>
svc-node     ClusterIP   10.233.24.252   <none>        80/TCP    5h29m   app=node-web


remote ❯ kubectl describe service svc-node

Name:              svc-node
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=node-web
Type:              ClusterIP
IP Families:       <none>
IP:                10.233.24.252
IPs:               10.233.24.252
Port:              <unset>  80/TCP
TargetPort:        8080/TCP
Endpoints:         10.233.103.15:8080,10.233.110.17:8080,10.233.110.18:8080
Session Affinity:  None
Events:            <none>
```

그렇다. 특별히 지정하지 않으면, 기본값으로 ClusterIP type으로 설정이 된다.
그리고, Endpoints가 어떻게 지정되고 있는지도 살펴보면 좋다~!

Service Update - NodePort

앞에서 선언한 Service를 업데이트 하는 방식으로 진행해보자.

svc-node-web-nodeport.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-node

spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30123

  selector:
    app: node-web
```

NodePort를 이용해
Node 밖에서도
접근할 수 있음을 볼 수 있다!

```
remote ❯ cd advanced-kubernetes/04-week/Service

remote ❯ kubectl apply -f ./svc-node-web-nodeport.yaml

Warning: resource services/svc-node is missing the kubectl.kubernetes.io/last-applied-configuration annotation
which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by
either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
service/svc-node configured

remote ❯ kubectl get services -o wide

NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE     SELECTOR
kubernetes   ClusterIP   10.233.0.1      <none>        443/TCP        114d    <none>
svc-node     NodePort    10.233.24.252   <none>        80:30123/TCP   5h45m   app=node-web

remote ❯ curl http://192.168.100.200:30123

You've hit rs-labels-r52ch

remote ❯ curl http://192.168.100.201:30123

You've hit rs-labels-98pvb

remote ❯ curl http://192.168.100.202:30123

You've hit rs-labels-98pvb
```

Service Environments

Pod에서 Service 설정 값을 확인해볼 수 있다.

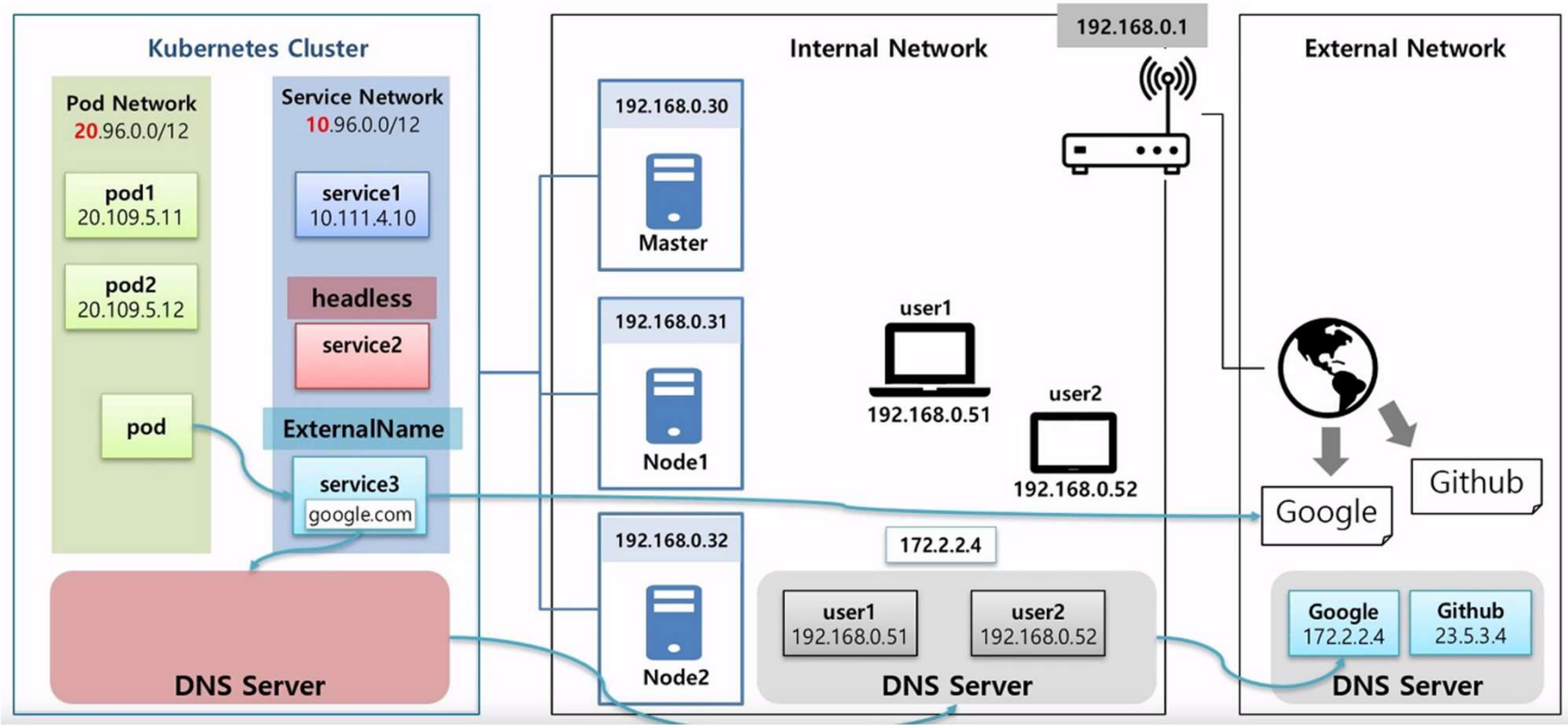
※ Pod가 생성될 시점에서의 Service 정보가 반영된다.
즉, Pod가 생성된 이후의 Service 정보는 모른다.

```
remote ❯ kubectl get services

NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.233.0.1      <none>        443/TCP        114d
svc-node     NodePort    10.233.24.252   <none>        80:30123/TCP   6h27m


remote ❯ kubectl get pods

NAME              READY   STATUS    RESTARTS   AGE
rs-labels-98pvb   1/1     Running   0          6h1m
rs-labels-r52ch   1/1     Running   0          6h2m
rs-labels-tws2l   1/1     Running   0          6h3m


remote ❯ kubectl exec -it rs-labels-tws2l -- env

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=rs-labels-tws2l
TERM=xterm
SVC_NODE_SERVICE_HOST=10.233.24.252
SVC_NODE_PORT=tcp://10.233.24.252:80
SVC_NODE_PORT_80_TCP=tcp://10.233.24.252:80
SVC_NODE_PORT_80_TCP_PROTO=tcp
SVC_NODE_PORT_80_TCP_ADDR=10.233.24.252
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP=tcp://10.233.0.1:443
...
```

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=rs-labels-tws2l
TERM=xterm
SVC_NODE_SERVICE_HOST=10.233.24.252
SVC_NODE_PORT=tcp://10.233.24.252:80
SVC_NODE_PORT_80_TCP=tcp://10.233.24.252:80
SVC_NODE_PORT_80_TCP_PROTO=tcp
SVC_NODE_PORT_80_TCP_ADDR=10.233.24.252
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP=tcp://10.233.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_SERVICE_PORT=443
SVC_NODE_SERVICE_PORT=80
SVC_NODE_PORT_80_TCP_PORT=80
KUBERNETES_SERVICE_HOST=10.233.0.1
KUBERNETES_PORT=tcp://10.233.0.1:443
KUBERNETES_PORT_443_TCP_ADDR=10.233.0.1
NODE_VERSION=15.14.0
YARN_VERSION=1.22.5
HOME=/root
```

ExternalName - Recap 1/3

- ExternalName service3에 google 도메인 이름을 넣으면 DNS를 타고 타서 Google 아이피를 가져올 수 있음
- 추후 google을 github으로 service3의 ExternalName만 변경하면 google이 아닌 github에서 가져오게 할 수 있음



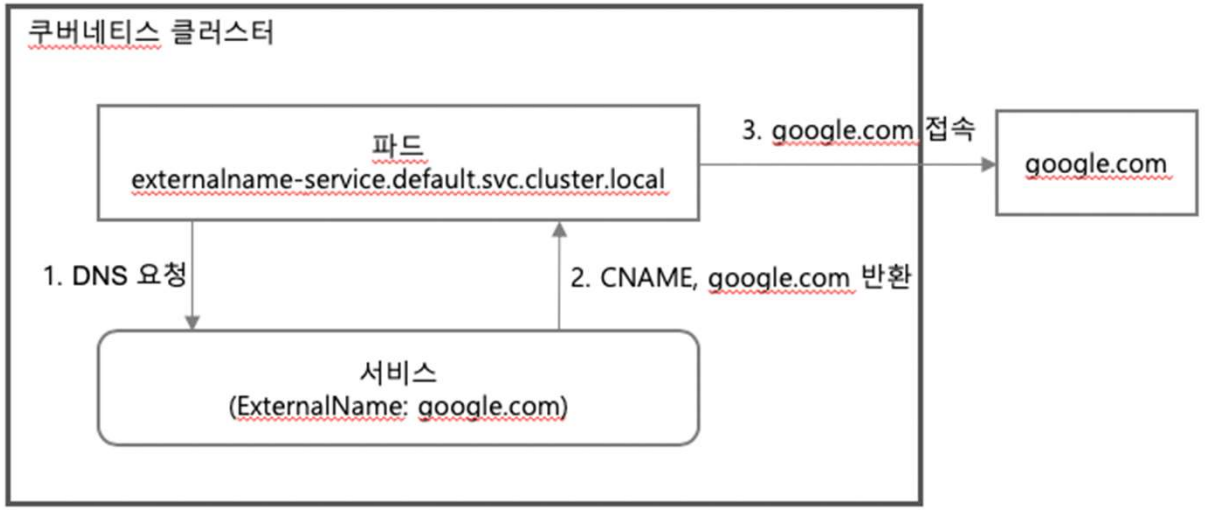
※ 참고 : https://junior-developer.tistory.com/74

ExternalName - Recap 2/3

- 외부 서비스의 별칭으로 사용되는 서비스
  - Pod는 서비스의 FQDN을 사용하는 대신 external-service.default.svc.cluster.local 로 외부 서비스에 연결
  - Service를 사용하는 Pod에서 실제 서비스 이름과 위치가 숨겨짐
  - 나중에 Service spec 수정하여 다른 Service를 가리킬 수 있음

external-name.yaml

```
apiVersion: v1
kind: Service

metadata:
  name: external-service

spec:
  type: ExternalName
  externalName: google.com
```



※ 참고 : https://livebook.manning.com/book/kubernetes-in-action/chapter-5/144

ExternalName - Recap 3/3

- 일반적인 selector에 대한 Service가 아닌, DNS 이름에 대한 Service에 매핑

- ExternalName은 IPv4 주소 문자열 및 숫자로 구성된 DNS 이름도 허용

  . But IPv4 주소와 유사한 ExternalName은 CoreDNS 또는 Ingress-nginx에 의해 확인되지 않음

  . ExternalName은 정식(canonical) DNS 이름을 지정하기 때문 (Proxy, Forwarding이 아닌 DNS라는 점이 중요!?!?)

  . IP 주소를 하드 코딩하려면, 헤드리스(headless) 서비스 사용 권장

- HTTP 및 HTTPS를 포함한, 몇몇 일반적인 프로토콜에 ExternalName을 사용하는 것은 문제점 존재

  . 클러스터 내부의 클라이언트가 사용하는 호스트 이름(hostname)이 ExternalName이 참조하는 이름과 다름

  . 호스트 이름을 사용하는 프로토콜의 경우, 이러한 차이로 인해 오류가 발생하거나 예기치 않은 응답이 발생할 수 있음

  . HTTP 요청에는 오리진(origin) 서버가 인식하지 못하는 'Host :' 헤더가 존재

  . TLS 서버는 클라이언트가 연결된 호스트 이름과 일치하는 인증서를 제공할 수 없음

※ 참고 : https://kubernetes.io/ko/docs/concepts/services-networking/service/#externalname

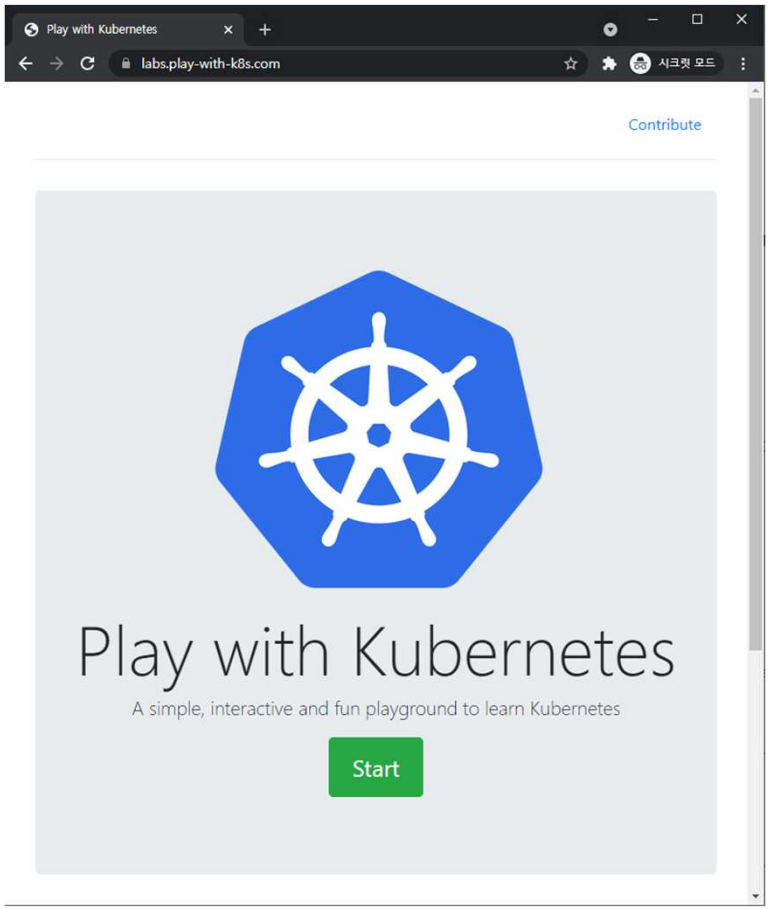
///

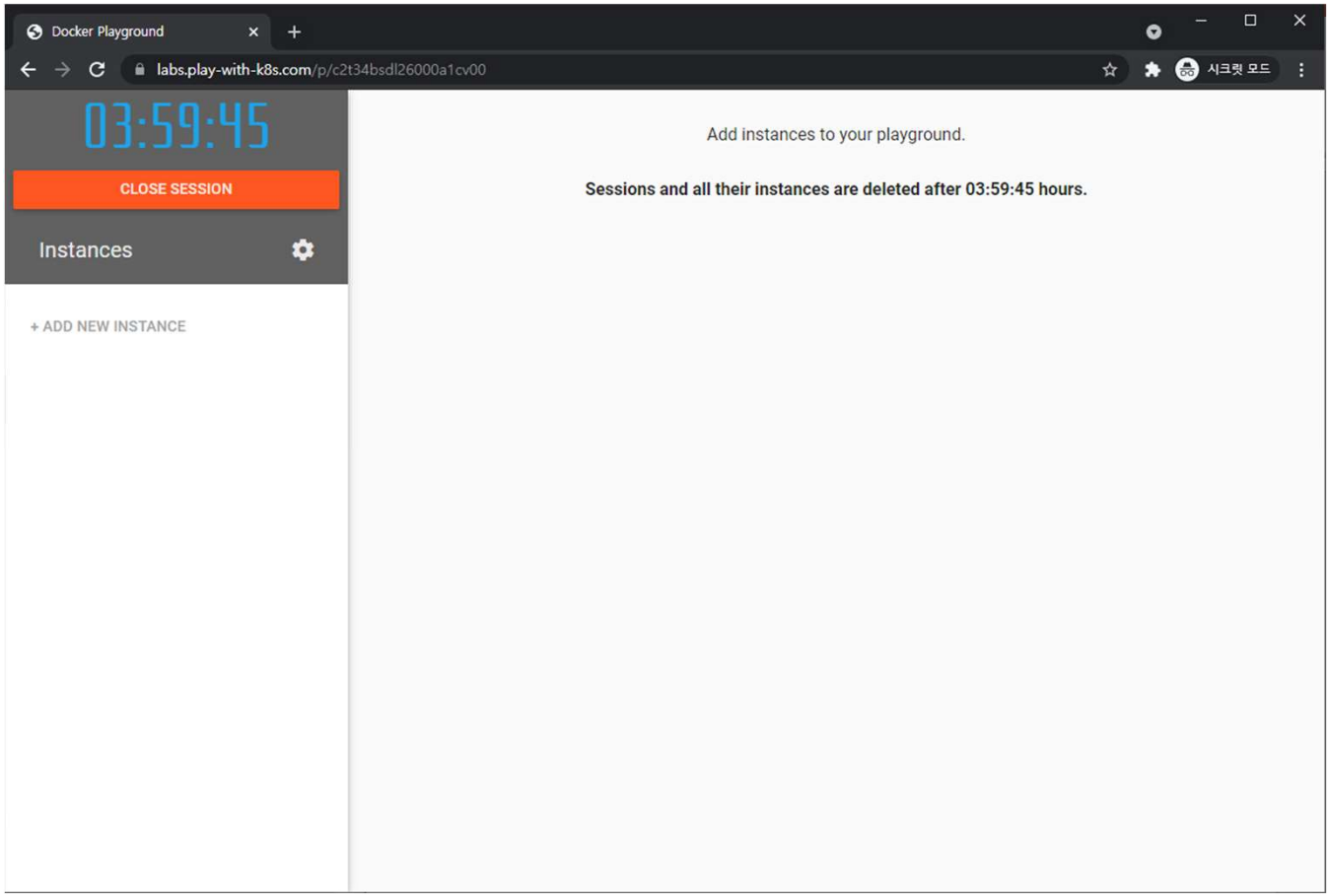
kubernetes - THE DOCUMENTARY

※ 참고 : https://www.youtube.com/watch?v=BE77h7dmoQU&t=5s

실습 환경 : https://labs.play-with-k8s.com/

- 여러 개의 instance 생성 가능, but 4시간 무료 사용
- disk 관련된 제약 等 불편함은 존재





///

https://kahoot.it/

[Score]

이민준 (4)
이혜정 (3)
김상호 (2)
정현찬 (1)
김정은 (1)

자습 (복습)

▷ ReplicaSet / DaemonSet / Job / CronJob

▷ Service - ClusterIP / NodePort / ExternalName

///