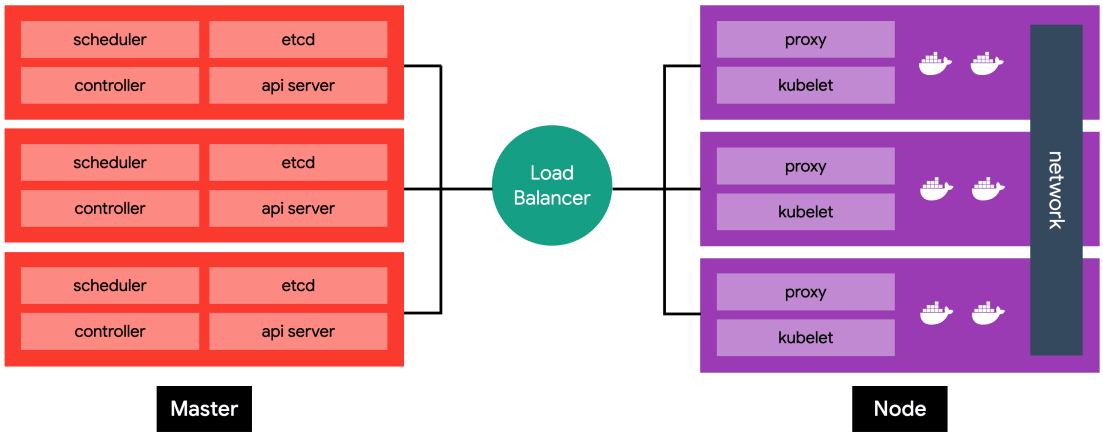
week8

▼ 섹션 2. 쿠버네티스 실습 준비

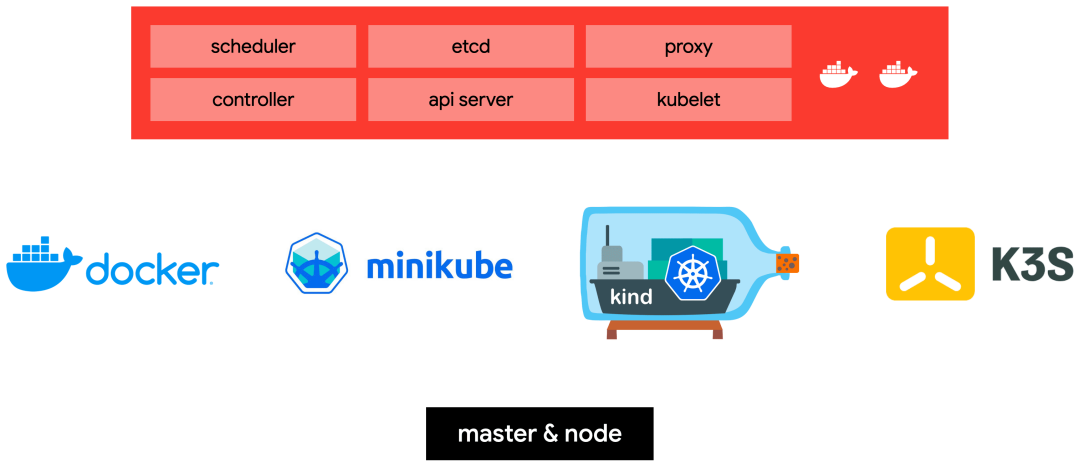
- 🐣 YAML 문법
- 🐣 windows 환경
- ▼ 🐣 쿠버네티스 설치 (windows)

  쿠버네티스를 운영환경에 설치하기 위해선 **최소 3대의 마스터 서버**와 **컨테이너 배포를 위한 n개의 노드 서버**가 필요합니다.



  이러한 설치는 과정이 복잡하고 배포 환경(AWS, Google Cloud, Azure, Bare Metal, ...)에 따라 방법이 다르기 때문에 처음 공부할 때 바로 구축하기는 적합하지 않습니다.

  여기선 개발 환경을 위해 **마스터와 노드를 하나의 서버에 설치**하여 손쉽게 관리하는 방법을 사용합니다. (미니쿠베는 단일 노드!)



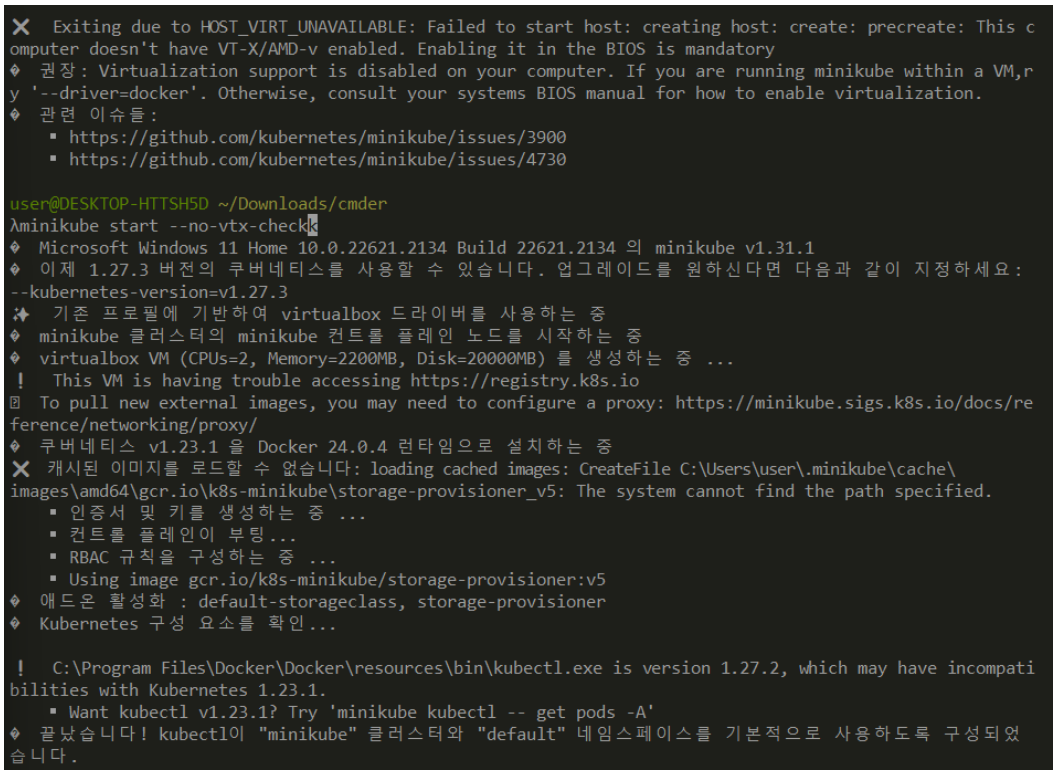
  대표적인 개발 환경 구축 방법으로 **minikube**, **k3s**, **docker for desktop**, **kind**가 있습니다.

- 기본 명령

```
# 버전확인
minikube version
```

```
# 가상머신 시작
minikube start --driver=virtualbox --kubernetes-version=v1.23.1

#Error 발생 : Minikube가 호스트 시스템의 가상화 기술을 확인하지 않고도 Kubernetes 클러스터를 시작
minikube start --no-vtx-check

# 상태확인
minikube status

# 정지
minikube stop

# 삭제
minikube delete

# ssh 접속
minikube ssh

# ip 확인
minikube ip
```

- Error



▼ 🐣 kubectl 설치 (windows)

```
#설치
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.23.5/bin/windows/amd64/kubectl.exe

#버전 확인
kubectl version
```

▼ 섹션 3. 쿠버네티스 기본 실습

▼ 🐣 시작하기

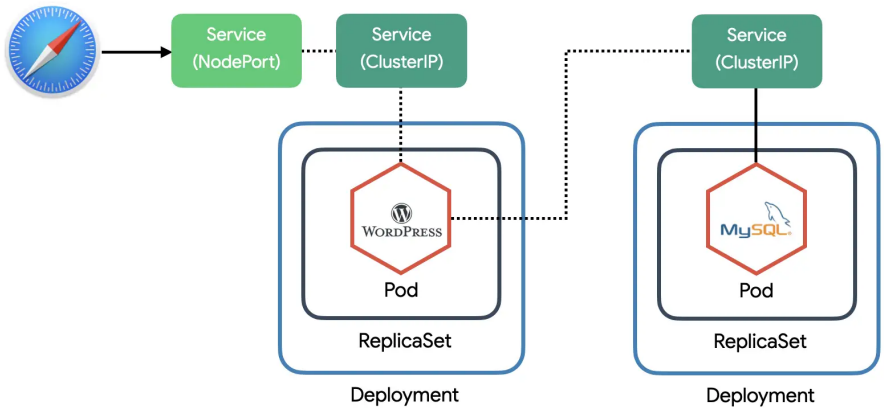
- 🎀 기본 명령어

```
# 버전확인
minikube version

# 가상머신 시작
minikube start

#Error 발생 : Minikube가 호스트 시스템의 가상화 기술을 확인하지 않고도 Kubernetes 클러스터를 시작
minikube start --no-vtx-check

# 상태확인
minikube status

# 정지
minikube stop

# 삭제
minikube delete

# ssh 접속
minikube ssh

# ip 확인
minikube ip
```

- 🎀 워드프레스 배포
  - ▼ 도커로 배포
    - docker-compose.yml

```
#vim 편집기 normal 모드에서 +p 입력시 text 붙여넣기 가능
vi docker-compose.yml

docker-compose up -d

cat docker-compose.yml #30000번 포트 확인

#localhost:30000 접속

docker-compose down
```

  - 쿠버네티스로 배포



    - wordpress-k8s.yml
    - wordpress-k8s.yml에 정의된 내용을 쿠버네티스에 적용

      `kubectl apply -f wordpress-k8s.yml`
    - 현재 default namespace에 배포된 리소스 확인

      `kubectl get all`

```
NAME                                READY   STATUS              RESTARTS   AGE
pod/wordpress-74757b6ff-hvpfn       0/1     ContainerCreating   0          2m13s
pod/wordpress-mysql-5447bfc5b-f5jmm 0/1     ContainerCreating   0          2m13s

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        37m
service/wordpress       NodePort    10.107.168.184   <none>        80:30166/TCP   2m12s
service/wordpress-mysql ClusterIP   10.103.173.249   <none>        3306/TCP       2m13s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress         0/1     1            0           2m13s
deployment.apps/wordpress-mysql   0/1     1            0           2m14s

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/wordpress-74757b6ff          1         1         0       2m13s
replicaset.apps/wordpress-mysql-5447bfc5b    1         1         0       2m13s
```

- `minikube ip` → http://192.168.59.100:30166/ 접속

- 컨테이너가 죽거나 노드 서버가 죽어버려도 자동으로 관리해서 새로 띄운다는 장점이 있음

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λ kubectl deletepod/wordpress-74757b6ff-hvpfn
pod "wordpress-74757b6ff-hvpfn" deleted
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/wordpress-74757b6ff-x5qrd       1/1     Running   0          25s
pod/wordpress-mysql-5447bfc5b-f5jmm 1/1     Running   0          9m27s
```

- 노드가 하나 죽어버려도 정상적으로 작동하도록 replicas 설정하기 (로드밸런싱)
  - `vi wordpress-k8s.yml`

    ```
    spec:
      replicas: 2
    ```

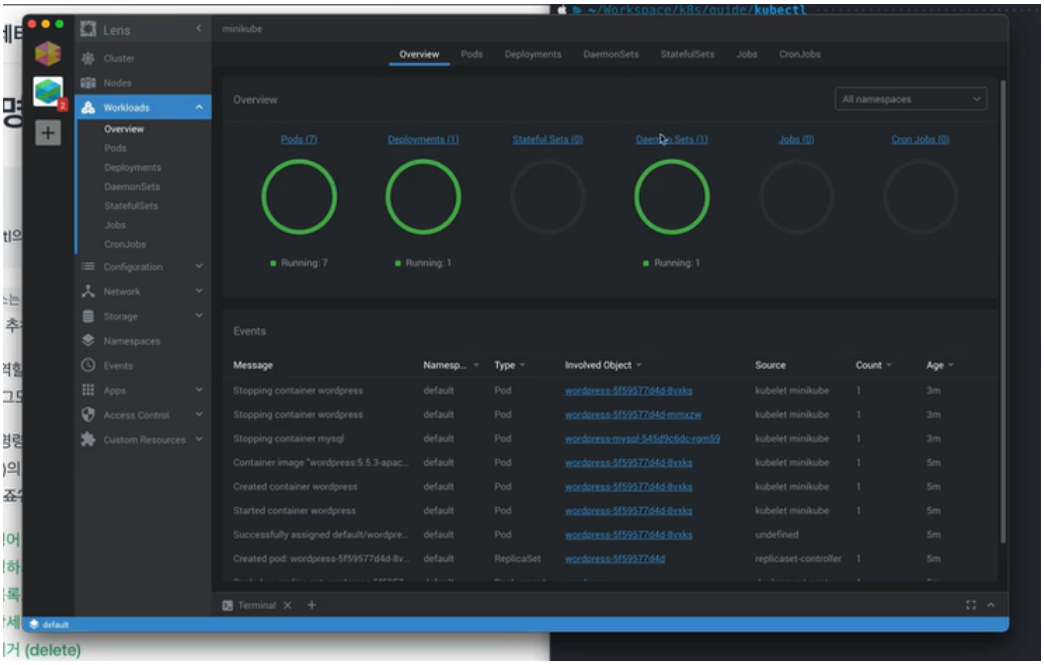
  - `kubectl apply -f wordpress-k8s.yml`
  - `kubectl get all` → 2개 생긴거 확인할 수 있음

    ```
    user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
    λkubectl get all
    NAME                                READY   STATUS    RESTARTS   AGE
    pod/wordpress-74757b6ff-fngm5       1/1     Running   0          5s
    pod/wordpress-74757b6ff-x5qrd       1/1     Running   0          7m31s
    pod/wordpress-mysql-5447bfc5b-f5jmm 1/1     Running   0          16m
    ```

- 제거 : `kubectl delete -f wordpress-k8s.yml`

▼ 🐣 기본 명령어

- kubectl 외에 Lens라는 프로그램도 있음



- 🎀 명령어

명령어	설명
`apply`	원하는 상태를 적용합니다. 보통 `-f` 옵션으로 파일과 함께 사용합니다.
`get`	리소스 목록을 보여줍니다.
`describe`	리소스의 상태를 자세하게 보여줍니다.
`delete`	리소스를 제거합니다.
`logs`	컨테이너의 로그를 봅니다.
`exec`	컨테이너에 명령어를 전달합니다. 컨테이너에 접근할 때 주로 사용합니다.
`config`	kubectl 설정을 관리합니다.

- 🎀 alias로 편하게

```
# alias 설정
alias k='kubectl'

# shell 설정 추가
echo "alias k='kubectl'" >> ~/.bashrc
source ~/.bashrc
```

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λ alias k='kubectl'
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λ k version
```

- 🎀 **상태 설정하기 (apply) :** `kubectl apply -f [파일명 또는 URL]`

`kubectl apply -f https://subicura.com/k8s/code/guide/index/wordpress-k8s.yml`

- 🎀 **리소스 목록보기 (get) :** `kubectl get [TYPE]`

```
# Pod 조회
kubectl get pod
#상태 확인 옵션
kubectl get pod -o wide
kubectl get pod -o yaml

# 줄임말(Shortname)과 복수형 사용가능
kubectl get pods
kubectl get po
```

```
# 여러 TYPE 입력
kubectl get pod,service
kubectl get po,svc

# Pod, ReplicaSet, Deployment, Service, Job 조회 => all
kubectl get all

# 결과 포맷 변경
kubectl get pod -o wide
kubectl get pod -o yaml
kubectl get pod -o json

# Label 조회 : pod에 할당된 labels를 볼 수 있음
kubectl get pod --show-labels

# k8s에서 사용할 수 있는 리소스 명령어 조회
kubectl api-resources
```

- 🎀 **리소스 상세 상태보기 (describe) :** `kubectl describe [TYPE]/[NAME] 또는 [TYPE] [NAME]`

```
# Pod 조회로 이름 검색
kubectl get pod

# 조회한 이름으로 상세 확인 (evenet 페이지를 많이 보게 될 것!)
kubectl describe pod/wordpress-74757b6ff-vz6tl
```

- 🎀 **리소스 제거 (delete):** `kubectl delete [TYPE]/[NAME] 또는 [TYPE] [NAME]`

```
# Pod 조회로 이름 검색
kubectl get pod

# 조회한 Pod 제거
kubectl delete pod/wordpress-74757b6ff-vz6tl
```

- 🎀 **컨테이너 로그 조회 (logs) :** `kubectl logs [POD_NAME]`

```
# Pod 조회로 이름 검색
kubectl get pod

# 조회한 Pod 로그조회
kubectl logs wordpress-74757b6ff-9sp4j

# 실시간 로그 보기
kubectl logs -f wordpress-74757b6ff-9sp4j
```

- 🎀 **컨테이너 명령어 전달 (exec) :** `kubectl exec [-it] [POD_NAME] -- [COMMAND]`
  - 어느 서버에 떠있든지 자동으로 접속하고 싶은 pod에 연결시켜준다는 장점

```
# Pod 조회로 이름 검색
kubectl get pod

# 조회한 Pod의 컨테이너에 접속
kubectl exec -it wordpress-74757b6ff-9sp4j -- bash
```

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λkubectl exec -it wordpress-74757b6ff-9sp4j -- bash
root@wordpress-74757b6ff-9sp4j:/var/www/html# ls -al
total 244
drwxr-xr-x  5 www-data www-data  4096 Aug 16 11:34 .
drwxr-xr-x  1 root     root      4096 Mar  1  2022 ..
-rw-r--r--  1 www-data www-data   261 Mar 11  2022 .htaccess
-rw-r--r--  1 www-data www-data   405 Feb  6  2020 index.php
-rw-r--r--  1 www-data www-data 19915 Jan  1  2022 license.txt
-rw-r--r--  1 www-data www-data  7437 Dec 28  2021 readme.html
-rw-r--r--  1 www-data www-data  7165 Jan 21  2021 wp-activate.php
drwxr-xr-x  9 www-data www-data  4096 Feb 22  2022 wp-admin
-rw-r--r--  1 www-data www-data   351 Feb  6  2020 wp-blog-header.php
-rw-r--r--  1 www-data www-data  2338 Nov  9  2021 wp-comments-post.php
-rw-rw-r--  1 www-data www-data  5480 Mar 11  2022 wp-config-docker.php
-rw-r--r--  1 www-data www-data  3001 Dec 14  2021 wp-config-sample.php
-rw-r--r--  1 www-data www-data  5584 Aug 16 11:34 wp-config.php
drwxr-xr-x  5 www-data www-data  4096 Feb 22  2022 wp-content
-rw-r--r--  1 www-data www-data  3939 Aug  3  2021 wp-cron.php
drwxr-xr-x 26 www-data www-data 16384 Feb 22  2022 wp-includes
-rw-r--r--  1 www-data www-data  2496 Feb  6  2020 wp-links-opml.php
-rw-r--r--  1 www-data www-data  3900 May 15  2021 wp-load.php
-rw-r--r--  1 www-data www-data 47916 Jan  4  2022 wp-login.php
-rw-r--r--  1 www-data www-data  8582 Sep 22  2021 wp-mail.php
-rw-r--r--  1 www-data www-data 23025 Nov 30  2021 wp-settings.php
-rw-r--r--  1 www-data www-data 31959 Oct 25  2021 wp-signup.php
-rw-r--r--  1 www-data www-data  4747 Oct  8  2020 wp-trackback.php
-rw-r--r--  1 www-data www-data  3236 Jun  8  2020 xmlrpc.php
```

- 🎀 **설정 관리 (config)**

```
# 현재 컨텍스트 확인
kubectl config current-context

# 컨텍스트 설정
kubectl config use-context
```

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λkubectl config current-context
minikube
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/index
λkubectl config use-context minikube
Switched to context "minikube".
```

- 🎀 **그 외**

```
# 전체 오브젝트 종류 확인
kubectl api-resources

# 특정 오브젝트 설명 보기
kubectl explain pod
```

- 워드프레스 리소스 제거

`kubectl delete -f https://subicura.com/k8s/code/guide/index/wordpress-k8s.yml`

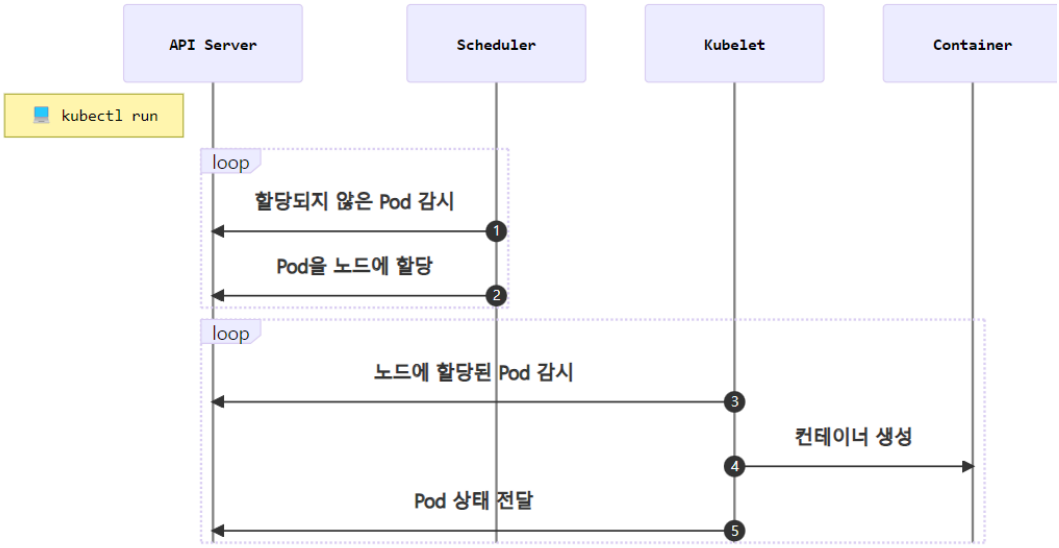
▼ 🐥 Pod

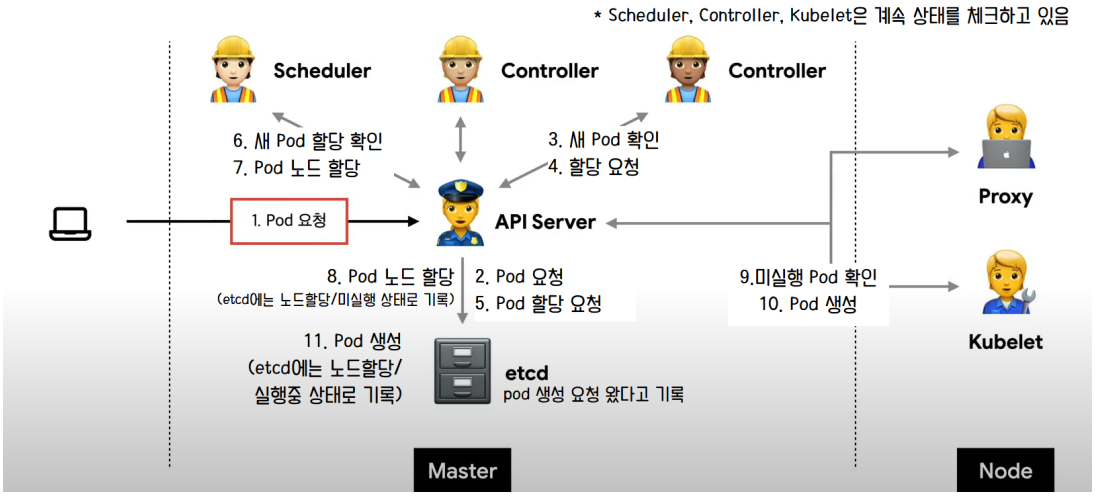
- 🎀 빠르게 Pod 만들기
  - `kubectl run echo --image ghcr.io/subicura/echo:v1`

```
λkubectl get pod
NAME   READY   STATUS              RESTARTS   AGE
echo   0/1     ContainerCreating   0          14s
```

- `kubectl describe pod/echo`
- 제거 : `kubectl delete pod/echo`

▼ 🎀 Pod 생성 분석





1. `Scheduler`는 API서버를 감시하면서 할당되지 않은 `Pod`이 있는지 체크
2. `Scheduler`는 할당되지 않은 `Pod`을 감지하고 적절한 `노드`에 할당 (minikube는 단일 노드)
3. 노드에 설치된 `kubelet`은 자신의 노드에 할당된 `Pod`이 있는지 체크
4. `kubelet`은 `Scheduler`에 의해 자신에게 할당된 `Pod`의 정보를 확인하고 컨테이너 생성
5. `kubelet`은 자신에게 할당된 `Pod`의 상태를 `API 서버`에 전달

▼ 🎀 YAML로 설정파일 Spec 작성하기

- echo-pod.yml

```
apiVersion: v1
kind: Pod
metadata: #리소스의 보
  name: echo
  labels:
    app: echo
spec: #Pod에 대한 spec
```

```
  containers:
    - name: app
      image: ghcr.io/subicura/echo:v1
```

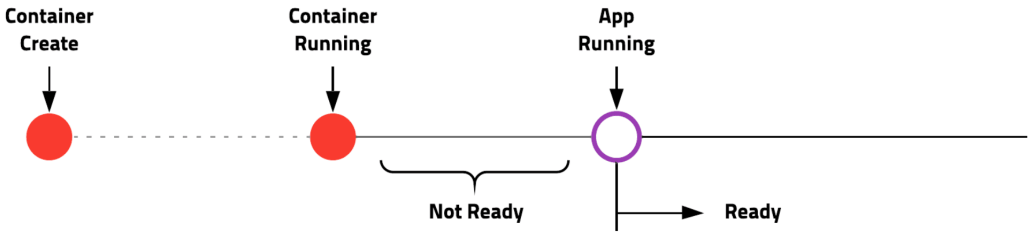
- **필수 요소**

정의	설명	예
`version`	오브젝트 버전	v1, app/v1, networking.k8s.io/v1, ...
`kind`	종류	Pod, ReplicaSet, Deployment, Service, ...
`metadata`	메타데이터	name과 label, annotation(주석)으로 구성
`spec`	상세명세	리소스 종류마다 다름

`version`, `kind`, `metadata`, `spec`는 리소스를 정의할 때 반드시 필요한 요소입니다.

```
# Pod 생성
kubectl apply -f echo-pod.yml

# Pod 목록 조회
kubectl get pod

# Pod 로그 확인
kubectl logs echo
kubectl logs -f echo

# Pod 컨테이너 접속
kubectl exec -it echo -- sh
# ls
# ps
# exit

# Pod 제거
kubectl delete -f echo-pod.yml
```

▼ 🎀 컨테이너 상태 모니터링



- `컨테이너 생성`과 실제 `서비스 준비`는 약간의 차이가 있습니다. 서버를 실행하면 바로 접속할 수 없고 짧게는 수초, 길게는 수분의 초기화 시간이 필요한데 **실제로 접속이 가능할 때** `서비스가 준비되었다` **고 말할 수 있습니다.**
- 쿠버네티스는 컨테이너가 생성되고 **서비스가 준비되었다는 것을 체크하는 옵션을 제공**하여 초기화하는 동안 서비스되는 것을 막을 수 있습니다.
  - 내부적으로 서비스에 실제 접속하고 성공적으로 서비스에 접속되면, 외부에서 접속되도록 합니다.

▼ 1. livenessProbe

컨테이너가 정상적으로 동작하는지 체크하고 정상적으로 동작하지 않는다면 **컨테이너를 재시작**하여 문제를 해결합니다.

`정상`이라는 것은 여러 가지 방식으로 체크할 수 있는데 여기서는 `http get` 요청을 보내 확인하는 방법을 사용합니다.

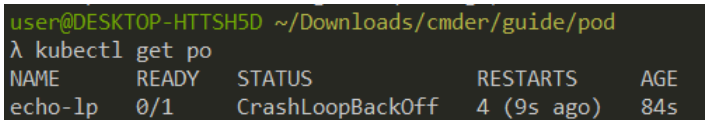
- echo-lp.yml 에서 일부러 존재하지 않는 path(/not/exist)와 port(8080)를 입력하였습니다.

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: echo-lp
  labels:
    app: echo
spec:
  containers:
    - name: app
      image: ghcr.io/subicura/echo:v1
      livenessProbe:
        httpGet:
          path: /not/exist
          port: 8080
        initialDelaySeconds: 5
        timeoutSeconds: 2 # Default 1
        periodSeconds: 5 # Defaults 10
        failureThreshold: 1 # Defaults 3
```

`kubectl apply -f echo-lp.yml`

- **상태 확인**



정상적으로 응답하지 않았기 때문에 Pod이 여러 번 재시작되고 `CrashLoopBackOff` 상태로 변경되었습니다.

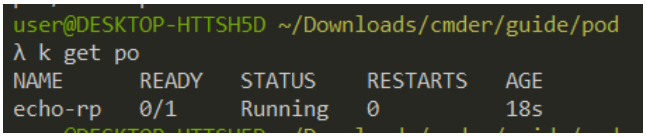
▼ 2. **readinessProbe**

- 컨테이너가 준비되었는지 체크하고 정상적으로 준비되지 않았다면 **Pod으로 들어오는 요청을 제외**합니다.
- livenessProbe와 차이점은 **문제가 있어도 Pod을 재시작하지 않고 요청만 제외한다는 점**입니다.
- **guide/pod/echo-rp.yml**

```
apiVersion: v1
kind: Pod
metadata:
  name: echo-rp
  labels:
    app: echo
spec:
  containers:
    - name: app
      image: ghcr.io/subicura/echo:v1
      readinessProbe:
        httpGet:
          path: /not/exist
          port: 8080
        initialDelaySeconds: 5
        timeoutSeconds: 2 # Default 1
        periodSeconds: 5 # Defaults 10
        failureThreshold: 1 # Defaults 3
```

`kubectl apply -f echo-rp.yml`

- **상태 확인**



READY상태가 `0/1`인 것을 확인할 수 있습니다.

▼ 3. **livenessProbe + readinessProbe**

보통 `livenessProbe`와 `readinessProbe`를 같이 적용합니다. 상세한 설정은 애플리케이션 환경에 따라 적절하게 조정합니다.

- **guide/pod/echo-pod-health.yml**

```
apiVersion: v1
kind: Pod
metadata:
  name: echo-health
  labels:
    app: echo
spec:
  containers:
    - name: app
      image: ghcr.io/subicura/echo:v1
      livenessProbe:
        httpGet:
          path: /
          port: 3000
      readinessProbe:
        httpGet:
          path: /
          port: 3000
```

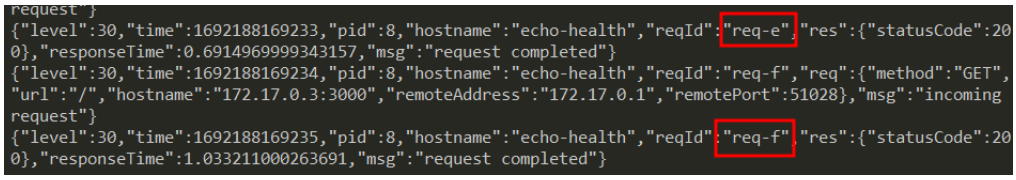
`kubectl apply -f echo-pod-health.yml`

`3000`번 포트와 `/` 경로는 정상적이기 때문에 Pod이 오류없이 생성된 것을 확인할 수 있습니다.

- 상태확인

  `k logs -f echo-health`

  주기적으로 서버가 살아있는지 체크하고 있는 것을 볼 수 있음



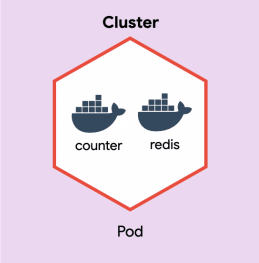
▼ 🎀 다중 컨테이너

- 대부분 `1 Pod = 1 컨테이너`지만 여러 개의 컨테이너를 가진 경우도 꽤 흔합니다.
- 하나의 Pod에 속한 컨테이너는 서로 네트워크를 **localhost로 공유하고 동일한 디렉토리를 공유**할 수 있습니다.
- **guide/pod/counter-pod-redis.yml**

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
  labels:
    app: counter
spec:
  containers:
    - name: app
      image: ghcr.io/subicura/counter:latest
      env:
      # 환경변수 :
      # counter app이 localhost로 redis에 접근
        - name: REDIS_HOST
          value: "localhost"
    - name: db
      image: redis
```

요청횟수를 redis에 저장하는 간단한 웹 애플리케이션을 다중 컨테이너로 생성합니다.

다중 컨테이너를 포함한 Pod은 다음과 같습니다.



같은 Pod에 컨테이너가 생성되었기 때문에 counter앱은 redis를 `localhost`로 접근할 수 있습니다.

```
/app # apk add curl busyboc-extras
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/main/x
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/commun
ERROR: unable to select packages:
  busyboc-extras (no such package):
    required by: world[busyboc-extras]
/app # curl localhost:3000
1
/app # curl localhost:3000
2
/app # telnet localhost 6379
Connected to localhost
dbsize
:1
KEYS *
*1
$5
count
Get count
$1
2
quit
+OK
Connection closed by foreign host
/app # exit
command terminated with exit code 1
```

```
# Pod 생성
kubectl apply -f counter-pod-redis.yml

# Pod 목록 조회
kubectl get pod

# Pod 로그 확인
kubectl logs counter # 오류 발생 (컨테이너 지정 필요)
kubectl logs counter app
kubectl logs counter db

# Pod의 app컨테이너 접속
kubectl exec -it counter -c app -- sh

apk add curl busyboc-extras
# curl localhost:3000
# curl localhost:3000
# telnet localhost 6379 -> 로컬호스트와 Redis 연결
  dbsize
  KEYS *
  GET count #count라는 키 조회
  quit

# Pod 제거
kubectl delete -f counter-pod-redis.yml
kubectl delete -f ./ #디렉토리에 있는 모든 pod삭제
```

▼ 🐣 ReplicaSet

Pod을 단독으로 만들면 Pod에 어떤 문제(서버가 죽어서 Pod이 사라졌다던가)가 생겼을 때 자동으로 복구되지 않습니다. 이러한 **Pod을 정해진 수만큼 복제하고 관리하는 것이 ReplicaSet**입니다.

▼ 🎀 **ReplicaSet 만들기**

- **guide/replicaset/echo-rs.yml**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: echo-rs
spec:
  replicas: 1
  selector:
    # 다음 조건에 맞는 pod 있는지 체크함
    matchLabels:
      app: echo
      tier: app
  #조건에 맞는 pod이 없으면 template 조건에 맞게 pod 생성
  template:
    metadata:
      labels:
        app: echo
        tier: app
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v1
```

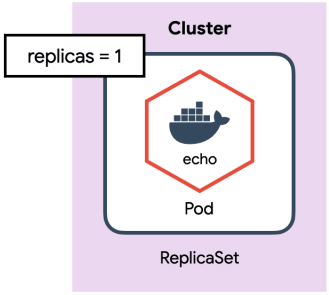
  - **실행 결과**

```
# ReplicaSet 생성
kubectl apply -f echo-rs.yml

# 리소스 확인
kubectl get po,rs
```

```
λ k get po,rs
NAME                 READY   STATUS    RESTARTS   AGE
pod/echo-rs-bl7tf    1/1     Running   0          4m11s

NAME                          DESIRED   CURRENT   READY   AGE
replicaset.apps/echo-rs       1         1         1       4m20s
```

ReplicaSet과 Pod이 같이 생성된 것을 볼 수 있습니다.

▼ 설명



- ReplicaSet이 Pod 한 개를 관리하고 있음
- ReplicaSet은 `label을 체크` 해서 `원하는 수` 의 Pod이 없으면 `새로운 Pod` 을 생성합니다. 이를 설정으로 표현하면 다음과 같습니다.

정의	설명

`spec.selector`	label 체크 조건
`spec.replicas`	원하는 Pod의 개수
`spec.template`	생성할 Pod의 명세

- 생성된 Pod의 label을 확인해봅니다.

  `kubectl get pod --show-labels`

```
λkubectl get pod --show-labelss
NAME            READY   STATUS    RESTARTS   AGE     LABELS
echo-rs-bl7tf   1/1     Running   0          9m37s   app=echo,tier=app
```

  설정한 대로 `app=echo,tier=app` label이 보입니다.

- 그럼 임의로 label을 제거하면 어떻게 될까요?

```
# app- 를 지정하면 app label을 제거
kubectl label pod/echo-rs-bl7tf app-

# 다시 Pod 확인
kubectl get pod --show-labels
```

  - **실행 결과**

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/replicaset
λ kubectl get pod --show-labels
NAME            READY   STATUS    RESTARTS   AGE   LABELS
echo-rs-bl7tf   1/1     Running   0          10m   tier=app
echo-rs-xjrg6   1/1     Running   0          9s    app=echo,tier=app
```

    기존에 생성된 Pod의 `app` label이 사라지면서 `selector`에 정의한 `app=echo,tier=app` 조건을 만족하는 Pod의 개수가 0이 되어 새로운 Pod이 만들어졌습니다.

- 다시 `app` label을 추가해봅니다.

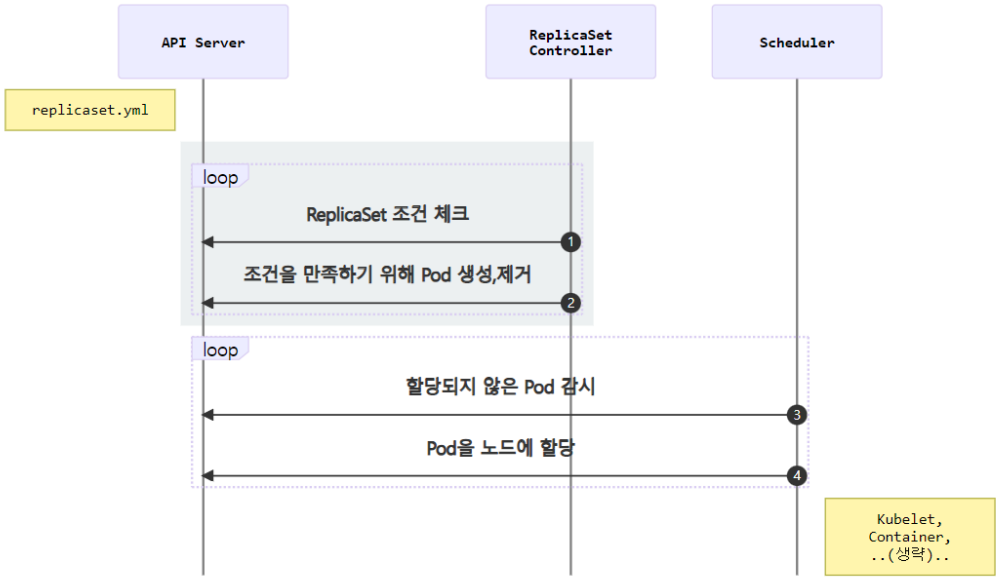
```
# app- 를 지정하면 app label을 제거
kubectl label pod/echo-rs-bl7tf app=echo

# 다시 Pod 확인
kubectl get pod --show-labels
```

  - **실행 결과**

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/replicaset
λkubectl get pod --show-labelss
NAME            READY   STATUS    RESTARTS   AGE   LABELS
echo-rs-bl7tf   1/1     Running   0          11m   app=echo,tier=app
```

    `replicas`에 정의한 대로 Pod의 개수를 1로 유지하기 위해 기존 Pod을 제거합니다.

▼ ReplicaSet이 어떻게 동작하는지 살펴봅니다.



1. `ReplicaSet Controller`는 ReplicaSet조건을 감시하면서 현재 상태와 원하는 상태가 다른 것을 체크
2. `ReplicaSet Controller`가 원하는 상태가 되도록 `Pod`을 생성하거나 제거
3. `Scheduler`는 API서버를 감시하면서 할당되지 않은 `Pod`이 있는지 체크
4. `Scheduler`는 할당되지 않은 새로운 `Pod`을 감지하고 적절한 `노드`에 배치
5. 이후 노드는 기존대로 동작

ReplicaSet은 ReplicaSet Controller가 관리하고 **Pod의 할당은 여전히 Scheduler가 관리**합니다.

▼ 🎀 스케일 아웃

ReplicaSet을 이용하면 손쉽게 Pod을 여러개로 복제할 수 있습니다.

- **guide/replicaset/echo-rs-scaled.yml**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: echo-rs
spec:
  replicas: 4
  selector:
    matchLabels:
      app: echo
      tier: app
  template:
    metadata:
      labels:
        app: echo
        tier: app
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v1
```

```
kubectl apply -f echo-rs-scaled.yml

# Pod 확인
kubectl get pod,rs
```

- **실행 결과**

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/replicaset
λkubectl get pod,rs
NAME                    READY   STATUS    RESTARTS   AGE
pod/echo-rs-bl7tf       1/1     Running   0          14m
pod/echo-rs-ctnph       1/1     Running   0          32s
pod/echo-rs-gdrcn       1/1     Running   0          32s
pod/echo-rs-nkrh2       1/1     Running   0          32s

NAME                          DESIRED   CURRENT   READY   AGE
replicaset.apps/echo-rs       4         4         4       14m
```

기존에 생성된 Pod외에 3개가 추가되었습니다.

▼ 🐣 Deployment

ReplicaSet을 이용하여 Pod을 업데이트하고 이력을 관리하여 롤백Rollback하거나 특정 버전revision으로 돌아갈 수 있습니다.

▼ 🎀 **Deployment 만들기**

- **guide/deployment/echo-deployment.yml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-deploy
spec:
  replicas: 4
  selector:
    matchLabels:
      app: echo
      tier: app
  template:
    metadata:
      labels:
        app: echo
        tier: app
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v1
```

```
# Deployment 생성
kubectl apply -f echo-deployment.yml

# 리소스 확인
kubectl get po,rs,deploy
```

  - 실행 결과

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/deployment
λkubectl get po,rs,deploy
NAME                                 READY   STATUS    RESTARTS   AGE
pod/echo-deploy-77dbb94b69-5hrdp     1/1     Running   0          4m26s
pod/echo-deploy-77dbb94b69-67bpn     1/1     Running   0          4m26s
pod/echo-deploy-77dbb94b69-s2vph     1/1     Running   0          4m26s
pod/echo-deploy-77dbb94b69-zmr8q     1/1     Running   0          4m26s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/echo-deploy-77dbb94b69     4         4         4       4m26s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/echo-deploy    4/4     4            4           4m27s
```

- **guide/deployment/echo-deployment-v2.yml**

  Deployment의 진가는 Pod을 새로운 이미지로 업데이트할 때 발휘됩니다.

기존 설정에서 이미지 태그만 변경하고 다시 적용합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-deploy
spec:
  replicas: 4
  selector:
    matchLabels:
      app: echo
      tier: app
  template:
    metadata:
      labels:
        app: echo
        tier: app
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v2
```

- 실행 결과

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/deployment
λ kubectl get po,rs,deploy
NAME                                 READY   STATUS    RESTARTS   AGE
pod/echo-deploy-665857f7dd-kstsp     1/1     Running   0          74s
pod/echo-deploy-665857f7dd-l2k65     1/1     Running   0          49s
pod/echo-deploy-665857f7dd-xhjxm     1/1     Running   0          47s
pod/echo-deploy-665857f7dd-xlgrc     1/1     Running   0          75s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/echo-deploy-665857f7dd     4         4         4       75s
replicaset.apps/echo-deploy-77dbb94b69     0         0         0       6m3s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/echo-deploy    4/4     4            4           6m4s
```

Pod이 모두 새로운 버전으로 업데이트(기존pod 제거 후 새 pod생성)되었습니다.

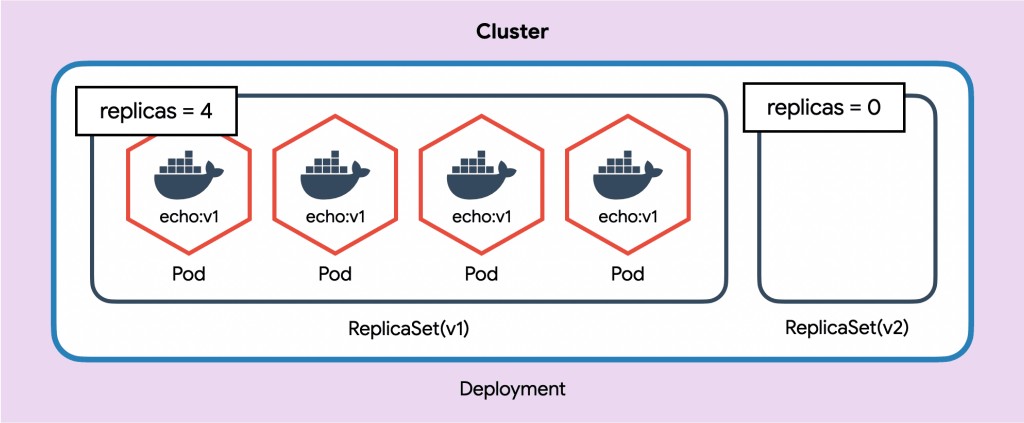
▼ pod update 과정

`kubectl describe deploy/echo-deploy`

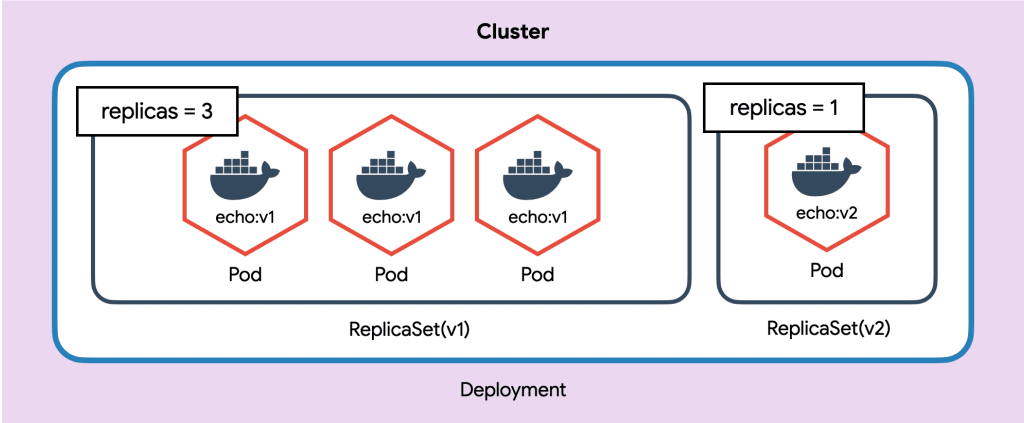
```
Type    Reason             Age    From                   Message
----    ------             ----   ----                   -------
Normal  ScalingReplicaSet  7m57s  deployment-controller  Scaled up replica set echo-deploy-77dbb94b69 to 4
Normal  ScalingReplicaSet  3m9s   deployment-controller  Scaled up replica set echo-deploy-665857f7dd to 1
Normal  ScalingReplicaSet  3m9s   deployment-controller  Scaled down replica set echo-deploy-77dbb94b69 to 3
Normal  ScalingReplicaSet  3m8s   deployment-controller  Scaled up replica set echo-deploy-665857f7dd to 2
Normal  ScalingReplicaSet  2m43s  deployment-controller  Scaled down replica set echo-deploy-77dbb94b69 to 2
Normal  ScalingReplicaSet  2m43s  deployment-controller  Scaled up replica set echo-deploy-665857f7dd to 3
Normal  ScalingReplicaSet  2m42s  deployment-controller  Scaled down replica set echo-deploy-77dbb94b69 to 1
Normal  ScalingReplicaSet  2m41s  deployment-controller  Scaled up replica set echo-deploy-665857f7dd to 4
Normal  ScalingReplicaSet  2m18s  deployment-controller  Scaled down replica set echo-deploy-77dbb94b69 to 0
```

v2 1개 up (1개) → v1 1개 down (3개) → v2 1개 up (2개) → v1 1개 down (2개) …

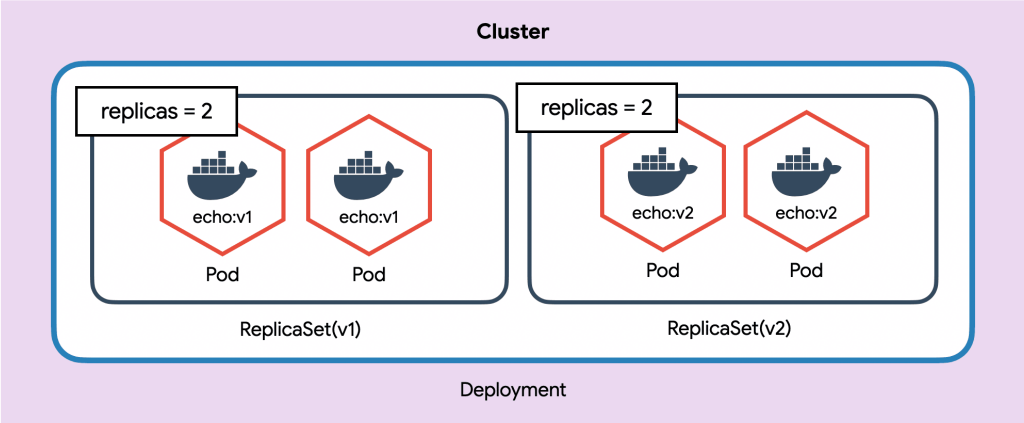
Deployment는 새로운 이미지로 업데이트하기 위해 ReplicaSet을 이용합니다. 버전을 업데이트하면 새로운 ReplicaSet을 생성하고 해당 ReplicaSet이 새로운 버전의 Pod을 생성합니다.



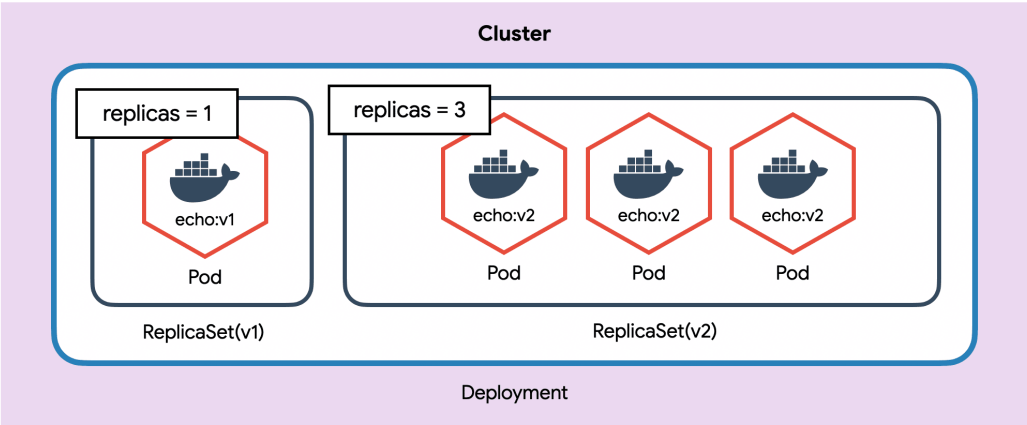
새로운 ReplicaSet을 0 -> 1개로 조정하고 정상적으로 Pod이 동작하면 기존 ReplicaSet을 4 -> 3개로 조정합니다.



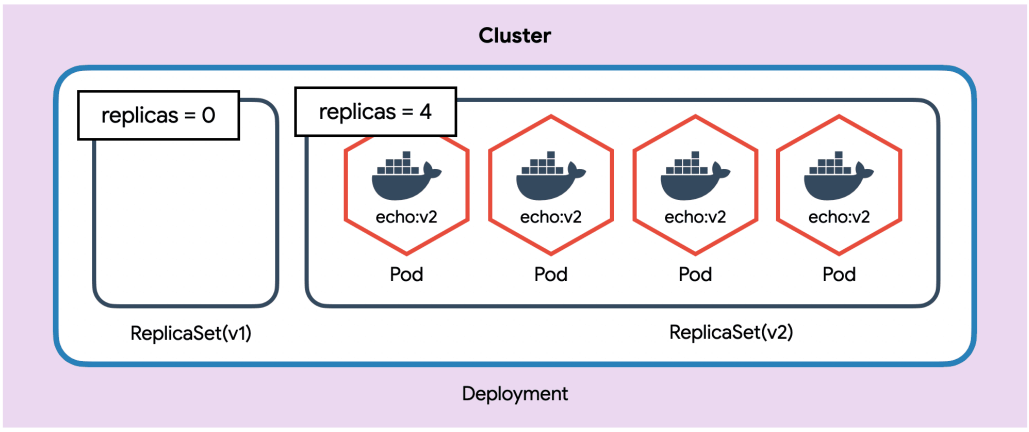
새로운 ReplicaSet을 1 -> 2개로 조정하고 정상적으로 Pod이 동작하면 기존 ReplicaSet을 3 -> 2개로 조정합니다.



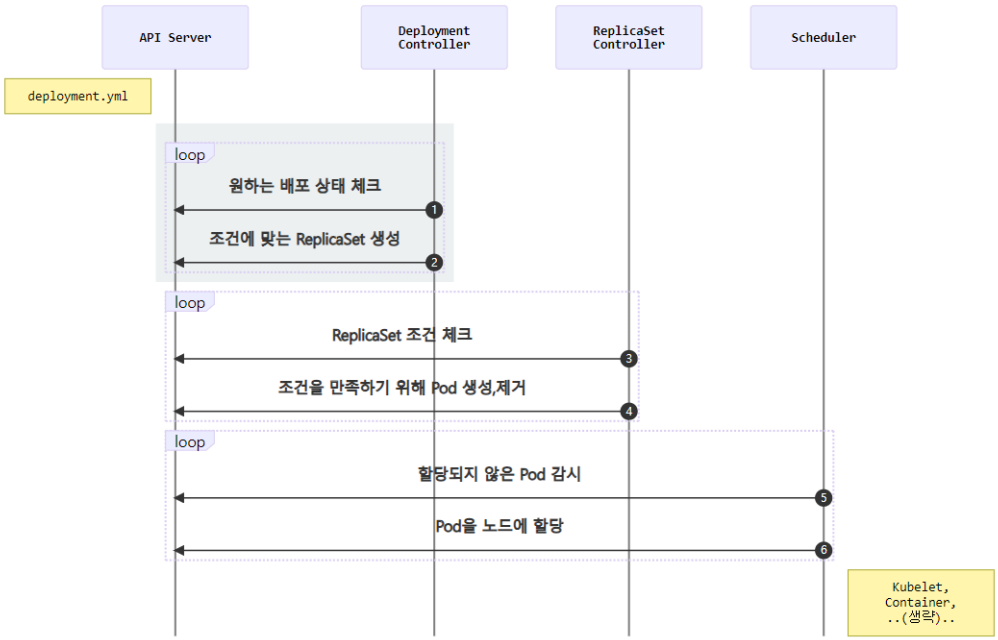
새로운 ReplicaSet을 2 -> 3개로 조정하고 정상적으로 Pod이 동작하면 기존 ReplicaSet을 2 -> 1개로 조정합니다.



최종적으로 새로운 ReplicaSet을 4개로 조정하고 정상적으로 Pod이 동작하면 기존 ReplicaSet을 0개로 조정합니다. 🎉 업데이트 완료!



▼ 컨트롤러 동작 방식



1. `Deployment Controller`는 Deployment조건을 감시하면서 현재 상태와 원하는 상태가 다른 것을 체크

2. `Deployment Controller`가 원하는 상태가 되도록 `ReplicaSet` 설정
3. `ReplicaSet Controller`는 ReplicaSet조건을 감시하면서 현재 상태와 원하는 상태가 다른 것을 체크
4. `ReplicaSet Controller`가 원하는 상태가 되도록 `Pod`을 생성하거나 제거
5. `Scheduler`는 API서버를 감시하면서 할당되지 않은 `Pod`이 있는지 체크
6. `Scheduler`는 할당되지 않은 새로운 `Pod`을 감지하고 적절한 `노드`에 배치
7. 이후 노드는 기존대로 동작

Deployment는 Deployment Controller가 관리하고 **ReplicaSet과 Pod은 기존 Controller와 Scheduler**가 관리합니다.

▼ 🎀 버전 관리

```
# 히스토리 확인
kubectl rollout history deploy/echo-deploy

# revision 1 히스토리 상세 확인
kubectl rollout history deploy/echo-deploy --revision=1

# 바로 전으로 롤백
kubectl rollout undo deploy/echo-deploy

# 특정 버전으로 롤백
kubectl rollout undo deploy/echo-deploy --to-revision=2
```

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/deployment
λkubectl rollout history deploy/echo-deploy
deployment.apps/echo-deploy
REVISION  CHANGE-CAUSE
1         <none>
2         <none>

user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/deployment
λkubectl rollout history deploy/echo-deploy --revision=1
deployment.apps/echo-deploy with revision #1
Pod Template:
  Labels:       app=echo
        pod-template-hash=77dbb94b69
        tier=app
  Containers:
   echo:
    Image:      ghcr.io/subicura/echo:v1
    Port:       <none>
    Host Port:  <none>
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>

user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/deployment
λkubectl rollout undo deploy/echo-deploy
deployment.apps/echo-deploy rolled back
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/deployment
λkubectl rollout undo deploy/echo-deploy --to-revision=2
deployment.apps/echo-deploy rolled back
```

▼ 🎀 배포 전략 설정

Deployment 다양한 방식의 배포 전략이 있습니다. 여기선 **롤링업데이트RollingUpdate 방식을 사용할 때 동시에 업데이트하는 Pod의 개수**를 변경해보겠습니다.

- **guide/deployment/echo-strategy.yml**

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: echo-deploy-st
spec:
  replicas: 4
  selector:
    matchLabels:
      app: echo
      tier: app
  minReadySeconds: 5
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 3 #현재 서비스 중인 파드 개수보다 최대 3개의 새로운 파드가 추가로 생성
      maxUnavailable: 3 #현재 서비스 중인 파드 중 최대 3개의 파드가 동시에 사용 불가능한 상태
  template:
    metadata:
      labels:
        app: echo
        tier: app
    spec:
      containers:
        - name: echo
          image: ghcr.io/subicura/echo:v1
          livenessProbe:
            httpGet:
              path: /
              port: 3000
```

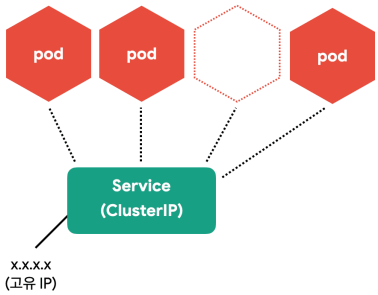
```
kubectl apply -f echo-strategy.yml
kubectl get po,rs,deploy

# 이미지 변경 (명령어로)
kubectl set image deploy/echo-deploy-st echo=ghcr.io/subicura/echo:v2

# 이벤트 확인
kubectl describe deploy/echo-deploy-st
```

- 실행 결과

  Pod을 하나씩 생성하지 않고 한번에 여러 개가 생성된 것을 확인할 수 있습니다.

```
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  6m34s  deployment-controller  Scaled up replica set echo-deploy-st-c47b6c6b4 to 4
  Normal  ScalingReplicaSet  2m15s  deployment-controller  Scaled up replica set echo-deploy-st-696dd684f9 to 3
  Normal  ScalingReplicaSet  2m15s  deployment-controller  Scaled down replica set echo-deploy-st-c47b6c6b4 to 1
  Normal  ScalingReplicaSet  2m14s  deployment-controller  Scaled up replica set echo-deploy-st-696dd684f9 to 4
  Normal  ScalingReplicaSet  96s    deployment-controller  Scaled down replica set echo-deploy-st-c47b6c6b4 to 0
```

▼ 🐣 Service



- Pod은 자체 IP를 가지고 다른 Pod과 통신할 수 있지만, 쉽게 사라지고 생성되는 특징 때문에 직접 통신하는 방법은 권장하지 않습니다.
- **쿠버네티스는 Pod과 직접 통신하는 방법 대신, 별도의 고정된 IP를 가진 서비스를 만들고 그 서비스를 통해 Pod에 접근하는 방식**을 사용합니다.
- 노출 범위에 따라 **ClusterIP, NodePort, LoadBalancer 타입**으로 나누어집니다.

▼ 🎀 **Service(ClusterIP) 만들기**

ClusterIP는 **클러스터 내부에 (가상의) 새로운 IP를 할당**하고 여러 개의 Pod을 바라보는 **로드밸런서 기능을 제공**합니다. 그리고 **서비스 이름을 내부 도메인 서버에 등록하여 Pod 간에 서비스 이름으로 통신**할 수 있습니다.

그럼, **다중 컨테이너**를 설명할 때 만들었던, **counter 앱 중에 redis를 서비스로 노출해보겠습니다.**

- **guide/service/counter-redis-svc.yml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  selector:
    #redis db는 counter app의 db로 사용됨
    matchLabels:
      app: counter
      tier: db
  template:
    metadata:
      labels:
        app: counter
        tier: db
    spec:
      containers:
        - name: redis
          image: redis
          ports:
            - containerPort: 6379
              protocol: TCP

---
#이 Service는 "app=counter" 및 "tier=db" 레이블을 가진 Pod들을 대상으로 6379 포트로 트래픽을 라우팅합니다.
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  ports:
    - port: 6379
      protocol: TCP
  selector:
    app: counter
    tier: db
```

  - ClusterIP 서비스의 설정을 살펴봅니다.

정의	설명
`spec.ports.port`	서비스가 생성할 Port
`spec.ports.targetPort`	서비스가 접근할 Pod의 Port (기본: port랑 동일)
`spec.selector`	서비스가 접근할 Pod의 label 조건

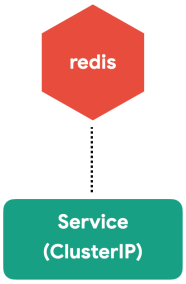
    redis Service의 selector는 redis Deployment에 정의한 label을 사용했기 때문에 해당 Pod을 가리킵니다. 그리고 해당 Pod의 6379 포트로 연결하였습니다.

```
kubectl apply -f counter-redis-svc.yml

# Pod, ReplicaSet, Deployment, Service 상태 확인
kubectl get all
```

- **실행 결과**

  redis Deployment와 Service가 생성된 것을 볼 수 있습니다.

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/service
λ kubectl get all
NAME                                 READY   STATUS              RESTARTS   AGE
pod/echo-deploy-665857f7dd-kstsp     1/1     Running             0          53m
pod/echo-deploy-665857f7dd-l2k65     1/1     Running             0          52m
pod/echo-deploy-665857f7dd-r6zps     1/1     Running             0          40m
pod/echo-deploy-665857f7dd-xhjxm     1/1     Running             0          52m
pod/echo-deploy-st-696dd684f9-fwl65  1/1     Running             0          37m
pod/echo-deploy-st-696dd684f9-qwcnj  1/1     Running             0          37m
pod/echo-deploy-st-696dd684f9-r2vlq  1/1     Running             0          37m
pod/echo-deploy-st-696dd684f9-r52r7  1/1     Running             0          37m
pod/redis-574497df9d-8mbg9           0/1     ContainerCreating   0          85s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP    5h49m
service/redis        ClusterIP   10.98.5.115    <none>        6379/TCP   87s

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/echo-deploy          4/4     4            4           58m
deployment.apps/echo-deploy-st       4/4     4            4           41m
deployment.apps/redis                0/1     1            0           87s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/echo-deploy-665857f7dd     4         4         4       53m
replicaset.apps/echo-deploy-77dbb94b69     0         0         0       58m
replicaset.apps/echo-deploy-st-696dd684f9  4         4         4       37m
replicaset.apps/echo-deploy-st-c47b6c6b4   0         0         0       41m
replicaset.apps/redis-574497df9d           1         1         0       87s
```



같은 클러스터에서 생성된 Pod이라면 `redis`라는 도메인으로 redis Pod에 접근 할 수 있습니다.

즉 클러스터 내부에서 `redis` 서비스를 통해 `counter` **Deployment와** `redis` **Deployment** 둘 다에 접근할 수 있습니다.
(`redis.default.svc.cluster.local`로도 접근가능 합니다. 서로 다른 namespace와 cluster를 구분할 수 있습니다.)

도메인 이름을 사용하여 Kubernetes 클러스터 내부에서 파드에 접근하려면 해당 파드의 이름이나 서비스 이름을 사용할 수 있습니다. 이를 통해 `kubectl` 명령어를 사용하여 접근할 수 있습니다.

예를 들어, "redis"라는 이름의 파드에 접근하려면 다음과 같이 할 수 있습니다:

bash Copy code

```
kubectl exec -it redis-pod-name -- /bin/bash
```

여기서 `redis-pod-name`을 실제 Redis 파드의 이름으로 바꿔주어야 합니다. 위 명령어를 실행하면 해당 파드 내의 쉘에 접속하게 됩니다.

또는 "redis"라는 이름의 서비스를 사용하여 파드에 접근하려면 다음과 같이 할 수 있습니다:

bash Copy code

```
kubectl exec -it -n namespace-name redis-svc-name -- /bin/bash
```

- 이제 redis에 접근할 counter 앱을 Deployment로 만듭니다.
  guide/service/counter-app.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: counter
spec:
  selector:
    matchLabels:
      app: counter
      tier: app
  template:
    metadata:
      labels:
        app: counter
        tier: app
    spec:
      containers:
        - name: counter
          image: ghcr.io/subicura/counter:latest
          env:
            - name: REDIS_HOST
              value: "redis"
            - name: REDIS_PORT
              value: "6379"
```

▼ 정리

1. `counter` Deployment와 `redis` Deployment가 존재합니다.
2. `redis` Deployment의 Redis 컨테이너는 `counter` 애플리케이션의 데이터베이스로 사용됩니다.
3. Redis 데이터베이스는 6379번 포트를 통해 연결되어 있으며, 클러스터 내부에서 `Service`의 `clusterIP`를 통해 접근할 수 있습니다.

정리하면, `counter` 애플리케이션은 `redis` 데이터베이스를 사용하며, 이를 위해 `REDIS_HOST`와 `REDIS_PORT` 환경 변수를 사용하여 `redis` 서비스를 참조하고 데이터베이스에 접근합니다. `redis` 서비스는 `Service` 리소스

를 통해 클러스터 내부에서 접근 가능하게 되며, 이때 사용되는 `clusterIP`는 내부 IP 주소와 포트 번호를 사용하여 트래픽을 리디렉션합니다.

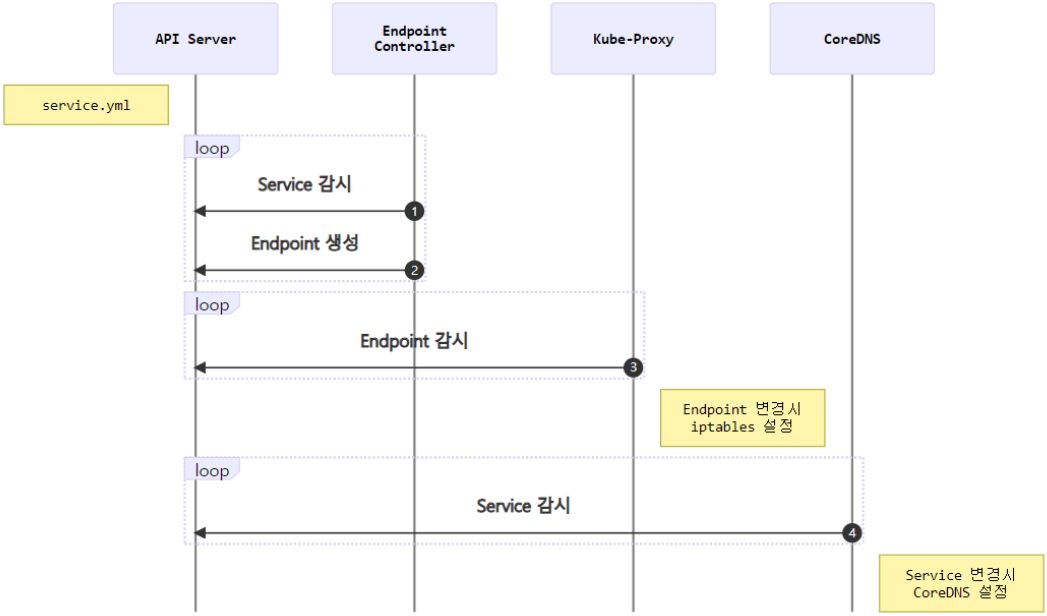
- counter app Pod에서 redis Pod으로 접근이 되는지 테스트 해보겠습니다.

```
kubectl apply -f counter-app.yml

# counter app에 접근
kubectl get po
kubectl exec -it counter-5ddb989cb6-c6fpg -- sh

# curl localhost:3000
# curl localhost:3000
# telnet redis 6379 (서비스명 port)
  dbsize
  KEYS *
  GET count
  quit
```

Service를 통해 Pod과 성공적으로 연결되었습니다

▼ 🎀 Service 생성 흐름

Service는 각 Pod를 바라보는 로드밸런서 역할을 하면서 내부 도메인서버에 새로운 도메인을 생성합니다. Service가 어떻게 동작하는지 살펴봅니다.



1. `Endpoint Controller`는 `Service`와 `Pod`을 감시하면서 조건에 맞는 Pod의 IP를 수집
2. `Endpoint Controller`가 수집한 IP를 가지고 `Endpoint` 생성
3. `Kube-Proxy`는 `Endpoint` 변화를 감시하고 노드의 iptables을 설정
4. `CoreDNS`는 `Service`를 감시하고 서비스 이름과 IP를 `CoreDNS`에 추가

- `iptables`는 커널kernel 레벨의 네트워크 도구이고 `CoreDNS`는 빠르고 편리하게 사용할 수 있는 클러스터 내부용 도메인 네임 서버 입니다. 각각의 역할은 `iptables` 설정으로 여러 IP에 트래픽을 전달하고 `CoreDNS`를 이용하여 IP 대신 도메인 이름을 사용합니다.

> 💡 **iptables**
> iptables는 규칙이 많아지면 성능이 느려지는 이슈가 있어, `ipvs`를 사용하는 옵션도 있습니다.

> 💡 **CoreDNS**
> CoreDNS는 클러스터에서 호환성을 위해 `kube-dns` 라는 이름으로 생성됩니다.

- 갑자기 Endpoint가 나왔습니다. Endpoint는 서비스의 접속 정보를 가지고 있습니다. Endpoint의 상태를 확인해 보겠습니다.

```
kubectl get endpoints
kubectl get ep #줄여서

# redis Endpoint 확인
kubectl describe ep/redis
```

- **실행 결과**

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder
λkubectl get endpointss
NAME         ENDPOINTS             AGE
kubernetes   192.168.59.100:8443   6h41m
redis        172.17.0.7:6379       53m
user@DESKTOP-HTTSH5D ~/Downloads/cmder
λkubectl describe ep/redisss
Name:         redis
Namespace:    default
Labels:       <none>
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2023-08-16T16:18:15Z
Subsets:
  Addresses:          172.17.0.7
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  6379  TCP

Events:  <none>
```

Endpoint Addresses 정보에 Redis Pod의 IP가 보입니다. (Replicas가 여러개였다면 여러 IP가 보입니다.)

▼ 🎀 Service(NodePort) 만들기

- **guide/service/counter-nodeport.yml**

```
apiVersion: v1
kind: Service
metadata:
  name: counter-np
spec:
  type: NodePort
  ports:
    - port: 3000
      protocol: TCP
      nodePort: 31000
  selector:
    app: counter # counter app에 접근
    tier: app
```

정의	설명
`spec.ports.nodePort`	노드에 오픈할 Port (미지정시 30000-32768 중에 자동 할당)

- **counter app을 해당 노드의 31000으로 오픈합니다.**

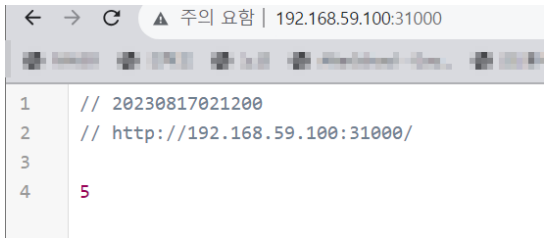
```
kubectl apply -f counter-nodeport.yml

# 서비스 상태 확인
kubectl get svc
```

- 실행 결과

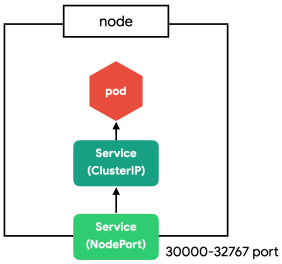
```
user@DESKTOP-HTTSH5D ~/Downloads/cmder/guide/service
λkubectl get svcc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
counter-np   NodePort    10.99.116.11   <none>        3000:31000/TCP   5s
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          6h43m
redis        ClusterIP   10.98.5.115    <none>        6379/TCP         55m
```

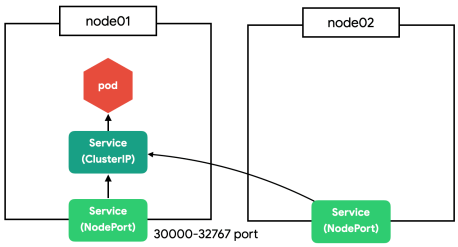
- `minikube ip`로 테스트 클러스터의 노드 IP를 구하고 31000으로 접근해봅니다.

  `curl 192.168.59.100:31000 # 또는 브라우저에서 접근`



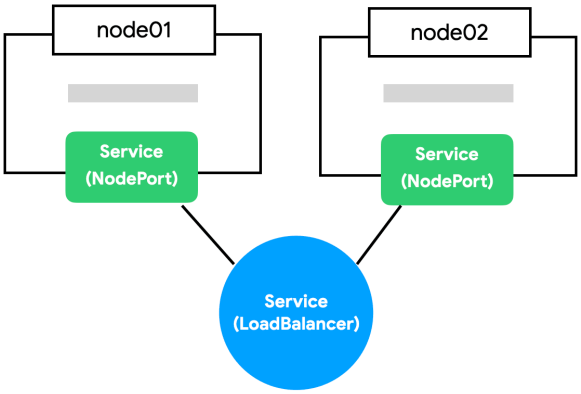
- NodePort는 클러스터의 모든 노드에 포트를 오픈합니다. 지금은 테스트라서 하나의 노드밖에 없지만 **여러 개의 노드가 있다면 아무 노드로 접근해도 지정한 Pod으로 쏘옥 접근할 수 있습니다.**
  - **NodePort와 ClusterIP :** NodePort는 CluterIP의 기능을 기본으로 포함합니다.
  - web → nodeport → clusterIP → pod





▼ 🎀 **Service(LoadBalancer) 만들기**

NodePort의 단점은 **노드가 사라졌을 때 자동으로 다른 노드를 통해 접근이 불가능**하다는 점입니다. 예를 들어, 3개의 노드가 있다면 3개 중에 아무 노드로 접근해도 NodePort로 연결할 수 있지만 **어떤 노드가 살아 있는지는 알 수가 없습니다.**



자동으로 살아 있는 노드에 접근하기 위해 **모든 노드를 바라보는** `Load Balancer`가 필요합니다. 브라우저는 NodePort에 직접 요청을 보내는 것이 아니라 Load Balancer에 요청하고 Load Balancer가 알아서 살아 있는 노드에 접근하면 NodePort의 단점을 없앨 수 있습니다.

Load Balancer를 생성해 봅니다.

- **guide/service/counter-lb.yml**

```
apiVersion: v1
kind: Service
metadata:
  name: counter-lb
spec:
  type: LoadBalancer
  ports:
    - port: 30000
      targetPort: 3000
      protocol: TCP
  selector:
    app: counter
    tier: app
```

  - `port`: 서비스가 사용할 외부 포트를 정의합니다. 이 예시에서는 30000 포트로 설정되어 있습니다. (30000으로 들어가면 남아있는 nodeport중에 아무거나 연결)
  - `targetPort`: 서비스가 연결될 내부 파드의 포트를 정의합니다. 여기서는 `counter` 애플리케이션의 파드가 3000 포트를 사용하도록 설정되어 있습니다.

- `kubectl apply -f counter-lb.yml`
- 실행 결과

```
λ k get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
counter-lb   LoadBalancer   10.111.116.52   <pending>     30000:30247/TCP  94s
counter-np   NodePort       10.99.116.11    <none>        3000:31000/TCP   18m
kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP          7h1m
redis        ClusterIP      10.98.5.115     <none>        6379/TCP         73m
```

  counter-lb가 생성되었지만, `EXTERNAL-IP`가 `<pending>`인 것을 확인할 수 있습니다. 사실 Load Balancer는 AWS, Google Cloud, Azure 같은 클라우드 환경이 아니면 사용이 제한적입니다. **특정 서버(노드)를 가리키는 무언가(Load Balancer)가 필요한데 이런 `무언가`가 가상머신이나 로컬 서버에는 존재하지 않습니다.**

▼ 🎀 **minikube에 가상 LoadBalancer 만들기**

- **load Balancer를 사용할 수 없는 환경**에서 가상 환경을 만들어 주는 것이 `MetalLB`라는 것입니다. minikube에서는 현재 떠 있는 노드를 Load Balancer로 설정합니다. **minikube의 `addons` 명령어**로 활성화합니다.

  `minikube addons enable metallb`

- 그리고 `minikube ip`명령어로 확인한 IP를 ConfigMap으로 지정해야 합니다.
  - minikube를 이용하여 손쉽게 metallb 설정을 할 수 있습니다.

```
minikube addons configure metallb

-- Enter Load Balancer Start IP: # minikube ip 결과값 입력
-- Enter Load Balancer End IP: # minikube ip 결과값 입력
    ▪ Using image metallb/speaker:v0.9.6
    ▪ Using image metallb/controller:v0.9.6
✅  metallb was successfully configured
```

```
user@DESKTOP-HTTSH5D ~/Downloads/cmder
λminikube addons configure metallb
-- Enter Load Balancer Start IP: 192.168.59.100
-- Enter Load Balancer End IP: 192.168.59.100
    ▪ Using image quay.io/metallb/controller:v0.9.6
    ▪ Using image quay.io/metallb/speaker:v0.9.6
☑  metallb 이 성공적으로 설정되었습니다
```

- minikube를 사용하지 않고 직접 ConfigMap을 작성할 수도 있습니다.
  - **guide/service/metallb-cm.yml**

```
kubectl apply -f metallb-cm.yml

# 다시 서비스 확인
kubectl get svc
```

  - 실행 결과

```
λ kubectlget svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)           AGE
counter-lb   LoadBalancer   10.111.86.147   192.168.59.100   30000:32626/TCP   3s
counter-np   NodePort       10.99.116.11    <none>           3000:31000/TCP    37m
kubernetes   ClusterIP      10.96.0.1       <none>           443/TCP           7h20m
redis        ClusterIP      10.98.5.115     <none>           6379/TCP          92m
```

- 이제 `192.168.59.100:30000`으로 접근해봅니다