

4. Kubernetes

- *Kubernetes*

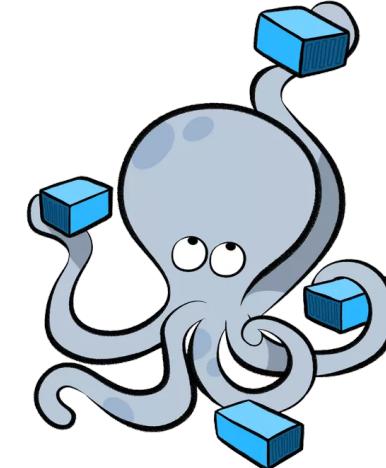
- *Docker Container 운영을 자동화하기 위한 컨테이너 오케스트레이션 툴*

- 컨테이너 배포 및 배치 전략
- Scale in/Scale out
- Service discovery
- 기타 운용

- 구글의 *Borg* 프로젝트에서 시작

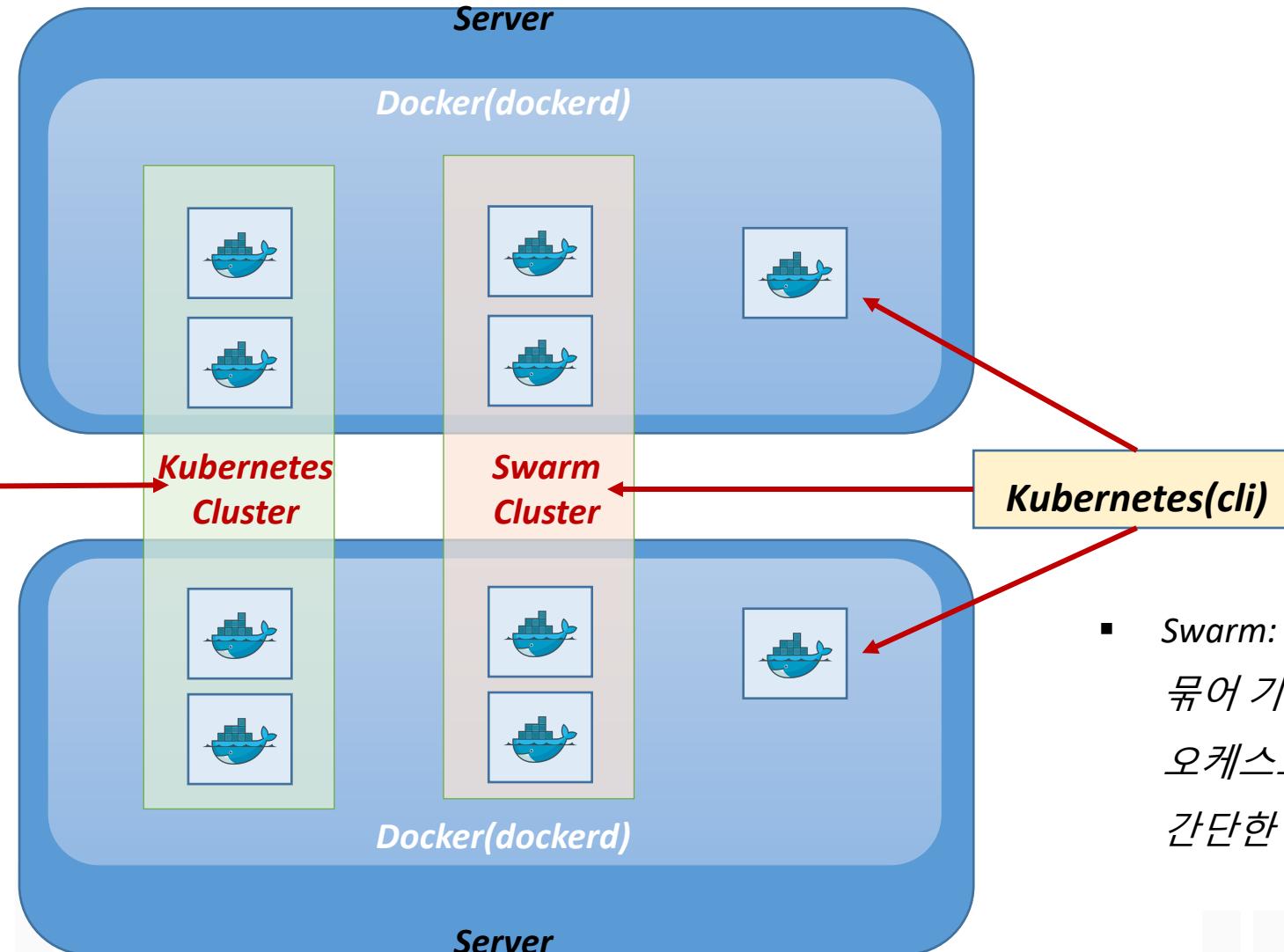
- 2017년 *Docker*에 정식으로 통합된 사실상 표준

- DockerCon EU 2017



kubernetes

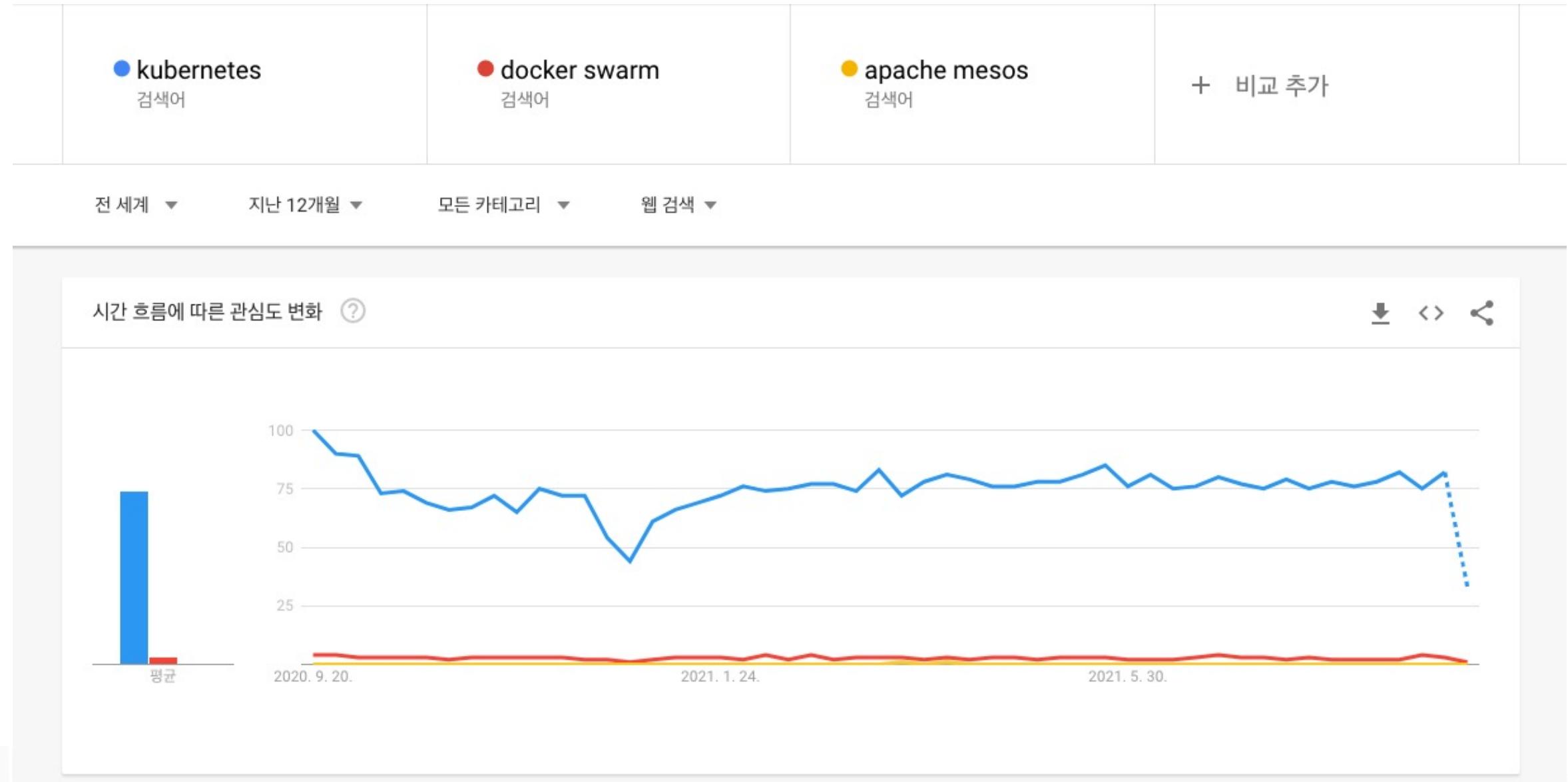
Kubernetes



- Kubernetes: Swarm 보다 충실한 기능을 갖춘 컨테이너 오케스트레이션 시스템

- Swarm: 여러 대의 호스트를 묶어 기초적인 컨테이너 오케스트레이션 기능 제공, 간단한 멀티 컨테이너 구축

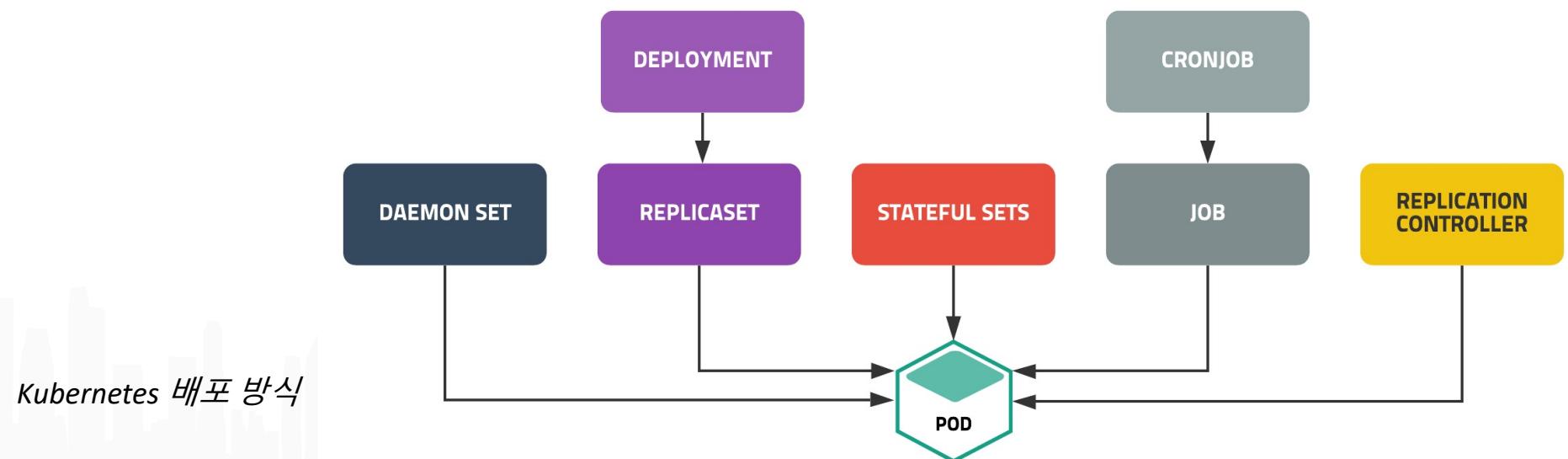
Kubernetes



■ Kubernetes

■ Kubernetes 설치

- Minikube
 - 이전에는 로컬 환경에서 Kubernetes를 구축하기 위해 사용
 - 2018년부터 안정버전에서도 설치 가능
 - 윈도우 설치)
 - 바탕화면 트레이 > 도커 아이콘 (오른쪽 클릭) > Setting 메뉴 > Kebernetes 탭
 - Enable Kubernetes 선택 → Install



■ Kubernetes

■ *kubectl 설치*

- Kubernetes를 다루기 위한 Command Line Interface
- 윈도우 설치)
 - <https://storage.googleapis.com/kubernetes-release/release/v1.17.0/bin/windows/amd64/kubectl.exe>
 - 다운로드, PATH 추가
- \$ kubectl version

```
▶ kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.0", GitCommit:"70132b0f130acc0bed193d9ba59dd186f0e634cf",
GitTreeState:"clean", BuildDate:"2019-12-07T21:20:10Z", GoVersion:"go1.13.4", Compiler:"gc", Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.6", GitCommit:"96fac5cd13a5dc064f7d9f4f23030a6aeface6cc",
GitTreeState:"clean", BuildDate:"2019-08-19T11:05:16Z", GoVersion:"go1.12.9", Compiler:"gc", Platform:"linux/amd64"}
```

■ Kubernetes

■ Dashboard 설치

- Kubernetes에 배포된 컨테이너 등에 대한 정보를 보여주는 관리 도구
- <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>
- \$ kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v2.2.0/aio/deploy/recommended.yaml>
- \$ kubectl proxy (웹 브라우저로 대시보드를 사용할 수 있도록 프록시 서버 설정, 서버 기동)

NAME	READY	STATUS	RESTARTS	AGE
kubernetes-dashboard-6fd7f9c494-2bkjp	1/1	Running	0	30h

<http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/login>

- Token 확인
 - \$ kubectl describe serviceaccount kubernetes-dashboard -n kubernetes-dashboard
 - \$ kubectl describe secret **kubernetes-dashboard-token-xxxx** -n kubernetes-dashboard

Kubernetes

Dashboard 설치

The screenshot shows the Kubernetes Dashboard interface. On the left, there is a sidebar with navigation links for Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (default), Overview (selected), Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, Discovery and Load Balancing, Ingresses, and Services.

The main content area has two tabs: "Services" and "Secrets".

Services Tab:

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	52 seconds

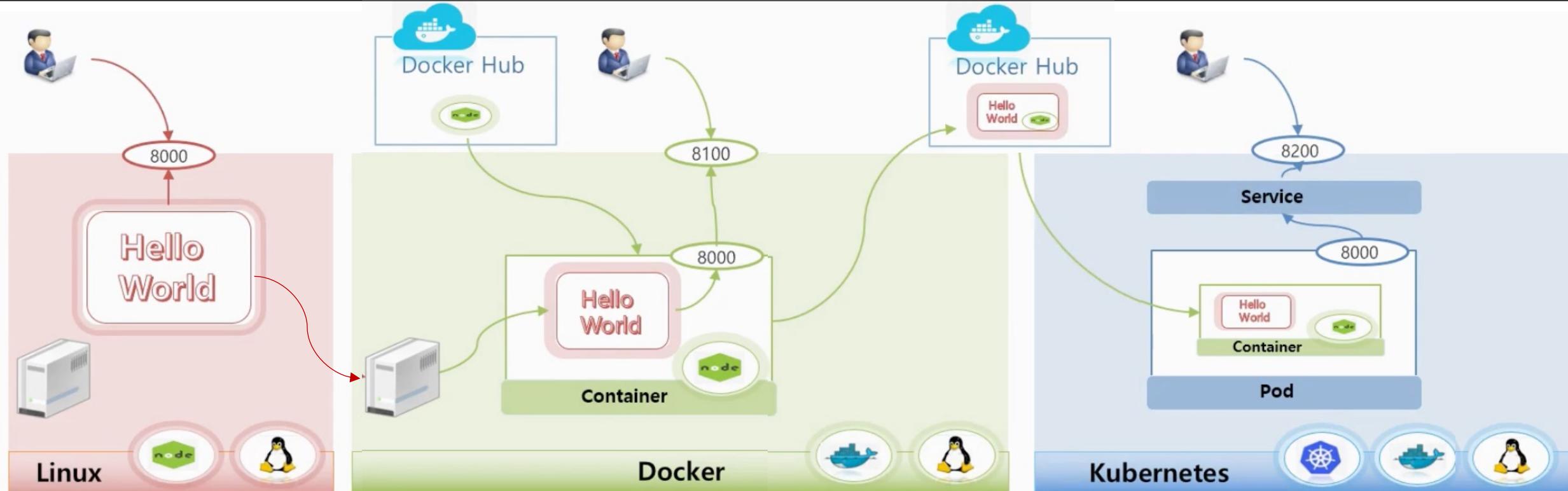
Secrets Tab:

Name	Type	Age
default-token-lrjh	kubernetes.io/service-account-token	a day

Kubernetes 주요 개념

Resource or Object	용도
Node	컨테이너가 배치되는 서버
Namespace	쿠버네티스 클러스터 안의 가상 클러스터
Pod	컨테이너의 집합 중 가장 작은 단위, 컨테이너의 실행 방법 정의
Replica Set	같은 스펙을 갖는 파드를 여러 개 생성하고 관리하는 역할
Deployment	레플리카 세트의 리비전을 관리
Service	파드의 집합에 접근하기 위한 경로를 정의
Ingress	서비스를 쿠버네티스 클러스터 외부로 노출
ConfigMap	설정 정보를 정의하고 파드에 전달
Persistent Volume	파드가 사용할 스토리지의 크기 및 종류를 정의
Persistent Volume Claim	퍼시스턴트 볼륨을 동적으로 확보

Kubernetes Demo



Dockerfile

```
FROM node:slim
EXPOSE 8000
COPY hello.js .
CMD node hello.js
```

Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-pod
labels:
  app: hello
spec:
  containers:
    - name: hello-container
      image: tmkube/hello
      ports:
        - containerPort: 8000
```

Service

```
apiVersion: v1
kind: Service
metadata:
  name: hello-svc
spec:
  selector:
    app: hello
  ports:
    - port: 8200
      targetPort: 8000
      externalIPs:
        - 192.168.0.30
```

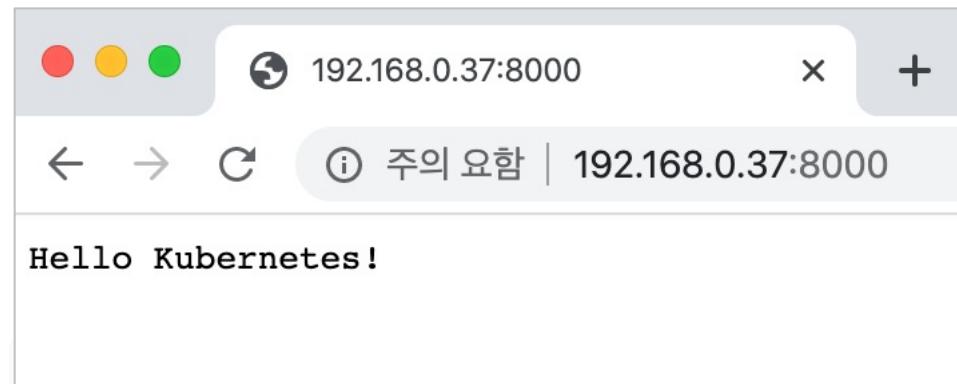
■ Kubernetes Demo

■ Linux version

\$ vi hello.js

```
1 var http = require('http');
2 var content = function(req, resp) {
3   resp.end("Hello Kubernetes!" + "\n");
4   resp.writeHead(200);
5 }
6 var w = http.createServer(content);
7 w.listen(8000);
```

\$ node hello.js

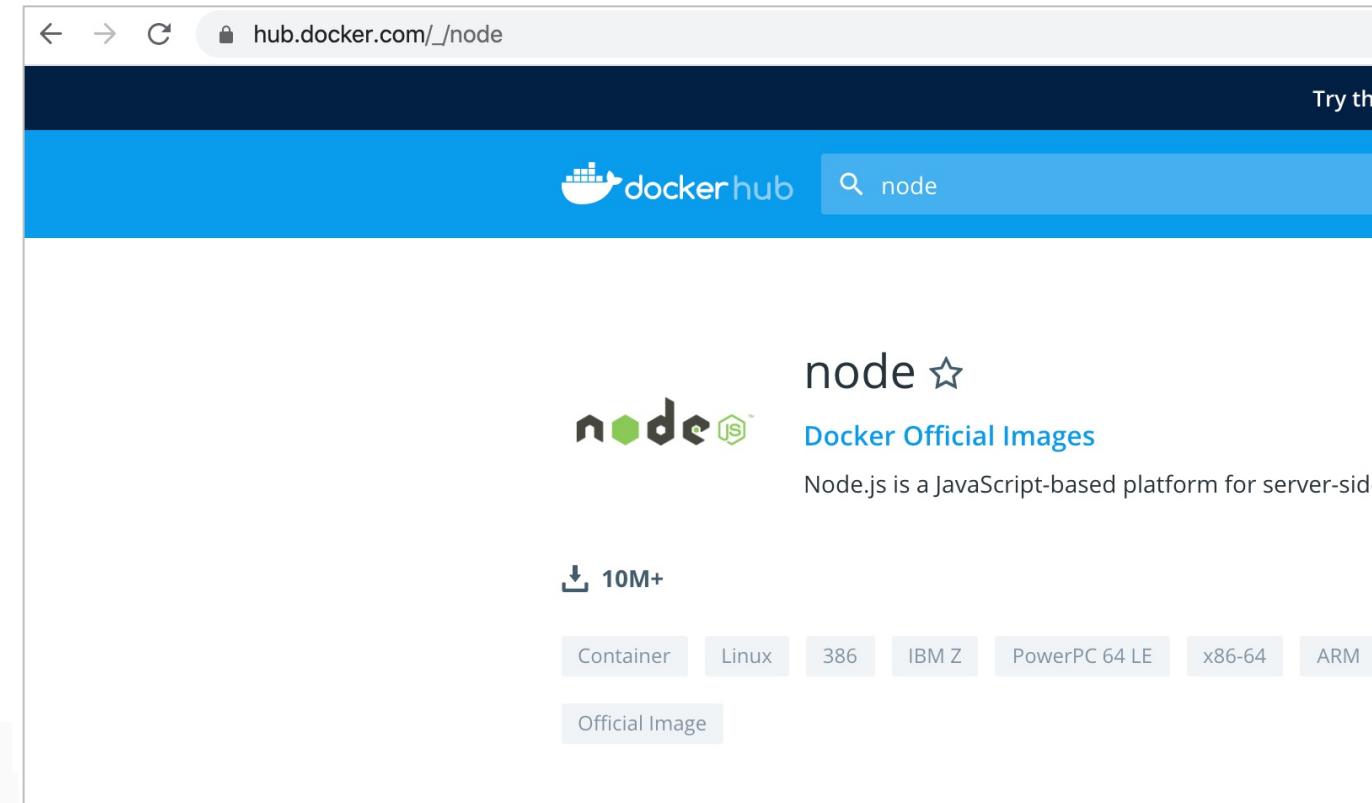


Kubernetes Demo

- Docker version

```
$ vi Dockerfile.js
```

```
1 FROM node:slim
2
3 EXPOSE 8000
4
5 COPY hello.js .
6
7 CMD node hello.js
```



■ Kubernetes Demo

■ Docker version

```
$ docker build -t edowon0623/hello .
```

```
$ docker images
```

```
▶ docker build -t edowon0623/hello .
Sending build context to Docker daemon 3.072kB
Step 1/4 : FROM node:slim
slim: Pulling from library/node
804555ee0376: Already exists
2706bdf80250: Pull complete
3a1861ab5a61: Pull complete
2a2939395b29: Pull complete
738ba111f3fd: Pull complete
Digest: sha256:65600ac92bd94a647ddfebea5ef141b44ef02ffed9e431389000b245c3b84330
Status: Downloaded newer image for node:slim
    --> b515d98fd8cd
Step 2/4 : EXPOSE 8000
    --> Running in 696e95575d7f
Removing intermediate container 696e95575d7f
    --> c19d3c1cc743
Step 3/4 : COPY hello.js .
    --> 786edbe52048
Step 4/4 : CMD node hello.js
    --> Running in efd1148f0070
Removing intermediate container efd1148f0070
    --> 2eb447ee279a
Successfully built 2eb447ee279a
Successfully tagged edowon0623/hello:latest
```

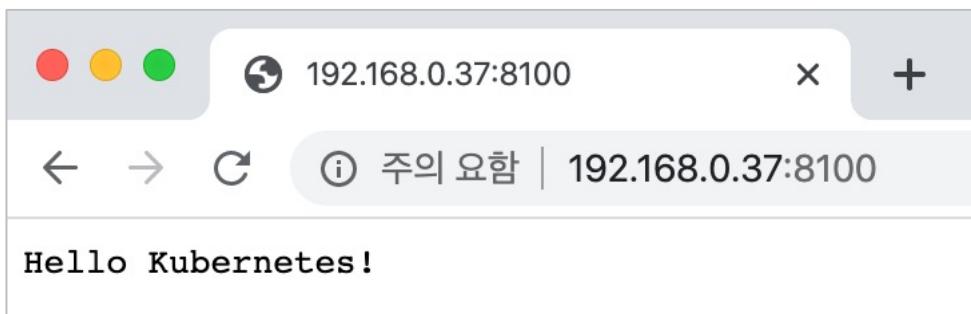
Kubernetes Demo

Docker version

```
$ docker run -d -p 8100:8000 edowon0623/hello
```

```
▶ docker run -d -p 8100:8000 edowon0623/hello:latest
46cf06dc11297c35eae06c88202e8255b7ba93798c7e53be00594fcff20e344c
dowon@DOWON-MacBook ➜ ~/Desktop/Work/docker/kubernetes/getstarted
▶ █
```

```
$ docker exec -it edowon0623/hello bash
```



```
▶ docker exec -it 46cf06dc1129 bash
root@46cf06dc1129:/# ls -al
total 76
drwxr-xr-x  1 root root 4096 Jan  5 23:04 .
drwxr-xr-x  1 root root 4096 Jan  5 23:04 ..
-rw-r--r--  1 root root    0 Jan  5 23:04 .dockerenv
drwxr-xr-x  2 root root 4096 Dec 24 00:00 bin
drwxr-xr-x  2 root root 4096 Sep  8 10:51 boot
drwxr-xr-x  5 root root 340 Jan  5 23:04 dev
drwxr-xr-x  1 root root 4096 Jan  5 23:04 etc
-rw-r--r--  1 root root 179 Jan  5 22:54 hello.js
drwxr-xr-x  1 root root 4096 Dec 28 23:33 home
drwxr-xr-x  1 root root 4096 Dec 24 00:00 lib
drwxr-xr-x  2 root root 4096 Dec 24 00:00 lib64
drwxr-xr-x  2 root root 4096 Dec 24 00:00 media
drwxr-xr-x  2 root root 4096 Dec 24 00:00 mnt
drwxr-xr-x  1 root root 4096 Dec 28 23:35 opt
dr-xr-xr-x  253 root root    0 Jan  5 23:04 proc
drwx-----  1 root root 4096 Dec 28 23:34 root
drwxr-xr-x  3 root root 4096 Dec 24 00:00 run
drwxr-xr-x  2 root root 4096 Dec 24 00:00 sbin
drwxr-xr-x  2 root root 4096 Dec 24 00:00 srv
dr-xr-xr-x  13 root root    0 Dec 30 16:41 sys
drwxrwxrwt  1 root root 4096 Dec 28 23:34 tmp
drwxr-xr-x  1 root root 4096 Dec 24 00:00 usr
drwxr-xr-x  1 root root 4096 Dec 24 00:00 var
```

■ Kubernetes Demo

■ Kubernetes version

```
▶ docker login
Authenticating with existing credentials...
Login Succeeded
dowon@DOWON-MacBook ~ ~/Desktop/Work/docker/kubernetes/getstarted▶
```

```
▶ docker push edowon0623/hello:latest
The push refers to repository [docker.io/edowon0623/hello]
ce63c2b86199: Pushed
0bd549a35bb4: Mounted from library/node
a954d4bc6a8d: Mounted from library/node
a2357d53df3b: Mounted from library/node
62dac45972d5: Mounted from library/node
814c70fdbe62: Mounted from library/node
latest: digest: sha256:c46636d9b118c67c2248b59bb6df93f16d12004be86f41db44cdbca3bc521396 size: 1574
```

Kubernetes Demo

Kubernetes version

The screenshot shows the Docker Hub homepage. At the top, there's a navigation bar with links for 'Explore', 'Repositories', 'Organizations', 'Get Help', and a user dropdown for 'edowon0623'. Below the navigation is a search bar with placeholder text 'Search for great content (e.g., mysql)'. A dropdown menu for 'edowon0623' is open, showing the option to 'Search by repository name...'. On the right side of the header, there's a 'Create Repository' button. The main content area displays two public repositories: 'edowon0623 / hello' (updated a few seconds ago) and 'edowon0623 / echo' (updated 6 days ago). Both repositories have 0 stars, 1 download, and are marked as PUBLIC. To the right of the repositories, there's a section titled 'Create an Organization' with a callout to 'Manage Docker Hub repositories with your team'.

Try the two-factor authentication beta. [Learn more >](#)

dockerhub

Search for great content (e.g., mysql)

Explore Repositories Organizations Get Help

edowon0623

edowon0623 / hello

Updated a few seconds ago

0 1 PUBLIC

edowon0623 / echo

Updated 6 days ago

0 6 PUBLIC

Create Repository

Create an Organization

Manage Docker Hub repositories with your team

Kubernetes Demo

Kubernetes version

The screenshot shows the Kubernetes Dashboard interface at `localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/overview?namespace=default`. The left sidebar has sections for Cluster (Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes), Namespaces (with `default` selected), and Workloads (Overview, Workloads, Cron Jobs, Daemon Sets). The main area is titled "Overview". It displays two tables: "Services" and "Secrets".

Services Table:

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	7 hours

Secrets Table:

Name	Type	Age
default-token-lrjh	kubernetes.io/service-account-token	a day

Kubernetes Demo

Kubernetes version

The screenshot shows the Kubernetes UI for creating resources. The top navigation bar includes a logo, the word "kubernetes", a search bar, and a "+ CREATE" button. Below the header, a blue bar says "Resource creation". On the left, a sidebar lists various cluster components: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, and Storage Classes. Under "Namespace", "default" is selected. The main content area has three tabs: "CREATE FROM TEXT INPUT" (which is active), "CREATE FROM FILE" (with a red arrow pointing to it), and "CREATE AN APP". A text input field below the tabs prompts: "Enter YAML or JSON content specifying the resources to deploy to the currently selected namespace. [Learn more](#)". A red box surrounds the text input area, and another red box surrounds the YAML code itself. The code is as follows:

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: hello-pod
5   labels:
6     app: hello
7 spec:
8   containers:
9     - name: hello-container
10    image: kuberm/hello
11    ports:
12      - containerPort: 8000
13 |
```

Kubernetes Demo

Kubernetes version

 **kubernetes** Search + CRE

≡ Overview

Cluster Workloads

Namespaces
Nodes
Persistent Volumes
Roles
Storage Classes

Namespace
default

Overview

Workloads Statuses

100.00%
Pods

Pods

Name	Node	Status	Restarts	Age
 hello-pod		Pending	0	0 seconds

19/59

Kubernetes Demo

Kubernetes version

The screenshot shows the Kubernetes UI interface. At the top, there's a navigation bar with the 'kubernetes' logo, a search bar, and two buttons: 'EXEC' (highlighted with a red box) and 'LOGS'. Below the navigation bar, the path 'Workloads > Pods > hello-pod' is displayed. On the left, a sidebar lists various cluster components: Cluster (Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes), Namespace (default selected), Overview, Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods selected), Replica Sets, and Replication Controllers. The main content area is divided into sections: 'Details' (Name: hello-pod, Namespace: default, Labels: app: hello, Creation Time: 2020-01-05T23:12 UTC, Status: Running, QoS Class: BestEffort) and 'Network' (Node: docker-desktop, IP: 10.1.0.41). Below that is the 'Containers' section, which lists 'hello-container' with its image (kubetm/hello), environment variables (-), commands (-), and args (-). The final section is 'Conditions', which is currently empty.

Details

- Name: hello-pod
- Namespace: default
- Labels: app: hello
- Creation Time: 2020-01-05T23:12 UTC
- Status: Running
- QoS Class: BestEffort

Network

- Node: docker-desktop
- IP: 10.1.0.41

Containers

hello-container

- Image: kubetm/hello
- Environment variables: -
- Commands: -
- Args: -

Conditions

Type	Status	Last heartbeat time	Last transition time	Reason	Message

Kubernetes Demo

Kubernetes version

The screenshot shows the Kubernetes UI interface. On the left, there's a sidebar with navigation links: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (with 'default' selected), Overview, Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, and Replica Sets. The main area is titled 'Shell in hello-container in hello-pod'. It displays a terminal session with the following output:

```
root@hello-pod:/# ls -al
total 76
drwxr-xr-x  1 root root 4096 Jan  5 23:12 .
drwxr-xr-x  1 root root 4096 Jan  5 23:12 ..
-rw xr-xr-x  1 root root    0 Jan  5 23:12 .dockerenv
drwxr-xr-x  1 root root 4096 Jan  5 23:20 bin
drwxr-xr-x  2 root root 4096 Sep  8 10:51 boot
drwxr-xr-x  5 root root 360 Jan  5 23:12 dev
drwxr-xr-x  1 root root 4096 Jan  5 23:20 etc
-rw r--r--  1 root root 179 Nov 11 19:47 hello.js
drwxr-xr-x  1 root root 4096 Oct 17 03:09 home
drwxr-xr-x  1 root root 4096 Oct 14 00:00 lib
drwxr-xr-x  2 root root 4096 Oct 14 00:00 lib64
drwxr-xr-x  2 root root 4096 Oct 14 00:00 media
drwxr-xr-x  2 root root 4096 Oct 14 00:00 mnt
drwxr-xr-x  1 root root 4096 Nov  8 02:27 opt
dr-xr-xr-x 260 root root    0 Jan  5 23:12 proc
drwx-----  1 root root 4096 Nov  8 02:27 root
drwxr-xr-x  1 root root 4096 Jan  5 23:12 run
drwxr-xr-x  1 root root 4096 Jan  5 23:20 sbin
drwxr-xr-x  2 root root 4096 Oct 14 00:00 srv
dr-xr-xr-x  13 root root    0 Dec 30 16:41 sys
drwxrwxrwt  1 root root 4096 Jan  5 23:20 tmp
drwxr-xr-x  1 root root 4096 Oct 14 00:00 usr
drwxr-xr-x  1 root root 4096 Oct 14 00:00 var
root@hello-pod:/# curl http://localhost:8000
Hello Kubernetes!
root@hello-pod:/#
```

Kubernetes Demo

Kubernetes version

- Service 생성

The screenshot shows the Kubernetes UI for creating a new resource. The top navigation bar includes a 'kubernetes' logo, a search bar, and a '+ CREATE' button highlighted with a red box. The main header says 'Resource creation'. On the left, a sidebar lists 'Workloads' (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), 'Discovery and Load Balancing' (Ingresses, Services), and 'Config and Storage' (Config Maps, Persistent Volume Claims, Secrets). The 'Services' item in the sidebar is also highlighted with a red box. The central area has three creation options: 'CREATE FROM TEXT INPUT' (selected), 'CREATE FROM FILE', and 'CREATE AN APP'. Below these is a text input field containing a YAML configuration for a Service:

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: hello-svc
5 spec:
6   selector:
7     app: hello
8   ports:
9     - port: 8200
10    targetPort: 8000
11   type: LoadBalancer
12   externalIPs:
13     - 172.30.56.67
```

At the bottom are 'UPLOAD' and 'CANCEL' buttons.

Kubernetes Demo

Kubernetes version

- Service 생성

The screenshot shows the Kubernetes Dashboard interface. On the left, a sidebar lists various workload types: Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, and two sections under 'Discovery and Load Balancing': Ingresses and Services. The 'Services' section is highlighted with a red box and a red arrow pointing to it from the main content area. The main content area is titled 'Services' and contains a table with the following data:

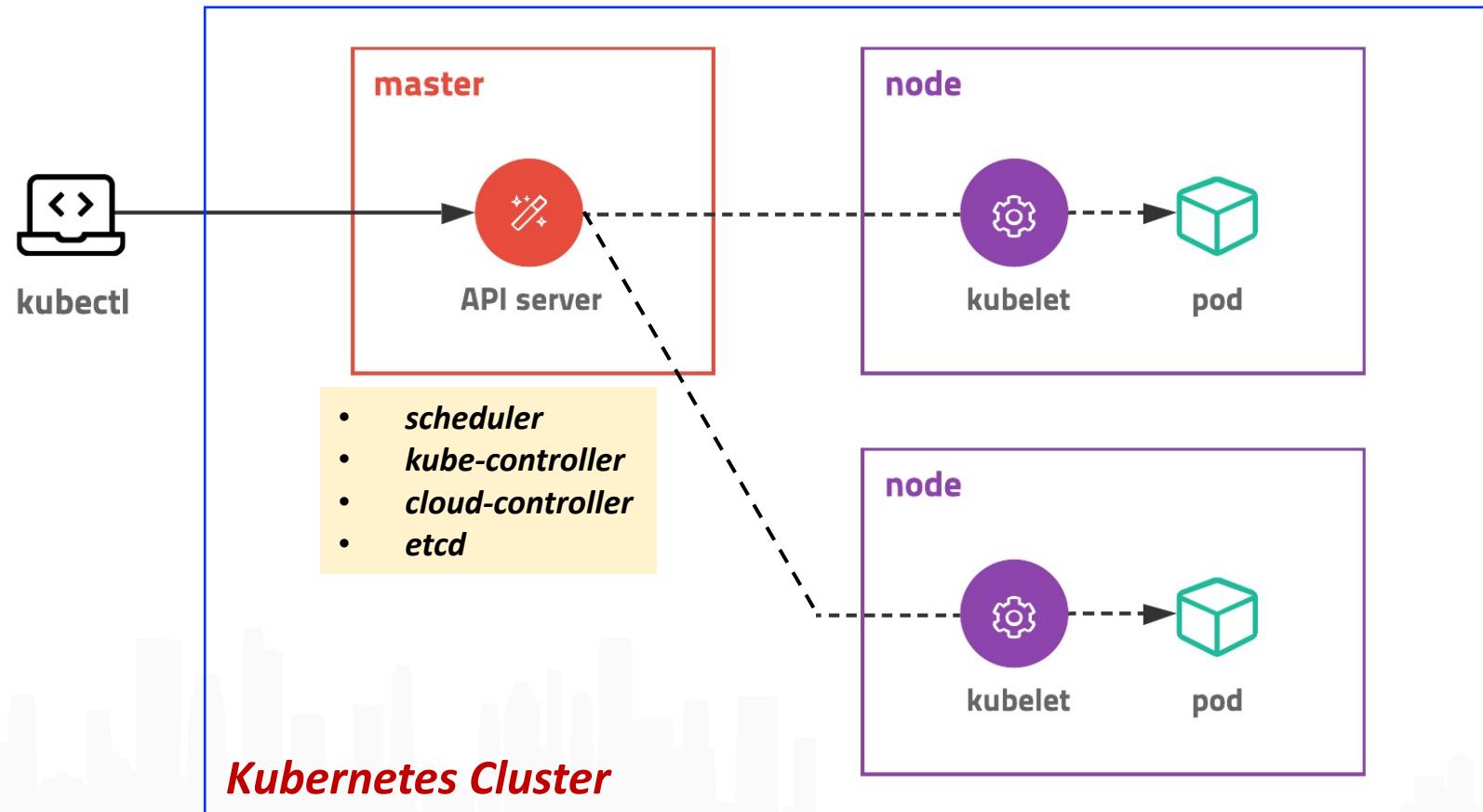
Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
hello-svc	-	10.97.208.182	hello-svc:8200 TCP hello-svc:31670 TCP	localhost:8200 172.30.56.67:8200	-
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	8 days

A red box highlights the 'Internal endpoints' column for the 'hello-svc' row. A red arrow points from this row to a browser window at the bottom right. The browser window title is 'hello-svc - Kubernetes Dashboard'. The address bar shows 'localhost:8200'. The page content displays 'Hello Kubernetes!'

■ Kubernetes – Cluster, Node

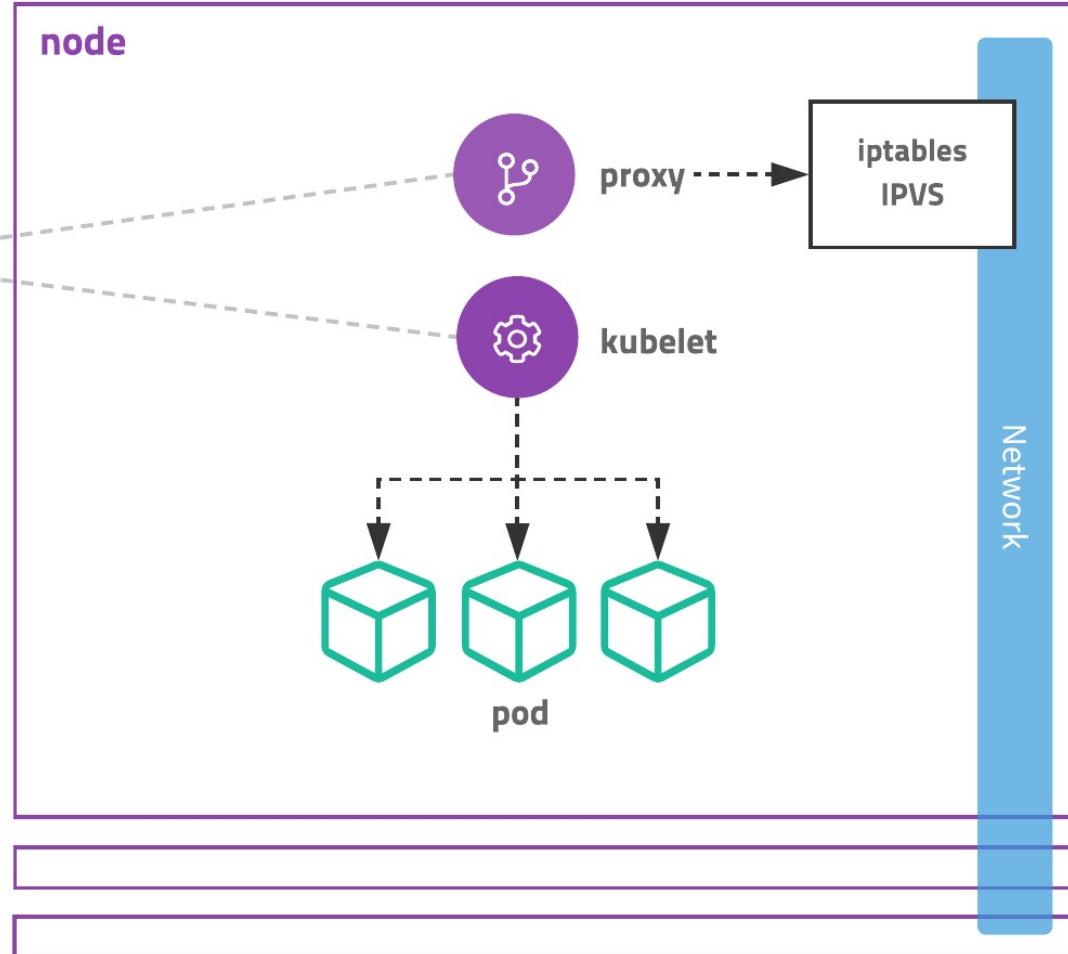
■ Kubernetes 오브젝트 중 가장 큰 개념은 Node

- Kubernetes 클러스터의 관리 대상으로 등록된 **Docker host**로, 컨테이너가 배치되는 대상
- 클러스터 전체를 관리하는 Master Node가 하나 이상 존재해야 함
 - SPOF를 피하기 위해 3개 추천



■ Kubernetes – Cluster, Node

■ Kubernetes 오브젝트 중 가장 큰 개념은 Node



- **kubelet**
 - 노드에 할당된 Pod의 생명주기 관리
 - Pod 생성, Pod 안의 컨테이너에 이상이 없는지 확인
 - API 서버의 요청을 받아 컨테이너의 로그 전달
- **kube-proxy**
 - Pod로 연결되는 네트워크 관리
 - TCP, UDP, SCTP 스트림 포워딩

Kubernetes – Cluster, Node

- 클러스터 생성 시 만들어진 가상 머신이 노드로 등록되어 있음

- \$ kubectl get nodes

```
kubectl get nodes
NAME           STATUS  ROLES   AGE    VERSION
docker-desktop  Ready   master  31h    v1.14.6
```

The screenshot shows the Kubernetes UI interface. On the left, there's a sidebar with navigation links: Cluster, Namespaces, Nodes (which is selected and highlighted in grey), Persistent Volumes, Roles, Storage Classes, Namespace (with 'default' selected), and Overview. The main content area has a header with a search bar and a 'CREATE' button. Below the header, it says 'Cluster > Nodes'. The table below lists the node details:

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
docker-desktop	beta.kubernetes.io/arch: ... beta.kubernetes.io/os: lin... kubernetes.io/arch: amd64 kubernetes.io/hostname: ... kubernetes.io/os: linux	True	0.76 (19.00%)	0.01 (0.25%)	160 Mi (8.00%)	360 Mi (18.01%)	a day

Kubernetes – Namespace

네임스페이스

- Kubernetes 클러스터 안에 가상 클러스터 생성

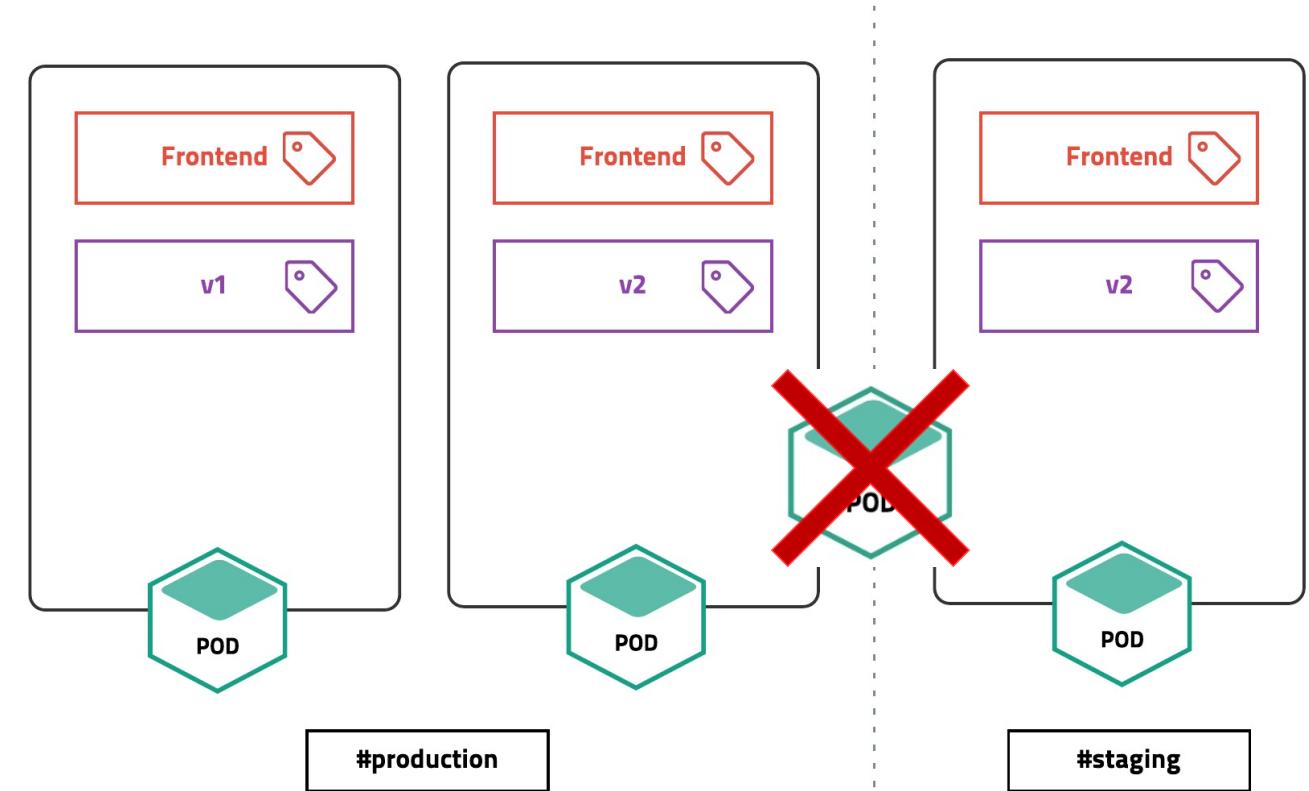
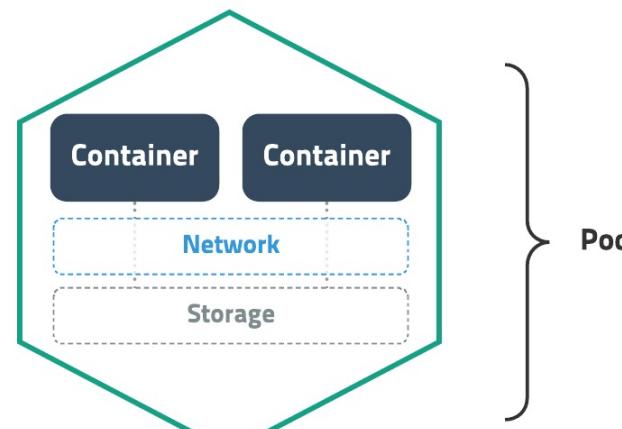
```
▶ kubectl get namespaces
NAME        STATUS  AGE
default    Active  31h
docker     Active  31h
kube-node-lease  Active  31h
kube-public   Active  31h
kube-system   Active  31h
```

The screenshot shows the Kubernetes UI interface. On the left, there is a sidebar with navigation links: Cluster, Namespaces (which is selected and highlighted in blue), Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (with 'default' selected), and Overview. The main content area has a title 'Namespaces' and a table with the following data:

Name	Labels	Status	Age
docker	-	Active	a day
default	-	Active	a day
kube-node-lease	-	Active	a day
kube-public	-	Active	a day
kube-system	-	Active	a day

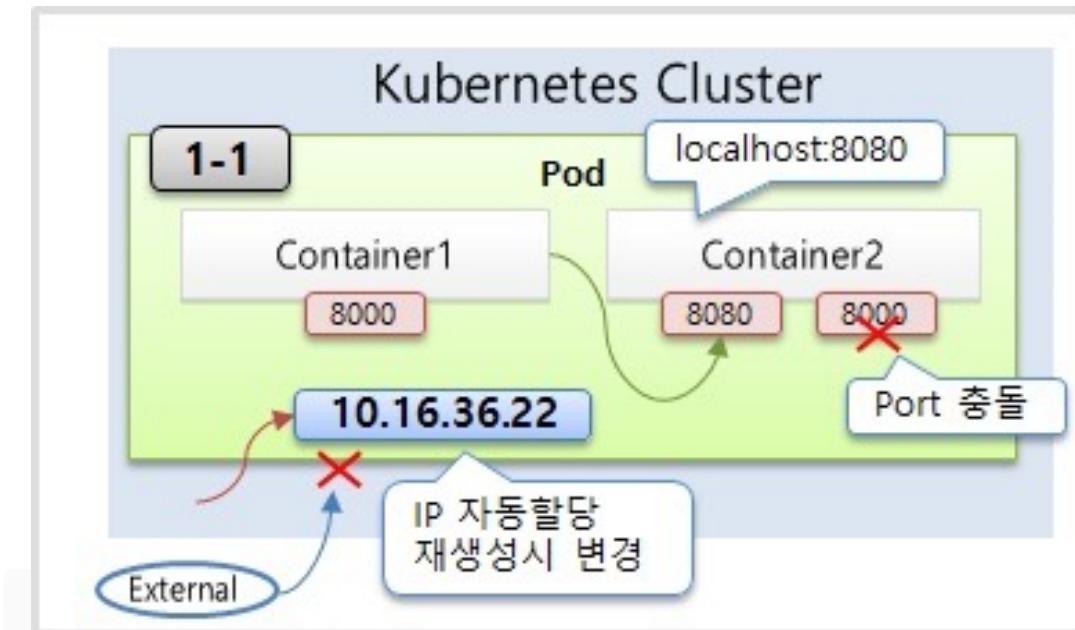
■ Kubernetes – Pod

- 컨테이너가 모인 집합체 단위, 하나 이상의 컨테이너로 구성
- Docker + Kubernetes에서는 하나의 컨테이너라도 반드시 Pod에 포함하여 배포

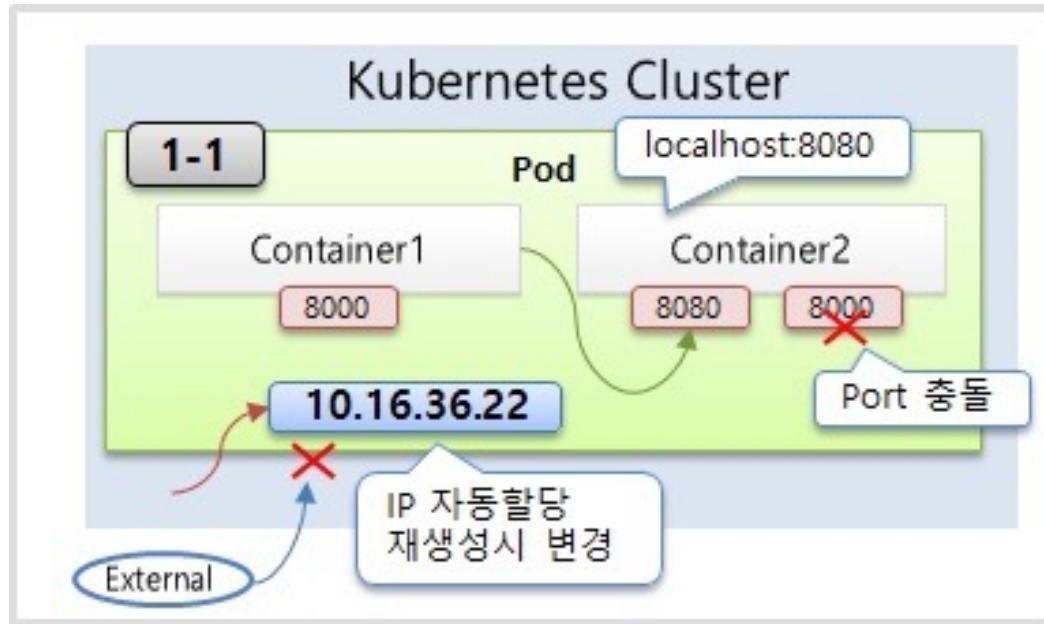


■ Kubernetes – Pod

- 컨테이너에서는 서로 연결할 수 있는 **중복되지 않는 Post 제공**
- 고유의 IP 할당
 - Cluster 내부에서 접속(외부에서는 접속 안됨)
- Pod에 문제가 생길 시 Pod 삭제 후 자동으로 다시 생성
 - IP 변경됨



■ Kubernetes – Pod

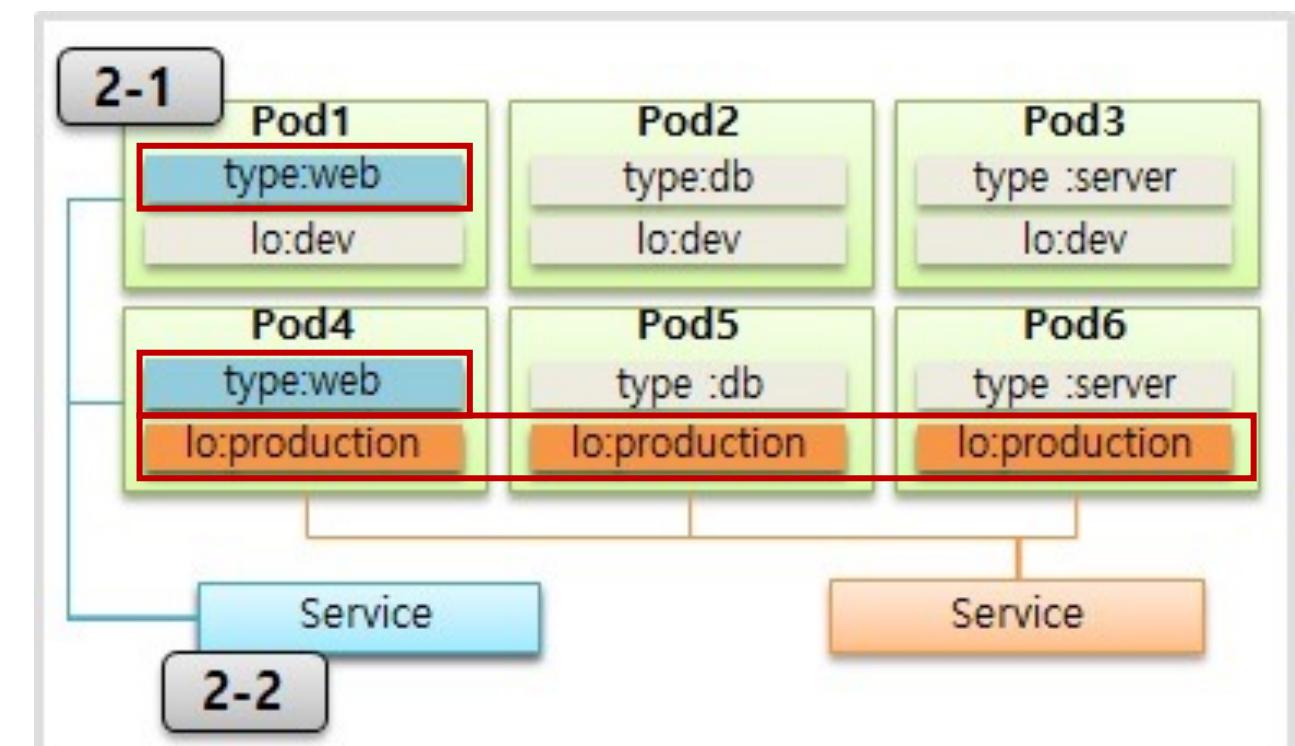


```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: pod-1
5 spec:
6   containers:
7     - name: container1
8       image: kubetm/p8000
9       ports:
10      - containerPort: 8000
11    - name: container2
12      image: kubetm/p8080
13      ports:
14        - containerPort: 8080
```

■ Kubernetes – Pod

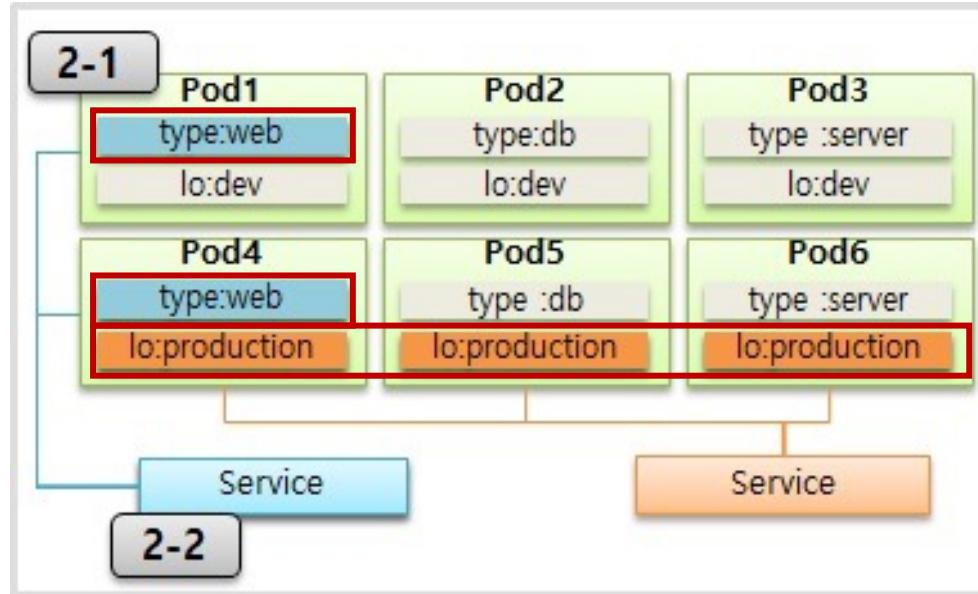
■ Label 지정

- 모든 오브젝트에 지정 가능 (Pod에서 가장 많이 사용)
- 오브젝트 분류하고, 분류 된 Label에 따라 작업 지정 가능
- **Key : Value**로 구성
- 사용 목적에 따라 **분류 → 검색** 가능



■ Kubernetes – Pod

■ Label 지정



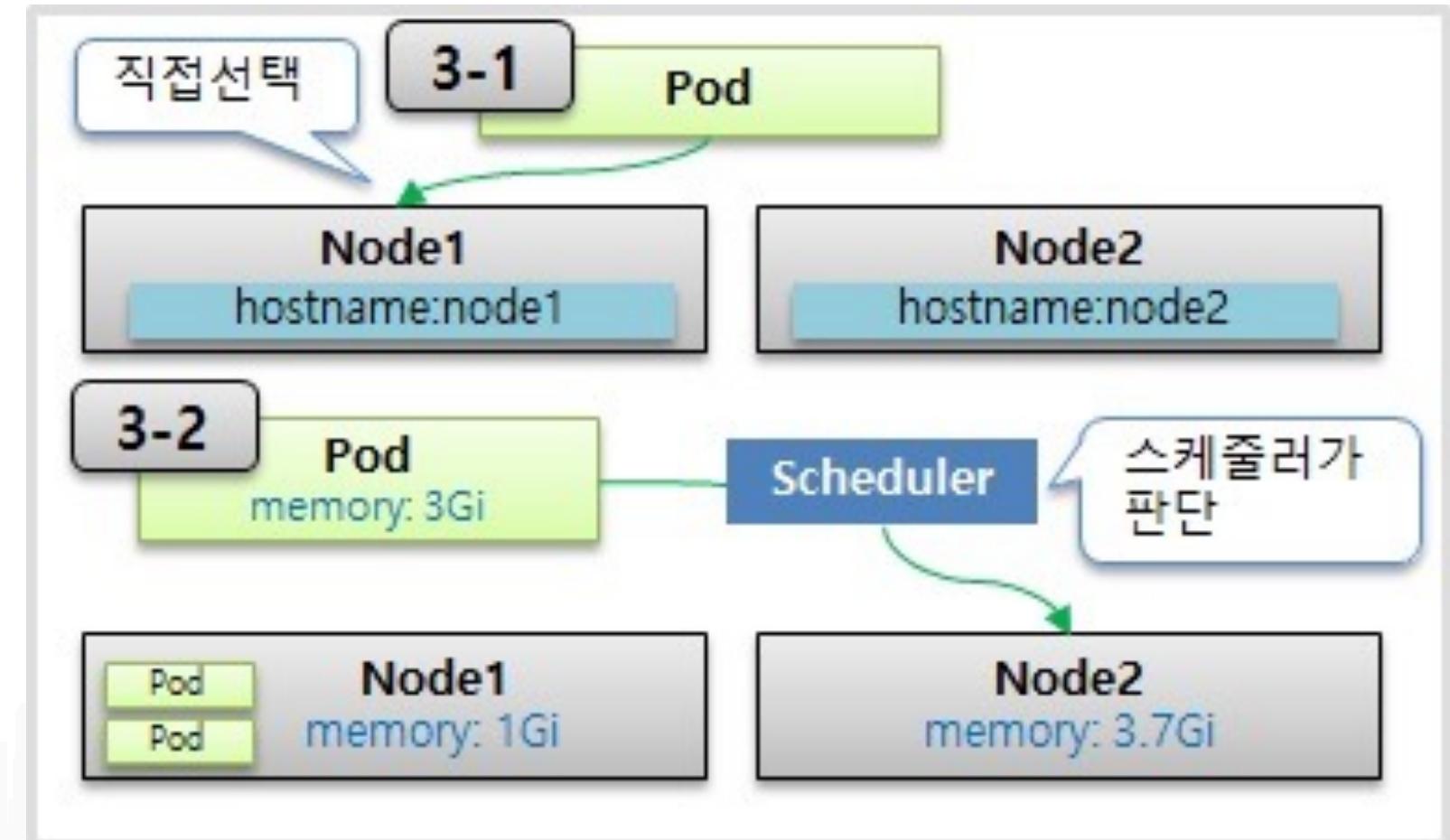
```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: pod-2
5   labels:
6     type: web
7     lo: dev
8 spec:
9   containers:
10  - name: container
11    image: kubetm/init
```

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: svc-1
5 spec:
6   selector:
7     type: web
8   ports:
9     - port: 8080
```

■ Kubernetes – Pod

■ NodeSchedule

- Pod는 하나의 Node에 지정되어 실행 됨
- 자동/수동 가능



■ Kubernetes – Pod

■ NodeSchedule

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: pod-3
5 spec:
6   nodeSelector:
7     kubernetes.io/hostname: k8s-node1
8   containers:
9     - name: container
10    image: kubetm/init
```

Node label 을 selector로 지정

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: pod-4
5 spec:
6   containers:
7     - name: container
8       image: kubetm/init
9       resources:
10      requests:
11        memory: 2Gi
12      limits:
13        memory: 3Gi
```

스케줄러가 판단

- 전체 사용 가능량을 판단하여 지정
- Pod에 부하가 걸릴 때 리소스를 무한정

사용하지 않기 위하여

■ Kubernetes – Pod

- 실습) nginx-proxy, echo 애플리케이션을 Pod로 배포

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: simple-echo
5  spec:
6    containers:
7      - name: nginx
8        image: gihyodocker/nginx:latest
9        env:
10       - name: BACKEND_HOST
11         value: localhost:8080
12       ports:
13         - containerPort: 80
14      - name: echo
15        image: gihyodocker/echo:latest
16        ports:
17          - containerPort: 8080
```

\$ kubectl apply -f simple-pod.yaml

\$ kubectl get pod

\$ kubectl exec -it simple-echo --container nginx -- bash

\$ kubectl logs -f simple-echo --container echo

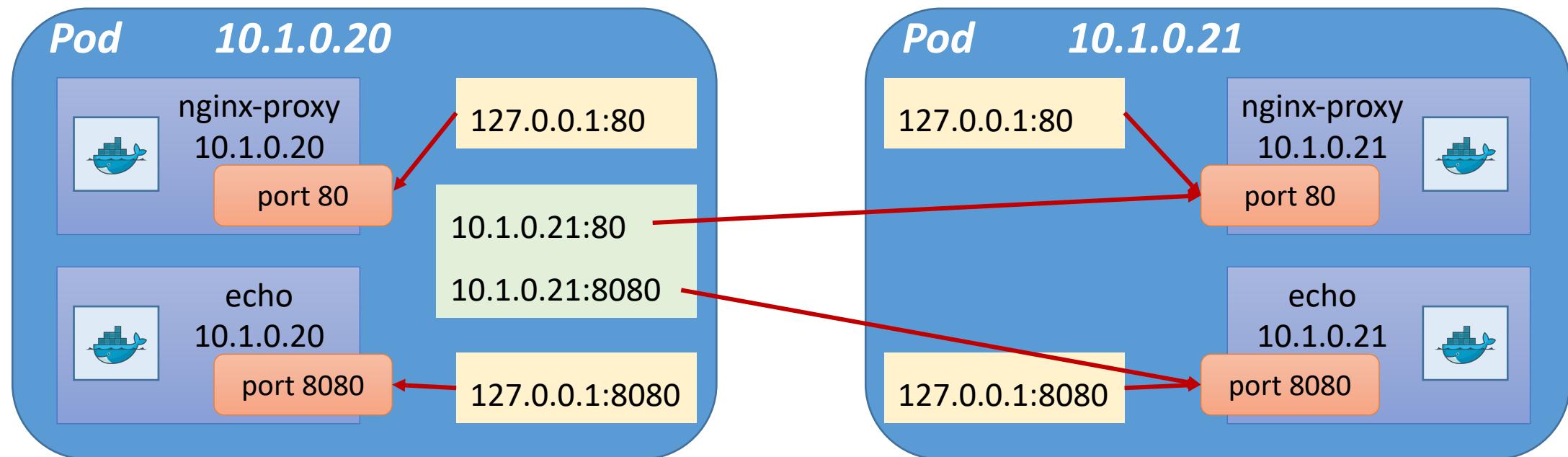
\$ kubectl delete pod simple-echo

(or \$kubectl delete -f simple-echo.yaml)

simple-pod.yaml

■ Kubernetes – Pod

- Pod에 할당된 가상IP 주소는 해당Pod에 속하는 모든 컨테이너가 공유



■ Kubernetes – ReplicaSet

- Pod는 하나의 Pod만을 생성
- 같은 Pod를 여러 개 생성하기 위해 ReplicaSet 사용

```
1  apiVersion: apps/v1
2  kind: ReplicaSet
3  metadata:
4    name: echo
5    labels:
6      app: echo
7  spec:
8    replicas: 3
9    selector:
10   matchLabels:
11     app: echo
12 template: # template 아래는 파드 리소스 정의와 같음
13   metadata:
14     labels:
15       app: echo
```

simple-replicaset.yaml

```
16  spec:
17    containers:
18      - name: nginx
19        image: gihyodocker/nginx:latest
20        env:
21          - name: BACKEND_HOST
22            value: localhost:8080
23        ports:
24          - containerPort: 80
25      - name: echo
26        image: gihyodocker/echo:latest
27        ports:
28          - containerPort: 8080
```

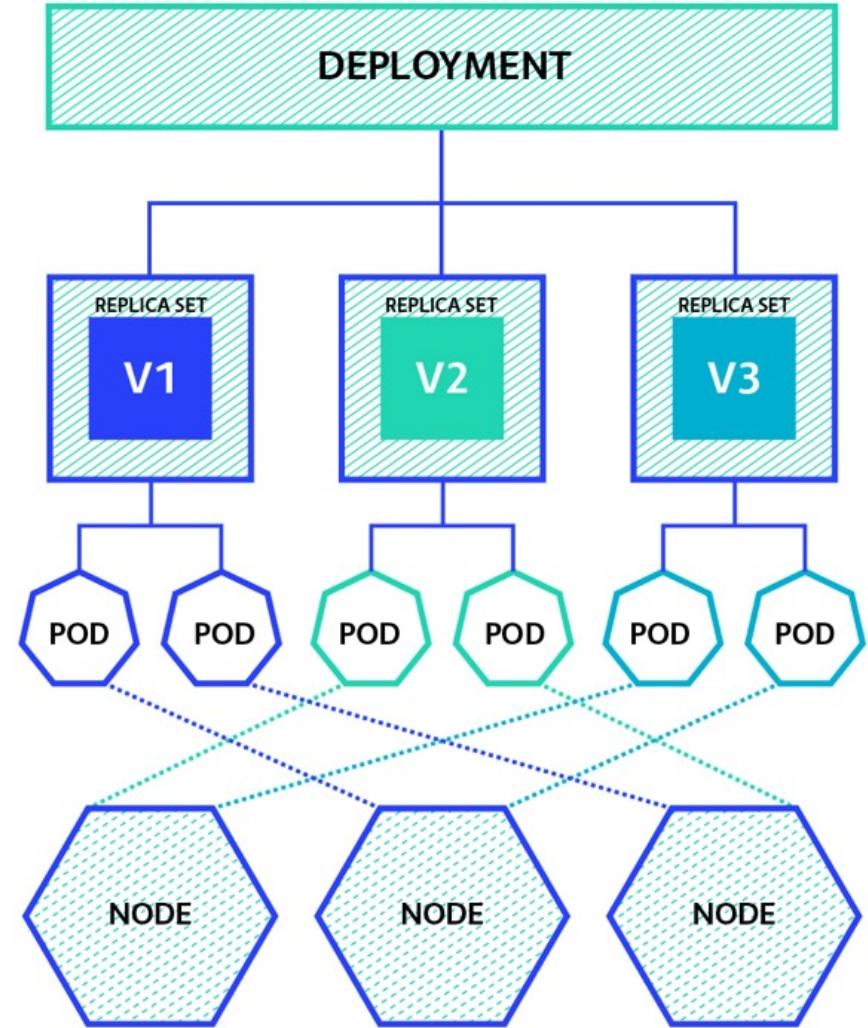
\$ kubectl apply -f simple-replicaset.yaml

\$ kubectl get pod

\$ kubectl delete -f simple-replicaset.yaml

■ Kubernetes – Deployment

- 애플리케이션 배포(deploy)의 기본 단위
- Deployment는 ReplicatSet를 관리하기 위한 Object
- Kubernetes가 애플리케이션의 인스턴스를 어떻게 생성하고 업데이트해야 하는지를 지시



Kubernetes – Deployment

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: echo
5   labels:
6     app: echo
7 spec:
8   replicas: 4
9   selector:
10    matchLabels:
11      app: echo
12 template: # template 아래는 파드 리소스 정의와 같음
13   metadata:
14     labels:
15       app: echo
```

simple-deployment.yaml

```
16   spec:
17     containers:
18       - name: nginx
19         image: gihyodocker/nginx:latest
20         env:
21           - name: BACKEND_HOST
22             value: localhost:8080
23         ports:
24           - containerPort: 80
25       - name: echo
26         image: gihyodocker/echo:latest
27         env:
28           - name: HOGE
29             value: fuga
30         ports:
31           - containerPort: 8080
```

\$ kubectl apply -f simple-deployment.yaml

\$ kubectl get pod,replicaset,deployment –selector app=echo

\$ kubectl rollout history deployment echo

■ Kubernetes – Deployment

■ ReplicaSet lifecycle

- Deployment 수정하면 ReplicaSet가 새로 생성 → 기존 ReplicaSet와 교체

■ Pod 개수 수정 → ReplicaSet 생성 되지 않음

```
$ kubectl apply -f simple-deployment.yaml --record
```

```
$ kubectl get pod
```

```
$ kubectl rollout history deployment echo
```

■ 컨테이너 수정 → ReplicaSet 새로 생성

```
$ kubectl apply -f simple-deployment.yaml --record
```

```
$ kubectl get pod --selector app=echo → REVISION=2
```

```
$ kubectl rollout history deployment echo
```

```
$ kubectl rollout undo deployment echo --to-revision=1
```

■ 이미지 변경

```
$ kubectl set image deployment echo [container name]=edowon0623/hello:2.0
```

```
$ kubectl rollout history deployment echo
```

■ Kubernetes – Deployment

■ Rollback 실행

```
$ kubectl rollout history deployment echo --revision=1
```

■ undo를 실행하여 직전 Revision으로 Rollback

```
$ kubectl rollout undo deployment echo
```

```
$ kubectl rollout undo deployment echo --to-revision=1
```

■ Deployment 삭제

```
$ kubectl delete -f simple-deployment.yaml
```

```
$ kubectl get pods -o=custom-columns='NameSpec:.metadata.namespace, \
Name:.metadata.name, \
Containers:.spec.containers[*].name'
```

■ Kubernetes – Service

- Kubernetes Cluster 안에서 Pod의 집합(주로 ReplicaSet)에 대한 경로나 Service Discovery를 제공하는 Object
- Pod에 접속(연결)하기 위한 Network endpoint

```
1  apiVersion: apps/v1
2  kind: ReplicaSet
3  metadata:
4    name: echo-spring
5  labels:
6    app: echo
7    release: spring
8  spec:
9    replicas: 1
10   selector:
11     matchLabels:
12       app: echo
13       release: spring
14   template:
15     metadata:
16       labels:
17         app: echo
18         release: spring
```

```
19   spec:
20     containers:
21       - name: nginx
22         image: gihyodocker/nginx:latest
23         env:
24           - name: BACKEND_HOST
25             value: localhost:8080
26         ports:
27           - containerPort: 80
28       - name: echo
29         image: gihyodocker/echo:latest
30         ports:
31           - containerPort: 8080
```

■ Kubernetes – Service

- Kubernetes Cluster 안에서 Pod의 집합(주로 ReplicaSet)에 대한 경로나 Service Discovery를 제공하는 Object

```
33 ---  
34 apiVersion: apps/v1  
35 kind: ReplicaSet  
36 metadata:  
37   name: echo-summer  
38   labels:  
39     app: echo  
40     release: summer  
41 spec:  
42   replicas: 2  
43   selector:  
44     matchLabels:  
45       app: echo  
46       release: summer  
47   template:  
48     metadata:  
49       labels:  
50         app: echo  
51         release: summer
```

simple-replicaset-with-label.yaml

```
52   spec:  
53     containers:  
54       - name: nginx  
55         image: gihyodocker/nginx:latest  
56         env:  
57           - name: BACKEND_HOST  
58             value: localhost:8080  
59         ports:  
60           - containerPort: 80  
61       - name: echo  
62         image: gihyodocker/echo:latest  
63         ports:  
64           - containerPort: 8080
```

\$ kubectl apply -f simple-replicaset-with-label.yaml

\$ kubectl get pod -l app=echo -l release=spring

\$ kubectl get pod -l app=echo -l release=summer

■ Kubernetes – Service

- Kubernetes Cluster 안에서 Pod의 집합(주로 ReplicaSet)에 대한 경로나 Service Discovery를 제공하는 Object

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: echo
5  spec:
6    selector:
7      app: echo
8    ports:
9      - name: http
10     port: 80
```

simple-service.yaml

```
$ kubectl apply -f simple-service.yaml
```

```
$ kubectl get svc echo
```

```
$ kubectl run -iy --rm debug --image=gihyodocker/fundamental:0.1.0 --restart=Never -- bash -il
```

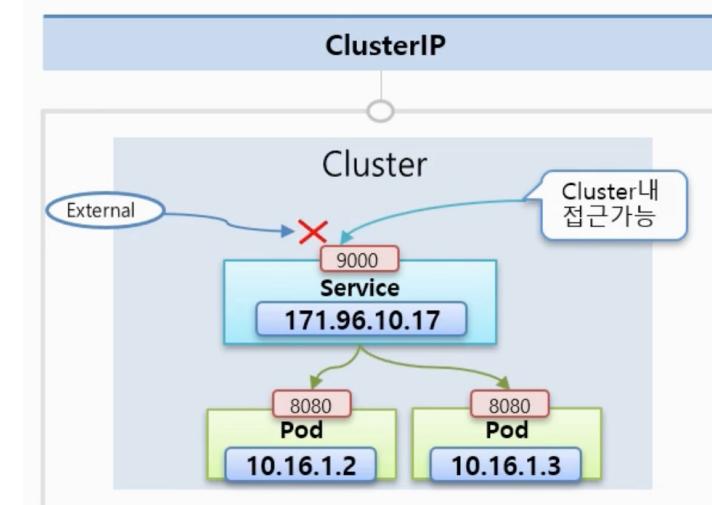
```
$ kubectl logs -f echo-summer-dtblk -c echo
```

■ Kubernetes – Service

- Pod는 재생성이 가능하며, 재생성 시 IP는 변경됨(동적IP, 불완전IP)
- Service는 한번 생성 후 삭제 시키기 전까지는 계속 유지
 - 서비스를 통해 Pod를 지정해 줌으로써, 지속적인 Pod를 사용할 수 있도록 지원

▪ ClusterIP Service

- 내부 네트워크 설정의 기본 값
- **Kubernetes Cluster** 내부 IP 주소에만 서비스 공개
(Only reachable from within cluster)
- Pod에서 다른 Pod로 접근 할 때 사용
- 외부로부터 접근 안됨
 - 서비스 상태 디버깅



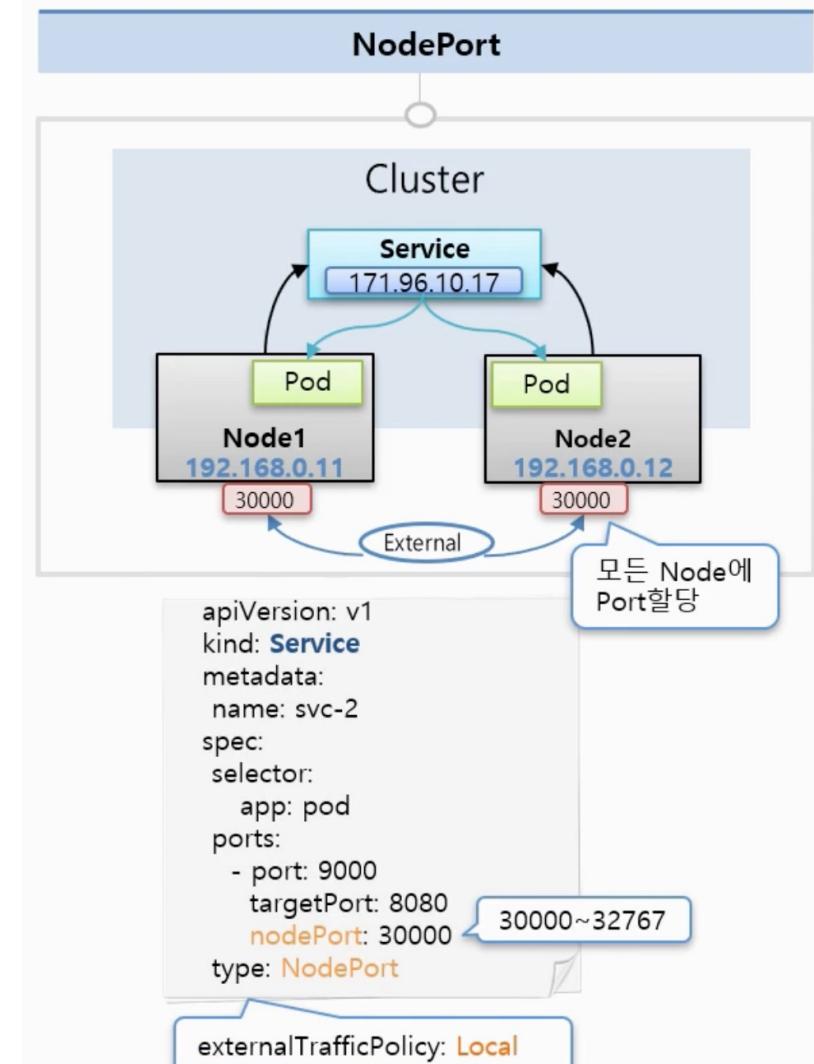
```
apiVersion: v1
kind: Service
metadata:
  name: svc-1
spec:
  selector:
    app: pod
  ports:
    - port: 9000
      targetPort: 8080
      type: ClusterIP
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  labels:
    app: pod
spec:
  containers:
    - name: container
      image: tmkube/app
      ports:
        - containerPort: 8080
```

■ Kubernetes – Service

■ NodePort Service

- 외부에서 접근할 수 있는 설정 (모든 Node에 대해 Port가 Open)
 - simple-service.yaml 수정
 - \$ kubectl get svc echo
 - \$ curl <http://127.0.0.1:31058>
- 내부망 연결
 - 데모나 임시 연결용



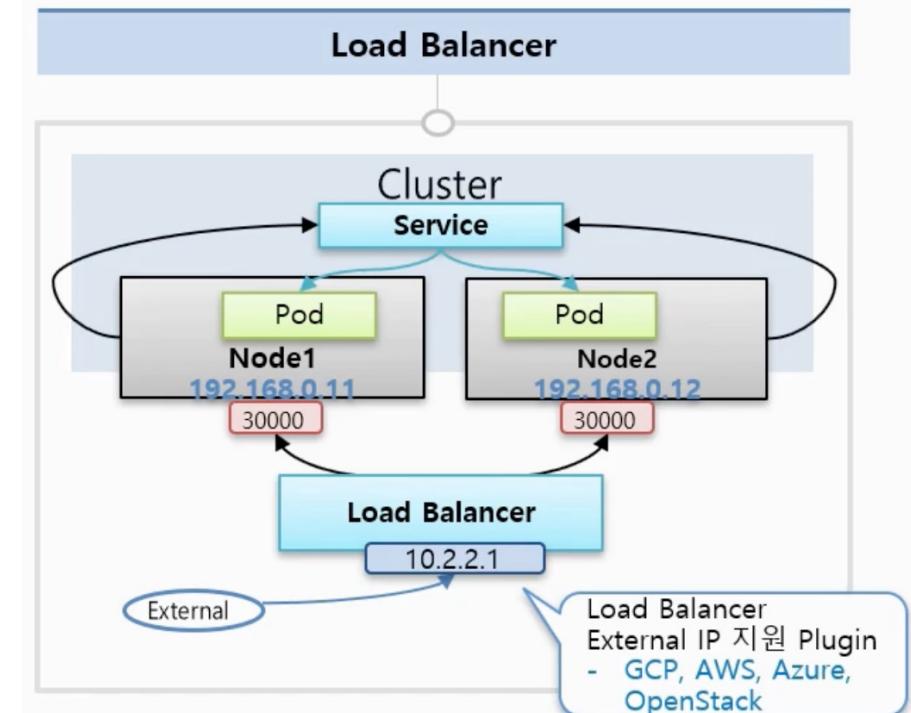
■ Kubernetes – Service

■ LoadBalancer Service

- Cluster *or* LB endpoint 제공
- Only available when infra provider gives you a LB (AWS ELB, etc)
- 외부 시스템 노출 시 사용

■ ExternalName Service

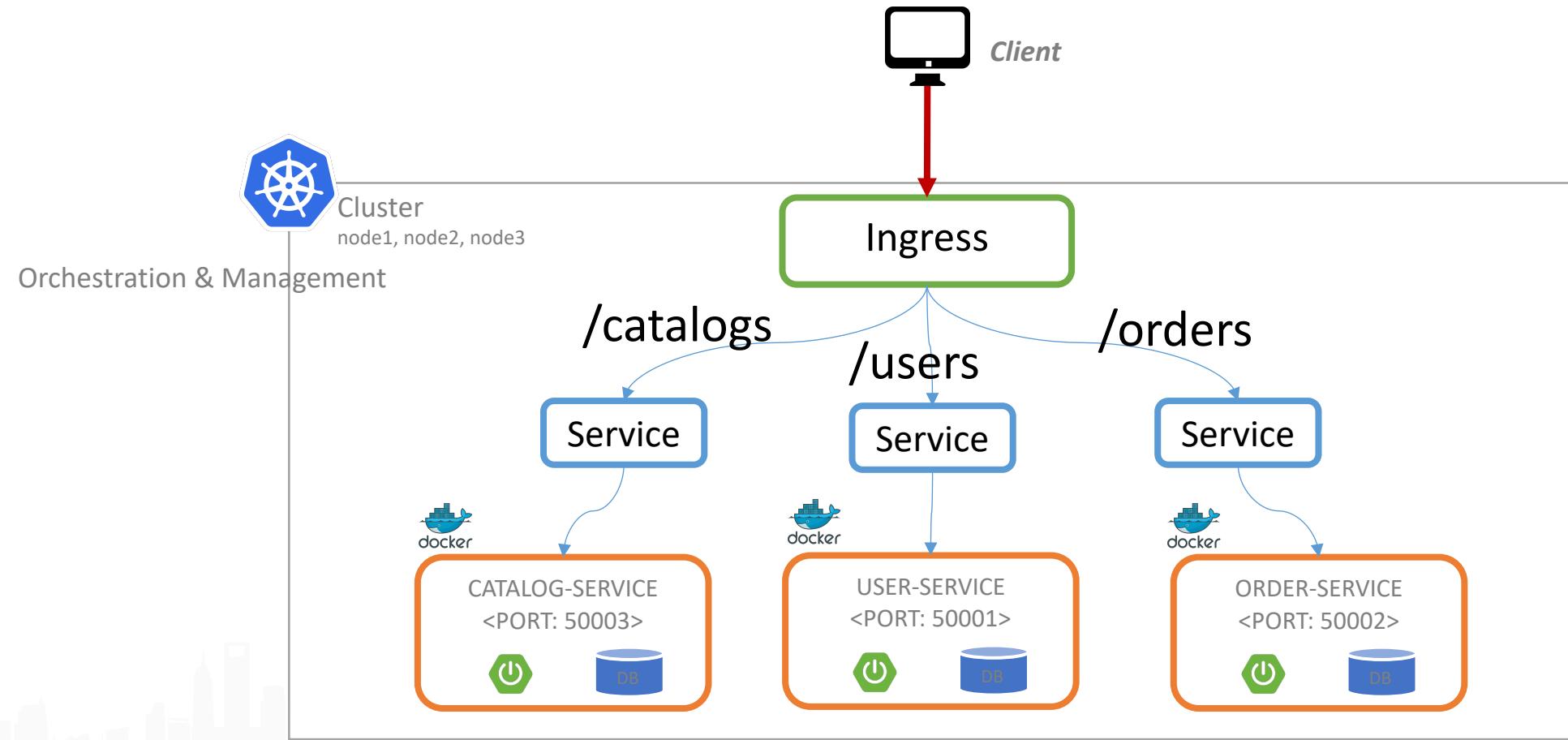
- CNAME DNS record to CoreDNS



```
apiVersion: v1
kind: Service
metadata:
  name: svc-3
spec:
  selector:
    app: pod
  ports:
    - port: 9000
      targetPort: 8080
  type: LoadBalancer
```

■ Kubernetes – Ingress

- **HTTP(S) 기반의 L7 로드밸런싱 기능을 제공하는 컴포넌트**
 - k8s는 기본적으로 L4 레이어로 TCP 단에서 Pod를 로드밸런싱



■ Kubernetes – Ingress

■ Ingress

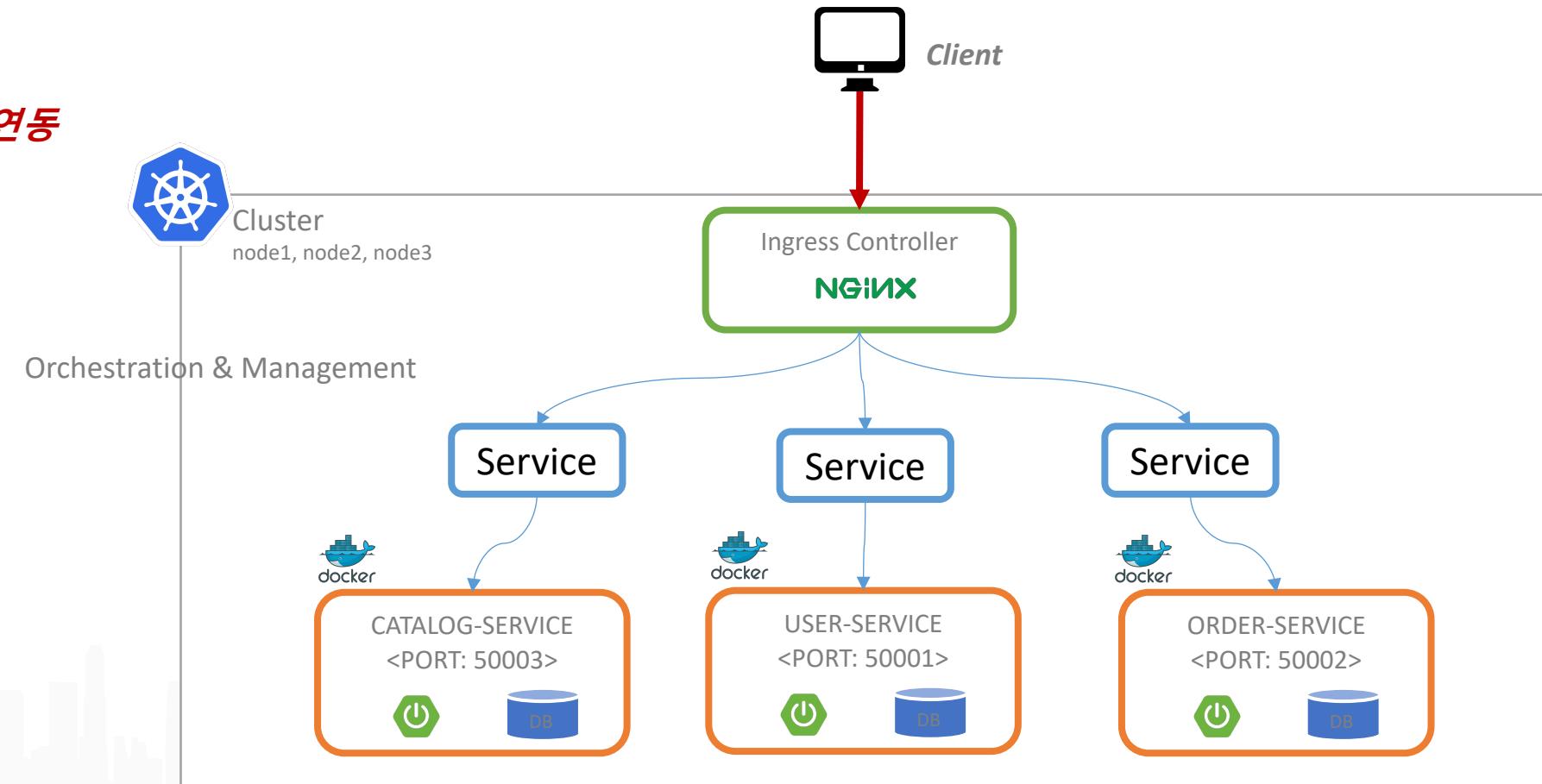
- HTTP 경로의 라우팅 규칙 정의

■ Ingress-Controller

- 규칙들을 동작

■ Ingress-nginx

- Ingress 와 Ingress Controller 연동



Kubernetes – Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: service-loadbalancing
spec:
  rules:
  - http:
    paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: svc-shopping
            port:
              number: 8080
      - path: /customer
        pathType: Exact
        backend:
          service:
            name: svc-customer
            port:
              number: 8080
      - path: /order
        pathType: Exact
        backend:
          service:
            name: svc-order
            port:
              number: 8080
```

Ingress-nginx 적용

```
$ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-
v0.47.0/deploy/static/provider/cloud/deploy.yaml
```

■ Kubernetes – Ingress

■ Ingress 확인

\$ kubectl get ingress --all-namespaces

ubuntu@ip-172-31-25-234:~/app/tta_cloud_native/k8s/msa\$ kubectl get ingress --all-namespaces						
NAMESPACE	NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
default	nginx-ingress-sample	<none>	*	100.67.243.230	80	30m

■ Ingress Pod 확인

\$ kubectl get pod --namespace ingress-nginx

ubuntu@ip-172-31-25-234:~/app/tta_cloud_native/k8s/msa\$ kubectl get pod --namespace ingress-nginx						
NAME	READY	STATUS	RESTARTS	AGE	IP	NO
nginx-ingress-controller-7fcb6cffc5-ms5sw	1/1	Running	0	44m	100.96.3.8	ip

■ Service 확인

\$ kubectl get service --namespace ingress-nginx

ubuntu@ip-172-31-25-234:~/app/tta_cloud_native/k8s/msa\$ kubectl get svc --namespace ingress-nginx						
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
ingress-nginx	NodePort	100.67.243.230	<none>	80:32152/TCP,443:32284/TCP	47m	

Kubernetes – Persistent Volume

Volume

- 컨테이너 간에 파일 공유를 위해 사용

임시 볼륨	로컬 볼륨	네트워크 볼륨	네트워크 볼륨 (클라우드 종속적)
emptyDir	hostpath local	iSCSI NFS cephFS glusterFS ...	gcePersistentDisk awsEBS azureFile

Kubernetes – Persistent Volume

■ 임시 볼륨 - emptyDir

- 임시 공유 디렉토리

```
- name: echo
  image: gihyodocker/echo:latest
  ports:
    - containerPort: 8080
  volumeMounts:
    - mountPath: /data
      name: my-empty-volume
  volumes:
    - name: my-empty-volume
      emptyDir: {}
```

■ 로컬 볼륨 - hostPath

- HOST와 파일 공유
- HOST 변경 되면 다시 지정
 - nodeSelector로 Host 지정
 - HOST의 디렉토리가 생성되어 있어야 함

```
spec:
  containers:
    - name: nginx-container
      image: nginx
      volumeMounts:
        - mountPath: /mydata-container
          name: my-hostpath-volume
  volumes:
    - name: my-hostpath-volume
      hostPath:
        path: /Users/downlee/Desktop/git/
              seocho_blockchain/09.src/day17/mydata
        type: Directory
```

■ *Kubernetes – Persistent Volume*

■ 로컬 볼륨 - local

- 로컬 볼륨이 존재하는 HOST를 찾아서 컨테이너를 할당
- Persistent Volume을 통해서 사용 가능

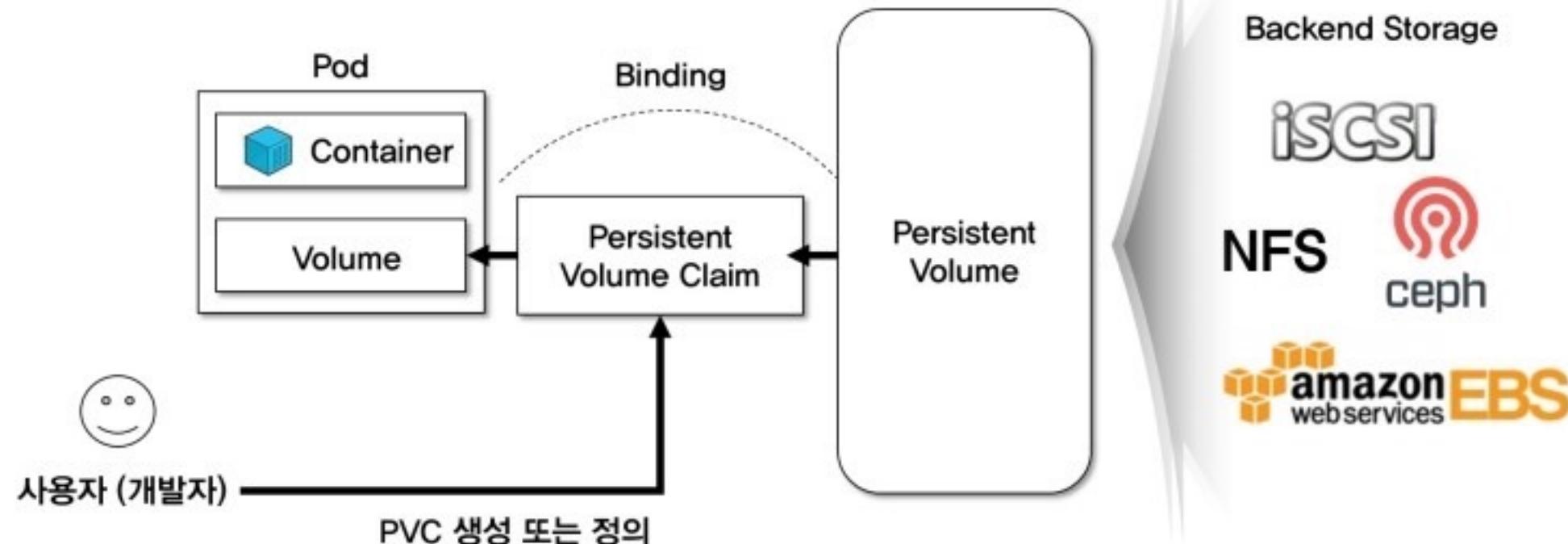
■ 네트워크 볼륨

- **Persistent Volume**
 - 실제 물리 디스크를 생성한 후에, 이 디스크를 PersistentVolume이라는 이름으로 쿠버네티스에 등록
 - 시스템 관리자가 생성한 물리 디스크를 쿠버네티스 클러스터에 표현한 것
- **Persistent Volume Claim**
 - POD 생성 시 Volume을 정의하기 위해, 물리적 디스크에 대한 특성을 정의하는 것이 아니라 관리자가 생성한 PV와 연결한 것
 - 여기서 Pod의 Volume과 이 PV를 연결을 위해 PVC를 등록

■ Kubernetes – Persistent Volume

■ 네트워크 볼륨

- **Persistent Volume, Persistent Volume Claim**



출처: https://blog.naver.com/alice_k106/221348788068

■ Kubernetes – Persistent Volume

- Persistent Volume, Persistent Volume Claim 라이프 사이클 (Reclaim Policy)
 - PV 상태 → **Available, Bound, Released, Fail**
 - **spec.persistentVolumeReclaimPolicy** 나 Storage Class의 reclaimPolicy에서 정의

```
5   spec:  
6     persistentVolumeReclaimPolicy: Recycle  
7     capacity:  
8       storage: 100Gi  
9     accessModes:  
10      - ReadWriteMany
```

- spec.persistentVolumeReclaimPolicy 정책
 - Retain → NFS, iSCSI 뿐만 아니라 Google PD, AWS EBS 등에 적합
 - Delete → PVC가 삭제되어 PV Released 상태가 되면, PV의 디스크 삭제 (재사용 위해 PV 다시 생성)
 - Recycle → 사용하던 볼륨 내부의 파일을 전부 삭제 (NFS와 HostPath를 사용하는 PV에 적용)

Kubernetes – Persistent Volume

Persistent Volume, Persistent Volume Claim 생성

Persistent Volume 예제

```
1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: dev-pv
5  spec:
6    capacity:
7      storage: 2Gi
8    volumeMode: Filesystem
9    accessModes:
10   - ReadWriteOnce
11   storageClassName: manual
12   persistentVolumeReclaimPolicy: Delete
13   hostPath:
14     path: /Users/downonlee/Desktop/git/secho_blockchain/09.src/day17/mydata
```

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: dev-pvc
5  spec:
6    accessModes:
7      - ReadWriteOnce
8    volumeMode: Filesystem
9    resources:
10   requests:
11     storage: 2Gi
12   storageClassName: manual
```

Persistent Volume Claim 예제

Kubernetes – Persistent Volume

- Persistent Volume, Persistent Volume Claim → Pod, Deployment 연결

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: pod-volume-test
5  spec:
6    containers:
7      - name: container
8        image: kubetm/init
9        volumeMounts:
10       - name: pvc-pv
11         mountPath: /my_mount
12    volumes:
13      - name : pvc-pv
14        persistentVolumeClaim:
15          claimName: dev-pvc
```

Pod 예제

```
12   template:
13     metadata:
14       labels:
15         app: test-deployment
16     spec:
17       containers:
18         - name: test-deployment
19           image: nginx
20           ports:
21             - containerPort: 8080
22             volumeMounts:
23               - mountPath: "/var/log/test.log"
24                 name: dev-volume
25             volumes:
26               - name: dev-volume
27               persistentVolumeClaim:
28                 claimName: dev-pvc
```

Deployment 예제

■ Kubernetes – Persistent Volume

■ 네트워크 볼륨 – Cloud

- gcePersistentDisk, **awsEBS**, azureFile
- \$ aws ec2 create-volume --availability-zone=[ap-northeast-2c] --size=10 --volume-type=gp2

apiVersion: v1

kind: Pod

metadata:

name: test-ebs

spec:

containers:

- **image:** k8s.gcr.io/test-webserver

- name:** test-container

- volumeMounts:**

- **mountPath:** /test-ebs

- name:** test-volume

volumes:

- **name:** test-volume

This AWS EBS volume must already exist.

awsElasticBlockStore:

- volumID:** "<volume id>"

- fsType:** ext4

키	값
KubernetesCluster	jonecluster2.k8s.local