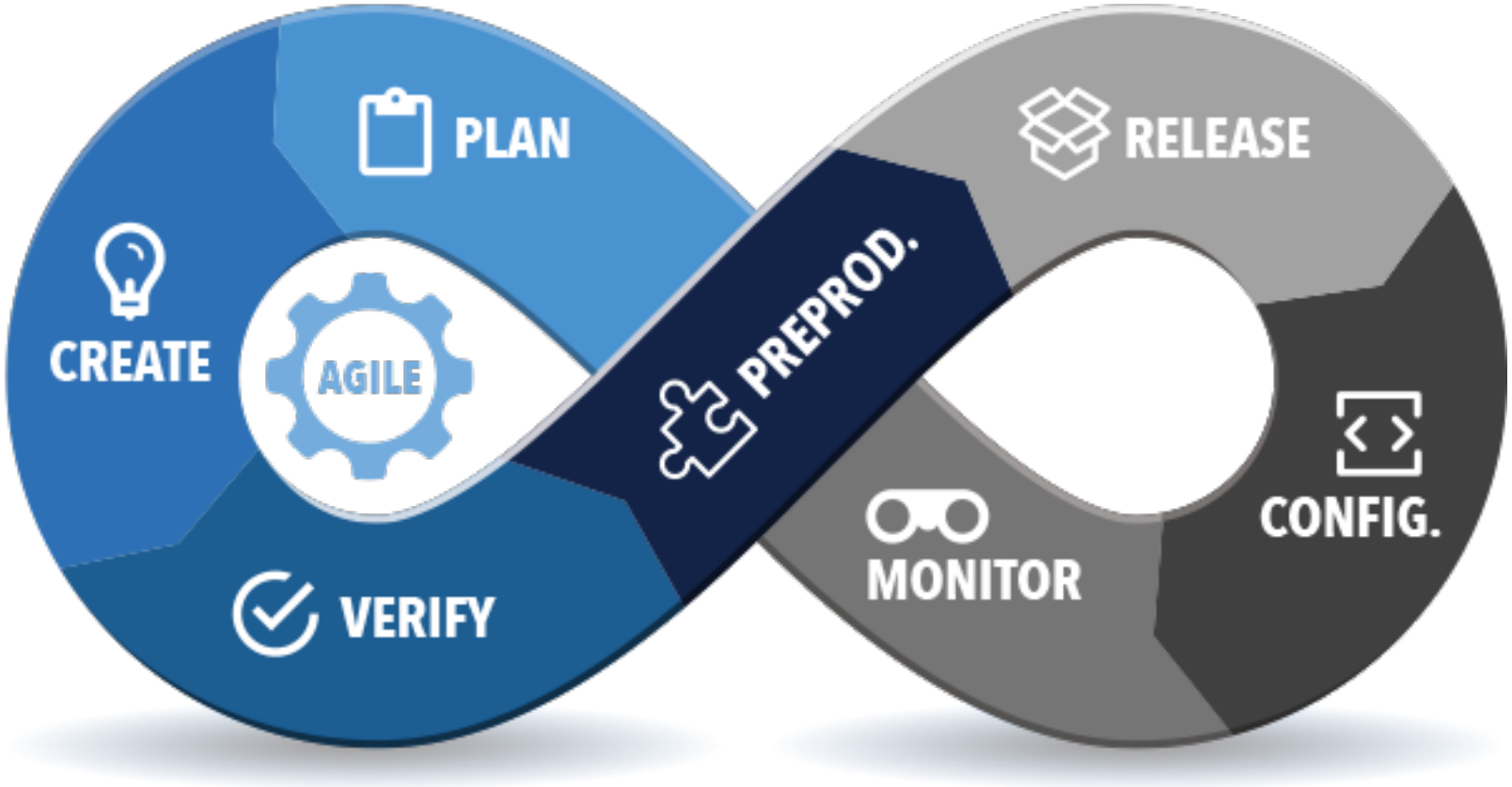
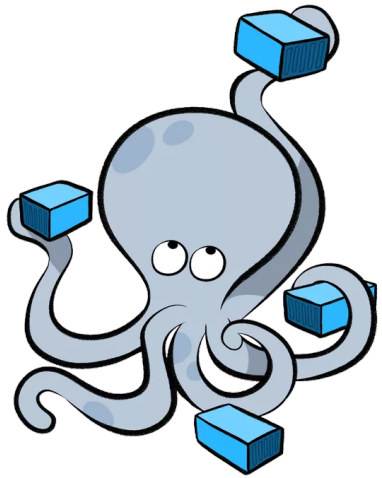


4. Kubernetes

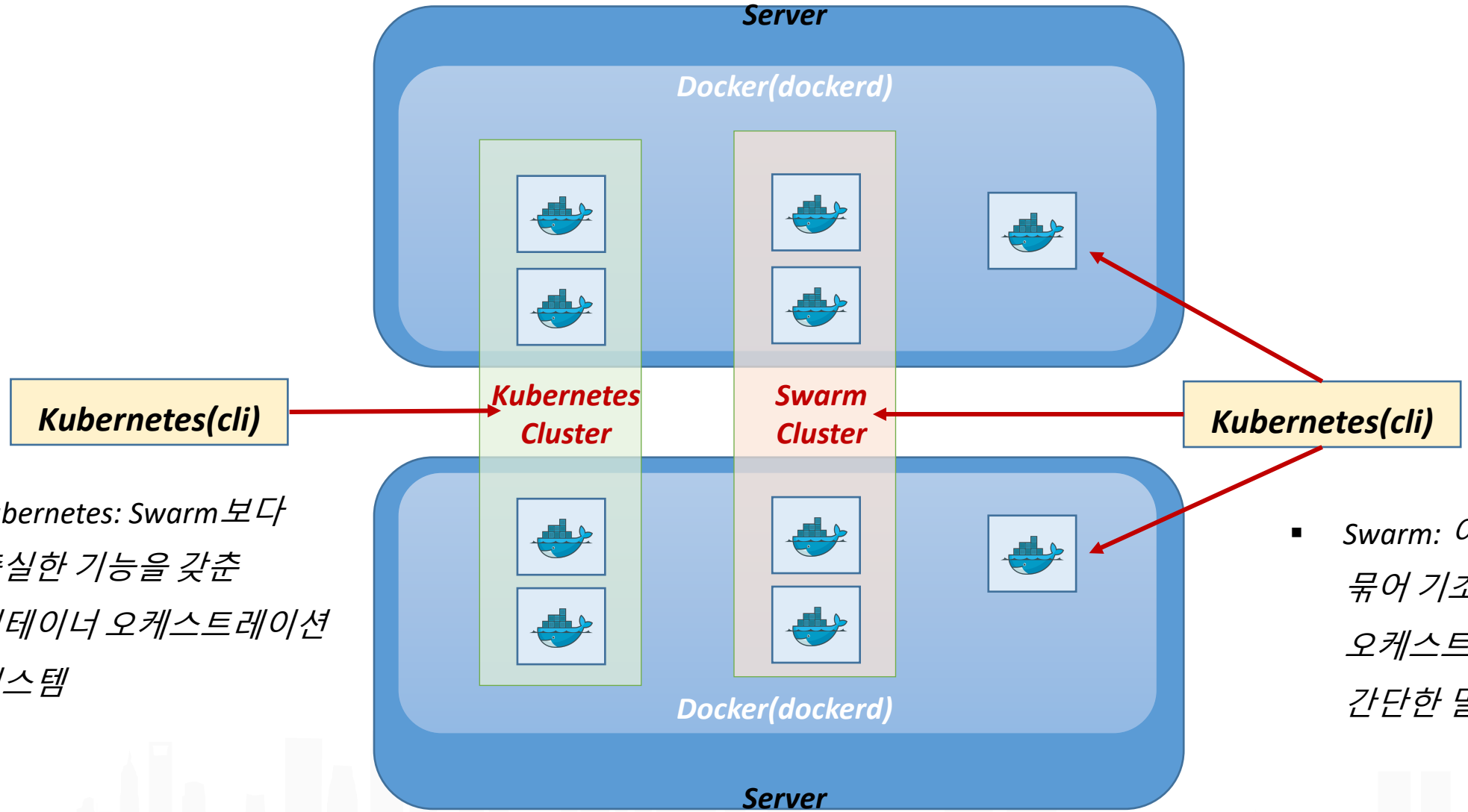
Kubernetes

- ***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 Docker에 정식으로 통합된 사실상 표준***
  - DockerCon EU 2017



kubernetes

Kubernetes



- Kubernetes: Swarm보다 충실한 기능을 갖춘 컨테이너 오케스트레이션 시스템

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

Kubernetes

kubernetes
검색어

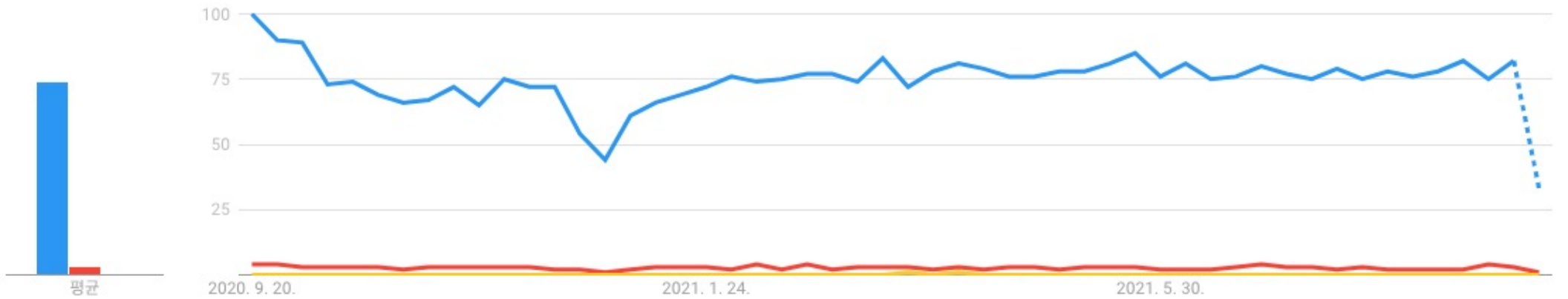
docker swarm
검색어

apache mesos
검색어

+ 비교 추가

전 세계 ▾ 지난 12개월 ▾ 모든 카테고리 ▾ 웹 검색 ▾

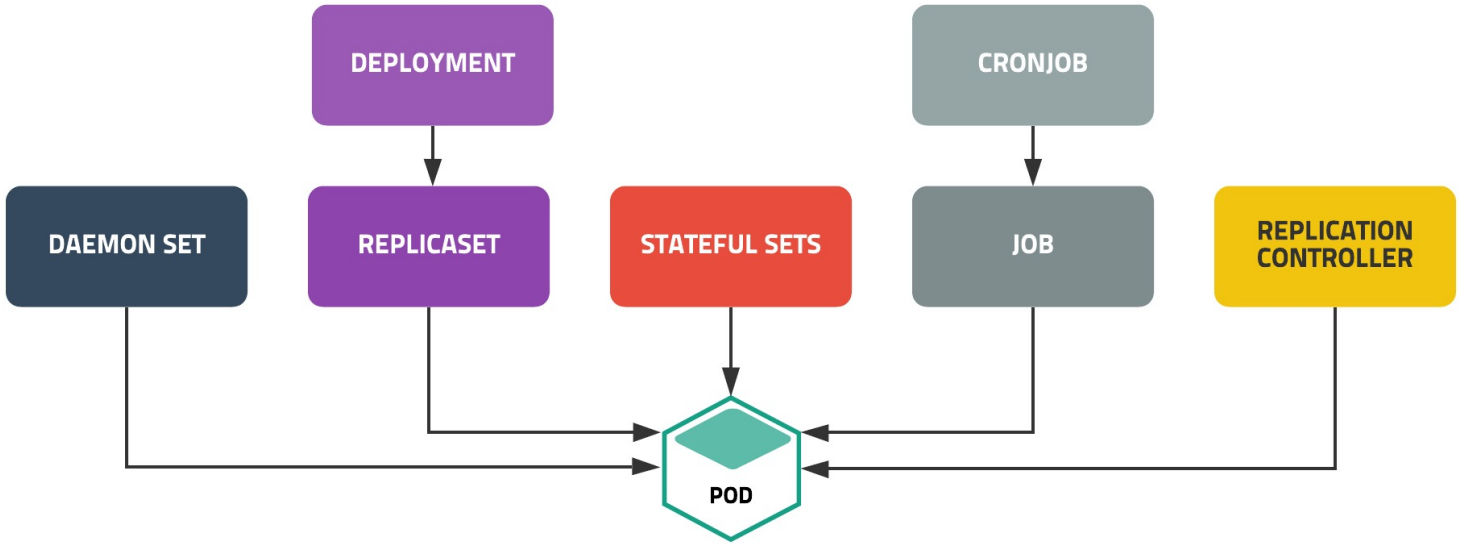
시간 흐름에 따른 관심도 변화



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 배포 방식

- ***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"}
```

- ***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)

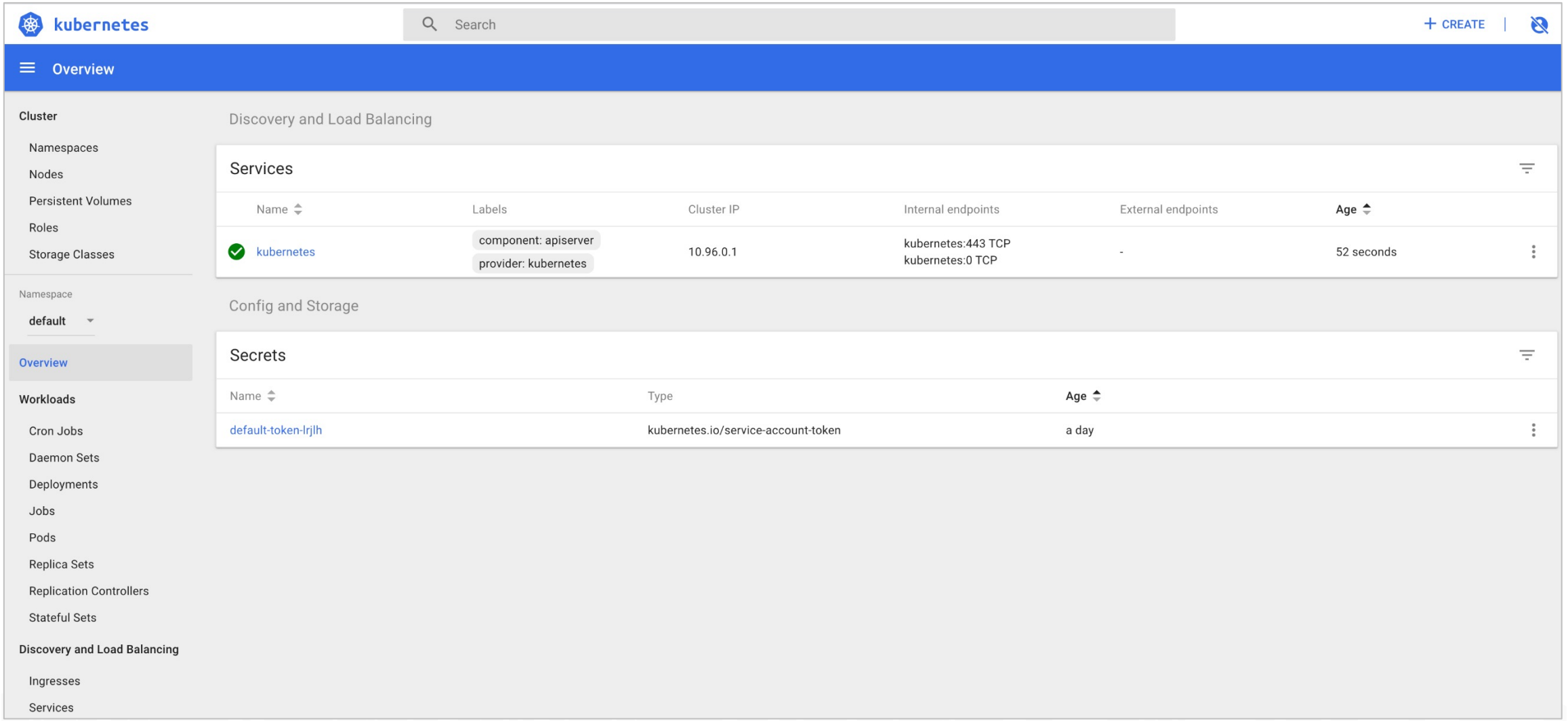
```
NAME                                    READY   STATUS    RESTARTS   AGE
kubernetes-dashboard-6fd7f9c494-2bkjp   1/1     Running   0          30h
```

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

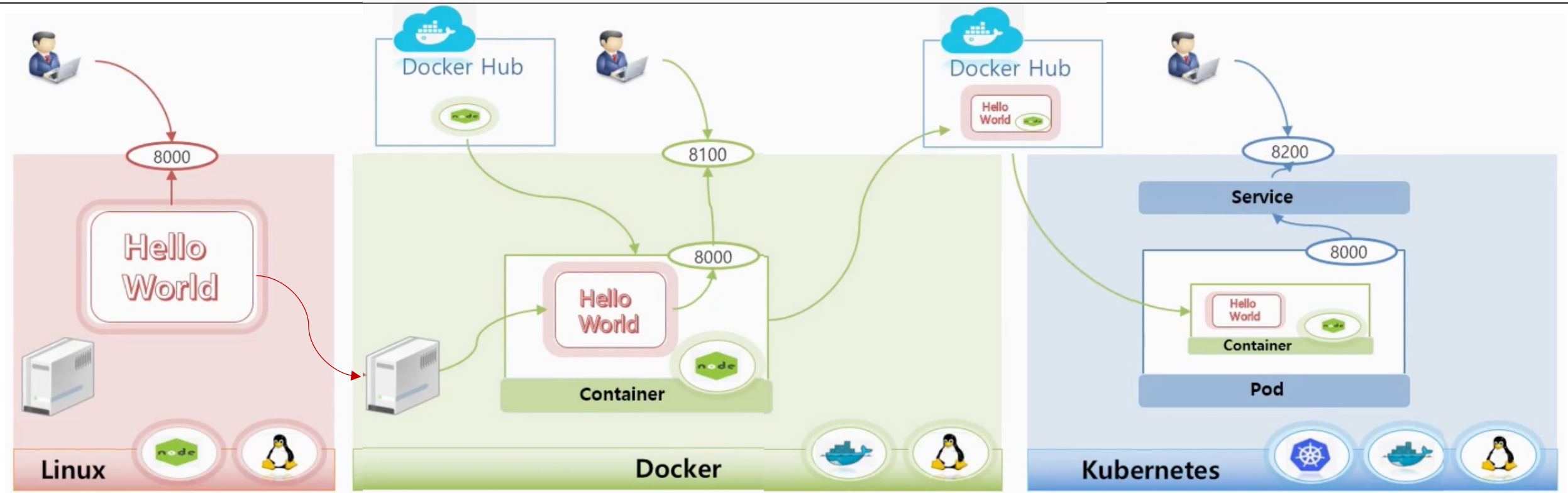
Dashboard 설치



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);
```

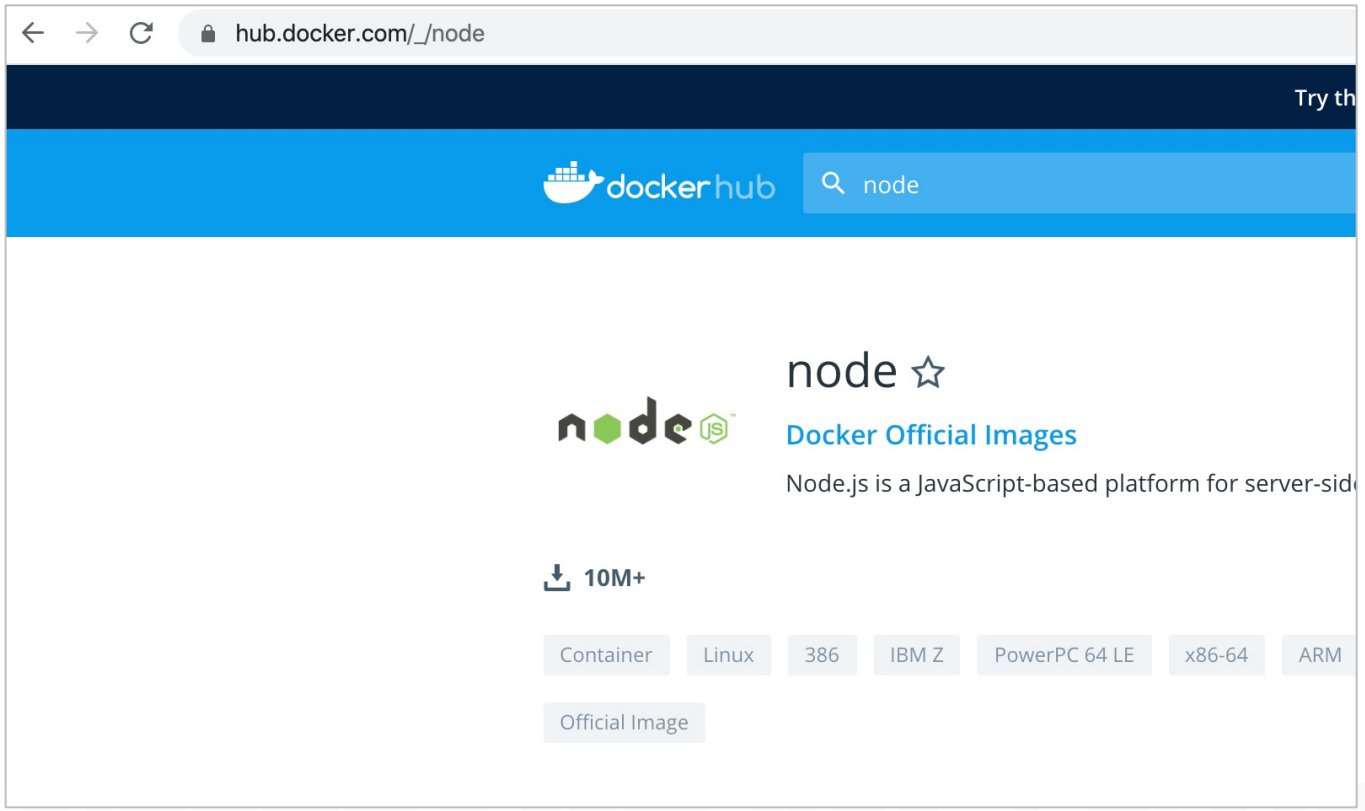
$ node hello.js

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
```

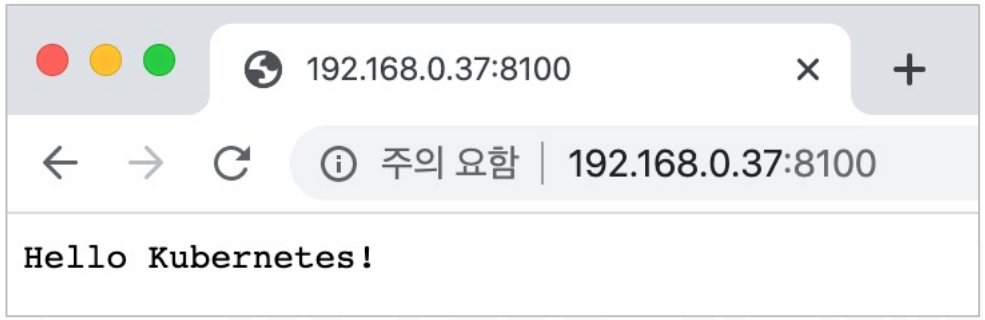
Kubernetes Demo

- ***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 ..
-rwxr-xr-x   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

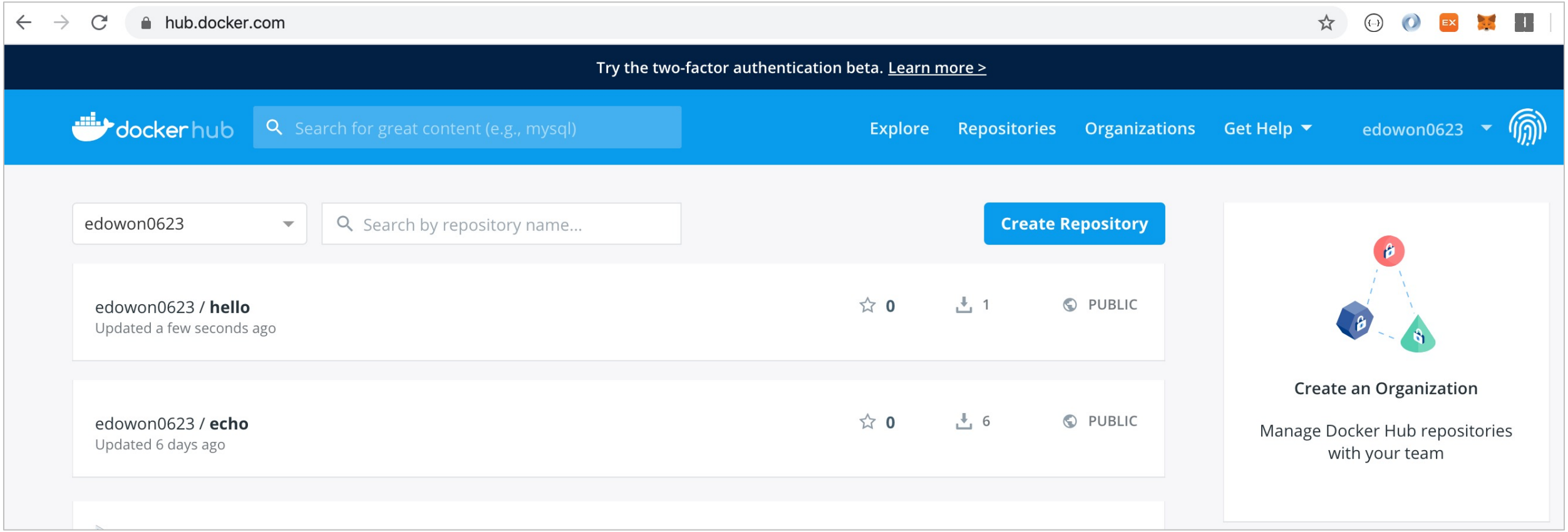
- ***Kubernetes version***

```
> 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
814c70fdae62: Mounted from library/node
latest: digest: sha256:c46636d9b118c67c2248b59bb6df93f16d12004be86f41db44cdbca3bc521396 size: 1574
```

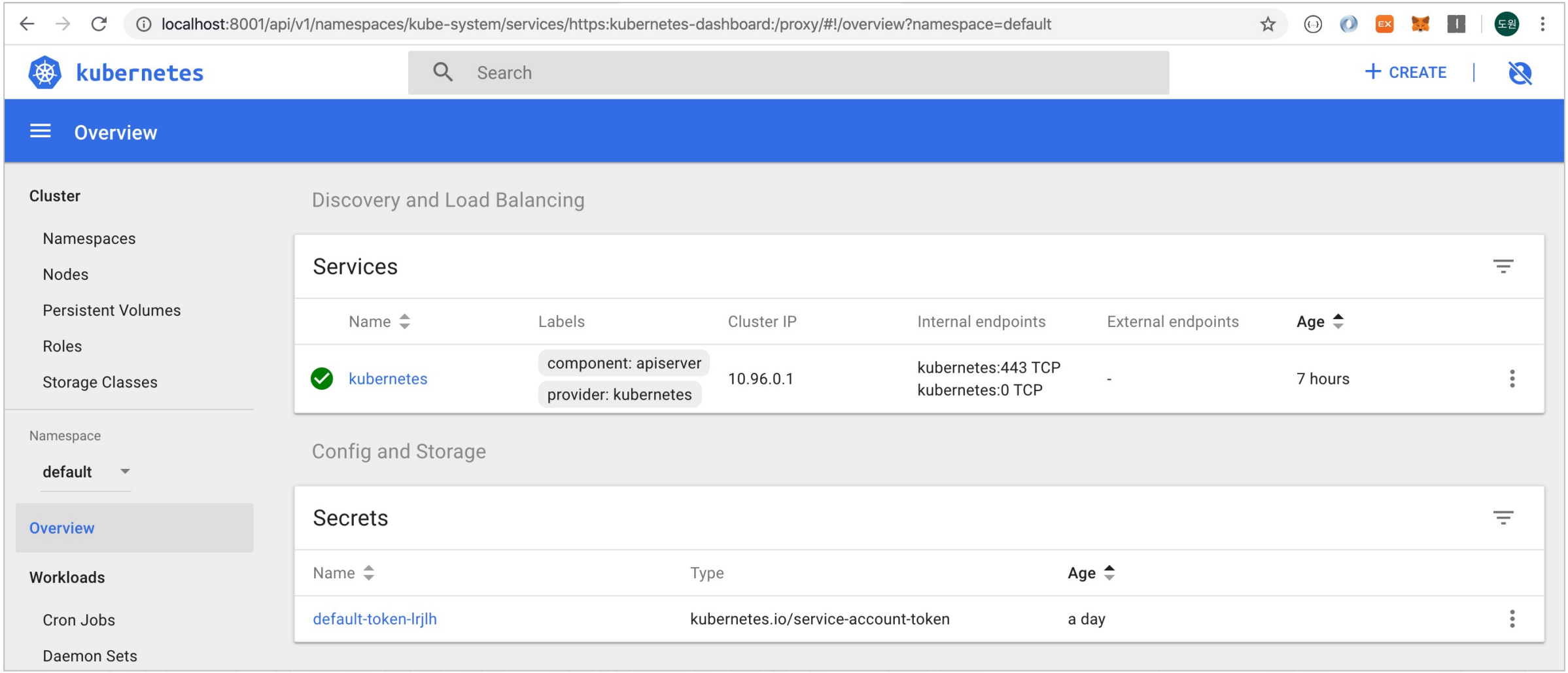
Kubernetes Demo

- ***Kubernetes version***



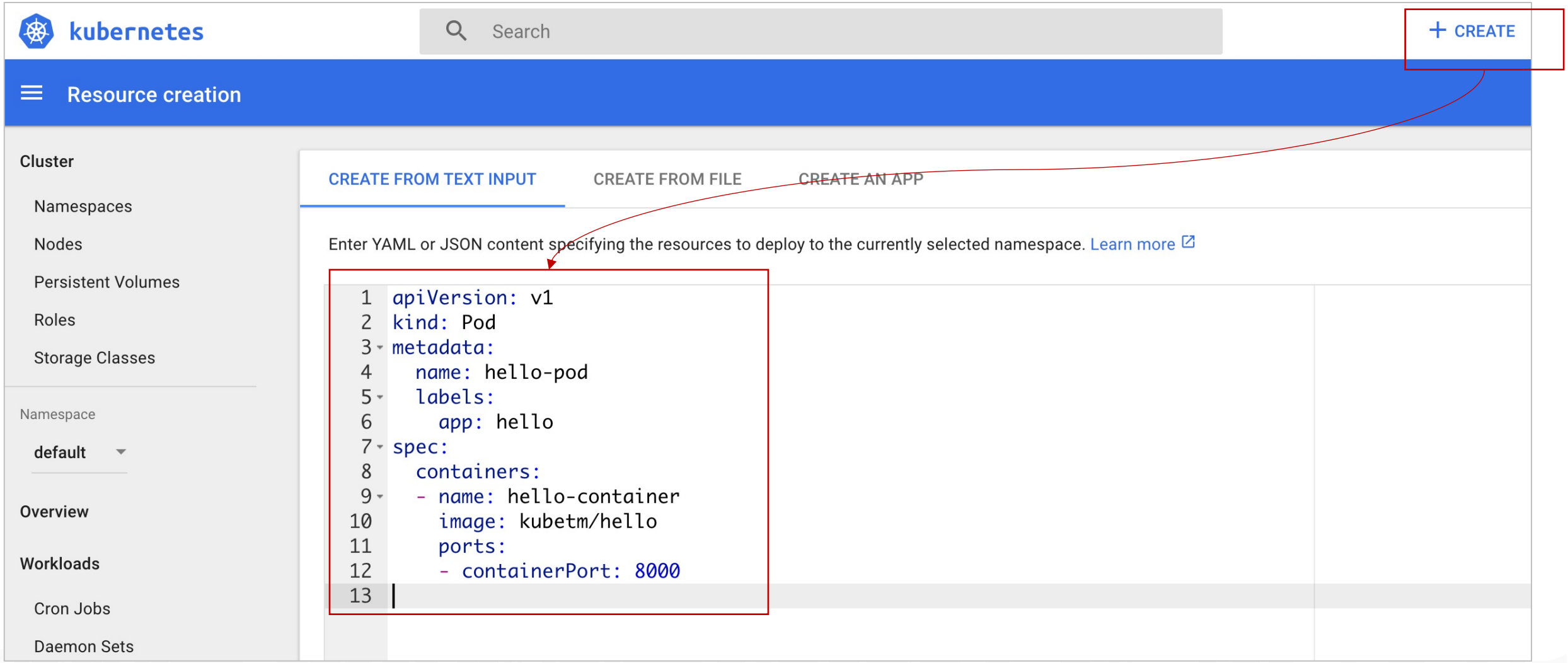
Kubernetes Demo

- ***Kubernetes version***



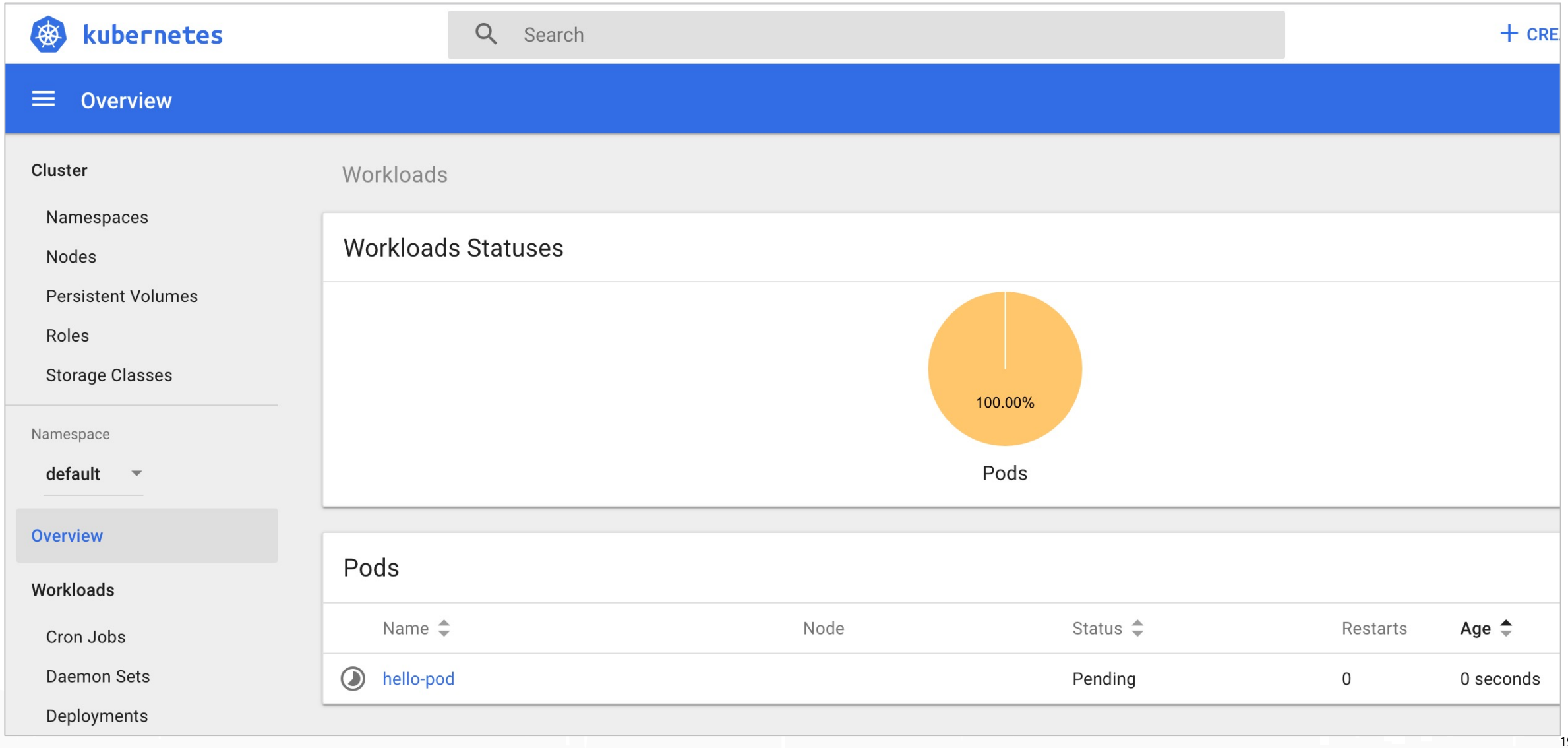
Kubernetes Demo

- ***Kubernetes version***



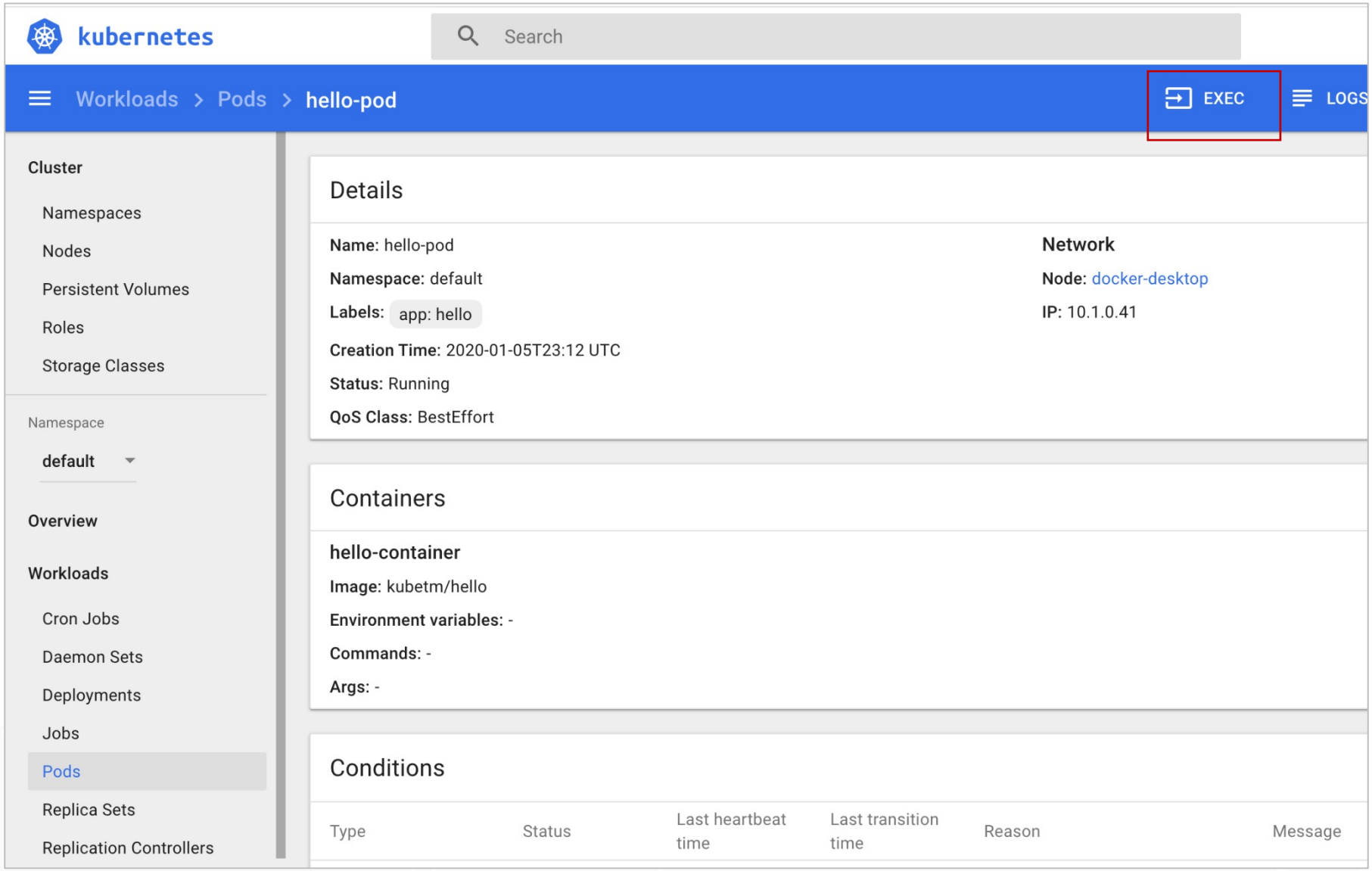
Kubernetes Demo

- ***Kubernetes version***



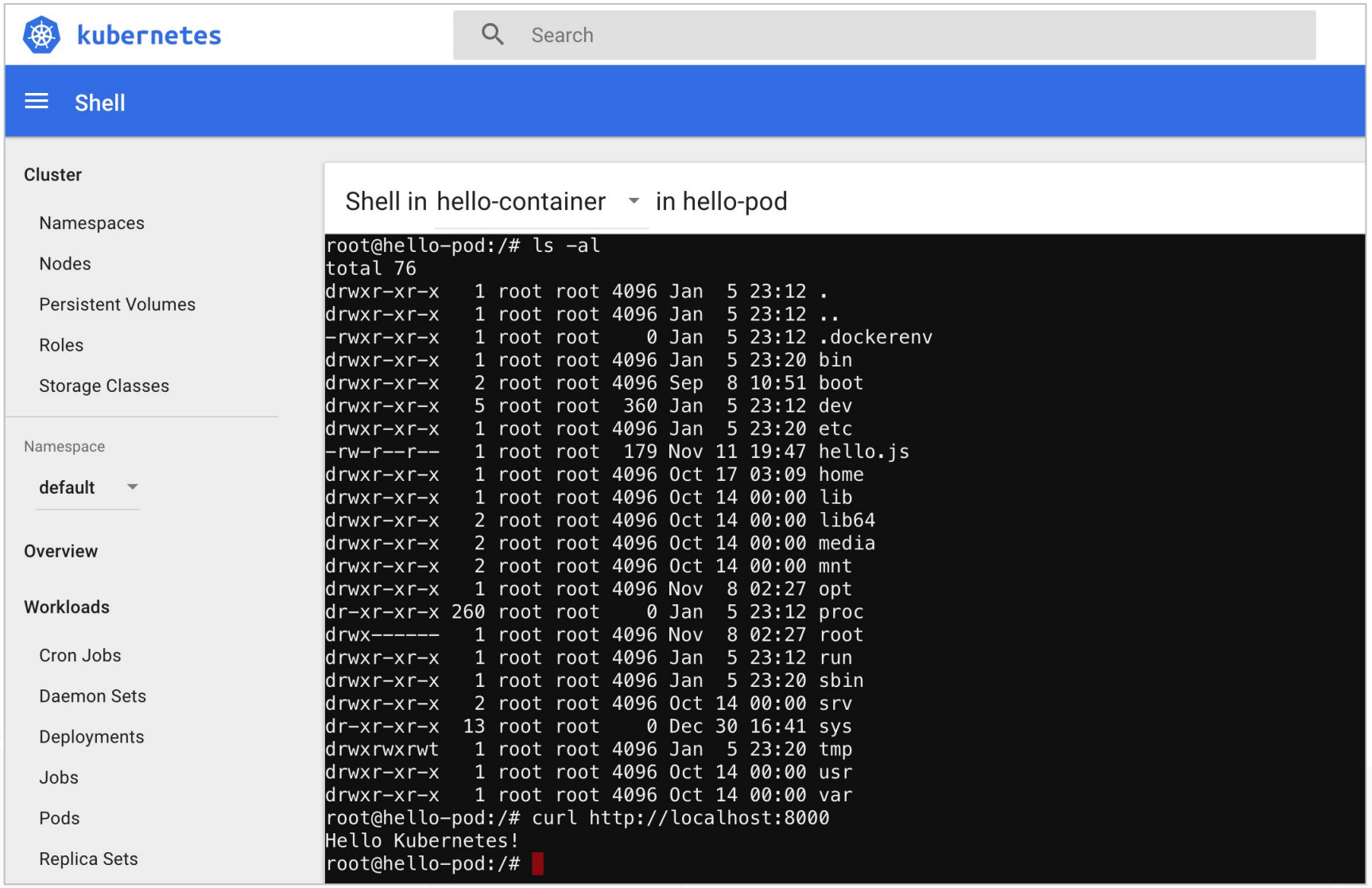
Kubernetes Demo

- ***Kubernetes version***



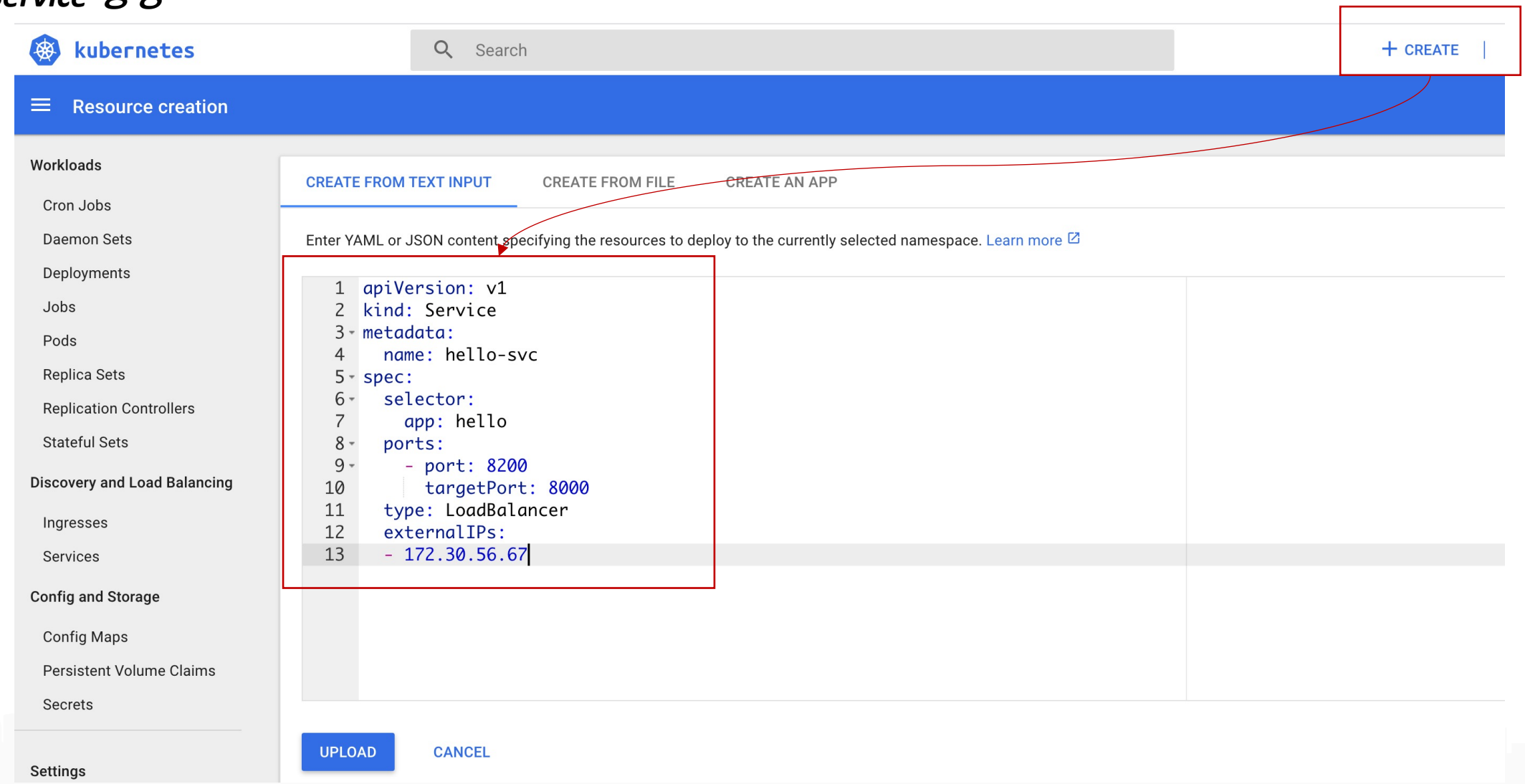
Kubernetes Demo

- ***Kubernetes version***



Kubernetes Demo

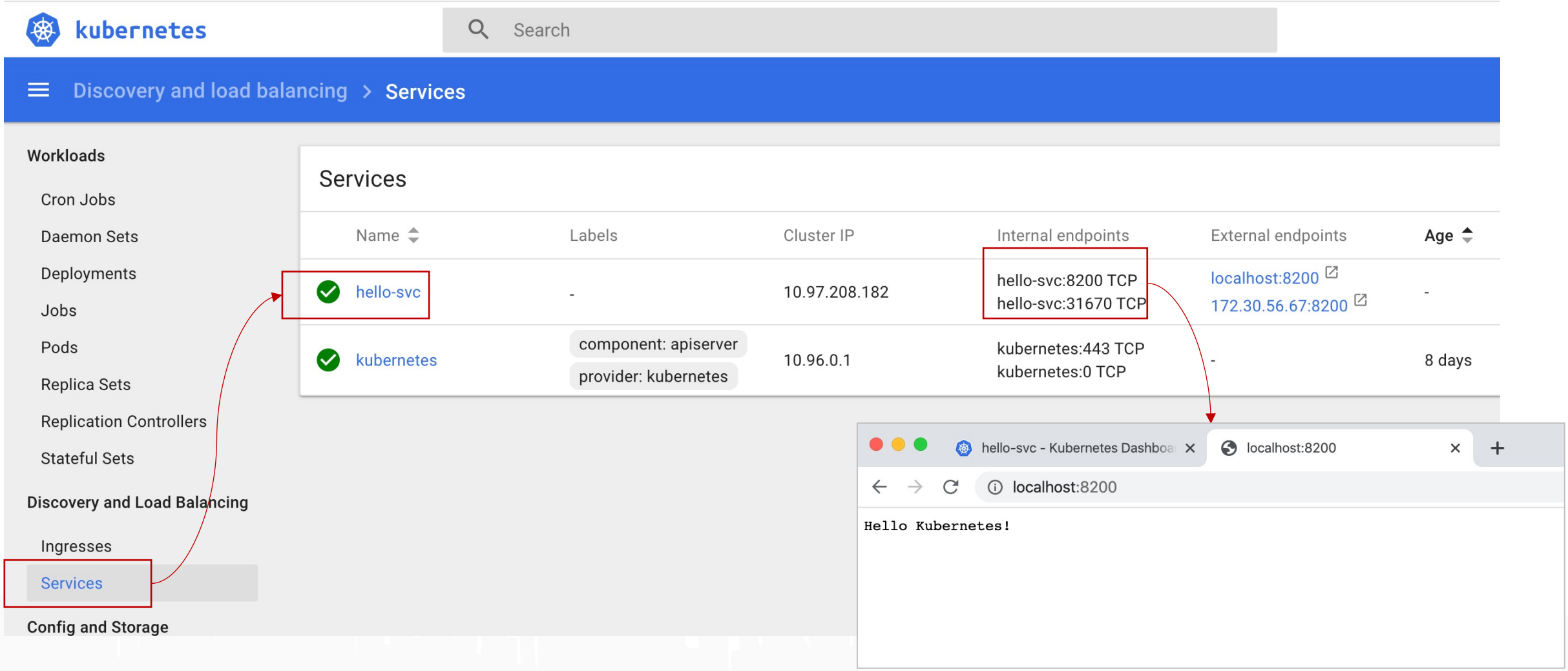
- **Kubernetes version**
  - **Service 생성**



Kubernetes Demo

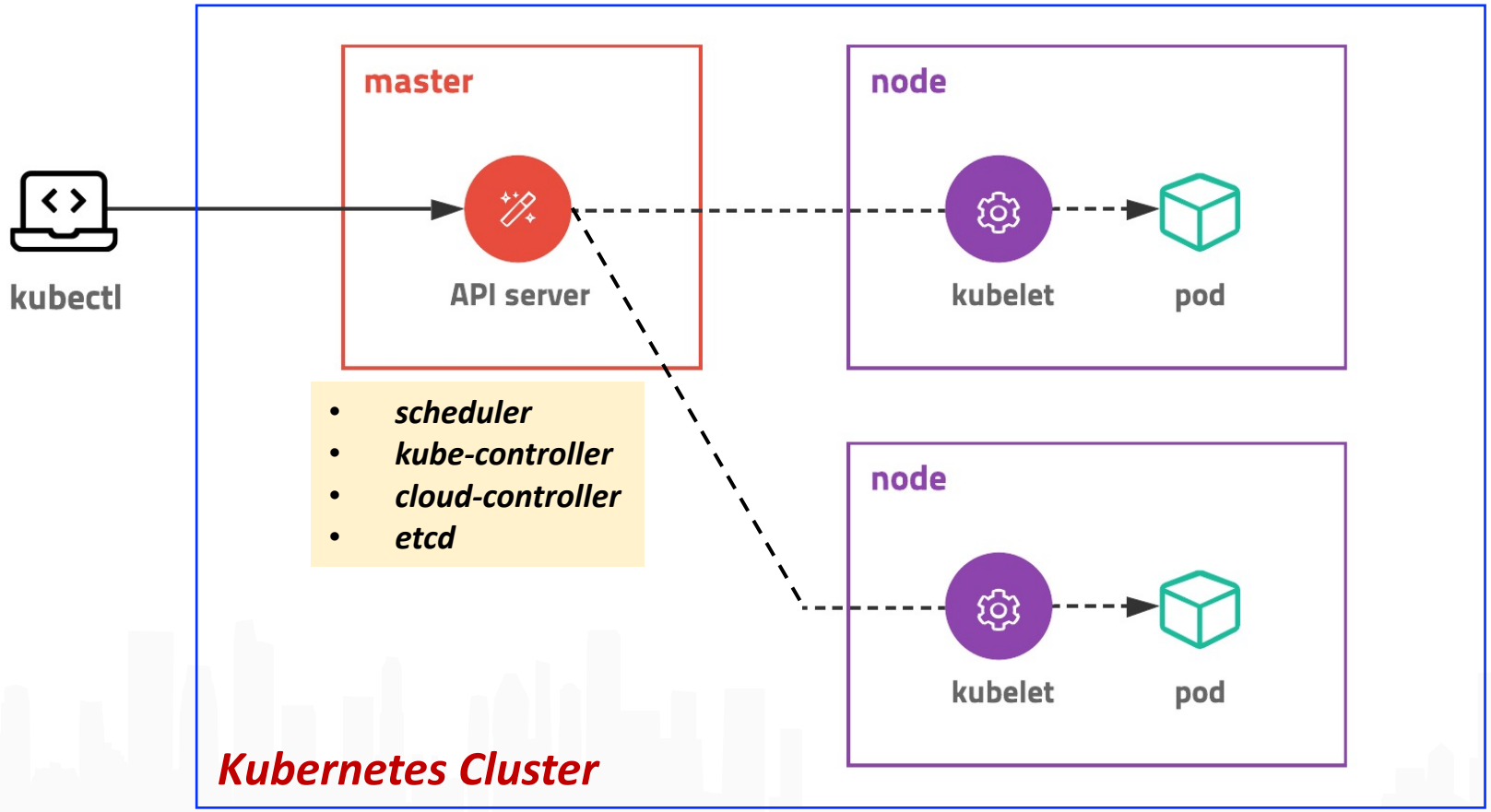
- **Kubernetes version**

  - Service 생성



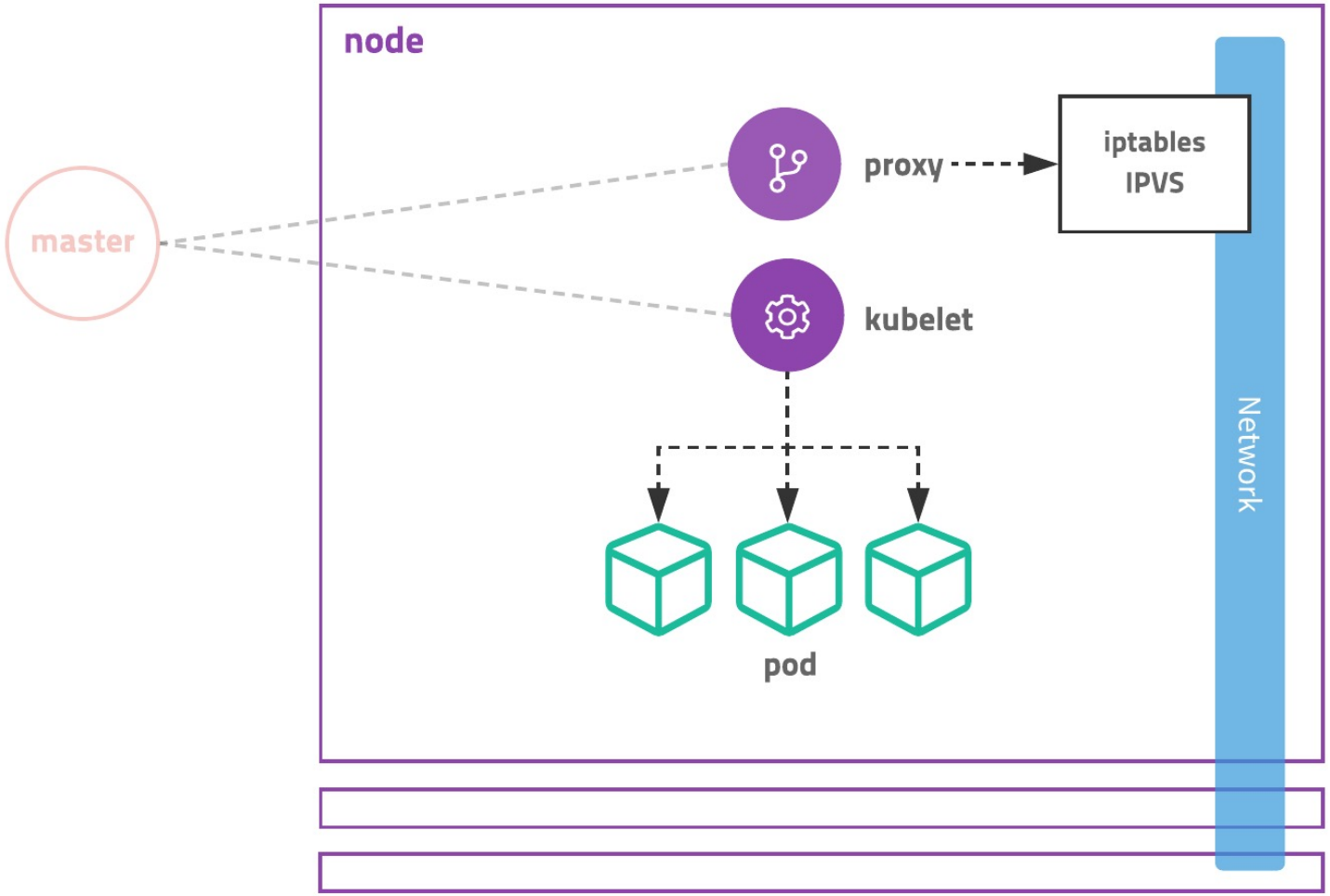
Kubernetes – Cluster, Node

- ***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***

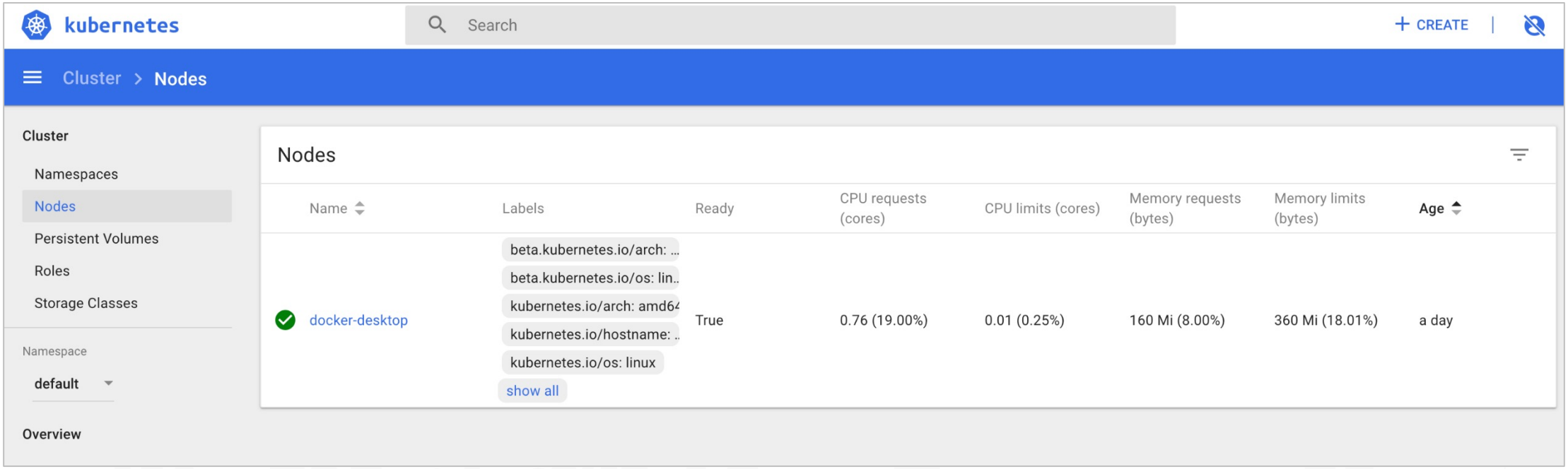


- kubelet
  - 노드에 할당된 Pod의 생명주기 관리
  - Pod 생성, Pod 안의 컨테이너에 이상이 없는지 확인
  - API 서버의 요청을 받아 컨테이너의 로그 전달
- kube-proxy
  - Pod로 연결되는 네트워크 관리
  - TCP, UDP, SCTP 스트림 포워딩

Kubernetes – Cluster, Node

- ***클러스터 생성 시 만들어진 가상 머신이 노드로 등록 되어 있음***
  - $ kubectl get nodes

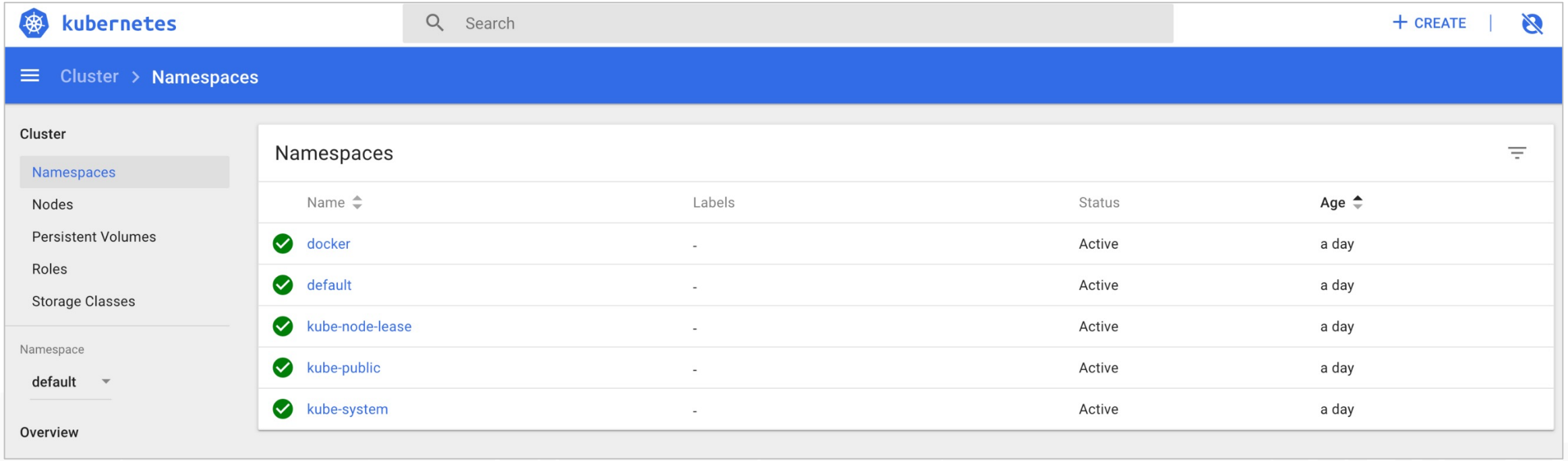
```
kubectl get nodes
NAME                 STATUS   ROLES    AGE   VERSION
docker-desktop       Ready    master   31h   v1.14.6
```



Kubernetes – Namespace

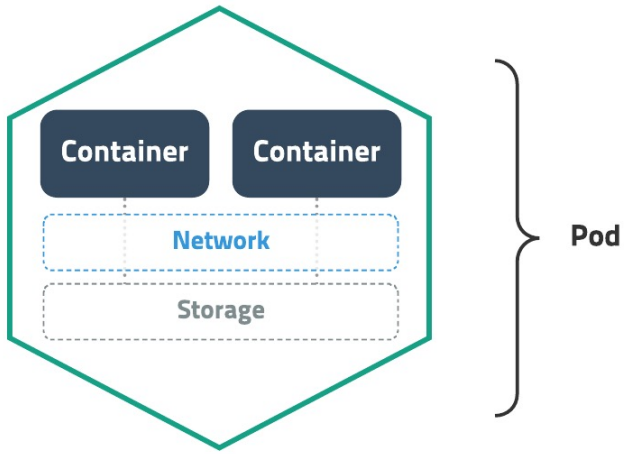
- *네임스페이스*
  - Kubernetes 클러스터 안에 가상 클러스터 생성

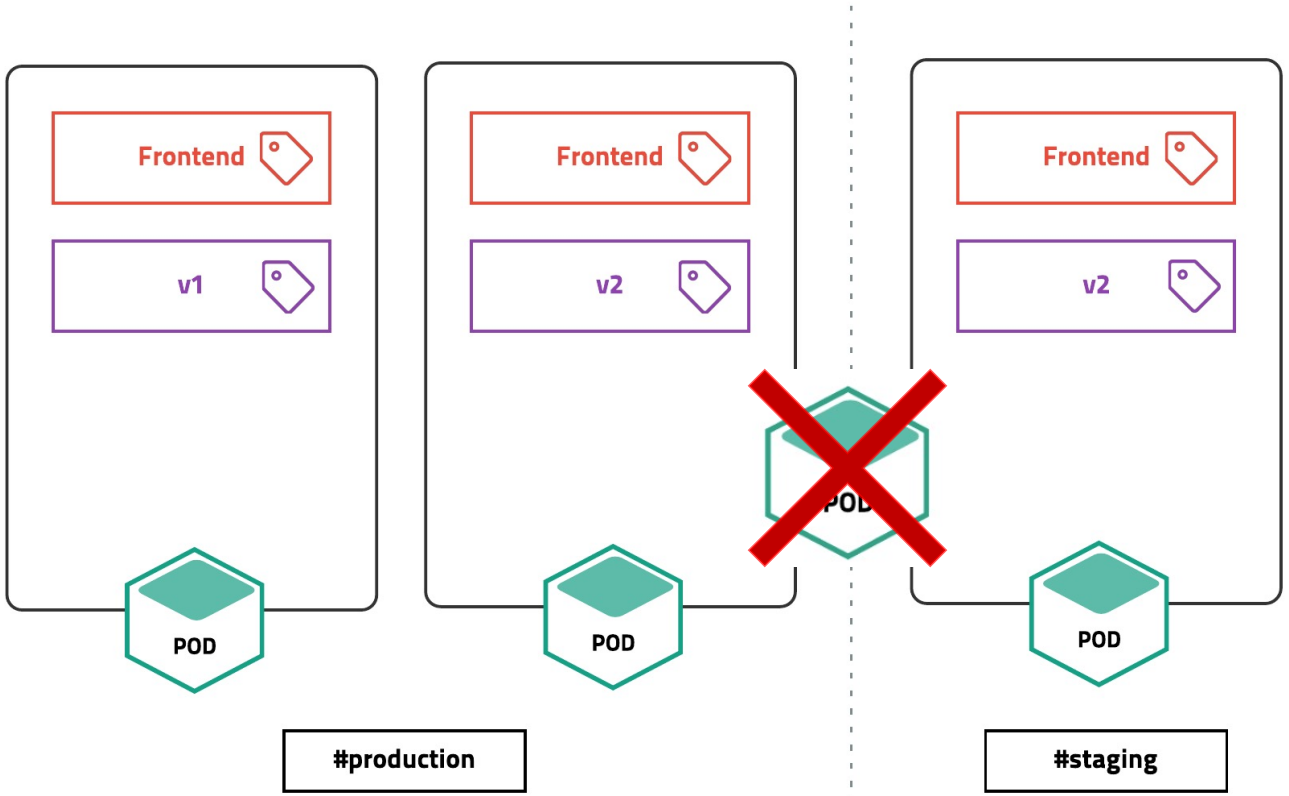
```
kubectl get namespaces
NAME              STATUS   AGE
default           Active   31h
docker            Active   31h
kube-node-lease   Active   31h
kube-public       Active   31h
kube-system       Active   31h
```



Kubernetes – Pod

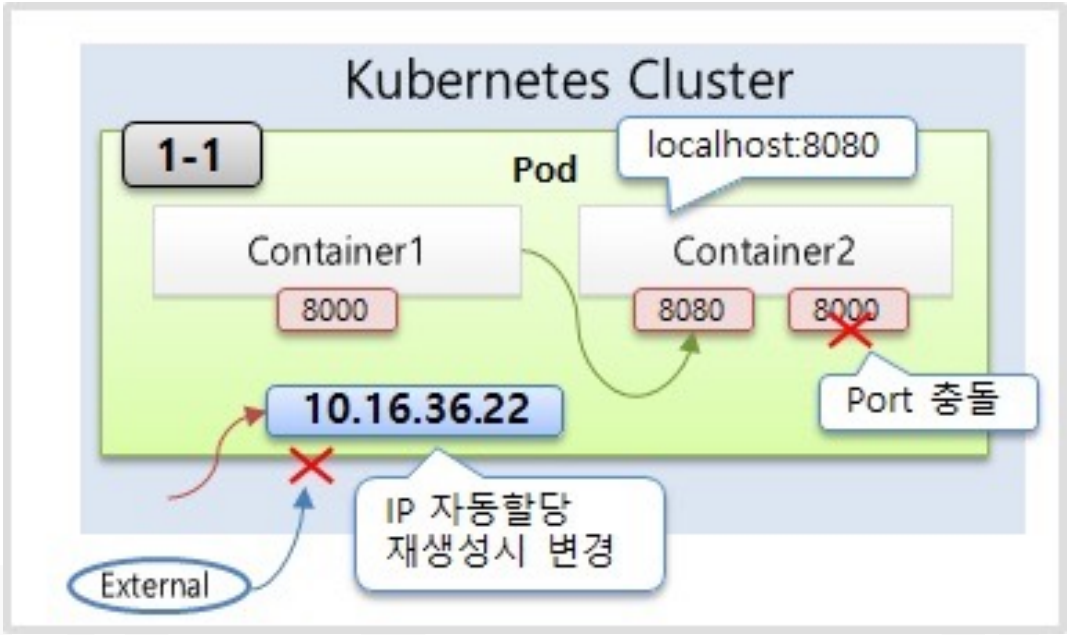
- 컨테이너가 모인 집합체 단위, 하나 이상의 컨테이너로 구성
- Docker + Kubernetes 에서는 하나의 컨테이너라도 반드시 Pod 에 포함하여 배포



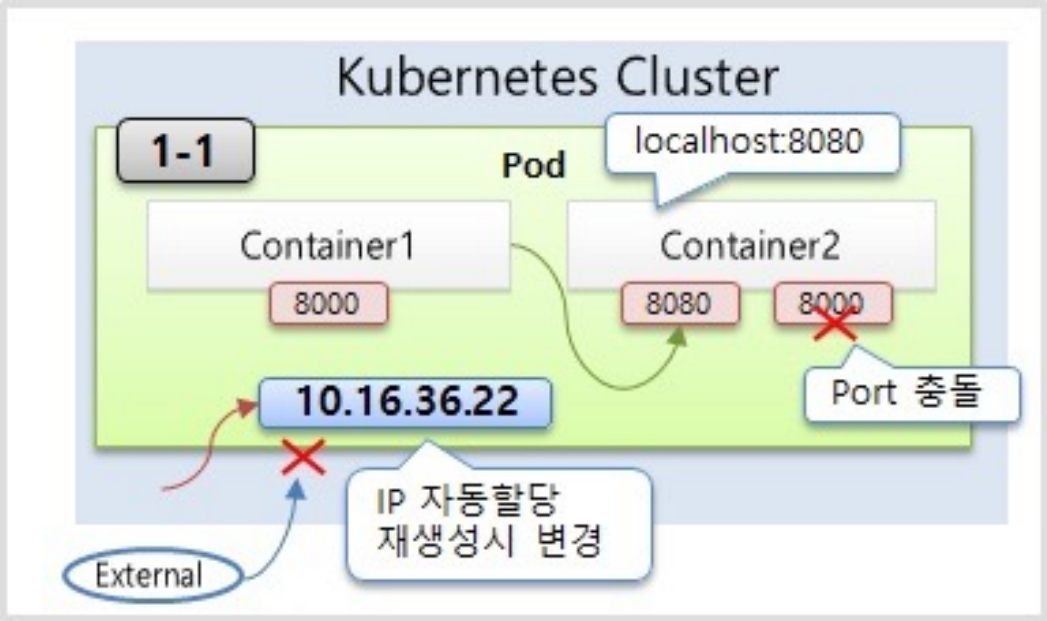


Kubernetes – Pod

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



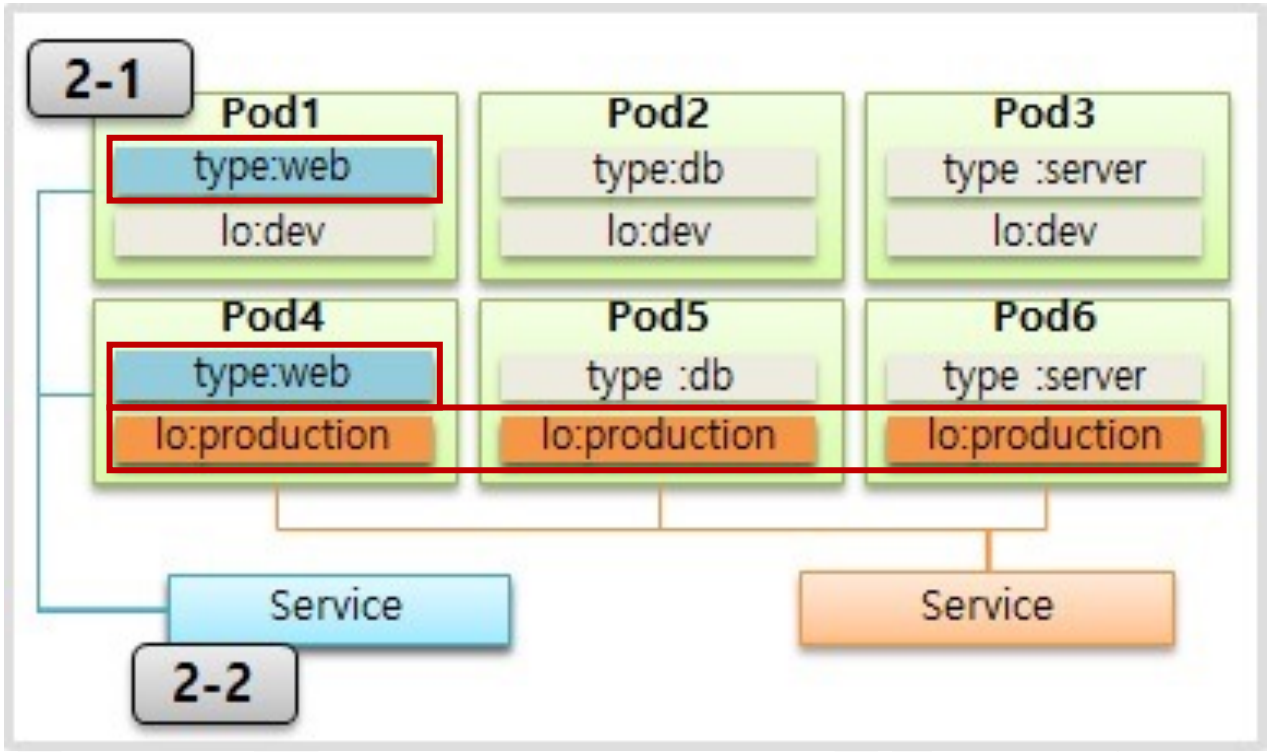
Kubernetes – Pod




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
spec:
  containers:
  - name: container1
    image: kubetm/p8000
    ports:
    - containerPort: 8000
  - name: container2
    image: kubetm/p8080
    ports:
    - containerPort: 8080
```

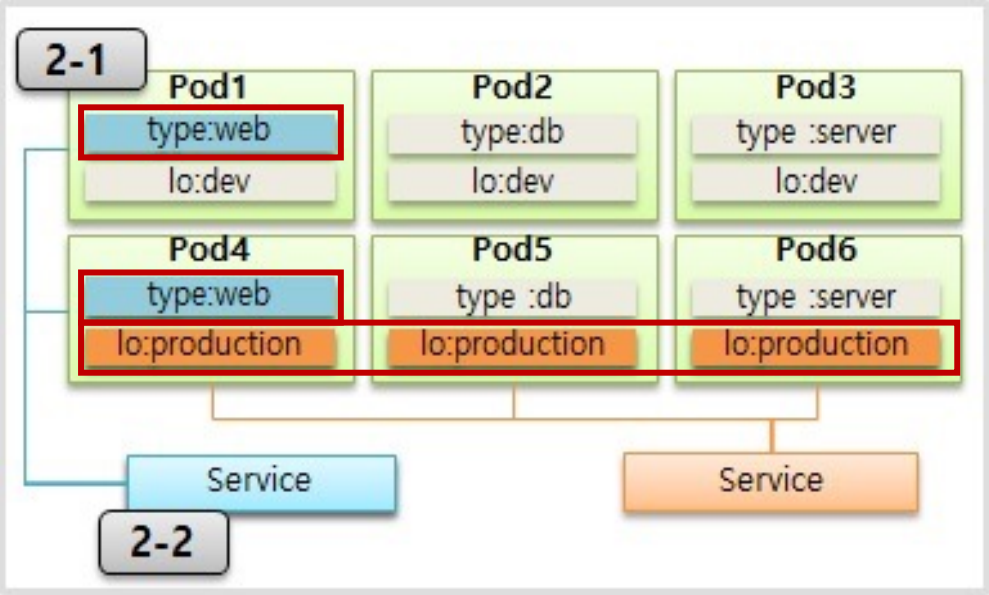
Kubernetes – Pod

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



Kubernetes – Pod

- **Label 지정**

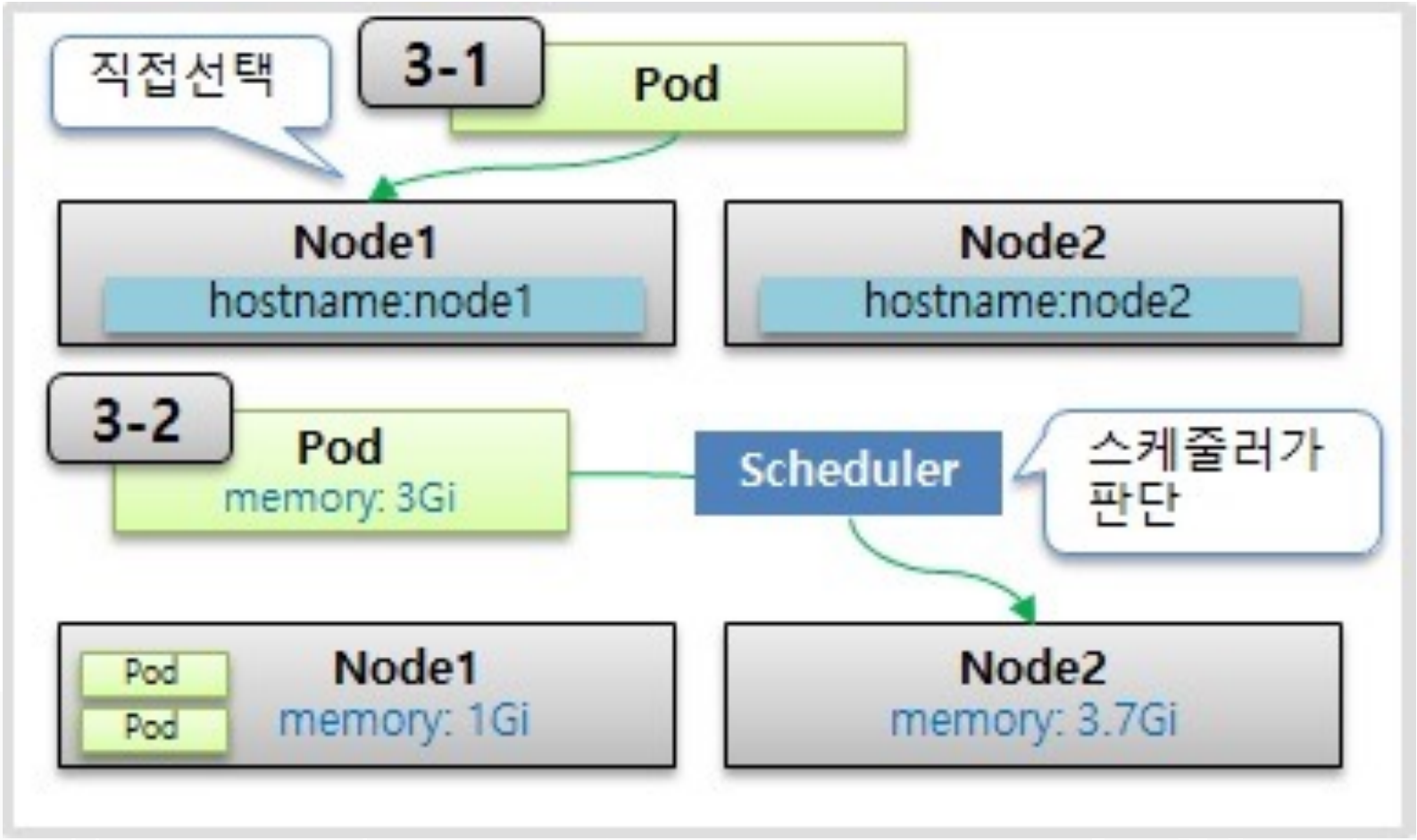



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-2
  labels:
    type: web
    lo: dev
spec:
  containers:
  - name: container
    image: kubetm/init
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-1
spec:
  selector:
    type: web
  ports:
  - port: 8080
```

Kubernetes – Pod

- **NodeSchedule**
  - Pod는 하나의 Node에 지정되어 실행 됨
  - 자동/수동 가능



Kubernetes – Pod

- **NodeSchedule**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-3
spec:
  nodeSelector:
    kubernetes.io/hostname: k8s-node1
  containers:
  - name: container
    image: kubetm/init
```

Node label을 selector로 지정

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-4
spec:
  containers:
  - name: container
    image: kubetm/init
    resources:
      requests:
        memory: 2Gi
      limits:
        memory: 3Gi
```

스케줄러가 판단

- 전체 사용가능량을 판단하여 지정
- Pod에 부하가 걸릴 때 리소스를 무한정 사용하지 않기 위하여

Kubernetes – Pod

- *실습) nginx-proxy, echo 애플리케이션을 Pod로 배포*

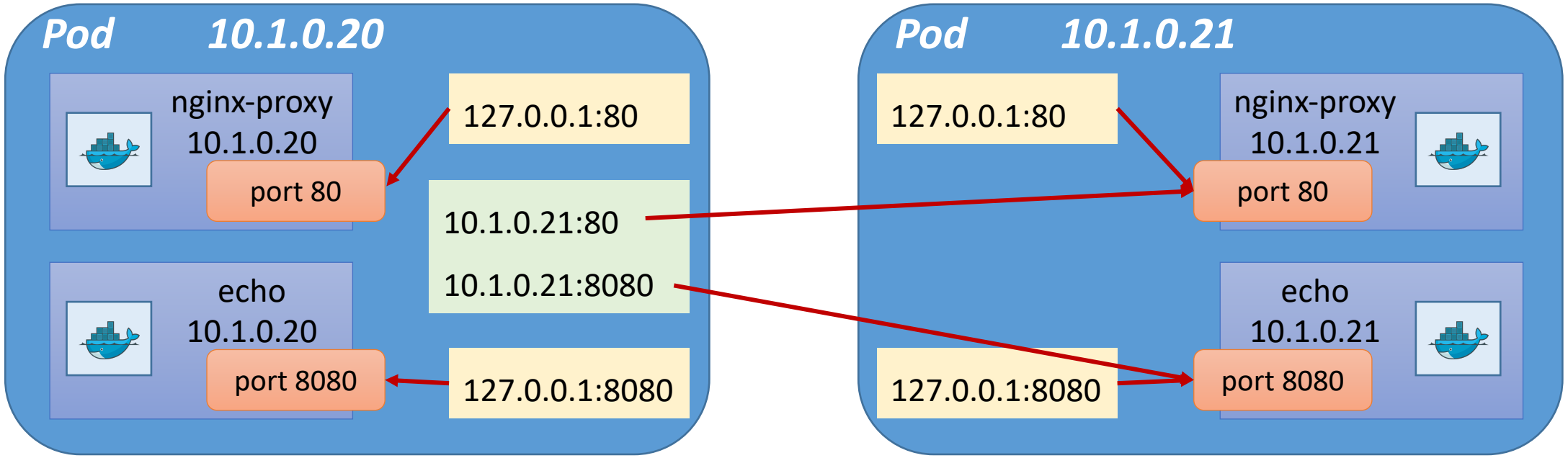
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-echo
spec:
  containers:
  - name: nginx
    image: gihyodocker/nginx:latest
    env:
    - name: BACKEND_HOST
      value: localhost:8080
    ports:
    - containerPort: 80
  - name: echo
    image: gihyodocker/echo:latest
    ports:
    - containerPort: 8080
```

simple-pod.yaml

```
$ 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)
```

Kubernetes – Pod

- Pod에 할당 된 가상IP 주소는 해당Pod에 속하는 모든 컨테이너가 공유



Kubernetes – ReplicaSet

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

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: echo
  labels:
    app: echo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: echo
  template: # template 아래는 파드 리소스 정의와 같음
    metadata:
      labels:
        app: echo
```

simple-replicaset.yaml

```yaml
  spec:
    containers:
    - name: nginx
      image: gihyodocker/nginx:latest
      env:
      - name: BACKEND_HOST
        value: localhost:8080
      ports:
      - containerPort: 80
    - name: echo
      image: gihyodocker/echo:latest
      ports:
      - containerPort: 8080
```

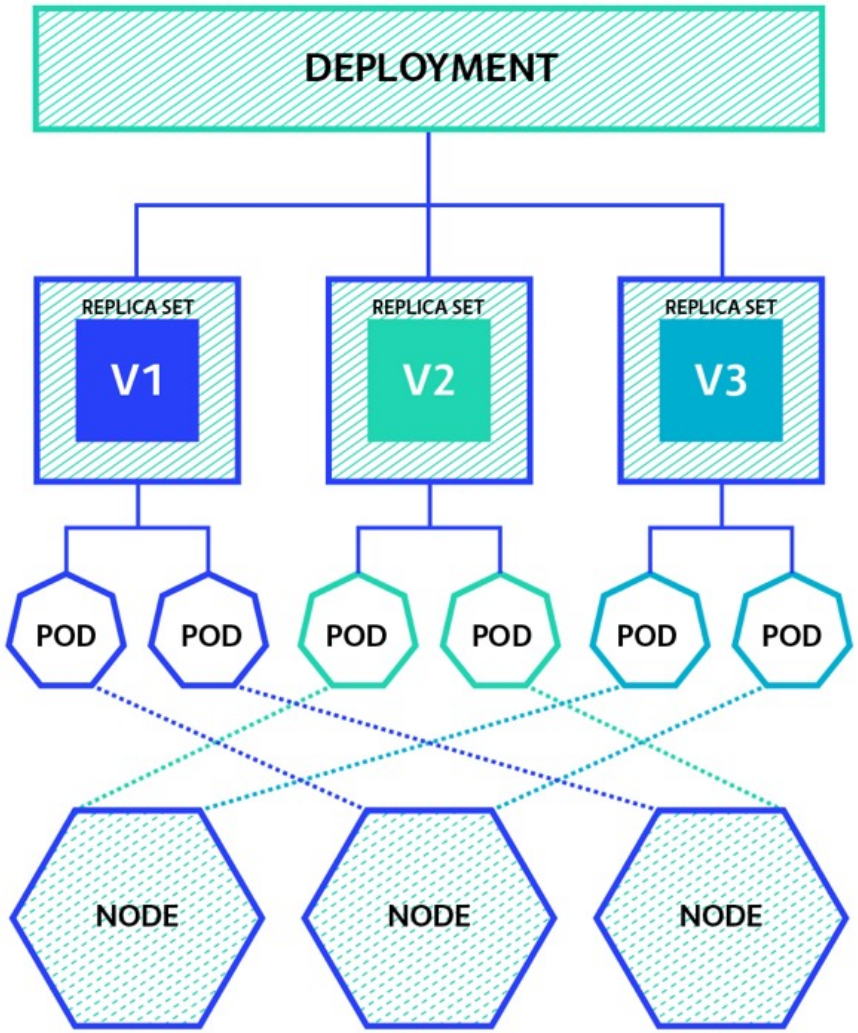
$ kubectl apply -f simple-replicaset.yaml

$ kubectl get pod

$ kubectl delete -f simple-relicaset.yaml

Kubernetes – Deployment

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



Kubernetes – Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo
  labels:
    app: echo
spec:
  replicas: 4
  selector:
    matchLabels:
      app: echo
  template: # template 아래는 파드 리소스 정의와 같음
    metadata:
      labels:
        app: echo
    spec:
      containers:
      - name: nginx
        image: gihyodocker/nginx:latest
        env:
        - name: BACKEND_HOST
          value: localhost:8080
        ports:
        - containerPort: 80
      - name: echo
        image: gihyodocker/echo:latest
        env:
        - name: HOGE
          value: fuga
        ports:
        - containerPort: 8080
```

simple-deployment.yaml

$ 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 uudo 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

- **Deployement 삭제**

  $ kubectl delete -f simple-deploymenet.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*

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: echo-spring
  labels:
    app: echo
    release: spring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: echo
      release: spring
  template:
    metadata:
      labels:
        app: echo
        release: spring
```

```yaml
  spec:
    containers:
    - name: nginx
      image: gihyodocker/nginx:latest
      env:
      - name: BACKEND_HOST
        value: localhost:8080
      ports:
      - containerPort: 80
    - name: echo
      image: gihyodocker/echo:latest
      ports:
      - containerPort: 8080
```

Kubernetes – Service

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

```yaml
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: echo-summer
  labels:
    app: echo
    release: summer
spec:
  replicas: 2
  selector:
    matchLabels:
      app: echo
      release: summer
  template:
    metadata:
      labels:
        app: echo
        release: summer
```

simple-replicaset-with-label.yaml

```yaml
  spec:
    containers:
    - name: nginx
      image: gihyodocker/nginx:latest
      env:
      - name: BACKEND_HOST
        value: localhost:8080
      ports:
      - containerPort: 80
    - name: echo
      image: gihyodocker/echo:latest
      ports:
      - 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***

```yaml
apiVersion: v1
kind: Service
metadata:
  name: echo
spec:
  selector:
    app: echo
  ports:
    - name: http
      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

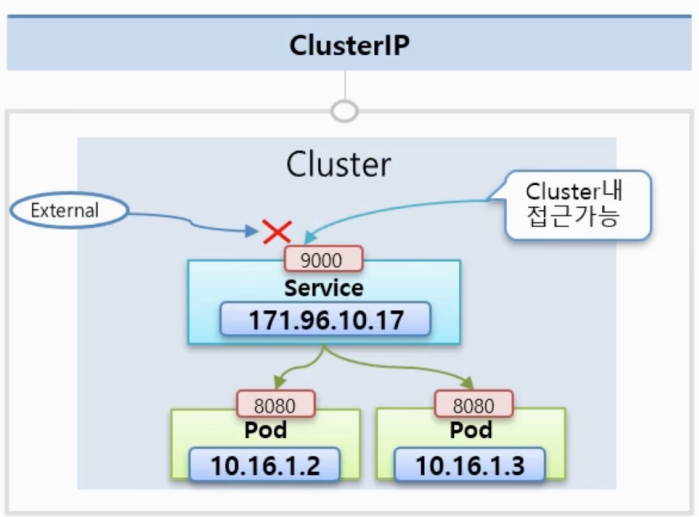
$ kubectl logs -f echo-summer-dtblk -c echo

Kubernetes – Service

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

- ***CluserIP 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

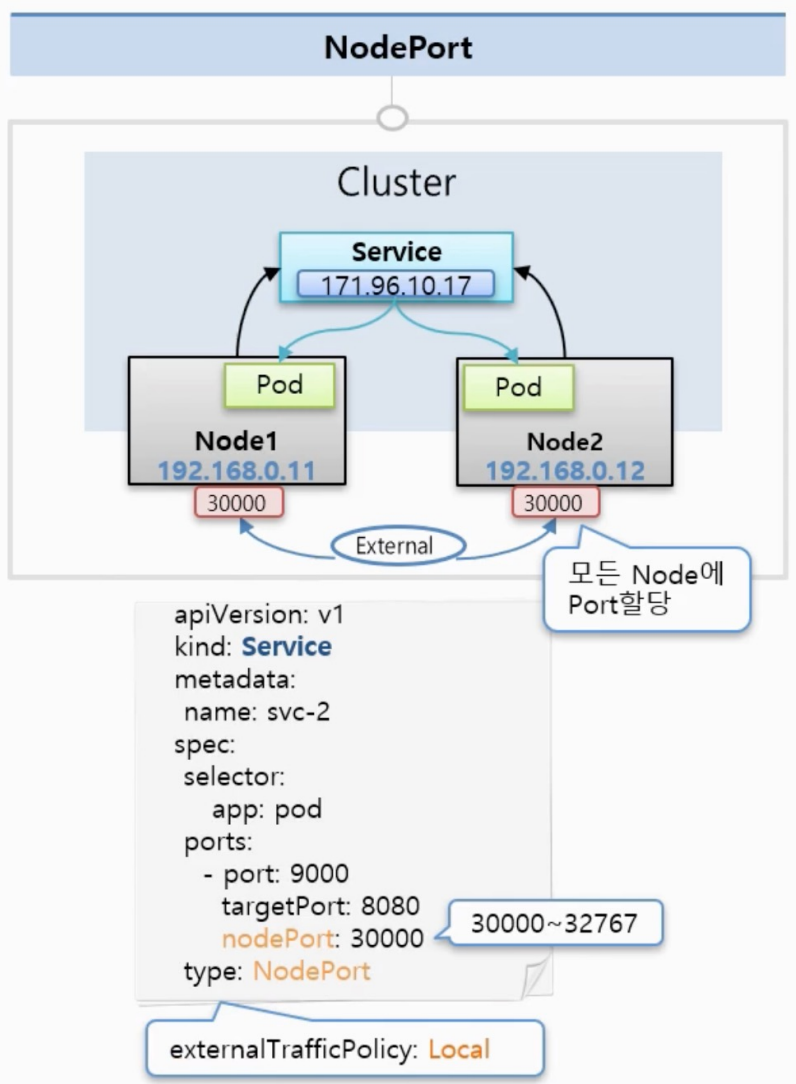



```
apiVersion: v1
kind: Service
metadata:
  name: svc-1
spec:
  selector:
    app: pod
  ports:
    - port: 9000
      targetPort: 8080
  type: ClusterIP
```

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

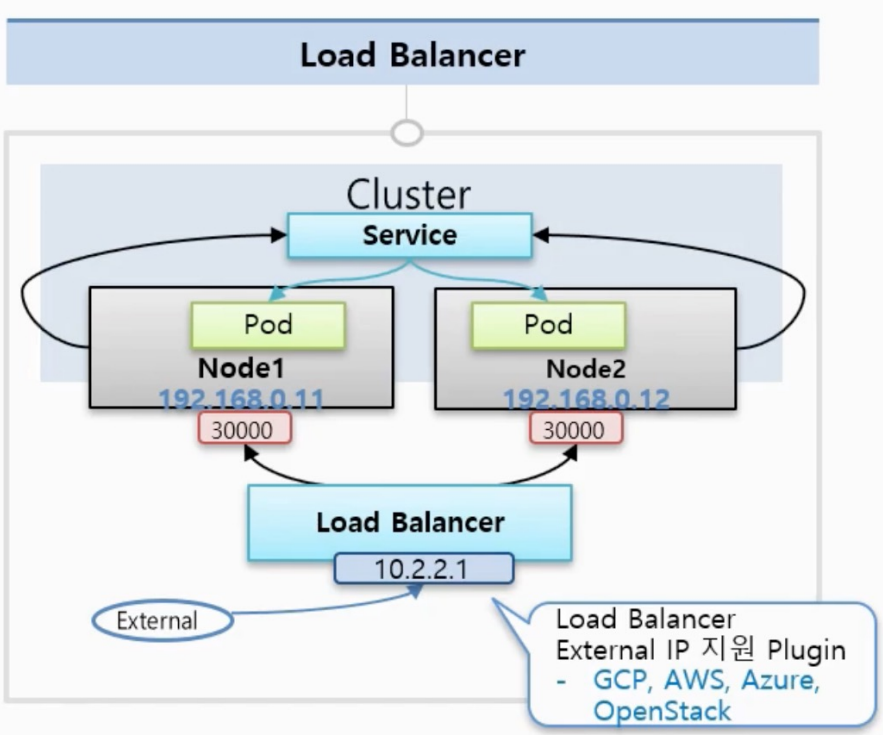
Kubernetes – Service

- ***NodePort Service***
  - ***외부***에서 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

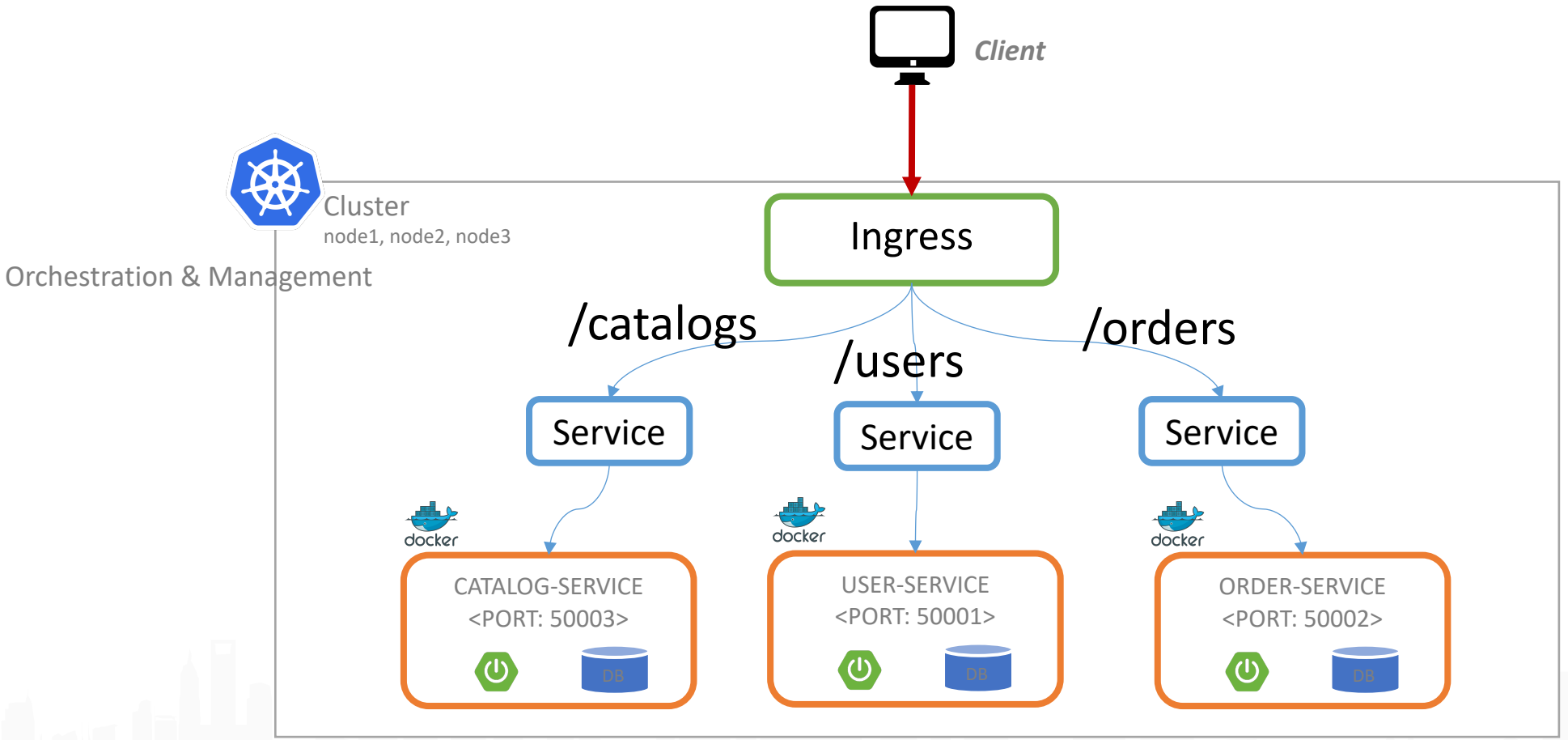
- *LoadBalancer Service*
  - *Cluster에 LB endpoint 제공*
  - *Only available when infra provider gives you a LB (AWS ELB, etc)*
  - *외부 시스템 노출 시 사용*
- *ExternalName Service*
  - *CNAME DNS record to CoreDNS*




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

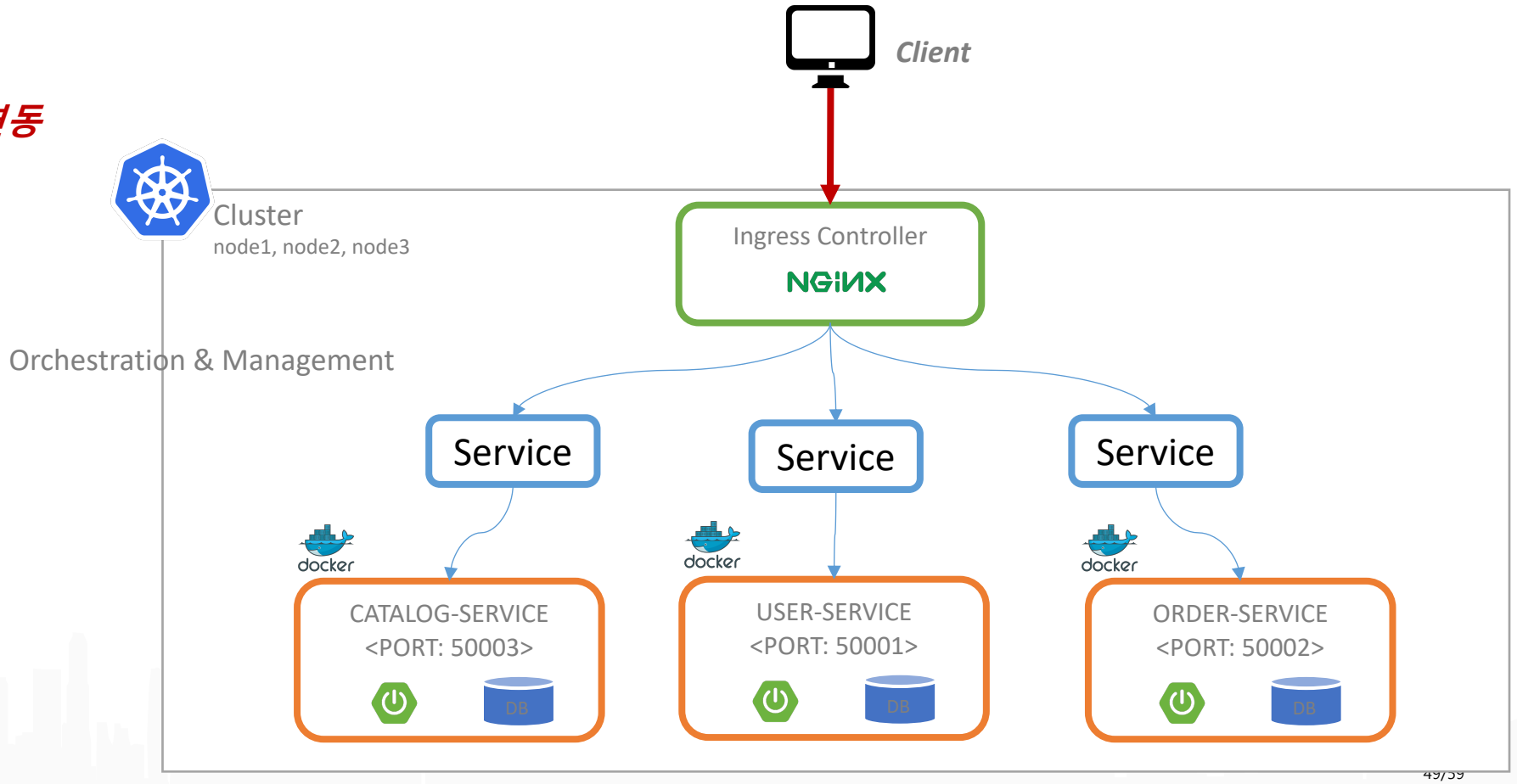
Kubernetes – Ingress

- 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***
  - *Ingress와 Ingress Controller 연동*



Kubernetes – Ingress

```yaml
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-ng
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

```yaml
- 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

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

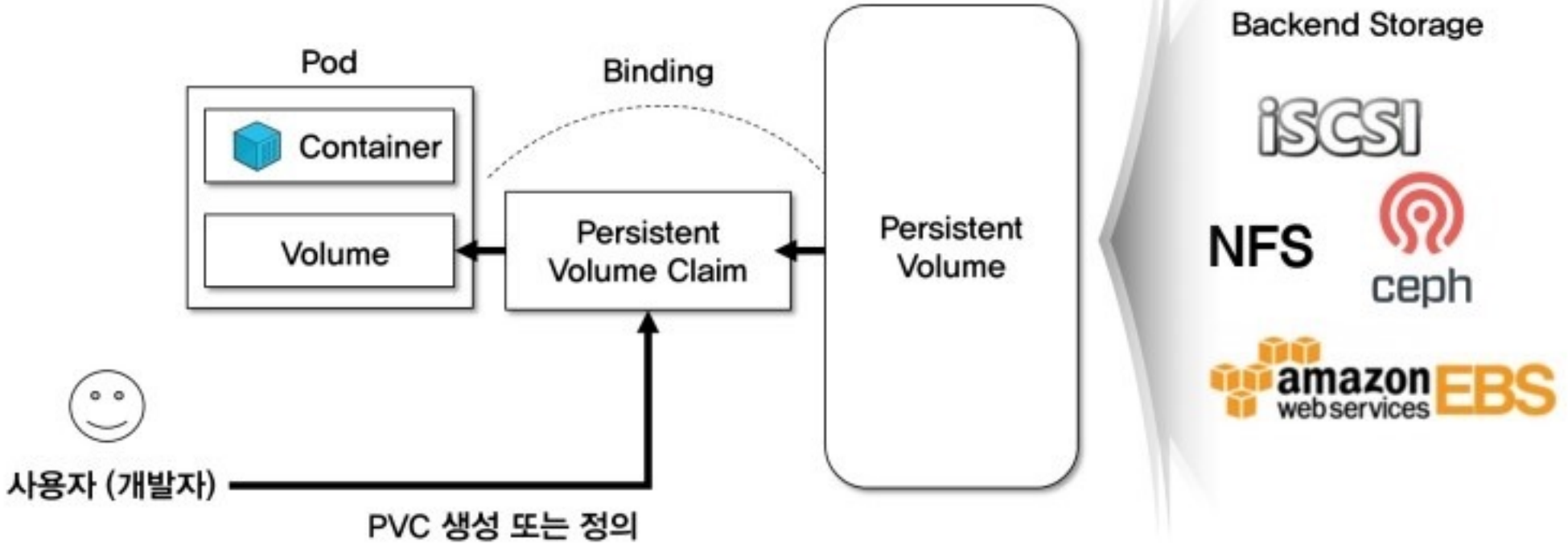
Kubernetes – Persistent Volume

- 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 PVC을 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

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: dev-pv
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  storageClassName: manual
  persistentVolumeReclaimPolicy: Delete
  hostPath:
    path: /Users/dowonlee/Desktop/git/seocho_blockchain/09.src/day17/mydata
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dev-pvc
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 2Gi
  storageClassName: manual
```

Persistent Volume Claim 예제

Kubernetes – Persistent Volume

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

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-volume-test
spec:
  containers:
  - name: container
    image: kubetm/init
    volumeMounts:
    - name: pvc-pv
      mountPath: /my_mount
  volumes:
  - name : pvc-pv
    persistentVolumeClaim:
      claimName: dev-pvc
```

Pod 예제

```yaml
template:
  metadata:
    labels:
      app: test-deployment
  spec:
    containers:
    - name: test-deployment
      image: nginx
      ports:
      - containerPort: 8080
      volumeMounts:
      - mountPath: "/var/log/test.log"
        name: dev-volume
    volumes:
    - name: dev-volume
      persistentVolumeClaim:
        claimName: dev-pvc
```

Deployment 예제

■ 네트워크 볼륨 – Cloud

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

```yaml
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:
        volumeID: "<volume id>"
        fsType: ext4
```