**섹션0 – Introduction**

쿠버네티스 사용 이유 – 큰 기업들은 대규모의 서비스 운영을 위해 최대한 자원을 효율적으로 써야 비용적으로 유리함. 가상화 기술에 관심을 가짐.

최초의 자원격리 기술 : linux (chroot, namespace, cgroup)

vm / rackspace / openstack /

자동화는 되는데 시스템 효율이 안남 -> vm가상화를 위한 무거운 os를 띄워야하는 근본적인 부분

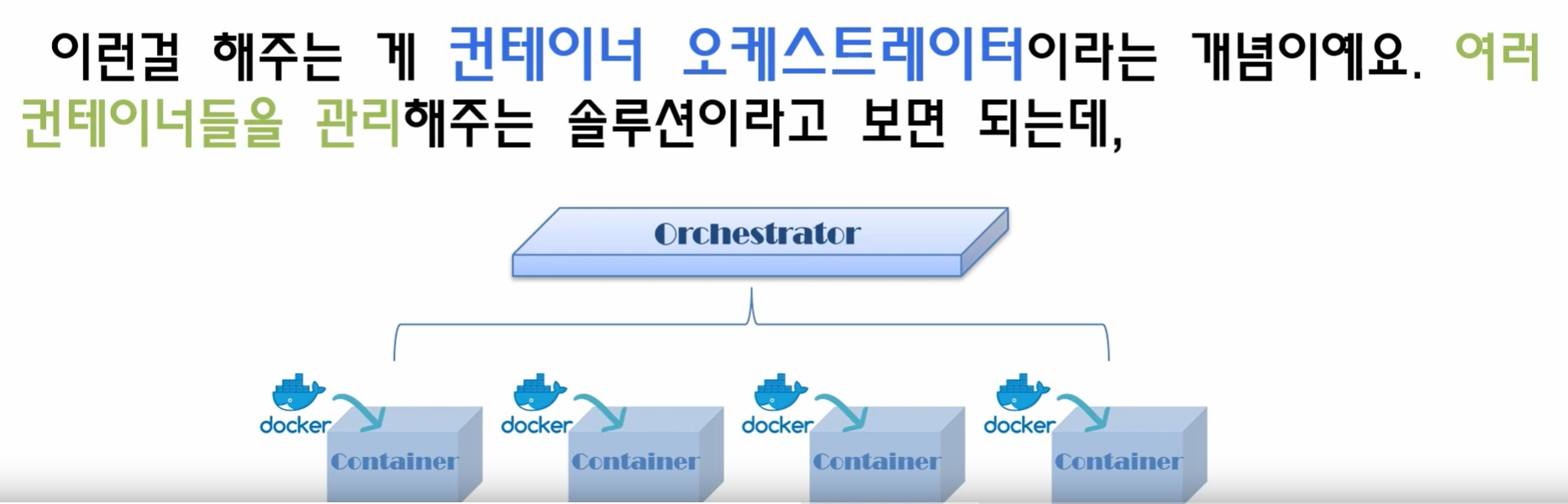
**컨테이너 가상화 기술**

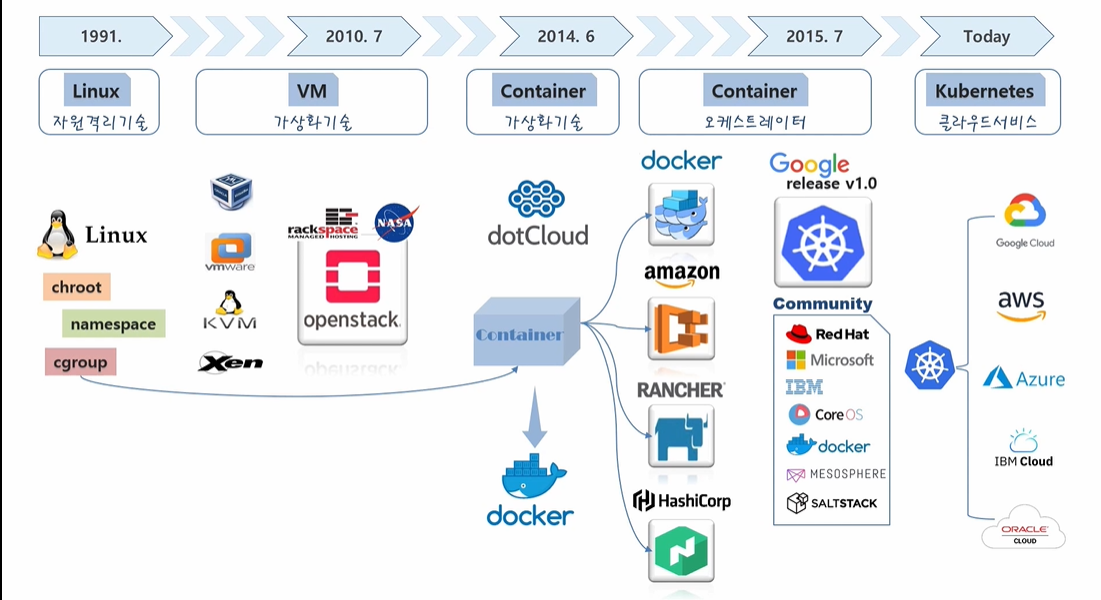
-서비스간에 자원격리를 하는데 os를 별도로 안띄워도 됨

-Os 가동시간이 없어서 자동화시에 엄청 빠름

-자원 효율도 매우 높음

도커 -> 하나의 서비스를 컨테이너로 가상화 후 배포 / 엄청 많은 서비스 운영히 일일히 배포 운영X

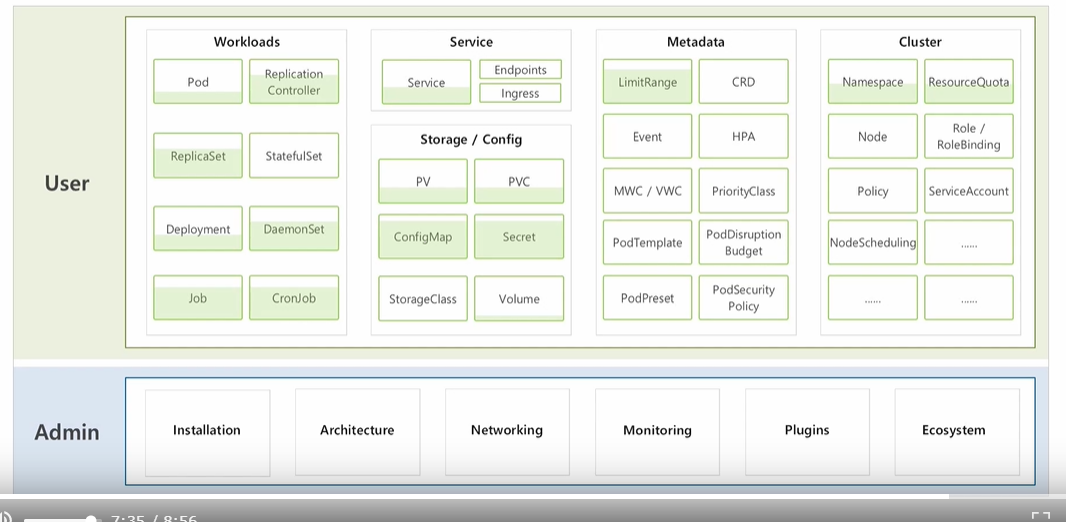
**컨테이너 오케스트레이터** – 여러 컨테이너 관리해주는 솔루션



오픈소스 – 직접 설치 후 최적화 가능

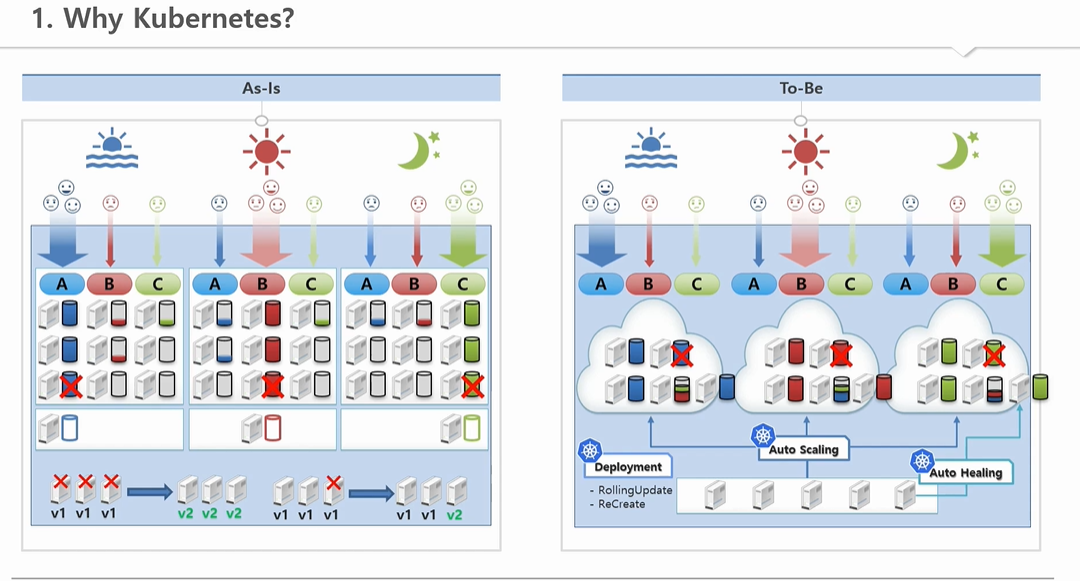
클러스터 운영하는 admin

쿠버네티스 기능 활용해 서비스 배포하는 user



**섹션1 – [기초편] 기초 다지기**

**Why kuberentes?**



12대(백업서버까지) -> 5대(여분서버)!

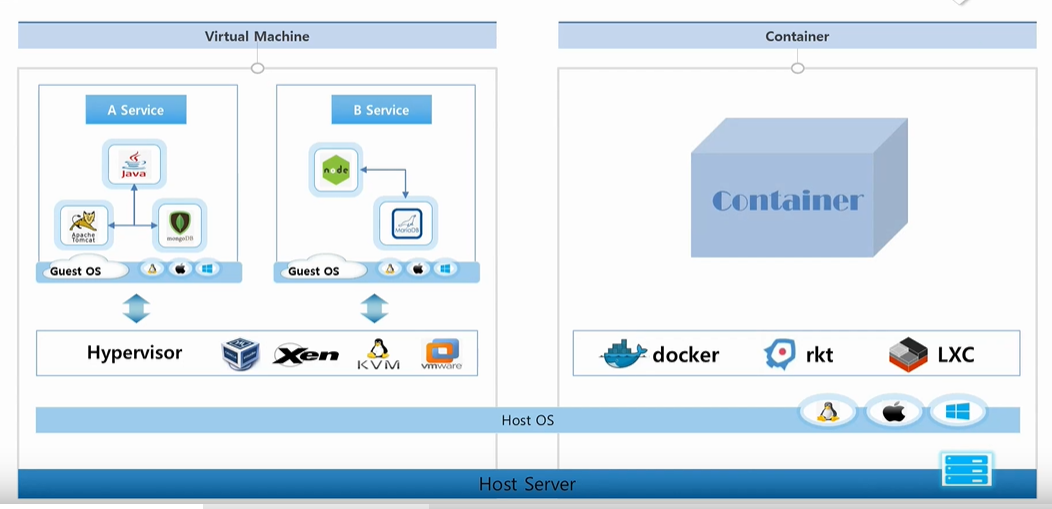
Auto scaling – 트래픽 양에 따라 서비스 자원량 변경시켜줌

Auto healing – 장애가 난 서버위에 있는 서비스들이 다른 서버로 옮겨줌 (여분 서버) -> 서비스 유지 가능

Deployment – 버전 업데이트 자동적으로 처리, rollingupdate, recreate

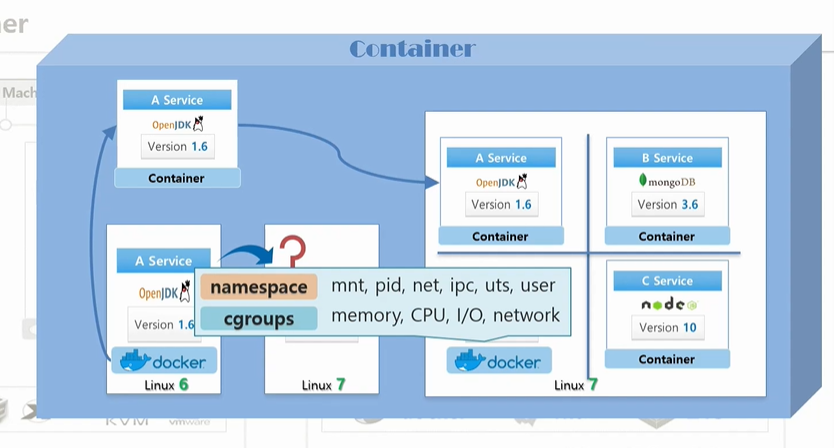
서버적음 – 유지비용 절약

**VM vs Container**

****

Virtual machine : 한 서버 – host os – hypervisor – guest os를 통한 vm

container: 한 서버 – host os – 컨테이너 가상화 (도커 갓 lxc 등)



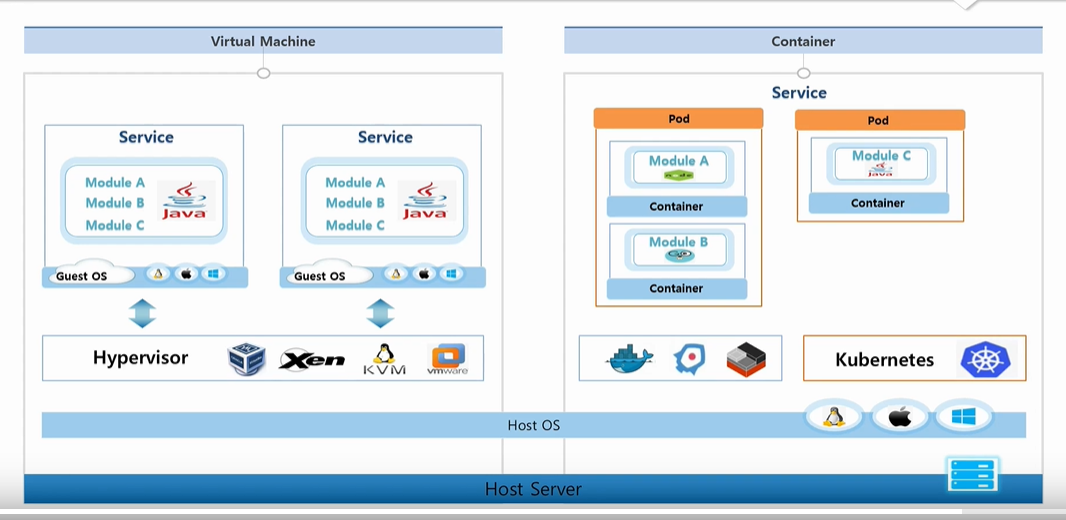
여러 컨테이너들 간의 host자원 분리해서 사용하게 함

Namespace(커널 관련 영역), cgroups(자원 관련 영역)를 사용해 격리

**도커와 같이 컨테이너 가상화 솔루션들은 os에서 제공하는 자원격리기술을 이용해서 컨테이너 단위로 서비스 분리할 수 있음**

**–> 컨테이너 가상화가 깔린 os서는 개발환경에 상관없이 배포 가능**

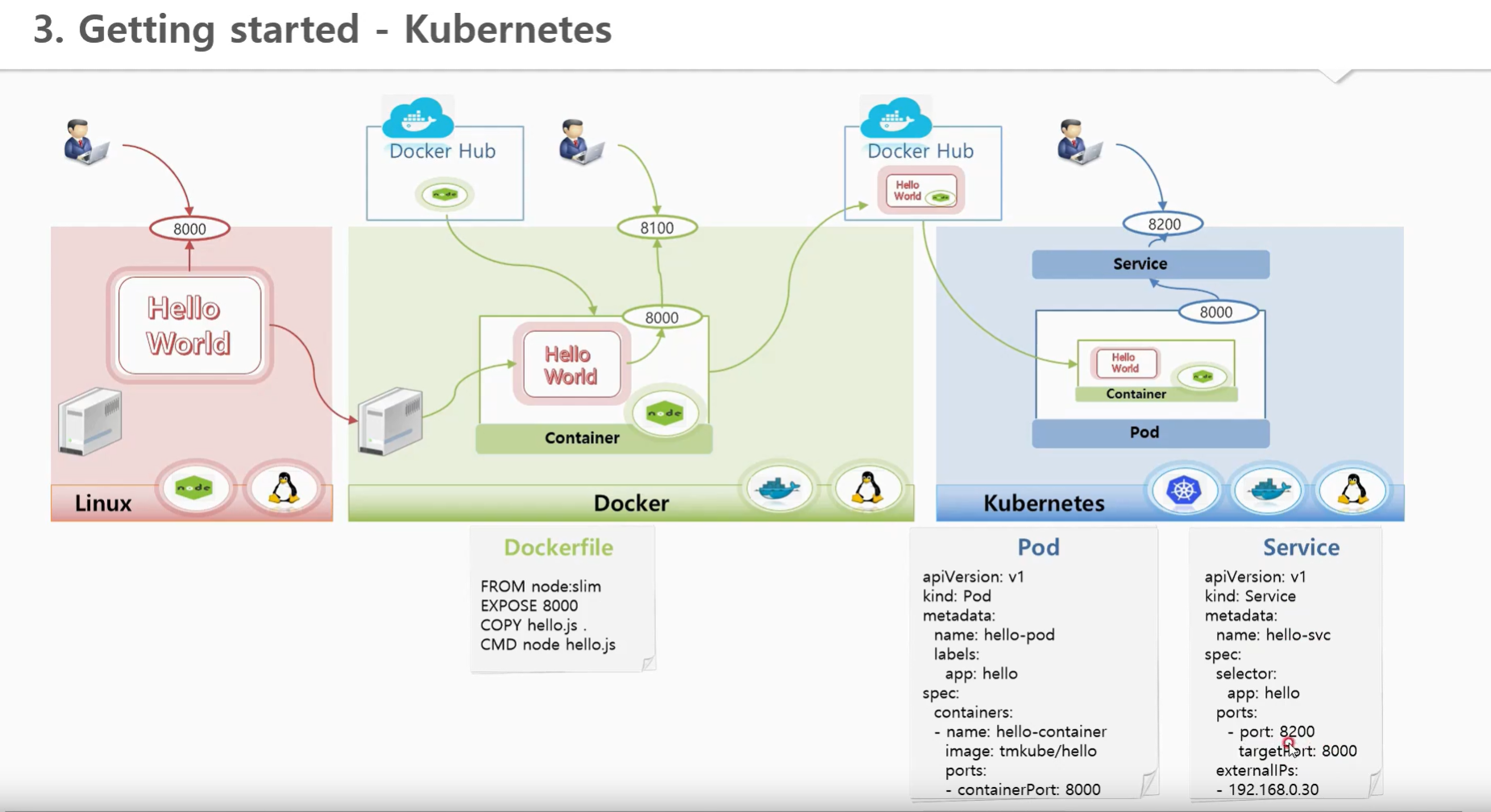
단점 : 리눅스 os에서 window os 못띄움 – 완벽한 격리가 안되어서 위험



VM : 한 패키지 때문에 모듈C 때문에 게스트 OS 하나 더 띄워야 함

Container : **pod – 컨테이너 묶음, 배포 단위** / 시스템을 모듈별로 쪼개서 개발할 때 효과있음

**Getting started - Kubernetes**



**Getting started – Kubernetes – 실습**

쿠버네티스 대시보드 -> 보안 취약으로 권장하지 않음

**3-1) Pod**

apiVersion: v1

kind: Pod

metadata:

name: hello-pod

labels:

app: hello

spec:

containers:

- name: hello-container

image: kubetm/hello

ports:

- containerPort: 8000

**3-2) Service**

apiVersion: v1

kind: Service

metadata:

name: hello-svc

spec:

selector:

app: hello -> pod의 키밸류와 매칭됨

ports:

- port: 8200

targetPort: 8000

externalIPs:

- 192.168.0.30

**Kubernetes Overview**

서버 한 대 – master (쿠버네티스 전반적인 기능들을 컨트롤)

다른 서버 – node(자원 제공 역할), 한 마스터에 여러 노드들 연결 -> **하나의 쿠버네티스 클러스터**

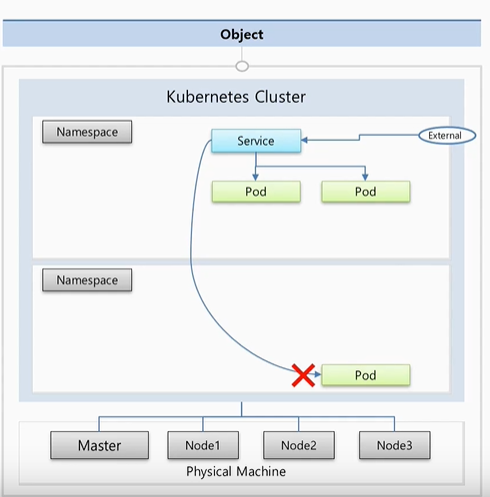
클러스트 전체 자원 증가 -> 노드 추가

Namespace – 쿠버네티스 오브젝트들을 독립된 공f간으로 분리해줌 (-> pod, service)

->쿠버네티스 최소 배포 단위인 **pod**들과

-> pod들이 외부와 연결이 가능하도록 연결이 가능하게 하는 **service** 가 들어있음

Service : 서로 다른 namespace의 pod에는 연결 불가



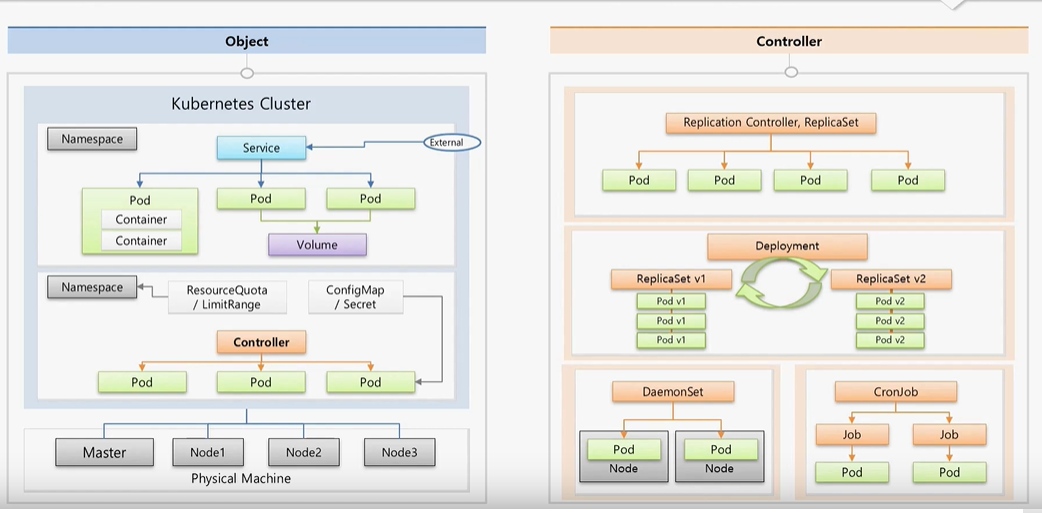
Pod – 여러 컨테이너 (한 컨테이너 – 한 앱) , 여러 앱들이 돌아감.

Pod에 volume을 만들어 하드에 연결하여 데이터 별도 보관 (pod 재생성시 데이터 초기화 방지)

**‘ResourceQuota / LimitRange’** – 한 namespace 에서 사용할 수 있는 자원의 양 한정, 파드 개수, 메모리 cpu 등 제한 가능

**configMap / Secret** – 파드 생성시 컨테이너 안에 환경변수 값을 넣거나 파일 마운팅 -> 세팅가능하게 함

**controller** – 파드들 관리, 종류가 많음

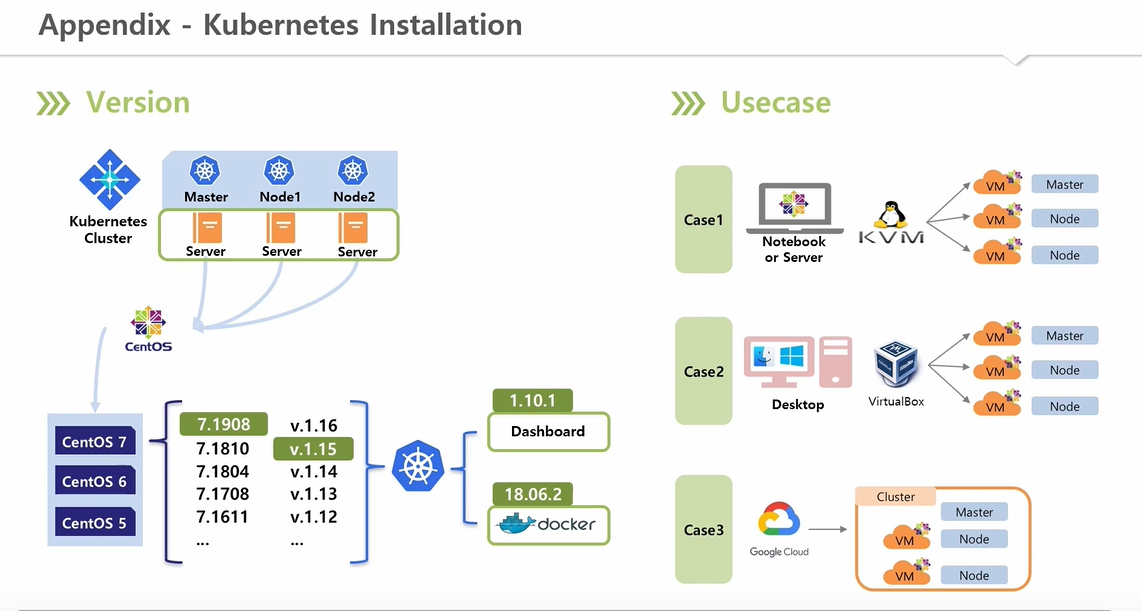


기본 controller 설명

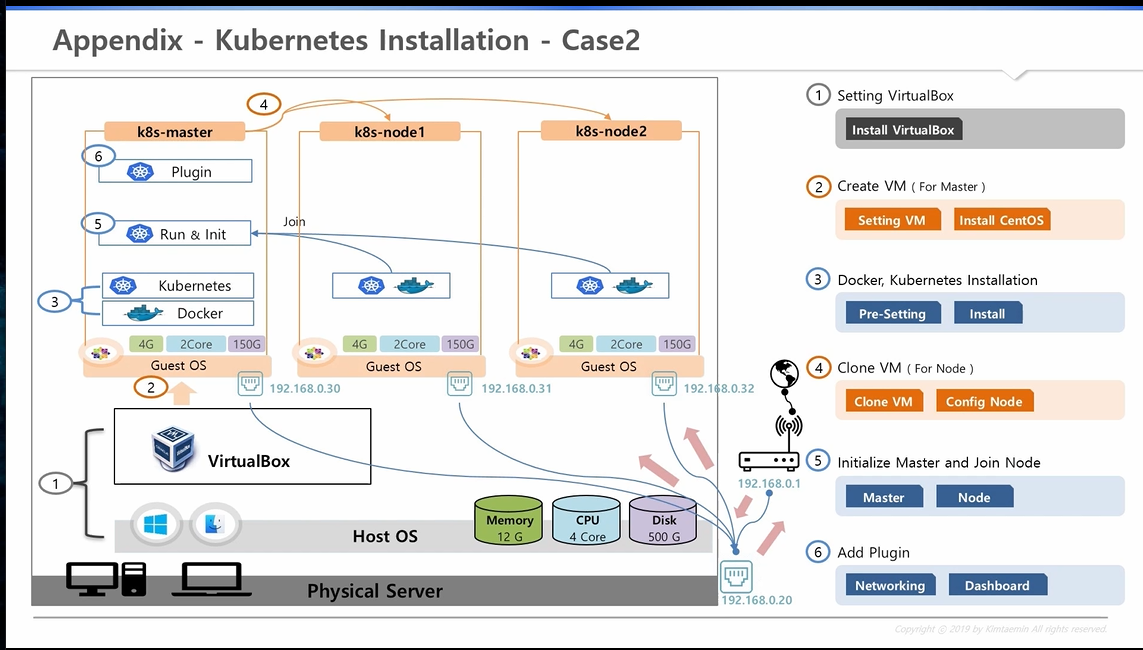
|  |  |
| --- | --- |
| **Replication controller, replicaset** | 파드 죽을 시 감지, 파드 수 조절 가능 |
| **Deployment** | 배포후 파드들 새 버전 업그레이트, 롤백 가능 |
| **DaemonSet** | 한 노드에 한 파드만 있도록 유지하게 함 |
| **Job** | 한 특정작업만 하고 종료시키게 파드가 작동하게 함 |
| **Cronjob** | 위의 job들을 주기적으로 실행시킴 |

**섹션2 – [기초편] 쿠버네티스 설치**

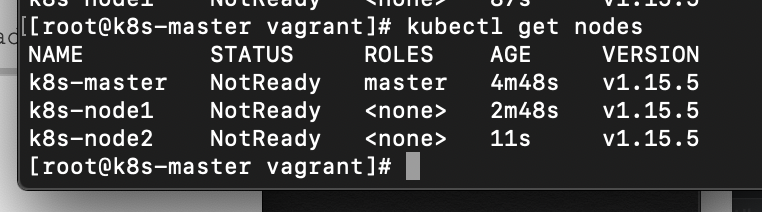
**Kuberentes 설치 개요**

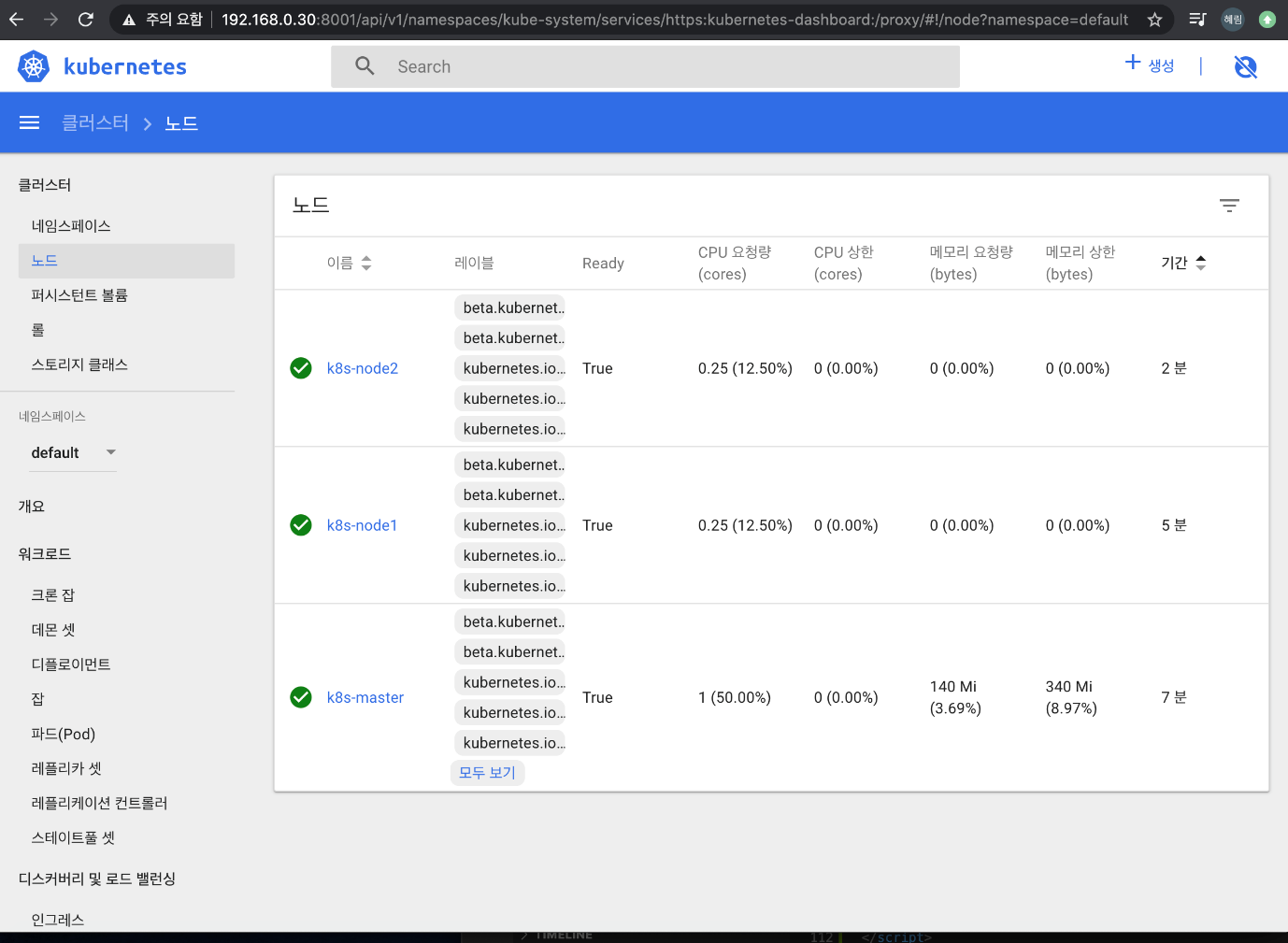


**설치 – case2 [내PC + VirtualBox] 강의환경**



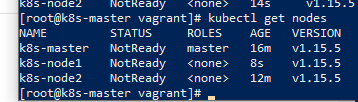
Vagrant 설치 파일 – 원종님 도움

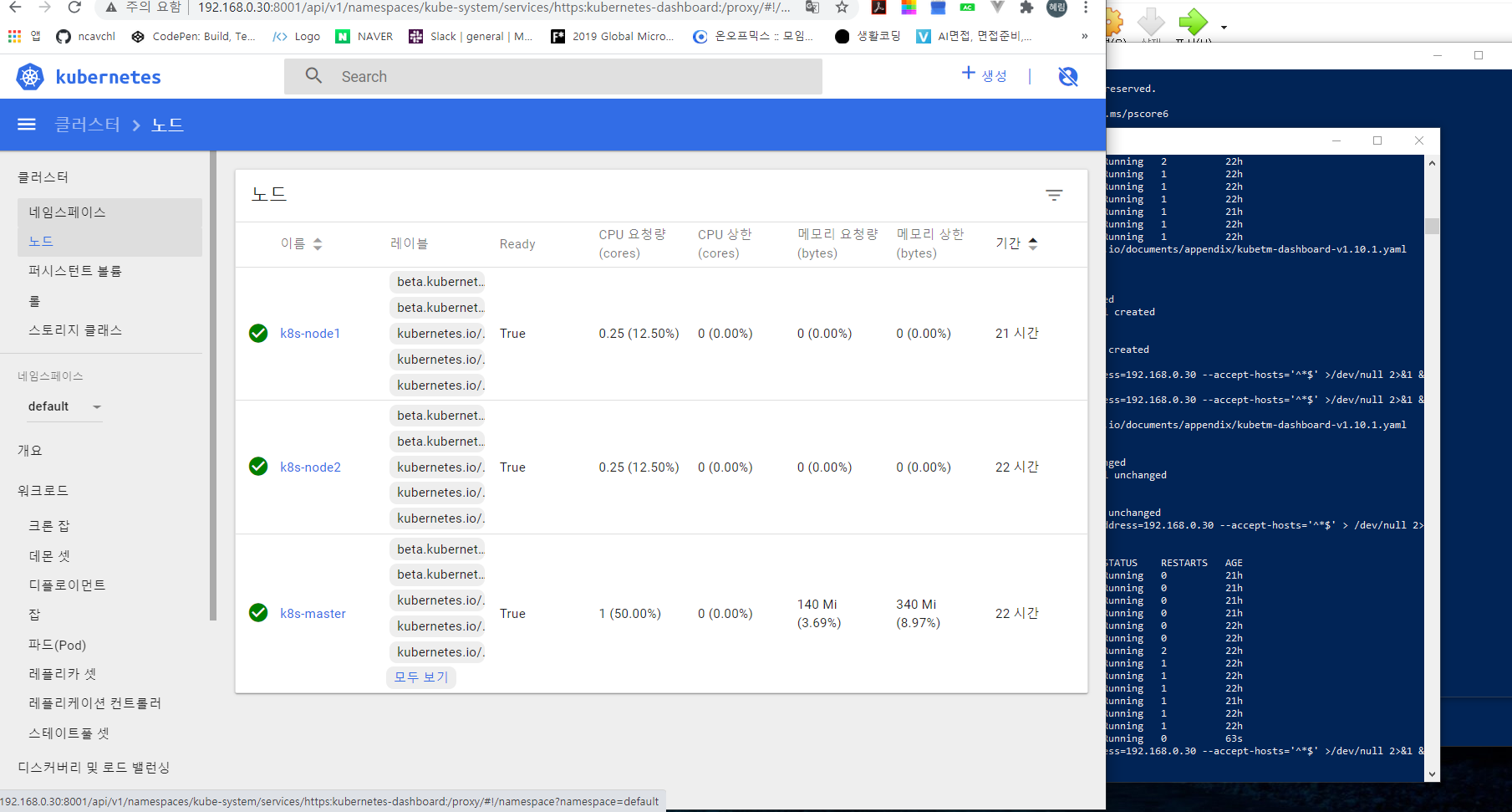




https://kubetm.github.io/practice/appendix/installation\_case2/

하.. 맥에선 성공…

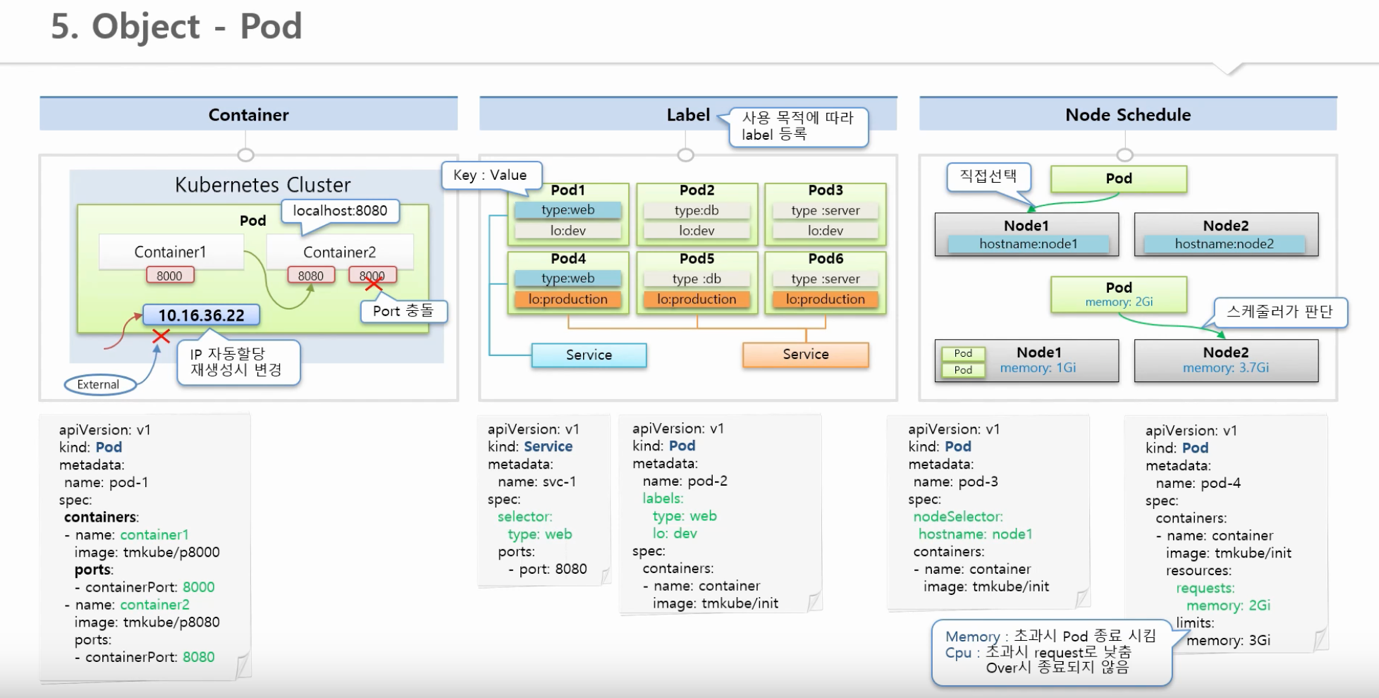




윈도우도 성공 데헷

**섹션3 – [기초편] 기본 오브젝트**

**Pod – Container, Label, NodeSchedule**



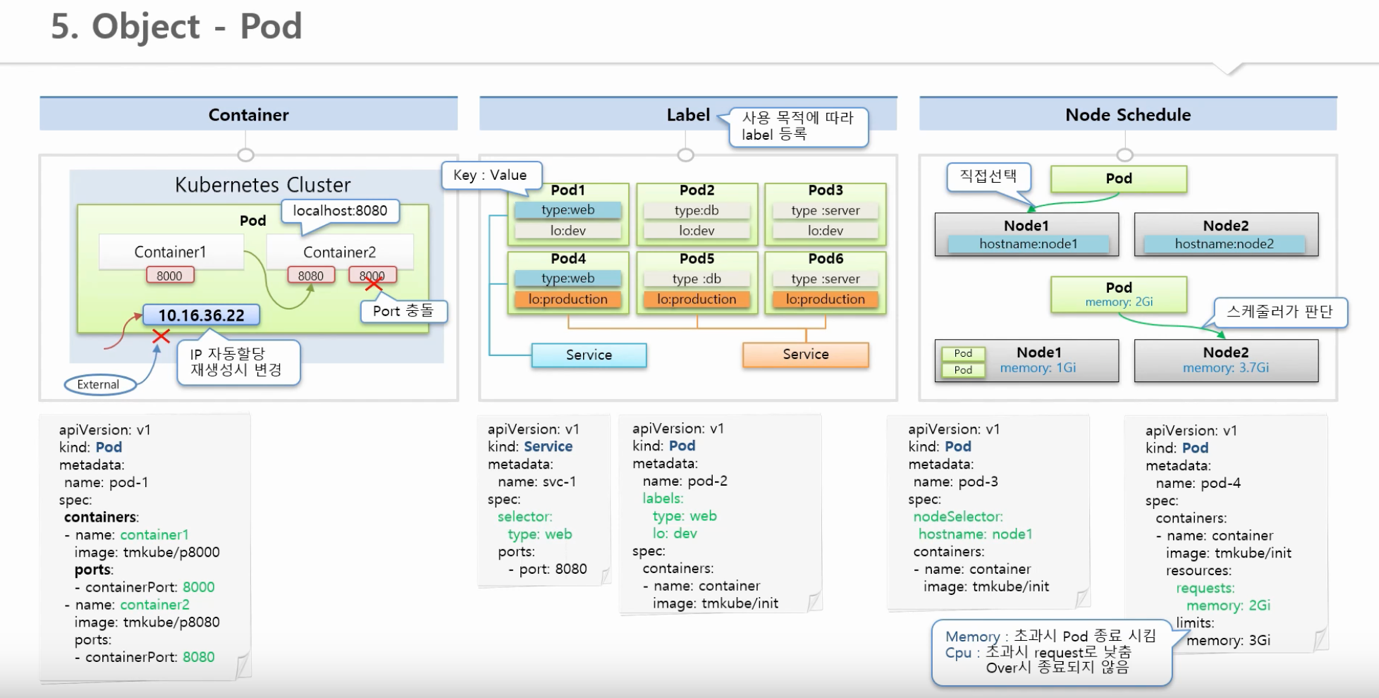
Pod안에는 하나의 독립서비스 구동 가능한 (container) 들이 있음

Container -> 서비스 연결을 위한 port를 가짐

* 한 pod내에서 컨테이너들끼리 port 중복 안됨
* 한 host 로 묶여있음 / 서로 로컬호스트포트로 접근 가능

Pod 생성시 고유의 IP 주소 할당됨 – 쿠버네티스 클러스터 내에서만 ip 통해 pod 로 접근 가능(외부 접근 불가)

Pod 문제시 시스템이 삭제 후 재생성 (ip 재생성 – 휘발성)



LABEL

-pod 뿐만 아니라 모든 Object 에 달 수 있다 / pod에서 가장 많이 사용

-목적에 따라 object 분류, 분류된 오브젝트들만 따로 골라 연결하기 위함

-구성 : key : value / 한 pod 에 여러 라벨 달 수 있음

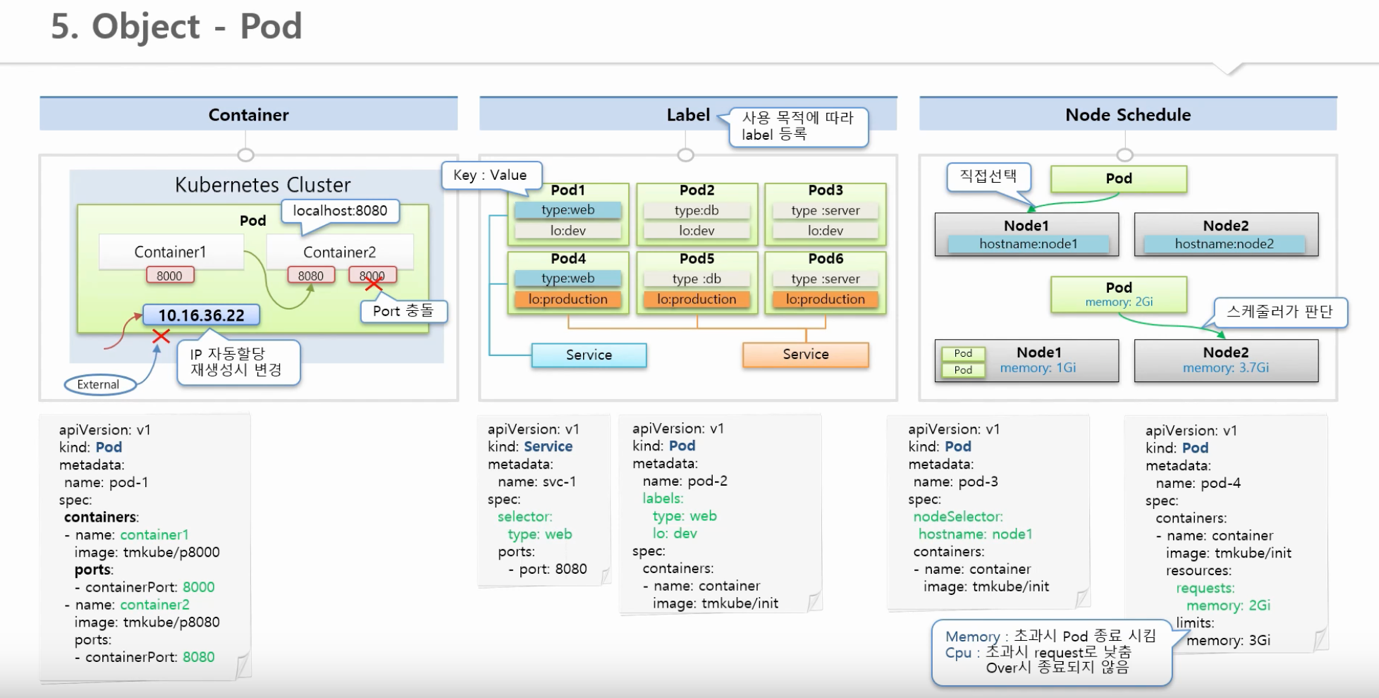
Ex) web이라는 pod 2개, 하나는 dev, 하나는 production(상용)에 있음

Ex)웹 화면만 보고 싶을 때, 타입이 web인 라벨이 달린 pod들을 서비스에 연결 후 서비스 정보를 웹개발자에게 알려줌

Ex)상용환경 담당자에게라면, lo:production 라벨 달린 파드들을 서비스에 연결해서, 서비스 정보를 운영자에게 알려주어 파드에 접속하게 함.

-원하는 파드 선택 후 태그처럼 사용 가능

Service -> selector에 라벨 넣으면 그 라벨에 맞는 pod들이 연결됨



Node Schedule

-pod는 여러 노드들 중 한 노드에 올라가야함 (직접 선택, 쿠버네티스 자동 지정)

(1) 직접 선택 – node에 라벨을 달아 파드와 연결

(2)쿠버네티스 스케줄러가 판단 후 지정 – 노드의 사용가능 자원량(메모리)

Ex)노드1은 남은 용량 1기가 / 노드2는 남은 용량 3.7기가 -> 파드의 2기가 감당 가능한 노드2로 매칭 // 자원 할당을 위해 스케줄러 판단 필요

메모리-cpu : 다르게 동작 , 각 자원의 특성 탓 /

Cpu - 동시 복사 시 첫번째 파일이 느려지면서 복사됨 //

메모리 – 메모리 침범시 프로세스간 문제 일으킴