Kubernetes

Kubernetes

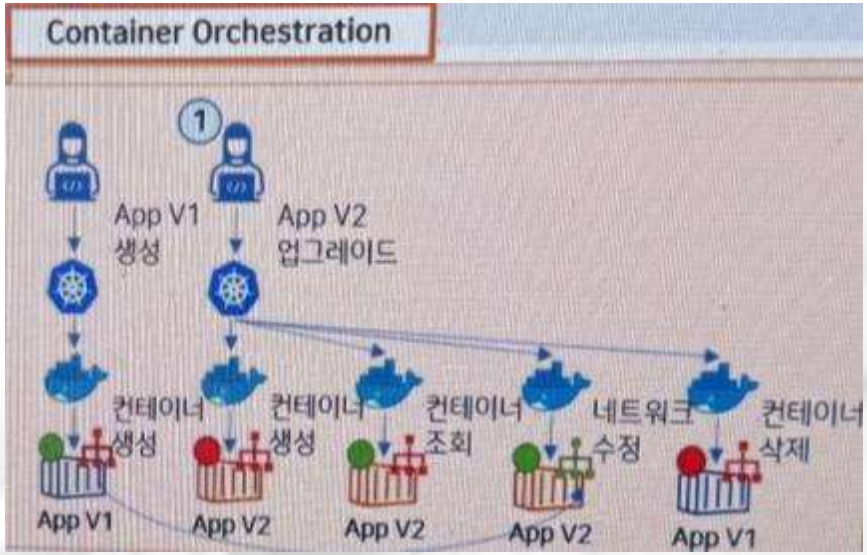
- ❖ Container 런타임
  - ✓ 개요
    - 일반적인 가상화 환경은 하드웨어 수준에서 가상화되지만 Container는 운영체제 수준에서 가상화를 수행
    - Container는 운영체제의 커널을 공유하기 때문에 상대적으로 가볍고 유연하게 운영 가능
    - Container 화된 애플리케이션은 몇 초 내로 빠르게 실행되며 가상머신과 비교했을 때 자원을 더 적게 사용해서 하나의 시스템에서 더 많은 애플리케이션을 구동할 수 있음
    - 운영체제를 공유해서 사용하기 때문에 패치, 업데이트 등 유지관리와 관련한 오버헤드가 줄어든다는 장점이 있음
    - 빠르게 변화하는 비즈니스 환경에서는 Container가 최적의 솔루션이라고 할 수 있음
    - 런타임은 프로그래밍 언어가 구동되는 환경을 의미하는데 자바스크립트가 브라우저에서 실행되면 런타임은 브라우저
    - Container를 생성하고 실행할 수 있도록 도와주는 것이 바로 Container Runtime

Kubernetes

- ❖ Container 런타임
  - ✓ 종류
    - Containerd
      - ◆ 컨테이너 실행을 관리하는 오픈 소스 런타임
      - ◆ Docker 및 Kubernetes와 같은 컨테이너 오케스트레이션 시스템에서 컨테이너 실행을 처리하는 데 사용
      - ◆ Container 이미지를 다운로드 받아 압축을 푸는 과정부터 Container를 실행하고 감독하는 과정까지 Container의 전체 수명 주기를 관리하는 기능을 제공
    - Docker
      - ◆ Containerd 위에 설치되는 데몬
      - ◆ Container를 생성하고 관리하는데 필요한 모든 기능을 제공
      - ◆ Docker는 컨테이너의 전체 라이프사이클(빌드, 배포, 실행)을 관리하는 반면 Containerd는 컨테이너 실행과 이미지 관리를 담당하는 핵심 컴포넌트로 Docker는 내부적으로 Containerd를 사용하며 Kubernetes도 Containerd를 기본 런타임으로 지원
    - CRI-O
      - ◆ Redhat, Intel, SUSE, Hyper, IBM 등의 관리자 및 커뮤니티를 중심으로 개발된 오픈 소스 런타임
      - ◆ CRI-O는 Docker를 대체하기 위한 툴로 개발

Kubernetes

- ❖ Container Orchestration
  - ✓ 개요
    - 다수의 Container를 유기적으로 연결 및 실행할 뿐만 아니라 상태를 추적하고 보존하는 등 Container를 안정적으로 사용할 수 있게 만들어주는 솔루션
    - Container를 효과적으로 관리하도록 도와주는 것이 Container 오케스트레이션
    - Container 오케스트레이션은 여러 시스템에 Container를 분산해서 배치하거나 문제가 생긴 Container를 교체하는 등 여러 역할을 수행



Kubernetes

- Container Orchestration
  - ✓ Container 오케스트레이션 솔루션

- Docker Swarm
  - 간단하게 설치할 수 있고 사용하기도 용이
  - 기능이 다양하지 않아 대규모 환경에 적용하려면 사용자 환경을 변경해야 할 수 있기 때문에 소규모 환경에서는 유용하지만 대규모 환경에서는 잘 사용하지 않는 편
- Mesos
  - Apache의 오픈 소스 프로젝트로 트위터, AirBnB, 애플, Uber 등 다양한 곳에서 이미 검증된 솔루션
  - 2016년 DC/OS(Data Center OS, 대규모 서버 환경에서 자원을 유연하게 공유하며 하나의 자원처럼 관리하는 도구)의 지원으로 매우 간결하지만 기능을 충분히 활용하려면 분산 관리 시스템과 연동해야 하기 때문에 여러가지 솔루션을 유기적으로 구성해야 하는 부담이 있음

Kubernetes

- Container Orchestration
  - ✓ 대표적인 Container 오케스트레이션 솔루션
    - Nomad
      - Vagrant를 만든 HashiCorp 사의 Container 오케스트레이션으로 Vagrant처럼 간단한 구성으로 Container 오케스트레이션 환경을 제공
      - HashiCorp의 Consul(서비스 검색, 구성 및 분할 기능 제공) 과 Vault(암호화 저장소) 와 연동이 원할하므로 이런 도구에 대한 사용 성숙도가 높은 조직이라면 노매드 도입을 고려해볼 수 있음
    - Kubernetes
      - 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 운영하는 오픈 소스 오케스트레이션 플랫폼
      - 구글이 내부적으로 사용하던 Borg 시스템에서 발전한 기술로 현재는 Cloud Native Computing Foundation(CNCF)에서 관리

Kubernetes

- Container Orchestration
  - ✓ 대표적인 Container 오케스트레이션 솔루션

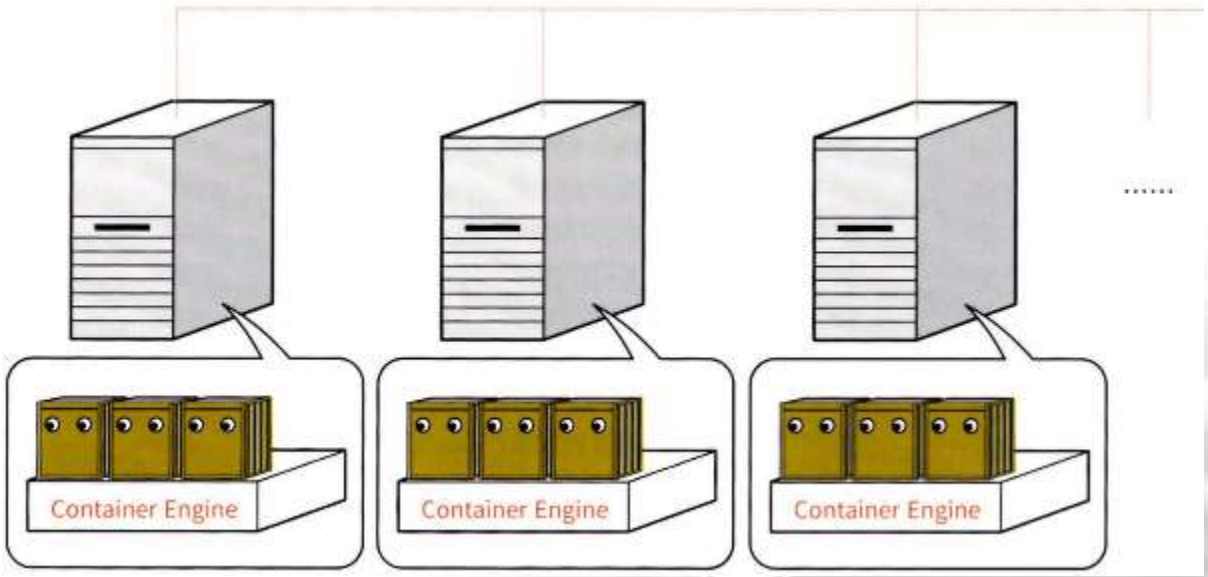
구분	도커 스웜	메소스	노매드	쿠버네티스
설치 난이도	쉬움	매우 어려움	쉬움	어려움
사용 편의성	매우 좋음	좋음	매우 좋음	좋음
세부 설정 지원	거의 없음	있음	거의 없음	다양하게 있음
안정성	매우 안정적임	안정적임	안정적임	매우 안정적임
확장성	어려움	매우 잘 됨	어려움	매우 잘 됨
정보량	많음	적음	적음	매우 많음
에코 파트너	없음	거의 없음	있음	매우 많음
학습 곡선	쉬움	매우 어려움	어려움	어려움

Kubernetes

- Kubernetes 개요
  - ✓ Kubernetes는 Container 기반의 애플리케이션을 개발하고 배포할 수 있도록 설계된 오픈 소스 플랫폼으로 Container 오케스트레이션 도구의 일종
  - ✓ Kubernetes는 k8s 라고도 하는데 k 와 s 사이에 8개의 글자가 있다는 의미의 약칭으로 Kubernetes와 관련된 검색어로 유용
  - ✓ Kubernetes는 일반적인 프로그래머가 관리하는 일은 거의 없음
    - 대규모 시스템을 개발하는 프로그래머가 대규모 서버 군을 직접 관리할 일이 드물듯이 Kubernetes를 직접 관리하는 일은 드뭄
    - Kubernetes로 어떤 일을 할 수 있는가에 대한 지식은 시스템을 개발할 때 유용할 수 있는데 Kubernetes로 관리되는 시스템은 이를 전제로 개발되어야 하는데 그렇지 않다면 Kubernetes의 이점을 제대로 살릴 수 없기 때문에 프로젝트 매니저나 시스템 엔지니어를 담당하는 사람도 Kubernetes로 어떤 일을 할 수 있는가는 잘 이해해야 함

Kubernetes

- ❖ Kubernetes 개요
  - ✓ Kubernetes는 여러 대의 물리적 서버에 걸쳐 실행되는 것을 전제로 함
    - Kubernetes는 여러 대의 물리적 서버가 존재하는 것을 전제
    - 물리적 서버 한 대 한 대마다 제각기 여러 개의 Container를 실행한다고 가정



Kubernetes

- Kubernetes 개요
  - ✓ Kubernetes는 여러 대의 물리적 서버에 걸쳐 실행되는 것을 전제로 함
    - 여러 대의 서버에서 일일이 Container를 실행하고 관리하기는 쉬운 일이 아닌데 Kubernetes는 이를 위한 도구
    - Docker 만을 이용해서 20개의 Container를 만들려면 docker run 커맨드를 20번 실행해야 할 것인데 이는 매우 지겹고 지루한 작업
    - Docker Compose를 사용한다 해도 물리적 서버가 여러 대라면 반복 작업은 사라지지 않고 어떻게든 Container를 생성해 실행했다 해도 물리적 서버를 일일이 모니터링하면서 장애가 일어나면 Container를 다시 실행해야 하고 Container를 업데이트하는 경우 큰 수고가 따름
    - Kubernetes는 번거로운 Container 생성이나 관리의 수고를 덜어주는 도구로 Docker Compose에서 사용되는 Compose 파일과 비슷한 정의 파일만 작성하면 이 정의에 따라 모든 물리적 서버에 Container를 생성하고 생성한 Container를 관리해 줌

Kubernetes

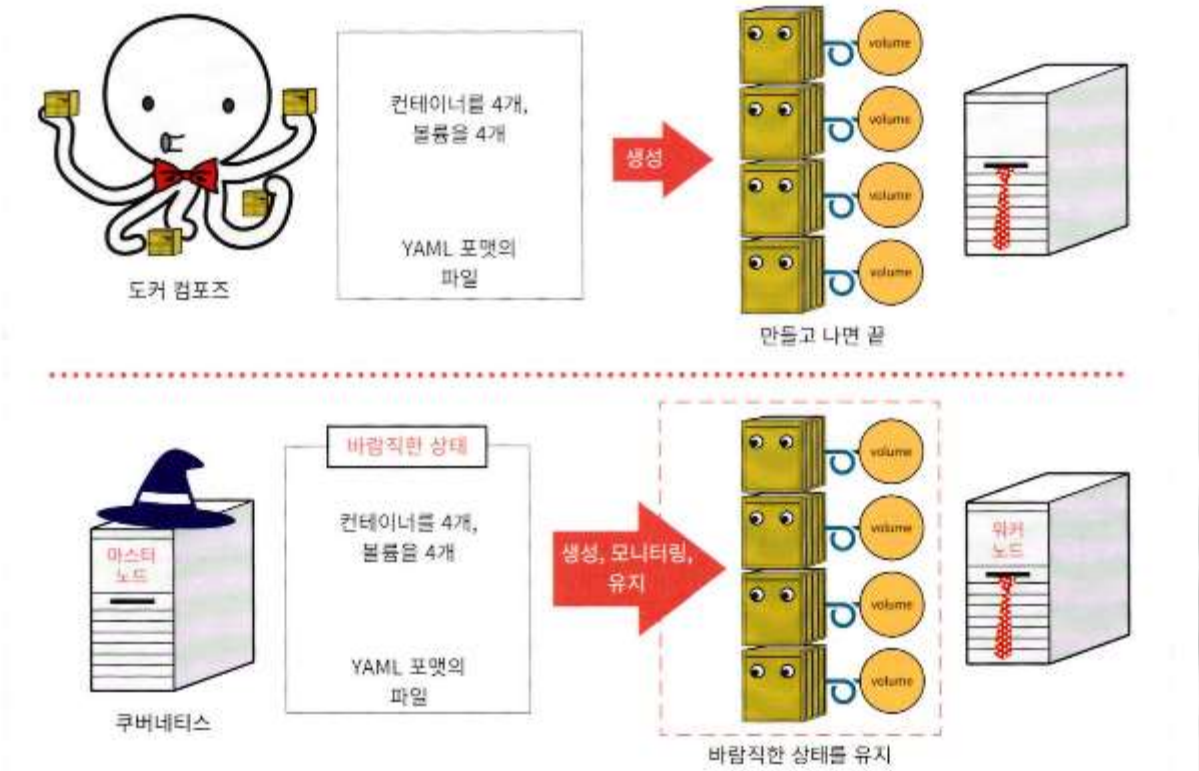
- Kubernetes를 사용하는 이유
  - ✓ 자동화된 컨테이너 오케스트레이션
    - 컨테이너의 배포, 관리, 확장, 복구를 자동으로 수행
    - 여러 개의 컨테이너를 그룹화하여 하나의 애플리케이션처럼 운영
  - ✓ 무중단 서비스 – Rolling Update
    - 새 버전의 Pod를 점진적으로 생성
    - 새 Pod가 준비되면, 구 버전의 Pod를 점진적으로 종료
    - 이 과정을 통해 전체 Pod를 한 번에 교체하는 것이 아니라 일부 인스턴스 별로 순차적으로 교체하여 서비스의 가용성을 유지
  - ✓ 효율적인 자원 사용
    - Kubernetes는 Pod가 사용할 수 있는 자원(CPU, 메모리 등)을 사전에 지정할 수 있어서 시스템(노드)의 전체 자원을 관리할 수 있기 때문에 자원을 효율적으로 사용할 수 있음
    - Traffic Routing: 트래픽은 주로 Service와 Ingress를 통해 원하는 Pod로 전달
  - ✓ 유연한 확장성
    - Auto-scaling: CPU/메모리 사용량을 모니터링하여 자동으로 확장 또는 축소하는 것
    - Kubernetes는 Pod의 자원 사용률에 따라 Pod의 개수를 늘리거나 줄일 수 있기 때문에 CPU 사용률이 200%로 증가하는 경우 Pod를 1개에서 5개로 늘리고 CPU 사용률이 감소하면 Pod를 다시 1개로 줄일 수 있음

Kubernetes

- Kubernetes를 사용하는 이유
  - ✓ 항상 바람직한 상태를 유지
    - Kubernetes도 Container를 생성하거나 삭제할 수 있지만 일일이 명령어를 입력하는 방식을 사용하지는 않음
    - Container는 OO개 볼륨을 XX개로 구성하라 와 같이 어떤 바람직한 상태를 YAML 파일에 정의하면 이를 읽어서 자동으로 Container를 생성하거나 삭제하면서 이 상태를 만들고 유지하는 것이 Kubernetes의 기본적인 아이디어
    - Docker Compose는 옵션을 지정해 수동으로 Container의 수를 바꿀 수는 있어도 모니터링 기능이 없어서 Container를 만들 때 외에는 관여하지 않지만 이에 비해 Kubernetes에는 이 상태를 유지하는 기능이 있음

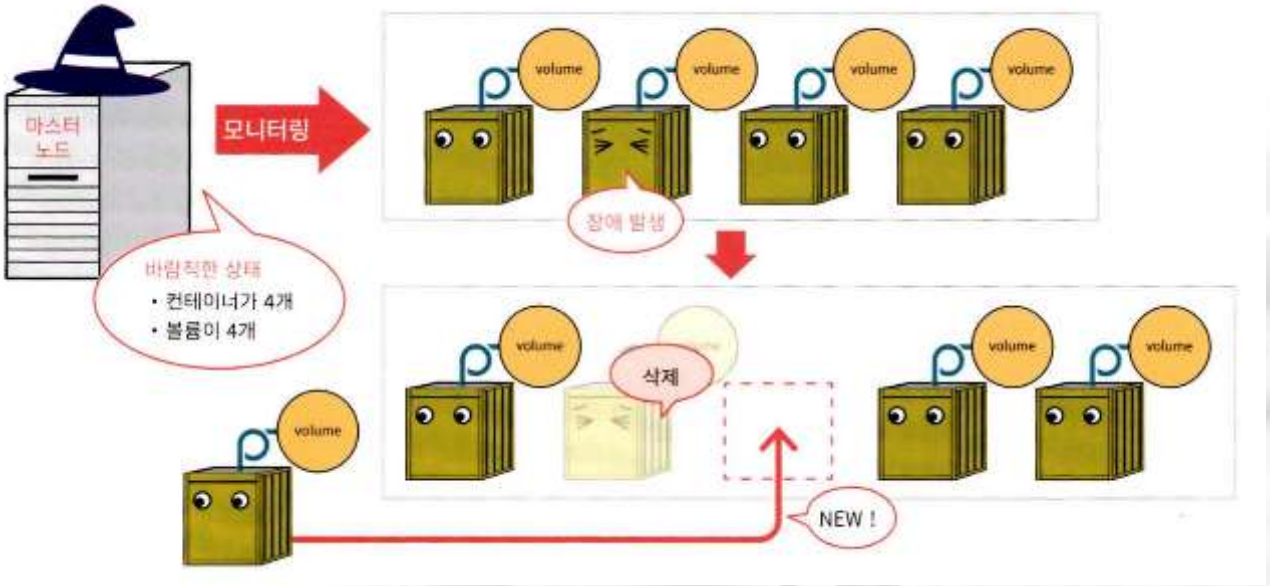
Kubernetes

- Kubernetes를 사용하는 이유
  - ✓ 항상 바람직한 상태를 유지



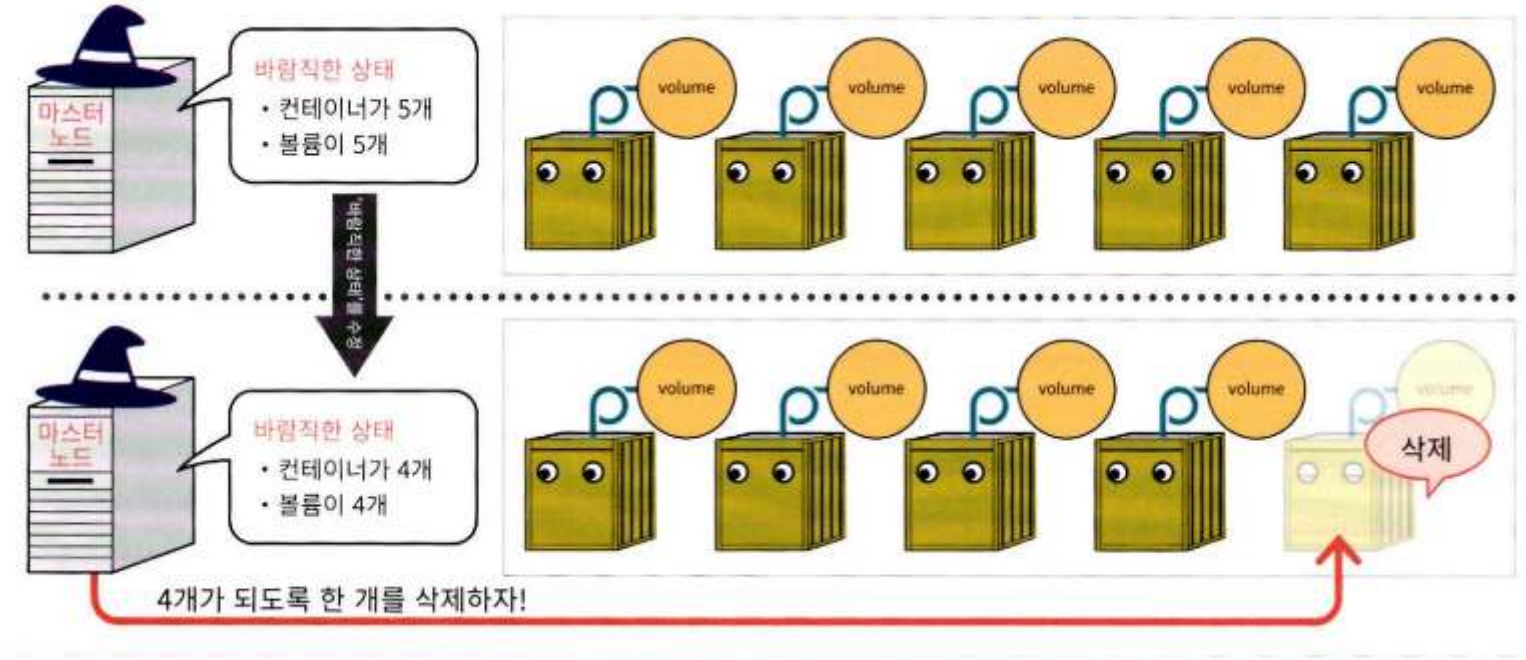
Kubernetes

- Kubernetes를 사용하는 이유
  - ✓ 항상 바람직한 상태를 유지
    - Container 5개를 Container 4개로 수정하면 Container를 한 개 삭제
    - Self-Healing
      - 장애가 발생한 컨테이너를 자동으로 다시 시작
      - 노드(Node) 장애 시 애플리케이션을 다른 노드로 이동



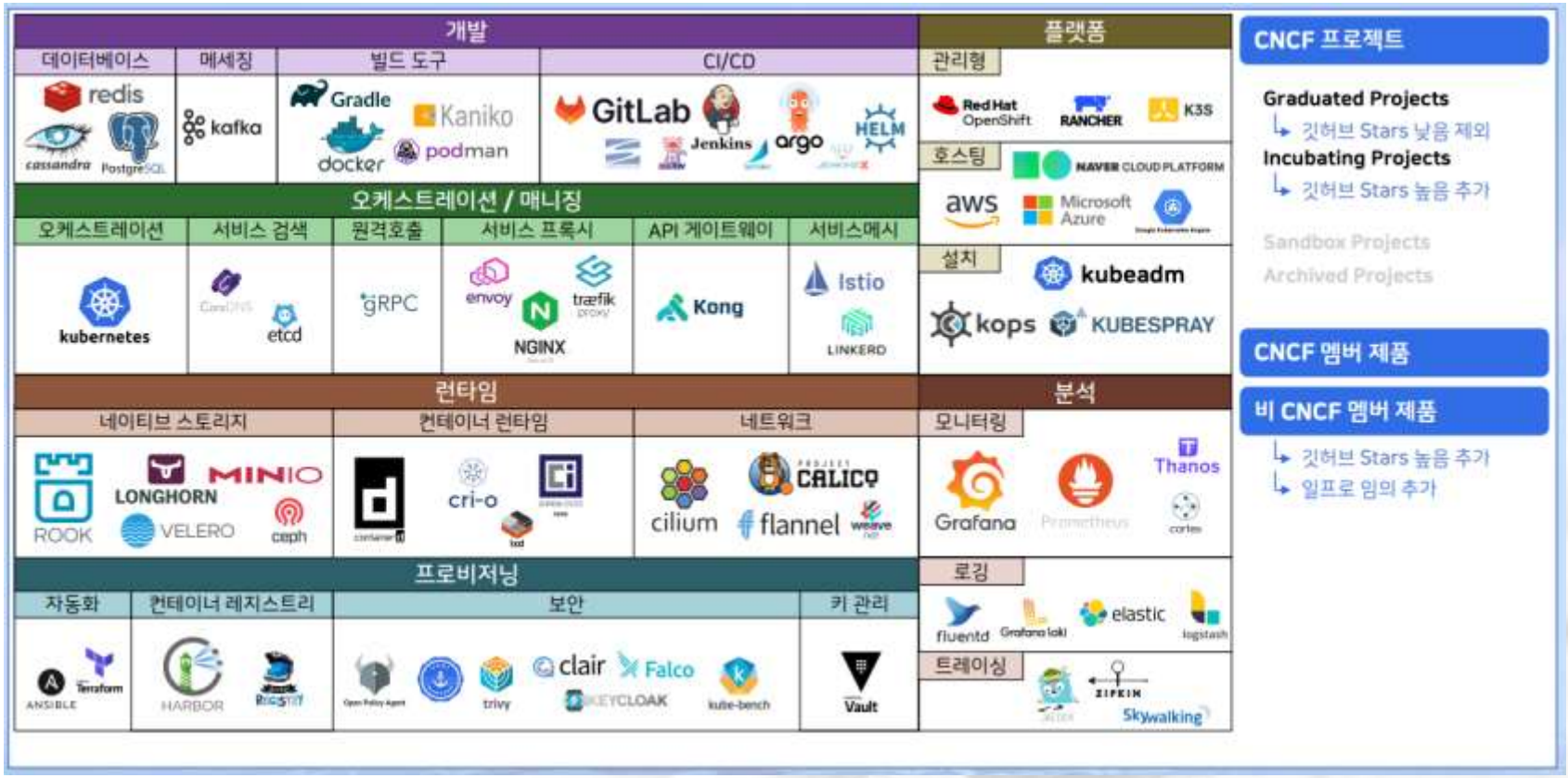
Kubernetes

- Kubernetes를 사용하는 이유
  - ✓ 항상 바람직한 상태를 유지



Kubernetes

- Kubernetes를 사용하는 이유
  - ✓ 쿠버네티스 표준 생태계로 편해진 IT 인프라 구축: 클라우드 벤더 종속성 해결

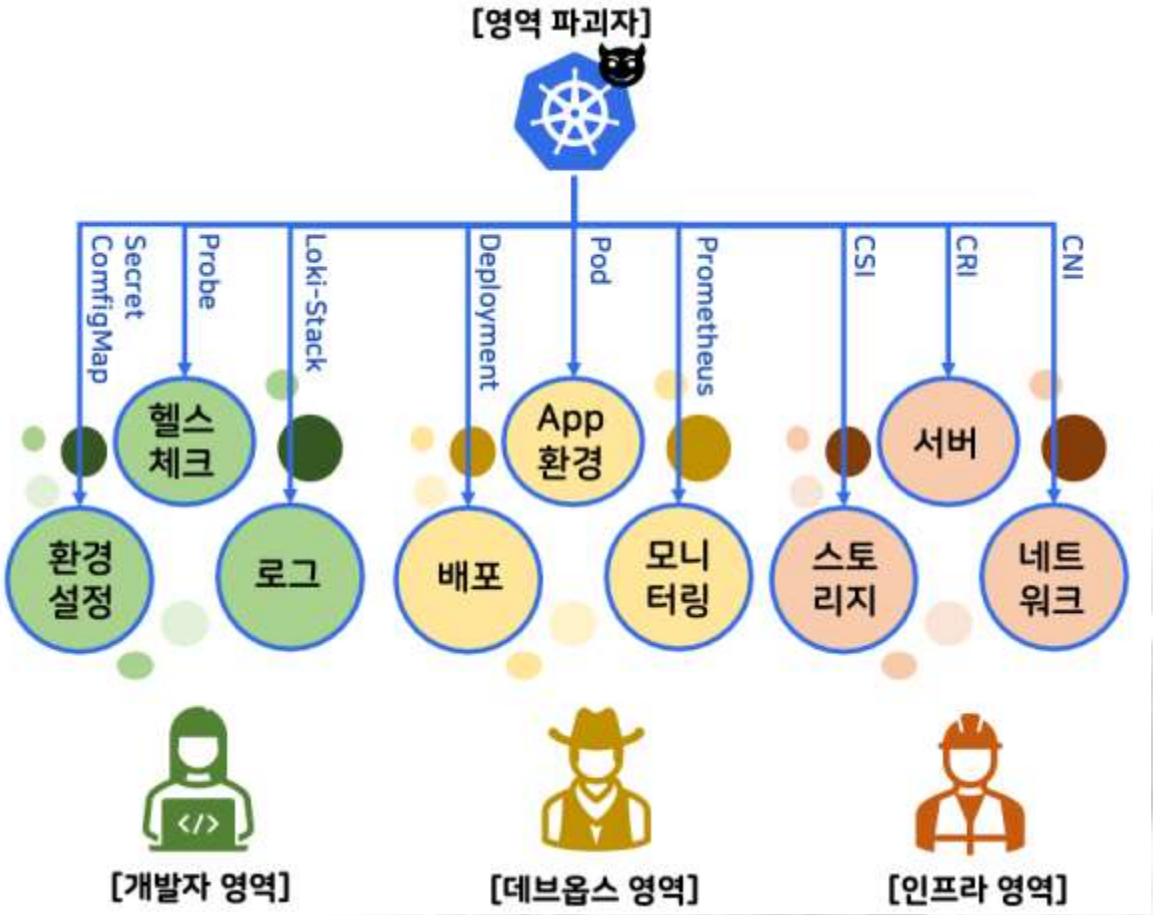


Kubernetes

- Kubernetes를 사용하는 이유
  - ✓ 실제 프로젝트를 할 때 구조적인 문제
    - 개발과 모니터링 시스템이 서로 엮일 수 밖에 없는 구조
    - 개발에서는 한번도 써보지 않은 개발 시스템을 위한 모니터링 시스템을 만드는 문제
    - 오픈 시 개발 프로젝트와 서로 다른 범위의 App들을 모니터링 하게 되는 구조

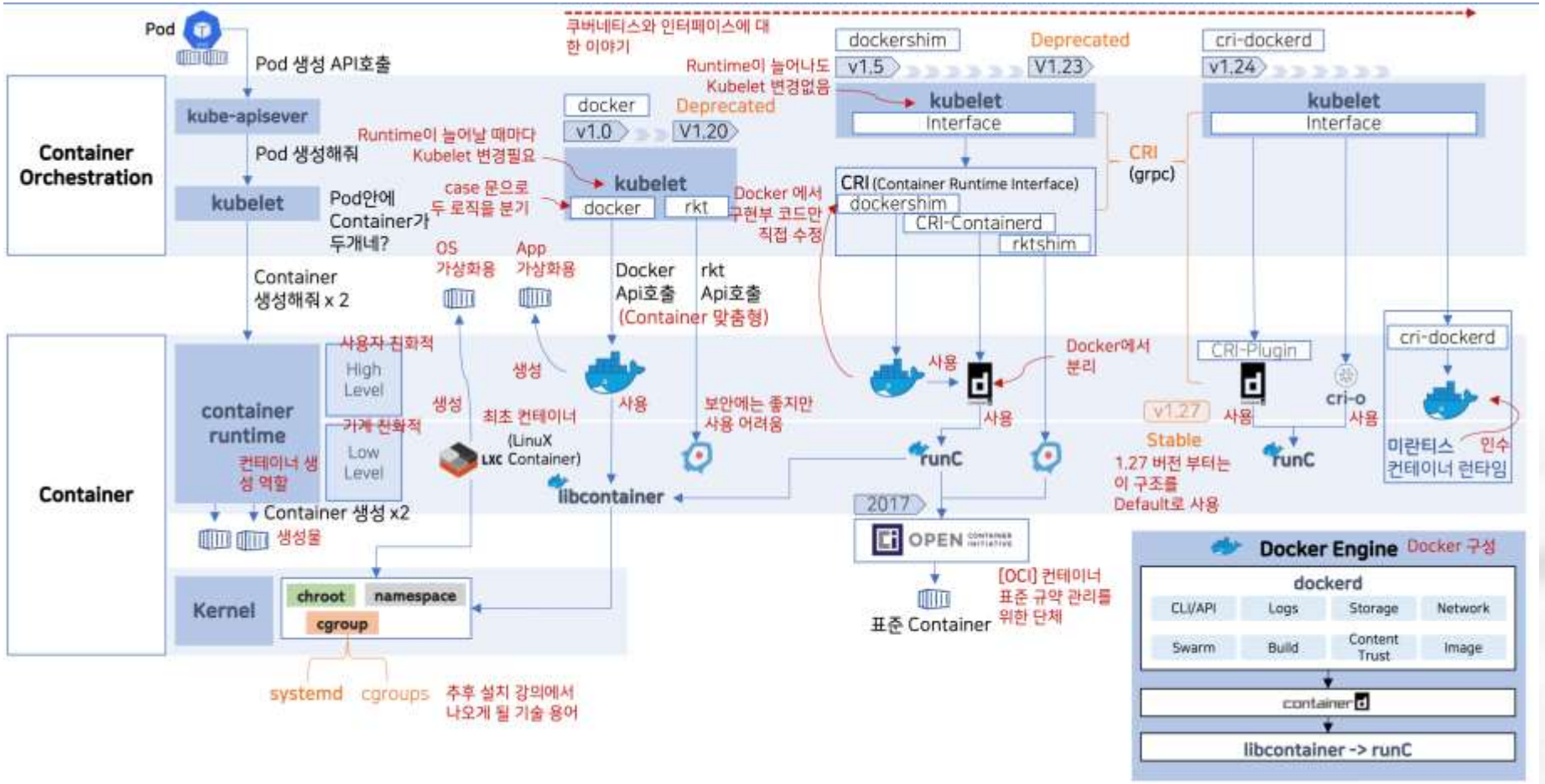
Kubernetes

❖ 영역 파괴자



Kubernetes

- Kubernetes 특징
  - ✓ 기술의 흐름으로 이해하는 컨테이너



Kubernetes

- Kubernetes 리소스
  - ✓ 리소스 분류
    - Workload API: 컨테이너 실행에 관련된 리소스
      - Pod
      - ReplicationController
      - ReplicaSet
      - Deployment
      - DaemonSet
      - StatefulSet
      - Job
      - CronJob

Kubernetes

- ❖ Kubernetes 리소스
  - ✓ 리소스 분류
    - Service API: 컨테이너를 외부에 공개하는 End Point를 제공하는 리소스
      - ◆ ClusterIP
      - ◆ ExternalIP
      - ◆ NodePort
      - ◆ LoadBalancer
      - ◆ Headless(None)
      - ◆ ExternalName
      - ◆ None-Selector
      - ◆ Ingress

Kubernetes

- Kubernetes 리소스
  - ✓ 리소스 분류
    - Config & Storage API: 설정/기밀 정보/영구 볼륨 등에 관련된 리소스
      - Secret
      - ConfigMap
      - PersistentVolumeClaim

Kubernetes

- Kubernetes 리소스
  - ✓ 리소스 분류
    - Cluster API: 보안이나 쿼터 등에 관련된 리소스
      - Node
      - Namespace
      - PersistentVolume
      - ResourceQuota
      - ServiceAccount
      - Role
      - ClusterRole
      - RoleBinding
      - ClusterRoleBinding
      - NetworkPolicy

Kubernetes

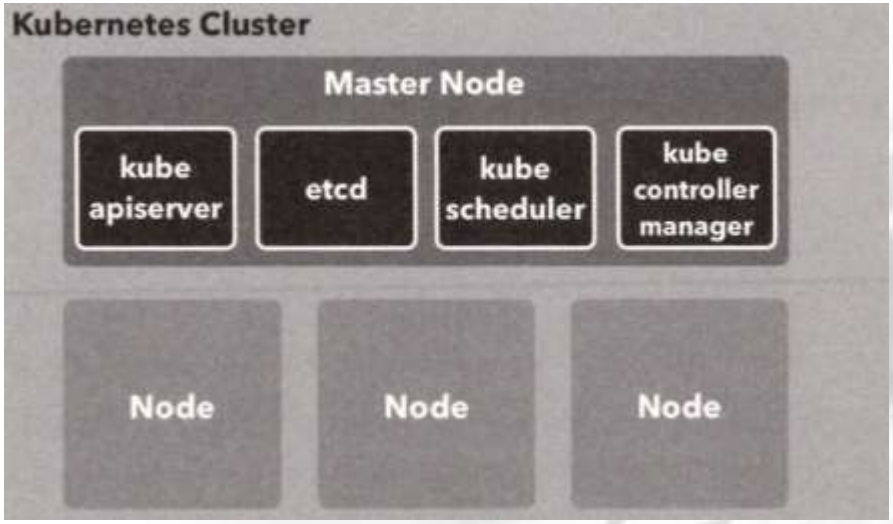
- Kubernetes 리소스
  - ✓ 리소스 분류
    - Meta Data API: 클러스터 내부의 다른 리소스를 관리하기 위한 리소스
      - LimitRange
      - HorizontalPodAutoScaler
      - PodDistruptionBudget
      - CustomResourceDefinition

Kubernetes

- ❖ Kubernetes Architecture
  - ✓ Pod
    - ● 하나의 애플리케이션을 생성하기 위해서는 하나 이상의 Pod가 필요한데 Pod는 Kubernetes에서 생성할 수 있는 가장 작은 배포 단위이면서 단일 혹은 다수의 Container를 포함
    - ● Pod 외에 Service, Volume, Namespace 등을 묶어서 Object라 하고 Object는 Kubernetes에서 상태 관리가 필요한 대상
    - ● Pod 와 Container
      - ◆ Pod는 Kubernetes에서 가장 기본적인 배포 단위이면서 하나 이상의 Container를 포함
      - ◆ Kubernetes의 특징 중 하나는 Container를 개별적으로 하나씩 배포하는 것이 아니라 유사한 역할을 하는 Container를 Pod라는 단위로 묶어서 배포
      - ◆ 쇼핑몰 웹 사이트에는 사용자가 상품을 검색하고 주문을 하는 애플리케이션 과 상품 정보 및 이미지를 저장하는 저장소(데이터베이스)가 있는데 이 둘은 서로 유기적인 관계를 갖기 때문에 애플리케이션과 저장소는 별도의 Container지만 이 둘을 쇼핑몰 웹 사이트 용도의 Pod 하나로 묶어서 배포할 수 있음

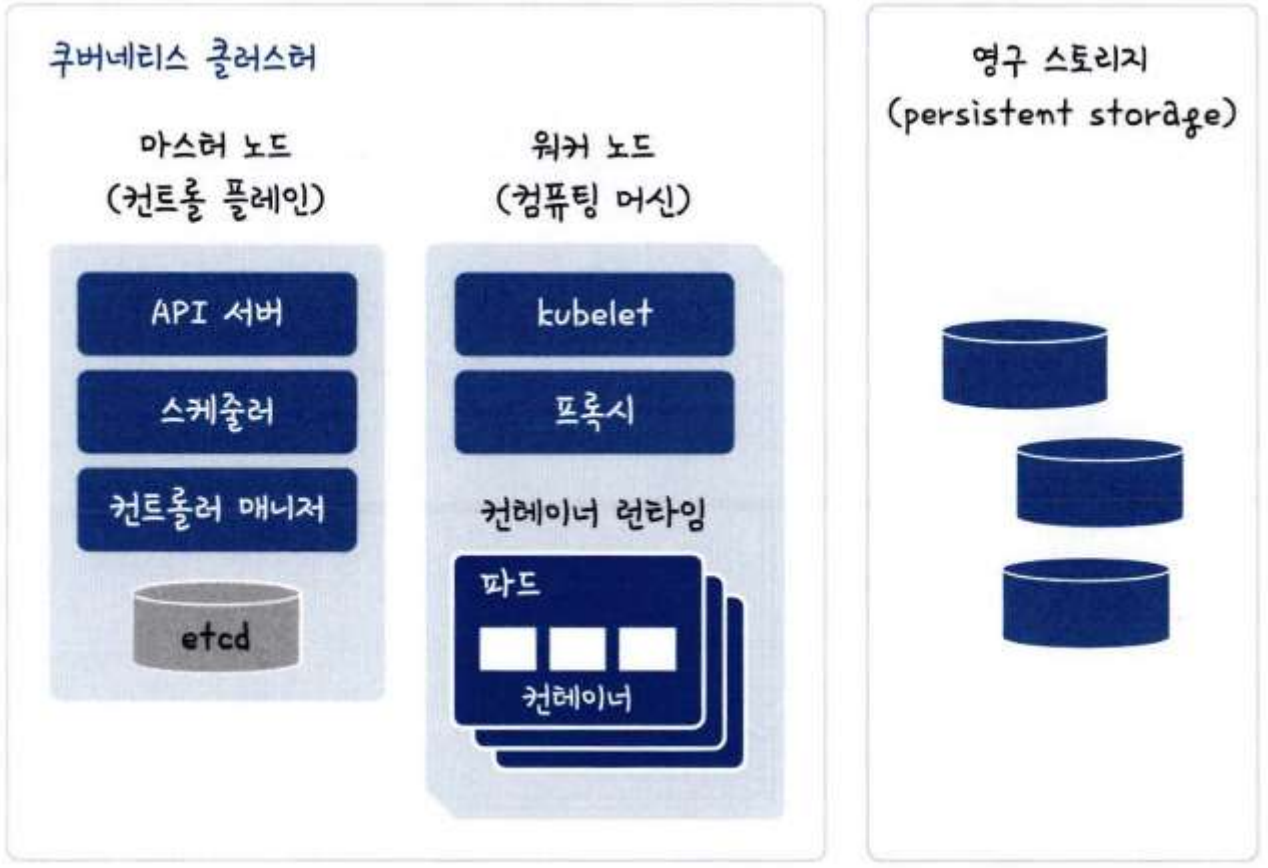
Kubernetes

- ❖ Kubernetes Architecture
  - ✓ Kubernetes 구조
    - Kubernetes Cluster: Kubernetes의 여러 리소스를 관리하기 위한 집합체로 Master Node 와 Worker Node를 이용해 하나의 Kubernetes Cluster를 구성
    - Node
      - ◆ Kubernetes 리소스 중에서 가장 큰 개념은 노드
      - ◆ 노드는 Kubernetes Cluster의 관리 대상으로 등록된 Container가 배치되는 대상
      - ◆ Kubernetes Cluster 전체를 관리하는 서버인 마스터(Control Plane)가 하나 이상 있어야 함
      - ◆ Kubernetes Cluster는 마스터와 노드의 그룹으로 구성



Kubernetes

- Kubernetes Architecture
  - ✓ Kubernetes 구조



Kubernetes

- ❖ Kubernetes Architecture
  - ✓ Kubernetes 구조
    - Master Node
      - ◆ Kubernetes Cluster 전체를 관리하는 시스템으로 Control plane
      - ◆ Master Node를 설정하는 관리자의 컴퓨터에는 kubectl을 설치하는데 kubectl을 설치해야 Master Node에 로그인해 초기 설정을 진행하거나 추후 조정 가능
    - Persistent Storage: Pod는 휘발성이므로 Worker Node에 떠 있는 Pod가 삭제되면 Pod 안의 모든 데이터도 삭제되기 때문에 중요한 데이터는 Pod 외부에 있는 저장소에 저장해 두어야 하며 CSI(Container Storage Interface)로 외부 저장소를 Pod에 연결할 수 있음

Kubernetes

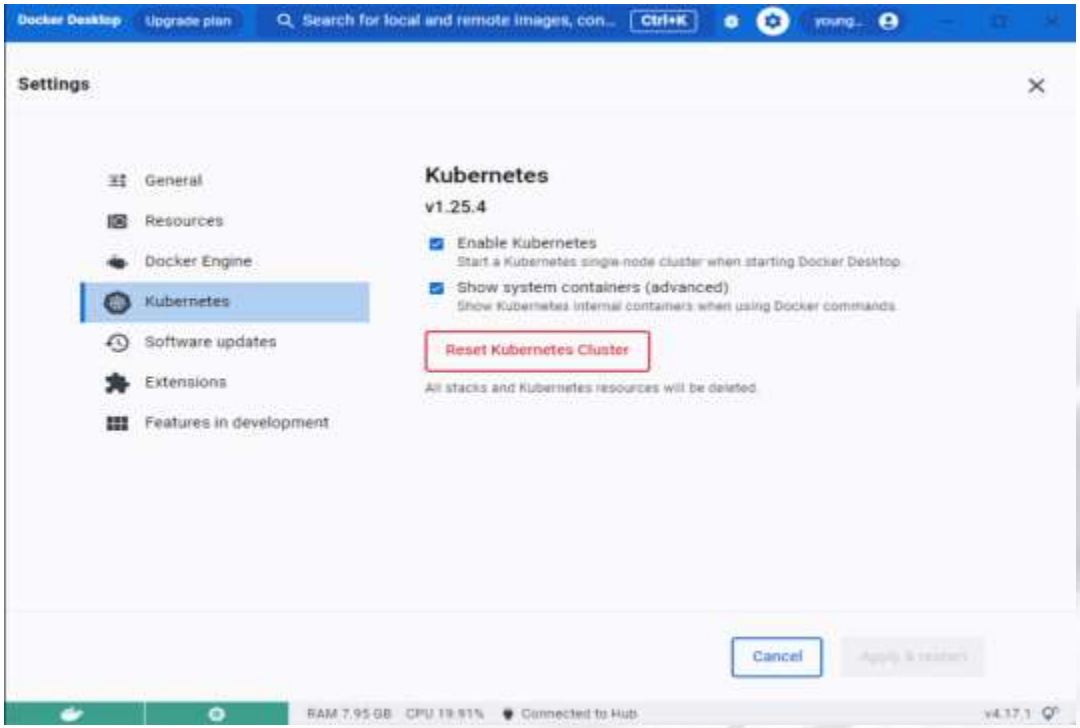
- ❖ Kubernetes Architecture
  - ✓ Kubernetes 구조
    - ● Worker Node
      - ◆ Container Runtime이 설치된 환경으로 Pod가 실제로 생성되고 유지되는 Node
      - ◆ Linux 위에 Container 런타임을 설치하고 그 위에 하나 이상의 Container가 포함된 Pod를 생성하면 각 Container에서는 그 목적에 맞는 서비스가 실행됨

Kubernetes Cluster 구축

- 방법
  - ✓ Local Kubernetes: 물리 머신 1대에 구축해서 사용
  - ✓ Kubernetes 클러스터 구축 도구 이용: Onpremise 나 Cloud 환경에 클러스터를 구축하여 사용
  - ✓ 관리형 Kubernetes: Public Cloud의 관리형 서비스로 제공하는 클러스터를 사용

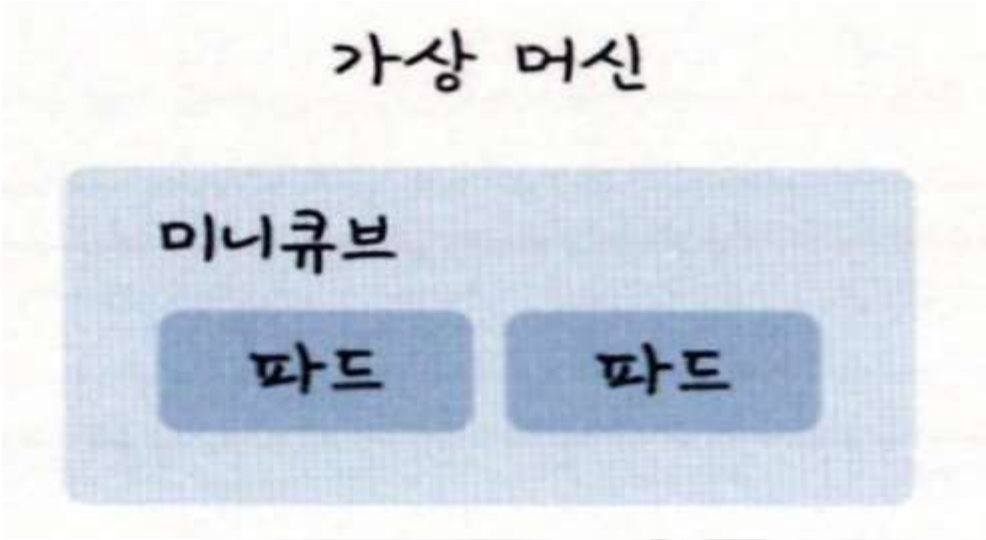
Kubernetes Cluster 구축

- ❖ Local Kubernetes
  - ✓ Docker Desktop 활용
    - Docker 실행 후 톱니바퀴를 눌러주시면 Kubernetes 메뉴가 옆에 있는데 Enable Kubernetes를 체크하고 Apply & Restart를 누르면 Kubernetes 설치가 진행됨



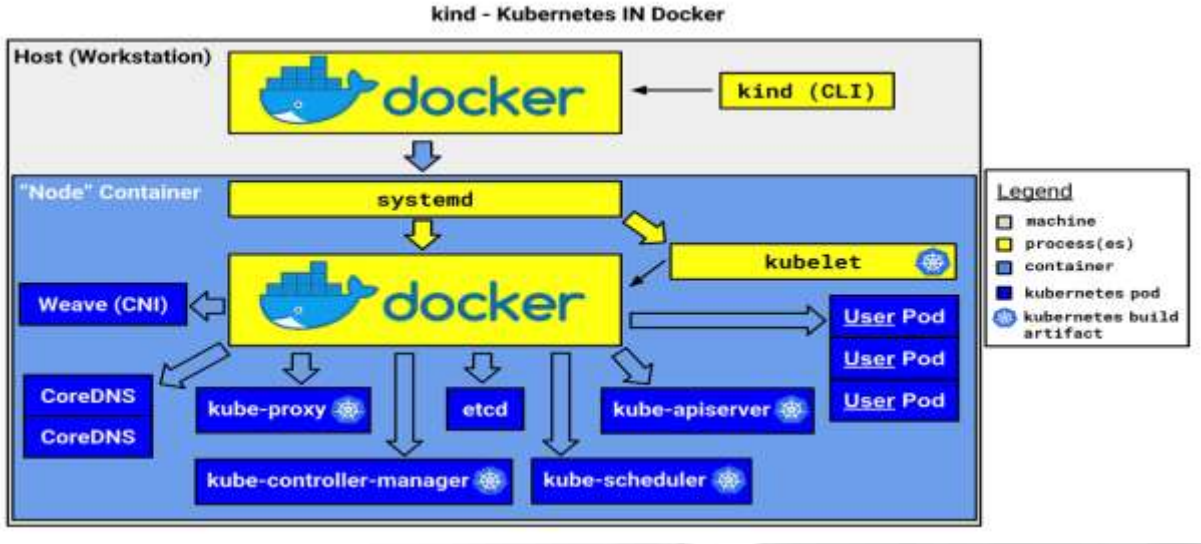
Kubernetes Cluster 구축

- Local Kubernetes
  - ✓ Minikube 활용
    - 물리 머신에 로컬 쿠버네티스를 쉽게 구축하고 실행할 수 있는 도구로 쿠버네티스의 SIG-Cluster-Lifecycle이라는 분과회(special interest group)에서 만듬
    - Minikube는 로컬 가상 머신에 쿠버네티스를 설치하기 위해 하이퍼바이저가 필요한데 도커 또는 Virtual Box 나 베어메탈 등
    - 설치: https://minikube.sigs.k8s.io/docs/start/



Kubernetes Cluster 구축

- Local Kubernetes
  - ✓ Kind
    - kind는 Docker 컨테이너 노드를 사용하여 로컬 Kubernetes 클러스터를 실행하기 위한 도구
    - kind는 주로 Kubernetes 자체를 테스트하기 위해 설계되었지만 로컬 개발이나 CI에 사용될 수도 있음



Kubernetes Cluster 구축

- Local Kubernetes
  - ✓ Kind
    - 설치: https://kind.sigs.k8s.io/docs/user/quick-start
    - kubectl 설치: https://kubernetes.io/ko/docs/tasks/tools/install-kubectl-linux/
    - 명령어
      - 클러스터 생성: kind create cluster
      - 클러스터 삭제: kind delete cluster
      - 설정 파일을 이용한 클러스터 생성: kind create cluster --cofing 설정파일경로 --name 클러스터이름

Kubernetes Cluster 구축

- ❖ Local Kubernetes
  - ✓ Kind
    - ● 여러 개의 Worker를 가진 클러스터 생성
      - ◆ 설정 파일을 생성하고 작성: kind.yaml

        apiVersion: kind.x-k8s.io/v1alpha4

        kind: Cluster

        nodes:

        - role: control-plane

        - role: worker

        - role: worker

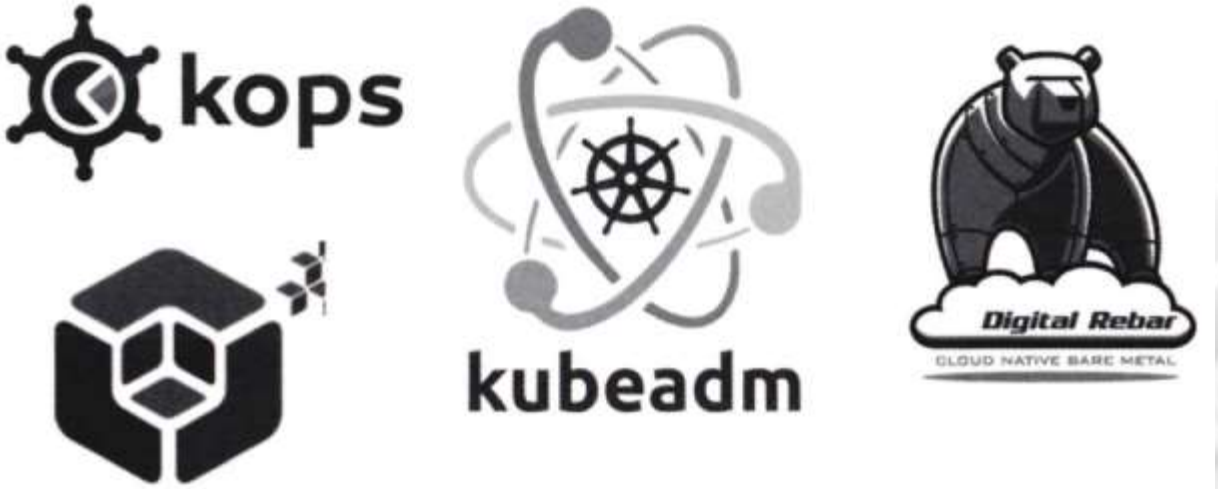
        - role: worker
      - ◆ 클러스터 생성: ./kind create cluster --config kind.yaml --name kindcluster
      - ◆ 여러 개의 클러스터 생성한 경우 클러스터 전환: kubectl config use-context kind-클러스터이름

Kubernetes Cluster 구축

- Local Kubernetes
  - ✓ K3s & K0S
    - 두 프로젝트들은 모두 Kubernetes 클러스터를 구축하는 데에 필수적인 구성 요소들을 간추려서 단일 파일로 설치 및 실행하기 때문에 리소스 경량화와 클러스터 구축 간소화를 모두 얻을 수 있음
    - 기존 Kubernetes는 클러스터의 데이터 저장소로 etcd를 사용하는데 K3S와 K0S는 경량화를 위해 etcd보다 더 가벼운 sqlite3를 기본 데이터 저장소로 사용하며 etcd나 MySQL 같은 다른 저장소도 지원하기 때문에 필요한 경우라면 사용하는 것도 가능
    - K3S와 K0S의 최소 요구사항
      - K3S: 1 CPU / 512MB RAM
      - K0S: 1 CPU / 1GB RAM
    - K3S와 K0S 모두 테스트 및 배포 환경에도 적합하지만 적은 리소스 사용량 덕에 IoT 환경에도 사용하기 적합
    - 두 프로젝트 모두 단일 파일로 클러스터 구성요소를 작동시키는 방식이기 때문에 현재 Linux 계열 운영체제만 공식 지원
    - K3S와 K0S는 Linux 환경에 익숙하면서 Kubernetes 개발 또는 테스트 환경이 필요하신 분들에게 추천하는 경량화 버전
    - 설치
      - K3S: https://docs.k3s.io/
      - K0S: https://docs.k0sproject.io/head/install/

Kubernetes Cluster 구축

- 구성형 Kubernetes
  - ✓ 사용하는 시스템에 Kubernetes Cluster를 자동으로 구성해주는 솔루션을 사용
  - ✓ 주요 솔루션으로는 kubeadm, kops(Kubernetes Operations), KRIB(Kubemetes Rebar Integrated Bootstrap), kubespray 등이 있음
  - ✓ kubeadm이 가장 널리 알려져 있는데 kubeadm은 사용자가 변경하기도 수월하고 온프레미스(On-Premises)와 클라우드를 모두 지원하며 배우기도 쉬운 편



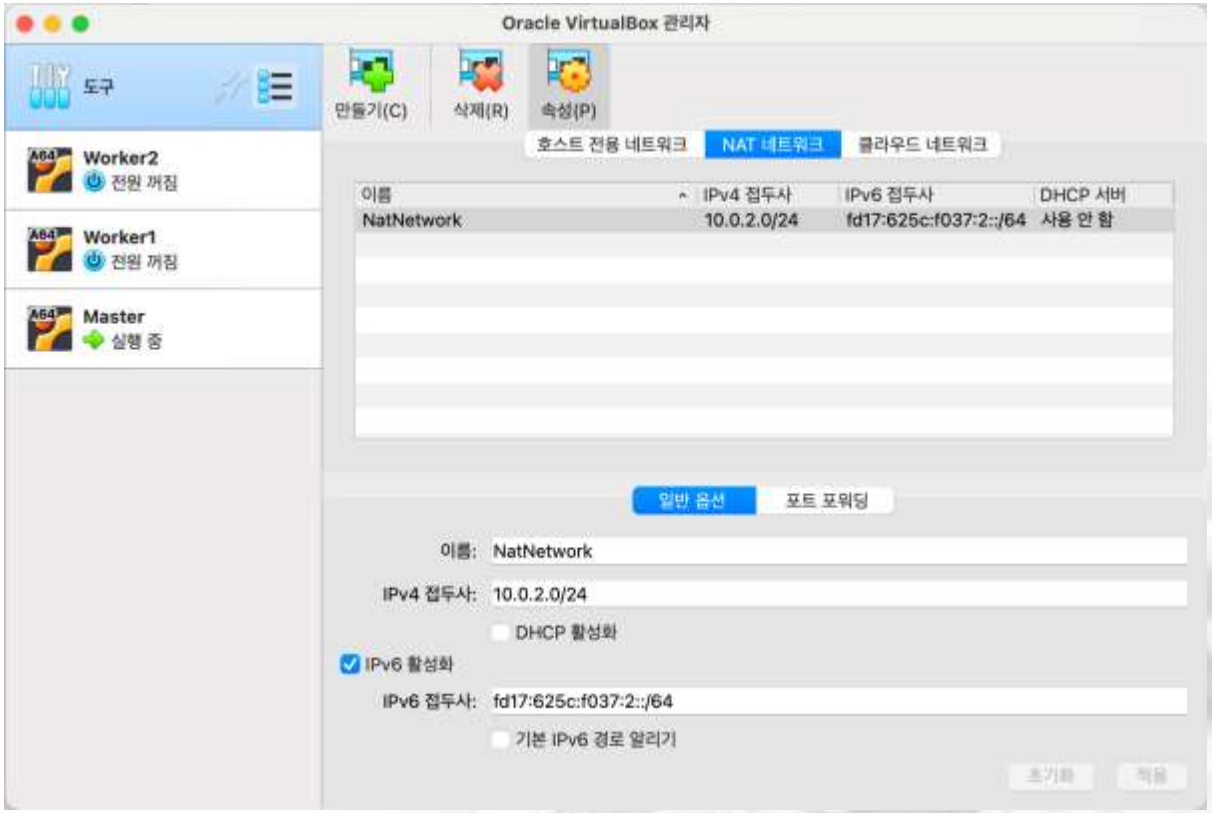
Kubernetes Cluster 구축

- 구성형 Kubernetes
  - ✓ Kubernetes Cluster 구성 솔루션

구분	KRIB	kops	kubespray	kubeadm
정보량	적음	많음	많음	매우 많음
세부 설정 변경	가능함	가능함	제한적으로 가능함	다양함
사전 요구 조건	적음	적음	적음	매우 적음
온프레미스 지원	매우 좋음	안 됨	좋음	매우 좋음
클라우드 지원	안 됨	매우 좋음	좋음	좋음
학습 곡선	매우 어려움	어려움	쉬움	어려움
자동화 기능	제공됨	제공됨	쉽게 제공됨	제한적으로 제공됨

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ Virtual Box를 이용한 Ubuntu 운영체제에 네트워크 연결
    - Nat Network 생성



Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ Virtual Box를 이용한 Ubuntu 운영체제에 네트워크 연결
    - ● Ubuntu 설정
      - ◆ 호스트 이름 수정
        - ▪ sudo hostnamectl set-hostname master
        - ▪ cat /etc/hostname

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ Virtual Box를 이용한 Ubuntu 운영체제에 클러스터 구축
    - ● Ubuntu 설치
      - ◆ IP 수정
        - ▪ 현재 네트워크 인터페이스 확인: ip a
        - ▪ netplan 설정 파일 수정: sudo nano /etc/netplan/00-installer-config.yaml

```
network:
  version: 2
  ethernets:
    enp0s8:
      dhcp4: no
      addresses:
        - 10.0.2.15/24
      routes:
        - to: default
          via: 10.0.2.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ Virtual Box를 이용한 Ubuntu 운영체제에 클러스터 구축
    - ● Ubuntu 설치
      - ◆ IP 수정
        - ▪ 소유권을 root로 변경: sudo chown root:root /etc/netplan/00-installer-config.yaml
        - ▪ 권한을 600(소유자 읽기/쓰기만 허용)으로 변경: sudo chmod 600 /etc/netplan//etc/netplan/00-installer-config.yaml
        - ▪ 변경 사항 적용: sudo netplan apply

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ Virtual Box를 이용한 Ubuntu 운영체제에 클러스터 구축
    - ● Ubuntu 설치
      - ◆ IP 주소 와 호스트 이름 매핑: sudo nano /etc/hosts

        127.0.0.1 localhost

        127.0.1.1 master

        10.0.2.15 master

        10.0.2.16 worker1

        10.0.2.17 worker2

Kubernetes Cluster 구축

- 구성형 Kubernetes
  - ✓ Virtual Box를 이용한 Ubuntu 운영체제에 클러스터 구축
    - 필요한 만큼 워커 노드는 복제해서 설정

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - Ubuntu 24.04에 쿠버네티스 클러스터 구축: https://hostnextra.com/learn/tutorials/how-to-install-kubernetes-k8s-on-ubuntu
    - Master Node
      - ◆ CPU가 2개 이상: Core가 1개이면 설치는 성공을 하지만 마스터 노드로 실행할 때 에러가 발생
      - ◆ 포트 개방
        - API Server: 6443
        - etcd Server: 2379, 2380
        - kubelet API: 10250
        - kube scheduler: 10251
        - kube controller manager 10252
        - Flannel CNI 플러그인에 대한 포트: 8285, 8472

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - ● Worker Node
      - ◆ 하드웨어 제약 없음
      - ◆ 포트 개방
        - ▪ API Server: 6443
        - ▪ kube-proxy가 서비스를 Load Balancing하기 위해서 사용하는 포트: 26443
        - ▪ Flannel CNI Plugin에 대한 포트: 8285, 8472
        - ▪ NodePort 로 사용할 포트: 30000-32767

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - 패키지 정보 업데이트
      - sudo apt update && sudo apt upgrade -y
      - sudo apt install -y curl apt-transport-https ca-certificates software-properties-common gnupg
    - Memory Swap 비활성화: 가상 메모리 사용으로 인한 문제점을 제거
      - sudo swapoff -a
      - sudo sed -i '/ swap / s/^₩(.*₩)$/#₩1/g' /etc/fstab
      - 확인: sudo free -m 과 sudo swapon -s

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - ● iptable 설정
      - ◆ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

```
overlay
br_netfilter
EOF
```

      - ◆ sudo modprobe overlay
      - ◆ sudo modprobe br_netfilter
      - ◆ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

```
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

      - ◆ sudo sysctl --system

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - ● 컨테이너 런타임 설치
      - ◆ sudo apt install -y containerd
      - ◆ sudo mkdir -p /etc/containerd
      - ◆ containerd config default | sudo tee /etc/containerd/config.toml
    - ● systemd를 cgroup으로 전환
      - ◆ sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml
      - ◆ sudo systemctl restart containerd
      - ◆ sudo systemctl enable containerd

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - ● 쿠버네티스 설치
      - ◆ sudo curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
      - ◆ echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list
      - ◆ sudo apt update
      - ◆ sudo apt install -y kubelet kubeadm kubectl
      - ◆ sudo apt-mark hold kubelet kubeadm kubectl

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - ● 마스터 노드에서 클러스터 생성
      - ◆ kubeadm을 초기화하고 파드의 네트워크 대역을 설정: sudo kubeadm init --pod-network-cidr=10.244.0.0/16
        - ▪ 아래 내용이 출력되는데 이 내용이 워커 노드에서 마스터 노드에 연결하기 위한 코드

          kubeadm join 172.31.1.97:6443 --token f1dop8.fcqs8gricz2az5ye ₩

          --discovery-token-ca-cert-hash sha256:8a806679be5851bbb2778662a0ca454188ef8fa4d446c6829ab9836ed564b462
      - ◆ 쿠버네티스 환경 설정 파일 생성
        - ▪ mkdir -p $HOME/.kube
        - ▪ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
        - ▪ sudo chown $(id -u):$(id -g) $HOME/.kube/config
      - ◆ CNI 설치: kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Kubernetes Cluster 구축

- ❖ 구성형 Kubernetes
  - ✓ kubeadm을 이용한 클러스터 구축
    - ● 마스터 노드에서 클러스터 생성
      - ◆ 방화벽 중지
        - ▪ sudo systemctl stop ufw
        - ▪ sudo systemctl disable ufw
      - ◆ 노드 확인: kubectl get nodes
      - ◆ 토큰 확인: kubeadm token list
      - ◆ 토큰 생성: kubeadm token create
      - ◆ 해시값 확인: openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt
    - ● 워커 노드에서 마스터 노드에 연결

```
sudo kubeadm join 172.31.1.97:6443 --token f1dop8.fcqs8gricz2az5ye ₩
        --discovery-token-ca-cert-hash
sha256:8a806679be5851bbb2778662a0ca454188ef8fa4d446c6829ab9836ed56
4b462
```

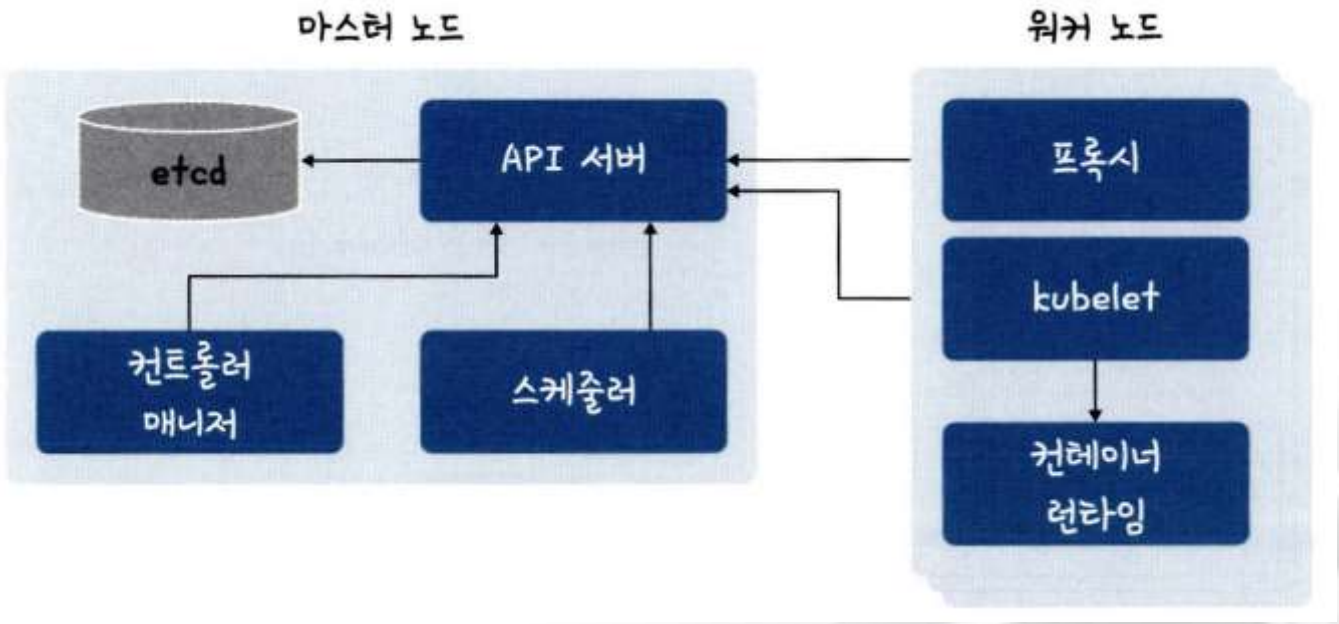
Kubernetes Cluster 구축

- 외부 관리형 서비스 사용
  - ✓ 기업에서 Kubernetes를 사용하는 비중이 높아지면서 Public Cloud 서비스 제공업체들은 Kubernetes를 PaaS 형식의 서비스로 출시
  - ✓ Public Cloud의 대표적인 서비스
    - AWS의 EKS(Elastic Kubernetes Service - https://docs.aws.amazon.com/ko_kr/eks/latest/userguide/getting-started-console.html)
    - MS에서는 AKS(Azure Kubernetes Service - https://learn.microsoft.com/ko-kr/azure/aks/learn/quick-kubernetes-deploy-portal?tabs=azure-cli)
    - Google에서는 GKE(Google Kubernetes Engine - https://cloud.google.com/apigee/docs/hybrid/v1.2/install-create-cluster?hl=ko)
    - Kakao Cloud 의 DKOS
    - NHN 의 NKS
  - ✓ 수세의 Rancher, 레드햇의 OpenShift와 같은 플랫폼에서 제공하는 설치형 Kubernetes를 사용할 수 있음

Kubernetes Component

❖ 개요

✓ Master Node에는 Cluster를 유지하고 제어하기 위한 다양한 Component가 있고 Worker Node에는 애플리케이션을 실행하기 위한 Component들이 있음



Kubernetes Component

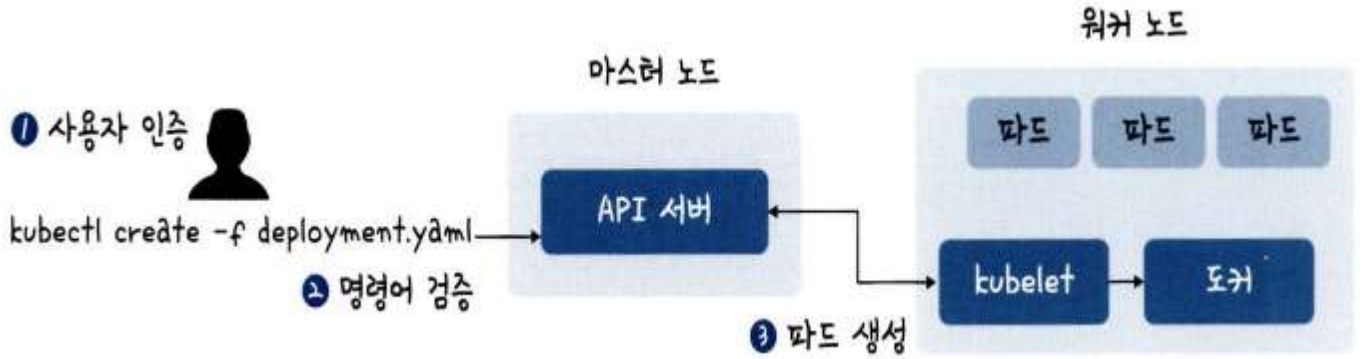
- kubectl
  - ✓ 쿠버네티스 클러스터에 명령을 내리는 역할
  - ✓ 다른 구성 요소들과 달리 바로 실행되는 명령 형태인 binary로 배포되기 때문에 마스터 노드에 있을 필요는 없음
  - ✓ 통상적으로 API Server 와 통신하므로 마스터 노드에 구성하는 것이 일반적

Kubernetes Component

❖ API Server

✓ API Server는 Kubernetes Cluster의 API를 사용할 수 있게 해주는 프로세스로 Cluster로 요청이 들어왔을 때 그 요청이 유효한지 검증

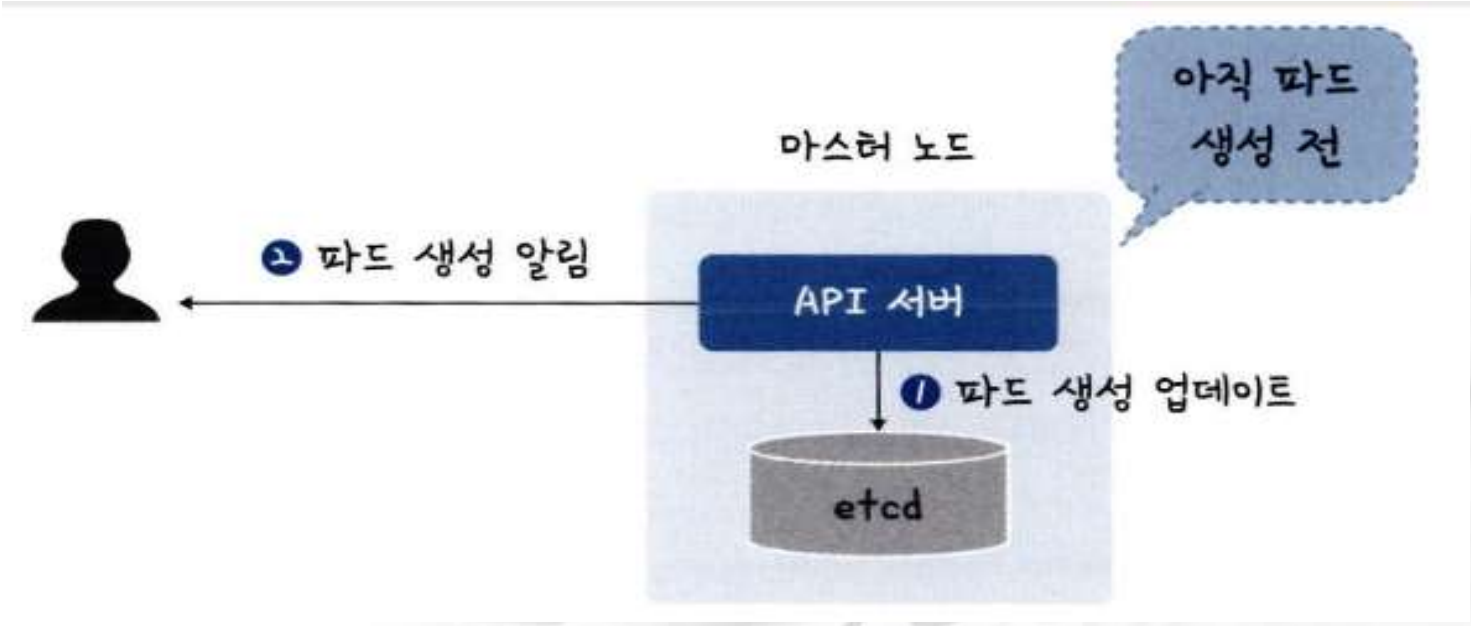
✓ 검증 과정



- Cluster에 접속할 권한이 있는 사용자인지를 검증
- 사용자가 보낸 명령어(kubectl create -f deployment.yaml)를 검증하는데 명령어가 문법에 맞게 작성되었는지 철자가 맞는지 등을 검증
- 사용자의 요청에 따라 명령(Pod 생성)을 수행하는데 API Server가 Worker Node에 Pod를 생성하도록 요청했지만 아직 생성은 안된 상태

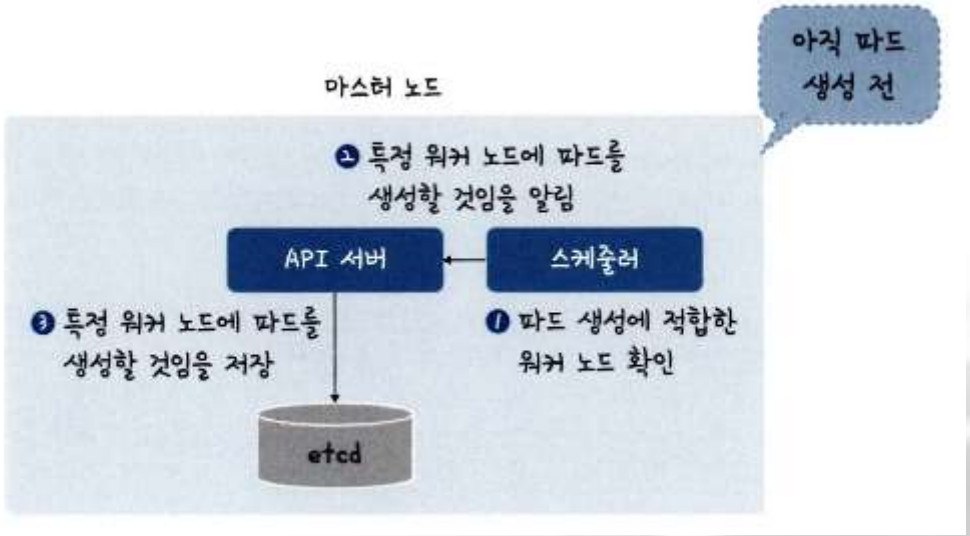
Kubernetes Component

- etcd
  - ✓ Cluster에 필요한 Pod 와 같은 리소스들의 상태 정보가 담겨 있는 곳(데이터베이스)
  - ✓ API Server는 Pod를 만든다는 사실을 etcd에 알려서 상태를 저장하고 사용자에게 Pod가 생성되었음을 알림
  - ✓ 정보들은 key-value 형태로 저장되는데 사용자에게 Pod가 생성되었음을 알렸지만 내부적으로는 여전히 Pod가 생성되지 않았음
  - ✓ etcd를 여러 곳에 저장해두면 장애가 나더라도 시스템의 가용성을 확보할 수 있음 – Multi Master Node



Kubernetes Component

- scheduler
  - ✓ 스케줄러는 Pod를 어떤 노드에 할당해야 할지를 결정하는데 Worker Node의 리소스 사용량 (CPU 나 메모리 등의 사용량)이나 레이블 그리고 노드의 상태 등 을 참조해 결정

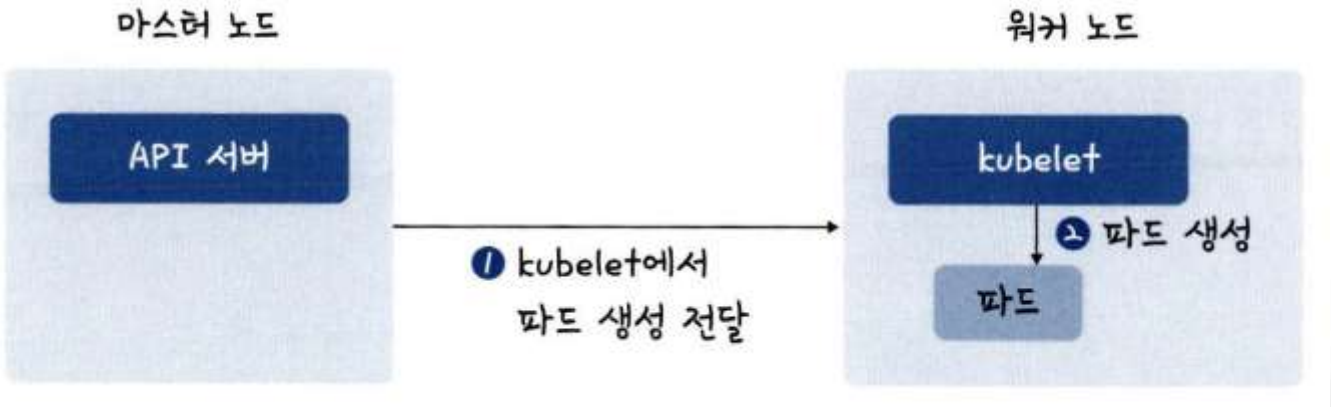


Kubernetes Component

- Controller Manager
  - ✓ 클러스터의 상태를 지속적으로 감시하고 선언된(desired) 상태와 실제 상태를 일치시키기 위해 다양한 컨트롤러(Controller)들을 실행하는 핵심 컴포넌트
  - ✓ 역할
    - 클러스터 내 리소스를 지속적으로 모니터링
    - 선언된 상태와 실제 상태를 비교하여 필요한 변경 수행
    - 컨트롤러들을 관리하고 실행
    - 장애가 발생하면 자동으로 복구 작업 수행
    - 다양한 Component의 상태를 지속적으로 모니터링하는 동시에 실행 상태를 유지하는 역할을 하는데 Controller Manager가 특정 Worker Node 와 통신이 불가능하다고 판단되면 해당 Node에 할당된 Pod를 제거하고 다른 Worker Node에서 Pod를 생성해 서비스가 계속되도록 함

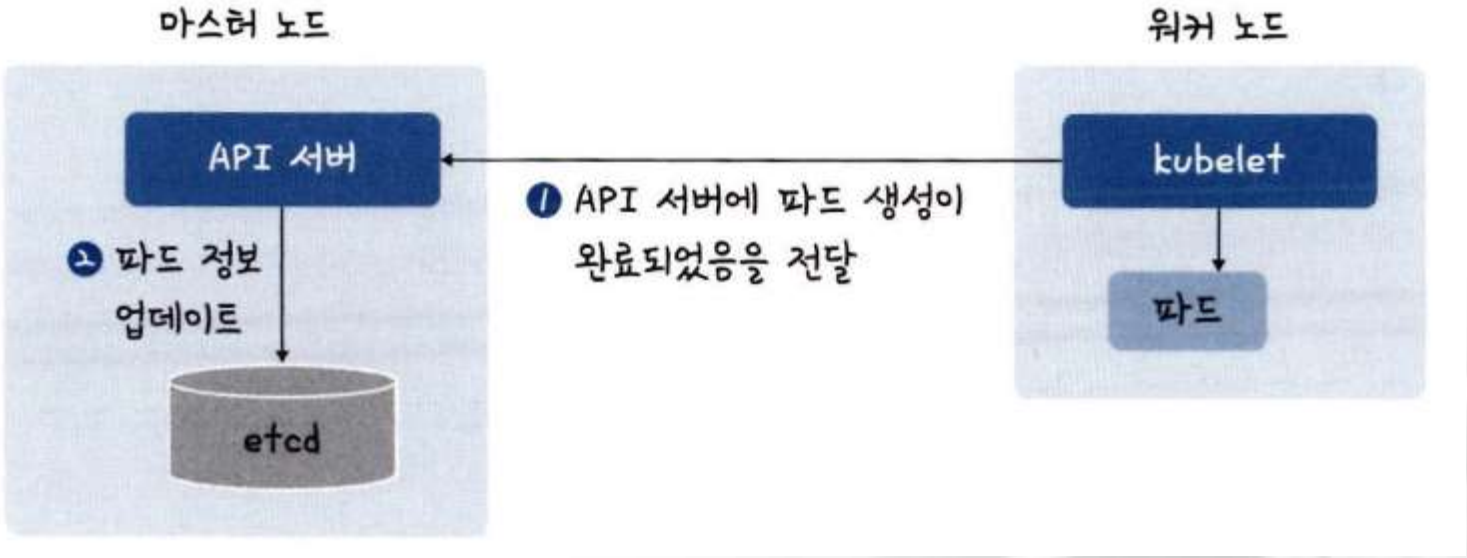
Kubernetes Component

- kubelet
  - ✓ API Server는 Pod가 생성될 Worker Node에 있는 kubelet에 Pod의 생성 정보를 전달
  - ✓ kubelet은 해당 정보를 이용해 Pod를 생성하는데 kubelet은 Cluster의 각 노드에서 실행되는 에이전트로 Pod에서 Container의 동작(생성 및 운영)을 관리



Kubernetes Component

- kubelet
  - ✓ Pod를 생성했다면 kubelet은 API Server에 생성된 Pod의 정보를 전달하고 API Server는 다시 etcd를 업데이트하는데 최종적으로 어떤 Worker Node에 어떤 Pod가 생성되었는지 etcd에 저장



Kubernetes Component

- kube-proxy
  - ✓ Kubernetes 클러스터 내에서 네트워크 프록시 역할을 하는 컴포넌트로 서비스(Service)와 파드(Pod) 간의 트래픽을 라우팅하는 역할을 수행
  - ✓ kube-proxy의 역할
    - 클러스터 네트워크 트래픽 관리
      - Kubernetes의 서비스 개념을 구현하기 위해 각 노드에서 실행됨
      - 클러스터 내부에서 서비스와 파드 간 트래픽을 적절히 분배
    - 서비스 IP 와 포트 포워딩
      - Kubernetes 서비스가 가상 IP(ClusterIP)를 사용해도 실제 파드로 트래픽이 전달되도록 설정
      - iptables, ipvs, 또는 userspace 모드를 활용하여 트래픽을 제어
  - ✓ kube-proxy의 실행 방식
    - Kubernetes 클러스터의 각 노드에서 데몬셋(DaemonSet) 으로 실행됨
    - kubectl get pods -n kube-system 명령어로 확인 가능

Kubernetes Component

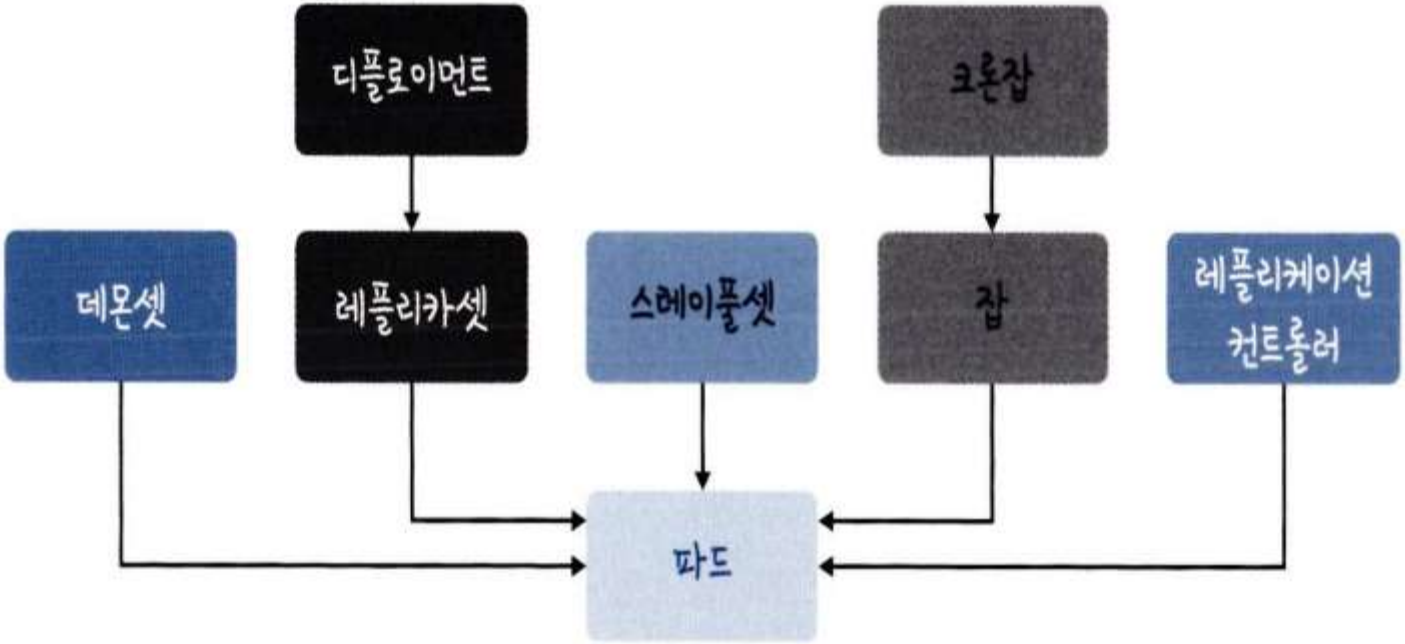
- ❖ kube-proxy
  - ✓ kube-proxy의 작동 방식
    - iptables 모드 (기본값)
      - ◆ 각 노드에서 iptables 규칙을 생성하여 패킷을 올바른 파드로 전달
      - ◆ 빠르고 효율적 (커널 레벨에서 처리됨)
      - ◆ 단점: 수천 개의 서비스가 있을 경우 iptables 규칙이 많아져 성능 저하 가능
    - ipvs 모드 (권장)
      - ◆ ipvsadm을 사용하여 커널 공간에서 패킷을 처리
      - ◆ 고성능, 대규모 트래픽 처리에 적합
      - ◆ L4 로드 밸런싱을 지원 (Round Robin, Least Connection 등)
      - ◆ ipvs 커널 모듈이 필요
    - userspace 모드 (비추천)
      - ◆ 트래픽을 kube-proxy 프로세스를 거쳐 전달 (유저 공간에서 처리)
      - ◆ 성능이 낮아서 현재는 잘 사용되지 않음

Kubernetes Component

- Container Runtime
  - ✓ Container 런타임은 Container 실행을 담당
  - ✓ Kubernetes는 다양한 종류의 런타임을 지원하는데 Docker(Kubernetes 1.20 버전부터 Docker는 지원 중단), Containerd, CRI-O 등

Kubernetes Component

- Kubernetes Controller
  - Kubernetes Controller는 Pod를 관리하는 역할을 수행
  - Kubernetes에서 제공하는 Controller는 DaemonSet, Deployment, ReplicaSet, StatefulSet, Job, CronJob, ReplicationController가 있음

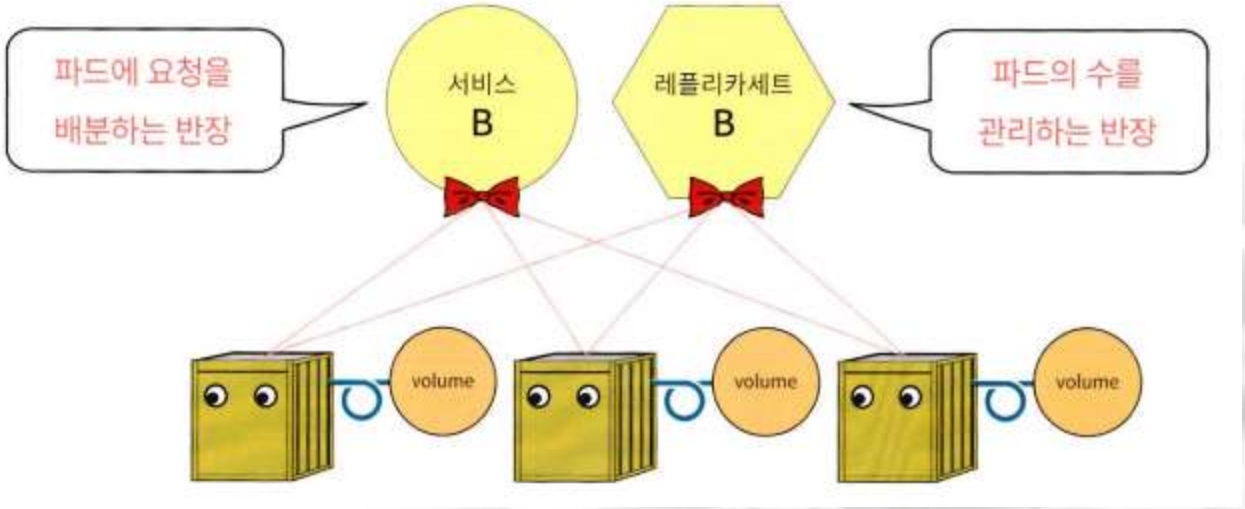


Kubernetes Component

- ❖ Kubernetes Controller
  - ✓ ReplicaSet
    - 몇 개의 Pod를 유지할 지 결정하는 Controller
    - Manifest

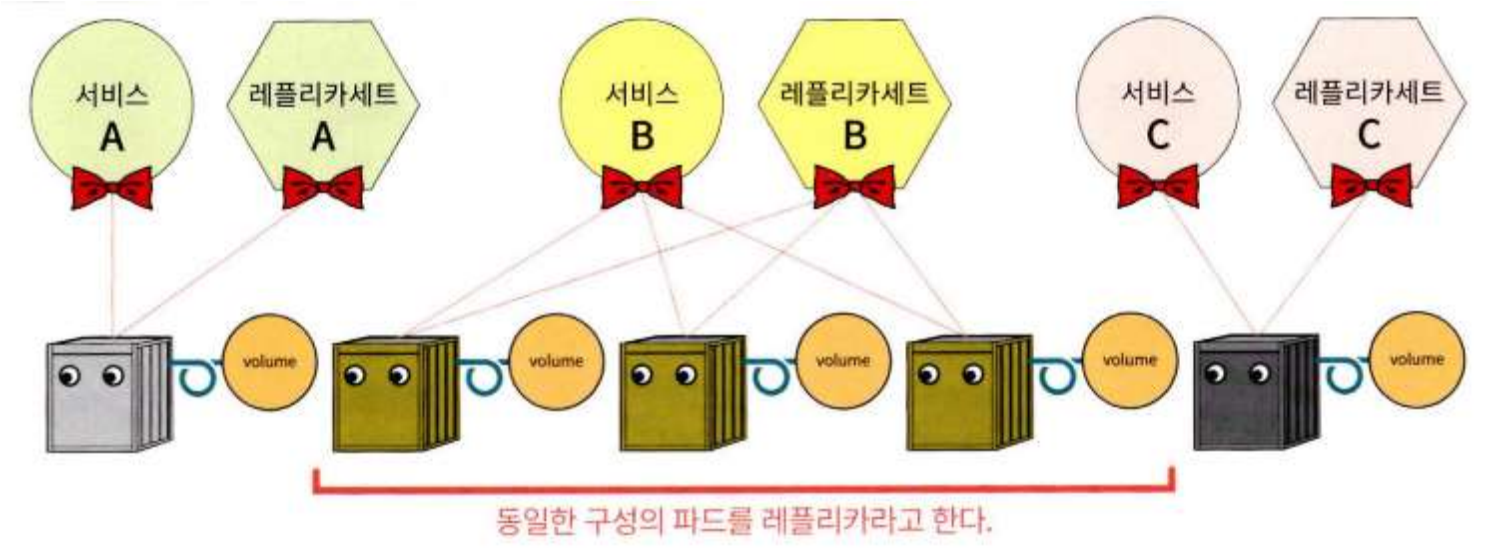
      apiVersion: apps/v1

      kind: ReplicaSet



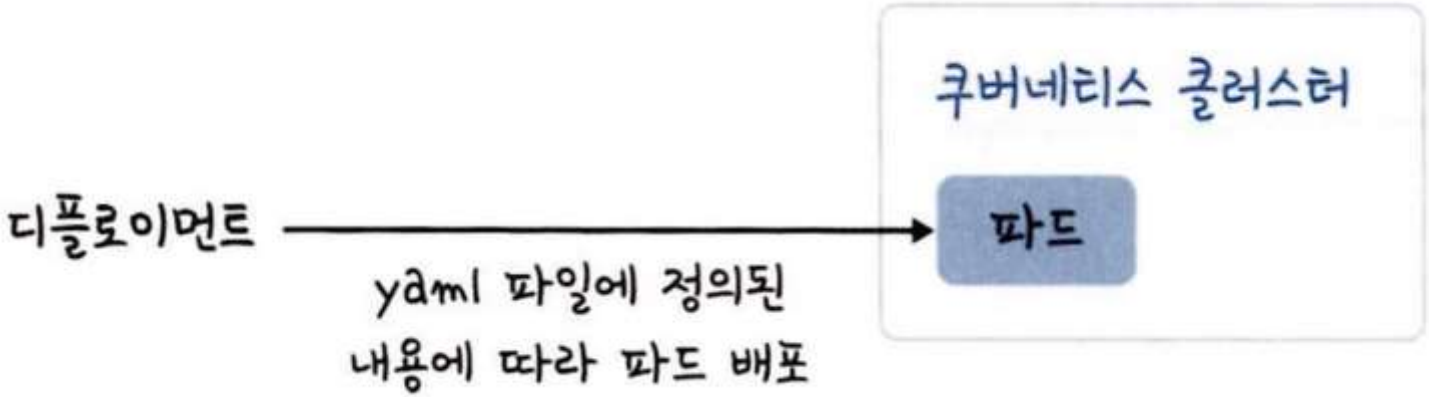
Kubernetes Component

- ❖ Kubernetes Controller
  - ✓ ReplicaSet
    - ReplicaSet이 관리하는 동일한 구성의 Pod를 Replica라고 부름
    - Pod의 수를 조정하는 것을 Replica의 수를 조정한다고 하거나 결정한다고 표현



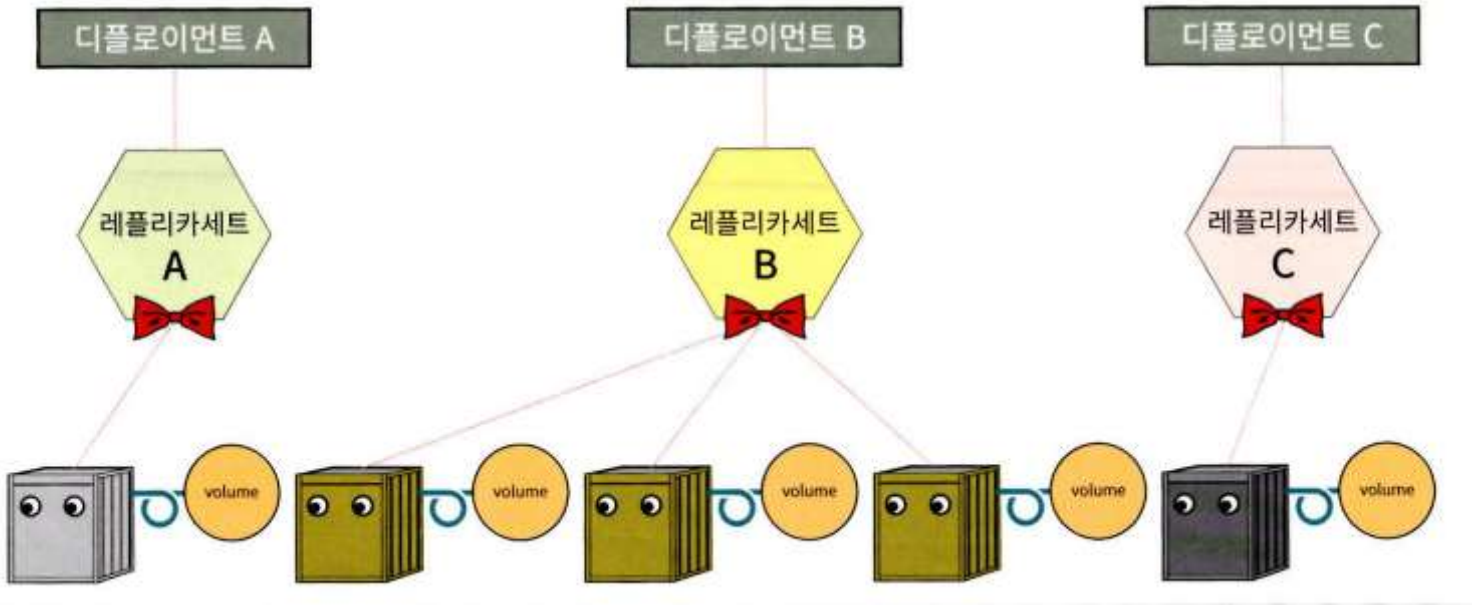
Kubernetes Component

- ❖ Kubernetes Controller
  - ✓ Deployment
    - Deployment는 Kubernetes에서 stateless 애플리케이션을 배포 할 때 사용하는 가장 기본적인 Controller
    - ReplicaSet의 상위 개념이면서 Pod를 배포할 때 사용하고 다양한 방법을 지원해서 배포할 때 세밀한 조작이 가능



Kubernetes Component

- Kubernetes Controller
  - ✓ Deployment



Kubernetes Component

- Kubernetes Controller
  - ✓ Deployment
    - Manifest

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

Kubernetes Component

- Kubernetes Controller
  - ✓ DaemonSet
    - Pod를 생성하고 관리하는 Controller
    - Deployment가 실행해야 할 Pod의 개수와 배포 전략을 세분화해 조작할 수 있다면 DaemonSet은 모든 Node에 Pod를 배포하고 관리하는 Controller
    - DaemonSet은 대부분 Node마다 배치되어야 하는 성능 수집 및 로그 수집 같은 작업에 사용
    - taint 와 toleration
      - Kubernetes Cluster를 운영하다 보면 특정 Worker Node에는 특정 성격의 Pod만 배포하고 싶을 때가 있는데 GPU가 설치된 Worker Node에는 GPU가 필요한 Pod만 배포하고 싶은 경우에 사용하는 것이 taint 와 toleration
      - taint가 설정된 노드에는 일반적으로 사용되는 Pod는 배포될 수 없으나 toleration을 적용하면 배포할 수 있음
    - Manifest

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

Kubernetes Component

- Kubernetes Controller
  - ✓ StatefulSet
    - StatefulSet은 애플리케이션의 Stateful을 관리하는데 사용하는 워크로드 API 오브젝트
    - Pod 집합의 Deployment 와 Scaling을 관리하며 Pod들의 순서 및 고유성을 보장
    - Deployment 와 유사하게 StatefulSet은 동일한 Container 스펙을 기반으로 둔 Pod들을 관리
    - Deployment 와는 다르게 StatefulSet은 각 Pod의 독자성을 유지하는데 이 Pod들은 동일한 스펙으로 생성되었지만 서로 교체는 불가능
    - 각각은 재 스케줄링 간에도 지속적으로 유지되는 식별자를 가짐
    - Storage Volume을 사용해서 워크로드에 지속성을 제공하려는 경우 솔루션의 일부로 StatefulSet을 사용할 수 있음
    - Manifest

```
apiVersion: apps/v1
kind: StatefulSet
```

Kubernetes Component

- Kubernetes Controller
  - ✓ Job
    - 하나 이상의 Pod를 지정하고 지정된 수의 Pod가 성공적으로 실행되도록 해주는 Controller
    - 노드의 하드웨어 장애나 재부팅 등으로 Pod가 비정상적으로 작동하면(혹은 실행되지 않으면) 다른 노드에서 Pod를 시작해 서비스가 지속되도록 함
    - Manifest

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

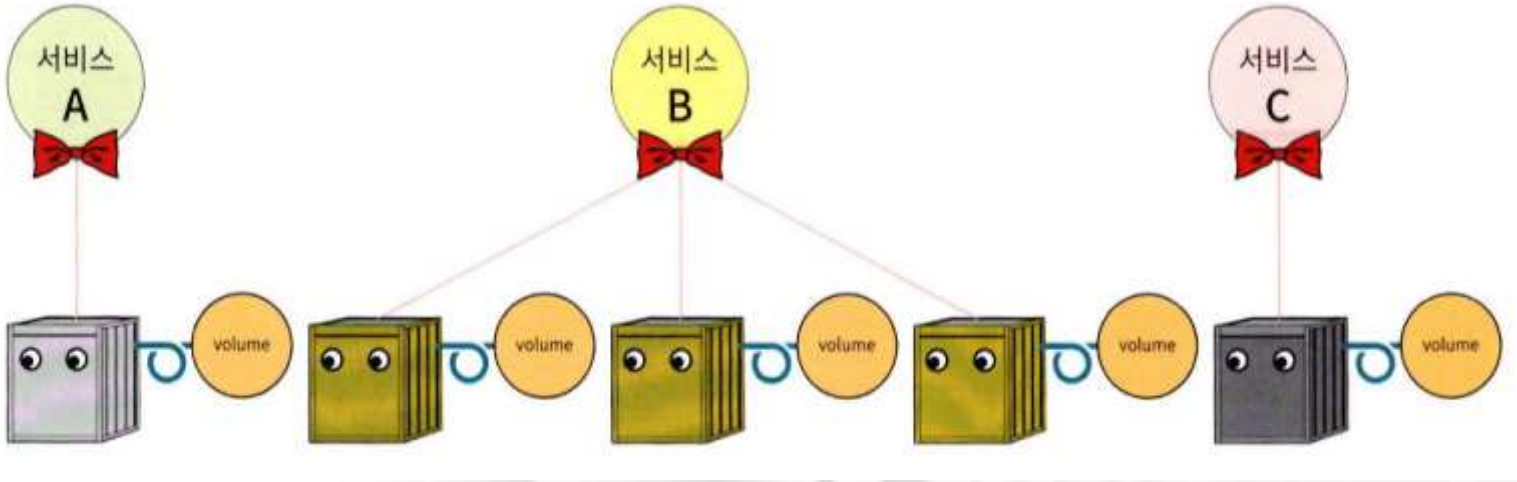
Kubernetes Component

- Kubernetes Controller
  - ✓ CronJob
    - Job의 일종으로 특정 시간에 특정 Pod를 실행시키는 것과 같이 지정한 일정에 따라서 Job을 실행시킬 때 사용
    - CronJob은 애플리케이션 프로그램, 데이터베이스 등에서 데이터를 백업하는 데 주로 사용
    - Manifest

```
apiVersion: batch/v1
kind: CronJob
Spec:
  schedule: */1 * * * *
```

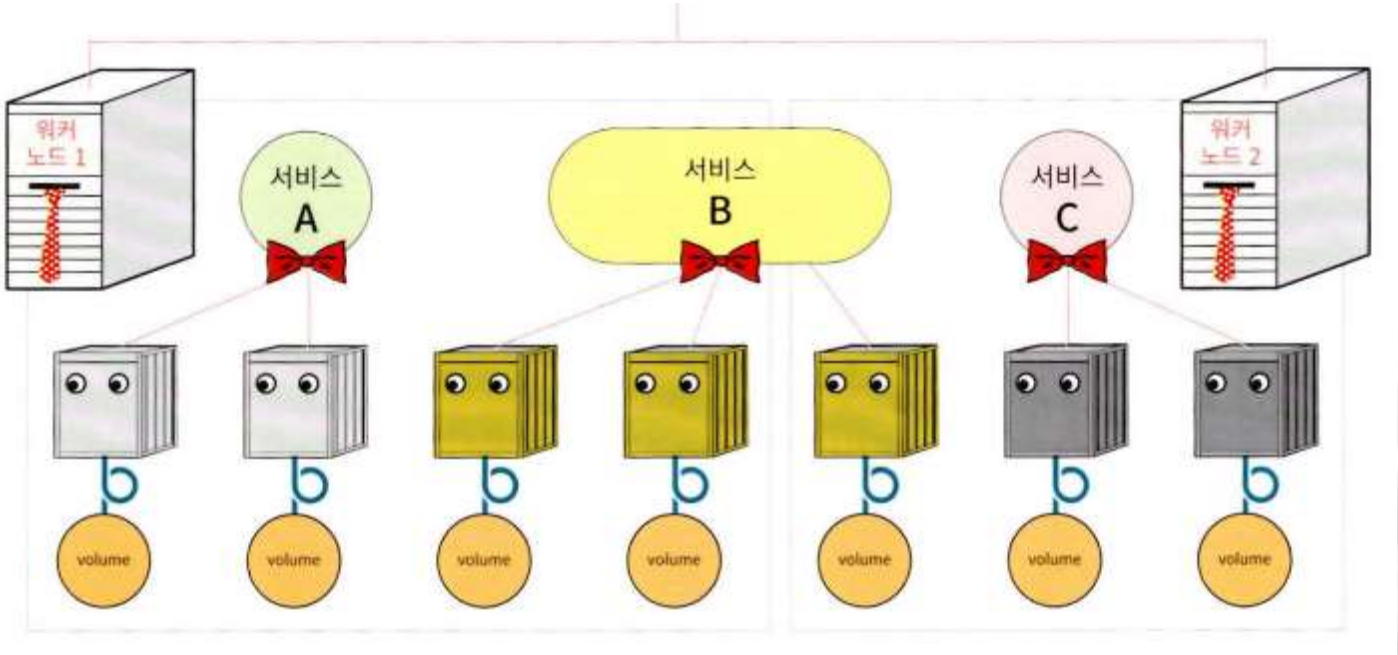
Kubernetes Communication

- Kubernetes Service
  - ✓ 개요
    - Pod는 Kubernetes Cluster 안에서 옮겨 다니는 특성이 있는데 Pod가 실행 중인 Worker Node에 문제가 생기면 다른 Worker Node에서 Pod가 다시 생성될 수 있고 Pod IP가 변경되기도 함
    - 동적으로 변하는 Pod에 고정된 방법으로 접근하기 위해서 사용하는 것이 Service
    - Service를 사용하면 Pod가 Cluster 내의 어디에 떠 있든지 고정된 주소를 이용해 접근할 수 있으며 Cluster 외부에서 Pod에 접근할 수도 있음
    - 하나의 Service가 관리하는 Pod는 모두 동일한 구성을 갖는데 구성이 다른 Pod는 별도의 Service로 관리



Kubernetes Communication

- ❖ Kubernetes Service
  - ✓ 개요
    - ● 여러 개의 Worker Node에 걸쳐 실행되더라도 동일한 구성의 Pod는 하나의 Service가 관리 가능



Kubernetes Communication

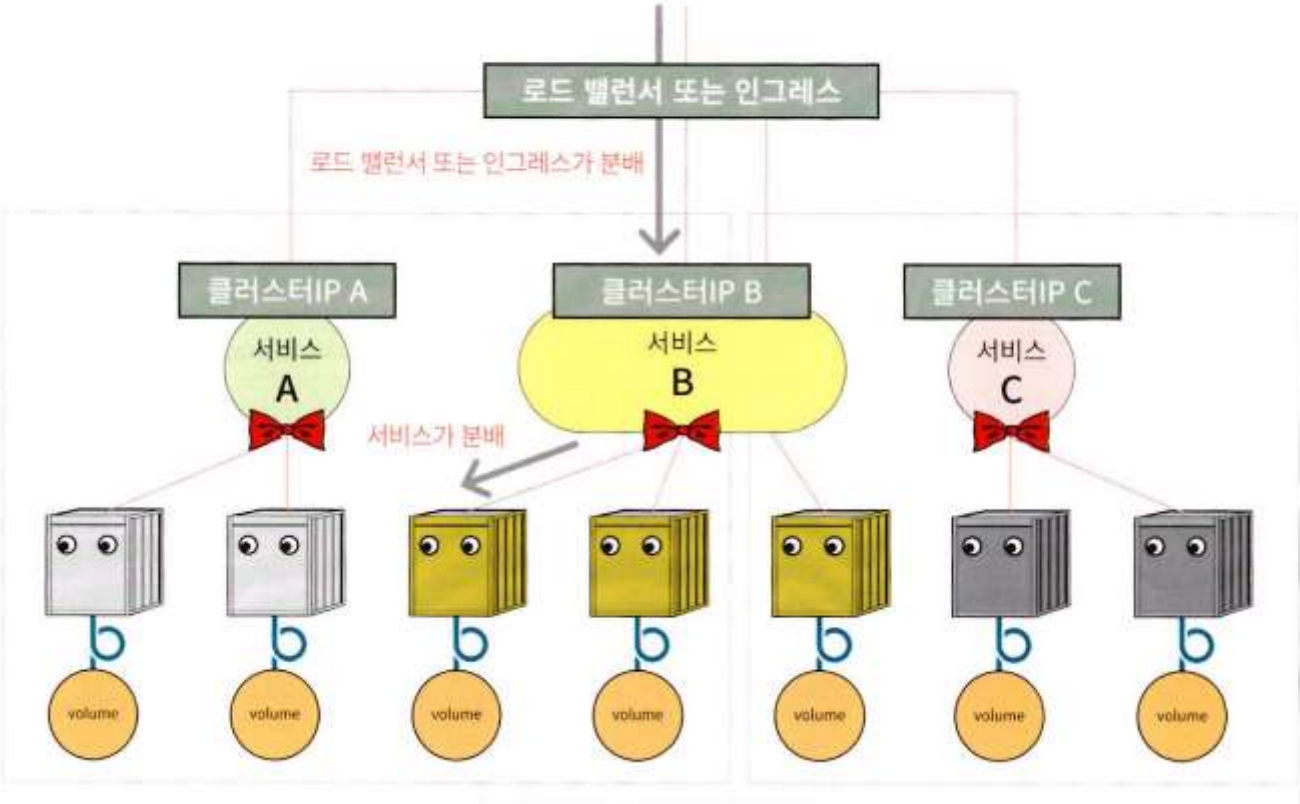
- ❖ Kubernetes Service
  - ✓ 개요
    - 서비스의 역할은 Load Balancer(부하 분산 장치)로 각 서비스는 자동적으로 고정된 IP 주소(cluster IP) 를 설정해서 이 주소로 들어오는 통신을 처리하는 객체
    - 내부적으로는 여러 개의 Pod가 있어도 밖에서는 하나의 IP 주소(cluster IP)만 볼 수 있으며 이 주소로 접근하면 서비스가 통신을 적절히 분배해주는 구조
    - 서비스가 분배하는 통신은 한 Worker Node 안으로 국한되며 여러 Worker Node 간의 분배는 Load Balancer 또는 Ingress가 담당하는데 이들은 Master Node도 Worker Node도 아닌 별도의 노드에서 동작하거나 물리적 전용 하드웨어

Kubernetes Communication

- Kubernetes Service
  - ✓ 종류
    - Cluster IP: Kubernetes Cluster 내의 Pod들은 기본적으로 외부에서 접근할 수 있는 IP를 할당받지 않지만 같은 Cluster 내부에서는 Pod들이 통신할 방법을 제공해주어야 하는데 그것이 Cluster IP로 Cluster 내의 모든 Pod가 해당 Cluster IP 주소로 접근할 수 있음
    - Node Port: 서비스를 외부로 노출 할 때 사용하는 것으로 노드 포트로 서비스를 노출하기 위해 Worker Node의 IP 와 포트를 이용하는데 Worker Node의 IP가 192.168.2.3 이고 30010이라는 Node Port를 사용한다면 외부에서 192.168.2.3:30010 으로 접근할 수 있음
    - Load Balancer: Load Balancer는 Public Cloud에 존재하는 Load Balancer에 연결하고자 할 때 사용되는 것으로 사용자는 Load Balancer의 외부 IP를 통해 접근
    - Ingress: URI, Hostname, Path 등과 같은 웹 개념을 이해하는 프로토콜로 인지형 (protocol-aware configuration) 설정 메커니즘을 이용하여 HTTP(혹은 HTTPS) Network 서비스를 사용 가능하게 해주는 것으로 Ingress 개념은 kubernetes API를 통해 정의한 규칙에 기반하여 트래픽을 다른 Service에 매핑할 수 있게 해줌
    - External Name: 외부의 특정 FQDN(Fully Qualified Domain Name - 완전한 도메인 이름)에 대한 CNAME(Canonical Name - 도메인 이름을 다른 도메인 이름으로 매핑하는 역할) 매핑을 제공하는 것으로 Pod가 CNAME을 이용해 특정 FQDN과 통신하기 위함.

Kubernetes Communication

- Kubernetes Service



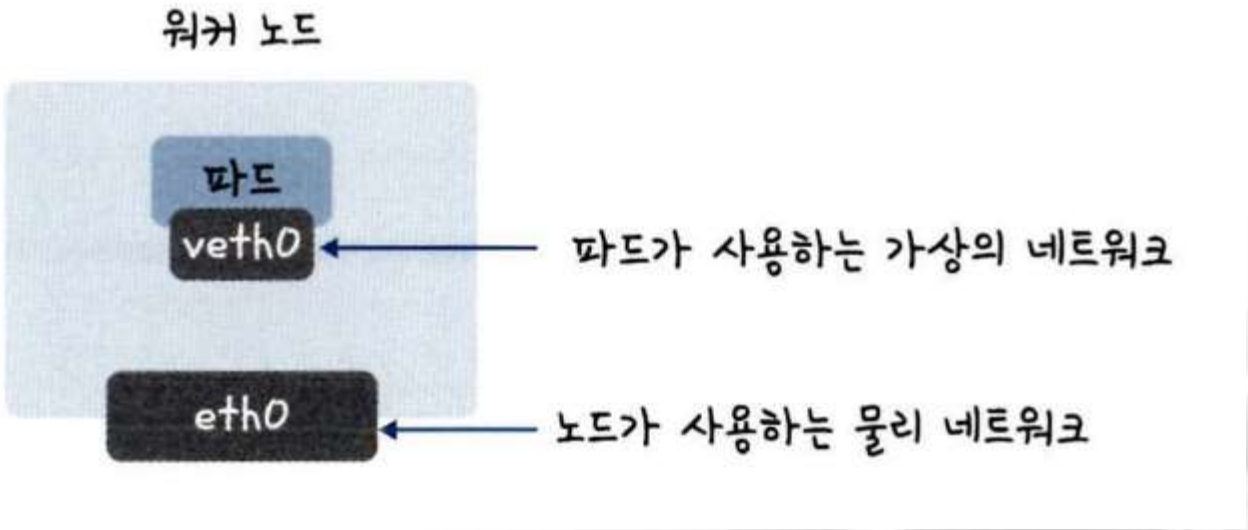
Kubernetes Communication

- Kubernetes Service
  - ✓ Manifest

```
apiVersion: v1
Kind:Service # Service 작업으로 명시
spec:
        type: ClusterIP
        selector:
```

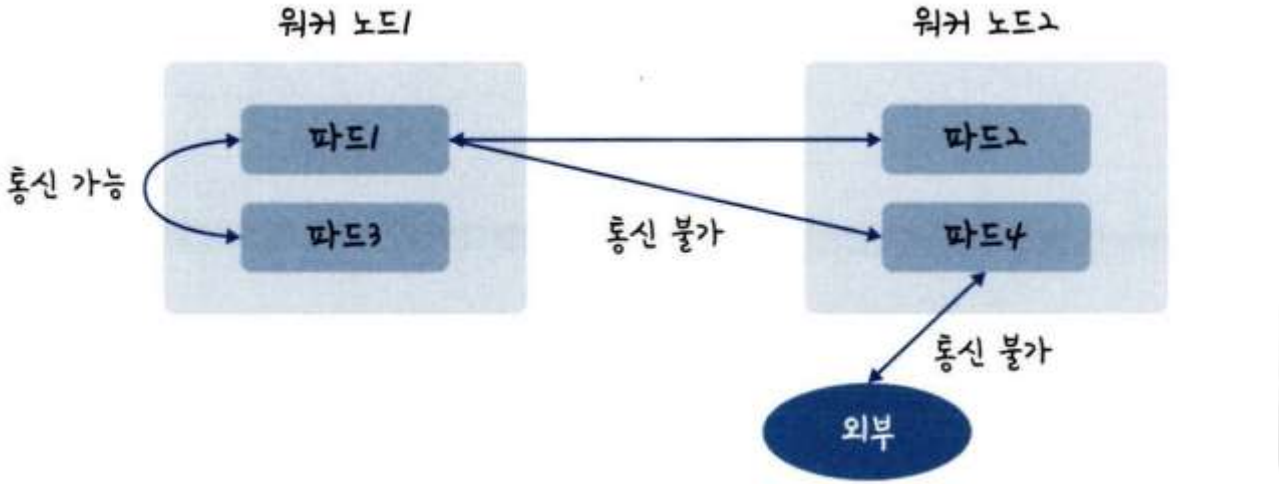
Kubernetes Communication

- Kubernetes에서 통신할 때 나타나는 특징
  - ✓ Pod가 사용하는 Network와 호스트(노드)가 사용하는 Network는 다름: 노드 내의 Pod들은 가상의 Network(Pod 및 Container가 사용하는 Network)를 사용하고 호스트는 물리 Network(Master 및 Worker Node가 사용하는 Network)를 사용



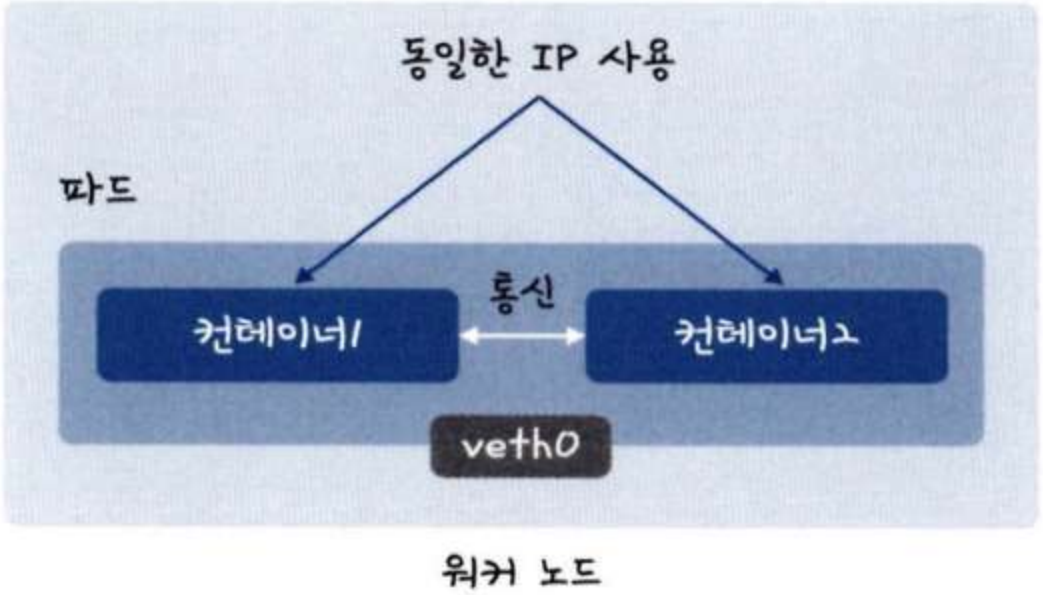
Kubernetes Communication

- Kubernetes에서 통신할 때 나타나는 특징
  - ✓ 동일한 Node에 떠 있는 Pod 끼리 통신할 수 있음: 동일한 Node에 떠 있는 Pod들은 통신이 가능하지만 다른 노드의 Pod 또는 외부 와의 통신은 불가능



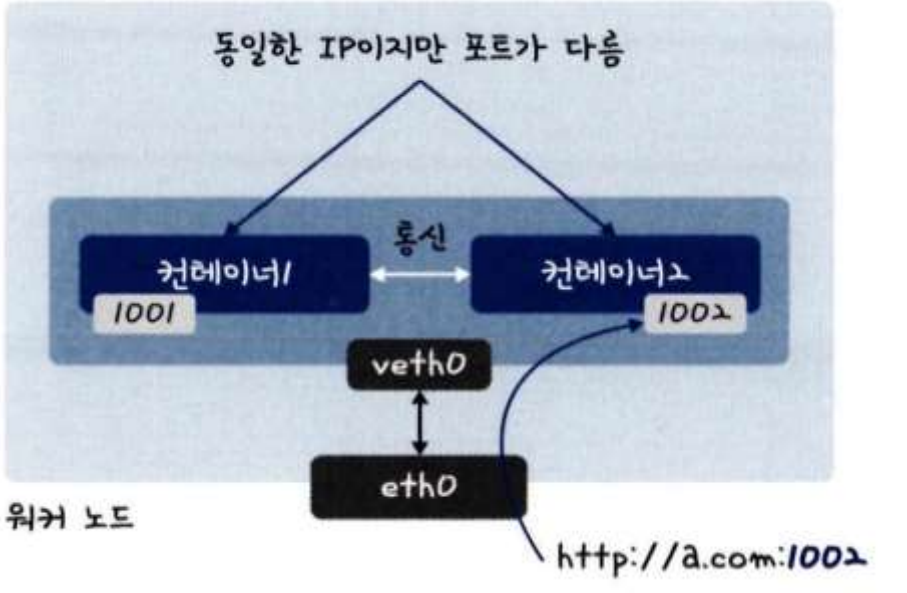
Kubernetes Communication

- 같은 Pod에 포함된 Container 간 통신
  - ✓ 기본적으로 같은 Pod 내에 있는 Container 간의 통신은 직접 통신(로컬 호스트 통신)이 가능한데 하나의 Pod에는 하나의 가상 Network가 생성되며 그 Pod 내에 존재하는 Container들은 같은 가상 Network를 사용하기 때문
  - ✓ 하나의 Pod 내에 존재하는 Container들은 모두 동일한 IP 사용



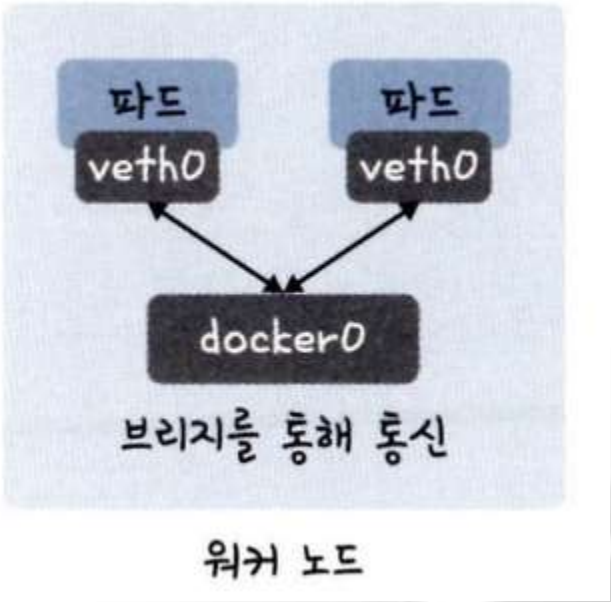
Kubernetes Communication

- 같은 Pod에 포함된 Container 간 통신
  - ✓ 같은 Pod 내에 존재하는 Container들은 모두 동일한 IP를 사용하는데 이를 구별하기 위해서 포트 번호를 이용
  - ✓ IP는 같지만 포트 번호가 다르므로 두 개의 Container를 구분할 수 있음
  - ✓ 사용자가 Container2의 서비스에 접근한다면 http://a.com:1002처럼 URL에 포트 정보를 입력하면 이후 eth0와 veth0을 거쳐 서비스(Container2)에 접근



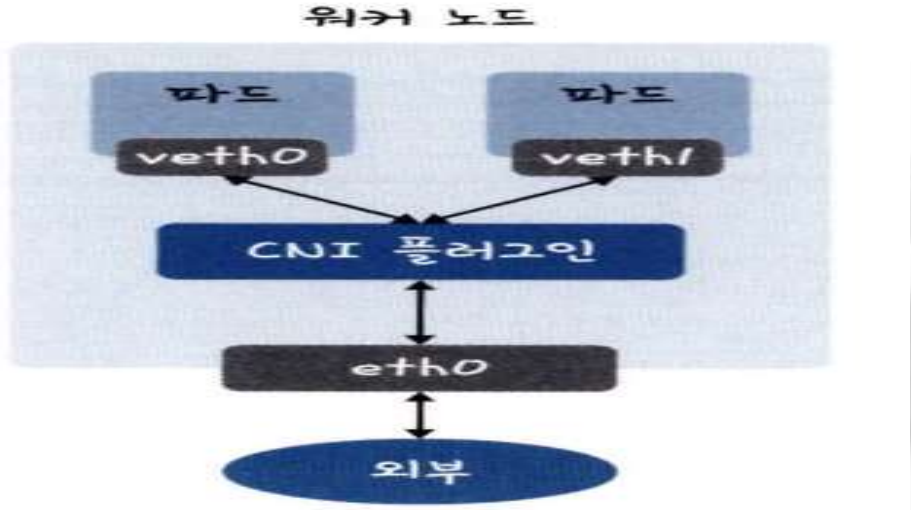
Kubernetes Communication

- 단일 Node에서 Pod간 통신
  - ✓ veth0 과 eth0 사이에는 docker0이라고 하는 브리지가 존재
  - ✓ 단일 노드에 떠 있는 Pod들은 같은 대역(veth0: 172.10.0.1, veth0: 172.10.0 2)을 사용하므로 docker0 라는 브리지를 통해 서로 통신



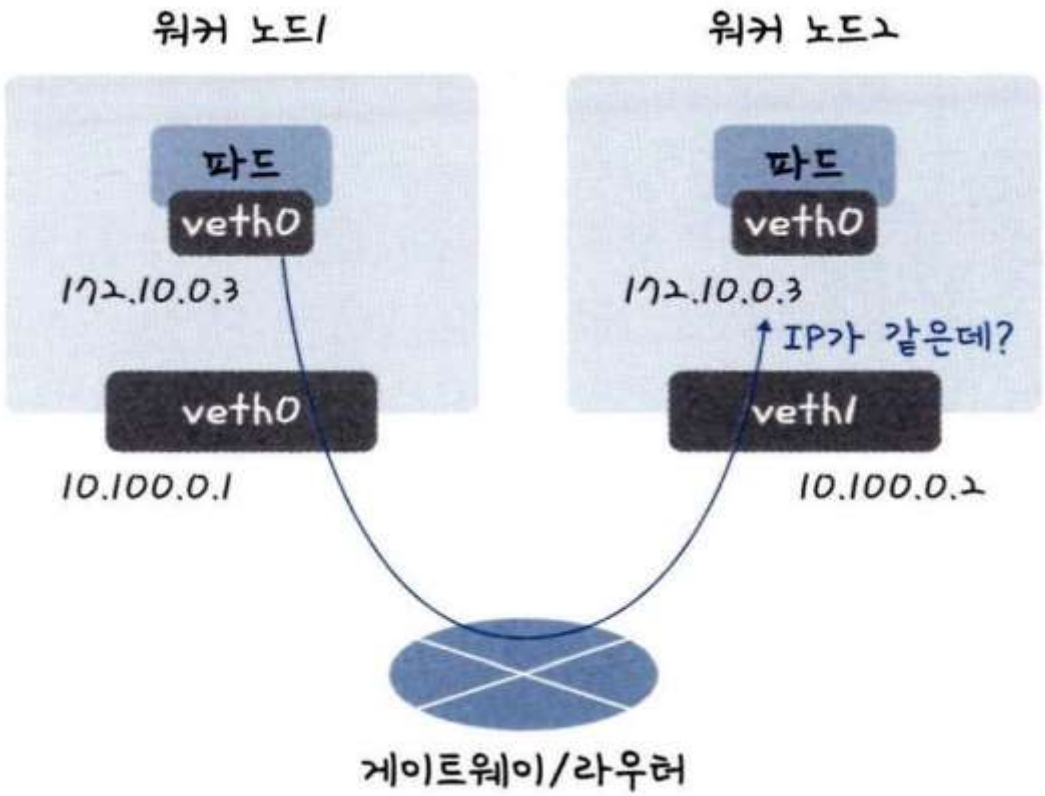
Kubernetes Communication

- 다른 노드의 Pod와 통신하려면 CNI Plugin 필요
  - ✓ CNI(Container Network Interface)는 Container 간의 통신을 위한 Network 인터페이스
  - ✓ CNI Plugin은 Container들의 Network를 연결하거나 삭제하며 삭제할 때 할당된 자원을 제거



Kubernetes Communication

- 다수 Node에서 Pod 간 통신
  - ✓ 다수 노드에서 Pod 간 통신 문제

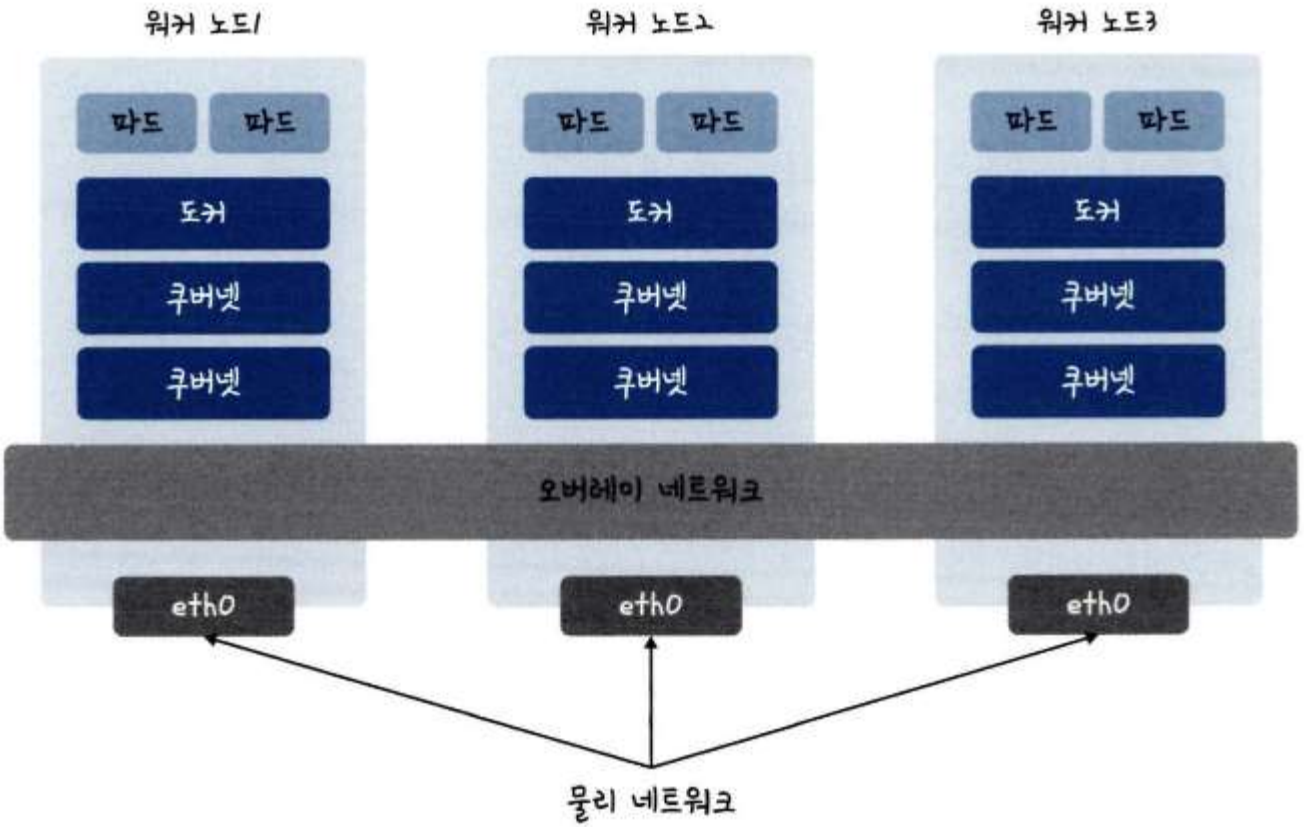


Kubernetes Communication

- 다수 Node에서 Pod 간 통신
  - ✓ 서로 다른 Worker Node에 존재하는 Pod의 IP가 동일하게 할당되는 문제가 발생하는데 이 문제를 해결할 수 있는 방법이 Overlay Network
  - ✓ Overlay Network란 노드에서 사용하는 물리적인 Network 위에 가상의 Network를 구성하는 것
  - ✓ Overlay Network를 사용하면 Kubernetes Cluster로 묶인 모든 노드에 떠 있는 Pod 간의 통신이 가능
  - ✓ Kubernetes는 Overlay Network를 구성하기 위해 Cluster를 구성할 때 CNI 규약을 따르는 Plugin을 함께 설치하는 것을 강제
  - ✓ Kubernetes는 기본적으로 kubenet이라는 기본적이고 간단한 Network Plugin을 제공하지만 다수의 노드에 떠 있는 Pod 간의 통신이나 Network 정책 설정 같은 고급 기능은 제공하지 않기 때문에 CNI의 스펙을 준수하는 Network Plugin을 사용해야 함

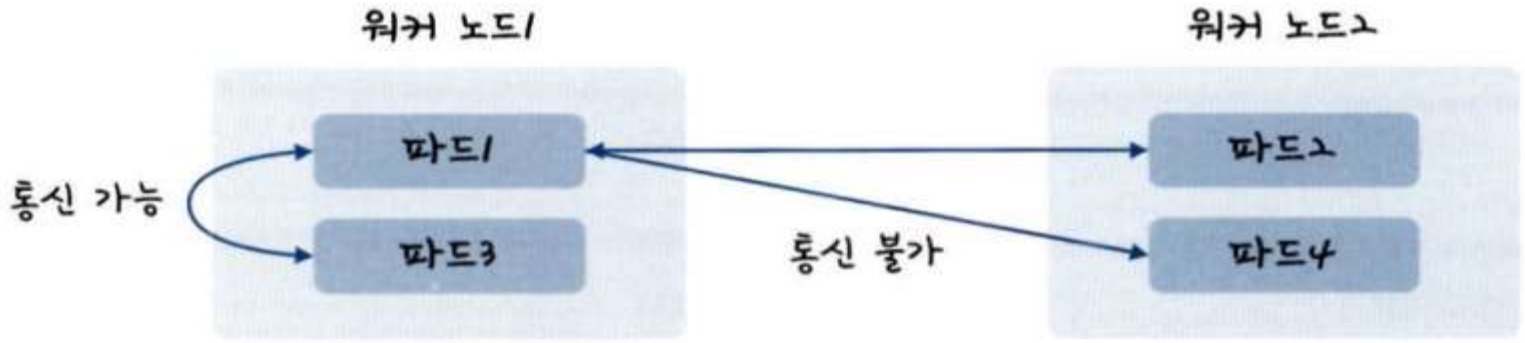
Kubernetes Communication

❖ 다수 Node에서 Pod 간 통신



Kubernetes Communication

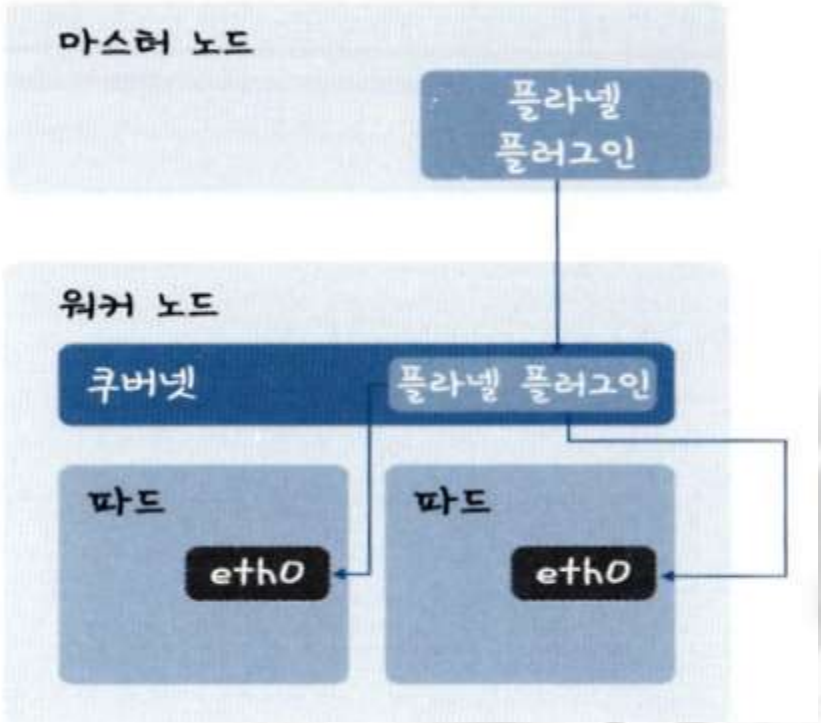
- 다수 Node에서 Pod 간 통신
  - ✓ CNI(Container Network Interface)
    - Pod 간의 통신은 동일한 호스트 내에서만 가능
    - Pod1 과 Pod3은 통신할 수 있지만 Pod1 과 Pod 2는 통신할 수 없고 Pod1 과 Pod2가 통신하기 위해 CNI 플러그 인을 사용



- 각 노드에 설치된 CNI Plugin을 통해 Pod들이 통신
- Kubernetes Cluster 구성에 필요한 kubeadm은 기본적으로 CNI 기반 Plugin을 지원
- Kubernetes에서 기본으로 제공하는 kubenet이 아닌 CNI Plugin을 별도로 구성해서 사용

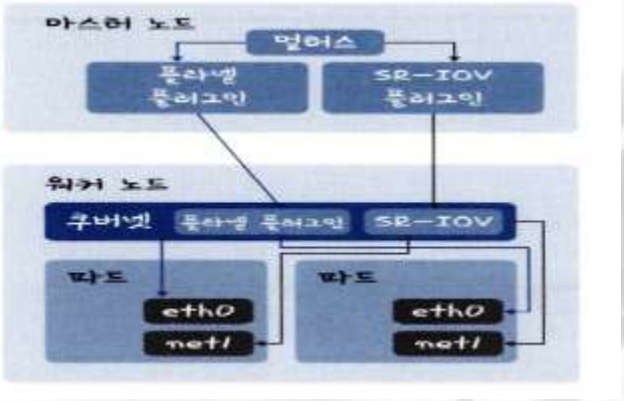
Kubernetes Communication

- 다수 Node에서 Pod 간 통신
  - ✓ Kubernetes에서 사용할 수 있는 CNI 플러그인
    - Flannel: Kubernetes와 함께 사용할 수 있는 Overlay Network 플러그인



Kubernetes Communication

- 다수 Node에서 Pod 간 통신
  - ✓ Kubernetes에서 사용할 수 있는 CNI 플러그인
    - Calico: 캡슐화 또는 Overlay 없이 파트 간에 통신이 가능한 Network를 제공하는 Plugin
    - Cilium: 리눅스 커널 기술을 이용해 데이터 경로의 필터링, 트래픽 모니터링 및 리디렉션을 수행
    - Multus: Kubernetes에서 다중 Network 지원을 위한 Multi Plugin으로 2개 이상의 Plugin을 사용할 때 유용한데 멀터스는 CNI 사양을 구현하는 다양한 유형의 Plugin(Flannel, DHCP, Macvlan, Calico, Cilium)을 지원하고 NFV 기반 응용 프로그램과 함께 SRIOV, DPDK, OVS-DPDK, VPP도 지원

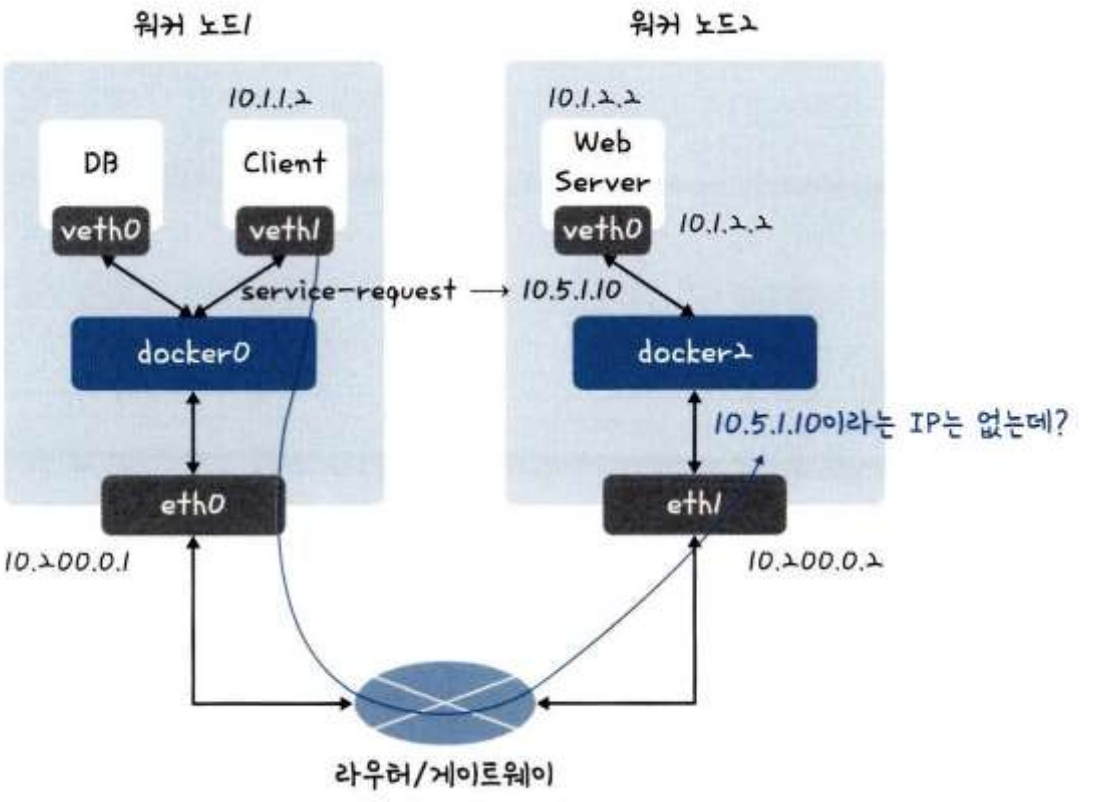


Kubernetes Communication

- Pod 와 Service 간 통신
  - ✓ Service의 특성
    - Service도 Pod와 같이 IP를 갖음
    - Pod 와 Service에서 사용하는 IP 대역은 다름 - Pod의 IP 대역은 10.244.1.x지만 Service에서 사용하는 IP 대역은 10.101.X.X
    - Service에서 사용하는 가상 Network는 ifconfig나 Routing Table에서 확인할 수 없음
  - ✓ Service IP 와 관련된 정보를 Routing Table에서 확인할 수 없지만 외부에서 서비스 IP로 Service를 요청하면 Service IP는 특정 Pod에 전달됨

Kubernetes Communication

❖ Pod 와 Service 간 통신

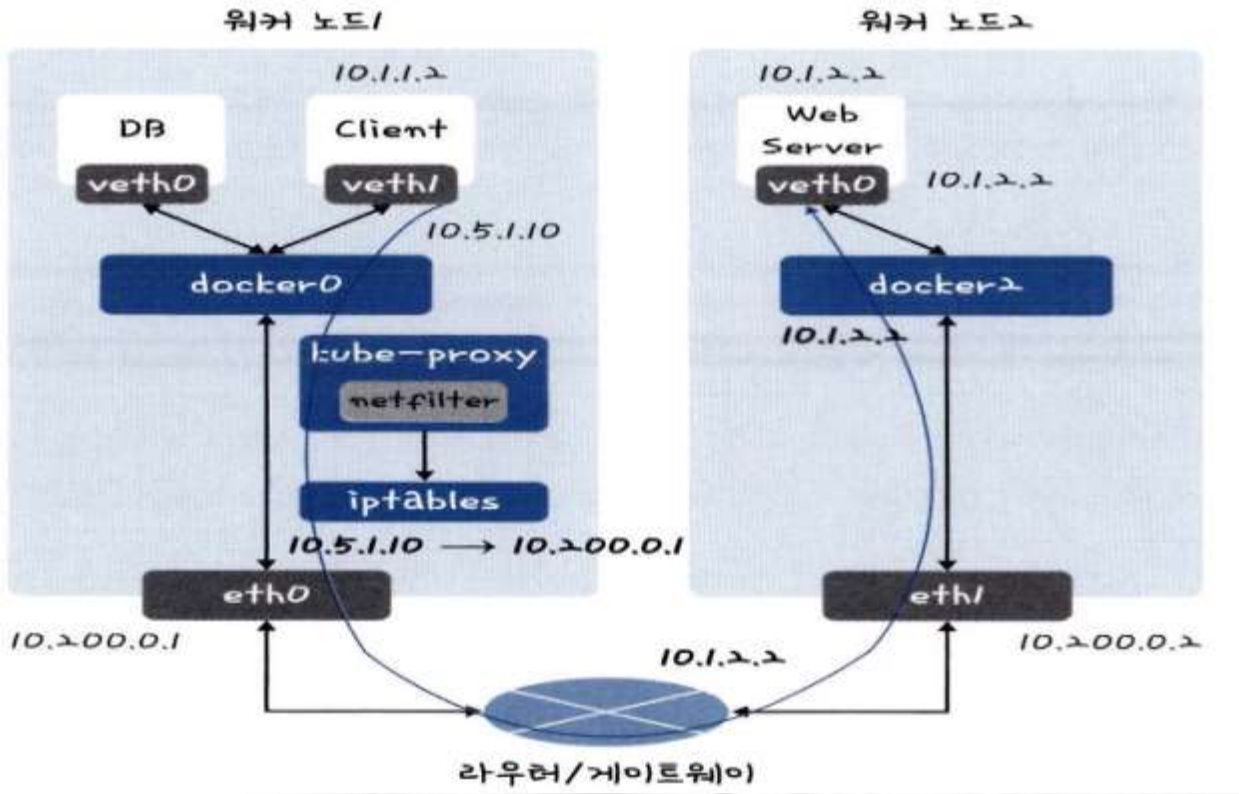


Kubernetes Communication

- Pod 와 Service 간 통신
  - ✓ Client Pod에서 Service Name(service-request)으로 http 요청을 하면 Cluster DNS 서버(Coredns)는 Service Name(service-request)에 해당하는 서비스 IP(10.5.1.10)를 Client Pod에 전달하고 Service IP를 전달받은 Client Pod는 해당 IP를 이용해 http 요청을 하는데 이때 Worker Node1이라는 호스트에서는 자신의 라우팅 테이블에 10.5.1.10 이라는 IP가 있는지 검색한 후 검색 결과 IP가 없으면 해당 요청을 라우터/게이트웨이로 전달하는데 다른 Worker Node에서도 10.5.1.10 이라는 IP는 찾을 수 없을 것(브리지나 라우팅 테이블에서 서비스 IP는 검색되지 않음)
  - ✓ 10.5.1.10이라는 서비스 IP로 Web Server Pod(IP: 10.1.2.2)를 찾아갈 수 있도록 해주는 것은 리눅스 커널에서 제공하는 netfilter와 iptables
  - ✓ netfilter는 규칙 기반의 패킷 처리 엔진으로 서비스 IP를 특정 IP로 변경할 수 있는 규칙을 저장하고 변환하는 역할을 하는데 netfilter를 Proxy라고도 함
  - ✓ netfilter에서는 서비스 IP(10.5.1.10)를 만나면 서버 IP(10.200.0.1)로 변환하고 라우터/게이트웨이로 패킷을 전달하는데 동일한 작업이 Worker Node2에서도 진행되고 이후에는 라우터/게이트웨이를 거쳐서 Worker Node2의 Web Server Pod에 접근할 수 있음

Kubernetes Communication

- Pod 와 Service 간 통신
  - ✓ netfilter



Kubernetes Communication

- 외부와 통신할 수 있게 하는 서비스 유형
  - ✓ Node Port: Node Port는 Node IP(eth)에 포트를 붙여서 외부에 노출시키는 것을 의미하는데 노드의 IP가 192.168.10.2라면 포트를 붙여서 192.168.10.2:30001 같은 형태를 만들어서 외부에 노출
  - ✓ Load Balancer: AWS, Azure, GCP과 같은 Public Cloud에서 제공하는 Load Balancer 와 Pod를 연결한 후 해당 Load Balancer IP를 이용해 Cluster 외부에서 Pod에 접근할 수 있도록 해줌
  - ✓ Ingress: Cluster 외부에서 내부로 접근하는 요청들을 어떻게 처리할지에 관한 규칙 모음으로 Ingress 자체는 Cluster 외부에서 URL로 접근할 수 있도록 Load Balancing, SSL 인증서 처리 등의 규칙을 정의한 것에 불과하며 실제로 동작시키는 것은 Ingress- Controller

Kubernetes Communication

❖ 외부와 통신할 수 있게 하는 서비스 유형