(예진) 인프라 환경 공부 - [강의] 초보를 위한 쿠버네티스 안내서

Done	☑
Do Date	@2023년 8월 1일

▼ [1] 컨테이너 오케스트레이션이란?

💡 어떻게 하면 **서버 관리를 쉽게** 할 수 있을까?

1. **문서화**를 잘 해보자~~ (화면 하나하나를 잘 캡처하자)

   → 문제: 문서 업데이트, 프로그램 버전 업데이트

2. **설정 관리 도구**의 등장! 문서보다는 코드지~ 스크립트 직접 입력 X

   → 문제: 설정 관리 도구 사용을 배워야 함. 서버를 복잡하게 관리하면, 도구의 사용법 난이도도 높아짐! 한 서버에서 여러 개의 버전 돌릴 수 X

3. **가상 머신** 등장! 가상 머신 띄우고, 프로그램 설치하면 문제가 발생할 일이 별로 X

   → 문제: 느림, 관리가 직관적 X, 클라우드 환경에 맞지 X, 특정 벤더에 dependency가 생김

   ✍🏽 벤더?
   vendor → 판매 회사
   (AWS 같이 서비스 판매 회사에서만 사용 가능하다는 뜻인듯!)

⇒ **도커** 등장!

- 모든 실행환경을 컨테이너로!
- 어디서든 동작
- 쉽고 효율적
- 느리지 X

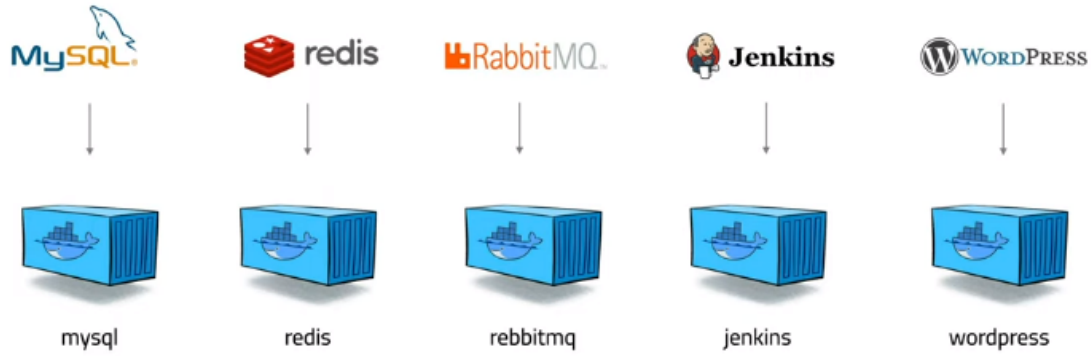
컨테이너의 특징

- 가상머신과 비교해 컨테이너 생성이 쉽고 효율적
- 컨테이너 이미지를 이용한 배포, 롤백이 간단

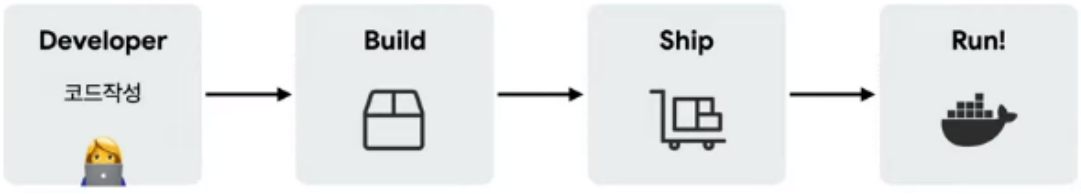
- 언어/프레임워크에 상관없이 애플리케이션을 동일한 방식으로 관리
- 개발, 테스팅, 운영 환경은 물론 로컬 pc와 클라우드까지 동일한 환경을 구축
- 오픈소스이며, 특정 클라우드 벤더에 종속적이지 않음

도커의 등장

- **containerization**: 프로그램을 컨테이너화해서 사용함



- **프로그램 개발의 과정이 정형화**됨

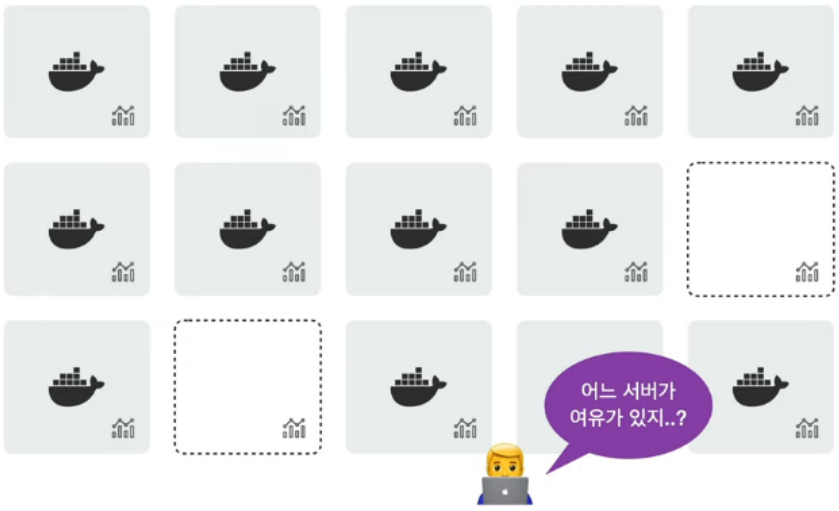


→ 수십, 수백개로 늘어난 **컨테이너를 어떻게 관리**해야 하지?

도커의 문제점?

1. **배포(deployment)는 어떻게?**
    - IP도 하나하나 관리 필요
    - 접속 및 실행도 하나하나 해야 함~

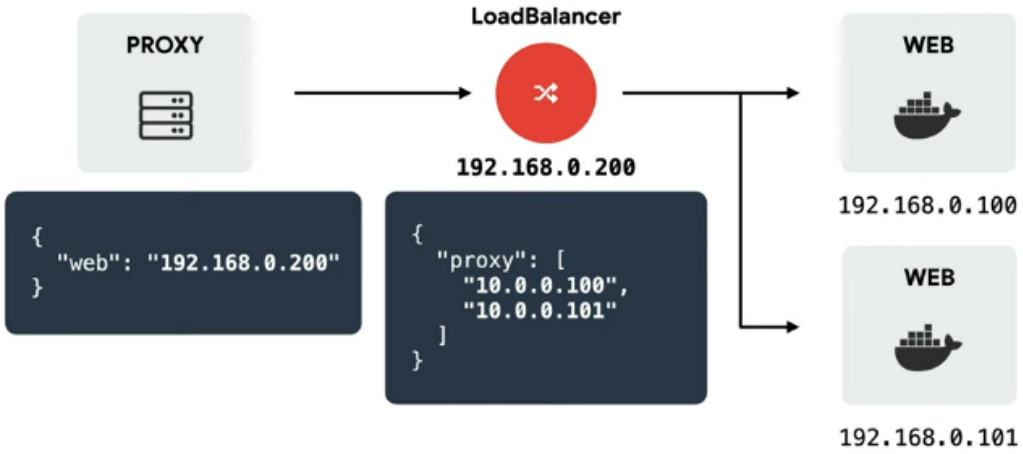
    - 도커가 많아지면, 빈 공간이 생김
        - 컨테이너가 실행되고 있지 않은 서버
        - 빈 서버에 새로운 컨테이너를 실행시켜주는 것이 좋은데, 어떤 서버가 여유 있는지 알 수 X



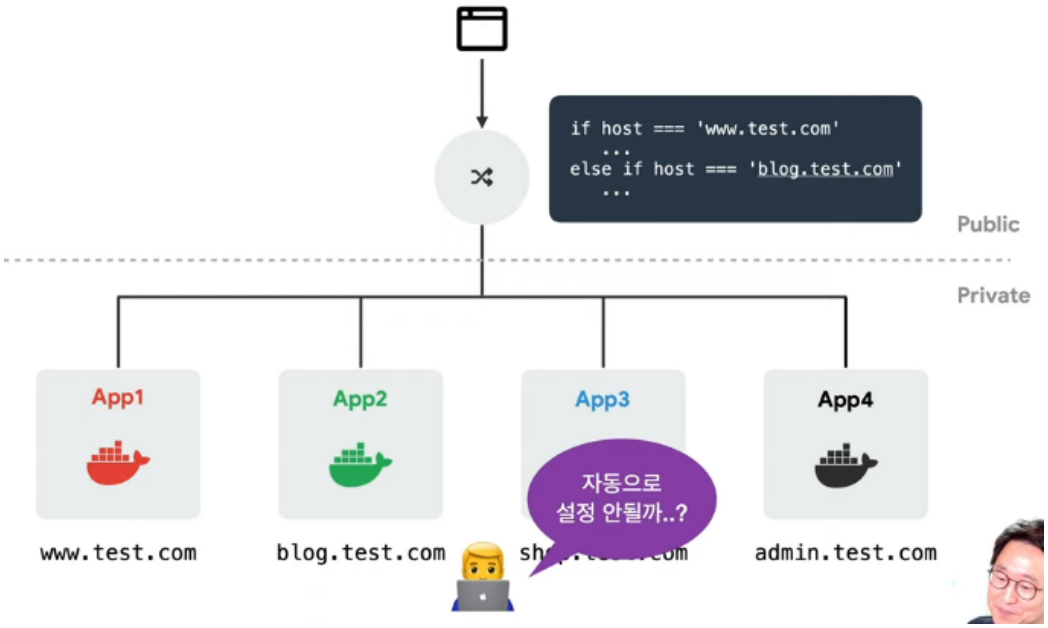
    - 롤아웃/롤백이 번거로움. 중앙에서 한번에 할 수 X

2. **서비스 검색(service discovery)은 어떻게?**
    - 로드밸런서 설치해서, 프록시는 로드밸런서를 바라보게 함.
        - 로드 밸런서에 요청이 들어오면 부하 분산시켜줌

          → 내부 서비스에 통신이 많아짐. ip 바뀔 때마다 업데이트 해주는 등 서비스 검색 필요



3. **서비스 노출(gateway)은 어떻게?**
    - 내부에 있는 서비스를 외부로 노출
    - proxy 서버를 하나 두고, 들어오는 host에 따라 내부 서비스 연결



→ 자동으로 설정할 수 없을까?

4. **서비스 이상, 부하 모니터링**은 어떻게?

컨테이너 오케스트레이션 (Container Orchestration)

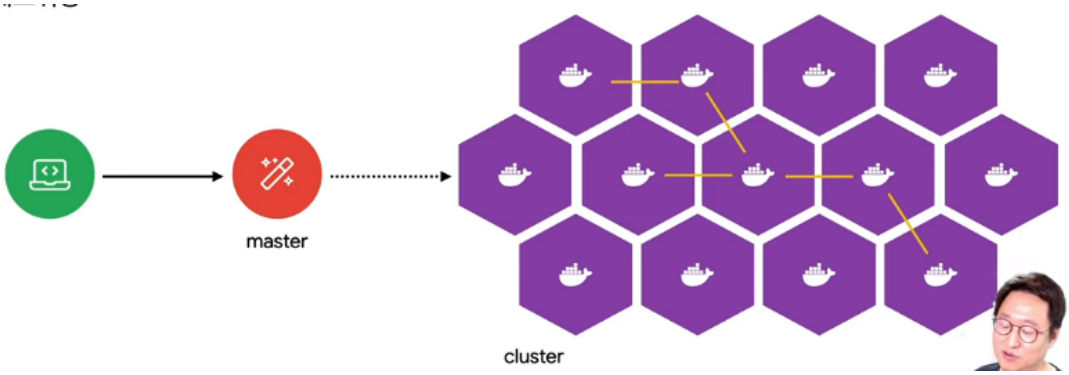
→ **복잡한 컨테이너 환경을 효과적으로 관리**하기 위한 도구

특징

- **클러스터의 특징**

- 기존에 서버가 적을 때는 하나하나 관리했음

  → 점점 많은 서버가 생김



  - (**중앙제어** (master-node)) - cluster 단위로 추상화하여 관리, master 서버를 두고 관리함
  - (**네트워킹**) - 클러스터 내 노드들끼리 네트워크 통신
  - (**노드 스케일**) - 노드의 수가 엄청 많아지더라도 잘 돌아가야 함

- (**상태 관리**) - 원하는 상태를 작성하면, 직접 조치하지 않고도 컨테이너 오케스트레이션이 자동으로 상태를 맞춰줌
- (**배포 관리**) - 서버의 상태를 자동으로 체크해주는 스케줄링 기능
- (**배포 버전 관리**) - 중앙에서 배포한 버전의 관리 가능
- (**서비스 등록 및 조회**) - 등록하면, 저장소에 ip 저장, proxy 서버가 저장소를 계속 관찰 → 설정 변경, 프로세스 재시작

- (**볼륨 스토리지**) - 노드에 필요한 볼륨(스토리지)를 설정하여 자동으로 관리

▼ [2] 왜 쿠버네티스인가?

쿠버네티스 (kubernetes)

→ 컨테이너를 **쉽고 빠르게 배포/확장**하고 **관리를 자동화**해주는 오픈소스 플랫폼

- 특징
  - 1주일에 20억개의 컨테이너를 생성하는 구글이 컨테이너 배포 시스템으로 사용하던 borg를 기반으로 만든 **오픈소스**
  - 운영에서 사용가능한 컨테이너 오케스트레이션
  - 구글(행성 스케일!)보다 적게 사용하면 OK
  - 다양한 요구사항을 만족시킬 수 있는 유연함
  - 어디서나 동작

- 특히 왜 인기가 많을까?
  - **오픈소스**
    - 오픈소스에 참여하는 기업들이 google, red hat, huawai, vmware, microsoft, ibm, intel, … 등 매우 쟁쟁한 업체가 참여
    - 커뮤니티가 매우 발달됨 (전세계 150개가 넘는 모임, 활발한 활동)
  - **엄청난 인기**
  - **무한한 확장성**
    - 쿠버네티스 위에서 머신러닝, CI/CD, 서비스메시, 서버리스 등이 동작가능함
  - **사실상의 표준 (de facto), Cloud Native의 핵심 역할**

    ✍🏻 **de facto**: **사실상**의 의미로 쓰이는 표현으로, 법적으로 공인된 사항이 아니더라도 실제 존재하는 사례를 가리키는 말

    - 완전히 새롭게 만들어야 하는 것 X. 쿠버네티스 위에 본인들의 장점을 커스터마이징해서 사용!

Rancher (by SUSE)

Red Hat OpenShift (by IBM)

Tanzu (by VMware)

    - 도커에서도 쿠버네티스 지원
    - 전세계 클라우드의 3대장(?) 전부 쿠버네티스를 managed service로 사용하고 있음

EKS

Amazon
Elastic Kubernetes Service

AKS

Azure
Kubernetes Service

GKE

Google
Kubernetes Engine

Cloud Native computing Foundation (CNCF)

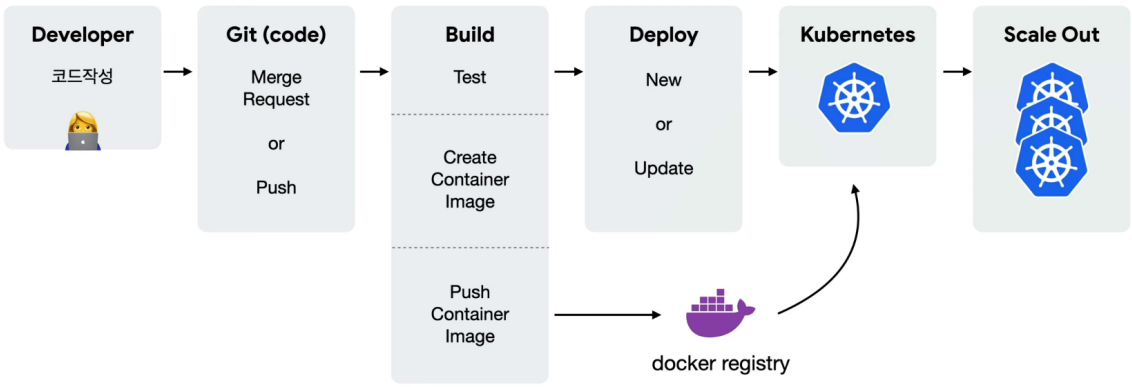
→ 클라우드 환경에 적합한 컴퓨팅 기술을 지원하는 오픈소스 단체.

리눅스 foundation의 소속

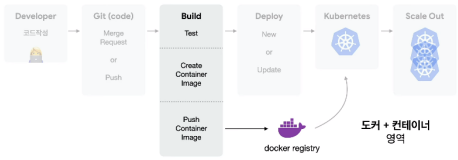
- 컨테이너, CI/CD, 오케스트레이션, 모니터링, 서비스 디스커버리, 네트워크 & 보안, 분산 DB & 저장소, 메시징, 컨테이너 런타임, 배포(인증) 등의 서비스를 제공함
- 위 서비스의 핵심역할을 쿠버네티스가 하고 있음!

▼ [3] 어떤걸 배울까?

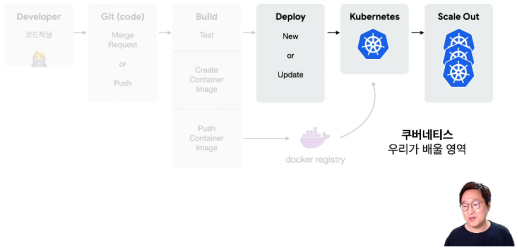
배포 과정



- 선행 공부가 필요한 영역 (도커+컨테이너)



- 수업에서 배울 영역



학습 범위

- **도커 컨테이너 실행**하기
  - 도커와 도커컴포즈를 이용한 멀티 컨테이너 관리
- **쿠버네티스에 컨테이너 배포**하기
  - 실습(hands-on) 환경 만들기
  - kubectl(쿠버네티스에 명령을 보내는 클라이언트, kube control) 사용법
  - pod, deployment, service 등 기본 리소스 학습
- **외부 접속 설정**하기
  - 서비스 타입(Cluster IP, NodePort, LoadBalancer, Ingress 등) 학습
  - 서비스 디스커버리 학습
- **스케일 아웃** 하기
  - 부하에 따른 컨테이너 개수 조정
  - 최소 리소스 요청 설정

- 오토스케일링
- **그 외 고급기능** 소개
  - HELM 패키지 매니저 소개
  - GitPos, ServiceMesh 소개

다루지 않는 범위

- 다양한 환경별 특징 (bar metal, EKS, ..)
- 쿠버네티스 패턴 (사이드카, 어댑터, …)
- 관련 생태계 (서비스메시, 서버리스, …)
- GitOps CI/CD
- 승인제어 등 고급 기능

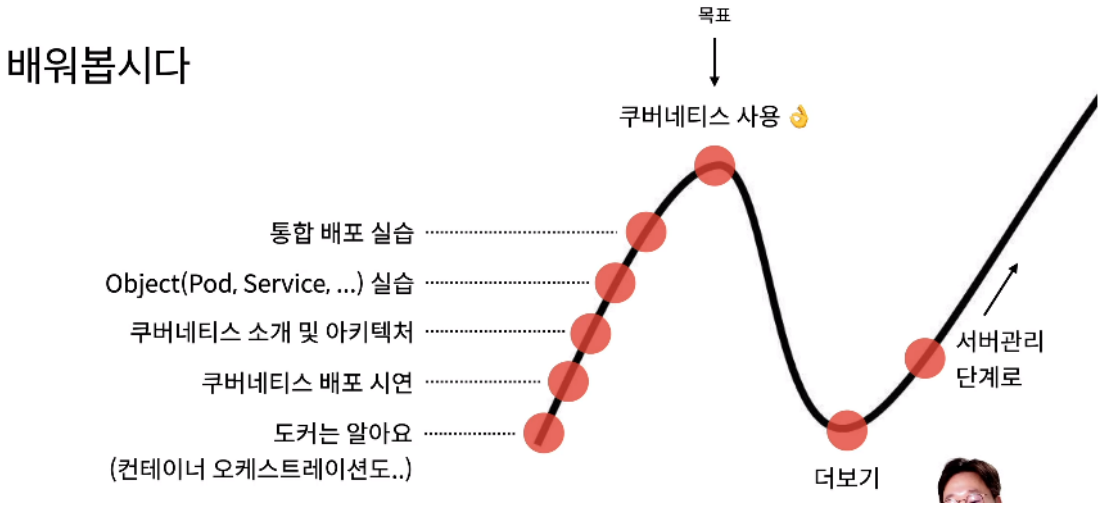
▼ [4] 쿠버네티스 소개

- 발음 정리

*의미가 통하는게 중요!

용어	발음
master	**마스터**
node	**노드** (구 minion 미니언)
k8s	**쿠버네티스**, 케이에잇츠, 케이팔에스
kubectl	**큐브 컨트롤**, 큐브 시티엘, 큐브커들
etcd	**엣지디**, 엣시디, 이티시디
pod	**팟**, 파드, 포드
istio	**이스티오**
helm	**헬름**, 햄, 햄
knative	**케이 네이티브**

- 우리의 목표!



쿠버네티스

- **자동화** : 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리
- **논리적인 단위** : 컨테이너를 쉽게 관리하고, 연결하기 위해 논리적인 단위로 그룹화
- **노하우**: google에서 15년간 경험을 토대로 최상의 아이디어와 방법들을 결합
- 그리스어로 조타수/조종사에서 따옴
- **k8**(ubernete가 8글자)**s**, **kube**(쿠베/큐브)라고 부르기도 함
- CNCF를 졸업한 프로젝트~!

ClundNative

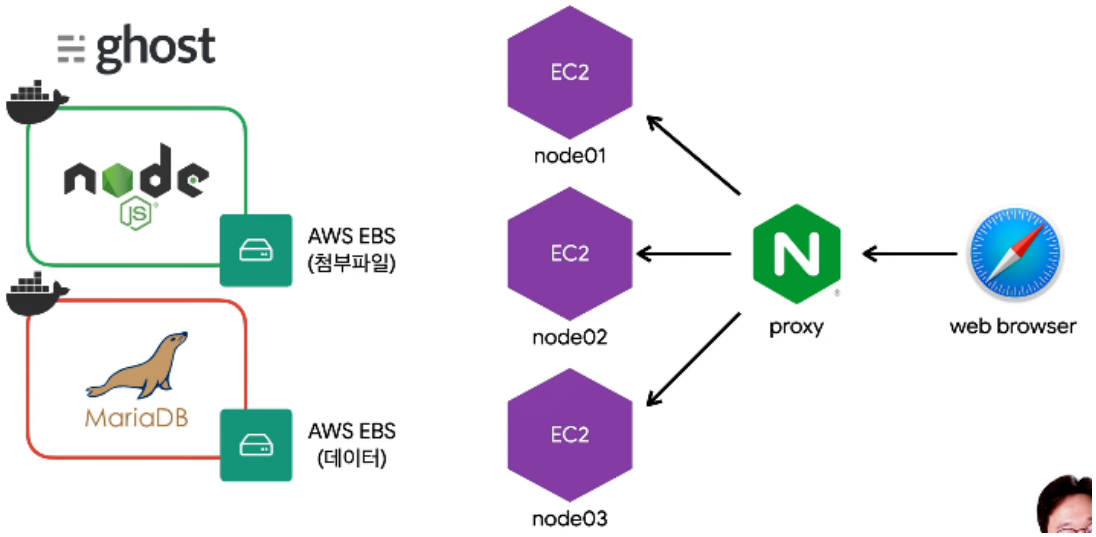
- 클라우드 이전에는, 리소스를 한 땀 한 땀 직접 관리했음!
- 클라우드 이후, 수많은 리소스를 자유롭게 사용하고, 추상적으로 관리!

클라우드 환경에서 **어떻게 애플리케이션을 배포**하는게 좋은걸까?

→ **이러한 방법들이 cloud native!** 하다고 말할 수 있음~
→ **쿠버네티스 위에서 이런 개념들을 구현하기 쉬움!**
⇒ **쿠버네티스는 클라우드에 어울리는 배포시스템**이다!

쿠버네티스 데모

- AWS 클라우드에 Ghost 블로그 배포

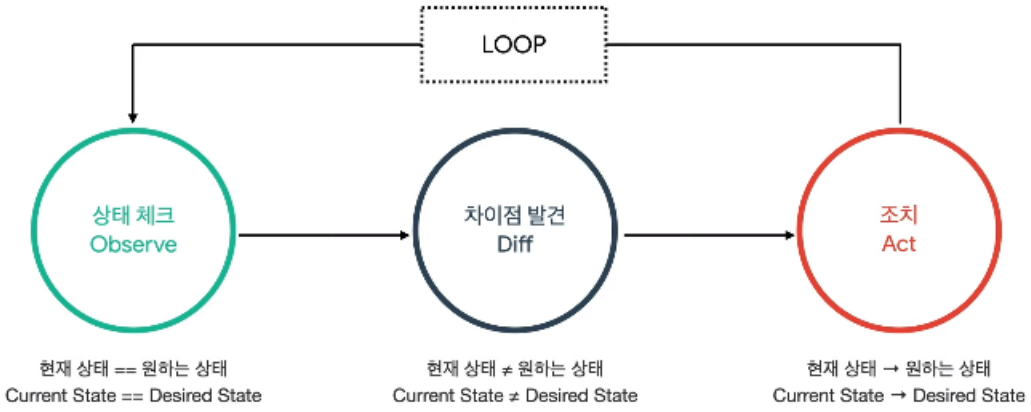


[5] 쿠버네티스를 이용한 배포 데모

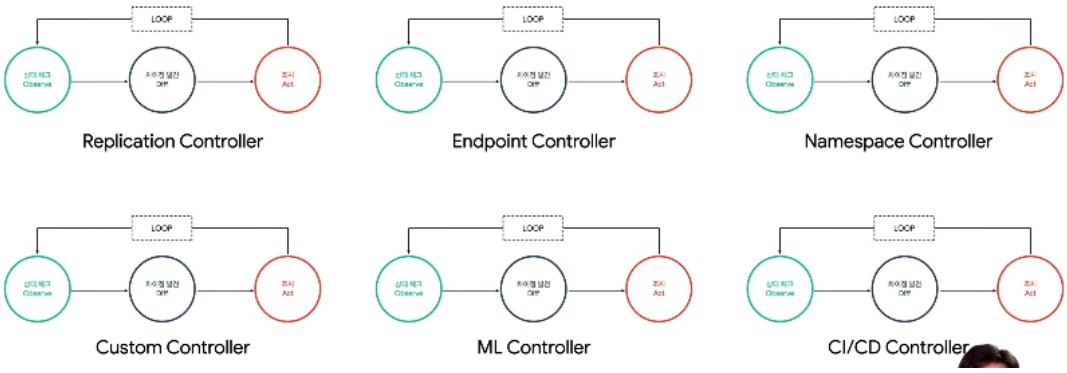
▼ [6] 쿠버네티스 아키텍처 1/3 (구성/설계)

Desired State (중요한 개념!)

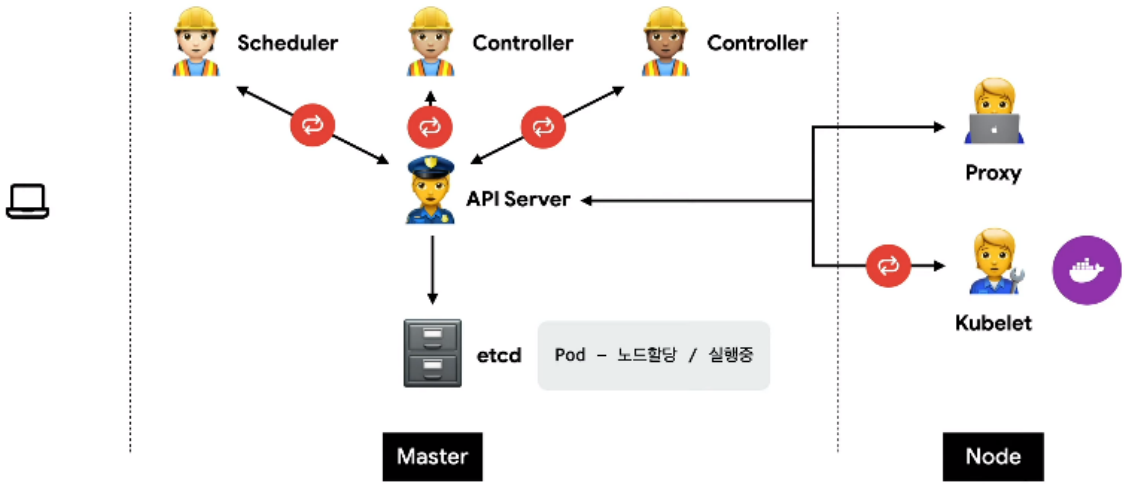
- 쿠버네티스의 근간을 아우르는 시스템



- 원하는 상태를 체크하는 컨트롤러가 계속 필요…



쿠버네티스 아키텍처



- **Master**

  : 체크, 실행하는 영역

- **Node**

  : 실제로 컨테이너가 실행되는 부분

- **API Server**

  : 중앙에서 교통정리 (조회/요청은 모두 API Server를 통함!)

  - 상태를 바꾸거나 조회
  - etcd와 유일하게 통신하는 모듈
  - REST API 형태로 제공
  - 권한을 체크하여 적절한 권한이 없을 경우 요청을 차단
  - 관리자 요청 뿐 아니라 다양한 내부 모듈과 통신
  - 수평으로 확장되도록 디자인

- **Scheduler**

  : 어떤 노드에 어떤 컨테이너를 연결할지 결정

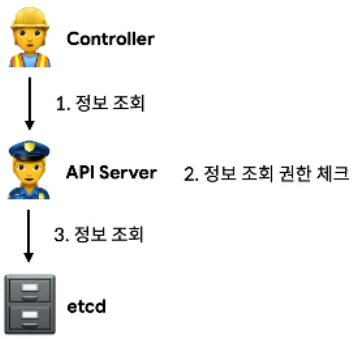
  - 새로 생성된 Pod를 감지하고 실행할 노드를 선택함
  - 노드의 현재 상태와 Pod의 요구사항을 체크
    - 노드에 라벨을 부여
    - ex) a-zone, b-zone 또는 gpu-enabled, …

- **Controller**

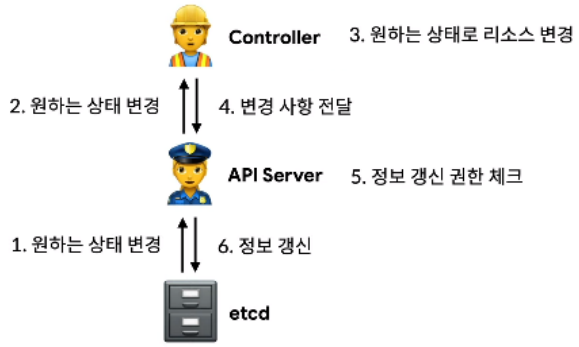
    - 논리적으로 다양한 컨트롤러가 존재
        - 복제 컨트롤러, 노드 컨트롤러, 엔드포엔트 컨트롤러, …
    - 끊임 없이 상태를 체크하고 원하는 상태를 유지
    - 복잡성을 낮추기 위해 하나의 프로세스로 실행
- **etcd**

  : 상태 저장 및 조회하는 데이터베이스
    - 모든 데이터를 확실하게 관리
    - 모든 상태와 데이터를 저장
    - 분산 시스템으로 구성하여 안전성을 높임 (고가용성) (날아가면 큰일남!!)
    - 가볍고 빠르면서 정확하게 설계 (일관성)
    - Key(directory)-Value 형태로 데이터 저장
    - TTL(time to live), watch 같은 부가 기능 제공
    - 백업은 필수!
- **Addons**
    - (CNI, DNS는 필수적인 부분이긴 함!)
    - CNI (네트워크)
    - DNS (도메인, 서비스 디스커버리)
    - 대시보드 (시각화)

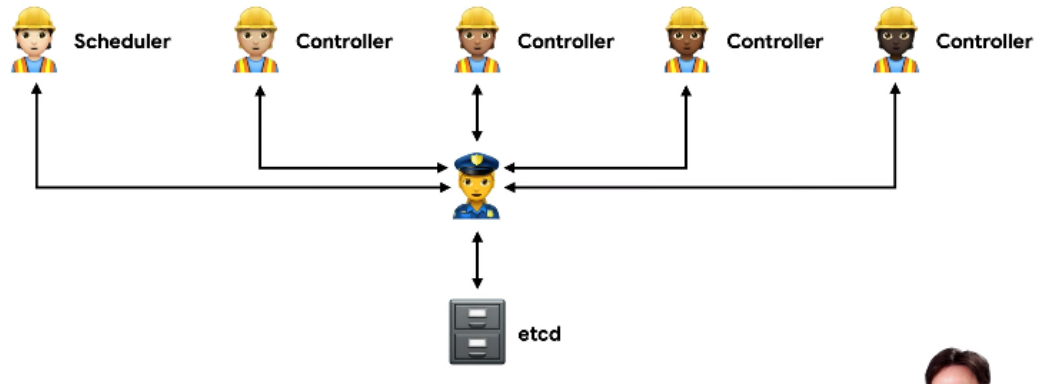
- **조회 흐름**



- **기본 흐름**

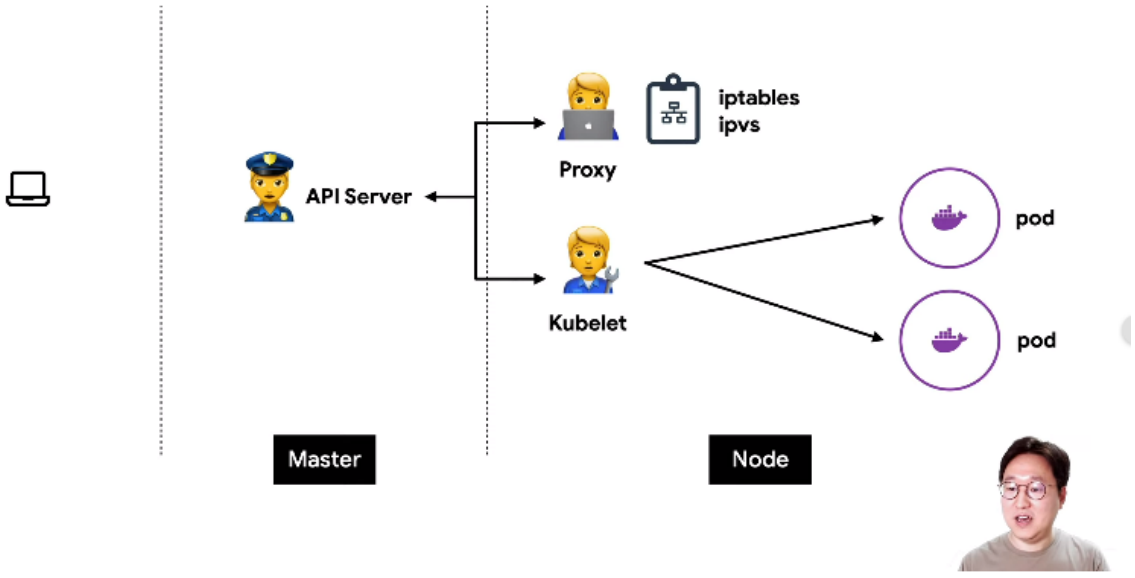


- **API Server 통신**



  - 내부 컴포넌트들이 etcd와 직접적으로 통신 X
  - API Server를 거쳐야만 etcd와 통신할 수 있음

쿠버네티스 노드



- **kubelet**

  : 컨테이너 관리 확실하게!

  - 각 노드에서 실행

- Pod를 실행/중지하고 상태를 체크
- CRI (Container Runtime Interface)
  - docker, Containerd, CRI-O, …

- **proxy**

  : 내/외부 통신 설정

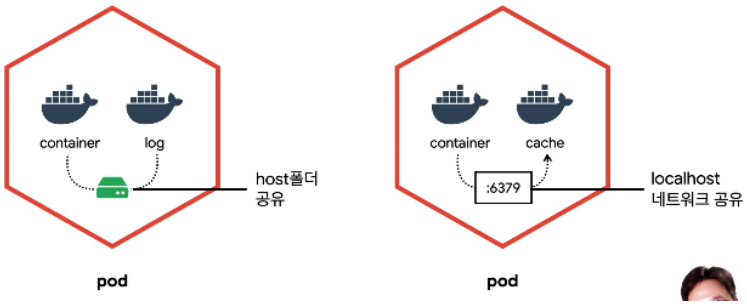
  - 네트워크 프록시와 부하 분산 역할
  - 성능상의 이유로 별도의 프록시 프로그램 대신 iptables 또는 IPVS를 사용 (설정만 관리)

▼ [7] 쿠버네티스 아키텍처 2/3 (오브젝트)

Pod

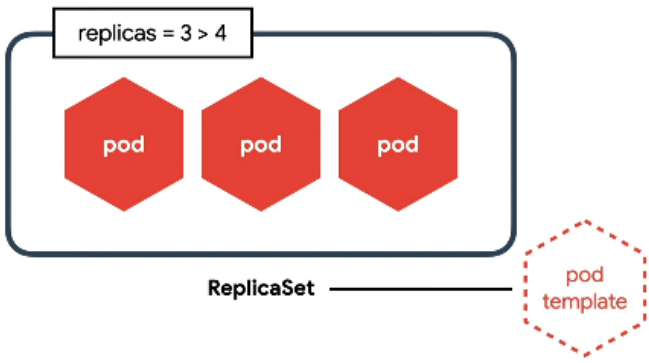
- 가장 작은 배포 단위
- 컨테이너를 관리하는 단위
- 전체 클러스터에서 고유한 IP를 부여 받음
- 여러 개의 컨테이너가 하나의 Pod에 속할 수 있음

  ex)



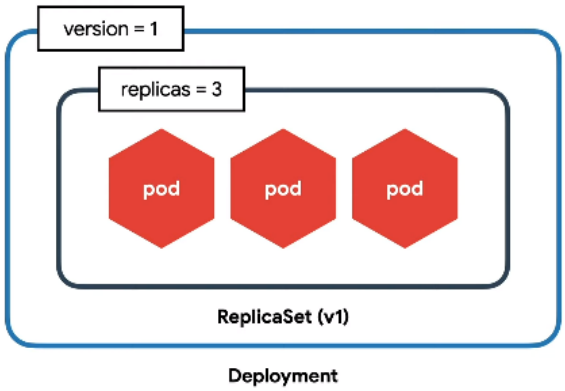
ReplicaSet

- 여러개의 Pod를 관리
- 몇 개의 Pod를 관리할지 설정
- 새로운 Pod은 Template을 참고해 생성

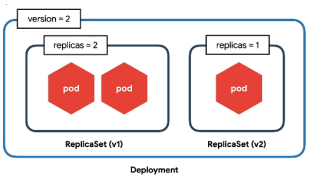


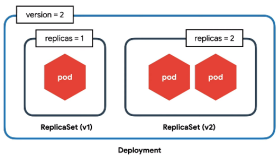
Deployment

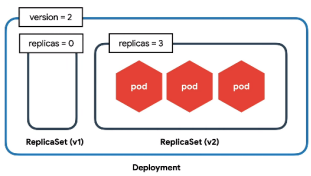
- 배포 버전을 관리



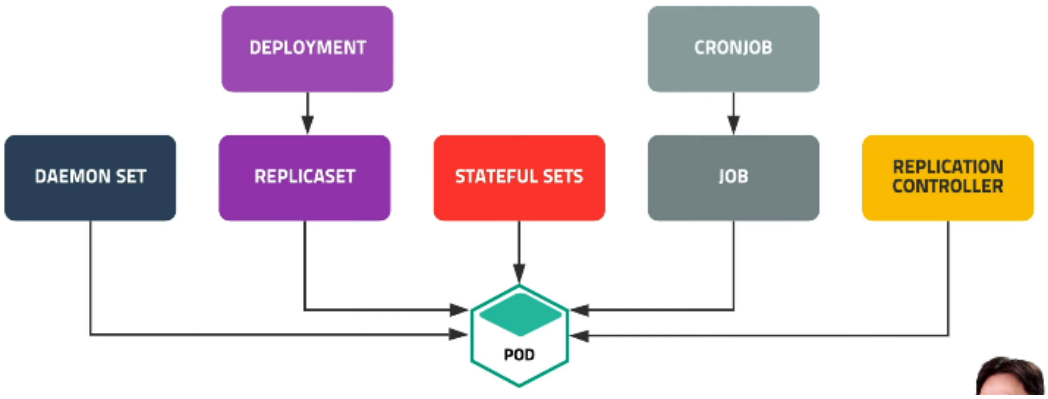
  - 내부적으로 ReplicaSet을 이용







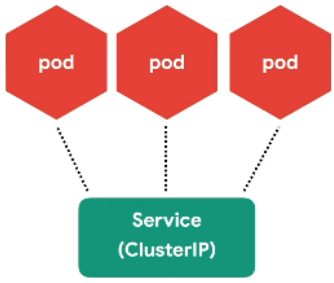
다양한 Workload



쿠버네티스 오브젝트

Service - ClusterIP

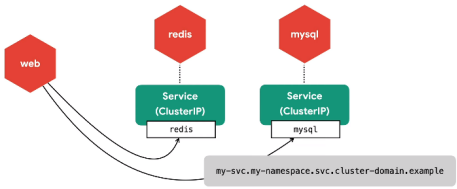
- 클러스터 내부에서 사용하는 프록시

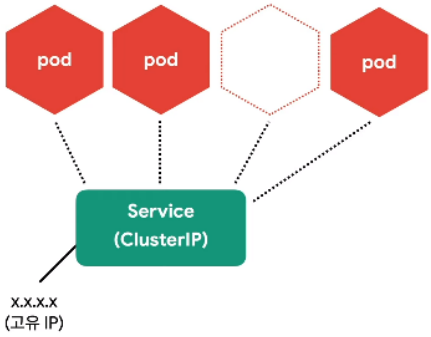


- Pod은 동적이지만 서비스는 고유 IP를 가짐

  → 서비스에 요청을 보내면, 원하는 pod에 정상적으로 요청을 보낼 수 있게 됨

- 클러스터 내부에서 서비스 연결은 DNS를 이용



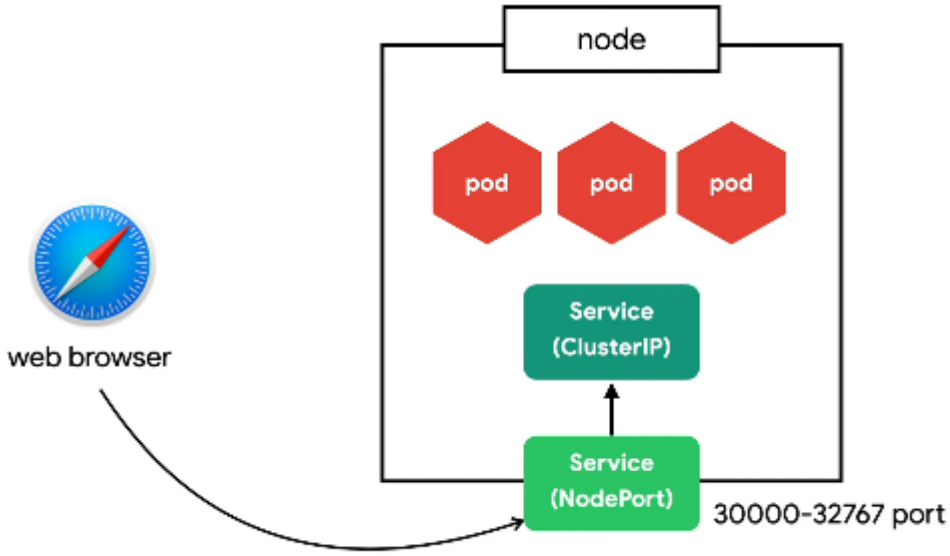


💡 문제점

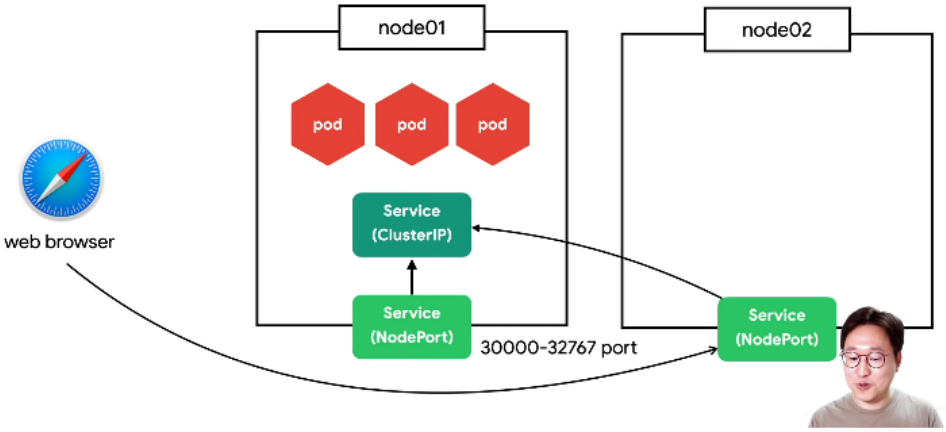
→ **내부에서만 통신 가능! 외부 브라우저와는 통신 불가능!**

Service - NodePort

- 노드(host)에 노출되어 **외부에서 접근 가능한 서비스**



- 모든 노드에 동일한 포트로 생성

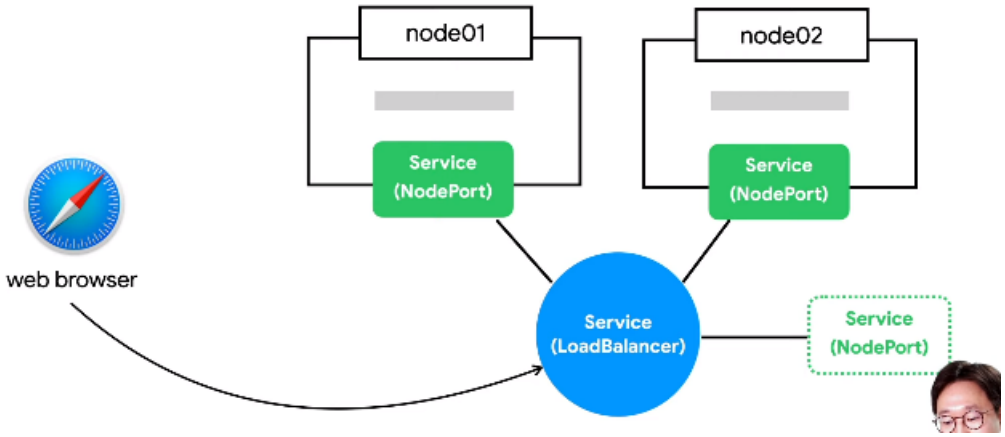


💡 문제점

→ **서버 자체가 사라지게 됐을 때, 노드에 접근하면 접속 X**

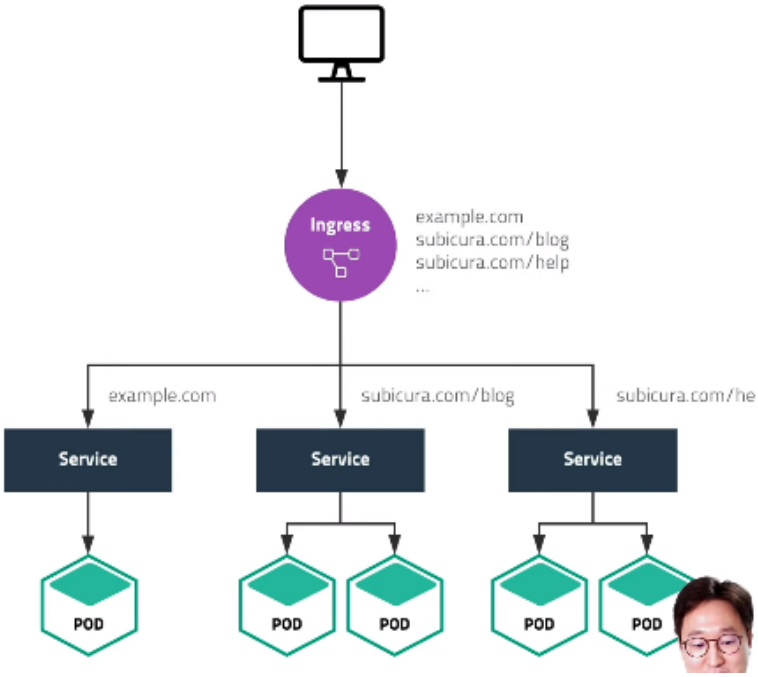
Service - LoadBalancer

- 하나의 IP 주소를 외부에 노출
  - 사용자가 loadbalancer에 요청 → 요청이 nodeport에 전달 → clusterIP → pod에 요청 전달



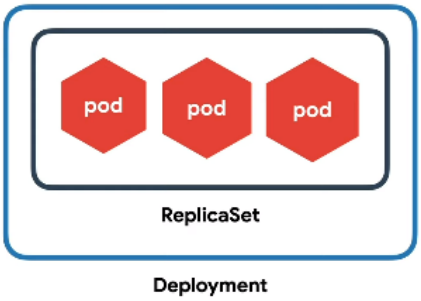
Ingress

- 도메인 또는 경로별 라우팅
  - Nginx, HAProxy, ALB…



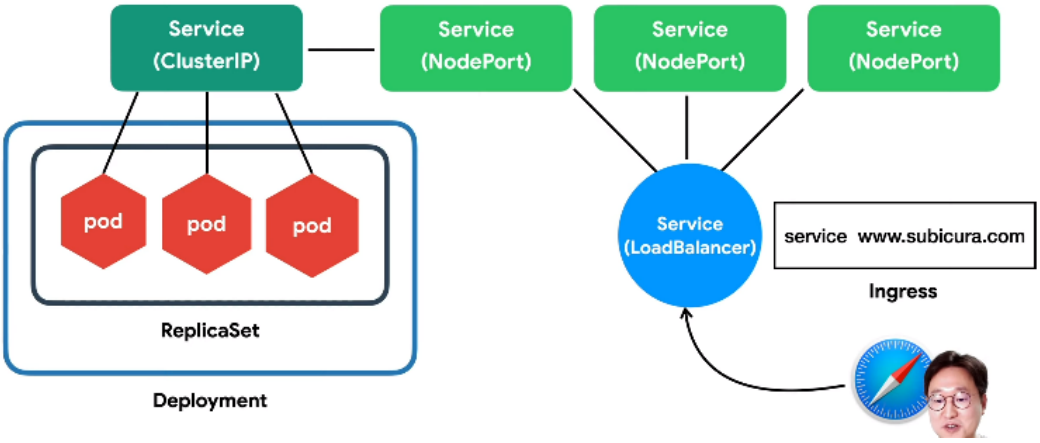
일반적인 구성

- **Deployment** 생성 → 자동으로 Deployment가 **ReplicaSet** 생성 → 자동으로 ReplicaSet이 **pod** 생성



- 외부에 노출시키기 위해 **Service**를 붙임 → **Ingress**도 붙임 → 자동으로 **NodePort**, **LoadBalancer**도 생성됨

  ⇒ **클라이언트는 도메인을 통해 pod에 연결** 가능!



그 외 기본 오브젝트

- **Volume** - Storage (EBS, NFS, …)
- **Namespace** - 논리적인 리소스 구분
- **ConfigMap/Secret** - 설정
- **ServiceAccount** - 권한 계정
- **Role/ClusterRole** - 권한 설정 (get, list, watch, create, …)
- …

▼ [8] 쿠버네티스 아키텍처 3/3 (API 호출)

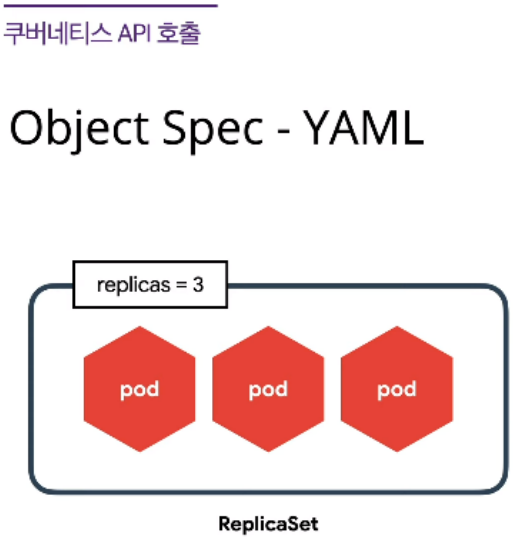
Object Spec - YAML

- ex 1) Pod 생성
  - YAML로 명세 작성 → API Server가 명세를 보고 dtcd에 정보 저장, 각 컨트롤러들 동작

pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  containers:
  - name: busybox
    image: busybox:1.25
```

- ex 2) ReplicaSet 생성




```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: web
        image: image:v1
```

- ex 3) ArgoCD(Custom Resource) 생성

쿠버네티스 API 호출

Object Spec - YAML

ArgoCD (Custom Resource)

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL:
https://github.com/argoproj/argocd-exa
mple-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    server:
https://kubernetes.default.svc
    namespace: guestbook
```

Object Spec

- **apiVersion**
  - apps/v1, v1, batch/v1, networking.k8s.io/v1, …
- **kind**
  - Pod, Deployment, Service, Ingress, …
- **metadata**
  - name, label, namespace, …
- **spec**
  - 각종 설정 (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18)

    → 현재 버전 바뀌어서 안되는 듯 (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.27/)
- **status (read-only)**
  - 시스템에서 관리하는 최신 상태

API 호출하기

→ **원하는 상태(desired state)**를

→ **다양한 오브젝트(object)**로

→ **정의(spec)**하고,

→ **API 서버에 yaml 형식으로 전달**