

쿠버네티스가 바꾼 IT 운영 방식

서버를 사람이 운영하던 시대에서 인프라가 스스로 운영되는 시대로
컨테이너를 처음 배우는 사람을 위한 개념 설명

개발팀이 운영 환경을 몰라도 되게 만든 것이 무엇인가

PROJECT CODE

K8S-INTRO-2026

DATE

2026.09

오늘 이야기는 두 갈래

왜 필요해졌나 · 무엇이 달라지나

PART 1 · 왜 쿠버네티스가 필요해졌나

슬라이드 04 – 12

사람 손으로 버티던 시절과 그 한계

01

손으로 만들던 서버 운영

이용률 10%, 재현되지 않는 환경

02

사람이 만질 수 없는 규모

구글 Borg 에서 나온 컨테이너 표준

03

VM 과 컨테이너의 차이

설비를 각자 갖추느냐, 함께 쓰느냐

묶어 보면 — 이런 도구가 왜 나왔는지

PART 2 · 무엇이 달라지나

슬라이드 13 – 21

도입한 뒤 운영과 개발이 바뀌는 지점

04

스스로 되돌리는 제어 루프

차이가 생기면 컨트롤러가 다시 맞춤

05

코드로 옮겨 간 일상 업무

배포·장애 대응·증설, 그리고 골든 패스

핵심 한 줄 · 사람은 원하는 상태만 적고, 유지는 플랫폼이 담당

묶어 보면 — 도입하면 무엇이 편해지는지

KEY FOCUS

두 갈래를 잇는 질문 하나 — 개발팀이 운영 환경을 몰라도 되게 만든 것이 무엇인가

PART 01

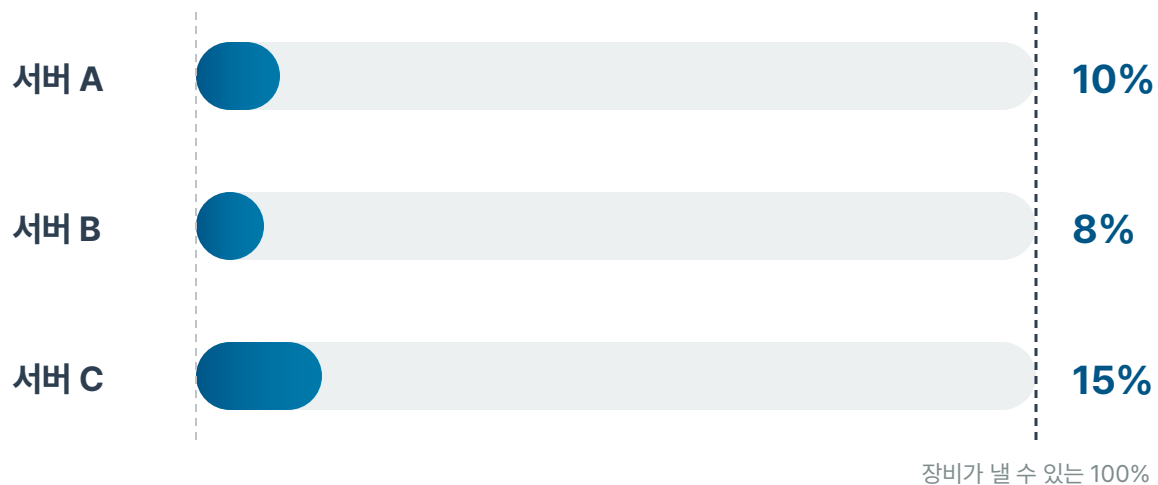
왜 쿠버네티스가 필요해졌나

손으로 만들던 서버 운영 · 사람이 만질 수 없는 규모 · VM 과 컨테이너의 차이

KEY INSIGHT

서버를 나눠 쓰지 못한 원인은 성능이 아닌 격리 — 한 앱의 라이브러리 변경이 옆 앱을 멈추게 해 아예 따로 사는 편이 안전했음

서버 한 대에서 실제로 쓰이는 용량



비어 있는 용량에 다른 앱을 올리지 못함

한쪽의 라이브러리 버전을 바꾸면 옆 앱이 멈춰, 아예 장비를 분리해 구매

이용률과 무관하게 나가는 비용

100% 청구

장비값 · 전기료 · 상면료

서버가 10%만 일해도 고지서는 그대로 도착



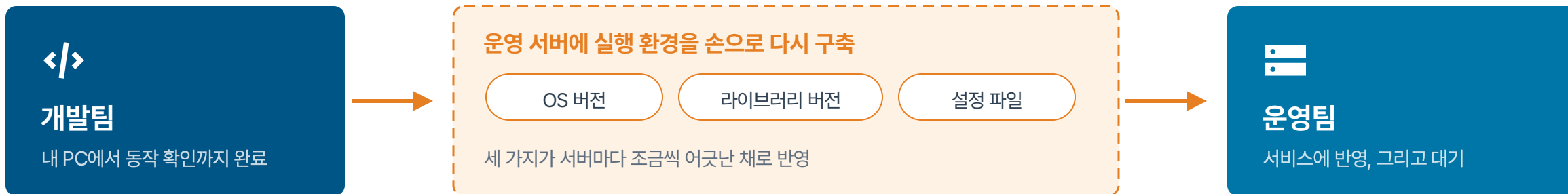
당시의 기본 규칙

서버 한 대에 애플리케이션 하나

안전을 위해 성능을 포기한 선택

KEY INSIGHT

"제 PC에서는 됩니다"의 원인은 개인 실수가 아닌 이관 구조 — 환경을 사람이 옮겨 적는 한 특정 서버에서만 나는 장애가 계속 생김



"제 PC에서는 됐는데요"

— 개인의 실수처럼 들리지만, 환경을 손으로 옮기는 구조가 만들어 낸 말



OS 버전이 갈림

개발은 최신 패치, 운영은
몇 달 전 상태로 고정



라이브러리가 섞임

같은 이름 다른 버전이
서버마다 다르게 설치



장애가 재현되지 않음

특정 서버에서만 발생해
원인 추적이 어려움

KEY INSIGHT

느린 것보다 사람에게 묶인 것이 더 큰 문제 — 배포·장애·증설이 전부 특정 담당자의 시간과 기억을 기다려야 진행됨

사람에게 얹혀 있던 일

그래서 현장에서 실제로 벌어진 일



서버마다 조금씩 다른 설정

어느 한 서버에서만 나는 장애 발생 — 재현도 원인 규명도 어려움



배포를 두려워함

주말 새벽 작업 · 롤백 계획서 · 전사 공지를 매번 준비



장애가 곧 사람 호출

새벽 3시에 담당자가 깨어 직접 서버에 접속해 복구



증설에 몇 주가 필요

장비 구매 · 입고 · 설치 · 세팅을 거치면 이벤트는 이미 종료



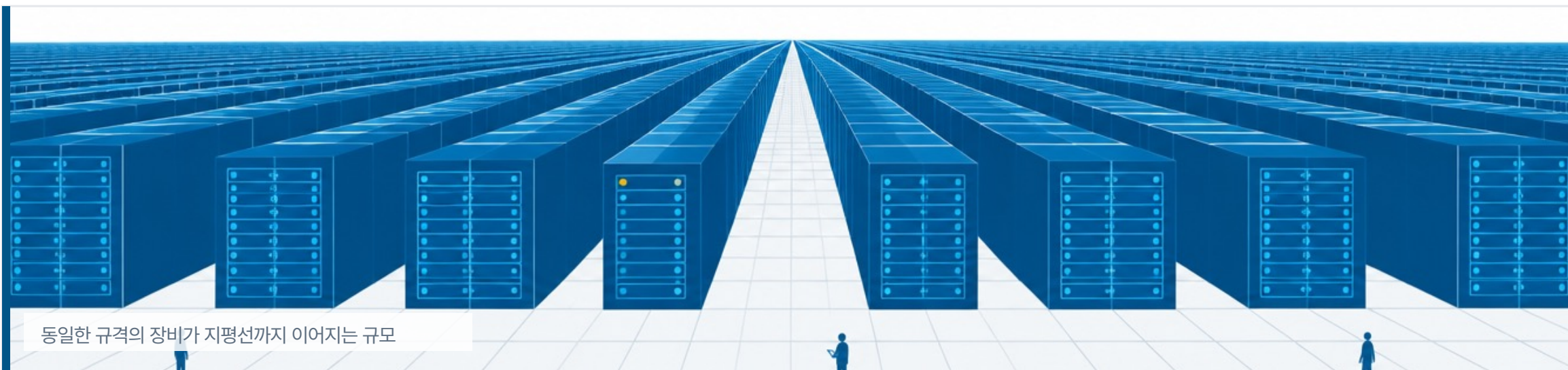
장비마다 이름을 붙여 관리

한 대가 멈추면 그 장비를 대체할 방법이 없어 서비스도 함께 중단

규모가 커질수록 사람의 기억과 시간이 먼저 한계에 도달

KEY INSIGHT

구글이 먼저 부딪힌 것은 속도보다 인원 — 수만 대에서 매일 생기는 고장을 사람 수로 따라갈 수 없어 기계에 넘겨야 했음



데이터센터 한 곳

수만 대

사람이 직접 만지는 것이 불가능

함께 돌던 애플리케이션

수천 종

종류마다 요구 환경이 제각각

주당 컨테이너 실행

20억 개

2014년 공개 시점의 발표 수치

Borg 가동 시작

2003~2004

쿠버네티스 공개보다 10년 앞섬

KEY INSIGHT

컨테이너를 고른 기준은 크기보다 조작 가능성 — 규격이 통일돼야 배치·복구·확장을 기계에 맡길 수 있음

01

한 대에 뽁뽁이 채우기 위해

Borg 논문이 적은 네 기법 — admission control · efficient task-packing · over-commitment · machine sharing
서버를 10%만 쓰던 시대를 끝내는 일이 첫 목적

02

고장을 전제로 설계하기 위해

수만 대 가운데 매일 몇 대는 반드시 멈추고, 사람의 손은 그 속도를 따라가지 못함
기계가 대신 복구하려면 무엇을 어떻게 실행할지가 기계가 읽을 수 있는 규격이어야 함

03

개발자를 인프라에서 떼어내기 위해

declarative job specification language(원하는 상태를 적어 내는 명세 언어)를 개발자에게 제공
자원 관리와 장애 처리는 그 아래로 감추고, 개발자는 앱만 보게 만들



구글이 컨테이너에서 본 가치

가벼운 가상 서버로 본 것이 아니라, 자동화가 시작될 수 있는 최소 규격으로 봄 — 규격이 통일돼야 기계가 사람 대신 다룸

KEY INSIGHT

기동이 빨라진 원인은 게스트 OS 한 벌이 빠진 것 — 커널을 빌려 쓰므로 실을 짐이 수 GB 에서 수백 MB 로 줄어듦

단독주택 단지 · VM



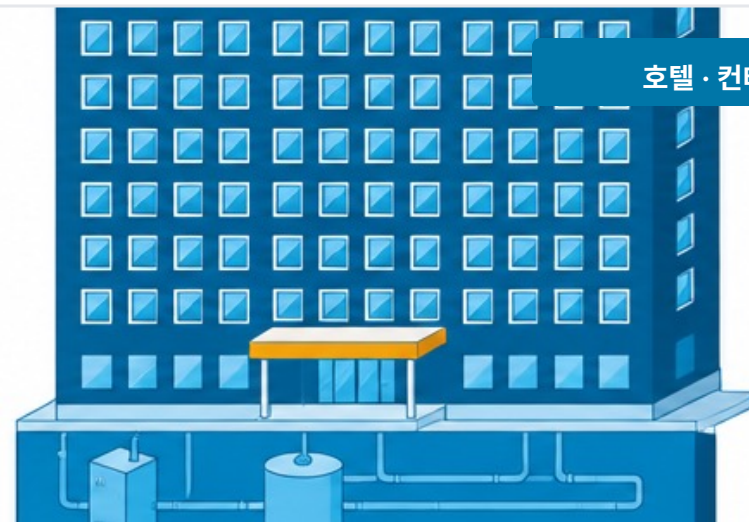
집마다 따로 갖춘 설비

기초 · 보일러 · 수도를 집집마다 중복해서 구비

→ 가상 머신마다 게스트 OS 한 벌을 통째로 탑재

땅은 넓은데 몇 채밖에 세우지 못하고, 수리도 집집마다 따로 부름

호텔 · 컨테이너



호텔이 한 번 갖춘 공용 설비

골조 · 배관 · 보일러 · 엘리베이터를 모든 객실이 공유

→ 컨테이너가 호스트 OS 커널 하나를 함께 사용

같은 땅에 객실 수백 개, 투숙객은 짐만 들고 와서 바로 체크인

KEY INSIGHT

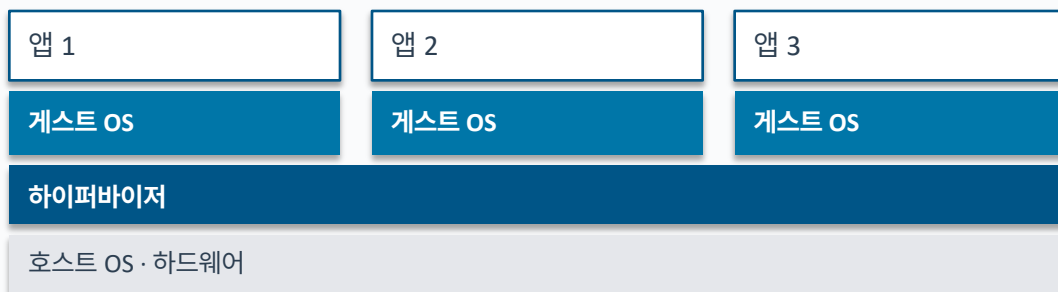
여섯 줄 가운데 운영 주체를 바꾸는 것은 설비 책임자 한 줄 — 나머지는 정도의 차이여서 도입 결정을 가르지 못함

비교 항목	VM · 단독주택	컨테이너 · 호텔
기초 · 설비	게스트 OS 를 통째로 따로 탑재	호스트 커널 하나를 공유
기동 시간	수십 초 ~ 수 분	수 밀리초 ~ 수 초
이미지 크기	수 GB ~ 수십 GB	수십 ~ 수백 MB
한 대당 밀도	몇 개	32GB 서버에 60개 이상
성능 손실	1~6% 오버헤드	거의 없음, 네이티브 수준
설비 책임자	앱을 올린 팀이 직접	플랫폼이 대신
이 한 줄이 운영 주체를 바꿈		

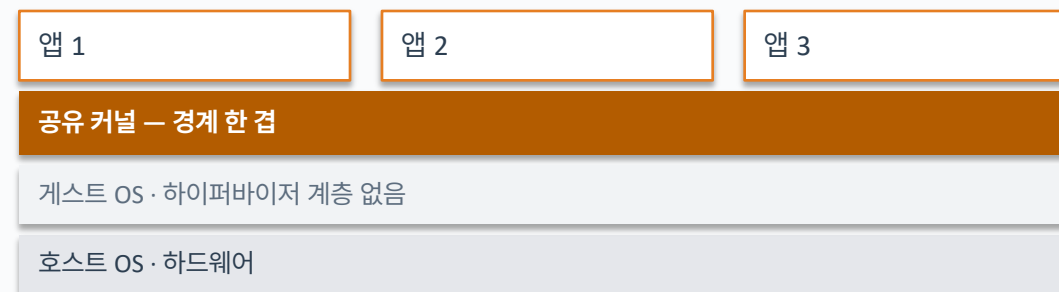
KEY INSIGHT

컨테이너가 VM 을 전부 대체하지는 못함 — 커널을 함께 쓰므로 규제·보안 요구가 강한 구간에서는 VM 과 병행이 현실적

VM · 게스트 OS 가 경계를 나눔



컨테이너 · 커널 하나를 함께 씬



VM 은 앱마다 게스트 OS 가 경계를 세우지만, 컨테이너는 커널 한 겹을 함께 써서 경계가 얇다

✓ 커널을 공유해서 얻는 것

- 같은 서버에 수십 개를 채우는 밀도
- 수 밀리초에서 수 초 사이로 끝나는 기동
- 커널과 OS 를 플랫폼이 떠안아 줄어든 확인 항목

⚠ 그 대가로 감수하는 것

- 커널을 함께 쓰므로 VM 보다 낮은 격리 수준
- 보안 요구가 강한 구간은 VM 과 병행하는 구성
- 관리 주체가 바뀐 것이지 관리가 사라진 것은 아님

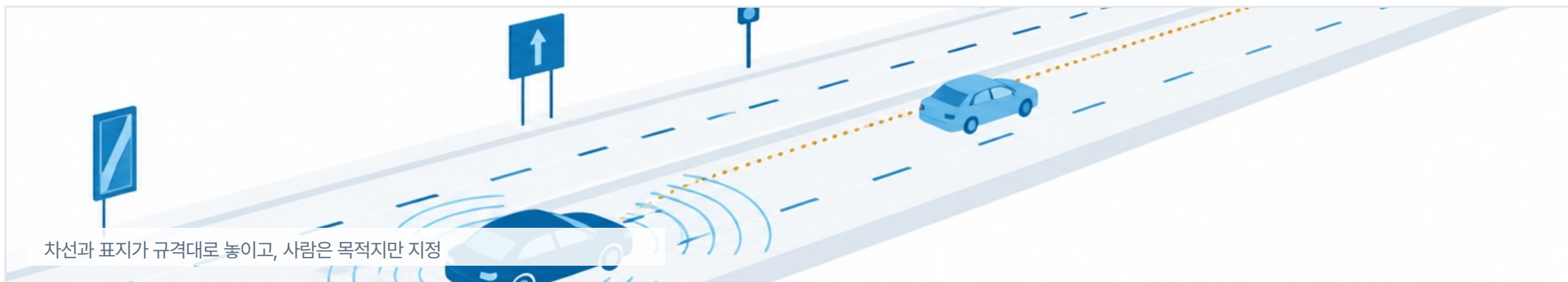
PART 02

무엇이 달라지나

스스로 되돌리는 제어 루프 · 코드로 옮겨 간 일상 업무 — 도입 뒤 운영과 개발이 바뀌는 지점

KEY INSIGHT

다섯 조건은 함께 있어야 의미 — 하나라도 빠지면 사람이 계속 지켜봐야 해서 자동화가 스크립트 수준에 머무름



무인 주행이 성립한 조건

도로 · 차선 · 신호가 규격화되어 있음

핸들과 브레이크를 전자적으로 조작

센서가 현재 상태를 항상 읽음

사람은 목적지만 지정

차이가 생기면 차가 스스로 조향

클러스터 자율 운영이 성립한 조건

01 컨테이너라는 실행 규격이 생김

02 모든 자원을 API 로 조작

03 클러스터 상태를 항상 관측

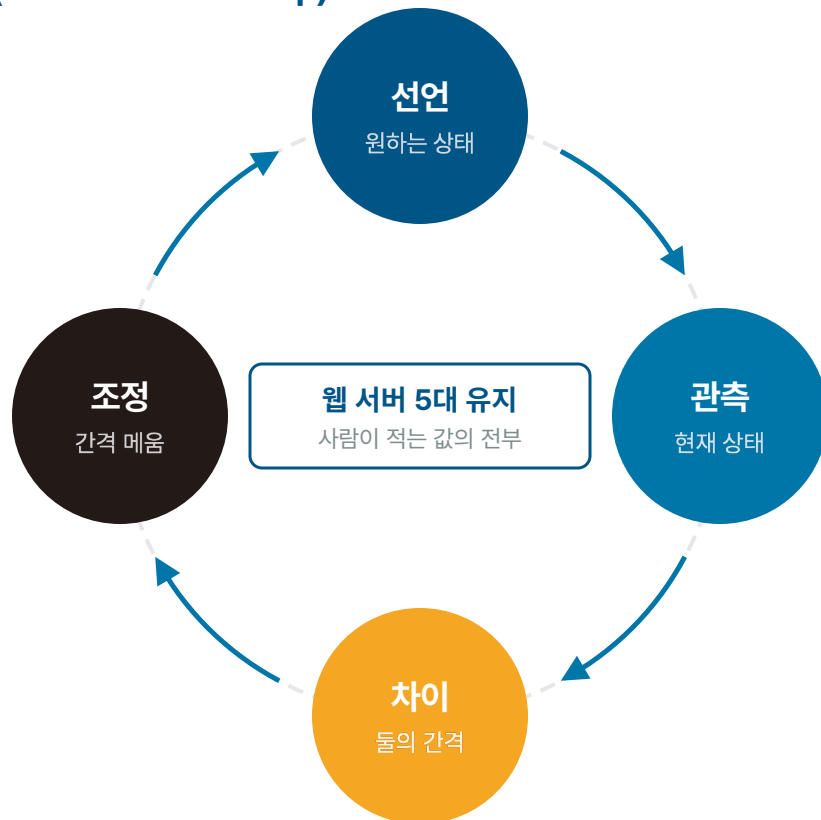
04 사람은 원하는 상태만 선언

05 차이가 생기면 컨트롤러가 스스로 조정

KEY INSIGHT

사람이 하는 일이 절차 수행에서 목표 선언으로 줄어들 — 컨테이너 사망·트래픽 급증·서버 고장 세 경우를 같은 한 줄이 처리

제어 루프 (Reconciliation Loop)



이 한 줄이 처리하는 세 가지 상황



컨테이너가 죽음

선언된 5대보다 적어지면 컨트롤러가 즉시 다시 띄움
사람은 다음 날 원인만 확인



트래픽이 급증

기준을 넘으면 대수를 늘리고, 가라앉으면 다시 줄임
장비 구매 없이 숫자 하나로 끝남



서버가 고장

그 서버에 있던 몫을 남은 서버로 옮겨 실행
새벽에 사람을 부르지 않아도 됨

KEY INSIGHT

가장 크게 바뀐 것은 속도보다 개발팀의 관심사 — "어디서 어떻게 돌지" 를 묻지 않게 되면서 기능에 쓸 시간이 늘어남

일상 업무	예전 · 사람이 운영	지금 · 플랫폼이 운영
배포 주기	분기 · 월 단위, 주말 새벽 작업	하루에도 여러 번, 업무 시간에
장애 대응	담당자가 깨어 직접 복구	자동 복구, 사람은 다음 날 원인만 확인
증설	장비 구매와 설치에 수 주	숫자 하나 변경으로 수 초
운영 지식	담당자 기억과 위키 문서	YAML 코드에 기록 — 검토 · 버전관리 · 복제 가능
서버의 위치	장비마다 이름을 붙여 개별 관리	언제든 교체되는 동일 규격 자원
환경 차이	"제 PC에서는 됩니다"	같은 이미지를 쓰므로 어디서나 동일
개발팀의 관심사	"어디서 어떻게 돌지"	"무엇을 만들지"

KEY INSIGHT

개발팀이 내놓을 것이 두 가지로 줄어듦 — 배치·복구·확장·롤백을 플랫폼이 맡아 OS 와 미들웨어를 몰라도 배포가 성립

예전 · 개발팀이 알아야 했던 범위

앱 · 런타임 · OS 버전 · 미들웨어 설정 · 네트워크 · 로그 · 백업 · 서버



경계선이 여기로 내려옴

지금 · 경계선이 내려온 자리

개발팀

플랫폼 — 배치 · 복구 · 확장 · 네트워크 · OS · 자원 배분 · 롤아웃 · 롤백

제출물 01



컨테이너 이미지

앱과 그 앱이 필요로 하는
것들이 이미 담긴 규격 상자

제출물 02



원하는 상태 한 장

"이것을 3개 띄우고
80번 포트로 열어 줄 것"

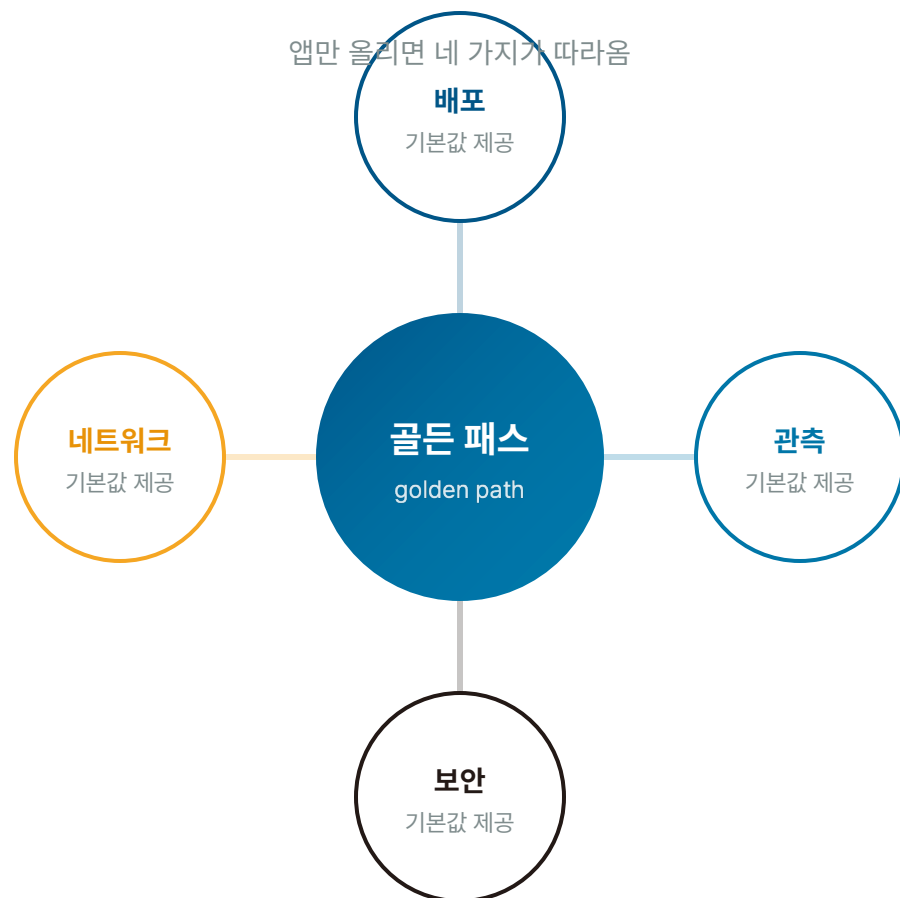
그 아래는 전부 플랫폼의 몫

- 어느 서버에 배치할지
- 멈추면 어떻게 살릴지
- 바쁘면 몇 개로 늘릴지
- 새 버전으로 어떻게 무중단 전환할지

개발자는 이 중 무엇도 몰라도 됨

KEY INSIGHT

플랫폼 엔지니어링이 등장한 이유는 인지 부하 — 쿠버네티스·네트워크·보안을 전원이 익히게 하면 학습 자체가 병목이 됨



왜 닦아 두었나

모든 개발자에게 쿠버네티스 · 네트워크 ·
보안 전문가가 되라고 요구하면

인지 부하 자체가 병목이 됨



경로 밖으로 나갈 수 있음

막지는 않되, 나간 만큼은
그 팀이 직접 책임짐

다른 이름도 같은 뜻

업계에서는 paved road(포장도로)라고도
부름 — 잘 닦아 놓은 기본 경로라는 뜻

KEY INSIGHT

초보자에게 끝까지 남는 것은 그림 한 장 — 설비 공유·책임 경계·교체되는 수명을 함께 담아야 두 번째 설명이 필요 없음

비유	누가 쓰는가	전달되는 것	전달되지 않는 것
오케스트라 지휘자	Red Hat Dave Egts 등 다수	"여러 개를 조율한다"는 인상	예전 방식이 왜 안 되는지, 무엇이 본질적으로 달라졌는지
항구 · 컨테이너선	이름의 어원, 해외 블로그	파드 · 노드 같은 용어 암기에 유용	개발팀의 일이 왜 바뀌는지
단독주택 vs 호텔 이 자료의 주 비유	국내 표준인 주택·아파트 비유를 파드 수명까지 넓힌 형태	설비 공유 · 밀도 · 책임 경계 교체되는 수명까지 한 그림에	차이를 스스로 메우는 자동 조정 (자율주행 비유가 맞음)
자율주행 이 자료의 보조 비유	제어 루프 설명에서 파생	자율 운영이 왜 가능해졌는지의 전제 조건	어디에 무엇이 있는지의 구조

두 비유를 함께 쓰는 이유

앞의 것은 구조 · 책임 · 수명을, 뒤의 것은 자동화의 조건을 설명하며 서로 겹치지 않음
지휘자와 항구 비유는 결과만 묘사해 이 자료에서는 보조로 내림

KEY INSIGHT

파드는 언제든지 교체되는 단위 — 고정 IP 를 기대하고 붙으면 재기동마다 연결이 끊김



같은 호텔 그림으로 이어지는 수명

투숙객이 체크인했다가 떠나듯
파드도 수시로 뜨고 사라짐
오래 사는 거주자가 아닌 투숙객



방 번호가 계속 바뀜

새로 뜬 파드는 새 IP 를 받아,
어제의 주소로는 닿지 않음
IP 를 고정으로 적어 두면 곧 끊김



그래서 Service 를 거침

프런트 데스크에 해당하는 Service 가
바뀌지 않는 진입점을 제공
뒤에서 파드가 바뀌어도 호출은 그대로

KEY INSIGHT

순서를 뒤집으면 실패 — 규격 없이 자동화부터 시도한 과거 방식들이 무너진 자리가 정확히 이 지점

01

예전 방식

서버 한 대에 앱 하나
90% 를 놀리면서도
다른 앱은 못 올림

사람이 손으로 다시 구축하던
실행 환경

02

규격 채택

구글이 수만 대를 사람이
못 만진다는 것을 먼저 겪음
컨테이너를 표준 규격으로 채택
기계가 다룰 수 있게 통일
Borg, 그리고 쿠버네티스

03

설비 공유

각자 갖추던 설비가
공용 설비로 바뀜

**본질은 가벼움보다
몰라도 되는 범위의 확대**
대신 격리 수준은 낮아짐

04

자율 운영

규격 · 관측 · 선언이
갖춰지자 비로소 성립

**사람은 목표만 선언하고
나머지는 컨트롤러가 맞춤**
복구 · 확장 · 재배치



결과

개발팀이 OS 와 인프라를 알지 않아도 개발이 진행됨

THANK YOU

쿠버네티스가 만든 개발팀과 플랫폼의 새로운 분담

컨테이너 규격 위에서만 성립하는 자율 운영 — 오늘 확인한 결론

참고한 출처

- Google Research — Large-scale cluster management with Borg
- The Next Platform — A Decade of Container Control at Google
- Cloud Native Forum 쿠버네티스 eBook 2.1.2

- OPENMARU — 컨테이너 기술과 가상화 기술
- CNCF — Decoding the self-healing Kubernetes
- Red Hat — What is a golden path for software development?

Cloud Native Forum

Cloud Native Forum

<https://cncf.co.kr>

hello@cncf.co.kr