Cloud Native Forum

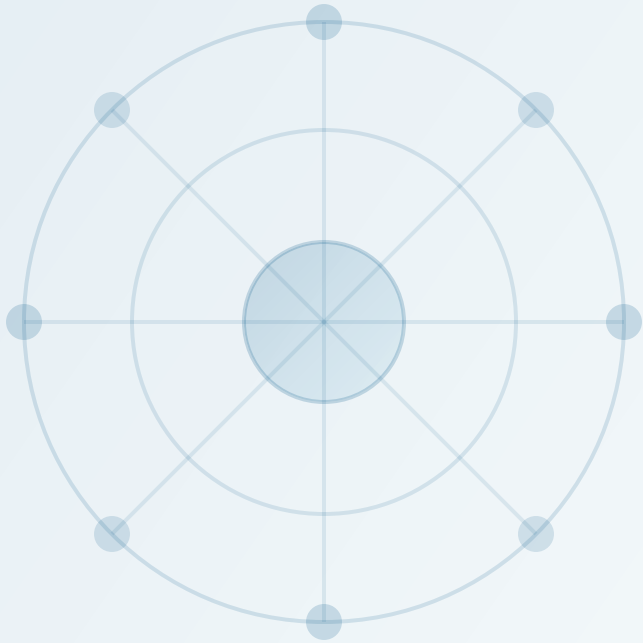
Container Orchestration Platform

Kubernetes

컨테이너 오케스트레이션의 표준

왜 컨테이너인가 · 불변 인프라 · 자가 조정의 원리

PROJECT CODE	DATE
K8S-INTRO	2026 · 07



PUBLIC

CONTENTS

Agenda

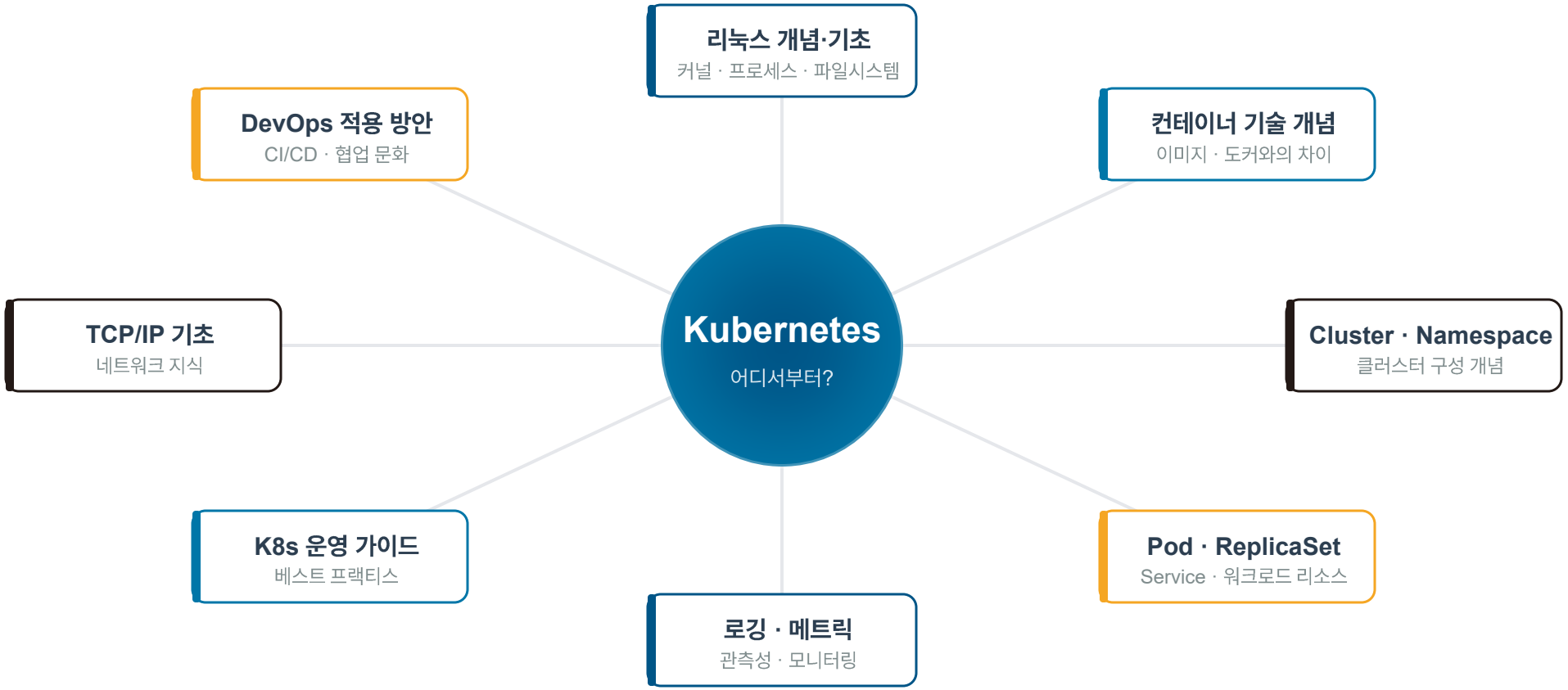
KEY FOCUS 개념 → 패러다임 → 도입까지, Kubernetes를 하나의 흐름으로 이해합니다.

CHAPTER 01

왜 쿠버네티스인가

컨테이너 시대의 운영체제(Cloud Kernel)

KEY INSIGHT 컨테이너를 제대로 활용하려면 넓은 배경지식이 필요하다 — 이 장벽을 낮추는 것이 이 자료의 목표.



Source: 원본 교육자료(Slide 2) 재구성

Kubernetes = 클라우드의 커널(OS)

KEY INSIGHT 리눅스 커널이 하드웨어를 추상화하듯, 쿠버네티스는 클라우드 리소스를 추상화하고 관리한다.

Linux Kernel

하드웨어를 추상화

- 프로세스 관리
- 메모리 관리
- 장치 관리 (Device)
- 네트워크 스택
- 프로세스 간 통신 (IPC)
- 사황 감시 (watchdog)

Kubernetes (Cloud Kernel)

클라우드를 추상화

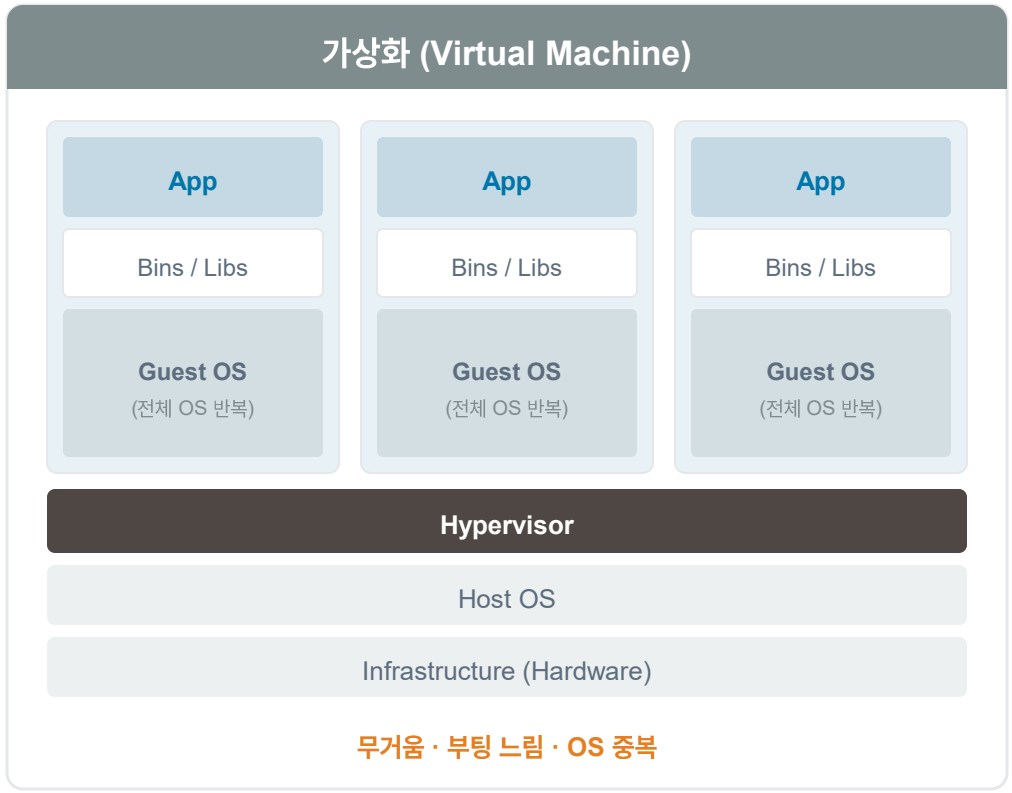
- 컨테이너 스케줄링
- CPU · 메모리 리소스 관리
- CSI — 스토리지 인터페이스
- CNI — 컨테이너 네트워크
- Service · 서비스 디스커버리
- 헬스체크 · 자가 치유

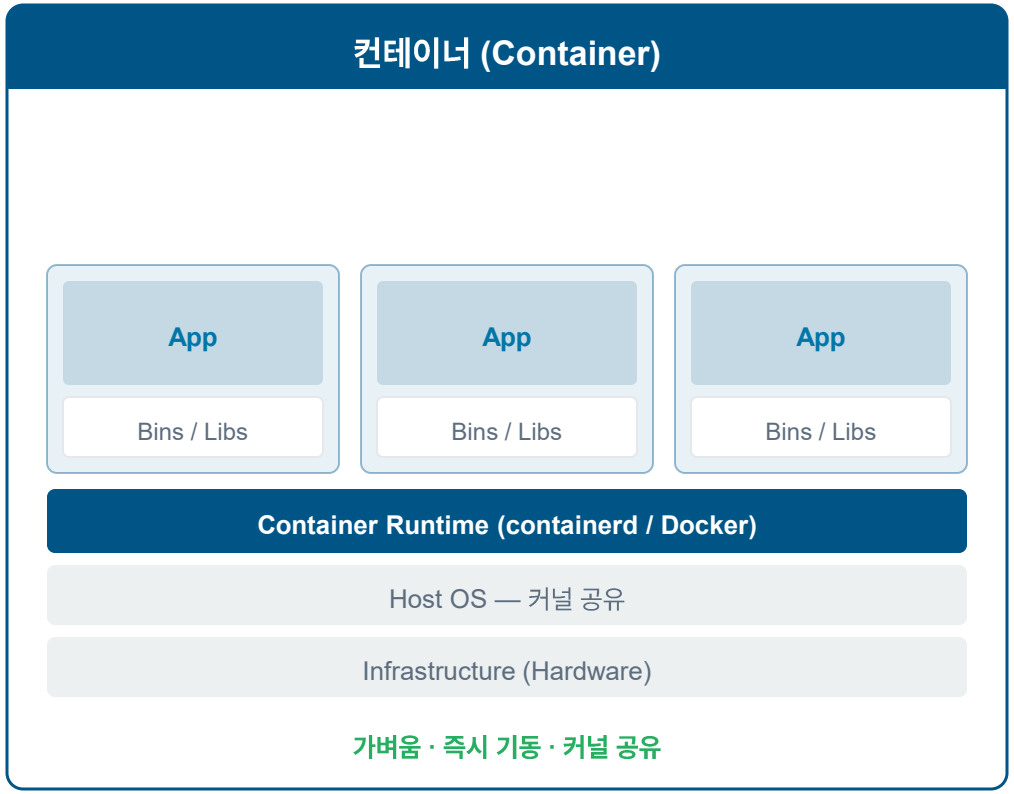
Legacy Application

Cloud Native Application

Source: 원본 교육자료(Slide 3) 재구성 · Linux/Kubernetes 아키텍처 유비

KEY INSIGHT 컨테이너는 게스트 OS 없이 호스트 커널을 공유한다 — 그래서 가볍고, 빠르고, 어디서나 일관되게 돈다.





Low Overhead — 낮은 오버헤드

Run Anywhere — 어디서나 실행

Consistent Runtime — 일관성

Source: 원본 교육자료(Slide 4·12) 재구성

CHAPTER 02

Google과 컨테이너

주당 20억 컨테이너에서 태어난 표준

Google은 모든 것을 컨테이너로 실행한다

KEY INSIGHT 10여 년간 프로덕션에서 컨테이너를 운영한 경험 — 그 지식이 cgroups와 Kubernetes로 공개되었다.

Google의 업무 방식

- Gmail · 검색 · 지도 · YouTube — 모든 제품이 컨테이너에서 실행
- MapReduce · GFS · Colossus 등 내부 시스템도 컨테이너 기반
- Google Compute Engine 가상 머신도 컨테이너 위에서 동작
- 매주 20억 개 이상의 컨테이너를 생성·실행

커뮤니티에 지식을 공유

- cgroups 기능을 Linux 커널에 기여 (컨테이너 격리의 토대)
- 내부 도구 설계를 오픈소스 Kubernetes 프로젝트로 공개

20억+

매주 생성되는 컨테이너 수

Billions of containers / week

10+

년간 프로덕션

컨테이너 운영 경험

2008

cgroups 커널 병합

Linux 2.6.24

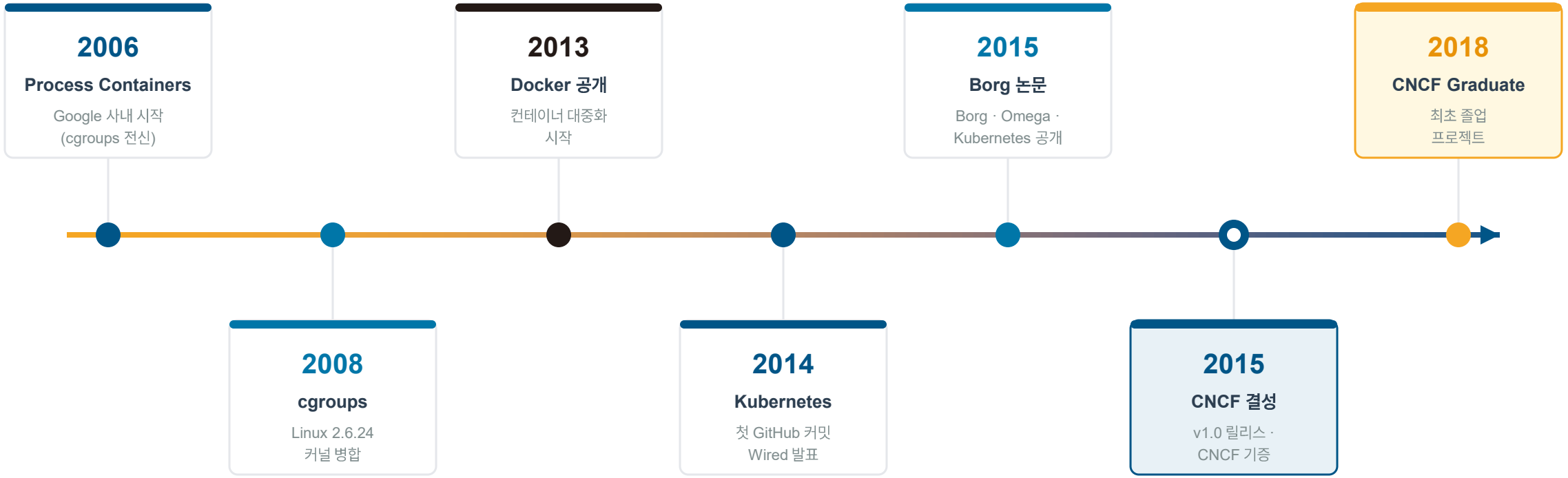
"이 전문 지식을 클라우드로 구현하여
크고 작은 조직이 최신 컨테이너 기술을
쉽게 활용할 수 있게 했다."

— Google Cloud, Containers

Source: cloud.google.com/containers · 원본 교육자료(Slide 5·6)

Kubernetes History

KEY INSIGHT 2006년 사내 실험(Process Containers)에서 2018년 CNCF 최초 졸업까지 — 12년에 걸쳐 표준이 되었다.



Source: CNCF · "Borg, Omega, and Kubernetes" · 원본 교육자료(Slide 7)

KEY INSIGHT 특정 인프라에 묶이지 않는 이식성(Portability) — 이것이 곧 벤더 중립성이며 CNCF의 핵심 가치다.

베어메탈
Bare Metal

물리 서버에 직접 배포

가상화 계층 없이 최대 성능

온프레미스
On-Premise

자체 데이터센터 운영

데이터 주권 · 규제 대응

Public Cloud
AWS · GCP · Azure

관리형 K8s: EKS · GKE · AKS

운영 부담 최소화

프라이빗 클라우드
Private Cloud

VMware · OpenStack 위에

폐쇄망·보안 요구 환경

하이브리드 클라우드
Hybrid Cloud

온프레 + 퍼블릭 연계 운영

워크로드 유연 배치

엣지 · IoT
Edge Computing

경량 배포판(K3s 등)

현장·소형 디바이스

한 번 정의한 워크로드를, 모든 환경에서 동일하게

이식성(Portability)은 특정 공급자에 대한 종속(lock-in)을 제거한다 — 이것이 오픈 거버넌스의 목적이다.

Source: 원본 교육자료(Slide 8) 재구성

CHAPTER 03

컨테이너 오케스트레이션

다수 컨테이너 운영의 과제와 해결

KEY INSIGHT 컨테이너 하나는 쉽다 — 수백 개를 사람이 직접 관리하는 순간, 운영은 감당하기 어려운 문제가 된다.

컨테이너 배포
Deployment

- 배포할 호스트(노드) 결정과 실제 배치
- 컨테이너에 할당할 MAC · IP · 호스트 이름 관리
- 애플리케이션에 대한 라우팅 경로 설정
- 데이터를 유지하는 영구 저장소(Persistent Volume) 할당

장애 대응
Failure

- 노드 장애 시 — 다른 노드로 컨테이너 재배치
- 중복 결함 시 — 지정한 복제(replica) 수 확보

자동 확장·축소
Scaling

- 부하 증가 시 — 스케일 아웃(Scale-out)
- 부하 감소 시 — 스케일 인(Scale-in)으로 자원 회수

⚠ **이 모든 작업을 사람이 수작업으로 — 인프라 관리자의 컨테이너 운영 부담**

Source: 원본 교육자료(Slide 13) 재구성

03 Kubernetes 도입 시 변화 포인트

KEY INSIGHT 도입은 기술만의 변화가 아니다 — 인프라·개발·조직 세 축이 함께 바뀌어야 성공한다.

인프라 · 운영 변화

- 네트워크 구조 재설계
- 스토리지 동적 프로비저닝
- 롤링 · 카나리 배포 적용
- 모니터링 · 감시 체계 전환
- 선언적 운영 · GitOps 도입
- 로그 수집 · 집계 재구성
- 리소스 중심 문제 해결 방식

개발 · 기술 변화

- 애플리케이션 컨테이너화 전환
- YAML 기반 설정 자동화
- 다양한 운영 자동화 기능 적용

(운영이 사라지는 것이 아니라,
자동화된 운영으로 바뀌는 것)

선언적(Declarative)

"어떻게"가 아니라
"원하는 상태"를 기술한다

Desired State → 시스템이 수렴

조직 · 보안 변화

- RBAC · NetworkPolicy · PodSecurity 중심 보안 설계
- DevOps · 플랫폼 팀 중심 협업 구조로 전환
- K8s 기반 CI/CD 재구축
- Liveness · Readiness Probe 중심 장애 대응 전략

Source: 원본 교육자료(Slide 10) 재구성

KEY INSIGHT 운영자가 상시 감시하지 않아도, 클러스터가 스스로 "원하는 상태"를 유지한다.

자동 장애 복구
Self-Healing

비정상 Pod를 감지하고
자동으로 재시작·교체한다.

자동 확장·축소
Auto Scaling (HPA)

부하에 따라 Pod·노드
수를 자동 조절한다.

서비스 디스커버리
Service Discovery

고정 IP 없이 서비스 간
통신을 동적으로 연결한다.

롤링 업데이트
Rolling Update

서비스 중단 없이 점진적
으로 업데이트한다.

로드 밸런싱
Load Balancing

L4·L7 장비 없이 소프트
웨어로 부하를 분산한다.

스케줄링
Scheduling

CPU·메모리를 보고 최적
노드에 컨테이너를 배치한다.

영구 볼륨
Persistent Volume

컨테이너 재기동에도
데이터를 보존한다.

선언적 배포
Declarative

"원하는 상태"를 기술하면
시스템이 스스로 수렴한다.

부하 대응 · 배포 · 복구 · 스케줄링을 자동화 — 운영자는 "무엇을" 원하는지만 정의한다

Source: 원본 교육자료(Slide 11·36·37) 통합·보강

03 컨테이너 과제 → Kubernetes의 해결

KEY INSIGHT Kubernetes는 여러 노드를 중앙에서 통합 관리하며, 사람이 하던 운영 작업을 선언적 자동화로 대체한다.

Kubernetes 특징

- 여러 컨테이너 노드를 중앙에서 통합 관리
- Pod 라이프사이클 모니터링 (시작 · 정지)
- Pod 네트워크 설정
- Pod 스토리지 설정
- 부하에 따른 자동 확장·축소

선언적 · 자동화된
중앙 통합 운영

컨테이너의 과제	Kubernetes의 해결
컨테이너 배포	설정에 따라 컨테이너를 자동 배치·배포 Pod 단위 자동 IP 부여 및 연결 경로 설정 영구 저장소 자동 연결
자동 장애 복구 가용성 확보	자동 복구로 가용성 확보 설정한 Pod 복제 수를 일정하게 유지 노드 장애 시 지정 수량에 맞춰 자동 복구
자동 부하 대응 Auto Scaling	피크 시 스케일 조작·자원 관리 부하 경감 접속량을 모니터링하여 부하에 따라 자동으로 추가 배치

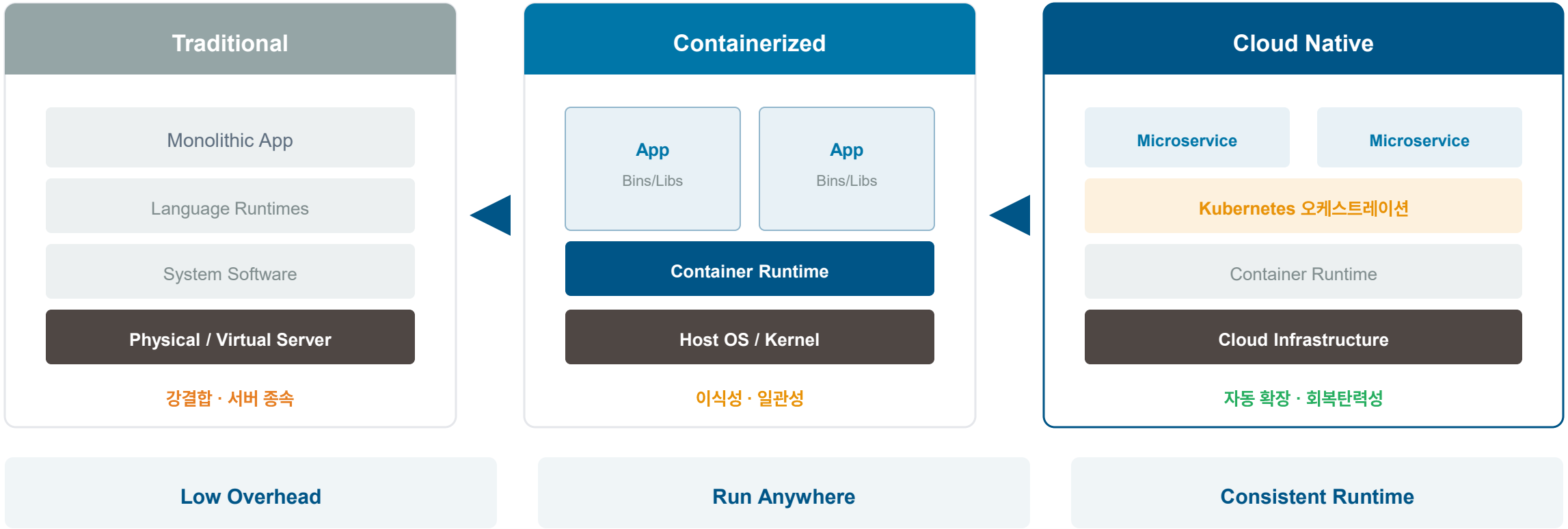
Source: 원본 교육자료(Slide 14) 재구성

CHAPTER 04

불변 인프라 패러다임

Mutable vs Immutable Infrastructure

KEY INSIGHT 컨테이너화의 이유는 가상화의 이유와 같다 — 그리고 클라우드 네이티브로 가면 확장성과 회복탄력성까지 얻는다.



Source: 원본 교육자료(Slide 12) 재구성

Configuration Drift — 인프라가 제각각이 되는 현상

KEY INSIGHT 손으로 고친 임시 수정이 쌓이면 같던 서버들이 시간이 갈수록 서로 달라진다 — 재현 불가능한 인프라.

정의

- 손으로 직접 한 임시 수정·업데이트와 전반적 엔트로피 증가로, 서버들이 시간이 갈수록 서로 다른 상태가 되는 현상.
- 장비가 라이프사이클 동안 초기 설정에서 멀어지고(drift), 다른 장비들과도 달라진다.

결과

- 장애 재현·원인 추적 어려움
- "그 서버에서만" 발생하는 버그
- 신규 서버 구성 불일치

배포 직후 — 4대가 동일

Server A	Server B	Server C	Server D
Java 1.7.0	Java 1.7.0	Java 1.7.0	Java 1.7.0
Lib 2.1 · RHEL 7.4	Lib 2.1 · RHEL 7.4	Lib 2.1 · RHEL 7.4	Lib 2.1 · RHEL 7.4

수작업 패치·핫픽스 누적 (200일)

운영 200일 후 — 제각각으로 drift

Server A	Server B	Server C	Server D
Java 1.7.9	Java 1.7.2	Java 1.7.5	Java 1.7.1
Lib 2.1 · +hotfix	Lib 2.4 · RHEL 7.6	Lib 2.1 · patch3	Lib 2.0 · RHEL 7.4
⚠ 수동 튜닝	⚠ 임시 업글	⚠ 핫픽스	⚠ 롤백됨

눈덩이(Snowflake) 서버

아무도 정확한 상태를 모르고, 똑같이 다시 만들 수 없는 서버가 된다.

Source: 원본 교육자료(Slide 18) 재구성 · Configuration Drift 개념

KEY INSIGHT 업데이트는 덮어쓰는 것이 아니라, 버리고 새로 만드는 것 — "No upgrade needed."

정의

- 서버가 배포된 이후 절대 변경하지 않는 인프라 패러다임
- 업데이트 = 덮어쓰기가 아니라 버리고 새로 만드는 것

"No Upgrade Needed"

- Disposable Components — 언제든 폐기 가능한 구성요소

멱등성 (Idempotency)

- 같은 작업을 여러 번 해도 결과가 항상 동일하다

f(f(x)) = f(x)

한번 설정된 서버는 수정 없이 파기

- → 항상 동일한 결과가 보장되어 멱등성이 성립한다

장점

- 서버를 쉽게 삭제·증설 가능
- 인프라 환경 변경이 쉬워짐
- 서버를 항상 깨끗이 유지

Configuration Drift 없음

눈덩이 서버 걱정에서 해방

"Trash Your Servers and Burn Your Code"

업그레이드가 필요한가요? 새 시스템을 도입하고 이전 것을 버리세요. 새 버전 배포도 마찬가지 — 새 서버를 만들고 옛것을 제거하세요.

— Chad Fowler, 2013 (Immutable Deployments)

KEY INSIGHT 불변 인프라는 서버를 '가축'처럼 다룬다 — 아프면 고치는 게 아니라, 교체한다.

🐾 Pets — 애완동물

Mutable · Legacy

- 이름을 붙이고 애지중지 키운다 (web01, db-master…)
- 아프면 정성껏 치료한다 (수작업 패치·튜닝)
- 대체 불가능한 유일한 시스템으로 다룬다
- 한 대가 죽으면 큰일 — 다운타임 발생

"운영자님, 아직도 서버를 애지중지 키우고 계신가요?"

"우리 서버 이쁘죠? 구축할 때 얼마나 고생했는데요."
"제가 휴가라 그 서버는 손 못 대요…"

🐄 Cattle — 가축

Immutable · Cloud

- 번호(태그)로 관리한다 (node-a1b2, node-c3d4…)
- 문제가 생기면 고치지 않고 교체한다
- 자동화로 대량을 동일하게 관리한다
- 한 대가 죽어도 무방 — 자동 대체로 무중단

고치지 말고, 교체하라

- 모든 서버가 동일 → 재현 가능
- 교체가 곧 복구 → 자가 치유의 토대
- Kubernetes의 Pod가 바로 이 방식

Source: 원본 교육자료(Slide 21·28) 재구성 · "Pets vs Cattle" (Bill Baker)

04 서버 업그레이드: In-Place vs Replace

KEY INSIGHT 기존 서버를 그 자리에서 고치면 다운타임과 drift가, 이미지를 새로 만들어 교체하면 무중단이 남는다.

Mutable — In-Place 업그레이드

1. 운영 중인 서버에 SSH 접속
2. 애플리케이션 정지 (stop)
3. repo 업데이트 · 라이브러리 업그레이드
4. 재부팅 (reboot)
5. 테스트 · 디버그 · 애플리케이션 재기동

남는 것

× Downtime (서비스 중단) × Configuration Drift

× 취약점 누적 × 실패 시 롤백 어려움

Immutable — Replace 업그레이드

1. 새 버전으로 Golden Image 빌드
2. 이미지 테스트 · 디버그 (배포 전)
3. 새 이미지로 신규 서버 배포
4. 트래픽을 새 서버로 전환
5. 이전 서버 폐기 (Disposable)

남는 것

✓ Zero Downtime ✓ Drift 없음

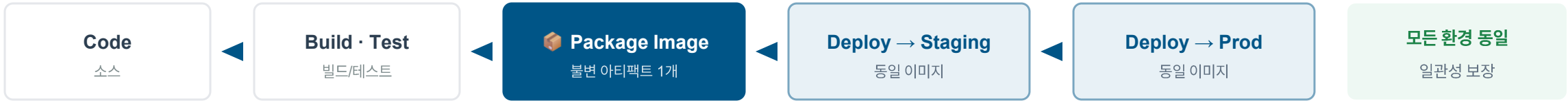
✓ 즉시 롤백(이전 이미지) ✓ 재현 가능

Source: 원본 교육자료(Slide 22·23) 재구성

KEY INSIGHT 불변 파이프라인은 이미지를 한 번만 빌드해 모든 환경에 동일하게 배포한다 — "한 번 빌드, 어디서나 동일".

Mutable (In-Place) — 환경마다 서버를 다시 구성

Immutable (Replace) — 한 번 만든 이미지를 그대로 배포



핵심 차이: 변경의 대상

- Mutable — 배포할 때마다**실행 중인 서버**를 바꾼다 → 환경마다 상태가 달라진다.
- Immutable — 배포할 때마다**이미지를 교체**한다 → 서버는 절대 손대지 않는다.

Source: Adrian Hornsby (AWS), "Reliability, consistency, and confidence through immutability" · 원본(Slide 24·25)

KEY INSIGHT 도자기 컵은 깨지면 끝이지만, 종이컵은 하나 버리는 데 부담이 없다 — 서버를 어떻게 다룰 것인가.

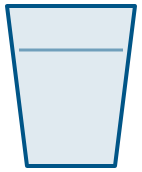
고급 도자기 컵 = Mutable

지속적 패치가 필요한 Legacy Infra

- 항상 운영 가능한 상태로 유지해야 함
- 대체 불가능한 유일한 시스템으로 취급
- 지속적 Patch 관리가 필요

깨지면 모든 것이 끝난다

일회용 종이컵 = Immutable



한 번 쓰고 버리는 Cloud Infra

- 서버를 일회용으로 취급
- 하나 버리는 데 부담이 없음
- 문제 생기면 새 컵으로 교체 (Disposable)

버려도 아무 문제 없다

Source: 원본 교육자료(Slide 26) 재구성

머신 중심 → 애플리케이션 중심 인프라

KEY INSIGHT 컨테이너화는 데이터센터를 '서버 중심'에서 '애플리케이션 중심'으로 전환한다 — 관심의 단위가 바뀐다.

Machine-Centric Infrastructure

Server: CPU/Mem

Server: CPU/Mem

Server: CPU/Mem

관리 단위 = 물리 서버 · CPU · 메모리

개발자·운영자가 서버 세부사항을 직접 신경 씀

서버에 애플리케이션을 맞춘다

컨테이너화

Application-Centric Infrastructure

App A | App B | App C | App D

추상화된 리소스 풀 (Cluster)

관리 단위 = 애플리케이션 · 컨테이너

서버 세부사항은 추상화되어 숨겨짐

애플리케이션에 인프라를 맞춘다

추상화

서버·운영 환경 세부사항을
개발/운영팀에게서 감춘다

유연성

앱 영향 최소화하며 하드웨어·
운영 환경을 업그레이드

앱 메트릭 오토스케일

CPU·메모리뿐 아니라 앱 지표를
연결해 자동 확장

Source: 원본 교육자료(Slide 27) 재구성

CHAPTER 05

Kubernetes 이해하기

오픈소스 · 표준 · 생태계

KEY INSIGHT 컨테이너화된 앱을 자동으로 배포·스케일·관리하는 오픈소스 — 특정 벤더가 아닌 CNCF가 중립적으로 관리한다.

정의 · 오픈소스

- 컨테이너화된 애플리케이션을 자동으로 배포 · 스케일링 · 관리하는 오픈소스 SW
- "K8s" = K + (ubernete 8글자) + s
- Go로 작성 · Apache License 2.0
- Linux Foundation 산하 CNCF가 관리
- Google의 내부 도구 "Borg"에서 유래

"Kubernetes is open source — a contrast to Borg and Omega." — Borg, Omega, and Kubern

Portability (이식성)의 표준 인터페이스

cloud-controller-manager
클라우드 작업(L4 LB·블록 스토리지) 추상화
VMware · GCP · AWS · Azure · OpenStack

CNI — Container Network Interface
컨테이너 네트워크의 표준화
Calico · Flannel · Weave Net

CRI — Container Runtime Interface
컨테이너 런타임 표준
containerd · CRI-O

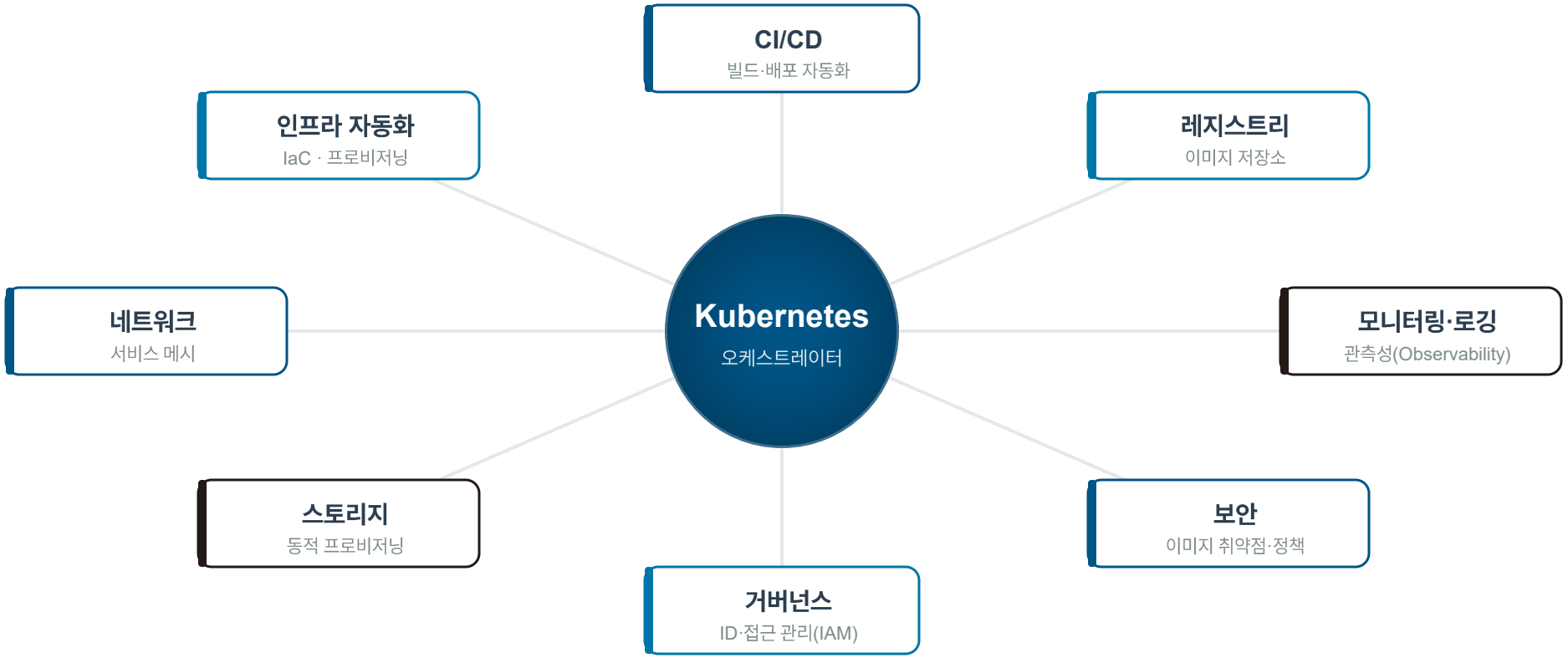
표준 인터페이스 = 특정 구현·벤더에 종속되지 않음

CNCF 최초 Graduated 프로젝트
오픈 거버넌스 · 오픈 디자인(github.com/kubernetes/community) · SIGs 커뮤니티가 함께 만든다.

Source: 원본 교육자료(Slide 31·32) 재구성 · kubernetes.io

05 Kubernetes만으로는 충분하지 않다 — 생태계

KEY INSIGHT Kubernetes는 오케스트레이터일 뿐 — 프로덕션에는 CI/CD·관측성·보안 등 생태계가 함께 필요하다.



Source: 원본 교육자료(Slide 33) 재구성 · CNCF Landscape

05 Containers & Kubernetes: The Time Is Now

KEY INSIGHT 컨테이너를 개발에서 운영으로 옮기는 "학습 절벽(Learning Cliff)"을 Kubernetes가 메운다.

Containers in DEVELOPMENT

- Container
- Container

로컬에서는 쉽다

✓ 빌드 · 실행 간단

Kubernetes — 절벽을 잇는 다리

Load Balancing 소프트웨어 부하분산	**Security** RBAC · 정책	**High Availability** 고가용성
App Updates 롤링 업데이트	**Scaling** 자동 확장·축소	**Repeatable Deploy** 반복 가능한 배포
Replication 복제 관리	**Scheduling** 최적 배치	**Self-Healing** 자가 치유

운영이 요구하는 기능을 Kubernetes가 표준으로 제공한다

THE "LEARNING CLIFF" — 이 격차를 넘는 것이 도입의 핵심

Containers in PRODUCTION

- 수백 개 컨테이너
- 무중단 · 고가용

운영은 어렵다

× 수작업으로는 불가

Source: VMware Cloud-Native, "Containers & Kubernetes benefits" · 원본 교육자료(Slide 34)

KEY INSIGHT 파편화되어 경쟁하던 오케스트레이터들이 Kubernetes로 통합되었다 — 사실상의 업계 표준.

7년 전 — Fragmented Landscape

- Docker Swarm
- Apache Mesos
- Cloud Foundry Diego
- Nomad
- AWS Blox
- 기타 오케스트레이터…

난립 · 상호 비호환 · 학습 부담

어떤 것을 선택해야 할지 불확실

→ 표준 부재로 인한 파편화

통합

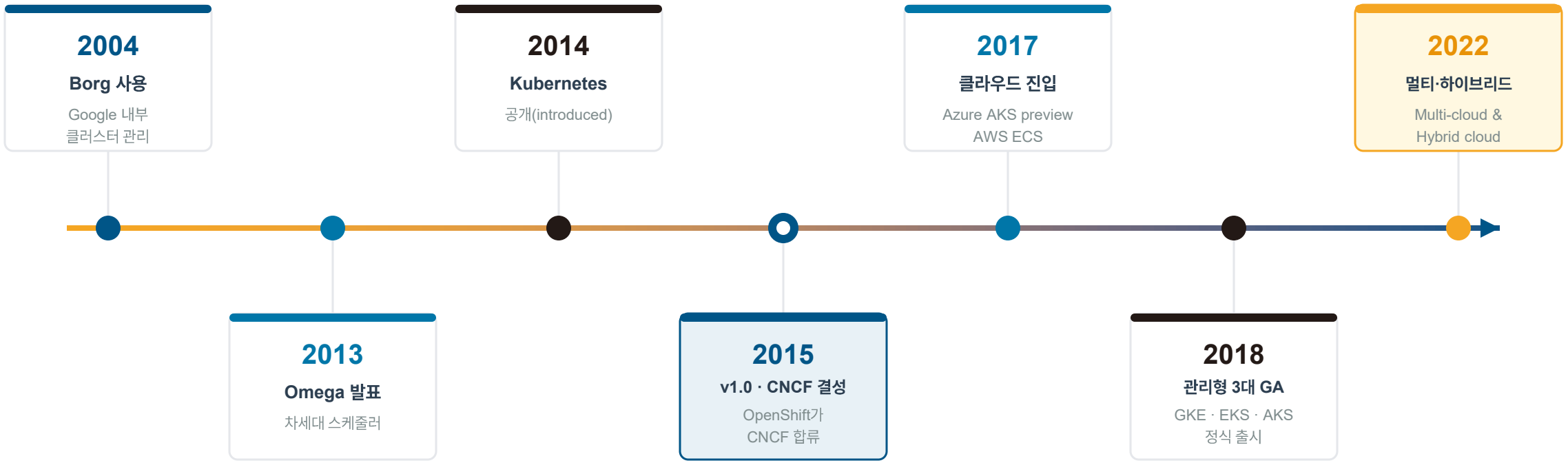
오늘 — Kubernetes Consolidation

단일 표준으로 수렴

모든 주요 클라우드가 관리형 K8s 제공

Source: 원본 교육자료(Slide 35) 재구성

KEY INSIGHT 사내 도구에서 시작해 모든 주요 클라우드의 관리형 서비스로 — 분산·하이브리드 환경의 표준이 되었다.



Source: CNCF · Timeline of Kubernetes Events · 원본 교육자료(Slide 41)

05 Kubernetes는 누구에게 필요한가

KEY INSIGHT 잦은 변경·배포와 예측 불가한 부하를 다루는 팀일수록 Kubernetes의 자동화 효과가 크다.

대상 팀

IT 운영 부담을 줄이려는

- 개발팀 · 운영팀
- 네트워크팀
- DevOps 팀

공통점

변화가 빠르고, 자동화로
운영 부담을 줄여야 하는 팀

이런 상황이라면 Kubernetes가 답이다

1. **애플리케이션 변경이 잦고, 하루에 여러 번 배포한다**
   잦은 릴리스 → 롤링 업데이트·무중단 배포가 필수
2. **이벤트성 부하에 대응할 수 있는 시스템이 필요하다**
   트래픽 급증 → 자동 확장(HPA)으로 대응
3. **빈번한 설정 변경·시스템 작업의 이력 관리가 필요하다**
   선언적 매니페스트 → GitOps로 변경 이력 추적
4. **복잡한 작업 절차서를 단순화하고 싶다**
   수동 절차 → 자동화된 오케스트레이션으로 대체
5. **명확한 배포 정책과 롤백 방안이 필요하다**
   배포 전략(카나리·블루그린)과 즉시 롤백 지원

Source: 원본 교육자료(Slide 17) 재구성

CHAPTER 06

도입 전략과 로드맵

무엇을, 어떻게 시작할 것인가

KEY INSIGHT 경영진(CIO) 관점에서 Kubernetes 도입은 비용·속도·유연성·확장성·전환 다섯 축에서 이득을 준다.

01

IT 비용 최적화

IT cost optimization

리소스 사용률을 높여 인프라 비용 절감

02

신속한 개발

Faster time to market

배포 자동화로 출시 주기를 단축

03

멀티 클라우드

Multi-cloud flexibility

특정 클라우드에 종속되지 않는 유연성

04

확장성·가용성

Scalability & availability

자동 확장·자가 치유로 확장성과 가용성 향상

05

클라우드 네이티브

Cloud-native migration

효율적인 클라우드 네이티브 전환의 기반

Source: 원본 교육자료(Slide 39·40) 재구성 (vendor-neutral)

Kubernetes 도입 로드맵 4단계

KEY INSIGHT 한 번에 전환하지 말고 검토 → 시범 → 확산 → 내재화의 단계로 성숙도를 높여간다.

01 도입 검토 — Exploring

자사 인프라·애플리케이션에
도입 가능성을 검토

PoC · 학습 · 타당성 평가

02 시범 운영 — Initial Adoption

일부 비즈니스·내부 업무
시스템을 K8s 기반 운영

저위험 워크로드부터 적용

03 본격 확산 — Production

운영 환경(Production)에도
본격 적용·확산

SLA·모니터링·보안 체계화

04 내재화·고도화 — Mature

핵심 IT 인프라로 내재화하고
전략적으로 활용

플랫폼 엔지니어링·최적화

성숙도 · 활용 범위 →

Source: 원본 교육자료(Slide 42) 재구성

성공적인 도입을 위한 4가지 실행 단계

KEY INSIGHT 대상 선정 → 플랫폼 선정 → 성공 기준 정의 → 조직 문화 정렬, 이 순서로 준비해야 도입이 성공한다.

1 구현 대상 선정
Select an Implementation Target

- 느슨한 결합이거나 재설계를 검토 중인 애플리케이션
- 마이크로서비스로 개발되었거나 전환 중인 앱
- 리소스 요구사항을 충분히 이해
- 비즈니스 영향이 측정 가능한 상태

2 플랫폼 선정
Choose a Kubernetes Platform

- 호스팅·클라우드·온프레·멀티클라우드 및 스케일링 지원
- 기존 앱과 상호호환성 (RBAC·UI·Active Directory 통합)
- 컨테이너 운영을 전제로 한 비용·지급 모델 타당성

  관리형(EKS/GKE/AKS) vs 자체 구축 비교

3 성공 기준 정의
Plan your Success Criteria

- 기존 대비 성능이 동일한지·차이가 어디서 나는지 확인
- 릴리스 주기 단축 · 딜리버리 파이프라인 표준화
- 전반적 안정성·유연성 개선
- 예측 불가한 부하에 대한 성능·안정성

4 조직 문화 정렬
Align with Corporate Culture

- 비즈니스 요구에 맞게 확장 가능한 클라우드 네이티브 플랫폼을 구축
- 새 도구 사용·변화에 잘 대응하는 방법을 찾아간다
- 배포·릴리스 방법 변화에 대한 자동화 추진

Source: 원본 교육자료(Slide 43–46) 재구성

CHAPTER 07

관리자에게 Kubernetes란

운영 방식의 전환과 클라우드 네이티브

07 운영자가 알아야 할 것들

KEY INSIGHT Kubernetes 운영으로 넘어가기 전, 운영자가 반드시 답을 가져야 할 다섯 가지 질문.

? 기존 운영환경 구성과의 차이점은 무엇인가?

선언적 운영·자동화로 관리 방식이 근본적으로 달라진다.

→ Ch.3 컨테이너 오케스트레이션

? 불변의 인프라스트럭처는 어떤 개념인가?

서버를 고치지 않고 교체하는 '가축' 방식의 인프라.

→ Ch.4 불변 인프라 패러다임

? 가상화 대비 컨테이너 기술의 장점은?

커널 공유로 가볍고, 이식성과 일관성이 뛰어나다.

→ Ch.1 가상화 vs 컨테이너

? 오토스케일링·오토힐링은 어떻게 동작하나?

Reconciliation Loop가 원하는 상태를 계속 유지한다.

→ Ch.8 Reconciliation Loop

? 클라우드 네이티브는 또 무엇인가?

클라우드가 클라우드답게, 앱을 구축·실행하는 방식.

→ 다음 페이지

Source: 원본 교육자료(Slide 48) 재구성

07 클라우드 네이티브 기반 PaaS로 전환

KEY INSIGHT 클라우드 네이티브 = "클라우드가 클라우드다울 수 있도록" 애플리케이션을 구축·실행하는 방식.

Legacy System

- 모놀리스 애플리케이션
- 미들웨어 · 데이터베이스
- OS · 가상화
- **하드웨어**

개발·운영 분리 · 수작업 많음

PaaS 전환

Cloud Native System (PaaS)

Microservices	CI/CD	DevOps 운영
Dev	지속 배포	Ops

Kubernetes · 컨테이너 (PaaS Layer)

클라우드 인프라 (온프레·퍼블릭·하이브리드)

개발·운영 통합(DevOps) · 자동화

Source: 원본 교육자료(Slide 49) 재구성

KEY INSIGHT 웹 시스템은 노드가 많고 장애에 민감하다 — 수작업 운영은 환경 불일치와 Human Error를 낳는다.

웹 시스템의 요구사항과 특징

- Scale-out형 인프라 · 계층형 아키텍처
- 가상화·클라우드에 적합, 인스턴스 수가 많고 노드 간 연결성이 높음
- 장애에 민감 — 신속한 복구와 재발 방지가 중요
- 설치되는 관련 소프트웨어가 많아 환경 검증 필요

"서버 에러?"

"사용자 에러!"

시스템 운영 이슈

시스템 환경의 불일치
Dev / Stage / Prod, 서버별로 상태가 다름

긴 배포 시간

수작업으로 인한 Human Error

IT Agility 부족 → 운영팀 축소
개발팀이 직접 IT 인프라를 운영하게 됨

문제 발견·조치에 많은 시간 소요

→ 자동화·표준화가 없으면 반복되는 문제

Source: 원본 교육자료(Slide 50) 재구성

07 애플리케이션 배포 기간 단축

KEY INSIGHT 개발팀·인프라팀 간의 잦은 수작업 상호작용을 자동 배포로 대체하면 배포 기간이 크게 단축된다.

기존 — 수작업 · 잦은 상호작용

작업계획서 · 협의 반복

서버에 수작업 설치·설정 → 서버

서버 | 서버 | 서버

문제점

× 배포까지 오랜 리드타임 × 팀 간 병목

× 수작업 오류 × 환경 불일치

Kubernetes — 자동 배포

인프라팀은 플랫폼만 제공 (상호작용 최소화)

동일 이미지 · 선언적 배포 · 무중단

효과

✓ 배포 리드타임 대폭 단축 ✓ 셀프서비스

✓ 오류 감소 ✓ 환경 일관성

Source: 원본 교육자료(Slide 51) 재구성

CHAPTER 08

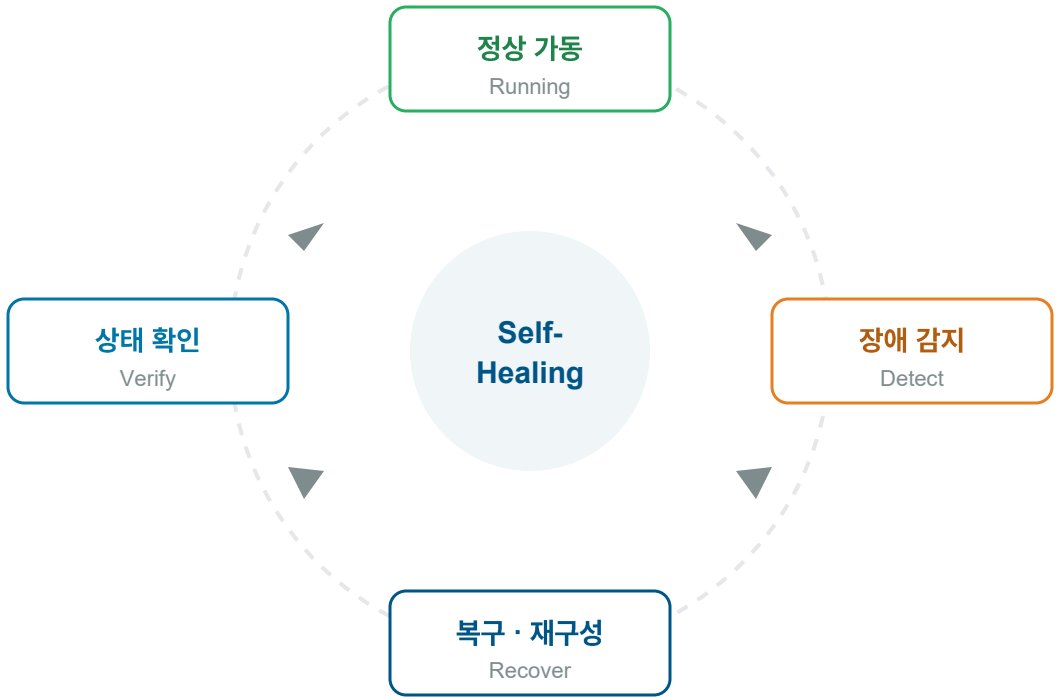
Reconciliation Loop

원하는 상태를 유지하는 자가 조정의 원리

안정된 운용 — 자가 치유(Self-Healing)

KEY INSIGHT 오케스트레이션은 장애를 감지하고 스스로 복구한다 — 운영자의 개입 없이 정상 상태를 되찾는다.

Self-Healing 루프



무상태 — Stateless Application

- 상태를 갖지 않아 어느 복제본이든 동일
- 장애 시 즉시 동일 복제본으로 교체

정상 가동 → 정상 여부 확인 → (장애 시) 즉시 재생성

상태 — Stateful Application

- 데이터·상태를 보존해야 함 (Persistent Volume)
- 장애별 대응(Customize)으로 동적 복구

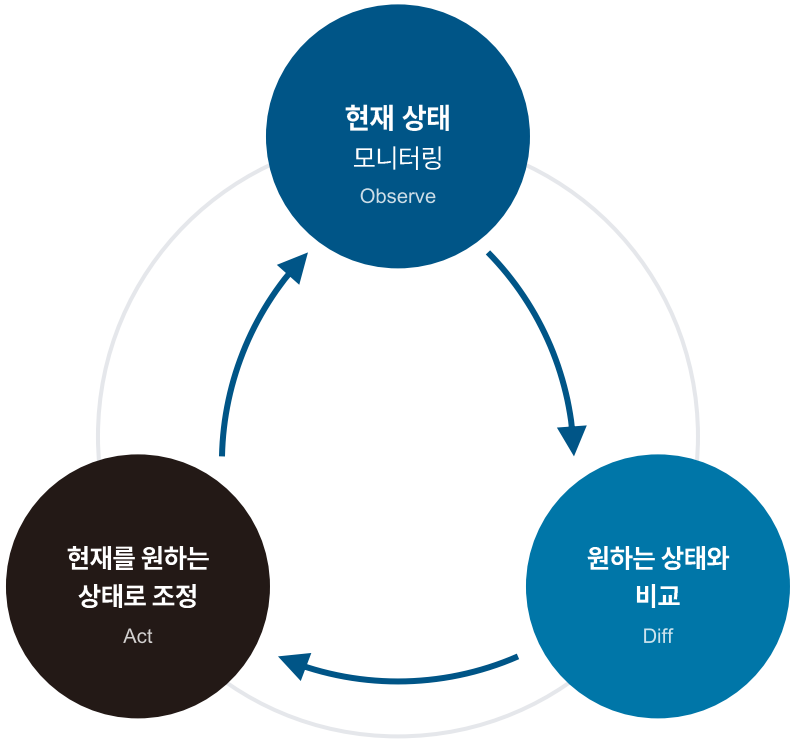
장애 감지 → 복구 작업(버그 수정·재구성) → 상태 확인

Source: 원본 교육자료(Slide 53) 재구성

Control Loop — 컨트롤러의 조정 원리

KEY INSIGHT 컨트롤러는 현재 상태(actual)를 원하는 상태(desired)로 맞추기 위해 관찰·비교·조정을 끝없이 반복한다.

Control Loop (제어 루프)



Desired State vs Actual State

Desired State
원하는 상태

"Pod 5개를 유지하라"
— 매니페스트에 선언

Actual State
현재 상태

"지금 Pod 3개 실행 중"
— 클러스터 관찰

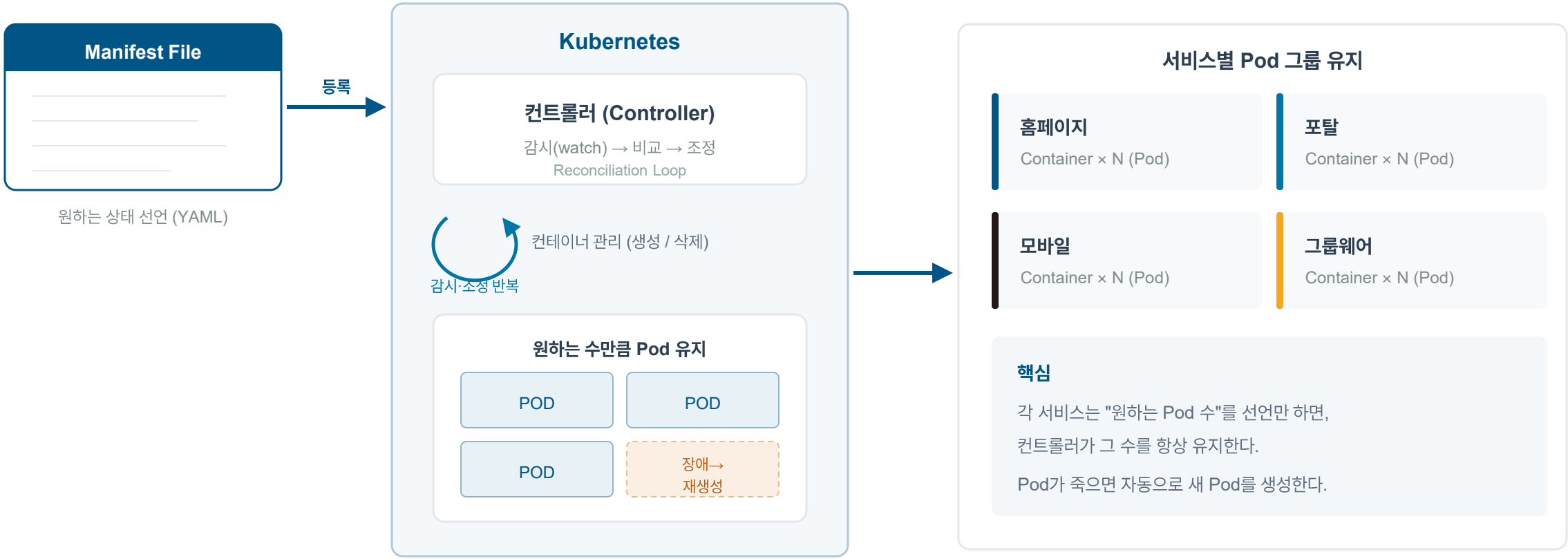
컨트롤러가 하는 일

- 클러스터 상태를 관찰(watch)한다
- 원하는 상태와 현재 상태의 차이를 계산한다
- 차이를 없애도록 생성·삭제·변경을 요청한다
- actual = desired 가 될 때까지 무한 반복한다

Source: 원본 교육자료(Slide 54) 재구성 · kubernetes.io

08 Reconciliation Loop의 동작

KEY INSIGHT Manifest를 등록하면 Kubernetes가 감시하며 Pod를 생성·삭제해 선언한 상태를 유지한다.



Source: 원본 교육자료(Slide 55) 재구성

컨테이너 수 조정: Observe → Analyze → Act

KEY INSIGHT 원하는 컨테이너 수 5, 현재 2 → 관찰하고, 부족분을 계산하고, 3개를 생성해 5로 맞춘다.

① 현재 상태 확인 (Observe)

원하는 상태 — 컨테이너 수:**5**

현재 상태 — 컨테이너 수:**2**

현재 몇 개인지 관찰

클러스터의 실제 상태를 확인

② 비교 (Analyze)

원하는 상태:**5**· 현재:**2**

차이(diff) 계산

3개 부족

원하는 상태와 비교

"5 − 2 = 3개를 더 만들어야"

③ 일치 (Act)

원하는 상태:**5**· 현재:**5**

↑ 새로 생성된 3개

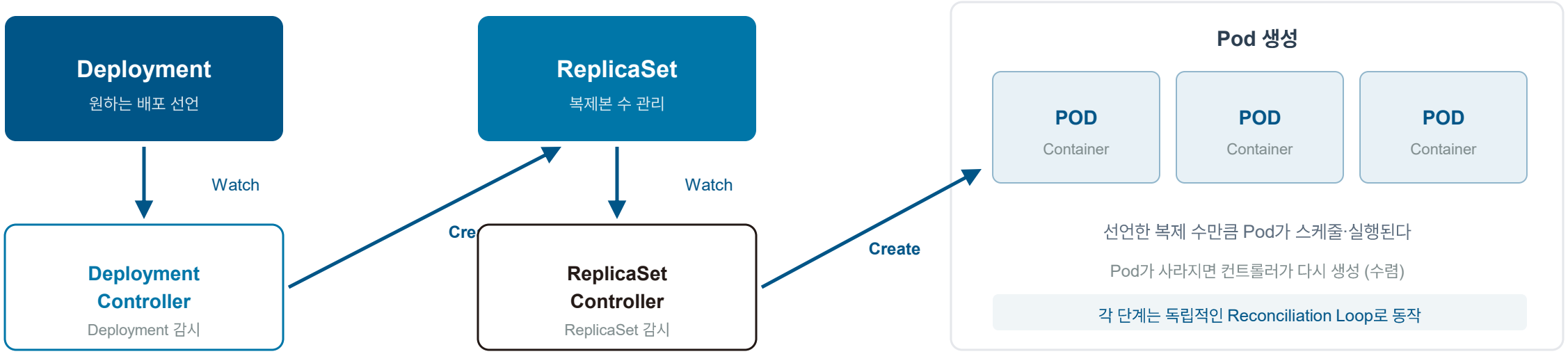
✓ **상태 일치**

부족분을 생성해 5개로

actual = desired 달성 → 루프 반복

Source: 원본 교육자료(Slide 56) 재구성

KEY INSIGHT 여러 컨트롤러가 각자의 리소스를 감시하며 단계적으로 조정한다 — Deployment → ReplicaSet → Pod.



컨트롤러의 역할

클러스터 리소스를 감시하고,
원하는 상태를 유지하도록 조정하는
핵심 구성 요소.

Level-Based API

중간 상태(이벤트)를 무시하고,
항상 최신 Spec을 기준으로
상태를 조정한다.

핵심 구성 요소

Informer · WorkQueue ·
Reconciler 로 효율적·안정적
상태 관리를 수행한다.

Source: kubebuilder book · 원본 교육자료(Slide 57–59)

핵심 정리 — 이것만 기억하세요

KEY INSIGHT 컨테이너로 가볍게, 불변으로 안전하게, Kubernetes로 자동으로, 오픈 표준으로 종속 없이.

1 컨테이너 = 이식성 · 일관성

Containers

게스트 OS 없이 커널을 공유해 가볍고,
개발부터 운영까지 어디서나 동일하게 돈다.

가상화의 이점 + 낮은 오버헤드

2 불변 인프라 = 교체의 미학

Immutable Infrastructure

서버를 고치지 말고, 이미지를 새로 만들어
교체한다 — Pets가 아니라 Cattle처럼.

Drift 없음 · 재현 가능 · 무중단 롤백

3 Kubernetes = 표준 + 자가 조정

Orchestration & Reconciliation

원하는 상태를 선언하면, Reconciliation
Loop가 그 상태를 끊임없이 유지한다.

자동 확장 · 자가 치유 · 무중단 배포

4 오픈 · 벤더 중립 = 종속 회피

Open Governance

CNCF 거버넌스와 표준 인터페이스
(CNI · CRI · CSI)로 특정 공급자에 묶이지 않는다.

Open by design · Interoperable

Source: 본 자료 종합 (Cloud Native Forum)

THANK YOU

감사합니다

Cloud Native, Open by Design.

Cloud Native Forum

Container Orchestration Platform

https://cncf.co.kr

hello@cncf.co.kr

PUBLIC