Cloud Native Forum

Cloud Native Forum

몰라서 어려운 쿠버네티스 손이 많이 가는 기존 운영

두 가지 어려움의 성격 구분

PROJECT CODE
K8S-DRAWBACKS

DATE
2026-09

PUBLIC

CONTENTS

오늘 확인하는 두 가지

01

쿠버네티스, 알면 쉬운 기술

엘리베이터처럼 사용법을 알면 쉽고

모르면 어렵게 느껴지는 기술

02

기존 운영 방법의 한계

하드웨어 가상화라 새로 배울 것은 없지만

수작업이고 절차 개선이 어려운 구조

KEY FOCUS 어려움의 성격 구분 — 배우면 사라지는 것과 그대로 남는 것

CHAPTER 01

엘리베이터를 몰라 계단으로 가는 쿠버네티스, 알면 쉬운 기술

학습과 적응으로 해소되는 항목을 순서대로 확인

01 쿠버네티스 6가지 단점 – 시차에 따라 적응이 필요

KEY INSIGHT 여섯 가지가 동시에 오지 않음 — 초기 검토만으로는 총량이 보이지 않음

학습 곡선 · YAML 설정

개념 계층을 아래층부터 익히고
설정 파일을 쓰는 구간

도입 0~3개월

디버깅 방식 전환

서버 시절 장애 대응 절차가
그대로 무효가 되는 구간

도입 3~6개월

운영 비용 · 인력 의존

예산 초과와 담당자 공백이
함께 드러나는 구간

도입 6개월 이후

규모 불일치

처음 선택이 어긋났다면
내내 부담으로 남는 항목

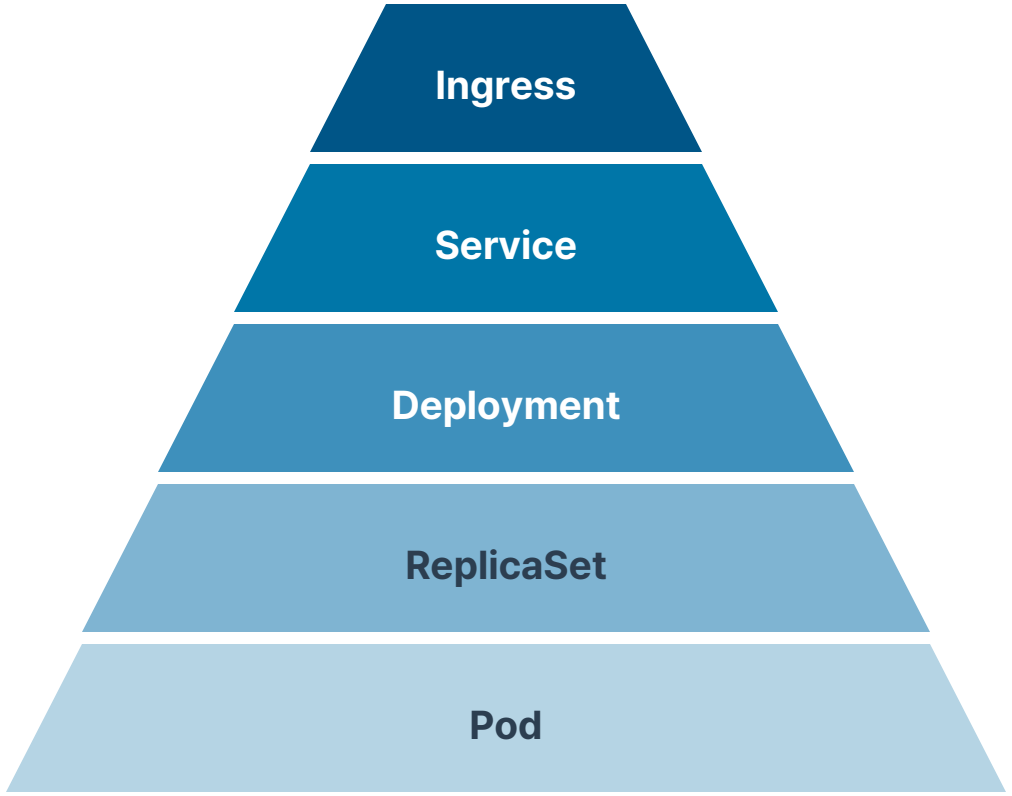
시점 무관

도입 검토 자리에서 보이는 것은 앞의 두 칸뿐 — 뒤의 네 항목은 시간이 지나야 드러남

Source: 에스코어 · arp242.net · CIO Korea 종합

쿠버네티스 개념 5계층 – 순서대로 쌓아야 하는 학습 구조

KEY INSIGHT 순서를 바꿀 수 없어 학습 기간이 압축되지 않음 — 집중해도 2~3개월 필요



5층 Ingress
외부 요청을 규칙에 따라 내부 Service 로 전달

4층 Service
바뀌는 Pod 주소를 고정된 이름 하나로 연결

3층 Deployment
ReplicaSet 을 갈아 끼우며 버전을 전환

2층 ReplicaSet
같은 Pod 을 정해진 수만큼 유지

1층 Pod
컨테이너가 실제로 도는 최소 단위

⚠ 가장 흔한 함정 — 쿠버네티스를 기능 많은 도커로 이해한 채 2층부터 시작

Source: 용찬호, 시작하세요! 도커/쿠버네티스 (위키북스) · 에스코어

학습 기간 2~3개월은 집중 투입 기준. 도커 경험이 출발점을 앞당기지 못한다는 점이 공통 지적

YAML은 띄어쓰기 한 칸만 틀려도 배포 전체가 오류

KEY INSIGHT 한 파일의 들여쓰기 오류가 배포 전체를 멈춤 — 파일이 늘면 생성 도구가 또 필요

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
spec:
    replicas: 3
        image: myapp:1.0
```

← 한 칸 더 들어감

replicas 의 하위 항목으로 해석되어 배포 전체 거절

무인 주문기와 같은 구조

메뉴 이름을 한 글자만 틀려도 주문 자체가 반려되고, 무엇이 틀렸는지는 따로 찾아야 함

서비스가 늘면 파일도 늘어남

서비스 수십 개 → 설정 파일 수백 장 → 파일을 찍어내는 도구(Helm 템플릿)가
다시 학습 대상으로 추가

설정을 파일로 적어 제출

화면 클릭으로 끝나지 않고
형식 규칙을 지킨 파일이 입력

관리 대상이 파일 수백 장

한 서비스당 여러 파일이라
변경 추적이 별도 업무로 분리

업계 통칭 YAML 지옥

설정 작성과 검증에 드는 시간이
개발 시간을 넘어서는 구간

Source: 쿠버네티스의 단점은 무엇인가요 (OMDM, 2026-09)

01 상태가 없는 구조에 대한 이해 – 파일을 수정해도 원복되는 특성

KEY INSIGHT 기존 서버 대응 절차가 그대로 무효 — 로그 반출 설정이 장애보다 먼저 필요

기존 서버 습관

1. 서버에 직접 접속
2. 파일을 그 자리에서 수정
3. 프로세스 재기동

수정분이 사라짐
새 Pod 이 뜨는 순간 원래 이미지로 되돌아감

쿠버네티스에서의 경로

1. 컨테이너 이미지 재빌드
2. 매니페스트 파일 갱신
3. 배포 명령 실행

변경이 유지됨
새 Pod 으로 교체되어 다음 배포에도 남음

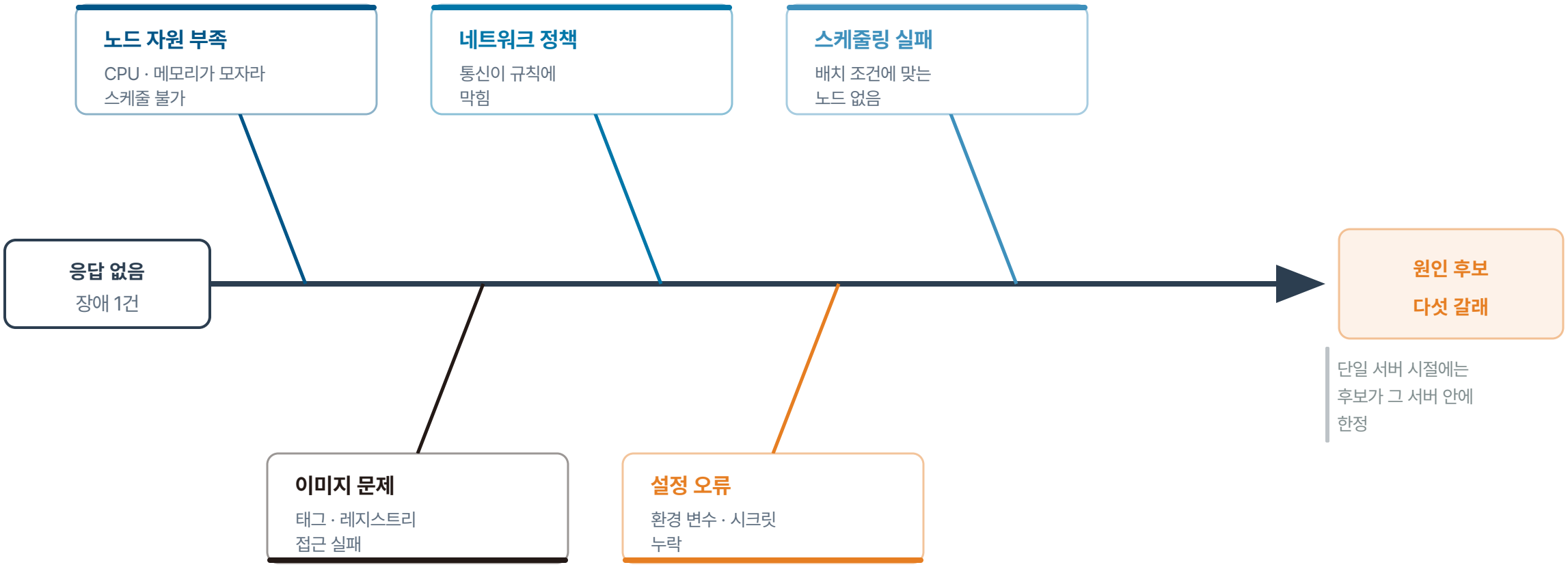
Pod 이 사라지면 그 안의 로그도 함께 사라짐
원인을 남기려면 로그를 클러스터 밖으로 내보내는 설정이 장애보다 먼저 있어야 함

Source: arp242, You (probably) don't need Kubernetes

Pod 은 버리고 새로 찍는 물건 — 고쳐 쓰지 않는 것이 클라우드 네이티브 통용 원칙

여러 업무를 함께 운영하는 클라우드에서 문제 파악을 위한 Observability는 필수

KEY INSIGHT 도입과 같은 시점에 관측 도구가 필요 — 없으면 복구 시간이 통째로 늘어남



Source: DevOps Pink, Is Kubernetes Overkill?

한 문제가 여러 갈래로 나뉜 상태 — 로그 · 지표 · 추적(metrics · logs · traces) 수집이 대응의 전제

01 단점 여섯 가지는 학습과 적응을 통해 해결하는 것이지 문제점이 아님

KEY INSIGHT 여섯 가지가 같은 조건에서 오지 않음 — 해당하는 행만 골라 읽으면 됨

단점	한 줄 비유	언제 문제가 되나
가파른 학습 곡선	층을 건너뛸 수 없는 빌딩	팀에 경험자가 없을 때
YAML	한 글자 틀리면 반려되는 주문기	서비스가 늘어날수록
어려워진 디버깅	범인이 치워진 뒤 도착하는 현장	장애가 났을 때
끝나지 않는 운영비	분양비보다 큰 사룟값	도입 6개월 이후
인력 의존과 보안	운전자 없는 스포츠카	전담 인력이 빠질 때
과잉	마트 주차장의 페라리	서비스가 작을 때

Source: 쿠버네티스의 단점은 무엇인가요 (OMDM, 2026-09)

우리 팀에 해당하는 행이 몇 개인지가 도입 시점 판단의 입력값

CHAPTER 02

새로 배울 것은 없지만 손으로 해야 하는 기존 운영 방법의 한계

수작업 운영과 개선하기 어려운 절차를 확인

KEY INSIGHT 단점 절반의 책임은 분산 시스템 자체 — 값을 지금 치를지는 별개 질문

흔한 오해

쿠버네티스를 안 쓰면 복잡함이 사라짐

도구를 걷어내면 그 도구가 만든
복잡함도 함께 사라진다는 기대

이 기대가 맞으려면

무중단 배포도 자동 복구도 필요 없는
서비스여야 함. 그렇다면 애초에
도입 대상이 아니었던 것

CNCF 의 반론

쿠버네티스를 안 쓰면 같은 복잡함을 직접 구현

아래 여덟 가지는 도구를 바꿔도 남는
서비스 운영 요구사항

- 무중단 배포
- 자동 복구
- 로드밸런싱
- 서비스 디스커버리
- 헬스체크
- 롤백
- 시크릿 관리
- 오토스케일

Source: CNCF, Too Complex: It's Not Kubernetes, It's What It Does (2026-03)

손으로 만든 구현이 더 복잡하고 더 잘 깨진다는 것이 반론의 근거

서버 운영 업무의 소멸이 아닌 클러스터로의 이동

KEY INSIGHT 도입 효과를 인력 감축으로 계산하면 빗나감 — 서버 담당을 줄이면 클러스터 운영이 공백으로 남음

줄어드는 일 — 서버 쪽

- 서버마다 직접 설치
- 배포 때마다 수동 작업
- 장애 시 사람이 재기동
- 용량 증설을 손으로 계산

늘어나는 일 — 클러스터 쪽

- 클러스터 버전을 주기마다 올림
- YAML 설정 파일을 계속 유지
- 관측 도구를 따로 구축
- 권한 체계를 직접 운영

양쪽 항목 수가 비슷 — 단점을 묻는 질문의 답은 오른쪽 칸에 집중

Source: 쿠버네티스의 단점은 무엇인가요 (OMDM, 2026-09)

쿠버네티스 자동 운영 기능 8가지 vs. 기존 수동 운영

KEY INSIGHT 단점 절반은 이 기능들의 값 — 걷어내도 요구는 남아 직접 구현 부담으로 되돌아옴

무중단 배포

구버전을 내리기 전에
신버전을 띄워 요청 유지

자동 장애 복구

죽은 파드를 감지해
같은 수만큼 새로 생성

로드밸런싱

들어온 요청을 살아 있는
파드로 나눠 전달

서비스 디스커버리

주소가 바뀌어도 같은
이름으로 찾아가게 연결

서비스 헬스체크

응답 못 하는 파드를
트래픽 대상에서 제외

롤백

직전 정상 버전을
한 번에 되돌림

시크릿 관리

비밀번호와 토큰을
코드 밖에 분리 보관

자동 확장/축소

부하에 따라 파드 수를
자동으로 늘리고 줄임

Source: CNCF, Too Complex: It's Not Kubernetes, It's What It Does (2026-03)

KEY INSIGHT 설치는 사람 없이 되지만 운영은 되지 않음 — 담당자 공백이 곧 접근 통제 공백

역할 기반 접근 제어(RBAC, Role-Based Access Control)

RBAC

누가 무엇을 할 수 있는지 정하는 설정. 초기값을 그대로 두면 필요 이상의 권한이 남음

클러스터 접속 자격증명 파일(kubeconfig)

kubeconfig

한 번 공유되면 회수가 어렵고, 파일 하나가 클러스터 전체 접근 권한과 같은 무게

담당자 이탈 시 남는 것

인수인계

설정 파일은 남지만 왜 그렇게 정했는지는 남지 않아 다음 사람이 처음부터 다시 판단

⚠ 가장 비싼 순서 — 먼저 설치하고 담당자는 나중에 구하는 방식

Source: 쿠버네티스의 단점은 무엇인가요 (OMDM, 2026-09)

클라우드 환경이라 단일 시스템이 아닌 여러 시스템 운영에 적합

KEY INSIGHT 크기 적합성이 가르는 도입 성패 — 작은 서비스에서는 얻는 것이 부담을 못 넘김

	작은 서비스	큰 서비스
변동 큼	**규모 작음 · 변동 큼** 관리형 컨테이너 서비스가 중간 선택지 클러스터를 직접 떠안지 않고도 자동 증감 확보	**규모 큼 · 변동 큼** 쿠버네티스가 치른 값을 회수하는 구간 무중단 배포와 오토스케일이 매일 쓰임
변동 작음	**규모 작음 · 변동 작음** 관리형 호스팅으로 충분한 구간 도입하면 얻는 것 없이 운영 부담만 추가	**규모 큼 · 변동 작음** 일부 서비스만 부분 도입해 볼 구간 전면 전환은 비용 회수까지 시간이 걸림

자주 인용되는 비유 — 차고가 낮아 동네 마트 주차장을 빠져나오지 못하는 페라리

Source: arp242.net · DevOps Pink, Is Kubernetes Overkill?

PaaS (Platform)가 제공하는 기능을 통한 운영 효율화

KEY INSIGHT 세 질문이 모두 예일 때만 값이 회수됨 — 하나만 예면 값만 치르게 됨

1

서비스 수와 배포 빈도가 수작업 한계를 넘었는가

손으로 배포할 수 있는 양이면 자동화 대상이 아직 아님

2

클러스터를 맡을 담당자가 지금 있는가

설치는 사람 없이 되지만 운영은 담당자 없이는 멈춤

3

관측 도구와 업그레이드를 감당할 예산이 있는가

연 3회 업그레이드와 로그 · 지표 수집이 상시 항목

세 질문 모두 예

도입 검토를 진행할 만한 구간 — 치르는 값을 운영에서 회수

단점 여섯 가지는 그대로 오지만 얻는 것이 그보다 큼

하나라도 아니오

관리형 서비스나 기존 배포 방식 유지가 더 맞는 구간

아니오인 항목을 먼저 해소하는 것이 도입보다 앞선 순서

Source: 쿠버네티스의 단점은 무엇인가요 (OMDM, 2026-09)

THANK YOU

몰라서 어려운 것과 알아도 손이 가는 것의 구분

쿠버네티스는 배우면 쉬워지는 기술이고, 기존 운영 방법은 익숙하지만
수작업과 절차 개선의 한계가 남는 구조

Cloud Native Forum

Cloud Native Forum

https://cncf.co.kr

hello@cncf.co.kr

PUBLIC