원본 2015 년 8 월 24 일 · 채문창 → 현행화 2026 년 11 월

SaaS · DevOps · Container, 2026 년 다시 쓰기

바이브 코딩과 에이전트 AI 시대에 클라우드 서비스를 만들고 운영하는 법

2015 → 2026

원칙은 그대로, 도구와 '누가 코드를 쓰는가'가 바뀌었습니다

11 년 동안 바뀐 것 , 한 장으로

원래 강의의 순서 그대로 — 왼쪽이 2015 년 , 오른쪽이 2026 년입니다

주제	2015	2026
클라우드 모델	IaaS · PaaS · SaaS	**+ 서버리스 · AI 모델 API · 에이전트가 쓰는 SaaS**
개발 규범	The Twelve-Factor App	**12-Factor 오픈소스 개정 (2024) + 12-Factor Agents(2025)**
컨테이너	Docker · CoreOS · fleet	**OCI 표준 · Kubernetes 가 공통 기반**
개발 환경	Vagrant 가상머신	**Dev Containers · 에이전트용 샌드박스**
배포	Ansible 스크립트 · 수동 staging	**IaC + GitOps · 카나리 · 자동 롤백**
테스트	단위 테스트 · Selenium	**+ Playwright E2E · LLM 기능 평가 (eval)**
개발자	사람이 코드를 씀	**사람이 설계 · 검증하고 에이전트가 코드를 씀**

가장 큰 변화 : 자동화의 대상이 '빌드 · 배포' 에서 '개발 그 자체' 로 넓어졌습니다

누가 무엇을 관리하나 : 2026 년판 책임 분담

2015 년의 4 단 (On-Premise · IaaS · PaaS · SaaS) 에 서버리스와 AI 모델 API 가 더해졌습니다

On-Premise	IaaS	PaaS	Serverless	SaaS	AI 모델 API
AI 기능	AI 기능	AI 기능	AI 기능	AI 기능	AI 기능
Applications	Applications	Applications	Applications	Applications	Applications
Data	Data	Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Servers · Storage · Network	Servers · Storage · Network	Servers · Storage · Network	Servers · Storage · Network	Servers · Storage · Network	Servers · Storage · Network

우리가 관리 공급자가 관리

AI 모델 API: 모델은 빌려 쓰고 , 그 위의 앱과 데이터 (프롬프트 · 문서 · 로그) 는 우리 책임

클라우드 컴퓨팅 계층 , 무엇이 더해졌나

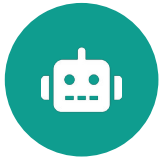
맨 위의 '사용자' 에 사람뿐 아니라 AI 에이전트가 들어왔습니다

	2015	2026 에 더해진 것
Clients	웹 브라우저 · 모바일 앱 · 씬 클라이언트 · 터미널	+ 코딩 에이전트 · 업무 에이전트 · MCP 클라이언트 (사람 대신 API 를 부름)
SaaS	CRM · 이메일 · 가상 데스크톱 · 커뮤니케이션 · 게임	+ AI 기능이 기본인 SaaS: 요약 · 검색 · 초안 · 업무 대행
PaaS	실행 런타임 · 데이터베이스 · 웹 서버 · 개발 도구	+ 모델 API · 벡터 DB · 에이전트 실행 플랫폼 · 서버리스
IaaS	가상머신 · 서버 · 스토리지 · 로드밸런서 · 네트워크	+ GPU 클러스터 · AI 가속기 · 대용량 추론 인프라

에이전트가 사용자가 되면 , 화면보다 API · 권한 · 감사 로그가 먼저 설계 대상이 됩니다

SaaS 는 어떻게 달라지고 있나

확정된 통계가 아니라 업계에서 뚜렷하게 보이는 흐름 네 가지입니다



사용자가 에이전트로

화면을 클릭하는 사람 대신 API · MCP 로 부르는 에이전트가 늘어남 → '에이전트가 쓰기 좋은 SaaS' 가 경쟁력



도구에서 '일 대행' 으로

기능을 제공하던 SaaS 가 요약 · 분류 · 응답 같은 업무 자체를 처리 (Service-as-Software 라는 표현도 쓰임)



과금 방식의 변화

좌석 (사람 수) 과금에 더해 사용량 · 토큰 · 처리 건수 기반 과금이 늘어남 → 원가에 모델 비용이 들어옴



1 인 SaaS 의 현실화

바이브 코딩 · 코딩 에이전트로 1 인이 만들고 운영하는 SaaS 가 늘어남 → 대신 운영 · 보안 책임도 1 인에게

2026 년 클라우드 · AI 지형 (예시)

2015 년의 ‘Cloud Companies’ 지도를 범주 중심으로 다시 그렸습니다 — 순위나 추천이 아닙니다

하이퍼스케일러

AWS · Microsoft Azure · Google Cloud

국내 클라우드

네이버클라우드 · KT Cloud · NHN Cloud 등

AI 모델 제공

Anthropic · OpenAI · Google · 국내 LLM · 오픈 웨이트 모델

코드 · 협업 · CI

GitHub · GitLab · Gitea(자체 구축)

컨테이너 · 오케스트레이션

Kubernetes · containerd · Docker · OpenShift

관측성 · 운영

OpenTelemetry · Prometheus · Grafana · Datadog

2015 년 지도의 많은 회사는 인수 · 합병 · 서비스 종료로 사라졌습니다 — 지도는 금방 낡고, 범주와 원칙이 오래 갑니다

국내에서는 : 클라우드냐 온프렘이냐보다 '데이터 등급'

국가망보안체계 (N²SF) — 망분리를 없애는 게 아니라 , 등급에 따라 AI · 클라우드를 쓸 길을 여는 체계

C **기밀**
엄격한 보호가 필요한 정보 → **내부망 · 온프레미스 AI**

S **민감**
내부 공유는 되나 외부 공개는 어려운 정보 → **추가 통제 후 제한적 활용**

O **공개**
대외 배포가 가능한 정보 → **외부 생성형 AI · 클라우드 활용**

2026 년 현황

- 2025 년 실증으로 적용 모델 확보
- 2026 년 검증된 정보서비스 모델 6 종을 공공기관에 도입 지원
- 그중 하나가 업무환경의 생성형 AI 활용
- AI 이용 로그 · 유출 방지 (AI-DLP) 검증

개발자에게 주는 뜻 같은 서비스도 데이터 등급에 따라 '외부 모델 API' 와 '온프레미스 모델 (vLLM 등으로 서빙하는 오픈 웨이트 모델)' 을 나눠 써야 합니다 . 그래서 모델을 교체 가능한 백엔드 서비스로 설계하는 것 (→ 12-Factor IV) 이 2026 년에 더 중요해졌습니다 .

The Twelve-Factor App, 2026 년에 다시 읽기

12 개 원칙은 지금도 유효합니다 — 바뀐 것은 원칙을 지키는 도구입니다

I 코드베이스

Git 하나 · 여러 배포 (그대로)

II 종속성

lock 파일 · 컨테이너 이미지

III 설정

환경 변수 + 비밀 관리 도구

IV 백엔드 서비스

+ AI 모델 API · 벡터 DB

V 빌드 · 릴리즈 · 실행

CI/CD · 서명된 이미지

VI 프로세스

무상태 컨테이너 (파드)

VII 포트 바인딩

그대로 · 서비스 · 인그레스

VIII 동시성

오토스케일 (HPA · KEDA)

IX 폐기 가능

롤링 업데이트 · 그레이스풀 종료

X Dev/Prod 일치

Dev Containers · 미리보기 환경

XI 로그

OpenTelemetry (로그 · 메트릭 · 트레이스)

XII Admin 프로세스

Job · CronJob · 일회성 컨테이너

주황 테두리 = 도구가 바뀐 원칙 · 회색 = 사실상 그대로

12-Factor 는 2024 년 오픈소스로 다시 고쳐지는 중

만든 지 13 년 , ‘ 핵심은 유효하지만 세부가 낡아 보인다’ 는 판단에서 시작된 개정

2011
Heroku 개발자들이 12-Factor 발표

2015
우리 강의 : SaaS 개발 규범으로 소개

2024 · 11
12-Factor 오픈소스화 발표 · 커뮤니티 개정 시작

2025 ~
12-Factor Agents 등 AI 앱용 파생 원칙 등장

개정에서 다루는 방향

컨테이너 기반 배포

2011 년엔 막 등장하던 방식을 원칙 안으로

서버리스 · 클라우드 네이티브

플랫폼이 대신해 주는 부분이 커진 환경

앱과 플랫폼 사이의 인터페이스

앱이 플랫폼에 무엇을 약속하고 기대하나

수업에서는 : 원문 (12factor.net) 과 GitHub 저장소의 토론을 함께 읽고 , ‘ 우리 서비스에 맞지 않는 원칙’ 을 찾아보게 하면 좋습니다

AI 앱을 위한 파생 원칙 : 12-Factor Agents (2025)

HumanLayer 가 정리한 원칙 — ‘프레임워크의 마법’ 보다 프롬프트 · 컨텍스트 · 제어 흐름을 직접 소유하라

직접 소유하기

1. 자연어를 도구 호출로
2. 프롬프트를 직접 소유
3. 컨텍스트 윈도우를 직접 소유
8. 제어 흐름을 직접 소유

상태와 실행

4. 도구는 구조화된 출력일 뿐
5. 실행 상태와 업무 상태를 하나로
6. 단순한 API 로 시작 · 멈춤 · 재개
12. 에이전트는 무상태 리듀서로

사람 · 오류 · 크기

7. 도구 호출로 사람에게 연락
9. 오류를 압축해 컨텍스트에
10. 작고 집중된 에이전트
11. 어디서나 트리거 , 사용자가 있는 곳에서

원래 12-Factor 와 같은 정신 : 무상태 · 명시적 상태 · 작게 나누기 · 어디서나 같은 방식으로 실행

우리 과정이 덧붙이는 실무 원칙 6 가지

12-Factor + 12-Factor Agents 를 실제 서비스 운영 관점에서 정리한 제안입니다

모델은 교체 가능한 백엔드 서비스

게이트웨이 (LiteLLM 등) 뒤에 두고, 외부 API ↔ 온프레미스 모델을 설정으로 바꾼다

프롬프트 · 평가셋도 코드

프롬프트와 시나리오 (기대 결과) 를 Git 에서 버전 관리하고 리뷰한다

비밀은 금고에, 권한은 최소로

키는 비밀 관리 도구에만, 에이전트 도구 권한은 필요한 만큼만

관측성에 토큰 · 비용 · 품질

지연 시간과 오류율에 더해 요청당 토큰 · 비용 · 근거 여부를 기록한다

사람 승인 지점을 설계에 넣기

되돌릴 수 없는 일 (송금 · 삭제 · 배포) 은 사람이 승인한다

정해진 방식으로 실패하기

AI 결과를 못 읽거나 근거가 없으면 안전한 쪽 (이관 · 중단) 으로 (fail-closed)

컨테이너 : 2015 년의 약속은 지켜졌습니다

‘ 어떤 화물이든 같은 상자에 담아 어디로든 옮긴다’ — 지금은 사실상 모든 클라우드의 기본 단위

2013	2015 · 6	2015 · 7	2018	2020	2026
Docker 공개	OCI 결성 (컨테이너 표준)	Kubernetes 1.0 · CNCF	Red Hat, CoreOS 인수	CoreOS Container Linux 지원 종료	에이전트도 컨테이너 안에서 일함

← 우리 강의 (2015 년 8 월) 는 OCI · Kubernetes 1.0 직후였습니다

단위는 그대로

앱 · DB · 캐시 · 큐를 컨테이너 단위로 빌드 · 배포

관리자는 바뀜

fleet · CoreOS 대신 Kubernetes 가 표준 오케스트레이터

새 거주자

코딩 에이전트 · 모델 서버 · MCP 서버도 컨테이너로

2015 년 스택 → 2026 년 스택

같은 문제를 푸는 도구가 어떻게 바뀌었나

하는 일	2015	2026
컨테이너 전용 OS	CoreOS Container Linux	**Fedora CoreOS · Bottlerocket · Talos 등**
오케스트레이션	fleet · etcd	**Kubernetes (etcd 는 안에서 계속 쓰임)**
서비스 발견 · 부하 분산	confd + nginx	**Service · Ingress · Gateway API · 서비스 메시**
개발 · QA 환경	Vagrant 가상머신	**Dev Containers · Docker Compose · 클라우드 개발환경**
인프라 구성	Ansible	**Terraform / OpenTofu (IaC) + Ansible**
배포	스크립트 · 수동 staging	**GitOps (Argo CD · Flux) · 카나리 배포**
이미지 신뢰	—	**이미지 서명 (Sigstore) · SBOM · 취약점 스캔**

참고 : Vagrant 를 만든 HashiCorp 는 2023 년 라이선스를 BSL 로 바꿨고 2025 년 IBM 에 인수되었습니다

Container vs VM, 2026 년에는 선택지가 넷

에이전트가 만든 코드를 실행하는 일이 늘면서 ‘얼마나 강하게 격리하나’가 다시 중요해졌습니다

가상머신 (VM)

OS 째로 분리

격리 강도 ●●●●

시작 속도 ●○○○

주로 쓰는 곳

다른 OS · 강한 격리가 필요한 업무 · 레거시

마이크로 VM

VM 격리 + 빠른 시작

격리 강도 ●●●●

시작 속도 ●●●○

주로 쓰는 곳

서버리스 함수 · 남이 쓴 코드 · 에이전트 샌드박스

컨테이너

커널 공유 · 가벼움

격리 강도 ●●○○

시작 속도 ●●●●

주로 쓰는 곳

일반 앱 · API · 대부분의 서비스

WebAssembly

더 작고 빠른 실행 단위

격리 강도 ●●○○

시작 속도 ●●●●

주로 쓰는 곳

엣지 · 플러그인 · 일부 서버리스

원칙 : 내가 쓴 코드는 컨테이너로 , 남 (또는 AI) 이 쓴 검증 안 된 코드는 더 강한 격리 안에서

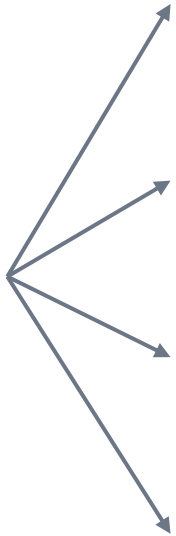
개발 환경 : Vagrant 에서 Dev Containers 로

‘개발 · QA · 운영 환경을 같게’ 라는 2015 년 목표는 그대로 , 단위가 VM 에서 컨테이너로

팀 저장소

```
.devcontainer/
  devcontainer.json
  Dockerfile
compose.yaml
AGENTS.md
src/ · tests/ · cases/
```

환경도 코드로 저장



내 노트북 — VS Code 등에서 같은 컨테이너로 열기

클라우드 개발환경 — 브라우저에서 같은 환경 바로 시작

코딩 에이전트 — 같은 컨테이너 안에서 코드 작성 · 테스트 실행

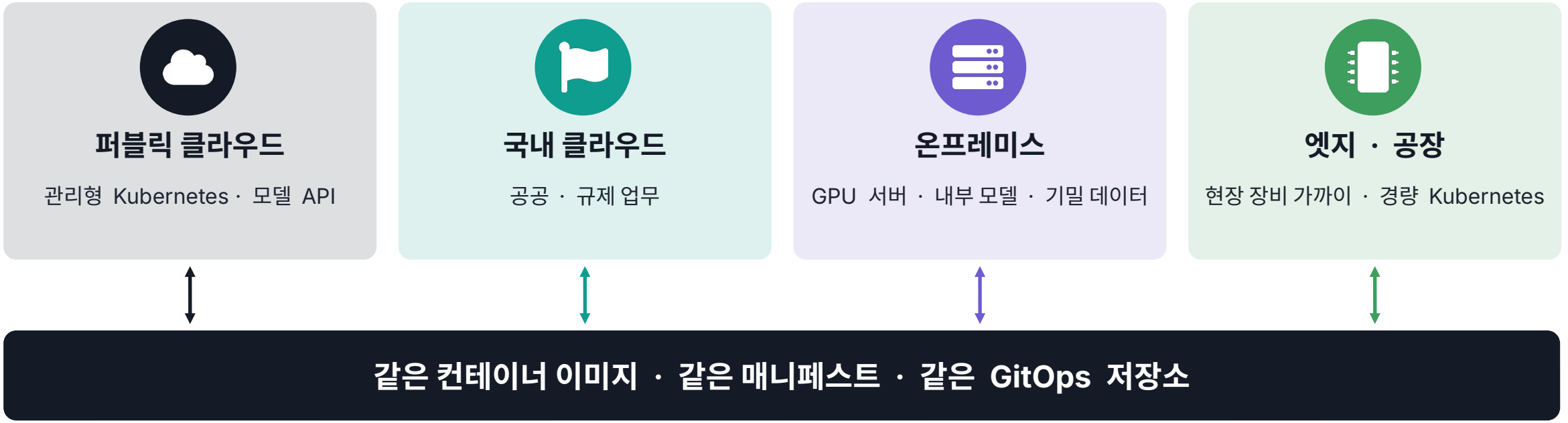
CI — 같은 이미지로 빌드 · 시험

2015 개발자 · QA 마다 Vagrant VM 을 띄우고 운영과 맞추려 애씀

2026 **사람 · 에이전트 · CI 가 ‘같은 이미지’ 를 쓰니 차이가 처음부터 작음**

멀티 클라우드 · 하이브리드 : 공통 언어는 Kubernetes

2015 년 ‘Container - multi-cloud’ 그림의 2026 년판 — 어디서든 같은 방식으로 배포



무엇이 쉬워졌나

같은 이미지와 선언으로 여러 곳에 배포 · 이동

무엇은 여전히 어렵나

데이터 이동 · 네트워크 · 비용 · 기관별 보안 규정

개발 방식의 진화 : 2015 년 그림에 두 단계를 더하면

전통 방식 → TDD → DevOps 까지가 2015 년 강의 , 그 뒤 두 단계가 새로 생겼습니다

2015

전통 방식

개발 · QA · 운영이 따로

2015

개선 (TDD)

테스트를 먼저 쓰는 개발

2015

DevOps

개발 · QA · 운영이 한 고리

자동화로 끊임없이 순환

2026

DevSecOps · 플랫폼

보안 내장 · 내부 개발자 플랫폼

팀은 플랫폼 위에서 셀프서비스

2026

AI 네이티브 개발

사람 + 코딩 에이전트

사람은 설계 · 검증 ,
에이전트는 구현

자동화와 협업의 범위가 넓어지는 방향

DevOps 의 정의는 그대로 , 재는 방법이 생겼습니다

2015 년 정의 (왼쪽) 에 2026 년에 꼭 덧붙여야 할 것 (오른쪽)

2015 2015 년 강의의 정의

- CI 에서 좀 더 진화된 형태
- 개발 · 테스트 · 배포를 모두 자동화해 끊임없이 순환
- 개발 속도를 최대화하는 개발 스타일
- 서비스 연동만큼 필요한 배포 · 테스트

2026 2026 년에 덧붙일 것

- 속도와 안정성을 숫자로 잰다 (DORA 지표)
- 보안을 파이프라인 안에 넣는다 (DevSecOps)
- 팀이 셀프서비스로 쓰는 내부 플랫폼
- AI 는 좋은 관행을 ‘증폭’ 할 뿐 대신하지 않는다

DORA 소프트웨어 전달 지표

배포 빈도
얼마나 자주

변경 리드타임
커밋 → 운영까지

변경 실패율
배포가 장애로

복구 시간
장애 → 복구까지

재작업률
계획 없던 수정 배포

청록 = 처리량 (속도) · 빨강 = 안정성

DORA 2025: AI 는 거울이자 증폭기

전 세계 기술 전문가 약 5,000 명 조사 — 많이 쓰지만 완전히 믿지는 않습니다

90%

AI 도입률

업무에 AI 를 쓰는 비율 (전년 대비 14%p 증가)

80%+

생산성 향상 체감

AI 로 생산성이 올랐다고 응답

24%

높은 신뢰

AI 결과를 높게 · 매우 높게 신뢰 (낮음 · 없음은 30%)

AI 효과를 키우는 7 가지 역량 (DORA AI Capabilities Model)

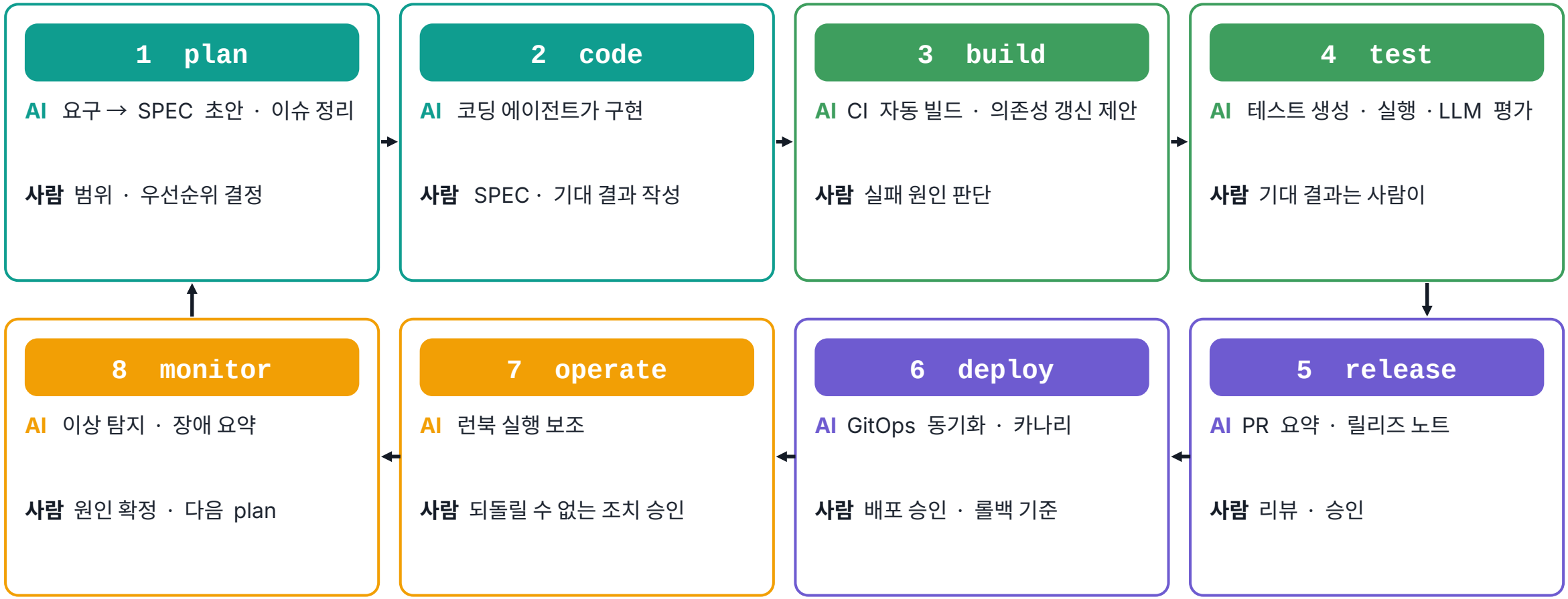
1. 명확하게 공유된 AI 방침
2. 건강한 데이터 생태계
3. AI 가 접근할 수 있는 내부 데이터
4. **강한 버전 관리 관행**
5. **작은 배치로 일하기**
6. 사용자 중심
7. 좋은 내부 플랫폼

주황 = 2015 년 DevOps 에서 이미 강조한 것

협업이 잘 되는 팀에서는 효율을 높이고, 파편화된 팀에서는 기존 약점을 드러냅니다

“Continuous” DevOps, 2026: 고리마다 AI 가 들어왔습니다

8 단계 순환은 그대로 — 각 단계에서 AI 가 맡는 일과 사람이 지키는 일



monitor 의 결과가 다시 plan 의 이슈가 되어 고리가 닫힙니다

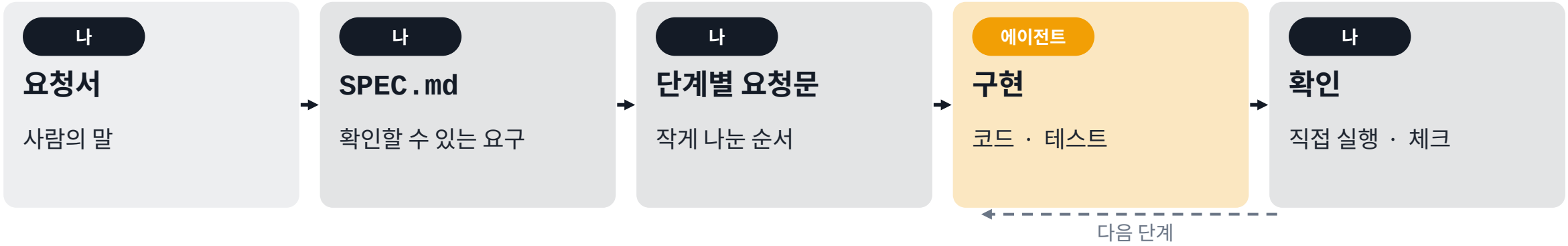
plan · code: 바이브 코딩에서 에이전틱 엔지니어링으로

‘바이브 코딩’ — 2025 년 2 월 Andrej Karpathy 가 쓴 말 , 같은 해 Collins 사전 올해의 단어

자동완성	채팅	바이브 코딩	에이전틱 코딩	백그라운드 · 멀티 에이전트
줄 단위 제안	질문하고 붙여 넣기	결과만 보고 계속 요청	SPEC · 테스트 · 리뷰로 통제	여러 작업을 맡기고 PR 로 받기

맡기는 범위가 커질수록 , 사람이 쓰는 문서 (요구 · 기대 결과) 가 더 중요해집니다

우리 과정의 바이브 코딩 순서



요청 → 요구 , 요구 → 단계는 사람이 합니다 . 에이전트에게는 ‘단계 → 코드’ 만 맡깁니다

build · test: 무엇을 자동으로 시험하나

2015 년 항목은 모두 유효 — 에이전트가 코드를 쓰면서 '기대 결과를 누가 쓰나'가 핵심이 되었습니다

2015 2015 년 강의

- build 자동화 (1 일 n 번), 코드 변경이 빌드를 촉발
- 단위 테스트 · 커버리지
- UI 자동화 테스트 (Selenium)
- 실패하면 리포트 → code 단계로

2026 2026 년

- 모든 PR(에이전트 PR 포함) 마다 빌드 · 시험
- 단위 · 통합 테스트 + E2E (Playwright 등)
- LLM 기능은 평가셋 (eval) 으로 회귀 시험 · 3 회 반복
- 비밀 · 의존성 · 이미지 보안 스캔
- 실패 로그를 에이전트가 읽고 수정 제안

가장 중요한 규칙 **기대 결과 (테스트 · 시나리오) 는 사람이 먼저 쓰고 , 에이전트가 통과시키려고 기대 결과를 고치지 못하게 합니다 . 테스트를 에이전트가 썼다면 사람이 리뷰합니다 .**

release · deploy: 저장소가 곧 운영의 정답

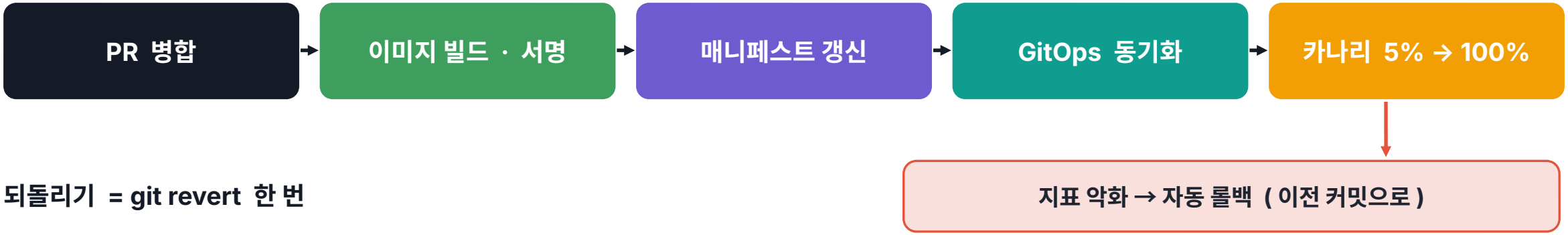
GitOps — 원하는 상태를 Git 에 선언하면 , 클러스터가 그 상태를 따라갑니다

2015 2015 년 강의

- test 성공 뒤 release 자동화
- 사이트별 차이는 설정 정보로 분리
- 개발자는 Vagrant, 배포는 Ansible
- 위치 · 사이트별 staging

2026 2026 년

- 설정은 환경 변수 · 비밀 관리 도구로 (12-Factor III)
- 이미지에 서명하고 SBOM 을 함께 저장
- 인프라는 IaC, 배포는 GitOps
- 카나리 · 피처 플래그로 조금씩 공개
- 지표가 나빠지면 자동 롤백



되돌리기 = git revert 한 번

operate · monitor: AI 기능까지 관측하기

로그 · 메트릭 · 트레이스에 더해 , AI 기능은 비용과 품질도 함께 봅니다

2015 **2015 년 강의**

- 운영 환경 상태 모니터링
- 운영 환경의 자동화된 테스트
- 문제 발견 → 버그 리포팅 툴 → plan
- staging 에 따라 단계적 업그레이드

2026 **2026 년**

- OpenTelemetry 로 로그 · 메트릭 · 트레이스 통합
- SLO 와 에러 버짓으로 배포 속도 조절
- 장애 요약 · 원인 후보는 AI 가 , 확정 · 조치 승인은 사람이
- 관측 결과가 자동으로 이슈 → plan

AI 기능에서 추가로 볼 것

지연

요청당 처리 시간 (p50 · p95)

비용

요청당 토큰 · 모델별 비용

품질

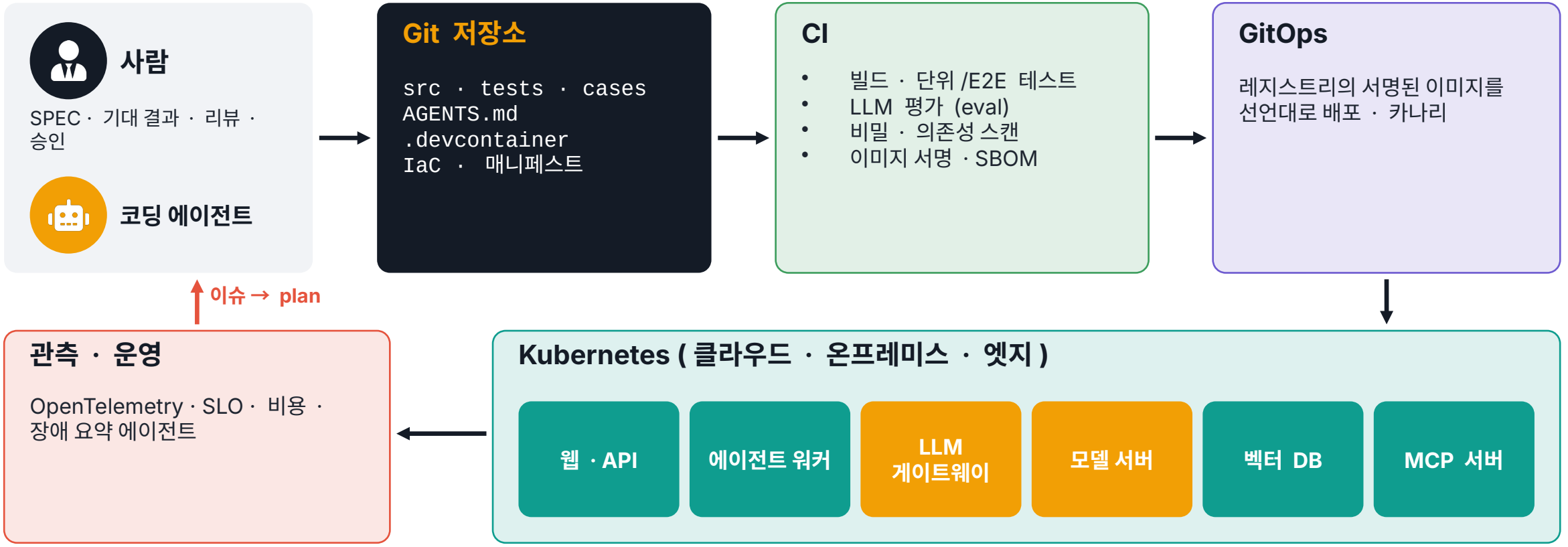
근거 있는 답 비율 · 평가 점수

안전

차단 · 개인정보 가림 · 이상 요청

All together, 2026

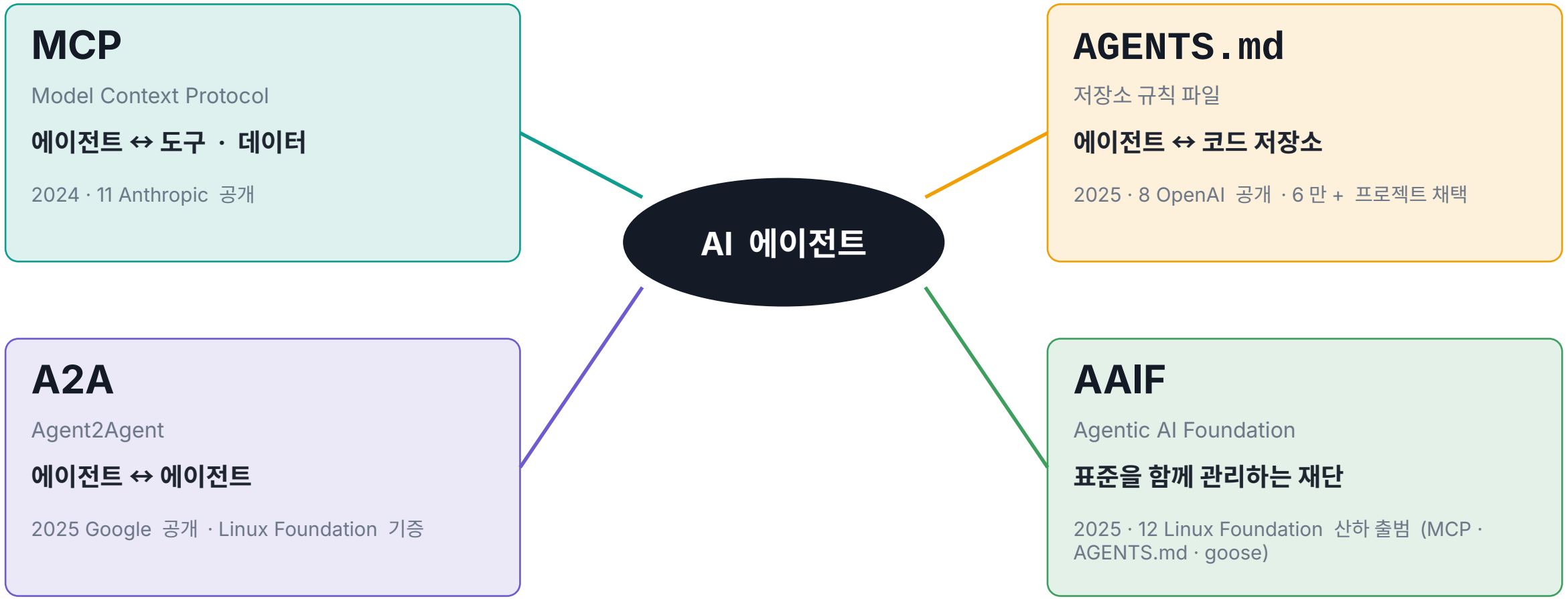
사람과 에이전트가 같은 저장소 · 같은 파이프라인 · 같은 규칙 위에서 일합니다



원본 31 쪽 ‘All together’ 의 2026 년판 — 바뀐 것은 고리 안에 에이전트와 AI 구성요소가 들어왔다는 것

에이전트를 연결하는 표준들

2015 년에 OCI 가 컨테이너를 표준화했듯 , 2025 년에는 에이전트 연결 방식이 표준이 되어 가고 있습니다



주요 회원 : AWS · Anthropic · Block · Bloomberg · Cloudflare · Google · Microsoft · OpenAI (플래티넘)

에이전트가 팀원일 때의 DevOps 규칙 7 가지

새 규칙이 아니라 2015 년 DevOps 규칙을 에이전트에게도 똑같이 적용하는 것

1	**작게 나눈다**	한 번에 한 단계 · 작은 PR (DORA: 작은 배치)
2	**기대 결과를 먼저 쓴다**	테스트 · 시나리오는 사람이 먼저 , 에이전트가 고치지 않게
3	**모든 변경은 PR 로**	리뷰 · CI 를 통과해야 병합 (DORA: 버전 관리)
4	**규칙은 AGENTS.md 에**	작업 순서 · 금지 사항 · 확인 명령을 저장소에
5	**샌드박스 · 최소 권한**	운영 키 · 운영 DB 에 직접 손대지 않게
6	**비밀은 커밋 전에 막는다**	pre-commit 비밀 검사 (예 : tscan staged)
7	**비용을 잰다**	토큰 · 반복 횟수 · 요청당 비용 (예 : tcost)

6 · 7 번 예시는 우리 과정 73 · 75 일차에 바이브 코딩으로 만든 도구입니다

새로 생긴 위험 다섯 가지와 대응

2015 년에는 없던 종류의 사고들 — 대부분 ‘사람의 확인’ 을 건너뛸 때 생깁니다

프롬프트 주입	문서 · 웹 · 이슈 속 숨은 지시를 에이전트가 따름	**도구 권한 최소화 · 외부 입력은 데이터로 취급 · 위험 작업 승인**
공급망 오염	존재하지 않는 패키지 이름을 지어내고 , 공격자가 그 이름으로 악성 패키지 등록	**lock 파일 · 신뢰된 레지스트리 · 의존성 스캔**
이해 부채	동작은 하지만 팀의 누구도 설명하지 못하는 코드가 쌓임	**작은 PR · 리뷰 · 설계 문서 · ‘ 설명할 수 있어야 병합’**
비용 폭주	에이전트 반복 · 긴 컨텍스트로 토큰 비용이 급증	**예산 한도 · 반복 상한 · 비용 관측**
데이터 유출	기밀 데이터가 외부 모델 API 로 나감	**데이터 등급 (N²SF) · 온프레미스 모델 · AI-DLP · 로그**

오른쪽 초록 칸 = 대응

HowTo - Container, 2026

기본 단위는 여전히 컨테이너 — 새로 들어온 구성요소도 같은 방식으로 다룹니다

2015

2015 년 강의

- build ~ monitor 모든 단계의 기본 단위
- 백엔드 서비스 (DB · 메일) 또는 REST API 앱 · 마이크로서비스
- 작업 큐 · 세션 캐시 (redis · memcached)

2026

2026 년에도 그대로 + 새 단위

- 위 세 가지는 모두 그대로
- + 모델 서버 · LLM 게이트웨이
- + 벡터 DB · MCP 서버 · 에이전트 워커 · 샌드박스

모델 서버

온프레미스 GPU 에서 오픈 웨이트 모델 서빙 (vLLM 등)

LLM 게이트웨이

모델 라우팅 · 폴백 · 한도 · 비용 기록

벡터 DB

RAG 색인 · 버전별 컬렉션

MCP 서버

사내 시스템을 에이전트 도구로 공개

HowTo - Scale Out / Load Balance, 2026

무상태로 늘리는 원칙은 그대로 , AI 요청의 특성 (느림 · 비쌈 · 흔들림) 을 더해 설계합니다

항목	2015	2026
부하 분산	CoreOS + confd + nginx	**Service · Ingress · Gateway API**
무상태 API 확장	stateless REST API 를 늘림	**오토스케일 (CPU · 요청 수 · 큐 길이 기준)**
DB 확장	분산 NoSQL · DB	**관리형 DB · 읽기 복제 · 벡터 DB 분리**
캐시 · 큐	작업 큐 · 읽기 캐시	**+ 결과 캐시 · 같은 질문 재사용**
AI 요청 (새로)	—	**오래 걸리는 작업은 큐 + 상태 저장 → 재개 가능하게**
모델 호출 (새로)	—	**게이트웨이에서 한도 · 재시도 · 폴백 · 모델별 라우팅**

AI 요청은 1~ 수십 초 · 요청마다 비용 · 같은 입력에도 다른 결과 — 그래서 큐 · 한도 · 재시도 설계가 필수

필수 요소 , 2026

2015 년의 두 가지는 그대로 유효 — 두 가지를 더합니다

1 같은 환경

개발 · CI · 운영이 같은 컨테이너 이미지를 쓰고 , 다른 것은 deploy 설정뿐

2015 ① 의 현행판 : Vagrant → Dev Containers

2 시나리오 기반 자동 테스트

사용자 시나리오 E2E 테스트 + AI 기능은 평가셋으로 반복 시험

2015 ② 의 현행판 : Selenium → Playwright + eval

3 사람의 승인 지점

기대 결과 · PR 병합 · 운영 배포 · 되돌릴 수 없는 조치는 사람이 확인

새로 추가 : 에이전트가 팀원이 되었기 때문

4 관측과 비용

로그 · 메트릭 · 트레이스 + 토큰 · 비용 · 품질을 함께 기록

새로 추가 : AI 기능은 응답만으로 정상인지 모름

2015 → 2026: 그대로인 것과 바뀐 것

원칙을 알면 도구가 바뀌어도 따라갈 수 있습니다

그대로인 것

- 12-Factor 의 원칙 (설정 분리 · 무상태 · Dev/Prod 일치 · 로그는 스트림)
- 컨테이너가 배포의 기본 단위
- plan → monitor 의 끊임없는 자동화 순환
- 사용자 시나리오 기반 자동 테스트
- 같은 환경에서 개발하고 운영한다

바뀐 것

- 누가 코드를 쓰나 → 사람이 설계 · 검증 , 에이전트가 구현
- 무엇을 시험하나 → 테스트 + AI 평가 (eval)
- 무엇을 관측하나 → + 토큰 · 비용 · 품질
- 무엇을 지키나 → + 프롬프트 주입 · 공급망 · 데이터 등급
- 무엇이 표준인가 → OCI · Kubernetes · MCP · AGENTS.md

자동화의 대상이 ‘빌드 · 배포’ 에서 ‘개발 그 자체’ 로 넓어졌을 뿐 , DevOps 의 정신은 그대로입니다

Q & A

감사합니다

원본 : SaaS / DevOps / Container (2015. 8. 24, 채문창) · 현행화 : 2026. 11