The Twelve-Factor App · 2026 개정판

SaaS 개발 규범 2026

컨테이너 · 쿠버네티스 · AI 에이전트 · 바이브 코딩 시대의 Twelve-Factor

원작 2015.08.21 → 개정 2026.11

채문창 · AI 애플리케이션 개발 과정

2015 → 2026

11년 동안 바뀐 것과 바뀌지 않은 것

구분	2015 (원작 발표 당시)	2026 (현재)
실행 단위	VM, Heroku dyno	컨테이너 · 쿠버네티스 Pod · 서버리스
플랫폼	Heroku · Cloud Foundry 같은 PaaS	쿠버네티스 위의 사내 플랫폼 (플랫폼 엔지니어링)
배포	git push 배포, 스크립트	CI/CD + GitOps, 불변 이미지
운영 관측	로그 스트림	로그 · 메트릭 · 트레이스 (OpenTelemetry)
백엔드 서비스	DB · 큐 · 캐시 · 메일	+ LLM API · 벡터 DB · MCP 서버
코드 작성자	사람 개발자	사람 + AI 코딩 에이전트 (바이브 코딩)
원작 문서	Heroku가 관리 (12factor.net)	2024.11 오픈소스 전환, 커뮤니티가 개정 중

바뀌지 않은 것: **12가지 원칙 그 자체** — 오히려 실행 주체가 늘어난 지금 더 중요해졌다.

클라우드 서비스 모델 2026

IaaS · PaaS · SaaS 사이에 새 층이 생겼다

모델	내가 관리	2026년 대표 예
On-Premise	전부	사내 서버 · 폐쇄망 GPU 서버
IaaS	OS부터 위	EC2 · Compute Engine · GPU 인스턴스
CaaS · K8s (신규)	컨테이너 이미지와 설정	EKS · GKE · AKS · 사내 쿠버네티스
PaaS	앱과 데이터	Heroku · Cloud Run · Vercel · Fly.io
FaaS (확산)	함수 하나	Lambda · Cloud Functions · Workers
SaaS	사용만	Slack · Notion · Salesforce
AI 모델 API (신규)	프롬프트 · 데이터 · 도구	LLM API · 임베딩 · 관리형 에이전트 플랫폼

왜 지금 다시 Twelve-Factor인가

AI가 코드를 빨리 쓸수록, 규범이 더 중요해진다

원작의 목적 (2015)

새 개발자가 프로젝트에 합류하는 시간과 비용을 최소화

2026년의 "새 개발자"

매 세션 처음 합류하는 AI 코딩 에이전트 — 맥락 없이 바로 코드를 쓴다

그래서

12요소는 AI에게 주는 **설계 규칙 · 리뷰 체크리스트**가 된다

선언적 설정, 명확한 종속성, 환경 간 이식성, 개발 · 운영 일치 — 원작의 목표는 **사람과 AI 모두**에게 그대로 적용된다.

Factors 한눈에 보기 ①

I – VI: 원칙은 그대로, 도구는 새로

요소	원칙 (2015 그대로)	2026 실천
I. 코드베이스	버전관리되는 하나의 코드베이스, 다양한 배포	Git + PR, GitOps, AI는 브랜치 · worktree에서 작업
II. 종속성	명시적으로 선언되고 분리된 종속성	uv · pnpm 잠금 파일, 컨테이너 이미지, SBOM
III. 설정	환경에 저장된 설정	환경 변수 + 시크릿 매니저, 모델명 · 프롬프트 버전도 설정
IV. 백엔드 서비스	연결된 리소스로 취급	LLM API · 벡터 DB · MCP 서버도 URL로 교체 가능하게
V. 빌드 · 릴리즈 · 실행	철저하게 분리된 단계	CI가 불변 이미지 빌드, 태그 = 커밋 해시, GitOps 배포
VI. 프로세스	Stateless 프로세스	Pod는 언제든 사라진다, 에이전트 상태도 외부 저장소로

Factors 한눈에 보기 ②

VII – XII: 운영의 원칙이 플랫폼 기능이 되다

요소	원칙 (2015 그대로)	2026 실천
VII. 포트 바인딩	포트 바인딩으로 서비스 공개	컨테이너 포트 + Service · Ingress · 게이트웨이
VIII. 동시성	프로세스 모델을 사용한 확장	HPA · KEDA 자동 확장, 큐 길이 기준 워커 증감
IX. 폐기 가능	빠른 시작과 Graceful Shutdown	헬스 프로브, SIGTERM 처리, 긴 LLM 작업은 재개 가능하게
X. Dev/Prod 일치	개발 · 스테이징 · 운영을 비슷하게	Compose · devcontainer, PR별 미리보기 환경, IaC
XI. 로그 → 텔레메트리	로그를 이벤트 스트림으로	stdout + OpenTelemetry로 로그 · 메트릭 · 트레이스, LLM 토큰 · 비용
XII. Admin 프로세스	일회성 프로세스로 실행	K8s Job · compose run, 마이그레이션 · 재색인 · 평가 실행

Factor I · II

코드베이스와 종속성

I. 코드베이스

원칙: 코드베이스와 앱은 1:1, 공유 코드는 라이브러리로

- Git(GitHub · GitLab · Gitea) + PR + 보호된 main
- 모노레포도 가능 — 단, 앱마다 배포 단위는 하나
- 배포 상태도 Git에: GitOps(Argo CD · Flux)

AI · 바이브 코딩

에이전트는 별도 브랜치 · worktree에서 작업하고 PR로만 합친다. AGENTS.md도 코드베이스에 함께

II. 종속성

원칙: 명시적 선언, 암묵적 시스템 도구 의존 금지

- Python: pyproject.toml + uv.lock (pip → uv)
- Node: package.json + 잠금 파일 (npm · pnpm)
- OS 종속성은 컨테이너 이미지로 고정, SBOM 생성

AI · 바이브 코딩

AI가 존재하지 않는 패키지 이름을 지어낼 수 있다 — 설치 전 실제 패키지 · 버전인지 확인

Factor III · IV

설정과 백엔드 서비스

III. 설정

원칙: 배포마다 달라지는 모든 것은 환경 변수로

- 로컬은 .env(커밋 금지), 키 목록만 .env.example로
- 운영 비밀값은 시크릿 매니저 · K8s Secret · Vault
- 기능 플래그로 코드 배포와 기능 공개를 분리

AI · 바이브 코딩

LLM_MODEL · LLM_BASE_URL · 프롬프트 버전도 설정이다. API 키 하드코딩은 AI가 가장 자주 하는 실수

IV. 백엔드 서비스

원칙: 로컬 · 외부 구분 없이 URL로 연결된 리소스

- PostgreSQL · Redis(Valkey) · Kafka · NATS · S3
- 관리형 서비스로 바꿔도 코드는 그대로, 설정만 교체
- 서비스 연결 정보는 플랫폼이 주입

AI · 바이브 코딩

LLM API · 임베딩 · 벡터 DB · MCP 서버도 백엔드 서비스. 게이트웨이(LiteLLM 등)로 모델을 URL 하나로 교체

Factor V · VI

빌드 · 릴리즈 · 실행과 프로세스

V. 빌드 · 릴리즈 · 실행

CI 빌드 → 이미지:SHA → + 설정 → **실행**

- 빌드: CI가 불변 컨테이너 이미지 생성
- 릴리즈: 이미지 + Helm · Kustomize 값 = 고유 ID
- 롤백 = 이전 태그로 되돌리기

AI · 바이브 코딩

프롬프트 · 모델 버전도 릴리즈에 묶는다 — 답변 품질이 바뀌면 어느 릴리즈인지 추적 가능

VI. 프로세스

원칙: Stateless, 유지할 자료는 백엔드 서비스에

- Pod는 언제든 재시작 · 이동된다 — 로컬 디스크 신뢰 금지
- 세션은 Redis, 파일은 오브젝트 스토리지
- Sticky Session 금지 원칙은 그대로

AI · 바이브 코딩

대화 이력 · 에이전트 상태를 메모리에 두지 말 것. 체크포인터(DB)에 저장해 어느 인스턴스에서든 이어서 실행

Factor VII · VIII

포트 바인딩과 동시성

VII. 포트 바인딩

원칙: 앱이 스스로 HTTP 서버를 내장하고 포트로 공개

- uvicorn · Go net/http가 PORT 환경 변수로 바인딩
- 외부 공개는 Service · Ingress · Gateway API가 담당
- 원작의 host:80 → 5000 예시 = 지금의 리버스 프록시

AI · 바이브 코딩

MCP 서버도 같은 원칙 — 원격 MCP는 HTTP 포트로 공개되는 하나의 백엔드 서비스

VIII. 동시성

원칙: 프로세스 타입별로 개수를 늘려 Scale-Out

- web · worker 타입을 따로 배포하고 따로 확장
- HPA(CPU) · KEDA(큐 길이)로 자동 증감
- 프로세스 안은 async · 고루틴 · 가상 스레드

AI · 바이브 코딩

LLM 호출은 느리고 비싸다 — 웹 요청에서 기다리지 말고 큐 + 워커로, 동시 호출 수는 rate limit에 맞춰

Factor IX · X

폐기 가능성과 Dev/Prod 일치

IX. 폐기 가능 (Disposability)

원칙: 수 초 안에 시작, SIGTERM에 Graceful Shutdown

- readiness · liveness 프로브, preStop 훅
- 워커는 미완료 작업을 큐로 반환 — 작업은 멱등하게
- 스팟 인스턴스 · 노드 교체에도 견디는 설계

AI · 바이브 코딩

수 분짜리 에이전트 작업은 단계마다 체크포인트 — 죽어도 처음부터가 아니라 마지막 단계부터 재개

X. Dev/Prod 일치

원칙: 시간 · 담당자 · 도구의 차이를 최소화

- 로컬도 운영과 같은 이미지 — Docker Compose · devcontainer
- PR마다 미리보기 환경, 인프라는 Terraform · OpenTofu로 코드화
- 시간 차이: 하루에도 여러 번 배포

AI · 바이브 코딩

AI 에이전트도 같은 컨테이너 · 샌드박스에서 실행하고 테스트하게 — “내 PC에선 됐어요”를 AI도 말한다

Factor XI · XII

로그에서 텔레메트리로, 그리고 Admin 프로세스

XI. 로그 → 텔레메트리

원칙: 앱은 stdout으로만, 수집 · 저장은 실행 환경이

- stdout 원칙 유지 + 구조화(JSON) 로그
- OpenTelemetry로 로그 · 메트릭 · 트레이스를 한 표준으로
- Prometheus · Loki · Tempo → Grafana

AI · 바이브 코딩

LLM 호출마다 모델 · 토큰 · 비용 · 지연 · 도구 호출을 트레이스로 (Langfuse 등)

XII. Admin 프로세스

원칙: 일회성 작업도 같은 코드 · 같은 설정 · 같은 릴리즈로

- DB 마이그레이션(Alembic 등)은 K8s Job · compose run
- 운영 서버에 SSH로 들어가 손으로 고치지 않는다
- 작업 스크립트도 코드베이스에, 리뷰를 거쳐

AI · 바이브 코딩

임베딩 재색인 · 평가(eval) 실행 · 백필도 일회성 프로세스. AI에게 운영 DB 직접 쓰기 권한은 주지 않는다

12요소를 넘어

2026년에 추가로 지켜야 할 요소들

API 우선

OpenAPI 계약을 먼저 정하고 구현. 사람 · 앱 · AI 에이전트가 같은 계약을 읽는다

텔레메트리

로그를 넘어 메트릭 · 트레이스 · 헬스 체크를 설계 단계부터

인증 · 인가 · 신원

서비스 간 신원(workload identity), 최소 권한. 에이전트에도 사람과 같은 수준의 권한 관리

공급망 보안

SBOM · 이미지 서명 · 취약점 스캔. AI가 추가한 종속성도 같은 검사를 통과해야

API 우선 · 텔레메트리 · 인증 3가지는 Kevin Hoffman의 『Beyond the Twelve-Factor App』(2016)이 제안한 추가 요소로 널리 쓰인다.

AI 에이전트를 위한 Twelve-Factor

12-Factor Agents: 같은 정신, 새 대상

01 자연어를 도구 호출로 바꾼다	02 프롬프트를 직접 소유한다	03 컨텍스트 윈도우를 직접 설계한다	04 도구는 구조화된 출력일 뿐
05 실행 상태와 비즈니스 상태를 통합	06 단순한 API로 시작 · 중지 · 재개	07 사람에게 연락도 도구 호출로	08 제어 흐름을 직접 소유한다
09 오류를 컨텍스트에 압축해 넣는다	10 작고 초점이 분명한 에이전트	11 어디서든 호출, 사용자가 있는 곳에서	**12** **에이전트를 무상태 리듀서로**

출처: HumanLayer, “12-Factor Agents” (GitHub 공개). 한국어 제목은 번역.

App ↔ Agent

원작의 원칙이 에이전트에서 다시 나타난다

Twelve-Factor App (2011~)	12-Factor Agents	공통된 생각
III. 설정을 코드와 분리	02 프롬프트를 직접 소유	동작을 바꾸는 것은 버전 관리되는 명시적 자산
IV. 백엔드 서비스	04 도구는 구조화된 출력	외부 기능은 계약으로 연결되는 리소스
VI. 무상태 프로세스	05 상태 통합 · 12 무상태 리듀서	상태는 밖에, 실행은 교체 가능하게
IX. 폐기 가능	06 시작 · 중지 · 재개	언제 멈춰도 이어서 실행할 수 있게
VIII. 프로세스 타입별 확장	10 작고 초점이 분명한 에이전트	한 가지 일을 잘하는 단위로 쪼갠다
XI. 로그를 이벤트 스트림으로	09 오류를 컨텍스트에 압축	실행 기록은 다음 판단의 재료
VII. 포트 바인딩	11 어디서든 호출	표준 인터페이스로 열어 두면 누구나 부를 수 있다

바이브 코딩에 적용하기

12요소를 AI 규칙 파일에 넣는다

```
# AGENTS.md — Twelve-Factor 규칙
## 설정 (III)
- 비밀값 · URL · 모델명은 환경 변수로만 읽는다
- 새 변수는 .env.example에 키만 추가한다
## 종속성 (II)
- uv add로만 추가, 잠금 파일을 함께 커밋한다
## 프로세스 (VI · IX)
- 메모리 · 로컬 파일에 상태를 두지 않는다
- SIGTERM을 받으면 작업을 마무리하고 종료한다
## 로그 (XI)
- print 대신 JSON 로그를 stdout으로
## 검증 (X)
- docker compose up 후 테스트 통과가 완료 조건
```

원칙을 문장으로

"12요소를 지켜"보다 구체적 금지 · 허용 문장이 AI에게 효과적이다

번호를 붙여 둔다

리뷰에서 "III 위반"처럼 짧게 지적하고 AI에게 고치게 할 수 있다

기계가 검사하게

비밀값 스캔 · 린트 · 컨테이너 빌드를 CI에서 자동으로

AI가 만든 코드 리뷰

Twelve-Factor 리뷰 체크리스트

I PR로 들어왔고, 다른 앱 코드를 복사하지 않았나?

II 새 패키지가 실제로 존재하고 잠금 파일에 있나?

III 키 · URL · 모델명이 코드에 박혀 있지 않나?

IV 외부 서비스를 설정만 바꿔 교체할 수 있나?

V 실행 중에 코드를 고치는 경로가 없나?

VI 메모리 · 로컬 디스크에 상태를 두지 않았나?

VII 포트를 환경 변수로 받나?

VIII 느린 LLM 작업을 웹 요청 안에서 기다리지 않나?

IX 종료 신호를 처리하고, 작업이 재실행에 안전한가?

X 운영과 같은 컨테이너에서 테스트했나?

XI 로그를 파일이 아닌 stdout으로 내보내나?

XII 일회성 작업이 같은 코드 · 설정으로 실행되나?

주황색 두 항목(III · VI)은 AI 생성 코드에서 흔히 어긋나는 지점이다.

정리

원칙은 그대로,
지키는 도구와 지켜야 할 대상이 늘었다

그대로

12가지 원칙 — 2015년 슬라이드 그대로 유효

새 도구

컨테이너 · K8s · GitOps · OpenTelemetry가 원칙을 자동화

새 대상

AI가 쓰는 코드, AI 에이전트 자체에도 같은 원칙을

참고: 12factor.net · 12factor.net/blog · github.com/humanlayer/12-factor-agents · Q&A