에이전트 플랫폼 운영과 조직 AX 전환

Linux 인프라에서 메모리 · 디스패치 · 개발 루프 · 문서 전파까지

김정한 (JUNGHAN KIM)

포트폴리오 — 공고의 언어에서 실제 기록으로

1. 에이전트 플랫폼 운영 — 5개월, 20회 이상의 버전 사이클

맥락. 에이전트 런타임(OpenClaw)을 2026년 2월부터 6월까지 운영했습니다. Oracle Cloud ARM 인스턴스 위의 Docker, 메신저 채널 다중 연동, 실사용자 트래픽이 걸린 환경입니다.

핵심 조건이 하나 있습니다. **업스트림은 1인이 유지하는 프로젝트이고, 우리 조직에는 업스트림 담당자가 없습니다.** 그래서 "버전을 올린다"가 곧 "릴리즈의 의미를 내 환경으로 번역한다"가 됩니다. 이 조건이 아래 두 인시던트의 배경입니다.

인시던트 ① — 회귀를 계측하고, 정확한 판으로 되돌린다. 두 버전을 한 번에 건너뛴 뒤 응답 지연이 급증했습니다.

증상 응답 latency 급증, `stuck session: state=processing age=164s`

계측 게이트웨이 프로세스가 단일 스레드에서 CPU 102%로 회전. 자식 프로세스는 생성되지 않음. 부팅 88초 — 정상은 11초.

가설 배제 클린 부팅에서도 재현. 운영자 설정 오류가 아님을 입증.

근본 원인 가설 콜드 영속화된 플러그인 레지스트리와 자동 수정 도구의 충돌. 레지스트리 재빌드가 활성 플러그인을 축소시키고, 첫 요청이 핫패스 한복판에서 의존성 설치를 트리거.

롤백 대상 한 단계 아래가 아니라 두 단계 아래의 마지막 정상 판. 이미지 생성 경로가 그 판에서만 기존 인증으로 라우팅되기 때문.

검증 준비 11.3초, 유휴 CPU 0.07%, stuck-session 진단 0건.

비용 운영자 주의력 5시간.

여기서 실제로 값이 나가는 판단은 롤백 대상 선택입니다. "한 단계 내린다"가 아니라 이미지 생성이 별도 *API* 키를 요구하지 않는 마지막 판이 어디인가 를 물었습니다. 최신을 고집하지 않는 것과 아무 데나 되돌리는 것은 다릅니다.

그리고 고친 것은 코드가 아니라 규칙입니다.

- 두 버전 건너뛰기 금지, wait-and-watch.
- 비프로덕션 에이전트에 **24시간 이상 스테이지한 뒤 승격.**
- `stuck session` 한 줄이면 그것만으로 배포 중단 사유.
- **응답성이 SLO다.** 체감 지연은 P0.

인시던트 ② — 업스트림 릴리즈 노트에서 내 증상의 정공법 수정을 식별한다. 다음 판으로 재시도했더니 **10분 만에 같은 인시던트가 재현됐습니다.** 여기서 릴리즈 노트를 읽는 방식이 달라집니다. "무엇이 새로 생겼나"가 아니라 내 증상의 원인을 정면으로 고친 줄이 있는가 를 찾았습니다.

두 줄을 식별했습니다. 광범위한 런타임 프리로드를 실제 설정된 플러그인 id로 한정한 것, 그리고 플러그인 도구 서술자를 캐시한 것. **정확히 인시던트 ①의 근본 원인 가설이 가리키던 지점입니다.** 중간 판들을 건너뛰고 그 판으로 직행했습니다.

Author's Contact Information: 김정한 (Junghan Kim).

지표	before	after
부팅	45.4초	7.3초 → 하드닝 후 5.8초
메모리	816 MiB	246 MiB
핫패스 의존성 스테이징	발생	0

동반 작업으로 운영에서 물러난 컴포넌트를 정리하고, 정체·유령 세션을 72개에서 16개로 줄이고, 컴파일 캐시와 재기동 억제 옵션으로 하드닝했습니다.

인시던트 ③ — 엄격해진 검증이 죽은 설정을 드러낼 때. 이후 판이 플러그인 탐색을 엄격하게 바꾸면서, 오래전에 사라진 컴포넌트를 가리키던 죽은 설정 경로가 hard-fail로 승격되어 크래시 루프가 났습니다. 설정을 통째로 되돌리지 않고 **그 줄만 외과적으로 제거해 복구했습니다.** 결과는 설정 경고 0건.

판단의 규율 — *AX* 도입 실패의 전형에 대한 반례. 진단 도구가 낸 보안 권고를 우리 배포 맥락에서 검토해 **거짓 양성으로 판정하고 적용하지 않았습니다.** 설정을 재작성하는 자동 수정 옵션도 쓰지 않았습니다. 그 도구는 인시던트 ①의 근본 원인 가설에 등장하는 바로 그 도구입니다.

이유를 모른 채 무조건 따르면 위험하다 는 판단을 기록으로 남겼습니다. **AX 도입이 실패하는 가장 흔한 방식이 도구의 권고를 검증 없이 따르는 것이고, 저는 그 반례를 운영 기록으로 갖고 있습니다.**

계보 — 이 서류의 세로축. 운영은 한 제품 설정 안에 머물지 않고 세 계층으로 분리되었습니다.

```
OpenClaw 운영 (업스트림 추적 · 버전 · 인시던트 · 메모리)
  |
  ├─→ andenken       메모리를 런타임에서 떼어내 독립 계층으로
  ├─→ entwurf        하네스 간 디스패치와 연속성
  └─→ forge-config   검토 가능한 개발 루프 · sweeper
```

"에이전트 하네스를 쓴 사람"이 아니라 "에이전트 플랫폼을 운영하고, 그 위에 자기 계층을 쌓아 올린 사람"입니다.

2. 메모리 계층 — andenken

andenken은 과거 세션과 공개 지식베이스를 **같은 검색면에서 다루는 시맨틱 메모리입니다.** 런타임 내부 기능으로 채팅 기록을 가두지 않고, 다른 하네스가 소비할 수 있는 독립 계층으로 꺼냈습니다.

설계의 핵심은 메모리를 세 축으로 가른 것입니다.

축	하는 일	상태
active recall	답변 전 차단형 리콜, 타임아웃 경계	하네스 쪽 (이 리포 밖)
embedding	벡터와 BM25 하이브리드, 점수 정규화	구현·운영 중
dream	야간 통합, 기억의 증류	미구현 (별도 로드맵)

구현 스펙은 LanceDB, Qwen3-Embedding-8B(4096차원), 세션과 공개 가든 두 트랙, 한↔영 교차 질의 확장(한국어 형태소 분석과 영어 태그 매핑을 별도 CLI로 분리), 그리고 회수 추적입니다. 무엇이 실제로 다시 불려 나왔는가는 기억 통합의 입력이 됩니다.

RAG를 "청킹·임베딩·벡터DB"로 묶어 말하지 않는 이유가 여기 있습니다. 검색은 세 축 중 하나이고, 언제 리콜을 차단형으로 걸 것인가와 무엇을 남기고 무엇을 증류할 것인가가 나머지 둘입니다.

3. 디스패치와 연속성 — entwurf

entwurf는 Claude Code, Codex 같은 서로 다른 하네스가 **상대의 인증·대화록·런타임을 소유하지 않고,** 정체성만으로 서로를 호출하게 하는 얇은 기층입니다.

여기에는 자기 리포로 증명되지 않는 것이 하나 있습니다 — **수용(reception).** 자기 저장소는 능력을 보여주지만, 남이 그것을 실제로 썼는지는 보여주지 못합니다.

- 외부 개발자가 Snowflake Cortex Code ACP 백엔드를 기여했습니다 — entwurf #40. **제가 그은 확장 경계를 제3자가 실제로 사용한 기록입니다.**
- 제 코드가 남의 프로젝트에 받아들여진 기록도 함께 둡니다 — ghostel #343, #510.

이 두 줄은 **가장 위조하기 어렵고, 회의적인 독자가 가장 먼저 확인하는 줄입니다.** 그래서 서류에서 지우지 않습니다.

4. 검토 가능한 개발 루프 — forge-config

forge-config는 도메인 오너와의 대화에서 발견된 요구를 셀프호스팅 Forgejo 이슈로 굳히고, 봇이 그것을 해당 도메인 에이전트에게 읽기 전용 1차 리뷰로 넘기고, 결과를 durable store에 남기는 루프입니다.

```
도메인 오너가 도메인 봇과 대화
  → 요구·버그·반복되는 고통이 감지됨 (봇 또는 sweeper)
  → 라벨 + 소스 컨텍스트를 달아 이슈 생성
  → forgebot이 라벨/웹훅으로 깨어남
  → 해당 owner agent에게 read-only 1차 리뷰 요청
  → reality check · risk · scope · 구현필요여부 · 우선순위 반환
  → forgebot이 리뷰를 기록하고 triage 종료
  → 사람이 정렬된 백로그를 보고 focused batch로 구현을 부른다
```

값어치는 "하지 않기로 한 것"에 있습니다. 문서에 명시적으로 박아둔 non-goal이 셋입니다. **자동 코딩 공장이 아니고,** 운영팀용 대시보드 제품이 아니고, 무엇을 구현할지 사람이 정하는 일을 대체하지 않습니다.

- `forgebot` 은 **dispatcher이자 recorder이지 implementer가 아닙니다.**
- "완료" 라벨은 1차 리뷰 triage 완료이지 **구현 완료가 아닙니다.**
- 자동화의 초기 타깃은 **capture, first review, sorting** 입니다.

AX 도입이 실패하는 두 번째 전형이 "에이전트에게 구현을 통째로 맡기는 것"입니다. 호스팅 코딩 에이전트를 타깃으로 삼지 않은 이유를 저는 문서로 설명할 수 있습니다.

상태 권위 — 에이전트의 기억보다 *durable store*가 이깁니다. sweeper와 auto-fix 레인에서는 보수적 기본값, 변경 전 리뷰, durable report, 마커가 실린 코멘트, 스냅샷 드리프트 가드, 결정적 변경 게이트를 채택했습니다. 템플릿 마커가 스키마, 리포트 id, 세션 키, 이슈 갱신시각, 라이프사이클 라벨, 모델, 설정 커밋을 기록합니다.

목적은 하나입니다 — **세션 메모리와 웹훅 재생(replay)이 현재 Forge 상태를 절대 이기지 못하게.** 에이전트가 "내가 아까 처리했다"고 기억해도, 권위는 durable store에 있습니다.

실증 착지: auto-fix v0가 두 리포에서 GREEN입니다. 라벨 감지에서 리포트 생성, 완료 전이까지 돌았고, **재생·멱등성 스모크에서 리포트 중복 생성이 없었습니다.** v1에서는 경계가 제한된 워크스페이스 가드 패치를 실제로 수행하고, 매칭 실패를 또 하나의 비치명적 sweep 케이스로 노출하고, 수정 후 회귀 통과까지 확인했습니다.

책임 경계를 문장으로 그을 수 있다는 것. 상류 런타임은 transport, auth, model, gateway, lifecycle을 **"봇이 깨어날 때까지"** 소유하고, forge-config는 lifecycle protocol, auto-fix semantics, sweeper semantics, validation loop, follow-up 규칙을 소유합니다. **이 경계를 그을 수 있다는 것 자체가 플랫폼 운영자의 표식입니다.**

5. 레거시 시스템에 에이전트 접점 붙이기

이미 돌아가는 IoT · 상담 시스템을 새 플랫폼으로 갈아엎지 않았습니다. **자연어 질의를 기존 DDL, 디바이스 로그, 런타임 미러, 상담 근거로 연결하되 읽기 전용 경계를 유지했습니다.**

- 모든 시각을 하나의 시간축(KST)으로 정규화합니다. **시간축이 흔들리면 장애 분석은 추측이 됩니다.**
- 수치는 원 대화와 원 이벤트까지 역추적하게 둡니다. 요약만 남고 근거가 사라지면 운영팀이 그것을 신뢰할 이유가 없습니다.
- 덤프를 따로 쌓지 않습니다. **살아 있는 원본에서 회수합니다.**

이것이 우대사항의 "DB · MQ · 컨테이너 이해"보다 **한 층 위의 답이라고 생각합니다.** 도메인의 주인이 자기 데이터를 자기 언어로 물을 수 있게 만드는 일이고, 비개발 영역 AX 전환의 실제 모양입니다.

경계: 고객사와 사내 도구의 식별 가능한 세부는 공개면에 쓰지 않습니다. 여기서는 역할과 구조만 말하고, 검증 가능한 공개 저장소를 그 옆에 둡니다.

6. Linux에서 제품까지 — 배경 신뢰도

Zigbee/Wi-Fi 펌웨어, Go 서버, Flutter 앱을 연결해 **양산 제품으로 출하했습니다.** 연결형 배포는 AWS IoT를, 폐쇄망은 로컬 백엔드가 같은 프로토콜을 제공하고, 펌웨어는 수정 없이 양쪽을 사용합니다.

이 경험은 이 서류의 중심축이 아닙니다. 다만 AX 계층 아래 의 DB · MQ · 프로토콜 · 디바이스 현실을 이해하고 있다는 배경 신뢰도로 둡니다. **에이전트를 붙일 대상이 무엇으로 만들어졌는지 모르면, 붙이는 방식도 틀립니다.**

검증 가능한 기록

공개 표면

주장	확인할 곳
에이전트 디스패치와 연속성	entwurf
세션 · 지식 메모리	andenken
검토 가능한 개발 루프	forge-config
임베디드와 엣지 AI	homeagent-config
재현 가능한 Linux 기반	nixos-config
공개 지식 · 문서 체계	Digital Garden
라이브 작업 · 통계 표면	Agenda · /api/stats
제3자의 확장	entwurf #40
제3자가 받아들인 코드	ghostel #343 · #510

마지막 두 줄이 **수용(reception)의 증거입니다.** 자기 저장소는 능력을 보여주지만, 저자가 유일한 심판인 닫힌 루프를 의심하는 독자에게는 답이 되지 않습니다.

시간축

사람이 손으로 기록한 생활과 저널, 에이전트가 남긴 stamp, 실제 커밋과 노트를 하나의 KST 축에서 읽습니다. 원본(FULL)은 제목과 참조를 품으므로 **공개하지 않습니다.** 아래는 allowlist로 집계한 공개 판독이고, `timeline/project.py`가 FULL과 스냅샷의 짝(바이트 지문)을 검증한 뒤에만 생성합니다. 같은 입력이면 같은 바이트가 나오고, 날짜는 하나도 손으로 적히지 않았습니다.

에이전트를 수십 개 붙여도 지나간 시간은 만들어낼 수 없습니다. 이 축은 산출물이 아니라 하루를 셉니다. 같은 기간을 어느 깊이까지 읽느냐에 따라, 살았지만 아무 산출물도 남기지 않은 날이 보이거나 사라집니다.

깊이	무엇이 보이나	기록의 성격	커버리지
0	산 대로의 하루 — 수면 · 가족 · 독서 · 본짓	의도	194일 / 194일
1	그날 손으로 적은 것	의도	133일 / 194일
2	에이전트가 남긴 스탬프	잔여물	136일 / 194일
3	커밋과 노트	잔여물	187일 / 194일

렌즈를 바꾸면 며칠이 사라지는가. 창은 2026-01-01 이상 2026-07-14 미만(KST)이고, 그 사이의 모든 날 194일이 분모입니다.

읽는 렌즈	보이는 날	사라지는 날
깊이 2 · 3 — 잔여물만 (에이전트 스탬프 · 커밋 · 노트)	192일	2일
깊이 1 · 2 · 3 — 잔여물 + 손으로 적은 저널	193일	1일
깊이 0 포함 — 산 대로의 하루	194일	0일

잔여물이 통째로 침묵한 날. 커밋도 노트도 에이전트 스탬프도 없는데, 산 시간은 있습니다. 산출물만 읽는 판독기에게 이 날들은 공백입니다.

2026-02-07 가족 611.6분 · 수면 484.6분 · 독서 358.6분 (합 1,455분) — 저널 0건 · 스탬프 0건 · 커밋 · 노트 0건
2026-07-11 수면 504.0분 · 독서 477.2분 · 가족 408.3분 (합 1,390분) — 저널 2건 · 스탬프 0건 · 커밋 · 노트 0건

이 판독의 출처. 인용은 내용 지문 하나로 하지 않습니다 — 같은 내용이라도 창이 다르면 다른 판독이고, 소스가 부분적이면 다른 품질이며, 깊이 0을 만든 도구가 다르면 다른 손이 적은 것입니다.

창 (`as_of`) 2026-07-14T00:00:00+09:00 — 배타적 상한
소스 상태 git ok(9,776) · note partial(5,511) · agenda ok(3,682) · journal ok(1,292) · timelog ok(8,400)
깊이 0 도구 (`src_tree`) lifetract `bbcc7d483f4537b9 5d130e1c356c57d1 31353a71`
이벤트 28,661건 / 엔티티 23,865건

관측 지문 `content_sha256`
`11be45396ace7834 37e1f393d7dc36cd f455332977117322 e3d39da6d8237f7b`
수집기 `code_sha256` — collector 0.7.0
`a847831b2edd8294 6279753e1ec5b3a0 77a50d83a01e2dd5 0308a99ffb6f3ca4`

이 지문이 증명하지 않는 것. 해시는 바이트와 내용이 같은지, 바꿔치기가 없었는지를 확인할 뿐입니다. 이 코드가 실제로 돌았음도, 소스가 진실하고 완전함도 증명하지 않습니다. 깊이 0은 폰에서 손으로 export한 1인칭 기록이라 외부에서 원본을 재구성할 길이 없고, 확인할 수 있는 것은 파생의 일관성입니다. 원본(FULL)은 제목과 참조를 품으므로 공개하지 않습니다 — 이 문서에는 집계만 나옵니다. 시간 블록은 시작 날짜에 귀속되며, 하루를 24시간 파티션으로 나눈 것이 아닙니다.

수치의 원칙

큰 숫자는 장식이 아니라 더 깊은 기록으로 들어가는 입구입니다. 공개 통계는 라이브 endpoint와 함께 제시하고, 프로젝트 수치는 기간 · 포함정책 · 검증 링크를 붙입니다.

경계

주장을 키우는 것보다 **어디까지가 내 것이 아닌지 먼저 말하는 것이 신뢰를 만든다고 봅니다.**

- Java/Spring 직접 경험을 Go/Clojure 경험으로 바꾸어 쓰지 않습니다.
- BMAD를 사용했다고 쓰지 않습니다. 같은 문제층의 독립 구현과 그 차이를 보여줍니다.
- Tailwind 직접 사용 경험은 없습니다. React/Preact/Lit과 디자인 토큰 경험을 제시합니다.

- 파인튜닝 중심이 아닙니다. 임베딩 서빙, 리랭킹, 검색 품질, 프롬프트·체인이 강점축입니다.
- OpenClaw는 상류 소프트웨어입니다. 제 것은 그 위의 운영층·메모리 계층·개발 루프이고, 이 구분을 흐리지 않습니다.
- 회사 코드와 식별 가능한 운영 데이터는 공개하지 않습니다. 공개 저장소 옆에서 구조와 판단만 설명합니다.
- 해시는 입력과 산출물의 동일성을 확인할 뿐, 원천의 진실성이나 완전성을 대신 증명하지 않습니다.