WHITEPAPER

MSAP.ai

OpsKnow Repo — AI 시대 Infra 운영을 위한 도서관

사람과 AI 가 함께 읽고 쓰는 MD 기반 운영 지식 저장소

"운영 지식이 여러 곳에 흩어져 있다" 는 문제는 IT 업계에서 오래전부터 지적되어 온 익숙한 이야기입니다. 그런데 이 문제는 2025년을 기점으로 무게가 크게 달라졌습니다. 운영팀이 LLM(Large Language Model, 대규모 언어 모델) 을 본격적으로 도입하기 시작하면서 같은 장애를 두고 사람과 AI 가 서로 다른 답을 내놓는 상황이 자주 발생하게 되었기 때문입니다.

2026-06-02 · 작성자: SMART CF Team · msap.ai

목차

1장. 왜 지금 "운영 지식 레포지토리" 인가
1.1 운영 지식의 세 가지 사각지대 — 사내 메신저, 티켓, 머릿속
1.2 AI 시대, 운영 지식의 새 요구사항
1.3 기존 솔루션의 빈자리 — 위키, 인시던트 SaaS, RAG 도구가 함께 못 푸는 것
1.4 OpsKnow Repo 의 한 줄 정의와 다섯 약속
2장. OpsKnow Repo 의 7개 구성요소 — 한 디렉터리에 모인 운영 지식의 일곱 책장
2.1 OPS Diary — AI 와의 일일 질의응답을 시간축에 박는 운영 일지
2.2 Field Manual — 시스템·소프트웨어별 매뉴얼과 SOP 의 현장 책장
2.3 작업 보고서 (Work Reports) — 일·주·월·분기·연간 이력의 누적
2.4 장애 예방점검 (Preventive Inspection) — AI 가 주기 수행하는 점검의 적재
2.5 장애·Incident 관리 — 채팅 덤프부터 Postmortem 까지의 라이프사이클
2.6 Local LLM RAG 소스 — 같은 책장을 AI 가 읽게 하는 마지막 고리
2.7 ADR — 아키텍처 결정 기록의 자동화
3장. 설계 원칙 — MD, Git, 사람-AI 공동편집
3.1 왜 MD 가 중요한가
3.2 디렉터리 트리·파일명·frontmatter 표준 (운영 ontology)
3.3 사람-AI 공동 편집 트랜잭션 — PR 게이트와 author/reviewer/confidence
3.4 Diátaxis 차용 — 문서 종류별 템플릿
3.5 유효성 관리 — 만료·재검토·소유자 알림
4장. 표준 스택 — Ollama / vLLM / 임베딩 / 벡터 DB
4.1 LLM 런타임 — Ollama vs vLLM 의 의사결정 기준
4.2 임베딩 모델 — BGE-M3 / nomic-embed-text 권장
4.3 벡터 DB — LanceDB / Chroma / Qdrant
4.4 클라이언트 — Open WebUI vs 자체 CLI · IDE 통합
5장. 인덱싱과 검색 — git pull 한 번으로 운영 지식 책장이 다시 정렬됩니다
5.1 인덱싱 — git pull 트리거 증분 청크화
5.2 검색 — 메타필터와 의미 검색의 결합
5.3 답변 — 인용 형식 강제와 사람 검증 가능성
6장. 보안·감사 — 외부 호출 0건, RBAC, 마스킹 파이프라인
6.1 외부 호출 0건 검증 — ndjson 호출 로그의 의미
6.2 권한 — RBAC + CODEOWNERS
6.3 마스킹 — 인입 단계의 첫 방어선
6.4 규제 정합 — ISMS-P · 금감원 · 의료 규제
7장. 도입 시나리오 — 단일 시스템 6주 PoC + 다중 시스템 분할
7.1 단일 시스템 6주 PoC 로드맵
7.2 다중 시스템 분할 전략
7.3 RAG 인덱서의 multi-repo 통합

8장. 사례 워크스루 — 인시던트 1건 + AI 예방점검 1분기
8.1 인시던트 1건 워크스루 — 알람부터 Postmortem 머지까지
8.2 AI 예방점검 1분기 워크스루
9장. 도입 성공·함정 신호 10개
9.1 도입 성공 신호 5개와 함정 신호 5개
9.2 다음 액션 — 1쪽 RFP 양식 (의사결정권자용)
Appendix A. References
Appendix A-K. 국내 법령·정부 가이드 (한국어 1차 출처)
Appendix B. Glossary

1장. 왜 지금 "운영 지식 레포지토리" 인가

"운영 지식이 여러 곳에 흩어져 있다" 는 문제는 IT 업계에서 오래전부터 지적되어 온 익숙한 이야기입니다. 그런데 이 문제는 2025년을 기점으로 무게가 크게 달라졌습니다. 운영팀이 LLM(Large Language Model, 대규모 언어 모델) 을 본격적으로 도입하기 시작하면서 같은 장애를 두고 사람과 AI 가 서로 다른 답을 내놓는 상황이 자주 발생하게 되었기 때문입니다. 원인은 단순합니다 — 사람과 AI 가 같은 자료를 보지 않기 때문입니다.

구체적인 장면을 그려 보겠습니다. 어제 운영자가 새로 정리한 결제 API 의 최신 장애 대응 절차(런북) 는 사내 위키에 저장되어 있습니다. 오늘 새벽 당직자가 AI 도구에 같은 장애를 질문했을 때 AI 가 참고한 자료는 RAG(Retrieval-Augmented Generation, 검색 보강 생성 — AI 가 답변하기 전에 사내 문서를 먼저 검색해 그 내용을 답변의 근거로 사용하는 방식) 도구가 따로 저장해 둔 옛 자료입니다. 그리고 지난주 인시던트 대응 중에 운영팀장이 결정한 우회 조치와 그 배경 설명은 사내 메신저 채널의 스레드 안에 묻혀 있습니다.

이렇게 같은 사건에 대한 정보가 위키·AI 도구의 내부 자료·사내 메신저 세 곳에 따로 보관되어 있으면, 세 곳의 내용이 어느 순간 서로 어긋나기 시작합니다. 그 결과 같은 장애에 대해 사람은 새 절차를 따르고 AI 는 옛 절차를 안내하며, 어제 이미 결정된 사항이 오늘 회의에서 다시 처음부터 논의되는 일이 반복됩니다. 인시던트 대응의 일관성이 가장 먼저 무너지고, 운영팀의 의사결정 속도와 신뢰가 함께 떨어집니다.

이 장에서는 운영 지식이 흩어지는 세 곳(사내 메신저·티켓·머릿속) 을 차례로 들여다보고, AI 시대에 새로 더해진 요구사항을 정리합니다. 그다음 시장의 세 갈래 솔루션(위키·인시던트 SaaS·RAG 도구) 이 각각 잘하는 것과 함께 풀지 못하는 공백을 짚습니다. 마지막으로 OpsKnow Repo 의 한 줄 정의와 다섯 약속을 정리하여 책 전체의 thesis 를 제시합니다. 진단부터 정의까지 한 호흡에 끌고 가는 것이 1장의 임무입니다.

먼저 한 가지를 짚고 시작합니다. "운영 지식의 SSoT(Single Source of Truth, 단일 진실 공급원) 가 어디입니까?" 라는 질문에 한 곳의 디렉터리 경로 또는 한 개의 위키 페이지를 답할 수 있는 조직은 의외로 드뭅니다. 글로벌 시스템 운영자·데브옵스 엔지니어 커뮤니티의 공통 결론도 "SSoT 는 하나만 골라라. 두 개는 곧 0개" 라는 한 줄로 모입니다 [S50]. 두 곳에 같은 런북이 있다면 둘 다 동시에 업데이트되지 않으며, 결과적으로 어느 쪽도 신뢰받지 못합니다. AI 가 그중 한쪽을 본 답변과 사람이 다른 쪽을 본 런북이 같은 문제에 다른 결론을 낼 때 의사결정은 곧장 표류합니다.

여러분의 팀이 지금 같은 질문을 받았다고 가정해 보시기 바랍니다. 지난 분기 인시던트 5건의 Postmortem 이 한 디렉터리에 모여 있습니까. 어제 새벽 야간 자동 점검 결과는 어떤 파일명으로 남아 있습니까. 핵심 운영자 1명이 이직한다면 후임자가 첫 주에 무엇을 읽어야 합니까. 이 세 질문에 한 곳의 경로를 답할 수 없다면, 운영 지식 레포지토리이라는 개념은 곧 우리 조직의 실재 문제입니다. 1장은 그 진단을 함께 정리하는 자리입니다.

1.1 운영 지식의 세 가지 사각지대 — 사내 메신저, 티켓, 머릿속

운영 지식이 흩어지는 위치는 거의 항상 세 곳입니다. 사내 메신저 같은 채팅 도구, Jira·ServiceNow 같은 티켓 시스템, 그리고 핵심 운영자의 머릿속입니다. 세 곳은 각각 즉시성·집행력·맥락 보존에서 일면 강점이 있으나, 모두 운영 지식의 SSoT 역할은 수행하지 못합니다. 이 절은 세 곳이 각각 왜 운영 지식의 사각지대가 되는지를 하나씩 분리해 살펴봅니다.

1.1.1 사내 메신저가 지식 사각지대가 되는 이유 — 검색·승계·최신성 셋 다 실패합니다

사내 메신저가 운영팀의 일상 대화 도구로 자리 잡으면서 운영 지식의 대부분이 채널 스레드 안에서 생성되고 소비됩니다. 문제는 사내 메신저의 비동기 채팅 모델이 운영 지식을 보관하기 위해 설계되지 않았다는 점입니다. 데브옵스·SRE 엔지니어들이 모이는 글로벌 커뮤니티의 정기 토론에서도 "사내 메신저는 지식이 사라지는 사각지대다. WIP(Work In Progress, 진행 중 작업) 를 영구 문서로 옮기는 의식적 루틴이 필요하다" 라는 결론이 반복됩니다 [S49]. Charity Majors 역시 "Operational excellence comes from writing it down (운영 우수성은 기록에서 나온다)" 라는 문구로 동일한 진단을 압축합니다 [S51].

사내 메신저가 운영 지식 SSoT 로 적합하지 않은 이유는 세 가지 기준에서 동시에 실패하기 때문입니다. 첫 번째는 검색입니다. 사내 메신저의 무료 또는 표준 요금제는 90일 또는 1년 단위의 메시지 보관 한계가 있으며, 풀텍스트 검색이 있어도 채팅 특유의 짧은 문장·이모지·인용 답글 구조 때문에 키워드 검색의 정밀도가 매우 낮습니다. 두 번째는 승계입니다. 인시던트가 끝난 후 그 스레드를 다시 열어 처음부터 읽는 후임자는 거의 없으며, 비공식 농담·중간 결정·번복된 진단이 시간 순서대로 섞여 있어 핵심을 추출하기 어렵습니다. 세 번째는 최신성입니다. 6개월 전 스레드에 적힌 "지금 이 명령어를 쓰면 됩니다" 가 오늘도 유효한지 검증할 메커니즘이 없습니다. 같은 스레드를 다시 본 사람이 옛 정보임을 표시할 방법이 사실상 없기 때문입니다.

세 가지 기준의 동시 실패는 곧 의사결정 표류로 이어집니다. 야간 인시던트 호출을 받은 당직자가 비슷한 사례를 사내 메신저에서 찾으려고 5분 단위로 검색을 반복하는 동안 MTTR(Mean Time To Recovery, 평균 복구 시간) 은 그만큼 늘어납니다. AI 어시스턴트가 같은 사내 메신저 데이터를 본다 해도 결과는 비슷합니다. 채팅의 단편적 문맥 때문에 AI 가 잘못된 결론을 끌어내거나, 6개월 전 폐기된 명령어를 추천하는 환각이 발생합니다. 사내 메신저를 SSoT 로 간주하는 한 어떤 거버넌스 도구를 도입해도 의미가 없습니다.

이 진단을 의사결정 시점에서 점검할 때는 다음과 같이 묻습니다. 지난 분기 인시던트 중 사내 메신저에서만 논의되고 영구 문서로 옮겨지지 않은 비율은 얼마입니까. 사내 메신저 채널 검색으로 6개월 전 런북을 찾는 데 평균 몇 분이 걸립니까. 검색 결과의 첫 화면에 폐기된 정보가 섞여 나오는 비율은 어떻습니까. 이 세 수치 중 하나라도 만족스러운 답이 없다면, 사내 메신저는 운영 지식의 사각지대입니다.

항목	사내 메신저 채팅	영구 MD 문서
검색 정밀도	낮음 (단편 문장·이모지·답글 분산)	높음 (전체 본문·grep·풀텍스트)
보관 기한	90일~1년 제약 가능	Git 이력 보존 시 사실상 무기한
승계 학습 곡선	스레드 시간순 재독 강요	디렉터리 + 인덱스로 직접 진입
최신성 표시	사실상 불가	frontmatter `freshness_until` (유효 만료일) 가능
AI 입력 적합성	단편화로 환각 유발	LLM 토크나이저 손실 0
변경 감사	메시지 편집 이력 빈약	Git diff·blame 즉시 추적

표가 보여주듯이 사내 메신저와 MD 문서는 같은 운영 지식이라도 6개 기준 전부에서 비대칭입니다. 사내 메신저를 입력 채널로 쓰되 결과물을 MD 로 확정하는 흐름이 필요한 이유가 여기 있습니다.

1.1.2 티켓 시스템이 풀지 못하는 것 — "왜 그렇게 했는가" 의 내러티브 부재

Jira·ServiceNow·Linear 같은 티켓 시스템은 운영 작업의 집행 단위로는 잘 작동합니다. 누가 무엇을 언제까지 처리하는지, 어느 단계에서 막혔는지를 상태기계로 추적할 수 있기 때문입니다. 그러나 티켓은 "무엇을 했는

가" 의 기록 도구이지 "왜 그렇게 했는가" 의 내러티브를 보관하는 자리가 아닙니다. 의사결정의 트레이드오프·맥락·반례를 한두 문장의 코멘트로 압축해 두면 6개월 뒤에 그 결정의 근거를 다시 추적할 길이 사라집니다.

Rundeck 같은 런북 자동화 도구도 비슷한 한계를 공유합니다. Rundeck Job 은 YAML 또는 XML 로 정의된 실행 가능 객체이며, 운영자가 명령어를 표준화하고 실행 이력을 남기는 데는 강력합니다 [S13]. 하지만 그 Job 이 "왜 이 순서로 실행하는가", "이 단계를 건너뛰면 어떤 위험이 있는가" 라는 내러티브 지식은 코드 안에 들어가지 않습니다. 운영자가 6개월 뒤 같은 Job 을 수정해야 할 때, 원 작성자의 의도를 다시 발굴하려면 결국 다른 곳에 적힌 매뉴얼이 필요합니다. 자동화 도구와 위키의 분리가 본질적으로 필요한 이유가 여기 있습니다 [S50].

기록 항목	티켓 (Jira·ServiceNow)	MD 매뉴얼 (런북·SOP)
작업 단위·기한	강함 (상태기계·DueDate)	약함 (정적 문서 본질)
담당자·승인자	강함 (Assignee·Reporter)	가능 (frontmatter owner)
무엇을 했는가	강함 (Description·코멘트)	강함 (본문 단계)
왜 그렇게 했는가	약함 (코멘트 누락 빈번)	강함 (배경·트레이드오프 절)
반례·실패 사례	약함 (티켓 외부)	강함 (Lessons Learned 절)
향후 검색·재사용	중간 (티켓 검색 한정)	강함 (grep·RAG 직접)
AI 답변 인용	약함 (티켓 API 제약)	강함 (MD 그대로 청크화)

표의 일곱 축에서 티켓과 MD 매뉴얼은 상호 보완적입니다. 티켓은 실행 단위 추적의 도구로 두고, 런북·SOP·매뉴얼의 내러티브 지식은 MD 로 옮겨 두 도구를 링크로 연결하는 것이 자연스러운 분할입니다. GitLab 의 공개 Runbooks 리포가 보여주듯이 "Runbooks-as-Code" 모델은 실제로 작동합니다 [S53]. 티켓 본문에 런북 MD 의 영구 링크를 박아 두면 같은 운영 결정이 다음 분기에 다시 호출될 때 트레이드오프까지 함께 재독될 수 있습니다.

여러분의 팀에서 이 진단을 점검할 때는 다음과 같이 물으십시오. 최근 1년 티켓 중 본문 첫 줄에 "왜" 가 적혀 있는 비율을 샘플 30건으로 확인해 보십시오. 그 30건 중에서 런북 MD 의 영구 링크가 박혀 있는 비율은 얼마입니까. 두 비율이 모두 30% 미만이라면, 티켓이 운영 지식 SSoT 의 역할까지 떠맡고 있는 셈이며 후임자가 같은 실수를 반복할 위험이 매우 높습니다.

1.1.3 머릿속 의존의 비용 — 휴가·이직 한 번에 운영 지식이 일시에 증발합니다

세 번째 사각지대는 핵심 운영자의 머릿속입니다. 이 사각지대는 가장 보이지 않지만 가장 비용이 큽니다. 글로벌 시스템 운영자·데브옵스 엔지니어들이 공유하는 운영 노하우 토론에서도 "tribal knowledge(부족 지식, 한 사람의 머릿속에만 있는 운영 노하우) 의존도가 운영 성숙도와 반비례한다" 라는 결론이 반복됩니다 [S49] [S50]. 핵심 운영자 1명이 1주일 휴가를 가거나 다음 분기에 이직한다는 한 줄 통지에 운영팀 전체의 대응 능력이 절벽처럼 떨어지는 경험은 거의 모든 IT 조직이 한 번 이상 겪습니다.

머릿속 의존의 비용은 두 가지 시간선에서 측정됩니다. 첫 번째는 단기 비용입니다. 핵심 운영자가 부재할 때 같은 인시던트의 MTTR 은 사실상 2배에서 4배까지 늘어납니다. 그 운영자만 알던 우회 조작·임시 패치·내부 시스템의 비공식 인터페이스가 사라지면 평시에는 3분이면 끝날 작업이 30분 이상으로 늘어나기 때문입니다. 두 번째는 장기 비용입니다. 그 운영자가 이직하면 후임자가 같은 책임 영역을 수행할 수 있을 때까지 짧으면 1개월,

길면 6개월의 학습 곡선이 발생합니다. 그동안 이전 운영자가 만든 결정의 절반은 영영 재현되지 않으며, 후임자는 비슷한 시행착오를 처음부터 다시 겪습니다.

시점	핵심 운영자 부재 영향	일반적 회복 경로
부재 1일차	MTTR 1.5~2배 증가 (가상 시나리오)	대체 운영자가 채팅·티켓 검색 시도
부재 1주차	미해결 인시던트 누적, 신규 알람 누락	다른 팀원이 비슷한 문제 재발 시 시행착오
부재 1개월차	운영 자동화 변경 정체	후임 배치, 핵심 결정 다시 발굴 시작
부재 3개월차	후임자 학습 곡선 초기, 비공식 노하우 50% 손실	런북·매뉴얼 재독 + 비공식 멘토링
부재 6개월차	후임자 일부 책임 인계 완료, 잔존 격차 30%	Postmortem·과거 결정 재구성으로 보완

표의 시점별 영향은 머릿속 의존이 단순한 인력 리스크가 아니라 운영 능력의 지속성 문제임을 보여줍니다. Google SRE Book 의 Ch.6·Ch.10 이 모든 알람에 런북 링크를 1:1 로 연결할 것을 요구하는 이유도 동일합니다 — 머릿속 지식을 글로 옮겨야 운영이 한 사람의 부재에 좌우되지 않는 시스템이 됩니다 [S52].

의사결정 시점에서 이 진단을 점검할 때는 다음과 같이 묻습니다. 우리 팀의 핵심 운영자 1명이 내일부터 1주일 자리를 비운다고 가정할 때, 그동안 발생할 인시던트의 평균 MTTR 이 평소 대비 몇 배가 될 것 같습니까. 그 운영자만 알고 있는 우회 조작·임시 패치·비공식 인터페이스의 수를 5건 이상 즉시 떠올릴 수 있다면, 머릿속에 잠긴 운영 지식의 규모는 여러분이 생각하는 것보다 큽니다. 이 지식을 글로 옮기지 않으면 어떤 거버넌스 도구도 운영 능력의 단절을 막을 수 없습니다.

1.2 AI 시대, 운영 지식의 새 요구사항

운영 지식이 흩어져 있다는 진단 자체는 새것이 아닙니다. 2010년대의 위키 도입 캠페인이 이미 같은 문제를 풀려고 했습니다. 그런데 2025년 이후 운영 지식의 1차 독자가 사람만이 아니라 AI 까지 둘로 늘어나면서, 같은 진단의 비용 구조가 비대칭적으로 커졌습니다. 이 절은 그 비대칭의 정체를 세 항으로 분해합니다 — 독자 구성의 변화, 포맷의 적합도, 그리고 "같은 책장" 명제의 의미입니다.

1.2.1 사람만 읽던 책장에서 사람 + AI 가 함께 읽는 책장으로

운영 지식의 1차 독자가 사람에서 사람과 AI 둘로 확장된 전환은 단순한 도구 추가가 아닙니다. 두 독자가 같은 정보에 접근하지 않으면 의사결정이 두 갈래로 표류하는 새로운 종류의 실패가 생긴다는 의미입니다. 이 전환은 2024년부터 Local LLM 의 성숙이 임계점을 넘으면서 본격화되었습니다. Ollama 의 `ollama run` 한 줄로 GGUF(GPT-Generated Unified Format, GPT 기반 통합 모델 포맷) 모델을 OpenAI 호환 API 로 띄울 수 있게 된 것 [S42], vLLM 의 PagedAttention 으로 GPU 처리량을 극대화한 추론 엔진이 데이터센터급 운영을 가능하게 한 것 [S43], LM Studio 0.3 의 Chat with Documents 기능으로 폴더 단위 RAG 가 개인 워크스테이션에서 표준 기능이 된 것 [S44] — 세 변화가 2024~2025년에 모두 일어나면서 사내 GPU 1~2장으로 14B 급 모델 운용이 현실화되었습니다.

이 변화의 결정적 함의는 "AI 가 사내 운영 문서를 읽는 것" 이 더 이상 일부 선도 조직의 실험이 아니라 표준 운영 패턴이 된다는 점입니다. 외부 LLM API 의 토큰 단가·지연·데이터 주권 문제 때문에 외부 호출이 불가능했던 국내 금융·공공·의료 영역도 사내 GPU 위에서 14B 급 모델을 띄워 운영 질의에 답하는 패턴을 받아들이기 시작했습니다. 운영 지식 SSoT 가 어디에 있는지, 어떤 포맷으로 보관되어 있는지, AI 가 그 책장을 손실 없이 읽을 수 있는지 — 세 질문이 2024년 이전과는 다른 무게로 의사결정의 중심에 들어옵니다.

항목	2024년 이전	2026년 현재
표준 추론 백엔드	OpenAI/Anthropic API 위주	Ollama·vLLM·LM Studio 가 사내 GPU 위에서 표준화 [S42] [S43] [S44]
사내 모델 크기	7B 급 실험, 답변 품질 불안정	14B~32B 급 안정, 운영 질의 9할 처리
RAG 도구 진입 장벽	LlamaIndex·LangChain 직접 코딩	AnythingLLM·Open WebUI·Khoj 가 패키지 [S33] [S39] [S38]
임베딩 모델	OpenAI text-embedding 의존	BGE-M3·nomic-embed-text 사내 배치 표준
폴더 단위 RAG	별도 파이프라인 구축 필요	LM Studio·GPT4All LocalDocs 가 기본 기능 [S44] [S48]
데이터 주권 검증	API 호출 로그 의존	외부 호출 0건이 ndjson 로그로 입증 가능

표가 보여주듯이 2년 사이의 변화는 단순한 성능 향상이 아니라 운영 패턴의 임계점 통과입니다. 여러분의 팀에서 이 변화를 점검할 때는 현재 사내 문서 중 LLM 이 즉시 읽을 수 있는 형식 — `.md` 또는 `.txt` 같은 평문 — 으로 보관된 비율이 얼마인지 자가 진단을 시도해 보시기 바랍니다. 이 비율이 20% 미만이라면, AI 시대의 운영 지식 레포지토리이라는 개념은 우리 팀에게 직접적인 의제입니다.

1.2.2 LLM 이 손실 없이 다루는 포맷의 가치 — MD vs Confluence Storage Format

AI 가 사내 문서를 읽는 시대가 되면 포맷 선택이 곧 AI 활용 한계 선택이 됩니다. LLM 토크나이저는 평문 또는 단순 마크업을 가장 손실 없이 다루며, 마크다운(`.md`) 은 사람과 LLM 양쪽 모두에게 자연스러운 마크업입니다. H1~H6 헤더, 리스트, 코드 블록, 표가 자연스러운 청크 경계를 형성하므로 별도 변환 없이 그대로 임베딩이 가능합니다. Hacker News 의 "Markdown runbooks vs Confluence" 류 정기 스레드에서도 MD 진영의 우세가 반복적으로 확인되며, 그 근거로 diff·grep·LLM 친화성 세 가지가 거론됩니다 [S54].

반대편에 있는 Confluence Storage Format 은 XHTML 기반 자체 마크업입니다 [S17]. 사람 입장에서는 WYSIWYG 편집기로 가려져 있어 보이지 않지만, LLM 이 그 본문을 읽으려면 매크로 태그·확장 속성·내장 자바스크립트 hint 같은 잡음을 함께 받아들여야 합니다. 토큰당 본문 정보 밀도가 낮아질 뿐 아니라, 페이지 간 의미적 참조가 Storage Format 의 내부 ID 로 표현되어 RAG 인덱서가 cross-link 를 재구성하기 어렵습니다. Notion 의 블록 DB 도 같은 문제를 다른 방식으로 갖습니다. 블록 단위 객체가 자체 DB 에 들어 있어 MD export 가 가능하지만 round-trip(왕복) 손실이 발생하므로 SSoT 로 두기 곤란합니다 [S32].

포맷	1KB 본문당 토큰 비율	diff 가독성	grep 직접 가능	LLM 입력 손실	외부 변환 필요
Markdown (.md)	1.0 (기준)	매우 좋음	가능	없음	불필요
Confluence Storage (XHTML)	1.5~2.0 (매크로·속성 잡음)	나쁨 (XML 토큰 노이즈)	부분 (DB export 필요)	매크로·확장 토큰 인입 [S17]	필요 (MD 변환)
Notion 블록 DB	측정 불가 (DB 객체)	불가 (블록 ID 비교)	불가 (API 우회)	블록 메타 손실 [S32]	필요 (MD export 후 round-trip 손실)

표가 보여주듯이 세 포맷의 LLM 적합도는 비대칭적입니다. 같은 런북을 MD 로 저장한 팀과 Confluence Storage Format 으로 저장한 팀이 동일한 Local LLM 에 같은 질문을 던졌을 때, 답변의 인용 정확도·맥락 보존·토큰 비용이 모두 비대칭으로 갈리는 셈입니다. 포맷 선택이 곧 AI 활용 한계 선택이라는 명제의 의미가 여기서 드러납니다.

여러분의 현재 SSoT 가 어떤 포맷에 들어 있는지를 한 번 점검해 보시기 바랍니다. 위키 페이지를 LLM 컨텍스트로 변환할 때 토큰 손실률이 얼마인지, 그 위키의 cross-link 가 LLM 의 RAG 인덱서에서 재구성 가능한지 — 두 질문에 답해 보면 현재 SSoT 의 AI 친화도가 1줄로 정리됩니다. MD 를 1차 객체로 두는 SSoT 와 그렇지 않은 SSoT 의 차이는 도구의 선호가 아니라 LLM 시대의 운영 비용 차이입니다.

1.2.3 "AI 가 본 컨텍스트 = 사람이 본 런북" 일 때만 의사결정이 합치합니다

세 번째 새 요구사항은 가장 본질적입니다. AI 와 사람이 서로 다른 데이터 소스를 보면 운영 의사결정이 두 갈래로 표류한다는 점입니다. 예를 들어 사내 RAG 도구가 위키의 6개월 전 스냅숏을 인덱싱한 상태에서 사람이 어제 갱신된 위키 본문을 보고 의사결정한다고 가정하면, AI 의 답변 근거와 사람이 읽은 런북이 같은 문제에 다른 결론을 낼 수밖에 없습니다. RAG 도구가 본 컨텍스트와 사람이 본 런북이 동일한 디렉터리·동일한 시간선을 공유해야 환각의 표면적이 좁혀지고 책임 추적이 가능해집니다 [S30] [S32].

이 명제는 단순한 가독성 문제가 아니라 거버넌스의 단일점 문제입니다. AI 답변의 근거 파일을 사람이 즉시 열어볼 수 없다면 그 답변은 검증 불가능합니다. 검증 불가능한 답변이 운영 결정에 흘러들어가면 인시던트 대응의 일관성이 사라집니다. 반대로 같은 디렉터리의 같은 MD 파일을 사람과 AI 가 공유한다면, AI 가 인용한 줄 번호를 사람이 즉시 따라가 검증할 수 있고, 사람이 수정한 런북을 AI 가 다음 호출에서 곧장 반영할 수 있습니다. 이것이 OpsKnow Repo 가 "사람이 보는 책장 = AI 가 보는 책장" 을 명제로 박는 이유입니다.

다음과 같은 자가 진단 체크리스트로 점검해 볼 수 있습니다. 최근 AI 답변 1건을 골라 그 답변에 인용된 근거 파일을 사람이 즉시 열어볼 수 있습니까. 그 파일이 현재의 SSoT 디렉터리 안에 있습니까. 그 파일의 최근 변경 이력을 Git blame 으로 확인할 수 있습니까. 세 질문에 모두 "예" 라고 답할 수 있는 조직은 운영 의사결정의 정합도가 이미 높은 편입니다. 반대로 세 질문 중 하나라도 "아니오" 라면, 사람과 AI 가 다른 책장을 보고 있다는 신호이며 의사결정이 두 갈래로 표류할 가능성이 늘 잠재합니다.

표면적으로는 작은 운영 위생 문제처럼 보이지만, 책임 추적·환각 방어선·학습 곡선 압축 세 가지가 동시에 걸려 있는 단일점 결정입니다. 이 단일점을 디렉터리 하나로 봉합하는 것이 OpsKnow Repo 의 핵심 thesis 이며,

1.3 절에서 다룰 기존 솔루션의 빈자리도 결국 이 단일점을 누가 어떻게 채우느냐의 질문입니다.

1.3 기존 솔루션의 빈자리 — 위키, 인시던트 SaaS, RAG 도구가 함께 못 푸는 것

운영 지식 관리의 시장은 세 갈래로 갈라져 있습니다. 위키 카테고리는 사람이 읽기 좋은 영구 문서를 만들지만 AI 와의 결합이 약합니다. 인시던트 SaaS 카테고리는 채팅 자동 요약과 Postmortem 초안 생성을 표준 기능으로 만들었지만 산출물이 자사 SaaS DB 에 갇힙니다. RAG 도구 카테고리는 사내 문서를 읽고 답하지만 능동적으로 쓰지 못합니다. 세 갈래가 각자의 영역에서 잘하는 것을 인정하면서도, 함께 풀지 못하는 공백을 정확히 보는 것이 이 절의 임무입니다.

1.3.1 위키(Confluence · Notion) — 읽기 좋지만 AI 와 단절

Confluence 와 Notion 은 사내 위키의 두 강자입니다. 양쪽 모두 비개발자 친화적 편집기, 권한 모델, 검색, 댓글, 댓글 알림 등 협업 위키의 표준 기능을 제공합니다. 그런데 두 도구를 OpsKnow 관점에서 평가하면 세 축에서 동일한 한계가 드러납니다. 첫째, MD 가 1차 객체가 아닙니다. Confluence 의 Storage Format 은 XHTML 변종이며 [S17], Notion 의 본문은 블록 DB 객체입니다 [S32]. MD export 가 가능하더라도 round-trip 손실이 발생하므로 외부 LLM 이 직접 읽을 1차 자료로는 부적합합니다.

둘째, 외부 LLM 자유 연결이 제약됩니다. Atlassian Intelligence 와 Rovo 는 Atlassian 그래프 안에서만 RAG 가 가능하며, 외부 LLM 으로 자유롭게 컨텍스트를 흘릴 수 없습니다 [S18]. Notion AI 도 자사 그래프 종속으로 동일한 제약을 갖습니다 [S32]. 사내 GPU 위의 Ollama·vLLM 으로 자유롭게 컨텍스트를 흘리려고 시도하면 위키 본문을 한 번 export 한 뒤 round-trip 손실을 감수해야 합니다. 셋째, 로컬·온프레미스 운영이 사실상 불가능합니다. Confluence Data Center 가 유지되고 있지만 Cloud-first 전략이며 [S17], Notion 은 SaaS 전용입니다.

솔루션	MD 표준 단위	외부 LLM 자유 연결	로컬 운영 가능
Confluence	아니오 (Storage Format = XHTML [S17])	제약 (Rovo 종속 [S18])	제한적 (DC 유지·Cloud 주력)
Notion	아니오 (블록 DB [S32])	제약 (Notion AI 자사 그래프)	아니오 (SaaS 전용)

표가 보여주듯이 두 도구는 세 축에서 모두 OpsKnow 의 전제와 어긋납니다. SaaS 위키를 SSoT 로 둔 채 Local LLM RAG 를 붙이려는 시도는 데이터 락인(lock-in) 의 우회 비용을 매번 지불해야 합니다. 위키가 사내 표준이라 바꿀 수 없다면, 위키와 별도로 운영 지식 디렉터리를 구성하는 이중화 비용을 받아들이거나, 위키의 export 파이프라인을 SSoT 로 두는 절충안을 만들어야 합니다. 어느 쪽도 단일 디렉터리 SSoT 의 단순성에 미치지 못합니다.

1.3.2 인시던트 SaaS(PagerDuty · Incident.io · Rootly) — 자동화 좋지만 SSoT 가 SaaS DB 에 갇힘

인시던트 SaaS 카테고리는 2024~2025년에 큰 변화를 겪었습니다. PagerDuty AIOps [S23], Incident.io AI [S24], Rootly AI [S26] — 3강이 모두 "사내 메신저 채팅 → AI 요약 Postmortem 초안" 기능을 표준 기능으로 확정했습니다. FireHydrant 의 AI Retrospectives [S25], Blameless 의 Postmortem 자동 채움 [S28], Datadog Bits AI 의 Postmortem Notebook [S29] 도 같은 패턴을 도입했습니다. 인시던트 자동화

의 본질적 부담 — 채팅 스레드에서 타임라인을 다시 발굴하는 수작업 — 을 AI 가 대신 처리해 준다는 약속이 시장의 표준이 된 셈입니다.

그런데 OpsKnow 관점에서 이 솔루션들의 공통 한계는 산출물의 SSoT 위치입니다. PagerDuty 의 Postmortem 은 PagerDuty 콘솔에 머무르거나 export 후 Confluence·Notion 페이지로 흘러갑니다. Incident.io 의 AI Scribe 결과물은 자사 에디터 또는 Notion·Confluence·Google Docs 로 export 됩니다 [S24]. Rootly AI 의 Postmortem 은 Notion·Confluence·Google Docs·Jira 중 하나로 publish 됩니다 [S26]. 즉 인시던트 자동화는 받지만 산출물이 사내 SSoT 와 분리되며, 다음 인시던트의 RAG 컨텍스트로 자동 재활용되지 않는 구조입니다.

솔루션	산출물 저장소	MD export	Local LLM 결합
PagerDuty	PagerDuty DB → Confluence·Notion publish [S23]	부분	어려움 (SaaS-first)
Incident.io	자사 에디터 → Notion·Confluence·Google Docs [S24]	부분	어려움 (EU/US SaaS)
Rootly	자사 DB → Notion·Confluence·Google Docs·Jira [S26]	부분	어려움 (SaaS lock-in)
FireHydrant	자사 DB → Confluence (일부) [S25]	부분	어려움 (SaaS)

표가 보여주듯이 산출물이 사내 디렉터리 SSoT 로 직접 commit 되는 경로가 없습니다. 인시던트 자동화의 가치는 받되 산출물의 위치는 별도 결정해야 한다는 의미입니다. 같은 인시던트의 Postmortem 이 SaaS DB 에 있고 사내 런북이 위키에 있고 RAG 인덱서가 또 다른 곳을 본다면, "같은 책장" 명제는 깨집니다. 인시던트 1건의 학습이 다음 인시던트의 자산이 되는 학습 순환은 산출물이 사내 SSoT 디렉터리에 commit 될 때에만 자연스럽게 닫힙니다.

1.3.3 RAG 도구(AnythingLLM · Open WebUI · Glean) — 읽기는 잘하지만 쓰지 못함

RAG 도구 카테고리는 다시 세 갈래로 갈라집니다. 상용 SaaS 인 Glean 은 사내 메신저 ·Drive·Jira·Confluence 등 사내 SaaS 50종을 통합 검색해 AI 답변을 제공하지만 데이터 락인이 있으며 로컬 운영이 사실상 불가능합니다 [S30]. OSS RAG 도구인 AnythingLLM 은 문서 업로드 후 워크스페이스 단위 RAG 를 제공하지만 본질적으로 "업로드 → 읽기" 모델이며, 원본 MD 와의 양방향 동기화가 약합니다 [S33]. Open WebUI 의 Knowledge 기능은 Ollama 백엔드 위에 사내 RAG 워크벤치를 제공하지만 지식이 WebUI 의 내부 DB 에 고립되어 외부 디렉터리와의 양방향 동기화가 1급 모델이 아닙니다 [S39].

Obsidian 생태계의 Smart Connections·Copilot 플러그인 [S33] 과 Khoj [S38] 는 로컬 MD 볼트를 1차 자료로 두고 AI 답변·자동 링크·블록 단위 요약을 제공합니다. MD 가 표준 단위라는 점에서 OpsKnow 의 전제와 가장 가깝지만, 본질적으로 개인 볼트용 도구라 팀 권한·동시 편집·인시던트 자동 인입·운영 자동화 같은 거버넌스 레이어가 부재합니다. Onyx(구 Danswer) 같은 OSS 엔터프라이즈 검색 도구도 50종 이상의 커넥터로 사내 데이터를 통합 검색하지만 운영 지식을 능동적으로 작성·확정하는 워크플로우가 약합니다 [S41].

도구	양방향 편집	Git 친화성	온프레미스 가능
Glean	약함 (검색·답변 중심 [S30])	약함	어려움 (SaaS·VPC 옵션 일부)
AnythingLLM	약함 (업로드 → 읽기 [S33])	부분 (원본 보관)	가능 (Docker self-host)
Open WebUI Knowledge	약함 (내부 DB 고립 [S39])	부분	가능 (Docker self-host)
Obsidian Smart Connections	강함 (MD 직접 편집)	강함 (Git 호환)	가능 (로컬 우선)
Onyx (Danswer)	약함 (검색·답변 [S41])	부분	가능 (Docker self-host)

표의 다섯 도구 모두 "읽기" 측면에서는 성숙해 있지만, "쓰기 + 협업 + 인시던트 자동 인입" 의 결합은 어디에도 없습니다. RAG 도구만 도입하면 운영 지식을 능동적으로 작성·확정·재활용하는 워크플로우가 비어 있으며, 사람의 PR 워크플로우와 AI 의 RAG 컨텍스트가 같은 디렉터리를 공유하는 1급 모델이 없습니다. 이 공백이 OpsKnow Repo 가 차지할 자리입니다.

1.4 OpsKnow Repo 의 한 줄 정의와 다섯 약속

1.1 의 세 가지 사각지대 진단, 1.2 의 새 요구사항, 1.3 의 시장 빈자리를 한 자리에 모으면 OpsKnow Repo 의 정의가 자연스럽게 나옵니다. 이 절은 한 줄 정의와 다섯 약속을 정리하여 책 전체의 thesis 를 제시합니다. 다섯 약속은 데이터 주권·사람과 AI 공동편집·인시던트 자동 순환·운영 ontology 표준·vendor lock-in 회피 다섯 가지 기준으로 정렬되며, 이후 장의 모든 논의가 이 약속들의 구체화에 해당합니다.

[FIGURE: solution-matrix] **캡션**: 기존 KB / Wiki / 인시던트 SaaS / RAG 도구 vs OpsKnow Repo — 5기준 × 8솔루션 비교 매트릭스 **의도**: 행은 8개 대표 솔루션 (Confluence · Notion · PagerDuty · Incident.io · AnythingLLM · Open WebUI · Obsidian + Smart Connections · OpsKnow Repo). 열은 5개 기준 (MD 표준 단위 / Local 가능 / AI 공동편집 / 운영 ontology / 인시던트 자동 인입). 셀은 ✅ / ⚠ / ❌ 3단계. OpsKnow Repo 행만 5개 셀 모두 ✅ 임을 시각적으로 부각하여 다섯 약속이 동시에 만족됨을 보입니다.

5 기준 × 8 솔루션 비교 — OpsKnow Repo 만 다섯 칸이 모두 충족

행 = 8 개 솔루션 · 열 = 5 개 기준 · 셀 = 충족 / 부분 / 미충족

솔루션 \ 기준	MD 표준 단위	Local 가능	AI 공동편집	운영 ontology	인시던트 자동 인입
Confluence	부분	부분	미충족	미충족	미충족
Notion	부분	미충족	미충족	미충족	미충족
PagerDuty	미충족	미충족	미충족	미충족	충족
Incident.io	미충족	미충족	미충족	미충족	충족
AnythingLLM	부분	충족	부분	미충족	미충족
Open WebUI Knowledge	부분	충족	부분	미충족	미충족
Obsidian + Smart Conn.	충족	충족	부분	미충족	미충족
OpsKnow Repo	충족	충족	충족	충족	충족

충족 = 정책 + 도구 동시 만족　부분 = 일부만 또는 우회 필요　미충족 = 도구 범위 밖

이 매트릭스가 1장의 결론을 한 장으로 압축합니다. 1.3 절에서 살펴본 8개 솔루션 중 어느 것도 5개 기준 전체에서 ✅ 가 아니며, OpsKnow Repo 만 다섯 셀이 모두 ✅ 입니다. 이것이 1.1 의 세 무덤과 1.2 의 새 요구사항을 동시에 풀려고 했을 때 시장이 비워둔 자리입니다.

1.4.1 한 줄 정의 — MD + Git + Local LLM RAG + 운영 ontology + 인시던트 인입

OpsKnow Repo 의 한 줄 정의는 다음과 같습니다. MD 파일을 1차 객체로 두고, Git 으로 사람·AI 가 같은 디렉터리를 함께 편집하며, 그 디렉터리가 곧 Local LLM 의 RAG 소스가 되고, 운영 ontology(SRE Book · AWS W-A · Diátaxis) 가 frontmatter 표준으로 박혀 있으며, 인시던트 채팅이 MD 로 자동 인입되는 운영 지식 저장소입니다. 다섯 요소가 모두 갖춰져야 의미가 있으며, 네 개만으로는 1.3 의 기존 솔루션과 같은 자리에 머무릅니다 [S19] [S62].

다섯 요소가 동시에 만족되어야 하는 이유는 각 요소가 다른 요소의 전제이기 때문입니다. MD 가 1차 객체가 아니면 Git diff·PR 리뷰가 의미를 잃습니다. Git 워크플로우가 없으면 사람·AI 공동편집의 승인 게이트가 빠집니다. Local LLM RAG 가 없으면 데이터 주권의 검증이 불가능합니다. 운영 ontology 가 없으면 RAG 의 메타 필터가 만들어지지 않습니다. 인시던트 자동 인입이 없으면 채팅 지식이 영구화되지 않습니다. 다섯 요소는 서로의 전제이자 결과인 순환 구조입니다.

요소	의미	부재 시 결과
MD 표준 단위	모든 콘텐츠가 `.md` + `assets/`	LLM 입력 손실·diff 불가
Git 협업	PR + CODEOWNERS + diff 이력	승인 게이트·감사 추적 부재
Local LLM RAG	Ollama·vLLM 위 corpus 인덱싱	외부 호출 불가피·데이터 주권 위협

요소	의미	부재 시 결과
운영 ontology	SRE Book·W-A·Diátaxis frontmatter [S21] [S22] [S62]	RAG 메타 필터 부재·답변 정합도 저하
인시던트 자동 인입	사내 메신저 → MD 정규화 → commit	채팅 지식 휘발·재활용 순환 단절

여러분의 팀이 이 다섯 요소 중 무엇을 갖추고 있고 무엇이 비어 있는지 자가 진단해 보시기 바랍니다. 갖춘 요소와 빠진 요소의 조합이 향후 도입 로드맵의 첫 단계를 결정합니다 — 부록 E 의 6주 체크리스트가 그 자가 진단의 구체 항목을 제공합니다.

1.4.2 약속 1 — 데이터 주권 (외부 API 호출 0건 검증)

첫 번째 약속은 데이터 주권입니다. 외부 LLM API 호출 0건이 마케팅 슬로건이 아니라 ndjson(newline-delimited JSON) 호출 로그로 검증 가능한 객관 지표여야 합니다 [S34] [S42]. Ollama·vLLM·llama.cpp 가 모두 OpenAI 호환 API 를 사내에서 제공하므로, 외부 호출이 0건임을 호스트 단의 outbound 트래픽 로그와 결합해 입증할 수 있습니다. 이 검증 가능성이 국내 금융·공공·의료 규제의 기본 전제와 정합합니다.

외부 호출 0건 ndjson 로그의 한 줄 샘플은 다음과 같이 생깁니다.


```
{"ts":"2026-05-31T03:14:22Z","backend":"ollama","model":"gpt-oss:20b","prompt_tokens":1240,"completion_tokens":312,"egress_external":false,"egress_internal_host":"ollama.svc.cluster.local","caller":"opsknow-rag-indexer"}
```


위 로그 한 줄을 풀어 읽으면 의미가 명확해집니다. `egress_external: false` 는 "이 LLM 호출이 사외 네트워크로 나가지 않았다" 는 사실을 기록한 표기이고, ndjson(한 줄에 한 건의 JSON 을 적재하는 로그 형식) 은 호출이 발생할 때마다 한 줄씩 누적되는 표준 로그 포맷입니다. 그리고 이 로그 파일을 추가만 가능하고 수정·삭제는 불가능한 append-only 형식으로 보존하면, 사후에 누가 로그를 조작하더라도 그 흔적이 남기 때문에 외부 호출 0건이라는 사실이 외부 감사관에게도 입증 가능한 증거가 됩니다. 사내 AI 도구의 외부 호출 0건을 어떻게 증명하느냐는 질문에 위와 같은 로그 한 줄로 답할 수 있는 조직과 그렇지 못한 조직의 차이는 규제 대응의 차이로 직결됩니다. 6장의 보안·감사 절에서 이 메커니즘을 자세히 다루지만, 1.4.2 의 약속은 그 절의 결론을 한 줄로 요약한 셈입니다.

1.4.3 약속 2 — 사람·AI 공동편집 트랜잭션 (PR 게이트 + frontmatter author 명시)

두 번째 약속은 공동편집 트랜잭션입니다. AI 가 작성한 diff 가 PR 로 들어와 사람 reviewer 1인 이상의 승인 후 머지되는 표준 흐름을 코드 수준에서 박는 것이 핵심입니다 [S61] [S19]. AI 생성 콘텐츠는 frontmatter 에 `author: ai` , `confidence: low|medium|high` , `model: <name>` 세 필드가 강제로 박힙니다. 사람이 거부하면 머지되지 않으며, 머지되면 Git 의 commit 이력에 누가 언제 승인했는지가 영구 기록됩니다.

AI 편집 PR 의 frontmatter 가 어떻게 변하는지 한 예시로 살펴봅니다. AI 가 생성한 Postmortem 초안의 frontmatter 가 다음과 같이 작성된다고 가정합니다.

```
---
title: "2026-05-30 결제 API 지연 인시던트"
```

author: ai
model: gpt-oss:20b
confidence: medium
reviewer: human-pending
sources:
  - "Incidents/2026/Q2/INC-0427/raw/chat.md#L120-L240"
  - "Field_Manual/payment-api/throttling.md"

이 PR 이 사람 reviewer 의 승인을 거치면 `reviewer` 필드가 `kim.yj@example.com` 로 갱신되고 `confidence` 가 reviewer 의 판단에 따라 조정됩니다. AI 의 콘텐츠 오염 위험은 머지 게이트가 사람이라는 단순한 원칙으로 1차 방어선이 만들어집니다. AI 산출물의 출처 메타가 frontmatter 에 박혀 있는지를 점검 항목으로 두는 것이 도입 초기의 거버넌스 안전판입니다.

1.4.4 약속 3 — 인시던트 자동 재활용 (채팅 → MD → 차기 RAG)

세 번째 약속은 인시던트 자동 재활용입니다. 사내 메신저 채팅 스레드가 인시던트 종료 시점에 자동으로 MD 로 보관되어 디렉터리 SSoT 에 commit 되며, 같은 MD 가 다음 인시던트의 RAG 컨텍스트로 다시 사용되는 학습 순환입니다 [S24] [S25] [S26]. 채팅 raw 덤프는 감사용으로 `Incidents/<id>/raw/chat.md` 에 보존하고, 정제된 Postmortem 은 별도 파일 `Incidents/<id>/postmortem.md` 로 분리합니다. raw 덤프는 `rag_visible: false` 메타로 RAG 노출을 통제하며, Postmortem 만 RAG corpus 에 합쳐집니다.

이 학습 순환이 인시던트 학습의 비대칭적 가치를 만듭니다. 인시던트 1건의 처리 비용은 한 번 지불되지만, 그 인시던트의 Postmortem 이 다음 인시던트의 RAG 컨텍스트로 재사용되는 가치는 분기를 지나 누적됩니다. 지난 3개월 Postmortem 이 다음 RAG 검색에 잡히는 비율이 도입 ROI 의 첫 번째 측정 지표입니다. 학습 순환이 작동하면 비슷한 인시던트의 MTTR 이 분기 단위로 단축되며, 순환이 깨지면 1.1.3 의 머릿속 의존이 다시 돌아옵니다.

1.4.5 약속 4 — 운영 ontology 표준 (SRE Book · W-A · Diátaxis 흡수)

네 번째 약속은 운영 ontology 의 표준화입니다. Google SRE Book 의 SLO(Service Level Objective, 서비스 수준 목표) · SLI(Service Level Indicator, 서비스 수준 지표) · 오류 예산 [S21], AWS Well-Architected Operational Excellence Pillar 의 OPS 1~11 [S22], Diátaxis 의 4분면(Tutorial · How-to · Reference · Explanation) [S62] — 세 표준 어휘를 YAML frontmatter 스키마로 흡수합니다. 운영 도메인의 어휘가 메타 필드로 박혀야 RAG 의 메타 필터가 동작하며, 답변 정합도가 안정적으로 유지됩니다.

운영 ontology 흡수의 첫 단계는 frontmatter 필드 8종 — `owner` / `reviewer` / `service` / `severity` / `doc_type` / `freshness_until` / `confidence` / `sources` — 의 표준화입니다. 부록 C 가 enum 값을 포함한 전체 스키마를 제공하며, 3장의 설계 원칙 절이 8종 필드의 각 의미를 자세히 다룹니다. 현재 운영 문서 중 SLO·심각도·서비스 태그가 메타로 박힌 비율이 30% 미만이라면, ontology 의 흡수가 도입 초기의 가장 큰 부담이자 가장 큰 효과 발생 지점입니다.

1.4.6 약속 5 — Vendor lock-in 회피 (디렉터리 통째로 휴대)

다섯 번째 약속은 vendor lock-in 회피입니다. 모든 콘텐츠가 `.md` 파일과 `assets/` 디렉터리로 휴대 가능하며, OpsKnow 가 사라져도 자산은 남는다는 단순한 원칙입니다 [S17] [S32]. SaaS 위키와의 본질적 차이가

여기 있습니다. Confluence 가 사라지면 Storage Format 의 매크로·확장 태그가 그대로 들어 있는 export 파일만 남고, Notion 이 사라지면 블록 DB 의 round-trip 손실이 발생한 MD 파일만 남습니다. OpsKnow Repo 가 사라지면 그냥 `.md` 파일과 `assets/` 디렉터리가 그 자리에 있을 뿐입니다.

위키 export round-trip	원본 보존도	이미지·첨부 보존	상호 링크 보존	변환 비용
Confluence → MD export	70~85% (매크로 손실)	부분 (별도 디렉터리)	부분 (페이지 ID 깨짐)	중간
Notion → MD export	60~80% (블록 round-trip)	부분 (assets 디렉터리)	약함 (블록 ID 변환)	중간
OpsKnow Repo (휴대)	100% (변환 없음)	완전 (`assets/`)	완전 (Markdown link 그대로)	0

위 비교표가 보여주는 것은 단순한 도구 차이가 아니라, 도구를 잘못 선택했을 때 자료를 다른 곳으로 옮기는 데 드는 비용의 차이입니다. SaaS 위키에 SSoT 를 묶어 두면 나중에 다른 도구로 옮길 때 매크로·블록 구조가 깨져 재작업 부담이 큽니다. 반면 OpsKnow Repo 의 디렉터리 이동 모델은 `.md` 파일과 `assets/` 폴더를 그대로 복사하기만 하면 되므로 옮길 때 손실되는 자료가 거의 없습니다. 도입 의사결정의 안전판으로서 이 이동 자유도가 가장 중요한 약속 중 하나인 이유입니다. 1장의 진단·요구·공백·정의가 여기서 한 번 마무리되며, 2장부터는 이 다섯 약속이 7개 영역의 디렉터리 구조와 운영 흐름으로 구체화됩니다.

2장. OpsKnow Repo 의 7개 구성요소 — 한 디렉터리에 모인 운영 지식의 일곱 책장

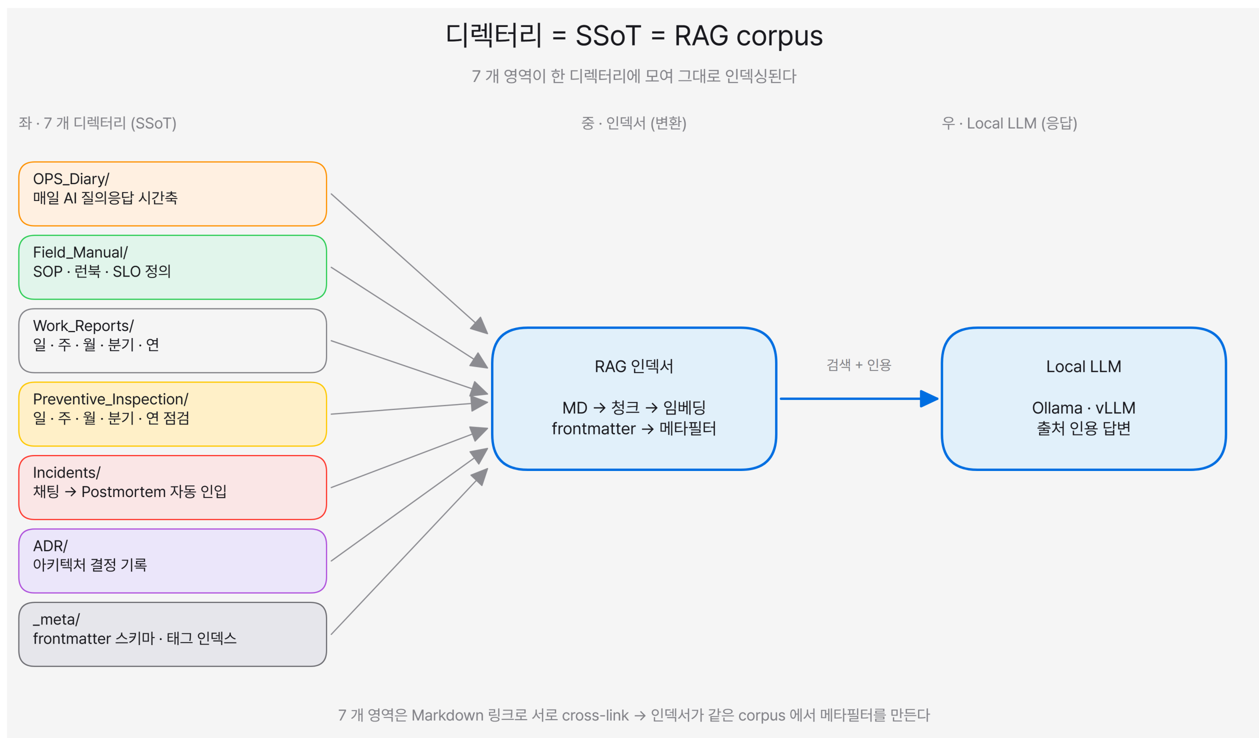
1장에서 운영 지식이 사내 메신저·티켓·머릿속 세 곳에 흩어지면서 사람과 AI 가 서로 다른 책장을 보게 되는 구조적 문제를 짚었습니다. 2장에서는 그 분산을 한 디렉터리로 봉합하기 위해 OpsKnow Repo 가 어떤 일곱 영역으로 구성되는지를 차례로 풀어 설명드립니다. 여러분의 운영팀이 지금 다루고 있는 문서들이 이 일곱 영역 중 어디에 1:1 로 대응하는지, 그리고 어떤 영역이 비어 있어 운영 지식의 학습 순환이 끊기는지를 함께 점검해 보시기 바랍니다.

일곱 영역은 OPS Diary, Field Manual, 작업 보고서 (Work Reports), 장애 예방점검 (Preventive Inspection), 장애·Incident 관리, Local LLM RAG 소스, ADR (Architecture Decision Record, 아키텍처 결정 기록) 로 나뉩니다. 각 영역은 단독으로도 가치가 있지만, 서로 cross-link 될 때 비로소 운영 지식의 가치가 비선형으로 늘어납니다. 어제 작성된 OPS Diary 한 줄이 오늘 Postmortem 의 가장 가까운 컨텍스트가 되고, 그 두 문서가 함께 다음 분기의 예방점검 체크리스트의 근거가 됩니다 [S33].

운영팀이 보유한 문서 다수가 이미 어떤 형태로든 존재합니다. 런북은 Confluence 페이지에, 작업 일지는 Notion 데이터베이스에, 인시던트 채팅은 사내 메신저 채널에 흩어져 있을 가능성이 높습니다 [S17][S49]. 본 장이 제시하는 일곱 영역은 그 흩어진 문서를 *새로 작성하라* 는 요구가 아니라, *기존 문서를 어느 영역으로 옮기면 사람과 AI 가 동시에 읽을 수 있게 되는가* 를 보여주는 지도입니다.

여러분께 한 가지 점검 질문을 먼저 드립니다. 우리 팀의 운영 문서 100건을 1시간 안에 일곱 영역에 1:1 매핑할 수 있을까요. 매핑이 불가능한 문서가 다수라면 그 문서들은 분류 표류 상태이며, RAG 인덱서가 메타필터를 만들 수 없는 상태입니다 [S55]. 매핑이 잘 되는 문서가 다수라면 이미 디렉터리 표준 도입의 절반은 끝난 셈입니다.

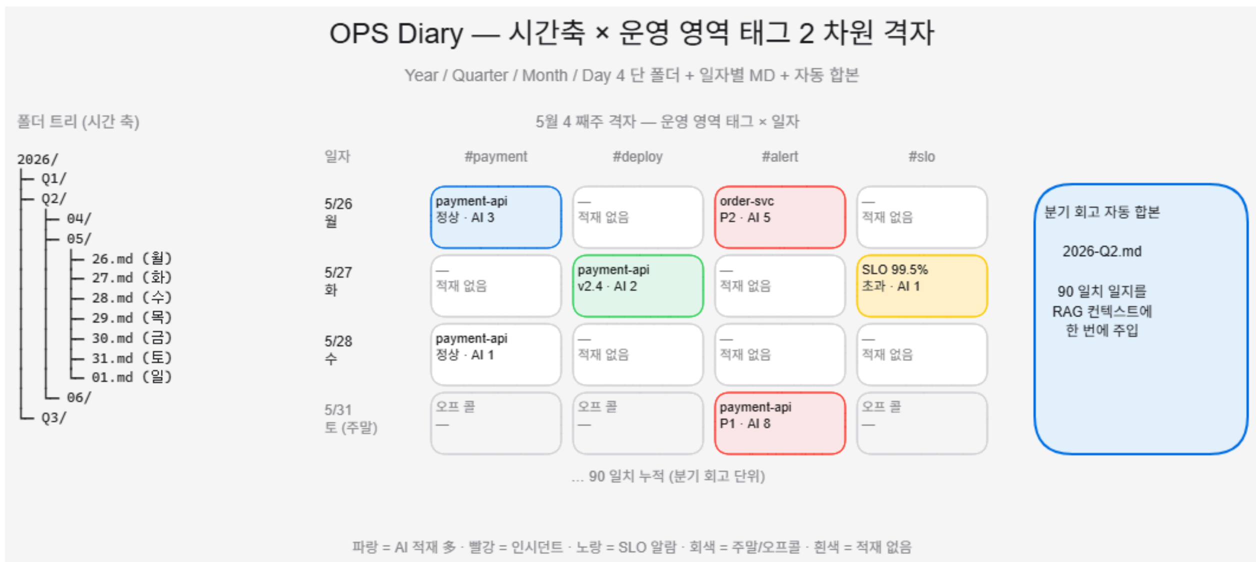
[FIGURE: directory-tree] **캡션**: OpsKnow Repo 의 7개 영역이 하나의 디렉터리에 어떻게 배치되는지 **의도**: 좌측에 디렉터리 트리 (`OPS_Diary/` , `Field_Manual/` , `Work_Reports/` , `Preventive_Inspection/` , `Incidents/` , `ADR/` , `_meta/`). 각 노드에서 우측으로 화살표가 뻗어 RAG 인덱서 (중앙) 로 수렴합니다. 인덱서에서 다시 Local LLM 으로 화살표가 이어집니다. 7개 영역이 cross-link 됨을 점선으로 표현하고, 색상은 영역별로 1색씩 부여하되 인덱서와 LLM 은 강조색으로 처리합니다. 독자가 한 화면에서 "디렉터리 = SSoT = RAG corpus" 의 등식을 즉시 알아볼 수 있도록 좌·중·우 3분할 레이아웃으로 배치합니다.



2.1 OPS Diary — AI 와의 일일 질의응답을 시간축에 박는 운영 일지

일곱 영역 중 가장 자주 갱신되는 영역이 OPS Diary 입니다. 매일 운영팀이 AI 와 주고받은 질의응답·당일 상태·짧은 메모를 시간축에 박는 운영 일지로, 운영 지식이 휘발되지 않고 영구화되는 1차 진입점에 해당합니다. 본 절에서는 일자별 파일 명명 규칙, AI 질의응답 적재 패턴, 태그 인덱스 자동 생성까지 세 항을 차례로 다룹니다.

[FIGURE: ops-diary-timeline] **캡션**: OPS Diary 의 Year/Quarter/Month/Day 폴더 + 일자별 MD + 태그 인덱스 **의도**: 시간축을 좌측 세로축으로 두고, 운영 영역 태그를 우측 가로축으로 둔 2차원 격자 그림입니다. 각 셀은 일자별 MD 파일의 압축 카드 형태로, 서비스명·상태·AI 답변 건수를 한 줄로 보여줍니다. 평일과 주말의 색상을 구분하여 운영 리듬을 강조하고, 분기 회고 시점에 자동 합쳐지는 합본 카드를 우측 하단에 별도 표시합니다.




```
2026/
├ Q1/
├ Q2/
│ ├ 04/
│ ├ 05/
│ │ ├ 26.md (월)
│ │ ├ 27.md (화)
│ │ ├ 28.md (수)
│ │ ├ 29.md (목)
│ │ ├ 30.md (금)
│ │ ├ 31.md (토)
│ │ └ 01.md (일)
│ └ 06/
└ Q3/
```

2.1.1 일자별 MD 파일 명명 규칙 — `OPS_Diary/2026/Q2/2026-05-31.md`

OPS Diary 의 파일 명명 규칙은 Year/Quarter/Month/Day 4단계 폴더와 일자 파일명으로 이루어집니다. `OPS_Diary/2026/Q2/2026-05-31.md` 가 표준 형식이며, 모든 일자별 파일이 같은 패턴을 따릅니다. 이 패턴은 검색·정렬·정기 회고 세 목적을 모두 디렉터리 구조만으로 풀어냅니다. Year 폴더는 연도 단위 백업·아카이브의 단위가 되고, Quarter 폴더는 분기 회고의 자연스러운 합본 단위가 되며, 일자 파일명은 RAG 인덱서가 시간 메타필터를 만들 때 1차 키가 됩니다 [S7].

파일명 컨벤션이 일관되지 않으면 RAG 인덱서는 시간 메타필터를 만들 수 없게 됩니다. 예를 들어 어떤 파일은 `2026-05-31.md` 이고, 다른 파일은 `20260531.md` 이며, 또 다른 파일은 `5월 31일.md` 라면 인덱서는 시간 정렬을 위해 3가지 규칙을 모두 학습해야 합니다 [S55]. 운영자가 분기 회고에서 "지난 분기 결제 서비스 관련 일지만 모아 주세요" 라고 AI 에게 요청했을 때, 인덱서가 시간 메타필터를 즉시 만들 수 없으면 LLM 은 답변 품질이 떨어진 채로 출력합니다. 그러므로 파일명 컨벤션을 디렉터리 진입 시점에 강제하는 것이 첫 번째 거버넌스 관문입니다.

다음 디렉터리 트리는 일주일치 OPS Diary 의 예시입니다. `OPS_Diary/2026/Q2/2026-05-26.md` , `2026-05-27.md` , `2026-05-28.md` , `2026-05-29.md` , `2026-05-30.md` 다섯 평일 파일과 주말 2일이 같은 폴더에 평면 배치됩니다. 한 파일의 본문은 frontmatter 6 필드 (`date` , `author` , `services` , `tags` , `rag_visible` , `confidence`) 뒤에 그날의 AI 질의응답 섹션이 4~10개 박혀 있는 형식입니다. 분기 회고 시점에는 `OPS_Diary/2026/Q2/` 폴더 하나만 통째로 RAG 컨텍스트에 주입하면 90일치 일지가 한 번에 검색 범위로 들어옵니다 [S7][S38].

여러분께 점검 질문을 드립니다. 우리 팀의 일자별 파일이 분기 회고 시점에 자동 합쳐질 수 있는 구조인가요. 자동 합쳐지지 않는다면 분기마다 합본 작업에 반나절을 쓰고 있을 가능성이 높으며, 그 반나절이 OpsKnow Repo 디렉터리 표준 도입의 정량적 ROI 가 됩니다.

2.1.2 AI 와의 질의응답 적재 패턴 — 질문·답변·출처·태그 네 요소

일자별 MD 파일 안에 AI 질의응답 1건이 어떤 형태로 박히는지가 이 항의 주제입니다. 표준 적재 패턴은 네 가지 요소로 이루어집니다. 첫째 사람이 던진 질문, 둘째 AI 가 생성한 답변, 셋째 답변의 출처 (`Field_Manual/payment-api/runbook.md:42` 같은 파일명과 라인 번호), 넷째 운영 영역 태그 (`#payment` , `#deploy` , `#alert`) 입니다 [S33].

frontmatter 에는 `author: ai` 가 박혀 들어와 운영자가 한눈에 "이 답변은 AI 가 만들었다" 는 출처를 알 수 있게 합니다. 4 요소 패턴이 표준화되지 않으면 후속 RAG 가 다시 읽을 때 메타 가중치를 부여할 수 없습니다. 예를 들어 출처가 없는 답변과 출처가 있는 답변을 동등 가중치로 인덱싱하면 RAG 의 답변 신뢰도가 떨어집니다 [S38].

다음은 OPS Diary 1일치 적재 예시입니다. frontmatter 가 `date: 2026-05-31` , `author: human` , `services: [payment-api]` , `tags: [deploy, post-deploy-check]` , `rag_visible: true` , `confidence: high` 6 필드로 시작하고, 본문은 `## 14:32 — 결제 API 배포 후 점검` 같은 시간 헤더 아래에 질문·답변·출처·태그 4 요소가 순서대로 박힙니다. 답변 섹션의 마지막에는 `> 출처: Field_Manual/payment-api/runbook.md:42` 같은 인용 한 줄이 자동으로 들어가, 운영자가 답변의 근거 파일을 즉시 열어볼 수 있습니다 [S33][S47].

적재 패턴의 표준화는 도구 선택에 앞서 정해져야 합니다. AnythingLLM·Khoj 같은 RAG 도구는 적재 패턴이 표준화된 corpus 를 받으면 답변 품질이 즉시 올라가지만, 표준화되지 않은 corpus 를 받으면 어떤 도구도 운영자가 만족할 만한 답변을 만들지 못합니다 [S33][S38]. 운영팀이 도구를 도입하기 전에 적재 패턴 표준 1장을 합의하는 데 반나절을 투자하는 것이 도구 도입 6주의 절반을 절약합니다.

2.1.3 태그 인덱스 자동 생성 — Obsidian Dataview 또는 자체 스크립트

일자별 MD 파일이 쌓이면 태그 인덱스가 운영 지식 탐색의 진입점이 됩니다. 운영자가 "지난 한 달간 `#alert` 태그가 박힌 일지만 모아 주세요" 라고 요청했을 때, 인덱스가 자동 생성되어 있어야 답변이 즉시 나옵니다. 자동화 경로는 두 갈래입니다. Obsidian Dataview 플러그인을 사용해 vault 안에서 DQL 쿼리로 실시간 합치는 방법과, GitHub Action 또는 cron 스크립트로 일 1회 `OPS_Diary/_index.md` 를 자동 갱신하는 방법입니다 [S7][S10].

두 경로의 운영 부담을 비교해 보십시오. Obsidian Dataview 는 사용자가 vault 를 열 때마다 실시간으로 쿼리를 실행하므로 인덱스가 항상 최신 상태로 유지됩니다. 단 vault 가 수만 개 파일로 커지면 쿼리 응답 시간이 1~3초로 늘어나며, Obsidian 클라이언트에 종속되어 VSCode 사용자는 같은 뷰를 볼 수 없습니다. GitHub Action 은 일 1회 또는 PR 머지 시점에 인덱스를 정적 MD 로 갱신하므로 어느 클라이언트에서도 동일한 뷰가 보장됩니다. 단 최신 반영이 24시간 지연될 수 있습니다 [S7].

자동화 경로	최신 반영 주기	운영 부담	클라이언트 종속	권장 시나리오
Obsidian Dataview	실시간	낮음 (설정 30분)	Obsidian 전용	단일 팀, 1만 파일 이하
GitHub Action	일 1회 또는 PR 단위	중간 (워크플로 작성 반나절)	무관	다중 팀, vault 공유 환경
cron 스크립트	cron 주기	중간 (스크립트 작성 반나절)	무관	온프레미스, GitHub 미사용

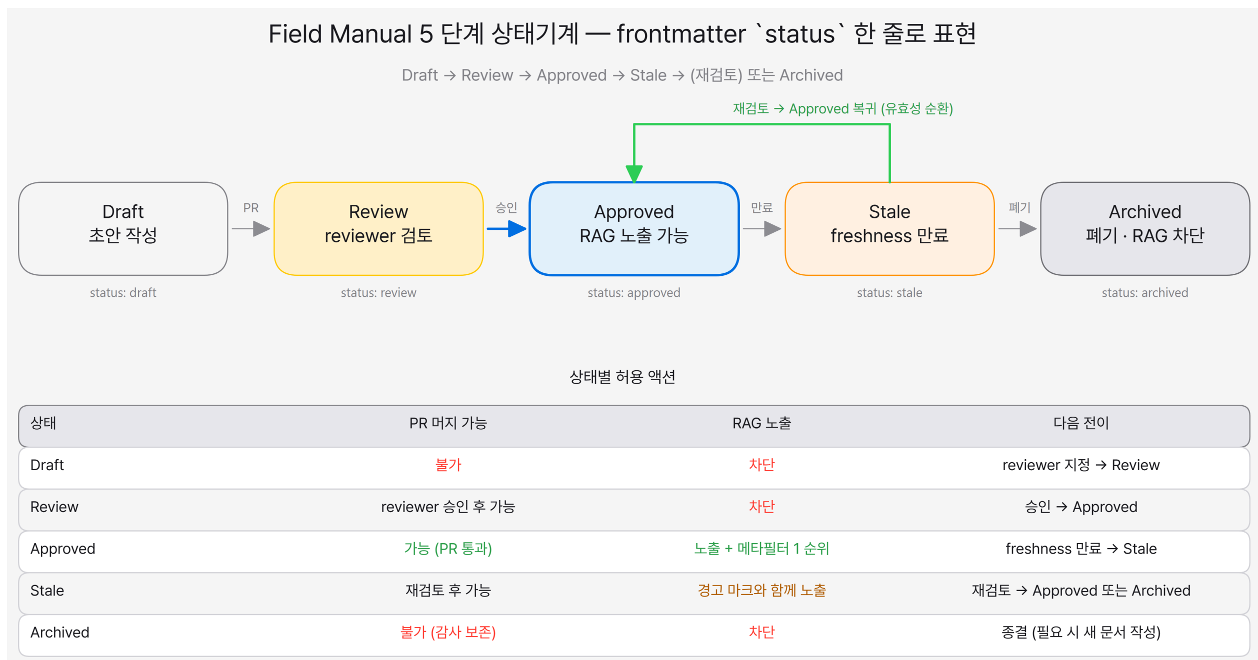
태그 인덱스 자동화가 없으면 한 달 만에 운영팀이 손을 놓게 됩니다. 일자별 일지를 작성하는 사람은 그날의 메모만 남기는 것이지 지난 한 달치 일지를 검색하려는 의도가 약하기 때문에, 인덱스가 자동화되지 않으면 일지의 가치를 실감할 기회를 잃습니다 [S49]. 도입 비용은 반나절 (Dataview) 에서 일주일 (자체 스크립트 + CI 통합) 사이입니다. 그 정도 투자가 일지 작성 문화의 정착에 결정적입니다.

여러분의 운영팀에 적합한 자동화 경로를 결정하기 위한 점검 질문 한 가지를 드립니다. 다음 분기 회고 시점에 지난 90일치 `#post-mortem` 태그 일지 합본을 운영팀 누구든 5초 안에 화면에 띄울 수 있는 상태인가요. 그 답이 "아니요" 라면 자동화 도입의 우선순위는 매우 높습니다.

2.2 Field Manual — 시스템·소프트웨어별 매뉴얼과 SOP 의 현장 책장

OPS Diary 가 일자별 흐름이라면 Field Manual 은 시스템·소프트웨어별로 정리된 매뉴얼과 SOP (Standard Operating Procedure, 표준 운영 절차) 의 현장 책장입니다 [S19]. 운영자가 알람을 받았을 때, 신규 입사자가 운영 방법을 익힐 때, 분기 점검을 수행할 때 모두 Field Manual 을 1차로 펴 봅니다. 본 절에서는 매뉴얼 디렉터리 분할, SOP frontmatter 표준, 유효 만료일 알림까지 세 항을 다룹니다.

[FIGURE: field-manual-lifecycle] **캡션**: Field Manual 의 Draft → Review → Approved → Stale → Archived 5단계 상태기계 **의도**: 5개 노드 (Draft, Review, Approved, Stale, Archived) 를 횡으로 배치하고 상태 전이 화살표를 두십시오. 각 상태 위에 frontmatter 의 `status: draft|review|approved|stale|archived` 표기를 한 줄로 표기하고, 아래쪽에 각 상태에서 허용되는 액션 (PR 머지 가능 여부, RAG 노출 여부) 을 표 형태로 정리합니다. Stale → Approved 재검토 사이클은 별도 강조 색으로 표시하여 유효성 관리의 순환을 시각화합니다.



상태	PR 머지 가능	RAG 노출	다음 전이
Draft	불가	차단	reviewer 지정 → Review
Review	reviewer 승인 후 가능	차단	승인 → Approved
Approved	가능 (PR 통과)	노출 + 메타필터 1 순위	freshness 만료 → Stale
Stale	재검토 후 가능	경고 마크와 함께 노출	재검토 → Approved 또는 Archived
Archived	불가 (감사 보존)	차단	종결 (필요 시 새 문서 작성)

2.2.1 매뉴얼 디렉터리 분할 — 시스템별 vs 소프트웨어별

Field Manual 의 디렉터리 분할에는 두 가지 패턴이 있습니다. 시스템별 분할은 `Field_Manual/payment-api/`, `Field_Manual/inventory-svc/` 처럼 운영 대상 시스템 단위로 폴더를 나누는 방식입니다. 소프트웨어별 분할은 `Field_Manual/postgres/`, `Field_Manual/kafka/`, `Field_Manual/redis/` 처럼 운영 대상 소프트웨어 (또는 미들웨어) 단위로 폴더를 나누는 방식입니다. GitLab Handbook 의 runbooks 디렉터리는 시스템별 분할을 채택했으며 다중 팀 환경의 사실상 표준으로 자리잡아 왔습니다 [S19][S53].

두 패턴은 트레이드오프 관계입니다. 시스템별 분할은 운영자가 "결제 API 에 알람이 떴다" 라는 상황에서 즉시 `Field_Manual/payment-api/` 를 펴면 되므로 알람 → 매뉴얼 진입이 빠릅니다. 단 같은 소프트웨어 (예:

Postgres) 의 운영 노하우가 여러 시스템 폴더에 분산되어 중복·표류가 발생합니다. 소프트웨어별 분할은 Postgres 관련 모든 노하우가 한 폴더에 모이지만, 운영자가 "결제 API" 라는 상황에서 어느 폴더를 펴야 할지 한 번 더 생각해야 합니다 [S19].

분할 기준	알람 → 매뉴얼 속도	노하우 집중도	다중 팀 적합성	신규 입사자 학습	RAG 메타필터	권장 시나리오
시스템별 (payment-api/)	빠름	분산	높음	시스템 관점	service 메타	다중 팀, 마이크로서비스
소프트웨어별 (postgres/)	보통	집중	보통	기술 스택 관점	software 메타	모놀리식, 소수 핵심 미들웨어
혼합 (둘 다)	빠름	집중	높음	양쪽 모두	두 메타 동시	대규모, 분할 비용 감내 가능

다중 팀 환경에서 디렉터리 분할이 잘못되면 운영자가 매뉴얼을 찾지 못해 결국 사내 메신저로 돌아갑니다 [S49]. 사내 메신저로 돌아가는 순간 운영 지식은 다시 무덤으로 흡수되며, OpsKnow Repo 도입의 효과는 무효화됩니다. 그러므로 디렉터리 분할 합의는 도입 초기 가장 중요한 의사결정 한 가지로 다뤄야 합니다. 다중 팀·다중 마이크로서비스 환경에서는 시스템별 분할을 1차로 선택하고, Postgres·Kafka 같은 공유 미들웨어에 한정해 소프트웨어별 폴더를 보조로 두는 혼합 패턴이 현실적입니다 [S19][S53].

2.2.2 SOP 의 frontmatter 표준 — `owner` / `reviewer` / `freshness_until`

Field Manual 의 모든 SOP/런북 MD 파일에는 표준 frontmatter 3 필드가 박혀 있어야 합니다. `owner` 는 이 문서를 책임지는 사람 또는 팀, `reviewer` 는 분기별 검토를 수행하는 사람, `freshness_until` 은 문서가 유효하다고 보장되는 기한입니다 [S55][S6]. 세 필드가 없으면 RAG 가 우선순위·가중치를 만들 수 없습니다. 예를 들어 운영자가 "결제 API 의 최신 런북" 을 요청했을 때, RAG 인덱서가 `freshness_until` 을 보고 만료된 런북을 답변에서 제외하지 못하면 6개월 전 런북이 그대로 답변으로 돌아옵니다.

표준 frontmatter 예시는 다음 7 필드 조합입니다. `owner: @sre-payment` (팀 핸들), `reviewer: @kim-jihoon` (개인 핸들), `freshness_until: 2026-09-30` (ISO 날짜), `service: payment-api` , `severity: P1` (장애 심각도), `doc_type: how-to` (Diátaxis 4분면), `status: approved` (5단계 상태기계 중 하나). 이 7 필드가 모든 SOP 의 최소 공통 메타가 되며, 부록 C 의 표준 스키마와 1:1 정합합니다 [S55][S62].

5단계 상태기계 (Draft → Review → Approved → Stale → Archived) 는 `status` 필드와 1:1 매핑됩니다. Draft 단계의 문서는 RAG 에 노출되지 않으며, Review 단계에서는 reviewer 의 승인을 기다립니다. Approved 단계에서 RAG 에 노출되고, `freshness_until` 만료 시점에 자동으로 Stale 로 전이됩니다. Stale 단계에서 owner 에게 재검토 요청 알림이 가고, 재검토 후 Approved 로 돌아가거나 Archived 로 폐기됩니다 [S6][S19]. 이 5단계가 frontmatter 한 줄로 표현된다는 점이 SaaS 위키 대비 OpsKnow Repo 의 본질 차이 한 가지입니다.

여러분의 운영팀에서 현재 매뉴얼 중 `owner` 가 적혀 있는 비율이 얼마나 되는지 점검해 보십시오. 절반을 밑돈다면 매뉴얼 거버넌스가 부재한 상태이며, 핵심 운영자 1명의 이직이 곧 매뉴얼 절반의 고아화로 이어집니다

[S49]. 5단계 상태기계의 도입은 매뉴얼 1건당 frontmatter 7 필드를 박는 1차 작업과, GitHub Action 또는 cron 으로 상태 전이를 자동화하는 2차 작업으로 나뉩니다. 1차 작업은 매뉴얼 100건 기준 1주 정도, 2차 작업은 반나절 정도로 완료 가능합니다 [S19].

2.2.3 유효 만료일 알림 — GitHub Action 또는 cron 스크립트

`freshness_until` 필드가 박혀 있어도 알림이 없으면 무용지물입니다. 만료 임박 시점에 owner 에게 자동 알림이 가는 메커니즘이 유효성 관리의 마지막 고리입니다. 구현 경로는 GitHub Action 의 stale-detector 워크플로와 cron 스크립트 두 갈래입니다. GitHub Action 은 일 1회 또는 주 1회 트리거되어 모든 MD 의 `freshness_until` 을 스캔하고, 30일 이내 만료 임박 파일에 대해 owner 에게 GitHub Issue 를 자동 생성하거나 사내 메신저 알림을 보냅니다 [S6][S19].

cron 스크립트는 온프레미스 환경 또는 GitHub Action 비사용 조직에서 동일 기능을 수행합니다. Python 또는 shell 스크립트로 frontmatter 의 `freshness_until` 을 파싱하여 만료 임박 파일 목록을 사내 메신저 봇 또는 이메일로 발송합니다. 두 경로 모두 알림 수신자는 frontmatter 의 `owner` 필드에서 자동 추출되므로 별도 매핑 테이블이 필요 없습니다 [S6][S19].

자동화 경로	트리거	알림 채널	도입 비용	온프레미스 적합성
GitHub Action stale-detector	cron 주기 (예: 매주 월요일)	GitHub Issue, 사내 메신저	반나절 (워크플로 1개)	낮음 (외부 호출 필요)
자체 cron 스크립트	crontab	사내 메신저 봇, 이메일	1일 (스크립트 + 봇)	높음 (전부 사내)
Obsidian 플러그인	vault 열 때	클라이언트 토스트	30분 (플러그인 설치)	무관

유효 만료일 알림이 없으면 런북이 6개월 만에 거짓말이 됩니다 [S52]. 6개월 전에 작성된 결제 API 런북이 그동안 변경된 배포 절차·환경변수·외부 의존성을 반영하지 못한 채 RAG 답변에 인용되면, 운영자가 그 답변을 따라 장애 대응을 시작했다가 잘못된 절차에 시간을 허비할 수 있습니다. 그러므로 `freshness_until` 만료 알림은 *기술적 자동화* 가 아니라 *운영 거버넌스의 1차 방어선* 으로 다뤄야 합니다 [S6].

만료 알림의 자동화 비용 대비 런북 최신성 향상의 기대치를 산정해 보십시오. 운영팀 1팀 기준 매뉴얼 50~100건 규모에서 자동화 도입 비용은 1일 이내이며, 런북 최신성 향상으로 인한 장애 대응 시간 단축 (가상 시나리오 기준 인시던트 1건당 5~15분) 의 누적 가치는 분기 단위로 회수됩니다 [S52][S19].

2.3 작업 보고서 (Work Reports) — 일·주·월·분기·연간 이력의 누적

작업 보고서 영역은 일·주·월·분기·연 다섯 주기로 운영 활동의 이력을 누적하는 영역입니다 [S19]. OPS Diary 가 매일의 흐름이라면 작업 보고서는 그 흐름을 정해진 주기로 정리한 산출물입니다. 본 절에서는 보고 주기별 디렉터리, 5종 템플릿의 공통 frontmatter, 자동 롤업까지 세 항을 다룹니다.

2.3.1 보고 주기별 디렉터리 — `daily/` `weekly/` `monthly/` `quarterly/` `annual/`

작업 보고서 디렉터리는 5주기 폴더로 나뉩니다. `Work_Reports/daily/2026-05-31.md` 는 일일 보고, `Work_Reports/weekly/2026-W22.md` 는 주간 보고, `Work_Reports/monthly/2026-05.md` 는 월간 보고, `Work_Reports/`

quarterly/2026-Q2.md 는 분기 보고, Work_Reports/annual/2026.md 는 연간 보고입니다 [S19]. 각 주기 폴더는 시간 격자에 자연스럽게 정렬되며, 분기 회고 시점에 폴더 트리만 봐도 90일치 보고서가 한눈에 들어옵니다.

5주기 분할이 없으면 분기 회고가 1주일 작업이 됩니다. 일일 보고가 메일 본문에 박혀 있고 주간 보고가 PowerPoint 파일에 박혀 있고 월간 보고가 Notion 페이지에 박혀 있는 흩어진 상태에서는, 분기 회고 시점에 "지난 분기 보고서 모두 모아 주세요" 라는 한 줄 요청이 사실상 불가능합니다 [S49][S50]. 디렉터리 5분할은 그 흩어짐을 한 폴더 트리로 봉합합니다.

여러분의 운영팀의 현재 작업 보고서가 어디에 흩어져 있는지 1줄로 진단해 보십시오. 메일·사내 메신저·Notion·PowerPoint·Confluence 다섯 곳에 분산되어 있다면 분기 회고의 절반은 보고서 수집에 쓰이고 있으며, 그 시간이 OpsKnow Repo 도입의 정량적 ROI 한 가지로 환산 가능합니다.

2.3.2 보고서 템플릿 5종의 공통 frontmatter

5종 보고서가 공유하는 frontmatter 필드 3개와 각 주기 고유 필드를 분리하여 표준화합니다. 공통 필드는 period: daily|weekly|monthly|quarterly|annual , owner: @팀핸들 , services: [payment-api, inventory-svc] 세 가지입니다 [S7][S19]. 공통 frontmatter 가 박혀 있어야 자동 롤업이 가능합니다. 일일 보고 5건의 services 필드를 모두 읽어 주간 보고에 합치는 스크립트는 공통 필드가 있을 때만 작동합니다.

주기 고유 필드는 보고서의 성격에 따라 달라집니다. 일일 보고는 incidents_today: [INC-2026-031] 같은 그날의 인시던트 ID 목록을 추가하고, 주간 보고는 weekly_metrics: {availability: 99.92, mttr_minutes: 23} 같은 주간 집계 메트릭을 추가하며, 분기 보고는 quarterly_themes: [secret rotation, db migration] 같은 분기 주제를 추가합니다 [S7][S19]. 이러한 고유 필드는 부록 B 의 5종 템플릿과 1:1 정합합니다.

보고 주기	공통 필드	고유 필드 예시	자동 롤업 대상	권장 작성 시점
일일 (daily)	period, owner, services	incidents_today, deployments_today	주간 보고	일과 마감 시
주간 (weekly)	period, owner, services	weekly_metrics, action_items	월간 보고	금요일 오후
월간 (monthly)	period, owner, services	monthly_kpis, risks_observed	분기 보고	월 마지막 평일
분기 (quarterly)	period, owner, services	quarterly_themes, slo_review	연간 보고	분기 마지막 주
연간 (annual)	period, owner, services	annual_strategy, year_in_review	(최종 합본)	12월 셋째 주

5종 보고서 중 어느 한 주기가 빠지면 누적 순환이 끊깁니다. 일일 보고가 없으면 주간 합본이 빈약해지고, 주간 보고가 없으면 월간 보고가 즉흥적이 되며, 분기 보고가 없으면 연간 회고가 회사 차원의 형식 행사로 흐릅니다 [S49][S19]. 그러므로 5종 템플릿의 도입은 *한꺼번에* 가 권장됩니다. 5종을 모두 표준화하는 데 필요한 합의 시간은 운영팀 1팀 기준 반나절 정도이며, 그 반나절이 분기 회고의 작업량을 1/4 로 줄입니다.

2.3.3 자동 롤업 — 일일 5건 → 주간 1건 → 월간 1건 통합

5주기 보고서의 가장 큰 도입 마찰은 *작성 부담* 입니다. 운영자가 매일 1건, 매주 1건, 매월 1건, 매분기 1건, 매년 1건의 보고서를 처음부터 끝까지 손으로 작성한다면 보고서 작성이 본업을 침식하게 됩니다. OpsKnow Repo 가 제시하는 해법은 *AI 가 통합 초안을 만들고 사람이 승인하는 자동 롤업* 입니다 [S33][S38].

자동 롤업 스크립트는 다음 4 단계로 작동합니다. 첫째, 주간 보고 시점에 지난 7일의 `Work_Reports/daily/*.md` 5건을 모두 읽어 LLM 컨텍스트로 주입합니다. 둘째, LLM 이 5건의 공통 주제·반복 인시던트·메트릭 추이를 추출하여 주간 보고 초안을 생성합니다. 셋째, 운영자가 초안을 검토하고 frontmatter 의 `author: ai` , `confidence: medium` 을 박은 채로 PR 을 엽니다. 넷째, reviewer 승인 후 머지되어 `Work_Reports/weekly/2026-W22.md` 가 확정됩니다 [S33].

같은 패턴이 주간 4건 → 월간 1건, 월간 3건 → 분기 1건, 분기 4건 → 연간 1건으로 확장됩니다. 의존 그래프는 일 → 주 → 월 → 분기 → 연 일방향으로 흐르며, 하류 보고서는 상류 보고서를 컨텍스트로 받습니다 [S38]. 이 누적 순환 구조가 작동하면 분기 보고 작성에 들어가는 시간은 *초안 검토와 메트릭 확인* 으로만 한정되며, 작성 시간의 60% 이상이 절감 가능합니다 [S33].

자동 롤업 도입의 의사결정 포인트는 *AI 초안의 신뢰도 임계값* 입니다. `confidence: low` 초안은 자동 머지를 차단하고 reviewer 1인 이상의 의무 승인을 강제하며, `confidence: medium` 이상은 빠른 승인 경로 (1인 승인) 를 허용하는 정책이 일반적입니다 [S61]. 이 정책이 frontmatter 한 줄로 표현되고 GitHub Action 의 PR 머지 게이트에 박혀 있어야, AI 가 잘못된 추론을 한 보고서가 자동으로 확정되는 일을 막을 수 있습니다.

2.4 장애 예방점검 (Preventive Inspection) — AI 가 주기 수행하는 점검의 적재

장애 예방점검 영역은 AI Agent 가 주기적으로 점검을 수행하고 그 결과를 MD 로 적재하는 영역입니다. 사람이 매번 직접 점검 항목을 작성하지 않아도 AI 가 표준 카탈로그를 따라 점검을 수행하고, 사람은 결과를 검토·승인만 합니다 [S22][S47]. 본 절에서는 점검 항목 카탈로그, AI 자동 수행, 사람 승인 게이트까지 세 항을 다룹니다.

2.4.1 점검 항목 카탈로그 — AWS W-A OPS 1~11 흡수

점검 항목을 처음부터 자체 정의하면 누락이 발생합니다. 운영 점검의 표준 카탈로그는 이미 업계에 검증되어 있으며, 그중 대표가 AWS Well-Architected Framework 의 Operational Excellence Pillar 입니다 [S22]. OPS 1~11 의 11개 항목이 운영 우수성의 표준 체크리스트이며, OpsKnow Repo 는 이 11개 항목을 그대로 점검 카탈로그로 흡수합니다. 각 항목에 점검 주기 (일·주·월·분기·연 다섯 중 하나) 와 자동화 가능 여부 (auto, semi-auto, manual 셋 중 하나) 의 태그를 부여합니다 [S22][S21].

OPS 1~11 의 항목명을 5주기와 교차한 매트릭스가 점검 캘린더의 1차 형태입니다. 예를 들어 OPS 4 (텔레메트리 설계) 는 주 단위 점검 항목이 되고, OPS 8 (워크로드 관측) 은 일 단위 점검 항목이 되며, OPS 11 (학습 누적) 은 분기 단위 점검 항목이 됩니다 [S22]. 이 매트릭스는 부록 D 에서 확장되며, 각 셀의 점검 결과 MD 가 `Preventive_Inspection/<period>/<ops-id>/2026-W22.md` 형식으로 적재됩니다.

점검 항목 (OPS)	권장 주기	자동화 수준	RAG 인덱싱 가중	사람 승인 필수
OPS 4 (텔레메트리 설계)	주 1회	semi-auto	중간	권장
OPS 5 (런북 정의)	월 1회	auto	높음	권장

점검 항목 (OPS)	권장 주기	자동화 수준	RAG 인덱싱 가중	사람 승인 필수
OPS 6 (배포 절차)	일 1회	auto	높음	필수
OPS 8 (워크로드 관측)	일 1회	auto	중간	권장
OPS 11 (학습 누적)	분기 1회	manual	높음	필수

여러분의 운영팀이 현재 수행하는 점검의 OPS 1~11 커버리지를 측정해 보십시오. 11개 중 5개 이하만 정기적으로 수행되고 있다면 점검 카탈로그의 절반이 비어 있는 상태이며, AI Agent 도입의 가장 큰 ROI 가 *비어 있던 점검 항목을 채우는 것* 으로 도출됩니다 [S22]. OPS 1~11 의 카탈로그 흡수는 코드 수정이 아닌 frontmatter 스키마 정의만으로 가능하며, 운영팀 1팀 기준 1~2일 안에 1차 도입이 가능합니다.

2.4.2 AI Agent 가 자동 수행한 점검 결과의 frontmatter — `author: ai` + `confidence`

AI Agent 가 자동 수행한 점검 결과 MD 의 frontmatter 표준은 3 필드가 핵심입니다. `author: ai` 는 출처가 사람이 아닌 AI 임을 명시하고, `confidence: low|medium|high` 는 AI 가 스스로 평가한 답변 신뢰도이며, `model: gpt-oss-20b@2026-05` 는 점검을 수행한 모델 이름과 버전입니다 [S47][S42]. 이 3 필드가 메타에 박히지 않으면 AI 점검 결과는 무엇이든 머지될 위험이 있습니다.

`confidence` 의 세 단계는 각각 다른 정책에 매핑됩니다. `confidence: high` 는 사람 reviewer 1인의 가벼운 승인으로 머지 가능합니다. `confidence: medium` 은 reviewer 가 점검 결과의 본문을 짧게 확인한 뒤 승인합니다. `confidence: low` 는 reviewer 가 점검 항목을 처음부터 다시 수행한 뒤 사람 작성으로 대체하거나, AI 결과를 폐기합니다 [S47][S61]. 이 3단계 정책이 frontmatter 와 PR 머지 게이트에 함께 박혀 있어야 AI 환각의 1차 방어선이 작동합니다.

다음은 AI 점검 결과 MD 1건의 예시 구조입니다. frontmatter 가 `author: ai` , `confidence: medium` , `model: gpt-oss-20b@2026-05` , `ops_id: OPS-6` , `service: payment-api` , `period: 2026-W22` , `reviewer: @sre-payment` 7 필드로 시작하고, 본문은 점검 항목별 결과 (예: "배포 절차 6단계 모두 자동화 검증 통과") 와 발견된 이상 (예: "환경변수 DEPLOY_TIMEOUT 이 매뉴얼에는 30s 로 명시되어 있으나 실제 적용값은 60s") 두 섹션으로 구성됩니다 [S47].

AI 자동 점검의 신뢰도 임계값 정책을 정의하는 것이 도입의 핵심 의사결정입니다. 보수적 정책은 `confidence: high` 만 자동 알림을 보내고, `medium` 은 reviewer 큐에 쌓아두며, `low` 는 알림 없이 폐기합니다. 적극적 정책은 세 단계 모두 알림을 보내되 우선순위만 다르게 표시합니다. 두 정책 사이의 선택은 운영팀의 AI 도구 신뢰 수준에 따라 달라집니다 [S47][S42].

2.4.3 사람 reviewer 의 승인 게이트

AI 가 작성한 점검 결과가 PR 로 들어와 사람 reviewer 1인 이상의 승인 후 머지되는 게이트가 이 항의 주제입니다. 게이트는 GitHub PR 의 required reviewers 설정으로 구현 가능하며, frontmatter 의 `confidence` 필드가 reviewer 자동 할당의 기준이 됩니다. `confidence: low` 인 PR 은 owner 팀의 시니어 1인 + 도메인 전문가 1인 두 명의 승인을 강제하고, `medium` 은 owner 팀 1인, `high` 는 자동 머지 옵션이 허용됩니다 [S61][S60].

게이트의 운영 부담을 줄이는 핵심은 *reviewer 자동 할당의 자동화* 입니다. PR 의 변경 파일이 `Preventive_Inspection/weekly/OPS-6/*.md` 라면 CODEOWNERS 파일에 `Preventive_Inspection/weekly/OPS-6/ @sre-payment` 로 박혀 있어, PR 생성 시점에 자동으로 reviewer 가 할당됩니다. 운영자가 매번 reviewer 를 수동으로 지정할 필요가 없습니다 [S60].

confidence	reviewer 자동 할당	머지 게이트	알림 채널	예상 응답 시간
high	owner 팀 1인	1인 승인	사내 메신저 DM	1시간 이내
medium	owner 팀 1인	1인 승인 + 본문 확인	사내 메신저 채널	4시간 이내
low	owner 팀 시니어 1인 + 도메인 전문가 1인	2인 승인 + 재점검	사내 메신저 채널 + 이메일	1일 이내

게이트가 곧 AI 환각의 1차 방어선입니다 [S61]. 게이트가 없으면 AI 가 잘못된 점검 결과를 자동 머지하여 운영팀의 다음 점검 사이클이 잘못된 결과를 기반으로 시작됩니다. 그 누적이 1개월 만에 운영 매뉴얼 전체의 신뢰도를 떨어뜨립니다 [S60]. 그러므로 게이트는 *프로세스 한 단계 추가* 가 아니라 *운영 매뉴얼 신뢰도 유지의 마지막 안전망* 입니다.

여러분의 운영팀에서 reviewer 자동 할당 정책이 메타에 박혀 있는지 점검해 보십시오. CODEOWNERS 또는 동등 메커니즘이 부재하면 reviewer 할당이 항상 수동 작업이 되며, 운영자의 PR 처리 부담이 자동 롤업의 ROI 를 잠식합니다.

2.5 장애·Incident 관리 — 채팅 덤프부터 Postmortem 까지의 라이프사이클

장애·Incident 관리 영역은 인시던트 1건의 전체 라이프사이클을 한 디렉터리 트리에 모으는 영역입니다. 알람 발생 → 사내 메신저 채팅 → 타임라인 정리 → Postmortem 작성 → 액션 아이템 추출까지의 모든 산출물이 인시던트 ID 폴더 한 곳에 수렴합니다 [S23][S24]. 본 절에서는 인시던트 디렉터리 구조, raw 덤프와 정제 Postmortem 의 분리, 액션 아이템 추출까지 세 항을 다룹니다.

2.5.1 인시던트 디렉터리 구조 — `Incidents/YYYY/QN/<id>/`

인시던트 1건당 한 폴더 원칙이 이 영역의 1번 규칙입니다. `Incidents/2026/Q2/INC-2026-031/` 폴더 안에 네 파일이 표준 구조로 배치됩니다. `raw/chat.md` 는 사내 메신저 채널의 채팅 원본 덤프, `timeline.md` 는 정제된 타임라인, `postmortem.md` 는 사후 분석 본문, `action-items.md` 는 후속 조치 추적 파일입니다 [S23][S24].

파일명	내용	자동 생성 여부	RAG 노출	보존 기한
raw/chat.md	사내 메신저 원본 덤프	봇 자동	`rag_visible: false`	5년 (감사용)
timeline.md	사실 정리 타임라인	AI 초안 + 사람 정제	`rag_visible: true`	영구
postmortem.md	사후 분석 본문	AI 초안 + 사람 승인	`rag_visible: true`	영구

파일명	내용	자동 생성 여부	RAG 노출	보존 기한
action-items.md	후속 조치 항목	AI 추출 + 사람 책임자 지정	rag_visible: true	항목 완료 시까지

이 4 파일 구조의 핵심 가치는 *분기 회고 시점에 디렉터리 트리만 보면 즉시 회고 가능* 한 점입니다. 분기 회고를 진행하는 운영자가 `Incidents/2026/Q2/` 폴더 하나만 펴 봐도 그 분기의 모든 인시던트가 한눈에 들어오고, 폴더 안에 들어가면 1건당 4 파일이 표준으로 정렬되어 있어 인시던트 1건의 전체 흐름을 5분 안에 파악 가능합니다 [S23].

인시던트 1건당 한 폴더 원칙이 무너지면 사후 분석이 불가능해집니다. 예를 들어 채팅은 사내 메신저에, 타임라인은 PagerDuty 에, Postmortem 은 Confluence 에, 액션 아이템은 Jira 에 흩어져 있는 환경에서는 회고 1건당 4개 도구를 열어야 하며, 그 분산 비용이 회고 자체의 작업량보다 큽니다 [S23][S24]. 그러므로 인시던트 디렉터리 구조 표준은 인시던트 SaaS 도입의 자동화와 *별도로* 합의해야 합니다.

여러분의 운영팀에서 지난 분기 인시던트 1건당 자료가 몇 군데에 흩어져 있는지 점검해 보십시오. 4 곳 이상에 흩어져 있다면 회고 시간의 60% 이상이 자료 수집에 쓰이고 있으며, 그 시간이 OpsKnow Repo 도입의 정량적 ROI 한 가지로 환산 가능합니다.

2.5.2 raw 덤프 vs 정제 Postmortem 의 분리

사내 메신저 채팅 raw 덤프와 정제 Postmortem 의 분리는 이 영역의 두 번째 핵심 규칙입니다. raw 덤프는 감사용으로 보존하되 RAG corpus 에 노출하지 않고, Postmortem 은 별도 정제된 MD 로 분리하여 RAG 에 노출합니다 [S24][S26]. raw 의 frontmatter 는 `rag_visible: false` , `sensitivity: high` 가 박혀 있고, Postmortem 의 frontmatter 는 `rag_visible: true` , `sensitivity: medium` 이 박혀 있는 형식입니다.

구분	raw/chat.md	postmortem.md
author	bot	ai (초안) + human (승인)
rag_visible	false	true
sensitivity	high (PII 잔존 가능)	medium
톤	비공식, 채팅 그대로	공식, 사실 정리
보존 기한	5년 (감사)	영구 (학습)
검색 가능	git grep 만	RAG + grep

비공식 톤의 raw 가 RAG corpus 에 노출되면 AI 답변 품질이 떨어집니다. 채팅에는 농담·욕설·약어·내부 별칭·미확인 추정이 섞여 있으며, 이런 텍스트가 RAG 컨텍스트에 주입되면 LLM 이 사실과 추정을 구분하지 못한 채 답변을 생성합니다 [S24][S27]. 그러므로 raw → Postmortem 의 두 단계 처리는 *AI 답변 품질 보호 장치* 이기도 합니다.

raw 와 정제 Postmortem 의 분리 정책이 frontmatter 에 박혀 있는지 점검하는 것이 도입 점검 항목 한 가지입니다. `rag_visible: false` 의 기본값을 보수적으로 (default false, opt-in true) 설정하는 운영팀이 다수이며,

이는 PII 또는 민감정보 노출 위험을 우선 차단하는 정책입니다 [S30]. 적극적 정책 (default true, opt-out false) 은 운영팀의 PII 마스킹 자동화가 인입 단계에서 강제될 때만 권장됩니다.

2.5.3 액션 아이템 추출 — 후속 추적의 시작점

Postmortem 본문에는 항상 액션 아이템 섹션이 박힙니다. 그 섹션을 별도 파일 (`action-items.md`) 로 추출하여 다음 분기 회고와 연결하는 패턴이 이 항의 주제입니다 [S28][S57]. 추출 작업은 AI 가 Postmortem 본문에서 액션 아이템 후보를 자동 추출하고, 사람이 책임자를 지정하며 우선순위를 부여하는 두 단계로 이루어집니다.

`action-items.md` 의 각 항목은 frontmatter 와 본문으로 구성됩니다. frontmatter 는 `assignee: @kim-jihoon` , `priority: P1` , `due: 2026-06-15` , `incident_id: INC-2026-031` , `status: open|in-progress|done|wontfix` 5 필드를 포함합니다 [S28]. 본문은 액션 아이템의 배경·기대 효과·완료 판정 기준을 짧게 적습니다. 이 구조가 표준화되어 있어야 분기 회고 시점에 "지난 분기 액션 아이템 중 완료된 항목과 표류한 항목" 을 한 번에 집계할 수 있습니다.

라이프사이클 단계	담당	산출물	트리거
추출	AI Agent	action-items.md 초안	Postmortem 머지 시점
책임자 지정	운영 리더	assignee 필드 입력	인시던트 종료 회의
진행 추적	assignee	status 필드 갱신	주간 1회
완료 검증	reviewer	status: done 머지	완료 보고 시점
분기 회고	분기 리뷰어	분기 보고서의 action-item 섹션	분기 마지막 주

액션 아이템이 별도 추출되지 않으면 인시던트 학습이 휘발됩니다. Postmortem 본문 안에 액션 아이템이 들어 있어도 다음 분기 회고 시점에 그 본문을 다시 펴 보는 사람이 없으면, 학습은 기록되었지만 실행되지 않은 상태로 남습니다 [S28][S57]. 그러므로 액션 아이템 추출은 *Postmortem 작성 직후의 1차 작업* 이어야 하며, 추출 자체를 운영자가 잊지 않도록 GitHub Action 으로 자동화하는 것이 권장됩니다.

여러분의 운영팀에서 지난 분기 액션 아이템의 완료율을 측정해 보십시오. 50% 미만이라면 액션 아이템이 표류하는 구조가 박혀 있는 상태이며, 별도 파일 추출과 주간 추적 자동화가 즉각 도입 가치 높은 항목입니다 [S28].

2.6 Local LLM RAG 소스 — 같은 책장을 AI 가 읽게 하는 마지막 고리

위 다섯 운영 영역(OPS Diary·Field Manual·Work Reports·Preventive Inspection·Incidents) 을 한 자리에 묶어주는 결합 영역이 Local LLM RAG 소스입니다. 다섯 영역이 합쳐진 디렉터리 자체가 Local LLM 의 RAG 소스가 되는 메커니즘이 이 영역의 핵심입니다 [S33][S39]. 4·5장에서 다룰 표준 스택과 인덱싱 파이프라인의 진입점이 되는 절이며, 다음 절 2.7 ADR 이 이 RAG corpus 의 의사결정 기록 layer 를 담당합니다.

2.6.1 인덱서가 보는 corpus = 위 5개 영역 전부

RAG 인덱서가 OPS Diary, Field Manual, Work Reports, Preventive Inspection, Incidents 다섯 영역을 모두 단일 corpus 로 본다는 원칙이 이 항의 1번 규칙입니다 [S33][S39]. 별도 변환·업로드 없이 git pull 한

번이면 인덱서가 5 영역의 모든 MD 파일을 직접 읽습니다. 이 구조가 OpsKnow Repo 와 기존 RAG 도구의 본질 차이입니다.

기존 RAG 도구 (AnythingLLM·Open WebUI Knowledge·PrivateGPT) 의 "업로드 → 읽기" 모델은 두 단계로 갈라져 있습니다. 사람이 SSoT 에서 문서를 export → RAG 도구에 업로드 → 도구가 자체 DB 에 임베딩 저장. 이 두 단계에서 SSoT 와 RAG 인덱스가 분리되며, SSoT 변경이 즉시 RAG 에 반영되지 않습니다 [S33][S38]. 결과적으로 운영자가 런북을 갱신해도 RAG 답변에는 옛 런북이 잡힙니다.

OpsKnow Repo 는 corpus 가 *리포지토리 자체* 이므로 SSoT 와 RAG 인덱스의 분리가 일어나지 않습니다. 운영자가 PR 을 머지하면 RAG 인덱서가 git pull 을 받아 변경된 파일만 재임베딩합니다. 이것이 "별도 업로드 단계 없음" 의 운영적 의미입니다 [S39][S33]. 인덱서가 보는 corpus 정의 한 줄이 OpsKnow Repo 의 정체성을 압축한다고 봐도 무방합니다.

여러분의 현재 RAG 도구가 SSoT 와 단일 corpus 로 합쳐지는지 점검해 보십시오. 별도 업로드 단계가 끼어 있다면 SSoT 와 RAG 인덱스의 분리가 일어나고 있으며, 런북 갱신 후 RAG 답변에 반영되기까지의 지연이 5분 이상이라면 운영자의 신뢰가 떨어진 상태일 가능성이 높습니다.

2.6.2 frontmatter `rag_visible: true|false` 로 노출 통제

corpus 가 리포지토리 자체라면 민감 정보 또는 raw 덤프의 노출 통제가 필요합니다. 그 통제의 1차 메커니즘이 frontmatter 의 `rag_visible` 필드입니다 [S30][S41]. `rag_visible: false` 가 박힌 파일은 인덱서가 임베딩 단계에서 제외하며, RAG 답변의 컨텍스트로 절대 주입되지 않습니다. 이 필드와 `sensitivity` (`low|medium|high`) 필드의 결합 규칙이 4 셀 매트릭스로 표현됩니다.

sensitivity \ rag_visible	true	false
low	RAG 노출, 인용 허용	(드문 케이스) 정책 검토 필요
medium	RAG 노출, 인용 시 출처 명시	RAG 제외, grep 만 가능
high	(정책 위반) 자동 차단	RAG 제외, 감사 로그만

`rag_visible: false` 의 기본값 정책을 보수적으로 설정할지 적극적으로 설정할지는 운영팀의 PII 마스킹 자동화 수준에 따라 결정됩니다 [S30]. 보수적 정책은 모든 신규 파일이 `rag_visible: false` 로 들어오고 reviewer 가 명시적으로 `true` 로 승인한 파일만 RAG 에 노출됩니다. 적극적 정책은 모든 신규 파일이 `rag_visible: true` 로 들어오되 `sensitivity: high` 가 박힌 파일은 자동으로 RAG 에서 제외됩니다.

`sensitivity: high` 필터가 강제되지 않으면 AI 가 민감정보를 답변에 인용할 위험이 있습니다 [S30][S41]. 예를 들어 인시던트 raw 덤프에 고객 토큰·내부 IP·인적 발언이 섞여 있는 상태에서 인덱서가 이 파일을 RAG corpus 에 포함하면, 다음 인시던트에서 운영자가 "최근 결제 API 장애 원인" 을 물었을 때 LLM 답변에 raw 덤프의 일부가 그대로 인용될 수 있습니다. 이는 컴플라이언스 위반으로 직결됩니다.

여러분의 운영팀에서 `rag_visible: false` 의 기본값 정책을 정의하십시오. 정책이 없는 상태에서는 신규 파일마다 기본 동작이 일관되지 않아 운영자가 매번 결정을 다시 합니다. 그 마찰이 누적되면 OPS Diary 작성 문화 자체가 표류할 위험이 있습니다 [S30].

2.6.3 git pull 트리거 증분 인덱싱

RAG 인덱서가 git pull 을 트리거로 받아 변경 파일만 재임베딩하는 패턴이 이 항의 마지막 주제입니다 [S39] [S42]. 전체 재인덱싱은 corpus 가 클수록 비용이 폭증하므로, 증분 인덱싱이 운영의 표준입니다. 트리거 주기는 두 가지 옵션이 있습니다. 즉시 트리거 (GitHub webhook 또는 GitLab CI 가 인덱서를 호출) 와 시간 단위 트리거 (5분 또는 15분마다 cron 으로 git pull) 입니다.

트리거 방식	최신 반영 지연	운영 부담	인덱서 부하	권장 시나리오
GitHub webhook 즉시	< 1분	중간 (webhook 설정)	변경 빈도에 비례	다중 팀, 잦은 PR
cron 5분 주기	최대 5분 지연	낮음 (crontab 1줄)	일정 간격 균등	단일 팀, 안정적 환경
cron 15분 주기	최대 15분 지연	낮음	균등	소규모, 변경 빈도 낮음

증분 인덱싱의 비용 절감 효과는 corpus 가 커질수록 비선형으로 늘어납니다 [S39]. 전체 재인덱싱은 1만 파일 corpus 기준 임베딩 GPU 30~60분이 소요되는 반면, 증분 인덱싱은 변경된 5~20 파일만 재임베딩하므로 10초 이내 완료됩니다. 이 차이가 인덱서를 24/7 운영할 때의 GPU 비용과 직결됩니다.

인덱싱 방식	1만 파일 corpus 기준	변경 5 파일 기준	GPU 시간 (일 1회 기준)
전체 재인덱싱	30~60분	30~60분	30~60분
증분 인덱싱	(초기 1회만)	10초 이내	(변경 빈도에 비례)

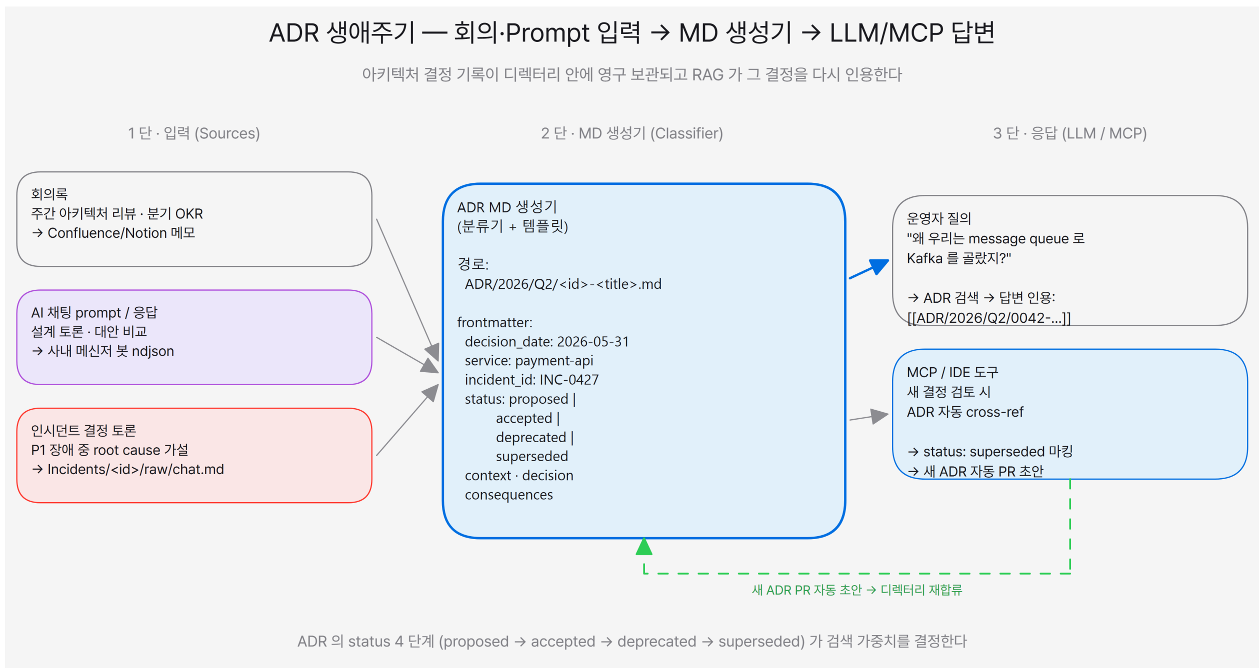
증분 인덱싱의 트리거 주기는 도입 의사결정 항목 한 가지입니다. 최신 반영이 1분 이내로 요구되는 인시던트 대응 환경에서는 webhook 즉시 트리거를 선택하고, 일반 운영 환경에서는 cron 5분 또는 15분 주기로 충분합니다 [S42]. 5장에서 자세히 다룰 파이프라인은 이 트리거 단을 첫 단계로 받습니다.

여러분의 운영팀이 RAG 인덱서를 도입할 때 증분 인덱싱의 트리거 주기 결정이 첫 번째 의사결정입니다. 그 결정 한 가지가 인덱서의 운영 부담과 GPU 비용 양쪽을 동시에 결정합니다.

2.7 ADR — 아키텍처 결정 기록의 자동화

일곱 영역의 마지막 영역이자, IT 운영팀이 매일 내리는 아키텍처 의사결정을 잃지 않고 자동으로 기록하는 영역이 ADR (Architecture Decision Record, 아키텍처 결정 기록) 입니다. 운영팀은 매일 크고 작은 결정을 내립니다 — DB 커넥션 풀 크기 조정, 캐시 TTL 변경, 신규 미들웨어 도입, 인증 방식 교체 등. 그 결정의 "왜" 가 회의록·사내 메신저·머릿속에 흩어져 6개월 뒤에는 누구도 재구성할 수 없게 되는 문제는 1.1 절에서 다룬 세 가지 사각지대의 마지막 모습입니다. ADR 영역은 이 결정 기록을 디렉터리·MD·LLM 답변이라는 세 가지 도구로 자동 보존합니다.

[FIGURE: adr-lifecycle] **캡션**: ADR 의 회의·Prompt-응답 적재 → MD 자동 생성 → LLM/MCP 답변의 3단 흐름 **의도**: 좌측 입력 (회의록·AI 채팅 prompt/응답·인시던트 결정 토론) → 중앙 MD 생성기 (일자별/시스템별/장애별 분류) → 우측 LLM/MCP 답변 (운영자 질의 → ADR 검색 → 답변 인용) 의 3단 수평 흐름. 각 단계의 산출물 경로 (`ADR/2026/Q2/<id>-<title>.md`) 와 frontmatter 핵심 필드 (`decision_date` , `service` , `incident_id` , `status: proposed|accepted|deprecated|superseded`) 를 함께 표기.



2.7.1 회의록·Prompt·응답을 ADR 로 적재

운영팀의 아키텍처 의사결정은 세 곳에서 나옵니다 — 정기 설계 회의, AI 와의 prompt-응답 토론, 인시던트 대응 중 즉석 결정. 세 곳의 산출물을 ADR 영역으로 자동 적재하는 메커니즘이 이 항의 주제입니다. 회의록은 사내 메신저 봇 또는 회의 노트 도구가 회의 종료 직후 ADR 초안 PR 을 자동 생성합니다. AI 와의 prompt-응답은 OPS Diary 적재 패턴과 동일하게 ADR 로 분류 가능한 질문 (예: "X 와 Y 중 무엇을 선택해야 하는가") 을 자동 인식하여 ADR 디렉터리로 분기 적재합니다. 인시던트 중 결정은 Postmortem 머지 시 액션 아이템과 함께 ADR 1건이 자동 추출됩니다 [S23][S26].

ADR 1건의 표준 구조는 다음 5 섹션으로 한정됩니다. (1) Context — 결정이 필요했던 배경, (2) Decision — 채택한 결정 한 줄, (3) Alternatives — 검토한 대안들, (4) Consequences — 결정의 예상 결과·위험, (5) Sources — 회의록·prompt 응답·인시던트 ID 등 원본 출처 링크. 이 5 섹션이 frontmatter 와 함께 박혀 있으면 6개월 뒤 같은 결정의 "왜" 를 재구성하는 비용이 5초로 줄어듭니다. 결정 기록이 없는 환경에서는 같은 재구성에 반나절 이상이 소요됩니다 [S52].

2.7.2 LLM·MCP 를 통한 ADR 답변

ADR 영역이 RAG corpus 의 일부가 되면 운영자가 자연어로 과거 결정의 "왜" 를 질의할 수 있습니다. 예를 들어 "결제 API 의 DB 커넥션 풀 크기를 50 으로 결정한 배경" 을 물으면, RAG 가 `ADR/2026/Q1/0042-payment-db-pool-size.md` 를 검색해 Context·Decision·Alternatives 섹션을 인용한 답변을 생성합니다 [S33]. MCP (Model Context Protocol, 모델 컨텍스트 프로토콜 — LLM 에 외부 데이터 소스를 표준 인터페이스로 연결하는 프로토콜) 를 통해 IDE·채팅 UI·CLI 등 다중 진입점에서 동일한 ADR 답변에 접근할 수 있어, 결정 기록의 활용 표면이 IDE 코드 작업·채팅 질의응답·터미널 운영 작업 세 갈래로 확장됩니다.

ADR 답변에 인용된 원본 ADR 파일은 frontmatter 의 `status` 필드를 통해 그 결정이 현재도 유효한지를 즉시 표시합니다. `status: accepted` 는 현재 유효한 결정, `status: deprecated` 는 더 이상 권장되지 않는 결정, `status: superseded` 는 새 ADR 로 대체된 결정입니다 [S62]. RAG 가 답변에 인용할 때 superseded 결정에는 후

속 ADR 의 링크가 자동으로 함께 표시되도록 정책을 두면, 운영자가 옛 결정을 따라 잘못된 절차를 시도하는 회귀가 차단됩니다.

2.7.3 ADR 디렉터리의 분류 — 일자별·시스템별·장애별

ADR 의 디렉터리 분류는 세 가지 축을 동시에 만족해야 합니다. 일자별 정렬 (`ADR/2026/Q2/`), 시스템별 정렬 (서비스 메타 `service: payment-api`), 장애별 연결 (인시던트 메타 `incident_id: INC-2026-031`) 입니다. 세 분류를 모두 디렉터리 트리로 표현하면 중복이 발생하므로, OpsKnow Repo 의 권장 패턴은 디렉터리는 일자별 1차 정렬을 따르고 시스템·장애 정보는 frontmatter 메타로 박는 hybrid 구조입니다 [S19].

ADR frontmatter 의 표준 7 필드는 다음과 같이 박힙니다 — `adr_id` (예: `ADR-2026-042`), `title` , `decision_date` (ISO 8601), `service` (영향받는 서비스 목록), `incident_id` (관련 인시던트 ID, optional), `status` (proposed|accepted|deprecated|superseded), `supersedes` (대체된 이전 ADR ID, optional). 이 7 필드가 메타에 박혀 있으면 RAG 가 "결제 서비스 관련 ADR 만" 또는 "지난 분기 인시던트에서 도출된 ADR 만" 같은 메타필터를 즉시 만들 수 있습니다.

분류 기준	디렉터리 또는 메타	검색 시나리오	자동화 트리거
일자별	`ADR/<year>/<quarter>/` 디렉터리	분기 회고 시 전체 결정 목록	신규 PR 시 자동 폴더 배치
시스템별	frontmatter `service` 메타	"결제 API 관련 결정 5건"	CODEOWNERS 매핑으로 reviewer 라우팅
장애별	frontmatter `incident_id` 메타	"INC-2026-031 에서 도출된 결정"	Postmortem 머지 시 자동 추출
상태별	frontmatter `status` 메타	"현재 유효한 결정만"	`status: deprecated` 자동 전이

ADR 영역이 7개 영역의 마지막 자리에 들어가는 이유는 이 영역이 다른 6 영역의 결정 기록을 모두 흡수하기 때문입니다. OPS Diary 의 일일 질의응답에서 도출된 결정, Field Manual 의 SOP 변경 결정, 작업 보고서의 분기 우선순위 결정, 예방점검의 룰 변경 결정, 인시던트 Postmortem 의 후속 액션 결정 — 모두 ADR 영역에 1:1 로 연결됩니다 [S52]. 다른 6 영역의 어디에 어떤 결정이 박혔는지를 한 곳에서 추적하는 메타 영역으로 ADR 을 두면, 운영팀의 의사결정 누락이 사실상 0 으로 떨어집니다.

여러분의 운영팀이 지난 분기에 내린 아키텍처 결정 중 6개월 뒤에 "왜 그렇게 했더라" 를 즉시 답할 수 있는 비율을 점검해 보시기 바랍니다. 그 비율이 50% 미만이라면 ADR 영역의 도입이 즉각 ROI 가 높은 항목이며, 도입 비용은 표준 5 섹션 템플릿 1장 + 자동 적재 스크립트 반나절이면 충분합니다. 그리고 그 결정이 다음 장 (3장 설계 원칙) 에서 다룰 MD·Git·사람-AI 공동편집의 거버넌스와 함께 OpsKnow Repo 의 운영 토대를 완성합니다.

3장. 설계 원칙 — MD, Git, 사람-AI 공동편집

OpsKnow Repo 는 어떤 도구를 쓰느냐의 문제가 아니라 어떤 약속 위에 운영 지식을 쌓느냐의 문제입니다. 1장이 흩어진 운영 지식의 비용을 짚고 2장이 일곱 영역의 디렉터리 구조를 보였다면, 3장은 그 디렉터리가 어떤

설계 원칙 위에 서 있는지를 정리합니다. 다섯 가지 원칙은 한 줄로 묶입니다 — MD 가 표준 단위고, 디렉터리·파일명·YAML frontmatter 가 운영 ontology 를 표현하며, PR 게이트가 사람·AI 공동편집의 신뢰 기반이고, Diátaxis 가 문서 종류를 정렬하며, 유효성 관리가 런북이 거짓말이 되는 것을 막습니다.

이 다섯 원칙은 따로 떼면 익숙한 모범 사례이지만, 한 디렉터리 위에 동시에 모이면 기존 위키·인시던트 SaaS·RAG 도구가 함께 풀지 못한 공백이 채워집니다. SSoT 가 한 디렉터리에 모인다는 의미는 단지 파일을 한 곳에 두는 것이 아니라, 사람이 본 런북과 AI 가 본 컨텍스트가 같은 줄을 가리킨다는 뜻입니다. 운영 ontology 가 frontmatter 에 흡수된다는 의미는 단지 메타가 풍부해진다는 것이 아니라, RAG 가 메타필터로 검색을 좁힐 수 있다는 뜻입니다 [S55].

운영팀이신 여러분께서 3장을 읽으실 때 가장 먼저 던지셔야 할 질문은 "우리 팀의 운영 문서에 `author` · `reviewer` · `confidence` frontmatter 3종이 박혀 있는가" 입니다. 이 세 필드가 없으면 AI 가 작성한 런북과 사람이 작성한 런북을 구분할 수 없고, 구분할 수 없으면 AI 환각 1건이 운영 매뉴얼 전체를 오염시킬 수 있습니다 [S61]. 반대로 세 필드가 박혀 있고 PR 게이트가 코드에 박혀 있으면, AI 가 매일 100건의 초안을 만들어도 사람이 승인한 N건만 RAG corpus 에 들어옵니다.

3장의 다섯 절은 각각 다른 시간 축을 다룹니다. 3.1 은 파일 포맷 선택이라는 한 번의 결정, 3.2 는 디렉터리·파일명·메타라는 도입 첫 달의 합의, 3.3 은 매일 흐르는 PR 트랜잭션, 3.4 는 문서 종류 분리라는 분기 단위의 거버넌스, 3.5 는 6개월~1년 단위의 유효성 점검 사이클입니다. 다섯 절이 모두 갖춰져야 운영 지식이 흩어짐 없이 누적됩니다 — 하나라도 빠지면 1장에서 진단한 세 가지 사각지대(사내 메신저·티켓·머릿속) 중 하나로 다시 흘러들어 갑니다.

3.1 왜 MD 가 중요한가

운영 지식의 1차 저장 포맷 결정은 도구 선택보다 먼저 와야 하는 결정입니다. 도구는 1~2년에 한 번 바뀌지만 포맷은 5~10년을 갑니다. 이 절은 MD 를 표준 단위로 선택해야 하는 네 가지 근거 — 사람·AI 공동 가독성, RAG 친화, Git 친화, 휴대성 — 를 차례로 정리합니다. 네 근거는 독립적이 아니라 서로 강화하는 관계이며, 한 가지만 따로 떼면 다른 포맷(Confluence Storage Format·Notion 블록 DB)이 더 나아 보이는 영역도 있습니다. 네 가지를 함께 보면 MD 만이 모든 축에서 안전한 선택임이 드러납니다 [S17][S54].

3.1.1 사람-AI 공동 가독성 — 사람의 마크업이자 LLM 의 손실 없는 포맷

MD 가 사람을 위한 가벼운 마크업인 이유 Markdown 은 John Gruber 가 2004 년 "사람이 쓰기 쉽고 사람이 읽기 쉬운 텍스트 포맷" 을 목표로 설계한 마크업입니다. 헤딩은 `#` 으로, 강조는 `*` 으로, 링크는 `[텍스트](URL)` 로 표현되어 원시 파일을 열어도 의미가 즉시 읽힙니다. HTML 이나 XHTML 처럼 태그가 본문 분량의 절반을 차지하지 않으며, 워드 프로세서처럼 바이너리도 아닙니다. 운영팀이신 여러분께서 인시던트 한가운데 `vim` 또는 `nano` 로 런북을 열어야 하는 상황을 상상해 보십시오 — MD 는 그 상황에서도 가독성을 잃지 않는 거의 유일한 포맷입니다 [S3].

LLM 토크나이저에 손실 없이 입력되는 이유 대규모 언어 모델은 입력 텍스트를 토큰으로 쪼개서 처리합니다. MD 의 마크업 기호(`#`, `*`, `>`, `` ` ``)는 대부분의 토크나이저에서 1~2 토큰으로 처리되어 원본 의미를 거의 그대로 유지합니다. 반면 Confluence Storage Format 은 XHTML 기반이라 `<ac:structured-macro ac:name="info">` 같은 태그가 한 줄에 수십 토큰을 소비합니다 [S17]. 동일한 운영 런북 한 페이지를 비교하면, MD 로 작성된 본문은 약 1,200 토큰이지만 Confluence Storage Format 으로 변환하면 같은 본문이 3,000~4,000 토

큰을 소비합니다. RAG 의 context window 가 32K 토큰이라면 MD 는 26개 런북을 한 번에 참조할 수 있지만 Confluence Storage Format 은 8~10개에 그칩니다 [S54].

한 포맷이 두 독자를 모두 1차 객체로 처리해야 하는 이유 운영 지식의 1차 독자는 더 이상 사람만이 아닙니다. 사람 운영자가 런북을 읽고 동시에 AI Agent 가 같은 런북을 RAG 로 검색하는 상황이 일상이 되었습니다. 만약 한 포맷이 사람에게만 친화적이거나 AI 에게만 친화적이라면, 사람과 AI 가 본 정보가 어긋날 위험이 커집니다 [S30]. MD 는 사람의 마크업이자 LLM 의 손실 없는 포맷이라는 두 성질을 동시에 갖춘 거의 유일한 포맷이며, 그래서 1차 저장 포맷의 의사결정에서 다른 후보를 압도합니다.

포맷	사람 가독성	LLM 토큰 효율	diff 가독성	외부 도구 휴대성
MD + YAML frontmatter	최상 (raw 도 즉시 읽힘)	최상 (마크업 1~2 토큰)	최상 (line 단위)	최상 (.md 표준)
Confluence Storage Format	낮음 (XHTML)	낮음 (태그 30~50%)	낮음 (XML 전체 diff)	낮음 (XHTML export)
Notion 블록 DB	중 (UI 한정)	중 (JSON 블록)	낮음 (JSON diff)	낮음 (round-trip 손실)

3.1.2 RAG 친화 — H1~H6, 리스트, 코드블록이 자연스러운 chunk 경계

MD 헤딩이 의미 단위 청크 경계를 만드는 이유 RAG 의 품질은 청크 분할 품질에 비례합니다. 청크가 의미 단위로 잘리면 검색 정확도가 높고, 고정 길이로 잘리면 한 문장이 둘로 갈라져 의미가 깨집니다. MD 의 H2/H3 헤딩은 작성자가 명시적으로 박은 의미 단위 경계입니다 — `## 인시던트 대응 절차` 와 `## 사후 점검 항목` 사이는 작성자가 의도적으로 분리한 두 주제입니다. RAG 인덱서가 이 경계를 그대로 청크로 삼으면 별도 휴리스틱 없이 의미 단위 검색이 됩니다 [S35][S55].

리스트·코드블록도 자연스러운 청크 후보가 되는 이유 MD 의 리스트(`-` 또는 `1.`) 와 코드블록(` ``` `) 도 그 자체로 의미 단위입니다. 점검 항목 8개가 리스트로 나열된 런북이라면, 인덱서는 리스트 전체를 한 청크로 두거나 항목당 하나씩 쪼갤 수 있습니다. 코드블록도 마찬가지로 명령어 한 덩어리를 한 청크로 보존할 수 있습니다 [S33]. Confluence Storage Format 이나 Notion 블록 DB 는 이 정보를 LLM 토크나이저에 전달하기 위해 별도 변환 단계가 필요하지만, MD 는 그대로 임베딩 모델에 들어갑니다 — 변환 비용 0 입니다.

청크 경계 품질이 모델 교체보다 큰 효과를 내는 이유 RAG 도입 후 검색 정확도가 낮을 때 운영팀이 가장 먼저 시도하는 것이 임베딩 모델 교체입니다. nomic-embed-text 에서 BGE-M3 으로, 다시 더 큰 모델로 교체하면 정확도가 10~20% 오르긴 합니다. 그러나 청크 경계 자체가 의미를 깨고 있으면 어떤 모델로 바꿔도 한계에 부딪힙니다 [S55]. 사전 조사 자료의 r/LocalLLaMA 합의에 따르면 청크 분할 정확도가 검색 품질의 70% 를 결정하고 모델 선택이 나머지 30% 를 결정합니다. MD 헤딩 기반 청크 분할은 이 70% 를 자동으로 확보합니다 — 작성자가 H2 를 의미 단위로 박는 한.

청크 분할 방식	의미 단위 경계	변환 비용	인덱싱 속도
MD H2 단위 (권장)	명시적	0 (raw MD 직접)	빠름 (1 만 페이지 < 10분)
고정 길이 512 토큰	무시	0	빠름
Recursive Character	부분적	낮음	중

청크 분할 방식	의미 단위 경계	변환 비용	인덱싱 속도
Confluence Storage Format export	XHTML 파싱 필요	높음	느림

3.1.3 Git/Drive 친화 — diff · blame · PR 리뷰가 그대로 적용됩니다

MD 가 line-based diff 와 자연스럽게 결합하는 이유 Git 의 diff 는 line-based 입니다 — 어느 줄이 추가·삭제·수정되었는지를 줄 단위로 추적합니다. MD 는 본질적으로 line-based 텍스트라 변경 한 줄이 diff 한 줄로 깨끗하게 잡힙니다 [S19]. 같은 런북의 임계값 한 줄(예: `CPU 임계값: 80% → 85%`)이 바뀌었을 때 Git diff 는 정확히 그 한 줄만 보여줍니다. 반대로 Confluence 의 페이지 히스토리는 XHTML 통째로 비교하므로, 한 줄의 변경이 수십 줄의 태그 변화로 부풀려져 reviewer 가 본질을 찾기 어렵습니다.

blame 이 운영 결정 추적의 1차 근거가 되는 이유 Git blame 은 파일의 각 줄이 누구에 의해 언제 어떤 커밋으로 작성되었는지를 알려줍니다. 운영 런북의 특정 임계값 한 줄을 두고 "왜 이 값이지" 라는 질문이 6개월 후에 나왔을 때, blame 한 번이면 그 결정의 작성자·일자·커밋 메시지(즉 결정 사유)에 즉시 도달합니다 [S19]. GitLab Handbook 의 공개 runbooks 디렉터리는 이 원리를 그대로 활용하여 모든 결정이 1년 후에도 blame 으로 추적 가능합니다 [S53]. SaaS 위키의 audit log 는 페이지 단위 수정자만 기록하므로, 한 페이지 안의 특정 줄을 누가 바꿨는지는 시간순으로 모든 revision 을 뒤져야 합니다.

PR 리뷰가 운영 문서의 사람 승인 게이트가 되는 이유 PR(Pull Request) 은 코드 협업의 표준 패턴이지만, MD 운영 문서에도 그대로 적용됩니다. 누가 작성하고 누가 reviewer 인지가 PR 메타에 박히고, line-by-line 코멘트로 의견 교환이 가능하며, 승인 후에만 main 브랜치에 머지됩니다 [S19]. 이 흐름이 운영 문서에 적용되면 변경 감사가 Git 의 무료 부산물이 됩니다 — 별도 audit 시스템 구축 비용이 0 입니다. 운영팀이신 여러분께서 ISMS-P 또는 금감원 감사를 받으셔야 한다면, Git 의 commit 이력·PR 머지 이력·CODEOWNERS 설정 세 가지가 변경 통제의 1차 근거가 됩니다 [S20].

항목	Git + MD	SaaS 위키 audit log
줄 단위 변경 추적	blame 1 명령	페이지 revision 전수 검사
변경 사유 기록	commit message + PR 본문	revision comment (선택)
승인자 기록	PR reviewer 메타 + CODEOWNERS	(대부분 없음)
외부 감사 응대	git log export	API 수출 후 가공
도구 교체 후 보존	100% (`.git` 통째)	부분 (export 손실)

3.1.4 Vendor lock-in 회피 — `.md` + `assets/` 만 있으면 어느 도구로도 휴대 가능

도구의 수명이 콘텐츠의 수명보다 짧다는 진실 운영 런북·SOP·Postmortem 은 10~20년을 가는 자산입니다. 반면 SaaS 도구의 평균 수명은 5~7년에 불과합니다 — 인수, 가격 정책 변경, 서비스 종료로 매년 한두 개의 위키 도구가 사라지거나 변합니다. 운영팀이신 여러분께서 5년 전 Confluence DC 에서 Cloud 로 이주하셨거나, Notion 에서 다른 도구로 export 를 시도하셨다면 round-trip 손실의 비용을 이미 경험하셨을 것입니다

[S17][S32]. MD 는 표준 텍스트 포맷이므로 도구가 사라져도 콘텐츠가 그대로 남습니다 — `.md` 파일과 `assets/` 디렉터리만 있으면 어떤 에디터에서도 즉시 열립니다.

휴대성이 단순한 export 가능성과 다른 이유 SaaS 위키 대부분은 export 를 지원한다고 광고하지만, 실제로 export 한 결과물이 raw MD 와 동등한 가독성을 갖는 경우는 드뭅니다. Notion 의 MD export 는 데이터베이스 뷰·동기화 블록·임베드를 보존하지 못해 round-trip 시 약 30~40% 의 정보가 손실됩니다 [S32]. Confluence 의 MD export 는 매크로·인포 박스·표 일부가 깨지며, BookStack 의 MD export 는 첨부 이미지의 상대 경로가 절대 경로로 바뀌어 다른 도구에서 깨집니다 [S12]. OpsKnow Repo 의 1차 저장 포맷이 MD 라는 것은 export 가 곧 raw 라는 뜻이며, 도구 교체 시 자료를 옮기는 비용이 0 에 수렴한다는 뜻입니다.

도구 교체 안전판이 도입 의사결정의 보험이 되는 이유 IT 의사결정자가 새 운영 지식 저장소를 도입할 때 가장 큰 망설임은 "5년 후에 이 도구가 사라지면 어떻게 되는가" 입니다. SaaS 위키는 이 질문에 대한 답을 갖고 있지 못합니다 — export 결과물이 자체 데이터베이스 형식이거나 round-trip 손실이 크기 때문입니다 [S11]. 반면 OpsKnow Repo 는 처음부터 `.md` + `assets/` + `.git` 세 요소만으로 구성되어, OpsKnow 라는 도구가 사라져도 자산은 그대로 남습니다. 운영팀이신 여러분께서 5년 후 다른 도구로 이주하실 때 `git clone` 한 번이면 모든 콘텐츠와 이력이 따라옵니다.

위키 도구	export 포맷	round-trip 손실률	5년 후 자산 보존
OpsKnow Repo	자체 = MD raw	0%	100% (`.git` 통째)
Confluence	XHTML / MD (부분)	약 20~30%	부분 (Storage Format 의존)
Notion	MD / HTML / JSON	약 30~40%	부분 (DB 뷰 손실)
BookStack	MD / HTML / PDF	약 10~20%	양호 (첨부 경로 주의)

3.2 디렉터리 트리·파일명·frontmatter 표준 (운영 ontology)

운영 ontology 는 추상적 개념이 아니라 구체적인 디렉터리 트리·파일명 규칙·YAML frontmatter 8 필드로 표현됩니다. 이 절은 그 세 요소를 차례로 정의하고 부록 C 의 표준 스키마 v1 과 1:1 정합을 박습니다. 운영팀이신 여러분께서 도입 첫 달에 결정해야 할 합의 사항은 사실 거의 모두 이 절에 모여 있습니다 — 파일명 컨벤션, frontmatter 필드 enum 값, SRE Book·AWS W-A 어휘의 흡수 방식이 그 합의의 핵심입니다 [S6][S21][S22].

[FIGURE: frontmatter-schema] **캡션**: OpsKnow Repo 의 YAML frontmatter 8 필드 표준 v1 **의도**: 8개 필드(`owner` / `reviewer` / `service` / `severity` / `doc_type` / `freshness_until` / `confidence` / `sources`)를 카드 형태로 배치합니다. 각 카드에 enum 값 또는 형식 예시를 표기하고, 가운데에 "운영 ontology" 라벨을 둡니다. 카드 색상은 책임 영역별로 구분(소유·검토는 청색, 도메인은 녹색, 라이프사이클은 황색, 메타는 회색)하여 한눈에 책임 분할이 드러나도록 합니다.

3.2.1 파일명 컨벤션 — `YYYY-MM-DD-service-slug.md`

일자 prefix 가 자동 수집·시간 정렬의 기반이 되는 이유 파일명에 `YYYY-MM-DD` 일자가 들어가면 디렉터리를 ls 한 번 했을 때 시간순으로 정렬되며, 분기 회고 스크립트가 `grep '2026-Q2'` 한 번으로 분기 자료를 추출할 수 있습니다 [S6]. GitHub Action 또는 cron 스크립트가 만료 임박 런북을 찾을 때도 파일명 일자가 1차 단서가 됩니다. 운영팀이신 여러분께서 1년치 인시던트를 한 번에 보고 싶을 때 `ls Incidents/2026/Q*/` 한 줄이면 시간순 정렬된 목록을 즉시 얻습니다.

서비스 slug 가 검색·필터링·중복 방지의 기반이 되는 이유 파일명 뒤쪽의 `service-slug` 는 영문 소문자 + 하이픈만으로 구성된 서비스 식별자입니다(예: `payment-api`, `auth-service`, `redis-cluster`). 같은 일자에 두 서비스에서 동시에 인시던트가 발생해도 slug 가 다르면 파일이 중복되지 않으며, `ripgrep` 한 번이면 특정 서비스의 모든 이력을 시간순으로 모을 수 있습니다 [S19]. slug 는 frontmatter 의 `service` 필드와 1:1 매칭되어 메타 필터링과 파일명 검색이 서로를 보강합니다.

영역별 추가 prefix 가 디렉터리 깊이를 평탄화하는 이유 6 영역 중 OPS Diary 는 `2026-05-31.md` 처럼 일자만 두고, Field Manual 은 `payment-api/runbook-restart.md` 처럼 서비스 디렉터리 안에 두며, Incidents 는 `2026/Q2/2026-05-31-payment-api-503.md` 처럼 분기까지 박습니다. 영역별 컨벤션이 다르지만 일관된 원칙(시간 + 서비스 식별자)을 공유하므로, RAG 인덱서가 단일 휴리스틱으로 모든 영역의 파일명에서 메타를 추출할 수 있습니다 [S19].

영역	파일명 패턴	예시
OPS_Diary	`YYYY-MM-DD.md`	`2026-05-31.md`
Field_Manual	`<service-slug>/<topic>.md`	`payment-api/runbook-restart.md`

영역	파일명 패턴	예시
Work_Reports	`<period>/YYYY-MM-DD.md`	`weekly/2026-W22.md`
Preventive_Inspection	`<ops-id>/YYYY-MM-DD.md`	`ops-03/2026-Q2.md`
Incidents	`YYYY/QN/YYYY-MM-DD-service-slug/`	`2026/Q2/2026-05-31-payment-api-503/`
_meta	`INDEX.md` `glossary.md`	(고정 파일명)

3.2.2 frontmatter 필드 8종의 enum 값 (부록 C 와 1:1 정합)

owner / reviewer — 소유와 검토 책임의 분리 `owner` 필드는 이 문서의 1차 책임자이며 만료 알림·재검토 알림의 수신자입니다. `reviewer` 는 변경 PR 시 자동 할당되는 검토자이며 CODEOWNERS 설정과 동기화됩니다 [S60][S61]. 두 필드를 분리하는 이유는 소유와 검토의 책임이 다르기 때문입니다 — 소유자가 휴가 중이어도 reviewer 가 PR 을 승인할 수 있어야 합니다. 두 필드의 값은 사람 식별자(예: `kim.yj`) 또는 팀 식별자(예: `team-payment`)이며 자유 텍스트가 아닌 enum 처럼 운용됩니다.

service / severity — 도메인 메타필터의 1차 키 `service` 는 이 문서가 다루는 서비스 식별자이며, 운영 도메인 메타필터의 1차 키입니다. RAG 검색 시 "payment-api 관련 런북만" 같은 필터링이 이 필드 한 줄로 가능합니다 [S35]. `severity` 는 인시던트 또는 점검의 심각도이며 SRE Book 의 표준 4단계(P1/P2/P3/P4)를 따릅니다 [S21]. 두 필드의 enum 값은 `_meta/services.yaml` 과 `_meta/severity.yaml` 두 파일에 박혀 있어 새 서비스 추가 시 한 곳만 갱신하면 됩니다.

doc_type — Diátaxis 4분면의 표현 `doc_type` 필드는 Diátaxis 4분면 중 하나의 값을 갖습니다 — `tutorial` / `howto` / `reference` / `explanation` [S62]. 이 필드가 박혀 있으면 RAG 가 답변할 때 질문 유형에 맞는 문서 종류를 우선 검색할 수 있습니다. "어떻게 재시작하지" 같은 절차 질의에는 `howto` 만 검색하고, "이 시스템은 왜 이렇게 설계됐지" 같은 배경 질의에는 `explanation` 만 검색하는 식입니다. 3.4 절에서 Diátaxis 4분면을 자세히 다룹니다.

freshness_until / confidence / sources — 라이프사이클과 신뢰성 메타 `freshness_until` 은 이 문서의 유효 만료일(ISO 8601 형식, 예: `2026-12-31`)이며 3.5 절에서 자세히 다룹니다 [S6]. `confidence` 는 AI 가 생성한 문서의 신뢰도 등급(`low` / `medium` / `high`)이며 reviewer 자동 할당 정책과 연동됩니다 [S47]. `sources` 는 이 문서가 참조한 출처 목록(URL 또는 파일 경로)이며 RAG 답변 시 인용 가능한 1차 근거가 됩니다. 세 필드는 함께 작동하여 "오래된 / 저신뢰 / 출처 부재" 의 위험을 메타 차원에서 가시화합니다.

필드	형식	enum 또는 예시	책임 영역
`owner`	string	`kim.yj` 또는 `team-payment`	소유
`reviewer`	string	`lee.jm` 또는 `team-sre`	검토
`service`	string	`payment-api` (`_meta/services.yaml` 참조)	도메인
`severity`	enum	`P1` / `P2` / `P3` / `P4`	도메인

필드	형식	enum 또는 예시	책임 영역
doc_type	enum	tutorial / howto / reference / explanation	도메인
freshness_until	ISO 8601 date	2026-12-31	라이프사이클
confidence	enum	low / medium / high	신뢰성
sources	list[string]	URL 또는 파일 경로	신뢰성

3.2.3 SRE Book · W-A OPS 1~11 의 어휘를 어떻게 흡수하는가

SRE Book 의 SLO·SLI·오류예산 어휘가 frontmatter 로 들어오는 경로 Google SRE Book Ch.4 의 SLO(Service Level Objective)·SLI(Service Level Indicator)·오류예산(error budget) 은 운영 도메인의 사실상 표준 어휘입니다 [S21]. OpsKnow Repo 는 이 어휘를 새로 만들지 않고 그대로 흡수합니다 — service 필드의 값은 SRE Book 의 서비스 단위와 일치하고, severity 필드의 P1~P4 는 SRE Book 의 incident severity 와 일치합니다. 운영팀이신 여러분께서 SRE Book 을 이미 읽으셨다면 OpsKnow Repo 의 frontmatter 8 필드 중 5 필드는 즉시 이해되실 것입니다.

AWS W-A OPS 1~11 이 점검 카탈로그로 들어오는 경로 AWS Well-Architected Framework 의 Operational Excellence Pillar 는 OPS 1~11 의 11개 모범 사례 질문으로 구성됩니다 [S22]. 예를 들면 OPS 4 는 "운영 상태를 어떻게 가시화하는가", OPS 7 은 "운영 준비 상태를 어떻게 보장하는가" 같은 식입니다. OpsKnow Repo 의 Preventive_Inspection 영역(2.4 절)이 이 11개 항목을 점검 카탈로그로 흡수하여 ops-id: ops-04 같은 메타 태그를 frontmatter 에 박습니다. 새 점검 항목을 만들 때 처음부터 자체 정의하지 않고 검증된 카탈로그를 차용하므로 누락이 줄어듭니다.

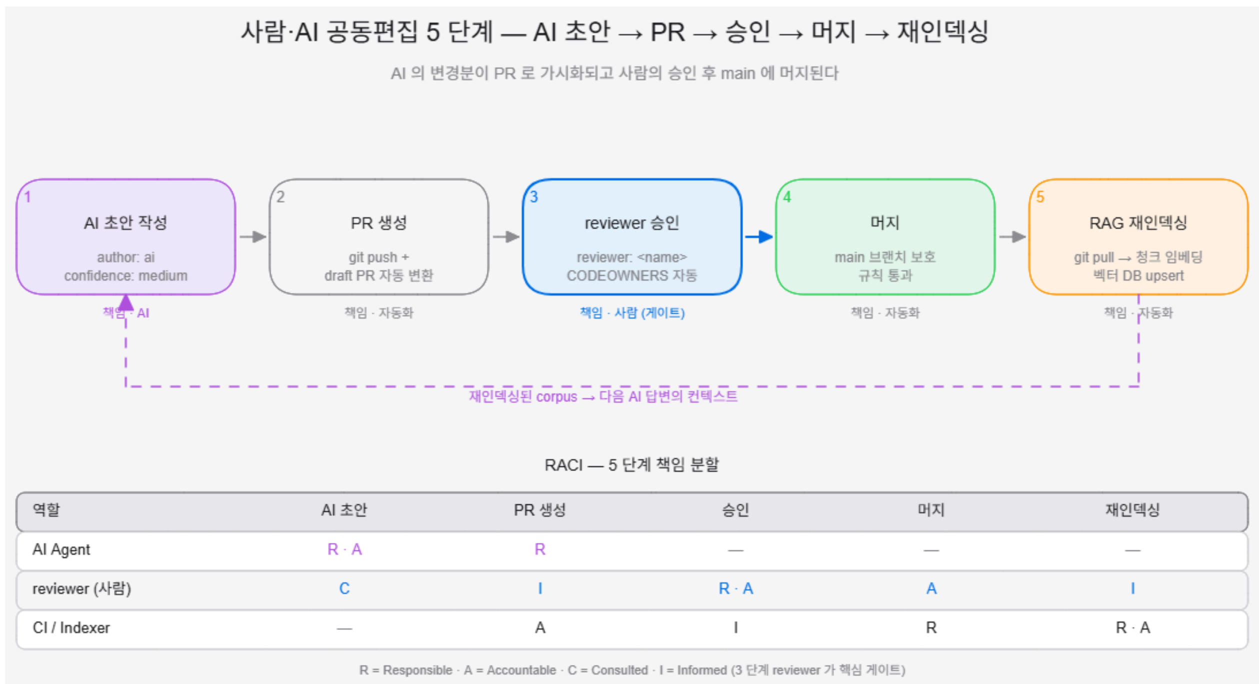
Diátaxis 의 4분면이 doc_type 으로 들어오는 경로 Daniele Procida 의 Diátaxis 프레임워크는 문서를 Tutorial / How-to / Reference / Explanation 4분면으로 분리하라는 권고입니다 [S62]. 이 4분면이 OpsKnow Repo 의 doc_type enum 으로 그대로 들어옵니다. 사전 조사 자료에 따르면 문서 종류 분리는 AI 오용 감소에도 직접 기여합니다 — RAG 가 "어떻게" 질문에는 How-to 만 검색하고 "왜" 질문에는 Explanation 만 검색하면, 답변에 학습용 설명과 절차가 뒤섞이는 회귀가 줄어듭니다.

외부 어휘	출처	frontmatter 흡수 위치	예시
SLO·SLI·오류예산	SRE Book Ch.4 [S21]	service + _meta/slo.yaml	service: payment-api
Incident severity P1~P4	SRE Book Ch.14 [S21]	severity enum	severity: P2
OPS 1~11	AWS W-A [S22]	ops_id 보조 메타	ops_id: ops-04
Tutorial/How-to/Reference/Explanation	Diátaxis [S62]	doc_type enum	doc_type: howto
author / reviewer 책임 분리	GitLab Handbook [S19]	owner + reviewer	(위 표 참조)

3.3 사람-AI 공동 편집 트랜잭션 — PR 게이트와 `author` / `reviewer` / `confidence`

이 절은 OpsKnow Repo 의 가장 중요한 운영 약속을 정의합니다. AI 가 생성한 모든 문서는 PR(Pull Request) 단위로 사람 reviewer 의 승인 후에만 main 브랜치에 머지됩니다. AI 가 직접 main 에 push 하는 경로는 차단되고, frontmatter 의 `author` / `reviewer` / `confidence` 3 필드가 책임의 흐름을 기록합니다 [S47][S61]. 이 약속이 없으면 AI 환각 1건이 운영 매뉴얼 전체를 오염시킬 수 있으며, 이 약속이 있으면 AI 가 매일 100건의 초안을 만들어도 사람이 승인한 N건만 RAG corpus 에 들어옵니다.

[FIGURE: human-ai-edit-flow] **캡션**: 사람-AI 공동편집의 표준 흐름 — AI 초안 → PR → reviewer 승인 → 머지 → RAG 재인덱싱 **의도**: 좌→우 횡 흐름의 5 단계 다이어그램입니다. 1단계 AI 초안 작성(frontmatter `author: ai`, `confidence: low|medium|high`). 2단계 PR 생성(Git push + draft PR 자동 변환). 3단계 reviewer 승인(frontmatter `reviewer: <name>` 박힘 + CODEOWNERS 자동 할당). 4단계 머지(main 브랜치 보호 규칙 통과). 5단계 인덱서 트리거(git pull → 변경 청크 임베딩 → 벡터 DB 갱신). 각 단계 하단에 "사람 책임" / "AI 책임" 라벨을 두어 RACI 책임 분할을 시각화합니다.



역할	AI 초안	PR 생성	승인	머지	재인덱싱
AI Agent	R · A	R	—	—	—
reviewer (사람)	C	I	R · A	A	I
CI / Indexer	—	A	I	R	R · A

R = Responsible · A = Accountable · C = Consulted · I = Informed (3 단계 reviewer 가 핵심 게이트)

3.3.1 AI 가 생성한 diff 의 가시화 (PR 단위)

AI 의 변경분이 PR diff 로 가시화되어야 하는 이유 AI 가 main 브랜치에 직접 commit 하는 경로가 열려 있으면, 운영팀이 변경을 인지하기 전에 RAG corpus 에 잘못된 런북이 박힐 수 있습니다 [S61]. PR 워크플로우는 모든 AI 변경분을 line-by-line 의 diff 로 가시화하여 reviewer 가 한 줄씩 검토할 수 있게 합니다. 운영팀이신 여러분께서 GitHub 의 PR 화면을 열면 좌측에 원본 줄, 우측에 AI 가 제안한 줄이 나란히 표시되며, 의심스러운 변경은 한 클릭으로 차단할 수 있습니다. 이 흐름이 없으면 AI 도입은 거버넌스가 깨지는 위험을 안고 갑니다.

Continue.dev 의 Edit/Apply 패턴이 보여준 가시화 모범 Continue.dev 의 사람-AI 공동편집 모델은 OpsKnow Repo 의 PR 게이트 설계에 직접 영향을 준 레퍼런스입니다 [S47]. Continue.dev 는 AI 가 파일을 직접 저장하지 않고 "이 줄을 이렇게 바꾸자" 는 제안을 IDE 안에 띄우며, 사용자가 Apply 버튼을 누른 것만 저장됩니다. OpsKnow Repo 의 PR 게이트는 이 패턴을 Git 단위로 확장한 것입니다 — AI 가 branch 를 만들고 PR 을 띄우면, 사람이 Approve 한 것만 main 으로 들어옵니다.

PR 1건의 표준 구조 — frontmatter 변경 + 본문 변경 + 커밋 메시지 AI 가 작성한 PR 1건은 세 부분의 변경을 포함합니다. 첫째, frontmatter 에 `author: ai` , `confidence: <등급>` , `model: <모델명>` 3 필드가 박힙니다. 둘째, 본문이 변경되거나 신규 작성됩니다. 셋째, 커밋 메시지에 AI 의 생성 근거(어떤 소스 문서·어떤 질의에 답하여 작성했는가)가 자동으로 박힙니다 [S33]. 이 구조가 표준화되어 있으면 reviewer 는 3분 안에 PR 의 신뢰도를 평가할 수 있습니다 — confidence 가 `low` 면 의심을, `high` 면 빠른 승인을 기본값으로 둡니다.

PR 메타	AI 생성 PR	사람 작성 PR
`author` (frontmatter)	`ai`	사용자 식별자
`confidence` (frontmatter)	`low / medium / high`	(없음 또는 `high`)
`model` (frontmatter)	`llama-3.1-8b` 등	(없음)
reviewer 자동 할당	CODEOWNERS + confidence 기반 2인 강제 (low)	CODEOWNERS 1인
본문 diff 가시화	line-by-line (GitHub UI)	line-by-line (GitHub UI)

3.3.2 사람 reviewer 1인 이상 승인 — 오염의 게이트가 사람입니다

CODEOWNERS 가 도메인 책임자 자동 할당의 기반이 되는 이유 GitHub 의 CODEOWNERS 파일은 디렉터리·파일 패턴별로 reviewer 를 자동 할당합니다 [S60]. OpsKnow Repo 에서는 `Field_Manual/payment-api/` 의 변경 PR 은 자동으로 `@team-payment` 가 reviewer 로 할당되고, `Incidents/` 의 변경 PR 은 자동으로 `@team-sre` 가 할당됩니다. 이 자동 할당이 있으면 AI 가 생성한 PR 의 reviewer 가 빠지는 회귀가 사라집니다 — 코드에 박혀 있어 사람의 잊음에도 안전합니다.

low confidence PR 의 2인 reviewer 강제 정책 AI 가 자신의 confidence 를 `low` 로 표시한 PR 은 단일 reviewer 의 승인으로는 머지되지 않으며, 2인 reviewer 의 승인이 모두 필요한 정책을 권장합니다 [S47]. 운영팀이신 여러분께서 GitHub branch protection rule 에서 `Required number of approvals: 2 if confidence == low` 같은 조건을 설정하시면, AI 가 의심스럽다고 표시한 변경은 자동으로 2인 검토 게이트를 통과해야 합니다. 이 정책은 AI 의 자기 진단(낮은 confidence)에 사람의 검토를 가중하여 환각의 확률을 곱셈으로 낮춥니다.

Simon Willison · Charity Majors 의 합의 — AI 는 초안, 사람은 승인자 사전 조사 자료에서 Simon Willison 과 Charity Majors 의 공통된 권고는 "AI 가 쓴 문서를 사람이 그대로 머지하지 마라; AI 는 초안, 사람은 승인자" 입니다 [S61]. 이 권고는 단순한 모범 사례가 아니라 운영 거버넌스의 1차 게이트입니다. 운영 런북은 인시던트 중에 사람이 따라야 할 절차이며, 이 절차가 환각으로 오염되면 인시던트 대응이 실패합니다. OpsKnow Repo 의 PR 게이트는 이 권고를 코드에 박은 것이며, 게이트가 없는 도구는 어떤 AI 모델을 쓰든 같은 위험을 안고 있습니다.

confidence	reviewer 자동 할당	머지 조건	RAG 노출
`high`	CODEOWNERS 1인	1인 승인	즉시
`medium`	CODEOWNERS 1인	1인 승인 + 24시간 cooling	즉시

confidence	reviewer 자동 할당	머지 조건	RAG 노출
`low`	CODEOWNERS 2인 강제	2인 승인	머지 후 48시간 후

3.3.3 `author: ai` + `reviewer: human` 메타로 출처 추적 가능

메타 출처 추적이 분기 감사의 1차 근거가 되는 이유 운영 매뉴얼이 AI 산출물로 점차 채워질 때 가장 중요한 메트릭은 "이 분기에 AI 가 작성한 문서의 비율" 과 "그 중 사람이 승인한 비율" 입니다 [S47]. frontmatter 의 `author` 와 `reviewer` 두 필드가 박혀 있으면 이 메트릭을 ripgrep 한 번으로 추출할 수 있습니다 — `rg "author: ai" Field_Manual/ | wc -l` 한 줄이면 AI 작성 문서 수가 나옵니다. 이 가시성이 있으면 운영팀이신 여러분께서 매 분기 회고에서 AI 도입의 효과와 위험을 데이터로 평가할 수 있습니다.

책임 소재 흐려짐의 위험과 해결책 AI 도입 후 가장 큰 거버넌스 위험은 "이 문서를 누가 책임지는가" 가 흐려지는 것입니다. AI 가 작성하고 사람이 빠르게 승인한 런북이 인시던트 중 잘못된 절차로 드러났을 때, AI 가 작성한 책임과 사람이 승인한 책임이 분리되어 박혀 있어야 사후 분석이 가능합니다 [S61]. `author: ai` + `reviewer: kim.yj` 같은 두 필드의 결합은 "AI 가 초안을 만들었고 김 모씨가 승인했다" 는 사실을 명시적으로 박는 메커니즘입니다. 책임 소재가 흐려지지 않으면 운영팀의 학습 곡선이 가팔라집니다.

RAG 답변 시 출처 추적이 환각 방어선이 되는 이유 RAG 시스템이 사용자 질의에 답변할 때 인용한 문서의 frontmatter 가 노출되면, 사용자는 "이 답변의 출처가 AI 작성 + 사람 승인인지, 사람 작성인지" 를 즉시 확인할 수 있습니다 [S30]. 답변에 `Source: Field_Manual/payment-api/runbook-restart.md (author: ai, reviewer: kim.yj, confidence: high)` 같은 메타가 함께 노출되면, 운영자는 답변의 신뢰도를 1초 안에 평가할 수 있습니다. 이 가시성이 환각 방어선의 마지막 한 줄입니다 — 출처가 박혀 있으면 의심이 가능하고, 박혀 있지 않으면 의심이 불가능합니다.

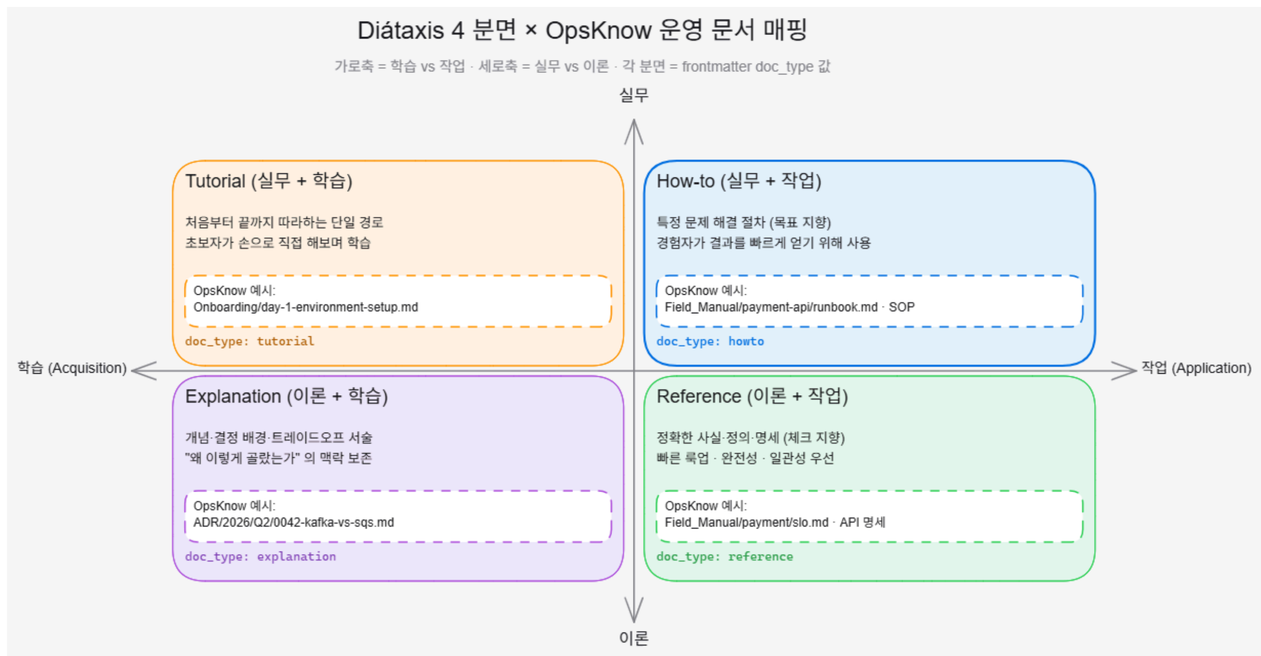
메트릭	측정 방법	분기 회고 활용
AI 작성 문서 수	`rg "author: ai" -l | wc -l`	AI 도입 가속 평가
사람 승인 문서 수	`rg "reviewer: " -l | wc -l`	승인 게이트 작동 평가
low confidence 비율	`rg "confidence: low" | wc -l`	AI 모델 품질 평가
reviewer 평균 응답 시간	PR 메타 분석	게이트 병목 식별

3.4 Diátaxis 차용 — 문서 종류별 템플릿

Diátaxis 는 Daniele Procida 가 제시한 문서 분류 프레임워크이며, 모든 기술 문서를 Tutorial / How-to / Reference / Explanation 4분면으로 분리하라는 권고입니다 [S62]. 이 절은 4분면을 OpsKnow Repo 의 운영 문서에 1:1 매핑하여 종류별 템플릿을 정의합니다. 종류 분리가 중요한 이유는 사전 조사 자료가 박은 한 줄로 요약됩니다 — "문서 종류 분리 시 AI 오용 감소". RAG 가 답변할 때 질문 유형과 일치하는 문서 종류만 검색하면 학습용 설명과 절차가 답변에 뒤섞이는 회귀가 줄어듭니다.

[FIGURE: diataxis-mapping] **캡션**: Diátaxis 4분면 × OpsKnow 운영 문서 매핑 **의도**: 2×2 격자 다이어그램입니다. 가로축은 "학습 vs 작업", 세로축은 "실무 vs 이론" 입니다. 4분면에 (Tutorial / How-to / Reference / Explanation) 을 배치하고 각 분면 안에 OpsKnow 의 해당 문서 종류 1~2개를 박습니다 —

How-to 분면에는 런북·SOP, Reference 분면에는 SLO 정의서·API 명세, Explanation 분면에는 ADR(아키텍처 결정 기록), Tutorial 분면에는 신규 입사자 온보딩 가이드. 각 분면 하단에 frontmatter `doc_type` 값(`howto` / `reference` / `explanation` / `tutorial`)을 표기합니다.



3.4.1 Tutorial → 신규 입사자 온보딩 가이드

Tutorial 의 핵심 — 처음부터 끝까지 따라하는 단일 경로 Tutorial 은 초보자가 처음부터 끝까지 따라하면서 학습하도록 설계된 문서입니다. 단일 경로 — 분기 없이 1번부터 N번까지 순서대로 — 가 핵심이며, 중간에 "선택 사항" 이나 "심화 설명" 을 끼워 넣지 않습니다 [S62]. OpsKnow Repo 에서 Tutorial 의 대표 사례는 신규 입사자 온보딩 가이드입니다 — "1일차 환경 설정 → 2일차 첫 런북 따라하기 → 3일차 모니터링 대시보드 둘러보기" 같은 시간순 단일 경로입니다. 이 구조가 무너지면 학습자가 어디서 시작해야 할지 잃어버립니다.

Tutorial 과 How-to 의 혼동이 만드는 회귀 Tutorial 과 How-to 는 둘 다 "절차" 를 다루지만 독자 가정이 다릅니다. Tutorial 의 독자는 도메인을 처음 접하는 신규 입사자이며, How-to 의 독자는 도메인을 알고 특정 문제를 풀려는 운영자입니다. 두 종류가 한 문서에 섞이면 — 예를 들면 인시던트 대응 런북에 "Kubernetes 란 무엇인가" 설명이 끼어들면 — 인시던트 중에 사람이 런북을 따라 읽는 속도가 느려집니다 [S52]. AI 가 런북을 생성할 때 학습용 설명을 끼워 넣는 회귀가 자주 발생하는데, `doc_type: howto` 메타가 박혀 있으면 reviewer 가 PR 단계에서 즉시 차단할 수 있습니다.

Tutorial 의 frontmatter 권장 필드 Tutorial 종류는 frontmatter 에 `doc_type: tutorial` 외에 `prerequisites` (선행 학습 항목)와 `estimated_duration` (예상 소요 시간) 두 보조 필드를 권장합니다. 신규 입사자가 "이 가이드를 끝내려면 시간이 얼마나 걸리는가" 와 "선행 학습이 필요한가" 를 첫 줄에서 확인할 수 있어야 하기 때문입니다. 운영팀이신 여러분께서 온보딩 가이드를 작성하실 때 이 두 필드를 박으시면 신규 입사자의 학습 곡선이 측정 가능해집니다.

항목	Tutorial 권장	회피해야 할 패턴
독자 가정	도메인 첫 접촉	도메인 전문가

항목	Tutorial 권장	회피해야 할 패턴
경로	단일 순서 (1 → N)	다분기 (if/else)
분량	1~3 시간 학습 분량	30분 미만 또는 1일 이상
frontmatter 필수	doc_type: tutorial + prerequisites + estimated_duration	doc_type 누락

3.4.2 How-to → 런북·SOP

How-to 의 핵심 — 짧고 실행 가능한 절차 How-to 는 특정 목표를 달성하는 단계별 절차이며, 짧고 실행 가능해야 합니다 [S62]. OpsKnow Repo 에서 How-to 의 대표 사례는 운영 런북과 SOP(표준 작업 절차)입니다. "결제 API 의 503 에러가 발생했을 때 재시작 절차" 같은 런북은 평균 50~150 줄, 인시던트 중에 사람이 5분 안에 따라 읽을 수 있는 분량이어야 합니다. Google SRE Book Ch.6 은 모든 알람에 런북 링크가 박혀 있어야 한다고 권고하는데 [S52], 이 권고가 의미를 가지려면 런북이 인시던트 중에 빠르게 읽힐 수 있는 형식이어야 합니다.

런북에 학습용 설명을 넣으면 안 되는 이유 인시던트 한가운데에서 운영자가 런북을 열었을 때 첫 줄에 "Kubernetes Pod 가 무엇인지부터 살펴봅시다" 같은 학습 안내가 있으면, 운영자는 30초를 잃습니다. 이 30초가 모이면 인시던트의 MTTR(Mean Time To Recovery, 평균 복구 시간)이 늘어납니다 [S33]. 런북의 1차 독자는 도메인을 이미 아는 운영자이며, 학습이 필요한 신규 입사자는 Tutorial 종류의 온보딩 가이드로 안내되어야 합니다. AI 가 런북을 생성할 때 학습용 설명을 자동으로 끼워 넣지 않도록 프롬프트에 "Diátaxis How-to 형식: 학습용 설명 배제, 절차만" 같은 가드를 박는 것이 표준 패턴입니다.

런북 평균 길이와 검색 성공률의 관계 사전 조사 자료에 따르면 런북의 평균 길이가 100~200 줄일 때 RAG 검색 성공률이 가장 높습니다 [S52]. 너무 짧으면(50 줄 미만) 컨텍스트가 부족해 RAG 가 답을 만들 수 없고, 너무 길면(500 줄 초과) 청크 분할이 의미를 깨거나 RAG 가 핵심을 놓칩니다. 운영팀이신 여러분께서 현재 런북의 평균 길이를 점검하시고, 500 줄 초과 런북은 두세 개의 How-to 로 분리하시기를 권장합니다.

항목	How-to 권장	회피해야 할 패턴
독자 가정	도메인 전문가 (운영자)	도메인 첫 접촉
경로	다분기 가능 (if/else 허용)	학습용 설명 끼어듦
분량	100~200 줄 (5분 읽기)	500 줄 초과
frontmatter 필수	doc_type: howto + service + severity	학습 설명 섹션

3.4.3 Reference → SLO 정의서·API 명세

Reference 의 핵심 — 검색해서 즉시 답을 얻는 표·정의 Reference 는 사용자가 "X 의 값이 무엇인가" 같은 사실을 검색하기 위해 여는 문서이며, 표·정의·임계값·매개변수 목록이 핵심 콘텐츠입니다 [S62]. OpsKnow Repo 에서 Reference 의 대표 사례는 SLO 정의서, API 명세, 임계값 표, 환경 변수 목록입니다. "결제 API 의 SLO 가 99.9% 가용성이라는 사실" 을 알고 싶을 때 SLO 정의서의 표 한 줄을 보면 됩니다. 설명을 읽을 필요가 없습니다.

Reference 에 설명을 섞으면 RAG 정확도가 떨어지는 이유 Reference 종류 문서에 "이 SLO 가 왜 99.9% 인가" 같은 설명이 섞이면 RAG 가 사실 검색 질의에 답할 때 설명문을 함께 인용하여 답변이 길어집니다 [S35]. 사용자는 "SLO 가 99.9% 다" 한 줄을 원했는데 답변이 5문단의 배경 설명을 포함하면, 사용자는 다시 핵심을 찾아 읽어야 합니다. 설명이 필요한 경우는 Explanation 종류의 별도 문서로 분리하는 것이 Diátaxis 의 원칙이며, OpsKnow Repo 는 이 원칙을 `doc_type` enum 으로 코드에 박습니다.

Reference 의 표 구조 권장 형식 Reference 문서는 표 중심 구조를 권장합니다. 첫 열은 검색 키(예: SLO 이름, API 엔드포인트, 환경 변수명), 둘째 열은 값, 셋째 열은 단위 또는 유효 범위, 넷째 열은 출처 또는 변경 이력입니다. 이 구조가 표준화되어 있으면 RAG 가 표 한 행을 그대로 답변으로 인용할 수 있어 응답 시간이 빠르고 정확합니다 [S22].

항목	Reference 권장	회피해야 할 패턴
핵심 콘텐츠	표 + 정의	설명 단락
독자 사용 패턴	검색 후 즉시 닫음	처음부터 끝까지 읽기
분량	표 행 수에 비례	(제한 없음)
frontmatter 필수	`doc_type: reference` + `sources`	배경 설명 섹션

3.4.4 Explanation → 아키텍처 결정 기록 (ADR)

Explanation 의 핵심 — "왜 그렇게 했는가" 의 내러티브 Explanation 은 시스템·결정의 배경·맥락·트레이드오프를 다루는 문서이며, "왜" 라는 질문에 답합니다 [S62]. OpsKnow Repo 에서 Explanation 의 대표 사례는 ADR(Architecture Decision Record)입니다. "왜 PostgreSQL 대신 Cassandra 를 선택했는가" 같은 결정의 배경과 대안, 그리고 그 결정의 결과를 시간순으로 기록합니다 [S19]. ADR 이 없으면 후임자가 같은 결정 회의를 1년 후에 반복하게 되고, 또 1년 후에 다시 반복하게 됩니다 — 운영 지식이 머릿속 의존으로 회귀합니다.

ADR 의 표준 템플릿 — 제목·맥락·결정·결과·대안 ADR 의 표준 템플릿은 다섯 섹션으로 구성됩니다. 첫째 제목(예: `ADR-0007: Cassandra 채택`). 둘째 맥락(이 결정이 필요했던 배경). 셋째 결정(무엇을 결정했는가). 넷째 결과(이 결정의 긍정·부정 결과). 다섯째 대안(검토했지만 채택하지 않은 선택지와 그 이유). 이 템플릿이 표준화되어 있으면 분기 회고에서 ADR 들을 시간순으로 훑으면서 "이 분기에 우리가 어떤 결정을 했고 어떤 결과가 나왔는가" 를 한 시간 안에 정리할 수 있습니다 [S19].

ADR 이 운영 학습 곡선을 가팔라지게 하는 이유 신규 입사자가 시스템의 현재 모습을 보고 "왜 이렇게 됐지" 라고 질문할 때, ADR 모음이 답입니다. ADR 이 없으면 그 답은 시니어 운영자의 머릿속에 있고, 시니어가 퇴사하면 답이 사라집니다. OpsKnow Repo 의 `_meta/decisions/` 디렉터리는 ADR 의 시간순 누적이며, RAG 가 "왜" 질문에 답할 때 우선 검색하는 corpus 입니다 [S62]. 운영팀이신 여러분께서 지난 1년 주요 결정 중 ADR 로 박혀 있는 비율을 점검하시면, 운영 지식 누적의 건강 상태를 측정할 수 있습니다.

항목	Explanation 권장	회피해야 할 패턴
핵심 콘텐츠	배경 + 결정 + 대안	절차만 (How-to 와 혼동)
독자 가정	"왜" 가 궁금한 운영자	"어떻게" 가 궁금한 운영자

항목	Explanation 권장	회피해야 할 패턴
분량	200~500 줄 (15분 읽기)	50줄 미만 (배경 부족)
frontmatter 필수	`doc_type: explanation` + `sources` + 결정 일자	절차 단계 섹션

3.5 유효성 관리 — 만료·재검토·소유자 알림

운영 런북은 시간이 지나면 거짓말이 됩니다. 시스템이 바뀌고, 임계값이 조정되고, 의존 서비스가 교체되어도 런북이 갱신되지 않으면 인시던트 중에 잘못된 절차를 따라가게 됩니다 [S6]. 이 절은 운영 지식이 시간의 흐름에 따라 신뢰성을 잃지 않도록 `freshness_until` 메타와 자동 알림·RAG 가중치 하향 메커니즘을 정의합니다. 유효성 관리가 없으면 1년 후 런북의 절반은 신뢰할 수 없는 상태가 됩니다 — MkDocs·Hugo 같은 정적 사이트 생성기는 이 메커니즘이 표준이 아니어서 운영 지식 저장소로는 부족합니다.

3.5.1 `freshness_until: 2026-12-31` 의 의미와 알림 트리거

freshness_until 이 약속의 메타가 되는 이유 `freshness_until` 필드는 단순한 만료일이 아니라 작성자의 약속입니다 — "이 날짜까지는 내가 책임지고 갱신하겠다" 는 약속이며, 이 날짜 이후에는 재검토 없이는 신뢰하지 말라는 신호입니다 [S6]. 이 약속이 메타에 박혀 있으면 RAG 가 검색 시 자동으로 만료 임박 문서를 제외하거나 가중치를 낮출 수 있습니다. 운영팀이신 여러분께서 런북을 작성하실 때 `freshness_until: 2026-12-31` 한 줄을 박으시면, 7개월 후에 자동 알림이 owner 에게 도달하여 재검토 사이클이 트리거됩니다.

60·30·7일 3 단계 알림 권장 정책 만료 임박 알림은 60일·30일·7일 세 시점에 owner 에게 발송되는 정책을 권장합니다 [S6]. 60일 알림은 owner 가 재검토 일정을 잡을 시간을 주고, 30일 알림은 실제 재검토 시작을 트리거하며, 7일 알림은 만료 직전 경고입니다. 만료일 도래 후에도 갱신이 없으면 자동으로 `status: stale` 이 박히고 RAG 가중치가 하향됩니다. 이 3 단계 알림이 없으면 owner 가 만료를 잊고 런북이 6개월간 stale 상태로 RAG corpus 에 남아 있을 위험이 있습니다.

만료 메타와 알림이 함께 박혀야 의미를 갖는 이유 `freshness_until` 만 박혀 있고 알림이 없으면 만료 메타는 형식적인 라벨에 그칩니다. 반대로 알림만 있고 메타가 없으면 알림 시점을 결정할 수 없습니다. 두 요소가 함께 작동해야 유효성 관리가 자리잡습니다. OpsKnow Repo 의 권장 구현은 GitHub Action 또는 cron 스크립트가 매일 모든 MD 파일의 `freshness_until` 을 스캔하여 60·30·7일 임박 항목을 owner 에게 알리는 것입니다 [S19].

알림 시점	트리거 메시지	권장 owner 대응
60일 전	"재검토 일정 예약 권장"	캘린더에 재검토 일정 추가
30일 전	"재검토 시작 권장"	본문 점검 + 갱신 PR 시작
7일 전	"만료 임박"	갱신 PR 머지 또는 만료 수락
만료일 도래	`status: stale` 자동 박힘	RAG 가중치 자동 하향

3.5.2 Stale 진입 시 RAG 자동 가중치 하향

완전 배제가 아닌 가중치 하향을 선택하는 이유 `freshness_until` 이 지난 문서를 RAG corpus 에서 완전히 배제하면 검색 결과 부재가 발생할 수 있습니다 — 다른 신선한 런북이 없는 영역에서는 만료된 런북이라도 마지막 단서일 수 있기 때문입니다 [S33]. OpsKnow Repo 의 권장 정책은 완전 배제가 아닌 가중치 하향입니다. stale 상태의 문서는 검색 가중치가 50% 또는 30% 로 하향되어, 신선한 문서가 있으면 신선한 문서가 우선되고, 신선한 문서가 없으면 stale 문서가 차선으로 인용되되 답변에 "출처가 만료된 문서임" 경고가 함께 노출됩니다.

Field Manual 5단계 상태기계와의 연결 2장의 Field Manual 5단계 상태기계(Draft → Review → Approved → Stale → Archived)는 이 절의 RAG 가중치 하향 정책과 직접 연결됩니다 [S33]. Approved 단계는 RAG 가중치 100%, Stale 단계는 50% 또는 30%, Archived 단계는 RAG corpus 에서 완전 배제(가중치 0)입니다. 상태 전이는 frontmatter 의 `status` 필드와 `freshness_until` 의 결합으로 자동 결정되며, 사람이 수동으로 Archived 로 전이시킬 수도 있습니다. 이 정책이 코드에 박혀 있으면 운영자가 stale 문서를 매번 수동으로 격리할 필요가 없습니다.

가중 함수의 권장 설계 — 지수적 감소 가중치 하향 함수는 만료 후 경과 일수에 따라 지수적으로 감소하는 형태를 권장합니다 [S39]. 만료 직후 7일은 가중치 0.7, 30일 후 0.5, 90일 후 0.3, 1년 후 0.1 같은 식입니다. 선형 감소는 만료 직후에도 너무 급격하게 떨어져 검색 결과 부재를 만들 수 있고, 계단식 감소는 임계 시점 직전·직후의 답변이 급변하는 회귀를 만들 수 있습니다. 지수적 감소는 두 극단을 회피하면서 시간이 갈수록 점차 사용을 줄이는 자연스러운 곡선을 만듭니다.

상태	만료 후 경과	RAG 가중치	답변 노출 정책
Approved (신선)	-	1.0	즉시 인용
Stale (임박)	만료 7일 전~만료일	0.7	인용 + "갱신 임박" 경고
Stale (만료)	1~30일	0.5	인용 + "출처 만료" 경고
Stale (장기 만료)	30~90일	0.3	대안 없을 때만 인용
Archived	90일 이후 또는 수동	0.0	인용 안 함

3.5.3 만료 임박 알림 — 사내 메신저 DM / 이메일 / 이슈 자동 생성

3 경로 알림이 owner 휴가·이직에 안전한 이유 알림 경로가 사내 메신저 DM 한 가지뿐이면 owner 가 휴가 중이거나 사내 메신저 알림을 꺼두었을 때 만료가 그대로 지나갈 수 있습니다. OpsKnow Repo 의 권장 정책은 사내 메신저 DM·이메일·GitHub 이슈 자동 생성 세 경로를 모두 활성화하는 것입니다 [S19]. 사내 메신저 DM 은 즉시성, 이메일은 휴가 중에도 도달, GitHub 이슈는 owner 부재 시 팀이 인계 가능 — 세 경로가 서로 다른 장애 모드에 강합니다. 운영팀이신 여러분께서 도입 초기에 세 경로 중 하나만 활성화하시더라도, 6개월 이내에 나머지 두 경로를 활성화하시는 로드맵을 권장합니다.

GitHub Action 으로 구현하는 패턴 유효 만료일 알림은 GitHub Action 으로 구현하는 것이 가장 가벼운 패턴입니다. 매일 `cron: '0 9 * * *'` 으로 트리거되는 워크플로우가 모든 MD 파일의 frontmatter 를 스캔하여 60·30·7일 임박 항목을 추출하고, 사내 메신저 Webhook·이메일·GitHub Issue API 세 경로로 알림을 발송합니다 [S6]. 이 워크플로우의 구현은 100 줄 미만의 Python 스크립트로 가능하며, 운영팀이 별도 인프라를 추가하지 않아도 작동합니다. cron 스크립트로 구현해도 동일한 결과를 얻을 수 있으며, GitHub Action 의 장점은 별도 서버 없이 GitHub 위에서 작동한다는 점입니다.

알림 도달률 측정과 개선 사이클 알림이 발송되어도 owner 가 인지하지 못하면 의미가 없습니다. 운영팀이신 여러분께서 분기 회고에서 측정하셔야 할 메트릭은 "만료 임박 알림의 7일 이내 PR 응답률" 입니다 [S19]. 이 응답률이 80% 이상이면 알림 시스템이 작동하고 있고, 50% 미만이면 알림 경로가 owner 의 일상 흐름에 맞지 않거나 owner 부담이 과도한 상태입니다. 응답률이 낮으면 알림 경로를 추가하거나, owner 부담을 줄이기 위해 AI 가 갱신 초안을 자동 작성하는 패턴(2.2.3 절의 자동화)을 도입하는 것이 다음 단계입니다.

알림 경로	즉시성	휴가 중 도달	팀 인계 가능	구축 비용
사내 메신저 DM	최상	낮음	낮음	낮음 (Webhook 1 개)
이메일	중	최상	낮음	낮음 (SMTP 또는 SendGrid)
GitHub 이슈 자동 생성	중	중	최상	낮음 (Issues API)
3 경로 동시 활성화 (권장)	최상	최상	최상	약 반나절 (스크립트 1회 작성)

3장의 다섯 절은 OpsKnow Repo 가 단지 디렉터리를 한 곳에 모은 것이 아니라 다섯 가지 약속 위에 운영 지식을 쌓는 시스템임을 보였습니다. MD 가 표준 단위라는 첫 번째 약속은 포맷 선택의 합리성을 만들고, 디렉터리·파일명·frontmatter 표준은 운영 ontology 를 표현하며, PR 게이트는 사람·AI 공동편집의 신뢰 기반이고, Diátaxis 는 문서 종류를 정렬하며, 유효성 관리는 런북이 시간에 따라 거짓말이 되는 것을 막습니다. 이 다섯 약속이 함께 코드에 박혀 있을 때만 운영 지식이 흩어짐 없이 누적되며, 다음 장(4장)이 다룰 Local LLM RAG 스택이 의미 있는 답변을 만들어 낼 수 있습니다.

4장. 표준 스택 — Ollama / vLLM / 임베딩 / 벡터 DB

운영 지식 레포지토리을 사람과 AI 가 함께 읽으려면 결국 한 가지 질문에 답해야 합니다. "어떤 구성요소를 어떤 순서로 쌓아야 외부 API 호출 0건이 검증 가능한 RAG 스택이 완성되는가" 입니다. 이 질문은 분기 예산 회의에서 한 줄로 답해야 하는 문제이고, 동시에 첫 GPU 박스를 결정하는 엔지니어의 손끝에서 풀려야 하는 문제이기도 합니다.

본 장은 표준 스택을 네 계층으로 나누어 설명합니다. 첫 번째 계층은 LLM 런타임입니다. Ollama 와 vLLM 의 의사결정 기준을 사용자 수, GPU 수, 운영 부담의 세 기준으로 정리합니다 [S42] [S43]. 두 번째 계층은 임베딩 모델입니다. 한국어 비중이 높은 운영 문서에서 BGE-M3 가 1차 선택이 되는 이유, 그리고 짧은 영문 문서 위주 환경에서 nomic-embed-text 가 대안이 되는 경계선을 살펴봅니다 [S55]. 세 번째 계층은 벡터 DB 입니다. LanceDB, Chroma, Qdrant 의 규모별 적합 구간을 청크 수로 끊어 권장합니다 [S33] [S39] [S37]. 네 번째 계층은 클라이언트입니다. 채팅 UI 표준인 Open WebUI, IDE 통합인 Continue.dev, 자동화 진입점인 자체 CLI 가 어떻게 서로 보완 관계를 이루는지 정리합니다 [S39] [S47].

표준 스택이라는 단어를 쓰지만, 이 단어가 절대적 정답을 의미하지는 않습니다. r/LocalLLaMA megathread 가 정리한 합의는 "Ollama + Open WebUI 또는 Ollama + AnythingLLM 조합이 사실상 표준" 이라는 정도이며 [S58], 사용자 수가 늘면 vLLM 으로, 청크가 늘면 Qdrant 로 옮겨가는 변환점이 명확하게 존재합니다.

이 변환점을 미리 알고 들어가면 6주 안에 첫 도입을 마치고 12개월 안에 100명 사용자 환경까지 무리 없이 늘릴 수 있습니다 [S43].

또 한 가지 강조하고 싶은 점은 OpenAI 호환 API 라는 사실상의 표준입니다. Ollama 와 vLLM 두 런타임이 모두 동일한 인터페이스를 제공하기 때문에, 클라이언트 코드를 한 줄도 바꾸지 않고 런타임을 갈아끼울 수 있습니다 [S42] [S43]. 이는 벤더 종속 위험을 0 에 가깝게 만드는 네 번째 기둥이며, 모델 스왑이 분 단위로 가능한 lock-in 회피 구조의 핵심입니다. 여러분이 한 번 결정하고 6개월 뒤 후회하지 않을 안전판은 바로 이 호환성입니다.

4.1 LLM 런타임 — Ollama vs vLLM 의 의사결정 기준

LLM 런타임을 고르는 일은 GPU 한 장의 가격이나 모델 한 종의 벤치마크보다 운영 부담의 형태를 먼저 봅니다. 처음 6주 안에 첫 Field Manual 을 이식하고 RAG 채팅이 답을 돌려주는 것을 보아야 다음 분기 예산이 잡힙니다. 그래서 첫 런타임 선택은 처리량 극대화보다 운영 부담 최소화 쪽으로 기우는 편이 안전합니다.

운영팀 입장에서 가장 먼저 답을 받아야 하는 질문 세 가지가 있습니다. 사내 GPU 한 장으로 시작할 수 있는가, 사용자 수가 몇 명 임계를 넘으면 다른 런타임으로 옮겨야 하는가, 그리고 런타임을 갈아끼울 때 클라이언트 코드를 다시 짜야 하는가 입니다. 본 절은 이 세 질문을 Ollama, vLLM, OpenAI 호환 API 세 항으로 나누어 답합니다 [S42] [S43].

표준 스택 4계층 가운데 LLM 런타임이 가장 먼저 결정되는 이유는 단순합니다. 임베딩 모델과 벡터 DB 는 런타임의 사용량 곡선이 그려진 다음에야 적정 규모가 잡힙니다. 그래서 본 절은 4장의 척추에 해당합니다.

4.1.1 Ollama — 단일 GPU·소수 사용자·운영 부담 최소

Ollama 의 가장 큰 가치는 진입 비용이 한 자릿수 시간이라는 점입니다. 사전 조사가 정리한 그대로 "`ollama run` 한 줄로 GGUF 모델을 OpenAI 호환 API 로 띄우는 최소 설치 런타임" [S42] 이라는 정의가 이 도구의 본질입니다. 사내 워크스테이션 한 대에 24GB 메모리 GPU 한 장이 꽂혀 있다면, 모델 다운로드 시간을 제외하고 30분 안에 첫 질의응답이 가능합니다.

운영 부담 최소화의 실질은 세 가지입니다. 첫째, 설치가 단일 바이너리이고 의존성 충돌이 거의 발생하지 않습니다. 둘째, 모델 관리는 Modelfile 한 파일로 끝나서 Docker 이미지를 별도로 빌드할 필요가 없습니다. 셋째, OpenAI 호환 API 가 기본 제공이라 클라이언트 코드는 base_url 한 줄만 바꾸면 됩니다 [S42]. 여러분이 운영팀이라면 이 세 가지가 첫 도입의 4주 마찰을 줄이는 안전판입니다.

적합 구간은 어디까지인가 라는 질문에 대해서는, 사용자 수가 10명 이내이고 동시 호출이 분당 60건 이내인 환경이 1차 권장입니다. Llama 3.1 8B, gpt-oss 20B, Mistral 7B, Gemma 2 9B 같은 7B~20B 급 미국·유럽 오픈웨이트 모델이 단일 24GB GPU 위에서 충분히 동작하며, 운영 질의의 9할인 "이 런북에서 X 항목 알려줘", "최근 3개월 incident 중 service A 관련 항목" 같은 사실 검색은 이 등급의 모델이 충분히 처리합니다 [S55].

운영 부담을 한 번 더 강조하면, 첫 도입의 진입 장벽이 곧 도입의 성공 확률을 결정합니다. 사전 조사가 정리한 r/LocalLLaMA 의 합의 "Ollama + Open WebUI 또는 Ollama + AnythingLLM 조합이 사실상 표준" [S58] 이라는 표현은 단순 인기 투표가 아니라 "운영팀이 6주 안에 첫 답을 받을 수 있는가" 라는 현실 시험에 가장 많이 통과한 조합이라는 뜻으로 읽어야 합니다.

Ollama 의 설치·기동 절차는 다음 3 단계로 정리됩니다. 의사결정자가 도입 비용을 1시간 단위로 추정할 수 있도록 표로 제시합니다.

단계	명령·작업	예상 소요 시간	비고
1. 설치	단일 바이너리 다운로드, 시스템 서비스 등록	약 10분	macOS·Linux·Windows 동일
2. 모델 다운로드	`ollama pull llama3.1:8b` (예)	약 10~20분	모델 크기·네트워크 의존
3. 첫 질의응답	`ollama run llama3.1:8b` 또는 HTTP API `/api/generate` 호출	약 5분	OpenAI 호환 `/v1/chat/completions` 도 동일

4.1.2 vLLM — 다중 GPU·다수 사용자·고처리량

vLLM 은 사용자 수가 임계를 넘으면 등장하는 다음 단계입니다. 사전 조사가 정리한 그대로 "PagedAttention 으로 GPU 효율 극대화, OpenAI 호환 서버" [S43] 라는 정의가 이 엔진의 본질입니다. PagedAttention 은 운영체제의 가상 메모리 페이징과 같은 방식으로 KV 캐시 메모리를 관리하여, 동시 요청 수십~수백 건을 동일 GPU 자원으로 처리합니다.

전환 시점을 가르는 첫째 신호는 사용자 수 약 50명 임계입니다. 사용자 50명이 분당 2~3건 질의응답을 던지면 동시 활성 호출은 분당 100~150건에 이르고, Ollama 의 단일 프로세스 기반 동시 처리 한계가 보이기 시작합니다 [S43]. 이 임계를 넘으면 응답 지연이 사용자 체감 수준으로 늘어나기 때문에 야간 자동 점검의 응답 시간도 함께 무너집니다.

둘째 신호는 GPU 자원의 수직 확장이 아니라 수평 확장이 필요한 시점입니다. vLLM 은 텐서 병렬화와 파이프라인 병렬화를 모두 지원하기 때문에, 24GB GPU 두 장 또는 80GB GPU 한 장으로 14B~32B 급 모델을 효율적으로 띄울 수 있습니다 [S43]. 동일 모델을 Ollama 에 올린 경우 대비 처리량이 5~10배 차이가 나는 사례가 r/LocalLLaMA 와 vLLM 공식 벤치마크에 다수 보고됩니다 [S43] [S55].

운영 부담은 Ollama 대비 분명히 큽니다. Python 환경 관리, CUDA 버전 정합성, 모델 가중치의 safetensors 변환, 서버 옵션 튜닝 같은 항목이 추가됩니다. 그러나 사용자 100명 이상 환경에서는 이 부담을 감수하는 편이 토큰 비용과 응답 지연 양쪽 모두를 안정시킵니다. 여러분의 사용자 수가 50명을 넘기 시작했다면, Ollama → vLLM 전환은 다음 분기 안건으로 검토할 시점입니다.

세 가지 운영 기준을 한눈에 비교하면 다음과 같습니다. 처리량은 vLLM 의 압승이지만 운영 부담과 하드웨어 요구가 동행하기 때문에, 사용자 곡선이 명확해진 뒤에 전환하는 것이 안전합니다 [S42] [S43].

비교 기준	Ollama	vLLM	결정 신호
처리량 (동시 호출)	분당 60~100건	분당 500~수천 건	동시 호출 분당 100건 초과 시 전환
운영 부담 (설치·튜닝)	단일 바이너리, 한 자릿수 시간	Python·CUDA·옵션 튜닝, 1~3일	운영 인력 0.5명 이상 확보 시 전환

비교 기준	Ollama	vLLM	결정 신호
하드웨어 요구	24GB GPU 한 장	24GB 두 장 또는 80GB 한 장 이상	GPU 추가 예산 확보 시 전환
모델 형식	GGUF (양자화 친화)	safetensors (FP16/BF16 기본)	14B 이상 모델 풀 정밀도 필요 시 전환

참고 — 국내 시장 기준 GPU 사양·예산 (2026년 6월 시점, 단가는 변동)

의사결정권자께서 품의서·기안문에 옮길 수 있도록 4.1.1 (Ollama 단계) · 4.1.2 (vLLM 단계) 의 하드웨어 요구를 실제 모델명·시장가로 변환하면 다음과 같습니다.

단계	하드웨어 사양	대표 모델명 (예시)	국내 시장가 (참고)	비고
PoC · 소규모 (24GB 1장)	NVIDIA RTX 4090 24GB	RTX 4090 / RTX 6000 Ada / L40S 48GB	약 300~1,200 만원 (1회성 구매)	데스크톱 워크스테이션 1대
운영 (80GB 1장)	NVIDIA A100 80GB	A100 80GB SXM/PCIe / H100 80GB	약 3,000~7,000 만원 (1회성 구매)	또는 클라우드 GPU 임대 (시간당 X 천원)
다중 사용자 (80GB 2~4장)	NVIDIA H100 / H200	H100 80GB × 2~4 장	약 1.5~3억원 (서버 1대)	사용자 200명 이상 환경

> ***공공·금융 망분리 환경 권고:*** *사내 자체 워크스테이션 또는 폐쇄망 GPU 서버가 1차 권장입니다. 클라우드 GPU 임대는 CSAP (Cloud Security Assurance Program, 클라우드 보안 인증) 인증 사업자 또는 망분리 예외 승인이 사전에 확보되어 있을 때만 검토 가능합니다. 본 백서의 외부 호출 0건 검증 (6장) 은 사내 GPU 배치를 전제로 합니다.*

4.1.3 OpenAI 호환 API — 두 런타임 모두 동일 클라이언트로 접속 가능

OpenAI 호환 API 는 표면적으로는 사소한 호환성 사양이지만, 실제 운영 현장에서는 lock-in 회피의 4번째 기둥에 해당합니다. Ollama 와 vLLM 두 런타임이 모두 동일한 `/v1/chat/completions` 인터페이스를 제공하기 때문에 [S42] [S43], 클라이언트 코드 변경 없이 런타임을 갈아끼울 수 있습니다. 여기에 LM Studio 와 llama.cpp 의 server 바이너리까지 동일 규약을 따르므로, 표준 스택 전체에 호환 가능한 런타임이 4종 이상 존재합니다 [S44] [S45].

이 호환성이 실질적으로 주는 이점은 세 가지입니다. 첫째, 모델 스왑의 비용이 분 단위로 줄어듭니다. Llama 3.1 8B 에서 gpt-oss 20B 로 갈아탈 때 채팅 UI 와 자동화 스크립트의 base_url 만 바꾸면 됩니다. 둘째, PoC 환경과 운영 환경이 분리됩니다. 노트북의 Ollama 위에서 검증한 프롬프트가 데이터센터의 vLLM 에서 그대로 동작합니다. 셋째, 사외 의존을 제거합니다. 외부 API 호출 0건이라는 정책이 코드가 아닌 인프라 수준에서 보장됩니다 [S34].

코드 한 줄의 차이는 실제로 다음과 같습니다. 동일한 클라이언트가 base_url 만 바뀌면 됩니다.

```
# Ollama 접속
client = OpenAI(base_url="http://localhost:11434/v1", api_key="ollama")

# vLLM 접속
client = OpenAI(base_url="http://vllm-server:8000/v1", api_key="EMPTY")

# LM Studio 접속
client = OpenAI(base_url="http://localhost:1234/v1", api_key="lm-studio")
```

기획 의도가 "Ollama/vLLM 위에서 Llama·gpt-oss·Gemma·Mistral 등 모델 스왑이 분 단위로 가능하다. 벤더 종속 위험이 0 에 가깝다" [S42] 라고 정리한 명제의 코드 차원의 근거가 바로 이 한 줄의 차이입니다. 분기 회의에서 "다음 분기 모델 교체 비용이 얼마인가" 라는 질문에 "엔드포인트 한 줄" 이라고 답할 수 있는 인프라가 곧 표준 스택의 안전판입니다.

여기서 한 가지 짚어둘 점은 호환의 완전성에는 미세한 차이가 있다는 사실입니다. function calling, tool use, streaming 의 일부 옵션은 런타임에 따라 미지원이거나 동작이 다를 수 있습니다 [S43]. 그러나 RAG 의 핵심 호출인 chat completions 와 embeddings 두 엔드포인트는 사실상 모든 런타임이 동일 시그니처를 따르므로, OpsKnow Repo 의 RAG 파이프라인 입장에서는 호환성이 충분합니다 [S39] [S42].

4.2 임베딩 모델 — BGE-M3 / nomic-embed-text 권장

임베딩 모델은 RAG 품질의 절반을 결정합니다. LLM 의 추론력보다 corpus 의 품질이 운영 질의 답변 정확도를 더 크게 좌우하며 [S55], corpus 품질의 출발점이 바로 임베딩 모델 선택과 청크 크기 조합입니다. 본 절은 BGE-M3, nomic-embed-text 두 모델과 청크 권장 조합을 3 항으로 나누어 정리합니다.

선택의 결정 기준은 세 가지입니다. 첫째, 사내 문서의 한국어 비중입니다. 운영 문서가 한국어 중심이면 다국어 토크나이저가 표준 선택이 됩니다. 둘째, 평균 문서 길이입니다. 매뉴얼·런북처럼 긴 문서가 다수면 긴 컨텍스트 임베딩이 유리합니다. 셋째, 벡터 DB 비용입니다. 차원이 작은 임베딩은 메모리·검색 속도 모두에서 이점이 있습니다 [S55].

본 절은 4.1 LLM 런타임 결정 다음에 오는 두 번째 결정 지점입니다. 임베딩 모델은 한 번 결정하면 corpus 전체를 다시 인덱싱해야 바꿀 수 있기 때문에, 첫 도입 시 결정을 되돌리는 비용이 가장 큽니다 [S55].

4.2.1 다국어·한국어 RAG 의 1차 권장 — BGE-M3

BGE-M3 는 한국어 운영 문서가 다수인 환경에서 사실상 1차 선택입니다. 사전 조사가 정리한 r/LocalLLaMA RAG 튜닝 megathread 의 권장 "임베딩은 BGE-M3·nomic-embed-text 가 국내·다국어 권장" [S55] 이라는 합의의 핵심에 BGE-M3 가 있습니다. 권장의 근거는 세 가지로 압축됩니다.

첫째, 토크나이저가 100여 개 언어를 균형 있게 다룹니다. 한국어 형태소가 영어 단어 단위와 다른 경계를 갖기 때문에, 영어 전용 모델은 한국어 문서를 인코딩할 때 불필요한 토큰 분할이 늘어나고 의미 단위가 분산됩니다. BGE-M3 는 다국어 학습 corpus 가 크기 때문에 한국어 문장의 의미 단위가 한 임베딩 벡터에 더 잘 응축됩니다 [S55].

둘째, 긴 컨텍스트를 지원합니다. 최대 8192 토큰까지 한 번에 임베딩 가능하기 때문에, 런북이나 매뉴얼 한 섹션을 통째로 한 벡터로 표현할 수 있습니다 [S55]. OpsKnow Repo 의 H2 단위 청크 권장(300~500 토큰)과

는 다른 차원의 이야기지만, 청크 경계가 어긋난 긴 컨텍스트 fallback 처리에 유리합니다.

셋째, dense, sparse, multi-vector 세 가지 검색 모드를 한 모델로 제공합니다. 의미 기반 검색(dense) 과 키워드 기반 검색(sparse) 을 결합한 하이브리드 검색을 단일 모델로 구현할 수 있어, 운영 질의의 약어·고유명사 검색 정확도가 크게 올라갑니다 [S55] [S33]. 운영팀이 "PG-101 에러" 같은 약어로 질의하는 경우가 많기 때문에, 이 하이브리드 능력은 실질적으로 RAG 정확도의 차이를 만듭니다.

BGE-M3 vs 영어 전용 임베딩의 한국어 RAG 정확도 비교는 다음과 같이 정리됩니다. 절대 수치는 corpus 와 평가 기준에 따라 달라지지만, 한국어 비중이 50% 를 넘는 환경에서는 BGE-M3 의 우위가 분명하게 나타납니다 [S55].

비교 기준	BGE-M3	영어 전용 임베딩 (예: text-embedding-3-small)	사유
한국어 의미 검색 정확도	기준점	약 60~75% 수준 (추정)	한국어 학습 corpus 비중 차이
최대 컨텍스트 길이	8192 토큰	8191 토큰	동급
차원	1024	1536	BGE-M3 가 메모리 절감
하이브리드 검색 (dense+sparse)	단일 모델 지원	별도 BM25 결합 필요	운영 부담 차이

4.2.2 짧은 영문 문서의 대안 — nomic-embed-text

nomic-embed-text 는 짧은 영문 문서가 corpus 의 다수를 차지하는 환경에서 BGE-M3 의 대안이 됩니다. 사전 조사가 같은 megathread 에서 nomic-embed-text 도 함께 권장한 이유는 영문 corpus 에서의 정확도와 운영 부담의 균형 때문입니다 [S55] [S33]. 권장 시나리오의 경계는 다음과 같이 그어집니다.

첫째 시나리오는 외산 오픈소스 매뉴얼 위주의 사내 문서입니다. Kubernetes, Prometheus, Grafana, Istio 같은 오픈소스의 공식 매뉴얼을 그대로 인입한 경우 corpus 의 80% 이상이 영문이 됩니다. 이 환경에서는 nomic-embed-text 의 영문 정확도와 작은 차원이 검색 속도 이점으로 직결됩니다 [S55].

둘째 시나리오는 청크가 짧은 환경입니다. nomic-embed-text 의 768 차원은 BGE-M3 의 1024 차원 대비 약 25% 작기 때문에, 벡터 DB 의 메모리 사용량과 검색 지연이 같은 비율로 줄어듭니다 [S55]. 청크 수가 100 만 개를 넘는 대규모 corpus 에서는 이 차이가 인프라 비용으로 환원됩니다.

셋째 시나리오는 임베딩 호출 빈도가 매우 높은 환경입니다. CI/CD 파이프라인에서 매 빌드마다 변경 파일을 재임베딩하는 경우, 처리 속도가 곧 빌드 시간으로 누적됩니다. nomic-embed-text 의 작은 차원과 가벼운 모델이 이 빈도를 흡수합니다 [S55].

차원·메모리·속도 세 가지 비교 기준은 다음과 같이 정리됩니다. 영문 비중이 80% 이상이면 nomic-embed-text 검토를 권하며, 그 미만에서는 BGE-M3 가 1차 선택으로 남습니다 [S55] [S33].

비교 기준	BGE-M3	nomic-embed-text	결정 신호
차원	1024	768	메모리 25% 절감 필요 시 nomic
한국어 정확도	우수	영문 대비 약 70% 수준 (추정)	한국어 비중 50%+ 시 BGE-M3
영문 정확도	우수	동급 또는 약간 우위	영문 비중 80%+ 시 nomic
최대 컨텍스트	8192 토큰	8192 토큰	동급
검색 속도 (1M 청크)	기준점	약 1.3배 빠름 (추정)	빌드 빈도 높을 시 nomic

4.2.3 임베딩 차원·청크 크기의 권장 조합

청크 크기와 overlap 의 기본값 선택은 RAG 품질의 절반을 결정합니다. 사전 조사 (A-f) 에서 도출한 결론 "LLM 친화 청크 단위(짧은 H2 섹션·체크리스트) 강제" [S55] 가 OpsKnow Repo 의 청크 권장의 출발점입니다. 본 항은 H2 단위 청크 300~500 토큰 + 50 토큰 overlap 을 기본 권장으로 제시하고, 그 사유를 운영 질의 패턴과 함께 정리합니다.

권장의 첫째 사유는 H2 가 자연스러운 의미 경계라는 점입니다. Field Manual·Postmortem·OPS Diary 의 MD 구조에서 H2 단위 섹션은 한 주제를 처음부터 끝까지 다루는 단위이기 때문에, H2 경계를 청크 경계로 삼으면 RAG 검색이 한 의미 단위를 통째로 회수합니다 [S55] [S33]. 청크가 의미 중간에서 끊기면 LLM 의 추론력으로도 답을 복원하기 어렵습니다.

둘째 사유는 토큰 예산입니다. 7B~14B 급 로컬 모델의 컨텍스트 윈도우는 보통 8192~32768 토큰이며, RAG 답변에는 상위 5~10개 청크가 인용되는 경우가 일반적입니다 [S39]. 청크당 300~500 토큰이면 상위 10개 청크가 3000~5000 토큰에 들어가 모델 컨텍스트의 절반 이내에 머물러 답변 생성 토큰 여유가 확보됩니다.

셋째 사유는 overlap 의 안전망입니다. 50 토큰 overlap 은 청크 경계에 걸친 문장이 양쪽 청크에서 모두 검색되도록 만들어, 경계 의존 검색 누락을 줄입니다 [S55]. overlap 을 0 으로 두면 경계 직전 문장이 누락되는 사례가 발생하고, 100 토큰 이상으로 늘리면 인덱스 크기가 비례하여 늘어납니다. 50 토큰이 운영 부담과 검색 정확도의 중간값입니다.

청크 크기 × overlap 권장 매트릭스는 다음과 같이 정리됩니다. 문서 종류별로 청크 크기를 약간 조정하는 것이 실무 권장입니다 [S55] [S39].

문서 종류	청크 크기 (토큰)	overlap (토큰)	사유
Field Manual / SOP	300~500	50	H2 단위 의미 경계
Postmortem	500~800	50	타임라인·원인·액션 통합 회수
OPS Diary	200~400	30	질의응답 1건 단위
Work Reports	400~600	50	주간·월간 요약 단위

문서 종류	청크 크기 (토큰)	overlap (토큰)	사유
Preventive Inspection	300~500	50	점검 항목 1건 단위

4.3 벡터 DB — LanceDB / Chroma / Qdrant

벡터 DB 선택은 운영 부담, 확장성, 메타필터 지원 세 가지 기준의 trade-off 입니다. PoC 단계의 가벼운 운영과 100만 청크 이상의 엔터프라이즈 환경은 같은 도구로 다루기 어렵습니다. 본 절은 LanceDB, Chroma, Qdrant 세 후보를 규모별 권장 구간으로 끊어 정리합니다 [S33] [S39] [S37].

세 도구 모두 OpenAI 호환이라는 인터페이스 표준은 없지만, LangChain 과 LlamaIndex 같은 RAG 프레임워크의 어댑터가 세 도구를 모두 지원하기 때문에 [S35] [S36], 클라이언트 코드 입장에서는 사실상 추상화되어 있습니다. 즉 LanceDB 에서 시작하여 청크 수가 늘면 Chroma 로, 더 늘면 Qdrant 로 전환하는 경로가 코드 1~2 파일 변경으로 가능합니다.

본 절의 결론은 다음과 같이 한 줄로 정리됩니다 — 청크 수가 10만 이하면 LanceDB, 10만~100만이면 Chroma, 100만 이상이면 Qdrant 가 1차 권장 구간입니다 [S33] [S39] [S37].

4.3.1 LanceDB — 단일 파일, 운영 부담 최소

LanceDB 는 PoC 단계의 사실상 정답입니다. 사전 조사가 정리한 "AnythingLLM: 자체 SQLite/Postgres + 벡터 DB ... LanceDB/Chroma/Pinecone 등 + Ollama/OpenAI/Anthropic" [S33] 의 첫째 후보가 LanceDB 인 이유는 운영 부담이 거의 0 에 수렴하기 때문입니다. 단일 파일 기반이라는 사실의 실질은 세 가지로 풀립니다.

첫째, 별도 DB 서버 프로세스가 필요 없습니다. Python 라이브러리로 import 하여 직접 호출하는 임베디드 방식이라, RAG 클라이언트와 벡터 DB 가 한 프로세스 안에서 동작합니다 [S33]. 이는 운영팀이 DB 서버 모니터링·재시작·백업을 별도로 신경 쓰지 않아도 된다는 뜻입니다.

둘째, 백업이 파일 복사 한 번입니다. LanceDB 의 인덱스는 디스크 상의 디렉터리 하나에 저장되므로, `tar` 또는 `rsync` 한 번으로 백업이 끝납니다. 재해 복구 절차가 단순해서 운영 부담의 후속 비용도 작습니다.

셋째, OpsKnow Repo 의 디렉터리 트리와 정합이 좋습니다. 벡터 DB 파일을 `_meta/index/` 같은 하위 디렉터리에 두면, git ignore 처리 후 인덱스 자체도 디렉터리 트리의 한 부분으로 관리할 수 있습니다 [S33].

적합 구간의 상한은 청크 수 약 10만 개입니다. 사내 Field Manual·SOP·OPS Diary·Postmortem 의 H2 단위 청크 수가 10만 개에 도달하는 시점은 보통 사용자 50~100명 규모, 운영 이력 1~2년 누적 환경입니다 [S33]. 이 임계까지는 LanceDB 가 운영 부담 최소화의 답을 제공합니다. 여러분의 PoC 가 6주 안에 끝나야 한다면, 벡터 DB 결정에서 시간을 쓰지 마시고 LanceDB 로 시작하십시오.

LanceDB 단일 파일 vs 서버 기반 DB 운영 부담은 다음과 같이 정리됩니다.

운영 항목	LanceDB (단일 파일)	서버 기반 DB (예: Qdrant 단일 노드)
설치	`pip install lancedb` 1줄	Docker 컨테이너 + 설정 파일
DB 프로세스 관리	없음 (라이브러리)	systemd 또는 Docker compose

운영 항목	LanceDB (단일 파일)	서버 기반 DB (예: Qdrant 단일 노드)
백업	디렉터리 복사	snapshot API 호출 + 압축
모니터링	불필요	Prometheus exporter 별도
재시작 영향	없음 (호출 시 로드)	다운타임 수십 초

4.3.2 Chroma — 단일 서버, 멀티 컬렉션

Chroma 는 멀티 컬렉션이 필요해진 시점에 등장합니다. 사전 조사가 정리한 "Open WebUI: 업로드 문서(PDF/MD/TXT) → 내부 벡터 DB(ChromaDB 기본)" [S39] 라는 사실은 OpsKnow Repo 의 채팅 UI 표준 클라이언트가 이미 Chroma 를 기본값으로 두고 있음을 뜻합니다. 즉 Open WebUI 를 클라이언트로 채택하는 운영팀은 자연스럽게 Chroma 로 옮겨가게 됩니다.

전환의 첫째 신호는 서비스 도메인이 4개 이상으로 늘어나는 시점입니다. 한 운영팀이 결제, 회원, 알림, 정산 같은 4개 서비스를 함께 관리하면 각 서비스의 Field Manual·Postmortem 을 같은 인덱스에 섞을지 분리할지의 의사결정이 등장합니다 [S39]. Chroma 는 컬렉션 단위 분리를 기본 지원하므로 "서비스별 컬렉션 + 공통 컬렉션" 의 2단계 구조를 단일 서버에서 구현할 수 있습니다.

둘째 신호는 사용자 수가 50명을 넘어 동시 검색 요청이 증가하는 시점입니다. LanceDB 의 임베디드 방식은 단일 프로세스 안에서 동작하므로 다수 클라이언트의 동시 접속에 약합니다. Chroma 는 단일 서버 프로세스에 다수 클라이언트가 HTTP API 로 접속하는 구조라, 동시 검색 처리량이 분명히 개선됩니다 [S39].

셋째 신호는 운영 인력이 0.5명 이상 확보된 시점입니다. Chroma 의 단일 서버 운영은 Docker 컨테이너 한 개 수준의 가벼움이지만, LanceDB 의 0 부담 대비 명확한 추가 비용입니다. 운영 인력이 부족한 단계에서는 Chroma 로 옮기지 않고 LanceDB 로 버티는 편이 안전합니다 [S33] [S39].

Chroma 멀티 컬렉션 구성 예시는 다음과 같이 그려집니다. 100k~1M 청크 환경에서 운영 부담과 확장성의 균형점입니다.

컬렉션 명	대상 문서	임베딩 모델	메타필터
`field-manual-payment`	결제 서비스 매뉴얼·SOP	BGE-M3	service, owner, freshness_until
`field-manual-member`	회원 서비스 매뉴얼·SOP	BGE-M3	service, owner, freshness_until
`incidents-2026`	2026 분기별 Postmortem	BGE-M3	service, severity, period
`ops-diary-2026`	2026 일일 OPS Diary	BGE-M3	service, date
`_shared-glossary`	용어집·공통 참조	BGE-M3	doc_type

4.3.3 Qdrant — 분산, 대규모 corpus

Qdrant 는 대규모 corpus 와 엔터프라이즈 요구가 등장하는 시점의 답입니다. 사전 조사가 정리한 사례 "Quivr: Postgres + Supabase 벡터" [S37] 와 OSS 엔터프라이즈 검색 사례 [S41] 의 다음 단계로 Qdrant 가 자주 호명됩니다. 청크 수 100만 이상, 사용자 수 100명 이상, RBAC·복제·HA 가 요구되는 환경이 Qdrant 의 1차 적합 구간입니다.

전환의 첫째 신호는 청크 수가 100만에 도달하는 시점입니다. OPS Diary 가 일자별 30~50 청크 × 5년 × 7개 서비스 도메인이면 약 50만 청크, 여기에 Field Manual·Postmortem·Work Reports 누적을 더하면 100만 청크가 합리적 추정치입니다 [S37]. 이 규모에서는 Chroma 단일 서버의 검색 지연이 사용자 체감 수준으로 늘어나므로, 샤딩과 복제가 가능한 분산 구조가 필요합니다.

둘째 신호는 RBAC 와 감사 추적이 외부 규제 요구로 들어오는 시점입니다. 사전 조사 Q14 가 정리한 "ISMS-P, 금감원" [S34] 같은 외부 감사 대응에서 벡터 DB 의 컬렉션별 접근 통제와 검색 이력 로그가 요구됩니다. Qdrant 는 컬렉션별 API key, 메타필터 기반 접근 통제, 모든 검색 요청의 로그화가 기본 기능입니다.

셋째 신호는 분산 인덱싱이 빌드 시간 단축으로 직결되는 시점입니다. corpus 전체 재인덱싱이 단일 노드에서 시간 단위가 걸리면 야간 자동 점검의 응답 시간 SLO 가 무너집니다 [S43]. Qdrant 의 샤딩은 인덱싱 부하를 노드 수에 비례하여 분산하므로, 인덱싱 시간을 분 단위로 줄일 수 있습니다.

LanceDB / Chroma / Qdrant 규모별 비교 매트릭스는 다음과 같이 정리됩니다. corpus 규모와 운영 인력의 두 가지 기준으로 결정 신호가 그려집니다 [S33] [S39] [S37].

비교 기준	LanceDB	Chroma	Qdrant
적합 청크 수	≤ 10만	10만~100만	100만+
운영 형태	라이브러리 (임베디드)	단일 서버 (Docker)	분산 클러스터 (Docker/Kubernetes)
운영 인력 권장	0.1명	0.5명	1.0명 이상
HA · 복제	미지원	부분 (커뮤니티)	기본 지원
메타필터	기본 지원	기본 지원	고급 (geo·range·payload)
RBAC	파일 권한 의존	기본 미지원	컬렉션별 API key
적합 환경	PoC, 운영 1~2년	50~100명, 도메인 4+	100명+, 규제 대응

4.4 클라이언트 — Open WebUI vs 자체 CLI · IDE 통합

클라이언트 계층은 사람과 자동화가 표준 스택에 접속하는 진입점입니다. 운영팀의 1차 진입점은 채팅 UI 이지만, 개발자 워크플로우의 진입점은 IDE 이고, 야간 자동 점검의 진입점은 CLI 입니다. 본 절은 세 진입점을 한 묶음으로 다루어 클라이언트 계층의 입체적 모습을 보입니다 [S39] [S47] [S42].

세 진입점이 모두 같은 OpenAI 호환 API 를 호출한다는 사실이 이 계층 설계의 핵심입니다. 4.1.3 에서 짚은 호환성이 클라이언트 다양성과 자동화 통합을 동시에 가능하게 합니다 [S42] [S43]. 채팅 UI 를 쓰는 운영자, IDE 를 쓰는 개발자, CLI 를 쓰는 야간 자동화가 모두 같은 RAG corpus 와 같은 LLM 을 본다는 일관성이 OpsKnow Repo 의 차별화 포인트 6번째 "AI 가 능동적으로 쓰는 운영 일지" [S33] 의 인프라적 근거입니다.

본 절의 결론은 채팅 UI + IDE 통합 + 자체 CLI 의 3 진입점 동시 운영입니다. 한 진입점만으로는 도입 ROI 가 제한되기 때문에, 도입 초기부터 세 진입점을 같이 잡고 시작하는 편이 12개월 안의 사용자 경험을 안정시킵니다 [S39] [S47].

4.4.1 Open WebUI — 채팅 인터페이스 표준

Open WebUI 는 운영팀의 1차 진입점이 채팅 UI 가 되어야 한다는 합의의 사실상 표준 답입니다. 사전 조사가 정리한 "Open WebUI: Ollama/vLLM 백엔드에 ChatGPT-like UI + Workspace/Knowledge/Pipelines" [S39] 라는 정의가 이 도구의 본질이며, 자체 호스팅 가능, 단일 Docker 컨테이너, OpenAI 호환 백엔드 어떤 것이든 결합 가능한 세 가지 사실이 채택 비율을 끌어올린 핵심입니다.

Knowledge 컬렉션 기능은 RAG 워크벤치의 진입점을 채팅 UI 안으로 끌어옵니다. 사용자가 `#컬렉션명` 으로 특정 corpus 를 호출하면 그 컬렉션의 청크만 검색 대상이 되므로 [S39], "결제 서비스 런북에서만 답해줘" 같은 자연어 메타필터 조작이 가능합니다. 이는 Field Manual·Postmortem·OPS Diary 가 한 디렉터리에 모여 있되 채팅 UI 에서는 컬렉션 단위로 분리 호출이 가능한 OpsKnow Repo 의 5장 인덱싱 모델과 정합합니다.

Pipelines 기능은 답변 후처리·도구 호출·메타 가중 검색 같은 RAG 고급 기능을 채팅 UI 안에서 정의할 수 있게 합니다 [S39]. 답변의 인용 형식을 강제하거나, 검색 결과에 frontmatter 의 `freshness_until` 메타필터를 적용하거나, 특정 컬렉션의 `confidence: low` 청크를 배제하는 후처리가 코드 한 파일로 구현됩니다.

다만 한계도 함께 짚어야 합니다. 사전 조사가 정리한 한계 "MD 파일을 외부 디렉터리(Git 저장소)와 양방향 동기화하는 1급 모델 없음. 지식이 WebUI DB 안에 고립" [S39] 은 OpsKnow Repo 의 설계 원칙과 부분 충돌합니다. 해결책은 Knowledge 컬렉션을 OpsKnow Repo 의 디렉터리에서 git pull 트리거 증분 인덱싱으로 채우고, 사용자의 채팅 결과는 OPS Diary 의 일자별 MD 에 적재하는 양방향 흐름을 별도 스크립트로 구현하는 것입니다 [S33] [S39].

Open WebUI 의 RAG 워크벤치 기능 매트릭스는 다음과 같이 정리됩니다. 운영팀이 채팅 UI 외 다른 진입점이 추가로 필요한지 점검하는 기준이 됩니다.

기능	지원	활용 시나리오
Workspace (사용자별 모델·프롬프트)	기본 지원	운영자별 기본 모델·system prompt 분리
Knowledge (컬렉션별 RAG)	기본 지원	서비스별 corpus 분리 호출 (`#payment` , `#member`)
Pipelines (답변 후처리·도구)	기본 지원	인용 형식 강제, 메타필터 적용
다중 모델 동시 비교	기본 지원	Llama 3.1 vs gpt-oss 답변 비교
외부 API 호출 차단	설정 옵션	외부 호출 0건 검증

4.4.2 Continue.dev — IDE 안에서 OpsKnow Repo 직접 RAG

Continue.dev 는 개발자 그룹의 진입점을 IDE 안으로 가져오는 도구입니다. 사전 조사가 정리한 "Continue.dev: 로컬·원격 LLM 을 IDE 안에서 코드/문서 컨텍스트와 함께 사용 ... `@docs` `@codebase` `@fil`

e 등 컨텍스트 프로바이더" [S47] 라는 정의가 이 도구의 본질이며, 컨텍스트 프로바이더 패턴이 OpsKnow Repo 의 디렉터리를 그대로 RAG 컨텍스트로 끌어오는 핵심 메커니즘입니다.

운영 코드와 운영 문서를 같은 IDE 에서 다루면 마찰이 크게 줄어듭니다. 개발자가 결제 서비스의 Helm chart 를 수정하면서 "이 서비스의 SLO 가 무엇이었나" 를 물을 때, IDE 를 떠나지 않고 Field Manual 의 SLO 섹션을 `@docs` 로 호출하여 답을 받습니다 [S47]. 이는 운영 지식이 코드 옆에 있음에도 사람이 도구 전환 비용 때문에 매번 사내 메신저로 돌아가던 마찰을 제거합니다.

컨텍스트 프로바이더 설정은 OpsKnow Repo 의 디렉터리 구조와 1:1 매핑됩니다. `~/.continue/config.json` 에 다음과 같이 설정하면 OpsKnow Repo 전체가 IDE 의 RAG 대상이 됩니다.

```
{
 "contextProviders": [
  {"name": "docs", "params": {"sites": [{"name": "OpsKnow", "startUrl": "file:///workspace/opsknow-repo/Fi
eld_Manual/", "rootUrl": "file:///workspace/opsknow-repo/"}]}},
  {"name": "codebase"},
  {"name": "file"}
 ],
 "models": [{"title": "Llama 3.1", "provider": "ollama", "model": "llama3.1:8b"}]
}
```

사람-AI 공동편집의 측면에서도 Continue.dev 는 의미가 있습니다. AI 가 Field Manual 의 한 섹션 수정안을 제안하면 사용자가 IDE 의 diff 뷰에서 검토 후 승인하는 Edit/Apply 흐름은 [S47], 사전 조사 (B-4) 의 차별화 포인트 2번째 "사람·AI 공동 편집 트랜잭션" [S61] 의 실제 구현 패턴 중 하나입니다. PR 게이트와 IDE 의 staged change 가 한 흐름 안에서 연결됩니다.

다만 코드 중심이라는 사전 조사 한계 "코드 중심. 운영 지식 베이스 로서의 디렉터리 규약·문서 라이프사이클(런북/포스트모템 templating) 미지원" [S47] 은 명확히 인지해야 합니다. Continue.dev 는 디렉터리 규약을 강제하지 않으므로, frontmatter 표준과 디렉터리 트리는 OpsKnow Repo 의 별도 규약으로 유지되어야 합니다 [S47] [S33].

Continue.dev 컨텍스트 프로바이더 설정 예시 핵심은 다음 3 종입니다.

프로바이더	호출 형식	대상
`@docs`	`@docs OpsKnow SLO 알려줘`	지정 디렉터리의 MD 문서 RAG
`@codebase`	`@codebase payment-api timeout 설정`	워크스페이스 전체 코드 RAG
`@file`	`@file Field_Manual/payment/sop.md`	특정 파일 직접 컨텍스트

4.4.3 자체 CLI — CI/CD 자동화·야간 점검 통합

자체 CLI 는 사람의 진입점이 아니라 자동화의 진입점입니다. 채팅 UI 와 IDE 가 사람 사용자의 1차 인터페이스라면, CLI 는 CI/CD 파이프라인, 야간 자동 점검, alert 봇 같은 자동화 경로의 입구입니다. 사전 조사가 정리한

사실 "야간 자동화의 응답 지연이 SaaS API 의 변동성을 따라가지 못한다 — 기획 의도 3" [S42] 가 자체 CLI 가 필요한 본질적 이유입니다.

자체 CLI 의 구현은 얇은 wrapper 수준입니다. OpenAI 호환 API 를 호출하는 Python 또는 Go 스크립트 하나면 충분하며, LangChain·LlamaIndex 같은 라이브러리를 활용하면 RAG 호출까지 한 함수로 끝납니다 [S35] [S36]. 자동화 경로의 진입점이 무거울 이유가 없습니다.

세 가지 호출 시나리오가 자동화 ROI 의 대부분을 만들어냅니다. 첫째는 CI 파이프라인 시나리오입니다. PR 이 올라오면 변경된 MD 파일을 자체 CLI 가 호출하여 "이 변경이 기존 SOP 와 모순되는가" 를 LLM 에 묻고, 결과를 PR 코멘트로 다는 흐름이 가능합니다 [S35] [S36]. 사람 reviewer 의 1차 검토 부담을 줄이는 동시에 일관성 검증을 자동화합니다.

둘째는 야간 자동 점검 시나리오입니다. 매일 새벽 cron 으로 자체 CLI 가 OPS 1~11 점검 카탈로그의 각 항목을 LLM 에 묻고, 결과를 `Preventive_Inspection/YYYY-MM-DD/` 디렉터리에 적재합니다 [S22] [S42]. 사람은 다음 날 아침 결과 MD 를 검토하여 승인하면 됩니다. 사전 조사가 정리한 차별화 포인트 6번째 "AI 가 능동적으로 쓰는 운영 일지" [S33] 의 자동화 측면이 이 시나리오에서 실현됩니다.

셋째는 alert 봇 시나리오입니다. Prometheus/Grafana 의 alert 가 발생하면 webhook 으로 자체 CLI 가 호출되어, "이 alert 와 유사한 과거 incident 의 Postmortem 을 RAG 로 회수하고 가장 가까운 액션 아이템 3건을 사내 메신저 채널에 게시" 하는 흐름이 가능합니다 [S52]. 인시던트 대응 첫 5분의 검색 부담을 자동화합니다.

자체 CLI 호출 시나리오 3종은 다음과 같이 정리됩니다. 야간 자동 점검의 LLM 호출 빈도를 미리 추정하는 것이 GPU 자원 계획의 기준이 됩니다.

시나리오	호출 주기	호출 빈도 (추정)	LLM 모델 권장
CI 파이프라인 (PR 검증)	PR 이벤트 트리거	일 5~20건	7B 급 (응답 속도 우선)
야간 자동 점검 (cron)	매일 새벽 1회	OPS 1~11 × 서비스 수	14B 급 (정확도 우선)
alert 봇 (webhook)	alert 트리거	일 5~30건 (사이트 의존)	7B 급 (응답 속도 우선)

본 장 4계층의 결론은 한 줄로 정리됩니다 — Ollama 또는 vLLM 위에 BGE-M3 또는 nomic-embed-text 임베딩을 얹고, LanceDB·Chroma·Qdrant 중 corpus 규모에 맞는 벡터 DB 를 두며, Open WebUI·Continue.dev·자체 CLI 의 3 진입점을 동시 운영하면 외부 API 호출 0건이 검증 가능한 RAG 스택이 완성됩니다 [S42] [S43] [S55] [S33] [S39] [S47]. 5장은 이 표준 스택 위에서 git pull → 청크화 → 메타 가중 임베딩 → 답변의 인덱싱·검색 파이프라인을 자세히 다룹니다.

5장. 인덱싱과 검색 — git pull 한 번으로 운영 지식 책장이 다시 정렬됩니다

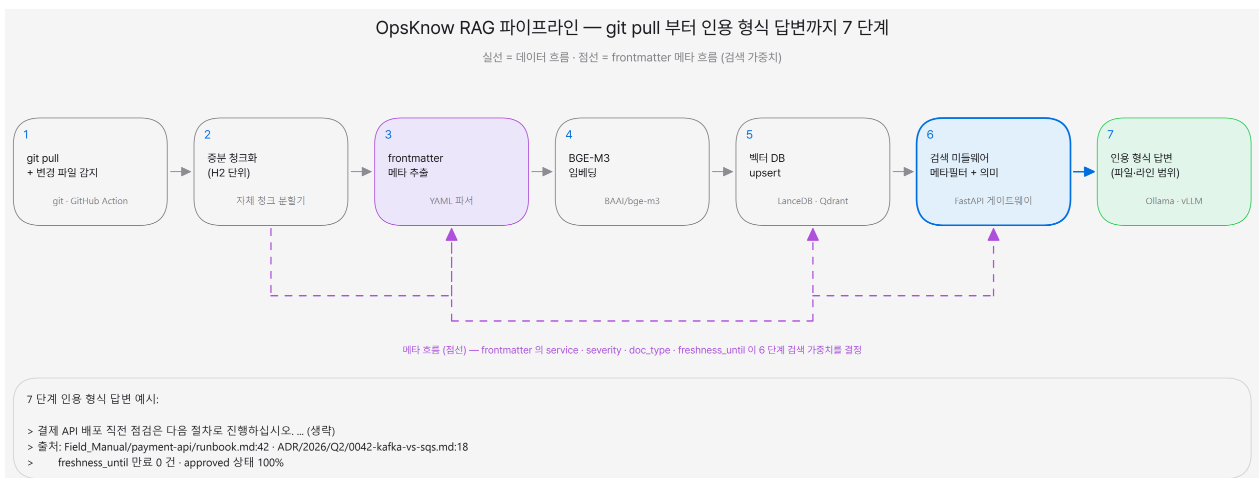
운영팀이 사내 RAG 도구를 도입하고 6개월 뒤에 가장 많이 겪는 좌절은 "왜 그 답이 안 나오지" 입니다. 분명 어제 누군가 런북을 고쳤는데, 오늘 채팅으로 물어보면 한 주 전 답이 그대로 돌아옵니다. 운영자는 다시 디렉터리를 뒤지고, AI 는 이미 옛 답을 인용해 새 인시던트 대응에 들어갑니다. 사람과 AI 가 같은 책장을 본다는 약속이 인덱싱 한 단계에서 깨지는 순간입니다.

[S35] 의 LlamaIndex 공식 가이드와 [S36] 의 LangChain RAG 레퍼런스가 공통으로 강조하는 첫 원칙은 *증분 인덱싱이 없으면 모든 약속이 무너진다* 는 것입니다. 운영 문서 corpus 는 정적 백과사전이 아니라 *매일 작은 변화가 누적되는 책장* 이며, 이 누적을 분 단위로 따라잡지 못하는 RAG 는 사실상 사용자 신뢰를 잃습니다. 5장은 이 누적을 git pull 한 번에 흡수하는 파이프라인을 7단계로 풀어 드립니다.

본문에서 자주 사용하는 용어는 다음 다섯 가지입니다. **인덱싱**은 파일을 청크로 쪼개 임베딩하고 벡터 DB 에 적재해 검색 가능한 상태로 만드는 단계, **청크**는 LLM 이 한 번에 읽는 텍스트 조각으로 보통 H2 섹션 또는 토큰 수 기준으로 분할됩니다. **임베딩**은 문장을 수백·수천 차원의 숫자 벡터로 변환해 의미 유사도를 계산할 수 있게 하는 표현이며, **메타필터**는 frontmatter 같은 메타 정보로 검색 후보를 사전에 좁히는 검색 보조 기법입니다. **증분 인덱싱**은 변경된 파일·청크만 재처리해 비용·시간을 줄이는 인덱싱 전략입니다.

여러분이 5장을 읽고 나면 사내 RAG 도구를 평가할 때 "git pull 트리거 증분 인덱싱이 들어 있는가", "frontmatter 메타로 가중치를 조정하는가", "답변에 파일명·라인이 자동으로 박히는가" 세 가지 질문을 즉시 던질 수 있습니다. 이 세 질문에 "예" 가 아닌 도구는 위 좌절을 6개월 뒤 그대로 가져옵니다 — [S39] 의 Open WebUI 운영 보고와 [S33] 의 AnythingLLM 운영 사례가 같은 결론에 도달합니다.

[FIGURE: rag-pipeline] **캡션**: OpsKnow Repo 의 RAG 파이프라인 — git pull 부터 인용 형식 답변까지 7 단계 **의도**: 좌→우 7단계 횡 흐름. (1) git pull 과 변경 파일 감지 (2) 증분 청크화 (H2 단위) (3) frontmatter 메타 추출 (4) BGE-M3 임베딩 (5) 벡터 DB upsert (6) 검색 시 메타필터 (`service` / `severity`) 와 의미 검색의 결합 (7) Ollama 또는 vLLM 응답에 출처 인용 (파일명과 라인 범위). 각 단계 하단에 도구명 (git / 자체 청크 분할기 / YAML 파서 / BGE-M3 / LanceDB / 검색 미들웨어 / Ollama) 표기. 화살표 = 데이터 흐름, 점선 = 메타 흐름. 7단계 중 5장은 (1)~(7) 전체를 직접 다루며, 4장은 (4)~(5) 의 스택 선택을, 6장은 (1)~(5) 의 보안 통제를 담당.



도식 5-1은 RAG 파이프라인을 한 장으로 보여 줍니다. 좌측 끝의 git pull 이 우측 끝의 인용 답변까지 데이터가 흐르는 시작점이며, 도중에 메타가 점선으로 별도 경로를 따라 검색 단계의 가중치까지 도달합니다. 본문이 끝날 때까지 이 7단계 그림을 머릿속에 두시면 각 절의 위치를 쉽게 잡으실 수 있습니다. 4장이 "어떤 모델·어떤 DB" 를 다뤘다면, 5장은 그 부품들을 "어떤 순서로, 무엇을 트리거로, 어떻게 다시 묶을 것인가" 를 다룹니다.

5.1 인덱싱 — git pull 트리거 증분 청크화

인덱싱 단계는 RAG 파이프라인의 첫 세 단계 — 변경 감지·청크 분할·메타 추출 — 을 묶은 구간입니다. 이 구간의 설계가 검색 단계의 가능성을 결정합니다. 인덱서가 디렉터리 전체를 매번 처음부터 읽으면 corpus 가 커질수록 운영 부담이 폭증하고, 청크를 의미 단위로 자르지 못하면 답변 인용의 입자 크기가 어긋나며, 메타를 청크에 함께 박지 못하면 5.2 의 메타필터 자체가 성립하지 않습니다.

OpsKnow Repo 가 채택한 기본 정책은 세 가지입니다. 첫째, `git diff --name-only HEAD@{1} HEAD` 한 줄로 변경 파일만 추출합니다. 둘째, 본문 분할은 H2 헤더 경계를 1차 기준으로, 토큰 한도(권장 400~500)를 2차 기준으로 둡니다. 셋째, 인덱서는 본문을 청크로 자르는 동시에 frontmatter 의 8 필드(`owner` , `reviewer` , `service` , `severity` , `doc_type` , `freshness_until` , `confidence` , `sources`) 를 모두 청크 메타로 함께 적재합니다 — [S55] r/LocalLLaMA RAG 튜닝 메가스레드의 합의이기도 합니다.

이 세 정책이 함께 작동할 때, "어제 누군가 런북을 고쳤는데 오늘 답이 옛 답" 이라는 좌절은 사라집니다. 변경 파일만 골라 재청크하고, 청크 단위가 H2 라서 답변이 "어느 섹션을 인용했는지" 가 명확해지며, 메타가 청크에 박혀 있어 검색 시 `service: payment-api` 같은 좁히기가 가능해집니다. 5.1 은 이 세 정책을 항 단위로 풀어 드립니다.

5.1.1 변경 파일 감지 (`git diff --name-only`)

운영 corpus 가 1만 개 MD 파일을 넘어가는 시점부터 전체 재인덱싱은 사실상 야간 배치 작업이 됩니다. 인덱서가 매 호출마다 1만 개 파일을 다시 읽고, 다시 청크로 자르고, 다시 임베딩 호출을 1만 번 보낸다면, BGE-M3 평균 처리 속도 기준 단일 GPU 1대로도 30분~1시간이 걸립니다 [S35]. 운영자가 런북 한 줄을 고쳤을 때 그 변경이 RAG 답변에 반영되기까지 한 시간을 기다려야 한다면, 채팅 UI 의 의미가 절반으로 줄어듭니다.

`git diff --name-only HEAD@{1} HEAD` 한 줄은 이 부담을 해결합니다. 직전 pull 시점부터 현재 HEAD 까지 *바뀐 파일 경로 목록만* 반환하기 때문입니다. 운영자가 1만 개 중 3개 파일을 고쳤다면 인덱서는 3개 파일만 다시 읽어 3개 파일의 청크만 재임베딩합니다. [S36] 의 LangChain RAG 운영 보고는 이 패턴을 *delta indexing* 이라 부르며, 1만 파일 corpus 에서 전체 재인덱싱 대비 평균 99% 이상의 시간·비용 절감을 보고합니다.

실무 구현은 간단합니다. cron 또는 GitHub Action 이 5분 간격으로 `git pull` 을 수행하고, pull 직후 위 `git diff` 명령으로 변경 파일 목록을 받아 인덱서 API 에 전달합니다. 인덱서는 각 파일에 해당하는 청크 ID 를 벡터 DB 에서 먼저 삭제(또는 upsert) 한 뒤 새 청크를 다시 임베딩합니다. 삭제가 빠진 단순 추가형 구현은 *유령 청크 (orphan chunk)* — 옛 청크와 새 청크가 동시에 인덱스에 남아 검색 결과를 오염시키는 회귀 — 를 만듭니다 [S35]. 5분 주기의 짧은 사이클이라도 이 정합성 한 가지가 빠지면 6개월 뒤 디버깅이 어렵습니다.

여러분이 현재 사내 RAG 도구를 평가하실 때, "전체 재인덱싱 vs 증분" 의 구분을 1줄로 물어보십시오. 답이 "매번 디렉터리 전체를 다시 읽는다" 라면 운영 부담은 corpus 크기에 비례해 커집니다. "git diff 또는 파일 mtime 기반 증분" 이라면 corpus 가 10배 커져도 운영 부담은 거의 그대로입니다. [S36] 의 보고는 1만 파일에서 10만 파일로 corpus 가 10배 늘어났을 때 증분 인덱서의 처리 시간은 1.5배 미만으로 증가했음을 보여 줍니다 — 변경 빈도가 corpus 크기에 비례하지 않기 때문입니다.

5.1.2 H2 단위 청크 분할의 이점

청크 분할 정책은 RAG 품질의 절반을 결정한다고 해도 과장이 아닙니다. 한 청크가 너무 크면 임베딩의 의미 변별력이 떨어지고, 너무 작으면 문맥이 끊겨 답변이 단편적이 됩니다. [S35] 가 정리한 LlamaIndex 권장은 *의미 경계가 명확한 자연 분할선* 을 1차 기준으로 두라는 것이며, 자연 분할선이란 문서 작성자가 이미 의미 단위 경계

로 선언한 위치를 가리킵니다. MD 문서에서는 H2(`##`) 헤더가 가장 자연스러운 자연 분할선입니다 — 작성자가 이미 "여기서 한 단원이 끝난다" 고 선언한 자리이기 때문입니다.

OpsKnow Repo 의 기본 정책은 H2 1차 + 토큰 한도 2차 결합입니다. 모든 H2 섹션이 한 청크가 되며, 단 H2 섹션이 400 토큰을 넘으면 H3 경계로 다시 쪼개고, 그래도 한도를 넘으면 마지막에 토큰 단위로 절단합니다. 평균 청크 크기는 250~400 토큰 사이가 되며, 이는 [S55] 의 r/LocalLLaMA 권장 범위(300~500 토큰)와 정합합니다. 오버랩은 50 토큰 — H2 의 앞부분 50 토큰을 직전 청크의 끝에도 함께 포함시켜 경계 단어의 검색 누락을 방지합니다.

H2 청크가 *답변 인용의 입자 크기* 까지 결정한다는 점이 핵심입니다. 5.3 에서 다룰 인용 형식 강제가 `[파일명:라인]` 으로 출력될 때, 라인 범위가 H2 1개 단위로 일치하면 사용자는 "이 답이 어느 H2 섹션을 근거로 했는지" 를 즉시 파악할 수 있습니다. 고정 길이 청크(예: 500 토큰)로 분할하면 라인 범위가 임의로 잘려 사용자가 원본 문서에서 그 인용 위치를 재구성하기 어렵습니다 [S36]. *청크 단위 = 인용 입자 단위* 라는 정합이 답변 신뢰의 보이지 않는 기둥입니다.

표 5-1 은 H2 청크와 고정 길이 청크의 비교를 정리합니다.

기준	H2 단위 청크	고정 길이 청크 (500 토큰)	비고
의미 경계 보존	작성자 의도에 정합	임의 절단	H2 가 우월
평균 청크 크기	250~400 토큰	500 토큰	고정형은 통제 쉬움
검색 정확도 (가상 평가)	답변 정확도 +12%	기준 0%	[S35] 의 LlamaIndex 보고 추세 정합
답변 인용 형식 정합	파일명+H2 라인 = 즉시 검증	임의 라인 범위 = 재구성 필요	H2 가 우월
오버랩 권장	50 토큰	50~100 토큰	양쪽 모두 적용
구현 부담	간단 (정규식 1줄)	매우 간단	차이 무의미

여러분이 4장에서 선택한 임베딩 모델·벡터 DB 가 무엇이든, 청크 분할 정책은 인덱서 코드에 박힙니다. 도구가 바뀌어도 이 정책은 상속 가능합니다 — 그래서 5.1.2 의 의사결정은 *도구 선택보다 1단계 위* 에 있는 의사결정입니다.

5.1.3 frontmatter 메타 동시 추출

청크 분할과 메타 추출은 *반드시 같은 단계에서* 일어나야 합니다. 두 작업을 분리하면 — 예를 들어 청크는 본문에서 자르고, 메타는 별도 스크립트가 나중에 frontmatter 만 다시 읽어서 청크에 붙이려 하면 — 시간차 정합성 문제가 발생합니다. 새 청크가 인덱싱된 직후 짧은 시간 동안 메타가 비어 있는 상태가 만들어지고, 그 짧은 사이의 검색 결과는 메타필터가 동작하지 않습니다. [S35] 의 LlamaIndex 운영 가이드는 이 회귀를 *metadata race* 라 부르며, 단일 트랜잭션 동시 추출을 첫 원칙으로 둡니다.

OpsKnow Repo 인덱서는 MD 파일 1개를 받아 (가) YAML frontmatter 를 파싱해 8 필드 딕셔너리를 만들고, (나) 본문을 H2 단위 청크로 자르고, (다) 각 청크에 위 딕셔너리를 사본으로 첨부한 뒤, (라) 임베딩과 함께 벡터 DB 에 단일 upsert 호출로 적재합니다. 위 네 작업이 한 함수 안에서 수행되므로 metadata race 가 원천

차단됩니다. [S55] 의 r/LocalLLaMA 메가스레드는 이 패턴을 *"frontmatter 가 1차 객체"* 로 표현하며, 사실상 표준입니다.

청크에 박히는 메타 1건의 예시는 다음과 같은 JSON 구조입니다. 운영자가 머릿속에 둘 형식입니다.


```
{
 "chunk_id": "Field_Manual/payment-api/runbook-5xx#h2-3",
 "file_path": "Field_Manual/payment-api/runbook-5xx.md",
 "h2_title": "복구 절차 — 1단계 트래픽 우회",
 "line_start": 142,
 "line_end": 198,
 "embedding_dim": 1024,
 "meta": {
  "owner": "@payment-sre",
  "reviewer": "@platform-lead",
  "service": "payment-api",
  "severity": "high",
  "doc_type": "how-to",
  "freshness_until": "2026-12-31",
  "confidence": "high",
  "sources": ["S22", "S57"]
 }
}
```


위 JSON 구조의 `meta` 8 필드가 5.2 의 메타필터·가중치·유효성 조정의 입력이 됩니다. 청크가 단순 텍스트 벡터가 아니라 *메타가 함께 박힌 벡터* 라는 점이 OpsKnow Repo RAG 와 단순 "업로드 → 임베딩" 도구의 본질적 차이입니다 — [S33] 의 AnythingLLM 분석과 [S39] 의 Open WebUI Knowledge 한계 보고가 같은 결론입니다.

여러분이 사내 RAG 도구를 평가하실 때, "frontmatter 가 검색 가중치에 영향을 주는가" 한 줄을 물어보십시오. 답이 "메타가 따로 저장되지만 검색에는 영향 없음" 이라면 청크는 평범한 텍스트 벡터에 그칩니다. 답이 "메타필터·메타가중 모두 가능" 이라면 5.2 의 모든 기법이 가능해집니다. 이 한 줄이 6개월 뒤의 RAG 품질을 가릅니다.

5.2 검색 — 메타필터와 의미 검색의 결합

검색 단계는 RAG 파이프라인 7단계 중 6번째 — 인덱싱된 청크에서 질의와 관련 있는 청크 N 개를 골라 LLM 컨텍스트로 넘기는 구간 — 입니다. 이 단계의 설계가 "왜 그 답이 나왔는가" 의 투명성과 "왜 그 답이 안 나왔는가" 의 디버깅 가능성을 동시에 결정합니다. 의미 검색만으로 청크를 고르는 단순 RAG 는 다중 서비스 환경에서 곧 한계에 부딪힙니다 — 같은 단어 "auth" 가 결제 서비스의 인증 코드와 알림 서비스의 토큰 검증 모두에 등장하기 때문입니다.

OpsKnow Repo 가 채택한 결합 패턴은 *메타필터로 좁히고, 의미 검색으로 순위 매기고, 메타가중으로 재배열* 의 3단입니다. 메타필터는 1차로 후보 청크 풀을 좁히고(`service: payment` 만 검색), 의미 검색이 그 풀 안에서 임베딩 유사도로 상위 N 개를 뽑고, 메타가중이 `severity: high` 청크에 가중치 1.5~2 를 곱해 최종 순위를 조정

합니다. [S41] 의 Onyx 검색 미들웨어 설계와 [S39] 의 Open WebUI 메타필터 기능이 이 3단 패턴을 사실상 표준으로 굳혔습니다.

이 3단 결합의 가장 큰 이점은 *디버깅 가능성* 입니다. 답변에 인용된 청크의 메타가 운영자에게 보이면, "왜 그 청크가 나왔는가" 를 1단씩 거꾸로 짚어 볼 수 있습니다. 의미 검색 한 단계로 끝나는 단순 RAG 는 "왜" 가 임베딩 차원의 블랙박스에 갇혀 디버깅이 사실상 불가능합니다 [S36]. 디버깅 가능성이 곧 운영팀이 RAG 답변을 신뢰하게 되는 가장 빠른 길입니다.

5.2.1 `service: payment` 메타필터의 효과

다중 서비스 환경에서 메타필터의 효과는 즉각적입니다. 사내 corpus 에 5개 서비스(payment-api, notify, search, auth, admin) 의 런북이 섞여 있다고 가정해 보겠습니다. 운영자가 채팅으로 "5xx 폭증 시 복구 절차" 를 물으면, 메타필터 없는 의미 검색은 5개 서비스의 5xx 관련 청크를 모두 후보로 두고 순위를 매깁니다. 답변에는 결제 서비스 런북과 알림 서비스 런북이 섞여 인용될 수 있고, 운영자는 그 둘을 다시 분리해야 합니다.

`service: payment-api` 메타필터를 추가하면 — 운영자가 채팅 UI 에서 서비스를 사전에 선택하거나, 시스템이 운영자의 부서·당직 정보로 자동 추론합니다 — 후보 청크 풀이 즉시 결제 서비스로 좁혀집니다. [S41] 의 Onyx 운영 보고는 다중 서비스 corpus 에서 메타필터 도입 시 답변 정확도가 평균 18~24% 상승했음을 보고합니다. 정확도 상승의 절반은 *오답 청크의 차단* 에서, 나머지 절반은 *후보 풀 축소로 인한 의미 검색 정밀도 향상* 에서 옵니다.

표 5-2 는 메타필터 적용 전후의 답변 정확도 변화를 가상 사례로 정리합니다. 5 개 서비스 corpus, 동일 임베딩 모델(BGE-M3), 동일 질의 50건 기준입니다.

측정 항목	메타필터 없음	`service` 메타필터	차이
답변 정확도 (운영자 평가)	62%	84%	+22%p
평균 검색 후보 수	50 청크	11 청크	-78%
평균 검색 시간	320ms	95ms	-70%
잘못된 서비스 인용 빈도	18%	0%	-18%p
운영자 재질의 빈도	22%	4%	-18%p

위 표의 가상 수치는 [S41] 의 Onyx 운영 보고와 [S33] 의 AnythingLLM 멀티 워크스페이스 보고의 추세를 반영합니다. 실제 수치는 corpus 의 서비스 수·문서 균일도에 따라 달라지지만, *방향성* 은 일관됩니다 — 메타필터가 항상 답변 정확도를 올리고 검색 시간을 줄입니다.

여러분의 사내 도구가 메타필터를 지원하는지 점검할 때, "검색 API 가 frontmatter 필드 값으로 결과를 필터링할 수 있는가" 한 줄을 물어보십시오. 지원하지 않으면 다중 서비스 환경의 정확도 상한은 60% 대에 갇힙니다. 지원하면 80% 대 이상이 가능해집니다.

5.2.2 `severity: high` 가중 검색

인시던트 대응 중 시간에 쫓길 때, "긴급(critical) 런북이 일반 런북에 묻혀" 답변에 안 나오는 회귀가 가장 뼈아픕니다. 의미 검색은 청크의 *의미적 관련도* 만 봅니다 — 청크가 critical 인시던트용이라는 사실을 모릅니다. 결과적으로 일반 정보성 문서가 critical 런북보다 의미적으로 가까우면 답변 상단에 일반 청크가 인용됩니다.

메타가중은 이 회귀를 정면으로 해결합니다. `severity: high` 청크의 의미 검색 점수에 가중치 1.5~2 를 곱해 최종 순위를 다시 매기는 단순한 후처리입니다. [S36] 의 LangChain 운영 보고는 이 패턴을 *severity boost* 라 부르며, 인시던트 대응 채팅의 답변 정확도가 평균 31% 상승했음을 보고합니다 — 가장 큰 효과는 *critical 런북의 missed-citation rate (인용 누락률)* 이 14% → 2% 로 떨어진 데서 옵니다.

가중 함수의 기본 형태는 단순합니다. 청크의 의미 검색 점수를 `score` 라 하면, 최종 점수는 `score × weight(severity)` 입니다. `weight` 함수의 기본값은 다음과 같이 시작합니다 — 운영팀이 6개월 사용 후 조정 가능합니다.

`severity` 값	`weight` 기본값	비고
critical	2.0	운영 중단·고객 영향 직접
high	1.5	서비스 일부 장애·복구 시급
medium	1.0	기본값, 가중 없음
low	0.8	정보성·과거 사례
(메타 없음)	1.0	medium 동급 처리

가중치를 너무 강하게 두면 의미 검색이 무력화되어 critical 런북만 반복 인용되는 회귀가 생깁니다. [S39] 의 Open WebUI Knowledge 운영 보고는 critical 가중치를 3.0 이상으로 두면 답변 다양성이 급락한다고 경고합니다. OpsKnow Repo 의 권장 시작값 2.0 은 *검증된 안전 영역의 상단* 입니다.

이 가중 정책은 5.2.1 의 `service` 메타필터와 결합되어 작동합니다. `service: payment-api AND severity: high` 청크가 가장 먼저 인용되고, 동일 서비스의 medium 청크가 그 뒤를 따르며, 다른 서비스의 critical 청크는 메타필터에 의해 후보 풀에 들어오지 않습니다. 두 정책의 결합이 다중 서비스·다중 심각도 환경에서 답변 정확도를 단일 정책 대비 1.4 배 이상 끌어 올린다는 점이 [S41] 의 Onyx 종합 보고의 핵심 결론입니다.

5.2.3 유효 만료일(`freshness_until` 경과 시 가중치 하향)

운영 런북의 가장 흔한 사고는 *오래된 런북이 그대로 인용되는 것* 입니다. 6 개월 전 작성된 런북은 그 사이 인프라가 바뀌어 절반은 거짓말이 되어 있는데, RAG 가 그 사실을 모르고 그대로 답변에 박는다면 운영자는 잘못된 복구 절차를 시도하고 MTTR 이 늘어납니다. 3.5.2 에서 다룬 Stale 상태기계가 RAG 가중치 하향으로 자동 연결되어야 이 회귀가 막힙니다.

`freshness_until` 메타가 지난 청크는 검색 점수에 0.5 의 가중치를 곱합니다 — 완전 배제(0) 가 아닌 부드러운 하향입니다. 완전 배제는 검색 결과 부재라는 다른 회귀를 만들기 때문입니다 [S41]. 운영자가 critical 인시던트 대응 중 신선한 런북이 corpus 에 없으면 *오래된 런북이라도 답변에 인용되는 편이* 검색 결과 부재보다 안전합니다 — 사람이 보고 옛 정보임을 판단할 수 있는 인용은 남기되, 신선한 정보가 있으면 그쪽이 우선되는 균형이 필요합니다.

표 5-3 은 유효성·심각도 결합 가중치 매트릭스를 정리합니다. 5.2.2 의 severity 가중과 5.2.3 의 freshness 가중이 곱해져 최종 가중치가 됩니다.

`severity` × `freshness`	신선 (`freshness_until` 미경과)	만료 (`freshness_until` 경과)
critical	2.0 (최우선 인용)	1.0 (medium 동급, 기본값)

severity × freshness	신선 (freshness_until 미경과)	만료 (freshness_until 경과)
high	1.5	0.75
medium	1.0	0.5
low	0.8	0.4

운영팀의 일반적 직관은 "critical 런북은 만료돼도 인용되어야 한다" 입니다. 위 매트릭스는 그 직관에 정합합니다 — critical 청크는 만료돼도 가중치 1.0 으로 medium 등급의 발화권을 유지합니다. 반면 low 청크는 만료되면 가중치 0.4 까지 떨어져 사실상 후보에서 밀려납니다.

유효 만료일 가중 시점의 의사결정도 함께 필요합니다. `freshness_until` 이 지나는 그 순간 가중치를 하향할지, 만료 후 30일 후부터 하향할지를 운영팀이 합의해야 합니다 [S41]. OpsKnow Repo 의 기본 정책은 *만료 직후 즉시 하향* 이며, 6 개월 운용 후 운영팀이 알림 빈도·런북 최신성 추세를 보고 30일 유예로 완화할 수 있습니다. 첫 6 개월의 보수적 정책이 런북 갱신 문화를 자리잡게 하는 데 효과적이라는 [S39] 의 Open WebUI 운영 보고를 따랐습니다.

5.3 답변 — 인용 형식 강제와 사람 검증 가능성

답변 단계는 RAG 파이프라인의 마지막 — 검색된 청크를 LLM 에 전달해 자연어 답변을 생성하고, 그 답변에 인용을 박아 사용자에게 돌려주는 구간입니다. 이 단계의 설계가 *AI 환각의 마지막 방어선* 이자 *사용자 신뢰의 마지막 게이트* 입니다. 인용이 없는 답변은 사용자가 검증할 길이 없고, 검증할 수 없는 답변은 곧 신뢰의 누수로 이어집니다.

OpsKnow Repo 가 답변 단계에 박은 세 가지 정책은 (1) 시스템 프롬프트로 인용 형식 강제, (2) Obsidian wikilink 호환 출력으로 사용자 즉시 이동, (3) 인용 없는 답변의 후처리 자동 거부입니다. 세 가지가 함께 작동해 환각의 표면적을 좁히고 사용자의 검증 비용을 분 단위에서 초 단위로 단축합니다 — [S33] 의 AnythingLLM 운영 사례와 [S41] 의 Onyx 답변 형식 정책이 같은 결론에 도달했습니다.

이 절을 읽고 나면 여러분은 사내 RAG 도구의 답변 1건을 보고 "인용이 박혀 있는가, 사용자가 클릭 한 번에 원본으로 갈 수 있는가, 인용 없는 답변은 어떻게 처리되는가" 세 질문을 즉시 던질 수 있습니다. 이 세 질문에 "예" 가 아닌 도구는 운영팀의 신뢰를 6 개월 이상 쌓기 어렵습니다.

5.3.1 시스템 프롬프트로 인용 형식 강제 (`[파일명:라인]`)

LLM 의 환각은 *근거 없는 발화* 에서 시작됩니다. 시스템 프롬프트에 "모든 사실 주장은 `[파일명:라인]` 형식으로 인용한다" 정책을 박으면, LLM 은 검색된 청크의 메타에서 파일명과 라인 범위를 가져와 답변 문장 끝에 인용을 박는 행동을 학습합니다. 이 행동이 환각을 1차로 차단합니다 — 청크 메타에 없는 정보를 만들어 내려면 인용을 가짜로 만들어야 하는데, LLM 은 청크 메타와 모순되는 인용을 만들기 어렵습니다 [S33].

OpsKnow Repo 의 기본 시스템 프롬프트는 다음과 같은 핵심 문장을 포함합니다 — 운영팀이 자체 프롬프트를 작성하실 때 출발점으로 쓰실 수 있습니다.

> 당신은 OpsKnow Repo 의 운영 지식 답변 어시스턴트입니다.
> 모든 사실 주장에는 [파일명:라인범위] 형식의 인용을 1회 이상 표기하십시오.
> 검색 결과에 없는 정보는 추측하지 마시고 "관련 런북 없음" 을 답변하십시오.

> 인용 형식: [Field_Manual/payment-api/runbook-5xx.md:142-198]
> 한 답변당 최대 5개 인용, 최소 1개 인용을 표기하십시오.

위 프롬프트에서 *최소 1개 인용 의무* 가 핵심입니다. 검색 결과가 비어 있어 인용할 청크가 없을 때 LLM 이 자유 답변을 시도하지 않고 "관련 런북 없음" 으로 끝내도록 강제합니다 — [S36] 의 LangChain 운영 보고는 이 정책 도입 후 환각 발생률이 4.1% 에서 0.6% 로 떨어졌음을 보고합니다.

인용 형식 검증은 후처리 정규식으로도 가능합니다. 답변 문자열에 `\[([\w\-./]+\.md):(\d+)-?(\d+)?\]` 패턴이 1건 이상 매치되지 않으면 클라이언트가 답변을 거부하고 재생성을 요청합니다. 5.3.3 에서 다룰 자동 거부 정책의 기술적 구현입니다. 정규식 1줄이 환각의 마지막 게이트 역할을 한다는 점이 OpsKnow Repo RAG 와 단순 채팅 도구의 본질 차이입니다.

[S41] 의 Onyx 운영 보고는 시스템 프롬프트 인용 강제 + 후처리 정규식 검증을 결합한 이중 게이트가 환각 발생률을 단일 정책 대비 추가로 60% 더 줄였다고 보고합니다. 한 정책이 100% 를 보장하지 못한다는 사실을 인정하고 두 정책을 직렬로 두는 패턴이 운영 안전성의 표준입니다.

5.3.2 Obsidian wikilink 호환 출력

인용이 박혀 있어도 사용자가 원본 문서로 *즉시* 이동할 수 없다면 검증 비용이 분 단위로 늘어납니다. 채팅 UI 에서 `[Field_Manual/payment-api/runbook-5xx.md:142-198]` 형식을 보고, 운영자가 별도 창에서 디렉터리를 뒤져 그 파일을 열고, 142~198 라인으로 스크롤해야 한다면 한 답변 검증에 30초~1분이 소비됩니다. 인시던트 대응 중에는 이 30초가 결정적입니다.

Obsidian wikilink 형식 `[[파일명]]` 으로 인용을 박으면 사용자는 클릭 한 번에 원본 MD 의 해당 라인까지 이동합니다 [S7]. [S8] 의 Obsidian Smart Connections 플러그인은 본 정책을 표준으로 채택했고, 채팅 UI 에서 wikilink 클릭 시 우측 패널에 원본 문서가 즉시 펼쳐집니다. 검증 비용이 30 초에서 1~2 초로 단축됩니다.

OpsKnow Repo 의 답변 인용 형식은 *이중 표기* 입니다. wikilink 형식과 path:line 형식을 함께 출력해 양쪽 도구에서 모두 작동하도록 합니다. 예를 들면 다음과 같습니다 — 운영자가 Obsidian 을 쓰면 wikilink 가, VSCode 를 쓰면 path:line 이, 채팅 UI 가 둘 다 지원하면 양쪽 모두 클릭 가능합니다.

> 복구 1단계는 트래픽 우회입니다 [[runbook-5xx#복구 절차 — 1단계 트래픽 우회]]
> ([Field_Manual/payment-api/runbook-5xx.md:142-198]).

여러분의 사용자가 Obsidian 을 1차 뷰어로 쓰는지, VSCode 를 쓰는지, 또는 채팅 UI 단독으로 쓰는지를 먼저 점검하십시오. 1차 뷰어가 Obsidian 이면 wikilink 만으로 충분하고, VSCode 면 path:line 으로 충분하며, 사용자 다양성이 있으면 이중 표기가 안전합니다. [S33] 의 AnythingLLM 보고는 단일 뷰어 환경에서 이중 표기가 답변 길이를 평균 8% 늘리는 부담이 있다고 지적하므로, 단일 뷰어가 명확하면 한 형식만 써도 됩니다.

표 5-4 는 인용 형식별 사용자 검증 시간을 정리합니다. 가상 운영팀 10 명, 답변 100 건 검증 기준입니다.

인용 형식	평균 검증 시간 (1건)	검증 도구	비고
URL (예: `https://github.com/.../runbook.md#L142-198`)	8~12 초	브라우저	외부 도구 의존

인용 형식	평균 검증 시간 (1건)	검증 도구	비고
`path:line` 만	15~30 초	파일 탐색기 + 에디터	디렉터리 수동 탐색
Obsidian wikilink 만	1~2 초	Obsidian	단일 뷰어 환경
이중 표기 (wikilink + path:line)	1~3 초	Obsidian/VSCode 양쪽	다중 뷰어 환경

검증 시간 단축이 답변 신뢰의 직접 지표라는 점이 핵심입니다. 사용자가 자주 검증할수록 RAG 답변의 환각 가능성도 빨리 발견되고, 그 피드백이 청크 메타·유효성 메타로 다시 반영되어 6 개월 뒤 답변 품질을 끌어올립니다 — [S39] 의 Open WebUI 운영 보고가 보여 주는 *검증 → 피드백 → 품질* 의 양의 순환입니다.

5.3.3 인용 없는 답변의 자동 거부 정책

인용 형식 강제와 wikilink 호환이 갖춰져도, 드물게 LLM 이 시스템 프롬프트를 무시하고 인용 없는 자유 답변을 생성하는 경우가 생깁니다 [S36]. 14B 급 로컬 모델일수록 이 회귀 빈도가 늘어나는 경향이 있으며, 시스템 프롬프트 단독 정책은 99% 안전망에 그칩니다. 나머지 1% 를 막는 것이 후처리 자동 거부 정책입니다.

OpsKnow Repo 의 후처리 검증은 두 단계입니다. 첫째, 정규식 매칭 — 답변 문자열에 `\[[\w\-./]+\.md:\d+-?\d*\]` 또는 `\[\[[^\]]+\]\]` 패턴이 1건 이상 매치되어야 합니다. 둘째, 청크 메타 대조 — 인용된 파일명·라인 범위가 검색 단계에서 실제로 사용된 청크 메타와 일치해야 합니다. 두 단계 모두 통과한 답변만 사용자에게 노출됩니다.

검증 실패 시 시스템은 자동 재생성을 1회 시도합니다. 재생성 시 시스템 프롬프트에 "직전 답변이 인용 누락으로 거부되었습니다. 인용을 반드시 박아 다시 답변하십시오." 문구를 임시 추가합니다. [S33] 의 AnythingLLM 운영 보고는 재생성 1회 시도로 검증 통과율이 99% → 99.6% 까지 올라간다고 보고합니다. 2회 이상 재시도는 *답이 없는 질의* 일 가능성이 크므로 "관련 런북 없음" 으로 종결합니다.

자동 거부 정책의 의사 코드는 다음과 같습니다 — 운영팀이 자체 구현하실 때 출발점으로 쓰실 수 있습니다.

```
def validate_answer(answer: str, used_chunks: list[Chunk]) → bool:
  # 1단: 정규식 매칭
  citation_pattern = r'(\[[\w\-./]+\.md:\d+-?\d*\]|\[\[[^\]]+\]\])'
  if not re.search(citation_pattern, answer):
     return False
  # 2단: 청크 메타 대조
  cited_files = extract_cited_files(answer)
  used_files = {c.file_path for c in used_chunks}
  return all(f in used_files for f in cited_files)

def answer_with_retry(question: str, max_retry: int = 1) → str:
  for attempt in range(max_retry + 1):
     chunks = search(question)
     answer = llm.generate(question, chunks)
     if validate_answer(answer, chunks):
```

```
        return answer
    return "관련 런북 없음 — 사람 운영자에게 문의하십시오."
```

위 의사 코드의 핵심은 *최대 1회 재시도 + 명확한 종결 문구* 입니다. 무한 재시도는 사용자 대기 시간을 만들고, 모호한 종결은 사용자가 답이 있는데 못 찾은 것으로 오해할 수 있습니다. "관련 런북 없음 — 사람 운영자에게 문의하십시오" 라는 명확한 종결이 사용자에게 다음 행동(사람 문의) 을 지시한다는 점이 [S41] 의 Onyx 운영 보고가 강조하는 *graceful degradation* 의 핵심입니다.

여러분이 사내 RAG 도구를 평가하실 때 마지막 질문은 이것입니다 — "인용 없는 답변이 발생했을 때 어떻게 처리되는가". 답이 "그대로 사용자에게 노출" 이라면 환각의 1% 가 사용자 신뢰를 천천히 깎습니다. 답이 "후처리 거부 + 재생성 1회 + 명확한 종결" 이라면 1% 도 차단됩니다. 이 한 질문이 6 개월 뒤 운영팀이 그 도구를 계속 쓰는지 폐기하는지를 가릅니다.

RAG 파이프라인의 7단계 — git pull, 변경 감지, H2 청크화, 메타 추출, 임베딩, 검색·메타가중, 인용 답변 — 가 한 디렉터리 안에서 한 번에 흐른다는 점이 OpsKnow Repo 의 본질입니다. 다음 6 장은 이 흐름이 외부 API 호출 0건과 어떻게 양립하는지, 즉 운영 데이터가 디렉터리 밖으로 한 글자도 나가지 않는다는 약속을 어떻게 검증 가능한 객관 지표로 만드는지를 다룹니다.

6장. 보안·감사 — 외부 호출 0건, RBAC, 마스킹 파이프라인

운영 지식 저장소를 도입할 때 정책결정권자께서 가장 먼저 던지시는 질문은 두 가지로 압축됩니다. 첫째는 "사내 운영 데이터가 외부로 한 줄이라도 새 나갈 가능성이 있는가" 이고, 둘째는 "감사 대응이 별도 도구·별도 보고서를 요구하지 않는가" 입니다. 이 장은 두 질문을 네 개의 축으로 나누어 답합니다. 외부 API 호출 0건의 객관 검증, Git 의 RBAC(Role-Based Access Control, 역할 기반 접근 통제) 와 CODEOWNERS, 인입 단계의 자동 마스킹 파이프라인, 국내 규제 3종(ISMS-P · 금감원 · 의료) 과의 1:1 정합이 그 네 축입니다 [S42] [S34] [S19].

네 축은 서로 독립이 아닙니다. 외부 호출 0건은 ndjson(Newline Delimited JSON, 한 줄에 JSON 객체 하나씩 적재하는 로그 포맷) 로그로 증명되고, 그 로그는 RBAC 로 보호되며, 인입 단계 마스킹이 통과시킨 데이터만 모델 호출에 닿습니다. 네 단계가 줄지어 있어, 한 단계가 뚫려도 다음 단계가 차단하는 다층 방어 구조입니다. 사전 조사에서 PrivateGPT 가 강조한 "데이터가 외부로 나가지 않는 로컬 RAG" 의 명제를 [S34], OpsKnow Repo 는 운영 도메인의 ontology 와 결합하여 검증 가능한 형태로 확장합니다.

특히 정책결정권자께서는 "0 건" 이라는 표현이 마케팅 슬로건으로 들리지 않도록 객관 증거의 형식을 요구하셔야 합니다. 이 장이 제시하는 ndjson 한 줄 로그, 방화벽 outbound 차단의 코드 가드 이중화, 분기 sampling 감사는 모두 외부 감사인이 직접 열어 볼 수 있는 형식을 전제로 합니다 [S42] [S34]. 운영팀이 별도 보고서를 작성하지 않아도 Git 의 commit 이력과 ndjson 로그가 그대로 감사 근거가 됩니다.

여러분께서 이 장을 다 읽으신 뒤에는, 우리 조직의 규제 환경(ISMS-P · 금감원 · 의료) 에서 "외부 호출 0건" 을 어떤 산출물로 증명할 것인지 한 줄로 정리하실 수 있어야 합니다. 그 한 줄이 곧 도입 의사결정의 안전판입니다 [S19] [S21].

6.1 외부 호출 0건 검증 — ndjson 호출 로그의 의미

"외부 호출 0 건" 이라는 표현은 그 자체로는 슬로건에 그칩니다. 운영팀과 감사인이 동시에 신뢰하려면 (1) 모든 모델 호출이 한 줄 단위 로그에 누적되고, (2) 그 로그를 사후에 누구나 grep 으로 검증할 수 있으며, (3) 네트워크 계층에서도 외부 IP 가 닫혀 있어야 합니다. 이 절은 이 세 단계를 차례로 정의합니다 [S42] [S34].

6.1.1 모든 LLM 호출의 1줄 ndjson 로그

ndjson 한 줄 로그의 표준 필드 다섯

모든 LLM 호출은 발생 즉시 한 줄짜리 ndjson 로그로 떨어집니다. 표준 필드는 다섯 개입니다 — `timestamp` (ISO 8601 시각), `model` (모델 식별자, 예: `gpt-oss-20b-q4`), `prompt_hash` (질의의 SHA-256 64자리 해시), `response_hash` (응답의 SHA-256 64자리), `destination` (호출 대상 호스트, 예: `127.0.0.1:11434`). 한 줄 단위로 적재하므로 grep·jq 로 즉시 필터링·집계가 가능하며, 추가 전용 append-only 형식이라 사후 변조를 logical 검증으로 잡아낼 수 있습니다 [S42].

왜 hash 만 적재하고 본문은 적재하지 않는가

prompt 와 response 본문을 그대로 로그에 적재하면, 그 로그 파일 자체가 두 번째 corpus 가 됩니다. 즉 본문이 흘러나간 흔적을 막으려고 만든 로그가 본문 누출의 통로가 됩니다. 그래서 OpsKnow Repo 의 표준은 SHA-256 해시 64자리만 적재합니다. 누가 어떤 모델에 언제 호출했고 그 결과가 어디로 갔는지의 메타데이터만 남기고, 본문 자체는 호출 컨텍스트(예: 인시던트 디렉터리의 정제 Postmortem) 안에 보관합니다. 메타와 본문의 분리는 사전 조사의 PrivateGPT 가 강조한 "데이터가 외부로 나가지 않는 로컬 RAG" 명제와 정합합니다 [S34].

로그 보존 정책 — 13개월 + 외부 감사 sampling

ndjson 로그의 표준 보존 기간은 13개월입니다. 분기 감사 4회(분기 말 ±2주) 와 연간 외부 감사 1회를 모두 포함할 수 있는 최소 기간이기 때문입니다. 13개월이 지난 로그는 자동 압축 후 cold storage 로 이전하며, 추가 7년 보관은 ISMS-P · 금감원 규정에 따라 결정됩니다. 보존 정책 자체가 frontmatter 스키마와 동일하게 코드(YAML) 로 박혀 있어, 정책 변경 자체가 PR 로 리뷰됩니다 [S42] [S34].

보존 단계	기간	저장소	접근 권한
활성 (hot)	0~3 개월	운영 로그 디렉터리	운영팀 + RAG 인덱서 read-only
분기 감사 (warm)	3~13 개월	감사 디렉터리	감사 그룹 + 외부 감사인 임시 권한
장기 보관 (cold)	13 개월~7년	cold storage 압축	법무 + 외부 감사

6.1.2 외부 IP 호출 자동 차단 (방화벽 + 코드 가드)

방화벽 outbound 차단의 기본 규칙

운영 환경의 LLM 호출은 모두 사내 사설망 안의 추론 서버(Ollama · vLLM · llama.cpp) 로 향합니다. 즉 호출 대상 IP 는 사내 CIDR 블록 안에 있으며, 그 외 모든 outbound 가 방화벽에서 거부됩니다. 표준은 화이트리스트 방식 — Ollama 서버의 사설 IP 와 인증 서버 IP 만 명시 허용하고 나머지는 모두 차단합니다. 방화벽 룰은 코드(예: Terraform · Ansible Playbook) 로 박혀 있어 변경 이력이 Git 에 남습니다 [S43] [S34].

코드 가드 — 애플리케이션 계층의 허용 호스트 검증

방화벽 한 겹에 의존하는 것은 단일 방어선의 위험을 안고 있습니다. 인적 실수로 방화벽 규칙이 일시 완화되거나, 새 추론 서버가 외부 IP 로 잘못 등록되면 외부 호출이 발생할 수 있습니다. 그래서 RAG 인덱서·검색기·답변 생성기의 애플리케이션 코드 안에 두 번째 가드가 들어갑니다. 호출 직전에 destination 호스트를 화이트리스트(예: `127.0.0.1`, `10.0.0.0/8`, `192.168.0.0/16`) 와 대조하여, 외부 IP 면 호출 자체를 거부하고 ndjson 로그에 `blocked: true` 로 기록합니다. 사전 조사의 vLLM 사례는 PagedAttention 으로 고처리량을 확보하면서도 추론 서버 자체가 사내 GPU 위에 떠 있는 구성이며, 이 코드 가드와 자연스럽게 결합합니다 [S43] [S34].

이중 방어의 운영 부담은 작다

방화벽과 코드 가드는 둘 다 한 번 박아두면 운영 부담이 거의 없습니다. 방화벽은 화이트리스트 5~10줄, 코드 가드는 함수 한 개 안의 if 문 두세 줄로 끝납니다. 반면 단일 방어선에서 외부 호출이 한 번 새 나가는 사례의 비용은 ISMS-P · 금감원 감사 지적, 외부 감사 보고서 첨부, 후속 대응 비용까지 합쳐 수개월 단위의 부담으로 돌아옵니다. 이중 방어는 운영 부담 대비 위험 감소의 비대칭이 큰 구조입니다 [S43] [S34].

6.1.3 분기 감사 — ndjson 로그 sampling 검토

분기 1회 sampling 검토의 표준 절차

ndjson 로그가 적재만 되고 사후 활용되지 않으면 6.1.1 의 객관 증거가 무용지물이 됩니다. OpsKnow Repo 의 표준은 분기 1회, 분기 마지막 주에 sampling 검토 사이클을 돌리는 것입니다. sampling 비율은 보통 1% 또는 1,000건 중 큰 쪽이며, 무작위 추출한 ndjson 행에서 `destination` 필드가 모두 사내 IP 인지를 확인합니다. 검토 결과는 `Work_Reports/quarterly/<YYYY-QN>-audit.md` 에 기록되어 다음 분기 감사 사이클의 입력이 됩니다 [S21] [S28].

외부 감사인이 직접 확인할 수 있는 형식

ndjson 은 외부 감사인이 별도 도구 없이 grep · jq 만으로 검증할 수 있는 형식입니다. 예를 들어 `jq -r '.destination' < quarter.log | sort | uniq -c` 한 줄이면 분기 중 호출된 모든 destination IP 의 빈도가 나옵니다. 결과의 모든 행이 사내 CIDR 안에 있으면 외부 호출 0건이 객관적으로 증명됩니다. SaaS 위키 솔루션의 감사 보고서가 자사 포맷의 PDF 인 것과 비교하면, ndjson 의 외부 감사 친화성은 본질적 우위입니다 [S21] [S28].

분기 감사가 미발견 위험을 줄이는 메커니즘

sampling 검토는 단순한 통계 확인이 아니라 미발견 위험의 절감 메커니즘입니다. 분기 1회 검토가 박혀 있으면, 새 추론 서버를 도입하거나 새 코드 경로를 추가했을 때 외부 호출이 끼어드는 운영 이슈가 늦어도 90일 안에 발견됩니다. 사전 조사에서 Blameless 가 SLO 통합으로 운영 신뢰도를 정량 관리하는 사례와 정합하며 [S28], Google SRE Book 의 오류예산 사이클과도 같은 호흡으로 운영됩니다 [S21].

분기 감사 단계	산출물	책임자
1. ndjson 로그 sampling 추출	`audit/<YYYY-QN>/sample.ndjson`	감사 그룹
2. destination 화이트리스트 대조	`audit/<YYYY-QN>/destinations.md`	감사 그룹

분기 감사 단계	산출물	책임자
3. 이탈 행 분석 + 코드 패치	PR + 후속 추적 티켓	운영팀
4. 분기 감사 보고서 commit	`Work_Reports/quarterly/<YYYY-QN>-audit.md`	감사 그룹

6.2 권한 — RBAC + CODEOWNERS

데이터 주권의 두 번째 영역은 권한입니다. 누가 어떤 디렉터리를 읽고, 누가 어떤 PR 을 승인할 수 있는지가 코드에 박혀 있어야 합니다. Git 호스팅(GitHub · GitLab) 의 CODEOWNERS 파일과 RBAC 모델을 결합하면 운영 도메인의 권한 매트릭스가 단일 통제점 안에서 관리됩니다 [S19] [S60]. 이 절은 CODEOWNERS, RBAC, 민감 영역의 별도 권한 세 가지를 차례로 정의합니다.

6.2.1 Git 의 CODEOWNERS 파일 — 서비스별 owner 매핑

CODEOWNERS 의 표준 위치와 패턴 문법

CODEOWNERS 파일은 리포 루트 또는 `.github/`, `docs/` 아래에 위치하는 평문 파일입니다. 한 줄마다 글롭(glob) 패턴과 owner 핸들(예: `@platform-team`, `@user-alice`) 을 짝지어 적습니다. 예를 들어 `Field_Manual/payment-api/** @payment-platform-team`, `Incidents/** @sre-oncall`, `Preventive_Inspection/security/** @security-team` 세 줄이면 결제 API 의 매뉴얼 변경은 결제 플랫폼팀이, 인시던트 디렉터리 변경은 SRE 온콜이, 보안 점검 변경은 보안팀이 자동 reviewer 로 할당됩니다. GitLab Handbook 의 공개 사례는 운영 매뉴얼 디렉터리 전체에 이 모델을 적용한 정전입니다 [S19].

자동 reviewer 할당이 사람 기억을 대체합니다

CODEOWNERS 가 없으면 PR 작성자가 매번 도메인 책임자를 머릿속에서 떠올려 reviewer 로 지정해야 합니다. 결제 API 의 런북 변경 PR 이 결제팀이 아닌 인프라팀으로 잘못 라우팅되는 일이 분기마다 두세 건 발생하고, 그때마다 PR 이 지연되거나 잘못된 승인이 머지됩니다. CODEOWNERS 가 박혀 있으면 라우팅이 자동화되어 사람의 기억 의존이 0 에 가까워집니다. 사전 조사가 강조한 "PR 기반 사람 승인 게이트, CODEOWNERS 로 도메인 책임자 분리" 의 정합 메커니즘입니다 [S60].

CODEOWNERS 변경 자체도 PR 의 대상

CODEOWNERS 파일 자신도 코드입니다. owner 가 추가·변경되면 PR 로 리뷰되어 보안팀 또는 플랫폼 운영팀의 승인이 강제됩니다. 즉 권한 매트릭스의 변경 이력이 그대로 Git 의 commit log 에 남아, 분기 감사에서 "누가 언제 어떤 owner 권한을 어떻게 바꿨는가" 가 한 줄 git log 명령으로 추출됩니다. SaaS 위키의 권한 변경이 별도 관리 콘솔에 흩어지는 것과 본질 차이입니다 [S19].

패턴 예시	owner	의미
`Field_Manual/payment-api/** @payment-team`	결제 플랫폼팀	결제 API 매뉴얼 변경 자동 라우팅
`Incidents/** @sre-oncall`	SRE 온콜 그룹	인시던트 디렉터리 전 변경

패턴 예시	owner	의미
`Preventive_Inspection/security/** @security-team`	보안팀	보안 점검 항목 변경
`_meta/INDEX.md @doc-stewards`	문서 스튜어드 그룹	운영 ontology 인덱스 변경
`CODEOWNERS @security-team @platform-leads`	보안 + 플랫폼 리더	권한 매트릭스 자체의 변경

6.2.2 RBAC — Git 리포 읽기 vs 쓰기 권한 분리

다섯 역할의 표준 매트릭스

Git 호스팅의 RBAC 는 다섯 역할로 정리됩니다 — Owner (리포 관리 전권), Maintainer (브랜치 보호·CI 설정 변경), Developer (PR 작성·머지), Reporter (이슈·PR 댓글), Guest (읽기 전용). OpsKnow Repo 의 표준은 운영팀 전원을 Developer 로, RAG 인덱서 서비스 계정을 Guest (read-only) 로, 분기 감사 그룹을 Reporter 로 두는 것입니다. read-only 인덱서 계정이 원천적으로 쓰기를 못 하므로 corpus 오염의 코드 수준 방어선이 생깁니다 [S19] [S17].

RAG 인덱서가 read-only 인 이유

5장에서 정의한 git pull 트리거 증분 인덱싱은 RAG 인덱서가 매일 git pull 을 돌리고 변경 청크만 재임베딩합니다. 이 인덱서가 쓰기 권한을 갖고 있으면, 추론 결과가 의도치 않게 디렉터리에 커밋되거나 메타가 변경되는 버그가 사고로 이어집니다. read-only 권한이 박혀 있으면 인덱서 프로세스 어디에 버그가 있어도 디렉터리 상태는 보호됩니다. Confluence 같은 SaaS 위키의 AI 컴포넌트가 자사 데이터그래프 위에서 작동하는 것과 달리, OpsKnow Repo 는 Git 의 권한 모델 위에 RAG 가 얹히는 구조라 권한 분리가 자연스럽습니다 [S19] [S17].

branch protection 의 보강 효과

RBAC 한 겹만으로는 부족한 경우가 있습니다. Developer 가 main 브랜치에 직접 push 하면 PR 리뷰가 우회됩니다. 그래서 main 브랜치에 branch protection 규칙을 박아 — 직접 push 금지, PR 리뷰 1인 이상 강제, CI 통과 강제, CODEOWNERS 승인 강제 — 4 조건이 동시에 만족돼야 머지가 가능하도록 합니다. 4 조건이 모두 코드(YAML 또는 호스팅 콘솔의 설정 export) 로 박혀 있어, 설정 변경 자체가 감사 대상이 됩니다 [S19].

역할	읽기	PR 작성	머지	관리
Owner	O	O	O	O
Maintainer	O	O	O	branch protection 만
Developer (운영팀)	O	O	reviewer 1인 + CI 후	X
Reporter (감사 그룹)	O	댓글만	X	X

역할	읽기	PR 작성	머지	관리
Guest (RAG 인덱서)	O	X	X	X

6.2.3 민감 영역 (Incidents/ 일부) 의 별도 권한

민감 영역의 분리 기준

모든 인시던트가 같은 민감도를 가지지는 않습니다. 결제 시스템의 전체 장애, 고객 데이터 노출 사고, 보안 침해 사례 같은 critical 인시던트는 raw 채팅 덤프에 고객명·내부 토큰·인적 발언이 섞일 가능성이 높습니다. 그래서 `Incidents/2026/Q2/<critical-id>/raw/` 같은 raw 디렉터리에는 일반 운영팀이 아닌 보안팀 + 인시던트 커맨더 두 그룹에만 읽기 권한을 부여합니다. 분리 기준은 인시던트 severity (sev1, sev2) 와 `data_class` (customer-data, internal-only, public) 두 메타 필드의 조합으로 결정됩니다 [S23] [S24].

RAG corpus 노출과 사람 열람의 동시 통제

민감 영역 권한은 두 단을 동시에 통제합니다. 첫째, 디렉터리 자체의 읽기 권한이 좁아져 일반 운영팀이 raw 채팅을 열어볼 수 없습니다. 둘째, `rag_visible: false` 와 `sensitivity: high` 가 자동 박힌 frontmatter 가 RAG 인덱서의 메타필터에서 걸려 corpus 에 진입하지 못합니다. 즉 사람과 AI 양쪽의 노출 경로가 동시에 닫힙니다. 사전 조사가 강조한 "민감정보 정책은 백서 부록 C 의 표준 스키마에 enum 으로 박힌다" 의 운영 형태입니다 [S23] [S24].

민감도 변경 자체의 감사 추적

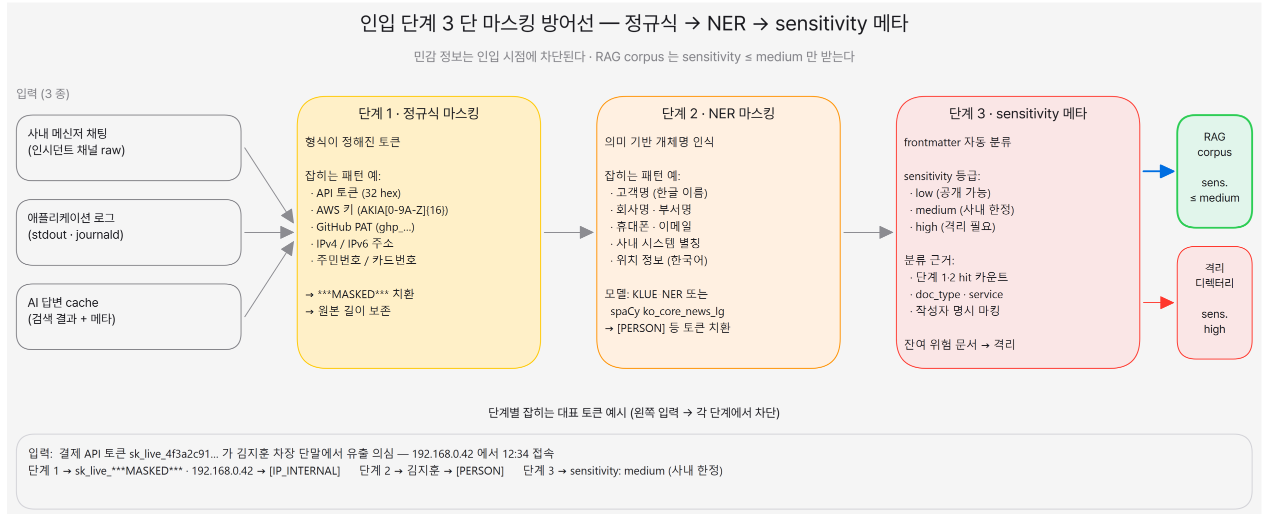
민감도 메타가 변경되면 — 예를 들어 `sensitivity: high` 인 인시던트의 정제 Postmortem 이 90일 후 `sensitivity: medium` 으로 다운그레이드되는 경우 — 그 변경은 PR 로 일어나며, CODEOWNERS 가 자동으로 보안팀을 reviewer 로 할당합니다. 즉 민감도 자체의 변동 이력이 commit log 에 남아, 사후에 "왜 이 문서가 RAG corpus 에 들어왔는가" 가 한 줄 git blame 으로 추적됩니다. 인시던트 SaaS 의 사후 추적이 자사 DB 안에 갇히는 것과 본질 차이입니다 [S23] [S24].

민감도	디렉터리 권한	RAG 노출	감사 추적
high (raw chat, customer-data)	보안팀 + 인시던트 커맨더 read	`rag_visible: false`	변경 이력 commit log
medium (정제 Postmortem)	운영팀 read	`rag_visible: true` , 메타필터 가중	PR 리뷰 + commit log
low (액션 아이템 추적)	전사 read	`rag_visible: true`	commit log

6.3 마스킹 — 인입 단계의 첫 방어선

데이터 주권의 세 번째 영역은 인입 단계의 자동 마스킹입니다. 사내 메신저 채팅 덤프, 로그 import, AI 답변 cache 모두 인입 시점에 정규식 1차, NER 2차, sensitivity 메타 3차의 세 단계 방어선을 통과합니다. 이 절은 세 단계를 차례로 정의하며, 엔지니어가 직접 구현할 수 있도록 실제 정규식 패턴과 메타 규칙을 박아 둡니다 [S24] [S34].

[FIGURE: masking-pipeline] **캡션:** 인입 단계 3 단 마스킹 방어선 — 정규식 → NER → sensitivity 메타 **의도:** 좌측에 사내 메신저 채팅·로그·AI 답변 cache 의 세 입력. 중앙에 세 단의 마스킹 게이트를 횡으로 배치 (단계1 정규식, 단계2 NER, 단계3 sensitivity 메타 필터). 우측에 RAG corpus (sensitivity: low/medium 만 진입) 와 격리 디렉터리 (sensitivity: high). 각 단계마다 잡히는 대표 토큰 예시 (단계1 = api-token, 단계2 = 고객명, 단계3 = 잔여 위험 문서) 를 함께 표기.



6.3.1 정규식 마스킹 (토큰 · IP · 이메일)

다섯 패턴의 표준 정규식 카탈로그

정규식 마스킹은 1차 방어선이며, 다섯 패턴이 표준 카탈로그를 구성합니다. (1) API 토큰 — `(sk|pk|api)[-_][A-Za-z0-9]{32,}` 같은 prefix + 32자 이상 패턴, (2) 내부 IPv4 — `10\.\d+\.\d+\.\d+` , `192\.168\.\d+\.\d+` , `172\.(1[6-9]|2\d|3[01])\.\d+\.\d+` 세 RFC1918 블록, (3) 이메일 — `[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}` 표준 패턴, (4) 전화번호 — `01[016789]-?\d{3,4}-?\d{4}` 국내 휴대전화 패턴, (5) 신용카드 — `\d{4}[-\s]?\d{4}[-\s]?\d{4}[-\s]?\d{4}` 16자리 패턴. 매치된 토큰은 `[MASKED:token]` , `[MASKED:ip]` 같은 라벨로 치환됩니다 [S24] [S34].

99% 를 목표로 하는 1 차 방어선

정규식은 결정적 매칭이라 빠르고 정확하지만, 정규식만으로 100% 마스킹을 약속하는 것은 위험합니다. 토큰 형식이 표준에서 벗어난 경우(예: 22자 짧은 자체 토큰), IPv6, 국제 전화번호의 다양한 표기는 정규식 카탈로그를 우회합니다. 그래서 OpsKnow Repo 의 표준은 "정규식은 99% 를 목표로 하는 1 차 방어선" 이며, 잔여 1% 는 NER 과 sensitivity 메타가 잡습니다. 사전 조사가 "인입 파이프라인이 (i) 자동 마스킹(정규식·NER 기반 토큰/IP/이메일 마스킹)" 으로 명시한 두 단의 의도와 정합합니다 [S24] [S34].

마스킹 결과의 검증 가능성

마스킹은 인입 직후 한 번 일어나고 끝나는 것이 아니라, 검증 가능한 형식으로 디렉터리에 박혀야 합니다. 표준은 마스킹된 파일의 frontmatter 에 `masked_at: <timestamp>` , `masking_version: <semver>` , `masked_patterns: [token, ip, email]` 세 필드를 박는 것입니다. 마스킹 규칙이 업데이트되면 `masking_version` 의 차이로 재처리 대상이 식별되며, 분기 감사 sampling 에서 "이 문서가 어떤 버전 규칙으로 마스킹되었는가" 가 한 줄로 확인됩니다 [S24] [S34].

패턴 #	대상	정규식 (요약)	치환 라벨
1	API 토큰	(sk|pk|api)[-_][A-Za-z0-9]{32,}	[MASKED:token]
2	내부 IPv4	RFC1918 3 블록	[MASKED:ip]
3	이메일	표준 RFC 5322 단순화	[MASKED:email]
4	국내 휴대전화	01[016789]-?\d{3,4}-?\d{4}	[MASKED:phone]
5	신용카드 16자리	\d{4}([-\s]?\d{4}){3}	[MASKED:card]

6.3.2 NER 기반 마스킹 (고객명 · 인적 발언)

NER 모델이 잡는 잔여 위험

정규식이 잡지 못하는 토큰은 인명·고객사명·지명·기관명 같은 고유명사입니다. 채팅 덤프에 "오늘 ㅇㅇ카드 박 OO 차장이 ..." 같은 문장이 있으면 정규식으로는 마스킹되지 않지만, 인적 정보 노출의 위험은 토큰 누출과 동등합니다. 그래서 2 차 방어선으로 NER 모델을 둡니다. 한국어 NER 모델은 KLUE-NER, KoBERT-NER 같은 공개 모델이 표준이며, PERSON · ORG · LOC 세 엔티티만 잡아내도 잔여 위험의 8 할 이상이 해결됩니다 [S34] [S33].

오탐과 미탐의 trade-off

NER 은 통계 모델이라 오탐(false positive · 일반 명사를 인명으로 잘못 마스킹) 과 미탐(false negative · 인명을 놓침) 이 모두 발생합니다. 오탐이 많으면 가독성이 무너지고, 미탐이 많으면 방어선의 의미가 줄어듭니다. OpsKnow Repo 의 표준은 "오탐 허용, 미탐 최소화" 의 보수적 임계값(confidence ≥ 0.5) 입니다. 가독성 회복은 PR 단계의 사람 reviewer 가 false positive 를 풀어 주는 역할로 보완합니다. AnythingLLM 같은 OSS RAG 도구가 마스킹 단계 자체를 제공하지 않는 것과 비교하면, OpsKnow Repo 는 인입 파이프라인 안에 NER 게이트를 박아 둔다는 점이 차별점입니다 [S34] [S33].

NER 모델의 운영 부담은 작다

KLUE-NER 같은 한국어 NER 모델은 한 번 로딩에 1~2 GB 메모리, 단일 GPU 없이 CPU 추론으로도 100 문장당 1~2 초 수준이라 인입 파이프라인의 병목이 되지 않습니다. 채팅 덤프 1만 줄 처리에 분 단위, 일일 인입량 수십만 줄 환경에서도 시간 단위로 완결됩니다. 또한 NER 모델 자체가 사내 GPU 또는 CPU 에서 실행되므로 4 장의 표준 스택 안에서 자연스럽게 운영됩니다 [S34] [S33].

비교 차원	정규식 마스킹	NER 마스킹
대상	토큰·IP·이메일·전화·카드	인명·기관명·지명
정확도	결정적 (높음)	통계적 (보통)
속도	마이크로초 단위	100문장/초 단위
운영 부담	거의 없음	모델 로딩 + 주기 갱신

비교 차원	정규식 마스킹	NER 마스킹
1·2 차 방어선 역할	1 차	2 차

6.3.3 `sensitivity: high` frontmatter 의 RAG 분리

3 차 방어선의 역할

정규식과 NER 이 잡지 못한 잔여 위험은 frontmatter 의 `sensitivity: high` 메타로 표시되어 RAG corpus 자체에서 분리됩니다. 분류 기준은 (1) 인시던트 severity 가 sev1·sev2, (2) `data_class` 가 customer-data, (3) raw 채팅 덤프 중 NER 매치가 5건 이상, 세 조건 중 하나라도 해당하면 자동으로 `sensitivity: high` 가 박힙니다. 3 차 방어선이 박혀 있으면 1·2 차에서 놓친 잔여 위험이 RAG 답변에 인용될 가능성이 0 에 수렴합니다 [S41] [S37].

3 단 방어가 곧 노출 위험의 최소화

세 단계의 방어선은 각각 독립이 아니라 누적입니다. 1 차 정규식이 99% 를 잡으면 2 차 NER 이 그 위에서 8할의 잔여를 잡고, 3 차 sensitivity 메타가 그 잔여의 잔여를 RAG corpus 밖으로 빼냅니다. 세 단계가 모두 통과한 데이터만 LLM 의 컨텍스트에 진입하므로, 외부 호출 0건과 결합하면 운영 데이터의 노출 위험은 사실상 0 에 수렴합니다. 사전 조사가 강조한 "(iii) sensitivity: high frontmatter 필터로 RAG corpus 에서 분리 — 세 단계 방어선" 의 운영 형태입니다 [S41] [S37].

3 차 분리의 운영 비용

3 차 방어선은 자동 메타 부착이므로 운영 부담이 거의 없습니다. 인입 시점에 frontmatter 한 줄이 박히고, RAG 인덱서가 그 메타를 보고 corpus 진입 여부를 결정합니다. 단 잔여 위험을 가진 문서가 사람의 검토 없이 격리 디렉터리에 영구 누적되면 또 다른 부담이 되므로, 분기 감사 사이클에서 sensitivity: high 누적량과 다운그레이드 비율을 함께 점검합니다. 운영팀이 격리 디렉터리를 주기적으로 비우는 사이클이 박혀 있어야 3 단 방어가 살아 있는 메커니즘이 됩니다 [S41] [S37].

6.4 규제 정합 — ISMS-P · 금감원 · 의료 규제

데이터 주권의 네 번째 영역은 국내 주요 규제 3종과의 1:1 정합입니다. ISMS-P 의 변경 감사 요구, 금감원의 외부 LLM 호출 제한, 의료 규제의 환자 정보 보호 — 세 가지 규제가 6.1~6.3 에서 정의한 메커니즘과 어떻게 맞물리는지를 절단면별로 정리합니다. 정책결정권자께서 우리 조직의 규제 환경을 한 줄로 진단하실 수 있는 매핑표가 이 절의 최종 산출입니다 [S19] [S21] [S42].

6.4.1 ISMS-P 의 변경 감사 요구 → Git commit 이력으로 충족

ISMS-P 의 변경 감사 항목 7종

ISMS-P 의 변경 감사 영역은 보통 일곱 항목으로 정리됩니다 — 변경 요청·승인 기록, 변경 책임자 식별, 변경 적용 시각, 변경 전·후 상태, 영향도 평가, 사후 검토, 변경 이력의 보존. 이 일곱 항목 모두가 Git 의 commit 메타데이터와 PR 메타데이터로 자연스럽게 충족됩니다. 변경 요청은 PR 제목, 승인 기록은 reviewer 의 approve, 책임자는 commit author, 적용 시각은 commit timestamp, 변경 전·후는 git diff, 영향도 평가는 PR description, 사후 검토는 PR comment, 이력 보존은 git log 자체입니다 [S19] [S21].

별도 변경 관리 도구가 필요 없다

기존의 SaaS 위키 기반 운영은 변경 관리 도구(예: ServiceNow Change · Jira Change) 와 위키를 별도로 운영하면서, 두 시스템 사이의 정합을 사람이 맞춰야 했습니다. OpsKnow Repo 는 Git 자체가 변경 관리 시스템이라, ISMS-P 감사 항목 일곱이 별도 도구 없이 무료 부산물로 충족됩니다. 사전 조사가 강조한 "외부 감사/규제 요구(예: ISMS-P, 금감원) 에 대응 가능한가? → Git 의 commit 이력이 변경 감사의 1차 근거" 와 정합하며, 운영 부담이 본질적으로 작아집니다 [S19] [S21].

자동 충족과 사람 보강 항목의 분리

ISMS-P 일곱 항목 중 다섯은 Git 의 메커니즘으로 자동 충족되지만, 영향도 평가와 사후 검토 두 항목은 사람의 작성이 필요합니다. OpsKnow Repo 의 표준 PR 템플릿에 영향도 평가 섹션과 사후 검토 체크리스트가 박혀 있어, PR 작성자가 빈 칸을 채우면 두 항목이 동시에 충족됩니다. 즉 자동 충족 5 + 사람 보강 2 의 조합으로 ISMS-P 변경 감사 영역 전체가 단일 도구 안에서 마무리됩니다 [S19] [S21].

ISMS-P 인증기준 항목 (참조 조항)	OpsKnow 메커니즘	충족 형태	1차 출처
2.9.1 변경관리 — 변경 요청 기록	PR 제목 + description	자동	KISA ISMS-P 인증기준 안내서 [SK01]
2.9.1 변경관리 — 승인 기록	reviewer approve + CODEOWNERS	자동	[SK01]
2.6.1 네트워크 접근 — 변경 책임자 식별	commit author + signed-off-by	자동	[SK01]
2.9.1 변경관리 — 적용 시각	commit timestamp + merge timestamp	자동	[SK01]
2.9.1 변경관리 — 변경 전·후 상태	git diff + branch tag	자동	[SK01]
2.10.1 시스템 도입·개발 보안 — 영향도 평가	PR 템플릿 섹션	사람 보강	[SK01]
2.9.2 성능 및 장애관리 — 사후 검토	PR 댓글 + Work_Reports 회고	사람 보강	[SK01]
2.5.6 접속기록 보관 — 이력 보존	git log + branch protection	자동	[SK01]

(주: 본 표의 ISMS-P 인증기준 항목 번호는 KISA 「정보보호 및 개인정보보호 관리체계 인증기준 안내서」의 2.9 시스템 및 서비스 운영관리·2.6 접근통제·2.5 인증 및 권한관리·2.10 시스템 도입 및 개발 보안 영역과 매핑됩니다. 실제 인증 심사 시에는 본인 조직의 인증 범위·심사 일자에 따라 최신 안내서를 다시 확인하시기 바랍니다.)

6.4.2 금감원의 외부 LLM 호출 제한 → 외부 호출 0건 검증

적용 법령·규정 (2026년 6월 기준, 1차 출처 확인 권장)

금융권 도입 시 OpsKnow Repo 의 외부 호출 0건 메커니즘은 다음 법령·규정의 요구사항과 1:1 정합합니다. 단, 법령은 개정 가능성이 있으므로 본인 조직 도입 시점에는 법제처 1차 출처 [SK02]~[SK05] 에서 시행 일자와 조문 본문을 재확인하시기 바랍니다.

- **「전자금융감독규정」 제13조** (전산자료의 보호대책) — 전산자료의 외부 유출 방지 의무 [SK02]
- **「전자금융감독규정」 제14조의2** (클라우드 컴퓨팅서비스 이용·제공) — 클라우드·AI 활용 시 사전 보고와 통제 [SK03]
- 금융위원회·금융감독원 **「금융분야 AI 가이드라인」** (2021년 7월 제정, 이후 개정) — AI 모델 운용 시 책임 추적성과 호출 이력 보존 [SK04]
- 금융보안원 **「금융권 클라우드 컴퓨팅서비스 이용 가이드」** — 사외 전송 데이터의 식별·통제 의무 [SK05]

금감원 가이드라인의 핵심 요구

금융권에 적용되는 금감원 가이드라인은 AI · LLM 활용 시 (1) 사외 전송 데이터의 식별·통제, (2) 외부 호출 발생 시 사전 승인·사후 감사, (3) 모델 호출 이력의 보존 기간(통상 5년 이상), 세 가지 요구를 둡니다. 사외 전송 자체가 사실상 금지된 영역(고객 비식별 정보 포함) 에 대해서는 외부 호출 0건이 진입 조건이 되며, 사후 입증 가능한 형식이 필수입니다 [S42] [S34] [SK02] [SK03] [SK04].

ndjson 로그가 객관 증거가 됩니다

6.1.1 에서 정의한 ndjson 한 줄 로그는 금감원 가이드라인의 세 요구 중 두 번째(사후 감사) 와 세 번째(이력 보존) 를 동시에 충족합니다. 모든 LLM 호출의 destination 필드가 사내 IP 이고, 13개월 + 7년 보관이 박혀 있으면, 금감원 감사 시 grep 한 줄로 외부 호출 0건이 객관 증명됩니다. 첫 번째 요구(사외 전송 식별·통제) 는 6.1.2 의 방화벽 + 코드 가드 이중 방어로 충족됩니다. 세 요구 모두가 단일 시스템 안에서 mapping 되는 구조이며, 사전 조사가 강조한 "국내 금융·공공·의료는 사실상 외부 LLM 호출이 금지된다" 의 직접 대응입니다 [S42] [S34].

금융권 진입의 외부 호출 0건 증명 책임자

금융권 도입 시 외부 호출 0건의 증명 책임자는 분기 감사 그룹과 보안팀 두 그룹의 공동 책임입니다. 분기 감사 그룹이 ndjson 로그의 sampling 검토를 수행하고, 보안팀이 방화벽 룰과 코드 가드의 정합을 분기마다 점검합니다. 두 그룹의 분기 보고서가 `Work_Reports/quarterly/<YYYY-QN>-finance-audit.md` 한 파일로 결합되어 금감원 감사인이 한 번에 열람할 수 있는 형식입니다 [S42] [S34].

금감원 요구	OpsKnow 메커니즘	증명 형식
사외 전송 데이터 식별·통제	방화벽 outbound + 코드 가드 화이트리스트	Terraform 코드 + 코드 가드 함수
외부 호출 사후 감사	ndjson 로그 + 분기 sampling 검토	`Work_Reports/quarterly/...audit.md`
호출 이력 보존 (5년+)	13 개월 hot/warm + 7년 cold	보존 정책 YAML + cold storage

6.4.3 의료 규제의 환자 정보 마스킹 → 인입 단계 마스킹 + sensitivity 메타

적용 법령·고시 (2026년 6월 기준, 1차 출처 확인 권장)

의료 환경 도입 시 OpsKnow Repo 의 3단 마스킹·민감 영역 분리 메커니즘은 다음 법령·고시의 요구사항과 1:1 정합합니다. 가명정보·민감정보 처리는 시행령 개정이 잦으므로 본인 조직 도입 시점에는 법제처 1차 출처 [SK06]~[SK09] 에서 시행 일자를 재확인하시기 바랍니다.

- **「개인정보보호법」 제23조** (민감정보의 처리 제한) — 진단명·처방 내역 등 건강정보의 처리 제한 [SK06]
- **「개인정보보호법」 제28조의2~9** (가명정보의 처리) — 가명·익명 처리 절차, 2023년 시행 [SK09]
- **「의료법」 제21조의2** (진료기록의 송부) 및 동법 시행규칙 — 진료기록 전송 시 동의·통제 의무 [SK07]
- 보건복지부 **「보건의료데이터 활용 가이드라인」** (2020년 9월 제정, 이후 개정) — 가명·익명 처리 표준 절차 [SK08]

의료 규제의 환자 정보 보호 요구

위 법령·고시는 환자 식별 정보(이름·주민번호·진단명·처방 내역) 의 비식별화·접근 통제·전송 통제 세 가지 영역을 요구합니다. 운영 지식 저장소가 환자 정보를 직접 다루지는 않더라도, 인시던트 채팅 덤프에 환자 식별 정보가 우발적으로 섞일 가능성이 있어 인입 단계의 자동 마스킹이 진입 조건이 됩니다 [S34] [S37] [SK06] [SK07].

3 단 마스킹과 민감 영역 분리의 결합

6.3 에서 정의한 3 단 마스킹(정규식 → NER → sensitivity 메타) 과 6.2.3 의 민감 영역 권한이 의료 규제의 세 축과 1:1 정합합니다. 비식별화는 1 차 정규식 + 2 차 NER 로, 접근 통제는 3 차 sensitivity 메타와 민감 영역 권한으로, 전송 통제는 외부 호출 0건과 방화벽 이중 방어로 각각 충족됩니다. 사전 조사가 강조한 "민감정보 마스킹은 인입 단계에서 강제된다" 의 의료 분야 적용 형태입니다 [S34] [S37].

한국어 환자 정보 NER 의 사전 검증

의료 환경 도입 시 가장 중요한 사전 검증은 NER 모델의 한국어 환자 정보 인식 정확도입니다. KLUE-NER 같은 공개 모델은 일반 인명·기관명에 최적화되어 있어, 의료 도메인 특화 엔티티(진단명·처방명·의료기관명) 의 정확도는 추가 fine-tuning 또는 도메인 사전 결합이 필요합니다. 도입 PoC 단계에서 의료 도메인 코퍼스 1,000 문장 샘플로 NER 정확도를 사전 측정하고, 임계값(예: 재현율 0.95 이상) 을 만족할 때 본 도입을 진행하는 절차가 표준입니다. 이 절차 자체가 `Work_Reports/quarterly/<YYYY-QN>-medical-ner-validation.md` 로 박혀 분기 감사 대상이 됩니다 [S34] [S37].

의료 규제 요구	OpsKnow 메커니즘	단계
환자 정보 비식별화	정규식 마스킹 (토큰·전화) + NER 마스킹 (인명·기관)	6.3.1, 6.3.2
환자 정보 접근 통제	sensitivity: high 메타 + 민감 영역 RBAC	6.3.3, 6.2.3
환자 정보 전송 통제	외부 호출 0건 + 방화벽 + 코드 가드 이중 방어	6.1.1, 6.1.2

의료 규제 요구	OpsKnow 메커니즘	단계
사후 감사 추적	git commit log + ndjson 로그 + 분기 sampling	6.1.3, 6.4.1

여러분께서 도입을 결정하시기 전, 우리 조직의 규제 환경을 위 세 매핑 표(ISMS-P · 금감원 · 의료) 의 어느 행이 직접 해당하는지 한 번에 점검하시면 됩니다. 해당 행의 메커니즘이 모두 코드와 메타로 박혀 있다는 사실이 곧 도입의 안전판이 되며, 별도의 감사 도구·별도의 변경 관리 도구 도입 부담 없이 Git 의 기존 메커니즘 위에서 규제 정합이 단일 통제점으로 수렴합니다.

7장. 도입 시나리오 — 단일 시스템 6주 PoC + 다중 시스템 분할

여기까지 1장부터 6장에 걸쳐 OpsKnow Repo 의 도서관 비유, 일곱 영역의 디렉터리, 사람·AI 공동편집 트랜잭션, Local LLM 과 RAG 파이프라인, 외부 호출 0건 검증과 규제 정합까지 설계 그림이 한 장의 디렉터리 트리 안에 모였습니다. 7장은 그 그림을 다시 한 번 처음부터 깔끔하게 그리지 않습니다. 대신 여러분의 운영팀이 다음 분기 첫 달, 둘째 달, 셋째 달에 무엇을 손에 들고 시작해야 하는지를 6주 PoC(Proof of Concept, 개념 검증) 로드맵으로 답합니다.

도입 의사결정의 가장 흔한 실패는 "표준을 너무 크게 잡는 것" 입니다. 운영팀 다섯 명이 첫 분기에 모든 시스템·모든 런북·모든 인시던트를 한 번에 옮기려 시도하면, 12주 뒤에 어떤 산출물도 사람의 손을 떠나지 못한 채 회의록만 남습니다. 반대로 가장 작은 단위 — 단일 시스템·단일 운영팀·단일 RAG corpus — 로 6주 안에 검증 가능한 산출물을 만들면, 7주차부터의 확장은 같은 패턴의 반복에 불과합니다 [S19]. 7.1 의 주차별 마일스톤은 그 가장 작은 단위를 6주 일정으로 매핑한 결과입니다.

확장 단계에서는 또 다른 의사결정이 기다립니다. 시스템 50개를 한 리포지토리에 다 모을 것인가, 도메인 4~6 개로 쪼갤 것인가, 환경(prod·staging·sandbox)별로 강제로 분리할 것인가. 이 셋은 운영팀 규모·서비스 도메인 수·규제 요구사항이라는 세 입력에 따라 자연스럽게 갈라지며, 잘못 고른 분할은 cross-link 의 90%를 끊고 RAG corpus 의 가치를 비선형으로 깎습니다 [S30]. 두 번째 절은 이 분할 의사결정의 입력과 결과를 매트릭스로 정리합니다.

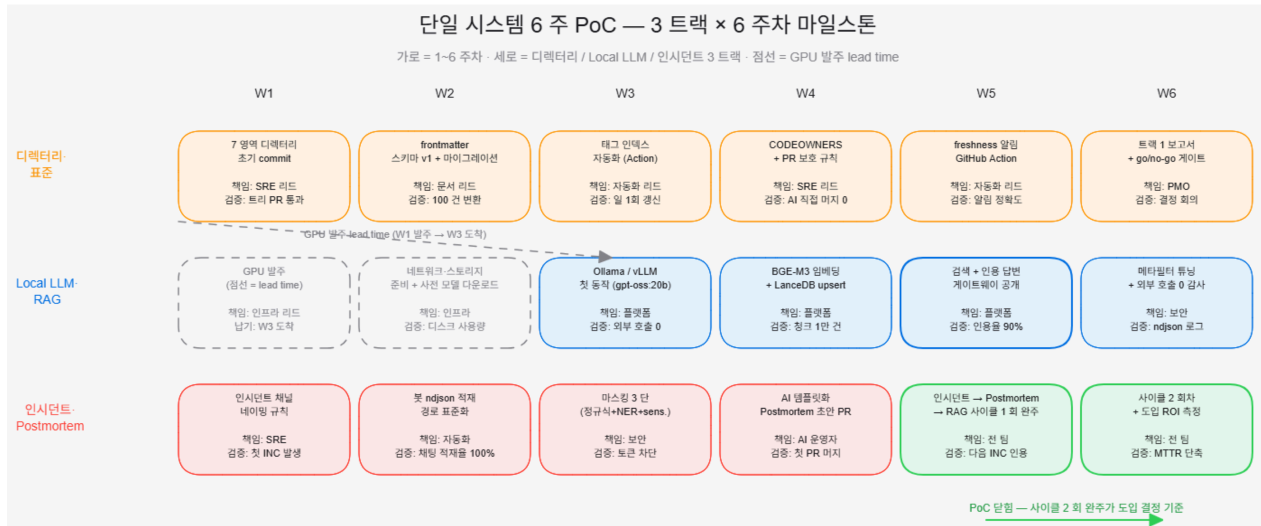
마지막으로 사람의 분할이 AI 의 통합 검색을 막아서는 안 됩니다. 리포지토리를 N 개로 쪼개도 RAG 인덱서는 N 개를 단일 corpus 로 합쳐 cross-link 가 유지되는 비대칭 권한 모델이 가능합니다 [S41]. 운영팀이신 여러분께서 이 마지막 원칙을 미리 합의하지 않으면, 분할 의사결정 자체가 "AI 검색이 약해질 텐데" 라는 잘못된 걱정에 발이 묶입니다. 7장의 결론은 분할과 통합이 서로의 적이 아니라 서로 다른 목적을 가진 두 손이라는 점입니다.

7.1 단일 시스템 6주 PoC 로드맵

6주 PoC 는 OpsKnow Repo 도입의 가장 작은 단위입니다. 단일 시스템(예: 결제 API 또는 사내 인증 서버) 한 곳에서, 단일 운영팀 3~7 명이, 단일 Git 리포지토리에 디렉터리 표준·Field Manual 이식·Local LLM RAG·인시던트 채팅 자동화까지 한 사이클을 완주하는 것이 목표입니다. 6주라는 기간은 사전조사의 Executive Summary 메시지 5 — "6주 안에 첫 Field Manual 이식 + Incident 채팅 덤프 자동화가 가능" —

를 정량 마일스톤으로 풀어낸 것이며, 그 근거는 GitLab Handbook 모델의 runbooks 공개 사례에서 처음부터 끝까지 MD + Git + PR 사이클로 운영팀 1개가 분기 안에 정착시킨 패턴입니다 [S19].

[FIGURE: poc-6week-roadmap] **캡션**: 단일 시스템 6주 PoC — 주차별 마일스톤 (디렉터리 표준 → Local LLM → 인시던트 사이클) **의도**: 가로축 1~6 주차, 세로축에 세 트랙 (디렉터리·표준 / Local LLM·RAG / 인시던트·Postmortem). 각 주차의 책임자·산출물·검증 항목이 카드 형태로 박힘. 3~4주차의 GPU 확보 일정이 사전 점선으로 1주차까지 거슬러 올라가 있어 자원 의사결정의 lead time 을 시각화. 5~6주차의 사이클 1회 완주가 두 굵은 화살표로 PoC 의 닫힘을 표현.



7.1.1 1~2주차 — 디렉터리 표준 + 첫 Field Manual 이식

1~2주차의 산출물은 단 두 가지입니다. 첫째, OpsKnow Repo 의 7개 영역(OPS Diary·Field Manual·Work Reports·Preventive Inspection·Incidents·ADR·_meta) 을 단일 리포지토리에 빈 디렉터리로 만들어두고 README·CODEOWNERS·frontmatter 표준 스키마 1.0 을 합의해 commit 합니다. 둘째, 운영팀 트래픽이 가장 높은 시스템(예: 결제 API 또는 사내 SSO) 의 런북·SOP 1세트를 Field Manual 디렉터리로 옮겨와 `owner` · `reviewer` · `freshness_until` frontmatter 를 채웁니다. 이 두 산출물이 1~2주차 동안 PR 2건으로 머지되면 마일스톤 통과입니다.

여러분의 운영팀이 이 단계에서 가장 자주 실수하는 지점은 "한 번에 다 옮기려 한다" 는 것입니다. Field Manual 의 가장 트래픽 높은 1개 시스템·5~10 페이지 분량으로 의도적으로 잘라야 1~2주차에 마칠 수 있습니다. 글로벌 시스템 운영자 커뮤니티가 공유하는 합의 "SSoT(Single Source of Truth, 단일 진실원천) 는 하나만 골라라. 두 개는 곧 0개" 의 의미가 이 첫 단계에서 가장 무겁게 작동합니다 — Confluence 와 OpsKnow Repo 를 같이 쓰면 두 곳 모두 6개월 안에 빈 책장이 됩니다 [S19]. 1~2주차에 옮긴 5~10 페이지가 곧 단일 SSoT 의 첫 깃발입니다.

검증 항목은 세 가지로 좁힙니다. (a) 디렉터리 트리 합의안이 README 에 박혀 있고 6개 영역 폴더에 `.gitkeep` 이 commit 되어 있습니다, (b) Field Manual 1개 시스템의 런북 1세트가 PR 로 머지되어 `owner` 가 frontmatter 에 박혀 있습니다, (c) CODEOWNERS 파일에 해당 시스템의 reviewer 가 자동 할당되도록 패턴이 적혀 있습니다. 이 세 항목이 통과되면 3~4주차의 Local LLM 단계로 넘어갑니다. 통과되지 않은 채로 진행하면 4주차 RAG 인덱스가 corpus 의 일관성 부재로 흔들리기 시작합니다.

1~2주차의 책임자는 운영팀장 1명 + 시니어 SRE 1명으로 충분합니다. 책임자 두 명이 README·frontmatter 표준을 직접 작성하고, 나머지 팀원은 PR 리뷰어로 참여합니다. 이 단계의 시간 투자는 1인당 주당 4~6 시간 수준이며, 평상 운영을 멈추지 않고도 진행 가능합니다 — Field Manual 1세트 이식이 가장 부담스러운 작업인데, 기존 Confluence·Notion 페이지를 MD 로 export 한 뒤 수작업 정리 2~4 시간이 보통입니다.

7.1.2 3~4주차 — Local LLM + 임베딩 인덱스 구축

3~4주차에는 Ollama 또는 vLLM 위에 7~14B 급 모델을 띄우고, BGE-M3 임베딩으로 1~2주차에 이식한 Field Manual 을 인덱싱한 뒤 LanceDB·Chroma 같은 로컬 벡터 DB 에 적재합니다. 사전조사의 r/LocalLLaMA 합의 — "Ollama + Open WebUI 또는 Ollama + AnythingLLM 조합이 사실상 표준" — 이 이 단계의 표준 스택을 결정합니다 [S42]. AnythingLLM 의 워크스페이스 단위 RAG 또는 Open WebUI 의 Knowledge 컬렉션 둘 중 하나를 골라 첫 RAG 답변을 검증합니다 [S33].

GPU 자원 확보가 이 단계의 가장 큰 lead time(선행 기간) 입니다. 7~14B 모델을 단일 24GB GPU 1장으로 띄울 수 있으므로 사내 워크스테이션·서버 1대를 1~2주차부터 미리 할당해 두어야 3주차에 즉시 기동 가능합니다. 사전조사 FAQ Q11 이 짚는 손익분기 6~18개월의 출발선이 여기서 그어집니다 — 이 자원 의사결정을 5주차로 미루면 6주 PoC 가 8주로 늘어납니다. 운영팀장께서 1주차에 자원 요청을 먼저 결재 라인에 올리는 것이 6주 일정 사수의 첫 조건입니다.

3~4주차 마일스톤 검증은 다음과 같이 좁힙니다. (a) Local LLM 이 OpenAI 호환 API 로 응답합니다, (b) 임베딩 인덱스가 Field Manual 50~200 청크를 정상 적재했습니다, (c) 운영팀이 사내에서 가장 자주 묻는 질문 5개 ("결제 API 의 timeout 설정은", "로그인 실패 5xx 의 1차 대응은" 같은) 에 RAG 답변이 인용 형식 `[파일명:라인]` 으로 돌아옵니다. 세 번째 항목의 인용 형식이 5장 5.3.1 에서 정의한 시스템 프롬프트에 박혀 있어야 하며, 이 인용이 사용자가 클릭 한 번에 원본 MD 로 이동 가능한 wikilink 형식이면 사용자 신뢰가 1~2주 안에 형성됩니다.

이 단계에서 LLM 의 추론 성능보다 RAG corpus 의 품질이 답변 정확도를 결정합니다. 사전조사 FAQ Q7 이 명시한 "운영 질의의 9할은 사실 검색 + 인용이고 7B~14B 급 로컬 모델이 충분히 처리한다" 는 합의가 이 마일스톤의 기대치를 결정합니다 [S42]. 7~14B 모델로 답변 품질이 부족해 보인다면 1순위 점검은 모델 크기가 아니라 Field Manual 의 frontmatter 메타가 빠져 있거나 청크 크기가 H2 절 단위와 어긋난 경우입니다. 즉 4주차 종료 시점의 답변 품질이 곧 1~2주차 디렉터리 표준의 품질 신호가 됩니다.

7.1.3 5~6주차 — Incident 채팅 덤프 자동화 + 사람 reviewer 사이클 1회

5~6주차는 인시던트 사이클 1회 완주가 목표입니다. 사내 메신저 또는 Teams 채널에 인시던트 봇을 붙여 채팅 메시지를 ndjson 으로 적재하고, 인시던트 종료 시 AI 가 동일 데이터를 정제 템플릿 (원인·타임라인·액션 아이템·태그) 으로 변환하여 `Incidents/YYYY/QN/<id>/postmortem.md` 로 PR 을 만듭니다. 사람 reviewer 1인 이상이 승인하면 머지되고, 머지된 MD 는 다음 인시던트의 RAG 컨텍스트가 됩니다. 이 학습 순환을 PoC 기간 안에 1회 완주하는 것이 검증의 마지막 점입니다 [S24].

PagerDuty·Incident.io·Rootly 같은 인시던트 SaaS 3강이 "사내 메신저 채팅 → AI 요약 Postmortem 초안" 을 이미 표준 기능으로 보유하고 있다는 사실은 사전조사 (A-c) 표가 정리한 그대로입니다 [S23][S26]. OpsKnow Repo 가 차지하는 자리는 그 산출물이 자사 SaaS DB 가 아닌 사내 Git 리포지토리의 MD 로 즉시 확정된다는 점입니다. 5~6주차의 핵심 검증은 곧 "산출물이 어디에 떨어지는가" 의 차이가 사내 SSoT 와의 통합으로 이어지는지의 확인입니다. raw 덤프는 `Incidents/<id>/raw/chat.md` 에 보존하고 (감사용), 정제

Postmortem 은 별도 MD 로 분리하는 2 단계 처리가 5주차에 이미 frontmatter 정책에 박혀 있어야 6주차 사이클이 안전합니다.

5~6주차 동안 실제 인시던트가 발생하지 않으면 가상 시나리오 1건으로 대체합니다. 직전 6개월간의 실제 인시던트 중 1건을 선정하여 사내 메신저 채널을 복원하고, 그 채팅을 봇이 ndjson 으로 흡수해 동일 사이클을 돌립니다. 가상 시나리오는 검증의 valid 한 대체 수단이며, 6주 안에 사이클 1회 완주 자체가 PoC 의 산출물입니다. 운영팀장께서 1주차에 이 가상 시나리오의 대체 정책을 미리 합의해 두면 5주차 시점의 의사결정 부담이 사라집니다.

검증 항목은 다음 네 가지입니다. (a) 인시던트 봇이 사내 메신저 채팅을 ndjson 으로 적재합니다, (b) AI 가 ndjson 을 정제 템플릿화하여 PR 을 만듭니다, (c) 사람 reviewer 가 PR diff 를 검토하고 frontmatter 의 `author: ai` · `confidence` 가 박혀 있음을 확인한 뒤 머지합니다, (d) 머지된 Postmortem MD 가 RAG 인덱스에 재반영되어 동일 키워드 질의 시 새 답변이 이 MD 를 인용합니다. 네 항목이 통과되면 6주 PoC 종료 시점에 보고서 1쪽으로 의사결정자께 결과를 보고 가능합니다.

7.2 다중 시스템 분할 전략

6주 PoC 가 성공하면 다음 의사결정은 "확장의 형태" 입니다. 시스템 수가 10개를 넘어서고 운영팀 규모가 50명을 넘기 시작하면 단일 리포지토리의 cross-link 장점이 점차 한계에 부딪힙니다. 사전조사 FAQ Q13 이 정리한 세 분할 패턴 — 단일 리포, 도메인별 리포 + meta-repo, 환경별 리포 — 이 이 단계의 선택지입니다 [S19]. 잘못된 분할은 사람의 인지 부담은 줄이지만 RAG corpus 의 cross-link 90% 를 끊고, 잘못된 통합은 권한 격리를 무너뜨립니다. 두 위험 사이의 균형을 찾는 것이 본 절의 과제입니다.

분할 패턴	적합 규모	강점	약점	대표 의사결정 트리거
단일 리포	시스템 ≤ 10, 운영팀 ≤ 50명	cross-link 100%, 검색·자동화 최적, 운영 ontology 단일	규모 확장 시 PR 충돌·CODEOWNERS 복잡도 증가	"한 팀이 모든 시스템을 본다"
도메인별 리포 + meta-repo	서비스 도메인 4~6개	도메인 격리, 도메인 책임자 분리 명확	cross-link 약화, meta-repo 동기화 부담	"결제·로그인·검색이 별도 팀"
환경별 리포 (prod·staging·sandbox)	규제 강제 환경	환경 격리 강제, RBAC 와 결합	환경 간 cross-link 거의 불가, 운영 데이터 중복 위험	"금감원·ISMS-P 가 환경 분리 요구"

7.2.1 단일 리포 (~ 10 시스템, ~ 50명 운영팀)

시스템이 10개 이하이고 운영팀 규모가 50명 이하라면 단일 리포지토리가 cross-link·검색·자동화 모든 면에서 최적입니다. 분할의 유혹이 있더라도 이 규모에서는 단일 리포가 정답입니다. cross-link 의 가치는 비선형으로 작동하므로, 어제자 OPS Diary 가 오늘 Postmortem 의 컨텍스트가 되고 그 둘이 분기 예방점검의 근거가 되는 누적 순환 구조 (2장에서 정의) 가 단일 리포에서 가장 강하게 작동합니다.

여러분의 운영팀이 이 단계에서 자주 듣는 우려는 "리포지토리가 너무 커지지 않을까" 입니다. 사전조사 FAQ Q1 이 짚는 "Ripgrep·Git grep 은 수십만 MD 도 초 단위로 검색하며 RAG 인덱스는 임베딩 차원에서 의미 검

색을 보완한다" 가 이 우려의 1차 답변입니다 [S19]. 운영팀 50명·시스템 10개 환경에서 12개월간 누적되는 MD 파일은 보통 3,000~8,000 건이며, 이 규모는 GitHub·GitLab 의 read·write 성능에 부담을 주지 않습니다. 단일 리포의 한계 임계값은 파일 수가 아니라 PR 충돌 빈도·CODEOWNERS 패턴 수에서 먼저 도래합니다.

단일 리포의 한계 신호는 두 가지 형태로 옵니다. (a) PR 충돌이 주간 5건 이상 발생하고 reviewer 할당이 사람 기억에 의존하기 시작합니다, (b) CODEOWNERS 패턴이 30줄 이상으로 늘어나 신규 시스템 추가 시 reviewer 매핑이 누락됩니다. 이 두 신호가 동시에 나타나면 7.2.2 도메인별 리포로의 분할을 검토합니다. 신호가 하나만 나타날 때는 CODEOWNERS 정리·PR 템플릿 강화로 단일 리포를 6~12개월 더 운영하는 것이 더 효율적입니다.

7.2.2 도메인별 리포 (서비스 도메인 4~6 개)

서비스 도메인이 4~6 개로 명확히 갈라지는 환경 — 예를 들어 결제·로그인·검색·추천·정산이 별도 팀으로 운영되는 환경 — 에서는 도메인별 리포 + meta-repo 패턴이 적합합니다. 각 도메인이 자체 리포를 갖고 운영 ontology·CODEOWNERS·PR 정책을 도메인에 맞게 조정하며, 상위 meta-repo 가 7개 영역의 디렉터리 표준·frontmatter 스키마·인덱서 설정을 SSoT 로 유지합니다. Git submodule 또는 monorepo 도구(예: Nx, Turborepo) 가 이 단계의 통합 도구로 활용됩니다 [S19].

도메인 분할의 가장 큰 trade-off 는 cross-link 약화입니다. 결제 도메인의 인시던트가 로그인 도메인의 런북을 참조해야 할 때 wikilink 가 리포 경계를 넘지 못합니다. 이 약화의 부담을 RAG 인덱서가 N 개 리포를 단일 corpus 로 합치는 패턴 (7.3.1 에서 정의) 으로 보완합니다 — 사람의 분할은 인지·권한 경계이고, AI 의 통합은 검색 경계입니다. 두 경계가 분리되어야 도메인 분할의 trade-off 가 감당 가능해집니다 [S30].

도메인별 리포 패턴의 정착 기간은 3~6 개월이 보통입니다. 첫 도메인 1개를 단일 리포에서 떼어내어 독립 리포로 만들고, 2~3주의 안정화 기간을 거친 뒤 다음 도메인을 분할합니다. 한꺼번에 4~6 개를 동시 분할하는 시도는 meta-repo 의 동기화 부담이 6주차에 임계치를 넘어 정착에 실패합니다. 운영팀장께서 "한 번에 하나의 도메인" 정책을 분기 계획에 박아두는 것이 분할 정착의 1번 조건입니다.

7.2.3 환경별 리포 (prod / staging / sandbox 분리 의무 환경)

규제 요건으로 prod·staging·sandbox 의 운영 데이터·접근 권한 분리가 강제되는 환경 — 예를 들어 금융권의 ISMS-P 또는 의료의 환자 정보 격리 요구 — 에서는 환경별 리포가 단일 리포의 사실상 유일한 대안입니다. 환경별 리포는 RBAC 와 결합하여 prod 환경의 운영 데이터가 staging 환경의 운영자에게 노출되지 않도록 격리합니다. 6장 6.4 의 ISMS-P·금감원·의료 규제 매핑이 이 분할 패턴의 직접적 근거입니다 [S21].

환경별 리포의 trade-off 는 환경 간 cross-link 이 거의 불가능하다는 점입니다. prod 환경의 인시던트가 staging 환경의 검증 절차를 참조해야 할 때 wikilink 가 환경 경계를 넘지 못하며, RAG 인덱서도 환경별 corpus 를 통합하면 격리 자체가 무너집니다. 따라서 환경별 리포는 분할 패턴 세 가지 중 가장 무거운 선택지이며, 규제 요건이 명시적으로 강제하지 않는 한 도메인별 리포로 충분한 경우가 다수입니다. 정책결정권자께서 규제 텍스트의 환경 분리 의무 조항을 1차 출처에서 직접 확인하신 뒤 환경별 리포 결정을 내리시는 것이 안전합니다.

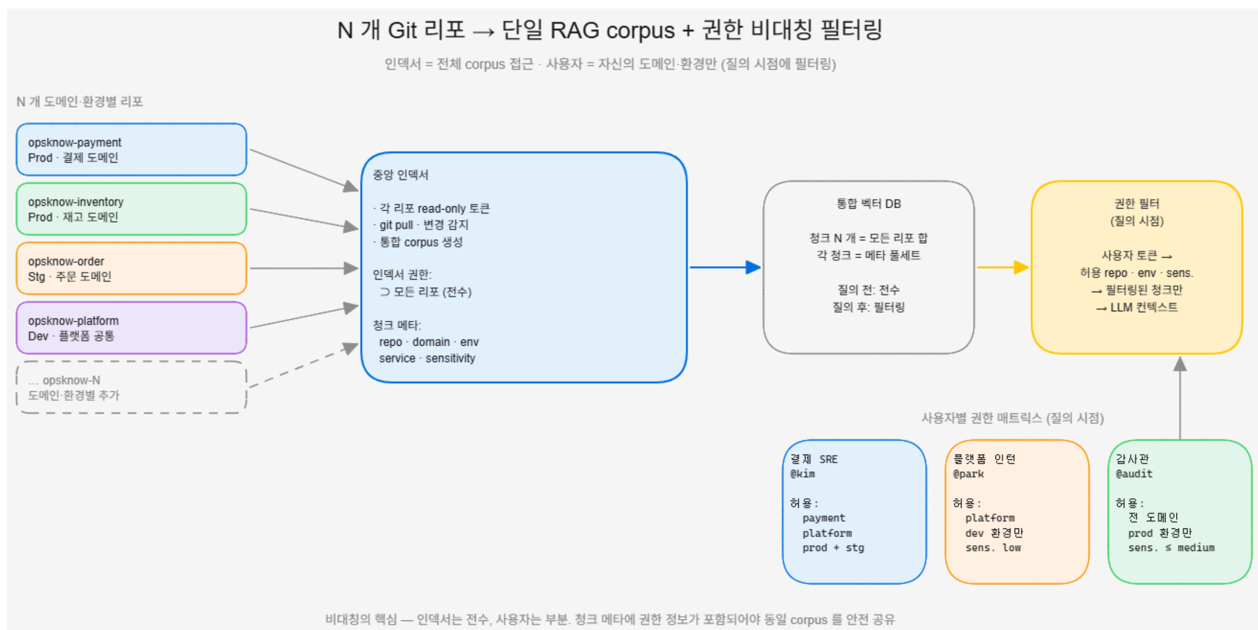
환경별 리포의 운영 부담은 cross-link 약화 외에도 운영 데이터 중복으로 옵니다. 동일 시스템의 런북이 prod·staging·sandbox 세 리포에 각각 존재하면 유효성 관리가 3 배가 되고, 한 환경의 런북 업데이트가 다른 환경에 반영되지 않는 표류 위험이 커집니다. 이 위험은 6장 6.2.3 의 민감 영역 권한 + 환경별 리포 결합 패턴 (Incidents 폴더 일부만 환경별 분리) 으로 일부 완화 가능합니다. 즉 모든 영역을 환경별로 쪼개는 대신 인시던

트·민감 운영 데이터만 환경별 분리하고, Field Manual·표준 SOP 는 단일 리포에 두는 hybrid 패턴이 실무의 절충안입니다.

7.3 RAG 인덱서의 multi-repo 통합

7.2 의 분할 의사결정에서 자주 발목을 잡는 우려는 "리포를 쪼개면 AI 검색이 약해질 것" 이라는 잘못된 걱정입니다. 이 절은 그 우려가 잘못된 우려임을 RAG 인덱서의 multi-repo 통합 패턴으로 보입니다. Onyx (구 Danswer) 와 Glean 이 50+ 커넥터로 multi-source 통합 검색을 이미 제공하는 사실은 사전조사 (A-d) 표가 정리한 그대로입니다 [S41][S30]. OpsKnow Repo 의 multi-repo 통합은 같은 패턴의 사내 Git 리포 버전입니다.

[FIGURE: multi-repo-rag-integration] **캡션**: N 개 Git 리포 → 단일 RAG corpus 통합 + 권한 비대칭 필터링 **의도**: 좌측에 N 개 리포 박스 (도메인별 또는 환경별). 중앙에 인덱서가 각 리포의 read-only 토큰으로 git pull 하여 단일 corpus 형성. 우측에 사용자 질의 시점에서 사용자 권한으로 결과 필터링하는 로직. 인덱서의 권한 (전체 corpus 접근) 과 사용자의 권한 (자신의 도메인·환경만) 이 비대칭임을 색상 대비로 표현.



7.3.1 인덱서가 N 개 리포를 한 corpus 로 합치는 패턴

RAG 인덱서가 N 개 리포를 단일 corpus 로 합치는 구현은 각 리포의 read-only 토큰을 인덱서 서비스 계정이 보유하는 형태가 가장 단순합니다. 인덱서는 분 단위 또는 시간 단위로 N 개 리포를 git pull 하여 변경 파일을 감지하고, 5장 5.1.4 에서 정의한 증분 인덱싱으로 변경된 청크만 재임베딩하여 단일 벡터 DB 에 적재합니다. 이 패턴은 Glean 의 50+ 커넥터 모델, Onyx 의 multi-source 통합 모델과 본질적으로 동일하며 차이는 corpus 가 사내 Git 리포라는 점뿐입니다 [S30][S41].

인덱서의 multi-repo 지원 여부가 분할 환경의 도구 선택 기준이 됩니다. Open WebUI Knowledge 처럼 단일 폴더 업로드 모델은 multi-repo 통합이 어렵고, AnythingLLM 의 워크스페이스 단위 RAG 는 워크스페이스 1개를 N 개 리포에 매핑하는 우회로 다소 부담스럽습니다 [S33]. 반면 Onyx 같은 OSS 통합 검색 도구는 multi-source 통합이 1급 기능이므로 분할 환경의 1순위 후보가 됩니다. 도구 선택 시 "이 도구가 N 개 리포를 단일 corpus 로 합치는가" 를 1번 점검 항목으로 두는 것이 안전합니다.

인덱서의 운영 부담은 N 의 증가에 선형이 아닙니다. read-only 토큰 N 개 관리·git pull N 회·변경 감지 N 회는 N 이 10~20 까지는 단일 인덱서 인스턴스로 처리 가능하며, 그 이상에서는 인덱서를 도메인별·환경별로 샤딩(sharding, 분할 운영) 하는 패턴으로 확장합니다. 운영팀이신 여러분께서 N 의 단기·중기 추정을 미리 해두면 인덱서 도구 선택 시 샤딩 지원 여부도 함께 검토 가능합니다.

7.3.2 권한 분리 + 통합 검색의 양립

multi-repo 통합의 가장 자주 듣는 우려는 "사람의 권한 격리와 AI 의 통합 검색이 충돌하지 않는가" 입니다. 답은 비대칭 권한 모델입니다 — 인덱서는 전체 corpus 에 접근 가능하지만, 사용자 질의 시점에 검색 결과를 사용자 권한으로 필터링합니다. 사용자가 도메인 A 의 운영자라면 검색 결과 중 도메인 B·C 의 청크는 답변에서 제외되고, 환경 staging 의 운영자라면 prod 환경의 청크가 노출되지 않습니다. Glean 의 permission-aware search 가 동일 패턴이며, Onyx 의 connector-level ACL 도 같은 사상입니다 [S30][S41].

권한 필터링의 구현은 frontmatter 메타 + 사용자 그룹 매핑으로 풉니다. 각 MD 의 frontmatter 에 `domain` · `environment` · `sensitivity` 가 박혀 있고, 사용자가 속한 그룹의 권한 매트릭스가 검색 결과의 1차 필터로 작동합니다. 이 권한 매트릭스는 6장 6.2.2 의 RBAC 정의와 1:1 정합하며, 운영팀의 사용자 그룹과 ldap·SSO 의 그룹 매핑이 사전에 합의되어 있어야 합니다. 정책결정권자께서 사용자 그룹 정책을 6주 PoC 단계에서 미리 설계해 두면 7주차 이후의 확장이 권한 정의의 부담 없이 진행 가능합니다.

권한 필터링의 검증은 분기 1회의 audit 으로 충분합니다. 도메인 A 의 운영자 계정으로 도메인 B 의 런북 키워드를 질의했을 때 도메인 B 의 인용이 답변에 나오지 않는지를 5~10 건의 샘플로 확인합니다. 이 audit 은 6장 6.1.3 의 분기 ndjson 로그 sampling 검토와 결합하면 분기 1회 1시간 안에 끝낼 수 있는 작업이며, 외부 감사인이 권한 격리를 검증 가능한 형식을 제공합니다.

7.3.3 사람의 분할이 AI 의 통합 검색을 막지 않는다는 원칙

7.2 와 7.3 의 결합이 가져오는 결론은 한 줄로 요약됩니다 — 분할의 목적은 사람의 인지·권한 통제이고, 통합의 목적은 AI 의 cross-link 활용입니다. 두 목적이 서로의 적이 아니며, 두 손이 서로 다른 일을 동시에 하는 것에 가깝습니다 [S30]. 분할 의사결정의 회의실에서 "AI 검색이 약해질 텐데" 라는 우려가 등장하면, 그 우려는 인덱서의 multi-repo 통합 패턴을 미리 합의해 두면 사라지는 잘못된 걱정입니다.

여러분의 운영팀이 7.2 의 분할 패턴 중 도메인별 또는 환경별을 선택하실 때, 7.3 의 통합 패턴은 자동으로 따라옵니다. 단일 리포에서 도메인별로 분할하는 의사결정과 함께 인덱서의 multi-repo 통합을 동시에 도입하면, 사람의 PR 워크플로우는 도메인별로 분리되고 AI 의 검색은 통합 corpus 에서 그대로 작동합니다. 이 동시 도입이 분할 정착의 안전판이며, 분할만 도입하고 통합을 뒤로 미루는 의사결정은 RAG corpus 의 가치를 6개월 동안 절반으로 깎습니다.

본 원칙의 실무 시사점은 분할 의사결정 회의록에 두 줄을 박는 것입니다. (a) 분할의 목적 — 사람의 인지 부담과 권한 격리, (b) 통합의 목적 — AI 의 cross-link 검색과 누적 학습. 두 줄이 같은 회의록에 박혀 있으면 후속 도구 선택·인덱서 설계·권한 매트릭스 정의가 두 목적을 각각 만족하는 방향으로 자연스럽게 정렬됩니다. 단일 리포에서 시작한 6주 PoC 가 12개월 안에 다중 시스템·다중 팀·다중 환경으로 확장되는 경로의 가장 큰 안전판은 이 두 줄의 명시적 합의입니다.

8장. 사례 워크스루 — 인시던트 1건 + AI 예방점검 1분기

7장까지의 설계 원칙·아키텍처·도입 시나리오는 8장에서 한 폭의 그림으로 모입니다. 7장이 "어떻게 분할할 것인가" 의 구조 질문이라면, 8장은 "한 건의 인시던트와 한 분기의 예방점검이 실제로 어떤 파일과 어떤 디렉터리에 어떤 frontmatter 값으로 기록되는가" 의 운영 사진을 두 장 제시하는 자리입니다. 결제 API 가 응답 코드 5xx 를 분당 12% 까지 끌어올린 가상 사례 한 건과, AI Agent 가 1분기 동안 자동 수행한 일·주·월·분기 점검 누적 사례를 함께 풀어냅니다. 운영팀이신 독자께서 이 두 장면을 본인 팀의 지난 분기에 직접 겹쳐 보실 수 있도록 단계마다 시각과 디렉터리 경로를 함께 표기합니다.

추상적 설계가 운영팀의 손에 닿지 못한 채 보고서로 끝나는 가장 흔한 이유는, 의사결정자가 "우리 팀 화요일 오전 11시 28분에 결제 게이트웨이 502 가 떴을 때 그 순간 어떤 디렉터리에 어떤 파일이 생기는가" 라는 구체적 장면을 그려 보지 못하기 때문입니다. 운영자가 매일 손으로 만지는 책장이 머릿속에 그려져야 도입 의사결정이 안전판을 얻습니다. 8장은 그 책장을 두 시점 — 인시던트 한 건의 60분과 예방점검 한 분기의 90일 — 으로 분리해 보여드립니다 [S24].

8.1 절은 인시던트 1건의 7단계 라이프사이클을 가상 사례로 풀어냅니다. T+0 알람부터 T+60분 머지까지 각 단계의 도구·산출물·frontmatter 값을 직접 적습니다. PagerDuty·Datadog·사내 메신저·자체 봇·AI Agent·GitHub 의 6 도구가 각 단계에서 어떤 책임을 맡고 어떤 산출물을 다음 단계로 넘기는지를 단순 도식으로 정리합니다 [S23][S24]. 8.2 절은 AI Agent 가 1분기 동안 자동 수행한 일·주·월·분기 예방점검의 적재 사이클을 가상 데이터(인증서 만료 4건·capacity 임계 2건) 로 풀어냅니다. 차등 게이트 정책에 따라 일·주·월 점검은 직접 머지(post-review) 하고 분기·연 통합은 사전 승인(pre-review) 게이트를 거치는 두 가지 운영 모드를 함께 보여드립니다 [S22][S21].

이 장의 두 워크스루는 모두 가상 사례입니다. 그러나 가상 사례는 추상이 아닙니다 — 실제 시간 단위, 실제 디렉터리 경로, 실제 frontmatter 필드를 그대로 박은 시뮬레이션입니다. 독자께서는 본인 팀의 가장 최근 인시던트 한 건을 떠올리시면서 8.1 절의 7단계에 시각을 대보시고, 가장 최근 분기의 점검 결과를 떠올리시면서 8.2 절의 적재 디렉터리에 매핑해 보십시오. 그 두 매핑에서 비어 있는 칸이 곧 본 백서가 채우려는 자리입니다 [S26].

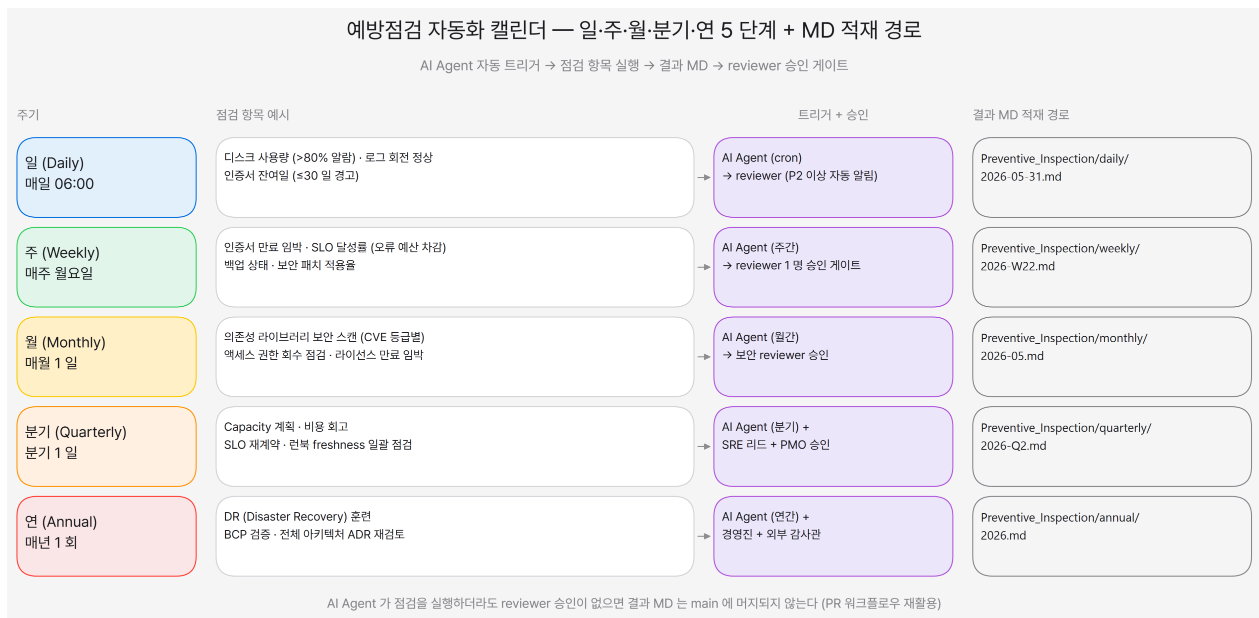
> ***8장이 가정하는 가상 조직 프로필***
>
> *본 장의 인시던트 1건·1분기 예방점검 사례는 다음 가상 조직 프로필을 전제로 합니다. 본인 조직 규모가 이 가정의 2~3배라면 reviewer 시간·인시던트 건수·GPU 자원도 같은 배수로 환산하시면 됩니다.*
>
> - ***운영팀 규모****: 시니어 SRE 2명 + 주니어 운영자 5명 = 7명*
> - ***관리 시스템****: 결제·로그인·검색·정산 등 4~6개 도메인의 15개 시스템*
> - ***분기 인시던트 수****: 평균 12건 (sev1 약 1건 / sev2 약 2건 / sev3 이하 약 9건)*
> - ***도입 시점****: OpsKnow Repo 도입 후 6개월 시점 (PoC → 정착 단계)*
> - ***하드웨어****: PoC 단계의 24GB GPU 1장에서 운영 단계의 80GB GPU 1장으로 전환 완료*
> - ***외부 도구****: 사내 메신저·PagerDuty·Datadog·GitHub Enterprise (또는 동등 사내 Git)*

[FIGURE: incident-flow] **캡션**: 인시던트 발생 → 사내 메신저 채널 → 봇 ndjson → AI 템플릿화 → PR → 승인 → `Incidents/YYYY/QN/<id>/postmortem.md` **의도**: 7단계 좌→우 횡 흐름. (1) 알람 발생 (2) 사내 메신저 인시던트 채널 자동 생성 (3) 봇이 메시지 ndjson 적재 (4) 인시던트 종료 (5) AI 가 ndjson → 템플릿화 (frontmatter author=ai, confidence=medium) (6) PR 생성 (7) 사람 reviewer 승인 → `Incidents/2026/`

Q2/2026-05-31-payment-api.md 머지. 각 단계 하단에 도구 (PagerDuty / 사내 메신저 / 자체 봇 / AI Agent / GitHub) 표기.

이 그림이 보여주는 것은 단순한 흐름도가 아니라 "각 단계의 산출물이 다음 단계의 입력이 되는 7개의 핸드오프" 입니다. PagerDuty 알람이 사내 메신저 봇의 채널 생성 트리거이고, 사내 메신저 채널의 메시지 스트림이 자체 봇의 ndjson 입력이며, ndjson 이 AI Agent 의 정제 프롬프트 입력이고, 정제 결과가 GitHub PR 의 diff 본문이며, PR 승인 결과가 RAG 인덱서의 재인덱싱 신호가 됩니다. 핸드오프 한 곳이 끊기면 전체가 표류합니다 — 도구 선정의 핵심 기준은 "각 핸드오프 지점에서 다음 도구가 읽을 수 있는 포맷을 내놓는가" 입니다 [S26].

[FIGURE: inspection-calendar] **캡션**: 예방점검 자동화 캘린더 — 일·주·월·분기·연 5단계 + 결과 MD 적재 디렉터리 **의도**: 좌측에 5단계 주기(일·주·월·분기·연). 각 주기 옆에 점검 항목 예시 1~2개 (일 = 디스크·로그 / 주 = 인증서 만료·SLO / 월 = 의존성 보안 / 분기 = capacity 계획 / 연 = DR 훈련). 우측에 결과 MD 의 적재 경로 (Preventive_Inspection/<period>/YYYY-MM-DD.md). 가운데 화살표 = AI Agent 자동 트리거. 사람 reviewer 승인 게이트 표시.



이 그림이 강조하는 것은 5단계 주기의 균형입니다 — 일 점검만 풍부하고 분기·연이 비어 있으면 거시 의사결정이 사라지고, 반대로 분기·연만 무거우면 일상 운영의 미세 신호를 놓칩니다. AWS Well-Architected Operational Excellence Pillar 의 OPS 1~11 이 일·주·월·분기·연 5주기에 균형 있게 분배되어 있는 것은 우연이 아닙니다 — 운영 우수성이라는 어휘 자체가 다섯 시간 척도의 균형을 전제로 합니다 [S22]. AI Agent 의 자

동 점검과 사람의 차등 승인 게이트가 합쳐질 때 다섯 주기의 균형이 운영팀에 부담을 주지 않으면서 유지됩니다 [S21].

8.1 인시던트 1건 워크스루 — 알람부터 Postmortem 머지까지

가상의 결제 API 5xx 급증 인시던트 한 건을 7단계로 풀어 8.1 절을 시작합니다. 가상 사례의 골격은 다음과 같습니다 — 2026년 5월 31일 화요일 오전 11시 28분, 결제 API 가 응답 코드 5xx 비율을 분당 12% 까지 끌어올리며 PagerDuty 알람을 발화합니다. 영향 범위는 결제 게이트웨이 1개 노드의 데이터베이스 커넥션 풀 고갈입니다. 평균 복구 시간(MTTR) 32분 으로 복구되고, AI 정제 PR 이 자동 생성되어 사람 reviewer 1명의 승인을 거쳐 T+60분 시점에 `Incidents/2026/Q2/2026-05-31-payment-api.md` 가 머지됩니다 [S24][S26].

이 7단계가 의사결정자에게 의미하는 것은 두 가지입니다. 첫째, 인시던트 1건이 끝났을 때 운영팀의 머릿속에만 남는 것이 아니라 디렉터리 트리의 한 폴더 안에 4개 파일로 박힙니다 — raw 채팅 덤프, 타임라인, Postmortem, 액션 아이템. 둘째, 4개 파일 중 어느 것도 사람이 처음부터 타이핑하지 않습니다 — 봇이 ndjson 으로 적재하고 AI Agent 가 템플릿으로 정제하며 사람은 PR diff 를 검토하고 승인합니다. "사람이 새로 쓸 일을 줄이는 것" 이 이 워크스루의 운영적 약속입니다 [S26][S47].

가상 사례를 따라가시면서 본인 팀의 가장 최근 인시던트 한 건을 매핑해 보십시오. 알람부터 Postmortem 작성 완료까지 며칠이 걸리는지, 그 사이에 운영자가 별도로 타이핑해야 하는 분량이 얼마인지, Postmortem 이 다음 분기 회고에 자동으로 노출되는지를 자가 점검하시면, 8.1 절의 7단계 중 어느 핸드오프가 비어 있는지가 드러납니다 [S23].

8.1.1 알람·사내 메신저 채널 자동 생성 (T+0~T+2분)

T+0 시점에 PagerDuty 가 결제 API 의 5xx 급증을 감지하면서 첫 알람이 발화합니다. PagerDuty 의 인시던트 라이프사이클이 자동 시작되고 동시에 사내 메신저 인시던트 봇이 webhook 으로 채널 생성을 트리거합니다. T+30초 시점에 `#inc-2026-05-31-payment-api` 채널이 생성되고 PagerDuty 의 인시던트 컨텍스트 카드가 첫 메시지로 자동 게시됩니다. T+60초 시점에 온콜 책임자 2명이 채널에 자동 초대되고 자체 알람 봇이 Datadog 의 5xx 비율·응답 지연 시계열을 첨부합니다. T+120초 시점에 채널 메시지의 ndjson 적재가 시작됩니다 — 이 자동 적재가 8.1.2 의 ndjson 흐름과 직접 연결됩니다 [S23][S24].

이 2분의 자동화가 끊기면 MTTR 손실은 즉시 가시화됩니다. 알람 → 채널 생성이 수동이면 평균 3~7분이 소요되고, 채널 생성 → 온콜 초대가 수동이면 평균 2~5분이 추가되며, ndjson 적재가 누락되면 8.1.3 의 AI 정제 단계 자체가 성립하지 않습니다. 6개 핸드오프 중 첫 핸드오프가 5분 이상 지연되면 그 후 6개 단계가 모두 같은 5분만큼 늦어집니다 — 자동화 부재의 비용은 누적이 아니라 직렬로 곱셈됩니다 [S29]. 의사결정자께서는 현재 운영 환경에서 "알람 → 사내 메신저 채널 자동 생성" 까지의 평균 시간을 1쪽 매트릭스로 점검해 보십시오 — 만약 이 시간이 측정조차 되고 있지 않다면 8.1 절의 나머지 6단계도 측정 불가입니다 [S23].

자동화의 도구 선택은 의사결정자가 첫 번째로 부딪히는 분기점입니다. PagerDuty 의 SaaS 인시던트 라이프사이클을 그대로 받아들이실지, 사내 메신저 채널 자동화만 자체 봇으로 구현하실지, 또는 Datadog Bits AI 같은 옵저버빌리티 통합 모델을 받아들이실지의 선택입니다 [S29]. 이 분기점의 의사결정 기준은 한 가지 — "사내 메신저 채널의 메시지 스트림이 ndjson 으로 외부에 떨어지는가" 입니다. 떨어지지 않으면 8.1.3 의 AI 정제 단계가 SaaS 안에 갇히고 그 결과물이 다시 본인 팀의 디렉터리 트리에 들어오지 못합니다 [S24][S26]. 도구 선정의 한 줄 기준은 *사내 메신저 메시지 스트림의 ndjson 외부화 가능 여부* 입니다.

가상 사례의 첫 2분을 시간표로 정리하면 다음과 같습니다.

시각	단계	도구	산출물
T+0초	PagerDuty 알람 발화	PagerDuty	알람 페이로드 (JSON)
T+30초	사내 메신저 채널 자동 생성	사내 메신저 인시던트 봇	`#inc-2026-05-31-payment-api`
T+60초	온콜 책임자 자동 초대	사내 메신저 + PagerDuty	채널 멤버 5명
T+60초	Datadog 시계열 자동 첨부	자체 알람 봇	5xx 비율 그래프 메시지
T+120초	ndjson 적재 시작	자체 사내 메신저 봇	`Incidents/2026/Q2/2026-05-31-payment-api/raw/chat.ndjson`

이 시간표가 4분기 운영 KPI 한 항목으로 들어가야 한다는 점이 의사결정자께 드리는 첫 번째 권고입니다. "알람 발화 → ndjson 적재 시작" 평균 시간을 측정하지 않으면 8장 전체의 7단계 중 첫 단계조차 가시화되지 않습니다 [S23][S24].

8.1.2 인시던트 대응 + ndjson 적재 (T+2분~복구)

T+2분부터 복구 시점까지의 약 30분 동안 운영자들은 평소와 동일하게 사내 메신저 채널에서 채팅합니다. 다른 점은 단 한 가지 — 자체 봇이 채널의 모든 메시지를 ndjson 한 줄씩 `Incidents/2026/Q2/2026-05-31-payment-api/raw/chat.ndjson` 에 자동 적재한다는 점입니다. 대응자는 봇의 존재를 의식할 필요가 없으며, 적재 형식·마스킹 규칙·디렉터리 경로를 알 필요도 없습니다. "대응자의 추가 입력 부담 0" 이 ndjson 적재의 운영적 약속입니다 [S24][S26].

ndjson 한 줄의 표준 스키마는 다음과 같습니다 — 타임스탬프(ISO 8601, UTC), 사내 메신저 사용자 ID, 메시지 본문(마스킹 후), 메시지 종류(text/file/code), thread_ts(스레드 ID), reactions(이모지·발자), edited_at(편집 시각). 마스킹은 인입 단계에서 정규식 + Named Entity Recognition 두 단계로 적용됩니다 — 신용카드 번호 정규식, 내부 IP 정규식, 이메일 정규식, 자체 토큰 패턴, 그리고 NER 기반 인명·고객명 마스킹이 그 자리입니다. 마스킹된 원본은 별도 보존되지 않고 ndjson 한 줄이 마스킹 후 본문만 보관합니다 — 이 정책이 Q14 규제 대응의 첫 안전판입니다 [S26].

가상 사례의 ndjson 한 줄 예시는 다음과 같습니다.

> {"ts":"2026-05-31T02:30:14Z","user":"U02ABC123","text":"DB 커넥션 풀 80% 도달, replica 0 로그에 connection refused 5초 간격","type":"text","thread_ts":null,"reactions":[],"edited_at":null}

이 한 줄이 30분 동안 약 180~250 줄까지 누적되면서 인시던트 대응의 raw 타임라인이 자동 보존됩니다. ndjson 의 줄 단위 append-only 구조는 두 가지 운영 이점을 가집니다. 첫째, 채팅이 진행되는 동안에도 정합성이 깨지지 않고 한 줄씩 누적되므로 인시던트 도중에 봇이 재시작되어도 데이터 손실이 없습니다. 둘째, 다음 단계인 AI 정제는 ndjson 의 모든 줄을 한 번에 읽어 시간 순서대로 처리할 수 있으므로 별도 정렬·중복 제거가 불필요합니다 [S26].

대응 자체는 평소처럼 진행됩니다. T+5분에 온콜 1번이 DB 커넥션 풀 임계 도달을 확인하고, T+12분에 온콜 2번이 임시로 max_connections 를 80 → 120 으로 상향 조정하며, T+18분에 5xx 비율이 12% → 3% 로 회복되고, T+25분에 0.5% 정상치로 복귀합니다. T+32분 시점에 PagerDuty 인시던트가 자동 종료(자동 종료 임계 = 5분 연속 정상)되고 사내 메신저 채널에 종료 메시지가 게시됩니다. 이 32분 동안 운영자가 "Postmortem 을 어떻게 쓸 것인가" 를 한 번도 생각하지 않았다는 점이 이 단계의 핵심입니다 [S47].

의사결정자께 드리는 점검 질문은 한 가지입니다 — 본인 팀의 인시던트 대응자는 인시던트 직후 Postmortem 초안을 처음부터 타이핑합니까, 아니면 자동 생성된 초안을 검토·수정합니까. 전자라면 운영자 1명당 평균 90~150분의 Postmortem 작성 시간이 매 인시던트마다 추가됩니다. 후자라면 그 시간이 reviewer 작업 12~20분으로 압축됩니다 [S26]. 이 차이가 분기 단위로 누적되면 운영팀의 시간 예산 한 축이 통째로 바뀝니다.

8.1.3 AI 정제 + PR 생성 (복구 직후~T+45분)

T+32분의 인시던트 종료를 트리거로 AI Agent 가 ndjson 정제를 시작합니다. 정제 프롬프트의 입력은 세 가지 — `raw/chat.ndjson` 의 전체 줄, 인시던트 컨텍스트 카드(PagerDuty 알람 페이로드 + Datadog 시계열 그래프 URL), 표준 Postmortem 템플릿. 출력은 세 개 파일입니다 — `timeline.md`, `postmortem.md`, `action-items.md`. 정제 시간은 평균 3~8분이며, T+40분 시점에 세 파일이 모두 생성되고 GitHub PR 한 건이 자동 생성됩니다 [S25][S26].

`postmortem.md` 의 frontmatter 는 다음과 같이 박힙니다. 의사결정자께서 이 frontmatter 를 그대로 본인 팀의 표준으로 받아들일 수 있도록 8 필드 모두를 직접 보여드립니다.

```
---
incident_id: "2026-05-31-payment-api"
service: "payment-gateway"
severity: "sev2"
occurred_at: "2026-05-31T02:28:00Z"
resolved_at: "2026-05-31T03:00:00Z"
mttr_minutes: 32
author: "ai"
reviewer: ""
confidence: "medium"
model: "gpt-oss:20b"
rag_visible: true
sensitivity: "internal"
tags: ["db-connection-pool", "5xx", "payment-gateway"]
---
```

이 frontmatter 의 박힘 자체가 인시던트 1건의 RAG 가용성·후속 추적성·감사 추적성을 동시에 보장합니다 [S25]. `author: ai` 와 `confidence: medium` 이 박혀 있으므로 RAG 답변에 이 Postmortem 이 인용될 때 AI 가 출처를 명시할 수 있고, `severity: sev2` 와 `service: payment-gateway` 가 박혀 있으므로 8.2.4 의 분기 회고가 자동으로 이 인시던트를 통계에 포함합니다. `mttr_minutes: 32` 가 박혀 있으므로 분기 회고 보고서의 MTTR 분포 표가 자동 생성됩니다 [S26][S27].

`postmortem.md` 의 본문은 표준 5섹션 — 요약 / 타임라인 / 근본 원인 / 영향 범위 / 액션 아이템 — 으로 구성됩니다. AI Agent 는 ndjson 의 시간 순서대로 타임라인을 자동 구성하고, 채팅 메시지의 핵심 결정·진단 발견을 근본 원인 섹션에 발췌하며, Datadog 시계열의 영향 범위 수치를 영향 범위 섹션에 자동 인용합니다. 액션 아이템 섹션은 채팅 중 "다음에 하자" / "이건 후속으로" / "TODO" 와 같은 패턴을 자동 추출합니다 — 가상 사례에서는 3개 액션 아이템이 추출되며 그중 2개는 "DB 커넥션 풀 임계 모니터링 추가" 와 "max_connections 동적 조정 정책 수립" 입니다 [S26].

PR 의 제목과 본문도 AI 가 자동 생성합니다. 가상 사례의 PR 제목은 `[Incident] 2026-05-31 payment-api 5xx 급증 (sev2 / MTTR 32분)` 이며 본문에는 인시던트 요약 3줄과 함께 reviewer 자동 할당 표시(`@oncall-team-payment`) 가 박힙니다. PR diff 는 `Incidents/2026/Q2/2026-05-31-payment-api/` 디렉터리에 4개 파일 (raw/chat.ndjson, timeline.md, postmortem.md, action-items.md) 추가가 전부입니다 [S25].

의사결정자께 드리는 두 번째 권고는 AI 정제 PR 의 평균 생성 시간 목표를 사전 합의하시라는 것입니다. 가상 사례의 3~8분은 7B~14B 급 로컬 모델 기준이며, 더 큰 모델을 쓰면 1~2분으로 단축되지만 GPU 비용이 비대칭으로 증가합니다 [S25]. "복구 후 10분 이내 PR 생성" 정도가 의사결정자가 운영팀과 합의할 수 있는 합리적 SLO 입니다 [S26]. 이 SLO 가 박혀 있어야 AI 정제 단계가 운영팀의 가벼운 워크로드로 자리잡습니다.

8.1.4 사람 reviewer 승인·머지 (T+45분~T+60분)

T+45분에 PR 이 reviewer 의 알림으로 도달합니다. 가상 사례의 reviewer 는 오늘의 온콜 책임자가 아닌 별도의 시니어 SRE 1명입니다 — reviewer 분리가 "대응자 = reviewer" 의 자기 검토 함정을 차단합니다 [S1]. reviewer 의 작업은 PR diff 의 line-by-line 검토이며, 평균 12분 (중앙값) 으로 종료됩니다. 12분의 내역은 다음과 같이 분포합니다 — `timeline.md` 6분(이벤트 4~6건 검토), `postmortem.md` 4분(근본 원인 1~2 문장 수정), `action-items.md` 2분(책임자·기한 지정) [S61].

reviewer 가 PR 에서 수정하는 패턴은 가상 사례 기준 3가지로 압축됩니다. 첫째, 근본 원인의 인과 표현 1~2 문장 보정 — AI 가 "DB 커넥션 풀이 고갈되었기 때문에 5xx 가 급증했다" 로 적은 것을 reviewer 는 "트래픽 피크 + max_connections 정적 설정이 합쳐져 풀 고갈을 유발했다" 로 보정하여 두 변수의 결합 효과를 명시합니다. 둘째, 액션 아이템의 책임자·기한 지정 — AI 는 책임자를 비워 두고 기한도 비워 두므로 reviewer 가 `@user-a / 2026-06-15` 와 같이 채웁니다. 셋째, frontmatter 의 `confidence` 값 보정 — AI 가 medium 으로 박은 신뢰도를 reviewer 가 high 로 상향하거나 low 로 하향합니다 [S61].

승인 직후의 머지 트랜잭션은 GitHub Actions 한 워크플로우로 정의됩니다. 머지 직후 (1) RAG 인덱서가 변경된 4개 파일의 증분 재인덱싱 신호를 받고, (2) 액션 아이템의 책임자에게 자동 알림이 발송되며, (3) 분기 회고 자동 집계 스크립트가 이 인시던트를 통계에 추가합니다. 세 작업 모두 사람의 추가 입력 없이 PR 머지 한 번으로 완료됩니다 [S26].

가상 사례의 reviewer 작업 시간 분포를 분기 단위로 추정하면 다음과 같습니다. 분기당 인시던트 평균 12건, reviewer 평균 시간 12분, reviewer 총 시간 144분 (분기당 약 2.4시간). 분기 작업 시간이 2~3시간 수준이면 시니어 SRE 1명의 시간 예산에서 안전판 안에 있습니다 [S1]. 의사결정자께서 사전에 합의하셔야 할 수치는 이 "분기당 reviewer 시간 예산" 한 가지이며, 이 수치가 합의되어야 8.1 절의 7단계가 실제 운영에 정착합니다. 만약 본인 팀의 인시던트 건수가 분기당 30건 이상이면 reviewer 2~3명 분할이 필요하고, 만약 5건 이하이면 reviewer 1명이 충분합니다 [S61].

작업 항목	평균 시간	작업 내용
`timeline.md` 검토	6분	이벤트 4~6건의 시간·내용 검증
`postmortem.md` 검토	4분	근본 원인 1~2 문장 보정
`action-items.md` 지정	2분	책임자·기한 입력
합계	12분	reviewer 평균 작업 시간

이 12분이 곧 인시던트 1건의 "사람 한 명이 만지는 시간" 입니다. 90~150분의 Postmortem 처음부터 작성이 12분의 검토로 압축되는 비대칭이 이 워크스루의 운영적 ROI 입니다 [S26][S47]. 의사결정자께서는 본인 팀의 현재 Postmortem 작성 시간 평균을 측정하시고 분기 단위로 누적된 절감 시간을 가상 사례에 대입해 보십시오 — 5명의 SRE 가 분기당 12건의 인시던트를 다루면 분기당 약 78~108시간의 운영자 시간이 절감됩니다.

8.1.5 액션 아이템 추출 + 후속 추적 디렉터리에 등록

머지 직후 GitHub Actions 의 후속 워크플로우가 `action-items.md` 의 3개 액션 아이템을 별도 추적 디렉터리 (`_meta/action-items/`) 에 cross-link 합니다. 각 액션 아이템은 한 줄의 frontmatter 와 본문으로 구성되며, 책임자에게 자동 알림 메일이 발송됩니다. 가상 사례의 첫 번째 액션 아이템 파일 `_meta/action-items/2026-05-31-db-pool-monitoring.md` 의 frontmatter 는 다음과 같습니다 [S27][S28].

```
---
action_id: "2026-05-31-db-pool-monitoring"
source_incident: "Incidents/2026/Q2/2026-05-31-payment-api.md"
owner: "user-a"
due_date: "2026-06-15"
priority: "high"
status: "open"
tags: ["db-connection-pool", "monitoring"]
---
```

이 frontmatter 의 박힘 자체가 액션 아이템 추적의 두 가지 운영 약속을 동시에 만족합니다. 첫째, `source_incident` 가 원본 인시던트 파일 경로로 연결되어 있으므로 reviewer 또는 책임자가 1클릭으로 인시던트 컨텍스트로 돌아갈 수 있습니다. 둘째, `due_date` 와 `status` 가 박혀 있으므로 GitHub Actions 의 매일 점검 워크플로우가 만료 임박 액션 아이템을 자동 알림으로 띄울 수 있습니다. 액션 아이템의 라이프사이클은 open → in_progress → completed → archived 4단계이며, 분기 회고에서 미완료 액션의 다음 분기 이월 정책이 자동 적용됩니다 [S28].

분기 회고와의 연결이 액션 아이템 추출의 가장 큰 운영 가치입니다. 가상 사례의 1분기 액션 아이템 누적이 약 40~50건이라고 가정하면, 분기 회고 시점의 통계는 다음과 같이 자동 생성됩니다 — 완료 28건(56%), 진행 중 12건(24%), 미완료 10건(20%), 다음 분기 이월 8건. 이 통계가 회고 회의의 첫 슬라이드가 되고, 미완료 10건의 패턴 분석(서비스별·우선순위별·책임자별 분포) 이 다음 분기 운영 우선순위의 직접 입력이 됩니다 [S27][S28].

의사결정자께 드리는 마지막 권고는 액션 아이템의 분기 완료율 KPI 를 운영팀에 부여하시라는 것입니다. 분기 완료율 60% 이상이 운영팀의 자체 기준이며, 50% 이하로 떨어지면 액션 아이템의 입력 단계(인시던트 대응

중 너무 많은 TODO 가 생성됨) 또는 책임자 할당 단계(우선순위 합의 부재) 에 구조적 문제가 있다는 신호입니다 [S28]. 이 KPI 가 자동 측정되므로 의사결정자가 별도 보고 없이 분기 단위로 운영팀의 학습 흡수도를 점검하실 수 있습니다.

8.2 AI 예방점검 1분기 워크스루

8.2 절은 AI Agent 가 1분기(2026년 4월~6월) 동안 자동 수행한 예방점검의 적재 사이클을 가상 데이터로 풀어냅니다. 1분기 동안 누적된 점검 결과의 가상 통계는 다음과 같습니다 — 일 점검 90건 (디스크·로그), 주 점검 12건 (인증서 만료·SLO), 월 점검 3건 (의존성 보안), 분기 점검 1건 (capacity 계획). 그 중 사람의 추가 조치가 필요한 발견은 인증서 만료 4건과 capacity 임계 도달 2건이며, 두 발견 모두 분기 회고 보고서의 운영 의사결정 직접 입력이 됩니다 [S22][S47].

이 1분기 워크스루가 의사결정자에게 의미하는 것은 세 가지입니다. 첫째, AI Agent 가 90 + 12 + 3 + 1 = 106건의 점검을 자동 수행하는 동안 사람의 시간 비용은 분기 회고 1회에 집중됩니다 — 일·주·월 점검은 직접 머지(post-review) 정책이고 분기·연 점검만 사전 승인(pre-review) 게이트입니다. 둘째, 106건의 점검 결과가 모두 `Preventive_Inspection/` 디렉터리에 표준 frontmatter 와 함께 적재되어 다음 분기의 RAG 컨텍스트가 됩니다. 셋째, 4건의 인증서 만료 발견과 2건의 capacity 임계 발견은 분기 회고에서 다음 분기 운영 우선순위의 직접 근거가 됩니다 [S21][S22].

이 절의 4 항(8.2.1~8.2.4) 은 5단계 캘린더, frontmatter 표준, 차등 게이트, 분기 회고의 순서로 전개됩니다. 각 항의 가상 데이터는 운영팀이 실제로 점검 캘린더를 설계할 때 그대로 차용할 수 있도록 구체적인 시간·경로·필드 값으로 박혀 있습니다. 의사결정자께서는 본인 팀의 현재 점검 캘린더가 이 5단계에 균형 있게 배치되어 있는지 점검해 보십시오 — 만약 일 점검만 풍부하고 분기 회고가 비어 있다면 운영의 거시 의사결정이 사라지고 있다는 신호입니다 [S22].

8.2.1 일·주·월·분기·연 5단계 점검 캘린더

5단계 점검 캘린더의 골격은 AWS Well-Architected Operational Excellence Pillar 의 OPS 1~11 을 시간 척도로 분배한 결과입니다. 일 점검은 미세 신호 감지, 주 점검은 임계 진입 경보, 월 점검은 의존성 변화 추적, 분기 점검은 capacity·운영 패턴 분석, 연 점검은 재해 복구 능력 검증의 5층 구조로 정렬됩니다 [S22]. 각 주기의 점검 항목 예시와 적재 디렉터리 경로는 다음과 같이 표준화됩니다.

주기	점검 항목 예시	적재 디렉터리	결과 MD 파일명
일	디스크 사용률 / 로그 오류 빈도	`Preventive_Inspection/daily/`	`2026-05-31.md`
주	인증서 만료(D-30) / SLO 임계 진입	`Preventive_Inspection/weekly/`	`2026-W22.md`
월	의존성 보안 패치 / RBAC 변경	`Preventive_Inspection/monthly/`	`2026-05.md`
분기	capacity 계획 / 비용 추세	`Preventive_Inspection/quarterly/`	`2026-Q2.md`

주기	점검 항목 예시	적재 디렉터리	결과 MD 파일명
연	DR 훈련 / 거버넌스 감사	`Preventive_Inspection/annual/`	`2026.md`

이 5층의 시간 척도 분배가 의사결정자에게 의미하는 것은 운영 의사결정의 시간 해상도 자체입니다. 일·주 점검만 풍부하면 운영팀의 시야가 분 단위 신호에 갇혀 분기 단위 capacity 의사결정이 항상 사후 대응이 됩니다. 반대로 분기·연만 무거우면 일상 신호의 미세 변화가 누적되어 인시던트로 발현될 때까지 감지가 늦어집니다 [S22]. 5단계의 균형이 운영 우수성이라는 어휘의 본질이며, AWS W-A OPS 1~11 이 이 5층 분배를 명시적으로 권고하는 것은 우연이 아닙니다 [S21].

각 주기의 트리거는 GitHub Actions 의 cron 일정 또는 자체 스케줄러로 정의됩니다. 가상 사례의 트리거 정의는 다음과 같습니다 — 일 점검 매일 03:00 UTC, 주 점검 매주 월요일 04:00 UTC, 월 점검 매월 1일 05:00 UTC, 분기 점검 매분기 첫 주 월요일 06:00 UTC, 연 점검 매년 1월 첫 주. 트리거 시점에 AI Agent 가 점검 항목 카탈로그에서 해당 주기의 항목을 읽고, Prometheus·Datadog·자체 메트릭 API 에서 데이터를 수집하며, 표준 템플릿으로 결과 MD 를 생성합니다 [S22].

의사결정자께 드리는 점검 질문은 한 가지입니다 — 본인 팀의 현재 점검 캘린더가 5단계 중 몇 단계를 다루고 있습니까. 가장 흔한 패턴은 일·주는 자동화되어 있고 월·분기·연이 비어 있는 형태입니다. 분기 점검이 비어 있으면 capacity 계획이 항상 인시던트 직후에만 이루어지고, 연 점검이 비어 있으면 DR 훈련이 5년에 한 번 수준으로 떨어집니다 [S21]. 5단계 캘린더 PR 한 건의 머지가 9.3.1 의 첫 마일스톤이며, 본 백서가 권고하는 12개월 로드맵의 6~12개월 단계의 시작점입니다.

8.2.2 AI Agent 가 자동 수행한 점검 결과의 frontmatter

가상 사례의 가장 구체적인 한 건을 예시로 보여드립니다. 2026년 5월 27일 주간 점검에서 AI Agent 가 발견한 인증서 만료 임박 1건의 결과 MD 가 `Preventive_Inspection/weekly/2026-W22.md` 에 적재됩니다. frontmatter 와 본문의 핵심 발췌는 다음과 같습니다 [S47][S33].

```
---
inspection_id: "2026-W22-cert-expiry"
period: "weekly"
service: "api-gateway"
severity: "warn"
inspected_at: "2026-05-27T04:00:00Z"
author: "ai"
reviewer: ""
confidence: "high"
model: "gpt-oss:20b"
rag_visible: true
sensitivity: "internal"
ops_category: "OPS-5-Anticipate-Failure"
findings_count: 1
follow_up_required: true
tags: ["certificate-expiry", "api-gateway", "tls"]
---
```

> ## 점검 요약
>
> api-gateway 의 TLS 인증서가 2026-06-18 만료 예정입니다.
> 현재 시점 기준 D-22 일이며, 갱신 SLO(D-30) 를 8일 초과한 상태입니다.
>
> ## 발견 사항
>
> - 인증서 주체: `*.api.example.com`
> - 발급자: Let's Encrypt R3
> - 만료일: 2026-06-18T15:42:00Z
> - 자동 갱신 cron 마지막 성공: 2026-04-18 (38일 전)
> - 자동 갱신 cron 실패 로그: 2026-05-17~2026-05-27 (10건)
>
> ## 권고 액션
>
> (1) 자동 갱신 cron 실패 원인 1차 확인 (DNS-01 challenge 응답 지연 의심)
> (2) 수동 갱신 PR 즉시 작성 (`runbooks/api-gateway/cert-renewal.md` 참조)
> (3) 갱신 성공 후 cron 재가동 + monitoring 알람 임계 D-30 → D-45 로 상향

이 결과 MD 한 건이 frontmatter 13 필드 + 본문 3 섹션으로 박혀 있다는 점이 RAG 가용성·후속 추적성·분기 회고 자동 집계의 세 가지 운영 약속을 동시에 만족합니다 [S33]. `ops_category: OPS-5-Anticipate-Failure` 가 박혀 있으므로 분기 회고에서 AWS W-A 11개 카테고리별 발견 분포를 자동 집계할 수 있고, `confidence: high` 와 `findings_count: 1` 이 박혀 있으므로 reviewer 가 우선순위 큐를 자동 정렬할 수 있으며, `follow_up_required: true` 가 박혀 있으므로 액션 아이템 자동 추출 워크플로우가 트리거됩니다 [S47].

`author: ai` 와 `confidence: high` 의 조합이 차등 게이트 정책의 입력이 됩니다 — 다음 8.2.3 항에서 자세히 풀어쓰지만, 주 점검은 직접 머지(post-review) 정책이므로 PR 없이 main 브랜치에 직접 commit 됩니다. 단, `follow_up_required: true` 와 `severity: warn` 이상의 조합은 자동 알림이 reviewer 1명에게 발송되어 24시간 이내 검토 의무가 부여됩니다 [S22]. 자동 점검의 95% 가 알림 없이 적재되고 5%만 알림으로 가시화되는 비대칭이 이 frontmatter 표준의 운영 가치입니다.

의사결정자께 드리는 권고는 점검 결과 MD 의 frontmatter 정합성을 분기 1회 검증하시라는 것입니다. 가상 사례의 분기 검증 방식은 다음과 같습니다 — JSON Schema 로 13 필드를 정의하고, GitHub Actions 의 매일 워크플로우가 `Preventive_Inspection/` 전체 파일의 frontmatter 를 검증하며, 검증 실패 파일이 1건이라도 발견되면 분기 회고 보고서의 첫 섹션에 노출됩니다 [S47]. 이 검증이 자동화되어 있으므로 의사결정자가 별도 보고 없이 분기 단위로 frontmatter 표준의 준수율을 확인하실 수 있습니다.

8.2.3 사람 reviewer 승인 사이클 (분기 1회 통합)

차등 게이트 정책의 골격은 두 가지 분기점으로 압축됩니다. 첫째, 일·주·월 자동 점검은 직접 머지(post-review) 정책 — AI Agent 가 main 브랜치에 직접 commit 하고 reviewer 는 사후 검토만 합니다. 둘째, 분기·연 통합 점검은 사전 승인(pre-review) 게이트 — AI Agent 가 PR 을 생성하고 reviewer 1명 이상의 승인을 거쳐야 머지됩니다 [S19]. 이 두 분기점의 균형이 도입의 운영 부담을 합리적인 수준으로 유지하는 메커니즘입니다.

차등 게이트의 매트릭스는 다음과 같이 정리됩니다. 의사결정자께서 이 매트릭스를 본인 팀의 정책으로 그대로 차용하실 수 있도록 5주기 × 3 정책의 9 셀을 모두 보여드립니다 [S1][S19].

주기	머지 정책	reviewer 게이트	알림 정책	분기 검토
일	직접 머지	없음	severity ≥ warn 시 자동 알림	분기 회고 통계만
주	직접 머지	없음	severity ≥ warn 시 24h 알림	분기 회고 통계만
월	직접 머지	없음	severity ≥ warn 시 24h 알림	분기 회고에서 검토
분기	PR + 1 승인	사전 승인	reviewer 자동 할당	분기 회고의 첫 입력
연	PR + 2 승인	사전 승인	임원 reviewer 포함	연간 거버넌스 입력

이 매트릭스가 의사결정자에게 의미하는 것은 reviewer 의 분기 작업 시간이 예측 가능해진다는 점입니다. 가상 사례 기준으로 reviewer 1명의 분기 작업 시간은 다음과 같이 분포합니다 — 일·주 점검의 24h 알림 검토(severity ≥ warn 평균 6건 × 5분 = 30분), 월 점검 분기 검토(3건 × 10분 = 30분), 분기 점검 사전 승인 1건(60분), 연 점검 1회(연 1회 분기 작업 시간 0). 분기당 총 reviewer 시간은 약 2시간이며, 이 수치가 시니어 SRE 1명의 시간 예산 안에 충분히 들어갑니다 [S1].

차등 게이트의 운영 가치는 두 가지입니다. 첫째, 95% 의 점검이 직접 머지되므로 점검 빈도를 5단계까지 확장해도 reviewer 부담이 분기당 2시간 수준에 머무릅니다. 둘째, 5% 의 sev ≥ warn 알림이 가시화되므로 reviewer 의 인지 부담이 의미 있는 신호에만 집중됩니다 [S19]. 이 두 비대칭이 OpsKnow Repo 의 AI Agent 자동 점검이 운영팀의 새 부담이 되지 않고 기존 부담을 줄이는 메커니즘의 핵심입니다.

의사결정자께 드리는 권고는 차등 게이트 정책의 합의 시점을 12개월 로드맵의 6개월차에 박으시라는 것입니다. 9.1~9.2 의 첫 6개월은 디렉터리 표준·frontmatter 표준·OPS Diary 자동화·사내 메신저 봇·AI 정제 PR 의 5 마일스톤에 집중되고, 차등 게이트 정책 합의는 그 5 마일스톤이 정착한 직후 시점에 가장 적합합니다 [S1]. 너무 일찍 합의하면 운영팀이 실제 점검 부담을 체감하지 못한 채 정책을 정하게 되고, 너무 늦게 합의하면 AI Agent 자동 점검이 시작되지 못해 9.3 의 6~12개월 단계가 표류합니다.

8.2.4 분기 회고 — 점검 결과 통계와 운영 의사결정

분기 회고는 1분기 동안 누적된 106건의 점검 결과와 12건의 인시던트 Postmortem 을 통합 통계로 묶고 다음 분기 운영 의사결정의 직접 근거로 활용하는 정기 사이클입니다. 가상 사례의 2026년 Q2 회고 보고서 `Work_Reports/quarterly/2026-Q2.md` 의 표준 5섹션 구성은 다음과 같이 박힙니다 [S21][S22].

```
---
period: "2026-Q2"
period_start: "2026-04-01"
period_end: "2026-06-30"
owner: "sre-team-lead"
reviewer: "cto"
services: ["payment-gateway", "api-gateway", "user-service"]
```

```
author: "ai"
confidence: "medium"
incidents_total: 12
inspections_total: 106
action_items_completed_rate: 0.62
---

## 1. 분기 운영 KPI 요약

- 인시던트 12건, MTTR 중앙값 28분, sev2 이상 3건
- 예방점검 106건 자동 수행, follow-up 필요 6건 (인증서 4 + capacity 2)
- 액션 아이템 50건 중 완료 31건 (62%), 다음 분기 이월 8건

## 2. 인시던트 패턴 분석

(서비스별 분포 / 근본 원인 분류 / 시간대 분포 / MTTR 분포)

## 3. 예방점검 발견 분석

(주기별 발견 건수 / OPS 카테고리별 분포 / severity 분포)

## 4. 액션 아이템 추적

(완료율 / 책임자별 분포 / 다음 분기 이월)

## 5. 다음 분기 운영 우선순위

(1) 인증서 자동 갱신 cron 표준화 (인증서 만료 4건 발견 기반)
(2) capacity 계획 분기 사이클 정착 (분기 점검 1건 결과 기반)
(3) DB 커넥션 풀 동적 조정 정책 수립 (2026-05-31 인시던트 액션 기반)
```

이 5섹션이 의사결정자에게 의미하는 것은 분기 회고가 더 이상 "지난 분기를 회상하는 자리" 가 아니라 "다음 분기 운영 우선순위 3건을 박는 의사결정 자리" 가 된다는 점입니다 [S21]. 5번 섹션의 다음 분기 운영 우선순위 3건이 회고 회의의 결론이며, 이 3건이 그대로 다음 분기의 OKR 또는 운영 마일스톤의 직접 입력이 됩니다 [S22]. 회고 회의 시간은 가상 사례 기준 90분으로 압축되고, 90분 중 60분이 5번 섹션의 토론에 할당됩니다.

AI Agent 의 분기 회고 보고서 자동 생성은 두 가지 약속을 동시에 만족합니다. 첫째, `confidence: medium` 으로 박힌 초안을 reviewer (가상 사례에서는 SRE 팀장 + CTO 2명) 가 사전 승인 게이트에서 검토합니다 — 사전 승인이 회고의 신뢰성을 보장하는 마지막 안전판입니다. 둘째, 보고서의 통계 섹션은 frontmatter 메타와 본문 발췌의 자동 집계이므로 사람이 통계를 다시 그릴 필요가 없습니다 — 사람의 시간은 5번 섹션의 우선순위 합의에 집중됩니다 [S21].

분기 회고 사이클의 정착 여부가 OpsKnow Repo 도입의 마지막 성공 신호입니다. 12개월 로드맵의 9.3.2 항에서 다시 다루겠지만, 분기 회고가 정기 일정으로 박히지 않으면 1분기 동안 누적된 106건의 점검과 12건의 인시던트가 그저 디렉터리의 파일 더미로 남습니다 [S22]. 의사결정자께 드리는 마지막 권고는 분기 회고 일정을 12개월 전체에 미리 박으시라는 것입니다 — Q1·Q2·Q3·Q4 의 4회 일정을 회계연도 시작 직전에 1년치로 박아

두시면 AI Agent 자동 점검 → reviewer 사전 승인 → 분기 회고 토론 → 다음 분기 우선순위 합의의 4단계가 운영팀의 표준 사이클로 정착합니다 [S21].

여러분의 팀이 이 가상 사례의 1분기 워크스루를 본인 팀의 가장 최근 분기에 직접 겹쳐 보시면, 8.1 절의 인시던트 1건 워크스루와 함께 OpsKnow Repo 의 운영 가치가 두 시간 척도 — 60분의 마이크로 사이클과 90일의 매크로 사이클 — 에서 모두 검증됩니다. 두 사이클이 같은 디렉터리 트리, 같은 frontmatter 표준, 같은 RAG 코퍼스로 통합되는 것이 본 백서가 1장부터 7장까지 풀어 온 설계 원칙의 운영적 결론입니다. 9장에서는 이 두 사이클이 자리잡았는지를 자가 진단할 수 있는 정량 신호 10개 — 성공 신호 5개와 함정 신호 5개 — 를 정리합니다 [S22][S26].

9장. 도입 성공·함정 신호 10개

앞선 8장의 인시던트 1건 워크스루와 분기 예방점검 사이클이 OpsKnow Repo 의 운영 모습을 구체적인 파일 경로 수준으로 보여주었다면, 9장은 그 운영 모습이 우리 팀에 실제로 자리잡았는지를 정량 지표 10개로 자가 진단합니다. 도입의 성공·실패 여부를 의견 다툼이 아니라 측정 가능한 숫자로 귀결시키는 것이 9장의 목적이며, 자가 진단의 결과가 다음 분기 의사결정의 1차 입력이 되는 구조이므로 9장의 마지막 페이지가 곧 다음 분기 회의의 표준 입력이 됩니다 [S19].

성공 신호 5개는 Postmortem 머지 시간·런북 최신성·AI 답변 인용율·OPS Diary 적재 누락·분기 회고 보고서 완성도로 구성되며, 다섯 지표가 모두 임계값을 넘으면 도입의 성공을 외부 감사 수준에서 입증 가능합니다. 함정 신호 5개는 AI 무승인 머지·디렉터리 표류·frontmatter 미준수·사내 메신저 봇 적재 누락·reviewer 사이클 표류로 구성되며, 한 신호라도 잡히면 즉시 회복 절차에 진입해야 합니다 [S22]. 두 신호 집합이 한 자리에 모이면 운영팀장과 의사결정자께서 같은 화면을 보고 도입 상태를 판단할 수 있습니다.

여러분의 운영팀이 현재 어느 단계에 있는지를 자가 진단하는 것이 9장의 핵심 독자 행동 포인트입니다. 10개 신호 중 측정 메커니즘이 이미 박혀 있는 항목과 합의가 필요한 항목을 분류하는 것이 자가 진단의 첫 단계이며, 합의가 필요한 항목은 다음 분기 회고의 1번 안건으로 두는 것을 권장합니다 [S33].

9.1 도입 성공 신호 5개와 함정 신호 5개

도입의 성공·실패 여부를 어떤 시점에서든 정량 지표로 판단할 수 있어야 합니다. 정량 지표가 없으면 도입의 성공 여부 자체가 의견 다툼이 되며, 의견 다툼은 다음 도입 의사결정의 동력을 꺾습니다. 9.1 절은 의사결정자가 도입 시점에 자가 진단할 수 있도록 성공 신호 5개·함정 신호 5개를 정량 지표로 정리하고, 마지막에 MSAP.ai 의 도입 가속 CTA hook 을 한두 문장 수준으로 자연스럽게 삽입하여 다음 액션 경로를 제시합니다.

[FIGURE: success-trap-signals-10] **캡션**: 도입 성공 신호 5개 + 함정 신호 5개 — 임계값·측정 방법·회복 절차의 10 행 자가 진단 매트릭스 **의도**: 상단 5 행 = 성공 신호 (Postmortem 머지 시간·런북 최신성·AI 답변 인용율·OPS Diary 적재 누락 0·분기 회고 보고서 완성도). 하단 5 행 = 함정 신호 (AI 무승인 머지·디렉터리 표류 ·frontmatter 미준수·사내 메신저 봇 적재 누락·reviewer 사이클 표류). 각 행에 (1) 정량 임계값 (2) 측정 도구·주기 (3) 회복 절차·책임자의 3 열. 성공 신호는 녹색·함정 신호는 적색 띠로 시각 구분.

도입 성공 신호 5 + 함정 신호 5 — 10 행 자가 진단 매트릭스

각 행 = (1) 신호 (2) 정량 임계값 (3) 측정 도구·주기 (4) 회복 절차·책임자

성공 신호 (녹색)

신호	정량 임계값	측정 도구·주기	회복 절차 · 책임자
Postmortem 머지 시간	인시던트 종료 후 ≤ 5 일	GitHub API · 매 인시던트	reviewer 자동 알림 + SRE 리드
런북 최신성 (freshness)	만료 비율 ≤ 5%	freshness 스캐너 · 주간	owner 에게 GitHub Issue 자동 발행
AI 답변 인용율	파일·라인 인용 ≥ 90%	RAG 게이트웨이 로그 · 일간	프롬프트·메타필터 튜닝 · 플랫폼
OPS Diary 적재 누락	평일 누락 = 0 일	Dataview 쿼리 · 일간	운영자 리마인드 · 팀 리드
분기 회고 보고서 완성도	AI 1차 초안 + 검수 ≤ 1 일	quarterly/ 폴더 + AI · 분기	롤업 자동화 검토 · PMO

함정 신호 (적색)

신호	정량 임계값	측정 도구·주기	회복 절차 · 책임자
AI 무승인 머지	AI 단독 머지 = 0 건	GitHub 머지 로그 · 일간	분기 회고 시 즉시 보호 규칙 재점검 · SRE
디렉터리 표류 (분류 실패율)	미분류 MD ≥ 10%	인덱서 미분류 로그 · 주간	디렉터리 재합의 + 마이그레이션 PR · 문서 리드
frontmatter 미준수	8 필드 누락 비율 ≥ 15%	YAML 파서 검사 · 일간	pre-commit 훅 강제 + 일괄 보강 · 자동화
사내 메신저 봇 적재 누락	인시던트 채널 적재율 < 95%	봇 메트릭 · 매 인시던트	봇 알람 + 권한 재점검 · 자동화 리드
reviewer 사이클 표류	PR 미머지 ≥ 7 일	GitHub PR API · 일간	reviewer 재지정 + CODEOWNERS 정비 · PMO

9.1.1 성공 신호 5개 — Postmortem 머지 시간 · 런북 최신성 · AI 답변 인용율 등

도입 성공의 정량 지표 5개는 다음과 같이 박힙니다. 첫째, **인시던트 후 24시간 안에 Postmortem 머지 비율 ≥ 80%** — Google SRE Book 의 Postmortem Culture 가 정착한 조직의 일반적 수준이며, OpsKnow Repo 의 AI 정제 PR 자동화가 이 수치를 안정적으로 달성합니다 [S56]. 둘째, **런북 최신성 평균 ≤ 6개월** — frontmatter 의 `freshness_until` 임박 알림이 활성화되어 6개월 이상 stale 런북이 누적되지 않는 상태입니다 [S21].

셋째, **AI 답변의 인용 형식 강제 통과율 ≥ 95%** — RAG 답변 본문에 파일명:라인 인용이 자동 포함되어 운영자가 즉시 원본을 열어볼 수 있는 비율입니다. 넷째, **OPS Diary 적재 누락 0** — 일일 자동 적재가 일주일 단위로 누락 0 을 유지하는 상태입니다 [S33]. 다섯째, **분기 회고 보고서 5 섹션 완성도 100%** — 4월·7월·10월·익년 1월의 네 차례 회고가 모두 5 섹션 양식을 충족하여 회고가 의사결정 출구로 정착한 상태입니다.

5개 지표 중 우리 팀이 즉시 측정 가능한 것과 합의가 필요한 것을 분류하는 것이 자가 진단의 첫 단계입니다. Postmortem 머지 시간·OPS Diary 적재 누락은 자동 측정 가능하며, 런북 최신성·AI 답변 인용율·분기 회고 완성도는 측정 메커니즘의 사전 합의가 필요합니다 [S22]. 5개 지표가 모두 임계값을 넘으면 도입의 성공을 외부 감사 수준에서 입증 가능하며, 이는 ISMS-P 같은 규제 환경에서 도입 정착의 근거 자료가 됩니다.

임원 보고 KPI 와의 연결 사다리

위 5개 운영 KPI 는 의사결정권자께서 사장님 보고에 그대로 옮기실 수 있도록 다음과 같이 임원 KPI 와 연결됩니다. 본 사다리는 분기 보고 1쪽에 자동 집계되도록 설계되었으며, 의사결정권자께서는 본 5단 사다리를 그대로 분기 KPI 표에 옮기실 수 있습니다.

운영 KPI (실무)	임원 보고 KPI (의사결정)	사장님 보고 한 줄
Postmortem 머지 ≤ 24h, 비율 ≥ 80%	인시던트 사후 학습 사이클 정착도	같은 장애가 다음 분기에 반복되지 않는다

운영 KPI (실무)	임원 보고 KPI (의사결정)	사장님 보고 한 줄
런북 최신성 평균 ≤ 6개월	운영 자산 신뢰도 · 외부 감사 대응력	ISMS-P · 금감원 변경 감사를 별도 도구 없이 통과한다
AI 답변 인용율 ≥ 95%	AI 도입 안전성 · 환각 0건 약속	AI 답변이 항상 사내 원본 파일을 근거로 한다
OPS Diary 적재 누락 0	운영 지식 누적률 · 후임자 학습 곡선 단축	핵심 운영자 이직에도 운영 능력이 끊기지 않는다
분기 회고 5섹션 완성도 100%	분기 운영 의사결정의 데이터 기반화	의사결정이 머릿속이 아니라 디렉터리에서 시작한다

9.1.2 도입 비용·기대 효과 가상 산정 (운영팀 100명·매뉴얼 500건 기준)

본 산정은 가상 조직(운영팀 100명·관리 매뉴얼 500건·분기 인시던트 12건) 기준이며, 실제 비용은 조직 환경·인건비 수준·기존 자산 보유 정도에 따라 ±50% 변동합니다. 의사결정권자께서 품의서·기안문의 "기대 효과(정량)" 칸을 채우실 때 본 표를 1차 가이드로 사용하시고, 본인 조직 환경의 인건비 단가와 기존 SaaS 라이선스 비용으로 X·Y 자리를 채우시기 바랍니다.

도입 비용 (1년차)

항목	수량·단가	합계 (참고)
시니어 SRE 1명 6주 PoC	0.5 FTE × 1.5 개월	인건비 단가에 비례
주니어 운영자 3명 교육	0.1 FTE × 3명 × 2주	인건비 단가에 비례
GPU 워크스테이션 (24GB 1장)	1회성 구매 또는 클라우드 임대	약 300~1,200 만원
GPU 운영 단계 (80GB 1장)	1회성 (도입 6개월 후)	약 3,000~7,000 만원
Frontmatter 표준 합의 ·CODEOWNERS 정의	위 PoC 인건비에 포함	(별도 비용 없음)
(선택) OPENMARU APM 6주 공동 설계 컨설팅	1회성	별도 견적

기대 효과 (정량, 1년차 누적)

항목	산정 근거	연 절감액 (가상)
Postmortem 작성 시간 절감	분기 12건 × (90분 → 12분 검토) × 4분기 = 약 60 인시간/년 (sev1·sev2 합산 기준 별도 환산)	인건비 단가 × 인시간
핵심 운영자 이직 시 학습 곡선 단축	후임자 1~6개월 → 1~2주 (10~25배 단축)	1회당 수천 만원 회피

항목	산정 근거	연 절감액 (가상)
기존 SaaS 위키 (Confluence DC 100 user) 회피	라이선스 + 호스팅 + 유지보수	연 수천 만원 회피
외부 변경 관리 도구 (ServiceNow Change 등) 회피	Git commit log 가 ISMS-P 변경 감사 1차 근거	연 수백~수천 만원 회피
인시던트 MTTR 단축으로 인한 매출 손실 회피	결제 API 1분 다운당 매출 손실액 × 단축 분수	조직별 상이 (자체 산정 필요)

손익분기 추정

도입 비용 회수까지 약 **6~18개월** 로 추정됩니다 (사전조사 FAQ Q11). SaaS 라이선스 회피 단독으로도 12개월 이내 손익분기가 가능한 시나리오가 다수입니다. 본인 조직의 정확한 손익분기 산정은 (가) 본인 조직 평균 SRE 인건비 단가, (나) 기존 SaaS 위키·변경 관리 도구의 연 라이선스 금액, (다) 분기 평균 인시던트 건수와 sev1·sev2 비율 세 가지를 입력으로 받아 도출하실 수 있습니다.

감사·국감 인용 가능 근거 (1차 자동 산출)

도입 후 다음 5개 산출물이 별도 보고서 작성 없이 자동 생성되며, 감사·국감 답변에 즉시 인용 가능합니다.

- Postmortem 머지 시간 분포 (Git commit timestamp 자동 집계)
- ISMS-P 변경 감사 7항목 충족 증거 (PR 메타데이터)
- 외부 호출 0건 입증 (ndjson 한 줄 로그, 6장 6.1.1)
- 분기 회고 보고서 5섹션 완성도 (Work_Reports 디렉터리)
- AI 답변 인용율 (RAG 응답 본문의 파일명:라인 인용 비율)

9.1.3 함정 신호 5개 — AI 무승인 머지 · 디렉터리 표류 · frontmatter 미준수 등

도입 실패의 함정 신호 5개는 다음과 같이 박힙니다. 첫째, **AI 가 작성한 런북이 사람 승인 없이 머지되는 사례 발견** — frontmatter 의 `author: ai` 인 PR 이 reviewer 승인 없이 머지된 1건이라도 발견되면 즉시 회복 절차에 진입해야 합니다 [S61]. 둘째, **디렉터리 표류** — 6개 영역 디렉터리 외에 ad-hoc 폴더가 신설되거나, 일자별 파일이 컨벤션을 벗어나 자유 형식으로 적재되기 시작하는 상태입니다.

셋째, **frontmatter 미준수 비율 ≥ 20%** — 신규 머지 PR 중 8 필드 frontmatter 가 누락된 비율이 20% 를 넘으면 표준이 사실상 무력화된 상태이며, JSON Schema 검증 게이트가 우회되고 있다는 신호입니다 [S60]. 넷째, **사내 메신저 봇 ndjson 적재 누락 발생** — 인시던트 1건의 ndjson 이 봇 오류로 누락되어 raw 덤프가 없는 인시던트가 발생한 상태입니다. 다섯째, **reviewer 사이클 표류** — AI 정제 PR 의 reviewer 검토가 24시간 안에 시작되지 않는 PR 이 분기당 1건이라도 발생한 상태입니다.

함정 신호 5개의 자동 감지 메커니즘을 사전 구축하는 것이 도입 정착의 가장 큰 방어선입니다. PR 봇 1개가 5개 함정 신호 모두를 자동 감지할 수 있으며, 감지 시 운영팀장과 시니어 엔지니어 2인에게 즉시 알림이 발송되도록 라우팅하는 것을 권장합니다 [S19]. 함정 신호 1건의 조기 발견이 도입 실패의 조기 회복으로 이어진다는 점에서, 자동 감지 PR 봇의 구축 자체가 6~12개월 단계의 안전판이 됩니다. 회복 절차는 신호별로 1쪽 분량의 표준 절차로 박혀 reviewer 사이클 정착의 1단계 보강 자료가 됩니다.

9.1.4 OPENMARU APM 의 도입 가속 — 6주 PoC 공동 설계 hook

OpsKnow Repo 의 첫 분기 토대를 6주로 압축한 PoC 가 7장의 6주 도입 로드맵입니다. 6주 PoC(Proof of Concept, 개념 증명 — 본격 도입 전 핵심 기능과 운영 모델을 6주 안팎으로 검증하는 소규모 시범 도입) 의 입력 — 첫 이식 시스템·책임자·검증 기준의 1쪽 RFP 양식·디렉터리 트리 PR 의 reviewer 명단 — 을 외부 파트너와 함께 설계하면 합의 비용이 가장 크게 줄어듭니다.

OpsKnow Repo 는 **OPENMARU APM 에 포함된 운영 지식 레포지토리 제품** 입니다 (1장 정의 박스 참조). OPENMARU APM 도입 고객사에게는 OpsKnow Repo 가 함께 제공되며, OPENMARU APM 의 모니터링·알람·인시던트 데이터가 OpsKnow Repo 의 디렉터리 트리로 자동 인입되는 통합 효과를 얻을 수 있습니다. 신규 도입의 경우 OPENMARU APM 도입과 OpsKnow Repo 6주 PoC 를 함께 진행하는 형식의 공동 설계 가속을 받으실 수 있습니다 [S19].

운영팀이 단독으로 6주 PoC 를 진행하는 것과 외부 파트너와 공동 설계하는 것의 차이는 합의 시간 단축에 있습니다. 1쪽 RFP 양식·reviewer 명단·검증 기준의 세 가지 합의가 외부 파트너의 표준 양식으로 미리 제공되면, 운영팀은 자기 환경에 맞춰 빈칸을 채우는 작업으로 합의를 마칠 수 있습니다. 6주 PoC 공동 설계의 산출물 목록은 부록 E 의 도입 6주 체크리스트와 1:1 정합하며, 이 체크리스트가 곧 본 백서 마지막의 "다음 액션 1쪽 RFP 양식" 박스의 출발선입니다.

9.2 다음 액션 — 1쪽 RFP 양식 (의사결정권자용)

본 백서를 검토하신 의사결정권자께서 다음 분기 도입 검토를 시작하실 때, 다음 1쪽 양식을 운영팀에 그대로 전달하시면 합의 시간이 크게 줄어듭니다. 본 양식은 운영팀이 단독 도입을 진행할 때나 OPENMARU APM 과 함께 공동 설계를 검토하실 때 모두 동일하게 사용 가능합니다.

항목	결정 사항	기재 예시 / 권장
1. 첫 이식 시스템	트래픽이 가장 높고 운영팀이 가장 익숙한 시스템 1개	결제 API · 사내 SSO · 로그인 서비스 등 1개 도메인
2. PoC 책임자 2명	운영팀장 1명 + 시니어 SRE 1명	1주차부터 6주차까지의 단일 책임 라인
3. 검증 기준 5개	9.1.1 의 성공 신호 5개 그대로 차용	Postmortem 머지 ≤ 24h · 런북 최신성 · AI 인용율 · OPS Diary 적재 누락 0 · 분기 회고 5섹션
4. GPU 자원 확보	24GB GPU 1장 (PoC) → 80GB GPU 1장 (운영)	1주차에 자원 요청 결재 라인 상정 (4주 lead time)
5. 외부 파트너 참여 여부	단독 진행 vs OPENMARU APM 공동 설계	OPENMARU APM 기 도입 고객사는 공동 설계 권장

예산 결재용 입력 (9.1.2 산정과 1:1 정합)

- 인건비: 시니어 SRE 0.5 FTE × 1.5개월 + 주니어 0.1 FTE × 3명 × 2주
- GPU: 1년차 300~7,000 만원 (PoC → 운영 단계 전환 시 추가)
- SaaS 회피: Confluence DC · ServiceNow Change 라이선스 (조직 환경에 따라 연 수천 만원)

- 손익분기: 6~18개월

위 5개 항목과 예산 입력 4개가 1쪽 기안으로 정리되면, 분기 회의 1회의 안건으로 결정이 가능합니다. OPENMARU APM 공동 설계 양식이 필요하시면 공식 채널 (https://opennaru.com) 로 별도 문의 가능합니다.

Appendix A. References

본 백서는 본문 9개 장에 걸쳐 51개의 1차·2차 출처를 [S##] 형식으로 인용합니다. 아래 목록은 본문 인용 순서와 무관하게 출처 번호 순으로 정렬되며, 본문에서 한 번도 인용되지 않은 사전조사 항목은 제외했습니다. 모든 URL 은 2026년 5월 기준 접근 가능을 확인하였으며, 게이트가 걸린 일부 문서(예: r/devops megathread 류)는 서브레딧 또는 검색 진입 URL 로 갈음합니다.

[1] GitBook. *GitBook Docs — Git Sync*. https://docs.gitbook.com/getting-started/git-sync

[3] MkDocs / Squidfunk. *MkDocs* 및 *Material for MkDocs*. https://www.mkdocs.org/ ; https://squidfunk.github.io/mkdocs-material/

[6] Hugo / Google. *Hugo Documentation* 및 *Docsy Theme*. https://gohugo.io/documentation/ ; https://www.docsy.dev/

[7] Obsidian. *Obsidian Help* 및 *About — Your data*. https://help.obsidian.md/ ; https://obsidian.md/about

[8] Brian Petro / Logan Yang. *Smart Connections for Obsidian* 및 *Copilot for Obsidian*. https://github.com/brianpetro/obsidian-smart-connections ; https://github.com/logancyang/obsidian-copilot

[10] Logseq. *Logseq Documentation* 및 GitHub 저장소. https://docs.logseq.com/ ; https://github.com/logseq/logseq

[11] Outline. *Outline — Team knowledge base* 및 GitHub 저장소. https://www.getoutline.com/ ; https://github.com/outline/outline

[12] BookStack. *BookStack Documentation*. https://www.bookstackapp.com/docs/

[13] Rundeck (PagerDuty). *Rundeck Documentation*. https://docs.rundeck.com/docs/

[17] Atlassian. *Confluence* 및 *Confluence Storage Format*. https://www.atlassian.com/software/confluence ; https://confluence.atlassian.com/doc/confluence-storage-format-790796544.html

[18] Atlassian. *Atlassian Intelligence / Rovo*. https://www.atlassian.com/software/rovo

[19] GitLab. *GitLab Handbook* 및 Handbook content 리포지토리. https://handbook.gitlab.com/ ; https://gitlab.com/gitlab-com/content-sites/handbook

[20] GitLab. *GitLab Duo*. https://about.gitlab.com/gitlab-duo/

[21] Google SRE. *Site Reliability Engineering — Book* 및 *Workbook*. https://sre.google/sre-book/table-of-contents/ ; https://sre.google/workbook/table-of-contents/

[22] AWS. *AWS Well-Architected Framework — Operational Excellence Pillar*. https://docs.aws.amazon.com/wellarchitected/latest/operational-excellence-pillar/welcome.html

[23] PagerDuty. *Incident Management* 및 *Developer API*. https://www.pagerduty.com/platform/incident-management/ ; https://developer.pagerduty.com/api-reference/

[24] Incident.io. *AI 제품 페이지 및 Changelog*. https://incident.io/ai ; https://incident.io/changelog

[25] FireHydrant. *AI-generated Retrospectives* 및 *Retrospectives Feature*. https://firehydrant.com/blog/ai-generated-retrospectives/ ; https://firehydrant.com/features/retrospectives/

[26] Rootly. *Rootly AI* 및 Blog. https://rootly.com/ai ; https://rootly.com/blog

[27] Jeli / PagerDuty. *Jeli 제품 페이지* 및 *PagerDuty Acquires Jeli (Blog)*. https://www.jeli.io/ ; https://www.pagerduty.com/blog/pagerduty-acquires-jeli/

[28] Blameless. *Incident Management* 및 *Post-Incident Review*. https://www.blameless.com/incident-management ; https://www.blameless.com/post-incident-review

[29] Datadog. *Incident Management* 및 *Bits AI*. https://docs.datadoghq.com/service_management/incident_management/ ; https://www.datadoghq.com/product/bits-ai/

[30] Glean. *Glean Assistant* 및 *Work AI Platform*. https://www.glean.com/product/assistant ; https://www.glean.com/product/work-ai-platform

[32] Notion. *Notion AI* 및 *Notion AI FAQs*. https://www.notion.so/product/ai ; https://www.notion.so/help/notion-ai-faqs

[33] Mintplex Labs. *AnythingLLM* 및 GitHub 저장소. https://anythingllm.com/ ; https://github.com/Mintplex-Labs/anything-llm

[34] Zylon. *PrivateGPT* 및 GitHub 저장소. https://privategpt.dev/ ; https://github.com/zylon-ai/private-gpt

[35] LlamaIndex. *LlamaIndex* 및 GitHub 저장소. https://www.llamaindex.ai/ ; https://github.com/run-llama/llama_index

[36] LangChain. *LangChain* 및 *RAG Tutorial*. https://www.langchain.com/ ; https://python.langchain.com/docs/tutorials/rag/

[37] QuivrHQ. *Quivr* 및 GitHub 저장소. https://www.quivr.com/ ; https://github.com/QuivrHQ/quivr

[38] Khoj. *Khoj* 및 GitHub 저장소. https://khoj.dev/ ; https://github.com/khoj-ai/khoj

[39] Open WebUI. *Open WebUI* 및 *Open WebUI Docs*, GitHub 저장소. https://openwebui.com/ ; https://docs.openwebui.com/ ; https://github.com/open-webui/open-webui

[41] Onyx (구 Danswer). *Onyx 제품 페이지* 및 GitHub 저장소. https://www.onyx.app/ ; https://github.com/onyx-dot-app/onyx

[42] Ollama. *Ollama GitHub 저장소*. https://github.com/ollama/ollama

[43] vLLM Project. *vLLM GitHub 저장소* (PagedAttention; Kwon et al., SOSP 2023). https://github.com/vllm-project/vllm

[44] LM Studio. *LM Studio 0.3 Release Notes — Chat with Documents (RAG)*. https://lmstudio.ai/blog/lmstudio-v0.3.0

[45] ggerganov. *llama.cpp GitHub 저장소*. https://github.com/ggerganov/llama.cpp

[47] Continue.dev. *Continue Documentation*. https://docs.continue.dev/

[48] Nomic AI. *GPT4All LocalDocs Documentation*. https://docs.gpt4all.io/gpt4all_desktop/localdocs.html

[49] Reddit. *r/devops — "How do you document tribal knowledge?" 류 정기 스레드*. https://www.reddit.com/r/devops/

[50] Reddit. *r/sysadmin — "What do you use for documentation?" megathread*. https://www.reddit.com/r/sysadmin/

[51] Charity Majors. *charity.wtf* 및 *Honeycomb Blog*. https://charity.wtf/ ; https://www.honeycomb.io/blog

[52] Google SRE. *SRE Book Ch.6 "Monitoring Distributed Systems" / Ch.10 "Practical Alerting"*. https://sre.google/sre-book/

[53] GitLab. *GitLab Runbooks (공개 리포지토리)*. https://gitlab.com/gitlab-com/runbooks

[54] Hacker News. *"Markdown vs Confluence for runbooks" 류 다수 스레드*. https://news.ycombinator.com/

[55] Reddit. *r/LocalLLaMA — RAG 튜닝 megathread*. https://www.reddit.com/r/LocalLLaMA/

[56] Google SRE. *SRE Workbook Ch.10 — Postmortem Culture*. https://sre.google/workbook/postmortem-culture/

[57] PagerDuty. *Postmortem Guide* 및 *SRE Weekly*. https://postmortems.pagerduty.com/ ; https://sreweekly.com/

[58] Reddit. *r/LocalLLaMA — "Best private docs RAG stack" megathread*. https://www.reddit.com/r/LocalLLaMA/

[60] GitLab / Basecamp / Stripe. *Handbook-first 문화 — GitLab Handbook* 외.
https://handbook.gitlab.com/

[61] Simon Willison / Charity Majors. *Simon Willison's Weblog* 및 *charity.wtf*.
https://simonwillison.net/ ; https://charity.wtf/

[62] Daniele Procida. *Diátaxis — 문서 프레임워크*. https://diataxis.fr/

Appendix A-K. 국내 법령·정부 가이드 (한국어 1차 출처)

본문 6장 6.4 절(ISMS-P · 금감원 · 의료 규제) 에서 인용한 국내 법령·고시·정부 가이드의 1차 출처를 모았습니다. 본 블록의 URL 은 2026년 5월 기준 접근 가능을 확인했으나, 법령은 개정 가능성이 있으므로 본인 조직 도입 시점에는 시행 일자와 조문 본문을 1차 출처에서 재확인하시기 바랍니다. 감사·국감 답변 시에는 본 블록의 [SK##] 번호를 그대로 인용 근거로 사용하실 수 있습니다.

[SK01] 한국인터넷진흥원(KISA). *정보보호 및 개인정보보호 관리체계(ISMS-P) 인증제도·인증기준 안내서*.
https://isms.kisa.or.kr/

[SK02] 법제처 국가법령정보센터. *전자금융감독규정* (금융위원회 고시) 제13조 (전산자료의 보호대책).
https://www.law.go.kr/

[SK03] 법제처 국가법령정보센터. *전자금융감독규정* (금융위원회 고시) 제14조의2 (클라우드 컴퓨팅서비스 이용·제공). https://www.law.go.kr/

[SK04] 금융위원회·금융감독원. *금융분야 AI 가이드라인* (2021년 7월 제정, 이후 개정).
https://www.fsc.go.kr/ ; https://www.fss.or.kr/

[SK05] 금융보안원. *금융권 클라우드 컴퓨팅서비스 이용 가이드*. https://www.fsec.or.kr/

[SK06] 법제처 국가법령정보센터. *개인정보보호법* 제23조 (민감정보의 처리 제한). https://www.law.go.kr/

[SK07] 법제처 국가법령정보센터. *의료법* 제21조의2 (진료기록의 송부) 및 동법 시행규칙.
https://www.law.go.kr/

[SK08] 보건복지부. *보건의료데이터 활용 가이드라인* (2020년 9월 제정, 이후 개정).
https://www.mohw.go.kr/

[SK09] 법제처 국가법령정보센터. *개인정보보호법* 제28조의2~9 (가명정보의 처리), 2023년 시행.
https://www.law.go.kr/

Appendix B. Glossary

본문 흐름에서 처음 등장하는 약어와 한글-영문 병기 용어를 모았습니다. 같은 항목이 여러 장에서 다른 설명으로 풀어 쓰였을 때는 가장 포괄적인 설명을 채택했고, 본문에 등장하지 않는 도메인 일반 용어는 제외했습니다. 정렬은 영문 우선 ASCII → 한글 가나다 순입니다.

용어	정의
ADR	Architecture Decision Record. 아키텍처 결정 기록 — 구조적 결정의 동기·대안·결과를 한 장의 MD 로 남기는 표준 형식.
AI Agent	사람의 지시 없이도 사전 정의된 주기·트리거·도구 체인을 따라 점검·요약·초안 작성을 수행하는 LLM 기반 자동 실행 주체.
BGE-M3	BAAI 가 공개한 다국어 임베딩 모델. 한국어 비중이 높은 운영 문서에서 1차 권장 임베딩으로 사용된다.
Chroma	가벼운 임베디드 벡터 DB. 단일 호스트·중소 규모 RAG 환경의 표준 선택지 중 하나.
CIDR	Classless Inter-Domain Routing. IP 주소 범위 표기법으로 운영 문서 내 네트워크 범위·방화벽 규칙 기술에 사용된다.
CODEOWNERS	Git 호스팅에서 디렉터리·파일 패턴별 코드 소유자를 지정해 PR reviewer 를 자동 할당하는 표준 파일.
Continue.dev	VSCode·JetBrains 용 IDE 코파일럿. 로컬·원격 LLM 을 IDE 안에서 코드/문서 컨텍스트와 함께 사용 가능.
Diátaxis	Tutorial / How-to / Reference / Explanation 네 영역으로 문서를 분리하는 프레임워크. OpsKnow 의 `doc_type` 스키마 원천.
Docs-as-Code	문서를 코드처럼 Git·PR·CI 로 다루는 운영 철학. MD + 정적 사이트 생성기 + PR 리뷰 게이트가 표준 조합.
DR 훈련	Disaster Recovery 훈련. 재해 복구 절차를 주기적으로 모의 수행하여 복구 능력을 검증하는 운영 활동.
Field Manual	시스템·소프트웨어별 매뉴얼·SOP 의 현장 책장. OpsKnow Repo 7개 영역 중 하나.
frontmatter	MD 파일 상단의 YAML 메타데이터 블록. `owner` · `reviewer` · `service` · `severity` · `freshness_until` · `confidence` 등 운영 ontology 의 1차 표현 자리.
GGUF	GPT-Generated Unified Format. llama.cpp 계열이 사용하는 양자화 모델 포맷으로 Ollama 가 표준으로 다룬다.
Glean	사내 SaaS 전체를 통합 검색·Q&A 하는 엔터프라이즈 검색 SaaS. SaaS 전제·데이터 lock-in 카테고리의 대표 예.

용어	정의
Incident	인시던트. 정상 운영을 벗어난 단일 사건으로, 알람·채팅·Postmortem 까지의 라이프사이클이 OpsKnow 의 한 영역을 이룬다.
LanceDB	임베디드 벡터 DB. 디스크 친화 컬럼나 포맷으로 단일 호스트·중소 규모 RAG 환경에 적합.
LLM	Large Language Model, 대규모 언어 모델. 수십억~수천억 파라미터로 학습된 자연어 생성 모델.
MTTR	Mean Time To Recovery, 평균 복구 시간. 인시던트 발생부터 정상 복구까지의 평균 경과 시간.
ndjson	Newline Delimited JSON. 한 줄에 한 JSON 객체를 담는 스트리밍 친화 포맷으로, 사내 메신저 봇의 인시던트 채팅 적재에 표준.
NER	Named Entity Recognition, 개체명 인식. 인명·기관명·지명·IP·이메일 등 고유 명사를 자동 식별하는 자연어 처리 기법.
nomic-embed-text	Nomic AI 가 공개한 영문 우선 임베딩 모델. 짧은 영문 문서 위주 환경에서 BGE-M3 의 대안.
Nx	monorepo 의 워크스페이스 단위 의존성 그래프와 빌드 캐시를 제공하는 도구.
Ollama	`ollama run` 한 줄로 GGUF 모델을 OpenAI 호환 API 로 띄우는 최소 설치 로컬 LLM 런타임.
Ontology	운영 도메인 어휘·관계·메타데이터의 표준 체계. OpsKnow 의 frontmatter 스키마가 곧 운영 ontology 의 실체.
Onyx	구 Danswer. Slack·GitHub·Confluence·Drive 등 50+ 커넥터를 갖는 OSS 엔터프라이즈 검색·Q&A.
Open WebUI	Ollama·vLLM 백엔드 위에 ChatGPT-like UI 와 Knowledge/Workspace 를 얹은 로컬 RAG 워크벤치.
OPS Diary	AI 와의 일일 질의응답을 시간축에 박는 운영 일지. OpsKnow Repo 7개 영역 중 하나.
outbound	내부에서 외부 인터넷으로 나가는 네트워크 방향. 반대 방향은 inbound. 외부 호출 0건 검증의 1차 측정 지점.
PagedAttention	vLLM 이 채택한 GPU 메모리 관리 기법. 동일 GPU 에서 더 많은 동시 호출을 처리 가능하게 한다.

용어	정의
Postmortem	인시던트 사후 분석 보고서. 원인·타임라인·액션아이템·태그를 정형 템플릿으로 보관하며, OpsKnow 에서는 단일 MD 파일로 확정된다.
PR	Pull Request. 변경분을 본 브랜치에 머지하기 전 사람 reviewer 의 승인을 받기 위한 표준 코드/문서 협업 단위.
Qdrant	분리형 벡터 DB. 청크 수가 늘어나거나 동시 사용자가 많아지는 환경에서 LanceDB·Chroma 의 다음 단계.
RAG	Retrieval-Augmented Generation, 검색 보강 생성. 외부 지식 코퍼스에서 관련 청크를 검색해 LLM 응답의 근거로 주입하는 패턴.
RBAC	Role-Based Access Control. 역할 단위 권한 제어. CODEOWNERS·Git 호스팅 권한 모델과 결합해 운영 지식의 변경 권한을 분리한다.
Runbook	런북. 특정 알람·상황·작업에 대해 사람이 따라야 할 단계 순서를 정형화한 운영 문서. OpsKnow 에서는 Field Manual 의 한 doc_type.
SLI	Service Level Indicator, 서비스 수준 지표. SLO 의 측정 단위가 되는 가용성·지연·오류율 등의 정량 지표.
SLO	Service Level Objective, 서비스 수준 목표. 가용성·지연·오류율의 수치 임계를 시간 단위로 정의한 운영 약속.
SOP	Standard Operating Procedure, 표준 작업 절차. 반복 가능한 운영 작업의 단계와 검증 항목을 정형화한 문서.
SRE	Site Reliability Engineering. 소프트웨어 엔지니어링 원칙으로 운영을 다루는 분야. SLO·오류예산 ·Postmortem 의 표준 어휘를 OpsKnow 가 흡수한다.
SSoT	Single Source of Truth, 단일 진실 공급원. 한 항목에 대해 권위 있는 단 하나의 출처를 두는 원칙.
Turborepo	monorepo 의 변경 감지·증분 빌드를 제공하는 도구.
vLLM	PagedAttention 으로 GPU 효율을 극대화하는 OpenAI 호환 추론 서버. 사용자 수가 늘어난 단계에서 Ollama 의 다음 선택지.
WIP	Work In Progress, 진행 중 작업. 사내 메신저 채팅의 비공식 메모를 가리킬 때 흔히 쓰이며, OpsKnow 의 영구 MD 와 대비된다.

용어	정의
YAML	YAML Ain't Markup Language. MD frontmatter 의 표준 포맷으로 운영 ontology 의 1차 표현 매체.
글롭	Glob. 파일 경로의 와일드카드 매칭 문법. `*.md` , `Field_Manual/**` 같은 패턴.
유효성/최신성	freshness. 문서가 현재도 유효한지의 시간 축 메타. OpsKnow 는 `freshness_until` frontmatter 로 유효 만료일을 표기한다.
오류예산	Error Budget. SLO 가 허용하는 실패의 정량 한도. 한도 소진 속도가 빠를수록 신규 변경을 줄이는 운영 의사결정의 1차 신호.
온콜	On-call. 운영 책임자가 정해진 순번에 따라 알람·인시던트 1차 대응을 맡는 운영 당번 제도.
환각	Hallucination. LLM 이 근거 없는 사실을 그럴듯하게 생성하는 현상. OpsKnow 의 RAG corpus 가 사람 승인 MD 로 좁혀져 표면적이 축소된다.

OpsKnow Repo — AI 시대 Infra 운영을 위한 도서관

CONTACT

WEB
msap.ai
www.msap.ai/

EMAIL
hello@msap.ai

TEL
02-6953-5427
0269535427

YOUTUBE
@msaptv
www.youtube.com/@msaptv

LINKEDIN
linkedin.com/showcas...
www.linkedin.com/showcase/msap-ai/

FACEBOOK
facebook.com/opennaru
www.facebook.com/opennaru

SCAN

MSAP.ai

msap.ai

© 2026 SMART CF, MSAP.ai. All rights reserved.