CLOUD NATIVE FORUM

Notion 대체 오픈소스 협업 도구 AppFlowy·AFFiNE·Outline 비교 백서

> 협업 도구는 더 이상 단순한 문서 편집기가 아닙니다. 팀의 회의록, 제품 기획, 고객 데이터, 계약 초안, 인사 평가까지 조직의 핵심 지식이 한곳에 쌓이는 그릇이 되었습니다. Notion 같은 SaaS(클라우드형 구독 서비스) 도구가

WHITEPAPER · TECH BRIEF

2026-06-14 · 작성자: CNCF Korea Team · cncf.co.kr

목차

1장: 왜 지금 셀프호스팅 협업 도구를 검토해야 하는가

협업 도구는 더 이상 단순한 문서 편집기가 아닙니다. 팀의 회의록, 제품 기획, 고객 데이터, 계약 초안, 인사 평가까지 조직의 핵심 지식이 한곳에 쌓이는 그릇이 되었습니다. Notion 같은 SaaS(클라우드형 구독 서비스) 도구가 빠르게 보급되면서 문서를 만들고 공유하는 속도는 분명히 빨라졌습니다. 그러나 동시에 구독 비용은 매년 늘고, 민감한 자료가 외부 서버에 쌓이는 양도 함께 늘었습니다.

여러분이 IT 의사결정을 맡고 있다면 최근 1~2년 사이 이런 질문을 한 번쯤 받아 보셨을 것입니다. "우리 회사 모든 문서가 해외 서버에 있는데, 이대로 괜찮은가." "사용자가 200명을 넘어가니 연간 구독료가 적지 않은데, 이 비용을 계속 부담해야 하는가." "특정 산업 규제가 강화되면 지금 쓰는 도구로 컴플라이언스를 맞출 수 있는가." 이 세 질문은 데이터 주권, 비용 통제, 컴플라이언스라는 서로 다른 압력으로 동시에 다가오고 있습니다.

이 1장은 그 압력의 실체를 정리하고, 왜 지금이 셀프호스팅 협업 도구를 검토할 시기인지 답을 드립니다. 셀프호스팅이란 외부 SaaS에 가입해서 쓰는 대신, 회사가 직접 관리하는 서버나 사내 클라우드에 협업 도구를 설치해서 운영하는 방식을 말합니다. 과거에는 설치와 운영이 까다로워 큰 조직만 시도할 수 있었지만, 최근 몇 년 사이 Docker(컨테이너 기반 배포 도구)와 클라우드 네이티브 기술이 보편화되면서 중소 규모 팀도 충분히 다룰 수 있게 되었습니다.

또한 AppFlowy, AFFiNE, Outline 세 가지 오픈소스 협업 도구가 각각 GitHub에서 수만 명의 개발자와 사용자에게 지지를 받으며 빠르게 성장하고 있습니다 [S1] [S4] [S7]. 9장에 걸친 본 비교 백서는 이 세 도구를 11개 평가 기준으로 살펴보고, 여러분이 어떤 상황에서 어느 도구를 선택해야 하는지 의사결정 근거를 제공합니다. 1장은 그 출발점입니다.

1.1 SaaS 협업 도구의 구조적 한계

SaaS 협업 도구의 성공 비결은 가입 즉시 사용할 수 있다는 점에 있었습니다. 그러나 같은 편의성이 시간이 지나면 누적 비용과 외부 의존이라는 두 가지 구조적 한계로 돌아옵니다. 이 절에서는 두 가지 한계가 어떻게 조직의 발목을 잡는지 정리합니다.

1.1.1 누적 구독 비용과 팀 규모에 따른 부담 증가

Notion 비즈니스 요금제는 연간 약정 기준 사용자당 월 18~20달러 수준이며, 부가가치세 10%가 별도로 부과됩니다(2026년 6월 기준). 환율을 1,400원으로 단순 환산하면 1인당 월 2.8만 원 안팎, 연 33~36만 원입니다. 50명 규모 팀이라면 연간 1,700만 원 안팎, 200명 규모라면 6,700만 원 안팎의 고정 비용이 발생합니다. 처음 도입할 때는 한 사람당 월 3만 원 안쪽이라 부담이 작아 보이지만, 인원이 늘고 도구가 늘수록 누적 비용은 빠르게 커집니다. 협업 도구 한 가지만 쓰는 조직은 거의 없기 때문입니다.

같은 팀이 보통 이슈 추적(Jira·Linear), 메신저(Slack·네이버웍스), 화상 회의(Zoom·Google Meet), 디자인(Figma), 코드 저장소(GitHub·GitLab)까지 별도 SaaS를 함께 씁니다. 도구마다 사용자당 월 10~30달러가 붙으면 1인당 연간 SaaS 지출은 100만 원을 가볍게 넘기는 일이 흔합니다. 인원이 늘면 비용도 비례해서 늘기 때문에 인건비 다음으로 무시할 수 없는 고정 지출이 됩니다.

문제는 비용 자체보다 통제 가능성입니다. 구독료는 해외 공급사의 가격 정책에 따라 일방적으로 오를 수 있고, 원/달러 환율이 흔들리면 원화 부담은 그대로 늘어납니다. 2022~2024년의 환율 급등 국면에서 SaaS 예산 초

과거 사내 IT 예산의 단골 이슈였던 경험이 이를 잘 보여 줍니다. 사용량이 줄어도 좌석 단위로 계약했다면 즉시 비용을 줄이기 어렵습니다. 반면 셀프호스팅 도구는 라이선스 비용이 0이거나 매우 낮고, 늘어나는 비용은 서버 자원과 운영 인건비로 구성되므로 조직 안에서 예측·조정할 수 있습니다. AppFlowy가 "라이선스 비용 없음, 사용자당 과금 없음, 기능 차단 없음"을 공식 안내하는 까닭도 여기에 있습니다 [S3].

도구 하나를 두 해만 운영해도 셀프호스팅이 SaaS 구독 누적액을 따라잡는 손익분기점이 보이기 시작합니다. 따라서 비용 통제는 단순한 절약 문제가 아니라, 향후 3~5년의 IT 예산 구조를 설계하는 문제로 다루어야 합니다.

1.1.2 외부 서버 의존으로 인한 데이터 통제권 약화

SaaS 협업 도구를 쓰면 모든 문서, 첨부 파일, 댓글, 사용자 로그가 공급사 서버에 저장됩니다. 대부분의 공급사는 보안 인증을 갖추고 있지만, 어떤 데이터가 어디에 있는지, 어떤 인력이 접근하는지를 고객사가 직접 확인하기는 어렵습니다. 데이터 주권이 약해진다는 뜻입니다.

국내 금융권, 공공기관, 의료·제약 분야는 「개인정보 보호법」 제29조 안전조치 의무, 「전자금융감독규정」, 「공공기관 정보보안 기본지침」 같은 산업별·국가 규제로 인해 핵심 자료를 국외 서버에 둘 수 없는 경우가 많습니다. 공공기관의 경우 정보통신망 망분리(국가정보원·행정안전부 가이드), 금융권의 경우 금융보안원 망분리 가이드와 「전자금융감독규정」 제15조에 따른 망분리 의무가 적용됩니다. 망분리 환경에서는 외부 SaaS 접속 자체가 차단됩니다. 이런 조직이 외부 협업 도구를 쓰려면 별도 게이트웨이를 두거나 일부 업무만 도구에 올리는 절충안을 만들어야 하는데, 결국 도구의 본래 목적인 단일 협업 공간이 무너집니다.

또한 공급사가 갑작스러운 서비스 종료를 선언하거나, 약관을 변경해 데이터 이전 기능을 제한하면 조직은 큰 위험에 노출됩니다. 실제로 글로벌 SaaS 시장에서는 매년 여러 도구가 인수·합병되거나 단종되고 있습니다. 이때 데이터 이전 비용은 도구 선택 시점에는 보이지 않다가 전환 시점에 한꺼번에 발생합니다.

셀프호스팅 협업 도구는 데이터가 처음부터 자사 서버에 있으므로, 정책 변경이나 서비스 종료의 영향을 받지 않습니다. 백업, 접근 제어, 감사 로그를 조직이 직접 설계할 수 있어 ISMS-P·CSAP 같은 국내 정보보호 인증 대응도 한결 명확합니다. 데이터 통제권을 도구 선택의 우선 기준으로 두는 조직이 늘어나는 배경입니다.

1.2 오픈소스 셀프호스팅 대안의 부상

과거 셀프호스팅이 어렵게 느껴졌던 이유는 설치, 업그레이드, 백업, 모니터링까지 모두 손으로 처리해야 했기 때문입니다. 그러나 클라우드 네이티브 기술이 보편화되면서 운영 부담은 빠르게 줄었고, 같은 시기에 Notion 대체를 노리는 오픈소스 도구가 활발히 등장했습니다. 이 절에서는 그 두 흐름을 정리합니다.

1.2.1 클라우드 네이티브 운영 성숙과 Docker 기반 배포 보편화

Docker는 애플리케이션과 의존성을 컨테이너라는 단위로 묶어 동일한 환경에서 실행할 수 있게 해 주는 도구입니다. 협업 도구처럼 데이터베이스, 캐시, 검색 엔진, 웹 서버가 함께 돌아가야 하는 시스템도 docker-compose라는 파일 한 장이면 한꺼번에 띄울 수 있습니다. 운영자가 각 구성 요소의 버전과 설정을 일일이 맞추던 시대는 사실상 끝났습니다.

이 흐름은 본 백서가 다루는 세 도구에서도 분명히 드러납니다. 예를 들어 AppFlowy-Cloud는 운영, 개발, 지속 통합(CI) 환경별로 docker-compose 파일을 따로 제공하며, 컨테이너 기반 배포를 공식 방식으로 안내합니다 [S2]. 운영자는 명령 한두 줄이면 동일한 구성을 재현할 수 있습니다.

쿠버네티스(Kubernetes, 컨테이너 오케스트레이션 표준)까지 활용하면 무중단 업그레이드, 자동 확장, 장애 복구도 어렵지 않게 적용할 수 있습니다. 사내 인프라팀이 이미 쿠버네티스를 운영 중이라면, 협업 도구 한두 가지를 추가로 올리는 일은 새로운 기술 도입이 아니라 기존 운영 패턴의 확장입니다. 즉 셀프호스팅의 진입 장벽이 의사결정 기준선에서 거의 사라졌다는 뜻입니다.

또한 백업, 모니터링, 로그 수집을 위한 오픈소스 표준 도구가 함께 자리 잡으면서, 협업 도구 운영도 같은 표준에 얹어 관리할 수 있게 되었습니다. 운영 표준화가 이루어지면 도구 한 가지를 더 들여놓아도 운영 인건비가 비례해서 늘지 않습니다.

1.2.2 세 도구의 등장 배경과 활성도 개관

본 백서가 비교하는 세 도구는 모두 Notion이 보여 준 블록 기반 문서 모델을 자기 방식으로 다시 만들었습니다. 등장 시기와 기술 선택은 서로 다르지만, Notion 대체라는 목표는 공통입니다. 시장 활성도는 GitHub Star 수로 가늠해 볼 수 있는데, Star는 개발자와 운영자가 그 저장소를 즐겨찾기에 추가한 횟수에 가깝습니다.

AppFlowy는 2026년 5~6월 기준 메인 저장소에서 약 7만 2,000개의 GitHub Star를 얻으며 세 도구 중 가장 큰 관심을 모으고 있습니다 [S1]. 공식 소개에서 스스로 "선도적인 오픈소스 Notion 대안"이라 표현하고, "데이터 통제권을 잃지 않으면서 더 많은 일을 해낼 수 있는 AI 협업 워크스페이스"를 지향한다고 밝히고 있습니다 [S1].

AFFiNE는 같은 시기 약 7만 개의 Star를 얻어 AppFlowy와 비슷한 규모의 관심을 받고 있으며, 빠른 릴리스 주기로 유명합니다. 최신 카나리 빌드는 v2026.6.7-canary.1000으로 표기되며 거의 매일 새 빌드가 공개될 만큼 개발이 활발합니다 [S4]. 신기능을 빠르게 시험해 보려는 조직에 매력적인 신호입니다.

Outline은 약 3만 8,800개의 Star로 세 도구 중 가장 오래된 축에 속하지만, 위키 중심의 안정적인 사용성으로 꾸준한 지지를 얻고 있습니다 [S7]. 2026년 6월 기준 안정 릴리스는 v1.8.1이며, 라이선스도 2030년 6월 6일에 BSL(Business Source License)에서 Apache 2.0으로 자동 전환되도록 예고되어 있습니다 [S7] [S8]. 장기적 라이선스 안정성도 의사결정에 영향을 주는 신호입니다.

1.3 이 백서의 비교 방법과 독자 안내

세 도구가 모두 매력적인 선택지라면, 의사결정은 결국 "우리 조직의 우선순위에 어느 도구가 가장 잘 맞는가"라는 질문으로 좁혀집니다. 이 절은 그 답을 찾기 위한 비교 방법과 독자 활용 안내를 정리합니다.

1.3.1 11개 평가 기준 선정 근거와 비교 방법

본 비교는 11개 평가 기준을 사용합니다. 각 기준은 시장에서 자주 쓰이는 오픈소스 비교 항목과 의사결정자가 가장 자주 묻는 질문을 모아 선정했습니다. 구체적으로는 라이선스, 데이터 모델, 편집 경험, 협업 기능, 검색·AI 기능, 보안·권한, 배포·운영 난이도, 확장성, 통합·연동, 커뮤니티 활성도, 비용 구조의 열한 가지입니다.

기준을 11개로 늘린 까닭은 단순합니다. 기능 비교만으로는 의사결정이 어렵기 때문입니다. 라이선스와 비용 구조를 보지 않고 기능만 보면, 도입 6개월 뒤 라이선스 충돌이나 예상 외 비용 청구로 후회하는 일이 생깁니다. 운영 난이도를 보지 않으면 도입 직후 인프라팀에서 반발이 생깁니다. 따라서 기능·운영·라이선스·비용의 네 영역을 균형 있게 포함했습니다.

비교 방법은 두 단계로 진행합니다. 먼저 각 기준마다 세 도구의 사실 정보를 정리합니다. 출처가 공식 저장소, 공식 문서, 라이선스 파일인 항목만 사실로 다루며, 본문에는 [S##] 식별자로 출처를 표기합니다. 다음으로 같

은 기준 안에서 세 도구를 상대 비교해 강점과 약점을 정리합니다.

마지막으로 도구별 적합 시나리오를 제시합니다. 같은 도구라도 조직 규모, 산업 규제, 기존 인프라에 따라 평가가 달라지기 때문입니다. 따라서 본 백서의 결론은 "어느 도구가 최고인가"가 아니라 "어느 상황에서 어느 도구가 가장 잘 맞는가"입니다.

1.3.2 백서 활용 가이드와 9장 구성의 독자 안내

이 문서는 9장으로 구성됩니다. 1장은 지금 살펴보는 도입 장이며, 2장은 세 도구의 개요와 등장 배경, 3장은 라이선스와 비용 구조, 4장은 기능과 사용성, 5장은 배포·운영, 6장은 보안과 컴플라이언스, 7장은 확장성과 통합, 8장은 도구별 적합 시나리오, 9장은 결론과 의사결정 체크리스트입니다.

여러분이 시간이 부족하다면 두 가지 경로로 읽을 것을 권합니다. 첫째, 의사결정권자 경로입니다. 1장과 2장으로 전체 맥락을 잡고, 3장에서 비용·라이선스 위험을 확인한 뒤, 8장에서 자사 상황에 맞는 시나리오를 고른 다음, 9장 결론과 체크리스트로 마무리하면 약 25분 안에 핵심을 살펴볼 수 있습니다.

둘째, IT 담당자 경로입니다. 4장 기능 비교와 5장 배포·운영 난이도, 7장 확장성과 통합, 6장 보안 통제를 차례로 읽고, 1·2·3장으로 돌아와 배경과 비용 구조를 보완하는 순서가 효율적입니다. 도구 도입 가능성을 기술 관점에서 빠르게 확인할 수 있습니다.

본 백서가 다루는 것은 사실, 비교, 시나리오까지입니다. 도구 선택의 최종 책임은 여러분 조직에 있습니다. 이 문서는 그 결정을 더 빠르고 정확하게 내릴 수 있도록 정리된 출발점입니다. 2장부터는 세 도구를 한 도구씩 차분히 살펴보겠습니다.

2장: AppFlowy·AFFiNE·Outline 정체성과 개발 배경

세 도구는 외부에서 보면 모두 "Notion 대안" 이라는 하나의 묶음으로 소개됩니다. 그러나 출발점, 개발 주체, 기술 스택, 그리고 어떤 사용자를 머릿속에 두고 제품을 만들었는가 하는 지향점은 서로 크게 다릅니다. 이 차이를 모르고 단순히 별 개수나 화면 모양만 비교하면, 도입 후 1년 안에 "우리 조직에는 잘 맞지 않았다" 는 후회로 이어지기 쉽습니다. 의사결정권자께서는 본격 비교에 앞서 세 도구의 태생을 먼저 이해하시기를 권합니다.

AppFlowy 는 캐나다 토론토에서 시작된 프로젝트로, Notion 의 데이터 주권 문제를 직접 해결하겠다는 명확한 문제의식에서 출발했습니다 [S1]. AFFiNE 은 중국 항저우 토토로 인 (Toeverything) 팀이 만든 도구로, 문서와 화이트보드를 한 화면에서 다루는 새로운 작업 방식을 실험합니다 [S4]. Outline 은 미국 샌프란시스코 기반 팀이 2016년부터 다듬어 온 팀 위키 전용 도구로, 다른 두 도구보다 훨씬 좁은 영역에 집중하여 완성도를 끌어올렸습니다 [S7].

별 개수만 보면 AppFlowy 약 72,000 개, AFFiNE 약 70,000 개, Outline 약 38,800 개로 세 도구가 비슷한 규모의 관심을 받는 듯 보입니다. 그러나 별 개수는 호기심의 지표일 뿐, 실제 운영 적합성을 보장하지 않습니다. 같은 별 1만 개라도 어떤 사람이 누른 별인지, 어떤 목적으로 쓸 수 있는 도구인지 확인해야 합니다. 이번 장에서는 세 도구의 정체성을 라이선스 원문과 공식 저장소 기준으로 정리하고, 어떤 조직 유형에 어떤 도구가 자연스럽게 들어맞는지 판단 근거를 제공합니다.

이 장은 3장의 라이선스 상세 해설, 4장의 기능 비교로 이어지는 출발선입니다. 여기서 다루는 내용은 "왜 이 도구가 이런 구조로 만들어졌는가" 입니다. 구조의 이유를 알면, 이후 장에서 다룰 기술 제약과 비용 모형이 훨씬 자연스럽게 읽힙니다.

2.1 AppFlowy의 정체성과 개발 배경

AppFlowy 는 세 도구 가운데 가장 야심 찬 범위를 표방합니다. 문서, 위키, 프로젝트 관리, 데이터베이스, 그리고 AI 도우미까지 하나의 작업 공간에 담아 Notion 을 정면으로 대체하겠다는 목표를 공식 소개에서 분명히 밝힙니다. 제품 구성은 넓지만 개발 주체는 비교적 작은 팀이며, 라이선스와 배포 방식 모두 셀프호스팅 사용자에게 우호적입니다.

2.1.1 Notion 형 올인원 워크스페이스를 지향하는 AppFlowy

AppFlowy 공식 소개는 제품의 정체성을 한 문장으로 요약합니다. "Bring projects, wikis, and teams together with AI. AppFlowy is the AI collaborative workspace where you achieve more without losing control of your data. The leading open source Notion alternative." [S1] 이 문장은 두 가지 약속을 동시에 담고 있습니다. 첫째, 프로젝트와 위키와 팀 협업을 한 도구로 묶겠다는 기능 약속입니다. 둘째, 데이터 통제권을 사용자에게 남겨두겠다는 운영 약속입니다. 의사결정권자께서는 이 두 약속이 실제 코드와 라이선스에서 어떻게 구현되었는지 확인하셔야 합니다.

라이선스 측면에서 AppFlowy 는 가장 명확합니다. 저장소 LICENSE 파일은 다음과 같이 선언합니다. "AGPL-3.0 — All content that resides under the AppFlowy repository is licensed under the GNU Affero General Public License v3. No license fees, no per-user charges, no feature gates." [S3] AGPL-3.0 은 GPL 의 강한 카피레프트 (copyleft, 파생물에도 같은 라이선스를 적용하도록 의무화) 에 네트워크 사용 조항을 더한 라이선스입니다. 사내 도입자 입장에서 가장 중요한 표현은 "no feature gates" 입니다. 유료 버전에서만 열어 주는 기능 잠금이 없다는 뜻이며, 셀프호스팅 사용자가 SaaS 사용자와 동일한 기능을 받을 수 있다는 약속입니다.

기능 범위와 라이선스 자유도가 양손에 함께 주어진다는 점은 매력적입니다. 그러나 의사결정권자께서는 이 자유도의 반대편에 놓인 두 가지 제약을 함께 보셔야 합니다. 첫째, AGPL 의 네트워크 조항은 AppFlowy 를 수정하여 외부에 서비스 형태로 제공할 경우 소스 공개 의무를 발생시킵니다. 자사 직원 전용 셀프호스팅에서는 거의 문제가 되지 않지만, 외부 고객에게 SaaS 형태로 재판매하는 사업 모형이라면 법무 검토가 필요합니다. 둘째, 넓은 기능 범위는 그만큼 검증되지 않은 영역이 많다는 의미입니다. 별 7만 개라는 외형 뒤에 어떤 기능이 안정 단계인지, 어떤 기능이 아직 실험 단계인지 4장에서 자세히 다룹니다.

정체성을 한 줄로 요약하자면, AppFlowy 는 "Notion 의 기능 폭을 유지하면서 데이터 소유권을 돌려주는 도구" 입니다. 조직이 Notion 의 작업 방식 자체를 좋아하고, 다만 외부 SaaS 의존이 부담스러워 셀프호스팅으로 옮기고 싶다면 AppFlowy 가 가장 자연스러운 후보입니다. 반대로 Notion 의 작업 방식 자체에 큰 매력을 느끼지 못한다면, 굳이 AppFlowy 의 넓은 기능 범위를 떠안을 이유는 약해집니다.

2.1.2 Rust 기반 AppFlowy-Cloud 와 Docker 배포 스택

기술 스택 관점에서 AppFlowy 의 가장 큰 특징은 백엔드와 코어 동기화 엔진을 Rust 로 구현한 점입니다. Rust 는 메모리 안전성과 동시성 처리를 컴파일 단계에서 검증하는 시스템 언어로, 장시간 운영되는 서버에 적합합니다. AppFlowy-Cloud 라는 이름의 서버 구성 요소는 다중 사용자 동기화, 인증, 데이터 저장을 담당하며 이를 컨테이너 형태로 배포합니다. 공식 문서는 운영 방식을 명확히 안내합니다. "The deployment infrastructure relies on Docker containerization." [S2]

Docker 컨테이너 기반 배포는 의사결정권자께서 검토하실 두 가지 의미를 동시에 가집니다. 첫째, 표준 컨테이너 환경 (Docker Compose, Kubernetes, OpenShift 등) 위에서 그대로 동작합니다. 사내에 이미 컨테이너

운영 역량을 갖춘 조직이라면 도입 장벽이 낮습니다. 둘째, 표준 컨테이너 환경이 없는 조직, 즉 가상머신 1대에 모든 서비스를 올려 두고 운영해 온 조직이라면 Docker 학습 비용이 새로 발생합니다. 이 경우 도입 결정과 별개로 운영 체계를 함께 정비해야 합니다.

엔지니어가 함께 검토해야 할 항목은 AppFlowy-Cloud 가 의존하는 외부 구성 요소입니다. AppFlowy 는 PostgreSQL 을 주 저장소로, Redis 를 캐시 계층으로, MinIO 같은 S3 호환 객체 스토리지를 첨부 파일 저장소로 사용합니다. 운영팀 입장에서는 이 네 구성 요소 (AppFlowy-Cloud, PostgreSQL, Redis, 객체 스토리지) 의 가용성과 백업 전략을 모두 설계해야 한다는 뜻입니다. 백업 절차 1개로 끝나는 단순 도구가 아니라, 작은 분산 시스템을 운영하는 셈입니다.

Rust 기반 백엔드와 Docker 배포 스택은 AppFlowy 의 장기 운영 적합성을 받치는 두 기둥입니다. 동시에 의사결정권자께서는 이 기둥이 의미하는 운영 책임을 미리 인지하셔야 합니다. "오픈소스라 무료" 라는 표현이 가리키는 것은 라이선스 비용이지, 운영 비용이 아닙니다. 이 구분을 명확히 하실수록 도입 후 예산 충돌이 줄어듭니다.

2.2 AFFiNE의 정체성과 개발 배경

AFFiNE 은 세 도구 가운데 가장 독특한 사용자 경험을 제안합니다. 문서, 표, 그리고 무한 캔버스 (Whiteboard) 를 하나의 작업 공간에서 자유롭게 오갈 수 있다는 점이 차별점입니다. 그러나 라이선스 구조는 가장 복잡합니다. 단일 라이선스가 아니라 디렉터리별 이중 구조를 채택하여, 도입 전에 반드시 확인해야 할 항목이 있습니다.

2.2.1 문서와 무한 캔버스를 결합한 BlockSuite 기반 AFFiNE

AFFiNE 의 핵심 기술 기반은 BlockSuite 라는 자체 편집기 프레임워크입니다. BlockSuite 는 블록 단위 편집 모델 위에 무한 캔버스 (Edgeless mode) 를 결합하여, 사용자가 문서를 작성하다가 그 자리에서 도식을 그리고, 다시 문서로 돌아오는 흐름을 자연스럽게 지원합니다. 의사결정권자께서 익숙하실 비유는 "Notion 과 Miro 를 한 화면에 합친 도구" 입니다.

개발 속도는 세 도구 가운데 가장 빠릅니다. AFFiNE 의 최신 canary 빌드는 v2026.6.7-canary.1000 으로, 매주 새로운 빌드가 공개됩니다 [S4]. canary 는 "최신 시험 빌드" 를 뜻하며, 안정 빌드와는 별개 채널입니다. 빠른 개발 속도는 새 기능을 빨리 받아볼 수 있다는 장점이지만, 동시에 안정성 검증이 충분한지 따로 확인해야 한다는 의미이기도 합니다. 운영 환경에서는 canary 채널이 아닌 stable 채널을 선택하시기를 권합니다.

대상 사용자 측면에서 AFFiNE 은 "시각적 사고를 자주 하는 팀" 에 가장 잘 들어맞습니다. 예를 들어 제품 기획팀이 와이어프레임과 사용자 흐름도를 글로 풀어 쓴 문서 옆에 나란히 두고 토론하는 방식, 디자인 시스템 팀이 컴포넌트 설명서 안에 즉석에서 도형 예시를 그려 넣는 방식이 자연스럽게 어울립니다. 반대로 정형화된 회의록과 정책 문서가 대부분인 조직에서는 무한 캔버스 기능의 활용도가 낮아 도구의 매력이 절반으로 줄어듭니다.

여러분의 팀이 화이트보드 도구와 문서 도구를 따로 운영하느라 맥락 전환 비용을 치르고 있다면 AFFiNE 은 가장 흥미로운 대안입니다. 반대로 그러한 통합이 시급한 문제가 아니라면, 다음 절에서 다룰 라이선스 구조의 복잡성을 감수할 만한 동기가 약해집니다. 의사결정권자께서는 이 두 질문을 함께 던지셔야 합니다.

2.2.2 packages/backend EE 라이선스의 이중 구조 개요

AFFiNE 의 라이선스 구조는 세 도구 가운데 가장 정교한 동시에 가장 까다롭습니다. 공식 LICENSE 파일은 다음과 같이 두 영역을 나눕니다. "All content that resides under the 'packages/backend' and 'packages/common/native' directory is licensed under the AFFiNE Enterprise Edition license, while content outside of these directories is available under the MIT license." [S6] 즉 코드 저장소 안에서도 디렉터리에 따라 라이선스가 달라집니다. 프런트엔드와 대부분의 공통 코드는 관대한 MIT 라이선스로 자유롭게 쓸 수 있는 반면, 백엔드 핵심 영역은 별도의 Enterprise Edition (EE) 라이선스 적용을 받습니다.

EE 라이선스 본문은 사용 조건을 다음과 같이 규정합니다. EE 영역의 코드는 "may only be used in production, if you (and any entity that you represent) have agreed to, and are in compliance with, the AFFiNE Subscription Terms of Service and possess a valid AFFiNE Enterprise Edition subscription for the correct number of user seats." [S5] 운영 환경 (production) 에서 EE 영역을 사용하려면 유료 구독과 좌석 수 산정이 필수라는 뜻입니다.

다행히 EE 라이선스는 비운영 용도에 대해서는 길을 열어 둡니다. "you may copy and modify the Software for development and testing purposes, without requiring a subscription." [S5] 개발과 시험 목적이라면 구독 없이 자유롭게 복제·수정할 수 있습니다. 이 조건은 PoC (Proof of Concept, 개념 검증) 단계에서는 비용 부담 없이 도구를 충분히 평가할 수 있다는 의미입니다. 다만 평가가 끝난 뒤 운영으로 전환할 때 라이선스 비용이 발생할 수 있다는 점은 반드시 사전에 고려하셔야 합니다.

이 이중 구조는 단순한 가격 정책이 아닙니다. AFFiNE 의 사업 모형 자체를 결정하는 설계입니다. 의사결정권자께서는 이 부분을 정확히 이해하셔야 PoC 결과를 운영 결정으로 옮길 때 예산 충격을 피할 수 있습니다. 디렉터리별 라이선스 적용 범위, EE 와 CE (Community Edition) 의 기능 차이, 좌석 수 산정 기준은 3장에서 라이선스 원문과 표 형식으로 상세히 비교합니다. 이 장에서는 "AFFiNE 은 단일 라이선스 도구가 아니다" 라는 정체성 자체를 먼저 기억해 두시면 충분합니다.

2.3 Outline의 정체성과 개발 배경

Outline 은 세 도구 가운데 범위가 가장 좁고, 그 좁음을 강점으로 만든 도구입니다. 문서·위키 영역에만 집중하여 다른 기능을 의도적으로 덜어냈고, 그 결과 완성도와 안정성 측면에서 가장 성숙한 인상을 줍니다. 라이선스 모형 또한 독특한데, BSL (Business Source License, 비즈니스 소스 라이선스) 1.1 을 채택하여 일정 기간 후 Apache 2.0 으로 자동 전환되는 시한부 구조를 가집니다.

2.3.1 팀 위키에 집중한 TypeScript 기반 Outline의 완성도

Outline 의 정체성은 "팀 위키 전용 도구" 라는 한 줄로 요약됩니다. 프로젝트 관리, 데이터베이스, 무한 캔버스 같은 부가 기능을 의도적으로 배제하고, 문서 작성·구조화·검색·권한 관리에만 집중합니다. 2016년 첫 공개 이후 약 10년에 걸쳐 같은 방향을 유지해 온 결과, 작은 기능 하나하나가 정성스럽게 다듬어져 있다는 평가를 받습니다. 2026년 6월 기준 최신 안정 버전은 v1.8.1 입니다 [S7].

기술 스택 측면에서 Outline 은 단일 언어 비중이 매우 높습니다. 저장소 통계에 따르면 TypeScript 가 전체 코드의 96.6% 를 차지합니다 [S7]. 프런트엔드와 백엔드 모두 TypeScript 로 작성되었다는 의미이며, IT 담당자 입장에서는 유지보수 인력 충원과 코드 이해 측면에서 큰 장점입니다. JavaScript/TypeScript 개발자가 풍부한 국내 인력 시장 환경에서, 단일 언어 스택은 장기 운영 위험을 낮춥니다.

별 개수는 약 38,800 개로 AppFlowy 와 AFFiNE 의 절반 수준입니다. 그러나 별 개수가 적다는 사실을 "인기 없음" 으로 읽으시면 잘못된 결론에 이릅니다. Outline 은 처음부터 팀 위키라는 좁은 영역만 표방했고, 그 영역에서의 만족도가 높아 별을 누른 사용자 한 명 한 명이 실제 운영 사용자에 가깝습니다. 별 7만 개의 일부가 "신기해서 별을 눌렀다" 는 호기심 사용자라면, Outline 의 별 3만 개는 "실제로 쓰고 있다" 는 운영 사용자에 가까울 가능성이 높습니다.

여러분의 조직이 "Notion 대안" 을 찾는 진짜 이유가 사실은 "팀 위키 대안" 이라면, Outline 은 가장 정직한 선택입니다. 반대로 프로젝트 관리, 데이터베이스, 화이트보드 같은 부가 기능이 도입의 핵심 동기라면 Outline 은 범위 자체가 맞지 않습니다. 도구를 도입하기 전에 조직이 진짜로 원하는 것이 무엇인지 먼저 확인하시기를 권합니다.

2.3.2 BSL 1.1 라이선스와 2030년 Apache 2.0 전환 개요

Outline 의 라이선스는 BSL (Business Source License) 1.1 입니다. BSL 은 일정 기간 동안 일부 사용 방식을 제한하다가, 정해진 날짜 이후에는 자동으로 더 관대한 라이선스로 전환되는 시한부 구조를 가집니다. Outline LICENSE 파일은 두 핵심 날짜를 명시합니다. "Change Date: June 6, 2030. Change License: Apache License, Version 2.0." [S8] 2030년 6월 6일이 되면 현재 코드는 자동으로 Apache 2.0 라이선스로 전환된다는 뜻입니다.

전환 이전까지 어떤 사용이 제한되는가 하는 질문이 핵심입니다. BSL 본문은 다음과 같이 규정합니다. "you may not use the Licensed Work for a Document Service — defined as offering the software's functionality to third parties via team and document creation." [S8] Outline 을 SaaS 형태로 제3자에게 재판매하는 것을 막는 조항입니다. 사내 임직원 전용으로 셀프호스팅하는 일반 기업 사용 사례는 이 제약에 해당하지 않습니다. 즉 의사결정권자께서 머릿속에 두실 시나리오 대부분은 BSL 제약과 무관합니다.

다만 두 가지 경우에는 법무 검토가 필요합니다. 첫째, 자회사·관계사를 포함한 그룹사 전체에 단일 인스턴스를 제공하는 방식이 "third parties" 에 해당하는지 여부입니다. 둘째, 외부 협력사·고객사 직원에게 계정을 발급하여 협업하는 방식이 "Document Service" 정의에 들어가는지 여부입니다. 두 경우 모두 일반적인 사내 도구 사용에 가깝지만, 규모와 형태에 따라 해석이 달라질 수 있으므로 도입 전 라이선스 원문 검토를 권합니다.

BSL 의 가장 큰 매력은 시간이 자기편이라는 점입니다. 2030년 6월 6일 이후 모든 제약이 사라지고 Apache 2.0 의 자유로움이 적용됩니다. 장기 운영을 전제로 한 조직에게는 안심이 되는 구조입니다. 다만 그날까지의 기간 동안 사용 방식에 제약이 있다는 사실은 분명히 인지하셔야 합니다. BSL 의 구체 조건, 다른 도구 라이선스와의 비교, 시나리오별 적용 여부는 3장에서 표 형식으로 상세히 다룹니다.

이 장에서 다룬 세 도구의 정체성을 한 문장씩으로 정리하면 다음과 같습니다. AppFlowy 는 "Notion 의 폭을 AGPL 자유로 옮긴 도구" 입니다. AFFiNE 은 "문서와 캔버스를 합친, 이중 라이선스의 정교한 도구" 입니다. Outline 은 "팀 위키 한 길에 시한부 BSL 을 더한 성숙한 도구" 입니다. 이 차이가 분명해지셨다면 3장 라이선스 상세 해설로 자연스럽게 넘어가실 수 있습니다.

3장: 오픈소스 라이선스와 셀프호스팅 운영 스택 비교

여러분이 도입 후보 세 도구의 공식 사이트를 둘러보면, 모두 "오픈소스" 와 "무료 셀프호스팅" 이라는 표현을 전면에 내세웁니다. 그러나 이 표현 뒤에 놓인 라이선스 조항을 한 줄씩 읽어 보면, 사내 배포 허용 범위와 재배포 의무, 유료 트리거 조건이 세 도구에서 모두 다릅니다. AppFlowy 는 AGPL-3.0 을 그대로 적용했고, AFFiNE

은 MIT 와 EE 라이선스를 디렉터리 경계로 나눈 이중 구조를 택했으며, Outline 은 BSL 1.1 이라는 시한부 제한 라이선스를 사용합니다 [S3] [S5] [S6] [S8].

도입 의사결정에서 라이선스는 기능 비교표의 한 칸이 아닙니다. 라이선스는 4년 뒤·10년 뒤의 운영 비용을 결정하고, 사내 다부서 배포가 합법인지 위법인지 가르고, 소스 코드 공개 의무를 사외 변호사 자문 대상에 올릴지 말지를 정합니다. 라이선스 한 줄을 잘못 읽으면 도입 1년 뒤에 라이선스 위반 통지서를 받거나, 도입 3년 뒤에 갑작스러운 유료 전환 요구를 받을 수 있습니다.

특히 국내 기업 법무 담당자 관점에서 이 세 라이선스는 익숙한 GPL·MIT·Apache 와 결이 다릅니다. AGPL-3.0 은 SaaS 형태로 외부에 서비스를 제공할 때도 소스 공개 의무가 살아 있고, BSL 은 미국 MariaDB 가 처음 만든 비교적 새로운 라이선스이며, AFFiNE 의 CE+EE 이중 구조는 공식 LICENSE 원문 단 한 줄로 디렉터리 경계를 가릅니다. 익숙하지 않은 조항은 도입 의사결정 회의에서 "일단 무료니까 깔고 보자" 라는 결론으로 흐르기 쉽습니다.

이번 장에서는 세 라이선스의 핵심 조항을 원문 인용과 함께 풀이하고, 국내 사내 배포·계열사 배포·외부 고객 서비스라는 세 가지 시나리오에서 각 라이선스가 어떤 의무를 발생시키는지 정리합니다. 마지막으로 운영 관점에서 Docker 의존 서비스 구성과 외부 IdP 요구 여부를 비교하여, 망분리 환경의 적합도까지 함께 다룹니다.

3.1 AppFlowy AGPL-3.0 해설

AppFlowy 는 저장소 전체에 단일 라이선스를 적용한다는 점에서 세 도구 중 가장 단순합니다. 공식 저장소는 "All content that resides under the AppFlowy repository is licensed under the GNU Affero General Public License v3" 라고 명시하며, 라이선스 수수료·사용자당 과금·기능 제한이 모두 없다고 선언합니다 [S3]. 라이선스 해석은 단순하지만, AGPL-3.0 자체가 가진 네트워크 사용 조항이 도입 의사결정의 핵심 변수입니다.

3.1.1 AGPL-3.0의 재배포·SaaS 소스 공개 의무 조항

AGPL-3.0 의 가장 큰 특징은 GPL-3.0 에 비해 한 단계 강화된 "네트워크 제공" 조항입니다. 일반 GPL-3.0 은 소프트웨어를 외부에 바이너리 형태로 배포할 때만 소스 공개 의무가 발생합니다. AGPL-3.0 은 여기에 더해 사용자가 네트워크를 통해 소프트웨어와 상호작용할 때도 사용자에게 소스 코드를 제공할 의무를 더했습니다. 즉 사내에서 수정한 AppFlowy 를 외부 고객이 웹 브라우저로 접속해 사용한다면, 그 외부 고객에게도 수정 소스를 받을 권리가 생깁니다 [S3].

그림 3-1. 세 도구의 라이선스 비교표. AppFlowy(AGPL-3.0) · AFFiNE(MIT+EE 이중) · Outline(BSL 1.1)의 사내 배포 허용 범위, 재배포 의무, 유료 트리거 조건, 외부 SaaS 제공 가능 여부를 한 장으로 비교합니다. 이 그림은 본문의 조항 해설을 한눈에 대조하여 법무 담당자가 도입 후보 선정 회의에서 즉시 참조할 수 있도록 의도했습니다.

이 조항이 두려운 곳은 SaaS 사업자입니다. 만약 어느 회사가 AppFlowy 를 기반으로 자체 협업 서비스를 만들어 외부에 유료로 제공한다면, 그 서비스의 모든 수정 사항을 외부 고객이 요구할 때 공개해야 합니다. AppFlowy 가 라이선스 페이지에 "no license fees" 를 강조하는 동시에 AGPL-3.0 을 고른 이유가 바로 여기에 있습니다 [S3]. 무료로 쓰되, 외부 SaaS 사업으로 수익을 내려면 소스를 함께 공개하라는 정책입니다.

다만 사내 배포라면 이 조항이 발동하지 않습니다. AGPL-3.0 은 "외부 사용자에게 네트워크로 제공" 할 때 소스 공개 의무를 발생시키므로, 동일 법인 내부 직원만 접속하는 사내 시스템은 외부 제공이 아닙니다. 여기서 핵심은 "동일 법인" 의 범위입니다. 모회사·자회사 관계의 다른 법인 직원이 접속한다면 외부 제공으로 해석될 여지가 있고, 이 경우 법무 자문이 필요합니다.

3.1.2 국내 사내 배포 시 AGPL-3.0의 실무 함의

국내 기업이 AppFlowy 를 사내에서 쓸 때 가장 자주 마주치는 질문은 두 가지입니다. 첫째, 사내에서 소스를 수정하면 공개해야 합니까. 둘째, 계열사 직원도 함께 쓰면 공개해야 합니까. 첫 번째 질문의 답은 "사내 직원만 접속한다면 공개 의무가 발동하지 않는다" 입니다. AGPL-3.0 의 네트워크 조항은 "외부 사용자" 의 접속을 전제로 하기 때문입니다.

두 번째 질문의 답은 "계열사 관계에 따라 다르다" 입니다. 동일 법인 내 부서 간 사용은 외부 제공이 아니지만, 별도 법인인 계열사 직원이 접속한다면 외부 제공으로 볼 여지가 생깁니다. 이때 법무 담당자는 두 법인 간 관계를 어떻게 정의할지 선택해야 합니다. 한 가지 안전한 방법은 각 계열사가 자체 AppFlowy 인스턴스를 운영하여 법인 경계를 명확히 나누는 것입니다.

또 한 가지 자주 묻는 질문은 "AppFlowy 소스를 한 줄도 수정하지 않으면 의무가 없는가" 입니다. 그렇습니다. 무수정 배포라면 AGPL-3.0 은 소스 공개 의무를 발동하지 않습니다. 그러나 실무에서는 사내 SSO 연동, 한국어 메뉴 수정, 한국 표준시 적용 같은 작은 수정이 거의 항상 들어갑니다. 따라서 "우리는 수정 안 할 거니까 괜찮다" 라는 가정은 도입 1년 뒤에 깨질 가능성이 큽니다.

국내 법무 실무 관점에서 AGPL-3.0 의 가장 큰 장점은 조항이 명확하고 해석 논쟁이 적다는 점입니다. Free Software Foundation 이 공식 FAQ 를 운영하고, 국제 판례도 누적되어 있어 사외 변호사 자문에서 명확한 답을 얻기 쉽습니다. 국내에서도 정보통신산업진흥원(NIPA) 공개SW포털과 한국저작권위원회의 「오픈소스SW 라이선스 가이드」가 AGPL 적용 사례와 의무를 정리해 두었으므로 사내 검토 시 1차 참조로 활용하실 수 있습니다. 반면 AGPL-3.0 의 가장 큰 단점은 "외부 SaaS 로 확장할 가능성이 조금이라도 있다면 도입 시점부터 소스 공개 정책을 함께 세워야 한다" 라는 점입니다.

한 가지 추가로 강조할 사실은 AGPL-3.0 제13조(Remote Network Interaction)의 발동 조건이 "프로그램을 수정한 경우" 라는 점입니다. 무수정 그대로 사내 배포한다면 13조 자체가 발동하지 않으며, 수정한 코드가 있을 때 비로소 해당 소스 제공 의무가 외부 사용자에게 발생합니다. 이 점은 도입 초기 "수정 없이 그대로 띄운다" 라는 정책으로 의무를 회피할 수 있다는 의미가 아니라, 거의 모든 실제 운영에서 사내 SSO·로그 연계·UI 한글화 같은 수정이 누적되기 때문에 도입 시점부터 소스 관리 절차를 세워야 한다는 의미로 해석하시는 편이 안전합니다.

3.2 AFFiNE CE+EE 이중 라이선스 유료 트리거

AFFiNE 은 한 저장소 안에서 두 라이선스를 동시에 적용한다는 점에서 세 도구 중 구조가 가장 복잡합니다. 공식 LICENSE 원문은 디렉터리 경로 한 줄로 MIT 영역과 Enterprise Edition 영역을 나누고, EE 영역의 프로덕션 사용 조건을 따로 정해 두었습니다. 도입 의사결정 시 가장 먼저 봐야 할 것은 "우리가 쓰려는 기능이 어느 디렉터리에 있는가" 입니다 [S5] [S6].

3.2.1 packages/backend EE 영역과 MIT 영역의 디렉터리 경계

AFFiNE 공식 LICENSE 는 "All content that resides under the 'packages/backend' and 'packages/common/native' directory is licensed under the AFFiNE Enterprise Edition license, while content outside of these directories is available under the MIT license" 라고 명시합니다 [S6]. 두 디렉터리 안의 코드는 EE 라이선스를, 그 밖의 코드는 MIT 라이선스를 따른다는 단 한 줄의 경계 정의입니다.

이 경계가 가르는 기능은 명확합니다. `packages/backend` 는 서버 측 API·인증·데이터 동기화·실시간 협업 백엔드를 담고, `packages/common/native` 는 운영체제 의존 네이티브 모듈을 담습니다. 즉 사내 다인원이 함께 쓰는 셀프호스팅 협업 기능은 모두 EE 영역에 들어 있습니다. 반대로 클라이언트 UI·에디터·블록 시스템·로컬 데이터 저장 로직은 모두 MIT 영역입니다.

도입 의사결정 관점에서 이 구분은 결정적입니다. "우리는 단일 사용자가 로컬에서 노트 작성만 한다" 라면 EE 영역을 건드릴 일이 없고 MIT 라이선스로 자유롭게 쓸 수 있습니다. 그러나 "팀 단위로 서버에 올려 함께 쓰겠다" 라는 전형적인 셀프호스팅 시나리오라면 EE 영역의 백엔드를 반드시 띄워야 하고, EE 라이선스 조항이 발동합니다.

여러분이 도입 검토 단계에서 확인할 첫 번째 사실은 "셀프호스팅 = EE 영역 사용" 이라는 등식입니다. AFFiNE 의 MIT 영역만으로 팀 협업을 구성할 방법은 사실상 없습니다. 따라서 AFFiNE 셀프호스팅을 고려한다면 EE 라이선스의 프로덕션 조항을 사외 변호사와 함께 읽어야 합니다.

3.2.2 프로덕션 환경 유료 트리거 조건과 개발·테스트 무료 사용 범위

AFFiNE EE 라이선스의 핵심 문장은 두 개입니다. 첫째, "may only be used in production, if you (and any entity that you represent) have agreed to, and are in compliance with, the AFFiNE Subscription Terms of Service and possess a valid AFFiNE Enterprise Edition subscription for the correct number of user seats" 라는 조항입니다 [S5]. 프로덕션 환경에서는 사용자 좌석 수만큼 유료 구독을 보유해야만 한다는 뜻입니다.

둘째, "you may copy and modify the Software for development and testing purposes, without requiring a subscription" 이라는 조항입니다 [S5]. 개발·테스트 목적의 사용은 무료 허용 범위 안에 있습니다. 즉 사내 평가용 PoC 환경에서 일주일·한 달 단위로 EE 백엔드를 띄워 검토하는 활동은 라이선스 위반이 아닙니다.

문제는 "프로덕션" 의 정의입니다. AFFiNE 공식 LICENSE 원문은 프로덕션을 명시 정의하지 않습니다. 통상적으로 "실제 업무 데이터를 다루며 일상 업무에 사용되는 환경" 으로 해석합니다. 따라서 사내 직원 5명이 일상 업무 노트를 EE 백엔드에 저장한다면, 좌석 수가 적더라도 프로덕션 사용에 해당하여 구독 의무가 발동할 가능성이 큽니다. 반대로 IT 부서가 도입 검토를 위해 띄운 평가 환경은 개발·테스트 범주에 들어갑니다.

여기서 짚어야 할 미해결 사실이 하나 있습니다. AFFiNE EE 라이선스 원문에는 "몇 명까지 무료 프로덕션 사용이 가능한가" 라는 무료 상한선이 명시되어 있지 않습니다. 공식 GitHub Discussion 과 사용자 문의에서 자주 등장하는 질문이지만, 공식 LICENSE 원문은 "프로덕션 사용 시 좌석 수만큼 구독 필요" 라고만 적습니다. 따라서 "1인용 셀프호스팅은 무료다" 같은 통념은 공식 근거가 없으며, 정확한 적용 범위는 AFFiNE 측 영업 채널을 통해 사전에 확인해야 합니다.

국내 법무 실무에서 AFFiNE 의 가장 큰 위험은 "도입 1년 뒤 갑작스러운 유료 전환 요구" 입니다. 도입 초기에 "1인 PoC 라서 괜찮다" 라며 시작했다가, 사내 다부서로 확산되는 순간 프로덕션 정의에 걸리고, 좌석 수만큼 즉시 구독료가 발생합니다. 따라서 AFFiNE 을 도입한다면, 도입 시점에 향후 24개월 동안의 좌석 수 시나리오를 미리 그려 두는 편이 안전합니다.

3.3 Outline BSL 1.1 허용 범위와 2030년 전환

Outline 의 라이선스는 BSL 1.1 (Business Source License) 이라는, 비교적 최근에 등장한 시한부 라이선스입니다. BSL 은 "지정된 날짜까지는 특정 용도를 제한하다가, 그 날짜가 지나면 자동으로 오픈소스 라이선스로 전환된다" 라는 독특한 구조를 가집니다. Outline 의 경우 2030년 6월 6일이 그 전환 기점이며, 전환 후 라이선스는 Apache 2.0 입니다 [S8].

3.3.1 BSL 1.1의 "Document Service" 금지 조항 해설

Outline 의 LICENSE 원문은 "you may not use the Licensed Work for a Document Service — defined as offering the software's functionality to third parties via team and document creation" 라는 한 줄을 핵심 제한으로 둡니다 [S8]. 풀이하면 "Outline 의 기능을 제3자에게 팀·문서 생성 형태로 제공하는 서비스" 를 만들 수 없다는 뜻입니다. 정확히 SaaS 사업화를 막는 조항입니다.

이 조항이 막는 사용 사례는 명확합니다. 어느 회사가 Outline 소스를 받아 자체 협업 SaaS 를 만들어 외부 고객에게 팔면, BSL 위반입니다. 반대로 사내 직원만 쓰는 셀프호스팅은 "제3자 제공" 이 아니므로 허용됩니다. AGPL-3.0 이 "외부 SaaS 도 가능하되 소스 공개" 를 요구한다면, BSL 1.1 은 "외부 SaaS 자체를 금지" 합니다.

문제는 "제3자" 의 정의입니다. 동일 법인 내 직원은 명백히 제3자가 아닙니다. 그러나 자회사·계열사 직원, 외주 협력사 직원, 공공기관 산하 기관 직원은 해석의 회색지대에 들어갑니다. 국내 대기업이 그룹 전체에 단일 Outline 인스턴스를 배포한다면, "그룹 본사가 운영하는 인스턴스를 계열사 직원이 접속하여 사용" 하는 구도가 됩니다. 이때 계열사 직원이 BSL 의 "third parties" 인지 여부는 사외 변호사 자문이 필요합니다.

미해결 사실 한 가지를 짚으면, Outline 공식 LICENSE 원문은 "사내 다부서·계열사 배포" 의 적법성을 직접 명시하지 않습니다. 따라서 국내 기업이 그룹사 단위로 Outline 을 도입할 때는, 도입 전에 사외 변호사의 의견서를 받아 두는 편이 안전합니다.

3.3.2 2030년 Apache 2.0 전환 조항의 실무 의미

Outline LICENSE 의 또 다른 핵심 줄은 "Change Date: June 6, 2030. Change License: Apache License, Version 2.0" 입니다 [S8]. 2030년 6월 6일이 되면, Outline 의 모든 코드가 Apache 2.0 으로 자동 전환됩니다. Apache 2.0 은 가장 관대한 오픈소스 라이선스 중 하나이며, SaaS 사업화도 사내 배포도 모두 자유롭습니다.

이 조항이 도입 의사결정에 주는 의미는 두 가지입니다. 첫째, 2030년 이후에는 BSL 의 SaaS 금지 조항이 사라지므로, 그때부터는 Outline 기반 SaaS 사업화도 가능해집니다. 둘째, 2030년 6월 6일까지 출시된 버전만 Apache 2.0 으로 전환되며, 그 이후 출시될 새로운 메이저 버전은 각각 새로운 BSL 조항을 가질 가능성이 있습니다.

도입 의사결정 관점에서 BSL 의 시한부 구조는 양날의 검입니다. 한쪽 날은 "4년만 기다리면 완전 오픈소스가 된다" 라는 안심입니다. 다른 한쪽 날은 "지금 도입하는 버전과 4년 뒤 신규 메이저 버전 사이에 라이선스 정책 변동 가능성이 있다" 라는 불확실성입니다.

따라서 Outline 을 도입한다면 두 가지 시나리오를 함께 검토해야 합니다. 단기 시나리오는 "2030년 이전에는 사내 직원만 쓰는 셀프호스팅으로 운영" 입니다. 장기 시나리오는 "2030년 이후 Apache 2.0 전환을 기점으로 외부 협력사·고객까지 확장 가능 여부 재검토" 입니다. 이 두 시나리오를 미리 정리해 두면, 4년 뒤 라이선스 전환이 도입 의사결정의 발목을 잡지 않습니다.

3.4 셀프호스팅 Docker 운영 스택 비교

라이선스가 법무 관점의 도입 의사결정 변수라면, Docker 운영 스택은 인프라 관점의 도입 의사결정 변수입니다. 세 도구 모두 Docker 컨테이너 기반 셀프호스팅을 지원하지만, 의존하는 외부 서비스의 수와 종류는 도구마다 차이가 큽니다. 특히 국내 공공·금융권에서 흔한 망분리 환경에서는 외부 IdP 의존이 도입 가능 여부를 가르는 결정적 요인이 됩니다 [S2] [S7].

3.4.1 세 도구의 Docker 의존 서비스 비교

AppFlowy-Cloud 의 공식 저장소는 환경별로 세 가지 docker-compose 파일을 제공합니다. "The deployment infrastructure relies on Docker containerization. The repository includes multiple docker-compose files for different environments: docker-compose.yml (production), docker-compose-dev.yml (development), docker-compose-ci.yml (CI)" 라고 명시되어 있습니다 [S2]. 프로덕션 스택에는 PostgreSQL, Redis, MinIO (S3 호환 객체 스토리지), AppFlowy Cloud 서버, GoTrue (인증 서비스) 가 포함됩니다.

Outline 의 셀프호스팅 스택은 PostgreSQL, Redis, S3 호환 객체 스토리지, 그리고 외부 OIDC IdP 가 필수입니다 [S7]. AFFiNE 의 셀프호스팅 스택은 PostgreSQL, Redis 를 기본으로 하며, 객체 스토리지는 로컬 디스크 또는 S3 호환 스토리지 중 선택할 수 있습니다.

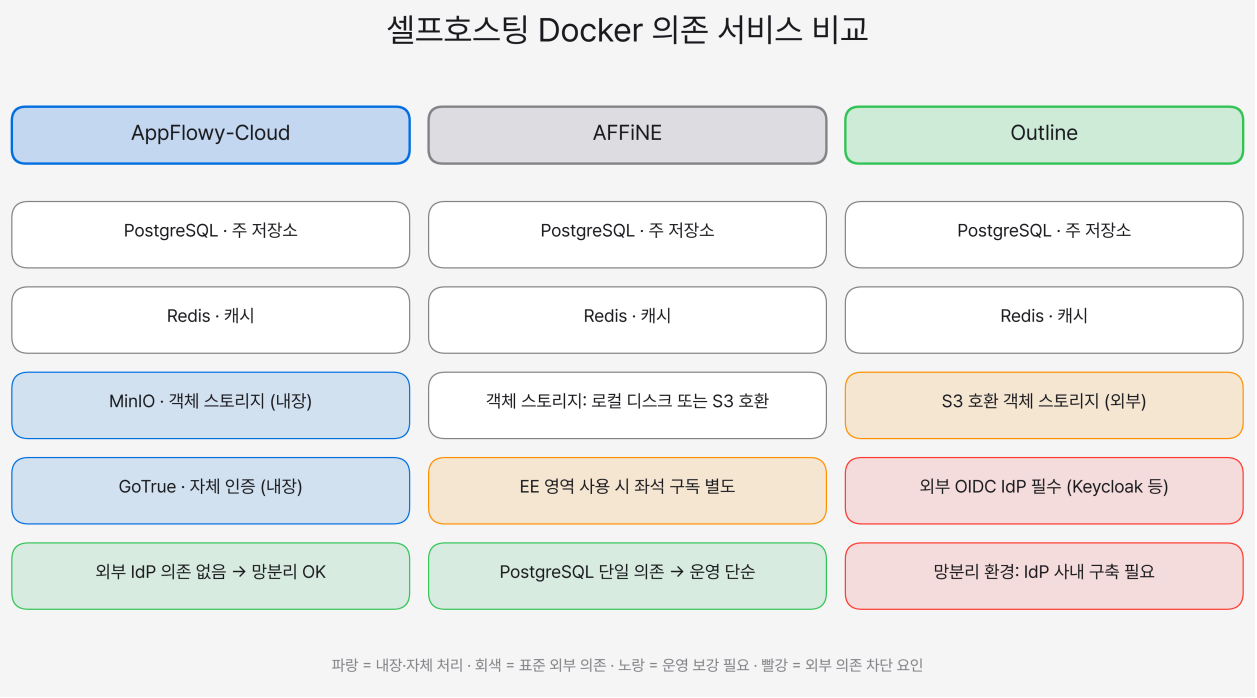

그림 3-2. 세 도구의 Docker 의존 서비스 구성도. AppFlowy / Outline / AFFiNE 의 셀프호스팅 스택에 포함되는 컨테이너와 외부 의존 서비스 (인증 IdP, 객체 스토리지, 데이터베이스, 캐시) 를 한 장으로 비교합니다. 이 그림은 인프라 담당자가 도입 후보별 운영 복잡도를 한눈에 견적할 수 있도록 의도했습니다.

세 도구의 공통 의존성은 PostgreSQL 과 Redis 입니다. 두 서비스는 국내 대부분의 사내 인프라에 이미 존재하므로 신규 도입 부담이 작습니다. 차이를 만드는 의존성은 객체 스토리지와 인증 서비스입니다. 객체 스토리지의 경우 AppFlowy 는 MinIO 를 컨테이너 안에 함께 띄우는 일체형 구성을 제공하여 별도 S3 계약이 필요 없습니다. Outline 은 S3 호환 스토리지를 외부 의존으로 가정하여 사내 MinIO 또는 클라우드 S3 를 별도 구성해야 합니다.

3.4.2 망분리 환경 적합도와 외부 IdP 의존성

국내 공공기관·금융기관의 망분리 환경에서는 「국가정보보안 기본지침」(국가정보원)과 「전자금융감독규정」 제15조(금융보안원 망분리 가이드)에 따라 외부 인터넷 접속이 원천 차단됩니다. 이 환경에서 Outline 은 결정적 약점을 가집니다. 공식 Outline 은 외부 OIDC/SSO IdP 가 필수이며, 자체 ID/비밀번호 로그인을 권장 구성으로 제공하지 않습니다 [S7]. 따라서 망분리 망 안에서 Outline 을 띄우려면, 동일 망 안에 OIDC 호환 IdP(Keycloak·Authentik 등 오픈소스, 또는 국내 SSO 제품)를 별도 구축해야 합니다.

AppFlowy-Cloud 는 GoTrue 라는 자체 인증 서비스를 컨테이너에 포함하여, 외부 IdP 없이도 사내 ID/비밀번호 로그인이 가능합니다 [S2]. 망분리 환경에서 가장 간단하게 띄울 수 있는 구성입니다. AFFiNE 역시 자체 사용자 관리 기능을 가지지만, 셀프호스팅 시 추가 설정이 필요한 부분이 있습니다.

망분리 환경 적합도를 정리하면 다음과 같이 갈립니다. AppFlowy 는 외부 의존성이 가장 낮아 망분리 망 안에서 단독 운영이 가능합니다. AFFiNE 은 사용자 관리는 자체 해결되지만 EE 라이선스의 프로덕션 사용 조건과

망분리 환경의 라이선스 활성화 절차가 별도 검토 사항입니다. Outline 은 외부 OIDC IdP 가 사실상 필수여서, 망분리 환경에 IdP 가 이미 운영 중이지 않다면 도입 비용이 급격히 늘어납니다.

여러분이 공공·금융권 도입을 검토한다면, 라이선스 비교와 Docker 의존 서비스 비교를 함께 봐야 합니다. 라이선스가 도입 자체의 합법성을 결정한다면, Docker 의존 서비스 구성은 도입 이후의 운영 복잡도와 망분리 적합도를 결정하기 때문입니다. 두 관점이 모두 통과해야 도입 의사결정의 위험이 진정으로 통제됩니다.

4장: Claude(LLM) 연동 가능성 최중점 비교

여러분이 셀프호스팅 협업 도구를 검토하실 때 가장 먼저 떠올리시는 기준은 대개 UI 완성도, 데이터베이스 뷰의 다양성, 모바일 앱 품질 같은 사용자 경험 영역입니다. 그러나 2026년 현재의 평가 기준에는 항목 하나가 더 추가되어야 합니다. 바로 Claude 같은 대형 언어 모델(LLM, Large Language Model)이 그 도구의 문서 저장소를 얼마나 매끄럽게 읽고 쓸 수 있느냐 하는 문제입니다. 이 능력은 표면적인 기능 목록만 봐서는 드러나지 않으며, MCP(Model Context Protocol) 공식 지원 여부와 REST·GraphQL API 의 검색 능력을 따져봐야 비로소 보입니다.

본 장은 백서 전체에서 가장 비중이 큰 장으로 배치되었습니다. AppFlowy·AFFiNE·Outline 세 도구가 Claude 와 어떻게 연결되는지, 그 연결 경로의 공식성·완성도·실용성을 평가하기 위해서입니다. 같은 셀프호스팅 카테고리에 묶여 있어도 세 도구는 AI 연동 측면에서 서로 다른 위치에 있습니다. 한쪽은 공식 안내가 명문화되어 있고, 한쪽은 커뮤니티 서버 한 개에 의존하며, 또 한쪽은 외부 LLM 과의 직접 통로가 사실상 막혀 있는 상태입니다.

이러한 차이를 파악하지 못한 채 도구를 선택하시면, 도입 후 6개월 이내에 사내 RAG(Retrieval-Augmented Generation, 검색 보강 생성) 파이프라인을 구축할 때 큰 비용을 다시 치르게 되십니다. 문서 저장소를 옮기는 작업은 단순한 데이터 이전이 아니라 권한 체계와 워크플로우의 재구성을 수반하기 때문입니다. 그래서 본 장은 의사결정권자께서 도구 선정 단계에서 Claude 연동 가능성을 별도의 독립 항목으로 다루시기를 권합니다.

장의 흐름은 다음과 같이 잡았습니다. 먼저 AI 연동을 독립 평가 기준으로 삼아야 하는 이유를 정리합니다. 이어서 AppFlowy·AFFiNE·Outline 순서로 각 도구의 AI 접근 경로와 한계를 살펴봅니다. 마지막으로 세 도구를 종합 비교하고 도입 시나리오별 권고를 제시합니다. 인용은 본 백서가 정리한 외부 출처 인덱스를 그대로 사용하며, 모든 주장은 공식 문서나 깃허브 저장소의 실측 자료에 근거합니다.

4.1 AI 연동을 독립 평가 기준으로 삼는 이유

협업 도구 비교 콘텐츠는 대체로 UI 스크린샷과 기능 매트릭스를 나란히 두는 방식으로 구성됩니다. 이 방식은 사용자 체험을 직관적으로 보여주는 장점이 있지만, 문서 저장소가 점차 AI 의 컨텍스트 공급원으로 활용되는 흐름을 반영하지 못합니다. 본 절은 AI 연동을 별도 평가 기준으로 두어야 하는 근거를 두 항목으로 풀어드립니다.

4.1.1 문서 저장소를 RAG·MCP 소스로 활용하는 시대의 도구 평가

여러분의 조직이 사내 문서를 모아둔 협업 도구는 이제 단순한 저장소가 아닙니다. RAG 파이프라인의 1차 데이터 소스이자, MCP 를 통해 Claude 같은 LLM 이 직접 조회·갱신하는 라이브 백엔드 역할까지 맡고 있습니다. MCP 는 Anthropic 이 공개한 표준 프로토콜로, AI 어시스턴트가 외부 시스템의 도구·자원·프롬프트를 일관된 인터페이스로 호출할 수 있게 해줍니다.

이 흐름 안에서 문서 저장소를 고르실 때 점검하셔야 할 첫 질문은 "Claude 가 이 저장소를 어떻게 읽어낼 수 있는가" 입니다. 답이 "공식 MCP 서버가 있다" 라면 도입 첫날부터 Claude Desktop 이나 사내 에이전트가 바로 연결됩니다. 답이 "커뮤니티 서버 한 개가 깃허브에 있다" 라면, 그 서버의 활성도와 신뢰도를 추가로 검증하셔야 합니다. 답이 "REST API 는 있지만 부분 검색이 약하다" 라면 별도의 임베딩 파이프라인을 구축하셔야 합니다.

두 번째 점검 사항은 검색의 정밀도입니다. Claude 가 문서 저장소를 활용하려면 키워드 검색만으로는 부족합니다. 본문 일부를 발췌해 컨텍스트로 넘기는 부분 검색 능력, 검색 결과의 메타데이터(작성자·갱신일·태그) 노출, 권한 경계를 존중하는 응답 필터링이 모두 필요합니다. 이 능력이 없는 도구는 Claude 연동을 형식적으로만 지원하게 됩니다.

세 번째 점검 사항은 양방향성입니다. 단순히 읽기만 가능한 도구는 RAG 소스로만 쓰입니다. 그러나 Claude 가 문서를 생성하고 갱신하며 데이터베이스 행을 추가할 수 있다면, 그 도구는 에이전트 자동화의 워크스페이스가 됩니다. 이 차이는 사용 시나리오의 폭을 결정합니다.

4.1.2 일반 비교글의 UI 중심 평가가 놓치는 AI 시대 도구 가치

시중의 비교 콘텐츠는 대부분 "AppFlowy 는 오프라인 우선이라 빠르다", "AFFiNE 는 캔버스 뷰가 독창적이다", "Outline 은 위키 구조가 깔끔하다" 같은 문장으로 결론을 맺습니다. 이러한 평가는 사용자 1인의 체험 가치를 정확히 짚지만, 조직 단위에서 LLM 과 결합했을 때의 가치를 평가하기에는 불충분합니다.

UI 가 아무리 매끄러워도 외부 LLM 과의 통로가 제한적이면, 그 도구는 사내 AI 에이전트의 1차 소스가 되기 어렵습니다. 반대로 UI 가 다소 투박해도 공식 MCP 채널이 열려 있고 검색 API 가 RAG 친화적이라면, 그 도구는 AI 시대의 핵심 자산이 됩니다. 평가의 무게중심을 옮겨야 하는 이유가 여기에 있습니다.

또 하나 주의하실 점은 "AI 기능 내장 여부" 를 두고 도구의 AI 친화성을 평가하면 곤란하다는 것입니다. 도구 내부에 챗봇이 들어 있다고 해서 외부 Claude 와 잘 연결되는 것은 아닙니다. 오히려 내장 AI 가 폐쇄적으로 동작하면 외부 LLM 연결을 막거나 어렵게 만드는 경우도 있습니다. 이 문서는 두 개념을 분리해서 다룹니다.

마지막으로 의사결정권자께서 기억하셔야 할 원칙은 "셀프호스팅이라고 다 같지 않다" 는 점입니다. 데이터 주권을 확보하셨다 해도 LLM 통로가 차단되어 있다면 데이터의 활용도가 떨어집니다. AI 연동 가능성은 셀프호스팅 도구를 고르실 때 데이터 주권과 동등한 비중으로 다루셔야 할 평가 기준입니다.

4.2 AppFlowy의 AI 접근 경로와 한계

AppFlowy 는 Rust·Flutter 기반의 오픈소스 협업 도구로 Notion 의 대안으로 가장 자주 언급됩니다. 그러나 외부 LLM 과의 연결 경로를 따져보면 평가가 한층 신중해집니다. 본 절은 AppFlowy 의 내장 AI 와 외부 연결 옵션, 그리고 커뮤니티 MCP 서버 한 개의 실용성을 차례로 살펴봅니다.

4.2.1 AppFlowy 내장 AI 와 외부 LLM 직접 연결의 제약

AppFlowy 는 자체적으로 AppFlowy AI 라는 내장 기능을 제공합니다 [S1]. 사용자가 페이지 안에서 글쓰기 보조와 요약을 수행할 수 있도록 설계되었으며, 셀프호스팅 환경에서도 일정 범위 안에서 작동합니다. 다만 이 내장 기능과 외부 Claude 의 직접 연결은 별개의 문제입니다.

외부 LLM 이 AppFlowy 데이터에 접근하시려면 공식 API 또는 MCP 채널이 필요합니다. 그러나 2026년 6월 시점에서 AppFlowy 의 공식 MCP 서버는 존재하지 않습니다 [S11]. AppFlowy Cloud 의 REST API 도 공개

범위가 제한적이어서, 외부 에이전트가 일반적인 RAG 소스화 용도로 활용하기 어렵습니다. 셀프호스팅 환경에서는 PostgreSQL 데이터베이스를 직접 조회하는 우회 경로가 거론되지만 권한·스키마 안정성 측면에서 권장되지 않습니다.

이러한 상황은 AppFlowy 의 설계 철학과 무관하지 않습니다. AppFlowy 는 오프라인 우선과 데이터 주권을 강조하는 도구로 시작했으며, 외부 통합 인터페이스를 우선 과제로 두지 않았습니다. 결과적으로 외부 LLM 과의 직접 연결을 시도하시는 조직은 자체 통합 레이어를 구현하시거나 커뮤니티 프로젝트에 의존하셔야 합니다.

엔지니어 여러분이 AppFlowy 를 검토하실 때 가장 먼저 확인하셔야 할 사항은 사내 RAG 파이프라인과의 호환 가능성입니다. 임베딩 색인을 위한 정기 export 경로가 있는지, 변경 이벤트를 감지하는 webhook 이 있는지, 권한 메타데이터를 추출할 수 있는지를 점검하시기 바랍니다. 현재까지 이 영역의 공식 지원은 부족하며, 자체 구현이 필요한 부분이 큽니다.

4.2.2 커뮤니티 MCP(AppFlowy Cloud MCP Jemo69) 의 실용성 평가

AppFlowy 의 외부 LLM 연결 공백을 메우는 시도가 커뮤니티에서 진행되고 있습니다. 대표적인 사례가 Jemo69 가 깃허브에 공개한 AppFlowy Cloud MCP 서버입니다 [S11]. 이 서버는 AppFlowy Cloud 의 내부 API 를 MCP 표준에 맞게 노출시키는 어댑터로 작동합니다.

제공 도구는 총 9개로 집계됩니다 [S11]. 인증 관련 2개, 워크스페이스·데이터베이스 조회 3개, 행 조작 4개의 구성입니다. 인증 도구는 토큰을 통한 사용자 식별을 담당하고, 워크스페이스·데이터베이스 도구는 구조와 메타데이터를 노출시키며, 행 조작 도구는 데이터베이스 행의 생성·수정·삭제·조회를 지원합니다. 페이지 단위의 풀텍스트 검색이나 변경 이벤트 구독은 포함되어 있지 않습니다.

이 구성을 두고 실용성을 판단하시면 다음과 같습니다. 사내 RAG 소스로 활용하시려는 시나리오에서는 풀텍스트 검색 부재가 큰 제약입니다. 검색이 없으면 Claude 가 컨텍스트를 자율적으로 가져올 방법이 좁아지며, 별도의 임베딩 파이프라인을 추가로 구축하셔야 합니다. 반면 데이터베이스 중심의 정형 자동화 시나리오에서는 행 조작 도구 4개가 일정 수준 유용합니다.

또 다른 고려 사항은 커뮤니티 서버의 운영 책임 소재입니다. 공식 팀이 유지보수하지 않는 어댑터는 AppFlowy 본체의 API 변경에 즉각 대응하지 못할 수 있습니다. 운영 환경에서 LLM 에이전트가 의존하는 통로가 갑작스럽게 끊기면 업무 흐름이 멈출 위험이 있습니다. 의사결정권자께서는 이 점을 도입 전에 명확히 평가하시기 바랍니다.

종합하면 AppFlowy 는 데이터 주권과 사용자 경험 측면에서 매력적이지만, 2026년 6월 시점의 Claude 연동 성숙도는 세 도구 중 가장 낮습니다. 본격적인 AI 에이전트 워크스페이스로 쓰시려면 자체 통합 작업이 상당히 필요합니다.

4.3 AFFiNE 서드파티 MCP 서버 구조

AFFiNE 는 문서·화이트보드·데이터베이스를 단일 캔버스에서 다루는 점이 특징인 도구입니다. AI 연동 측면에서는 공식 MCP 서버는 없지만, 활성도 높은 서드파티 서버가 한 개 존재합니다. 본 절은 그 서버의 기술 구조와 v1.6.0 의 기능 확장을 살펴봅니다.

4.3.1 GraphQL·WebSocket·CRDT 기반 affine-mcp-server 구조

DAWNCR0W 가 깃허브에 공개한 affine-mcp-server 는 AFFiNE 워크스페이스에 대한 표준 MCP 인터페이스를 제공합니다. 공식 저장소 설명에 따르면 이 서버는 "AFFiNE workspaces 와 GraphQL·WebSocket 연결을 통해 통합되어, 문서 작업·실시간 협업·버전 관리·사용자 관리·blob 저장소를 CRDT(Conflict-free Replicated Data Type, 충돌 없는 복제 데이터 타입) 기반 갱신으로 지원" 하는 구조입니다 [S10].

기술 스택의 의미를 풀어드리겠습니다. GraphQL 채널은 워크스페이스의 문서 트리·데이터베이스·메타데이터를 정밀하게 쿼리할 수 있게 해줍니다. WebSocket 은 변경 이벤트의 실시간 구독을 가능하게 합니다. CRDT 는 동시 편집 시 충돌 없이 상태를 병합하므로, AI 에이전트가 사람과 동시에 같은 문서를 수정할 때 일관성을 보장합니다.

이 구조는 Claude 같은 LLM 에이전트가 단순 RAG 소스 이상의 역할을 수행하실 수 있게 해줍니다. 에이전트가 문서를 읽고 요약할 뿐 아니라, 그 결과를 다시 워크스페이스에 새 문서로 작성하거나 데이터베이스 행으로 입력하는 양방향 흐름이 가능합니다. WebSocket 구독을 통해 사용자의 갱신을 실시간으로 감지하고 대응하는 에이전트 설계도 열립니다.

엔지니어 여러분이 도입을 검토하실 때 점검하실 부분은 AFFiNE 본체 버전과 MCP 서버 버전의 호환 범위, 그리고 GraphQL 스키마 변경에 대한 어댑터의 추종 속도입니다. CRDT 기반 시스템은 갱신 모델이 일반 CRUD 와 다르기 때문에, AI 에이전트의 쓰기 작업이 의도치 않은 병합 결과를 만들지 않도록 테스트 시나리오를 충분히 마련하셔야 합니다.

4.3.2 v1.6.0 의 데이터베이스 편집 도구와 활성도 진단

affine-mcp-server 의 v1.6.0 버전은 기능 확장이 의미 있게 진행된 분기점입니다. 저장소 변경 내역에 따르면 v1.6.0 은 "태그 워크플로우, 마크다운 import/export/replace 워크플로우, add_database_column·add_database_row 같은 직접적인 데이터베이스 편집 도구를 추가했으며, 종단 검증 커버리지를 포함" 한다고 명시되어 있습니다 [S10].

데이터베이스 편집 도구의 추가는 AI 에이전트 자동화 시나리오의 폭을 넓힙니다. add_database_column 은 스키마 변경을, add_database_row 는 인스턴스 추가를 가능하게 합니다. 마크다운 import/export/replace 는 외부 소스를 워크스페이스로 끌어들이거나 워크스페이스 내용을 다른 시스템으로 내보내는 작업을 표준화합니다. 종단 검증 커버리지는 자동 테스트가 도구 사용 시나리오 전반을 검증한다는 뜻으로, 운영 신뢰도에 기여합니다.

활성도 지표도 함께 보겠습니다. 2026년 6월 시점 affine-mcp-server 의 깃허브 Star 수는 171개로 집계됩니다 [S10]. 절대 수치는 크지 않지만 서드파티 MCP 서버 카테고리 안에서는 의미 있는 수준이며, 릴리스 빈도와 기능 확장 속도가 함께 유지되고 있다는 점이 더 중요합니다. 단일 메인테이너 의존 위험은 여전히 존재하지만, 종단 검증 커버리지가 도입되었다는 점은 코드 품질 관리에 신경을 쓰고 있음을 보여줍니다.

다만 의사결정권자께서 잊지 말아야 하실 점은 이 서버가 공식 AFFiNE 팀의 산물이 아니라는 사실입니다. AFFiNE 본체가 GraphQL 스키마를 비호환 방식으로 변경하면 어댑터가 일시적으로 작동하지 않을 가능성이 있습니다. 운영 환경에서는 백업 경로(GraphQL 직접 호출 또는 export 기반 RAG)를 함께 준비해 두시기를 권합니다.

종합하면 AFFiNE 는 공식 MCP 부재라는 약점이 있지만, 서드파티 서버 한 개의 기술 구조와 기능 확장 속도가 AppFlowy 의 커뮤니티 서버보다 한 단계 높은 위치에 있습니다.

4.4 Outline 공식 MCP·REST API 의 RAG 소스화 효율

Outline 은 위키 구조의 협업 도구이자 셀프호스팅이 가능한 오픈소스 프로젝트입니다. 세 도구 가운데 Claude 연동 성숙도가 가장 높은 위치에 있으며, 그 근거는 docs.getoutline.com 의 공식 안내와 REST API 의 부분 검색 효율에서 찾을 수 있습니다.

4.4.1 getoutline.com 공식 MCP 안내와 Streamable HTTP 전송 방식

Outline 의 공식 도메인인 docs.getoutline.com 은 MCP 통합을 별도 항목으로 안내합니다. 공식 문서는 "MCP 는 AI 어시스턴트가 Outline 워크스페이스의 문서를 직접 검색·읽기·생성·편집하는 것을 가능하게 한다. Outline 은 Streamable HTTP transport 만 지원한다" 라고 명시합니다 [S9].

이 안내가 갖는 의미를 풀어보겠습니다. 첫째, MCP 가 부가 기능이 아니라 공식 통합 채널로 위치한다는 점이 명확합니다. 둘째, 지원 도구가 검색·읽기·생성·편집 네 가지 영역을 모두 포함하므로 양방향 에이전트 시나리오가 가능합니다. 셋째, 전송 방식이 Streamable HTTP 로 통일되어 있다는 점은 운영자가 단일 프로토콜 가정으로 게이트웨이와 인증을 설계할 수 있게 해줍니다.

Streamable HTTP 는 단일 HTTP 엔드포인트 위에서 양방향 스트리밍을 지원하는 전송 방식으로, MCP 1.x 의 권장 트랜스포트입니다. 기존 SSE(Server-Sent Events) 기반 트랜스포트가 갖던 연결 관리 복잡도를 줄여주며, 표준 리버스 프록시·로드밸런서 뒤에서도 안정적으로 동작합니다. Outline 이 이 트랜스포트만 지원한다는 결정은 운영 표준화 측면에서 도움이 됩니다.

여러분이 셀프호스팅 환경에서 이 채널을 활용하시려면 인증 토큰 발급, 워크스페이스 권한 매핑, 그리고 게이트웨이에서의 HTTP 스트리밍 허용 설정을 준비하시면 됩니다. 세부 구성은 본 백서 5장의 운영 가이드에서 다룹니다.

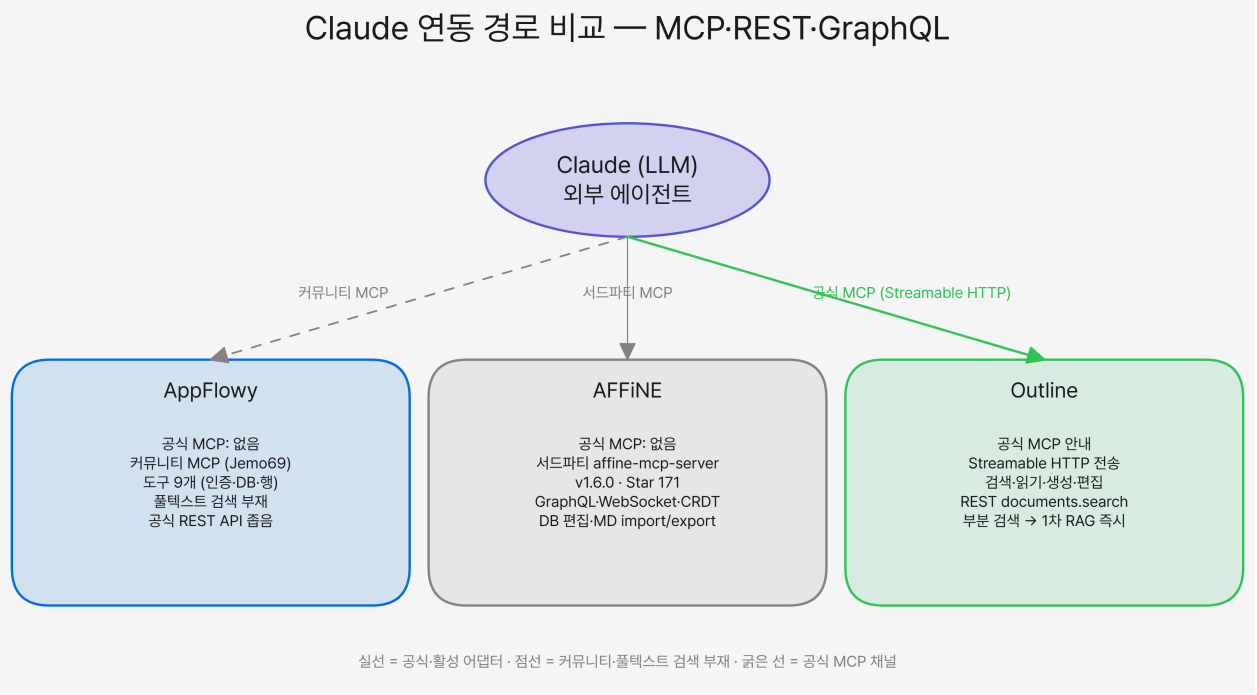

그림 4-1. AppFlowy·AFFiNE·Outline 세 도구의 Claude 연동 경로 비교. Outline 은 공식 MCP 채널 (Streamable HTTP)과 REST API documents.search 가 양방향 경로를 형성합니다. AFFiNE 는 서드파티

affine-mcp-server 가 GraphQL·WebSocket 위에 MCP 표준 인터페이스를 얹습니다. AppFlowy 는 커뮤니티 MCP 서버 1개가 데이터베이스 행 조작 중심으로 작동하며 풀텍스트 검색 경로가 좁습니다. 본 도해의 의도는 동일한 셀프호스팅 카테고리 안에서도 LLM 통로의 폭과 공식성이 크게 다르다는 점을 한눈에 보여드리는 것입니다.

4.4.2 REST API documents.search 의 부분 검색·RAG 소스화 효율

Outline 의 REST API 는 MCP 와 별개로 RAG 파이프라인 구축에 활용하실 수 있는 강력한 통로입니다. 특히 documents.search 엔드포인트는 키워드 매칭만이 아니라 본문의 일부 발췌를 반환하므로, 임베딩 색인을 별도로 구축하시지 않더라도 1차 RAG 소스로 곧바로 활용이 가능합니다 [S9].

부분 검색 능력의 가치는 두 가지로 정리됩니다. 첫째, Claude 가 검색 결과를 컨텍스트 윈도우에 직접 넣을 수 있어 별도의 청크 분할 로직을 줄일 수 있습니다. 둘째, 검색 결과에 문서 ID·경로·갱신 시각 같은 메타데이터가 함께 포함되어 인용 출처 명시가 자연스럽게 이루어집니다. 두 효과는 RAG 응답 품질을 끌어올리는 핵심 요소입니다.

또한 Outline 의 REST API 는 컬렉션·문서·권한·구독자를 모두 다루는 폭넓은 엔드포인트 집합을 제공합니다. 이는 MCP 채널 외에도 운영자가 자체적으로 통합 자동화를 구성하실 수 있는 여지를 넓혀줍니다. 일정 주기로 변경 문서를 폴링하여 임베딩 데이터베이스를 갱신하는 파이프라인을 가볍게 구현하실 수 있습니다.

엔지니어 여러분이 점검하셔야 할 사항은 검색 응답의 토큰 비용, 권한 필터링의 정확성, 그리고 Rate Limit 운영 정책입니다. 사내 RAG 시스템이 documents.search 를 빈번하게 호출하실 경우 캐싱 레이어를 두어 응답 시간과 비용을 안정화시키시기 바랍니다.

4.5 세 도구 AI 연동 종합 비교

지금까지 살펴본 세 도구의 AI 연동 가능성을 한 화면에 정리해드립니다. 의사결정권자께서 한눈에 위치를 파악하시고 도입 시나리오에 맞춰 선택하실 수 있도록 종합 비교와 권고를 함께 제시합니다.

4.5.1 공식 MCP·서드파티 MCP·REST/GraphQL 종합 매트릭스

세 도구를 공식 MCP 지원, 서드파티 MCP 서버 활성도, REST/GraphQL 검색 효율의 세 가지 기준으로 비교해드리겠습니다. 첫째, 공식 MCP 지원 여부에서 Outline 만 docs.getoutline.com 의 공식 안내가 존재합니다 [S9]. AppFlowy 와 AFFiNE 은 공식 MCP 서버가 존재하지 않습니다 [S10][S11].

둘째, 서드파티 MCP 서버 활성도에서 AFFiNE 의 affine-mcp-server 가 v1.6.0 까지 기능을 확장하며 종단 검증 커버리지를 갖춘 상태입니다 [S10]. AppFlowy 의 커뮤니티 서버는 도구 9개의 비교적 작은 지원 범위을 갖고 있으며 풀텍스트 검색이 없는 점이 제약입니다 [S11]. Outline 은 공식 MCP 가 있으므로 서드파티 의존도가 낮습니다.

셋째, REST/GraphQL 검색 효율에서 Outline 의 documents.search 가 부분 검색을 지원하여 RAG 친화도가 가장 높습니다 [S9]. AFFiNE 은 GraphQL 의 풍부한 쿼리 표현력이 강점이지만 검색 자체는 본체 기능에 의존합니다. AppFlowy 는 외부 RAG 용도로 활용하기 위한 검색 API 공개 범위가 가장 좁습니다.

이 세 기준을 종합하면 Outline → AFFiNE → AppFlowy 순으로 Claude 연동 성숙도가 형성됩니다. 다만 이 순서가 도구 전반의 우열을 의미하지는 않습니다. UI·오프라인 지원·데이터 구조 표현력 같은 다른 평가 기준에서는 순위가 달라질 수 있으며, 본 백서의 다른 장이 이를 다룹니다.

4.5.2 도입 시나리오별 AI 연동 권고

여러분의 도입 시나리오에 맞춰 세 가지 권고를 제시합니다. 첫 번째 시나리오는 사내 문서 검색 RAG 입니다. 사내 위키를 Claude 에 연결해 직원 질문에 답하는 챗봇을 구축하시는 경우라면, Outline 이 가장 빠른 도입 경로를 제공합니다. 공식 MCP 와 documents.search 의 조합이 별도 색인 파이프라인 구축 없이 1차 RAG 를 가능하게 해드립니다.

두 번째 시나리오는 에이전트 자동화입니다. Claude 에이전트가 문서 작성, 데이터베이스 갱신, 회의록 정리 같은 양방향 작업을 수행하시는 환경에서는 AFFiNE 의 affine-mcp-server v1.6.0 이 적합한 위치에 있습니다 [S10]. 데이터베이스 편집 도구와 마크다운 워크플로우, CRDT 기반 동시 편집 안정성이 결합되어 에이전트의 쓰기 작업을 견고하게 처리합니다. Outline 도 동일한 시나리오를 처리할 수 있으며, 공식 채널 안정성이 강점입니다.

세 번째 시나리오는 다중 도구 연결입니다. 사내에 이미 AppFlowy 가 도입되어 있고 추후 Claude 연동을 확장하시려는 경우라면, Jemo69 의 커뮤니티 MCP 서버를 활용해 데이터베이스 자동화부터 시작하시고, 풀텍스트 검색이 필요한 영역은 별도 임베딩 파이프라인을 구축하시는 단계적 접근을 권합니다 [S11]. 다만 운영 책임 소재와 본체 API 변경 추종 위험을 사전에 평가하시기 바랍니다.

마지막으로 의사결정권자께서 기억하셔야 할 한 가지는 AI 연동 가능성이 도구 선정의 유일한 기준은 아니지만, 2026년 현재 가장 빠르게 변화하는 평가 기준이라는 점입니다. 본 장에서 정리한 세 도구의 위치는 6개월 후에 바뀔 수 있으며, 도입 후에도 정기적으로 재평가하시기를 권합니다. 본 백서의 다음 장은 데이터 주권과 보안 측면에서 세 도구를 비교하여, AI 연동 가능성과 함께 두 축이 어떻게 균형을 이루는지 살펴봅니다.

5장: Markdown·Obsidian 기존 자산 연속성 확보 경로

여러분의 조직이 Notion 대체를 검토할 때 가장 먼저 떠올리는 질문은 "지금까지 쌓아둔 Markdown 문서와 Obsidian vault 를 어떻게 옮기느냐" 일 것입니다. 협업 도구 전환에서 가장 많은 비용을 발생시키는 항목은 라이선스도, 인프라도 아닌 기존 지식 자산의 이전과 연속성 확보이기 때문입니다. 수년에 걸쳐 작성한 회의록·기획서·기술 노트가 새로운 도구에서 그대로 열리지 않거나 일부 메타데이터를 잃는다면, 전환 의사결정 자체가 지연되거나 무산됩니다.

세 도구는 모두 "Markdown 지원" 을 표방합니다. 그러나 동일한 단어 뒤에 숨은 구현은 매우 다릅니다. Outline 은 Markdown 을 1차 저장 포맷으로 사용하지만 [S7], AppFlowy 는 자체 데이터 모델(Database) 에 콘텐츠를 저장하고 Markdown 은 가져오기·내보내기 통로로만 활용합니다 [S1]. AFFiNE 은 페이지 모드의 텍스트 블록은 Markdown 으로 평탄화 가능하지만 edgeless 캔버스의 그래픽 블록은 Markdown 표현이 존재하지 않아 손실이 발생합니다 [S4]. 같은 "Markdown 지원" 이라는 표현 뒤에 보존 범위·동기화 방향·운영 부담이 완전히 다른 세 모델이 자리하고 있습니다.

이 장은 세 도구의 Markdown 처리 방식 차이를 본질에서부터 풀어낸 뒤, 각 도구별 가져오기 손실 항목·Obsidian 연동 가능성·실무에서 권고할 운영 분리 패턴을 차례로 살펴봅니다. 의사결정 담당자께서는 이 장을 통해 "우리 조직의 기존 자산을 잃지 않고 옮길 수 있는 도구가 어느 쪽인가" 라는 질문에 구체적인 근거로 답하실 수 있습니다.

또한 본 장은 단순한 가져오기 호환성 비교를 넘어, 도구 선택 이후의 장기 운영을 고려한 백업·동기화·외부 도구 연계 시나리오까지 함께 다룹니다. 단방향 가져오기만 가능한 도구와 양방향 동기화가 가능한 도구는 도입 후 3

년 시점의 데이터 이동성에서 큰 격차를 만들기 때문입니다.

5.1 세 도구의 Markdown 처리 방식 차이

세 도구의 Markdown 지원을 같은 층위에서 비교하려면 먼저 "Markdown 이 1차 저장 포맷인가, 아니면 변환 통로인가" 를 구분해야 합니다. 이 구분이 가져오기 손실 항목·양방향 동기화 가능 여부·백업 운영 부담을 모두 결정하기 때문입니다.

5.1.1 파일 기반 저장과 데이터베이스 기반 저장의 본질적 차이

Outline 은 문서의 원본을 Markdown 텍스트로 저장하는 파일 기반 모델을 채택합니다 [S7]. 사용자가 에디터에서 작성한 콘텐츠는 PostgreSQL 에 Markdown 문자열로 그대로 저장되며, 화면 표시 시점에 클라이언트가 렌더링합니다. 이 구조의 장점은 명확합니다. 데이터베이스 백업을 받든, REST API 로 documents.export 를 호출하든, 결과물이 항상 표준 Markdown 이므로 사용자가 직접 열어보거나 다른 도구에 붙여 넣어도 정보 손실이 사실상 0 입니다. 또한 git 저장소 동기화나 Obsidian vault 와의 양방향 연계를 구상할 때 변환 단계가 없어 운영 부담이 크게 줄어듭니다.

AppFlowy 는 정반대 접근을 취합니다. 모든 콘텐츠가 자체 데이터베이스 스키마(Database, Grid, Board, Calendar) 위에 구조화 객체로 저장되며, Markdown 은 가져오기와 내보내기 단계에서만 사용되는 변환 포맷입니다 [S1]. 즉, 사용자가 Markdown 파일을 가져오면 AppFlowy 가 이를 파싱하여 내부 블록 모델로 변환하고, 내보내기 시 다시 Markdown 으로 직렬화합니다. 이 과정에서 Markdown 표준에 1:1 로 대응되지 않는 모든 메타데이터, 예를 들어 Board(Kanban) 의 컬럼 정의나 Grid 의 관계 필드(relation field), 수식(formula) 필드는 손실되거나 단순 텍스트로 평탄화될 수 있습니다.

AFFiNE 은 페이지 모드와 edgeless 모드라는 이중 구조를 갖습니다. 페이지 모드의 텍스트·표·리스트는 Markdown 으로 가져오기·내보내기가 가능하며, 최근 MCP v1.6.0 에서 "markdown import/export/replace workflows" 를 정식으로 추가했습니다 [S10]. 그러나 edgeless 캔버스에 배치된 도형·프레임·연결선·자유 위치 블록은 Markdown 명세에 존재하지 않으므로 평탄화 시 손실이 발생합니다 [S4]. AFFiNE 은 결국 "텍스트 영역은 Markdown 친화, 그래픽 영역은 자체 포맷" 이라는 하이브리드 모델로 정리됩니다.

이 본질적 차이는 도구 선택 단계에서 반드시 짚어야 할 첫 번째 분기점입니다. 여러분의 팀이 기존에 Obsidian vault 나 git 저장소에 Markdown 자산을 축적해 왔다면, Outline 의 파일 기반 저장은 가장 자연스러운 이행 경로를 제공합니다. 반면 Notion 처럼 Database·Kanban·Calendar 같은 구조화 뷰를 적극 활용해 왔다면, AppFlowy 의 DB 우선 모델이 더 잘 맞지만 그 대가로 Markdown 왕복 시 손실을 감수해야 합니다.

5.1.2 양방향 동기화 가능 여부와 일방향 가져오기의 차이

Markdown 처리 방식의 두 번째 분기점은 양방향 동기화 가능 여부입니다. "가져오기는 되는데 내보내기는 안 된다" 또는 "내보내기는 되는데 외부 편집 결과를 다시 반영할 수 없다" 같은 일방향 제약은 도입 초기에는 잘 드러나지 않지만 3년 시점에 데이터 이동성을 크게 떨어뜨립니다.

Outline 은 documents 엔드포인트를 통해 생성·조회·수정·삭제(CRUD) 가 모두 REST API 로 가능하며, documents.search 같은 부분 검색 엔드포인트도 제공합니다 [S9]. 따라서 외부 git 저장소·Obsidian vault·CI 파이프라인에서 변경된 Markdown 을 documents.update 로 다시 반영하는 양방향 동기화 스크립트를 구성할 수 있습니다. 이 점이 Outline 을 "운영팀이 직접 자동화 가능한" 도구로 만들어 줍니다.

AppFlowy 는 현시점에서 Markdown 가져오기·내보내기를 UI 메뉴와 일부 CLI 경로로 제공하지만, 외부 Markdown 파일의 변경을 다시 가져와 동일한 페이지에 반영하는 양방향 동기화는 공식 워크플로우로 정립되어 있지 않습니다 [S1]. 즉, 한 번 AppFlowy 내부 DB 로 옮긴 콘텐츠는 그 뒤로는 AppFlowy 안에서만 편집하는 것이 현실적이며, 외부 Markdown 자산과 지속적으로 동기화하려는 시나리오는 별도의 사용자 정의 스크립트와 충돌 해결 로직이 필요합니다.

AFFiNE 은 MCP v1.6.0 에서 markdown import/export/replace workflows 를 추가하여 양방향성에 한 걸음 다가갔습니다 [S10]. 특히 replace 워크플로우는 기존 페이지 콘텐츠를 새로운 Markdown 으로 교체하는 의미이므로, 외부 도구에서 편집한 결과를 다시 AFFiNE 으로 밀어넣는 패턴을 지원합니다. 다만 edgeless 캔버스 블록은 이 워크플로우의 대상이 아니며, 텍스트 영역에 한정된다는 점을 운영 매뉴얼에 명시해야 합니다.

여러분이 이 절을 통해 얻어야 할 결론은 다음과 같습니다. "Markdown 지원" 이라는 단일 체크박스로 도구를 평가하면 전환 후 6개월~3년 시점에 데이터 이동성·백업·외부 연계 비용 측면에서 큰 차이가 드러납니다. 가져오기 방향·내보내기 방향·외부 편집 반영 방향이라는 세 가지 흐름을 각각 검증해야 의사결정의 정확도가 올라갑니다.

5.2 AppFlowy의 DB 저장 방식 제약과 Markdown 가져오기 경로

AppFlowy 가 매력적인 이유는 Notion 의 Database·Board·Calendar 뷰를 가장 충실히 재현한 오픈소스 대안이라는 점입니다. 그러나 이 강점이 그대로 Markdown 호환성의 제약으로 이어집니다. 데이터 모델이 풍부할수록 표준 Markdown 으로 표현할 수 없는 메타데이터가 늘어나기 때문입니다.

5.2.1 AppFlowy Markdown 가져오기에서 보존되는 항목과 잃는 항목

AppFlowy 의 Markdown 가져오기는 텍스트 블록 중심으로 동작합니다 [S1]. 헤더(H1~H6), 단락, 리스트(순서·비순서), 인용, 코드 블록, 인라인 코드, 강조(굵게·기울임), 링크, 이미지 같은 CommonMark 표준 블록은 안정적으로 보존됩니다. GitHub Flavored Markdown 의 표(table) 도 가져오기 시 AppFlowy 의 표 블록으로 변환되어 행·열 구조가 유지됩니다.

문제는 Markdown 표준을 벗어나는 영역입니다. 첫째, Obsidian 의 위키 링크(`[[페이지명]]`) 는 AppFlowy 가 자체 페이지 참조로 자동 변환하지 못하므로 일반 텍스트로 들어가거나 사용자가 가져오기 후 수동으로 다시 연결해야 합니다. 둘째, frontmatter YAML 블록은 가져오기 시 페이지 속성으로 자동 매핑되지 않고 본문 상단의 코드 블록으로 남는 경우가 일반적입니다. 셋째, Board(Kanban) 의 컬럼 정의·Grid 의 관계 필드·Calendar 의 날짜 매핑 같은 데이터 모델 메타정보는 Markdown 명세에 존재하지 않으므로 가져오기 단계에서 표현할 방법 자체가 없습니다 [S1].

여러분이 AppFlowy 가져오기를 검토할 때 가장 먼저 실측해야 할 항목은 다음과 같습니다. 첫째, 기존 Obsidian vault 의 위키 링크가 본문 텍스트로 평탄화된 후 사용자가 수동 재연결할 수 있는 분량인지. 둘째, 기존 Notion export 의 Board·Database 메타데이터가 단순 표로 평탄화되었을 때 업무 흐름이 유지되는지. 셋째, 이미지·첨부 파일이 절대 경로/상대 경로/내장 base64 중 어느 형태로 들어가는지. 이 세 항목은 모두 도구의 공식 문서가 아니라 실제 가져오기 파일럿으로만 답을 얻을 수 있습니다.

내보내기 측에서는 역방향 손실도 존재합니다. AppFlowy 의 Board 컬럼·관계 필드·수식 필드를 Markdown 으로 내보내면 단순 표 또는 텍스트로 평탄화되며, 이를 다시 다른 도구로 가져오면 원래의 동작이 복원되지 않

습니다 [S1]. 따라서 AppFlowy 를 도입한 뒤 3년 시점에 다른 도구로 이행해야 한다면, Markdown 내보내기만으로는 충분치 않고 AppFlowy 의 자체 백업 포맷(JSON·workspace export)을 함께 보관해야 합니다.

5.2.2 Obsidian vault 직접 연동 시도와 커뮤니티 플러그인 현황

AppFlowy 와 Obsidian vault 를 직접 연동하는 공식 경로는 현재 제공되지 않습니다. 두 도구의 저장 모델이 근본적으로 다르기 때문입니다. Obsidian 은 로컬 파일시스템의 `.md` 파일을 직접 편집하는 모델인 반면, AppFlowy 는 자체 DB(로컬 SQLite 또는 클라우드 PostgreSQL) 에 블록 단위로 저장합니다 [S1]. 양자를 실시간으로 동기화하려면 파일 변경 감지·블록 모델 변환·충돌 해결 로직이 모두 필요한데, 이는 AppFlowy 공식 로드맵에 포함되어 있지 않습니다.

커뮤니티 차원에서는 Obsidian vault 의 Markdown 파일을 일괄 가져오기하는 스크립트성 접근이 보고되고 있습니다. 다만 이는 일회성 마이그레이션을 목표로 한 일방향 가져오기에 가깝습니다. Obsidian vault 에서 편집한 결과를 다시 AppFlowy 페이지에 반영하거나, AppFlowy 에서 편집한 결과를 vault 파일로 내보내 git 저장소에 커밋하는 자동화는 사용자가 직접 구축해야 합니다.

여러분의 조직이 Obsidian 을 개인 노트 도구로 계속 활용하면서 AppFlowy 를 팀 협업 도구로 도입하려 한다면, 두 도구의 역할 분담을 명확히 설계하는 것이 현실적입니다. 예를 들어, 개인 연구 노트·일지·임시 메모는 Obsidian vault 에서 작성·축적하고, 팀에 공유하기로 결정된 시점에 AppFlowy 로 일방향 가져오기하여 협업하는 모델입니다. 반대 방향(AppFlowy → Obsidian) 은 자동화하지 않고, 필요한 시점에만 Markdown 내보내기를 수행해 vault 에 수동 복사하는 정도가 안정적입니다.

운영 측면에서 추가로 고려할 항목은 백업 전략입니다. AppFlowy 셀프호스팅 환경에서는 PostgreSQL 덤프와 함께 사용자별 workspace export(JSON 포맷) 를 주기적으로 보관할 것을 권고합니다. Markdown 내보내기만으로는 Board·Grid·Calendar 메타정보가 복원되지 않기 때문입니다 [S1]. 이 점은 도입 결정 단계에서 운영 조직이 함께 검토해야 할 운영 비용 항목입니다.

5.3 AFFiNE edgeless 손실과 현실적 대응

AFFiNE 은 페이지 모드와 edgeless 모드를 단일 문서 안에서 자유롭게 전환할 수 있다는 차별점을 강조합니다. 같은 문서를 위키 페이지로도, 무한 캔버스의 화이트보드로도 사용할 수 있다는 의미입니다. 그러나 이 이중 모드 구조가 Markdown 호환성 측면에서는 가장 까다로운 손실 지점을 만들어 냅니다.

5.3.1 edgeless 캔버스 블록의 Markdown 평탄화 손실 범위

AFFiNE 의 edgeless 캔버스에는 텍스트 블록 외에도 자유 위치 도형(Shape: 사각형·원·삼각형·화살표), 프레임(Frame: 그룹화 영역), 연결선(Connector: 도형 간 화살표), 자유 곡선(brush), 자유 위치 이미지·임베드 같은 그래픽 객체가 배치됩니다 [S4]. 이들은 캔버스의 (x, y) 좌표·크기·회전·색상·연결 관계 같은 시각적 메타데이터를 갖는데, Markdown 명세에는 이를 표현할 구문이 존재하지 않습니다.

따라서 edgeless 캔버스를 Markdown 으로 내보내면 다음과 같은 손실이 발생합니다. 첫째, 도형·프레임·연결선은 Markdown 출력에서 누락되거나 텍스트로 평탄화됩니다. 둘째, 자유 위치 텍스트 블록은 위치 정보를 잃고 단순 단락으로 순차 나열됩니다. 셋째, 도형 간 연결선이 표현하던 다이어그램의 논리(예: 아키텍처 도식, 흐름도) 가 텍스트만으로는 복원 불가능해집니다. 결과적으로 edgeless 에서 작업한 시각적 산출물은 Markdown 왕복 시 실질적으로 소실됩니다.

AFFiNE MCP v1.6.0 이 도입한 markdown import/export/replace workflows 는 페이지 모드의 텍스트 블록에 한정되며, edgeless 캔버스의 그래픽 객체는 대상이 아닙니다 [S10]. 즉, AFFiNE 의 Markdown 워크플로우 강화는 페이지 모드 사용자에게는 명확한 진전이지만, edgeless 중심 사용자에게는 직접적인 이득이 적습니다.

여러분의 팀이 AFFiNE 을 도입할 때, edgeless 캔버스의 산출물을 장기 보존하려면 Markdown 이 아닌 다른 경로를 함께 마련해야 합니다. 첫째, AFFiNE 자체 workspace 백업 포맷(BlockSuite snapshot 등) 을 주기적으로 보관하는 방법이 가장 안전합니다. 둘째, 중요한 다이어그램은 PNG/SVG 이미지로 별도 내보내기하여 Markdown 페이지에 첨부하는 방식으로 시각 정보를 어느 정도 보존할 수 있습니다. 다만 이 경우 이미지로 변환된 다이어그램은 더 이상 편집 가능한 객체가 아니라는 점을 사용자에게 안내해야 합니다.

5.3.2 페이지 모드 중심 운영과 edgeless 사용 분리 권고

위의 손실 구조를 고려할 때, 셀프호스팅 운영에서는 페이지 모드와 edgeless 모드의 용도를 명시적으로 분리하는 운영 정책을 권고합니다. 페이지 모드는 위키·기획서·회의록·기술 문서처럼 장기 보존이 필요한 텍스트 콘텐츠에, edgeless 모드는 일회성 브레인스토밍·임시 다이어그램·워크숍 산출물처럼 단기 시각화에 사용하는 분리 모델입니다.

이 분리 원칙을 적용하면 다음과 같은 이점이 생깁니다. 첫째, 장기 보존이 필요한 콘텐츠는 모두 페이지 모드에 있으므로 Markdown 내보내기·외부 백업·다른 도구로의 이행이 모두 안전해집니다. 둘째, edgeless 캔버스에서 생성된 다이어그램이 "정식 문서" 가 되어야 할 때는 명시적인 변환 절차(중요 도형을 PNG/SVG 로 내보내기 → 페이지 모드 문서에 첨부) 를 거치므로 손실 위험이 통제됩니다. 셋째, 사용자 교육 시 "어떤 콘텐츠를 어디에 작성해야 하는가" 라는 의사결정 부담이 줄어듭니다.

운영 가이드 측면에서는 다음 항목을 사용자 매뉴얼에 명시할 것을 권고합니다. (1) edgeless 캔버스는 1회성 시각화 용도로만 사용한다. (2) 정식 문서화가 필요한 다이어그램은 PNG/SVG 로 내보낸 후 페이지 모드 문서에 첨부한다. (3) AFFiNE workspace 의 정기 백업은 BlockSuite snapshot 단위로 수행한다. (4) 페이지 모드 콘텐츠는 MCP v1.6.0 의 markdown export workflow 를 통해 별도 git 저장소에 주기 동기화한다 [S10].

여러분의 의사결정 입장에서, AFFiNE 은 "edgeless 캔버스의 시각적 자유도" 라는 강점과 "그 시각적 산출물의 Markdown 비호환" 이라는 약점을 동시에 갖는 도구입니다. 두 특성을 모두 인정하고 운영 정책으로 손실 위험을 통제하는 접근이, 도입 후 1~2년 시점에 발생할 수 있는 데이터 이동성 문제를 줄여 줍니다.

5.4 Outline Markdown 네이티브와 REST 동기화

세 도구 중 Markdown 자산의 연속성 확보 측면에서 가장 직접적인 경로를 제공하는 쪽은 Outline 입니다. Markdown 이 1차 저장 포맷이라는 단순한 사실이 백업·이행·외부 도구 연계의 모든 단계에서 누적된 이점을 만들어 냅니다.

5.4.1 Markdown 1차 저장의 운영 안정성과 백업 용이성

Outline 은 문서 콘텐츠를 Markdown 텍스트로 PostgreSQL 에 저장합니다 [S7]. 이 구조는 운영팀 관점에서 세 가지 직접적인 이점을 줍니다.

첫째, 백업의 가독성입니다. 데이터베이스 덤프를 풀어 보지 않더라도 documents.export REST API 호출만으로 모든 문서를 표준 Markdown 파일로 받아낼 수 있고, 그 결과물은 어떤 텍스트 에디터에서도 그대로 열립니다. 셀프호스팅 환경에서 디스크 장애·실수 삭제·도구 자체 이탈 같은 사고가 발생해도, 백업된 Markdown 파일 자체가 곧 사용 가능한 자산입니다. 다른 도구로의 이행이 필요한 시점에도 변환 손실 없이 그대로 가져오기에 사용할 수 있습니다.

둘째, 외부 도구와의 자연스러운 호환성입니다. git 저장소에 commit 하든, Obsidian vault 디렉터리에 배치하든, MkDocs·Hugo 같은 정적 사이트 생성기에 입력으로 넣든, Outline 의 내보내기 결과는 별도 변환 없이 동작합니다. 이는 조직이 Outline 을 협업 도구로 사용하면서 동시에 별도의 공개 문서 사이트나 내부 위키 미러를 유지하려 할 때 큰 운영 이점이 됩니다.

셋째, 검색·인덱싱·LLM 활용의 단순함입니다. Outline 의 documents.search 엔드포인트는 부분 검색을 지원하며 [S9], 동일한 Markdown 콘텐츠를 외부 검색 엔진이나 LLM RAG 파이프라인에 그대로 투입할 수 있습니다. AppFlowy·AFFiNE 처럼 자체 블록 모델을 Markdown 으로 변환하는 단계가 없으므로 인덱싱 파이프라인이 짧고 안정적입니다.

여러분의 조직이 "협업 도구이지만 동시에 정적 사이트 생성기·LLM RAG·git 기반 문서 워크플로우와 연계해야 한다" 는 요구 사항을 갖고 있다면, Outline 의 Markdown 1차 저장 모델은 그 자체로 큰 운영 가치를 제공합니다. 도구 선택 단계에서 "백업이 곧 사용 가능한 자산인가" 라는 질문에 가장 명확히 답하는 쪽이 Outline 입니다.

5.4.2 REST API documents 엔드포인트 기반 양방향 동기화 패턴

Outline 의 두 번째 강점은 documents 엔드포인트가 제공하는 양방향 동기화 가능성입니다. documents.create·documents.update·documents.delete·documents.export·documents.search 같은 엔드포인트가 REST API 로 모두 노출되어 있어 [S9], 외부 시스템과의 동기화 스크립트를 비교적 단순하게 구성할 수 있습니다.

대표적인 패턴은 다음 세 가지입니다.

첫째, git 저장소 ↔ Outline 양방향 동기화. CI 파이프라인이 git 저장소의 `.md` 파일 변경을 감지하면 documents.update 로 Outline 의 동일 문서를 갱신하고, 반대로 Outline 에서 발생한 변경(웹훅 또는 폴링)을 git 저장소에 commit·push 합니다. 충돌 해결은 git 의 표준 merge 전략이나 단순한 "마지막 쓴 쪽이 이긴다" 규칙으로 처리할 수 있습니다.

둘째, Obsidian vault ↔ Outline 일방향 발행. 개인 vault 의 특정 디렉터리(예: `published/`) 에 배치된 노트만 주기적으로 Outline 에 발행하는 모델입니다. documents.create·documents.update 만 사용하므로 충돌 해결이 단순하며, 개인 작성 단계는 vault 에서, 팀 공유 단계는 Outline 에서 수행하는 역할 분담이 명확합니다.

셋째, 검색·요약·자동 분류 같은 LLM 자동화 연계. documents.search 로 관련 문서를 검색하고 [S9], documents.export 로 본문을 가져와 LLM 에 투입한 뒤, 생성된 요약이나 태그를 documents.update 로 다시 반영합니다. 이 패턴은 협업 도구를 단순한 콘텐츠 저장소가 아니라 자동화 가능한 지식 베이스로 확장합니다.

여러분이 이 절에서 가장 주목해야 할 점은, 위의 세 가지 패턴이 모두 Outline 의 공식 REST API 표면에서 직접 구현 가능하며 별도의 자체 변환 레이어가 필요하지 않다는 사실입니다. AppFlowy 의 경우 양방향 동기화에는 사용자 정의 변환·충돌 해결 로직이 필요하고, AFFiNE 의 경우 MCP v1.6.0 의 markdown workflows 가 페이지 모드에 한정된다는 제약이 있습니다 [S10]. Outline 의 documents 엔드포인트는 Markdown 1차 저장이라는 본질적 단순함과 결합하여, 도입 후 자동화·외부 연계 비용을 가장 낮게 유지하는 경로를 제공합니다.

결론적으로, 기존 Markdown·Obsidian 자산의 연속성을 최우선으로 두는 조직이라면 Outline 이 가장 안전한 1차 선택지입니다. Notion 의 Database·Board 유사 경험이 핵심 요구라면 AppFlowy 를, 페이지·캔버스 혼용 환경이 필요하다면 AFFiNE 을 검토하되, 양자 모두 백업 전략에 Markdown 외 자체 포맷을 반드시 포함시켜야 한다는 점을 의사결정 문서에 명시해 두실 것을 권고합니다 [S1] [S4] [S7].

6장: Kanban·실시간 협업·지식 구조화 실무 기능 비교

여러분이 Notion 대체 도구를 검토할 때 가장 먼저 부딪히는 벽은 "기능 목록이 비슷해 보인다"는 점입니다. 세 도구 모두 문서·데이터베이스·협업이라는 큰 그림은 공통으로 갖추고 있지만, 막상 팀에 도입하여 실무를 굴려보면 완성도 차이가 명확하게 드러납니다. 특히 칸반 보드, 실시간 협업, 지식 구조화 세 영역은 도구 선정의 갈림길이 됩니다. 이 장에서는 세 도구가 어떤 지점에서 강점과 빈틈을 보이는지 실무 기능 단위로 풀어 보겠습니다.

도입 검토 회의에서 자주 듣는 질문은 "우리 팀이 쓰던 Notion 의 보드 뷰가 그대로 옮겨질까", "여러 명이 동시에 같은 문서를 편집할 때 글자가 깨지지 않을까", "위키 페이지 사이의 연결을 시각적으로 한눈에 볼 수 있을까" 같은 것입니다. 모두 사용자가 매일 마주치는 실무 경험에 직결됩니다. 따라서 기능 항목을 단순히 O·X 로 나열하기보다, 워크플로우 관점에서 무엇이 가능하고 무엇이 어려운지를 비교해야 의사 결정이 흔들리지 않습니다.

칸반 보드는 AppFlowy 와 AFFiNE 이 네이티브로 제공하지만 구현 방식이 다릅니다. 실시간 협업은 AFFiNE 의 MCP 문서가 명시적으로 강조하는 강점이지만, AppFlowy 와 Outline 의 접근법도 각자 다른 색깔을 가집니다 [S10]. 양방향 링크와 그래프 뷰로 대표되는 지식 구조화는 도구마다 철학이 갈리는 영역입니다. AFFiNE 의 edgeless 캔버스, AppFlowy 의 데이터베이스 관계, Outline 의 위키 트리 구조는 같은 목표를 다른 길로 풀어냅니다 [S4][S7].

이 장의 핵심 메시지는 단순합니다. 칸반·실시간 협업·지식 구조화 세 영역에서 세 도구의 완성도는 동일하지 않으며, 팀 워크플로우가 어디에 무게 중심을 두느냐에 따라 적합 도구가 달라집니다. 여러분이 IT 담당자 또는 엔지니어로서 이 비교를 읽는다면, 각 절 끝에서 "우리 팀에는 어느 기능이 결정적인가"를 자문해 보시기 바랍니다.

6.1 칸반(Kanban) 보드 네이티브 지원 수준

칸반 보드는 프로젝트 진행 현황을 카드 단위로 시각화하는 가장 보편적인 방법입니다. Notion 의 보드 뷰에 익숙해진 팀이 대체 도구를 검토할 때 가장 먼저 확인하는 항목이기도 합니다. 세 도구 가운데 AppFlowy 와 AFFiNE 은 칸반을 네이티브 뷰로 제공하지만 출발점이 다르고, Outline 은 칸반을 제공하지 않습니다 [S1] [S4][S7].

6.1.1 AppFlowy Board 와 AFFiNE 데이터베이스 Kanban 뷰의 완성도 비교

AppFlowy 의 Board 는 처음부터 칸반을 기본 뷰로 설계한 결과입니다. 데이터베이스의 한 표현 양식으로 보드를 두기 때문에, 카드 안에 자유롭게 필드를 추가하고 그룹 기준을 단일 선택(Single select) 필드로 잡으면 곧바로 칼럼이 생성됩니다. 카드 안에서 하위 페이지를 열어 본문을 길게 작성할 수 있다는 점은 Notion 에 가장 가까운 사용 경험을 제공합니다 [S1].

AFFiNE 은 칸반을 별도의 보드 모듈이 아니라 데이터베이스의 한 뷰로 노출합니다. 같은 데이터 집합을 표·갤러리·칸반으로 전환하며 보는 구조이므로, 이미 데이터베이스로 정리해 둔 작업을 시각적으로 추적하기에는 자연스럽습니다. 다만 카드의 시각적 디테일, 그룹 기준의 유연성, 필터·자동화 항목이 AppFlowy Board 만큼 풍부하게 다듬어졌는지에 대해서는 실제 워크로드로 측정해 봐야 결론을 내릴 수 있습니다 [S1][S4].

기능 완성도라는 표현을 풀어쓰면 네 가지 축으로 나눌 수 있습니다. 첫째, 필터링이 다중 조건을 안정적으로 처리하는가. 둘째, 카드 이동을 트리거로 한 자동화가 가능한가. 셋째, 수식 필드로 진척률·우선순위 점수를 계산할 수 있는가. 넷째, 관계 필드로 다른 데이터베이스의 카드와 연결되는가. 두 도구 모두 이 네 항목을 어디까지 다루는지는 도입 전 PoC 에서 동일 시나리오로 점검해 비교 표를 만들어 두기를 권장합니다 [S1][S4].

실무 관점에서 보자면, AppFlowy 는 보드 자체의 사용 경험을 다듬는 데 무게를 두고 있고, AFFiNE 은 같은 데이터를 여러 시각화로 돌려 보는 통합성에 무게를 둡니다. 여러분의 팀이 칸반을 메인 업무 도구로 쓴다면 AppFlowy 가 손에 익기 쉽고, 데이터베이스를 다양한 뷰로 가공해 인사이트를 뽑아내는 패턴이 많다면 AFFiNE 의 접근이 어울립니다.

6.1.2 Outline의 Kanban 부재와 외부 도구 연동 보완 전략

Outline 은 처음부터 팀 위키에 집중한 완성형 지식베이스이며, 보드 뷰나 데이터베이스 칸반을 네이티브로 제공하지 않습니다 [S7]. 페이지 트리, 컬렉션, 강력한 검색을 중심으로 문서 자체의 가독성과 발견 가능성을 끌어올리는 방향에 충실합니다. 이러한 설계 철학은 "위키는 위키답게, 보드는 보드답게" 라는 도구 분리 원칙과도 통합니다.

칸반이 필요한 팀이 Outline 을 선택한다면 외부 도구 연동으로 보완하는 전략을 고려해야 합니다. 셀프호스팅 환경에서 자주 짝지어지는 대안은 GitLab Issues 보드, Plane, OpenProject, Wekan 같은 오픈소스 보드 도구입니다. Outline 페이지 본문에 이슈 링크를 임베드하고, 회의록·정책 문서는 Outline 에서, 작업 추적은 별도 보드에서 다루는 분업 구조를 만들 수 있습니다.

연동 전략을 선택할 때 점검할 항목은 세 가지입니다. 첫째, SSO IdP 를 보드 도구와 위키가 함께 쓰는가. 공식 Outline 은 외부 OIDC/SSO IdP 가 사실상 필수에 가깝다는 점이 알려져 있으므로, 보드 도구 역시 같은 IdP 에 묶을 수 있어야 운영 부담이 줄어듭니다 [S7]. 둘째, 양쪽 도구의 권한 모델이 부서·팀 단위로 일관되게 매핑되는가. 셋째, 검색이 두 도구의 데이터를 가로질러 동작하는지 또는 별도 검색 도구를 추가해야 하는지입니다.

이 분업 구조는 단점만 있는 것이 아닙니다. 위키와 보드를 한 도구에 몰아 넣으면 데이터 모델이 비대해지면서 권한·검색·백업이 복잡해지는 부작용이 따라옵니다. Outline 은 그 복잡도를 위키 본연의 영역으로 한정함으로써 운영 비용을 낮추는 선택지를 제공합니다. 여러분이 위키 품질을 최우선 가치로 두는 팀이라면, 칸반 부재는 결함이 아니라 설계 결정으로 받아들이고 보완 도구를 함께 그리는 접근이 더 현실적입니다 [S7].

6.2 실시간 협업·멘션·권한 관리

실시간 협업은 도구 도입의 체감 만족도를 좌우하는 가장 즉각적인 기능 영역입니다. 같은 문서를 여러 명이 동시에 편집할 때 커서가 자연스럽게 보이는지, 댓글·멘션이 알림으로 정확히 전달되는지, SSO 로 묶인 권한이 페

이지 단위로 깔끔하게 동작하는지가 운영 품질을 결정합니다.

6.2.1 실시간 공동 편집과 멘션·댓글 기능의 차이

AFFiNE 의 공식 MCP 문서는 도구의 핵심 가치로 "real-time collaboration, version control, user management" 세 가지를 명시적으로 강조합니다 [S10]. 이 표현은 단순한 마케팅 문구가 아니라, AFFiNE 의 협업 엔진이 처음부터 다수 동시 편집을 전제로 설계되었음을 보여주는 단서입니다. 여러 사용자가 같은 페이지의 다른 단락을 동시에 수정해도 커서 위치, 선택 영역, 본문 변경이 충돌 없이 합쳐지는 흐름은 작업 호흡을 끊지 않습니다.

AppFlowy 도 실시간 협업을 표방하지만, 클라우드 협업 기능이 셀프호스팅 사용자에게 어디까지 동등하게 제공되는지는 배포 모드별로 차이가 있습니다. 셀프호스팅을 검토하실 때는 동시 편집, 댓글, 멘션 알림 세 기능이 사용 중인 빌드에서 정확히 어떻게 동작하는지 별도 PoC 로 확인해 두시기를 권장합니다.

Outline 은 실시간 공동 편집을 안정적으로 지원하며, 본문 안에서 사용자를 멘션해 알림을 보내는 흐름이 위키 문화에 잘 어울리도록 다듬어져 있습니다 [S7]. 댓글은 페이지 단위로 스레드를 형성하여 의사 결정의 맥락을 남기기에 적합합니다. 결정 사항을 본문에 반영하고 토론은 댓글 스레드에 남기는 운영 패턴은 위키의 신뢰도를 유지하면서도 협업의 흔적을 잃지 않게 해 줍니다.

세 도구를 같은 잣대로 비교하면, AFFiNE 은 동시 편집의 기술적 견고함을 전면에 내세우고, Outline 은 위키 워크플로우 안에서 협업 기능을 다듬는 데 집중하며, AppFlowy 는 데이터베이스와 보드를 포함한 다양한 작업 종류에서 협업을 확장하는 방향에 무게를 둡니다. 여러분의 팀이 동시 작성의 빈도가 높다면 AFFiNE 의 강점이 두드러지고, 결정 기록·문서 합의를 중시한다면 Outline 의 멘션·댓글 모델이 적합합니다 [S7][S10].

6.2.2 SSO 권한 거버넌스와 이메일/비밀번호 로그인 옵션 차이

권한 거버넌스의 출발점은 인증 방식입니다. 공식 Outline 은 외부 OIDC/SSO IdP 를 사실상 필수로 요구한다는 점이 문서에서 거듭 확인됩니다 [S7]. Keycloak, Authentik, Auth0 같은 IdP 를 별도로 준비해야 하므로, 인증 인프라가 이미 갖춰진 조직에는 자연스럽지만 단독 도구만 빠르게 띄우려는 소규모 팀에는 진입 장벽이 됩니다. 반면 이 강제는 조직 전체의 인증을 단일 IdP 로 통합하는 장기적 거버넌스 관점에서는 분명한 장점입니다.

AppFlowy 와 AFFiNE 은 셀프호스팅 빌드에서 이메일/비밀번호 로그인을 별도 IdP 없이 활성화하는 옵션을 제공하는 경우가 있습니다. 이는 초기 도입 속도를 높여 주지만, 사용자 수가 늘어나면 결국 SSO 로 통합해야 한다는 점은 동일합니다. 여러분이 IT 담당자라면 도입 첫 단계부터 "최종 상태는 SSO 통합"이라는 그림을 그려 두고, 도구별 SSO 지원 범위(OIDC, SAML, LDAP, SCIM 사용자 동기화) 를 비교 표로 정리해 두시기를 권장합니다.

권한 모델 자체도 도구마다 결이 다릅니다. Outline 은 컬렉션 단위 권한과 그룹 기반 권한을 결합해 위키 구조와 권한 구조가 자연스럽게 맞물리도록 설계했습니다 [S7]. AppFlowy 와 AFFiNE 은 워크스페이스·페이지 단위 권한을 제공하며, 데이터베이스 행이나 보드 카드 수준까지 권한이 내려가는지는 빌드와 버전에 따라 차이가 있습니다. 민감 정보를 다루는 조직이라면 페이지 트리 어느 깊이까지 권한이 상속되는지, 외부 게스트 공유가 가능한지를 도입 전 반드시 확인해야 합니다.

감사 로그와 사용자 수명 주기 관리도 거버넌스의 핵심 항목입니다. 누가 언제 어떤 페이지를 봤는지, 권한이 언제 변경되었는지, 퇴사자의 계정이 자동으로 비활성화되는지가 보안 감사 응대의 출발점입니다. 세 도구 모두 기

본 감사 로그를 제공하지만, SCIM 기반 자동 프로비저닝·디프로비저닝까지 닿는 수준에는 차이가 있으므로 조직의 보안 정책과 매핑해 검증하시기 바랍니다 [S7].

6.3 온톨로지화: 관계형 필드·양방향 링크·시각적 그래프

지식 구조화는 도구를 오래 쓸수록 가치가 커지는 영역입니다. 페이지가 수백, 수천 개로 늘어나도 의미 있게 연결되고 검색되는가, 새로운 문서를 만들 때 기존 지식과 자동으로 이어지는가가 위키의 장기 수명을 결정합니다. 이 영역에서 세 도구의 철학은 가장 뚜렷하게 갈립니다 [S1][S4].

6.3.1 데이터베이스 관계형 필드와 양방향 링크 지원 수준

관계형 필드는 한 데이터베이스의 행을 다른 데이터베이스의 행과 연결하는 기능입니다. 프로젝트 데이터베이스의 한 행이 고객 데이터베이스의 여러 행과 연결되거나, 회의록 페이지가 참여자 데이터베이스와 묶이는 패턴이 대표적입니다. AppFlowy 의 Board 는 데이터베이스의 한 뷰이므로 관계 필드를 통해 카드 사이의 의미적 연결을 만들 여지가 있고, AFFiNE 의 데이터베이스 또한 같은 방향으로 발전 중인 것으로 보입니다 [S1][S4].

양방향 링크는 위키의 핵심 기능입니다. 페이지 A 에서 페이지 B 를 언급하면 페이지 B 의 백링크 영역에 A 가 자동으로 나타나는 흐름은, 문서 작성자가 명시적으로 양쪽을 잇지 않아도 지식이 그물처럼 짜이게 해 줍니다. 양방향 링크가 견고하면 오래된 문서가 새 문서에 자연스럽게 호출되어 사장되지 않습니다 [S1][S4].

세 도구의 양방향 링크 지원 수준은 동일하지 않습니다. AFFiNE 은 edgeless 캔버스와 결합하여 링크의 시각적 표현까지 강화하는 방향을 추구합니다. AppFlowy 는 페이지·데이터베이스·보드 사이의 데이터 모델이 통합되어 있어, 같은 객체를 여러 뷰에서 다루는 과정에서 자연스럽게 연결 망이 형성됩니다. Outline 은 위키 본연의 페이지 트리와 멘션 기반 링크에 집중합니다 [S4][S7].

엔지니어 관점에서 보자면, 관계 필드와 양방향 링크가 데이터 모델 안에서 어떻게 표현되는지, 그리고 백업·이전 시 그 연결이 깨지지 않는지가 핵심 점검 항목입니다. 도구 잠금을 줄이려면 관계 정보가 단순한 ID 참조로 보존되는지, 마크다운 또는 표준 포맷으로 내보내질 때 링크가 어떻게 표현되는지 미리 확인해 두시기를 권장합니다 [S1][S4].

6.3.2 시각적 그래프 뷰와 캔버스 기반 구조화의 차이

시각적 그래프 뷰는 페이지·태그·링크를 노드와 엣지로 그려 지식의 전체 윤곽을 한눈에 보여 줍니다. 그래프 뷰는 새로운 연결을 발견하거나 고립된 페이지를 찾아내는 데 유용합니다. 다만 그래프 뷰가 단순한 시각화에 머무는지, 아니면 그래프 위에서 곧바로 편집·재구성이 가능한지에 따라 실무 가치가 크게 달라집니다 [S4].

AFFiNE 은 edgeless 캔버스라는 무한 캔버스 모델을 전면에 내세웁니다 [S4]. 캔버스 위에 페이지·이미지·도형·다른 캔버스를 자유롭게 배치하고 연결선으로 묶을 수 있어, 문서의 시각적 구조와 의미적 구조를 같은 표면 위에서 다루는 경험을 제공합니다. 화이트보드 도구와 위키 도구의 경계가 사라지는 방향이며, 디자인·기획 워크플로우와 잘 맞물립니다.

AppFlowy 는 데이터베이스 중심으로 구조화를 풀어내는 색깔이 강합니다. 캔버스보다는 표·보드·갤러리 같은 정형 뷰에서 데이터를 정리한 뒤, 페이지 본문에서 이들을 임베드해 의미를 부여하는 패턴이 자연스럽습니다. 그래프 뷰가 별도 기능으로 제공되지 않더라도, 데이터베이스 관계와 페이지 트리만으로 충분히 운용되는 팀이라면 단순한 모델이 오히려 장점이 됩니다.

Outline 은 시각적 그래프나 캔버스 뷰보다는 페이지 트리, 컬렉션, 백링크, 강력한 전문 검색의 조합으로 지식을 구조화합니다 [S7]. 화려한 시각화가 없더라도 검색이 빠르고 정확하면 위키는 충분히 살아 움직입니다. 여러분의 팀이 "그래프가 멋지지만 결국 검색으로 찾는다" 는 경험을 자주 한다면, Outline 의 절제된 접근이 운영 비용을 줄여 줍니다.

결론을 정리하자면, 시각적 캔버스의 자유도가 핵심 가치라면 AFFiNE, 정형 데이터베이스로 잘 정돈된 구조를 원한다면 AppFlowy, 검색과 위키 트리에 충실한 운용을 원한다면 Outline 이 각자의 답을 제시합니다 [S4] [S7]. 어느 도구도 모든 팀의 정답이 되지는 않으며, 워크플로우 중심으로 선택하는 사고가 도입 성공의 출발점이 됩니다.

7장: Notion에서 세 도구로 전환하는 현실적 경로

여러분이 Notion 에서 셀프호스팅 도구로 옮기겠다고 마음먹은 순간, 가장 먼저 떠올리는 그림은 보통 "내보내기 한 번에 모든 페이지가 새 도구로 옮겨가는 장면" 입니다. 그러나 현실은 그렇게 단순하지 않습니다. Notion 의 페이지는 단순한 문서 묶음이 아니라 블록·데이터베이스·관계·임베드·동기화 블록이 얽힌 작은 응용 프로그램이며, 이 응용 프로그램을 다른 도구로 옮기는 일은 파일 복사가 아니라 모델 번역에 가깝습니다.

이 장은 세 도구로의 이전을 세 갈래로 나누어 살펴봅니다. 먼저 사용자가 마주칠 화면 차이, 즉 UX 유사성과 학습 곡선을 비교합니다. 그다음 Notion 이 제공하는 내보내기 형식의 한계와 각 도구가 받아들이는 가져오기 경로를 살핍니다. 마지막으로 이전 과정에서 빠짐없이 발생하는 손실 항목을 분류하고, 도구별로 그 손실을 메우는 보완 전략과 사내 정착까지의 작업 견적을 인공수로 환산해 제시합니다.

미리 결론을 알려드리면 이렇습니다. AppFlowy 는 사용자가 가장 적게 흔들리지만 데이터베이스 관계 복원에 손이 많이 듭니다. AFFiNE 은 화이트보드와 결합한 새로운 사고 모델을 요구해 사용자 적응 부담이 가장 크지만, 옮기는 데이터의 폭은 좁힐 수 있습니다. Outline 은 데이터베이스를 포기하고 위키만 남기는 결정만 받아들이면 가장 깔끔하게 정착합니다. 세 갈래 모두 "버릴 것을 먼저 정하는 의사결정" 이 성공의 절반을 차지합니다.

마지막으로 한 가지 당부를 드립니다. 이전을 기술 작업으로만 보면 실패합니다. 사용자가 새 도구에서 어떤 흐름으로 일하게 될지, 어떤 페이지가 옮겨갈 가치가 있는지를 먼저 정리한 뒤 도구를 고르시기 바랍니다. 이전은 정리의 기회이지 복제의 의무가 아닙니다.

7.1 Notion과의 UX·기능 유사성 비교

사용자 화면의 친숙도는 이전 성공률을 가르는 첫 번째 변수입니다. 세 도구는 Notion 과의 거리감이 매우 다르며, 이 거리감이 그대로 사내 교육 비용과 저항의 크기를 결정합니다.

7.1.1 AppFlowy의 Notion UX 정공법 추종과 사용자 적응 부담

AppFlowy 는 Notion 의 UX 를 가장 가깝게 따라간 도구입니다 [S1]. 좌측 사이드바에 페이지 트리가 있고, 본문은 슬래시 명령으로 블록을 추가하는 방식이며, 데이터베이스는 표·보드·캘린더·갤러리 같은 뷰로 같은 데이터를 여러 모습으로 보여 줍니다. 페이지 안에 데이터베이스를 끼워 넣거나 데이터베이스 한 행을 다시 페이지로 여는 동작도 Notion 과 거의 같은 손맛으로 작동합니다.

이 정공법 추종은 의사결정권자에게 매우 매력적입니다. 사내에 Notion 사용자가 많을수록 재교육 시간이 짧아지기 때문입니다. 슬래시 명령, 토글, 콜아웃, 인용 같은 자주 쓰는 블록의 호출 방식이 같고 단축키 체계도 유사

해서, 사용자 다수는 첫 화면을 마주한 순간 "Notion 비슷한 도구 구나" 하고 자연스럽게 받아들입니다. 학습 곡선이 거의 보이지 않는다는 점이 가장 큰 장점입니다.

다만 "거의 같다" 와 "완전히 같다" 사이에는 미세한 틈이 있고, 이 틈이 이전 초기 한두 주 동안 사용자 불만의 80% 를 만들어 냅니다. 특정 임베드가 안 보인다거나 동기화 블록 자리가 일반 블록으로 바뀌어 있다거나 데이터베이스 관계가 단순 텍스트로 풀려 있는 경우가 그 예입니다. 사용자는 "같다고 들었는데 왜 다르냐" 며 더 크게 실망합니다. 그래서 AppFlowy 로 옮길 때는 "닮은 점" 을 강조하기보다 "다른 점 목록" 을 미리 공유하는 편이 사용자 적응 부담을 낮추는 데 효과적입니다.

여러분이 사용자 저항을 최소화하고 싶다면 AppFlowy 가 가장 안전한 선택입니다. 단, 적응 부담은 사라지지 않고 "기능이 거의 같다는 기대" 와 "실제 미세 차이" 사이의 간극으로 옮겨 갑니다. 이 간극을 좁히는 일은 도구의 몫이 아니라 이전 담당자의 몫임을 처음부터 분명히 합의해 두시기 바랍니다.

7.1.2 AFFiNE 과 Outline의 차별화된 UX와 학습 곡선

AFFiNE 은 Notion 과의 거리가 가장 먼 도구입니다. 같은 데이터를 문서 모드와 무한 캔버스 모드에서 동시에 보여 주는 이중 모드 화면을 채택했기 때문입니다 [S4]. 사용자는 문서 한 편을 쓰다가 같은 페이지를 캔버스로 열어 화이트보드 위에 끌어다 놓고 다른 페이지·이미지·도형과 함께 배치할 수 있습니다. 이 모델은 회의록·설계 토론·디자인 리뷰처럼 시각적 사고가 필요한 작업에 강력하지만, "글만 쓰고 싶은 사용자" 에게는 두 모드 사이를 오가야 한다는 부담을 줍니다.

학습 곡선은 두 단계로 나뉩니다. 첫 단계는 문서 모드만 쓰는 데 적응하는 며칠이고, 둘째 단계는 캔버스 모드의 의미를 이해하고 자기 업무에 끼워 넣기 시작하는 한 달 안팎입니다. 사용자가 둘째 단계까지 넘어가지 못하면 AFFiNE 은 그저 "조금 낯선 Notion" 으로 남고, 도구의 진짜 가치가 살아나지 않습니다. 따라서 AFFiNE 도입을 결정한 조직은 캔버스 모드를 적극적으로 시연·교육할 내부 챔피언을 함께 정해 두는 편이 좋습니다.

Outline 은 또 다른 결의 차별화를 보여 줍니다. Outline 은 처음부터 "팀 위키" 라는 단일 목적에 집중한 완성형 지식베이스이며, 데이터베이스·캘린더·보드 같은 Notion 의 다용도 기능을 의도적으로 덜어 냈습니다 [S7]. 화면은 문서 트리, 검색, 본문 편집기 세 가지로 단순화돼 있고, 사용자가 새 도구에서 익혀야 할 메뉴가 적습니다. 위키만 필요한 조직이라면 학습 곡선이 가장 짧은 도구입니다.

그러나 Outline 의 단순함은 양날의 검입니다. Notion 에서 데이터베이스를 적극 쓰던 팀은 "이 도구로는 우리가 하던 일을 못 한다" 는 인식을 빠르게 갖게 됩니다. 의사결정권자는 이 인식을 부정하지 말고 받아들여야 합니다. Outline 으로 옮기는 결정은 "위키만 남기고 데이터베이스 영역은 다른 도구로 분리한다" 는 영역 분할의 의사결정이지, Notion 의 모든 쓰임새를 한 도구로 옮기겠다는 결정이 아닙니다. 이 점을 사용자에게 솔직하게 설명하면 학습 곡선보다 기대치 조정이 더 큰 변수가 됩니다.

7.2 Notion 데이터 내보내기와 이전 경로

이전 작업의 기술 출발점은 Notion 이 제공하는 내보내기 기능입니다. Notion 은 사용자가 워크스페이스 단위·페이지 단위로 내용을 추출할 수 있는 세 가지 형식을 제공하며, 각 형식은 무엇을 살리고 무엇을 잃는지가 다릅니다.

7.2.1 Notion 의 Markdown·CSV·HTML 내보내기 형식과 각 형식의 손실

Markdown & CSV 형식은 가장 흔히 쓰이는 내보내기 옵션입니다. 일반 페이지는 `.md` 파일로, 데이터베이스는 `.csv` 파일로 빠지고 첨부 파일은 별도 폴더에 모입니다. 이 형식의 큰 장점은 사람이 읽을 수 있고 어떤 도구로든 받아들이기 쉽다는 점입니다. 단점은 콜아웃·토글·동기화 블록·임베드 같은 Notion 특유의 블록 다수가 평탄한 텍스트나 인용 표시로 풀려 버린다는 데 있습니다. 데이터베이스의 다중 뷰는 단일 CSV 한 장으로 합쳐지며 뷰의 정렬·필터·그룹 설정은 사라집니다.

HTML 형식은 시각적 충실도가 가장 높습니다. 페이지의 모습을 화면 그대로 옮기려는 목적이라면 HTML 이 가장 안전합니다. 그러나 HTML 은 다른 협업 도구의 가져오기 통로가 잘 받아들이지 못하는 형식이라 실제 이전 작업에서는 거의 쓰이지 않습니다. HTML 은 보관용 스냅숏으로는 좋고, 작동하는 데이터로 옮기기에는 적합하지 않습니다.

세 형식 모두가 공통으로 잃는 항목이 있습니다. 데이터베이스 관계(릴레이션) 가 그 대표 사례입니다. Notion 에서 두 데이터베이스를 잇던 관계 컬럼은 내보낸 CSV 에서 단순 텍스트(대상 행의 제목) 로 풀려 버리며, 가져오는 쪽에서 자동으로 복원되지 않습니다. 롤업(다른 데이터베이스 값을 끌어다 계산한 컬럼) 도 사라지고, 페이지에 끼워 넣은 데이터베이스(인라인 데이터베이스) 도 일반 링크 한 줄로 평탄화됩니다. AppFlowy·AFFiNE·Outline 을 포함한 8종을 비교한 자료에서도 Notion 의 데이터베이스 관계 손실을 명시적으로 경고하고 있습니다.

여러분이 IT 담당자라면 내보내기 형식을 고르기 전에 "워크스페이스 안에 데이터베이스 관계가 얼마나 많이 쓰이고 있는지" 를 먼저 측정해 보시기 바랍니다. 관계가 거의 없는 워크스페이스라면 Markdown & CSV 한 형식으로 충분하며, 관계가 많은 워크스페이스라면 형식 선택 이전에 "관계를 어떻게 복원할지" 라는 별도 계획이 필요합니다.

7.2.2 세 도구의 Notion 가져오기 경로 비교

AppFlowy 는 세 도구 중 Notion 가져오기에 가장 적극적으로 대응합니다 [S1]. Notion 의 Markdown & CSV 내보내기를 그대로 받아들여 페이지 트리·일반 블록·데이터베이스 표 뷰까지 복원하는 통로를 제공합니다. 이 통로는 페이지 계층과 표 뷰까지는 잘 살리지만, 데이터베이스 관계와 다중 뷰 설정은 살리지 못합니다. 이전 후에는 표 뷰만 남고 보드·캘린더·갤러리 뷰는 사용자가 직접 다시 만들어야 합니다.

AFFiNE 은 Notion 의 Markdown 가져오기를 지원하지만, AFFiNE 의 핵심 가치인 캔버스 모드는 Notion 데이터를 자동으로 변환해 주지 않습니다 [S4]. 가져온 데이터는 문서 모드 안의 페이지로 자리 잡고, 캔버스 모드의 활용은 이전 후 사용자가 직접 시작해야 합니다. 이 특성은 손실이라기보다 도구 철학의 차이입니다. AFFiNE 으로 옮기는 일은 "Notion 데이터를 시작점으로 새 작업 방식을 도입한다" 는 의미에 가깝습니다.

Outline 은 Markdown 가져오기를 받아들이지만 데이터베이스를 도구 안에 두지 않는다는 설계 결정 때문에 CSV 데이터를 위키 페이지로 직접 옮기지 않습니다 [S7]. 이전을 결정한 조직은 데이터베이스를 별도 도구(전용 데이터베이스·스프레드시트·외부 도구) 로 분리하고 Outline 에는 위키로 정리할 수 있는 페이지만 옮기는 방식을 택하게 됩니다. 대신 Outline 은 REST API 의 documents 계열 엔드포인트를 통해 페이지 생성·갱신·검색을 외부 스크립트에서 자동화할 수 있어, Notion CSV 를 적당한 위키 페이지 묶음으로 변환하는 사내 도구를 만들어 일괄 등록하는 경로가 잘 통합니다 [S9].

세 도구의 가져오기 경로를 한 줄로 정리하면 이렇습니다. AppFlowy 는 가장 자동에 가깝고, AFFiNE 은 데이터는 받지만 가치 전환은 사용자의 몫이며, Outline 은 API 자동화를 끼워 넣어야 가장 깔끔합니다. 도구 선정 단계에서 이 경로의 차이를 먼저 시험해 보시기 바랍니다.

7.3 이전 과정 데이터 손실 항목과 보완 방법

가져오기 통로가 가장 잘 갖춰진 AppFlowy 로 옮기더라도 손실은 0 이 되지 않습니다. 이 절에서는 손실 항목을 분류하고, 도구별로 그 손실을 메우는 작업이 어느 정도의 인공수를 요구하는지를 함께 살핍니다.

7.3.1 데이터베이스 관계·임베드·페이지 계층 손실의 분류

손실 항목은 크게 네 갈래로 나뉩니다. 첫 갈래는 데이터베이스 관계와 롤업입니다. 이는 단순 컬럼 손실이 아니라 데이터 모델의 손실입니다. 두 데이터베이스 사이의 연결이 사라지므로, "이 행이 어느 행에 연결되어 있었다" 는 정보를 외부 스크립트나 사용자 수동 작업으로 복원해야 합니다.

둘째 갈래는 임베드와 동기화 블록입니다. Notion 에서 외부 서비스를 끼워 넣었던 블록(특히 Figma, Loom, Miro 등) 은 내보내기 시 단순 링크로 풀리고, 동기화 블록은 일반 블록으로 변환되면서 동기화 관계가 끊깁니다. 이 손실은 자동 복원이 불가능에 가깝고, 새 도구에서 어떤 외부 임베드를 다시 지원할지 도구 선택 시 함께 결정해야 합니다.

셋째 갈래는 페이지 계층과 권한입니다. 페이지의 트리 구조는 대체로 잘 옮겨가지만, Notion 의 페이지별 공유·접근 권한 설정은 형식에 담기지 않습니다. 사용자·그룹별 접근 권한은 새 도구의 권한 체계에 맞춰 다시 정의해야 합니다. Outline 처럼 그룹 기반 권한이 명확한 도구라면 오히려 이전을 기회로 권한을 정돈할 수 있습니다.

넷째 갈래는 블록 호환성입니다. 콜아웃·토글·다중 컬럼·수식 같은 블록은 도구마다 받아들이는 정도가 다릅니다. AppFlowy 는 대부분을 비슷한 모양으로 재현하고, AFFiNE 은 문서 모드 안에서 다수를 재현하며, Outline 은 일부를 단순화해 받아들입니다. 이 차이는 "이전 후 사용자가 페이지 하나당 손볼 시간" 의 총합으로 환산됩니다.

7.3.2 도구별 보완 전략과 사내 정착까지의 작업 견적

AppFlowy 로 옮길 때의 보완 전략은 "데이터베이스 관계 복원" 에 집중됩니다. CSV 안에서 관계 컬럼은 대상 행의 제목으로 풀려 있으므로, 가져온 뒤 외부 스크립트로 두 데이터베이스의 행을 짝지어 관계 컬럼을 다시 채우는 작업이 필요합니다. 페이지 1,000개·데이터베이스 10개 규모의 일반 워크스페이스 기준으로 보면 가져오기 자체에는 1인주 안팎, 관계 복원과 뷰 재구성에는 추가 2~3인주, 사용자 적응 지원에 1~2인주가 듭니다. 합계 4~6인주가 표준 견적입니다.

AFFiNE 으로 옮길 때는 "옮길 데이터의 폭" 을 결정하는 일이 가장 큰 비중을 차지합니다. 모든 페이지를 다 옮기기보다 "캔버스 모드와 결합해 새 가치를 얻을 페이지" 와 "단순 보관용 페이지" 를 가르고, 후자는 별도 보관소(또는 export 압축본) 에 두는 편이 효과적입니다. 같은 규모 기준으로 보면 데이터 분류와 가져오기에 2~3인주, 사용자 챔피언이 캔버스 사용 사례를 만들어 사내에 시연하는 작업에 2~4인주, 도구 정착에 한 달 안팎의 후속 지원이 필요합니다. 합계 5~7인주가 표준 견적이며 후속 지원 기간이 가장 깁니다.

Outline 으로 옮길 때의 보완 전략은 두 가지로 나뉩니다. 첫째는 데이터베이스를 어디로 분리할지를 정하는 작업이고, 둘째는 위키로 옮길 페이지를 정돈하는 작업입니다. 데이터베이스 분리 결정에는 1~2인주, REST API documents 엔드포인트를 활용한 일괄 등록 스크립트 작성에 1~2인주, 위키 정착과 권한 정돈에 1인주가 듭니다 [S9]. 합계 3~5인주로 세 도구 중 작업 견적이 가장 작습니다. 단, "데이터베이스를 분리한다" 는 결정 자체가 의사결정권자의 사전 합의를 필요로 한다는 점은 인공수 외의 비용입니다.

세 견적을 비교해 보면, 이전 비용은 도구의 기술 특성뿐 아니라 조직의 결정 속도에도 좌우됨을 알 수 있습니다. 여러분의 조직이 "Notion 사용 방식을 그대로 가져간다" 는 보수 노선을 택한다면 AppFlowy 가, "이전을 기회

로 사고 방식을 바꾼다" 는 적극 노선을 택한다면 AFFiNE 이, "위키와 데이터베이스를 분리한다" 는 단순화 노선을 택한다면 Outline 이 작업 견적과 잘 맞물립니다. 이전 작업은 도구가 결정하는 것이 아니라 노선이 결정한다는 점을 마지막으로 다시 한 번 강조합니다.

8장: 기업 도입 전 점검 항목

기능 비교표만 보고 도입을 결정하면 운영 6개월 시점에 후회할 가능성이 높습니다. 앞선 장에서 AppFlowy·AFFiNE·Outline 세 도구의 기능과 아키텍처를 비교했지만, 실제 기업 환경에 들이는 일은 라이선스 조항을 법무팀과 합의하고 운영 인력을 배정하며 망분리 정책에 맞게 인프라를 구성하는 작업의 합입니다. 본 장은 의사결정 직전에 반드시 점검해야 할 네 가지 항목을 정리합니다.

여러분의 조직이 어떤 규모이든 도입 전 점검은 동일한 순서를 따르는 편이 안전합니다. 라이선스 적합도를 먼저 확인하여 법무 리스크의 상한을 가늠하고, 운영 복잡도와 인프라 의존성을 점수화하여 인력 부담을 산정하며, 한국어 지원과 모바일 성숙도로 사용자 경험의 하한을 검증하고, 마지막으로 오픈소스 활성도 지표로 장기 운영 위험을 진단합니다. 이 순서를 거꾸로 진행하면 GitHub Star 수가 많다는 인상에 끌려 라이선스 조항을 늦게 발견하는 결과로 이어집니다.

특히 국내 기업 환경은 망분리 규정, 개인정보보호법, 클라우드보안인증(CSAP) 요건 같은 제도적 제약을 포함합니다. 본 장에서 다루는 점검 항목은 일반적인 해외 도입 가이드보다 한 단계 더 보수적인 기준을 따르며, 한국어 처리 품질처럼 영문 가이드에서 거의 다루지 않는 항목까지 포함합니다.

마지막으로 강조할 점은 본 장의 모든 평가가 2026년 6월 시점의 실측 데이터를 기반으로 한다는 사실입니다. 오픈소스 프로젝트의 활성도와 라이선스 조항은 분기 단위로 변할 수 있으므로 도입 직전에 다시 한 번 GitHub 저장소와 LICENSE 파일을 직접 확인하시기 바랍니다.

8.1 라이선스 컴플라이언스와 법무 리스크 점검

라이선스 조항은 도입 후 변경이 어려운 결정입니다. 3장에서 세 도구의 라이선스 종류를 비교했으므로 본 절은 시나리오별 적합도와 법무 합의 절차에 집중합니다. 사내 단독 사용, 계열사 공유, 외부 SaaS 형태의 재배포 세 가지 사용 패턴은 각각 다른 조항에 걸립니다.

8.1.1 사내 사용·계열사 공유·SaaS 재배포 시나리오별 라이선스 적합도

Outline 은 BSL 1.1 라이선스 [S8] 를 채택하여 사내 사용에는 제약이 없지만 외부 고객에게 호스팅 서비스 형태로 제공하는 행위는 4년 경과 전까지 금지됩니다. 사내 위키나 부서간 협업 용도라면 안전하며, 자회사 직원에게 동일 인스턴스를 열어주는 것도 일반적으로 허용 범위에 들어갑니다. 다만 자회사가 별도 법인이고 별도 과금이 발생한다면 라이선스 조항의 해석을 법무팀과 사전에 합의하시기 바랍니다.

AppFlowy 는 AGPL-3.0 [S3] 으로 배포됩니다. 이 라이선스는 네트워크 너머로 서비스를 제공할 때도 소스 코드 공개 의무가 발동되는 강한 카피레프트 조항을 포함합니다. 사내 사용은 문제가 없지만, AppFlowy 를 기반으로 커스터마이즈한 협업 도구를 외부 SaaS 로 판매하려면 수정한 모든 소스를 사용자에게 제공해야 합니다. 사내 임직원에게만 사용을 허용하는 경우에도 외부 협력사가 동일 인스턴스에 접속한다면 AGPL 조항이 발동될 여지가 있으므로 접속 경로 설계를 신중히 검토해야 합니다.

AFFiNE 은 가장 복잡한 CE+EE [S5][S6] 구조를 가집니다. 커뮤니티 에디션은 MIT 영역(packages/backend·packages/common/native 외) 위에서 동작하고, 엔터프라이즈 에디션 라이선스는 위 두 디렉터리의 백엔드 핵심에 별도 상용 조건이 적용됩니다. 핵심 쟁점은 EE 무료 사용 상한선이 LICENSE 원문에 명시되지 않았다는 점 [S5] 입니다. 좌석 수 기준 구독을 요구하면서도 무료 사용 상한이 공개되지 않은 상태이므로, 사내 운영을 전제로 한 도입 검토 시 영업 채널을 통한 사전 확인을 권합니다.

세 시나리오를 표로 정리하면 사내 단독 사용은 세 도구 모두 적합하고, 계열사 공유는 AGPL-3.0 의 해석 여지를 검토해야 하며, 외부 SaaS 재배포는 Outline 의 BSL 조항과 AFFiNE EE 의 미공개 상한선이 발목을 잡습니다.

8.1.2 법무 합의 준비 단계와 라이선스 모니터링 체계

법무팀과의 합의는 도입 의향서를 작성하기 전 단계에서 시작해야 합니다. 첫 번째 단계는 GitHub 저장소에서 LICENSE 원문을 다운로드하여 사내 법무 문서 관리 시스템에 등록하는 작업입니다. AFFiNE 처럼 MIT 영역과 EE 영역이 분리된 경우 [S5] 두 LICENSE 파일을 모두 보관하고 각 파일의 SHA-256 해시를 함께 기록하면 추후 조항 변경 시점을 추적할 수 있습니다. 국내 기업이라면 정보통신산업진흥원(NIPA) 공개SW포털의 「오픈소스SW 라이선스 가이드」와 한국저작권위원회 가이드를 함께 참조하여, 사내 법무팀 검토 양식에 라이선스 카테고리(허용형·약한 카피레프트·강한 카피레프트), 의무 사항, 면책 조항을 항목별로 매핑해 두시기를 권합니다.

두 번째 단계는 사용 시나리오를 문서화하는 작업입니다. "사내 임직원 N명이 사내 네트워크에서 위키 용도로 사용한다" 정도의 한 문장 시나리오를 라이선스 조항과 매핑하여 법무팀이 GO 결정을 내릴 수 있도록 준비합니다. 이때 외부 접속 경로(VPN, 협력사 게스트 계정, 모바일 외부망 접속)를 모두 열거해야 AGPL 같은 카피레프트 조항의 발동 여부를 정확히 판단할 수 있습니다.

세 번째 단계는 라이선스 모니터링 체계입니다. 오픈소스 프로젝트는 메이저 버전 변경 시점에 라이선스를 변경하는 선례가 있습니다. Elastic 이 2021년 ELv2 로 전환한 선례, MongoDB 가 2018년 SSPL 로 전환한 선례가 대표적입니다. 분기 1회 LICENSE 파일 해시 비교와 릴리스 노트의 라이선스 관련 키워드 검색을 자동화하시면 라이선스 변경을 조기에 인지할 수 있습니다.

네 번째 단계는 백업 전환 계획입니다. 라이선스가 갑자기 변경되어 더 이상 사용할 수 없게 되었을 때 어떤 도구로 옮길 것인지를 사전에 정해두어야 합니다. AppFlowy → AFFiNE CE 또는 Outline → 다른 위키 도구 같은 전환 경로를 도입 단계에서부터 설계하면 라이선스 리스크의 상한을 추정할 수 있습니다.

8.2 운영 복잡도·인프라 의존성·망분리 적합성

운영 복잡도는 도입 후 가장 빨리 체감되는 부담입니다. 의존 서비스가 많을수록 장애 지점이 늘어나고 백업·복구 절차가 복잡해지며 망분리 환경에서는 외부 의존이 곧 차단 요인이 됩니다. 본 절은 1점부터 5점까지의 정성 평가와 망분리 적합성을 함께 다룹니다.

8.2.1 운영 복잡도 1~5점 점수 비교와 의존 서비스 부담 평가

운영 복잡도 점수는 의존 서비스 개수, 상태 저장 컴포넌트의 종류, 백업 대상의 분산도, 인증 통합의 난이도를 기준으로 산정합니다. AppFlowy-Cloud 는 PostgreSQL, Redis, MinIO, GoTrue [S2] 네 종류의 컴포넌트를 모두 운영해야 하므로 복잡도 4점에 해당합니다. 각 컴포넌트가 별도 상태를 가지므로 백업 일관성을 맞추는 작업이 단일 PostgreSQL 백업보다 복잡합니다.

Outline 은 PostgreSQL, Redis, S3 호환 스토리지, 외부 OIDC IdP [S7] 네 종류에 의존하지만 OIDC 는 외부 의존이고 Redis 는 캐시 용도이므로 실질적인 상태 저장 컴포넌트는 PostgreSQL 과 객체 스토리지 두 개입니다. 운영 복잡도 3점이 적절합니다. 단, OIDC 외부 의존이 망분리 환경에서는 차단 요인으로 작용한다는 점을 별도로 평가해야 합니다.

AFFiNE 의 의존성은 PostgreSQL 중심 [S6] 으로 상대적으로 단순합니다. 운영 복잡도 3점에 해당하지만 canary 릴리스 채널을 사용하는 경우 [S4] 잦은 업데이트와 마이그레이션 부담이 더해지므로 실질적인 운영 부담은 4점 수준으로 상승합니다. 안정 채널만 사용한다면 3점을 유지할 수 있지만 안정 채널의 릴리스 주기가 길어지면 보안 패치 적용이 늦어지는 트레이드오프가 있습니다.

세 도구 모두 컨테이너 기반 배포가 가능하므로 Kubernetes 환경이라면 운영 복잡도 차이가 어느 정도 흡수됩니다. 그러나 백업·복구 절차는 컨테이너 오케스트레이션과 별개로 설계해야 하며, 특히 객체 스토리지를 사용하는 AppFlowy 와 Outline 은 PostgreSQL 백업 시점과 객체 스토리지 백업 시점을 동기화하는 절차를 마련해야 합니다.

8.2.2 망분리 환경에서 외부 IdP·외부 SaaS 의존의 차단 요인

망분리 환경의 첫 번째 차단 요인은 외부 OIDC IdP 의존입니다. Outline 은 외부 OIDC IdP 가 필수 [S7] 이므로 폐쇄망 내부에 자체 OIDC 제공자(Keycloak·Authentik 등)를 구축해야 합니다. 사내에 이미 SK쉴더스·SGA솔루션즈 등 국내 SSO 솔루션이나 사내 LDAP·AD 가 운영 중이라면 OIDC 브리지를 끼워 통합이 용이하지만, 그렇지 않다면 OIDC 인프라 구축 자체가 별도의 프로젝트가 됩니다.

AppFlowy 는 GoTrue 를 내장 [S2] 하여 인증을 자체 처리하므로 외부 IdP 의존이 없습니다. 망분리 환경에 가장 적합한 구조이지만, 사내 통합 SSO 정책이 있다면 GoTrue 와 사내 SSO 간 사용자 동기화 절차를 별도로 설계해야 합니다. SCIM 같은 표준 프로토콜 지원 여부는 도입 직전에 확인하시기 바랍니다. 또한 「개인정보 보호법」 제29조 안전조치 의무에 따라 접속 기록 1년(고유식별정보·민감정보 2년) 보존이 강제되므로, 인증 로그 보관 정책을 GoTrue 운영 절차에 명시해야 합니다.

AFFiNE 은 자체 인증과 외부 OIDC 연동을 모두 지원하는 것으로 알려져 있지만 EE 기능 의존 여부 [S5] 가 명확하지 않습니다. CE 만으로 사내 SSO 통합이 가능한지를 LICENSE 와 별도로 기능 문서에서 확인해야 합니다. 두 번째 차단 요인은 외부 SaaS 의존입니다. 세 도구 모두 핵심 기능은 사내에서 완결되지만, AI 기능이나 임베드 콘텐츠 미리보기 같은 부가 기능이 외부 API 를 호출하는 사례가 있으므로 망분리 환경에서는 해당 기능을 사전에 비활성화하는 설정이 필요합니다. CSAP 인증을 받은 국내 클라우드(KT Cloud·NHN Cloud·네이버 클라우드 등) 상의 폐쇄망 VPC 에 배포하는 시나리오라면 외부 호출 차단 정책을 보안그룹 레벨에서 함께 잠가두는 편이 안전합니다.

세 번째 차단 요인은 컨테이너 이미지의 출처입니다. 망분리 환경에서는 Docker Hub 직접 접근이 차단되는 경우가 많으므로 사내 컨테이너 레지스트리(Harbor·Quay·국내 클라우드 사업자 레지스트리)에 이미지를 미러링하는 절차를 미리 준비해야 합니다. 이미지 해시 검증과 Trivy·Grype 같은 도구로 취약점 스캔을 미러링 과정에 포함시키면 ISMS-P 인증 심사 대응이 용이합니다.

8.3 한국어 지원·검색 품질·모바일 성숙도

한국어 지원은 도입 후 사용자 만족도를 좌우하는 결정적 요소입니다. UI 번역은 표면적인 지표일 뿐이고, IME 입력 안정성과 검색의 형태소 분석 품질이 실제 사용 경험을 결정합니다. 본 절은 공식 문서로 검증하기 어려운

영역이므로 사용자 보고와 이슈 트래커 확인 한정의 정보임을 미리 밝힙니다.

8.3.1 한국어 IME 입력 안정성과 전체 텍스트 검색의 형태소 분석 품질

한국어 IME 입력 안정성은 조합 중인 글자가 중복 입력되는 현상, 자모가 분리되어 표시되는 현상, 한자 변환 후 커서 위치가 어긋나는 현상 등으로 평가합니다. 본 백서 작성 시점에서 세 도구 모두 공식적인 한국어 IME 지원 명세를 공개하지 않았으므로 GitHub 이슈 트래커와 사용자 커뮤니티 보고를 한정적으로 참고할 수밖에 없습니다. 도입 전 PoC 단계에서 실제 한글 문서 1만 자 작성 테스트를 거치시기를 권고합니다.

블록 기반 에디터의 특성상 한 줄 단위로 블록이 분리되므로 줄바꿈 시점에 IME 조합이 깨지는 사례가 보고되는 경향이 있습니다. AppFlowy 와 AFFiNE 은 모두 블록 기반이므로 이 위험에 노출되어 있고, Outline 은 ProseMirror 기반의 보다 전통적인 에디터를 사용하여 상대적으로 안정적이라는 사용자 보고가 있습니다. 다만 이는 한정된 사용자 표본의 의견이므로 자체 검증이 필수입니다. PoC 단계에서는 macOS·Windows·웹 브라우저(Chrome/Edge/Safari) 환경에서 두벌식·세벌식 입력기와 macOS 한글 입력기를 교차 테스트하여 자모 분리·중복 입력·한자 변환 동작을 모두 점검하시기 바랍니다.

한글 폰트 렌더링도 사용자 만족도의 핵심 변수입니다. 세 도구 모두 시스템 폰트를 기본으로 사용하므로 Windows에서는 맑은 고딕, macOS에서는 Apple SD Gothic Neo 가 적용되어 가독성 자체는 무난합니다. 다만 사내 브랜드 일관성을 위해 Pretendard(국내 오픈 라이선스 한글 폰트) 또는 Noto Sans KR 를 셀프호스팅 자산으로 함께 배포하는 패턴이 늘고 있으며, 세 도구 모두 CSS 사용자 정의가 가능하므로 사내 폰트 적용은 운영 단계에서 어렵지 않습니다.

전체 텍스트 검색의 형태소 분석 품질은 한국어 사용자 경험의 또 다른 결정 항목입니다. PostgreSQL 의 기본 전체 텍스트 검색은 영문 어간 추출(stemming) 기반이므로 한국어에는 적용되지 않으며, "협업"으로 검색했을 때 "협업하는"이 결과에 포함되지 않는 한계가 있습니다. 한국어 형태소 분석을 위해서는 PostgreSQL 의 mecab-ko 확장 또는 별도 검색 엔진(Elasticsearch + Nori 분석기, OpenSearch + Seunjeon) 연동이 필요합니다. Nori 는 Elastic 공식 한국어 분석기이고 Seunjeon 은 국내 커뮤니티에서 활발히 유지되는 대안입니다.

세 도구 모두 공식적으로 한국어 형태소 분석 통합을 지원한다고 명시하지 않았으므로 검색 정확도는 N-gram 분할 또는 부분 문자열 매칭에 의존할 가능성이 높습니다. 의사결정권자께서는 도입 전 PoC 에서 실제 사내 문서 1,000건을 색인한 뒤 한국어 키워드 10개로 정밀도와 재현율을 측정하시기 바랍니다. 가능하다면 사내 회의록·기술 문서·고객 응대 매뉴얼처럼 어휘 분포가 다른 세 종류 코퍼스를 모두 포함시키는 편이 평가의 일반화 가능성을 높입니다.

8.3.2 모바일 앱 성숙도와 오프라인 동기화 지원

모바일 앱 성숙도는 외근이 많은 조직이나 임원 사용자가 많은 조직에서 도입 만족도를 결정합니다. AppFlowy 는 Flutter 기반으로 iOS·Android 네이티브 앱을 제공하며 GitHub 저장소 [S1] 에서 모바일 빌드 디렉터리가 활발히 관리됩니다. 오프라인 우선(local-first) 설계 [S3] 의 영향으로 네트워크 단절 시에도 편집이 가능한 점이 강점입니다.

AFFiNE 도 모바일 앱을 제공하며 Electron 기반 데스크톱 앱과 동일한 코드베이스 [S6] 를 공유하는 것으로 알려져 있습니다. canary 릴리스 채널 [S4] 에서 모바일 관련 변경이 잦은 점은 활발한 개발의 신호이기도 하지만 안정 채널 사용자에게는 모바일 기능이 늦게 도달한다는 트레이드오프가 있습니다.

Outline 은 공식 모바일 네이티브 앱 대신 모바일 웹 뷰를 권장하는 정책으로 알려져 있습니다 [S7]. 모바일 브라우저에서 PWA 형태로 사용하는 방식이 가능하지만 푸시 알림이나 오프라인 편집 같은 네이티브 기능은 제한적입니다. 외근이 적고 데스크톱 사용이 주력인 조직이라면 문제가 되지 않지만, 모바일 사용 비중이 높은 조직이라면 보수적으로 평가해야 합니다.

오프라인 동기화는 AppFlowy 의 local-first 아키텍처가 가장 앞서 있고, AFFiNE 은 CRDT 기반 동기화를 자체적으로 구현하여 부분적인 오프라인 지원을 제공합니다. Outline 은 서버 의존이 강한 구조이므로 오프라인 편집은 제한적입니다. 도입 직전에 실제 모바일 단말로 30분 비행기 모드 테스트를 거쳐 동기화 충돌 처리 방식을 검증하시기 바랍니다.

8.4 오픈소스 활성도 GitHub 지표 실측 비교

오픈소스 활성도는 장기 운영 가능성을 가늠하는 가장 객관적인 지표입니다. GitHub Star 수는 인지도의 대리 지표이고, 릴리스 주기는 개발 속도의 지표이며, 기여자 수는 단일 회사 의존도의 지표입니다. 본 절은 2026년 6월 시점의 실측 데이터를 기반으로 비교합니다.

8.4.1 GitHub Star·릴리스 주기·기여자 수·이슈 해소율 정량 비교

2026년 6월 시점의 GitHub Star 수는 AppFlowy 가 약 72,000개 [S1], AFFiNE 이 약 70,000개 [S4], Outline 이 약 38,800개 [S7] 입니다. 상위 두 도구는 인지도가 거의 동등하며 Outline 은 그 절반 수준입니다. Star 수만 놓고 보면 AppFlowy 와 AFFiNE 이 우세하지만, Outline 은 2018년부터 운영되어 온 상대적으로 성숙한 프로젝트이므로 Star 증가 속도와 절대값을 함께 봐야 합니다.

릴리스 주기를 보면 AFFiNE 은 canary 채널에서 거의 매일 릴리스를 발행합니다. 본 백서 작성 시점의 최신 canary 는 v2026.6.7-canary.1000 [S4] 으로 빌드 번호가 1,000번대에 도달했습니다. Outline 은 안정 채널 v1.8.1 [S7] 을 운영하며 수 주에서 수 개월 단위의 릴리스 주기를 가집니다. AppFlowy 도 안정 채널 중심의 릴리스 정책을 채택하여 분기에 1~2회 메이저 릴리스를 발행하는 경향을 보입니다.

기여자 수와 이슈 해소율은 단일 회사 의존도를 판단하는 지표입니다. 세 도구 모두 단일 모회사(AppFlowy.IO, Toeverything, General Outline)가 주도하는 구조이지만 외부 기여자 비율은 도구마다 다릅니다. GitHub Insights 의 contributor 그래프와 PR 처리 시간 통계를 직접 확인하시기 바랍니다. 도입 직전 시점의 실측이 백서 작성 시점의 데이터보다 의사결정에 유효합니다.

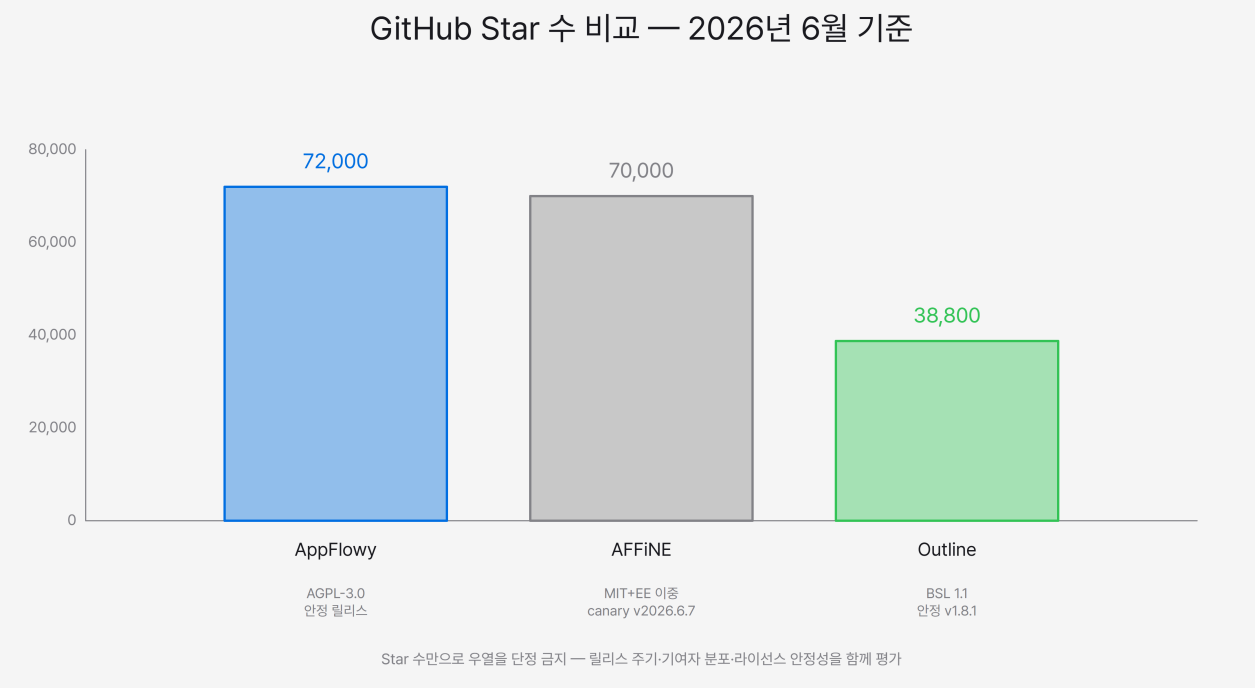

그림 8-1. AppFlowy·AFFiNE·Outline GitHub Star 추이 비교 (2024~2026, 분기별). 의사결정권자께서 세 도구의 Star 성장 곡선과 절대값을 한 화면에서 비교하여 장기 운영 가능성과 커뮤니티 모멘텀을 직관적으로 파악하실 수 있도록 합니다. AppFlowy 와 AFFiNE 의 상위 그룹과 Outline 의 안정 그룹 사이의 격차를 시각화합니다.

8.4.2 활성도 해석과 장기 운영 위험 진단

GitHub Star 수가 높다고 해서 장기 운영이 보장되는 것은 아닙니다. 인지도가 높은 프로젝트일수록 인수합병이나 라이선스 변경 같은 결정의 영향이 크기 때문입니다. AFFiNE 의 canary 채널 활발한 릴리스 [S4] 는 개발 속도가 빠르다는 긍정적 신호이지만 동시에 안정 채널이 상대적으로 늦게 따라온다는 신호이기도 합니다. 엔터프라이즈 도입이라면 안정 채널의 릴리스 주기를 별도로 추적해야 합니다.

Outline 은 안정 채널 중심 [S7] 의 보수적인 릴리스 정책을 채택하여 운영 안정성이 강점입니다. Star 수가 상위 두 도구의 절반 수준이지만 BSL 라이선스 [S8] 의 명확한 상업적 보호 장치가 모회사의 지속 가능성을 보장하는 면도 있습니다. 활성도 지표가 낮다고 위험이 큰 것은 아니며, 오히려 변동성이 낮은 안정적인 선택지로 해석할 수도 있습니다.

AppFlowy 는 AGPL-3.0 [S3] 의 강한 카피레프트 조항이 외부 기여자 유입과 동시에 상업적 통제력을 유지하는 균형점을 만듭니다. Star 수가 상위 그룹에 속하고 모바일 지원이 가장 앞서 있다는 강점이 있지만, AGPL 의 영향으로 일부 기업이 도입을 꺼리는 면도 고려해야 합니다. 장기 운영 위험은 라이선스 조항과 활성도를 함께 평가해야 정확하게 진단됩니다.

종합하면 세 도구 모두 단기 도입에는 적합한 수준의 활성도를 갖추었습니다. 다만 3년 이상의 장기 운영을 전제로 한다면 활성도 지표를 분기 단위로 재측정하고 라이선스 조항 변경 가능성을 함께 모니터링하는 체계를 갖추시기를 권고합니다. 본 장에서 제시한 네 가지 점검 항목 — 라이선스, 운영 복잡도, 한국어 지원, 오픈소스 활성도 — 는 도입 직전과 도입 후 1년 시점에 각각 재평가하실 수 있는 체크리스트로 활용하시기 바랍니다.

9장: 조직 유형별 선택 가이드와 기술적 우위 요약

여러분께서 1장부터 8장까지 함께 검토해 오신 내용을 한 자리에 모아 결론을 도출할 차례입니다. 이 문서는 Notion 의존을 줄이고자 하는 조직이 셀프호스팅 가능한 오픈소스 협업 도구를 선택할 때 어떤 기준으로 판단해야 하는지를 다루어 왔습니다. AppFlowy, AFFiNE, Outline 세 도구는 모두 Notion 의 일부 가치를 대체할 수 있으나 그 방식과 강점이 서로 다릅니다. AppFlowy 는 Notion UX 를 가장 충실하게 재현하면서 AGPL-3.0 단일 라이선스를 채택해 완전 오픈소스 원칙을 지키고 있으며 [S1][S3], AFFiNE 은 페이지와 무한 캔버스를 결합한 하이브리드 편집 모델 위에 AI 와 MCP 통합을 적극적으로 확장하고 있고 [S4][S10], Outline 은 팀 위키와 SSO 거버넌스 그리고 공식 MCP 기반 RAG 응대를 무기로 합니다 [S7][S9].

2장과 3장에서 라이선스와 거버넌스 모델을 살펴보았고, 4장에서는 Claude(LLM) 연동 가능성을, 5장에서는 Markdown 자산 연속성을, 6장에서는 칸반·실시간 협업·지식 구조화 실무 기능을, 7장에서는 Notion 이전 경로를, 8장에서는 기업 도입 전 점검 항목을 정리했습니다. 9장은 이 모든 분석을 11개 평가 기준으로 응축한 종합 비교표를 제시하고, 조직 유형별로 어느 도구가 적합한지 판단 기준을 제시한 뒤, 세 도구의 기술적 우위를 한 줄로 압축해 마무리합니다. 도입 후 단계별 행동 계획은 본 백서의 범위 밖이며, 각 조직의 변경 관리 역량과 인프라 사정에 따라 다르게 결정되어야 한다는 점을 미리 밝혀 둡니다.

본 장에서 강조하고 싶은 결론은 "단일 정답은 없다" 입니다. 세 도구 모두 활발히 개발되고 있고 GitHub Star 가 각각 약 72,000 (AppFlowy), 약 70,000 (AFFiNE), 약 38,800 (Outline) 수준에 이를 만큼 커뮤니티 신뢰를 확보하고 있습니다 [S1][S4][S7]. 따라서 의사결정권자께서는 "어느 것이 가장 좋은가" 가 아니라 "우리 조직이 가장 중요하게 여기는 평가 기준은 무엇인가" 를 먼저 정의하셔야 합니다. 11개 축의 가중치를 어떻게 부여하느냐에 따라 결론이 달라지며, 이 문서는 그 의사결정을 돕는 비교 프레임을 제공할 뿐 강제하지 않습니다. CNCF 의 벤더 중립 원칙에 따라 이 문서는 특정 도구를 우월한 것으로 결론짓지 않고 세 도구의 기술적 특성을 가능한 한 사실에 기반해 정리하는 데 집중합니다.

또한 본 장은 의사결정권자뿐 아니라 정책결정권자, IT 담당자, 엔지니어가 함께 읽을 수 있도록 구성되어 있습니다. 9.1 절은 종합 비교표를 통해 한눈에 비교할 수 있게 했고, 9.2 절은 조직 유형별 시나리오로 풀어 설명했으며, 9.3 절은 각 도구의 기술적 우위를 응축했습니다. 여러분의 조직 상황을 9.2 절의 시나리오와 대조하시면 적합한 후보군을 좁히실 수 있습니다.

9.1 11개 평가 기준 종합 비교표

본 절은 1장부터 8장까지의 분석 결과를 11개 평가 기준으로 응축한 종합 비교표를 제시합니다. 11개 축은 라이선스, 거버넌스 모델, Markdown 의 위상, 데이터 모델, AI 통합, MCP 지원, 셀프호스팅 운영 부담, 보안 거버넌스, 커뮤니티 신뢰도, 릴리스 안정성, 온톨로지화 친화성으로 구성됩니다. 각 축은 정량 지표 또는 정성 평가로 표시되었으며, 출처가 명확한 항목은 인용 표기를 함께 제공합니다.

9.1.1 라이선스·운영·Markdown·AI·온톨로지화 등 11개 기준 매트릭스

다음 표는 세 도구를 11개 평가 기준에 따라 정리한 매트릭스입니다. 표는 절대 평가가 아니라 비교 평가이며, 여러분 조직의 우선순위에 따라 해석이 달라질 수 있습니다.

평가 기준	AppFlowy	AFFiNE	Outline
라이선스	AGPL-3.0 단일 [S3]	MIT + EE 이중 디렉터리 [S5][S6]	BSL 1.1 (2030년 Apache 2.0 전환) [S8]

평가 기준	AppFlowy	AFFiNE	Outline
거버넌스 모델	단일 OSS 라이선스, 커뮤니티 중심 [S1][S3]	디렉터리별 이중 라이선스, Toeverything 주도 [S4][S5]	BSL 시한부, Outline 팀 주도 [S7][S8]
Markdown 위상	보조 (DB 우선, Markdown 가져오기·내보내기) [S1]	페이지 모드 1차급, edgeless 와 하이브리드 [S4]	1차 저장 포맷 (`.md` 네이티브) [S7]
데이터 모델	블록 + 데이터베이스 우선 [S1]	페이지 + 무한 캔버스 하이브리드 [S4]	위키 문서 트리 + Markdown 파일 [S7]
AI 통합	커뮤니티 MCP 9 도구 [S11]	서드파티 affine-mcp-server v1.6.0 [S10]	공식 MCP + 내장 AI [S9]
MCP 지원	비공식 (커뮤니티) [S11]	서드파티 (활성 어댑터) [S10]	공식 (벤더 안내) [S9]
셀프호스팅 운영 부담	보통 (PostgreSQL·Redis·MinIO·GoTrue 4 종) [S2]	보통 (PostgreSQL 중심, canary 시 ↑) [S4][S6]	낮음~보통 (PostgreSQL·Redis·S3·외부 OIDC IdP) [S7]
보안 거버넌스	OSS 표준, SSO 는 별도 구성, GoTrue 내장 [S2]	EE 영역 분리, 사내 운영 시 EE 조건 검토 필요 [S5]	외부 OIDC IdP 필수, SSO 일관성 강함 [S7]
커뮤니티 신뢰도 (GitHub Star)	약 72,000 [S1]	약 70,000 [S4]	약 38,800 [S7]
릴리스 안정성	안정 릴리스 중심	canary v2026.6.7-canary.1000 등 빠른 사이클 [S4]	v1.8.1 안정 우선 [S7]
온톨로지화 친화성	데이터베이스 스키마 강함 [S1]	캔버스 시각 온톨로지화 강함 [S4]	Markdown 평문 기반 RAG 친화 [S7][S9]

표를 읽으실 때 주의하실 점은 "공식 MCP" 와 "서드파티 MCP", "커뮤니티 MCP" 의 차이입니다. Outline 은 공식 도메인(docs.getoutline.com)이 MCP 안내를 명문화하여 책임 범위가 비교적 명확한 반면 [S9], AFFiNE 은 활성도 있는 서드파티 어댑터에 의존하며 [S10], AppFlowy 는 도구 9개의 좁은 표면적을 가진 커뮤니티 어댑터에 의존합니다 [S11]. 이 차이는 보안 감사를 받아야 하는 조직에 의미가 큽니다.

또한 라이선스 축에서 AGPL-3.0, MIT+EE 이중, BSL 1.1 은 세 가지가 모두 다른 의무를 부과합니다. AGPL-3.0 은 네트워크 서비스로 제공할 때도 소스 공개 의무가 발생하며 [S3], BSL 1.1 은 2030년 6월 6일 Apache 2.0 으로 자동 전환되는 시한부 구조 [S8] 를 가지고, AFFiNE 의 MIT+EE 이중은 packages/backend·packages/common/native 디렉터리 안의 코드가 프로덕션 사용 시 좌석 수 기준 구독을 요구합니다 [S5][S6]. 각 라이선스가 여러분 조직의 사용 시나리오와 충돌하는지 법무 검토를 거치셔야 합니다.

마지막으로 릴리스 안정성 기준은 운영 부담과 직결됩니다. AFFiNE 의 canary 사이클은 새 기능 도입이 빠른 대신 운영자가 잦은 검증 작업을 부담해야 하며 [S4], Outline 의 안정 릴리스 위주 정책은 변경 빈도가 낮아 운영이 평이합니다 [S7]. AppFlowy 는 중간 정도의 릴리스 빈도를 보입니다.

9.1.2 평가 기준별 가중치 적용 시 결론이 어떻게 달라지는가

같은 11개 기준이라도 가중치를 어떻게 부여하느냐에 따라 결론이 크게 달라집니다. 본 항에서는 세 가지 대표 시나리오를 통해 가중치의 영향을 보여 드립니다. 여러분께서는 각 시나리오 중 어느 쪽이 자기 조직 상황과 가장 가까운지 확인하실 수 있습니다.

첫 번째 시나리오는 "AI 우선" 가중치입니다. AI 통합, MCP 지원, 온톨로지화 친화성 세 축에 높은 가중치를 부여하면 공식 MCP 안내를 제공하는 Outline 과 활성 서드파티 MCP 를 가진 AFFiNE 이 앞서며 [S9][S10], 그 중에서도 페이지·캔버스 하이브리드 모델로 시각 온톨로지를 자연스럽게 구축할 수 있는 AFFiNE 이 우위에 섭니다 [S4]. Outline 은 Markdown 평문 기반 RAG 응대 품질이 강점이므로 텍스트 중심 AI 사용에 적합합니다 [S7][S9]. AppFlowy 는 커뮤니티 MCP 의존이라는 점에서 이 시나리오의 가중치에는 불리합니다 [S11].

두 번째 시나리오는 "라이선스 우선" 가중치입니다. 완전 오픈소스를 조직 정책으로 정한 경우 AGPL-3.0 단일 라이선스를 채택한 AppFlowy 가 가장 명료한 선택입니다 [S3]. BSL 1.1 의 시한부 제한을 4년만 견디면 Apache 2.0 으로 자동 전환된다는 점은 Outline 의 강점이며 [S8], MIT 영역과 EE 영역이 디렉터리로 갈리는 AFFiNE 은 EE 좌석 비용을 운영 모델에 반영해야 하므로 상대적으로 불리합니다 [S5]. 라이선스 우선 시나리오에서는 AppFlowy 가 가장 직관적입니다.

세 번째 시나리오는 "운영 부담 우선" 가중치입니다. 인프라 운영 인력이 적은 조직은 안정 릴리스 사이클을 갖춘 Outline 이 운영 변경 부담이 가장 적습니다 [S7]. 다만 외부 OIDC IdP 가 사전 조건이라는 점은 인력이 적은 조직에 추가 부담이 될 수 있습니다. AppFlowy 는 GoTrue 를 내장하여 망분리 환경에서 가장 단순하게 띄울 수 있고 [S2], AFFiNE 은 canary 사이클이 빨라 운영자가 변경 사항을 자주 추적해야 합니다 [S4]. 이처럼 같은 도구도 어느 축에 가중치를 두느냐에 따라 평가가 달라지므로, 결론에 앞서 평가 기준 가중치를 합의하시기를 권합니다.

9.2 조직 유형별 적합 도구 판단 기준

본 절은 조직 유형별로 어느 도구가 적합한지를 시나리오 형태로 정리합니다. 여러분의 조직이 어느 유형에 가까운지 점검하시면 후보군을 좁히실 수 있습니다. 단, 모든 조직은 고유의 제약을 가지고 있으므로 시나리오를 그대로 적용하지 마시고 자기 조직 변수를 반영해 재해석하셔야 합니다.

9.2.1 Notion UX·완전 오픈소스 우선 조직 — AppFlowy

Notion 의 사용자 경험에 익숙한 구성원이 많고, 완전 오픈소스 원칙을 조직 정책으로 정한 곳이라면 AppFlowy 가 가장 적합한 후보입니다. AppFlowy 는 Notion 의 블록 편집과 데이터베이스 기반 정보 구성 방식을 가장 충실하게 재현한 오픈소스 대안이며 [S1], 라이선스가 AGPL-3.0 단일이므로 디렉터리별 이중 라이선스의 EE 영역을 따로 검토할 필요가 없습니다 [S3]. GitHub Star 수가 약 72,000 으로 세 도구 중 가장 많아 커뮤니티 자산이 풍부합니다 [S1].

다만 AppFlowy 도입을 고려하실 때 두 가지 변수를 함께 고려하셔야 합니다. 첫째, AGPL-3.0 은 네트워크 서비스로 제공할 때도 소스 공개 의무가 발생할 수 있으므로, 외부 고객에게 SaaS 형태로 제공하는 시나리오라면

법무 검토가 필수입니다 [S3]. 둘째, MCP 가 커뮤니티 어댑터에 의존하므로 [S11] AI 기능 운영 책임이 조직 자체에 있게 됩니다. SSO 통합 역시 외부 IdP 와 별도 구성이 필요합니다.

전반적으로 AppFlowy 는 "Notion UX 와 완전 오픈소스의 결합" 이라는 가치 명제가 명확합니다. 여러분의 조직이 이 두 가치를 핵심으로 본다면 AppFlowy 가 가장 부합하는 선택입니다.

9.2.2 AI·MCP·캔버스 통합 우선 조직 — AFFiNE

AI 와 MCP 통합을 핵심 가치로 보고, 페이지와 무한 캔버스를 결합한 하이브리드 작업 환경을 선호하는 조직이라면 AFFiNE 이 우선 후보입니다. AFFiNE 은 페이지 모드와 edgeless 캔버스를 단일 데이터 모델로 통합해 텍스트 문서와 시각 다이어그램을 끊김 없이 오갈 수 있게 합니다 [S4]. 활성도 있는 affine-mcp-server v1.6.0 어댑터 [S10] 가 GraphQL·WebSocket·CRDT 위에 MCP 표준 인터페이스를 얹어 양방향 에이전트 시나리오를 가능하게 하며, canary v2026.6.7-canary.1000 [S4] 같은 빠른 릴리스 사이클이 새 기능 도입 속도를 보여 줍니다.

AFFiNE 도입 시 가장 주의하실 점은 packages/backend·packages/common/native 디렉터리의 EE 라이선스 사용 조건입니다 [S5]. 셀프호스팅으로 팀 협업 기능을 띄우는 순간 EE 영역 코드가 운영에 들어가며, 프로덕션 사용 시 좌석 수만큼 구독이 요구된다는 조항이 발동합니다. 운영 모델 설계 시 PoC 단계의 무료 사용 범위와 운영 전환 시 좌석 비용 산정을 반드시 사전 합의해 두셔야 합니다. 또한 canary 사이클은 운영자에게 잦은 검증 부담을 안기므로 [S4], 운영 인력 여력이 있고 변경에 적극적인 조직에 더 적합합니다.

AFFiNE 의 강점은 "AI 와 캔버스 통합을 통한 새로운 작업 방식" 입니다. 텍스트 위주 문서를 넘어 다이어그램, 화이트보드, AI 자동화가 결합된 환경을 원하시는 조직이라면 AFFiNE 의 가치 명제가 가장 잘 맞습니다.

9.2.3 팀 위키·SSO 거버넌스·AI RAG 우선 조직 — Outline

팀 위키 형태의 문서 구조, SSO 기반 사용자 거버넌스, 그리고 공식 MCP 기반 RAG 응대를 중시하는 조직이라면 Outline 이 적합합니다. Outline 은 Markdown 을 1차 저장 포맷으로 사용해 [S7] 평문 자산화가 자연스럽고, 공식 MCP 안내를 통해 내부 문서를 AI 에 안전하게 노출할 수 있습니다 [S9]. BSL 1.1 라이선스는 2030년 6월 6일 Apache 2.0 으로 자동 전환되는 시한부 구조 [S8] 라서 장기 운영 시 라이선스 부담이 사라진다는 점이 매력입니다.

Outline 선택 시 사전 조건이 있습니다. 첫째, 외부 OIDC IdP(예: Keycloak·Authentik 같은 오픈소스 또는 Okta·Azure AD 같은 상용)가 이미 운영되고 있어야 SSO 기능을 즉시 활용할 수 있습니다 [S7]. 망분리 환경에서는 사내에 IdP 를 별도 구축해야 하므로 도입 비용에 IdP 구축 비용을 포함시켜야 합니다. 국내 공공·금융권에서는 사내 LDAP/AD 를 OIDC 브리지(Keycloak 의 LDAP federation 기능 등)로 감싸는 패턴이 흔합니다. 둘째, GitHub Star 수가 약 38,800 으로 세 도구 중 가장 적지만 [S7] 안정성과 운영 단순성 면에서 충분히 검증되어 있습니다. 셋째, 칸반·캔버스 같은 부가 기능이 의도적으로 배제되어 있으므로 위키 외 워크플로우는 별도 도구로 분리해야 한다는 결정을 받아들여야 합니다.

Outline 의 가치 명제는 "팀 위키 + 거버넌스 + AI RAG 의 통합" 입니다. 문서 자산을 평문으로 관리하면서 SSO 와 감사 로그를 통해 통제력을 갖추고, RAG 기반 AI 응대까지 갖추고자 하시는 조직이라면 Outline 이 가장 적합한 균형점을 제공합니다.

9.3 세 도구의 기술적 우위 요약

본 절은 세 도구의 기술적 우위를 한 줄로 응축해 정리하고, 그 뒤에 CNCF 벤더 중립 원칙과 본 백서의 한계를 안내합니다. 한 줄 요약은 의사결정권자께서 임원 보고 자료에 인용하실 수 있는 형태로 제시했습니다.

9.3.1 AppFlowy·AFFiNE·Outline 각 도구의 기술적 우위 1줄 요약

AppFlowy 의 기술적 우위는 한 줄로 "Notion UX 충실 재현 + AGPL-3.0 단일 라이선스를 통한 완전 오픈소스 보장" 입니다 [S1][S3]. 블록 편집과 데이터베이스 중심 데이터 모델은 Notion 사용자의 학습 부담을 최소화하며 [S1], AGPL-3.0 은 디렉터리별 이중 라이선스의 EE 영역 검토 부담을 제거합니다 [S3]. 약 72,000 GitHub Star 의 커뮤니티 자산은 장기 운영 신뢰성의 강력한 신호입니다 [S1].

AFFiNE 의 기술적 우위는 한 줄로 "페이지·edgeless 캔버스 하이브리드 데이터 모델 + 활성 서드파티 affine-mcp-server 기반 AI 통합" 입니다 [S4][S10]. 텍스트 문서와 무한 캔버스가 동일 데이터 모델 위에서 작동해 시각 온톨로지화에 자연스럽고 [S4], GraphQL·WebSocket·CRDT 기반 MCP 어댑터를 통해 AI 통합의 양방향 시나리오가 열립니다 [S10]. 빠른 canary 릴리스 사이클은 새 기능 도입 속도의 강점입니다.

Outline 의 기술적 우위는 한 줄로 "Markdown 1차 저장 포맷 + 공식 MCP RAG + BSL 1.1 시한부에 따른 장기 라이선스 안정성" 입니다 [S7][S9][S8]. Markdown 평문 저장은 자산 이식성과 RAG 친화성을 동시에 보장하며 [S7], 공식 MCP 안내는 AI 통합 책임을 벤더가 일부 분담합니다 [S9]. 2030년 6월 6일 Apache 2.0 자동 전환은 4년 후 라이선스 제약이 사라진다는 장기 안정성의 신호입니다 [S8].

9.3.2 CNCF 벤더 중립 입장과 본 백서의 한계

이 문서는 CNCF 의 벤더 중립 원칙을 따라 특정 도구를 우월한 것으로 결론짓지 않습니다. 세 도구는 각각 다른 가치 명제와 다른 기술적 트레이드오프를 가지고 있으며, 어느 도구가 적합한지는 조직의 우선순위와 제약에 따라 달라집니다. 본 백서가 제시한 11개 평가 기준은 그 의사결정을 돕는 프레임일 뿐, 결론을 강요하는 도구가 아닙니다. 여러분께서는 자기 조직의 가치와 제약을 명확히 정의하신 뒤 본 백서의 비교 매트릭스를 참조하시기를 권합니다.

본 백서의 한계도 함께 밝힙니다. 첫째, 2026년 6월 시점의 정보를 기준으로 작성되었으며, 세 도구 모두 활발히 개발 중이므로 라이선스, 기능, MCP 지원 범위가 변경될 수 있습니다. 특히 AFFiNE 의 canary 릴리스 [S4] 와 Outline 의 안정 릴리스 [S7] 사이클이 다르므로 정기적인 정보 갱신이 필요합니다. 둘째, 이 문서는 셀프호스팅을 전제로 했으므로 SaaS 형 협업 도구와의 직접 비교는 다루지 않습니다. 셋째, 한국어 환경에서의 검색 품질, 한글 입력기 호환성, 폰트 렌더링 같은 지역화 요소는 별도 검증이 필요합니다.

마지막으로 본 백서가 향후 갱신될 필요성을 안내합니다. 세 도구의 MCP 사양 변경, 라이선스 정책 변경, 신규 AI 기능 추가 등이 발생하면 본 백서의 11개 평가 기준 매트릭스도 함께 갱신되어야 합니다. 의사결정권자께서는 본 백서를 출발점으로 삼되, 실제 도구 선정 직전에는 각 프로젝트의 최신 릴리스 노트와 공식 문서를 재확인하시기를 권합니다. 본 백서가 여러분의 합리적 의사결정에 보탬이 되기를 바라며 마무리하겠습니다.

Appendix References

본 백서가 본문에 인라인 `[S##]` 형태로 인용한 출처를 한곳에 모았습니다. 모든 항목은 2026년 6월 10일에 접속·확인하였습니다. 라이선스 조항과 활성도 지표는 분기 단위로 변동될 수 있으므로 도입 직전 재확인을 권합니다.

ID	제목	URL	게시일/버전	저자/주관	신뢰도
S1	AppFlowy 메인 저장소	https://github.com/AppFlowy-IO/AppFlowy	2026-05-16 (최종 업데이트 확인)	AppFlowy-IO	95
S2	AppFlowy-Cloud 저장소	https://github.com/AppFlowy-IO/AppFlowy-Cloud	2026-05-16 (v0.9.64 기준)	AppFlowy-IO	95
S3	AppFlowy LICENSE 원문 (AGPL-3.0)	https://github.com/AppFlowy-IO/AppFlowy/blob/main/LICENSE	—	AppFlowy-IO	95
S4	AFFiNE 메인 저장소	https://github.com/toeverything/AFFiNE	2026-06-07 (canary 최신)	Toeverything	95
S5	AFFiNE backend/server LICENSE 원문 (EE 조항)	https://github.com/toeverything/AFFiNE/blob/canary/packages/backend/server/LICENSE	—	Toeverything	95
S6	AFFiNE 메인 LICENSE 원문 (MIT + EE 디렉터리 경계 정의)	https://github.com/toeverything/AFFiNE/blob/canary/LICENSE	—	Toeverything	95
S7	Outline 메인 저장소	https://github.com/outline/outline	2026-06-06 (v1.8.1)	outline 팀	95
S8	Outline LICENSE 원문 (BSL 1.1)	https://github.com/outline/outline/blob/main/LICENSE	—	outline 팀	95
S9	Outline 공식 MCP 안내	https://docs.getoutline.com/	2026	getoutline.com	90

ID	제목	URL	게시일/버전	저자/주관	신뢰도
		s/guide/doc/mcp-6j9jtENNKL			
S10	affine-mcp-server (DAWNCR0W) — PulseMCP	https://www.pulsemcp.com/servers/dawncr0w-affine	2026	pulsemcp.com	70
S11	AppFlowy Cloud MCP (Jemo69) — Conare	https://conare.ai/marketplace/mcp/appflowy-cloud	2026	conare.ai	65

Appendix Glossary

본 백서의 본문에 등장한 영문 약어와 외래 전문 용어를 한국어 풀이와 함께 정리했습니다. 도입 회의에서 사외 변호사·법무·인프라 담당자와 용어 정의를 공유할 때 활용하실 수 있습니다.

용어	풀이
AGPL-3.0	GNU Affero General Public License v3. 네트워크 너머로 서비스를 제공할 때도 소스 코드 공개 의무가 발동되는 강한 카피레프트(copyleft) 라이선스
BSL 1.1	Business Source License 1.1. 지정된 Change Date까지 일부 사용을 제한하다가 그 날짜 이후 자동으로 더 관대한 오픈소스 라이선스(예: Apache 2.0)로 전환되는 시한부 라이선스
BlockSuite	AFFiNE 의 자체 편집기 프레임워크. 블록 단위 편집 모델 위에 무한 캔버스(edgeless)를 결합
CE+EE	Community Edition + Enterprise Edition. 한 저장소 안에서 디렉터리별로 다른 라이선스를 적용하는 이중 구조
CommonMark	Markdown 표준 명세. 다양한 구현체 사이의 호환성을 정의
CRDT	Conflict-free Replicated Data Type, 충돌 없는 복제 데이터 타입. 여러 사용자가 동시에 같은 문서를 수정해도 자동으로 병합되는 데이터 구조
CSAP	클라우드보안인증. 국내 공공기관용 클라우드 서비스 보안 인증 제도

용어	풀이
Docker	애플리케이션과 의존성을 컨테이너 단위로 묶어 동일 환경에서 실행할 수 있게 해 주는 도구
Docker Compose	여러 컨테이너를 한 파일로 정의하여 함께 실행·관리하는 도구
edgeless	AFFiNE 의 무한 캔버스 모드. 좌표 기반 자유 배치 블록(도형·프레임·연결선)을 다룸
GoTrue	AppFlowy-Cloud 에 내장된 자체 인증 서비스
GraphQL	API 쿼리 언어. 클라이언트가 필요한 데이터의 구조를 명시하여 정밀하게 조회
IdP	Identity Provider, 신원 제공자. SSO 환경에서 사용자 신원 인증을 담당하는 외부 시스템 (Keycloak·Authentik·Okta 등)
IME	Input Method Editor, 입력기. 한글·한자처럼 조합형 문자를 입력하는 소프트웨어
LLM	Large Language Model, 대형 언어 모델
MCP	Model Context Protocol, 모델 컨텍스트 프로토콜. Anthropic 이 공개한 표준 프로토콜로 AI 어시스턴트가 외부 시스템의 도구·자원·프롬프트를 일관된 인터페이스로 호출
MinIO	S3 호환 객체 스토리지 오픈소스
OIDC	OpenID Connect. OAuth 2.0 위에 구축된 표준 인증 프로토콜
OSI	Open Source Initiative. 오픈소스 라이선스 승인 단체
PoC	Proof of Concept, 개념 검증. 본격 도입 전 시험 환경에서 도구를 평가하는 단계
PostgreSQL	오픈소스 관계형 데이터베이스
ProseMirror	풍부한 텍스트 편집기 프레임워크. Outline 의 본문 편집기에 활용
PWA	Progressive Web App, 점진적 웹앱. 모바일 브라우저에서 네이티브 앱과 유사한 사용 경험을 제공
RAG	Retrieval-Augmented Generation, 검색 보강 생성. 외부 지식 기반에서 관련 문서를 검색해 LLM 응답을 보강하는 패턴

용어	풀이
Redis	인메모리 키-값 저장소. 캐시·세션 저장 용도로 자주 사용
REST API	Representational State Transfer API. HTTP 위에서 자원을 표현하는 API 양식
S3 호환 객체 스토리지	아마존 S3 API 와 호환되는 객체 저장소. MinIO·Ceph·Wasabi 등
SaaS	Software as a Service, 클라우드형 구독 서비스
SCIM	System for Cross-domain Identity Management. 사용자 자동 프로비저닝·디프로비저닝 표준
SSO	Single Sign-On, 단일 인증. 한 번의 로그인으로 여러 시스템 접근
Streamable HTTP	MCP 1.x 의 권장 트랜스포트. 단일 HTTP 엔드포인트 위에서 양방향 스트리밍을 지원
TCO	Total Cost of Ownership, 총소유비용. 구매·운영·이전 비용을 모두 포함한 비용 총합
WebSocket	양방향 실시간 통신을 위한 웹 표준 프로토콜
망분리	외부 인터넷과 사내망을 물리·논리적으로 분리하는 국내 공공·금융권 보안 정책

협업 도구는 더 이상 단순한 문서 편집기가 아닙니다.

2026-06-14 · 작성자: CNCF Korea Team · cncf.co.kr

Contact

커뮤니티 참여 / 표준 채택 협업

Web
cncf.co.kr
www.cncf.co.kr

Email
info@cncf.co.kr

Tel
+82-2-1670-1010
+82216701010

YouTube
@cncf
www.youtube.com/@cncf

LinkedIn
linkedin.com/company/cncf-korea
www.linkedin.com/company/cncf-korea

Facebook
facebook.com/cloudnativ
www.facebook.com/cloudnativ

CLOUD NATIVE FORUM