Outline 협업문서 시스템 — 자체 호스팅 협업 위키 도입 의사결정 가이드

Outline 은 자체 호스팅이 가능한

Outline 은 자체 호스팅이 가능한 오픈소스 협업 위키입니다. 사내에 직접 설치해 운영하는 지식 저장소이면서, 동시에 여러 사람이 한 문서를 실시간으로 함께 편집하는 협업 도구의 성격을 함께 갖습니다. 도입을 검토하는 조직 입장에서 먼저 확인할 것은 기능 목록이 아니라 이 도구가 어떤 배경에서 만들어졌고, 누가 어떤 구조로 유지하며, 어떤 라이선스 위에서 사내 운영을 허용하는가입니다.

2026-06-22 · 작성자: CNCF Korea Team · cncf.co.kr

목차

Outline AI 시대의 협업문서 시스템 — 자체 호스팅 협업 위키 도입 의사결정 가이드

1장: Outline 의 시작과 정체성

Outline 은 자체 호스팅이 가능한 오픈소스 협업 위키입니다. 사내에 직접 설치해 운영하는 지식 저장소이면서, 동시에 여러 사람이 한 문서를 실시간으로 함께 편집하는 협업 도구의 성격을 함께 갖습니다. 도입을 검토하는 조직 입장에서 먼저 확인할 것은 기능 목록이 아니라 이 도구가 어떤 배경에서 만들어졌고, 누가 어떤 구조로 유지하며, 어떤 라이선스 위에서 사내 운영을 허용하는가입니다. 이 세 가지가 분명해야 도입 후 수년에 걸친 운영 안정성을 사실 기반으로 판단할 수 있습니다.

분산된 문서 자산을 단일 저장소로 통합하려는 조직은 흔히 두 가지 운영 문제를 동시에 안고 있습니다. 하나는 자료가 여러 도구에 흩어져 검색·이력 관리가 끊긴다는 문제이고, 다른 하나는 도입한 도구의 라이선스나 공급 주체가 도중에 바뀌어 재이행 비용이 발생할 위험입니다. Outline 의 시작 배경과 회사 구조, 라이선스 변경 이력은 바로 이 두 번째 위험을 평가하는 1차 신호입니다. GitHub 저장소는 2016-05-22 에 생성되어 약 39,000 개 이상의 별을 모았고, 라이선스는 현재 Business Source License 1.1(BUSL 1.1)로 표기됩니다 [S50].

라이선스 구조가 도입 판단에 직접 영향을 줍니다. Outline 은 처음 BSD-3-Clause 로 공개되었다가 2021 년 BUSL 1.1 로 전환했고, 이 라이선스에는 2030-06-06 에 Apache License 2.0 으로 자동 전환된다는 조항이 들어 있습니다 [S1]. BUSL 은 외부에 동일 클라우드 서비스로 재판매하는 행위만 차단할 뿐 사내 자체 운영은 전면 허용합니다 [S5]. 도입 검토 단계에서 "OSI 승인 오픈소스가 아니다"라는 표면적 사실만 보고 반려하는 흐름을 막으려면, 허용 범위와 자동 전환 조항을 1차 출처로 확정해 두어야 합니다.

이 장은 그 1차 신호를 한 자리에 모읍니다. 2016 년 사이드 프로젝트로 시작해 2020 년 General Outline, Inc. 로 법인화된 시작 배경, 외부 자본에 의존하지 않는다는 회사의 자기 규정, BSD-3-Clause 에서 BUSL 1.1 을 거쳐 Apache 2.0 으로 이어지는 라이선스 timeline, 그리고 "communal long-term memory" 라는 제품 컨셉과 시장에서의 위치가 차례로 드러납니다. 이를 통해 의사결정권자는 라이선스 timeline 과 회사 지속가능성을 한 화면에서 확인하고 장기적으로 안전한 도입 방안을 검토할 수 있습니다.

1.1 시작 배경과 회사 정체성

2016 년 사이드 프로젝트의 시작. Outline 의 GitHub 저장소 `outline/outline` 은 2016-05-22 에 생성되었습니다 [S50]. 창업자 Tom Moor 는 본인이 외부 문서용으로 쓰던 지식 베이스와, 당시 Coinbase 에 있던 지인이 사내용으로 만들던 지식 베이스를 합쳐 사이드 프로젝트로 출발시켰습니다 [S49]. 처음부터 상용 제품을 목표로 설계한 것이 아니라, 실제 운영에서 부족했던 사내 지식 저장소의 필요가 코드로 이어진 사례입니다. 도입을 검토하는 조직 입장에서는 이 출발점이 곧 "이 도구가 실제 사용 현장의 문제에서 자랐는가"를 보여 주는 신호입니다.

규모 신호는 세 가지를 한 자리에서 보면 빠르게 잡힙니다. 저장소 생성일, 누적 별 수, 창업자 이력이 그것입니다. 저장소는 2016-05-22 에 생성되어 2026 년 시점 기준 약 39,000 개 이상의 별을 모았고, GitHub 라이선스 표기는 현재 "Other" 로 분류되는 BUSL 입니다 [S50]. 별 수는 단순한 인기 지표를 넘어, 자체 호스팅 위키 영역에서 커뮤니티가 지속적으로 관심을 유지하고 있다는 증거로 읽힙니다.

항목	값	출처
GitHub 저장소 생성일	2016-05-22	[S50]
누적 별 수	약 39,000+	[S50]
창업자	Tom Moor (외부·사내 지식 베이스를 합쳐 시작)	[S49]
GitHub 라이선스 표기	Other (BUSL 1.1)	[S50]

이 세 건을 함께 두는 이유는 명확합니다. 별 수만으로는 일시적 화제성과 지속적 채택을 구분하기 어렵고, 생성일만으로는 활동성을 알 수 없으며, 창업자 이력만으로는 운영 연속성을 보장하지 못합니다. 세 신호를 겹쳐 보면 2016 년부터 약 10 년에 걸쳐 같은 주체가 유지해 온 도구라는 그림이 한눈에 잡힙니다. 도입 검토에서 가장 먼저 확인해야 할 "이 도구가 얼마나 오래 살아남을 수 있는가"의 질문에 대해, 이 시작 배경은 긍정적인 1차 답을 제공합니다.

같은 신호는 GitHub 메타데이터로 누구나 같은 값을 재확인할 수 있다는 점에서도 의미가 있습니다. 저장소 생성일과 별 수, 라이선스 표기는 모두 공개 API 로 조회되는 값이므로, 도입 검토 자료에 인용한 수치를 검토자가 직접 검증할 수 있습니다 [S50]. 의사결정권자가 마케팅성 주장이 아니라 검증 가능한 1차 신호를 근거로 삼을 수 있다는 사실은, 사내 지식 저장소처럼 장기 운영을 전제하는 도구를 고를 때 신뢰의 출발점이 됩니다 [S49].

General Outline, Inc. 법인화와 회사 자기 규정. 사이드 프로젝트로 시작한 Outline 은 약 3 년의 오픈소스 개발을 거쳐 2020 년 초에 General Outline, Inc. 로 법인화되었습니다. 공식 About 페이지는 이 시점과 회사 운영 구조를 다음과 같이 직접 밝힙니다 [S3].

> *"founded at the beginning of 2020, building on three years of development on the open source project. It is bootstrapped, profitable, and not reliant on outside capital to survive."*

여기서 의사결정권자가 가중치를 크게 두어야 할 두 단어는 **bootstrapped, profitable** 입니다. 외부 벤처 자금에 의존하지 않고 자체 수익으로 운영되며 흑자를 유지한다는 뜻으로, 외부 투자 회수 압박에 따라 갑자기 라이선스를 닫거나 제품을 매각·종료할 위험이 상대적으로 낮은 구조를 가리킵니다. 국내 SaaS·SI 사가 사내 핵심 지식 저장소를 외부 도구에 의존할 때 가장 우려하는 지점이 바로 공급 주체의 갑작스러운 방향 전환인데, 외부 자본 비의존 구조는 그 위험을 줄이는 변수로 작동합니다.

벤더 리스크 평가 관점에서 이 자기 규정은 라이선스 조항과 짝을 이뤄 해석해야 합니다. 회사가 외부 자본에 의존하지 않는다는 점은 운영 연속성의 한 축이고, 뒤에서 다룰 BUSL 1.1 의

2030-06-06 Apache 2.0 자동 전환 조항은 공급 주체에 변동이 생기더라도 라이선스 자체가 더 개방적인 방향으로 고정되어 있다는 또 다른 축입니다 [S1]. 두 축을 함께 보면, 회사 차원의 안정성과 라이선스 차원의 안전장치가 서로를 보완하는 구조임을 확인할 수 있습니다.

이러한 운영 구조는 도입 후 수년 단위의 안정성 평가에 직접 연결됩니다. 외부 자본에 의존하지 않고 흑자로 운영된다는 사실은, 단기 성장 지표를 위해 무리한 기능 추가나 가격 정책 변경을 강행할 유인이 낮다는 의미로도 읽힙니다 [S3]. 의사결정권자는 이 자기 규정을 별도 해석 없이 1차 출처 그대로 법무·구매 검토 자료에 인용할 수 있습니다.

"communal long-term memory" 의 제품 컨셉. Outline 이 스스로를 어떤 도구로 규정하는지는 도입 후 이 도구를 조직 내 어디에 둘지를 정하는 멘탈 모델을 결정합니다. 공식 About 페이지는 제품의 가치를 다음과 같이 표현합니다 [S3].

> *"the value of Outline comes through sharing knowledge ... a sort of communal long-term memory."*

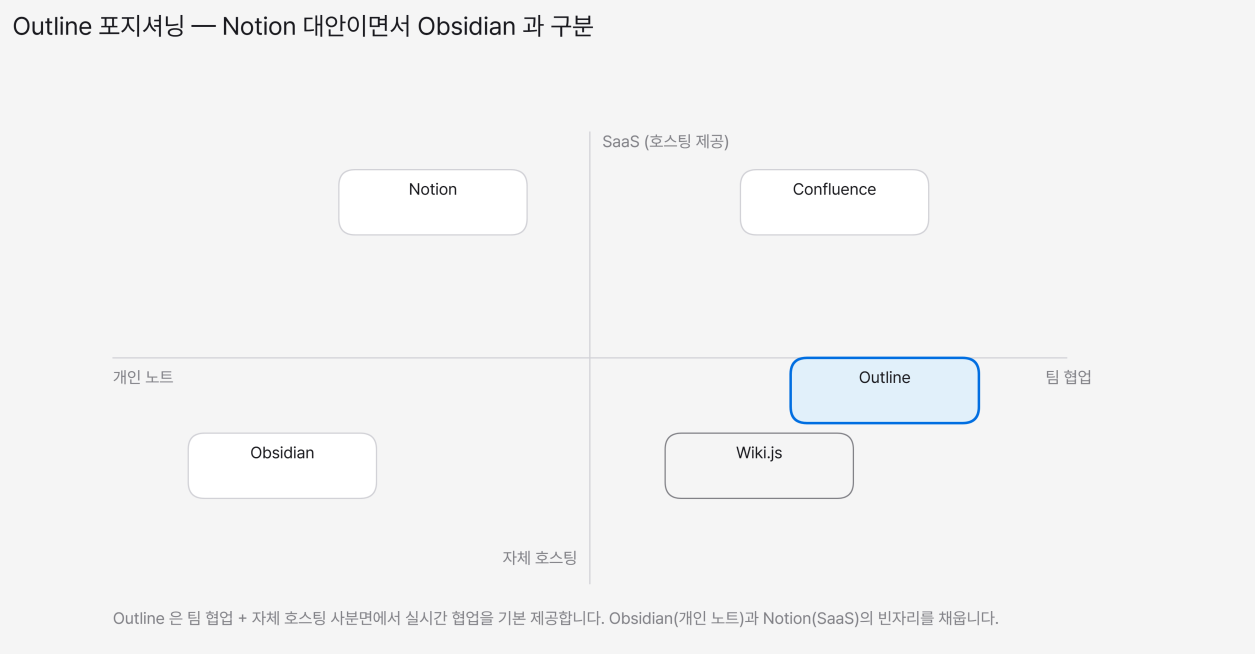
여기서 핵심은 "communal long-term memory" 라는 표현입니다. 개인이 혼자 메모를 쌓는 도구가 아니라, 조직 구성원이 지식을 공유하면서 함께 쌓아 가는 장기 기억 저장소라는 자기 규정입니다. 이는 곧 개인 노트 도구와는 다른 위치에 두어야 한다는 의미이기도 합니다. 회사가 내세우는 정의와 사용자 커뮤니티의 인식이 일치하는지를 함께 보면 이 컨셉의 신뢰도가 올라갑니다.

사용자 커뮤니티 쪽 인식도 같은 방향을 가리킵니다. 메인테이너 Tom Moor 는 Hacker News 토론(#39012054)에서 Outline 의 위치를 직접 다음과 같이 정리했습니다 [S4].

> *"a self-hosted collaborative alternative to Notion rather than a personal note taking tool like Obsidian."*

이 답글은 두 가지를 동시에 분명히 합니다. 첫째, Outline 은 Notion 의 자체 호스팅 협업 대안으로 설계되었다는 점입니다. 둘째, 개인 노트 도구인 Obsidian 과는 지향이 다르다는 점입니다. 회사 자기 규정의 "공유 기반 장기 기억" 과 메인테이너의 "자체 호스팅 협업 도구" 라는 외부 설명이 어긋나지 않고 한 방향으로 맞물립니다.

세 도구를 옆에 두면 포지셔닝이 한 문장으로 정리됩니다. Notion 은 클라우드 기반 협업 문서 도구, Obsidian 은 개인 중심 노트 도구, Confluence 는 엔터프라이즈 위키이며, Outline 은 이 중 "자체 호스팅이 가능한 Notion 형 협업 위키" 자리를 차지합니다 [S4]. 도입 검토 조직은 이 한 문장으로 "Outline 을 개인 메모 대체가 아니라 사내 공동 지식 저장소로 둔다" 는 멘탈 모델을 확정할 수 있습니다.



캡션: Notion 대안이면서 Obsidian 과 구분되는 Outline 의 자체 호스팅 협업 위키 포지셔닝.

세 도구를 가로축(개인 ↔ 협업)과 세로축(클라우드 전용 ↔ 자체 호스팅 가능)으로 배치하고, Outline 이 "협업 + 자체 호스팅" 사분면에 위치함을 표시합니다. Notion 은 클라우드 협업 쪽, Obsidian 은 개인 노트 쪽, Confluence 는 엔터프라이즈 협업 쪽에 두어 Outline 의 차별 위치를 시각적으로 분리합니다.

1.2 라이선스 timeline 과 BUSL 1.1 의 의미

BSD-3-Clause 에서 BUSL 1.1 로의 전환. Outline 의 라이선스는 처음부터 BUSL 이었던 것이 아닙니다. 초기 버전대인 v0.30.0 부터 v0.40.0 시기(2020-02 무렵)까지는 BSD-3-Clause 로 공개되어 있었습니다 [S1][S2]. BSD-3-Clause 는 OSI 승인 허가형 라이선스로 사실상 제약이 거의 없는 형태이며, 이 시기에 도구를 접한 사용자들은 Outline 을 전형적인 오픈소스로 인식했습니다.

전환 시점은 LICENSE 파일 커밋 이력에서 확인됩니다. 2021-09-11 의 "Update LICENSE" 커밋에서 라이선스가 BUSL 1.1 로 변경되었습니다 [S2]. 정확한 Release 태그는 v0.41.0 과 v0.48.0 사이로 좁혀지지만 단일 태그까지는 1차 출처로 아직 확정되지 않은 상태입니다 [S2]. 도입 검토 시에는 커밋 날짜와 커밋 메시지를 그대로 인용해 두는 편이 신뢰도 확보에 유리합니다.

항목	내용	출처
전환 전 라이선스	BSD-3-Clause (v0.30.0 ~ v0.40.0, 2020-02 무렵)	[S1][S2]
전환 커밋	"Update LICENSE" (2021-09-11)	[S2]
전환 후 라이선스	Business Source License 1.1	[S1]
정확한 Release 태그	v0.41.0 ~ v0.48.0 사이 (단일 태그 미확정)	[S2]

라이선스 전환 시점을 1차 출처로 확정하는 작업은 도입 검토에서 흔히 제기되는 "라이선스 회귀" 우려를 해소합니다. 즉 이미 한 차례 더 폐쇄적인 방향으로 바뀐 라이선스가 또 바뀌어 사내 운영까지 막히는 것 아니냐는 의문입니다. 이 우려에 대한 답은 다음 항에서 다룰 BUSL 의 실제 허용 범위와 자동 전환 조항에서 분명해집니다. 전환의 사실관계 자체는 커밋 이력에 남아 있어 누구나 같은 1차 출처로 재확인할 수 있습니다 [S2].

BUSL 1.1 의 사내 운영 허용 범위. BUSL 1.1 의 본문은 핵심 조건 네 가지를 명시합니다. Outline 의 LICENSE 파일은 다음과 같이 적습니다 [S1].

> *Change Date: June 6, 2030 / Change License: Apache License 2.0 / Licensor: General Outline, Inc. / License: Business Source License 1.1*

법무 검토 단계에서 자주 발생하는 오해는 "OSI 승인 오픈소스가 아님" 이라는 표면적 분류만 보고 사내 도입까지 막는 것입니다. 그러나 BUSL 의 실제 제약은 그보다 훨씬 좁습니다. 메인테이너 Tom Moor 는 Discussion #3301 에서 허용 범위를 직접 다음과 같이 밝혔습니다 [S5].

> *"The license allows all use except competing with the cloud offering."*

즉 차단되는 행위는 Outline 의 클라우드 서비스와 경쟁하는 형태, 다시 말해 외부 고객에게 동일 서비스를 SaaS 로 재판매하는 경우뿐입니다 [S5]. 그 외의 모든 사용, 특히 조직이 자체 인프라에 설치해 사내 구성원이 쓰는 자체 호스팅 운영은 별도 비용 없이 전면 허용됩니다. 사내 지식 저장소로서의 활용은 정확히 이 허용 범위 안에 들어갑니다.

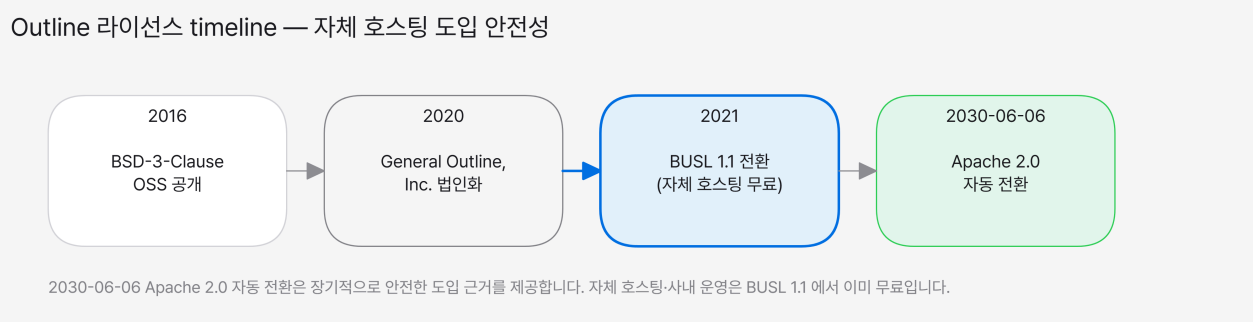
BUSL 은 시간 제한형 소스 공개 라이선스로, 일정 기간 상용 경쟁만 제약한 뒤 지정된 날짜가 지나면 더 개방적인 라이선스로 자동 전환되는 구조를 갖습니다. Outline 의 경우 그 제약 대상이 "클라우드 서비스 경쟁" 한 가지로 좁게 정의되어 있어, 사내 운영 조직이 실무에서 마주칠 제약은 사실상 없습니다 [S1][S5]. 법무 검토에서는 라이선스의 분류명이 아니라 본문에 적힌 허용·차단 조건을 직접 읽는 것이 정확한 판단으로 이어집니다.

구분	행위	허용 여부	출처
사내 자체 호스팅 운영	조직 내부 구성원이 설치·사용	허용 (무료)	[S5]
클라우드 재판매	외부 고객에게 동일 SaaS 로 제공	차단	[S5]
라이선스 자동 전환	2030-06-06 Apache License 2.0	자동 적용	[S1]

이 2 라인 비교가 정책결정권자의 법무 검토에 그대로 들어갈 결론입니다. 사내 운영은 허용되고 SaaS 재판매만 차단된다는 사실을 명확히 두면, "오픈소스가 아니므로 도입 불가" 라는 잘못된 해석으로 검토가 반려되는 패턴을 사전에 차단할 수 있습니다 [S5]. 라이선스 분류명이 아니라 실제 허용 행위를 기준으로 판단하는 것이 핵심입니다.

2030-06-06 Apache 2.0 자동 전환의 장기 도입 안전성. BUSL 1.1 의 가장 중요한 안전장치는 Change Date 조항입니다. Outline 의 LICENSE 는 2030-06-06 을 Change Date 로, Apache License 2.0 을 Change License 로 지정합니다 [S1]. 이는 별도 협상이나 라이선스 재계약 없이, 그 날짜가 지나면 해당 코드가 자동으로 Apache 2.0 으로 전환된다는 뜻입니다. Apache 2.0 은 OSI 승인 허가형 라이선스로 사내 운영은 물론 재배포·상용 활용에도 제약이 거의 없습니다.

이 조항이 의사결정권자에게 주는 의미는 장기적으로 안전한 도입 근거입니다. 현재 시점에서 사내 운영은 이미 BUSL 아래에서 무료로 허용되며, 2030-06-06 이후에는 라이선스 자체가 더 개방적인 Apache 2.0 으로 고정됩니다 [S1]. 설령 그사이 공급 주체에 변동이 생기더라도, 이미 공개된 코드의 자동 전환 조항은 소급해서 닫히지 않습니다. 도입 후 수년 단위의 운영을 전제하는 조직에게 이 자동 전환은 라이선스 리스크의 상한을 미리 정해 주는 장치로 작동합니다.



캡션: BSD-3-Clause → BUSL 1.1 → 2030-06-06 Apache 2.0 자동 전환으로 이어지는 라이선스 timeline 과 도입 안전성.

가로 시간축 위에 네 개의 마커를 배치합니다. 2016 년 오픈소스 공개, 2020-02 BSD-3-Clause(v0.30~v0.40), 2021-09-11 BUSL 1.1 전환 커밋, 2030-06-06 Apache License 2.0 자동 전환입니다. 여기에 "현재 시점" 과 "도입 검토 시점" 마커를 추가로 표시해, 도입 시점부터 Apache 자동 전환까지 남은 여유 기간을 한 화면에서 읽도록 합니다.

도표 D1 은 이 timeline 을 의사결정권자가 5 초 안에 해석하도록 압축합니다. 라이선스가 한 차례 더 폐쇄적인 방향으로 바뀐 과거 전환과, 앞으로 더 개방적인 방향으로 자동 전환될 미래 조항을 같은 축 위에 두면, 현재 도입 시점이 어느 구간에 있는지가 시각적으로 드러납니다 [S1] [S2]. 이 한 장이 라이선스 리스크 평가의 출발점이자 결론을 동시에 제공합니다.

1.3 자기 규정과 시장 포지셔닝

"Notion 대안 + Slack 채널 멘탈 모델" 의 외부 인식. 의사결정권자가 자료를 검색하다 보면 Outline 에 대한 외부 설명을 여러 경로에서 마주칩니다. 이때 메인테이너의 자기 포지셔닝과 사용자의 사용 경험 설명이 어긋나지 않는지를 먼저 확인해 두면 추가 검색 부담이 줄어듭니다. 두 인식을 같은 자리에 두고 비교하면 정합성이 분명해집니다.

메인테이너 측 정의는 앞서 인용한 Hacker News 답글에서 다시 확인됩니다 [S4].

> *"a self-hosted collaborative alternative to Notion rather than a personal note taking tool like Obsidian."*

사용자 측 인식은 같은 토론에서 다음과 같이 나타납니다 [S4].

> *"Outline's collections and Slack's channels are equivalent"*

이 사용자 평은 Outline 의 Collections(문서 묶음 계층)가 Slack 의 채널과 같은 멘탈 모델로 동작한다는 설명입니다. 즉 주제별로 채널을 나누듯 Collections 로 지식을 구획한다는 직관으로, 협업 도구를 써 본 조직 구성원이 별도 학습 없이 구조를 이해하도록 돕습니다. 메인테이너의 "Notion 대안" 정의와 사용자의 "Slack 채널 같은 구획" 인식은 모두 협업 중심 도구라는 한 방향을 가리킵니다 [S4]. 두 인식이 일치한다는 사실 자체가 의사결정권자에게 추가 교차 검증의 출발점이 됩니다.

의사결정권자의 교차 검증 경로 — 인접 자료 7 건. 단일 자료에만 의존하지 않고 여러 1·2 차 출처를 교차로 확인하는 것은 도입 검토의 기본 패턴입니다. 본 자료 이외에 의사결정권자가 직접 교차 확인할 만한 인접 자료를 한 자리에 모아 둡니다. 메인테이너 자기 진술, 공식 회사 페이지, 제3자 비교 분석, 자체 호스팅 큐레이션, 창업자 인터뷰가 고르게 포함되어 한쪽 시각에 치우치지 않도록 구성했습니다.

자료	성격	교차 확인 포인트	출처
Hacker News #39012054	메인테이너 직접 답글	"Notion 자체 호스팅 대안" 자기 포지셔닝	[S4]
Outline About 페이지	공식 1차 자료	법인화 시점·외부 자본 비의존·제품 컨셉	[S3]
Docmost 비교 블로그	경쟁 진영 2차 자료	BUSL 전환 이후 시장 인식	[S20]
MassiveGRID 비교	제3자 비교 분석	XWiki·BookStack 대비 협업·Slack 통합 우위	[S21]
selfh.st 대안 카탈로그	자체 호스팅 큐레이션	Notion 대안 후보 중 매칭도	[S22]
dev.to Wiki.js vs Outline	운영자 비교 글	UX·협업 대 단순함·무료의 선택 기준	[S23]
GitHub API 메타데이터	1차 정량 신호	생성일·별 수·라이선스 표기	[S49]

이 7 건을 함께 두는 이유는 한 자료의 주장을 다른 자료가 보강하거나 반증하도록 하기 위함입니다. 공식 About 페이지의 자기 규정은 Hacker News 메인테이너 답글과 맞물려 신뢰도가 올라가고 [S3][S4], 경쟁 진영인 Docmost 의 블로그는 BUSL 전환을 자기 강점으로 소구하면서도 그 전환 사실 자체를 외부에서 확인해 줍니다 [S20]. 제3자 비교 분석과 자체 호스팅 큐레이션은 Outline 의 협업·Slack 통합 강점을 동일한 방향으로 가리킵니다 [S21][S22][S23].

이러한 교차 검증 경로를 한 표로 압축해 두면, 의사결정권자는 본 자료의 결론을 외부 출처로 직접 재확인하면서 단일 자료 의존 위험을 줄일 수 있습니다. 메인테이너 진술, 공식 페이지, 경쟁 진영, 중립 비교, 정량 메타데이터가 모두 같은 그림을 가리킨다는 사실은, Outline 의 정체성과 라이선스 안전성에 대한 판단이 한 출처의 해석이 아니라 여러 출처의 합의에 근거함을 보여줍니다 [S3][S4][S20][S49].

2장: Markdown 호환 — 기술 아키텍처와 한계

Outline 의 도입 검토에서 가장 자주 오해되는 지점은 "Markdown 친화적인 도구"라는 직관입니다. 화면에서 보이는 편집 경험은 분명히 Markdown 문법을 받아들이지만, 실제 데이터가 어떤 형식으로 저장되는지는 그 직관과 다릅니다. 1 차 저장은 ProseMirror JSON 이며 Markdown 은 손실을 동반한 export(lossy export)일 뿐입니다 [S8][S9]. 이 차이는 단순한 구현 세부가 아니라, 기존 자산을 옮길 때의 손실과 추후 다른 도구로 다시 옮길 때의 export 보장을 동시에 결정하는 의사결정 변수입니다.

여기서 다루는 핵심 사실은 두 가지입니다. 하나는 아키텍처의 정량 구성입니다. 백엔드·프런트엔드·에디터·인프라 4 개 레이어가 각각 어떤 기술에 의존하는지를 1 차 소스로 제시해 자체 호스팅 운영의 복잡도를 가늠하게 합니다 [S6][S7][S10]. 다른 하나는 저장 모델입니다. `server/models/Document.ts` 의 세 컬럼이 각기 다른 표현형을 동시에 보관하며, 그 역할 분리가 Markdown round-trip 의 보장 범위를 결정합니다 [S8].

운영 관점에서 이 사실들은 두 가지 부담을 줄입니다. 첫째, 인프라가 PostgreSQL 과 Redis 두 컴포넌트로 끝나므로 별도 검색 엔진이나 메시지 브로커를 운영할 필요가 없습니다 [S10]. 둘째, export 형식이 명확히 정의되어 있어 도입 후 이탈(exit) 시나리오를 사전에 평가할 수 있습니다. Markdown export 가 손실을 동반한다는 사실을 미리 알면, 이행 단계에서의 가정 오류와 그로 인한 재작업을 피할 수 있습니다 [S9][S48].

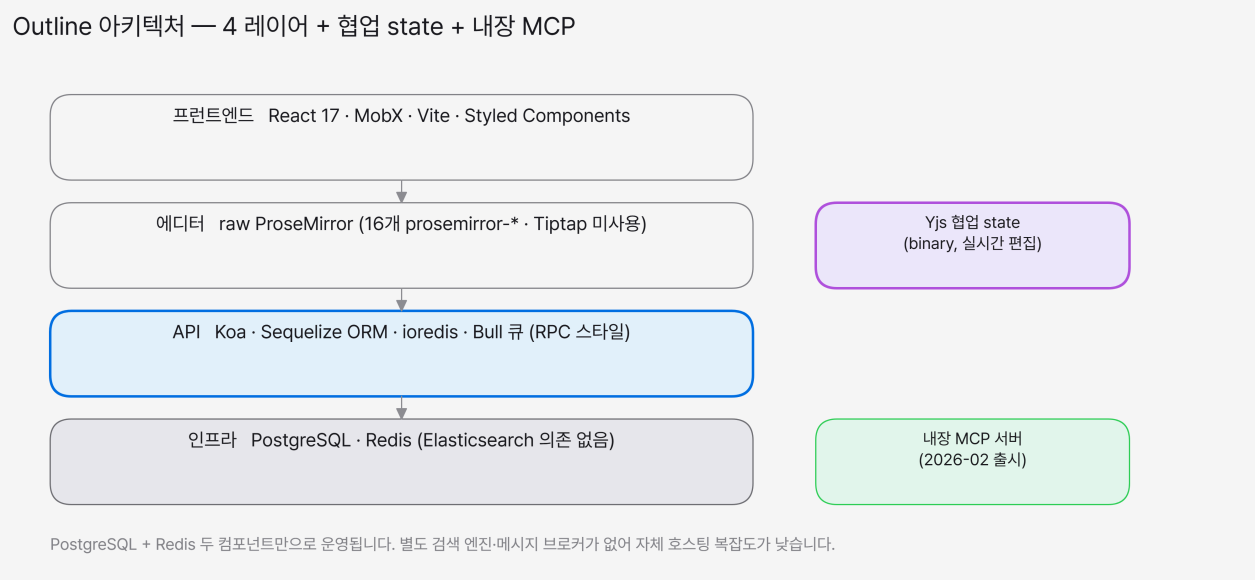
여기서 다루는 정보는 모두 공개된 소스코드와 공식 문서, 메인테이너의 1 차 답변에 근거합니다. 저장 컬럼 정의는 TypeScript 데코레이터를 그대로 인용하고, Markdown 저장 방침은 메인테이너 본인의 답글을 직접 인용합니다. 그래서 사내 엔지니어가 별도 추정 없이 호환성 경계를 확인할 수 있습니다.

2.1 아키텍처 개요 — Koa + React + ProseMirror + PostgreSQL + Redis

자체 호스팅 운영의 복잡도는 의존하는 컴포넌트의 수와 종류로 결정됩니다. Outline 은 백엔드·프런트엔드·에디터·인프라 4 개 레이어로 나뉘며, 각 레이어의 핵심 의존성을 1 차 소스로 확인할 수 있습니다. 공식 아키텍처 문서는 전체 구조를 한 문장으로 요약합니다. "Outline is built using Koa on the backend with a React frontend, communicating to the API which is RPC-style." [S7] 백엔드와 프런트엔드가 RPC 스타일 API 로 통신하는 단일 구조이며, 이 구조 위에 에디터와 저장소가 얹힙니다.

이 절의 목적은 의존성 목록을 사내 기술 표준과 즉시 대조하게 하는 것입니다. Node 런타임 버전, 백엔드 프레임워크, ORM, 큐 시스템, 프런트엔드 상태 관리, 에디터 라이브러리가 각각 무엇인지 확정되면 도입 검토팀은 학습 곡선과 운영 부담을 정량으로 추정할 수 있습니다. 잘못된

가정 — 예컨대 "에디터는 Tiptap 일 것"이나 "검색은 Elasticsearch 가 필요할 것" — 을 사전에 차단하는 것도 이 절의 역할입니다.



캡션: Outline 4 레이어 구성 — Koa(API) + React(MobX·Vite) + ProseMirror(에디터) + PostgreSQL(저장) + Redis(큐·캐시), 그리고 Yjs 협업 state 와 내장 MCP 서버.

이 다이어그램은 RPC 스타일 API 를 중심으로 백엔드(Koa·Sequelize·ioredis·Bull), 프런트엔드(React 17·MobX·Vite), 에디터(raw ProseMirror), 인프라(PostgreSQL·Redis)의 4 레이어가 어떻게 연결되는지를 한 화면에 보여줍니다. 별도 검색 엔진·메시지 브로커 노드가 없다는 점, 그리고 협업 state(Yjs)와 2026 년 추가된 내장 MCP 서버가 같은 코어에 붙는다는 점이 시각적으로 드러납니다.

백엔드 — Koa + Sequelize + ioredis + Bull

백엔드 런타임은 Node.js ≥ 20.12 이며, 프레임워크는 Express 가 아니라 Koa 입니다 [S6][S7]. 이 구분은 사내 표준이 Express 미들웨어 생태계에 맞춰져 있을 때 의미가 있습니다. Koa 는 async/await 기반 미들웨어 모델을 사용하므로, 사내 자작 미들웨어를 이식할 때 Express 와 동일하게 동작한다고 가정하면 오류가 발생합니다. ORM 은 Sequelize 이고, Redis 클라이언트는 ioredis, 비동기 작업 큐는 Bull 입니다 [S6].

이 네 가지를 한 줄로 정리하면 `Node.js ≥ 20.12 / Koa / Sequelize / ioredis + Bull` 입니다. 사내 표준 Node 버전이 20 미만이라면 런타임 업그레이드가 선행되어야 하고, ORM 표준이 Prisma 나 TypeORM 이라면 데이터 계층 운영 지식을 Sequelize 로 보강해야 합니다. 큐 시스템이 Bull 이라는 사실은 Redis 가 캐시뿐 아니라 작업 큐의 백엔드로도 쓰임을 뜻하므로, Redis 가용성이 Outline 의 비동기 작업(알림, 내보내기, 웹훅 발송 등) 전체에 영향을 줍니다 [S6][S10].

운영 부담의 핵심은 이 의존성들이 모두 성숙한 표준 컴포넌트라는 점입니다. Node·PostgreSQL·Redis 는 사내에서 이미 운영 중일 가능성이 높고, Koa 와 Sequelize 는 별도 데몬이 아니라 애플리케이션 프로세스 내부의 라이브러리입니다. 즉 추가로 띄워서 관리해야 하는 외부 서비스가 늘어나지 않습니다. 이것이 자체 호스팅 운영을 단순하게 만드는 첫 번째 구조적 이유입니다 [S7][S10].

백엔드와 프런트엔드가 RPC 스타일 단일 API 로 통신한다는 점도 운영 관점에서 중요합니다 [S7]. 클라이언트 앱이 호출하는 API 와 서버가 노출하는 API 가 동일하므로, 사내 자작 통합이나 마이그레이션 스크립트를 작성할 때 별도의 내부 API 와 외부 API 를 구분해 학습할 필요가 없습니다. 동일한 엔드포인트 집합이 화면과 외부 도구 양쪽에 그대로 쓰이며, 이는 사내 도구 연동의 진입 장벽을 낮춥니다 [S7].

프런트엔드 — React 17 + MobX + Vite + raw ProseMirror

프런트엔드는 React 17 기반이며, 상태 관리는 Redux 가 아니라 MobX 입니다 [S6]. 빌드 도구는 Vite, 스타일링은 Styled Components 를 사용합니다. 사내 프런트엔드 표준이 React + Redux 조합이라면, MobX 의 관찰 가능(observable) 상태 모델은 별도 학습이 필요합니다. Redux 의 단방향 흐름과 MobX 의 반응형 모델은 디버깅 방식과 상태 추적 패턴이 다르므로, 커스터마이징을 계획한다면 이 차이를 사전에 반영해야 합니다.

에디터 레이어에서 가장 자주 발생하는 오해는 Tiptap 가정입니다. Outline 은 Tiptap 을 사용하지 않습니다. 16 개의 `prosemirror-*` 패키지에 직접 의존하며, 커스텀 래퍼는 `shared/editor/` 아래에 자체 구현되어 있습니다 [S6][S7]. 이른바 raw ProseMirror 구성입니다. Tiptap 이 ProseMirror 위에 추상화 계층을 한 단계 더 얹어 학습 부담을 낮추는 도구라면, raw ProseMirror 는 그 추상화 없이 ProseMirror 의 스키마·노드·플러그인을 직접 다룹니다.

이 구분이 의사결정에 미치는 영향은 분명합니다. 에디터 동작을 사내 요구에 맞춰 확장하려는 조직은 ProseMirror 자체의 깊은 운영 지식이 필요하며, Tiptap 문서나 커뮤니티 패턴을 그대로 적용할 수 없습니다 [S6]. 반대로 에디터를 그대로 사용하는 조직에는 이 부담이 발생하지 않습니다. "Tiptap 아님"을 명시적으로 확인해 두면, 도입 후 확장 견적을 잡을 때의 가정 오류를 차단할 수 있습니다.

인프라 2 컴포넌트 — PostgreSQL + Redis

공식 `docker-compose.yml` 은 PostgreSQL 과 Redis 두 서비스만 의존합니다 [S10]. Elasticsearch, RabbitMQ, Meilisearch 같은 별도 검색 엔진이나 메시지 브로커 의존성은 없습니다 [S10]. 이 사실은 자체 호스팅 운영 복잡도 평가에서 가장 큰 단순화 요인입니다. 검색은 PostgreSQL 의 전문 검색(full-text search) 기능으로, 비동기 작업 큐는 Redis 위의 Bull 로 처리하므로, 운영팀이 추가로 관리해야 하는 상태 보유(stateful) 서비스가 두 개로 한정됩니다.

두 컴포넌트만으로 끝난다는 점은 사내 표준 docker-compose 운영 방식에 직접 들어맞습니다. PostgreSQL 은 데이터·검색 인덱스·문서 본문을 모두 보관하고, Redis 는 캐시·세션·작업 큐를 담당합니다. 이 구조에서는 백업 대상도 PostgreSQL 한 곳으로 사실상 수렴하므로, 재해 복구(DR) 설계와 백업 운영이 단순해집니다 [S10]. 별도 검색 엔진을 운영하지 않으니 인덱스 동기화·재색인·클러스터 헬스 모니터링 같은 부수 작업도 사라집니다.

다만 이 단순함에는 대가가 있습니다. 검색을 PostgreSQL FTS 에 의존하는 구조는 한국어 같은 비공백 언어의 토큰화에서 한계를 가지며, 이 문제는 별도 장에서 소스코드 수준으로 다룹니다. 인프라 단순함과 검색 품질은 같은 설계 선택의 양면입니다. 도입 검토 단계에서는 "2 컴포넌트로 운영이 단순하다"는 장점과 "검색 품질은 별도 우회가 필요할 수 있다"는 한계를 함께 기록해 두는 것이 정확한 평가입니다 [S10].

2.2 저장 모델 — ProseMirror JSON 1 차, Markdown lossy, Yjs binary

저장 모델은 이 장 전체의 핵심 메시지가 소스코드로 확정되는 지점입니다. "Markdown 친화적"이라는 직관과 실제 저장 형식의 차이는 추측이 아니라 `server/models/Document.ts` 의 컬럼 정의로 확인할 수 있습니다 [S8]. 이 파일은 한 문서가 세 가지 표현형 — deprecated Markdown 텍스트, 현행 ProseMirror JSON, 실시간 협업용 Yjs binary — 을 동시에 보관함을 명시합니다. 세 컬럼의 역할을 정확히 이해하면, 추후 다른 도구로 옮길 때의 export 형식과 기존 자산 이행 시의 손실 패턴을 동시에 판단할 수 있습니다.

이 절은 세 컬럼의 정의를 그대로 인용하고, 그중 어느 것이 1 차 저장인지를 메인테이너의 직접 답변으로 확정한 뒤, 세 표현형의 역할 분리를 비교 도표로 정리합니다. 의사결정권자가 "이 도구는 Markdown 으로 데이터를 보관하는가"라는 질문에 사실로 답할 수 있게 하는 것이 목표입니다.

3 개 컬럼의 verbatim 정의

`server/models/Document.ts` 는 문서 본문을 세 개의 컬럼으로 보관합니다 [S8]. 데코레이터를 그대로 인용하면 다음과 같습니다.

```
 @Column(DataType.TEXT) text: string;              // Deprecated markdown
@Column(DataType.JSONB) content: ProsemirrorData | null;  // 현행 표준
@Column(DataType.BLOB) state?: Uint8Array | null;         // Yjs 협업 binary
```

세 컬럼의 역할은 명확히 분리됩니다 [S8]. `text` 는 `DataType.TEXT` 로 선언된 Markdown 문자열이며 주석에 deprecated 로 표기되어 있습니다. 과거 1 차 저장이었으나 현재는 호환·이전용으로 남은 컬럼입니다. `content` 는 `DataType.JSONB` 로 선언된 ProseMirror JSON 이며 현행 표준 1 차 저장입니다. PostgreSQL 의 JSONB 타입을 쓰므로 본문 구조 전체가 쿼리 가능한 JSON 으로 보관됩니다. `state` 는 `DataType.BLOB` 으로 선언된 Yjs binary 로, 실시간 공동 편집의 동기화 state 를 담습니다.

이 세 컬럼이 한 행에 공존한다는 사실이 export 형식과 이행 손실을 동시에 결정합니다. 다른 도구로 옮길 때 보장되는 형식은 deprecated `text` 가 아니라 현행 `content` (JSON)에서 변환된 결과이며, Markdown 은 그 변환의 손실 동반 출력입니다 [S8][S9]. 기존 Markdown 자산을 가져올 때도 입력이 곧바로 1 차 저장이 되는 것이 아니라 ProseMirror JSON 으로 파싱된 뒤 `content` 에 저장되므로, 파서가 표현하지 못하는 구조는 그 시점에 사라집니다. 이 단일 파일이 호환성 판단의 가장 중요한 1 차 출처입니다.

ProseMirror JSON 이 1 차 저장인 이유

1 차 저장이 ProseMirror JSON 이라는 사실은 메인테이너의 직접 답변으로 확정됩니다. Discussion #7396 에서 Markdown 저장 여부를 묻는 질문에 대한 답변은 다음과 같습니다 [S9].

> *"That's correct – and the reason we moved from Markdown is that it cannot represent all the many things that a modern text editor must achieve, but we'll always support a lossy Markdown output."*

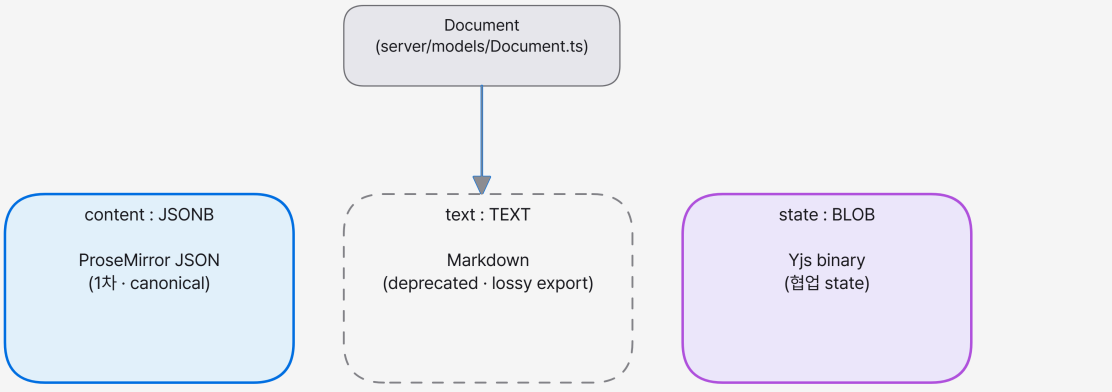
이 답변에는 두 가지 사실이 함께 담겨 있습니다. 첫째, 저장 형식이 Markdown 에서 옮겨졌다는 사실(we moved from Markdown)입니다. 그 이유는 Markdown 이 현대 텍스트 에디터가 다뤄야 하는 많은 요소를 표현하지 못하기 때문입니다 [S9]. 표 안의 복잡한 서식, 임베드 블록, 콜아웃, 협업 메타데이터 같은 구조는 평문 Markdown 의 문법으로는 손실 없이 담기 어렵습니다. 그래서 구조 전체를 보존하는 JSON 이 1 차 저장이 된 것입니다.

둘째, Markdown 출력은 앞으로도 계속 지원되지만 그 성격이 손실 동반(lossy)이라는 사실입니다 [S9]. "we'll always support a lossy Markdown output" 이라는 표현은 export 가능성 자체는 장기적으로 보장하되, 그 출력이 원본 구조를 100% 복원하지 못함을 메인테이너가 직접 인정한 것입니다. 정리하면 이렇습니다 — 1 차 = ProseMirror JSON, export = lossy Markdown. 이 정의는 도입 검토 시 "Markdown 친화적인가"라는 직관 질문에 대한 정확한 답입니다. 화면 편집 경험은 Markdown 문법을 받지만, 데이터의 진실은 JSON 에 있습니다 [S8][S9].

저장 모델 비교 표 — storage-model

세 컬럼이 공존하는 의미를 한눈에 파악하려면 역할과 사용 시점을 함께 보는 것이 효과적입니다.

Outline 저장 모델 — 한 문서, 세 가지 표현형



canonical 은 JSONB content 입니다. Markdown 은 1차 저장이 아니라 lossy export 이며, 이 사실이 Obsidian 자산 이행 손실과 추후 export 보장을 동시에 결정합니다.

캡션: Outline 은 한 문서를 세 가지 표현형으로 동시에 보관합니다 — ProseMirror JSON(content, 1 차) / Markdown TEXT(text, deprecated) / Yjs binary(state, 협업 state).

이 비교 도표는 세 컬럼을 역할(1 차 저장 / 호환·이전용 / 협업 동기화)과 사용 시점(상시 저장 / export 변환 / 실시간 공동 편집)의 두 축으로 정리합니다 [S8]. `content`(JSONB)는 상시 1 차 저장, `text`(TEXT)는 deprecated 상태로 lossy export·하위 호환에 관여, `state`(BLOB)는 다중 사용자가 동시 편집할 때만 활성화되는 Yjs 동기화 state 임을 한 화면에 보여줍니다. 핵심 결론은 한 문장입니다. Outline 은 세 가지 표현형을 동시에 보관하며, 그중 데이터의 진실은 ProseMirror JSON 에 있습니다.

이 역할 분리를 이해하면 두 가지 운영 판단이 명확해집니다. 첫째, 백업·이전 시 진본은 `content` 컬럼의 JSON 이므로, 데이터 정합성을 검증할 때 deprecated `text` 의 Markdown 문자열을 기준으로 삼으면 안 됩니다 [S8]. 둘째, 실시간 협업 state(`state`)는 편집 세션의 동기화를 위한 운영 데이터에 가까우므로, 장기 보관·이행의 대상은 `state` 가 아니라 `content` 입니다. 세 컬럼이 같

은 행에 있다는 사실만으로 세 가지를 모두 동등하게 다루면 이행 설계가 어긋날 수 있습니다 [S8].

2.3 Markdown round-trip 의 보장 범위와 lossy 구간

저장 모델이 ProseMirror JSON 1 차라는 사실을 확정하면, 다음 질문은 자연스럽게 "그렇다면 Markdown 으로 들어오고 나가는 변환은 어디까지 안전한가"가 됩니다. Markdown round-trip — Markdown 을 가져와(import) 다시 Markdown 으로 내보내는(export) 왕복 — 의 보장 범위는 변환을 담당하는 코드와 공식 import 정책으로 가늠할 수 있습니다. Outline 은 자체 ProseMirror serializer 와 markdown-it 기반 파서를 사용하며, 공식 문서는 import 결과에 예상치 못한 부분이 생길 수 있음을 미리 경고합니다 [S8][S48].

이 절은 변환 코드의 위치와 보장 범위, 그리고 공식 경고 문구를 함께 둡니다. 사내 자산 이행 단계에서 "어디까지 잃지 않고 옮길 수 있는가"의 정량 경계를 사전에 확정하는 것이 목적입니다. 구체적인 Obsidian 자산별 손실 매트릭스는 별도 장에서 정량화되며, 이 절은 그 진입점 역할을 합니다.

markdown-it + ProseMirror serializer 의 round-trip 보장

Markdown 변환을 담당하는 코드는 `shared/editor/lib/markdown/` 아래에 있습니다 [S8]. Markdown 에서 ProseMirror 로의 파싱(MD → PM)은 markdown-it 기반의 rules 가, ProseMirror 에서 Markdown 으로의 직렬화(PM → MD)는 자체 구현된 serializer 가 담당합니다 [S8][S9]. 즉 양방향 변환 모두 외부 표준 라이브러리를 그대로 쓰는 것이 아니라, markdown-it 파서와 Outline 자체 serializer 가 ProseMirror 스키마에 맞춰 동작합니다.

round-trip 보장의 경계는 ProseMirror 스키마가 표현할 수 있는 노드 집합과 일치합니다. 표준 CommonMark 와 GFM(GitHub Flavored Markdown) 확장 — 제목, 단락, 목록, 코드 블록, 표, 강조, 링크, 이미지 — 처럼 스키마에 대응 노드가 있는 구조는 왕복 변환에서 안정적으로 보존됩니다. 반면 ProseMirror 스키마에 대응 노드가 없는 구조는 파싱 단계에서 가장 가까운 노드로 치환되거나 사라지며, 이때 발생한 손실은 export 시점에 복원되지 않습니다 [S8][S9]. 보장과 비보장의 경계를 정리하면 다음과 같습니다.

구분	대상	round-trip 동작
보장	CommonMark·GFM 표준 노드(제목·목록·표·코드·강조·링크·이미지)	ProseMirror 스키마 대응 노드로 안정 보존
비보장	스키마에 대응 노드가 없는 비표준 확장 구조	파싱 시 치환·소실, export 시 복원 불가

이 경계가 의사결정에 주는 함의는 분명합니다. 변환의 안전 범위는 "Markdown 문법 전체"가 아니라 "ProseMirror 스키마가 표현하는 범위"로 한정됩니다. 따라서 사내 자산이 표준 Markdown 위주라면 이행 손실이 작고, 특정 도구의 비표준 확장 문법(특수 콜아웃, 쿼리 블록, 인라인 메타데이터 등)에 의존한다면 그 부분에서 손실이 집중됩니다. 이행 견적을 잡을 때는 자산이 어느 쪽에 가까운지를 먼저 표본 점검하는 것이 정확합니다 [S8].

공식 import 정책의 lossy 경고

손실 가능성은 코드 분석만의 결론이 아니라 공식 문서가 이미 명시한 사실입니다. import 정책 문서는 다음과 같이 경고합니다 [S48].

> *"If your current tool allows for exports in HTML or Markdown, you can import those files into your Outline Workspace ... some unexpected results may occur."*

이 문구는 두 가지를 동시에 전달합니다. HTML 또는 Markdown export 를 지원하는 도구라면 그 파일을 Outline 으로 가져올 수 있다는 경로 안내, 그리고 그 과정에서 예상치 못한 결과(some unexpected results)가 생길 수 있다는 경고입니다 [S48]. 앞 절의 메인테이너 답변("lossy Markdown output")과 이 공식 경고는 같은 사실을 양쪽에서 확인합니다 [S9][S48]. 한쪽은 export 가, 다른 한쪽은 import 가 손실을 동반할 수 있음을 인정합니다.

도입 검토 관점에서 이 경고의 가치는 가정 오류를 사전에 차단한다는 데 있습니다. "공식 문서가 import 무손실을 보장한다"는 잘못된 기대로 이행 계획을 세우면, 실제 이행 단계에서 발견한 손실이 일정 지연과 재작업으로 이어집니다. 반대로 공식 문서가 이미 손실을 인정한다는 사실을 알고 시작하면, 표본 검증·수동 보정·우회 도구 도입 같은 대응을 계획 단계에 미리 반영할 수 있습니다 [S48]. 구체적으로 어떤 항목에서 얼마나 손실이 발생하는지의 정량 매트릭스는 별도 장의 Obsidian 호환성 분석에서 항목별로 다룹니다. 이 절에서 확정할 것은 단 하나입니다 — 손실은 추정이 아니라 공식 문서와 소스코드가 함께 인정한 사실입니다 [S9][S48].

3장: 차별화 — 기존 협업 문서 도구·오픈소스 위키와의 차이

자체 호스팅 위키와 협업 문서 도구의 후보군은 넓습니다. Wiki.js, BookStack, DokuWiki, MediaWiki 같은 전통 위키부터 Notion, Confluence 같은 상용 SaaS, Docusaurus, GitBook 같은 문서 사이트 생성기까지 성격이 제각각입니다. 도입 검토 조직이 단일 표만 보고 후보를 좁히려면, 각 도구가 어떤 운영 전제 위에서 동작하는지를 같은 축으로 펼쳐 두어야 합니다. Outline 의 차별화는 한두 개 기능의 유무가 아니라 네 가지 운영 전제의 조합으로 정의됩니다.

첫째는 에디터 위에서 작동하는 실시간 협업입니다. Outline 은 ProseMirror 와 Yjs 기반의 동시 편집을 기본으로 탑재하며, 이 방식은 Wiki.js·BookStack·MediaWiki 가 제공하지 않는 운영 특성입니다. 둘째는 가변 계층 철학입니다. Outline 은 Collections 와 중첩 문서로 구조를 자유롭게 설계하는 반면, BookStack 은 Shelves → Books → Chapters → Pages 의 고정 4단 구조를 강제합니다 [S21]. 셋째는 Slack 을 1급 통합으로 다루는 멘탈 모델이며, 넷째는 빌트인 이메일·비밀번호 로그인의 부재입니다.

이 네 전제는 의사결정에 직접 영향을 줍니다. 실시간 협업이 핵심 요구라면 후보군 자체가 좁아지고, 정보 구조의 자유도가 중요하면 고정 4단 구조 도구는 후순위가 됩니다. 빌트인 인증이 없다는 사실은 도입 전 외부 IdP(Identity Provider) 구축이 선행되어야 함을 의미하므로, 사내 SSO 인프라의 유무가 도입 일정과 예산에 영향을 줍니다 [S32]. 따라서 이 장은 기능 나열이 아니라, 조직의 평가 차원에 맞는 후보를 좁히는 결정 지원 자료로 구성됩니다.

네 전제 가운데 어느 하나도 단독으로는 결정적이지 않다는 점이 중요합니다. 실시간 협업만 보면 상용 SaaS 도 후보가 되고, 자체 호스팅만 보면 전통 위키도 후보가 됩니다. Outline 의 위치를 만드는 것은 자체 호스팅과 실시간 협업과 Markdown 호환을 동시에 거는 제약입니다. 외부 비교 자료가 Outline 을 Notion 대안 자체 호스팅 후보 중 가장 가까운 매칭으로 분류하는 이유도 이 동시 제약에 있습니다 [S22]. 도입 검토 조직은 자사 요구가 이 동시 제약에 해당하는지를 먼저 확인하는 편이 효율적입니다.

실시간 협업이 운영 특성의 분기점이 되는 이유는 도구마다 협업의 구현 위치가 다르기 때문입니다. Notion·Confluence 는 상용 SaaS 로서 동시 편집을 제공하지만 완전한 자체 호스팅 무료 운영의 제약이 따르고, 전통 위키 계열인 Wiki.js·BookStack·MediaWiki 는 자체 호스팅은 자유로우나 에디터 차원의 동시 편집을 제공하지 않습니다 [S21][S25]. 자체 호스팅과 실시간 협업을 동시에 만족하는 후보가 좁다는 점이 Outline 의 상대적 위치를 결정합니다. 도입 검토 조직은 이 두 제약이 동시에 핵심 요구인지를 먼저 판단하면 후보군을 빠르게 줄일 수 있습니다.

같은 고민을 가진 다른 조직도 이 비교를 그대로 재사용할 수 있도록, 평가 차원과 결론을 사실 기반으로 정리합니다. 9종 도구를 8개 평가 차원으로 펼친 매트릭스, 계층 철학의 직접 대비, 인증 빌트인 부재의 운영 의미 세 갈래로 나누어 다룹니다. 각 항목은 외부 비교 자료와 공식 문서의 1차 근거로 뒷받침하므로, 본문의 표와 결론은 사내 도입 검토 문서나 2차 기술 자료로 그대로 인용할 수 있습니다.

3.1 비교 매트릭스 — 9 종 자체 호스팅 위키·협업 문서 도구

자체 호스팅 위키와 협업 문서 도구는 한 가지 기준만으로 우열을 가릴 수 없습니다. 어떤 조직은 실시간 협업을 가장 먼저 보고, 어떤 조직은 운영 단순함과 완전 무료를 우선합니다. 단일 비교 표가 결정에 쓸모가 있으려면 평가 차원이 명확히 정의되어야 하고, 각 차원이 의사결정에 어떻게 기여하는지가 함께 제시되어야 합니다. 이 절은 8개 평가 차원을 먼저 정의하고, 9종 도구를 같은 축에 펼친 뒤, 세 가지 적용 시나리오로 결론을 좁힙니다 [S21][S22][S23].

8 개 평가 차원의 정의

8개 평가 차원은 도입 검토 체크리스트에서 가장 자주 등장하는 항목으로 구성됩니다. 각 차원은 한 줄로 정의되며, 그 차원이 어떤 의사결정에 기여하는지를 명시합니다. 차원의 정의가 사내 검토 기준과 정합하는지를 먼저 확인하면, 이어지는 매트릭스 본체를 읽는 부담이 줄어듭니다.

다음 표는 8개 차원의 정의와 의사결정 기여를 정리한 것입니다. 평가 차원의 정의를 본체 매트릭스와 분리해 두면 표의 가독성이 확보됩니다.

평가 차원	정의	의사결정 기여
에디터	본문 작성에 쓰는 편집기 방식 (WYSIWYG / Markdown / 위키 마크업)	사용자 학습 곡선과 작성 생산성
실시간 협업	동시 편집·커서 공유의 기본 탑재 여부	동시 작업 빈도가 높은 조직의 핵심 요구

평가 차원	정의	의사결정 기여
계층	정보 구조가 가변인지 고정인지	정보 자산 분류의 자유도와 거버넌스 부담
인증 빌트인	이메일·비밀번호 로그인 자체 제공 여부	외부 IdP 선행 구축 필요성
셀프호스팅	자체 호스팅 가능 여부와 라이선스 비용	데이터 주권과 운영 비용
Slack 통합	Slack 연동의 깊이와 1급 통합 여부	사내 협업 도구와의 정합
검색	전문 검색 엔진과 다국어 토큰화 수준	사내 자료 검색 품질
DB/Kanban 뷰	데이터베이스·칸반 등 구조화 뷰 제공 여부	문서 외 워크플로 관리 가능 범위

각 차원은 독립적으로 가중치를 둘 수 있습니다. 실시간 협업과 Slack 통합을 중시하는 조직과, 셀프호스팅 비용과 검색 품질을 중시하는 조직은 같은 매트릭스에서 서로 다른 행을 우선하게 됩니다. 차원을 분리해 두는 이유가 여기에 있습니다. 특히 DB/Kanban 뷰 차원은 Notion 과 그 외 위키 도구를 가르는 결정적 분기입니다. Notion 은 데이터베이스·칸반·캘린더 뷰를 제공하지만, Outline 을 포함한 다수의 위키 도구는 문서 전용이며 구조화 뷰를 제공하지 않습니다 [S24].

에디터 차원도 학습 곡선의 차이를 만듭니다. Outline 은 Markdown 호환 WYSIWYG 를 제공해 작성자가 마크업 문법을 외우지 않아도 즉시 작성할 수 있는 반면, DokuWiki·MediaWiki 는 고유한 위키 마크업을 요구하므로 신규 작성자의 진입 장벽이 상대적으로 높습니다 [S21][S22]. 검색 차원에서는 자체 호스팅 위키 대부분이 데이터베이스 기반 전문 검색을 쓰며, 다국어 토큰화 품질은 도구별로 편차가 큽니다. 셀프호스팅 차원은 라이선스 비용과 직결되므로, 완전 무료 운영을 전제하는 조직은 이 차원에서 후보를 먼저 거를 수 있습니다. 평가 차원을 사내 검토 체크리스트와 1:1 로 대응시켜 두면, 매트릭스 본체를 읽을 때 조직에 무관한 행을 빠르게 건너뛸 수 있습니다.

D4 9 종 비교 매트릭스 본체

9종 도구를 8개 차원으로 펼치면 Outline 의 상대적 위치가 한 자리에 드러납니다. 외부 비교 자료는 행별 결론을 제공합니다. 자체 호스팅 위키 3종 비교에서 Outline 은 협업·Slack 통합에서 우위를 보이고, BookStack 은 단순함과 완전 무료에서 강점을 가지며, 엔터프라이즈 확장성은 별도 도구가 앞선다는 정리가 있습니다 [S21]. Notion 대안 큐레이션에서는 Outline 이 시각·기능 매칭이 가장 가까운 후보로 분류됩니다 [S22]. Wiki.js 와의 직접 비교에서는 UX·협업을 강조하면 Outline, 단순함·완전 무료를 강조하면 Wiki.js 라는 결론이 제시됩니다 [S23].

자체 호스팅 위키·협업 도구 비교 (핵심 차원)

평가 차원	Outline	Wiki.js	BookStack	Notion
실시간 협업	기본 탑재	없음	없음	있음
계층 구조	가변 nested	Path	고정 4단	자유 nested
셀프호스팅	가능(BUSL)	가능(AGPL)	가능(MIT)	불가
Slack 통합	1급	플러그인	제한	있음

Outline 행은 실시간 협업·가변 계층·Slack 1급 통합에서 상대적 우위를 가집니다. 전체 9종·8차원 비교는 본문 표 참조.

캡션: 9종 자체 호스팅 위키·협업 문서 도구 × 8개 평가 차원 비교 매트릭스. Outline 행은 실시간 협업·Slack 통합·가변 계층 3개 차원에서 상대적 우위를 가집니다.

이 도식은 Outline · Wiki.js · BookStack · Notion · Confluence · Docusaurus · GitBook · DokuWiki · MediaWiki 의 9종을 8개 평가 차원에 배치해 Outline 의 우위 차원과 약점 차원을 한눈에 보이게 합니다. Outline 행은 강조 색으로 두고, 실시간 협업·Slack 통합·가변 계층의 세 우위 차원을 캡션에서 명시합니다. 인증 빌트인 차원에서는 Outline 의 부재 표기를, DB/Kanban 뷰 차원에서는 Notion 만 제공함을 대비로 보여 줍니다.

9종을 한 표에 펼치면 도구의 성격이 세 갈래로 나뉘는 것이 드러납니다. 첫째는 Outline·Wiki.js·BookStack·DokuWiki·MediaWiki 처럼 자체 호스팅 무료 운영이 가능한 위키 계열이고, 둘째는 Notion·Confluence 처럼 실시간 협업과 구조화 뷰가 강한 상용 SaaS 계열이며, 셋째는 Docusaurus·GitBook 처럼 문서 사이트 생성에 특화된 계열입니다 [S22][S25]. Docusaurus 는 정적 사이트 빌드 방식이라 동시 편집·빌트인 인증의 개념 자체가 다른 운영 모델에 놓입니다. 같은 표 안에서도 비교의 의미가 도구 계열에 따라 달라지므로, 도입 검토 조직은 자사 요구가 어느 계열에 속하는지를 먼저 정한 뒤 같은 계열 안에서 행을 비교하는 편이 정확합니다.

다음 표는 매트릭스 본체를 정리한 것입니다. 외부 비교 자료의 결론을 행별로 반영했습니다 [S21][S22][S23][S24][S25].

도구	에디터	실시간 협업	계층	인증 빌트인	셀프호스팅	Slack 통합	검색	DB/Kanban 뷰
Outline	WYSIWYG(Markdown 호환)	기본 탑재(ProseMirror+Yjs)	가변(Collections+중첩)	없음(OIDC/SAML/OAuth 필수)	가능(BUSL 1.1)	1급 통합	PostgreSQL FTS	없음
Wiki.js	Markdown/WYSIWYG	미지원	가변(트리)	빌트인+OAuth	가능(무료)	부분	DB/Elasticsearch 선택	없음

도구	에디터	실시간 협업	계층	인증 빌트인	셀프호스팅	Slack 통합	검색	DB/Kanban 뷰
BookStack	WYSIWYG/Markdown	미지원	고정 4단	빌트인+LDAP/SAML	가능(무료)	부분	DB 기반	없음
Notion	블록 WYSIWYG	기본 탑재	가변(중첩)	빌트인	불가(SaaS)	통합	자체	제공
Confluence	WYSIWYG	기본 탑재	Space 기반	빌트인	Data Center 유료	통합	자체	부분
Docusaurus	Markdown(빌드형)	미지원	파일 기반	없음(정적)	가능(무료)	없음	플러그인	없음
GitBook	WYSIWYG/Markdown	부분	가변	빌트인	제한적	통합	자체	없음
DokuWiki	위키 마크업	미지원	네임스페이스	빌트인+플러그인	가능(무료)	플러그인	자체	없음
MediaWiki	위키 마크업	미지원	카테고리	빌트인	가능(무료)	플러그인	자체/확장	없음

이 표에서 실시간 협업을 기본 탑재한 자체 호스팅 후보는 제한적입니다. Wiki.js·BookStack·DokuWiki·MediaWiki 는 동시 편집을 제공하지 않으며, Notion·Confluence 는 실시간 협업을 제공하지만 완전한 자체 호스팅 무료 운영이 어렵습니다 [S21][S25]. Outline 은 자체 호스팅과 실시간 협업을 동시에 만족하는 후보로 분류됩니다. 반면 DB/Kanban 뷰는 Notion 만 온전히 제공하므로, 문서를 넘어 구조화된 워크플로 관리가 필요한 조직은 이 차원에서 다른 결론에 이를 수 있습니다 [S24].

매트릭스 결론 — 3 가지 적용 시나리오

매트릭스를 결정에 쓰려면 차원의 묶음을 적용 시나리오로 환원하는 편이 효율적입니다. 도입 검토 조직이 마주하는 우선순위는 대체로 세 갈래로 수렴합니다. 각 시나리오는 1순위 후보를 명시해 후보 좁히기를 즉시 지원합니다.

다음 세 시나리오는 매트릭스의 행을 우선순위 기준으로 재배열한 결론입니다.

적용 시나리오	우선 차원	1순위 후보	근거
실시간 협업 + Slack 통합 우선	실시간 협업, Slack 통합, 가변 계층	Outline	UX·협업 강조 시 Outline [S23]

적용 시나리오	우선 차원	1순위 후보	근거
단순함 + 완전 무료 우선	셀프호스팅 비용, 운영 단순함	Wiki.js / BookStack	단순함·완전 무료 강조 시 Wiki.js [S23], BookStack 단순함 강점 [S21]
엔터프라이즈 확장성 우선	확장성, 거버넌스, DB/뷰	별도 엔터프라이즈 도구	엔터프라이즈 확장성은 별도 도구 우위 [S21]

첫 번째 시나리오에서 실시간 동시 편집과 Slack 채널형 정보 구조를 핵심 요구로 두는 조직은 Outline 이 가장 가까운 후보입니다 [S23]. Notion 대안을 자체 호스팅으로 찾는 조직에도 Outline 이 시각·기능 매칭이 가장 가까운 후보로 분류되므로, 이 시나리오의 1순위 위치는 외부 큐레이션과도 일치합니다 [S22]. 두 번째 시나리오에서 운영 단순함과 완전 무료 라이선스를 우선하면 Wiki.js 또는 BookStack 이 앞섭니다 [S21][S23]. 이 경우 실시간 협업의 부재를 받아들이는 대신 운영 부담과 라이선스 비용을 낮추는 맞바꿈이 전제됩니다. 세 번째 시나리오에서 대규모 권한 모델과 광범위한 확장성을 우선하는 조직은 엔터프라이즈 지향 도구를 별도로 검토하게 됩니다 [S21].

세 시나리오는 상호 배타적이지 않으므로, 조직은 두 시나리오에 걸친 가중치를 두고 최종 후보를 좁힐 수 있습니다. 예를 들어 실시간 협업을 1순위로 두면서도 라이선스 비용을 강하게 제약하는 조직은 첫 번째와 두 번째 시나리오의 교집합에서 Outline 의 무료 self-hosted 빌드를 검토 후보로 둘 수 있습니다. 매트릭스의 각 행과 시나리오의 결론은 사내 도입 검토 보고서에 근거 출처와 함께 그대로 옮겨 후보 좁히기 단계의 결정 자료로 재사용할 수 있습니다.

3.2 계층 철학 — Collections 가변 vs BookStack 4 단 고정

정보 구조의 철학은 도입 후 거버넌스 부담을 좌우합니다. 같은 양의 문서라도 구조를 어떻게 강제하느냐에 따라 운영팀의 관리 비용과 사용자의 자유도가 달라집니다. Outline 은 Collections 와 중첩 문서로 구조를 가변적으로 설계하게 하고, BookStack 은 Shelves → Books → Chapters → Pages 의 고정 4단 구조를 강제합니다 [S21]. 두 철학은 우열의 문제가 아니라 조직 성향과의 정합 문제입니다.

Collections = Slack 채널 멘탈 모델

Outline 의 Collections 는 두 역할을 동시에 수행합니다. 하나는 권한 경계로, 어떤 사용자가 어떤 문서 묶음에 접근하는지를 결정합니다. 다른 하나는 사용자 멘탈 모델로, 정보를 어디에 두어야 하는지의 직관적 분류 기준이 됩니다. 이 두 역할이 한 단위에 합쳐져 있으므로, 권한 설계와 정보 분류를 별도 작업으로 분리하지 않아도 됩니다.

외부 검증은 이 멘탈 모델을 Slack 채널에 빗댑니다. 사용자 평은 Outline 의 Collections 와 Slack 의 채널이 동등한 개념이라고 정리합니다 [S4]. Slack 채널을 이미 운영하는 조직은 동일한 분류 직관을 그대로 가져올 수 있습니다. 채널 단위로 사람과 정보를 묶는 습관이 Collections 단위 분류로 자연스럽게 이어지므로, 사내 정보 자산 분류의 운영 부담이 줄어듭니다.

다음 정의는 Collections 의 두 역할을 한 자리에 둔 것입니다.

> 🎯 *결정 포인트*
>
> - *Collections = 권한 경계: 접근 제어의 단위*
> - *Collections = 멘탈 모델: 정보 분류의 직관적 기준*
> - *외부 검증: Slack 채널과 동등한 개념으로 인식됩니다 [S4]*

Slack 채널 구조를 사내 정보 구조와 정합시킬 수 있는지의 평가는, Slack 을 이미 핵심 협업 도구로 쓰는 조직에서 도입 결정의 가속 요인이 됩니다. 채널 명명 규칙과 권한 경계를 그대로 Collections 로 옮기면 별도 정보 구조 설계 비용을 줄일 수 있습니다. 권한 경계와 멘탈 모델이 한 단위에 합쳐져 있다는 점은 운영 측면에서도 의미가 있습니다. 권한을 따로 설계하고 정보 분류를 따로 설계하는 이중 작업이 사라지므로, 초기 정보 구조 수립의 검토 항목이 줄어듭니다.

BookStack 4 단 고정과의 운영 의미 차이

BookStack 의 Shelves → Books → Chapters → Pages 4단 구조는 명확성을 제공합니다 [S21]. 모든 문서가 정해진 깊이의 자리에 놓이므로, 신규 작성자가 어디에 글을 두어야 하는지를 고민할 여지가 적습니다. 구조가 고정되어 있으니 운영팀이 분류 규칙을 따로 정의하고 강제할 부담도 작습니다. 반면 깊이가 4단으로 한정되므로, 더 깊은 분류가 필요한 자료나 4단에 맞지 않는 정보는 구조를 비틀어 넣어야 합니다.

Outline 의 가변 중첩 구조는 유연성을 제공합니다. Collections 아래 문서를 임의 깊이로 중첩할 수 있으므로, 자료의 성격에 맞춰 구조를 맞춥니다 [S21]. 대신 구조의 자유도가 높은 만큼 운영팀이 분류 규칙과 거버넌스를 정의하지 않으면 구조가 일관성을 잃을 수 있습니다. 명확성과 유연성은 맞바꿈 관계에 있습니다.

다음 표는 두 계층 철학의 운영 의미를 대비한 것입니다.

항목	Outline (가변 중첩)	BookStack (고정 4단)
구조	Collections + 임의 깊이 중첩	Shelves → Books → Chapters → Pages
강점	자료 성격에 맞춘 유연성	일관된 명확성
약점	거버넌스 미정의 시 일관성 저하	4단을 벗어나는 자료의 수용 한계
운영팀 거버넌스 부담	중간~높음	낮음
사용자 자유도	높음	중간

운영 부담의 정성 평가에서 Outline 은 분류 규칙 정의가 필요해 중간에서 높음, BookStack 은 구조가 고정되어 낮음으로 평가됩니다 [S21]. 거버넌스 자원이 충분하고 자료 성격이 다양한 조직은 가변 구조의 유연성이 유리하고, 운영 인력이 제한적이며 단순 명료함을 우선하는 조직은 고정 구조의 명확성이 유리합니다. 조직 성향에 맞춰 선택하는 것이 결정의 핵심입니다.

계층 철학의 선택은 도입 이후에 되돌리기 어렵다는 점도 고려해야 합니다. 일단 자료가 특정 구조에 쌓이면 다른 철학의 도구로 옮길 때 구조를 재설계해야 하므로, 초기 선택의 무게가 큽니다. 가변 구조를 택한 조직은 도입 초기에 분류 규칙과 명명 규칙을 거버넌스 문서로 정해 두는 편이 장기 일관성에 유리하고, 고정 구조를 택한 조직은 4단 깊이에 맞지 않는 자료의 예외 처리 방침을 미리 합의해 두는 편이 유리합니다. 어느 쪽이든 도구 선택과 함께 거버넌스 방침을 같이 결정하는 것이 운영 안정성을 높입니다.

3.3 인증 빌트인 부재의 의미 — 외부 OIDC / SAML / OAuth 필수

Outline 은 빌트인 이메일·비밀번호 로그인을 제공하지 않습니다 [S32]. 이 한 가지 사실은 도입 전제에 직접 영향을 줍니다. 사용자 인증을 외부 IdP 에 전적으로 위임하므로, 외부 OIDC·SAML·OAuth 중 하나의 경로가 도입 전에 준비되어야 합니다. 사내 IdP 가 없는 조직은 도입 자체에 앞서 IdP 구축이 선행 과제가 됩니다. 이 절은 그 운영 전제를 정리하고, 6장의 Keycloak 연계 가이드로 이어지는 진입점 역할을 합니다.

BookStack · Wiki.js 와의 운영 전제 차이

다른 자체 호스팅 위키는 인증 방식에서 다른 전제를 가집니다. BookStack 은 이메일·비밀번호 빌트인 로그인에 더해 LDAP·SAML 연동을 제공하므로, 외부 IdP 없이도 즉시 사용자를 등록해 운영을 시작할 수 있습니다 [S33]. Wiki.js 역시 빌트인 로그인과 OAuth 연동을 함께 제공합니다. 이들 도구는 IdP 가 없는 환경에서도 빌트인 계정만으로 초기 운영이 가능합니다.

Outline 은 이 지점에서 운영 전제가 다릅니다. 빌트인 로그인이 없으므로, 첫 사용자를 등록하는 시점부터 외부 인증 경로가 동작해야 합니다 [S32]. 이는 약점이라기보다 설계 선택이며, 사내 SSO 를 단일 인증 출처로 강제하려는 조직에는 오히려 정합합니다. 다만 Keycloak·Entra ID·Okta 같은 IdP 가 아직 없는 조직은 도입 전 IdP 구축이 선행되어야 한다는 사실을 사전에 인지해야 합니다.

이 차이는 도입 일정 산정에 직접 반영됩니다. 빌트인 로그인을 가진 도구는 설치 직후 계정을 만들어 시범 운영을 시작할 수 있지만, Outline 은 IdP 연동 검증이 끝나야 첫 로그인이 가능합니다. 사내에 운영 중인 IdP 가 있는 조직은 OIDC 환경변수 설정만으로 빠르게 연결되지만, IdP 자체가 없는 조직은 IdP 도입 프로젝트가 Outline 도입의 선행 단계로 묶입니다 [S32][S33]. 따라서 인증 인프라의 현황을 도입 검토 초기에 확인하면, 일정과 자원 배분의 오차를 줄일 수 있습니다.

> 🎯 *결정 포인트 IdP 선행 필수 — 사내 IdP(Keycloak / Entra ID / Okta)가 없는 조직은 Outline 도입 전 IdP 구축이 선행 과제입니다. 빌트인 로그인을 제공하는 BookStack·Wiki.js 와 달리 첫 사용자 등록 시점부터 외부 인증 경로가 동작해야 합니다 [S32][S33].*

다음 표는 세 도구의 인증 전제를 대비한 것입니다.

도구	빌트인 로그인	외부 연동	IdP 선행 필요
Outline	없음	OIDC / SAML / OAuth 7종	필요

도구	빌트인 로그인	외부 연동	IdP 선행 필요
BookStack	있음	LDAP / SAML	불필요(선택)
Wiki.js	있음	OAuth	불필요(선택)

SAML 게이팅과 Generic OIDC 의 무료 self-hosted 경로

인증 경로를 선택할 때 라이선스 경계를 함께 확인해야 합니다. SAML 인증은 라이선스가 부여된 Business·Enterprise 에디션에서만 제공됩니다 [S18]. 따라서 SAML 을 전제로 도입을 검토하면 라이선스 구입 결정이 동반되며, 이는 컴플라이언스·예산 합의를 도입 일정에 포함시켜야 함을 뜻합니다. SAML 가정이 잠재된 검토는 이 사실을 사전에 고지하지 않으면 후반에 예산 재합의로 지연될 수 있습니다 [S18][S32].

무료 self-hosted 빌드에서의 인증 경로는 Generic OIDC 와 빌트인 OAuth provider 의 조합입니다. 빌트인 OAuth provider 는 다음 7종입니다.

`Slack` · `Google` · `Microsoft` · `Discord` · `GitHub` · `GitLab` · `Generic OIDC`

이 7종 중 Generic OIDC 는 표준 OpenID Connect 를 지원하는 모든 IdP 와 연동하는 범용 경로입니다 [S32]. Keycloak 같은 자체 호스팅 IdP 는 이 Generic OIDC 경로로 연결되며, 8개의 OIDC 환경변수만 설정하면 무료 빌드에서 SSO 가 동작합니다. SAML 이 반드시 필요한 컴플라이언스 요건이 아니라면, 무료 self-hosted 빌드는 Generic OIDC 단일 경로로 사내 IdP 와 정합할 수 있습니다.

다음 표는 무료 self-hosted 빌드에서 선택 가능한 인증 경로를 정리한 것입니다.

인증 경로	무료 self-hosted 가능	비고
빌트인 OAuth 6종	가능	Slack/Google/Microsoft/Discord/GitHub/GitLab [S32]
Generic OIDC	가능	Keycloak 등 표준 OIDC IdP 연동 [S32]
SAML	불가(라이선스 필요)	Business·Enterprise 에디션 한정 [S18]

이 표는 라이선스 경계를 단일 자리에 둡니다. 무료 빌드에서 사내 IdP 와 연동하려는 조직은 Generic OIDC 경로를 1순위로 검토하면 라이선스 비용 없이 SSO 를 확보할 수 있습니다. SAML 을 컴플라이언스 요건으로 명시한 조직만 라이선스 구입을 도입 결정에 포함시키면 되므로, 인증 요건을 먼저 확정하면 라이선스 예산 판단이 단순해집니다 [S18][S32].

빌트인 인증의 부재는 약점과 강점의 양면을 가집니다. 약점은 도입 전 IdP 선행 구축이라는 진입 비용이고, 강점은 인증 출처가 사내 IdP 로 단일화되어 계정 관리·접근 통제·감사 추적이 한 곳으로 모인다는 점입니다. 사내 SSO 를 이미 운영하는 조직에는 후자의 가치가 전자의 비용을 상회합니다. 구체적 Keycloak Realm·Client 설정과 8개 OIDC 환경변수의 단계별 매핑은 6장에서

실측 기준으로 다루며, 여기서는 그 진입점으로서 인증 전제와 라이선스 경계로 범위를 한정합니다.

4장: AI 시대의 통합 — 내장 MCP 와 에코시스템

사내 지식 저장소의 가치는 그 안에 담긴 문서의 양에서 끝나지 않습니다. 그 문서를 기존 업무 도구가 얼마나 자연스럽게 끌어다 쓰는지, 그리고 LLM 에이전트가 그 지식을 직접 다룰 수 있는지가 도입 후 실제 활용도를 결정합니다. Outline 은 20 종 이상의 공식 통합 카탈로그, RPC 스타일 REST API, Webhooks, 자체 OAuth provider 역할, 그리고 2026-02-18 출시된 내장 MCP(Model Context Protocol) 서버를 한데 묶어 이 활용도를 뒷받침합니다 [S11][S14].

이 통합 구성이 필요한 이유는 분명합니다. 사내 표준 도구가 Slack, GitHub, Linear, Figma 같은 협업 도구로 이미 고정된 조직에서, 새 위키가 그 흐름 밖에 홀로 서 있으면 문서는 빠르게 방치됩니다. Outline 은 공식 통합으로 그 흐름 안에 들어가고, REST API 와 Webhooks 로 사내 자작 연동의 진입점을 제공합니다 [S11][S12][S13]. 통합이 단방향 임베드에 그치지 않고 양방향 자동화까지 닿는다는 점이 운영 부담을 줄이는 핵심입니다.

가장 큰 변화는 내장 MCP 서버입니다. MCP 는 LLM 에이전트가 외부 데이터·도구에 표준 방식으로 접근하도록 정의된 개방형 사양이며, Outline 이 이 사양을 서버 측에 내장하면서 Claude, Cursor 를 비롯한 에이전트가 사내 문서를 직접 검색·읽기·생성·편집할 수 있는 경로가 열렸습니다 [S14]. 이는 cloud native 환경에서 지식 자산을 LLM 워크플로우에 연결하는 표준 인터페이스가 제품 안으로 들어왔음을 뜻합니다.

이 장은 의사결정권자가 두 가지를 동시에 판단할 수 있도록 구성됩니다. 하나는 AI Answers 같은 일부 기능이 Cloud 호스팅·라이선스 에디션 전용으로 게이팅되어 있다는 경계이고 [S16], 다른 하나는 그 게이팅에도 불구하고 무료 self-hosted 빌드가 내장 MCP 서버와 사내 LLM 에이전트의 조합으로 동등한 가치를 확보할 수 있다는 우회 경로입니다 [S14]. 두 사실을 함께 보면 라이선스 비용과 AI 활용 전략의 균형을 사전에 설계할 수 있습니다.

4.1 공식 통합 카탈로그 — 20+ 통합과 RPC 스타일 REST API

Outline 의 공식 통합 카탈로그는 협업 메신저, 코드 호스팅, 이슈 트래킹, 디자인, 자동화 도구를 폭넓게 포괄합니다. 도입 검토 조직이 가장 먼저 확인하는 항목은 본인 조직의 표준 도구 5~7 종이 이 카탈로그 안에 들어 있는지입니다 [S11]. 이 절은 카탈로그를 단일 표로 정리해 그 매칭 부담을 줄입니다.

20+ 공식 통합 카탈로그의 사내 도구 매칭

공식 통합 페이지는 메신저(Slack), 코드 호스팅(GitHub, GitLab), 이슈 트래킹(Linear), 디자인(Figma, Miro, Lucidchart, Diagrams.net, InVision), 자동화(Zapier, Make), 문서·표 도구(Google Docs, Microsoft, Airtable), 그리고 다수의 임베드 대상(Loom, Trello, Framer, Whimsical, Mindmeister, Pitch, Prezi, Typeform, Codepen, Spotify·YouTube·Vimeo)을 함께 제공합니다 [S11]. 임베드는 문서 본문 안에 외부 콘텐츠를 그대로 표시하는 방식이고, 양방향 통합은 알림·

링크 미리보기·자동 동기화까지 닿습니다. 다음 표는 사내 표준 도구 분류별로 카탈로그를 정리한 것입니다.

분류	공식 통합 도구	통합 성격
협업 메신저	Slack	알림·검색·링크 미리보기(1급 통합)
코드 호스팅	GitHub, GitLab	링크 미리보기·연동
이슈 트래킹	Linear, Trello	링크 미리보기·임베드
디자인·다이어그램	Figma, Miro, Lucidchart, Diagrams.net, InVision, Whimsical, Mindmeister	임베드
자동화	Zapier, Make	이벤트 기반 워크플로우
문서·데이터	Google Docs, Microsoft, Airtable	임베드·연동
영상·미디어	Loom, Spotify, YouTube, Vimeo	임베드
콘텐츠·기타	Framer, Pitch, Prezi, Typeform, Codepen, Alfred	임베드
AI 에이전트	내장 MCP 서버	LLM 직접 검색·생성·편집

Slack 통합은 단순 임베드를 넘어 알림과 검색까지 닿는 1급 통합으로 다뤄지며, 이는 Outline의 Collections 계층이 Slack 채널의 멘탈 모델과 정합한다는 외부 인식과도 맞물립니다 [S11]. 표의 마지막 행에 둔 내장 MCP 서버는 4.3 에서 별도로 다룹니다.

RPC 스타일 REST API 와 Bearer 인증

Outline 의 REST API 는 자원 중심의 전형적 REST 가 아니라 RPC 스타일입니다. 즉 `/documents.create`, `/documents.search` 처럼 동작을 메서드로 노출하는 방식이며, Outline 자체 애플리케이션이 동일한 API 위에 구축되어 있다는 점이 공식 Developer Portal 에 명시되어 있습니다 [S12] [S7]. 제품과 동일한 API 를 외부에 공개한다는 사실은 사내 자작 통합이나 마이그레이션 스크립트가 1급 기능에 접근할 수 있음을 뜻합니다.

인증은 Bearer 토큰 방식입니다. Settings → API Keys 에서 발급한 키를 다음과 같이 요청 헤더에 담습니다.

> Authorization: Bearer YOUR_API_KEY

이 단일 헤더 모델은 사내 스크립트 작성 부담을 낮춥니다 [S12]. 다만 API Key 는 발급 사용자 권한을 그대로 위임하므로, 자작 연동을 운영에 올릴 때는 키의 보관·회전 정책을 함께 설계하는 편이 안전합니다. API 접근 모델을 사전에 확정해 두면 5장에서 다루는 한국어 검색 우회나 7장의 Obsidian 자산 이행 스크립트도 동일한 인증 경로 위에서 설계할 수 있습니다.

4.2 Webhooks 와 자체 OAuth provider 역할

REST API 가 Outline 으로 들어가는 호출(pull)이라면, Webhooks 는 Outline 에서 나가는 알림(push)입니다. 두 방향을 함께 갖추면 사내 자작 통합은 폴링 없이 이벤트 기반으로 동작할 수 있습니다. 이 절은 Webhooks 페이로드 구조와 자동 비활성화 정책, 그리고 Outline 이 외부 시스템에 인증을 위임하는 자체 OAuth provider 역할을 함께 정리합니다 [S13][S14].

Webhooks 페이로드 구조와 25 회 실패 자동 비활성화

Webhooks 는 문서 생성·수정 같은 이벤트가 발생할 때 등록된 외부 endpoint 로 JSON 페이로드를 전송합니다. 공식 docs 가 정의하는 페이로드 필드는 `id`, `actorId`, `webhookSubscriptionId`, `createdAt`, `event`, `payload` 의 6 종이며, 이 구조를 안다면 수신 측 핸들러를 사전에 설계할 수 있습니다 [S13]. 각 필드는 이벤트 식별자, 발생 주체, 구독 식별자, 발생 시각, 이벤트 종류, 그리고 이벤트별 상세 데이터를 담습니다.

운영 안정성 관점에서 가장 중요한 정책은 자동 비활성화 임계값입니다. 수신 endpoint 가 25 회 연속으로 전송에 실패하면 해당 Webhook 구독은 자동으로 비활성화됩니다 [S13]. 이는 응답 없는 endpoint 로의 무한 재시도가 시스템 부하로 번지는 것을 막는 안전장치이지만, 사내 수신 서버의 일시 장애가 길어지면 구독이 조용히 꺼질 수 있다는 뜻이기도 합니다. 따라서 사내 자작 통합을 운영에 올릴 때는 수신 측 가용성 모니터링과 비활성화 이후 재등록 절차를 운영 절차서에 포함하는 편이 안전합니다.

자체 OAuth provider 역할과 SSRF 차단

Outline 은 외부 IdP 의 클라이언트로만 동작하는 데 그치지 않고, 스스로 OAuth provider 역할을 수행할 수 있습니다. 2024~2025 년 사이 동적 클라이언트 등록(dynamic client registration)이 추가되면서, 외부 애플리케이션이 사전 수동 등록 없이 Outline 에 OAuth 클라이언트로 접속하는 경로가 확보되었습니다 [S14]. 이는 사내 다른 시스템이 Outline 을 인증 소스로 활용하는 구성을 가능하게 합니다.

> 🎯 *결정 포인트 정보보호 책임자의 검토 단계에서 외부 호출의 보안 경계를 먼저 확인하는 편이 안전합니다. Outline 은 외부로 나가는 호출이 내부망의 사설 주소를 향하는 SSRF(Server-Side Request Forgery) 공격에 노출되지 않도록, 외부 요청을 `request-filtering-agent` 를 경유시켜 사설 IP 대역 접근을 차단합니다 [S12]. Webhooks 와 임베드 미리보기처럼 사용자가 임의 URL 을 입력할 수 있는 기능에서 이 필터링이 작동하므로, 자작 통합을 추가할 때도 동일한 필터링 경로를 유지하는지 검토 항목에 두는 편이 좋습니다.*

4.3 내장 MCP 서버 — Claude / Cursor 의 사내 문서 직접 활용

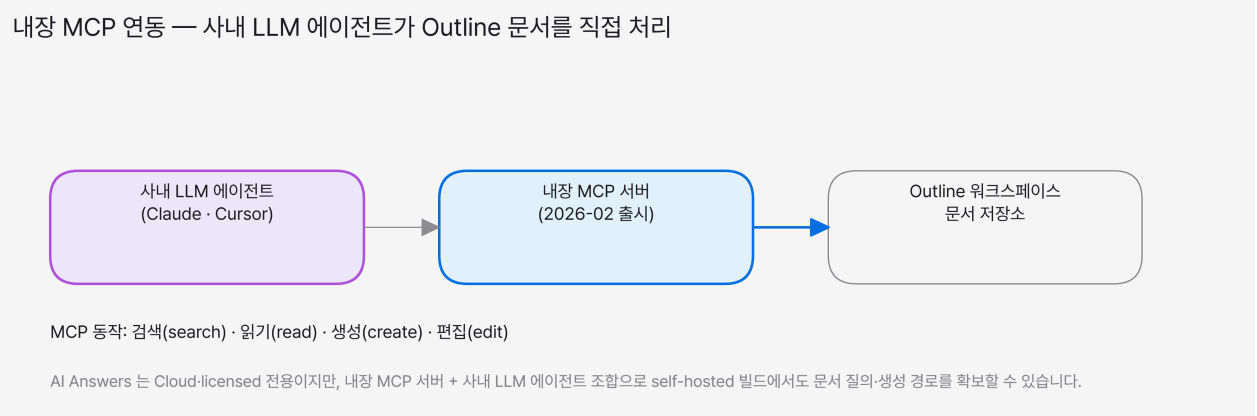
내장 MCP 서버는 Outline 통합 구성에서 가장 새로운 변수이자, AI 시대 도입 판단의 중심에 놓이는 기능입니다. MCP 는 LLM 에이전트가 외부 데이터 소스와 도구에 일관된 방식으로 연결되도록 정의한 개방형 사양이며, 벤더 종속 없이 여러 에이전트가 같은 인터페이스를 공유한다는 점에서 cloud native 환경의 표준 통합 패턴에 가깝습니다 [S14]. 이 절은 출시 사실과 사양을 먼저 확정하고, 사내 활용 시나리오로 연결합니다.

2026-02-18 출시와 MCP 사양 개요

공식 Changelog 는 2026-02-18 자로 내장 MCP 서버 출시를 다음과 같이 기록합니다 [S14].

> *"Outline now ships with a built-in MCP server, allowing Claude, Cursor, and other LLM agents to search, read, create, and edit documents."*

이 한 문장이 정의하는 핵심은 네 가지 기능입니다 — `search`(검색), `read`(읽기), `create`(생성), `edit`(편집). LLM 에이전트가 사내 워크스페이스 문서를 단순히 읽기만 하는 것이 아니라 새 문서를 만들고 기존 문서를 고칠 수 있다는 점에서, MCP 서버는 지식 소비뿐 아니라 지식 생산까지 에이전트에게 위임하는 인터페이스입니다 [S14]. 출시일이 명확한 1차 출처로 확정되므로, 도입 검토 시 이 기능의 가용 시점을 판단 근거로 삼을 수 있습니다.



캡션: 사내 LLM 에이전트가 내장 MCP 서버를 통해 Outline 문서를 검색·읽기·생성·편집하는 연동 흐름.

LLM 에이전트(Claude / Cursor / 사내 에이전트)가 MCP 클라이언트로 동작하고, Outline 내장 MCP 서버가 search / read / create / edit 4 기능을 노출하며, 그 뒤로 ProseMirror JSON 으로 저장된 워크스페이스 문서가 PostgreSQL 에 놓인 3단 흐름을 시각화합니다. 에이전트 → MCP 서버 → 문서 저장소의 단방향 호출과, 생성·편집 결과가 다시 문서 저장소에 반영되는 양방향 경로를 함께 표현합니다.

사내 시나리오 — Claude / 사내 LLM 에이전트의 직접 활용

내장 MCP 서버는 무료 self-hosted 빌드에서도 사내 LLM 에이전트와 결합해 구체적 가치를 만듭니다. 다음 세 가지 시나리오가 대표적입니다. 첫째, 지식 질의응답입니다 — 사내 에이전트가 `search` 와 `read` 로 워크스페이스 전반을 탐색해 흩어진 문서에서 답을 종합하므로, Cloud 전용으로 게이팅된 AI Answers 와 유사한 가치를 self-hosted 환경에서 확보할 수 있습니다 [S14] [S16]. 둘째, 문서 자동 생성입니다 — 회의록·기술 결정 기록 같은 정형 문서를 에이전트가 `create` 로 초안화해 작성 부담을 줄입니다. 셋째, 일관성 유지 편집입니다 — 용어·링크·구조가 어긋난 기존 문서를 에이전트가 `edit` 로 정합시킵니다.

> *데이터 AI Answers 는 공식 문서상 "available only in Cloud hosted and licensed editions of Outline" 으로 명시되어 무료 self-hosted 빌드에는 포함되지 않습니다 [S16]. 그러나 내장 MCP 서버는 2026-02-18 출시 이후 제품에 기본 탑재되므로 [S14], self-hosted*

> *환경에서도 사내 LLM 에이전트를 MCP 클라이언트로 연결하면 검색·요약·생성의 핵심 가치를 우회 확보할 수 있습니다. 즉 AI 활용을 위해 곧바로 라이선스 비용을 부담해야 하는 것은 아니며, 사내 에이전트 운영 역량이 있는 조직은 무료 빌드에서 비교 우위를 확보할 수 있습니다.*

4.4 Roadmap 4 건과 AI Answers / SAML 의 라이선스 게이팅

도입 후 확장성을 판단하려면 향후 기능 추가 방향과 그 기능들이 어느 에디션에 들어오는지를 함께 봐야 합니다. 무료 self-hosted 빌드, Cloud 호스팅, 라이선스 에디션의 경계가 어디에 그어지는지를 사전에 확정하면, 도입 이후 라이선스 검토 시점과 비용 구조를 미리 설계할 수 있습니다. 이 절은 공식 Roadmap In Development 4 건과 AI Answers·SAML·Pricing 의 게이팅을 정리합니다 [S15][S16][S17][S18].

Roadmap In Development 4 건

2026-06 시점의 공식 Roadmap 은 In Development 단계에 4 건을 두고 있으며, 각 항목의 라이선스 게이팅 상태는 향후 확장이 무료 빌드에 닿는지를 결정합니다 [S15]. 다음 표에 항목별 게이팅을 정리합니다.

Roadmap 항목	내용	라이선스 게이팅
Document retention settings	문서 보존 기간 정책	Enterprise
External group syncing	외부 그룹 동기화	Enterprise / Cloud
Editor drag-and-drop improvements	에디터 끌어놓기 개선	All(전 에디션)
Suggested edits	제안 편집	All(전 에디션)

표에서 보듯 향후 확장 4 건 중 2 건은 Enterprise 또는 Cloud 로 게이팅되고, 나머지 2 건은 전 에디션에 공통으로 적용됩니다 [S15]. 특히 External group syncing 은 무료 self-hosted 빌드에서 그룹 매핑을 자동화하려는 조직에 직접 관련된 항목이며, 이 기능이 Enterprise/Cloud 로 게이팅되는 한 사내 외부 매핑 경로를 별도로 준비해야 한다는 점은 6장에서 이어집니다.

AI Answers / SAML / Pricing 의 라이선스 경계

게이팅의 핵심 세 항목은 AI Answers, SAML, 그리고 Cloud 요금 구조입니다. AI Answers 는 "available only in Cloud hosted and licensed editions of Outline" 으로, SAML 인증은 "SAML authentication is only available in the licensed Business + Enterprise editions" 로 각각 명시됩니다 [S16][S18]. 다음 표는 세 항목의 경계를 정리한 것입니다.

항목	무료 self-hosted	Cloud / 라이선스	비고
AI Answers	미포함	Cloud 호스팅·라이선스 전용 [S16]	내장 MCP + 사내 에이전트로 우회 가능 [S14]

항목	무료 self-hosted	Cloud / 라이선스	비고
SAML 인증	미포함	Business + Enterprise 전용 [S18]	self-hosted 는 Generic OIDC 경로(6장)
Cloud 요금	해당 없음	3 단계 구간 [S17]	사용자 수 기준 과금

Cloud 요금은 Starter 월 $10(1~10 명), Team 월 $79(11~100 명, AI Q&A·SSO·API/Webhooks 포함), Business 월 $249(101~200 명, Security audit log)의 3 단계이며, 200 명 초과 구간은 Enterprise 별도 협의입니다 [S17]. 이 구조를 보면 self-hosted 무료 빌드로 시작한 조직이 향후 어느 기능 때문에 어느 단계의 비용을 부담하게 되는지가 분명해집니다. AI 활용은 내장 MCP 로 우회할 여지가 있으나 SAML 이 정책상 필수인 조직은 라이선스 구입이 동반되므로, 도입 검토 단계에서 컴플라이언스 요건과 예산을 함께 합의하는 편이 안전합니다 [S16][S17][S18].

5장: 사내 한국어 환경 — 검색 한계와 우회법

검색은 사내 위키의 사용 빈도를 좌우하는 핵심 기능입니다. 문서가 아무리 많이 쌓여도 원하는 항목을 한 번에 찾지 못하면 구성원은 위키를 외면하고 개인 메모로 회귀합니다. Outline 의 전문 검색(full-text search)은 별도 검색 엔진 없이 PostgreSQL 의 내장 FTS(tsvector/tsquery) 위에서 동작하며, 운영 의존성을 PostgreSQL 과 Redis 두 가지로 묶어 둔 단순한 설계입니다 [S10]. 운영 관점에서는 Elasticsearch 같은 별도 클러스터를 띄울 필요가 없다는 장점이 있지만, 한국어 자산이 많은 조직에서는 이 단순함이 곧 약점으로 작동합니다.

이 장이 다루는 핵심은 한 가지로 모입니다. Outline 의 검색 인덱싱과 쿼리가 양쪽 모두 `'english'` regconfig 를 하드코딩하고 있다는 사실입니다 [S26][S27]. PostgreSQL 의 `english` 텍스트 검색 설정은 영어 snowball stemmer 와 공백 기준 토큰화를 전제하므로, 어절 단위로 조사가 붙는 한국어 문장에서는 의미 있는 토큰을 만들어 내지 못합니다. 결과적으로 "오픈소스"로 검색해도 "오픈소스를", "오픈소스의" 같은 변형이 같은 토큰으로 묶이지 않아 검색 누락이 발생합니다.

도입 검토 조직 입장에서 중요한 점은 이 한계가 가려진 결함이 아니라 소스코드와 공개 Issue 로 추적 가능한 사실이라는 것입니다. CJK(중국어·일본어·한국어) 검색 지원을 요청한 Issue #3207 은 2022년 개설 이후 메인테이너 응답 없이 stale-bot 으로 자동 종료되었고 [S28], 따라서 공식 로드맵에 의존해 한국어 검색이 곧 개선되리라 기대하기는 어렵습니다. 이 장은 그 사실을 1차 출처로 확정한 뒤, 운영 부담과 검색 품질을 정량으로 견줄 수 있는 세 가지 우회 경로를 제시합니다.

세 가지 우회 경로는 각각 목적이 다릅니다. 인덱싱·쿼리 양쪽의 `english` 를 `simple` 로 바꾸는 최소 변경 경로, 형태소 분석기 확장(`pg_jieba` 계열 또는 mecab-ko 기반)을 PostgreSQL 에 붙이는 경로, 그리고 `BaseSearchProvider` 추상 클래스를 통해 Meilisearch·Typesense·Elasticsearch 어댑터를 자작하는 경로입니다 [S30][S31]. 조직의 한국어 자산 비중과 기존 검색 인프라 보유 여부에 따라 이 중 하나를 선택하면, 한국어 검색 한계는 닫을 수 있는 문제로 바뀝니다.

5.1 PostgreSQL FTS 의 `english` regconfig 하드코딩

한국어 검색 한계의 1차 원인은 추측이 아니라 두 개의 소스 파일에서 직접 확인됩니다. 하나는 검색 인덱스를 만드는 데이터베이스 마이그레이션이고, 다른 하나는 사용자 입력을 SQL 쿼리로 변환하는 검색 프로바이더입니다. 두 파일 모두 PostgreSQL 텍스트 검색 설정(regconfig)을 `'english'` 로 고정하고 있어, 한국어 문장이 들어오면 인덱싱 단계와 조회 단계에서 동일한 토큰화 실패를 겪습니다. 이 절은 그 두 지점을 verbatim 으로 확인합니다.

이 구조가 IT 담당자에게 갖는 의미는 우회 작업의 범위입니다. 인덱싱과 쿼리가 별도 파일에서 각각 `english` 를 지정하므로, 검색 동작을 바꾸려면 한 군데만 손봐서는 부족하고 두 지점을 함께 수정해야 합니다. 한쪽만 `simple` 로 바꾸면 인덱스의 토큰 형태와 쿼리의 토큰 형태가 어긋나 오히려 검색이 전혀 되지 않는 상태가 됩니다. 이 절의 두 항은 그 두 지점을 차례로 다룹니다.

5.1.1 인덱싱 트리거 — `setweight(to_tsvector('english', ...))`

문서가 저장되거나 갱신될 때마다 PostgreSQL 은 `searchVector` 컬럼을 다시 계산합니다. 이 계산을 정의한 마이그레이션 `20160711071958-search-index.js` 의 트리거 본문은 다음과 같습니다 [S26].

```
new."searchVector" :=
 setweight(to_tsvector('english', coalesce(new.title, '')), 'A') ||
 setweight(to_tsvector('english', coalesce(new.text, '')), 'C');
```

여기서 `to_tsvector('english', ...)` 가 1차 원인입니다. `english` 설정은 영어 snowball stemmer 와 공백·구두점 기준 토큰화를 적용하므로, "오픈소스 위키입니다" 같은 한국어 어절을 의미 단위로 분해하지 못하고 통째로 또는 부정확하게 잘라 인덱스에 넣습니다. 제목에는 가중치 `'A'`, 본문에는 `'C'` 가 부여되어 제목 일치가 본문 일치보다 상위로 노출되지만, 토큰 자체가 한국어를 제대로 표현하지 못하므로 이 가중치 설계의 이점도 한국어 문서에서는 충분히 살지 못합니다.

핵심은 검색이 깨지는 출발점이 조회가 아니라 저장 시점이라는 데 있습니다. 인덱싱 단계에서 이미 `english` 가 하드코딩되어 어절 토큰화가 어긋나므로, 쿼리만 바꿔서는 결코 해결되지 않습니다. 인덱스에 잘못 들어간 토큰은 어떤 쿼리로도 정확히 매칭되지 않기 때문입니다. 따라서 우회 작업은 반드시 이 트리거의 regconfig 교체와 기존 인덱스 재생성을 포함해야 합니다.

5.1.2 쿼리 — `to_tsquery('english', :query)` + `pg-tsquery`

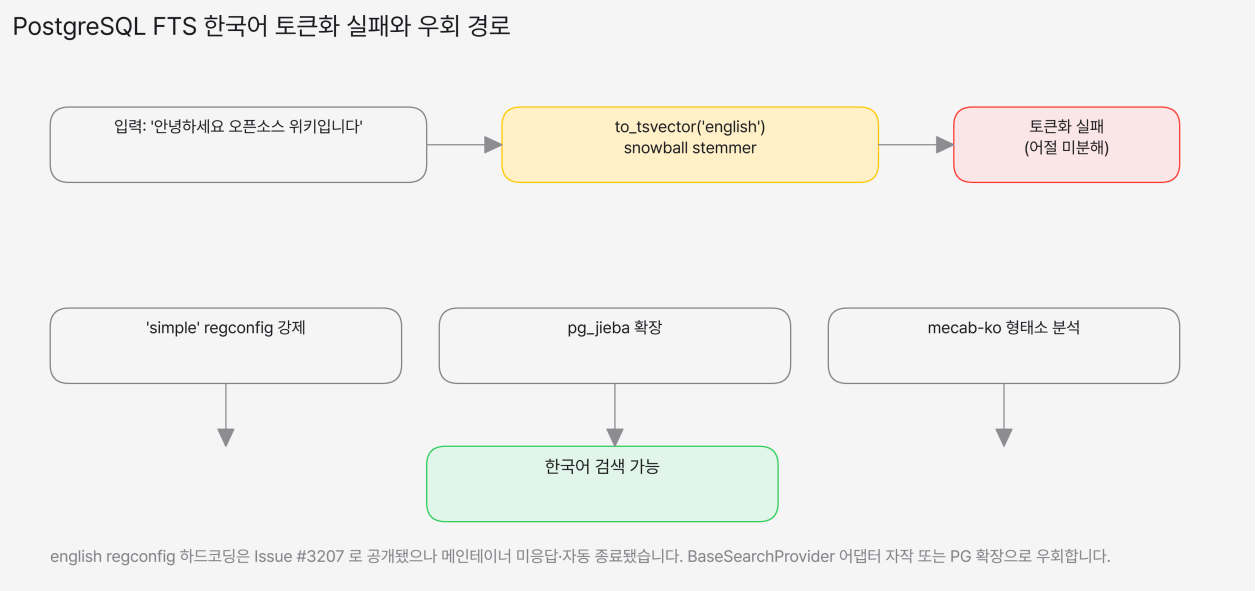
조회 단계의 코드는 검색 프로바이더 `PostgresSearchProvider.ts` 에 있습니다. 이 파일은 사용자가 입력한 검색어를 PostgreSQL 의 `to_tsquery('english', :query)` 로 직접 변환해 `searchVector` 와 매칭합니다 [S27]. 주목할 점은 자유 입력 문자열을 비교적 관대하게 받아 주는 `websearch_to_tsquery` 대신 `to_tsquery` 를 직접 호출한다는 것입니다. `to_tsquery` 는 입력이 엄격한 연산자 문법을 따라야 하므로, 검색어를 그대로 넘기면 구문 오류가 나기 쉽습니다.

이 간극을 메우려고 Outline 은 `pg-tsquery` 라이브러리를 두어 사용자 입력을 `to_tsquery` 가 이해하는 연산자 형태로 미리 변환합니다 [S27]. 그러나 입력을 연산자로 정리하는 과정과 별개로, 변환의 기준이 되는 텍스트 검색 설정은 여전히 `english` 입니다. 즉 쿼리 단계에서도 `english` 가 한 번 더 강제되며, 이 설정은 인덱싱 트리거의 그것과 정확히 짝을 이뤄야 검색이 성립합니다.

> 🎯 *결정 포인트 한국어 검색을 회복하려면 인덱싱 트리거(`20160711071958-search-index.js`)와 검색 프로바이더(`PostgresSearchProvider.ts`)의 regconfig 를 **양쪽 모두** 동일한 값으로 교체하고, 교체 후 기존 `searchVector` 인덱스를 재생성해야 합니다. 한 군데만 바꾸면 인덱스와 쿼리의 토큰 형태가 어긋나 검색이 멈춥니다 [S26][S27].*

5.1.3 토큰화 실패 패턴

이 두 지점이 만들어 내는 실제 결과를 도식으로 정리합니다.



캡션: `to_tsvector('english', '안녕하세요 오픈소스 위키입니다')` 의 토큰화 실패와 세 가지 우회 경로의 비교

`english` regconfig 가 한국어 입력 문장 "안녕하세요 오픈소스 위키입니다" 를 처리할 때, snowball stemmer 와 공백 기준 토큰화가 어절을 의미 단위(어간 + 조사)로 분해하지 못하고 그대로 lexeme 로 굳히는 과정을 왼쪽에 둡니다. 오른쪽에는 `simple` regconfig(stemmer 비활성, 어절 단위 보존), `pg_jieba` ·mecab-ko(형태소 분석으로 "오픈소스"·"위키" 추출), `BaseSearchProvider` 어댑터(외부 검색 엔진의 한국어 분석기 사용) 세 갈래가 같은 입력을 어떻게 다르게 토큰화하는지 대비합니다. 가운데 화살표로 인덱싱과 쿼리 두 지점에서 동일한 교체가 필요함을 표시합니다.

5.2 Issue #3207 의 의미 — CJK 미지원과 stale-bot 자동 종료

소스코드가 한국어 검색 한계의 원인을 보여 준다면, 공개 Issue 트래커는 그 한계가 언제 개선될지에 대한 단서를 줍니다. 결론부터 보면, 공식 로드맵에 기대어 한국어 검색 개선을 기다리는 전략은 권장하기 어렵습니다. CJK 검색 지원 요청이 4년 가까이 응답 없이 자동 종료된 기록이 남아 있기 때문입니다 [S28]. 도입 검토 조직은 이 사실을 근거로, 검색 우회 경로를 도입 시점에 미리 준비해야 합니다.

다만 검색 모듈 전체가 방치된 것은 아니라는 점도 함께 봐야 균형 잡힌 판단이 가능합니다. 검색 관련 Issue 중 일부는 메인테이너가 직접 수정해 종료한 사례도 있어, 검색 안정성과 검색 언

어 지원은 서로 다른 우선순위로 다뤄져 왔습니다 [S29]. 이 절은 두 종류의 Issue 를 나란히 두어, 한국어 검색이 "버그가 아니라 우선순위에서 밀린 기능 요청"이라는 성격을 분명히 합니다.

5.2.1 Issue #3207 의 4년 미해결 패턴

Issue #3207 "Search should support CJK" 는 2022년 3월 7일에 개설되었습니다 [S28]. 이 Issue 는 중국어·일본어·한국어 검색 지원을 요청했지만, 메인테이너의 직접 응답이나 연결된 PR 없이 활동이 멈췄고, 일정 기간 활동이 없는 Issue 를 자동으로 닫는 stale-bot 에 의해 종료되었습니다. 개설부터 종료까지 약 4년에 이르는 이 패턴은, 해당 기능이 거부된 것이 아니라 우선순위에서 지속적으로 밀렸음을 보여 줍니다.

시점	사건
2022-03-07	Issue #3207 "Search should support CJK" 개설 [S28]
개설 이후	메인테이너 직접 응답·연결 PR 부재
활동 정지 후	stale-bot 의 자동 라벨링 및 종료

기술적 배경은 5.1 에서 확인한 `english` 설정과 직결됩니다. `english` snowball stemmer 는 공백을 기준으로 토큰을 자르고 영어 어미 규칙으로 어간을 추출하므로, 어절 중간에 의미 단위가 붙는 한국어를 제대로 잡지 못합니다 [S28]. 한국어는 CJK 라는 큰 묶음 안에서 중국어·일본어와 토큰화 난점을 상당 부분 공유하므로, CJK 미지원이라는 Issue 의 결론이 한국어에도 그대로 적용됩니다.

5.2.2 Issue #2111 의 tsquery 오류 패턴과의 정합

같은 검색 모듈에서, 성격이 다른 Issue 하나를 함께 볼 필요가 있습니다. Issue #2111 은 `Document.searchForUser` 경로에서 `syntax error in tsquery` 스택트레이스가 발생하는 안정성 문제였고, 이 Issue 는 메인테이너(tommoor)가 직접 수정해 종료되었습니다 [S29]. 5.1.2 에서 본 `to_tsquery` 직접 호출과 `pg-tsquery` 변환 계층은 바로 이런 구문 오류를 막기 위한 장치이며, 실제로 수정이 이뤄진 영역입니다.

두 Issue 를 나란히 두면 검색 모듈의 운영 성격이 분명해집니다. 검색이 깨지거나 오류를 던지는 안정성 결함은 비교적 빠르게 수정되는 반면, 한국어를 포함한 비영어권 언어 지원이라는 기능 요청은 미응답 상태로 자동 종료됩니다. 이는 단일 Issue 의 문제가 아니라 검색 모듈 전반의 운영 우선순위가 반영된 패턴입니다.

Issue	성격	처리 결과	출처
#3207 CJK 검색 지원	언어 지원 기능 요청	미응답 stale 자동 종료	S28
#2111 tsquery 구문 오류	검색 안정성 버그	메인테이너 fix 후 종료	S29

이 비교가 의사결정권자에게 시사하는 바는 명확합니다. 검색 안정성은 공식 유지보수에 맡길 수 있지만, 한국어 검색 품질은 도입 조직이 직접 책임지고 우회 경로로 확보해야 하는 영역이

라는 점입니다.

5.3 우회 경로 — simple regconfig · pg_jieba · BaseSearchProvider 어댑터

한국어 검색 한계는 닫을 수 있는 문제입니다. 우회 경로는 크게 세 가지이며, 각각 코드 변경량·운영 부담·검색 품질의 균형점이 다릅니다. 어느 경로를 택할지는 조직의 한국어 자산 비중, 사내 검색 인프라 보유 여부, 운영 인력이 감당할 수 있는 복잡도에 따라 달라집니다. 이 절은 세 경로를 차례로 평가한 뒤 3차원 비교 표로 마무리합니다.

세 경로를 한 문장으로 요약하면 이렇습니다. simple regconfig 는 코드 두 줄을 바꾸는 최소 변경 경로, 형태소 분석기 확장은 PostgreSQL 에 한국어 분석 능력을 더하는 표준 경로, BaseSearchProvider 어댑터는 외부 검색 엔진을 연결하는 최대 품질 경로입니다 [S30][S31]. 출하본이 search-postgres 단일 프로바이더라는 점이 세 경로의 출발선입니다.

5.3.1 simple regconfig 강제 — 최소 변경 우회

가장 단순한 우회는 5.1 에서 확인한 두 지점의 'english' 를 'simple' 로 교체하는 것입니다. PostgreSQL 의 simple 설정은 stemmer 를 적용하지 않고 입력을 토큰 경계에서 자른 그대로 lexeme 로 만듭니다 [S26][S27]. 영어 어미 규칙이 한국어 어절에 잘못 적용되는 일이 사라지므로, "오픈소스" 라는 어절이 인덱스에 온전히 들어가고 같은 어절로 검색했을 때 매칭됩니다. 코드 변경 범위가 인덱싱 트리거와 쿼리 프로바이더 두 곳으로 한정되어 운영 부담이 가장 낮습니다.

다만 회복은 부분적입니다. simple 은 stemmer 가 없으므로 영어 동의어·어형 정규화 이점이 사라지고, 무엇보다 한국어 어절을 형태소 단위로 분해하지 못합니다. "오픈소스를" 과 "오픈소스의" 는 여전히 서로 다른 토큰으로 남아, 조사가 붙은 변형끼리는 매칭되지 않습니다. 따라서 simple 경로는 **최소 변경 / 부분 회복** 으로 위치를 규정할 수 있으며, 한국어 자산 비중이 높지 않거나 PoC 단계에서 빠르게 검증하려는 조직에 적합합니다.

5.3.2 pg_jieba / mecab-ko — 형태소 분석기 확장

검색 품질을 본격적으로 끌어올리려면 PostgreSQL 에 형태소 분석기를 확장으로 붙이는 방법이 있습니다. 중국어 환경에서는 pg_jieba 가 대표적인 형태소 분석기 확장으로 알려져 있으며, PostgreSQL 의 텍스트 검색 파이프라인에 형태소 단위 토큰화를 더합니다 [S31]. 한국어 환경에서는 pg_jieba 가 그대로 맞지 않으므로, mecab-ko 기반의 별도 확장을 도입하거나 mecab-ko 토큰화 결과를 PostgreSQL FTS 에 연결하는 어댑터를 자작해야 합니다 [S31].

형태소 분석이 들어가면 "오픈소스를", "오픈소스의", "오픈소스입니다" 가 모두 "오픈소스" 라는 형태소로 묶여, 조사 변형과 무관하게 검색이 일치합니다. 검색 품질은 세 경로 중 가장 한국어 친화적인 수준으로 올라갑니다. 대신 PostgreSQL 확장 빌드·배포, 분석기 사전 관리, 버전 업그레이드 시 호환성 확인 같은 운영 부담이 중간 수준으로 늘어납니다. 이 경로는 **운영 부담 중간 / 검색 품질 높음** 으로 규정되며, 한국어 자산 비중이 큰 조직이 PoC 단계에서 가장 먼저 검토할 1순위 우회 경로입니다.

5.3.3 BaseSearchProvider 어댑터 자작 — Meilisearch · Typesense · Elasticsearch

세 번째 경로는 PostgreSQL FTS 자체를 우회해 외부 검색 엔진으로 검색 책임을 옮기는 것입니다. Outline 의 검색 계층에는 `BaseSearchProvider` 추상 클래스가 있어, 검색 백엔드를 교체할 수 있는 플러그인 경로가 코드 구조상 열려 있습니다 [S30]. 이론적으로는 이 추상 클래스를 구현해 Meilisearch·Typesense·Elasticsearch 같은 외부 엔진을 연결할 수 있습니다. 이들 엔진은 자체 한국어 분석기를 갖추고 있어 검색 품질을 가장 높게 끌어올릴 수 있습니다.

중요한 전제는 출하본의 실제 구성입니다. 추상 클래스가 존재하더라도 메인 트리에 포함되어 출하되는 프로바이더는 `search-postgres` 단 하나뿐이며, Elasticsearch·Meilisearch·Typesense 어댑터는 메인 트리에 들어 있지 않습니다 [S30]. 따라서 이 경로를 택하면 어댑터 구현·검색 엔진 운영·색인 동기화·업그레이드 추적을 모두 도입 조직이 직접 떠안게 됩니다. 운영 부담이 세 경로 중 가장 크지만, 사내에 이미 Elasticsearch 같은 검색 인프라가 있는 조직이라면 그 인프라를 재활용할 수 있어 1순위 선택지가 됩니다. 이 경로는 **검색 품질 최고 / 운영 부담 최대** 로 규정됩니다.

세 경로를 비용·품질·운영 부담의 3차원으로 정리하면 다음과 같습니다.

우회 경로	코드 변경량	검색 품질(한국어)	운영 부담	적합 조직
`simple` regconfig 강제	최소(2개 지점)	부분 회복(조사 변형 미매칭)	가장 낮음	한국어 비중 낮음·PoC 검증 단계
`pg_jieba` / mecab-ko 확장	중간(확장 빌드·사전)	높음(형태소 단위 매칭)	중간	한국어 자산 비중 큰 조직(1순위)
`BaseSearchProvider` 어댑터	큼(어댑터 자작)	최고(외부 엔진 분석기)	최대	사내 검색 인프라 보유 조직

> *데이터 출하본의 검색 프로바이더는 `search-postgres` 단일 구성이며, 인덱싱·쿼리 양쪽이 `'english'` regconfig 를 하드코딩합니다 [S26][S27][S30]. CJK 검색 지원 요청(Issue #3207)은 2022년 개설 이후 미응답 stale 자동 종료되었습니다 [S28]. 따라서 한국어 검색 품질은 위 세 경로 중 하나를 도입 시점에 선택해 확보하는 것이 안전합니다.*

세 경로 모두 한국어 검색 한계를 닫을 수 있는 실행 가능한 선택지입니다. 도입 검토 조직이 먼저 할 일은 본인 조직의 한국어 자산 비중을 가늠하고, 사내 검색 인프라 보유 여부를 점검한 뒤, 위 표에서 자기 상황에 맞는 한 경로를 고르는 것입니다.

6장: 자체 호스팅 — Keycloak SSO 와 컨테이너 배포

자체 호스팅 협업 위키를 사내에 들이는 일은 사내 신원 체계와의 연결에서 성패가 갈립니다. Outline 의 self-hosted 빌드는 ID/PW 회원가입을 기본 진입로로 두지 않고, 외부 신원 공급자(IdP)를 통한 SSO 로그인을 표준 흐름으로 삼습니다. 사내에 Keycloak 이 이미 운영 중이라면, Outline 은 별도 커넥터 없이 표준 OpenID Connect(OIDC) 경로로 그 Keycloak 에 붙습니다. 이 장은 그 연결을 끊김 없이 완성하기 위한 환경변수·endpoint·배포·함정의 단일 가이드를 제공합니다.

연결 자체는 어렵지 않습니다. `.env.sample` 에 정의된 8개 OIDC 환경변수를 사내 Keycloak Realm 의 endpoint 4건과 맞추면 SSO 로그인이 동작합니다 [S32]. 기본값이 이미 Keycloak 의 표준 claim·scope 와 동일하게 잡혀 있어, 사내 IT 담당자가 추가로 손볼 변수가 많지 않다는 점도 도입 부담을 낮춥니다. 동일한 Generic OIDC 경로는 Keycloak 뿐 아니라 Entra ID·Okta 같은 다른 IdP 에도 그대로 적용됩니다.

다만 실측 환경에서는 같은 증상이 반복되는 4건의 함정이 존재합니다. Keycloak 20 이상의 Client Scope 구성, 사용자 프로필 필수 필드 누락, self-signed 인증서, 그리고 그룹·role 자동 매핑의 부재가 그것입니다 [S35][S36][S37][S38]. 이 가운데 그룹 매핑은 무료 self-hosted 빌드에서 미지원이며 사내 외부 매핑 스크립트로 우회하는 약점입니다. 이 장은 그 약점을 가리지 않고 우회 경로와 함께 둡니다.

마지막으로 컨테이너 배포 표준을 다룹니다. Outline 은 PostgreSQL 과 Redis 두 서비스만 의존하는 가벼운 docker-compose 구성으로 동작하며, 첫 부팅 시 데이터베이스 마이그레이션을 1회 수행합니다 [S10][S34]. 사내 운영의 첫 부팅 실패 대부분은 이 마이그레이션과 `FORCE_HTTPS` 설정에서 비롯되므로, 환경변수 정의·Realm 매핑·함정 회피·배포 표준의 순서로 적용하면 사내 Keycloak 연계를 한 번에 완성할 수 있습니다.

6.1 Generic OIDC 8 환경변수의 verbatim 정의

사내 IT 담당자가 가장 먼저 마주하는 자료는 Outline 저장소의 `.env.sample` 파일입니다. 이 파일은 SSO 연계에 필요한 환경변수를 변수명·기본값까지 그대로 담고 있어, 추측 없이 복사·적용할 수 있는 1차 자료가 됩니다. 잘못된 변수명이나 빈 기본값 가정으로 인한 silent 실패를 차단하려면 이 verbatim 정의가 출발점이 되어야 합니다 [S32]. 아래에서는 OIDC 8개 변수를 그대로 인용하고, 그 위에 7종 빌트인 OAuth provider 카탈로그를 정리합니다.

OIDC 8 환경변수 verbatim

Outline 의 `.env.sample` 은 Generic OIDC 연계를 위한 환경변수를 다음과 같이 정의합니다. 변수명·기본값을 한 글자도 바꾸지 않고 인용합니다 [S32].

```
OIDC_CLIENT_ID=
OIDC_CLIENT_SECRET=
OIDC_AUTH_URI=
OIDC_TOKEN_URI=
OIDC_USERINFO_URI=
OIDC_LOGOUT_URI=
OIDC_USERNAME_CLAIM=preferred_username
OIDC_DISPLAY_NAME=OpenID Connect
OIDC_SCOPES=openid profile email
```

각 변수의 의미는 한 줄씩 짚을 수 있습니다. `OIDC_CLIENT_ID` 와 `OIDC_CLIENT_SECRET` 은 Keycloak Client 등록 시 발급되는 식별자와 비밀값입니다. `OIDC_AUTH_URI` · `OIDC_TOKEN_URI` · `OIDC_USERINFO_URI` · `OIDC_LOGOUT_URI` 네 개는 IdP 의 표준 endpoint 로, 6.2 절에서 Keycloak Realm 패턴과 매핑합니다. `OIDC_USERNAME_CLAIM` 의 기본값 `preferred_username` 은 Keycloak

이 발급하는 표준 클레임명과 정확히 일치하며, `OIDC_SCOPES` 의 기본값 `openid profile email` 역시 Keycloak 의 표준 scope 구성과 동일합니다 [S32]. 즉 사내 Keycloak 을 표준 설정으로 운영한다면 이 두 값은 손대지 않아도 됩니다.

이 기본값 일치는 도입 의사결정의 관점에서 중요한 신호입니다. 사내 IdP 가 표준 OIDC 를 준수하는 한, 별도 클레임 매핑 코드를 작성하거나 커스텀 어댑터를 유지보수할 필요가 없습니다. 운영 부담과 기술 부채를 함께 줄이는 구조이며, 이는 vendor-neutral 한 오픈 스탠다드 채택의 직접적 효과입니다. 변수를 채울 때 비워 둘 값은 Client 식별 정보와 4개 endpoint URI 뿐이라는 점만 기억하면 됩니다.

7종 빌트인 OAuth provider 와 Generic OIDC 의 위치

Outline 은 별도 플러그인 없이 7종의 OAuth provider 를 내장합니다. 사내 IdP 선택의 폭을 가늠하려면 이 카탈로그를 먼저 보는 편이 정확합니다 [S32].

구분	provider	사내 연계 의미
전용 커넥터	`Slack`	메시징 계정 기반 로그인
전용 커넥터	`Google`	Google Workspace 신원
전용 커넥터	`Microsoft`	Microsoft 365 신원
전용 커넥터	`Discord`	커뮤니티형 신원
전용 커넥터	`GitHub`	개발자 신원
전용 커넥터	`GitLab`	개발자 신원
범용 경로	`Generic OIDC`	Keycloak / Entra ID / Okta 등 임의 IdP

여기서 사내 Keycloak 연계의 단일 진입점은 마지막 행의 `Generic OIDC` 입니다 [S32]. 앞의 6종이 특정 SaaS 계정에 묶인 전용 커넥터인 데 반해, `Generic OIDC` 는 표준 OIDC 를 따르는 어떤 IdP 와도 연결됩니다. 따라서 사내가 Keycloak 을 쓰든 Entra ID 나 Okta 로 전환하든, Outline 측 연계 코드 경로는 동일하게 유지됩니다. 이 점은 향후 IdP 교체 시에도 협업 위키 쪽 재설계가 발생하지 않음을 의미하므로, 의사결정권자에게 사전 고지할 가치가 있습니다.

6.2 Keycloak Realm endpoint 4 건의 매핑 가이드

8개 환경변수 가운데 사내 IT 담당자가 실제로 채워야 하는 핵심은 4개 endpoint URI 입니다. 이 네 값은 사내 Keycloak Realm 의 표준 URL 패턴에서 기계적으로 도출되므로, 패턴만 알면 추측 없이 채울 수 있습니다. 외부 공개 가이드는 이 패턴을 verbatim 으로 제시하며, 동시에 Keycloak Client 의 인증 설정 한 가지를 함께 권장합니다 [S33]. 이 절은 그 URL 패턴을 그대로 인용하고, 매핑 관계 전체를 한 장의 도표로 시각화합니다.

4 endpoint URL 패턴 verbatim

사내 Keycloak Realm 의 endpoint 는 다음 패턴을 따릅니다. 외부 가이드의 표기를 그대로 인용하며, `KEYCLOAK_DOMAIN` 과 `REALM` 두 자리만 사내 값으로 치환하면 됩니다 [S33].

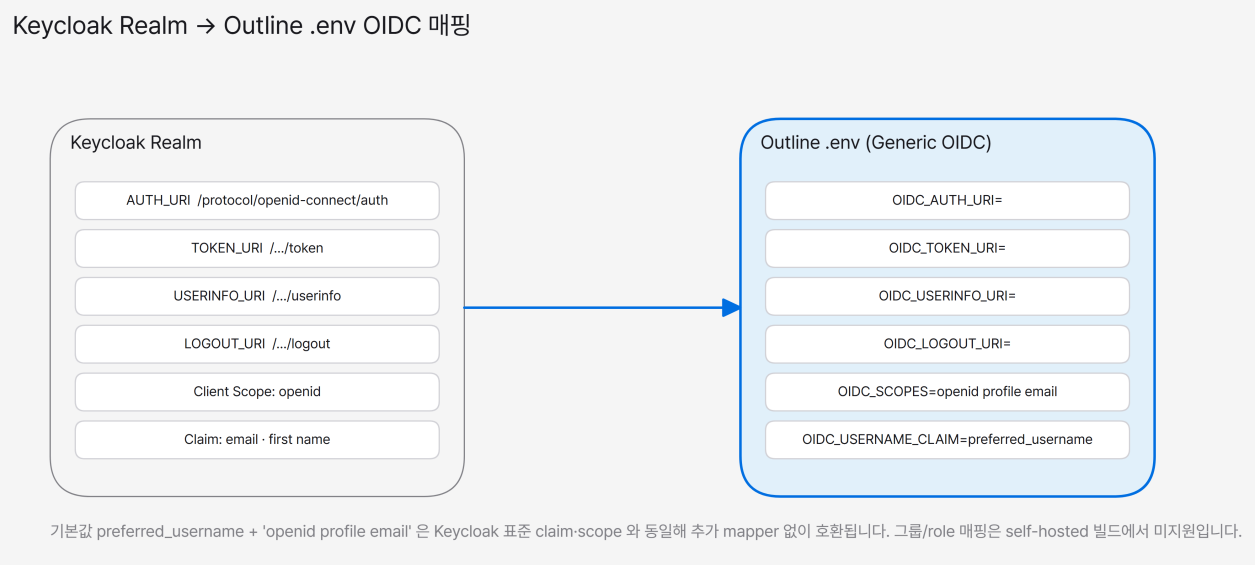
```
OIDC_AUTH_URI=https://KEYCLOAK_DOMAIN/realms/REALM/protocol/openid-connect/auth
OIDC_TOKEN_URI=https://KEYCLOAK_DOMAIN/realms/REALM/protocol/openid-connect/token
OIDC_USERINFO_URI=https://KEYCLOAK_DOMAIN/realms/REALM/protocol/openid-connect/userinfo
OIDC_DISPLAY_NAME=Keycloak
```

이 패턴에서 주의할 부분은 경로의 `realms` 위치입니다. Keycloak 17 미만의 구버전은 `/auth/realms/...` 처럼 `/auth` 접두 경로를 사용했으나, 현재 표준 배포는 위와 같이 `/auth` 가 빠진 `/realms/...` 형식을 씁니다. 구 가이드를 그대로 복사하면 endpoint 가 404 로 응답하므로, 사내 Keycloak 버전에 맞는 경로인지 확인하는 단계가 필요합니다. `KEYCLOAK_DOMAIN` 은 Keycloak 서버의 도메인, `REALM` 은 사내가 운영하는 Realm 이름으로 치환합니다 [S33].

Client 쪽 설정도 한 가지 함께 처리해야 합니다. 외부 가이드는 Keycloak Client 에서 "Enable 'Client authentication' and disable 'Direct access grants'" 를 권장합니다 [S33]. 즉 Client 를 confidential 유형으로 두어 `OIDC_CLIENT_SECRET` 이 발급되도록 하고, 비밀번호 직접 부여(Direct access grants) 흐름은 비활성화하여 표준 authorization code 흐름만 남기는 구성입니다. 이 설정을 빠뜨리면 Client Secret 이 비어 인증이 거부되거나, 의도치 않은 grant 경로가 열린 채로 운영될 수 있습니다.

D6 keycloak-oidc-mapping 다이어그램

지금까지의 매핑 관계는 다음 도식으로 5초 안에 파악할 수 있습니다. 이 도식은 사내 IT 담당자 인수인계 자료로도 그대로 재활용됩니다.



캡션: 4 endpoint + 1 Scope(openid) + 2 필수 클레임(email, first name) — Keycloak Realm/Client 에서 Outline `.env` 로의 OIDC 매핑

좌측에 사내 Keycloak Realm/Client 설정 블록을, 우측에 Outline 의 `.env` OIDC 변수 블록을 배치합니다. 좌측 Realm 의 AUTH·TOKEN·USERINFO·LOGOUT 4개 endpoint 가 각각 우측의 `OIDC_AUTH_URI` · `OIDC_TOKEN_URI` · `OIDC_USERINFO_URI` · `OIDC_LOGOUT_URI` 로 화살표 연결되도록 그립

니다. 그 위에 별도 노드로 Client Scope 1개(`openid`)와 필수 클레임 2개(`email` , `first name`)를 두고, 이 셋이 빠지면 로그인이 silent 실패한다는 경고 표식을 답니다 [S33][S35]. 도식 하단에는 "4 endpoint + 1 Scope + 2 필수 클레임" 의 3라인 결론을 둡니다.

6.3 알려진 함정 4 건 — Client Scope · 필수 프로필 · self-signed cert · 그룹 매핑

환경변수와 endpoint 를 정확히 채웠는데도 로그인이 되지 않는 상황은 실측에서 드물지 않습니다. 공개된 Discussion 기록을 보면 같은 4건의 함정이 반복적으로 보고됩니다 [S35][S36] [S37][S38]. 공통점은 대부분 명시적 오류 메시지 없이 로그인 페이지가 새로고침되는 silent 실패라는 점이며, 그래서 원인 추적에 시간이 많이 듭니다. 이 절은 4건을 각각 증상·원인·해결의 카드 형태로 분리해, 사내 IT 담당자의 운영 시나리오에 직접 대응되게 정리합니다.

함정 1 — Keycloak 20+ Client Scope (Discussion #4566)

첫 번째이자 가장 빈번한 함정은 Keycloak 20 이상에서 `openid` Client Scope 가 자동으로 포함되지 않는다는 점입니다. 증상은 자격 증명을 입력해도 로그인 페이지가 단순히 새로고침되는 형태로 나타나며, 오류 메시지가 표시되지 않아 원인을 짐작하기 어렵습니다. 원인은 Keycloak 20+ 의 Client Scope 모델 변경으로, `openid` scope 가 Client 의 default scope 에 들어 있지 않은 상태입니다. 해결은 Keycloak 에서 `openid` Client Scope 를 별도로 생성한 뒤 해당 Client 의 default scope 에 추가하는 것입니다 [S35]. 사내 Keycloak 이 20 이상이라면 이 단계를 연계 체크리스트 최상단에 두는 편이 안전합니다.

함정 2 — 필수 프로필 필드 누락 (Discussion #3996)

두 번째 함정은 사용자 프로필의 필수 필드가 비어 있을 때 발생합니다. Outline 은 로그인 사용자의 email 과 first name 을 요구하는데, Keycloak 사용자 계정에 이 값이 채워져 있지 않으면 로그인이 완료되지 않습니다. 공개 기록은 그 증상을 "attempting to log into web app will simply refresh the login page" 라고 적고 있습니다 [S36]. 함정 1과 증상이 동일하게 로그인 페이지 새로고침으로 나타나므로, 두 원인을 함께 점검하는 것이 효율적입니다. 해결은 사내 사용자 계정의 email·first name 을 사전에 채우는 것이며, 이는 사내 사용자 프로필 마이그레이션 체크리스트에 별도 항목으로 추가해 두는 편이 좋습니다.

함정 3 — self-signed cert 와 사설 CA (Discussion #2606)

세 번째 함정은 인증서 신뢰 체인에서 비롯됩니다. 사내 Keycloak 이 self-signed 인증서나 사내 사설 CA 가 서명한 인증서를 사용하면, Outline 서버가 IdP 의 endpoint 를 호출할 때 `UNABLE_TO_VERIFY_LEAF_SIGNATURE` 오류로 연결이 끊깁니다 [S37]. 이는 망 격리 환경에서 거의 필수로 마주치는 항목입니다. 사내 운영용 권장 경로는 사설 CA 의 루트·중간 인증서를 Outline 컨테이너의 신뢰 저장소에 추가해 신뢰 체인을 완성하는 것입니다 [S37]. 사설 CA 자체가 아직 없는 조직이라면, PoC 단계에 한해 Let's Encrypt 로 정식 인증서를 임시 적용하는 방법도 가능합니다.

함정 4 — 그룹 / role 매핑 부재 (Discussion #3779)

네 번째 함정은 기능 부재에 해당하므로 성격이 다릅니다. self-hosted OSS 빌드에는 `OIDC_GROUPS_CLAIM` 류의 그룹 매핑 변수가 존재하지 않으며, Keycloak 의 그룹·role 을 Outline 권한으로

자동 동기화하는 기능은 제공되지 않습니다 [S38]. 외부 그룹 동기화(External group syncing)는 Enterprise/Cloud 로드맵의 In Development 항목으로 분류되어 있어, 무료 self-hosted 빌드에서는 미지원 상태입니다 [S38][S15]. 이는 사내 그룹 기반 권한 정책의 자동 반영을 기대하는 정책결정권자에게 반드시 사전 고지해야 할 약점입니다. 우회 경로는 사내 외부 매핑 스크립트가 Keycloak 의 그룹 정보를 읽어 Outline API 로 컬렉션·멤버십 권한을 부여하는 흐름이며, 이 약점은 사내 스크립트로 닫을 수 있습니다.

6.4 docker-compose 배포 표준과 `FORCE_HTTPS` 설정

신원 연계가 완성되면 남는 것은 컨테이너 배포 표준입니다. Outline 은 PostgreSQL 과 Redis 두 서비스만 핵심 의존성으로 두는 가벼운 구성이라, 외부 검색엔진이나 메시지 큐 같은 부가 인프라 없이도 동작합니다 [S10]. 외부 공개 가이드는 이 구성을 docker-compose 스니펫으로 제시하지만, 그 안의 이미지 태그가 오래된 시점의 값이라는 점에 주의해야 합니다. 이 절은 스니펫을 verbatim 으로 인용하되 최신 태그 갱신 필요성을 명시하고, `FORCE_HTTPS` 와 첫 부팅 마이그레이션의 운영 절차를 정리합니다.

docker-compose 스니펫과 이미지 태그

외부 가이드의 docker-compose 스니펫은 다음과 같습니다 [S34].

```
outline:
 image: outlinewiki/outline:0.68.1    # 2026 도입 시 최신 stable v1.8.1 로 교체
 env_file: ./docker.env
 networks: [outline-network]
 ports: ["7215:3000"]
 depends_on: [postgres, redis, minio]
 command: sh -c "yarn db:migrate --env production-ssl-disabled && yarn start"
```

여기서 `depends_on` 에 명시된 `postgres` · `redis` 는 핵심 의존성이고, `minio` 는 첨부 파일 저장소(S3 호환) 용도로 함께 묶인 선택 구성입니다 [S10][S34]. `ports` 의 `7215:3000` 은 컨테이너 내부 3000 포트를 호스트 7215 로 노출하는 매핑으로, 사내 리버스 프록시 구성에 맞춰 외부 포트를 조정할 수 있습니다. 가장 중요한 주의점은 이미지 태그입니다. 스니펫의 `0.68.1` 은 2023~2024 시점 값이며, 2026 도입 시에는 최신 stable 태그 `v1.8.1` (2026-06-06)로 갱신해야 합니다 [S34]. 외부 가이드의 오래된 태그를 그대로 적용하면 최신 보안 패치와 기능이 빠진 채로 운영되므로, 도입 시점에 GitHub Releases 에서 태그를 재확인하는 절차가 필요합니다.

`FORCE_HTTPS` 와 운영 환경변수 보강

스니펫의 `command` 행은 첫 부팅 시 `yarn db:migrate` 로 데이터베이스 스키마 마이그레이션을 1회 수행한 뒤 `yarn start` 로 본체를 기동하는 패턴입니다 [S34]. 사내 운영 첫 부팅 실패의 상당수가 이 마이그레이션 단계에서 발생하므로, 실패 시 절차를 미리 정해 두는 편이 좋습니다. 또한 `FORCE_HTTPS` 환경변수는 Outline 이 모든 요청을 HTTPS 로 강제하도록 하는 설정으로, 사내 리버스 프록시가 TLS 종단을 처리하는 구성에서는 프록시 헤더와의 정합을 확인해야 합니다. 프록시 뒤에서 HTTP 로 들어오는 트래픽에 `FORCE_HTTPS` 가 무리하게 적용되면 무한 리디렉션이 발생할 수 있어, 사내 망 구조에 맞춘 값 결정이 필요합니다.

세 가지 운영 시나리오의 절차를 짧게 정리하면 다음과 같습니다 [S34][S10].

시나리오	동작	권장 절차
첫 부팅	`yarn db:migrate` 후 `yarn start`	PostgreSQL·Redis 기동 완료를 먼저 확인한 뒤 컨테이너를 올립니다
정상 재시작	마이그레이션은 멱등 처리 후 기동	동일 태그로 재기동하면 스키마 변경이 없어 즉시 기동됩니다
마이그레이션 실패	스키마 적용 중단	데이터베이스 연결·권한을 점검하고 `yarn db:migrate` 만 재실행한 뒤 본체를 기동합니다

이 4단계 — `.env` 8개 변수, Realm endpoint 4건, Client·함정 점검, docker-compose 배포 — 를 순서대로 적용하면 사내 Keycloak 연계와 컨테이너 배포가 단일 가이드로 완결됩니다. 그룹 매핑 부재라는 약점은 사내 외부 매핑 스크립트로 닫을 수 있으므로, 무료 self-hosted 빌드만으로도 사내 SSO 표준을 충족하는 협업 위키 운영이 가능합니다.

7장: Obsidian 통합과 사내 마이그레이션 권장 경로

국내 도입 검토 조직 가운데 상당수는 이미 Obsidian 으로 개인 또는 소규모 팀 단위 지식 자산을 축적해 두고 있습니다. Obsidian 은 로컬 Markdown 파일과 위키링크, 콜아웃, frontmatter, Dataview, 인라인 태그, 각주, 수식 등 고유 확장 문법을 풍부하게 제공하는 도구이며, 이렇게 축적된 자산을 자체 호스팅 협업 위키로 옮기는 작업은 도입 의사결정의 마지막 관문이 됩니다. 이 장은 Obsidian 의 8개 핵심 문법 항목이 Outline 으로 옮겨질 때 어떻게 처리되는지를 소스코드 수준 근거와 함께 정리합니다.

핵심은 Markdown 이 Outline 의 1차 저장 포맷이 아니라는 사실입니다. 4장에서 다룬 대로 Outline 의 현행 1차 저장 컬럼은 ProseMirror JSON 이며 Markdown 은 lossy export 출력일 뿐입니다 [S9]. 따라서 Obsidian Markdown 을 Outline 으로 임포트하는 작업은 단순 파일 복사가 아니라 한 편집기 모델에서 다른 편집기 모델로의 변환이며, 이 변환 과정에서 일부 문법은 조용히 손실됩니다. 공식 import 문서가 "some unexpected results may occur" 라는 경고를 명시하는 이유가 여기에 있습니다 [S48].

이 장이 줄이려는 운영·도입 문제는 명확합니다. 손실 항목을 도입 이후에 발견하면 이미 운영 전환이 끝난 뒤이므로 복구 비용이 큽니다. 손실이 발생하는 항목을 사전에 정량으로 분류해 두면, 어떤 항목을 사내 정규화 스크립트가 책임지고 어떤 항목을 PoC 단계에서 실측해야 하는지가 명확해집니다. 이 분류가 곧 마이그레이션 작업의 견적이자 위험 통제 지점이 됩니다. 호환성을 막연한 가부로 두지 않고 항목별 처리 동작까지 소스코드 수준에서 확정하는 이유도 같습니다. 처리 동작이 명확해야 정규화 스크립트의 작성 분량과 PoC 검증 항목 수가 함께 정량화됩니다.

Outline 이 Obsidian Vault 일괄 임포트를 제품 우선순위로 두지 않는다는 점도 사전에 인지할 사실입니다. 관련 Discussion 에서 메인테이너는 Vault 임포트가 우선순위가 아니라고 답하며 REST API 활용을 권합니다 [S42]. 이는 Outline 이 개인 노트 도구가 아니라 자체 호스팅 협업 위키를 지향한다는 제품 포지셔닝과도 맞물립니다 [S4]. 따라서 마이그레이션은 제품이 제공하는 임포트 기능에만 기대지 않고, 조직이 사전 정규화 단계를 직접 설계해 책임지는 구성이 합리적입니다.

7장은 세 단계로 진행됩니다. 먼저 8개 항목의 호환성 매트릭스를 소스코드 근거와 함께 제시하고, 다음으로 외부 변환기 3종의 책임 범위와 사내 정규화 스크립트의 보완 영역을 비교하며, 마지막으로 1~6장의 사실 기반 결론을 의사결정권자용 요약으로 압축하고 Obsidian → Outline 마이그레이션의 권장 경로를 단일 결론으로 제시합니다.

7.1 호환성 매트릭스 — 8 개 항목의 silent loss 정량

Obsidian 의 8개 핵심 문법 항목은 Outline 으로 옮겨질 때 세 가지 상태로 나뉩니다. 부분 호환(재작성 또는 그대로 작동), 하드 비호환(조용한 손실), 실측 미확정의 세 갈래입니다. 이 분류의 정확도는 사내 정규화 스크립트의 작성 범위를 좁히는 직접적인 근거가 됩니다. 막연히 "Markdown 호환" 이라고 표기하는 일반 비교 자료와 달리, 이 절은 각 항목의 처리 동작을 Outline 의 실제 소스코드와 공식 Discussion 에서 확인합니다.

분류의 의미는 단순합니다. 부분 호환 항목은 변환 규칙만 적용하면 의미가 보존되므로 위험이 낮습니다. 하드 비호환 항목은 변환 규칙 자체가 없어 그대로 두면 손실되므로, 사내 정규화 스크립트가 사전에 처리해야 하는 1순위 대상입니다. 실측 미확정 항목은 버전에 따라 동작이 달라질 수 있어 PoC 단계에서 직접 확인해야 합니다. 이 절은 이 세 분류를 7.1.1~7.1.3 순서로 다룹니다.

7.1.1 콜아웃 · 수식 — 부분 호환 항목 2 건의 소스코드 evidence

부분 호환 항목은 콜아웃과 수식 두 가지입니다. 두 항목 모두 Outline 측에 대응되는 처리 경로가 소스코드 수준에서 확인되므로, 변환 규칙만 적용하면 의미를 보존할 수 있습니다.

콜아웃의 경우 Outline 의 `shared/editor/nodes/Notice.tsx` 에 `export enum NoticeTypes { Info, Success, Tip, Warning }` 라는 열거형이 정의되어 있습니다 [S40]. Outline 은 알림 블록을 `:::info` , `:::success` , `:::tip` , `:::warning` 형식의 fenced 블록으로 표현하므로, Obsidian 의 `> [!info]` , `> [!success]` , `> [!tip]` , `> [!warning]` 콜아웃은 이 네 가지 노드 타입으로 재작성하면 매핑됩니다. 다만 Obsidian 콜아웃 종류는 이 네 가지보다 많고(`note` , `abstract` , `question` , `danger` 등), 대응되지 않는 종류는 가장 가까운 타입으로 환원하거나 일반 인용 블록으로 강등되는 손실이 발생합니다. 콜아웃 항목의 결론은 "재작성하면 네 가지 종류는 보존, 나머지는 환원" 입니다.

수식의 경우 `shared/editor/rules/math.ts` 에서 `inlineMathDelimiter = "$"` , `blockMathDelimiter = "$$"` 로 구분자가 정의되어 있고 KaTeX 가 렌더 엔진으로 로드됩니다 [S41]. Obsidian 역시 인라인 수식은 `$...$` , 블록 수식은 `$$...$$` 구분자를 사용하므로 두 구분자 체계가 그대로 일치합니다. Obsidian 의 수식 렌더 역시 MathJax 기반으로 동일한 LaTeX 문법을 따르므로, KaTeX 가 지원하는 표현 범위 안에서는 표기가 그대로 재현됩니다. 따라서 수식은 별도 변환 없이 그대로 옮겨도 작동합니다. 수식 항목의 결론은 "그대로 작동" 입니다.

두 부분 호환 항목을 한 줄로 정리하면, 콜아웃은 재작성이 필요하고 수식은 그대로 작동합니다. 이 두 항목은 silent loss 위험이 낮으므로 정규화 스크립트의 작업량 산정에서 후순위로 둘 수 있습니다. 콜아웃 재작성 규칙은 네 가지 노드 타입 매핑 표 한 장으로 정의되고, 수식은 무처리로 통과시키면 되기 때문입니다. 부분 호환 항목의 경계를 소스코드 수준에서 확정해 두면, 정규화 스크립트의 핵심 작업량이 다음 항의 하드 비호환 4건에 집중된다는 사실이 분명해집니다.

7.1.2 위키링크 · 인라인 태그 · 각주 · Dataview — 하드 비호환 4 건

하드 비호환 항목은 위키링크, 인라인 태그, 각주, Dataview 네 가지입니다. 이 네 항목은 Outline 측에 대응되는 파서나 노드가 존재하지 않아, 그대로 임포트하면 조용히 손실됩니다. silent loss 의 핵심 위험이 여기에 집중되므로, 사내 정규화 스크립트의 1순위 처리 대상입니다.

항목	Obsidian 표기	Outline 처리 동작	1차 근거
위키링크	`[[Page]]`	리터럴 텍스트로 통과 — 링크 미생성, `[label](/doc/{id})` mention 으로 사전 치환 필요	[S42]
인라인 태그	`#tag`	미지원 — 태그 노드 자체가 없음	[S44]
각주	`[^1]`	미지원 — footnote rule 없음, 마커가 본문에 남거나 strip	[S43]
Dataview	```` ```dataview ```` 쿼리	런타임 쿼리 엔진이라 직렬화 불가 — fenced code 로만 보존, 결과 미재현	[S48]

위키링크는 Obsidian Vault 임포트 Discussion 에서 우선순위가 아니라는 답변과 함께 API 사용 권유로 갈음됩니다 [S42]. Outline 은 문서를 ID 기반 경로(`/doc/{id}`)로 식별하므로 `[[Page]]` 형식의 위키링크는 파싱되지 않고 리터럴 텍스트로 남습니다. 따라서 정규화 스크립트가 임포트 전에 각 위키링크를 대상 문서의 mention 링크로 사전 치환해야 링크 관계가 보존됩니다.

인라인 태그는 2018년에 개설된 Issue #765 에서 확인되듯 Outline 에 태그 시스템 자체가 존재하지 않습니다 [S44]. `#tag` 표기는 의미 없는 텍스트로 남습니다. 각주는 2021년부터 2026년까지 미해결 상태인 Discussion #2582 에서 footnote rule 부재가 확인되며 [S43], `[^1]` 마커가 본문에 그대로 노출되거나 제거됩니다. Dataview 는 Obsidian 플러그인이 로컬에서 실시간으로 평가하는 쿼리 엔진이므로, 정적 Markdown 으로 직렬화하면 쿼리 코드 블록만 남고 결과 표는 재현되지 않습니다 [S48]. 네 항목의 동작을 한 단어씩 요약하면 위키링크는 리터럴 통과, 인라인 태그는 strip, 각주는 strip, Dataview 는 fenced code 처리입니다.

> ⚠️ ***silent loss 경고*** *— 위 네 항목은 임포트 시 오류나 경고 없이 조용히 손실됩니다. 원본 Obsidian 자산이 그대로 남아 있더라도, Outline 으로 옮겨진 문서에서는 링크 그래프, 태그*

> *분류, 각주 참조, 동적 쿼리 표가 사라집니다. 사전 정규화 없이 임포트하면 손실 사실 자체를 도입 이후에야 발견하게 됩니다.*

7.1.3 frontmatter — 실측 미확정 1 건과 D7 호환성 매트릭스

여덟 번째 항목인 YAML frontmatter 는 실측 미확정 상태입니다. Obsidian Markdown 의 선두 `---` 블록이 Outline 임포트 시 (a) 통째로 drop 되는지, (b) 수평선(HR) + 본문 텍스트로 파싱되는지, (c) 메타데이터로 보존되는지 가운데 어떤 동작을 보이는지는 공개된 소스코드만으로 단정하기 어렵습니다 [S48]. 세 동작은 결과가 크게 다릅니다. drop 이면 메타데이터가 사라지고, HR + 텍스트 파싱이면 본문 첫머리에 불필요한 키-값 텍스트가 끼어들며, 보존이면 문제가 없습니다. 이 불확실성은 PoC 단계에서 실제 임포트를 수행해 직접 확인할 검증 항목으로 남겨 둡니다.

여덟 항목 전체를 종합하면 다음과 같습니다. 부분 호환 2건(콜아웃·수식 — S40, S41), 하드 비호환 4건(위키링크·인라인 태그·각주·Dataview — S42, S43, S44), 실측 미확정 1건(frontmatter — S48), 그리고 이미지 임베드 1건입니다. 이미지 임베드의 경우 Obsidian 의 위키 임베드 문법(느낌표와 이중 대괄호로 파일을 감싸는 표기)은 표준 Markdown 이 아니므로 파싱되지 않고, 표준 `![](path)` 형식으로 사전 변환하고 이미지 자산을 별도 업로드해야 합니다. 8개 항목의 상태를 한눈에 종합한 도표를 둡니다.

캡션: Obsidian 8개 문법 항목의 Outline 호환성 매트릭스 — 부분 호환 2건, 하드 비호환 4건, 실측 미확정 1건, 사전 변환 필요 1건

위키링크 / 이미지 임베드 / 콜아웃 / frontmatter / Dataview / 인라인 태그 / 각주 / 수식의 8개 항목을 행으로, Obsidian 원본 표기·Outline 처리 동작·상태(직접 호환 / 부분 변환 / silent loss / 실측 미확정)·1차 근거(S40~S48)를 열로 배치합니다. 상태 열은 색상 코드로 구분하여 silent loss 4건을 시각적으로 강조합니다. 캡션 하단에는 "PoC 검증 1항목(frontmatter 실측)" 의 검증 액션을 표기합니다.

7.2 silent loss 경고와 변환기 3 종의 책임 범위

7.1 절에서 정리한 손실 항목을 줄이려면 변환 도구가 필요합니다. 공개된 외부 변환기는 세 가지가 있으며, 각각 책임 범위와 동작 모드가 다릅니다. 이 절은 세 변환기를 비교해 외부 도구가 처리하는 영역과 처리하지 못하는 영역을 구분하고, 사내 정규화 스크립트가 보완해야 할 책임 범위를 구분합니다. 핵심 결정 변수는 외부 변환기를 활용할지, 사내 정규화 스크립트를 자작할지, 또는 둘을 결합할지입니다.

세 변환기는 모두 불완전합니다. 어느 하나도 7.1 절의 하드 비호환 4건을 전부 처리하지 못하므로, 외부 변환기만으로는 손실이 남습니다. 따라서 실무 권장 구성은 외부 변환기로 처리되는 부분 호환 항목을 위임하고, 처리되지 않는 silent loss 항목을 사내 정규화 스크립트가 책임지는 결합 방식입니다. 외부 변환기는 유지보수 주체가 조직 외부에 있으므로 업데이트 중단 위험이 존재한다는 점도 고려 대상입니다. 핵심 손실 보정 로직을 사내 스크립트에 두면, 외부 변환기의 유지보수 상태와 무관하게 손실 통제 책임을 조직 내부에서 확보할 수 있습니다.

7.2.1 변환기 3 종의 기능 비교

공개된 변환기 3종은 동작 모드가 단방향 push, 양방향 동기화, 일회성 마이그레이션으로 각각 다릅니다. 이 차이가 곧 운영 방식의 차이로 이어지므로, 모드를 기준으로 비교합니다.

변환기	동작 모드	처리 범위	한계	근거
defcon1702/obsidian-outline	단방향 push (Obsidian → Outline)	`[[Note]]→/doc/{id}` 부분 매핑, `> [!NOTE]→:::info` 콜아웃 재작성	위키링크 매핑이 부분적, 태그·각주·Dataview 미처리	[S45]
gmiles32/obsout	양방향 mtime 동기화 (Python)	수정 시각 기반 양방향 파일 동기화	문법 변환보다 파일 동기화 중심, 하드 비호환 항목 미보정	[S46]
red-avtovo/outline-migration-scripts	일회성 vault → API	Vault 전체를 Outline API 로 일괄 임포트	일회성 스크립트, 손실 항목 사전 정규화 없음	[S47]

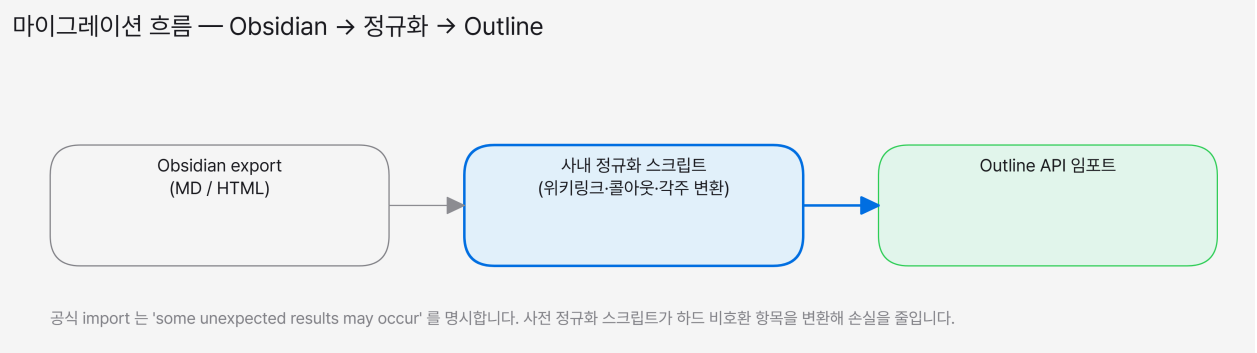
defcon1702/obsidian-outline 는 Obsidian 플러그인 형태로 동작하는 단방향 push 변환기로, 위키링크를 `/doc/{id}` 경로로 부분 매핑하고 `> [!NOTE]` 콜아웃을 `:::info` 로 재작성합니다 [S45]. 2026-02 시점까지 업데이트가 유지되어 세 변환기 중 콜아웃·위키링크 처리가 가장 앞서 있으나, 위키링크 매핑이 모든 경우를 포괄하지 못하고 태그·각주·Dataview 는 손대지 않습니다. gmiles32/obsout 은 파일 수정 시각을 기준으로 한 양방향 동기화 도구로 [S46], 운영 중 지속 동기화에 적합하지만 문법 변환보다 파일 동기화에 무게가 실려 하드 비호환 항목을 보정하지 않습니다. red-avtovo/outline-migration-scripts 는 Vault 전체를 Outline API 로 한 번에 옮기는 일회성 스크립트로 [S47], 초기 일괄 이전에는 유용하나 손실 항목을 사전 정규화하지는 않습니다. 세 변환기의 모드를 단방향 / 양방향 / 일회성으로 구분하면 용도가 명확해집니다.

7.2.2 사내 정규화 스크립트의 책임 범위

외부 변환기 비교에서 드러난 공통 공백은 하드 비호환 4건(Dataview / 인라인 태그 / 각주 / frontmatter)을 어느 변환기도 책임지지 않는다는 점입니다. 이 공백이 곧 사내 정규화 스크립트의 1순위 책임 범위입니다. 사내 정규화 스크립트는 외부 변환기를 대체하는 도구가 아니라, 외부 변환기가 다루지 않는 손실 항목을 임포트 전에 보정하는 전처리 계층입니다.

사내 정규화 스크립트의 구체적 책임은 다음과 같습니다. 위키링크는 대상 문서 mention 링크로 사전 치환하고, 인라인 태그는 본문 텍스트 또는 Collection 분류로 환원하며, 각주는 본문 인라인 참조로 펼치고, Dataview 쿼리는 정적 표로 사전 렌더한 결과를 본문에 고정합니다. frontmatter 는 7.1.3 절의 실측 결과에 따라 보존하거나 본문 메타 블록으로 전환합니다. 이미지 임베드는 Obsidian 위키 임베드 표기를 표준 `![](path)` 로 변환하고 자산을 업로드 큐에 등록합니다.

전체 작업 흐름은 세 단계로 정리됩니다. Obsidian 에서 Markdown 과 자산을 export 하고, 사내 정규화 스크립트로 손실 항목을 사전 보정한 뒤, Outline REST API 로 임포트하는 순서입니다. 이 3단계 구성은 공식 import 정책이 권장하는 방향과도 일치하며 [S48], 단방향 push 변환기를 정규화 단계에 결합하면 콜아웃·위키링크 처리를 위임할 수 있습니다 [S45].



캡션: Obsidian → 사내 정규화 스크립트 → Outline API 임포트의 3단계 마이그레이션 워크플로

Obsidian export(Markdown + 자산), 사전 정규화 스크립트(위키링크 치환·태그 환원·각주 펼침·Dataview 정적화·frontmatter 처리·이미지 변환), Outline API 임포트의 3개 단계를 좌→우 흐름으로 배치합니다. 각 단계 아래에 처리 항목을 작은 목록으로 두고, 정규화 단계에는 외부 변환기 결합 가능 표시를 둡니다.

7.3 결론 — 기술 우위 요약과 Obsidian → Outline 마이그레이션 권장 경로

이 절은 1~6장에서 다룬 사실 기반 결론을 의사결정권자용 한 페이지 요약으로 압축하고, Obsidian → Outline 마이그레이션의 권장 경로를 단일 결론으로 제시합니다. Outline 은 자체 호스팅과 Markdown 호환, 실시간 협업이라는 세 제약을 동시에 만족하는 거의 유일한 오픈소스 위키이며 [S3], "communal long-term memory" 라는 제품 컨셉이 사내 장기 지식 자산의 위치와 정확히 맞물립니다 [S3].

7.3.1 기술 우위 요약 — 6 변수 결정 신호

도입 의사결정을 좌우하는 6개 변수와 각 변수의 결정 신호를 요약하면 다음과 같습니다. 도입 가능, 보류, 우회 필요의 세 신호로 구분합니다.

변수	핵심 사실	결정 신호	근거
라이선스	BUSL 1.1, 자체 호스팅·사내 운영 전부 허용, 2030-06-06 Apache 2.0 자동 전환	도입 가능	[S1]
아키텍처	Koa + React + ProseMirror + PostgreSQL + Redis, Yjs 협업, 의존성 최소	도입 가능	[S7]
차별화	Notion 대안 자체 호스팅 위키, Collections 계층·Slack 통합 우위	도입 가능	[S4]
에코시스템 + AI	20+ 통합, 2026-02 내장 MCP 서버, AI Answers 는 Cloud·라이선스 한정	도입 가능 (AI Answers 는 보류)	[S14]
한국어 검색	PostgreSQL FTS `english` stemmer, CJK 토큰화 미흡, 우회 필요	우회 필요	[S28]
SSO + 배포	OIDC 8 환경변수로 Keycloak 연동, SAML·그룹 동기화는 라이선스 한정	도입 가능 (그룹 동기화는 우회)	[S32]

여섯 변수 가운데 라이선스·아키텍처·차별화 세 변수는 도입 가능 신호가 분명합니다. 에코시스템과 SSO 두 변수는 핵심 기능은 도입 가능하되 AI Answers 와 그룹 동기화처럼 라이선스에 묶인 일부 기능에 보류 또는 우회 판단이 필요합니다. 한국어 검색 한 변수만 우회 필요 신호를 보이며, PostgreSQL FTS 의 CJK 토큰화 한계를 `simple` regconfig 나 형태소 분석기 확장으로 보완하는 작업이 전제됩니다 [S28]. 6변수 가운데 단일 차단 요인은 없으며, 한국어 검색이 유일한 사전 보완 과제로 남습니다.

> 🎯 ***결정 포인트*** *— 6변수 중 도입을 가로막는 단일 차단 요인은 없습니다. 한국어 검색 우회 한 건과 라이선스 게이팅 기능(AI Answers·그룹 동기화)의 운영 영향 평가가 도입 판단의 마지막 두 항목입니다.*

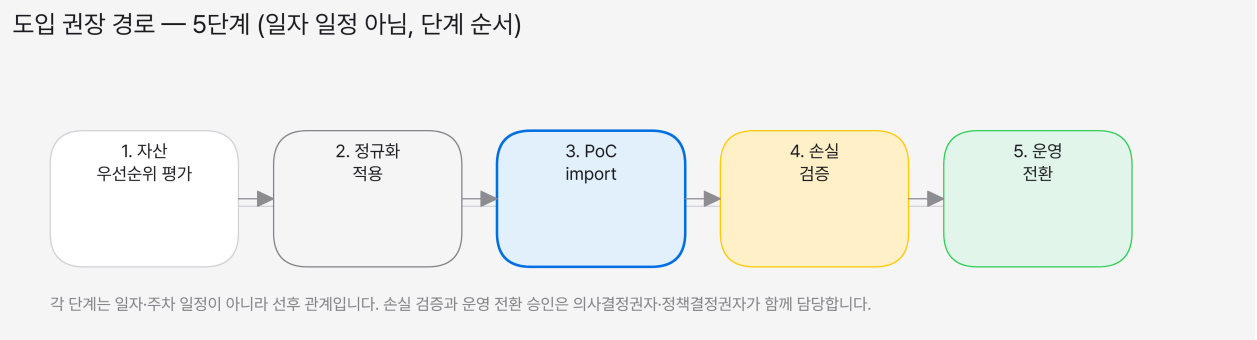
7.3.2 Obsidian → Outline 마이그레이션 권장 경로

백서의 모든 사실과 인용, 도표가 수렴하는 단일 결론은 다섯 단계의 마이그레이션 권장 경로입니다. 일자나 주차 단위 일정이 아니라 책임자와 검증 항목으로 구성된 순서이며, 조직의 자산 규모에 따라 각 단계의 소요는 달라집니다.

첫째, 사내 자산 우선순위 평가입니다. 어떤 Obsidian 자산을 옮길지, 그 자산이 7.1 절의 8개 항목 가운데 어떤 문법을 얼마나 사용하는지 분류합니다. 이 단계의 책임은 자산을 보유한 IT 담당자에게 있으며, 손실 위험이 큰 항목(하드 비호환 4건)의 사용 빈도를 정량화합니다.

둘째, 정규화 스크립트 적용입니다. 7.2.2 절의 책임 범위에 따라 위키링크 치환·태그 환원·각주 펼침·Dataview 정적화·이미지 변환을 수행하고, 외부 변환기 결합 여부를 결정합니다 [S45]. 이 단계의 책임은 IT 담당자에게 있습니다. 셋째, PoC 환경 import 입니다. 정규화된 자산을 별도 PoC 환경의 Outline 으로 임포트하고, 7.1.3 절의 frontmatter 실측을 포함해 손실 여부를 확인합니다 [S48].

넷째, 손실 검증입니다. PoC 임포트 결과를 원본과 대조해 silent loss 가 남은 항목을 확인하고, 정규화 스크립트로 보정 가능한지 판단합니다. 이 단계의 결과는 도입 가부의 핵심 근거가 되므로 의사결정권자가 검토합니다. 다섯째, 운영 전환입니다. 검증을 통과한 자산을 운영 환경으로 옮기고, SSO 연동과 한국어 검색 보완을 포함한 운영 구성을 결정합니다. 운영 전환의 최종 승인은 정책결정권자와 의사결정권자가 함께 담당합니다.



캡션: 자산 우선순위 평가 → 정규화 적용 → PoC import → 손실 검증 → 운영 전환의 5단계 권장 경로

자산 우선순위 평가(IT 담당자), 정규화 스크립트 적용(IT 담당자), PoC 환경 import(IT 담당자), 손실 검증(의사결정권자 검토), 운영 전환(정책결정권자·의사결정권자 승인)의 5개 단계를 좌→우 flow 로 배치합니다. 각 단계 하단에 책임자를 표기하고, 손실 검증 단계에서 가부 분기가 갈라지는 결정 지점을 강조합니다.

결론적으로 Obsidian 자산의 Outline 이전은 단순 파일 복사가 아니라 lossy 변환이며, 손실 항목을 사전 분류해 정규화 책임 범위로 옮기는 작업이 마이그레이션의 핵심입니다 [S9]. 8개 항목의 호환성 매트릭스와 변환기 3종의 책임 범위, 6변수 결정 신호가 이 단일 결론으로 수렴합니다.

Appendix A. References

1. [S1] GitHub outline/outline LICENSE @ main — https://github.com/outline/outline/blob/main/LICENSE (접속일 2026-06-21)
2. [S2] LICENSE commit history — https://github.com/outline/outline/commits/main/LICENSE (접속일 2026-06-21)
3. [S3] Outline About 페이지 — https://www.getoutline.com/about (접속일 2026-06-21)
4. [S4] Hacker News tommoor 답글 #39012054 — https://news.ycombinator.com/item?id=39012054 (접속일 2026-06-21)
5. [S5] Discussion #3301 (BUSL 자체 호스팅 영향) — https://github.com/outline/outline/discussions/3301 (접속일 2026-06-21)
6. [S6] package.json @ main — https://raw.githubusercontent.com/outline/outline/main/package.json (접속일 2026-06-21)
7. [S7] docs/ARCHITECTURE.md — https://github.com/outline/outline/blob/main/docs/ARCHITECTURE.md (접속일 2026-06-21)
8. [S8] server/models/Document.ts — https://github.com/outline/outline/blob/main/server/models/Document.ts (접속일 2026-06-21)
9. [S9] Discussion #7396 (Markdown lossy 답변) — https://github.com/outline/outline/discussions/7396 (접속일 2026-06-21)
10. [S10] docker-compose.yml @ main — https://github.com/outline/outline/blob/main/docker-compose.yml (접속일 2026-06-21)
11. [S11] Outline 공식 통합 카탈로그 — https://www.getoutline.com/integrations (접속일 2026-06-21)
12. [S12] Outline Developer Portal — https://www.getoutline.com/developers (접속일 2026-06-21)
13. [S13] Webhooks docs — https://docs.getoutline.com/s/guide/doc/webhooks-gB7HYhS6yq (접속일 2026-06-21)
14. [S14] Outline Changelog (MCP 출시 포함) — https://www.getoutline.com/changelog (접속일 2026-06-21)
15. [S15] Outline Roadmap — https://docs.getoutline.com/s/roadmap (접속일 2026-06-21)
16. [S16] AI Answers docs — https://docs.getoutline.com/s/guide/doc/search-ai-answers-NIKPvYrx06 (접속일 2026-06-21)
17. [S17] Outline Pricing — https://www.getoutline.com/pricing (접속일 2026-06-21)
18. [S18] SAML hosting docs — https://docs.getoutline.com/s/hosting/doc/saml-hCmJIfmAjt (접속일 2026-06-21)

19. [S20] Docmost 비교 블로그 — https://docmost.com/blog/outline-wiki-alternatives/ (접속일 2026-06-21)
20. [S21] MassiveGRID XWiki·BookStack·Outline 비교 — https://massivegrid.com/blog/xwiki-vs-bookstack-vs-outline-comparison/ (접속일 2026-06-21)
21. [S22] selfh.st Notion 대안 카탈로그 — https://selfh.st/alternatives/notion/ (접속일 2026-06-21)
22. [S23] dev.to/selfhostingsh Wiki.js vs Outline — https://dev.to/selfhostingsh/wikijs-vs-outline-which-to-self-host-lo1 (접속일 2026-06-21)
23. [S24] dev.to/selfhostingsh Outline vs Notion — https://dev.to/selfhostingsh/outline-vs-notion-self-hosted-alternative-4jg8 (접속일 2026-06-21)
24. [S25] Outline Confluence 비교 페이지 — https://www.getoutline.com/compare/confluence-alternative (접속일 2026-06-21)
25. [S26] search-index migration 20160711071958 — https://github.com/outline/outline/blob/main/server/migrations/20160711071958-search-index.js (접속일 2026-06-21)
26. [S27] PostgresSearchProvider.ts — https://github.com/outline/outline/blob/main/plugins/search-postgres/server/PostgresSearchProvider.ts (접속일 2026-06-21)
27. [S28] Issue #3207 (CJK 검색) — https://github.com/outline/outline/issues/3207 (접속일 2026-06-21)
28. [S29] Issue #2111 (tsquery 오류) — https://github.com/outline/outline/issues/2111 (접속일 2026-06-21)
29. [S30] BaseSearchProvider.ts — https://github.com/outline/outline/blob/main/server/utils/BaseSearchProvider.ts (접속일 2026-06-21)
30. [S31] pg_jieba 저장소 — https://github.com/jaiminpan/pg_jieba (접속일 2026-06-21)
31. [S32] .env.sample @ main — https://github.com/outline/outline/blob/main/.env.sample (접속일 2026-06-21)
32. [S33] wenkdth.org Keycloak + Outline 가이드 — https://wenkdth.org/posts/keycloak-outline-setup/ (접속일 2026-06-21)
33. [S34] heyvaldemar.com Keycloak + Outline 가이드 — https://heyvaldemar.com/install-outline-and-keycloak-using-docker-compose/ (접속일 2026-06-21)
34. [S35] Discussion #4566 (Keycloak 20+ Scope) — https://github.com/outline/outline/discussions/4566 (접속일 2026-06-21)
35. [S36] Discussion #3996 (필수 프로필 필드) — https://github.com/outline/outline/discussions/3996 (접속일 2026-06-21)

36. [S37] Discussion #2606 (self-signed cert) — https://github.com/outline/outline/discussions/2606 (접속일 2026-06-21)
37. [S38] Discussion #3779 (그룹 매핑 부재) — https://github.com/outline/outline/discussions/3779 (접속일 2026-06-21)
38. [S40] shared/editor/nodes/Notice.tsx — https://github.com/outline/outline/blob/main/shared/editor/nodes/Notice.tsx (접속일 2026-06-21)
39. [S41] shared/editor/rules/math.ts — https://github.com/outline/outline/blob/main/shared/editor/rules/math.ts (접속일 2026-06-21)
40. [S42] Discussion #4601 (Obsidian Vault import) — https://github.com/outline/outline/discussions/4601 (접속일 2026-06-21)
41. [S43] Discussion #2582 (Footnote 지원) — https://github.com/outline/outline/discussions/2582 (접속일 2026-06-21)
42. [S44] Issue #765 (인라인 태그) — https://github.com/outline/outline/issues/765 (접속일 2026-06-21)
43. [S45] defcon1702/obsidian-outline 변환기 — https://github.com/defcon1702/obsidian-outline (접속일 2026-06-21)
44. [S46] gmiles32/obsout 양방향 동기화 — https://github.com/gmiles32/obsout (접속일 2026-06-21)
45. [S47] red-avtovo/outline-migration-scripts — https://github.com/red-avtovo/outline-migration-scripts/blob/main/obsidian_to_outline.py (접속일 2026-06-21)
46. [S48] Outline import 정책 docs — https://docs.getoutline.com/s/guide/doc/import-D2ZvLqz411 (접속일 2026-06-21)
47. [S49] dev.to Tom Moor 인터뷰 — https://dev.to/craft-of-open-source/tom-moor-founder-outline-getoutline-com (접속일 2026-06-21)
48. [S50] GitHub API repo metadata — https://api.github.com/repos/outline/outline (접속일 2026-06-21)

Appendix B. Glossary

용어	정의
BUSL 1.1	Business Source License 1.1. 자체 호스팅·사내 운영은 무료로 허용하고 클라우드 SaaS 재판매만 제한하는 소스 가용(source-available) 라이선스입니다. 지정된 전환일에 오픈소스 라이선스로 자동 전환됩니다.

용어	정의
Apache License 2.0	허용적(permissive) 오픈소스 라이선스입니다. BUSL 1.1 의 전환 라이선스로 지정되어 있습니다.
ProseMirror	구조화된 문서 모델 위에서 동작하는 리치 텍스트 에디터 프레임워크입니다. Outline 의 실시간 협업 편집 기반입니다.
Yjs	여러 사용자의 동시 편집을 충돌 없이 병합하는 CRDT 기반 실시간 협업 라이브러리입니다.
Koa	Node.js 용 경량 웹 프레임워크입니다. 미들웨어 기반으로 동작합니다.
MobX	관찰 가능한 상태로 화면을 갱신하는 React 용 상태 관리 라이브러리입니다.
Sequelize	Node.js 용 ORM(객체-관계 매핑) 도구입니다.
ORM	Object-Relational Mapping. 객체와 관계형 데이터베이스를 잇는 매핑 계층입니다.
JSONB	PostgreSQL 의 이진 JSON 저장 타입입니다. Outline 문서의 표준 저장 형식입니다.
FTS	Full-Text Search(전문 검색). PostgreSQL 내장 검색 기능으로 tsvector·tsquery 타입을 사용합니다.
tsvector	PostgreSQL 전문 검색의 색인 표현형입니다.
tsquery	PostgreSQL 전문 검색의 질의 표현형입니다.
regconfig	PostgreSQL 전문 검색에서 토큰화·형태소 처리 방식을 결정하는 텍스트 검색 구성입니다(예: english, simple).
pg_jieba	CJK 토큰화를 PostgreSQL 전문 검색에 추가하는 확장입니다.
OIDC	OpenID Connect. OAuth 2.0 위에서 동작하는 표준 인증 프로토콜입니다.
SAML	Security Assertion Markup Language. 기업용 SSO 에 널리 쓰이는 XML 기반 인증·인가 표준입니다.
SSO	Single Sign-On(통합 인증). 한 번의 로그인으로 여러 서비스에 접근하는 방식입니다.
Keycloak	오픈소스 아이덴티티·접근 관리(IAM) 서버입니다. OIDC·SAML 기반 SSO 를 제공합니다.

용어	정의
Realm	Keycloak 에서 사용자·클라이언트·인증 정책을 묶는 격리 단위입니다.
MCP	Model Context Protocol. LLM 에이전트가 외부 데이터·도구에 표준 방식으로 접근하도록 하는 프로토콜입니다.
Webhook	특정 이벤트 발생 시 외부 URL 로 HTTP 요청을 보내는 알림 메커니즘입니다.
REST API	자원 중심으로 설계한 웹 API 방식입니다.
RPC	Remote Procedure Call. 원격 함수 호출 중심의 API 설계 방식입니다.
Collections	Outline 에서 문서를 묶고 권한 경계를 형성하는 최상위 그룹입니다.
SSRF	Server-Side Request Forgery. 서버가 신뢰되지 않은 URL 로 요청하도록 유도하는 공격입니다.
KaTeX	웹에서 수식을 빠르게 렌더링하는 라이브러리입니다.

Outline 협업문서 시스템 — 자체 호스팅 협업 위키 도입 의사결정 가이드

CONTACT

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...
www.linkedin.com/company/cloud-na...

FACEBOOK
facebook.com/cncf
www.facebook.com/cncf

SCAN