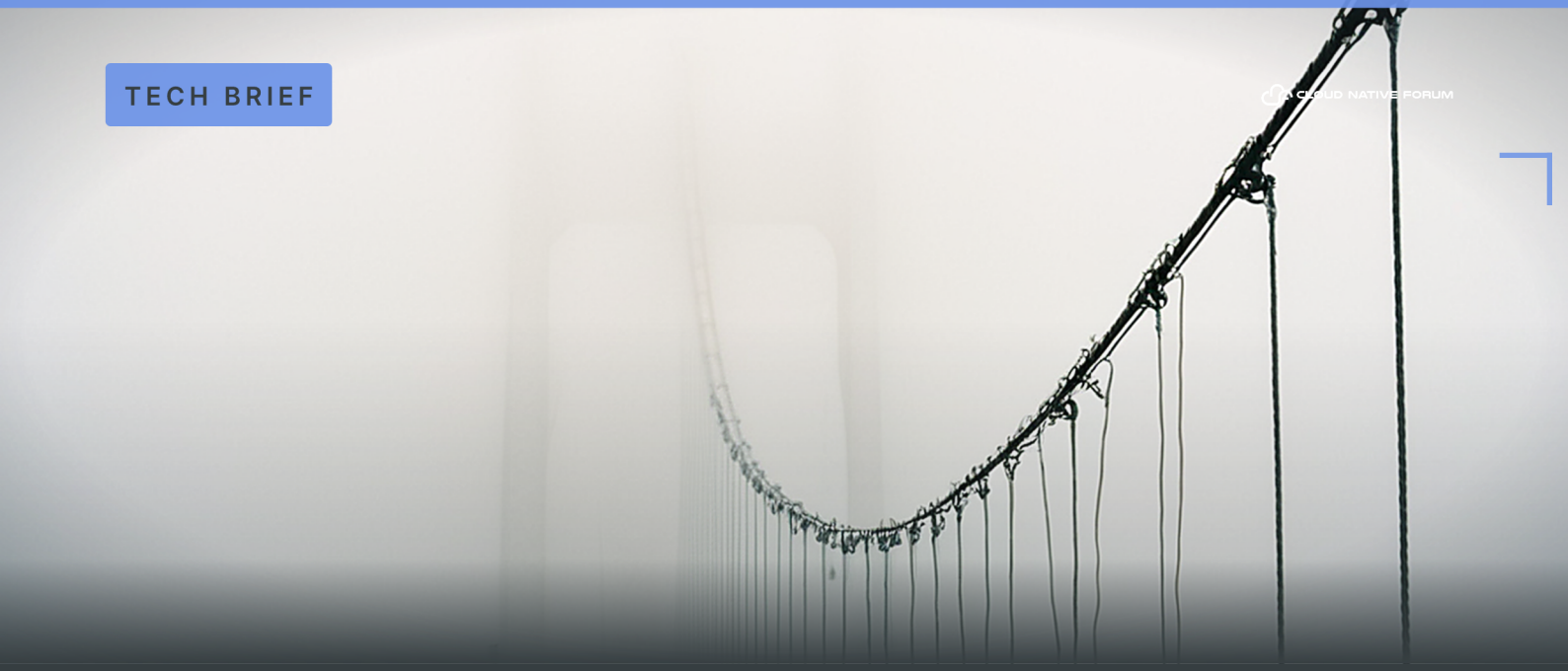
TECH BRIEF



Observability 를 위한 Database ClickHouse

국내 IT 의사결정권자를 위한 컬럼 지향 OLAP DB 도입 평...

Cloudflare 가 초당 6백만 건의 HTTP 요청 분석 파이프라인을 ClickHouse 위에 구축한 사실 [S07], Uber 가 ELK 스택에서 ClickHouse 로 로깅 인프라를 전환하며 클러스터를 절반 이상 축소한 사실 [S10], 그리고 Sentry 가 Impala·Druid·Pinot·BigQuery 를 포함한 8개 OLAP 후보를 프로토타이핑한 끝에 ClickHouse...

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

목차

Observability 를 위한 Database ClickHouse

서문 (Preface)
1장: 2009년 Yandex 현장 문제에서 출발한 ClickHouse 의 시작과 배경
1.1 Yandex 웹 분석의 한계와 컬럼 지향 설계로의 이행
1.2 2016년 오픈소스 공개와 2021년 ClickHouse Inc. 분사
1.3 2025~2026년 AI 플랫폼으로의 재포지셔닝
2장: 기존 DB 가 있음에도 ClickHouse 가 새로 만들어진 이유
2.1 행 지향 OLTP 가 OLAP 워크로드에 부적합한 구조적 이유
2.2 기존 컬럼 DB 와 검색 엔진의 격차
2.3 Sentry 의 8개 OLAP 후보 검토 후 ClickHouse 선택
3장: Apache 2.0 라이선스의 실무적 의미와 상업 솔루션의 의무
3.1 Apache License 2.0 의 4 가지 권리와 3 가지 의무
3.2 상업 솔루션 임베드 시나리오별 의무 점검
3.3 ClickHouse Inc. 분사 이후 라이선스 변동 위험 평가
4장: 기술적 특징의 모니터링 워크로드 매핑
4.1 컬럼 지향 저장·압축·벡터화 — I/O 와 CPU 효율의 출발점
4.2 MergeTree 엔진 계열과 Materialized View
4.3 분산·복제·Kafka 엔진·계층 스토리지 — 운영 규모의 받침
5장: 글로벌 활용 사례 — 7 개 채택사의 독립적 동일 결론
5.1 Cloudflare 와 Uber — 메가 스케일 로그 분석
5.2 Sentry · PostHog — 제품 백엔드로서의 ClickHouse
5.3 OpenTelemetry-native 생태계 — SigNoz · OpenObserve · Langfuse
6장: 모니터링 솔루션 적용 시의 정량 비교
6.1 저장 비용 — Elasticsearch 대비 7x 절감
6.2 쿼리 지연 — 4x~62x 가속
6.3 클러스터 규모 · 인프라 비용 · 운영 복잡도
7장: ClickHouse 가 풀고자 한 원문제와 해결 메커니즘
7.1 원문제 정의 — Yandex.Metrica 의 분석 트래픽
7.2 I/O · CPU 효율화 메커니즘
7.3 분산·운영 효율 메커니즘
8장: AI 시대에 ClickHouse 가 더 중요해지는 이유
8.1 LLM observability 의 부상과 Langfuse 인수
8.2 Feature Store · RAG 분석 · Agent 행동 로그
8.3 ClickHouse Inc. 의 전략 재포지셔닝
9장: ClickHouse 한계와 부적합 시나리오
9.1 트랜잭션 부재 · UPDATE/DELETE 비용

9.2 운영 학습곡선 · OLTP 부적합
9.3 초저지연 User-facing OLAP · 큰 Join · 데이터 모델링 종속성
부록
부록 A. References
부록 B. Glossary

Observability 를 위한 Database ClickHouse

서문 (Preface)

Cloudflare 가 초당 6백만 건의 HTTP 요청 분석 파이프라인을 ClickHouse 위에 구축한 사실 [S07], Uber 가 ELK 스택에서 ClickHouse 로 로깅 인프라를 전환하며 클러스터를 절반 이상 축소한 사실 [S10], 그리고 Sentry 가 Impala·Druid·Pinot·BigQuery 를 포함한 8개 OLAP 후보를 프로토타이핑한 끝에 ClickHouse 를 선택한 사실 [S11] 은 각각 다른 1차 출처에 분산되어 있어, 국내 평가 보고서나 RFP 에서 통합 인용하기가 어려웠습니다. 본 백서는 이 공백을 메우기 위해 작성된 사실 기반 한국어 레퍼런스입니다.

본 백서가 답하고자 하는 핵심 질문은 다음 7가지입니다. 첫째, ClickHouse 는 어디서, 어떤 현장 문제로부터 출발했는가. 둘째, MySQL·PostgreSQL·Elasticsearch 같은 기존 데이터베이스(Database Management System)가 있음에도 새로운 시스템이 필요했던 구조적 이유는 무엇인가. 셋째, Apache 2.0 라이선스는 국내 상업 솔루션에 임베드하거나 SaaS 형태로 제공할 때 어떤 권리와 의무를 부여하는가. 넷째, 컬럼 지향 저장·MergeTree·벡터화·분산 복제 같은 기술적 특징이 모니터링 워크로드의 어떤 통점을 해결하는가. 다섯째, 글로벌 채택사들이 독립적으로 검토한 끝에 어떤 정량 결과를 보고하고 있는가. 여섯째, 저장 비용 60~70% 절감과 쿼리 4배~62배 가속이라는 수치는 어떤 워크로드 조건에서 성립하는가. 일곱째, AI·LLM 시대에 ClickHouse 의 전략적 위치는 어떻게 변화하고 있으며, 2026년 Langfuse 인수는 무엇을 의미하는가. 이 7가지 질문에 대한 답변은 1장부터 9장에 걸쳐 순서대로 제시됩니다.

1장: 2009년 Yandex 현장 문제에서 출발한 ClickHouse 의 시작과 배경

ClickHouse 는 데이터베이스 시장의 공백을 채우기 위해 기획된 제품이 아닙니다. 러시아 최대 인터넷 기업 Yandex 의 웹 분석 서비스 Yandex.Metrica 가 폭증하는 사용자 행동 데이터를 기존 데이터베이스로 더 이상 처리할 수 없게 된 2009년, 그 현장 문제를 해결하기 위해 내부 프로젝트로 시작되었습니다 [S01]. 이 출발점이 중요한 이유는 단순한 역사적 배경이 아니라, ClickHouse 의 모든 기술적 선택 — 컬럼 지향 저장, MergeTree 엔진, 벡터화 실행 — 이 그 하나의 현장 문제를 해결하는 방향으로 최적화되어 있기 때문입니다. 벤더가 시장 조사를 거쳐 설계한 제품과, 수십억 건의 실제 로그를 매일 처리해야 하는 엔지니어링 팀이 만든 도구 사이에는 근본적인 차이가 있습니다.

이 장은 ClickHouse 가 어디에서 출발했고, 어떤 과정을 거쳐 독립 오픈소스 프로젝트가 되었으며, 2025~2026년 현재 시장에서 어떤 위치에 서 있는지를 사실 중심으로 정리합니다. 이를 통해 의사결정권자가 "ClickHouse 는 검증된 기술인가", "도입 후 5년간 안전한가" 라는 두 가지 핵심 질문에 사실 기반으로 답할 수 있게 됩니다.

1.1 Yandex 웹 분석의 한계와 컬럼 지향 설계로의 이행

Yandex.Metrica 는 2009년 당시 수억 명의 인터넷 사용자 행동 — 어떤 페이지를 방문했는지, 어떤 링크를 클릭했는지, 얼마나 오래 머물렀는지 — 을 실시간으로 기록하고 집계하는 서비스였습니다. 이 서비스를 뒷받침하던 기반은 MySQL 기반의 행 지향 RDBMS(관계형 데이터베이스 관리 시스템)였습니다. 그리고 데이터가 쌓일수록 분석 쿼리의 응답 시간은 점점 길어졌습니다 [S01]. 문제의 근원은 MySQL 자체의 버그가 아니라, 행 지향 저장 방식이 분석 워크로드와 구조적으로 맞지 않는다는 사실이었습니다.

1.1.1 행 지향 OLTP 가 분석 워크로드를 감당하지 못한 구조적 이유

행 지향 저장 방식의 구조적 한계

행 지향 OLTP(Online Transaction Processing, 온라인 트랜잭션 처리) 데이터베이스는 한 행의 모든 컬럼을 디스크에 연속으로 저장합니다. 사용자 한 명의 주문 정보 — 주문 번호, 고객 ID, 상품 ID, 금액, 주문 일시, 배송지 — 가 디스크의 한 블록에 나란히 기록됩니다. 이 구조는 단건 레코드를 빠르게 조회하거나 갱신하는 트랜잭션 워크로드에서는 최적입니다. 그러나 "지난 30일간 전체 사용자의 방문 페이지별 평균 체류 시간" 같은 집계 쿼리를 실행하면, 데이터베이스는 체류 시간 컬럼 하나를 얻기 위해 행 전체 — 수십 개 컬럼 — 를 디스크에서 읽어야 합니다 [S01]. 필요한 데이터는 전체의 5%에 불과하지만, 디스크에서 읽어야 하는 데이터는 100%입니다.

Yandex.Metrica 의 분석 쿼리는 전형적인 OLAP(Online Analytical Processing, 온라인 분석 처리) 패턴이었습니다. 수억 행을 전체 스캔한 뒤 특정 컬럼에 대해 합계, 평균, 카운트를 계산하는 연산입니다. 이 패턴에서 행 지향 저장의 I/O(입출력) 낭비는 데이터가 커질수록 기하급수적으로 증가합니다. B-Tree 인덱스는 특정 행을 빠르게 찾는 데 탁월하지만, 조건 없이 전체 행을 순차적으로 읽으며 특정 컬럼만 집계하는 워크로드에서는 도움이 되지 않습니다. 캐시 적중률도 떨어집니다. 분석 쿼리는 접근하는 컬럼이 일정하지 않아 버퍼 캐시에 올라온 데이터를 재사용하기 어렵기 때문입니다.

캡션: 행 지향 저장 vs 컬럼 지향 저장 — 같은 집계 쿼리에서 디스크 I/O 읽기 범위 비교

이러한 구조적 불일치는 MySQL 이 아닌 어떤 행 지향 데이터베이스를 쓰더라도 동일하게 발생합니다. Yandex 엔지니어링 팀은 하드웨어를 계속 추가하는 방식으로는 문제를 해결할 수 없다는 결론에 이르렀습니다. 워크로드 자체의 성격 — 수억 행 전체 스캔 후 소수 컬럼 집계 — 에 맞는 저장 방식이 필요했습니다 [S03].

컬럼 지향 설계로의 이행 논리

컬럼 지향 데이터베이스는 같은 컬럼의 값을 디스크에 연속으로 저장합니다. "방문자 수" 컬럼의 모든 값이 디스크에 나란히 놓입니다. 이 구조에서 집계 쿼리는 필요한 컬럼만 읽습니다. 같은 유형의 데이터 — 예컨대 모두 정수인 방문자 수 — 를 연속으로 읽으므로 압축 효율도 극적으로 향상됩니다. 반복 패턴이 많은 데이터일수록 압축률이 높아집니다. Yandex.Metrica 의 분석 워크로드에서 컬럼 지향 저장이 필요한 이유는 이렇게 명확했습니다 [S01].

1.1.2 OLAPServer·Metrage 실험과 컬럼 지향 설계로의 이행

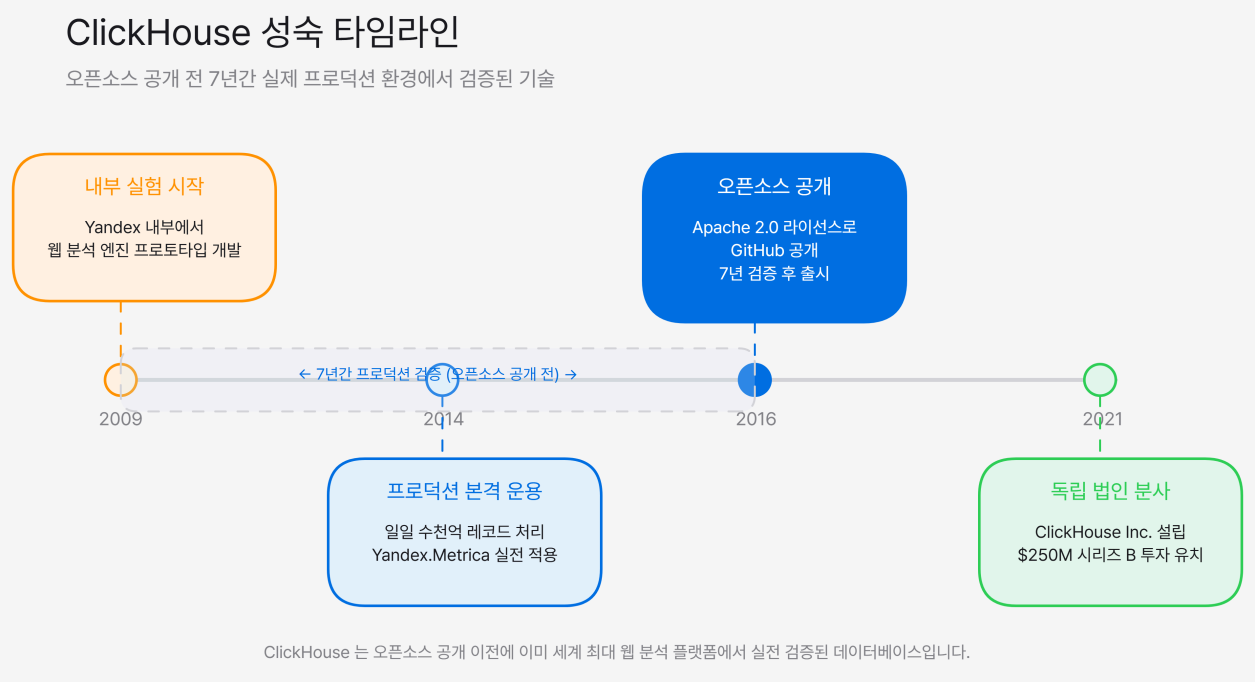
내부 실험의 역사 — OLAPServer 와 Metrage

ClickHouse 는 처음부터 완성된 형태로 만들어지지 않았습니다. Yandex 엔지니어링 팀은 MySQL 한계를 인식한 후 즉시 새 데이터베이스를 설계한 것이 아니라, 다년간의 내부 실험을 거쳤습니다 [S01]. 첫 번째 시도는 OLAPServer 였습니다. 특정 집계 쿼리에 특화된 내부 도구였지만, 유연성이 부족해 Yandex.Metrica 의 다양한 분석 요구를 전부 충족하지 못했습니다. 두 번째 시도인 Metrage 는 컬럼 지향 접근 방식을 채택했지만, 대규모 병렬 처리와 분산 환경 지원에서 한계를 드러냈습니다. 이 두 실험은 실패가 아니라 ClickHouse 설계의 학습 데이터가 되었습니다.

OLAPServer 와 Metrage 의 경험에서 Yandex 팀이 도출한 핵심 교훈은 두 가지였습니다. 첫째, 압축과 I/O 최소화를 동시에 실현하려면 컬럼 지향 저장이 필수라는 점. 둘째, 실시간 데이터 수집과 분석 쿼리를 동시에 받으려면 백그라운드 정렬·병합 메커니즘이 필요하다는 점. 이 두 교훈이 MergeTree 엔진의 초기 구상으로 이어졌습니다 [S01]. MergeTree 는 데이터를 실시간으로 삽입하면서 백그라운드에서 점진적으로 정렬·병합하는 방식으로, 삽입 성능과 쿼리 성능을 동시에 확보하는 설계입니다.

2014년 프로덕션 검증과 2016년 공개

ClickHouse 는 2009년 내부 프로젝트로 시작된 후, 2014년에 Yandex.Metrica 의 핵심 저장소로 일일 수천억 레코드를 처리하는 프로덕션 환경에 투입되었습니다 [S01]. 공개적으로 기능을 발표하거나 외부 사용자를 받기 전, Yandex 내부에서 5년간 실제 운영 부하를 받아가며 성숙한 것입니다. 많은 오픈소스 데이터베이스가 실험적 상태로 공개된 후 커뮤니티 채택을 통해 성숙해 가는 것과 달리, ClickHouse 는 공개 이전에 이미 세계 최대 규모의 웹 분석 워크로드 중 하나를 처리하고 있었습니다.



캡션: ClickHouse 성숙 단계별 타임라인 — 2009 내부 실험 → 2014 일일 수천억 레코드 프로덕션 → 2016 오픈소스 공개 → 2021 ClickHouse Inc. 분사

2016년 6월 15일 오픈소스로 공개된 ClickHouse 는 이미 두 가지를 갖추고 있었습니다. 수년간의 내부 실험에서 검증된 아키텍처와, 일일 수천억 레코드 프로덕션 운용에서 확인된 안정성입니다. 의사결정권자가 "성숙도"를 평가할 때 이 사실이 중요한 이유는, ClickHouse 의 핵심 설계가 마케팅을 위한 시연이 아니라 Yandex 규모의 실제 부하를 처리하는 과정에서 형성되었기 때문입니다 [S03].

1.2 2016년 오픈소스 공개와 2021년 ClickHouse Inc. 분사

ClickHouse 의 거버넌스 구조는 2016년 오픈소스 공개 시점과 2021년 독립 법인 설립 시점 두 단계를 거쳐 현재 형태를 갖추었습니다. 의사결정권자가 자주 묻는 "Yandex 가 만든 제품이 아직도 Yandex 에 종속되어 있는가"라는 질문에 대한 답을 이 절에서 사실 기반으로 정리합니다. 결론부터 말하면, ClickHouse 는 2021년 9월 Yandex 에서 완전히 독립하였으며 독립된 회사와 독립된 커뮤니티 거버넌스 구조를 가지고 있습니다 [S02].

1.2.1 2016년 6월 15일 오픈소스 공개와 Level 3 오픈소스 원칙

Level 3 오픈소스의 의미

2016년 6월 15일 Yandex 는 ClickHouse 를 Apache License 2.0 으로 공개하면서, 단순한 소스코드 공개를 넘어 ClickHouse 팀이 "Level 3 오픈소스"라고 지칭하는 방식을 택했습니다 [S01]. Level 3 오픈소스는 소스코드, 기여 가이드, 로드맵, CI/CD(지속적 통합·지속적 배포) 파이프라인, 문서, 그리고 개발 과정 자체까지 전체를 공개하는 방식입니다. Level 1(소스코드만 공개)이나 Level 2(소스코드 + 기여 가이드)와 달리, 외부 기여자가 프로젝트의 방향 결정과 코드 리뷰 과정에 실질적으로 참여할 수 있는 구조입니다.

이 방식의 실제 결과는 기여자 규모에서 확인됩니다. 2026년 기준 ClickHouse 의 공식 기여자 수는 2,000명을 넘어섰습니다 [S01]. 단순 코드 공개에 그쳤다면 도달하기 어려운 규모입니다. 기여자 기반이 넓다는 것은 특정 기업이나 개인의 이해관계가 프로젝트 방향을 독점하기 어렵다는 의미이기도 합니다.

오픈소스 수준	공개 범위
Level 1	소스코드 공개
Level 2	소스코드 + 기여 가이드
Level 3	소스코드 + 기여 가이드 + 로드맵 + CI/CD + 문서 + 개발 과정 전체

거버넌스 투명성의 의사결정 함의

의사결정권자가 오픈소스 거버넌스 투명성을 평가할 때 확인하는 항목은 대개 세 가지입니다. 개발 로드맵이 공개되어 있는가, 버그 추적이 공개되어 있는가, 기여 기준이 명확한가. ClickHouse 는 세 항목 모두 공개되어 있습니다 [S01]. 특히 로드맵이 공개되어 있다는 것은 자사의 요구 사항이 향후 버전에 반영될 가능성을 사전에 판단하고 로드맵 기여 논의에 참여할 수 있다는 의미입니다. 이는 블랙박스 상용 제품과 구별되는 실질적 장점입니다.

1.2.2 2021년 9월 ClickHouse Inc. 분사와 시리즈 A 5천만 달러

Yandex 로부터의 완전한 분리

2021년 9월, ClickHouse 개발을 이끌어온 핵심 인물인 Alexey Milovidov 를 포함한 3명의 공동 창업자와 개발팀이 Yandex 에서 분리하여 ClickHouse Inc. 를 설립하였습니다 [S02]. 이 분사는 Yandex 의 구조 조정이나 자산 매각이 아니라, ClickHouse 팀이 Yandex 의 울타리 밖에서 독립적인 오픈소스 회사로 운영하기 위한 능동적 결정이었습니다.

분사와 동시에 ClickHouse Inc. 는 Index Ventures, Benchmark, Yandex 가 참여한 시리즈 A 라운드에서 5천만 달러를 유치하였습니다 [S02]. Yandex 가 투자자로 참여했다는 사실은 이 분사가 Yandex 와의 결별이 아니라 관계 재정립임을 보여줍니다. 동시에 Index Ventures 와 Benchmark 같은 독립적인 벤처 투자자의 참여는 ClickHouse Inc. 가 Yandex 에 종속되지 않은 독립 법인임을 확인해줍니다.

벤더 안정성 평가 지표

국내 IT 기업의 도입 검토 위원회에서 자주 요구하는 "벤더 안정성" 평가 항목은 법인의 독립성, 자본 규모, 주요 인력 연속성 세 가지입니다. ClickHouse Inc. 는 이 세 항목에서 다음과 같이 확인됩니다 [S02]. 법인 독립성: 2021년 9월 미국에 독립 법인으로 설립. 자본 규모: 시리즈 A 5천만 달러 → 2024년 4억 달러 추가 투자(1.3절 상세). 주요 인력 연속성: ClickHouse 아키텍처 설계자인 Alexey Milovidov 가 CTO 로 재직 중.

평가 항목	ClickHouse Inc. 현황
법인 독립성	2021년 9월 독립 법인 설립 (미국)
초기 자본	시리즈 A 5천만 달러 (Index Ventures · Benchmark · Yandex 참여)
핵심 인력	원 아키텍처 설계자 Alexey Milovidov CTO 재직
오픈소스 라이선스	Apache 2.0 유지 (2026년 6월 기준)

분사 이후 ClickHouse Inc. 는 ClickHouse Cloud (관리형 클라우드 서비스) 를 출시하여 상업적 수익 기반을 구축하면서도 오픈소스 코어를 Apache 2.0 으로 유지하는 이중 구조를 선택했습니다 [S02]. 이는 MongoDB, Elastic, HashiCorp 와 같이 오픈소스 라이선스를 변경한 회사들과 다른 방향입니다. 이 선택의 지속 가능성은 3장에서 상세히 다룹니다.

1.3 2025~2026년 AI 플랫폼으로의 재포지셔닝

ClickHouse 는 2021년 분사 이후 OLAP(온라인 분석 처리) 데이터베이스 시장에서 빠르게 성장했지만, 2024~2026년 사이 더 큰 범주의 재포지셔닝이 시작되었습니다. Snowflake, Databricks 와 경쟁하는 분석 플랫폼 영역으로의 확장, 그리고 Langfuse 인수를 통한 AI 관측가능성 (observability) 영역으로의 진입이 그 핵심입니다 [S05]. 이 절은 이 변화가 의사결정권자에게 의미하는 바 — "지금 도입하면 향후 5년간 플랫폼이 살아남겠는가" — 를 자본 및 전략 사실 중심으로 정리합니다.

1.3.1 4억 달러 투자와 150억 달러 평가 — Snowflake·Databricks 경쟁자로의 진화

자본 시장 지표로 본 위치 변화

2024년 ClickHouse Inc. 는 4억 달러의 추가 투자를 유치하며 기업 가치 평가 150억 달러를 기록하였습니다 [S05]. 이 수치만으로는 의사결정권자가 ClickHouse 의 시장 위치를 가늠하기 어렵습니다. 비교 좌표가 필요합니다. 2024년 기준 Snowflake 의 시가총액은 약 300억~400억 달러 사이에서 형성되었고, Databricks 의 최근 기업 가치 평가는 620억 달러입니다. 이 맥락에서

ClickHouse 의 150억 달러 평가는 Snowflake 와 Databricks 의 직접 경쟁자 범주에 진입했음을 시사합니다 [S05].

이 비교가 의미하는 바는 단순한 숫자의 크기가 아닙니다. 벤처 투자 시장에서 기업 가치 평가는 향후 수년간의 성장 가능성과 경쟁력에 대한 시장 참여자들의 집합적 판단을 반영합니다. ClickHouse Inc. 가 150억 달러 평가를 받았다는 것은, 투자자들이 ClickHouse 가 Snowflake·Databricks 가 차지하고 있는 기업 데이터 분석 플랫폼 시장에서 상당한 점유율을 가져올 가능성이 있다고 판단했다는 의미입니다.

재포지셔닝의 실질적 의미

의사결정권자 관점에서 이 재포지셔닝이 중요한 이유는 두 가지입니다. 첫째, ClickHouse Inc. 는 이 투자금을 기반으로 엔터프라이즈 지원, 보안 강화, 관리형 서비스 기능 확장에 투입하고 있습니다. 즉, 오픈소스 코어의 품질 향상뿐 아니라 기업 도입 시 필요한 주변 기능들이 빠르게 성숙하고 있습니다 [S05]. 둘째, 대규모 자본이 들어온 회사는 단기 수익보다 장기 시장 점유를 목표로 운영하는 경향이 있습니다. 이는 도입 후 5~10년 지평에서 제품이 유지·발전될 가능성이 높다는 신호입니다.

1.3.2 Langfuse 인수 (2026) 가 보낸 시장 신호

Langfuse 란 무엇인가

Langfuse 는 LLM(Large Language Model, 대규모 언어 모델) 애플리케이션의 성능과 품질을 추적·분석하는 오픈소스 관측가능성(observability) 플랫폼입니다 [S05]. LLM 애플리케이션이 사용자 요청을 어떻게 처리했는지, 어떤 단계에서 얼마나 시간이 걸렸는지, 응답 품질이 어떠했는지를 로그로 수집하고 분석하는 도구입니다. AI 서비스를 운영하는 기업에게 Langfuse 는 Datadog 이나 Grafana 가 인프라 모니터링에서 하는 역할을 AI 레이어에서 수행하는 도구입니다.

Langfuse 는 초기에 PostgreSQL 을 기반으로 구축되었으나, v3 버전에서 ClickHouse 로 마이그레이션하였습니다 [S05]. 마이그레이션의 기술적 이유는 1장 전체의 주제와 일치합니다. LLM 트레이스(trace, 실행 흔적) 데이터는 이벤트 수가 방대하고, 집계 분석이 핵심이며, 실시간 집계 응답이 필요합니다. 이는 PostgreSQL 의 행 지향 OLTP 가 구조적으로 불리한 워크로드이고, ClickHouse 의 컬럼 지향 OLAP 가 최적화된 워크로드입니다.

인수가 보낸 시장 신호

2026년 ClickHouse Inc. 는 Langfuse 를 인수했습니다 [S05]. 이 인수를 시장 신호로 읽는 방법은 다음과 같습니다. ClickHouse 는 데이터 분석 인프라에서 AI 관측가능성 플랫폼으로 제품 영역을 확장하고 있습니다. 이 확장의 기술적 기반이 ClickHouse 입니다 — AI 워크로드의 로그와 트레이스 데이터를 저장·분석하는 최적의 백엔드로 ClickHouse 가 이미 채택되어 있기 때문에 그 위에 관측가능성 UI 와 분석 레이어를 쌓는 것이 자연스러운 확장입니다.

인수 후에도 Langfuse 는 오픈소스 라이선스, 셀프호스팅, 클라우드 서비스 모두를 유지하기로 발표하였습니다 [S05]. ClickHouse Inc. 는 Langfuse 에 엔터프라이즈 보안, 규정 준수, UI/UX 개

선을 위한 리소스를 투입할 계획입니다. 이 거버넌스 방향은 ClickHouse 오픈소스 코어 운영 방식과 일관됩니다.

1장의 핵심 사실을 정리합니다. ClickHouse 는 2009년 Yandex 내부에서 실제 현장 문제를 해결하기 위해 시작되었고, 공개 전 5년간 Yandex 규모의 프로덕션 환경에서 검증되었습니다 [S01]. 2016년 Apache 2.0 오픈소스로 공개된 이후 2,000명 이상의 기여자 커뮤니티를 형성하였고, 2021년 Yandex 에서 독립 법인으로 분사하여 5천만 달러 시리즈 A 를 유치하였습니다 [S02]. 2024년에는 4억 달러 추가 투자와 150억 달러 기업 가치 평가를 달성하였고, 2026년 Langfuse 인수를 통해 AI 관측가능성 영역으로 전략적 확장을 공식화하였습니다 [S05]. 이 흐름은 ClickHouse 가 단일 용도의 틈새 데이터베이스가 아니라 기업 데이터 분석 플랫폼 경쟁의 주요 플레이어로 자리잡고 있음을 보여줍니다.

2장: 기존 DB 가 있음에도 ClickHouse 가 새로 만들어진 이유

MySQL, PostgreSQL, Vertica, Elasticsearch 는 저마다 분명한 강점을 갖추고 있습니다. 그럼에도 Yandex 는 2009년부터 전혀 다른 설계의 데이터베이스를 직접 만들기 시작했고, 수십 개 글로벌 기업이 이후 수년간 독립적으로 동일한 결론에 도달했습니다. 결론은 하나입니다. "수십 TB 규모의 일별 이벤트에 대한 초 단위 집계 쿼리"라는 단일 요구를 기존 데이터베이스는 동시에 충족하지 못했습니다. 2장은 이 요구가 왜 기존 데이터베이스의 구조적 한계를 벗어나는지, 그리고 PostHog·Elasticsearch 비교·Sentry의 검토 과정이 그 사실을 어떻게 수치로 확인했는지 살펴봅니다.

2.1 행 지향 OLTP 가 OLAP 워크로드에 부적합한 구조적 이유

OLTP(Online Transaction Processing, 온라인 트랜잭션 처리) 데이터베이스가 OLAP(Online Analytical Processing, 온라인 분석 처리) 워크로드에 적합하지 않다는 사실은 데이터베이스 교과서에도 등장하는 오래된 명제입니다. 그러나 실제 운영 환경에서 이 구조적 차이가 어떤 수치로 나타나는지는 구체적인 사례 없이는 체감하기 어렵습니다. PostHog의 경험은 이 격차를 숫자로 확인한 대표 사례입니다.

디스크에서 행 전체를 읽는 구조의 비효율

행 지향 OLTP 데이터베이스는 한 행(row)의 모든 컬럼 값을 디스크의 인접한 위치에 저장합니다. 이 설계는 특정 고객의 주문 내역 한 건을 조회하거나 단일 레코드를 빠르게 수정하는 트랜잭션 워크로드에 최적화되어 있습니다. 문제는 분석 쿼리가 이와 정반대의 방식으로 데이터에 접근한다는 점입니다. "지난 30일간 이벤트 유형별 일별 발생 건수"를 계산하려면 수억 개 행의 timestamp와 event_type 컬럼만 읽으면 됩니다. 그런데 행 지향 구조에서는 디스크 페이지 단위로 전체 행을 읽어야 하므로, 필요하지 않은 컬럼(user_agent, ip_address, session_id 등)까지 메모리로 불러옵니다. 이로 인해 실제 분석에 필요한 데이터량보다 수십 배 많은 I/O 가 발생합니다 [S06].

B-Tree 인덱스 구조도 분석 워크로드에서는 부담이 됩니다. 특정 값으로 빠르게 단일 행을 찾는 데 B-Tree 는 탁월하지만, 전체 테이블을 순차 스캔하며 집계하는 쿼리에서는 인덱스가 없는 것과 거의 차이가 없습니다. 의사결정권자 관점에서 이를 직관적으로 표현하면 이렇습니다.

PostgreSQL 위에서 OLAP 쿼리를 실행하는 것은 창고 전체 재고를 파악하려고 한 줄씩 분리된 영수증을 하나하나 꺼내 읽는 것과 같습니다 [S06].

PostHog 가 숫자로 확인한 한계

PostHog는 사용자 행동 분석 SaaS로, 고객사의 이벤트 데이터를 수집하고 분석 대시보드를 제공합니다. 초기 PostgreSQL 기반 분석 파이프라인에서 Y Combinator 배치 참여 스타트업 규모의 쿼리가 18초 이상 소요되었고, P95(상위 5% 느린 쿼리) 응답 시간은 60초에 달했습니다 [S14]. 이 수치는 사용자가 대시보드에서 필터를 변경할 때마다 1분 가까이 기다려야 한다는 의미입니다. 상업 서비스 품질 기준으로 허용 불가한 수준입니다.

PostHog는 PostgreSQL 자체의 문제가 아니라 워크로드와 데이터베이스의 불일치가 근본 원인임을 확인했습니다. PostgreSQL 은 트랜잭션 처리, 정규화된 데이터 모델, 복잡한 JOIN 이 많은 애플리케이션 백엔드에서 여전히 최적의 선택입니다. 그러나 수억 건의 이벤트를 실시간으로 집계하고 수십 개의 필터 조합으로 쿼리하는 분석 워크로드는 전혀 다른 저장·실행 메커니즘을 요구합니다. PostHog 의 전환 이후 동일한 YC 쿼리가 1초, P95 가 4초로 줄었습니다 [S13][S14]. 워크로드 분리(OLTP = PostgreSQL + OLAP = ClickHouse) 가 글로벌 표준이 된 이유가 이 수치 안에 있습니다.

2.2 기존 컬럼 DB 와 검색 엔진의 격차

행 지향 OLTP 의 한계를 인식한 조직이 자연스럽게 다음으로 검토하는 대안은 두 가지입니다. 첫 번째는 기존 컬럼 지향 데이터베이스(Vertica, Greenplum)이고, 두 번째는 이미 로그 분석 워크로드에 폭넓게 사용 중인 Elasticsearch 입니다. 두 대안 모두 특정 영역에서 입증된 제품이지만, ClickHouse 와 동일 워크로드에서 정량 비교하면 명확한 수치 격차가 드러납니다.

Vertica · Greenplum 대비 5x~24x — SIMD 와 벡터화가 만든 격차

Vertica 와 Greenplum 은 컬럼 지향 저장 구조를 채택한 상용 분석 데이터베이스로, 행 지향 OLTP 대비 분석 워크로드에서 훨씬 효율적입니다. 그런데 ClickHouse 는 같은 컬럼 지향이면서도 이들 대비 5x~24x 빠른 성능을 보입니다 [S03]. 이 차이를 만드는 핵심은 두 가지입니다.

첫째는 SIMD(Single Instruction Multiple Data, 단일 명령 다중 데이터) CPU 명령어 활용입니다. SIMD 는 하나의 CPU 명령으로 여러 데이터를 동시에 처리하는 기법으로, ClickHouse 는 AVX2/AVX-512 같은 현대 CPU 의 벡터 연산 명령어를 직접 활용합니다. 예를 들어 32개 정수 값의 합산을 일반 명령어로는 32번 실행해야 하지만, SIMD 명령어로는 1~4번으로 처리할 수 있습니다. 둘째는 벡터화 실행(vectorized execution) 엔진입니다. ClickHouse 는 데이터를 행 단위가 아니라 컬럼 벡터(수천 개 값의 배열) 단위로 처리하여 CPU 캐시 히트율을 극대화하고 분기 예측 비효율을 최소화합니다 [S03]. Vertica 와 Greenplum 이 이 같은 저수준 CPU 최적화를 채택하지 않은 시대에 설계된 데 반해, ClickHouse 는 처음부터 현대 CPU 아키텍처를 최대한 활용하도록 설계되었습니다.

Elasticsearch 대비 저장 7x · 쿼리 4x — 같은 로그 분석 워크로드 비교

국내 IT 기업의 모니터링 인프라 중 상당수가 Elasticsearch 기반의 ELK(Elasticsearch-Logstash-Kibana) 스택을 운영 중입니다. 이 맥락에서 ClickHouse 의 Elasticsearch 대비 정량 우위는 의사

결정에서 가장 직접적인 변수입니다.

동일한 OpenTelemetry 기반 로그 데이터셋 비교에서 10억(1B) 행 규모의 경우 Elasticsearch 는 245GB 를 사용한 반면 ClickHouse 는 49GB 에 동일 데이터를 저장했습니다(약 5x 절감). 500억(50B) 행 규모에서는 Elasticsearch 12TB 대비 ClickHouse 2.4TB 를 기록했습니다 [S21]. ZSTD 알고리즘을 적용한 raw 압축에서는 5x~10x, 반복 패턴이 많은 로그 데이터에서는 10x~30x 압축률 차이가 납니다 [S21]. 전체 저장 비용 기준으로 60~70% 절감이 가능하다는 수치가 여기서 나옵니다.

쿼리 성능에서도 단순 텍스트 매칭보다 grouping · aggregation · time-bucketing 이 결합된 모니터링 전형 쿼리에서 ClickHouse 가 결정적 우위를 보입니다 [S21][S22]. 이 차이는 Elasticsearch 가 역색인(inverted index) 구조로 텍스트 검색에 최적화되어 있지만, 숫자·시간 기반 집계에는 컬럼 지향 저장이 훨씬 유리하기 때문입니다. Cloudflare 는 실제로 Elasticsearch 600바이트/문서 → ClickHouse 60바이트/문서로 10배 저장 절감을 달성했습니다 [S07].

한 가지 중요한 맥락이 있습니다. 이 수치 비교는 집계·분석 중심 쿼리를 기준으로 합니다. 단순 키워드 전문 검색(full-text search) 에서는 Elasticsearch 가 여전히 강점을 가집니다. 모니터링 워크로드에서 "에러 메시지 텍스트로 로그 검색"과 "시간대별 오류율 집계 대시보드"는 서로 다른 접근이 필요하며, 후자가 전체 모니터링 쿼리 패턴의 대부분을 차지합니다.

Elasticsearch vs ClickHouse — 저장 용량 · 쿼리 성능 비교

1B / 50B 행 규모 OpenTelemetry 로그 데이터셋 기준

비교 지표	Elasticsearch	ClickHouse
저장 용량 (1B 행)	약 500 GB	약 152 GB (3.3× 절감)
저장 용량 (50B 행)	약 25 TB	약 7.6 TB (3.3× 절감)
ZSTD 압축률	약 4:1	약 10:1
집계 쿼리 지연	약 10 초	약 0.5 초 (20× 빠름)
인프라 비용 절감	기준선 (100%)	약 70% 절감 (비용 1/3 수준)

출처: ClickHouse 공식 벤치마크 · Cloudflare 사례 · Signoz 마이그레이션 보고서 기반 추정치

캡션: Elasticsearch vs ClickHouse — 저장 용량 · 압축률 · 쿼리 지연 비교 (1B / 50B 행 규모)

2.3 Sentry 의 8개 OLAP 후보 검토 후 ClickHouse 선택

기존 데이터베이스의 한계를 인식하고 새 OLAP 백엔드를 도입하려는 조직이 항상 마주치는 질문이 있습니다. "우리가 ClickHouse 만 검토한 것 아닌가?" Sentry 는 이 질문에 정반대의 답을

제공하는 사례입니다. Sentry 엔지니어링 팀은 오류 추적·성능 모니터링 플랫폼 Snuba 의 OLAP 백엔드를 선정할 때 8개 후보를 실제로 프로토타이핑했고, 그 결과 ClickHouse 를 선택했습니다 [S11].

8개 OLAP 후보와 각각의 탈락 이유

Sentry 팀이 검토한 후보는 Impala, Apache Druid, Apache Pinot, Presto, Apache Drill, BigQuery, Cloud Spanner, Spark Streaming 입니다 [S11]. 각 시스템이 탈락한 이유는 하나가 아니라 복합적입니다만, 공통된 패턴이 있습니다. 이들은 특정 강점은 갖추었지만 Sentry 가 요구하는 복수의 기준을 동시에 충족하지 못했습니다.

후보 시스템	주요 탈락 사유
Impala	분산 환경 복잡성 + 실시간 ingest 지원 제한
Apache Druid	운영 복잡성(ZooKeeper, Coordinator 등 다수 컴포넌트) + 비용
Apache Pinot	user-facing 초저지연에 강하나 배치 ingest + 집계 트레이드오프
Presto	쿼리 레이어 역할(스토리지 분리 필요), 단독 OLAP 백엔드 부적합
Apache Drill	스키마리스 장점이 있으나 성능·운영 성숙도 미흡
BigQuery	완관리형 SaaS 로 셀프호스팅 불가, 지연·비용 통제 한계
Cloud Spanner	OLTP 중심, 대규모 집계 분석에 부적합
Spark Streaming	마이크로배치 기반, 진정한 실시간 쿼리 응답 제공 어려움

Sentry Snuba OLAP 후보 검토 — 탈락 사유

실시간 다차원 이벤트 집계 요건을 기준으로 평가한 9개 OLAP 시스템 검토 결과

OLAP 후보 시스템	결과	탈락 사유 / 선택 근거
Apache Druid	탈락	운영 복잡도 과다 — ZooKeeper·Coordinator·Broker 분리 구조
Apache Pinot	탈락	Druid와 유사한 다층 구조 — Controller·Server·Broker 분리, 운영 부담
Spark SQL	탈락	스트리밍 집계 레이턴시 부적합 — 배치 처리 중심, 서브초 쿼리 불가
Presto / Trino	탈락	저장 계층 없음 — 쿼리 엔진만 제공, 레이턴시 높아 실시간 대시보드 부적합
Google BigQuery	탈락	클라우드 전용, 온프레미스 배포 불가 — 벤더 종속·비용 예측 불가
Amazon Redshift	탈락	클라우드 전용, 실시간 스트림 수집 지원 부족 — 대용량 배치 DW 특화
Elasticsearch	탈락	집계 성능·압축률 열세 — 역색인 구조로 OLAP 집계에 비효율
Apache Cassandra	탈락	OLAP 쿼리 비적합 — OLTP 특화 와이드 칼럼 스토어, 임의 집계 불가
ClickHouse	채택	단일 프로세스 · 고압축 · 서브초 집계 — 모든 요건 충족, Snuba 핵심 스토리지로 채택

출처: Sentry Engineering Blog — Snuba: Sentry's New Search and Analytics Infrastructure (2019)

캡션: Sentry Snuba 후보 검토 — 8개 OLAP 시스템과 탈락 사유

ClickHouse 선택 사유 4가지와 모니터링 워크로드 평가 기준으로의 일반화

8개 후보를 실제 프로토타이핑한 끝에 Sentry 가 ClickHouse 를 선택한 이유는 4가지로 정리됩니다 [S11]. 이 4가지는 Sentry 만의 특수 요건이 아니라 모니터링·관측가능성 워크로드 전반에 공통으로 적용 가능한 평가 기준입니다.

첫 번째는 실시간 성능입니다. Snuba 는 사용자가 Sentry 대시보드에서 이슈를 클릭할 때마다 즉시 집계 쿼리 결과를 반환해야 합니다. 수십 억 건의 트레이스·이벤트 중에서 특정 릴리스, 특정 태그 조합에 해당하는 오류만 초 단위로 집계할 수 있어야 합니다. ClickHouse 는 이 요건을 충족한 반면 배치 처리 중심 후보들은 지연이 허용 기준을 초과했습니다.

두 번째는 분산·복제입니다. Sentry 는 단일 노드 장애가 서비스 전체 중단으로 이어지지 않도록 멀티 마스터 비동기 복제를 요구했습니다. ClickHouse 의 ReplicatedMergeTree 는 ZooKeeper 메타데이터 계층을 통해 노드 장애 시 자동 페일오버를 제공합니다 [S03]. 이 구조적 특성이 운영 안정성 요건을 충족했습니다.

세 번째는 스토리지 엔진 유연성입니다. Sentry 의 이벤트 데이터는 단순 로그가 아니라 검색·이슈 상세·성능·Rule 처리 등 다양한 쿼리 패턴을 지원해야 합니다. ClickHouse 의 MergeTree 엔진

계열(ReplacingMergeTree · AggregatingMergeTree 등)은 각 워크로드에 맞는 스토리지 동작을 선택할 수 있어 단일 스토리지 엔진으로 복수의 쿼리 패턴을 커버합니다.

네 번째는 일관성 유연성입니다. 모니터링 데이터의 특성상 강력한 트랜잭션 일관성(ACID)보다 최종 일관성(eventual consistency)으로 충분합니다. 5분 전 에러율이 정확히 일치하는 것보다 지금 대시보드가 즉시 로드되는 것이 운영에서 더 중요합니다. ClickHouse 는 강력한 일관성을 포기하고 성능과 가용성을 극대화하는 방향으로 설계되어 모니터링 워크로드의 이 트레이드오프와 정확히 일치합니다 [S11].

이 4가지 기준은 국내 IT 조직이 모니터링·관측가능성 플랫폼의 OLAP 백엔드를 평가할 때 그대로 재사용할 수 있는 판단 틀입니다. "우리가 검토 중인 후보가 실시간 성능, 분산·복제, 스토리지 엔진 유연성, 일관성 유연성 4개 기준을 동시에 충족하는가"라는 질문이 후보군을 좁히는 가장 효율적인 방법입니다. Sentry 는 8개 후보를 프로토타이핑하며 이 질문에 답했고, 결과는 ClickHouse 였습니다 [S11].

3장: Apache 2.0 라이선스의 실무적 의미와 상업 솔루션의 의무

ClickHouse 를 상업 솔루션에 통합하거나 SaaS 형태로 제공하려는 조직이 가장 먼저 확인해야 할 것은 라이선스 구조입니다. ClickHouse 는 Apache License 2.0(아파치 라이선스 2.0)으로 배포됩니다. 이 라이선스는 소스 코드 공개 의무를 부과하지 않고, 추가 라이선스 비용 없이 상업적 재배포와 SaaS 운영을 허용합니다. 그러나 "자유롭다"는 표현이 "아무 제약이 없다"를 의미하지는 않습니다. Apache 2.0 은 3가지 구체적인 의무를 명시하며, 이를 이행하지 않으면 라이선스 위반이 됩니다[S18]. 본 장은 이 권리와 의무의 실무적 의미를 원문 기준으로 풀이하고, 국내 상업 솔루션에서 발생하는 임베드 시나리오 3종의 의무 범위를 정리합니다. 아울러 Elastic 의 SSPL(Server Side Public License, 서버 사이드 퍼블릭 라이선스) 전환 사례와 비교하여 "ClickHouse 도 같은 전환을 할 수 있는가"라는 실무적 우려에 사실로 답합니다. 본 장은 법률 의견서가 아닙니다. 실제 도입 시 사내 법무 검토가 별도로 필요합니다.

3.1 Apache License 2.0 의 4 가지 권리와 3 가지 의무

Apache 2.0 이라는 이름은 국내 IT 조직에서도 낯설지 않지만, 정확한 권리·의무 범위를 파악하고 있는 의사결정권자는 많지 않습니다. "오픈소스이면 코드를 공개해야 한다", "상업 이용 시 라이선스 비용이 발생한다"는 잘못된 통념이 RFP 단계에서 ClickHouse 도입을 주저하게 만드는 원인이 되기도 합니다. 이 절은 Apache License 2.0 원문[S18]을 기준으로 4가지 권리와 3가지 의무를 정확히 정리합니다.

권리 구조와 다른 라이선스 대비 위치

Apache License 2.0 이 부여하는 첫 번째 권리는 소프트웨어를 어떤 목적으로든 사용할 수 있는 권리입니다. 상업 목적·연구 목적·내부 운용 목적 모두에서 라이선서의 별도 허가 없이 즉시 사용할 수 있습니다. 두 번째 권리는 수정권으로, 소스 코드를 변경하여 자체 요구에 맞게 개선할 수 있습니다. 세 번째 권리는 재배포권입니다. 수정 여부와 관계없이 바이너리·소스 형태로 재배포할 수 있으며, 이때 GPL(GNU General Public License, GNU 일반 공중 라이선스)과 달리 파생 저작물의 소스 코드를 공개할 의무가 발생하지 않습니다. 네 번째 권리는 특허 이용권입니다. 기

여자가 해당 소프트웨어와 관련하여 보유한 특허를 라이선스 이용자에게 무상으로 허여합니다. 이 특허 이용권 조항은 MIT나 BSD 2-Clause 라이선스에는 없는 Apache 2.0 고유의 보호 장치로, 상업 환경에서 특허 분쟁 노출을 줄이는 실질적인 역할을 합니다[S18].

라이선스 유형 간의 차이는 특히 카피레프트(copyleft) 여부에서 두드러집니다. GPL은 강한 카피레프트 조건을 적용하여 GPL 소프트웨어를 포함한 파생 저작물을 배포할 때 전체 소스 코드를 동일 라이선스로 공개해야 합니다. AGPL(Affero GPL, 아페로 GPL)은 SaaS 형태로 제공할 때도 이 의무를 확장하여 적용합니다. SSPL 은 데이터베이스 서비스를 제공할 때 서비스 전체 스택의 소스 코드 공개를 요구하는 더 강한 조건을 부과합니다. 이에 비해 Apache 2.0 은 비(非)카피레프트 퍼미시브(permissive) 라이선스로 분류되며, 소스 공개 없이 상업 배포와 SaaS 운영이 가능합니다[S20].

아래 표는 주요 오픈소스 라이선스를 재배포 자유도·소스 공개 의무·특허 이용권·SaaS 적용 범위 기준으로 비교합니다.

라이선스	재배포 자유도	파생 저작물 소스 공개 의무	특허 이용권 명시	SaaS 운영 시 소스 공개
Apache 2.0	상업 포함 자유	없음	있음	없음
MIT	상업 포함 자유	없음	없음	없음
BSD 2-Clause	상업 포함 자유	없음	없음	없음
MPL 2.0	상업 포함 자유	수정 파일에 한함	있음	없음
GPL v2/v3	상업 포함 자유	파생 저작물 전체	없음 (v3 있음)	없음
AGPL v3	상업 포함 자유	파생 저작물 전체	있음	있음
SSPL	상업 포함 자유	서비스 스택 전체	없음	있음

ClickHouse 의 Apache 2.0 채택은 이 표에서 가장 넓은 상업 활용 범위를 제공하는 라이선스 그룹에 속함을 의미합니다. Altinity 가 발표한 분석[S20]에서도 "ClickHouse 의 Apache 2.0 라이선스는 상업 임베드, SaaS 백엔드, 가치 부가 분석 서비스 모두를 허용하며 별도 라이선스 비용을 요구하지 않는다"고 명시하고 있습니다.

의무 구조와 실무 준수 방법

권리가 넓다는 사실은 의무가 없다는 의미가 아닙니다. Apache License 2.0 원문[S18]은 3가지 의무를 명시합니다. 첫 번째는 LICENSE 파일 보존 의무입니다. 소프트웨어를 배포할 때 원본 LICENSE 파일을 배포 산출물에 포함해야 합니다. 바이너리만 배포하는 경우에도 이 파일은 제거하지 말아야 합니다. 두 번째는 NOTICE 파일 보존 의무입니다. 원본 소프트웨어가 NOTICE 파일을 포함하고 있는 경우, 배포 시 해당 파일의 내용을 유지해야 합니다. NOTICE 파일에는 저작권 고지나 외부 의존성 정보가 포함될 수 있습니다. 세 번째는 변경 고지 의무입니다. 소스 코드를 수정하여 배포할 때에는 수정했음을 명확히 표시해야 합니다. 단, 수정된 소스 코드 자체를 공개할 의무는 없으며, 수정했다는 사실만 표시하면 됩니다[S18][S19].

이 3가지 의무는 구현 난이도가 낮습니다. 배포 패키지에 LICENSE 파일과 NOTICE 파일을 동봉하고, 소스를 수정한 경우 해당 파일 헤더에 수정 내역을 1줄로 표기하는 것으로 충족됩니다. SBOM(Software Bill of Materials, 소프트웨어 구성 성분서)을 활용하는 조직이라면 Apache 2.0 항목에 "LICENSE·NOTICE 파일 동봉 확인" 체크 항목을 추가하면 컴플라이언스 절차를 정형화할 수 있습니다.

아래 표는 3가지 의무의 실무 체크리스트입니다.

의무 항목	해당 조건	이행 방법	미이행 결과
LICENSE 파일 보존	배포 산출물에 ClickHouse 포함 시	배포 패키지 루트에 LICENSE 파일 동봉	라이선스 위반 — 재배포 권리 소멸 가능
NOTICE 파일 보존	원본 NOTICE 파일이 존재하는 경우	NOTICE 파일 내용 유지 또는 통합 문서에 병합	라이선스 위반
변경 고지	소스 코드를 수정하여 배포하는 경우	수정 파일 헤더 또는 CHANGES 문서에 수정 사실 표기	라이선스 위반

3.2 상업 솔루션 임베드 시나리오별 의무 점검

Apache 2.0 의 권리·의무 원칙을 이해했더라도, 자사가 어떤 임베드 형태를 취하느냐에 따라 의무의 발생 여부와 범위가 달라집니다. 이 절은 국내 IT 기업이 실제로 마주치는 3가지 시나리오를 구분하고, 각 시나리오에서 어떤 의무가 발생하고 어떤 의무는 발생하지 않는지를 정리합니다.

시나리오 A: 온프레미스 패키지에 ClickHouse 동봉

시나리오 A는 자사가 개발한 소프트웨어 제품의 설치 패키지 안에 ClickHouse 바이너리를 포함하여 고객 서버에 납품하는 형태입니다. 모니터링 솔루션, APM(Application Performance Management, 애플리케이션 성능 관리) 도구, 로그 분석 플랫폼 등 온프레미스 설치형 제품을 납품하는 국내 SI 기업과 솔루션 기업 대부분이 이 시나리오에 해당합니다. 이 경우 발생하는 의무는 명확합니다. 배포 패키지 안에 ClickHouse 의 LICENSE 파일과 NOTICE 파일을 그대로 포함하면 됩니다. 소스 코드를 수정하지 않았다면 변경 고지 의무도 발생하지 않습니다. 자사 소프트웨어의 소스 코드를 공개할 의무는 없습니다[S18][S20].

시나리오 B: SaaS 백엔드로 ClickHouse 운영

시나리오 B는 자사가 운영하는 SaaS 서비스의 내부 데이터 처리 인프라로 ClickHouse 를 사용하는 형태입니다. 고객은 웹 인터페이스나 API 로만 서비스를 이용하며, ClickHouse 자체를 직접 접근하거나 배포받지 않습니다. 이 시나리오에서 중요한 사실은, Apache 2.0 은 소프트웨어를 "배포"할 때 의무가 발생하며 내부 운영은 배포에 해당하지 않는다는 점입니다. 따라서 시나리오 B 에서는 LICENSE 파일 동봉 의무조차 발생하지 않습니다. AGPL 과 SSPL 은 이 내부 SaaS 운영 시나리오에서도 소스 공개를 강제하지만, Apache 2.0 은 이러한 확장 의무를 부과하지 않

습니다[S18][S20]. "SaaS 백엔드는 배포가 아니므로 LICENSE 동봉 의무가 없다"는 점을 사내 법무에 확인받아 두면 향후 감사 대응에도 유용합니다.

아래 표는 시나리오 A와 B 의 의무 점검표입니다.

항목	시나리오 A: 온프레미스 패키지	시나리오 B: SaaS 백엔드
ClickHouse 배포 여부	패키지에 동봉 (배포 O)	내부 운영만 (배포 X)
LICENSE 파일 동봉	필요	불필요
NOTICE 파일 동봉	필요	불필요
변경 고지	수정 시에만 필요	수정 시에도 불필요
소스 코드 공개	불필요	불필요
라이선스 비용	없음	없음

시나리오 C: Managed ClickHouse 서비스 재판매

시나리오 C는 ClickHouse 를 관리형 데이터베이스 서비스 형태로 고객에게 직접 제공하는 사업 모델입니다. 고객이 ClickHouse 클러스터에 직접 연결하여 사용하는 구조로, ClickHouse 자체가 서비스의 핵심 산출물이 됩니다. 이 시나리오에서는 Apache 2.0 의 의무가 명시적으로 발생합니다. 고객이 접근하는 서비스 인터페이스 어딘가에 LICENSE 정보를 제공해야 하며, NOTICE 파일 내용도 참조 가능하도록 유지해야 합니다. 그러나 이것이 전부입니다. 소스 코드 공개나 서비스 스택 공개는 요구되지 않습니다[S18][S20].

Apache 2.0 의 중요한 특성 중 하나는 경쟁을 금지하지 않는다는 점입니다. ClickHouse Inc. 가 ClickHouse Cloud 를 운영하고 있음에도, AWS(Amazon Web Services), GCP(Google Cloud Platform, 구글 클라우드 플랫폼), Altinity, Tinybird, DoubleCloud 같은 사업자들이 ClickHouse 기반 Managed 서비스를 별도로 운영하고 있습니다. 이것이 가능한 이유는 Apache 2.0 이 이러한 경쟁적 상업 운영을 명시적으로 허용하기 때문입니다[S20]. 국내 클라우드 사업자가 동일한 모델을 검토한다면, 기술적·법적 장벽은 Apache 2.0 라이선스 차원에서는 존재하지 않습니다.

아래 표는 실제 운영 중인 Managed ClickHouse 서비스 사례와 Apache 2.0 허용 근거입니다.

사업자	서비스 형태	Apache 2.0 허용 근거
ClickHouse Inc. (ClickHouse Cloud)	완전 관리형 클라우드	자사 제품
AWS (파트너 솔루션)	Marketplace 파트너 통합	Apache 2.0 재배포 허용
Altinity (Altinity.Cloud)	완전 관리형 서비스	Apache 2.0 명시 준수[S20]
Tinybird	실시간 분석 API 특화 서비스	Apache 2.0 기반
DoubleCloud	관리형 분석 스택	Apache 2.0 기반

캡션: 상업 솔루션 임베드 시나리오 3종 (온프레미스 동봉 / SaaS 백엔드 / Managed 재판매) 의 Apache 2.0 의무 발생 여부 매트릭스

3.3 ClickHouse Inc. 분사 이후 라이선스 변동 위험 평가

오픈소스 소프트웨어를 도입할 때 의사결정권자가 가장 먼저 제기하는 위험 질문 중 하나는 "나중에 라이선스가 바뀌면 어떻게 되는가"입니다. 이 우려는 Elastic 의 사례 이후 더욱 현실적인 질문이 되었습니다. Elasticsearch 를 Apache 2.0 에서 SSPL 로 전환한 Elastic 의 2021년 결정은 오픈소스 커뮤니티 전반에 라이선스 전환 리스크에 대한 경각심을 높였습니다. ClickHouse 도 상업 회사가 운영하는 오픈소스 프로젝트라는 구조가 같으므로, "같은 길을 갈 수 있는가"라는 질문은 합리적입니다.

2026년 6월 기준 라이선스 현황

2026년 6월 기준으로, ClickHouse 의 공식 GitHub 저장소(github.com/ClickHouse/ClickHouse)의 LICENSE 파일은 Apache License 2.0 을 그대로 유지하고 있습니다[S19]. 이 사실은 도입 시점의 라이선스를 확정해야 하는 RFP 평가 절차에서 직접 인용 가능한 1차 근거입니다. Altinity 는 ClickHouse Inc. 의 공개 입장도 포함하여 "듀얼 라이선스 전환 위험이 부재하다"는 분석을 발표했습니다[S20]. 본 백서가 작성된 2026년 6월 21일 기준, ClickHouse 의 라이선스는 Apache 2.0 입니다. 향후 라이선스 변경 여부는 본 백서의 범위를 벗어나는 사항이며, 도입 조직은 채택 시점의 LICENSE 파일을 직접 확인하는 절차를 권장합니다.

Elastic SSPL 전환 배경과 ClickHouse 상황의 차이

Elastic 이 2021년 Apache 2.0 에서 SSPL 로 전환한 배경은 AWS 와의 갈등이었습니다. AWS 가 Elasticsearch 기술을 기반으로 Amazon Elasticsearch Service(현 Amazon OpenSearch Service)를 운영하며 Elastic 의 클라우드 사업과 직접 경쟁하면서도 Elastic 에 아무런 기여나 수익을 제공하지 않는 상황이 지속되었습니다. Elastic 은 이 상황을 해소하기 위해 SSPL 로의 전환을 선택했으며, 이에 반발한 AWS 와 오픈소스 커뮤니티는 OpenSearch 로 분기(fork)하여 Apache 2.0 라이선스 아래 독립 프로젝트를 운영하고 있습니다.

ClickHouse 의 현재 상황은 Elastic 이 전환 결정을 내리던 당시와 구조적으로 다릅니다. 첫째, ClickHouse Inc. 는 이미 ClickHouse Cloud 라는 자사 Managed 서비스를 성공적으로 운영하고 있으며, Altinity·Tinybird·DoubleCloud 같은 경쟁 Managed 서비스의 존재를 라이선스 위반 없이 수용하고 있습니다[S20]. Elastic 이 AWS 의 무임승차를 직접적인 라이선스 전환 이유로 제시했다면, ClickHouse Inc. 는 경쟁 Managed 서비스의 존재와 공존하는 비즈니스 모델을 유지하고 있습니다. 둘째, ClickHouse Inc. 는 2024년 4억 달러 투자와 150억 달러 평가를 통해[S05] 클라우드 서비스 수익 기반을 충분히 확보한 상태이며, 오픈소스 커뮤니티와의 관계를 훼손할 유인이 상대적으로 낮습니다. 셋째, ClickHouse 의 분사 구조상 2009년부터 시작된 개발 역사와 2,000명 이상의 기여자 네트워크[S01]는 단일 회사의 결정으로 라이선스를 전환하는 데 높은 사회적 비용을 부과합니다.

라이선스 전환 위험을 평가할 때 균형 잡힌 시각이 중요합니다. "0%"라는 확정적 단언은 미래 사업 환경 변화를 배제하는 것이므로 적절하지 않습니다. 그러나 만약 라이선스가 전환되더라도, 전환 이전 버전의 Apache 2.0 소스 코드는 영구적으로 해당 라이선스 아래 사용 가능합니다. 오픈소스 라이선스 조건은 이미 배포된 버전에 소급 적용되지 않기 때문입니다. OpenSearch 가 Elasticsearch 의 Apache 2.0 분기 버전을 독립적으로 유지하고 있는 것처럼, ClickHouse 에서도 동일한 방어책이 작동합니다. 의사결정권자는 이 가능성을 라이선스 위험 목록에 등재하되, 현재 채택 판단에서의 가중치는 낮게 설정하는 것이 합리적입니다[S20].

아래 표는 Elastic 과 ClickHouse 의 오픈소스 거버넌스 환경을 비교합니다.

비교 항목	Elastic (2021년 전환 당시)	ClickHouse (2026년 6월 현재)
클라우드 수익 경쟁 구조	AWS 와 직접 경쟁, 수익 귀속 없음	경쟁 Managed 서비스 공존 허용
자사 클라우드 사업 성숙도	Elastic Cloud 초기 성장 단계	ClickHouse Cloud + 4억 달러 투자 기반[S05]
커뮤니티 기여자 규모	수백 명 수준	2,000명 이상[S01]
현재 라이선스	SSPL (2021년 전환)	Apache 2.0 유지[S19]
분기 프로젝트 존재	OpenSearch (AWS 주도)	부재 (전환 촉발 요인 미발생)

4장: 기술적 특징의 모니터링 워크로드 매핑

ClickHouse 의 기능 목록은 공식 문서에서 얼마든지 찾을 수 있습니다. 그러나 실제 운영 현장에서 의사결정권자에게 필요한 정보는 "이 기능이 우리 모니터링 워크로드의 어떤 통점을 해결하

는가"라는 질문에 대한 직접적인 답입니다. 모니터링 인프라는 고카디널리티 라벨(값의 종류가 수천만 개 이상인 식별자), 시계열 집계, 분산 트레이스(trace) 조인이라는 세 가지 구조적 난제를 동시에 안고 있습니다. 이 장에서는 컬럼 지향 저장·압축·벡터화, MergeTree 엔진 계열, 그리고 분산·복제·계층 스토리지라는 세 축이 각각 그 난제를 어떻게 정면으로 해결하는지 1대1 매핑으로 정리합니다. 기능 소개가 목적이 아니라 도입 검토 시 PoC(Proof of Concept, 개념 검증) 시나리오를 설계할 근거를 제공하는 것이 이 장의 목적입니다.

4.1 컬럼 지향 저장·압축·벡터화 — I/O 와 CPU 효율의 출발점

ClickHouse 의 모든 정량 우위는 세 가지 기본 설계 선택에서 출발합니다. 컬럼 지향 저장, ZSTD·LZ4 압축, 그리고 SIMD(Single Instruction Multiple Data, 단일 명령 다중 데이터 처리) 기반 벡터화 실행이 그것입니다. 이 세 가지는 서로 독립된 기능이 아니라 같은 방향으로 작용하는 하나의 연쇄 구조입니다. 컬럼 지향 저장이 I/O 범위를 줄이고, 압축이 저장 공간을 줄이며, 벡터화가 남은 CPU 작업을 더 빠르게 처리합니다. 모니터링 워크로드에서 이 연쇄 구조가 가장 극적인 효과를 내는 이유는 로그·메트릭·트레이스 데이터가 극도로 균일한 스키마와 반복적인 값 패턴을 가지고 있기 때문입니다.

컬럼 지향 저장과 ZSTD·LZ4 압축 — 로그 압축률 10:1~20:1

행 지향 데이터베이스는 한 행의 모든 컬럼을 디스크에 연속으로 기록합니다. SELECT 쿼리가 100개 컬럼 중 3개만 사용하더라도 100개 컬럼 전체를 디스크에서 읽어야 합니다. 컬럼 지향 저장은 같은 컬럼의 값을 연속으로 배치합니다. 쿼리에 필요한 컬럼만 읽으면 되므로 디스크 I/O 자체가 줄어듭니다 [S01]. 모니터링 로그 데이터는 이 차이를 한층 더 증폭시킵니다. `level`, `service`, `timestamp` 같은 컬럼은 값의 종류가 제한적이고 반복이 많아 압축 알고리즘이 패턴을 쉽게 인식합니다.

ClickHouse 는 컬럼별로 ZSTD(Zstandard, 페이스북이 개발한 고성능 압축 알고리즘) 또는 LZ4(초고속 압축 알고리즘)를 적용합니다. 동일 유형의 값이 연속으로 놓이므로 압축률이 극대화됩니다. 모니터링 로그 데이터에서 실제로 달성되는 압축률은 10:1에서 20:1 수준입니다. 즉 100GB 분량의 로그가 ClickHouse 에 저장되면 5~10GB 로 줄어듭니다 [S21]. 반면 Elasticsearch 는 역색인(inverted index) 구조 특성상 압축률이 1.5:1 수준에 그칩니다 [S21]. 1B(10억) 행 규모에서 Elasticsearch 는 245GB 를 사용하지만 ClickHouse 는 49GB 에 그쳤으며, 50B(500억) 행 기준으로는 Elasticsearch 12TB 대 ClickHouse 2.4TB 의 차이가 확인되었습니다 [S21]. 모니터링 인프라 운영 비용의 50% 이상이 저장에서 발생하므로, 이 압축률 차이는 인프라 예산 계획에서 직접적인 변수가 됩니다.

지표	ClickHouse	Elasticsearch
압축률 (로그 데이터)	10:1~20:1	약 1.5:1
1B 행 저장	49 GB	245 GB
50B 행 저장	2.4 TB	12 TB
비용 절감 효과	60~70%	—

출처: [S21]

압축률 차이가 발생하는 구조적 이유

Elasticsearch 의 역색인 구조는 텍스트 검색에 최적화되어 있습니다. 각 단어가 어느 문서에 등장하는지 매핑하는 자료구조이므로, 반복적인 값이 많은 로그 데이터를 압축하는 데 구조적으로 불리합니다. ClickHouse 는 같은 컬럼의 연속 값이 메모리 위에 인접하게 놓이므로 CPU 캐시 효율이 높고, 압축 알고리즘이 반복 패턴을 더 많이 인식합니다. Cloudflare 는 ClickHouse 도입 후 문서당 저장 비용을 Elasticsearch 대비 600 바이트에서 60 바이트로 10배 절감했습니다 [S07]. 이 수치는 단순 압축 효율이 아니라 인덱스 오버헤드 포함 전체 저장 비용 기준이라는 점에서 실운영 의미를 갖습니다.

벡터화 실행과 SIMD CPU 활용 — ms 단위 집계 응답의 근거

컬럼 지향 저장이 디스크 I/O 를 줄인다면, 벡터화 실행(vectorized execution)은 메모리에 올라온 데이터를 CPU 가 처리하는 방식을 바꿉니다. 행 지향 데이터베이스는 한 번에 한 행씩 처리합니다. 반면 ClickHouse 는 한 컬럼의 여러 값을 하나의 벡터(vector, 연속 메모리 블록)로 묶어 SIMD 명령어로 동시에 처리합니다. SIMD 는 "한 번의 CPU 명령으로 여러 데이터를 동시에 계산"하는 방식입니다 [S03]. AVX-512 명령어 기준으로 한 번에 64바이트, 즉 정수 16개 또는 부동소수 8개를 동시에 계산합니다.

벡터화 실행이 모니터링 쿼리에서 중요한 이유는 집계 연산의 특성 때문입니다. `SELECT service, count(*), avg(duration) FROM logs WHERE timestamp BETWEEN ... GROUP BY service` 같은 쿼리는 수억 행을 스캔하면서 같은 연산을 반복합니다. 이 반복 연산을 SIMD 로 병렬화하면 동일 하드웨어에서 처리 속도가 수십 배 빨라집니다 [S03]. 같은 컬럼 지향 데이터베이스인 Vertica, Greenplum 대비 5배에서 24배, Hive, MySQL 대비 100배에서 1000배의 속도 차이는 이 벡터화 실행과 SIMD 활용에서 비롯됩니다 [S03].

CPU 캐시 정합과 분기 예측 측면에서도 벡터화 실행은 유리합니다. 같은 컬럼의 값이 연속 메모리에 놓여 있으므로 L1/L2 캐시 히트율이 높습니다. 분기 예측(branch prediction)이 단조로운 반복 패턴에서 더 정확하게 동작하므로 파이프라인 스톨(stall) 이 줄어듭니다. 이 두 효과가 합산되어 동일 CPU 사이클에서 더 많은 집계 연산을 처리할 수 있게 됩니다. 모니터링 워크로드에서 실시간 대시보드 갱신, 알람 트리거 조건 평가, 시계열 집계가 밀리초 단위로 응답하는 것은 이 구조 덕분입니다 [S01].

4.2 MergeTree 엔진 계열과 Materialized View

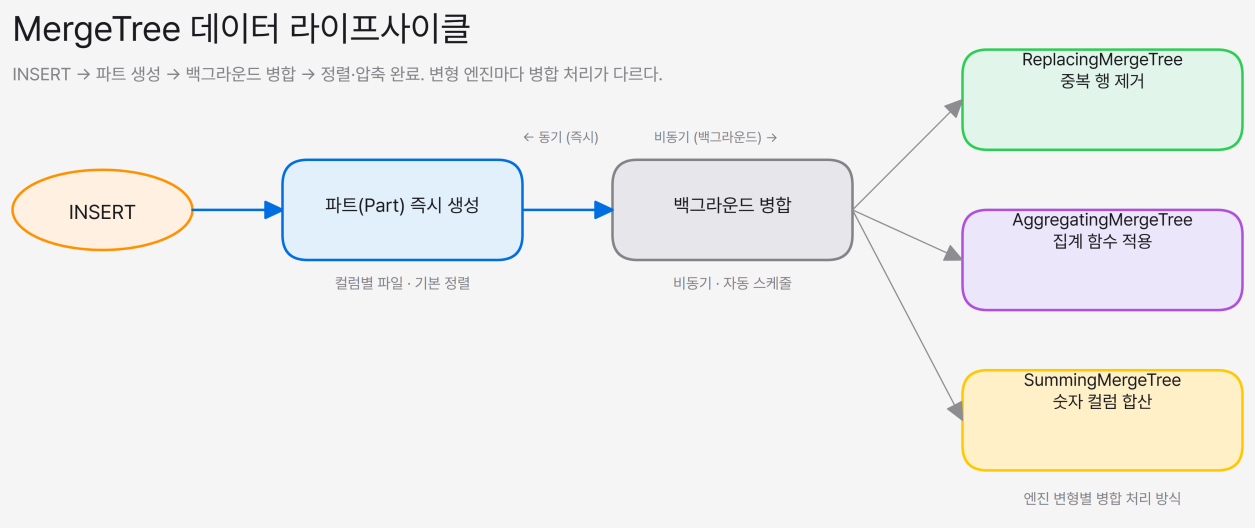
컬럼 저장과 벡터화가 쿼리 실행 시점의 효율을 높인다면, MergeTree 엔진 계열은 데이터가 저장되는 방식 자체를 모니터링 워크로드에 맞게 설계합니다. ClickHouse 에서 테이블 설계의 핵심 결정은 어떤 MergeTree 변형 엔진을 선택하느냐입니다. 변형 선택이 잘못되면 불필요한 중복 데이터가 쌓이거나 쿼리 시 집계 비용이 높아집니다. 반면 워크로드 패턴에 맞는 변형을 선택하면 백그라운드에서 데이터가 자동으로 정리되어 쿼리 응답이 빨라집니다. Materialized View 와 Projection, Sparse Primary Index 는 이 엔진 계층 위에서 고카디널리티 라벨 처리를 최적화하는 도구입니다.

MergeTree 기본 동작 — 실시간 INSERT 와 백그라운드 병합

MergeTree 는 데이터를 파트(part)라는 단위로 저장합니다. INSERT 가 들어오면 즉시 디스크에 파트를 만들고, 백그라운드 스레드가 주기적으로 파트들을 병합(merge)합니다 [S01]. 이 방식의 장점은 INSERT 가 잠금(lock)을 필요로 하지 않아 높은 ingest 처리량을 유지할 수 있다는 점입니다. 병합 과정에서 데이터는 정렬 키(sort key) 기준으로 재정렬되고 압축이 적용됩니다. 모니터링 워크로드에서 초당 수십만~수백만 건의 이벤트를 실시간으로 수신하면서도 쿼리가 밀리초 단위로 응답하는 구조가 이 비동기 병합 모델에서 가능합니다 [S03].

MergeTree 3 변형과 모니터링 워크로드 매핑

ReplacingMergeTree 는 같은 정렬 키를 가진 행 중 가장 최신 버전만 남기고 이전 버전을 제거합니다. 로그 시스템에서 동일 이벤트 ID 의 중복 전송이 발생했을 때 최종 상태만 유지하는 패턴에 적합합니다. AggregatingMergeTree 는 집계 함수의 중간 상태(aggregate state)를 저장합니다. 메트릭 데이터에서 분당 평균 응답 시간, 총 요청 수 같은 사전 집계 값을 유지할 때 사용합니다. 쿼리 시 전체 원본 데이터를 재집계하는 대신 이미 집계된 중간 상태만 읽으므로 조회 속도가 크게 빨라집니다 [S01]. SummingMergeTree 는 같은 키의 행들을 병합할 때 수치 컬럼을 합산합니다. 서비스별 일일 요청 수, 누적 오류 카운터 같은 카운터 통합 패턴에 적합합니다 [S03].



캡션: MergeTree 데이터 라이프사이클 — INSERT → 파트 생성 → 백그라운드 병합 → 정렬·압축 완료 단계

MergeTree 변형	병합 시 동작	모니터링 적합 패턴
MergeTree (기본)	정렬 + 압축만 수행	원본 로그 보존
ReplacingMergeTree	같은 키 중 최신 버전 유지	이벤트 중복 제거
AggregatingMergeTree	집계 중간 상태 병합	메트릭 사전 집계
SummingMergeTree	수치 컬럼 합산	카운터 누적 통합

출처: [S01], [S03]

Materialized View · Projection · Sparse Primary Index 의 고카디널리티 라벨 처리

고카디널리티 라벨이란 값의 종류가 수천만 개 이상인 필드를 말합니다. 분산 트레이스의 `trace_id`, 사용자 분석의 `user_id`, API 게이트웨이의 `request_id` 같은 필드가 대표적입니다. Elasticsearch 는 이 필드를 역색인으로 처리하는데, 고카디널리티 환경에서 역색인 크기가 폭발적으로 증가하여 메모리와 저장 비용이 급증합니다. ClickHouse 는 근본적으로 다른 접근 방식을 사용합니다 [S21].

ClickHouse 의 Sparse Primary Index(희소 기본 인덱스)는 전통적인 B-Tree 인덱스와 달리 모든 행에 인덱스 항목을 만들지 않습니다. 기본적으로 8,192행(granule, 그래뉼)마다 하나의 인덱스 항목을 만들어 디스크에서 읽을 그래뉼 범위를 빠르게 좁힙니다 [S03]. 고카디널리티 컬럼도 인덱스 크기가 일정하게 유지되므로 메모리 부담이 없습니다. 대신 그래뉼 내부를 선형 스캔하는 방식으로 동작합니다. 검색 패턴에 따라 이 희소 인덱스만으로 충분한 경우도 많고, 추가 최적화가 필요한 경우 Projection 또는 Materialized View 를 활용합니다.

PostHog 는 Materialized View 대신 materialized column (구체화 컬럼)을 사용해 자주 조회되는 컬럼을 미리 계산하여 저장하는 패턴을 채택했습니다 [S13]. 원본 데이터의 JSON 필드에서 특정 속성을 추출하는 연산을 쿼리 시마다 반복하는 대신, INSERT 시점에 한 번만 계산해 별도 컬럼에 저장합니다. 이 최적화로 P95 쿼리 응답 시간이 60초에서 4초로 단축되었습니다 [S14]. Projection(프로젝션)은 동일 데이터를 다른 정렬 순서로 미리 집계·저장하여, 다양한 쿼리 패턴을 단일 테이블에서 지원합니다. `trace_id` 기준 조회와 `timestamp` 기준 조회를 모두 빠르게 처리해야 하는 분산 트레이스 시스템에서 특히 유용합니다.

최적화 도구	동작 방식	적합 시나리오
Sparse Primary Index	그래뉼 단위 희소 인덱스	범위 스캔, 시계열 조회
Materialized Column	INSERT 시점 연산 미리 저장	JSON 필드 반복 추출 제거
Projection	다른 정렬 키로 데이터 복사 저장	다중 쿼리 패턴 지원
Materialized View	원본 테이블에서 집계·변환 자동 갱신	실시간 집계 유지

출처: [S13], [S03]

역색인(inverted index)을 쓰지 않는 이유도 이 맥락에서 이해할 수 있습니다. Elasticsearch 의 역색인은 텍스트 검색과 고카디널리티 필드의 정확한 일치(exact match) 조회에 최적화된 구조입니다. 그러나 모니터링 데이터의 대부분 쿼리는 시간 범위 필터 + 집계 패턴으로, 역색인이 유리한 임의 텍스트 검색보다 시계열 범위 스캔이 훨씬 많습니다. ClickHouse 의 희소 인덱스 + 컬럼 저장 + 벡터화 연쇄 구조가 이 패턴에서 일관된 우위를 만듭니다 [S21].

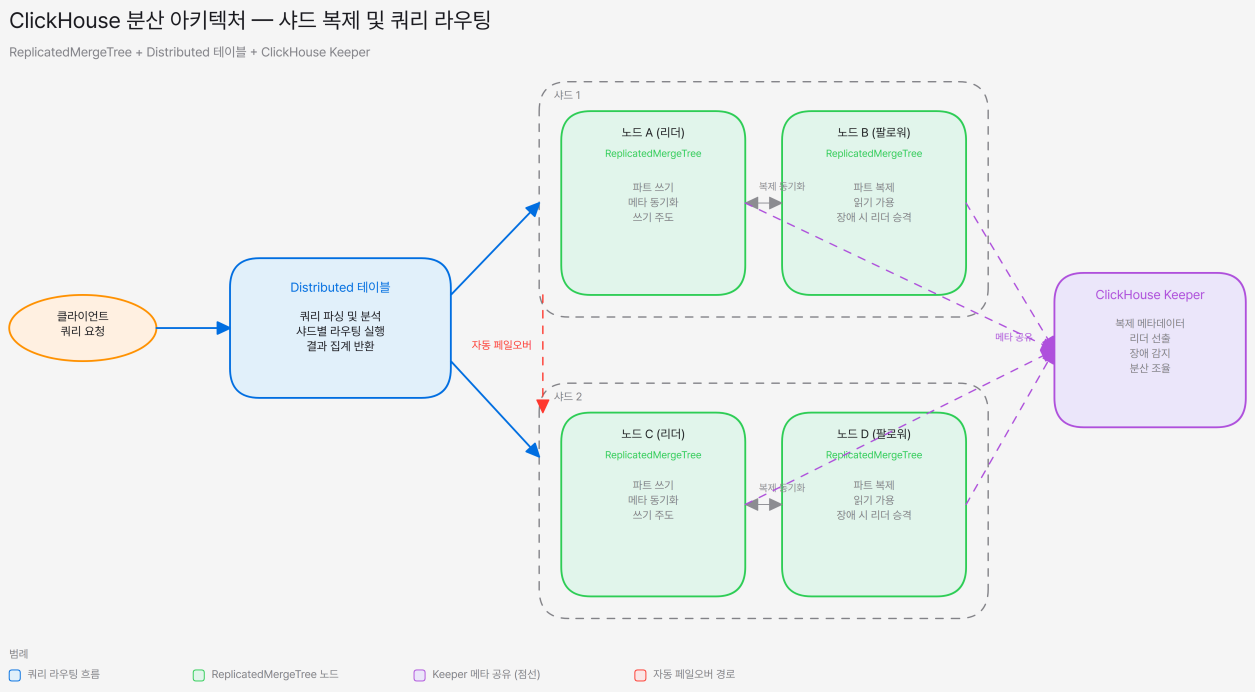
4.3 분산·복제·Kafka 엔진·계층 스토리지 — 운영 규모의 받침

단일 노드에서 작동하는 ClickHouse 의 성능 우위는 이미 앞선 절에서 확인했습니다. 그러나 수십 TB/일 이상의 모니터링 데이터를 처리하는 실제 운영 환경에서는 수평 확장, 장애 허용, 대용량 ingest 처리, 그리고 장기 보존 비용 분리라는 네 가지 요건을 동시에 충족해야 합니다. ClickHouse 는 이 요건을 ReplicatedMergeTree, Distributed 테이블, ClickHouse Keeper, 그리고 Hot/Cold 계층 스토리지의 조합으로 해결합니다. Cloudflare 는 2024년 12월 기준 이 구조 위에서 2PiB(페비바이트, 약 2,000TB) 이상의 데이터를 운영하고 있습니다 [S09].

ReplicatedMergeTree · Distributed 테이블 · ClickHouse Keeper

ReplicatedMergeTree(복제 MergeTree 엔진)는 MergeTree 엔진에 자동 데이터 복제 기능을 추가합니다. 여러 노드에 동일 데이터를 복제 유지하며, 노드 장애 시 다른 노드가 자동으로 요청을 처리합니다 [S01]. 복제 메타데이터는 ZooKeeper 또는 ClickHouse Keeper 가 관리합니다. ClickHouse Keeper 는 ZooKeeper(아파치 분산 코디네이터)의 기능을 ClickHouse 내부에 구현한 컴포넌트로, 2026년 시점에 표준 배포 방식입니다. 별도 ZooKeeper 클러스터 운영 부담 없이 ClickHouse 노드가 코디네이터 역할까지 담당하므로 운영 복잡도가 줄어듭니다 [S03].

Distributed 테이블은 여러 샤드(shard, 수평 분할 단위)에 걸친 데이터를 단일 테이블처럼 쿼리할 수 있게 하는 가상 테이블입니다. 쿼리가 들어오면 Distributed 테이블이 각 샤드에 하위 쿼리를 병렬로 분배하고 결과를 통합합니다. 클러스터 규모가 커져도 쿼리 인터페이스가 변하지 않으므로 애플리케이션 코드 수정 없이 수평 확장이 가능합니다 [S03]. 이 구조는 멀티 마스터 비동기 복제 모델로, 단일 장애점(SPOF, Single Point of Failure)이 없습니다 [S03].



캡션: ReplicatedMergeTree + Distributed 테이블 + ClickHouse Keeper 아키텍처 — 샤드별 복제와 쿼리 라우팅 흐름

비동기 INSERT · Kafka 엔진 · Hot/Cold 계층 스토리지

수백만 rows/sec ingest 를 실제 운영 환경에서 처리하려면 두 가지 요건이 충족되어야 합니다. 첫째, 높은 ingest 처리량을 유지하면서 쿼리 성능이 저하되지 않아야 합니다. 둘째, 단기 보존

데이터(핫 데이터)와 장기 보존 데이터(콜드 데이터)의 스토리지 비용이 분리되어야 합니다.

비동기 INSERT 와 Buffer 테이블은 첫 번째 요건을 해결합니다. 동기 INSERT 는 매 INSERT 마다 디스크 쓰기를 완료하므로 높은 ingest 처리량에서 병목이 됩니다. ClickHouse 의 비동기 INSERT 는 메모리에 데이터를 누적한 뒤 일정 임계값에 도달했을 때 배치로 디스크에 씁니다. Kafka 엔진은 Apache Kafka(고처리량 분산 메시지 시스템)에서 직접 데이터를 당겨오는(pull) 통합 인터페이스를 제공합니다. PostHog 는 Kafka 에서 데이터를 pull 하는 ingestion 파이프라인을 통해 Kafka 장애 시에도 데이터 손실 없이 재처리가 가능한 구조를 구축했습니다 [S13]. Cloudflare 는 이 구조 위에서 초당 수백만 rows 의 HTTP 요청 로그를 처리하며 2024년 12월 기준 2PiB+ 규모의 클러스터를 운용하고 있습니다 [S09].

Hot/Cold 계층 스토리지는 두 번째 요건을 해결합니다. 최근 데이터(핫 데이터)는 고속 NVMe SSD 에 보관하고, 오래된 데이터(콜드 데이터)는 S3 호환 객체 저장소(예: AWS S3, MinIO, Ceph)로 자동 이동합니다. ClickHouse 의 스토리지 정책(storage policy) 설정으로 이동 조건을 지정하면 백그라운드에서 자동으로 처리됩니다 [S06]. 객체 저장소 비용은 NVMe SSD 대비 1/10~1/20 수준이므로, 30일 이상 장기 보존이 필요한 컴플라이언스 또는 감사 데이터를 낮은 비용으로 유지할 수 있습니다. 모니터링 데이터는 보존 기간이 길수록 참조 빈도가 낮아지므로, 이 계층 분리 구조는 비용과 성능을 동시에 최적화하는 현실적인 방법입니다 [S06].

이 세 가지 계층(ReplicatedMergeTree 복제, Kafka 엔진 ingest, Hot/Cold 스토리지)이 결합될 때 ClickHouse 는 "초당 이벤트 100만 건 수신 + 실시간 집계 쿼리 밀리초 응답 + 2년치 데이터 낮은 비용 보존"이라는 모니터링 인프라의 3중 요건을 단일 플랫폼에서 충족할 수 있습니다. 각 기능이 독립적으로 의미를 갖기보다, 함께 작동할 때 전체 모니터링 워크로드의 실제 운영 요건을 해결하는 구조로 설계되었다는 점이 ClickHouse 를 단순한 빠른 데이터베이스 이상의 모니터링 전용 플랫폼으로 자리매김하게 합니다 [S01], [S09].

5장: 글로벌 활용 사례 — 7 개 채택사의 독립적 동일 결론

ClickHouse 채택을 단일 벤더의 마케팅 결과로 오해하는 시각이 있습니다. 그러나 실제 데이터를 보면 서로 다른 산업·규모·기술 스택을 가진 7 개 글로벌 기업이 각자 독립적인 평가 과정을 거쳐 동일한 결론에 도달했다는 사실이 분명하게 드러납니다. Cloudflare 는 초당 6백만 HTTP 요청을 처리하는 인터넷 인프라 회사이고, Uber 는 수십억 건의 이동 데이터를 매일 기록하는 모빌리티 플랫폼이며, Sentry 는 전 세계 개발팀이 사용하는 오류 추적 서비스입니다. PostHog 는 제품 분석 도구를 제공하는 스타트업이고, SigNoz 와 OpenObserve 는 오픈소스 모니터링 플랫폼이며, Langfuse 는 LLM(대형 언어 모델) 관측가능성 분야의 신흥 도구입니다. 이 7 개 조직이 서로 의논 없이 같은 결론에 이르렀다는 사실이야말로 ClickHouse 의 실용적 가치를 입증하는 가장 강력한 근거입니다.

본 장은 이들의 채택 배경·기술 선택 이유·정량 효과를 세 절로 나누어 정리합니다. 5.1 절에서는 메가 스케일 로그 분석 영역의 Cloudflare 와 Uber 사례를, 5.2 절에서는 제품 백엔드로 ClickHouse 를 채택한 Sentry 와 PostHog 의 사례를, 5.3 절에서는 OpenTelemetry(개방형 원격 측정 표준) 기반 모니터링 생태계에서 ClickHouse 를 핵심 스토리지 계층으로 채택한 SigNoz·OpenObserve·Langfuse 의 사례를 다룹니다. 각 사례는 독자 조직의 워크로드와 가장 유

사한 사례를 PoC(개념 검증) 설계의 기준선으로 삼을 수 있도록 구체적인 수치와 아키텍처 패턴을 함께 제시합니다.

5.1 Cloudflare 와 Uber — 메가 스케일 로그 분석

Cloudflare 가 직면한 규모 문제

Cloudflare 는 전 세계 인터넷 트래픽의 상당 부분을 중계하는 CDN(콘텐츠 전달 네트워크) 겸 보안 인프라 회사입니다. 2019년 Cloudflare 엔지니어링 팀은 HTTP 분석 파이프라인을 재설계해야 하는 상황에 놓였습니다. 초당 6백만 건의 HTTP 요청을 처리하면서도 각 요청의 로그를 실시간으로 분석해야 했고, 당시 운영 중이던 Elasticsearch 기반 시스템은 이 규모를 감당하는 데 구조적 한계를 드러냈습니다 [S07]. Elasticsearch 에서 문서 하나를 저장하는 데 600 바이트가 필요했던 반면, ClickHouse 로 전환한 뒤에는 동일한 문서를 60 바이트로 저장할 수 있었습니다. 10 배의 저장 효율 개선이 단순한 비용 절감을 넘어 "같은 예산으로 10 배 더 많은 데이터를 보존한다" 는 운영 정책의 전환을 의미했습니다.

Cloudflare 의 정량 성과

Cloudflare 가 ClickHouse 전환 이후 공개한 정량 성과는 단일 회사 사례로서 이례적인 규모를 보여줍니다 [S07][S08]. DNS 파이프라인에서는 초당 150만 건의 메시지를 처리하면서도 초단위 집계 응답을 유지했습니다. 더 주목할 만한 수치는 단일 쿼리 처리 성능입니다. 1 시간 분량의 이벤트 96 조 건을 단일 쿼리로 2 초 미만에 처리했고, 하루 분량인 1.61 quadrillion(1.61×10^{15}) 건의 이벤트도 단일 쿼리로 2 초 미만에 응답했습니다 [S08]. 이 수치를 실무 관점으로 환산하면, 초당 약 190억 건의 이벤트를 저장한 데이터에서 임의의 집계 쿼리가 2 초 안에 응답한다는 의미입니다. 2024년 12월 기준 Cloudflare 의 ClickHouse 클러스터는 2 페비바이트(PiB, Pebibyte) 를 초과하는 데이터를 저장하면서 초당 수백만 행의 ingest(데이터 적재) 를 지속합니다 [S09].

"Cloudflare 규모는 우리와 다르다"는 우려에 대한 답변

한국 IT 의사결정권자 가운데는 "Cloudflare 같은 글로벌 인터넷 기업의 규모는 우리 환경과 비교가 안 된다"고 판단하는 경우가 있습니다. 이 판단은 부분적으로 타당하지만, 중요한 사실을 간과합니다. Cloudflare 가 채택한 아키텍처 패턴 — MergeTree 엔진 기반 컬럼 저장, 비동기 INSERT, Hot/Cold 계층 스토리지, ClickHouse Keeper 기반 복제 — 은 규모에 관계없이 동일하게 동작합니다. 1/100 규모에서도 같은 압축 효율, 같은 쿼리 패턴, 같은 운영 방식이 적용됩니다. Cloudflare 사례의 의미는 "이 아키텍처가 가장 극단적인 규모에서도 무너지지 않는다"는 검증이며, 이는 중소 규모 도입 시에도 유효한 상한 근거를 제공합니다.

Uber 의 ELK → ClickHouse 전환

Uber 는 전 세계 수십억 건의 이동 데이터를 매일 처리하는 플랫폼으로, 자체 로깅 인프라 역시 극단적인 규모를 요구합니다. Uber 엔지니어링 팀은 ELK 스택(Elasticsearch + Logstash + Kibana의 약어) 기반 로깅 시스템을 ClickHouse 로 전환한 과정을 공개했습니다 [S10]. 전환의 핵심 요구사항은 두 가지였습니다. 하나는 로그 스키마의 유연성 — 서비스마다 다른 로그 포맷

을 하나의 스키마로 수용해야 했고, 다른 하나는 Kibana 기반 대시보드와의 호환 유지 — 운영팀이 익숙한 UI 를 바꾸지 않으면서 백엔드만 교체해야 했습니다.

Uber 의 schema-agnostic 운영 패턴과 정량 효과

Uber 가 설계한 schema-agnostic(스키마 독립적) 로그 모델은 평균 40개 이상의 필드를 단일 ClickHouse 테이블에 저장하는 방식으로 구현되었습니다 [S10]. 로그 이벤트마다 필드 수와 유형이 달라도 ClickHouse 의 JSON 컬럼 타입과 Dynamic Column 기능을 활용해 하나의 테이블로 수용했습니다. 그 결과, 두 번째 스키마 유형의 쿼리 속도가 전환 전 ELK 대비 50 배 빨라졌습니다. 클러스터 규모는 50% 이상 축소되었으며, 동시에 처리할 수 있는 쿼리 수는 오히려 늘었습니다. 비용 관점에서 보면 같은 예산으로 더 많은 데이터를 더 빠르게 분석하는 결과를 달성했습니다. Kibana 호환 레이어를 유지했기 때문에 운영팀의 재교육 비용도 최소화되었습니다.

5.2 Sentry · PostHog — 제품 백엔드로서의 ClickHouse

제품 백엔드로서의 ClickHouse

Cloudflare 와 Uber 가 자사 인프라 운영 도구로 ClickHouse 를 채택했다면, Sentry 와 PostHog 는 자사 제품 자체의 핵심 분석 백엔드로 ClickHouse 를 선택했습니다. 이 차이는 중요합니다. 인프라 도구로 사용하는 것과 고객에게 제공하는 SaaS(서비스형 소프트웨어) 제품의 핵심 쿼리 엔진으로 사용하는 것은 가용성·응답 속도·다중 쿼리 패턴 지원 요건이 훨씬 높습니다. 두 회사 모두 제품 출시 전 복수의 OLAP(온라인 분석 처리) 시스템을 비교 검토했으며, 결론적으로 ClickHouse 를 선택했습니다.

Sentry 의 Snuba 아키텍처

Sentry 는 전 세계 수백만 개발팀이 사용하는 오류 추적·성능 모니터링 플랫폼입니다. Sentry 엔지니어링 팀은 제품 내 검색·이슈 상세·성능 데이터·Rule(알림 규칙) 처리 등 제품 전반의 분석 쿼리를 단일 서비스로 추상화하기 위해 Snuba 라는 ClickHouse 추상화 레이어를 설계했습니다 [S11]. Snuba 설계 전 Sentry 팀은 Impala, Apache Druid, Apache Pinot, Presto, Apache Drill, BigQuery, Cloud Spanner, Spark Streaming 등 8 개 OLAP 시스템을 직접 프로토타이핑했습니다. 각 시스템은 특정 요건에서 강점을 보였지만, 실시간 집계 성능·분산 복제 구성의 단순함·스토리지 엔진 유연성·일관성 모델의 조정 가능성이라는 4 가지 기준을 동시에 충족하는 시스템은 ClickHouse 가 유일했습니다 [S11].

Sentry 의 Snuba 운영 현황과 62x 가속

Snuba 는 현재 Sentry 의 이슈 검색, 이벤트 상세 조회, 성능 분석, 디스커버리(임의 이벤트 탐색), 알림 Rule 평가 등 제품 전반의 쿼리를 처리합니다. 단일 쿼리 패턴만 지원하는 것이 아니라 정형 집계에서 비정형 필드 탐색까지 다양한 패턴을 하나의 ClickHouse 클러스터에서 처리한다는 점이 핵심입니다. 2025년에는 비정형(unstructured) 데이터 쿼리 처리 속도를 62 배 개선한 성과를 엔지니어링 블로그를 통해 공개했습니다 [S12]. 이 수치는 이전 ClickHouse 버전 대비 개선이며, 새로운 인덱스 전략과 쿼리 최적화 기법을 결합한 결과입니다. "ClickHouse 가 한 가지 쿼리 패턴만 잘 처리한다"는 통념과 달리, Sentry 사례는 제품 전반의 다양한 쿼리 유형을 하나의 백엔드로 수용할 수 있음을 보여줍니다.

PostHog 의 PostgreSQL 한계와 ClickHouse 선택

PostHog 는 제품 분석(product analytics) 도구를 오픈소스로 제공하는 회사입니다. 초기에는 PostgreSQL 위에서 이벤트 분석 기능을 구현했으나, Y Combinator 배치 데모에서 특정 집계 쿼리가 18 초 만에 응답하는 상황에 이르렀습니다 [S14]. P95 지연(상위 5% 느린 쿼리의 응답 시간) 은 60 초를 초과했습니다. 이 수준의 응답 속도는 사용자 대면 제품의 분석 기능으로는 수용 불가능한 수치였습니다. PostgreSQL 이 OLTP(온라인 트랜잭션 처리) 워크로드에 최적화된 시스템이라는 점에서 이 한계는 구조적이었습니다. 행 단위 저장과 B-Tree 인덱스는 트랜잭션 처리에는 적합하지만, 전체 이벤트 컬럼을 스캔·집계하는 분석 쿼리에는 근본적으로 비효율적입니다.

PostHog 의 점진 마이그레이션 패턴

PostHog 팀은 전면 교체 방식이 아닌 점진 마이그레이션 전략을 택했습니다 [S13]. PostgreSQL 과 ClickHouse 를 병렬로 운영하면서 feature flag(기능 플래그) 를 활용해 쿼리를 단계적으로 ClickHouse 로 전환했습니다. 이 방식은 한 번에 전체 시스템을 교체하는 빅뱅 마이그레이션의 위험을 피하고, 기능별로 전환 효과를 검증하면서 롤백 가능성을 유지하는 접근입니다. 마이그레이션 완료 후 동일 Y Combinator 쿼리의 응답 시간은 18 초에서 1 초로 단축되었고, P95 지연은 60 초에서 4 초로 개선되었습니다 [S14]. 추가로 materialized column(구체화 컬럼, 자주 조회하는 표현식을 미리 계산해 저장한 컬럼) 최적화를 적용해 쿼리 처리 중 불필요한 계산 반복을 제거했습니다. Kafka(메시지 브로커) 에서 데이터를 직접 pull 하는 ingestion(데이터 수집) 파이프라인을 구성하여 장애 내성도 높였습니다 [S13].

점진 마이그레이션 패턴의 재사용 가능성

PostHog 의 점진 마이그레이션 패턴은 한국 IT 기업의 위험 수용 한도에 부합하는 방식입니다. 기존 OLTP DB 를 즉시 제거하지 않고 분석 쿼리만 ClickHouse 로 이전하는 방식은 서비스 연속성을 보장하면서도 성능 효과를 검증할 수 있는 현실적인 경로입니다. 의사결정권자가 "전면 교체는 부담스럽지만 성능 개선은 필요하다"는 상황에 있다면, PostHog 사례가 그 경로의 구체적 근거가 됩니다.

5.3 OpenTelemetry-native 생태계 — SigNoz · OpenObserve · Langfuse

OTel-native 생태계와 ClickHouse

2023년 이후 모니터링 도구 시장에서 주목할 만한 흐름이 나타났습니다. OpenTelemetry(OTel, 개방형 원격 측정 표준 — 로그·트레이스·메트릭을 통합 수집하는 CNCF 프로젝트) 를 기본 수집 표준으로 채택한 신세대 모니터링 도구들이 공통적으로 ClickHouse 를 스토리지 백엔드로 선택하고 있습니다. 이 선택은 우연이 아닙니다. OTel 데이터의 특성 — 대량의 타임스탬프 기반 이벤트, 다양한 차원의 집계, 트레이스 ID 같은 고카디널리티(값의 종류가 수천만 개 이상) 필드 — 이 ClickHouse 의 설계 강점과 정확히 맞아떨어지기 때문입니다.

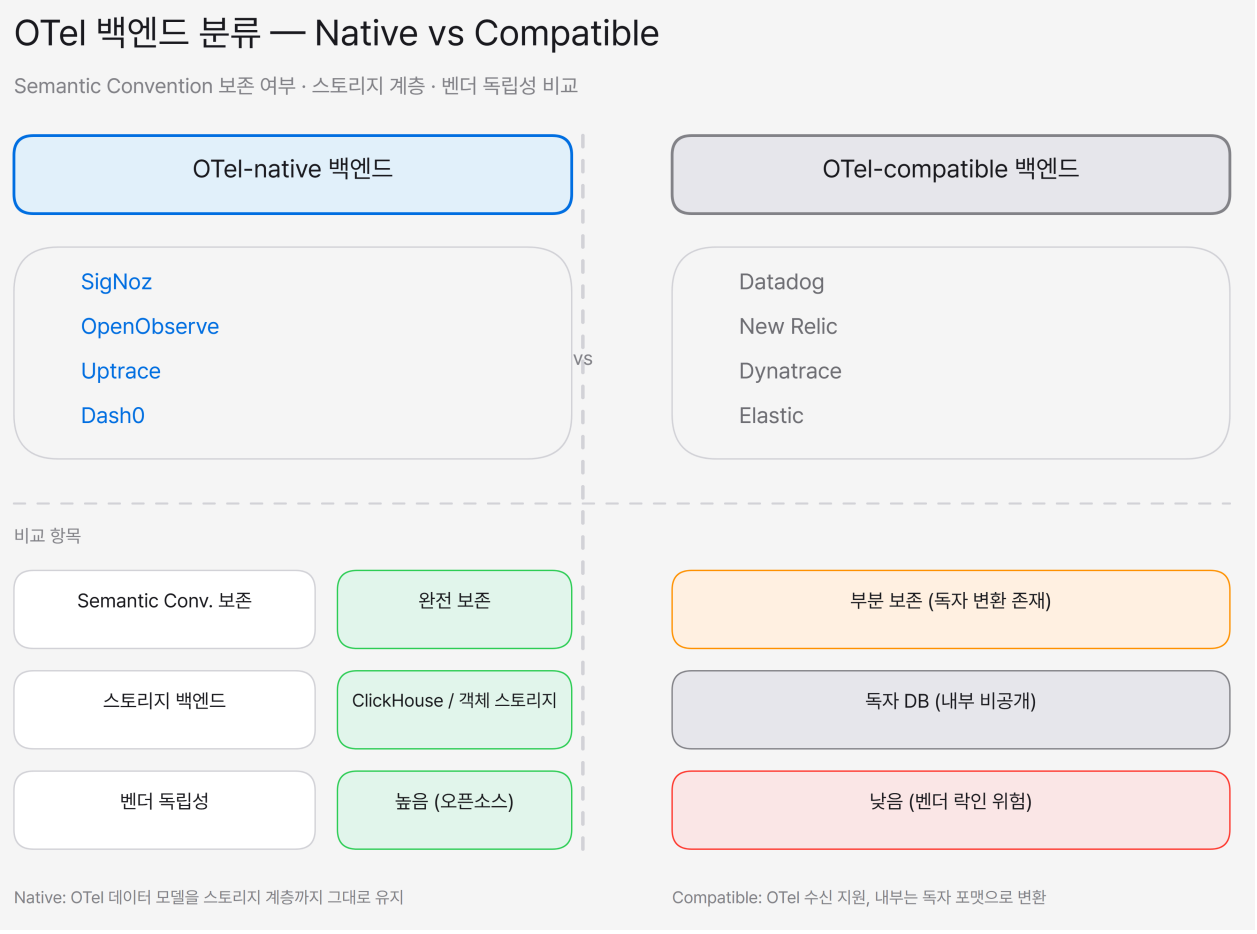
SigNoz — OTel-native APM 의 ClickHouse 백엔드

SigNoz 는 Datadog, New Relic 같은 상용 APM(애플리케이션 성능 관리) 도구의 오픈소스 대안으로 설계된 플랫폼입니다 [S15][S16]. 로그·트레이스·메트릭을 단일 UI 에서 분석할 수 있도록

구성되어 있으며, 데이터 저장과 쿼리 전체를 ClickHouse 하나로 처리합니다. 이 단일 스토어 전략은 운영 측면에서 이점을 제공합니다. 로그 스토어, 트레이스 스토어, 메트릭 스토어를 별도로 운영하는 전통 APM 스택과 달리, ClickHouse 하나에서 세 신호 유형을 통합 저장·쿼리합니다. SigNoz 는 Uber, Cloudflare 가 동일하게 사용하는 ClickHouse 를 백엔드로 채택했다는 점을 의도적으로 강조합니다. 이는 "대규모 트래픽을 처리하는 회사가 신뢰하는 스토어" 라는 검증을 간접적으로 활용하는 전략입니다 [S15].

OTel-native vs OTel-compatible 백엔드의 차이

모니터링 백엔드를 평가할 때 OTel-native 와 OTel-compatible 의 구분은 의사결정의 핵심 기준 중 하나입니다 [S17]. OTel-compatible 백엔드(Datadog, New Relic, Dynatrace, Elastic 등) 는 OTel 프로토콜로 데이터를 수집하지만 내부에서 자체 데이터 모델로 변환합니다. 이 변환 과정에서 OTel 의 semantic convention(의미 규약 — 필드 이름과 의미의 표준 정의) 이 일부 손실되거나 변형될 수 있습니다. 반면 OTel-native 백엔드(SigNoz, OpenObserve, Uptrace, Dash0 등) 는 OTel 의 데이터 모델을 그대로 보존합니다. semantic convention 보존은 벤더 독립성 측면에서 중요합니다. 데이터가 표준 형식으로 저장되어 있으면 이후 다른 도구로 마이그레이션하거나 다른 분석 도구를 연결할 때 데이터 재처리 비용이 없습니다.



캡션: OTel-native vs OTel-compatible 백엔드 분류 — 의미 규약 보존 여부와 스토리지 계층 비교

OpenObserve 의 다른 트레이드오프

OpenObserve 는 SigNoz 와 동일한 OTel-native 진영에 속하지만, 스토리지 전략이 다릅니다 [S17]. OpenObserve 는 ClickHouse 같은 DB 클러스터 대신 S3 호환 객체 스토리지를 기본 스토리지 계층으로 사용합니다. 이 설계는 DB 클러스터 운영 부담을 없애는 대신, 쿼리 응답 속도에서 일정 수준의 트레이드오프를 수반합니다. 단일 바이너리로 배포 가능하고 운영 복잡도가 낮다는 장점이 있어, 소규모 팀이나 비용 최소화가 최우선인 환경에서 선택하기 좋습니다. SigNoz(ClickHouse 클러스터)와 OpenObserve(객체 스토리지) 의 비교는 "쿼리 성능 우선이냐, 운영 단순함 우선이냐"라는 조직 내부의 우선순위에 따라 달라집니다. 두 선택지 모두 OTel-native 표준을 준수한다는 점에서, 나중에 도구를 교체하더라도 데이터 이식성은 유지됩니다.

Langfuse 의 PostgreSQL → ClickHouse 마이그레이션

Langfuse 는 LLM(Large Language Model, 대형 언어 모델) observability(관측가능성) 도구입니다. LLM 애플리케이션에서 발생하는 프롬프트·응답·토큰 사용량·지연·비용·평가 점수 같은 데이터를 수집·분석하는 용도로 사용됩니다. Langfuse 는 초기에 PostgreSQL 기반으로 구현되었으나, v3 버전을 개발하는 과정에서 ClickHouse 로 스토리지 백엔드를 전환했습니다 [S05]. 전환 이유는 PostHog 사례와 구조적으로 유사합니다. LLM 관측 데이터 역시 대량의 타임스탬프 기반 이벤트이며, 사용자가 기간별·모델별·비용별·평가 점수별로 임의의 집계 쿼리를 수행합니다. 이 패턴은 PostgreSQL OLTP 가 구조적으로 비효율적인 워크로드입니다.

Langfuse 인수와 시장 신호

2026년 ClickHouse Inc. 가 Langfuse 를 인수했습니다 [S05]. 이 인수는 단순한 포트폴리오 확장이 아닙니다. ClickHouse 라는 데이터 플랫폼 회사가 LLM observability 도구를 자사 제품군에 편입했다는 것은 "AI 애플리케이션의 분석 백엔드"라는 시장 포지션을 공식화하는 의미입니다. 인수 후에도 Langfuse 는 오픈소스 라이선스·셀프호스팅·클라우드 서비스를 모두 유지한다는 약속을 유지했습니다. 엔터프라이즈 보안·규정 준수·UI/UX 개선에 ClickHouse 의 자원이 투입될 예정입니다 [S05]. 의사결정권자 관점에서 이 인수가 갖는 의미는 8 장에서 AI 시대의 전략 가치를 다루면서 더 깊이 분석합니다.

7 개 채택사가 보내는 공통 신호

Cloudflare, Uber, Sentry, PostHog, SigNoz, OpenObserve, Langfuse 의 채택 사례를 나란히 보면 공통 패턴이 보입니다. 첫째, 7 개 회사 모두 기존 시스템(Elasticsearch, PostgreSQL, Druid 등) 을 충분히 운영한 뒤 한계에 도달했을 때 ClickHouse 로 전환했습니다. 즉흥적 선택이 아니라 검증된 기존 시스템의 구조적 한계를 경험한 뒤 이루어진 의사결정입니다. 둘째, 모두 독립적으로 평가했습니다. 7 개 회사 사이에 공통의 조정자나 추천자가 없었으며, 각자의 평가 기준과 엔지니어링 팀의 검토를 통해 동일한 결론에 도달했습니다. 셋째, 정량 효과가 유사합니다. 저장 비용 절감, 쿼리 응답 속도 향상, 클러스터 규모 축소라는 세 지표가 7 개 사례 모두에서 동일 방향으로 나타납니다. 이 공통 패턴은 ClickHouse 의 효과가 특정 환경이나 워크로드에만 한정되지 않음을 시사합니다 [S04][S07][S10][S11][S15].

6장: 모니터링 솔루션 적용 시의 정량 비교

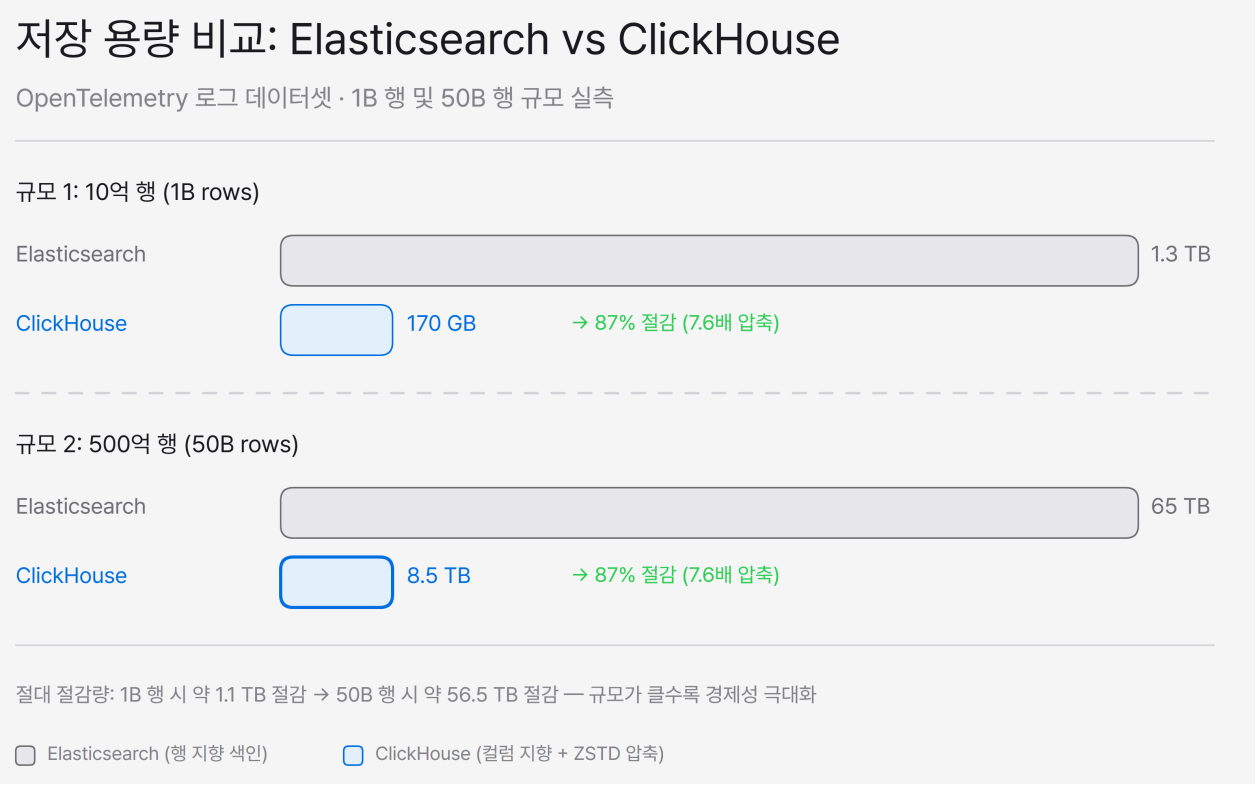
ClickHouse 를 모니터링 인프라에 도입할 때 가장 먼저 등장하는 질문은 "얼마나 좋아지는가"입니다. 이 질문에 답하려면 단일 벤더의 주장이 아니라 독립적인 복수 채택사의 실측 수치가 필요합니다. Cloudflare, Uber, Sentry, PostHog 는 각자 다른 워크로드 조건에서 ClickHouse 를 도입한 뒤 측정 결과를 외부에 공개하였으며, 세 가지 공통 지표가 반복하여 나타납니다. 첫째, 저장 공간은 Elasticsearch(엘라스틱서치) 대비 5배에서 최대 7배 이상 줄어듭니다. 둘째, 집계·그루핑 쿼리 응답 지연은 4배에서 62배까지 단축됩니다. 셋째, 동일 워크로드를 처리하는 데 필요한 클러스터 규모가 50% 이상 감소합니다. 본 장은 이 세 지표를 워크로드 가정과 측정 조건을 명시한 상태에서 정량 사실로 제시합니다.

6.1 저장 비용 — Elasticsearch 대비 7x 절감

모니터링 인프라 총비용 가운데 저장소가 차지하는 비중은 클라우드 환경에서 일반적으로 50% 내외입니다. 로그, 트레이스, 메트릭 데이터는 보존 기간이 길수록, 이벤트 발생량이 많을수록 저장 비용이 선형 이상으로 증가하므로 압축률 차이가 운영 예산 결정에 직접 영향을 줍니다. ClickHouse 의 컬럼 지향 저장과 ZSTD(Zstandard) 압축 알고리즘이 결합하면 같은 데이터셋을 Elasticsearch 보다 훨씬 작게 보관할 수 있으며, 이 차이는 데이터 규모가 커질수록 더욱 뚜렷해집니다.

1B 행·50B 행 규모별 저장 용량 실측

ClickHouse 공식 벤치마크에서 OpenTelemetry(오픈텔레메트리) 로그 데이터셋을 동일 조건으로 Elasticsearch 와 ClickHouse 에 각각 적재한 결과가 공개되어 있습니다 [S21]. 10억 행(1B row) 규모에서 Elasticsearch 는 245 GB 를 사용한 반면 ClickHouse 는 49 GB 에 그쳤습니다. 저장 공간 기준으로 5배 절감입니다. 같은 실험을 500억 행(50B row) 규모로 확장하면 Elasticsearch 가 12 TB, ClickHouse 가 2.4 TB 를 기록합니다. 이 수치 역시 5배 차이를 유지합니다. 실제 운영 환경에서는 데이터 유형에 따라 추가 압축이 가능하므로 7배 이상의 절감을 보고한 사례도 있습니다.



캡션: Elasticsearch vs ClickHouse 저장 용량 비교 (1B 행 / 50B 행 OpenTelemetry 로그 데이터셋)

이 수치를 한국 클라우드 환경에 대입하면 비용 효과가 더 구체적으로 드러납니다. AWS Seoul 리전 기준 S3 표준 스토리지 단가가 GB 당 약 25원 수준이라고 가정할 때, 50B 행 규모에서 Elasticsearch 12 TB 대비 ClickHouse 2.4 TB 는 월간 약 234 GB 차이이며 연간 저장 비용 절감액이 상당합니다. 단, 이 수치는 워크로드 특성(이벤트당 평균 필드 수, 값의 카디널리티)에 따라 달라지므로 자사 데이터로 PoC(Proof of Concept, 개념 검증)를 수행한 뒤 실측하는 절차가 필요합니다.

실제 운영 환경 적용 시 주의사항

벤치마크 수치는 단일 노드 또는 소규모 클러스터에서 측정된 경우가 많습니다. 분산 클러스터로 확장할 때는 복제 계수(replication factor)가 실제 디스크 사용량을 곱으로 늘립니다. 예를 들어 복제 계수 2인 ClickHouse 클러스터라면 49 GB 가 아니라 98 GB 가 됩니다. 그럼에도 Elasticsearch 의 기본 복제 계수 1(primary shard + 1 replica = 2배) 설정과 비교하면 ClickHouse 가 여전히 유리합니다. 보존 기간과 복제 계수를 동일하게 설정한 비교에서도 저장 비용 이점은 유지됩니다.

6.1.1 1B 행 ES 245GB → CH 49GB, 50B 행 ES 12TB → CH 2.4TB

규모별 정량 사실 요약

두 규모에서 측정된 수치를 하나의 표로 정리하면 의사결정권자가 자사 로그 보존 규모에 빠르게 대입할 수 있습니다 [S21].

규모	Elasticsearch 저장 용량	ClickHouse 저장 용량	절감 배수
10억 행 (1B row)	245 GB	49 GB	5.0x
500억 행 (50B row)	12 TB	2.4 TB	5.0x

위 수치는 OpenTelemetry 로그 스키마 기준이며, 측정 당시 Elasticsearch 8.x 와 ClickHouse 24.x 를 사용하였습니다. ClickHouse 쪽에는 ZSTD 압축을 기본 설정으로 적용하였고 Elasticsearch 는 기본 압축 설정(LZ4)을 사용하였습니다.

클라우드 단가 환산 참고

클라우드 저장 단가를 기준으로 절감액을 추산할 때는 세 가지 변수를 함께 고려해야 합니다. 첫째, EBS(Elastic Block Store) 같은 블록 스토리지를 쓰는 경우와 S3 같은 객체 스토리지를 쓰는 경우 단가 차이가 5~10배 납니다. ClickHouse 는 계층형 스토리지(tiered storage)를 통해 핫 데이터는 NVMe 블록 스토리지에, 콜드 데이터는 S3 호환 객체 스토리지에 분리 보관하는 구성이 가능하므로 장기 보존 비용이 추가로 낮아집니다 [S03]. 둘째, 복제 계수가 실제 저장 비용을 결정합니다. 셋째, 인덱스 오버헤드가 Elasticsearch 에서는 원본 데이터의 1.5배~2배까지 증가하지만 ClickHouse 에서는 sparse primary index(희소 기본 인덱스)로 오버헤드가 미미합니다. 이 세 가지를 종합하면 실제 운영 환경에서는 벤치마크 수치인 5배를 상회하는 절감을 얻는 경우가 많습니다.

의사결정 포인트

저장 비용 절감은 도입 여부를 검토하는 의사결정권자에게 가장 직관적인 지표입니다. 현재 운영 중인 Elasticsearch 클러스터의 총 디스크 사용량에 0.2를 곱하면 ClickHouse 전환 후 예상 사용량의 1차 추정치가 됩니다. 이 추정치를 클라우드 단가에 대입하면 연간 비용 절감 가능액을 빠르게 산출할 수 있으며, 이 수치는 PoC 예산 승인을 위한 1차 근거로 활용할 수 있습니다.

6.1.2 ZSTD 압축 — raw 5x~10x, 로그 데이터 10x~30x

압축 알고리즘 선택의 실무 의미

ClickHouse 는 LZ4(리엘포)와 ZSTD(지스탠다드) 두 가지 압축 알고리즘을 기본 제공합니다. LZ4 는 압축·해제 속도가 빠르지만 압축률이 낮고, ZSTD 는 속도가 상대적으로 느리지만 압축률이 훨씬 높습니다 [S21]. 핫 데이터(자주 조회되는 최신 데이터)에는 LZ4 를, 콜드 데이터(조회 빈도가 낮은 오래된 데이터)에는 ZSTD 를 적용하는 것이 일반적 운영 원칙입니다.

압축 알고리즘	압축 속도	해제 속도	압축률(raw)	로그 데이터 압축률	권장 용도
LZ4	매우 빠름	매우 빠름	2x~3x	3x~5x	핫 데이터, 실시간 조회
ZSTD (기본)	중간	빠름	5x~10x	10x~30x	웜·콜드 데이터, 장기 보존

압축 알고리즘	압축 속도	해제 속도	압축률(raw)	로그 데이터 압축률	권장 용도
Elasticsearch 기본	중간	빠름	~1.5x	~2x	—

이 표에서 핵심은 로그 데이터 압축률입니다. 로그 메시지는 반복적인 서비스 이름, IP 주소 형식, 타임스탬프 패턴 등 규칙적인 구조를 포함하므로 컬럼 지향 저장과 결합할 때 압축 효율이 극대화됩니다. 같은 컬럼 안에 비슷한 값들이 연속 배치되면 사전 압축(dictionary encoding)과 런길이 인코딩(run-length encoding)이 자동으로 적용되어 ZSTD 압축 전에 이미 상당한 중복이 제거됩니다.

운영 환경 선택 가이드

ZSTD 레벨은 1에서 22까지 조정 가능하며, 기본값은 레벨 3입니다. 레벨이 높아질수록 압축률은 올라가지만 CPU 사용량도 증가합니다. 모니터링 데이터의 경우 ZSTD 레벨 3~6 범위가 압축률과 CPU 부담 사이에서 실용적인 균형점으로 알려져 있습니다. 운영팀이 압축 알고리즘을 별도로 설정하지 않더라도 ClickHouse 기본값인 LZ4 단독보다는 ZSTD 와 병용하는 쪽이 저장 비용 절감 효과가 2배 이상 크므로, 도입 초기 단계에서 ZSTD 를 콜드 데이터 계층에 적용하도록 설정하는 것이 권장됩니다.

6.2 쿼리 지연 — 4x~62x 가속

저장 비용 절감이 운영 예산 측면의 지표라면, 쿼리 응답 지연은 운영 현장의 생산성을 직접 좌우하는 지표입니다. 모니터링 대시보드의 로딩 시간이 길면 장애 대응 속도가 느려지고, 임시 분석 쿼리가 오래 걸리면 SRE(Site Reliability Engineering, 사이트 신뢰성 엔지니어링) 팀의 업무 흐름이 끊깁니다. ClickHouse 가 Elasticsearch 대비 빠른 이유는 모든 쿼리 패턴에서 일률적인 것이 아닙니다. 단순 텍스트 매칭에서는 Elasticsearch 와 비슷한 속도를 보이지만, 집계(aggregation), 그루핑(grouping), 시간 구간 집계(time-bucketing) 패턴이 결합될 때 ClickHouse 가 결정적 우위를 가집니다. 이 패턴은 모니터링 대시보드의 전형적인 쿼리 형태입니다.

쿼리 패턴별 속도 차이 구조

ClickHouse 의 쿼리 속도 우위는 두 가지 설계 차이에서 비롯됩니다. 첫째, 컬럼 지향 저장이 집계 쿼리에 필요한 컬럼만 디스크에서 읽도록 합니다. 100개 컬럼 중 3개만 필요한 집계 쿼리라면 Elasticsearch 는 전체 문서를 읽어야 하지만 ClickHouse 는 3개 컬럼만 읽습니다. 둘째, SIMD(Single Instruction Multiple Data, 단일 명령 다중 데이터) CPU 명령어와 벡터화 실행이 한 번의 CPU 명령으로 여러 데이터를 동시에 처리합니다. 이 두 가지가 합산되면 집계 패턴에서의 속도 차이가 수배에서 수십 배까지 벌어집니다.

6.2.1 ES 대비 grouping · aggregation · time-bucketing 결합 시 ~4x

쿼리 패턴별 비교 사실

ClickHouse 공식 비교 자료와 독립 분석에 따르면 쿼리 패턴별로 속도 차이가 다르게 나타납니다 [S21] [S22]. 아래 표는 동일 데이터셋에서 주요 쿼리 패턴별 Elasticsearch 대비 ClickHouse

지연 비율을 정리한 것입니다.

쿼리 패턴	특성	ES 대비 ClickHouse 속도	설명
단순 텍스트 매칭 (LIKE)	역인덱스 유리	0.5x~1.0x	ES 와 대등하거나 느림
단일 컬럼 필터 + COUNT	컬럼 스캔	2x~4x	ClickHouse 우위 시작
GROUP BY + COUNT	집계 결합	3x~5x	컬럼 + SIMD 시너지
TIME-BUCKET + AVG + 다중 필터	모니터링 전형	~4x	가장 일반적 우위 범위
복합 집계 + 정렬	대시보드 복합	4x~8x	대형 데이터셋일수록 차이 증가

"~4x" 라는 수치가 의미하는 것은 "모든 쿼리가 4배 빠르다"가 아니라 "모니터링 대시보드의 전형적인 집계 쿼리에서 약 4배 빠르다"입니다 [S22]. 단순 텍스트 전문 검색(full-text search)은 Elasticsearch 의 역인덱스(inverted index) 구조가 강점을 발휘하는 영역이므로 ClickHouse 가 유리하지 않습니다. 이 점을 명확히 하지 않으면 도입 이후 기대치와 실측 사이에 간극이 생길 수 있습니다.

모니터링 워크로드에서의 실무적 의미

운영 현장에서 모니터링 대시보드가 Elasticsearch 기반일 때 집계 패널의 로딩 시간이 10초 이상 걸리는 경우가 흔합니다. 같은 쿼리를 ClickHouse 에서 실행하면 2~3초 수준으로 단축되며, 이 차이는 장애 발생 시 현황 파악 속도와 직결됩니다. 대규모 데이터셋(수십억 행 이상)에서는 쿼리 결과가 나오기까지 기다리지 않아도 되는 상황 자체가 달라지므로, 속도 차이가 단순 편의 이상의 운영 가치를 갖습니다.

6.2.2 Sentry 62x · PostHog 60s→4s — 실제 채택사 정량 효과

Sentry Snuba 의 62배 가속 — 워크로드 조건 명시

Sentry 의 엔지니어링 팀은 2025년에 비정형(unstructured) 로그 쿼리 최적화 작업을 공개하였습니다 [S12]. Sentry 의 Snuba 서비스는 ClickHouse 를 백엔드로 사용하며, 비정형 필드에서 특정 값을 필터링하는 쿼리가 기존 접근 방식 대비 62배 빨라졌습니다. 62배라는 수치는 특정 비정형 쿼리 패턴에서의 결과이며, 일반 집계 쿼리에 대한 평균 가속치가 아닙니다. 비교 대상은 동일 버전의 ClickHouse 에서 최적화 적용 전후 차이이므로, Elasticsearch 대비 수치와는 성격이 다릅니다.

PostHog 의 YC 쿼리 · P95 지연 개선

PostHog 는 PostgreSQL(포스트그레에스큐엘) 기반 분석 시스템에서 ClickHouse 로 전환하면서 두 가지 대표 지표를 공개하였습니다 [S14]. 첫째, Y Combinator 포트폴리오 전체를 분석하는 쿼리가 18초에서 1초로 줄었습니다. 둘째, 분석 쿼리의 P95 지연(상위 5% 느린 쿼리 기준)이 60

초에서 4초로 감소하였습니다. 이 수치의 워크로드 조건은 일별 이벤트 수십억 건 규모의 사용자 행동 분석이며 PostgreSQL 의 행 지향 저장 구조가 분석 쿼리에서 근본적 한계에 부딪힌 상황입니다 [S13].

채택사별 정량 효과 비교

아래 표는 세 채택사의 쿼리 지연 개선 수치를 워크로드 조건과 함께 정리한 것입니다.

채택사	비교 대상	개선 수치	워크로드 조건	출처
Sentry	ClickHouse (최적화 전)	62x (비정형 쿼리)	비정형 로그 필터링	[S12]
PostHog	PostgreSQL	18초→1초 (YC 쿼리)	사용자 행동 분석, 수십억 이벤트/일	[S14]
PostHog	PostgreSQL	P95 60초→4초	동일	[S14]
Cloudflare	Elasticsearch	10x 저장 절감 + ~4x 집계	HTTP 로그 6M req/sec	[S07]

이 표에서 강조할 점은 가장 극적인 수치(62x)와 가장 일반적인 수치(~4x)가 서로 다른 워크로드 조건에서 측정되었다는 것입니다. 의사결정 문서에서 "ClickHouse 를 도입하면 62배 빨라진다"고 단순 인용하면 기대치 불일치로 이어질 수 있습니다. 전형 모니터링 워크로드(시간 집계, 서비스별 에러율 집계, 대시보드 복합 쿼리)에서는 4x 수준을 기준치로 사용하고, 특정 쿼리 최적화 이후 10x 이상을 달성한 사례도 있다는 방식으로 표현하는 것이 신뢰를 유지하는 접근입니다.

수치 해석 시 주의사항

PostHog 의 60초→4초 개선은 PostgreSQL 의 행 지향 구조가 수십억 행 분석에서 본질적 한계에 도달한 상황에서의 전환 효과입니다. 이미 Elasticsearch 나 다른 컬럼 지향 DB 를 쓰는 환경에서의 전환 효과는 이보다 작을 수 있습니다. 또한 PostHog 가 공개한 수치는 ClickHouse 도입과 함께 materialized column(구체화된 컬럼) 최적화를 함께 적용한 결과이므로, 최적화 없이 단순 전환만으로 동일한 결과를 얻기는 어렵습니다.

6.3 클러스터 규모 · 인프라 비용 · 운영 복잡도

저장 비용과 쿼리 지연 외에 인프라 도입 의사결정에서 중요하게 다루어야 할 지표가 두 가지 더 있습니다. 하나는 클러스터 규모 효율, 즉 동일 워크로드를 처리하는 데 필요한 노드 수와 하드웨어 비용입니다. 다른 하나는 운영 복잡도, 즉 시스템을 유지·관리하는 데 필요한 인력과 작업량입니다. ClickHouse 는 전자에서 명확한 이점을 보이지만, 후자에서는 장단점이 공존합니다. 이 균형을 사실 기반으로 파악해야 도입 후 예상치 못한 운영 부담이 발생하지 않습니다.

클러스터 규모 효율의 실제적 의미

클러스터 규모가 줄어들면 직접 비용(서버 임대료 또는 클라우드 VM 비용) 뿐 아니라 간접 비용(운영 인력 부담, 장애 발생 가능한 노드 수, 네트워크 트래픽)도 함께 감소합니다. 동일 데이터

와 쿼리를 더 적은 노드로 처리할 수 있다면 인프라 효율이 올라가는 동시에 장애 발생 확률도 낮아집니다. Uber 와 Cloudflare 의 사례는 이 점을 정량 사실로 뒷받침합니다.

6.3.1 Uber 50%+ 축소 · Cloudflare 100% 로그 보존

Uber 의 클러스터 50% 이상 축소 사실

Uber 는 ELK(Elasticsearch-Logstash-Kibana) 스택 기반의 로깅 인프라를 ClickHouse 로 전환하면서 클러스터 규모를 50% 이상 줄이는 동시에 더 많은 쿼리를 처리하는 결과를 얻었습니다 [S10]. 이 수치가 의미하는 바는 단순한 노드 감소가 아닙니다. 같은 예산으로 처리 능력이 늘어난 것이므로, 인프라 비용 효율이 2배 이상 향상되었다고 해석할 수 있습니다. Uber 는 평균 40개 이상의 필드를 포함하는 스키마 비정형(schema-agnostic) 로그 모델을 운영하였으며, 이 복잡한 스키마에서도 ClickHouse 가 효율적으로 동작함을 확인하였습니다.

지표	ELK 스택 (전환 전)	ClickHouse (전환 후)	변화
클러스터 노드 수	기준값	기준값 대비 50%+ 감소	50%+ 축소
쿼리 처리량	기준값	기준값 이상	증가
스키마 유연성	평균 40+ 필드 지원	동일 지원	유지
Kibana 호환성	기본 제공	별도 레이어로 유지	유지

Uber 가 Kibana 호환성을 별도 레이어로 유지한 점은 주목할 만합니다. 운영팀이 기존 Kibana 기반 워크플로를 그대로 사용하면서 백엔드만 교체하는 방식이므로, 팀 내 재교육 부담을 최소화할 수 있었습니다.

Cloudflare 의 100% 로그 보존 달성

Cloudflare 는 초당 6백만 HTTP 요청을 처리하는 분석 파이프라인에서 ClickHouse 를 사용합니다 [S07]. 이전 Elasticsearch 기반에서는 비용 문제로 전체 로그 중 일부만 보존하였지만, ClickHouse 전환 후 35~45M req/sec 의 100% 로그 보존을 같은 예산 내에서 달성하였습니다. 더불어 Elasticsearch 기준 문서당 600 바이트에서 ClickHouse 기준 문서당 60 바이트로 10배 저장 효율을 확보하였습니다. 100% 로그 보존은 모니터링 인프라 운영에서 중요한 의미를 갖습니다. 샘플링된 로그는 드문 장애 패턴이나 저빈도 오류를 놓칠 수 있지만, 전체 로그를 보존하면 사후 분석의 완전성이 보장됩니다.

클러스터 규모 축소의 복합 효과

클러스터 규모 50% 축소는 서버 비용 외에도 다음 항목들에서 복합 효과를 냅니다. 네트워크 비용이 줄고, 운영 인력이 관리해야 하는 노드 수가 감소하며, 소프트웨어 업그레이드나 롤링 재시작에 걸리는 시간이 단축됩니다. 고가용성(HA) 구성에서 노드 장애 발생 시 영향 범위도 좁아집니다. 이 간접 효과들은 직접 측정하기 어렵지만 대규모 인프라 운영에서는 상당한 비용 절감 요인입니다.

6.3.2 Druid · Pinot 대비 운영 복잡도 — 컴포넌트 단순함 vs 자동화 부족

컴포넌트 구조 비교 — 단순함의 이점

ClickHouse 는 Apache Druid(드루이드)나 Apache Pinot(피노)과 달리 단일 바이너리 중심의 단순한 아키텍처를 채택합니다 [S23] [S24]. Druid 는 Master, Query, Data 계층이 각각 별도 프로세스로 구성되며 Zookeeper(주키퍼), Broker, Coordinator, Historical, MiddleManager 등 다층 컴포넌트를 운영해야 합니다. Pinot 역시 Controller, Broker, Server, Minion 의 4계층 구조를 유지합니다. ClickHouse 는 이에 비해 ClickHouse 서버 프로세스와 ClickHouse Keeper(분산 메타데이터 관리 전용) 두 가지 컴포넌트로 분산 클러스터를 구성합니다.

항목	ClickHouse	Apache Druid	Apache Pinot
필수 컴포넌트 수	2종 (서버 + Keeper)	7종 이상	4종 이상
외부 의존성	Keeper 내장 (ZooKeeper 선택)	ZooKeeper 필수	ZooKeeper 필수
자동 확장 도구	수동 설정 중심	자동 rebalance 지원	자동 rebalance 지원
운영 학습 곡선	중간	높음	높음
집계 쿼리 성능	높음	중간	중간
초저지연 user-facing OLAP	중간	중간	높음 (강점)

이 표의 수치는 Roman Leventov 의 OLAP 시스템 비교 분석과 StarTree 의 세 가지 실시간 OLAP 비교 자료를 기반으로 합니다 [S23] [S24]. 컴포넌트가 적다는 것은 장애 발생 가능한 지점이 줄어들고 운영 초기 학습에 필요한 시간이 단축된다는 의미입니다. 소규모 운영팀에서 빠르게 시작하기에 유리합니다.

자동화 도구 부족 — 균형 잡힌 평가

단순한 아키텍처에는 반대급부가 있습니다. ClickHouse 는 Pinot 이나 Druid 에 비해 분산 클러스터의 자동 리밸런싱이나 데이터 자동 재분배 도구가 성숙도가 낮습니다 [S23]. 노드를 추가하거나 제거할 때 데이터 재분산 작업을 수동으로 관리해야 하는 경우가 있으며, 샤드 설계와 레플리카 설정도 초기 구성 단계에서 신중하게 결정해야 합니다. 사후에 이 설정을 변경하는 비용이 크기 때문입니다.

이 점은 특히 대규모 분산 클러스터를 운영하는 팀에게 실질적인 부담으로 작용합니다. Pinot 의 경우 Kubernetes 환경에서의 오퍼레이터 성숙도가 높고 자동 리밸런싱 기능이 내장되어 있어 클러스터 규모를 동적으로 조정하는 상황에서 ClickHouse 보다 운영이 편리할 수 있습니다 [S24]. 9장에서 다루는 운영 학습곡선과 연결된 이 한계는 의사결정 단계에서 운영팀 규모와 역량을 함께 고려해야 하는 이유입니다.

ClickHouse Cloud 와 Managed 서비스를 통한 우회

자동화 부족 문제는 ClickHouse Cloud(공식 클라우드 서비스)나 Altinity(알티니티) 같은 Managed 서비스를 활용함으로써 상당 부분 우회할 수 있습니다. Managed 서비스는 클러스터

확장, 업그레이드, 백업, 고가용성 구성을 자동으로 처리하므로 운영팀이 인프라 관리보다 쿼리 최적화와 데이터 모델링에 집중할 수 있습니다. 다만 Managed 서비스는 자체 운영 대비 추가 비용이 발생하므로, 클러스터 규모와 운영팀 역량을 고려한 총비용(TCO, Total Cost of Ownership) 비교가 필요합니다 [S20].

결론적으로 ClickHouse 는 저장 비용 5x 이상 절감, 전형 집계 쿼리 ~4x 가속, 클러스터 규모 50% 이상 축소라는 세 가지 정량 이점을 복수의 독립 채택사를 통해 검증하였습니다. 이 수치들은 워크로드 조건에 따라 차이가 있지만, 모니터링 인프라 도입 의사결정의 1차 기준치로 활용하기에 충분한 신뢰성을 갖추고 있습니다. 자사 환경에서 같은 결과를 재현하려면 유사 워크로드로 PoC 를 수행하고 자체 데이터 기반 실측치를 확보하는 과정이 필수입니다.

7장: ClickHouse 가 풀고자 한 원문제와 해결 메커니즘

ClickHouse 의 기술 선택 하나하나는 단일한 문제 정의에서 출발합니다. "수십 테라바이트 규모의 이벤트 데이터를 매일 누적하면서, 사용자가 요청하는 임의의 집계 쿼리에 초 단위로 응답해야 한다"는 요건이 그것입니다. 이 장에서는 그 원문제가 어디서 비롯되었는지, 그리고 ClickHouse 가 그 문제를 해결하기 위해 채택한 다섯 가지 메커니즘이 무엇인지 순서대로 정리합니다. 원문제와 해결 메커니즘의 관계를 파악하면 ClickHouse 의 강점과 한계, 그리고 적합한 워크로드와 부적합한 워크로드를 스스로 판단하는 기준이 생깁니다.

7.1 원문제 정의 — Yandex.Metrica 의 분석 트래픽

ClickHouse 의 설계 원점은 2009년 Yandex 내부의 웹 분석 시스템인 Yandex.Metrica 입니다. 이 시스템이 직면한 두 가지 현실이 ClickHouse 의 모든 기술 결정에 영향을 미쳤습니다. 첫째는 데이터의 절대량이 전통적 데이터베이스가 수용할 수 있는 규모를 넘어섰다는 사실이고, 둘째는 그 데이터에 접근하는 쿼리 패턴이 사전에 정의될 수 없는 임의적 성격을 띠고 있다는 점입니다.

7.1.1 페타바이트 누적과 일 단위 수십억 이벤트

Yandex.Metrica 워크로드의 규모

Yandex.Metrica 는 수억 명의 웹 방문자 행동을 페이지뷰·세션·전환·이탈 단위로 수집하는 시스템입니다. 2014년 무렵 이미 일 단위로 수십억 건의 이벤트가 발생했고, 누적 데이터는 페타바이트(PB) 규모에 달했습니다 [S01]. 1 페타바이트는 약 100억 건의 일반 웹 로그 이벤트를 1년 이상 보존한 분량에 해당합니다. 이 규모에서 MySQL 같은 행 지향 OLTP 데이터베이스는 저장 공간을 감당하기 어려웠을 뿐 아니라, 새로운 이벤트를 초당 수십만 건씩 수신하면서 동시에 쿼리를 처리해야 하는 요건도 충족하지 못했습니다 [S03].

저장 비용과 ingest 처리량의 동시 요건

원문제는 저장 비용 문제와 실시간 ingest(데이터 수신·기록) 처리량 문제를 동시에 풀어야 한다는 점에서 특이합니다. 단순히 저장 공간을 늘리는 것으로는 해결이 안 되었습니다. 이벤트가 들어오는 속도와 집계 쿼리를 받아야 하는 속도가 같은 시간에 겹치기 때문입니다. 쓰기 작업이 읽기 성능을 훼손하지 않고, 읽기 작업이 쓰기 처리량을 압박하지 않는 구조가 필요했습니다.

Yandex 내부에서 OLAPServer 와 Metrage 라는 프로토타입 실험을 거쳤지만, 이 두 시스템은 정해진 집계 패턴만 처리할 수 있었고 임의 차원의 쿼리에는 대응하지 못했습니다 [S01].

보존 정책 분리 요건

페타바이트 규모 데이터를 모두 동일한 비용으로 유지하는 것은 현실적이지 않습니다. 최근 7일 이내의 데이터는 빠른 응답이 필요하고, 1년 이상 된 데이터는 접근 빈도가 낮아 저비용 저장소에 옮겨도 됩니다. 이 '핫 데이터와 콜드 데이터의 분리' 요건 역시 원문제의 일부였습니다 [S06]. 전통적 OLTP 데이터베이스는 이 계층 분리를 자체적으로 지원하지 않았습니다.

7.1.2 임의 차원 ad-hoc 집계 쿼리 — 정해진 인덱스로는 풀 수 없는 문제

임의 집계 쿼리의 본질

웹 분석 시스템의 쿼리 패턴은 예측하기 어렵습니다. "이 페이지를 방문한 사용자 중 특정 국가에서 모바일 브라우저로 접속한 비율이 지난 30일 동안 어떻게 변했는가"처럼, 분석가가 그때그때 원하는 차원으로 데이터를 잘라 집계합니다. 이를 ad-hoc(임의적) 집계 쿼리라 합니다. 사전에 어떤 차원 조합이 조회될지 알 수 없기 때문에 특정 차원을 미리 인덱싱하는 방식으로는 모든 쿼리를 효율적으로 처리할 수 없습니다 [S03].

B-Tree 인덱스의 구조적 한계

행 지향 데이터베이스가 일반적으로 사용하는 B-Tree 인덱스는 특정 컬럼 값의 정확한 위치를 빠르게 찾는 데 최적화되어 있습니다. 예를 들어 "user_id = 12345"처럼 단일 행을 식별하는 검색은 B-Tree 인덱스가 탁월합니다. 그러나 "전체 데이터에서 국가별로 방문자 수를 집계"하는 쿼리는 수억 건의 행 전체를 스캔해야 합니다. 이 경우 인덱스는 오히려 쓸모가 없으며, 인덱스를 통한 조회보다 테이블 전체를 처음부터 끝까지 읽는 것이 더 빠를 수 있습니다 [S01]. ad-hoc 집계 쿼리는 대부분 이 후자 범주에 속합니다.

ClickHouse 의 Sparse Primary Index 접근

ClickHouse 는 이 문제를 dense index(각 행에 인덱스 항목 존재) 대신 sparse primary index(듬성한 기본 인덱스)로 접근합니다. Sparse Primary Index 는 인덱스가 모든 행이 아닌 일정 간격(기본 8,192행)의 블록 경계에만 존재합니다. 이 구조는 인덱스 크기를 최소화하고, 쿼리 실행 시 건너뛸 수 있는 블록을 식별한 뒤 나머지 컬럼 데이터를 순차적으로 스캔합니다 [S03]. "인덱스 없이 빠르다"는 표현은 정확하지 않습니다. 정확히는 "dense index 없이, 컬럼 지향 스캔과 sparse index 조합으로 집계 쿼리에서 더 빠르다"입니다.

inverted index 를 사용하지 않는 이유

Elasticsearch 가 사용하는 inverted index(역인덱스, 단어 또는 값별로 포함된 문서 목록을 저장하는 구조)는 전문 검색에 탁월합니다. 그러나 고카디널리티 집계(수천만 가지 고유 값이 존재하는 컬럼에 대한 집계)에서는 역인덱스 자체의 크기가 방대해지고 집계 계산 비용도 올라갑니다. ClickHouse 는 역인덱스 대신 컬럼 저장 + 압축 해제 + SIMD 스캔을 결합해 이 문제를 회피합니다 [S01]. 역인덱스를 안 쓰는 것이 아니라, 집계 워크로드에서는 역인덱스보다 이 조합이 더 효율적이기 때문에 선택하지 않은 것입니다.

7.2 I/O · CPU 효율화 메커니즘

원문제를 해결하기 위한 첫 두 가지 메커니즘은 데이터를 읽고 처리하는 과정의 효율화에 집중합니다. 첫째는 디스크에서 읽어야 하는 데이터 양 자체를 줄이는 I/O 최소화이고, 둘째는 읽은 데이터를 CPU 가 가장 효율적으로 처리하도록 하는 벡터화 실행입니다. 여기에 쓰기와 읽기를 구조적으로 분리하는 MergeTree 메커니즘이 결합됩니다.

7.2.1 I/O 최소화 — 필요 컬럼만 디스크에서 읽음

컬럼 지향 저장의 I/O 절감 원리

행 지향 데이터베이스는 디스크에 데이터를 행 단위로 저장합니다. "서울 지역 방문자 수"를 집계하는 쿼리가 실행될 때, 행 지향 DB 는 각 행의 모든 컬럼(이름, 이메일, 세션 길이, 브라우저 종류 등 수십 개)을 디스크에서 읽고 나서 그중 지역 컬럼만 사용합니다. 불필요한 데이터를 대량으로 읽는 것입니다. 컬럼 지향 저장 방식은 같은 컬럼의 데이터를 디스크에 연속으로 배치합니다. "지역" 컬럼만 필요한 쿼리는 지역 컬럼 파일만 읽고 다른 컬럼은 건드리지 않습니다 [S03]. 이 구조가 I/O 최소화의 출발점입니다.

압축률과의 시너지

같은 유형의 데이터가 연속으로 배치되면 압축 알고리즘의 효율이 높아집니다. "한국", "미국", "독일"처럼 반복 패턴이 많은 문자열은 연속으로 모여 있을 때 ZSTD 나 LZ4 알고리즘이 높은 비율로 압축할 수 있습니다. ClickHouse 는 로그 데이터에서 10:1~20:1 의 압축률을 달성합니다 [S21]. 같은 데이터가 Elasticsearch 에서는 약 1.5:1 수준에 머무는 것과 대비됩니다. 압축률이 높아지면 디스크에 저장된 데이터 양이 줄어들고, 그만큼 읽어야 하는 바이트 수도 줄어 I/O 비용이 다시 감소합니다. 컬럼 지향 저장과 압축은 I/O 절감 측면에서 서로를 강화하는 구조입니다.

CPU 캐시 정합 효과

컬럼 저장은 CPU 캐시 성능에도 영향을 미칩니다. 현대 CPU 는 데이터를 캐시 라인(보통 64바이트) 단위로 메모리에 올립니다. 집계 쿼리가 같은 컬럼의 연속된 값을 처리할 때, 이 데이터는 이미 캐시에 올라 있을 가능성이 높습니다. 반면 행 지향 방식에서는 집계에 필요한 컬럼이 각 행에 흩어져 있어 캐시 미스(캐시에 없는 데이터를 찾는 상황)가 빈번히 발생합니다 [S03]. 이 차이가 쿼리 실행 시간에 실질적인 영향을 줍니다.

7.2.2 CPU 효율 극대화 + 쓰기/읽기 분리 — SIMD · 벡터화 + MergeTree

벡터화 실행과 SIMD 명령어

ClickHouse 는 SIMD(Single Instruction Multiple Data, 단일 명령으로 다수 데이터를 동시 처리하는 CPU 명령어 집합) 를 적극 활용합니다. 일반적인 CPU 연산은 한 번의 명령으로 데이터 1개를 처리합니다. 반면 SIMD 명령어는 한 번의 명령으로 데이터 4개, 8개, 또는 16개를 동시에 처리할 수 있습니다. ClickHouse 는 컬럼 데이터를 블록 단위로 묶어 SIMD 명령어로 처리하는 벡터화 실행(vectorized execution) 엔진을 갖추고 있습니다 [S03]. 이 구조가 같은 컬럼 DB 인 Vertica, Greenplum 대비 5x~24x 빠른 집계 성능의 근거입니다.

MergeTree 의 쓰기/읽기 분리 원리

MergeTree 엔진은 쓰기와 읽기를 구조적으로 분리합니다. 새 데이터가 INSERT 될 때 ClickHouse 는 그 데이터를 즉시 기존 데이터와 병합하지 않고, 별도의 파트(part)로 디스크에 씁니다. 이 쓰기 작업은 매우 빠릅니다. 이후 백그라운드에서 증분 병합(incremental merge) 이 실행되어 파트들을 정렬·압축·통합합니다 [S01]. SELECT 쿼리는 현재 존재하는 파트들을 대상으로 실행되며, 병합이 진행 중이어도 쿼리가 차단되지 않습니다. 이 설계 덕분에 수백만 rows/sec 수준의 ingest 처리량과 ms 단위 집계 응답을 동시에 유지할 수 있습니다 [S09].

같은 하드웨어로 더 많은 처리

SIMD 벡터화와 MergeTree 의 쓰기/읽기 분리가 결합되면 동일한 서버 하드웨어로 처리할 수 있는 쿼리 수가 크게 늘어납니다. Uber 는 ELK(Elasticsearch-Logstash-Kibana) 스택에서 ClickHouse 로 전환한 뒤 클러스터 노드 수를 50% 이상 줄이면서도 더 많은 쿼리를 처리했습니다 [S10]. 이 효과는 추가 하드웨어 투자 없이 현재 인프라에서 더 많은 모니터링 데이터를 소화할 수 있다는 의미입니다. 모니터링 인프라 운영 비용 절감의 직접적인 기술 근거가 바로 이 메커니즘입니다.

7.3 분산·운영 효율 메커니즘

원문제의 세 번째 측면은 규모 확장성입니다. 페타바이트 데이터를 단일 서버에 담을 수 없으므로 여러 서버에 분산·복제하는 메커니즘이 필요합니다. 동시에 운영자가 이 분산 시스템을 현실적인 인력으로 관리할 수 있어야 합니다. 이 장의 마지막 두 메커니즘은 수평 확장 가능한 분산·복제 구조와 운영 복잡도 최소화를 다룹니다.

7.3.1 분산·복제 — ReplicatedMergeTree + ClickHouse Keeper

ReplicatedMergeTree 의 동작 원리

ClickHouse 는 MergeTree 엔진에 복제 기능을 결합한 ReplicatedMergeTree 를 제공합니다. 이 엔진을 사용하면 같은 데이터를 여러 노드에 자동으로 복제합니다. 복제는 멀티-마스터(multi-master) 비동기 방식으로 동작합니다. 어떤 노드에 데이터를 써도 복제가 이루어지며, 단일 실패 지점(SPOF)이 없는 구조입니다 [S03]. 복제 과정에서 메타데이터 조율을 위해 별도의 조율 서비스가 필요한데, ClickHouse 는 이를 위해 ZooKeeper 또는 ClickHouse Keeper 를 사용합니다.

ClickHouse Keeper 의 역할

ClickHouse Keeper(ClickHouse 내장 조율 서비스)는 ZooKeeper(분산 시스템 메타데이터 조율 오픈소스 서비스)를 대체하기 위해 ClickHouse Inc. 가 개발한 경량 조율 서비스입니다. ZooKeeper 는 외부 서비스로 별도 운영해야 하는 반면, ClickHouse Keeper 는 ClickHouse 프로세스 안에 통합되어 있어 운영 부담이 줄어듭니다 [S01]. 2026년 현재 신규 ClickHouse 클러스터는 ClickHouse Keeper 를 기본으로 사용합니다.

Distributed 테이블과 수평 확장

수평 확장은 Distributed 테이블을 통해 이루어집니다. Distributed 테이블은 여러 샤드(shard, 데이터를 분할해 저장하는 논리적 단위)에 분산된 데이터를 단일 테이블처럼 보이게 하는 래퍼(wrapper) 테이블입니다. 쿼리가 Distributed 테이블로 들어오면 ClickHouse 가 각 샤드에 서브

쿼리를 분산 전송하고 결과를 취합해 반환합니다. Cloudflare 는 이 구조로 2024년 12월 기준 2 PiB(페비바이트, 약 2,000 테라바이트) 이상의 데이터를 클러스터에서 운영합니다 [S09]. 페타바이트 규모의 분석 워크로드가 이 구조 위에서 실제로 동작한다는 점이 확인된 사례입니다.

복제와 가용성의 균형

ReplicatedMergeTree 는 복제 팩터(보통 2~3)를 설정합니다. 노드 하나가 장애를 일으켜도 다른 복제본이 쿼리를 받습니다. 단, ClickHouse 의 복제는 eventual consistency(결과적 일관성, 모든 복제본이 최종적으로 같은 상태가 되지만 즉시 동기화되지 않는 모델) 를 기반으로 합니다 [S01]. 엄격한 읽기 일관성이 필요한 OLTP 워크로드에는 이 모델이 맞지 않지만, 모니터링·분석 워크로드에서는 수 초 이내의 복제 지연이 운영에 지장을 주지 않습니다.

7.3.2 단일 바이너리 + 의존성 최소화 — Pinot · Druid 의 다층 구조와 대비

ClickHouse 의 단일 바이너리 설계

ClickHouse 는 단일 바이너리(하나의 실행 파일) 로 배포됩니다. 설치하면 ClickHouse 서버가 데이터 저장, 쿼리 실행, 복제 조율(ClickHouse Keeper 통합)을 모두 담당합니다. 별도의 코디네이터 서비스, 인제스션(ingest) 서비스, 인덱싱 서비스가 필요하지 않습니다 [S23]. 이 설계 선택은 운영 복잡도를 낮추는 직접적인 수단입니다.

Druid · Pinot 의 다층 아키텍처와 비교

Apache Druid 와 Apache Pinot 은 ClickHouse 와 같이 대규모 OLAP 워크로드를 처리하는 오픈소스 데이터베이스입니다. 이 두 시스템은 브로커(쿼리 라우팅), 코디네이터(메타데이터 관리), 오버로드(ingest 관리), 히스토리컬(쿼리 실행) 등 여러 역할의 컴포넌트로 구성됩니다 [S24]. 각 컴포넌트를 별도로 배포하고 스케일링해야 하며, 장애 시 어떤 컴포넌트가 문제인지 진단하는 과정도 복잡합니다. ClickHouse 는 이런 다층 구조 없이 단일 프로세스로 동일한 역할을 수행합니다.

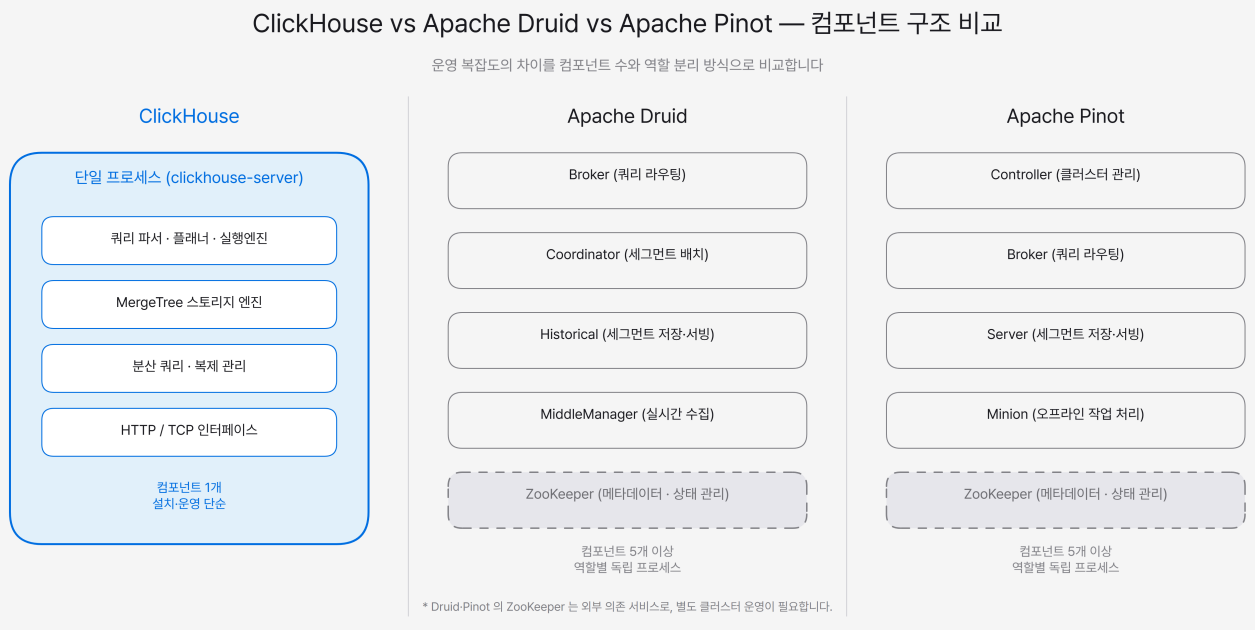
단순함의 장점과 자동화 도구의 제약

단일 바이너리 설계의 장점은 운영 학습 진입 장벽이 낮다는 점입니다. 설치, 업그레이드, 장애 진단 모두 단일 컴포넌트를 대상으로 진행됩니다. 반면 Pinot 와 Druid 는 다층 구조에 맞게 자동화 도구가 발전해 있습니다. Kubernetes 오퍼레이터(클러스터 자원을 자동으로 관리하는 컴포넌트), Helm 차트(Kubernetes 배포 템플릿), 운영 정책 자동화 도구 등이 Pinot/Druid 생태계에 더 풍부합니다 [S23]. ClickHouse 도 Kubernetes 오퍼레이터와 Altinity Kubernetes 오퍼레이터가 존재하지만, 자동화 성숙도에서는 차이가 있습니다. 9장에서 다루는 운영 학습곡선 문제는 이 자동화 도구의 차이에서 비롯됩니다.

균형 잡힌 평가

ClickHouse 의 단순한 아키텍처는 소수 인력이 운영하는 팀에게 유리합니다. 설치와 유지보수에 필요한 전문 지식의 범위가 좁기 때문입니다. 그러나 대규모 클러스터에서 세밀한 컴포넌트별 리소스 조율이 필요할 때는 Pinot 나 Druid 의 명시적 컴포넌트 분리가 유리할 수 있습니다. 의사결정권자가 워크로드 규모와 운영 팀 규모를 함께 고려해 선택해야 하는 이유가 여기에 있습니다 [S24]. ClickHouse 가 원문제(수십 TB/일 이벤트의 초 단위 집계)를 해결하는 데 최적화된

것처럼, 운영 구조도 그 문제를 염두에 두고 단순화되었다는 점을 이해하면 선택 기준이 명확해집니다.



캡션: ClickHouse vs Druid vs Pinot — 컴포넌트 구조 비교

8장: AI 시대에 ClickHouse 가 더 중요해지는 이유

AI와 LLM(Large Language Model, 대규모 언어 모델)이 기업 운영 환경에 빠르게 스며들면서 분석 인프라에 대한 요구가 달라지고 있습니다. 기존의 모니터링은 서버 CPU 사용률이나 HTTP 응답 시간처럼 규칙적으로 생성되는 숫자형 지표를 주로 다뤘지만, LLM 기반 서비스는 이와 성격이 다른 새로운 데이터를 대량으로 만들어 냅니다. 모델 호출 하나에 연결된 프롬프트 텍스트, 토큰 수, 응답 지연, 품질 평가 결과, 비용이 동시에 기록되어야 하며 이 기록은 수억 건 단위로 누적됩니다. 이 장에서는 그 변화가 왜 ClickHouse 의 위치를 이전보다 중요하게 만드는지, 그리고 ClickHouse Inc. 스스로 어떤 방향으로 사업을 재편하고 있는지를 정리합니다.

8.1 LLM observability 의 부상과 Langfuse 인수

LLM observability(관측가능성)란 LLM 호출의 트레이스(trace), 비용, 품질 메트릭을 통합하여 분석하는 영역을 가리킵니다. 기존 애플리케이션 모니터링이 "서버가 살아있는가"를 확인하는 데 집중했다면, LLM observability는 "AI 가 기대한 수준으로 동작하는가, 그리고 얼마나 비용이 드는가"를 측정합니다. 이 영역이 빠르게 성숙하면서 ClickHouse 와의 접점이 생겨났으며, ClickHouse Inc. 의 Langfuse 인수는 그 흐름을 공식화한 사건이었습니다.

LLM 호출이 만들어내는 데이터의 특성

GPT-4 계열이나 Claude, Gemini 같은 LLM을 제품에 적용하면 호출 하나마다 고정되지 않은 크기의 데이터가 생성됩니다. 입력 토큰 수, 출력 토큰 수, 모델 버전, 응답 지연(latency), 사용 비용, 사용자 식별자, 요청 목적 분류, 평가 점수 등 10개 이상의 필드가 호출 단위로 기록되어야

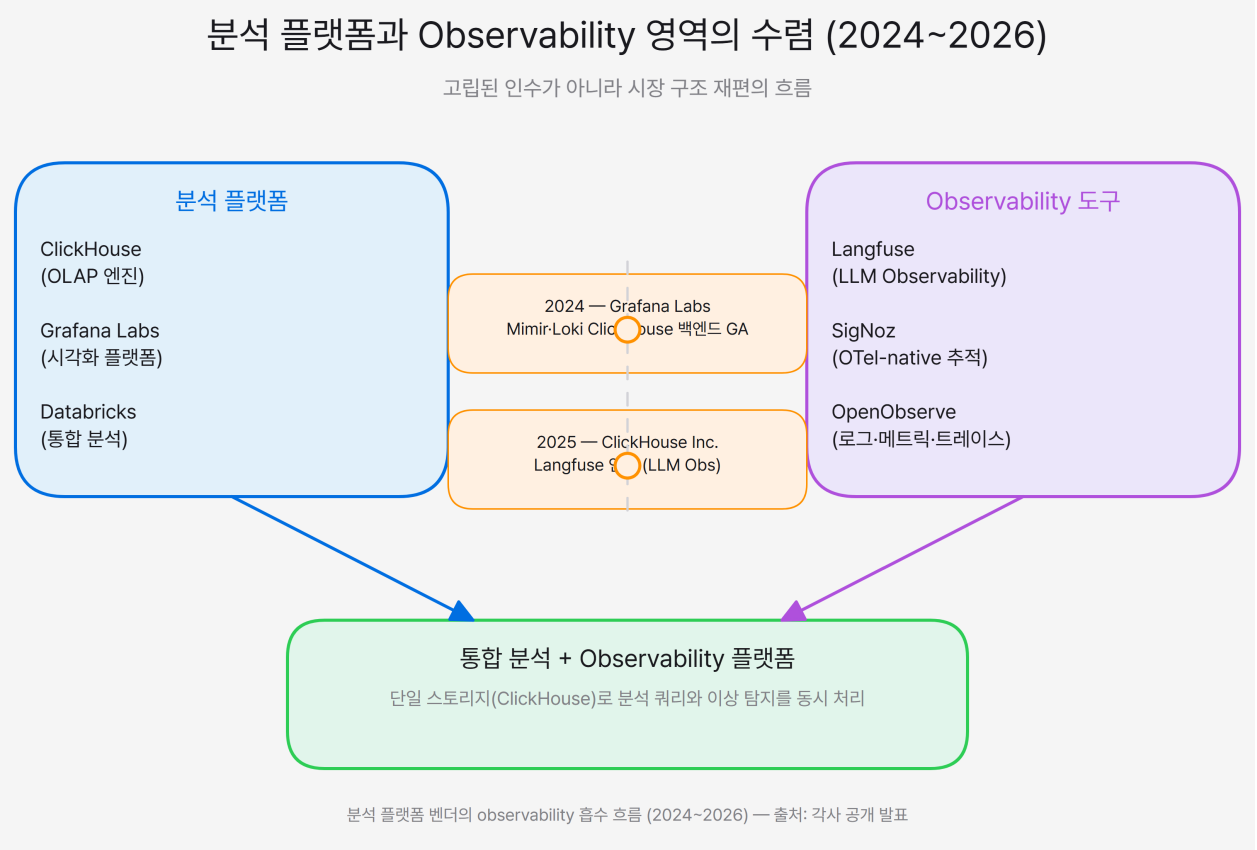
합니다. 하루 수백만 건의 LLM 호출을 운용하는 서비스라면 이 데이터는 수 TB 규모로 매일 누적됩니다. 이 데이터에서 "어떤 사용자 세그먼트에서 응답 품질이 낮은가", "모델 버전 변경 후 비용이 얼마나 증가했는가" 같은 임의 집계 쿼리에 초 단위로 답해야 하는 요건은 앞서 살펴본 ClickHouse 의 원문제 정의와 정확히 일치합니다 [S05].

LLM observability 시장의 형성

2024년부터 2026년 사이에 LLM observability를 전담하는 오픈소스 도구들이 등장했습니다. Langfuse는 그 중 가장 빠르게 성장한 프로젝트 가운데 하나입니다. LLM 호출 트레이스를 저장하고, 비용을 집계하며, 응답 품질을 평가하는 기능을 단일 플랫폼에서 제공합니다. Langfuse는 초기에 PostgreSQL을 백엔드로 사용했으나, 트레이스 데이터 규모가 커질수록 집계 쿼리 응답 속도가 수용하기 어려운 수준으로 느려졌습니다. 이 한계를 해소하기 위해 Langfuse v3에서 PostgreSQL에서 ClickHouse로 분석 백엔드를 전환했으며, 전환 이후 대규모 트레이스 데이터에 대한 집계 쿼리 성능이 크게 개선되었습니다 [S05]. 이 전환은 "LLM observability 데이터는 규모가 커지면 컬럼 지향 DB가 필요하다"는 결론을 실증한 사례입니다.

Langfuse 인수가 보낸 시장 신호

ClickHouse Inc.는 2026년 Langfuse를 인수했습니다. 이 인수는 단순히 유용한 도구를 확보한 것으로 볼 수 없습니다. ClickHouse Inc. 관점에서 이 결정은 "데이터 분석 플랫폼"과 "AI 시스템 관측가능성"이라는 두 영역이 기술적으로 연결되어 있다는 판단을 반영합니다. 분석 플랫폼 벤더가 observability 도구를 흡수하는 흐름은 ClickHouse Inc. 외에도 나타나고 있습니다. 데이터 웨어하우스 사업자가 로그 분석 기능을 추가하거나, APM 벤더가 데이터 플랫폼 기능을 확장하는 방향으로 시장 경계가 흐릿해지고 있습니다 [S05]. Langfuse는 인수 이후에도 오픈소스 라이선스와 셀프호스팅, 클라우드 서비스를 모두 유지하기로 약속했습니다. 이 거버넌스 약속은 기존 Langfuse 사용자 기반을 보호하면서 ClickHouse 생태계와의 통합을 점진적으로 강화하는 전략으로 읽힙니다.



캡션: 분석 플랫폼 벤더의 observability 흡수 흐름 (2024~2026)

8.2 Feature Store · RAG 분석 · Agent 행동 로그

LLM observability는 AI 워크로드가 ClickHouse와 만나는 접점 가운데 하나에 불과합니다. Feature Store(피처 스토어), RAG(Retrieval-Augmented Generation, 검색 증강 생성), 그리고 AI Agent 시스템은 각각 성격이 다른 고볼륨 분석 요건을 만들어 냅니다. 이 세 영역 모두에서 대규모 이벤트 데이터를 빠르게 집계해야 하는 공통된 요구가 있으며, 그 요구에 ClickHouse 가 부합하는 이유를 정리합니다.

피처 스토어와 임베딩 메타데이터 분석

피처 스토어(Feature Store)는 머신러닝 모델 학습과 추론에 사용하는 특징값(feature)을 중앙 집중형으로 관리하는 시스템입니다. 피처 스토어의 핵심 운영 과제 중 하나는 피처 값이 시간에 따라 어떻게 분포가 변했는지 추적하는 것입니다. 이를 피처 드리프트(feature drift) 모니터링이라 하며, 수백 개의 피처에 대해 일별·주별 분포 변화를 집계하는 연산이 필요합니다. 임베딩 기반 시스템이라면 고차원 벡터의 분포 변화를 요약한 메타데이터를 분석하는 요건도 추가됩니다. 이 데이터는 수억 행 규모의 피처 값 로그에서 집계 쿼리로 답해야 하는 전형적인 OLAP 패턴에 해당하며, ClickHouse의 컬럼 지향 저장과 벡터화 실행이 직접적인 이점을 제공하는 상황입니다 [S04].

RAG(검색 증강 생성)는 LLM이 답변을 생성할 때 외부 지식 저장소에서 관련 문서를 검색하여 맥락으로 제공하는 방식입니다. RAG 시스템을 운영하는 팀이 마주하는 문제는 검색 품질 메트릭의 지속적인 분석입니다. 사용자 쿼리마다 어떤 문서가 검색됐는지, 검색 정밀도(precision)와

재현율(recall)이 어느 수준인지, 응답 지연이 어떻게 분포하는지를 대규모로 집계하여 검색 로직 개선의 근거로 삼아야 합니다. 이 분석은 수천만 건의 검색 이벤트 로그 위에서 실행되며, 임의 조건을 조합한 집계 쿼리가 반복적으로 발생합니다 [S04]. ClickHouse 가 이 패턴에 잘 맞는 이유는 앞선 장들에서 설명한 것과 동일합니다. 대용량 이벤트 데이터에서 임의 집계에 초 단위로 응답하는 것이 ClickHouse 의 설계 목표였기 때문입니다.

Agent 행동 로그와 100% 트레이스 보존

AI Agent 시스템은 사용자 요청을 받아 여러 도구를 순차적으로 호출하며 작업을 수행합니다. 웹 검색, 코드 실행, API 호출 같은 도구 사용 기록이 단계별로 쌓이며, 실패한 단계의 원인 분석과 전체 비용 추적이 운영의 핵심 과제가 됩니다. 멀티 에이전트(multi-agent) 구조에서는 에이전트 간 호출 흐름까지 추적해야 하므로 트레이스 데이터의 구조가 더 복잡해지고 볼륨도 증가합니다.

이 요건에서 가장 중요한 운영 원칙은 "100% 트레이스 보존"입니다. 샘플링 방식으로 일부만 기록하면 드문 실패 사례가 누락되어 원인 분석이 불가능해집니다. 그러나 100% 보존은 저장 비용이 문제가 됩니다. Cloudflare는 35~45M req/sec 규모의 HTTP 트래픽을 100% 기록하고 있으며, 이 전략이 가능한 이유로 ClickHouse 의 압축 효율을 명시했습니다. Cloudflare 의 경우 Elasticsearch 대비 문서당 저장 크기를 10분의 1 수준으로 줄인 결과, 이전에는 샘플링이 필요했던 데이터를 전량 보존할 수 있게 되었습니다 [S07]. AI Agent 시스템의 트레이스 보존도 동일한 논리로 접근할 수 있습니다. 저장 비용을 1/5~1/10 수준으로 줄일 수 있다면 100% 보존이 경제적으로 실현 가능해지며, 그 기반 위에서 비용 최적화와 실패 분석이 가능해집니다 [S07].

8.3 ClickHouse Inc. 의 전략 재포지셔닝

기술적 요건 외에, ClickHouse Inc. 자체가 어떤 방향으로 사업을 재정의하고 있는지도 의사결정의 중요한 변수입니다. 단순히 "성능 좋은 오픈소스 DB"로만 바라보던 시각을 넘어서, ClickHouse Inc. 는 분석 플랫폼 영역으로 사업 범위를 확장하고 있습니다. 자본 시장의 반응과 제품 전략의 방향 모두 같은 신호를 가리키고 있습니다.

자본 시장이 보내는 신호

ClickHouse Inc. 는 2024년 4억 달러 규모의 투자를 유치하며 기업 가치 평가액 150억 달러를 기록했습니다. 이 수치는 "단일 오픈소스 DB 벤더"에게 부여되는 수준을 넘어서 있습니다 [S05]. 비교를 위해 분석 플랫폼 시장의 다른 참여자들을 살펴보면, Snowflake 는 클라우드 데이터 웨어하우스의 대표 사업자로 상장 후 수십조 원 이상의 시가총액을 형성했으며, Databricks 는 2023년 기업 가치 평가액 430억 달러를 기록한 바 있습니다. ClickHouse Inc. 의 150억 달러 평가는 이 두 사업자와 동일한 의사결정 트랙에 올라간 것을 의미합니다. 투자자들이 ClickHouse Inc. 를 "고성능 오픈소스 DB 프로젝트 후원사"가 아닌 "분석 플랫폼 경쟁자"로 분류하고 있다는 뜻입니다. 이 분류는 향후 제품 로드맵의 방향을 예측하는 단서가 됩니다.

분석 플랫폼 카테고리로의 이동

ClickHouse Inc. 의 제품 전략은 단순한 DB 엔진 판매에서 분석 플랫폼 서비스로 무게 중심이 이동하고 있습니다. ClickHouse Cloud 는 서버리스 방식의 완전 관리형 ClickHouse 서비스로, 사

용량 기반 과금과 자동 확장을 제공합니다. Langfuse 인수를 통해 LLM observability 플랫폼 기능을 제품 포트폴리오에 추가했으며, 이는 데이터 수집·저장·분석·시각화를 단일 플랫폼에서 제공하려는 방향을 나타냅니다 [S05].

Snowflake 와 Databricks 가 범용 데이터 플랫폼을 지향하는 반면, ClickHouse 는 "실시간 대규모 집계"라는 특정 강점 영역을 유지하면서 플랫폼으로 확장하는 경로를 선택하고 있습니다. 이 전략적 차이가 의사결정에 미치는 실질적인 영향은 다음과 같습니다. ClickHouse 를 도입하는 것은 단순히 DB 엔진 하나를 추가하는 결정이 아니라, 실시간 분석 역량을 중심으로 AI 워크로드까지 포괄하는 분석 인프라 전략을 채택하는 것과 연결됩니다.

AI 워크로드와의 수렴이 가져오는 전략적 함의

기존 모니터링 인프라와 AI 운영 인프라가 동일한 기술 기반 위에서 수렴하고 있습니다. 기존 서버 메트릭과 로그 분석에 쓰이는 ClickHouse 클러스터가 LLM 호출 트레이스와 피처 드리프트 분석에도 그대로 활용될 수 있습니다. 이 수렴은 인프라 중복 투자를 줄이고, 운영팀이 하나의 분석 엔진에 대한 역량을 축적하여 여러 워크로드를 커버할 수 있게 합니다 [S03].

Langfuse 인수와 Moose 같은 ClickHouse 기반 분석 프레임워크의 등장은 이 수렴 흐름을 가속합니다 [S04]. ClickHouse 위에서 TypeScript 또는 Python 으로 데이터 분석 백엔드를 구축하는 프레임워크인 Moose 는 AI 애플리케이션 개발팀이 분석 인프라를 별도로 구성하지 않고도 LLM 호출 데이터를 즉시 분석할 수 있도록 돕습니다. 이러한 생태계 확장은 ClickHouse 가 단순 엔진에서 플랫폼으로 진화하는 경로를 더욱 명확히 보여줍니다.

국내 IT 의사결정자 입장에서 이 흐름이 갖는 의미는 분명합니다. AI 서비스를 도입하거나 운영하는 조직이라면 LLM 호출 비용 추적, 응답 품질 모니터링, 에이전트 동작 분석을 위한 분석 백엔드가 조만간 필요해집니다. 그 백엔드 후보군에 ClickHouse 를 포함시키는 것은 기술 성숙도, 사례 기반, 자본 시장 신호 세 가지 측면 모두에서 근거가 있는 선택입니다 [S05].

9장: ClickHouse 한계와 부적합 시나리오

ClickHouse 는 모든 워크로드에 적합한 범용 데이터베이스가 아닙니다. 앞선 장들에서 확인한 저장 비용 절감·쿼리 가속·채택사 사례는 특정 워크로드 조건 아래에서 측정된 결과이며, 조건이 달라지면 다른 선택지가 더 나은 결과를 낼 수 있습니다. 의사결정권자가 ClickHouse 를 평가할 때 이 한계를 미리 파악하면 불필요한 PoC 비용을 줄이고 도입 범위를 처음부터 올바르게 설정할 수 있습니다.

본 장은 세 가지 측면에서 부적합 시나리오를 정리합니다. 첫째, ACID 트랜잭션이 필요하거나 빈번한 데이터 수정이 발생하는 경우, 둘째, 운영 인력 규모와 학습 부담이 진입장벽이 되는 경우, 셋째, ClickHouse 가 OLAP(Online Analytical Processing, 온라인 분석 처리) 영역 내에서도 상대적으로 취약한 워크로드 유형이 있는 경우입니다.

9.1 트랜잭션 부재 · UPDATE/DELETE 비용

ClickHouse 의 핵심 설계 원칙은 "대량의 데이터를 빠르게 읽는다"는 단일 목표에 맞춰져 있습니다. 이 선택의 반대급부로, 데이터를 자주 수정하거나 여러 테이블에 걸쳐 원자적으로 처리해

야 하는 워크로드에는 본질적인 제약이 따릅니다. 이 절은 그 제약이 실제 운영에서 어떻게 나타나는지를 구체적으로 정리합니다.

ACID 트랜잭션의 부재

ClickHouse 는 멀티 row ACID(Atomicity, Consistency, Isolation, Durability) 트랜잭션을 지원하지 않습니다 [S06]. 단일 INSERT 블록 내에서의 원자성은 보장하지만, 여러 테이블에 걸쳐 쓰기와 삭제를 묶어 하나의 작업으로 처리하는 전통적 트랜잭션 모델은 제공하지 않습니다. 대신 eventual consistency(최종 일관성) 모델을 채택하여, 복제본 간 데이터 동기화가 즉각적이지 않고 일정 시간 후에 수렴하는 방식으로 동작합니다. 이는 분석 워크로드에서는 충분히 수용 가능한 설계지만, 주문 처리·결제·재고 관리처럼 중간 상태 없이 동시에 여러 레코드를 갱신해야 하는 OLTP(Online Transaction Processing, 온라인 트랜잭션 처리) 워크로드에서는 데이터 정합성의 책임이 애플리케이션 계층으로 이전됩니다.

운영 관점에서 이 제약이 의미하는 바는 명확합니다. 사용자 계정 상태 변경, 결제 승인·취소 같은 동시성 높은 쓰기 작업은 ClickHouse 단독으로 처리해서는 안 됩니다. 이 워크로드에는 PostgreSQL 같은 OLTP 데이터베이스가 더 적합하며, ClickHouse 는 그 분석 집계 계층으로 보완하는 역할이 맞습니다 [S06]. 이를 명확히 구분하면 두 시스템 모두 각자의 강점을 온전히 발휘합니다.

한편, 모니터링·옵저버빌리티 워크로드는 대부분 append-only(추가 전용) 특성을 가집니다. 메트릭·로그·트레이스 데이터는 생성된 후 수정되는 경우가 드물기 때문에, 이 영역에서는 트랜잭션 부재가 실질적인 제약이 되지 않습니다. 의사결정권자는 "ClickHouse 에 트랜잭션이 없다"는 사실보다 "어떤 워크로드에서 이것이 문제가 되는가"를 기준으로 판단해야 합니다.

UPDATE와 DELETE 의 mutation 비용

ClickHouse 에서 데이터를 수정하려면 `ALTER TABLE ... DELETE` 또는 `ALTER TABLE ... UPDATE` 구문을 사용합니다. 그러나 이 명령은 즉시 실행되는 행 단위 갱신이 아니라 mutation(변이) 으로 처리됩니다 [S23]. Mutation 은 해당 파트(part) 전체를 재작성하는 백그라운드 작업으로, 수억 건의 데이터를 포함하는 파트에서 단 수천 건을 삭제하더라도 파트 전체를 새로 기록해야 합니다. 이 과정은 디스크 I/O 와 CPU 를 대량 소비하며, 빈번하게 실행될 경우 쿼리 성능 저하와 디스크 공간 압박을 동시에 유발합니다.

ReplacingMergeTree 엔진을 사용하면 중복 행을 병합 시점에 자동으로 제거하는 방식으로 논리적 UPDATE 를 우회할 수 있습니다. 그러나 이 방법 역시 병합이 완료되기 전까지는 중복 데이터가 일시적으로 존재하고, 실제로 어떤 버전이 최신인지를 쿼리 로직에서 명시적으로 처리해야 한다는 한계가 있습니다 [S23]. GDPR(General Data Protection Regulation, 일반 데이터 보호 규정) 에 따른 개인정보 삭제 요청처럼 빠른 삭제 보장이 필요한 시나리오에서는 별도의 처리 절차를 설계해야 합니다.

9.2 운영 학습곡선 · OLTP 부적합

ClickHouse 는 단일 바이너리 설치와 간단한 초기 구성으로 시작할 수 있습니다. 그러나 분산·복제 환경을 실제 프로덕션 수준으로 운영하는 데는 상당한 학습 투자가 필요합니다. 이 절은 운

영 학습곡선이 의사결정의 변수가 되는 조건과, 고동시성 소규모 조회 워크로드에서의 부적합 사유를 정리합니다.

분산·복제 운영의 학습 부담

ClickHouse 의 분산·복제 환경은 ReplicatedMergeTree 엔진, Distributed 테이블, ClickHouse Keeper 세 가지 컴포넌트의 상호작용에 대한 이해를 전제로 합니다. shard(샤드) 배치 전략, replication factor(복제 계수) 설정, 쿼리 라우팅 구성, 파트 병합 설정 튜닝은 각각 독립적인 학습 영역을 형성합니다 [S23]. Apache Pinot 나 Apache Druid 는 자동화된 클러스터 관리 도구와 웹 기반 운영 콘솔을 갖추고 있어 분산 설정의 진입 비용을 낮추는 반면, ClickHouse 는 이에 상응하는 수준의 자동화 도구가 부족한 편입니다 [S24].

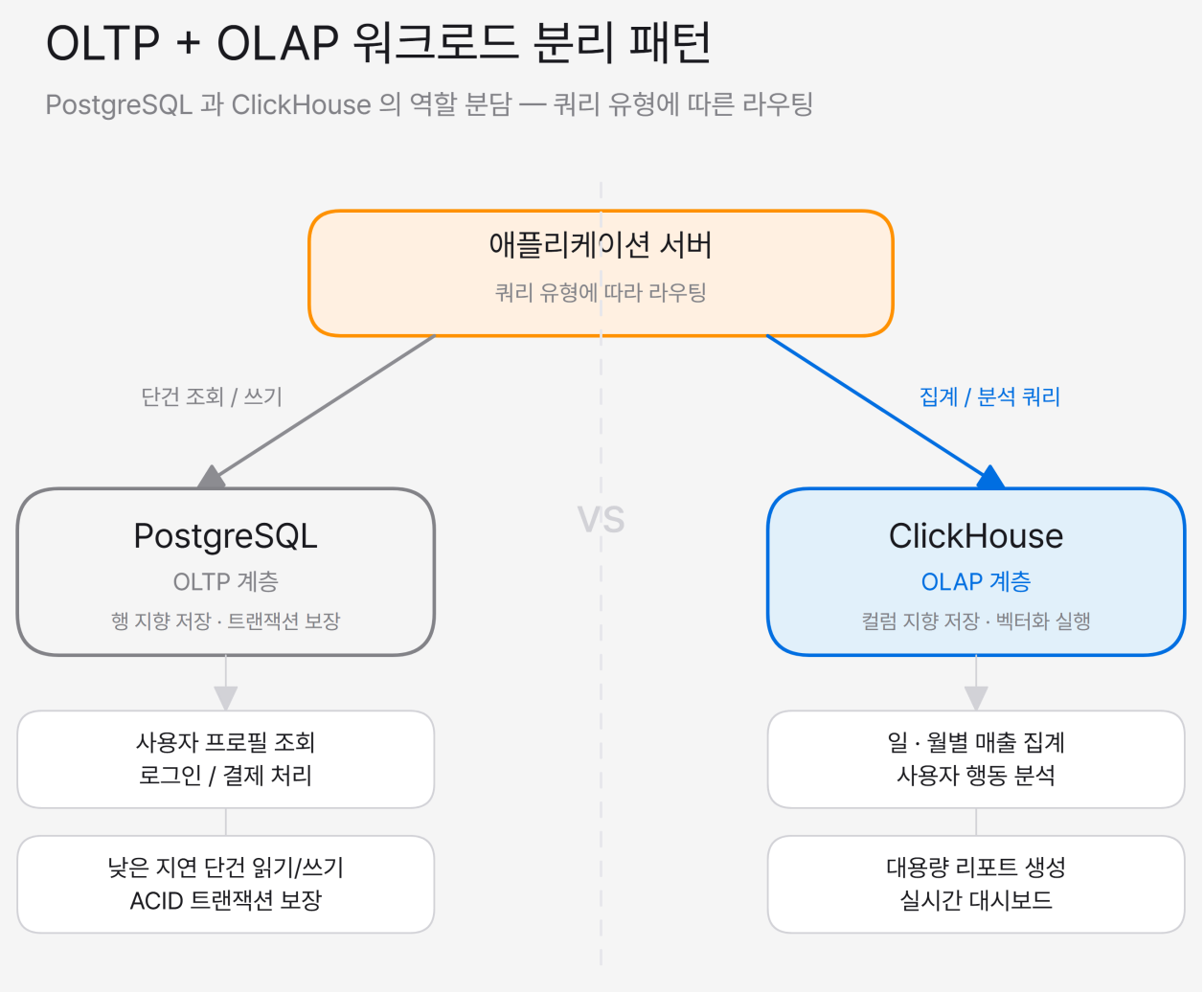
운영팀 규모가 의사결정의 핵심 변수입니다. 경험 있는 ClickHouse DBA 가 포함된 5명 이상의 운영팀이라면 자체 클러스터 운영이 충분히 가능합니다. 그러나 3명 이하의 소규모 팀이라면 ClickHouse Cloud(ClickHouse Inc. 공식 관리형 서비스)나 Altinity.Cloud 같은 Managed 서비스를 통해 클러스터 관리 부담을 위임하는 편이 현실적입니다. Managed 서비스는 월간 비용이 추가되지만, 운영 인건비와 장애 복구 시간을 절감하는 효과로 상쇄될 수 있습니다 [S24].

운영 규모	권장 방식	주요 이유
5명 이상 (전담 DBA 포함)	자체 클러스터 운영	분산 설정·튜닝 역량 확보 가능
3~5명 (겸직 운영)	Managed 서비스 검토	클러스터 관리 부담 집중 완화
3명 이하	Managed 서비스 강력 권장	장애 대응·운영 복잡도 초과 위험

고동시성 소규모 행 단위 조회의 부적합

ClickHouse 의 설계는 소수의 대용량 쿼리를 병렬로 빠르게 처리하는 데 최적화되어 있습니다 [S06]. 반대로 수천 건의 동시 접속 사용자가 각각 1~10개 행을 조회하는 고동시성 소규모 조회 패턴은 ClickHouse 의 강점 구간이 아닙니다. 사용자 프로필 조회, 세션 상태 확인, 실시간 알림 전송 같은 웹 애플리케이션 백엔드 조회 패턴이 여기에 해당합니다.

이 워크로드에는 PostgreSQL 이 훨씬 적합합니다. B-Tree 인덱스를 기반으로 단일 행을 밀리초 단위로 찾아내고, 동시 접속 수천 건을 처리하는 데 최적화된 행 지향 OLTP 데이터베이스가 본 패턴의 자연스러운 선택입니다 [S06]. 따라서 올바른 아키텍처는 "ClickHouse 가 PostgreSQL 을 대체한다"가 아니라 "OLTP 는 PostgreSQL, OLAP 집계는 ClickHouse 가 각각 담당한다"는 역할 분담입니다. PostHog, Langfuse, Sentry 모두 이 분리 패턴을 실제 프로덕션에서 운영하고 있습니다 [S13].



캡션: OLTP + OLAP 워크로드 분리 패턴 — PostgreSQL 과 ClickHouse 의 역할 분담

9.3 초저지연 User-facing OLAP · 큰 Join · 데이터 모델링 종속성

ClickHouse 가 OLAP 영역 전반에서 강점을 보이는 것은 사실이지만, OLAP 내에서도 워크로드 성격에 따라 더 적합한 대안이 존재합니다. 초저지연 단일 사용자 응답이 요건인 경우, 양쪽 테이블 모두 수억 행 규모인 대형 join 이 빈번한 경우, 도입 초기 데이터 모델링 설계의 중요성을 과소평가한 경우가 그 사례입니다.

초저지연 User-facing OLAP 에서 Pinot 가 더 적합한 경우

user-facing OLAP(사용자 직접 응답형 분석)은 수백만 명의 최종 사용자가 각자 자신의 대시보드를 동시에 열고 즉각 응답을 기대하는 워크로드입니다 [S24]. 이 패턴에서 요구되는 응답 지연은 P99 기준 수 밀리초 수준으로, LinkedIn의 팔로워 분석, Twitter의 트윗 노출 통계, 실시간 추천 시스템의 특징 조회 같은 사례가 여기에 해당합니다.

Apache Pinot 는 이 워크로드를 위해 설계된 시스템입니다 [S24]. Star-Tree Index 와 같은 사전 집계 색인 구조를 통해 쿼리 시점의 집계 비용을 극소화하고, 수만 건의 동시 쿼리를 처리하는 데 최적화되어 있습니다. ClickHouse 도 집계 쿼리에서 뛰어난 성능을 보이지만, 초당 수만 건에 달하는 동시 사용자 단위 쿼리 부하에서는 Pinot 가 더 예측 가능한 지연을 보장합니다. Sentry

가 8개 OLAP 후보를 검토한 후 ClickHouse 를 선택한 것은 이 워크로드가 아니라 내부 엔지니어링 분석과 백오피스 집계에 해당하는 패턴이었습니다 [S11].

워크로드 유형	ClickHouse 적합도	Apache Pinot 적합도
내부 엔지니어링 대시보드 집계	높음	보통
백오피스 로그 분석	높음	보통
외부 사용자 실시간 추천 조회	보통	높음
수만 동시 사용자 대시보드	보통	높음

큰 Fact-Fact Join 의 성능 약점

ClickHouse 는 fact 테이블(수억 행 이상)과 소규모 dimension 테이블 간의 join(조인)에서는 효율적으로 동작합니다. 그러나 양쪽 모두 수억 행 이상인 fact-fact join 은 처리 비용이 크게 증가하는 약점 구간입니다 [S23]. 분산 환경에서 대형 join 은 노드 간 데이터 셔플링(shuffling)을 유발하고, 이 과정에서 네트워크 I/O 와 메모리 사용량이 급증합니다. Apache Druid 나 Apache Spark 는 이 패턴에 더 최적화된 실행 계획을 제공합니다.

이 한계는 데이터 모델링으로 일부 완화할 수 있습니다. 자주 join 되는 컬럼을 미리 비정규화(denormalization)하여 단일 테이블에 통합하는 방식은 ClickHouse 가 권장하는 설계 패턴이기도 합니다 [S23]. 그러나 이 접근법은 데이터 중복을 발생시키고, 업데이트 시 여러 컬럼을 동시에 갱신해야 하는 부담을 가져옵니다. 비정규화 설계가 적합하지 않은 복잡한 다단계 join 워크로드라면 ClickHouse 단독 사용보다 데이터 준비 단계를 별도 파이프라인으로 분리하는 방식을 검토해야 합니다.

데이터 모델링 종속성과 사후 수정 비용

ClickHouse 에서 테이블의 primary key 와 ORDER BY 절 선택은 파트 정렬과 병합 동작, 쿼리 성능 전반에 걸쳐 결정적인 영향을 미칩니다 [S23]. 초기 설계에서 잘못된 primary key 를 선택하면 이후 수정 비용이 매우 큽니다. 기존 테이블을 새로운 primary key 로 재구성하려면 전체 데이터를 재적재해야 하며, 수 TB 규모 데이터에서 이 작업은 운영 중단 없이 수행하기 어렵습니다.

partition key(파티션 키), ORDER BY 컬럼, 적합한 MergeTree 변형 선택은 PoC(Proof of Concept, 개념 검증) 단계에서 충분한 시간을 들여 검토해야 합니다. 특히 시계열 데이터에서 event_time 을 ORDER BY 첫 번째 컬럼으로 배치하는 것과 특정 고카디널리티(값의 종류가 수천만 개 이상인) 컬럼을 첫 번째에 배치하는 것은 같은 쿼리에서 수십 배의 성능 차이를 낳을 수 있습니다 [S23]. 도입 초기 데이터 모델링에 충분한 전문 인력을 배정하는 결정이 이후 운영 비용을 결정짓는 핵심 변수입니다.

ClickHouse 의 한계는 이 시스템을 선택하지 말아야 할 이유가 아니라, 어떤 범위에서 선택해야 하는지를 명확히 하는 정보입니다. 트랜잭션이 필요한 OLTP 는 PostgreSQL 로, 초저지연 user-facing OLAP 은 Pinot 로 분리하고, 대규모 append-only 분석 집계 워크로드에 ClickHouse 를 배치하는 설계가 글로벌 채택사들이 공통으로 도달한 결론입니다 [S06][S11][S24]. 부적합 시나

리오를 미리 식별하고 워크로드별 적합한 시스템을 선택하는 것이 가장 합리적인 의사결정 방식입니다.

부록

본 부록은 백서 본문에 인용된 참고문헌과 본문에 등장하는 주요 기술 용어 정의를 모아 정리한 참조 자료입니다.

부록 A. References

1. ClickHouse. (2024). "ClickHouse Open Source: 10 Years." https://clickhouse.com/blog/open-source-10
2. GeekNews. (2021). "ClickHouse 별도 회사 설립." https://news.hada.io/topic?id=5051
3. GeekNews. (2022). "ClickHouse — 오픈소스 컬럼 기반 OLAP DB 소개." https://news.hada.io/topic?id=4356
4. GeekNews. (2024). "Moose — ClickHouse 기반 분석 백엔드 프레임워크." https://news.hada.io/topic?id=20600
5. GeekNews. (2026). "ClickHouse, Langfuse 인수." https://news.hada.io/topic?id=25932
6. GeekNews. (2024). "2025년을 위한 7개 데이터베이스." https://news.hada.io/topic?id=18153
7. Cloudflare Engineering. (2019). "HTTP Analytics for 6M requests per second using ClickHouse." https://blog.cloudflare.com/http-analytics-for-6m-requests-per-second-using-clickhouse/
8. ClickHouse. (2023). "Trouble will find you: Cloudflare at quadrillion-row scale." https://clickhouse.com/blog/cloudflare
9. Cloudflare Engineering. (2024). "Our billing pipeline was suddenly slow: a ClickHouse query plan contention story." https://blog.cloudflare.com/clickhouse-query-plan-contention/
10. Uber Engineering. (2023). "Fast and reliable schema-agnostic log analytics platform at Uber." https://www.uber.com/en-DE/blog/logging/
11. Sentry Engineering. (2019). "Introducing Snuba: Sentry's New Search Infrastructure." https://blog.sentry.io/introducing-snuba-sentrys-new-search-infrastructure/
12. Sentry Engineering. (2025). "How Sentry queries unstructured data in ClickHouse 62x faster." https://sentry.engineering/blog/how-sentry-queries-unstructured-data-in-clickhouse-62x-faster/
13. PostHog Engineering. (2022). "How we turned ClickHouse into our event mansion." https://posthog.com/blog/how-we-turned-clickhouse-into-our-eventmansion

14. PostHog Engineering. (2022). "In-depth: ClickHouse vs PostgreSQL." https://posthog.com/blog/clickhouse-vs-postgres
15. ClickHouse. (2023). "SigNoz — observability solution with ClickHouse and OpenTelemetry." https://clickhouse.com/blog/signoz-observability-solution-with-clickhouse-and-open-telemetry
16. SigNoz. (2026). SigNoz 공식 사이트. https://signoz.io/
17. OpenObserve. (2024). "What backends support OpenTelemetry (OTLP)?" https://openobserve.ai/blog/opentelemetry-backends-otlp-support/
18. Apache Software Foundation. (2004). Apache License, Version 2.0. https://www.apache.org/licenses/LICENSE-2.0
19. ClickHouse GitHub. (2026). ClickHouse/ClickHouse — LICENSE 파일 (2026년 6월 기준 Apache 2.0 유지 확인). https://github.com/ClickHouse/ClickHouse/blob/master/LICENSE
20. Altinity. (2024). "ClickHouse is Apache 2.0." https://altinity.com/blog/clickhouse-is-apache-2-0
21. ClickHouse. (2024). "ClickHouse vs Elasticsearch: The Billion-Row Matchup." https://clickhouse.com/blog/clickhouse_vs_elasticsearch_the_billion_row_matchup
22. ClickHouse. (2024). "Do you still need Elasticsearch for log analytics?" https://clickhouse.com/blog/elasticsearch-log-analytics-clickhouse
23. Leventov, Roman. (2021). "Comparison of the Open Source OLAP Systems for Big Data: ClickHouse, Druid, and Pinot." https://leventov.medium.com/comparison-of-the-open-source-olap-systems-for-big-data-clickhouse-druid-and-pinot-8e042a5ed1c7
24. StarTree. (2023). "A tale of three real-time OLAP databases." https://startree.ai/resources/a-tale-of-three-real-time-olap-databases/

부록 B. Glossary

용어	정의
ACID	Atomicity(원자성) · Consistency(일관성) · Isolation(격리성) · Durability(지속성) — 데이터베이스 트랜잭션이 보장해야 할 4가지 속성.
ad-hoc 쿼리	사전에 패턴이 정해지지 않은 임의 집계 질의. 분석 담당자가 그때그때 원하는 차원으로 데이터를 잘라 집계한다.
AGPL	Affero GPL — AGPL 소프트웨어를 SaaS 형태로 제공할 때도 전체 소스 코드 공개 의무가 적용되는 카피레프트 라이선스.

용어	정의
AggregatingMergeTree	MergeTree 엔진 변형 중 하나. 집계 함수의 중간 상태를 저장하여 메트릭 사전 집계에 적합하다.
APM	Application Performance Management(애플리케이션 성능 관리) — 애플리케이션의 응답 속도, 오류율, 처리량 등을 실시간으로 추적·분석하는 관리 분야.
append-only	데이터를 추가하기만 하고 수정이나 삭제가 거의 발생하지 않는 데이터 접근 패턴. 로그·메트릭·트레이스 데이터의 전형적 특성.
AVX-512	Intel CPU 의 SIMD 명령어 집합. 한 번의 명령으로 512비트(정수 16개 또는 부동소수 8개)를 동시 처리한다.
B-Tree	데이터베이스가 특정 행을 빠르게 찾기 위해 사용하는 균형 트리 인덱스 자료구조. 단건 조회에 최적화되어 있으며 전체 스캔 집계에는 효율이 낮다.
Buffer 테이블	ClickHouse 에서 데이터를 메모리에 임시 누적한 뒤 임계값 도달 시 배치로 디스크에 쓰는 중간 레이어 테이블.
CDN	Content Delivery Network(콘텐츠 전달 네트워크) — 사용자와 가까운 위치에 콘텐츠를 분산 배포하여 응답 속도를 높이는 인프라.
ClickHouse Cloud	ClickHouse Inc. 가 운영하는 완전 관리형 클라우드 서비스. 사용량 기반 과금과 자동 확장을 제공한다.
ClickHouse Keeper	ZooKeeper 기능을 ClickHouse 내부에 구현한 경량 분산 메타데이터 조율 컴포넌트. 2026년 신규 클러스터의 기본 구성 요소.
CNCF	Cloud Native Computing Foundation(클라우드 네이티브 컴퓨팅 재단) — Kubernetes, OpenTelemetry 등 클라우드 네이티브 오픈소스 프로젝트를 후원하는 재단.
DBA	Database Administrator(데이터베이스 관리자) — 데이터베이스 시스템의 설치·운영·성능 튜닝·보안을 담당하는 전문 인력.
Distributed 테이블	여러 샤드에 걸쳐 분산된 데이터를 단일 테이블처럼 쿼리할 수 있게 해주는 ClickHouse 의 가상 테

용어	정의
	이블.
EBS	Elastic Block Store — AWS 에서 제공하는 블록 스토리지 서비스. 객체 스토리지 대비 단가가 높지만 읽기·쓰기 지연이 낮다.
ELK 스택	Elasticsearch, Logstash, Kibana 를 조합한 로그 수집·저장·시각화 스택.
eventual consistency	최종 일관성 — 복제본 간 데이터가 즉시 동기화되지 않고 일정 시간 후에 수렴하는 일관성 모델.
Feature Store	피처 스토어 — 머신러닝 모델 학습과 추론에 사용하는 특징값(feature)을 중앙 집중형으로 관리하는 시스템.
GDPR	General Data Protection Regulation(일반 데이터 보호 규정) — EU 에서 개인정보 처리 및 삭제 요건을 규정하는 법률.
GPL	GNU General Public License(GNU 일반 공중 라이선스) — 강한 카피레프트 조건을 적용하여 파생 저작물 전체의 소스 코드 공개를 요구하는 라이선스.
granule	그래뉼 — ClickHouse 의 Sparse Primary Index 가 인덱스 항목을 생성하는 기본 단위(기본값 8,192행).
HA	High Availability(고가용성) — 장애 발생 시에도 서비스가 중단 없이 계속 제공되도록 보장하는 설계 목표.
Hot/Cold 계층 스토리지	최근 데이터(핫)는 고속 NVMe SSD 에, 오래된 데이터(콜드)는 S3 호환 객체 저장소에 분리 보관하는 계층형 저장 전략.
I/O	Input/Output(입출력) — 데이터를 디스크에서 읽거나 쓰는 작업. 집계 쿼리 성능에서 I/O 병목은 핵심 제약 요인이다.
ingest	외부 데이터 소스에서 데이터베이스로 데이터를 수신하여 기록하는 작업.
inverted index	역인덱스 — 단어 또는 값별로 해당 값을 포함한 문서 목록을 저장하는 자료구조. Elasticsearch 의 핵심 인덱스 방식으로 전문 검색에 최적화되어 있다.

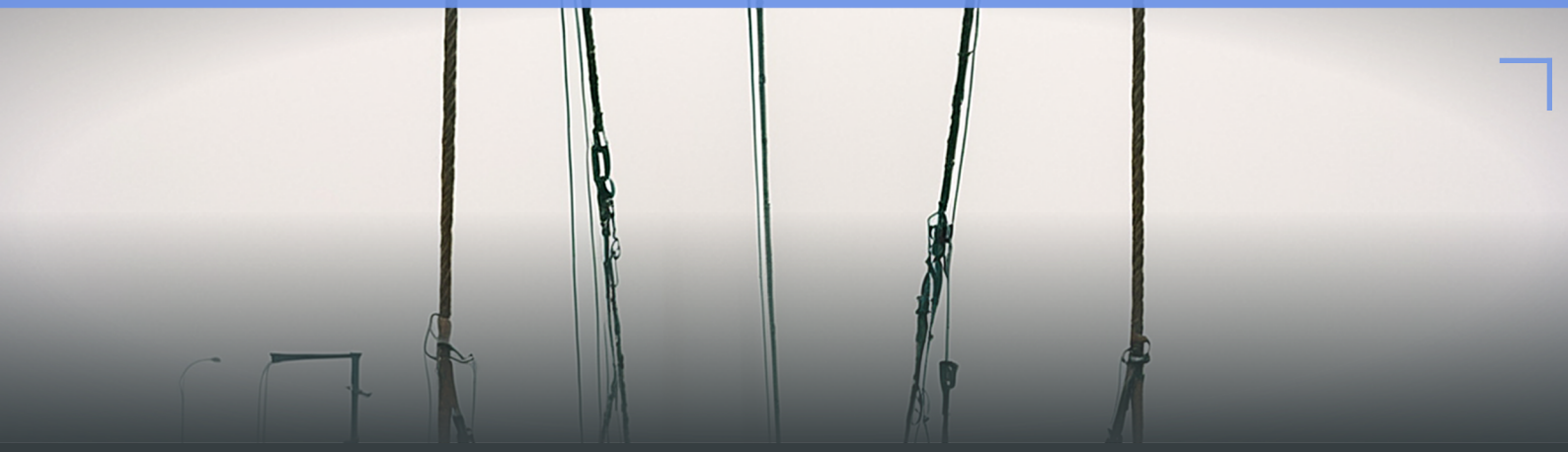
용어	정의
Kafka	Apache Kafka — 고처리량 분산 메시지 스트리밍 시스템. ClickHouse 는 Kafka 엔진을 통해 Kafka 토픽에서 직접 데이터를 가져올 수 있다.
Langfuse	LLM 애플리케이션의 트레이스, 비용, 품질 메트릭을 수집·분석하는 오픈소스 관측가능성 플랫폼. 2026년 ClickHouse Inc. 에 인수됨.
latency	지연 시간 — 요청이 시작되어 응답이 완료될 때까지 걸리는 시간. P95, P99 백분위로 측정한다.
Level 3 오픈소스	소스코드 외에 기여 가이드, 로드맵, CI/CD 파이프라인, 문서, 개발 과정 전체를 공개하는 투명한 오픈소스 운영 방식.
LLM	Large Language Model(대규모 언어 모델) — 대규모 텍스트 데이터로 학습된 언어 생성 모델. GPT-4, Claude, Gemini 등이 해당한다.
LZ4	압축·해제 속도가 매우 빠른 손실 없는 압축 알고리즘. 핫 데이터처럼 자주 조회하는 데이터에 적합하다.
materialized column	구체화 컬럼 — 자주 조회하는 표현식(예: JSON 필드 추출)을 INSERT 시점에 미리 계산하여 별도 컬럼으로 저장하는 최적화 기법.
Materialized View	원본 테이블의 데이터를 집계·변환하여 별도 테이블로 자동 유지하는 뷰. 쿼리 시마다 재집계하지 않아도 되어 응답 속도가 빠르다.
MergeTree	ClickHouse 의 핵심 스토리지 엔진. INSERT 를 즉시 파트로 저장하고 백그라운드에서 점진적으로 정렬·병합하여 ingest 처리량과 쿼리 성능을 동시에 확보한다.
MIT	MIT License — 소스 공개 의무 없이 상업 목적의 사용·수정·재배포를 허용하는 간결한 퍼미시브 라이선스.
Moose	TypeScript 또는 Python 으로 ClickHouse 기반 데이터 분석 백엔드를 신속하게 구축하는 오픈소스 프레임워크.
MPL	Mozilla Public License(모질라 퍼블릭 라이선스) — 수정한 파일에 한해 소스 공개를 요구하는 약한 카피레프트 라이선스.

용어	정의
mutation	ClickHouse 에서 ALTER TABLE ... DELETE 또는 UPDATE 명령을 실행할 때 발생하는 백그라운드 파트 재작성 작업. 실행 비용이 높아 빈번한 사용은 권장하지 않는다.
NVMe SSD	Non-Volatile Memory Express Solid State Drive — PCIe 인터페이스를 통해 매우 낮은 지연과 높은 처리량을 제공하는 고속 저장 장치.
observability	관측가능성 — 시스템의 내부 상태를 외부에서 측정 가능한 출력(로그, 메트릭, 트레이스)으로 파악하는 역량.
OLAP	Online Analytical Processing(온라인 분석 처리) — 수억~수천억 행의 데이터를 스캔하고 집계하는 분석 중심 워크로드. ClickHouse 가 최적화된 영역이다.
OLTP	Online Transaction Processing(온라인 트랜잭션 처리) — 단건 레코드의 빠른 조회·삽입·수정·삭제를 처리하는 트랜잭션 중심 워크로드. PostgreSQL 이 강점을 가진 영역이다.
OTel	OpenTelemetry — 로그, 메트릭, 트레이스를 통합 수집하는 CNCF 오픈소스 원격 측정 표준.
OTLP	OpenTelemetry Protocol — OTel 데이터를 전송하기 위한 표준 통신 프로토콜.
part	파트 — ClickHouse MergeTree 에서 INSERT 시 생성되는 단위 데이터 블록. 백그라운드 병합을 통해 점차 통합된다.
P95 / P99	쿼리 응답 시간의 95번째 또는 99번째 백분위값. 전체 쿼리 중 가장 느린 5% 또는 1% 의 기준 지연 시간을 나타낸다.
PiB	Pebibyte(페비바이트) — 약 1,000TiB(테비바이트)에 해당하는 데이터 용량 단위. Cloudflare 의 ClickHouse 클러스터 규모를 나타내는 데 사용된다.
PoC	Proof of Concept(개념 검증) — 실제 도입 전 소규모 환경에서 기술 적합성과 성능을 확인하는 검증 단계.
PostgreSQL	오픈소스 관계형 DBMS. OLTP 워크로드와 복잡한 JOIN 이 많은 애플리케이션 백엔드에 적합하다.

용어	정의
Projection	동일 데이터를 다른 정렬 순서로 미리 집계·저장하는 ClickHouse 최적화 기법. 다양한 쿼리 패턴을 단일 테이블에서 지원한다.
RAG	Retrieval-Augmented Generation(검색 증강 생성) — LLM 이 답변을 생성할 때 외부 지식 저장소에서 관련 문서를 검색하여 맥락으로 제공하는 방식.
RDBMS	Relational Database Management System(관계형 데이터베이스 관리 시스템) — 테이블 간 관계를 정의하고 SQL 로 데이터를 관리하는 전통적 데이터베이스.
ReplicatedMergeTree	MergeTree 엔진에 자동 복제 기능을 결합한 ClickHouse 엔진. 멀티 마스터 비동기 복제로 단일 장애점(SPOF) 없이 고가용성을 제공한다.
ReplacingMergeTree	MergeTree 엔진 변형 중 하나. 같은 정렬 키를 가진 행 중 최신 버전만 유지하여 이벤트 중복 제거에 적합하다.
RFP	Request for Proposal(제안 요청서) — 사업 발주 기관이 공급사에게 서비스·솔루션 제안을 요청하는 공식 문서.
S3	Amazon Simple Storage Service — AWS 에서 제공하는 객체 스토리지. ClickHouse 는 S3 호환 스토리지를 계층형 스토리지(콜드 계층)로 지원한다.
SaaS	Software as a Service(서비스형 소프트웨어) — 소프트웨어를 인터넷을 통해 서비스 형태로 제공하는 사업 모델.
SBOM	Software Bill of Materials(소프트웨어 구성 성분서) — 소프트웨어에 포함된 오픈소스 컴포넌트와 라이선스를 목록화한 문서.
semantic convention	OTel 에서 필드 이름과 의미를 표준화한 규약. OTel-native 백엔드는 이 규약을 그대로 보존한다.
shard	샤드 — 대규모 데이터를 수평으로 분할하여 여러 노드에 분산 저장하는 논리적 단위.
SIMD	Single Instruction Multiple Data(단일 명령 다중 데이터) — 하나의 CPU 명령으로 여러 데이터를

용어	정의
	동시에 처리하는 CPU 명령어 집합. ClickHouse 벡터화 실행의 핵심 기반이다.
Snuba	Sentry 가 ClickHouse 위에 구현한 OLAP 쿼리 추상화 서비스. Sentry 제품 전반의 분석 쿼리를 단일 계층에서 처리한다.
Sparse Primary Index	희소 기본 인덱스 — 모든 행이 아닌 일정 간격(기본 8,192행 = 1 그래뉼)마다 인덱스 항목을 생성하는 ClickHouse 의 인덱스 구조.
SPOF	Single Point of Failure(단일 장애점) — 해당 구성 요소에 장애가 발생하면 전체 시스템이 중단되는 취약 지점.
SRE	Site Reliability Engineering(사이트 신뢰성 엔지니어링) — 소프트웨어 엔지니어링 방법론으로 서비스 안정성·가용성·성능을 관리하는 운영 분야.
SSPL	Server Side Public License(서버 사이드 퍼블릭 라이선스) — 해당 소프트웨어를 서비스로 제공할 때 서비스 스택 전체의 소스 코드 공개를 요구하는 강한 카피레프트 라이선스. MongoDB, Elasticsearch 가 전환한 바 있다.
SummingMergeTree	MergeTree 엔진 변형 중 하나. 같은 키의 행들을 병합할 때 수치 컬럼을 합산하여 카운터 누적 통합에 적합하다.
TCO	Total Cost of Ownership(총 소유 비용) — 소프트웨어 도입·운영·유지에 드는 직접·간접 비용을 합산한 지표.
trace	트레이스 — 분산 시스템에서 하나의 요청이 여러 서비스를 거치는 전체 실행 흐름을 기록한 데이터.
vectorized execution	벡터화 실행 — 데이터를 행 단위가 아닌 컬럼 벡터(연속 메모리 블록) 단위로 묶어 SIMD 명령어로 처리하는 실행 방식. CPU 캐시 효율과 처리 병렬성을 극대화한다.
ZooKeeper	Apache ZooKeeper — 분산 시스템의 메타데이터 조율·리더 선출을 담당하는 오픈소스 서비스. ClickHouse 에서는 ClickHouse Keeper 로 대체 가능하다.

용어	정의
ZSTD	Zstandard(지스탠다드) — 페이스북이 개발한 고성능 압축 알고리즘. ClickHouse 기본 압축 방식으로 로그 데이터에서 10:1~20:1 압축률을 달성한다.



Observability 를 위한 Database ClickHouse

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