TECH BRIEF

CLOUD NATIVE FORUM

Datadog 대신 오픈소스 모니터링, SigNoz가 가능한가

옵저버빌리티(observability

옵저버빌리티(observability, 관측 가능성)는 시스템 내부에서 무슨 일이 벌어지는지를 외부에서 수집한 데이터만으로 추론할 수 있는 능력을 뜻합니다. 마이크로서비스(microservice, 작은 단위로 쪼갠 독립 서비스)와 컨테이너 환경이 보편화되면서, 장애가 어디서 시작됐는지 사람이 일일이 추적하기 어려워졌습니다.

2026-06-24 · 작성자: CNCF Korea Team · cncf.co.kr

목차

Datadog 대신 오픈소스 모니터링, SigNoz가 가능한가

1장. 오픈소스 옵저버빌리티의 등장과 SigNoz의 출발점

옵저버빌리티(observability, 관측 가능성)는 시스템 내부에서 무슨 일이 벌어지는지를 외부에서 수집한 데이터만으로 추론할 수 있는 능력을 뜻합니다. 마이크로서비스(microservice, 작은 단위로 쪼갠 독립 서비스)와 컨테이너 환경이 보편화되면서, 장애가 어디서 시작됐는지 사람이 일일이 추적하기 어려워졌습니다. 이 장에서는 옵저버빌리티 시장이 안고 있는 문제의식을 먼저 짚고, 그 문제에 대한 대답으로 등장한 SigNoz가 어떤 배경에서 만들어졌는지, 이름에 어떤 의미가 담겼는지를 살펴봅니다. 의사결정자가 도구 선택을 검토할 때 가장 먼저 확인해야 할 것은 "이 도구가 어떤 문제를 풀려고 태어났는가"이기 때문입니다.

1.1 옵저버빌리티 시장의 문제의식

옵저버빌리티 시장의 문제의식은 크게 두 갈래입니다. 하나는 상용 도구의 비용과 벤더 종속(vendor lock-in, 특정 공급사에 묶여 이탈이 어려워지는 상태)이고, 다른 하나는 그 대안으로 오픈소스를 골랐을 때 마주치는 스택 파편화입니다. 무엇을 다루는 절인지 먼저 말하면, 이 절은 상용 APM(application performance monitoring, 애플리케이션 성능 모니터링)의 과금 구조와 오픈소스 조합 운영의 현실을 나란히 놓고 비교합니다. 왜 필요한가 하면, 두 선택지의 한계를 동시에 이해해야 SigNoz 같은 통합형 오픈소스 도구가 왜 의미를 갖는지 판단할 수 있기 때문입니다. 이 검토는 도구 도입 후에 발생하는 예산 초과와 운영 인력 낭비라는 두 가지 비즈니스 문제를 미리 줄여 줍니다.

1.1.1 Datadog의 비용·락인 부담과 오픈소스 대안 수요

Datadog으로 대표되는 상용 SaaS APM은 도입이 쉽고 기능이 넓다는 장점이 분명합니다. 그러나 규모가 커질수록 비용이 기능에 비례하지 않고 더 가파르게 증가하는 구조라는 점이 의사결정자의 부담으로 작용합니다. 상용 APM의 과금은 대체로 모니터링 대상 호스트(host) 수를 기준으로 한 기본요금에, 사용자가 직접 정의하는 커스텀 메트릭(custom metric, 사용자 지정 지표)과 로그 수집량, 트레이스 보존 기간 같은 항목이 프리미엄으로 얹히는 방식입니다. 호스트가 늘고 지표가 세분화될수록 청구액이 규모에 비선형으로 늘어나기 쉽습니다. 데이터가 벤더의 폐쇄 포맷에 쌓이면, 나중에 다른 도구로 옮기려 해도 이미 축적된 이력과 설정 때문에 이탈 비용이 커지는 벤더 종속이 발생합니다[S03].

SigNoz가 처음부터 내세운 문제의식이 바로 이 지점입니다. 단일 애플리케이션 안에서 메트릭(metrics, 수치 지표), 트레이스(traces, 요청 추적), 로그(logs, 기록)를 한꺼번에 다루고, 셀프호스트(self-host, 자체 서버 운영)로 데이터를 직접 소유하여 비용과 락인을 함께 줄이자는 것이 핵심 주장입니다[S03]. 또한 SigNoz는 OpenTelemetry(OTel, 오픈텔레메트리)를 기본(native) 수집 표준으로 채택합니다. 반면 Datadog은 자사 에이전트를 우선하는 구조여서, 데이터 수집 경로

자체가 벤더에 맞춰지는 경향이 있습니다[S04]. 표준 기반 수집을 쓰면 도구를 바꿔도 계측 코드를 다시 짤 필요가 줄어듭니다.

아래 표는 상용 APM에서 청구액을 키우는 대표 과금 항목과, 규모가 커질 때 어떤 양상으로 비용이 늘어나는지를 정리한 것입니다. 구체적 단가는 플랜과 시점에 따라 달라지므로 게시 전 1차 출처 재확인이 필요한 항목으로 봅니다.

과금 항목	과금 기준	규모 확대 시 양상
호스트 모니터링	모니터링 대상 호스트 수	인프라 증설에 비례해 기본요금 증가
커스텀 메트릭	사용자 지정 지표 개수	지표 세분화 시 비선형으로 증가
로그 수집·보존	수집량(GB) + 보존 기간	트래픽·이벤트 증가에 따라 누적 증가
트레이스 보존	수집 span 수 + 보존 기간	요청량 증가에 정비례

정리하면, 상용 APM의 강점은 분명하지만 비용 곡선과 데이터 소유권 측면에서 부담이 쌓이고, 그 부담이 오픈소스 대안에 대한 수요로 이어졌습니다. SigNoz는 통합·셀프호스트·표준 수집이라는 세 축으로 이 수요에 응답하려고 만들어진 도구입니다[S03][S04].

1.1.2 파편화된 오픈소스 모니터링 스택의 한계

그렇다면 비용 부담을 피해 오픈소스만으로 모니터링을 꾸리면 문제가 해결될까요. 현실은 그렇게 단순하지 않습니다. 이 항에서 다루는 것은 시그널(signal, 관측 신호)별로 도구를 따로 골라 조합했을 때 생기는 스택 파편화입니다. 이 문제를 짚어야 하는 이유는, 오픈소스 자체는 무료여도 여러 도구를 붙여 운영하는 일에는 적지 않은 인력과 시간이 들기 때문입니다. 파편화를 줄이면 운영 인력의 분산과 장애 원인 추적 지연이라는 두 가지 문제가 함께 줄어듭니다.

오픈소스 모니터링은 전통적으로 시그널마다 대표 도구가 갈라져 있습니다. 메트릭은 Prometheus(프로메테우스)가 사실상 표준이고, 로그는 ELK 스택(Elasticsearch·Logstash·Kibana, 엘라스틱서치 기반 로그 분석 스택)이 자주 쓰이며, 트레이스는 Jaeger(예거)가 대표적입니다. Prometheus는 메트릭 수집과 질의에 특화된 도구이고, Elasticsearch 기반 ELK는 로그 검색과 분석에 중심을 둡니다. 두 도구는 다루는 데이터의 성격과 저장 방식이 서로 달라서, 메트릭과 로그를 한 화면에서 엮으려면 별도의 연동 작업이 필요합니다[S08].

시그널	대표 오픈소스 도구	주된 역할
메트릭(metrics)	Prometheus	수치 지표 수집·질의
로그(logs)	ELK(Elasticsearch 등)	로그 검색·분석
트레이스(traces)	Jaeger	분산 요청 추적

문제는 이 세 가지를 따로 설치하고, 따로 업그레이드하고, 따로 권한과 보존 정책을 관리해야 한다는 데 있습니다. 장애가 났을 때 운영자는 메트릭 대시보드에서 이상 징후를 본 뒤, 로그 도

구로 넘어가 원인을 찾고, 다시 트레이스 도구로 옮겨 요청 경로를 확인하는 식으로 화면을 오가야 합니다. 시그널 사이의 연결 고리가 끊겨 있으면 원인 추적이 늦어지고, 도구마다 다른 운영 지식이 필요해 담당 인력도 분산됩니다[S08]. 결국 "공짜 오픈소스"라는 표현 뒤에는 통합 운영의 숨은 비용이 자리합니다. SigNoz는 이 세 시그널을 하나의 애플리케이션으로 묶어 파편화를 줄이겠다는 발상에서 출발했고, 이 점이 다음 절에서 설명할 탄생 배경과 직접 이어집니다[S03].

1.2 SigNoz의 탄생과 이름의 의미

이 절은 SigNoz라는 도구가 누구의 손에서, 언제, 어떤 동기로 만들어졌는지를 다룹니다. 도구의 출신과 이름의 유래를 아는 일이 왜 중요한가 하면, 그것이 그 도구가 어떤 가치를 우선하는지를 압축해 보여 주기 때문입니다. 출발점을 이해하면 도입 검토 과정에서 마케팅 문구와 실제 설계 의도를 구분하기 쉬워지고, 그만큼 잘못된 기대로 인한 도입 후 실망을 줄일 수 있습니다.

1.2.1 "Signal vs Noise" — 이름의 유래와 창업자 배경

SigNoz라는 이름은 "Signal vs Noise(신호 대 잡음)"를 줄여 합친 것입니다[S01]. 신호와 잡음을 가르는 일은 옵저버빌리티의 본질과 맞닿아 있습니다. 수많은 모니터링 데이터 가운데 정말 의미 있는 신호를 가려내고, 무의미한 잡음을 걸러 내는 것이 모니터링 도구가 해야 할 일이기 때문입니다. 이름 자체가 도구의 목표를 그대로 드러내는 셈입니다.

이 작명은 창업자들의 배경과도 연결됩니다. SigNoz의 공동 창업자 Pranay Prateek과 Ankit Nayan은 전기공학(electrical engineering)을 전공했습니다[S01]. 전기·통신 공학에서 신호 대 잡음비(signal-to-noise ratio, 신호와 잡음의 비율)는 통신 품질을 좌우하는 핵심 개념입니다. 의미 있는 정보를 노이즈로부터 얼마나 또렷하게 분리해 내느냐가 시스템 성능을 결정한다는 사고방식이, 모니터링 도구의 이름과 설계 철학으로 옮겨 왔다고 볼 수 있습니다. 다만 창업자의 구체적 이력과 정확한 표기는 게시 전 1차 출처로 다시 확인할 항목으로 둡니다.

이름이 단순한 브랜딩을 넘어 의미를 갖는 이유는, 그것이 제품이 무엇을 우선하는지를 한 단어로 요약하기 때문입니다. SigNoz는 "데이터를 많이 모으는 것"이 아니라 "모은 데이터에서 신호를 뽑아내는 것"을 지향한다는 메시지를, 이름에서부터 내보이고 있습니다[S01].

1.2.2 누가·언제·왜 만들었나

SigNoz는 2021년에 공개됐고, 미국의 유명 스타트업 액셀러레이터(accelerator, 초기 기업 육성 기관)인 Y Combinator(와이 콤비네이터)의 지원을 받았습니다[S02]. 만든 사람은 공동 창업자 Pranay Prateek과 Ankit Nayan입니다. Pranay Prateek은 Microsoft에서 일한 경력이 있는 것으로 알려져 있습니다[S01]. 다만 창업 연도와 인명·이력의 정확한 세부는 확인이 필요한 항목으로 남겨 두고, 게시 전에 1차 출처로 재확인할 것을 권합니다.

이들이 SigNoz를 만든 동기는 분명합니다. 오픈소스 진영에 Datadog처럼 메트릭·트레이스·로그를 단일 도구로 통합해 주는 선택지가 없다는 문제의식에서 출발했습니다[S02]. 1.1절에서 짚었듯, 상용 도구는 비용과 락인 부담이 있고 오픈소스는 파편화 부담이 있었습니다. 두 부담을 동시에 풀려면 "오픈소스이면서도 통합된" 도구가 필요했고, 그 빈자리를 채우겠다는 것이 창업의 출발점이었습니다.

기술적 선택에서도 이 지향이 드러납니다. SigNoz는 초기에 Druid(드루이드)를 데이터 저장소로 검토했다가 ClickHouse(클릭하우스)로 전환했습니다[S02]. ClickHouse는 대량의 분석 질의에 강한 컬럼형 데이터베이스(columnar database, 열 단위 저장 데이터베이스)로, 메트릭·트레이스·로그처럼 양이 많고 빠른 집계가 필요한 데이터를 다루는 데 적합합니다. 통합과 성능을 함께 잡기 위한 실용적 결정이었던 셈입니다.

마지막으로 한 가지를 분명히 해 둘 필요가 있습니다. SigNoz는 OpenTelemetry를 네이티브로 채택해 CNCF(Cloud Native Computing Foundation, 클라우드 네이티브 컴퓨팅 재단) 생태계와 가깝게 맞닿아 있지만, SigNoz 자체가 CNCF가 호스팅하는 프로젝트는 아닙니다. CNCF 프로젝트는 OpenTelemetry 쪽이며, SigNoz는 그 표준 위에서 동작하는 독립 제품입니다. 이 구분을 흐리지 않는 것이 도입 검토에서 오해를 줄이는 데 도움이 됩니다.

2장. SigNoz 라이선스 구조와 오픈소스 Datadog 포지셔닝

SigNoz를 도입 후보로 검토할 때 의사결정자가 가장 먼저 확인해야 할 항목은 두 가지입니다. 하나는 "이 소프트웨어를 우리가 자유롭게 쓰고 고칠 수 있는가"라는 라이선스 문제이고, 다른 하나는 "정말 Datadog을 대체할 수 있는가"라는 포지셔닝 문제입니다. 두 질문은 따로 노는 것처럼 보이지만 실제로는 한 묶음입니다. 라이선스가 자유롭기 때문에 셀프호스트와 비용 절감이 가능하고, 셀프호스트가 가능하기 때문에 "오픈소스 Datadog"이라는 표현이 성립합니다. 이 장에서는 SigNoz의 오픈코어(open-core) 라이선스 구조를 먼저 풀어 설명한 뒤, 그 위에서 SigNoz가 어떤 근거로 Datadog의 대체재를 자처하는지 차근차근 짚겠습니다. 오픈코어란 소프트웨어의 핵심부는 자유 오픈소스 라이선스로 공개하고, 일부 부가 기능만 별도의 상용 라이선스로 묶는 사업 모델을 가리킵니다.

2.1 오픈코어 라이선스의 실체

이 절은 "SigNoz가 진짜 오픈소스인가"라는 질문에 사실 기반으로 답하기 위한 것입니다. 라이선스를 정확히 이해하지 못하면 도입 후 예상치 못한 사용 제약이나 법무 검토 지연을 겪을 수 있습니다. 여기서는 코드 저장소가 실제로 어떻게 라이선스되어 있는지, 그리고 그 구조가 자사의 사용 범위에 어떤 의미를 갖는지를 구분해 설명합니다. 이렇게 경계를 미리 확인해 두면 도입 검토 단계에서 법무·구매 부서와의 커뮤니케이션 비용을 줄일 수 있습니다.

2.1.1 코어 MIT와 ee 엔터프라이즈 분리

SigNoz의 코드 저장소는 하나의 라이선스로 통째로 묶여 있지 않고, 영역에 따라 두 가지 라이선스가 적용됩니다. 핵심 코드, 즉 코어(core)는 MIT 라이선스(정확히는 MIT Expat 형태)로 공개되어 있습니다 [S05]. MIT 라이선스는 가장 허용 범위가 넓은 자유 소프트웨어 라이선스 중 하나로, 저작권 고지만 유지하면 상업적 사용, 수정, 재배포, 사내 포크(fork)를 사실상 제약 없이 허용합니다. 의사결정자 관점에서 이는 코어 부분을 자사 인프라에 셀프호스트하고 필요에 맞게 코드를 고쳐 쓰더라도 추가 라이선스 비용이나 사용 승인 절차가 발생하지 않는다는 뜻입니다.

반면 저장소 안의 `ee` 폴더(enterprise edition)에 담긴 코드는 MIT가 아니라 SigNoz Enterprise 라이선스가 적용됩니다 [S05]. 이 영역은 상용 부가 기능에 해당하며, 자유롭게 가져다 쓰는 대상이 아닙니다. 즉 같은 저장소를 내려받더라도 어느 디렉터리의 코드를 쓰느냐에 따라 권리와

의무가 달라집니다. 도입을 검토하는 입장에서는 자사가 필요로 하는 기능이 코어에 있는지 `ee`에 있는지를 먼저 확인하는 일이 실무적으로 중요합니다.

두 영역의 경계를 표로 정리하면 다음과 같습니다.

구분	코어(core)	엔터프라이즈(ee 폴더)
적용 라이선스	MIT (Expat)	SigNoz Enterprise 라이선스
셀프호스트	자유	라이선스 조건에 따름
코드 수정·사내 포크	자유	제약 있음
상업적 사용	자유	라이선스 조건에 따름
비용	무료	별도

다만 한 가지 주의가 필요합니다. 위의 라이선스 표기(MIT Expat, `ee` 폴더 범위)는 SigNoz 측의 사업 방침과 저장소 구조 변경에 따라 달라질 수 있는 항목입니다. 따라서 실제 도입을 확정하기 전에는 해당 저장소의 LICENSE 파일과 각 디렉터리의 라이선스 헤더를 게시 시점 기준으로 직접 재확인하는 절차를 권장합니다. 이 부분은 "확인 필요" 항목으로 분류하는 편이 안전합니다.

2.1.2 "진짜 오픈소스냐" 논쟁과 해석

오픈코어 구조를 두고 SigNoz 커뮤니티 안에서는 "이게 진짜 오픈소스라고 부를 수 있느냐"는 논의가 이어져 왔습니다. 이 논쟁은 SigNoz를 깎아내리는 신호라기보다, 자유 소프트웨어를 진지하게 쓰는 사용자층이 라이선스 경계에 민감하다는 방증으로 읽는 편이 정확합니다. GitHub의 관련 Discussion(#4231)에서도 코어와 엔터프라이즈를 분리하는 방식이 오픈소스 정신에 부합하는지에 대한 의견 교환이 있었습니다 [S05]. 의사결정자는 이 논쟁의 존재 자체를 알아 두되, 그 무게를 과대평가하지 않는 균형이 필요합니다.

쟁점을 풀어 보면 핵심은 단어의 정의에 있습니다. 코어가 MIT로 공개되어 있으므로 "코어는 오픈소스가 맞다"는 점에는 이견이 적습니다. 논쟁이 생기는 지점은 "제품 전체를 오픈소스라고 부를 수 있는가"입니다. 일부 부가 기능이 상용 라이선스로 묶여 있는 만큼, 제품 전체를 가리켜 단정적으로 "100% 오픈소스"라고 말하기는 어렵습니다. 이 구분을 흐리지 않는 것이 정확한 커뮤니케이션입니다.

중요한 사실은 이런 오픈코어 모델이 SigNoz만의 예외적 선택이 아니라는 점입니다. 코어를 자유 라이선스로 공개해 채택을 늘리고, 운영 편의 기능이나 대규모 환경용 기능을 상용으로 분리해 수익을 내는 방식은 오늘날 인프라 소프트웨어 업계에서 표준에 가까운 수익화 전략입니다 [S03]. 지속 가능한 개발과 유지보수를 위해 어느 정도의 상용 영역을 두는 선택은, 오히려 프로젝트가 장기적으로 운영될 가능성을 높이는 요소로 볼 수 있습니다. 따라서 의사결정자에게 실질적으로 중요한 질문은 "이름이 순수 오픈소스냐"가 아니라 "우리가 실제로 쓰려는 기능이 자유롭게 쓸 수 있는 코어 영역에 있느냐"입니다. 이 질문에 답할 수 있다면 명칭 논쟁과 무관하게 도입 타당성을 판단할 수 있습니다.

2.2 "오픈소스 Datadog" 포지셔닝

이 절은 SigNoz가 스스로를 "오픈소스 Datadog 대안"으로 소개하는 근거가 무엇인지, 그리고 그 주장이 어디까지 타당한지를 검토합니다. 마케팅 문구를 그대로 받아들이면 도입 후 기대와 현실의 차이를 겪을 수 있으므로, 대체재라는 주장의 구체적 축과 그 이면의 운영 책임을 함께 살펴봅니다. 이렇게 양면을 같이 보면 기대치를 현실에 맞춰 잡을 수 있어 도입 후 재검토 비용을 줄일 수 있습니다.

2.2.1 대체재를 자처하는 근거

SigNoz가 Datadog의 오픈소스 대안을 표방하는 근거는 크게 세 축으로 정리됩니다 [S03]. 첫째는 통합입니다. 메트릭(metrics, 시계열 수치 지표), 트레이스(traces, 분산 요청 추적), 로그(logs)를 별도 도구로 나누지 않고 단일 애플리케이션 안에서 함께 다룹니다. 관측 데이터가 한곳에 모이면 장애를 추적할 때 도구 사이를 오갈 필요가 줄어듭니다. 둘째는 셀프호스트입니다. 자사 인프라에 직접 올려 운영할 수 있어 데이터가 외부 SaaS로 나가지 않습니다. 셋째는 비용 예측성입니다. SaaS형 관측 도구는 데이터 수집량이 늘수록 청구액이 함께 늘어 예산을 잡기 어려운 반면, 셀프호스트는 인프라 비용 중심이라 사용량 급증에 따른 청구 변동 위험이 작습니다.

이 세 축을 Datadog이 제공하는 가치와 나란히 매핑하면 다음과 같습니다.

Datadog의 가치	SigNoz의 대응
메트릭·트레이스·로그 단일 플랫폼	단일 애플리케이션에서 세 신호 통합 제공
매니지드 SaaS 운영 편의	셀프호스트로 데이터 외부 유출 없이 자사 운영
사용량 기반 과금 모델	인프라 비용 중심의 비용 예측성과 벤더 락인(lock-in) 회피

벤더 락인이란 특정 공급사의 제품에 데이터와 운영이 묶여 다른 도구로 옮기기 어려워지는 상태를 말합니다. SigNoz는 셀프호스트와 표준 기반 데이터 수집을 통해 이 종속을 줄이는 방향을 지향합니다 [S03]. 다만 위 매핑은 "기능 축이 대응된다"는 의미이지, 모든 세부 기능이 Datadog과 동등하다는 뜻은 아닙니다. 개별 기능의 성숙도 비교는 이후 장에서 별도로 다룹니다.

2.2.2 데이터 소유권과 셀프호스트의 가치

셀프호스트의 가장 분명한 장점은 데이터 소유권, 즉 데이터 주권(data sovereignty)입니다. SigNoz 커뮤니티 에디션(Community)은 셀프호스트 기준 비용이 $0이며, 관측 데이터가 자사 인프라 안에 머무릅니다 [S09]. 외부 SaaS로 로그와 트레이스를 내보내지 않으므로, 데이터 소재지 규제나 사내 보안 정책이 엄격한 조직에서 검토 부담이 줄어듭니다. 어떤 데이터를 얼마나 오래 보관할지도 자사가 직접 정할 수 있습니다. 이는 외부 사업자의 보관 정책과 과금 구조에 맞춰야 하는 SaaS 방식과 분명히 구분되는 지점입니다 [S03].

그러나 이 장점은 운영 책임과 한 묶음으로 따라옵니다. 셀프호스트는 라이선스 비용을 없애는 대신, 설치와 업그레이드, 용량 증설, 백업, 장애 대응 같은 운영 부담을 자사가 떠안는다는 것을 전제로 합니다 [S09]. SaaS에서는 공급사가 맡던 가용성과 확장성 관리를 자사 엔지니어가 책임지게 됩니다. 따라서 "비용 $0"이라는 표현은 라이선스 비용에 한정된 것이며, 운영 인력과 인프

라라는 보이지 않는 비용까지 0이라는 뜻이 아닙니다. 이 점을 분명히 구분해야 도입 후 총소유비용(TCO) 추정이 어긋나지 않습니다.

장점과 운영 책임을 대조하면 다음과 같이 정리됩니다.

셀프호스트의 장점	그에 따르는 운영 책임
데이터 주권 확보, 외부 유출 없음	설치·업그레이드·패치를 자사가 수행
라이선스 비용 $0(커뮤니티)	인프라·운영 인력 비용은 별도 발생
보관 기간·정책 자율 결정	용량 증설·백업·복구 책임 자사 부담
사용량 급증에도 비용 예측성 유지	가용성·확장성 관리를 직접 담당

결론적으로 셀프호스트는 통제권과 책임을 맞바꾸는 선택입니다. 자사가 데이터를 직접 통제하길 원하고 이를 감당할 운영 역량을 갖춘 조직이라면 SigNoz의 셀프호스트는 강력한 선택지가 됩니다. 반대로 운영 인력 여력이 부족하다면, 라이선스 비용 절감분이 운영 부담으로 상쇄될 수 있다는 점을 함께 따져 보아야 합니다. 이 판단의 기준선을 잡는 것이 다음 장에서 다룰 도입 적합성 검토의 출발점입니다.

3장. OpenTelemetry 네이티브 아키텍처와 단일 백엔드 통합

앞 장에서 SigNoz가 "오픈소스 Datadog"을 자처하는 근거를 살펴봤습니다. 이 장은 그 주장을 떠받치는 기술 구조를 풀어 봅니다. 핵심은 두 가지입니다. 첫째, 계측(instrumentation, 애플리케이션이 자기 상태를 신호로 내보내도록 코드에 심는 작업)은 OpenTelemetry라는 표준에 맡기고, 둘째, 그렇게 모인 신호는 단일 컬럼나(columnar, 데이터를 행이 아니라 열 단위로 저장하는 방식) 백엔드에 통합해 저장합니다. 이 두 결정이 합쳐지면 벤더 락인(vendor lock-in, 특정 공급사 제품에 묶여 교체 비용이 커지는 상태)이 줄고, 메트릭·트레이스·로그를 한 화면에서 함께 보는 통합 관측이 가능해집니다. 엔지니어에게는 데이터 흐름이, 의사결정자에게는 그 흐름이 비용과 이식성에 주는 영향이 이 장의 읽을거리입니다.

한 가지 구분을 먼저 명확히 하겠습니다. OpenTelemetry는 CNCF(Cloud Native Computing Foundation)가 호스팅하는 프로젝트이지만, SigNoz 자체는 CNCF 호스팅 프로젝트가 아닙니다. SigNoz는 OpenTelemetry라는 CNCF 표준을 기본으로 채택한 독립 제품입니다. 이 백서가 CNCF 중립 자료인 만큼, 표준과 제품의 경계를 흐리지 않고 서술합니다.

3.1 OpenTelemetry와 SigNoz의 관계

이 절은 OpenTelemetry와 SigNoz가 각각 무엇을 책임지는지, 왜 둘을 분리해서 이해해야 하는지를 다룹니다. 한 문장으로 요약하면 이렇습니다. OpenTelemetry는 벤더 중립 계측 표준이고, SigNoz는 그 표준이 내보낸 데이터를 저장·질의·시각화하는 백엔드입니다. 이 역할 분담을 분명히 해 두면, 뒤에 나올 "백엔드 교체 자유"라는 주장이 마케팅 구호가 아니라 구조에서 나온 결과임을 알 수 있습니다. 표준과 백엔드를 한 덩어리로 묶어 생각하면, 정작 락인이 어디서 풀리는지를 놓치게 됩니다.

3.1.1 OTel 네이티브 설계와 OTLP 직수집

이 항은 SigNoz가 OpenTelemetry를 "추가 지원" 항목이 아니라 설계 전제로 둔다는 점을 설명합니다. 둘의 차이는 데이터 호환성과 미래 이식성을 가르기 때문에 짚어 둘 가치가 있습니다.

먼저 용어부터 정리하겠습니다. OpenTelemetry(줄여서 OTel)는 메트릭(metrics)·트레이스(traces, 분산 추적 — distributed tracing, 하나의 요청이 여러 서비스를 거치는 경로를 잇는 기록)·로그(logs)를 표준 형식으로 만들어 내보내는 오픈 규격입니다. 이 데이터를 실어 나르는 전송 프로토콜이 OTLP(OpenTelemetry Protocol)입니다. SigNoz는 이 OTLP를 별도 변환 어댑터 없이 직접 수집합니다[S04]. 다시 말해 애플리케이션이나 OTel Collector가 OTLP로 보낸 신호가 그대로 SigNoz의 입구로 들어옵니다.

여기서 "OTel 네이티브"라는 표현의 무게가 드러납니다. 많은 상용 도구는 자사 에이전트(agent)가 받은 데이터를 1순위로 다루고, OTel 데이터는 변환을 거쳐 받아들입니다. SigNoz는 반대 순서입니다. OTLP와 OpenTelemetry 시맨틱(semantic, 신호에 붙는 속성 이름과 의미의 약속)을 기본으로 두고 설계했습니다[S04]. 그래서 속성 이름이 OTel 표준을 따르면 SigNoz 화면에서 별도 매핑 없이 의미 그대로 해석됩니다. 이 차이는 작아 보이지만, 데이터를 만든 쪽과 받는 쪽이 같은 표준을 기본으로 공유한다는 뜻이라 호환성 문제가 줄어듭니다.

의사결정자 관점에서 이 설계의 의미는 단순합니다. 계측을 OTel 표준으로 해 두면, 그 데이터를 받을 수 있는 백엔드가 SigNoz 하나로 제한되지 않습니다. 표준을 기본으로 받는 제품이 늘수록 선택지도 늘어나기 때문입니다. 계측에 들인 노력이 특정 제품에 종속되지 않는다는 점이 OTel 네이티브 설계의 실질적 이득입니다.

그림. OTel로 계측한 애플리케이션이 OTLP로 신호를 보내 SigNoz가 직접 수집하기까지의 흐름

3.1.2 계측-백엔드 분리로 벤더 락인 해소

이 항은 계측과 백엔드를 분리하면 왜 락인이 풀리는지를 설명합니다. 락인 해소는 추상적 이상이 아니라 장기 비용과 협상력에 직접 닿는 문제라서, 의사결정자가 가장 관심을 둘 대목입니다.

종속 구조의 핵심은 계측 방식에 있습니다. 상용 APM(Application Performance Monitoring, 애플리케이션 성능 관측 도구)은 대개 자사 전용 에이전트로 계측하도록 안내합니다. 이렇게 하면

코드 곳곳에 그 벤더의 라이브러리와 호출 규약이 스며듭니다. 나중에 백엔드를 바꾸려면 계측을 처음부터 다시 해야 합니다. 계측 자산이 백엔드에 묶여 있기 때문입니다.

OpenTelemetry는 이 묶임을 끊습니다. 애플리케이션을 OTel 표준으로 한 번 계측하면, 같은 데이터를 OTLP를 받는 어떤 백엔드로도 보낼 수 있습니다[S04]. 백엔드를 SigNoz에서 다른 도구로, 또는 그 반대로 바꿔도 계측 코드는 그대로 둡니다. 바뀌는 것은 데이터를 어디로 보낼지 가리키는 설정뿐입니다. 계측은 OpenTelemetry, 화면은 SigNoz라는 분리 구도가 여기서 나옵니다.

이 분리가 만드는 실질적 효과는 협상력입니다. 백엔드를 바꾸는 비용이 낮을수록, 현재 쓰는 공급사에 대한 의존도가 낮아집니다. 가격이 오르거나 조건이 나빠지면 옮길 수 있다는 사실 자체가 협상 카드가 됩니다. 다음 표는 락인 있는 계측과 OTel 표준 계측이 백엔드 교체 시점에 어떻게 갈리는지를 대비한 것입니다.

비교 항목	벤더 전용 에이전트 계측	OpenTelemetry 표준 계측
계측 코드의 소유	벤더 라이브러리에 결합	오픈 표준에 결합, 벤더 독립
백엔드 교체 시 작업	계측을 다시 수행	전송 대상 설정만 변경
데이터 형식	벤더 고유 형식	OTLP 표준 형식
이식성	낮음(재계측 비용 발생)	높음(코드 재사용)
협상력에 미치는 영향	약함(전환 비용이 장벽)	강함(전환 위협이 유효)

표가 보여 주듯, 락인 해소의 출처는 SigNoz라는 제품 자체가 아니라 OpenTelemetry라는 표준을 기본으로 받아들인 설계입니다[S04]. 이 점은 앞서 강조한 구분과도 이어집니다. 락인을 푸는 표준은 CNCF가 호스팅하는 OpenTelemetry이고, SigNoz는 그 표준을 충실히 따르는 백엔드 선택지 가운데 하나입니다.

3.2 단일 컬럼나 백엔드 통합

이 절은 SigNoz가 메트릭·트레이스·로그 세 신호를 어디에, 어떻게 저장하는지를 다룹니다. 여러 시그널을 따로 떨어진 도구에 나눠 담으면 상관분석이 어렵고 운영 동선이 늘어나는데, SigNoz는 이를 단일 저장소로 통합해 그 문제를 줄입니다. 저장 백엔드의 선택은 운영 난이도와 자원 비용을 함께 좌우하므로, 엔지니어와 의사결정자 모두에게 의미가 있는 주제입니다.

3.2.1 metrics·traces·logs 단일 ClickHouse 저장

이 항은 세 신호를 하나의 데이터스토어에서 처리한다는 SigNoz의 핵심 구조를 설명합니다. 통합 저장은 한 화면에서의 상관분석을 가능하게 해, 장애 대응 동선을 줄이는 효과로 이어집니다.

SigNoz는 메트릭·트레이스·로그를 모두 ClickHouse라는 단일 컬럼나 데이터베이스에 저장합니다[S06]. ClickHouse는 데이터를 열 단위로 저장하고 압축하는 분석용 데이터베이스로, 대량의 시계열·이벤트 데이터를 빠르게 집계하는 데 강합니다. 세 신호가 같은 저장소 안에 있으면, 예컨대 특정 트레이스에서 발견한 느린 구간을 같은 시각대의 메트릭·로그와 곧바로 연결해 볼 수 있습니다. 데이터를 옮기거나 도구를 바꿔 가며 맞춰 볼 필요가 없습니다.

이 구조가 왜 의미 있는지는 그 반대편을 보면 분명해집니다. 시그널마다 도구를 따로 붙인 스택에서는 메트릭은 한 도구에, 로그는 다른 도구에, 트레이스는 또 다른 도구에 흩어집니다. 도구가 셋이면 데이터 형식도 셋, 질의 언어도 셋, 운영 동선도 셋입니다. 이렇게 파편화된 스택은 각 도구가 제 역할은 잘하더라도, 신호를 가로질러 원인을 잇는 상관분석에서는 약합니다[S08]. 다음 표는 두 접근을 나란히 둔 것입니다.

비교 항목	파편화된 멀티 도구 스택	단일 컬럼나 백엔드(SigNoz)
저장 위치	신호마다 별도 저장소	메트릭·트레이스·로그 단일 ClickHouse
데이터 형식·질의	도구별로 상이	단일 백엔드로 일관
상관분석	도구 간 수동 대조 필요	한 화면에서 연결
운영 대상	여러 시스템 개별 운영	단일 백엔드 운영
장애 대응 동선	도구를 옮겨 다니며 추적	동일 화면에서 추적

핵심 메시지는 "세 도구 대신 하나의 컬럼나 백엔드"입니다. 다만 통합이 만능이라는 뜻은 아닙니다. 이미 Prometheus나 ELK를 깊이 운영 중인 조직이라면 그 자산과의 관계를 함께 따져야 하는데, 이는 5장에서 별도로 다룹니다. 이 항에서 짚을 점은, 단일 백엔드 구조가 상관분석과 운영 동선 단축이라는 구체적 이점을 설계 차원에서 제공한다는 사실입니다.

3.2.2 Druid에서 ClickHouse로의 전환 배경

이 항은 SigNoz가 초기 저장 백엔드였던 Apache Druid를 ClickHouse로 바꾼 배경을 설명합니다. 이 선택의 동기를 이해하면, 단일 노드 친화성과 자원 효율이 셀프호스트 운영에 어떤 의미인지 가늠할 수 있습니다.

SigNoz는 처음 Apache Druid를 저장 백엔드로 썼지만 이후 ClickHouse로 전환했습니다[S06]. 동기는 이상보다는 실리에 가깝습니다. Druid는 여러 종류의 노드 역할을 조합해야 제대로 도는 분산 시스템이라, 셀프호스트로 굴리려면 처음부터 신경 쓸 구성요소가 많습니다. 반면 ClickHouse는 단일 노드만으로도 의미 있게 동작하고, 필요해지면 클러스터로 확장하는 경로를 갖습니다. 노트북 한 대에서 시작해 점차 키우는 셀프호스트 시나리오와 잘 맞는 성질입니다.

자원 효율도 전환의 배경으로 언급됩니다. SigNoz는 시작 기준으로 ClickHouse가 Druid 대비 약 8.5배 적은 메모리로 동작했다고 밝힌 바 있습니다[S06]. 다만 이 수치는 측정 시점·구성·워크로드에 따라 달라질 수 있으므로 확인이 필요한 항목으로 둡니다. 백서가 단정할 성질의 절대값이 아니라, 전환을 정당화한 정성적 방향(가벼운 시작, 낮은 자원 요구)을 보여 주는 근거로 읽는 편이 정확합니다. 다음 표는 두 백엔드를 시작 시점의 운영 관점에서 대비한 것입니다.

비교 항목	Apache Druid(초기)	ClickHouse(현재)
구성 단위	여러 노드 역할 조합	단일 노드부터 동작
셀프호스트 시작 난이도	상대적으로 높음	상대적으로 낮음
시작 메모리 요구	더 큼	약 8.5배 적음(확인 필요)[S06]

비교 항목	Apache Druid(초기)	ClickHouse(현재)
확장 경로	분산 전제	단일 노드→클러스터 점진

이 전환이 의사결정에 주는 함의는 도입 장벽입니다. 저장 백엔드가 가벼운 시작을 허용할수록, PoC(Proof of Concept, 개념 검증)에 드는 초기 자원과 운영 부담이 낮아집니다. 단일 노드로 충분히 시작해 보고 가치를 확인한 뒤 확장한다는 경로가, 다음 절에서 다룰 배포 토폴로지의 출발점이 됩니다.

3.3 아키텍처 한눈에 보기

이 절은 앞에서 따로 살펴본 계측·전송·저장을 하나의 그림으로 잇고, 실제로 어떻게 배포하는지까지 정리합니다. 구성요소와 데이터 흐름을 알아야 운영 인력과 자원을 산정할 수 있고, 배포 규모의 단계를 알아야 도입의 첫발을 어디에 둘지 정할 수 있습니다. 엔지니어가 머릿속에 실제 배포 그림을 그릴 수 있게 하는 것이 이 절의 목표입니다.

3.3.1 핵심 컴포넌트와 데이터 흐름

이 항은 SigNoz를 이루는 주요 구성요소와, 데이터가 수집부터 화면까지 흐르는 경로를 설명합니다. 각 구성요소의 역할과 운영 책임을 알면 도입에 필요한 인력과 자원을 가늠할 수 있습니다.

데이터 흐름은 크게 세 단계입니다. 첫째는 수집입니다. OTel로 계측된 애플리케이션이 OTLP로 신호를 보내면, OTel Collector가 이를 받아 가공·라우팅합니다. Collector는 수집 지점에서 데이터를 모으고 형식을 맞춰 전달하는 중계 구성요소입니다. 둘째는 저장입니다. 모인 메트릭·트레이스·로그는 앞 절에서 본 ClickHouse에 통합 저장됩니다[S06]. 셋째는 질의와 시각화입니다. SigNoz UI가 ClickHouse에 질의해 대시보드·트레이스 뷰·알림으로 보여 줍니다. 다음 표는 각 구성요소의 역할과 셀프호스트 시 운영 책임 주체를 정리한 것입니다.

구성요소	역할	셀프호스트 운영 책임
OTel SDK(애플리케이션 측)	표준 형식으로 신호 생성	개발팀(계측 코드 관리)
OTel Collector	신호 수집·가공·라우팅	운영팀(수집 파이프라인)
ClickHouse	메트릭·트레이스·로그 통합 저장	운영팀(저장소 용량·백업)
SigNoz UI·질의 계층	대시보드·트레이스·알림 제공	운영팀(서비스 가용성)

표가 보여 주듯, 셀프호스트에서는 계측 코드부터 저장소 운영까지의 책임이 모두 자사로 넘어옵니다. 이는 데이터를 자사 인프라에 두는 데이터 소유권의 대가이기도 합니다. 구성요소별 책임을 미리 그려 두면, 도입에 필요한 운영 역량을 과소평가하지 않을 수 있습니다.

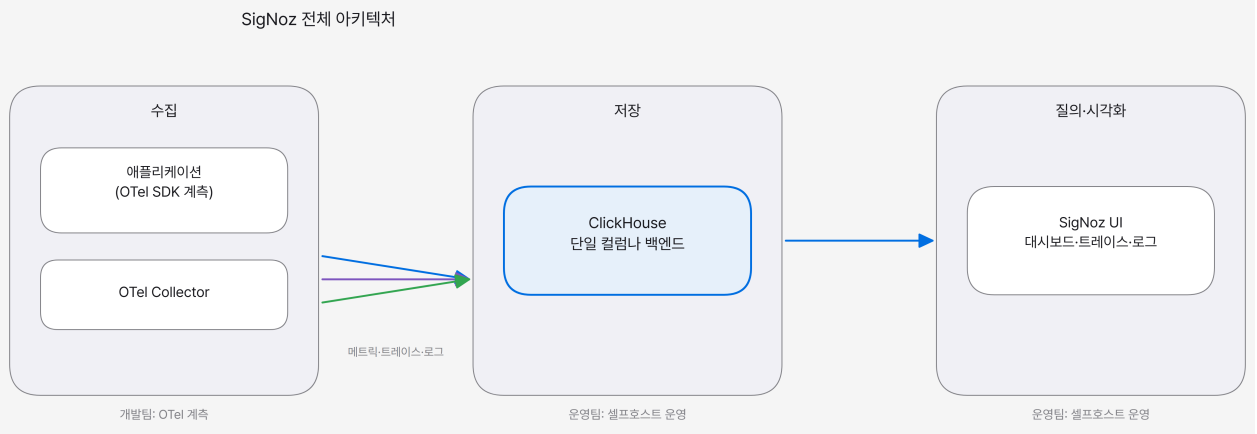

그림. 수집(OTel Collector)에서 저장(ClickHouse)을 거쳐 질의·시각화(SigNoz UI)로 이어지는 전체 아키텍처

3.3.2 셀프호스트 배포 토폴로지

이 항은 SigNoz를 어디서부터 어떻게 배포해 나가는지, 시작 규모와 확장 경로를 설명합니다. 시작 규모와 확장 경로가 초기 도입 장벽을 결정하므로, PoC를 계획하는 단계에서 미리 알아 둘 가치가 있습니다.

배포는 점진적입니다. 가장 작은 형태는 단일 노드 배포입니다. 앞서 본 대로 ClickHouse가 단일 노드부터 동작하기 때문에, 노트북이나 작은 서버 한 대에 모든 구성요소를 올려 시범 운영할 수 있습니다[S06]. 이 단계는 적합성을 검증하는 PoC에 적합합니다. 트래픽과 데이터가 늘면 다음 단계로 넘어갑니다. 수집 계층이나 저장 계층을 별도 노드로 분리하고, 이후 ClickHouse를 클러스터로 확장해 가용성과 처리량을 늘립니다. 다음 표는 배포 규모를 단계별로 정리한 것입니다.

배포 단계	구성	적합한 용도
단일 노드	모든 구성요소를 한 노드에	PoC·소규모 검증
분리 구성	수집·저장 계층 노드 분리	초기 운영·중간 규모
클러스터	ClickHouse 다중 노드 확장	운영 규모·고가용성

이 점진 경로의 핵심은 "노트북에서 시작해 클러스터로"라는 한 문장으로 요약됩니다. 처음부터 큰 분산 시스템을 세울 필요 없이, 작게 시작해 필요한 만큼만 키우면 됩니다. 이 성질은 도입 초기의 리스크와 자원 투입을 낮춰 주며, 7장에서 다룰 1~2주 PoC 설계의 전제가 됩니다. 다만 단계가 올라갈수록 운영 복잡도와 책임도 함께 커진다는 점은 분명히 해 두어야 합니다. 셀프호스트의 가벼운 시작은 진실이지만, 운영 규모에서의 부담까지 가벼운 것은 아닙니다.

4장. SigNoz가 Datadog을 대체할 수 있는 기능·비용 근거

앞선 장에서 SigNoz의 구조와 도입 시나리오를 살펴봤다면, 이 장은 의사결정자가 가장 먼저 묻는 두 가지 질문에 답합니다. 첫째, 기능 면에서 Datadog에 견줘 어디가 더 낫거나 비슷한가. 둘째, 비용이 실제로 줄어드는가, 줄어든다면 어떤 구조 때문인가. 결론을 미리 말하면, SigNoz의

우위는 화려한 신기능이 아니라 표준 준수와 과금 구조의 단순함에서 나옵니다. 이 두 축이 장기적으로 비용 예측성과 종속 위험을 함께 낮춥니다.

본격적인 비교에 앞서 한 가지 전제를 분명히 해 둡니다. 이 장에 나오는 모든 단가와 플랜 금액은 두 회사 모두 수시로 바뀝니다. 따라서 실제 견적이나 의사결정에 쓰기 전에는 게시 직전에 공식 가격 페이지에서 반드시 재확인하시기 바랍니다. 본문은 금액의 절대값보다 "무엇을 기준으로 돈을 매기는가"라는 과금 축에 초점을 둡니다.

4.1 Datadog 대비 기능적 우위

이 절은 SigNoz가 Datadog 대비 구조적으로 유리한 두 지점을 다룹니다. 하나는 관측 데이터를 수집하는 표준인 OpenTelemetry(오픈텔레메트리, 벤더 중립 관측 데이터 수집 표준. 이하 OTel)를 다루는 방식의 차이이고, 다른 하나는 그 표준을 통해 들어온 메트릭에 어떻게 과금하느냐의 차이입니다. 두 지점 모두 단순한 기능 항목이 아니라, 장기적으로 종속과 비용을 좌우하는 설계 선택입니다. 표준을 기본으로 지원하느냐 부가로 지원하느냐가 곧 데이터의 이동성과 청구서를 결정하기 때문입니다.

4.1.1 OpenTelemetry 기본 지원의 차이

OpenTelemetry는 메트릭·로그·트레이스를 특정 벤더에 묶이지 않는 방식으로 수집하고 내보내기 위한 업계 표준입니다. 이 표준을 도구가 얼마나 우선순위 높게 지원하느냐가 중요한 이유는 분명합니다. 표준을 기본으로 지원하면 계측 코드를 한 번만 작성해 두고 백엔드는 나중에 갈아끼울 수 있지만, 부가로 지원하면 결국 그 벤더의 전용 에이전트와 포맷에 다시 묶이기 때문입니다.

Datadog은 자사 에이전트(Datadog Agent)를 우선합니다. 가장 풍부한 기능과 정밀한 연동은 자사 에이전트 경로에서 제공되고, OTel은 그다음 등급의 호환 경로로 취급됩니다[S04]. 반면 SigNoz는 처음부터 OTel을 기본 수집 경로로 설계했습니다. 별도의 독자 에이전트로 사용자를 끌어들이는 구조가 아니라, 표준 OTel Collector(콜렉터, OTel 데이터를 받아 가공·전달하는 수집 구성요소)를 그대로 받아들이는 구조입니다.

이 차이는 실무에서 종속 위험으로 나타납니다. 자사 에이전트에 깊이 의존할수록 계측 코드가 벤더 전용 형태로 누적되고, 나중에 백엔드를 바꾸려면 계측을 다시 손봐야 하는 비용이 생깁니다. OTel을 기본으로 지원하면 같은 계측을 유지한 채 백엔드만 교체할 수 있어, 도구 전환의 문턱이 낮아집니다.

비교 항목	Datadog	SigNoz
기본 수집 경로	자사 에이전트 우선	OTel Collector 기본
OTel 지원 수준	부가(호환 경로) [S04]	기본(표준 경로)
계측 코드 이동성	벤더 전용 누적 시 낮음	표준 유지로 높음
백엔드 교체 비용	계측 재작업 발생 가능	계측 유지, 교체 용이

정리하면, OTel 기본 지원은 화면에 보이는 기능 차이라기보다 데이터 소유권과 이동성의 문제입니다. 표준을 우선하는 설계는 지금 당장의 편의보다 몇 년 뒤의 선택지를 넓혀 줍니다. 의사결정자 관점에서는 이것이 곧 협상력과 직결됩니다.

4.1.2 custom metric 과금 함정 회피

이 항목은 OTel 지원 방식의 차이가 어떻게 곧바로 청구서로 이어지는지를 보여 줍니다. 핵심 개념은 커스텀 메트릭(custom metric, 도구가 기본 제공하지 않는 사용자 정의 지표)입니다. 많은 모니터링 도구가 이 커스텀 메트릭을 별도의 프리미엄 과금 대상으로 분류하는데, 분류 기준이 명확하지 않으면 예상치 못한 비용이 누적됩니다. 이 절은 그 분류 차이가 비용 예측성을 어떻게 흔드는지 설명합니다.

문제는 Datadog이 OTel로 보낸 메트릭을 상당수 "custom metric"으로 분류해 프리미엄 과금 대상에 넣는다는 점입니다[S04]. 즉 표준 경로(OTel)로 데이터를 보낼수록 자사 에이전트 경로로 보낼 때보다 비싼 과금 구간으로 들어갈 수 있습니다. 표준을 따랐다는 이유로 비용이 늘 수 있는 셈이라, 사용자는 결국 자사 에이전트 쪽으로 다시 끌려가게 됩니다.

SigNoz는 메트릭을 출처나 정의 방식으로 차등하지 않고 모두 동일하게 처리합니다[S04]. 어떤 메트릭이 "커스텀"이라서 더 비싸지는 구분 자체가 없고, 과금은 수집한 샘플 수에 따릅니다. 공개된 단가는 메트릭 기준 100만 샘플당 약 0.1달러 수준이지만, 이 값은 변동성이 크므로 게시 직전 재확인이 필요합니다[S09].

과금 분류 관점	Datadog	SigNoz
OTel 메트릭 취급	custom으로 분류해 프리미엄 과금 [S04]	일반 메트릭과 동일 처리 [S04]
custom/일반 구분	있음(요금 차등)	없음
과금 기준	분류 등급에 따라 달라짐	수집 샘플 수 일원화 [S09]
비용 예측성	분류 규칙 의존, 변동 큼	수집량만 알면 추정 가능

이 차이가 실무에 주는 함의는 단순합니다. Datadog에서는 "이 메트릭이 custom으로 잡히는가"를 늘 신경 써야 하고, 그 판단이 청구서를 크게 흔들 수 있습니다. SigNoz에서는 그 변수가 사라지므로, 비용 추정이 "얼마나 많은 데이터를 수집하는가"라는 한 가지 축으로 단순해집니다.

4.2 비용 모델 비교

이 절은 두 도구의 과금 구조 자체를 비교합니다. 앞 절이 메트릭 분류라는 좁은 지점을 다뤘다면, 여기서는 과금의 뼈대 전체를 봅니다. 모니터링 비용이 호스트 수에 묶이는지, 사용자 좌석 수에 묶이는지, 아니면 순수 수집량에 묶이는지에 따라 조직이 성장할 때 비용 곡선이 완전히 달라지기 때문입니다. 이 절은 그 곡선의 모양과, 라이선스가 0이어도 남는 운영비까지 함께 다룹니다.

4.2.1 사용량 기반 과금과 무제한 사용자

이 항목은 SigNoz 과금의 가장 큰 특징인 사용량 기반 모델을 설명합니다. 사용량 기반 과금이란 수집한 데이터의 양을 기준으로만 비용을 매기는 방식입니다. 여기서 중요한 점은 무엇으로 과금하지 않는가입니다. SigNoz는 모니터링 대상 호스트 수로도, 대시보드에 접근하는 사용자 좌석 수로도 과금하지 않고, 사용자는 무제한입니다[S09]. 이 구조가 왜 비용 통제에 유리한지 짚어 보겠습니다.

좌석 과금이 없다는 것은 관측 데이터를 보는 사람이 늘어도 비용이 늘지 않는다는 뜻입니다. 개발자뿐 아니라 운영·기획·경영진까지 대시보드를 열어도 추가 요금이 붙지 않으므로, 조직 안에서 관측 문화를 넓히기에 부담이 적습니다. 반대로 좌석 과금 모델에서는 보는 사람을 늘릴수록 비용이 늘어, 무의식적으로 접근 권한을 좁히게 됩니다.

SigNoz의 실제 과금은 데이터 종류별 수집량에 붙습니다. 로그와 트레이스는 1GB당 약 0.30달러, 메트릭은 100만 샘플당 약 0.1달러 수준으로 공개돼 있습니다[S09]. 클라우드 유료 플랜의 경우 팀(Teams) 기본 요금이 월 49달러이고, 그 안에 49달러어치 사용량이 포함됩니다. 이 포함량은 대략 로그·트레이스 163GB 또는 메트릭 4.9억 샘플에 해당합니다[S11]. 다만 이 수치들은 모두 변동성이 크므로 게시 직전 재확인이 필요합니다.

과금 축	호스트 수	사용자 좌석	수집 데이터량
SigNoz 과금 여부	과금 안 함 [S09]	과금 안 함, 무제한 사용자 [S09]	과금 대상(종류별 단가) [S09]
조직 확장 시 영향	서버 늘려도 좌석·호스트 비용 없음	보는 사람 늘려도 비용 동일	수집량에 비례해 선형 증가
통제 지점	해당 없음	해당 없음	보존 기간·샘플링으로 조정

요약하면, SigNoz의 과금은 "얼마나 많은 사람이, 얼마나 많은 서버를"이 아니라 "얼마나 많은 데이터를"이라는 한 축으로 모입니다. 그래서 비용을 줄이려는 노력이 좌석 줄이기나 호스트 줄이기 같은 우회로가 아니라, 데이터 보존 기간과 샘플링이라는 본질적 조정으로 향합니다. 이는 관측 문화를 좁히지 않으면서 비용을 다루는 방법이 됩니다.

4.2.2 비용 예측성과 TCO 관점

마지막 항목은 라이선스 금액 너머의 총소유비용(TCO, total cost of ownership) 관점을 다룹니다. TCO는 도구를 쓰는 데 들어가는 모든 비용을 합산한 개념으로, 구독료뿐 아니라 운영 인력과 인프라까지 포함합니다. 이 관점이 필요한 이유는 명확합니다. SigNoz 커뮤니티 에디션은 셀프호스트(self-hosted, 자사 인프라에 직접 설치·운영) 시 라이선스가 0달러이지만[S09], 그렇다고 총비용이 0이 되는 것은 아니기 때문입니다. 라이선스가 사라진 자리에는 자사가 부담하는 인프라·운영비가 남습니다.

비용 예측성 측면에서 SigNoz의 사용량 기반 구조는 분명한 장점이 있습니다. 앞 절에서 본 대로 과금 변수가 수집량 한 축으로 단순하므로, 데이터 증가 추세만 추정하면 비용 곡선을 비교적 정확히 그릴 수 있습니다. custom 메트릭 분류처럼 청구서를 흔드는 숨은 변수가 적다는 점

도 예측성을 높입니다[S04]. 다만 셀프호스트를 택할 경우, 예측해야 할 대상이 구독료에서 인프라·운영비로 옮겨갈 뿐 사라지지는 않습니다.

따라서 의사결정 단계에서는 라이선스 금액만 비교하지 말고, 아래 구성요소를 함께 점검해 실제 TCO를 추정해야 합니다.

- 인프라 비용: 데이터 저장소와 처리 노드를 위한 서버·스토리지·네트워크 자원
- 운영 인력 비용: 설치·업그레이드·장애 대응·용량 관리에 드는 엔지니어 시간
- 데이터 보존 비용: 보존 기간이 길수록 늘어나는 스토리지와 처리 부담
- 가용성 확보 비용: 이중화·백업·모니터링의 모니터링에 드는 추가 자원
- 마이그레이션·학습 비용: 초기 전환과 팀 숙련에 드는 일회성 비용

셀프호스트 SigNoz는 라이선스를 0으로 만드는 대신 이 항목들을 자사가 떠안습니다. 반대로 SigNoz Cloud나 Datadog 같은 SaaS는 이 운영 부담을 벤더에게 넘기는 대신 구독료로 지불하는 구조입니다. 어느 쪽이 유리한지는 조직의 운영 역량과 데이터 규모에 달려 있으므로, 정답을 단정하기보다 위 체크리스트로 양쪽을 같은 기준에서 비교하는 편이 안전합니다. 그렇게 보면 SigNoz의 진짜 가치는 "공짜"가 아니라, 과금 구조가 단순하고 표준을 우선해 비용과 종속을 함께 다룰 수 있다는 데 있습니다[S09].

5장. 기존 모니터링 도구와 SigNoz의 경계 정리

모니터링 도구를 고를 때 가장 흔한 오해는 "이 도구 하나면 다 된다"는 기대입니다. 실제로는 도구마다 잘하는 영역이 갈립니다. 메트릭(metric, 시간에 따라 변하는 수치 지표)에 집중하는 도구가 있고, 로그(log, 시스템이 남기는 사건 기록) 검색에 강한 도구가 있으며, 전통적인 서버 상태 감시에 특화된 도구가 있습니다. SigNoz는 이 가운데 어디에 서 있는지, 그리고 어떤 환경에서 기존 도구를 대체하거나 보완하는지를 이 장에서 정리합니다. 비교의 목적은 우열을 가리는 것이 아니라, 운영하는 시스템의 성격에 맞는 선택을 돕는 것입니다.

5.1 Prometheus·Elasticsearch와의 차이

이 절은 오픈소스 모니터링 진영에서 가장 널리 쓰이는 두 도구인 Prometheus와 Elasticsearch를 SigNoz와 나란히 두고 봅니다. 두 도구는 각자 한 가지 신호(signal, 관측 데이터의 종류 — 메트릭·트레이스·로그)에 최적화되어 있어서, 전체 그림을 보려면 여러 도구를 엮어야 한다는 공통점이 있습니다. SigNoz가 무엇을 통합하려 하는지를 이해하려면 먼저 이 분업 구조를 짚어야 합니다. 이 절은 도구를 잘못 골라 나중에 신호 하나가 비는 상황을 줄이는 데 목적이 있습니다.

5.1.1 메트릭 전용·로그 중심 도구의 경계

Prometheus는 메트릭 수집과 저장에 특화된 오픈소스 도구입니다. 대상 시스템이 노출한 수치를 주기적으로 가져오는 풀(pull, 모니터링 서버가 대상에게 데이터를 끌어오는 방식) 방식으로 동작하며, CPU 사용률이나 요청 처리량 같은 시계열 수치를 다루는 데 강점이 있습니다 [S08]. 다만 설계 자체가 메트릭 중심이라, 로그 본문을 검색하거나 분산된 요청의 경로를 따라가는 작업은 Prometheus 단독으로 하기 어렵습니다.

Elasticsearch는 반대편에 있습니다. ELK 스택(Elasticsearch·Logstash·Kibana, 로그 수집·저장·시각화를 묶은 조합)은 대량의 로그를 모아 전문 검색(full-text search, 텍스트 본문 전체를 대상으로 한 검색)으로 빠르게 찾아내는 데 초점이 있습니다 [S08]. 장애가 났을 때 특정 오류 메시지를 키워드로 뒤지는 작업에는 잘 맞지만, 시계열 메트릭을 효율적으로 다루는 용도로 설계된 도구는 아닙니다.

SigNoz는 메트릭·트레이스·로그 세 신호를 하나의 제품 안에서 함께 다루는 것을 지향합니다 [S08]. 트레이스(trace, 하나의 요청이 여러 서비스를 거치는 경로를 이어 붙인 기록)까지 기본으로 포함한다는 점이 두 도구와 갈리는 지점입니다. 결과적으로 Prometheus는 메트릭, ELK는 로그, SigNoz는 세 신호 통합이라는 식으로 본령이 나뉩니다. 다음 표는 각 도구가 어느 신호를 일차적으로 책임지는지를 정리한 것입니다.

Prometheus · ELK · SigNoz 신호 커버리지

평가 축	Prometheus	ELK 스택	SigNoz
메트릭 수집	강점 (풀 시계열)	보조 (Metricbeat)	지원 (OTel 메트릭)
로그 검색	없음	강점 (전문 검색)	지원 (통합 로그)
분산 트레이스	없음	별도 (APM 모듈)	기본 포함
통합 저장소	신호별 분리	신호별 분리	단일 백엔드
주된 설계 목적	메트릭·경보	로그 분석	통합 관측

그림. Prometheus·ELK·SigNoz의 신호 커버리지 비교

도구 하나가 모든 신호를 똑같이 잘 다루지는 않습니다. Prometheus와 ELK를 함께 운영하는 조합이 오랫동안 표준처럼 쓰인 이유도 여기에 있습니다. SigNoz는 이 분업을 한 제품으로 합치려는 시도이며, 그래서 비교의 핵심은 "기능이 더 많은가"보다 "신호를 어떻게 엮는가"에 있습니다.

5.1.2 통합 백엔드가 주는 상관분석 이점

여러 신호를 따로 저장하면 장애를 추적할 때 도구 사이를 오가야 합니다. 메트릭 그래프에서 이상한 지점을 발견하고, 같은 시각의 로그를 다른 화면에서 찾고, 다시 트레이스를 또 다른 도구에서 뒤지는 식입니다. 이 절은 그 화면 전환 비용을 줄이는 통합 백엔드(backend, 데이터를 실제로 저장·질의하는 뒤단)가 왜 중요한지를 설명합니다.

SigNoz는 ClickHouse라는 컬럼형(columnar, 데이터를 행이 아니라 열 단위로 저장해 집계·분석 질의에 유리한 방식) 데이터베이스 하나에 세 신호를 함께 적재합니다 [S06]. 메트릭·트레이스·로

그가 같은 저장소에 들어가 있으면, 한 화면에서 시각과 식별자를 공유하며 신호 사이를 바로 넘나들 수 있습니다 [S06]. 예를 들어 응답 지연이 튄 메트릭 지점을 클릭해 그 시각의 느린 트레이스를 열고, 그 트레이스에 연결된 로그를 같은 자리에서 펼치는 식의 추적 동선이 가능합니다.

이런 상관분석(correlation, 서로 다른 신호를 시각·요청 단위로 연결해 원인을 좁히는 분석)의 이점은 신호가 흩어져 있을 때 더 두드러집니다. Prometheus와 ELK를 조합하면 각 도구의 강점은 살릴 수 있지만, 신호를 잇는 작업은 운영자의 손과 머릿속에서 일어납니다 [S08]. 시각을 맞추고, 같은 요청인지 가늠하고, 도구 사이 맥락을 이어 붙이는 수고가 장애 대응 시간을 늘립니다. 단일 백엔드는 이 수고의 상당 부분을 데이터 모델 차원에서 줄여 줍니다 [S06].

물론 통합 백엔드가 만능은 아닙니다. 세 신호를 한 저장소에 모으면 그 저장소의 용량과 성능이 전체 모니터링의 병목이 될 수 있고, ClickHouse 운영 지식도 따로 필요합니다. 그래서 통합의 가치는 "도구 수를 줄였다"가 아니라 "신호 사이를 잇는 비용을 줄였다"는 쪽에서 평가하는 편이 정확합니다. 운영 중인 시스템에서 장애가 여러 신호에 걸쳐 나타나는 일이 잦다면 통합 백엔드의 이점이 크고, 신호 하나만 집중적으로 보는 환경이라면 전용 도구의 단순함이 더 나을 수 있습니다.

5.2 Zabbix와의 차이

이 절은 SigNoz를 전통적인 인프라 모니터링 도구인 Zabbix와 비교합니다. 둘은 같은 "모니터링"이라는 이름을 쓰지만 겨냥하는 시스템의 세대가 다릅니다. Zabbix가 고정된 서버와 네트워크 장비를 오래 감시해 온 도구라면, SigNoz는 수시로 생겼다 사라지는 클라우드 네이티브(cloud-native, 컨테이너·오케스트레이션을 전제로 설계된 동적 환경) 시스템을 겨냥합니다. 이 차이를 분명히 하면 "어느 쪽이 더 좋은가"가 아니라 "어떤 환경에 무엇이 맞는가"를 가릴 수 있고, 도구를 환경에 잘못 끼워 맞추는 실수를 줄일 수 있습니다.

5.2.1 전통 인프라 모니터링 vs 클라우드 네이티브

Zabbix는 서버-에이전트(server-agent, 중앙 서버가 각 호스트에 설치된 에이전트로부터 상태를 수집하는 구조) 방식의 전통적인 인프라 모니터링 도구입니다. 호스트 목록이 비교적 고정된 환경에서 서버와 네트워크 장비의 상태를 촘촘히 감시하는 데 강점이 있습니다 [S07]. IP와 호스트명이 잘 바뀌지 않는 데이터센터형 인프라에서는 이 정적 모델이 오히려 안정적이고 예측 가능합니다.

SigNoz는 분산 추적(distributed tracing, 여러 서비스에 걸친 요청 경로를 추적하는 기능)을 중심에 둔 클라우드 네이티브 지향 도구입니다 [S07]. 감시 대상이 개별 호스트라기보다, 여러 서비스를 거치는 요청의 흐름입니다. 그래서 무엇을 일차적으로 보느냐가 두 도구의 출발선부터 다릅니다. Zabbix는 "이 서버가 살아 있는가"를, SigNoz는 "이 요청이 어디서 느려졌는가"를 먼저 묻습니다.

이 차이는 세대 차이로 보는 편이 공정합니다. Zabbix가 약한 도구라서가 아니라, 정적 호스트를 전제로 정교하게 다듬어진 도구이기 때문에 동적 환경에서는 결이 어긋날 뿐입니다 [S07]. 아래 표는 두 도구가 각각 어떤 환경을 전제로 설계되었는지를 정리합니다.

Zabbix vs SigNoz — 적합 환경

평가 축	Zabbix	SigNoz
수집 구조	서버-에이전트 폴링	OTel 텔레메트리
대상 안정성 전제	정적 호스트	동적·단명 대상
주 감시 단위	호스트·장비	서비스·요청
네트워크 장비 감시	강점 (SNMP)	주력 아님
분산 요청 추적	비주력	중심 기능

그림. Zabbix와 SigNoz의 적합 환경 비교

요약하면 Zabbix는 고정 인프라의 상태 감시, SigNoz는 동적 서비스의 요청 추적이라는 식으로 본령이 갈립니다. 한 조직 안에서도 레거시 인프라는 Zabbix로, 신규 마이크로서비스는 SigNoz로 나눠 보는 조합이 현실적인 선택이 되기도 합니다.

5.2.2 동적 마이크로서비스 환경 적합성

컨테이너 기반 환경에서는 인스턴스가 분 단위로 생겼다 사라집니다. 이 절은 그런 동적 환경에서 두 도구의 적합도가 왜 갈리는지를 봅니다. 핵심은 "감시 대상의 목록이 고정인가, 유동적인가"라는 전제 차이이며, 이 전제를 잘못 잡으면 모니터링이 실제 시스템을 따라가지 못하는 문제가 생깁니다.

Zabbix의 서버-에이전트 모델은 감시할 호스트를 비교적 명시적으로 등록하는 흐름을 전제로 다듬어져 있습니다 [S07]. 호스트가 거의 바뀌지 않는 환경에서는 이 명시성이 곧 안정성입니다. 그러나 쿠버네티스(Kubernetes, 컨테이너를 자동으로 배치·확장·교체하는 오케스트레이션 플랫폼)처럼 파드(pod, 컨테이너를 묶어 배포하는 최소 단위)가 수시로 교체되는 환경에서는, 대상이 끊임없이 바뀌는 만큼 정적 등록 모델이 따라잡아야 할 변화의 폭이 커집니다.

SigNoz는 동적 마이크로서비스(microservices, 기능을 작은 서비스로 잘게 나눈 아키텍처) 환경을 처음부터 겨냥했습니다 [S07]. 개별 인스턴스가 짧게 살다 사라져도, 추적의 단위가 호스트가 아니라 요청과 서비스이기 때문에 인스턴스 교체에 덜 흔들립니다. 어느 인스턴스에서 처리됐는지보다, 요청이 어떤 서비스 경로를 거쳐 어디서 지연됐는지가 추적의 중심입니다.

환경별 적합도 — Zabbix vs SigNoz

환경 유형	Zabbix	SigNoz
정적 베어메탈·VM	높음 (정적 강점)	보통 (메트릭만)
가상화 고정 인프라	높음	보통
컨테이너 동적 환경	보통 (추적 부담)	높음 (동적 설계)
마이크로서비스 분산 요청	낮음	높음 (분산 추적)

그림. 정적·동적 환경별 적합도 매트릭스

적합도는 환경의 성격으로 결정됩니다. 정적이고 안정적인 인프라에는 Zabbix의 성숙함이 유리하고, 컨테이너처럼 변화가 잦은 환경에는 SigNoz의 동적 전제가 유리합니다 [S07]. 한쪽이 다른 쪽을 전면 대체한다기보다, 인프라 세대에 맞춰 도구를 고르는 문제로 보는 것이 맞습니다.

5.3 AppDynamics·Instana 등 상용 APM과의 차별성

이 절은 SigNoz를 AppDynamics나 Instana 같은 상용 APM(Application Performance Monitoring, 애플리케이션 성능 모니터링)과 비교합니다. 앞 절들이 오픈소스 도구 사이의 분업을 다뤘다면, 여기서는 오픈 표준과 독점 제품이라는 결의 차이를 봅니다. 상용 APM은 오랜 기간 다듬어진 풍부한 기능을 갖추고 있지만, 그 기능이 특정 벤더의 에이전트와 플랫폼에 묶여 있는 경우가 많습니다. 이 절은 그 종속(lock-in, 특정 벤더 제품에 묶여 교체 비용이 커지는 상태)의 성격을 짚어, 도입 후에 갈아타기 어려워지는 상황을 미리 가늠하도록 돕습니다.

5.3.1 독점 에이전트 종속과 개방 표준

상용 APM 제품들은 대체로 벤더 고유의 독점 에이전트(proprietary agent, 특정 벤더만 제공하는 비공개 계측 소프트웨어)를 애플리케이션에 붙여 데이터를 수집합니다 [S04]. AppDynamics나 Instana 같은 제품이 이 방식에 해당합니다. 이 에이전트가 수집한 데이터는 해당 벤더의 백엔드와 짝을 이루도록 설계되어 있어서, 계측을 한 번 깔아 두면 데이터의 형식과 흐름이 그 제품에 맞춰집니다.

문제는 다른 제품으로 옮기려 할 때 드러납니다. 계측이 벤더 고유 방식이면, 도구를 바꾸는 순간 애플리케이션의 계측을 상당 부분 다시 손봐야 합니다 [S04]. 이 재계측 비용이 사실상 교체를 막는 종속으로 작동합니다. 기능이 마음에 들지 않아도 갈아타기가 부담스러워 머무는 상황이 생기는 것입니다.

SigNoz는 OpenTelemetry(OTel, 텔레메트리 데이터 수집을 위한 벤더 중립 오픈 표준) 표준 계측을 받아들이고, 셀프호스트(self-host, 벤더의 클라우드가 아니라 자체 인프라에 직접 설치·운영하는 방식)로 운영하는 방식을 택했습니다 [S04]. OTel로 계측해 두면 그 계측 자체는 특정 벤더에 묶이지 않으므로, 백엔드를 SigNoz에서 다른 OTel 호환 도구로 바꾸더라도 애플리케이션 계측은 대체로 그대로 살릴 수 있습니다. 데이터의 소유와 보관 위치를 직접 통제할 수 있다는 점도 셀프호스트의 특성입니다.

상용 APM vs SigNoz — 개방성·종속성

평가 축	상용 APM (AppDynamics·Instana)	SigNoz
계측 방식	독점 에이전트	OTel 표준
데이터 형식 개방성	벤더 포맷	개방 표준
백엔드 교체 용이성	재계측 비용 큼	이전 용이
데이터 보관 위치	벤더 SaaS	셀프호스트
벤더 종속 정도	높음	낮음

상용 APM의 OTel 데이터 수용 지원은 제품·버전마다 다름 — 도입 시점 확인 필요

그림. 상용 APM과 SigNoz의 개방성·종속성 비교

여기서 한 가지 단서를 달아 둡니다. 최근 상용 APM 제품들도 OTel 데이터를 받아들이는 방향으로 움직이고 있어, 개방성의 격차가 줄어드는 추세입니다. 다만 어느 제품이 OTel을 어느 수준까지 지원하는지는 제품과 버전에 따라 다르고 자주 바뀌므로, 도입 검토 시점에 직접 확인이 필요합니다. SigNoz 역시 CNCF가 호스팅하는 프로젝트는 아니라는 점은 짚어 둘 만합니다 — OTel 표준을 따르는 것과 CNCF 프로젝트인 것은 별개입니다.

5.3.2 기능 동등성과 도입 장벽 비교

도구를 바꿀 때 결정은 "개방적인가"만으로 내려지지 않습니다. 실무에서는 "지금 쓰는 기능을 그대로 쓸 수 있는가"와 "직접 운영할 부담을 감당할 수 있는가"가 함께 저울에 오릅니다. 이 절은 상용 APM의 엔터프라이즈 기능과 SigNoz의 대응 수준을 견주고, 셀프호스트가 더하는 운영 부담을 함께 봅니다. 목적은 개방성의 이점만 보고 결정했다가 운영 단계에서 예상 못한 부담에 부딪히는 일을 줄이는 것입니다.

상용 APM은 오랜 기간 다듬어진 풍부한 기능을 갖추고 있습니다. 자동 이상 탐지, 정교한 대시보드, 세분화된 알림 정책, 벤더가 책임지는 기술 지원과 운영 대행이 대표적입니다. 이 기능들은 벤더의 SaaS(Software as a Service, 벤더가 클라우드에서 운영해 주는 구독형 소프트웨어) 형태로 제공되므로, 도입한 쪽은 인프라 운영 부담 없이 기능만 쓰면 됩니다.

SigNoz는 메트릭·트레이스·로그 통합, 대시보드, 알림 같은 핵심 기능으로 이에 대응하며, OTel 표준 위에서 동작한다는 개방성을 더합니다 [S04]. 다만 셀프호스트라는 선택은 그만큼의 운영 책임을 함께 가져옵니다. ClickHouse를 비롯한 구성 요소를 직접 설치·확장·백업·업그레이드해야 하고, 데이터가 늘수록 용량과 성능을 직접 관리해야 합니다 [S06]. 벤더 SaaS가 대신 져 주던 인프라 부담이 운영팀으로 넘어오는 셈입니다.

따라서 비교의 축은 "기능이 동등한가" 하나가 아니라, 기능 동등성과 운영 부담, 그리고 개방성이라는 세 축의 균형입니다. 데이터 통제권과 표준 기반의 이동성이 중요하고 셀프호스트 운영 역량을 갖춘 조직이라면 SigNoz의 이점이 크게 다가옵니다. 반대로 운영 인력을 최소화하고 벤더의 지원과 성숙한 기능을 그대로 누리고 싶은 조직이라면 상용 APM의 SaaS 모델이 더 맞을 수 있습니다.

그림. 상용 APM과 SigNoz의 기능 동등성·도입 장벽 종합 매트릭스

정리하면, SigNoz는 기존 도구를 한 줄로 대체하는 제품이 아니라 신호 통합과 개방 표준이라는 두 축에서 차별화되는 선택지입니다. Prometheus·ELK가 나눠 맡던 신호를 한 백엔드로 모으고, Zabbix가 전제하던 정적 인프라 대신 동적 마이크로서비스를 겨냥하며, 상용 APM의 독점 에이전트 대신 OTel 표준과 셀프호스트를 택합니다. 이 차이가 우리 환경에 이점이 되는지는, 시스템의 성격과 운영 역량을 함께 놓고 판단할 문제입니다.

6장. 에디션 선택과 단계적 채택 — 0원에서 시작하는 길

SigNoz는 한 가지 제품이 아니라 여러 에디션(edition, 같은 제품의 기능·지원 범위가 다른 판본)으로 제공됩니다. 무료로 직접 운영하는 Community부터, 월정액으로 운영을 맡기는 Cloud, 대규모 조직을 위한 Enterprise까지 폭이 넓습니다. 이 장에서는 각 에디션이 어디까지 해 주고 무엇을 사용자에게 남기는지, 그리고 어떤 순서로 도입하면 비용 위험을 줄일 수 있는지를 살펴봅니다. 의사결정자가 도입을 검토할 때 가장 흔히 빠지는 함정이 "무료냐 유료냐"를 한 번에 정하려는 것이기 때문입니다. SigNoz의 진짜 강점은 0원으로 검증을 시작한 뒤, 거버넌스(governance, 접근 통제·감사·책임 체계)가 필요해지는 시점에만 비용을 지불하는 단계적 채택이 가능하다는 데 있습니다[S09].

6.1 에디션별 기능 경계

이 절은 SigNoz의 세 에디션이 기능과 책임을 어떻게 나누는지를 정리합니다. 같은 화면, 같은 질의 언어를 쓰더라도 SSO(single sign-on, 단일 로그인)나 RBAC(role-based access control, 역할 기반 접근 제어) 같은 기능은 에디션에 따라 들어 있기도 하고 빠지기도 합니다. 이 경계를 먼저 분명히 해야 하는 이유는, 무료 Community로 충분한지 아니면 처음부터 상용이 필요한지를 조직의 보안·규정 요건에 비추어 판단할 수 있기 때문입니다. 경계를 미리 파악해 두면 도입 후에 "필요한 기능이 무료판에 없더라"라며 다시 예산을 짜는 일을 줄일 수 있습니다[S10].

6.1.1 SSO·RBAC·운영 책임의 차이

에디션을 가르는 가장 실질적인 선이 바로 접근 통제와 운영 책임입니다. Community 에디션은 SigNoz의 핵심 기능, 곧 메트릭·트레이스·로그 통합과 대시보드, 알림을 모두 무료로 제공합니다. 다만 로그인은 Google OAuth(Google Workspace 계정 연동) 수준에 머무릅니다[S10]. 즉 구글 계정으로 묶어 로그인하게 할 수는 있지만, 사내 표준 인증 체계와 본격적으로 통합하는 단계까지는 가지 못합니다.

기업용 인증과 권한 관리는 상용 에디션의 영역입니다. SAML 2.0(보안 인증 정보 교환 표준)과 OIDC(OpenID Connect, OAuth 기반 인증 프로토콜)를 쓰는 SSO, 그리고 사용자별·팀별로 권한을 잘게 나누는 세분화 RBAC는 Cloud와 Enterprise에서만 제공됩니다[S10]. 이 차이는 단순한 편의 기능의 문제가 아닙니다. 규정 준수(compliance) 감사를 받는 조직에서는 "누가 어떤 데이터에 접근할 수 있는가"를 역할 단위로 통제하고 기록할 수 있어야 하므로, RBAC의 유무가 도입 가능 여부 자체를 가르기도 합니다.

나머지 한 축은 운영 책임의 소재입니다. Community는 무료인 대신 설치·업그레이드·백업·장애 대응을 모두 자사가 떠안는 셀프호스트(self-host, 자체 서버 운영) 방식입니다. 반대로 Cloud는 SigNoz가 인프라와 운영을 대신 맡고, Enterprise는 여기에 지원 SLA(service level agreement,

서비스 수준 합의)와 매니지드 운영을 더합니다. Enterprise는 형태도 한 가지가 아니어서, SigNoz가 운영하는 매니지드 Cloud, 자사 클라우드 계정 안에서 돌리는 BYOC(bring your own cloud, 자체 클라우드 반입), 그리고 지원 계약을 곁들인 셀프호스트 중에서 고를 수 있습니다[S10]. 데이터를 자사 경계 안에 두어야 하는 규제 환경이라면 BYOC가, 운영 부담을 최대한 덜고 싶다면 매니지드 Cloud가 맞는 식입니다.

아래 표는 라이선스부터 보존까지 일곱 항목으로 세 에디션의 경계를 한눈에 비교한 것입니다. 금액과 보존 옵션은 플랜 개편에 따라 자주 바뀌므로, 게시 직전에 1차 출처로 다시 확인할 항목으로 둡니다.

항목	Community	Cloud (Teams)	Enterprise
라이선스·비용	무료($0), 셀프호스트	$49/월부터(사용량 일부 포함)	$4,000/월부터(커스텀)
SSO 인증	Google OAuth 수준	SAML 2.0·OIDC SSO	SAML 2.0·OIDC SSO
RBAC	기본 수준(세분화 미제공)	세분화 RBAC	세분화 RBAC
운영 책임	자사(설치·운영·백업 전담)	SigNoz 매니지드	매니지드 / BYOC / 지원 셀프호스트
사용자 수	제한 없음(자체 운영)	무제한 사용자	커스텀
과금 방식	없음(인프라 비용만 자사 부담)	월정액 + 사용량	커스텀 계약
지원·SLA	커뮤니티 채널	표준 지원	지원 SLA·우선 대응

정리하면, Community와 상용 에디션을 가르는 본질은 기능의 많고 적음이 아니라 거버넌스와 운영 책임입니다[S10]. 핵심 관측 기능 자체는 무료판에서도 충분히 쓸 수 있고, 돈을 지불해 얻는 것은 주로 기업용 인증·권한·운영 대행입니다. 이 구분을 알고 나면 다음 절에서 다룰 보존·SLA 차이도 같은 맥락으로 읽힙니다.

6.1.2 데이터 보존과 지원 SLA

기능 경계의 또 다른 축은 데이터를 얼마나 오래 보관하느냐, 그리고 문제가 생겼을 때 누가 책임지고 대응하느냐입니다. 이 항에서 다루는 것은 보존 기간(retention, 수집한 데이터를 저장해 두는 기간)과 지원 SLA의 에디션별 차이입니다. 이 둘을 따로 짚는 이유는, 장기 보존과 보장된 지원이 규제 대응이나 사후 분석에서 결정적인데도 무료판에서는 자사가 직접 떠안아야 하는 영역이기 때문입니다. 미리 짚어 두면 "감사 대비로 1년치 로그가 필요한데 그 비용을 계산에 안 넣었다"는 식의 누락을 줄일 수 있습니다[S09].

Cloud 에디션의 보존 정책은 시그널마다 선택지가 다릅니다. 로그와 트레이스는 15일·30일·90일·180일·365일 중에서, 메트릭은 1개월·3개월·6개월·13개월 중에서 고르는 식입니다[S09]. 보존 기간을 길게 잡을수록 저장 비용이 올라가므로, 규제 요건과 사후 분석 필요에 맞춰 시그널별로

다르게 설정하는 것이 합리적입니다. 예를 들어 감사 추적이 필요한 로그는 길게, 단기 성능 추세만 보면 되는 메트릭은 짧게 두는 식의 조정이 가능합니다.

Community 셀프호스트에서는 이 보존을 자사가 직접 설계해야 합니다. SigNoz는 데이터를 ClickHouse(클릭하우스, 대량 분석에 강한 컬럼형 데이터베이스)에 저장하므로, 보존 기간은 곧 스토리지 용량과 직결됩니다. 무료라고 해서 보존이 공짜인 것은 아니며, 길게 보관할수록 디스크와 운영 부담이 자사 몫으로 쌓입니다. 반면 Cloud에서는 보존 기간 선택이 곧 요금에 반영되므로, 자사가 용량을 계산할 필요 없이 정책만 고르면 됩니다.

지원 SLA는 더 분명하게 갈립니다. Community는 공식 문서와 커뮤니티 채널에 의존하며, 장애가 나면 대응의 속도와 책임이 전적으로 자사에 있습니다. 매니지드 운영과 보장된 응답 시간을 갖춘 지원 SLA는 상용 에디션, 특히 Enterprise의 영역입니다[S10]. 아래 표는 보존과 지원을 에디션별로 견준 것입니다. 보존 옵션과 SLA 세부는 변동이 잦으므로 게시 직전 재확인이 필요합니다.

구분	Community(셀프호스트)	Cloud(Teams)	Enterprise
보존 기간	자사가 ClickHouse 용량으로 직접 설계	로그·트레이스 15~365일, 메트릭 1~13개월 선택	커스텀(요건에 맞춰 협의)
보존 비용	자체 스토리지·운영 부담	선택 기간이 요금에 반영	계약에 포함
장애 대응	자사 책임, 커뮤니티 참고	표준 지원	지원 SLA·우선 대응

요컨대 보존과 SLA는 "무료가 곧 0원"이라는 인식의 빈틈을 보여 줍니다[S09]. Community의 보존과 대응은 비용이 없는 게 아니라 자사의 스토리지와 인력으로 치르는 비용입니다. 그래서 장기 보존 의무나 보장된 지원이 요건이 되는 순간, 상용 전환을 검토할 근거가 생깁니다. 그 판단의 출발점이 다음 절에서 다룰 단계적 채택입니다.

6.2 단계적 채택 경로

이 절은 SigNoz를 한 번에 결정하지 않고 단계를 밟아 도입하는 방법을 제시합니다. 먼저 무료 Community로 검증한 뒤, 거버넌스 요건이 실제로 나타나는 시점에만 상용으로 넘어가는 경로입니다. 이렇게 나눠 접근해야 하는 이유는, 검증 없이 상용 계약부터 맺으면 쓰지도 않을 기능에 비용을 미리 묶어 두게 되기 때문입니다. 단계를 나누면 도입 초기의 예산 과다 집행과, 반대로 무료판 한계에 막혀 다시 검토하는 재작업을 함께 줄일 수 있습니다[S09][S11].

6.2.1 Community에서 시작하는 PoC

단계적 채택의 첫 단추는 비용 없이 검증하는 PoC(proof of concept, 개념 검증)입니다. 이 항에서 다루는 것은 무료 Community 에디션으로 SigNoz가 자사 환경에 맞는지를 먼저 확인하는 방법입니다. PoC를 앞세우는 이유는, SigNoz의 통합 관측·대시보드·알림 같은 핵심 기능이

Community에 모두 들어 있어 라이선스 비용 0원으로 실제 데이터를 흘려 보며 검증할 수 있기 때문입니다[S09]. 이렇게 하면 도구가 기대만큼 작동하는지, 운영 부담은 감당할 만한지를 돈을 쓰기 전에 가늠할 수 있습니다.

PoC 단계에서 점검해야 할 항목은 분명합니다. 자사의 애플리케이션을 OpenTelemetry(OTel, 오픈텔레메트리)로 계측해 메트릭·트레이스·로그가 한 화면에 제대로 모이는지, 평소 보던 대시보드를 SigNoz에서 재현할 수 있는지, 알림이 의도한 조건에서 울리는지를 확인합니다. 동시에 셀프호스트 운영의 현실, 곧 설치 난이도와 ClickHouse 저장 용량 증가, 업그레이드 절차도 함께 체감해 둡니다. 무료라는 이유로 운영 부담을 검증에서 빼면, 나중에 그 부담이 숨은 비용으로 드러나기 때문입니다.

이 단계에서 일부러 확인하지 않아도 되는 것도 있습니다. SSO와 세분화 RBAC, 지원 SLA는 Community에 없으므로 PoC에서는 검증 대상에서 빼고, 대신 "이 기능들이 우리에게 정말 필요한가"라는 질문만 남겨 둡니다[S10]. 그 답이 "그렇다"로 바뀌는 순간이 곧 다음 항에서 다룰 상용 전환 트리거입니다. 정리하면 Community PoC의 목적은 두 가지입니다. 핵심 기능이 자사 요건을 충족하는지 확인하는 것, 그리고 셀프호스트 운영 부담의 크기를 미리 재 보는 것입니다.

6.2.2 상용 전환 트리거 식별

단계적 채택의 마지막은 언제 비용을 지불할지를 판단하는 일입니다. 이 항에서 다루는 것은 무료 Community에서 상용 Cloud나 Enterprise로 넘어가야 할 신호, 곧 전환 트리거(trigger, 행동을 촉발하는 기준 조건)를 식별하는 방법입니다. 트리거를 미리 정해 두는 이유는, "어쩐지 불편해서"가 아니라 "이 요건이 충족돼서" 전환하도록 기준을 명확히 해 두면 과소·과잉 투자를 모두 피할 수 있기 때문입니다[S10]. 기준이 분명하면 전환 시점을 두고 조직 내에서 소모적으로 논쟁할 일도 줄어듭니다.

대표적인 전환 트리거는 거버넌스 요건이 실제로 발생하는 순간입니다. 사내 표준 인증과 통합해야 해서 SAML·OIDC 기반 SSO가 필요해질 때, 데이터 접근을 역할 단위로 통제해야 해서 세분화 RBAC가 요구될 때, 장애 대응의 책임과 응답 시간을 계약으로 보장받아야 할 때가 그렇습니다[S10]. 여기에 운영 부담 자체가 트리거가 되기도 합니다. 셀프호스트 운영에 들어가는 인력이 절약하려던 라이선스 비용을 넘어서면, 운영을 위탁하는 매니지드 Cloud가 오히려 합리적인 선택이 됩니다.

비용을 따질 때는 Cloud의 과금 구조를 함께 봐야 합니다. Cloud의 Teams 플랜은 월 $49가 기본이고, 이 금액 안에 $49어치의 사용량이 포함되며 사용자 수에는 제한이 없습니다[S11]. 사용자를 늘려도 인당 요금이 붙지 않는다는 점은, 팀이 커지는 조직에서 셀프호스트와 비교할 때 중요한 변수입니다. 다만 $49는 시작점이고 사용량이 포함분을 넘으면 추가 과금이 발생하므로, 자사의 데이터량을 PoC에서 미리 측정해 두는 것이 정확한 비교의 전제입니다. 금액과 포함 사용량은 변동이 잦으니 게시 직전에 1차 출처로 재확인할 항목으로 둡니다.

아래는 전환 여부를 판단할 때 점검할 트리거를 정리한 표입니다. 하나라도 "그렇다"가 분명하면, 그 항목이 상용 전환의 근거가 됩니다.

전환 트리거	점검 질문	충족 시 권장 방향
SSO 통합	사내 표준 인증(SAML·OIDC)과 연동해야 하는가	Cloud 또는 Enterprise
세분화 RBAC	역할 단위 접근 통제·감사가 규정 요건인가	Cloud 또는 Enterprise
지원 SLA	보장된 응답 시간·책임 대응이 필요한가	Enterprise
운영 부담	셀프호스트 운영 인건비가 라이선스 비용을 넘는가	매니지드 Cloud
데이터 경계	자사 클라우드 안에 데이터를 두어야 하는가	Enterprise(BYOC)

결국 SigNoz의 단계적 채택은 한 문장으로 요약됩니다. 0원짜리 Community로 검증을 끝낸 뒤, SSO·RBAC·SLA 같은 거버넌스가 실제로 필요해질 때만 그에 맞는 상용 에디션으로 넘어가는 것입니다[S09][S11]. 이 경로를 따르면 의사결정자는 검증 전에 비용을 묶지 않고, 동시에 무료판의 한계에 부딪혀 도입이 멈추는 일도 피할 수 있습니다. 무엇을 무료로 얻고 무엇에 비용을 치를지를 조직의 요건으로 직접 통제한다는 점이, SigNoz를 단계적으로 채택할 때 얻는 가장 큰 이점입니다.

7장. SigNoz PoC 착수 판단을 위한 의사결정 가이드

앞선 장에서 SigNoz의 구조와 라이선스, 비용 모델을 살펴봤습니다. 이 장은 그 내용을 실제 결정으로 바꾸는 자리입니다. 목표는 단 하나입니다. 독자가 자기 환경의 숫자를 대입해서 "우리도 1~2주 PoC를 돌려볼 가치가 있는가"를 근거 갖고 판단하도록 돕는 것입니다. PoC(Proof of Concept)는 본격 도입에 앞서 작은 범위로 가능성을 검증하는 시험 운영을 말합니다. 막연한 기대나 막연한 불안이 아니라, 체크리스트와 워크시트로 따져보고 결정하시길 권합니다.

7.1 의사결정 체크리스트

이 절은 "지금 시작해도 되는가"를 자가 진단하는 도구입니다. 의사결정자가 흔히 겪는 문제는 두 가지입니다. 하나는 우리 환경이 SigNoz와 맞는지 감으로만 판단하는 것이고, 다른 하나는 현행 Datadog 비용과 셀프호스트 비용을 사과 대 오렌지로 비교하는 것입니다. 아래 두 항목은 각각 적합성과 비용을 구조화된 질문으로 바꿔서 이런 오판을 줄입니다. 체크 결과가 좋게 나오면 PoC로 넘어가고, 그렇지 않으면 왜 아직 이른지 설명할 근거가 손에 남습니다.

7.1.1 도입 적합성 판단 기준

먼저 우리 조직이 SigNoz가 잘 맞는 환경인지부터 확인합니다. 이 진단이 필요한 이유는 분명합니다. 통합 관측(unified observability)이 필요 없는 단순한 환경이라면 굳이 새 도구로 갈아탈 동기가 약하고, 반대로 분산 시스템이 복잡한데 운영 인력이 없으면 셀프호스트가 부담이 됩니다. 통합 관측은 로그·메트릭·트레이스를 한 곳에서 연결해 보는 방식을 뜻합니다. SigNoz는

OpenTelemetry(이하 OTel, 관측 데이터 수집의 벤더 중립 표준)를 1차 수집 경로로 삼기 때문에, 이미 OTel로 계측했거나 계측할 의지가 있는 팀에 특히 잘 맞습니다 [S03].

아래 자가 진단에서 "예"가 많을수록 PoC 가치가 높다고 보시면 됩니다. 정답을 강요하는 표가 아니라, 우리 상황을 솔직히 비추는 거울로 쓰시길 권합니다.

진단 질문	예	아니오
마이크로서비스나 분산 시스템 비중이 높아 분산 트레이싱이 필요한가	적합성 높음	메트릭 중심이면 효용 제한적
이미 OTel로 계측했거나 표준 계측으로 옮길 의지가 있는가	마찰 적음	에이전트 재작업 부담
사용량 증가에 따라 SaaS 청구액이 부담스러운가	셀프호스트 동기 강함	전환 동기 약함
데이터를 자체 인프라에 두려는 거버넌스·규제 요건이 있는가	셀프호스트 가치 큼	SaaS로 충분할 수 있음
컨테이너·ClickHouse를 운영할 최소 인력(1명 이상)이 있는가	운영 가능	클라우드 옵션 우선 검토

여기서 주의할 점이 있습니다. SSO(Single Sign-On, 통합 인증)와 RBAC(Role-Based Access Control, 역할 기반 접근 제어) 같은 기능 일부는 커뮤니티 에디션이 아니라 상용 전용으로 제공됩니다 [S10]. 따라서 보안·거버넌스 요건이 까다로운 조직이라면, 적합성 진단에서 "예"가 나오더라도 이 부분을 별도로 확인해야 합니다. 완화 방안은 단순합니다. PoC 단계에서는 커뮤니티 에디션으로 핵심 가치를 검증하고, SSO·RBAC가 꼭 필요한지는 본 도입 시점에 상용 또는 클라우드 옵션과 함께 따로 판단하면 됩니다. 적합성과 거버넌스를 분리해서 보면, 한쪽 약점 때문에 전체 가능성을 성급히 접는 일을 줄일 수 있습니다.

7.1.2 현행 Datadog 비용 대비 추정

적합성을 확인했다면 다음은 돈입니다. 이 항목이 필요한 이유는, 비용 비교가 자칫 SaaS 청구서와 소프트웨어 라이선스비만 견주는 반쪽짜리 계산이 되기 쉽기 때문입니다. 정직한 비교는 셀프호스트에 따르는 인프라비와 운영 인력 시간까지 포함해야 합니다. 그래야 "정말 우리에게 이득인가"라는 질문에 흔들리지 않는 답을 낼 수 있습니다.

비교의 출발점은 현행 청구서입니다. Datadog은 호스트 수, 수집·보존하는 로그량, 커스텀 메트릭 수처럼 사용량 단위로 과금합니다 [S09]. 그래서 먼저 지난 몇 달 청구서에서 우리가 실제로 무엇에 얼마를 쓰는지 항목별로 분해해야 합니다. 그 사용량을 SigNoz로 옮기면 어떻게 바뀌는지를 환산합니다. 커뮤니티 에디션 자체의 소프트웨어 비용은 0에서 시작하지만 [S09], 셀프호스트에는 서버·스토리지 같은 인프라비가 생깁니다. 운영을 줄이고 싶다면 클라우드 옵션이 대안이며, 공개된 시작 가격대는 호스트당 월 단위로 책정돼 있어 청구서 환산 시 참조점이 됩니다 [S11].

아래 워크시트에 우리 숫자를 채워 넣으시면 됩니다. 이 표의 목적은 정확한 금액을 단정하는 것이 아니라, 빠지기 쉬운 비용 항목을 빠짐없이 테이블에 올리는 것입니다.

비용 항목	현행(Datadog)	SigNoz 셀프호스트	SigNoz 클라우드
소프트웨어·구독	사용량 기반 청구액	커뮤니티 에디션 0에서 시작 [S09]	호스트당 월 과금 [S11]
인프라(서버·스토리지)	포함됨	별도 발생(자체 부담)	제공사 부담
운영 인력 시간	거의 없음	설치·업그레이드·장애 대응 시간	축소됨
데이터 보존 확장	보존 기간별 추가 과금	ClickHouse 용량 증설로 흡수	플랜에 따름

운영 인력 시간을 비용으로 환산하는 단계를 건너뛰지 마시길 권합니다. 셀프호스트의 소프트웨어비가 0이라도, 누군가 ClickHouse를 운영하고 업그레이드를 챙기는 시간은 실제 비용입니다. 다만 이 부담은 처음부터 단일 노드로 시작해 규모에 맞춰 키우면 초기에는 작게 유지할 수 있습니다 [S06]. 절대 금액을 못 박기보다, 위 네 항목을 모두 채운 뒤 세 열을 나란히 놓고 비교하는 편이 의사결정에는 더 정직합니다.

7.2 PoC 로드맵과 다음 행동

이 절은 판단을 행동으로 바꾸는 부분입니다. 체크리스트와 워크시트에서 긍정적인 신호를 얻었다면, 이제 남은 일은 작게 시작해 직접 확인하는 것입니다. 여기서 흔한 실패는 두 가지입니다. 범위를 너무 크게 잡아 PoC가 끝나지 않거나, 성공 기준을 정하지 않아 결과를 두고 의견만 갈리는 경우입니다. 아래 두 항목은 PoC를 1~2주 안에 끝나도록 설계하고, 시작 전에 리스크를 눈에 보이게 만들어 이런 함정을 피하게 합니다.

7.2.1 1~2주 PoC 설계

PoC의 목적은 완벽한 도입이 아니라 빠른 검증입니다. 이 단계가 필요한 이유는, 우리 환경의 실제 데이터로 돌려보기 전까지는 어떤 비교표도 추정에 머물기 때문입니다. 핵심은 작게 시작하는 것입니다. ClickHouse는 단일 노드로 PoC를 시작했다가, 검증이 끝난 뒤 데이터가 늘면 클러스터로 확장하는 경로가 열려 있습니다 [S06]. 그래서 초기에는 서버 한 대로 충분하며, 인프라 준비 부담이 작습니다.

성공 여부는 시작 전에 숫자로 정의해야 합니다. "괜찮아 보인다"가 아니라 "수집 누락률 몇 % 이하, 대시보드 응답 몇 초 이내"처럼 측정 가능한 기준을 세워야, PoC가 끝난 뒤 가도 되는지 멈춰야 하는지를 두고 다툴 일이 줄어듭니다. 아래 일정과 지표를 출발점으로 삼아 우리 상황에 맞게 조정하시면 됩니다.

주차	범위	성공 지표(예시, 직접 조정)
1주차	단일 노드 설치, OTel로 핵심 서비스 2~3개 계측	트레이스·로그 수집 누락률 5% 이하

주차	범위	성공 지표(예시, 직접 조정)
1주차 후반	대시보드·알림 1차 구성	주요 화면 응답 3초 이내
2주차	현행 Datadog 사용량 일부를 병행 수집해 환산 비교	동일 사용량 기준 비용 차이 정량화
2주차 종료	결과 리뷰, 가/부 판단	위 지표 충족 여부로 결론

비용도 PoC 범위에 함께 넣으시길 권합니다. 검증 기간 동안 실제로 발생한 인프라 사용량을 기록하면, 7.1.2의 워크시트를 추정이 아닌 실측으로 채울 수 있습니다 [S09]. 2주를 넘기지 않도록 범위를 좁게 유지하는 것이 이 설계의 핵심입니다.

7.2.2 리스크와 한계 명시

마지막으로, 시작 전에 한계를 솔직하게 적어두는 단계입니다. 이 항목이 필요한 이유는, 리스크를 미리 드러내면 PoC 중에 놀랄 일이 줄고 의사결정자에게도 균형 잡힌 그림을 줄 수 있기 때문입니다. SigNoz는 강점이 분명하지만 완벽하지 않으며, 한계를 숨기지 않는 편이 신뢰할 만한 판단으로 이어집니다.

대표적인 리스크는 세 가지입니다. 첫째는 셀프호스트 운영 부담입니다. 설치·업그레이드·장애 대응을 우리가 직접 감당해야 합니다. 둘째는 상용 전용 기능 부재입니다. SSO·RBAC 같은 일부 기능은 커뮤니티 에디션에 없습니다 [S10]. 게다가 오픈코어 모델이라 특정 기능이 상용 전용으로 분리돼 있고, 라이선스 조건은 도입 전에 다시 확인할 필요가 있습니다 [S05]. 셋째는 가격 변동성입니다. 클라우드 옵션의 가격은 제공사 정책에 따라 바뀔 수 있어, 장기 비용을 한 시점 가격으로 단정하기 어렵습니다.

리스크	영향	완화 방안
셀프호스트 운영 부담	운영 인력 시간 소요	단일 노드로 작게 시작, 필요 시 클라우드 옵션으로 전환 [S06]
상용 전용 기능 부재(SSO·RBAC)	거버넌스 요건 충족 제약	PoC는 커뮤니티로 검증, 필요 기능은 본 도입 시 상용·클라우드로 확보 [S10]
오픈코어 라이선스 조건	기능별 사용 범위 제약	도입 전 라이선스 원문 재확인, 핵심 기능이 커뮤니티에 포함되는지 점검 [S05]
클라우드 가격 변동성	장기 비용 예측 불확실	셀프호스트와 클라우드 비용을 병렬로 추정해 한쪽에만 묶이지 않게 설계

이 표를 PoC 착수 문서에 그대로 붙여두시면, 검증 도중 마주치는 문제가 예상 범위 안인지 밖인지 바로 가릴 수 있습니다. 각 리스크 옆에 완화 방안이 한 줄씩 붙어 있다는 점이 중요합니다. 한계가 있다는 사실 자체가 도입을 막는 사유는 아니며, 대비책과 함께 보면 관리 가능한 항목으로 바뀝니다.

여기까지 따라오셨다면 다음 주에 할 일은 명확합니다. 먼저 7.1.1 자가 진단표를 채워 우리 환경의 적합성을 확인하고, 7.1.2 워크시트에 지난 청구서를 분해해 넣으십시오. 두 결과가 긍정적이면 7.2.1 일정대로 단일 노드 PoC를 띄우고, 핵심 서비스 두세 개를 OTel로 계측해 1~2주 동안 실제 데이터로 돌려보시길 권합니다. 시작 전 7.2.2의 리스크 표를 착수 문서에 붙이고, 성공 지표를 숫자로 정해 두면 됩니다. PoC가 끝나는 날, 추정이 아니라 우리 손으로 측정한 근거를 들고 도입 여부를 결정하실 수 있습니다.

부록 (Appendix)

Appendix A. References

본문에 인라인 식별자([S##])로 인용한 출처입니다. 모든 URL·수치·연도는 게시 직전 1차 출처로 재확인하시기를 권합니다.

ID	출처	URL
S01	SigNoz — About us (창업자·이름 유래)	https://signoz.io/about-us/
S02	SigNoz — Genesis of SigNoz (창업 배경·동기·스토리지 전환)	https://signoz.io/blog/genesis-of-signoz/
S03	SigNoz — Open-Source Alternative to DataDog	https://signoz.io/blog/open-source-datadog-alternative/
S04	SigNoz — OpenTelemetry vs Datadog	https://signoz.io/blog/opentelemetry-vs-datadog/
S05	SigNoz — GitHub LICENSE & Discussion #4231 (오픈소스 진위 논쟁)	https://github.com/SigNoz/signoz
S06	SigNoz — ClickHouse를 스토리지 백엔드로 (Druid→ClickHouse 전환)	https://signoz.io/blog/clickhouse-storage-monitoring/
S07	SigNoz — Zabbix vs Prometheus	https://signoz.io/comparisons/zabbix-vs-prometheus/
S08	SigNoz — Prometheus vs Elasticsearch stack	https://signoz.io/blog/prometheus-vs-elasticsearch/
S09	SigNoz — Pricing	https://signoz.io/pricing/
S10	SigNoz — SSO Overview / Enterprise	https://signoz.io/docs/manage/administrator-guide/sso/overview/

ID	출처	URL
S11	SigNoz — Cloud Teams Plan now at $49/Month	https://signoz.io/blog/cloud-teams-plan-now-at-49usd/

Appendix B. Glossary

용어	정의
옵저버빌리티 (Observability)	메트릭·로그·트레이스 같은 외부 출력으로 시스템 내부 상태를 추론하는 능력
OpenTelemetry (OTel)	텔레메트리 데이터 수집·전송을 위한 CNCF의 벤더 중립 오픈 표준
OTLP (OpenTelemetry Protocol)	OpenTelemetry 데이터를 전송하는 표준 프로토콜
분산 추적 (Distributed Tracing)	하나의 요청이 여러 서비스를 거치는 경로를 이어 붙여 추적하는 기능
트레이스 (Trace)	하나의 요청이 서비스들을 거친 경로를 기록한 단위
ClickHouse	SigNoz가 세 신호를 함께 저장하는 컬럼형 데이터베이스
컬럼형 (Columnar)	데이터를 행이 아니라 열 단위로 저장해 집계·분석 질의에 유리한 방식
오픈코어 (Open-core)	코어는 오픈소스로 공개하고 일부 기능만 상용 라이선스로 분리하는 모델
MIT (Expat) 라이선스	제약이 적은 허용형(permissive) 오픈소스 라이선스
셀프호스트 (Self-host)	벤더의 클라우드가 아니라 자체 인프라에 직접 설치·운영하는 방식
벤더 락인 (Vendor lock-in)	특정 벤더 제품에 묶여 교체 비용이 커지는 상태
RBAC (Role-Based Access Control)	역할 기반으로 데이터·기능 접근 권한을 제어하는 방식
SSO (SAML 2.0 / OIDC)	단일 인증으로 여러 시스템에 로그인하는 통합 인증
APM (Application Performance Monitoring)	애플리케이션 성능 모니터링
custom metric	Datadog이 별도 프리미엄으로 과금하는 사용자 정의 메트릭

용어	정의
TCO (Total Cost of Ownership)	라이선스·인프라·운영을 포함한 총소유비용
SaaS (Software as a Service)	벤더가 클라우드에서 운영해 주는 구독형 소프트웨어
클라우드 네이티브 (Cloud-native)	컨테이너·오케스트레이션을 전제로 설계된 동적 환경
마이크로서비스 (Microservices)	기능을 작은 서비스로 잘게 나눈 아키텍처

Datadog 대신 오픈소스 모니터링, SigNoz가 가능한가

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