MSAP.ai

AI Native 옵저버빌리티 세미나 · Session 01

eBPF 기반 여러 언어 서비스 모니터링

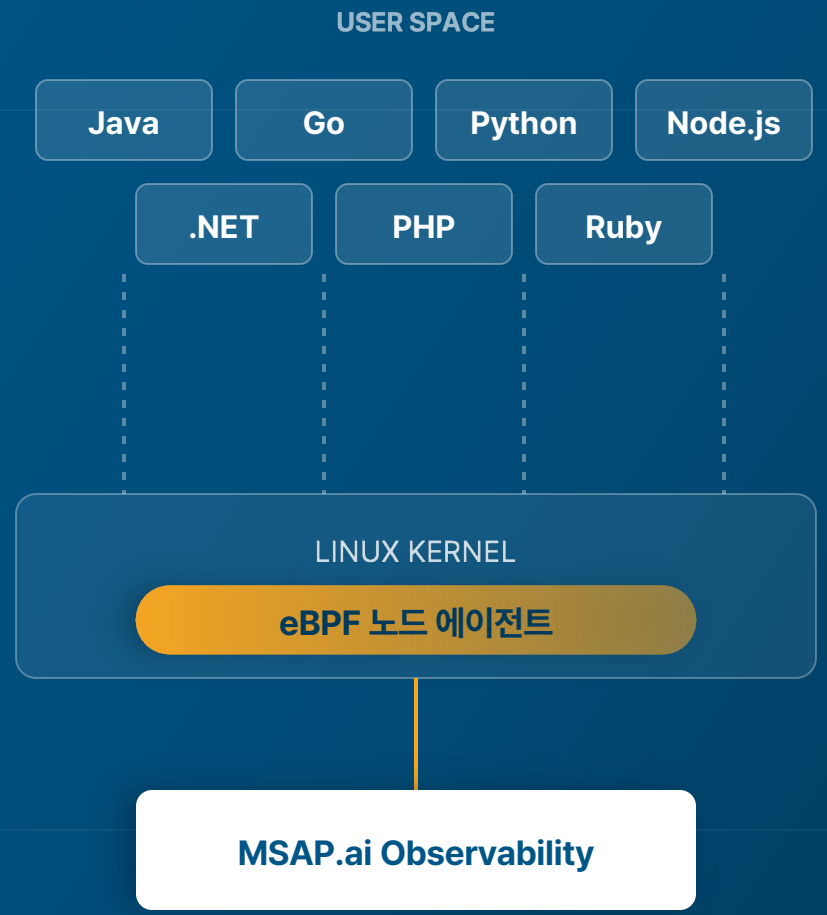
코드 수정·재기동·개발팀 협의 없이 시작하는

MSAP.ai Observability

SESSION

Session 01 · 코드 수정 없는 모니터링

개념 · 문제 · 해결 · 데모 · 정리



© MSAP.ai. All rights reserved.

Session 01 구성

KEY INSIGHT

에이전트 방식이 느린 이유를 먼저 확인하고

eBPF 방식이 그 원인을 어떻게 없애는지 데모로 확인함

01 개념

Observability 와 eBPF — 장애 원인 분석에 필요한 데이터와 수집 기술

02 문제

언어별 계측 작업과 조직 간 협의 — 도입 기간이 길어지는 원인

03 해결

커널에서 수집하는 eBPF 방식 — 언어·레거시 범위, 안전장치, 자원 사용량, 한계

04 데모

쇼핑몰 서비스 10개를 코드 수정 없이 수집한 화면

05 정리

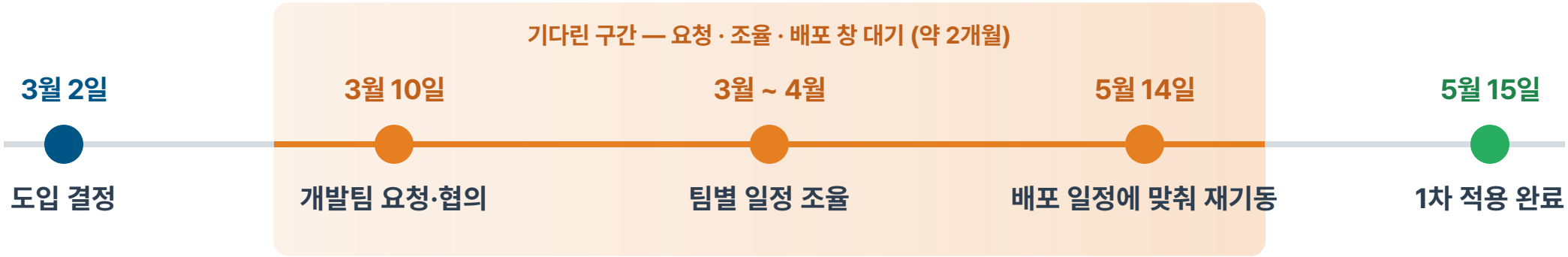
에이전트 방식과 eBPF 방식 비교, 도입 전 확인 항목과 적용 순서

3개월 뒤에도 서비스 47개 중 6개만 보임

KEY INSIGHT

시간 대부분이 개발팀 요청·일정 조율·배포 대기에 쓰임

나머지 41개는 일정도 못 잡은 상태



5월 15일 기준 모니터링되는 서비스

6 / 47

전체 서비스의 약 13%

- 시간이 걸린 단계는 제품 선정이나 설치가 아님
- 두 달 넘게 개발팀 요청·팀별 일정·배포 창을 기다림
- 나머지 41개 서비스는 적용 일정도 아직 없음

Source: 국내 APM 도입에서 자주 나타나는 일정 예시 · 기존 방식 도입 기간은 사내 제품 자료 기준 "수주~수개월"

반나절 도입의 세 가지 조건

KEY INSIGHT

세 작업이 빠지면 관측 시작 시점이 서비스 배포 일정과 분리됨

운영팀이 직접 도입 시점을 정할 수 있음

설치·협의·재기동이 빠지면 반나절 도입 가능

에이전트 설치 없음

서비스마다 에이전트를
붙이는 작업 없음

개발팀 협의 없음

코드·실행 옵션 변경을
요청할 일 없음

재기동 없음

운영 중인 서비스에
바로 적용

세 조건을 만족하는 방법: 애플리케이션 대신 리눅스 커널에서 수집

PART 01 · 개념

Observability 개념과 eBPF 수집 기술

장애 원인을 찾는 데 필요한 데이터와 그 데이터를 모으는 방법

01 모니터링과 Observability 의 차이

KEY INSIGHT **Observability 는 예상하지 못한 장애에도 원인을 좁힐 수 있음**
서버에 접속하지 않고 시스템이 내보내는 데이터만 사용함

모니터링

known unknowns

"CPU 가 80% 를 넘으면 알려줘"

- 미리 알고 있는 문제만 탐지
- 서버 하나하나를 따로 확인
- 무엇이 문제인지까지 확인
- 등록하지 않은 문제는 알림 없음

Observability

unknown unknowns

"어제 새벽 지연은 어느 서비스에서?"

- 예상하지 못한 문제까지 추적
- 서비스 사이의 호출 흐름 전체를 확인
- 왜 문제인지까지 확인
- 흩어진 신호를 겹쳐 원인을 좁힘

OBSERVABILITY 정의
서버나 운영 환경에 직접 접속하지 않고, 외부로 나오는 데이터로 시스템 내부 상태를 파악하는 능력

Source: CNCF TAG Observability Whitepaper v1.0 (2023-10)

01 장애 분석에 쓰는 네 가지 신호

KEY INSIGHT

메트릭으로 이상을 찾고 트레이스·로그·프로파일링으로 위치를 좁힘

네 신호가 한 화면에서 연결돼야 원인까지 도달함



METRICS

메트릭

무엇이 이상한가

CPU · 메모리
응답 시간 같은 숫자



LOGS

로그

무슨 일이 있었나

오류 메시지
반복 패턴 같은 사건 기록



TRACES

트레이스

어디서 지연됐나

서비스 간 호출 경로와
구간별 소요 시간



PROFILING

프로파일링

어느 코드가 느린가

함수별 CPU 점유
플레임 그래프

원인을 좁히는 순서 — 이상 발견 → 느린 구간 찾기 → 사건 확인 → 느린 코드 확인

프로파일은 2024년 OpenTelemetry 신호로 발표, 2026년 공개 알파 — 설문 응답 조직 중 프로파일 사용은 16%

Source: OpenTelemetry Signals 문서 · Profiles Public Alpha (2026) · Grafana Labs Observability Survey 2025

01 eBPF 개념과 관찰 위치

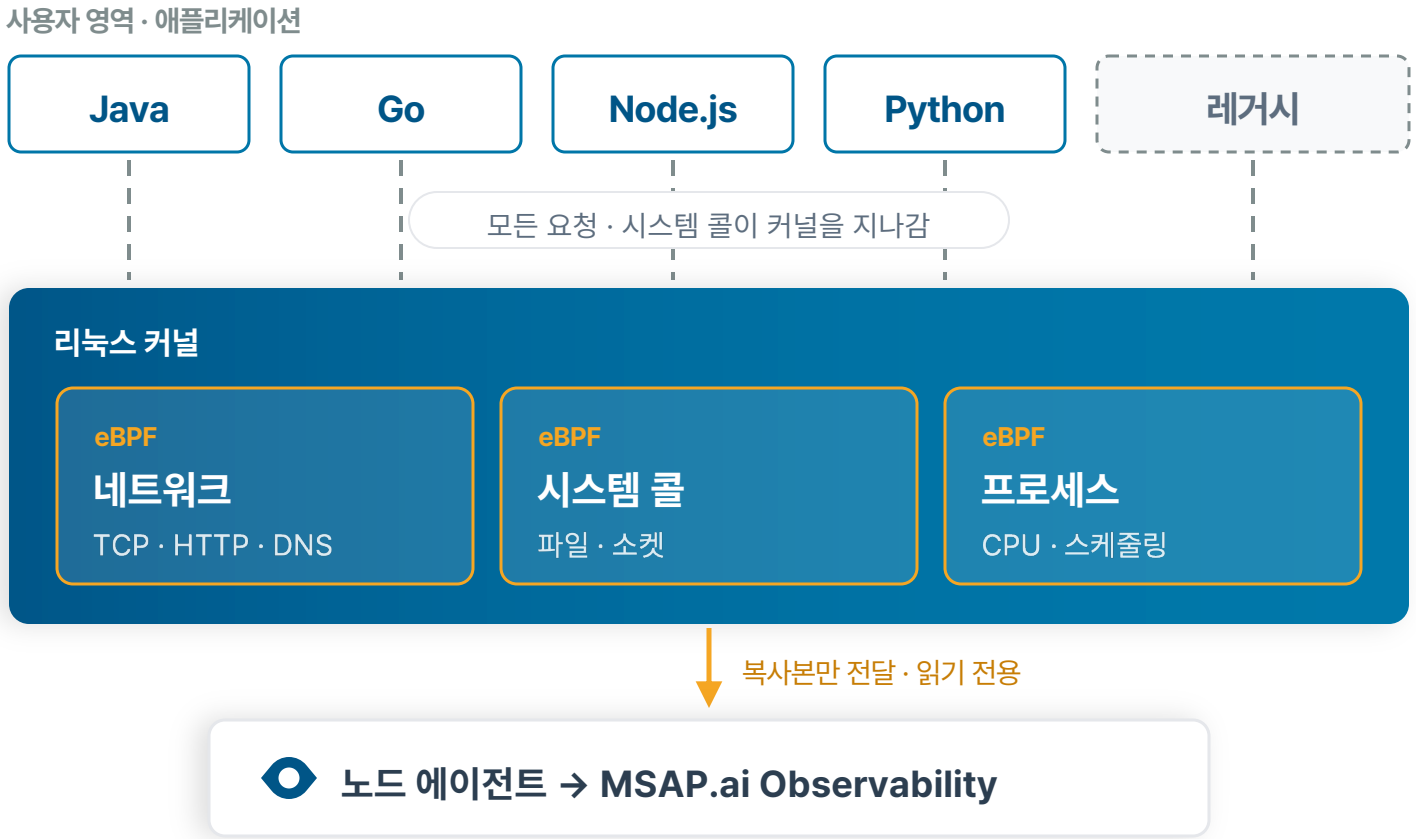
KEY INSIGHT **언어와 상관없이 모든 요청과 시스템 콜이 커널을 지나감**
그 지점에서 보면 애플리케이션을 고칠 필요가 없음

eBPF
extended Berkeley Packet Filter

커널을 고치지 않고 커널 안에서 작은 프로그램을 안전하게 실행하는 기술

- 커널 소스 변경·커널 모듈 없이 동작
- 지나가는 데이터의 복사본만 읽음
- 1992년 패킷 필터에서 출발
- 2014년 리눅스 커널 범용 기능으로 확장

Cilium · Falco 등 쿠버네티스 도구가 이미 사용



Source: ebpf.io "What is eBPF?" (2026) · eBPF Case Studies — Google · Netflix · Meta · Cloudflare 등 운영 사례

eBPF 프로그램 로드 과정

KEY INSIGHT

Verifier 가 로드 전에 무한 루프와 범위 밖 메모리 접근을 거부함

커널을 멈출 수 있는 코드는 실행 단계까지 가지 못함

1. **작성·컴파일** — 바이트코드 생성
2. **Verifier 검사** — 통과 못 하면 로드 거부
3. **JIT 컴파일** — CPU 명령어로 변환
4. **훅 연결** — 네트워크·시스템 콜
5. **Map 으로 전달** — 사용자 영역에서 읽음

Verifier 가 로드 전에 확인하는 네 가지

반드시 끝나는지

무한 루프 없음, 유한 반복만

메모리 접근 범위

범위 밖 메모리 접근 거부

크기·복잡도 상한

너무 큰 프로그램 거부

로드 권한

CAP_BPF 또는 root 권한 필요

범위 안내

Verifier 는 커널 안정성을 지키는 안전성 검사이며, 프로그램의 의도를 판정하는 보안 검사 도구는 아님

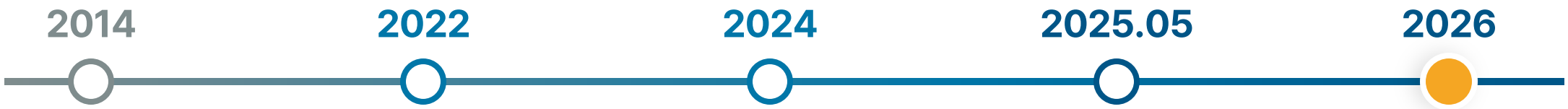
Source: ebpf.io "What is eBPF?" (2026) · kernel.org "eBPF verifier" 문서

eBPF 계측과 프로파일링의 표준화 과정

KEY INSIGHT

코드 수정 없는 eBPF 수집이 OpenTelemetry 공식 프로젝트가 됨

특정 벤더에 묶이지 않는 업계 공통 방식으로 정착 중



2014 — eBPF 확장

리눅스 커널의 범용 기능으로 편입

2022 — Profiling SIG

OpenTelemetry 에 프로파일링 작업 그룹 결성

2024 — 프로파일링 신호 발표

Elastic 이 eBPF 프로파일러를 OpenTelemetry 에 기증

2025.05 — Beyla 기증

Grafana Beyla 코드 기증 OpenTelemetry eBPF Instrumentation 출범

2026 — 표준으로 정착 중

OBI v1.0 출시 OpenTelemetry Profiles 공개 알파

OBI(OpenTelemetry eBPF Instrumentation) — 코드·설정 변경 없이 eBPF 로 계측하는 공식 프로젝트

커스텀 스팬·업무 속성은 여전히 언어별 SDK 가 필요하다고 공식 문서에 명시 · Profiles 는 아직 핵심 운영 환경 사용 비권장

Source: opentelemetry.io 블로그 (2024 profiling · 2026 obi-v1-release · 2026 profiles-alpha) · grafana.com 블로그 (2025-05) · Elastic Observability Labs (2024)

PART 02 · 문제

에이전트 방식 계측의 지연 원인

언어별 계측 작업과 조직 간 협의

02 언어마다 다른 계측 작업

KEY INSIGHT

작업 하나는 쉽지만 서비스 수만큼 반복됨

서비스 47개에 언어가 넷이면 네 종류 작업을 47번 수행함

JAVA

-javaagent

JVM 실행 옵션 변경
적용하려면 재기동

재기동 필요

PYTHON

패키지 + 래퍼

pip 로 계측 패키지 설치
실행 명령을 래퍼로 변경

배포 설정 수정

NODE.JS

로딩 순서

다른 의존성보다 먼저 로드
번들러와 충돌 가능

시작 코드 수정

GO

소스 수정

자동 계측이 제한적
SDK 를 직접 코드에 추가

재빌드·재배포

4종 × 47개

작업 종류 × 서비스 수

하나하나는 어렵지 않은 작업

서비스와 언어가 늘수록 작업도 같이 늘어남

Source: OpenTelemetry 언어별 계측 문서 · 사내 기술 문서 why-observability 01

02 도입 기간을 늘리는 원인은 조직 간 협의

KEY INSIGHT

관측 시작 시점이 서비스 조직의 배포 일정에 묶임

새 서비스가 들어올 때마다 같은 협의를 반복함

기술적인 일

30분

서비스당

- 실행 옵션 추가·패키지 설치
- 문서대로 따라 하면 완료

조직적인 일

3개월

47개 중 41개는 아직 시작 전

- 개발팀에 요청 후 백로그 대기
- 팀마다 다른 우선순위
- 배포 일정까지 대기
- 재기동 때문에 운영팀 승인
- 레거시는 담당자가 없음

업계 설문 · 옵저버빌리티 도입의 가장 큰 걸림돌



2년 연속 1위 걸림돌

응답 1,255명 · 1,363명

Source: Grafana Labs Observability Survey 2025 · 4th Annual Observability Survey (2026-03) · 사내 기술 문서 why-observability 01

PART 03 · 해결

애플리케이션 변경 없는 수집 방법

애플리케이션은 그대로 두고 그 아래 계층에서 수집

03 커널 계층에서 수집하는 구조

KEY INSIGHT

모든 통신이 커널을 지나므로 언어별 계측이 필요 없음

노드마다 에이전트 하나만 설치하고 애플리케이션은 변경 없음

애플리케이션 계층

Java | Go | Python | Node.js | 레거시 | 상용 패키지

컨테이너 · OS 계층

CPU · 메모리 같은 자원 지표

리눅스 커널

eBPF 노드 에이전트

모든 요청 · 시스템 콜이 지나가는 지점

쿠버네티스에서는 DaemonSet 으로 노드마다 하나

여기서 계측하면

언어마다 다른 계측 작업 필요

레거시는 계측 불가

어떤 도구든 비슷하게 수집

요청 흐름·쿼리는 보이지 않음

여기서 수집하면

- 코드 수정 없음
- 언어와 무관하게 같은 방식
- 재기동 없음

노드마다 에이전트 하나 — 애플리케이션 쪽에서는 바뀌는 것 없음

Source: 사내 기술 문서 why-observability 01 · 90-03

03 eBPF 방식에서 빠지는 세 가지 작업

KEY INSIGHT **개발팀 일정과 관계없이 운영 중에 바로 적용 가능함**
도입 기간이 수주~수개월에서 수시간 단위로 줄어듦

01 에이전트 설치

없어짐

서비스마다 붙일 것 없음
노드에 한 번만 설치

02 개발팀 협의

없어짐

코드도 실행 옵션도
건드리지 않아 요청할 일 없음

03 재기동

없어짐

배포 창을 기다리지 않음
운영 중인 서비스에 적용

도입 기간

수주 ~ 수개월 ▶ 수시간

Helm 설치 직후부터 수집 시작
개발팀 일정과 관계없이 운영팀이 진행

Source: 사내 제품 소개 자료 V04 (2026-03) 도입 시간 비교 · 사내 기술 문서 why-observability 01

03 언어 구분 없는 자동 수집

KEY INSIGHT

새 언어로 만든 서비스도 배포되는 즉시 목록에 나타남

언어가 늘어도 수집 작업은 늘지 않음

Java

Spring · Tomcat

Go

net/http · gRPC

Python

Django · FastAPI

Node.js

Express · NestJS

.NET

ASP.NET

PHP

웹 애플리케이션

Ruby

웹 애플리케이션

C · C++

네이티브 바이너리

커널은 그 위에서 어떤 언어가 돌아가는지 구분하지 않음 — 같은 방식으로 수집

참고 · 업계 표준

OpenTelemetry eBPF Instrumentation 공식 지원 언어도 9종 — Java · .NET · Go · Python · Ruby · Node.js · C · C++ · Rust

Source: 사내 제품 소개 자료 V04 (2026-03) · opentelemetry.io OpenTelemetry eBPF Instrumentation 문서 (2026)

03 소스 없는 레거시와 외부 솔루션까지 수집

KEY INSIGHT

SDK 를 넣을 수 없는 프로그램도 통신만 하면 같은 화면에 나타남

장애가 자주 나는 영역이 모니터링 사각지대에서 빠져나옴

LEGACY

소스 없는 레거시

담당자도 소스도 없는
오래된 jar · 바이너리

재빌드 없이 수집

3RD PARTY

외부 솔루션·상용 패키지

벤더가 만든 프로그램에는
SDK 를 넣을 수 없음

노드 위에서 돌면 같은 방식으로 수집

MIDDLEWARE

DB · 캐시 · 메시지 큐

에이전트를 설치하지 않은
DB 의 쿼리 · 응답 시간

프로토콜을 해독해 수집

코드 변경 없이 자동 인식하는 주요 프로토콜

HTTP · HTTP/2 · gRPC · PostgreSQL · MySQL · Oracle · MongoDB · Redis · Memcached · Kafka · Cassandra · RabbitMQ · NATS · Dubbo2 · ClickHouse · ZooKeeper · DNS

클러스터 밖 SaaS · 외부 API 는 우리 서비스가 호출하는 구간까지 확인 — 그 시스템 내부까지는 보이지 않음

Source: 사내 기술 문서 why-observability 90-04 (eBPF 프로토콜 지원 범위)

03 프로토콜 지원 깊이 세 단계

KEY INSIGHT **해독 깊이에 따라 DB 장애 분석 속도가 달라짐**
도구를 비교할 때는 지원 목록보다 해독 단계를 확인해야 함

"지원 프로토콜" 이라는 말의 범위

목록에 같은 이름이 있어도
어느 단계까지 읽는지에 따라
장애 분석에 쓸 수 있는 정보가 다름

1단계
감지
어떤 프로토콜인지 식별

2단계
요청 해독
SQL 문장 · HTTP 경로
Kafka 토픽

많은 eBPF 도구가 여기까지 지원

3단계
값 · 크기 해독
바인드 값 · 응답 바이트 · DB 오류 코드

`WHERE id = $1 → $1 = 12345`

· PostgreSQL · MySQL 바인드 값 표시
· Oracle TNS 해독 — SQL · ORA- 오류 (12c 이상)
· DB 6종과 HTTP 응답 크기 수집

느린 쿼리를 실제 값과 함께 재현 가능

바인드 값 수집은 설정으로 끌 수 있음 · DB 자체 암호화를 켠 연결은 내용을 읽지 못함

Source: 사내 기술 문서 why-observability 90-04 (eBPF 프로토콜 지원 범위)

03 eBPF 의 운영 서비스 보호 장치

KEY INSIGHT

검사를 통과한 코드만 읽기 전용으로 프로세스 밖에서 동작함

힙·스레드·GC 에 관여하지 않아 서비스 응답 시간이 늘지 않음



01

검증된 코드만 로드

Verifier 가 실행 전에 검사
무한 루프·잘못된 메모리 접근이
있으면 로드 거부



02

읽기 전용

요청을 가로채거나 바꾸지 않음
지나가는 데이터의 복사본만
확인



03

애플리케이션 밖에서 동작

프로세스에 붙는 것 없음
Java 에이전트와 달리
힙·스레드·GC 에 관여하지 않음

대규모 운영 사례

Google GKE Dataplane V2 · Meta Katran
Netflix 네트워크 분석

대규모 운영 환경이 이미 eBPF 위에서 동작



보안 검토 항목

노드 에이전트는 privileged
권한으로 실행됨

도입 전 권한 정책 확인 필요

Source: ebpf.io "What is eBPF?" · eBPF Case Studies (2026) · 사내 기술 문서 why-observability 01 · 90-03

03 노드 에이전트 자원 사용량 실측

KEY INSIGHT

상시 부담이 작아 운영 노드에 계속 켜 둘 수 있음

부하는 프로브를 건 이벤트 빈도에 비례하므로 일부 노드부터 켜고 확대함

상시 CPU 평균

0.61 코어

72코어 운영 노드 기준

개선 전 0.82코어

기동 순간 CPU 최고치

0.86 코어

128코어 · 파드 밀집 노드

개선 전 14.7코어

메모리

473~941 Mi

파드 26개 ~ 104개 노드

파드 수에 비례

서비스 응답 시간

영향 없음

수집이 애플리케이션
처리 경로 밖에서 일어남

외부 실측 참고

Netflix

eBPF 훅 하나당 600ns 미만으로 상시 운영

컨테이너별 스케줄링 지연 측정

Cloudflare

매우 잦은 시스템 콜에 프로브를 걸면 부하 증가

그래서 일부 데이터센터에서 먼저 켜고 확대

도입 순서 — 일부 노드에서 자원 사용량을 확인한 뒤 전체 노드로 확대

Source: 사내 기술 문서 why-observability 90-02 (2026-07 측정) · InfoQ Netflix 사례 (2024-09) · Cloudflare 블로그 (2018)

03 eBPF 로 보이지 않는 영역과 보완 방법

KEY INSIGHT

업무 식별자와 일부 암호화 구간은 커널에서 읽을 수 없음

필요한 서비스만 OpenTelemetry SDK 를 병행하면 같은 화면에서 보완됨

eBPF 로 보이지 않는 영역

업무 식별자

주문번호처럼 애플리케이션 안에만 있는 값

커스텀 스팬 · 비즈니스 이벤트

코드 안에서 직접 정의하는 구간과 속성

일부 HTTPS 본문

Java(JSSE), OpenSSL 을 정적 링크한 Node.js

DB 자체 암호화 연결 · 외부 SaaS 내부

암호화된 DB 프로토콜 내용, 호출 대상 시스템의 내부 동작

커널을 지나가지 않거나 암호화된 정보는 해독 불가

보완 방법

OpenTelemetry 데이터 함께 수집

SDK 로 보낸 트레이스를 같은 화면에 표시

핵심 서비스만 선택적으로 SDK 적용

업무 식별자가 꼭 필요한 서비스에 한정

DB 내부 지표는 DB 계정으로 수집

잠금 대기 · 느린 쿼리 통계는 DB 에 직접 조회

나머지는 eBPF 로 기본 수집

전체 서비스는 노드 에이전트, 깊이는 필요한 곳만

OpenTelemetry eBPF 공식 문서도 커스텀 스팬은 SDK 필요로 명시

Source: 사내 기술 문서 why-observability 01 · 90-04 · opentelemetry.io OpenTelemetry eBPF Instrumentation 문서 (2026)

PART 04 · 데모

여러 언어 서비스 수집 데모

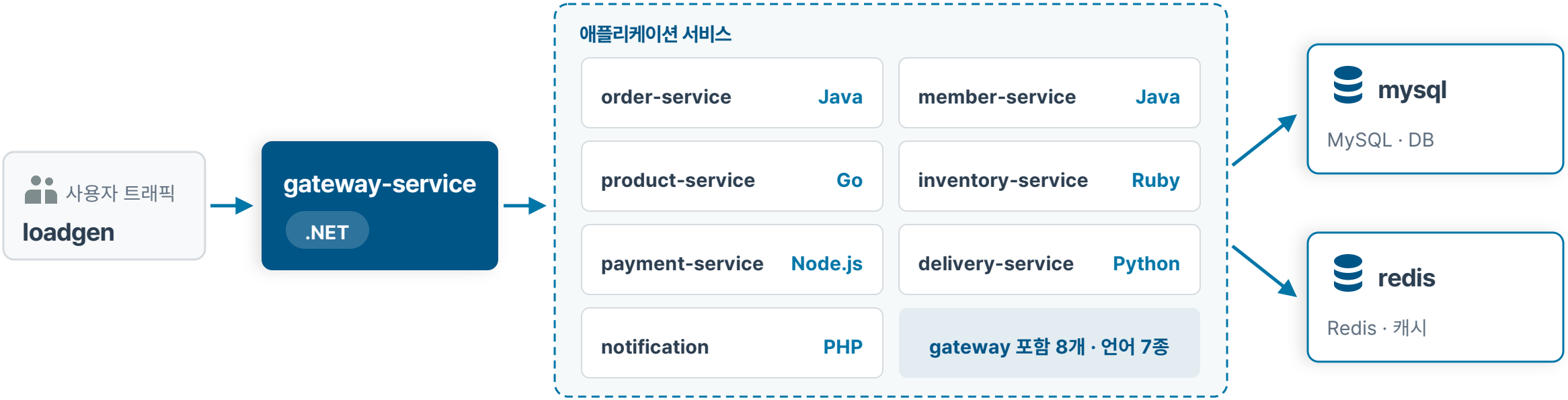
언어가 다른 쇼핑몰 서비스 10개를 코드 수정 없이 수집한 환경

04 데모 환경: 쇼핑몰 서비스 10개

KEY INSIGHT

언어가 다른 서비스 8개와 DB·캐시를 노드 에이전트 하나로 수집함

서비스 쪽에는 SDK 도 에이전트도 넣지 않음



노드마다 eBPF 에이전트 하나 — 서비스 10개를 모두 여기서 수집

언어가 달라도 코드는 그대로 · Session 02 장애 추적도 같은 환경에서 이어짐

Source: 데모 환경 commerce-platform (쿠버네티스 네임스페이스) 구성

04 데모에서 확인할 세 장면

KEY INSIGHT

서비스·DB·전체 현황을 화면 전환 없이 한 제품 안에서 확인함

도구마다 따로 보던 데이터가 한곳에 모임

SCENE 1 — 1

여러 언어 서비스를 코드 수정 없이 확인

Go 상품 · Python 배송
Node.js 결제 · Java 회원

메트릭 · 로그 · 프로파일 · 분산 추적

언어 무관 자동 수집

SCENE 2 — 2

손대지 않은 DB 의 실제 쿼리 확인

MySQL 쿼리별 횟수·소요 시간
쿼리를 호출한 서비스

DB 에 접속하지 않고 노드 에이전트로 수집

외부 솔루션 추적

SCENE 3 — 3

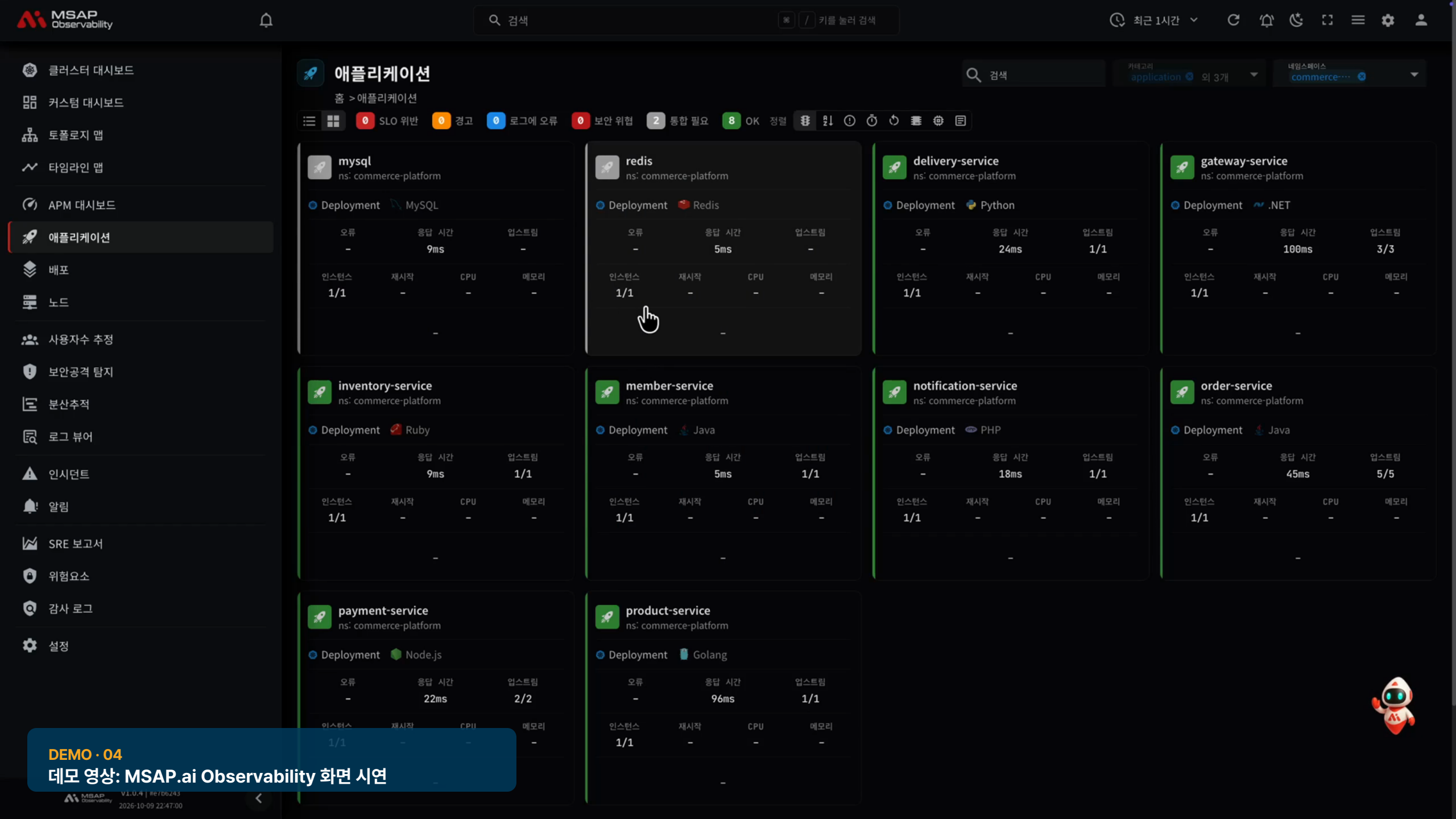
APM 대시보드의 전체 현황 확인

처리량 · 응답 시간 · 오류율
트랜잭션 히트맵

HTTP 요청과 DB 쿼리를 한 목록에 표시

전체 서비스 현황

다음 장 — 데모 영상 약 5분 · 처음부터 끝까지 MSAP.ai Observability 화면 하나에서 진행



DEMO · 04

데모 영상: MSAP.ai Observability 화면 시연

PART 05 · 정리

eBPF 방식 도입 효과와 준비 사항

보이는 범위와 작업량, 그리고 도입 순서

05 에이전트 방식과 eBPF 방식 비교

KEY INSIGHT **보이는 범위가 적용한 서비스에서 노드 위 전체 서비스로 넓어짐**
서비스가 늘어도 작업량은 그대로임

항목	에이전트 방식	eBPF 방식
도입 작업	개발팀에 계측 요청	**노드에 DaemonSet 배포**
일정	팀별 일정 조율	**개발팀 협의 없음**
적용 방식	배포 창 확보 후 재기동	**재기동 없음**
보이는 범위	적용한 서비스만	**노드 위 모든 서비스**
서비스 증가 시	언어·서비스만큼 작업 증가	**작업량 변화 없음**
도입 기간	수주 ~ 수개월	**수시간**

Source: 사내 제품 소개 자료 V04 (2026-03) 방식 비교 · 사내 기술 문서 why-observability 01

05 도입 전 확인 항목과 적용 순서

KEY INSIGHT

커널 버전과 권한 정책만 먼저 맞추면 설치는 Helm 한 번으로 끝남

보안 검토 범위를 도입 전에 정해야 함

도입 전 확인 항목

리눅스 커널 버전

제품 지원 버전 표와
노드 커널 대조

privileged 권한

노드 에이전트 실행 권한
보안 검토 · 승인

CPU 아키텍처

amd64 · arm64
노드 이미지 지원

배포 방식

쿠버네티스: Helm · DaemonSet
그 밖: Docker Compose

적용 순서

1 일부 노드에 설치
Helm 으로 DaemonSet 배포

2 수집·자원 사용량 확인
서비스 목록 · 노드 CPU · 메모리

3 전체 노드로 확대
새 노드는 자동으로 포함

4 필요한 곳만 SDK 병행
업무 식별자가 필요한 서비스

에이전트는 노드 · 클러스터 · DB 용으로 나뉘어 권한 범위를 각각 검토 가능

Source: 사내 기술 문서 why-observability 01 · 90-03 (배포와 운영)

SESSION 01 정리

모니터링 도입이 노드 설치 한 번으로 바뀜

여러 팀과 협의하던 프로젝트가 노드에 에이전트를 한 번 설치하는 작업으로 바뀜

- 코드 수정 없음
- 재기동 없음
- 노드 위 모든 서비스 수집

NEXT · SESSION 02

MSA 장애, AI 분석으로 원인 서비스부터 찾기

THANK YOU

도입 문의와 연락처

코드 수정·재기동·개발팀 협의 없이 시작하는 옵저버빌리티

MSAP.ai Observability

https://www.msap.ai/

02-6953-5427

hello@msap.ai

© MSAP.ai. All rights reserved.