MSAP.ai

2025.12

AI Native Observability : LLM과 관측 데이터의 결합으로 이룬 운영 혁신

쿠버네티스·MSA 시대에 폭발적으로 증가하는 운영 데이터와 LLM 기반 AI 분석을 결합하여, 기존 모니터링이 해결할 수 없던 운영 인텔리전스의 공백을 메우고, MTTR 단축·운영 자동화·비즈니스 효율성 향상을 실현하는 'AI Native Observability'의 기술적 구조와 실질적 활용 전략을 제시합니다.

Contact Us

02-6953-5427

hello@msap.ai

www.msap.ai

Contents

제1장. 클라우드 네이티브 환경의 가시성 확보와 Observ-ability 패러다임 전환

서론: 현대 IT 환경의 복잡성과 새로운 도전 과제

기업들이 마이크로서비스 아키텍처(MSA), 쿠버네티스(Kubernetes)와 같은 현대적인 기술을 도입하면서 IT 시스템의 복잡성은 전례 없는 수준으로 증가하고 있습니다. 이러한 동적이고 분산된 환경은 과거에는 경험하지 못했던 새로운 운영상의 도전 과제를 제시합니다. 사용자의 단 한 번의 클릭이 수십, 수백 개의 마이크로서비스와 인프라 구성 요소에 걸친 연쇄적인 호출을 유발하며, 각 구성 요소는 쿠버네티스 위에서 수시로 생성, 확장, 소멸하기를 반복합니다. 이처럼 예측 불가능성이 높아진 환경에서 전통적인 모니터링 방식은 더 이상 시스템의 전체 상태를 파악하는 데 유효하지 않습니다. 새로운 환경에서 안정적인 가시성을 확보하기 위해서는 '모니터링'을 넘어 '관측 가능성(Observability)'으로 나아가는 근본적인 패러다임의 전환이 필요합니다.

1.1 마이크로서비스(MSA)·쿠버네티스 환경의 운영 난제

클라우드 네이티브 환경이 내포한 운영상의 어려움을 전략적으로 이해하는 것은 매우 중요합니다. 이는 단순히 해결해야 할 기술적 장애물이 아니라, 서비스 안정성, 비용 효율성, 그리고 개발자 생산성에 직접적인 영향을 미치는 근본적인 도전 과제입니다. 아래에서 다룰 네 가지 핵심 난제는 왜 우리가 새로운 운영 모델로 전환해야만 하는지를 명확하게 보여줍니다.

1.1.1 동적 인프라 확장에 따른 모니터링 사각지대 발생

클라우드 네이티브 환경에서 인프라는 더 이상 정적이지 않습니다. 컨테이너와 파드(Pod) 같은 구성 요소들은 필요에 따라 자동으로 생성, 확장, 소멸하는 일시적인(ephemeral) 특징을 가집니다. 또한, 클라우드 비용 효율성을 극대화하기 위한 핵심 전략으로 임시 서버(spot instances)의 사용이 보편화되었습니다. 이러한 동적인 특성은 고정된 모니터링 대상을 가정하는 전통적인 도

구들에게 심각한 '모니터링 사각지대'를 만듭니다. 수시로 변화하는 IP 주소와 인스턴스를 추적하고 관리하는 것은 기존 방식으로는 거의 불가능에 가깝습니다.

1.1.2 분산 트랜잭션 추적의 어려움과 장애 원인 파악 지연

마이크로서비스 환경에서 사용자의 단 한 번의 클릭은 내부적으로 수십, 수백 개의 서비스 간 연쇄적인 호출을 유발할 수 있습니다. 이 복잡한 호출 관계 속에서 특정 기능이 왜 느려졌는지, 어떤 서비스의 장애가 전체 시스템에 영향을 미쳤는지 추적하는 것은 '건초더미에서 바늘 찾기'와 같이 어려운 일이 되었습니다. 이러한 복잡성은 장애의 근본 원인 분석(Root Cause Analysis)을 지연시키고, 결국 평균 장애 해결 시간(Mean Time To Repair, MTTR)을 크게 증가시키는 직접적인 원인이 됩니다.

1.1.3 Metric·Log·Trace 데이터 파편화(Silo) 문제

시스템 상태를 파악하는 세 가지 핵심 데이터는 특정 시간의 집계된 수치를 통해 '무슨 일이 일어나고 있는지'를 보여주는 메트릭(Metrics), 개별 이벤트의 상세 기록으로 '무슨 일이 있었는지'를 알려주는 로그(Logs), 그리고 요청의 전체 경로를 추적하여 '어디서 문제가 발생했는지'를 밝히는 트레이스(Traces)입니다. 전통적인 환경에서는 이 데이터들이 각각 다른 목적을 가진 별개의 도구로 수집되어 데이터 사일로(Silo)에 갇히는 경우가 많았습니다. 이로 인해 엔지니어는 장애가 발생했을 때 여러 시스템을 오가며 데이터를 수동으로 연관 분석해야만 했습니다. 이는 시간이 오래 걸리고 비효율적이며, 사람의 실수로 인해 잘못된 결론에 도달할 위험이 큰 작업 방식입니다.

1.1.4 Polyglot 언어 사용 증가와 서비스 간 호출 확산

마이크로서비스 아키텍처는 각 서비스의 특성에 가장 적합한 프로그래밍 언어를 선택하여 사용하는 것을 권장합니다. 그 결과 Java, Python, Go, Node.js 등 다양한 언어로 개발된 서비스가 공존하는 '폴리글랏(Polyglot)' 환경이 보편화되었습니다. 이는 특정 언어에 종속된 에이전트나 계측(Instrumentation) 방식에 의존하는 기존 모니터링 도구에 큰 도전 과제를 안겨줍니다. 다양한 언어를 통합적으로 지원하지 못하면 운영 복잡성은 가중될 수밖에 없으며, 언어와 무관한(language-agnostic) 통합 모니터링 솔루션의 필요성이 대두됩니다.

이 네 가지 도전 과제는 현대 IT 환경의 운영 현실을 정의하며, 전통적인 모니터링 방식의 근본적인 한계를 드러내고 관측 가능성으로의 패러다임 전환이 필연적임을 증명합니다.

1.2 모니터링(Monitoring)에서 관측 가능성(Observability)으로의 진화

모니터링에서 관측 가능성으로의 전환은 단순한 용어의 변경이 아닙니다. 이는 앞서 살펴본 클라우드 네이티브 환경의 복잡성에 대응하기 위한 필연적인 진화입니다. 이 새로운 패러다임은 우리가 시스템의 신뢰성을 확보하고 문제를 해결하는 방식을 근본적으로 변화시키고 있습니다.

1.2.1 "무엇이 고장 났는가"에서 "왜 고장 났는가"로의 질문 변화

모니터링과 관측 가능성의 핵심적인 차이는 우리가 시스템에 던지는 질문의 깊이에 있습니다.

- 모니터링 (Monitoring): 모니터링은 사전에 정의된 메트릭(예: CPU 사용률 90% 이상)을 기반으로 시스템의 상태를 감시합니다. 이는 "무엇이 고장 났는가?(What is broken?)"라는 질문에 답을 줍니다. 즉, 우리가 이미 예측하고 있는 문제, 즉 '알려진 미지(Known Unknowns)'를 탐지하는 데 중점을 둡니다.
- 관측 가능성 (Observability): 관측 가능성은 한 단계 더 나아가, 시스템이 출력하는 모든 데이터를 탐색하여 문제의 근본 원인을 이해할 수 있는 능력을 의미합니다. 이를 통해 "왜 고장 났는가?(Why is it broken?)" 라는 질문에 답을 찾을 수 있습니다. 이는 예측하지 못했던 새로운 유형의 문제, 즉 '알려지지 않은 미지(Unknown Unknowns)'를 조사하고 해결하는 역량입니다.

1.2.2 데이터 수집 자동화와 컨텍스트(Context) 연결의 중요성

진정한 관측 가능성의 힘은 메트릭, 로그, 트레이스 데이터를 공유된 컨텍스트(Context)로 연결하는 데서 나옵니다. 예를 들어, 엔지니어는 응답 시간 지연(메트릭) 알림을 받고, 클릭 한 번으로 해당 시점의 느린 요청(트레이스)을 확인한 뒤, 그 요청을 처리한 특정 컨테이너의 오류(로그)를 즉시 조회할 수 있어야 합니다. 이러한 유기적인 데이터 연동은 문제 해결 시간을 획기적으로 단축시킵니다. 특히, 쿠버네티스와 같이 동적인 환경에서는 수동으로 데이터를 계측하는 것이 불가능하므

로, 데이터 수집의 자동화는 관측 가능성 플랫폼의 필수 전제 조건입니다. 이러한 데이터 간의 유기적인 컨텍스트 연결은 엔지니어가 가설을 세우고 검증하는 과정을 수동적 분석이 아닌 능동적 탐색으로 전환시키는 핵심 역량입니다.

1.2.3 비즈니스 연속성을 위한 실시간 장애 탐지 및 MTTR 단축 전략

관측 가능성은 명확한 비즈니스 가치를 제공합니다. 핵심 성과 지표인 MTTR(Mean Time To Repair/Resolve, 평균 해결 시간) 은 장애 발생부터 복구까지 걸리는 시간을 의미합니다. 관측 가능성 플랫폼은 "왜" 문제가 발생했는지에 대한 답을 신속하게 제공함으로써, MTTR을 극적으로 단축시킵니다. 이는 서비스 중단으로 인한 매출 손실, 브랜드 평판 하락과 같은 비즈니스 충격을 최소화하고, 궁극적으로 비즈니스 연속성을 보장하는 핵심 전략입니다. 결론적으로, 관측 가능성은 기술적 지표 개선을 넘어, 서비스 중단으로 인한 직접적인 매출 손실과 브랜드 평판 하락을 방어하는 핵심적인 비즈니스 연속성 전략입니다.

1.3 MSAP Observability의 정의와 핵심 가치

MSAP Observability는 클라우드 네이티브 환경의 복잡한 운영 난제를 해결하기 위해 특별히 설계된 통합 관측 가능성 플랫폼입니다. 이 플랫폼은 현대적인 관측 가능성의 원칙을 구현하여, 운영팀과 개발팀 모두에게 실질적인 가치를 제공합니다.

1.3.1 MSAP Observability의 아키텍처 개요 및 통합 접근 방식

MSAP Observability의 핵심 기술은 eBPF(extended Berkeley Packet Filter) 입니다. 이 혁신적인 리눅스 커널 기술은 애플리케이션 코드 수정이나 언어별 에이전트(language-specific agents)를 추가로 배포할 필요 없이, 호스트에 단일 에이전트만으로 시스템의 상세 데이터를 수집하는 "제로-인스트루먼트 관측 가능성(Zero-Instrument Observability)" 접근 방식을 가능하게 합니다. 쿠버네티스나 오픈시프트(OpenShift) 환경의 호스트에 단일 에이전트만 배포하면, 해당 노드에서 실행되는 모든 애플리케이션의 원격 측정 데이터를 자동으로 수집할 수 있습니다.

이는 Java, Python, Go 등 다양한 언어가 혼재된 폴리글랏 환경을 완벽하게 지원하여 운영 복잡성을 크게 낮춥니다.

1.3.2 단순 시각화를 넘어선 능동적 인사이트 제공 플랫폼

MSAP Observability는 수집된 데이터를 단순히 보여주는 것을 넘어, 문제 해결에 필요한 능동적인 인사이트를 제공합니다.

- 자동 서비스 토폴로지 맵 (Automatic Service Topology Map): 플랫폼이 자동으로 서비스와 그 의존 관계를 탐지하여 실시간 트래픽과 요청 흐름을 시각화합니다. 이를 통해 복잡한 마이크로서비스 아키텍처를 한눈에 파악하고 병목 지점을 직관적으로 식별할 수 있습니다.
- 운영 환경에서의 지속적인 프로파일링 (Continuous Profiling in Production): eBPF 기술을 활용하여 1% 수준의 CPU와 250MB의 메모리라는 극히 낮은 오버헤드로 실제 운영 환경에서 애플리케이션의 성능 프로파일링을 지속적으로 수행합니다. 이는 과거 성능 문제 해결을 위해 운영 환경에 부하를 줄 것을 감수해야 했던 프로파일링 작업을, 이제는 비즈니스 영향 없이 상시 수행할 수 있게 되었음을 의미합니다. 이를 통해 과거에는 운영 환경에서 파악하기 불가능했던 커널 시스템 콜 레벨까지의 성능 병목을 정밀하게 진단하고, 실행 가능한 깊이 있는 개선점을 도출할 수 있습니다.

1.3.3 운영자·개발자 간 공통 언어를 제공하는 관측 데이터 허브

MSAP Observability는 메트릭, 로그, 트레이스, 프로파일 데이터를 하나의 상관관계가 있는 플랫폼으로 통합하여 개발(Dev)팀과 운영(Ops)팀 간의 데이터 사일로를 허물어뜨립니다. 이렇게 통합되고 컨텍스트가 부여된 데이터는 두 팀이 동일한 증거를 기반으로 소통하고 협업할 수 있는 “공통 언어(common language)” 역할을 합니다. 이는 1.1.3절에서 지적한 데이터 사일로 문제를 근본적으로 해결하며, 개발팀과 운영팀이 각자의 도구에서 얻은 단편적인 데이터가 아닌, 통합된 진실의 원천(Single Source of Truth)을 기반으로 협업하도록 만듭니다.

결론적으로, MSAP Observability와 같은 eBPF 기반 플랫폼을 통해 관측 가능성으로 전환하는 것은, 예측 불가능성이 일상화된 클라우드 네이티브 시대에 안정성과 비즈니스 민첩성을 동시에 확보하기 위한 가장 현명하고 필수적인 전략적 투자입니다.

제2장. MSAP Observability 아키텍처와 Zero-instrument 수집 구조

2.1. 전체 아키텍처 구성

2.1.1. MSAP COP 내에서의 Observability 모듈 배치

MSAP Observability는 독립적으로 존재하는 모니터링 솔루션이 아니라, MSAP COP(Container Orchestration Platform)라는 더 큰 엔터프라이즈 플랫폼의 핵심 구성 요소로 설계되었습니다. 이러한 유기적 통합은 단순히 기능을 추가하는 것을 넘어, 플랫폼의 각 요소와 시너지를 창출하여 전체 시스템의 안정성과 운영 효율성을 극대화하는 전략적 중요성을 가집니다.

MSAP COP는 쿠버네티스를 기반으로 애플리케이션의 개발, 배포, 운영에 필요한 모든 기능을 통합한 플랫폼입니다. 여기에는 고성능 APM, In-Memory Data Grid 기반의 세션 클러스터링, 그리고 AI 기반의 지능형 운영을 지원하는 VibeOps와 같은 핵심 모듈이 포함됩니다. Observability 모듈은 이 모든 기능과 긴밀하게 연동됩니다. 예를 들어, APM이 수집한 트랜잭션 데이터와 Observability가 수집한 인프라 메트릭, 로그를 연계 분석하여 장애의 근본 원인을 단일 뷰에서 파악합니다. VibeOps는 이처럼 유기적으로 통합된 데이터를 활용하여, 단순히 'APM 지표 임계값 초과'와 '인프라 메트릭 이상'을 별개로 경고하는 수준을 넘어섭니다. 대신, '특정 Java 메서드의 지연 시간 증가(APM)가 해당 파드의 CPU 스로틀링(Observability)과 시간적으로 일치함'을 인과관계로 추론하여, 단편적인 경고가 아닌 근본 원인에 대한 깊이 있는 통찰과 자동화된 해결 방안을 제시합니다. 이처럼 Observability는 MSAP COP의 '신경계'와 같은 역할을 수행하며, 플랫폼 전반에 걸쳐 완전한 가시성을 완성하는 핵심 요소입니다.

2.1.2. Node Agent / Cluster Agent / 수집·저장·분석 파이프라인 구조

MSAP Observability의 데이터 수집 아키텍처는 효율성과 확장성을 극대화하기 위해 세 가지 핵심 요소로 구성됩니다. 각 요소는 명확한 역할을 수행하며, 대규모 클러스터 환경에서도 안정적으로 텔레메트리 데이터를 처리합니다.

- Node Agent: 각 쿠버네티스 워커 노드(Worker Node)에 데몬셋(DaemonSet) 형태로 배포되는 경량 에이전트입니다. 이 에이전트의 가장 큰 특징은 eBPF(extended Berkeley Packet Filter) 기술을 활용하여 리눅스 커널 레벨에서 데이터를 직접 수집한다는 점입니다. 이를 통해 애플리케이션의 코드를 수정하거나, 언어별 SDK를 추가하거나, 컨테이너 이미지를 변경하는 등의 작업 없이도 네트워크 트래픽, 시스템 리소스, 커널 이벤트 등 풍부한 데이터를 자동으로 수집할 수 있습니다. 이것이 바로 'Zero-Instrument'를 가능하게 하는 핵심 기술입니다.
- Cluster Agent: 각 노드 에이전트로부터 수집된 데이터를 취합하는 중앙 게이트웨이 역할을 수행합니다. 클러스터 에이전트는 데이터를 중앙 파이프라인으로 전송하기 전에 기본적인 필터링이나 집계 작업을 수행하여 네트워크 부하를 줄이고, 데이터 수집 정책을 클러스터 전체에 일관되게 적용하는 역할을 담당할 수 있습니다.
- 수집·저장·분석 파이프라인: 클러스터 에이전트로부터 전송된 데이터는 OpenTelemetry Collector와 같은 중앙 데이터 파이프라인 허브를 통해 처리됩니다. 이 파이프라인은 OpenTelemetry Protocol(OTLP), Prometheus 등 다양한 형식의 데이터를 수신(Receive)하고, 메타데이터 추가나 민감 정보 마스킹 같은 처리(Process) 과정을 거친 후, 최종적으로 Prometheus, Jaeger 등 목적에 맞는 여러 백엔드 시스템으로 데이터를 내보내는(Export) 표준화된 3단계 과정을 수행합니다.

2.1.3. 통합 관측성 플랫폼으로서의 역할

MSAP Observability는 클라우드 네이티브 환경에서 기술적 유연성을 확보하기 위해 데이터 수집을 OpenTelemetry(OTel) 표준과 연계합니다. 하지만 MSAP Observability는 OpenTelemetry의 세 가지 핵심 요소(Logs, Metrics, Traces)를 통합하고, 여기에 Profiling 기능까지 더하여 단일 플랫폼 내에서 통합된 관측성을 제공하는 것을 목표로 합니다.

이러한 통합 아키텍처는 사용자가 Prometheus, Jaeger, Loki와 같은 개별 오픈소스 도구들을 따로 설치하고 구성할 필요 없이, 시스템의 전반적인 상태를 한눈에 파악할 수 있도록 지원합니다. 데이터 저장 및 분석을 위해 자체적인 고성능 구성 요소를 활용하고, MSAP APM과도 통합되어 정확한 장애 원인 분석을 위한 상세 정보를 제공합니다.

MSAP Observability가 벤더 중립적인 OpenTelemetry 표준을 수용하면서도, 외부 오픈소스 솔루션에 대한 의존 없이 데이터 수집부터 저장, 분석, 시각화까지 하나의 제품으로 완성하여 운영의 복잡성을 획기적으로 낮추는 것입니다.

이러한 통합 가시성을 가능하게 하는 핵심 동력 중 하나는 바로 다음 섹션에서 자세히 다룰, 언어와 무관하게 정보를 수집하는 Zero-Instrument 데이터 수집 방식입니다.

OpenTelemetry 표준과 Zero-Instrument(eBPF) 기술을 활용한 MSAP Observability의 데이터 수집 방식에 대해 더 자세히 알아볼까요?

2.2. Zero-Instrument Observability: 무설정 자동 감지

'Zero-Instrument Observability'는 개발팀의 개입을 최소화하면서 시스템의 관측성을 즉시 확보하는 현대적인 접근 방식입니다. 클라우드 네이티브 환경에서는 하루에도 수십, 수백 개의 새로운 마이크로서비스가 배포될 수 있습니다. 이때마다 개발자가 모니터링을 위해 코드를 수정하고, 라이브러리를 추가하고, 설정을 변경해야 한다면 개발 생산성은 심각하게 저하될 수밖에 없습니다. Zero-Instrument 방식은 이러한 부담을 제거하여 개발팀이 비즈니스 로직 개발에만 집중할 수 있게 하고, 신규 서비스가 배포되는 즉시 완벽한 관측성을 확보하여 서비스 안정성을 높이는 핵심적인 비즈니스 가치를 제공합니다.

2.2.1. 애플리케이션 코드 수정 없는 모니터링 환경 구성

Zero-Instrument 방식의 핵심 기술은 eBPF(extended Berkeley Packet Filter)입니다. eBPF는 리눅스 커널 내에서 안전하게 코드를 실행할 수 있게 해주는 혁신적인 기술로, 이를 통해 애플리케이션 코드나 설정을 전혀 변경하지 않고도 시스템의 내부 동작을 깊이 있게 관찰할 수 있습니다.

MSAP Observability의 Node Agent는 eBPF를 사용하여 커널 레벨에서 발생하는 시스템 콜(System Call)을 직접 후킹(hooking)합니다. 애플리케이션이 네트워크 통신을 하거나, 파일을 읽고 쓰거나, 새로운 프로세스를 생성할 때 발생하는 모든 활동을 커널 단에서 직접 감지하여 데이터를 수집합니다. 이는 기존의 방식, 즉 각 프로그래밍 언어별로 제공되는 SDK를 코드에 삽입하거나 APM 에이전트를 JVM 등에 연결해야 했던 방식과 근본적으로 다릅니다. 커널에서 직접 데이터를 수집하기 때문에 애플리케이션이 어떤 언어로 작성되었는지, 어떤 프레임워크를 사용하는지에 관계없이 일관된 모니터링이 가능합니다. 이는 개발팀을 '계측 세금(instrumentation tax)'으로부터 해방시키는 패러다임의 전환입니다. 더 이상 새로운 언어 버전이나 프레임워크 도입 시 모니터링 에이전트의 호환성을 검증하느라 배포가 지연되는 운영상의 마찰이 사라집니다.

2.2.2. 쿠버네티스 파드 배포 시 에이전트 자동 부착 및 설정 자동화

쿠버네티스 환경에서 Zero-Instrument 방식은 운영 자동화 측면에서 강력한 이점을 제공합니다. 개발팀이 새로운 애플리케이션을 담은 컨테이너 이미지를 빌드하여 쿠버네티스 클러스터에 파드(Pod)로 배포하면, 해당 파드가 스케줄링된 노드에 이미 실행 중인 MSAP Node Agent가 새로운 파드의 활동을 자동으로 감지하고 즉시 모니터링을 시작합니다.

이 과정에서 운영팀이나 개발팀의 추가적인 개입은 필요하지 않습니다. 에이전트를 수동으로 설치하거나, 환경 변수를 설정하거나, 별도의 구성 파일을 배포할 필요가 없습니다. 모든 것이 자동으로 이루어지기 때문에 휴먼 에러의 가능성이 사라지고, 모든 서비스에 일관된 모니터링 정책을 적용할 수 있습니다. 이는 IT 의사결정자에게 '모든 서비스는 배포 즉시 관측성을 확보해야 한다'는 거버넌스 정책을 코드가 아닌 플랫폼 레벨에서 강제할 수 있는 강력한 수단을 제공하며, 결과적으로 운영 비용 절감과 서비스 안정성이라는 두 마리 토끼를 모두 잡는 전략적 선택이 됩니다.

2.2.3. Java, Go, Python, Node.js 등 다양한 런타임 자동 감지 프로세스

eBPF 기반 접근 방식의 가장 큰 강점 중 하나는 특정 프로그래밍 언어나 런타임에 종속되지 않는다는 것입니다. 데이터 수집이 애플리케이션 코드가 아닌 운영체제 커널 레벨에서 이루어지기 때문에, 애플리케이션이 어떤 기술 스택으로 구현되었는지와 무관하게 동작을 관찰할 수 있습니다.

MSAP Observability는 Java, Golang, Python, Node.js, .NET, C, Ruby, PHP 등 오늘날 마이크로서비스 환경에서 사용되는 거의 모든 주요 언어와 런타임을 자동으로 감지하고 지원합

니다. 이는 여러 팀이 각기 다른 언어와 기술을 사용하여 다양한 마이크로서비스를 개발하는 폴리글랏(Polyglot) 환경에서 특히 중요합니다. Zero-Instrument 방식은 이처럼 다양한 기술 스택으로 구성된 복잡한 시스템 전체를 일관된 기준으로 통합 모니터링할 수 있게 해주는 핵심 요소입니다.

2.2.4. 운영체제·컨테이너 런타임 레벨의 메트릭·이벤트 수집

Zero-Instrument 방식을 통해 애플리케이션 코드 레벨에서는 확인하기 어려운 깊이 있는 시스템 레벨의 데이터를 풍부하게 수집할 수 있습니다. 이렇게 수집된 데이터는 애플리케이션의 성능 문제를 진단하고 근본 원인을 파악하는 데 결정적인 단서를 제공합니다.

- 네트워크 데이터: 서비스 간 통신의 건전성을 파악하기 위한 핵심 지표들을 수집합니다.
  - TCP 연결 성공/실패, 재전송 횟수, 연결 지연 시간
  - 네트워크 라운드 트립 시간(RTT)
  - DNS 요청/응답 시간 및 오류
  - 수신(inbound)/송신(outbound) 트래픽 양
- 시스템 리소스: 컨테이너와 파드의 리소스 사용 현황을 정밀하게 측정합니다.
  - 컨테이너별 CPU 사용량, 대기 시간(delay), 스로틀링(throttling) 시간
  - 컨테이너별 메모리 사용량(RSS), 페이지 캐시(Cache)
  - 디스크 I/O 부하 및 사용량
- 커널 레벨 이벤트: 시스템 내부의 깊은 동작을 이해할 수 있는 데이터를 수집합니다.
  - 시스템 콜(System Call) 추적
  - 파일 접근 및 입출력 이벤트
  - 프로세스 생성 및 종료 이벤트

이처럼 코드 수정 없이 자동으로 수집된 방대한 데이터는 시스템의 상태를 전례 없이 투명하게 보여줍니다. 그렇다면 이 막대한 양의 데이터를 어떻게 효율적으로 처리하고 저장하여 유의미한 분석을 수행할 수 있을까요? 다음 섹션에서는 이 질문에 대한 해답인 데이터 수집 파이프라인에 대해 자세히 알아보겠습니다.

2.3. Data Ingestion 파이프라인

대규모 클러스터 환경에서는 초당 수백만 개의 텔레메트리 데이터(로그, 메트릭, 트레이스)가 쏟아져 나옵니다. 이 방대한 데이터를 안정적으로 수집, 처리하여 분석 시스템으로 전달하는 데이터 수집(Ingestion) 파이프라인의 역할은 관측성 플랫폼의 성패를 좌우할 만큼 중요합니다. 데이터의 종류별 특성에 맞춰 최적화된 수집 및 처리 전략을 적용하는 것은 분석 성능을 보장하고, 장기적인 데이터 저장 비용을 최적화하는 데 핵심적인 역할을 합니다.

2.3.1. 시계열 메트릭 수집 및 고압축 저장 구조

시스템 전반의 CPU 사용률, 메모리 점유율, 요청 처리량과 같은 메트릭은 서비스의 상태 변화를 시간 축에서 정밀하게 파악하는 데 필수적인 데이터입니다. 이 데이터는 시간(Time)과 값(Value)을 축으로 구성되는 전형적인 시계열(Time-series) 형태이기 때문에, 이를 효율적으로 관리하려면 전용 구조의 수집 엔진과 고압축 저장 포맷이 필요합니다.

MSAP Observability는 자체 시계열 메트릭 파이프라인을 사용합니다. 이 파이프라인은 컨테이너·노드·애플리케이션 레이어에서 발생하는 메트릭을 실시간으로 수집하여, 병렬 처리 가능한 내부 스트림 구조를 통해 압축 저장소로 전달합니다.

이 방식은 외부 오픈소스 구성 요소의 제약 없이 MSAP 플랫폼 아키텍처에 최적화된 메트릭 처리 성능과 확장성을 제공합니다.

대규모 MSA 환경에서는 메트릭 생성량이 기하급수적으로 증가하기 때문에, 장기 보관 시 스토리지 효율성이 핵심 요건이 됩니다. MSAP Observability의 시계열 스토리지 엔진은 반복 패턴을 효과적으로 제거하는 고압축 알고리즘을 적용해 일반적인 RDBMS 대비 훨씬 적은 공간에 데이터를 보관할 수 있습니다.

이러한 구조는 수개월~수년 단위의 성능 추이를 비교하거나, 안정성 분석·용량 계획(Capacity Planning)·장애 전조 탐지 등의 업무를 수행할 때 비용 부담을 크게 낮추면서도 고해상도의 데이터를 활용할 수 있게 해줍니다.

2.3.2. 로그 구조화·패턴 인식·색인 전략

로그는 시스템에서 발생한 이벤트에 대한 상세한 기록이지만, 대부분 비정형 텍스트 형태로 생성되어 분석이 어렵습니다. MSAP Observability 파이프라인은 수집된 비정형 로그를 JSON과 같은 구조화된 형식으로 자동 변환합니다. 또한, 반복적으로 나타나는 로그 메시지의 패턴을 자동으로 인식하고 그룹화하여, 수백만 개의 개별 로그 라인을 몇 개의 의미 있는 패턴으로 요약해줍니다. 이를 통해 운영자는 노이즈를 줄이고 중요한 오류 패턴에 집중할 수 있습니다.

2.3.3. 분산 Trace 데이터의 Span 기반 저장·조회 모델

마이크로서비스 아키텍처(MSA)에서 사용자 요청 하나는 여러 서비스를 거치며 처리됩니다. 분산 추적(Distributed Tracing)은 이 전체 여정을 추적하는 기술이며, 그 기본 단위를 스팬(Span) 이라고 합니다. 스팬은 특정 서비스 내에서 수행된 작업 단위(예: API 호출, DB 쿼리)를 나타내며, 시작 시간과 종료 시간, 그리고 작업 관련 메타데이터를 포함합니다. 이러한 스팬들이 인과 관계(부모-자식)를 가진 트리 구조로 연결된 것이 바로 트레이스(Trace) 입니다.

OpenTelemetry 표준에 따라 생성된 트레이스 데이터는 스팬 단위로 저장소에 기록됩니다. 이러한 스팬 기반 저장 및 조회 모델은 MSA 환경의 복잡한 호출 관계를 분석하는 데 매우 효과적입니다. 특정 요청의 전체 경로를 시각화하여 어느 서비스에서 병목이 발생하는지 쉽게 파악할 수 있으며, 특정 서비스가 어떤 다른 서비스들을 호출하는지 의존성 분석도 가능합니다. 또한, 오류가 발생한 스팬을 중심으로 해당 오류가 어떤 상위 요청에서 시작되었고, 어떤 하위 서비스에 영향을 미쳤는지 신속하게 추적할 수 있습니다.

2.3.4. 대규모 클러스터를 위한 수집·저장·조회 성능 확장 방식

단일 서버 기반의 관측성 솔루션은 소규모 환경에서는 유용할 수 있지만, 수백, 수천 개의 노드로 구성된 대규모 클러스터 환경에서는 명백한 한계에 직면합니다. 예를 들어, 단일 파일 기반의 데이터베이스(예: SQLite)를 사용하는 백엔드는 데이터 이중화가 불가능하여 장애 발생 시 데이터 유실의 위험이 크고, 유입되는 데이터 양이 증가하면 쉽게 성능 병목에 도달합니다. 이는 파일럿 프로젝트나 소규모 환경에서는 용납될 수 있으나, 프로덕션급 대규모 클러스터에서는 데이터 유실이라는 치명적인 비즈니스 리스크를 내포하며, SRE(Site Reliability Engineering) 관점에서 용

납할 수 없는 단일 장애점(SPOF)이 됩니다.

MSAP Observability는 이러한 한계를 극복하기 위해 설계 초기부터 수평적 확장(Scale-out)이 가능하도록 설계되었습니다. 데이터를 수집하는 Node Agent부터 중앙 파이프라인인 Collector, 그리고 데이터를 최종적으로 저장하고 조회하는 백엔드 저장소에 이르기까지 모든 구성 요소를 필요에 따라 여러 인스턴스로 분산하여 처리 용량을 선형적으로 늘릴 수 있습니다. 이를 통해 클러스터의 규모가 커지더라도 안정적인 데이터 수집과 빠른 조회 성능을 일관되게 유지할 수 있습니다.

제3장. 통합 APM: MSAP Observability 기반 실시간 트러블슈팅

MSAP Observability 는 전통적인 APM(Application Performance Management)의 심층 분석 능력과 최신 분산 추적(Distributed Tracing) 기술을 유기적으로 결합한 통합 관측성 플랫폼을 제시합니다. 본 솔루션은 단순히 시스템의 이상 징후를 감지하는 것을 넘어, “왜” 문제가 발생했는지에 대한 명확하고 실행 가능한 답을 제공하는 데 중점을 둡니다.

본 장에서는 MSAP Observability가 어떻게 복잡하게 얽힌 서비스 간의 호출 흐름을 완벽하게 추적하고, 데이터를 기반으로 성능 병목 구간을 자동으로 식별하며, 궁극적으로 장애 평균 해결 시간(MTTR)을 획기적으로 단축시키는지 그 핵심 원리와 구체적인 활용 방안을 심층적으로 분석할 것입니다. 이를 통해 기업이 어떻게 예측 불가능한 장애 상황에서도 서비스 안정성을 확보하고 비즈니스 연속성을 유지할 수 있는지에 대한 전략적 청사진을 제시합니다.

3.1. MSAP APM 통합 구조: 단일 트랜잭션의 완벽한 가시성 확보

마이크로서비스 아키텍처(MSA) 환경에서 사용자의 단 한 번의 클릭은 시스템 내부에서 수많은 서비스 간의 연쇄적인 호출을 유발합니다. 이러한 복잡한 여정 속에서 어느 한 구간의 작은 지연이 전체 서비스의 성능 저하로 이어질 수 있기 때문에, 요청의 시작점부터 최종 종단점까지 전체 흐름을 추적하는 엔드투엔드(End-to-End) 가시성 확보는 필수적입니다. MSAP APM의 통합 아키

텍처는 운영자에게 익숙한 전통적인 3-Tier(Web-WAS-DB) 분석 모델의 직관성을 유지하면서도, 복잡한 분산 시스템의 숨겨진 병목 구간을 명확하게 식별할 수 있는 강력한 토대를 마련합니다. 이는 문제 해결의 시작점을 추측이 아닌 데이터로 전환시키는 전략적 가치를 제공합니다.

3.1.1. 실시간 트랜잭션 추적과 비즈니스 로직 병목 구간 탐지

MSAP Observability는 시스템에서 처리되는 모든 개별 트랜잭션을 실시간으로 추적하여 성능 문제를 분자 단위까지 분석합니다. 사용자 요청이 시스템에 유입되어 각종 비즈니스 로직을 거쳐 데이터베이스와 연동되고, 최종적으로 응답을 반환하기까지의 전 과정은 시각적인 'Call Tree' 형태로 상세하게 제공됩니다.

이 Call Tree는 어떤 클래스의 어떤 메서드가 호출되었는지, 각 단계에서 얼마의 시간이 소요되었는지를 명확하게 보여줍니다. 이를 통해 개발자와 운영자는 더 이상 "어디가 문제일까?"라는 막연한 추측을 할 필요가 없습니다. Call Tree는 전체 수행 시간 중 비정상적으로 많은 시간을 소요한 정확한 메서드 호출(method invocation) 또는 코드 경로(code path)를 특정하여, 관련 없는 시스템 구성요소로부터 근본 원인을 분리하고 개발자를 위한 문제 공간(problem space)을 획기적으로 축소시킵니다. 이처럼 데이터에 기반한 정밀한 분석은 성능 병목 지점을 정확히 찾아내고, 최적화 노력을 가장 효과적인 곳에 집중할 수 있도록 지원합니다.

3.1.2. 3-Tier(Web-WAS-DB) 연계 분석을 통한 구간별 응답시간 측정

복잡한 시스템 장애의 첫 번째 과제는 문제의 원인이 애플리케이션 로직에 있는지, 인프라에 있는지, 아니면 데이터베이스에 있는지를 신속하게 분리하는 것입니다. MSAP Observability는 사용자의 단일 요청에 대해 Web, WAS, DB 각 계층을 하나의 통합된 트랜잭션으로 연계하여 분석하는 강력한 기능을 제공합니다.

MSAP Observability는 전통적 3-Tier 환경(Web-WAS-DB)과 MSA 기반 내부 서비스 호출을 동일한 시간축(Timeline)에서 분석하도록 설계되었습니다.

계층	수집 방식	주요 분석항목
Web	OpenTelemetry Span	요청량, 응답코드, 지연 구간
WAS(Java)	OPENMARU APM Agent	메소드별 실행시간, SQL 구간, 내부 처리 시간

DB(MySQL/PostgreSQL 등)	eBPF 기반 네트워크 + DB 프로토콜 감지	Slow Query, RTT, 재시도, 패킷 지연

구성 요소별 분석 방식은 다음과 같습니다.

연계 분석 방식

1. APM Trace ID를 MSAP Observability의 Trace ID로 자동 전파
2. ClickHouse 기반 통합 저장소에서 메트릭·로그·분산 Trace를 조인
3. Web → WAS → DB까지 전체 호출 Graph를 하나의 Trace Stack으로 제공

이를 통해 다음 질문에 답할 수 있습니다.

- “응답시간 2초 증가의 원인이 WAS 처리 때문인지, DB 때문인지?”
- “커넥션 풀 고갈로 인해 대기시간이 늘어났는지?”
- “API 응답 지연이 네트워크 측 지연인지 DB Lock 때문인지?”

이러한 구간별 분석을 통해 운영자는 즉각적으로 장애의 책임소재를 분리하여(instantly isolate fault domains), 웹/네트워크팀과 애플리케이션/DB팀 간의 불필요한 책임 공방과 시간 낭비를 방지합니다. 이는 문제의 범위를 특정 계층으로 신속하게 좁히고, 더 심층적인 원인 분석으로 나아갈 수 있게 하여 문제 해결 속도를 극대화합니다.

3.1.3. 스냅샷·덤프 분석을 통한 예외 및 에러 상세 추적

시스템에서 예외(Exception)나 에러가 발생하는 순간은 장애의 근본 원인을 파악할 수 있는 가장 결정적인 단서를 포함하고 있습니다. MSAP Observability는 단순한 에러 발생 알림을 넘어, 문제 발생 시점의 상황을 완벽하게 재구성할 수 있는 핵심적인 법의학적 데이터(forensic data)를 자동으로 수집합니다.

에러가 발생하면, 시스템은 해당 시점의 스냅샷 정보와 함께 스택 트레이스(Stack Trace)를 자동으로 확보합니다. 이 스택 트레이스는 어떤 코드의 몇 번째 라인에서, 어떤 종류의 에러(예: `NullPointerException`, `SQLException`)가 발생했는지를 명확한 증거로 제시합니다. 이것이 법의학적 데이터로 불리는 이유는 실패 순간의 시스템 상태에 대한 불변의, 고충실도 기록

(immutable, high-fidelity record)을 제공하며, 이는 사후 분석(post-mortem)과 재발 방지에 필수적이기 때문입니다. 이처럼 명확한 증거 기반의 근본 원인 분석(RCA)은 장애 재발 방지를 위한 가장 확실하고 효과적인 전략입니다.

3-Tier 구조 분석이 시스템의 기본 골격을 이해하는 데 필수적이지만, 수많은 사용자가 경험하는 서비스의 품질을 정밀하게 관리하고 잠재적인 문제를 사전에 감지하기 위해서는 개별 트랜잭션 분석을 넘어 통계적 분석과 자동화된 패턴 감지가 반드시 필요합니다.

3.2. 애플리케이션 성능·품질 모니터링: 데이터를 통한 사용자 경험 관리

현대 애플리케이션 운영에서 '평균 응답 시간'과 같은 단일 지표는 종종 현실을 왜곡하는 함정이 될 수 있습니다. 대부분의 사용자는 빠른 응답을 경험하더라도, 소수의 사용자가 겪는 극심한 지연이 평균값에 가려져 실제 사용자 경험의 문제를 놓치게 만들기 때문입니다. 따라서 서비스 수준 목표(SLO)를 효과적으로 정의하고 관리하기 위해서는, P99와 같은 백분위수 기반의 성능 분석과 오류 패턴의 자동 감지를 통해 잠재적인 위험 신호를 사전에 포착하는 것이 전략적으로 매우 중요합니다.

3.2.1. P50·P95·P99 응답시간 기반 성능 분석

백분위수(Percentile) 응답 시간은 전체 사용자 경험의 분포를 이해하고, 단순 평균값이 놓치는 성능 문제를 발견하는 데 필수적인 지표입니다. MSAP Observability는 다음과 같은 핵심 백분위수 지표를 제공하여 IT 의사결정자가 비즈니스 관점의 인사이트를 얻을 수 있도록 지원합니다.

- P50 (Median): 중앙값을 의미하며, 전체 요청의 50%가 이 시간보다 빠르게 처리되었음을 나타냅니다. 이는 대부분의 일반적인 사용자가 경험하는 평균적인 성능 수준을 보여주는 지표로, 시스템의 보편적인 성능을 판단하는 기준이 됩니다.
- P95 (95th Percentile): 전체 요청의 95%가 이 시간보다 빠르게 처리되었음을 의미합니다. 즉, 상위 5%의 사용자는 이보다 느린 경험을 하고 있다는 뜻입니다. 이는 일부 사용자들이 경험하기 시작하는 성능 저하 지점을 나타내며, 서비스 품질에 민감한 사용자들의 불만을 야기할 수 있는 잠재적 문제 영역을 시사합니다.
- P99 (99th Percentile): 전체 요청 중 단 1%의 사용자가 경험하는 최악의 응답 시간을 나타냅니다. 이 지표는 시스템이 감당할 수 있는 잠재적인 성능 한계와 심각한 문제를 드러내

는 중요한 신호입니다. P99 값이 급증한다면, 이는 특정 사용자 그룹이 극심한 서비스 지연을 겪고 있으며, 시스템 어딘가에 심각한 병목 현상이나 결함이 존재할 가능성이 높다는 것을 의미합니다.

3.2.2. 오류율·재시작 횟수·에러 패턴 자동 감지

애플리케이션의 품질은 단순 응답시간뿐 아니라 실패율, 재시작 패턴, 로그 패턴으로 판단해야 합니다. MSAP Observability는 다음과 같은 핵심 지표를 자동으로 감지하고 추적하여 서비스 품질 저하를 조기에 경고합니다.

- 오류율(Error Rate): 전체 요청 중 실패한 요청의 비율을 실시간으로 추적합니다. 오류율의 증가는 서비스의 신뢰도가 하락하고 있음을 의미하며, 사용자 경험에 직접적인 악영향을 미치는 핵심 품질 지표입니다.
- 재시작 횟수(Restart Count): 컨테이너(Pod) 환경에서 비정상적인 재시작 횟수를 감지합니다. 잦은 재시작은 메모리 누수, 설정 오류, 리소스 부족 등 애플리케이션 내부에 심각한 불안정 요인이 존재함을 나타내는 명백한 신호입니다.
- 에러 패턴 자동 감지: 수많은 로그 메시지 속에서 반복적으로 발생하는 특정 유형의 에러 패턴을 자동으로 식별하고 그룹화합니다. 이를 통해 운영자는 산발적인 단일 에러가 아닌, 시스템 전반에 영향을 미치는 잠재적인 버그나 구조적 결함을 예측하고 선제적으로 대응할 수 있습니다.

이 방식은 기존의 “로그를 사람이 직접 검색”하는 방식에서

→ “주요 패턴을 자동 분류 후, 해당 패턴과 연관된 Trace로 이동”하는 방식으로 진화한 구조입니다.

3.2.3. 로그 패턴과 트랜잭션 간 상관 분석(Correlation)

관측성(Observability)의 핵심 가치는 흩어져 있는 데이터들을 연결하여 의미 있는 인사이트를 도출하는 ’상관 분석(Correlation)’에 있습니다. 이것이 바로 관측성의 세 가지 기둥(로그, 메트릭, 트레이스)이 실제 문제 해결 과정에서 어떻게 상호작용하는 지를 보여주는 대표적인 사례입니다.

Correlation 기능 예시

- 특정 에러 패턴의 로그 → 해당 에러가 포함된 Trace 목록 → Bottleneck 위치 → 호출한 상위 서비스 식별
- 특정 Trace ID → 관련 로그 자동 정렬 → Stack Trace → 오류 원인 구간 파악

이는 Debugging 시간을 획기적으로 단축합니다.

특정 메트릭(Metric)의 이상(예: P99 응답 시간 급증)이 감지되면, 운영자는 해당 시점의 트레이스(Trace)를 분석하여 병목 구간을 찾고, 마지막으로 트레이스와 연관된 로그(Log)에서 'DB Connection Timeout'과 같은 구체적인 컨텍스트를 확인하여 근본 원인을 확정합니다.

MSAP Observability는 이 과정을 자동화하여, 성능이 저하된 트랜잭션의 상세 분석 화면에서 해당 시점에 발생한 특정 에러 로그 패턴을 함께 제시합니다.

이 기능은 분석의 초점을 "무슨 일이 일어났는가(성능 저하)" 에서 "왜 일어났는가(DB 연결 문제)" 로 자연스럽게 심화시킵니다. 이를 통해 운영자는 여러 시스템을 오가며 데이터를 수동으로 조합할 필요 없이, 단일 화면에서 원인과 결과를 동시에 파악하여 문제 해결 시간을 획기적으로 단축할 수 있습니다.

개별 애플리케이션의 성능과 품질을 정밀하게 모니터링하는 것을 넘어, 여러 마이크로서비스가 복잡하게 얽혀있는 현대적 환경에서는 단일 요청이 시스템 전체를 여행하는 여정을 추적하는 분산 트레이싱이 필수적입니다.

3.3. 분산 트레이싱과 장애 원인 분석: 복잡성 속에서 명확성 찾기

마이크로서비스 환경에서 단일 사용자 요청은 때로는 수십 개의 독립적인 서비스를 거치며 처리됩니다. 이러한 분산 구조에서 장애의 근원지를 추적하는 것은 '건초더미에서 바늘 찾기'와 같이 어려운 과제입니다. 이러한 복잡성을 해결하기 위한 핵심 기술이 바로 '분산 트레이싱(Distributed Tracing)' 입니다.

MSAP Observability는 OpenTelemetry를 기반으로 이러한 분산 추적을 구현합니다. OpenTelemetry는 특정 벤더에 종속되지 않는 개방형 산업 표준으로, CNCF(Cloud Native Computing Foundation)의 최상위 등급인 Graduated 프로젝트입니다. 이는 OpenTelemetry가 쿠버네티스(Kubernetes)와 동일한 수준의 성숙도와 안정성을 인정받았으며, 클라우드 네이티브 환경에서 관측성 데이터 수집의 사실상 표준(de-facto standard)임을 의미합니다. 이 표준을 기반으로, MSAP Observability는 사용자 요청의 전체 경로를 추적하여 복잡성 속에 가려진 장애

의 근본 원인을 명확하게 찾아냅니다.

3.3.1. 이기종 마이크로서비스 간 호출 흐름 시각화

MSAP Observability는 '서비스 토폴로지 맵(Service Topology Map)' 기능을 통해 시스템의 전체 구조와 서비스 간의 상호작용을 한눈에 파악할 수 있도록 지원합니다. 이 기능은 Java, Python, Go 등 서로 다른 기술 스택으로 구성된 이기종 마이크로서비스 간의 모든 실시간 호출 관계와 의존성을 자동으로 탐지하여 시각적인 맵 형태로 보여줍니다.

이러한 자동 탐지는 eBPF와 같은 커널 레벨 기술을 통해 애플리케이션 코드 수정 없이 네트워크 트래픽을 분석하는 'Zero-Instrument Observability'를 기반으로 하기에 가능합니다. 이를 통해 HTTP, gRPC, PostgreSQL, MySQL, Redis, MongoDB 등 다양한 프로토콜을 자동으로 감지하고 지원합니다.

장애가 발생했을 때, 운영자는 이 맵을 통해 어떤 서비스에서 지연이나 에러가 발생하고 있는지(문제 지점), 그리고 그 영향이 어떤 다른 서비스로 전파되고 있는지(영향 범위)를 직관적으로 파악할 수 있습니다. 이는 문제의 근원지를 신속하게 좁혀나가고, 장애 대응의 우선순위를 정하는 데 결정적인 도움을 줍니다.

3.3.2. Trace ID 기반 메트릭·로그 자동 연계

분산 트레이싱은 'Trace ID'와 'Context Propagation' 이라는 두 가지 핵심 원리로 동작합니다. 사용자 요청이 시스템의 첫 관문에 도달하면, 해당 요청을 식별하기 위한 고유한 `Trace ID`가 생성됩니다. 이후 이 요청이 다른 마이크로서비스를 호출할 때마다, 이 `Trace ID`가 담긴 컨텍스트 정보가 W3C Trace Context 표준인 `traceparent` HTTP 헤더 등을 통해 다음 서비스로 계속해서 전달(전파)됩니다.

MSAP Observability는 이 `Trace ID`를 키(Key)로 활용하여, 특정 분산 트랜잭션과 관련된 모든 텔레메트리 데이터를 자동으로 연결합니다. 예를 들어, `Trace ID: abc-123`을 가진 요청이 A, B, C 세 개의 서비스를 순차적으로 거쳤다고 가정해 봅시다. MSAP Observability는 이 `Trace ID`를 통해 다음 정보들을 하나의 화면에서 통합하여 제공합니다.

- 관련 메트릭: 서비스 A, B, C 각각에서의 처리 지연 시간

- 관련 로그: 서비스 B에서 발생한 특정 `WARN` 레벨 로그, 서비스 C에서 발생한 `ERROR` 로그 및 스택 트레이스

이를 통해 운영자는 개별 데이터를 따로 조회하고 연관성을 추측할 필요 없이, 하나의 트랜잭션과 관련된 모든 정보를 유기적으로 분석하여 문제의 전체적인 맥락을 완벽하게 이해할 수 있습니다.

3.3.3. 느린 쿼리와 애플리케이션 지연 간 상관관계 분석

분산 트레이싱을 통한 상관관계 분석은 막연한 성능 문제를 구체적인 원인으로 귀결시키는 강력한 도구입니다. 예를 들어, 사용자가 '주문 내역 조회' API 호출이 매우 느리다고 보고한 상황을 가정해 보겠습니다. MSAP Observability를 통한 문제 해결 과정은 다음과 같습니다.

1. 느린 트랜잭션 식별: 운영자는 해당 API 호출과 관련된 분산 트레이스를 `Trace ID`를 기반으로 검색하여 특정 요청의 전체 처리 과정을 확인합니다.
2. 병목 구간 탐지: 트레이스의 시각화된 흐름(Waterfall view)을 통해, 전체 지연 시간의 대부분이 '재고-서비스(inventory-service)'에서 발생했음을 한눈에 파악합니다.
3. 근본 원인 드릴다운: '재고-서비스'의 스팬(span)을 상세 분석하여, 해당 서비스가 '상품-DB(product-db)'를 호출하는 과정에서 지연이 발생했음을 확인합니다.
4. Slow Query 확정: 최종적으로, 해당 DB 호출과 관련된 상세 정보에서 특정 SQL 쿼리(Slow Query)가 비효율적인 `JOIN` 연산으로 인해 5초 이상 소요된 것이 전체 지연의 근본 원인이었음을 정확히 찾아냅니다.

이처럼 분산 트레이싱은 "API가 느리다"는 모호한 문제에서 출발하여 "특정 DB의 특정 SQL 쿼리가 문제다"라는 실행 가능한(actionable) 결론으로 분석을 이끌어갑니다.

3.3.4. 장애 상황에 특화된 대시보드 및 알람 체계

수집되고 분석된 데이터를 실제 운영에 효과적으로 활용하기 위해서는, 정보를 적시에 올바른 담당자에게 전달하는 체계가 중요합니다. MSAP Observability는 장애 대응(Incident Response) 시나리오에 맞춰 사전 구성된 장애 분석 전용 대시보드를 제공합니다. 이를 통해 SRE(사이트 신

뢰성 엔지니어) 및 운영팀은 장애 발생 시 여러 화면을 헤맬 필요 없이, SLO 위반 현황, 오류율 급증 서비스, 지연 시간 상위 트랜잭션 등 문제 해결에 필요한 핵심 정보를 즉시 확인할 수 있습니다.

- 가용성 SLO 위반
- 응답시간 SLO 위반
- 로그 ERROR 패턴 급증
- 재시작 횟수 증가
- DB RTT 증가
- 네트워크 재전송·패킷 손실 증가

대시보드에 자동으로 Incident로 표시되며, Slack, Webhook 등 외부 채널과 연계하여 운영자의 대응 시간을 단축합니다.

또한, MSAP Observability의 알람 체계는 단순 임계값 기반 알람이 야기하는 과도한 노이즈와 알람 피로도(alert fatigue) 문제를 해결하는 지능형 알람을 제공합니다. 여러 지표 간의 상관관계를 분석하여 비정상적인 패턴을 감지하고, 예를 들어 '오류율(Error Rate)이 5% 증가하고, 동시에 P99 응답 시간이 2초를 초과하며, 특정 에러 로그 패턴이 급증하는' 복합적인 조건을 만족할 때만 알람을 발생시킵니다. 이러한 접근은 모든 알람이 '실행 가능한(actionable)' 정보가 되도록 보장하며, 운영팀이 정말 중요한 문제에만 집중하여 신속하게 대응할 수 있도록 지원합니다.

결론적으로, 본 장에서 다룬 세 가지 핵심 역량—기본적인 3-Tier 분석, 통계 기반 품질 관리, 그리고 분산 트레이싱—은 개별적인 기능이 아닌 통합된 관측성 전략의 필수 구성 요소입니다. 이들은 서로 유기적으로 결합하여, 기업의 운영 패러다임을 문제 발생 후 대응하는 수동적인 '장애 대응(firefighting)'에서, 데이터를 기반으로 문제를 예측하고 사전에 방지하는 능동적이고 예측적인 모델로 전환시키는 강력한 동력을 제공합니다.

제4장. Continuous Profiling: 운영 환경 메서드 레벨 성능 최적화

클라우드 네이티브 환경으로의 전환은 기업에 전례 없는 민첩성과 확장성을 제공했지만, 동시에 마이크로서비스, 컨테이너, 서버리스 아키텍처가 얽힌 극도의 복잡성을 야기했습니다. 사용자의 단 한 번의 클릭이 내부적으로는 수십, 수백 개의 서비스를 거치며 처리되는 오늘날의 분산 시스템에서, 성능 저하의 근본 원인을 찾는 것은 '건초더미에서 바늘 찾기'와 같이 어려운 과제가 되었습니다. CPU 사용률이나 응답 시간 같은 전통적인 모니터링 지표는 문제가 발생했음을 알려줄 수는 있지만, 코드의 어느 메서드가, 왜 병목을 일으키는지에 대한 심층적인 답변을 제공하지는 못합니다.

이러한 한계를 극복하기 위해 등장한 Continuous Profiling(상시 프로파일링) 은 운영 중인 서비스의 성능을 실시간으로, 그리고 코드 레벨까지 깊이 있게 분석하는 핵심 기술입니다. 과거 프로파일링은 높은 오버헤드로 인해 개발 환경이나 제한된 테스트 환경에서만 사용 가능했지만, 이제는 혁신적인 기술을 통해 운영 중인 서비스에 미치는 영향을 최소화하면서도 시스템의 모든 활동을 지속적으로 관찰할 수 있게 되었습니다.

본 장에서는 Continuous Profiling이 어떻게 현대 IT 시스템의 복잡성을 해결하고, 개발자와 운영자에게 코드 수준의 명확한 가시성을 제공하여 데이터 기반의 성능 최적화를 가능하게 하는지 심층적으로 분석합니다. 이는 단순히 기술적인 문제를 해결하는 도구를 넘어, 안정적인 서비스 운영과 비즈니스 연속성을 확보하기 위한 필수적인 전략 자산으로서의 가치를 증명할 것입니다.

4.1 올웨이즈 온(Always-On) 프로파일링 아키텍처

전통적인 프로파일링은 특정 성능 문제를 진단해야 할 때 일시적으로 활성화하는 '이벤트'성 작업이었습니다. 그러나 예측 불가능한 장애가 빈번한 현대적인 운영 환경에서는 문제가 발생한 후에 프로파일러를 켜는 것은 이미 늦은 대응입니다. 올웨이즈 온(Always-On) 프로파일링은 이러한 패러다임을 전환하여, 시스템에 항상 상주하며 중단 없이 성능 데이터를 수집하는 접근 방식입니다. 이는 마치 항공기의 블랙박스처럼, 시스템의 모든 순간을 기록하여 장애 발생 시 과거 시점으

로 돌아가 문제의 근본 원인을 코드 레벨에서 정밀하게 분석할 수 있는 능력을 부여합니다. 이러한 상시 데이터 수집 방식은 간헐적으로 발생하여 재현하기 어려운 문제를 해결하는 데 결정적인 역할을 하며, 안정적인 서비스 운영을 위한 필수적인 전략입니다.

4.1.1 운영환경 오버헤드를 최소화한 상시 프로파일링 기법

MSAP Observability는 운영 환경에서의 상시 프로파일링을 현실화하기 위해 eBPF(extended Berkeley Packet Filter)라는 혁신적인 기술을 채택했습니다. eBPF는 애플리케이션 코드나 설정을 변경하지 않고도 리눅스 커널 레벨에서 안전하게 코드를 실행하여 시스템의 상세한 데이터를 수집할 수 있게 해주는 강력한 기술입니다. 이를 통해 과거에는 불가능했던 낮은 오버헤드의 상시 프로파일링이 가능해졌습니다.

- eBPF 기술 활용
  - eBPF는 리눅스 커널의 이벤트(예: 시스템 콜, 네트워크 패킷 수신)에 '훅(Hook)'을 걸어, 특정 이벤트가 발생할 때마다 미리 정의된 코드를 실행합니다. 이 방식은 애플리케이션의 동작에 전혀 영향을 주지 않고, 커널 수준에서 직접 데이터를 수집하므로 성능 저하를 최소화하면서도 깊이 있는 가시성을 확보할 수 있습니다.
- 낮은 오버헤드
  - eBPF 기반의 프로파일링은 약 1%의 CPU와 250MB 정도의 메모리라는 극히 낮은 오버헤드만으로 운영이 가능합니다. 이는 성능에 민감한 운영 환경에서도 부담 없이 상시 프로파일링을 적용할 수 있음을 의미하며, 과거 높은 부하 때문에 프로파일링 도입을 망설였던 기업들에게 새로운 가능성을 열어줍니다.
- Zero-Instrument 접근 방식
  - MSAP Observability는 애플리케이션 코드에 별도의 데이터 수집 라이브러리(SDK)나 에이전트를 추가할 필요가 없는 'Zero-Instrument' 방식을 지향합니다. 개발자는 어떠한 코드 수정이나 의존성 추가 없이도, eBPF 에이전트 배포만으로 시스템의 성능 프로파일링을 즉시 시작할 수 있습니다. 이러한 아키텍처적 선택은 개발자의 마찰을 줄이고 전사적으로 관측성 프랙티스의 채택을 가속화하여 총소유비용(TCO)을 절감하는 직접적인 효과로 이어집니다.

결과적으로, MSAP Observability의 Continuous Profiling은 운영 환경에 상시 켜두더라도 SLO에 영향을 주지 않는 수준의 오버헤드로, “언제든 과거 특정 시점의 코드 레벨 상태를 되짚어 볼 수 있는 데이터”를 확보하는 데 목적을 둡니다. 업계에서도 이를 “프로덕션에서 항상 실행되는 저오버헤드 프로파일링”으로 정의하며 Observability의 네 번째 축으로 분류하고 있습니다.

4.1.2 샘플링 기반 CPU 사용률·메모리 할당 추적

시스템의 안정성은 결국 CPU와 메모리라는 핵심 자원을 얼마나 효율적으로 사용하는지에 달려있습니다. 특정 프로세스가 CPU를 과도하게 점유하거나, 비정상적인 메모리 할당 패턴으로 인해 메모리 누수가 발생하는 문제는 서비스 장애의 주요 원인이 됩니다. 따라서 이러한 자원 사용 패턴을 지속적으로 추적하고 분석하는 것은 안정적인 운영의 기본입니다.

MSAP Observability는 효율적인 데이터 수집을 위해 샘플링(Sampling) 기법을 사용합니다. 이는 시스템의 모든 함수 호출을 추적하는 대신, 일정한 간격으로 시스템의 상태를 스냅샷처럼 수집하는 방식입니다. 이 샘플링 데이터를 통계적으로 분석하면 매우 낮은 오버헤드로도 시스템 전체의 CPU 사용률 및 메모리 할당 패턴을 정확하게 파악할 수 있습니다.

1. CPU 프로파일링 (CPU Time / Wall Time)
    - 일정 주기마다 실행 중인 스레드의 스택을 캡처합니다.
    - 각 스택 프레임이 샘플에 등장한 빈도는 해당 메서드가 CPU를 얼마나 많이 사용했는지에 비례합니다.
    - CPU 바운드 워크로드의 경우, 샘플 빈도 분포만으로도 전체 CPU 시간에서 각 메서드가 차지하는 비율을 통계적으로 추정할 수 있습니다.
2. 쿠버네티스·MSA 컨텍스트와의 결합
    - 샘플에는 단순히 메서드 이름만 담기는 것이 아니라, 서비스 이름, 네임스페이스, Pod, 컨테이너, 트랜잭션 ID, 태그(배포 버전, 릴리즈 채널 등) 같은 메타데이터가 함께 붙습니다.
    - MSAP COP에서 수집된 APM 트랜잭션, 메트릭, 로그, 트레이스와 동일한 라벨 체계를 사용하므로, “CPU를 가장 많이 쓰는 메서드가 어느 MSA 서비스의 어느 버전에서 실행되는지”를 한 화면에서 확인할 수 있습니다.

이렇게 샘플링 기반으로 수집된 프로파일은 개별 호출 기록이 아니라, 시간 구간별 통계적 스냅샷 이기 때문에 저장량과 처리량도 비교적 안정적으로 관리할 수 있습니다.

4.1.3 언어 독립 호출 스택 수집·표준화 구조

마이크로서비스 아키텍처(MSA) 환경에서는 Java, Python, Go, Node.js 등 다양한 프로그래밍 언어로 개발된 서비스들이 공존하는 것이 일반적입니다. 이러한 환경에서 특정 언어에 종속된 모니터링 도구는 통합된 가시성을 제공하는 데 한계가 있습니다.

MSAP Observability는 다음과 같이 설계됩니다.

1. 언어/플랫폼 중립 프로파일 스키마
    - 저장 시점에는 “프로파일 타입(CPU, 메모리, 락, I/O 등)” 과 “스택 트리 + 메타데이터”만을 공통 구조로 보존합니다.
    - 자바/Go 여부에 관계없이, 상위 레벨에서는 동일한 쿼리 모델(예: `service = 주문 서비스 AND profile_type = cpu AND version = v2`)로 질의할 수 있습니다.

이렇게 수집된 다양한 언어의 호출 스택 정보는 플레임 그래프(Flame Graph) 와 같은 표준화된 데이터 구조로 시각화됩니다. 이를 통해 개발자와 운영자는 언어의 차이를 넘어, 직관적인 시각 자료를 통해 시스템의 성능 병목 지점을 일관된 방식으로 분석할 수 있습니다.

이처럼 ‘올웨이즈 온’ 아키텍처는 eBPF 기술을 통해 낮은 오버헤드로, 다양한 언어 환경에서 중단 없이 데이터를 수집합니다. 이렇게 수집된 방대한 프로파일링 데이터는 그 자체로는 의미를 파악하기 어렵습니다. 다음 절에서는 이 데이터를 어떻게 효과적으로 시각화하여 실제 성능 병목 지점을 신속하게 식별하고 분석하는지 구체적으로 살펴보겠습니다.

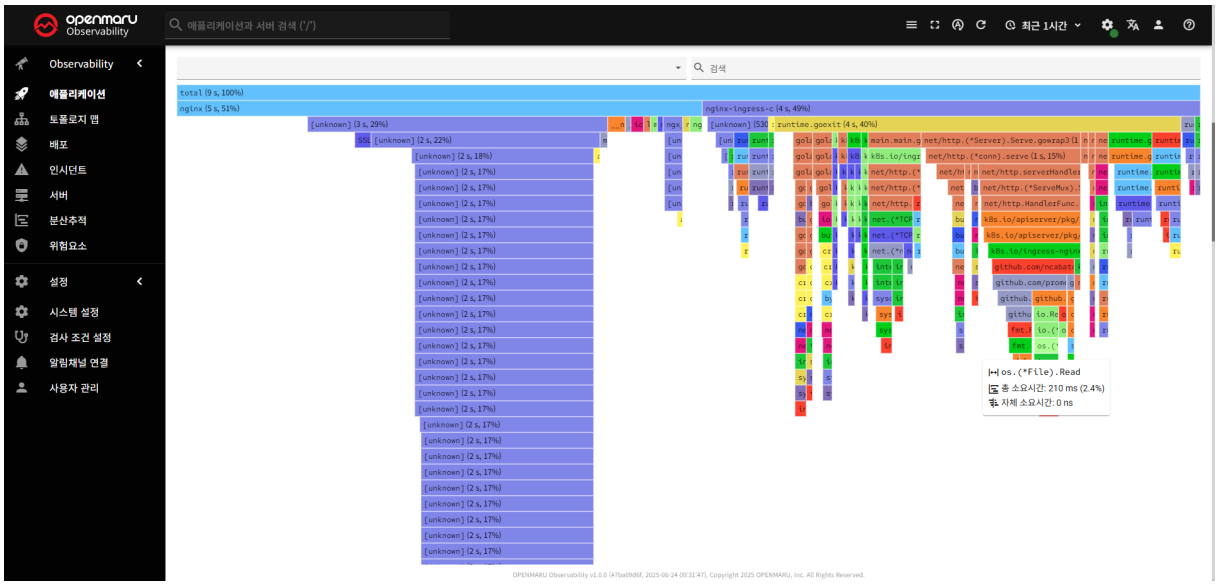
4.2 코드 레벨 병목 지점 시각화

상시 프로파일링을 통해 수집된 방대한 데이터는 그 자체로 가치를 가지기 어렵습니다. 수많은 함수 호출과 리소스 할당 기록 속에서 실제 성능에 영향을 미치는 핵심 원인을 찾아내는 것은 또 다른 도전 과제입니다. 따라서 수집된 데이터를 개발자와 운영자가 직관적으로 이해하고 신속하게

조치할 수 있는 형태로 시각화하는 과정이 필수적입니다. 효과적인 시각화는 복잡한 성능 데이터의 본질을 꿰뚫어 보고, 최적화가 필요한 코드의 위치를 명확하게 제시하여 문제 해결 시간을 획기적으로 단축시키는 역할을 합니다.

4.2.1 플레임 그래프를 통한 Hotspot 메서드 식별

Continuous Profiling 데이터를 사람이 이해하기 쉽게 만들어 주는 핵심 도구가 플레임 그래프(Flame Graph) 입니다. 플레임 그래프는 CPU 프로파일링 데이터를 시각화하는 가장 강력하고 직관적인 도구 중 하나입니다. 이는 전체 호출 스택을 하나의 그래프로 표현하여, 어떤 함수가 가장 많은 CPU 시간을 소비하는지, 즉 코드의 'Hotspot'이 어디인지 한눈에 파악할 수 있게 해줍니다.



[그림 1] 플레임 그래프

- 플레임 그래프의 구조
  - 가로축 (너비): 응답 시간을 의미합니다. 그래프에서 특정 노드(함수)의 너비가 넓을수록, 해당 함수 및 그 함수가 호출한 모든 하위 함수의 총 실행 시간이 길다는 것을 나타냅니다. 즉, 넓은 노드가 성능에 큰 영향을 미치고 있음을 의미합니다.
  - 세로축 (깊이): 호출 스택(Call Stack)의 깊이를 의미합니다. 그래프의 아래에서 위로 갈수록 함수 호출 관계를 나타내며, y축의 가장 아래쪽 노드가 최초 호출 함수이고, 위로 쌓이는 노드들이 순차적으로 호출된 함수들입니다. 그래프의 깊이가 깊을수록 중

첩된 함수 호출이 많다는 것을 의미합니다.

- Hotspot 식별 방법
  - 플레임 그래프에서 성능 병목의 주된 원인, 즉 Hotspot은 일반적으로 그래프의 상단에 위치하면서 너비가 넓은 노드일 가능성이 높습니다. 이러한 노드는 다른 함수를 호출하지 않으면서 자기 자신(self-time)의 실행 시간이 길다는 것을 의미하기 때문입니다. 개발자는 이 넓은 '고원' 같은 부분을 찾는 것만으로도 최적화가 필요한 대상을 직관적으로 식별하고 성능 개선 작업에 집중할 수 있습니다.

플레임 그래프를 통해 IT 의사결정자는 다음을 직관적으로 파악하실 수 있습니다.

- "어떤 서비스의 어떤 메서드가 전체 CPU의 대부분을 사용하고 있는지"
- "새로운 릴리스 이후 CPU 사용 패턴이 어떻게 변했는지"
- "GC나 런타임 내부 함수가 비정상적으로 많은 비율을 차지하고 있는지"

MSAP COP의 APM 트랜잭션 뷰와 연결하면, 특정 느린 트랜잭션을 클릭했을 때 해당 시간 구간의 플레임 그래프를 바로 열어, APM의 URL/트랜잭션 지연 → 코드 레벨 Hotspot 으로 자연스럽게 Drill-down 할 수 있도록 지원할 예정입니다.

4.2.2 스레드 경합·락 대기시간·동시성 이슈 분석

다중 스레드 환경에서 여러 스레드가 동시에 동일한 자원에 접근하려고 할 때 발생하는 스레드 경합(Thread Contention) 이나, 특정 자원을 사용하기 위해 다른 스레드가 끝날 때까지 기다리는 락(Lock) 대기는 눈에 잘 띄지 않는 성능 저하의 주된 원인입니다.

MSAP Observability는 이러한 락 프로파일을 플레임 그래프로 변환해, 어떤 락 객체가 가장 긴 시간 동안 점유되는지 , 어떤 호출 경로에서 락 경합이 발생하는지를 한눈에 보여줍니다.

예를 들어, 'CPU 대기 시간 차트' 는 CPU가 실제로 연산을 수행하지 못하고 I/O나 다른 리소스를 기다리며 낭비하는 시간을 보여줍니다. 이 차트에서 '다른 프로세스와의 경합'으로 인한 대기 시간이 높게 나타난다면, 이는 시스템 내에서 심각한 리소스 경합이 발생하고 있음을 시사합니다. 프로파일링 데이터를 통해 어떤 코드 영역에서 락 대기가 빈번하게 발생하는지 식별하고, 동기화 로직을 개선하거나 락의 범위를 최소화하는 등의 최적화를 수행하여 동시성 문제를 해결할 수 있습니다.

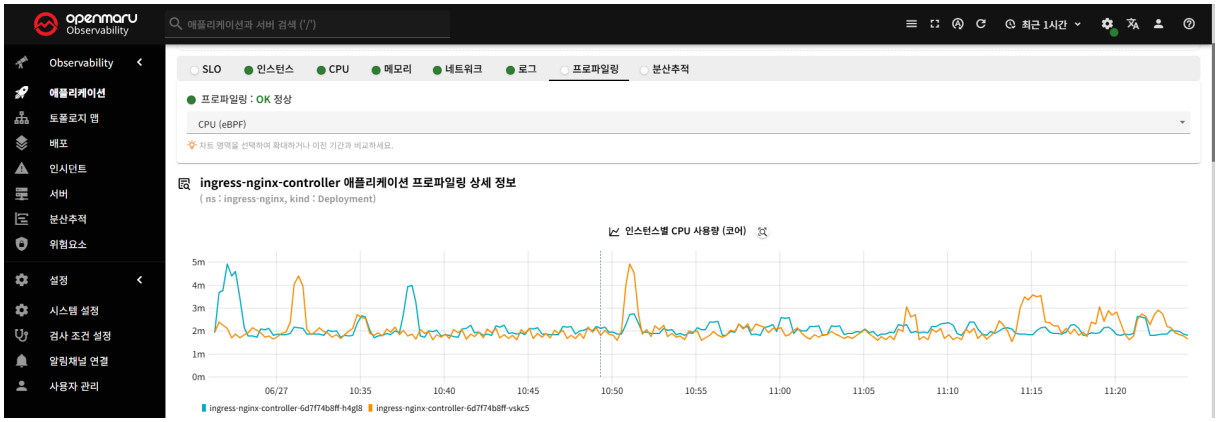
이처럼 코드 레벨의 병목 지점을 시각적으로 분석하는 기능은 개발자와 운영자가 막연한 추측이 아닌 명확한 데이터를 기반으로 성능 문제를 해결하도록 돕습니다. 이는 기술적인 문제 해결을 넘어, 개발팀과 운영팀이 공통의 데이터를 중심으로 소통하고 협력하는 문화를 촉진하는 기반이 됩니다. 다음 절에서는 이러한 프로파일링 데이터가 실제 개발 및 운영 프로세스에서 어떻게 활용되어 협업을 강화하는지 살펴보겠습니다.

4.3 개발·운영 협업을 위한 프로파일링 활용

Continuous Profiling을 통해 얻어진 데이터는 단순히 기술적 문제를 해결하는 도구를 넘어섭니다. 이는 개발팀과 운영팀(DevOps/SRE)이 동일한 데이터를 기반으로 시스템의 상태를 이해하고, 문제 해결을 위해 협력하는 강력한 매개체 역할을 합니다. 과거에는 운영팀이 “시스템이 느리다”고 보고하면, 개발팀은 “내 코드에는 문제가 없다”고 대응하는 등 단절된 소통이 빈번했습니다. 이제는 플레임 그래프와 같은 명확한 코드 레벨의 증거를 통해, 모든 팀이 문제의 원인에 대해 공통의 이해를 형성하고, 데이터 기반의 생산적인 논의를 통해 해결책을 찾아가는 협업 문화를 구축할 수 있습니다.

4.3.1 재현하기 어려운 간헐적 장애의 코드 레벨 원인 규명

실제 운영 환경에서는 “간헐적으로만 발생하는 장애”가 가장 골치 아픈 경우가 많습니다. 특정 시간대, 특정 입력 조합, 특정 부하 상황에서만 발생하기 때문에 테스트 환경에서 재현하기가 거의 불가능합니다.



[그림 2] MSAP Observability Continuous Profiling

Continuous Profiling의 강점은 “장애가 발생했든 안 했든, 그 시점의 코드 실행 상태가 이미 기록되어 있다” 는 점입니다. 시스템에 항상 상주하며 모든 활동을 기록하고 있기 때문에, 간헐적인 장애가 발생했을 때 별도의 조치를 취하지 않아도 해당 시점의 프로파일링 데이터가 이미 확보되어 있습니다. 장애 발생 후, 운영자와 개발자는 확보된 데이터를 통해 사후 분석을 진행할 수 있습니다. 플레임 그래프를 통해 장애가 발생한 바로 그 순간에 어떤 메서드가 비정상적으로 많은 CPU를 사용했는지, 또는 어떤 스레드에서 경합이 발생했는지 코드 레벨에서 명확하게 규명할 수 있습니다. 이는 더 이상 ’재현’에 의존하지 않고도 문제의 근본 원인을 찾아 해결할 수 있음을 의미합니다.

4.3.2 배포 전·후 프로파일링 데이터 비교를 통한 성능 회귀 검증

새로운 기능을 추가하거나 코드를 리팩토링한 후 배포했을 때, 의도치 않게 시스템 성능이 저하되는 성능 회귀(Performance Regression) 문제가 발생할 수 있습니다. 이러한 문제를 조기에 발견하지 못하면 사용자 경험에 심각한 영향을 미칠 수 있습니다.

MSAP Observability의 배포 추적 기능은 이러한 성능 회귀를 체계적으로 검증하는 시나리오를 제공합니다. 새로운 버전이 배포되면, 시스템은 자동으로 해당 배포 이벤트를 기록합니다. 이후, 새로운 버전의 프로파일링 데이터(CPU 사용량, 특정 메서드의 실행 시간 등)를 이전 버전의 데이터와 자동으로 비교 분석합니다. 만약 특정 지표에서 유의미한 성능 저하가 감지되면, 이를 즉시 개발팀과 운영팀에 알려줍니다. 이 과정을 통해 개발자는 자신이 수정한 코드가 시스템 전체에 어떤 영향을 미쳤는지 신속하게 파악하고, 문제가 있다면 빠르게 롤백하거나 수정하여 성능 저하를 방지할 수 있습니다.

4.3.3 튜닝 결과 검증 및 성능 개선 활동의 정량적 평가

성능 개선(튜닝) 활동은 그 결과를 정량적으로 평가하고 증명할 수 있을 때 진정한 가치를 가집니다. 막연히 “개선되었다”고 말하는 대신, 구체적인 수치로 효과를 입증해야 합니다.

Continuous Profiling은 다음과 같은 방식으로 정량 평가의 기반을 제공합니다.

1. 튜닝 전·후 Hotspot 비중 비교
    - 특정 메서드 튜닝 전에는 CPU 프로파일에서 전체의 35%를 차지하던 메서드가, 튜닝

후 10%로 떨어졌다면, 해당 최적화가 실제로 큰 효과를 냈음을 바로 확인할 수 있습니다.

2. 서비스/클러스터 자원 절감 효과 산출
    - Continuous Profiling과 노드·Pod 메트릭을 결합하면, 특정 튜닝으로 인해 CPU 사용량이 몇 코어 감소했는지, 그에 따라 클러스터 리소스를 얼마나 줄일 수 있는지 추정할 수 있습니다.
    - 이는 클라우드 비용 절감, 온프레미스 GPU/CPU 클러스터 효율 향상 등 경영적 관점의 인사이트로 바로 연결됩니다.
3. 개발팀·운영팀 OKR/KPI 연계
    - “주요 서비스 3개에 대해 CPU 사용량 20% 절감” 같은 목표를 세운 뒤, MSAP Observability의 프로파일 데이터를 기반으로 달성 여부를 측정할 수 있습니다.
    - 단순 인프라 비용뿐 아니라, 피크 타임 트래픽 흡수 여유도, 장애 발생 빈도 감소 등 운영 안정성에도 직접적인 영향을 줍니다.

이처럼 특정 코드나 쿼리를 수정한 후, 수정 전과 후의 프로파일링 데이터를 비교하면 “특정 메서드의 평균 실행 시간이 500ms에서 150ms로 단축되었다” 와 같이 구체적인 수치를 통해 성능 개선 활동의 성과를 객관적으로 평가하고, 이를 조직 내에 공유하여 성공적인 개선 문화를 정착시킬 수 있습니다.

MSAP Observability와 MSAP COP, 그리고 AI Native 애플리케이션 플랫폼인 MSAP.ai가 결합되면, 이런 정량 데이터를 기반으로 어떤 서비스·어떤 메서드를 우선적으로 튜닝해야 ROI가 높은지 를 AI가 추천하는 시나리오까지 확장할 수 있습니다.

결론적으로, Continuous Profiling은 현대 IT 시스템 운영의 복잡성을 해결하기 위한 필수적인 전략입니다. 이는 단순히 장애를 해결하는 사후 대응적 도구를 넘어, 시스템의 내부 동작을 코드 레벨까지 깊이 있게 이해하고, 데이터에 기반하여 성능을 사전에 최적화하는 사전 예방적 접근을 가능하게 합니다. 개발과 운영의 경계를 허물고, 모든 팀이 공통의 데이터를 통해 협업하며, 안정적이고 효율적인 서비스를 제공하는 것. 이것이 바로 Continuous Profiling이 지향하는 목표이자, 미래 지향적인 IT 운영의 핵심입니다.

제5장. Kubernetes·인프라 및 MSA 토폴로지 통합 가시성

사용자의 단 한 번의 클릭이 내부적으로는 수십, 수백 개의 마이크로서비스 간 연쇄적인 호출을 유발하며, 이 서비스들은 Kubernetes 클러스터 위에서 수시로 생성되고, 확장되고, 사라집니다. 이러한 환경에서 과거의 파편화된 모니터링 방식, 즉 로그, 메트릭, 트레이스를 각기 다른 도구로 수집하는 방식으로는 장애의 근본 원인을 신속하게 파악하는 데 명백한 한계를 드러냅니다. 엔지니어는 여러 시스템을 오가며 데이터를 수동으로 연관 분석해야 하므로 원인 규명에 많은 시간이 소요될 수밖에 없습니다.

쿠버네티스 기반 MSA 환경에서 "문제가 어디서 시작되었는지"를 찾으려면, 단순히 Pod 하나의 상태만 보는 것으로는 거의 불가능합니다.

노드(Node)–컨테이너–클러스터 인프라 계층과, 서비스–DB–외부 API로 이어지는 MSA 토폴로지가 하나의 관점에서 통합되어 보이는가가 관건입니다.

MSAP Observability는 바로 이 지점을 겨냥하여, 인프라 메트릭·이벤트·로그·트레이스를 단일 UX에서 연결해 줍니다.

아래에서는 이를 세 가지 관점, 즉 (1) 인프라 계층 분석, (2) 동적 서비스 맵, (3) 배포 영향도와 풀스택 관찰로 나누어 설명드리겠습니다.

5.1 Node·Container·클러스터 수준 분석

컨테이너화된 환경에서 애플리케이션의 성능은 결코 그 기반이 되는 인프라의 상태와 분리하여 생각할 수 없습니다. 안정적인 Node, 예측 가능한 Pod 스케줄링, 그리고 충분한 자원을 할당받은 Container가 보장되지 않는다면, 아무리 잘 만들어진 애플리케이션이라도 제 성능을 발휘할 수 없습니다. 따라서 Node, Pod, Container 수준의 세밀한 가시성을 확보하는 것은 시스템의 전반적인 안정성을 예측하고 보장하는 전략적인 첫걸음입니다.

5.1.1 CPU·메모리·I/O·네트워크·로드 지표 통합 보기

MSAP Observability는 Kubernetes 클러스터의 상태를 진단하는 데 필수적인 핵심 자원 지표를 통합된 대시보드에서 직관적으로 제공합니다. 이를 통해 운영자는 개별 시스템에 접속하지 않고도 클러스터 전반의 리소스 현황을 한눈에 파악하고 잠재적인 병목 지점을 예측할 수 있습니다.

그래서 MSAP Observability는 다음과 같은 축으로 데이터를 수집·분석합니다.

- Node 레벨: CPU 코어 사용률, Load Average, 시스템 메모리 사용률, 디스크 I/O 대기시간, 네트워크 대역폭·패킷 드롭 등
- Pod/Container 레벨: 컨테이너별 CPU 사용량/Limit 대비 사용률, 메모리 사용량·메모리 한계치, 네임스페이스별 네트워크 트래픽, 파일 디스크립터 수 등
- 클러스터 레벨: 노드 수, Ready/NotReady 노드 분포, 전체 Pod 수와 Pending/Running/CrashLoopBackOff 비율, Namespace별 리소스 사용 현황 등

중요한 점은, 이 지표들을 별도의 대시보드 조합이 아니라 “한 화면에서 상관관계까지 고려해 보는 것”입니다.

- CPU 스파이크와 동시에, 같은 시점에 특정 서비스의 평균 응답시간(P95 latency)이 급증했는지
- 특정 노드의 I/O Wait이 올라간 시점에, 그 노드에 스케줄된 Pod들의 에러율이 함께 증가했는지
- 네트워크 수신/송신량이 비정상적으로 증가한 뒤 곧바로 타임아웃 에러가 늘어났는지

MSAP Observability는 이러한 지표들을 시간축 기반으로 정렬하고, 동일 타임라인 위에 겹쳐 보여줍니다.

또한 MSAP COP·MSAP.ai와 연계하면, APM 트랜잭션 지연(애플리케이션 계층)과 Node/Pod 메트릭(인프라 계층)을 한 번에 연결해 볼 수 있어,

“어느 계층에서 병목이 시작되었는지”를 몇 번의 클릭만으로 좁혀갈 수 있습니다.

이 수준의 통합 가시성은, 단순히 “CPU가 높다/낮다”를 보는 모니터링과는 본질적으로 다릅니다.

“비즈니스 지연이 발생했을 때, 인프라 지표가 어떤 패턴을 동반하는지”를 함께 보는 것이 운영자가 실제로 필요로 하는 관점입니다.

5.1.2 OOM, Throttling, I/O Wait, 디스크 공간 부족 탐지

MSAP Observability는 단순한 자원 사용률 모니터링을 넘어, 시스템 안정성에 치명적인 영향을 미치는 주요 이상 '상태'를 자동으로 탐지하고 알려줍니다. 이는 문제가 발생한 후에 대응하는 것이 아니라, 장애로 이어질 수 있는 잠재적 위험을 사전에 인지하고 조치할 수 있게 합니다.

이상 상태	정의 및 원인	비즈니스 영향
메모리 부족(OOM)	컨테이너가 할당된 메모리 한계(limit)를 초과하여 Kubernetes의 OOM Killer에 의해 강제 종료되는 상황. 메모리 누수나 잘못된 리소스 설정이 주원인입니다.	서비스의 갑작스러운 중단으로 인한 진행 중인 사용자 트랜잭션 유실 및 직접적인 매출 손실 위험.
CPU 스로틀링(Throttling)	컨테이너가 할당된 CPU 사용량 한계(limit)에 도달하여 성능이 강제로 저하되는 현상. CPU 집약적인 작업이 급증할 때 발생합니다.	애플리케이션 응답 시간 급증으로 인한 사용자 경험 저하 및 서비스 이탈률 증가.
I/O 대기(I/O Wait)	CPU가 디스크 읽기/쓰기 등 I/O 작업이 완료되기를 기다리며 유휴 상태에 빠지는 시간. 느린 스토리지나 과도한 디스크 접근이 원인입니다.	스토리지 병목으로 인한 전체 시스템 처리량 저하 및 SLA 위반 가능성.
디스크 공간 부족	Pod가 사용하는 영구 볼륨(Persistent Volume)의 공간이 부족하여 더 이상 데이터를 기록하지 못하는 상태.	신규 데이터 처리 불가로 서비스 기능 마비 및 데이터 정합성 훼손.

요약하면, 이 절에서 중요한 것은 "지표의 종류를 나열하는 것"이 아니라, OOM/Throttling/I/O Wait/디스크 부족이라는 전형적인 패턴을 빠르게 식별하고, 애플리케이션 영향도와 연결해서 보는 능력입니다.

MSAP Observability는 바로 이 패턴 식별을 표준화된 UX로 제공합니다.

5.1.3 Pod 스케줄링 실패·재시작·CrashLoop 원인 분석

쿠버네티스 운영에서 가장 자주 보는 상태가 바로 Pending, ImagePullBackOff, CrashLoopBackOff입니다. **MSAP Observability**는 애플리케이션의 '인스턴스' 탭에서 개별 Pod의 비정상적인 재시작 횟수를 실시간으로 추적하고, 사전에 설정된 임계값(예: 5회 초과)을 넘어서면 이를 즉시 감지하여 경고합니다.

특히 `CrashLoopBackOff` 상태와 같이 Pod가 시작과 비정상 종료를 반복하는 현상은 심각한 문제의 징후입니다. 이러한 현상은 단지 애플리케이션 코드의 버그 때문만이 아니라, 앞서 설명한 인프라 문제와 깊이 연관되어 있을 수 있습니다. MSAP Observability의 통합된 가시성을 통해, 운영자는 CrashLoopBackOff 알림을 받은 즉시 해당 Pod의 '인스턴스' 탭에서 재시작 시간대를 확인하고, 클릭 한 번으로 동일 시간대의 '메모리' 탭으로 이동하여 OOM Kill 이벤트를 발견하거나 'CPU' 탭에서 심각한 Throttling 발생을 확인하는 등, 파편화된 도구로는 수십 분이 걸릴 근본 원인 분석을 단 몇 분 안에 완료할 수 있습니다.

MSAP Observability는 이 상태들을 단순히 카운트하는 수준을 넘어, 왜 그런 상태로 빠졌는지를 구조적으로 분석할 수 있게 합니다.

1. Pod 스케줄링 실패(Pending)
    - 리소스 부족(CPU/메모리/스토리지), 노드 셀렉터/태인트/톨러런스 불일치, PodAntiAffinity 조건 충돌 등으로 발생합니다.
    - 운영자는 보통 `kubectl describe pod`로 이벤트를 하나씩 확인하지만, 규모가 커지면 이 방식은 한계가 있습니다.
    - MSAP Observability는 클러스터 전체에서 Pending 상태 Pod를 집계하여, 실패 사유별(Insufficient CPU, Insufficient memory, NodeAffinity 등)로 분류하고, 어느 네임스페이스·어느 Deployment에서 집중적으로 발생하는지를 시각화합니다.
2. Pod 재시작·CrashLoopBackOff
    - 애플리케이션 초기화 실패, 환경 변수/Secret 설정 오류, 의존 서비스(DB, 외부 API) 연결 실패, OOM, ReadinessProbe 실패 등 다양한 원인이 있습니다.
    - MSAP Observability는 Pod 이벤트 로그 + 컨테이너 로그 + Readiness/ Liveness Probe 결과 + APM 에러 스택을 한 화면에서 확인할 수 있도록 구성할 수 있습니다.
    - 예를 들어, 배포 직후 특정 마이크로서비스가 CrashLoopBackOff에 빠졌다면,
        - 마지막으로 실행된 컨테이너 로그의 예외 메시지
        - 같은 시점에 외부 DB 커넥션 타임아웃이 증가했는지
        - 해당 마이크로서비스가 의존하는 상위/하위 서비스의 에러율 변화를 한 번에 함께 보는 것이 실제 트러블슈팅 흐름에 가깝습니다.

3. 원인-결과 체인(Chain) 관점

   - “노드 리소스 부족 → 스케줄링 실패 증가 → 일부 서비스 인스턴스 수 감소 → 트래픽 집중 → 응답시간 증가 및 에러율 상승”과 같은

     연쇄적인 Chain을 시간 순으로 복원할 수 있는지가 관찰 가능성(Observability)의 핵심입니다.

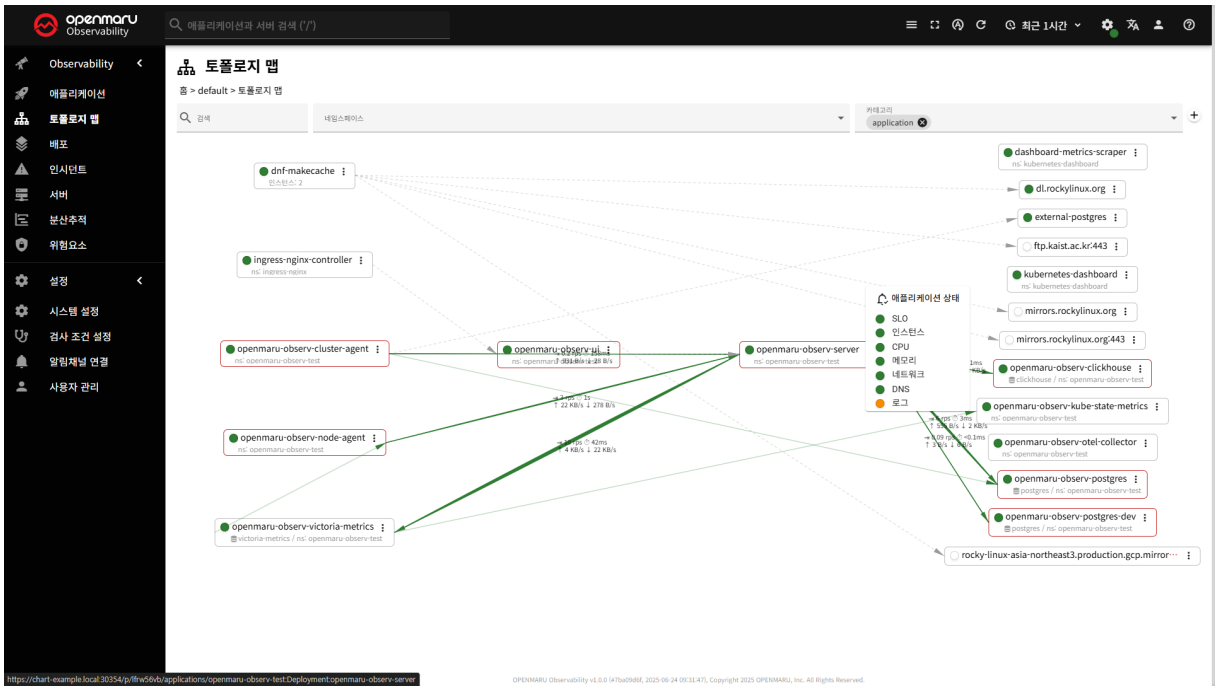
   - MSAP Observability는 Node/Pod 이벤트, Deployment/ReplicaSet 변경, APM 트랜잭션, 로그를 모두 같은 타임라인에 배치하여,

     운영자가 이 Chain을 시각적으로 추적할 수 있도록 돕습니다.

이처럼 인프라 수준의 깊이 있는 통찰력은 애플리케이션의 실제 동작과 서비스 간의 복잡한 상호작용을 이해하는 견고한 기반이 되며, 이는 동적인 마이크로서비스 환경을 분석하는 다음 단계로 자연스럽게 이어집니다.

5.2 동적 MSA 서비스 맵과 의존성 분석

마이크로서비스 아키텍처(MSA)에서는 서비스 간의 호출 관계가 배포, 오토스케일링, 기능 플래그 등에 따라 실시간으로 변화합니다. 이러한 동적인 환경에서 전통적인 구성 관리 데이터베이스(CMDB)나 정적인 아키텍처 다이어그램은 현실을 반영하지 못하고 무의미해집니다. **MSAP Observability**가 제공하는 자동화된 서비스 맵은 코드 수정이나 복잡한 설정 없이 시스템의 현재 상태를 있는 그대로 시각화하는 ‘실시간 청사진’ 역할을 수행하며, 복잡한 시스템을 이해하는 데 핵심적인 도구가 됩니다.



[그림 3] MSAP Observability 토폴로지 맵

5.2.1 서비스 간 통신 빈도·지연·에러율을 반영한 토폴로지 자동 생성

MSAP Observability의 서비스 토폴로지 맵은 Linux 커널 기술인 **eBPF**를 기반으로 합니다. 이는 애플리케이션 코드를 수정하거나(Code Instrumentation), 언어별로 별도의 에이전트를 설치할 필요 없이(Zero-Instrument) 시스템의 네트워크 트래픽을 커널 레벨에서 직접 관찰합니다. 이는 개발팀이 애플리케이션 출시 일정을 지연시키는 고통스러운 코드 계측(code instrumentation) 작업에서 완전히 해방됨을 의미하며, 서비스 간의 모든 통신을 누락 없이 자동으로 감지하고 맵을 생성합니다.

토폴로지 맵의 각 서비스 노드를 연결하는 선에는 다음과 같은 핵심 성능 지표가 실시간으로 표시되어 서비스 간의 관계 상태를 직관적으로 보여줍니다.

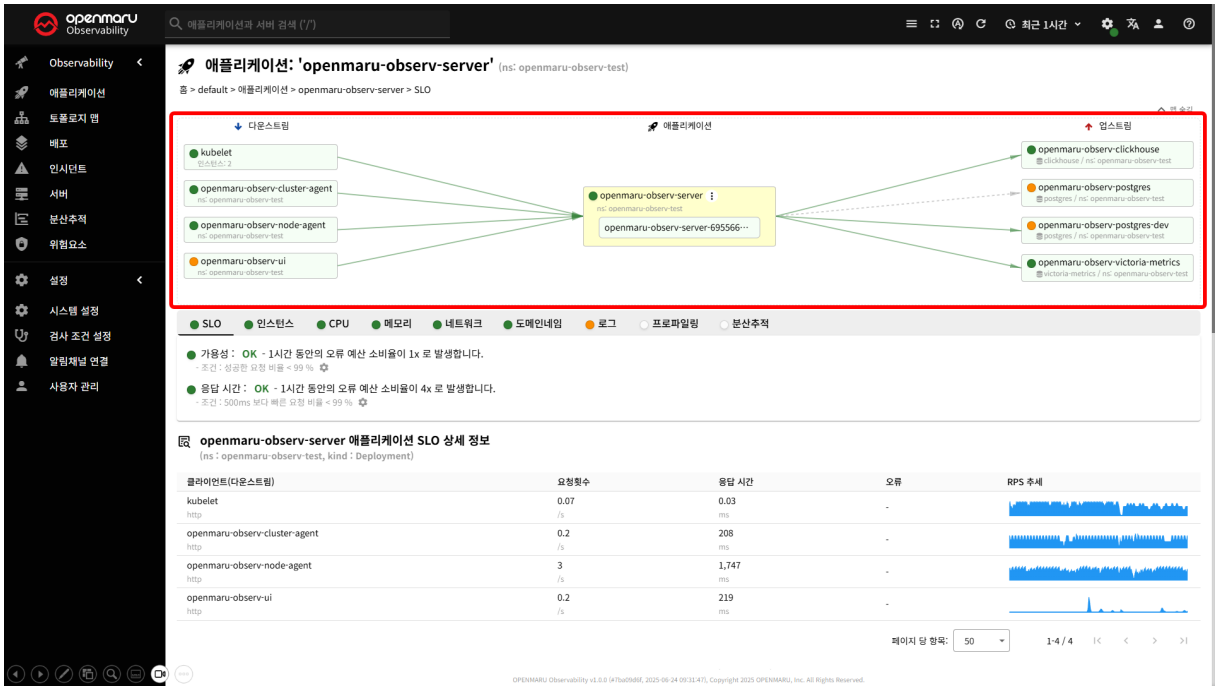
- □ 초당 요청 수 (RPS): 서비스 간의 통신 빈도를 나타내며, 비정상적인 트래픽 증가나 감소를 즉시 파악할 수 있습니다.
- □ 평균 응답 시간: 서비스 간의 지연 시간을 나타내며, 병목이 발생하는 구간을 시각적으로 식별하는 데 도움을 줍니다.
- ↑ ↓ 트래픽 양(Bytes/sec): 송수신되는 데이터의 양을 보여주어 네트워크 대역폭 사용량을 분석할 수 있습니다.

이 동적 서비스 맵은, MSAP COP에 포함된 APM/Observability 기능과 결합되어,Web/WAS·Session Clustering·AI 호출까지 포함한 전체 애플리케이션 체인을 한 번에 보여주는 역할을 합니다.

5.2.2 업스트림·다운스트림 연계 영향도 분석

장애 분석에서 정말 중요한 질문은 “지금 문제를 일으키는 서비스가 남에게 어떤 영향을 주는지”와 “반대로 누구 때문에 내가 피해를 보는지”입니다.

MSAP Observability의 서비스 맵은 **다운스트림(Downstream) -> 애플리케이션(Application) -> 업스트림(Upstream)**이라는 명확한 구조로 의존 관계를 시각화합니다.



[그림 4] 애플리케이션 맵

- 업스트림(Upstream): 현재 분석 중인 애플리케이션이 의존하는 서비스 (예: 데이터베이스, 외부 API, 메시지 큐)
- 애플리케이션(Application): 분석의 중심이 되는 서비스
- 다운스트림(Downstream): 현재 애플리케이션을 호출하는 서비스 (예: 프론트엔드, 다른 마이크로서비스)

이 구조를 통해 특정 서비스의 장애가 시스템 전체에 미치는 연쇄적인 영향을 신속하게 분석할 수 있습니다. 예를 들어, **주문 DB**(업스트림)의 응답 시간이 급증하면, 토폴로지 맵에서 **주문 DB**와

연결된 선의 색상이 주황색이나 빨간색으로 변합니다. 운영자는 이를 통해 **주문 서비스**(애플리케이션)의 성능 저하가 **주문 DB** 때문임을 즉시 인지하고, 나아가 **주문 서비스**를 호출하는 **결제 서비스**와 **UI 프론트엔드**(다운스트림)까지 영향이 전파되고 있음을 한눈에 파악할 수 있습니다.

이를 통해, 운영자는 단순히 “장애 서비스 S”에 머무르지 않고, “장애 전파 범위”와 “우선 대응해야 할 경로”를 빠르게 판단할 수 있습니다.

5.2.3 DB·Cache·외부 API 등 외부 의존 서비스 호출 현황 가시화

MSA 환경의 애플리케이션은 다양한 외부 서비스에 의존하여 동작합니다. MSAP Observability는 eBPF를 통해 다양한 프로토콜을 자동으로 감지하고, 다음과 같은 외부 의존 서비스를 토폴로지 맵에 자동으로 시각화합니다.

- 관계형 데이터베이스: PostgreSQL, MySQL 등
- NoSQL 및 캐시: Redis, etcd, MongoDB 등
- 메시지 큐: Kafka, ActiveMQ 등
- 외부 API: 외부 서비스와의 HTTP/gRPC 통신
- 서비스 메시: Istio, Envoy 등
- DNS: CoreDNS 등

이러한 자동 탐지 기능의 핵심 가치는 공식 아키텍처 문서에는 누락되었을 수 있는 ’비공식적 의존성(shadow dependencies)’까지 명확히 밝혀, 시스템의 실제 상호작용에 대한 완전한 그림을 제공한다는 점입니다.

동적 서비스 맵은 시스템의 현재 상태에 대한 가장 정확한 ’지도’를 제공합니다. 이 지도는 시스템에 새로운 배포나 설정 변경이 발생했을 때 그 영향 범위를 예측하고 이해하는 데 필수적인 기반이 되며, 이는 다음 섹션에서 다룰 ’배포 영향도 분석’의 핵심적인 전제 조건이 됩니다.

이 모든 정보는 MSAP COP와의 통합을 통해, Web/WAS–Session–APM–Observability–외부 의존성까지 포함한 엔드투엔드 호출 체인으로 재구성됩니다.

이 호출 패턴 자체를 학습 데이터로 활용해AI 기반 SLO 위반 예측, 장애 패턴 추천, VibeOps 시나리오 자동화까지 확장할 수 있습니다.

5.3 배포 영향도·Full-Stack 관찰

CI/CD 파이프라인을 통한 신속하고 빈번한 배포는 현대 DevOps의 핵심이지만, 동시에 모든 변경 사항은 잠재적인 장애 요인이 될 수 있다는 양면성을 가집니다. 새로운 기능, 버그 수정, 설정 변경이 포함된 배포 직후에 시스템의 성능 변화를 정밀하게 관찰하는 것은, 작은 문제가 큰 장애로 번지기 전에 신속하게 대응하고 서비스 안정성을 유지하는 데 매우 중요합니다.

5.3.1 쿠버네티스 리소스 변경 이벤트 자동 감지

실제 장애의 상당수는 코드 자체보다, 쿠버네티스 리소스 변경(Deployment, ConfigMap, Secret, HPA, Ingress 등)에서 출발합니다.

MSAP Observability는 클러스터에서 발생하는 주요 이벤트를 자동 수집합니다.

- Deployment/StatefulSet 롤아웃 시작·완료·롤백 이벤트
- Replica 수 변경, HPA 스케일 아웃/인 이벤트
- ConfigMap/Secret 마운트 변경, 환경 변수 변경
- Ingress/Service 설정 변경, LoadBalancer IP/포트 변경

이러한 이벤트는 단순히 로그로 남기는 수준을 넘어, 해당 시점의 APM·메트릭·로그와 함께 시각화됩니다.

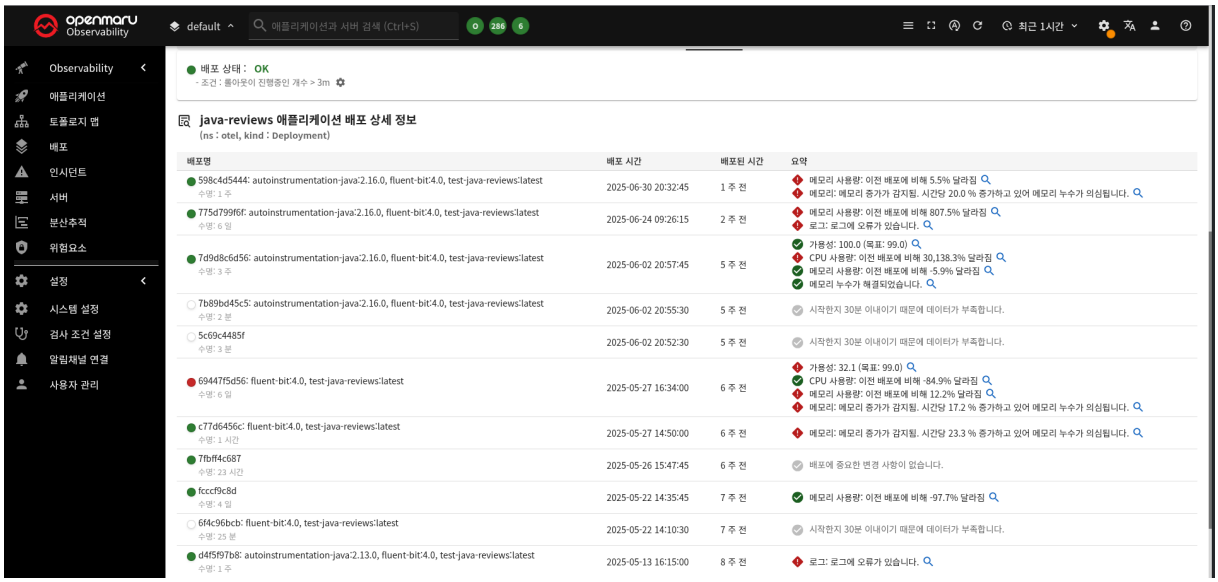
MSAP Observability는 Kubernetes API 서버를 지속적으로 관찰하여 Deployment, StatefulSet 등과 같은 핵심 워크로드 리소스의 변경 사항을 자동으로 감지합니다. 예를 들어, Deployment의 Pod 템플릿에서 컨테이너 이미지 버전이 `v1.1`에서 `v1.2`로 업데이트되면, 이를 새로운 '배포 이벤트'로 즉시 인지하고 기록합니다. 이 자동 감지 기능이 모든 배포 영향도 분석의 시작점이 됩니다.

운영자는 "지금부터 이상 현상이 발생했다"라는 시점 근처에 어떤 리소스 변경이 있었는지를 타임라인에서 바로 확인할 수 있습니다.

5.3.2 CI/CD 연동 없이 배포 시점·버전 변경 추적

현실적으로 모든 배포 파이프라인을 Observability 시스템과 완벽히 연동하기는 어렵습니다.

그래서 MSAP Observability는 CI/CD 도구와의 깊은 통합 없이도 배포 시점과 버전 변화를 추적할 수 있어야 합니다.



[그림 5] Deployment의 배포 내역을 확인할 수 있고, 이전 배포 버전과의 차이점을 확인

이 기능의 가장 큰 장점은 Jenkins, GitLab CI, ArgoCD 등 특정 CI/CD 파이프라인에 대한 별도의 연동 작업이나 플러그인 설치가 전혀 필요 없다는 점입니다. MSAP Observability는 Jenkins 파이프라인의 실행이나 ArgoCD의 Git 커밋을 추적하는 것이 아니라, 그 결과로 Kubernetes API 서버에 적용된 '선언적 상태(declarative state)의 최종 결과물', 즉 Pod 템플릿의 변경을 직접 감지합니다. 이 때문에 어떤 CI/CD 도구를 사용하든 상관없이 일관된 배포 추적이 가능합니다.

5.3.3 배포 전·후 CPU·메모리·지연시간·오류율 비교 분석

배포 영향도를 평가할 때 가장 중요한 것은, "배포 전과 후를 정량적으로 비교했는가"입니다. MSAP Observability의 배포 전후 비교 분석 기능은 '이번 배포가 서비스를 개선했는가, 아니면 새로운 문제를 야기했는가?'라는 가장 중요한 질문에 대해 데이터에 기반한 명확한 답을 제공하여, CI/CD 파이프라인의 핵심적인 품질 게이트 역할을 수행합니다.

새로운 버전이 배포되고 30분이 지나면, MSAP Observability는 자동으로 새로운 버전의 성능 지표를 배포 직전의 이전 버전과 비교 분석하여 그 결과를 요약 리포트로 제공합니다.

분석되는 핵심 항목은 다음과 같습니다.

- SLO (가용성 및 지연시간): 배포 후 서비스의 가용성과 응답 시간이 사전에 정의된 서비스 수준 목표(SLO)를 계속 충족하는지 평가합니다.
- 오류 발생량: 애플리케이션 로그를 분석하여 새로운 유형의 오류가 발생하거나, 특정 오류의 빈도가 급증하지 않았는지 확인합니다.
- Pod 재시작 횟수: 배포된 Pod들이 안정적으로 실행되는지, 아니면 비정상적인 재시작이 증가했는지 추적하여 배포 안정성을 평가합니다.
- CPU 및 메모리 사용량: 새로운 버전의 리소스 효율성을 분석합니다. CPU나 메모리 사용량이 예기치 않게 급증했다면, 코드의 비효율성이나 메모리 누수 등의 잠재적 문제를 의심할 수 있습니다.

이렇게 하면, "이번 배포가 성능을 개선했는지/악화했는지"를 주관이 아니라 객관적 데이터에 기반하여 평가할 수 있습니다.

또한 이 정보는 향후 VibeOps 시나리오에서, 배포 자동 롤백 판단 근거로 활용될 수 있습니다.

5.3.4 클러스터 전체 현황부터 개별 컨테이너 로그까지 드릴다운 UX

마지막으로, Observability 시스템의 실질적인 사용성을 좌우하는 것은 드릴다운(Drill-down) UX입니다. MSAP Observability는 Full-Stack 관점에서 문제를 해결하는 일관된 워크플로우를 제공합니다. 운영자는 여러 도구를 오갈 필요 없이, 단일 플랫폼 내에서 점진적으로 드릴다운하며 근본 원인을 찾아갈 수 있습니다.

1. 1단계 (전체 상황 인지): 서비스 토폴로지 맵에서 '결제 서비스'와 업스트림 '카드사 API'를 잇는 연결선의 응답 시간이 급증하며 붉게 변한 것을 발견합니다.
2. 2단계 (영향도 분석): '결제 서비스'의 상세 화면으로 이동하여 '배포' 탭을 확인합니다. 10분 전에 새로운 버전이 배포되었으며, 배포 직후부터 응답 시간 SLO 위반이 발생했음을 인지합니다.
3. 3단계 (심층 분석): 'CPU' 탭으로 드릴다운하여, 배포된 Pod 중 특정 하나(`paymentservice-7d...`)의 CPU 사용량이 한계치에 근접하며 5.1.2에서 설명한 'CPU 스로틀링(Throttling)'이 발생하고 있음을 확인합니다.

4. 4단계 (근본 원인 규명): 최종적으로 해당 컨테이너의 '로그' 탭으로 이동합니다. 로그 메시지에서 `[ERROR] com.payment.service.card.CardApiClient - Connection timeout after 3000ms while connecting to upstream cards.api.vendor.com` 과 같은 특정 오류가 반복적으로 기록된 것을 확인하고, 새로운 버전의 외부 API 호출 로직에 문제가 있음을 특정합니다.

이처럼 MSAP Observability가 제공하는 인프라, 토폴로지, 배포에 걸친 통합된 가시성은 단순히 장애 발생 후 원인을 찾는 사후 대응적인 문제 해결 도구를 넘어섭니다. 과거 운영팀이 장애 발생 후 여러 시스템을 오가며 원인을 추측하던 수동적이고 반응적인 '소방수(firefighting)' 역할에서 벗어나, 실시간으로 시스템의 건강 상태를 진단하고, 변경 사항의 영향을 즉시 평가하며, 잠재적인 위험을 사전에 예측하게 함으로써, 서비스 안정성을 사전에 확보하고 개발 속도를 가속화하는 '프로액티브(Proactive) 운영 플랫폼' 으로 기능합니다. 이는 복잡한 클라우드 네이티브 환경을 자신감 있게 운영하기 위한 핵심 역량이 될 것입니다.

제6장. LLM 기반 지능형 Observability — MSAP Observability와 CogentAI

MSAP Observability는 eBPF 기반 Zero-Instrument 시스템 관측, OpenTelemetry 기반 트레이스·로그·메트릭 통합, Continuous Profiling, APM 연계 기능을 결합한 통합 Observability 플랫폼입니다. 여기에 CogentAI(LLM 기반 운영 분석 엔진)를 결합하면 기존 모니터링 도구로는 감지하기 어려운 이상 패턴 탐지, 자연어 기반 운영 질의, RCA 자동화, 예측 기반 운영 등이 가능해집니다.

LLM 기반 Observability는 "데이터의 양"이 아니라 "데이터 간의 관계를 얼마나 이해하는가"가 핵심입니다. 즉, Telemetry(메트릭·로그·트레이스·프로파일링·이벤트)를 컨텍스트로 결합하고 LLM이 이를 지능적으로 해석함으로써 운영자에게 사실(Fact) → 원인(Cause) → 해결(Action) 흐름을 제공하는 구조입니다.

6.1 AI Native Observability와 CogentAI의 역할

전통적인 Observability는 시스템 상태를 이해하는 데 필수적인 메트릭, 로그, 트레이스 데이터를 제공했지만, 오늘날의 복잡한 클라우드 네이티브 환경에서는 단순히 데이터를 나열하는 것만으로는 충분하지 않습니다. 이러한 복잡성에 대응하기 위해 AI Native Observability라는 새로운 패러다임이 부상하고 있습니다. 이는 방대한 데이터를 수동으로 분석하던 방식에서 벗어나, 대규모 언어 모델(LLM)을 활용하여 텔레메트리 데이터를 자동으로 해석하는 접근 방식입니다. AI Native Observability는 "무엇(What)"이 일어났는지를 넘어, 장애가 "왜(Why)" 발생했는지에 대한 근본적인 질문에 답을 제시하며, IT 운영을 사후 대응에서 사전 예측 및 자동화 단계로 진화시키는 핵심 동력입니다.

6.1.1 LLM과 관측 데이터(Telemetry)의 결합 모델

AI Native Observability의 핵심은 신뢰할 수 있는 데이터와 뛰어난 추론 능력의 결합에 있습니다. MSAP Observability는 Kubernetes와 같은 복잡한 분산 환경에서 발생하는 모든 원격 측정 데이터, 즉 메트릭(Metrics), 로그(Logs), 트레이스(Traces)를 높은 충실도(high-fidelity)로 수집하도록 설계되었습니다. 특히 'Zero-Instrument Observability' 접근 방식을 통해 애플리케이션 코드 수정 없이 데이터를 수집하므로, 빠르고 포괄적인 가시성을 확보합니다.

이러한 높은 충실도의 데이터는 기존 시스템에서 흔히 발생하는 샘플링 격차나 집계 오류가 없어, CogentAI에 왜곡되지 않은 사실 기반(ground truth)을 제공합니다. 이를 통해 LLM은 실제 이상 징후와 데이터 수집 과정의 노이즈를 명확히 구분하여 결정론적 분석을 수행하고, 오탐지를 획기적으로 줄일 수 있습니다. 이 결합 모델의 중심에는 지능형 운영 플랫폼인 VibeOps 프레임워크가 있으며, 특수화된 프로토콜인 MCP(Model Context Protocol)를 사용하여 실시간 운영 데이터를 CogentAI에 공급합니다. 이를 통해 CogentAI는 단순한 데이터 상관 분석을 초월하여, 시스템의 동적인 상태 변화에 대한 깊이 있는 맥락 인지(Context-aware) 분석을 수행할 수 있습니다.

LLM 기반 Observability는 단순히 "로그를 요약해주는 도구"가 아니라, MSAP Observability가 수집한 다층 Telemetry를 LLM이 하나의 의미적 모델로 통합하는 방식으로 동작합니다.

구성 요소

1. MSAP Observability Telemetry Layer
    - eBPF 기반 호출 관계·RTT·TCP 지연·I/O Wait·스레드 상태
    - OpenTelemetry 기반 메트릭·로그·트레이스
    - Continuous Profiling(플레임그래프) 데이터
    - APM(Java 세부 메소드·SQL 쿼리)
    - SLO·Incident 정보
2. CogentAI LLM Reasoning Layer
    - Telemetry의 시간·서비스·호출관계·리소스 소비 패턴을 LLM Context로 결합
    - 모델 내부에서 Root Cause 후보군을 생성하는 Reasoning Graph 수행
    - 비정상 이벤트의 Prioritization(위험도 점수화)
3. Operational Knowledge Graph (OKG)
    - 과거 인시던트
    - 배포 이벤트
    - 아키텍처 Topology
    - 서비스 의존성 맵(CCR: Call Chain Relationship)
    - 사용자 정의 Runbook

LLM은 단순 자연어 이해가 아니라 Telemetry → 의미적 Context로 변환 → 원인 탐색 → 요약/판단 흐름을 수행합니다.

LLM 결합의 이점

- 서로 다른 Telemetry 간 연결 관계를 자동 생성
- 비정상 패턴(Spike, Regression, Drift)을 LLM이 앞뒤 맥락으로 판단
- 단일 이벤트가 아닌 복합적 조건에서 발생한 장애 탐지
- 쿠버네티스·MSA 구조에서 발생하는 다회성 연쇄 장애 파악 가능

6.1.2 자연어 질의를 통한 메트릭·로그·트레이스 검색 및 분석

AI Native Observability가 제공하는 가장 혁신적인 변화 중 하나는 운영자와 시스템 간의 상호작용 방식을 근본적으로 바꾸는 것입니다. 더 이상 운영자는 PromQL과 같은 복잡한 쿼리 언어를 학습하거나 여러 대시보드를 오가며 데이터를 수동으로 조합할 필요가 없습니다. 대신, VibeOps 환경에서 자연어 기반의 대화형 인터페이스를 통해 시스템의 상태를 직관적으로 질문하고 즉각적인 답변을 얻을 수 있어 운영 효율성을 극적으로 향상시킵니다.

예시 1: 성능 병목 구간 신속 식별

사용자 질의: “지난 10분간 가장 응답 시간이 느렸던 애플리케이션 5개를 알려줘.”

CogentAI 분석 프로세스:

1. 자연어 질의의 의도(“가장 느린 앱”, “상위 5개”, “지난 10분”)를 파악합니다.
2. MCP를 통해 MSAP Observability의 트레이스 및 메트릭 저장소에 질의를 실행합니다.
3. 지정된 시간 범위 내에서 p95, p99 응답 시간(Latency) 메트릭이 가장 높은 상위 5개의 서비스를 식별합니다.
4. 분석 결과를 표 또는 자연어 요약 형태로 사용자에게 즉시 제공합니다.

예시 2: 특정 오류의 근본 원인 요약

사용자 질의: “결제 서비스 트랜잭션에서 발생한 NullPointerException의 원인을 요약해줘.”

CogentAI 분석 프로세스:

1. ‘결제 서비스’, ’NullPointerException’이라는 핵심 키워드를 추출합니다.
2. MSAP Observability의 로그 저장소에서 해당 서비스의 `NullPointerException` 로그를 검색합니다.
3. 동일한 `trace_id`를 가진 분산 트레이스 데이터를 조회하여, 예외가 발생한 정확한 코드 경로와 당시의 요청 파라미터를 식별합니다.
4. 관련 메트릭(예: 에러율, CPU 사용량)을 함께 분석하여 시스템 전반의 영향을 파악합니다.
5. “결제 서비스의 특정 코드 라인에서 외부 PG사로부터의 응답 데이터가 null일 때 예외가 발생했으며, 이는 최근 배포된 코드 변경과 관련이 있을 가능성이 높습니다.” 와 같이 원인을 종합적으로 요약하여 답변합니다.

MSAP Observability의 통합 데이터 모델이 제공되므로, LLM은 쿼리 언어 PromQL·LogQL·ClickHouse SQL을 몰라도 운영자가 필요한 정보를 반환합니다.

6.1.3 쿼리 언어 학습 없이 대화형으로 인사이트 도출

자연어 기반의 대화형 인터페이스는 단순히 편의성을 높이는 것을 넘어, IT 운영의 민주화를 실현하는 전략적 가치를 지닙니다. 과거에는 시스템의 심층 분석이 소수의 숙련된 SRE(사이트 신뢰성 엔지니어) 전문가에게만 가능한 영역이었지만, 이제는 신입 엔지니어나 다른 직무의 담당자도 복잡한 시스템에 대해 의미 있는 질문을 던지고 인사이트를 얻을 수 있게 되었습니다.

전통적인 Observability는 PromQL/LogQL/SQL 숙련도가 필요합니다.

CogentAI는 이를 대체하여 다음을 수행합니다.

- 자연어 → 질의 DSL 자동 변환
- 결과 데이터셋을 시각화 가능한 형태로 집계
- LLM이 직접 “이상 패턴”을 설명
- 운영자 질문 기반 Drill-Down

예)

“이 서비스의 CPU Spike가 왜 발생했는지 알려줘”

→ LLM은 다음 데이터를 결합해 설명합니다.

- 동일 시점 Pod 재시작
- 특정 API 트래픽 증가
- GC 정지 시간 증가
- RTT 지연 또는 TCP 재전송
- 직전 배포 이력
- Storage I/O Wait 증가

운영자는 명령어·대시보드 구성 없이 “질문-답변-추가 분석” 흐름으로 운영에 집중할 수 있습니다.

이는 장애 해결 시간(MTTR) 단축, 운영 교육 비용 절감, 데이터 기반 의사결정 강화라는 실질적인 비즈니스 효과로 이어집니다. 결론적으로, CogentAI와 MSAP Observability의 결합은 단

순히 데이터를 보여주는 도구에서, 질문에 대한 답을 제공하는 지능형 플랫폼으로의 근본적인 전환을 의미합니다. 데이터의 민주화는 단순한 운영 개선이 아니라, 모든 엔지니어가 데이터 기반 의사결정을 자신 있게 내릴 수 있도록 지원함으로써 혁신 주기를 가속화하는 전략적 원동력입니다. 이러한 지능은 실시간 질의응답을 넘어, 장애 발생 시 근본 원인을 자동으로 분석하고 해결책까지 제시하는 수준으로 확장됩니다.

6.2 자동화된 근본 원인 분석(RCA) 및 해결 제안

분산 시스템에서 장애가 발생했을 때, 가장 많은 시간과 노력이 소요되는 과정은 근본 원인 분석(Root Cause Analysis, RCA)입니다. 수많은 서비스의 로그, 메트릭, 트레이스를 수동으로 비교하고 상관관계를 추론하는 작업은 매우 느리고, 인간의 실수에 취약하며, 숙련된 전문가의 직관에 의존하는 경우가 많습니다. AI 기반의 자동화된 RCA는 이러한 한계를 극복하고, 장애 대응 패러다임을 사람이 주도하는 반응적 조사에서 AI가 안내하는 자동화된 해결 워크플로우로 전환합니다. 본 섹션에서는 MSAP Observability가 수집한 풍부한 데이터를 CogentAI가 어떻게 분석하여 근본 원인을 식별하고, 구체적인 해결 방안까지 제시하는지 상세히 탐구합니다.

6.2.1 메트릭·로그·트레이스 패턴을 종합 분석하는 AI 기반 RCA

CogentAI의 가장 큰 강점은 Observability의 세 가지 기둥(메트릭, 로그, 트레이스)에서 발생하는 신호들을 단일 컨텍스트로 묶어 종합적으로 분석하는 능력에 있습니다. AI는 인간이 놓치기 쉬운 미세한 데이터 패턴과 상관관계를 식별하여, 높은 신뢰도를 가진 근본 원인 가설을 신속하게 도출합니다.

다음은 피크 타임의 플래시 세일(flash sale) 중에 발생한 장애에 대한 AI 기반 RCA 시나리오입니다.

- 메트릭에서 이상 징후 탐지: `checkout-service`(결제 서비스)의 `p99 latency`(99번째 백분위수 응답 시간) 메트릭에서 비정상적인 급증이 탐지되며, 이를 최초 장애 신호로 인지합니다.
- 트레이스를 통한 상관관계 분석: CogentAI는 해당 시간대의 분산 트레이스 데이터를 자동으로 분석하여, 외부 `payment-gateway`로 향하는 특정 다운스트림 gRPC 호출 구간에서 대부분의 지연이 발생하고 있음을 병목 지점으로 식별합니다.

- 로그를 통한 증거 확인: AI는 세워진 가설을 검증하기 위해 동일 시간대의 로그를 분석하고, `payment-gateway` 호출과 관련된 `PostgreSQL connection timeout` 에러 로그가 높은 빈도로 발생하고 있음을 확인합니다.

이처럼 다각적인 데이터를 종합 분석함으로써 CogentAI는 높은 확신도의 결론을 도출합니다. 이러한 분석 과정은 수동으로 진행할 경우 수십 분에서 수 시간이 걸릴 수 있지만, AI는 단 몇 초 만에 완료하여 장애 대응의 골든타임을 확보합니다.

6.2.2 에러 로그 설명 및 해결 가이드 자동 생성

복잡한 기술적 오류 메시지나 긴 스택 트레이스는 문제 해결의 핵심 단서이지만, 그 의미를 정확히 파악하는 것은 숙련된 개발자에게도 어려운 일일 수 있습니다. CogentAI는 이러한 기술적 데이터를 운영자가 즉시 이해하고 조치할 수 있는 실행 가능한 정보(Actionable Intelligence)로 변환합니다.

첫째, LLM은 암호와 같은 에러 메시지나 수백 줄에 달하는 Java 스택 트레이스를 파싱하여, 그 핵심 의미를 명확하고 간결한 자연어로 요약해줍니다. 둘째, VibeOps는 여기서 한 걸음 더 나아가, 조직 내부에 축적된 런북(Runbook)과 과거 장애 처리 이력과 같은 지식 베이스를 참조하여 구체적인 해결 가이드를 자동 생성합니다.

예시: `java.lang.OutOfMemoryError: Java heap space` 오류가 감지되었습니다. 원인은 `ProductCache` 컴포넌트의 메모리 누수일 가능성이 높습니다. 권장 조치: 내부 기술 문서 Runbook-75B에 따라, 분석을 위해 힙 덤프(heap dump)를 캡처하고 다음 유지보수 기간에 서비스의 롤링 리스타트(rolling restart)를 고려하십시오.

이러한 기능은 신입 엔지니어도 숙련된 전문가처럼 문제에 대응할 수 있도록 지원하며, 장애 처리 절차를 표준화하는 효과를 가져옵니다.

6.2.3 과거 인시던트 이력을 활용한 유사 사례 추천

VibeOps 플랫폼의 지능은 고정되어 있지 않으며, 발생하는 모든 장애와 해결 과정을 통해 지속적으로 학습합니다. 새로운 장애가 발생하면, CogentAI는 해당 장애의 특징(Signature), 즉 메트릭 이상 패턴, 트레이스 구조, 핵심 로그 메시지 등의 조합을 분석합니다. 그 후, 이 시그니처를 기반으로 과거 인시던트 데이터베이스를 검색하여 가장 유사한 과거 사례들을 추천합니다. 이 기능

은 검증된 해결 경로를 제시하여 시행착오를 줄이고, 소수 시니어 엔지니어의 경험에 의존하던 비공식적 지식을 시스템화하여 조직 전체의 자산으로 만듭니다.

MSAP Observability의 Incident 기능(SLO 기반 위반 탐지)과 연계하여 CogentAI는 다음을 지원합니다.

- 과거 동일 패턴의 인시던트 검색
- 과거 해결 조치 비교
- 유사 SLO 위반 패턴 매칭
- 유사 로그 패턴 추천

이 기능은 “반복되는 장애”에 대한 해결 속도를 크게 단축합니다.

과거로부터 배우고 현재를 분석하는 이러한 능력은 단순히 오늘의 장애를 해결하는 것을 넘어, 운영 성능의 미래를 예측하고 만들어가는 데 필요한 전략적 인텔리전스를 제공합니다.

6.3 사용자 친화적 리포팅 및 예측

지능형 Observability는 실시간 장애 대응이라는 전술적 가치를 넘어, IT 리더와 관리자가 데이터에 기반한 현명한 의사결정을 내릴 수 있도록 지원하는 전략적 운영 인텔리전스를 제공합니다. 본 섹션에서는 CogentAI가 MSAP Observability로부터 수집된 방대한 양의 원시 텔레메트리 데이터를 어떻게 가공하여, IT 리더를 위한 고수준의 요약 보고서와 전략적 계획 수립을 지원하는 예측 분석을 생성하는지 상세히 설명합니다.

6.3.1 일일·주간 운영 리포트 자동 요약

IT 관리자와 경영진은 시스템의 세부적인 기술 지표보다 전반적인 운영 상태와 핵심적인 변화 추이에 더 큰 관심을 가집니다. CogentAI는 매일 또는 매주 수집된 방대한 성능 데이터를 자동으로 종합 분석하여, 기술적 지식이 없는 사람도 쉽게 이해할 수 있는 자연어 형태의 이그제큐티브 서머리(Executive Summary)를 생성합니다.

이 자동 요약 리포트는 일반적으로 다음과 같은 핵심 요소들을 포함합니다.

- Overall Health Summary: 시스템 안정성 및 성능에 대한 최상위 수준의 평가.

- Key Performance Trends: 트랜잭션 양, 응답 시간, 에러율과 같은 핵심 지표의 중요한 변화 분석.
- Incident Breakdown: 해당 기간 동안 발생한 인시던트의 총 횟수와 심각도별 요약.
- Top Problem Areas: 가장 많은 에러를 유발했거나 성능 저하의 주된 원인이었던 상위 서비스 또는 트랜잭션 식별.

6.3.2 리소스 사용량·트래픽 추세 예측을 통한 용량 산정 지원

비용 효율적인 서비스 운영의 핵심은 미래의 수요를 예측하고 이에 맞춰 자원을 선제적으로 확보하는 것입니다. CogentAI는 MSAP Observability가 장기간 축적한 과거 리소스 사용률 및 트래픽 데이터를 기반으로 시계열 예측 모델을 구축하고, 이를 통해 IT 의사결정자가 즉시 행동에 옮길 수 있는 구체적인 권고안 형태로 미래의 자원 수요를 예측합니다.

예시: "과거 6주간의 트래픽 패턴을 분석한 결과, 다가오는 연말 프로모션 기간 동안 사용자 활동이 약 30% 급증할 것으로 예측됩니다. 현재의 서비스 수준 협약(SLA)을 안정적으로 유지하기 위해, 프로모션 시작 전 '상품 추천(product-recommendation)' 서비스의 복제본(replica) 수를 3개에서 5개로 선제적으로 확장할 것을 권장합니다."

이러한 기능은 트래픽 급증으로 인한 값비싼 서비스 장애를 예방하고, 불필요한 자원 낭비를 줄이는 사전 예방적 용량 계획(Proactive Capacity Planning)을 가능하게 합니다.

6.3.3 SLA·SLO 관점에서의 서비스 품질 리포트 생성

기술적인 메트릭 자체는 비즈니스에 미치는 영향을 직접적으로 보여주지 못합니다. 중요한 것은 이러한 기술 지표가 고객과의 약속인 서비스 수준 협약(SLA)과 내부 목표인 서비스 수준 목표(SLO)를 얼마나 잘 충족하고 있는지를 평가하는 것입니다. MSAP Observability는 가용성, 응답 시간 등 핵심 SLO 지표를 지속적으로 추적하며, CogentAI는 이 데이터를 비즈니스 관점의 품질 리포트로 변환합니다.

- 기술적 메트릭: "p95 응답 시간은 350ms였습니다."
- 비즈니스 인사이트 리포트: " '사용자 로그인(user-login)' 서비스는 성공적인 HTTP 2xx 응답을 기준으로 하는 99.95%의 가용성 SLO를 성공적으로 충족했습니다. 그러나 응답 시간 SLO는 위반하여, p95 레벨에서 측정된 전체 요청의 98.5%만이 목표 시간인 250ms

이내에 처리되었습니다 (SLO 목표: 99%). 이 위반의 주된 원인은 오전 피크 시간대에 발생한 데이터베이스 잠금 경합이 40% 증가했기 때문입니다."

이처럼 구체적인 원인 분석이 포함된 리포트는 단순히 목표 달성 여부를 넘어, 서비스 품질 개선을 위한 실행 계획 수립에 결정적인 근거를 제공합니다. 이러한 분석 및 예측 기능은 IT 운영을 자동화하는 더 큰 비전의 일부를 구성합니다.

6.4 VibeOps 연계 운영 자동화

지금까지 논의된 LLM 기반의 심층 분석, 자동화된 RCA, 그리고 예측적 리포팅 기능은 모두 IT 운영의 미래를 향한 하나의 큰 비전을 구성하는 요소들입니다. MSAP.ai 플랫폼은 이러한 패러다임 전환을 대표합니다. 이 플랫폼 내에서 MSAP Observability는 시스템의 모든 상태를 파악하는 중추 신경계 역할을 하며 데이터를 제공하고, CogentAI로 구동되는 VibeOps는 그 데이터를 해석하고 행동을 결정하는 지능적인 두뇌 역할을 수행합니다. 본 마지막 섹션에서는 이러한 구성 요소들이 어떻게 유기적으로 연계되어, 데이터 수집부터 잠재적인 자동 복구에 이르는 완전한 자동화 워크플로우를 완성하는지 설명합니다.

6.4.1 Observability 데이터를 활용한 VibeOps 자동 분석 플로우

장애 발생 시 VibeOps는 사전에 정의된 워크플로우에 따라 체계적이고 자동화된 분석을 수행하여 인간의 개입을 최소화하고, 데이터에 기반한 일관된 분석을 보장합니다.

1. 자동 탐지 및 요약 (Automated Detection & Summarization): MSAP Observability에서 심각한 SLO 위반이 감지되면 즉시 VibeOps 워크플로우가 트리거됩니다. CogentAI는 즉각적으로 초기 영향 범위(blast radius)를 분석하고 상황을 요약하여 전파합니다.
2. 심층 상관관계 분석 (Deep Correlation Analysis): VibeOps는 장애 발생 시점을 중심으로 관련된 모든 메트릭, 트레이스, 로그 데이터를 자동으로 쿼리하고 시계열로 정렬합니다. 이 과정을 통해 흩어져 있던 개별 데이터들이 하나의 완전한 장애 컨텍스트로 통합됩니다.
3. 근본 원인 식별 (Root Cause Identification): CogentAI는 통합된 컨텍스트 데이터를 분석하여 잘못된 배포, 데이터베이스 병목, 외부 종속성 장애 등 가장 확률 높은 근본 원인을 특정합니다.

4. 실행 가능한 권고안 생성 (Actionable Recommendation Generation): 마지막으로, CogentAI는 명확한 자연어로 작성된 해결 권고안을 생성합니다. 이 권고안에는 분석의 근거가 된 데이터(증거)가 함께 제시되며, 관련 내부 런북을 참조하여 구체적인 해결 절차를 안내합니다.

6.4.2 자연어 명령 기반 운영 작업(조회·분석·요약) 자동화

VibeOps의 자연어 인터페이스는 단순한 질의응답(Q&A)을 넘어, 복잡한 다단계 운영 작업을 자동화하는 명령 체계로 기능합니다. 운영자는 단 하나의 자연어 명령을 통해, 앞서 설명한 전체 분석 워크플로우를 실행하고 그 결과를 문서화된 형태로 받을 수 있습니다.

예시: "VibeOps, 마지막 배포 이후 '배송 서비스(shipping-service)'에서 발생한 성능 저하를 조사하고, 근본 원인을 식별한 뒤 사후 보고서(post-mortem report) 초안을 생성해줘."

이 명령 하나로 VibeOps는 배포 시점 이후의 모든 관련 텔레메트리 데이터를 수집 및 분석하여 성능 저하의 원인을 찾고, 그 결과를 바탕으로 구조화된 사후 보고서 초안을 생성하여 운영자에게 전달합니다. 이는 운영자의 수작업을 최소화하고 장애 대응과 문서화 작업을 동시에 처리하여 생산성을 극대화합니다.

6.4.3 MCP 기반 운영 API 호출과의 연계 방향(정책·자동 조치 등)

VibeOps의 최종 목표는 분석과 제안을 넘어, 시스템이 스스로를 치유하는 폐쇄 루프 자동화(Closed-loop Automation)를 구현하는 것입니다. 이 비전의 핵심 기술적 기반은 MCP(Model Context Protocol)입니다. MCP는 LLM이 데이터를 분석하는 것을 넘어, 분석 결과를 기반으로 운영 API를 호출하여 실제 시스템에 조치를 취할 수 있도록 연결하는 중요한 다리 역할을 합니다.

이러한 연계를 통해 다음과 같은 미래의 자동화 시나리오를 구현할 수 있습니다.

- 정책 기반 자동화 (Policy-Based Automation): 사전에 정의된 정책에 따라 제한된 범위의 자동 조치를 허용합니다. 예를 들어, "만약 VibeOps가 특정 서비스에서 반복적으로 발생하는 비-치명적 메모리 누수를 식별하면, 정책에 따라 다음 승인된 유지보수 기간에 자동으로 롤링 리스타트를 예약하도록 허용한다"와 같은 정책을 설정할 수 있습니다.
- 지능형 자동 복구 (Intelligent Automated Remediation): 더 발전된 형태의 '자가 치유(self-healing)' 시나리오입니다. 예를 들어, "VibeOps가 과거에 발생하여 검증된 런북으

로 해결된 이력이 있는 인시던트와 높은 신뢰도로 일치하는 시그니처를 가진 새로운 장애를 탐지했을 경우, 승인 하에 관련 복구 스크립트를 운영 API를 통해 자동으로 실행한다"는 시나리오입니다. 이는 이미 알려진 문제에 대한 MTTR을 획기적으로 단축시킵니다.

본 장에서는 LLM 기반 Observability가 단순 모니터링의 확장 기능이 아니라,MSA·쿠버네티스의 복잡성을 AI Reasoning으로 완화하는 핵심 기술이라는 점을 강조했습니다.

MSAP Observability가 제공하는 풍부한 Telemetry와 MSAP.ai·CogentAI의 LLM 분석 엔진은 다음을 가능하게 합니다.

- 자연어 기반 운영 질의
- RCA 자동화
- 예측 기반 장애 예방
- 인시던트 요약 및 Runbook 자동 생성
- MCP 기반 운영 자동화(VibeOps)

전통적인 모니터링이 데이터를 보여주는 역할이었다면, AI Native Observability는 데이터를 해석하고 조치까지 연결하는 운영 지능화입니다.

제7장. DIY 모니터링 스택 대비 MSAP Observability의 경쟁 우위

7.1. DIY(Do It Yourself) 모니터링 스택 구축·운영의 한계

클라우드 네이티브 환경이 IT 인프라의 표준으로 자리 잡으면서, Prometheus, Grafana와 같은 강력한 오픈소스를 활용하여 자체 관측성(Observability) 스택을 구축하는 DIY 접근법은 많은 IT 의사결정자에게 매력적인 선택지로 여겨집니다. 초기 라이선스 비용이 없다는 점은 분명한 장점처럼 보입니다. 그러나 이러한 초기 비용 절감의 이면에는 운영 복잡성, 예측 불가능한 숨겨진 비용, 그리고 장기적인 기술 부채라는, 간과하기 쉬우나 비즈니스에 치명적인 영향을 미칠 수 있는 기술

부채의 씨앗들이 존재합니다. 본 섹션에서는 DIY 모니터링 스택이 엔터프라이즈 환경에서 마주하게 되는 근본적인 한계를 심층적으로 분석하고, 왜 통합된 상용 솔루션이 전략적으로 더 우월한 선택이 될 수 있는지 논하고자 합니다.

7.1.1. 개별 메트릭 수집 도구·로그 플랫폼·대시보드 도구 조합의 복잡성

오픈소스를 활용한 DIY 관측성 플랫폼 구축의 가장 큰 난관은 개별 도구들의 '조합' 그 자체에서 비롯됩니다. CNCF(Cloud Native Computing Foundation) 생태계에는 메트릭을 위한 Prometheus, 로그를 위한 Loki, 트레이스를 위한 Jaeger 및 Tempo, 그리고 이 모든 것을 시각화하는 Grafana 등 각 목적에 특화된 훌륭한 프로젝트들이 존재합니다. 그러나 이들은 처음부터 하나의 통합된 제품으로 설계된 것이 아닌, 독립적으로 발전해 온 개별 컴포넌트들입니다. 이들을 엮어 엔터프라이즈급 관측성 플랫폼을 만드는 과정은 상당한 기술적 복잡성과 운영 부담을 수반합니다.

- 파편화된 구성 및 관리 각 컴포넌트(Prometheus, Loki, Tempo 등)는 개별적으로 설치, 구성, 버전 관리가 필요합니다. 운영팀은 여러 시스템의 설정 파일을 각각 관리해야 하며, Grafana 대시보드에서는 각 데이터 소스(Data Source)를 수동으로 연결하고 최적화해야 합니다. 이 과정은 복잡할 뿐만 아니라, 설정 오류나 버전 비호환성으로 인한 장애 발생 가능성을 내포하고 있어 지속적인 관리 부담을 야기합니다.
- 데이터 상관관계의 부재 로그, 메트릭, 트레이스는 근본적으로 서로 다른 시스템과 데이터 저장소에서 수집되고 관리됩니다. 이는 특정 장애 상황이 발생했을 때 심각한 문제로 이어집니다. 메트릭 대시보드에서 이상 징후를 발견하더라도, 해당 시점의 로그와 트레이스를 찾기 위해 운영자는 여러 시스템을 오가며 컨텍스트를 수동으로 재구성해야 합니다. 이는 마치 모든 장애 발생 후 디지털 포렌식 수사를 진행하며 귀중한 엔지니어링 시간을 낭비하는 것과 같으며, 그동안 서비스는 계속해서 저하된 상태로 방치됩니다.
- 높은 학습 곡선 DIY 스택을 효과적으로 운영하기 위해, 엔지니어링팀은 단일 솔루션이 아닌 여러 오픈소스 프로젝트 각각의 아키텍처, 설정 방법, 그리고 고유의 쿼리 언어(예: Prometheus의 PromQL, Loki의 LogQL)를 모두 학습해야 합니다. 이는 상당한 시간과 노력을 요구하며, 특정 도구에 대한 전문 지식을 가진 인력에 대한 의존도를 높입니다. 결과적으로 팀의 전반적인 운영 효율성이 저하되고, 신규 인력의 온보딩 과정 또한 길어질 수밖

에 없습니다.

결과적으로, DIY는 “도구들의 나열”에 불과하며 운영자가 원하는 형태의 상관관계 분석이나 통합 시각화는 추가 개발 없이는 구현하기 어렵다는 문제가 발생합니다.

7.1.2. 데이터 보존 주기·스토리지 확장·성능 튜닝 부담

클라우드 네이티브 환경에서 생성되는 관측성 데이터의 양은 폭발적으로 증가하며, 이는 DIY 스택 운영에 막대한 재정적, 기술적 압박을 가합니다. 데이터의 수집, 저장, 쿼리에 이르는 전 과정에서 발생하는 비용과 성능 문제는 DIY 스택의 지속 가능성을 위협하는 핵심 요인입니다. 업계에서는 관측성 데이터 저장 비용이 기본 인프라 비용을 초과하는 사례가 빈번하게 논의될 정도로, 이는 DIY 스택의 가장 큰 재정적 함정입니다.

- 고비용 스토리지 의존성 빠른 검색과 분석 성능을 보장하기 위해, 대부분의 오픈소스 로깅 및 메트릭 시스템은 수집된 데이터를 인덱싱하고 고가의 고성능 스토리지에 보관합니다. 데이터 양이 증가함에 따라 이 인덱싱과 핫 스토리지에 드는 비용은 기하급수적으로 늘어나, 때로는 기본 인프라 비용을 초과하는 상황까지 발생합니다.
- 데이터 관리의 복잡성 스토리지 비용을 최적화하기 위해서는 정교한 데이터 관리 전략이 필수적입니다. 운영팀은 데이터의 중요도와 사용 빈도에 따라 보존 주기를 설정하고, 직접 구축하고 관리해야 합니다. 또한, 방대한 로그 데이터를 경량의 메트릭으로 변환하여 저장 공간을 절약하는 등의 최적화 작업을 수동으로 구현하고 유지보수해야 하는 운영 부담을 안게 됩니다.
- 지속적인 성능 튜닝 데이터 수집량이 임계점을 넘어서면 Prometheus, Loki 등의 각 컴포넌트는 성능 저하를 겪게 됩니다. 이를 방지하기 위해 운영팀은 지속적으로 각 컴포넌트의 설정을 튜닝하고, 샤딩(Sharding)이나 클러스터링을 통해 수평적으로 확장하는 작업을 수행해야 합니다. 이는 해당 오픈소스에 대한 깊은 이해를 가진 전문 인력의 개입을 요구하는 고비용 작업이며, 실패할 경우 데이터 유실이나 전체 모니터링 시스템의 마비로 이어질 수 있습니다.

반면 MSAP Observability는 대규모 컬럼형 데이터 엔진을 채택하여,

- OTel Log·Trace를 컬럼형 DB에 저장

- 고압축률(7~11배), 고성능 대량 ingestion, 빠른 쿼리 성능을 제공

  (OpenTelemetry ClickHouse Exporter 표준 지원)

→ 운영팀은 “스토리지 확장·색인·보존주기 정책”에 대한 별도 튜닝 없이 바로 사용 가능합니다.

7.1.3. 컴포넌트 간 버전 호환성·보안 취약점 관리 부담

여러 독립적인 오픈소스 프로젝트를 조합한 DIY 스택은 장기적인 유지보수와 보안 관리 측면에서 구조적인 리스크를 안고 있습니다. 초기 구축의 성공에 가려져 간과하기 쉽지만, 이러한 리스크는 시스템의 안정성과 보안 컴플라이언스에 치명적인 영향을 미칠 수 있습니다.

- 버전 호환성 문제 (일명 ‘Dependency Hell’) DIY 스택은 여러 컴포넌트가 복잡한 의존성 관계로 얽혀있는 구조입니다. 예를 들어, Grafana의 새로운 버전을 도입할 때, 해당 버전이 특정 Prometheus 데이터 소스 플러그인이나 Loki의 API와 호환되지 않을 수 있습니다. 하나의 컴포넌트를 업데이트하는 결정이 전체 시스템의 안정성을 해치거나, 다른 컴포넌트와의 호환성 문제로 인해 필요한 업그레이드가 무기한 지연되는 ’의존성 지옥(Dependency Hell)’에 빠질 수 있습니다. 이는 결국 최신 기능 도입의 지연과 시스템 노후화로 이어집니다.
- 분산된 보안 관리 각 오픈소스 프로젝트는 저마다의 개발 주기와 보안 정책을 가집니다. 이는 보안 관리의 책임이 전적으로 스택을 운영하는 기업에게 있음을 의미합니다. 운영팀은 Prometheus, Grafana, Loki 등 스택을 구성하는 모든 개별 프로젝트의 보안 취약점(CVE) 공지를 지속적으로 추적해야 합니다. 새로운 패치가 발표될 때마다, 각 컴포넌트에 대해 신속하게 테스트를 진행하고 운영 환경에 적용하는 작업을 반복해야 합니다. 이러한 분산된 보안 관리 체계는 신속한 대응을 어렵게 만들며, 엔터프라이즈 환경에서 요구하는 엄격한 보안 컴플라이언스 요구사항을 충족하기 어렵게 만드는 핵심 요인입니다.

반면 MSAP Observability는:

- 통합된 릴리즈 과정에서 모든 컴포넌트 호환성을 사전 검증
- OpenShift/RKE2/Kubernetes 기준 운영 안정성 확보

- MSAP COP와의 통합 운영 지원

→ DIY 대비 운영 리스크가 크게 감소합니다.

DIY 스택은 이처럼 내재된 복잡성, 숨겨진 비용, 그리고 지속적인 관리 부담이라는 명백한 한계를 가집니다. 이러한 이유로 수많은 엔터프라이즈 환경에서는 초기 투자 비용이 들더라도, 장기적인 안정성과 총 소유 비용(TCO) 관점에서 보다 성숙하고 통합된 솔루션이 필요합니다. 다음 장에서는 이러한 엔터프라이즈의 요구사항을 충족시키기 위해 설계된 MSAP Observability에 대해 알아보겠습니다.

7.2. Enterprise-Ready 솔루션으로서의 MSAP Observability

앞서 분석한 DIY 스택의 본질적인 한계—파편화된 관리, 데이터 사일로, 높은 운영 비용, 그리고 보안 및 유지보수의 복잡성—는 엔터프라이즈 환경에서 관측성 플랫폼이 갖춰야 할 요건이 무엇인지를 명확하게 보여줍니다. MSAP Observability는 이러한 도전 과제들을 직접적으로 해결하기 위해 설계된 솔루션입니다. 이는 단순히 여러 오픈소스 도구를 조합한 것이 아닌, 기업 환경의 안정성, 성능, 그리고 운영 효율성이라는 핵심 가치를 충족시키기 위해 처음부터 통합된 플랫폼으로 구상되었습니다. 본 섹션에서는 MSAP Observability가 어떻게 즉각적인 가치 창출, 고성능 아키텍처, 그리고 전문적인 지원 체계를 통해 엔터프라이즈급(Enterprise-Ready) 솔루션으로서의 경쟁 우위를 확보하는지 상세히 분석합니다.

> DIY가 단일 기능성 모니터링 도구라면, MSAP Observability는 엔터프라이즈 MSA 운영 전체를 수용하는 완성형 Observability 제품입니다.

7.2.1. 설치 즉시 사용 가능한 통합 대시보드·프리셋 제공

MSAP Observability는 복잡한 설정과 통합 과정 없이 설치 즉시 가치를 제공하는 통합된 사용자 경험을 최우선으로 설계되었습니다. DIY 스택이 각 데이터 소스를 수동으로 연결하고 대시보드를 처음부터 구성해야 하는 것과 달리, MSAP Observability는 엔터프라이즈 운영에 필수적인 기능들을 사전 탑재하여 제공함으로써 관측성 확보에 필요한 시간과 노력을 획기적으로 단축시킵니다.

MSAP Observability는 MSAP COP·RKE2·OpenShift·Kubernetes 환경에 배포 즉시 다음 요소를 자동 구성합니다.

- 클러스터·노드·Pod 자원 지표
- 서비스 토폴로지 맵
- 애플리케이션 호출 지연·에러율·SLO 모니터링
- 배포 영향 분석(롤아웃 후 성능 변화 자동 감지)
- 로그 패턴 분석 및 오류 집계
- 분산 추적(OTel 기반)
- 코드 레벨 프로파일링(Flamegraph)

이는 Kubernetes 에 특화된 모니터링 대시보드와 OPENMARU APM 통합 설계에 근거합니다.

- 통합된 메인 대시보드 설치와 동시에 시스템에서 모니터링되는 모든 애플리케이션의 상태, 핵심 성능 지표(응답 시간, CPU/메모리 사용량), 오류 상황을 한 화면에서 종합적으로 파악할 수 있는 메인 대시보드를 기본으로 제공합니다. 이는 운영자가 시스템의 전반적인 상태를 즉시 파악하고 이상 징후를 신속하게 인지하여, 비즈니스에 영향을 미치기 전에 선제적으로 대응할 수 있음을 의미합니다.
- 자동화된 서비스 토폴로지 맵 eBPF 기술을 기반으로, 별도의 설정 없이 서비스 간의 호출 관계와 의존성을 자동으로 탐지하여 실시간 토폴로지 맵을 시각화합니다. 복잡한 마이크로서비스 아키텍처(MSA) 환경에서 어떤 서비스가 다른 서비스에 영향을 미치는지, 트래픽 병목이 어디에서 발생하는지를 직관적으로 이해할 수 있게 해줍니다. 이는 장애 발생 시 영향 범위를 수 분 내에 파악하고 근본 원인을 추적하여 비즈니스 손실을 최소화하는 의사결정을 지원합니다.
- 내장된 인시던트 및 SLO 관리 서비스 수준 목표(SLO) 위반을 자동으로 감지하고 인시던트(Incident)를 생성 및 관리하는 기능이 플랫폼에 내장되어 있습니다. 가용성, 응답 시간 등 핵심 지표에 대한 SLO를 설정하면, 시스템이 이를 지속적으로 추적하여 위반 시 즉시 인시던트를 발생시킵니다. 이를 통해 운영팀은 문제 상황을 놓치지 않고 신속하게 장애 대응 프로세스를 시작할 수 있습니다.

특히 한눈에 전체 MSA를 보는 토폴로지 맵과 SLO 기반 인시던트는 기존 DIY 스택에서는 구현하기 어려운 기능입니다.

7.2.2. 대용량 데이터 처리에 최적화된 고성능 백엔드 구조

대규모 엔터프라이즈 환경에서 발생하는 방대한 양의 텔레메트리 데이터를 안정적으로 수집, 처리, 분석하기 위해서는 고성능 아키텍처가 필수적입니다.

MSAP Observability는 OpenTelemetry Collector와 컬럼형 DB를 기반으로 대용량 수집·저장·분석을 수행합니다.

- 컬럼형 DB는 20TB/day 로그 처리, 압축률 7~11배 수준의 성능을 검증
- 하나의 노드에서도 고성능 처리 가능하며 노드 추가 시 선형 확장
- eBPF 기반 커널 레벨 수집으로 언어·에이전트 무관 자동 수집

따라서 쿠버네티스 기반 대규모 MSA 환경에서도 수집 지연·쿼리 병목 없이 안정적인 운영이 가능합니다.

MSAP Observability는 최신 기술을 기반으로 설계된 효율적인 데이터 파이프라인과 확장 가능한 백엔드 구조를 통해 이 요구사항을 충족합니다.

- eBPF 기반의 효율적인 데이터 수집 'Zero-Instrument Observability' 철학의 핵심인 eBPF 기술을 활용하여, 애플리케이션 코드 수정이나 별도의 언어별 에이전트 설치 없이 리눅스 커널 레벨에서 데이터를 수집합니다. 이는 약 1%의 CPU와 250MB의 메모리라는 최소한의 오버헤드로 Java, Python, Go 등 다양한 언어로 작성된 애플리케이션의 성능 데이터를 투명하게 확보할 수 있게 해줍니다. 이는 곧 관측성 확보를 위한 인프라 비용이 비즈니스 성장을 저해하지 않는다는 것을 의미하며, 한정된 클라우드 예산을 핵심 서비스에 집중할 수 있게 만듭니다.
- 운영 환경을 위한 Continuous Profiling 기존 프로파일링 도구는 높은 오버헤드로 인해 운영 환경에서 상시 사용하기 어려웠습니다. 하지만 eBPF 기반의 Continuous Profiling 기능은 매우 낮은 오버헤드로 운영 환경에서도 지속적인 성능 분석을 가능하게 합니다. 이를 통해 시스템의 성능을 저하시키는 코드 레벨의 병목 지점이나 비효율적인 시스템 콜과 같은 근본 원인을 실제 트래픽 환경에서 정확하게 찾아낼 수 있습니다.

- 확장 가능한 백엔드 설계 MSAP Observability는 대규모 데이터의 저장, 고속 인덱싱, 그리고 복잡한 쿼리 처리를 위해 설계된 엔터프라이즈급 백엔드 아키텍처를 채택하여, 일부 오픈소스가 사용하는 단일 파일 DB(SQLite)의 확장성 한계를 근본적으로 극복합니다. 이는 데이터 양이 폭증하더라도 안정적인 성능을 유지하며, 필요에 따라 유연하게 확장할 수 있는 기반을 제공하여 미래의 성장까지 대비할 수 있게 합니다.

7.2.3. 국내 엔터프라이즈·공공 환경에 최적화된 기술 지원·커스터마이징

소프트웨어의 진정한 가치는 기술적 성능뿐만 아니라, 실제 운영 환경에서 발생하는 문제를 얼마나 신속하고 효과적으로 해결할 수 있는 지에 달려있습니다. MSAP Observability는 국내 기업 및 공공기관의 고유한 요구사항과 특성을 깊이 이해하고, 이에 최적화된 기술 지원 및 맞춤형 서비스를 제공하여 차별화된 가치를 창출합니다.

- 전문적인 국내 기술 지원 장애 발생 또는 기술적 문의 시, 커뮤니티 포럼이나 해외 개발자의 응답을 기다릴 필요 없이 국내의 전문 엔지니어로부터 신속하고 정확한 지원을 받을 수 있습니다. 언어와 시차의 장벽 없는 원활한 소통은 문제의 근본 원인을 빠르게 분석하고 해결책을 제시함으로써, 평균 해결 시간(MTTR)을 단축하고 운영 안정성을 극대화하는 핵심 자산입니다.
- 맞춤형 기능 개발 및 연동 국내 기업 환경은 고유의 내부 시스템이나 규제 및 보안 요구사항을 가지는 경우가 많습니다. MSAP Observability는 고객사의 필요에 따라 특정 기능을 맞춤형으로 개발하거나, 내부 데이터베이스를 이용한 SMS 알림 전송 시스템과 같이 기존 인프라와의 유연한 연동을 지원합니다. 이러한 유연성은 오픈소스 DIY 스택이나 외산 솔루션이 제공하기 어려운 핵심적인 경쟁력입니다.
- 한국어 지원 및 문서화 솔루션의 모든 사용자 인터페이스(UI)가 완벽한 한국어로 제공되며, 상세한 사용자 가이드와 기술 백서 또한 한국어로 작성되어 있습니다. 이는 국내 운영자의 학습 곡선을 크게 낮추고 솔루션의 모든 기능을 원활하게 활용할 수 있도록 돕습니다. 이를 통해 솔루션 도입 효과를 극대화하고, 운영팀의 역량을 강화하는 데 기여합니다.

또한 OPENMARU는 국내 고객사를 통한 운영 경험을 바탕으로 정책 기반 SLO 운영, 알림 채널(PagerDuty, Slack, Teams 등), RBAC, 배포 영향 분석 기능을 제공하여 엔터프라이즈 현장의 실제 요구사항을 충족합니다.

결론적으로 MSAP Observability는 개별 도구의 기능을 제공하는 것을 넘어, 복잡한 시스템 운영의 패러다임을 '사후 대응'에서 '사전 예방'으로 전환시키는 전략적 플랫폼입니다. 이러한 엔터프라이즈급 기능들이 어떻게 총 소유 비용(TCO) 절감과 측정 가능한 비즈니스 가치 창출로 직접 연결되는지 다음 장에서 구체적으로 분석하겠습니다.

7.3. 총 소유 비용(TCO) 및 비즈니스 효과

IT 투자의 궁극적인 목표는 기술적 우위를 넘어 실질적인 비용 효율성과 비즈니스 가치 창출로 이어져야 합니다. 이제 기업이 마주한 현실은 기술적 장점이 어떻게 기업의 재무 건전성과 생산성에 직접적인 영향을 미치는가에 달려있습니다. 본 섹션에서는 MSAP Observability 도입이 어떻게 초기 투자 비용 이상의 가치를 제공하는지, 즉 직접적인 비용 절감과 운영 혁신을 통해 측정 가능한 투자수익률(ROI)을 달성하는지를 총 소유 비용(TCO) 관점에서 심층적으로 분석할 것입니다.

7.3.1. 자체 구축·운영 인건비와 솔루션 도입 비용 비교

DIY 스택을 고려할 때 표면적으로 보이는 '무료' 라이선스는 큰 착시를 유발합니다. 안정적인 상용 환경을 구축하고 유지하기 위해 필요한 수많은 외부 구성요소와 전문 인력 비용을 모두 고려한 총 소유 비용(TCO) 관점에서 접근해야 합니다.

DIY 스택 운영에 필요한 인력:

역할	필요 인력	설명
Observability 엔지니어	1–2명	Prometheus/Loki/Jaeger/OTEL Collector 유지보수
스토리지 엔지니어	1명	Elasticsearch/Loki/Object Storage 튜닝
SRE 엔지니어	1명	SLO·Alerting·대시보드 설계
애플리케이션 계측 담당	1명	언어별 Instrumentation·OTel SDK 적용

DIY 접근법은 결국 여러 벤더의 기술을 조립한 '프랑켄슈타인 아키텍처'와 개별적으로 관리해

야 하는 '조각난 라이선스' 문제로 귀결되어, 장기적으로는 더 높은 누적 비용을 발생시키는 경우가 많습니다.

비용 항목	DIY(오픈소스 조합) 스택	MSAP Observability
라이선스 비용	초기 비용은 없으나, 엔터프라이즈급 기능을 위해 각 컴포넌트의 상용 버전(예: Grafana Enterprise)이나 필수 플러그인 도입 시 예측하기 어려운 추가 비용이 발생할 수 있습니다.	웹서버, APM, 클러스터링, 관측성 기능이 모두 포함된 단일 라이선스 체계를 통해 중복 투자를 원천적으로 방지하고 예측 가능한 비용 구조를 제공합니다.
구축 및 통합 인력 비용	Prometheus, Loki, Jaeger 등 각 분야의 오픈소스 전문가들이 컴포넌트를 개별적으로 설치, 구성, 통합하는 과정에 상당한 시간과 높은 초기 인건비가 소요됩니다.	통합된 플랫폼과 자동화된 설치 도구(Installer)를 통해 복잡한 구축 과정을 최소화하고, 투입되는 전문 인력과 기간을 획기적으로 단축시킵니다.
운영 및 유지보수 인력 비용	툴 파편화로 인해 각 도구별 전문가가 필요하며, 버전 호환성 관리, 보안 패치, 성능 튜닝 등 지속적인 유지보수 활동에 고급 엔지니어의 리소스가 상시 투입되어야 합니다.	단일화된 관리 포인트와 직관적인 UI를 통해 운영 복잡성을 크게 낮추고, 적은 인력으로도 효율적인 시스템 관리가 가능하여 장기적인 인건비를 절감합니다.
인프라 비용	여러 종류의 에이전트와 서버 컴포넌트를 실행하기 위해 추가적인 CPU, 메모리, 스토리지 자원이 필요하며, 이는 클라우드 환경에서 직접적인 비용 증가로 이어집니다.	eBPF 기반의 경량화된 단일 에이전트를 통해 시스템 리소스 소모를 최소화하고, 데이터 수집 파이프라인을 최적화하여 인프라 비용 부담을 줄입니다.

7.3.2. 장애 대응 시간 단축에 따른 비즈니스 손실 최소화

장애로 인한 서비스 중단은 곧 매출 감소, 브랜드 신뢰도 하락, 고객 이탈과 같은 직접적인 비즈니스 손실로 이어집니다. 따라서 장애 발생 시 평균 해결 시간(MTTR)을 얼마나 단축시킬 수 있는지는 관측성 플랫폼의 가치를 평가하는 가장 중요한 척도 중 하나입니다.

- 신속한 근본 원인 분석 MSAP Observability는 로그, 메트릭, 트레이스 데이터를 단일 플랫폼에서 유기적으로 통합하여 제공합니다. 장애 징후가 포착되면, 운영자는 여러 시스템을 오갈 필요 없이 클릭 몇 번으로 관련 데이터 간의 상관관계를 즉시 분석할 수 있습니다. 특히, 자동화된 토폴로지 맵과 분산 추적 기능은 복잡한 서비스 호출 관계 속에서 병목 구간과 오

류 발생 지점을 직관적으로 식별하게 해줍니다. Elastic의 분석에 따르면, OpenTelemetry를 도입한 기업들은 장애 해결 시간을 평균 40% 단축하는 효과를 보았으며, 이는 서비스 중단 시간을 최소화하여 비즈니스 손실을 직접적으로 방어하는 핵심적인 역할을 합니다.

- 장애 예측 및 예방 최고의 장애 대응은 장애가 발생하기 전에 예방하는 것입니다. MSAP Observability는 AI 기반 장애 예측 기능을 통해 사후 분석을 넘어 선제적 대응을 지원합니다. 과거의 성능 데이터와 이벤트 패턴을 학습한 AI 모델은 메모리 누수, 점진적인 성능 저하, GC 시간 증가 등 심각한 장애로 이어질 수 있는 미세한 이상 징후를 사전에 탐지하고 경고합니다. 이를 통해 운영팀은 실제 서비스에 영향이 미치기 전에 선제적으로 원인을 분석하고 조치함으로써, 잠재적인 비즈니스 손실을 원천적으로 차단할 수 있습니다.

MSAP Observability는 다음 기능을 통해 MTTR을 비약적으로 개선합니다:

- 자동 서비스 맵으로 장애 위치 즉시 파악
- eBPF 기반 네트워크 지연·재전송 지표 자동 수집
- 분산 트레이싱으로 서비스 호출 경로 즉시 분석
- Continuous Profiling으로 코드 병목 직접 확인
- 로그 패턴 자동 분석

이런 연계 분석 능력은 DIY 스택에서는 제공되지 않으며, PM·개발·운영팀이 문제를 추적하는 시간을 단축합니다.

7.3.3. 운영 인력·도구 수 감소에 따른 조직 효율성 향상

여러 개의 개별 도구를 조합해 사용하는 DIY 방식은 필연적으로 '툴 스프롤(Tool Sprawl)' 현상을 야기합니다. 이는 기술적 비효율을 넘어 조직 전체의 생산성을 저해하는 심각한 문제입니다. MSAP Observability는 툴 통합을 통해 DevOps 및 SRE 조직의 운영 효율성을 근본적으로 개선합니다.

- 툴 스프롤(Tool Sprawl) 해소 모니터링, 로깅, 트레이싱을 위해 사용하던 여러 도구를 하나의 통합 플랫폼으로 대체함으로써 관리 포인트를 획기적으로 줄일 수 있습니다. 이는 엔지니어들이 여러 도구를 넘나들며 발생하는 컨텍스트 스위칭 비용을 최소화하고, 중복된 라이선스 및 유지보수 계약을 제거하여 직접적인 비용 절감 효과를 가져옵니다.

- 운영 프로세스 표준화 개발, 운영, SRE 등 모든 관련 팀이 동일한 플랫폼과 데이터를 기반으로 소통하고 문제를 해결하게 됩니다. 이는 데이터 해석의 차이로 인한 불필요한 논쟁을 줄이고, 팀 간의 협업을 원활하게 만듭니다. 결과적으로 장애 분석, 성능 최적화, 신규 배포 검증 등 핵심 운영 프로세스가 자연스럽게 표준화되어 조직 전체의 업무 효율성이 향상됩니다.
- 고급 인력의 가치 극대화 가장 중요한 자산인 고급 엔지니어들이 파편화된 도구를 유지보수하고, 데이터를 수동으로 연관 분석하는 저부가가치 업무에서 해방됩니다. MSAP Observability는 엔지니어들을 끊임없이 발생하는 문제에 대응하는 '디지털 소방관'에서, 장애를 예방하고 성능을 최적화하는 '선제적인 시스템 아키텍트'로 변화시킵니다. 이는 단순히 비용 절감을 넘어, 기술 조직을 혁신과 가치 창출의 핵심 동력으로 만드는 전략적 전환입니다.

반면 MSAP Observability는:

- Metric·Log·Trace·Profiling·Topology·SLO·Incident를 단일 제품에서 제공
- APM과의 통합으로 Java 성능 분석까지 포함
- MSAP COP 및 MSAP.ai와 자연스럽게 연계되는 전체 플랫폼 구조

→ 운영팀, 개발팀, SRE팀이 단일 관점에서 동일한 데이터를 기준으로 협업할 수 있게 됩니다.

결론

본 장에서는 오픈소스를 조합한 DIY 모니터링 스택과 엔터프라이즈급 솔루션인 MSAP Observability를 다각도로 비교 분석했습니다. DIY 스택은 초기 비용이 없다는 매력적인 제안 이면에 파편화된 관리, 데이터 사일로, 예측 불가능한 누적 비용, 그리고 복잡한 유지보수라는 숨겨진 비용과 구조적 한계를 안고 있습니다. 이는 결국 더 높은 총 소유 비용(TCO)과 운영 비효율로 이어질 수밖에 없습니다.

반면, MSAP Observability는 설치 즉시 가치를 제공하는 통합 플랫폼, 대용량 데이터 처리에 최적화된 고성능 아키텍처, 그리고 전문적인 기술 지원을 통해 이러한 한계를 명확히 극복합니다. 이는 단순히 기술적 우위를 넘어, 장애 해결 시간(MTTR) 단축, 운영 효율성 향상, 그리고 고급 인력의 가치 극대화라는 측정 가능한 비즈니스 효과로 직접 연결됩니다.

결론적으로, 복잡성이 지배하는 현대 클라우드 네이티브 환경에서 성공적인 관측성 전략의 핵심은 단순히 여러 기술을 조합하는 기술적 과제가 아닙니다. 이는 운영 효율성과 비즈니스 목표 달성과 직결되는 통합 플랫폼을 선택하는 전략적 의사결정입니다. MSAP Observability는 이러한 전략적 선택에 있어 가장 신뢰할 수 있고 비용 효율적인 해답을 제시합니다.

제8장. SRE 기반 운영 모델과 MSAP 플랫폼 생태계 연동

클라우드 네이티브 환경에서 MSA 기반 서비스를 안정적으로 운영하기 위해서는 단순 지표 모니터링을 넘어 SLO·SLI 중심의 운영 모델과 자동화된 인시던트 대응 체계가 필수적입니다.

MSAP Observability는 이러한 SRE 운영 체계를 Kubernetes·MSA·AI Native 애플리케이션 전반에 적용할 수 있도록 설계된 플랫폼이며, MSAP COP·MSAP.ai·OPENMARU iAP APM과의 유기적 연동을 통해 운영 자동화를 실현합니다.

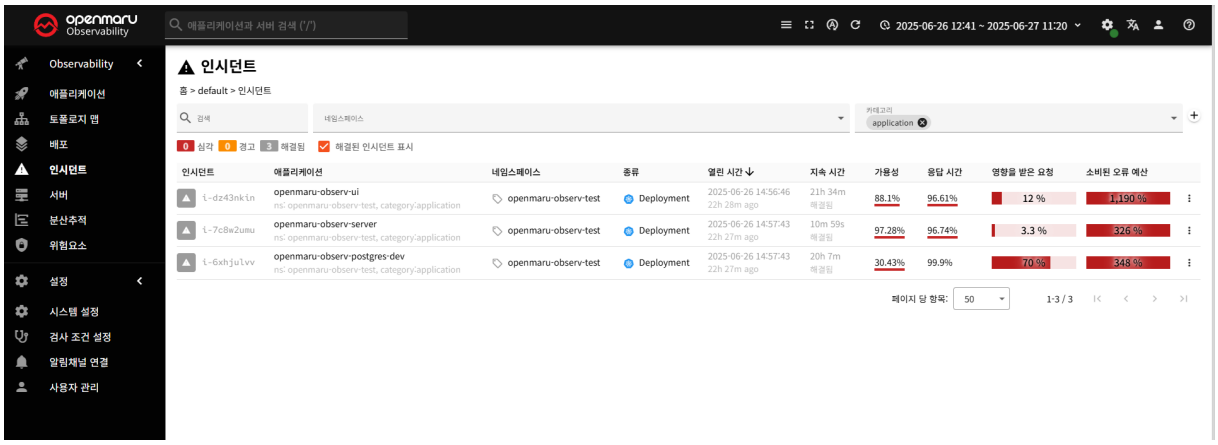
8.1. SLO·SLI·오류 예산 기반 운영 모델

8.1.1. 서비스 가용성·응답시간 지표 정의와 목표 설정

마이크로서비스 아키텍처(MSA)와 같은 현대적 분산 시스템에서, 전통적인 인프라 중심의 모니터링(CPU, 메모리 등)은 더 이상 서비스 품질을 보장하지 못합니다. 개별 시스템 리소스가 정상 범위에 있더라도, 특정 서비스의 미세한 응답 시간 저하가 연쇄적으로 파급되어 직접적인 매출 손실과 고객 신뢰도 하락으로 이어질 수 있기 때문입니다. 이는 비즈니스 리스크와 직결되는 문제입니다.

이러한 한계를 극복하기 위해 Google에서 시작된 사이트 신뢰성 엔지니어링(SRE) 방법론은 시스템의 내부 상태가 아닌, 최종 사용자의 경험에 초점을 맞춘 데이터 기반의 접근법을 제공합니다. SRE는 사용자 경험을 비즈니스와 연계된 측정 가능한 신뢰성 목표로 변환하는 체계적인 프레임워크를 제시하며, 그 핵심에는 서비스 수준 지표(SLI)와 목표(SLO)가 있습니다.

SRE 운영의 핵심은 모니터링 항목이 아니라 서비스 품질을 나타내는 SLI(Service Level Indicator) 와, 이를 기반으로 한 SLO(Service Level Objective) 입니다.



[그림 6] 인시던트 시스템은 SLO(Service Level Objective) 위반을 자동으로 감지하고 알림을 발송

MSAP Observability는 애플리케이션별로 다음 항목을 자동 수집하여 SLI로 사용합니다.

- 가용성(Availability): 성공한 요청 비율 기반 계산
- 응답시간(Latency): p50·p95·p99 백분위 기반 SLA 지연시간 분석
- 오류율(Error Rate)
- 트래픽(Throughput)

사용자는 각 서비스별로 가용성 SLO(기본 99%), 응답시간 SLO(기본 p95 기준) 를 설정하며, 이 임계값은 프로젝트·글로벌 단위에서 재정의할 수 있습니다.

이러한 구조는 Kubernetes 기반 MSA 환경에서 각 서비스의 품질을 정량적으로 관리하기 위한 필수 기반이며, 동적 스케일링/롤링업데이트 환경에서도 운영자가 서비스 품질을 안정적으로 통제할 수 있도록 합니다.

SRE 기반 운영 모델을 이해하기 위해서는 다음 두 가지 핵심 개념을 명확히 정의해야 합니다.

SLI (Service Level Indicator, 서비스 수준 지표) 서비스 품질의 특정 측면을 측정하는 정량적 지표입니다. 이는 시스템의 상태를 사용자의 관점에서 바라보는 구체적인 측정값으로, '요청 응답 시간(Latency)'과 '요청 성공률(Availability)'이 가장 대표적인 SLI의 예시입니다.

SLO (Service Level Objective, 서비스 수준 목표) 특정 기간 동안 달성하고자 하는 SLI의 목표치입니다. SLO는 막연한 기대치가 아닌, 측정 가능하고 달성 가능한 구체적인 목표여야 합니다. 예를 들어, "월간 전체 요청의 99.9%를 500ms 이내에 처리한다"와 같이 명확한 수치로 정의되어야 하며, 이는 서비스의 신뢰성 수준에 대한 팀과 비즈니스 이해관계자 간의 명시적인 약속이 됩니다.

MSAP Observability는 이러한 SRE의 핵심 원칙을 플랫폼에 내재화하여 제공합니다. 사용자는 'SLO' 탭에서 각 애플리케이션의 가용성(Availability) 및 응답 시간(Latency)에 대한 SLO를 직관적으로 설정하고 관리할 수 있습니다. 예를 들어, 가용성 목표를 99%로 설정하고, 응답 시간 목표를 95%의 요청이 250ms 이내에 처리되도록 설정한 후, 실시간으로 해당 목표의 준수율을 추적하고 시각화된 데이터를 통해 서비스 상태를 즉각적으로 파악할 수 있습니다.

SLO와 SLI를 설정하는 것이 안정적인 서비스 운영의 첫걸음이라면, 이 목표를 기준으로 혁신의 속도를 조절하고 데이터 기반의 의사결정을 내리는 데에는 '오류 예산(Error Budget)'이라는 개념이 핵심적인 역할을 합니다. 다음 섹션에서는 오류 예산을 어떻게 전략적으로 활용할 수 있는지 살펴보겠습니다.

8.1.2. 오류 예산(Error Budget)을 활용한 배포·변경 속도 관리

오류 예산(Error Budget)은 **100% - SLO 목표(%)**로 계산되는, 서비스가 허용할 수 있는 실패의 총량입니다. 예를 들어, 가용성 SLO가 99.9%라면, 0.1%의 요청은 실패해도 좋다는 것을 의미하며 이것이 바로 해당 기간 동안 사용할 수 있는 '오류 예산'이 됩니다. 이 개념은 "실패는 비정상이 아닌 정상적인 상태"라는 SRE의 핵심 철학을 반영합니다. 100% 완벽한 시스템을 추구하는 것은 기술적으로 비현실적일 뿐만 아니라 막대한 비용을 초래하므로, 비즈니스에 영향을 주지 않는 수준의 실패를 허용하고 이를 혁신을 위한 자원으로 활용하자는 전략적 접근입니다.

오류 예산의 가장 큰 가치는 개발팀의 혁신 속도와 운영팀의 안정성 요구 사이의 균형을 맞추는 데이터 기반 협상 프레임워크(data-driven negotiation framework)라는 점에 있습니다. 이는 신규 기능 출시와 안정성 확보라는 두 가지 목표를 정렬시키며, 더 이상 주관적인 판단이 아닌 공유된 데이터를 바탕으로 배포 및 변경 속도를 함께 조절할 수 있게 합니다.

오류 예산 상태	의미 및 대응 전략
예산이 충분할 경우	서비스가 SLO 목표를 초과 달성하며 안정적으로 운영되고 있음을 의미합니다. 개발팀은 이 '예산'을 활용하여 새로운 기능을 배포하고, A/B 테스트를 진행하며, 리팩토링을 수행하는 등 혁신적인 활동을 자유롭게 수행할 수 있습니다. 이는 안정성을 해치지 않는 선에서 비즈니스 가치를 빠르게 창출할 기회를 제공합니다.

예산이 소진될 경우	서비스 안정성이 SLO 목표치 이하로 떨어져 사용자 경험에 부정적인 영향을 미치고 있다는 명확한 신호입니다. 이 경우, 모든 팀은 신규 기능 배포를 동결(Freeze)하고, 소진된 오류 예산을 다시 확보하기 위해 안정성 개선 작업(버그 수정, 성능 최적화, 부하 테스트)에 역량을 집중해야 합니다.

오류 예산이 소진되거나 SLO가 목표치 이하로 떨어지는 것을 신속하게 감지하고, 이를 즉시 관련 담당자에게 알려 대응 프로세스를 시작하는 자동화된 흐름은 안정적인 서비스 운영의 필수 조건입니다.

8.1.3. SLO 위반 감지 및 자동 인시던트 생성 플로우

오류 예산이 소진되거나 SLO가 위반되었을 때, 이를 신속하게 감지하고 대응하는 것은 서비스 중단 시간을 최소화하는 데 결정적입니다. MSAP Observability는 이 과정을 자동화하여 운영팀의 부담을 줄이고 대응 속도를 극대화합니다.

MSAP Observability의 인시던트 시스템은 다음과 같은 자동화된 워크플로우를 통해 동작합니다.

- 실시간 SLO 준수율 모니터링: 플랫폼은 설정된 가용성 및 응답 시간 SLO를 기준으로 모든 애플리케이션의 상태를 24시간 실시간으로 추적합니다.
- 임계값 위반 자동 감지: SLO 준수율이 사전에 정의된 임계값 이하로 떨어지면, 시스템은 이를 즉시 이상 징후로 감지합니다.
- 인시던트 자동 생성 및 상태 변경: 감지된 SLO 위반에 대해 시스템은 자동으로 인시던트를 생성하고, 상태를 '심각(Critical)'으로 변경하여 문제의 중요도를 명시적으로 표시합니다.
- 지정된 알림 채널로 즉시 통보: 생성된 인시던트 정보는 사전에 설정된 알림 채널(예: Slack, MS Teams)로 즉시 전송되어 담당자가 문제 발생을 즉각 인지하도록 합니다.

이러한 자동화된 플로우는 장애를 인지하는 데 걸리는 평균 시간(MTTA, Mean Time to Acknowledge)을 획기적으로 단축시키는 효과를 가져옵니다. 더 이상 운영자가 주기적으로 대시보드를 확인하거나 수동으로 알람을 분석할 필요 없이, 시스템이 이상 징후를 자동으로 포착하고 필요한 정보를 즉시 전달해 줍니다. 이는 사람의 개입을 최소화하여 휴먼 에러의 가능성을 방지하

고, 운영팀이 보다 중요한 근본 원인 분석과 해결에 집중할 수 있도록 지원하여 전반적인 운영 효율성을 크게 개선합니다.

인시던트는 우리에게 문제가 발생했음을 알려줍니다. 이제 평균 해결 시간(MTTR, Mean Time to Repair)을 줄이기 위한 경쟁이 시작됩니다. 다음 섹션에서는 MSAP Observability가 이 경쟁에서 승리하기 위해 어떤 통합된 분석 도구들을 제공하는지 자세히 살펴보겠습니다.

8.2. 인시던트·Runbook 기반 운영 자동화

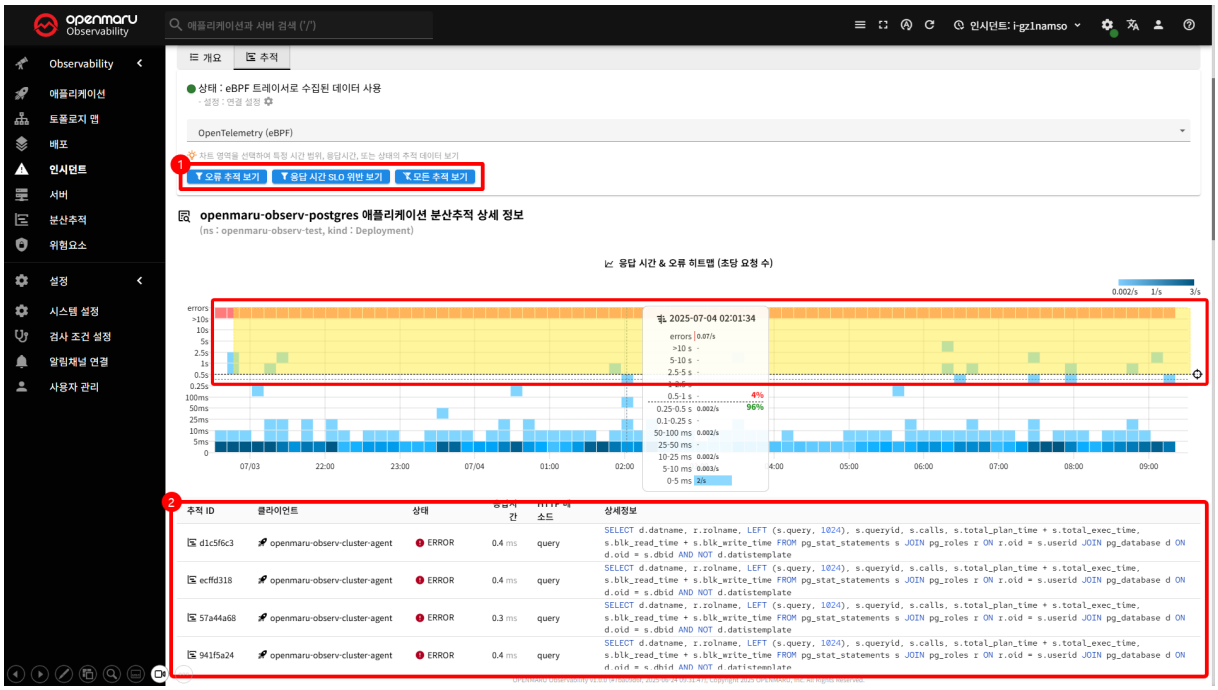
8.2.1. 인시던트 상세 분석 화면과 트레이스·로그 연계

SRE 기반 운영의 핵심 목표는 SLO 위반이라는 '신호'를 받은 후, 서비스를 정상 상태로 복구하는 데 걸리는 평균 해결 시간(MTTR)을 최소화하는 것입니다. 이를 위해서는 근본 원인 분석(RCA)을 가속화하는 것이 무엇보다 중요합니다. MSAP Observability는 인시던트, 트레이스, 로그를 유기적으로 연계하여 SRE의 분석 과정을 "What → Where → Why"의 논리적 흐름에 따라 안내합니다.

먼저 인시던트 상세 화면은 문제의 "What"을 명확히 알려줍니다. 운영자는 다음 정보를 통해 문제 상황을 직관적으로 파악할 수 있습니다.

- 문제 식별: 인시던트가 발생한 구체적인 애플리케이션과 문제의 지속 시간을 즉시 확인할 수 있습니다.
- 영향 범위 파악: 어떤 SLO 항목(가용성 또는 응답 시간)이 위반되었는지, 현재 준수율은 얼마인지, 그리고 이 문제로 인해 얼마나 많은 요청이 영향을 받았는지 파악할 수 있습니다.
- 긴급도 판단: 이번 인시던트로 인해 소진된 오류 예산의 비율을 통해 문제의 심각성과 대응의 긴급도를 판단할 수 있습니다.

문제의 개요를 파악했다면, 다음 단계는 "Where" 문제가 발생했는지 특정하는 것입니다. MSAP Observability의 '인시던트 추적 탭'은 이 과정을 획기적으로 단축시킵니다.



[그림 7] 인시던트 추적탭

운영자는 클릭 한 번으로 해당 인시던트에 직접적인 영향을 받은 모든 요청들의 분산 트레이스(Distributed Trace) 목록으로 즉시 이동할 수 있습니다. 분산 트레이스는 마이크로서비스 환경에서 사용자 요청의 전체 여정을 시각적으로 보여주어, 전체 호출 경로에서 어떤 서비스 구간의 응답 시간이 급증했는지, 즉 병목 지점이 어디인지를 신속하게 식별하는 데 결정적인 역할을 합니다.

마지막으로, 병목 지점을 찾았다면 “Why” 그 문제가 발생했는지에 대한 구체적인 증거를 찾아야 합니다. 분산 트레이스의 각 단계를 나타내는 스팬(Span)에서는 관련된 로그를 직접 조회할 수 있습니다. 이는 OpenTelemetry 산업 표준에 따라 로그 데이터에 `trace_id`를 포함시켜 트레이스와 로그를 자동으로 연결하기에 가능합니다. 운영자는 병목이 발생한 스팬에서 관련 로그를 확인함으로써, 코드 레벨의 오류(예: 예외 스택 트레이스, 잘못된 파라미터)와 같은 구체적인 원인을 찾아낼 수 있습니다.

이처럼 MSAP Observability는 인시던트(What), 트레이스(Where), 로그(Why)를 하나의 화면에서 유기적으로 연결함으로써, 운영자가 여러 도구를 오가며 수동으로 데이터를 연관 분석해야 했던 비효율을 제거하고 MTTR을 크게 단축시킵니다.

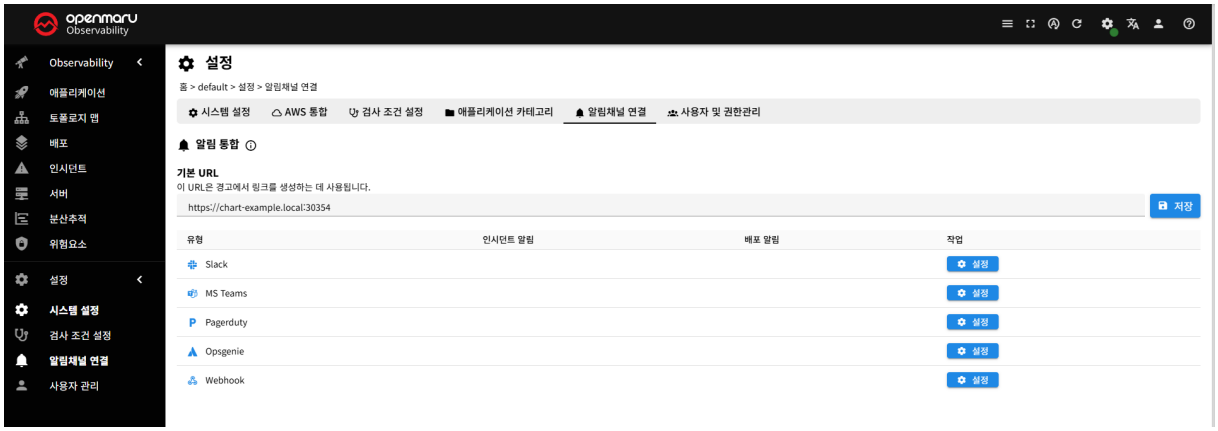
문제의 원인을 파악했다면, 이 정보를 적시에 적절한 담당자에게 효과적으로 전달하는 것이 신속한 해결의 다음 단계입니다. 이제 MSAP Observability가 제공하는 알림 채널 연계 정책에 대해 알아보겠습니다.

8.2.2. 알림 채널(메신저·알림 시스템) 연계 정책

효과적인 인시던트 대응 체계는 기술적인 분석 능력뿐만 아니라, 조직의 기존 워크플로우 및 협업 도구와 원활하게 통합될 때 비로소 완성됩니다. 문제가 발생했을 때, 담당자가 이미 일상적으로 사용하고 있는 도구로 알림을 즉시 전달하는 것은 대응 속도를 높이고 팀 간의 협업을 촉진하는 핵심 요소입니다.

MSAP Observability는 이러한 조직의 요구사항을 충족시키기 위해 다양한 외부 시스템과의 유연한 연동을 지원합니다. 플랫폼이 기본적으로 지원하는 주요 알림 채널은 다음과 같습니다.

- Slack
- MS Teams
- Webhook



[그림 8] 알림채널

특히 주목할 점은 Webhook 지원입니다. 이는 단순히 유연성을 제공하는 것을 넘어, MSAP Observability가 각 조직의 고유한 '운영 신경 시스템(operational nervous system)'에 직접 통합될 수 있도록 하는 전략적 확장성을 의미합니다. Webhook을 통해 기업은 다음과 같은 맞춤형 자동화 대응 문화를 구축할 수 있습니다.

- 내부 시스템 연동: 사내에서 사용하는 SMS 발송 시스템, 이슈 트래커(예: Jira), 또는 사내 메신저와 직접 연동하여 조직의 고유한 장애 대응 프로세스를 완벽하게 자동화할 수 있습니다.

- 유연한 정책 수립: 인시던트의 심각도나 발생한 서비스의 중요도에 따라 각기 다른 채널로 알림을 보내는 다단계 알림 정책을 구현할 수 있습니다. (예: 경고 수준은 Slack으로, 심각 수준은 Pagerduty와 SMS로 동시 전송)

이처럼 MSAP Observability의 알림 채널 연계 정책은 플랫폼을 단순한 알림 도구가 아닌, 각 기업의 운영 문화와 프로세스에 맞춰진 맞춤형 대응 체계를 구축하는 핵심 동력으로 만듭니다.

알림을 통해 문제를 인지하고 원인을 분석했다면, 다음 단계는 반복적인 해결 작업을 자동화하여 운영 부담을 줄이고 실수를 방지하는 것입니다. 이제 Runbook 및 자동화된 작업을 통한 AIOps 운영 패턴에 대해 논하겠습니다.

8.2.3. Runbook·자동 작업과 연계한 AIOps 운영 패턴

전통적인 AIOps(AI for IT Operations)가 AI를 활용해 방대한 데이터에서 이상 징후를 탐지하고 상관관계를 분석하여 인간에게 제시하는 시스템이라면, VibeOps는 여기서 한 걸음 더 나아갑니다. VibeOps는 LLM(거대 언어 모델)을 결합하여 운영 데이터의 맥락을 이해하고, 자연어로 소통하며, 실행 가능한 해결책을 권고하는 차세대 지능형 운영 체계입니다. 이는 운영자를 단순 분석가에서 핵심 의사결정자로 격상시킵니다.

VibeOps는 운영자가 더 이상 복잡한 쿼리나 대시보드 분석에 의존하지 않고, 마치 동료 전문가와 대화하듯 시스템의 상태를 파악하고 문제를 해결할 수 있는 새로운 운영 경험을 제공합니다. 인시던트 발생 시 운영자와 VibeOps 간의 상호작용은 다음과 같은 시나리오로 이루어질 수 있습니다.

- 운영자 질문: **"지금 발생한 장애 원인을 분석해줘."**
- VibeOps 답변:

VibeOps의 진정한 가치는 단순히 원인 분석에 그치지 않고, 문제 해결을 위한 구체적인 권고안을 제시하는 데 있습니다. VibeOps는 분석 결과를 바탕으로 사전 정의된 Runbook(자동화된 복구 스크립트 또는 표준 운영 절차 문서)과 연계하여 운영자가 취해야 할 다음 조치를 제안합니다.

예를 들어, VibeOps는 “외부 API 호출 타임아웃 설정을 검토하고, 관련 Runbook ’RB-102: 외부 서비스 지연 대응’에 따라 임시 조치를 수행하는 것을 권장합니다.”와 같은 구체적인 가이드를 제공할 수 있습니다. 이를 통해 운영자는 신속하고 정확하게 다음 조치를 결정하고 실행할 수

있으며, 반복적인 장애에 대해서는 Runbook을 자동 실행하여 인력의 개입 없이 문제를 해결하는 완전한 자동화 운영으로 나아갈 수 있습니다.

이처럼 MSAP Observability는 단순한 데이터 수집 및 시각화 도구를 넘어, VibeOps를 통해 지능형 분석과 운영 자동화까지 지원합니다. 이러한 강력한 기능은 MSAP COP 및 MSAP.ai와 같은 더 큰 플랫폼 생태계와 통합될 때 그 가치가 극대화됩니다.

8.3. MSAP COP·MSAP.ai와의 통합 운영 시나리오

8.3.1. MSAP COP 내 APM·Observability·Session Clustering·VibeOps 연계

MSAP Observability의 진정한 차별점은 독립적인 솔루션이 아니라, 쿠버네티스 기반의 통합 운영 플랫폼인 MSAP COP(Cloud native Operation Platform)의 핵심 구성요소로 설계되었다는 점입니다. 개별 도구들이 파편화되어 데이터 사일로를 형성하는 기존 환경과 달리, MSAP COP는 Observability, APM, Session Clustering, VibeOps 등 필수적인 운영 컴포넌트들을 단일 플랫폼 내에서 유기적으로 연동하여 일관되고 심층적인 통합 운영 경험을 제공합니다.

다음은 사용자가 실제 문제 해결 과정에서 MSAP COP의 각 컴포넌트를 어떻게 유기적으로 활용하는지를 보여주는 시나리오입니다.

1. [Observability] 문제 인지 및 영향 범위 파악 운영자는 **MSAP Observability**의 토폴로지 맵에서 특정 마이크로서비스로 향하는 연결선을 확인합니다. 연결선은 초당 요청 수(예: 150 rps)는 정상이지만, 평균 응답 시간(□)이 45ms에서 1.2s로 급증한 것을 보여주며 비정상 상태인 주황색으로 표시되어 있습니다. 동시에, 해당 서비스의 응답 시간 SLO 위반으로 인한 인시던트가 자동으로 생성되어 Slack으로 알림이 도착합니다. 이를 통해 운영자는 “어떤 서비스 간의 상호작용”에 문제가 발생했는지 즉각적으로 파악합니다.
2. [APM] 애플리케이션 내부 심층 분석 Observability가 eBPF 기반으로 서비스 간 상호작용(where)을 정확히 지목했지만, Java 애플리케이션 내부의 복잡한 로직까지는 파고들 수 없습니다. “왜” 애플리케이션 내부에서 지연이 발생하는지 진단하기 위해, 운영자는 인시던트 화면에서 클릭 한 번으로 **OPENMARU APM**의 상세 트랜잭션 Call Tree 화면으로 원활하게 전환합니다. APM은 WAS의 내부 스레드 상태, Heap Memory 사용량, Full GC 발생 이벤트부터 특정 SQL 쿼리 지연, 비효율적인 메소드 호출까지 코드 레벨의 병목 지점을 정밀

하게 식별합니다.

3. [Session Clustering] 상태 정보 확인 APM 분석 결과, 특정 SQL 쿼리가 아닌 전반적인 애플리케이션 처리 속도 저하가 관찰될 때, 다음으로 논리적인 가설은 상태 관리(State Management) 문제입니다. 운영자는 문제의 원인이 세션 데이터의 불일치나 지연일 가능성을 확인하기 위해, **MSAP COP** 통합 대시보드에서 **OPENMARU Cluster**(In-Memory Data Grid)의 상태를 조사합니다. 클러스터의 데이터 분포, 네트워크 지연, GC 상태 등을 점검하여 세션 데이터가 정상적으로 외부화되어 안정적으로 관리되고 있는지 확인합니다.
4. [VibeOps] 종합 분석 및 최종 해결 방안 제시 수집된 모든 데이터(Observability의 네트워크 지표, APM의 트랜잭션 데이터, Cluster의 상태 정보)를 **VibeOps**가 종합적으로 분석합니다. VibeOps는 각 데이터 간의 상관관계를 추론하여 최종적인 근본 원인 분석 리포트와 해결 방안을 자연어로 제공합니다. 예를 들어, "특정 SQL의 응답 시간 증가로 인해 APM에서 트랜잭션 지연이 발생했으며, 이 영향으로 MSAP Observability의 응답 시간 SLO를 위반했습니다. 해당 SQL에 대한 인덱스 튜닝을 권장합니다." 와 같이 명확한 결론과 다음 조치를 제시하여 MTTR을 획기적으로 단축시킵니다.

이처럼 MSAP COP는 전통적인 애플리케이션의 개발, 배포, 운영 전반을 통합하여 관리하지만, MSAP.ai는 여기서 한 걸음 더 나아가 AI Native 애플리케이션 모니터링이라는 새로운 시대의 과제까지 해결합니다.

8.3.2. MSAP.ai를 통한 AI Native 애플리케이션 모니터링 확장

LLM(거대 언어 모델) 기반 서비스와 같이 Python, Go 등 다양한 언어(Polyglot)로 구성된 AI Native 애플리케이션 파이프라인의 등장은 기존 Java 중심의 모니터링 방식에 새로운 도전을 제기합니다. 각 언어별로 별도의 에이전트를 설치하고 코드를 수정하는 방식은 복잡한 AI 워크플로우의 관측성을 확보하는 데 비효율적입니다.

이러한 문제에 대한 근본적인 해결책이 바로 eBPF 기술과 이를 기반으로 한 Zero-Instrument Observability입니다. 이는 MSAP.ai와 같이 다양한 기술 스택이 혼재된 환경에서 포괄적인 LLMOps를 가능하게 하는 foundational technology입니다. MSAP Observability는 eBPF를 활용하여 애플리케이션 코드를 전혀 수정하거나 언어별 에이전트를 설치할 필요 없이, OS 커널 레벨에서 AI 서비스들의 네트워크 통신, 시스템 호출 등을 자동으로 계측합니다. 특히 약 1%의

CPU와 250MB의 메모리라는 극히 낮은 오버헤드로 운영 환경에서의 Continuous Profiling을 가능하게 하여, 코드 수정 없이도 AI 파이프라인의 성능 병목을 식별할 수 있습니다.

AI 서비스 운영, 즉 LLMOps는 기존의 성능 지표를 넘어 새로운 유형의 관측성을 요구합니다. MSAP Observability는 이러한 요구사항에 다음과 같이 대응합니다.

- LLM 파이프라인 추적: 사용자의 프롬프트가 입력되어 벡터 DB 검색, 모델 추론, 후처리 등 LLM 파이프라인의 각 단계를 거치는 과정을 분산 트레이싱으로 추적하여 병목 구간을 식별합니다.
- 비즈니스 지표 모니터링: API 호출 당 토큰 사용량, 모델 추론 비용, RAG(검색 증강 생성)의 정확도와 같은 비즈니스 및 품질 지표를 수집하고 모니터링하여 AI 서비스의 ROI와 성능을 종합적으로 관리합니다.

이처럼 MSAP Observability는 AI Native 애플리케이션의 복잡성을 효과적으로 관리하고, 기술적 성능과 비즈니스 가치를 연결하는 핵심적인 도구로 그 역할이 확장됩니다. 이는 관측성이 단순히 운영 단계를 넘어 AI 기반 MSA의 설계와 배포를 포함한 전체 라이프사이클에서 핵심적인 역할을 수행함을 의미합니다.

8.3.3. AI 기반 MSA 설계·배포·운영 전주기에서 Observability의 역할

Observability는 더 이상 운영 단계에서 발생하는 문제를 해결하는 사후 대응 도구가 아닙니다. 이는 DevOps와 MLOps 전체 라이프사이클에 걸쳐 가치를 제공하며, 데이터 기반의 의사결정을 가능하게 하는 전략적 자산으로 재정의되어야 합니다. AI 기반 마이크로서비스 아키텍처(MSA)의 설계, 배포, 운영 전주기에서 Observability 데이터가 어떻게 활용되는지는 다음과 같습니다.

단계	Observability의 역할	기대 효과
설계(Design)	기존 시스템에서 수집된 서비스 의존성, 트래픽 패턴, 리소스 사용량 데이터를 분석하여, MSAP.ai가 더 효율적이고 안정적인 마이크로서비스 아키텍처를 설계하도록 지원합니다.	데이터 기반의 아키텍처 설계를 통해 잠재적인 병목 지점을 사전에 최소화하고, 서비스 간 결합도를 낮추어 변경 용이성을 높입니다.

배포(Deploy)	배포 전후의 SLO, 오류 예산, 응답 시간 등 핵심 지표를 자동으로 비교 분석하여 배포의 성공 여부를 판단하는 품질 게이트(Quality Gate) 역할을 수행합니다. (**배포 후 성능 분석** 기능 참조)	카나리(Canary), 블루/그린(Blue/Green)과 같은 점진적 배포 전략의 안정성을 크게 향상시키고, 실패한 배포의 영향 범위(blast radius)를 극적으로 줄입니다.
운영(Operate)	실시간 성능 모니터링, SLO 위반에 기반한 자동 인시던트 대응, 그리고 VibeOps를 통한 지능형 근본 원인 분석을 제공하여 안정적이고 예측 가능한 서비스 운영을 보장합니다.	수동적인 장애 대응(reactive firefighting)에서 벗어나 예측 기반의 선제적 유지보수(proactive maintenance)로 전환하여 운영 부담(toil)을 줄이고 MTTR을 획기적으로 단축합니다.

운영 복잡성이 새로운 상수가 된 시대에, IT 운영은 더 이상 개별 도구들의 파편화된 집합이 아닌, 지능적이고 통합된 플랫폼으로 진화해야 합니다. MSAP 생태계의 심장부에 위치한 MSAP Observability는 단순한 모니터링을 넘어, 분산 시스템의 혼란스러운 노이즈를 일관된 운영 인텔리전스로 변환합니다.

이는 SRE 원칙을 이론에서 자동화된 현실로 전환시키는 역할을 합니다. 따라서 MSAP Observability는 단순히 신뢰성을 위한 도구가 아니라, AI 네이티브 시대를 향한 비즈니스 민첩성과 혁신을 위한 전략적 엔진이 됩니다.

제 9장. References & Links

- MSAP Observability – https://www.openmaru.io/product/openmaru-observability/
- MSAP CogentAI – https://www.openmaru.io/product/openmaru-cogentai/
- MSAP COP – https://www.openmaru.io/product/openmaru-cop/
- OPENMARU 공식 홈페이지 (제품 소개) – https://www.openmaru.io/
- OPENMARU 기술 문서 (Docs) – https://www.openmaru.io/docs/

- OPENMARU 기술 블로그 – https://www.openmaru.io/openmaru-blog-home/
- OPENMARU 유튜브 채널 – https://www.youtube.com/@msaptv
- Google SRE Book – https://sre.google/books/
- OpenTelemetry 공식 문서 – https://opentelemetry.io/
- Grafana Continuous Profiling – https://grafana.com/docs/grafana/latest/explore/simplified-exploration/profiles/concepts/continuous-profiling/
- eBPF 공식 사이트 – https://ebpf.io/what-is-ebpf/
- Skydive with eBPF – http://skydive.network/blog/skydive-with-ebpf.html
- NetObserv – https://github.com/netobserv/flowlogs-pipeline
- Cilium – https://cilium.io/
- OpenTelemetry 공식 홈페이지 – https://opentelemetry.io/
- OpenTelemetry GitHub– https://github.com/open-telemetry
- OTel Languages – https://opentelemetry.io/docs/languages/
- Google Dapper 논문– https://research.google/pubs/dapper-a-large-scale-distributed-systems-tracing-infrastructure/
- OTel Collector Scaling– https://opentelemetry.io/docs/collector/scaling/
- OTel Kubernetes Operator – https://opentelemetry.io/docs/kubernetes/operator/
- OTel Demo – https://opentelemetry.io/docs/demo/
- OTel Collector ClickHouse Exporter – https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/exporter/clickhouseexporter

- ClickHouse Grafana Integration – https://clickhouse.com/docs/en/integrations/grafana/query-builder#logs

  https://clickhouse.com/docs/en/integrations/grafana/query-builder#traces

- ClickHouse MergeTree – https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/mergetree

- Liquibase – https://www.liquibase.com/community

- Elastic OpenTelemetry Profiling Agent – https://opentelemetry.io/blog/2024/elastic-contributes-continuous-profiling-agent/

  https://github.com/elastic/otel-profiling-agent

- OTel Network Flow Discussions – https://github.com/open-telemetry/community/issues/733

  https://youtu.be/F1VTRqEC8Ng?si=pP2WiDtTraXffQd-

- qryn – https://github.com/metrico/qryn

- SigNoz – https://signoz.io/

  https://github.com/SigNoz/signoz

- Uptrace – https://uptrace.dev/

  https://github.com/uptrace/uptrace/tree/master

- XObserve

  https://github.com/xobserve/xo/tree/main

- Aspire (.NET)

  https://github.com/dotnet/aspire/tree/main/src/Aspire.Dashboard

- InfoQ Cloud Native Architecture|Monitoring Challenges– https://www.infoq.com/articles/cloud-native-architecture-adoption-part2/

- IEEE Top Programming Languages 2024 – https://spectrum.ieee.org/top-programming-languages-2024

Contact Us

02-6953-5427

hello@msap.ai

www.msap.ai

MSAP.ai Blog

최신 기술 트렌드와
유용한 팁들을 가장 먼저
만나보세요.

MSAP.ai eBook

이제 나도 MSA 전문가
개념부터 실무까지

YouTube

클라우드 기반 기술과
인프라 전략을 다루는
전문 채널

MSAP.ai

엠에스에이피닷에이아이 | MSAP.ai

전화 : (02) 6953 - 5427
팩스 : (02) 469 - 7247
메일 : hello@msap.ai