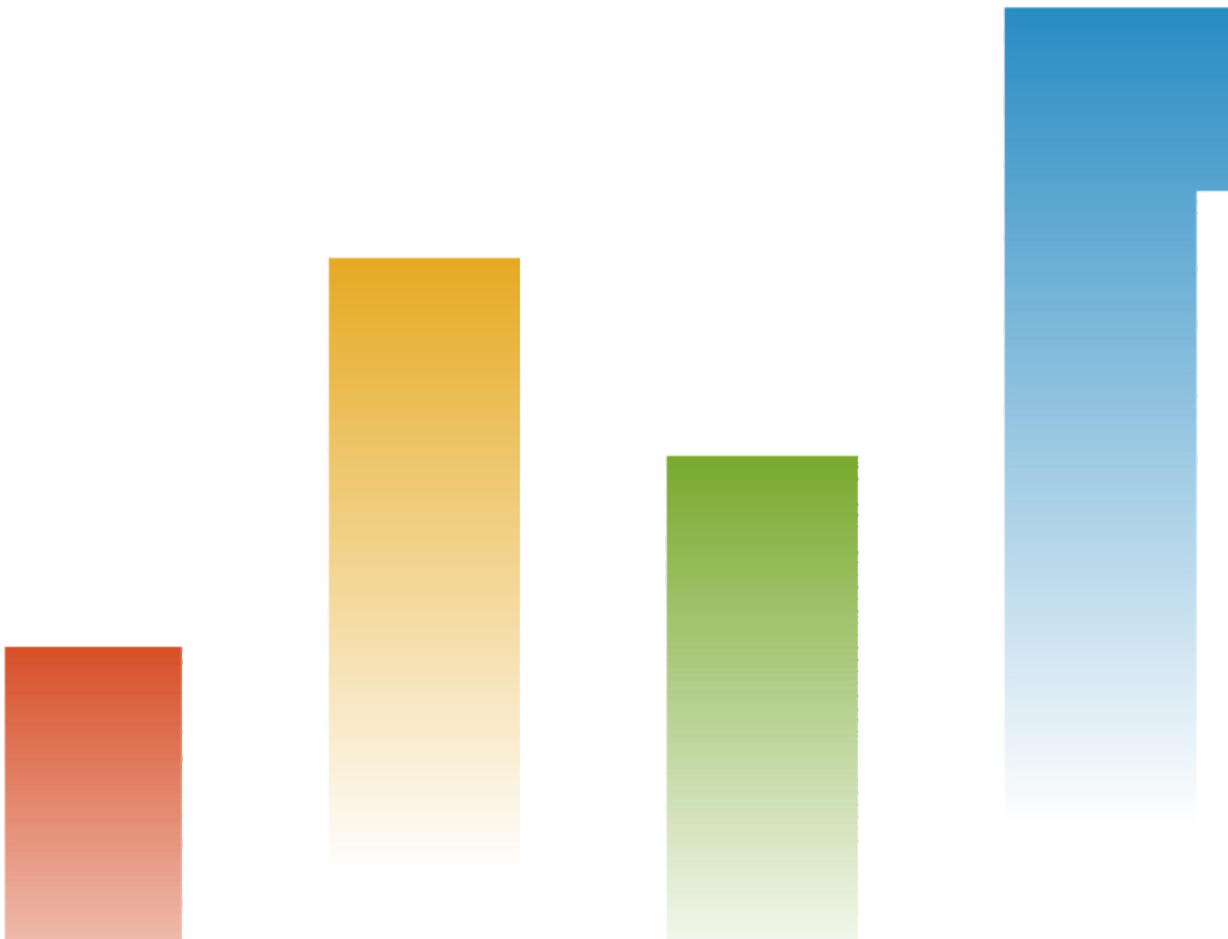
WhaTap

성능 장애 유형

release date. 2026. 09. 10.



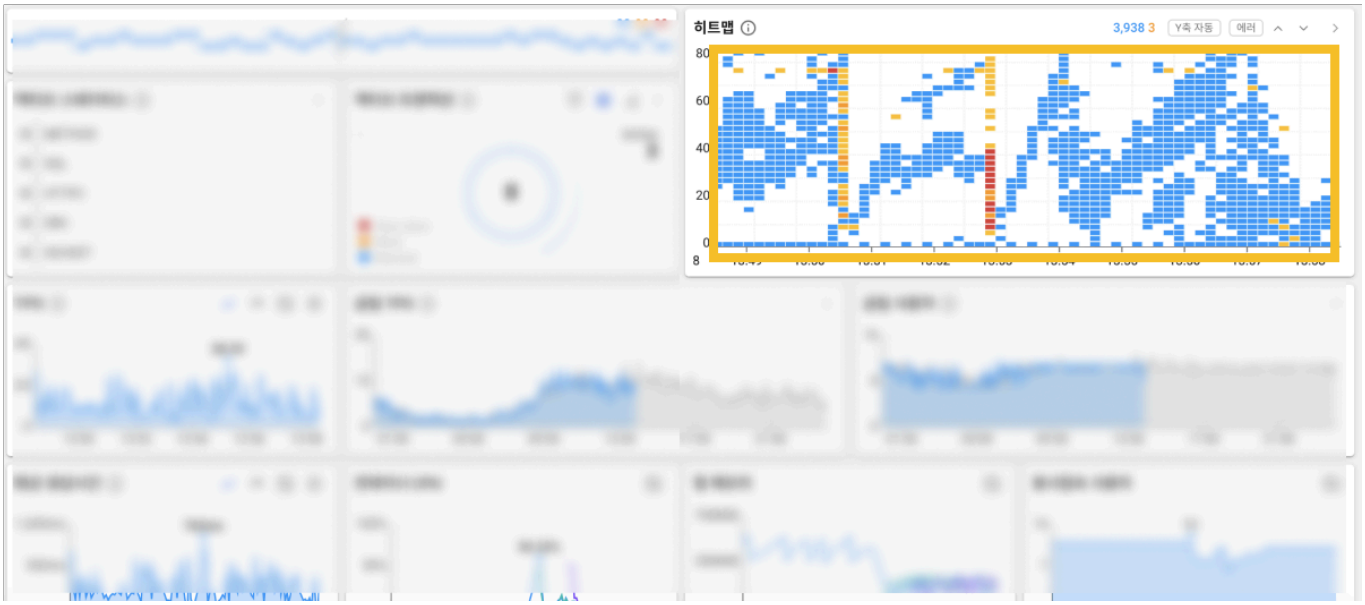
DB 연결 지연과 커넥션 풀

이 문서는 대표적인 애플리케이션 성능 장애 유형 중 하나인 **DB 연결 지연**(DBC 지연)과 **Connection Pool**의 상관관계를 모니터링 관점에서 정리합니다.

먼저 대시보드의 **히트맵** 위젯을 통해 1차적으로 장애 상황을 인지했다고 가정한 뒤, **분석** 메뉴 하위의 **히트맵** 메뉴로 이동해 시간을 특정하는 방식으로 장애 상황 전후를 살펴봅니다. 이를 토대로 **트레이스 정보** 창과 **메트릭스 차트** 등을 통해 장애 원인을 추론하는 과정을 다룹니다.

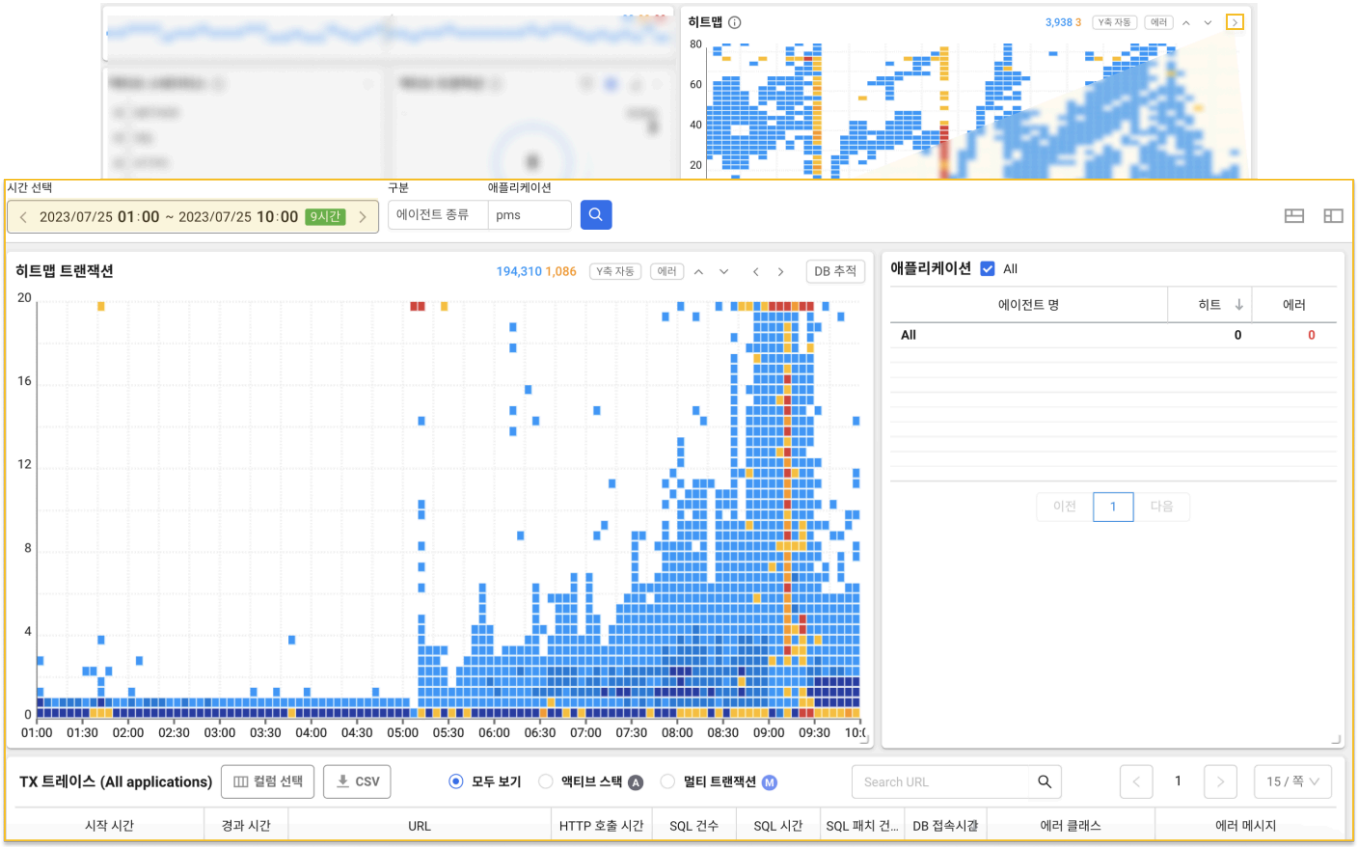
> **미리 알아보기**
>
> **애플리케이션 대시보드 → 히트맵 트랜잭션 → 트레이스 정보 → 메트릭스 차트**
>
> - 문제 현상을 인지했다면 과거 기록을 조회해 장애 현상 시점 전후를 조회합니다. **히트맵 트랜잭션** 차트에서 트랜잭션 과부하 패턴 확인 후 **TX 트레이스** 목록을 통해 트랜잭션 지연 원인을 DB 연결 문제로 예상할 수 있습니다.
> - 구간별 조회 후 트레이스 분석을 통해 상세 수행 이력을 스텝별로 살펴봅니다. **경과** 시간을 가장 많이 소요하고 있는 것이 DBC 스텝임을 확인할 수 있습니다.
> - **메트릭스 차트**를 통해 다양한 지표를 함께 살펴봅니다. **DB Pool 개수** 차트와 **DB Active 개수** 차트를 비교해 DB Connection Pool의 부족이 원인임을 특정할 수 있습니다.
> - 사용자 운영 환경에 따라 Connection Pool의 크기를 점차적으로 조절하거나 누수를 막기 위해 최적화 설정을 하는 방법을 활용할 수 있습니다.

히트맵 이상 패턴 감지



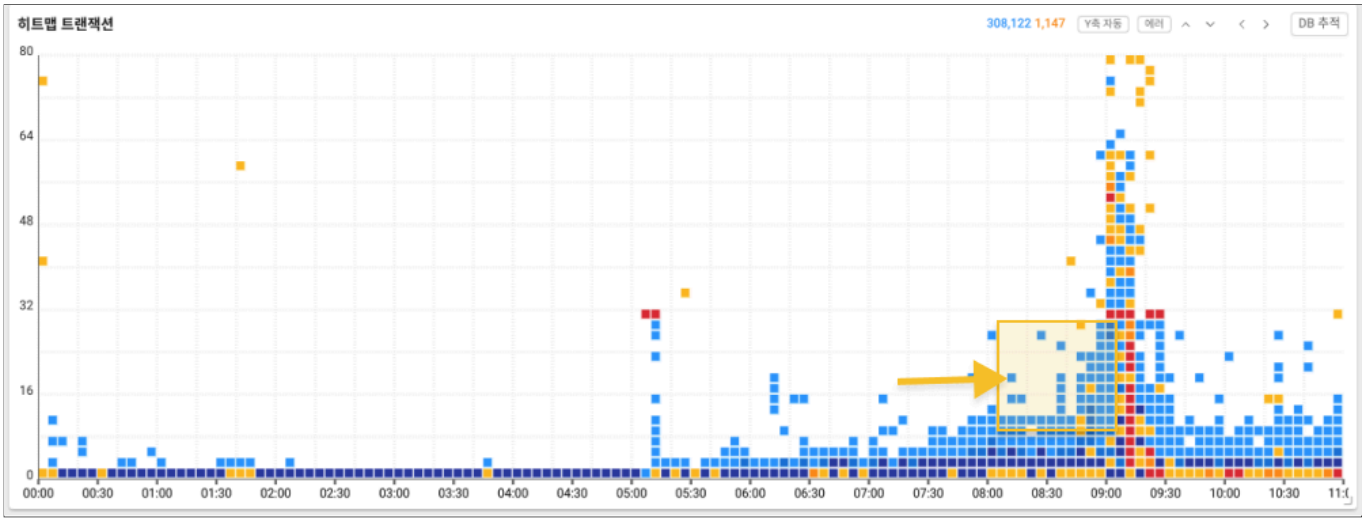
장애 현상을 인지하고 일차적인 분석을 통해 장애 원인을 추적하는 과정은 신속함이 핵심입니다. 와탭은 직관적인 뷰를 제공할 뿐만 아니라 근본 원인 추적을 위한 추가 정보를 확보할 수 있도록 과거 기록 조회 기능을 제공합니다. **애플리케이션 대시보드**의 **히트맵** 위젯에서 예시처럼 급격한 이상 징후를 감지했다면 다음 순서는 무엇일까요?

히트맵 위젯에서 문제 구간을 드래그해 트랜잭션 트레이스 분석을 빠르게 진행할 수도 있고, 또는 **분석** 메뉴 하위의 **히트맵**으로 이동해 시간 범위를 보다 넓게 특정하는 방식으로 전후 상황을 함께 살펴볼 수도 있습니다.

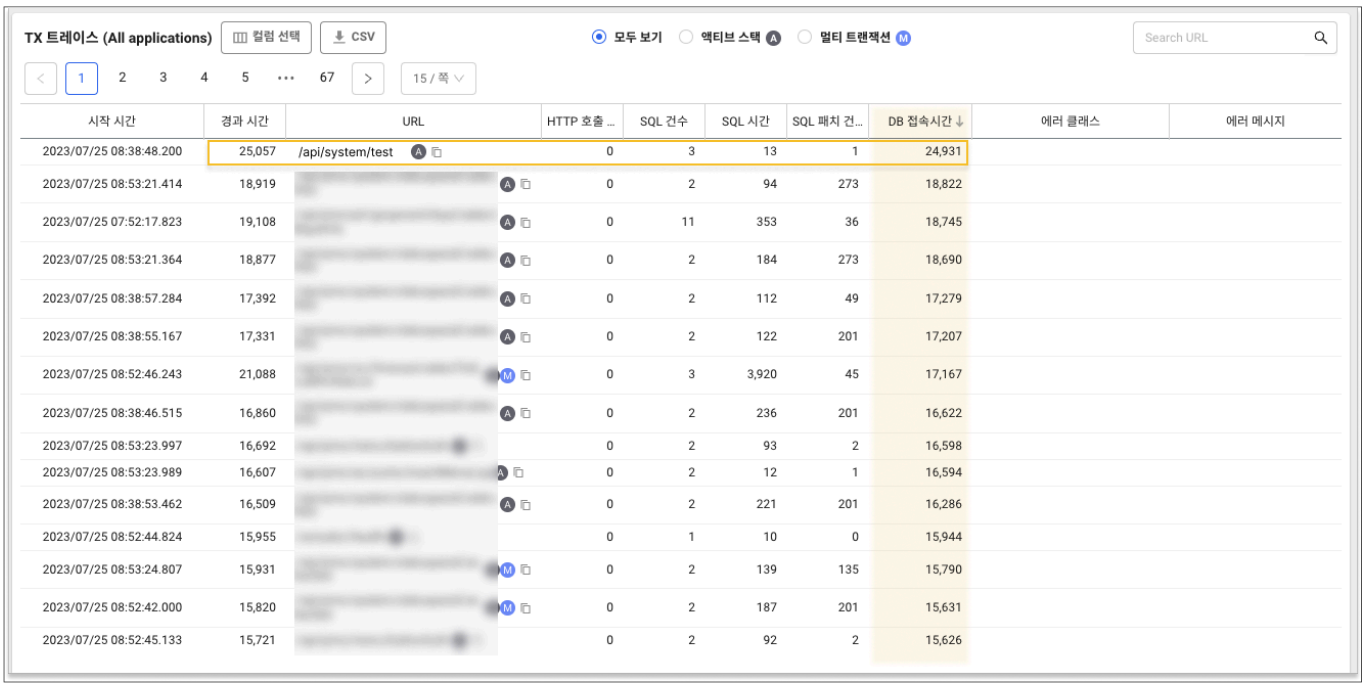


후자의 동선으로 진행해 보겠습니다. 대시보드 **히트맵** 위젯은 최근 10분간 종료된 트랜잭션 응답 시간 분포도입니다. 즉 시간 범위를 특정해 최근 10분 이상의 과거 내역을 확인하려면 **분석** 메뉴 하위의 **히트맵** 메뉴로 이동해야 합니다. 위젯 우측 상단의 › 화살표 아이콘을 클릭해 예시와 같이 이동할 수 있습니다.

히트맵 트랜잭션 차트 상단의 시간 선택자를 통해 원하는 시간 범위를 지정해 과거 기록을 조회할 수 있습니다. 이상 패턴을 확정하기 위해 문제 구간 전후로 넓게 살펴보니 응답이 지연되고 있는 트랜잭션이 밀집한 과부하 패턴을 확인할 수 있었습니다.



이와 같은 갑작스러운 응답 시간 증가와 트랜잭션 과부하의 원인을 알아보기 위해 **히트맵 트랜잭션** 차트에서 문제 영역을 구간별로 드래그해 트랜잭션 트레이스 목록(**TX 트레이스** 목록)을 조회해 보겠습니다.



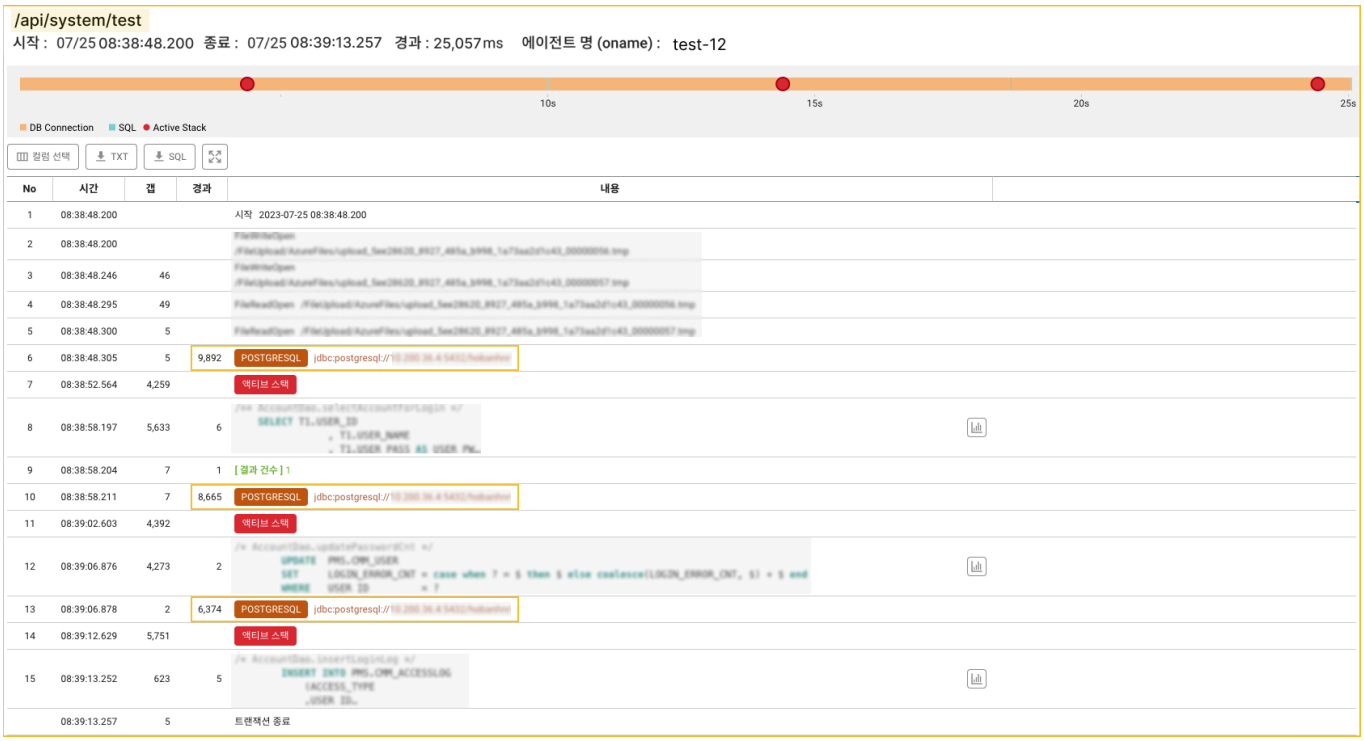
시작 시간	경과 시간	URL	HTTP 호출 ...	SQL 건수	SQL 시간	SQL 패치 건...	DB 접속시간 ↓	에러 클래스	에러 메시지
2023/07/25 08:38:48.200	25,057	/api/system/test	0	3	13	1	24,931		
2023/07/25 08:53:21.414	18,919		0	2	94	273	18,822		
2023/07/25 07:52:17.823	19,108		0	11	353	36	18,745		
2023/07/25 08:53:21.364	18,877		0	2	184	273	18,690		
2023/07/25 08:38:57.284	17,392		0	2	112	49	17,279		
2023/07/25 08:38:55.167	17,331		0	2	122	201	17,207		
2023/07/25 08:52:46.243	21,088		0	3	3,920	45	17,167		
2023/07/25 08:38:46.515	16,860		0	2	236	201	16,622		
2023/07/25 08:53:23.997	16,692		0	2	93	2	16,598		
2023/07/25 08:53:23.989	16,607		0	2	12	1	16,594		
2023/07/25 08:38:53.462	16,509		0	2	221	201	16,286		
2023/07/25 08:52:44.824	15,955		0	1	10	0	15,944		
2023/07/25 08:53:24.807	15,931		0	2	139	135	15,790		
2023/07/25 08:52:42.000	15,820		0	2	187	201	15,631		
2023/07/25 08:52:45.133	15,721		0	2	92	2	15,626		

TX 트레이스 목록을 통해 다수의 트랜잭션에서 **경과 시간** 대비 **DB 접속 시간**이 차지하는 비율이 큰 것을 확인할 수 있습니다. 예시로 `/api/system/test` URL의 경우 총 **경과 시간** 25,057ms 중 DB 연결에만 24,931ms가 소요되었습니다. 즉 일차적으로 DB 연결에 문제가 있다는 것을 추측할 수 있습니다. 다음으로 해당 트랜잭션을 선택해 **트레이스 정보** 창에서 상세 정보를 살펴보겠습니다.

- 히트맵의 가로축은 트랜잭션 종료시간을, 세로축은 응답 시간을 의미합니다. 히트맵에서 ^ 화살표 아이콘을 클릭해 세로축 응답 시간 지연 구간을 80초까지 조회할 수 있습니다.
- 과부하 패턴 외에 히트맵 패턴 분석에 대한 자세한 내용은 다음 문서를 참조하세요.

트랜잭션 트레이스 분석

트레이스 정보 창에서 트랜잭션 상세 수행 이력을 스텝별로 확인할 수 있습니다. 이와 같은 트랜잭션 트레이싱을 통해 트랜잭션의 성능 저하 또는 오류의 원인을 추적할 수 있습니다. **트레이스 정보** 창에서 가장 먼저 살펴봐야 할 정보는 스텝별 **경과** 시간입니다. 소요 시간을 비교하면 성능 저하나 오류의 원인을 빠르게 특정할 수 있습니다.



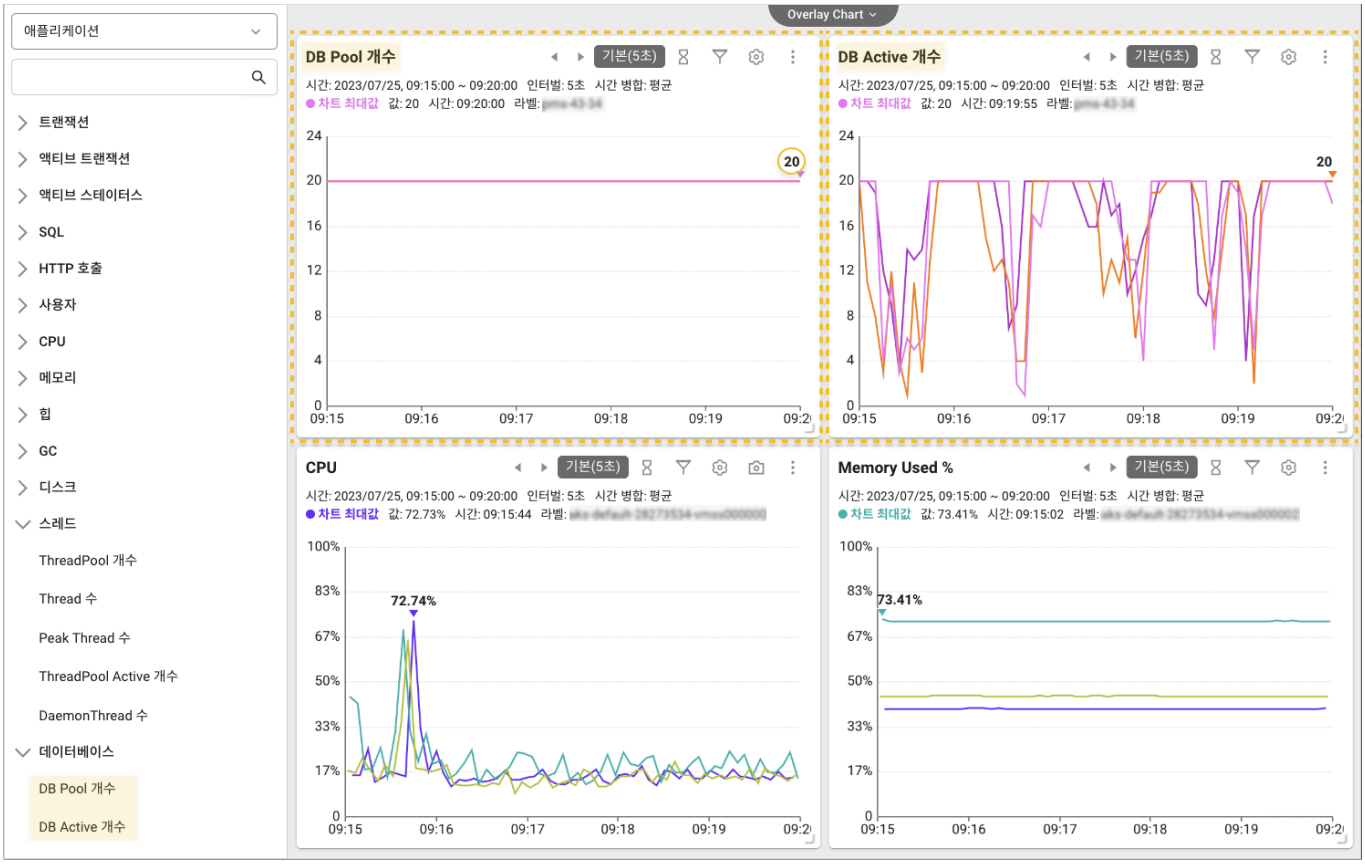
해당 URL의 상세 트레이스를 살펴보겠습니다. `/api/system/test` 수행 중 PostgreSQL DB에 접속해 처리하는 쿼리 수행 시간은 1ms에서 6ms 사이로 비교적 큰 문제가 없습니다. 하지만 해당 데이터베이스 접속 대기 시간이 그에 비해 확연히 오래 걸리고 있으며, **경과** 시간을 가장 많이 소요하고 있는 것이 **DB Connection** 스텝임을 확인할 수 있습니다.

다시 말해 이 경우 앞서 예상한 대로 DB 연결 지연으로 인해 트랜잭션 지연이 발생했음을 특정할 수 있습니다. 그렇다면 이런 DB 연결 지연을 일으키는 원인은 무엇일까요?

DB Connection Pool 확인

DB 연결 지연을 일으키는 원인 중 하나는 **Connection Pool의 부족**입니다. Connection Pool의 크기는 동시에 처리 가능한 연결 개수를 의미합니다. 데이터베이스는 요청에 의한 Connection을 생성하고 종료하는 과정에 많은 시간을 소모합니다. Connection Pool은 미리 일정 수의 Connection을 생성해 두고 다른 요청이 있을 때 현재 Connection Pool을 다시 사용할 수 있도록 합니다. 이를 통해 매번 새로운 Connection을 만들 필요가 없어지기 때문에 응답 시간을 줄일 수 있습니다. 문제는 미리 생성해 둔 Connection 개수가 애플리케이션이 필요로 하는 Connection 개수에 비해 부족한 경우로 이때 DB 연결 지연이 발생하게 됩니다.

DB Connection Pool의 상황을 확인하기 위해 **분석** 메뉴의 **메트릭스 차트**로 이동합니다. 먼저 **메트릭스 차트** 좌측 목록에서 **데이터베이스**를 선택해 **DB Pool 개수**와 **DB Active 개수** 차트를 확인하겠습니다.



각 차트에서 확인할 수 있듯이 **DB Pool**의 최대 연결 개수는 20개로 설정되어 있습니다. **DB Active 개수** 차트를 살펴보면 20개 Pool에서 지속적인 점유가 이루어지고 있는 것을 확인할 수 있습니다. 미리 설정된 Pool을 초과하는 요청이 들어오면 DB에 남는 Pool이 생길 때까지 대기하게 되고 이에 따라 지연이 발생하게 됩니다. CPU 및 메모리 사용량 등 자원 현황 조회 시 별다른

특이사항을 찾을 수 없었습니다. 즉 해당 프로젝트의 경우 DB Connection Pool의 부족이 DBC 지연을 일으킨 결정적인 원인이라고 볼 수 있습니다.

그렇다면 이처럼 DB Connection Pool이 부족한 경우 어떻게 해야 할까요? 물론 해결 방법은 사용자 환경마다 다를 수 있습니다. 이 문서에서는 **Connection Pool 크기 조절**과 **Connection 재활용 및 최적화**를 통한 선 조치를 간략히 살펴봅니다.

Connection Pool 크기 조절

먼저 Pool의 최대 연결 개수를 조절하는 방법을 활용할 수 있습니다. Pool을 점차 늘려 DB Active 개수가 Pool 최대치 내에서 유지되는 것을 확인해 적절한 Pool 개수를 설정하는 것입니다. 하지만 Pool의 개수를 지나치게 크게 지정한 경우 자원 소비량의 과도한 증가로 오히려 애플리케이션 성능에 악영향을 줄 수 있습니다. 충분한 메모리와 처리 능력을 갖추고 있는지 확인 후 각자의 운영 환경에 알맞게 조절해야 합니다.

Connection 재활용 및 최적화

두 번째로 Connection 재활용 및 최적화를 통한 방법을 활용할 수 있습니다. 예를 들어 애플리케이션이 사용한 Connection을 반환하지 않아 Connection Leak이 발생한 경우라면 아무리 Connection Pool의 개수를 늘린다고 해도 해당 현상을 해결할 수 없습니다. 이 경우 DB Active 개수는 우상향 그래프를 그리게 됩니다. Connection Leak을 막기 위해서는 사용한 Connection을 반드시 반환하도록 관리해야 합니다. Connection 사용 후 명시적으로 반환하도록 설정하거나 재사용하는 방식을 적용해 볼 수 있습니다.

인스턴스 성능 관리 활용하기

이 문서는 **인스턴스 성능 관리** 메뉴에서 자주 쓰는 두 가지 핵심 기능 — **로딩된 클래스 분석**과 **스레드 목록·덤프** — 의 활용법을 정리합니다. 특히 CPU 사용량이 급증했을 때 문제 스레드를 빠르게 식별하는 흐름을 다룹니다.

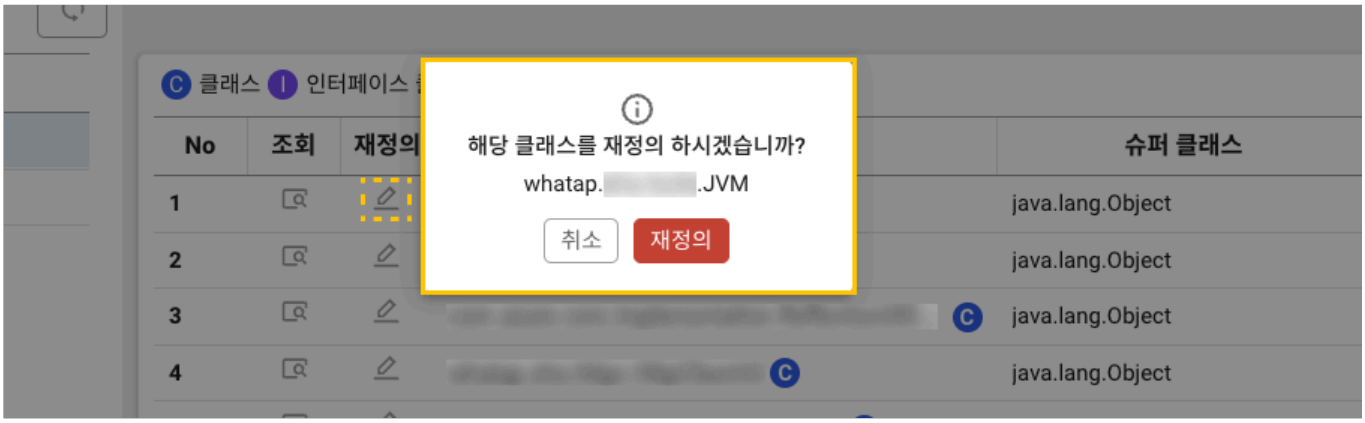
로딩된 클래스

홈 화면 > 프로젝트 선택 > **인스턴스 성능 관리** > **로딩된 클래스**

로딩된 클래스 메뉴는 애플리케이션에 로딩된 클래스의 구조와 메소드 시그니처 등 상세 정보를 제공하고 또한 중단 없는 클래스 동작 변경이 가능하도록 **재정의** 기능을 제공합니다.

재정의

재정의는 `Instrumentation` 의 `redefineClasses` 메소드를 호출하여 애플리케이션을 중단시키지 않고도 런타임 중에 이미 로딩된 클래스의 동작을 변경할 수 있게 합니다. 이는 운영 중인 서비스에 대한 수정이나 변경이 필요한 경우 유용합니다. 예를 들어 새로운 서비스 패턴 지정으로 추가적인 트랜잭션 시작점을 설정하고 응답 시간을 측정하고자 할 때, 관련된 와탭 설정 변경 후 해당 클래스를 **로딩된 클래스** 목록에서 찾아 ✎ 아이콘을 클릭해 **재정의**할 수 있습니다.



리소스

리소스 컬럼을 통해 해당 클래스가 물리적으로 어느 파일에서 로딩되었는지 조회할 수 있습니다. 자바 애플리케이션의 다계층 구조 특성상 이와 같은 **리소스** 정보 확인이 필요합니다. 복잡한 애플리케이션 환경에서는 클래스가 정확히 어느 jar 파일에서

로딩되었는지 파악하는 것이 중요합니다.

특정 클래스가 의도하지 않은 jar 파일에서 로딩된 경우 예상치 못한 동작이나 버그가 발생할 수 있습니다. 특히 클래스패스가 복잡하거나 다중 모듈 환경에서 정확히 어느 경로에서 로딩되었는지 확인하여 디버깅을 용이하게 할 수 있습니다.

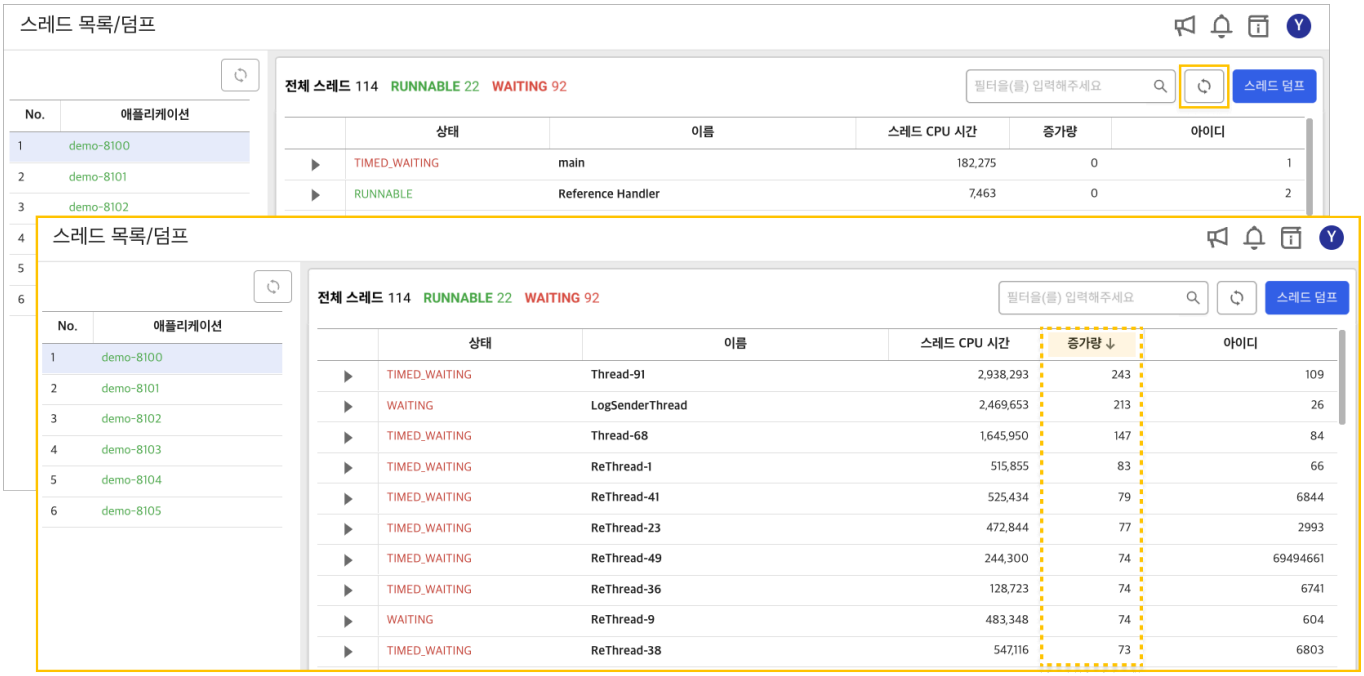
스레드 목록/덤프

홈 화면 > 프로젝트 선택 > **인스턴스 성능 관리** > **스레드 목록/덤프**

스레드 목록/덤프 메뉴에서는 현재 실행 중인 스레드 목록과 해당 스레드의 개별 스냅샷, 전체 스레드 덤프 정보를 조회할 수 있습니다. **히트맵**이 트랜잭션 응답 시간 위주 최적화의 핵심이라면, **스레드 목록/덤프**은 CPU 최적화의 핵심 요소입니다. 애플리케이션 성능 최적화·문제 진단·안정성 유지에 중요한 역할을 합니다.

어떤 스레드가 CPU를 많이 쓰고 있는지 파악하는 것은 병목 원인 식별에 중요합니다. 특히 특정 시간대에 CPU 사용이 급증한 스레드를 찾으려면 시계열 데이터를 조회할 수 있어야 합니다.

스레드 목록/덤프 메뉴의 **스레드 CPU 시간**은 해당 스레드가 CPU를 점유한 누적 시간을 의미합니다. **새로 고침** 버튼을 클릭하면 현재 시점과 직전 시점의 CPU 시간 차이인 **증가량**이 표시되어, 조회 시점에 CPU를 가장 많이 쓰는 스레드를 특정할 수 있습니다. **증가량** 컬럼을 클릭해 내림차순으로 정렬하면 증가량이 가장 큰 스레드를 확인할 수 있습니다.



이렇게 스레드 특정 후 해당 스레드 스냅샷을 확인해 자주 호출되거나 오래 실행되는 메소드 등을 찾아낼 수 있습니다. 이를 통해

이상 징후를 조기에 발견하고 대응하여 시스템을 안정적으로 관리할 수 있습니다.

CPU 사용량이 높은 스레드 확인

애플리케이션 CPU 사용량이 급증한 경우는 여러 가지 원인이 있을 수 있습니다. 이를 힙 메모리가 가득 찬 경우, 실제 요청이 많은 경우, 그리고 로직 문제인 경우 세 가지 유형으로 분류할 수 있습니다.

이때 가장 먼저 확인하는 것은 식별이 용이한 힙 메모리입니다. 힙 메모리가 가득 찬 경우 JVM이 GC를 빈번히 실행해 CPU 사용량이 급증할 수 있습니다. 이 경우 **애플리케이션 대시보드**의 **힙 메모리** 그래프를 통해 쉽게 확인할 수 있습니다. 두 번째는 실제 요청이 많은 것으로 애플리케이션이 많은 수의 클라이언트 요청을 처리해 CPU 사용량이 높아진 경우입니다. 각 요청은 자바 스레드를 생성해 처리되기에 CPU가 많은 작업을 수행하게 됩니다. 이 경우 **애플리케이션 대시보드**에서 TPS 관련 지표를 통해 확인할 수 있습니다.

세 번째로 애플리케이션 코드에 비효율적인 알고리즘이나 무한 루프 등이 포함되어 있을 경우 CPU 사용률이 비정상적으로 높아질 수 있습니다. 이 경우 **스레드 목록/덤프** 메뉴를 통해 CPU 사용량이 높은 스레드를 특정 후 해당 스레드 덤프를 확인해 문제가 되는 코드를 확인할 수 있습니다.

> ⓘ **스레드 CPU 시간** 확인 → **새로고침** → **증가량** 기준으로 목록 정렬 → **CPU 사용량 높은 스레드 식별**

설치 + 프리셋으로 바로 시작하기

인프라 모니터링 도입의 가장 큰 장벽은 **에이전트 설치보다도 알람·임계치 설정에 드는 공수**인 경우가 많습니다. Zabbix 같은 도구는 CPU·Memory·Filesystem 등 지표별 템플릿마다 알람을 일일이 설정해야 합니다. WhaTap Server Monitoring은 **설치만으로 대부분의 감시 지표와 알람이 프리셋**되어 바로 작동하도록 설계되어 있습니다.

1~2분 설치, 그 다음에 자동으로 일어나는 일

1. 감시 대상 서버에 에이전트 다운로드 후 실행 — **평균 1~2분**
2. 에이전트가 프로젝트에 **Active** 상태로 연결
3. **주요 지표 자동 수집 시작** (CPU·메모리·디스크·네트워크·프로세스·파일시스템·TCP 등)
4. **프리셋 알람 자동 활성** — 별도 임계치 설정 없이 이상 시 알림 발송
5. 이후 업데이트는 업데이트 프로그램 실행만으로 끝납니다

프리셋이 덮는 범위

공통 감시 지표

- CPU 사용률·Load Average
- 메모리·Swap 사용률
- 디스크 I/O·Filesystem 사용률
- 네트워크 송수신량
- 프로세스 수·좀비 프로세스
- TCP 연결 상태

클라우드 운영에 특화된 지표

수만 대 규모의 **온프레미스 + 클라우드 서버 운영 노하우**를 바탕으로, 클라우드 환경에서 특히 중요한 지표도 프리셋 포함:

- **CPU Stealtime** — 하이퍼바이저가 CPU를 뺏긴 시간 (클라우드 성능 문제의 흔한 원인)
- **Swap 사용률** — 클라우드 인스턴스 메모리 부족 시 사전 경보
- 기타 클라우드 관리 작업에서 빈번한 지표들

수동 설정 없이 이런 지표까지 자동으로 감시되는 점이 다른 도구와의 큰 차이입니다.

프리셋으로 시작해서 커스터마이즈까지

프리셋은 "시작점"이지 종착지가 아닙니다. 운영하면서 다음 순서로 확장하세요.

1단계 — 그대로 며칠 운영

프리셋 알람을 그대로 두고 2~3일 관찰. 팀 상황에서 **실제로 울리는 알람과 노이즈**를 파악합니다.

2단계 — 팀 맥락 반영

- 울리지 않는데 중요한 지표 → 커스텀 이벤트 규칙 추가 → 첫 경고 알림 붙이기
- 자주 울리지만 대응 불필요 → 임계치 조정·쓰로틀링

3단계 — 팀 단위 운영

- 알림 수신자 역할별 분리 → 팀별 알림 라우팅 설정하기
- 팀 대시보드 공유 → 팀과 대시보드 같이 보기

서버 부하에 대한 고려

WhaTap Server Monitoring은 **감시 대상 서버에 미치는 부하를 CPU 사용률 2% 미만**을 목표로 설계되어 있습니다. 부하가 높아지는 상황에 대비한 완충 장치도 내장되어 있습니다.

- 서버 부하 증가 시 **프로세스 정보 수집 주기**를 60초(기본 20초)로 자동 완화
- 에이전트가 일정 이상 메모리/핸들(Windows) 사용 시 자동 재시작

덕분에 운영 서버에 안전하게 상주할 수 있습니다.

활용 시나리오

신규 서버 온보딩 (5분 내)

1. 서버 에이전트 설치 → 1~2분
2. WhaTap 콘솔에서 에이전트 Active 확인 → 30초
3. 프리셋 대시보드·알람 작동 확인 → 1분
4. 팀 Slack 채널에 에이전트 등록 공지

수십 대 일괄 도입

- 설치 스크립트를 구성관리 도구(Ansible, Chef, Puppet)로 배포
- 모든 서버에 프리셋 알람 동일 적용 → **팀마다 지표 정의가 달라지는 표준화 문제** 제거

장애 대응 조기 도입

- 설치만으로 기본 알림이 바로 작동 → "이상 징후 놓침" 리스크 최소화
- 운영 중 세부 임계치는 점진 조정

다음 단계

- **서버 설치 가이드** → Server Monitoring 소개
- **리소스 보드 해석** → 리소스 보드 살펴보기
- **알림 커스터마이즈** → 첫 경고 알림 붙이기
- **팀 단위 운영으로 확장** → 팀과 대시보드 같이 보기

리소스 보드 살펴보기

이 문서는 와탭 서버 모니터링의 **리소스 보드** 메뉴에서 차트형 위젯을 해석하고 서버 문제를 파악하는 방법을 정리합니다. 기능 개요는 리소스 보드 문서를 참조하세요.

서버 모니터링의 핵심은 프로세스입니다. 프로세스가 정상 동작 범위 내에 중단 없이 운영될 수 있도록 모니터링을 통해 서버 현황을 빠르게 파악하고 각종 장애 상황에 대응할 수 있어야 합니다. 자원 사용량을 조회해 추론하는 것이 그 시작입니다. 와탭의 **리소스 보드**는 데이터 집약적인 화면에서 자원 사용량을 한눈에 확인할 수 있도록 CPU, Memory, Disk, Network 관련 연관 지표들로 구성된 시계열 차트 위젯을 제공합니다.

- 서버 상태 요약: **Server**, **OS**, **Total Cores**, **Avg CPU**, **Avg Memory**, **Avg Disk**, **Server Status Map**
- 시스템 운영 통계: **CPU Resource Map**
- 조기 경보 및 알림: **CPU TOP5**, **Memory TOP5**, **Disk I/O TOP5**, **프로세스 CPU TOP5**, **프로세스 TOP5**, **실시간 알림**

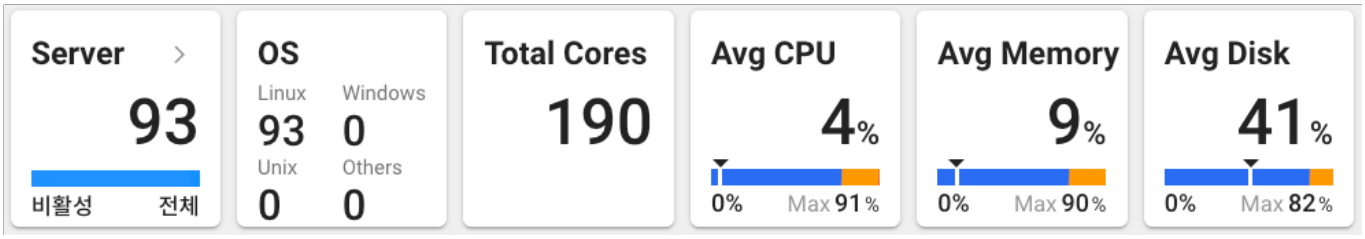
서버 모니터링 핵심 지표

- **CPU**: CPU의 성능 지표는 서버의 성능을 파악하는 가장 중요한 척도로 사용합니다. CPU의 사용률이 일정 이상을 넘어가면 서비스에 영향을 주기 시작합니다. 문제 상황이 발생한다면 하드웨어를 추가로 구매하거나 CPU를 사용하는 애플리케이션의 성능을 조정하는 등의 방법을 포함하여 CPU 사용량의 관리 목표치에 도달하기 위한 조치를 취할 수 있습니다.

  > ⓘ IT 관련 솔루션들은 서버의 성능에 따른 가격 체계를 가지는 경우가 많습니다. 대부분 CPU의 코어를 기준으로 가격을 책정하는 것도 같은 이유입니다.

- **Memory**: 버퍼 및 캐시 메모리를 포함하여 메모리의 사용량을 확인합니다. 메모리의 사용량이 너무 빨리 소모되거나 지속적으로 사용량이 떨어지지 않는다면 메모리 사용량의 관리 목표치에 도달하기 위한 조치를 취할 수 있습니다.
- **Disk I/O**: Disk I/O는 네트워크 드라이브를 사용하는 경우 꼭 확인해야 하는 모니터링 요소입니다. 디스크의 읽는 속도, 쓰기 속도, 대기열, 대기 시간의 비율 등을 모니터링합니다.
- **Network**: 네트워크 지표는 네트워크 인터페이스의 입출력 트래픽 속도와 오류 패킷 등을 모니터링합니다.

서버 상태 요약

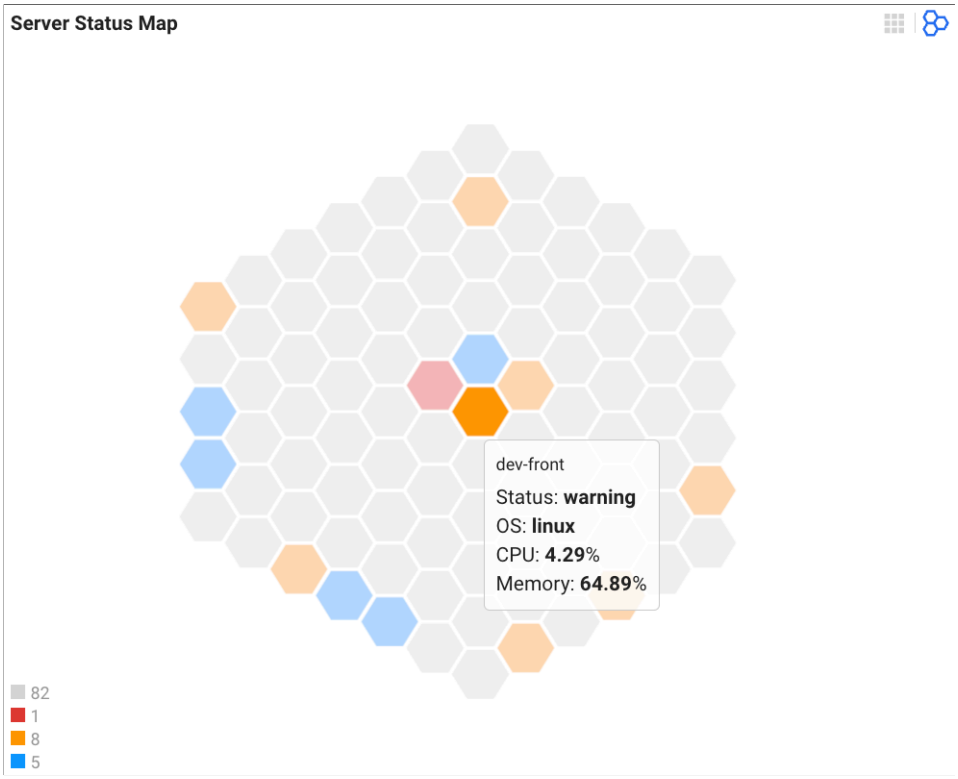


사용자는 **리소스 보드**의 상단 인포 패널을 통해 프로젝트에 등록된 전체 서버에 대한 요약 지표들을 쉽게 확인할 수 있습니다.

Server 위젯에서 전체 서버의 수와 장애가 있는 서버의 수를 조회할 수 있습니다. > 버튼을 클릭하면 **서버 목록** 메뉴로 이동합니다. **상태** 컬럼에서 조치가 필요할 수 있는 위험(빨간색), 향후 문제 발생의 가능성을 미리 알리는 경고(주황색) 단계의 예외 상태와 보통(녹색), 비활성(회색) 상태를 아이콘과 색으로 쉽게 구분해 확인할 수 있습니다. **서버 목록**에 대한 자세한 설명은 다음 문서를 참조하세요.

OS 위젯은 프로젝트 내 운영체제의 수, **Total Cores**는 전체 서버들 코어의 합, **Avg CPU**는 전체 서버들의 CPU 평균 사용량, **Avg Memory**는 전체 서버들의 메모리 평균 사용량, **Avg Disk**는 전체 서버 디스크 장치들의 평균 사용량을 나타냅니다.

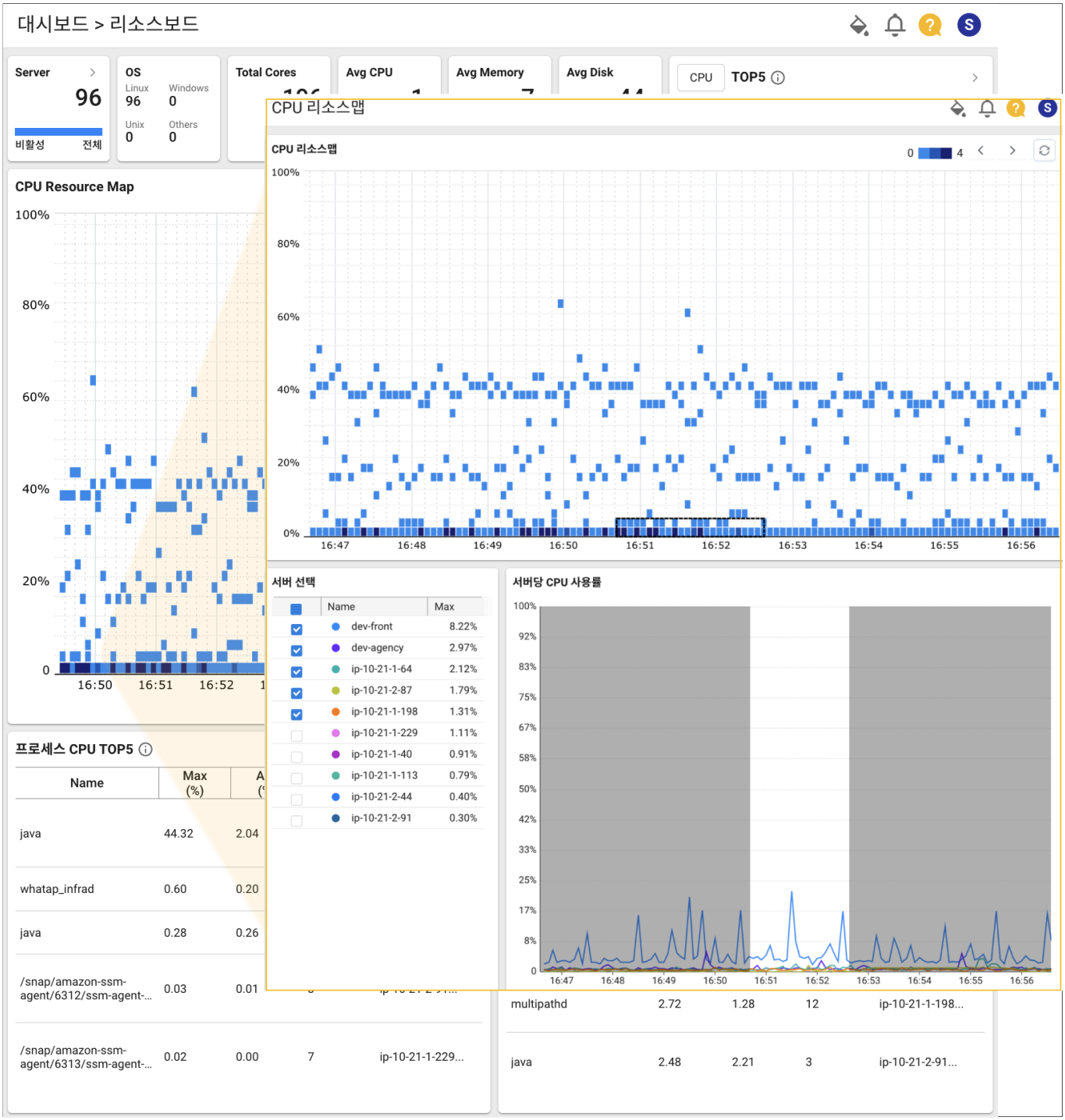
또한 **리소스 보드** 중앙 메인 차트의 아이콘을 클릭해 **Server Status Map**으로 이동하면 벌집 차트를 통해 프로젝트 내 서버의 상태를 한눈에 볼 수 있습니다. 벌집 차트 뷰는 서버 대수가 많을 경우 유용합니다. 개별 육각형 하나가 서버 에이전트 하나를 의미합니다. 문제가 발생한 서버는 색깔로 시각화하여 직관적인 파악이 가능합니다. 개별 육각형을 클릭하면 해당 서버 에이전트의 **서버 상세** 페이지로 이동합니다.



와탭 **리소스 보드**의 각 위젯을 활용해 전체 자원 사용량의 윤곽과 서버 상태를 쉽게 살필 수 있습니다.

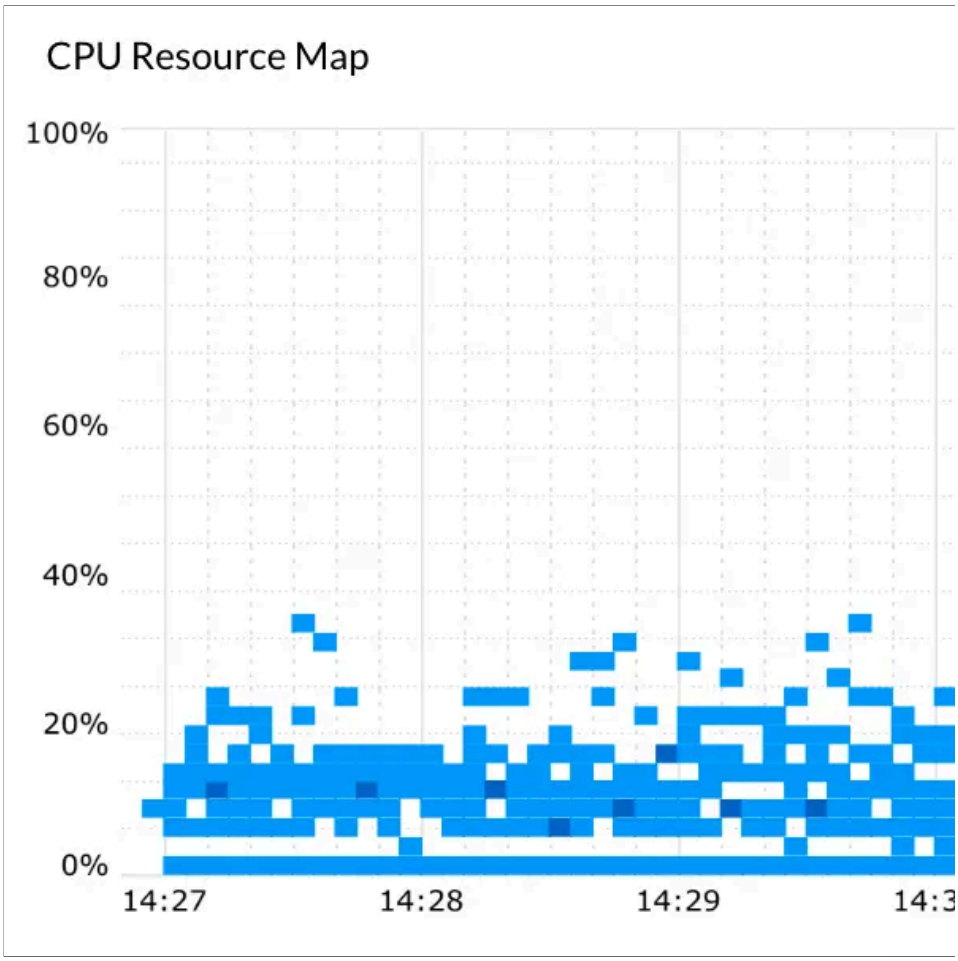
시스템 운영 통계

프로세스 정상 동작을 확인하기 위해 가장 중요한 요소는 CPU 사용량입니다. CPU 지표가 시스템 부하 상황을 가장 빠르게 반영하기 때문입니다. 와탭은 전체 시스템 운영 상황을 한눈에 확인할 수 있도록 CPU 사용량을 분포도 차트로 제공합니다. **리소스 보드** 중앙의 아이콘을 클릭하면 **CPU Resource Map** 위젯을 조회할 수 있습니다. **CPU Resource Map** 위젯에서 프로젝트 내 전체 서버의 CPU 사용량 분포도를 확인할 수 있습니다. 셀 영역을 드래그해 해당 구간의 상세 정보를 조회해 보세요.



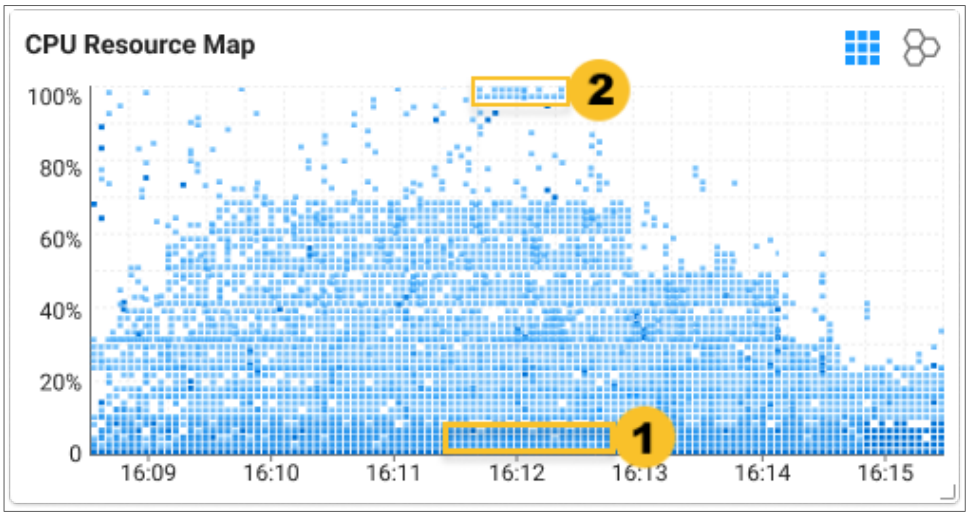
클라우드 IT 자산을 효율적으로 운영하려면 자원의 **과다**와 **과소** 상태를 모두 피하고 사용량을 적절하게 조절해야 합니다. 다음의 예시 화면은 운영 최적화를 위해 자원 사용량 기준을 50% 전후로 삼아 비용 및 성능 효율의 균형을 추구하고 있습니다. 예시와

같은 경우 사용량 급증 시 50%의 성능 마진으로 일시적인 장애를 회피할 수 있습니다.

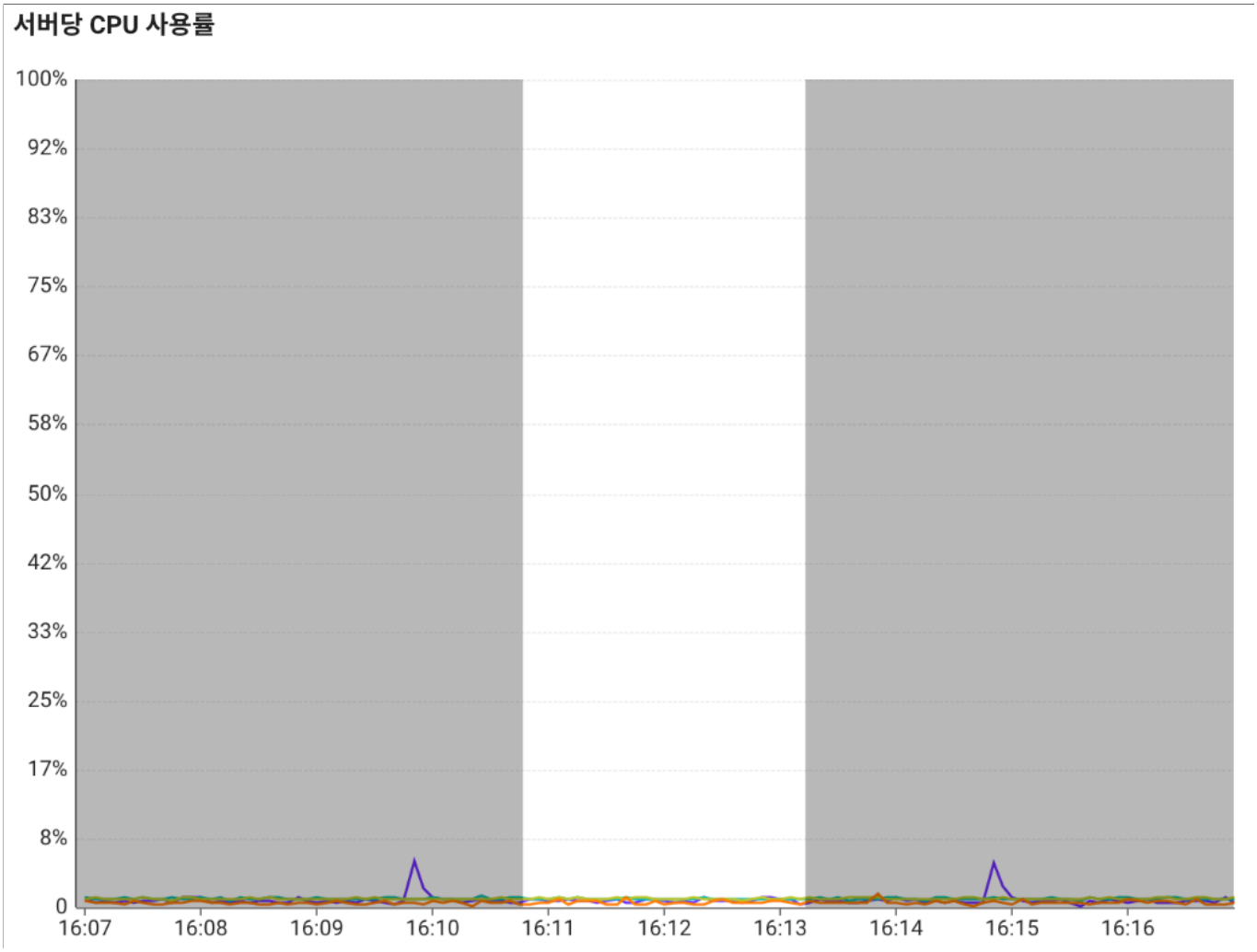


과다 투입은 서비스에 비해 시스템이 과하게 설치되어 인프라 비용이 필요 이상으로 지출되고 있는 상태입니다. 시스템 안정성이 보장되기에 사용자가 피크 상황에서 받는 영향이 크지 않습니다. 그래서 모니터링에 투자하는 시간을 줄이기 위해 사용량의 평균값을 낮추는 과다 투입 경향을 보이는 경우가 있습니다. 하지만 과다 투입으로 가는 경향이 커질 수록 비용이 증가하기에 빠른 상태 파악이 중요합니다. 과소 투입은 시스템 변경 등으로 인한 부하 증가 때문에 시스템 사용량이 관리 범위를 벗어난 경우로 서비스의 품질 악화 및 사용 편의성 저하를 야기합니다. 사용자의 서비스 이탈로 이어지기 전에 적절한 조치가 필요합니다.

와탭의 **CPU Resource Map**으로 과다 투입과 과소 투입 상황을 쉽게 파악할 수 있습니다. 맵의 하단은 대체로 과다 투입 경향을 보이고, 상단은 과소 투입 경향을 보입니다.

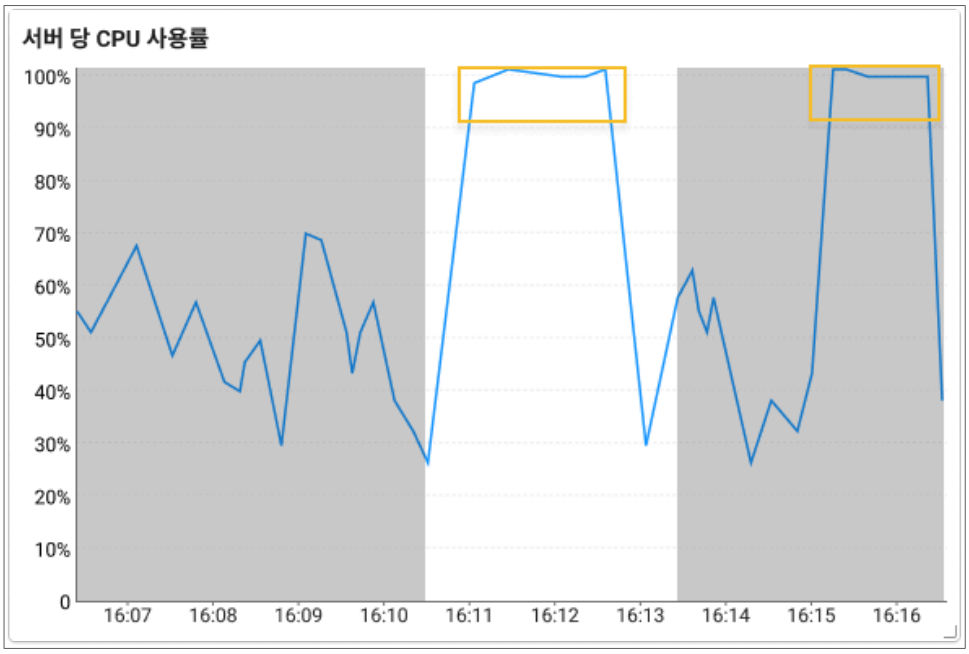


예시 화면 **CPU Resource Map** 하단의 1 영역을 드래그해 상세 정보를 조회 시 **서버당 CPU 사용률** 그래프에서 다음과 같이 과다 투입 경향을 확인할 수 있습니다.



이는 높은 경향성을 제시한 것으로 하단에서 조회되는 서버가 모두 과다 투입 상태인 것은 아닙니다. 주기적인 피크 부하가 있는 서버의 경우 차트 해석에 주의해야 합니다.

CPU Resource Map 상단의 2 영역을 드래그해 상세 정보를 조회 시 **서버당 CPU 사용률** 그래프에서 과소 투입 경향을 확인할 수 있습니다. 다음의 상세 정보 예시는 전형적인 CPU 부족 즉 Starvation 상태 화면으로 해당 서버는 피크와 해소가 반복되는 요주의 서버입니다.



IT 자원은 끊임없는 효율과 개선의 대상으로 지속적인 검토가 필요합니다. 와탭의 서버 모니터링은 비정상 상황을 빠르게 확인하고 프로세스를 바로 조회할 수 있도록 엔지니어의 노하우를 시각화에 반영했습니다. **CPU Resource Map**을 활용해 자원 과다 및 과소 상태를 쉽게 파악하는 것과 더불어 다음에 설명할 자원 사용량 상위 5개 목록 위젯으로 핵심 프로세스를 한눈에 확인할 수 있습니다. 여러 가지 사전 설정을 요구하는 불필요한 차트 구성을 줄이고 데이터 집약적인 와탭의 대시보드를 통해 과대와 과소 투입 상태 및 핵심 프로세스를 빠르게 조회해 보세요.

조기 경보 및 알림

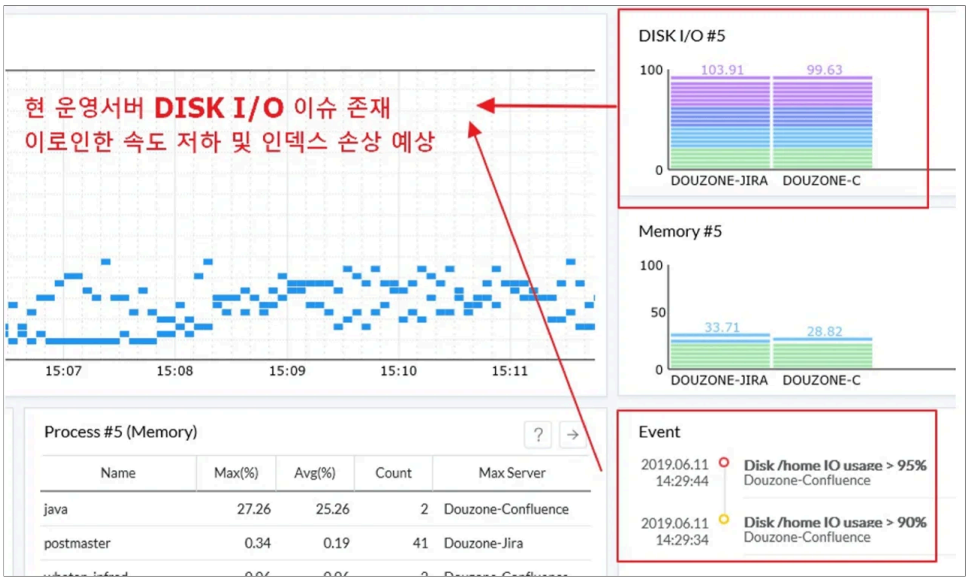
와탭의 **리소스 보드**는 자원 사용량이 가장 높은 서버와 프로세스 상위 5개 목록 차트를 제공합니다. 사용자가 문제 소지가 있는 서버를 파악할 수 있도록 돕는 조기 경보 관점의 위젯으로 각각 우측과 하단에서 확인할 수 있습니다.

자원 사용량이 높으면 문제 상황이 반드시 발생합니다. 와탭은 Top 5 목록 차트를 통해 서버 모니터링의 핵심 지표인 CPU, Memory, Disk I/O, Network 등의 자원 사용량이 높은 요주의 대상 서버와 CPU, Memory 사용량이 높은 프로세스를 한눈에 조회하고 이슈를 조기에 해결할 수 있도록 돕습니다.

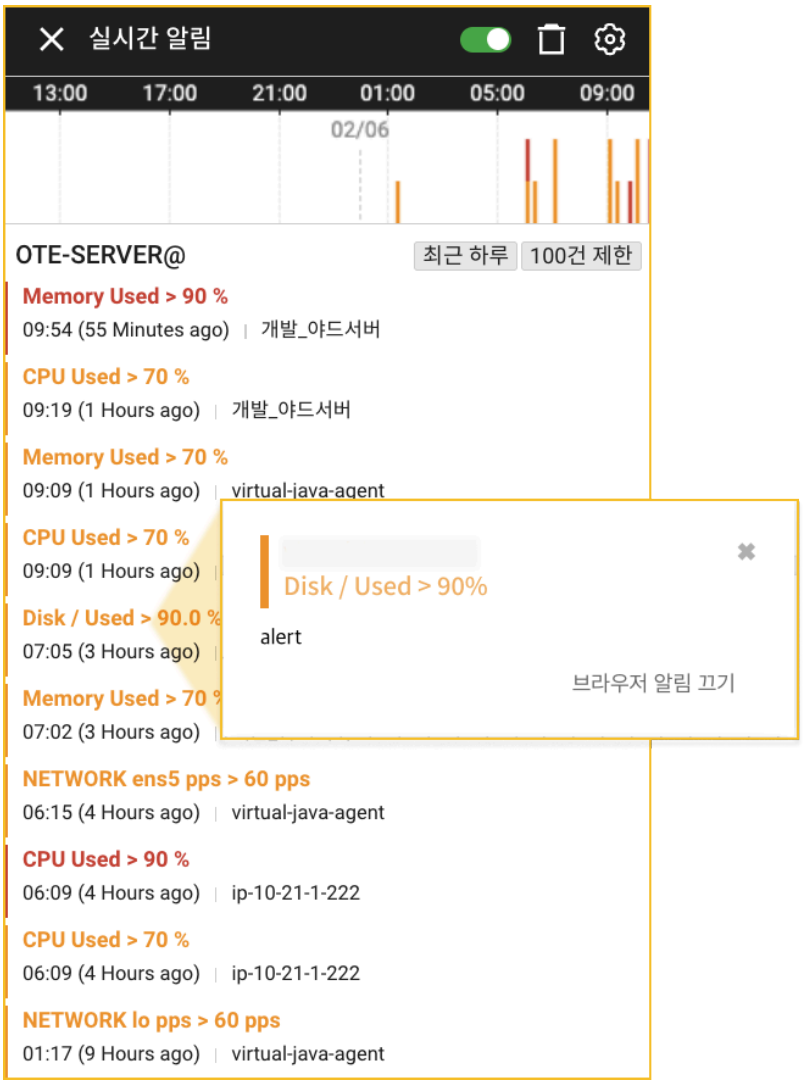
더 큰 문제 상황이 발생하기 전에 와탭의 **리소스 보드**를 통해 시스템의 주요 이슈를 탐지한 사례를 하나 소개하겠습니다. 다음 예시 화면의 **Disk I/O Top 5** 위젯을 보면 가용량 대비 Disk I/O가 높은 장비의 현황을 파악할 수 있습니다.

예시의 현황 정보를 기반으로 동일 장비에서 발생한 장애 이력을 검토한 결과, 속도 저하 현상과 인덱스 파일 손상 등의 이상 상황이 발생한 것을 확인할 수 있었습니다. 이에 SSD를 도입하고 대용량으로 도입하기 어려운 부분은 NAS 스냅샷 백업을

활용하는 조치를 취해 Disk I/O 이슈를 해결했습니다.



최근 이벤트 추이를 확인할 수 있는 **실시간 알림**은 **리소스 보드** 상단의 고정 메뉴에서 🔔 버튼을 클릭해 확인할 수 있습니다.
와탭은 독자적인 알림 임계치 기본 설정을 제공합니다. 에이전트를 설치하고 데이터가 수집되면 사전 설정 없이도 그때부터 알림을 확인할 수 있습니다. 예를 들어 Disk I/O(%) 지표는 5초 동안의 디스크 사용률을 보여줍니다. Disk I/O(%)가 80%를 넘으면 시스템 성능에 영향을 줄 수 있고 100%라면 디스크가 쉬지 않고 일하고 있다는 의미입니다. 와탭 서버 모니터링에서 기본으로 설정된 Disk I/O(%)의 경고 값은 90%입니다. 다시 말해 90%가 넘으면 사용자가 임계치 세부 설정을 하지 않았더라도 바로 알림이 발생합니다.



와탭의 기본 알림 설정은 사용자가 복잡한 추가 과정 없이도 에이전트 설치와 동시에 모니터링을 시작할 수 있게 합니다. 간편하게 설정할 수 있는 프로세스 알림 등 서버 모니터링 알림 설정에 대한 자세한 내용은 다음 문서를 참조하세요.

리소스 보드는 전체 서버의 현황 정보를 요약하여 간결하고 이슈 위주로 볼 수 있는 효율적인 대시보드입니다. CPU 이슈가 존재하는 서버의 대수 및 추이를 동시에 감지하기 위한 메인 차트(**CPU Resource Map**)와 OS 모니터링의 핵심 지표별 Top 5 목록을 통해 이슈 발생 가능성이 높은 대상 자원을 노출시키며 서버에서 발생한 알림 내역을 최신 순으로 표시합니다. 와탭의 직관적이고 간결한 대시보드 구성은 대규모 시스템을 모니터링 해야 하는 경우 더욱 유용합니다.

> ⓘ **대시보드 추가 활용**
>
> - 자원별 Top 5 위젯에서 > 버튼을 클릭하면 **리소스 이퀄라이저** 메뉴로 이동해 전체 서버에 대한 실시간 사용량을

볼 수 있습니다.

- 서버 단위의 상세 정보는 위젯 내 해당 차트 영역을 클릭해 이동하는 **서버 상세** 메뉴에서 확인할 수 있습니다.
- 서버 자원 소모 패턴을 조회하거나 부하 설계와 비교하고자 한다면 **분석** > **메트릭 차트** 메뉴를 활용하세요.

TCP 포트 감시 활용

이 문서는 TCP 포트 모니터링이 왜 필요한지, 와탭 서버 **이벤트 설정** 메뉴에서 어떻게 활성화하는지를 정리합니다.

TCP 포트 감시

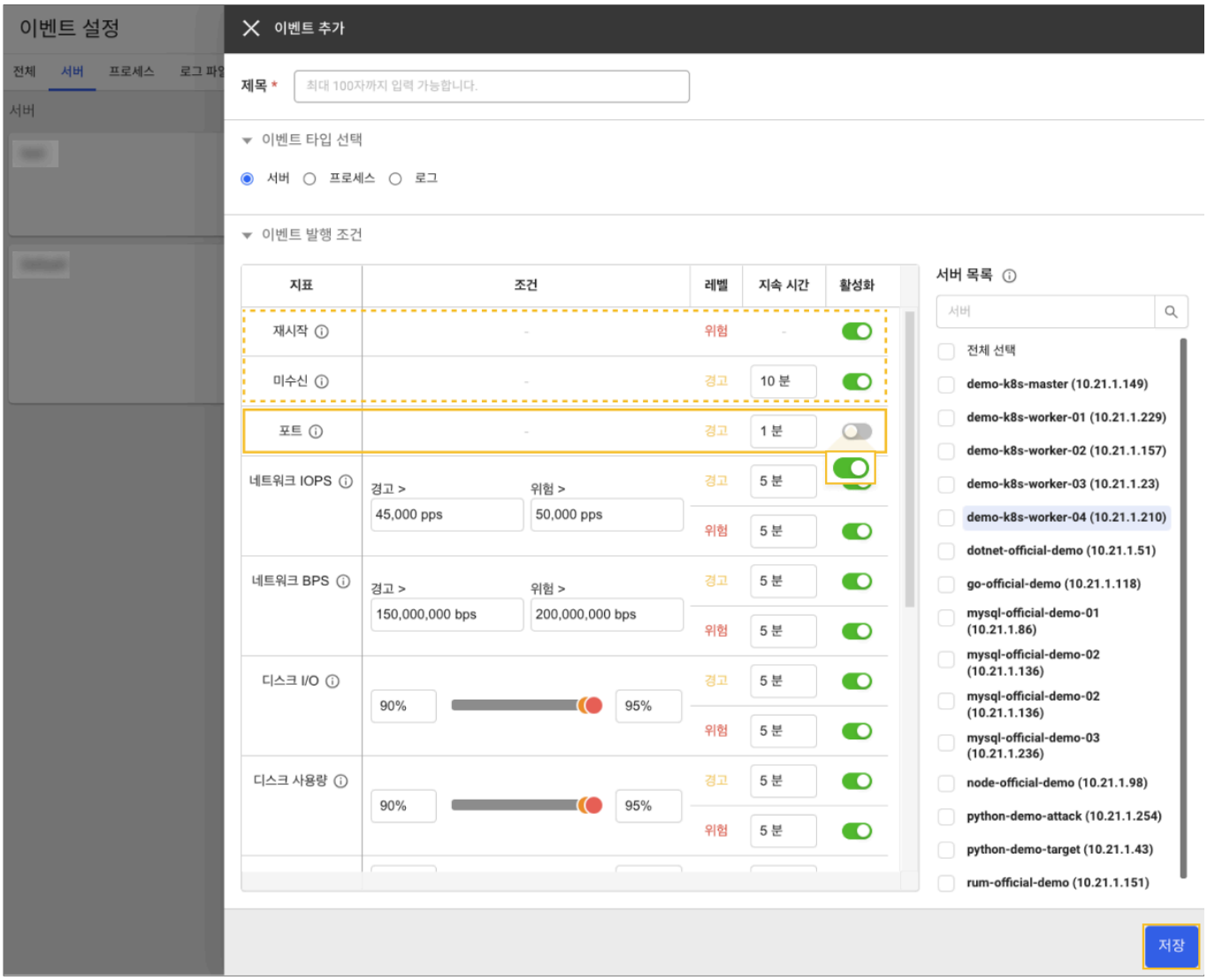
IT 인프라 관리에서 TCP 포트 모니터링은 운영 중인 서비스의 상태를 확인하는 기본적이면서도 중요한 방법 중 하나입니다. TCP 포트는 네트워크상의 애플리케이션·서비스가 통신하는 주요 경로입니다. 각 서비스는 특정 TCP 포트로 통신하며, 포트 상태가 곧 서비스의 가용성과 성능을 직간접적으로 나타냅니다. TCP 포트 모니터링은 서비스가 정상적으로 작동하는지를 실시간으로 감시하여 시스템의 안정성을 보장하는 데 중요한 역할을 합니다.

예를 들어 웹 서버가 사용하는 80 또는 443과 같은 HTTP 포트가 응답하지 않을 경우 웹 서버가 다운되었거나 네트워크 문제로 인해 접근이 차단되었을 수 있습니다. 이러한 상황에서 TCP 포트 모니터링은 문제를 신속하게 감지하고 알림을 발송하여 관리자가 즉각적으로 대응할 수 있도록 지원합니다.

와탭의 서버 모니터링 에이전트는 모니터링 대상 서버의 상태를 실시간으로 확인할 수 있을 뿐만 아니라, 서버와 다른 서비스 간의 TCP 통신 상태까지 모니터링합니다. 이를 통해 서비스 중단을 예방하고 시스템의 가용성을 극대화하여, 비즈니스 연속성을 유지할 수 있습니다.

알림 활성화

> ⓘ 홈 화면 > 프로젝트 선택 > **경고 알림** > **이벤트 설정** > **서버** 탭 클릭 후 **+ 이벤트 추가** 버튼을 클릭하세요.



기본 알림

와탭 서버 모니터링은 **기본 알림**을 제공합니다. 사용자는 다음과 같이 이벤트 템플릿 내에서 서버 **재시작** 및 데이터 **미수신** 항목을 활성화하는 것만으로 추가적인 조작 없이 기본 알림 설정을 완료할 수 있습니다.

- **재시작**: 모니터링 대상 서버가 시스템 재시작을 수행할 경우 알림이 발송됩니다.
- **미수신**: 서버의 다운타임이나 네트워크 문제로 인해 설정된 시간 동안 모니터링 에이전트로부터 데이터가 수신되지 않을 경우 해당 상태에 대한 알림이 발송됩니다.

TCP 포트 감시 알림

와탭 서버 모니터링 이벤트 템플릿에서 **포트** 항목을 활성화 후 다음 스크립트를 통해 모니터링 대상 엔드포인트와 TCP 포트를 설정하세요.

- Linux Shell Windows Powershell

```
#아래 변수에 모니터링 대상 아이피를 지정합니다.
export TARGET_IP=127.0.0.1
#아래 변수에 모니터링 대상 포트를 지정합니다.
export TARGET_PORT=80
echo "tcp.check.$TARGET_PORT=tcp://$TARGET_IP:$TARGET_PORT" |sudo tee -a /usr/whatap/infra/conf/whatap.conf
```

```
#아래 변수에 모니터링 대상 아이피를 지정합니다.
$TARGET_IP="127.0.0.1"
#아래 변수에 모니터링 대상 포트를 지정합니다.
$TARGET_PORT="80"
Add-Content -Path "C:\Program Files\WhatapInfra\whatap.conf" -Value
"tcp.check.$TARGET_PORT=tcp://$TARGET_IP:$TARGET_PORT"
```

활성화 확인

whatap.conf

Linux 환경에서 `127.0.0.1` 으로 설정이 완료된 경우 /usr/whatap/infra/conf 경로의 whatap.conf 파일에 다음과 같은 내용이 추가됩니다.

whatap.conf

```
license=xxxxxxxxxx
whatap.server.host=xxx.xxx.xxx.xxx
createdtime=xxxxxxxxxx
tcp.check.8080=tcp://127.0.0.1:8080
```

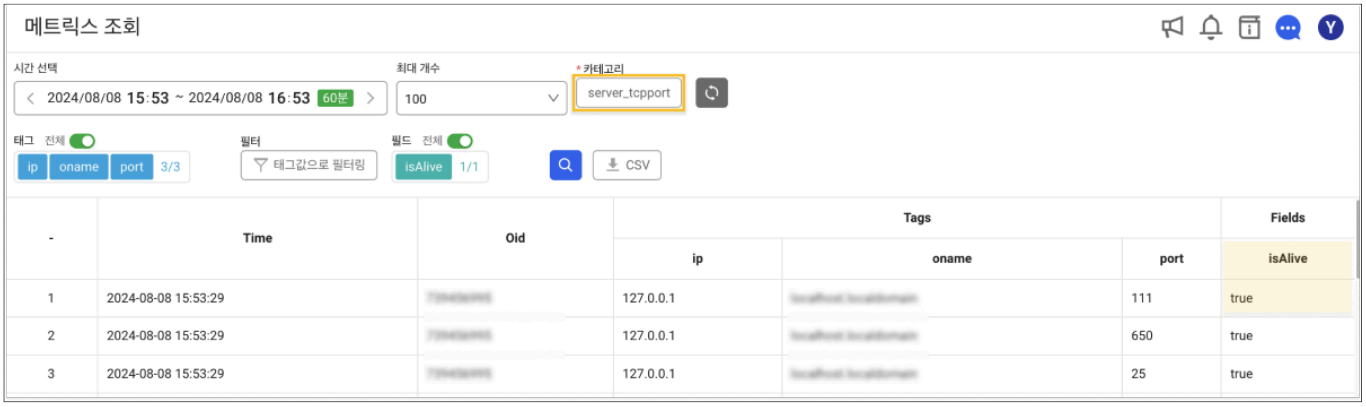
> **다중 포트 감시**
>
> 여러 개의 포트를 감시하기 위해서는 whatap.conf 파일 내 추가 설정이 필요합니다. 다음 예시를 참고하세요.

whatap.conf

```
# port 8080, port 80
tcp.check.8080=tcp://127.0.0.1:8080
tcp.check.80=tcp://127.0.0.1:80
```

메트릭스 조회

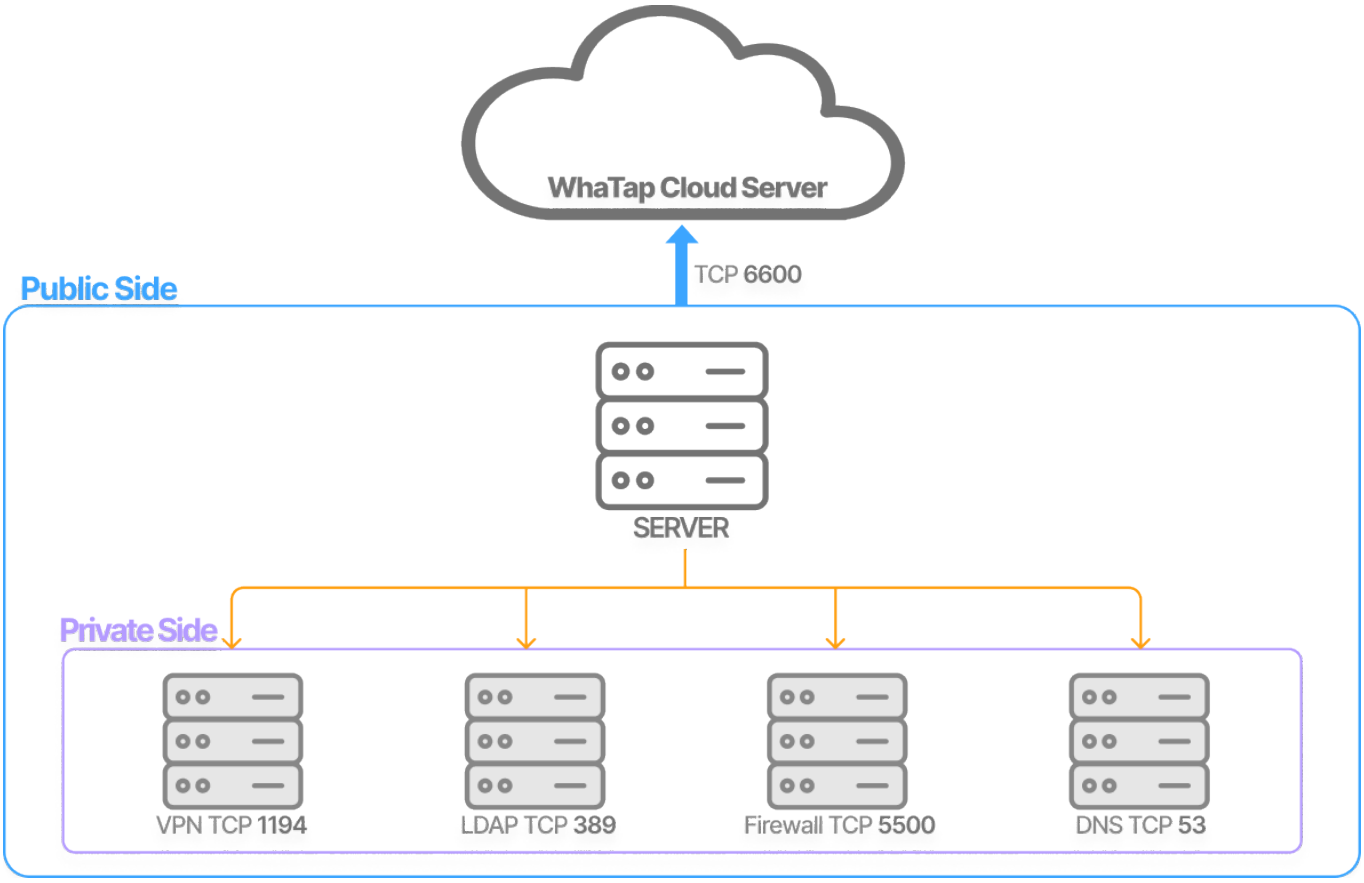
설정이 완료된 경우 **분석** > **메트릭스 조회** 메뉴에서 `server_tcpport` 카테고리가 추가됩니다. 해당 카테고리의 `isAlive` 필드 값에 따라 메트릭스 알림을 사용할 수 있습니다.



-	Time	Oid	Tags			Fields
			ip	oname	port	isAlive
1	2024-08-08 15:53:29	[redacted]	127.0.0.1	[redacted]	111	true
2	2024-08-08 15:53:29	[redacted]	127.0.0.1	[redacted]	650	true
3	2024-08-08 15:53:29	[redacted]	127.0.0.1	[redacted]	25	true

와탭 서버 모니터링과 TCP 포트 감시

와탭의 서버 모니터링에서 제공하는 TCP 포트 감시 기능을 활용하면, 에이전트를 설치할 수 없는 환경이나 모니터링 데이터를 외부로 직접 전송할 수 없는 프라이빗 네트워크 영역에서도 TCP 상태를 효과적으로 확인할 수 있습니다.



와탭의 서버 에이전트는 모니터링 대상 서버뿐만 아니라, 해당 서버 내부의 다른 엔드포인트에 대해서도 주기적으로 TCP 상태를 점검합니다. 따라서, 네트워크 통신이 가능한 환경에서는 에이전트를 통해 보다 포괄적이고 광범위한 범위의 모니터링을 수행할 수 있습니다. 이러한 기능은 복잡한 네트워크 환경에서의 서버 상태 및 연결 상태를 철저하게 감시하여, 장애 발생 시 신속한 대응을 가능하게 합니다.

이와 같은 기능을 활용하면 운영 중인 시스템의 네트워크와 서버 상태를 신속하게 파악하고 대응할 수 있습니다.

Kubernetes 옵저버빌리티 확보하기

Kubernetes 환경은 컨테이너가 수시로 뜨고 사라지는 **동적 토폴로지**이기 때문에 "CPU가 몇 %인가" 같은 상태 확인(모니터링)만으론 문제 대응이 어렵습니다. "왜 이 Pod가 재시작되는가", "왜 이 서비스가 느려졌는가"를 추적할 수 있는 **옵저버빌리티**가 필요합니다.

이 가이드는 WhaTap Kubernetes Monitoring으로 Cluster·Pod·Service 계층을 어떻게 묶어 보는지, 어떤 메뉴를 언제 쓰는지 정리합니다.

옵저버빌리티의 4가지 신호

Kubernetes 환경을 이해하려면 다음 네 가지 데이터를 **교차해서** 봐야 합니다.

표 | 옵저버빌리티의 4가지 신호

신호	역할	와탭에서 보는 곳
Metric	자원 사용률·수량 변화 추세	Cluster·Node·Pod 대시보드
Trace	서비스 간 호출 흐름·지연 구간	트랜잭션 트레이스
Log	시간 단위의 사건 원본	컨테이너 로그
Event	K8s 자체 이벤트(Pod 재시작 등)	K8s 이벤트 스트림

단편 데이터 하나로는 원인을 찾기 어렵습니다. 예를 들어 Pod CPU 스파이크(Metric)를 봤다면, 같은 시점 Trace로 어느 요청이 CPU를 썼는지, Log로 무슨 작업이었는지, Event로 스케줄링 변화가 있었는지를 함께 봐야 합니다.

WhaTap Kubernetes Monitoring의 통합 관점

와탭은 **인프라 (Cluster/Node) → 컨테이너 (Pod) → 애플리케이션 (Service)** 세 계층을 한 프로젝트 안에서 일원화된 시각으로 제공합니다.

- **Cluster**: 클러스터 전체 자원·상태 요약

- **Node**: 워커 노드별 CPU·메모리·네트워크·디스크
- **Pod / Container**: 컨테이너 단위 자원 사용 + 재시작 이력
- **Service (애플리케이션)**: WAS/앱 APM 연동으로 Trace·에러 분석

덕분에 **인프라 관리자와 애플리케이션 개발자가 같은 실시간 데이터를 보며 커뮤니케이션**할 수 있습니다.

상세 기능: Kubernetes 모니터링 소개

과거 시점 분석도 가능

문제가 발생한 **과거 시점**(배포 전후, 특정 장애 시간)의 Trace·Metric·Log·Event를 자동 수집·저장하므로, 재현 없이 과거 상황을 그대로 돌려볼 수 있습니다. "어제 새벽 3시에 발생한 Pod Evict이 왜 일어났는지"를 지금 앉아서 분석할 수 있습니다.

관찰 시나리오 예시

특정 서비스가 느려졌을 때

1. **서비스 대시보드**에서 응답 시간·에러율 이상 확인
2. **트랜잭션 트레이스**로 느린 요청의 호출 스택 확인 → DB/외부 호출 중 어느 곳이 병목인지
3. 해당 Pod의 **컨테이너 리소스** 확인 → CPU/메모리 고갈 여부
4. 동시간대 **K8s 이벤트**에서 Pod 재시작/스케줄링 변화 확인
5. 필요시 **컨테이너 로그**에서 에러 메시지 조회

Pod가 반복 재시작될 때

1. **K8s 이벤트**에서 재시작 횟수·사유(OOMKilled, CrashLoopBackOff 등) 확인
2. 재시작 전 시점의 **Pod 메모리 사용률**·**애플리케이션 로그**를 타임라인에 맞춰 비교
3. 트래픽 급증 여부는 상위 **Service 대시보드**에서 교차 확인

다음 단계

- **설치·초기 설정** → Kubernetes 모니터링 설치
- **컨테이너 맵으로 네트워크 관계 파악** → 컨테이너 맵
- **애플리케이션 연동** → Kubernetes + APM 통합 관찰 (APM 자동 설치)
- **GPU 워크로드 모니터링** → GPU 대시보드
- **장애 대응 루틴** → 장애 대응 시나리오

사용자 체감 성능 모니터링 (RUM)

모던 웹 애플리케이션(특히 SPA)은 **클라이언트에서 일어나는 일**이 사용자 경험의 핵심입니다. 서버 응답이 아무리 빨라도 브라우저 렌더링·AJAX 호출이 느리면 고객은 "느린 서비스"로 받아들입니다. 이 가이드는 WhaTap 브라우저 모니터링으로 **실제 사용자 성능 (RUM, Real User Monitoring)**을 지속 감시하는 방법을 다룹니다.

Synthetic Monitoring vs RUM — 차이

Synthetic Monitoring은 미리 정의한 시나리오를 일정 주기로 실행해 **가상 환경에서 성능·가용성·기능을 측정**하는 방식입니다. 트래픽이 없는 새벽 시간대에도 24/7 가용성을 체크할 수 있고, 신규 배포 전 회귀 테스트의 베이스라인을 잡거나 지역별 외부 위치에서의 응답 속도를 검증할 때 유용합니다. Lighthouse, PageSpeed Insights, URL 모니터링이 대표적인 도구입니다.

RUM (Real User Monitoring)은 **실제 사용자의 브라우저에서 직접 수집한 성능 데이터**를 활용하는 방식입니다. 다양한 디바이스·네트워크·지역에서의 실제 체감 속도를 반영하므로, Synthetic Monitoring에서는 잡히지 않는 "현장의 실제 문제"를 드러냅니다.

핵심 차이는 **"이상적인 조건의 베이스라인"(Synthetic) vs "실제 사용자 경험"(RUM)** 입니다. 두 방식은 보완 관계이며, 한쪽만으로는 충분하지 않습니다. **서비스 품질 SLA**는 결국 RUM 데이터로 관리해야 합니다.

도구	측정 방식	강점	한계
브라우저 DevTools	개발자 로컬	디버깅·세밀 조정	지속 모니터링 부적합
Google Lighthouse	Synthetic	성능·접근성·SEO 종합	실 사용자 데이터 반영 안 됨
PageSpeed Insights	Synthetic + CrUX	Core Web Vitals 기준	특정 조건 분석·심층 추적 한계
Web Vitals Extension	브라우저 확장	Core Web Vitals 빠른 체크	복잡한 성능 문제 추적 부족
WhaTap RUM	**실 사용자**	**실제 체감 성능·지속 추적·에러 코드 위치까지**	—

WhaTap 브라우저 모니터링의 RUM 관점

① 페이지 로드 · AJAX 실시간 통계

사용자가 실제로 연 페이지의 로드 시간, AJAX 호출 응답 시간을 **실시간 대시보드**로 확인할 수 있습니다.

- **페이지 로드 분포**: 빠른·느린 사용자 비율 한눈에
- **AJAX 호출 성능**: API별 실패율·응답 시간
- **드래그 조작**으로 느린 구간만 선별 → 상세 분석

② 에러 추적 — 코드 라인까지

실 사용자 브라우저에서 발생한 JavaScript 에러를 수집해 **에러가 발생한 파일과 코드 라인**을 빠르게 찾을 수 있습니다. 문제 재현·원격 디버깅 없이 빠른 원인 파악 가능.

AI 에러 스택 분석과 결합하면 해석 시간이 더 줄어듭니다.

③ 사용자 접속 환경 분석

브라우저·OS·디바이스·지역별 성능 분포를 확인해 **특정 환경에서만 나타나는 문제**를 조기에 감지할 수 있습니다.

활용 시나리오

신규 기능 배포 후 사용자 체감 변화 확인

1. 배포 직전·직후 **페이지 로드 시간 분포** 비교
2. AJAX 호출 중 **새로운 느린 엔드포인트** 등장 여부
3. 에러 추적에서 **새로운 JS 에러** 발생 여부
4. Synthetic Monitoring 도구 결과와 교차 검증 → 릴리즈 검증 시나리오

SLA·Core Web Vitals 관리

- 월간 성능 리포트에 **RUM 기반 p75 응답 시간**, LCP/FID/CLS 같은 Core Web Vitals 포함 → 성능 리포팅 시나리오
- SLA 미준수 비율 추적

장애 알림의 첫 신호

서버 지표는 정상인데 사용자 쪽에서 체감 저하가 올 수 있음 (CDN, DNS, 네트워크 경로 등). RUM은 **서버 밖의 문제**까지 감지하는 감지망 역할 → 첫 경고 알림 붙이기에 RUM 기반 규칙도 추가 고려.

기존 브라우저 모니터링 가이드와의 관계

- **브라우저 모니터링 활용하기** — 페이지 로드·AJAX·접속 환경·에러 4종 분석 방법
- **이 가이드(RUM 관점)** — 실 사용자 데이터를 "SLA·사용자 경험 지표"로 관리하는 관점

둘을 함께 읽으면 기능 이해와 운영 관점이 함께 잡힙니다.

다음 단계

- **브라우저 설치·설정** — 브라우저 모니터링 소개
- **에러 AI 분석** → AI 브라우저 에러 스택 분석 (tracking-error.mdx 참조)
- **Synthetic Monitoring과 조합** → URL 모니터링: URL
- **사용자 측 이상 + 서버 측 연결** → 장애 대응 시나리오

Browser 모니터링 활용하기

이 문서는 와탭 Browser 모니터링으로 웹 애플리케이션의 **페이지 로드, AJAX, 사용자 접속 환경, 에러** 4개 영역을 분석하고 성능 저하 원인을 파악하는 방법을 정리합니다. 기본 안내는 Browser 모니터링 문서를 참조하세요.

페이지 로드 분석하기

웹 페이지 로드 속도는 사용자 경험에 큰 영향을 미칩니다. 페이지 로드가 느리면 사용자는 페이지를 떠나고 다른 사이트로 이동할 가능성이 높습니다. 이로 인해 사용자는 페이지를 볼 기회를 놓치게 되고 서비스나 제품을 이용하지 않거나 구매하지 않는 경우가 발생할 수 있습니다.

페이지 로드 속도를 최적화하고 사용자 경험을 개선하는 것은 매우 중요합니다. 페이지 로드 분석을 통해 어떤 요소가 페이지 로드를 느리게 만드는지 식별하고, 문제점을 개선하여 최적화된 사용자 경험을 제공할 수 있습니다. 이러한 노력은 사용자가 서비스나 제품을 더 많이 이용하고 더 긴 시간 동안 이용하도록 만들어 줍니다.

페이지 로드 분석은 모든 웹사이트와 서비스 운영에 필수적입니다. 사용자 경험을 개선하고 더 나은 비즈니스 성과를 달성할 수 있습니다.

브라우저 애플리케이션 상태 파악하기

Browser 모니터링 서비스의 **브라우저 모니터링 대시보드** 메뉴를 이용해 먼저 브라우저 애플리케이션의 페이지 로드 속도를 확인해야 합니다.

> ⓘ **브라우저 모니터링 대시보드** 메뉴에 대한 자세한 내용은 다음 문서를 참조하세요.

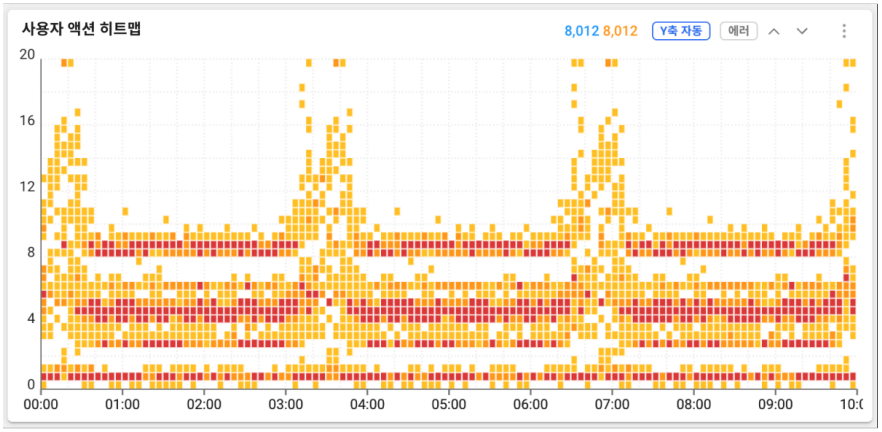
1. **타임 라인 위젯 확인하기**

페이지 로드 타임라인 위젯을 통해서 전체 페이지 로드 성능을 모니터링하세요. 실시간으로 브라우저 애플리케이션의 전반적인 성능을 파악할 수 있습니다. 어느 구간에서 페이지 로드가 느린지 확인할 수 있습니다.

2. **페이지 로드 시간 및 로드 수 확인하기**

많은 사용자가 접속하는 페이지의 로딩 시간이 길어진다면 해당 페이지의 성능을 개선하는 데 집중할 수 있습니다. 반대로 로딩 속도가 느린 페이지의 접속량이 크지 않다면, 해당 페이지의 개선보다는 다른 페이지에 더 많은 개발 리소스를 투자하는 것이 효율적일 수 있습니다.

3. **페이지 로드 히트맵 확인하기**



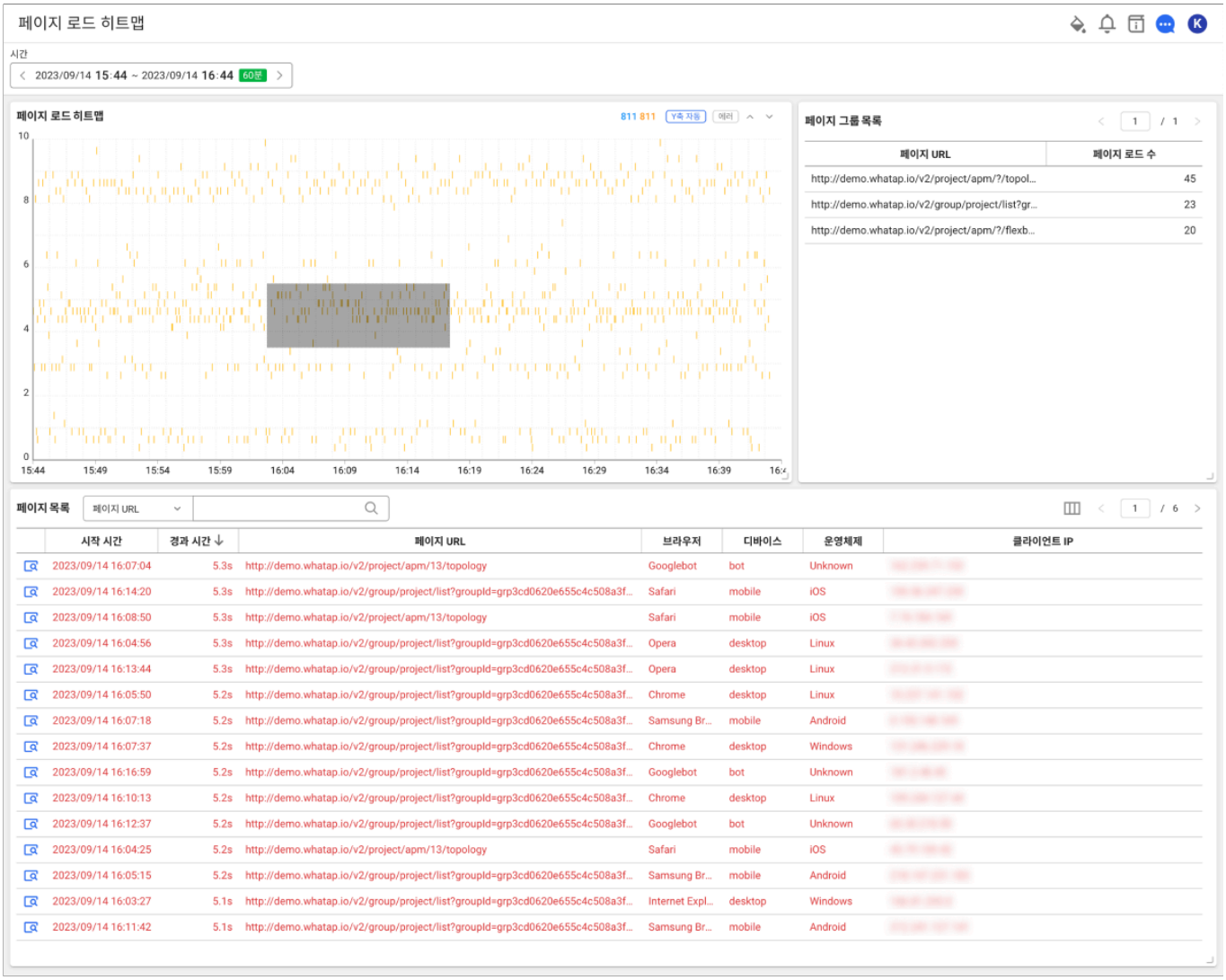
페이지 로드 이벤트를 히트맵 차트 형태로 제공합니다. 실시간으로 발생하는 페이지 로드 이벤트를 표시합니다. 페이지 로드가 느릴수록 차트 상단에 표시됩니다.

성능 저하 원인 파악 및 해결하기

페이지 로딩이 느린 페이지를 확인했다면 원인을 찾아 해결해야 합니다. **분석** > **페이지 로드 히트맵** 메뉴로 이동하세요. **페이지**

로드 히트맵은 시간에 따른 페이지 로드의 응답 시간을 분포도 차트로 표현합니다. 히트맵을 확인하면 개별 페이지 로드 이벤트에 대한 상세한 정보를 얻을 수 있습니다.

페이지 로드 히트맵 확인하기



1. **페이지 로드 히트맵**으로 브라우저 애플리케이션 상태를 파악하세요.

   특정 시간에 부하가 걸린 현상을 파악하거나 특정 페이지 로드 이벤트가 느린 현상을 파악할 수 있습니다. 차트 상단에 분포한 이벤트일수록 로딩 시간이 오래 걸린 경우이며 브라우저 애플리케이션의 작동 상태가 느리다는 것을 의미합니다.

2. 페이지 로드가 느린 영역을 드래그하면 **페이지 목록**에서 각 페이지 로드 이벤트를 확인하세요.

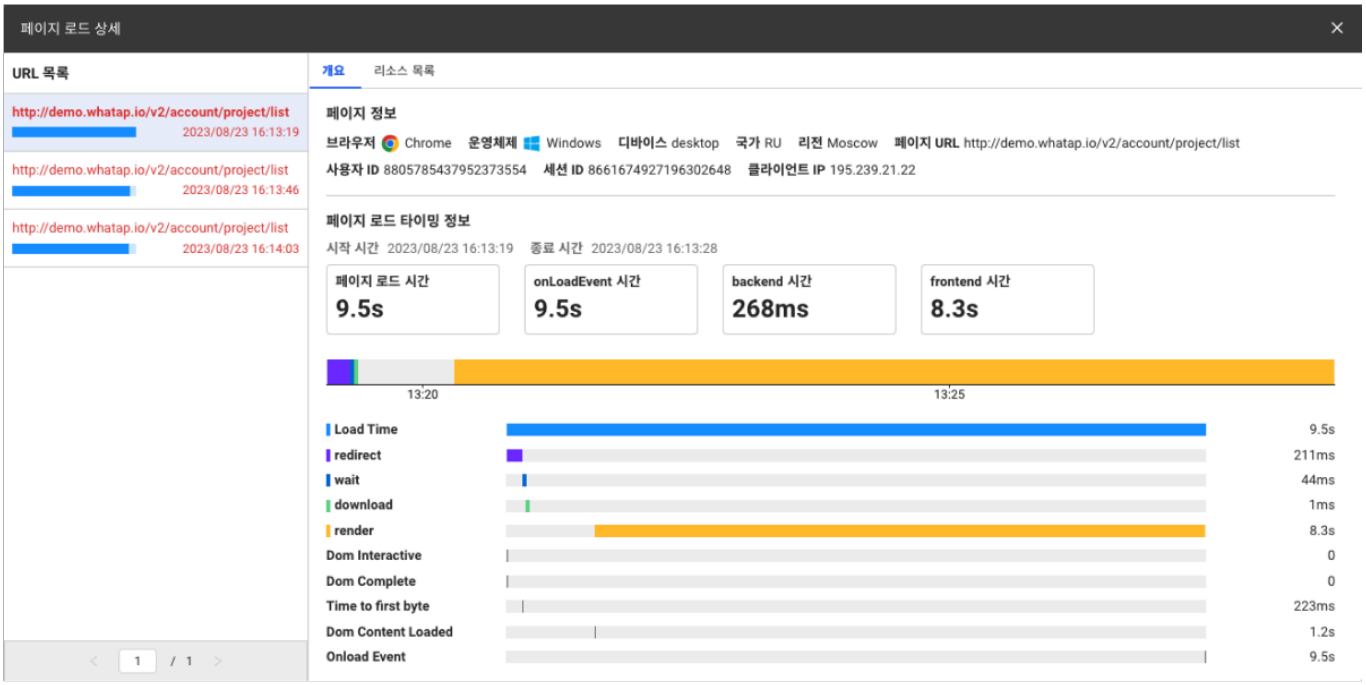
   페이지 목록에서는 기본적으로 경과시간, 페이지 URL, 지역 정보, 브라우저, 디바이스 정보 등을 확인합니다. 특정 환경 또는 페이지에서 발생한 문제를 파악하는데 활용할 수 있습니다.

3. 평균 페이지 로드 시간이 길었던 페이지를 필터링하여 볼 수도 있습니다.

   특정 페이지에 대한 페이지 로드 이벤트를 확인하려면 **페이지 그룹 목록**에서 필터링을 적용해 확인할 수 있습니다.

페이지 로드 상세 보기

페이지 목록에서 가장 왼쪽에 을 클릭하면 **페이지 로드 상세** 창이 나타납니다. 페이지 로드 성능을 분석하기 위한 다양한 지표와 성능에 영향을 주는 요소를 파악할 수 있습니다.



페이지 로드 과정 중 구간별 소요된 시간을 그래프 차트로 제공합니다. 차트를 통해 페이지 로드 과정 중 느린 구간을 파악하고 최적화를 진행할 수 있습니다.

- **페이지 정보**: 최종 사용자의 접속 환경, 접속 페이지 정보, 사용자 ID, 세션 ID 정보를 제공합니다. 이 정보들로 특정 환경에서만 발생하는 이상 현상을 판단해 볼 수도 있습니다.
- **페이지 로드 타이밍 정보**: 페이지 로드 시 구간별 소요 시간을 확인할 수 있습니다. 각 구간에 대한 자세한 내용은 다음 문서를 참조하세요.

- **리다이렉트**: 브라우저가 웹 페이지를 불러올 때 리다이렉트 과정에서 소요한 평균 시간입니다.
- **캐싱**: 브라우저가 웹 페이지를 불러오면서 캐시된 리소스를 검색하는데 소요한 평균 시간입니다.
- **DNS 탐색**: 브라우저가 웹 페이지를 불러오면서 웹사이트 도메인을 조회하는데 소요한 평균 시간입니다.
- **TCP 연결**: 브라우저가 웹 페이지를 불러올 때 TCP 핸드셰이크 과정에서 소요한 평균 시간입니다.
- **응답 대기**: 브라우저가 웹 페이지를 불러오면서 네트워크 요청을 보낸 후 서버로부터 첫 번째 바이트가 수신될 때까지 소요된 평균 시간입니다.
- **다운로드**: 브라우저가 웹 페이지를 불러오면서 서버로부터 리소스를 다운로드하는데 소요한 평균 시간입니다.
- **렌더링**: 서버로부터 다운로드한 리소스를 화면에 렌더링하고 페이지 로드 이벤트를 완료하는데 소요한 평균 시간입니다.
- **Load Time**: 브라우저가 웹 페이지를 완전히 불러오는데 소요한 평균 시간입니다.
- **Front-End**: 웹 페이지를 초기 렌더링하는데 소요한 평균 시간입니다.
- **Back-End**: 페이지 로드 요청부터 리소스를 다운로드하는데 소요한 평균 시간입니다.

백엔드 구간에서 최적화하기

- **캐시 설정 최적화**: 웹 페이지의 캐시 설정이 최적화되지 않거나 캐시 만료 시간이 너무 짧으면 캐싱 속도가 느려질 수 있습니다. 콘텐츠 유형별로 적절한 캐시 만료 시간을 설정하는 것이 중요합니다.
- **캐시 미스**: 캐시에 저장되지 않은 콘텐츠가 많을 경우 캐싱이 제대로 작동하지 않을 수 있습니다.
- **DNS 서버 성능 문제**: 사용 중인 DNS 서버의 성능이 떨어지거나 과부하 상태인 경우 DNS 조회 시간이 길어질 수 있습니다. 이를 해결하기 위해 더 빠르고 안정적인 DNS 서버로 변경하거나 여러 DNS 서버를 사용하여 부하를 분산할 수 있습니다. 또는 가능한 한 적은 수의 도메인을 사용하고 외부 리소스를 최소화하여 DNS 탐색 시간을 줄이는 것이 좋습니다.
- **원격 DNS 서버 위치**: DNS 서버가 사용자와 너무 멀리 떨어져 있으면 조회 시간이 늘어날 수 있습니다. 지리적으로 가까운 DNS 서버를 사용하거나 글로벌 캐시를 제공하는 DNS 서비스를 사용하여 이 문제를 해결할 수 있습니다.
- **서버 응답 시간**: 서버의 처리 속도가 느리거나 서버가 과부하 상태인 경우 응답 시간이 느려질 수 있습니다. 서버 성능을 최적화하거나 서버 용량을 증가시키는 조치 등이 필요할 수 있습니다.
- **서버 위치**: 서버와 사용자 간의 거리가 멀 경우 지연 시간이 길어질 수 있습니다. 이를 해결하기 위해 Content Delivery Network(CDN)과 같은 서비스를 사용하여 사용자와 가까운 위치에서 콘텐츠를 제공할 수 있도록 하세요.
- **네트워크 지연**: 인터넷 연결이 느리거나 불안정한 경우 백엔드 구간의 로딩 속도에 영향을 끼칠 수 있습니다. 이 문제는 사용자의 네트워크 상황에 따라 달라질 수 있습니다.
- **파일 크기**: 큰 파일을 다운로드할 때 속도가 느려질 수 있습니다. 이미지, 스크립트, 스타일 시트 등의 리소스를 최적화하고 압축하여 파일 크기를 줄이세요.

- **동시 다운로드 제한**: 브라우저는 동시에 다운로드할 수 있는 리소스의 수를 제한합니다. 동시에 다운로드되는 리소스 수를 줄이기 위해 여러 리소스를 하나로 병합하거나 비동기 로딩 기법을 사용할 수 있습니다.

프론트 엔드 구간에서 최적화하기

- **복잡한 DOM 구조**: 웹 페이지의 Document Object Model(DOM) 구조가 복잡한 경우 브라우저가 페이지를 렌더링하는 데 시간이 오래 걸릴 수 있습니다. DOM 구조를 간소화하고 불필요한 중첩 요소를 제거하여 렌더링 속도를 개선하세요.
- **무거운 CSS**: 많은 수의 CSS 규칙과 복잡한 선택자를 사용한다면 브라우저가 스타일 계산에 시간을 많이 소모할 수 있습니다. CSS 최적화 및 규칙 간소화, 불필요한 스타일 제거 작업을 통해 렌더링 속도를 개선하세요.
- **JavaScript 실행 지연**: 웹 페이지에 사용된 JavaScript 코드가 많거나 실행 시간이 오래 걸린다면 렌더링 속도에 영향을 줄 수 있습니다. JavaScript 코드를 최적화하고 실행 시간을 줄이며, 필요에 따라 비동기 로딩 방식을 사용하여 렌더링 속도를 개선하세요.
- **이미지 최적화**: 무거운 이미지 파일은 렌더링 속도를 느리게 만들 수 있습니다. 이미지를 압축하고 적절한 크기와 포맷으로 최적화하세요. 필요에 따라 레이지 로딩 기법을 사용하여 렌더링 속도를 개선하세요.
- **웹 폰트 사용**: 웹 폰트는 사용자에게 다양한 폰트 스타일을 제공할 수 있지만 로딩 시간에 영향을 줄 수 있습니다. 웹 폰트를 최적화하고 필요한 것만 사용하세요. 폰트 로딩 전에 텍스트를 표시하는 방식으로 렌더링 속도를 개선할 수도 있습니다.

리소스 목록 확인하기

브라우저가 서버로부터 다운로드하는 리소스는 페이지 로드 성능에 큰 영향을 줄 수 있습니다. **페이지 로드 상세** 창의 **리소스 목록**에서는 로딩 속도가 느리거나 파일 사이즈가 큰 리소스를 빠르게 파악할 수 있습니다.

각 리소스의 시작 시간을 기준으로 타임라인 차트를 제공해 로딩 속도가 느린 리소스를 파악할 수 있습니다. 각 리소스의 파일 크기를 같이 참조해 성능에 영향을 끼치는 부분을 개선하세요. 각 리소스를 선택하면 나타나는 **리소스 상세** 창을 통해 상세 시간 정보를 확인할 수 있습니다.

AJAX 모니터링 및 분석

Asynchronous JavaScript and XML (AJAX)는 비동기 방식으로 데이터를 교환하며 웹 페이지를 업데이트하는 기술입니다. AJAX는 사용자 경험을 개선하고 서버로의 요청을 줄이는 등의 장점을 가지고 있습니다. 그러나 AJAX는 브라우저에서 실행되는 JavaScript 코드로 구현되기 때문에 모니터링이 필요합니다. 다음과 같은 이유에서 AJAX 모니터링은 필요합니다.

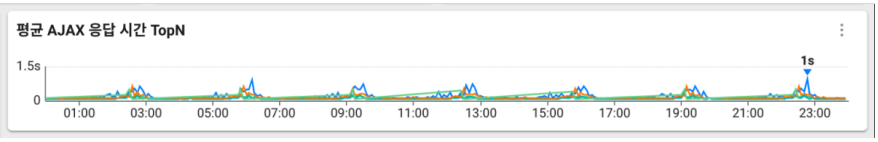
- AJAX 요청이 제대로 처리되는지 확인하기 위해
- AJAX 요청으로 인한 성능 저하를 파악하기 위해

- AJAX 요청이 보안상 문제를 유발할 가능성이 있는지 확인하기 위해
- AJAX 요청이 네트워크 대역폭을 낭비하고 있는지 확인하기 위해

브라우저 애플리케이션의 AJAX 요청 상태 파악하기

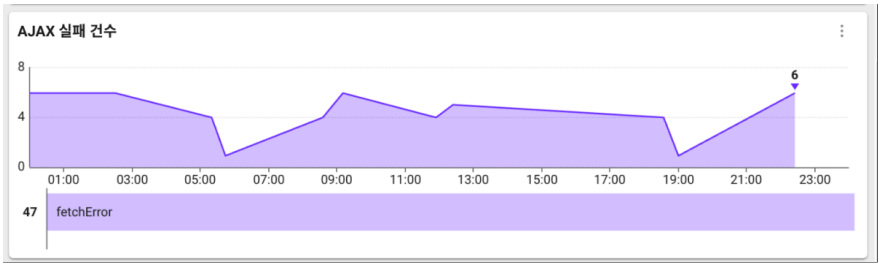
브라우저 모니터링 대시보드 메뉴의 AJAX 관련 위젯으로 브라우저 애플리케이션의 AJAX 요청 상태를 간단히 확인할 수 있습니다.

1. **AJAX 응답 시간 확인하기**



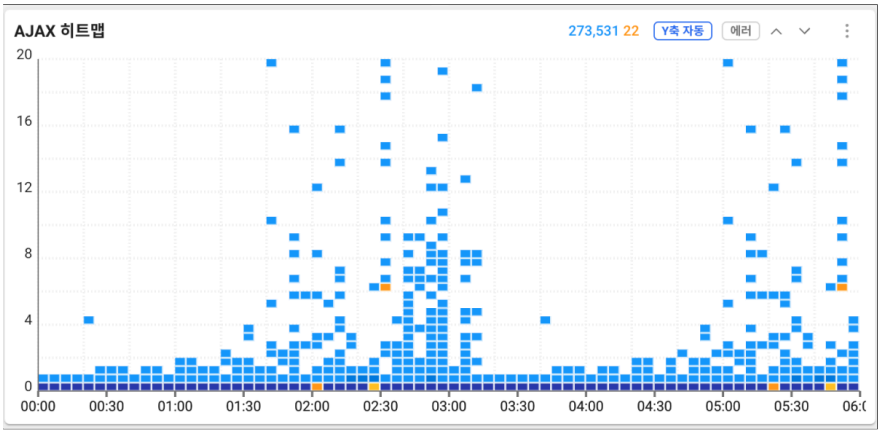
브라우저 애플리케이션에서 발생하는 AJAX의 평균 로드 시간을 확인할 수 있습니다. 로드가 오래 걸리는 호스트 또는 path를 파악하는데 용이합니다.

2. **AJAX 실패 건수 확인하기**



브라우저 애플리케이션에서 AJAX 요청이 정상적으로 이루어지지 않는 개수입니다. 브라우저 애플리케이션 전반에서 발생하고 있는 AJAX 실패 건수에 대해 실시간으로 확인할 수 있습니다.

3. **AJAX 히트맵 확인하기**

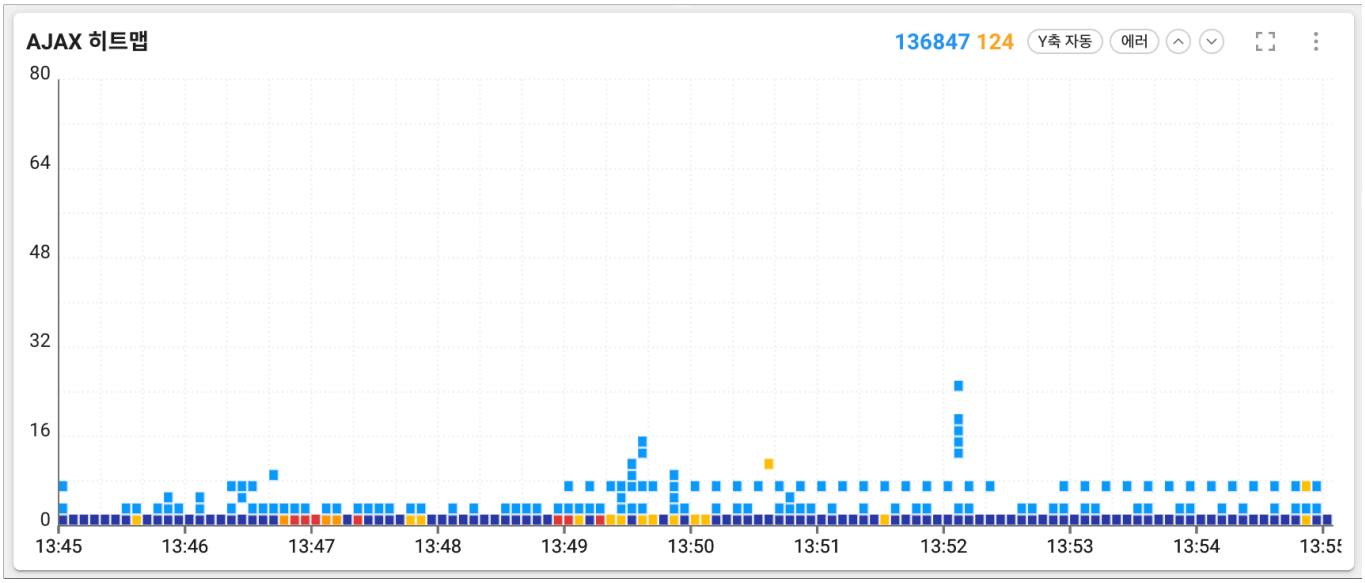


AJAX 요청을 히트맵 차트 형태로 제공합니다. 실시간으로 발생하는 AJAX 요청을 표시합니다. AJAX가 느릴수록 차트 상단에 위치하며 문제가 있는 AJAX에 대해선 황색 계열로 표시됩니다.

AJAX 히트맵으로 성능 저하 원인 파악 및 해결하기

AJAX 요청이 느리거나 문제가 있다면 각 AJAX가 어떤 상태인지 확인할 필요가 있습니다.

특정 AJAX 요청 에러

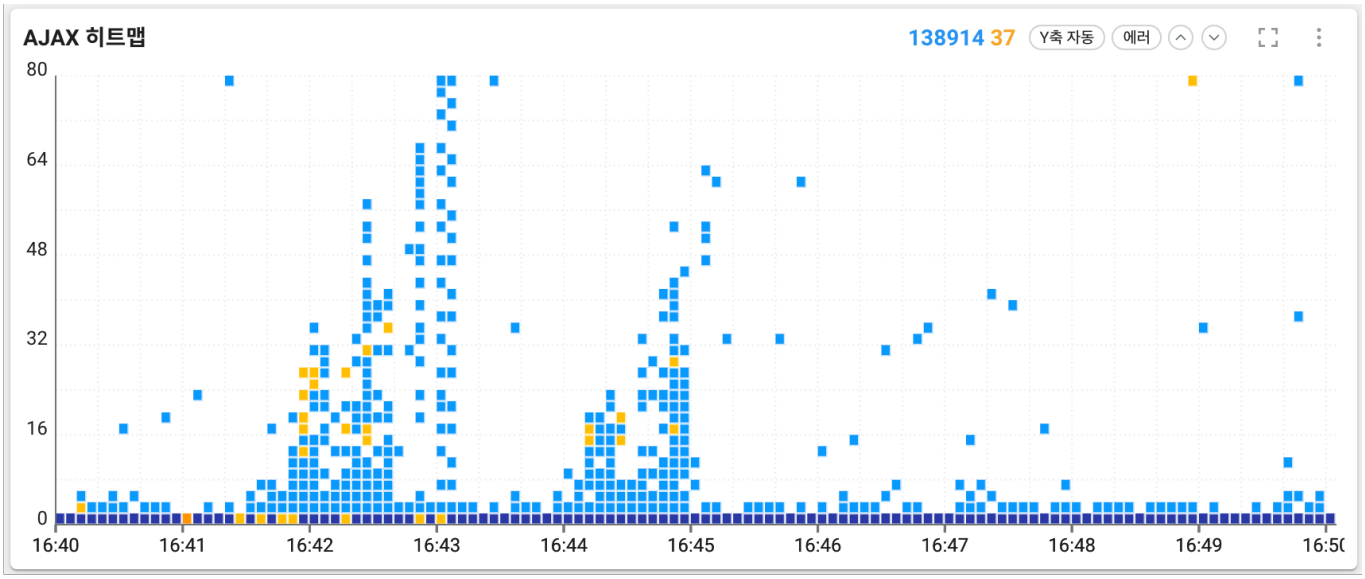


AJAX 요청 시 응답값이 400 이상이거나 요청 자체를 하지 못한 경우 황색 계열로 히트맵 차트에 표시됩니다. 이 색상이 나타나면

다음과 같은 상황이 발생한 것으로 추측할 수 있습니다.

- 서버에 문제가 발생하여 요청을 처리하지 못한 경우
- 자바스크립트 코드상 요청이 적절하지 않은 경우
- 인증 문제로 인해 요청이 실패한 경우
- 서버에서 응답을 제공하지 않은 경우

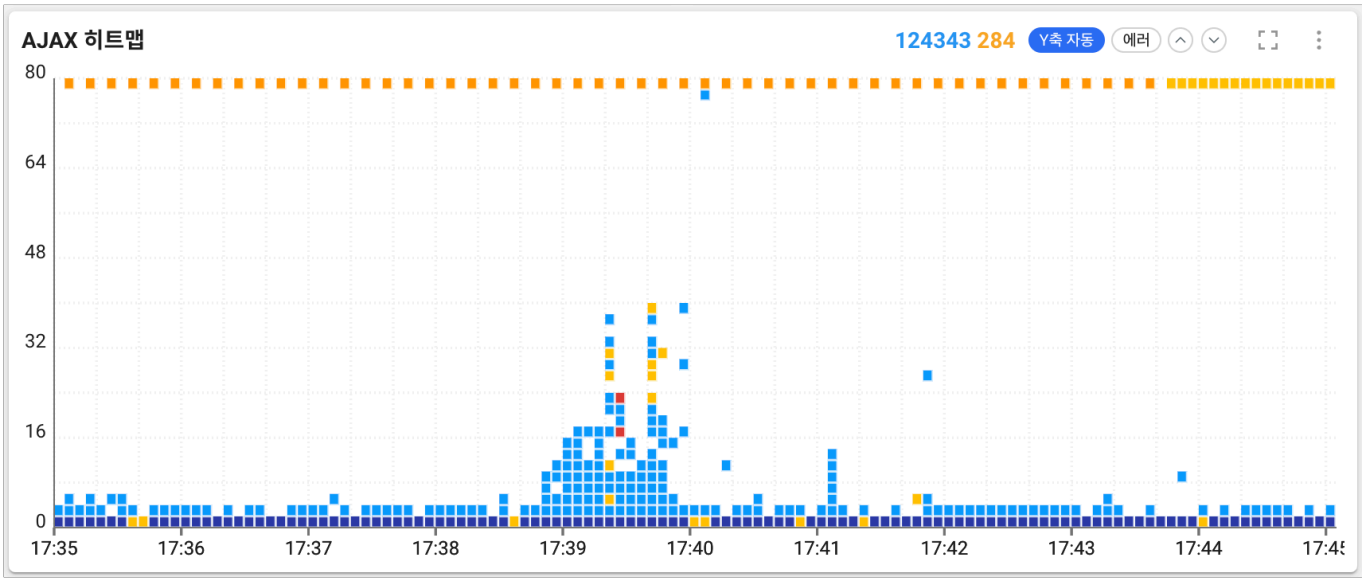
병목 현상



브라우저 애플리케이션 전반에서 AJAX 요청에 병목 현상이 발생한다면 다음과 같은 상황인지 확인하세요.

- **서버 성능 제한**: 서버 측에서 처리할 수 있는 요청의 수가 제한되거나 서버 자원이 부족한 경우 병목 현상이 발생할 수 있습니다.
- **동시 요청 수 제한**: 웹 브라우저는 동시에 처리할 수 있는 요청 수에 제한이 있습니다. 너무 많은 요청이 동시에 발생하면 병목 현상이 발생할 수 있습니다.
- **대량의 데이터 전송**: 전송되거나 받아야 하는 데이터의 크기가 매우 크다면 요청 처리에 시간이 오래 걸려 병목 현상이 발생할 수 있습니다.
- **자바스크립트 실행 성능**: AJAX 요청을 처리하는 자바스크립트 코드의 성능이 좋지 않거나 다른 스크립트와 충돌이 발생하는 경우 병목 현상이 발생할 수 있습니다.
- **응답 처리 지연**: 서버로부터 받은 응답을 처리하는 데 시간이 오래 걸리는 경우 병목 현상이 발생할 수 있습니다.
- **네트워크 대역폭 제한**: 인터넷 연결의 대역폭이 제한되어 있는 경우 동시에 처리할 수 있는 요청 수가 제한되어 병목 현상이 발생할 수 있습니다.

타임 아웃 현상

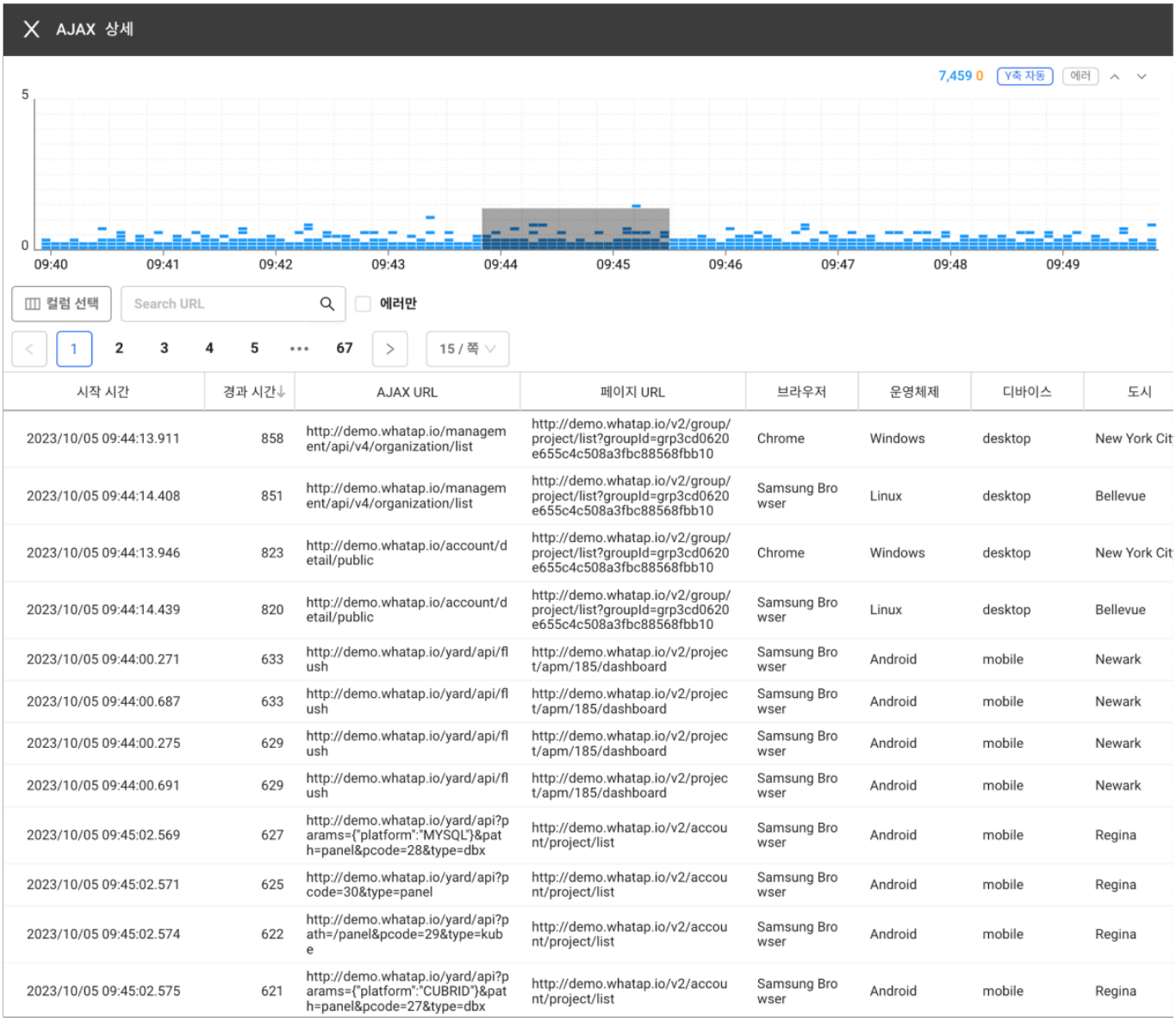


브라우저 애플리케이션 전반에서 AJAX 요청이 타임아웃 오류가 발생한다면 다음과 같은 상황인지 확인하세요.

- **서버 응답 지연**: 서버 측에서 처리하는 데 시간이 오래 걸리거나 서버가 다운된 경우 타임아웃이 발생할 수 있습니다.
- **네트워크 지연**: 인터넷 연결이 불안정하거나 지연이 발생하면 AJAX 요청이 제때 완료되지 않을 수 있습니다.
- **요청 처리 지연**: 브라우저에서 요청을 처리하는 데 시간이 오래 걸리는 경우 타임아웃이 발생할 수 있습니다.
- **대량의 데이터**: 전송되거나 받아야하는 데이터의 크기가 매우 크다면 요청 처리에 시간이 오래 걸려 타임아웃이 발생할 수 있습니다.
- **자바스크립트 실행 오류**: AJAX 요청을 처리하는 자바스크립트 코드에 문제가 있거나 다른 스크립트와 충돌이 발생하는 경우 타임아웃이 발생할 수 있습니다.
- **브라우저 호환성 문제**: AJAX 요청을 처리하는 데 문제가 있는 특정 브라우저에서만 오류가 발생할 수 있습니다.
- **요청 타임아웃 설정**: AJAX 요청의 타임아웃 값을 너무 낮게 설정한 경우 요청이 완료되기 전에 타임아웃이 발생할 수 있습니다.

AJAX 상세 정보 확인하기

AJAX로 인한 성능 저하 현상을 발견했다면 어떤 페이지에서 어떤 AJAX가 문제를 일으키고 있는지 확인해야 합니다. **AJAX 히트맵** 위젯에서 차트의 특정 영역을 드래그하면 선택한 영역의 AJAX에 대한 상세 정보를 확인할 수 있습니다.



시작 시간	경과 시간	AJAX URL	페이지 URL	브라우저	운영체제	디바이스	도시
2023/10/05 09:44:13.911	858	http://demo.whatap.io/management/api/v4/organization/list	http://demo.whatap.io/v2/group/project/list?groupId=grp3cd0620e655c4c508a3fbc88568fbb10	Chrome	Windows	desktop	New York Cit
2023/10/05 09:44:14.408	851	http://demo.whatap.io/management/api/v4/organization/list	http://demo.whatap.io/v2/group/project/list?groupId=grp3cd0620e655c4c508a3fbc88568fbb10	Samsung Browser	Linux	desktop	Bellevue
2023/10/05 09:44:13.946	823	http://demo.whatap.io/account/detail/public	http://demo.whatap.io/v2/group/project/list?groupId=grp3cd0620e655c4c508a3fbc88568fbb10	Chrome	Windows	desktop	New York Cit
2023/10/05 09:44:14.439	820	http://demo.whatap.io/account/detail/public	http://demo.whatap.io/v2/group/project/list?groupId=grp3cd0620e655c4c508a3fbc88568fbb10	Samsung Browser	Linux	desktop	Bellevue
2023/10/05 09:44:00.271	633	http://demo.whatap.io/yard/api/flush	http://demo.whatap.io/v2/project/apm/185/dashboard	Samsung Browser	Android	mobile	Newark
2023/10/05 09:44:00.687	633	http://demo.whatap.io/yard/api/flush	http://demo.whatap.io/v2/project/apm/185/dashboard	Samsung Browser	Android	mobile	Newark
2023/10/05 09:44:00.275	629	http://demo.whatap.io/yard/api/flush	http://demo.whatap.io/v2/project/apm/185/dashboard	Samsung Browser	Android	mobile	Newark
2023/10/05 09:44:00.691	629	http://demo.whatap.io/yard/api/flush	http://demo.whatap.io/v2/project/apm/185/dashboard	Samsung Browser	Android	mobile	Newark
2023/10/05 09:45:02.569	627	http://demo.whatap.io/yard/api?params={"platform":"MYSQL"}&path=panel&pcode=28&type=dbx	http://demo.whatap.io/v2/account/project/list	Samsung Browser	Android	mobile	Regina
2023/10/05 09:45:02.571	625	http://demo.whatap.io/yard/api?pcode=30&type=panel	http://demo.whatap.io/v2/account/project/list	Samsung Browser	Android	mobile	Regina
2023/10/05 09:45:02.574	622	http://demo.whatap.io/yard/api?path=/panel&pcode=29&type=kube	http://demo.whatap.io/v2/account/project/list	Samsung Browser	Android	mobile	Regina
2023/10/05 09:45:02.575	621	http://demo.whatap.io/yard/api?params={"platform":"CUBRID"}&path=panel&pcode=27&type=dbx	http://demo.whatap.io/v2/account/project/list	Samsung Browser	Android	mobile	Regina

문제가 발생한 AJAX의 경과시간, AJAX URL, 페이지 URL, 브라우저, 상태 코드, 에러 메시지 등을 확인할 수 있습니다. AJAX URL을 통해 어떤 AJAX 호출이 문제를 일으켰는지, 브라우저 정보를 통해 어떤 환경의 사용자가 문제를 겪고 있는지 파악할 수 있습니다. 상태 코드와 에러 메시지는 문제의 원인을 파악할 수 있는 중요한 정보입니다. 빠른 원인 파악은 문제 해결과 서비스 개선을 위한 지름길입니다.

사용자 접속 환경 분석

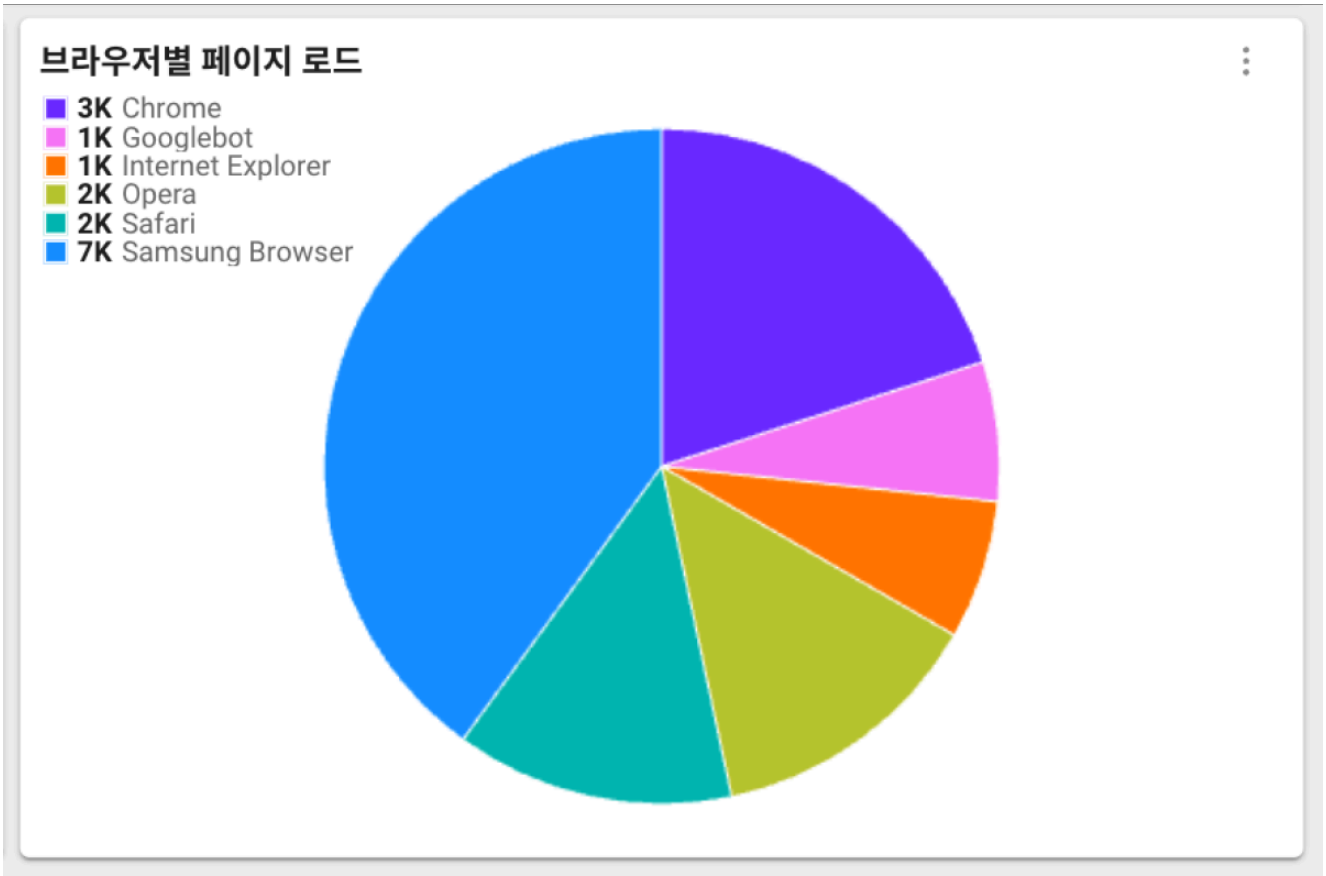
브라우저 애플리케이션을 이용하는 사용자의 환경은 다양합니다. 브라우저 애플리케이션을 모든 브라우저 및 접속 지역에 대응하여 개발하는 것은 불가능에 가깝습니다. 예상할 수 없는 문제를 모니터링하여 예방하고 최소화하는 기술적인 방법을 적용해야 합니다.

브라우저 애플리케이션을 사용자에게 안정적으로 제공할 수 있도록 아래와 같은 문제를 해결해야 합니다.

- **웹 브라우저 및 기기 호환성 문제 해결:** 사용자들이 다양한 브라우저 및 기기를 사용하기 때문에 각각의 브라우저나 기기에 따라 호환성 문제가 발생할 수 있습니다. 사용자들이 사용하는 브라우저 및 기기 정보를 파악하면 해당 브라우저나 기기에서 발생하는 문제를 더욱 쉽게 해결할 수 있습니다.
- **지역별 브라우저 애플리케이션 성능 차이 문제 해결:** 사용자들이 접속하는 지역에 따라 네트워크 속도가 다르기 때문에 브라우저 애플리케이션의 성능이 지연되는 경우가 발생할 수 있습니다. 이를 파악하면 해당 지역에서 발생하는 문제를 더욱 쉽게 해결할 수 있습니다.

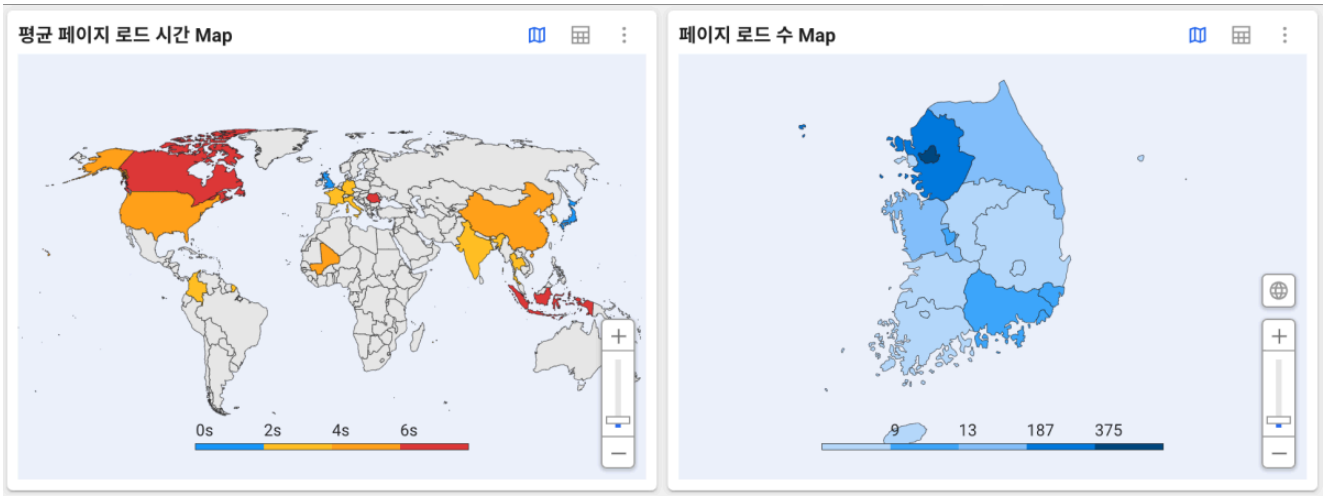
사용자의 접속 환경 파악하기

- **브라우저별 페이지 로드 수 확인하기**



브라우저 애플리케이션의 브라우저별 페이지 로드 수를 제공합니다. 이를 통해 브라우저별 페이지 로드 성능을 쉽게 비교할 수 있습니다.

- **지역 별 페이지 로드 시간 및 로드 수 확인하기**



브라우저 애플리케이션의 지역별 페이지 로드 시간과 페이지 로드 수를 지도 차트로 표현합니다. 사용자들은 지역별로 브라우저 애플리케이션의 성능을 쉽게 비교할 수 있습니다. 페이지 로드 시간이 느린 지역에서 웹사이트의 성능을 향상시키기 위한 방법을 찾을 수도 있습니다. 예를 들어 로드 시간이 느린 지역에서 서버를 추가하거나 캐시를 더욱 효율적으로 이용할 수 있는 방법을 찾을 수 있습니다. 이렇게 함으로써 사용자들은 더욱 빠르고 안정적인 브라우저 애플리케이션을 이용할 수 있습니다.

성능 저하 원인 파악하기

- 브라우저마다 같은 리소스 파일을 불러오는데 로딩 시간이 상이합니다. **페이지 로드 상세**에서 특정 브라우저에서 느려지는 원인을 파악하고 다음의 상황을 확인해 보세요.
  - **CSS 처리**: 브라우저마다 다른 CSS 처리 방식으로 렌더링 속도에 차이가 발생할 수 있습니다. 브라우저별로 지원하는 CSS 속성이나 기능에도 차이가 있으므로 이를 고려하여 웹 페이지를 구현해야 합니다.
  - **최적화 수준**: 브라우저별로 페이지 로딩을 최적화하는 기술과 정도에 차이가 있을 수 있습니다. 이미지 로딩, 코드 실행, 리소스 요청 순서 등 최적화 작업은 브라우저마다 다르게 작동할 수 있습니다.
  - **웹 페이지의 복잡성**: 웹 페이지의 복잡성과 구성 요소가 브라우저별로 다른 성능을 나타낼 수 있습니다. 복잡한 자바스크립트 코드나 CSS 애니메이션은 일부 브라우저에서 성능 저하를 일으킬 수 있습니다.
- 특정 브라우저에서만 에러가 발생한다면 **웹 API 및 기능 지원**을 확인해 보세요.

  브라우저마다 웹 API 및 기능 지원의 차이로 인해 일부 브라우저에서는 특정 기능이 느리게 동작하거나 전혀 작동하지 않을 수 있습니다. 이런 경우 폴리필(polyfill)을 사용해 해당 기능을 대체하거나 특정 브라우저에 대한 최적화를 고려해야 합니다.

에러 분석하기

다음과 같은 이유에서 브라우저 에러를 모니터링해야 합니다.

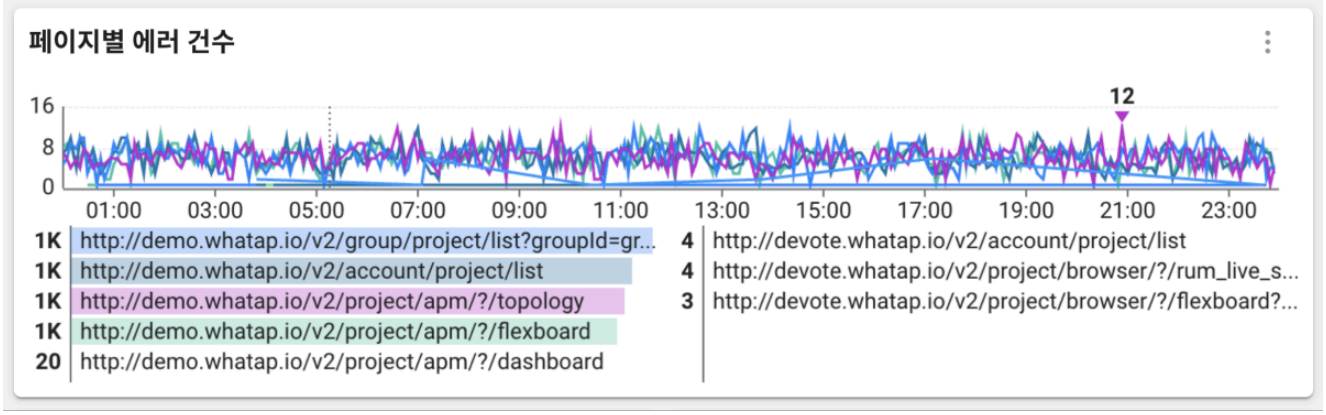
- **사용자 경험 개선**: 브라우저 에러가 발생하면 사용자가 애플리케이션을 제대로 사용하지 못할 수 있습니다. 따라서 브라우저 에러를 모니터링하고 가능한 빠르게 수정해야 합니다.
- **보안 취약점 방지**: 브라우저 에러는 웹사이트 보안에 큰 위협을 가할 수 있습니다. 에러를 모니터링하고 보안 취약점을 신속하게 수정해야 합니다.
- **성능 개선**: 브라우저 에러는 애플리케이션의 성능을 저하시킬 수 있습니다. 성능 저하는 서비스 품질과 사용자 만족도와 연관되어 있습니다. 성능 저하가 지속된다면 애플리케이션 사용자는 계속 줄어들 것입니다.

브라우저 에러 확인하기

대시보드 > **브라우저 에러** 메뉴에서는 브라우저 애플리케이션에서 발생하는 에러를 모니터링합니다. 다음과 같은 에러 유형을 파악할 수 있습니다.

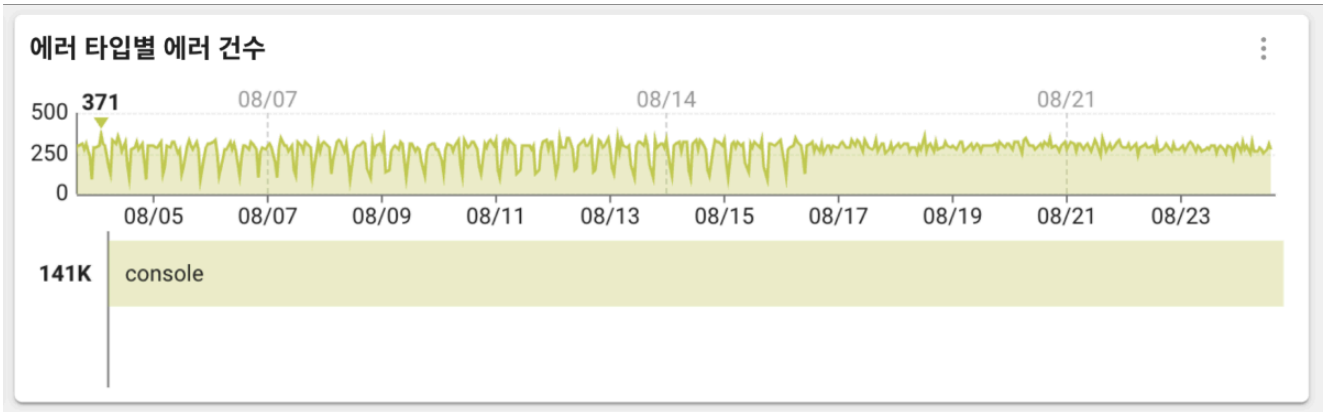
> ⓘ **브라우저 에러** 메뉴에 대한 자세한 내용은 다음 문서를 참조하세요.

- **페이지별 에러 건수 확인하기**



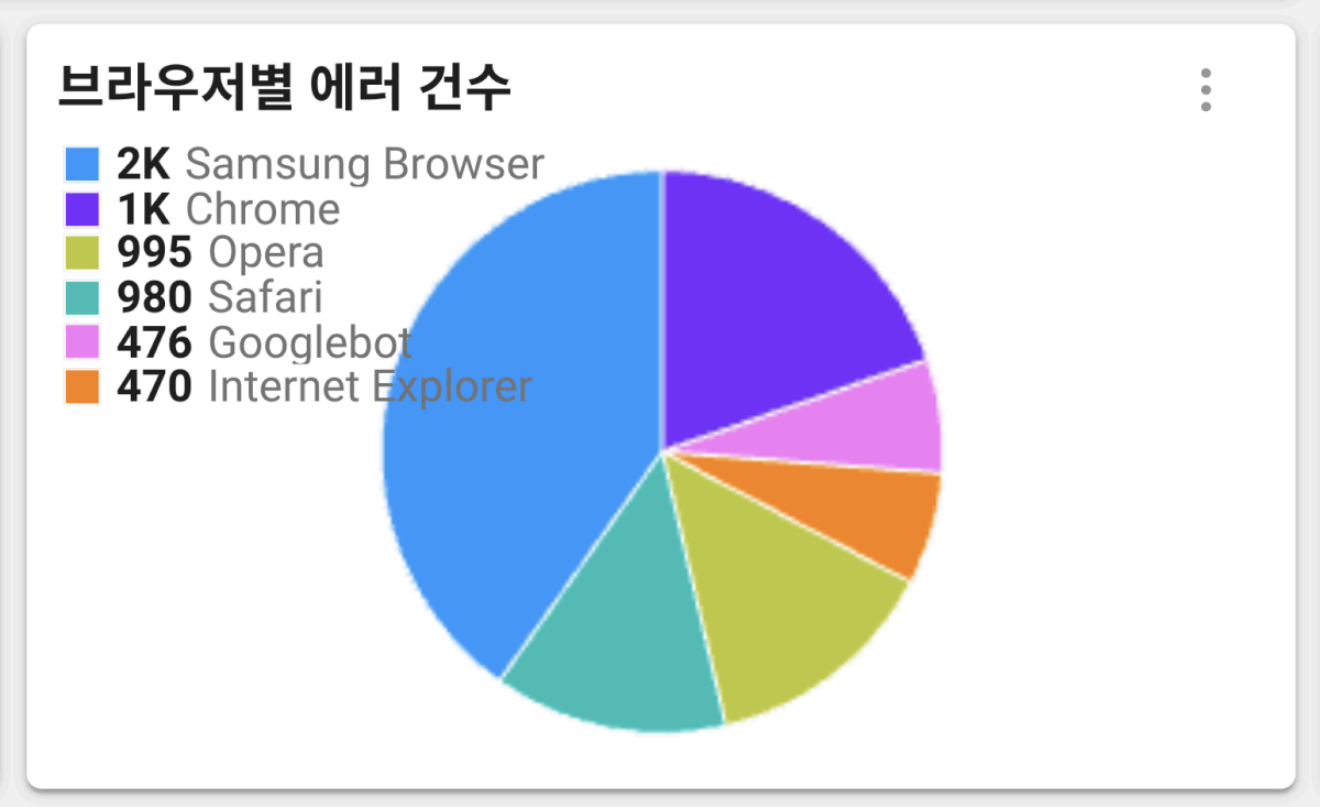
  페이지별로 발생하는 에러 건수를 실시간으로 확인할 수 있습니다. 문제가 발생하는 페이지를 빠르게 파악하고 대처할 수 있습니다.

- **에러 타입별 에러 건수 및 AJAX 에러 건수 확인하기**



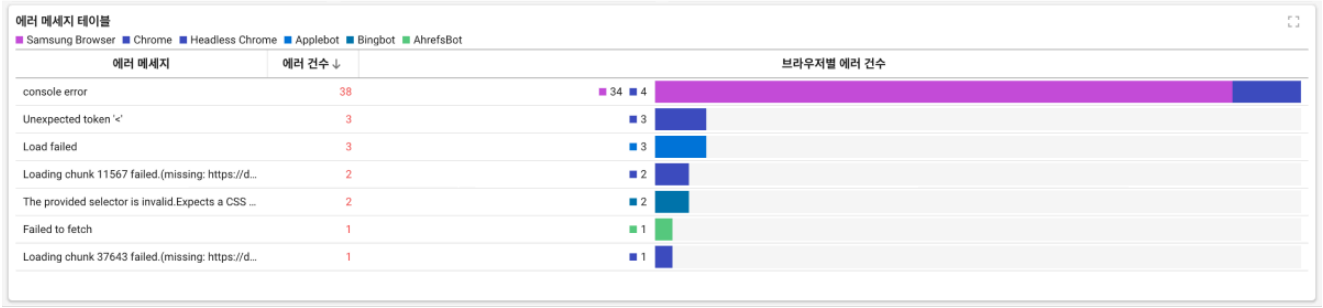
브라우저 애플리케이션 전반에서 발생하는 에러 건수에 대해서 에러 타입별로 확인할 수 있습니다.

- **브라우저별 에러 건수 확인하기**



브라우저별로 발생한 에러 건수를 파이 차트로 표시합니다. 브라우저가 어떤 에러를 자주 겪는지 쉽게 파악할 수 있습니다. 또한 특정 브라우저에서 발생하는 에러가 다른 브라우저에 비해 더 많다면 개발자는 해당 브라우저에 대한 최적화 작업을 진행할 수 있습니다.

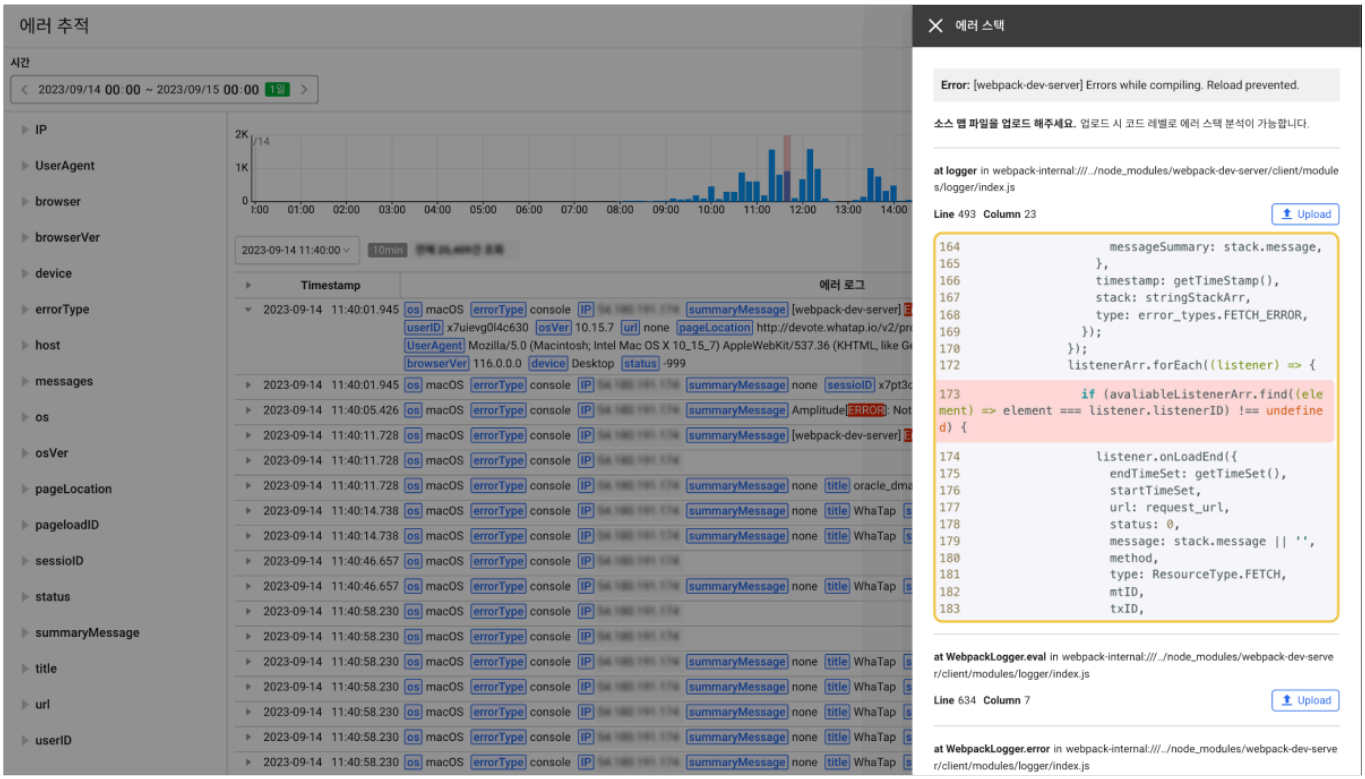
- **에러 메시지 테이블 확인하기**



에러 메시지를 기준으로 한 에러 수와 브라우저 별로 발생한 에러 수를 테이블 목록으로 표시합니다. 어떤 브라우저에서 어떤 에러가 가장 많이 발생하는지를 파악하여 문제점을 해결하는 데 이용할 수 있습니다. 에러 발생의 주요 원인이 무엇인 지를 파악하는 데에도 유용합니다. 새로운 에러가 발생하면 이 테이블을 참고하여 빠르게 대처할 수 있습니다.

에러 원인 파악하기

에러 메시지, 에러 발생 페이지를 찾았다면 **분석** > **에러 추적** 메뉴에서 에러에 대한 세부 정보를 확인하세요.



에러 추적 메뉴에서는 브라우저에서 발생한 에러 정보를 상세하게 확인할 수 있습니다. 에러 발생 페이지나 에러 메시지에 대해 필터를 적용해 검색할 수도 있습니다. 사용자가 발생시킨 에러는 사용자 환경 정보(브라우저, OS, 디바이스), 에러 스택 정보, 페이지 정보를 포함하고 있습니다.

에러 원인을 파악하려면 다음과 같은 단계로 진행할 것을 권장합니다.

1. 에러 메시지를 읽어보고 해당 페이지에서 어떤 작업을 수행했는지 확인하세요.
2. 해당 페이지에서 사용된 코드 및 라이브러리를 살펴봅니다.
3. 에러 스택 정보를 사용하여 어떤 함수나 라이브러리에서 에러가 발생했는지 확인하세요.
4. 브라우저, OS, 디바이스 정보를 사용하여 특정 환경에서만 발생하는 에러인지 확인하세요.
5. 페이지 정보를 사용하여 웹페이지의 특정 부분에서만 발생하는 에러인지 확인하세요.

에러 원인을 파악한 후에는 가능한 한 빠르게 수정하세요. 사용자 경험을 개선하고 브라우저 애플리케이션의 보안 취약점을 방지할 수 있습니다.

데이터베이스 실시간 감시

데이터베이스 성능 문제는 애플리케이션 속도 저하·서비스 중단으로 직결되고, 최종 사용자 경험과 매출에 즉각 영향을 줍니다. WhaTap Database Monitoring으로 **임계치를 벗어나는 운영 상태를 실시간 감시**하고, 문제가 생겼을 때 **원인까지 분석**하는 흐름을 안내합니다.

DB 감시 설계 시 점검할 5가지

- **목표 설정** — 쿼리 응답 시간 최대 허용치, 특정 작업 처리량 등 "무엇을 감시할지" 명확히
- **지표 선택** — CPU·메모리·디스크 I/O·네트워크·세션 현황·Wait Event 등
- **주기 설정** — DB 특성·부하에 맞춰 일정하게, 최신 상태가 수집되도록
- **도구 조합** — DBMS 내장 + 클라우드 관리형 + 3rd-party. 기능·확장성·편의성으로 선택
- **개선 검증** — 모니터링 결과로 문제 식별 → 해결 후 다시 모니터링으로 효과 확인

WhaTap DB 감시의 두 축

① 초 단위 가시화 (5초 간격)

DB가 제공하는 대부분의 Metric을 5초 간격으로 수집·시각화합니다.

- **온프레미스 DB + 클라우드 RDS 모두 동일한 화면**에서 감시 → DB 감시의 사각지대 제거
- 임계치 초과 시 **실시간 알림**으로 대응 촉구
- 여러 DB 인스턴스를 **한 프로젝트·한 대시보드**에서 통합 관찰

② 문제 원인 분석 도구

단순 Metric 감시를 넘어 "왜 느려졌는지"까지 파고드는 기능이 내장되어 있습니다.

표 | DB 문제 원인 분석 도구

기능	활용 시점
액티브 세션	지금 DB에 떠 있는 세션이 무엇을 기다리고 있나
Slow Query	느린 쿼리 목록·실행 계획·대기 이벤트
DB Lock 추적	Lock holder·waiter 관계, 교착 상황 파악
SQL 상세 조회	지연되는 개별 SQL의 실행 내역·호출 위치
SQL 통계	시간대별 쿼리 빈도·평균 응답 추이
DB 로그	에러·경고 로그 원본
Table·DB 사이즈 증감	테이블 크기 변화 추적(용량 계획)
파라미터 비교	운영·스테이지 DB 간 설정 차이 탐지

관찰 시나리오 예시

앱이 갑자기 느려졌을 때

1. 애플리케이션 **트랜잭션 트레이스**에서 DB 호출 구간이 느린지 확인하세요.
2. 해당 시간대의 **액티브 세션**에서 점유 세션과 대기 이벤트를 파악하세요.
3. **Slow Query**에서 느린 쿼리를 찾고, 실행 계획과 Wait Event를 확인하세요.
4. **DB Lock** 발생 여부를 교차 확인하세요.
5. 경고 알림 규칙과 연계하세요. 자세한 설정은 데이터베이스 경고 알림 설정을 참고하세요.

배포 후 쿼리 성능 회귀 감지

1. **SQL 통계**에서 배포 전후 주요 쿼리의 평균 응답 시간을 비교하세요.
2. 회귀 쿼리가 있으면 **파라미터 비교**로 환경 변화를 확인하세요.

3. 필요하면 릴리즈 검증 시나리오의 1주 추세 검증 흐름과 연결하세요.

용량 계획

1. **Table·DB 사이즈 증감**으로 월 단위 증가율을 추적하세요.
2. 주간·월간 성능 리포트에 포함하세요. 성능 리포팅 시나리오와 연계할 수 있습니다.

다음 단계

- **제품별 설치·연동** — PostgreSQL, Oracle, MySQL, SQL Server, Tibero
- **DB 경고 알림 설정** → 데이터베이스 경고 알림 설정
- **여러 제품을 함께 보기** → Flexboard로 제품 지표 조합하기
- **장애 시 원인 분석** → 장애 대응 시나리오

DB 메트릭스 경고 알림 설정

이 문서는 데이터베이스 프로젝트에서 메트릭스 경고 알림을 설정하는 방법을 안내합니다. 성능 관련 주요 지표를 기반으로 알림을 구성하면, 문제 상황을 빠르게 인지하고 대응할 수 있습니다.

프로세스(process) 경고 알림 설정하기

데이터베이스를 운영하다 보면 프로세스가 예기치 않게 종료되는 경우가 있습니다. 데이터베이스에서 실행 중인 프로세스가 종료될 경우 사용자에게 경고 알림을 보내 문제 상황을 빠르게 대처할 수 있습니다.

XOS 에이전트 설정

메트릭스 이벤트를 추가하기 전에 XOS 에이전트 설정(xos.conf)에 다음 옵션을 추가하세요. 다음 예제는 **top**과 **lock.sh** 프로세스를 감시 대상으로 설정하는 경우입니다. 쉼표(,)를 구분자로 이용해 여러 개의 프로세스를 등록할 수 있습니다.

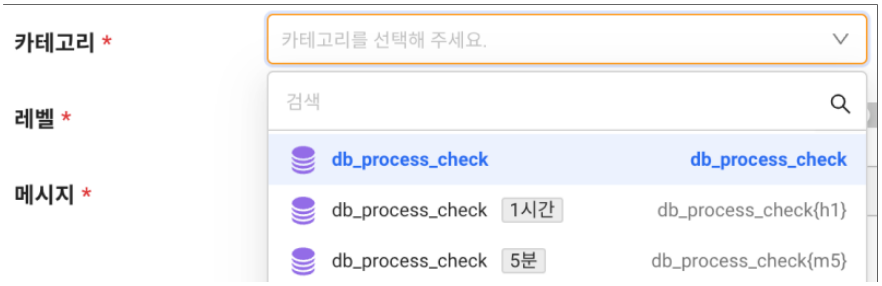
xos.conf

```
process=top,lock.sh
```

메트릭스 이벤트 추가

XOS 에이전트(xos.conf)에서 설정한 프로세스 감시 대상이 종료될 경우 경고 알림을 보내도록 설정하는 경우에 대한 예제입니다.

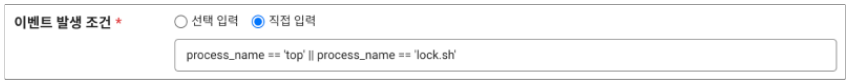
1. 사용자의 데이터베이스 프로젝트에서 **경고 알림** > **이벤트 설정** 메뉴로 이동하세요.
2. **메트릭스** 탭을 선택하세요.
3. **+ 이벤트 추가** 버튼을 클릭하세요.
4. **메트릭스 이벤트** 창이 나타나면 **이벤트명**을 입력하세요.
5. **카테고리**에서 **db_process_check** 항목을 선택하세요.



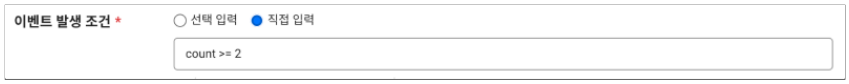
6. **레벨**에서 알림 수준을 선택하세요.
7. **메시지**에서 경고 알림 메시지를 입력하세요. 태그 또는 필드키 입력으로 메시지에 변수를 적용할 수 있습니다.

   (예, `${oname}` , `${count}` , `${process_name}`)
8. 다음 사례 중 원하는 값을 **이벤트 발생 조건**에 입력하세요.

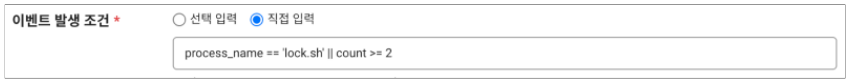
   - **top** 프로세스 또는 **lock.sh** 프로세스가 하나라도 종료될 경우:
     `process_name == 'top' || process_name == 'lock.sh'`

   - xos.conf 파일에서 설정한 프로세스 감시 대상 중 2개 이상이 종료될 경우: `count >= 2`

   - xos.conf 파일에서 설정한 프로세스 감시 대상 중 2개 이상이 종료되거나 lock.sh 프로세스가 종료될 경우:
     `process_name == 'lock.sh' || count >= 2`

9. 모든 입력을 완료했다면 **저장** 버튼을 클릭하세요.

- 이벤트 상태가 해소될 경우 알림을 원한다면 **이벤트 상태가 해소되면 추가 알림** 옵션을 선택하세요.
- 태그 또는 필드키는 **사이트 맵** > **메트릭스 조회** 메뉴에서 확인할 수 있습니다.
- 데이터베이스 프로젝트의 **메트릭스 경고 알림**에 대한 자세한 내용은 다음 문서를 참조하세요.

Flexboard로 제품 지표 조합하기

WhaTap 제품을 **여러 개 동시에 사용**하는 팀이라면, 제품별 기본 대시보드를 각각 들여다보는 것보다 **필요한 지표만 한 화면에 모으는 것**이 훨씬 빠릅니다. 이 가이드는 **Flexboard 템플릿**을 활용해 크로스 제품 대시보드를 구성하는 방법을 안내합니다.

> ⓘ
> - 단일 제품만 사용 중이라면 → 이미 **기본 대시보드**로 충분합니다. Flexboard는 불필요.
> - 여러 제품을 섞어 보고 싶다 → Flexboard의 진짜 역할. 이 가이드가 그 시점입니다.

> ⓘ
> - 템플릿 한 번 복제로 즉시 완성되는 팀 대시보드
> - 위젯을 만드는 게 아니라 **선택·교체**하는 수준의 작업
> - URL 하나로 팀 전체와 공유

사전 준비

- WhaTap 에이전트가 설치되어 **Active** 상태로 데이터가 수집되고 있음
- 프로젝트 **관리자** 또는 **편집자** 권한

1단계. 템플릿에서 시작하기

빈 Flexboard 대신 제공된 템플릿을 복제합니다.

1. 좌측 메뉴에서 **대시보드 > Flexboard** 메뉴를 선택하세요.
2. **템플릿** 탭(또는 **+ 템플릿에서 생성**)에서 목적에 맞는 템플릿을 선택하세요.
   - **Application 모니터링 기본** — TPS, 응답 시간, 에러율 등 앱 지표 중심
   - **서버 리소스 기본** — CPU, 메모리, 네트워크 등 인프라 지표 중심
   - **장애 대응 요약** — 이벤트·에러 집중 추적
3. **복제** 또는 **이 템플릿으로 시작** 버튼을 클릭하세요.

4. 이름을 입력하세요: `{팀명} 일일 상황판` (예: `Backend 일일 상황판`)

> • 애플리케이션 개발팀 → **Application 모니터링 기본**
> • SRE/인프라 → **서버 리소스 기본**
> • 장애 대응 담당팀 → **장애 대응 요약**
>
> 어떤 걸 고를지 모르겠으면 **Application 모니터링 기본**부터 복제하세요. 대부분의 팀에 잘 맞습니다.

2단계. 위젯 2~3개만 교체하기

복제된 대시보드는 이미 일반적인 지표로 구성되어 있습니다. **팀 맥락에 맞게 2~3개만 바꾸면 충분**합니다.

1. 팀에서 특별히 중요한 메트릭을 찾아 해당 위젯을 우클릭하거나 ⚙ 아이콘을 클릭한 뒤 **편집**을 선택하세요.
2. 메트릭을 교체하세요.
   - 예: "평균 응답 시간" → 팀이 SLA로 관리하는 특정 API의 응답 시간
   - 예: "전체 에러율" → 결제·로그인 등 비즈니스 핵심 트랜잭션의 에러율
3. 위젯 제목을 팀 언어로 변경하세요 (예: "결제 API 응답 시간").
4. **저장** 버튼을 클릭하세요.

> 처음부터 위젯 10개 이상을 맞추려 하면 "언젠가 채울 Flexboard"로 방치됩니다. 템플릿 상태 그대로 두고 **2~3개만 팀 맞춤**으로 바꾼 뒤 바로 공유하는 게 빠른 정착 비결입니다.

3단계. 팀과 공유하기

1. Flexboard 상단 **공유** 버튼을 클릭하세요.
2. 공개 범위를 선택하세요.
   - **팀 전체 공개**: 프로젝트 멤버 누구나 접근
   - **특정 사용자**: 초대한 사람만 접근

3. Flexboard URL을 복사해 팀 Slack 또는 협업 도구에 **고정 북마크**로 추가하세요.

> "모니터링 탭은 이 링크부터 열기"를 팀 규칙으로 정하면, 장애 감지 속도가 확연히 빨라집니다.

결과 확인

- **대시보드 > Flexboard** 목록에 새 Flexboard가 표시됨
- 템플릿에서 온 위젯들이 실시간 데이터로 갱신됨
- 공유 URL 접근 시 팀원도 동일한 화면을 봄

팀 전용 관측 포인트가 완성됐습니다. 복잡하게 느껴지던 Flexboard가 "템플릿 복제 → 약간만 커스텀"으로 실용 단계에 진입했습니다.

다음 단계

- **위젯별 임계값 알림 연결** → 첫 경고 알림 붙이기
- **위젯을 직접 설계**하고 싶어졌다면 → 2단계 시나리오 가이드의 Flexboard 심화 (준비 중)
- **대시보드를 주기 리포트로 추출** → 성능 리포팅 시나리오
- **여러 프로젝트를 한 Flexboard에 합치기** → 통합 Flexboard 구성 (심화)

OpenMetrics 탐색기

이 문서는 **OpenMetrics 탐색기** 메뉴에서 PromQL 쿼리로 메트릭을 조회하고 차트·테이블로 시각화하는 방법을 안내합니다.
처음 PromQL을 쓰는 사용자도 단계적으로 따라 할 수 있도록 기본 → 함수 → 실전 예제 순서로 정리했습니다.

주요 기능

- PromQL 쿼리를 통한 메트릭 조회
- Graph/Table 뷰로 데이터 시각화
- 시간 범위 선택 시 자동으로 PromQL에 시간 입력
- 실시간 메트릭 모니터링

화면 구성

- 상단: 시간 범위 선택기, 쿼리 입력창
- 중앙: Graph/Table/Stacked Bar 뷰 전환 버튼
- 하단: 조회 결과 표시 (그래프 또는 테이블)

왜 PromQL이 필요한가요?

OpenMetrics는 Prometheus 메트릭 포맷을 기반으로 다양한 시스템과 애플리케이션의 시계열 지표를 수집합니다.
Prometheus는 오픈소스 모니터링 도구로, CPU 사용률, 메모리 사용량, HTTP 요청 수, 에러율 등과 같은 수치 기반의 시간 흐름 데이터를 메트릭 지표로 저장합니다.

이러한 지표들은 대부분 수초 단위로 지속적으로 수집되며, 시간에 따라 끊임없이 변화하는 '시계열 데이터(time-series data)'의 형태를 가집니다.

하지만 단순히 데이터를 수집하는 것만으로 "지금 서비스가 정상인가?", "에러 비율이 평소보다 높아졌는가?", "어떤 API가 느려졌는가?" 같은 의미 있는 분석이 어렵습니다.

그래서 **OpenMetrics 탐색기**에서 수집된 메트릭을 분석할 수 있는 쿼리 언어가 필요하고, 그 역할을 하는 것이 바로

PromQL(Prometheus Query Language)입니다.

PromQL을 사용하면 다음과 같은 분석이 가능합니다.

- **실시간 모니터링**: 현재 시스템 상태를 즉시 확인 (현재 CPU 사용률, 활성 사용자 수 등)
- **추세 분석**: 시간에 따른 변화 패턴 파악 (최근 1시간 동안 초당 요청 수 증가율)
- **비교 분석**: 여러 서버나 서비스 간의 지표 비교 (서버별 에러율, 지역별 응답 시간)
- **집계 및 계산**: 복잡한 수학 연산과 통계 계산 (전체 에러율, 평균 응답 시간, 상위 10개 느린 API)
- **알림 조건 설정**: 특정 임계값 초과 시 자동 알림 (에러율 5% 이상, 메모리 사용률 90% 이상)

PromQL 사용자 가이드

쿼리 실행 방식

표 | PromQL 쿼리 실행 방식

쿼리	설명
Instant Query (즉시 쿼리)	특정 시점의 데이터를 조회함. Table 탭에서 사용됨
Range Query (범위 쿼리)	시작 시간과 종료 시간 사이의 데이터를 일정한 간격으로 조회함

데이터 타입

PromQL 표현식은 4가지 타입으로 평가됩니다.

표 | PromQL 데이터 타입

타입	설명
Instant Vector	각 시계열에서 단일 샘플을 포함하는 시계열 집합 (모두 동일한 타임스탬프)
Range Vector	각 시계열에서 시간 범위의 데이터 포인트를 포함하는 시계열 집합

타입	설명
Scalar	단순한 숫자 값
String	단순한 문자열 값

기본 사용법

1. 메트릭 선택(Instant Vector Selectors)

기본 메트릭 조회

```
nginx_http_response_count_total
```

레이블 필터링

```
nginx_http_response_count_total{status="200"}
```

여러 레이블 조건

```
nginx_http_response_count_total{method="GET", status="200"}
```

2. 레이블 매칭 연산자

표 | 레이블 매칭 연산자

연산자	설명
=	레이블 값이 정확히 일치
!=	레이블 값이 일치하지 않음
=~	레이블 값이 정규표현식과 일치
!~	레이블 값이 정규표현식과 일치하지 않음

정규표현식 예제

```
# staging, testing, development 환경만 조회
nginx_http_response_count_total{environment=~"staging|testing|development"}

# GET 메소드를 제외한 모든 메소드
nginx_http_response_count_total{method!="GET"}

# rep로 시작하는 replica 조회
nginx_http_response_count_total{replica=~"rep.*"}
```

3. 시간 범위 선택(Range Vector Selectors)

현재 시점부터 과거 특정 기간의 데이터를 선택합니다.

```
# 최근 5분간의 데이터
nginx_http_response_count_total[5m]

# 최근 1시간의 데이터
nginx_http_response_count_total{status="200"}[1h]
```

4. 시간 오프셋(Offset Modifier)

과거 특정 시점의 데이터를 조회합니다.

```
# 5분 전의 값
nginx_http_response_count_total offset 5m

# 1주일 전의 5분간 비율
rate(nginx_http_response_count_total[5m] offset 1w)

# 미래 비교 (음수 offset)
rate(nginx_http_response_count_total[5m] offset -1w)
```

실용 예제

예제 1: 상태 코드별 요청 수 조회

```
nginx_http_response_count_total{instance="http://192.168.100.122:4040/metrics"}
```

예제 2: 특정 상태 코드만 조회

```
# 200 응답만
nginx_http_response_count_total{status="200"}

# 4xx 에러만(정규표현식)
nginx_http_response_count_total{status=~"4.."}

# 5xx 에러만
nginx_http_response_count_total{status=~"5.."}
```

예제 3. 메소드와 상태 코드 조합

```
# GET 요청 중 200 응답
nginx_http_response_count_total{method="GET", status="200"}

# POST, PUT, DELETE 요청 중 404 응답
nginx_http_response_count_total{method=~"POST|PUT|DELETE", status="404"}
```

예제 4. 메트릭 이름으로 검색

```
# "job:"로 시작하는 모든 메트릭
{__name__=~"job:.*"}

# nginx로 시작하는 모든 메트릭
{__name__=~"nginx.*"}
```

예제 5. 최근 시간 범위 데이터

```
# 최근 5분간의 nginx 응답
nginx_http_response_count_total{status="200"}[5m]

# 최근 1시간의 404 에러
nginx_http_response_count_total{status="404"}[1h]
```

예제 6. 시간대 비교

```
# 현재 값과 1시간 전 값 비교
nginx_http_response_count_total{status="200"}
nginx_http_response_count_total{status="200"} offset 1h

# 어제 같은 시간의 데이터
nginx_http_response_count_total offset 1d
```

집계 함수

집계 함수(Aggregation Functions)는 여러 시계열 데이터를 하나로 결합하여 계산합니다.

sum()

모든 시계열의 값을 더합니다.

```
# 모든 인스턴스의 요청 수 합계
sum(nginx_http_response_count_total)

# 상태 코드별로 그룹화하여 합계
sum by (status) (nginx_http_response_count_total)

# 인스턴스별로 그룹화하여 합계
sum by (instance) (nginx_http_response_count_total)
```

avg()

모든 시계열의 **평균**값을 계산합니다.

```
# 모든 인스턴스의 평균 응답 수
avg(nginx_http_response_count_total)

# 상태 코드별 평균
avg by (status) (nginx_http_response_count_total)
```

count()

시계열의 **개수**를 셉니다.

```
# 전체 시계열 개수
count(nginx_http_response_count_total)

# 상태 코드별 시계열 개수
count by (status) (nginx_http_response_count_total)
```

max() / min()

시계열의 **최대값과 최소값**을 계산합니다.

```
# 최대값
max(nginx_http_response_count_total)

# 최소값
min(nginx_http_response_count_total)

# 상태 코드별 최대값
max by (status) (nginx_http_response_count_total)
```

topk() / bottomk()

지정한 수(K)만큼 **상위** 또는 **하위** 시계열을 반환합니다.

```
# 상위 5개 시계열
topk(5, nginx_http_response_count_total)

# 하위 3개 시계열
bottomk(3, nginx_http_response_count_total)

# 상태 코드별 상위 3개
topk(3, sum by (status) (nginx_http_response_count_total))
```

시계열 함수

rate()

가장 많이 사용되는 함수로, Counter 메트릭의 **초당 평균 증가율**을 계산합니다.

```
# 최근 5분간 초당 평균 요청 수
rate(nginx_http_response_count_total[5m])

# 상태 코드 200의 초당 평균 요청 수
rate(nginx_http_response_count_total{status="200"}[5m])

# 모든 인스턴스의 초당 총 요청 수
sum(rate(nginx_http_response_count_total[5m]))
```

주의사항

- Counter 메트릭에만 사용
- 시간 범위는 스크랩 간격의 최소 2배 이상 권장 (예: 스크랩 간격 30초 → [1m] 이상)
- 카운터 리셋(서버 재시작 등)을 자동으로 처리

increase()

지정한 시간 범위 동안의 **총 증가량**을 계산합니다.

```
# 최근 5분간 총 요청 수
increase(nginx_http_response_count_total[5m])

# 최근 1시간 동안의 에러 수
increase(nginx_http_response_count_total{status=~"5.."}[1h])
```

rate()와 increase()의 관계

`increase(v[5m]) = rate(v[5m]) × 300초`

- rate(): 초당 비율
- increase(): 절대 증가량

irate()

마지막 두 데이터 포인트를 기반으로 **초당 순간 증가율**을 계산합니다.

```
# 즉시 증가율
irate(nginx_http_response_count_total[5m])
```

irate() vs rate()

- irate(): 빠르게 변하는 카운터에 적합, 짧은 스파이크 감지
- rate(): 알림 및 느리게 변하는 카운터에 적합, 평균값 제공

시간 범위 함수

특정 시간 범위 내에서 값을 집계합니다.

avg_over_time()

시간 범위 평균 함수입니다.

```
# 최근 10분간 평균값
avg_over_time(nginx_http_response_count_total[10m])
```

max_over_time() / min_over_time()

지정한 시간의 **최대/최소값**을 계산하는 시간 범위 함수입니다.

```
# 최근 1시간 동안 최대값
max_over_time(nginx_http_response_count_total[1h])

# 최근 1시간 동안 최소값
min_over_time(nginx_http_response_count_total[1h])
```

sum_over_time()

지정한 기간의 값을 더한 **시간 범위 합계** 함수입니다.

```
# 최근 5분간 모든 값의 합
sum_over_time(nginx_http_response_count_total[5m])
```

중요 규칙: Rate then Sum

이 규칙은 Counter 메트릭에 rate(), increase(), irate() 같은 시계열 함수를 사용할 때 모든 집계 함수(sum, avg, max, min, count 등)에 적용됩니다.

> ⚠ **항상 rate() 먼저, 그 다음 sum()**
>
> Counter 메트릭을 집계할 때 반드시 이 순서를 지켜야 합니다.

🔵 올바른 방법

```
# rate()를 먼저 적용한 후 sum()
sum(rate(nginx_http_response_count_total[5m]))
```

❌ 잘못된 방법

```
# sum()을 먼저 하면 카운터 리셋 시 문제 발생
rate(sum(nginx_http_response_count_total[5m]))
```

Counter 메트릭을 sum()으로 먼저 합치면, 개별 서버의 카운터 리셋(재시작)을 감지할 수 없게 됩니다. 서버 한 대가 재시작되면 전체 합계가 갑자기 감소하고, rate()는 이를 음수 증가율이나 비정상적인 스파이크로 잘못 계산합니다. 반면 rate()를 먼저 적용하면 각 서버별로 카운터 리셋을 정확히 감지하고 처리할 수 있습니다.

실전 예제

예제 1. 초당 요청 수 (RPS) 계산

```
# 전체 RPS
sum(rate(nginx_http_response_count_total[5m]))

# 상태 코드별 RPS
sum by (status) (rate(nginx_http_response_count_total[5m]))

# 메소드별 RPS
sum by (method) (rate(nginx_http_response_count_total[5m]))
```

> ⓘ **전체 Requests Per Second(RPS) PromQL에 대한 설명**
>
> sum(rate(nginx_http_response_count_total[5m]))이 왜 RPS일까?
>
> 1. nginx_http_response_count_total은??
>   - Counter 메트릭: 서버 시작 이후 누적 응답 수
>   - 예: 1000 → 1050 → 1120 → 1180 → 1250...
> 2. rate(...[5m])은 무엇일까?
>   - 최근 5분간 데이터를 보고 초당 평균 증가율 계산
>   - 예시: 5분 전: 1000 현재: 1300 차이: 300개 (5분 = 300초) rate = 300 / 300초 = 1.0 requests/second
>
>   → rate()의 결과 = 초당 요청 수 (RPS)
> 3. sum()은 왜 필요한가?
>   - 여러 서버/레이블의 rate 값을 모두 더함
>   - 예: server1, status=200 → 10 req/s server1, status=404 → 2 req/s server2, status=200 → 15 req/s server2, status=404 → 3 req/s 합계 = 30 req/s
>
> 결론: sum(rate(nginx_http_response_count_total[5m])) = 모든 서버의 초당 총 요청 수 (RPS)
>
> 참고. 5분간 총 요청 수를 원하면 increase() 사용 sum(increase(nginx_http_response_count_total[5m])) = 5분간 총 요청 수

예제 2. 에러율 계산

```
# 전체 요청 대비 5xx 에러 비율 (%)
sum(rate(nginx_http_response_count_total{status=~"5.."}[5m])) / sum(rate(nginx_http_response_count_total[5m])) * 100

# 전체 요청 대비 4xx 에러 비율
sum(rate(nginx_http_response_count_total{status=~"4.."}[5m])) / sum(rate(nginx_http_response_count_total[5m]))
```

예제 3. 상위 N개 인스턴스 찾기

```
# 요청이 가장 많은 상위 5개 인스턴스
topk(5, sum by (instance) (rate(nginx_http_response_count_total[5m])))
```

```
# 에러가 가장 많은 상위 3개 인스턴스
topk(3, sum by (instance) (rate(nginx_http_response_count_total{status=~"5.."}[5m])))
```

예제 4. 시간대별 비교

```
# 현재 RPS
sum(rate(nginx_http_response_count_total[5m]))

# 1시간 전 RPS
sum(rate(nginx_http_response_count_total[5m] offset 1h))

# 어제 같은 시간 RPS
sum(rate(nginx_http_response_count_total[5m] offset 1d))
```

예제 5. 평균 응답 수

```
# 인스턴스별 평균 응답 수
avg by (instance) (rate(nginx_http_response_count_total[5m]))

# 상태 코드별 평균
avg by (status) (rate(nginx_http_response_count_total[5m]))
```

수학 연산자

산술 연산

```
# 더하기
sum(rate(nginx_http_response_count_total{status="200"}[5m]))
+
sum(rate(nginx_http_response_count_total{status="201"}[5m]))

# 빼기
sum(rate(nginx_http_response_count_total[5m])) - sum(rate(nginx_http_response_count_total{status=~"5.."}[5m]))

# 곱하기 (바이트를 MB로 변환)
nginx_http_response_size_bytes / 1024 / 1024
```

```
# 나누기 (비율 계산)
sum(rate(nginx_http_response_count_total{status=~"5.."}[5m])) / sum(rate(nginx_http_response_count_total[5m]))
```

비교 연산

```
# 100보다 큰 값만
nginx_http_response_count_total > 100

# 특정 값과 같은 경우
nginx_http_response_count_total == 200

# 범위 지정
nginx_http_response_count_total > 100 and nginx_http_response_count_total < 1000
```

실전 팁

1. 스크랩(수집 주기)별 시간 범위 선택

표 | 스크랩 간격별 권장 시간 범위

스크랩 간격	최소 시간 범위	권장 시간 범위	이유
15초	[30s]	[1m] 이상	2~4개 데이터 포인트 확보
30초	[1m]	[2m] 이상	2~4개 데이터 포인트 확보
1분	[2m]	[5m] 이상	2~4개 데이터 포인트 확보

2. 그룹화 레이블 선택

```
# 너무 세분화 (너무 많은 시계열)
sum by (instance, method, status, path) (rate(nginx_http_response_count_total[5m]))

# 적절한 그룹화
sum by (status) (rate(nginx_http_response_count_total[5m]))
```

3. 알림 규칙 작성 시

```
# 5분간 에러율이 5% 이상인 경우
sum(rate(nginx_http_response_count_total{status=~"5.."}[5m])) / sum(rate(nginx_http_response_count_total[5m]))
> 0.05
```

4. 함수 조합 순서

```
# 1. rate() 먼저
# 2. 집계 함수 (sum, avg 등)
# 3. 수학 연산

# 올바른 순서
sum(rate(nginx_http_response_count_total[5m])) * 100
```

기타 함수

abs()

값의 **절대값**을 반환하는 함수입니다.

```
abs(rate(nginx_http_response_count_total[5m]) - 100)
```

round()

값을 가장 가까운 정수 또는 지정한 단위로 **반올림**합니다.

```
# 소수점 반올림
round(sum(rate(nginx_http_response_count_total[5m])))

# 10 단위로 반올림
round(sum(rate(nginx_http_response_count_total[5m])), 10)
```

clamp_max() / clamp_min()

값이 지정한 **최대값이나 최소값을 넘지 않도록 제한**합니다.

```
# 최대값 1000으로 제한
clamp_max(nginx_http_response_count_total, 1000)

# 최소값 0으로 제한
clamp_min(nginx_http_response_count_total, 0)
```

주의사항

빈 레이블 값 매칭

빈 레이블 값을 매칭하면 해당 레이블이 설정되지 않은 모든 시계열도 선택됩니다.

```
# environment 레이블이 없거나 빈 값인 모든 시계열
nginx_http_response_count_total{environment=""}
```

필수 선택자

벡터 선택자는 반드시 메트릭 이름이나 빈 문자열과 일치하지 않는 레이블 매처를 하나 이상 지정해야 합니다.

```
# ❌ 잘못된 예
{job=~".*"}

# ✅ 올바른 예
{job=~".+"}
{job=~".*", method="get"}
```

성능 고려사항

효율적인 쿼리를 위한 팁

메트릭 이름만 사용하면 수천 개의 시계열이 선택될 수 있습니다.

- 항상 적절한 레이블 필터를 사용하여 결과를 제한하세요.
- 처음에는 Table 뷰에서 결과를 확인하고, 적절한 수준으로 필터링한 후 차트 뷰로 전환하세요.
- 수백 개 이하의 시계열이 이상적입니다.

Staleness (오래된 데이터)

시계열이 더 이상 수집되지 않으면 "stale"로 표시됩니다.

- Stale로 표시된 후에는 쿼리 결과에 포함되지 않습니다.
- 기본적으로 5분 동안 데이터가 수집되지 않으면 stale로 간주됩니다.

정규표현식

Prometheus는 RE2 문법을 사용합니다.

- 모든 정규표현식은 완전히 앵커링됩니다. (자동으로 `^`와 `$`가 추가됨)
- `env=~"foo"`는 `env=~"^foo$"`와 동일합니다.

주석

```
# 이것은 주석입니다
nginx_http_response_count_total{status="200"}  # 라인 끝 주석도 가능
```

추천 학습 순서

- 기초: rate(), sum(), avg()
- 중급: increase(), irate(), topk()
- 고급: 복잡한 비율 계산, 여러 함수 조합

참고 자료

- PromQL 기본 문서
- PromQL 함수 문서
- PromQL 사용 예제

MXQL

> MXQL을 알아보기 전에 메트릭스의 개념에 대해 먼저 알아보길 권장합니다. 메트릭스에 대한 자세한 내용은 다음 문서를 참조하세요.

MXQL이란?

MXQL은 와탭의 성능 데이터(메트릭스)를 유연하게 조회하기 위한 쿼리 언어입니다. 하나의 프로젝트에 포함된 여러 에이전트에서 수집한 메트릭스들을 종합적으로 조회하고 활용하기 위해서 사용합니다.

MXQL과 SQL의 차이

많이 알려진 SQL과 비교를 통해 MXQL의 개념을 알아봅니다.

용어

우선 SQL에서 사용하는 용어를 살펴봅니다.

SQL의 데이터 저장 구조

위와 같이 whatap의 데이터베이스(Database)에 product 테이블(Table)이 포함되어 있습니다. product 테이블(Table)은 id, description, 두 개의 컬럼(Column)이 포함합니다. SQL의 Database, Table, Column에 대응하는 MXQL의 용어는 각각 database, category, field입니다.

저장방식	MXQL	SQL
대분류	Database	Database
중분류	Category	Table
소분류	Field	Column

쿼리(Query)

MXQL과 SQL의 샘플 쿼리입니다. 각 라인의 오른쪽에 주석 내용을 참조하세요.

SQL query

```
SELECT time, pcode  -- Column 선택(time, pcode 컬럼만 조회하도록 설정합니다.)
FROM app_counter    -- Table 선택(app_counter 테이블에서 데이터를 조회합니다.)
WHERE tx_count = 1  -- 데이터 필터링(tx_count column의 값이 1인 데이터만 조회하도록 설정합니다.)
```

MXQL query

```
CATEGORY app_counter              -- app_counter 카테고리에서 데이터를 조회하도록 설정합니다.
TAGLOAD                           -- 데이터를 조회합니다.
SELECT [ time, pcode ]            -- 조회된 전체 컬럼 중에서 time, pcode 필드만 선택합니다.
FILTER { key : tx_count, value : 5}  -- tx_count 필드의 값이 5인 데이터만 남깁니다.
```

실행 결과

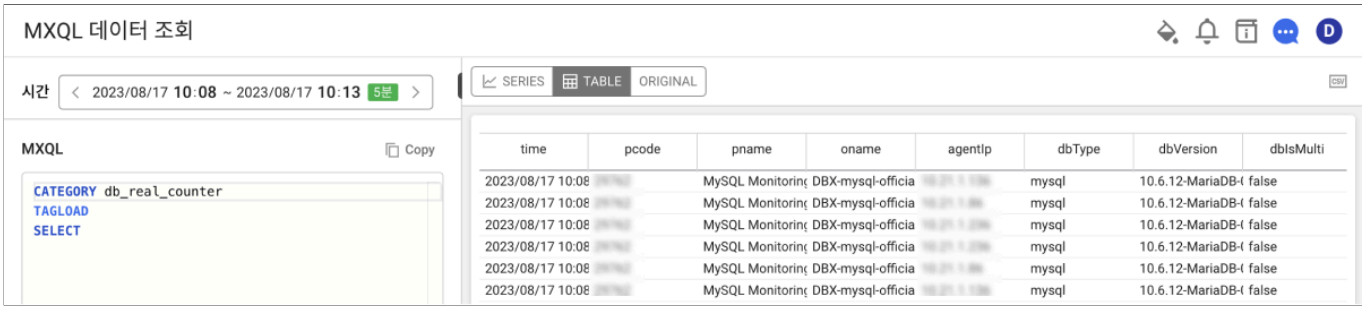
MXQL 쿼리를 수행하면 선택한 카테고리에서 선택한 필드의 메트릭스를 조회합니다.

`app_counter` 카테고리에서 `tx_count`, `tx_error` 지표를 조회하는 쿼리는 다음과 같습니다.

MXQL

```
CATEGORY app_counter              -- app_counter 카테고리에서 데이터를 조회하도록 설정합니다.
TAGLOAD                           -- 데이터를 조회합니다.
SELECT [time, oid, tx_count, tx_error] -- 조회하고 싶은 필드의 이름을 설정합니다.
```

쿼리를 수행하면 다음과 같이 메트릭스를 조회합니다.



메트릭스에는 항상 `time` , `oid` 값을 포함하기 때문에 MXQL 쿼리에서도 `time` , `oid` 필드를 항상 포함해 조회할 것을 권장합니다. 최종 조회한 데이터가 언제(`time`) 어떤 에이전트(`oid`)에서 수집한 메트릭스인지 확인할 수 있습니다.

MXQL 문법 가이드

형식

MXQL은 각 라인마다 **명령어**와 **오퍼랜드**로 구성되며 띄어쓰기로 구분합니다.

```
<명령어> <오퍼랜드>
```

`명령어` 는 한 단어의 예약어입니다. `명령어` 는 대문자로 입력하며 `오퍼랜드` 는 소문자로 입력합니다. `명령어` 마다 입력 가능한 `오퍼랜드` 의 형식은 정해져 있습니다. `오퍼랜드` 에는 4가지 타입의 값이 올 수 있습니다.

1. 오퍼랜드가 없는 경우

   ```
   TAGLOAD
   ```

2. 문자열(숫자 혹은 단어)

   ```
   CATEGORY app_counter
   ```

3. 문자열 배열

   ```
   SELECT [ time, pcode ]
   ```

4. JSON 문자열 타입

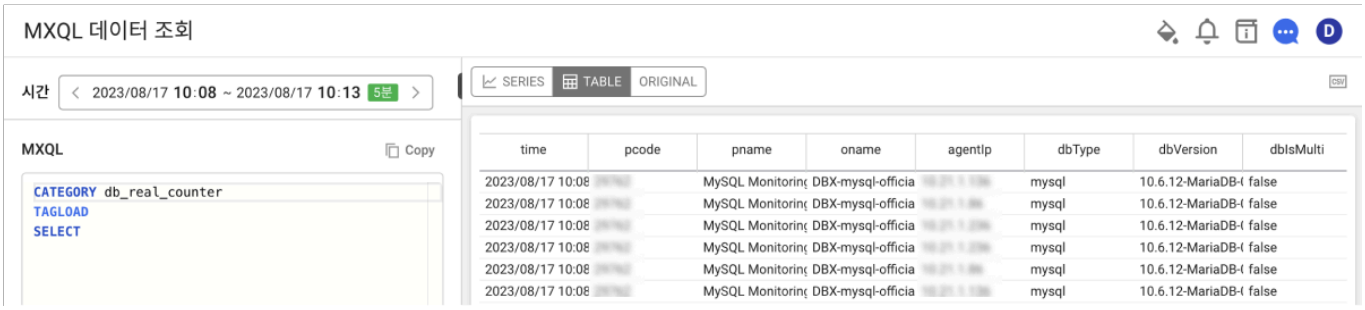
   ```
   FILTER { key : tx_count, value : 5}
   ```

Sample MXQL query

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, pcode ]
FILTER { key : tx_count, value : 5}
```

테스트 환경

홈 화면 > 프로젝트 선택 > **사이트맵** > **실험실** > **MXQL 데이터 조회** 메뉴에서 MXQL 쿼리를 테스트할 수 있습니다.



> ⓘ 메트릭스에는 태그와 필드가 구분되어 있지만 **MXQL 데이터 조회** 메뉴에서는 태그와 필드의 구분 없이 표현합니다.

단계별 구성

MXQL은 단계별 구성을 가지고 있습니다. 각 단계별로 사용할 수 있는 **명령어**의 종류가 정해져 있으며 각 단계의 이름과 특징은 다음과 같습니다.

1. **메트릭스 선택**: 어떤 에이전트에서 수집한 어떤 메트릭스를 사용할지 선택하세요.
2. **메트릭스 로딩**: 이전 단계에서 설정한 값들을 사용해서 메트릭스를 불러옵니다. 대부분의 경우 `TAGLOAD` 를 이용하며, 특수한 경우에만 `FLEXLOAD` 를 이용하세요. `FLEXLOAD` 를 사용해야 하는 경우는 다음 문서를 참조하세요.
3. **메트릭스 가공**: 이전 단계에서 불러온 메트릭스에 대해서 단계별로 가공을 수행합니다.

Example

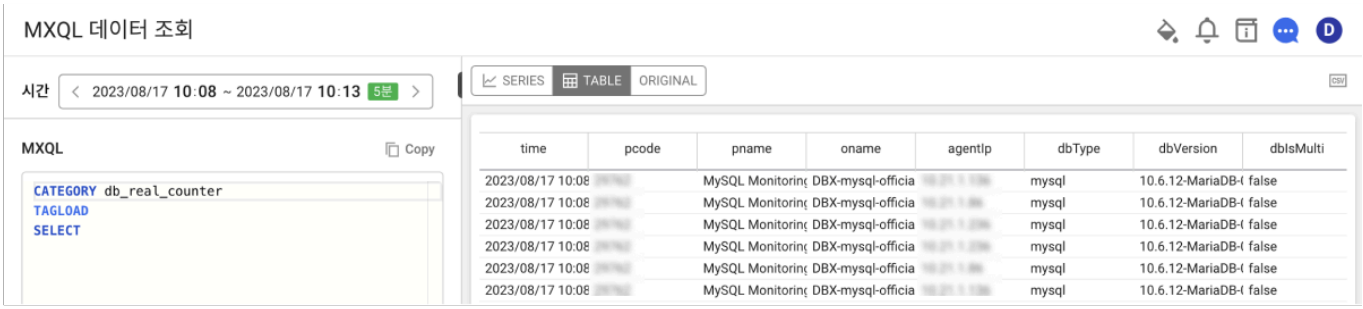
```
# 메트릭스 선택 단계
CATEGORY app_counter -- 카테고리 선택

# 메트릭스 로딩 단계
TAGLOAD -- 데이터 1000개 조회

# 메트릭스 가공 단계
```

```
SELECT [time, oid, active_tx_count, tx_count, tx_error] -- 1000개 데이터의 5개 필드만 다음 단계로 전달
FILTER {expr : "tx_count > 40"}       -- 데이터 1000개 중 100개만 통과
FILTER {expr : "active_tx_count > 10"} -- 데이터 100개 중 10개만 통과
FILTER {expr : "tx_error < 3"}        -- 데이터 10개 중 3개만 통과
```

다음은 메트릭스 가공 단계를 모두 통과한 메트릭스 예시입니다.



주석

"#" 또는 "--"으로 시작하는 문장은 무시됩니다.

Example

```
# 데이터 조회 설정
CATEGORY app_counter

# 데이터 조회
TAGLOAD

# 데이터 가공
SELECT [ time, pcode ]
FILTER { key : tx_count, value : 5}
```

MetricValue(복합값)

MetricValue(복합값)은 메트릭스 가공 단계에서 자주 사용하는 연산을 편리하게 지원하는 MXQL의 자료 구조입니다. 메트릭스 가공 단계의 GROUP, UPDATE 명령어는 메트릭스가 MetricValue 형태로 저장되어 있을 때만 사용할 수 있습니다.

예를 들어 다음과 같은 데이터가 있다고 가정해보겠습니다.

time	tx_count
2021/06/24 13:40:00	1
2021/06/24 13:40:10	2
2021/06/24 13:40:20	3
2021/06/24 13:40:30	4
2021/06/24 13:40:40	5
2021/06/24 13:40:50	6

위 데이터를 30초 간격으로 GROUP의 merge 옵션을 진행하면 다음과 같은 형태로 데이터를 변형할 수 있습니다.

time	tx_count
2021/06/24 13:40:00 ~ 2021/06/24 13:40:20	1,2,3에 대한 MetricValue
2021/06/24 13:40:30 ~ 2021/06/24 13:40:50	4,5,6에 대한 MetricValue

데이터를 MetricValue로 변환하면 총 6가지 옵션을 사용할 수 있습니다.

옵션	기능
sum	MetricValue에 포함된 값을 더합니다.
min	MetricValue에 포함된 값 중 최소값을 구합니다.
max	MetricValue에 포함된 값 중 최대값을 구합니다.
last	MetricValue에 포함된 값 중 마지막에 추가된 값을 구합니다.
avg	MetricValue에 포함된 값의 평균을 구합니다.

옵션	기능
cnt	MetricValue에 포함된 값의 갯수를 구합니다.

MetricValue 옵션은 UPDATE 명령어를 통해 이용할 수 있습니다.

UPDATE

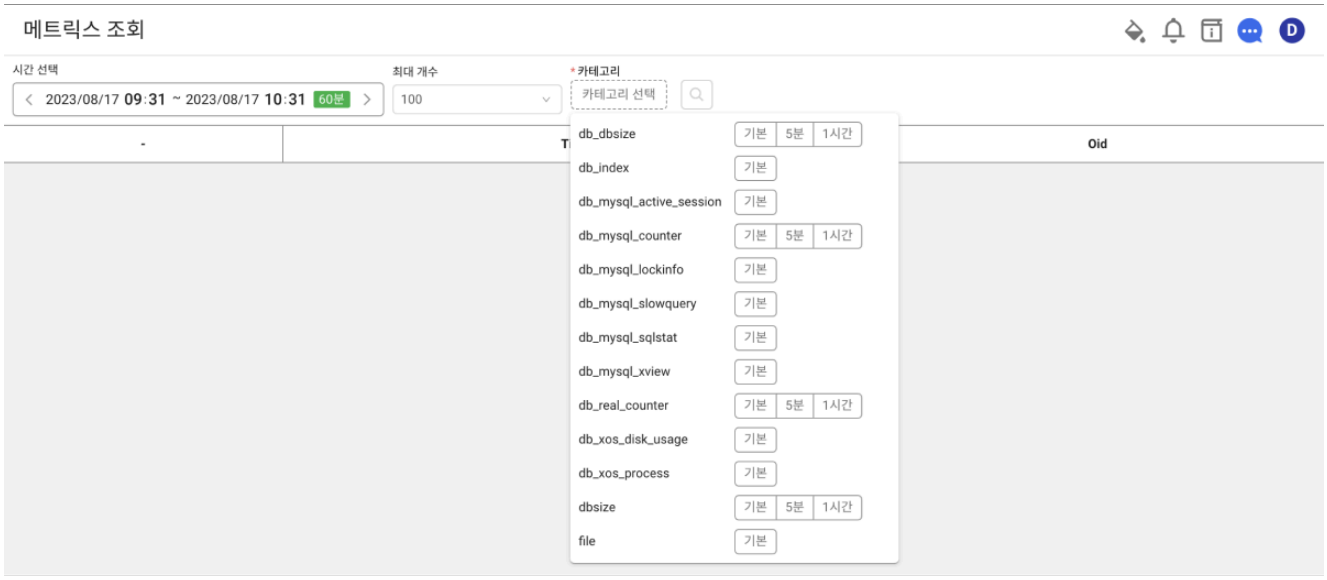
```
CATEGORY app_counter
TAGLOAD
SELECT [ time, okindName, okind, apdex_satisfied, apdex_tolerated, apdex_total]
-- GROUP 명령어의 merge 옵션을 통해 MetricValue로 변환할 field를 설정
GROUP { timeunit:5000, pk:okind, last:okindName, merge:[apdex_satisfied, apdex_tolerated, apdex_total] }
-- UPDATE 명령어를 통해 sum 옵션을 적용
UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
```

MetricValue 타입의 데이터 이용 방법

- GROUP 명령어의 merge 옵션에 필드를 설정합니다.

  ```
  CATEGORY app_counter
  TAGLOAD
  SELECT [ time, okindName, okind, apdex_satisfied, apdex_tolerated, apdex_total]
  -- GROUP 명령어의 merge 옵션을 통해 MetricValue로 변환할 field를 설정
  GROUP { timeunit:5000, pk:okind, last:okindName, merge:[apdex_satisfied, apdex_tolerated, apdex_total] }
  -- UPDATE 명령어를 통해 sum 옵션을 적용
  UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
  ```

- 수집서버에 데이터를 저장할 때부터 모든 필드에는 MetricValue 형식으로 저장된 카테고리가 있습니다. **사이트맵** > **분석** > **메트릭스 조회** > **카테고리** 옵션에는 **기본**, **5분**, **1시간** 단위로 선택할 수 있는 카테고리를 확인할 수 있습니다. 여기서 **5분** 또는 **1시간**을 선택할 수 있는 카테고리가 MetricValue 형식으로 저장된 카테고리입니다.



5분 또는 **1시간**을 선택할 수 있는 카테고리의 이름에 `{m5}` 또는 `{h1}`을 조합하면 `GROUP` 명령어의 merge 옵션을 적용하지 않고 바로 `UPDATE` 명령어의 `sum` 옵션을 적용할 수 있습니다.

```
CATEGORY app_counter{m5}
TAGLOAD
SELECT [time, pname, host_ip, pid, httpc_count]
-- GROUP 명령어를 적용하지 않아도 이미 데이터가 MetricValue 타입이기 때문에 UPDATE 명령어를 적용할 수 있습니다.
UPDATE { key : httpc_count, value : avg }
```

MetricValue 타입의 기본 연산 `avg`

MetricValue 형식 필드의 기본 출력 형태는 `avg` 입니다. 즉 MetricValue 형식의 필드는 별도의 옵션을 설정하지 않으면 `avg`를 적용합니다. 다음 두 쿼리는 같은 결과를 가집니다.

- **avg를 지정하지 않은 경우**

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode,pname, tps]
GROUP  {timeunit:5000, pk:pcode, last: pname, merge:tps}
```

- **avg를 지정한 경우**

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode,pname, tps]
GROUP  {timeunit:5000, pk:pcode, last: pname, merge:tps}
UPDATE {key:tps, value:avg}
```

사전 정의 MXQL 쿼리문

MXQL 쿼리를 직접 작성하지 않고 사전에 정의된 MXQL 쿼리파일의 경로를 설정해 MXQL을 수행할 수 있습니다. 예를 들어 '에이전트별 액티브TX 건수', '<구간별> 건수', '최근 15초'를 구하는 MXQL 쿼리는 다음과 같습니다.

```
HEADER {  act0$:I, act3$:I, act8$:I, act$:I}
TIME-RANGE {duration:15s, etime:$etime}
OIDSET {oid:$oid, okind:$okind, onode:$onode}
CATEGORY app_counter

TAGLOAD {backward:true}

SELECT [oid, oname, active_tx_0, active_tx_3, active_tx_8, active_tx_count, pcode]
FIRST-ONLY {key:oid}
RENAME {src:active_tx_0,  dst:act0}
RENAME {src:active_tx_3,  dst:act3}
RENAME {src:active_tx_8,  dst:act8}
RENAME {src:active_tx_count,  dst:act}
CREATE {key:_id_, from:oid}
CREATE {key:_name_, from:oname}

INJECT default
```

만약 위 쿼리가 수집서버에 등록되어 있다면 다음과 같이 입력하는 것만으로 같은 데이터를 조회할 수 있습니다. 설정한 경로에 저장한 서버의 파일 내용을 읽어서 호출해주는 방식입니다. 사전 정의한 파일의 경로를 입력하세요.

```
/app/act_tx/act_tx_oid
```

다음 문서에서 사용 가능한 쿼리를 확인할 수 있습니다.

참고자료

바인드 변수(파라미터)

MXQL에서는 바인드 변수를 사용할 수있습니다. 바인드 변수는 `$` 로 시작합니다. 또한 value에 해당하는 부분만 사용할 수 있습니다.

```
SKIP $skip_value
```

```
SKIP [$skip_value]
```

```
SKIP {value:$skip_value}
```

key로 바인드 변수를 전달할 수 없습니다.

```
SKIP {$option:10}
```

쿼리에서 바인드 변수를 사용했으면 MXQL을 실행할 때 입력할 값을 전달해야 합니다.

ⓘ 바인드 변수의 이름은 대소문자 알파벳만 가능합니다. 숫자 및 특수문자는 바인드 변수의 이름에 포함할 수 없습니다.

데이터 로딩 방식

MXQL에서 조회하는 데이터는 메트릭스의 구조에 따라 서로 다른 로딩 명령어를 사용합니다.

- **태그 기반 메트릭스**
  태그와 필드로 구분된 데이터는 `TAGLOAD` 명령어를 사용해 로드합니다.
- **플랫 구조 메트릭스**
  모든 데이터가 필드에 저장된 메트릭스는 `FLEXLOAD` 명령어를 사용해 로드합니다.
- **로그 원본 데이터**

로그 원본을 조회할 때는 `LogSink` 명령어를 사용합니다.

- **로그 카운트 집계 데이터**
  시간 단위로 로그 건수를 집계할 때는 `LogSinkCount` 명령어를 사용합니다.

사전 정의된 MXQL 쿼리 목록

MXQL 쿼리를 직접 작성하지 않고 경로(path)로 설정할 수 있는 기능의 주된 목적은 복잡한 쿼리를 간편하게 호출하기 위함이 아니라, 관리자가 직접 본인의 의도대로 쿼리를 작성하고 이를 사용하는데 있습니다. 따라서 이미 Yard에 어떤 쿼리들이 포함되어 있는지 확인하고 사용하기보다 직접 쿼리를 등록하는 방향으로 활용해야 합니다. `JOIN` 명령어를 사용해야하는 경우는 MXQL 쿼리를 사용하는 경우 중에서도 특별한 경우이기 때문에 관리자가 직접 쿼리를 등록하고 해당 파일의 경로(path)를 사용하도록 되어 있습니다.

> ⓘ yard.conf 파일의 `mxql_root` 에 설정한 경로 아래에 등록하고 싶은 쿼리를 파일로 저장할 수 있습니다. (default ./mxql)

에이전트별 액티브TX 건수, 건수, 최근15초

- 경로 : /app/act_tx/act_tx_oid
- 쿼리 :

```
HEADER {  act0$:I, act3$:I, act8$:I, act$:I}

TIME-RANGE {duration:15s, etime:$etime}

OIDSET {oid:$oid, okind:$okind, onode:$onode}

CATEGORY app_counter

TAGLOAD {backward:true}

SELECT [oid, oname, active_tx_0, active_tx_3, active_tx_8, active_tx_count]
FIRST-ONLY {key:oid}
RENAME {src:active_tx_0,  dst:act0}
RENAME {src:active_tx_3,  dst:act3}
```

```
RENAME {src:active_tx_8,  dst:act8}
RENAME {src:active_tx_count,  dst:act}

CREATE {key:_id_, from:oid}
CREATE {key:_name_, from:oname}
```

에이전트 별 상세정보 & 에이전트별 액티브TX 건수, 건수, 최근15초

- 경로 : /app/act_tx/agent_with_tx
- 쿼리 :

```
CATEGORY agent_list

OIDSET {oid:$oid, okind:$okind, onode:$onode}

FLEXLOAD

JOIN {query:'/app/act_tx/act_tx_oid', pk:oid, field:[act0,act3,act8, act] }

UPDATE {key:act0, notnull:0}
UPDATE {key:act3, notnull:0}
UPDATE {key:act8, notnull:0}
UPDATE {key:act, notnull:0}

RENAME {src:[act0, act3, act8, act],  dst:[normal, slow, verySlow, total]}

INJECT default
```

에이전트 별 상세정보 & 에이전트별 액티브TX 건수, 건수, 최근15초

메트릭스 선택

MXQL 문법을 이용해 메트릭스를 선택하는 명령어에 대해서 알아봅니다.

명령어	기능
CATEGORY	어떤 카테고리에서 데이터를 조회할지 선택합니다. 이 명령어와 오퍼랜드는 반드시 설정해야 합니다.
OKIND	특정 `OKIND`로부터 수집 데이터만 조회하도록 설정합니다.
OID	특정 `OID`로부터 수집 데이터만 조회하도록 설정합니다.
ONODE	특정 `ONODE`로부터 수집 데이터만 조회하도록 설정합니다.
HEADER	프론트 단에 전달하기 위한 값을 설정합니다.
TIME-RANGE	데이터를 조회할 시작 시간, 종료 시간을 설정합니다.

CATEGORY

어떤 카테고리에서 데이터를 조회할지 선택합니다. 이 **명령어**와 **오퍼랜드**는 반드시 설정해야 합니다.

오퍼랜드

- **문자열**: 지정한 카테고리의 데이터를 조회합니다. 조회시간에 상관없이 항상 `app_counter` 카테고리의 데이터를 로드합니다

  ```
  CATEGORY app_counter
  TAGLOAD
  ```

- **JSON**: 데이터 조회 시간의 길이에 따라서 카테고리를 선택하도록 설정할 수 있습니다.

  조회시간이 6시간, 3일, 그 이상인 경우에 대해 각각 `app_counter` , `app_counter{m5}` , `app_counter{h1}` 카테고리를 선택합니다.

```
CATEGORY {"app_counter":6h, "app_counter{m5}":3d, "app_counter{h1}":unlimit }
TAGLOAD
```

- `CATEGORY` 의 오퍼랜드로 하나의 값만 지정할 수 있습니다. 여러 `CATEGORY` 의 데이터를 한 번에 확인하고 싶은 경우 JOIN을 사용해야합니다.
- 로딩 방식(TAGLOAD 또는 FLEXLOAD)에 따라서 설정 가능한 카테고리가 달라집니다.
- `CATEGORY` 의 이름에 `{m5}` 또는 `{h1}` 가 붙어있는 경우 MetricValue를 참고해주세요.

OID, OKIND, ONODE

특정 `OID` , `OKIND` , `ONODE` 로부터 추출한 데이터만 조회하도록 설정합니다. `OID` , `OKIND` , `ONODE` 중 한 가지 값만 설정할 수 있습니다.

오퍼랜드

문자열로 하나의 값을 설정하거나 문자열 배열로 여러개의 값을 설정할 수 있습니다.

Example 1

```
CATEGORY app_counter
OID 1388369480
TAGLOAD
```

Example 2

```
CATEGORY app_counter
OID [1388369480, 1388369481]
TAGLOAD
```

Example 3

```
CATEGORY app_active_stat
ONODE 1693789385
```

```
TAGLOAD
```

- `OID` , `OKIND` , `ONODE` 중 한 가지 값만 설정할 수 있습니다.
- 오퍼랜드를 입력하지 않은(또는 파라미터의 값이 전달되지 않은) 명령어는 무시합니다.
- `OID` , `OKIND` , `ONODE` 를 중복해서 사용하는 경우 가장 마지막에 입력한 명령어만 적용합니다.
- `OIDSET` 은 deprecated되어 `OID` , `OKIND` , `ONODE` 중 한 가지 명령어를 사용하는 것을 권장합니다.
- 데이터 로드 단계에서부터 필요한 데이터만을 조회한다는 점에서 데이터 가공 단계의 FILTER를 적용하는 것과 차이점이 있습니다.

`OKIND` , `ONODE` 명령어의 경우 `CATEGORY` 명령어에서 설정한 카테고리에 `okind` , `onode` 관련 필드(`okind` , `onode` , `okindName` , `onodeName`)를 포함하는 경우만 사용할 수 있습니다. **사이트맵** > **분석** > **메트릭스 조회** 메뉴에서 어떤 카테고리에 어떤 필드를 포함하고 있는지 확인할 수 있습니다.

HEADER

MXQL로 조회한 데이터는 Flex 보드의 위젯에서 사용됩니다. MXQL로 조회한 데이터 중 어떤 필드를 어떤 타입으로 사용해서 Flex 보드의 위젯을 표현하는지에 대한 정보를 설정할 수 있습니다.

오퍼랜드

JSON 문자열로만 설정할 수 있습니다.

Example

```
HEADER { apdex_satisfied$:I, apdex_tolerated$:I, apdex_total$:I,  apdex$:F, category: app_counter}
OID $oid
CATEGORY app_counter
TAGLOAD
```

Flex 보드의 위젯에서 사용하는 형식에 맞추어 전달해야 합니다.

TIME-RANGE

데이터 조회 시간 범위를 설정할 수 있습니다. 기본적으로 테스트 환경의 DatePicker를 사용해서 시간을 설정합니다. 만약 테스트 환경에서 본 명령어를 사용하는 경우 DatePicker를 설정한 값을 무시합니다.

- 최근 15초 동안의 데이터만 조회

```
TIME-RANGE { recent : 15s }
CATEGORY app_counter
TAGLOAD
SELECT
```

- `etime` 이전 15초 동안의 데이터만 조회 (파라미터로 `etime` 을 전달)

```
TIME-RANGE {duration:15s, etime:$etime}
CATEGORY app_counter
TAGLOAD
SELECT
```

- `stime` 부터 15초 동안의 데이터만 조회 (파라미터로 `stime` 을 전달)

```
TIME-RANGE {duration:15s, stime:$stime}
CATEGORY app_counter
TAGLOAD
SELECT
```

메트릭스 로딩

MXQL 문법을 이용해 메트릭스를 불러오는 명령어에 대해서 알아봅니다.

명령어	기능
TAGLOAD	수집한 데이터를 `tag-field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.
FLEXLOAD	수집한 데이터를 `field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.
LogSink	로그 원본 데이터를 조회할 때 사용합니다.
LogSinkCount	시간 단위로 로그 건수를 집계할 때 사용합니다.

TAGLOAD

수집한 데이터를 `tag-field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.

옵션	기능
`{backward : true}`	시간 역순으로 데이터를 로드합니다.
`{filter : {key:fieldName, value :fieldValue}}`	fieldName 필드의 값이 fieldValue과 같은 데이터만 추출합니다.
`{filter : {key:fieldName, exclude :fieldValue}}`	fieldName 필드의 값이 fieldValue과 같은 데이터를 제외하고 추출합니다.
`{filter : {key:fieldName, like :fieldValue}}`	fieldName 필드의 값이 fieldValue을 부분 문자열로 가지는 데이터만 추출합니다.
`{filter : {key:fieldName, notlike :fieldValue}}`	fieldName 필드의 값이 fieldValue을 부분 문자열로 데이터를 제외하고 추출합니다.

- 옵션을 설정하지 않은 경우

```
CATEGORY app_counter
TAGLOAD
```

- backward 옵션을 설정한 경우

```
CATEGORY app_counter
TAGLOAD {backward : true}
```

- value filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:pid, value:905}}
```

- exclude filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:pid, exclude:905}}
```

- like filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:okindName, like:keeper}}
```

- notlike filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD {filter : {key:okindName, notlike:keeper}}
```

- 복수의 filter를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD { filter:[{key:'host_ip', exclude:'192.168.1.102'}, {key:'container', like:'gateway'}] }
```

> • `TAGLOAD` 와 `FLEXLOAD` 는 각각 설정할 수 있는 `CATEGORY` 의 값이 정해져있습니다.

- `fileter-like` 또는 `filter-notlike` 옵션을 사용할 때 값으로 숫자가 오는 경우 작은 따옴표('') 또는 큰 따옴표("")로 감싸주어야 동작합니다.

```
CATEGORY app_counter
TAGLOAD { filter:[{key:'host_ip', exclude:'192.168.1.102'},{key:okindName, like:"1"}] }
```

FLEXLOAD

수집된 데이터를 `field` 형식으로 저장하는 카테고리의 정보를 조회할 때 사용합니다.

옵션	기능
`{backward : true}`	시간 역순으로 데이터를 로드합니다.

데이터 검색조건 단계에서 설정한 정보에 따라 데이터를 로드합니다.

```
CATEGORY event_cache
FLEXLOAD {backward : true}
```

대부분의 카테고리는 `TAGLOAD` 를 사용합니다. 다음 문서에 포함한 카테고리의 데이터를 사용할 때에만 `FLEXLOAD` 를 사용합니다.

FLEXLOAD를 사용해야하는 카테고리 목록

- `agent_list` 카테고리

```
CATEGORY agent_list
FLEXLOAD
SELECT
```

- `db_agent_list` 카테고리

```
CATEGORY db_agent_list
FLEXLOAD
SELECT
```

- `agent_count` 카테고리

```
CATEGORY agent_count
FLEXLOAD
SELECT
```

- `event_cache` 카테고리

```
CATEGORY event_cache
FLEXLOAD
SELECT
```

LogSink

로그 원본 데이터를 조회할 때 사용합니다.

```
CATEGORY AppLog
LogSink
```

로그 필터링

특정 조건으로 로그를 필터링할 때 **LogTag**를 사용합니다.

```
CATEGORY AppLog
LogTag {key:city, value:"junju"}
LogTag {key:status, value:"200", exclude:true}
LogSink
```

LogTag 파라미터

- **key**: 검색할 인덱스 키
- **value**: 검색할 값 (와일드카드 사용 가능)
- **exclude**: 검색 제외 여부 (true/false)

중요사항

- LogTag는 LogSink나 LogSinkCount 앞에 기술되어야 합니다
- LogTag는 동일 키 혹은 다른 키에 대해서 다양하게 기술할 수 있습니다
- 검색 키워드 단위로 LogTag가 늘어나야 합니다

예제

특정 키워드가 포함된 로그의 건수를 조회하는 예제입니다.

```
CATEGORY AppLog
LogTag {key:content, value:"*DebugUtil*", exclude:false}
LogTag {key:content, value:"*HttpServletRequest*", exclude:true}
LogSink
SELECT
GROUP {timeunit:1d, pk:pcode, merge:[rows]}
RENAME {src:_rows_, dst:rows}
CREATE {key:_pk_, value:pcode}
UPDATE {key:rows, value:sum}
```

쿼리 설명

- 첫 번째 LogTag: content 필드에 "DebugUtil"이 포함된 로그 검색
- 두 번째 LogTag: content 필드에 "HttpServletRequest"가 포함된 로그 제외
- GROUP의 timeunit:1d로 일별 집계
- merge:[rows]로 로그 건수를 MetricValue 타입으로 변환
- UPDATE의 sum으로 일별 총 건수 계산

LogSinkCount

시간 단위로 로그 건수를 집계할 때 사용합니다.

기본 사용법

```
CATEGORY AppLog
LogSinkCount
```

시간대별로 로그 건수가 집계됩니다 (예: 1949, 1625, 1674... 건).

그룹별 집계

특정 필드를 기준으로 그룹화하여 로그를 집계할 수 있습니다.

단일 필드 그룹화

```
CATEGORY AppLog
LogSinkCount {group:oname}
```

여러 필드 그룹화

```
CATEGORY AppLog
LogSinkCount {group:"oname,oid"}
```

쉼표로 구분된 여러 개의 필드를 지정할 수 있습니다.

시간당 그룹별 집계

숫자 필드에 대한 집계 연산을 수행할 수 있습니다.

```
CATEGORY AppLog
LogSinkCount {group:oname, aggr:price.n}
```

> ⓘ Aggr에 저장할 수 있는 필드는 숫자(n) 필드에 한정됩니다.

여러 필드 집계

```
CATEGORY AppLog
LogSinkCount {group:oname, aggr:"price.n,age.n"}
```

aggr에는 쉼표로 구분된 여러 개의 필드를 입력할 수 있으나 계산은 각각 이루어집니다.

LogTag와 함께 사용

LogTag를 사용하여 특정 조건의 로그만 집계할 수 있습니다.

```
CATEGORY AppLog
LogTag {key:level, value:"ERROR"}
LogSinkCount {group:oname}
```

위 예제는 ERROR 레벨의 로그만 필터링하여 oname별로 건수를 집계합니다.

메트릭스 가공

MXQL 문법을 이용해 메트릭스를 가공하는 명령어에 대해서 알아봅니다. 메트릭스 가공 단계는 조회된 데이터들이 각 단계를 순차적으로 수행하는 구조입니다. 따라서 이 단계에 포함되는 **명령어**들의 입력 순서가 중요합니다.

명령어	기능
ROWNUM	줄번호 필드를 추가합니다.
SELECT	필드를 선택합니다. 선택되지 않은 필드은 조회되지 않습니다.
CREATE	필드를 추가합니다.
DELETE	필드를 삭제합니다.
RENAME	필드의 이름을 변경합니다.
GROUP	데이터를 그룹화합니다.
ORDER	데이터를 정렬합니다.
JOIN	다른 MQL에서 조회한 데이터를 본 데이터에 컬럼단위로 추가할때 사용합니다.
UPDATE	데이터를 가공, 정제합니다.
LIMIT	추출되는 데이터의 수를 제한합니다.
SKIP	해당 위치에서 조회되는 일부 데이터를 무시합니다.
FILTER-KEYS	특정 데이터를 가지고 있는 데이터만 추출합니다.
FIRST-ONLY	데이터 중에서 처음 데이터만을 통과 시키고 나머지는 버립니다.
TIME-FILTER	특정시간의 데이터를 SKIP할때 사용합니다.
INJECT	해당 위치에 MXQL 쿼리를 추가합니다.

명령어	기능
ADJUST	숫자 필드의 값을 변경하기 위해 사용합니다.
FILTER	특정 조건을 가지고 있는 데이터만 다음 단계로 전달합니다.

ROWNUM

줄번호 필드를 추가합니다.

Example

```
CATEGORY agent_list
FLEXLOAD
ROWNUM
```

SELECT

필드를 선택합니다. 선택하지 않은 필드는 다음 단계로 전달하지 않습니다.

옵션 이름	옵션 기능
default	`like`, `notlike`와 상관없이 조회하고 싶은 필드를 설정합니다.
like	설정한 값을 부분 문자열로 가지는 필드만 조회합니다.
notlike	설정한 값을 부분 문자열로 가지지 않는 필드만 조회합니다.

- 모든 필드를 선택하는 경우 1 (`SELECT` 명령어, 오퍼랜드를 모두 입력하지 않은 경우)

  ```
  CATEGORY app_counter
  TAGLOAD
  ```

- 모든 필드를 선택하는 경우 2 (`SELECT` 명령어의 오퍼랜드를 입력하지 않은 경우)

```
CATEGORY app_counter
TAGLOAD
SELECT
```

- 조회하고 싶은 필드 이름을 직접 설정하는 경우, 문자열배열 오퍼랜드를 사용합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode]
```

- `default`로 설정할 필드의 값이 하나인 경우와 `like` 옵션을 사용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT {default:time, like:_m}
```

- `default`로 설정할 필드의 값이 여러 개인 경우와 `like` 옵션을 사용하는 경우, `default`로 설정할 필드의 값이 여러 개인 경우와 `like` 옵션을 사용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT {default:[time,name], like:_m}
```

- `like`와 `notlike`를 모두 사용하고 싶은 경우 `SELECT` 명령어를 두 번에 나누어서 입력해야 합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT {default:[time,name], like:name}
SELECT {notlike:pname}
```

- 만약 전체 필드를 선택할때는 오퍼랜드를 입력하지 않습니다.
- `like`와 `notlike`는 한 번에 설정할 수 없습니다. 여러 번의 `SELECT`에서 나누어서 설정해야 합니다.

`SELECT` 명령어는 출력되는 필드 순서를 변경할 때도 사용합니다.

CREATE

필드를 추가합니다.

옵션 이름	옵션 기능
value	특정 값을 가지는 필드를 생성합니다.
from	설정한 필드의 값을 가지는 필드를 생성합니다.
expr	입력한 수식의 결과를 값으로 가지는 필드를 생성합니다. 수식에는 필드의 이름이 사용될 수 있습니다.
oname	`oid` 컬럼의 이름을 설정하여 `oid` 값에 대응되는 `oname`의 값을 가지는 컬럼을 생성합니다.
okind	`okind` 컬럼의 이름을 설정하여 `okind` 값에 대응되는 `okind name`의 값을 가지는 컬럼을 생성합니다.
onode	`onode` 컬럼의 이름을 설정하여 `onode` 값에 대응되는 `onode name`의 값을 가지는 컬럼을 생성합니다.

- `value` 속성을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
CREATE {key:active$, value:'#'}
```

- `from` 속성을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
CREATE {key:_id_, from:okind }
```

- `expr` 속성을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
CREATE { key:apdex, expr:" (apdex_satisfied(apdex_tolerated*0.5))/apdex_total " }
```

- `okind` 속성을 설정하는 경우

```
CATEGORY agent_list
FLEXLOAD
CREATE { key : my_okind_name, okind : okind}
SELECT [ time, okind, okindName, my_okind_name]
```

DELETE

필드를 삭제합니다.

Example

```
CATEGORY app_counter
TAGLOAD
DELETE [pcode]
```

> 반드시 문자열 배열로 입력해야합니다. `DELETE` `pcode` 는 동작하지 않습니다. (2021-06-23 기준)

RENAME

필드의 이름을 변경합니다.

- `pcode` 필드의 이름 `my_pcode` 로 변경합니다.

```
CATEGORY app_counter
TAGLOAD
RENAME { src : pcode, dst : my_pcode }
```

> `time` 은 `ORDER` 에서 최우선으로 사용하는 정렬 기준으로 `time` 의 이름을 임의로 변경하면 ORDER가 동작하지 않을 수 있습니다.

GROUP

데이터를 그룹화합니다.

옵션 이름	옵션 기능
timeunit	그룹을 나눌 시간 기준을 설정합니다.
pk or primaryKey	그룹의 `primaryKey`를 설정합니다.
last	설정한 컬럼(column)의 데이터 중 마지막 값만 저장하고 싶을 때 설정합니다. `oname`과 같이 반복해서 같은 값이 사용될 때 사용합니다. 내부적으로 설정한 값을 key로 가지는 값에 계속 덮어써집니다.
listup	설정한 컬럼(column)의 데이터를 모두 메모리에 저장하고 싶을 때 설정합니다. 내부적으로는 설정한 값을 key로 가지는 List에 계속 값이 추가됩니다.
user	실시간 사용자를 계산하기 위한 옵션입니다. Blob 타입의 데이터를 저장한 컬럼(column)만 설정할 수 있습니다.(`app_user` 카테고리의 `logbits` 등)
merge	설정한 컬럼(column)의 데이터를 MetricValue(복합값)으로 저장하고 싶을 때 설정합니다. 내부적으로는 설정한 값을 key로 가지는 MetricValue값에 추가됩니다.
rows	하나의 그룹에 최대로 저장될 수 있는 데이터의 수를 설정합니다. 기본값은 `10000`입니다.

- 설정한 필드에 대해 `merge`로 설정하여 MetricValue로 설정한 뒤 sum 연산을 수행합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, okindName, okind, apdex_satisfied, apdex_tolerated, apdex_total]
GROUP { timeunit:5000, pk:okind, last:okindName, merge:[apdex_satisfied, apdex_tolerated, apdex_total] }
UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
```

ⓘ 원칙적으로 `merge` 필드는 별도로 설정해야 합니다. 하지만 `last`, `merge`, `listup` 세 가지 속성을 모두 명시하지

ⓘ 않은 경우에는 모든 `number` 필드는 `merge` 필드로, `number` 필드가 아닌 필드는 `last` 필드로 자동 선택 됩니다.

⚠ 만약 레코드에 `time` 필드가 없으면 전체를 그룹핑합니다.

- `GROUP` 명령어가 수행되기 전에 `SELECT` 명령어로 `time` 필드를 설정하지 않은 경우
- `RENAME` 명령어로 `time` 필드의 이름을 변경한 경우
- `DELETE` 명령어로 `time` 필드를 삭제한 경우

UPDATE

필드의 데이터를 수정합니다. MetricValue 상태인 필드에 대해서 연산을 선택할 수 있습니다.

옵션	기능
sum	MetricValue에 포함된 값을 더합니다.
min	MetricValue에 포함된 값 중 최소값을 구합니다.
max	MetricValue에 포함된 값 중 최대값을 구합니다.
last	MetricValue에 포함된 값 중 마지막에 추가된 값을 구합니다.
avg	MetricValue에 포함된 값의 평균을 구합니다.
cnt	MetricValue에 포함된 값의 갯수를 구합니다.
datetime	시간 데이터의 형식을 변경합니다.
timezone	시간 데이터의 기준을 설정합니다.
notnull	설정한 컬럼의 값이 `null`인 경우 적용할 `default` 값을 설정합니다.
pct	`GROUP` 명령 수행 시 `percentile`을 위해 필드의 모든 값을 listup 했을 경우 `percentile` 값을 필드값으로 변경할

옵션	기능
	수있습니다.
decimal	필드의 숫자 데이터를 포멧팅 할 수 있습니다.

다음의 옵션을 설정해 데이터의 값을 수정할 수 있습니다.

- `value` 옵션을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pcode,pname, tps]
GROUP  {timeunit:5000, pk:pcode, last: pname, merge:tps}
UPDATE {key:tps, value:sum}
```

- `datetime` , `timezone` 옵션을 설정하는 경우 `CREATE {key:localtime, from:time}` 의 경우 `time` 필드의 값 `long` 타입의 값으로 복사합니다.

```
CATEGORY app_user
TAGLOAD
SELECT [time, pcode, pname, logbits]
CREATE {key:localtime, from:time}
UPDATE {key:localtime, datetime:'yyyyMMdd HH:mm:ss', timezone: GMT9}
```

- `notnull` 옵션을 설정하는 경우

```
UPDATE {key:tps, notnull:0}
```

- `pct` 를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, pcode, tx_count ]
GROUP { key : pcode, listup : tx_count}
UPDATE { key : tx_count, pct : 90}
```

- `decimal` 옵션을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [ time, oname, apdex_satisfied, apdex_tolerated, apdex_total]
GROUP  { timeunit:5m, pk:oname}
UPDATE { key:[apdex_satisfied, apdex_tolerated, apdex_total], value:sum }
CREATE { key:apdex, expr:" (apdex_satisfied(apdex_tolerated/2.0))/apdex_total " }
UPDATE { key:time, datetime:'yyyyMMdd HH:mm:ss', timezone:'GMT9'}
UPDATE { key:apdex, decimal:'0.000'}
ROWNUM
```

- `{datetime:'yyyyMMdd HH:mm:ss'}` 의 경우, 문자 쌍점(:)을 포함하고 있어, 반드시 작은 따옴표('') 또는 큰 따옴표("")로 감싸주어야 합니다.
- `pct: 90` 은 90%번째의 값을 선택한다는 의미입니다. 단, 해당 필드는 `GROUP` 명령을 수행할 때 `listup` 필드로 설정돼 있어야 합니다.
- format의 형식은 Java, Decimal Format을 사용합니다.

ORDER

데이터를 정렬합니다.

옵션	기능
key	정렬할 필드를 선택합니다.
sort	정렬한 direction을 설정합니다. (asc 또는 desc)
rows	같은 시간 값을 가지는 데이터를 최대 몇 개 남길지 설정합니다. default `10000`

- `key` , `sort` , `rows` 를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
```

```
SELECT [time, pname, host_ip, pid, httpc_count]
ORDER {key: [pid, host_ip, httpc_count] , sort: [desc, desc, desc], rows:2}
```

- 두 번에 나누어서 정렬하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT [time, pname, host_ip, pid, httpc_count]
ORDER {key: [pid, host_ip, httpc_count] , sort: [desc, desc, desc], rows:1000}
ORDER  {key:tps, sort:desc}
```

> 데이터에 `time` 필드이 포함되는 경우에는 `ORDER` 의 key에 `time` 이 포함되어 있지 않아도 `time` 이 정렬의 최우선 기준이 됩니다.

JOIN

`JOIN` 명령어에 대한 설명에 앞서 join의 개념을 알아 보겠습니다. join은 두 쿼리문의 결과를 합쳐서 확인하기 위해서 이용합니다. 이 때 두 쿼리의 결과 중 어떤 필드를 기준으로 합칠지에 대한 정보를 전달해야하며 이 필드를 `pk` 또는 `primaryKey` 라고 합니다.

Time	Oid	Fields			
2021-06-30 15:30:00	2031382584	field_name_1	field_name_2	field_name_3	field_name_4
		sampleData	123	2.543	testData

표1. JOIN 명령어 샘플 데이터 - 첫 번째 쿼리 결과(pk = field_name_4)

Time	Oid	Fields			
2021-06-30 15:30:00	2031382584	field_name_4	field_name_5	field_name_6	field_name_7
		testData	myData	testData	myData

표2. JOIN 명령어 샘플 데이터 - 두 번째 쿼리 결과(pk = field_name_4)

Time	Oid	Fields						
2021-06-30 15:30:00	2031382584	field_name_1	field_name_2	field_name_3	field_name_4	field_name_5	field_name_6	field_name_4
		sampleData	123	2.543	testData	myData	testData	myData

표3. JOIN 명령어 샘플 데이터 - 두 쿼리 결과를 pk를 기준으로 join한 결과

표1과 **표2**는 쿼리의 결과를 나타내며, `pk`로 설정한 `file_name_4` 필드는 파란색으로 표시되어 있습니다. **표3**은 `pk`로 설정한 `file_name_4`를 기준으로 두 쿼리의 결과가 합쳐진 모습입니다.

두 MXQL에서 조회한 데이터를 합쳐서 볼 수 있습니다. `RENAME` 명령어와 `INJECT` 명령어는 `JOIN` 명령어를 수행한 결과를 가공하는 과정으로 join의 동작에는 영향을 미치지 않습니다.

- 첫 번째 쿼리 : CATEGORY agent_list FLEXLOAD
- 두 번째 쿼리 : /app/act_tx/act_tx_oid

```
CATEGORY agent_list
FLEXLOAD
JOIN {query:'/app/act_tx/act_tx_oid', pk:oid, field:[act0,act3,act8, act] }
RENAME {src:[act0, act3, act8, act], dst:[normal, slow, verySlow, total]}
INJECT default
```

샘플 쿼리의 결과 예시는 다음과 같습니다.

Time	Oid	Fields							
2021-06-30 15:30:00	2031382584	pcode	pname	...	type	act0	act3	act8	act
		sampleData	123	...	testData	0	1	0	1

표4. 샘플 쿼리 결과

```
CATEGORY agent_list
FLEXLOAD
JOIN {query:'/app/act_tx/act_tx_oid', pk:oid, field:[act0,act3,act8,act] }
```

ⓘ Yard에 저장하는 모든 데이터는 `time`, `oid`의 값을 가지고 있습니다. 언제(`time`) 어떤 에이전트(`oid`)로부터 수집한 정보인지를 나타내기 위함입니다. 이 필드들도 `pk`로 사용될 수 있습니다.

> - `JOIN` 명령어에 사용하는 첫 번째 쿼리는 직접 작성한 MXQL 쿼리, 두 번째 쿼리는 path로 설정할 수 있는 쿼리만 가능합니다.
> - `JOIN` 명령어를 사용한 MXQL 쿼리 전체를 Yard에 파일로 등록해서 사용하면 3개 이상의 카테고리도 `JOIN` 할 수 있습니다.

LIMIT

추출하는 데이터의 수를 제한합니다. 앞에서부터 설정한 수만큼의 데이터를 다음 단계로 전달합니다.

가장 먼저 추출된 데이터 3개를 출력합니다.

```
CATEGORY app_counter
TAGLOAD
LIMIT 3
```

SKIP

이전 단계로부터 전달받은 데이터 중 일부 데이터를 무시합니다.

1~5번째 데이터는 제외되고 6번째부터 10개의 데이터를 보여줍니다.

```
CATEGORY app_counter
TAGLOAD
SKIP 5
LIMIT 10
```

FILTER-KEYS

특정 데이터를 가지고 있는 데이터만 추출합니다.

```
CATEGORY app_counter
TAGLOAD
```

```
FILTERKEYS {keys : [oid], values : [497765289]}
```

> ! key , value 가 아닌 keys , values 입니다. 복수형 s에 주의하세요.

FIRST-ONLY

특정 데이터(쌍)을 가진 첫 데이터만 다음 단계로 전달합니다.

```
CATEGORY app_counter
TAGLOAD
FIRST-ONLY {key:oid}
```

```
CATEGORY app_counter
TAGLOAD
FIRST-ONLY {key: [httpc_count, type]}
SELECT [httpc_count, type]
```

```
CATEGORY app_counter
TAGLOAD
FIRST-ONLY [httpc_count, type]
SELECT [httpc_count, type]
```

> ! 데이터 로드 단계에서 {backward : true} 를 사용했다면 본 명령어의 결과가 달라질 수 있습니다.

TIME-FILTER

특정시간의 데이터를 SKIP할때 사용합니다.

옵션	기능
time	yyyy/MM/dd HH:mm:ss 로 설정합니다. 설정한 시간 기준 duration:1000 을 설정합니다. (설정한 시간 기준

옵션	기능
	1000ms동안의 데이터 제외)
date	`yyyy/MM/dd`로 설정해야합니다. 설정한 시간 기준 duration:d1과 같이 설정합니다. (설정한 시간 기준 하루동안의 데이터 제외)
duration 또는 dur	필터링 범위를 설정합니다.(`d1` : 1일, `h1` : 1시간 , `m1` , `m5` , `m10` : 1분, 5분, 10분 , `number` : millisec)
timezone	데이터 타임존을 설정합니다. (예시, '`GMT9`')
gmt	데이터 타임존을 설정합니다. (예시, `9` 또는 `-9`)

```
CATEGORY app_counter
TAGLOAD
TIME-FILTER { date:'2020/07/28' , timezone:'GMT9'}
```

```
CATEGORY app_counter
TAGLOAD
TIME-FILTER {time:'2021/06/22 00:00:00', gmt:9 }
```

INJECT

해당 위치에 MXQL 쿼리를 추가합니다.

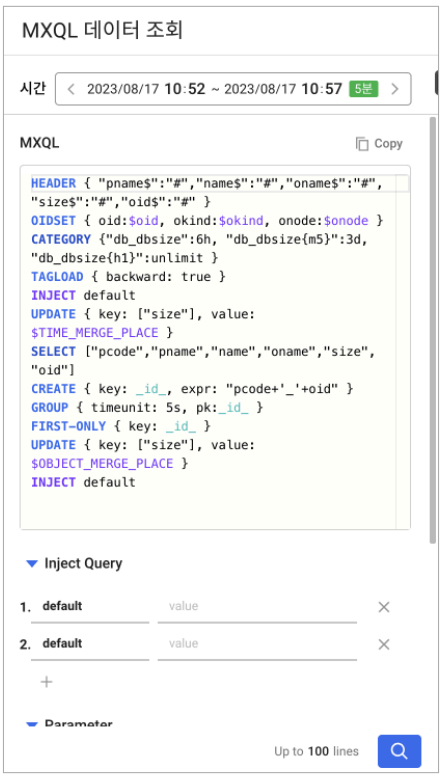
`default` 위치에 inject될 MXQL 쿼리를 전달해야 합니다.

```
CATEGORY app_counter
TAGLOAD
SELECT
INJECT default
ROWNUM
```

> ⚠ 프론트 단에서 `INJECT` 명령어의 오퍼랜드에 맵핑될 정보를 전달해주어야 합니다. 다음 예제는 키가 `default` 로

설정된 값을 추가합니다.

사이트 맵 > **MXQL 데이터 조회**'에서 INJECT 값 전달하는 예시



ADJUST

숫자 필드의 값을 변경하기 위해 사용합니다. (`time` 값은 변경할 수 없습니다.)

옵션 이름	옵션 기능
add	모든 숫자 데이터에 값을 더합니다.
sub	모든 숫자 데이터에 값을 뺍니다.

옵션 이름	옵션 기능
mul	모든 숫자 데이터에 값을 곱합니다.
div	모든 숫자 데이터에 값을 나눕니다.
over	모든 숫자 데이터에 최소값을 설정합니다.
under	모든 숫자 데이터의 최대값을 설정합니다.

- `mul` 을 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
ADJUST {mul : 100}
```

- `over` 를 설정하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
ADJUST { key:[rate], over:30}
```

- `under` 를 설정하는 경우

```
ADJUST { key:[rate], under:30}
```

FILTER

특정 조건을 가지고 있는 데이터만 다음 단계로 전달합니다.

옵션 이름	옵션 기능
expr	조건을 수식으로 입력합니다.
value	특정 값을 가지고 있는 데이터를 찾습니다.
exist	값이 있는 데이터를 찾습니다.
notexist	값이 없는 데이터를 찾습니다.
over	특정 값보다 같거나 큰 데이터를 찾습니다.(greater 또는 equal to)
under	특정 값보다 같거나 작은 데이터를 찾습니다.(less 또는 equal to)

- `expr` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER {expr : "tx_count != 0"}
```

- `value` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, value : 5}
```

- `exist` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, exist : true}
```

- `notexist` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, notexist : true}
```

- `under` 옵션을 적용하는 경우

```
CATEGORY app_counter
TAGLOAD
SELECT
FILTER { key : tx_count, under : 6}
```

> - 데이터가 0인 경우도 데이터가 존재하는 경우입니다. `{exist: true}` 에 적용됩니다.
> - `{exist : false}` 와 `{notexist : false}` 는 불가능합니다. `{notexist : true}` 와 `{exist : true}` 를 사용해주세요.

용어 사전

용어 사전은 지속적으로 업데이트됩니다.

기본 용어

모니터링

모니터링(Monitoring)이란 어떤 대상을 감시, 관찰한다는 뜻입니다. IT 서비스 분야에서는 한정된 비용과 리소스를 가지고 서비스를 제공합니다. 때문에 모니터링을 통해 예기치 못한 상황과 오류를 대비하고 극복하는 것이 매우 중요합니다.

SaaS (Software as a Service)

SaaS는 고객을 대신하여 소프트웨어와 데이터를 제공하고 관리합니다. SaaS는 개별 컴퓨터에 응용 프로그램을 다운로드하고 설치할 필요가 없습니다. 고객은 유지 보수 및 자원을 간소화하면서 비즈니스에 집중할 수 있습니다.

SaaS를 통해 서비스를 공급하는 업체는 고객을 대신해 데이터, 미들웨어, 서버 및 스토리지와 같은 모든 잠재적인 기술적 문제를 관리합니다.

애플리케이션 (Application Performance Management)

'애플리케이션 성능 관리'를 의미합니다. 보통은 APM 서비스로 불립니다.

- A는 Application을 의미합니다.
- P는 Performance, 애플리케이션의 성능을 의미합니다. 그리고 애플리케이션의 성능은 웹서비스의 응답속도를 통해 측정하게 됩니다. 웹 서비스의 응답속도를 구하기 위해 APM 서비스는 트랜잭션을 추적하고 분석하는 일을 합니다.
- M은 Management 또는 Monitoring이 사용됩니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 애플리케이션 모니터링이란?

에이전트

와탭에서 에이전트란 모니터링 대상으로부터 수집한 데이터를 와탭 수집 서버로 전달하는 애플리케이션입니다. 에이전트는 웹 서버에 설치됩니다.

에이전트는 모니터링 대상과 1 대 1 대응됩니다. 모니터링 대상이 애플리케이션이라면, 애플리케이션이 실행될 때 에이전트도 함께 실행됩니다. 모니터링 대상이 서버라면, 서버에 에이전트를 설치하고 실행합니다.

> ⓘ 에이전트 설치에 관한 자세한 내용은 다음 문서를 참조하세요.
>
> - Java 에이전트 설치
> - Linux 에이전트 설치
> - MySQL 에이전트 설치

처리량

성능에 있어서 처리량은 '얼마나 많은 요청을 완료할 수 있는가'를 의미합니다. 이 말은 '시스템이 얼마나 많은 트랜잭션을 처리하는가'를 나타냅니다.

처리량은 초 단위 또는 분 단위로 측정합니다. 처리량은 요청량과 다릅니다. 처리량은 마무리된 요청의 양입니다. 초당 요청량(RPS)이 100이지만 초당 처리량(TPS)이 10이라면 90건 요청은 아직도 처리되지 못한 상태입니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 성능 테스트 필수 항목
> - 성능 - 처리량

클라우드 컴퓨팅

클라우드 컴퓨팅은 인터넷으로 가상화된 IT 리소스를 서비스로 제공하는 것을 의미합니다. 그리고 클라우드 컴퓨팅에서 가상화하여 서비스로 제공하는 대상은 서버, 플랫폼, 소프트웨어입니다. 사용자는 사용한 만큼만 비용을 지불하면 됩니다.

클라우드 서비스의 종류는 크게 3가지가 있습니다.

- Infrastructure as a Service(IaaS, 아이아스, 이에스)
- Platform as a Service(PaaS, 파스)
- Software as a Service(SaaS, 사스)

튜닝

시스템의 성능을 개선하는 것을 의미합니다. 튜닝에는 크게 처리량 튜닝과 안정성 튜닝이 있습니다.

- 처리량 튜닝은 최대 처리량을 늘려주는 튜닝입니다.
- 안정성 튜닝은 과부하 상태에서도 시스템이 다운되지 않도록 유지하는 튜닝입니다.

평균 응답 시간

애플리케이션 서버가 사용자에게 요청 결과를 반환하는 데 걸리는 시간입니다. 와탭의 서비스는 5초 간격으로 트랜잭션의 평균 응답 시간을 계산합니다. 평균 응답 시간은 튜닝 지표로서 의미를 가집니다.

대시보드

대시보드

대시보드를 이용해 시스템 전체 현황을 실시간으로 파악할 수 있습니다. 대시보드에서는 진행 중 트랜잭션 현황, 종료 트랜잭션의 응답시간 분포, 사용자 수, CPU, 메모리 추이 등의 주요 지표를 보여줍니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 애플리케이션 대시보드
> - 서버 리소스 보드
> - 서버 컴파운드 아이

동시 접속 사용자 (Realtime User)

실시간 브라우저 사용자 수를 보여줍니다. 10초마다 최근 5분 이내에 트랜잭션을 일으킨 사용자를 카운팅 하여 보여줍니다.

사용자는 브라우저의 IP를 기반으로 카운팅 합니다. 에이전트 설정에서 사용자를 구분하기 위해 IP를 사용하거나 쿠키를 사용할 수 있습니다.

DISK I/O

Disk I/O(%) 지표는 디스크 사용률을 보여줍니다. Disk I/O(%)가 80%를 넘으면 시스템 성능에 영향을 줄 수 있습니다.

기본 경고 값은 90%입니다. Disk I/O(%)가 100%라면 디스크가 쉬지 않고 일하고 있다는 의미입니다.

DISK IOPS (Input/Output Operations Per Second)

초당 입력과 출력 작업을 나타내는 측정 단위입니다. 작업은 KiB 단위로 측정됩니다.

기본 드라이브 기술에 따라 볼륨 유형이 단일 I/O로 계산하는 최대 데이터 용량이 결정됩니다. 일반적으로 HDD의 IOPS 범위는 55-180입니다. SSD의 IOPS는 3,000 ~ 40,000입니다.

리소스 보드 (Resource Board)

리소스 보드에서 하나의 프로젝트에 등록된 모든 서버를 모니터링할 수 있습니다. 프로젝트 내 모든 서버의 요약 정보와 실시간 자원 사용량의 변화를 확인할 수 있는 CPU Resource Map 맵을 제공합니다.

전체 자원의 규모, CPU, 메모리, 프로세스의 사용률 Top 5를 보여줍니다. 리소스 보드를 통해 장애 상황을 즉시 인지하고 대응할 수 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

리소스 이퀄라이저

CPU, Memory, Disk I/O, Disk IOPS의 항목에 대해 상위 5개의 서버 목록을 실시간으로 보여줍니다.

MSA 분석 (Microservices Architecture)

URL을 기준으로 각각의 서비스 간의 호출 관계를 비율로 보여줍니다.

메트릭스 차트 (Metrics Chart)

수집된 데이터를 시계열 차트로 보여줍니다. 시간대를 선택한 후 원하는 측정 항목을 고르면 결과를 보여줍니다.

CPU 리소스 맵 (CPU Resource Map)

전체 서버들의 CPU 사용량을 나타내는 분포도입니다. 10분 동안의 데이터를 보여주고 10초 주기로 갱신됩니다.

Apdex (Application Performance Index)

Apdex는 애플리케이션 성능 지표를 의미합니다. Apdex는 응답 시간에 기반하며 전체 요청 중 만족과 허용건 비율로 수치화합니다. Apdex는 사용자 만족도에 대한 지표로 활용할 수 있으며, 0~1 사이의 값을 갖습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 성능 카운터
> - 애플리케이션 성능 지표

성능 추이

지정한 시간에 해당하는 성능에 영향을 끼치는 정보들을 조회할 수 있습니다. 또한 전체, 개별 애플리케이션 서버 별로 그래프를 확인할 수 있습니다. 성능에 많은 영향을 끼치는 애플리케이션 서버가 무엇인지 파악 가능합니다.

성능 추이에서 조회할 수 있는 정보는 다음과 같습니다.

- 실시간 사용자
- 트랜잭션/초(합계)
- 응답시간
- CPU
- 힙 메모리
- 액티브 TX
- 많이 처리된 트랜잭션 Top 10
- HTTP Call Top 10
- SQL Top 10

애플리케이션 토폴로지

프로젝트 범위에서 포함된 모든 애플리케이션의 관계 정보를 표현합니다.

액티브 스테이터스 (Active Status)

액티브 트랜잭션들을 각 상태별로 갯수를 보여줍니다.

- METHOD : 메소스 수행중인 상태
- SQL : SQL을 수행중인 상태
- HTTPC : 외부 API 호출 상태
- DBC : 트랜잭션이 Connection Pool로 부터 새로운 Connection을 획득(get)하려는 상태
- SOCKET : 외부로 TCP Socket을 연결 중인 상태

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

컴파운드 아이

와탭 에이전트가 설치된 각각의 서버를 하나의 눈(Eye)으로 표현합니다. 컴파운드 아이는 서버들을 한곳에 모아서 보여줍니다.

총 5가지의 정보를 제공합니다.

- CPU 사용량
- Memory 사용량
- Disk 사용량
- 네트워크의 Rx (수신량)
- 네트워크의 Tx (송신량)

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 컴파운드 아이

큐브

5분 단위로 만든 성능 통계를 큐브라고 부릅니다. 큐브 분석은 큐브에 저장된 5분 단위 성능 데이터를 활용한 분석 기능입니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 큐브
> - 큐브 데이터 저장소

트랜잭션 맵

종료된 개별 트랜잭션의 응답 시간 분포도입니다. 히트맵과 동일하게 분포 패턴에 따른 문제점을 발견하고 분석할 수 있습니다.

히트맵은 5분 시간 단위로 트랜잭션을 그룹화해서 보여주지만, 트랜잭션 맵은 트랜잭션을 개별로 표시합니다.

플렉스 보드

사용자가 원하는 방식으로 커스터마이즈 할 수 있는 대시보드입니다. 애플리케이션, 서버, 데이터베이스, 컨테이너 등 와탭 프로젝트의 데이터를 화면에 자유자재로 배치할 수 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

히트맵

응답 시간 분포 차트입니다. 특정 기간의 응답 시간 분포를 한눈에 파악할 수 있습니다.

트랜잭션 발생 위치에 따라 느린 트랜잭션을 쉽게 찾아낼 수 있습니다. 색상을 통하여 에러 발생 여부에 대해서도 빠르게 찾아낼 수 있습니다.
X축은 트랜잭션의 종료 시간, Y축은 응답 시간을 의미합니다. 히트맵 상의 특정 영역을 드래그하면 트랜잭션 목록을 보여주는 화면으로 이동합니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

힙 메모리 (Heap Memory)

JVM(자바 가상머신)은 프로그램을 실행하기 위해 메모리에 데이터 저장 공간을 할당합니다.

메모리 공간은 크게 3가지 영역으로 분류됩니다. Static, Stack, Heap 영역입니다. 객체(인스턴스), 배열 등 주요 데이터는 Heap 메모리 영역에 저장됩니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 대시보드 위젯 힙 메모리
> - 힙 메모리

로그

로그(Log)

로그는 애플리케이션 실행 중 발생하는 이벤트와 메시지 등을 기록한 파일입니다.

애플리케이션 활동과 발생한 이슈의 원인을 이해하려면 반드시 로그 파일을 들여다봐야 합니다.

로그 모니터링 (Log Monitoring)

와탭의 로그 모니터링을 통해서 실시간으로 로그를 조회할 수 있습니다. 혹은 특정 시간, 카테고리, 태그 또는 필터를 적용하여 원하는 로그만 선택적으로 볼 수도 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

서버

수집 서버 (Repository Server)

프로젝트 생성시 모니터링 데이터를 수집하는 서버를 의미합니다. SaaS형 서비스 사용시 와탭의 수집 서버를 사용하므로 별도로 수집 서버를 구축할 필요가 없습니다.

Redis 서버

인-메모리 방식의 데이터 저장소입니다. 와탭에선 HTTP 세션을 담는 세션 저장소로 쓰고 있습니다.

스택

스레드(Thread)

프로세스 내에서의 실행 단위입니다. 모든 프로세스에는 한 개 이상의 스레드가 존재하여 작업을 수행합니다.

그리고 두 개 이상의 스레드를 가지는 프로세스를 멀티스레드 프로세스(multi-thread process)라고 합니다. 각 스레드는 자신의 스택과 레지스터를 가지고 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

탑 스택 (Top Stack)

탑 스택은 트랜잭션의 스택 정보를 수집하여 실행중인 메소드의 사용량 통계를 제공합니다.

스택의 최상단의 사용량 통계를 통해 서비스에 가장 많은 영향을 주는 메소드를 확인합니다. 메소드의 호출 빈도를 파악하면 CPU 또는 메모리에 부하가 걸리는 원인을 분석할 수 있습니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

유니크 스택 (Unique Stack)

실행된 메소드의 집합이 동일한 경우에 대한 통계 정보입니다. 유니크 스택을 통해 많이 사용되는 스택의 정보를 알 수 있습니다.

유니크 스택에 많이 노출되었다면 통계적으로 빈번한 호출 또는 오랜 시간 동작하는 메소드입니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

액티브 스택 (Active Stack)

실행 중인 트랜잭션의 스택 정보를 수집하는 기능입니다. 스택 정보는 10초마다 수집됩니다. 수집된 데이터는 통계로 확인할 수 있습니다.

통계 정보는 긴 시간 실행되는 메소드, 짧은 시간 수행되지만 잦은 빈도로 실행되는 메소드 모두 비율을 통해 식별할 수 있습니다. 트랜잭션이 진행 중 메소드 레벨에서 어느 부분이 지연이 되었는지 확인 가능합니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.

알림

알림 (Alarm)

모니터링 프로젝트에서 문제 상황이 발생하면 다양한 채널을 통해 사용자에게 실시간으로 알림이 갑니다.

알림 서비스는 모니터링하려는 IT 시스템 특성에 맞게 사전 정의된 알림 규칙을 제공합니다. 필요 시 사용자는 구체적인 알림 조건을 설정할 수 있습니다.

> ⓘ • 경고 알림 설정하기

트랜잭션

트랜잭션 (Transaction)

애플리케이션 모니터링에 있어 트랜잭션이란, 애플리케이션에 유입된 단일 요청(request)이 애플리케이션의 처리 과정을 거쳐 응답(response)이 반환되기까지의 과정을 일컫습니다.

TPS (Transactions Per Second)

초당 처리된 트랜잭션 건수를 의미합니다. 서비스 성능지표 중 기준이 됩니다. 와탭은 프로젝트 전체 TPS를 실시간으로 보여줍니다.

트랜잭션 트레이스

단일 트랜잭션의 수행과정에서의 일련의 처리 과정을 의미합니다.

액티브 트랜잭션 (Active Transaction)

진행 중인 트랜잭션을 의미합니다.

> ⓘ 자세한 내용은 다음 문서를 참조하세요.
>
> - 액티브 트랜잭션
> - 액티브 트랜잭션이란
> - 장애를 가장 빠르게 알아내는 액티브 트랜잭션

멀티 트랜잭션

MSA와 같이 여러 애플리케이션의 호출 관계를 추적해야할 경우에 어느 부분에서 문제가 발생했고, 개선이 필요한지 식별할 수 있는 기능입니다.

Caller와 Callee

- **Caller**: 서비스를 호출한 트랜잭션(호출자)을 의미합니다.
- **Callee**: 서비스를 호출 당한 트랜잭션(피호출자)을 의미합니다.

기타

Okind

에이전트가 어떤 용도로 쓰이는지 종류를 구분하는데 쓰입니다. 예를 들어 다음과 같은 경우 해당 에이전트가 mobile_ui 용이라는 의미입니다.

```
-Dwhatap.okind=mobile_ui
```

CPU Steal Time

CPU Steal Time은 하이퍼바이저가 다른 가상 프로세서를 서비스하는 동안 가상 CPU가 실제 CPU를 기다리는 시간을 백분율로 표시한 값입니다.

가상 환경에서 동작하는 VM(Virtual Machine)은 단일 호스트에 있는 다른 인스턴스와 리소스를 공유합니다.

CPU Steal Time을 통해 VM에서 동작하는 CPU가 물리 머신으로부터 자원을 할당받기 위해 얼마나 대기하고 있는지 알 수 있습니다.