



MICROSERVICES ARCHITECTURE

Application Performance Management

Microservices Architecture 개요

KHAN
[a p m]
[s b m]
[M A C H A N]

What's Microservices?

"마이크로 서비스 아키텍처"라는 용어는 소프트웨어 응용 프로그램을 독립적으로 배치 가능한 서비스 조합 (*suite*)으로 설계하는 특정 방법을 가리키는 것으로, 지난 몇 년 동안 빠르게 확산되고 있습니다. 이 아키텍처 스타일에 대한 정확한 정의는 없지만, 비즈니스 수행(Capability)과 관련된 조직, 배포 자동화, 앤드 포인트에서의 인텔리전스 (intelligence) 그리고 프로그래밍 언어와 데이터의 분산 제어에 대한 명확한 공통적인 특징들이 존재합니다. "

Martin Fowler's blog - <https://martinfowler.com/articles/microservices.html>

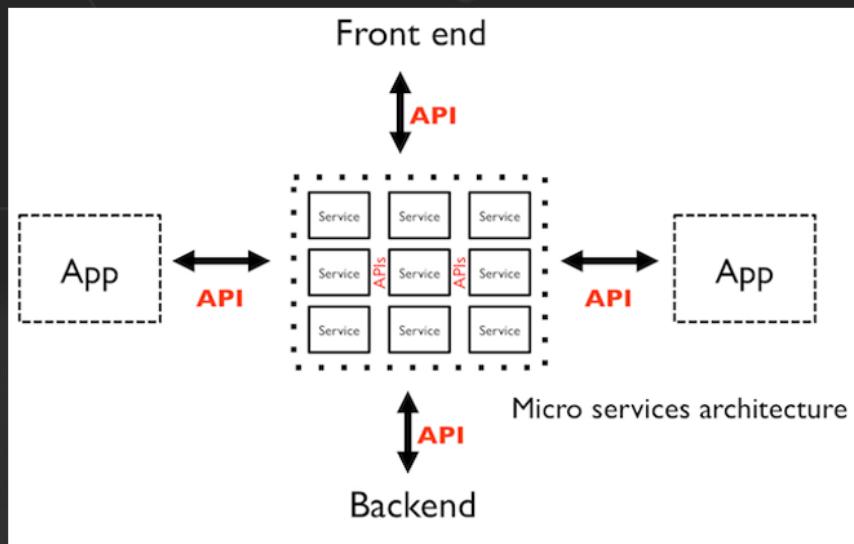
MSA 란?

- **Microservices Architecture** 약자
 - 2012년 ThoughtWorks의 James Lewis가 Java, the Unix way라는 제목의 발표에서 처음으로 언급
 - 2014년 3월 James Lewis와 Martin Fowler가 Microservices라는 타이틀로 패러다임을 정립한 기사를 발표
- 독립적이고 단순한 서비스로 전체 서비스를 구성
 - 데이터, 거버넌스, 아키텍처 등에 있어서 완전한 독립
- 독립적인 팀이 각 서비스의 개발과 운영을 담당
 - 책임과 권한에 있어서 완전한 독립을 추구



Microservices Architecture - 특징

- 시스템을 여러 개의 독립된 서비스로 나눠서, 이 서비스를 조합함으로서 기능을 제공하는 아키텍처 디자인 패턴
 - 작고(small)
 - API로 다른 서비스와 연계하며(communicate)
 - 자율적이며(autonomous)
 - 한 가지 일을 잘하는데 초점을 맞춘 서비스(focused on doing one thing well)



- 서비스란?
 - 단일된 기능 묶음으로 개발된 서비스 컴포넌트
 - REST API등을 통하여 기능을 제공
 - 데이터를 공유하지 않고 독립적으로 가공 저장
 - 인터페이스 : REST, Thrift,Protocol buffer, AMQP ...

서비스 지향 아키텍처(SOA)와의 차이점

- 서비스 기반은 같지만 목표로 하는 수준이 다름
 - SOA는 다수의 시스템이 상호작용하는 방법의 정의함
- 마이크로서비스는 독립적인 환경
 - SOA는 UI나 서비스의 변경이 필요한 경우 둘 이상의 팀에 대한 조정이 필요
 - 마이크로서비스는 개별 팀이 독립적인 환경에서 적용시키려 함

| | 서비스 지향 아키텍처 | 마이크로 서비스 아키텍처 |
|------------------|---------------------------|-----------------------------|
| 범위(scope) | 전사적 아키텍처 | 한 프로젝트에 대한 아키텍처 |
| 유연성(Flexibility) | 오케스트레이션에 의한 유연성 | 빠른 배포와 신속하고 독립적인 개발에 의한 유연성 |
| 조직(Organization) | 다양한 조직 단위에 의한 서비스 구현 | 동일 프로젝트 내의 팀들에 의한 서비스 구현 |
| 배포(Deployment) | 여러 서비스들의 단일 배포 | 각 서비스들의 독립적인 배포 |
| 사용자 인터페이스(UI) | 모든 서비스들을 위한 보편적인 UI로서의 포털 | 서비스가 UI를 포함 |

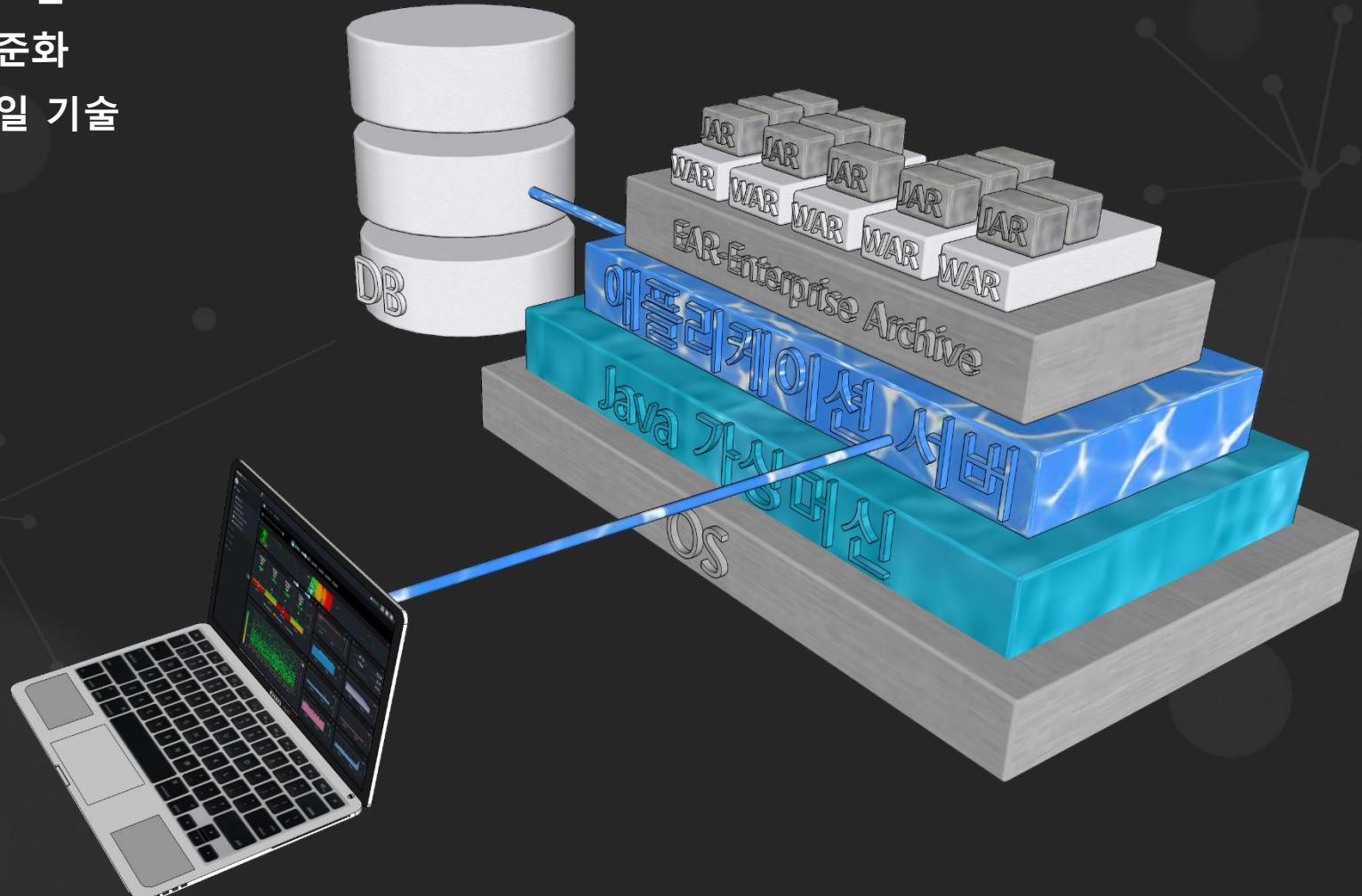
Application Performance Management

Microservices Architecture 배경

KHAN
[a p m]
m b s
M A R K E T I N G

Monolith Architecture

- 모놀리틱 아키텍처의 특징
 - 견고함
 - 표준화
 - 단일 기술



모놀리틱 아키텍처의 장단점



- 단점

- 배포가 전체 응용 프로그램에 영향을 미치는
- 개발의 영향 범위, 책임 분해 점이 어려운
- 학습장벽이 높음
- 확장성 – 스케일업
- 신기술 도입이 어려움
- 여러 개의 기술 혼용
- 배포 및 재기동 시간이 오래 걸림
- 수정이 용이하지 않음

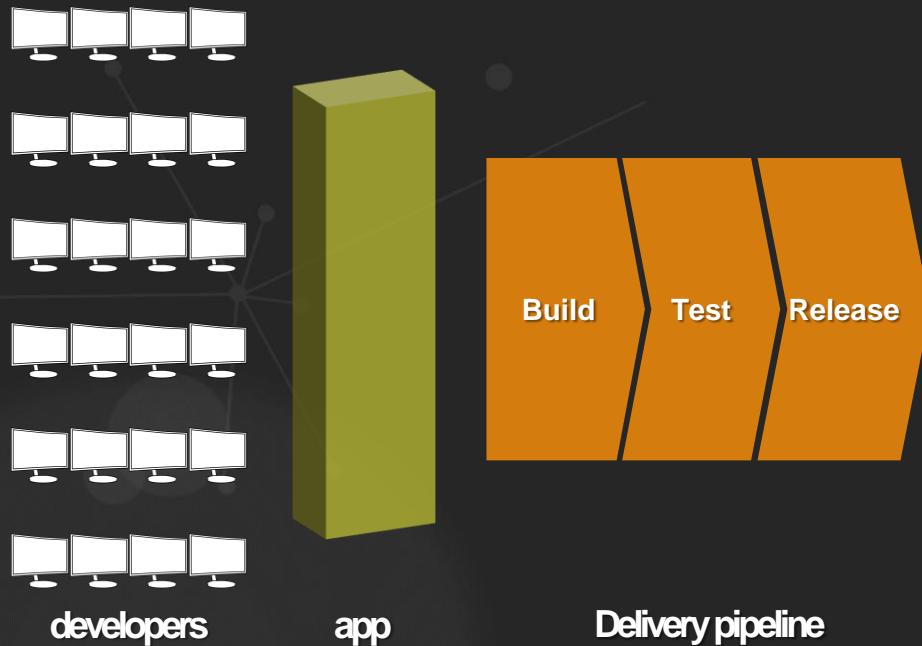
- 장점

- 기술 단일화
- 관리 용이성
- 심플한 구조
- 간편한 코드관리
- 간편한 트랜잭션관리
- 설계/테스트가 간편

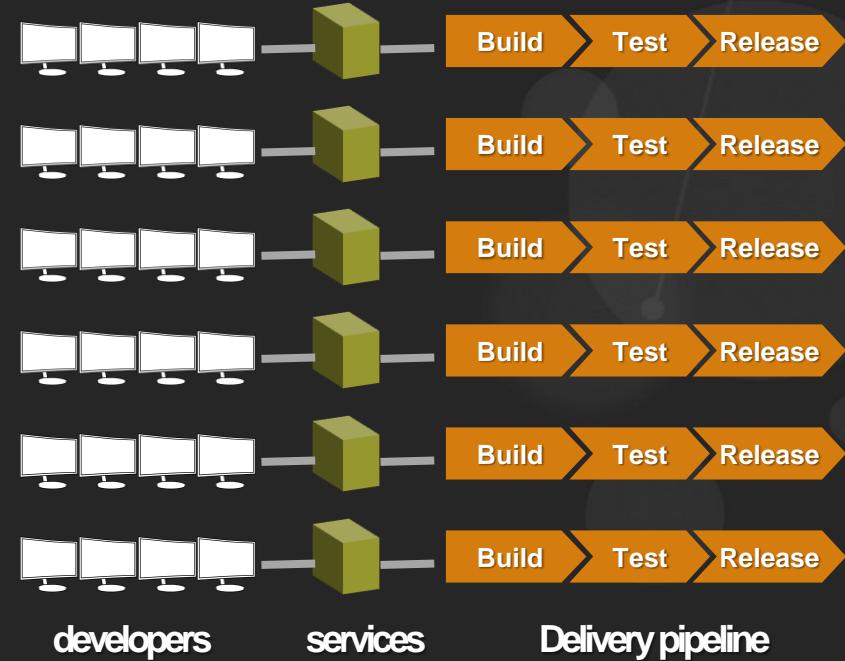
Monolith vs. Microservices

하나의 기능을 구현하는데, 여러 개의 서비스를 조합하여 기능을 제공
 예) 주문 하기 : 사용자 정보 조회, 상품 정보 조회, 신규 주문 생성

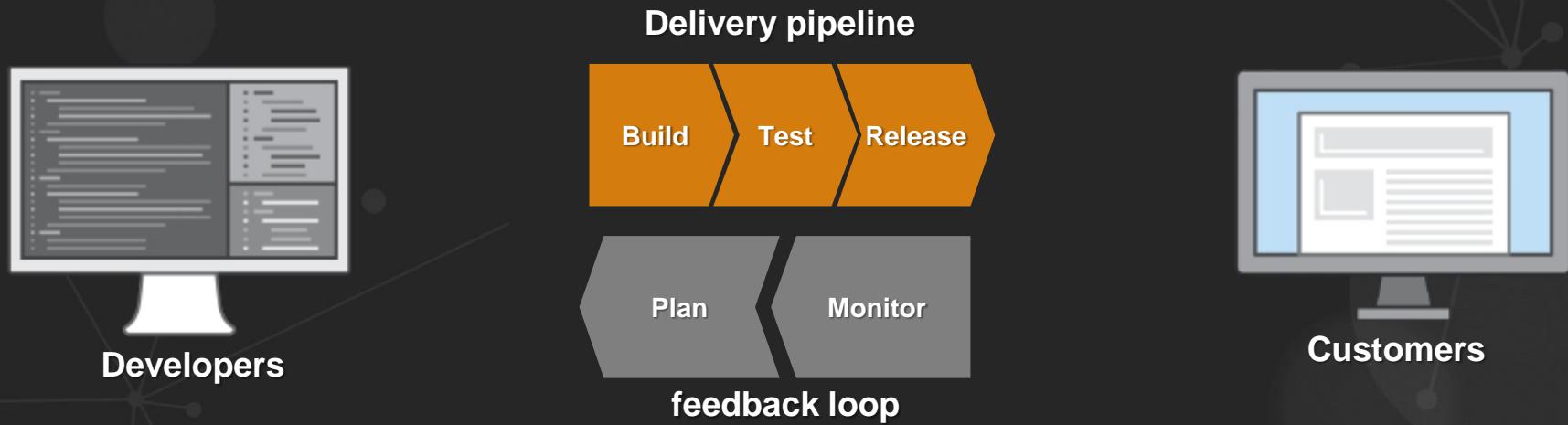
Monolith development lifecycle



Microservice development lifecycle



소프트웨어 개발 라이프 사이클



DevOps = 라이프 사이클을 효율적으로 단축

マイクロサービス ア키텍처 등장의 배경



- 기술 환경의 변화
 - 하드웨어의 저렴화와 고속화
 - 네트워크의 고속화 및 보급
 - 클라우드 환경의 충실
- 웹 서비스 레거시화
 - 기업의 IT 환경이 보다 거대하고 복잡하게 됨
 - 서비스마다 특성과 변화 속도가 크게 다르기 때문에 표준화 어려움
 - 빅뱅이 아닌 개별 서비스의 재구성
 - 거대한 웹 서비스를 관리하기 위한 필연적인 선택이 MSA
- 서비스 공유의 일반화
 - SDx 흐름에 따른 다양한 가상화 기술
 - 서비스에 대한 성능과 가용성을 보장
 - API-Gateway 를 통하여 서비스의 공유화를 실현

Conway's Law

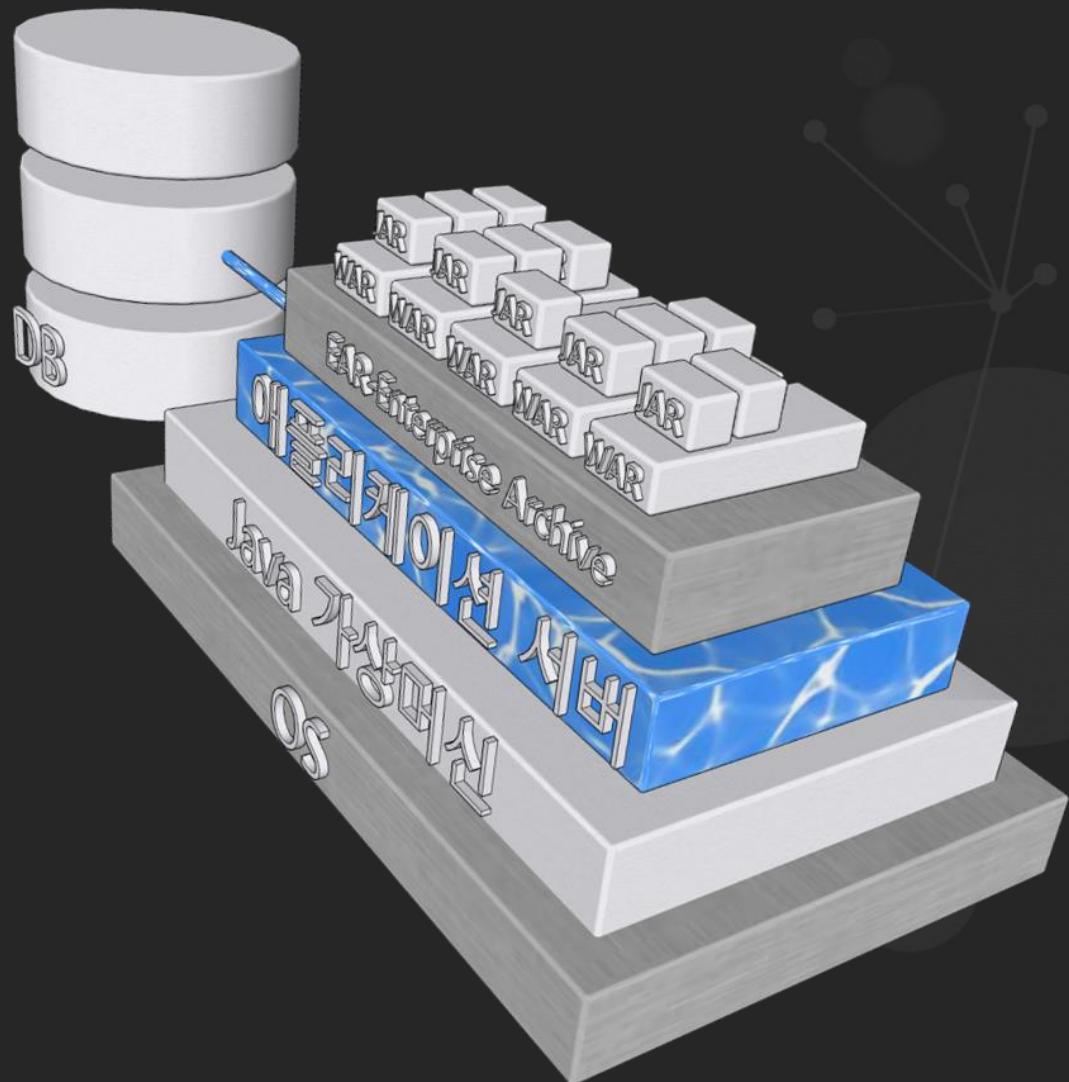
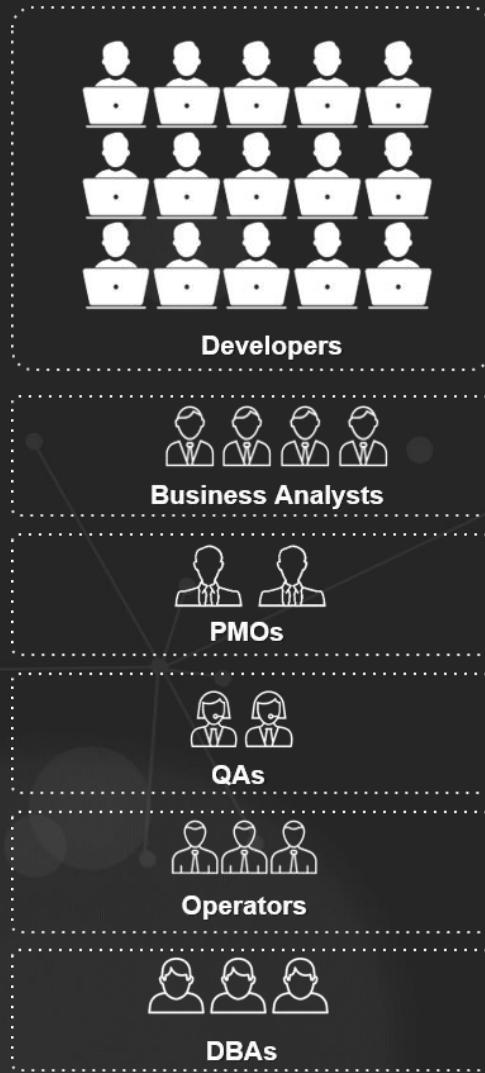
“Any organization that designs a system will inevitably **produce a design** whose structure is a **copy** of the organization’s **communication structure**.”

Melvin Conway , Datamation , 1968

MEL CONWAY'S HOME PAGE - http://www.melconway.com/Home/Conways_Law.html

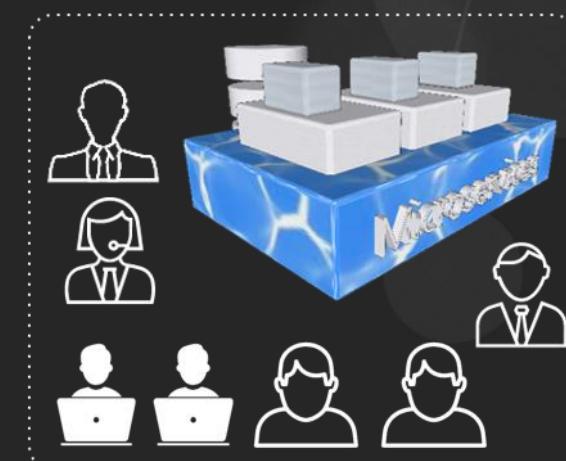
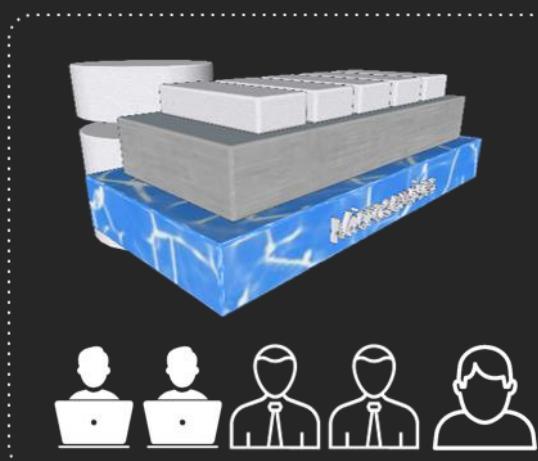
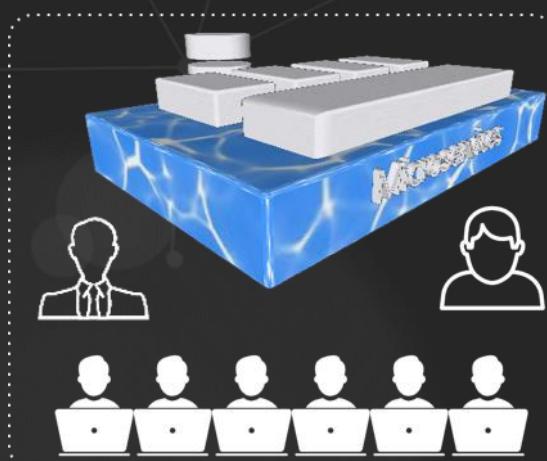
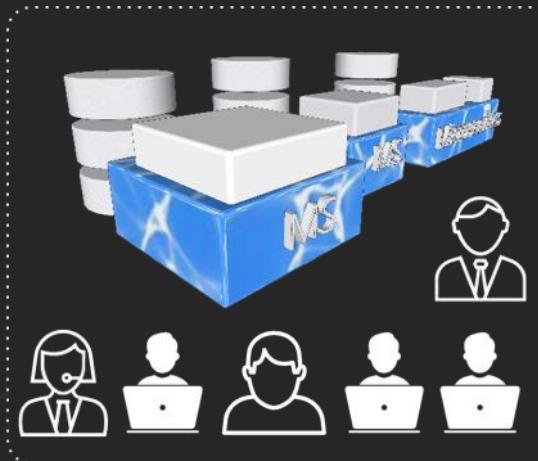
소프트웨어 아키텍처는 그것을 개발 한 조직 구조를 반영한다.

Monolith Architecture Team

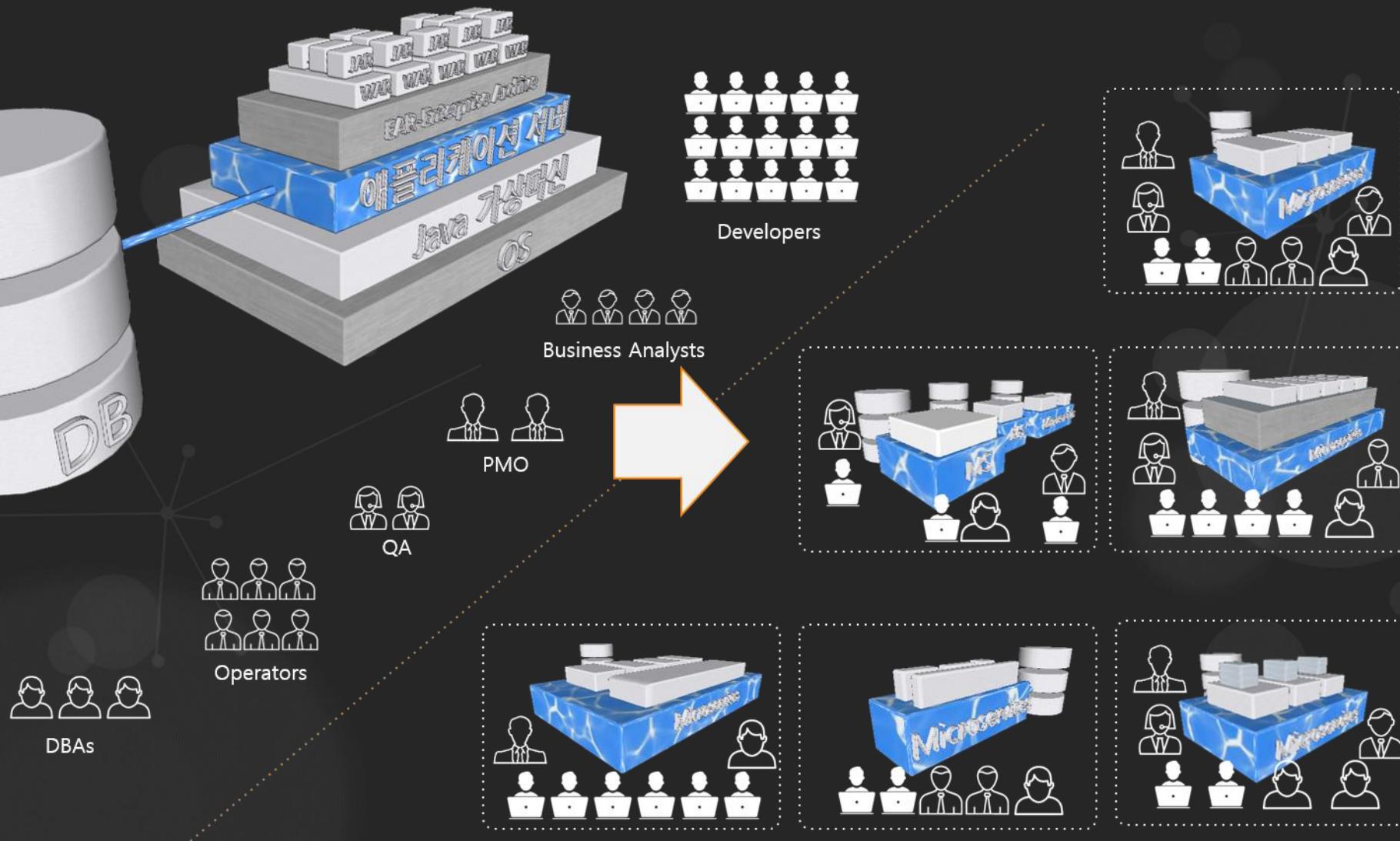


Microservices Architecture Team

- 팀을 작게 나누고 병행 개발

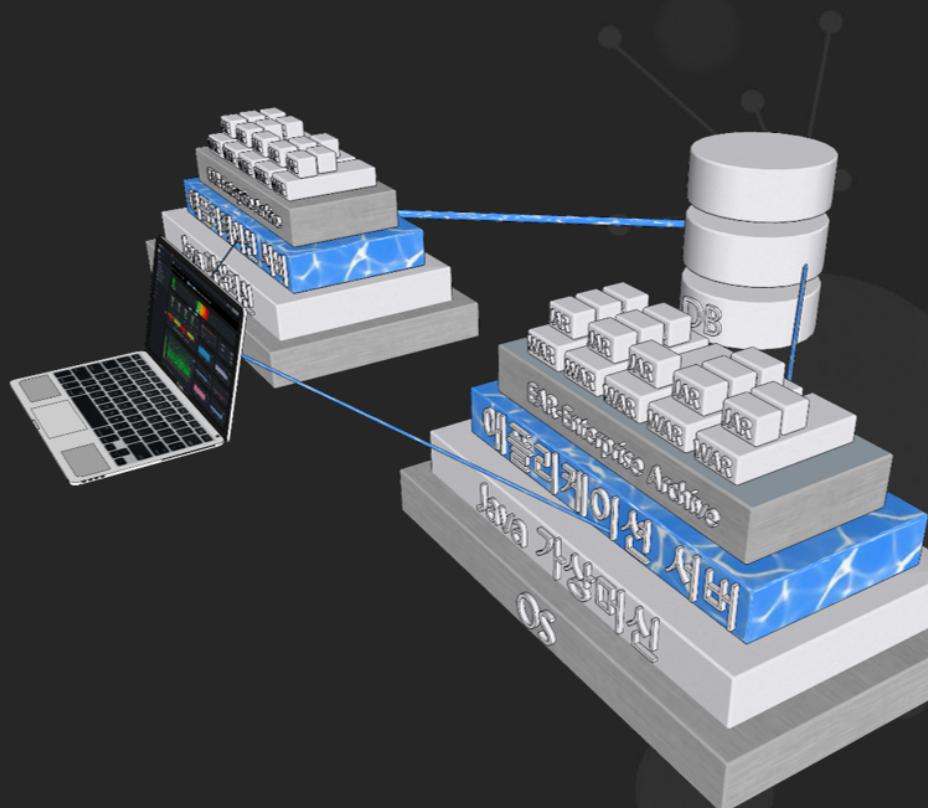


Quicker development



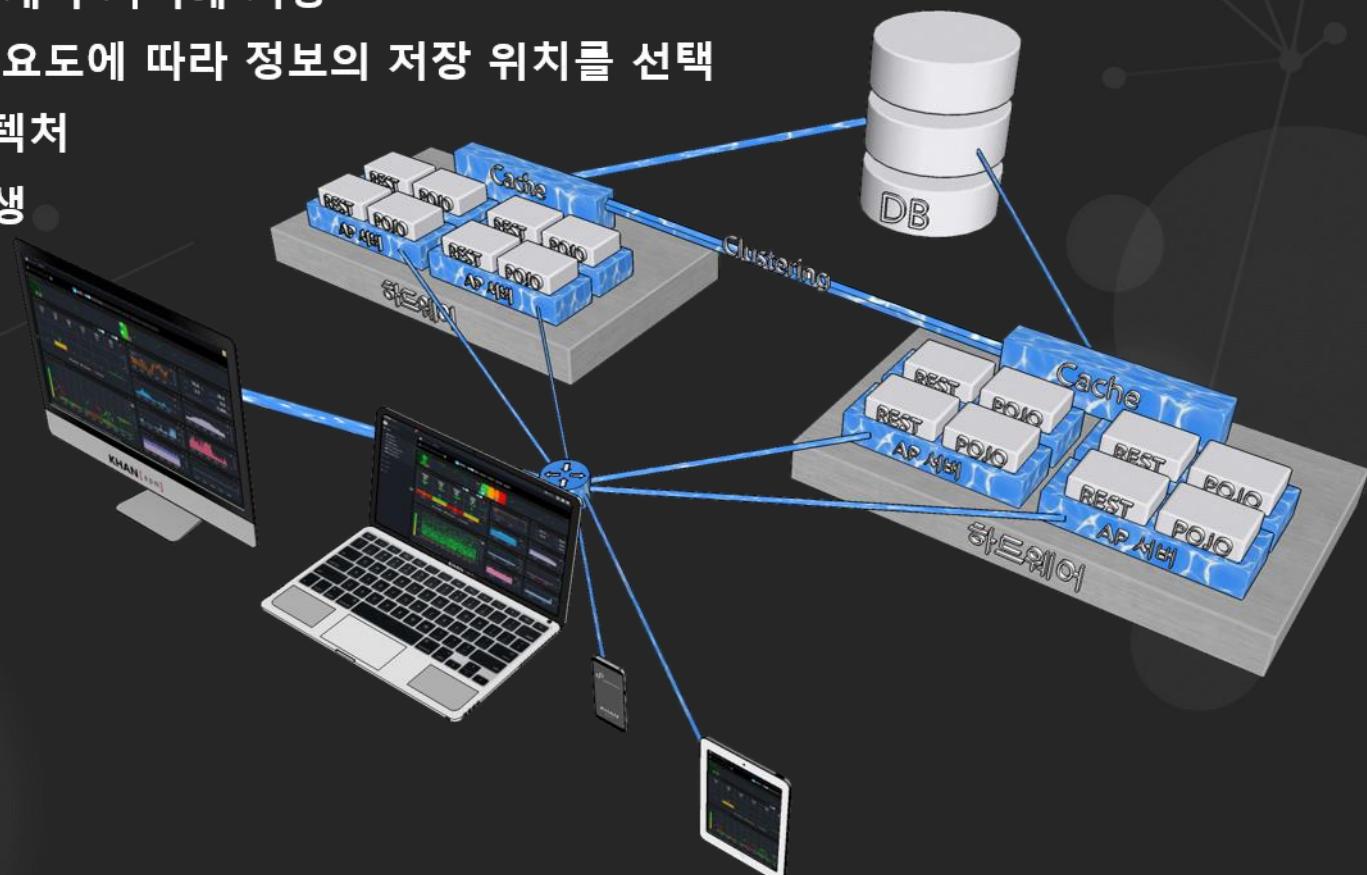
3티어 아키텍처 – 2000년대

- 클라이언트는 주로 웹브라우저를 사용하며, 서블릿이나 JSP 를 사용하여 서버 구현
- H/W와 N/W자원을 효율적으로 사용하기 위하여 App Server 인스턴스를 최소화하고 세션정보도 최소화
- 데이터 내용과 중요도에 관계없이 주요 저장소로 RDBMS를 사용
- 아키텍처 이슈
 - 인스턴스가 적기 때문에 장애의 영향이 큼
 - Hot Deploy 등을 사용하여 서버를 중지하지 않고 운영하는 구조
 - 많은 데이터가 RDBMS에 저장되기 때문에 RDBMS 튜닝에 의존
 - 확장하는 것이 거의 불가능



4티어 아키텍처 – 2010년 이후

- 멀티 디바이스와 옴니 채널 대응을 위한 REST가 주류
- H/W와 N/W자원이 저렴해져 서버 인스턴스를 여러 대 구성
 - 인스턴스가 많기 때문에 장애 범위가 작고, 서버 장애는 서비스 중지가 아님
- 세션 정보는 기본 캐시 서버에 저장
- 데이터 내용 및 중요도에 따라 정보의 저장 위치를 선택
- 확장 가능한 아키텍처
- 관리 오버헤드 발생



Application Performance Management

Microservices Architecture 특징

マイクロサービス 장점

기존 모놀리스 아키텍처 탈피

- 개별 마이크로서비스로 확장 가능



서비스 별로 독립적인 운영

- 서비스에 대해 유연한 추가, 변경이 가능
- 서비스의 변화나 장애가 전체 시스템에 영향을 주지 않음
- 서비스마다 성능을 확장

서비스 별로 독립적인 개발

- 모든 서비스에 대해 동일한 프로그램 언어나 DB 등을 사용할 필요 없음
- 서비스에 최적이라고 생각되는 것을 선택
- 최신 기술의 도입/실험에 부담이 없어 개발팀 역량 강화

서비스 간에 독립성이 유지

- 개별적인 문제가 전체에 영향을 주지 않음
- API 디자인 능력 향상

비즈니스 경계와 더 밀접하게 정렬됨

- 변화에 강한 시스템을 실현하기 쉬워짐
- 좀 더 유연한 비즈니스 지원 체계 구축

유연한 개발과 운영

- 병렬 개발과 배포 지원
- 신속한 배포

Microservices 9 가지 특징 (1/2)

1. 서비스 단위의 컴포넌트화 (Componentization via Services)

- 서비스를 HTTP 나 RPC에 연계하는 컴포넌트로 구현

2. 비즈니스 기능 중심의 구성 (Organized around Business Capabilities)

- 기술이 아닌 비즈니스 기능에 따라 서비스를 분할한다 - 콘웨이 법칙

3. 프로젝트가 아니라 제품 (Products not Projects)

- 기능을 제공하는 것이 끝이 아니고, 사용자가 사용할 수 있는 제품수준으로 개발

4. 스마트 엔드포인트와 간단한 파이프 (Smart endpoints and dumb pipes)

- 복잡한 프로토콜이 아닌 RESTful 한 HTTP API 및 경량 메시징 프로토콜을 사용

Microservices 9 가지 특징 (2/2)

5. 개발의 분권화 (Decentralized Governance)

- 특정 기술에 고착되고 중앙 집권적인 관리에서
가볍고 최적화된 기술 선택으로 전환

6. 데이터 관리의 분권화 (Decentralized Data Management)

- 개별 서비스마다 데이터베이스를 소유

7. 인프라 자동화 (Infrastructure Automation)

- 프로비저닝을 통한 인프라 구성에 대한 자동화

8. 장애를 전제로 한 설계 (Design for failure)

- 장애가 서비스에 영향을 주지 않도록 하며, 모니터링 체계 구축

9. 변화에 대응하는 설계 (Evolutionary Design)

- 변화에 쉽게 대응할 수 있도록 작은 규모의 조직 구성

Microservices 특징

기술측면 : 분산과 통합

- 서비스 단위의 컴포넌트화
- 스마트 엔드포인트와 간단한 파이프
- 데이터 관리의 분권화
- 인프라 자동화
- 장애를 전제로 한 설계

문화측면 : 지속성과 분권

- 비즈니스 영역에 따른 조직화
- 프로젝트가 아니라 제품
- 개발의 분권화
- 변화에 대응하는 설계

서비스를 서비스로 구성

- 정적 결합에서 동적으로 서비스를 조합

메시지에 의한 통합

- 개별 서비스는 표준 프로토콜로 통신

서비스 관리

- 모니터링, 의존성 관리, 장애 감지 및 복구, 버전 관리

서비스를 지속적으로 운영

- 소프트웨어 개발에서 IT 서비스 운영으로 전환

도메인 관련 기술과 운영

- 각 도메인 별로 독립성을 인정

도메인 별 라이프 사이클 관리

- 개별적인 구성을 허용

Application Performance Management

Microservices Architecture 설계

KHAN
a p m
m d s
w h s
N A H

- 개발 및 운영 복잡성
 - 서비스 단위로 분리하고 실제 서비스 가능한 상태로 만든다는 것은 그만큼 시스템이 늘어나며, 운영 복잡도가 증가함
- 코드 의존성
 - 개별 서비스를 독립적으로 배포할 수 있기 때문에 서로 다른 라이브러리 버전들이 바이너리 의존성 때문에 더 이상 호환되지 않을 수 있음
- 성능 저하
 - 서비스가 분리되어 늘어난 통신들로 인해 성능이 저하될 수 있음
- 인프라 스트럭처와 운영
 - 모니터링, 배포, 테스트가 자동화되어 있어야 함

マイクロサービス ア키텍처와 거버넌ス

- SOA : 하향식 vs. MSA : 상향식
 - SOA : 이상적이며 전체 변경 vs. 현실적이며 부분 최적화 변경
 - SOA : 전체 시스템을 통제 vs. 전체 서비스를 분할하여 통제
- 봉건 → 군주제 → 민주주의로의 변화와 유사

봉건제 :
지역 영주에
의한 분리 통치



Legacy : 연결성 없음

군주제 :
왕에 의한
중앙집권제 통치



SOA : 벤더 중심

민주주의:
국민에 의한 통치



MSA : 개별 시스템 중심

There's no silver bullet.



도메인 = 변화의 경계선

- 변화의 경계선을 찾는다
 - 모듈화는 변화의 경계선에 의해 일어난다
- 변화의 요인 (외부 / 내적)는 품질 특성에서 이해하기
 - 기능뿐만 아니라 비 기능도 중시해야 한다
- 변화의 경계에 선을 긋는다
 - 변화의 경계는 불분명하지만 선을 그릴 수 밖에 없다
- 도메인 경계를 유지한다
 - 패키지 문제 또는 재 구축 문제

12 Factor App 방법론

- 최근에는 소프트웨어가 서비스로 제공되며, 웹 애플리케이션과 **Software as a Service**라고 함
- **Twelve-Factor App**은 다음과 같은 **Software as a Service**를 만들기 위해 방법론
 - 설치 자동화에 대한 선언적인 형식을 사용하여 프로젝트에 추가 된 새로운 개발자가 시간과 비용을 최소화
 - 하부 OS에 의존 관계를 명확히 하고 실행 환경 사이의 이식성을 최대화
 - 현대적인 클라우드 플랫폼에서의 배포에 적합하며, 서버 관리 및 시스템 관리가 불필요
 - 개발 환경과 운영 환경의 차이를 최소화하고 민첩성을 극대화하는 지속적인 배포가 가능
 - 아키텍처 개발 사례를 크게 변경하지 않고 확장
- **Twelve-Factor** 방법론은 어떤 프로그래밍 언어로 작성된 응용 프로그램에도 적용 가능
- 백엔드 서비스 (데이터베이스, 메시지 서버, 메모리 캐시 등)와 조합하여 적용

The 12 Factor App (1/2)

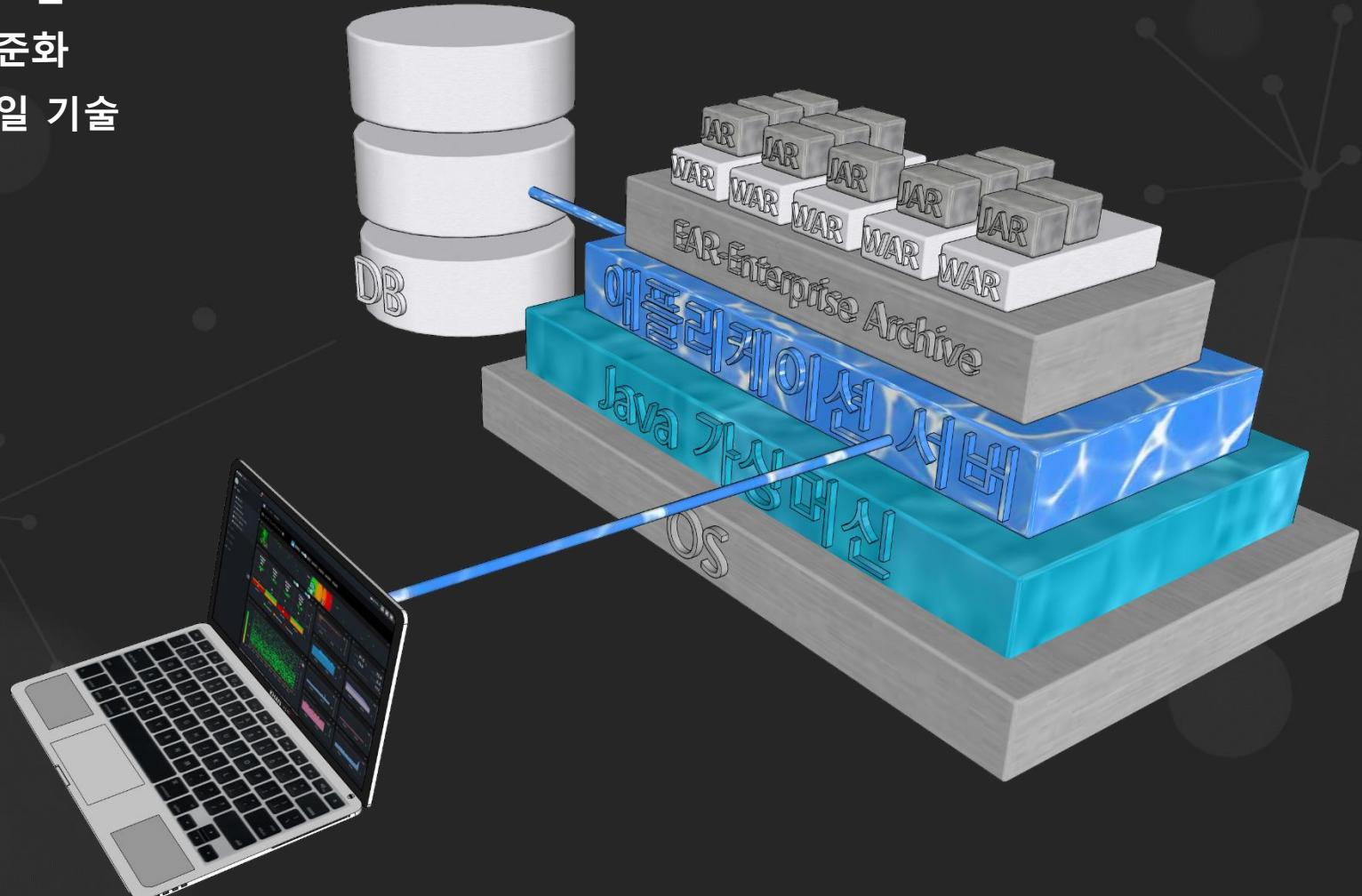
- I. 코드베이스
 - 버전 관리되는 하나의 코드베이스로 여러 곳에 배포
- II. 종속성
 - 의존 관계를 명시적으로 선언하고 분리
 - 환경에 의존하지 않도록 함
- III 설정
 - 설정을 환경 변수에 저장하기
- IV. 백엔드 서비스
 - 백엔드 서비스를 연결된 리소스로 취급
- V. 빌드 릴리스, 실행 (Build, release, run)
 - 빌드, 릴리스, 실행 3 단계를 엄격하게 분리
- VI 프로세스
 - 응용 프로그램을 하나 또는 여러 개의 독립적인 프로세스로 실행
- VII. 포트 바인딩
 - 포트 바인딩을 통해 서비스를 공개

The 12 Factor App (2/2)

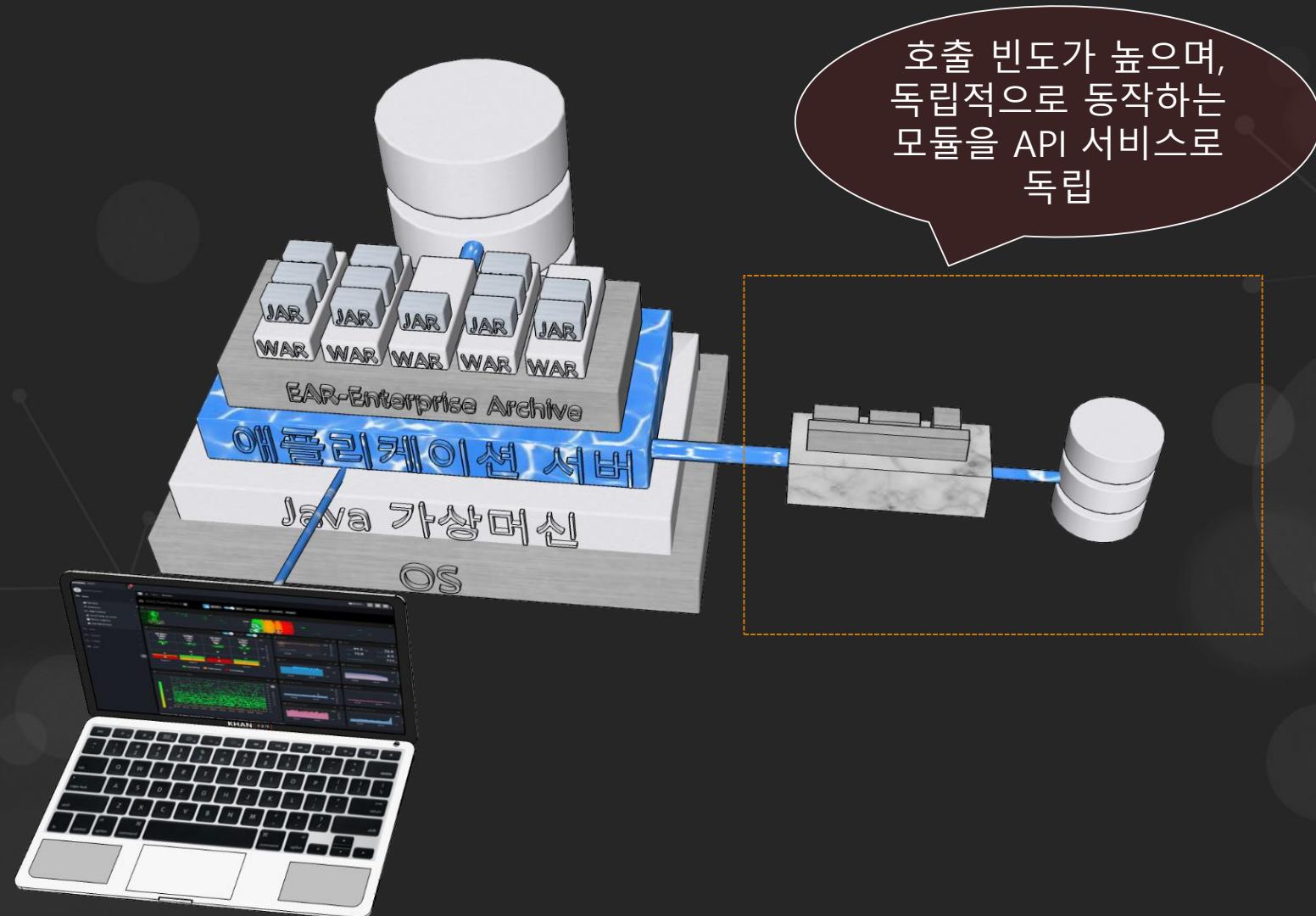
- **VIII. 동시성**
 - 프로세스 모델에 따라 확장
- **IX. 폐기 용이성**
 - 빠른 시작이 가능하며, Graceful Shutdown시 서비스에 영향을 미치지 않도록 함
- **X. 개발 / 운영 일치**
 - 개발 준비 프로덕션 환경과 최대한 일치된 상태를 유지
 - CI / CD (Continuous Integration/Continuous Delivery)환경이 갖춰져 있어야 함
- **XI. 로그**
 - 로그를 이벤트 스트림으로 취급함
 - 중앙 집권적인 서비스를 통해 로그 이벤트를 수집하고, 인덱싱하여 분석하는 환경이 가능해야 함
- **XII. 관리 프로세스**
 - 관리 작업을 일회성 프로세스로 실행

Monolith Architecture

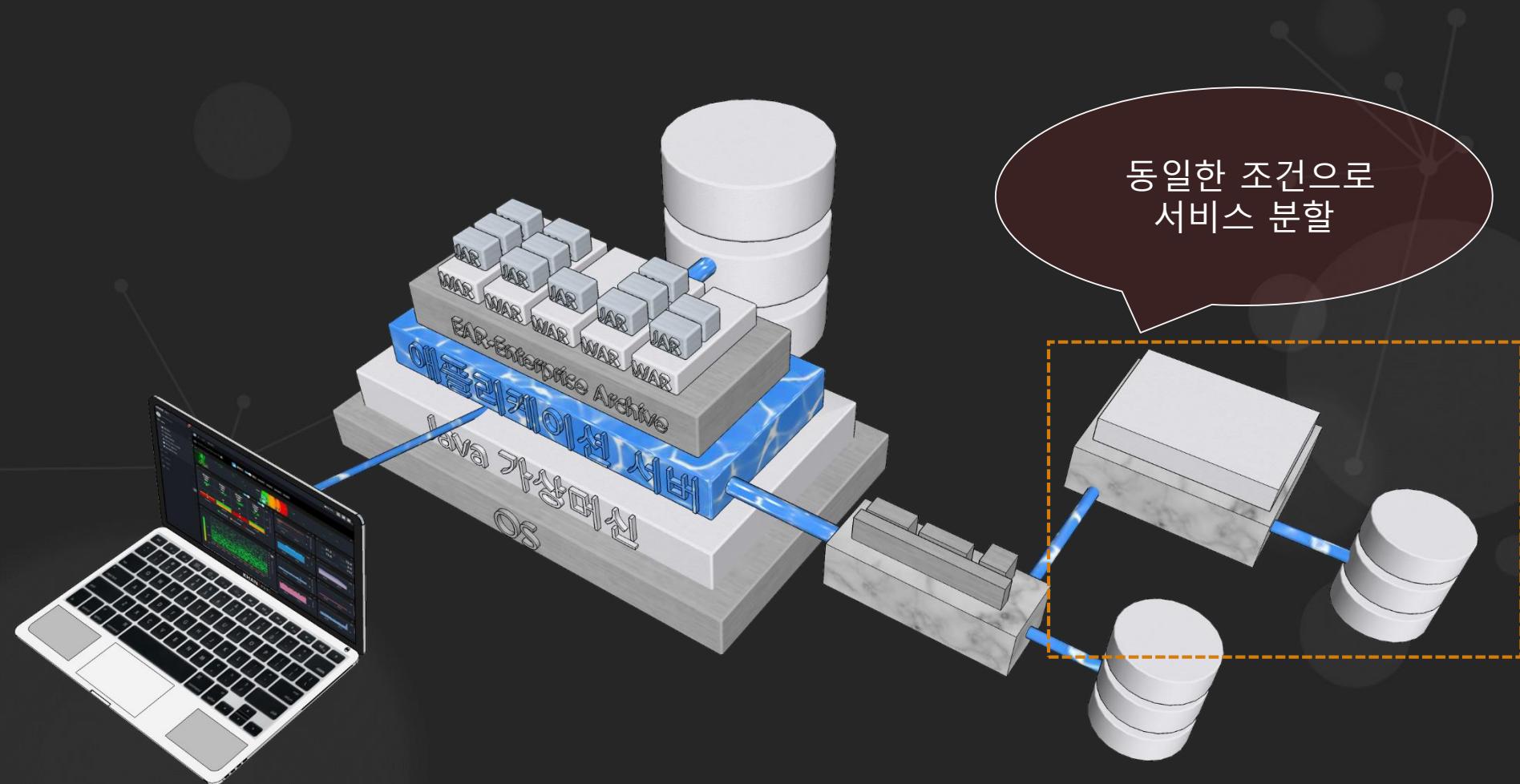
- 모노리스 아키텍처의 특징
 - 견고함
 - 표준화
 - 단일 기술



Monolith → Microservices – Phase 1

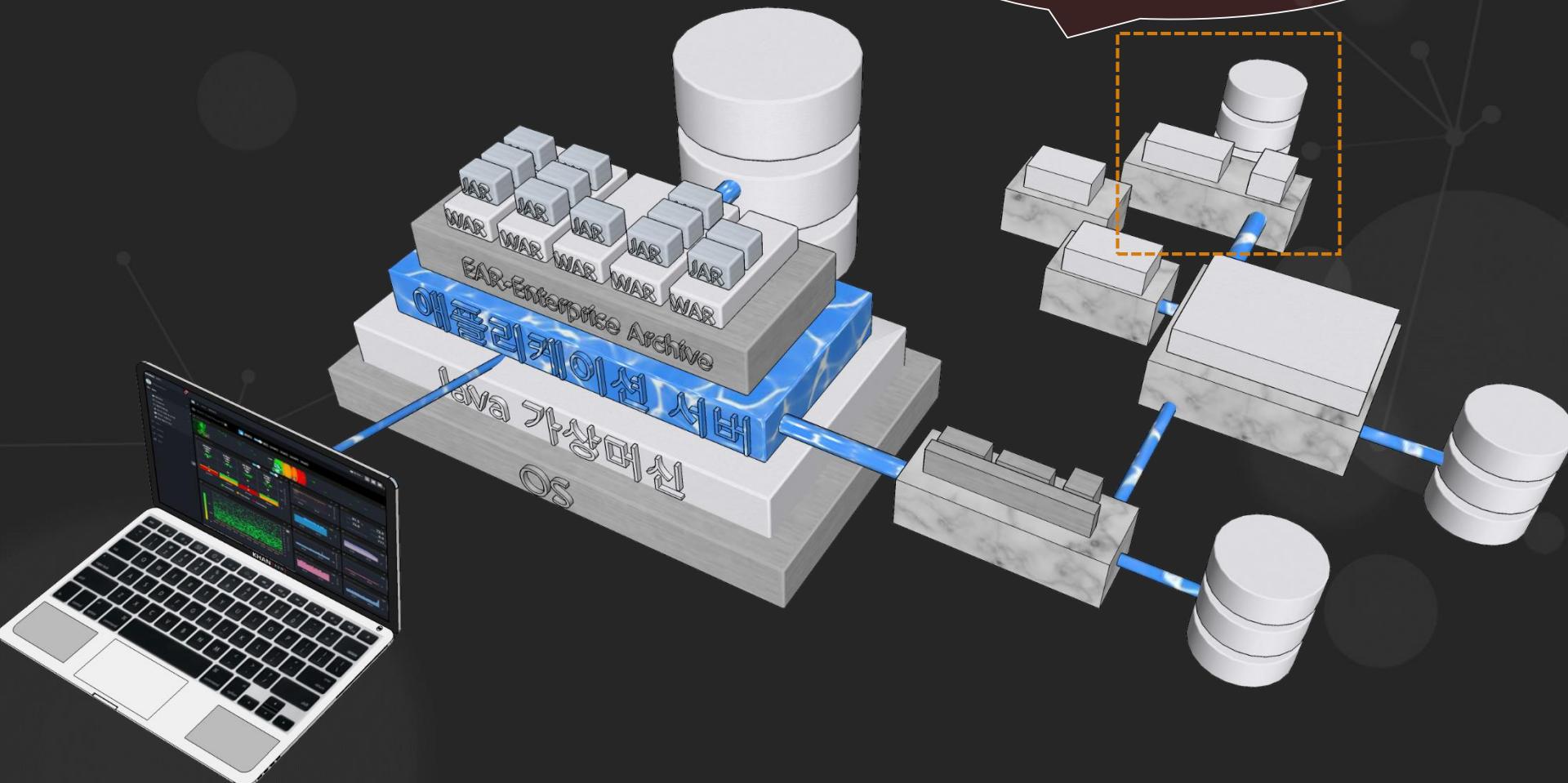


모노리스에서 마이크로서비스로 전환 – Phase 2

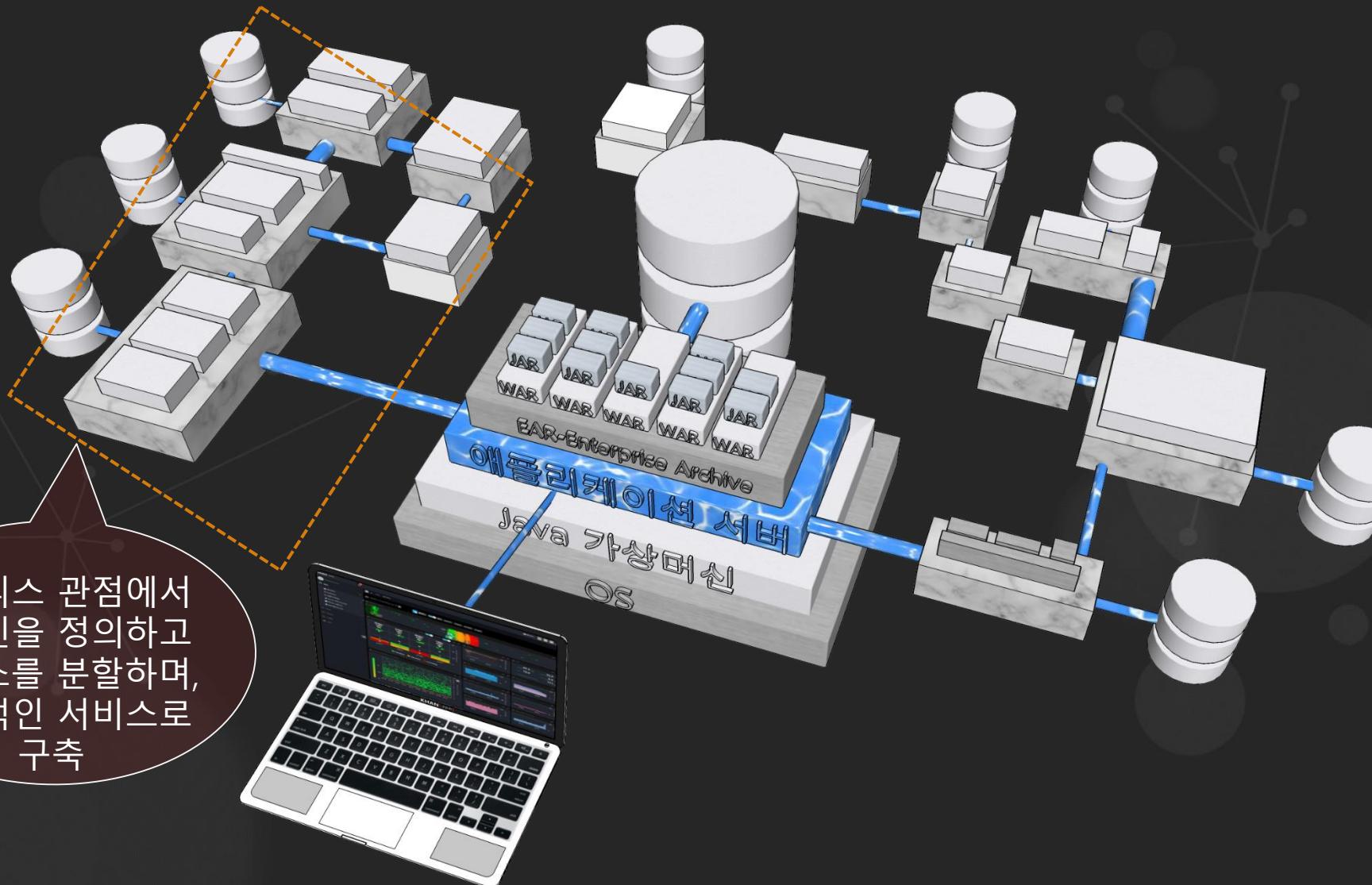


모노리스에서 마이크로서비스로 전환 – Phase 3

동일한 조건으로
서비스 분할

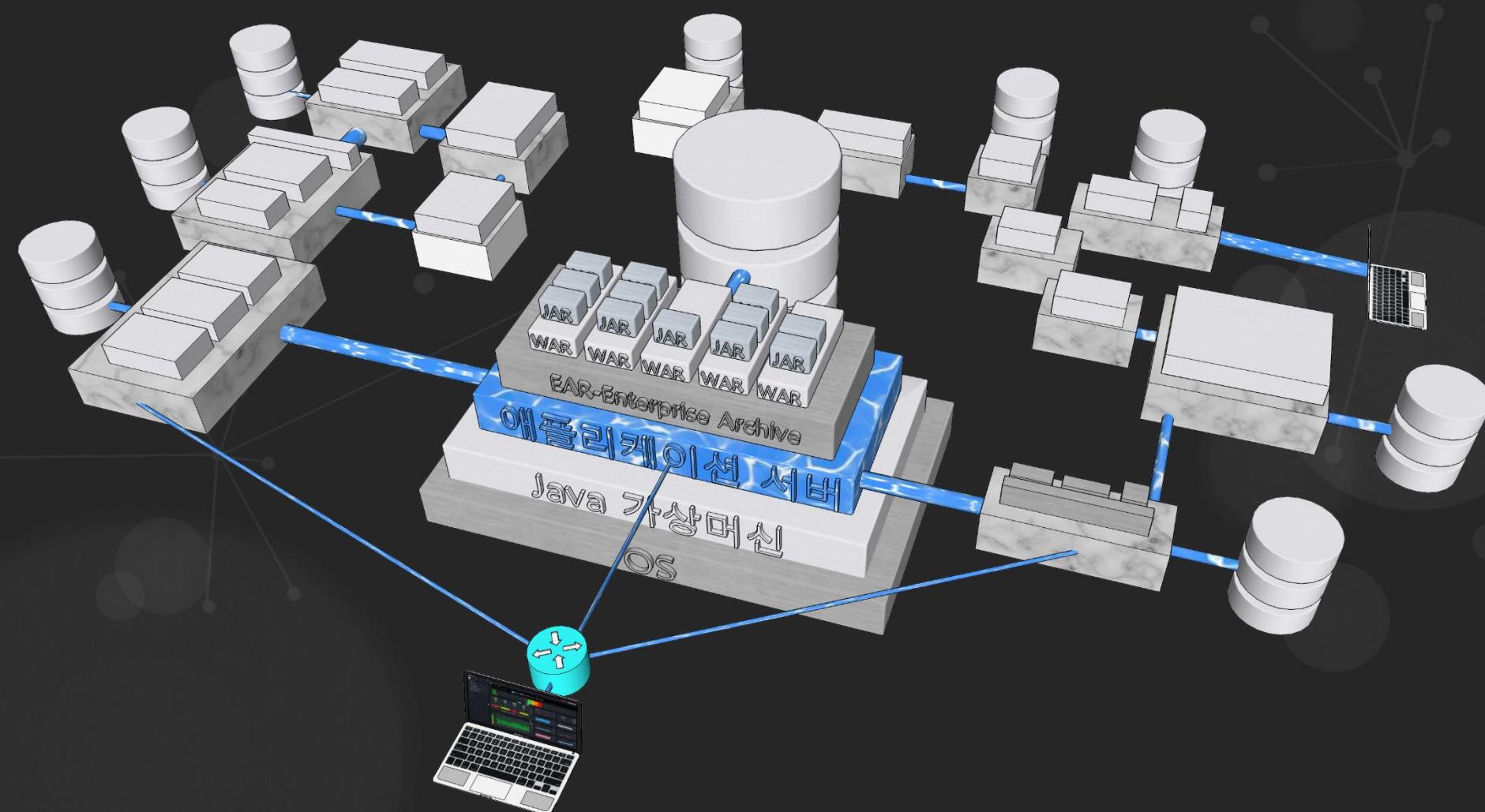


모노리스에서 마이크로서비스로 전환 – Phase 4



모노리스에서 마이크로서비스로 전환 – Phase 5

- 개선을 반복하여 최적의 마이크로 서비스를 제공



마이크로 서비스 데모 – 양말 가게



OFFER OF THE DAY Buy 1000 socks, get a shoe for free!

Logged in as User Name | Logout

weaveworks
SOCKS

HOME CATALOGUE ACCOUNT

0 items in cart

WE LOVE SOCKS!

Fun fact: Socks were invented by woolly mammoths to keep warm. They died out because stupid humans had to cut their legs off to get their socks.

BEST PRICES

We price check our socks with trained monkeys back at the office.

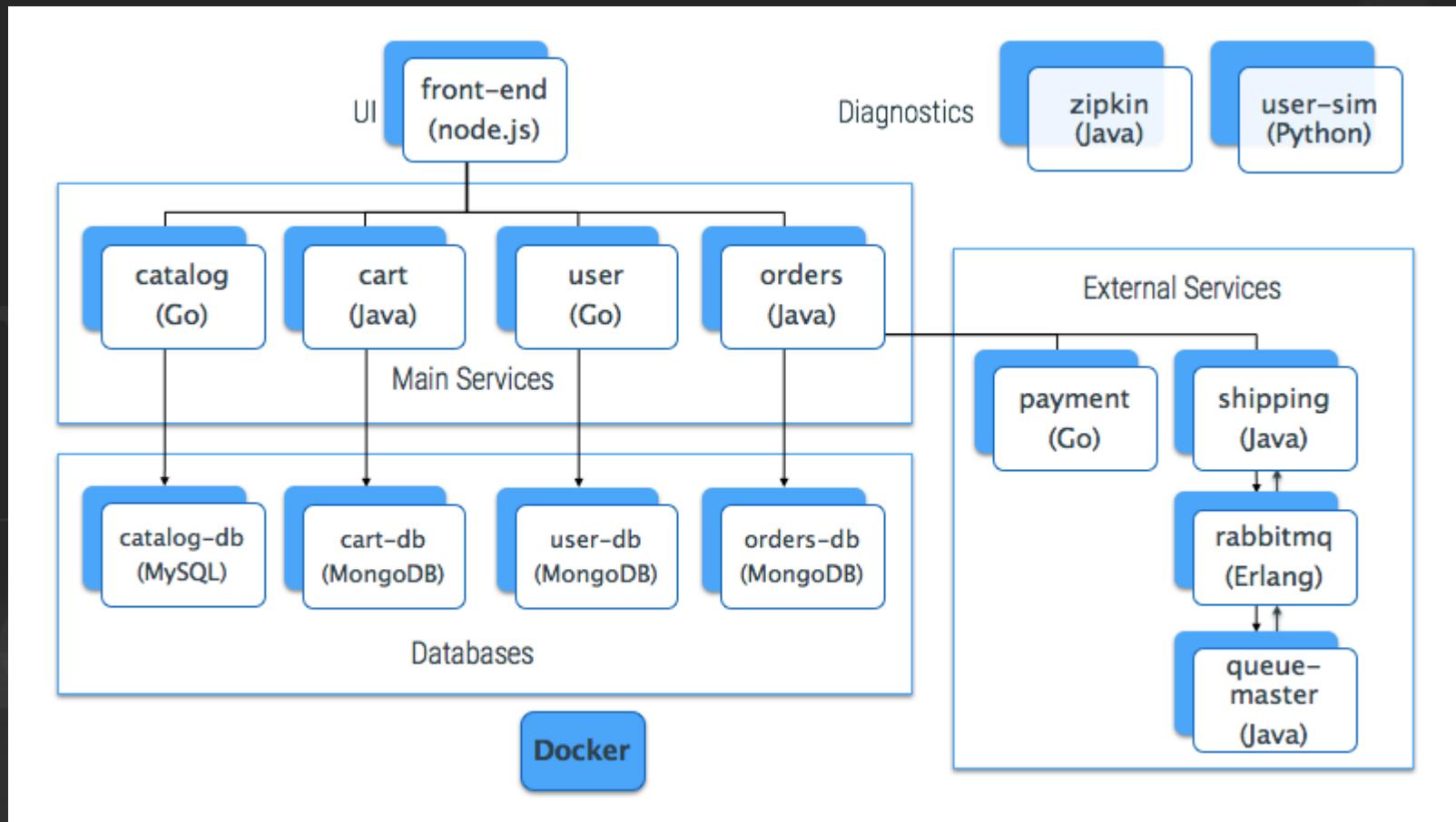
100% SATISFACTION GUARANTEED

Free returns on most items. Hamsters are non-returnable once spoken to.

<https://github.com/microservices-demo>

마이크로서비스 - 데모

- 다양한 언어를 혼합하여 사용한 마이크로 서비스



Opentrace(Zipkin)을 통한 마이크로 서비스 모니터링



Zipkin Investigate system behavior Find a trace Dependencies Go to trace

Duration: 394.256ms Services: 2 Depth: 3 Total Spans: 12 JSON

payment x2 user x10

Services

| | Duration | Service | Method | Path |
|---------|----------|---------|--------|----------------------|
| - | 1.423ms | user | get | /customers |
| - | 1.394ms | user | get | /users |
| user | 545μ | user | | : users from db |
| user | 837μ | user | | : attributes from db |
| user | 933μ | user | | : get /addresses |
| user | 915μ | user | | : get users |
| user | 885μ | user | | : addresses from db |
| user | 62.408ms | user | get | /cards |
| user | 62.385ms | user | get | /cards |
| user | 62.364ms | user | | : addresses from db |
| payment | 43.160ms | payment | post | /paymentauth |

GRAPH TABLE RESOURCES

CPU Memory

3 FILTERED

Item Containers Application Containers

Containers Running containers Both

Untained Hide Untained

The diagram illustrates the distributed nature of the services. The central yellow node represents a service instance, which interacts with multiple application containers (purple hexagons) running on the same host (khan-dev01). Each application container then connects to specific microservices (purple circles).



마이크로 서비스 동영상 데모

KHAN [apm] 서비스 호출 추적



| Export to Excel | | | | | tid : | <input type="text"/> | <input type="button" value="Search"/> |
|-----------------|-------------|---------------|--------------|-------------------------|-------|----------------------|---------------------------------------|
| Fetch Gab | Fetch Count | External Time | CPU Time(ms) | Start Time | | | |
| 0 | 0 | 0 | 397.59 | 2017-08-28 12:35:54.160 | | | |

Transaction ID로 검색 가능

HTTP 호출이 있을 경우 하위
Transaction의 상세 내역을 추적 가능

| | |
|----------------|--|
| User Agent | Apache-HttpClient/4.3.3 (java 1.5) |
| End Time | 2017-08-28 12:36:07.345 |
| CPU Time (ms) | 111.00 |
| Latency (ms) | 0 |
| Thread ID | 206 |
| Instance ID | server11 |
| Transaction ID | 15e26e9209f-3d970a26d2d10c3bec6c > 15e26e9226c-4075494d9a0ffb441 |

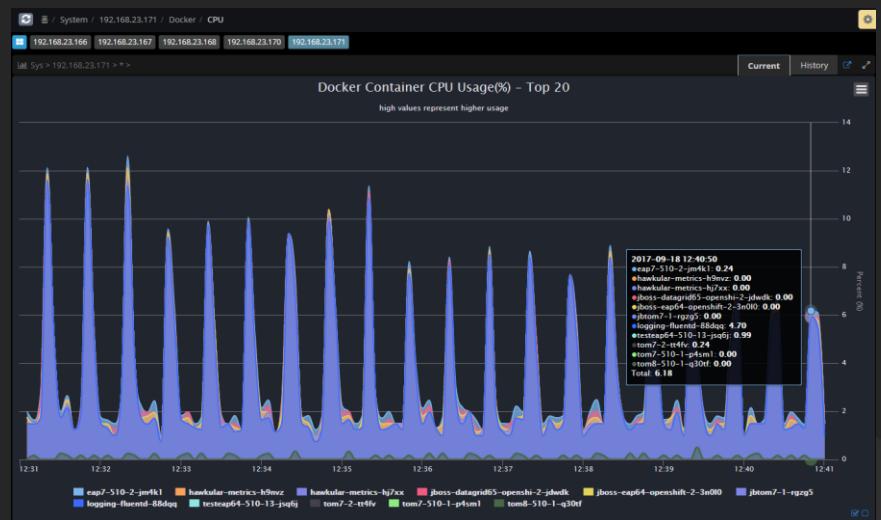
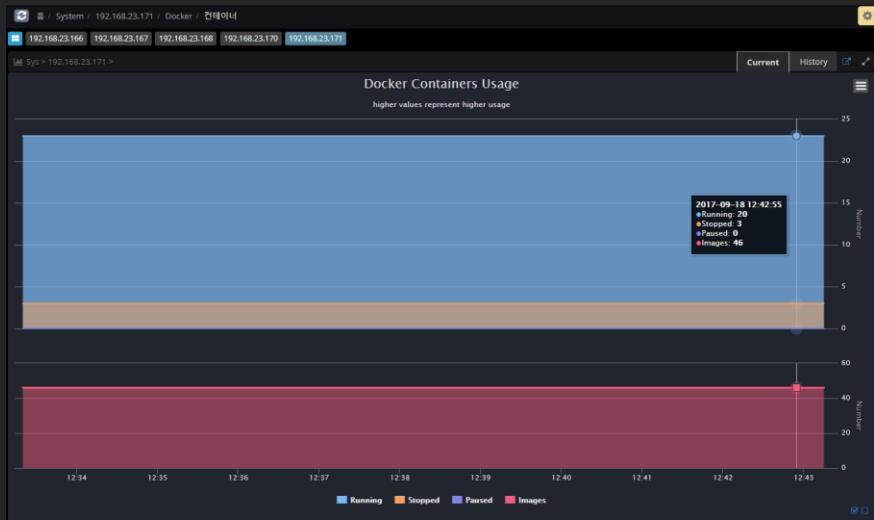
| [Num.] | [Start Time] | [Elapsed] | [%] | [Exclusi] | [A-Gab] | [CPUTime] | Method Call |
|--------|----------------|-------------|-------|-------------|-----------|-------------|--|
| [1] | [12:36:06.047] | 1,445 | 100 | 3 | 0 | 299.0 | + org.apache.catalina.connector.CoyoteAdapter.ser |
| [2] | [12:36:06.049] | 1,442 | 100 | 88 | 1,354 | 297.2 | + org.apache.jasper.servlet.JspServlet.service() |
| [3] | [12:36:06.137] | 1,354 | 94 | 323 | 1,031 | 226.2 | + org.apache.jsp.get4info_jsp._jspService() |
| [4] | [12:36:06.455] | 893 | 62 | 4 | 889 | 50.7 | + org.apache.http.impl.client.CloseableHttpClient.execute() |
| [5] | [12:36:06.459] | 889 | 62 | 1 | 888 | 47.3 | + org.apache.http.impl.client.CloseableHttpClient.execute() |
| [6] | [12:36:06.459] | 888 | 62 | 888 | 0 | 46.8 | + org.apache.http.impl.client.CloseableHttpClient.execute() |
| | | | | | | | > Infos: GET http://192.168.23.11:8180/test/info.jsp?param1=test 200 |
| [7] | [12:36:07.351] | 138 | 10 | 0 | 138 | 1.0 | + org.apache.http.impl.client.CloseableHttpClient.execute() |
| [8] | [12:36:07.351] | 138 | 10 | 138 | 0 | 1.7 | + org.apache.http.impl.client.CloseableHttpClient.execute() |
| | | | | | | | > Infos: GET http://192.168.23.11:8180/test/info.jsp?param1=test 200 |
| | | | | | | | <input checked="" type="checkbox"/> Child Transaction Detail |
| | | | | | | | <input checked="" type="checkbox"/> Child Transaction Detail |

상위 트랜잭션이 있는 경우
클릭하여 상위 트랜잭션 상세 표시

| Transaction for 15e26e9209f-3d970a26d2d10c3bec6c | | | | | | | | | | | | Export to Excel | |
|--|---------------|-------------|---------------------|--|--|--------|--------------|--------------|-----------|-------------|---------------|-----------------|-------------------------|
| Agent Type | IP address | Instance ID | URL | | | Status | Duration(ms) | SQL Time(ms) | Fetch Gab | Fetch Count | External Time | CPU Time(ms) | Start Time |
| | 192.168.23.11 | server11 | /test/get4info.jsp | | | 200 | 1,445 | 0 | 0 | 0 | 1,031 | 298.96 | 2017-08-28 12:36:06.047 |
| | 192.168.23.11 | server11 | ↳ /test/info.jsp | | | 200 | 837 | 0 | 0 | 0 | 693 | 111.00 | 2017-08-28 12:36:06.508 |
| | 192.168.23.11 | server11 | ↳ /test/oratest.jsp | | | 200 | 601 | 376 | 11 | 22 | 0 | 355.77 | 2017-08-28 12:36:06.611 |
| | 192.168.23.11 | server11 | ↳ /test/oratest.jsp | | | 200 | 63 | 12 | 20 | 22 | 0 | 14.71 | 2017-08-28 12:36:07.278 |
| | 192.168.23.11 | server11 | ↳ /test/info.jsp | | | 200 | 135 | 0 | 0 | 0 | 116 | 21.97 | 2017-08-28 12:36:07.353 |
| WAS | 192.168.23.11 | server11 | ↳ /test/oratest.jsp | | | 200 | 36 | 4 | 24 | 22 | 0 | 21.05 | 2017-08-28 12:36:07.383 |
| WAS | 192.168.23.11 | server11 | ↳ /test/oratest.jsp | | | 200 | 29 | 8 | 11 | 22 | 0 | 12.85 | 2017-08-28 12:36:07.454 |

하위 트랜잭션 호출을 Tree 형태로
표시

KHAN [apm] Docker 모니터링(CPU, Mem, Info, Containers)

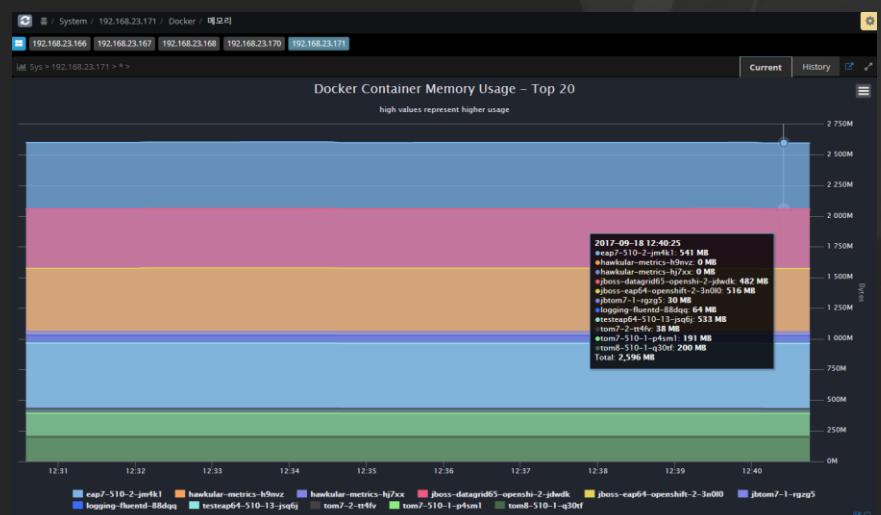


Docker Info / Docker 정보

Containers

```

Docker >
  Docker Info Images Containers
  Docker Info [View]
  # Container ... Names kds_name
  1 c5031662_kds_eap7-510_...
  2 08e0ea00_kds_POD-6fbd_POD
  3 569d43ee7b_kds_tom7-510_tom7-510
  4 6d3a1271b_kds_tom7-510_tom7-510
  5 3d94a3d16_kds_tom7-510_tom7-510
  6 b0c1e930_kds_jboss-datasagrid5-...
  7 3d3a009e_kds_boss-dae_jboss-datasagrid...
  8 4715c6597a_kds_te...
  9 9e4754fe_kds_jboss-ea_jboss-eap64-510...
  10 b40f09e40_kds_POD-6fbd_POD
  11 3547r3294_kds_POD-6fbd_POD
  12 6ade43a80_kds_POD-6fbd_POD
  13 02e206fd8_kds_POD-6fbd_POD
  14 b19de677a_kds_POD-6fbd_POD
  15 e0991763_kds_POD-6fbd_POD
  16 c50130cd_kds_POD-6fbd_POD
  17 e09e9005b_kds_hawskular_hawskular-metr...
  18 cd573b46_kds_hawskular_hawskular-metr...
  19 56a5e698_kds_fluentd-e...
  20 ac7d0be7c_kds_POD-6fbd_POD
  ...
  Record ID: 8 1-20 of 20
  
```



マイクロ 서비스로 향하는 길

- 모놀리스가 만능 해결책은 아니다
 - 기존의 모놀리스로도 충분한 경우, 초기 서비스 단계인 경우
- 모놀리스를 단계적으로 다수의 마이크로 서비스로 전환하기
 - 기존 시스템에서 분할이 가능한 항목 또는 신규 기능을 마이크로 서비스로 분리하여 추가하기
- 의존성 낮추기
 - 독립적인 배포가 가능하도록 서비스를 분리하기
- 조직 문화의 변화 수용
 - 마이크로 서비스는 만능 해결책이 아니므로, 구성원들의 skill-up에 대한 관심과 투자가 뒷받침 되어야 함

Application Performance Management

감사합니다.

KHAN
[a p m]
[s b w]
[M A E]



KHAN
[a p m]

제품이나 서비스에 관한 문의

콜 센터 : 02-469-5426 (휴대폰 : 010-2243-3394)

전자 메일 : sales@opennaru.com