

클라우드 네이티브를 위한  
SaaS 형 OPENMARU APM

# 공공 클라우드 전환 로드맵 손질...



클라우드 네이티브 부처별 추진·2030년 완료

## 국내 클라우드 정책(1차 기본계획)

2021년 클라우드 선도국가 도약을 위한 발걸음



### 「제1차 클라우드컴퓨팅발전 기본계획」 3대 추진전략 + 9대 정책과제

#### 공공부문의 선제적인 클라우드 도입

- 1 공공부문의 민간 클라우드 이용 활성화
- 2 지능정부 구현을 위한 정부 클라우드 전환 가속화

#### 민간부문 클라우드 이용 확산

- 3 안전한 클라우드 이용 환경 마련
- 4 클라우드 친화적인 제도개선
- 5 중소기업 및 산업의 혁신지원

#### 클라우드 산업성장 생태계 조성

- 6 클라우드 기술 경쟁력 강화
- 7 클라우드 서비스 해외진출 촉진
- 8 클라우드 전문 인력 양성
- 9 클라우드 데이터센터 경쟁력 강화

1차 클라우드 컴퓨팅 발전 기본계획 당시 정부의 클라우드 방향성(출처: NIA)

## 클라우드 네이티브 전환 로드맵 수립 설명회

- 행정안전부, 2023년 5월 3일

### 공공 클라우드 네이티브 전환 로드맵

추진 배경	<ul style="list-style-type: none"> <li>• 정부 재정투자 방향 변화</li> <li>• 보안인증제 개편</li> <li>• 신기술 보편화 등</li> </ul>
추진 환경	<ul style="list-style-type: none"> <li>• 행정안전부 전환 사업 예산 축소</li> <li>• 정부 정책 "민간 클라우드 우선 이용"</li> <li>• "클라우드 네이티브 우선 적용"으로 발전 등</li> </ul>
추진 방향	<ul style="list-style-type: none"> <li>• 범정부 정보자원 등록 및 관리시스템에 등록된 모든 시스템의 클라우드 네이티브 전환</li> </ul>
전환기간	<ul style="list-style-type: none"> <li>• 2024년부터 2030년까지 7개년 추진</li> </ul>
추진 목표	<ul style="list-style-type: none"> <li>• 2023년 시범 사업</li> <li>• 2024년 10% 달성</li> <li>• 2025년 30% 달성</li> <li>• 2026년 이후 70% 달성</li> </ul>

# 디지털플랫폼정부는 '클라우드 네이티브'로 구축하는 게 핵심이다



➡ 단순 클라우드 전환이 아닌, 클라우드 네이티브 전환 부터 다시 작성해야 함

**디지털플랫폼정부는** 초거대AI를 비롯한 디지털 기술을 적극 도입, 칸막이를 없애고 '원팀 정부'로 거듭나는 것을 목표로한다. 이를 위한 **공공 플랫폼을 단순 클라우드 전환이 아닌, 클라우드를 클라우드답게 쓸 수 있도록 '클라우드 네이티브'로 구축하는 게 이번 정책의 핵심**이다. 공공SW(소프트웨어) 사업 고질병을 극복할 해법을 제시하면서 각종 미래 IT 산업 육성의 요람이 될 것으로 기대된다.

전자신문 etnews Conference allshowTV Engli

경제·금융 전자·모빌리티 통신·미디어·게임 소재·부품 SW·보안 산업·에너지·환경

## [기획]행안부, 클라우드 네이티브 확산 위한 지원사업 추진

발행일: 2022-06-29 15:00 | 지역: 2022-06-30 | 18면

**클라우드 성숙도 단계**

- Level 3: Cloud Native (클라우드 네이티브 단계)**
  - 마이크 서비스 구조 사용 범위 확대
  - API 기반의 소프트웨어 아키텍처
- Level 2: Cloud Resilient (클라우드 탄력 단계)**
  - 장애를 고치는 데 시간이 걸리지 않는 운영 체제 기반의 운영 체제 서비스 제공
  - 장애 발생 시 복구 시간 최소화 - 가용성 99.99% 이상을 추구
  - 운영 집중화된 모니터링 확보
  - 재용역/재가용성 클라우드를 지원하는 스택의 전략
- Level 1: Cloud Friendly (클라우드 친화 단계)**
  - 단순한 시스템 운영 구조
  - 이종으로 하이브리드
  - 12-Factors App 원칙 준수
- Level 0: Cloud Ready (클라우드 준비 단계)**
  - 내부 컴퓨팅이 없는 환경
  - 서비스 기반
  - 컨테이너화 및 가상화 기반 운영

클라우드 네이티브는 클라우드 성숙도 단계 중 최고 단계로, 클라우드의 기능과 장점을 최대한 활용해 애플리케이션을 구축·실행하는 것을 의미한다.

## [테크&포커스] 정부·공공 시스템도 '클라우드 최첨단'... '디플정'의 도전

입력: 2023-04-16 16:07 | 평용현 기자

📄 📱 📺 📷

🔍 🔖 N 기사

尹정부 핵심과제 청사진 공개  
민첩한 개발·유연한 확장 가능  
초거대 AI 등 디지털기술 도입  
홈택스 등 사이트 통합도 추진

**디지털플랫폼정부 기대 효과(2026년)**

<b>대국민 서비스 강화</b> <ul style="list-style-type: none"> <li>• 공공서비스 1500여종 연계·통합</li> <li>• 혜택 알리미 총 1021종 제공</li> <li>• 정부서류제로화로 연 2조원 절감</li> </ul>	<b>민·관 성장 플랫폼 구축</b> <ul style="list-style-type: none"> <li>• SaaS기업 1만개 육성</li> <li>• AI 유니콘 기업 5개 육성</li> <li>• DPG 수출 연 20억달러 달성</li> </ul>
<b>정부·공공 시스템 혁신</b> <ul style="list-style-type: none"> <li>• 공공부문 종이 사용량 50% 감축</li> <li>• 대상 시스템 70% 클라우드 네이티브 전환</li> <li>• 광역·기초로 이원화된 지자체 시스템 통합</li> </ul>	<b>사이버보안·개인정보보호</b> <ul style="list-style-type: none"> <li>• 주요 분야 사이버데이터 융합체계 구축</li> <li>• 제로트러스트 등 새로운 보안체계 도입</li> <li>• 신기술 공공 적용, 보안산업 경쟁력 강화</li> </ul>

**모놀리식 아키텍처와 마이크로서비스 아키텍처(MSA) 차이**

모놀리식 아키텍처 VS 마이크로서비스 아키텍처

\*실용 단계 우호적인 모놀리식 아키텍처와 달리 MSA는 각 서비스가 병렬·분산돼 유연성과 가용성이 뛰어나다.

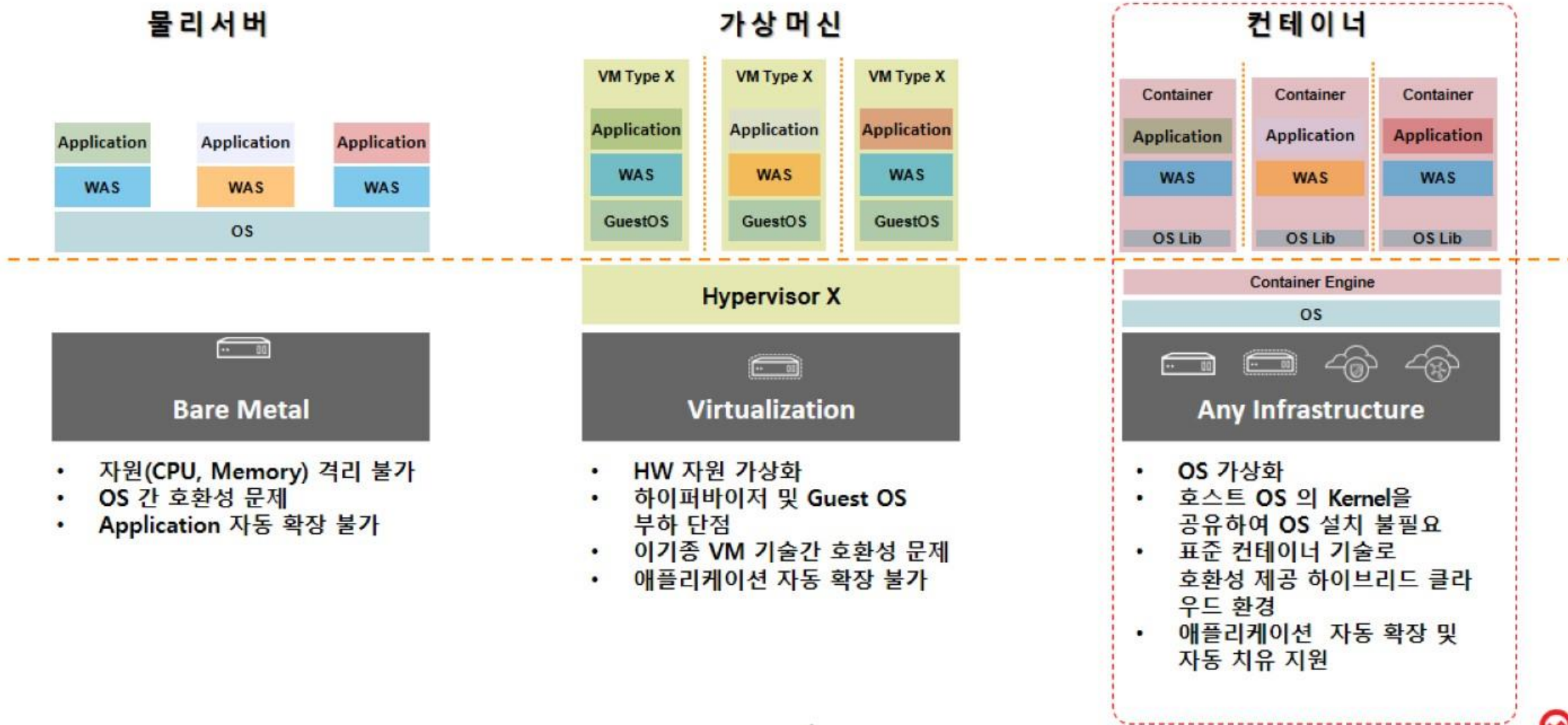
Application Performance Management



Container

# 컨테이너 차별성

➡ 자원 효율성, 자원 격리, 호환성, Auto Scaling, DevOps, MSA, 관리 편의성



# VM 의 고질적인 문제점



가상화 기술은 Guest OS 로 인하여 VM 간 불일치와 느린 시작시간과 이미지가 비대함

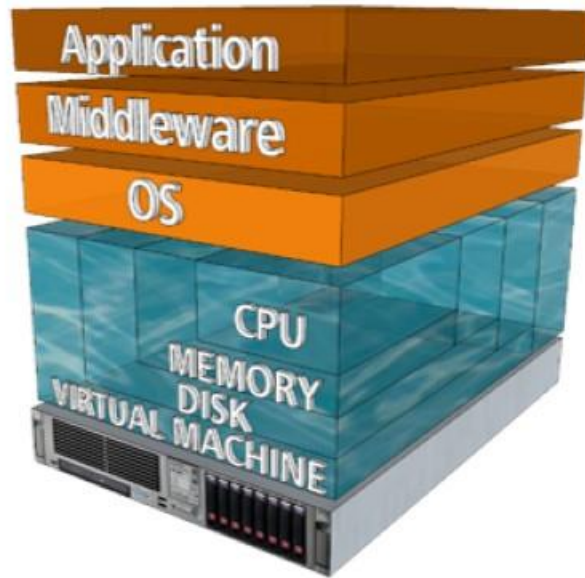
## 거대한 이미지 사이즈

- VM 을 템플릿 관리는 하지만  
사이즈가 커서 재사용성을  
높이기 어려움



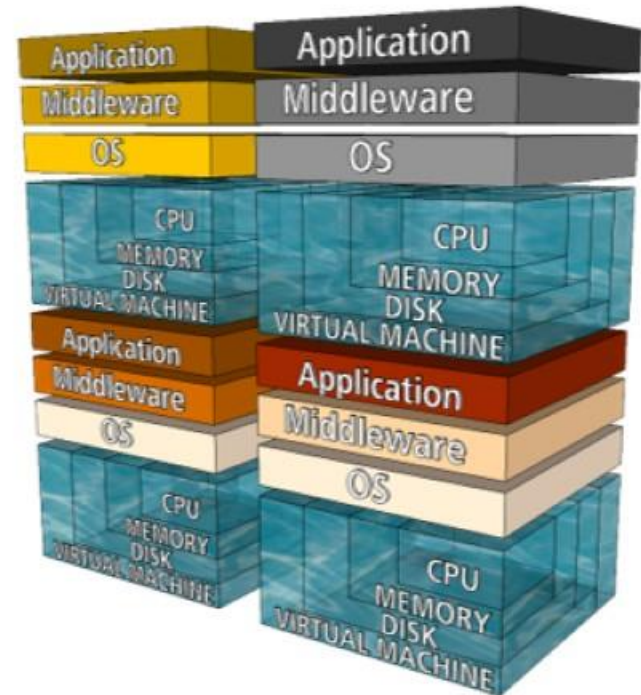
## 느린 시작 시간

- 부팅 시 Hypervisor - OS- 미들  
웨어 - 애플리케이션까지  
실행 되어야함



## VM 간의 환경 불일치

- VM 생성 후 개별로 변경 사항을  
관리하기 때문에 VM 간 구성이  
나 환경이 불일치



# 오버헤드 - Containers vs. VMs

➡ Guest OS 의 부팅으로 인하여 시간이 지연됨

- OS에서 응용 프로그램을 작동하는 경우, 하드웨어 가상화에서는 **가상화 된 하드웨어 및 하이퍼바이저**를 통해 처리하기 때문에 물리적 시스템보다 처리에 부가적인 시간 (오버 헤드)가 필요
- 컨테이너 형 가상화 커널을 공유하고 **개별 프로세스가 작업**을 하는 것과 같은 정도의 시간 밖에 걸리지 않기 때문에 대부분 오버 헤드가 없음

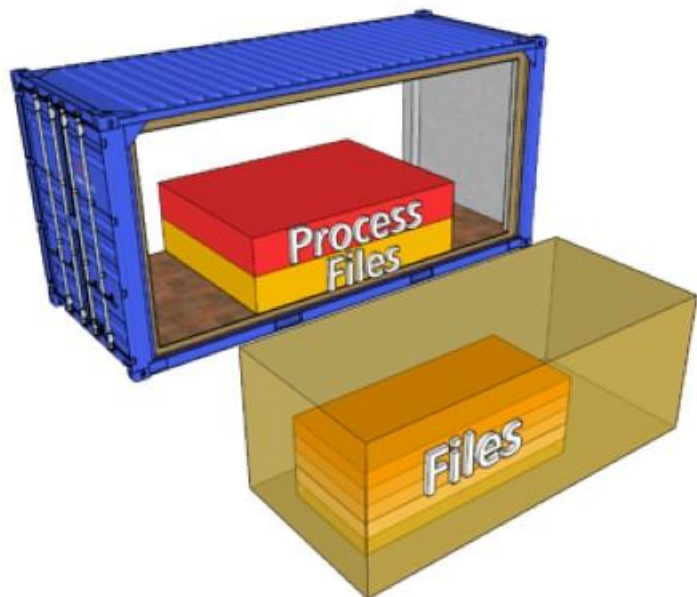


# 컨테이너를 통한 가상서버 문제점 해결

➡ 컨테이너 기술은 하이브리드 클라우드 와 DevOps 그리고 MSA 구현을 필수

## 작은 이미지 사이즈

- 컨테이너는 레이어 개념으로 이미지에 파일을 추가/삭제하여 관리함
- 레이어 사이즈를 최적화하여 이미지 사이즈를 최소화



## 빠른 시작 시간

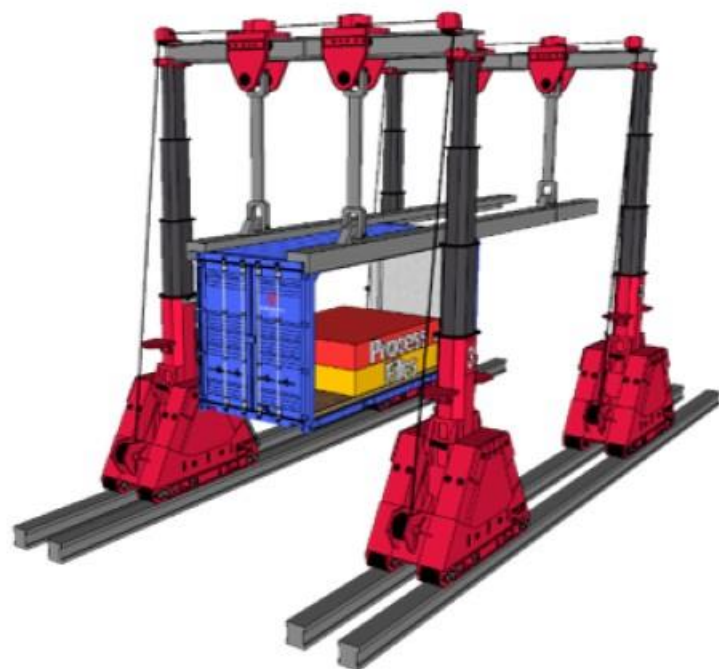
- 컨테이너는 분리된 프로세스 형식으로 OS 부팅이 필요 없기 때문에 부팅 시간을 최소화 할 수 있음



Container = Process

## 높은 이동성(Portability)

- 애플리케이션에 필요한 라이브러리나 의존 파일들을 이미지에 포함하기 때문에 환경에 의한 발생하는 문제가 거의 없음





# 가상화와 컨테이너 집적도 비교

동일한 H/W 에서 가상화 대비 컨테이너 집적도가 수 배 이상 높음

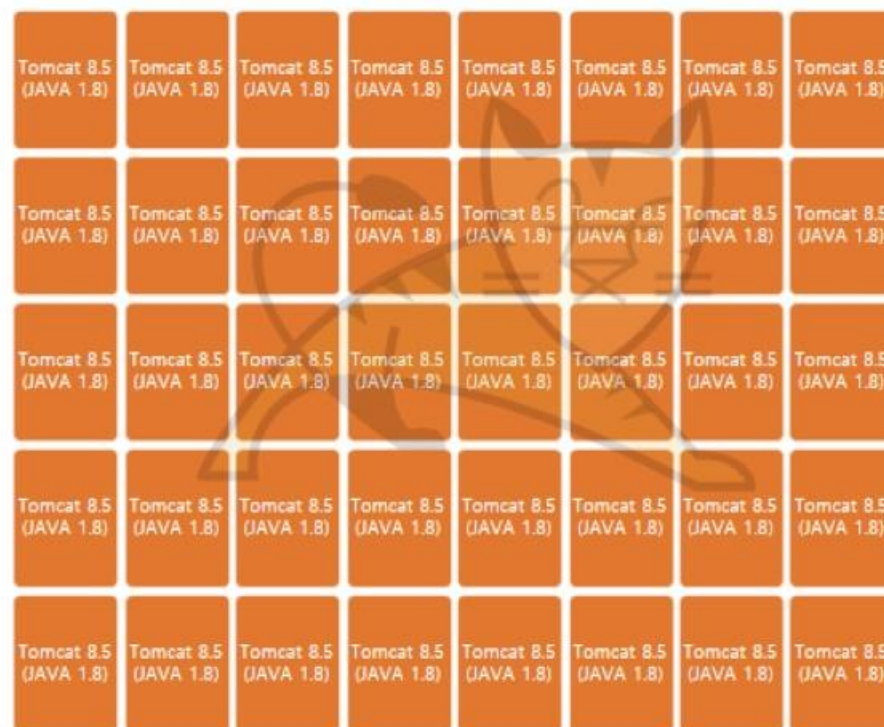
## 총 16개 톰캣 인스턴스 (가상화)



Red Hat Virtualization Red Hat Virtualization 4.4

Red Hat Enterprise Linux Red Hat Enterprise Linux 7.7 (4Core / 16GB)

## 총 40개 톰캣 인스턴스 (컨테이너)



Container Runtime

Red Hat Enterprise Linux Red Hat Enterprise Linux 7.7 (4Core / 16GB)



Application Performance Management

Cloud Native Computing



# 클라우드 네이티브를 저해하는 요인



➡ 생각과 시스템을 클라우드 네이티브 하게 전환하지 못하면, 클라우드라도 개선이 없음



## 클라우드를 임대하여 사용하는 것일 뿐

- 가상머신과 스토리지를 임대하여 사용하고 있을 뿐, 기존의 인프라와 다르지 않음



## 클라우드 특징에 맞게 설계하고 운영 하지 않음

- 클라우드 특성을 이해하지 못하고 기존 인프라를 단순히 대체하여 설계
- 비용 부분에서만 정액제 클라우드로 전환하였으나, 벤더 종속성과 비용만 높아짐



## 인프라만 클라우드 일 뿐 조직은 그대로

- 기존의 개발팀과 운영팀이 수행하던 역할과 프로세스 그대로 운영



## 클라우드로 전환했으나 구인난과 고비용 구조로 더 큰 문제

- 클라우드를 이용하고 있음에도 불구하고 수작업 프로세스에서 벗어나지 못함
- 운영 인력 부족과 업무 효율성을 개선하지 못함

# 클라우드 네이티브 시대



➡ 가상화 시대를 넘어 클라우드 네이티브 시대로

- 애플리케이션을 실행하기 위한 최적의 인프라 최적 솔루션 중 하나
  - 컨테이너 기술
    - 애플리케이션이 동작하기 위한 운영 환경을 함께 패키징
    - 개발자를 위한 이미지 빌드/배포 용이성
    - 빠른 애플리케이션 실행과 낮은 오버헤드
- Kubernetes(컨테이너 오케스트레이션)
  - 애플리케이션 실행을 먼저 생각할 때
  - 어떤 인프라를 만들지 주축으로 설계된 인프라 기반

Developer Experience 장점

Reconciliation model의 정교함

2016년 ~  
클라우드 네이티브

~2000년  
물리서버

2001~2009년  
가상화 기술 1세대

2010~2015년  
가상화 기술 2세대

# Cloud Native 에서 Native 는?



클라우드 네이티브는 "클라우드가 '클라우드 다울 수 있도록' 애플리케이션을 구축, 실행하는 방식"

네이티브(Native)의 사전적 의미는 '선천적인', '본래' 등이다.

'어린이 또는 성인이 되어 언어를 배운 것이 아닌 태어나서 부터 특정 언어를 사용해 온 사람'

## 네이티브 스피커

Hello      Selamat Datang      Kumusta



American      Indonesian      Filipinos

'어린이 또는 성인이 되어 스마트폰을 접한 것이 아닌 유아기부터 스마트폰을 사용해 온 사람'

## 스마트폰 네이티브



'애플리케이션을 계획/설계할 때부터 클라우드 특징과 장점을 기반으로 개발/운영'

## 클라우드 네이티브



# Cloud Native vs. Cloud Immigrant



가상화 기반 IaaS vs. 컨테이너 기반 PaaS



구분	Cloud Immigrant	Cloud Native
서비스 모델	가상화 기반 IaaS (Infrastructure As A Service)	컨테이너 기반 PaaS (Platform As A Service)
디자인	On Premise 에 구축된 시스템을 클라우드로 이전하여 운영	시작 단계부터 클라우드의 장점인 민첩성, 확장성 그리고 이동성을 최대한 활용할 수 있도록 설계
구현	특정 클라우드 벤더에 의존적인 설정이 있어 구축에 시간이 걸림	어떤 클라우드 환경에서도 빠르고 효율적으로 전환 (Portability)
확장성	애플리케이션 업데이트가 수작업이기 때문에 장시간의 다운타임일 필요하고 Scale In/Out 이 어려움	컨테이너와 MSA 기반으로 서비스에 영향을 주지 않고, 업데이트가 필요한 서비스만 변경할 수 있으며, 서비스 단위의 Scale In/out 지원
비용	애플리케이션이 커질 수록 인프라 비용이 상승	인프라 부분의 종속성이 없어 비용이 저렴
유지보수	버전관리, 설치 그리고 구성관리가 수작업이며 복잡함	CI (Continuous Integration) / CD (Continuous Delivery)

# CNCF Cloud Native Definition v1.0



클라우드 네이티브 선언문 1.0 - 클라우드 네이티브 컴퓨팅 재단

클라우드 네이티브 기술을 사용하는 조직은 현대적인 퍼블릭, 프라이빗, 그리고 하이브리드 클라우드와 같이 동적인 환경에서 확장성 있는 애플리케이션을 만들고 운영할 수 있다.

컨테이너, 서비스 메시, 마이크로서비스, 불변의 인프라스트럭처, 그리고 선언적 API가 전형적인 접근 방식에 해당한다.

이 기술은 회복성이 있고, 관리 편의성을 제공하며, 가시성을 갖는 느슨하게 결합된 시스템을 가능하게 한다.

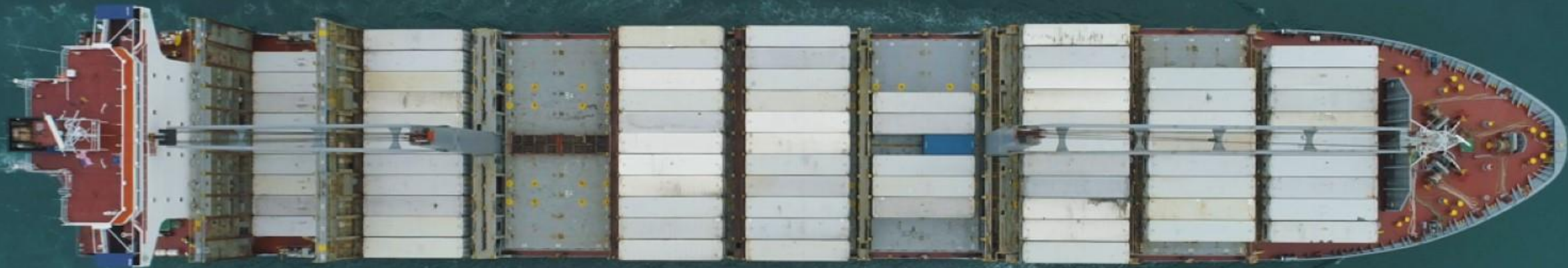
견고한 자동화와 함께 사용하면, 엔지니어는 영향이 큰 변경을 최소한의 노력으로 자주, 예측 가능하게 수행할 수 있다.

Cloud Native Computing Foundation은 **벤더 중립적인 오픈소스 프로젝트 생태계**를 육성하고 유지함으로써 해당 패러다임 채택을 촉진한다.

우리 재단은 최신 기술 수준의 패턴을 대중화하여 이런 혁신을 누구나 접근 가능하도록 한다.



Cloud Native

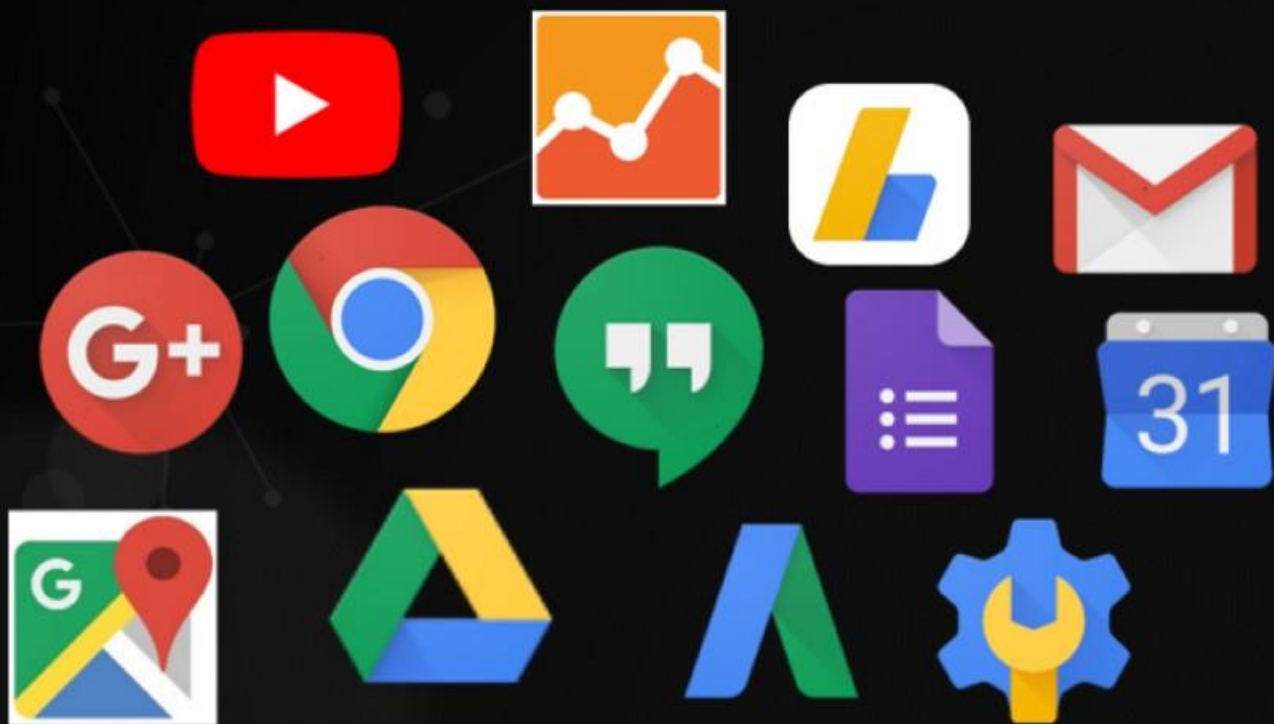


Kubernetes



# Google 의 모든 서비스는 **컨테이너** 에서 실행

- Gmail , 검색, 지도 ...
- MapReduce , GFS , Colossus ...
- Google Compute Engine 가상 머신도 **컨테이너** 에서 실행!
- 매주 20 억개 이상의 **컨테이너** 를 실행 중



# GOOGLE 과 컨테이너

Google의 업무 방식

Gmail에서 YouTube, 검색에 이르기까지 Google의 모든 제품은 컨테이너에서 실행됩니다.

개발팀은 컨테이너화를 통해 더욱 신속하게 움직이고, 효율적으로 소프트웨어를 배포하며 전례 없는 수준의 확장성을 확보할 수 있게 되었습니다. Google은 매주 수십억 개가 넘는 컨테이너를 생성합니다. 지난 10여 년간 프로덕션 환경에서 컨테이너화된 워크로드를 실행하는 방법에 관해 많은 경험을 쌓으면서 Google은 커뮤니티에 계속 이 지식을 공유해 왔습니다.

초창기에 cgroup 기능을 Linux 커널에 제공한 것부터 내부 도구의 설계 소스를 Kubernetes 프로젝트로 공개한 것까지 공유의 사례는 다양합니다. 그리고 이 전문 지식을 Google Cloud Platform으로 구현하여 개발자와 크고 작은 규모의 회사가 최신의 컨테이너 혁신 기술을 쉽게 활용할 수 있도록 하였습니다.





## About Kubernetes

- 쿠버네티스(K8s)는 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리해주는 오픈소스 소프트웨어
- 쿠버네티스", "쿠베르네티스", "K8s", "쿠베", "쿠버", "큐브"라고 부르며
- Go로 작성된 오픈 소스 , 오픈소스 S/W (Apache License 2.0) 라이선스
- 리눅스 재단 (Linux Foundation )산하 Cloud Native Computing Foundation (CNCF) 에서 관리
- 구글에서 개발하고 설계한 플랫폼으로서 사내에서 이용하던 컨테이너 클러스터 관리 도구인 "Borg"의 아이디어를 바탕으로 개발

"Kubernetes is open source-a contrast to Borg and Omega, which were developed as purely Google-internal systems. "

- Borg, Omega, and Kubernetes





“누군가가 나의 등잔의  
심지에서 불을 붙여가도  
내 등잔의 불은 여전히  
빛나고 있습니다.”

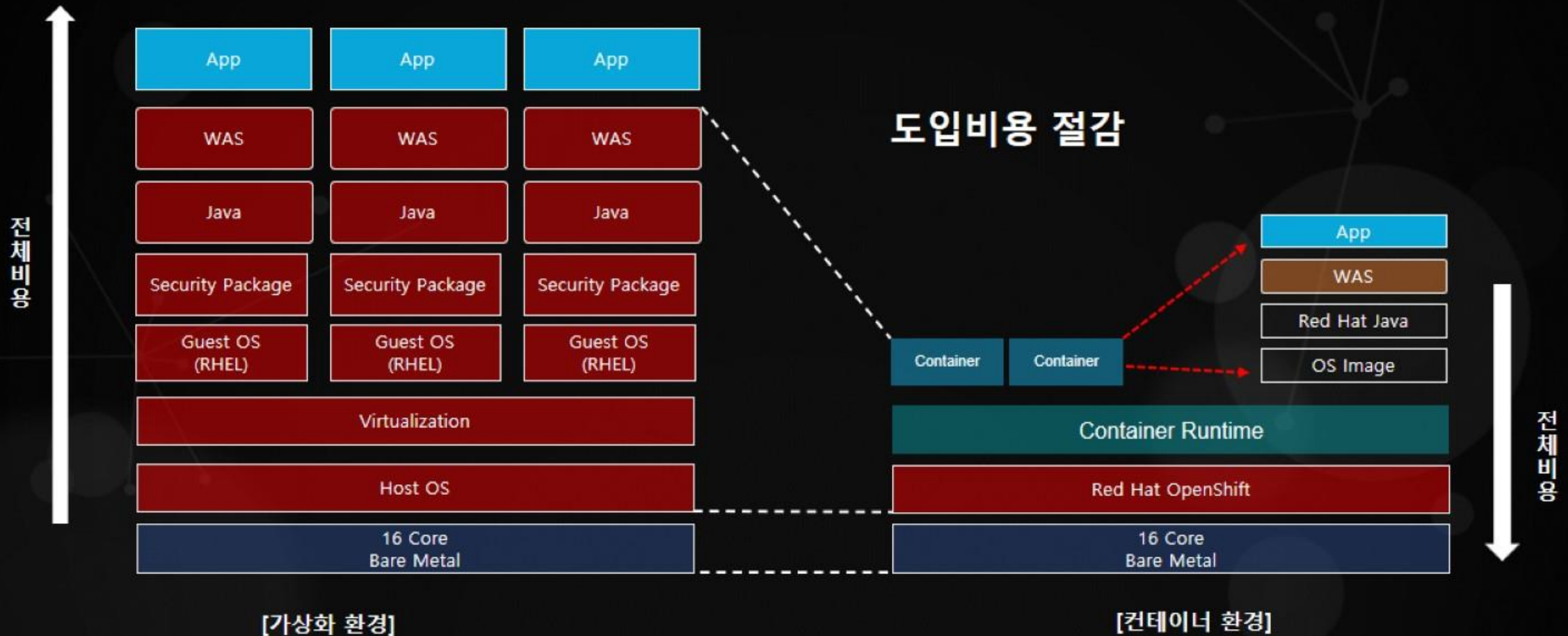
*미국의 정치가 토머스 제퍼슨*

컨테이너 기술의 차이점

# 인프라 운영의 변화

# 가상화 VS 컨테이너 비교 - 비용적인 측면

- 가상화 대비 Guest OS 유지보수, 라이선스, 관리비용 제거
- 서버 접근제어를 비롯한 보안 솔루션 제거



# 아직도 WAS BMT 나 POC를 하시나요?

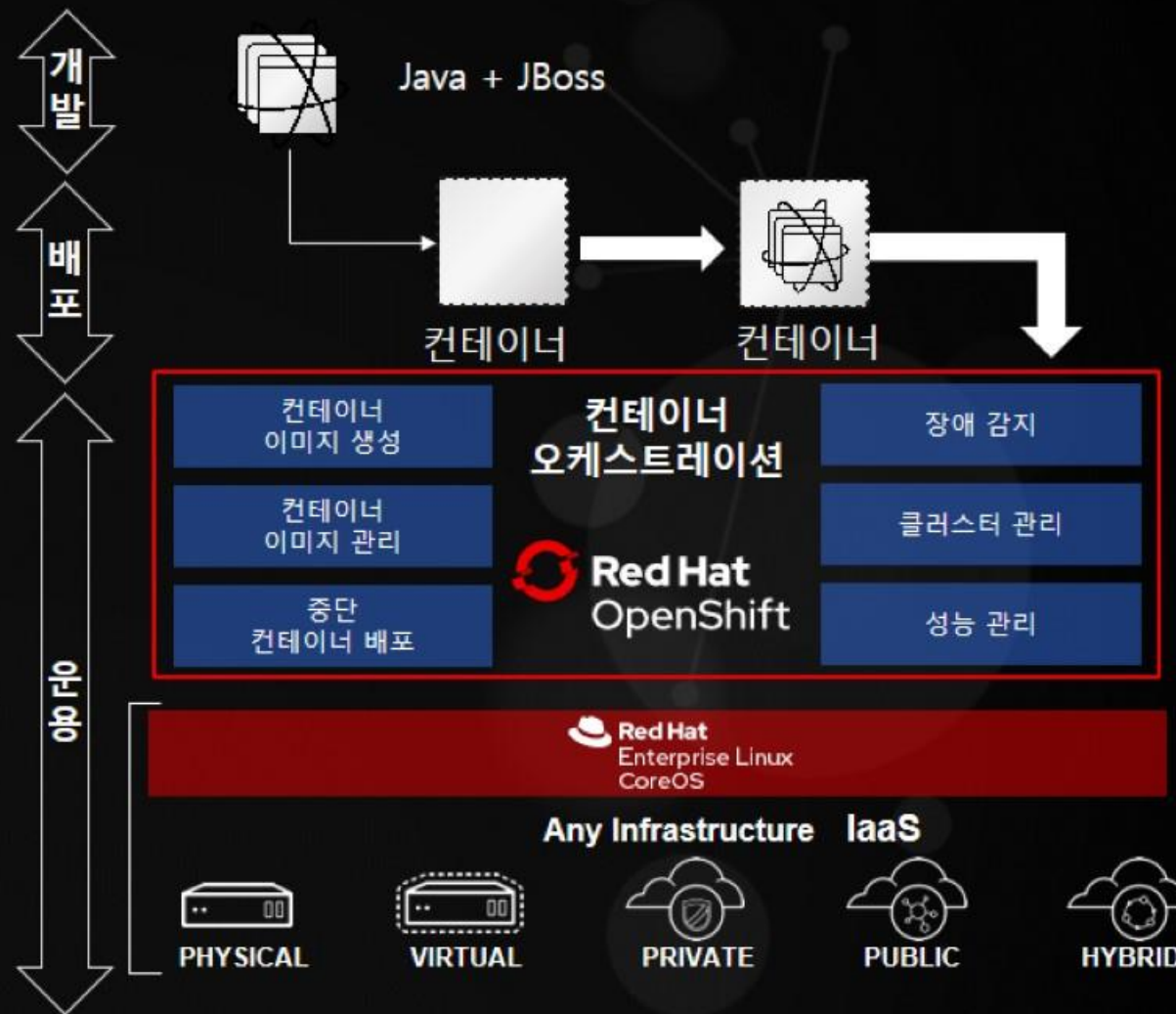
- WAS 의 가용성/확장성/성능/신뢰성 등의 기능은 플랫폼으로 이전
- 더 가볍고 더 빠르고, 자동화에 친화적인 WAS 로 전환

## WAS 제품 BMT



## RASP 에 의한 WAS평가

- 신뢰성 ( Reliability ):
  - 과부하 테스트
- 가용성 ( Availability ):
  - 일부 장애 발생시 전체 시스템 영향 최소화
- 확장성 ( Scalability ):
  - 자원 추가에 따른 선형적인 성능 개선
- 성능 ( Performance ):
  - TPS, 응답시간 등 평가
- 보안 기능 ( Security )
  - (RBAC 등)
- WAS 도구 평가 ( Manageability )
  - Admin Console



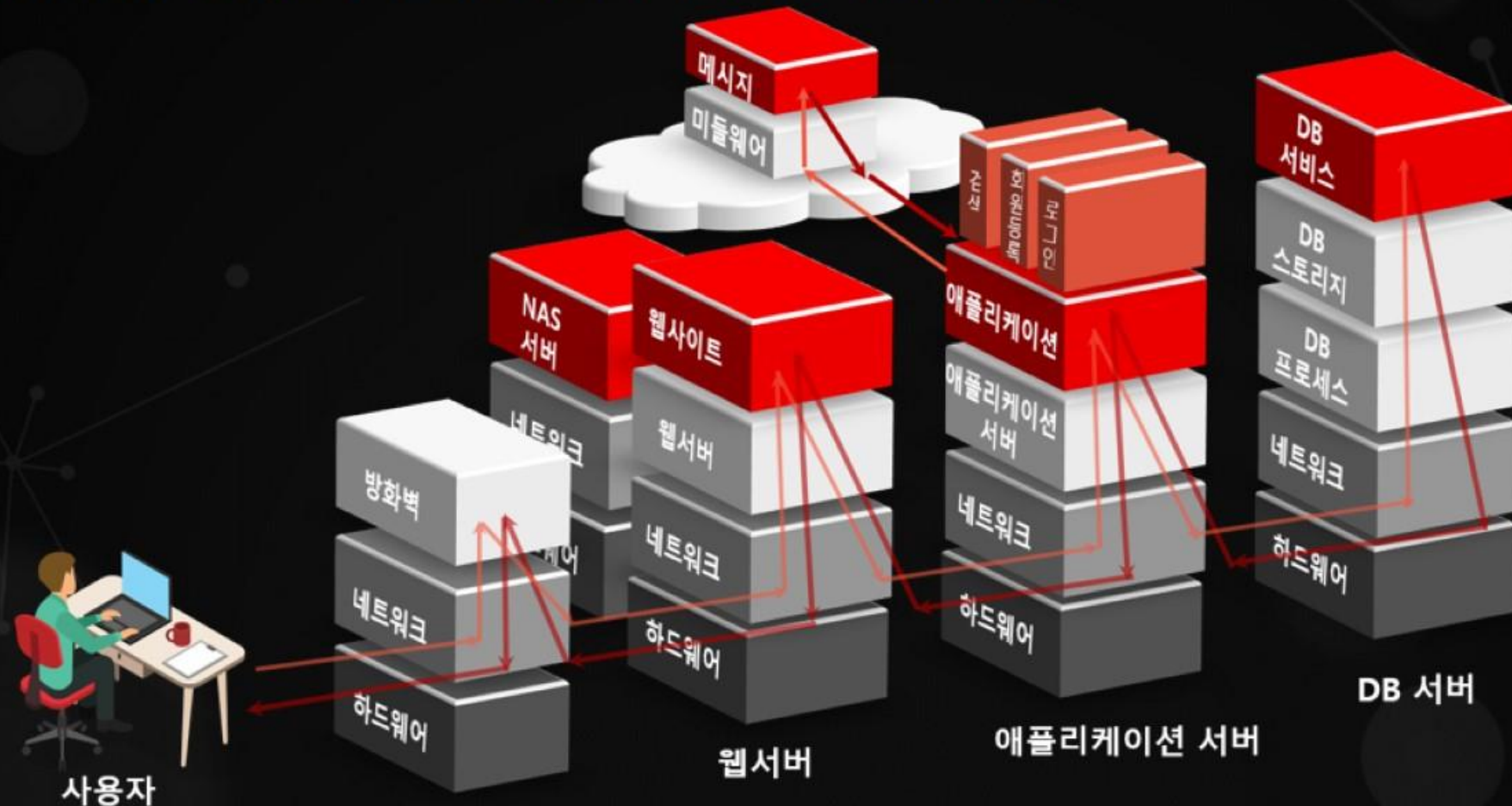


**APM**



## 미들웨어는 시스템 장애의 관문이자 시작점

- 데이터베이스가 50% 느려진다면 사용자 응답시간은 어떻게 될까요?
- DB 테이블 변경으로 SQL 에서 오류로 인하여 페이지가 오류가 난다면?



# [긴급] 비정상 상황 발생 - 대시보드



[심각-CRITICAL] 'Worker Usage %' (평균값: 98.9)이 심각(CRITICAL) 임계값 '95'을 넘었습니다.  
 발생에이전트 : apache@EBS-OC-PROD2-WEB05[172.17.11.35]

클릭하여 상세한 정보를 확인하세요... 10s



[심각-CRITICAL] 'Worker Usage %' (평균값: 99.95)이 심각(CRITICAL) 임계값 '95'을 넘었습니다.  
 발생에이전트 : apache@EBS-OC-PROD2-WEB02[172.17.11.5]

클릭하여 상세한 정보를 확인하세요... 10s

Request Viewer



Request Velocity



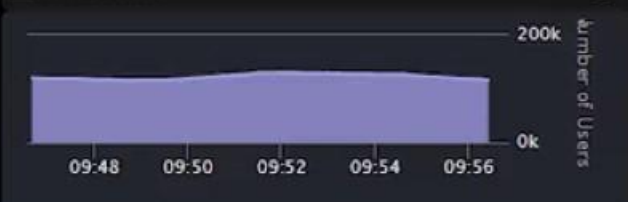
APDEX



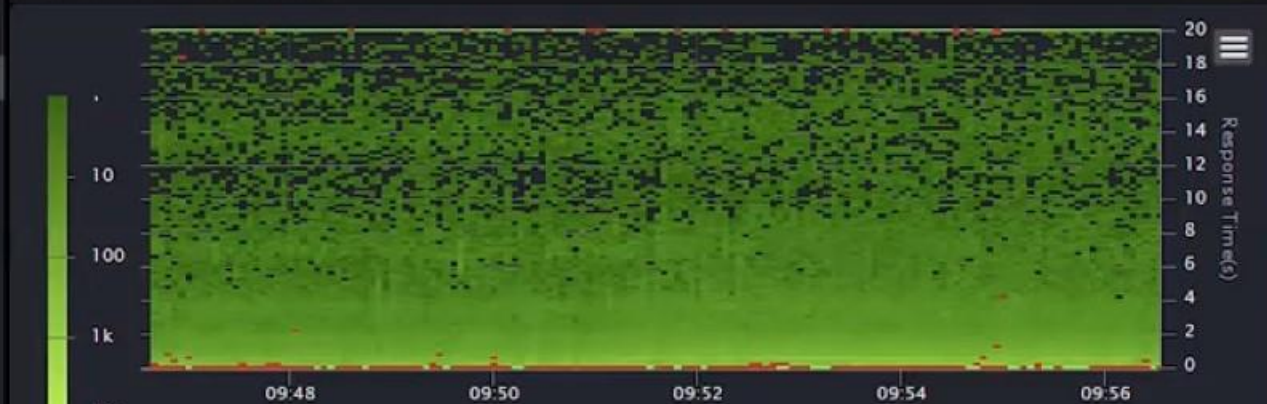
TPS



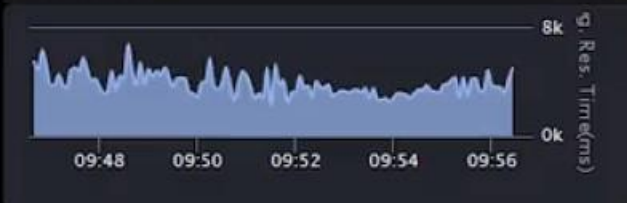
Active Users



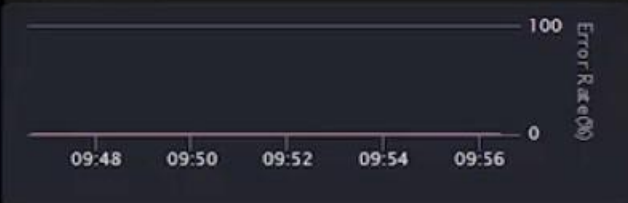
Transaction Heatmap (T-Map)



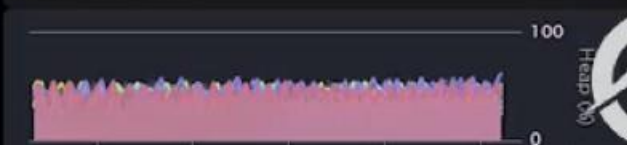
Avg. Res. Time



Error Rates

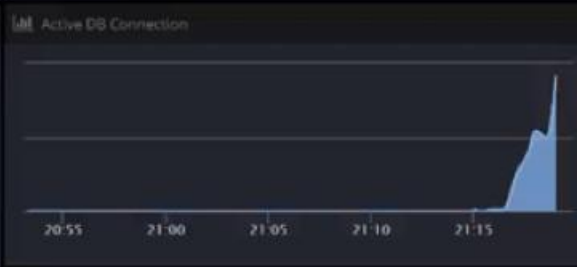
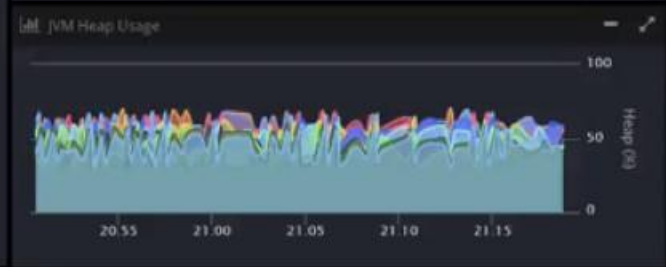
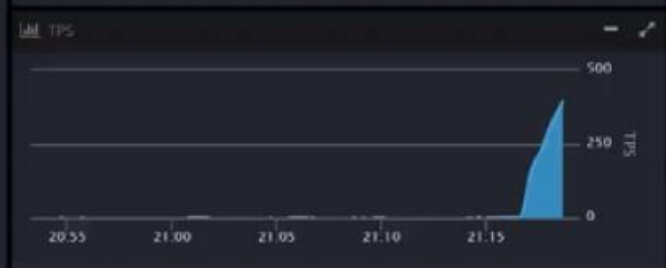
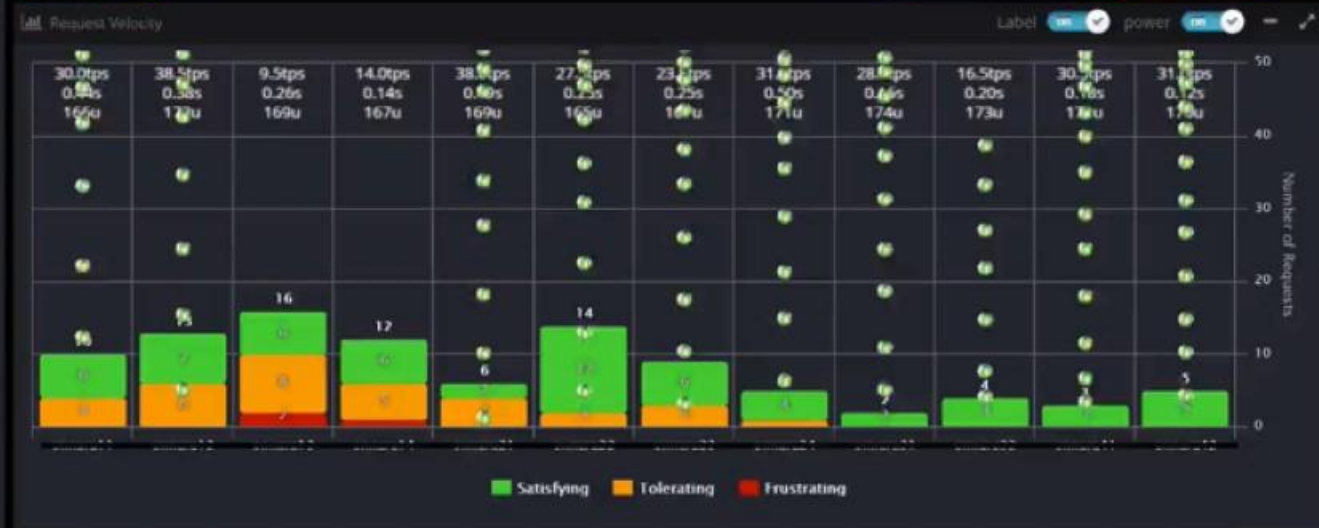


JVM Heap Usage



Active DB Connection







Application Performance Management

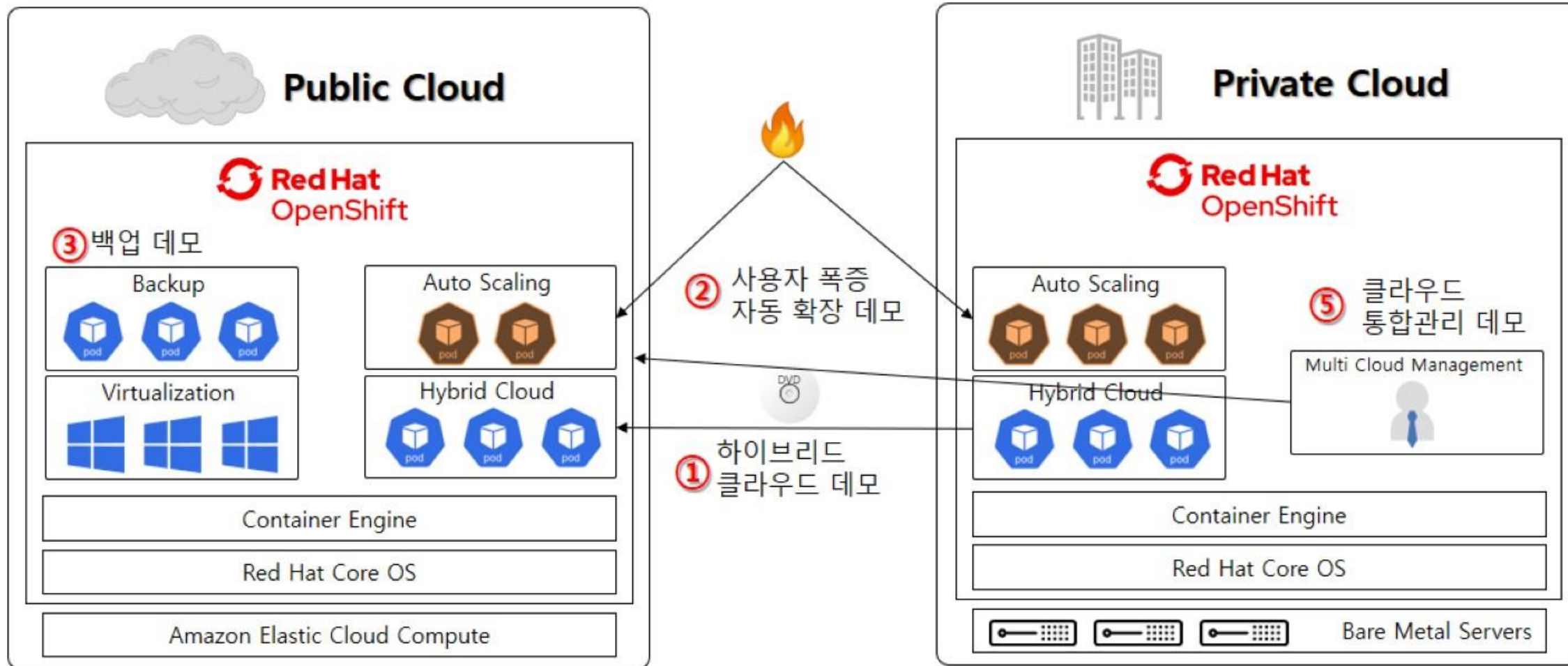
# Cloud Native OPENMARU APM DEMO

# 오픈 하이브리드 클라우드 데모



↻ 하이브리드 클라우드에서 같은 업무를 통합 운영, 관리하는 5가지 데모

④ 가상화 데모





조달청 디지털서비스몰 등록 

오픈마루

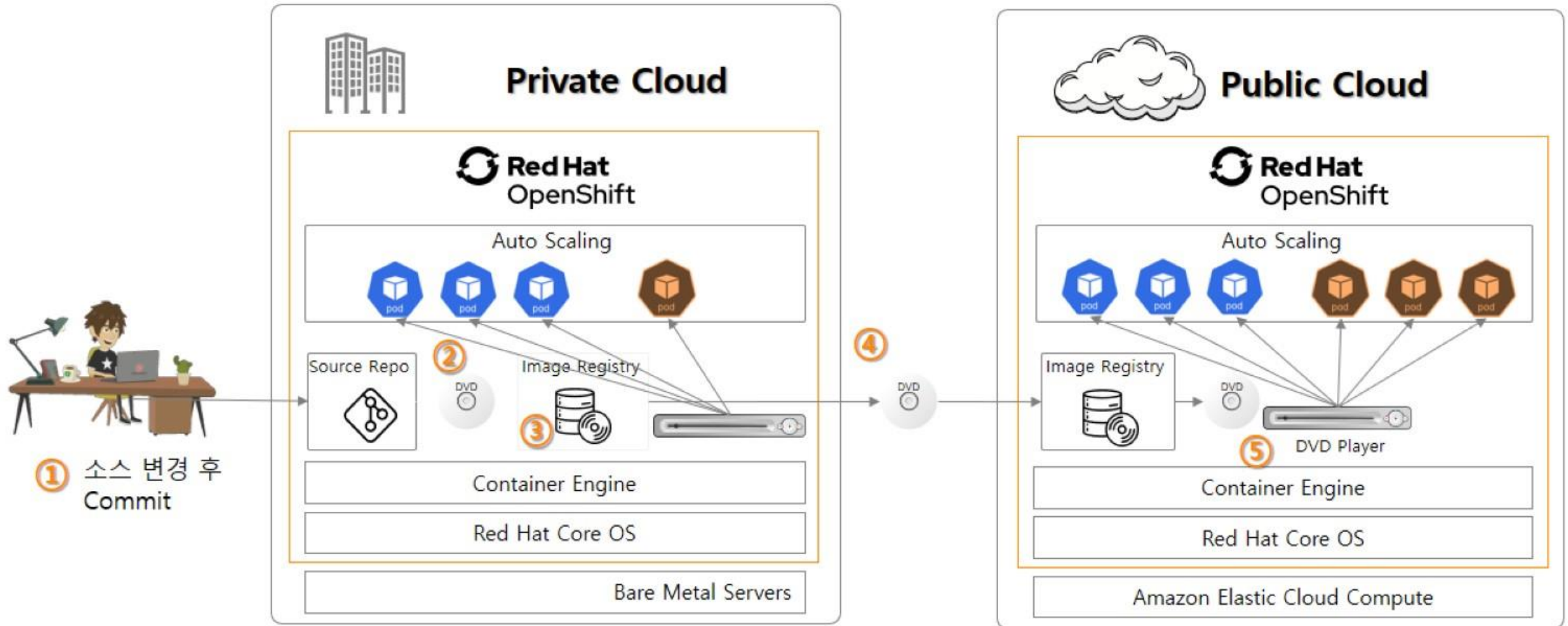


디지털서비스몰에서 **오픈마루** 를 검색하세요.

Cloud Native 환경에서 애플리케이션 배포 데모

# 개발은 내부 클라우드에서 운영은 외부 클라우드로

➡ 개발은 Private Cloud 에서 운영은 하이브리드로 Private/Public Cloud



# 전자정부 F/W 포탈에 대한 글로벌 서버 부하 분산



➡ **GSLB (Global Server Load Balancing)**을 통한 업무 부하 분산 데모

<input checked="" type="checkbox"/>	portal.egov.openmaru.io	가중치 기반
<input type="checkbox"/>	portal.egov.openmaru.io	가중치 기반
<input type="checkbox"/>	private.egov.openmaru.io	단순
<input type="checkbox"/>	public.egov.openmaru.io	단순

전자정부 F/W 포탈 서비스를 내부와 외부 클라우드에서 동시 운영

<https://portal.egov.openmaru.io>



30%

70%

<https://private.egov.openmaru.io>



<https://public.egov.openmaru.io>





# 하이브리드 클라우드 데모-개발환경 데모



eGovFrame 전자정부 표준프레임워크 공통컴포넌트 VERSION 3.9

로그인

아이디

비밀번호

아이디 저장

로그인

회원가입 | 아이디/비밀번호 찾기 | 인증서로그인 | 인증서안내

전자정부 포털 소스 수정



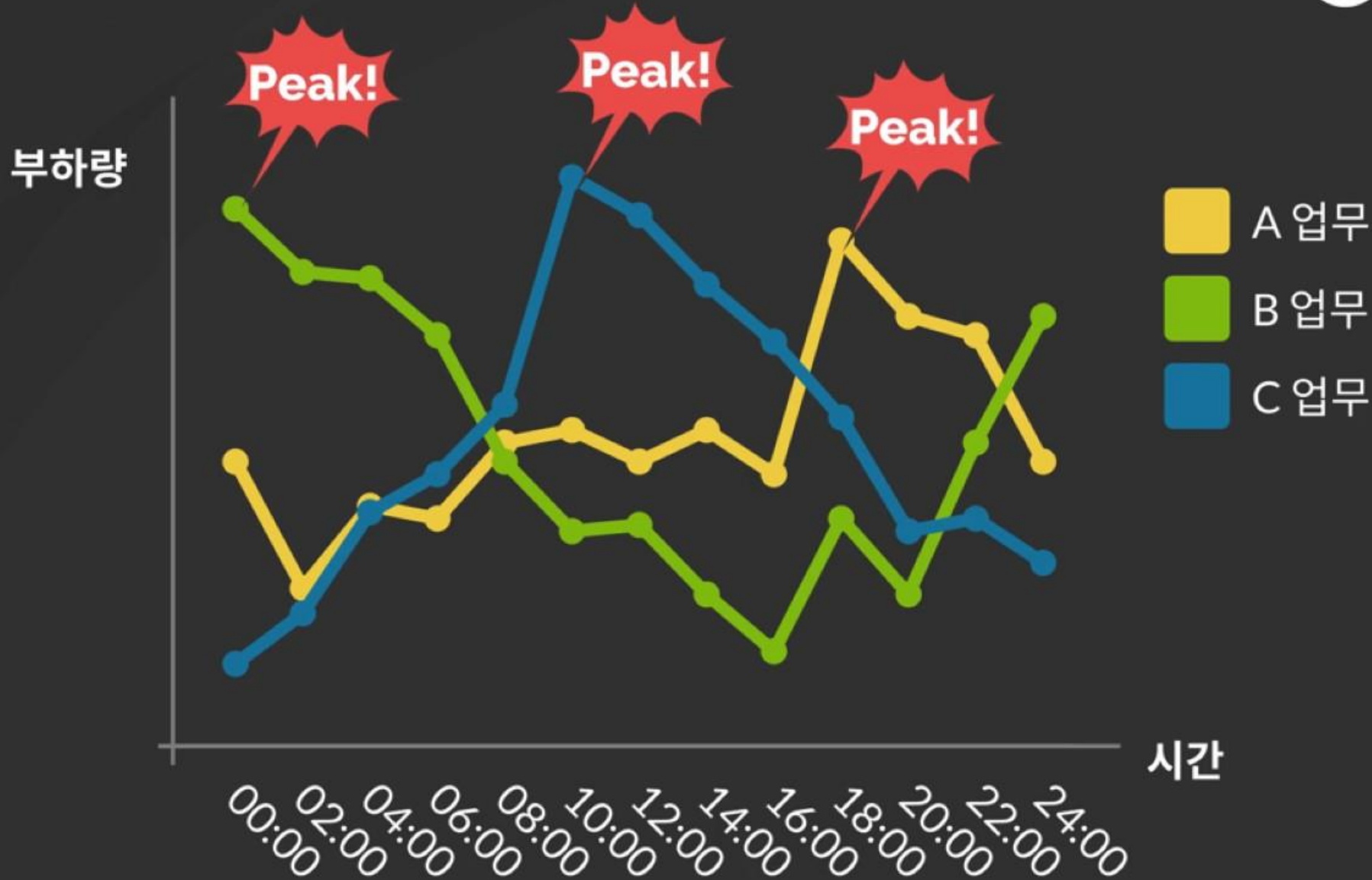
조달청 디지털서비스몰 등록 

오픈마루



디지털서비스몰에서 **오픈마루** 를 검색하세요.

통계청 차세대 나라통계  
멀티 애플리케이션에서 자동 확장



# 차세대 나라통계 - Peak 시점 별 자동 자원할당 활용 예시



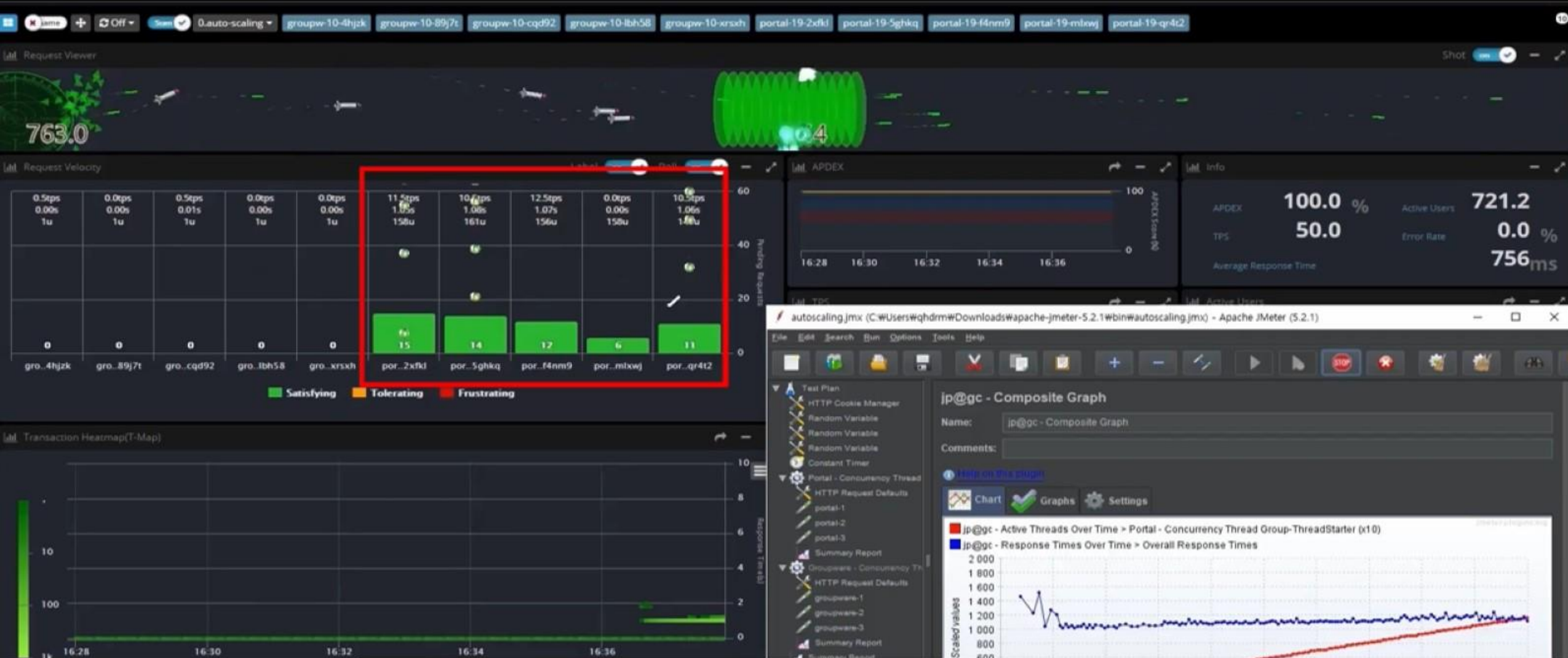
🔄 관리자의 개입없이 시스템 사용량에 따라 자동으로 확장/축소

- 한정된 서버자원을 Peak 시점에 따라 효율적으로 배분하여 사용 가능
- Peak 시점이 다른 통계조사에 대해 사람의 개입없이 자동자원할당이 가능



# T-01 : 인스턴스 개수 한정된 환경(기존 나라통계)

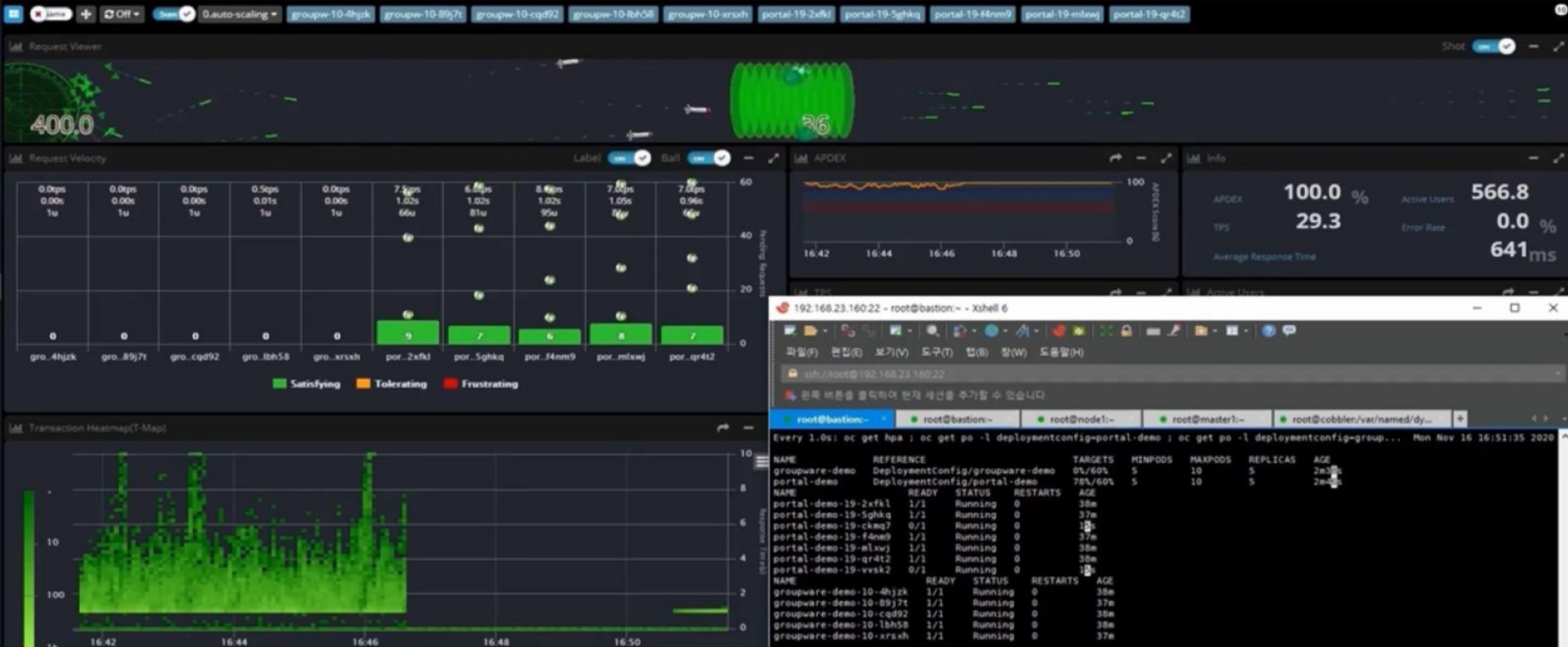
부하에 따른 응답시간 지연



# T-02 : 전국사업체조사 Peak일 때 자동부하분산 환경 부하테스트



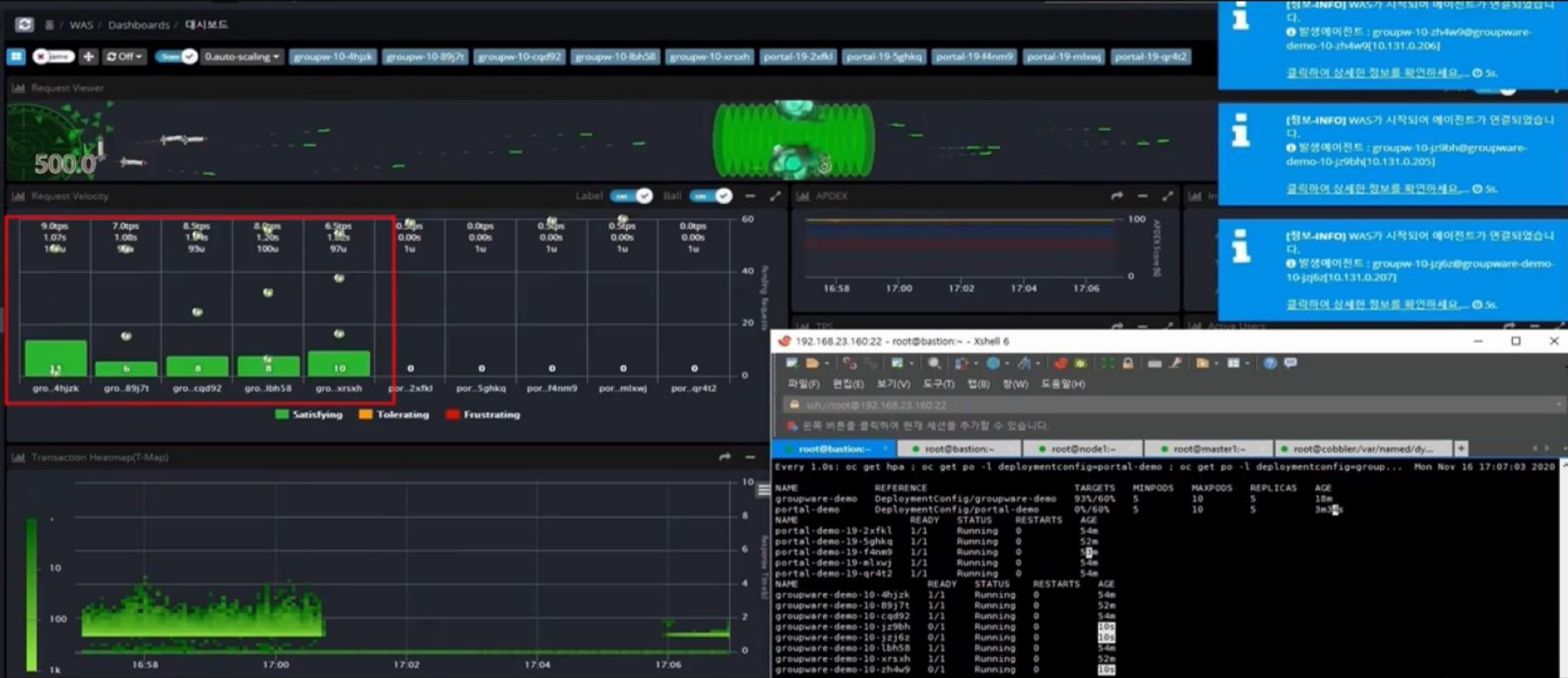
부하에 따른 컨테이너 자동확장으로 응답시간 보장과 TPS 증가



# T-03 : 초중고 사교육비조사 Peak일 때 자동부하분산 환경 부하테스트



부하에 따른 컨테이너 자동확장으로 응답시간 보장과 TPS 증가



# DEMO - 부하 테스트 결과 비교

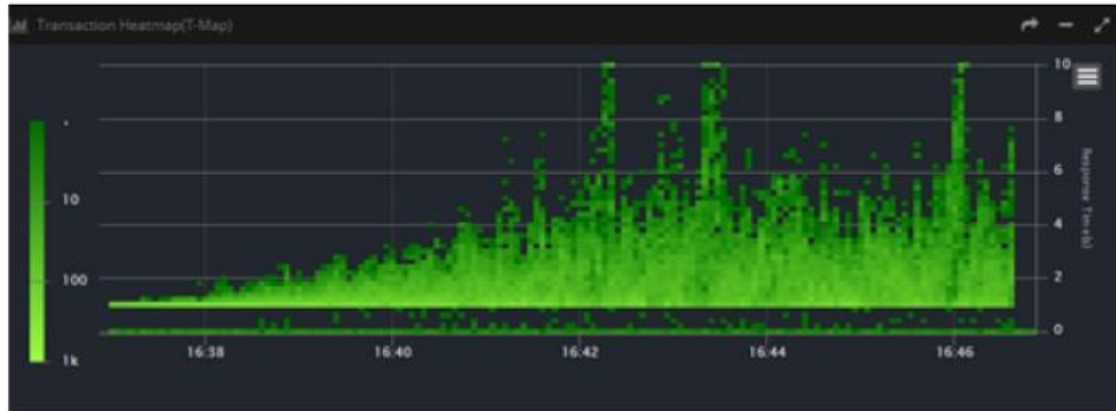


➡ Peak 시점이 다른 통계조사에 대해 별도 개입없이 자동자원할당이 가능함을 확인

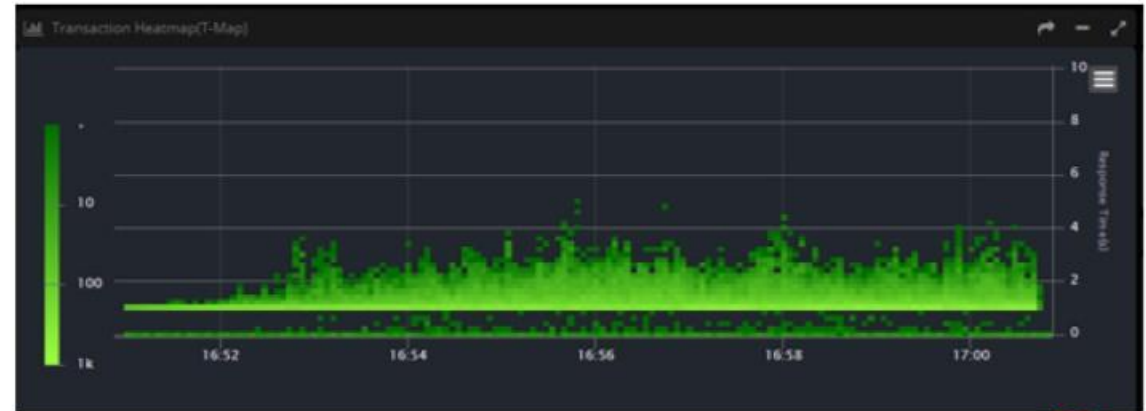
- 기존환경과 비교하여 자동자원할당이 되는 환경이 1.7 배 많은 양을 처리하며, 평균응답시간이 2.5 배 빠름

테스트 케이스	테스트 내용	시뮬레이션 테스트 환경	처리량	처리량 비교	평균응답시간	응답시간비교	최소응답시간	최대응답시간	TPS
T-01	자동자원할당이 안 되는 환경에서 부하테스트	기존 나라통계 환경	53,582	100%	2,374	100%	1,010	19,979	88
T-02	Peak 시점이 다른 자동 자원할당이 되는 환경에서 부하테스트	전국사업체 조사 Peak 일 때 환경	93,869	169%	919	258%	10	4,589	155
T-03	Peak 시점이 다른 자동 자원할당이 되는 환경에서 부하테스트	초중고 사교육비 조사 Peak일 때 환경	91,394	164%	971	244%	10	5,134	151

T-01 자동자원할당 안되는 기존 환경 응답시간분포



T-02/03 자동자원할당이 되는 환경 응답시간분포







조달청 디지털서비스몰 등록 

**오픈마루** 

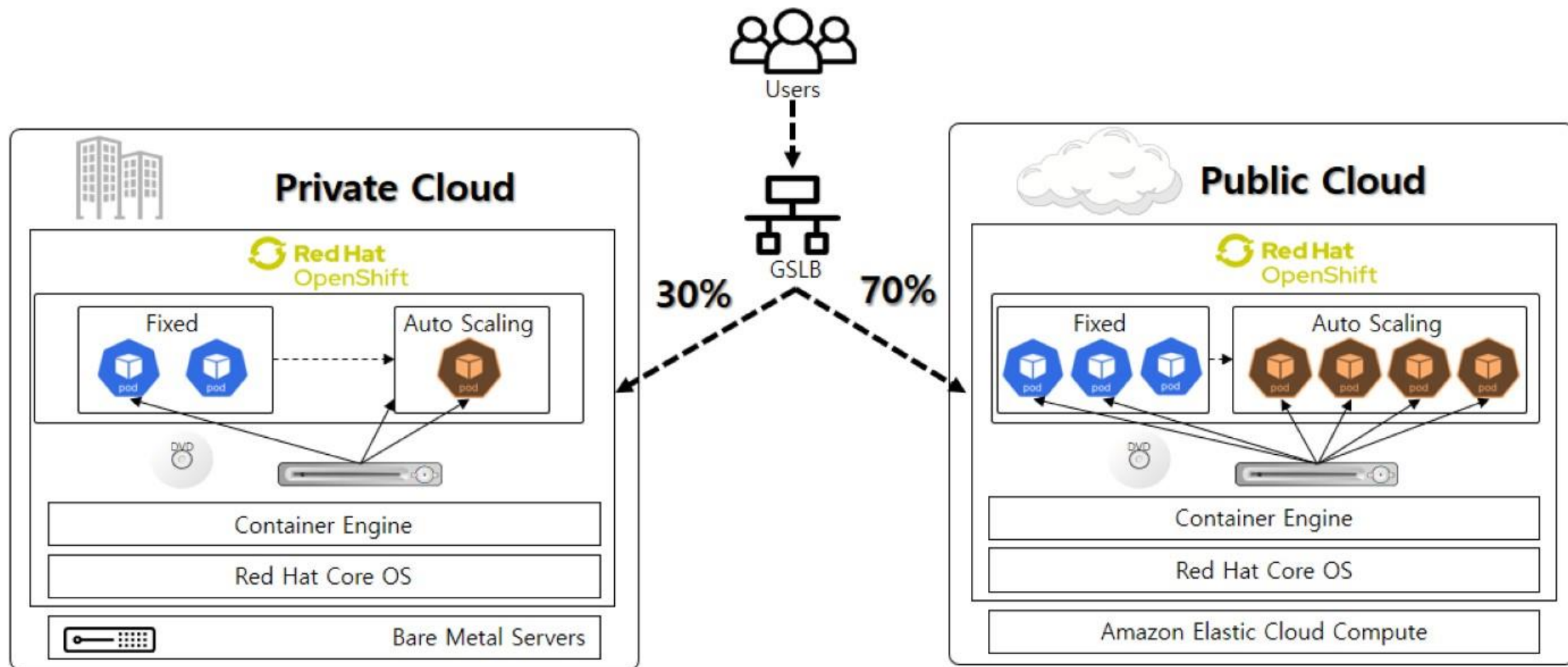
디지털서비스몰에서 **오픈마루** 를 검색하세요.

**하이브리드 클라우드 데모  
- 사용자 증가 자동 확장**

# 하이브리드 클라우드 데모 - 사용자 증가 자동 확장

↻ 하이브리드 클라우드로 운영되는 포털을 접속하는데 자동확장이 필요한 사용자 폭주상태를 가정

- 내부 클라우드와 외부 클라우드에 모두 동일한 홈페이지 서비스를 하고 프라이빗 30% vs. 퍼블릭 70% 로 운영 중



# 하이브리드 클라우드 데모 - 자동 확장 이전



openmaru  
APM openmaru, Inc

openmaru

Search... Shortcut: /, Ctrl ← → ↑ ↓

오른나루 한국어 ▼

홈 / 나의 대시보드 / EGOV



Private Cloud



Public Cloud

# 하이브리드 클라우드 데모 - 자동 확장 이후



주소: 주의 요항 | <https://private.egov.openmaru.io:10443> Gmail 이미지 주안

# Google

Google 검색 또는 URL 입력

- Google
- 대시보드
- Notion - The ...
- 받은편지함 (...)
- NAVER
- OPENMARU ...
- OPENMARU ...
- VMware /Sp...
- OPENMARU ...
- 바로가기 추가

- ### 내 Google Drive의 파일
- 2023년\_정기점검현황\_SUPPORT  
김사은님이 어제 수정함
  - 2023년 Performance Review 일정  
김연아님이 지난해에 공유함
  - 휴가신청 리스트  
류영훈님이 지난달에 수정함



**Private Cloud**



**Public Cloud**

Chrome 맞춤설정



조달청 디지털서비스몰 등록 



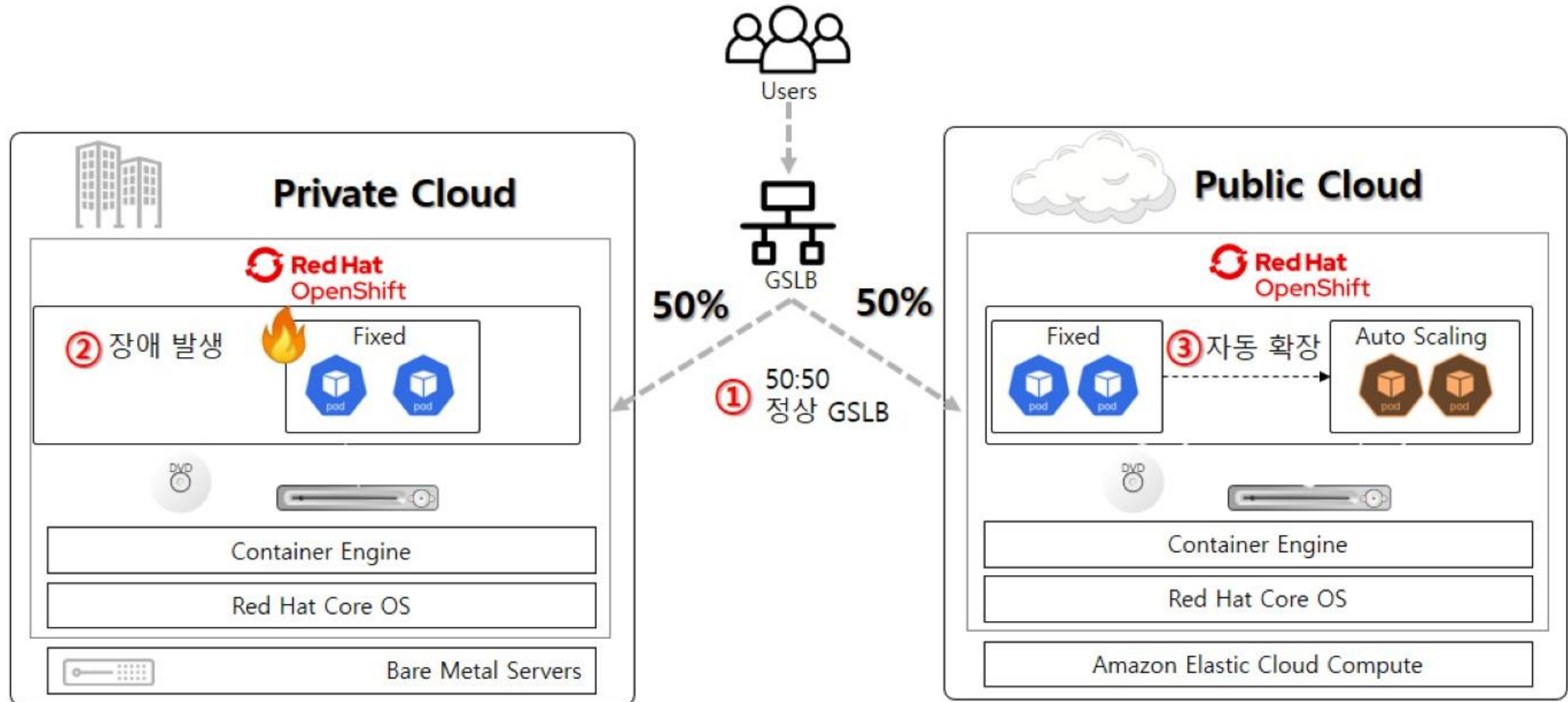
디지털서비스몰에서 **오픈마루** 를 검색하세요.

하이브리드 클라우드 데모  
- 장애시 자동 확장 데모

# 하이브리드 클라우드 데모 – Active Active DR

↻ 하이브리드 클라우드로 운영되는 포털을 접속하는데 프라이빗 클라우드 장애 발생을 가정

- 내부 클라우드와 외부 클라우드에 모두 동일한 홈페이지 서비스를 하고 프라이빗 50% vs. 퍼블릭 50% 로 운영 중



# 하이브리드 클라우드 데모 - 장애 발생 이전 Active-Active GSLB



openmaru  
APM openmaru, Inc.

openmaru

Search... - Shortcut -, Ctrl ← →

오픈마루 ☰ 📞 🔔 ⚙️ 🌐 🇰🇷 한국어 ▼



# 하이브리드 클라우드 데모 - Private Cloud 서비스 장애 발생





# 하이브리드 클라우드 데모 - Public Cloud 에서 부하에 따른 자동확장





조달청 디지털서비스몰 등록 

**오픈마루** 

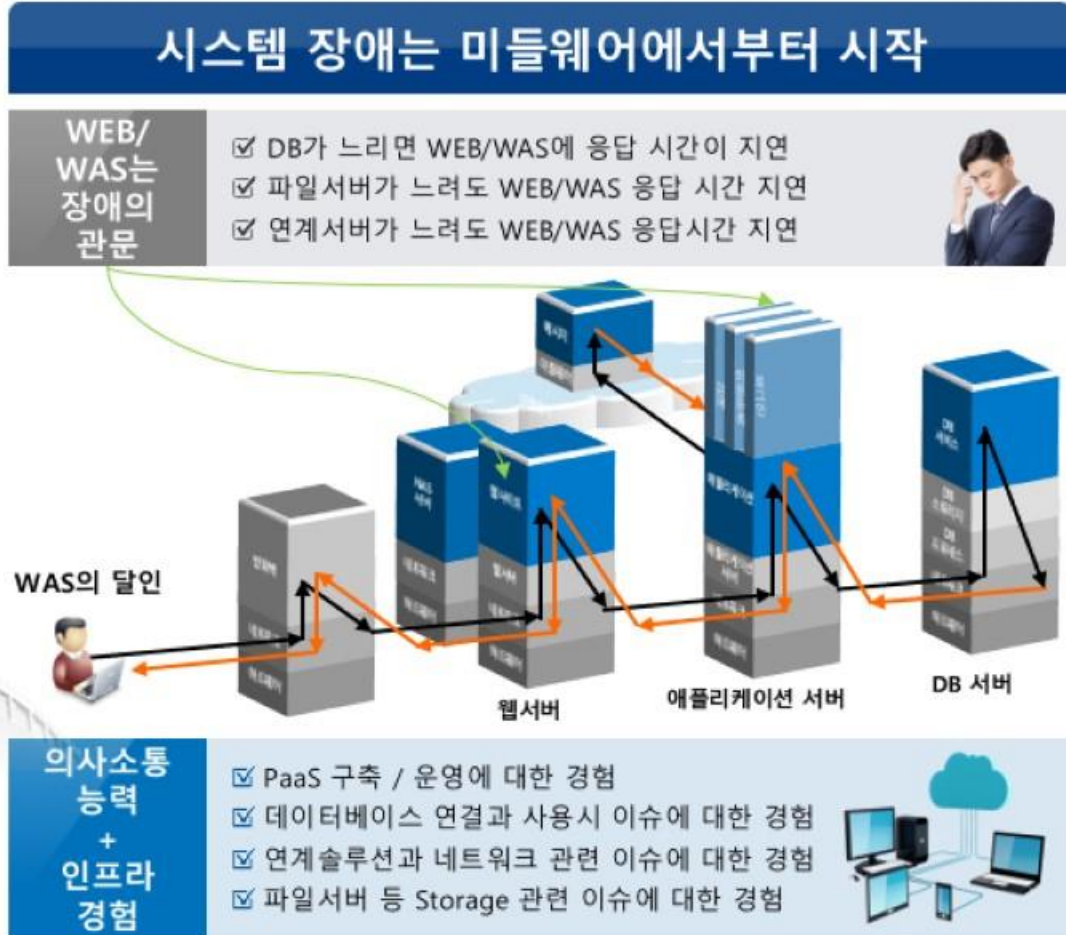
디지털서비스몰에서 **오픈마루** 를 검색하세요.

오픈마루 회사 소개

# 오픈소스와 WAS 전문가 집단 - 애플리케이션 중심 기술 지원



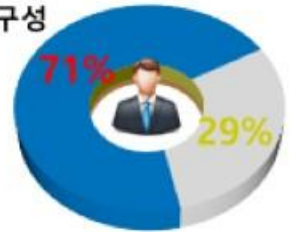
오픈소스 이지만 안전성, 보안, 성능은 WAS 달인을 통해서 해결



레드햇 PaaS 와 WAS 만 공급과 설치하는 오픈마루 달인들

오픈마루 인력 구성

- WAS & PaaS
- R&D



본 사업에 납품되는 버전 WAS 온라인 북



※ 저자 : 오픈마루 임직원들

# 애플리케이션 성능관리 전문 기업



➡ 어렵고 힘든 클라우드 사업, WEB/WAS 부분은 오픈마루가 책임지고 수행했습니다.

## 오픈마루는 레드햇 스페셜 파트너

**Red Hat OpenShift** | **Red Hat JBoss Enterprise Application Platform** **PaaS와 WAS 고객**

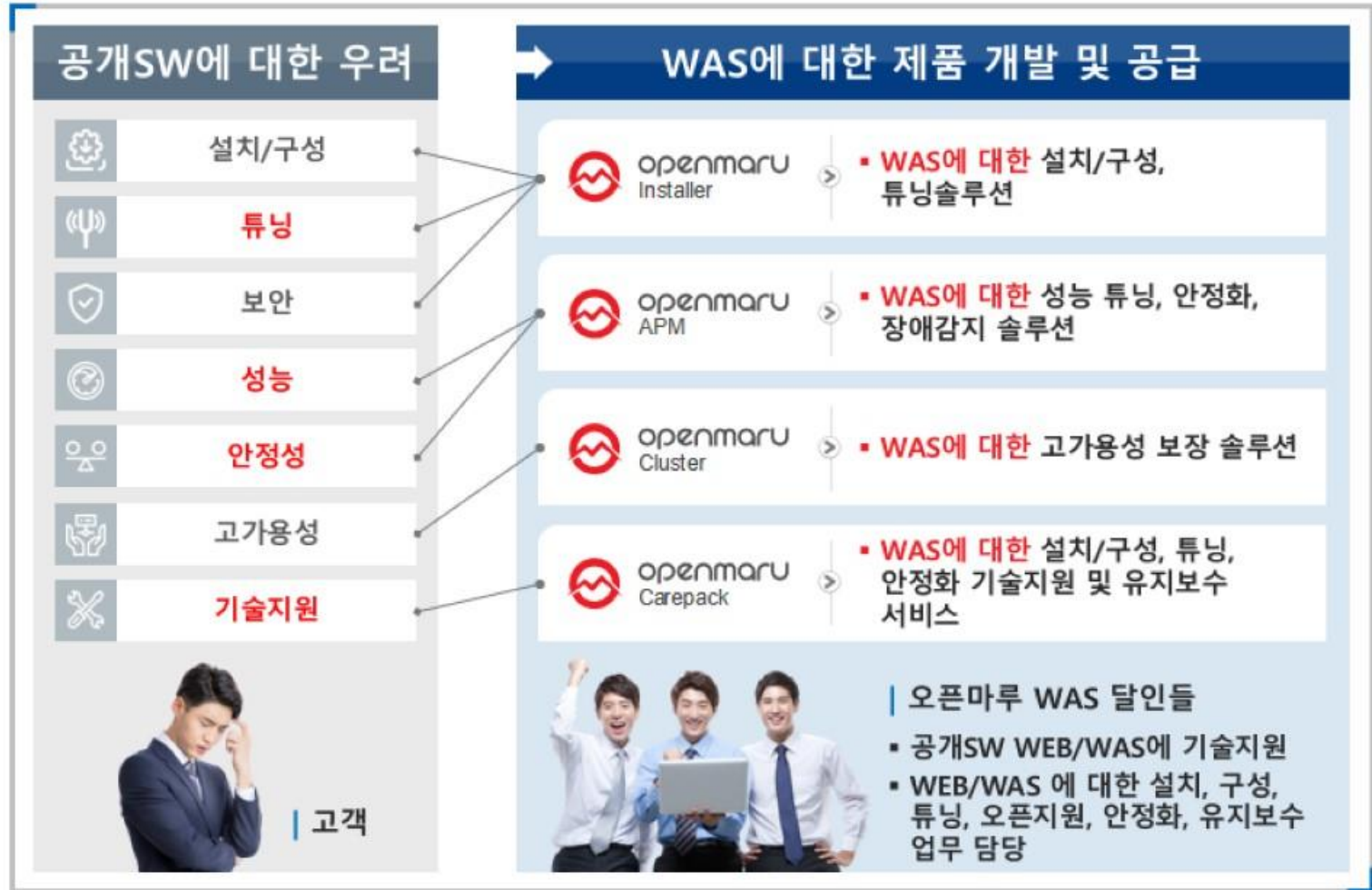
<b>KLID</b> 한국지식정보협회 온-나라 클라우드	<b>LOTTE CARD</b> 채널계, 계정계	<b>KAI</b> 한국항공우주연구원 클라우드 플랫폼
<b>KCB</b> 윌크레딧	<b>INHSAM</b> PaaS 시스템	<b>SAMSUNG</b> 삼성전자 PaaS 시스템

**Red Hat JBoss Enterprise Application Platform** **WAS 고객**

<b>Finnq</b> 핀테크	<b>adidas</b> 이커머스 사이트	<b>KOREAN AIR</b> Crew Manager
<b>DAEKYO</b> 온라인 교육 서비스	<b>EBS</b> 코로나 대비 온라인클래스	<b>KERIS</b> 디지털교과서

**Apache Tomcat** **Apache Tomcat 고객**

<b>MEDIAWILL</b> ERP 시스템	<b>해피피어니</b> 상품권 서비스	<b>ELAND</b> 이커머스 사이트
<b>SPC</b> 결제 서비스	<b>koscom</b> 모바일 서비스	<b>한국기술교육대학교</b> 학사행정, 홈페이지



# 애플리케이션 마이그레이션 전문 - U2L / U2V / U2C



➡ 오픈 소스 WAS 인 WAS 로 전환하더라도 성능, 안정성, 보안에 책임 질 수 있어야 합니다.

## U2L/U2C 전환 사업의 핵심은?



단순한 사업 규모를 고려하면 중요도와 난이도를 고려하면

## WAS 설치/구성/튜닝 시 95% 단축



## OPENMARU Installer 구축 사례

온라인 개학	대상	서버 수
2020년 4월 9일	중3, 고3	60 대
2020년 4월 16일	초4~6, 중1~2, 고1~2	240대
2020년 4월 20일	초1~3	300 대

- 3 주 만에 600대 서버 WAS 설치 튜닝 - EBS 온라인클래스
- 코로나로 인한 학습 공백 - 준비되지 않은 클라우드 환경 튜닝 및 안정화



**강동경희대학교병원**  
KYUNG HEE UNIVERSITY HOSPITAL AT GANGDONG

강동경희대학교병원 (2020)

**통합의료정보시스템 고도화 사업**

Unix2Linux\* HCI 장비 기반의 가상화

- JEUS 에서 WAS 교체 , 제니퍼에서 OPENMARU APM

**건국대학교병원**  
KONKUK UNIVERSITY MEDICAL CENTER

건국대학교병원 (2017)

**전산기기 교체 및 운영환경 고도화 사**

Unix2Linux\* Bare Metal서버 기반

- JEUS 에서 WAS 교체 , 제니퍼에서 OPENMARU APM

**adidas**

Adidas Korea (2019)

**E-Commerce/POS AWS 전환**

Unix2Cloud\* Oracle Database To RDS

- WebLogic To WAS
- OPENMARU APM

**KDIC 예금보험공사**  
Korea Deposit Insurance Corporation

예금보험공사(2015)

**IT인프라최적화 사업 (2015)**

Unix2Linux\* Oracle DB Upgrade

- WebSphere To WAS

포장이사 전문 - WAS 전환 핵심은 애플리케이션 마이그레이션

# 클라우드를 넘어 SaaS 서비스 까지 하는 APM 전문기술기업



🔄 설치형 APM 제품에서 부터 SaaS 클라우드 서비스까지 구축하고 운영중인 클라우드 전문 기술 기업입니다.

- 국내 최초로 OpenShift 와 Kubernetes 지원, 2021년에는 AWS Cloud 에 OPENMARU Cloud APM SaaS서비스를 오픈
- SaaS 서비스 구축은 클라우드 네이티브 기술과 보안, 멀티테넌시, 미터링, 데이터 관리 등 높은 수준의 기술들을 구현

laas,PaaS 부터
SaaS 까지



**SAMSUNG SDS Cloud**

삼성SDS 클라우드에 OPENMARU APM 등록



**AWS Cloud 에서 SaaS 클라우드로 서비스 중인 OPENMARU Cloud APM**

- 2021년 9월 정식 오픈
- 2022년 2월 현재 - 등록 사용자 :28명 / 모니터링 프로젝트: 76 개

완벽한 멀티테넌시 구현

3분 안에 모니터링 개시

가장 저렴한 가격

검증된 성능과 확장성

컨테이너 기반 미터링

홈페이지

<https://www.openmaru.io>



클라우드 콘솔

<https://console.openmaru.io>



OPENMARU APM

<https://<user>.apm.a-apne2.openmaru.io/>



삼성이 검증한 APM 제품

컨테이너 환경에 최적화된 APM

클라우드 적합한 가격 정책






# 다양한 운영 환경에서 검증된 APM 제품

➡ 성능과 장애 관리 도구인 APM은 물리/가상서버, 클라우드 등 다양한 WAS운영 환경에서 검증되어야 합니다.

- OPENMARU APM은 국내 최초로 PaaS 와 Kubernetes 에서 WAS와 클라우드 인프라 모니터링을 제공하였습니다.
- 중앙부처와 지방자치단체를 비롯한 금융권과 주요 국내 기업에서 OPENMARU APM 을 사용하고 있습니다.

**OPENMARU APM & PaaS 9개 이상**


**주요 지방자치단체 구축 사례 외 다수**


**주요 기업 구축 사례 외 다수 10개 이상**


**주요 공공기관 구축 사례 외 다수 30개 이상**


**학교 / 병원 외 다수 5개 이상**




# 대규모 PaaS 운영환경에서 검증된 OPENMARU APM

↻ 국내 공공기관 및 대기업에 도입된 OPENMARU APM 주요 구축사례입니다.

<p>발주처</p>				
<p>프로젝트 명</p>	<p>온-나라 클라우드 문서 2.0</p>	<p>스마트 플랫폼 구축</p>	<p>MSA 기반 개인신용정보</p>	<p>롯데카드 채널계 클라우드 구축</p>
<p>당면과제</p>	<ul style="list-style-type: none"> <li>1차 사업 - 성능 및 안정화 이슈</li> <li>2차 사업 - 오픈마루가 성능/장애 해결</li> </ul>	<ul style="list-style-type: none"> <li>한국항공우주산업의 글로벌 경쟁력확보에 필요한 스마트플랫폼 구축</li> </ul>	<ul style="list-style-type: none"> <li>개인신용정보서비스와 업신용정보서비스 같은 핵심적인 서비스 운용 중</li> <li>MSA 기반으로 핵심 비즈니스 서비스 구축</li> </ul>	<ul style="list-style-type: none"> <li>금융권 최초로 채널계 시스템 도입으로 대고객 접점 서비스</li> <li>국내최초로 OPENMARU APM으로 안정화 지원 및 성능 튜닝</li> </ul>
<p>오픈마루 역할</p>	<p>PaaS/WAS 성능관리 사업자</p>	<p>PaaS BMT 수행/구축/유지보수</p>	<p>PaaS 구축 및 안정화 지원</p>	<p>APM 공급 및 안정화 지원</p>



- PaaS 문제 발생!! 소방관
- WAS 문제 발생!! 해결사









# 코리아 크레딧 뷰로(KCB) - 클라우드 네이티브 전환



🔄 코리아 크레딧 뷰로(KCB) MSA 기반 개인신용정보, 기업신용정보, 마이데이터 서비스 - PaaS & OPENMARU APM



프로젝트 명	올크레딧 개인신용정보 서비스
얼마나 중요한 업무인가?	신용관리, 명의보호, 금융관리를 통합하여 관리하는 국가대표기관 시스템
규모는 어떻게 되는가?	개인신용정보서비스와 기업신용정보서비스 같은 핵심적인 서비스 운용중
오픈마루 역할은?	오픈소스 WAS 마이그레이션 컨설팅, 클라우드 네이티브 환경 도입 컨설팅

☑️ OPENMARU와 함께 PaaS 환경을 도입/구축한 서비스



카드본인확인서비스  
상용 WAS를 오픈 소스로 전환



개인신용정보서비스  
PaaS 도입



기업CB 구축사업  
MSA 기반으로 재개발



마이데이터 신규 구축 사업  
API G/W 도입

# 행정안전부 - 국가 80 여개 중앙부처 그룹웨어(온-나라) 시스템



한국지역정보개발원 온-나라 클라우드 문서 2.0 사업( 2018 년 ) - Red Hat PaaS & OPENMARU APM



프로젝트 명	온-나라 클라우드 문서 2.0
얼마나 중요한 업무인가?	정부부처 20군데와 지방자치단체 약 80군데 등 100만 공무원이 사용하는 문서 결재 시스템
규모는 어떻게 되는가?	Worker Node 80대 규모
오픈마루 역할은?	PaaS 전문인력 상주와 OPENMARU APM으로 성능관리를 담당하여 안정화 지원

OPENMARU APM은 국내 최초, 최다 컨테이너 & 클라우드 환경을 지원

**법정부 기안 업무관리시스템으로 효율적인 행정 구현**  
- 행정부, 26개 기관 대상으로 법정부 기안 온-나라시스템 고도화 착수 -

□ 행정안전부(장관 김부겸)는 17일 정부세종청사 행정안전부(내국선 소책)에서 중앙부처 및 지자체 온-나라시스템\* 담당자 및 관계자 등 20여명이 참석한 가운데 "클라우드(인터넷 기반 정보 통신 자원 통합·공유 서비스) 기반 온-나라시스템 고도화 사업" 착수보고회를 개최하였다.

\* 온-나라 시스템 : 행정기관의 업무에 대한 문서 작성·검토·결재·공공·공공 등 문서처리와 모든 과정을 기록·관리하는 전자결재시스템

○ 이번 고도화 사업은 국무조정실, 관공위원회 등 26개 기관을 대상으로 각 기관별로 보고서 및 문서를 저장·관리하는 기존방식에서 통합저장소(클라우드)에서 공동기안·결재가 가능하도록 하는 사업으로 2019년까지 전 중앙부처에 확산할 계획이다.

행정안전부 보도 자료  
법정부 기안 온-나라 시스템 고도화 착수



클라우드 전환 시 노후 장비 교체 대비  
2.6배의 비용 효과 발행



클라우드 온-나라 문서 2.0  
개념도



온나라 클라우드 APM  
성능 테스트

# 한국교육방송공사 - 코로나 대응을 위한 온라인 클래스 시스템



➡ EBS 온라인 클래스에 도입되어 동시접속자 130만명, VM 600 대, WAS 인스턴스 1200 여 개 모니터링



실제 600대 WEBWAS를 OPENMARU APM으로 모니터링하는 Dash Board 화면



프로젝트 명	EBS 온라인 클래스
얼마나 중요한 업무인가?	코로나로 인하여 초,중,고 300백만명의 학생이 접속하는 온라인 클래스 구축 업무
규모는 어떻게 되는가?	WEB/WAS 600대 / 인스턴스1,200개 규모
오픈마루 역할은?	1주일 내 600대 머신에 1,200개 인스턴스 구축 및 APM으로 오픈지원, 성능 튜닝 및 안정화 지원

✓ OPENMARU APM으로 안정화를 실현한 실제 EBS 방송 화면



교육 차관에게 OPENMARU APM으로 보고하는 화면



600여대의 서버의 총 1,228개 WAS 모니터링



실제 인스턴스 1,200개 그룹대시보드 모니터링 화면-2



실시간 Active User를 확인할 수 있는 모니터링 화면



조달청 디지털서비스몰 등록 



디지털서비스몰에서 **오픈마루** 를 검색하세요.

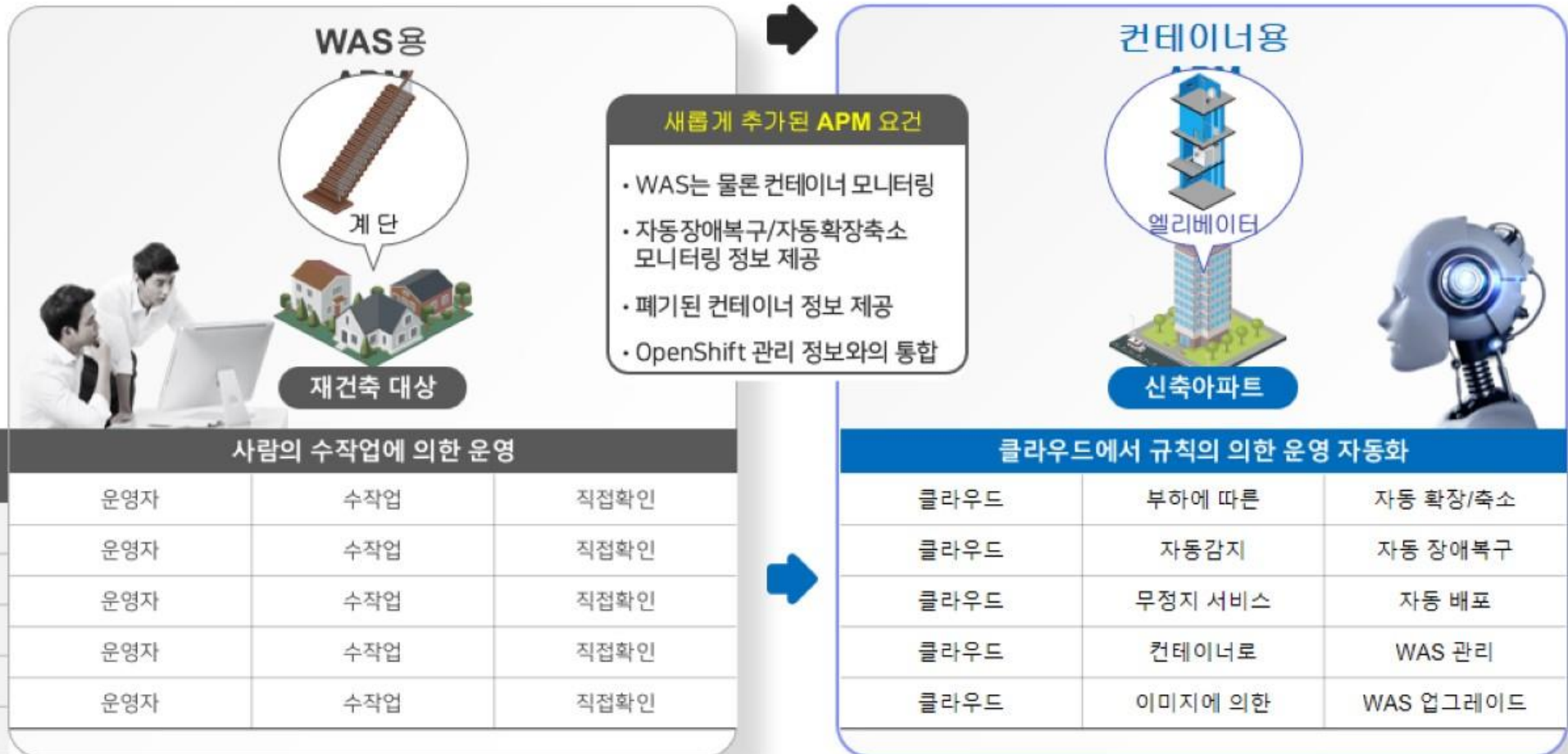
OPENMARU APM with Cloud Native



# 클라우드 전환에 따른 APM 요구사항의 변화

## 클라우드에서는 APM 모니터링 대상이 WAS에서 컨테이너로 변경

- 컨테이너 단위로 WAS에 대한 확장/축소, 장애 복구, 업그레이드, 패치 작업하여 WAS와 함께 컨테이너를 모니터링해야 합니다.
- 기존 물리서버나 가상서버와는 달리 컨테이너는 휘발성으로 상태를 가지고 있지 않습니다.

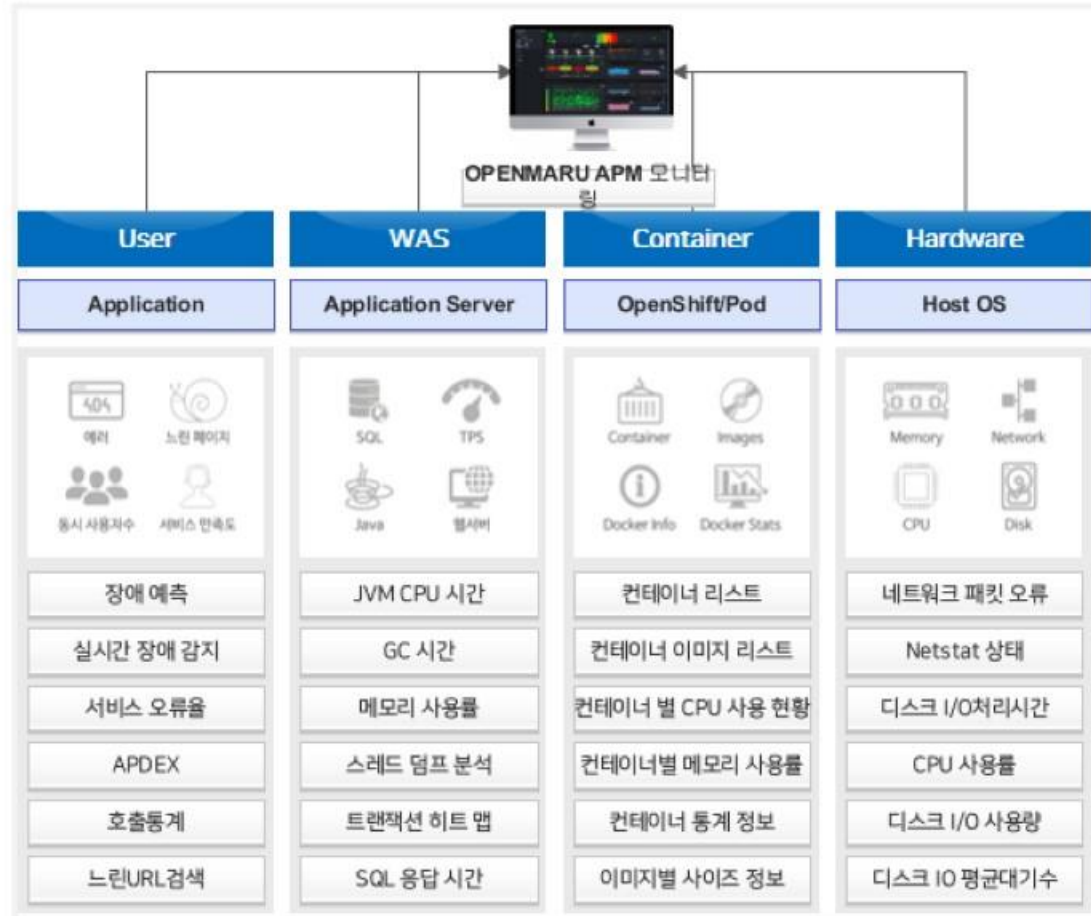
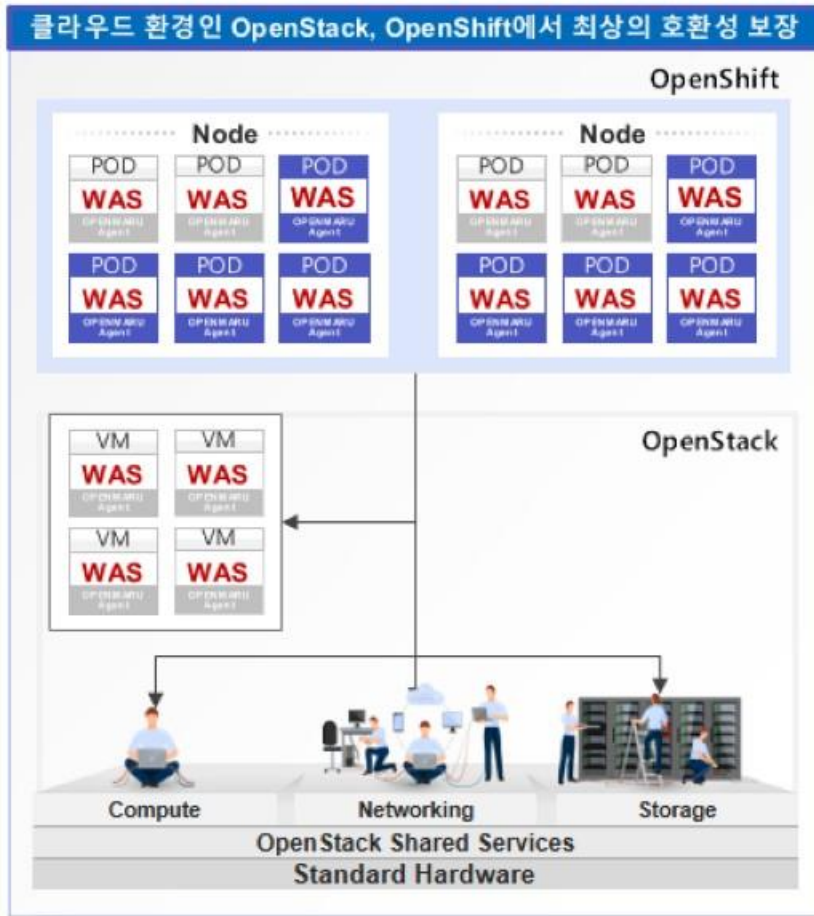




# 클라우드 환경에 최적화된 APM

## 클라우드 운영환경, 개발,검증 환경에 최적화된 기능 제공

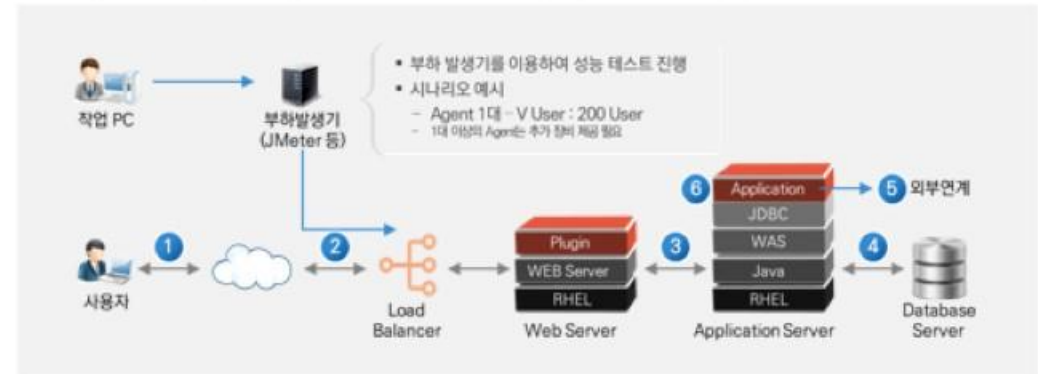
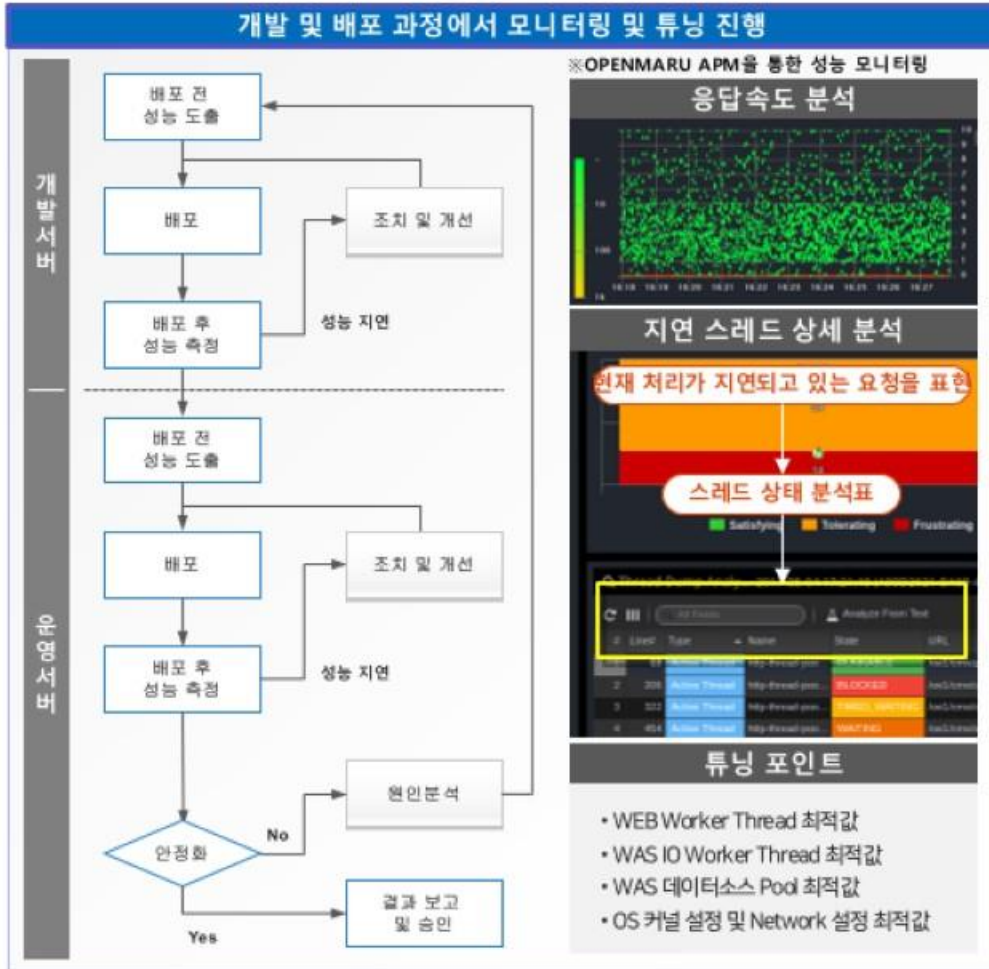
- 가상화(OpenStack 등), Kubernetes(OpenShift)에 최적화된 APM 입니다. 다수의 대규모 컨테이너 환경에 적용되어 적합성을 이미 검증 받았습니다.
- OPENMARU APM은 클라우드 환경(VM/Container)에 유리한 라이선스 정책을 적용하고 있습니다.



# 서비스 안정화 및 기술지원

➡ APM은 서비스 개발 및 오픈이후 안정화까지 모니터링과 튜닝이 필요합니다.

● OPENMARU APM을 활용하여 개발 시점부터 오픈 시점까지 지속적인 모니터링 및 튜닝을 시행하여 시스템의 안정적 오픈과 운영을 지원합니다.



☑ 주요 구간별 확인 및 튜닝 포인트

1	사용자 - 방화벽 및 L4(LB)	<ul style="list-style-type: none"> <li>방화벽</li> <li>대역폭 확인</li> </ul>
2	방화벽 및 L4(LB) - WEB	<ul style="list-style-type: none"> <li>목적지 연결 설정</li> <li>로드 밸런싱 옵션 (Client IP Hash)</li> </ul>
3	WEB - WAS	<ul style="list-style-type: none"> <li>MEM Worker Thread 튜닝</li> <li>ModJk 연결 옵션 튜닝</li> <li>ModJk Sticky 옵션</li> </ul>
4	WAS - DBMS	<ul style="list-style-type: none"> <li>IO Worker Thread 튜닝</li> <li>DB Connection Pool 튜닝</li> <li>DB Statement Cache 튜닝</li> <li>JVM Heap Memory 튜닝</li> </ul>
5	WEB/WAS - NAS	<ul style="list-style-type: none"> <li>IOPS, 지연 시간 확인</li> </ul>
6	WAS - 외부 인터페이스	<ul style="list-style-type: none"> <li>HTTP API Connection Pool 튜닝</li> </ul>
7	애플리케이션	<ul style="list-style-type: none"> <li>오류 확인</li> <li>비즈니스 로직 성능 확인</li> </ul>

# 컨테이너 기반 모니터링 기능



## ➔ 컨테이너(쿠버네티스) 기반 모니터링 기능 제공

- 컨테이너에서 작동하는 WAS의 모니터링 뿐만 아니라 컨테이너의 CPU, 메모리, 네트워크 트래픽/패킷 오류 를 확인 할 수 있습니다.
- 동작하는 컨테이너의 모니터링 뿐만 아니라 컨테이너 이미지의 메타 상세 정보도 확인 할 수 있습니다.



컨테이너(PaaS) 지원 APM 과 기존 APM 기능 비교			
구분	세부 항목	OPENMARU APM (컨테이너 지원 APM)	기존 APM 제품
기존 시스템	Java	• 모니터링 지원	모니터링 지원
	WAS	• 모니터링 지원	모니터링 지원
클라우드 시스템	PaaS 환경	• Docker Daemon 과 관련 이미지 정보 제공 • POD 에 대한 리소스 정보 제공 (CPU/Memory/Disk/Network)	모니터링 <b>지원 불가</b>
	오토스케일링	• 오토스케일링 관련 리소스 정보 제공 • WAS 상태 정보 제공	모니터링 <b>일부 지원</b>
	PaaS 에서 Java 정보	• POD 상에서 실행되는 Java 가상 머신 정보 제공 (Heap, Java 상태 정보 등)	모니터링 <b>일부 지원</b>
	PaaS 에서 WAS 정보	• POD 상에서 실행된 WAS 에 대한 정보	모니터링 <b>일부 지원</b>
	POD 상태 정보 제공	• 폐기된 POD 에 대한 검색 지원 • 과거 POD 에 대한 정보 제공	모니터링 <b>일부 지원</b>





# 컨테이너 환경 에이전트 관리방안

## ➡ 컨테이너 환경에서 WAS 인스턴스 관리 기능 제공

- 오토 스케일 인/아웃 상황에 따라 에이전트가 자동으로 추가되고, 자동으로 숨겨집니다.
- 컨테이너 중지로 인해 삭제된 인스턴스의 모니터링 정보는 삭제되지 않고 숨겨져 있으므로 필요 시 다시 분석 가능합니다.



Auto Scaling으로 증가된 인스턴스 자동 추가 및 모니터링

Auto Scaling으로 제거된 인스턴스 자동 삭제

# 컨테이너 기반 모니터링 기능



## ➡ 컨테이너(쿠버네티스) 기반 모니터링 기능 제공

- 컨테이너에서 작동하는 WAS의 모니터링 뿐만 아니라 컨테이너의 CPU, 메모리, 네트워크 트래픽/패킷 오류 를 확인 할 수 있습니다.
- 동작하는 컨테이너의 모니터링 뿐만 아니라 컨테이너 이미지의 메타 상세 정보도 확인 할 수 있습니다.



컨테이너(PaaS) 지원 APM 과 기존 APM 기능 비교			
구분	세부 항목	OPENMARU APM (컨테이너 지원 APM)	기존 APM 제품
기존 시스템	Java	• 모니터링 지원	모니터링 지원
	WAS	• 모니터링 지원	모니터링 지원
클라우드 시스템	PaaS 환경	• Docker Daemon 과 관련 이미지 정보 제공 • POD 에 대한 리소스 정보 제공 (CPU/Memory/Disk/Network)	모니터링 <b>지원 불가</b>
	오토스케일링	• 오토스케일링 관련 리소스 정보 제공 • WAS 상태 정보 제공	모니터링 <b>일부 지원</b>
	PaaS 에서 Java 정보	• POD 상에서 실행되는 Java 가상 머신 정보 제공 (Heap, Java 상태 정보 등)	모니터링 <b>일부 지원</b>
	PaaS 에서 WAS 정보	• POD 상에서 실행된 WAS 에 대한 정보	모니터링 <b>일부 지원</b>
	POD 상태 정보 제공	• 폐기된 POD 에 대한 검색 지원 • 과거 POD 에 대한 정보 제공	모니터링 <b>일부 지원</b>

# 컨테이너 환경 에이전트 관리방안



## 🔄 컨테이너 환경에서 WAS 인스턴스 관리 기능 제공

- 오토 스케일 인/아웃 상황에 따라 에이전트가 자동으로 추가되고, 자동으로 숨겨집니다.
- 컨테이너 중지로 인해 삭제된 인스턴스의 모니터링 정보는 삭제되지 않고 숨겨져 있으므로 필요 시 다시 분석 가능합니다.



Auto Scaling으로 증가된 인스턴스 자동 추가 및 모니터링

Auto Scaling으로 제거된 인스턴스 자동 삭제

# 확장가능한 수집서버 아키텍처

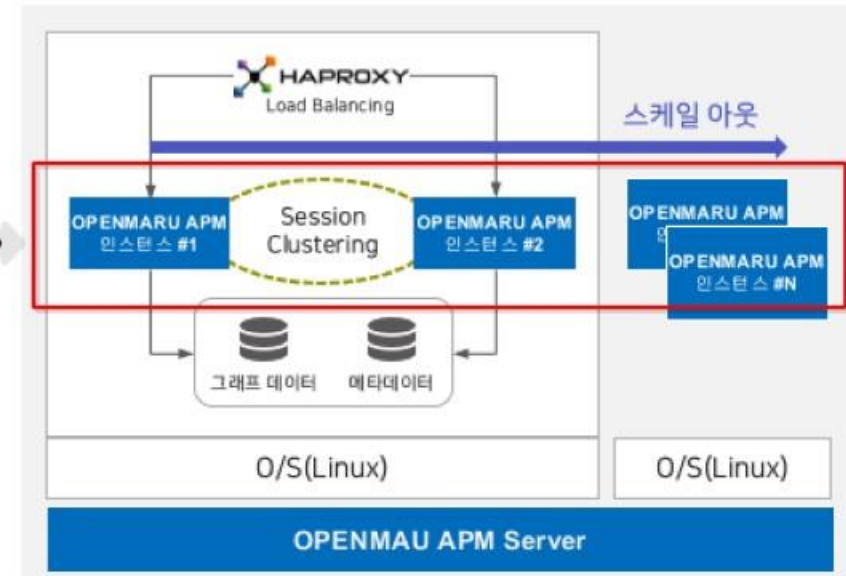


## ↻ 확장 가능한 아키텍처 지원 기능 제공

- 대용량 에이전트를 지원하기 위해 OPENMARU APM은 기본적으로 이중화로 구성되며, 필요한 경우 수집 인스턴스를 확장하여 부하를 분산할 수 있습니다.
- 이중화 된 OPENMARU APM 인스턴스는 HAProxy와 OPENMARU CLUSTER 세션 클러스터링을 통해 무 중단 서비스를 지원합니다.



- ⊙ 세션 유지를 위해 OPENMARU CLUSTER 세션 클러스터링 구성
- ⊙ 자체적인 이중화를 통해서 한쪽 서버가 중단되더라도 서비스뿐만 아니라 모니터링도 정상적으로 유지
- ⊙ 시스템 장애 발생 시 장애복구가 용이
- ⊙ 패치 및 업그레이드 과정 시에도 365일 24시간 안정적인 무중단 서비스 작업을 위한 이중화 구조

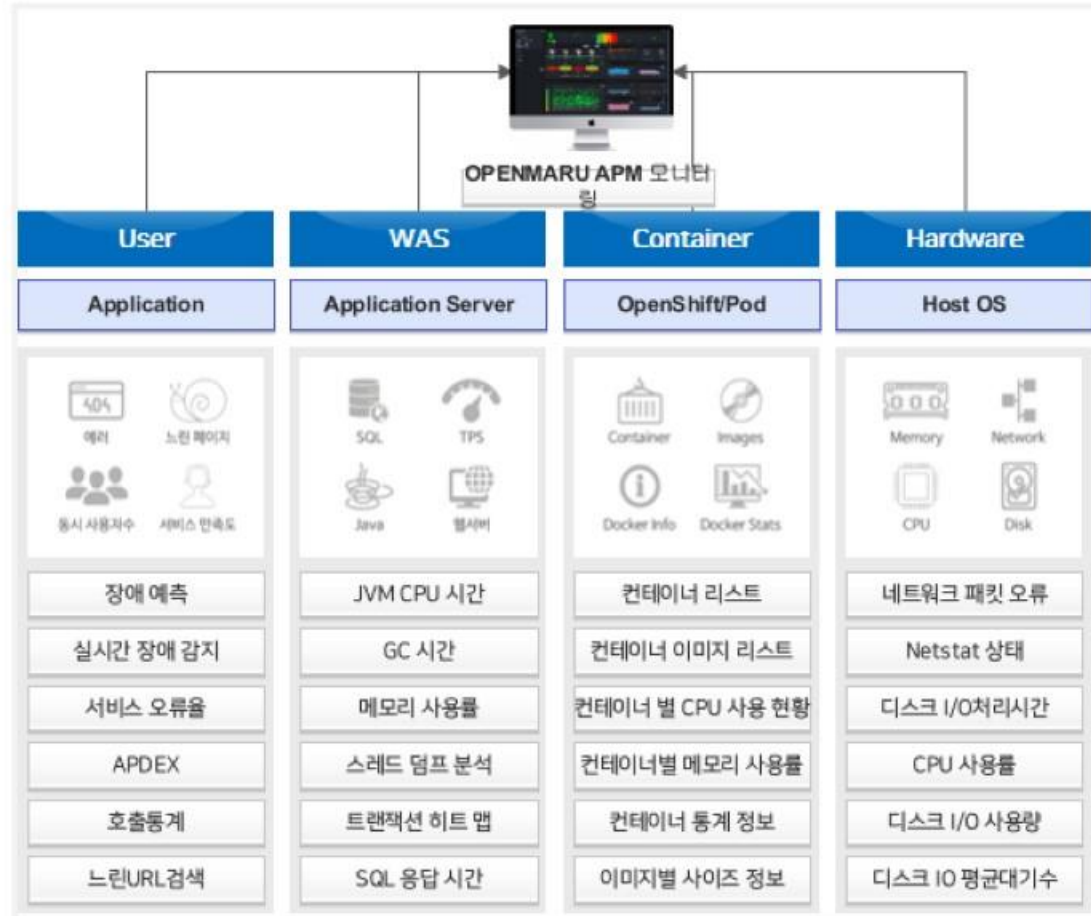
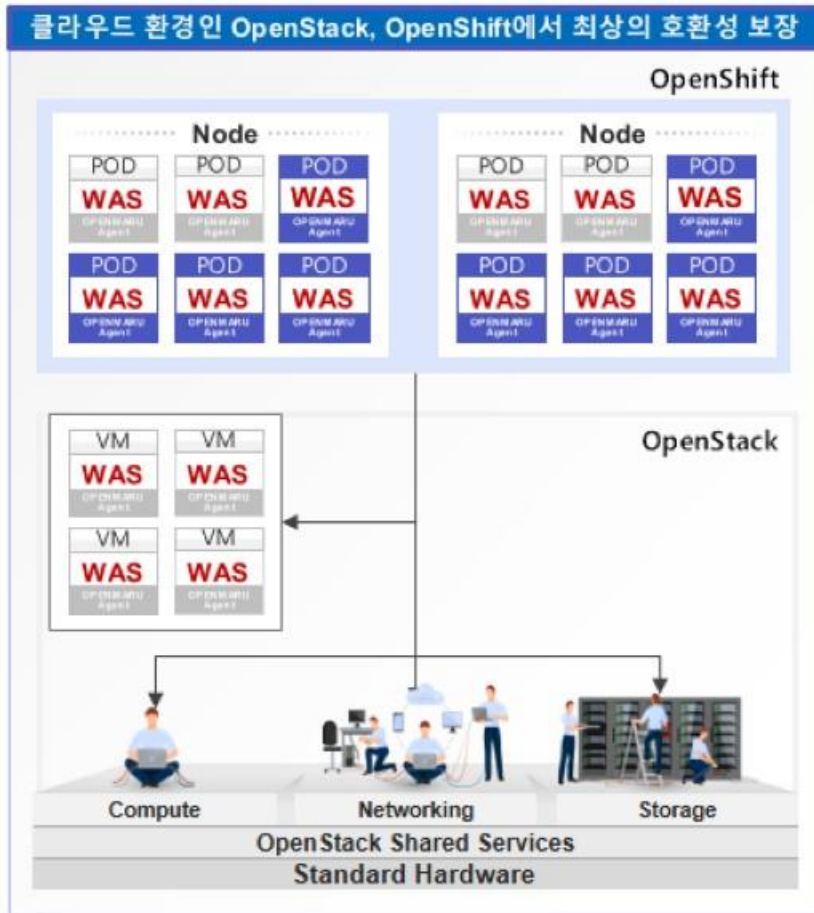


# 클라우드 환경에 최적화된 APM



## 클라우드 운영환경, 개발,검증 환경에 최적화된 기능 제공

- 가상화(OpenStack 등), Kubernetes(OpenShift)에 최적화된 APM 입니다. 다수의 대규모 컨테이너 환경에 적용되어 적합성을 이미 검증 받았습니다.
- OPENMARU APM은 클라우드 환경(VM/Container)에 유리한 라이선스 정책을 적용하고 있습니다.

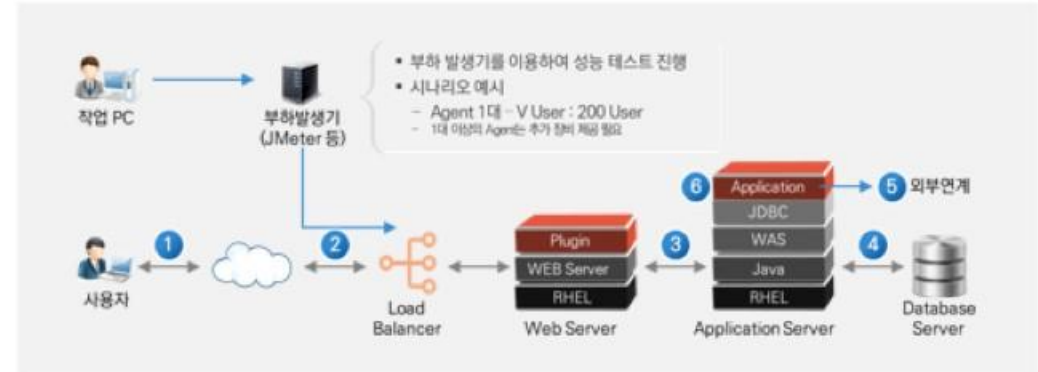
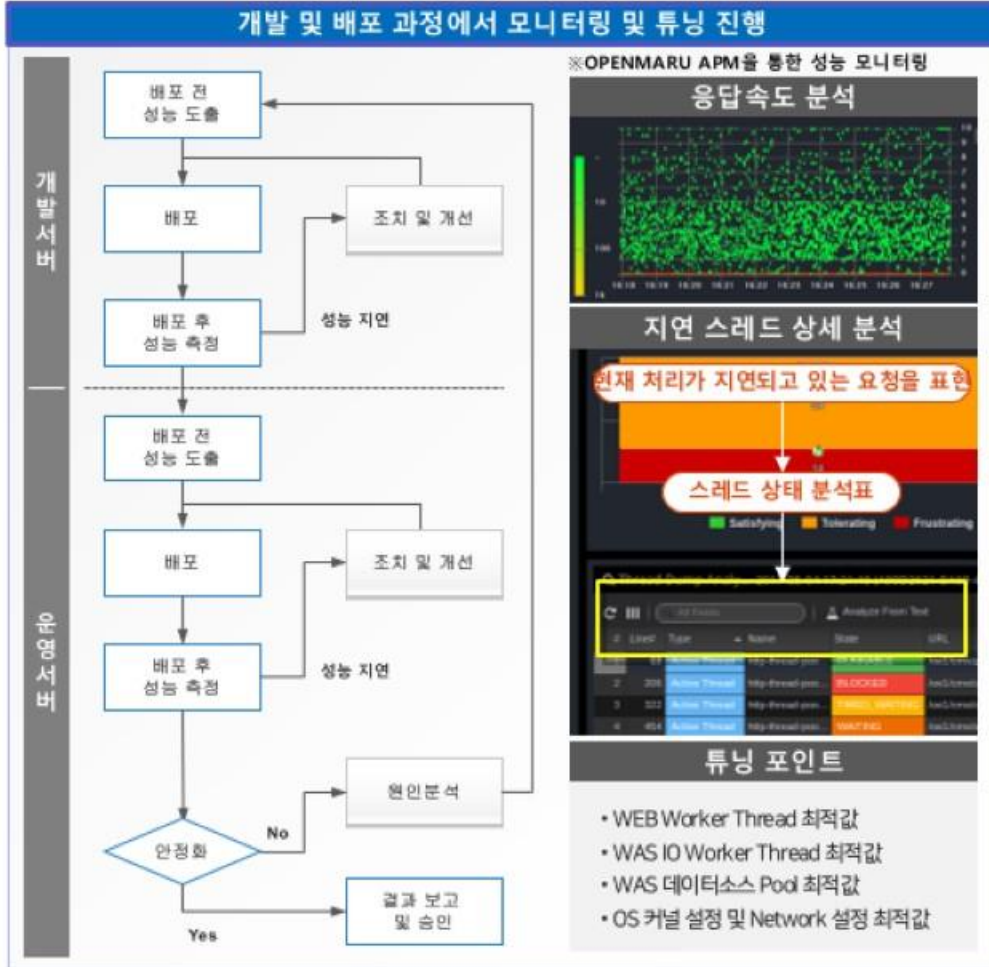


# 서비스 안정화 및 기술지원



➔ APM은 서비스 개발 및 오픈이후 안정화까지 모니터링과 튜닝이 필요합니다.

● OPENMARU APM을 활용하여 개발 시점부터 오픈 시점까지 지속적인 모니터링 및 튜닝을 시행하여 시스템의 안정적 오픈과 운영을 지원합니다.



☑ 주요 구간별 확인 및 튜닝 포인트

1	사용자 - 방화벽 및 L4(LB)	<ul style="list-style-type: none"> <li>▪ 방화벽</li> <li>▪ 대역폭 확인</li> </ul>
2	방화벽 및 L4(LB) - WEB	<ul style="list-style-type: none"> <li>▪ 목적지 연결 설정</li> <li>▪ 로드 밸런싱 옵션 (Client IP Hash)</li> </ul>
3	WEB - WAS	<ul style="list-style-type: none"> <li>▪ MEM Worker Thread 튜닝</li> <li>▪ ModJk 연결 옵션 튜닝</li> <li>▪ ModJk Sticky 옵션</li> </ul>
4	WAS - DBMS	<ul style="list-style-type: none"> <li>▪ IO Worker Thread 튜닝</li> <li>▪ DB Connection Pool 튜닝</li> <li>▪ DB Statement Cache 튜닝</li> <li>▪ JVM Heap Memory 튜닝</li> </ul>
5	WEB/WAS - NAS	<ul style="list-style-type: none"> <li>▪ IOPS, 지연 시간 확인</li> </ul>
6	WAS - 외부 인터페이스	<ul style="list-style-type: none"> <li>▪ HTTP API Connection Pool 튜닝</li> </ul>
7	애플리케이션	<ul style="list-style-type: none"> <li>▪ 오류 확인</li> <li>▪ 비즈니스 로직 성능 확인</li> </ul>

# APM 사용자 온라인 매뉴얼



## OPENMARU APM 사용자를 위한 온라인 매뉴얼 제공

- OPENMARU APM의 사용자 가이드 등 문서를 온라인 매뉴얼로 제공합니다.
- 또한, "거침없이 배우는 JBoss" 책을 웹 사이트에서 제공하고 있습니다.

### 온라인 매뉴얼 사이트 - <https://www.openmaru.io/docs>



OPENMARU APM은 고객의 원활한 서비스 운영을 위하여 적극적인 기술지원을 약속드립니다.

에이전트 설치 가이드

OPENMARU APM에 연결하기 위한 WAS, System 에이전트의 설치 방법을 안내합니다.

→ 바로가기

사용자 가이드 문서

OPENMARU APM을 사용하여 웹 애플리케이션 서버(WAS) 및 시스템 모니터링 추가 위한 사용자 안내서입니다.

→ 바로가기

지원 환경

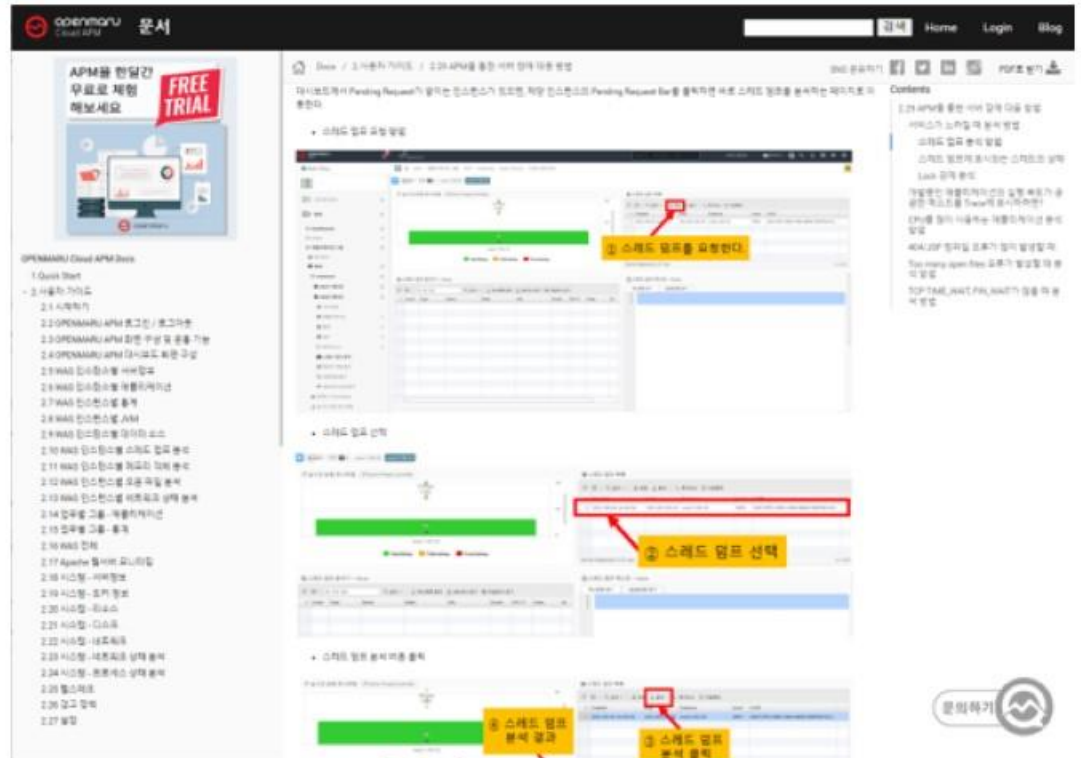
OPENMARU APM에서 지원하는 환경에 대한 안내입니다.

→ 바로가기

거침없이 배우는 JBoss

OPENMARU에서 제공하는 JBoss (AP) 관련 매스탈을 위한 책도 있습니다.

상당하기



문의하기

# Google Analytics 방식의 사용자 통계 적용



🔄 구글 애널리틱스와 동일한 사용자 측정 알고리즘을 이용한 방문자 수 계산 방법을 제공합니다.

- 구글 애널리틱스에서 활용한 알고리즘을 활용한 액티브 사용자 수 기능 제공, 사용자가 여러 WAS 인스턴스에 동시 접속하더라도 동일 사용자로 계산함
- 일일(DAU)/주간(WAU)/월간(MAU) 사용자 분석 기능 제공
- 로그인 세션 아이디를 기반으로 사용자 수를 계산하여 제공





# 마이크로 서비스(MSA) 호출 추적 기능(1/2)



➔ 마이크로 서비스(MSA)와 같이 애플리케이션이 분산되어 있는 환경에서 호출된 API를 추적하는 기능 제공

- MSA 환경에서 업무간 호출되는 REST API(HTTP) 호출에 대해 추적하는 기능 제공
- 호출관계를 추적하여 MSA 애플리케이션의 상세 수행을 확인할 수 있습니다.

**Child Transaction Detail**

Agent	Server IP	Client IP	Instance ID	URL	Status	Wta	Duration(ms)	SQL	Fetch	Sub	Count
WAS	10.128.2.102	192.168.48.10	api-gw-4-9c	/gateway.jsp	200	0	4,784				0
WAS	10.128.2.102	192.168.48.10	api-gw-4-9c	/gateway.jsp	200	0	4,794				0
WAS	10.128.2.102	192.168.48.10	api-gw-4-9c	/gateway.jsp	200	0	4,790				0

분석하고자 하는 MSA 시작점 Transaction

Transaction Detail

Method Trace

Num	Start Time	Elapsed	S	EndTime	A-Gab	CPUSlow	Method Call
1	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
2	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
3	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
4	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
5	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
6	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
7	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
8	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
9	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
10	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()

분석하고자 하는 Child Transaction

Child Transaction Detail

**Child Transaction Detail**

Agent	Server IP	Client IP	Instance ID	URL	Status	Wta	Duration(ms)	SQL	Fetch	Sub	Count
WAS	10.128.2.101	api-gw-4-9c	/	200	0	4,979					0
WAS	10.128.2.104	api-gw-3-4d1	/	200	0	4,797					0

분석하고자 하는 Child Transaction

Transaction Detail

Method Trace

Num	Start Time	Elapsed	S	EndTime	A-Gab	CPUSlow	Method Call
1	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.catalina.connector.ResponseFacade.getResponse()
2	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
3	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
4	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
5	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
6	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
7	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
8	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
9	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()
10	2022-07-06 00:04:31	0.000	0	2022-07-06 00:04:31	0.000	0.01	org.apache.jasper.runtime.BodyContentImpl.fillBodyContent()

분석하고자 하는 Child Transaction

Child Transaction Detail

**Child Transaction Detail**

Agent	Server IP	Client IP	Instance ID	URL	Status	Wta	Duration(ms)	SQL	Fetch	Sub	Count	Ext. Time	CPUSlow	Start Time
WAS	10.127.1.63	api-gw-3-4d1	/	200	0	10	0				0	0	0	2022-07-06 13:04:36

분석하고자 하는 MSA 끝단의 Transaction

Transaction Detail

Method Trace

Num	Start Time	Elapsed	S	EndTime	A-Gab	CPUSlow	Method Call
1	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
2	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
3	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
4	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
5	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
6	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
7	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
8	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
9	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()
10	2022-07-06 13:04:36	0.000	0	2022-07-06 13:04:36	0.000	0.00	org.apache.catalina.connector.ResponseFacade.getResponse()

끝단 Transaction의 쿼리 등 Detail

Child Transaction Detail



# 마이크로 서비스(MSA) 호출 추적 기능(2/2)

🔄 마이크로 서비스(MSA)와 같이 애플리케이션이 분산되어 있는 환경에서 호출된 API를 추적하는 기능 제공

- MSA 환경에서 REST API 호출 관계를 트리 형태로 보여주는 기능 제공
- REST API 호출관계를 트리형태로 표현합니다.

**Relationship**

Transactions > 2022-07-07 21:41:41 ~ 2022-07-07 21:41:58 > 8312ms ~ 9064ms

분석하고자 하는 MSA 시작점 Transaction

Agent	Server IP	Client IP	Instance ID	URL	Status	Wa	Duration	SQL Time(ms)	Fetch Gab	Fetch Count	Ext. Time	CPU(ms)	Start Time
WAS	10.128.2.102	192.168.65.11	api-ga-4-8g	/gateway.jsp	200	0	8,768	0	0	0	8,767	1.76	2022-07-08 10:41:3
WAS	10.128.2.102	192.168.65.11	api-ga-4-8g	/gateway.jsp	200	0	8,408	0	0	0	8,407	1.71	2022-07-08 10:41:3
WAS	10.128.2.102	192.168.65.11	api-ga-4-8g	/gateway.jsp	200	0	8,919	0	0	0	8,918	1.57	2022-07-08 10:41:4
WAS	10.128.2.102	192.168.65.11	api-ga-4-8g	/gateway.jsp	200	0	8,408	0	0	0	8,407	1.84	2022-07-08 10:41:4

Transaction Detail

Request URL: GET /gateway.jsp

Request Host: api-gateway.egov.apps.hanbit.egov.com

Client IP: 192.168.65.11

User Agent: Apache/2.4.18.1 (Ubuntu)

Start Time: 2022-07-07 21:41:37.480

End Time: 2022-07-07 21:41:46.246

Duration (ms): 8,768

CPU Time (ms): 1.76

ACDC Time (ms): 0

Thread Name: http-ssl-800-exec-12

IP: 10.128.2.102

Instance ID: api-ga-4-8g

Agent Type: WAS

Transaction ID: 1067036c-39a1-4309-9030-1a4e

Method Trace

Legend: 0 ~ 1,000ms (Green), 1,000ms ~ 3,000ms (Yellow), 3,000ms ~ 5,000ms (Red)

트랜잭션간 맺어진 Relationship

**Relationship**

Relationship을 맺은 모든 MSA 트랜잭션이 한번에 출력, Tree 구조로 호출관계 표시

Agent	IP address	Instance ID	URL	Status	Wa	Duration	SQL Time(ms)	Fetch Gab	Fetch Count	Ext.
WAS	10.128.2.10	api-ga-4-8g	/gateway.jsp	200	0	8,281	0	0	0	0
WAS	10.131.1.91	api-st-4-m5	/	200	0	3,948	0	0	0	0
WAS	10.131.1.65	egov-5-lmsr	/	200	0	18	5	0	0	0
WAS	10.128.2.10	api-co-3-4r5	/	200	0	4,314	0	0	0	0
WAS	10.131.1.65	egov-5-lmsr	/	200	0	12	3	0	0	0

Transaction Detail

Trace

Stack Trace

```

[ 27][21:48:53.680] 0| 0| 0| 0| 0.0] + org.postgresql.jdbc.PgConnection.prepareStatement()
[ 28][21:48:53.680] 0| 0| 0| 0| 0.0] + org.postgresql.jdbc.PgConnection.prepareStatement()
[ 29][21:48:53.680] 0| 0| 0| 0| 0.0] + org.postgresql.jdbc.PgConnection.prepareStatement()
[ 30][21:48:53.680] 1| 0| 0| 1| 0.1] + org.apache.commons.dbcp.DelegatingPreparedStatement.execute()
[ 31][21:48:53.680] 1| 0| 0| 1| 0.1] + org.postgresql.jdbc.PgPreparedStatement.execute()
[ 32][21:48:53.680] 1| 0| 1| 1| 0.1] + org.postgresql.jdbc.PgPreparedStatement.execute()

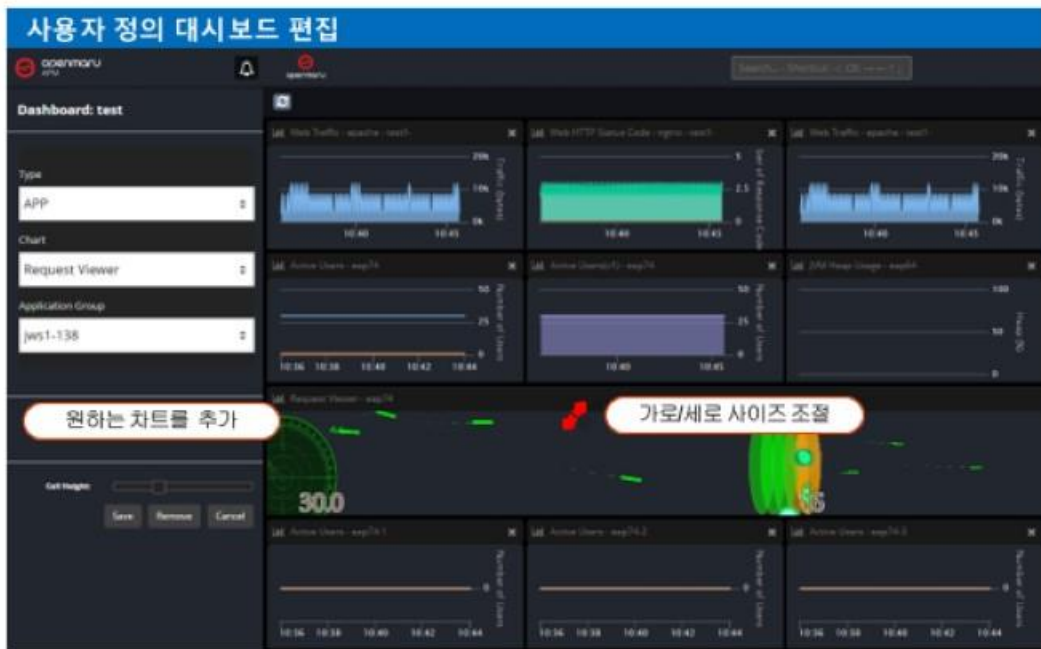
> DB
+ Query
INSERT INTO CONTWEBLOG (REQUEST_ID, URL, REQUESTER_ID,
/, /, ?, NEW())
+ Params
+ Params
WEBLOG_0000000270330', /index.do', '127.0.0.1'
INSERT INTO CONTWEBLOG (REQUEST_ID, URL, REQUESTER_ID,
    
```

# 사용자 정의 대시보드 기능 지원



## 🔄 사용자 목적에 맞게 대시보드를 구성할 수 있는 기능제공

- N개의 애플리케이션 그룹 및 개별 인스턴스 차트를 별도로 구분하여 사용자가 원하는 그래프들을 대시보드로 정의하여 모니터링 할 수 있는 기능 제공
- 대시보드 모니터 환경에 맞게 세부적으로 차트 사이즈를 조절 할 수 있는 기능 제공





# 빠른 인스턴스 검색 기능 제공

## ☞ 모니터링 대상 인스턴스 이름 및 IP로 빠른 검색 기능 제공

- 인스턴스 이름 검색을 통해 빠르게 메뉴를 이동 할 수 있는 기능을 제공
- 컨테이너 환경에서 삭제되어 숨겨진 인스턴스를 검색하고 곧바로 활성화하여 분석 할 수 있는 기능 제공

The screenshot displays the Openmaru APM interface with a search dropdown menu open. The menu lists several instance names under the path 'APP > jws1-138':

- APP > jws1-138
- WAS > jws1-138-87g94
- WAS > jws1-138-bhkjw
- WAS > jws1-138-pv6mm
- WAS > jws1-138-qjs24
- WAS > jws1-138-z2nlk

A search input field contains the text 'jws1-138'. A tooltip above the search field reads: '엔터키를 입력하면 숨겨진 인스턴스를 포함하여 검색' (Pressing the enter key will search for hidden instances). A secondary dropdown menu is visible for the selected instance '10.131.1.110[jws1-138-87g94] > jws1-138-87g94', listing metrics such as '서버 정보' (Server Info), '초당처리수(TPS)' (TPS), '메모리 크기' (Memory Size), and '스레드 덤프 분석' (Thread Dump Analysis).

The background interface shows a 'Request Velocity' chart with a value of 78.0, an 'APDEX' chart with a value of 41, and a summary dashboard with the following metrics:

Metric	Value
APDEX %	93.4 %
Active Users	76.0
TPS	376.5
Error Rate %	0.0 %
Average Response Time	989 ms



# 실행중인 트랜잭션에 대한 모니터링 및 제어기능

➡ 현재 실행중인 트랜잭션의 호출정보, 쿼리 정보를 모니터링하고 스레드를 중지하는 기능 제공

- 스레드 덤프를 생성하지 않고 실시간으로 트랜잭션을 모니터링 하는 기능 제공
- 실행중인 스레드 리스트, Current Stack(스택 트레이스), Transaction Trace(SQL, HTTP 등) 트랜잭션 기본 정보를 제공



### 현재 실행중인 스레드 목록

Thread Name	Status	URL	Duration(ms)
ajp-bio-8011-exec-2907	RUNNABLE	/okp/service/entFileDownload	1,834,275,665
ajp-bio-8011-exec-4188	RUNNABLE	/okp/ent/selectedMailView	1,834,141,980
ajp-bio-8011-exec-4331	RUNNABLE	/okp/ent/insertSendMail	1,834,290,686
ajp-bio-8011-exec-4343	RUNNABLE	/okp/ent/selectedMailWrite	1,834,281,251
ajp-bio-8011-exec-4303	RUNNABLE	/okp/ent/selectedMailPreview	1,834,140,747
ajp-bio-8011-exec-4336	RUNNABLE	/okp/ent/selectedMailView	1,834,144,007
ajp-bio-8011-exec-8563	RUNNABLE	/okp/ent/selectedMailView	1,758,681

실행중인 스레드에 대한 중지, 인터럽트 기능

### 현재 실행중인 스택 정보

Current Stack	Transaction Trace	Transaction Info
1 java.lang.Thread.State: RUNNABLE (RUNNABLE)		
2 java.net.SocketInputStream.socketRead0(Native Method)		
3 java.net.SocketInputStream.read(SocketInputStream.java:152)		
4 java.net.SocketInputStream.read(SocketInputStream.java:122)		
5 com.sun.mail.util.TraceInputStream.read(TraceInputStream.java:124)		
6 java.io.BufferedInputStream.fill(BufferedInputStream.java:235)		
7 java.io.BufferedInputStream.read(BufferedInputStream.java:254)		
8 com.sun.mail.iap.Response.readResponse(ResponseInputStream.java:95)		
9 com.sun.mail.iap.Response.<init>(Response.java:92)		
10 com.sun.mail.iap.Protocol.<init>(Protocol.java:60)		
11 com.sun.mail.iap.Protocol.<init>(Protocol.java:123)		
12 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
13 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
14 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
15 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
16 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
17 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
18 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
19 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
20 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
21 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
22 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
23 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
24 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
25 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
26 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
27 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		
28 com.sun.mail.iap.Protocol.<init>(Protocol.java:115)		

### 현재 실행중인 트랜잭션 정보

Current Stack	Transaction Trace	Transaction Info
1 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
2 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
3 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
4 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
5 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
6 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
7 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
8 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
9 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
10 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
11 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
12 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
13 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
14 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
15 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
16 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
17 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
18 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
19 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
20 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
21 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
22 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
23 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
24 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
25 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
26 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
27 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		
28 org.springframework.jdbc.datasource.DataSourceTransactionManager.doExecute()		

### 트랜잭션 URL 호출 정보

Current Stack	Transaction Trace	Transaction Info	
Name	Value	Name	Value
Request URL	POST /okp/ent/selectedMailWrite	Params	
Request Host	mg.openmaru.com	Status Code	200
Client IP	192.168.13.221	User Agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; NET CLR 2.0.50727; NET CLR 3.5.30729; NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Start Time	2018-09-11 09:57:28.096	End Time	2018-09-11 09:57:28.096
Duration (ms)	1,834,281,251	CPU Time (ms)	0.00
Jobc Time (ms)	1	Latency (ms)	0



# LDAP연동 로그인, Java 11, Spring Boot 지원

➡ 현재 실행중인 트랜잭션의 호출정보, 쿼리 정보를 모니터링하고 스레드를 중지하는 기능 제공

- 최신 Java 11 버전을 지원
- LDAP 및 Active Directory 를 이용한 APM 로그인 연동을 지원
- Spring Boot Embedded Dasource HikariCP 모니터링 지원

## 지원 Java 버전 업데이트



Spring Boot의 기본 Datasource인 HikariCP 모니터링

### LDAP / Active Directory와 APM 로그인 연동 기능

LDAP

LDAP 사용여부: TRUE

APM 유저 LDAP 인증 사용여부

LDAP 서버 URL: ldap://192.168.23.22:389

Enter LDAP server's URL  
ex) ldap://10.10.10.10:389 or ldaps://10.10.10.10:636

Connect 타임아웃: 3,000

LDAP 서버에 대한 Connect 타임아웃(ms)

Read 타임아웃: 3,000

LDAP 서버에 대한 Read 타임아웃(ms)

SSL 사용여부: FALSE

LDAP 연결에 SSL 사용여부

유저 DN 접두사(Prefix): uid=

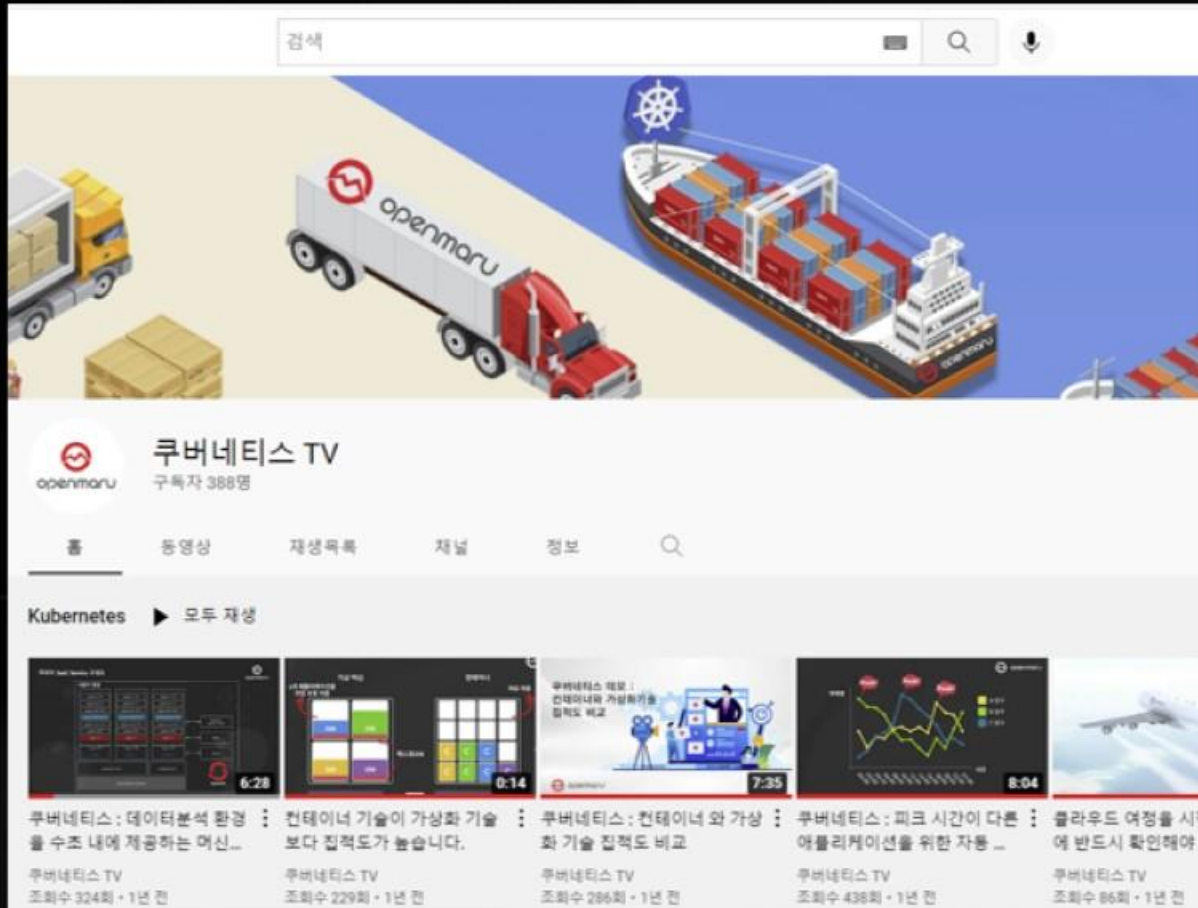
유저 DN(Distinguished Names) 접두사(Prefix). ex) uid=

LDAP/Active Directory과 연동한 로그인 기능 제공

# Cloud Native Channel – Kubernetes TV by OPENMARU



- Cloud Native 에 중심인 기술 기업
- 2022년에만 10회 이상의 PaaS On line Seminar



Kubernetes TV on YOUTUBE



Cloud Native Seminar - TalkIT



Cloud Seminar for LGCNS



Cloud Native Seminar - Youtube



Cloud Seminar for 국민연금공단



openmaru



Application Performance Management

감사합니다.



openmaru  
APM