


클라우드 네이티브 개념 이해

A wide-angle photograph of a cable-stayed bridge spanning a body of water at sunset. The sun is low on the horizon, creating a bright, shimmering reflection on the water's surface. The bridge's two tall, slender pylons are silhouetted against the sky, with numerous stay cables fanning out to support the bridge deck. In the background, a prominent, dark, pointed skyscraper stands out against the twilight sky. The overall atmosphere is serene and dramatic, with a mix of warm golden light and cooler, darker tones in the sky and water.

클라우드 네이티브 개념과 기술요소들

Cloud Native

클라우드 네이티브?

클라우드 네이티브
어플리케이션이란?

클라우드
네이티브 효과란?

클라우드 네이티브
기술은 뭐가 필요하지?

하이브리드
클라우드 어떻게
하면 좋을까?

클라우드 단어도 많이
회자되고 있는데요?
클라우드 네이티브는?

클라우드 기술의
표준은 없나요?

유튜브를 보고, 검색을
해봐도 대략은
알겠는데 정확히
뭔지 모르겠음!!



Development Process



WATERFALL



AGILE



DEVOPS



Application Architecture



MONOLITHIC



N-TIER



MICROSERVICES



Deployment & Packaging



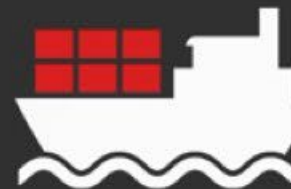
PHYSICAL SERVERS



VIRTUAL SERVERS



CONTAINERS



Application Infrastructure



DATA CENTER



HOSTED



CLOUD



CNCF Cloud Native Definition v1.0

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.

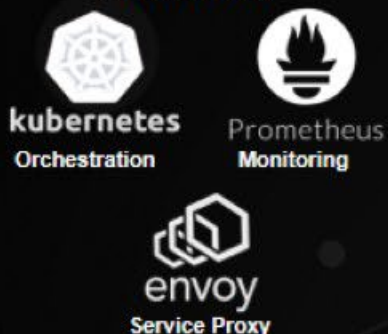
우리 재단은 최신 기술 수준의 패턴을 대중화하여 이런 혁신을 누구나 접근 가능하도록 한다.



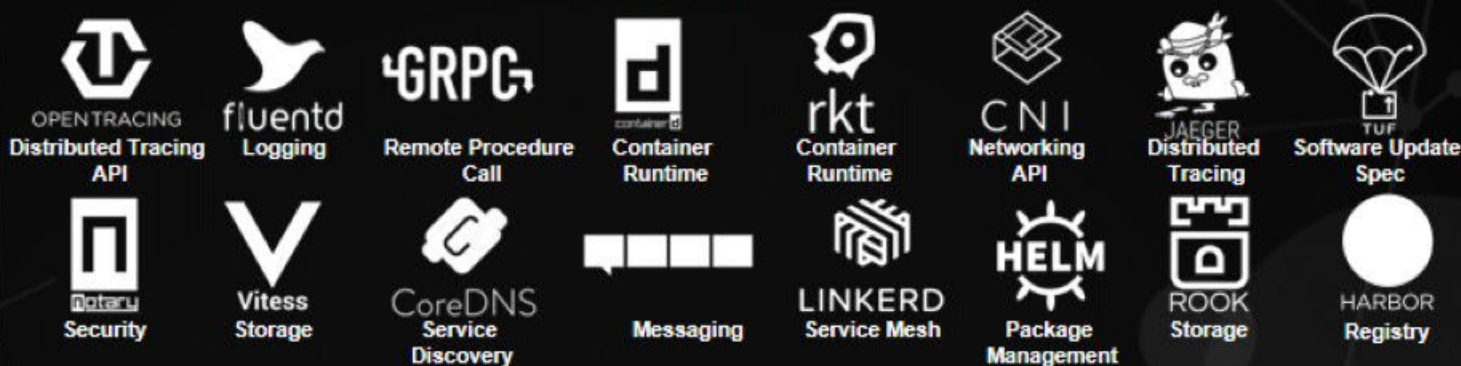
Cloud Native Computing Foundation

- Non-profit, part of the Linux Foundation; founded Dec 2015

Graduated



Incubating



- Platinum members:



before containerization

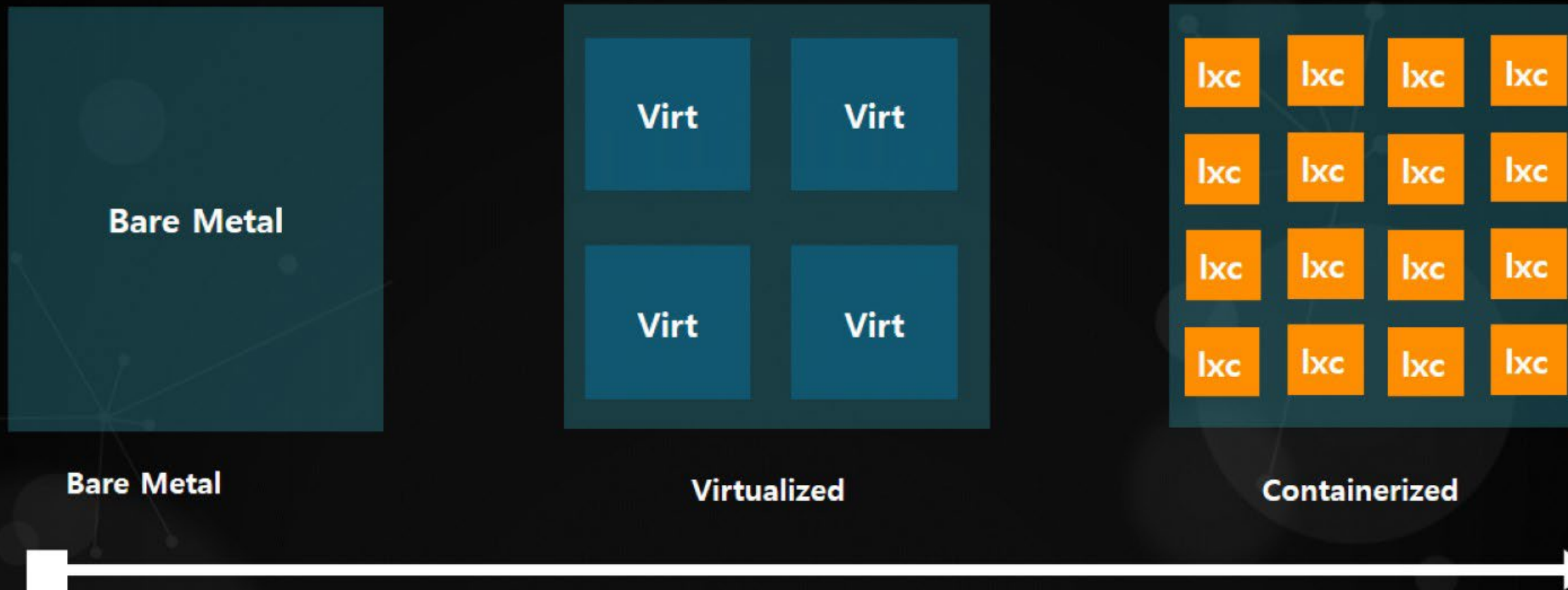


After Containerization

컨테이너에 화물을 싣고

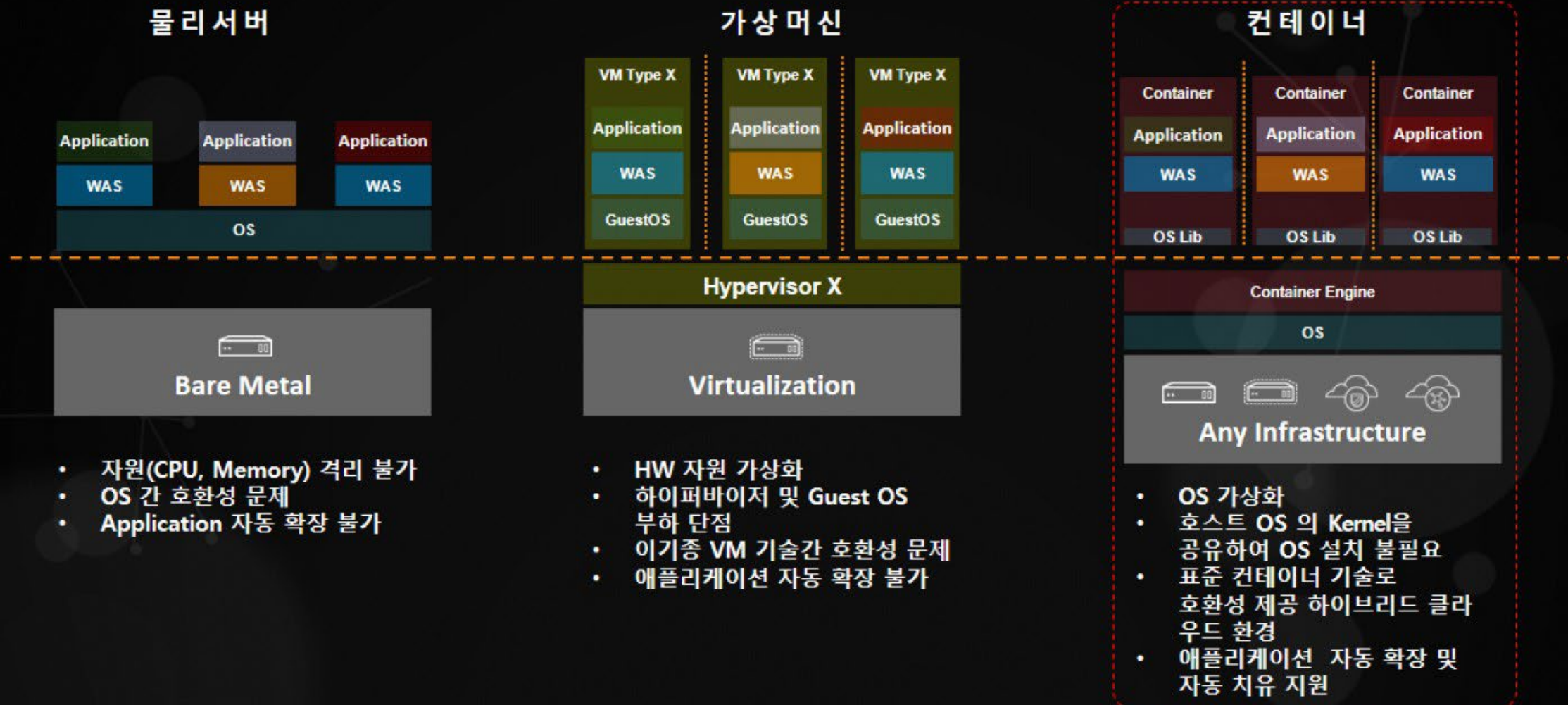


Evolution of Infrastructure Architectures



WHY CONTAINER ?

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



- 자원(CPU, Memory) 격리 불가
- OS 간 호환성 문제
- Application 자동 확장 불가

- HW 자원 가상화
- 하이퍼바이저 및 Guest OS 부하 단점
- 이기종 VM 기술간 호환성 문제
- 애플리케이션 자동 확장 불가

- OS 가상화
- 호스트 OS 의 Kernel을 공유하여 OS 설치 불필요
- 표준 컨테이너 기술로 호환성 제공 하이브리드 클라우드 환경
- 애플리케이션 자동 확장 및 자동 치유 지원

클라우드 네이티브 개념과 기술요소들

클라우드 네이티브를 위한 준비

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



클라우드 핵심 개념 : Architecture & Model



클라우드 네이티브


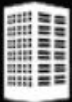

vs.



클라우드 기반



Cloud Delivery Model

구분	클라우드 네이티브	구조	특징	벤더
	Public Cloud	클라우드 서비스 업체가 인터넷을 통해 컴퓨팅 리소스를 제공하고, 서버의 유지 관리	<ul style="list-style-type: none"> • 여러 기업이 하나의 클라우드 인프라를 이용 (Multi-Tenant) • 더 적은 구축 비용 • 더 적은 유지 보수 	<ul style="list-style-type: none"> • AWS • Azure • Google Cloud • Oracle • Alibaba
	Private Cloud	기업이 클라우드 서버를 독점 아키텍처를 이용하여 자사의 데이터 센터 운영	<ul style="list-style-type: none"> • 하나의 기업에 하나의 인프라스트럭처 (싱글 테넌트) • On Premise 하드웨어 • 고객이 인프라 관리 	<ul style="list-style-type: none"> • HPE • VMWare • Dell EMC • IBM • Red Hat
	Hybrid Cloud	On Premise 인프라, 프라이빗 클라우드 과 퍼블릭 클라우드의 혼합 컴퓨팅 환경	<ul style="list-style-type: none"> • 데이터 처리를 더 • 개인 및 제어 가능 • 버스트 기능 • 기존 시스템을 함유 할 수있다 	<ul style="list-style-type: none"> • Red Hat • AWS • Azure • Google Cloud

클라우드 서비스 모델



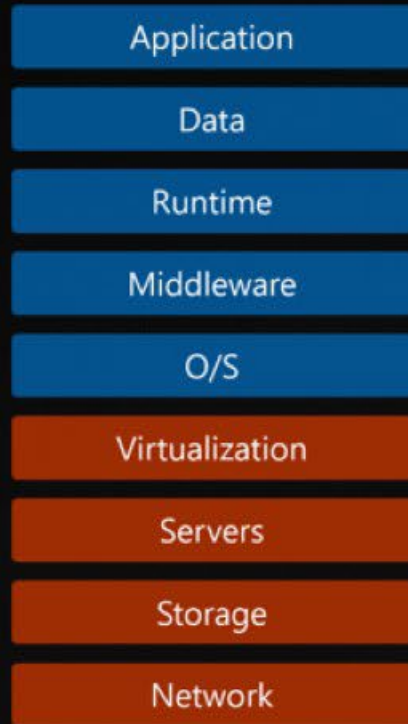
On-Premise



- 고객은 인프라 제공, 유지 및 애플리케이션 호스팅 모두 책임



Infrastructure-as-a-Service (IaaS)



- 공급 업체는 인터넷을 통해 컴퓨팅 인프라를 제공
- 예 : AWS EC2, MSFT Azure



Platform-as-a-Service (PaaS)



- 애플리케이션 개발을 위한 플랫폼을 제공
- 공급 업체는 서버, 스토리지, 네트워크를 관리
- 고객은 애플리케이션 관리
- 예 : Salesforce Lightning, Heroku



Software-as-a-Service (SaaS)



- 인터넷을 통해 제공되는 소프트웨어
- 공급 업체가 소프트웨어 구축, 유지, 운영
- 예 : G-suite, Microsoft 365

범례 :

기업 고객 관리

클라우드 공급자 관리

클라우드 네이티브 개념과 기술요소들

IaaS with Virtualization vs. PaaS with Container

가상화 기반 IaaS vs. 컨테이너 기반 PaaS

기존 IaaS (가상화 기술 기반)

가상화 기반 전용 클라우드

상용 SW 기반

규모의 경제

기술 및 공급자



향후 PaaS (컨테이너 기술 기반)

하이브리드
(전용+공용, 물리+가상)

오픈소스 SW 기반

범위의 경제

비즈니스 및 수요자

컨테이너는 클라우드 에서 Java 와 같이 벤더 종속성 해제

2000 년 - Java 를 통한 Vendor Lock-In 해제



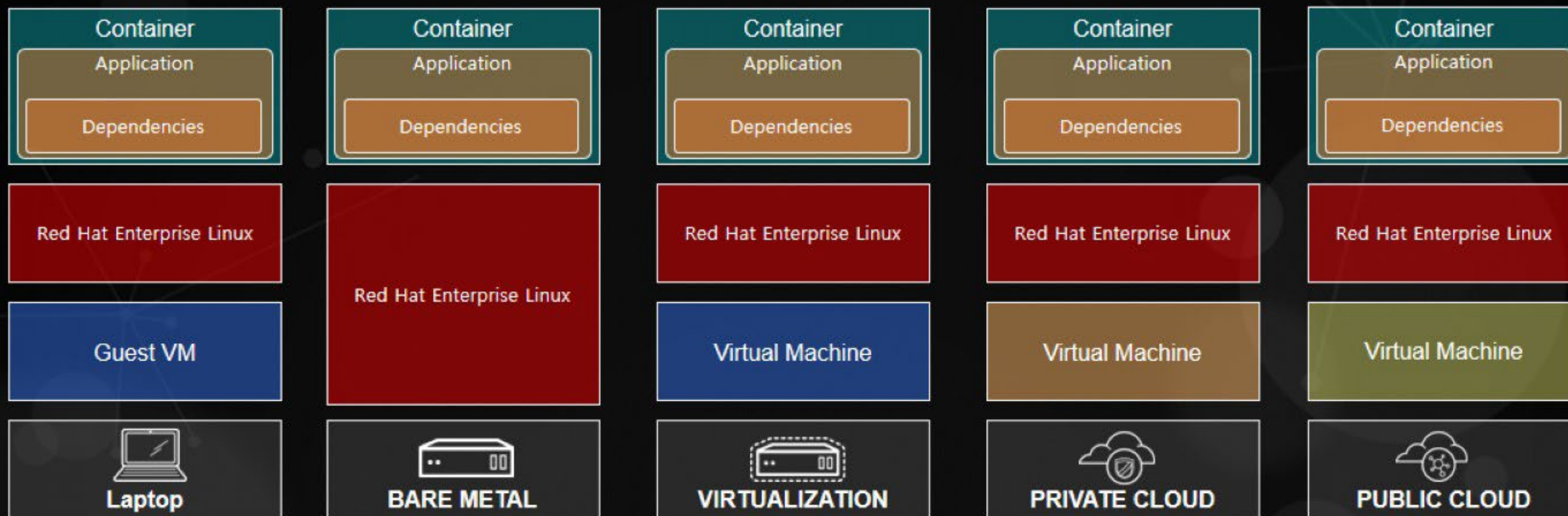
2020 년 - 컨테이너와 Kubernetes 를 통한 Vendor Lock-In 해제



Write Container, run anywhere!

컨테이너의 이동성

- Linux 커널 구조를 이용하고있다 = Linux 움직이는 모든 환경에서 이동성



클라우드 네이티브 개념과 기술요소들

Microservice Architecture

클라우드 애플리케이션 성숙도 단계

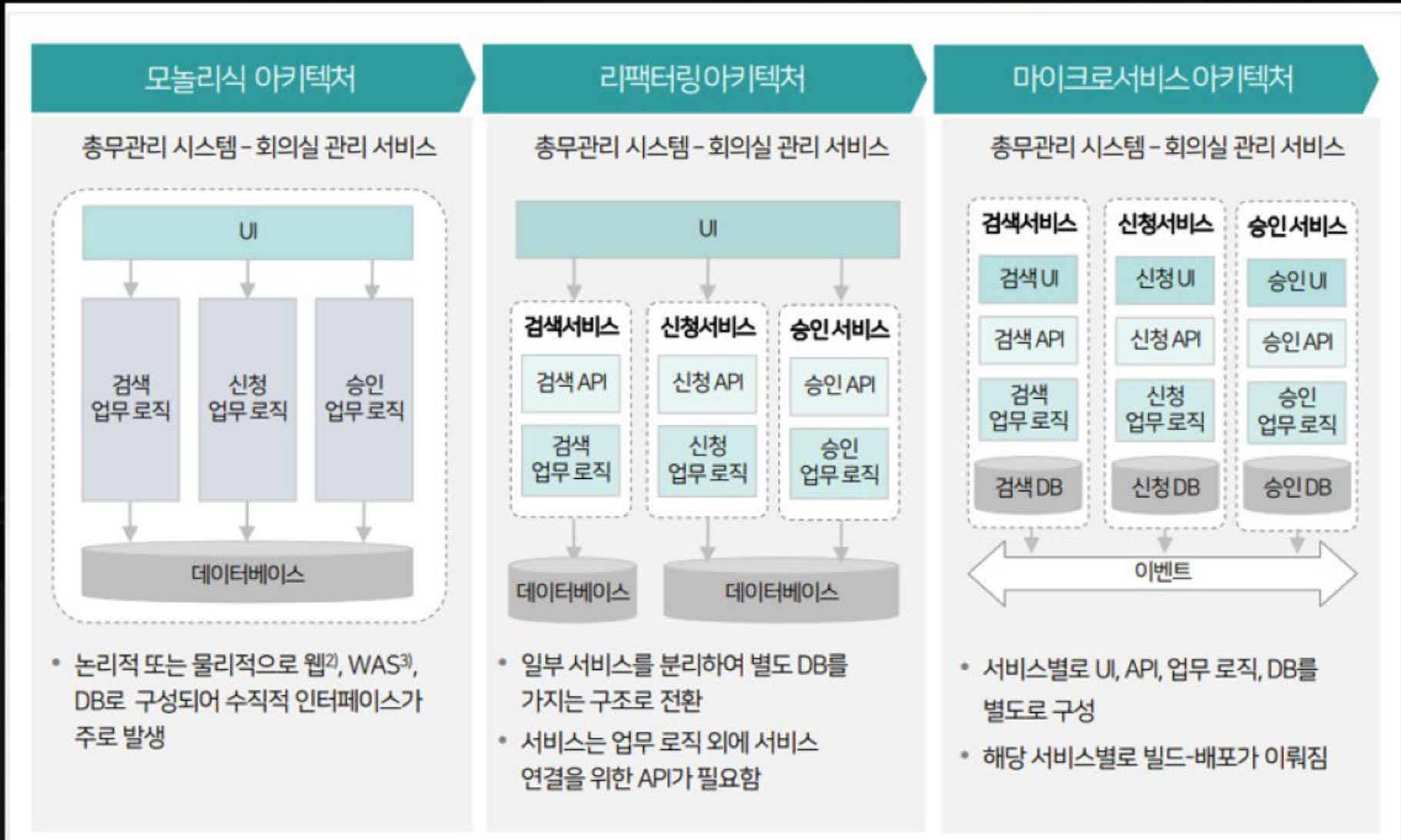
- 클라우드 네이티브는 컨테이너 기반의 PaaS로 시작하여, CI/CD, MSA 모두 포함된 단계
- Lv1. 클라우드 준비 단계 → Lv2. 클라우드 친화 단계 → Lv3. 클라우드 네이티브 단계
 - PaaS와 컨테이너를 도입하는 클라우드 친화 단계
 - 데브옵스, CI/CD, MSA를 적용하는 클라우드 네이티브 단계



한국지능정보사회진흥원 클라우드 네이티브 발주자 안내서

마이크로서비스 아키텍처로의 전환 예시

한국지능정보사회진흥원 클라우드 네이티브 발주자 안내서



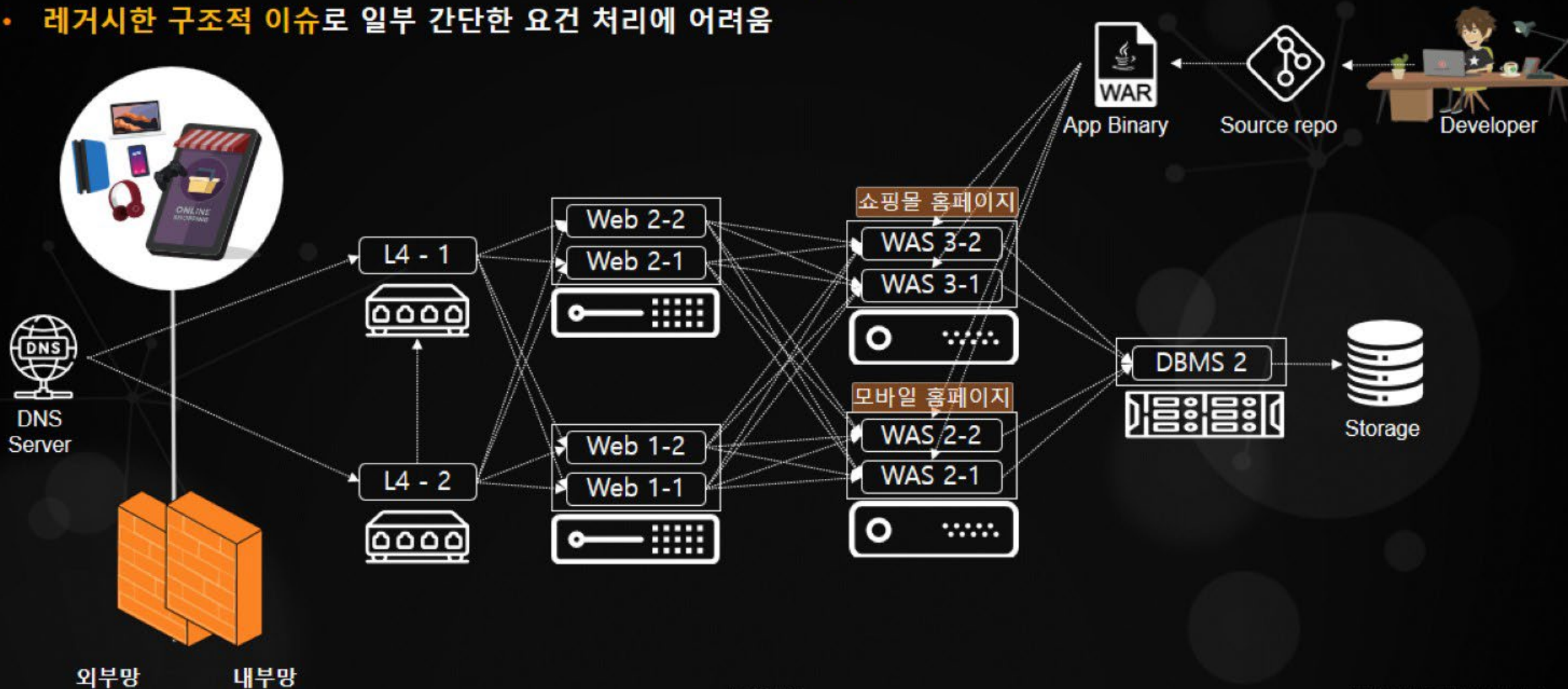
- 논리적 또는 물리적으로 웹²⁾, WAS³⁾, DB로 구성되어 수직적 인터페이스가 주로 발생

- 일부 서비스를 분리하여 별도 DB를 가지는 구조로 전환
- 서비스는 업무 로직 외에 서비스 연결을 위한 API가 필요함

- 서비스별로 UI, API, 업무 로직, DB를 별도로 구성
- 해당 서비스별로 빌드-배포가 이뤄짐

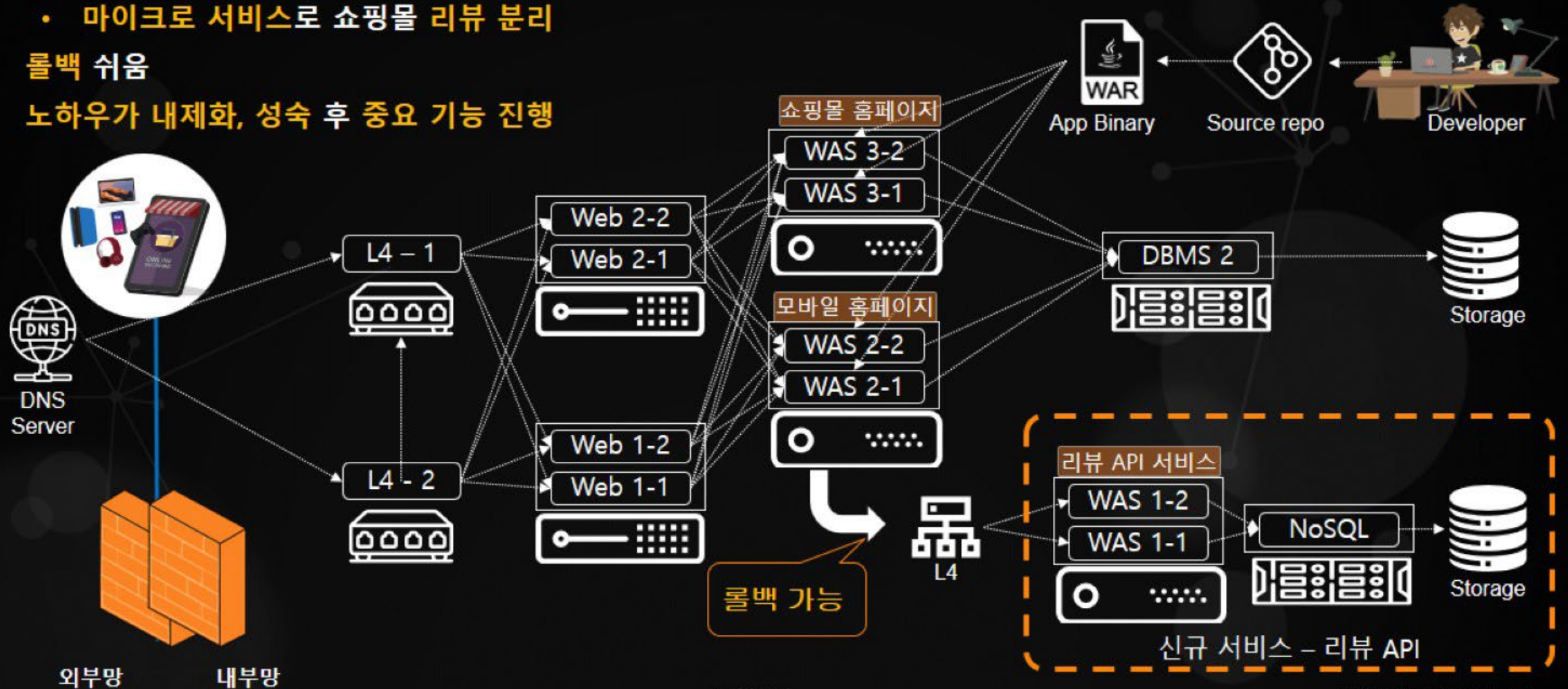
일반화 되어 있는 모놀리틱 아키텍처

- 웹서버, WAS 서버, 데이터베이스 의 역할 별로 티어를 나눈 3 티어 구조
 - 각 티어 별로 수동 확장과 관리, 오토스케일링 및 자동화 부족
- 레거시한 구조적 이슈로 일부 간단한 요건 처리에 어려움



Microservice Pilot – 리뷰API 서비스 도입

- 수많은 기능으로 동작하는 기존 서비스에서 **쉬운 부분부터 API로 분리**
 - 재설계 시 리스크 부담으로 시도 어려움
 - 마이크로 서비스로 쇼핑몰 리뷰 분리
- **롤백 쉬움**
- **노하우가 내제화, 성숙 후 중요 기능 진행**



기존 애플리케이션과 Cloud Native 애플리케이션 비교

분류	기존 애플리케이션	Cloud Native 애플리케이션
실행 환경	물리 서버 중심	컨테이너 중심
확장	Scale Up (수직 확장)	Scale Out (수평 확장)
결합	크고 조밀 결합	느슨하게 & 서비스 기반
인프라 의존성	인프라 의존	인프라 독립적으로 이동성 보장
Delivery 방법	폭포수형으로 장기간 개발	Agile & Continuous Delivery
개발 도구	로컬 IDE 개발 도구	클라우드 기반의 지능형 개발 도구
조직구조	사일로화 된 개발, 운영, 보안 팀	DevSecOps, NoOps 또는 협업

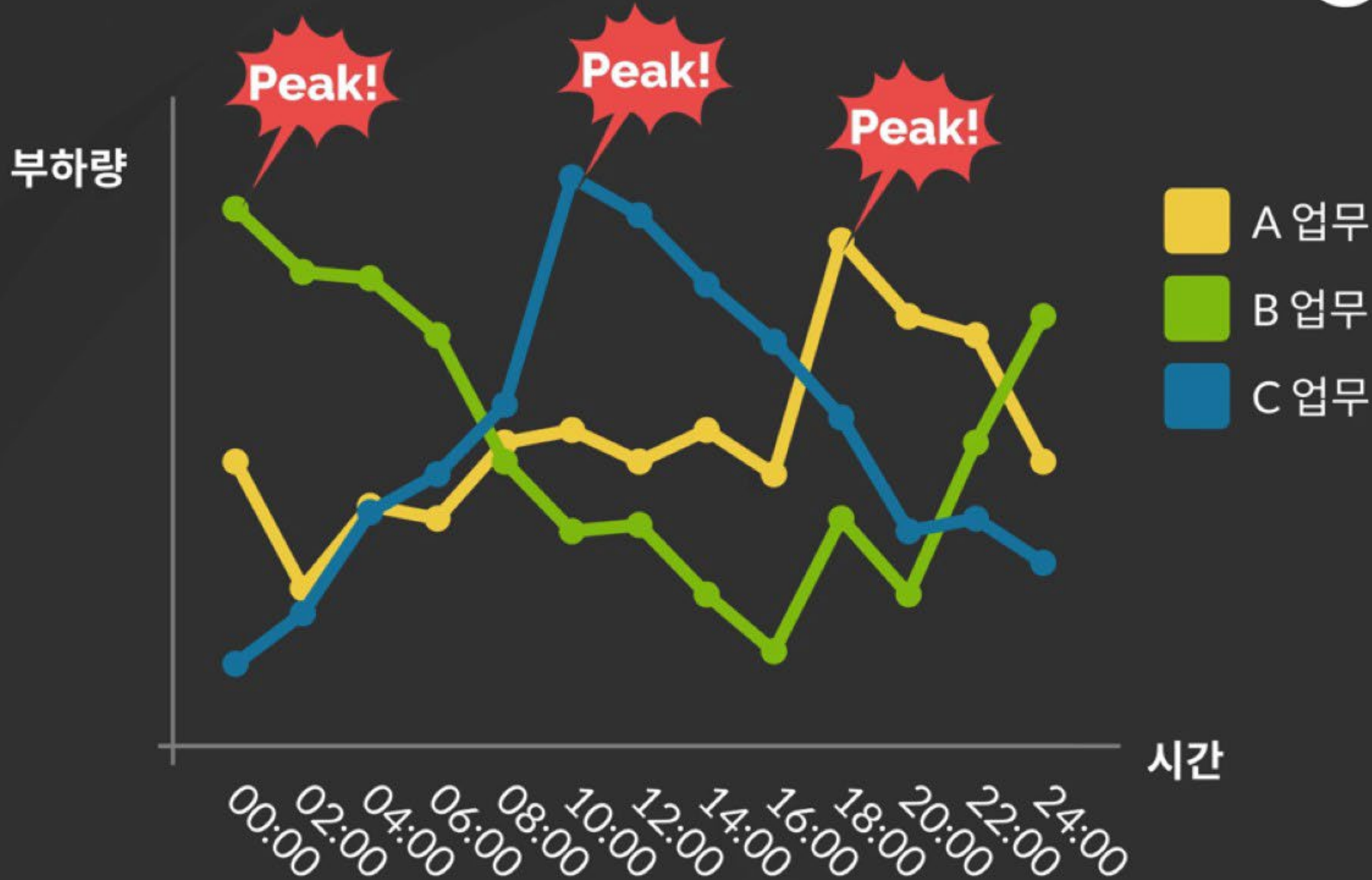


Application Performance Management

Peak 시점이 다른 애플리케이션의
Auto Scaling 모니터링 데모



openmaru
APM



Peak 시점 별 자동 자원할당 활용 예시

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



인스턴스 개수 한정된 환경(기존 시스템)

기존에는 가상화 환경으로 부하 상황에서 리소스를 효율적으로 활용할 수 없음



openmaru

Deployment Configs · Red Hat · POC - OPENMARU APM · POC - OPENMARU APM · +

주요 알림 | openmaru.poc/monitoring/console/#/was/dashboard-was?g=0.auto-scaling

openmaru APM | POC | 한국어

WAS / Dashboards / 대시보드

Request Viewer: Shot on, 763.0

Group	Steps	Time	Users
gro_4hjzk	0.5tps	0.00s	1u
gro_89j7t	0.0tps	0.00s	1u
gro_cqd92	0.5tps	0.01s	1u
gro_lbh58	0.0tps	0.00s	1u
gro_xrsxh	0.0tps	0.00s	1u
por_2xfkl	11.5tps	1.85s	158u
por_5ghkq	10.6tps	1.08s	161u
por_f4nm9	12.5tps	1.07s	156u
por_mlxwj	0.0tps	0.00s	158u
por_qr4t2	10.5tps	1.06s	146u

APDEX: 100.0% APDEX, 50.0% TPS, Active Users 721.2, Error Rate 0.0%, Average Response Time 756ms

Transaction Heatmap(T-Map): Response Time (s) vs Request Count

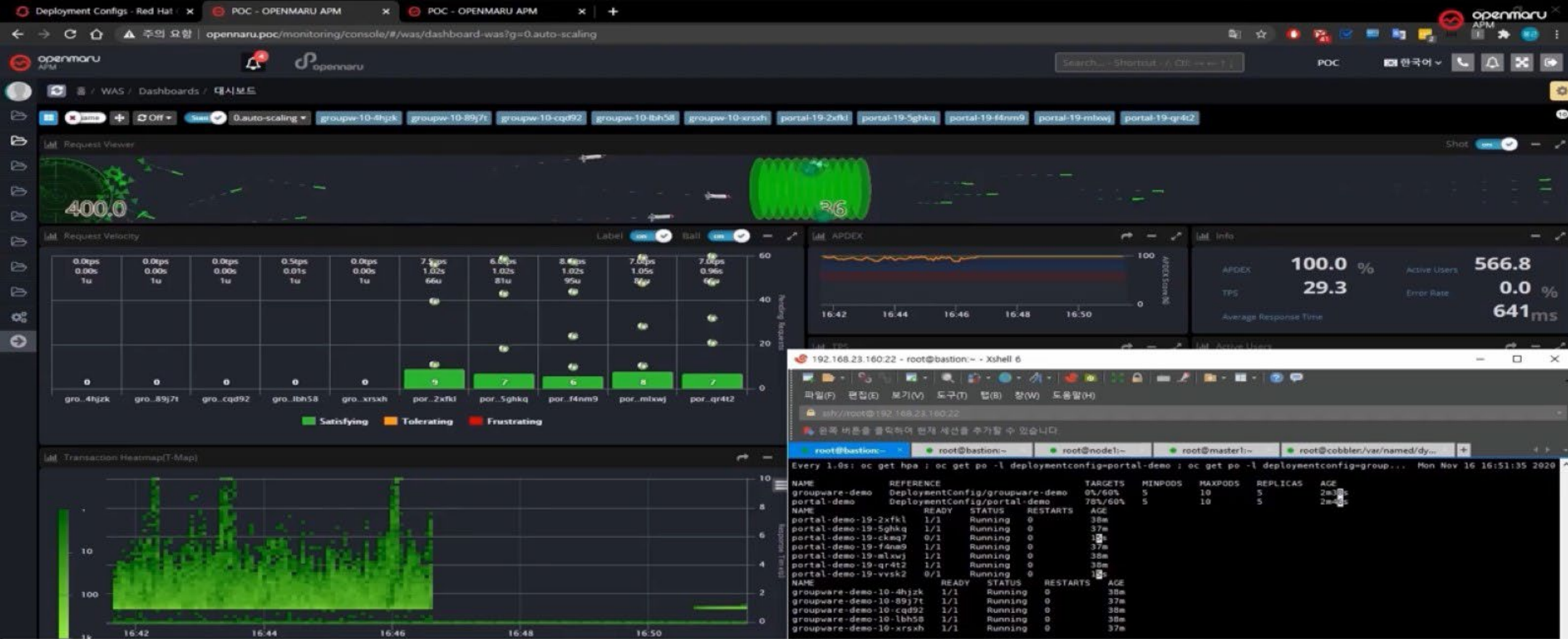
Apache JMeter (5.2.1): autoscaling.jmx (C:\Users\qhd\Downloads\apache-jmeter-5.2.1\bin\autoscaling.jmx)

jp@gc - Composite Graph: Active Threads Over Time (x10) vs Response Times

기초 하계에서 의적하 부하 발생

T-02 병원 업무 Peak일 때 자동부하분산 환경 부하테스트

부하에 따른 컨테이너 자동확장으로 응답시간 보장과 TPS 증가



동일한 애플리케이션에 Auto-Scale(5-10) 설정 후 동일한 부하 적용.

T-03 건강 검진 Peak일 때 자동부하분산 환경 부하테스트

부하에 따른 컨테이너 자동확장으로 응답시간 보장과 TPS 증가



Service	Request Velocity (TPS)	Request Velocity (u)
gro_4hjkz	9.0tps	1.07s
gro_89j7t	7.0tps	1.08s
gro_cqd92	8.5tps	1.09s
gro_lbh58	8.0tps	1.20s
gro_xrsxh	6.5tps	1.02s
por_2xfkl	0.5tps	0.00s
por_5ghkq	0.0tps	0.00s
por_14nm9	0.5tps	0.00s
por_mlxwj	0.5tps	0.00s
por_qr4t2	0.0tps	0.00s

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
groupware-demo	DeploymentConfig/groupware-demo	93%/60%	5	10	5	18m
portal-demo	DeploymentConfig/portal-demo	0%/60%	5	10	5	3m3s

NAME	READY	STATUS	RESTARTS	AGE
portal-demo-19-2xfkl	1/1	Running	0	54m
portal-demo-19-5ghkq	1/1	Running	0	52m
portal-demo-19-14nm9	1/1	Running	0	52m
portal-demo-19-mlxwj	1/1	Running	0	54m
portal-demo-19-qr4t2	1/1	Running	0	54m

NAME	READY	STATUS	RESTARTS	AGE
groupware-demo-10-4hjkz	1/1	Running	0	54m
groupware-demo-10-89j7t	1/1	Running	0	52m
groupware-demo-10-cqd92	1/1	Running	0	54m
groupware-demo-10-jz9bh	0/1	Running	0	10s
groupware-demo-10-jz6z	0/1	Running	0	10s
groupware-demo-10-lbh58	1/1	Running	0	54m
groupware-demo-10-xrsxh	1/1	Running	0	52m
groupware-demo-10-zh4w9	0/1	Running	0	10s

Group-ware 애플리케이션에 Auto-Scale(5-10) 설정 후 동일한 부하 발생.

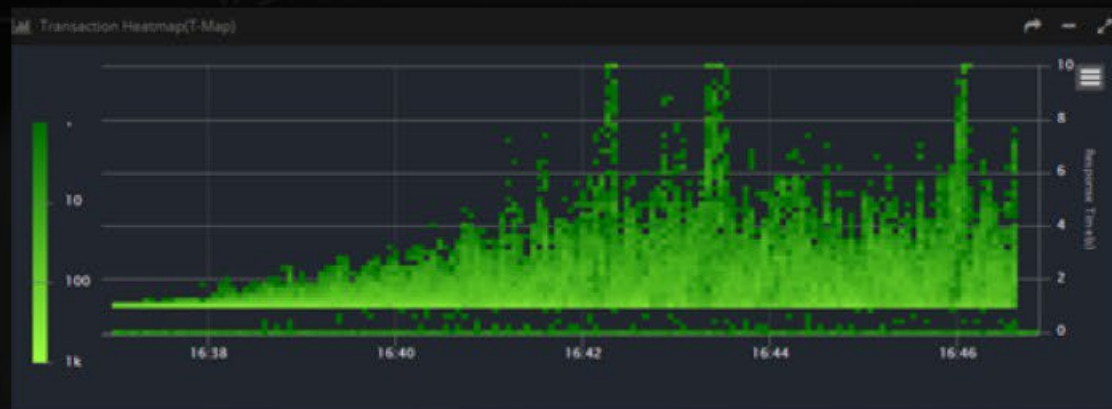
DEMO - 부하 테스트 결과 비교



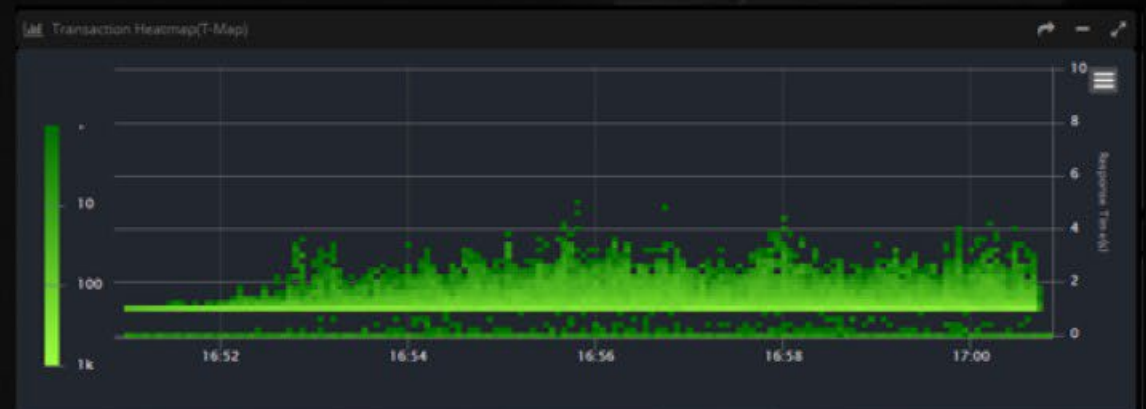
- Peak 시점이 다른 통계조사에 대해 별도 개입없이 자동자원할당이 가능함을 확인
- 기존환경과 비교하여 자동자원할당이 되는 환경이 1.7 배 많은 양을 처리하며, 평균응답시간이 2.5 배 빠름

테스트 케이스	테스트 내용	시뮬레이션 테스트 환경	처리량	처리량 비교	평균응답시간	응답시간비교	최소응답시간	최대응답시간	TPS
T-01	자원자동할당이 안 되는 환경에서 부하테스트	기존 병원 업무 시스템	53,582	100%	2,374	100%	1,010	19,979	88
T-02	Peak 시점이 다른 자동 자원할당이 되는 환경에서 부하테스트	병원 업무 Peak 일 때 환경	93,869	169%	919	258%	10	4,589	155
T-03	Peak 시점이 다른 자동 자원할당이 되는 환경에서 부하테스트	건강 검진 Peak 일 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





openmaru

Application Performance Management

감사합니다.



openmaru
APM