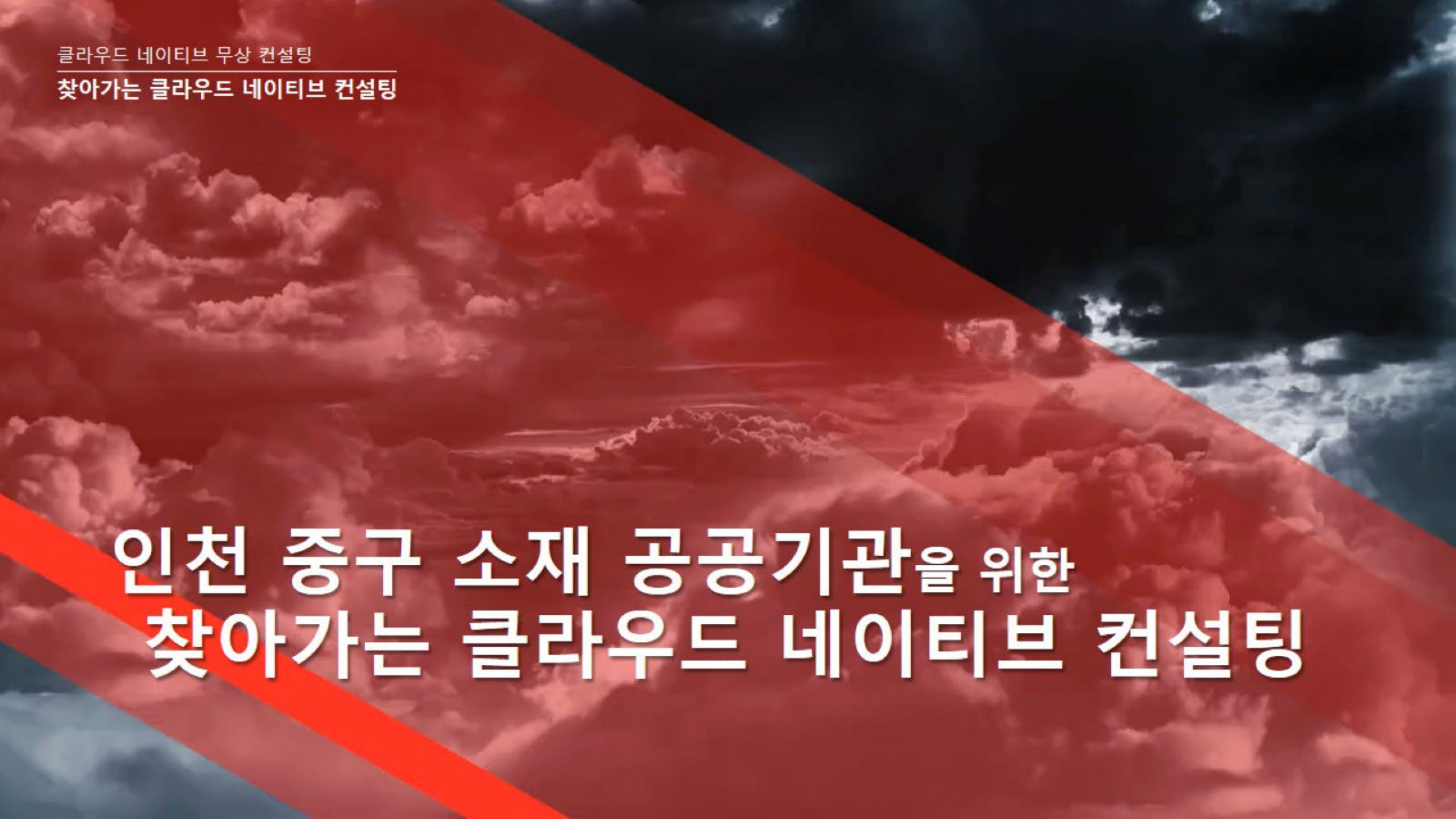


인천 중구 소재 공공기관을 위한 찾아가는 클라우드 네이티브 컨설팅

Document Creation Date
2024.03.10

Document Version
1.0

openmaru

클라우드 네이티브 도입 시 인프라 구성과 운영의 고려사항

OPENMARU Inc.
오픈마루 주식회사

Platform As A Service

개발/운영팀은 왜 클라우드 네이티브를 도입하는 걸까요?

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.

Source:https://github.com/cncf/toc/blob/master/DEFINITION.md#%ED%95%9C%EA%B5%AD%EC%96%B4

클라우드 네이티브 란?

openmaru

- 클라우드의 이점을 최대로 활용할 수 있도록 애플리케이션을 구축하고 실행하는 방식

DevOps (SRE)

애플리케이션 **개발-운영 간의 협업 프로세스**를 자동화하는 것을 말하며 결과적으로 **애플리케이션의 개발과 개선 속도향상**

- 속도와 안정성
- 협업 강화
- 문화
- 플랫폼 필요성 대두
- OnDemand Service
- SRE

Continuous Delivery

- **지속적인 통합**(CI)은 개발자가 작업한 **코드를 자동으로 테스트하고 통합**
- **지속적인 배포**(CD)는 코드를 리포지토리에 업로드하고, **서비스 배포로 릴리즈까지 자동화**

- Agile
- 짧고 지속적 반복
- 지속적 통합/배포/제공
- 플랫폼/애플리케이션 배포 자동화

Microservices

애플리케이션을 구성하는 서비스들을 독립적인 **작은 단위로 분해하여** 구축하고 각 구성 요소들을 **네트워크로 통신하는 아키텍처로** 서비스 안정성과 확장성(scaling)을 지원

- API First
- 도메인 주도 설계
- 독립적 확장 가능
- 보다 빠르고 독립적인 배포
- 간편한 개발 및 유지관리

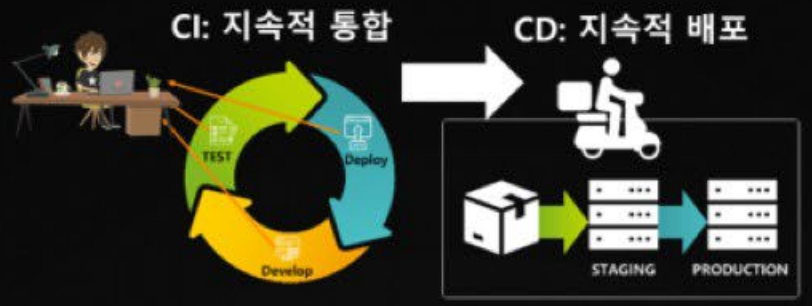
Containers

가상화 기술 중 하나로, **시스템을 가상화 하는 것이 아니라** 애플리케이션을 구동할 수 있는 컴퓨팅 작업을 패키징하여 **OS를 가상화** 한 것입니다.

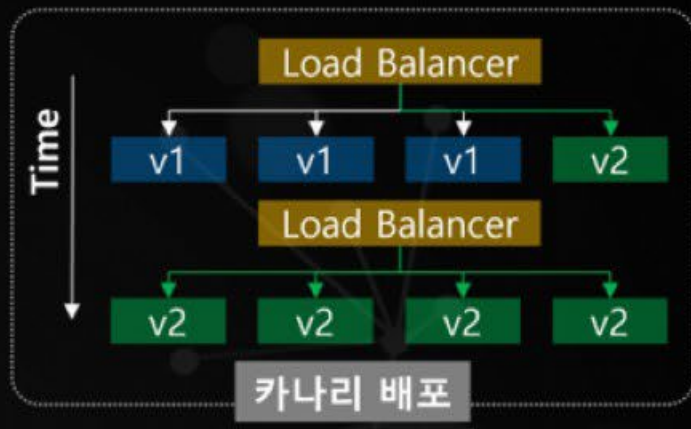
- 컨테이너 오케스트레이션
- 불변의 인프라스트럭처
- 변경이 아닌 폐기 후 생성
- 일관된 환경 유지

클라우드 네이티브 기반 환경의 장점

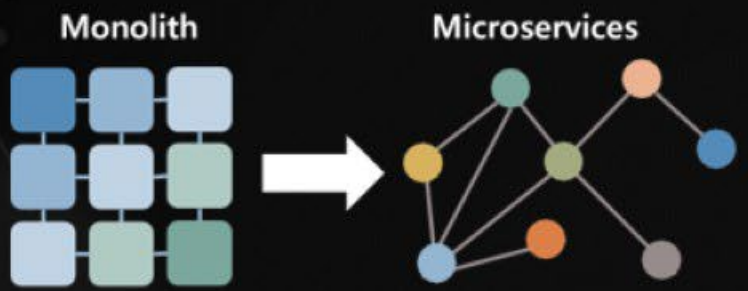
컨테이너 기반의 빠른 개발환경



신속한 개발과 편리한 배포



서비스 무중단



MSA개발에 적합한 환경

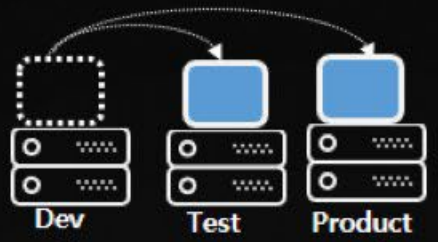
DevOps기반의 민첩한 개발

Cloud Native Computing으로 전환 효과

- **Cloud Native Computing** 환경은 클라우드가 제공하는 민첩성, 가용성, 확장성의 장점을 어플리케이션/서비스의 개발, 운영, 관리에 적용하여 기존 컴퓨팅 환경을 최적화 함

On Demand Delivery

필요한 컴퓨팅 자원을 즉시 제공



Consistency & Continuous

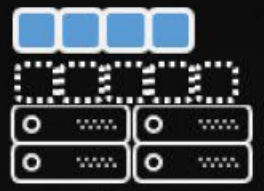
이미지 기반으로 구성, 배포 효율화
개발과 운영 환경의 일관성

Rolling Update

업그레이드 또는 패치 시
다운 타임은 제로 또는 최소화

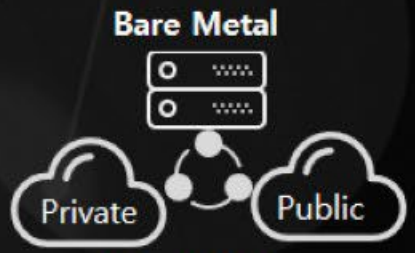
Self Recovery

- 비정상 애플리케이션 재시작
- 노드의 장애 발생시
  정상 서버 노드로 자동 재배치



Application Scaling

VM 단위가 아닌 어플리케이
단위의 오토스케일링



Portable

멀티/하이브리드 클라우드 기반
어플리케이션/서비스 운영

개발팀과 운영팀 누구의 잘못인가? 불행의 시작

- 하드웨어 도입부터, OS, 미들웨어, 빌드/배포, 기타 인프라 환경 등 복잡한 과정
- 관리 서버 증가

기존개발 운영 환경

계획 → 예산수립 → H/W 선정 → 입고 및 대기 → H/W 반입 → H/W 설치 → OS 도입 → OS 패치 / 튜닝 → 사용자 생성 → M/W 서버 도입 → 테스트 도구 선정 → 운영 배포 → 서버 구성 및 설정 → 운영서버 이전 → 운영서버 가동 → 서버 추가계획

코드 개발 → 빌드 → 정적 분석 → 시큐어 코딩 → 단위 테스트

테스트 배포 → 통합 테스트

- 개발과 운영팀은 분리
- 여러 단계의 개발과 운영 프로세스를 수작업으로 수행함

컨테이너 자동화 도구를 이용한 프로세스

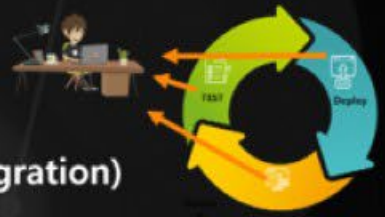
- 파이프라인 기반의 **자동화 환경**을 이용하여 **신속한** 개발 및 운영 **환경 구축**
- 필요에 따라 **구축과 삭제가 편리**

DevOps 환경

개발 환경 구축
(Service Catalog)

자동 확장/ 자동 복구
(Auto Scaling/ Healing)

CI: 지속적인 통합
(Continuous Integration)



CD: 지속적인 배포
(Continuous Delivery)

시스템 비대화로 작업 폭증과 인력 부족 어떻게 할까요?

장애의 65 %는 Human Error이며,시스템 복잡도와 난이도 증가

시스템 운용 업무의 45 %는 정기적으로 수행해야하는 반복 작업

운영 효율화를 통한 비용 절감의 요구

시스템의 대규모 화

높은 수준의 엔지니어 부족

지속적인 시스템 통합 요구

동일한 작업 반복

운영 품질 향상

운영 비용 (TCO) 절감 요구

업무 확대와 관련 데이터양의 비약적인 증가

가상화, 클라우드 등 다양한 운영 환경의 증가와 관리 효율화 요구

운영 품질에 대한 지속적인 향상 요청

10억 명 이상의 클라우드 서비스

openmaru

Google Cloud

- 2조 / 매년 - 구글 검색
- 650억/ 매년 - Google PlayStore 앱 다운로드
- 10억 명/ 매월 - 모바일 Chrome 브라우저.
- 2억 명/ 매월 - Google 포토

GOOGLE 과 컨테이너

openmaru

- **Google의 업무 방식**

Gmail에서 YouTube, 검색에 이르기까지 Google의 모든 제품은 컨테이너에서 실행됩니다.

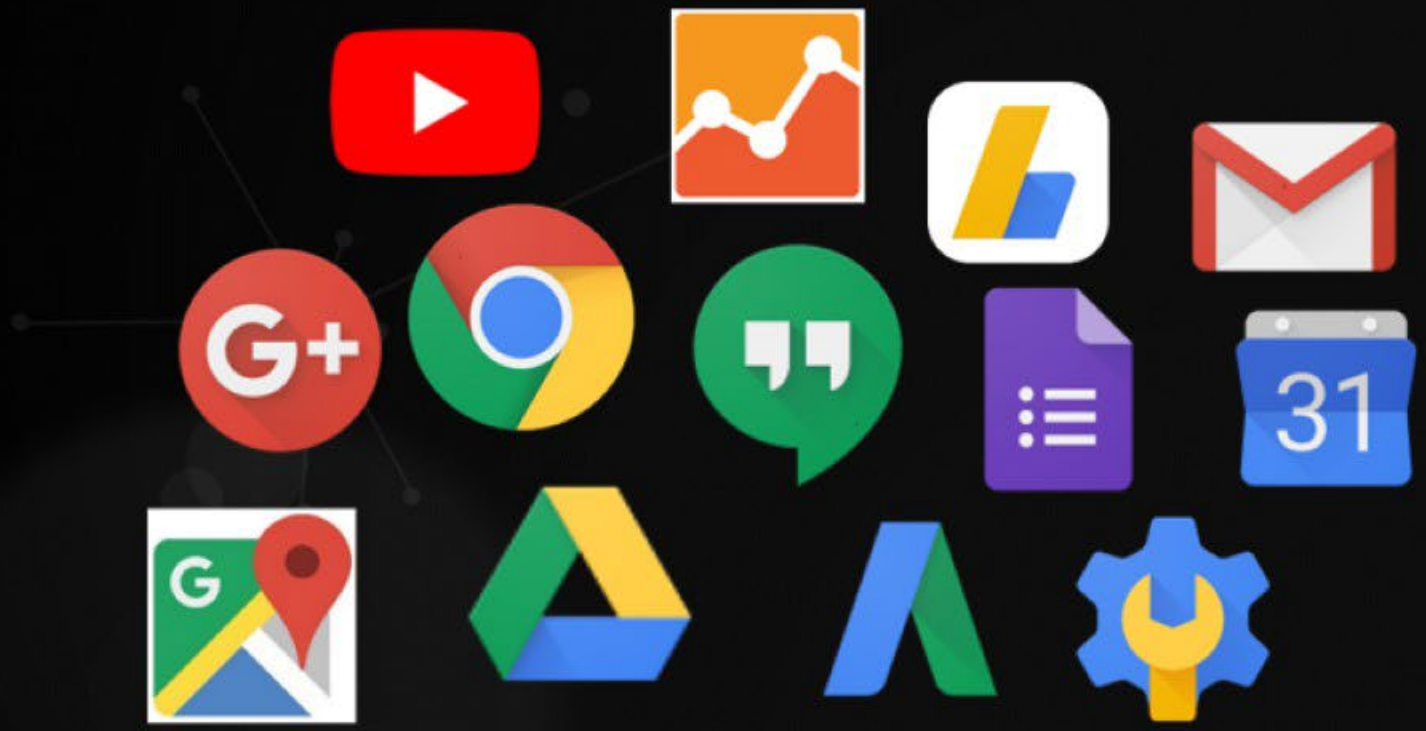
개발팀은 컨테이너화를 통해 더욱 신속하게 움직이고, 효율적으로 소프트웨어를 배포하며 전례 없는 수준의 확장성을 확보할 수 있게 되었습니다. **Google은 매주 수십억 개가 넘는 컨테이너를 생성**합니다. **지난 10여 년간 프로덕션 환경에서 컨테이너화된 워크로드를 실행하는 방법에 관해 많은 경험**을 쌓으면서 Google은 커뮤니티에 계속 이 지식을 공유해 왔습니다.

초창기에 cgroup 기능을 Linux 커널에 제공한 것부터 **내부 도구의 설계 소스를 Kubernetes 프로젝트로 공개**한 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.

Source - https://cloud.google.com/containers/

Google 는 모두 컨테이너에서 실행

- Gmail , 검색, 지도 ...
- MapReduce , GFS , Colossus ...
- Google Compute Engine 가상 머신도 컨테이너에서 실행!
- 매주 20 억개 이상의 컨테이너를 실행 중

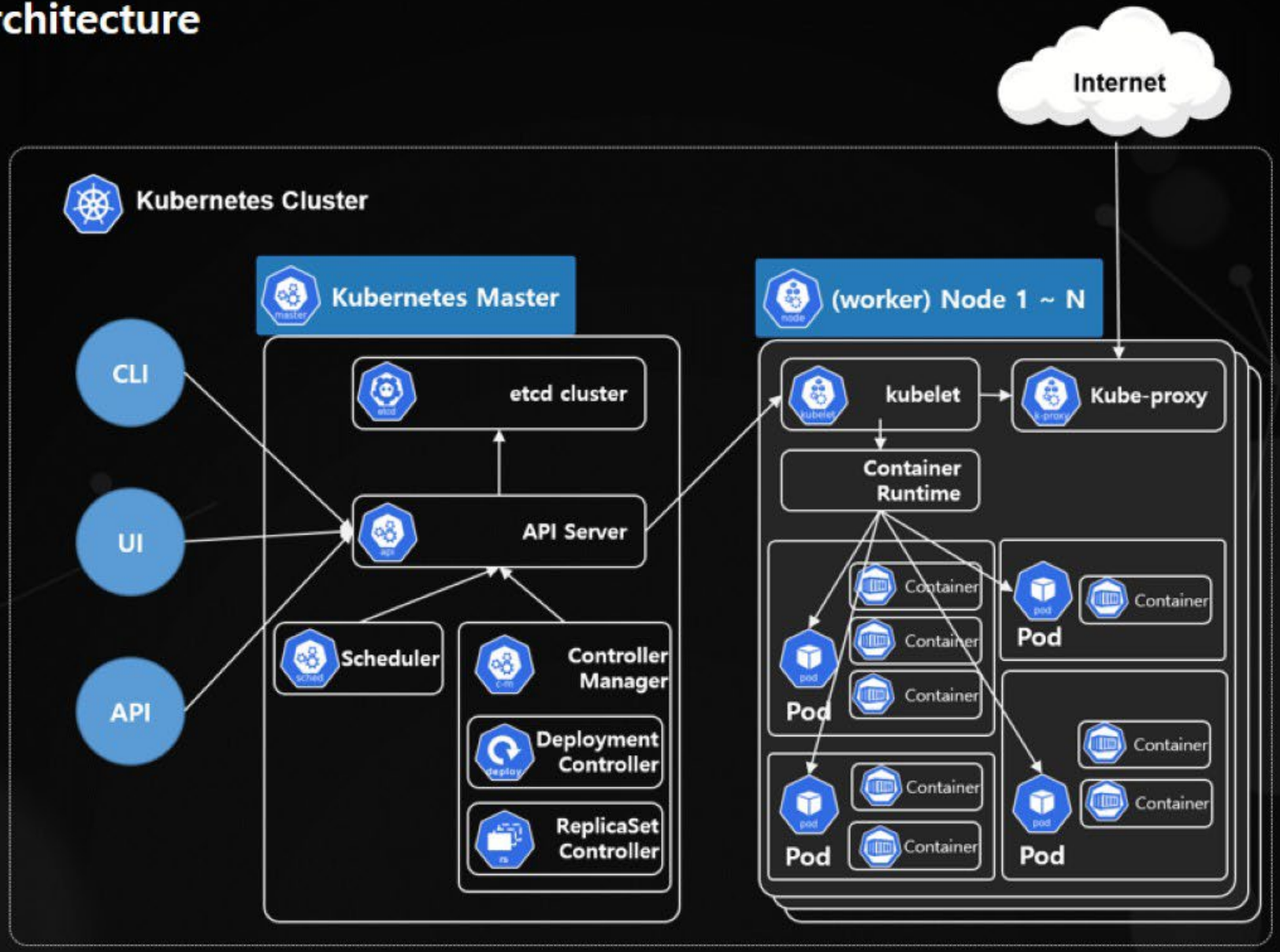


About Kubernetes

- 쿠버네티스(K8s)는 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리해주는 오픈소스 소프트웨어
- 쿠버네티스", "쿠베르네테스","K8s", "쿠베", "쿠버", "큐브"라고 부르며 Apache License 2.0 라이선스로 리눅스 재단 (Linux Foundation)산하 Cloud Native Computing Foundation (CNCF) 에서 관리
- Go로 작성된 오픈 소스 , OSS (Apache License 2.0) 라이선스
- 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*

Kubernetes Architecture

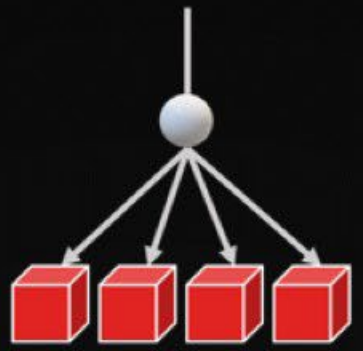


Kubernetes 주요 기능

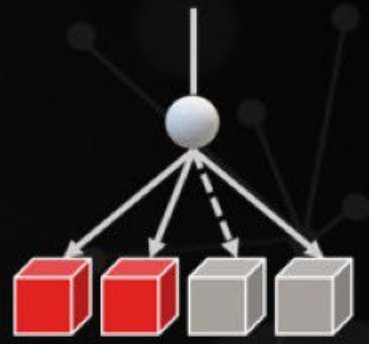
openmaru

Scale Out / In

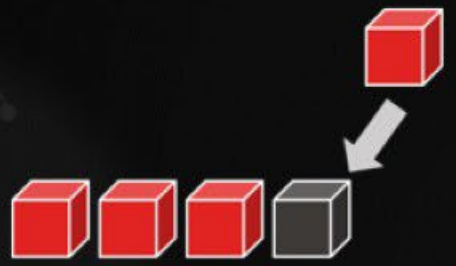
Load Balancer



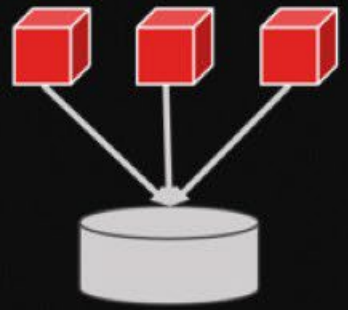
Rolling Update



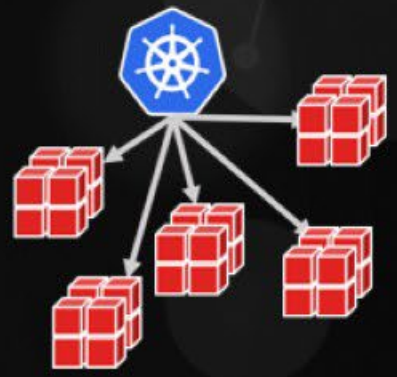
Auto Healing



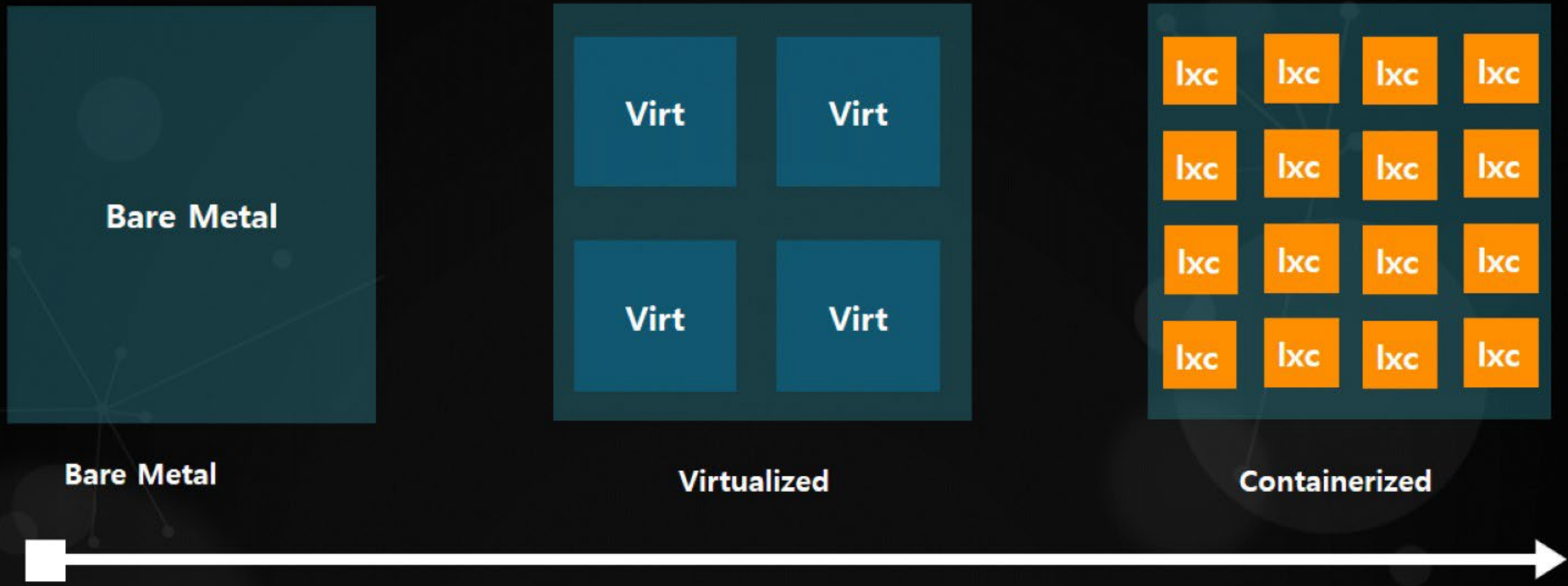
Persistence Volume



Container Orchestration

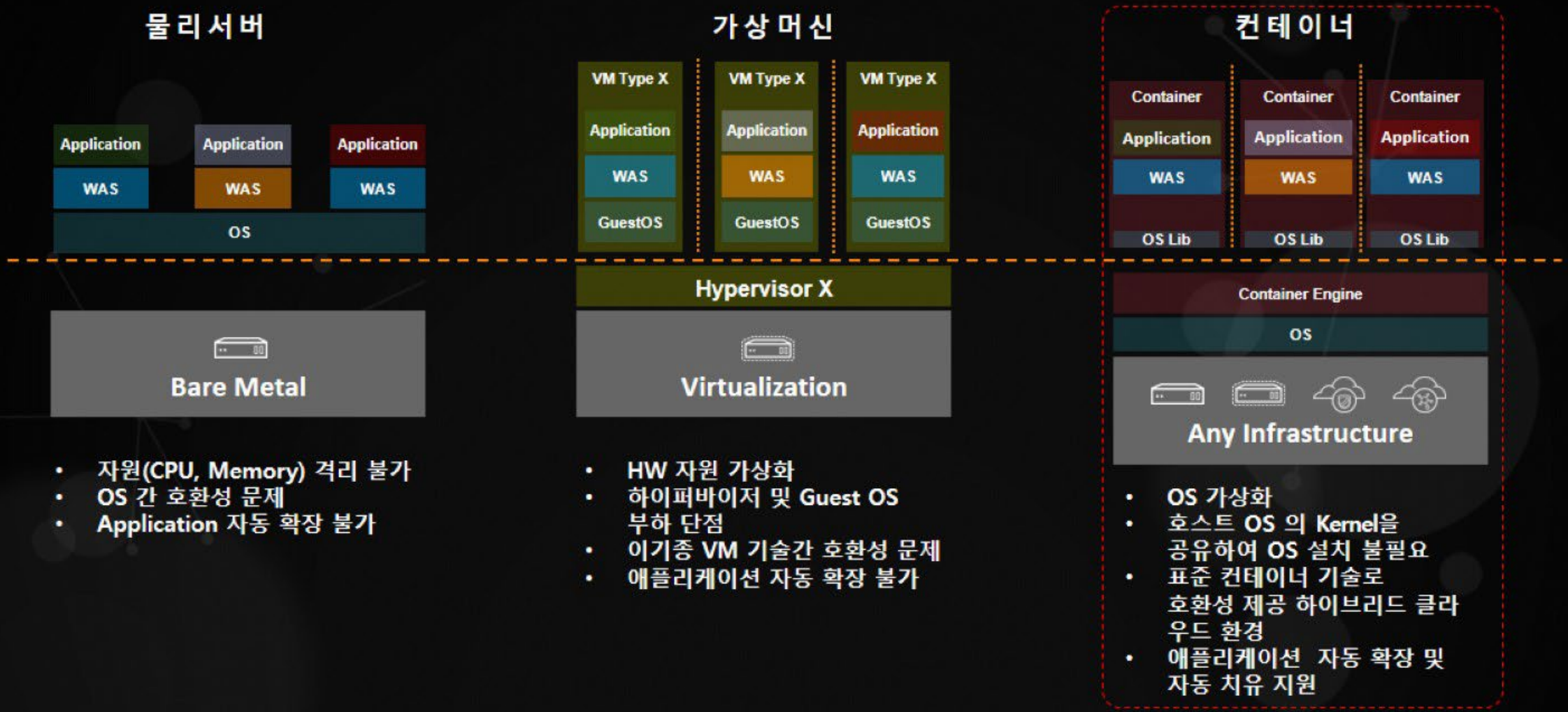


Evolution of Infrastructure Architectures



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

오버헤드 - Containers vs. VMs

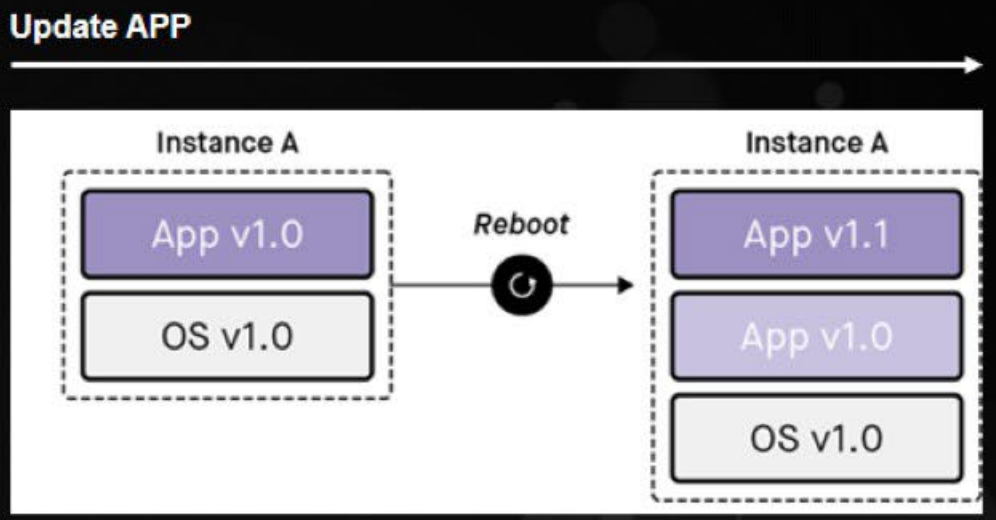
- OS에서 응용 프로그램을 작동하는 경우, 하드웨어 가상화에서는 가상화 된 하드웨어 및 하이퍼바이저를 통해 처리하기 때문에 물리적 시스템보다 처리에 부가적인 시간 (오버 헤드)가 필요
- 컨테이너 형 가상화 커널을 공유하고 개별 프로세스가 작업을 하는 것과 같은 정도의 시간 밖에 걸리지 않기 때문에 대부분 오버 헤드가 없음

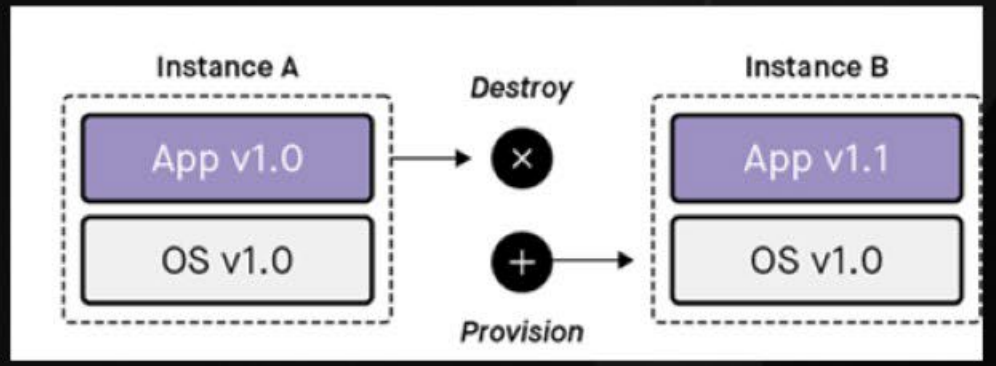
Mutable vs. Immutable Infrastructure (가변 vs. 불변)

openmaru

Mutable Infrastructure (물리서버, 가상화)

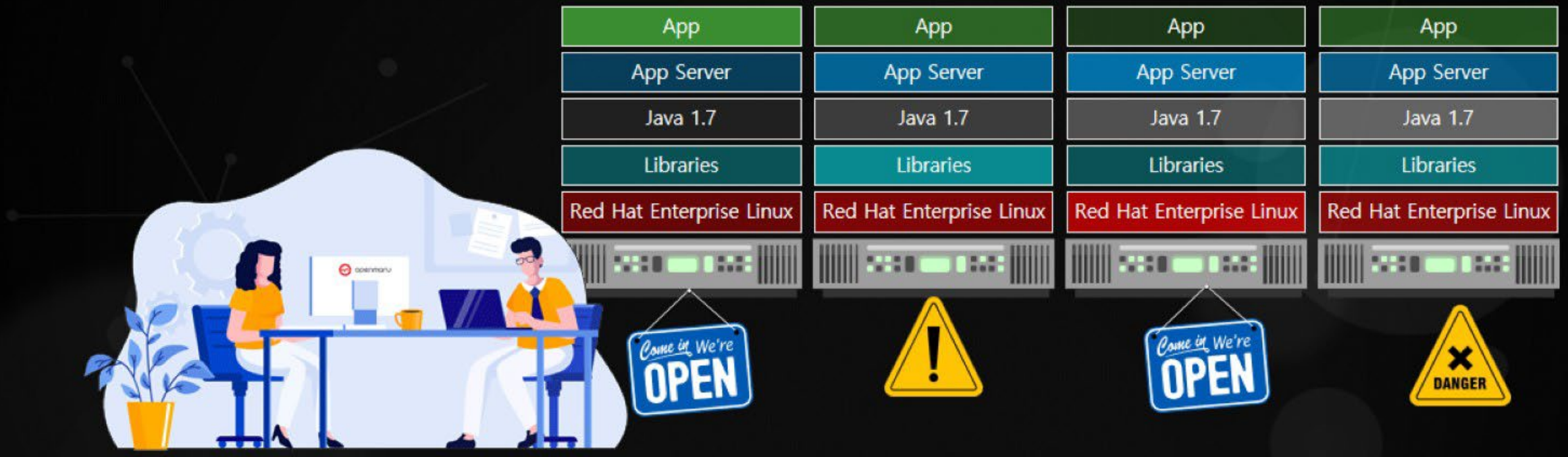
Immutable Infrastructure (컨테이너)





Configuration Drift

- 컨피그레이션 드리프트 (**Configuration Drift**)
  - 손으로 직접 수정한 임시 수정/업데이트와 전반적인 엔트로피(**entropy**) 증가로 인해 인프라의 서버들이 시간이 갈수록 점점 서로 다른 상태가 되는 현상
  - 장비의 라이프사이클 동안 초기 설정으로부터 멀어지고(**drift**) 다른 장비들 과도 서로 달라짐



Pets vs Cattle

openmaru

Pets

Cattle

컨테이너는 클라우드에서 Java 와 같이 벤더 종속성 해제

openmaru

2000 년 – Java 를 통한 Vendor Lock-In 해제

OS	HP-UX	AIX	Solaris	Linux	Windows
CPU	Intanium	Power	Sparc	X86	X86
Vendor	HP	IBM	Oracle	The Linux Foundation	Microsoft

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

Hypervisor	Xen Project	KVM	Hyper-V	Xen Project	Xen Project
Public Cloud	aws	Google Cloud	Microsoft Azure	Oracle	IBM Cloud

Platform As A Service

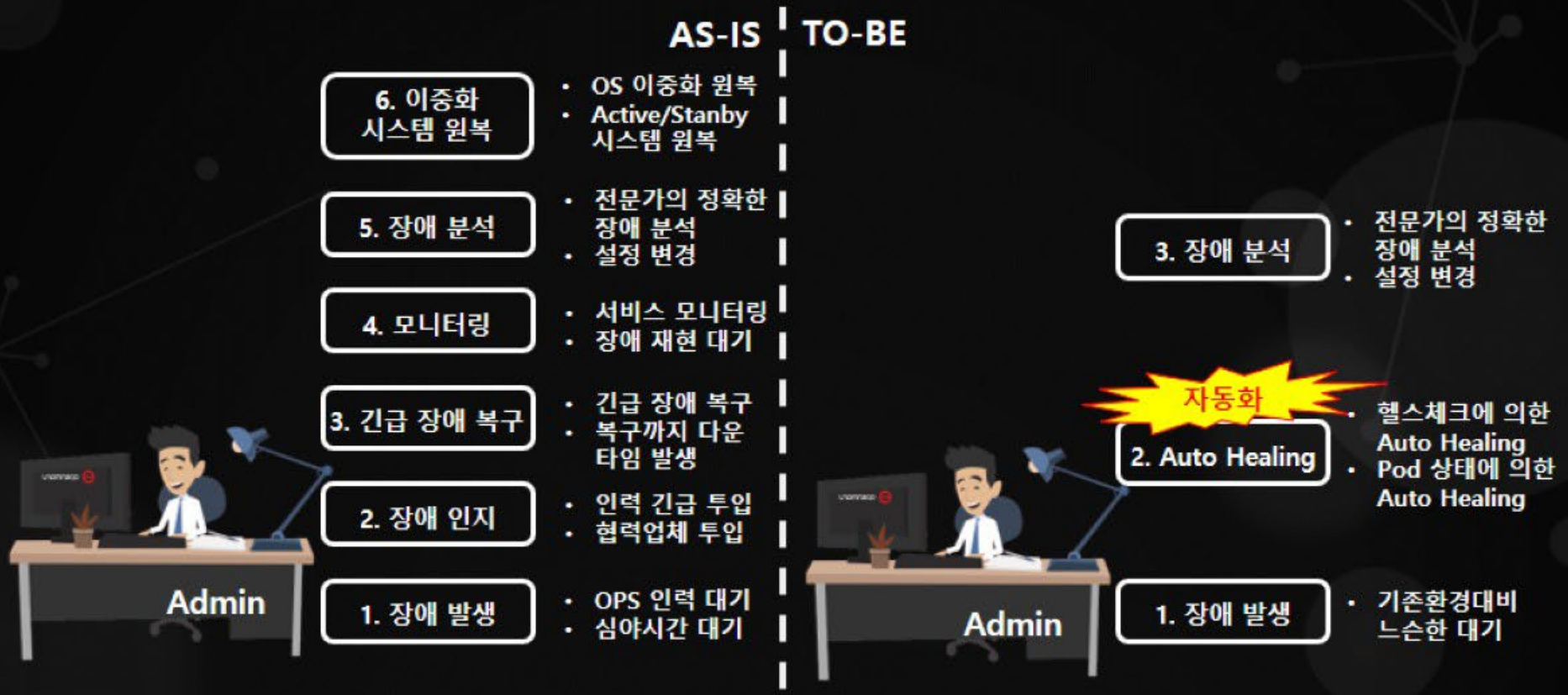
클라우드 네이티브의 특징으로 인하여 작업이 어떻게 바뀔까요?

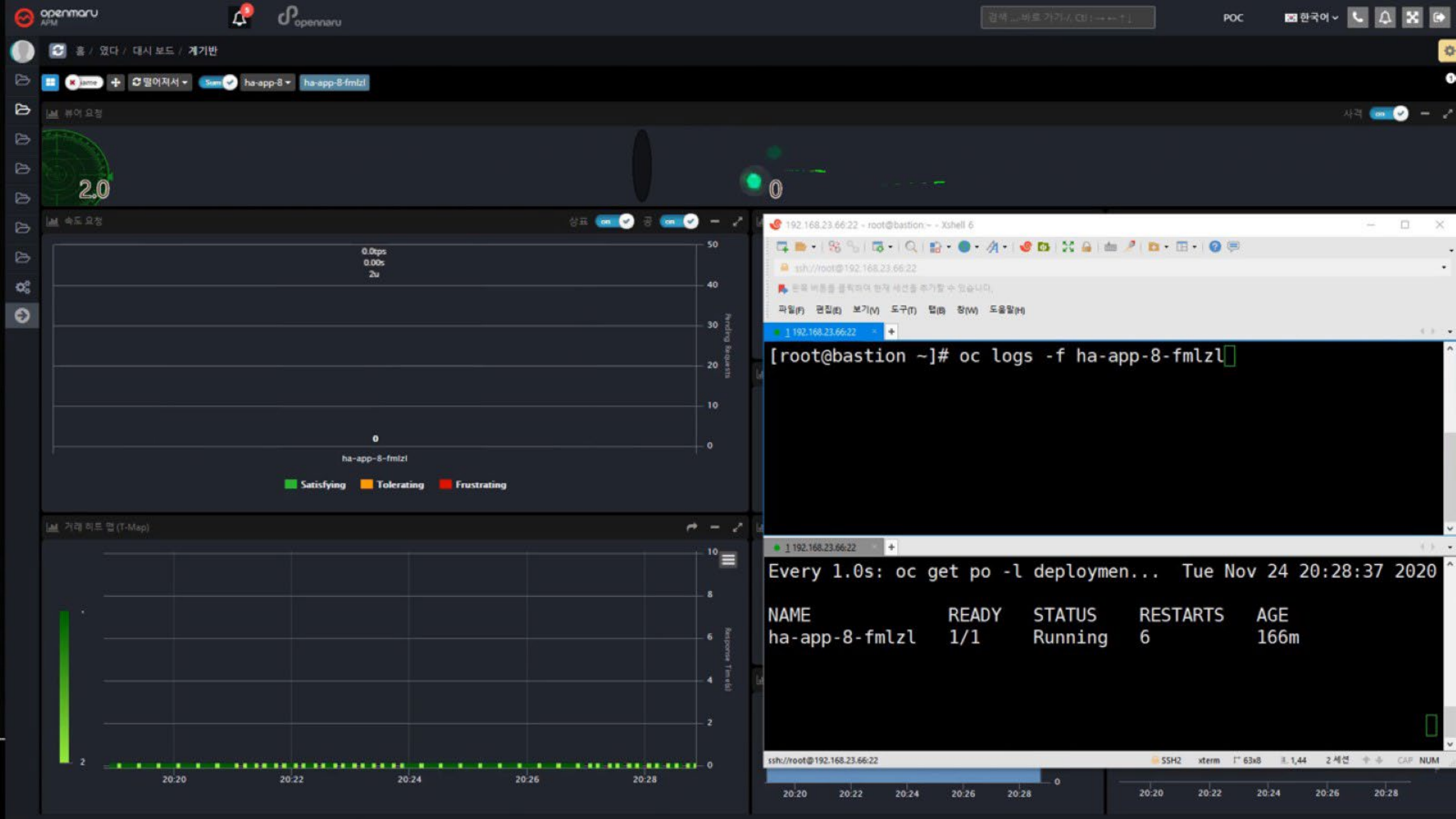
장애 상황 운영관리 비교

openmaru

인프라 팀(장애상황)

- 기존 환경은 장애 발생을 대비한 인력들이 상시 대기해야 함
- 야간 장애 발생시 **복구까지 긴 시간 소요, 서비스 다운 타임** 발생
- 컨테이너 환경은 Auto Healing으로 인해 장애 발생시 바로 자동 복구



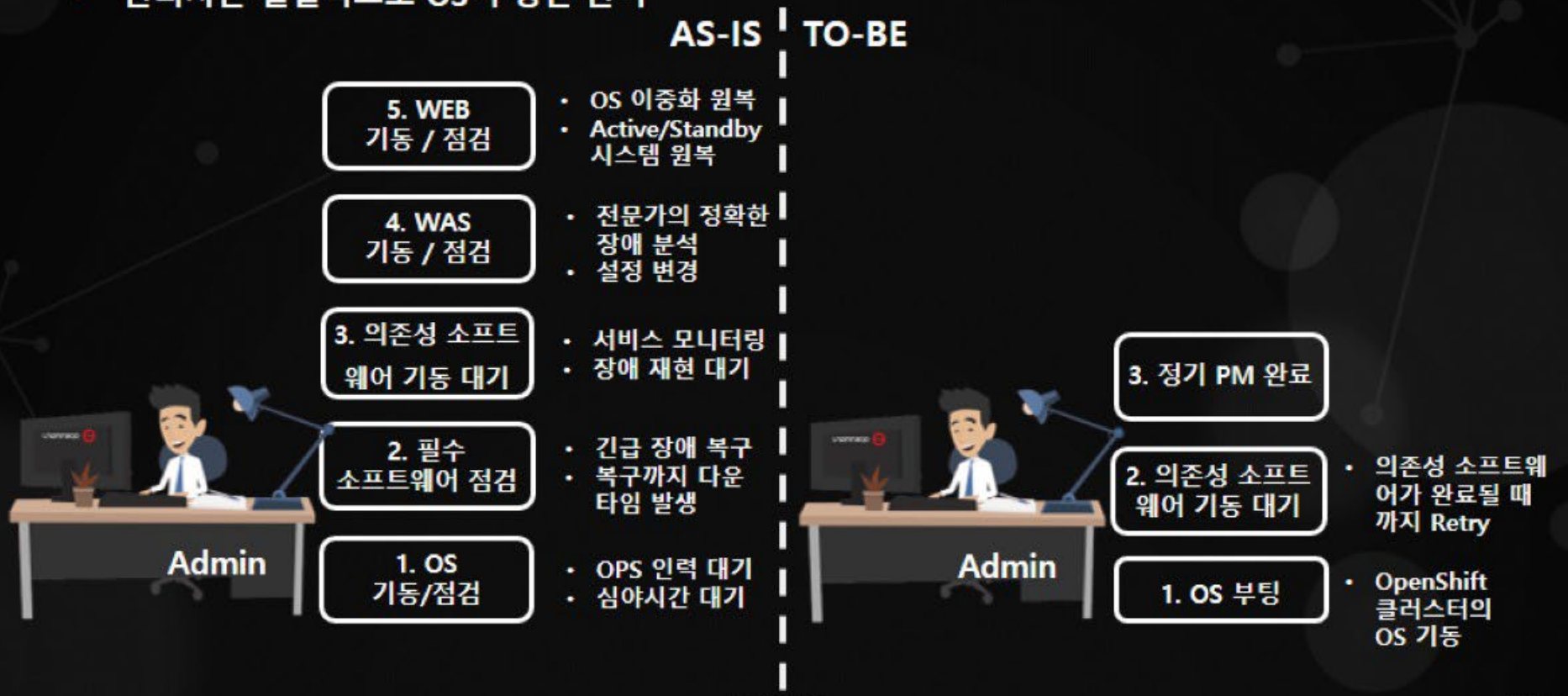

openmaru
APM
POC
한국어
홈 / 였다 / 대시 보드 / 계기반
ha-app-8
ha-app-8-fmlzl
2.0
0
0.0tps
0.00s
2u
ha-app-8-fmlzl
Satisfying
Tolerating
Frustrating
[root@bastion ~]# oc logs -f ha-app-8-fmlzl
Every 1.0s: oc get po -l deploymen... Tue Nov 24 20:28:37 2020
NAME READY STATUS RESTARTS AGE
ha-app-8-fmlzl 1/1 Running 6 166m
20:20
20:22
20:24
20:26
20:28

정기 PM 운영관리 비교

openmaru

인프라 팀(정기 PM)

- 기존 환경은 소프트웨어에 따라 기동 순서가 있음
- 의존성 소프트웨어(ex: DataBase)가 정상 기동 될 때 까지 관리자는 작업 대기
- 컨테이너 환경은 의존성 소프트웨어로 인해 기동 실패하더라도 성공 시 까지 Retry
- 관리자는 실질적으로 OS 부팅만 관여



부하 발생 및 부하 예정시 운영 관리 비교

openmaru

인프라 팀(부하 상황)

- 부하가 예고된 경우 수작업으로 Scale-Up 진행 → 애플리케이션 인스턴스도 늘려줘야 함.
- 예고되지 않은 부하의 경우 서비스의 사용자 만족도 하락
- 컨테이너 기반의 클라우드 네이티브 환경은 부하에 따라 자동으로 컨테이너(애플리케이션 인스턴스) 증설

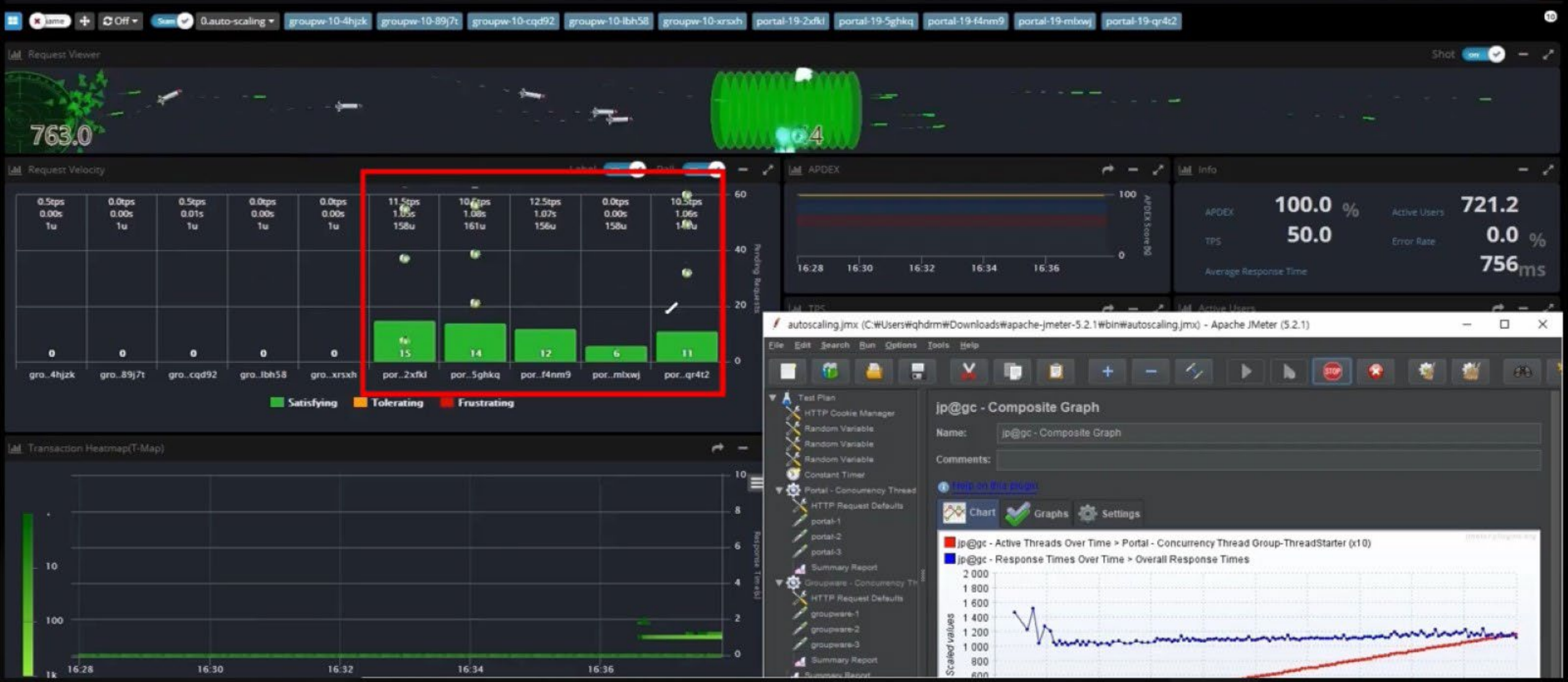
AS-IS

- 5. 서비스 기동
  - 부하 분산 확인
- 4. 애플리케이션 인스턴스 증설
  - 수작업 증설
  - L4, Web 라우팅 규칙 추가
- 3. VM서버 리소스 증설
  - 수작업으로 리소스 증설
- 2. VM서버 종료
  - 증설까지 다운타임 발생
- 1. 부하 상황 인지
  - OPS 인력 대기
  - 심야시간 대기

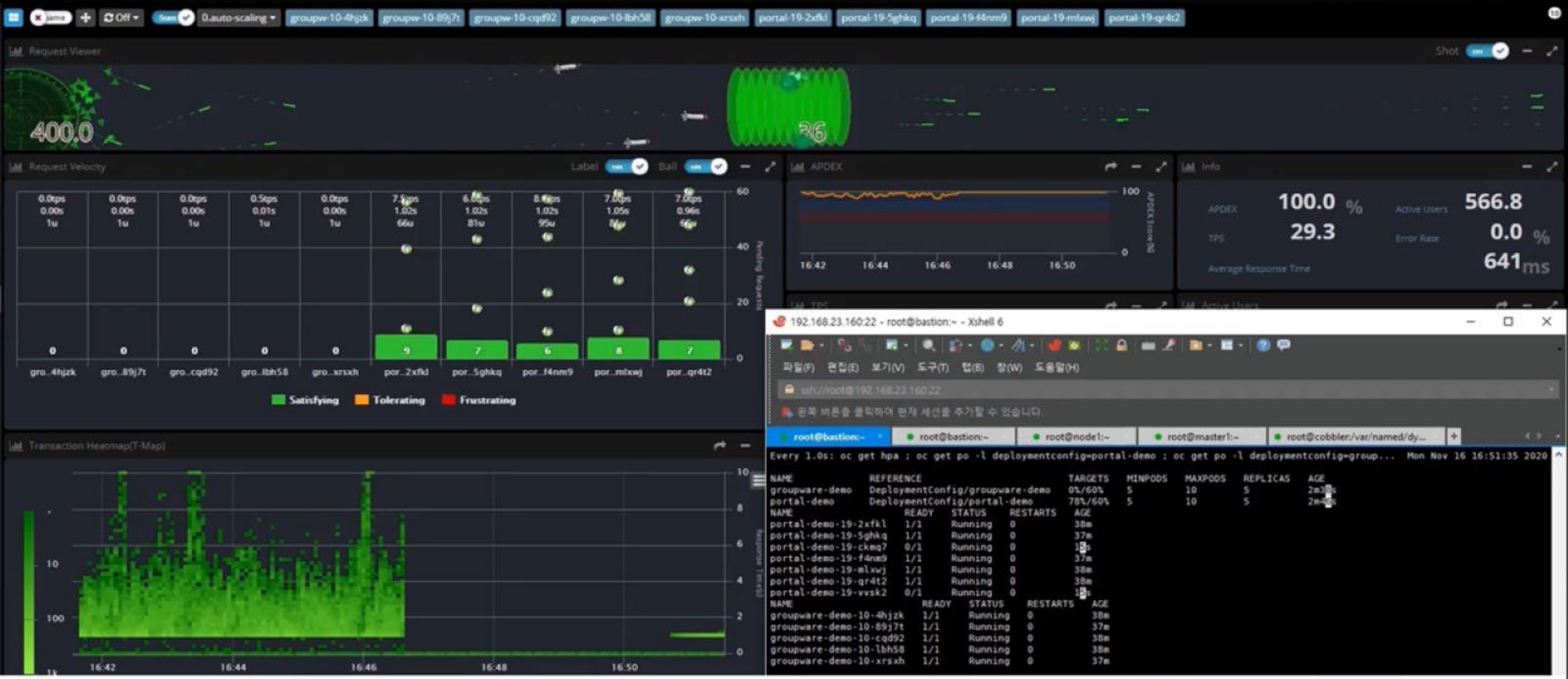
TO-BE

- 2. 서비스 증설 완료
- 자동화
- 1. 마우스 클릭으로 서비스 증설
  - 컨테이너 기반으로 인한 빠른 기동 시간
- OR
- 1. 부하에 따른 오토스케일링
  - 부하 생성시 애플리케이션 자동 증설
  - 부하가 줄었을 때에는 자원 회수

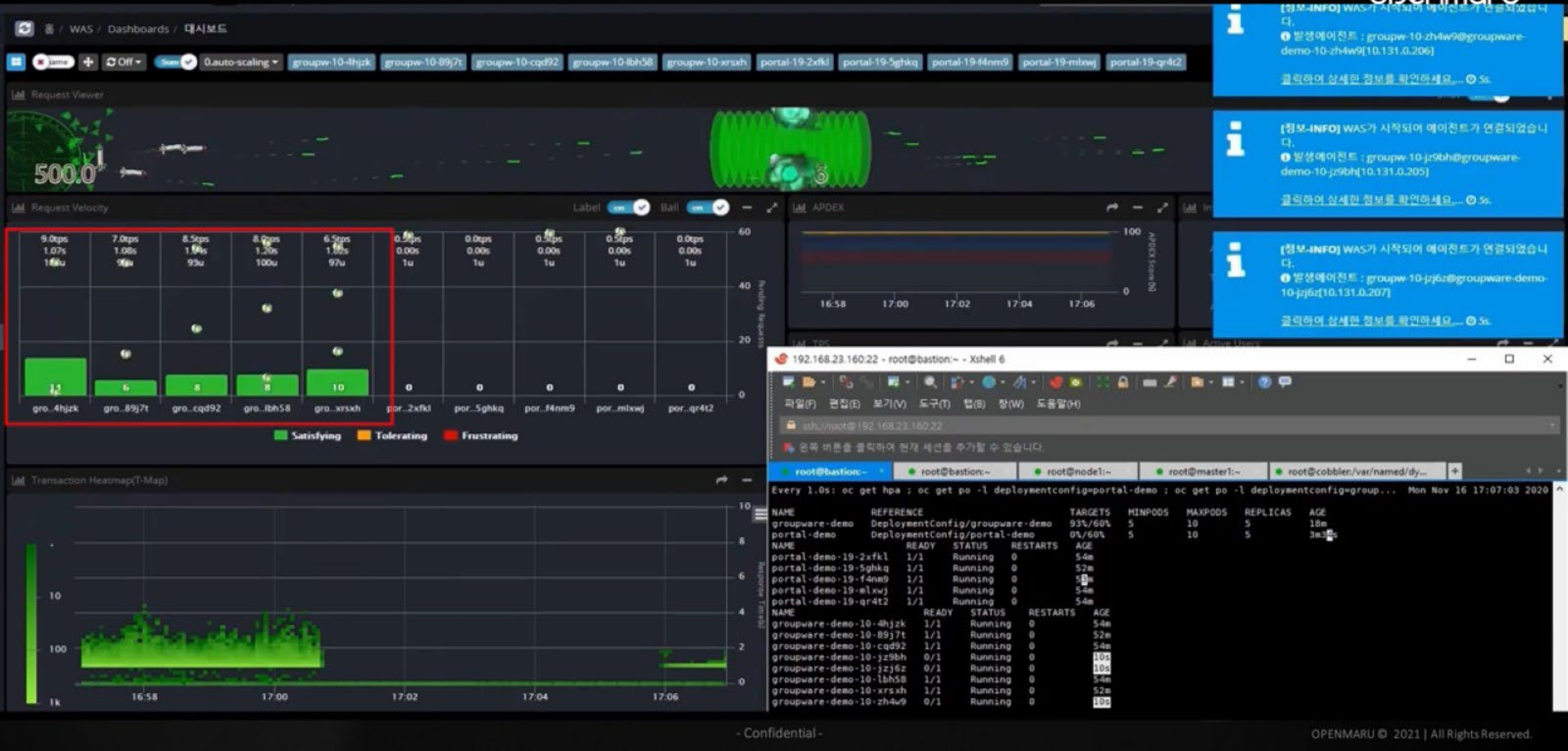
오토스케일링이 적용되지 않은 환경(통계청 나라통계 업무)



오토스케일링 적용 환경(통계청 전국사업체조사 업무)



오토스케일링 적용 환경(통계청 초중고 사교육비 조사)

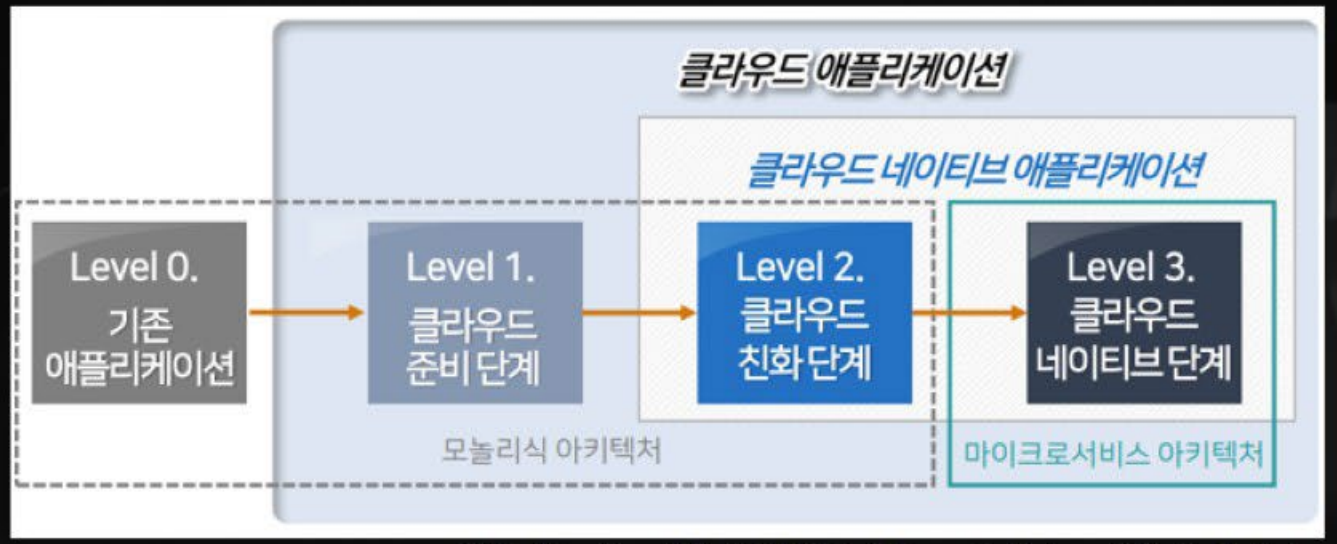



Platform As A Service
클라우드 네이티브 환경에서 해야할 것.
openmaru
APM

클라우드 네이티브를 가로막는 두려움

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

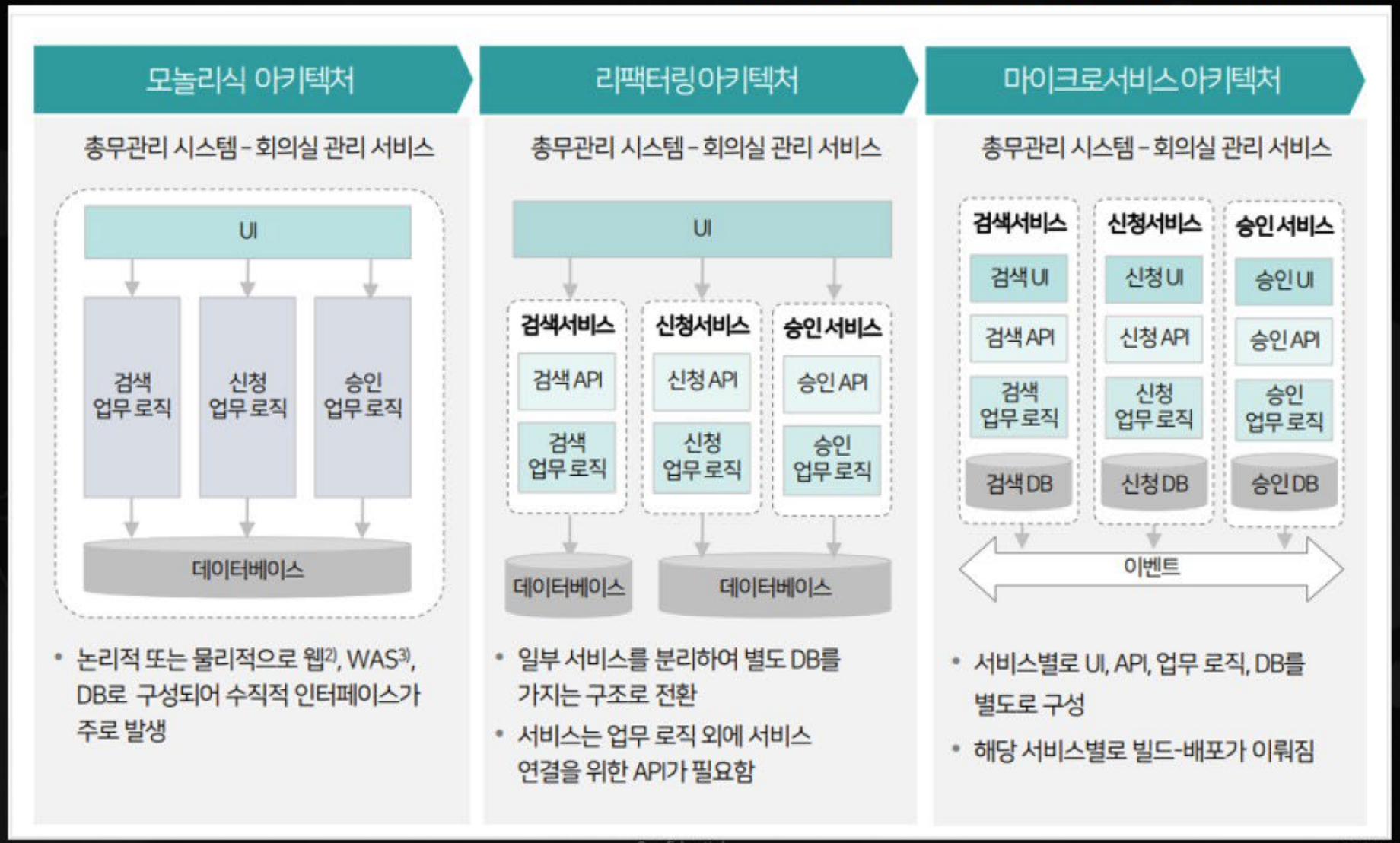


한국지능정보사회진흥원 클라우드 네이티브 발주자 안내서

마이크로서비스 아키텍처로의 전환 예시

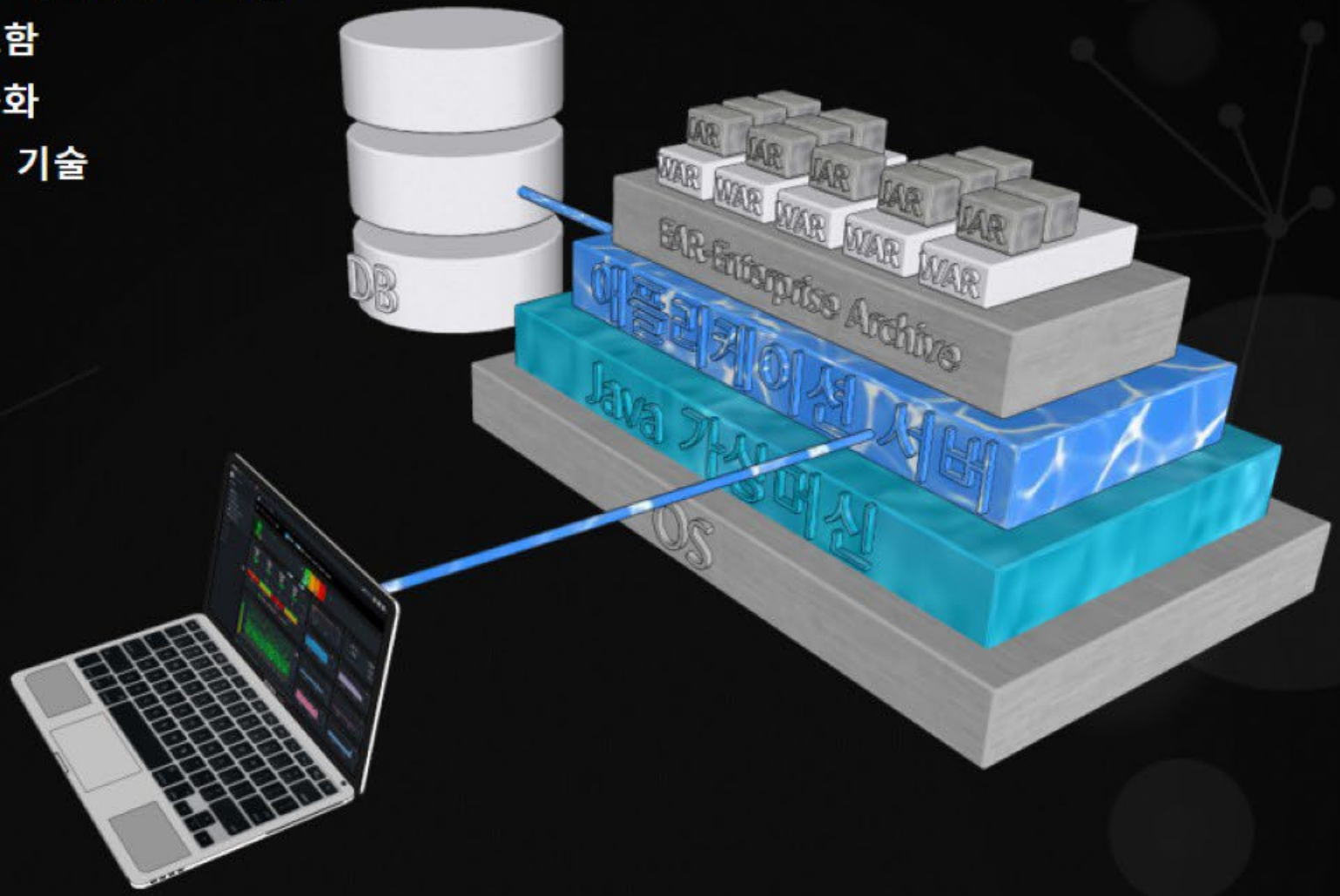
openmaru

한국지능정보사회진흥원 클라우드 네이티브 발주자 안내서



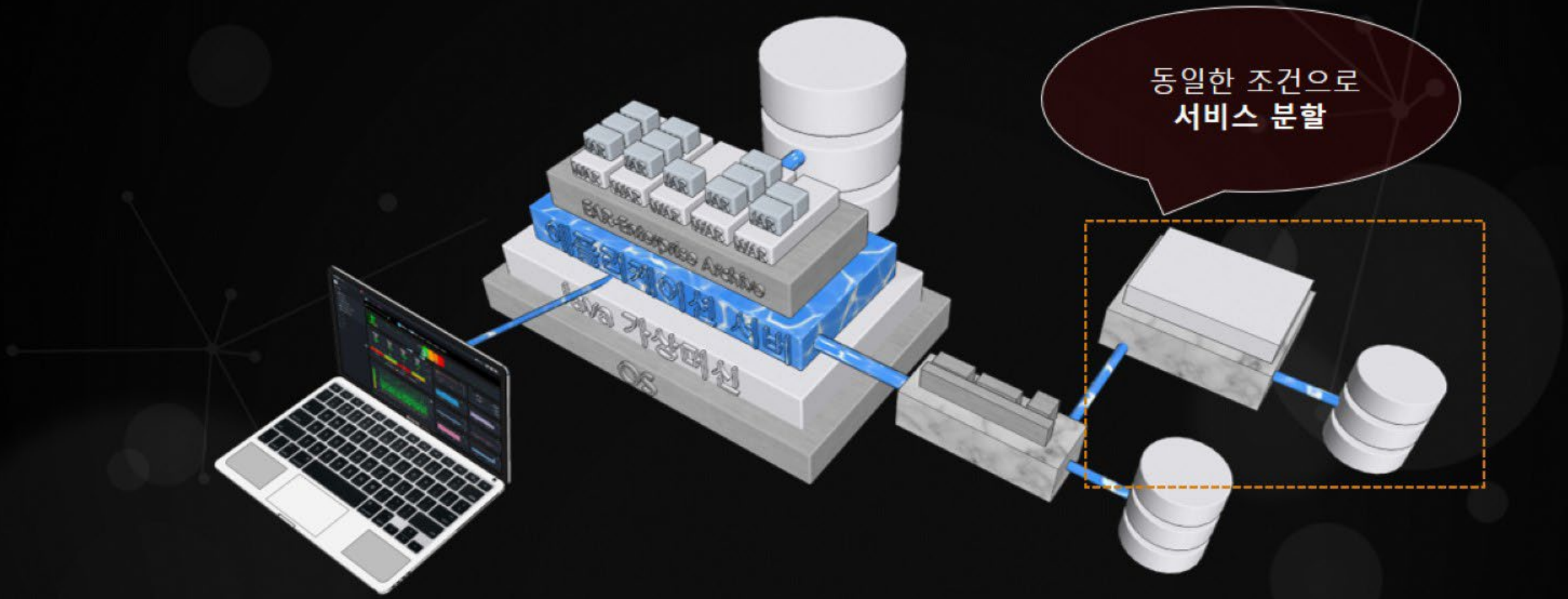
Monolith Architecture

- 모노리스 아키텍처의 특징
  - 견고함
  - 표준화
  - 단일 기술



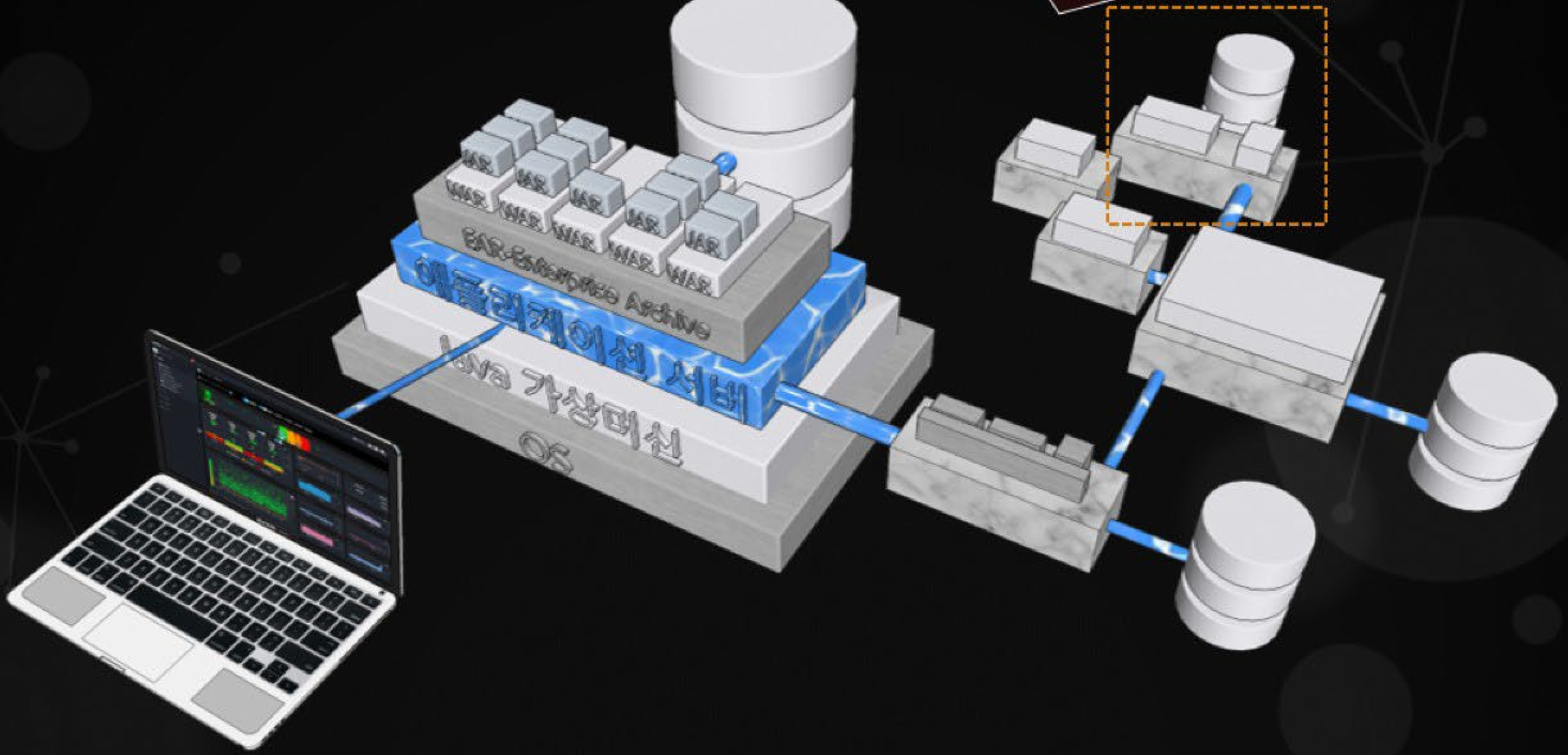
Monolith → Microservices – Phase 1

모노리스에서 마이크로서비스로 전환 – Phase 2



모노리스에서 마이크로서비스로 전환 – Phase 3

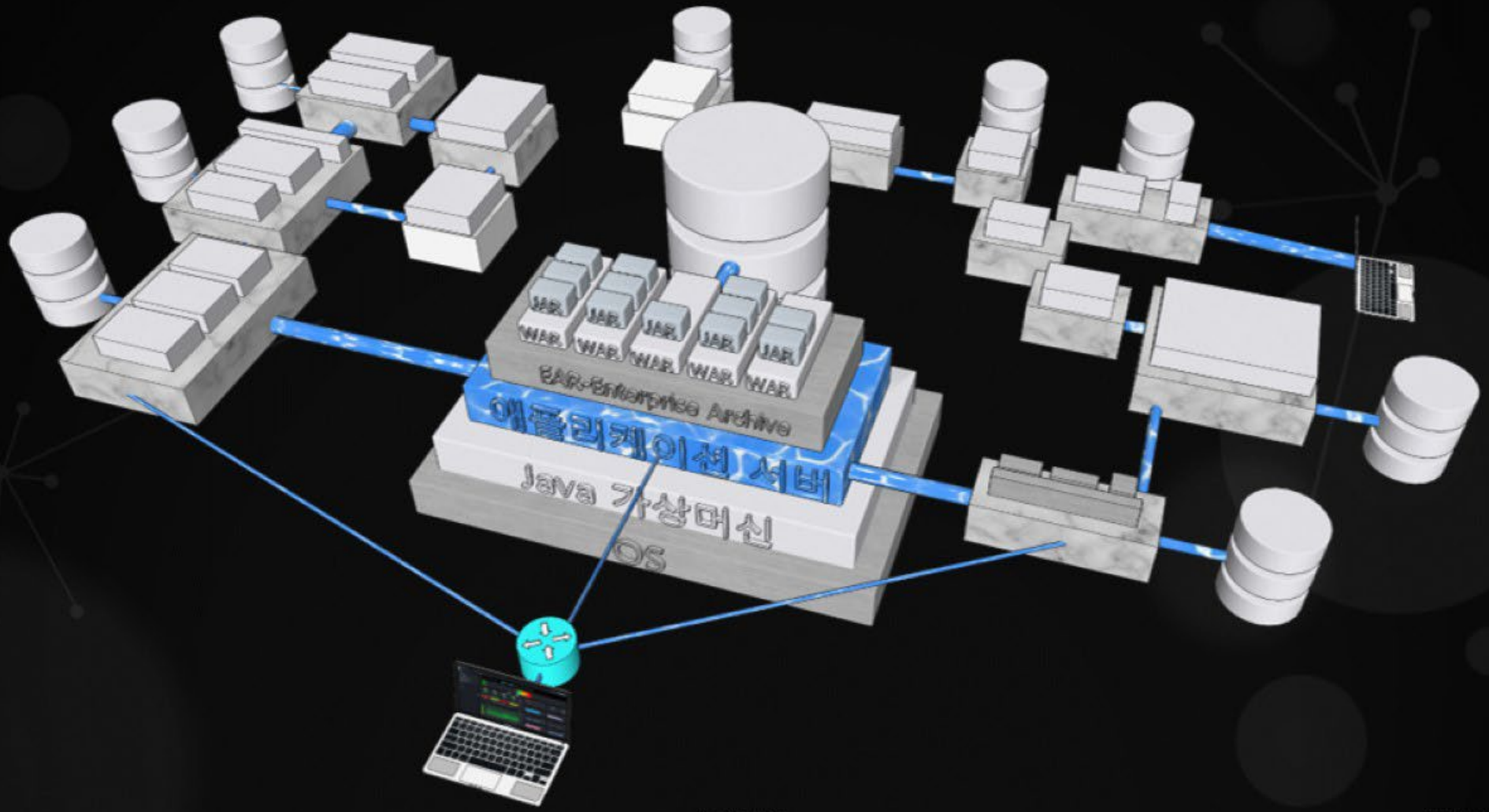
동일한 조건으로
서비스 분할



모노리스에서 마이크로서비스로 전환 – Phase 4

모노리스에서 마이크로서비스로 전환 – Phase 5

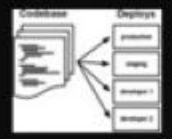
- 개선을 반복하여 최적의 마이크로 서비스를 제공



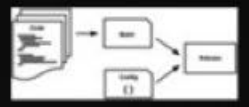
Cloud Native Application(애플리케이션 현대화) 도전!

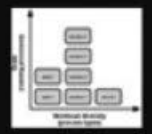
openmaru

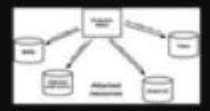
- 클라우드의 이점을 최대로 활용할 수 있도록 애플리케이션을 구축하고 실행하는 방식
- 신속한 개발과 클라우드 확장성 확보를 위한 클라우드 네이티브 애플리케이션 개발은 필수
  - SaaS 12-Factor, Cloud, MSA, Container, CI/CD 등 DevOps 중요
  - https://landscape.cncf.io/



12-Factor applications







Independent, autonomous, stateless, processes

Configuration sources, service references

OpenJDK kafka

Microservices redis

CouchDB mongoDB.

Application as composition of polyglot services

DevOps

Build, package, deploy, observe

The 12 Factor App (1/2)

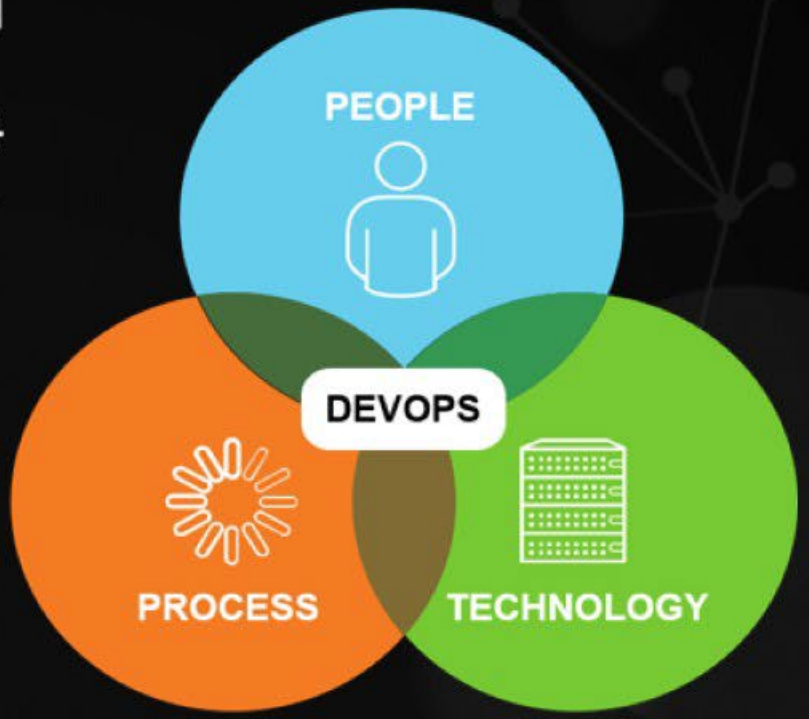
- **I. 코드베이스**
  - 버전 관리되는 하나의 코드베이스로 여러 곳에 배포
- **II. 종속성**
  - 의존 관계를 명시적으로 선언하고 분리
  - 환경에 의존하지 않도록 함
- **III 설정**
  - 설정을 환경 변수에 저장하기
- **IV. 백엔드 서비스**
  - 백엔드 서비스를 연결된 리소스로 취급
- **V. 빌드 릴리스, 실행 (Build, release, run)**
  - 빌드, 릴리스, 실행 3 단계를 엄격하게 분리
- **VI 프로세스**
  - 응용 프로그램을 하나 또는 여러 개의 독립적인 프로세스로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

DevOps 란 무엇인가?

- 개발 (Development) 과 운영 (Operations) 의 합성어
- 개발담당자 및 운영 담당자의 협력 개발방법론
- DevOps사업 부문을 추가 한 BizDevOps 라는 단어로도 확산

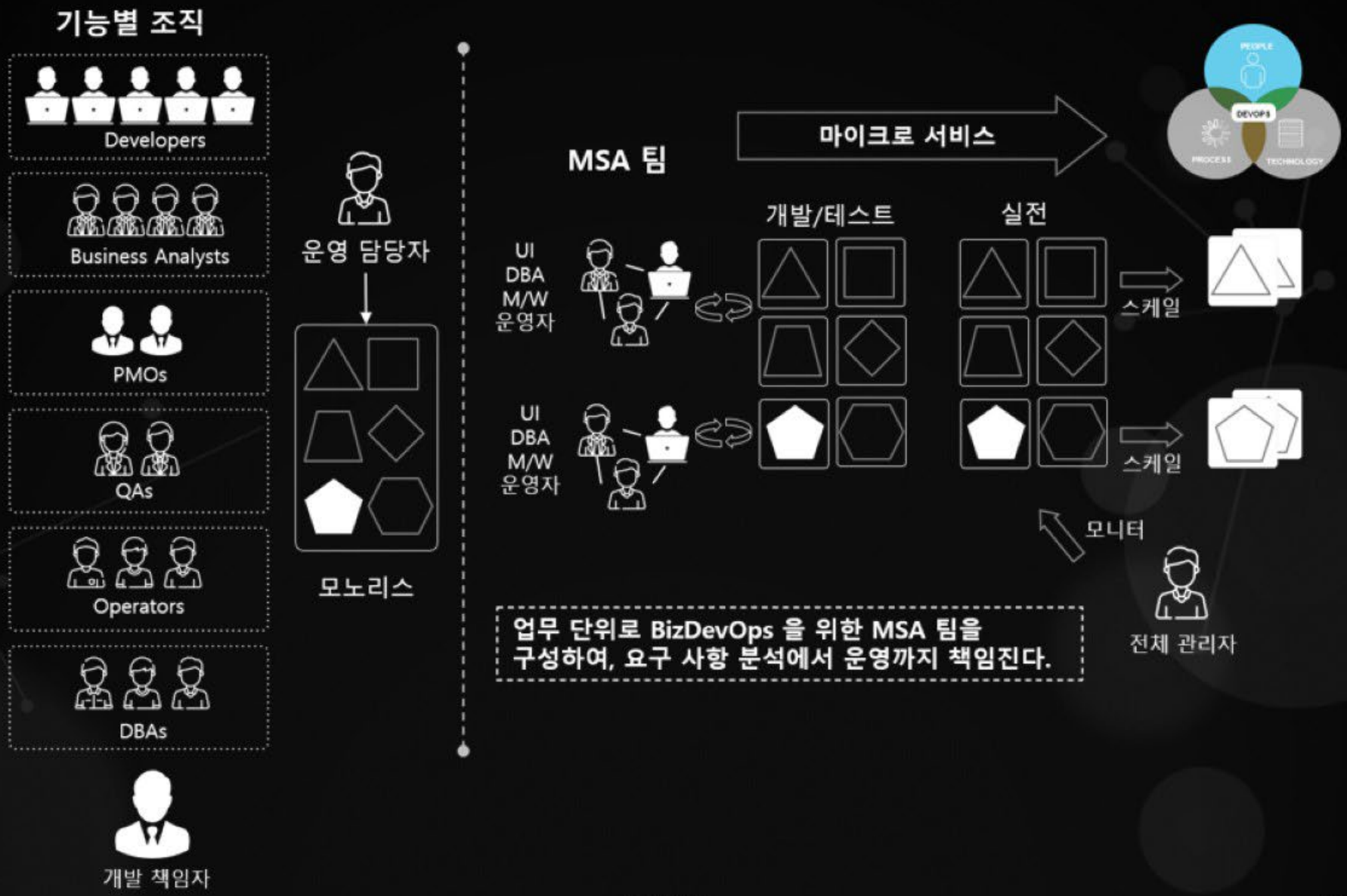


DevOps 목표

아이디어에서 구현까지의 기간을 압도적 단축

- 사용자 요구의 조기 실현
- 개발 기간 단축으로 비용 절감
- 짧은 주기의 혁신에 의한 품질 향상
  - 린스타트업 핵심
  - 디지털 트랜스 포메이션 필수 도구

시스템 개발과 운영 방법론의 혁신 DevOps



레거시 빌드 배포 환경의 해결 해야하는 과제들

openmaru

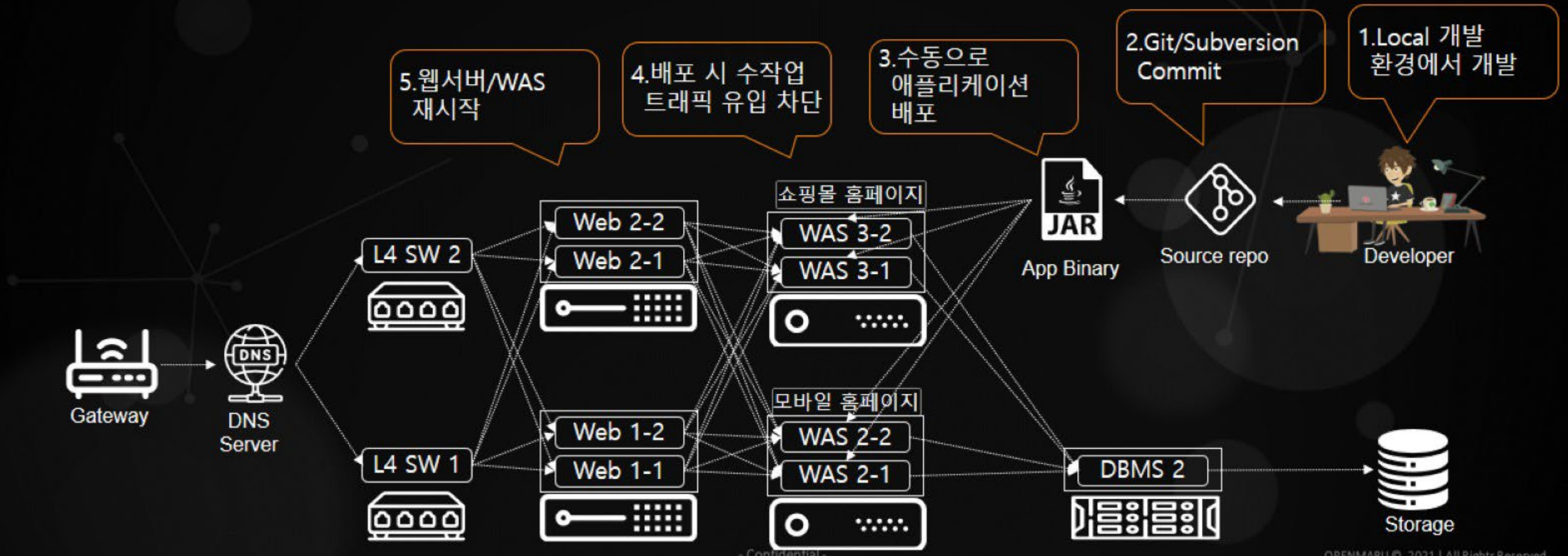
설계와 구축보다, 테스트와 배포에 시간이 더 소요됨

수작업으로 배포하면서 사람의 실수로 인한 문제가 발생

IT 개발과 운영을 위한 자동화 도구들이 없는 경우

기존 레거시에서 빌드/배포

- 수작업을 통한 복잡하고 반복적인 배포 플로우
- 시스템에 경험이 많은 배포 전문가가 집중에서 배포
- 애플리케이션 원복시에도 배포만큼의 수작업이 필요함

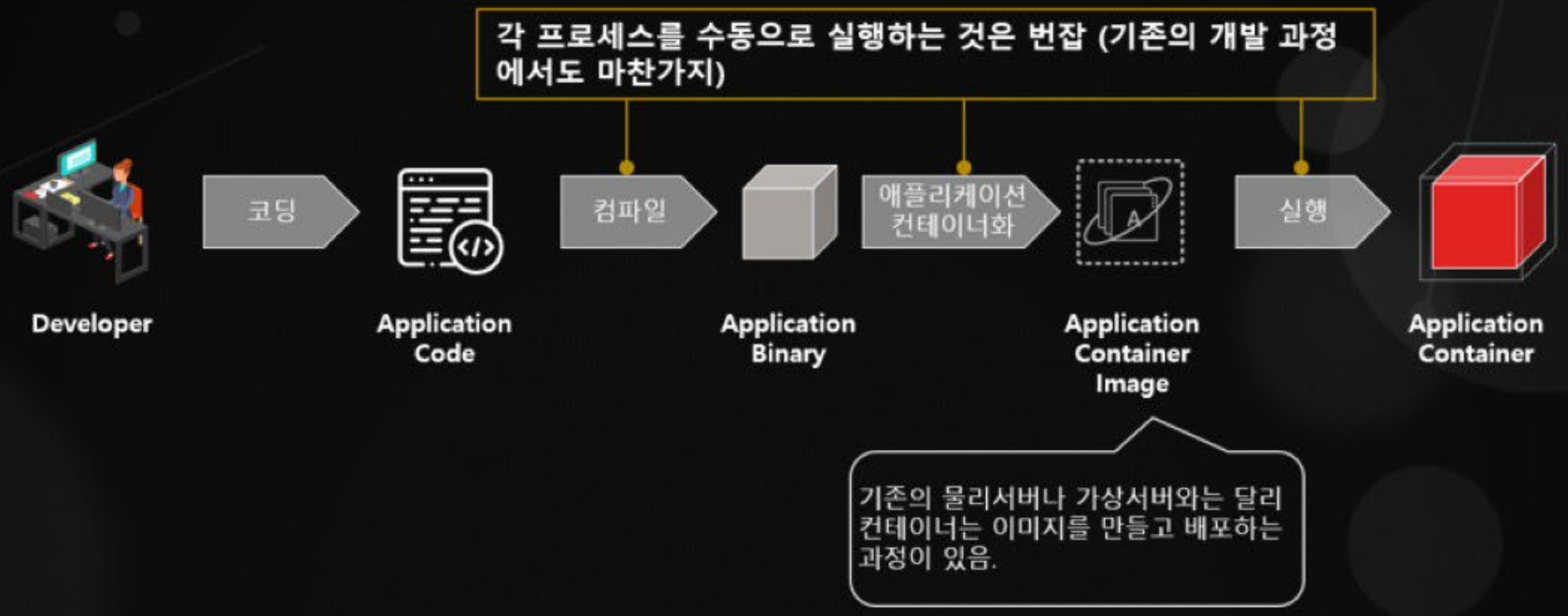


컨테이너 기반의 애플리케이션 빌드 배포 프로세스 예

openmaru

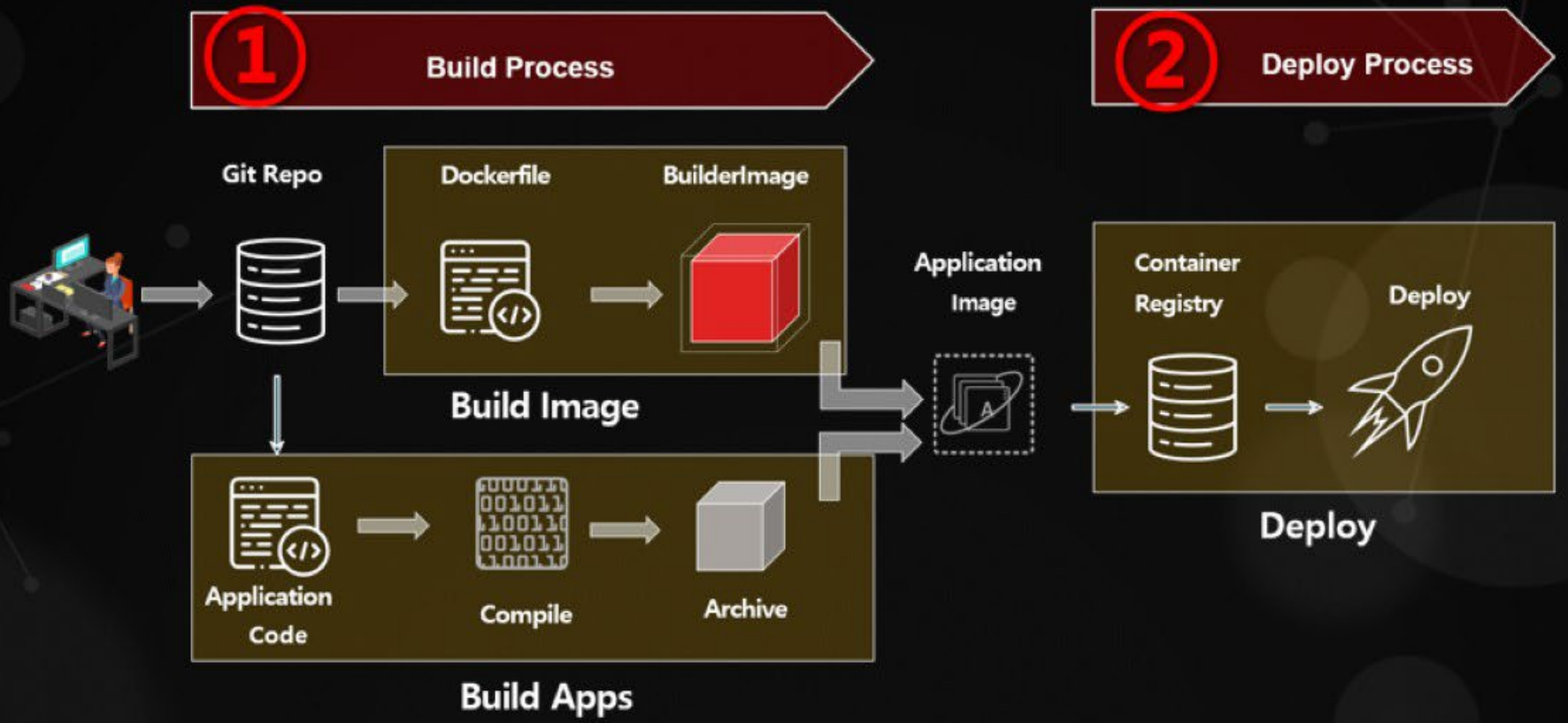
①

- 컨테이너 기반에서 애플리케이션을 개발하는 경우, 프로그래밍 테스트 단계의 일반적인 절차
- 컨테이너 이미지화 하여 개발 생산성 향상
  - 손쉬운 애플리케이션 환경 구성
  - 개발/스테이징/운영 환경의 차이 제거



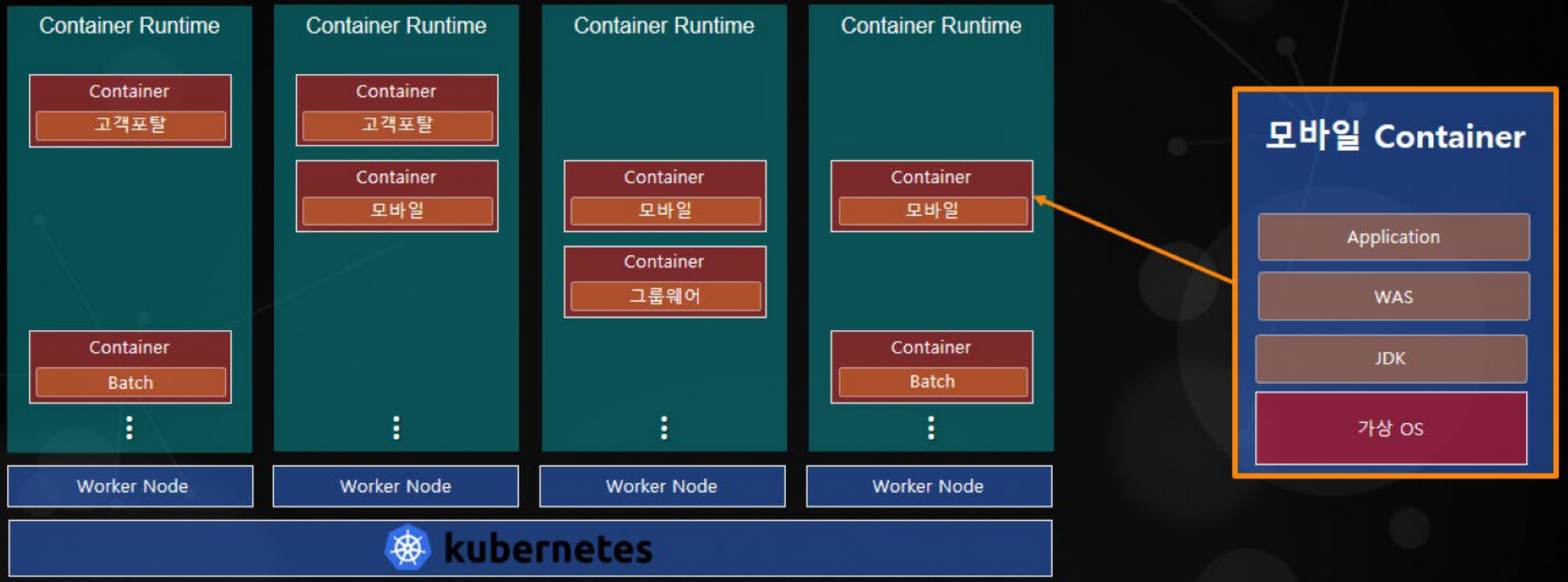
Build & Deploy Application

- Build Process 소스 코드 빌드 및 기반 이미지 빌드로 구성
- 애플리케이션 실행은 " Build Process "와" Deploy Process "로 구성



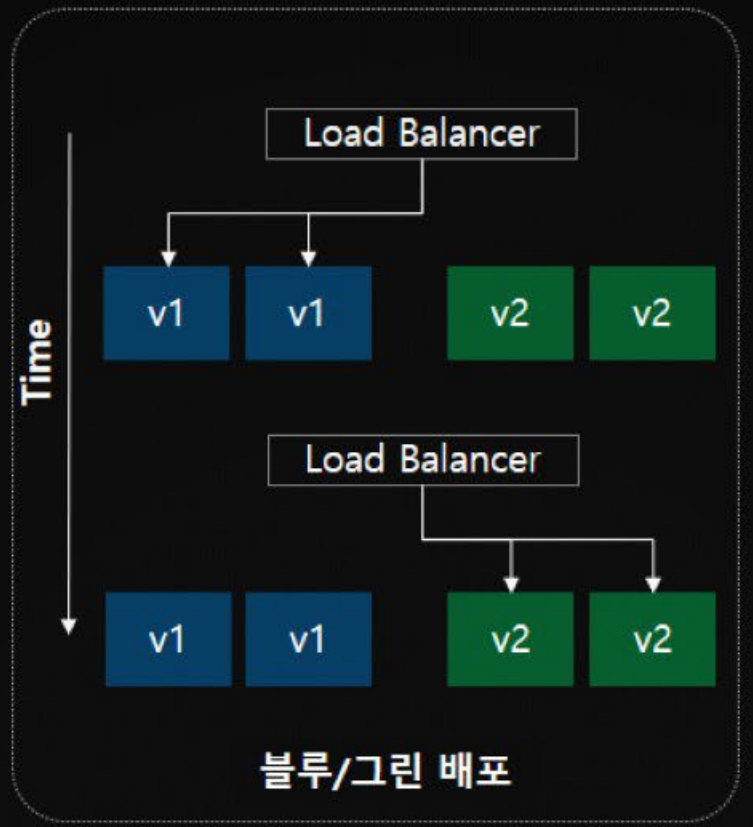
컨테이너 환경의 소프트웨어 컴포넌트 요약 구성

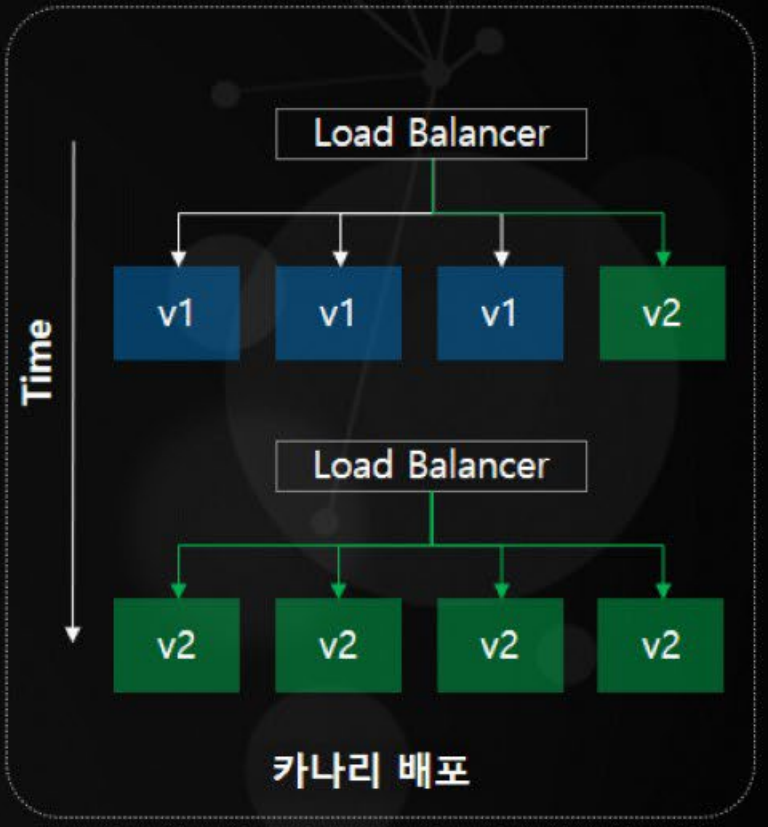
- **Container는 가상OS, JDK, WAS, Application이 포함된 Image를 기반으로 기동됨.**



무중단 뿐만 아닌 고도화된 배포 방식

- 롤링 업데이트 - 점차 새로운 버전으로 변경
- 블루/그린 배포 – 블루 버전을 유지한채로 그린 버전을 테스트 (신속한 롤백)
- 카나리 배포 – 소규모로 검증 후 전체 반영 (신속한 롤백)





클라우드 네이티브 환경의 애플리케이션 모니터링 어려움

openmaru

6. Monitor, log and troubleshoot from the start

The construction of applications from a set of microservice Legos considerably complicates how to monitor and troubleshoot systems and their performance. Various microservices often trigger a cascade of events that leads to an application failure. To minimize failures -- which aren't a *maybe*, but a reality -- incorporate monitoring and troubleshooting into microservices design.

https://www.techtarget.com/searchitoperations/tip/Follow-these-6-steps-to-deploy-microservices-in-production

- 마이크로 서비스 아키텍처일수록 **모니터링 방법이 상당히 복잡**해 집니다.
- 마이크로 서비스 아키텍처에 **모니터링과 트러블 슈팅 방법이 고려**되어야 합니다.

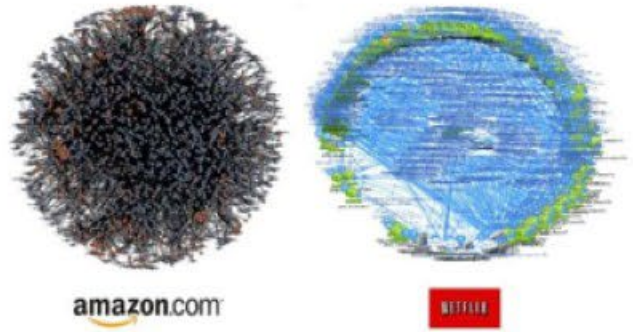
- Uber는 2014년 말 에 4,000개가 넘는 독점 마이크로 서비스와 점점 더 많은 수의 오픈 소스 시스템이 모니터링 시스템에 문제를 제기했다고 보고
- 컨테이너 인프라는 모니터링 시스템이 필수적인 환경
- **마이크로 서비스에 특화된 모니터링 시스템이 필요**

2. Microservices instead of a monolith

Following a microservice architecture, a typical monolith application would be broken down into a dozen or more microservices, each one potentially running its own programming language and database, each one independently deployed, scaled and upgraded.

Uber for example reported in late 2014 over 4,000 proprietary microservices and a growing number of open source systems which posed a challenge for their monitoring system.

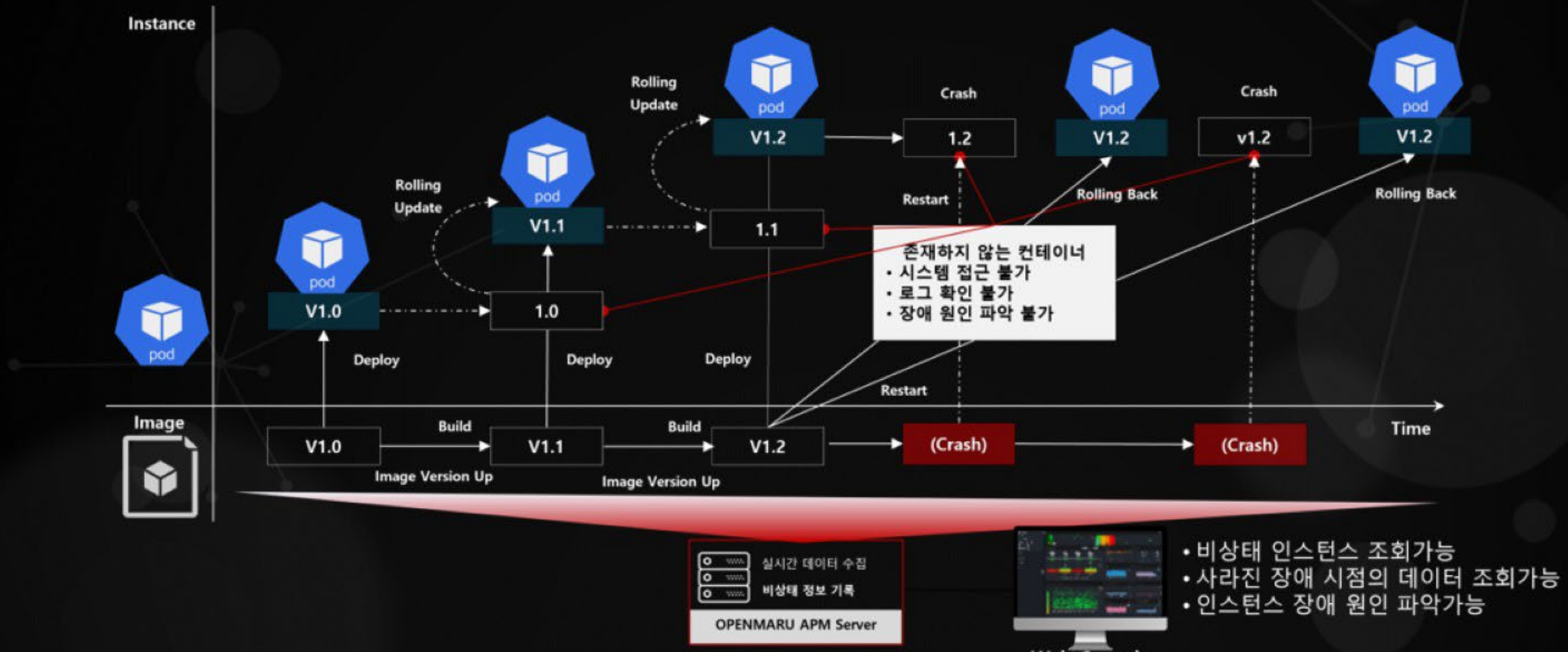
The Challenge: A surge in the number of discrete components you need to monitor.



https://www.infoq.com/articles/microservice-monitoring-right-way

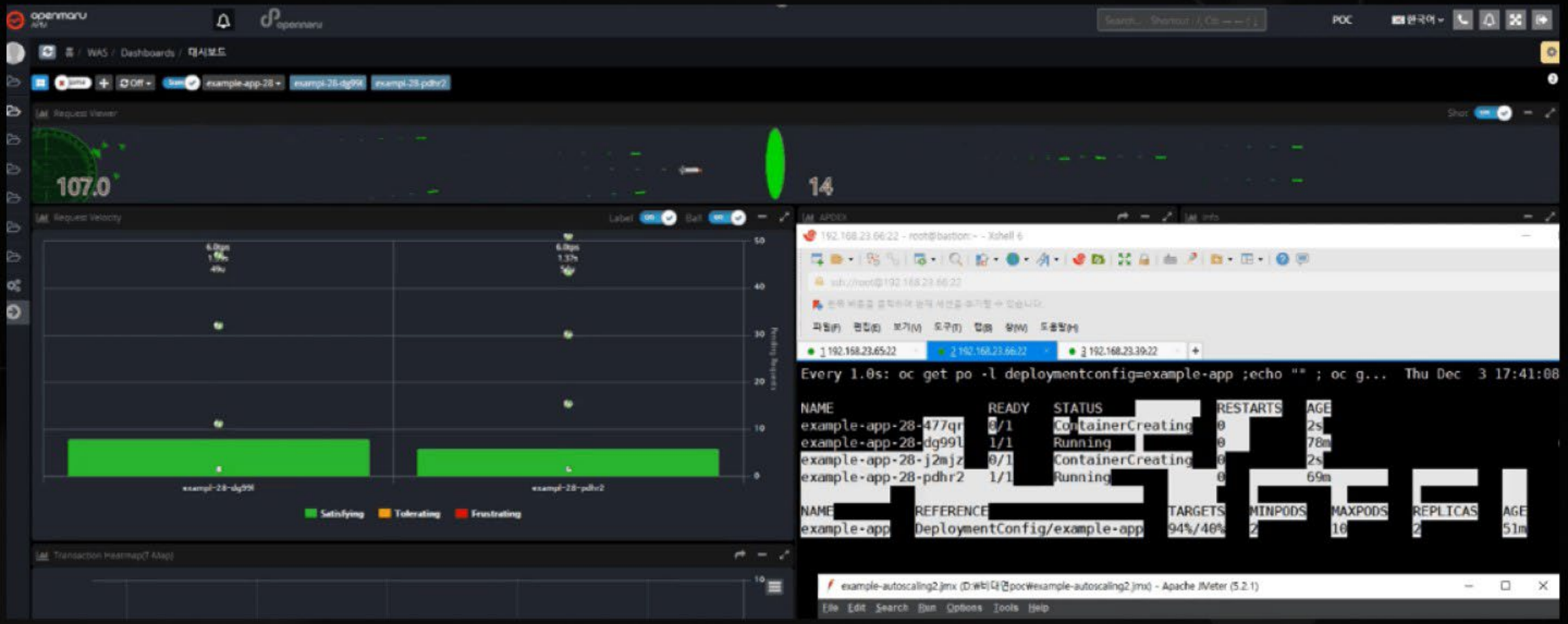
컨테이너는 무상태 (Immutable) 인프라스트럭처로 상태를 저장하지 않음

- PaaS 환경에서 컨테이너가 중지된 후 장애원인 파악을 위한 방법을 제공
- APM 에서 사라진 컨테이너에 대한 정보를 보관하여 장애원인을 파악할 수 있음



컨테이너 환경에서의 애플리케이션 모니터링

- **Auto Scaling으로 Container(Pod)가 유연하게 Scale Out/In이 발생하는 환경**
- **클라우드 네이티브 애플리케이션을 트러블 슈팅할 수 있는 기능이 있는 모니터링 시스템이 필요함**




openmaru

제품 / 서비스에 관한 문의

- 콜 센터 : 02-469-5426 (휴대폰 : 010-2243-3394)
- 전자 메일 : sales@opennaru.com