
openmaru
APM
클라우드 네이티브 환경에서 개발자가
꼭 고민하는 이슈와 해결 방안

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 지원을 위한 애플리케이션 현대화

- **Lift & Shift** – 이동성은 애플리케이션 컨테이너화만으로는 부족
- 클라우드 네이티브 애플리케이션을 통한 클라우드 완벽 지원

기존 온-프레미스 사용자

***Lift & Shift** Approach*

컨테이너 전환 도구

새로운 클라우드 사용자

***Cloud Native** Approach*

DevOps 구현 도구

Reliable ⟷ **Agile**

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

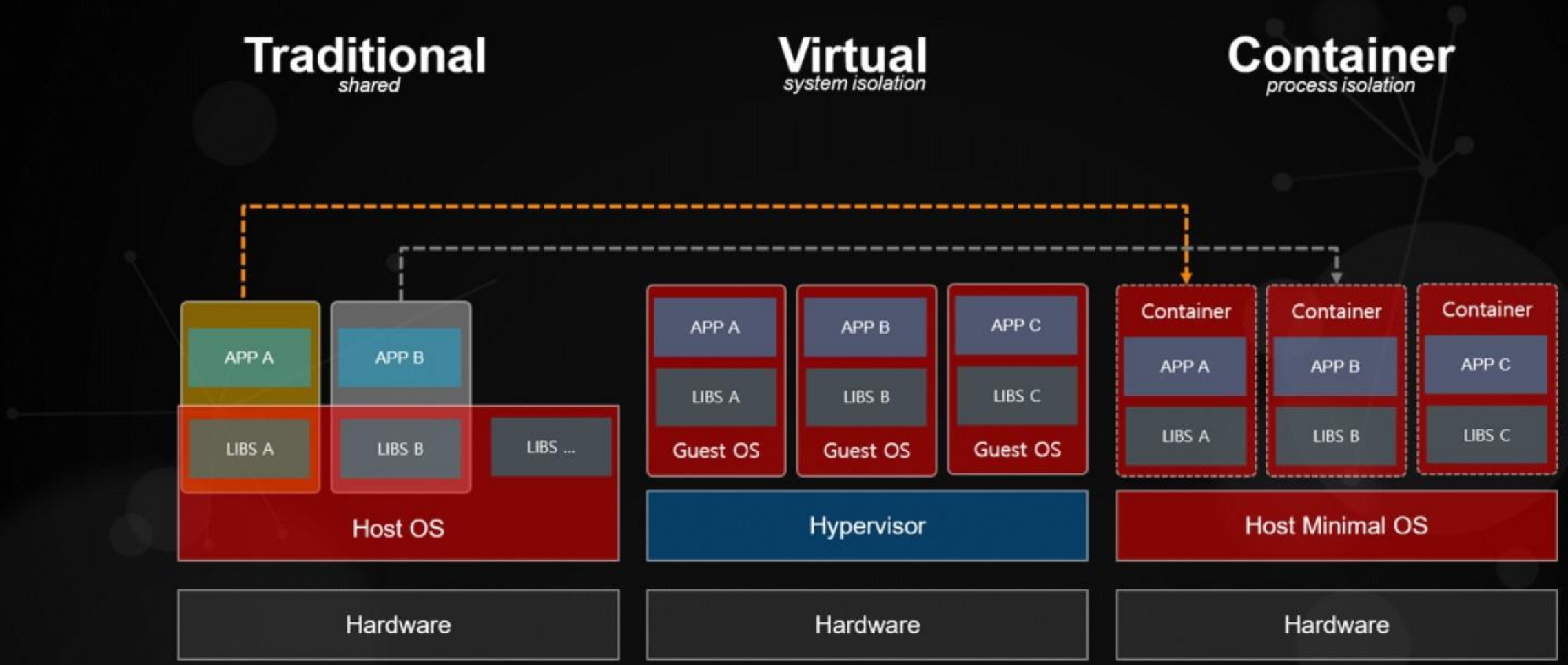
- 클라우드를 이용하고 있음에도 불구하고 수작업 프로세스에서 벗어나지 못함
- 운영 인력 부족과 업무 효율성을 개선하지 못함

클라우드 네이티브와 개발팀

openmaru
APM

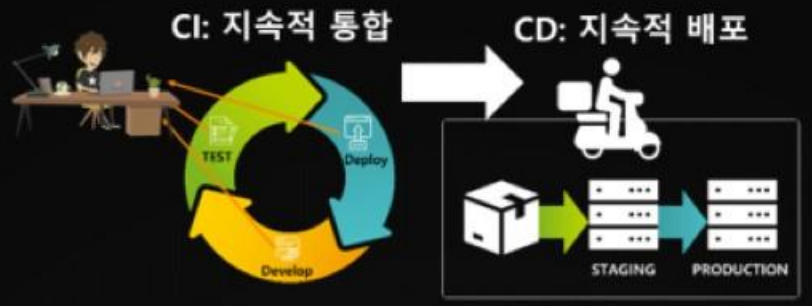
Evolution

- 물리서버, 가상화 환경 자동화 부족
- 단순 컨테이너가 아닌 클라우드 네이티브를 고려한 자동화 환경으로 전환

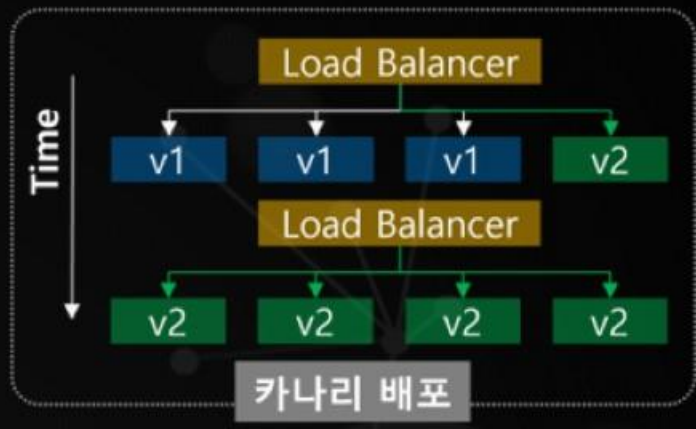


개발자 측면의 컨테이너 기반 환경의 장점

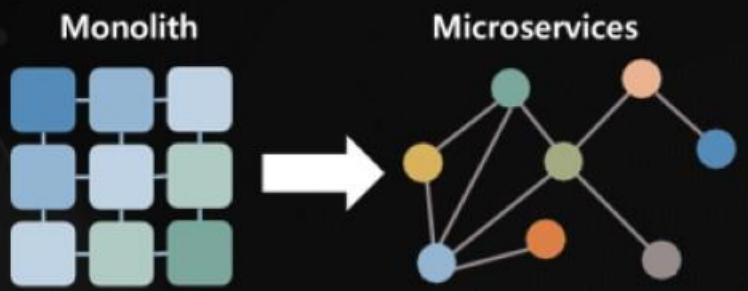
컨테이너 기반의 빠른 개발환경



신속한 개발과 편리한 배포



서비스 무중단



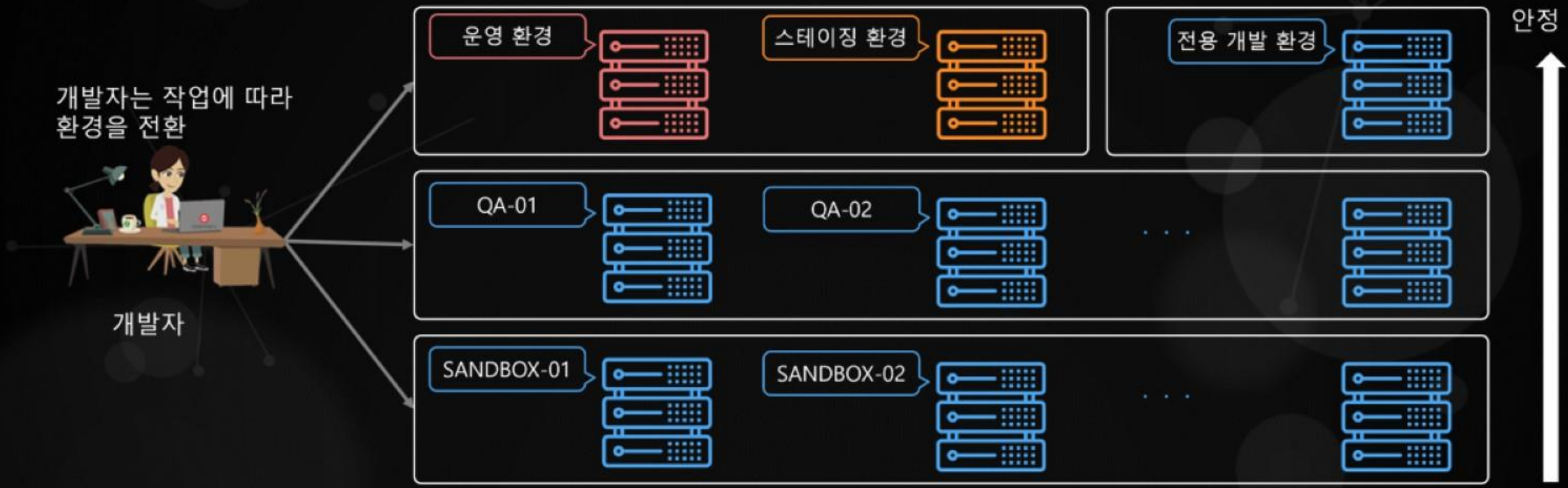
MSA개발에 적합한 환경

DevOps기반의 민첩한 개발

대규모 시스템의 개발을 운용을 위한 개발 환경

- 기본적으로 개발 환경은 여러 서버가 필요하며, 개발팀이 커질 수록 많이 필요
- 운영/스테이징/개발 및 샌드박스 환경 구축

“ 개발 환경이 늘어나는 원인 = 버전 × 용도 × 개발자 “



개발팀과 운영팀 누구의 잘못인가? 불행의 시작

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

기존개발 운영 환경

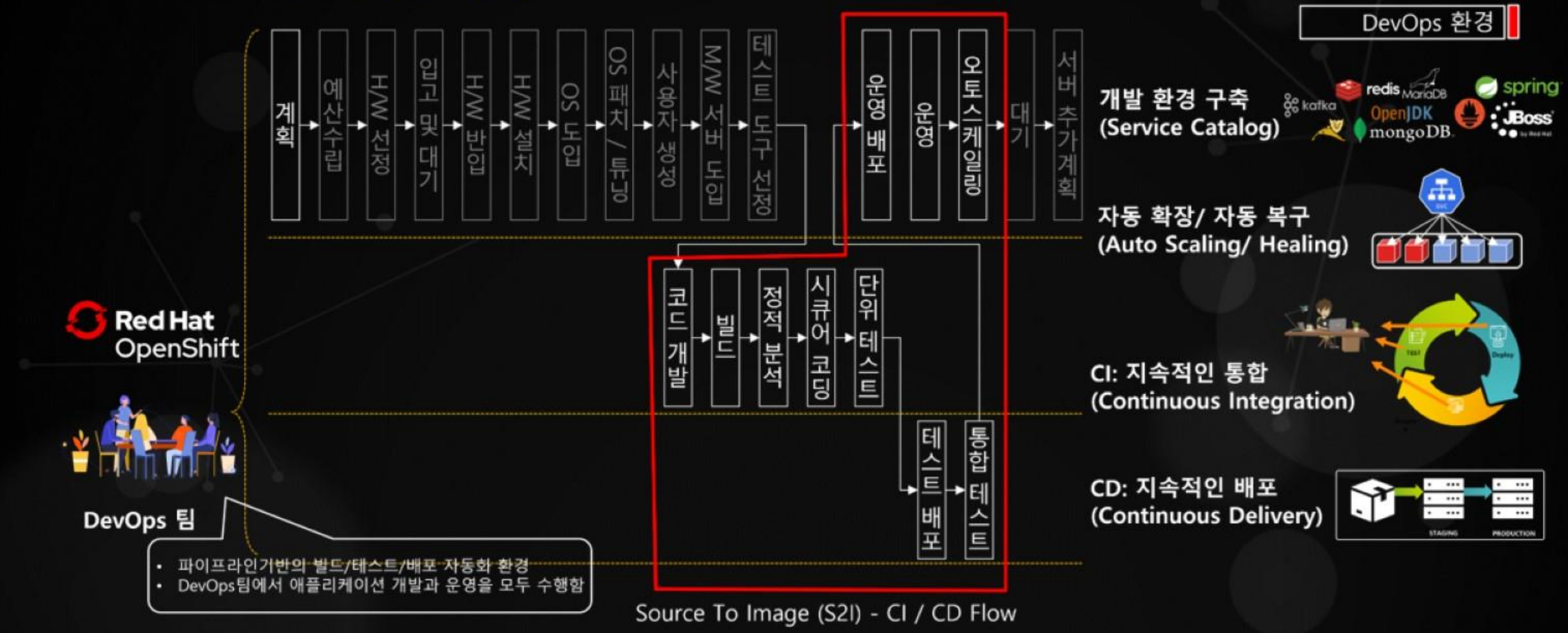
계획 → 예산수립 → H/W 선정 → 입고 및 대기 → H/W 반입 → H/W 설치 → OS 도입 → OS 패치 / 튜닝 → 사용자 생성 → M/W 서버 도입 → 테스트 도구 선정 → 운영 배포 → 서버 구성 및 설정 → 운영서버 이전 → 운영서버 가동 → 서버 추가계획

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 환경을 이용하여 신속한 개발 및 운영 환경 구축
- 필요에 따라 구축과 삭제가 편리




Cloud Native Application 도전...

클라우드 네이티브??? 개발팀에서 뭘 해야 하나요?

openmaru

클라우드 환경에서
개발이 뭐가 다른 지?

클라우드 네이티브에 대해
개발팀이 알아야 하나요?

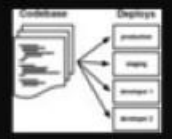
개발하기도 바쁜데
운영팀과 협업을
어떻게?

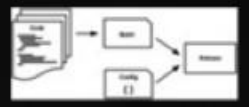
CI/CD 에 대한
적합한 책임자는
누구일까요?

대용량의 견고한 시스
템을 위한 노력 필요

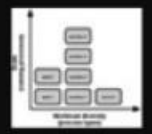
Cloud Native Application 도전!

- 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

하지만 현실은?

openmaru

공론화 전에 파일럿으로
살짝 해봤으면...

괜히 말 꺼내서
독박 쓰는거 아냐?

갈아 엎고 새로?
일부 기능 하나씩?

클라우드 네이트브?
Spring Cloud?

파일럿 환경은 누가?

클라우드에 대한 개발팀과 운영팀의 고민들

- **Cloud Native Computing**은 클라우드의 특성과 장점을 적용하여 구성된 컴퓨팅 환경으로, 인프라, 플랫폼, 어플리케이션/서비스와 개발, 운영, 관리의 전체 영역을 대상으로 함

맨날 시스템 장애에 변경사항만 엄청 나네!!

개발을 빨리해도 , 1주일에 1번 배포인데 무슨 소용?

개발/운영 하기도 바쁜데, 오픈소스 스터디는 어떻게?

사용자와 시스템이 늘어 나면 당연히 운영 인력을 충원 해야지 !!!

인프라에 투자한다고 서비스가 좋아지나? 안정된 서비스가 중요하지 !!!

개발팀은 책임도 못질 오픈소스만 자꾸 할까? 제대로 사용하려면 몇 년은 공부 해야지!!

애플리케이션 → 클라우드 네이티브 애플리케이션/서비스

기존 운영환경 → 클라우드 네이티브 운영 환경

혁신적인 IT 환경 구축
IT 조직의 운영 비용 절감과 비즈니스에 대한 민첩성 증대

클라우드에 최적화된 표준화
통합로그, 통합모니터링, 배포 자동화, 소스형상관리, 환경구성 표준화 등

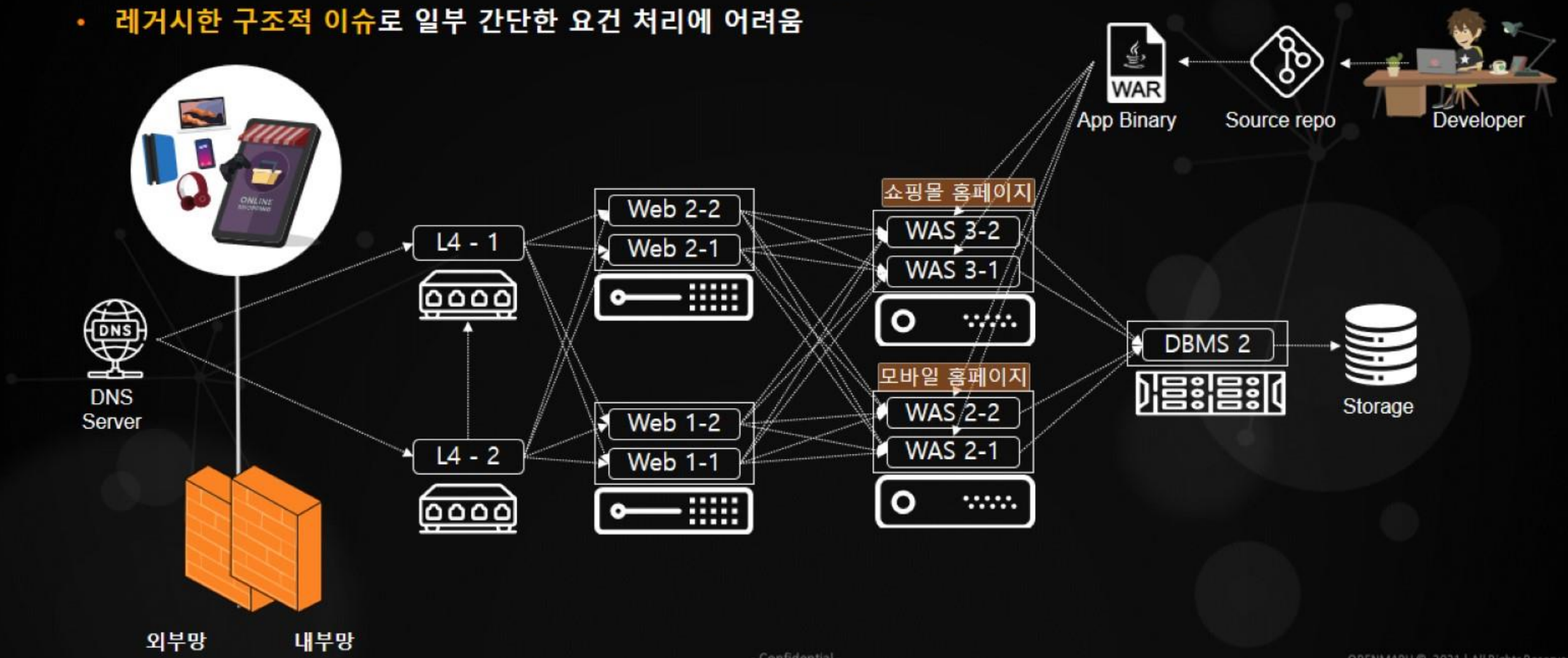
PaaS
개발팀에서 스스로 시스템 S/W 를 설치/구성, **개발에만 집중**

MSA
서비스 변경이 필요하면 언제든지 배포 (1일 10회 이상)

운영자동화
사용자가 폭증하더라도 인력에 의존하지 않고 안정성 있는 서비스

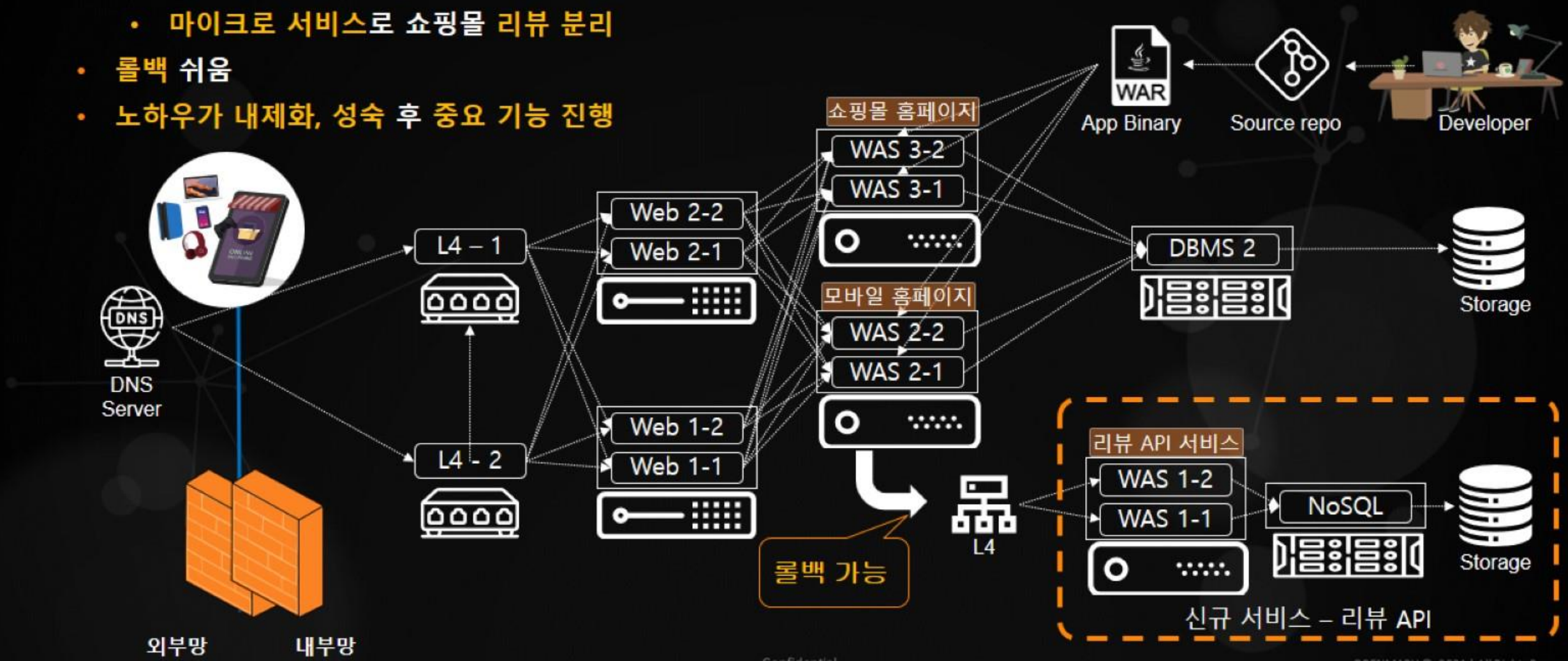
일반화 되어 있는 모놀리틱 아키텍처

- 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

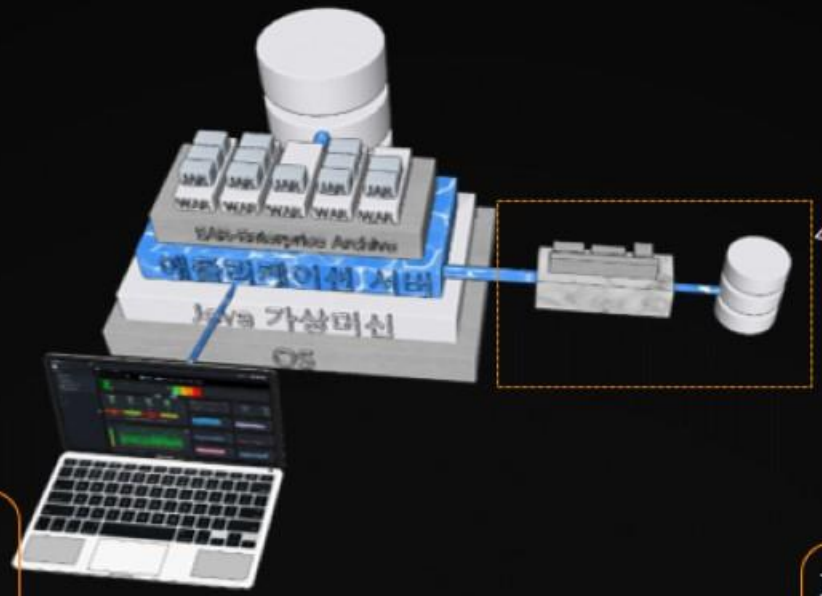


Microservice Pilot – 리뷰API 서비스 도입

- 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



이슈 - Microservice Pilot – 리뷰API 서비스 도입



호출 빈도가 높으며, 독립적으로 동작하는 모듈을 API 서비스로 독립 – 리뷰API

자원(VM) 및 OS/WAS 설치 요청합니다.

빌드/배포 자동화도 해주세요.

내부 API 용 L4 가 필요 해요.

개발팀

전문 엔지니어 일정 확인 해 볼께요.

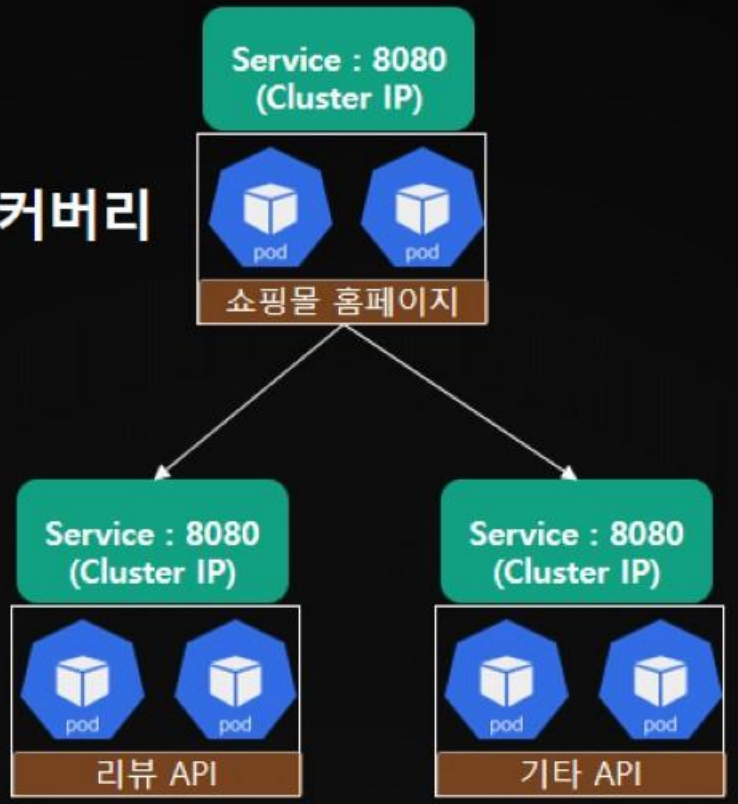
자원이 여유 있는지 확인 해볼께요.

내부 L4는 없어요.

운영팀

애플리케이션 배포 / 로드 발란서 / 서비스 디스커 버리

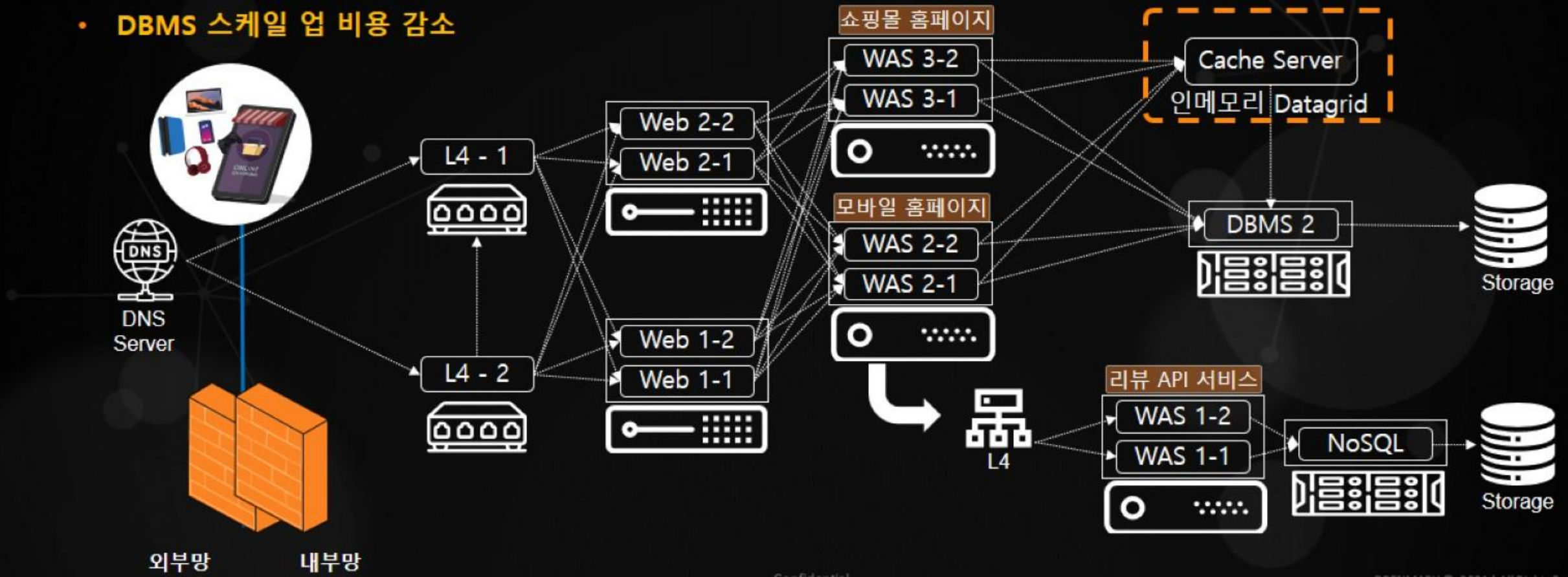
- 서비스 (K8S, OpenShift)
  - 로드 발란서
  - Core DNS 기반 서비스 디스커버리



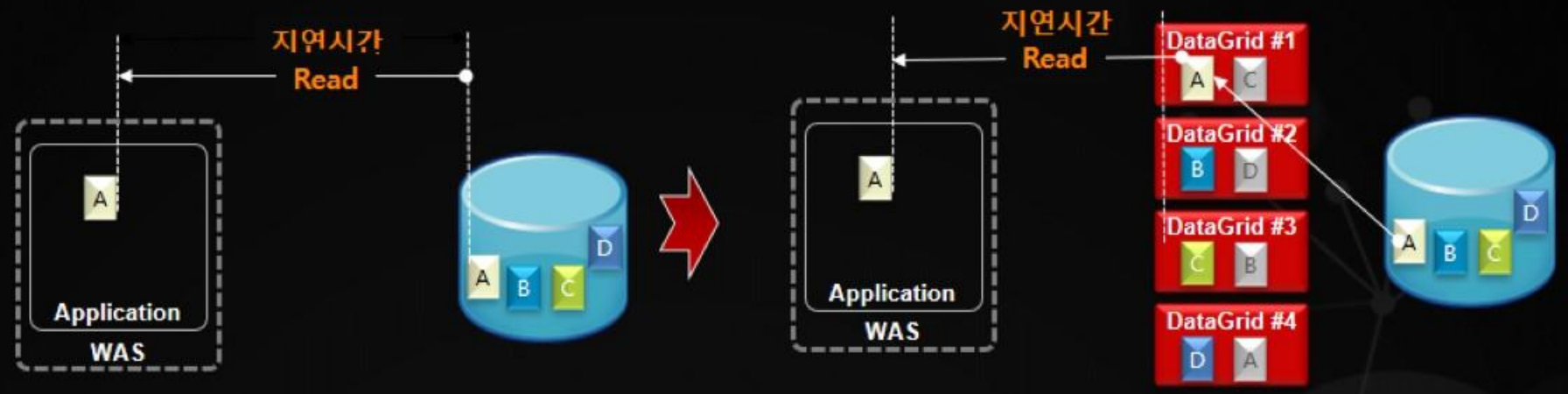
- Jenkins 필요 없음
- S2I(Source To Image) - GIT
  - Source 모드
    - Java
      - Maven, Gradle 등
      - WAS or Spring Boot 등
    - Nodejs, Python, PHP 등
  - Dockerfile 모드
  - Binary 모드
    - WAR, JAR , 드래그앤 드랍으로도 가능!!

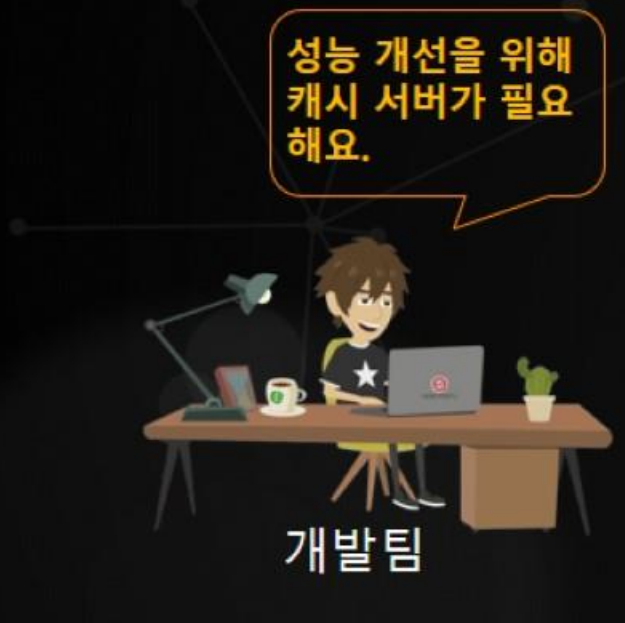
대규모 사용자을 위한 성능 대응 방안 – 엔터프라이즈 캐쉬 서버 도입

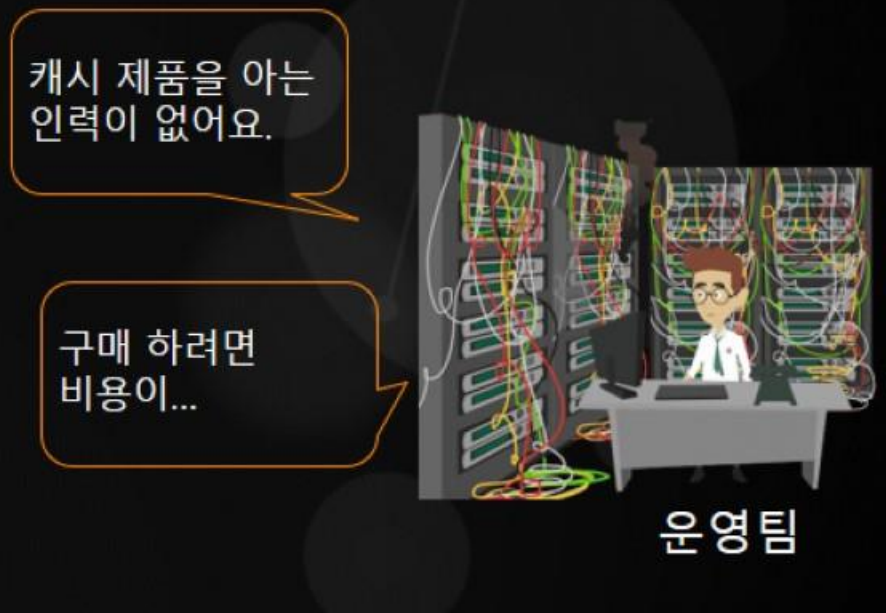
- 웹서비스에서 DB 요청 중 읽기 비율은 약 80% 이상
- 서비스 사용자 증가로 인한 성능 개선과 고가용성 확보
  - Cache Server 도입으로 DB 부하 감소
    - 공통코드, 상품 상세 정보
- DBMS 스케일 업 비용 감소



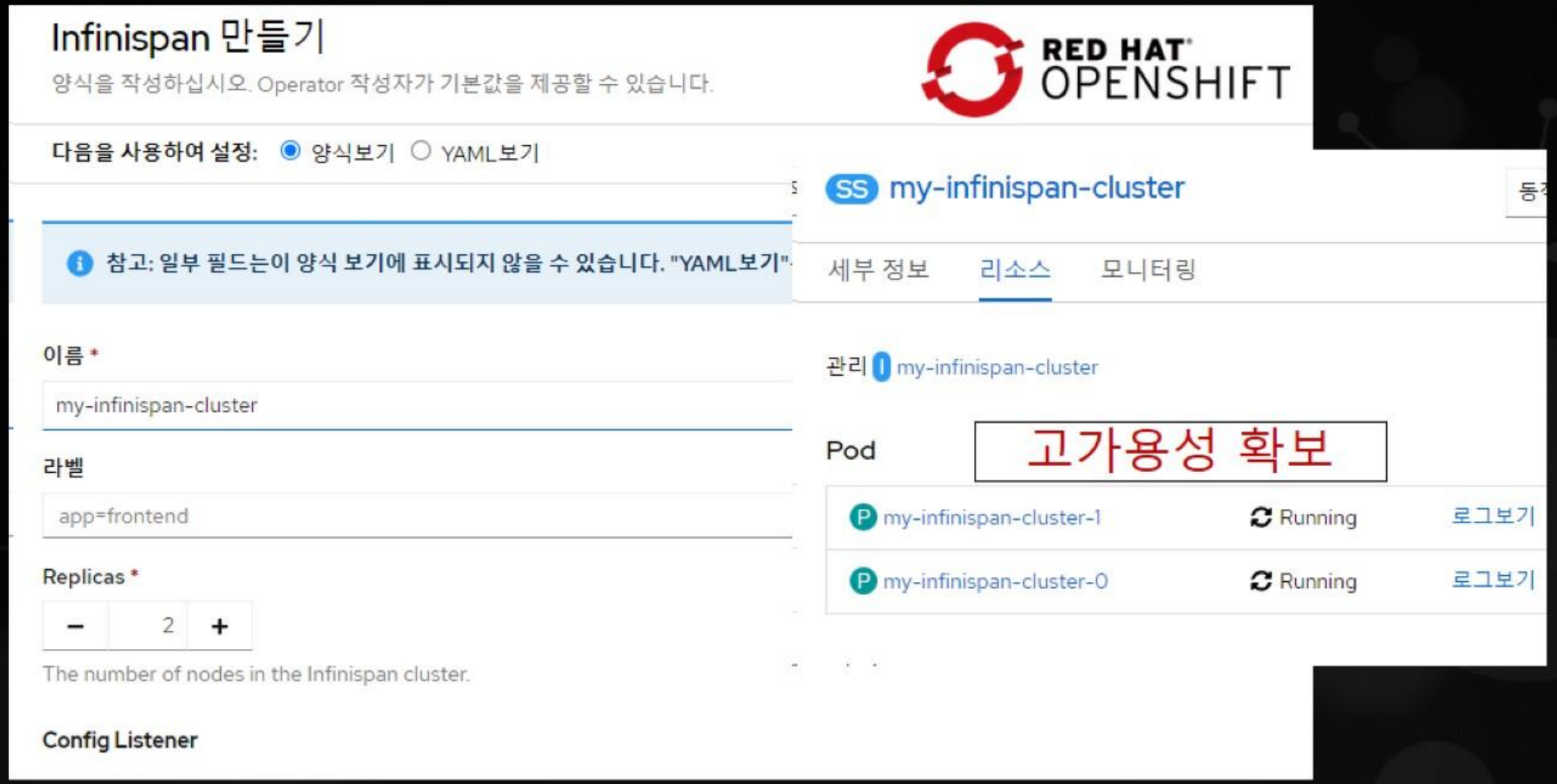
이슈 - 대규모 사용자을 위한 성능 대응 방안 (엔터프라이즈 캐쉬 서버 도입)





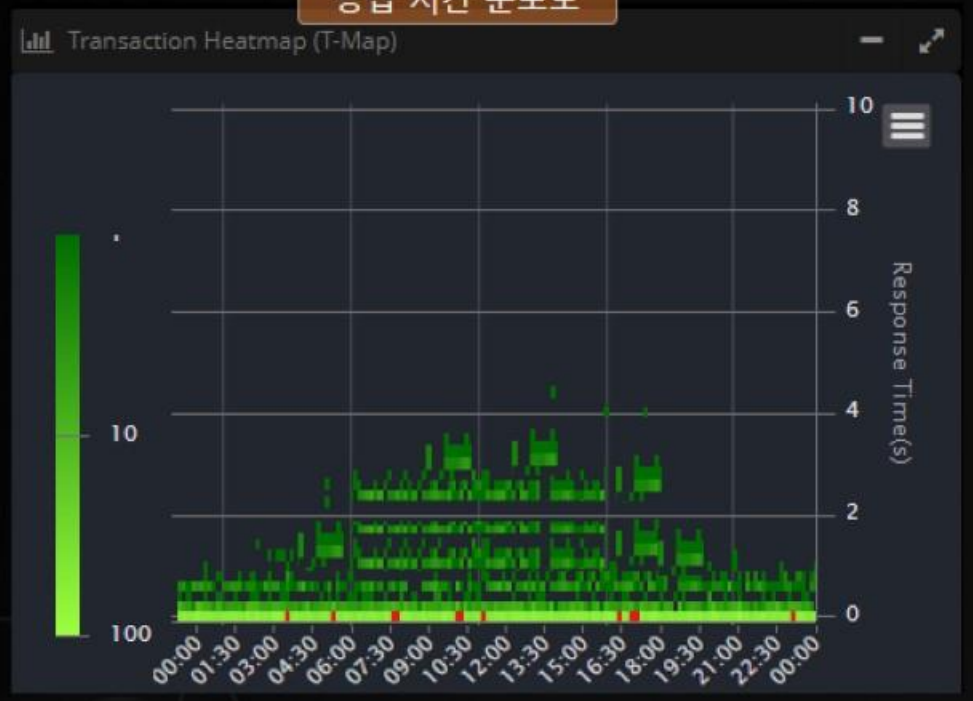


Cache 서버 5분만에 설치하는 방법



Cache 서버 5분만에 설치하는 방법

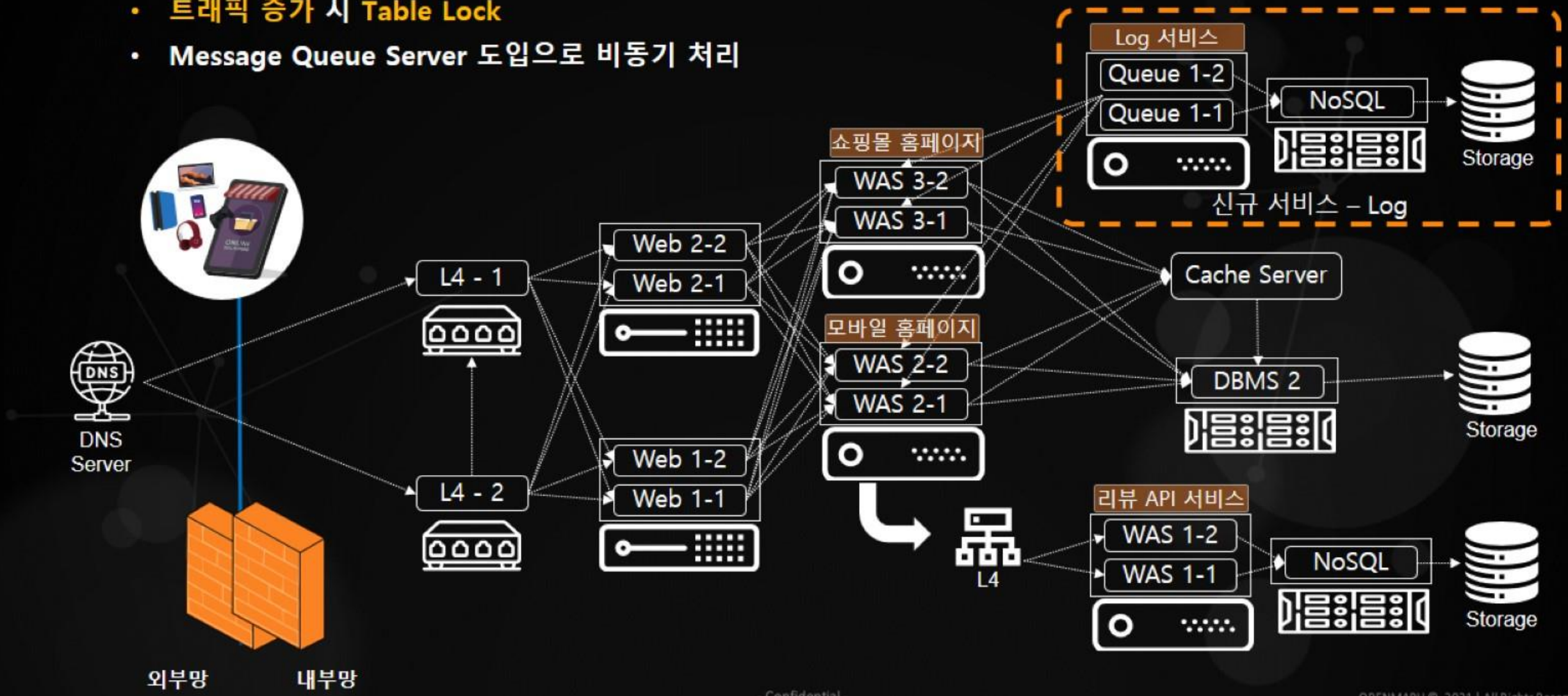
캐시 적용 전
응답 시간 분포도



캐시 적용 후
응답 시간 분포도

대규모 사용자을 위한 성능 대응 방안 – 비동기 처리 메시지 큐

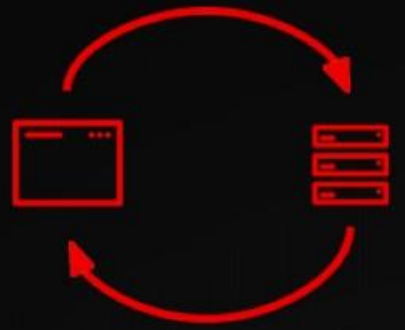
- 보안에 중요한 로그인 히스토리 및 중요 로그 DB화
  - 트래픽 증가 시 Table Lock
  - Message Queue Server 도입으로 비동기 처리



이슈 - 대규모 사용자을 위한 성능 대응 방안 – 비동기 처리 메시지 큐

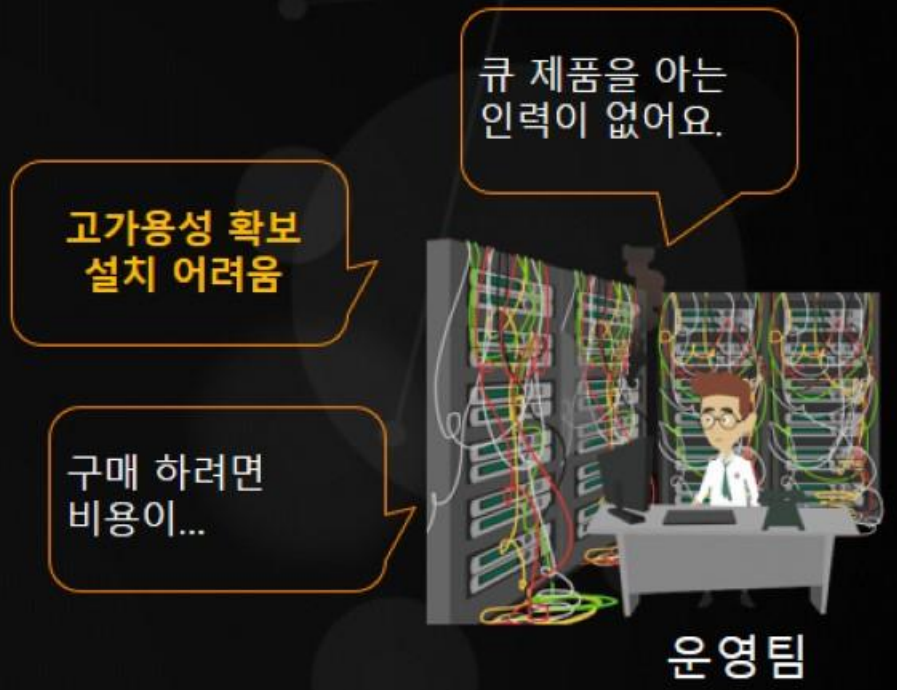
openmaru

동기식 호출



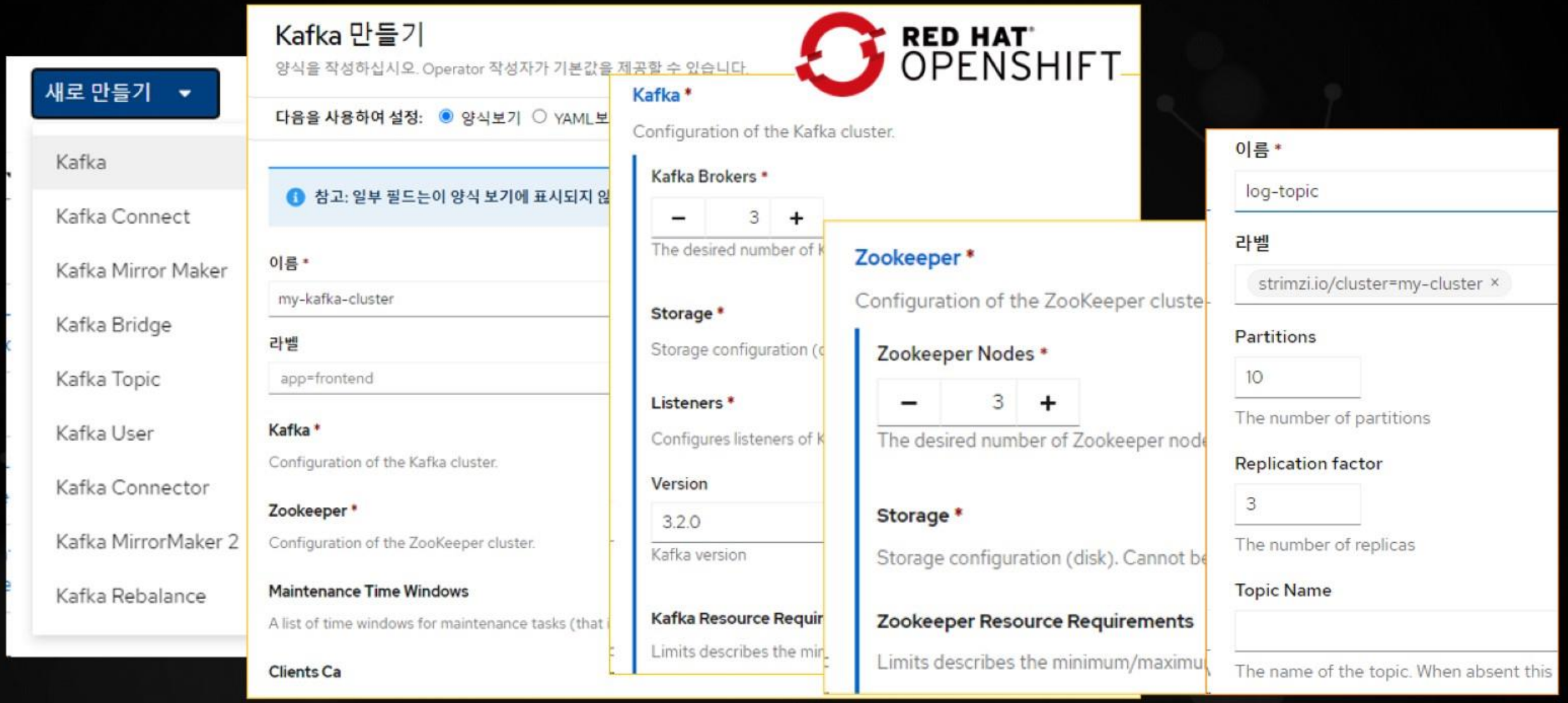
비동기 호출

개발팀

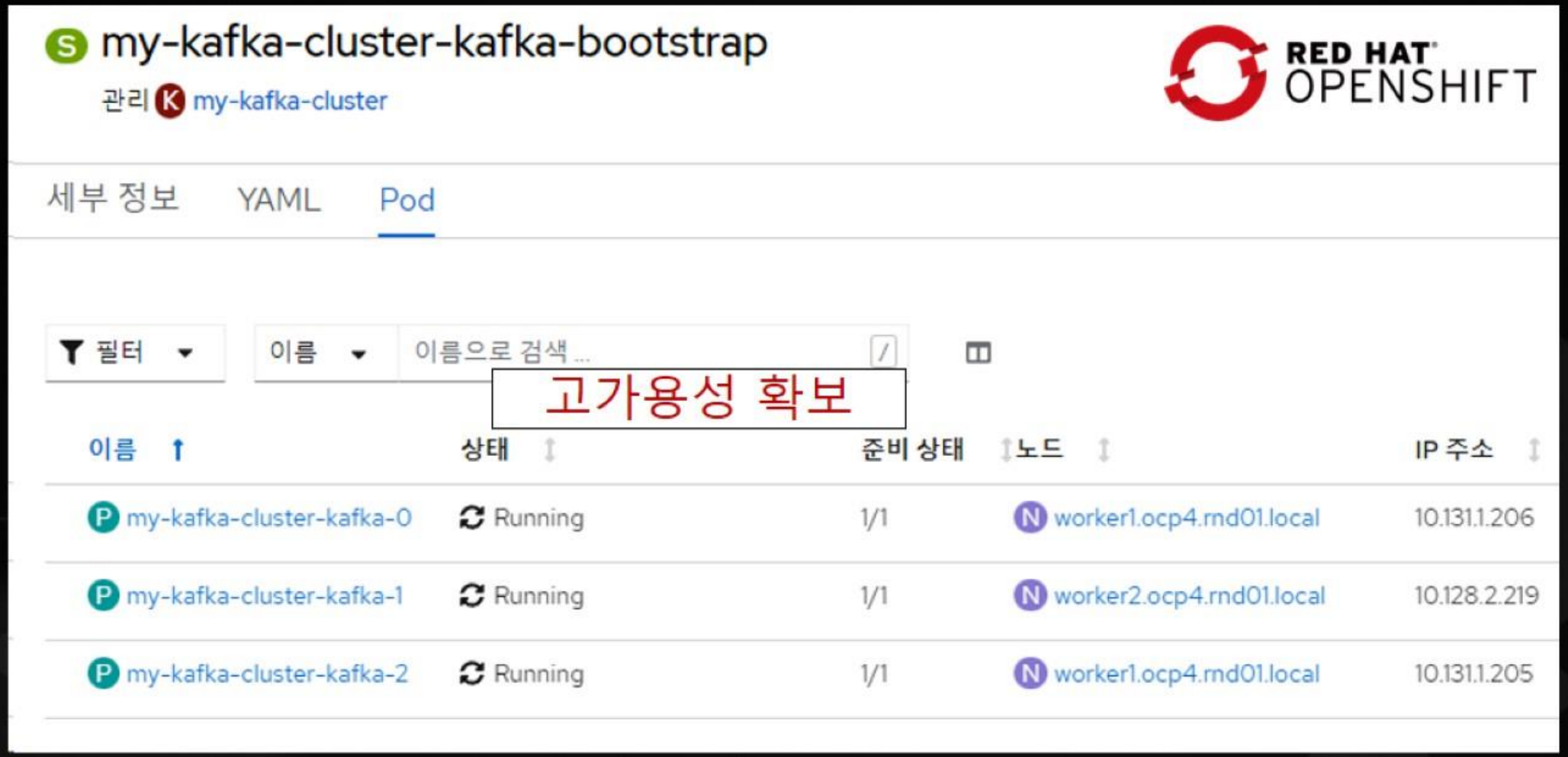


운영팀

Kafka 5분만에 설치하는 방법



Kafka 5분만에 설치하는 방법



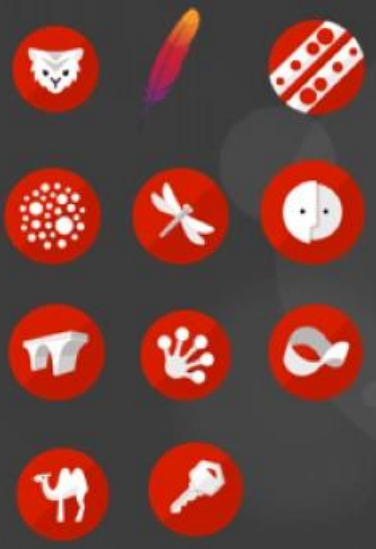

Private Cloud
OpenShift 만의 장점

이슈 해결을 위해 개발자 어떤 선택을?

	Spring Cloud	Private Cloud (K8S, OpenShift)	Public Cloud (AWS 등)
OS(가상, 물리)	인프라팀 요청	필요 없음	인프라팀
라우팅 및 Virtual Host	Spring Cloud Gateway (Netfilx Zuul)	Ingress(Router)	ELB
로드 발란서 (Load Balancer)	Spring Cloud LoadBalancer (Netflix Ribbon)	Service (Core DNS)	ELB
서비스 디스커버리 (Service Discovery)	Spring Cloud Netflix Eureka		
설정 파일	Spring Cloud Config, MQ 또는 개발 소스	ConfigMap, Securet	Parameter Store
추가 미들웨어	IT 팀(인프라팀, 개발팀)	제공 서비스 (Helm, Operator)	제공 서비스
담당 영역	개발팀	Cloud Platform 인프라팀 개발팀	Cloud Platform 인프라팀 개발팀
책임	개발팀		

OpenShift Template 카탈로그 리스트

- OpenShift 에서는 시스템 소프트웨어 자동 설치 할 수 있는 템플릿을 제공
- CI/CD, Databases, Languages, Middleware, 기타 항목의 현재 약 79 개의 템플릿을 제공

CI/CD	Databases	Languages	Middleware	Etc..
	  	     		 

OpenShift Template 카탈로그 리스트

- OpenShift 에서는 각 서비스들을 프로비저닝 할 수 있는 템플릿을 제공
- CI/CD, Databases, Languages, Middleware, 기타 항목의 총 79 개의 템플릿을 제공

CI/CD	Databases	Languages	Middleware	Etc..
jenkins	PostgreSQL MariaDB MySQL	Java Ruby PHP Python Node.js Dancer	JWS HTTPD A-MQ 3-scale Wildfly BPM BRMS Keycloak JDG Fuse Data Virtualization	Nginx Redis

OpenShift OPERATOR HUB

- 설치과정이 복잡한 시스템 소프트웨어 를 자동으로 설치, 관리
- AI/Machine Learning, Integration & Delivery, Logging & Tracing, Networking 등의 18 가지 항목의 426 가지 Operator 를 제공

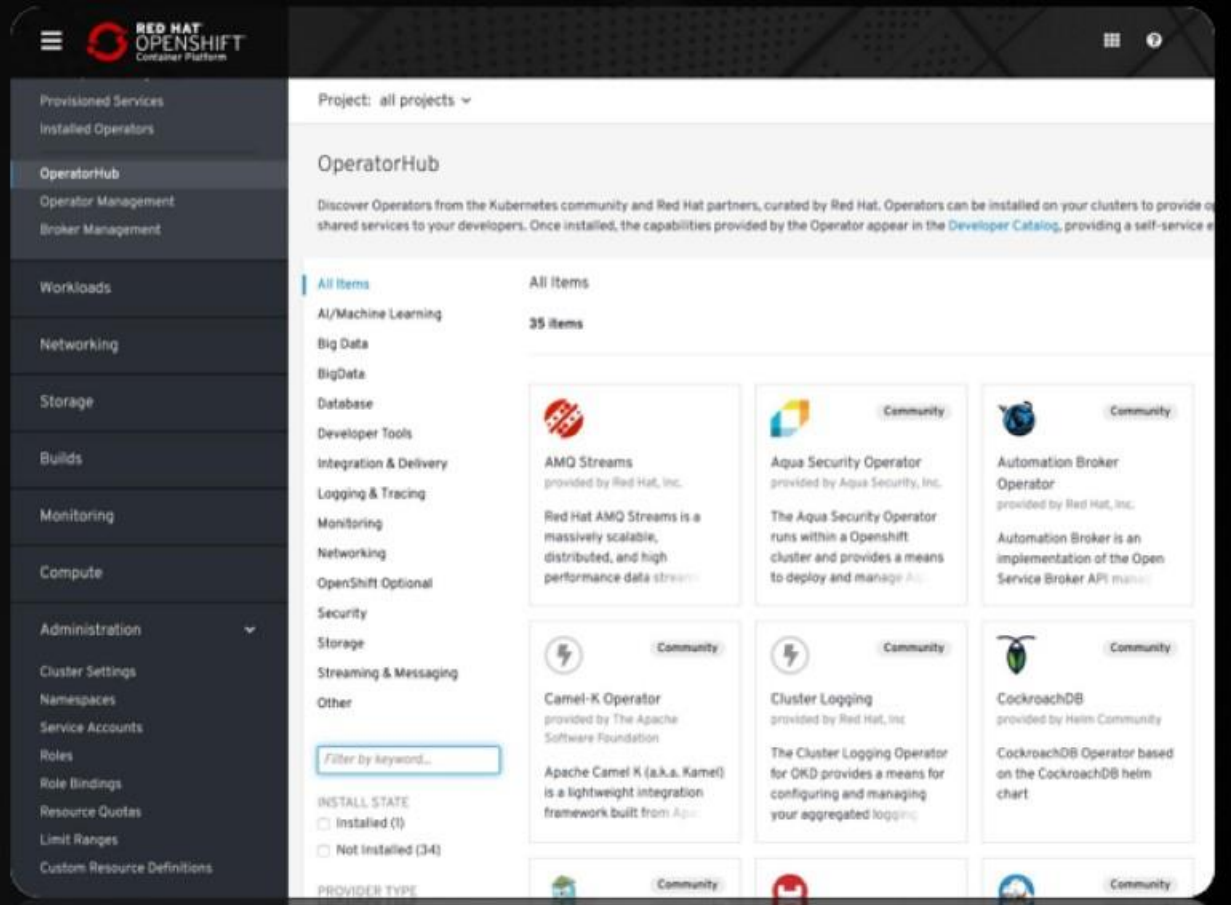
etcd Prometheus kafka dynatrace

Couchbase TensorFlow MySQL redis STRIMZI

mongoDB. confluent JAEGER

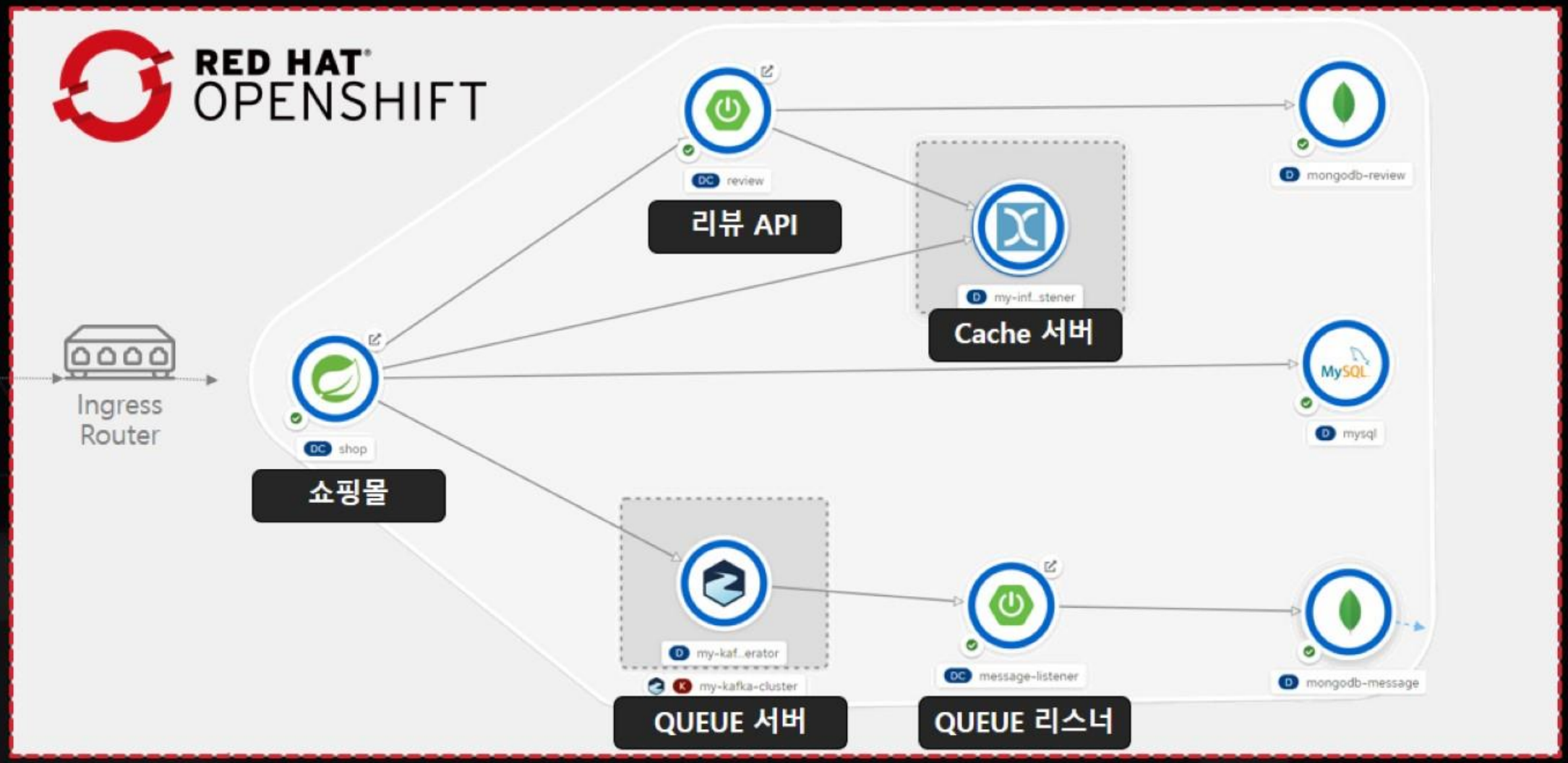
cassandra CRUNCHY Enterprise PostgreSQL Spark

...and many more



OpenShift 를 통한 클라우드 네이티브 애플리케이션으로 마이그레이션

openmaru



OPENMARU Cloud APM

- AWS Cloud 에서 SaaS 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

OPENSHIFT

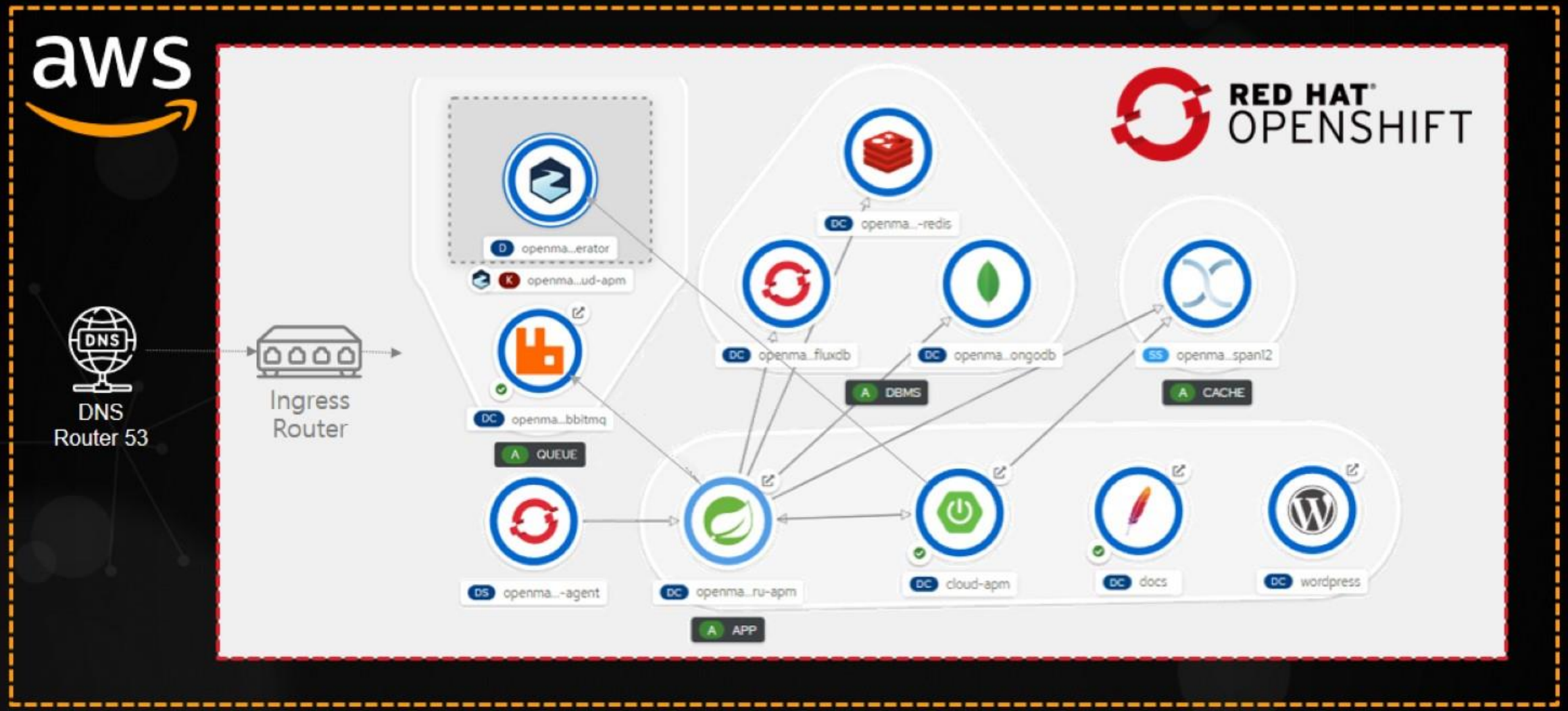
aws

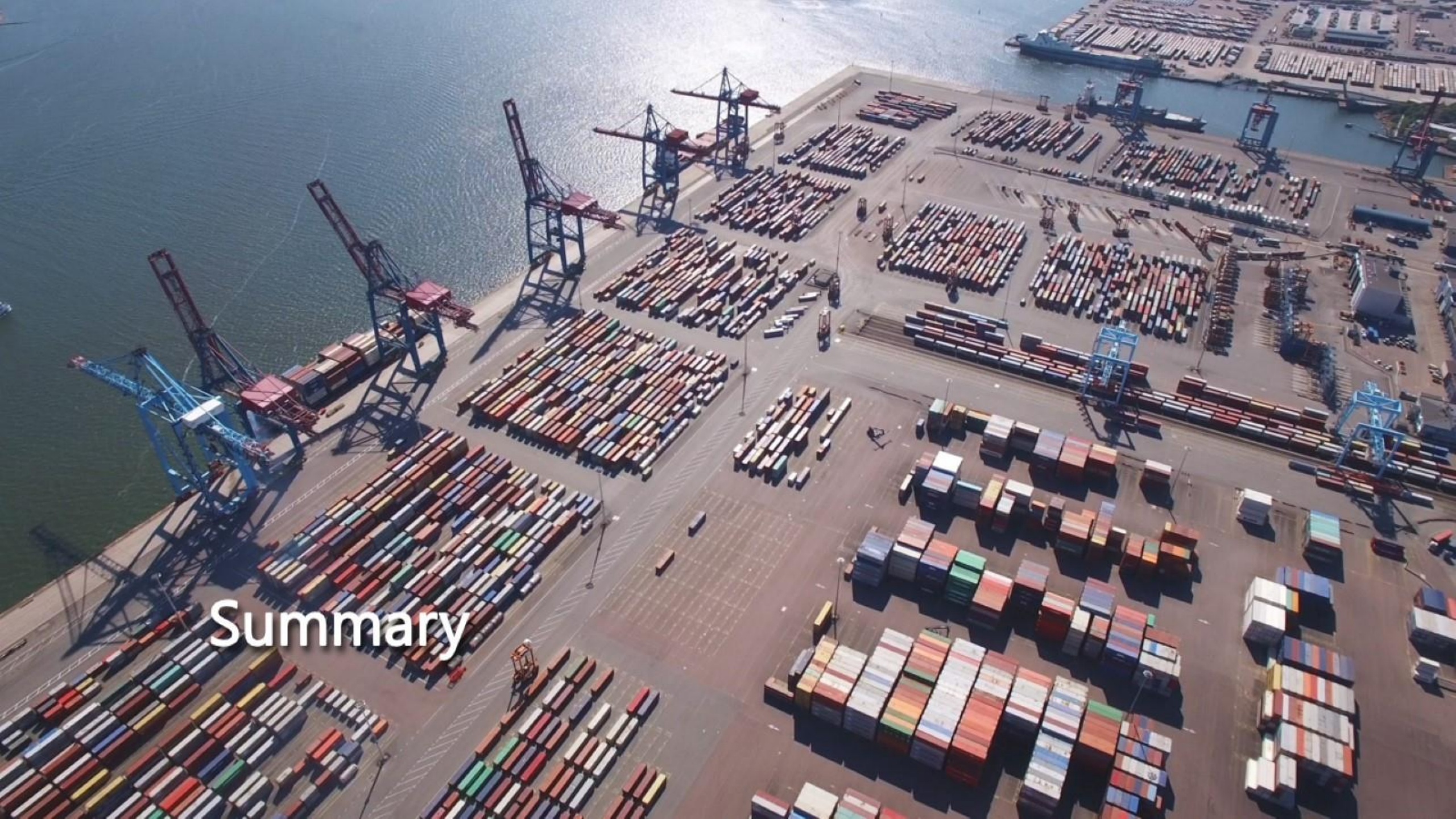
홈페이지
https://www.openmaru.io

클라우드 콘솔
https://console.openmaru.io

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

클라우드 네이티브를 활용한 OPENMARU CLOUD APM (SaaS)





Summary

새로운 비즈니스에 대응하기 위한 신속한 개발과 변화 요구

openmaru

개발 프로젝트 관리 방법

- 스케쥴 관리, 태스크 관리 방법이 단기 개발에 최적화 되지 않음
- 계획 변경에 유연하게 대응 하지 못함

→ Agile

개발팀과 운영팀의 협업

- 개발팀은 짧고 신속한 배포를 원하지만 운영팀은 서비스 중단과 인력 소모로 기피함
- 정보 공유 방법이 효율적이지 않고 협력이 부족
- 테스트에서 출시에 이르는 작업이 많아 운영자가 모두 대응하지 어려움

→ DevOps

개발과 운영 환경 구축의 리드 타임

- 처음부터 환경 구축하는 것은 리드 타임이 길어 짧은 프로젝트에 적용하기 어려움
- 요구 사항이 자주 변경되는 상황에서 유연하게 환경을 변경 할 수 있어야 함

→ Automation

클라우드 아키텍처

- MSA 기반으로 작은 단위의 아키텍처와 새로운 적용 기술 들에 대한 개발 검증 환경 제공
- 대규모 애플리케이션에서 변경 사항에 대한 영향도 분석과 테스트 시간이 오래 소요 됨

→ Microservices


openmaru