쿠버네티스 구축 시 개발자와 운영자가 당황하는 핵심 이슈들과 해결 방법

openmaru APM

목차

1. 기업들은 어떤 이유로 쿠버네티스를 도입하는 걸까요?

2. 컨테이너 기술의 특징으로 인하여 인프라는 어떻게 변경될까요?

3. 서비스 중지 없이 배포와 시스템 관리를 할 수 있을까요?

4. 컨테이너 환경에서 보안은 어떻게 바뀌었을까요?

OPENMARU © 2021 | All Rights Reserved.

Platform As A Service

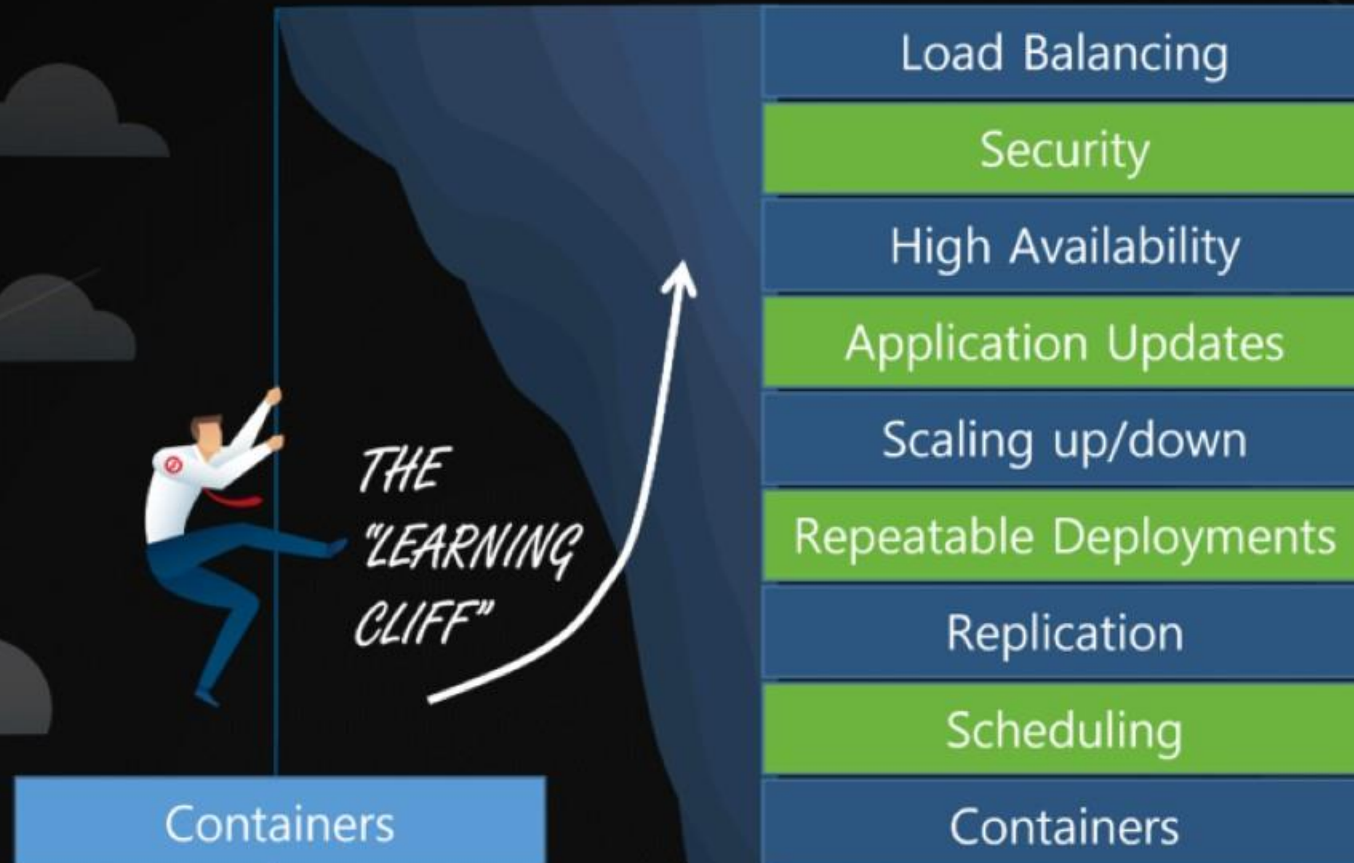
기업들은 어떤 이유로 쿠버네티스를 도입하는 걸까요?

Containers and Kubernetes: The Time Is Now

openmaru

CONTAINERS IN DEVELOPMENT

CONTAINERS IN PRODUCTION

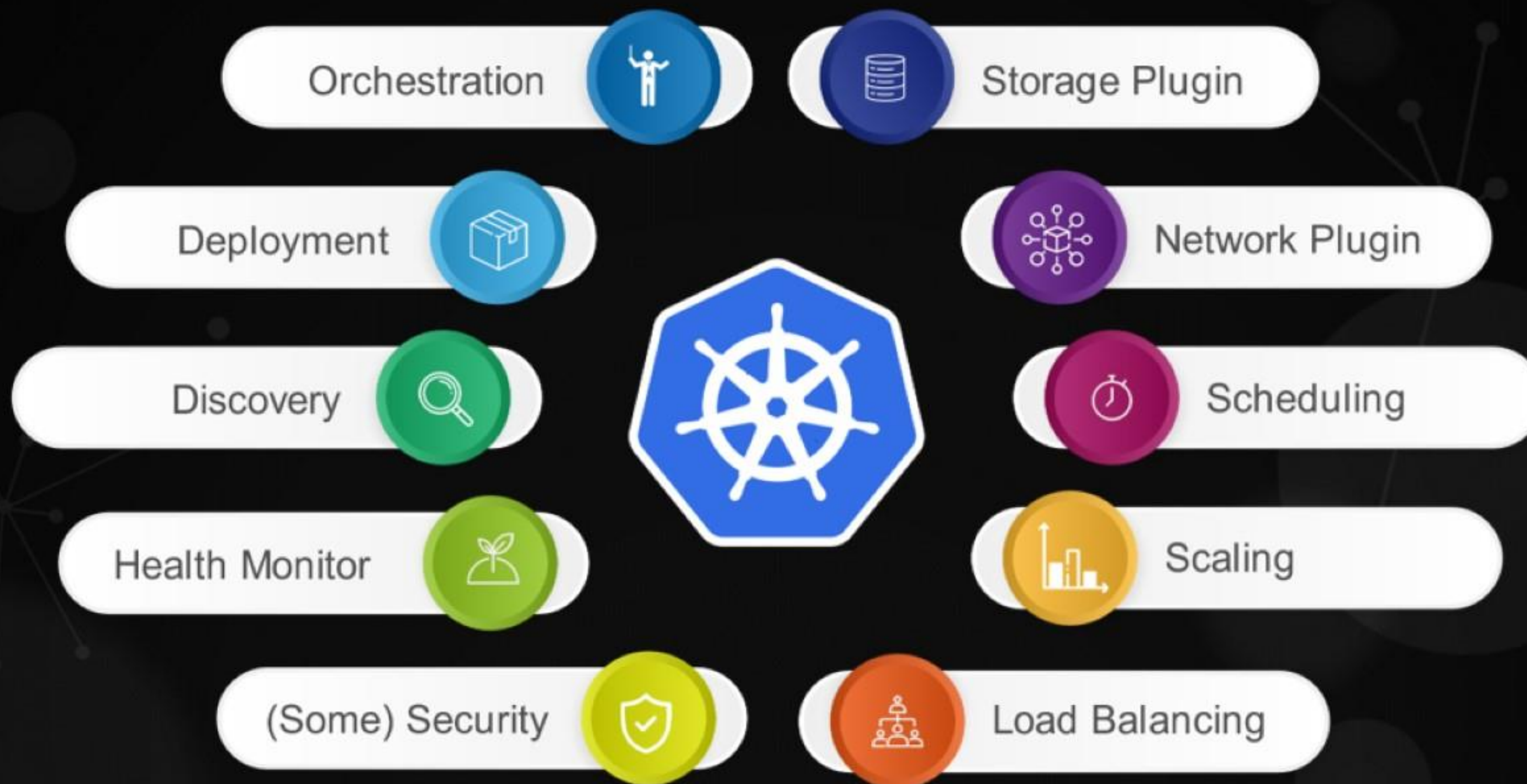


Source : https://blogs.vmware.com/cloudnative/2018/01/23/containers-kubernetes-benefits/

All Rights Reserved.

KUBERNETES DOES A LOT FOR YOU

openmaru



OPENMARU © 2021 | All Rights Reserved.

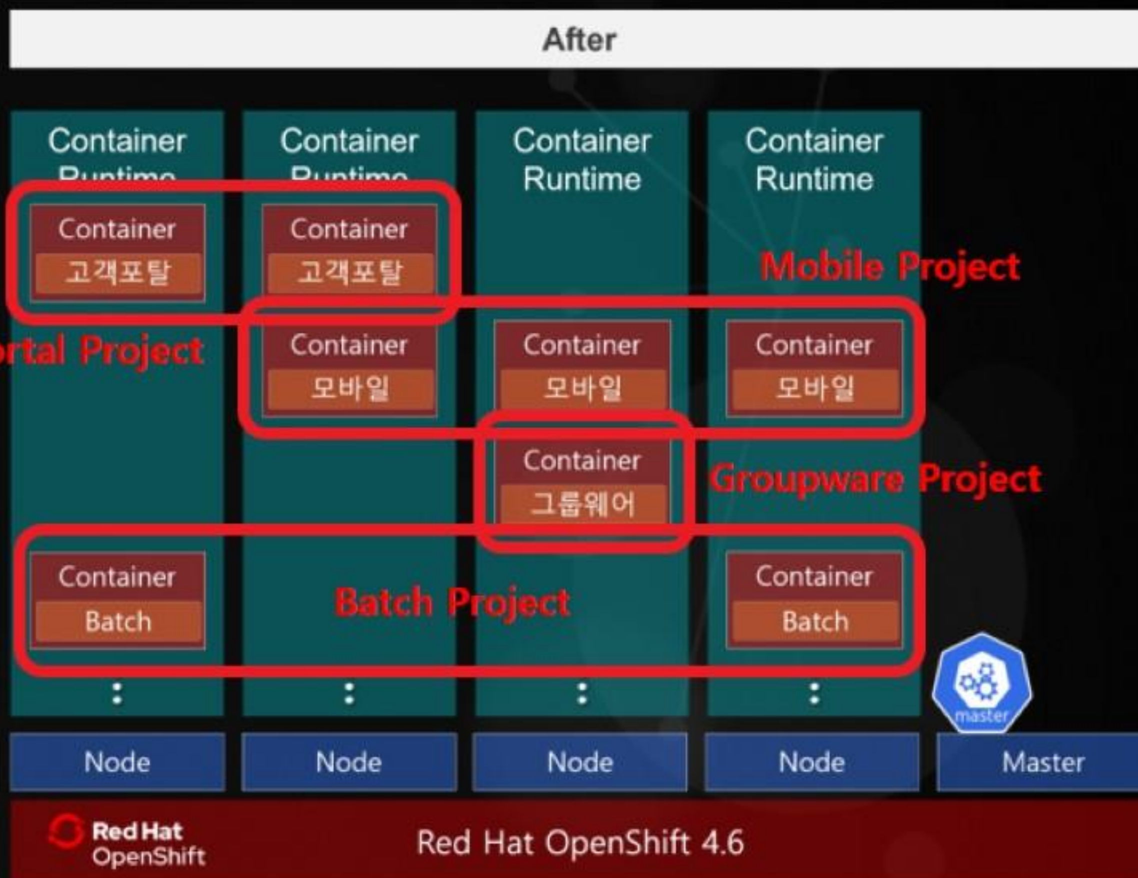
Kubernetes 의한 컨테이너 오케스트레이션의 실현

openmaru

Before

- 용도 당 VM 불필요한 자원 소비
- 환경 복제가 어려움

After



- Kubernetes이 Node의 자원 상황을 보고 적절한 컨테이너를 배치
- 손쉬운 개발환경 구축 과 관리자의 개입을 최소화 하여 자동 확장/복구 실현

OPENMARU © 2021 | All Rights Reserved.

소프트웨어 시대와 OpenShift

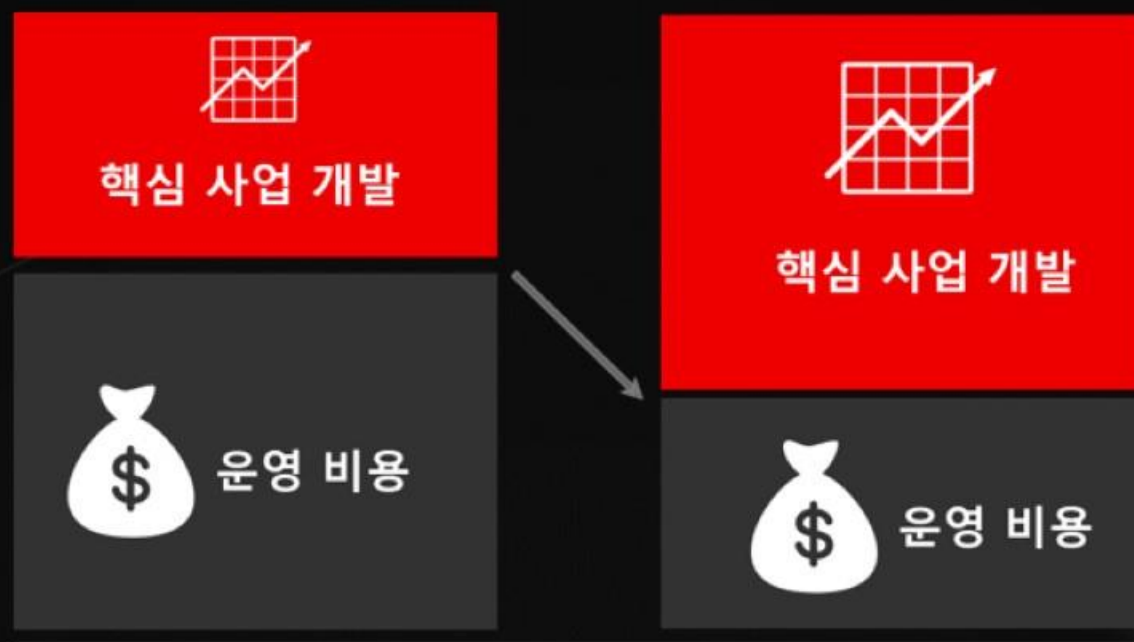
- 소프트웨어 개발의 혁신
  - 적은 인원으로 대규모 개발이 가능해짐
    - 오픈소스 소프트웨어 활용
    - 자동화 도구 발전
    - 클라우드 컴퓨팅 등장
  - DevOps와 Agile개발의 등장
    - 빠른 구현에서 배포에 이르는 사이클을 안정적 컨트롤 하는 기술과 프로세스가 등장
- 기업에 있어서 IT 위상변화
  - 기업 혁신을 위한 핵심 역량으로 IT 가 대두
    - AI/ML , 자동화,
  - 산업의 고도화로 인해 점차 핵심분야로 이동
  - 핵심비즈니스 영역에 포함되거나 아예 핵심 비즈니스 자체가 됨

OPENMARU © 2021 | All Rights Reserved.

PaaS 를 통한 사업 비용 절감

openmaru

- 매니지드 서비스로 운영 비용을 줄이고 귀중한 인적 자원을 핵심 사업의 개발에 집중



OPENMARU © 2021 | All Rights Reserved.

PaaS Cloud 혜택

- OpenShift 을 적용하여 민첩성 높은 서비스를 제공

작업 공수 절감

기존 수동으로 해왔 던 작업을 자동화하여 작업 공정 및 납기 단축

운영 품질 향상

관리자의 개입을 최소로 자동화하여 작업 품질을 균일하게 유지

시스템 운영의 표준화 촉진

- 자동화 및 버전 관리 함으로써 시스템 운영 정책 및 업무 표준화

작업 통제 강화

작업 작업을 자동화함으로써 내부 통제 및 보안 측면에서의 효과를 기대

- Confidential -

OPENMARU © 2021 | All Rights Reserved.

시스템 비대화로 작업 폭증과 인력 부족 어떻게 할까요?

openmaru

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

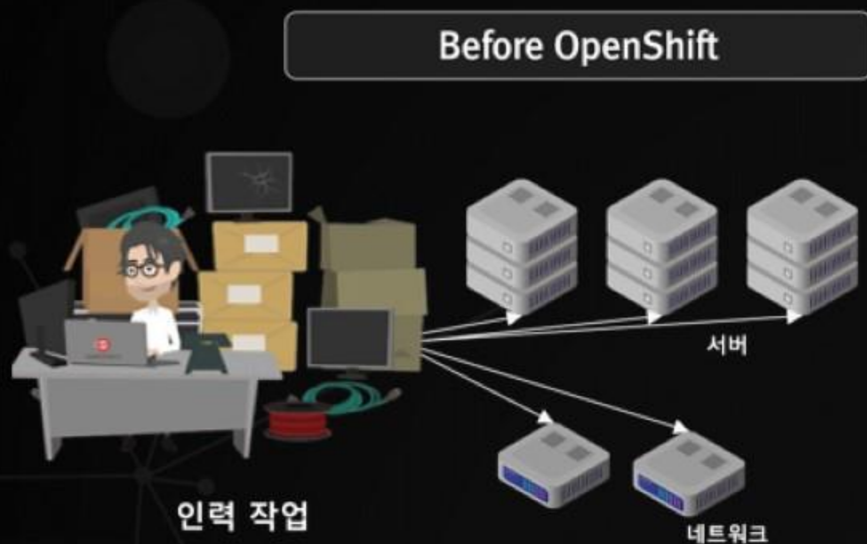
OPENMARU © 2021 | All Rights Reserved.

OpenShift 을 통한 IT 인프라 운영 자동화

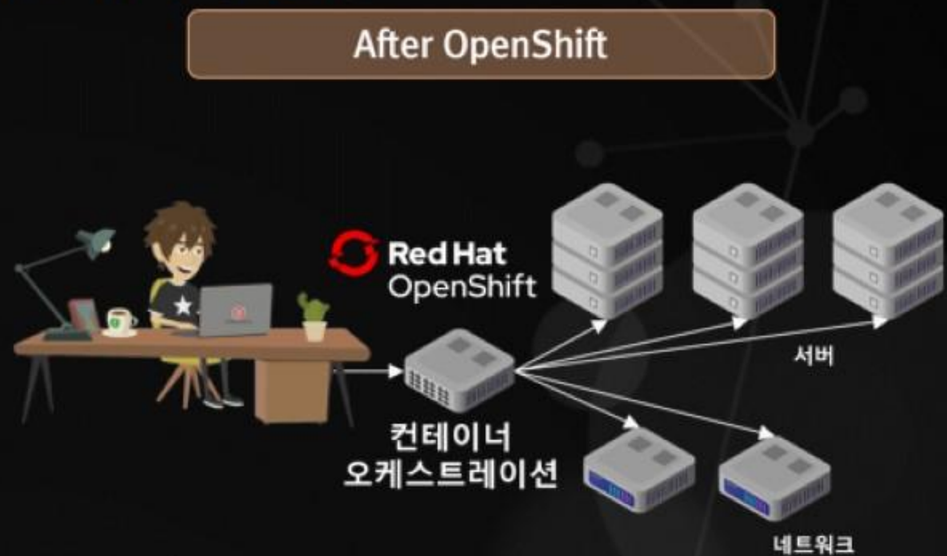
openmaru

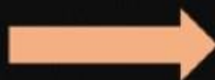
- IT 인프라의 대규모화, 고도화에 따라 IT 장비에 대한 환경설정 및 정보 취합이 복잡하고 어려움
- 작업 계획시간과 현장 작업 시간의 증가와 휴먼 에러의 증가

Before OpenShift



After OpenShift



Before OpenShift		After OpenShift
시스템 운영을 위한 관리 작업 증가		운영 기술 표준화를 통한 준비 시간 및 작업 시간 감소
현장 작업 시간 증가		시스템 일괄 설정 작업 시간 단축
Human Error 증가		시스템을 통한 작업으로 Human Error감소

OPENMARU © 2021 | All Rights Reserved.

Platform As A Service

컨테이너 기술의 특징으로 인하여 인프라는 어떻게 변경될까요?

Case 1. 고정적인 IP

WEB/WAS의
IP가 뭔가요?

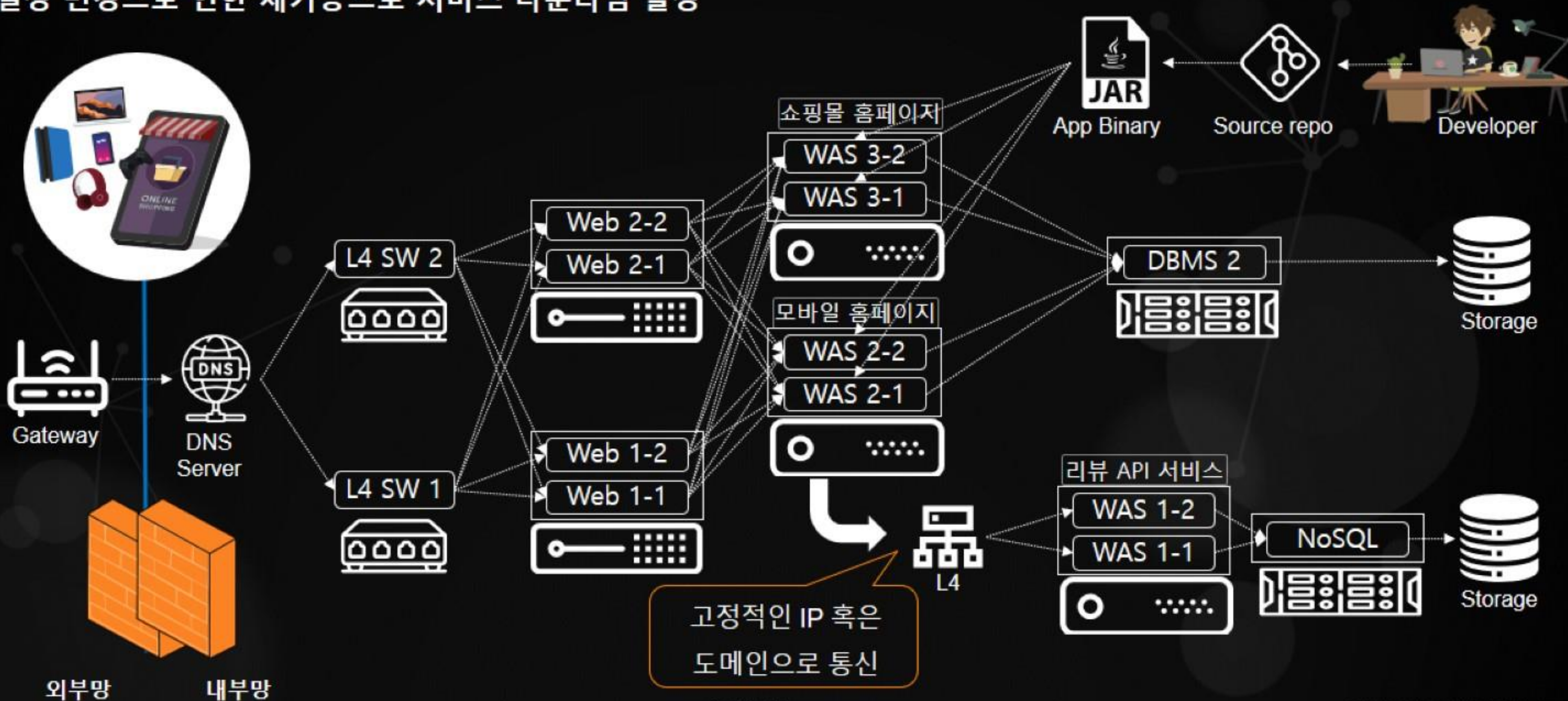
다른 애플리케이션에
어떻게 통신해야하나요?

클라우드 환경인데
IP 가 있는 게 말이
되나요?

OPENMARU © 2021 | All Rights Reserved.

AS-IS(레거시) 환경의 고정IP를 이용한 애플리케이션간 통신

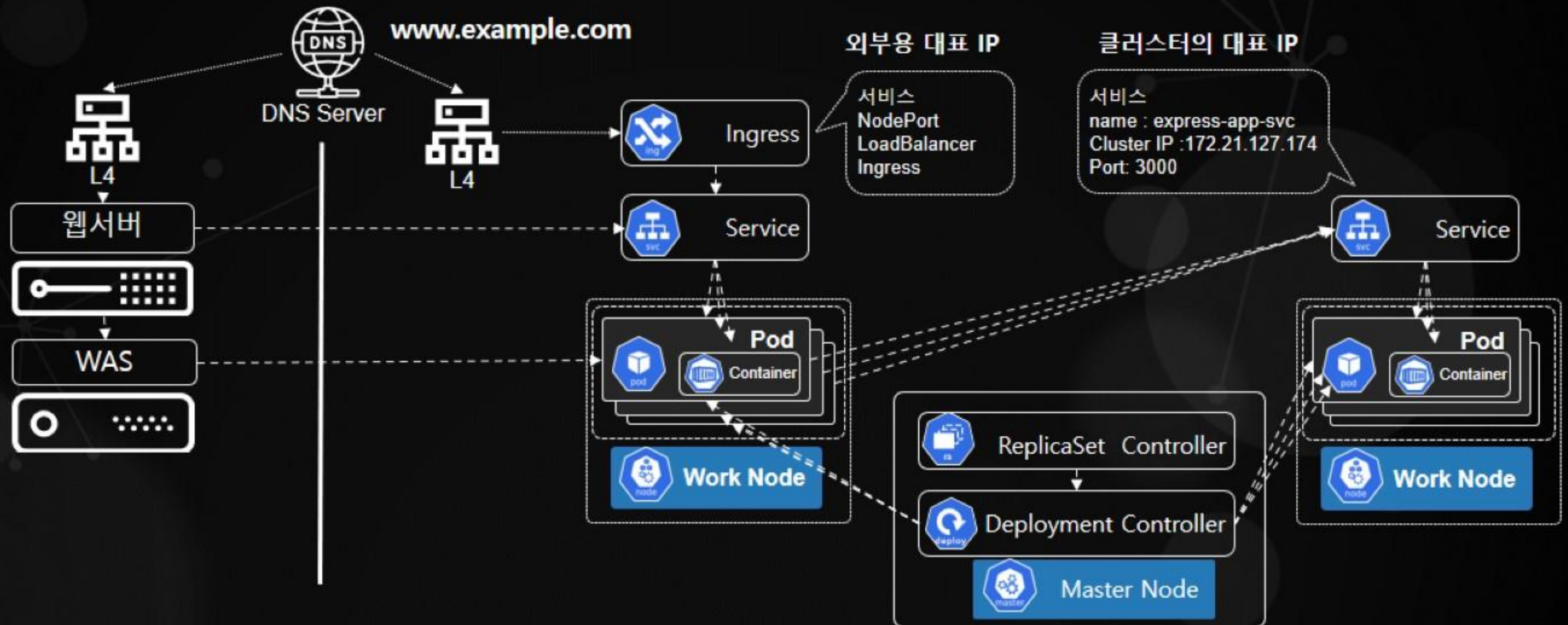
- 서비스 확장 시 모든 설정을 수작업
- 설정 변경으로 인한 재기동으로 서비스 다운타임 발생



OPENMARU © 2021 | All Rights Reserved.

컨테이너 간의 호출은 service를 이용

- 컨테이너 환경의 내부 Load Balancer(Domain 기반) 통신
- Service Object의 도메인 기반 으로 호출
- Pod는 유동적인 IP Pool을 갖으며 고정적인 IP를 가지지 않음



OPENMARU © 2021 | All Rights Reserved.

고정 IP 환경 (AS-IS) VS. 동적 IP 환경 (To-Be)

openmaru

	고정 IP 환경	동적 IP 환경
구조	• 최초 구성부터 고정된 IP 를 할당 • **IP기반 호출방식**	• 지정 IP 가 없으며, 장애/ 배포/ 재시작 등 상태에 따라 IP 변경 • **Service 호출방식**
개발자	• 코드에 IP 정보 하드코딩	• **Service Domain 기준으로 서비스 디스커버리**
운영자	• 서버별로 수동 배포 • **서버 환경 변경(추가삭제)시 수작업 환경설정**	• CI/CD 자동화 • Auto scaling • Auto healing
기획자	• 고가의 보안소프트웨어와 비용 발생	• 비즈니스 유연한 확장성 • 클라우드 대응력 향상 • 소프트웨어 비용 절감

- Confidential -

OPENMARU © 2021 | All Rights Reserved.

Case 2. 서버의 설정 파일

openmaru

WAS, 애플리케이션
설정을 바꾸려면
무엇을 바꾸나요?

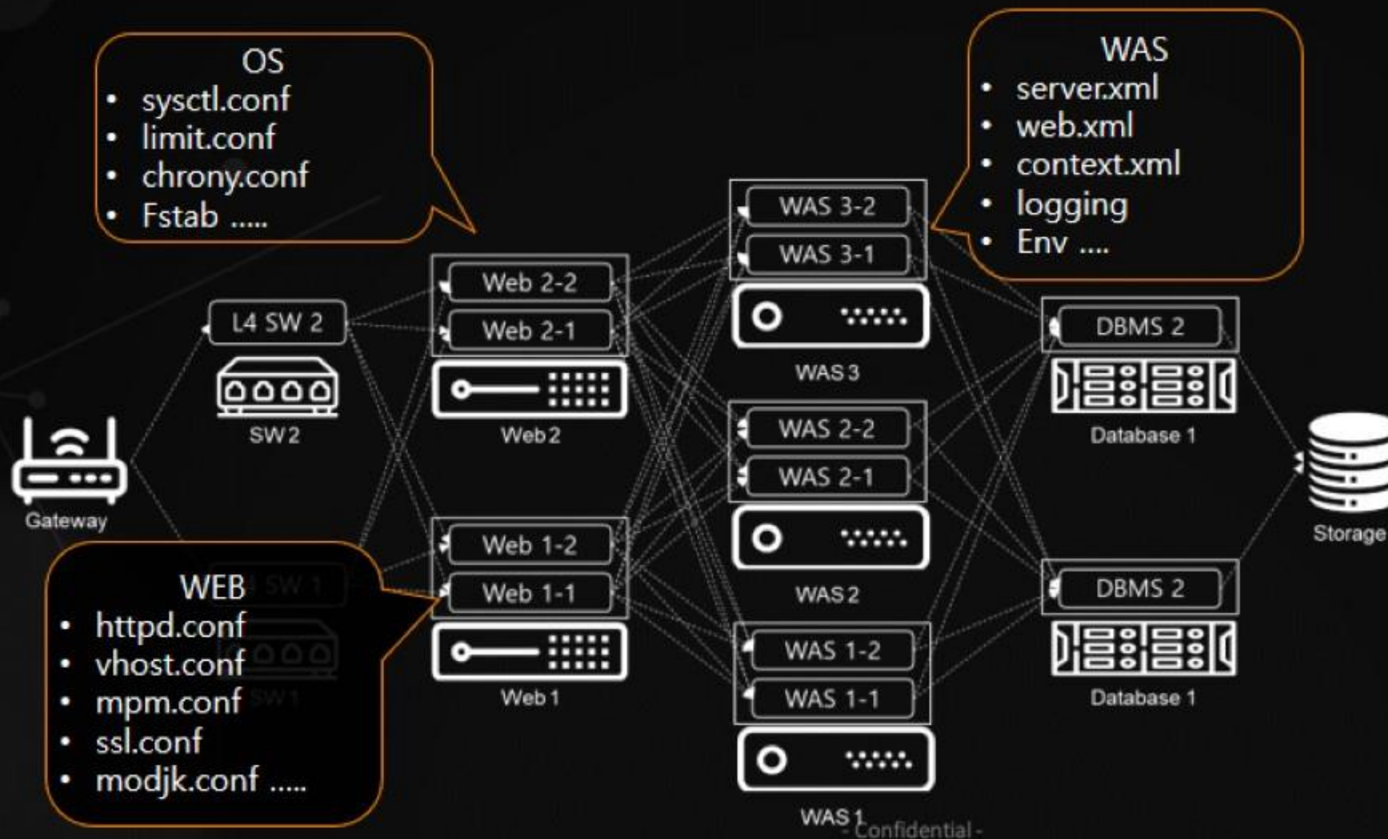
컨테이너를 직접
수정해도 되나요?

OS 설정은
어떻게 바꾸나요?

OPENMARU © 2021 | All Rights Reserved.

AS-IS : 개별 서버 별로 환경 설정파일 관리

- OS/WEB/WAS 설정을 파일로 관리
  - Configuration Drift 발생
  - WEB/WAS의 구성이 많을수록 관리포인트 증가
- 설정이 변경된 후 WEB/WAS 재기동 다운타임 / 복잡한 무중단 구현

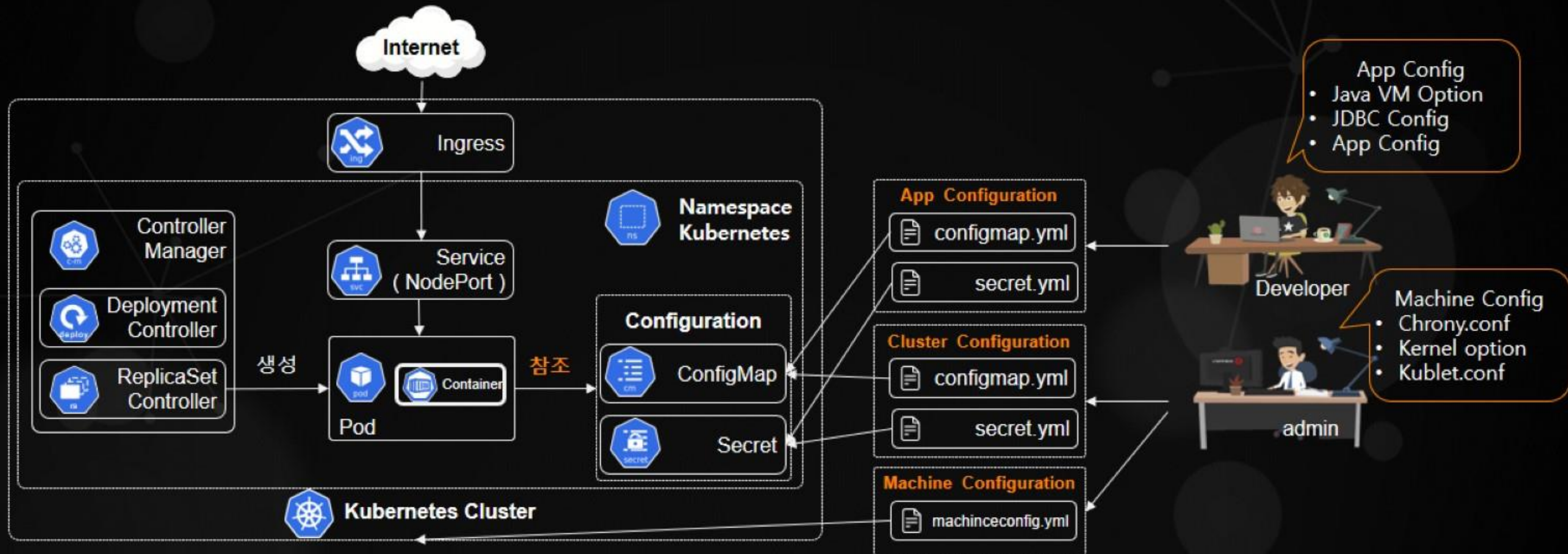


소규모 WEB/WAS구성임에도 관리포인트가 도대체 몇 개죠?

OPENMARU © 2021 | All Rights Reserved.

To-Be 환경 설정 통합 관리 (Configuration As A Service)

- 환경 의존 매개 변수를 외부화
- Configmap, Secret을 이용하여 WAS/APP 설정 통합 관리
- Machine Config를 이용하여 서버 설정 통합 관리



OPENMARU © 2021 | All Rights Reserved.

서버 별 환경설정 vs. 환경 설정 통합 관리(Configuration As A Service)

openmaru

	개별 서버 별 설정 관리	클라우드 네이티브 환경
구조	• **개별 서버별로 설정파일 관리**	• **머신 컨피그 (Machineconfig) 와 Configmap 활용**

개발자	• 개발/스테이징/운영 환경 설정이 달라짐
	• 개발과 장애 발생 시 환경 설정이 어려움
	• **표준 개발환경 구성이 어려움**
	  ▪ 구인과 인수/인계가 어려움
	  ▪ 개발 환경 교육 시간과 적응 기간이 오래 걸림
		• 개발/스테이징/운영 환경이 동일함
		• 애플리케이션 버전/개발환경 별로 필요한 환경 구성
		• 표준개발환경으로 신규 인력의 적응 기간 단축

운영자	• Configuration drift • **서버가 많으면 많을 수록 작업량 증가** • 서버 별 설정 파일 관리	• **OS는 Machineconfig 기반의 환경 통합 관리** • **Application은 secret, Configmap 기반의 환경 통합 관리** • Git ops로 확장 가능
기획자	• 고가의 보안소프트웨어와 비용 발생	• 비즈니스 유연한 확장성 • 클라우드 대응력 향상 • 소프트웨어 비용 절감

OPENMARU © 2021 | All Rights Reserved.

Case 3. 애플리케이션 로그

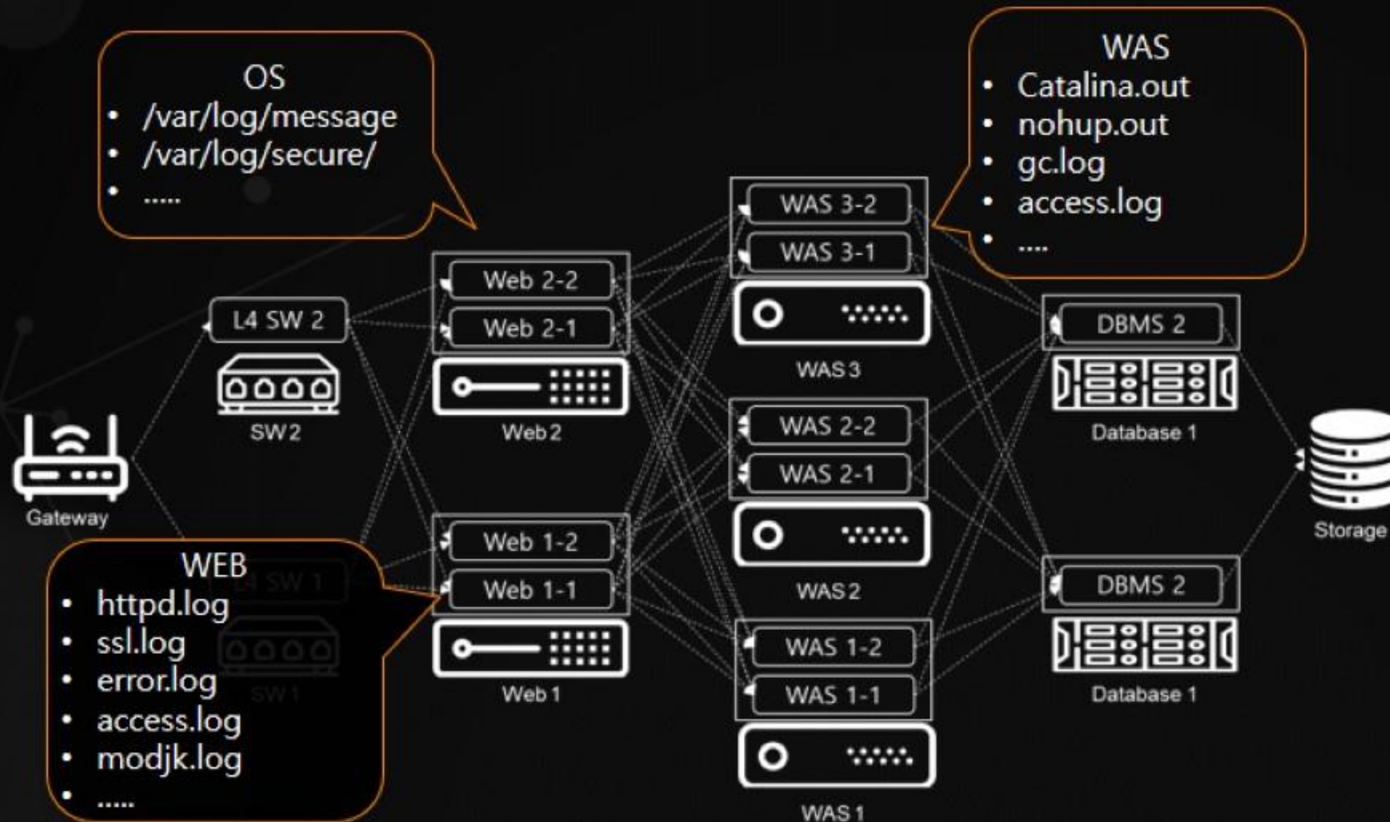
로그를 확인하고 싶은데,
어디서 확인하는거죠?

로그를 컨테이너의
어디에 쌓는건가요?

OPENMARU © 2021 | All Rights Reserved.

AS-IS – 개별 서버 별로 로그 파일 관리

- 발생하는 로그들이 각각의 서버 Local Disk에 저장됨
- 에러 발생시 에러가 발생한 서버를 찾아야 하고 발생한 에러로그를 분석
- 서버 대수가 늘어날 수록 모니터링 해야하는 Log의 수도 늘어남

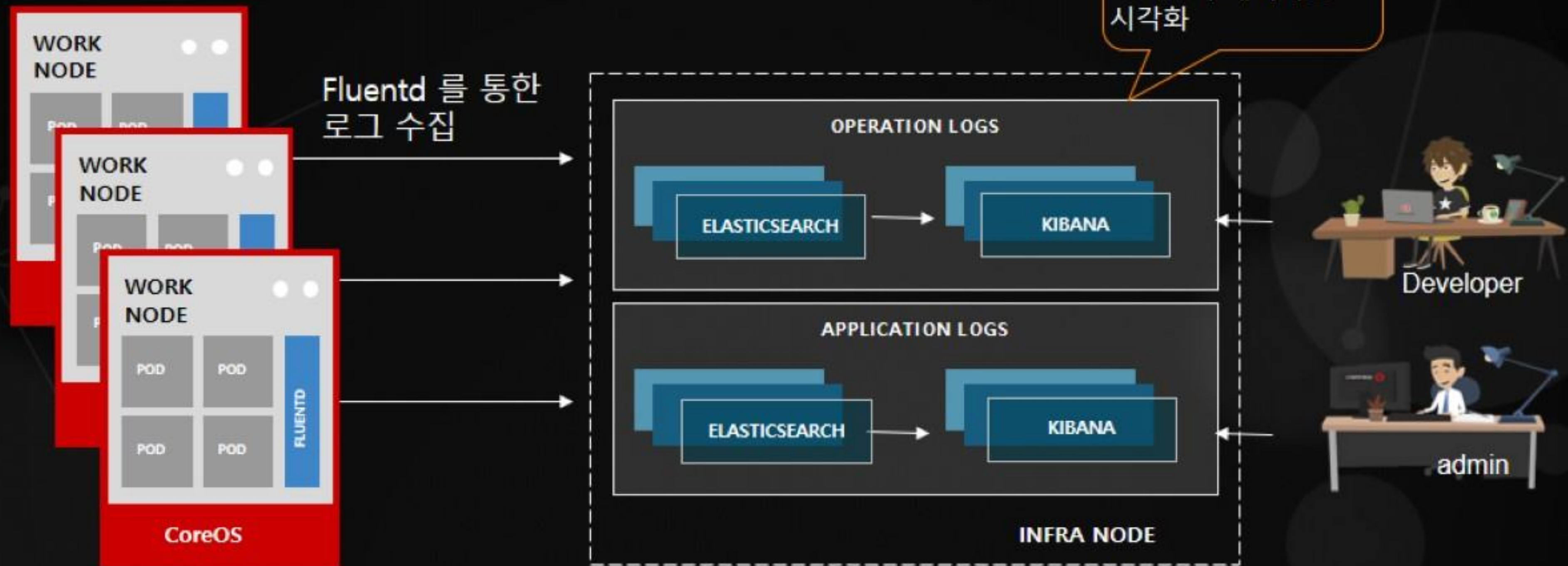


소규모 WEB/WAS구성임에도
관리포인트가 도대체 몇 개죠?

OPENMARU © 2021 | All Rights Reserved.

To-Be : 통합 로그 관리

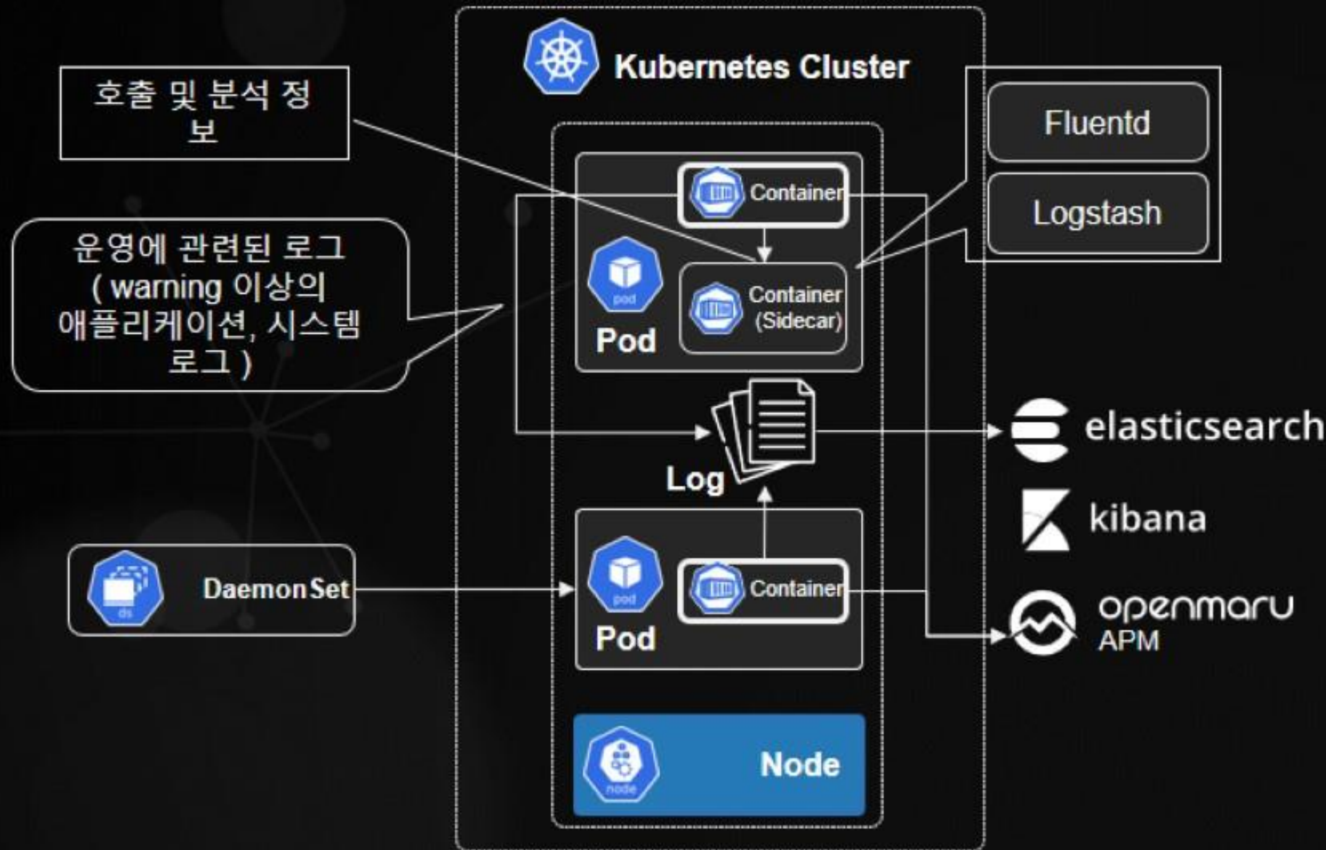
- PaaS 플랫폼의 경우 약 200개 이상의 플랫폼 인프라 컨테이너 기동
- 애플리케이션 컨테이너도 기존 환경에 비해 몇배 많은 수가 기동
- 수많은 컨테이너의 로그를 통합 모니터링하는 스택이 필수
- EF(L)K (Elastic Search + Fluentd(Logstash) + Kibana) 스택

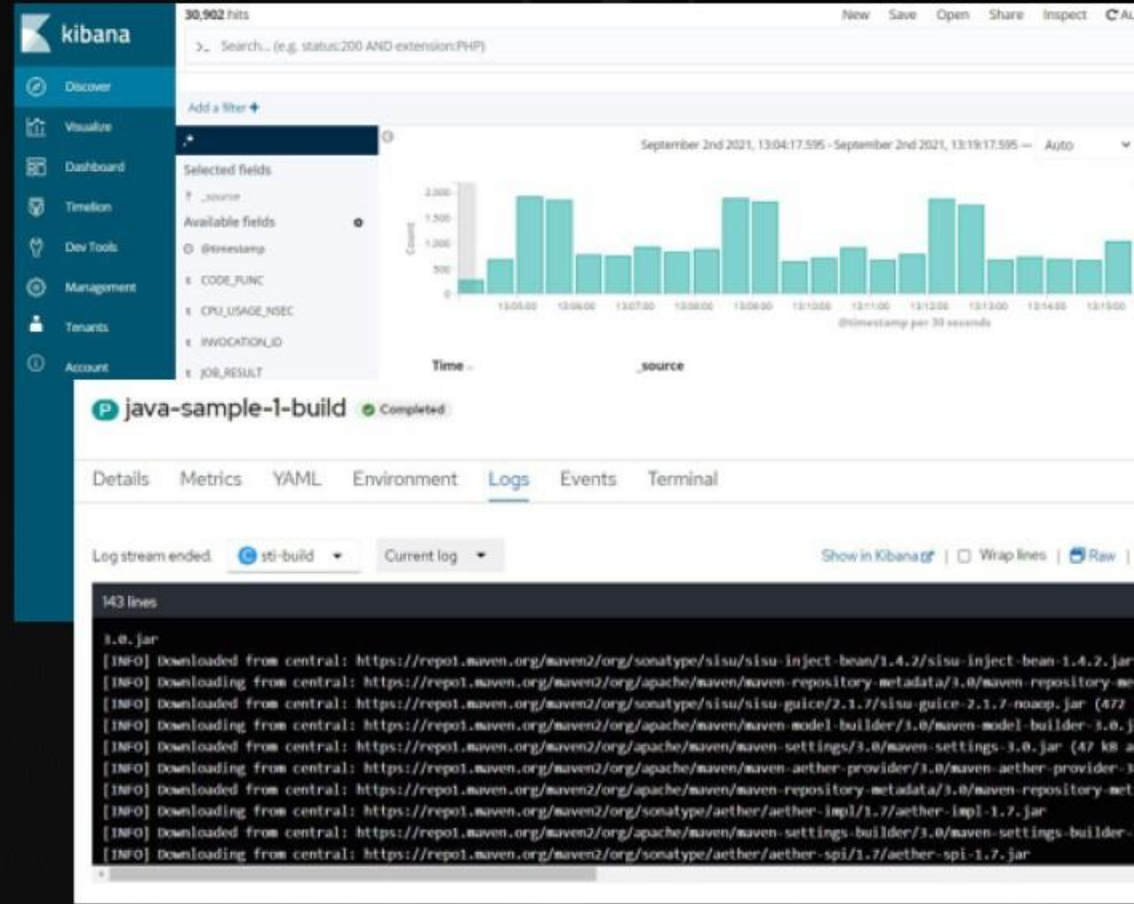


OPENMARU © 2021 | All Rights Reserved.

To-Be : 통합 애플리케이션 로그 관리 방안

- 기존 레거시 환경에서는 개별 서버에 로그 생성 후 별도 작업으로 통합 관리
- EFK (ElasticSearch, Fluentd, Kibana) 를 활용한 자동 로그 통합 관리
  - 애플리케이션의 로그 정책을 STDOUT으로 설정





OPENMARU © 2021 | All Rights Reserved.

개별 서버 로그 관리 vs. 통합로그 관리 (Observability)

openmaru

	개별 서버 로그관리	통합로그관리
구조	• 개별 서버에 로그가 쌓임	• EFK 스택을 통해 로그 자동 수집 및 통합 , 검색
개발자	• **서버별로 장애가 난 로그를 확인해야함** • 로그 파일 검색	• **Kibana를 통해서 로그 웹콘솔로 확인(서버접속불가)**
운영자	• 로그 로테이션 설정 • **로그로 인한 서버 접속 보안 관리**	• 시스템 운영상에 대한 메트릭 커스텀으로 생성 후 모니터링 • 애플리케이션, 서버로그도 EFK에 저장
기획자	• 로그통합 프로젝트 필요 • 프로젝트 비용 발생	• 비즈니스 유연한 확장성 • 클라우드 대응력 향상 • 소프트웨어 비용 절감

OPENMARU © 2021 | All Rights Reserved.

Case 4. 애플리케이션 모니터링

openmaru

컨테이너 환경이라고
애플리케이션 모니터링이
다른가요?

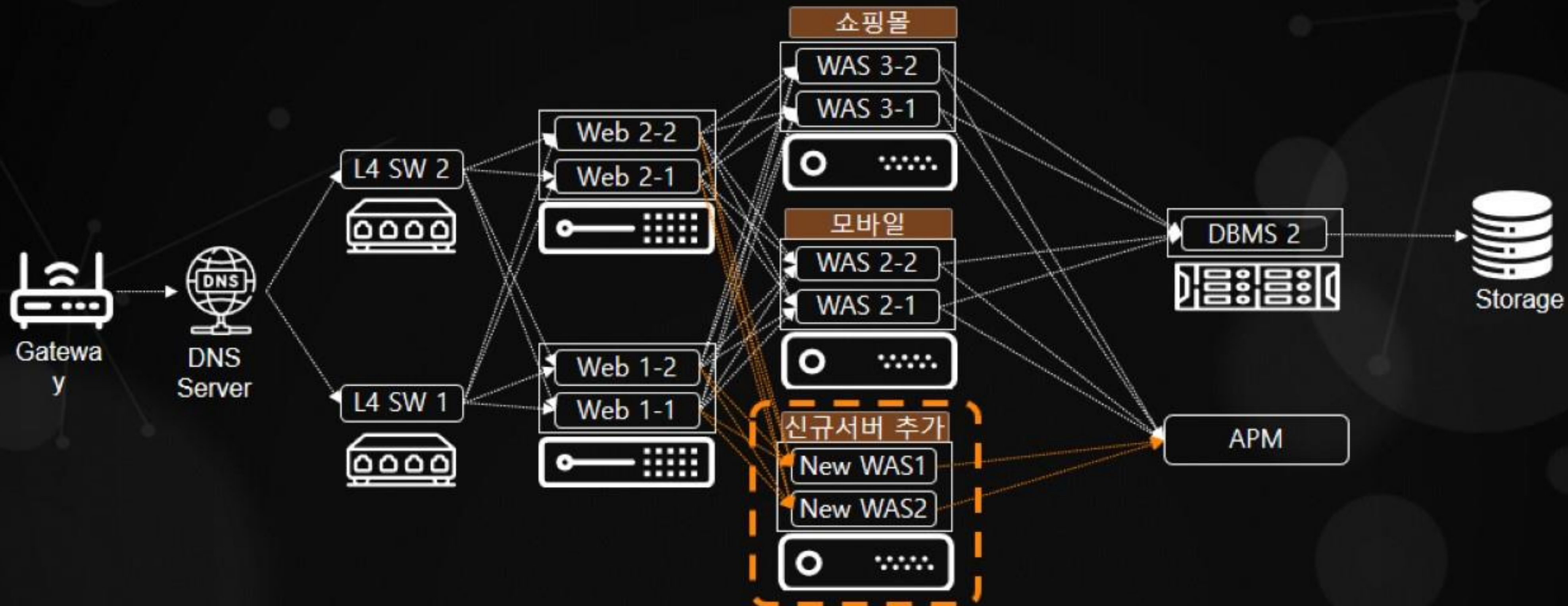
다르다면 무엇이
다른거죠?

OPENMARU © 2021 | All Rights Reserved.

AS-IS : 레거시에서 모니터링

openmaru

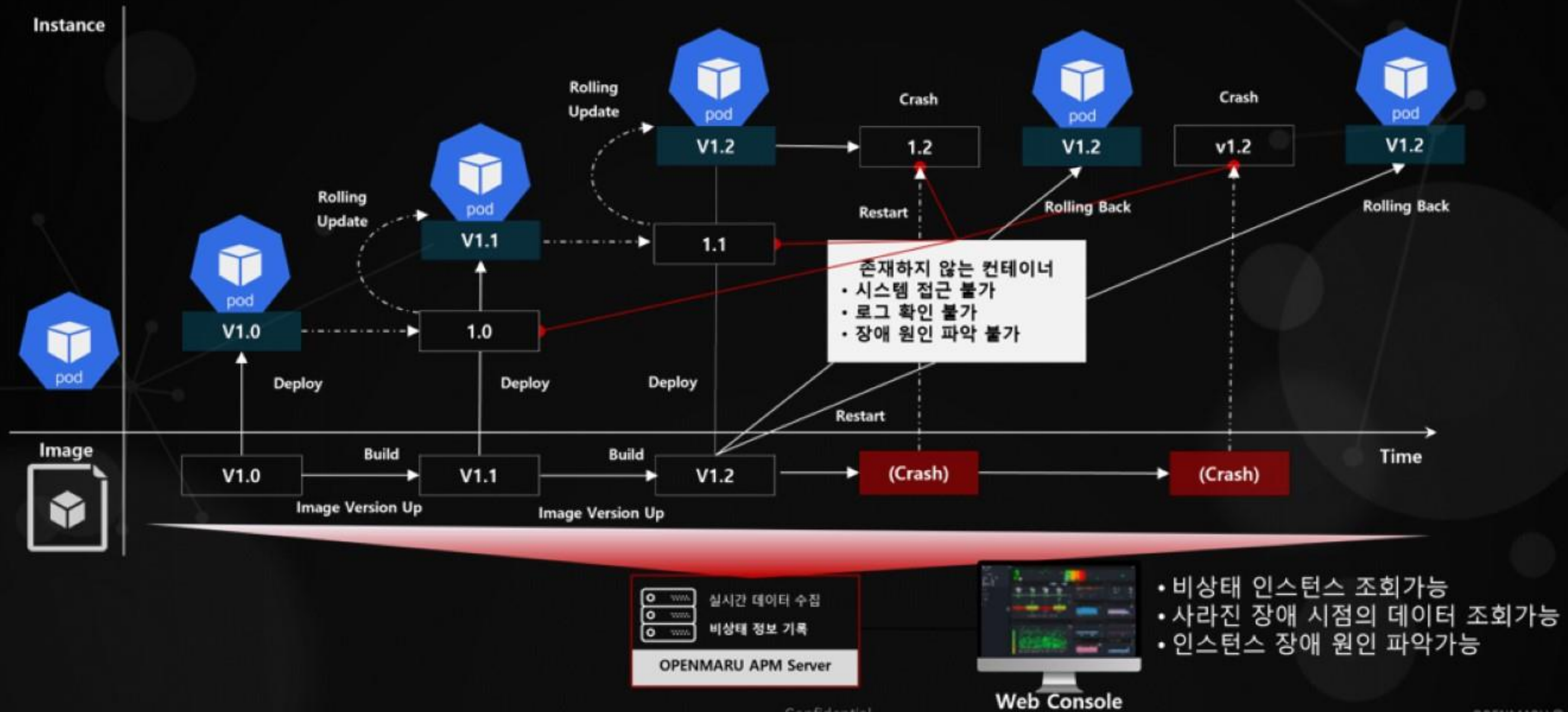
- 기존 환경은 WAS별 Instance가 고정적인 상태정보 (IP, Hostname 등)를 가지고 있음
- WAS 혹은 Instance들이 유연하게 확장/축소 되지 않는 환경
- 확장/축소가 유연하지 않음에 따라 APM도 그에 맞는 로직이 존재하지 않음
- WAS/Application에 장애가 발생하여도 기존 데이터가 사라지지 않음



OPENMARU © 2021 | All Rights Reserved.

To-Be : 무상태 (Immutable) 인프라스트럭처로 상태를 저장하지 않음

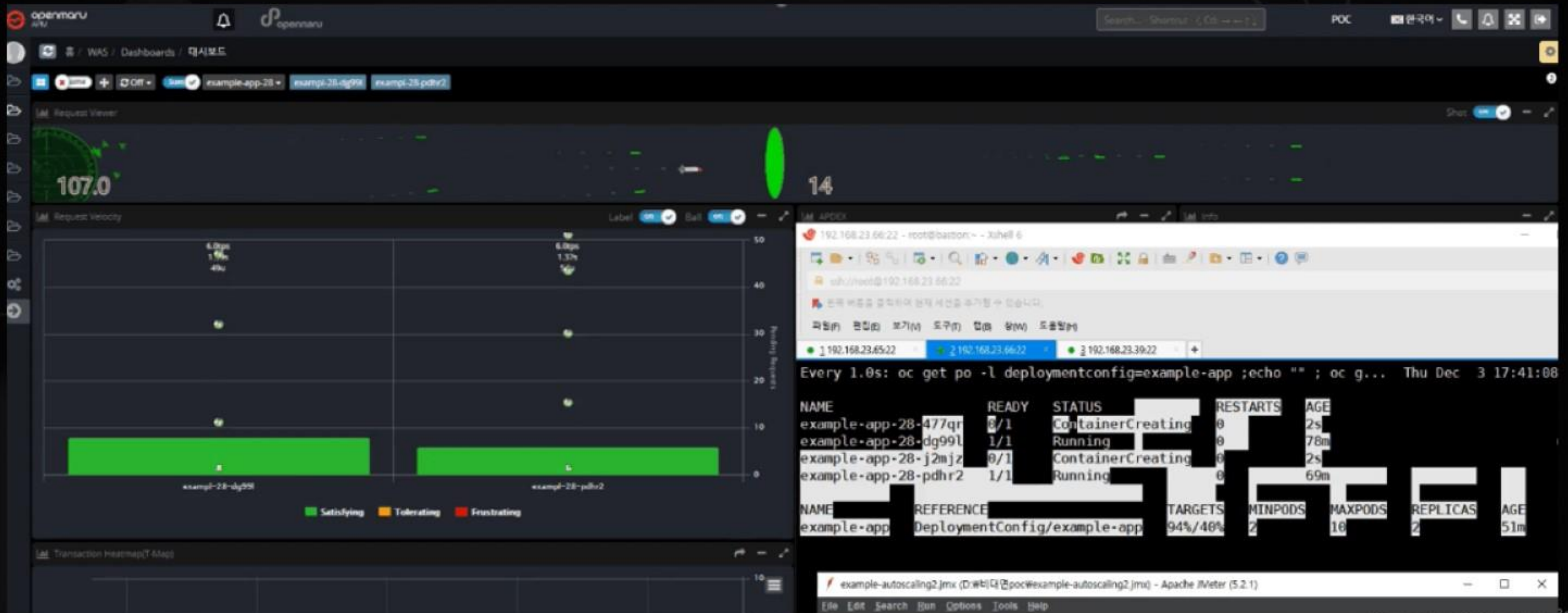
- PaaS 환경에서 컨테이너가 중지된 후 장애원인 파악을 위한 방법을 제공
- APM 에서 사라진 컨테이너에 대한 정보를 보관하여 장애원인을 파악할 수 있음



OPENMARU © 2021 | All Rights Reserved.

To-Be(컨테이너) 환경에서 모니터링 구조

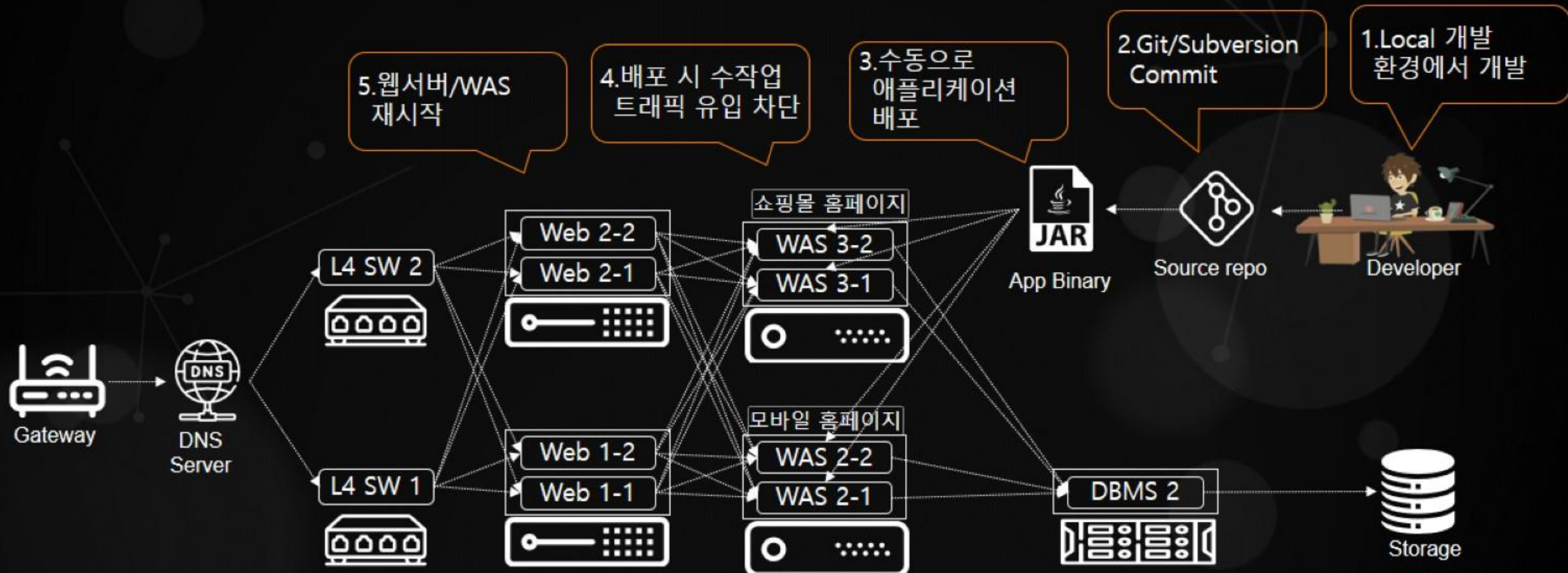
- Auto Scaling으로 Container(Pod)가 유연하게 Scale Out/In이 발생하는 환경
- APM 서버에는 그러한 환경을 뒷받침할 라이선스 정책과 로직이 필요함



서비스 중지 없이 배포와 시스템 관리를 할 수 있을까요?

AS-IS : 기존 레거시에서 빌드/배포

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



OPENMARU © 2021 | All Rights Reserved.

To-Be : 컨테이너 기반 자동 빌드/배포

- 신속한 개발과 효율적인 운영 환경 구축
- 코드 개발 이외에 개발자/관리자가 빌드 배포에 기여하는 수작업은 “빌드 명령” 밖에 없음

Developer
GIT SERVER
ARTIFACT REPOSITORY
Admin

1.Source Code Pull

EXISTING DELIVERY PROCESS

Image Build & Deploy
Promote to TEST
Promote to UAT
V
Promote to PROD

2.Container Image 자동 생성

3.트리거로 인해 자동 배포

OPENSHIFT IMAGE REGISTRY

OPENSHIFT IMAGE REGISTRY

4.자동 트래픽 유입 차단 무중단 배포

OPENSHIFT CLUSTER

OPENSHIFT CLUSTER

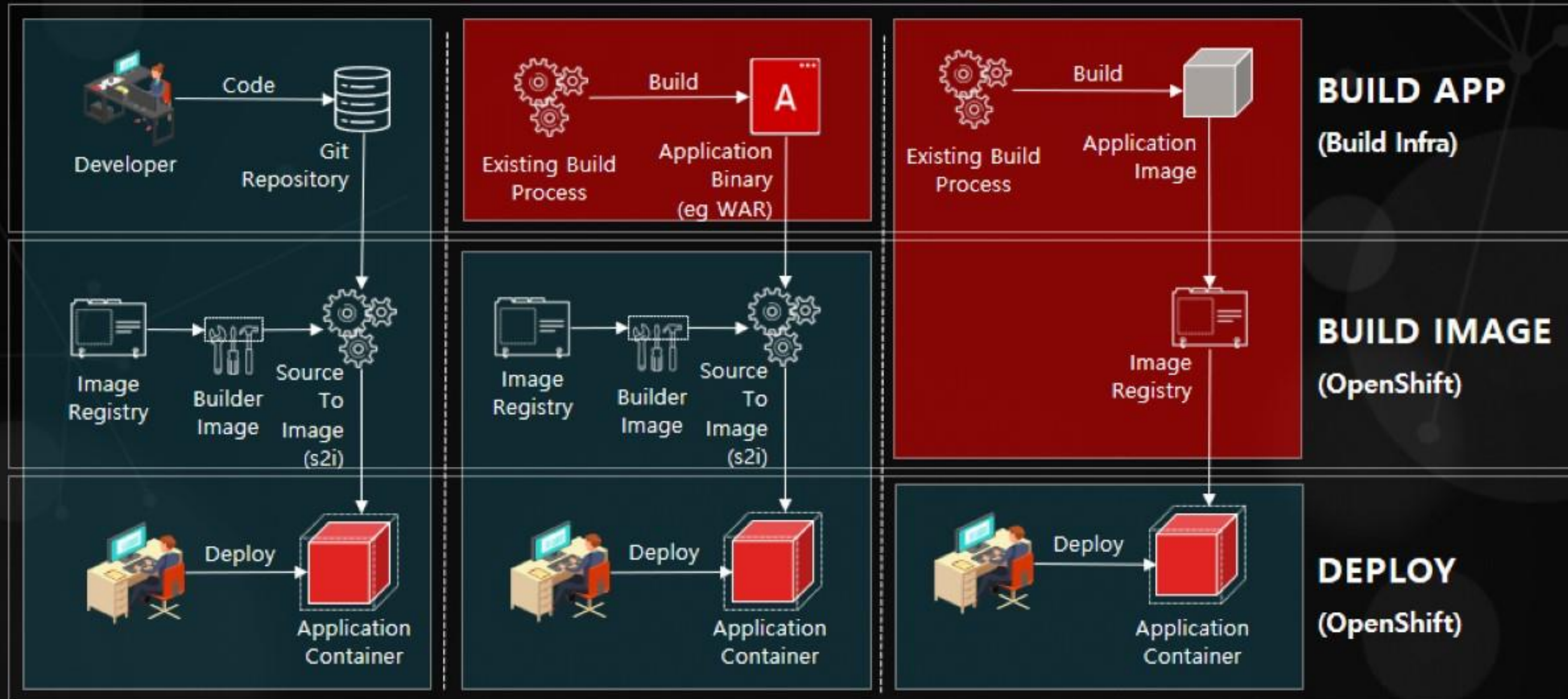
NON-PROD DEV TEST UAT PROD

OPENMARU © 2021 | All Rights Reserved.

Deploy in OpenShift

- 컨테이너 환경은 기존 환경에서 바이너리 빌드 후 배포와는 달리 이미지 빌드하는 추가작업이 필요
- OpenShift S2I 기능을 통해 Git 서버에 존재하는 소스코드만으로 서비스 가능한 Container Image 생성
- 컨테이너 환경을 구축 시 OpenShift S2I와 같은 자동화된 빌드 배포 Pipeline이 필수적

① Deploy Source Code with s2i ② Deploy App Binary with s2i ③ Deploy Container Images

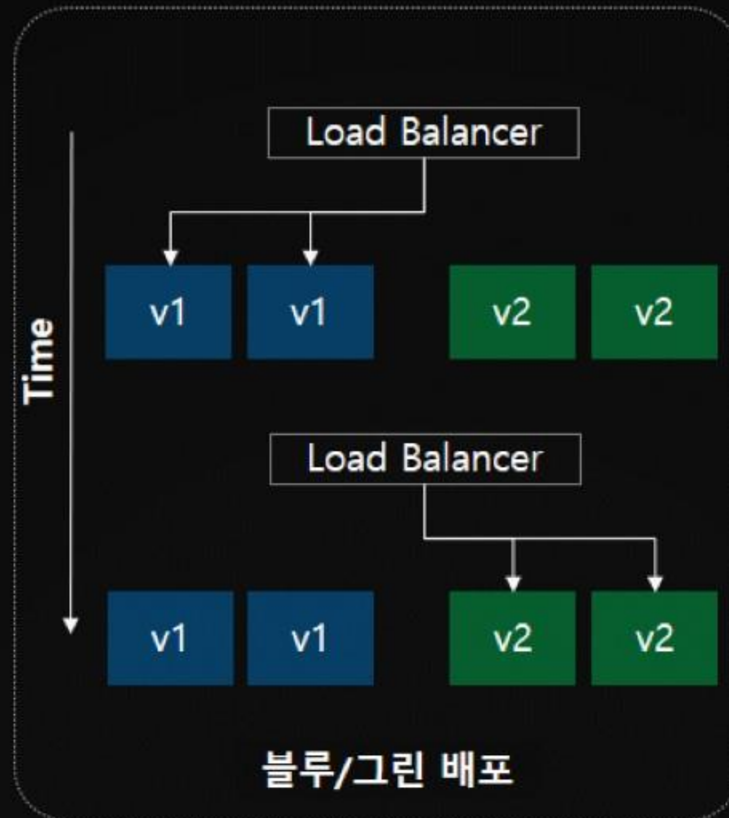


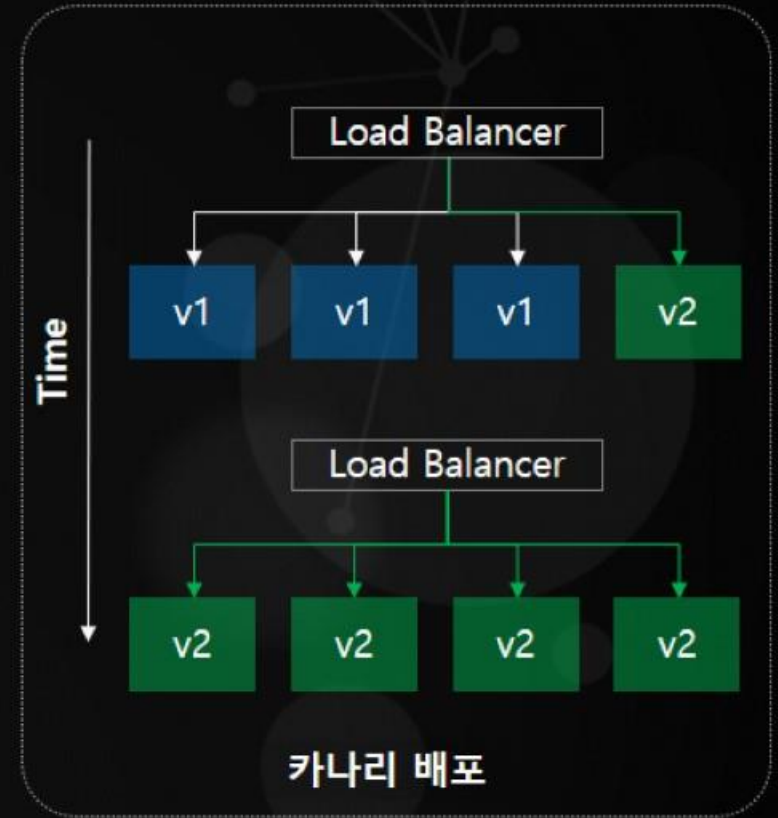
OPENMARU © 2021 | All Rights Reserved.

무중단 뿐만 아닌 고도화된 배포 방식

openmaru

- 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)





- Confidential -

OPENMARU © 2021 | All Rights Reserved.

개발이 끝난 애플리케이션을 컨테이너 환경에 배포 / OpenShift Template

openmaru

OpenShift 에서 생성할 오브젝트들의 집합을 매개변수화 하여 일괄 처리 할 수 있게 하는 기능

- 운영에 필요한 오브젝트 **한번에 생성**
  - ImageStream
  - BuildConfig
  - DeploymentConfig
  - Pod
  - Service
  - Route
  - ...
- 동일한 빌드 환경을 **반복적으로 이용 가능**
- 통합 관리가 용이

```yaml
    template.openshift.io/support-url: https://access.redhat.com
    version: "5.1.0-8.0-7.4.1"
  namespace: openshift
  name: openjdk11-ubi8-s2i-openmaru
objects:
  - apiVersion: v1
    kind: Service
    metadata:
      annotations:
        description: The web server's http port.
      labels:
        application: ${APPLICATION_NAME}
      name: ${APPLICATION_NAME}
    spec:
      ports:
        - port: 8080
          targetPort: 8080
      selector:
        deploymentConfig: ${APPLICATION_NAME}
  - apiVersion: route.openshift.io/v1
    id: ${APPLICATION_NAME}-http
    kind: Route
    metadata:
      annotations:
        description: Route for application's http service.
      labels:
        application: ${APPLICATION_NAME}
      name: ${APPLICATION_NAME}
    spec:
      host: ${HOSTNAME_HTTP}
      to:
        name: ${APPLICATION_NAME}
  - apiVersion: image.openshift.io/v1
    kind: ImageStream
    metadata:
      labels:
        application: ${APPLICATION_NAME}
      name: ${APPLICATION_NAME}
  - apiVersion: build.openshift.io/v1
    kind: BuildConfig
    metadata:
      labels:
        application: ${APPLICATION_NAME}
      name: ${APPLICATION_NAME}
    spec:
```

Openmaru-Template.yaml

OPENMARU © 2021 | All Rights Reserved.

운영환경에 적합하게 커스텀된 OPENMARU 표준 템플릿

openmaru

기존 템플릿들의 한계와 고객들이 자주 필요로 빌드 설정을 추가하여 보다 쉬운 빌드 및 커스텀이 가능

- **RedHat 템플릿에 OPENMARU 가 컨테이너 운영 환경에 맞게 자주 사용되고 변경하는 변수 추가 (Java Options, Maven mirror URL etc..)**
- **RedHat 템플릿 에서 제공하는 Maven 과 제공하지 않는 Gradle 빌드 관리 도구까지 지원하여 빌드 환경의 넓은 선택폭**

Instantiate Template

Namespace *
PR test3

Application Name *
jws-app
The name for the application.

Custom http Route Hostname

Custom hostname for http service route. Leave blank for default hostname, e.g.: <application-name>-<project>.<default-domain-suffix>

Git Repository URL *
http://192.168.23.210:7990/scm/sam/test-gradle-spring-boot.git
Git source URI for application

Git Reference
master

Git user Secret
gitlab
Git user secret

ImageStream Namespace *
openshift
Namespace in which the ImageStreams for Red Hat Middleware images are installed. These ImageStreams are normally installed in the openshift namespace. You should only need to modify this if you've installed the ImageStreams in a different namespace/project.

Maven mirror URL

Maven mirror to use for S2I builds

ARTIFACT_DIR

MEMORY_LIMIT
1Gi
Container memory limit

MaxMetaspaceSize
256
Max MetaSpace Size

CPU_LIMIT
1000m
Container CPU limit

Java Options *
-Dfile.encoding=UTF-8 -Dspring.profiles.active=dev
JAVA_OPTS_APPEND

Health Check URL *
/
Health Check URL

Time Zone *

OPENMARU © 2021 | All Rights Reserved.

템플릿을 사용한 OPENSHIFT 빌드 / 배포

openmaru

Q1

개발팀은 OPENSHIFT를 몰라서 할 수 있는게 없어요

몰라도 가능한 빌드 배포

Q2

개발팀이 OPENSHIFT에서 배포를 어떻게 해야 하나요?

템플릿 인스턴스화를 통한 배포

Q3

운영팀은 개발완료 후 OPENSHIFT 운영 환경을 어떻게 구축해야 하나요?

템플릿에서 한번에 구성해주는 운영 환경

Q4

운영팀 입장에서 프로젝트마다 OPENSHIFT 기술지원이 필요한가요?

하나의 템플릿으로 모든 프로젝트에 사용 가능

- Confidential -

OPENMARU © 2021 | All Rights Reserved.

Platform As A Service

컨테이너 환경에서 보안은 어떻게 바뀌었을까요?

AS-IS(레거시) 환경에서 필요한 서버 보안

- OS 보안, 서버 접근제어 등 보안 5종 S/W를 OS 설치
  - OS 마다 설치하기 때문에 물리서버는 1Copy이고, 가상화는 VM 개수 만큼 설치
- 서비스에 따라 부가적인 보안 소프트웨어 필요(개인정보 검출, 비정형 암호화 등)
- 법적인 근거로 인해 서버보안 – 전자금융법, ISMS 등의 요건

OPENMARU © 2021 | All Rights Reserved.

To-Be(컨테이너) 환경에서 필요한 보안

opɛnmaru

왜 AS-IS 환경의 보안들이 필요하지 않을까?

실질적으로 컨테이너환경에서 필요한 보안들

- 컨테이너 실행 권한에 대한 보안 : SCC
  - Container breakout
- 취약한 Container Image 사용
  - 악성코드, 채굴 프로그램이 포함된 이미지
- Role Base Access Controle : RBAC
  - 사용자별 필요 권한 부여
- 컨테이너 플랫폼의 Audit 로깅
  - EFK
- 신뢰할 수 있는 컨테이너 런타임 보안
  - Capabilities, SELinux, Seccomp & Userremap
- 컨테이너간의 네트워크 격리
  - Network Policy

- Confidential -

OPENMARU © 2021 | All Rights Reserved.

OpenShift Architecture 보안 솔루션 설치 이슈

openmaru

OS 보안 솔루션

- 백업 Agent
- 서버 OS 보안
- 시스템 취약점 스캐너
- 개인정보 검출
- 비정형암호화
- 로그 수집 Agent
- 백신
- 서버접근제어

sshd : Allow Bastion, Deny All

서버 보안솔루션 설치 불가

OpenShift Cluster

- OpenShift Cluster 내부로는 Bastion 서버 이외에는 원천 차단
- Linux Security 기술을 이용하여 OpenShift Cluster의 Host OS 보호
- SELINUX, NAMESPACES, SECCOMP, CGROUPS
- 기존에 Host OS에 설치해야 했던 보안솔루션에 대한 종속을 탈피

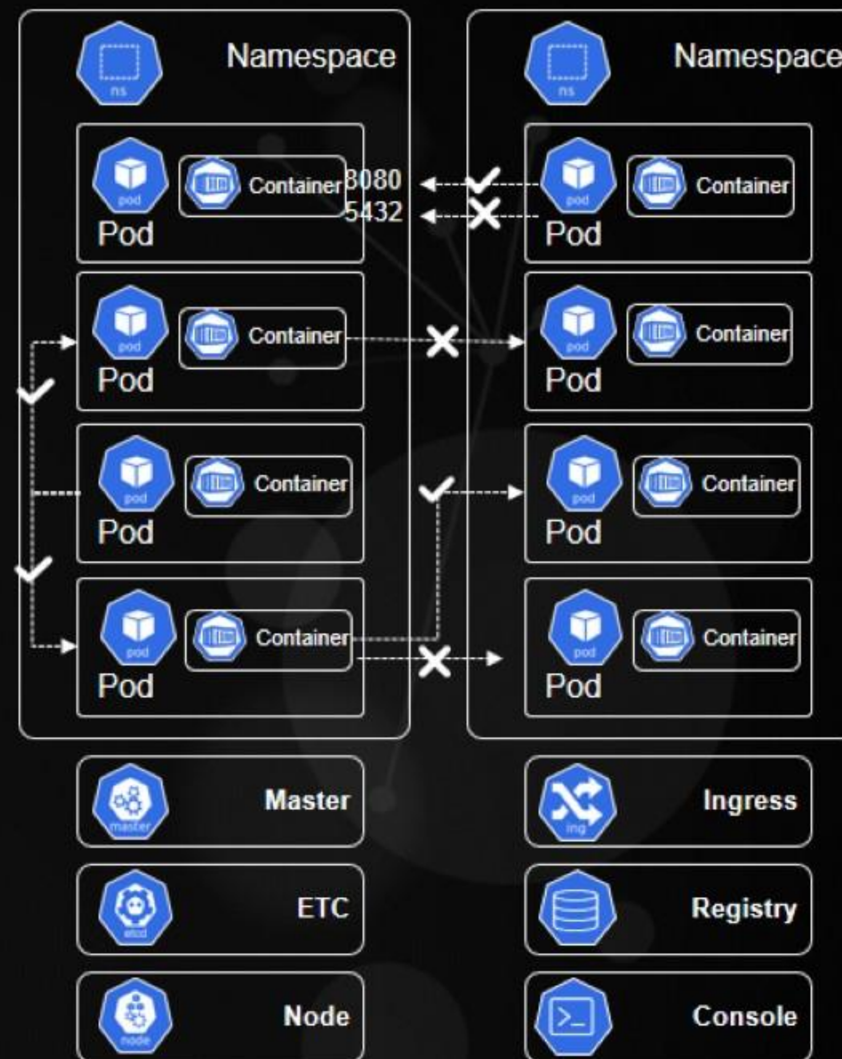
- Confidential -

OPENMARU © 2021 | All Rights Reserved.

OpenShift가 가지는 Network 보안

openmaru

- [기존]단독 서버 구성으로 방화벽이나 서버 접근 제어 솔루션으로 제어 가능
- [클라우드]단일하게 통합 관리하기 때문에 개별 서버에 대한 접근 제어가 불가능
  - OpenShift SDN에서 사용하는 NETWORKPOLICY로 Segmentation 구현
    - ✓ 클러스터 내의 서로 다른 애플리케이션을 분리.
    - ✓ 클러스터 내의 프로젝트를 분리.
  - Private 인증서 적용으로 각 컴포넌트에 대한 통신 암호화
    - ✓ MASTER, ETCD, NODES, INGRESS CONTROLLER, REGISTRY 등
  - 소프트웨어 정의 네트워킹(SDN)을 통해 클러스터 내부 포드 간의 통신을 제어
    - ✓ 클러스터 내 프로젝트 분리.
    - ✓ 클러스터 프로젝트 내 서로 다른 애플리케이션 분리.

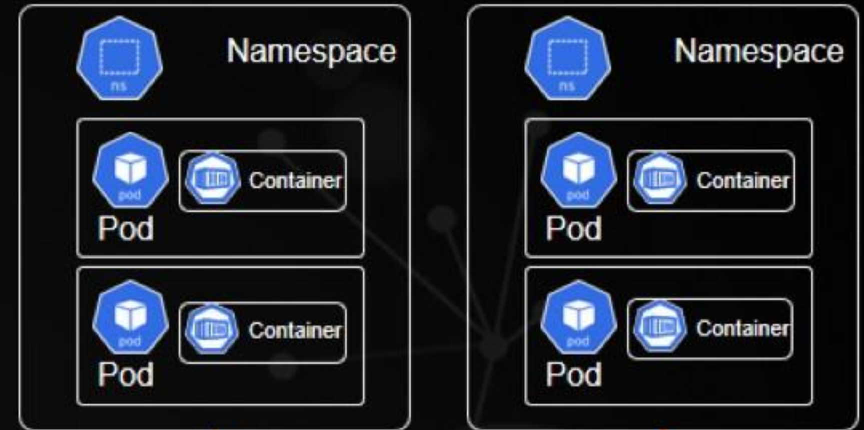


OPENMARU © 2021 | All Rights Reserved.

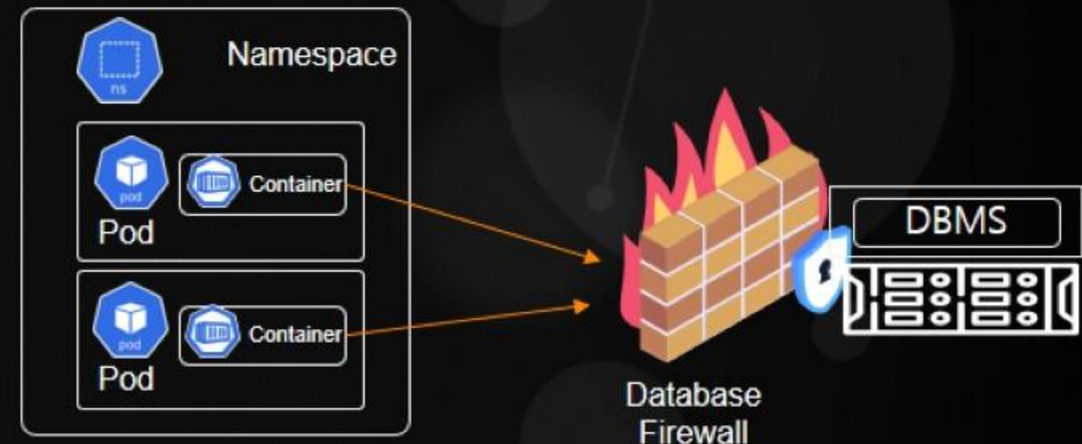
단일 클라우드 환경으로 물리적 네트워크 분리나 접근 제어 이슈

openmaru

- [기존]DMZ 를 통한 내부망과 외부망 분리 이슈
  - X사의 외부망 애플리케이션과 내부망 애플리케이션간의 통신을 격리시키고자 함
- [클라우드] Network Policy를 통한 네트워크 격리가 필요함
  - 컨테이너 환경은 기존 환경처럼 Server간 방화벽으로 제어하는 것이 아님



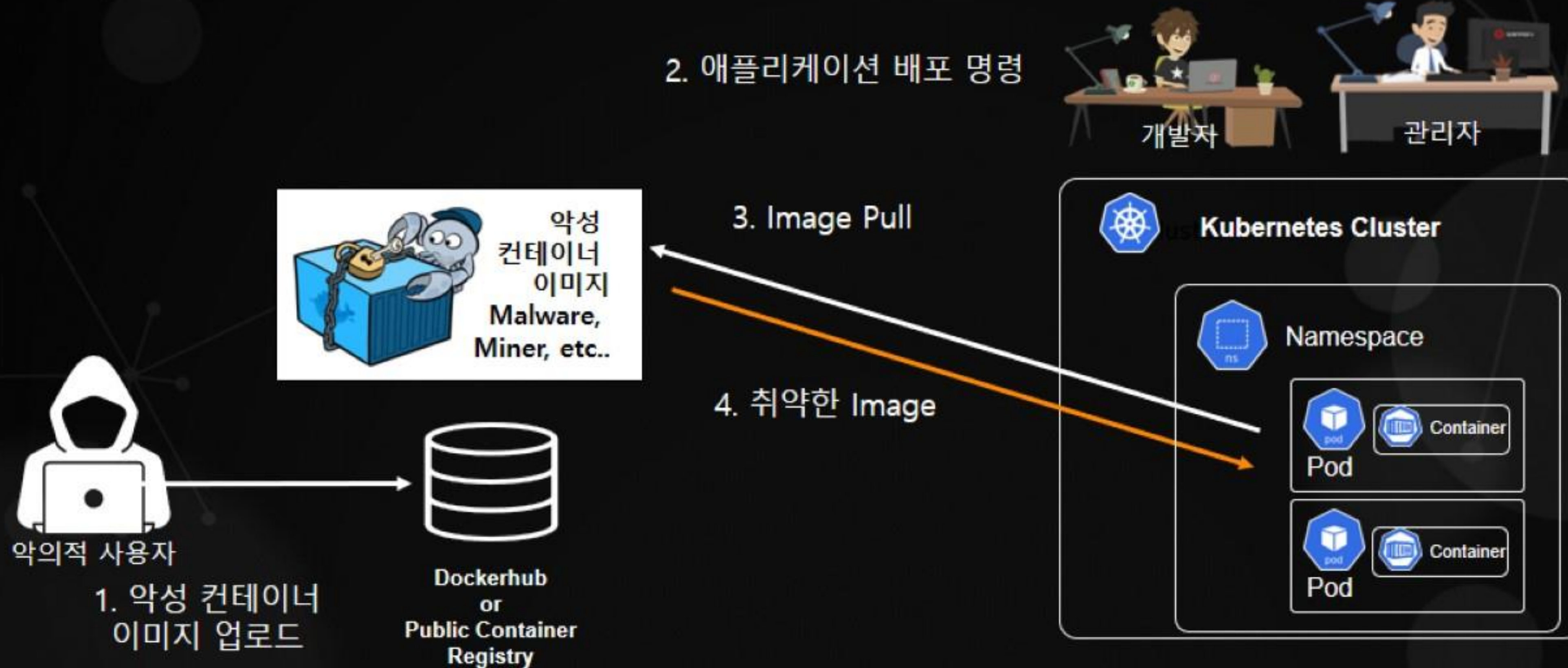
- [기존]서버별 IP 로 데이터베이스 접근 제어
  - X사의 외부망 애플리케이션과 내부망 애플리케이션간의 통신을 격리시키고자 함
- [클라우드] Egress 를 이용한 서비스 접근 제어
  - Database에 접속할 수 있는 Namespace에 Egress를 통해 외부 통신 특정 IP를 할당
  - Egress 정책이 할당되지 않은 Namespace는 애플리케이션에서 Database에 접속불가



OPENMARU © 2021 | All Rights Reserved.

신뢰할 수 없는 컨테이너 이미지 레지스트리

- **Public 한 Registry는 다양한 이미지가 있지만 취약점이 조치되지 않음**
- **Docker hub의 400만개 Container Image중 절반 이상이 1개 이상의 Critical 취약점 발견**
- **악성 소프트웨어가 포함된 Image 가능성**



OPENMARU © 2021 | All Rights Reserved.

Ending

openmaru

啐啄同時(줄탁동시)

- 병아리가 알에서 깨어나기 위해 달걀 속에 있는 병아리 뿐만 아니라 밖에 있는 어미 닭도 협력해야 한다

1. 내가 먼저 변해야 한다.
2. 경청해야 한다.
3. 타이밍이 중요하다.
4. 지속적인 노력이 있어야 한다.

고객분들의 노력과 관심 오픈나루와 레드햇의 협력으로 기존의 유연하지 않은 환경을 탈피, DX 실현

OPENMARU © 2021 | All Rights Reserved.

openmaru