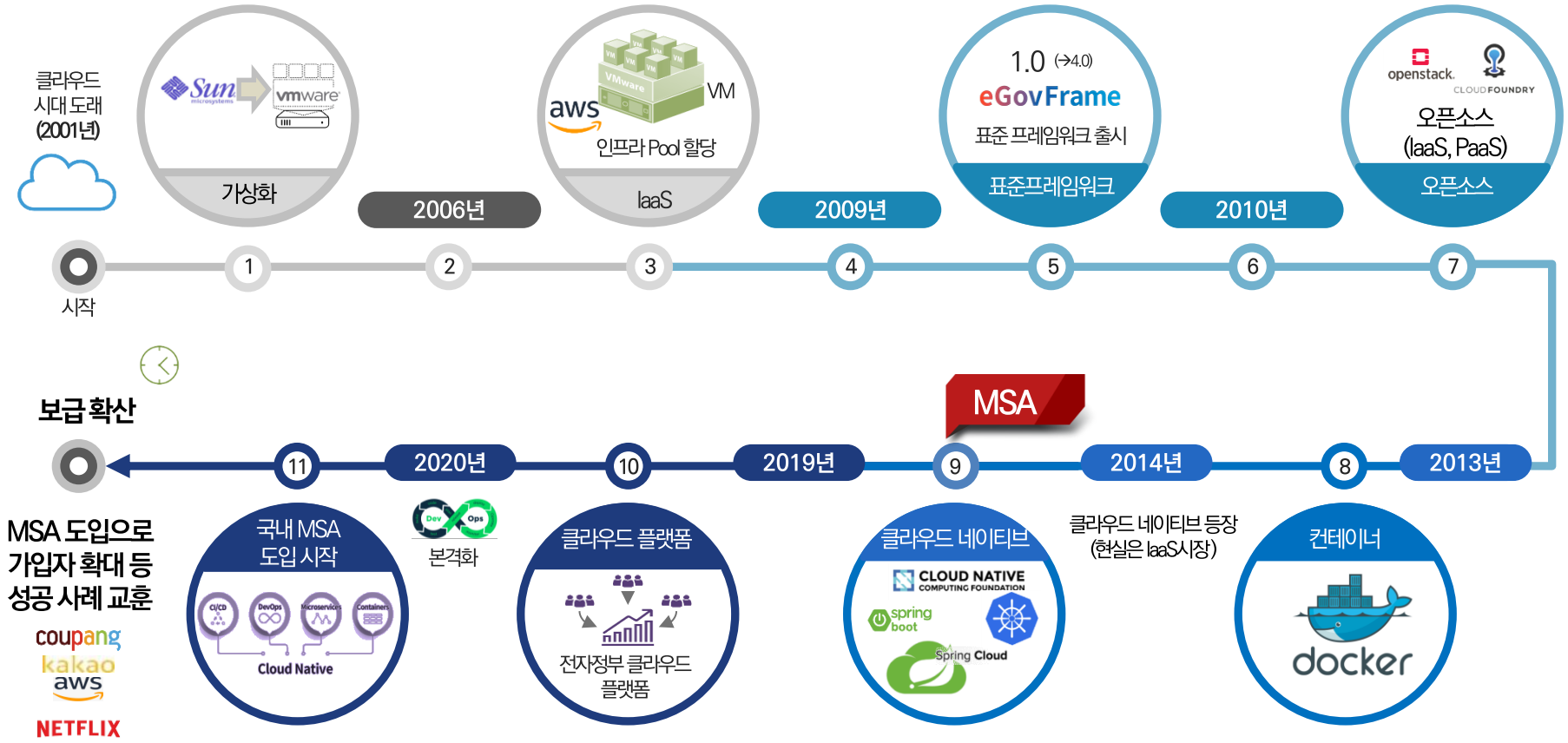
NIA 한국지능정보사회진흥원

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주가이드

교육기간 : ’21.11.22 ~ 11.28

클라우드 네이티브 발자취



***클라우드 네이티브** : MSA(Micro Service Architecture 약어)로 불리우며, 마이크로서비스 아키텍처 스타일은 단일 응용 프로그램을 자체 서비스로 실행하고, 경량 메커니즘(HTTP)으로 통신하는 작은 서비스 모음으로 개발하는 접근 방식임 (Martin Fowler)

클라우드 네이티브 발주자 가이드

발표 목차

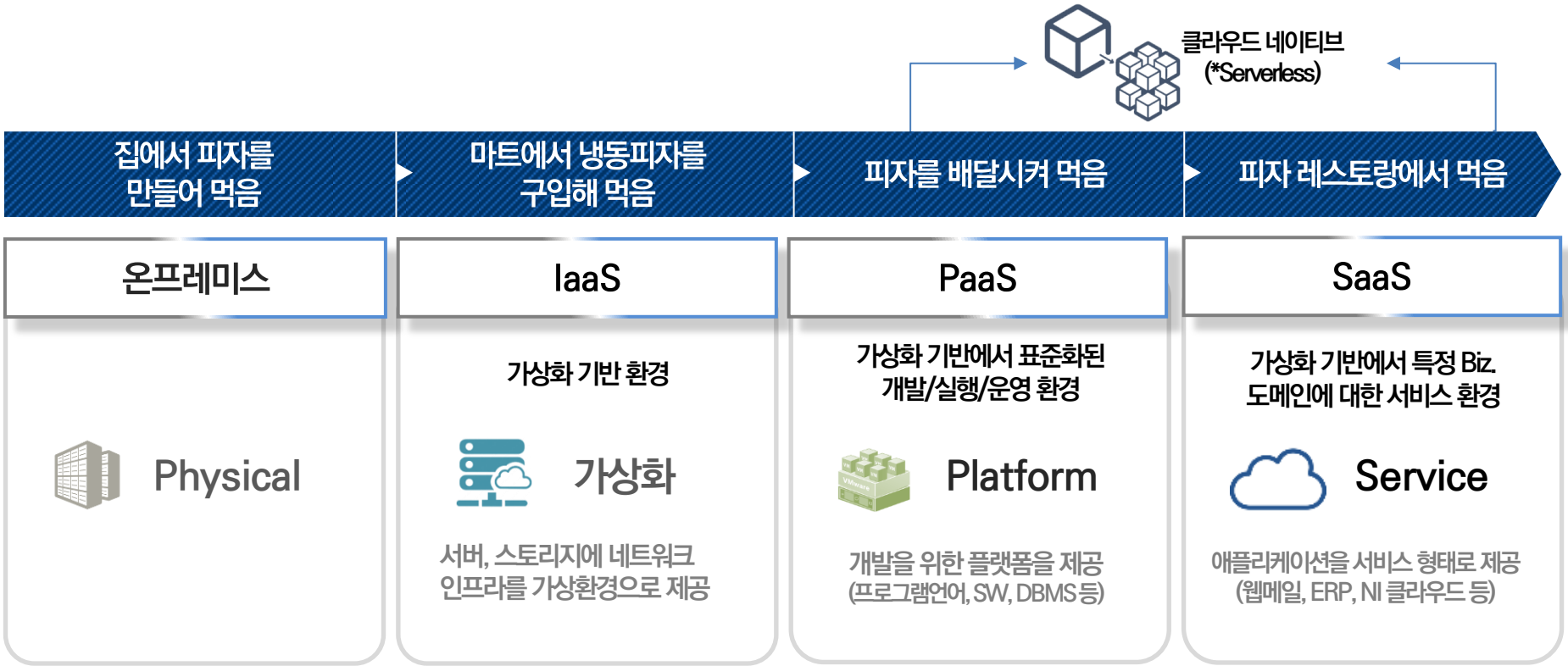
클라우드 네이티브 기반 행정·공공 서비스 확산 지원
클라우드 네이티브 발주자 가이드

클라우드와 클라우드 네이티브는 왜 필요한가?

NIA 한국지능정보사회진흥원

클라우드는 무엇인가?

가상화된 컴퓨팅 자원을 활용하여 소프트웨어, 플랫폼, 인프라 등의 IT 서비스를 제공하는 기술이며, 기존 IT 대비 효율적이고 신속한 서비스 제공이 가능함



* Severless : 서버리스 컴퓨팅은 서버가 존재하지만, 사용자의 운영 개입 없이 클라우드 플랫폼에서 자동으로 관리되는 서비스로, 애플리케이션 함수 실행 서비스 및 백엔드 서비스를 포괄

클라우드 서비스 모델

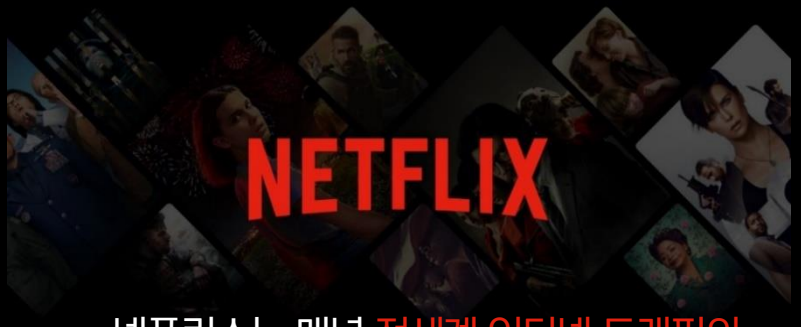
클라우드 서비스는 IaaS, PaaS, SaaS가 일반적으로 제공되는 형태이며, 컨테이너와 서버리스 등의 최신 기술이 발달되면서 CaaS와 FaaS 등 다양한 클라우드 서비스 모델로 세분화되고 있음

온프레미스	IaaS	CaaS	PaaS	FaaS	SaaS
인프라부터 SW까지 전부 구축하고 개발	인프라 요소를 서비스 형태로 제공하며, 원하는 사양의 서버를 VM으로 생성	서비스로 제공되는 컨테이너를 활용해 애플리케이션을 배포	애플리케이션 개발에 집중할 수 있도록 인프라와 런타임 환경제공 (서버 단위 확장)	실행할 함수 코드에 집중 가능 (서버 없이 함수 단위 확장) Serverless	제공되는 SW를 사용하는 형태
기능	기능	기능	기능	기능(Functions)	기능
애플리케이션	애플리케이션	애플리케이션	애플리케이션	애플리케이션	애플리케이션
런타임	런타임	런타임	런타임	런타임	런타임
미들웨어	미들웨어	컨테이너(Container)	컨테이너	컨테이너	컨테이너
O/S	O/S	O/S	O/S	O/S	O/S
가상화	가상화	가상화	가상화	가상화	가상화
하드웨어	하드웨어	하드웨어	하드웨어	하드웨어	하드웨어

*범례 : 이용기관 직접 준비 | 이용기관, 공급자 공동책임 | 공급자 제공

클라우드 네이티브 성공사례

Netflix는 클라우드 네이티브 성공사례로, 전세계 확산 서비스를 제공하여 사용자에게 고화질 동영상 서비스를 안정적으로 제공



넷플릭스는 매년 전세계 인터넷 트래픽의 15% 이상을 차지하고 있음

2021년에는 이미 2억 400만명
(3년 만에 2배 증가)의 구독자가 생겼고,
매 4분기 마다 5백만명이
200개 이상의 국가에서 구독 中

NETFLIX OSS
Netflix OpenSourceSoftware Center

spring boot + NETFLIX

기술팀이 8년간 노력해온 결과

- 인프라를 자체 센터에서 Public 클라우드 이전
- 모놀리틱 프로그램을 작게 관리할 수 있는 마이크로서비스 아키텍처로 변경
- 애플리케이션 함수 실행 서비스하는 서버리스 컴퓨팅 및 백엔드 아키텍처를 제공

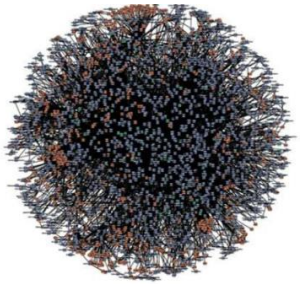
클라우드 네이티브 국내·외 선도 도입 사례

선진 사례

AWS

빠른 배포 구현

수 천개 팀(자율적 DevOps팀) X 마이크로서비스 아키텍처
X 지속적 배포(CD) X 다양한 개발 환경



수 천개 팀
마이크로서비스 아키텍처
지속적 배포(CD)
다양한 개발 환경

넷플릭스

가입자 대상서비스 확대

Netflix Open Source Software Center

You Build it　You Run it　You Support it

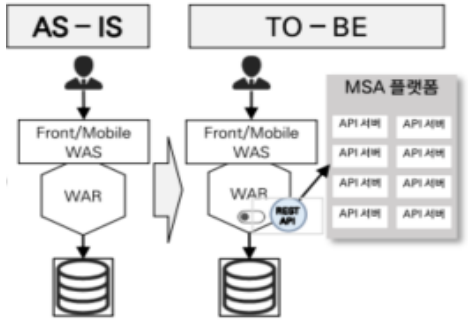
KAKAO

계열사 신규서비스 확대 및 빠른 출시 사례

카카오의 애자일 문화, 일하는 방식 관리를 위한
전담팀 및 개발플랫폼 운영

11번가

서비스 분리를 통한 점진적 MSA전환



Micro services

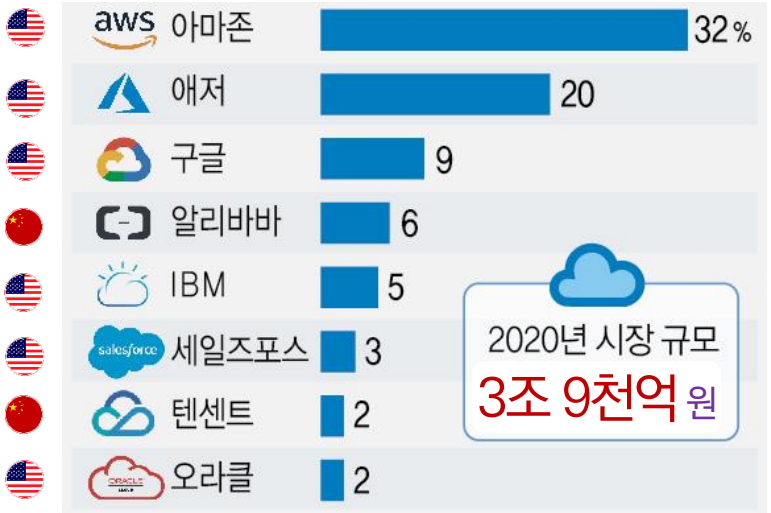
업무별 도메인별
분리된 애플리케이션
(Microservice)

국내·외 클라우드 시장 (1/3)

현재 국내·외 클라우드 시장은 외산이 주도
다양한 서비스, 손쉬운 개발·배포 환경으로 외산 클라우드 선호, 국내 퍼블릭 PaaS 성장中

국내 클라우드 시장의 외산 장악

"국내 클라우드 서비스 시장 점유율"
- 2020년 4분기 -



kt NAVER 국내기업 점유율은 미비

출처연합뉴스https://m.yna.co.kr/view/GYH20210205000400044

국내 퍼블릭 PaaS 시장 급격한 성장전망

"2020년 클라우드 부문 매출 현황"

NAVER	2,737억 원	40% 상승 ↑
kt	4,000억 원	20% 상승 ↑
NHN	1,600억 원	250% 상승 ↑

가비아, 스마일서브 등이 열심히 사업 진행 중

일반 기업 관련 동향

SAMSUNG 삼성전자

6,000억 중
AWS : 자사 클라우드 = 6:4

국내·외 클라우드 시장 (2/3)

현재 국내·외 클라우드 시장은 IaaS에서 PaaS로 중심축 이동 中

개발자를 위한 플랫폼 서비스, PaaS

경쟁력 있는 PaaS가
SaaS 및 IaaS 점유율 확대의 원천

- 운영 체제
- 실행 환경
- 개발 환경
- 분석 환경
- 데이터 베이스
- 웹 서버

클라우드 서비스의 차별성 PaaS가 결정

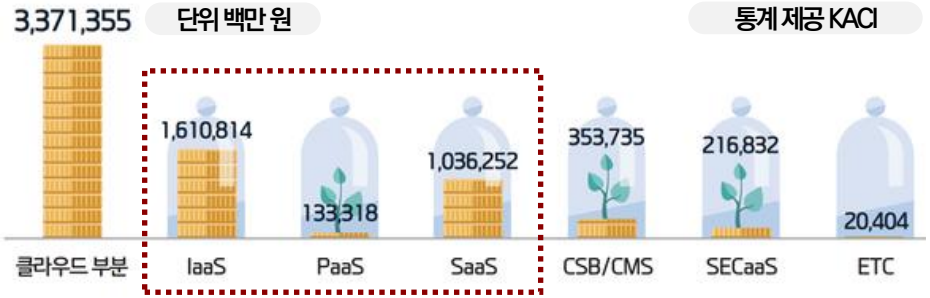
글로벌 PaaS 시장 규모 2026년까지
연평균 19.6%로 성장 예상

2020년 약 67조 원 → 2026년 약 196조 원

출처: '2026년까지 서비스로의 플랫폼 시장' 보고서 MarketsandMarkets

국내 PaaS 시장의 무한한 가능성에 주목해야 할 시점

2019년 국내 클라우드 서비스 부문별 매출 현황



IaaS와 SaaS 성장세는 이미 정점

클라우드 시장 확대와 함께 국내 PaaS 시장 성장 필수

PaaS 시장의 성장 가속 시점 도래

국내·외 클라우드 시장 (3/3)

국내 PaaS 시장의 규모는?

2020년 클라우드 서비스별 예상 매출액 (통계청 자료)

구 분		기업수(개)	IaaS	PaaS	SaaS	CSB/CMS	SECaaS	기타	전체
전체		(1225)	2,378,472	100,091	907,682	352,384	213,276	22,052	3,973,958
종사자수	1~9인	(312)	201,669	3,254	67,561	6,035	6,397	3,673	288,590
	10~29인	(437)	80,117	38,657	164,925	12,281	44,561	1,565	342,107
	30~99인	(289)	492,598	29,358	241,872	26,836	131,793	10,015	932,472
	100~299인	(121)	223,838	14,395	293,268	12,259	18,597	0	562,357
	300인 이상	(66)	1,380,250	14,427	140,056	294,973	11,928	6,798	1,848,432
기업규모	중견기업 이상	(83)	1,466,305	14,680	196,763	283,480	101,276	6,798	2,069,302
	중소기업	(1142)	912,168	85,412	710,920	68,904	112,000	15,254	1,904,656

전체 3조 9천 7백억 원

IaaS 2조 1천억 원 | SaaS 9천억 원 | 기타 6천억 원

PaaS 1천억 원

2019년 국내 클라우드 서비스별 매출 증가율

1위	2위	3위
SaaS 51.74% 증가	IaaS 22.0% 증가	PaaS 15.71% 증가

2019년 국내 클라우드 서비스별 기업 수 증가율

1위	2위	3위
PaaS 55→131 2017년 대비	SaaS 336→561 2017년 대비	IaaS 314→365 2017년 대비

현재 국내 PaaS 시장 매출규모는 미미하지만,
빠른 시장 성장성을 대비한 기업의 수는 증가하고 있는 상황

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주자 가이드

클라우드 네이티브는 어떤 특성을 가지고 있는가?

NIA 한국지능정보사회진흥원

컨테이너 기술 (1/3)

컨테이너 기술을 기반으로 이식성을 제공합니다.

컨테이너, “세계 경제사를 바꾼 大혁신적 발명품” → 수송시간 단축, 물류 수송비 인하

- 세계 화물 운송량 5배 증가
- 화물의 항구 체류 시간 75% 절감
- 해상운송비 60% 절감

형태에나 크기가 다른 물류는 이동 어려움

컨테이너
혁신적 변화

정형적인 형태의 모양으로 이동에 편리함

컨테이너 기술 (2/3)

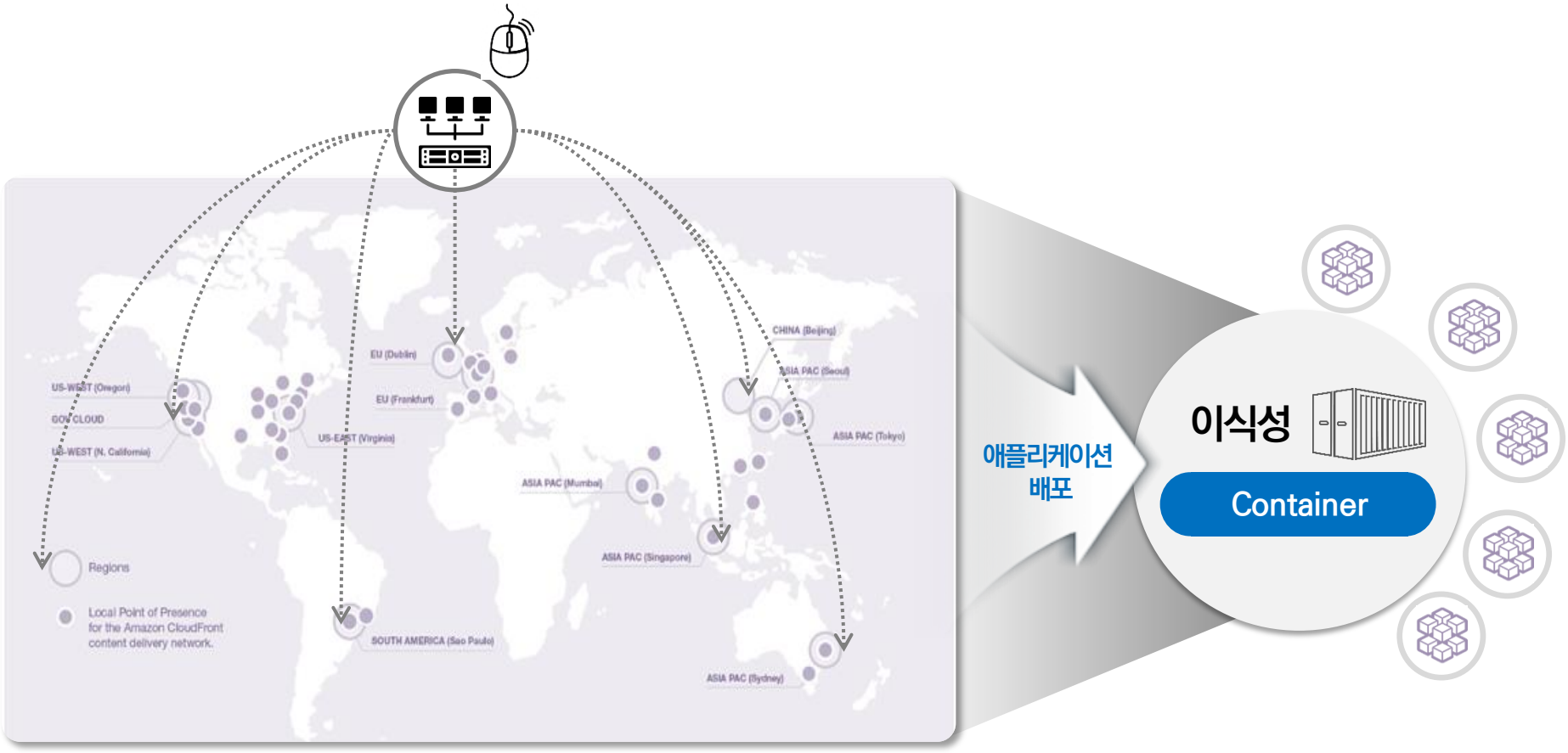
컨테이너 기술을 기반으로 이식성을 제공합니다.

CaaS(Containers-as-a-Service), "세계 IT를 바꾼 大혁신적 발명품" → 서비스 개발 및 운영의 현대화

컨테이너 기술 (3/3)

작고·가볍고 손쉽게 배포를 제공합니다.

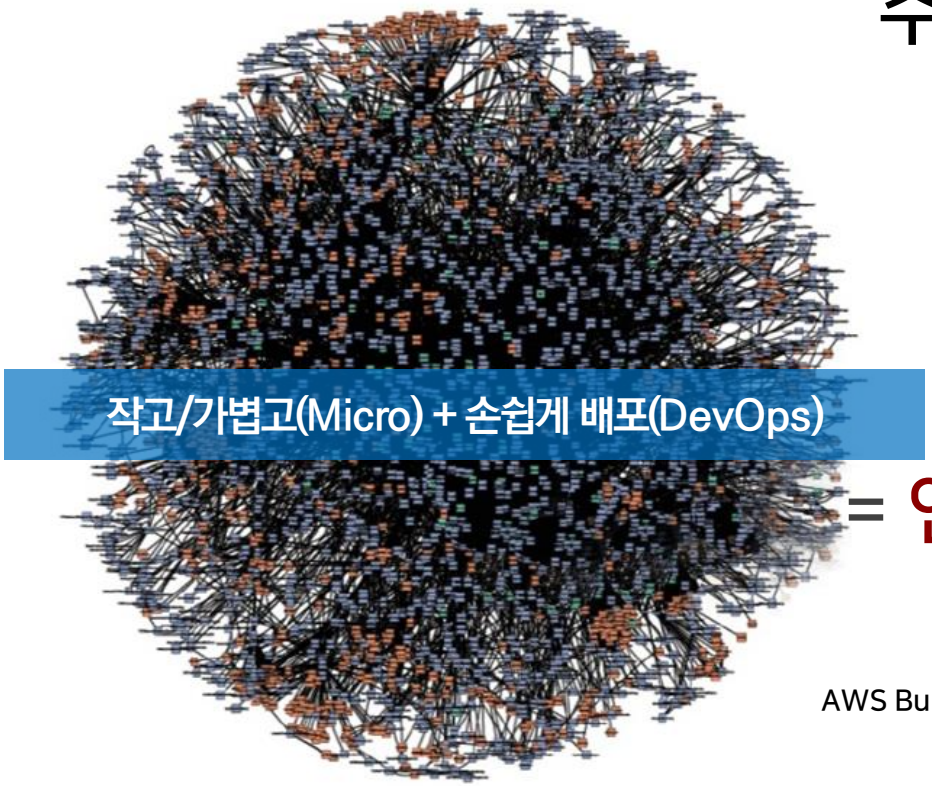
클라우드 환경에서의 서비스 배포 → 전세계에 한번의 클릭으로 애플리케이션 배포



클라우드 네이티브 (1/2)

작고·가볍고 손쉽게 배포를 제공합니다.

클라우드 환경에서의 서비스 배포 → 전세계에 한번의 클릭으로 애플리케이션 배포



작고/가볍고(Micro) + 손쉽게 배포(DevOps)

수 천개 팀 (자율적 DevOps팀)
× 마이크로서비스 아키텍처
× 지속적 배포(CD)
×다양한 개발 환경

= 연간 1억 9천만회 배포(2020년)

(매초 마다 6번의 배포)

AWS Builders, AWS Fargate와 Amazon ECS를 활용한 CI/CD 모범사례 2020”

aws.amazon.com

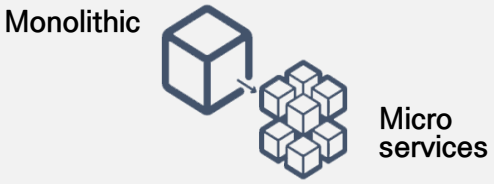
클라우드 네이티브 (2/2)

작고·가볍고 손쉽게 배포를 제공합니다.

클라우드 환경에서의 서비스 배포 → 전세계에 한번의 클릭으로 애플리케이션 배포

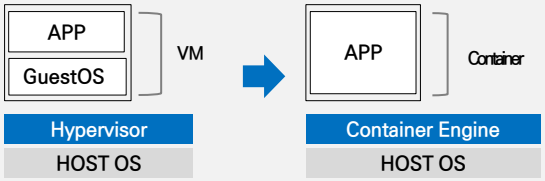
1 마이크로서비스 (MicroService)

애플리케이션을 작은 기능 단위,
마이크로서비스로 구축



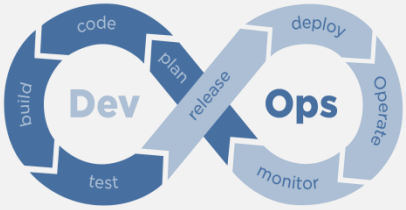
2 컨테이너 (Container)

경량화 된 가상화 기술,
컨테이너로 애플리케이션 배포



3 DevOps & CI/CD

자동화된 애플리케이션 개발-운영 프로세스 적용

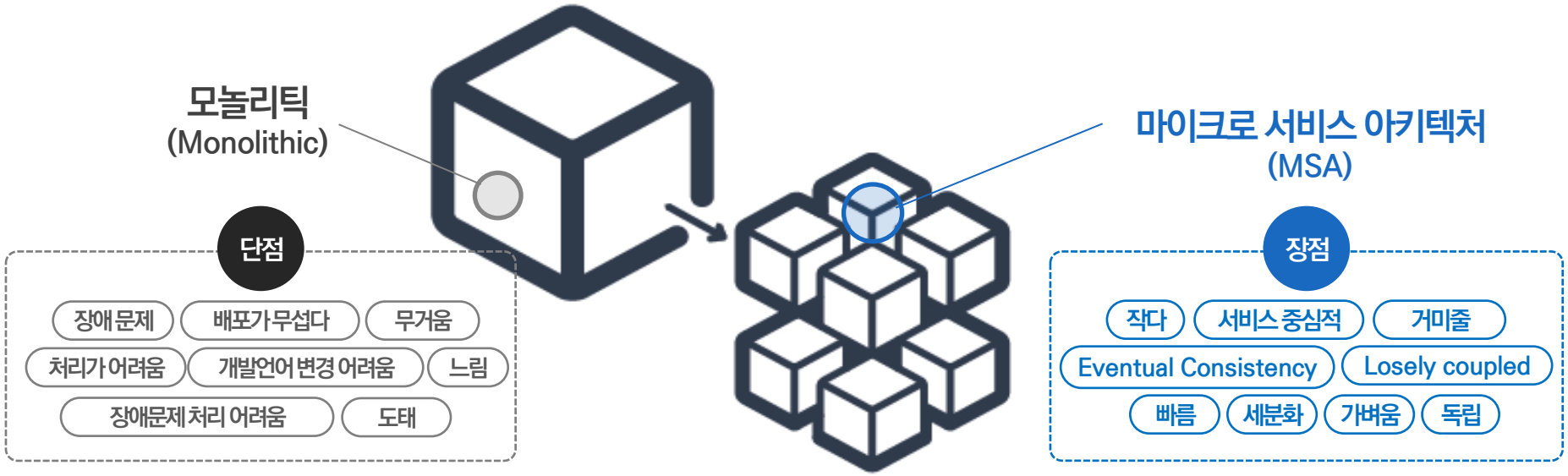


- 배포 자동화, 전주기 관리
- 지속적인 배포, 즉시배포
- 지속적 통합 도구제공

마이크로 서비스 아키텍처 (1/5)

탄력적, 선택적인 서비스 확장을 제공합니다.

API로 통신하는 소규모의 독립적인 서비스로 구성하여,
클라우드 환경에 최적화된 느슨한 결합 (빠른, 신속한, 편리한 업무)

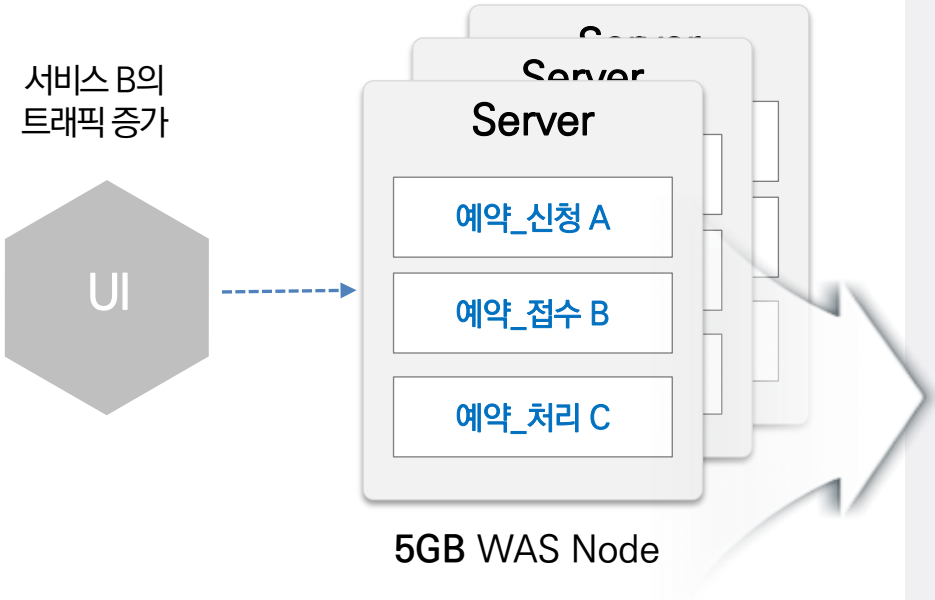


*** 마이크로서비스 아키텍쳐는** 통상 MSA로 불리우는데, 클라우드의 특성과 장점을 적용하여, 클라우드 환경에 최적화된 응용SW 설계 기법을 정의

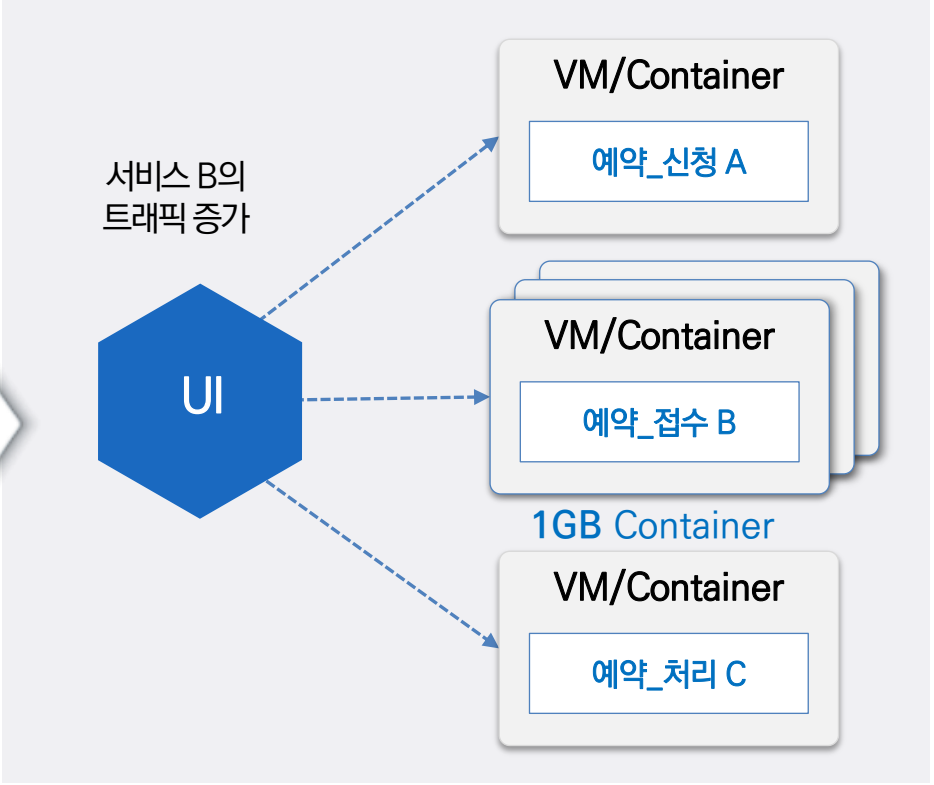
마이크로 서비스 아키텍처 (2/5)

탄력적, 선택적인 서비스 확장을 제공합니다.

기존에는 요구사항이나 전체를 다시 개발, 빠른 요구에 대응 부합하는 선택적 개발
업무 용도에 적합한 개발언어, DBMS 등의 기술 적용



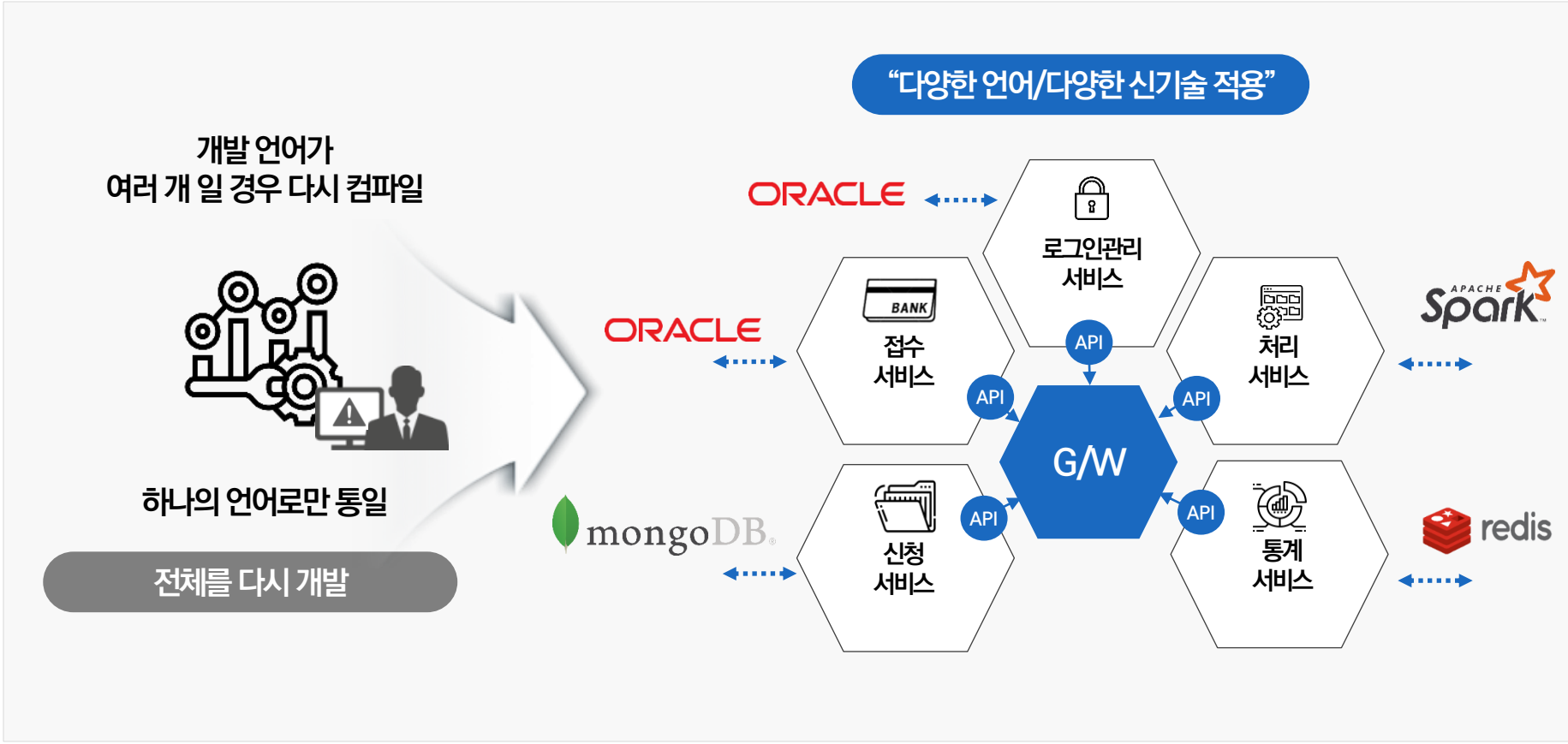
고사양의 서버 단위로 증설



마이크로 서비스 아키텍처 (3/5)

탄력적, 선택적인 서비스 확장을 제공합니다.

기존에는 요구사항이나 전체를 다시 개발, 빠른 요구에 대응 부합하는 선택적 개발
업무 용도에 적합한 개발언어, DBMS 등의 기술 적용

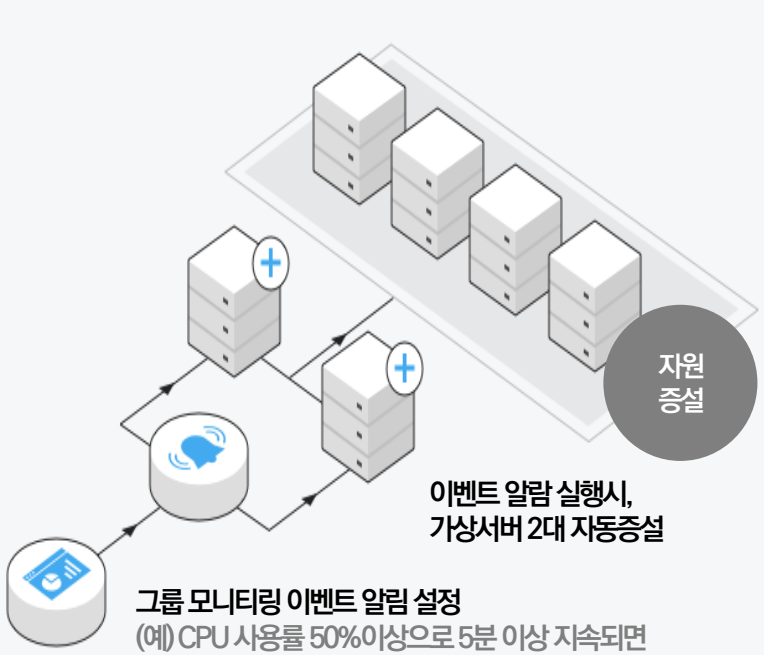


마이크로 서비스 아키텍처 (4/5)

탄력적, 선택적인 서비스 확장을 제공합니다.

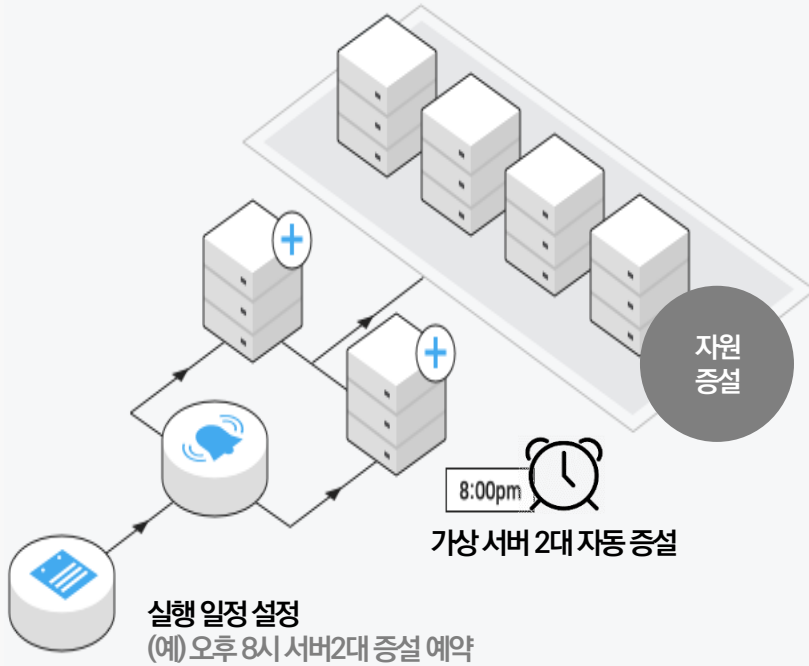
Auto Scaling 시스템 자원들의 메트릭(Metric) 값을 모니터링하여 서버 사이즈를
자동으로 조절하여 업무 예상치 못한 서비스 부하에 효과적으로 대응

1 서버 그룹 모니터링 기반 Auto Scaling



그룹 모니터링 이벤트 알림 설정
(예) CPU 사용률 50%이상으로 5분 이상 지속되면
2대를 증설하는 정책설정

2 스케줄링 기반 Auto Scaling

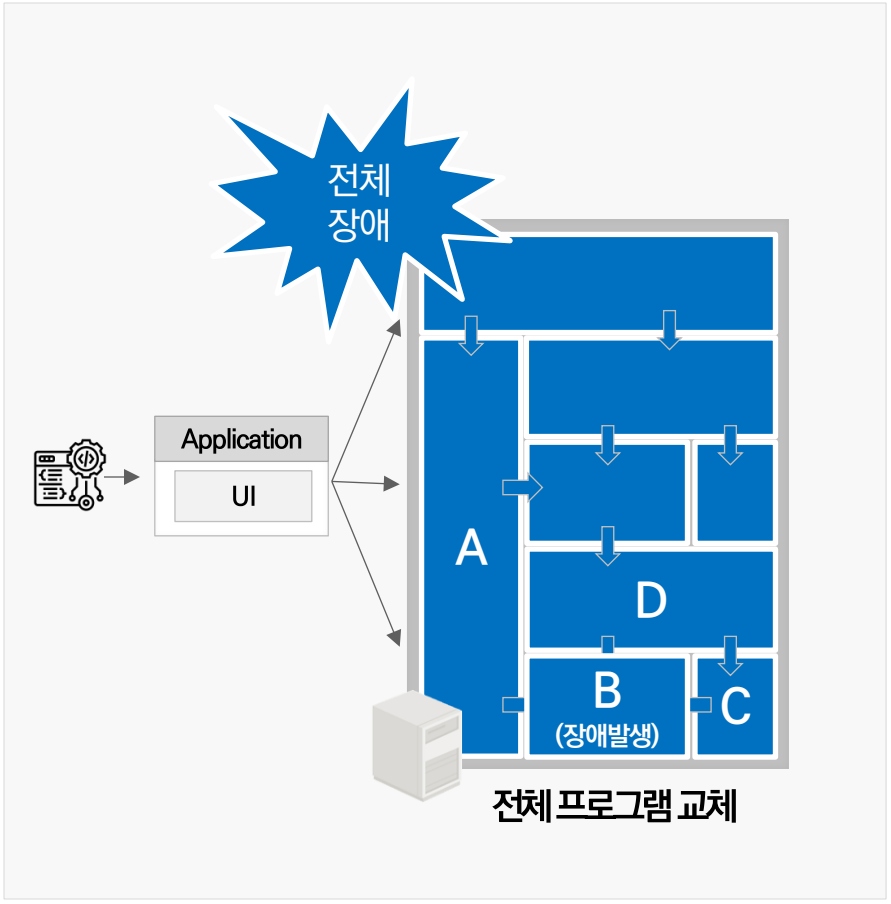


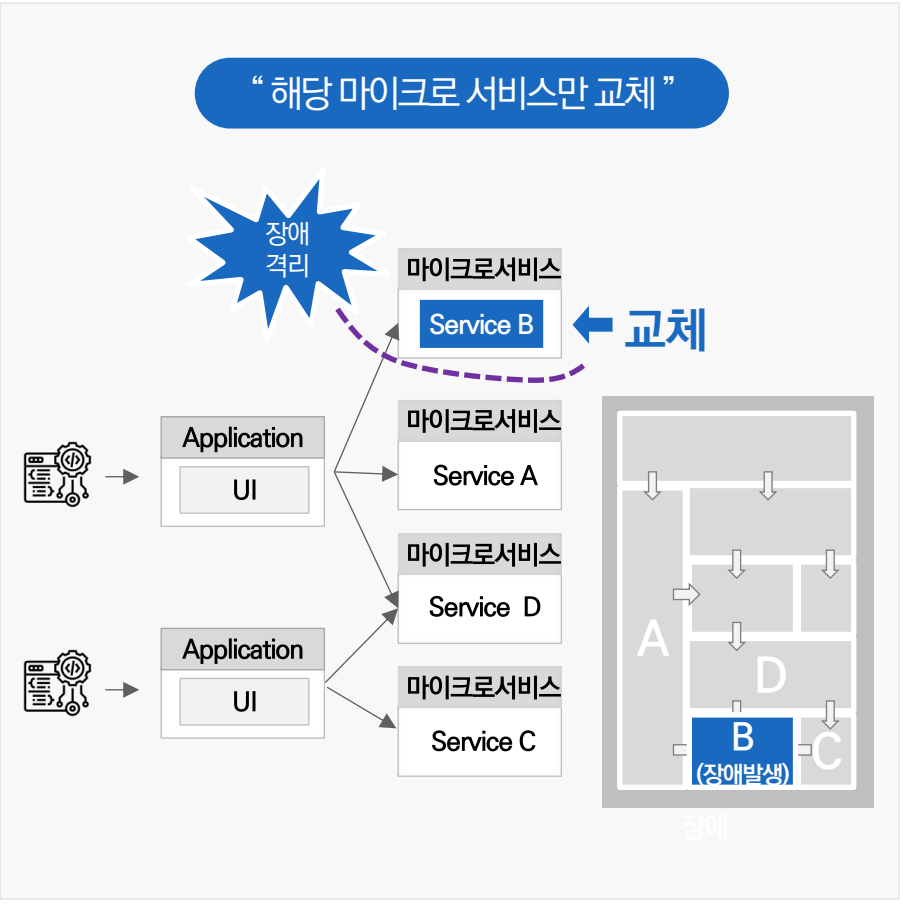
실행 일정 설정
(예) 오후 8시 서버2대 증설 예약

마이크로 서비스 아키텍처 (5/5)

탄력적, 선택적인 서비스 확장을 제공합니다.

작은 서비스 단위로 업데이트, 변경 및 교체를 제공하며
마이크로 서비스별 격리를 통하여 전체 장애 전파를 최소화





클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주자 가이드

클라우드 네이티브는 어떻게 발전하고 있는가?

NIA 한국지능정보사회진흥원

클라우드 네이티브 정의

클라우드 네이티브란?

클라우드 네이티브
(형용사/명사)

클라우드 컴퓨팅의 장점을 최대한 활용할 수 있는
(효율적인 자원이용, 탄력적 수요 대응 등)
정보시스템 분석·설계·구현 및 **실행하는 환경**

클라우드 네이티브
애플리케이션

클라우드 환경에서 실행되는 **애플리케이션**

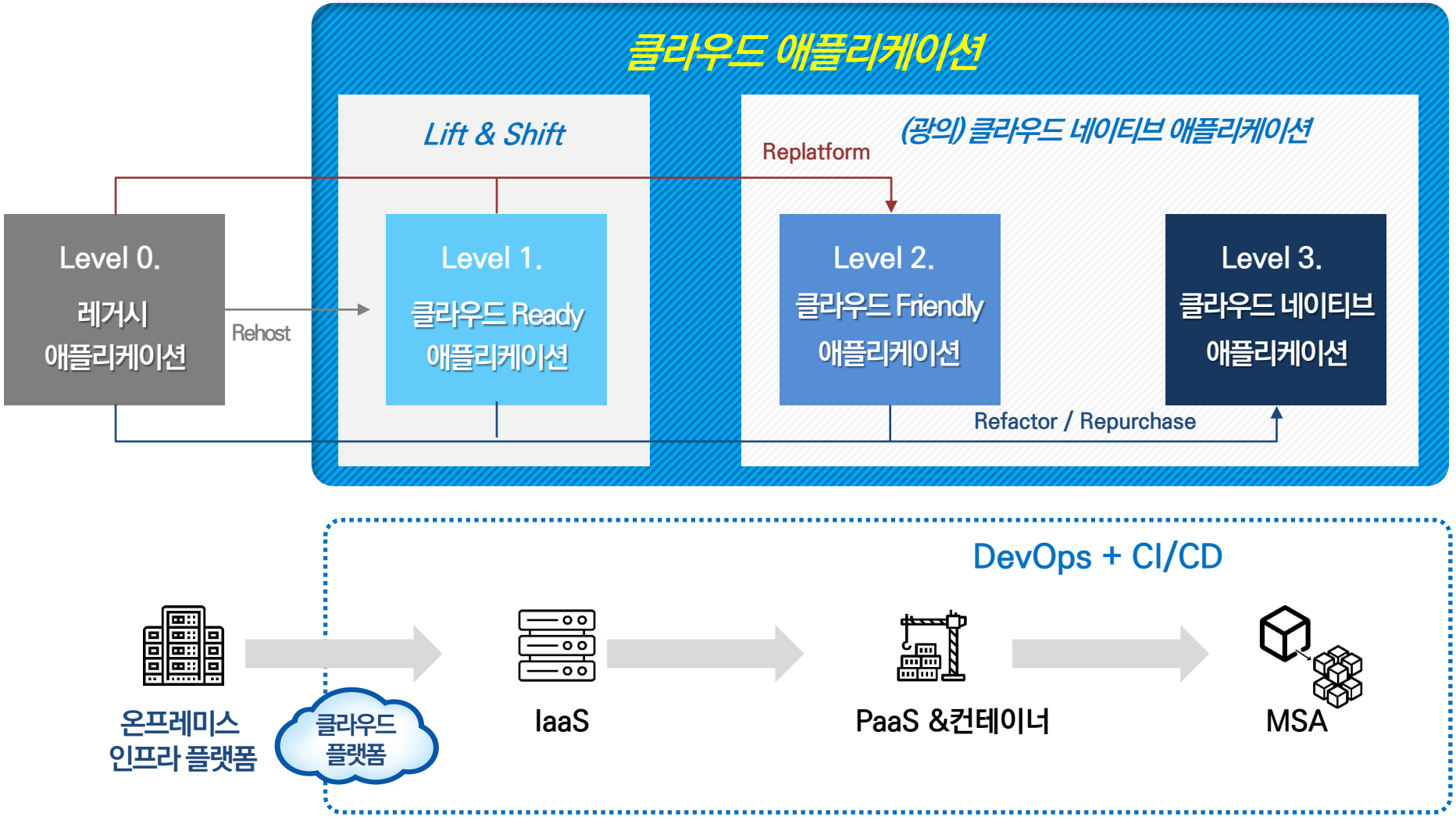
CNCF (Cloud Native Computing Foundation) v1.0
클라우드 네이티브 전환할 수 있는 기술 정의 및 오픈 소스를 관리하는 단체

- 퍼블릭, 프라이빗, 하이브리드 **클라우드 환경에서** 확장성 있는 **애플리케이션**
- 컨테이너, 서비스 메시(Mesh), **마이크로서비스(Micro Service) 인프라 구조**, 선언적 API로 접근
- 자동화, 회복성, 편리성, 가시성을 갖는 **느슨하게 결합된 시스템** (개발 및 실행 환경)
- 엔지니어는 최소한의 수고로, 영향력이 크고, 예측 가능한 변경을 할 수 있는 기술 정의

클라우드 네이티브 변화

클라우드 네이티브는 어떻게 발전하고 있는가?

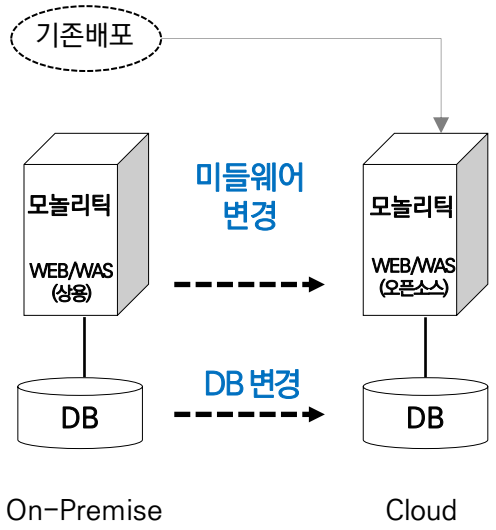
클라우드 네이티브 성숙도 단계 – 애플리케이션 관점



클라우드 네이티브 성숙도 단계 – 인프라 관점

Level 1: Cloud Ready
클라우드 준비 단계

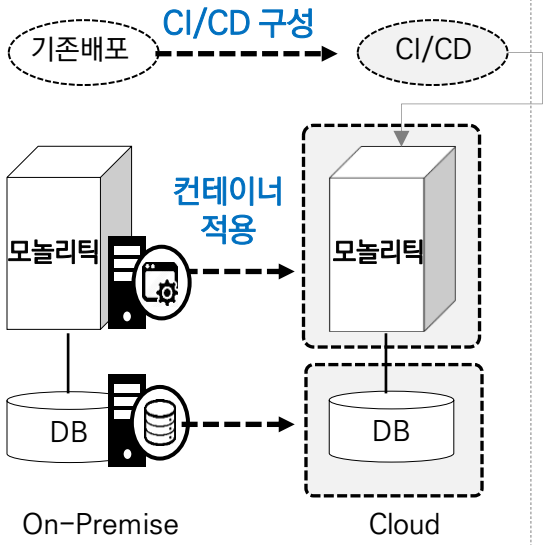
플랫폼 표준화, 비용효율



PaaS
(서버가상화–VM, x86)

Level 2 : Cloud Friendly
클라우드 친화단계

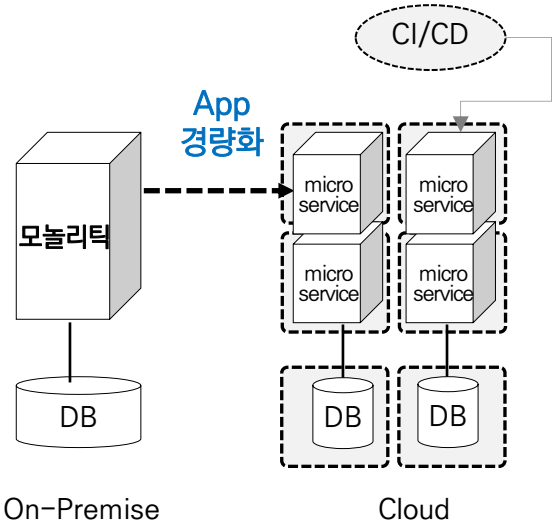
Auto-Scaling, 컨테이너 적용



PaaS + CI/CD + 12 Factors
(서버가상화–컨테이너, x86)

Level 3 : Cloud Native
클라우드 네이티브 단계

APP경량화, 업무 민첩성 대응



PaaS + CI/CD + 12 Factors + MSA
(서버가상화–컨테이너, x86)

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주자 가이드

공공 클라우드와 클라우드 네이티브의 정부정책을 추진합니다.

NIA 한국지능정보사회진흥원

디지털 정부 정책 변화 (1/2)

디지털 뉴딜, 디지털 정부혁신 및 대국민 수요변화에 능동적으로 대응하고 있습니다.

코로나19 언텍트 문화확산

디지털 정부 역할 확대

빠른 보급확산

정부정책 — 디지털 뉴딜

01 한국형 디지털 뉴딜
(DNA생태계 강화)

02 클라우드 산업발전전략
(국가클라우드 대전환)

03 클라우드컴퓨팅 발전계획
(SaaS/PaaS확산)

정부계획 — 디지털 정부혁신

01 민간·공공클라우드 전면전환
('25년 소규모전산실 중심통합)

02 클라우드 플랫폼 고도화
(디지털정부서비스개발환경적용)

03 디지털 서비스 전문계약제도 신설

행정서비스 — 대국민 수요변화

01 비대면 서비스 요구 증대

02 대국민 디지털 서비스 단기간 제공 급증

03 전자정부시스템 지원사업 SaaS기반 수출 필요

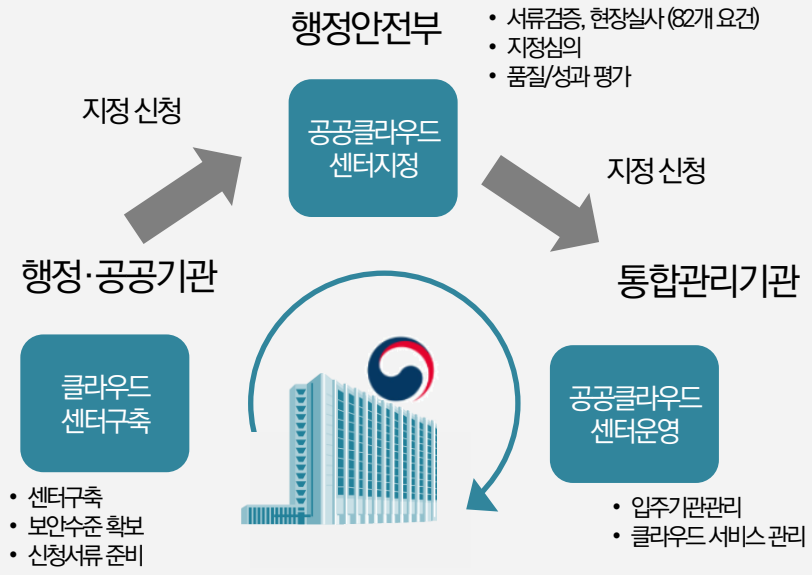
디지털 정부 정책 변화 (2/2)

'디지털 정부혁신 발전계획'과 '한국판 뉴딜 종합계획'의 일환으로 민간 클라우드 또는 공공 클라우드로 전환합니다.

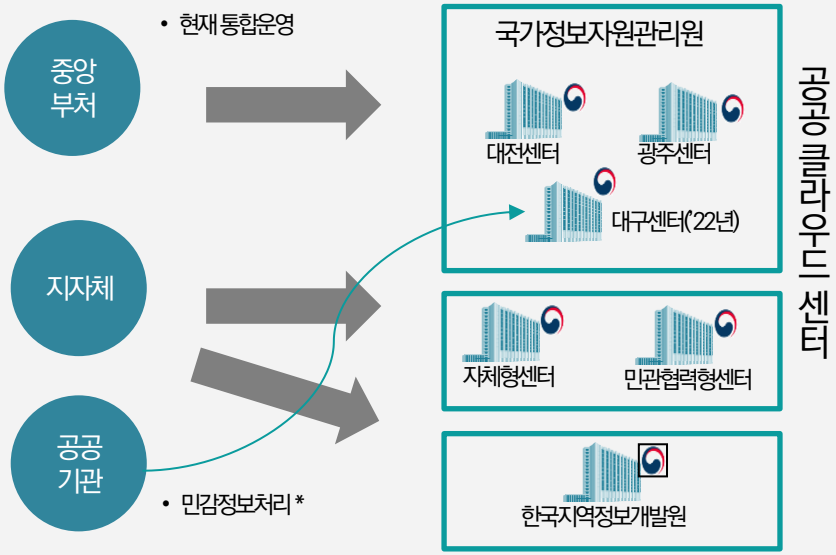
"공공 클라우드 센터"란 국가안보, 수사, 재판, 내부업무처리등 중요도가 높은 정보시스템을 통합관리.
데이터센터 중에서 행안부 장관이 지정한 데이터센터 (행정기관 및 공공기관 정보자원 통합기준)

전체 시스템 중 83%, 18.5만대, 25년까지 완료

공공 클라우드 센터 지정 및 운영



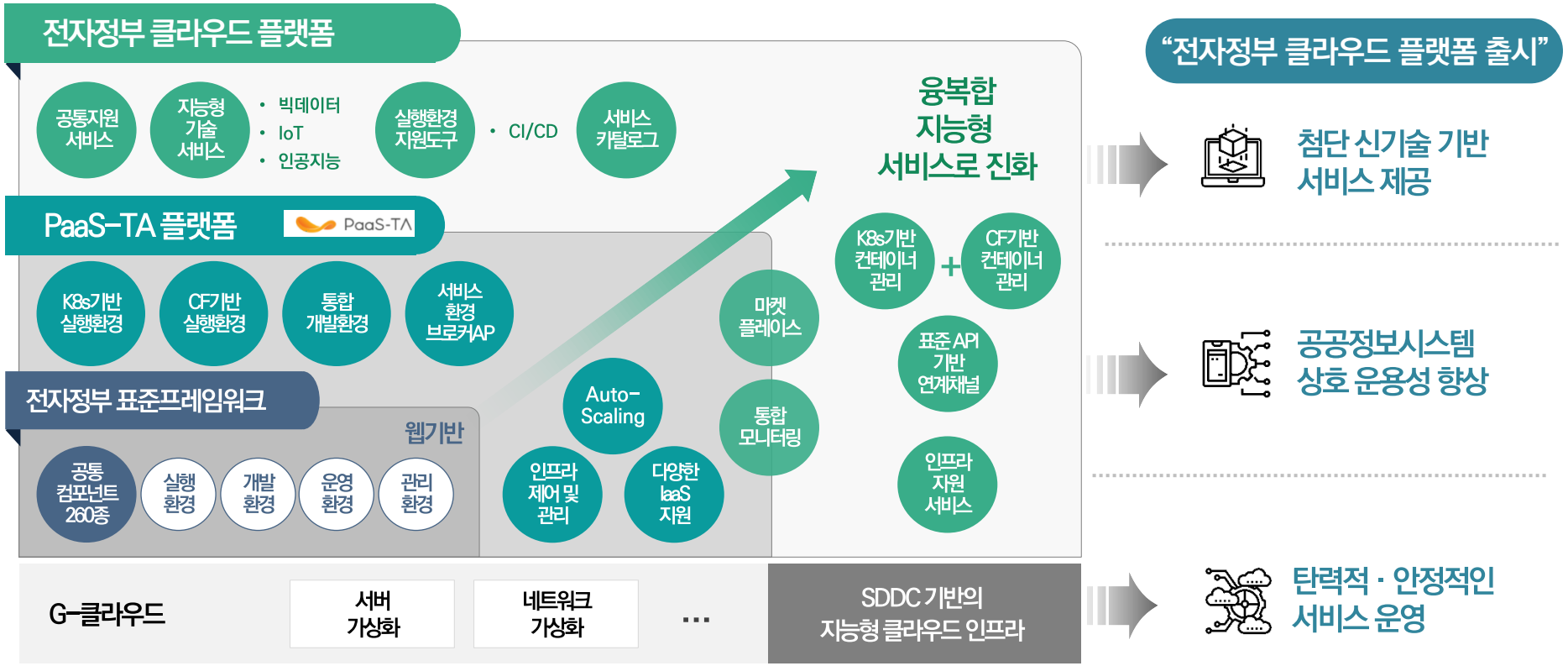
이용가능 공공클라우드센터 구분



디지털 정부 변화 (1/3)

'디지털 정부혁신 발전계획'과 '한국판 뉴딜 종합계획'의 일환으로 민간 클라우드 또는 공공 클라우드로 전환합니다.

민관 클라우드 확산 지원사업을 통해, 범 국가적인 상호 호환·운용 환경을 조성 및 개방형 클라우드 센터 설치.운영



디지털 정부 변화 (2/3)

사전 등록된 서비스를 국가기관이 필요할 때 필요한 만큼 사용할 수 있는 디지털 전문 계약 서비스 제도 신설하였습니다.

공공에서 클라우드 서비스를 도입할 때 입찰을 거치지 않고 구매할 수 있게 하는 것으로서,
정부 및 국가기관이 '디지털서비스 이용지원시스템'에 등록된 디지털 서비스를 계약할 수 있음

2019 **10. 국무회의** ----○ [디지털정부혁신계획] 디지털서비스 전문계약 제도 신설 발표

2020 **10.01** ----○ 디지털서비스 이용지원시스템(digitalmarket.co.kr)

정보 통신자원을 제공 → **클라우드 컴퓨팅 서비스**

클라우드 컴퓨팅 서비스 지원 → **클라우드 지원 서비스**

다른 기술·서비스가 융합된 서비스 → **융합 서비스**

- 매니지드 서비스
- 컨설팅 서비스
- 운영관리 서비스
- 마이그레이션 서비스

① 수의계약

② 카달로그 계약

디지털 정부 변화 (3/3)

해외는 클라우드 네이티브 전환을 위하여, 선진 정책, 컨설팅과 교육을 제공하고 있습니다.

미 국

2019년 6월 기존의
'클라우드 퍼스트'에서 새로운 전략인
'클라우드 스마트'로 전환,
효율적인 클라우드 도입 가이드 주력

영 국

2017년 2월
클라우드 퍼스트 정책을 재정립
퍼블릭 퍼스트에서 클라우드
네이티브로 전환하도록 장려

싱가포르

IMDA(정보매체개발청)의GoCloud
정책을 통해, ICT중소기업의 개발팀에게
클라우드 네이티브, MSA와 DevOps
관련 컨설팅과 교육제공

출처 : https://www.imda.gov.sg/programme-listing/gocloud

평가 도구

- 클라우드 네이티브 진단 도구를 통한 기업의 수준 진단

STEP 01 범위정의

- 클라우드 네이티브 장점 교육
- 전문기와의 협업을 통한 프로젝트 계획 및 범위 정의

STEP 02 교육훈련

- 클라우드 네이티브 애플리케이션 설계, 개발 관련 교육 및 자격증 제공

STEP 03 실습

- 컨설턴트와 코칭인력의 클라우드 네이티브 애플리케이션 전환과 클라우드 네이티브 애플리케이션 개발 지원

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주자 가이드

어떤 공공업무에 클라우드 네이티브의 적용이 가능할까요?

NIA 한국지능정보사회진흥원

클라우드 네이티브 대상 업무선정 방향 (전문가 의견)

클라우드 네이티브 업무는 학계, 업체, 정부정책을 반영하여 대상을 선정할 수 있습니다.

서비스 복잡도가 높은 시스템

– 마틴파울러(2015, 최초 용어정의) : “마이크로서비스는 복잡한 시스템에서 유용할 때 MSA전환”

명확한 경계가 가능한 시스템

– 샘뉴먼(2019, 저서) : “해당분야를 제대로 이해하지 못해 적절한 경계를 찾기 어렵다면 MSA전환 불리”

더 이상 확장할 수 없는 한계지점에 도달한 시스템

– 수잔파울러(2019, 저서) : “확장성 한계로 인해 심각한 안정 문제 발생하여, 개발생산성·효율성 저하 시 MSA 전환”

클라우드 네이티브 적용 검토 (1/4)

클라우드 네이티브는 변화가 많은 대국민 서비스 중심으로 우선 검토하여야 합니다.

정부내 전환가능 업무 식별

SRM(서비스 참조 모델)의 전환가능 업무

정부내지원 서비스 →	업무변화 少 (적용 불리)
공통기술 서비스 →	시스템 연계 多 (적용 불리)
대국민 서비스 →	업무변화 多/ 폭주성 高 (적용 유리)

기관·시스템특성을 반영한 업무선정

시스템 특성 반영 (시스템 복잡성)

규모(大) × 부서(多) × 시스템 메뉴(多)

※국가 및 기초자치 단체 226, 공공기관 338개 기관 업무대상

정부정책 특성 반영 (제도개선이 많은 업무)

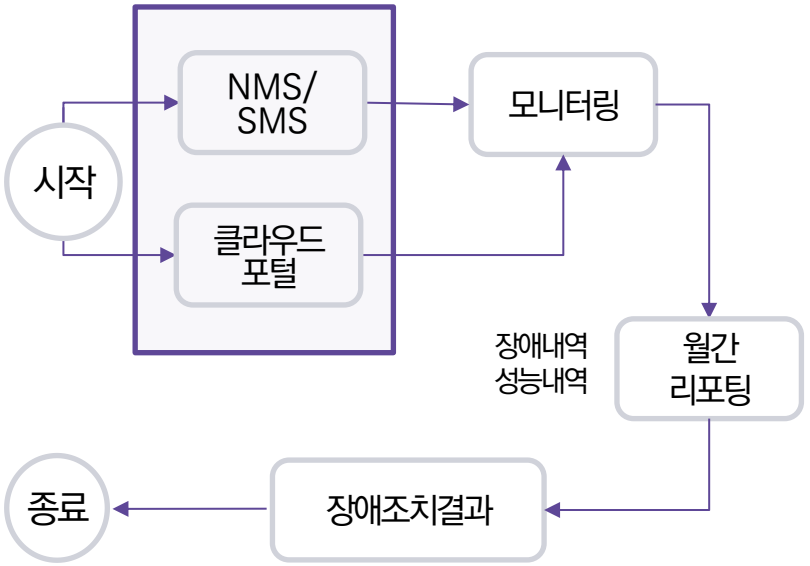
A기관	B기관	C기관	D기관

클라우드 네이티브 적용 검토 (2/4)

빈번한 장애, 폭주, 변경 및 배포가 존재하는 한계 지점에 도달한 시스템 중심으로 우선 검토하여야 합니다.

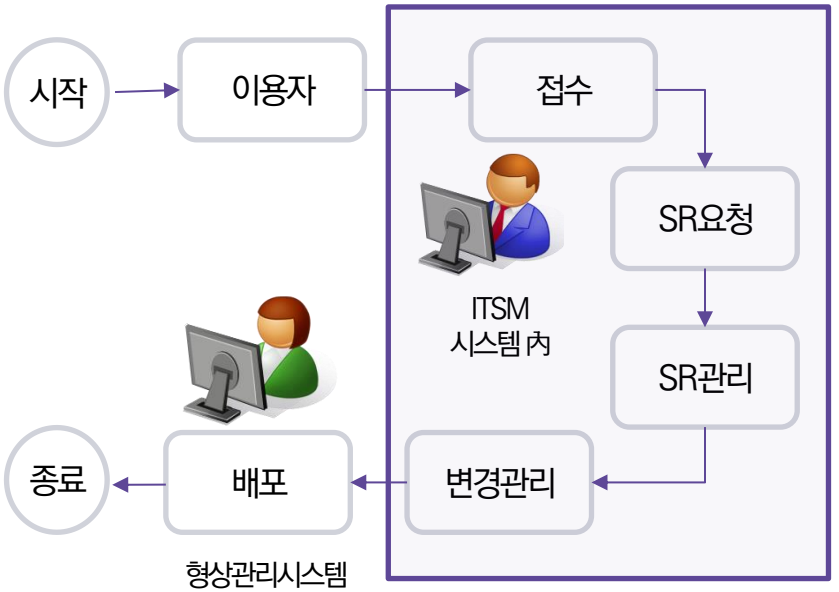
시스템 장애 및 폭주 발생 업무

시스템 장애 발생내역과 성능분석 절차



빈번한 시스템 변경과 배포 업무

SR 변경신청 및 배포횟수 식별

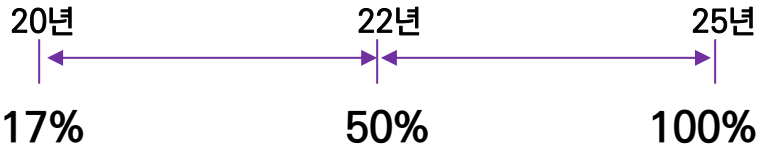


클라우드 네이티브 적용 검토 (3/4)

공공클라우드 전면 전환에 따른 클라우드 네이티브 상호 운용성 확보하여 서비스간 연결해야 합니다.

공공클라우드 전면 전환 사업진행

- 공공기관 전면전환 비율 (목표)

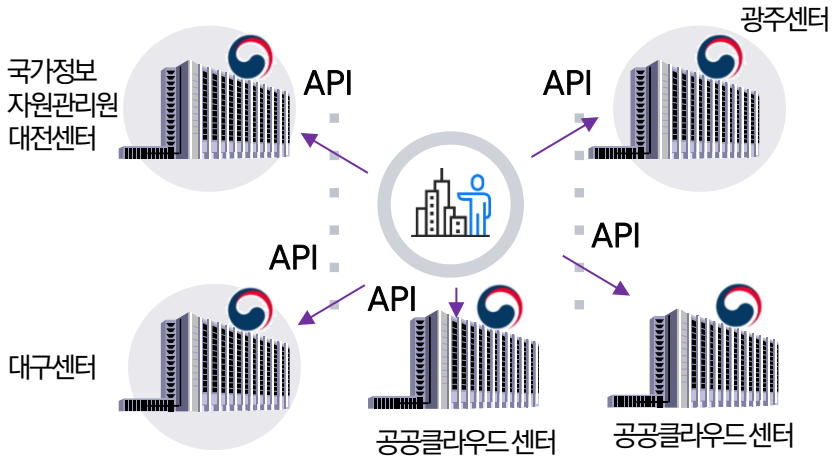


- 공공 (G-클라우드, 자체), 민간클라우드 센터

공공 클라우드센터	민간 클라우드센터
54%	46%

공공클라우드 센터 상호운용성 확보

- 디지털정부 서비스 개발환경인 클라우드 표준 플랫폼으로 고도화 필요



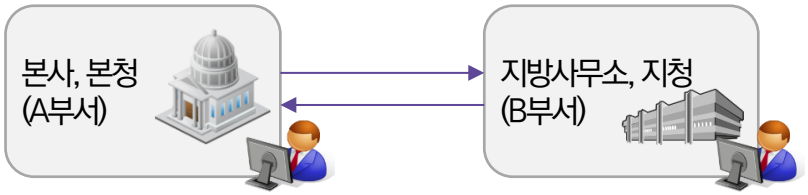
※ 마이크로 서비스 아키텍처에서는 API로 멀티센터의 서비스를 통합 제공

클라우드 네이티브 적용 검토 (4/4)

공공조직의 공통적인 특성(순환보직)을 감안하여, 담당 업무별 기능 단위로 서비스를 분리 구성하여 영향도를 최소화하여야 합니다.

공무원 인사규정 보직변경

통상적인 조직개편 및 잦은 순환보직



➔ 공무원 인사규정 :
동일 직위에서 2년 이상 근무한 경우 타 직위 또는 타 부서로 전보

내 업무 시스템만 관심

타 시스템 (多) – 관련부서 A, B, C, D ~ Z 부서

내 업무시스템(少)

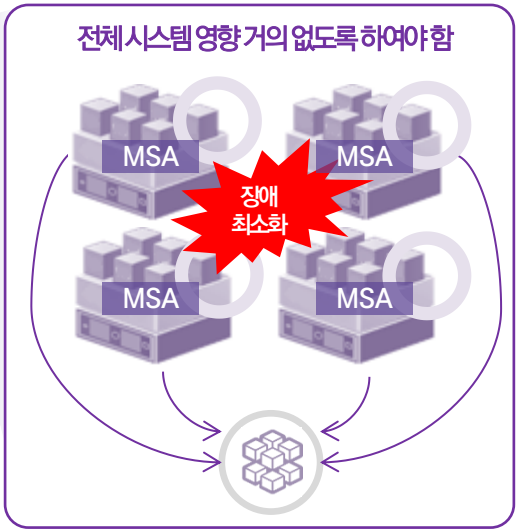
➔ 전체 시스템을 모두 이해하기 어려움. 업무담당 시스템 대비 타 시스템 관련(유관부서) 비중이 높음

전체시스템 변경협조는 차세대에서나 가능

전체 시스템 영향 없이, 내 업무 중심으로 관리하고 배포
(점차적으로 모든 부서가 동일하게 기능 및 서비스단위 분리)

내 업무시스템 변경

어떤 영향, 장애 걱정
➔
'내 프로그램 변경에 타 시스템이 영향 없도록 …'



정보시스템 자가진단 방법

클라우드 네이티브 구성요소의 특장점을 토대로, 현행 정보시스템에 대한 체크리스트를 도출 하였습니다.

정보시스템 자가진단 (1/3)

현행 정보시스템에 대한 체크리스트를 도출하였으며, 6개 이상 "Y"응답시, 클라우드 네이티브 도입이 필요한 것으로 판단할 수 있습니다.

정보시스템 자가진단 체크리스트 예시

구분(목표)		자가진단 항목	답변	
안정적 서비스 운영	1	• 초기개발비의 약 15%이상을 매년 추가개발 및 유지보수 비용으로 사용하고 있습니까?	✓	6개 이상 "YES" 응답 시 도입 검토
	2	• 다양한 원인에 의한 장애 발생 시 장애복구(예.시스템 증설, 업그레이드 등)를 위해 서비스를 중단한 적이 있습니까?	✓	
	3	• 특정시점(년, 월, 주, 시)에 트래픽이 증가로 접속지연으로 불만이 제기된 적이 있습니까?		
업무 및 기술 변화 대응	4	• 수시로 정책, 업무 요건 등의 변화에 따른 요구사항에 대해 신속한 대응이 필요합니까?		
	5	• 디지털 신기술(빅데이터, AI, 블록체인, IoT 등) 적용 및 다양한 언어 및 다양한 오픈소스에 대한 요구사항 반영이 필요합니까?	✓	
	6	• 소규모 서비스 단위로 기능과 DB의 명확한 분리가 가능하고, 독립적 단위로 실행이 가능합니까? (공통 기능 및 데이터 사용, 타 시스템과의 연계성, 서비스 의존관계 등 확인)	✓	
개발 품질 향상	7	• 시스템 개발 및 운영시 개발 및 운영 조직의 분리에 따라 의사소통, 개발 및 배포 지연 등의 문제가 존재합니까?	✓	
	8	• 소스코드의 복잡성으로 서비스 확장이 곤란하여 서비스 분리 및 소스코드 개선이 필요합니까?	✓	
개발기간	9	• 개발된 SW를 형상관리 시스템에 커밋한 후 개발계, 검증계, 운영계 서버에서 빌드, 테스트, 배포하는 과정에 빌드·테스트·배포 도구를 사용하지 않거나 부분적으로 사용하고 있습니까 ?		
	10	• 현행 시스템의 배포주기를 단축하고 싶습니까?		

정보시스템 자가진단 (2/3)

"YES" 답변이 3개 항목 이하라면, 기존 아키텍처를 유지를 권장합니다.

표준 프레임워크 포털 적용 예시

* 운영성과 지표 : 정보시스템 운영 성과측정 시 활용되는 성과지표

구분(목표)	자가진단 항목	*운영성과 지표	표준 프레임워크 포털검토결과	답변
안정적 서비스 운영	1 • 초기개발비의 약 15%이상을 매년 추가개발 및 유지보수 비용으로 사용하고 있습니까? • 단, 정보시스템(대국민서비스)의 이용자수 일평균 1만명 이상인가?	개발비(초기, 추가) 유지보수비	유지보수 담당자 : 1명	✓
	2 • 다양한 원인에 의한 장애 발생 시 장애복구(예.시스템 증설, 업그레이드 등)를 위해 서비스를 중단한 적이 있습니까?	고객증가율,장애복구소요시간	장애 발생하지 않음	✓
	3 • 특정시점(년, 월, 주, 시)에 트래픽이 증가로 접속지연으로 불만이 제기된 적이 있습니까?	신뢰성 및 가용성	개발자를 대상으로 한 정보 제공	
업무 및 기술 변화 대응	4 • 수시로 정책, 업무 요건 등의 변화에 따른 요구사항에 대해 신속한 대응이 필요합니까?	서비스요청(CSR), 적기 처리율	행안부와 NIA의 요구사항 발생 시	✓
	5 • 디지털 신기술(빅데이터, AI, 블록체인, IoT 등) 적용 및 다양한 언어 및 다양한 오픈소스에 대한 요구사항 반영이 필요합니까?	업그레이드 용이성	포털 자체는 해당사항이 없음 (표준 프레임워크 관련 R&D는 별도로 추진)	
	6 • 소규모 서비스 단위로 기능과 DB의 명확한 분리가 가능하고, 독립적 단위로 실행이 가능합니까? (공통 기능 및 데이터 사용, 타 시스템과의 연계성, 서비스 의존관계 등 확인)	업무기능, 통합성 정도, DB설계 수준	서비스 단위로 기능과 서비스 분리 가능	
개발 품질 향상	7 • 시스템 개발 및 운영시 개발 및 운영 조직의 분리에 따라 의사소통, 개발 및 배포 지연 등의 문제가 존재합니까?	유지보수 용이성	개발자 1명 (유지보수 10%, 운영 90%)	
	8 • 소스코드의 복잡성으로 서비스 확장이 곤란하여 서비스 분리 및 소스코드 개선이 필요합니까?	유지보수 용이성	취약점 분석을 통해 보완하고 있음	
개발기간	9 • 개발된 SW를 형상관리 시스템에 커밋한 후 개발계, 검증계, 운영계 서버에서 빌드, 테스트, 배포하는 과정에 빌드·테스트·배포 도구를 사용하지 않거나 부분적으로 사용하고 있습니까 ?	'운영성과 지표' 해당사항 없음	G-클라우드에서 제공하지 않음 (KT 내부 정책)	
	10 • 현행 시스템의 배포주기를 단축하고 싶습니까?	'운영성과 지표' 해당사항 없음	배포 관련 이슈 없음	

3개 이하 기존 유지 권장

정보시스템 자가진단 (3/3)

안정적 서비스운영, 업무·기술변화대응, 개발품질 향상, 개발기간 단축이라는 목표를 달성할 수 있습니다.

공용홈쇼핑 적용 예시

* 운영성과 지표 : 정보시스템 운영 성과측정 시 활용되는 성과지표

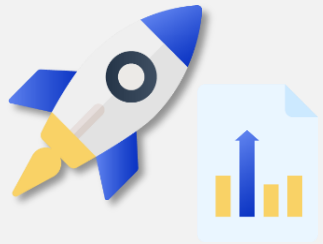
구분(목표)	자가진단 항목	*운영성과 지표	공용홈쇼핑 검토결과	답변
안정적 서비스 운영	1 • 초기개발비의 약 15%이상을 매년 추가개발 및 유지보수 비용으로 사용하고 있습니까? • 단, 정보시스템(대국민서비스)의 이용자수 일평균 1만명 이상인가?	개발비(초기, 추가) 유지보수비	개발비(초기, 추가) 유지보수비 → 15%이상으로 추정	YES
	2 • 다양한 원인에 의한 장애 발생 시 장애복구(예.시스템 증설, 업그레이드 등)를 위해 서비스를 중단한 적이 있습니까?	고객증가율,장애복구소요시간	가용성 99.6%수준으로 운영	NO
	3 • 특정시점(년, 월, 주, 시)에 트래픽이 증가로 접속지연으로 불만이 제기된 적이 있습니까?	신뢰성 및 가용성	없음	NO
업무 및 기술 변화 대응	4 • 수시로 정책, 업무 요건 등의 변화에 따른 요구사항에 대해 신속한 대응이 필요합니까?	서비스요청(CSR), 적기 처리율	홈쇼핑 특성 상 필요 (11번가 등과 유사)	YES
	5 • 디지털 신기술(빅데이터, AI, 블록체인, IoT 등) 적용 및 다양한 언어 및 다양한 오픈소스에 대한 요구사항 반영이 필요합니까?	업그레이드 용이성	계획 없음	NO
	6 • 소규모 서비스 단위로 기능과 DB의 명확한 분리가 가능하고, 독립적 단위로 실행이 가능합니까? (공통 기능 및 데이터 사용, 타 시스템과의 연계성, 서비스 의존관계 등 확인)	업무기능, 통합성 정도, DB설계 수준	서비스별 기능 및 DB 분리 가능	YES
개발 품질 향상	7 • 시스템 개발 및 운영시 개발 및 운영 조직의 분리에 따라 의사소통, 개발 및 배포 지연 등의 문제가 존재합니까?	유지보수 용이성	정기/수시 배포는 단계별 승인 후 진행	NO
	8 • 소스코드의 복잡성으로 서비스 확장이 곤란하여 서비스 분리 및 소스코드 개선이 필요합니까?	유지보수 용이성	2015년 개발되어 7년째 운영 중. 개선필요	YES
개발기간	9 • 개발된 SW를 형상관리 시스템에 커밋한 후 개발계, 검증계, 운영계 서버에서 빌드, 테스트, 배포하는 과정에 빌드·테스트·배포 도구를 사용하지 않거나 부분적으로 사용하고 있습니까 ?	'운영성과 지표' 해당사항 없음	빌드, 배포 도구 일부 사용	YES
	10 • 현행 시스템의 배포주기를 단축하고 싶습니까?	'운영성과 지표' 해당사항 없음	판매상품의 변화에 따라 신속한 배포 필요	YES

클라우드 네이티브 업무 도입 검토사항

클라우드 네이티브를 도입시 사전 고려사항을 검토합니다.

장기적 비즈니스 목적
(명확한 기능경계, 시스템 확장)

MSA에 필요한 기술 보유

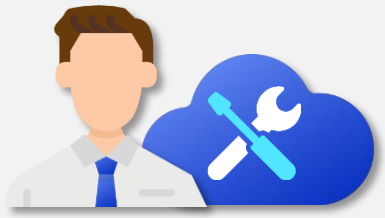


비즈니스 도달까지 충분한 시간
(다양한 신기술 적용을 위한 충분한 개발기간)

요구분석, 설계, 개발 테스트
출시를 반복적으로 수행

서비스 조직구성과 Agile 문화
(개발~운영을 합친 계약방식 도입 검토)

비즈니스 규모에 맞는
충분한 인력 사전 배치



개발자의 숙련도 및 도구 선정
(기술적 숙련도가 높은 개발자의 참여)

클라우드 네이티브 애플리케이션(MSA) 도입 조건

1. 빠르고 잦은 배포가 필요한 가 ?

도입 조건 1

MSA의 근본적인 목적은 빠른 비즈니스 대응, 민첩성에 있음
얼마나 사용자의 요구사항을 빠르게 반영하고 개선해 나갈 수 있을지에 대해 초점이 맞추어져 있음
자주 개선해야 되거나 배포해야 되는 시스템(서비스)이 아닌 경우는 MSA가 적합하지 않을 수 있음

2. 성능이 중요한 서비스 인가 ?

도입 조건 2

분리된 서비스들은 물리적인 네트워크 통신을 하게 되므로 네트워크 대기시간(Latency)이 발생할 수 있음
모놀리스에 비해 성능이 저하될 수 있으므로 성능에 민감한 시스템(서비스)인 경우 도입을 고려해야 함

3. 분산 트랜잭션이 필요한 서비스 인가 ?

도입 조건 3

모놀리스에서 여러개의 서비스들을 묶어 단일트랜잭션으로 처리할 수 있었다면, MSA에서는 단일 트랜젝션 처리가 어려움
이벤트 기반의 트랜젝션을 구현하여 데이터의 정합성을 유지할 수 있음

4. 모놀리틱에 비해 비용을 절감할 수 있는가 ? (비용대비 이득을 따져 볼 것)

도입 조건 4

서비스들이 늘어난다는 것은 관리 할 서버가 늘어난다는 것을 의미함. 그러므로 운영관리 비용이 증가할 수 있음
하지만 MSA가 주는 장점의 이득이 더 크다고 판단될 경우, MSA를 도입하는 것이 좋은 선택이 될 수 있음

5. 조직의 개발 문화는 준비되어 있는가 ?

도입 조건 5

단순히 아키텍처만 도입한다고 해서 MSA가 구현되는 것은 아니며, MSA에 적합한 조직의 개발문화(DevOps)를
정착시킬수 있도록 노력해야 함

정보시스템 운영 성과측정 지표 – 현행 정보시스템 지표 분석 (1/2)

정보시스템 운영 2차 성과측정 시 현행 정보시스템 분석 관련 지표도 클라우드 네이티브 도입 자가진단 체크리스트에서 활용할 수 있음

출처 : 정보시스템 운영 성과측정 매뉴얼, 2017, 행정안전부

업무수행 영향도	구현된 전체 기능수	미활용 기능수	저활용 기능수
사용 편의성	**편의성 조사 결과***	* 별도의 편의성 조사 결과를 반영	
업무성과 달성도	사전협의 수행률	업무처리 오류율	웹표준화 준수율
	클라우드화 지수	업무처리시간 개선율	정보시스템 구축 진척도
	사용자만족도	고객증가율	업무처리자동화율
	서비스요청 (CSR)적기 처리율	시스템 장애 발생 시간	시스템 응답속도
	시스템 가동률	장애복구소요시간	서비스 이용 증가율
	페이지뷰 수 증가율	정보공개 서비스 증가율	보안 관련 지적 및 보안사고 발생 건수
	보안 조치 이행률	사용자 정보화 교육시간	

정보시스템 운영 성과측정 지표 – 현행 정보시스템 지표 분석 (2/2)

정보시스템 운영 2차 성과측정 시 현행 정보시스템 분석 관련 지표도 클라우드 네이티브 도입 자가진단 체크리스트에서 활용할 수 있음

출처 : 정보시스템 운영 성과측정 매뉴얼, 2017, 행정안전부

애플리케이션	업무기능	성능	가용성
	응답시간	처리빈도	정확도
	시스템 보안	통합성 정도	분산처리
	유지보수 용이성*	유지보수 비용	

*** 유지보수 용이성**
- 운영자 개선 요구 빈도 및 관리절차
- 설계 및 사용언어의 모듈화 수준
- 시스템 사용연수, 시스템 수정 빈도 등

HW 및 사용SW	신뢰성 및 가용성	지원 기능 및 호환성	처리용량
	업그레이드 용이성	**유지보수 용이성****	유지보수 비용
	기술지원		

**** 유지보수 용이성**
- 운영자수 및 운영자 개입 정도
- 운영자의 기술 숙련 요구수준
- 유지보수 유형(일상 유지보수, 비상 유지보수)

데이터베이스	DB설계 수준	접근 용이성	통합성 정도
	백업/시스템복구/통제능력	**유지보수 용이성*****	

***** 유지보수 용이성**
- 데이터베이스 변경 이력 및 절차

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주자 가이드

클라우드 네이티브 도입 고려사항은 어떻게 되나요?

NIA 한국지능정보사회진흥원

정보화 사업관리 프로세스

공공기관에서 추진하는 정보화 사업관리는 기획·검토, 계획수립, 사업자 선정계약, 개발구축, 감리, 검사종료 단계에 따라 진행됩니다.

기획·검토

기획·검토
- 정보화사업 기획
- 정보화사업 검토

계획수립

사업계획서(안) 작성
- 사업계획서(안) 작성
- 기술적용계획 수립

사업계획서(안) 검토
- 기술평가 시행
- 보안성 검토

사업계획서(안)
- 사업계획서(안) 수정·확정

사업자 선정·계약

제안요청
- 제안요청서 작성
- 제안안내서 작성
- 입찰공고
- 제안요청설명회 개최(필요시)
- 입찰마감

사업자 선정 및 계약 체결
- 제안서 평가
- 협상 및 낙찰자 선정
- 제안서 보상
- 계약서류 준비 및 계약체결

개발·구축

사업착수
- 착수계획서 검토

진도/품질관리
- 진도/품질관리

변경관리
- 과업변경

예산관리
- 선금 지급
- 기성대가 지급

보안관리
- 준비단계 보안
- 수행단계 보안
- 종료단계 보안
- 보안적합성 검증필 정보시스템 도입
- 개인정보보호

하도급관리
- 하도급관리

감리

감리법인 선정
- 감리법인 선정

감리시행
- 예비조사 실시
- 감리계획 접수
- 감리시행
- 감리결과 조치
- 감리결과 조치 확인

감리 사업관리
- 감리 변경관리
- 감리 보안관리

감리 완료
- 검사 및 대가지급

검사종료

완료검사
- 완료검사

인수 및 하자보수
- 산출물 인수
- 하자보수 내용 확정

사업종료
- 지체상금 산정 및 정산(필요시)
- 잔금대가 지급

클라우드 네이티브 추진 관련 사항 검토 대상

출처 : 행정기관을 위한 정보화사업 단계별 관리점검 가이드 V2.0, 행정안전부 & NIA

사업발주를 위한 사업관리 단계별 고려사항 (1/2)

기획·검토, 계획수립, 사업자선정·계약, 개발·구축, 감리, 검사 종료, 사업자 선정·계약 이슈입니다.

발주 단계			클라우드 네이티브 사업단계별 고려사항
기획·검토	기획·검토	정보화사업 기획	• 클라우드 네이티브 시스템 구축 관련 성과계획 수립
계획수립	사업계획서(안) 작성	사업계획서(안) 작성	• 클라우드 네이티브 관련 요구사항 상세화 • 클라우드 네이티브 도입을 고려한 개발비 예산 산정
		기술적용계획 수립	• 클라우드 네이티브 애플리케이션 관련 기술적용계획 수립
	사업계획서(안) 검토	기술평가 시행	• 클라우드 네이티브 관련 요구사항 상세화
사업자 선정·계약	제안 요청	제안요청서 작성	• 클라우드 네이티브 관련 요구사항 상세화 • 클라우드 네이티브 도입을 고려한 개발비 예산 반영

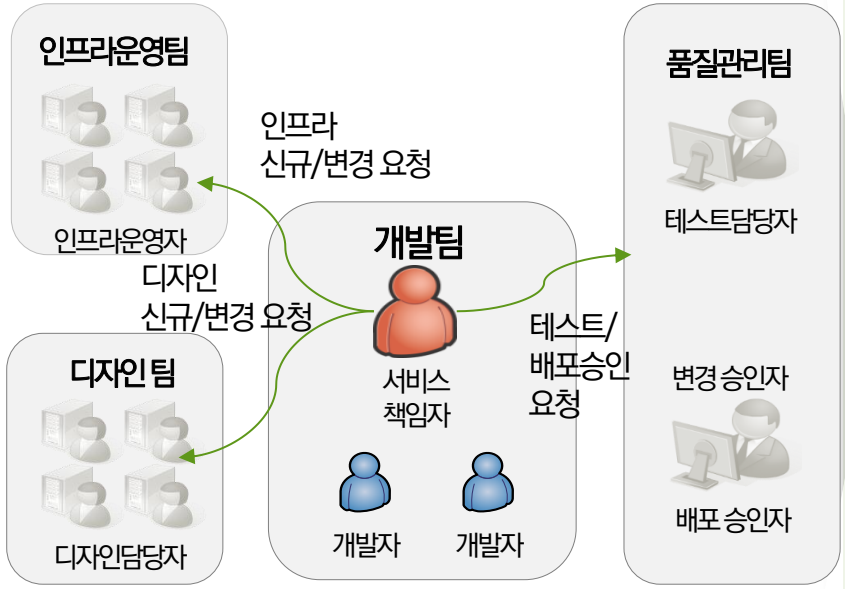
사업발주를 위한 사업관리 단계별 고려사항 (2/2)

발주 단계			클라우드 네이티브 사업단계별 고려사항
사업자 선정·계약	사업자 선정 및 계약 체결	제안서 평가	• 기술평가 수행 시 **클라우드 네이티브 적용 경험 및 기술력을 평가할 수 있는 항목 정의 및 배점** 부여
		협상 및 낙찰자 선정	• 기술협상 시 **클라우드 네이티브 관련 기술적 이행 사항에 대해 구체적으로 협상** 수행
개발·구축	사업착수	착수계획서 검토	• **착수계획서 내**에 클라우드 네이티브 관련 과업 내용이 누락 없이 잘 반영되어 있는지 확인
	진도/품질관리	진도/품질관리	• **방법론의 공정단계별 품질관리 활동 수행** – 개발방법론의 공정단계별 품질관리 활동 수행 – **클라우드 네이티브 관련 요구사항별 이행 과정 점검**
감리	감리시행	감리시행	• **감리계획서 및 수행결과보고서 검토 및 조치** – 감리계획서 및 수행결과보고서 내에 클라우드 네이티브 관련 요구사항에 대한 점검계획과 결과 검토 후 조치 요구
검사종료	완료검사	완료검사	• **완료검사 수행 시 클라우드 네이티브 관련 요구사항의 이행 여부 확인**

클라우드 네이티브 조직 변화

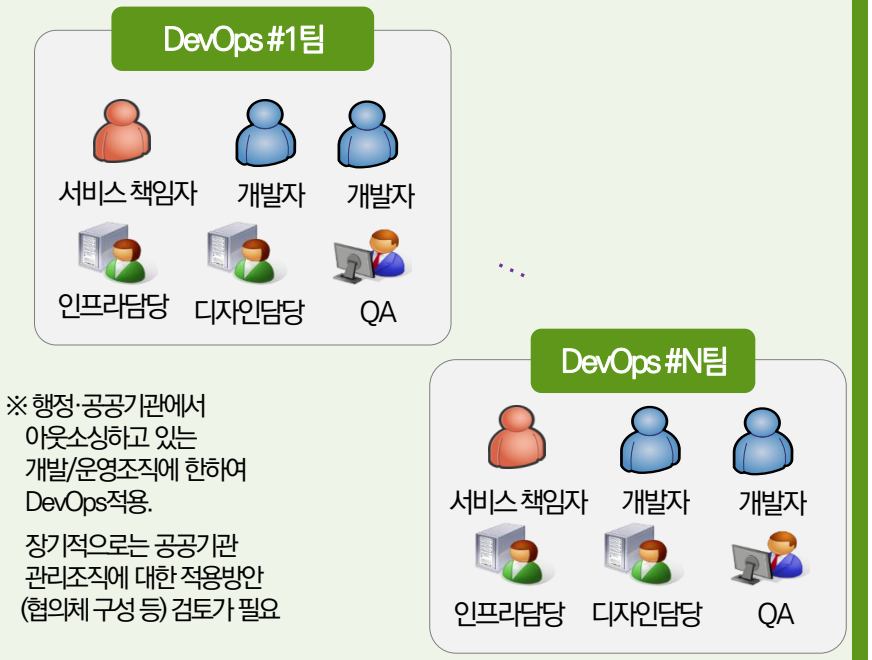
인프라·플랫폼/디자인/개발/운영 비즈니스에 민첩한 대응이 가능한 조직으로 변화해야 합니다.

일반적 개발·운영 방식



개발팀이 인프라/디자인/품질관리팀에 연락하여 신규·변경을 요청

마이크로서비스 제공을 위한 개발·운영 방식



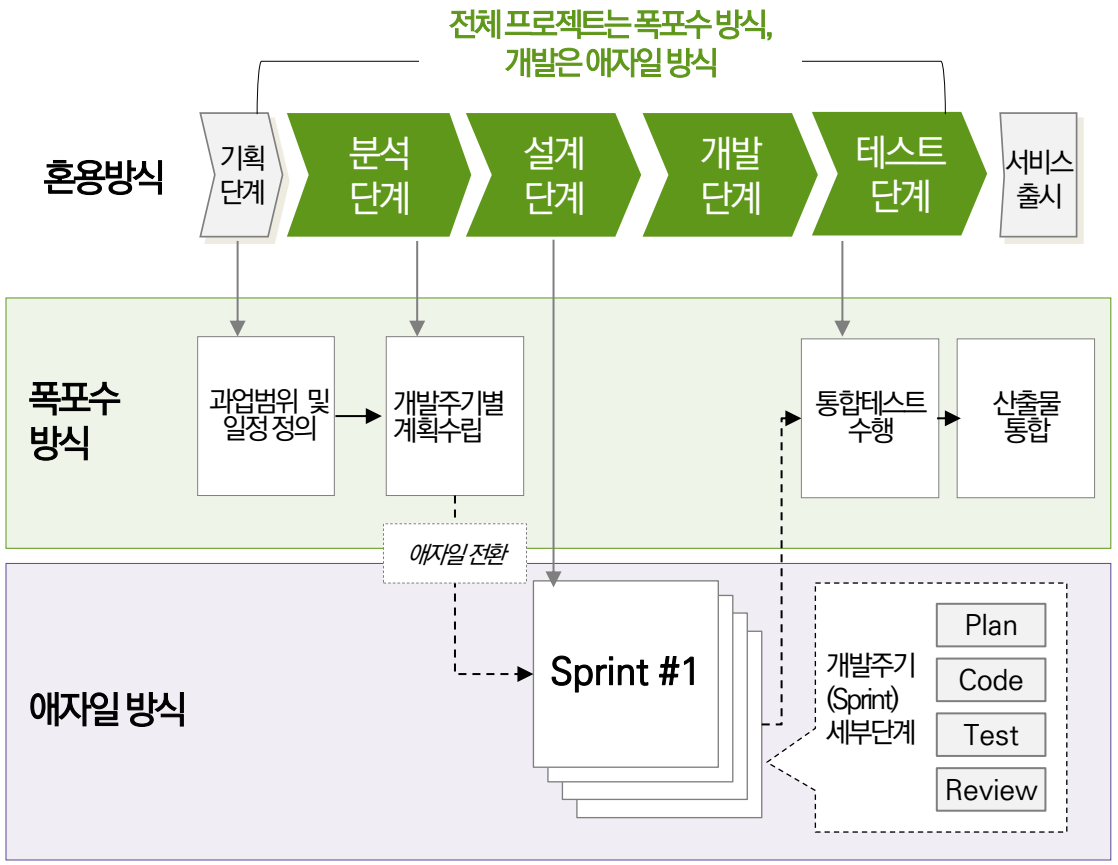
※ 행정·공공기관에서 아웃소싱하고 있는 개발/운영조직에 한하여 DevOps적용.

장기적으로는 공공기관 관리조직에 대한 적용방안 (협의체 구성 등) 검토가 필요

서비스 단위로 DevOps조직을 운영하고 운영환경에 직접배포

클라우드 네이티브 환경조성을 위한 애자일 방법론 적용

공공부문 애자일 적용은 현실적으로 애자일 방법론을 폭포수 방법론의 보완적인 역할로 폭포수-애자일 혼용방식(하이브리드)의 적용이 합리적입니다.



계약방식	• 확정된 예산금액 만큼 고정가 계약 • 과업범위, 수행인력 및 계약기간 명시
과업범위	• 요구사항 및 품질기준을 최대한 확정 후 프로젝트 착수
개발방식	• 개략적인 개발일정으로 프로젝트 착수, 개발주기(Sprint)별 상세계획에 따라 설계,개발, 테스트 및 평가 수행
결과물 (산출물)	• 개발주기(Sprint)별 산출물 분할 작성 후 프로젝트 종료시점까지 산출물 통합 및 검수 진행
수행기간	• 계약서상의 도급계약 기간 준수

※ 국내 공공부문의 SW 조달체계는 기본적으로 과업범위(요구사항 확정)와 예산이 확정된 고정가 계약을 전제로 하고 있어, 계약에 명시된 목적물(시스템 및 관련 산출물)을 정해진 기간 안에 제공.
국외 공공부문의 경우 애자일을 도입하는 사례가 점점 증가하고 있음

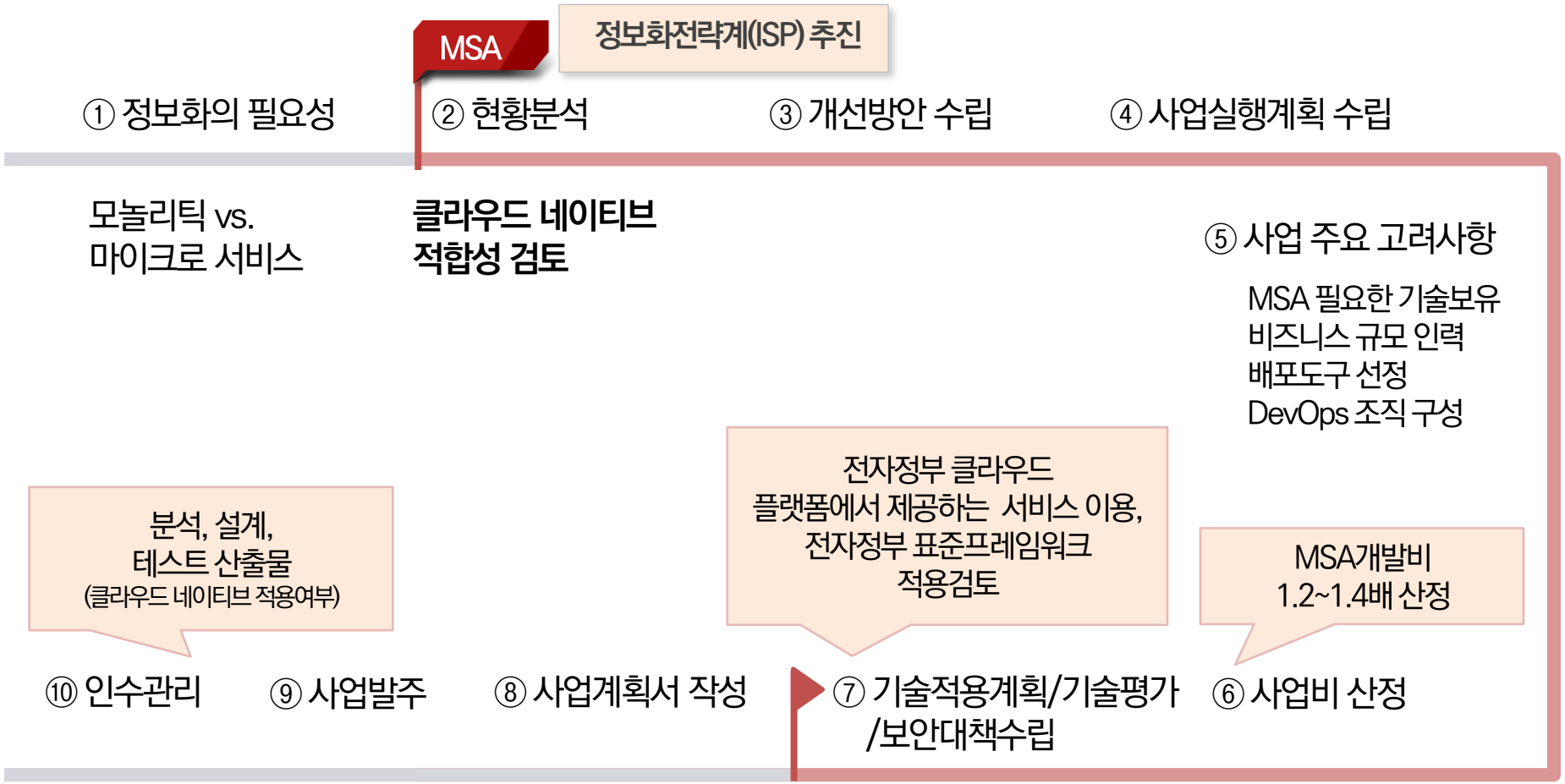
클라우드 네이티브 기반 행정·공공 서비스 확산 지원
클라우드 네이티브 발주자 가이드

클라우드 네이티브 도입 절차는 어떻게 되나요?

NIA 한국지능정보사회진흥원

클라우드 네이티브 프로세스

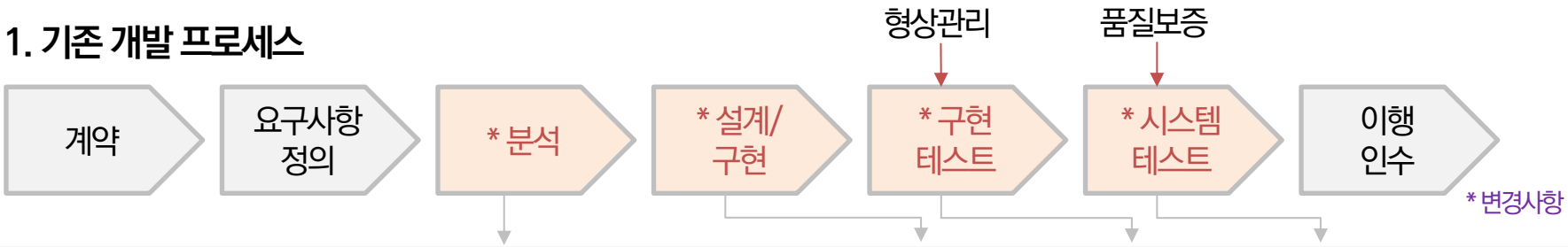
행정기관을 위한 정보화 사업 단계별 관리 및 점검가이드를 참조하여 수립한
클라우드 네이티브 발주자(기획자) 프로세스입니다.



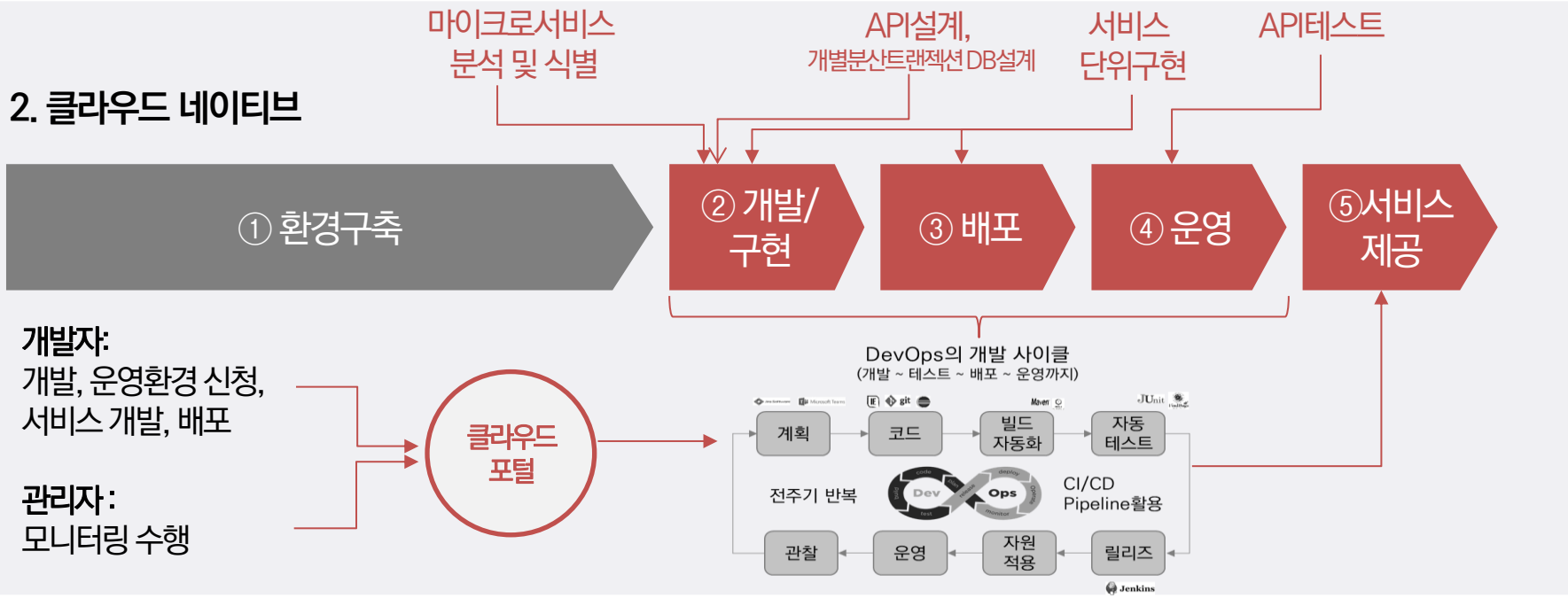
클라우드 네이티브 프로세스

행정기관을 위한 정보화 사업 단계별 관리 및 점검가이드를 참조하여 수립한 클라우드 네이티브 개발자 프로세스입니다.

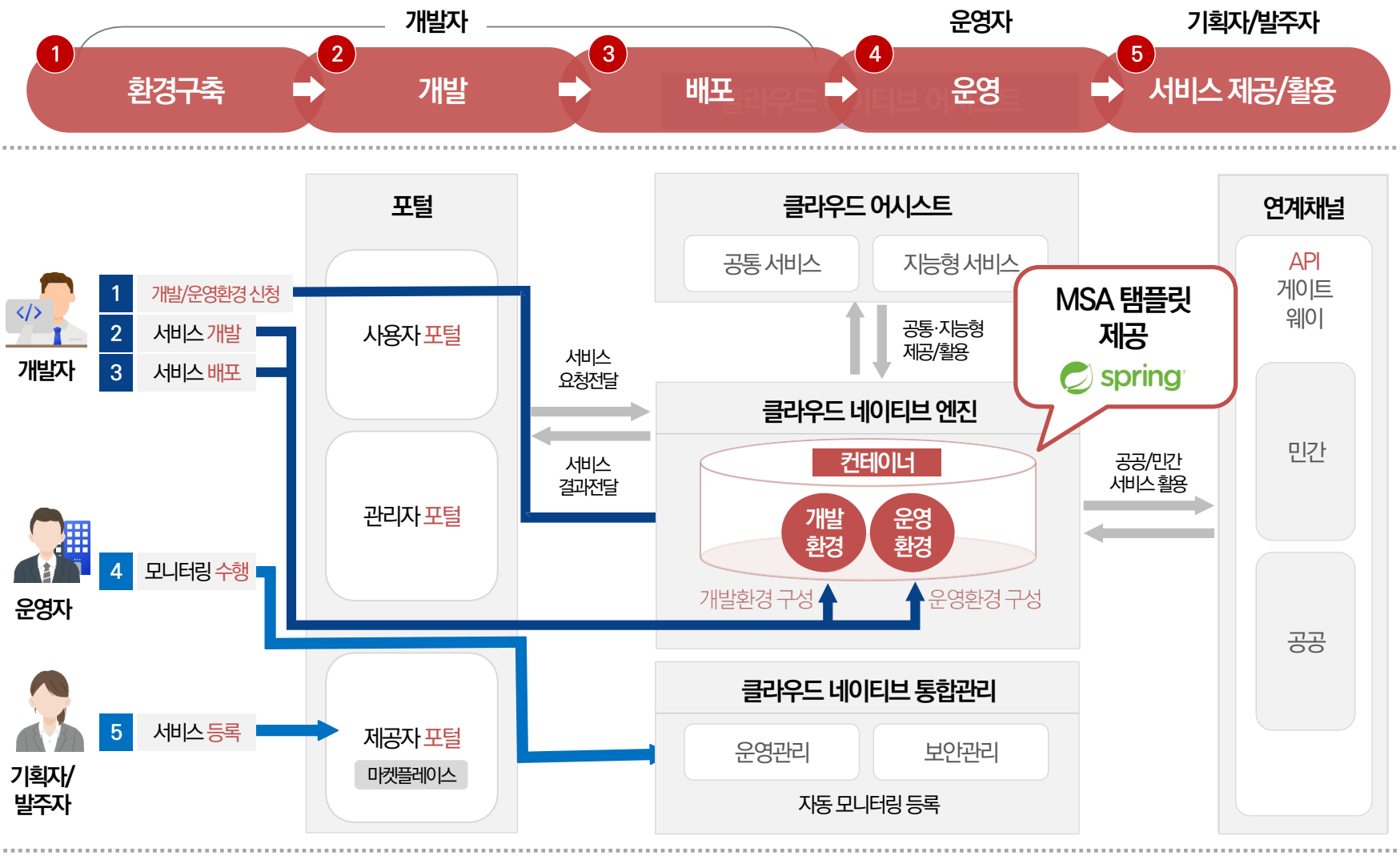
1. 기존 개발 프로세스



2. 클라우드 네이티브

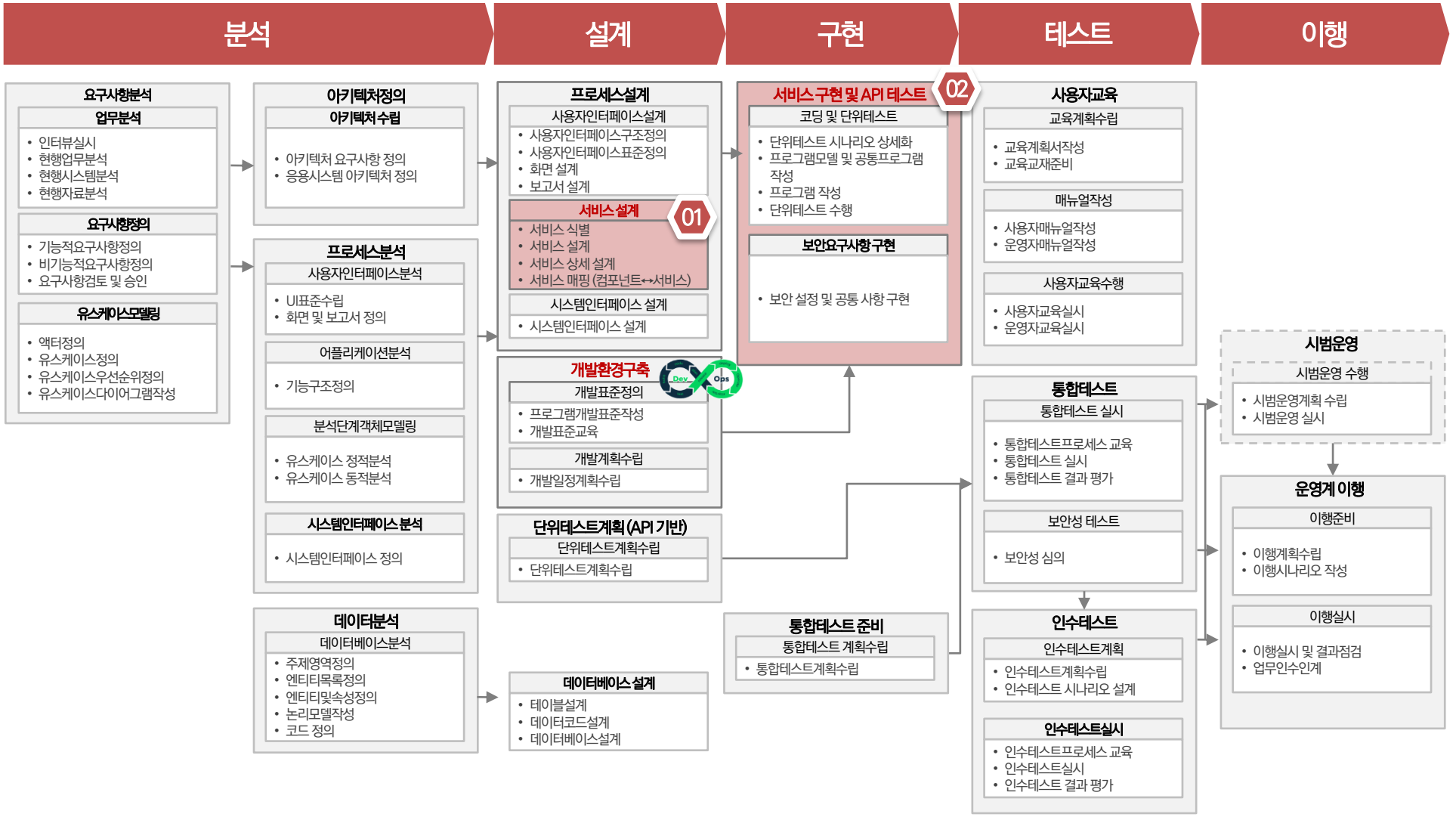


전자정부 클라우드 플랫폼 기반 클라우드 네이티브 애플리케이션 개발 절차



클라우드 네이티브 개발 절차 적용예시 (1/2)

CBD방법론을 기반한 마이크로 서비스 개발을 순차적으로 적용합니다.



클라우드 네이티브 개발 절차 적용예시 (2/2)

애자일 방법론을 중심으로 집중 적용합니다.

분석 → 설계 → 구현 → 테스트 → 이행

Sprint 실행 (설계 ~ 구현 ~ 테스트)

Life Cycle(WBS) vs. Sprint 실행 (1~N)

분석

요구사항분석

업무분석
- 인터뷰실시
- 현행업무분석
- 현행시스템분석
- 현행자료분석

요구사항정의
- 기능적요구사항정의
- 비기능적요구사항정의
- 요구사항검토 및 승인

이벤트 스토밍 01
- Scrum팀 구성
- 이벤트 스토밍의 준비사항
- 워크숍 방식 정의
- 포스트-잇 유형별 의미
- 요구사항 백로그 도출
- 도메인 이벤트 도출
- 커멘드 도출
- 앤티티(어그리게잇) 도출
- 컨텍스트 경계 그리기
- 컨텍스트 매핑하기

대상업무

설계

산출물 축소

아키텍처정의

아키텍처 수립
- 아키텍처 요구사항 정의
  - -서비스 분해지침
  - - API 설계원칙
  - - 표준 및 명명규칙
- 응용시스템 아키텍처 정의
- 인프라 구조 정의
  - -서비스 메쉬, G/W, CI/CD

02

마이크로서비스입력물정리 (API, DATA, Event, Interface)

산출물 축소

프로세스분석

사용자인터페이스분석
- UI표준수립
- 화면 및 보고서 정의

어플리케이션분석
- 기능구조정의

시스템인터페이스 분석
- 시스템인터페이스 정의

산출물 축소

데이터분석

데이터베이스분석
- 주제영역정의
- 엔티티 목록정의
- 엔티티 및 속성정의
- 논리모델작성, 코드 정의
- 분산트랜젝션 처리방안

산출물 축소

프로세스설계

Front-End 사용자인터페이스설계
- 사용자인터페이스구조정의
- 사용자인터페이스표준정의
- 화면 설계
- 보고서 설계

Back-End 서비스설계 03
- 서비스 식별
- 서비스 설계 (컴포넌트 X)
- 서비스 상세 설계 (API설계, 도메인 모델링)

Front-End 시스템인터페이스설계
- 시스템인터페이스 설계

개발환경구축 (Dev Ops)

개발표준정의
- 프로그램개발표준작성
- 개발표준교육

개발계획수립
- 개발일정계획수립

단위테스트계획 (API기반)

단위테스트계획수립
- 단위테스트계획수립

개별 분산 데이터베이스 설계 04
- 테이블설계
- 데이터코드설계
- 데이터베이스설계

구현

서비스 구현 및 API 테스트 05

코딩 및 단위테스트
- 단위테스트 시나리오 상세화
- 프로그램모델 및 공통프로그램 작성
- 프로그램 작성
- 단위테스트 수행

보안요구사항 구현
- 보안 설정 및 공통 사항 구현

통합테스트 준비

통합테스트계획수립
- 통합테스트계획수립

테스트

사용자교육

교육계획수립
- 교육계획서작성
- 교육교재준비

매뉴얼작성
- 사용자매뉴얼작성
- 운영자매뉴얼작성

사용자교육수행
- 사용자교육실시
- 운영자교육실시

통합테스트

통합테스트 실시
- 통합테스트프로세스 교육
- 통합테스트 실시
- 통합테스트 결과 평가

보안성 테스트
- 보안성 심의

인수테스트

인수테스트계획
- 인수테스트계획수립
- 인수테스트 시나리오 설계

인수테스트실시
- 인수테스트프로세스 교육
- 인수테스트실시
- 인수테스트 결과 평가

이행

시범운영

시범운영 수행
- 시범운영계획 수립
- 시범운영 실시

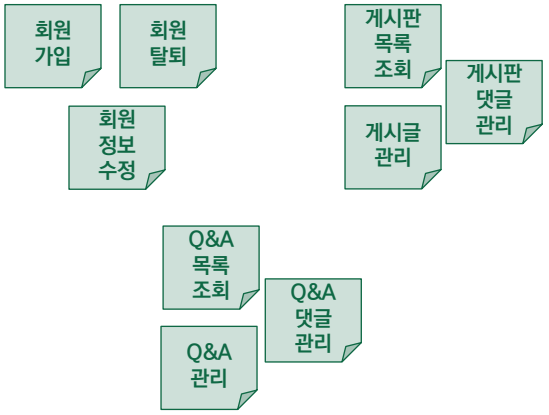
운영계 이행

이행준비
- 이행계획수립
- 이행시나리오 작성

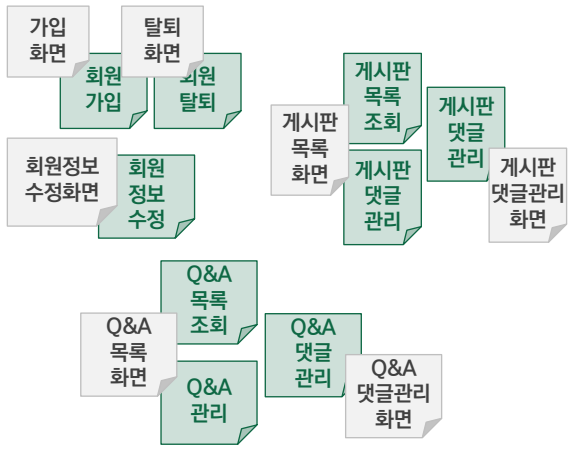
이행실시
- 이행실시 및 결과점검
- 업무인수인계

클라우드 네이티브 전환을 위한 마이크로 서비스 식별 결과 예시

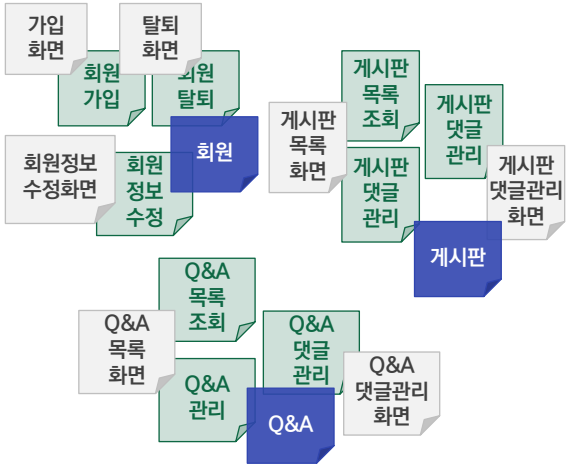
① 이벤트 도출



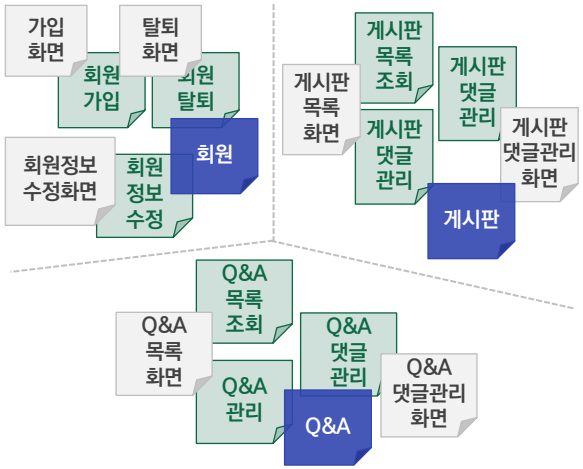
② 커멘드 도출



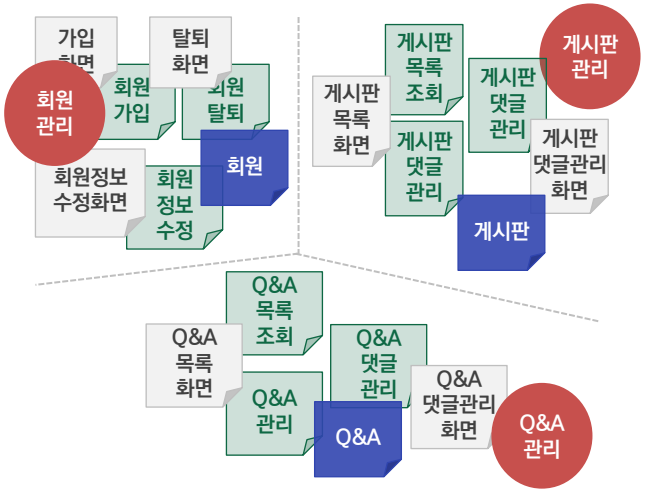
③ 엔티티(에그리게잇) 도출



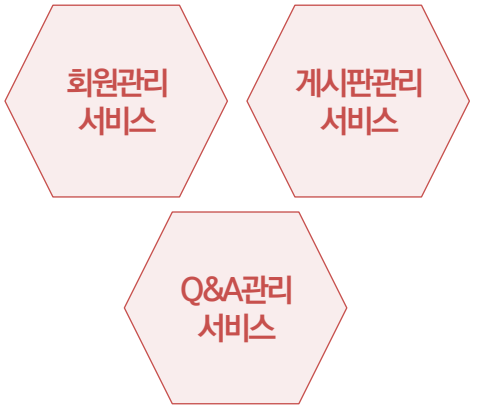
④ 컨텍스트 경계 그리기



⑤ 마이크로 서비스 식별(컨테스트 매핑)



⑥ 마이크로 서비스 식별 결과



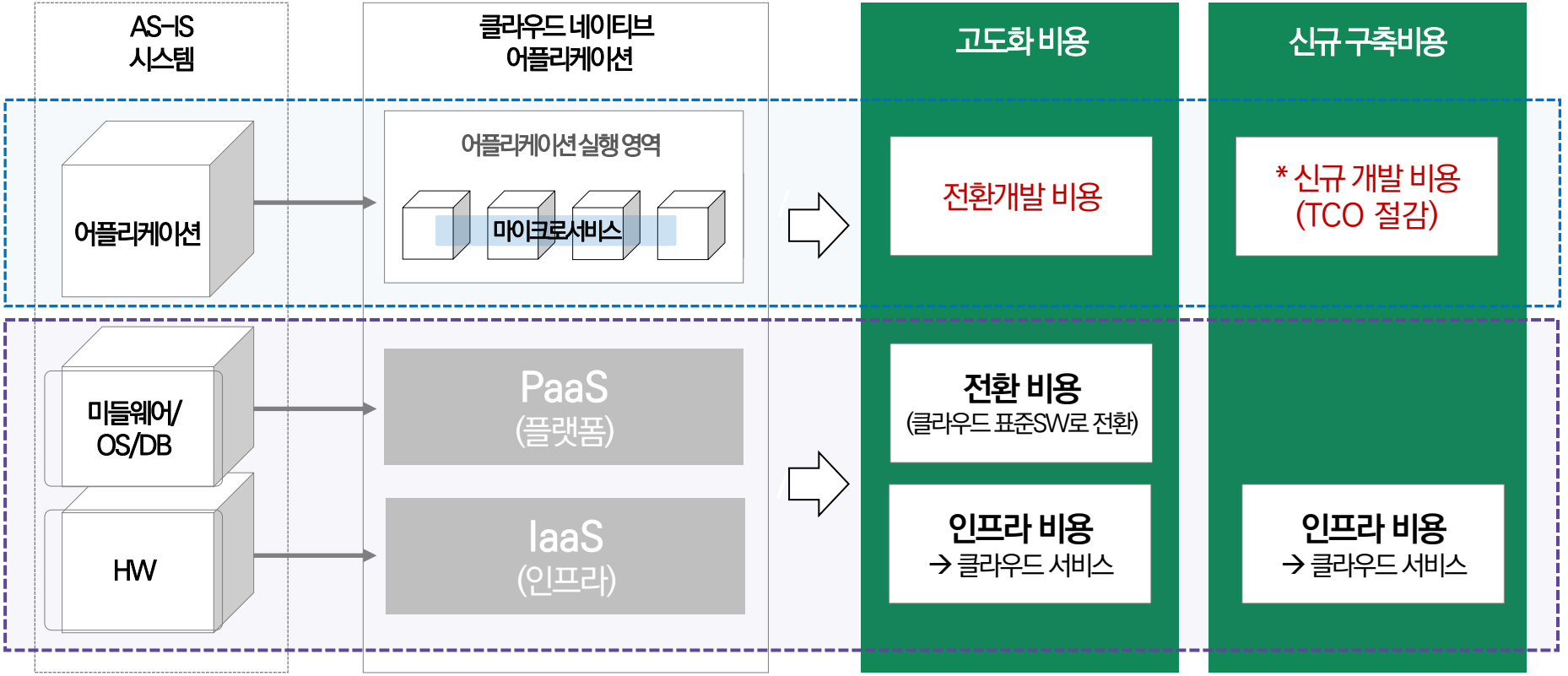
클라우드 네이티브 기반 행정·공공 서비스 확산 지원
클라우드 네이티브 발주자 가이드

클라우드 네이티브 도입 비용산정은 어떻게 해야 하나요?

NIA 한국지능정보사회진흥원

클라우드 네이티브 비용 산정 개요

클라우드 네이티브 어플리케이션 전환 및 신규 구축을 위한 예산을 수립한 후 발주절차를 진행합니다.



* 신규 구축 개발비용(용역비용) = 기존과 유사하게 정보시스템의 기능점수(FP)를 집계하는 방식으로 비용을 산정하되 클라우드 네이티브 어플리케이션 유형을 반영하도록 보정계수의 조정이나 기능점수(FP)의 추가

클라우드 네이티브 애플리케이션 개발 비용의 포지셔닝

'21년 정보화전략계획 수립 공통가이드에 의하면, 클라우드 네이티브 애플리케이션 구축 시 SW 개발비 산정에 대한 검토가 필요합니다.

대분류	중분류	소분류	관련 기준	비고
시스템 구축비	HW 구매비	HW 구매/설치비	정보시스템 HW 규모산정 지침(한국정보통신기술협회)	클라우드 서비스
	SW 구축비	상용 SW 구입비		클라우드 서비스
		SW 개발비	SW사업 대가산정 가이드 (한국소프트웨어산업협회)	
	DB 구축비	DB 설계비	SW사업 대가산정 가이드(한국소프트웨어산업협회)	
		데이터 제작		
	시스템 운용환경 구축비	운용환경 설계비	엔지니어링 사업대가의 기준(산업통상자원부)	클라우드 서비스
		운용환경 공사(시설/통신망)		
	기존 시스템 이전비	HW 이전비		
		데이터 이전비		
부대비	감리비		정보시스템 감리기준(행정안전부)	

출처 : 정보화전략계획(ISP) 수립 공통가이드(2021년) 참조

클라우드 네이티브 애플리케이션 구축과 관련된 개발 비용

클라우드 네이티브 애플리케이션 개발 비용 산정

클라우드 네이티브 도입 시 측정 단위는
트랜잭션과 데이터 관점에서 마이크로서비스 단위로 변화가 필요합니다.

현행 기능 분류체계

애플리케이션
(예. 총무관리 시스템)

세부 업무
(예. 회의실 관리)

단위 프로세스
(예. 회의실 예약)

마이크로서비스 기반 개발 시 업무 및 프로세스 재정의 필요

클라우드 네이티브 도입 시 기능 식별의 변화

트랜잭션(EI, EO, EQ)

FP 산정을 위한 업무 및 프로세스의 변화 예상

- 기존의 업무 및 프로세스 단위를 마이크로서비스 단위로 변경할 것인지 의사결정 필요
- 마이크로서비스 기반 개발 시 업무 및 프로세스가 세분화될 수 있음
- 기관의 결정에 따라 기존의 업무 및 프로세스 체계대로 작업할 수 있음 (단, 마이크로서비스 관련 백엔드 작업에 대한 작업 비중은 보정계수로 조정할 수 있음)

데이터(ILF, EIF)

데이터 처리 및 연계 관련 프로세스의 변화 예상

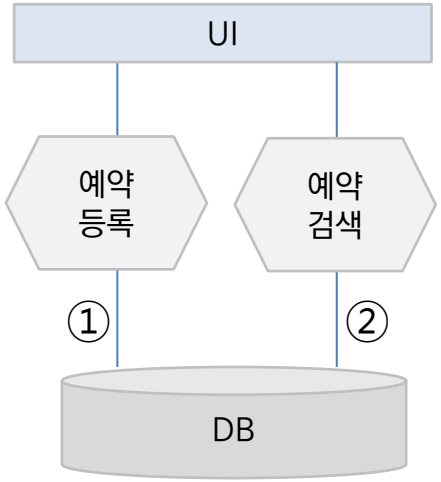
- 일반적으로 마이크로서비스 단위별로 자체 데이터를 관리하므로 내부데이터 처리(ILF)와 외부데이터 처리(EIF) 관련 기능점수가 높아질 수 있음
- 기관에서 기존의 데이터 처리방식대로 FP 작업을 수행할 가능성 있음

클라우드 네이티브 애플리케이션 개발 비용 산정

사용자 관점의 트랜잭션 처리 관련 기능점수의 변화는 없지만, API 게이트웨이를 통한 데이터 처리 관련 기능유형의 증가가 예상됩니다.

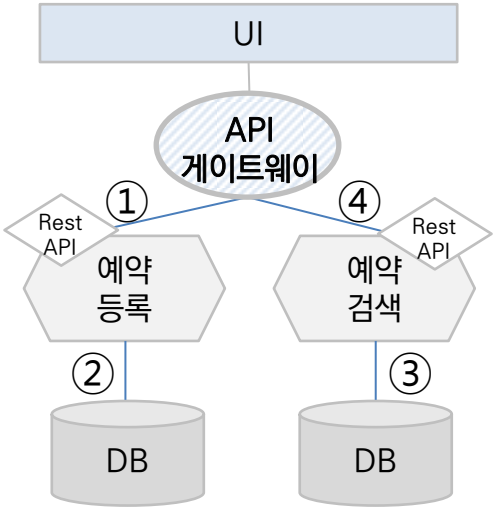
마이크로서비스 아키텍처에서의 기능 식별 방안

모놀리식 아키텍처



VS.

마이크로서비스 아키텍처



단위 프로세스	FP 유형
① 예약 등록	EI
② 예약 검색	EQ

단위 프로세스	FP 유형
① 예약 등록 API 연계	ILF
② 예약 등록	EI
③ 예약 검색	EQ
④ 예약 검색 API 연계	ILF 또는 EIF

기능 식별 시 고려사항

- 마이크로서비스 아키텍처는 API 게이트웨이를 통해 서비스 연계 작업을 수행하므로 이와 관련된 프로세스가 추가됨
  - 예약 등록 서비스의 경우, 예약등록 API 연계가 내부논리파일(ILF)로 추가됨
  - 예약 검색 서비스의 경우, 예약검색 API 연계가 내부논리파일(ILF)로 추가됨 (예약검색 DB가 없는 경우에 예약등록 DB를 읽어야 하므로 외부연계파일(EIF)로 정의함)

- **마이크로서비스 아키텍처 적용 시 모놀리식 아키텍처 대비 API 연계 프로세스의 추가에 따라 기능점수가 증가할 수 있음**
  → 즉 개발비 증가가 예상됨

클라우드 네이티브 기반 행정·공공 서비스 확산 지원
클라우드 네이티브 발주자 가이드

클라우드 네이티브의 아키텍처 참조모델은?

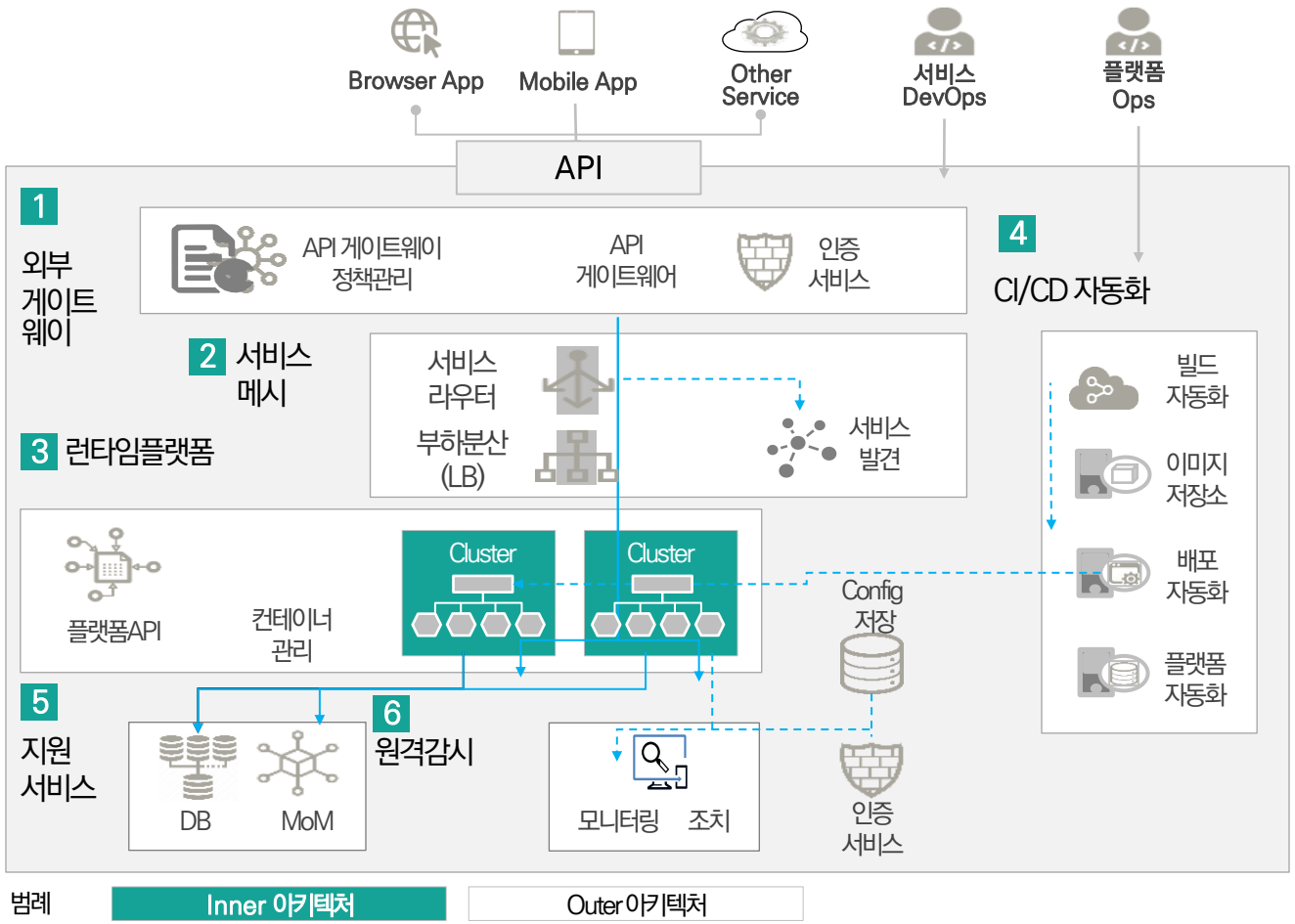
NIA 한국지능정보사회진흥원

가트너 참조모델

GARTNER

가트너는 클라우드 네이티브 아키텍처 참조모델을 제공합니다.

외부 게이트웨이, 서비스 메시, 런타임 플랫폼, CI/CD, 지원, 원격감시 구성



1 **외부로부터의 서비스 요청을 내부 구조를 드러내지 않고 처리하기 위한 구성요소**

2 **마이크로서비스 구성 요소 간의 네트워크를 제어**

3 **마이크로서비스를 실행하기 위한 컨테이너와 그 컨테이너를 관리하는 역할**

- Inner 아키텍처: 개별 마이크로 서비스 구축을 위한 아키텍처
- Outer 아키텍처: 개별 마이크로 서비스가 개발, 배포, 실행되는 전체 운영, 관리환경

4 **마이크로서비스의** 지속적인 통합, 지속적인 전달, **지속적인 배포 체계**

5 애플리케이션이 실행하기 위해 지원되는 모든 서비스

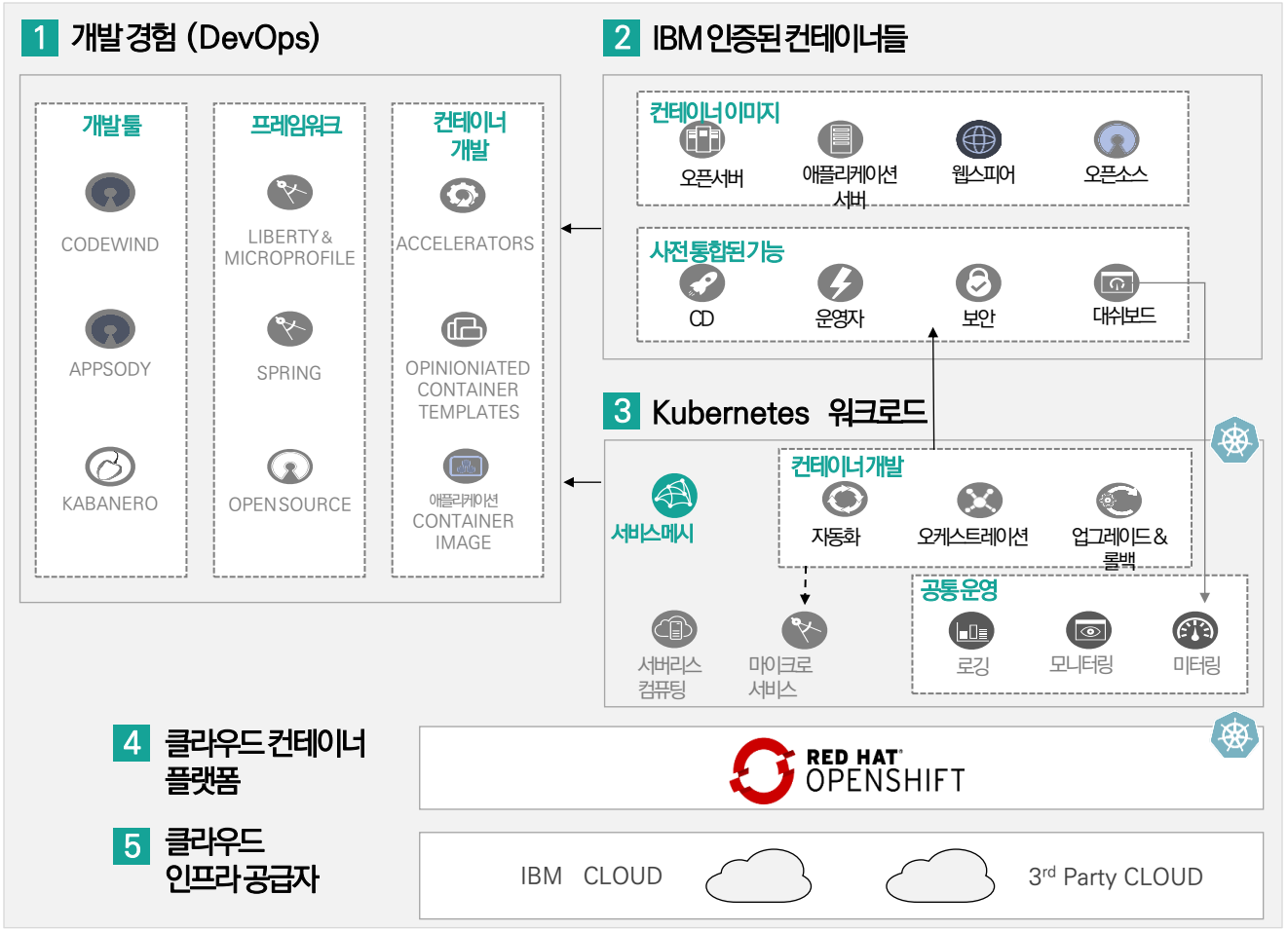
6 분산환경에서 서비스 모니터링 및 헬스체크

IBM 참조모델

IBM

IBM은 클라우드 네이티브 아키텍처 참조모델을 제공합니다.

컨테이너/ 개발경험(DevOps) 중심으로 구성, 클라우드 인프라와 플랫폼은 서비스 임차



1 **개발자 및 아키텍트를 위한 로컬 개발, 테스트 영역**
개발을 위한 프레임워크, 런타임 사전 제공, 검증

2 레드햇 오픈시프트 인증된 **컨테이너 이미지 제공**
오픈소스에 대한 지속적인 **통합 및 배포가 가능하도록 툴 제공**

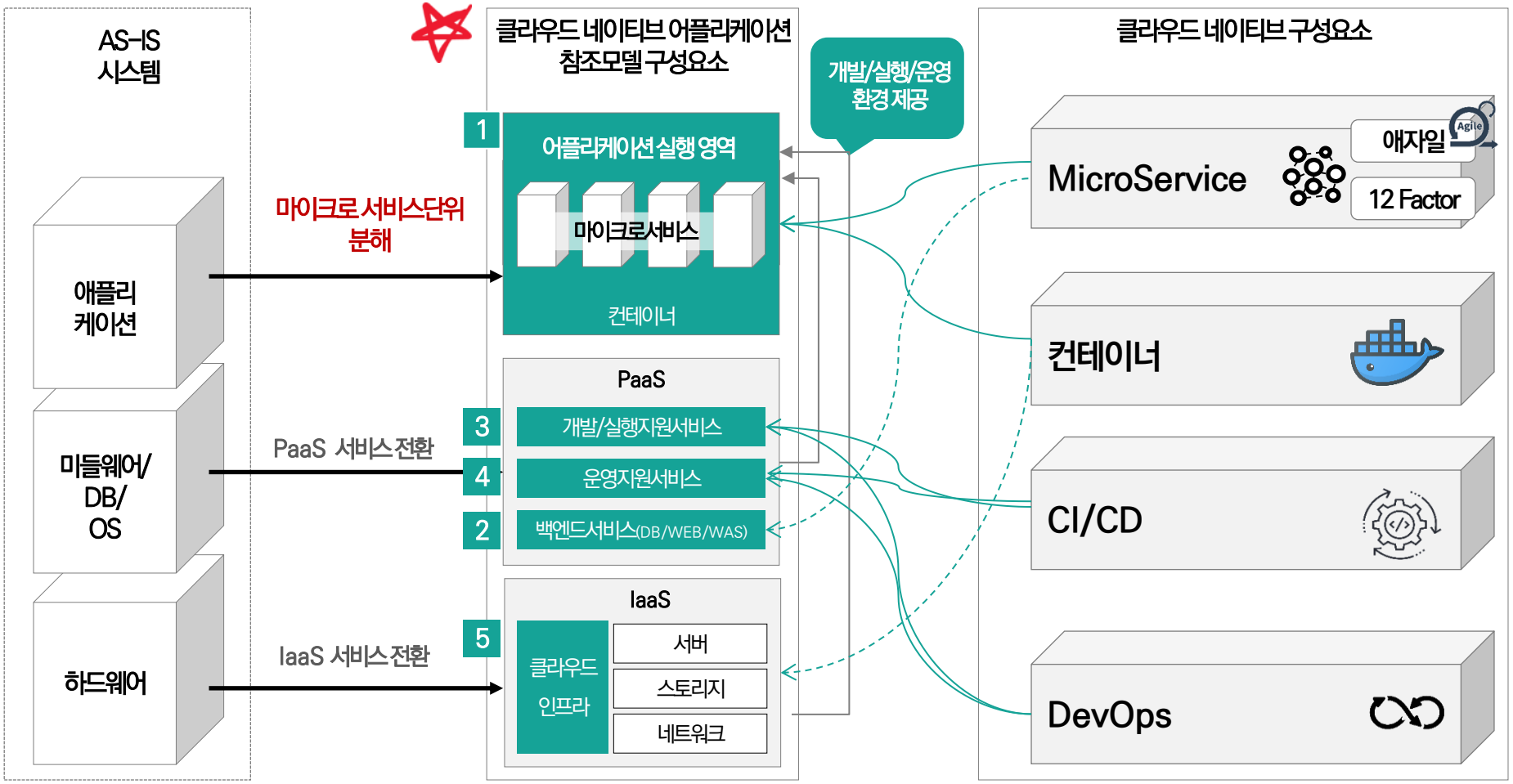
3 **서비스 메시를 사용할 수 있도록 Config 저장, 실행영역, 컨테이너 개발 및 운영팀을 위한 모니터링, 대시보드 등 관리기능** 제공

4 쿠버네티스 기반 컨테이너 오케스트레이션 및 실행 영역
컨테이너 배포, 업데이트 및 워크로드별 리소스 관리

5 마이크로서비스 운영하기 위한 인프라 스트럭처 영역
컨테이너 플랫폼에 VM 및 베어메탈 자원을 제공

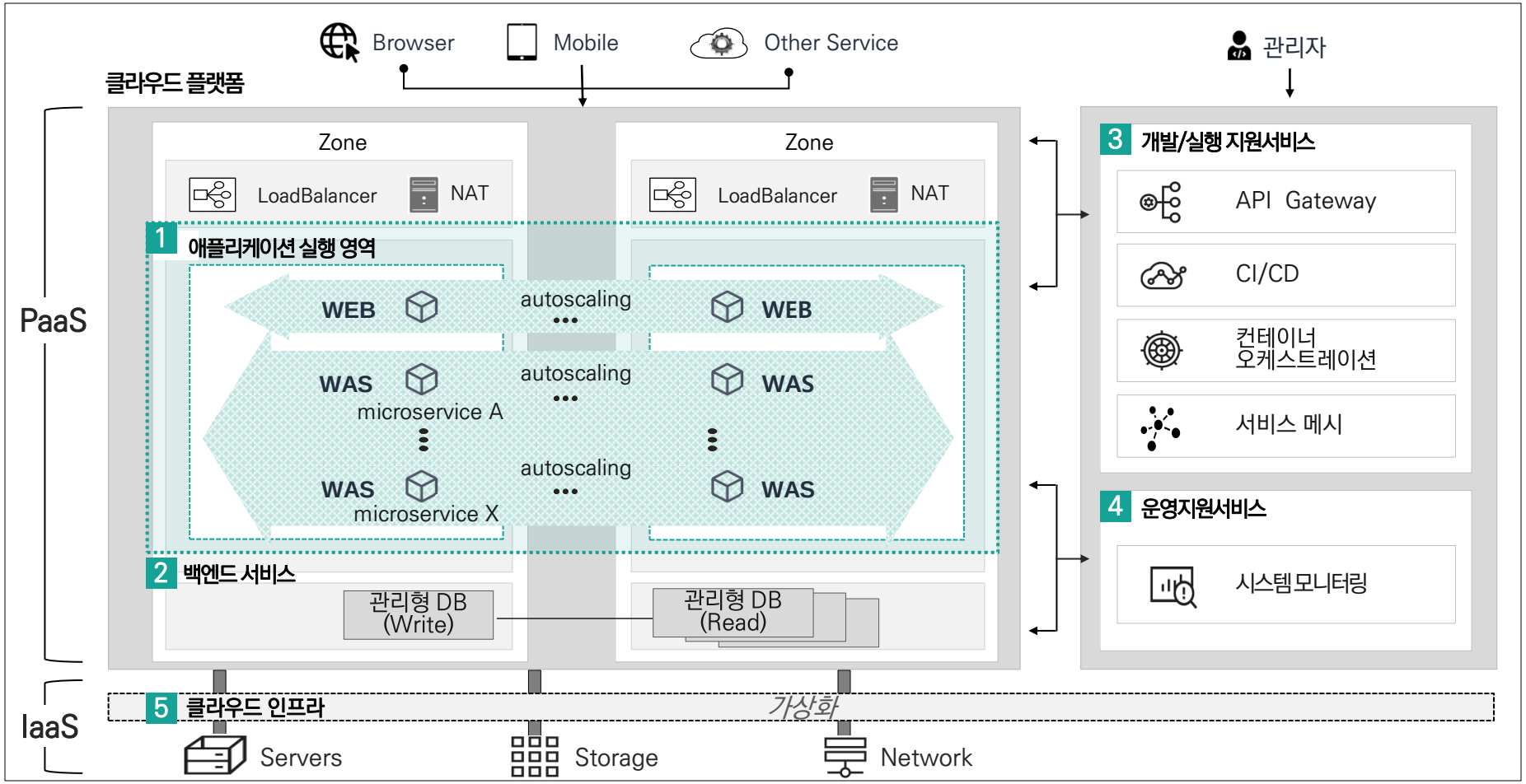
참조모델 구성요소

선진사례를 통하여 아키텍처 참조모델 구성요소를 정립합니다.



클라우드 네이티브 애플리케이션 아키텍처 참조모델

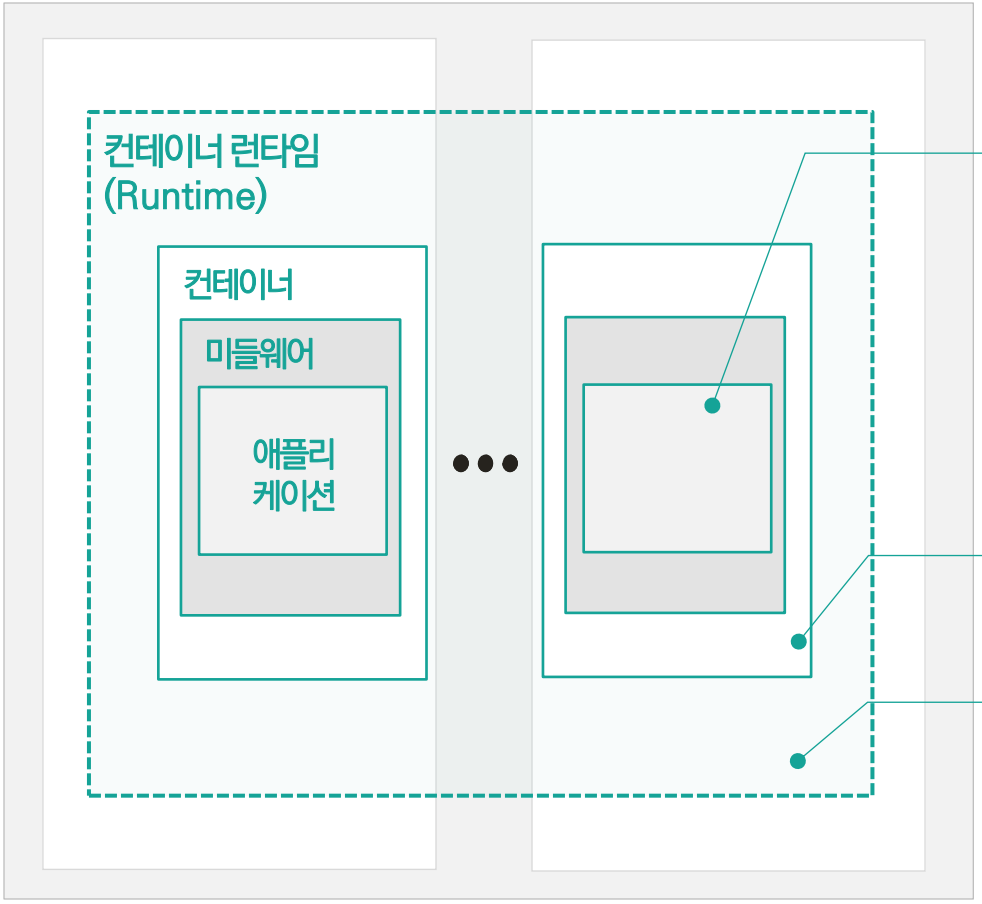
클라우드 네이티브 애플리케이션 아키텍처 참조모델
및 구성영역을 체계화합니다.



애플리케이션 실행 영역

1 아키텍처 참조모델의 애플리케이션 실행 영역입니다.

애플리케이션 실행 영역은 클라우드 네이티브 애플리케이션이 컨테이너 단위로 배치, 실행되는 영역으로
클라우드 네이티브 애플리케이션과 실행단위인 컨테이너 및 컨테이너 구동을 위한 컨테이너 런타임으로 구성됨



애플리케이션 실행 영역 구성

클라우드 네이티브 애플리케이션

- 미들웨어 (Web Server, WAS)를 포함하여 배포하여 개별적 실행 가능
- 배포대상에 따라 아래의 2가지 컨테이너로 구성
  - Web Server : 이미지 등 정적 컨텐츠 서비스
  - WAS : 마이크로서비스 단위 애플리케이션 서비스

컨테이너 (Container)

- 클라우드 네이티브 애플리케이션 배포 및 실행 단위

컨테이너 런타임 (Container Runtime)

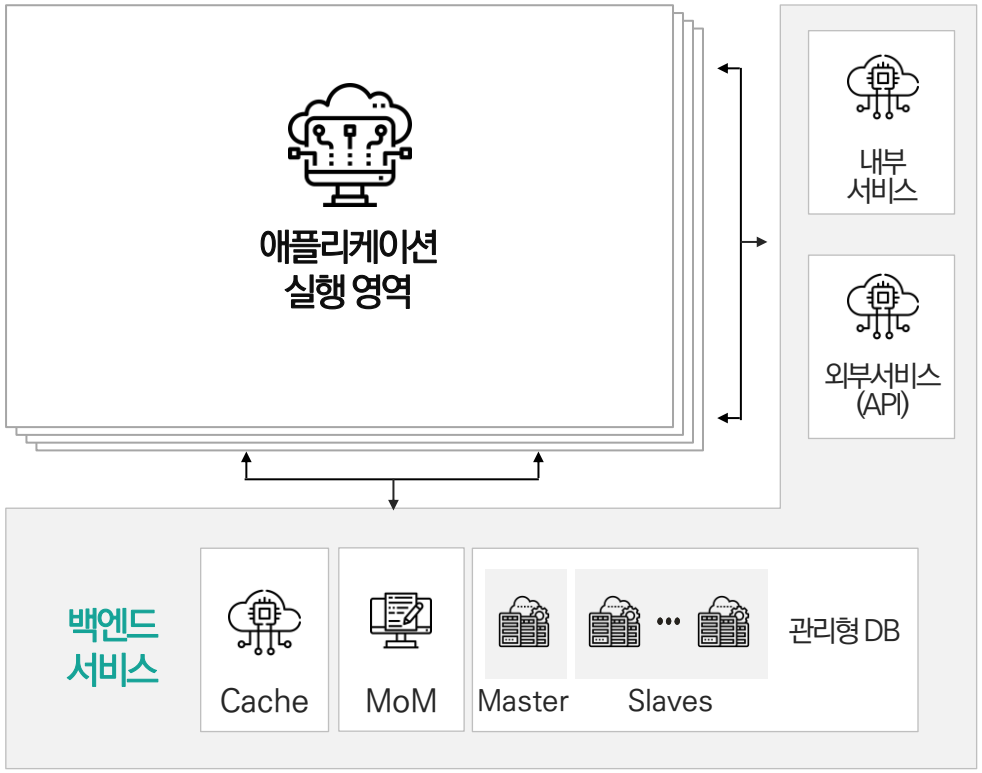
- 클라우드 실행환경 제공

  예) 도커, 쿠버네티스 등

백앤드 서비스 영역

2 아키텍처 참조모델의 백앤드 서비스 영역입니다.

클라우드 네이티브 애플리케이션을 실행하기 위해 네트워크로 연결된 모든 리소스를 백엔드 서비스라고 하며,
클라우드 플랫폼에서 제공하는 서비스, 외부 연계 서비스 및 직접 구축한 서비스를 모두 포함함



백엔드 서비스 종류

DB	• 클라우드 플랫폼에서 제공 및 관리되는 DB 서비스
Cache	• 세션관리, 성능향상 등의 용도를 위한 메모리 캐싱 서비스
MoM	• 비동기 메시징(Messaging) 서비스
외부 서비스	• 인증, 간편결제, OpenAPI 형태 외부 연계 서비스
내부 서비스	• AI, 빅데이터, 챗봇 등의 클라우드 플랫폼 제공 서비스

클라우드 네이티브 애플리케이션을 수정하지 않고 다른 백엔드 서비스로 전환이 가능하도록 구성되어야 함

개발·실행 지원서비스 영역

3 아키텍처 참조모델의 개발·실행 지원서비스 영역입니다.

클라우드 네이티브 애플리케이션을 실 환경에서 효율적으로 배포하고 안정적으로 운영하기 위해서는
클라우드 네이티브 애플리케이션에 최적화된 개발 및 실행환경을 구성해야 함

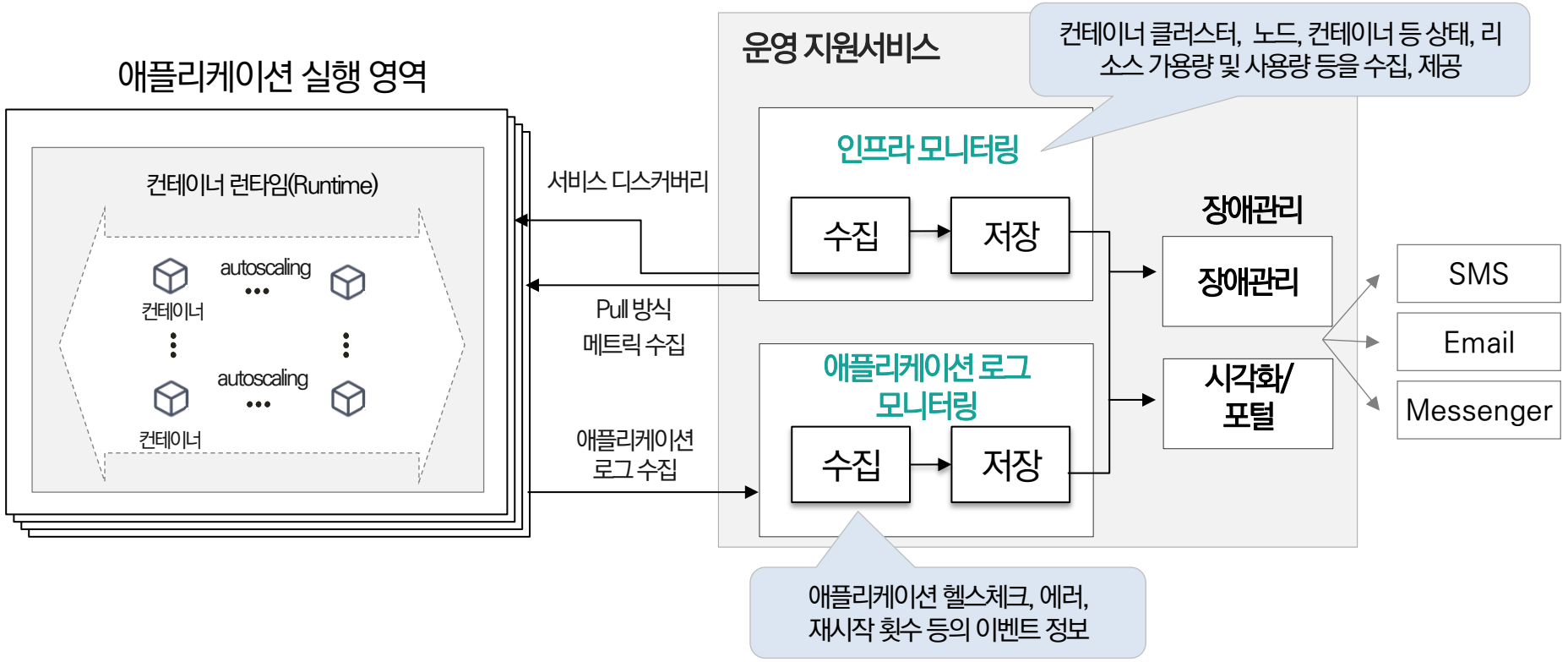
구분	지원서비스	서비스 내용	주요 솔루션
실행 환경	API Gateway	API 형태의 서비스를 제공하는 마이크로서비스 앞단에서 앤드포인트를 단일화하여 외부 사용자에 제공 API에 대한 인증과 인가 및 여러 서버로 라우팅 하는 기능 등을 담당함	Spring Cloud Gateway, zuul, kong 등
	컨테이너 오케스트레이션	여러 서버에 걸쳐 다수의 컨테이너에 대한 배포, 컨테이너 단위의 Auto scaling, 자원할당 , 장애 복구 및 모니터링 등의 운영 자동화를 제공하는 서비스	Kubernetes, Cloud Foundry, Docker Swarm 등
	서비스 메시	대규모의 마이크로서비스를 실시간으로 제어, 관리서비스 간에 통신을 하기 위해 Service discovery, 로드밸런싱, 장애 제어, Config 관리 등 제공	Spring Cloud, Istio 등
개발 환경	CI/CD	개발, 테스트, 배포 프로세스에 대한 도구 기반 자동화 및 모니터링 제공 CI(Continuous Integration)는 코드 변경 사항이 지속적으로 통합(Continuous Integration)하는 것이고 CD(Continuous Delivery)는 지속적인 서비스 제공(Continuous Delivery)을 의미	Jenkins, bamboo, git, svn 등

개발 및 실행환경 구성을 위해서 지원서비스의 활용도가 높아지고 있으며
1) 클라우드 플랫폼에서 제공되는 서비스를 이용하거나 2) 오픈소스 및 상용 SW를 활용하여 자체 구축이 가능함

운영 지원서비스 영역

4 아키텍처 참조모델의 운영 지원서비스 영역입니다.

운영 지원서비스는 클라우드 네이티브 애플리케이션 및 인프라 리소스 (CPU, Memory 등)에 대한
로그 및 사용데이터를 수집, 분석, 시각화 등 애플리케이션 전반의 모니터링 기능을 제공함



클라우드 네이티브 애플리케이션은 마이크서비스 단위로 작아지고 OS 수준에서 가상화 된 환경에서 모니터링 대상 컨테이너 또한 동적으로 생성되므로 기존 모니터링과 다르게 서비스 디스커버리 및 Pull 방식의 데이터 수집이 적용됨

클라우드 인프라 영역

5 아키텍처 참조모델의 클라우드 인프라 영역입니다.

클라우드 네이티브 환경에서 클라우드 인프라 위에
컨테이너와 같은 경량화된 가상화 기술 및 네트워크 및 스토리지 가상화를 적용하여 클러스터를 구축함

서버, 스토리지, 네트워크 장비
등과 같은 인프라 구성요소로
클라우드의 IaaS 에 해당하며,
클라우드 네이티브 애플리케이션은
컨테이너 기반의 실행환경 외에도
마이크로서비스 구조의 애플리케이션을
개발, 운영할 수 있는
클라우드 플랫폼 서비스(PaaS)가
갖춰진 클라우드 환경이 필요함

구성요소 , 방법론 및 원칙 등에 관련된 용어설명

- **MSA(Micro Service Architecture)** : 하나로 구성된 어플리케이션을 여러 개의 작고 느슨한 형태의 서비스로 쪼개어 **독립적으로 서비스하고 배포할 수 있도록 구성하는 아키텍처**

- **컨테이너** : 호스트 서버의 운영체제 수준의 **경량화 된 가상화 기술** 및 그 결과물인 격리된 인스턴스(빠른 수평적 확장)

- **도커(Docker)** : 리눅스 컨테이너 기술을 적용한 가장 대표적인 컨테이너 제공 솔루션

- **쿠버네티스** : 컨테이너 오케스트레이션 도구들이 출현하였으며 대표적인 도구

- **DevOps** : 개발과 운영 프로세스의 통합 및 자동화를 통해 신속한 서비스를 제공하도록 지원하는 **통합프로세스, 조직문화, 도구 등을 포함한 체계**

- **DevOps 툴체인 (Tool Chain)** : DevOps를 효과적으로 적용하기 위해서 DevOps 전 프로세스를 여러 가지 도구(Tool)로 연결하여 자동화하는 과정

- **CI/CD** : 반복적인 개발, 테스트, 배포 과정에 대한 자동화와 모니터링을 제공하는 **도구기반 프로세스**

- **애자일(Agile)** : 사용자의 요구사항을 이해, 신속하게 반영하기 위해 기획, 설계 과정에 많은 시간과 노력을 기울이지 않고 빠르게 프로토타입을 개발하여 사용자의 피드백과 방향성을 확인하고 지속적인 **개선 과정을 짧은 주기로 반복하는 개발방법론**

- **12 Factor** : 클라우드 환경에 최적화되어 클라우드 모든 기능을 제대로 사용할 수 있게 위해서 지켜야 할 규칙

12 Factor

클라우드 네이티브 기반 행정·공공 서비스 확산 지원
클라우드 네이티브 발주자 가이드

클라우드 네이티브의 아키텍처 구성요소는?

NIA 한국지능정보사회진흥원

컨테이너 – 컨테이너 정의 및 구조

컨테이너

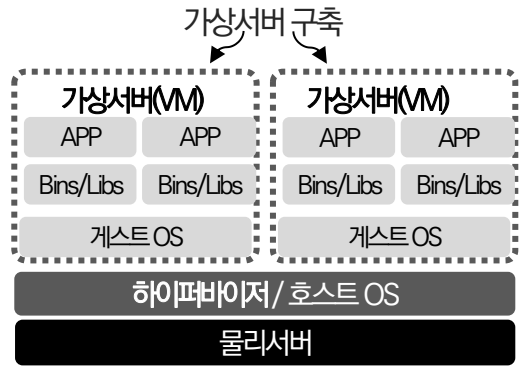
컨테이너는 어떤 환경에서나 실행하기 위해 필요한 애플리케이션 코드, 런타임 모듈, 라이브러리 등의 모든 요소를 포함하는 경량화된 소프트웨어 패키지임

기존 방식과 컨테이너 기술 비교

베어메탈

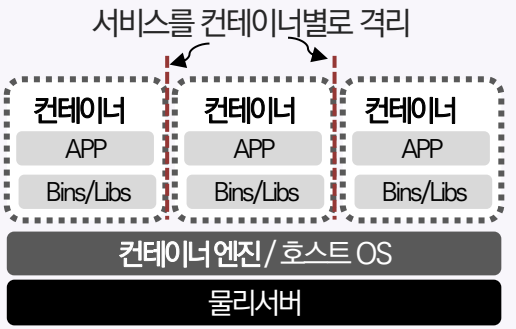
- 물리서버 전체를 하나의 OS 환경으로 사용

서버 가상화(HW 레벨)



- 물리서버 자원(CPU, 메모리 등)을 분할하여 논리적인 가상서버를 생성
- 복수의 OS 환경 사용
- IaaS (Infrastructure as a Service)

컨테이너 가상화(OS 레벨)



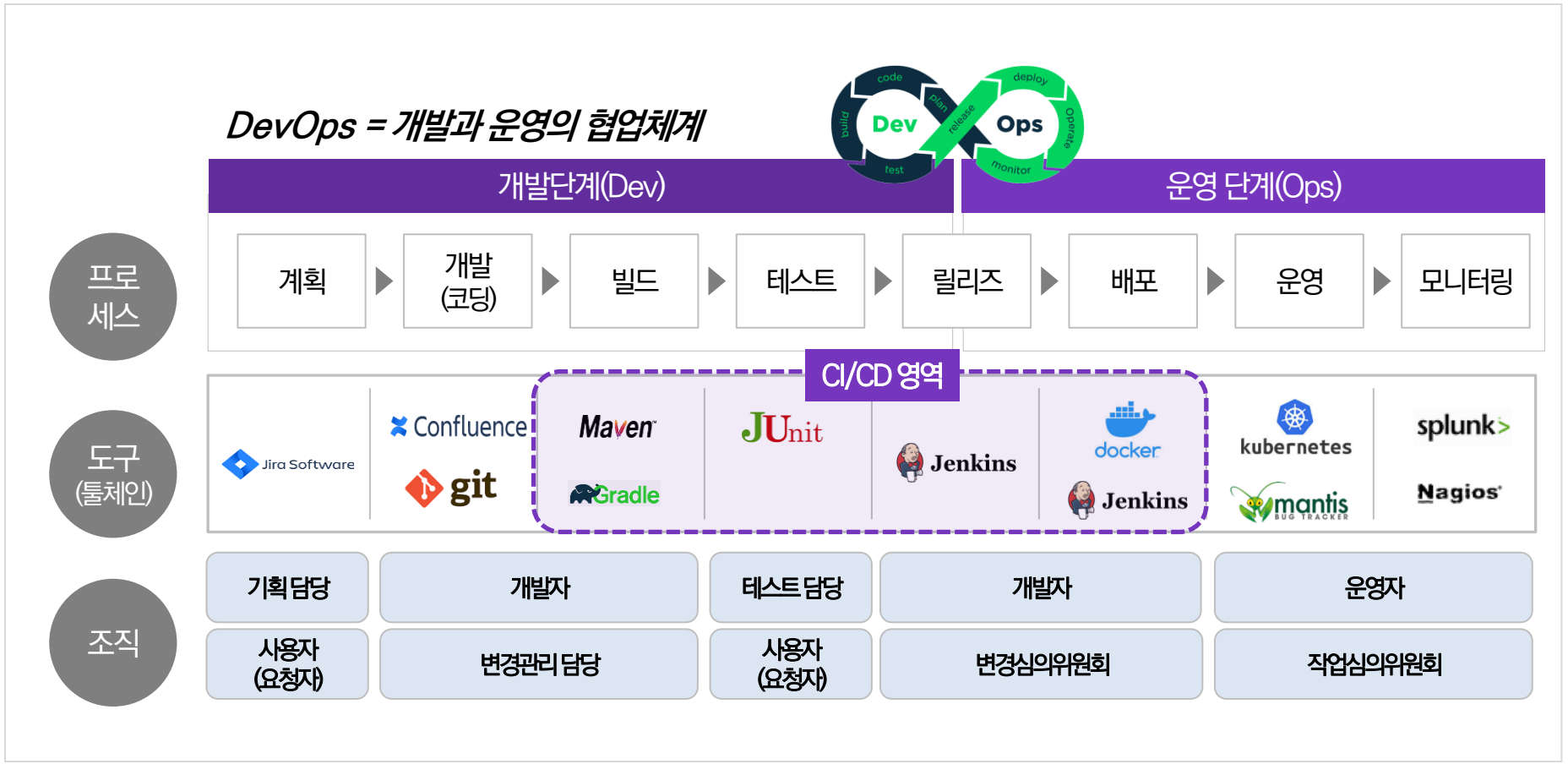
- 물리서버 자원(CPU, 메모리 등)을 분할하여 논리적인 컨테이너를 생성
- 하나의 OS 환경에 복수 실행 환경 사용
- CaaS, PaaS, FaaS

- 컨테이너 가상화는 기존의 서버 가상화 방식과 달리 컨테이너별로 실행환경을 격리하여 가볍고 빠른 서버환경을 제공하며, 하나의 물리서버에 많은 수의 컨테이너를 필요 시 즉각 생성할 수 있음
- 좀 더 효율적이고 경량화된(light weight) 애플리케이션 실행 환경에 대한 요구가 증가함에 따라 가상화 방식은 기존의 가상화 기술인 하이퍼바이저 방식에서 컨테이너 엔진 기반 가상화 방식으로 전환되고 있음

DevOps & CI/CD (1/2) – DevOps 개요

DevOps

- DevOps는 개발(Development)과 운영(Operatios)의 합성어로, 개발과 운영 프로세의 통합 및 자동화를 통해 신속한 서비스를 제공하도록 지원하는 **통합프로세스, 도구, 조직문화** 등을 포함한 체계
- **개발팀과 운영팀의 협업이 가능해져 업무 병목 구간을 최소화하고, 애플리케이션을 신속하게 개발, 배포할 수 있음**

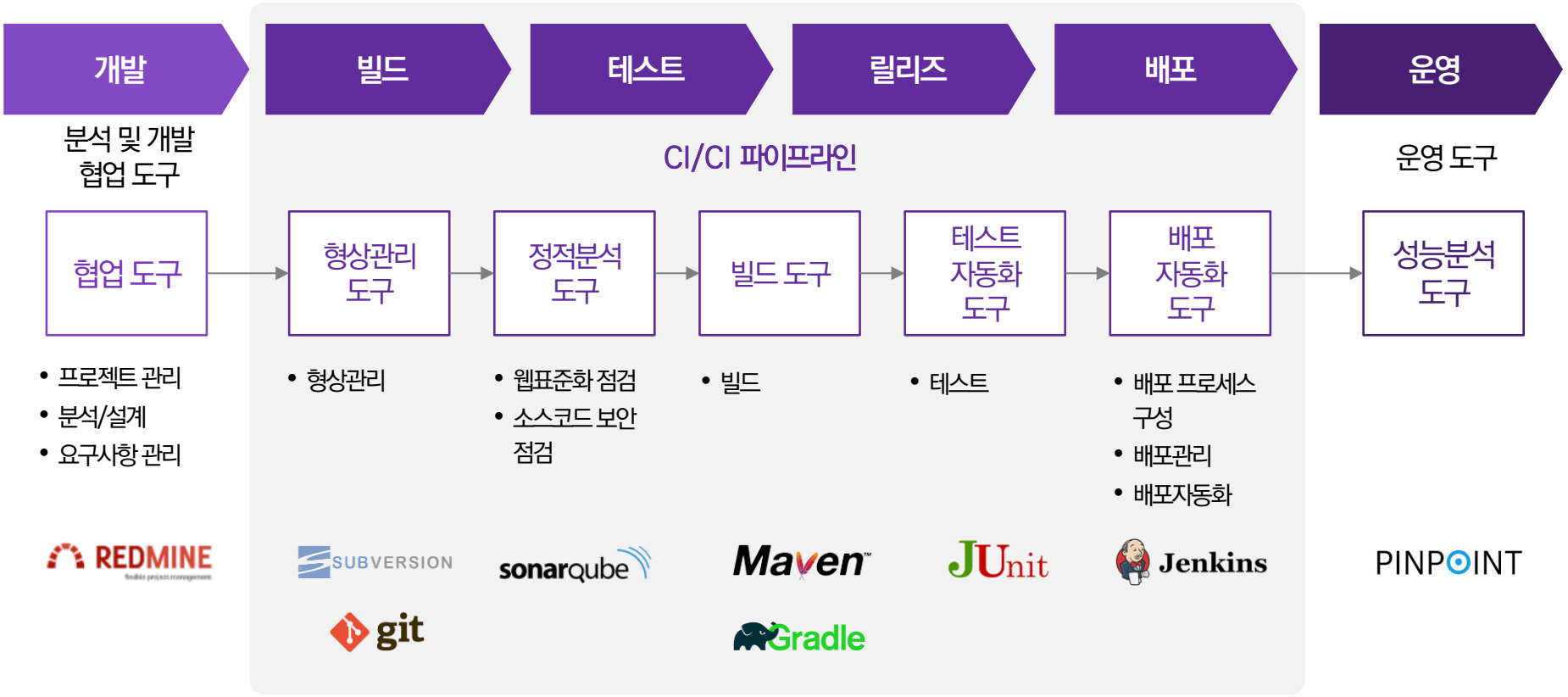


DevOps & CI/CD (2/2) - DevOps 개요

CI/CD

CI/CD 파이프라인은 개발 이후 빌드, 테스트, 릴리즈, 배포 과정에 자동화 도구를 활용하여 애플리케이션 배포주기를 단축하여, 서비스 요구에 신속하게 대응하도록 지원함

CI/CD 파이프라인 예시



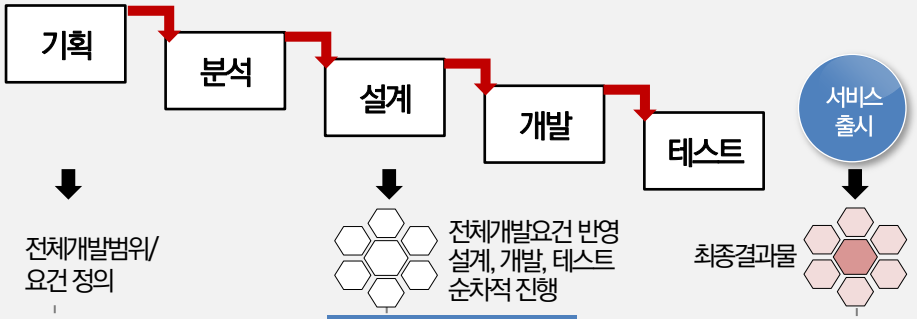
애자일 방법론

애자일 방법론

폭포수 방법론의 개발 공정은 기획, 분석, 설계, 개발, 테스트 단계가 위에서 아래로 순차적으로 진행되며, 애자일 방법론은 각 개발 공정을 명확하게 구분하지 않고 각 단계를 반복적으로 수행하면서 요구사항을 추가하거나 수정하면서 개발을 수행함

폭포수 방법론과 애자일 방법론 비교

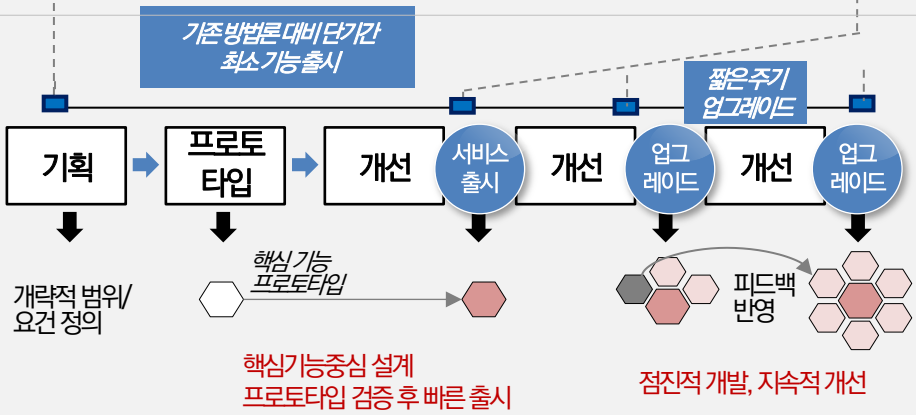
폭포수 방법론



폭포수 방법론의 문제점

- 기획단계 많은 시간 소요
- 초기 요구사항 불명확, 잦은 변경
- 정형화된 절차, 방법론 적용으로 많은 산출물 발생

애자일 방법론



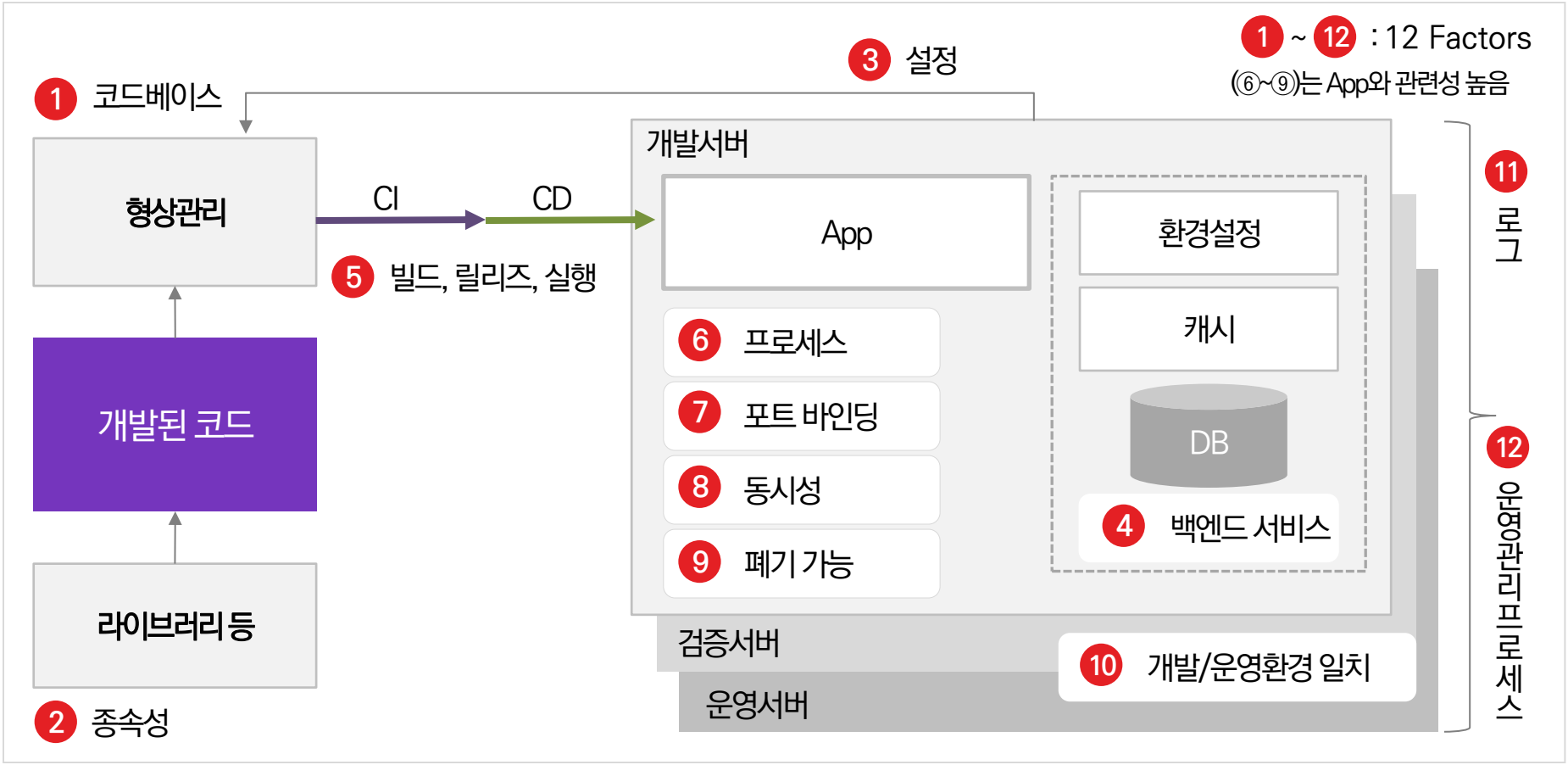
애자일 방법론 개발 방식

- 과도한 기획 지양
- 핵심기능에 집중 빠른 출시 목표
- 고객가치가 높은 기능 우선순위로 반복적으로 개선

12가지 원칙

12 Factors

12 Factors App 원칙은 2012년 Heroku에서 일하던 개발자들이 클라우드에 적합한 SaaS 애플리케이션 개발과 배포 방법에 맞는 12가지 원칙을 개념화한 것으로 클라우드 네이티브 환경에 적합하게 적용되어야 할 부분들을 명확하게 정의하고 있음



12 Factors

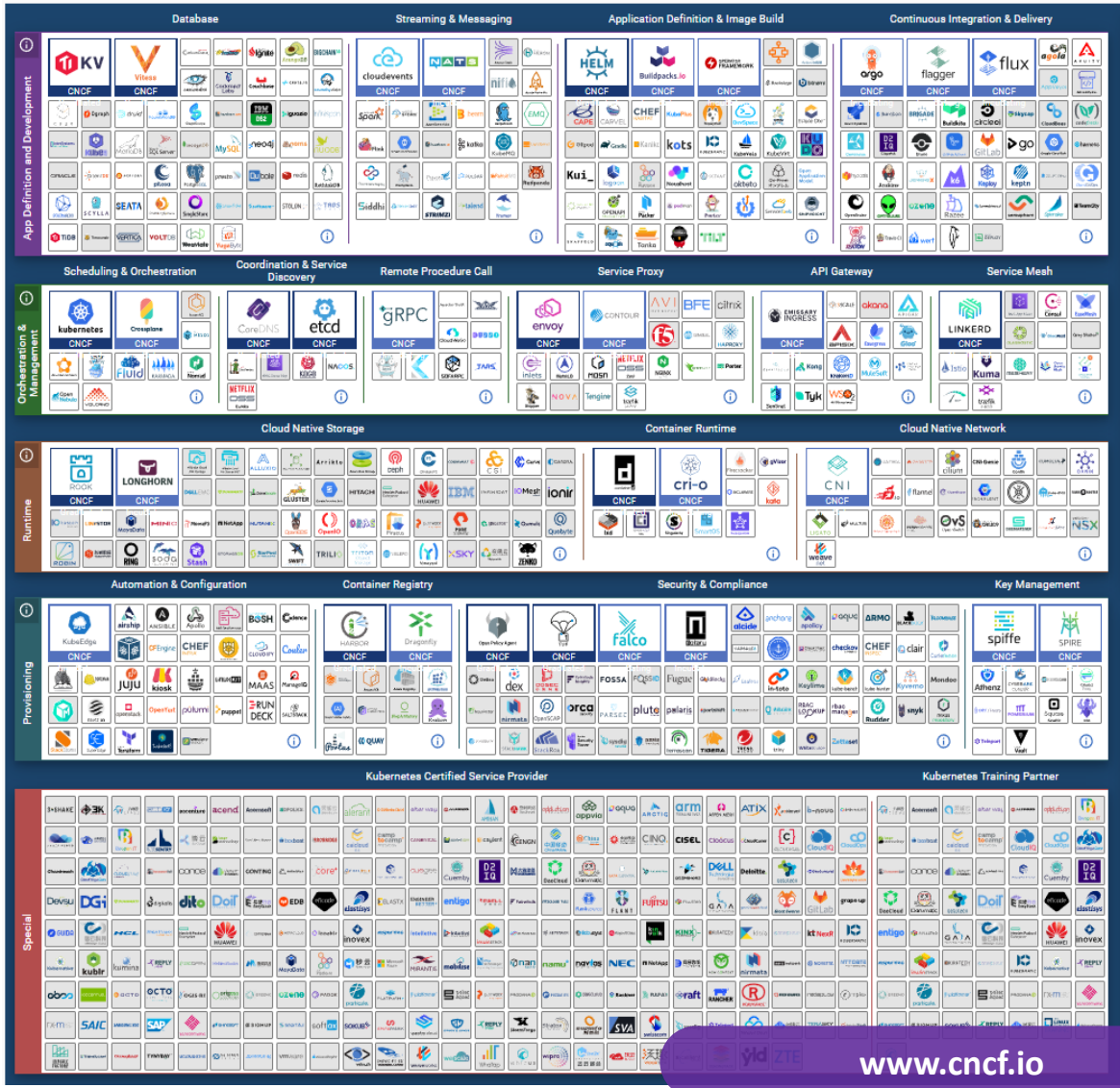
12 Factors

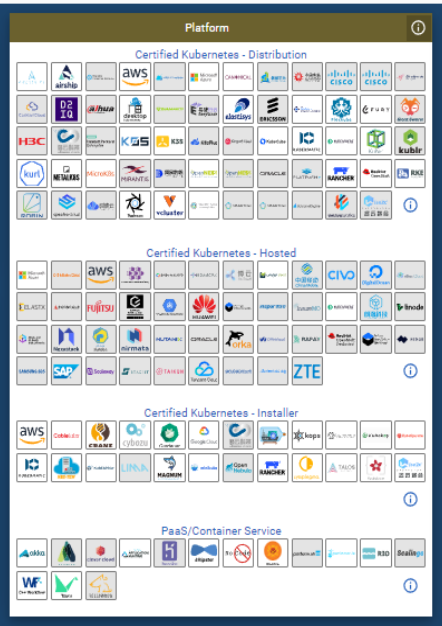
12 Factors App 원칙은 2012년 Heroku에서 일하던 개발자들이 클라우드에 적합한 SaaS 애플리케이션 개발과 배포 방법에 맞는 12가지 원칙을 개념화한 것으로 클라우드 네이티브 환경에 적합하게 적용되어야 할 부분들을 명확하게 정의하고 있음

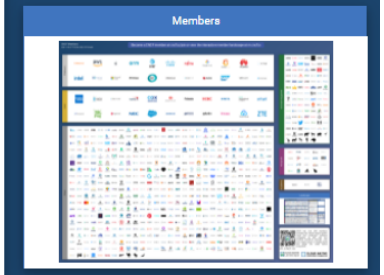
12 Factors 원칙 설명

1	**코드베이스**[1] (Codebase)	• 하나의 코드베이스(소스코드)로 버전관리하여, 이를 여러 곳에 배포
2	**종속성** (Dependencies)	• 패키지, 라이브러리 등 종속이 필요한 경우 명시적으로 선언하고 분리시켜 실행환경 종속성 제거
3	**설정** (Configuration)	• 소스 코드와 설정 정보를 분리, 실행시 코드에서 읽어서 사용
4	**백엔드서비스** (Backing Services)	• 애플리케이션 작동에 필요한 서비스(DB, 메시지큐, 캐시등)를 연결된 리소스로 취급하여 연결/분리가 용이
5	**빌드, 릴리즈, 실행** (Build, Release, Run)	• 빌드, 릴리즈, 실행단계를 엄격히 분리
6	**프로세스** (Processes)	• 애플리케이션 실행시 하나 혹은 여러 개의 stateless 프로세스로 실행
7	**포트바인딩** (Port-binding)	• 애플리케이션은 독립적이며, http 같은 포트 바인딩을 통해서 외부에 서비스 제공
8	**동시성** (Concurrency)	• 애플리케이션을 수평적으로 확장하며, 무상태(Stateless) 특성이 이런 확장을 단순하게 만듦
9	**폐기가능** (Disposability)	• 빠른 시작과 그레이스풀 셧다운(Graceful shutdown)을 통한 안정성 극대화
10	**개발/운영 환경 일치** (Dev/Prod Parity)	• 개발계/검증계/운영계 환경을 가능한 비슷하게 유지함으로써 지속적인 개발/배포 가능
11	**로그** (Log)	• 로그 파일을 이벤트 스트림으로 취급하여 이를 취합, 인덱싱, 분석할 수 있어야 함
12	**운영관리 프로세스** (Admin Process)	• 시스템관리 작업은 일회성 프로세스로 만들어서 실행

클라우드 네이티브 한눈에 알아 보기 – CNCF.io '21년







www.cncf.io

클라우드 네이티브 기반 행정·공공 서비스 확산 지원
클라우드 네이티브 발주자 가이드

클라우드 네이티브 애플리케이션의 개발원칙은 어떻게 되는가?

NIA 한국지능정보사회진흥원

12 Factors와 클라우드 네이티브 환경의 관련성

개발원칙과 환경

12 Factors 원칙은 플랫폼을 가정한 개발 원칙으로, 플랫폼이 제공하는 환경을 이해하고 어플리케이션의 개발 및 빌드/배포/운영 환경 전반의 고려사항을 제시함

1	**코드베이스** (Codebase) [CI/CD]	하나의 코드베이스(소스코드)로 버전관리 버전관리 및 배포 관리 자동화 환경
2	**종속성** (Dependencies) [Code] [CI/CD]	패키지, 라이브러리 등 **의존관계는 명시적으로 선언하고 분리**
3	**설정** (Config) [Code] [배포]	소스 코드와 설정 정보를 분리 **설정 정보는 환경변수에 저장**
4	**백앤드서비스** (Backing Services) [배포]	**백엔드 서비스**(DB, 메시지큐, 캐시등) **는 리소스 매핑으로 처리**(연결/분리 용이)
5	**빌드/배포/실행** (Build, release, run) [CI/CD]	**빌드와 실행 단계는 철저히 분리**
6	**프로세스** (Processes) [Code] [배포]	**어플리케이션은 Stateless 프로세스로 실행**
7	**포트바인딩** (Port binding) [배포]	애플리케이션은 독립적이며, http 같은 **포트 바인딩을 이용한 서비스 공개**
8	**동시성** (Concurrency) [운영]	애플리케이션을 수평적으로 확장, **프로세스 모델을 사용한 스케일아웃**
9	**폐기 가능** (Disposability) [Code] [운영]	**빠른 시작과 그레이스풀 셧다운** (graceful shutdown) **지원으로 안정성 극대화**
10	**개발/운영환경일치** (Dev/prod parity) [배포]	**가능한 동일한 개발, 스테이징, 운영 환경의 유지**
11	**로그** (Logs) [Code] [운영]	**로그파일을 이벤트 스트림으로 로그 처리** (취합, 인덱싱, 분석)
12	**운영관리 프로세스** (Admin processes) [운영]	**시스템 관리작업은 일회성 프로세스로 만들어서 실행**

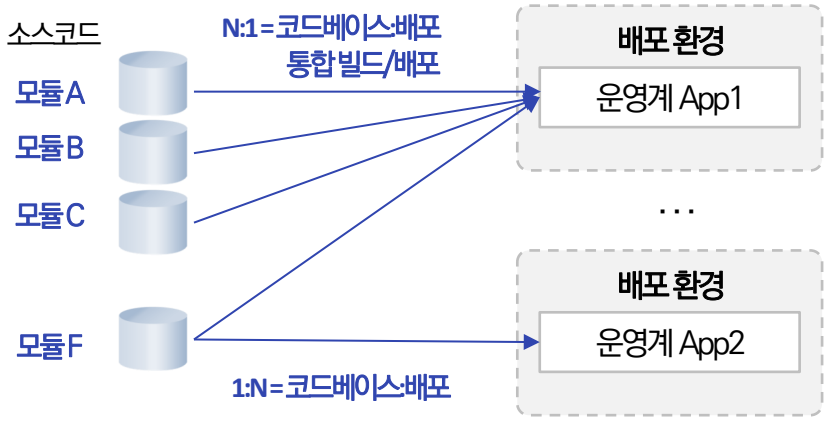
12가지 요소 : 1) 코드 베이스

코드 베이스

애플리케이션은 1개의 코드베이스를 통해 관리되어야 하며, 동일한 코드로 개발 및 운영 환경에 배포되어야 하며, 코드베이스는 각 애플리케이션마다 단 1개 존재 (애플리케이션은 소스코드의 변경과 여러 환경으로 수차례 배포 시에도 추적 가능)

AS-IS : 코드베이스와 배포 앱 간 N:N 관계

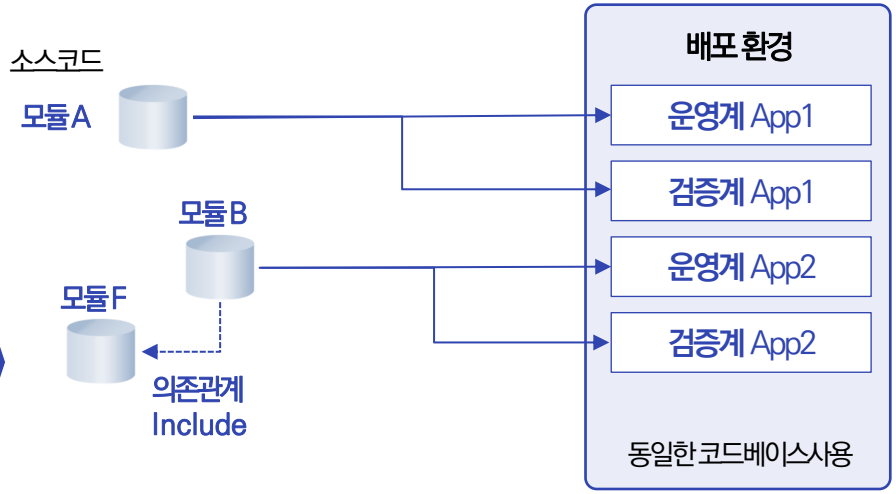
1:N, N:1 의 코드베이스와 배포 앱 관계



- 소스코드와 배포 앱 간의 관계가 1:N 혹은 N:1 관계 형성되어, 배포 앱의 코드 추적과 버전 관기 어려움이 있음
- 애플리케이션 관련 모든 자산, 소스코드, 프로비저닝 스크립트, 환경 설정 등 내용이 통일된 코드베이스(리파지토리)에 저장되지 못함

To-BE : 코드베이스와 배포 앱간 1: 1 관계 운영

코드베이스는 애플리케이션을 관리하는 리파지토리 (저장소)

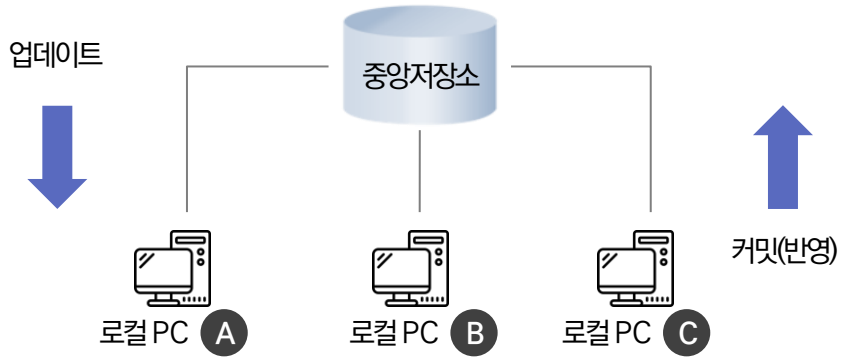


- 코드베이스는 SVN과 같은 중앙관리형과 Git과 같은 분산관리형이 존재함
- 애플리케이션 관련 모든 자산, 소스코드, 프로비저닝 스크립트, 환경 설정 등 내용이 코드베이스(리파지토리)에 저장되어 개발자 및 운영 담당자 등에 의해 사용됨
- 모든 배포는 1개의 코드베이스를 가지고, **코드베이스는 CI/CD 도구에 의해 관리**

코드베이스의 유형

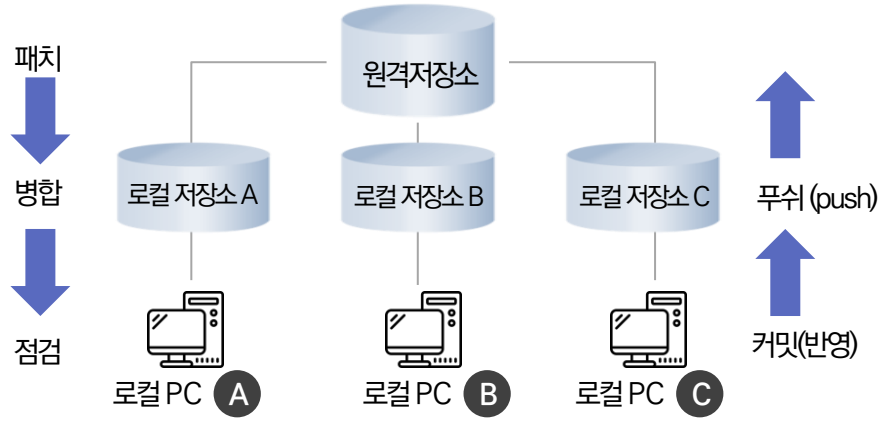
코드베이스는 SVN과 같은 중앙관리형과 Git과 같은 분산관리형으로 나뉘며, 각 방식의 장단점은 다음과 같음

중앙관리형 – SVN



- 로컬 PC에서 커밋(Commit)하면 중앙저장소에 반영
- **장점**
  - 직관적으로 사용 가능
  - 커밋하는 순간 모든 사람에게 공유되고, 모든 사람이 같은 자료를 받아 옴
- **단점**
  - 여러 사람이 하나의 파일을 동시에 수정하고 커밋할 경우, 충돌 발생 가능성
  - 중앙저장소 다운 시 작업의 어려움

분산관리형 – Git



Pull = 패치(fetch) + 병합(merge)

- 로컬 PC에서 커밋하면 로컬저장소에 반영되고, 로컬저장소에서 푸쉬(Push)하면 원격저장소에 반영
- **장점**
  - 하나의 파일에 대해 작업자별 다른 작업, 별도 이력관리 가능 → 동시다발 작업
  - 원격저장소 다운 시 로컬저장소로 작업 가능
  - 로컬저장소에 올리기 때문에 파일 유실 염려 저하, 작업이력 관리 효율 증대, 속도 향상
- **단점**
  - 직관적이지 못하고 개발자가 적응하는데 시간이 소요됨

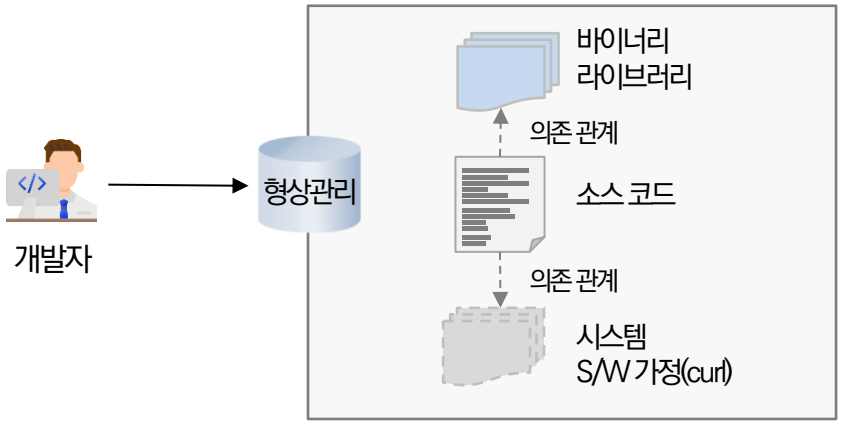
12가지 요소 : 2) 종속성

Code
종속성

종속관계에 있는 모든 애플리케이션은 명시적으로 종속성을 선언하며, 애플리케이션이 필요로 하는 라이브러리를 종속성 파일에 명시적으로 선언하여 사용함

AS-IS : 소스코드 내 시스템 및 라이브러리 포함

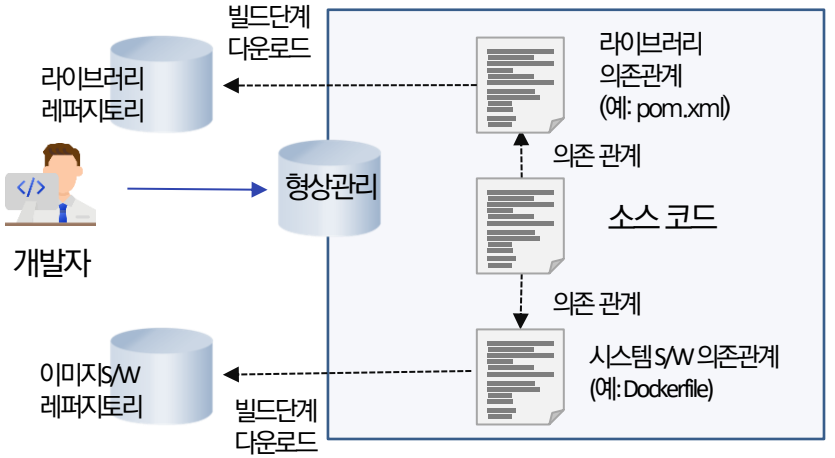
종속성이 라이브러리가 암묵적으로 포함



- 어플리케이션이 의존하는 라이브러리 및 시스템 S/W 가 암묵적으로 관리되어 버전 관리 및 환경 구성이 어려움
- 개발에 새롭게 참여하는 개발자의 환경설정은 재 구성하여야 함
- 애플리케이션 실행 시 종속성 분리(Dependency Isolation) 도구를 사용하지 않음
- 라이브러리 통합관리가 제공되지 않아, 다양한 플랫폼간 이식성을 제공하지 못함

To-BE : 종속성이 있는 라이브러리 통합관리

명시적 의존 관계 관리



- 애플리케이션의 모든 종속성을 명시적으로 선언하여 사용
- **애플리케이션이 필요로 하는 라이브러리를 종속성 매니페스트 파일에 명시적으로 선언하여 사용**
- 클라우드 네이티브 애플리케이션은 다양한 환경에 배포될 수 있으며 각 환경에서 정상 동작하도록 보장
- 모든 종속관계는 종속성 선언을 통해 완전하고 정확하게 이루어짐
- 반복적인 배포 지원

종속성 선언 도구 활용 예시

Maven, Gradle, Node npm 등과 같은 종속성 관리 및 빌드 도구를 활용하여 종속성을 관리하도록 함

종속성 관리 및 빌드 도구를 활용한 종속성 선언

라이브러리 A

Pom.xml

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring</artifactId>
  <version>2.5.5</version>
  <exclusions>
    <exclusion>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Maven

라이브러리 B

Build.gradle

```
dependencies {
    classpath 'com.android.tools.build:gradle:4.2.0'
}
```

Gradle

라이브러리 C

Node.js npm

```
{
  "name": "A",
  "dependencies": {
    "B": "^1.0.0",
    "C": "^2.0.0"
  }
}
```

node.js + npm

- 라이브러리 종속성 관리 및 빌드 도구의 사용
- 애플리케이션 소스코드는 코드베이스, 라이브러리는 라이브러리 저장소에서 관리되어야 함
- 모든 라이브러리는 라이브러리 저장소에서 관리되고 내려 받을 수 있어야 함
- **특정 버전 관리 중요**
- Node.js 패키지, Java .jar, .Net의 DLL과 같은 외부 아티펙트 파일은 실행 시 메모리에 로드된 종속성 선언 매니페스트에서 참조됨

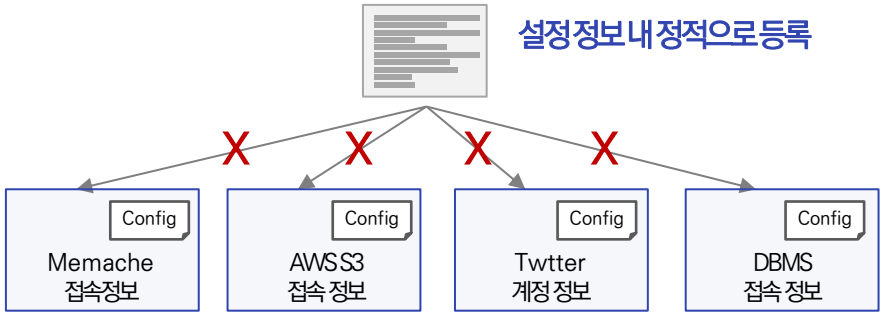
12가지 요소 : 3) 설정

설정

애플리케이션 실행 시 필요로 하는 설정 정보와 코드는 분리하여 관리
(예: 업무처리 로직과는 무관한 시스템 내외부의 리소스, 배포단계, OS 등에 따라 달라지는 설정정보)

AS-IS : 코드내 설정정보가 존재

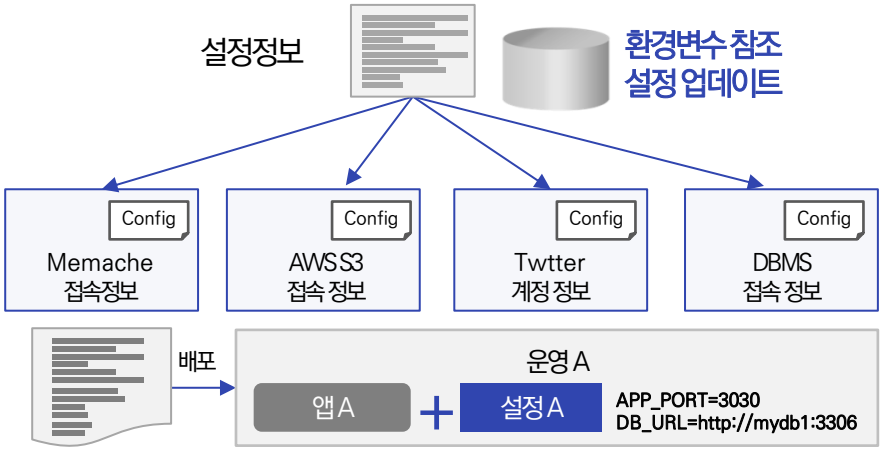
**코드 및 Properties 파일 내
애플리케이션 실행과 관련된 정적인 설정정보**



- 설정을 애플리케이션 코드의 외부에 별도로 두지 않음(소스코드 내부에 위치)
- 환경변수에 설정값 정의
- 애플리케이션 코드는 배포 환경과 관계없이 동일하게 구성할 수 없음

To-BE : 애플리케이션 코드와 설정의 엄격한 분리

**환경변수를 참조하여 환경에 따른
업데이트 가능하도록 설정정보를 정의**



- 배포환경을 위한 설정 파일 작성 시 Spring Cloud Config 사용
- 코드베이스는 여러 환경에 배포되며, 환경에 따라 DB, 스토리지, 기타 클라우드 리소스 정보가 달라짐
- 마이크로서비스 환경에서는 git(spring-cloud-config)와 같은 소스 제어에서 애플리케이션에 대한 설정을 관리

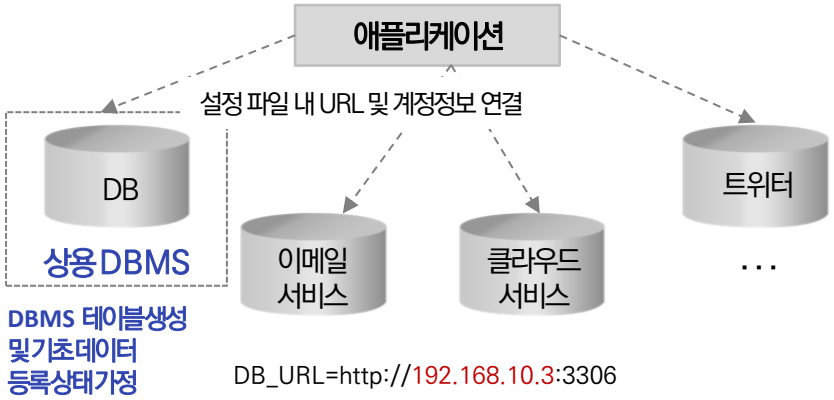
12가지 요소 : 4) 백앤드 서비스

백앤드 서비스

DB, 캐시, 메시지/큐, 이메일 서비스, SNS, 클라우드 서비스 등 백엔드 서비스는 애플리케이션에 연결된(Attached) 리소스로 취급됨

AS-IS : 기존 백엔드 서비스

애플리케이션에서 백엔드 서비스 정적인 환경으로 관리

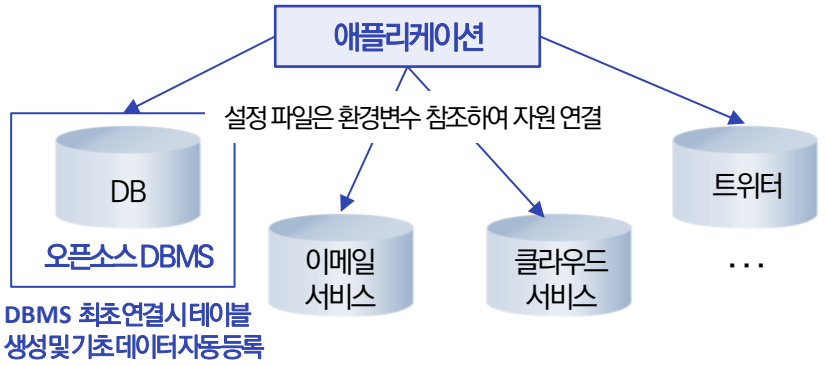


백엔드 서비스 유형
- 데이터베이스: Oracle, MySQL, CouchDB, MariaDB 등
- 메시지/큐: RabbitMQ, Beanstalkd 등
- 이메일(SMTP)서비스 Postfix 등
- 캐시 시스템: Memcached 등

- 어플리케이션 백엔드는 사전 준비된 연결 설정이 마쳐진 상태 (예: DBMS 기초 데이터 생성)
- 백엔드 서비스를 설정파일 내 정적 URL 및 계정정보로 관리

To-BE : 클라우드 네이티브 백앤드 서비스

애플리케이션에서 백엔드 서비스는 언제든 연결되고 변경될 수 있는 연결된 리소스로 관리



설정파일	변경	설정파일
APP_PORT=3030 DB_URL=http://dev:3306	→	APP_PORT=3030 DB_URL=http://prd:3306

"설정파일이나 Property 파일에 접속정보(URL, Locator) 포함"

- 애플리케이션연결된 DBMS의 변경시 애플리케이션 코드변경 없어야 함
- 독립적인 설정 파일에 백엔드 서비스 연결 정보 포함
- 설정파일의 해당 내용 변경만으로 리소스를 변경하도록 함

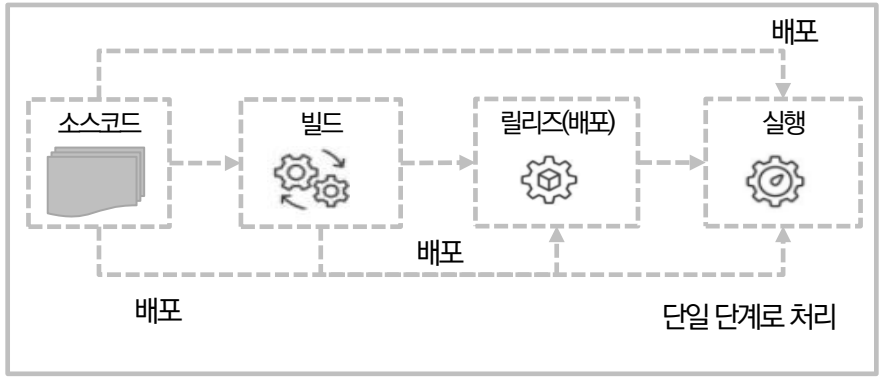
12가지 요소 : 5) 빌드·릴리즈·실행 환경

Code
빌드·릴리즈·실행 환경

코드베이스는 엄격하게 구분된 빌드, 릴리즈, 실행 3단계의 과정을 통해 배포가 이뤄지며, 각 단계는 엄격하게 분리되어야 함

AS-IS : 비순차적, 미분리된 실행환경

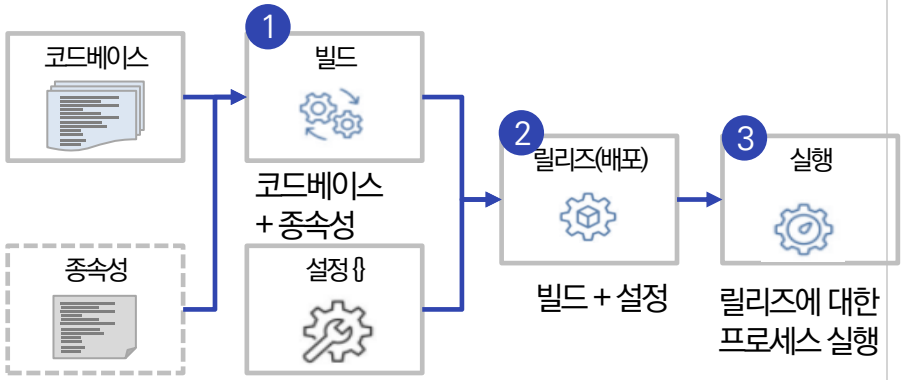
단계가 명확히 구분되지 않는 환경



- 빌드 단계, 릴리즈 단계, 실행 단계에 대한 명확한 구분이 없음
- 순차적인 단계에 따르지 않고, 직접 빌드/배포하는 경우가 다수임

To-BE : CI/CD기반 순차적 실행 환경

CI/CD 파이프라인 기반의 태스크 수행



1 빌드 단계
- 코드 저장소를 빌드라고 불리는 실행가능한 번들로 변환
- 배포 과정에서 지정된 특정 커밋 시점의 코드를 사용해 의존성을 설치하고 바이너리와 에셋 파일을 컴파일

2 릴리즈 단계
- 빌드 단계에서 생성된 빌드를 넘겨받아 현재 배포의 설정과 결합
- 고유한 릴리즈 아이디를 할당하며 변경될 수 없으며, 변경 시 새로 만들어 짐

3 실행 단계
- 특정한 릴리즈에서 필요한 프로세스들을 컨테이너로 실행

빌드·릴리즈·실행 환경 도구 예시

단계	수행 주체	수행 도구	설명
설계	Dev	Spring/Spring Boot, Gradle, Maven	• 개발자가 의존성에 대한 이해도가 가장 높음
1 개발 + **빌드단계**	CI	.war 또는 .jar생성	• **빌드단계에서는 코드베이스의 소스코드와 종속성(라이브러리)를 내려받아 컴파일한 후 하나의 패키지를 만듦** • 1번의 빌드로 다수 배포 • 빌드는 개발자에 의해 실행됨 • 빌드 시 오류는 개발자에 의해 해결 가능
2 **릴리즈 단계**	플랫폼	• Droplet, 도커 이미지	• **릴리즈단계에서는 빌드된 패키지에 환경 설정 정보를 조합** • **모든 릴리즈는 고유의 릴리즈 ID를 가지며, 변경 불가 (변경 시 새로운 릴리즈 ID 부여)** • 민첩한 배포, 업그레이드, 롤백 • 배포도구에서 릴리즈 기능 제공
3 **실행 단계**	플랫폼	• 컨테이너 + 프로세스	• **실행단계에서는 릴리즈에서 만들어진 결과물은 실행환경에서 애플리케이션으로 실행** • 속도(Speed) 중요 • 운영 담당자, 재부팅, 프로세스 재실행 등에 의해 실행 – 자동으로 실행될 수 있으므로 실행 단계의 변경 작업 최소화 필요

• 설계, 개발, 릴리즈, 실행 단계는 엄격하게 구분됨
– 실행 시 변경된 코드를 빌드 단계로 역전파할 수 없으며, 이러한 코드 수정은 불가능

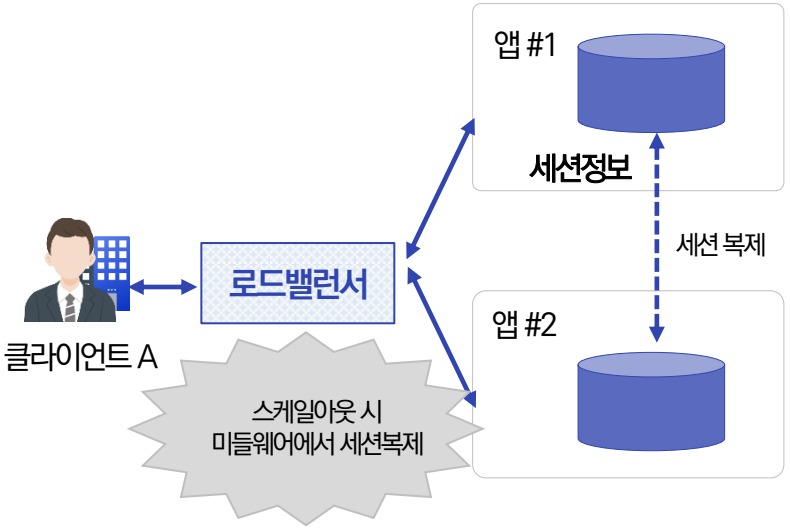
12가지 요소 : 6) 무상태 서비스

Code
무상태 서비스

상태 프로세스는 클라이언트와 세션 정보를 서버에 저장하므로 스케일아웃 시 관련 정보의 이동 작업이 필요한 반면, 무상태 프로세스는 서버에 저장하지 않고 외부 DB에 저장하여 스케일아웃이 용이한 구조임

AS-IS : 상태(Stateful) 프로세스

클라이언트와 서버간 세션의 상태(상태 정보 저장)에 기반하여 클라이언트에 응답을 보냄



- 클라이언트와 세션 정보를 서버에 저장함
- 스케일아웃 시, 클라이언트와 세션 정보를 옮겨주는 부수적 작업 필요

To-BE : 신규 무상태(Stateless) 프로세스

서버는 클라이언트, 서버간 세션 상태와 독립적으로 클라이언트에 응답을 보냄

- 애플리케이션은 실행환경에서 1개 이상의 프로세스로 실행되며, 각 프로세스는 무상태(Stateless)로 어떠한 것도 공유하지 않아야 함
- 클라이언트와 세션 정보를 서버에 저장하지 않고, 외부 DB에 저장함
- 서버에 클라이언트와 세션 정보가 없으므로 스케일아웃이 가능한 구조

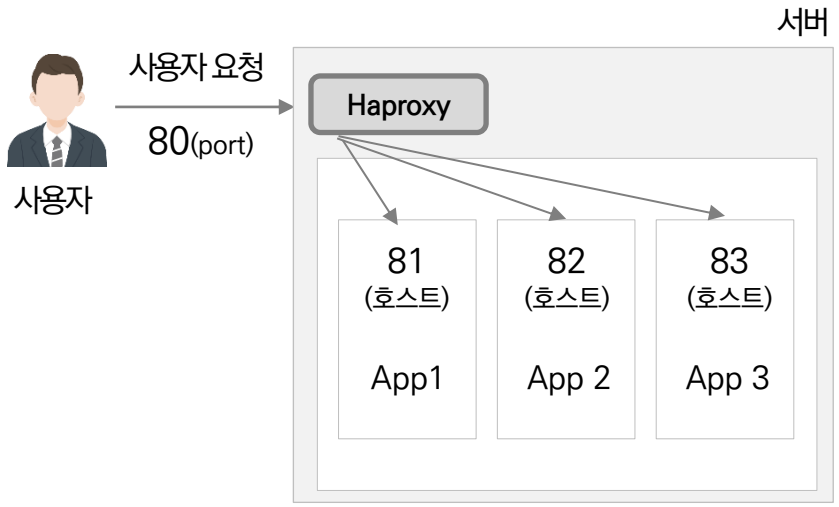
12가지 요소 : 7) 포트 바인딩

포트 바인딩

포트 바인딩은 메시지를 송수신하는 위치와 방법을 결정하는 구성 정보을 의미하며, 배포된 애플리케이션을 타 애플리케이션에서 접근할 수 있도록 포트 바인딩을 통해 서비스를 공개함(서비스 공개 및 접근성 제공)

AS-IS : 온프레미스 환경의 앱 포트 수동 정의

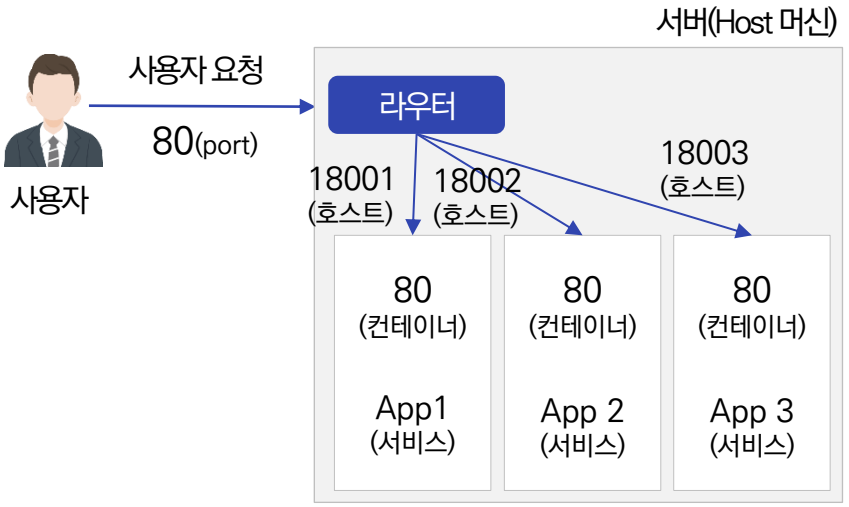
포트 충돌이 되지 않도록 배포 시 수동 관리



- 온프레미스에서 복수의 WAS 실행 시 서로 다른 포트로 실행
- 도메인 분기 처리를 위한 L7 라우터를 배치
- **서버 환경에 따라 배포 시 앱의 포트 변경이 필요**

To-BE : 클라우드 네이티브 앱의 포트 바인딩

플랫폼에서 포트 바인딩을 통해 자동으로 매핑 처리



- **호스트 포트와 컨테이너 포트간 바인딩을 플랫폼에서 처리**
- **사용자 도메인과 컨테이너 호스트 포트간 매핑을 플랫폼에서 처리**
- 사용자는 앱 서비스를 위한 도메인 지정만으로 서비스 라우팅을 요청할 수 있음

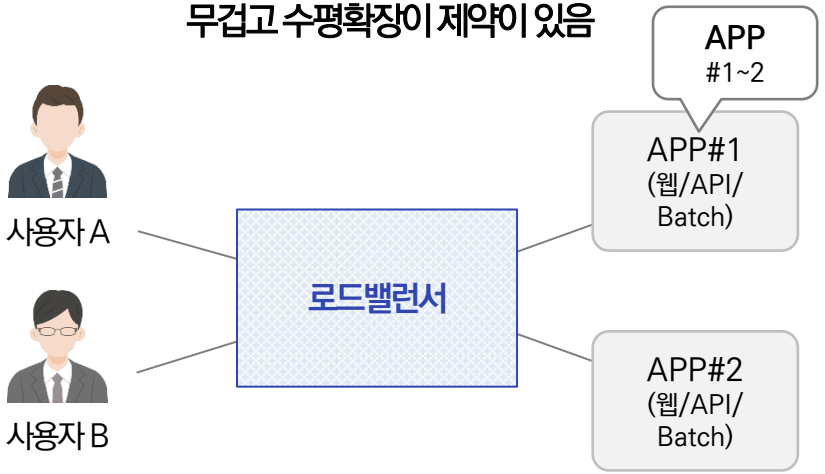
12가지 요소 : 8) 동시성(확장성 포함)

동시성

모든 일을 처리하는 하나의 프로세스 대신 기능별로 분리된 마이크로 프로세스를 실행하며, 프로세스 모델을 기반으로 수직적(Scale-up) 및 수평적(Scale-out) 확장성을 제공함

AS-IS : 모든 일을 처리하는 하나의 프로세스

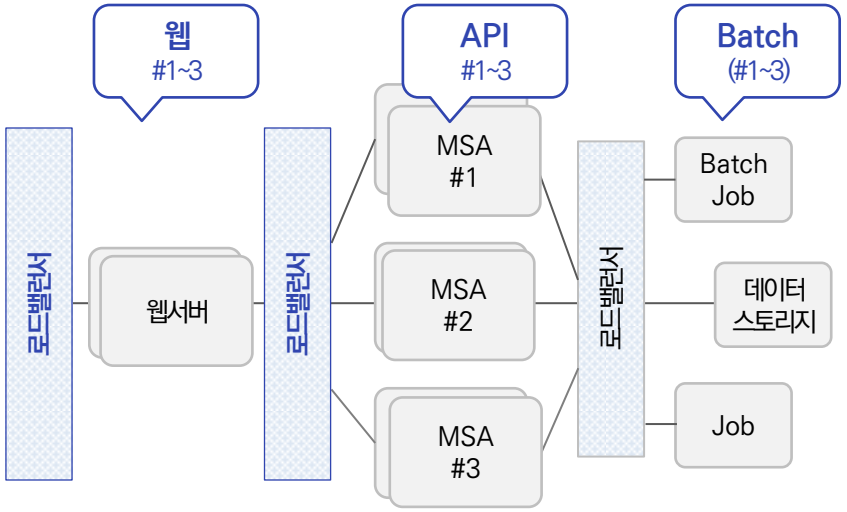
모든 일 (웹, API, Batch)을 처리하는 하나의 프로세스로 배포 무겁고 수평확장이 제약이 있음



- 하드웨어 또는 소프트웨어 기반 부하분산 (L4)
- 전동적인 웹로직과 같은 미들웨어 기반으로 웹, API, 배치 어플리케이션을 통합하여 배포하는 방식
- 특정 유형의 프로세스 스케일 아웃이 어렵고 전체 노드를 증설하거나 스케일 업 필요

To-BE : 프로세스 유형별 수평적 확장이 가능한 구조

수직적 (Scale-up) 확장 및 수평적 (Scale-out) 확장 제공



- 소프트웨어 기반 L4, L7기능 제공
- 프로세스 유형의 확장, 워크로드의 다양성, 마이크로서비스 지원
- 동시성을 지원하는 경우에는 해당 요청사항을 충족하도록 애플리케이션 유형 별 수평적으로 확장할 수 있음
- 비즈니스 서비스에 대한 부담으로 병목현상 발생 시 해당 계층을 독립적으로 확장 가능

12가지 요소 : 9) 폐기 가능성

Code
폐기 가능성

MSA는 각종 요청에 의해 스케일 업/다운이 빈번히 발생하므로 프로세스는 Shutdown 신호를 받았을 때 정상종료해야 함 .
빠른 시작(Start-up) 및 정상 종료(Graceful Shutdown)로 안정성 극대화

AS-IS : 기존 프로그램 처리

미들웨어 담당자에 의한 중지 및 기동으로, 정기 PM 시간에 기동 및 중지. 자원 초기화 및 종료를 모니터링 및 관리

- 미들웨어가 제공하는 기동/중지 시그널을 고려하여 앱의 설계 필요
- 미들웨어 관리자에 의해 기동/종료가 관리되고 리소스 초기화를 점검

To-BE : 빠른 시작과 정상종료가 가능한 폐기 가능성 처리

앱 부하 조건에 따라 자동적인 앱 스케일 발생하며, 애플리케이션 기동/중시 시 자원 관리가 자동으로 처리

- 플랫폼이 제공하는 기동/중지 시그널을 고려하여 앱의 설계 필요
- 어플리케이션은 마이크로서비스로 설계되어 수초 이내의 빠른 시작과 종료
- 플랫폼에 의해 자동으로 관리되며, 어플리케이션은 자동화된 자원 관리 설계 고려

애플리케이션 종료 시 DB 연결과 자원 사용 종료 예시

애플리케이션 사용 종료 시 모든 DB 연결 및 각종 리소스 사용이 올바르게 종료되고, 모든 종료 활동이 기록되어야 함

애플리케이션 사용 종료 시 DB 연결과 자원 사용 종료

```
const shutdown = async (signal) => {
   logger.info(`Disconnecting message broker at ${new Date()}`);
   messageBroker.disconnect();

   logger.info(`Disconnecting database at ${new Date()}`);
   database.disconnect();

   let shutdownMessage;

   if (!signal) {
       shutdownMessage = (`MyCool service shutting down at ${new
Date()}`);
   } else {
       shutdownMessage = (`Signal ${signal} : MyCool service
shutting down at ${new Date()}`);
   }
   const obj = {status: "SHUTDOWN", shutdownMessage, pid:
process.pid};
   await server.close(() => {
       console.log(obj);
       process.exit(0);
   });
};
```

- **Disposability의 원칙**
  - 애플리케이션이 신속하게 시작되고 정상적으로 종료되어야 함
- **애플리케이션의 시작**
  - 사용자가 애플리케이션에 접근하기 전에 DB 연결, 네트워크 리소스에 대한 액세스, 기타 구성 작업 등이 수행되어야 함
- **애플리케이션의 종료**
  - 사용자가 애플리케이션 사용을 종료할 경우 모든 DB 연결 및 각종 리소스 사용이 올바르게 종료되고, 모든 종료 활동이 기록되어야 함

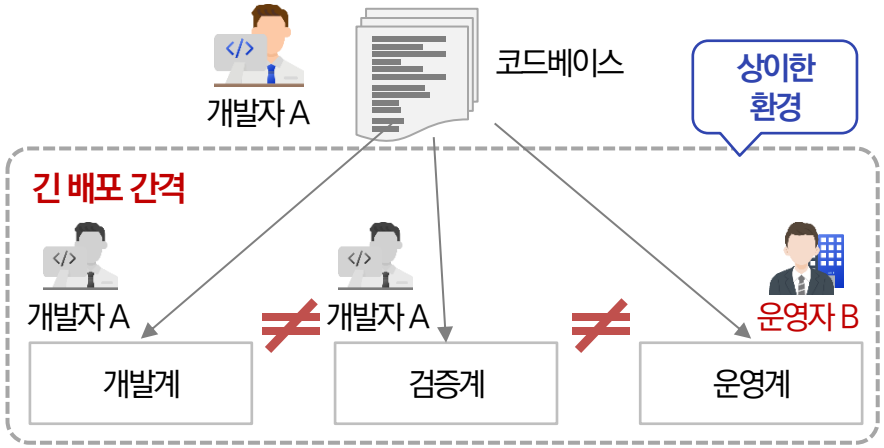
12가지 요소 : 10) 개발/운영 환경 일치

개발/운영 환경

가능한 동일한 개발/검증/운영 환경을 유지하여 장애 최소화 및 지속적 배포가 가능하게 함

AS-IS : 개발/운영 환경

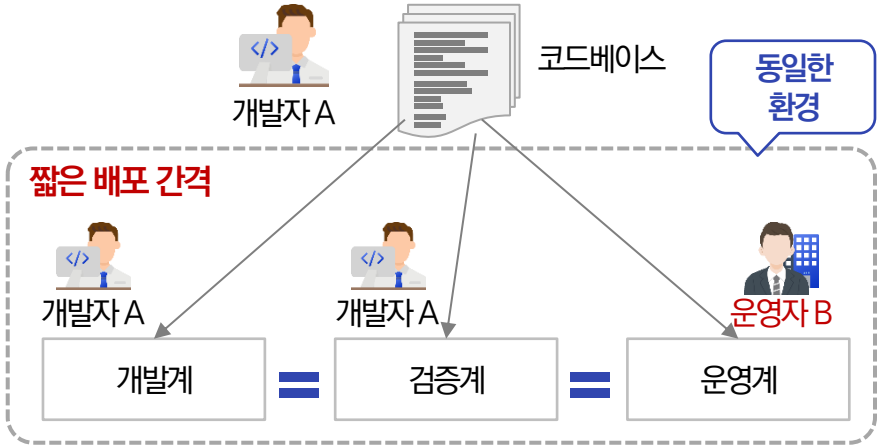
개발환경과 운영환경의 차이



- 코드 개발자와 코드 배포자 상이
  - **운영자가 운영계 배포를 담당**하므로 코드 개발자와 코드 배포자가 다름
- 긴 배포 간격
  - 개발자가 코드 작성 후 개발계, 검증계, 운영계 환경에 반영하기 까지 **수 일에서 수 개월 소요**
- **기존의 개발/운영 환경은 코드 개발자와 배포자가 상이** : 긴 배포 간격, 도구 등의 환경 차이가 존재하여 운영환경에서 장애 발생 가능성이 높음

To-BE : 12 Factor(클라우드 네이티브) 개발/운영 환경

개발, 테스트(검증), 운영 환경을 최대한 동일하게 유지



- 코드 개발자와 코드 배포자 일치
  - 운영자 아니라 코드 **개발자가 직접 배포**
- 짧은 배포 간격
  - 개발자가 코드 작성 후 개발계, 검증계, 운영계 환경에 반영하기 까지, **수 분에서 수 시간 소요**
- 개발 환경과 운영환경의 차이 최소화를 통한 **운영환경에서 예기치 않은 오류/다운타임의 위험성 감소 등**

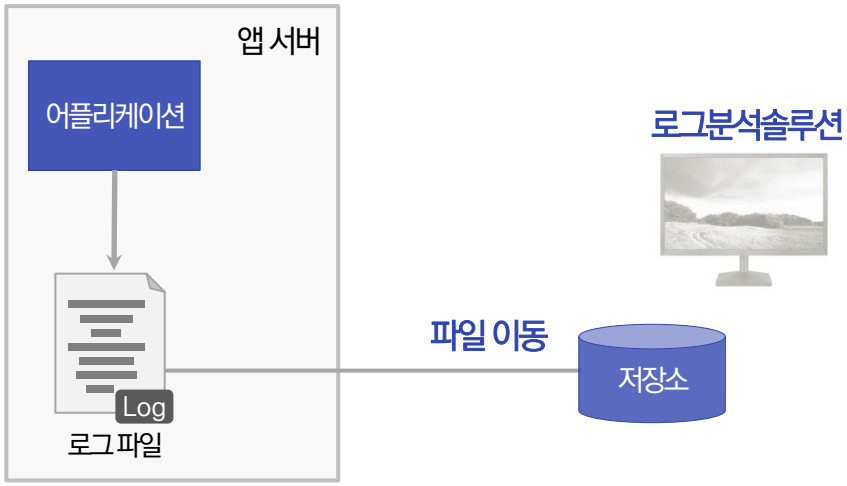
12가지 요소 : 11) 로그

Code 로그

어플리케이션 로그는 파일이 아닌 로그 스트림 형태로 표준 출력하며, 플랫폼에 의해 로그 스트림이 통합되고 관리됨

AS-IS : 기존 로그 수집

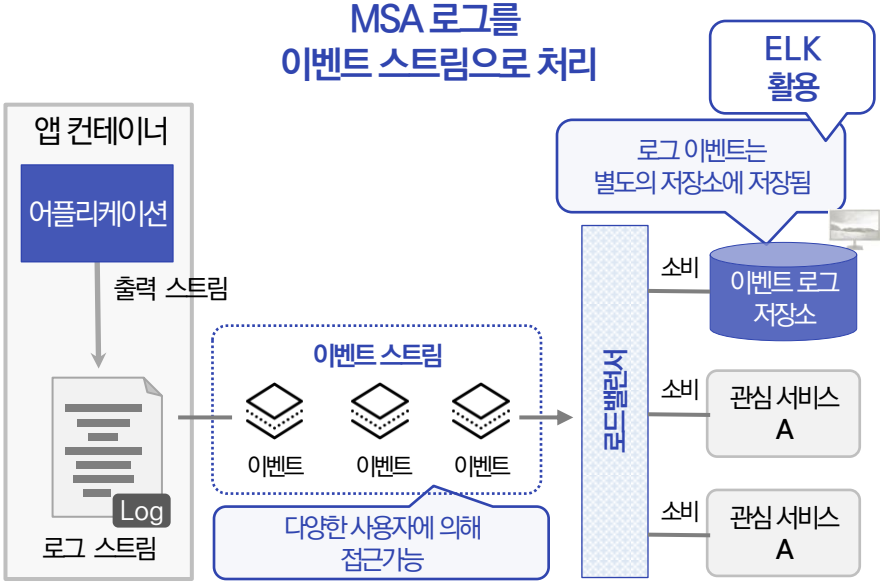
**로그는 로컬 파일로 저장하며,
로컬 파일을 중앙 저장소로 이동하여 통합하고 분석됨**



- 애플리케이션은 로그를 파일로 저장
- 파일 생성 및 로테이션 조건을 어플리케이션 혹은 서버 내 설정으로 관리
- 로그 분석 체계에 의해 로컬 로그 파일을 중앙 저장소로 복사하여 로그 분석 수행
  (애플리케이션에 대한 로그수집을 위한 프로그램 변경 및 Agent P/G설치 등)

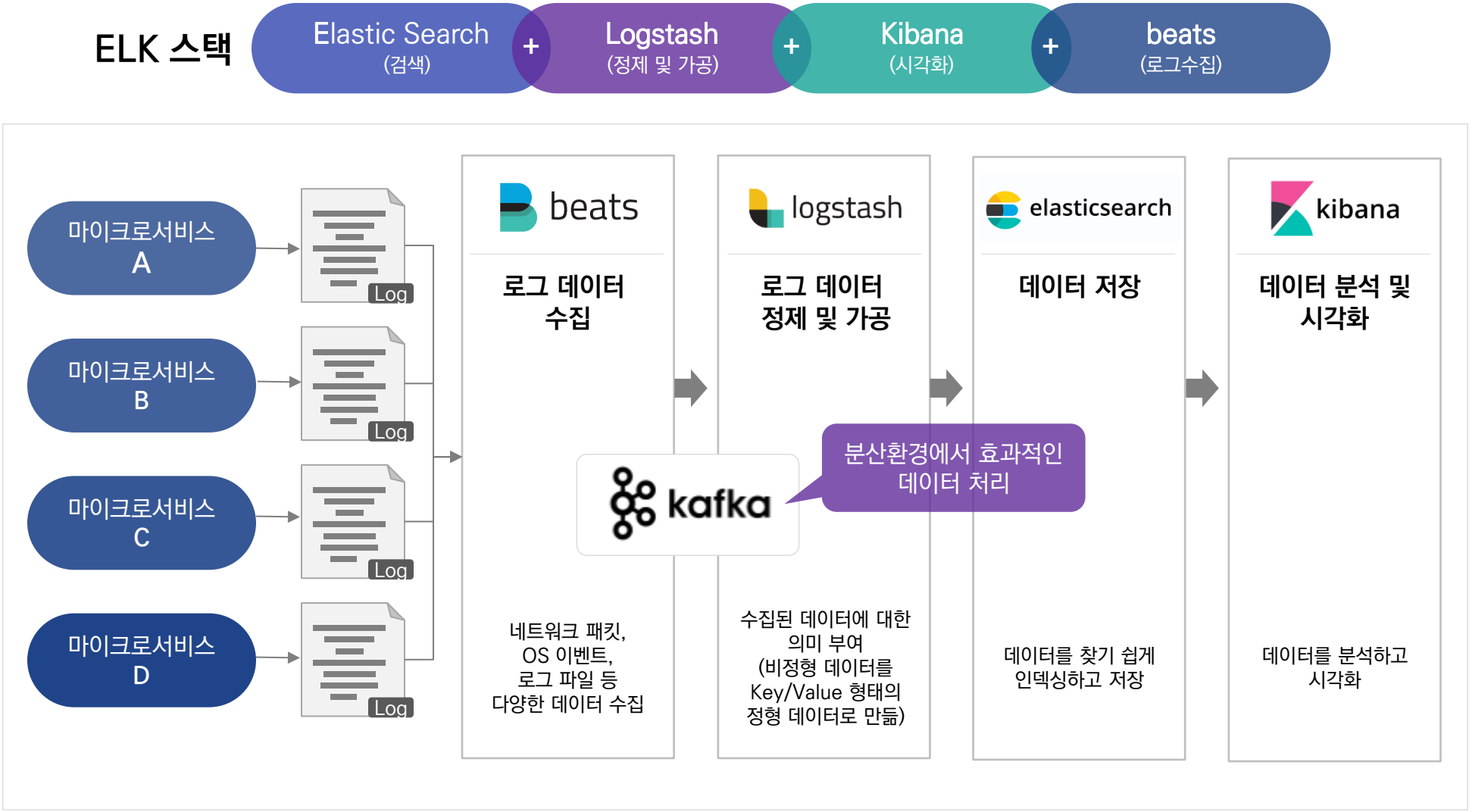
To-BE : ELK기반 로그수집 (Elastic Search, Logstash, Kibana)

**MSA 로그를
이벤트 스트림으로 처리**



- 로그는 실행중인 모든 프로세스와 서비스의 아웃풋 스트림으로부터 수집된 이벤트가 시간순으로 정렬되는 스트림임
- 로그를 이벤트 스트림으로 취급하여 로그를 로컬에 저장하지 않고, 별도의 저장소에 보관
  – 마이크로서비스는 사용량에 따라 변화하므로 인스턴스가 생성/삭제될 수 있으므로 로컬에 로그 저장 시 로그가 초기화될 수 있음

ELK 스택을 이용한 로그 분석 예시



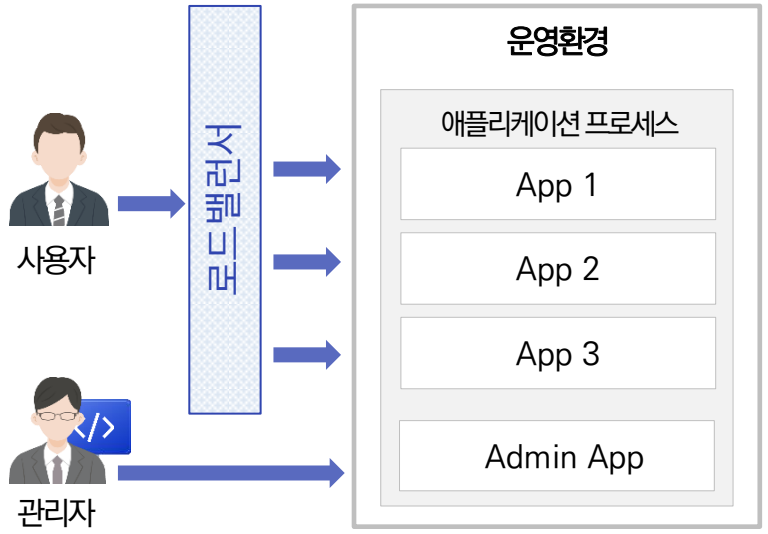
12가지 요소 : 12) 관리 프로세스

관리 프로세스

관리/유지보수 프로세스를 일회성 프로세스로 실행하며, 관리/유지보수 프로세스는 일회성 프로세스로 애플리케이션 프로세스와 분리되어 실행되지만, 애플리케이션과 동일한 빌드/릴리즈/배포 사이클로 실행됨

AS-IS : 프로세스 미분리

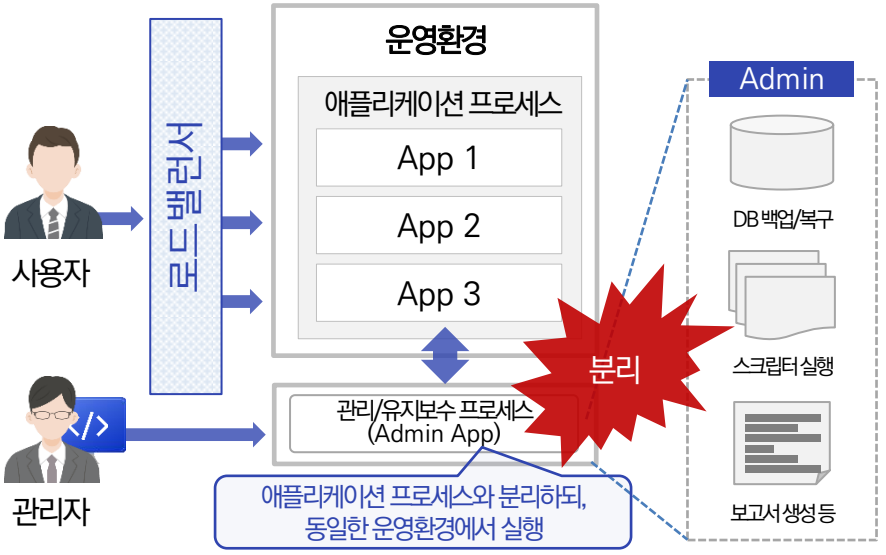
애플리케이션 프로세스와 관리 프로세스 동일환경에서 동작



- 관리/유지보수 프로세스는 애플리케이션 영역과 분리 되어있지 않음
- 관리 프로세스가 운영시스템에 영향을 줄 수 있음

To-BE : 사용자와 관리자 프로세스 분리

관리/유지보수 프로세스는 애플리케이션 영역과 분리되어 동작



- DB 마이그레이션, DB 백업/복구, 스토리지 이동, 스크립트 실행, 보고서 생성 등의 일회성 관리/유지보수 프로세스는 애플리케이션과 분리하되, 어플리케이션과 동일한 환경에서 실행되어야 함
- 일회성 처리를 위한 기능들도 동기화 문제를 해결하기 위하여 코드베이스와 함께 관리하고 빌드/릴리즈/배포 관리되어야 함

루비 및 파이썬의 관리 및 REPL을 활용한 일회성 스크립트 실행 예시

루비 및 파이썬 등의 개발언어 사용 시 관리 프로세스는 웹 프로세스와 같은 애플리케이션이 실행되는 환경과 동일한 곳에서 실행되어야 하고, REPL shell 제공 언어를 이용하여 일회성 관리 프로세스를 실행할 것을 권고함

루비

Ruby

동일한 디렉토리에서 실행

루비 웹 프로세스	$ bundle exec thin start
DB 마이그레이션	$ bundle exec rake db：migrate

파이썬

python

동일한 디렉토리(/bin/python)에서 실행

토네이도 웹 서버 (가상환경)	$ import tornado.web
DB 마이그레이션 (manage.py 사용)	$ python manage.py migrate 〈앱〉

12 Facter App은 별도의 설치나 구성 없이 REPL shell을 제공하는 언어를 선호

REPL

Read	(코드를) 읽고
Eval	(읽은 코드를) 평가(실행) 하고
Print	(실행한 결과를) 출력하는
Loop	루프 (반복)

REPL 은 사용자 입력한 하나의 문장(표현식) 단위로 평가/실행하고 결과를 출력하는 명령줄 셸 및 프로그래밍 언어와 유사한 환경을 의미

- **로컬 배포 시**
  - 개발자는 일회성 관리 프로세스를 어플리케이션을 체크아웃한 디렉토리 안에서 직접 실행
- **운영계 배포 시**
  - 운영계 배포 시에는 SSH나 운영계 환경이 제공하는 다른 원격 도구를 사용해 관리 프로세스를 실행할 수 있음

12가지 요소 적용 – CF vs. Kubernetes PaaS 플랫폼 환경에서 구현

구분		Cloud Foundry	Heroku	Kubernetes
개념		오픈소스 PaaS플랫폼	AWS에서 제공하는 PaaS플랫폼	오픈 소스 컨테이너플랫폼
특징		배포관리, 컨테이너 관리만 보유	빌드관리, 배포관리, 컨테이너 관리 모두 보유	빌드관리, 배포관리, 컨테이너 관리 모두 보유
비교 항목	빌드 관리		빌드관리 (Slug)	빌드관리 (Docker Image)
	배포 관리	배포관리 (Droplet)	배포관리 (Release)	배포관리 (Deployment / StatefulSet)
	컨테이너 관리	컨테이너관리 (Garden)	컨테이너 관리 (Dynos)	컨테이너 관리 (Pod + Docker Container)

12가지 요소 적용 – CF vs. Kubernetes PaaS 플랫폼 환경에서 구현

구분	활용 용도	Cloud Foundry	Kubernetes
1) 코드 베이스	단일 형상관리 기반의 버전 관리 및 배포 관리 자동화 환경 제공	• 어플리케이션의 버전관리를 제공하지 않음(변경 추적을 위한 별도의 버전관리시스템을 적용 필요) • 어플리케이션과 배포환경간 추적관리를 제공하지 않음 (별도 관리를 위한 시스템 구축 필요)	• 전체 시스템을 선언적으로 정의 (Docker 빌드, Deployment 배포) • 기본 Kubernetes는 Git 환경을 포함하고 있지 않음 (변경 추적을 위한 별도의 버전관리 및 GitOps 적용 필요) • GitOps 도구와 통합 서비스 제공 (예 weaveworks gitops)
2) 종속성	의존관계는 명시적으로 선언하고 분리	• 언어별 Buildpack 기본 환경 제공 • 언어별 maven, npm, pip, gem 등 의존 관계 관리	• Dockerfile 이용하여 언어별 기본환경 정의 • 언어별 maven, npm, pip, gem 등 Dockerfile 지원
3) 설정	설정 정보는 환경변수에 저장	$ cf cups myaccess -p '{"uri":"example.com:300"}' $ cf bind-service APP-NAME mysqldb	• ConfigMap, Deployment 내 명시
4) 백앤드 서비스	백엔드 서비스는 리소스 연결로 처리	• Service Broker 연결로 처리	• Service(Cluster IP) 연결로 처리
5) 빌드/ 릴리즈/ 실행	빌드와 실행 단계는 엄격히 분리	• 릴리즈 단계 아티팩트 관리(빌드 단계 관리는 없음) 빌드실행 단계에서 코드 변경 불가	• 빌드 및 릴리즈 단계 아티팩트 분리 빌드 단계 Docker Image 지원 실행 단계 Deployment, StatefulSet 타입의 릴리즈 관리 및 롤백을 지원

12가지 요소 적용 – CF vs. Kubernetes PaaS 플랫폼 환경에서 구현

구분	활용 용도	Cloud Foundry	Kubernetes
6) 프로세스	앱은 Stateless 프로세스로 실행	Stateless 지원 (Web Container는 Session Replication을 지원하지 않음)	Stateless & Stateful 모두 지원
7) 포트 바인딩	웹 어플리케이션은 포트 바인딩을 사용하여 서비스를 공개	공통 라우터에서 L7(HTTP/HTTPS), L4 지원	Ingress 오브젝트 이용하여 L7, L4(TCP) 서비스 지원
8) 동시성	프로세스 유형별 스케일 아웃 지원	단일 배포 유형 제공 설정 파일: manifest.yml 실행: $ cf push –f manifest.yml --- applications: - name: web path: web/build/libs/web.jar instances: 2 - name: worker path: worker/build/libs/worker.jar no-route: true instances: 4	Stateless, Stateful, Job, BatchJob 유형 제공
9) 폐기 가능성	빠른 시작과 정상 종료(graceful shutdown)를 지원하여 안정성을 극대화	기동 시 서비스 Probe 기능 제공(GoRouter Timeout은 300초)	Liveness Probe, Readiness Probe, Graceful Shutdown 기능 제공

12가지 요소 적용 – CF vs. Kubernetes PaaS 플랫폼 환경에서 구현

구분	활용 용도	Cloud Foundry	Kubernetes

10) 개발/실행 환경	가능한 동일한 개발, 스테이징, 운영 환경 유지	$ cf cups mysqldb -p '{"uri":"mysql://root:secret@dbserver.example.com:3306/mydatabase"}'
		$ cf bind-service APP-NAME mysqldb
		소스 코드: Cloud cloud = cloudFactory.getCloud(); DataSource ds = cloud.getServiceConnector("mysqldb", DataSource.class, null /* default config */);
			• Kubernetes 는 서비스간 의존성을 Service 객체로 처리
			• Kubernetes 내에 실행되는 서비스는 ClusterIP 타입 속성의 Service 를 이용하여 하위 서비스와 LB 역할 수행
			• Kubernetes 외에 실행되는 서비스는 ExternalIP 혹은 ExternalName 타입 속성으로 구성하여 외부 서비스와 LB 연결 수행

11) 로그	로그는 이벤트 스트림으로 처리	$ cf logs appname Monitoring/Logging 마켓플레이스	$ kubectl logs podname

12) 관리 프로세스	어드민 및 관리 작업은 일회성 프로세스로 실행	일반 App 처리
		$ cf ssh APP --no-route
		$ cf push APP -c 'rake db:migrate' -i 1
			Job, BatchJob 처리

Kubernetes Native 어플리케이션 환경 소개

Dcoker 기술

Docker는 이미지 빌드, 이미지 저장소, 컨테이너 실행환경으로 구성

Docker 이미지

- 개발 바이너리와 환경 설정을 패키징
- 파일시스템, 실행 파일 경로와 같은 메타데이터 포함
- Dockerfile 로 정의

```
$ cat Dockerfile
FROM node:7
ADD app.js /app.js
ENTRYPOINT ["node", "app.js"]
```

이미지 저장소 (Registry)

- Docker 이미지 공유를 위한 저장소
- 개발자가 Docker 이미지를 업로드(Push)하고, Docker 실행 시 다운로드(Pull)
- 공개 Docker 저장소(hub.docker.com)와 사설 Docker 저장소 이용 가능

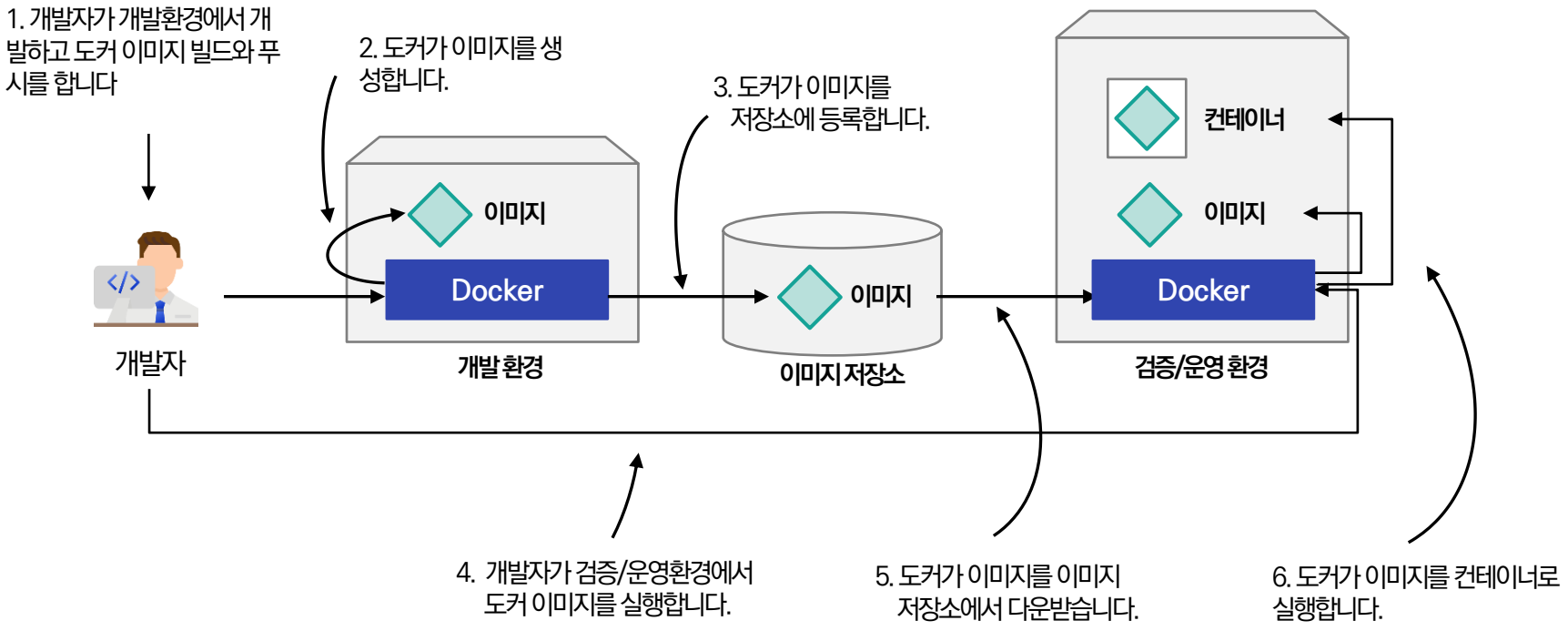
컨테이너 실행환경

- Docker 이미지를 기반으로 실행되는 일반 Linux 컨테이너
- 컨테이너는 Docker 실행 호스트에서 실행되는 하나의 프로세스
- 컨테이너는 호스트 및 타 프로세스와 격리

Kubernetes Native 어플리케이션 환경 소개

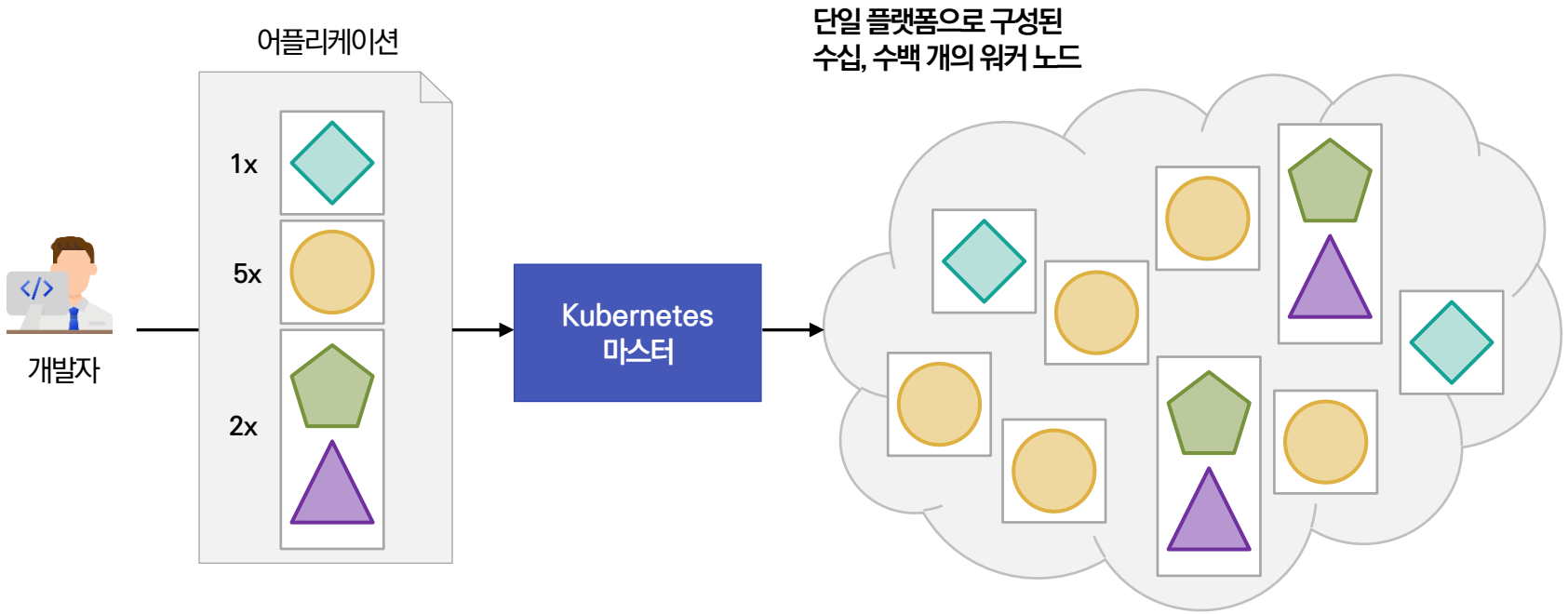
Dcoker 기술

Docker는 이미지 빌드, 이미지 저장소, 컨테이너 실행환경으로 구성



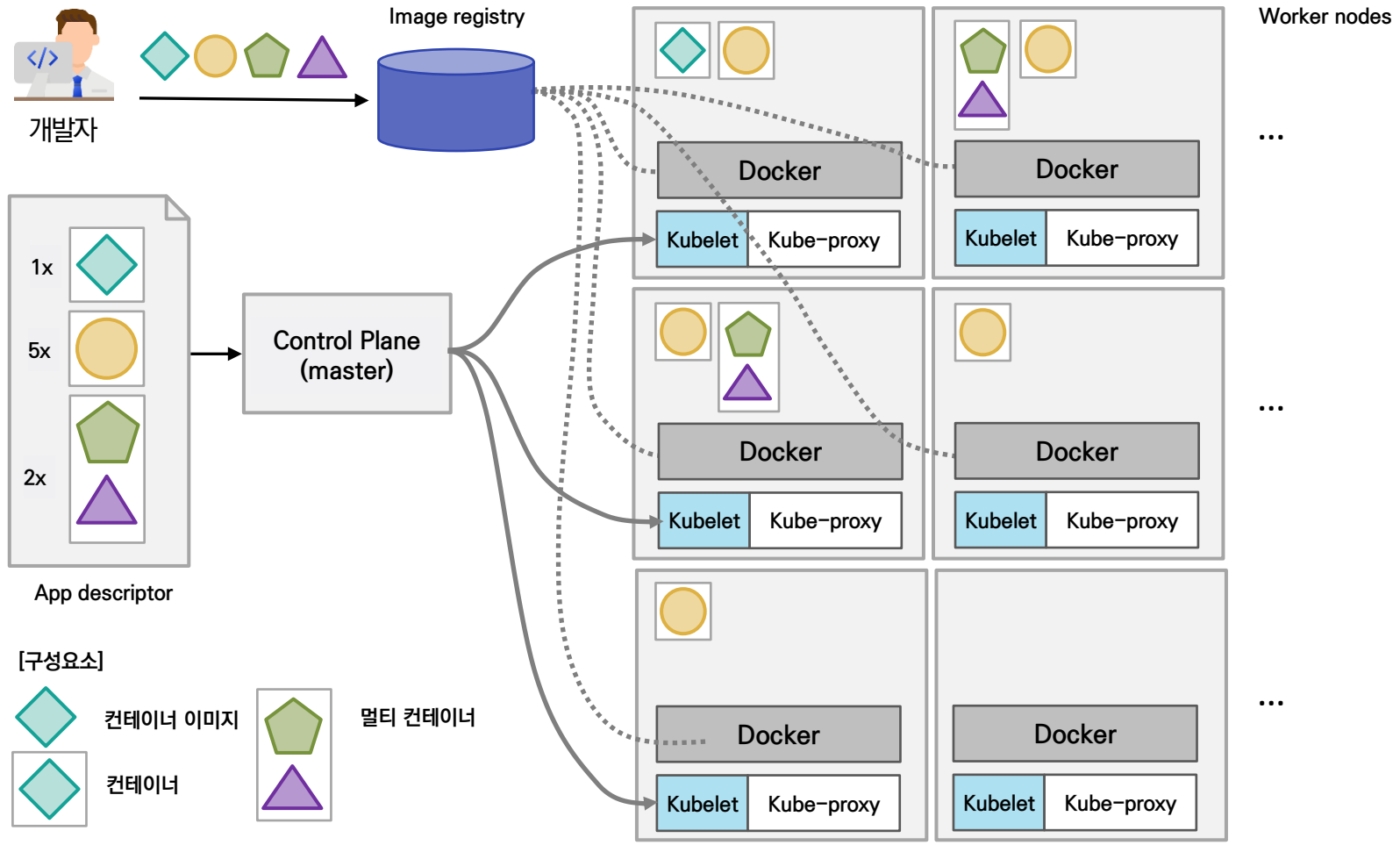
Kubernetes Native 어플리케이션 환경 소개

Kubernetes Kubernetes 는 복수의 Docker 실행환경을 통합 관리하는 PaaS 표준 플랫폼



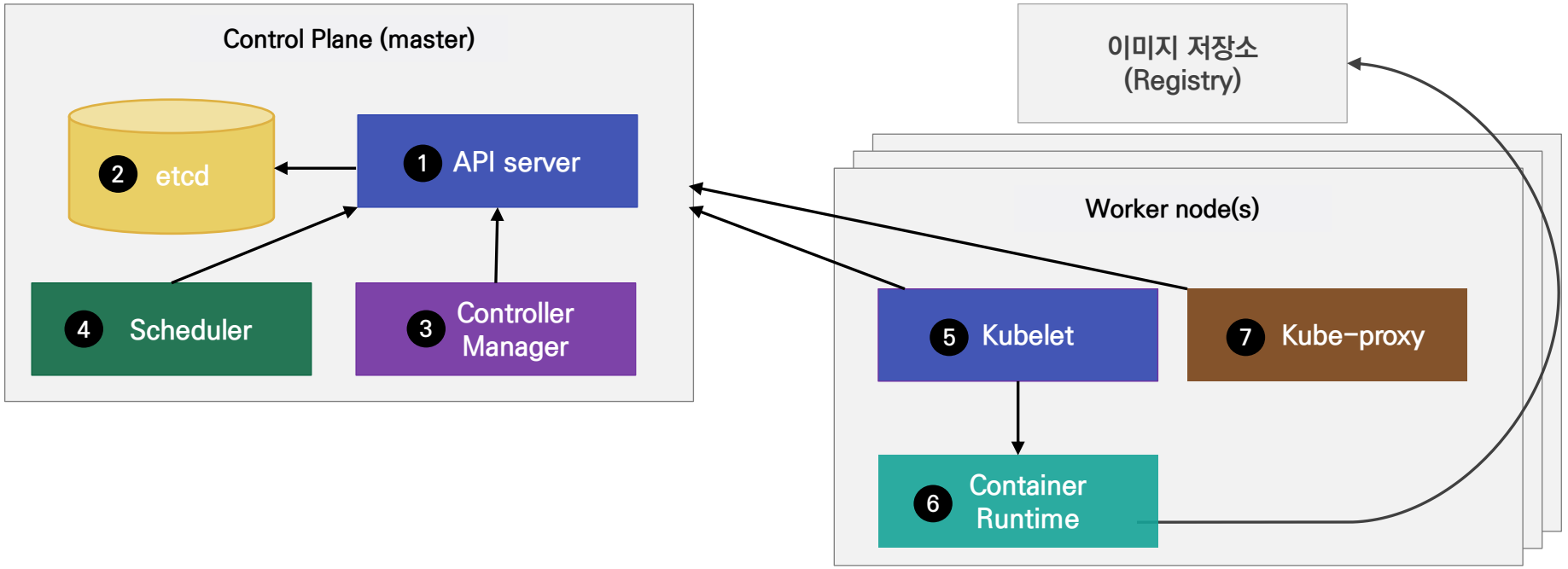
Kubernetes Native 어플리케이션 환경 소개

Kubernetes Kubernetes는 Master와 Worker 노드로 구성, 사용자 정의 앱의 배포와 운영을 수행



Kubernetes Native 어플리케이션 환경 소개

Kubernetes Kubernetes는 Master와 Worker 노드로 구성, 사용자 정의 앱의 배포와 운영을 수행



1. API Server : 클라이언트에 API 진입점을 제공하고 K8S 개체를 관리
2. etcd : 클러스터 정보를 저장하는 고가용성 키-값 저장소
3. Controller Manager : Pod 및 Service 생성 요청을 감시하고, 생성하는 역할
4. Scheduler : 노드가 배정되지 않은 Pod를 감지하고 노드를 지정
5. Kubelet : 노드에 할당된 Pod, 볼륨 관리
6. Container Runtime : Docker 이미지 다운받아 실행
7. Kube-proxy : Pod 접속을 위한 Service 네트워크 처리

Kubernetes Native 어플리케이션 환경 소개

Kubernetes

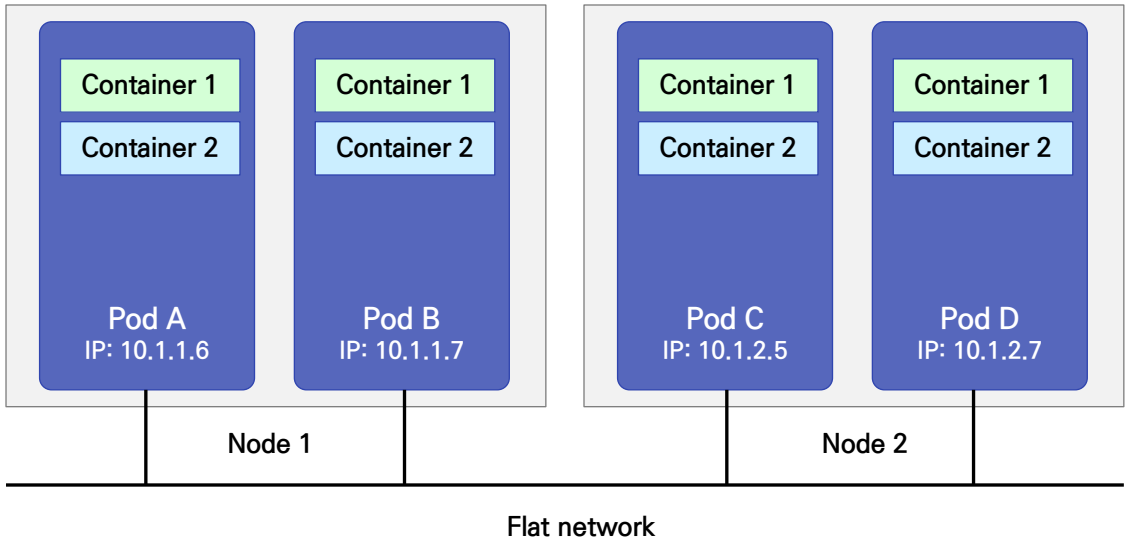
Kubernetes 제공 리소스 유형

리소스	용도	리소스	용도
Node	컨테이너가 배치되는 서버	Storage Class	퍼시스턴트 볼륨이 확보하는 스토리지의 종류를 정의
Namespace	쿠버네티스 클러스터 안의 가상 클러스터	StatefulSet	상태를 유지하는 파드를 생성 관리
Pod	컨테이너 집합 중 가장 작은 단위, 컨테이너의 실행 방법을 정의	Job	상주 실행을 목적으로 하지 않는 파드를 생성하고 종료
ReplicaSet	같은 스펙을 갖는 파드를 여러 개 생성하고 관리하는 역할	CronJob	Cron 문법으로 스케줄링되는 잡
Deployment	레플리카 세트의 리비전을 관리	Secret	인증 정보 같은 기밀 데이터를 정의
Service	파드의 집합에 접근하기 위한 경로 정의	Role	네임스페이스 안에서 조작 가능한 쿠버네티스 리소스의 규칙을 정의
Ingress	서비스를 쿠버네티스 클러스터 외부로 노출	RoleBinding	쿠버네티스 리소스 사용자와 롤을 연결
ConfigMap	설정 정보를 정의하고 파드에 전달	ClusterRole	클러스터 전체에서 조작 가능한 쿠버네티스 리소스의 규칙을 정의
Persistent Volume	파드가 사용할 스토리지의 크기 및 종류를 정의	ClusterRole Binding	쿠버네티스 리소스 사용자와 클러스터롤을 연결
Persistent VolumeClaim	퍼시스턴트 볼륨을 동적으로 확보	Service Account	파드가 쿠버네티스 리소스를 조작할 때 사용하는 계정

Kubernetes Native 어플리케이션 환경 소개

Pod

Pod 는 컨테이너를 포함하는 배포 단위로, Pod내 컨테이너는 IP와 볼륨을 공유



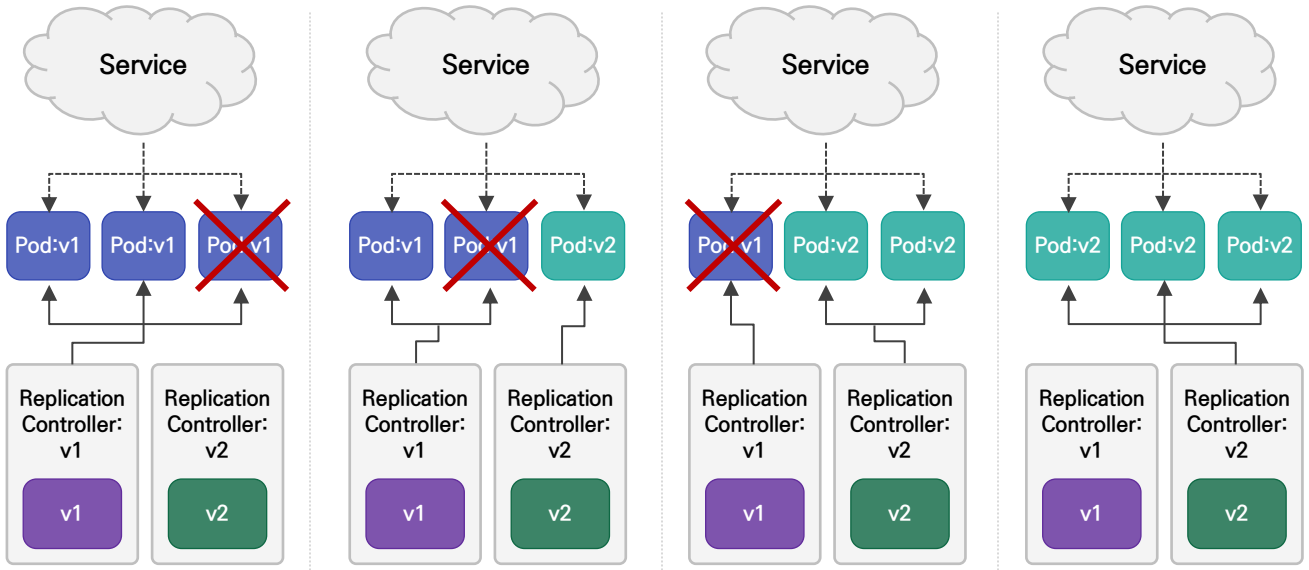

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-manual
spec:
  containers:
  - image: luksa/kubia
    name: kubia
    ports:
    - containerPort: 8080
      protocol: TCP
```

```
$ kubectl create -f kubia-manual.yaml
pod "kubia-manual" created
```

Kubernetes Native 어플리케이션 환경 소개

Deployment

Deployment는 선언적인 Pod 수 관리와 롤링 업데이트 관리를 지원

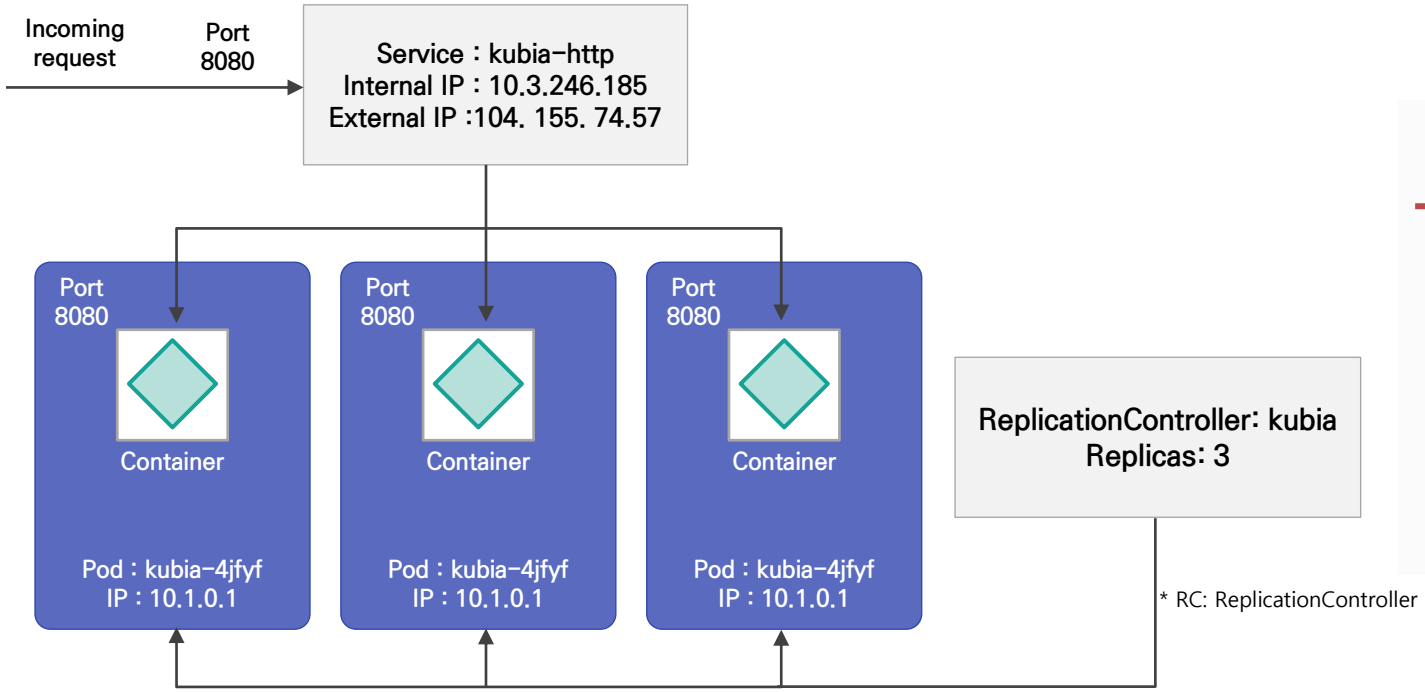



```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
```

Kubernetes Native 어플리케이션 환경 소개

Service

Service는 복수의 Pod 에 라우팅하며, IP와 Port를 가짐. RC*는 Pod 수를 관리



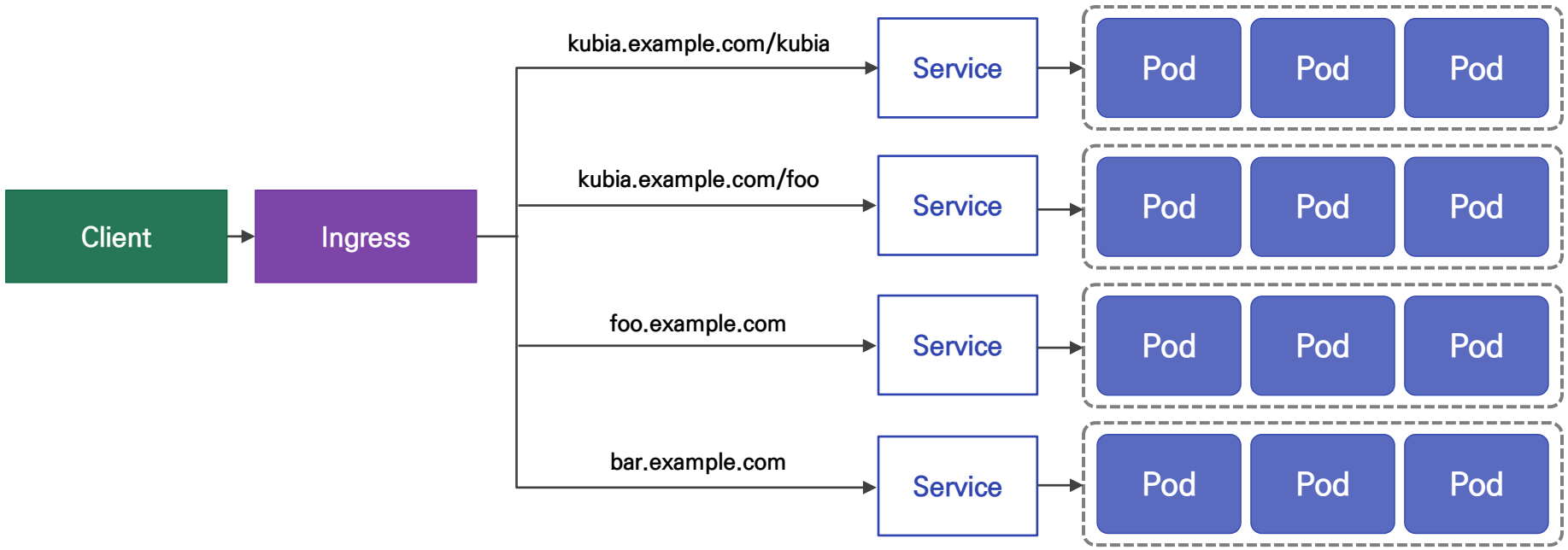

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

* RC: ReplicationController

Kubernetes Native 어플리케이션 환경 소개

Ingress

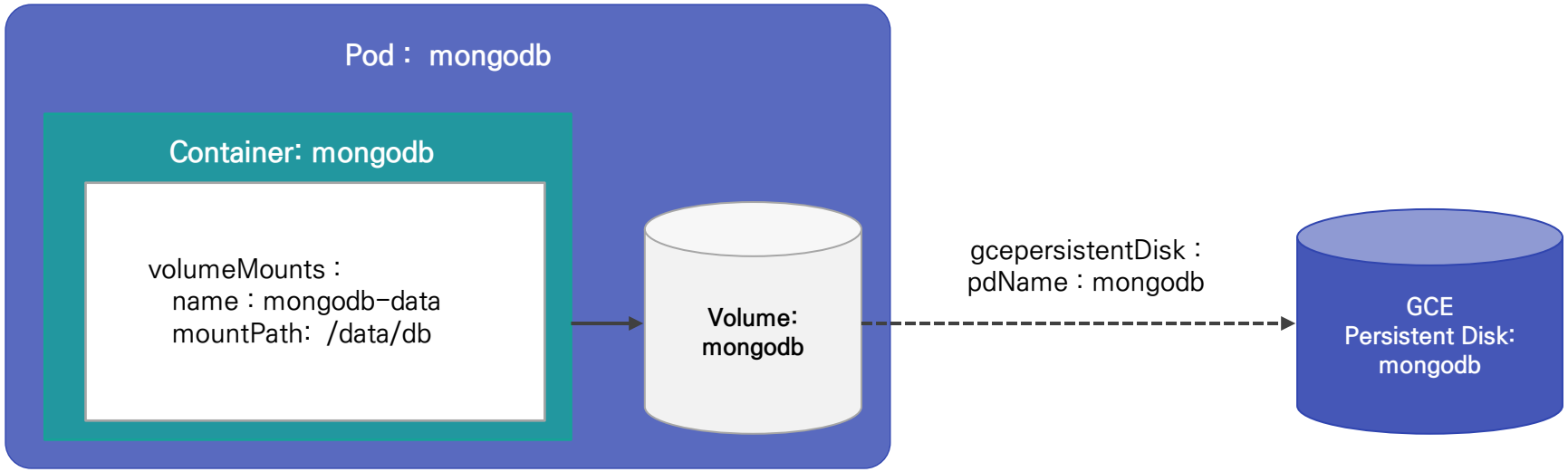
Ingress 는 대외에서 Service 접근을 허용하는 L7 로드밸런서 역할



Kubernetes Native 어플리케이션 환경 소개

Persistent Volume

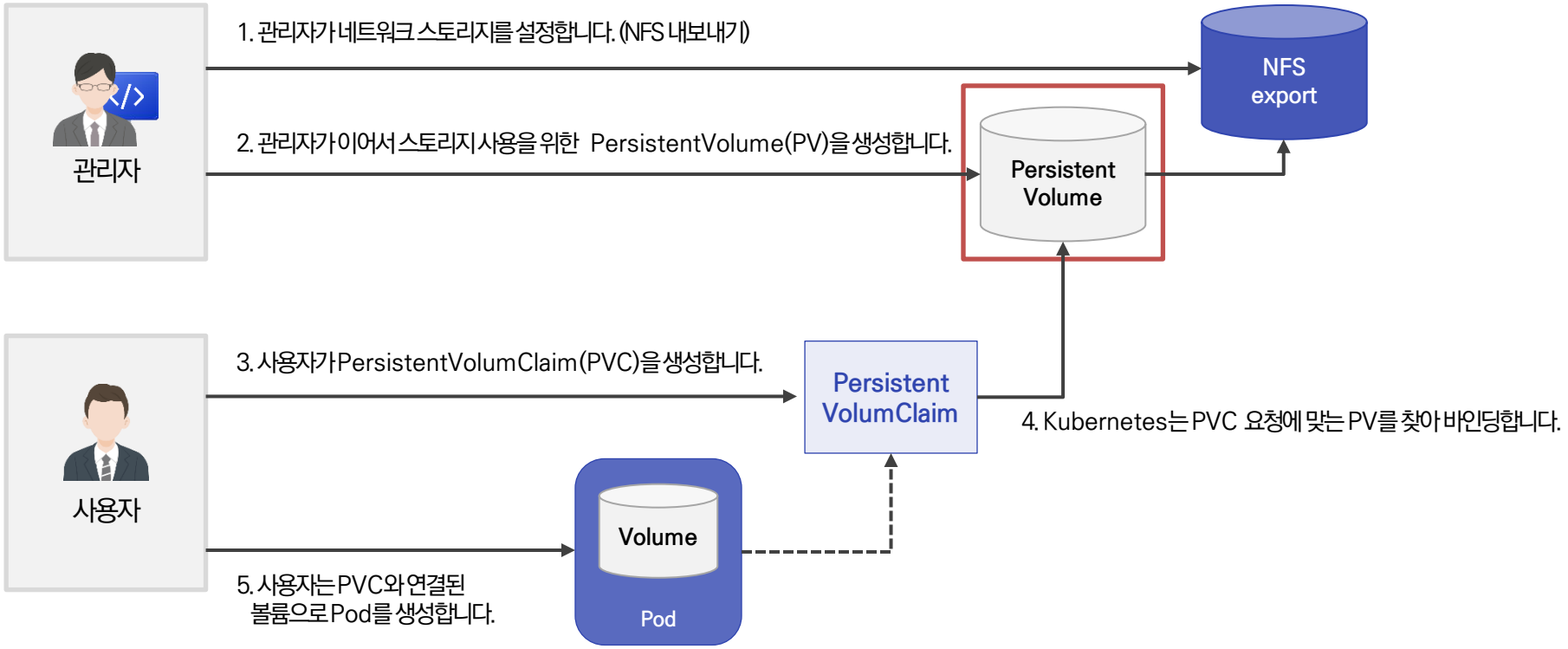
PersistentVolume 은 Pod 내 데이터 영속성을 유지하기 위해 사용



Kubernetes Native 어플리케이션 환경 소개

Persistent Volume

PersistentVolume 은 Pod 내 데이터 영속성을 유지하기 위해 사용

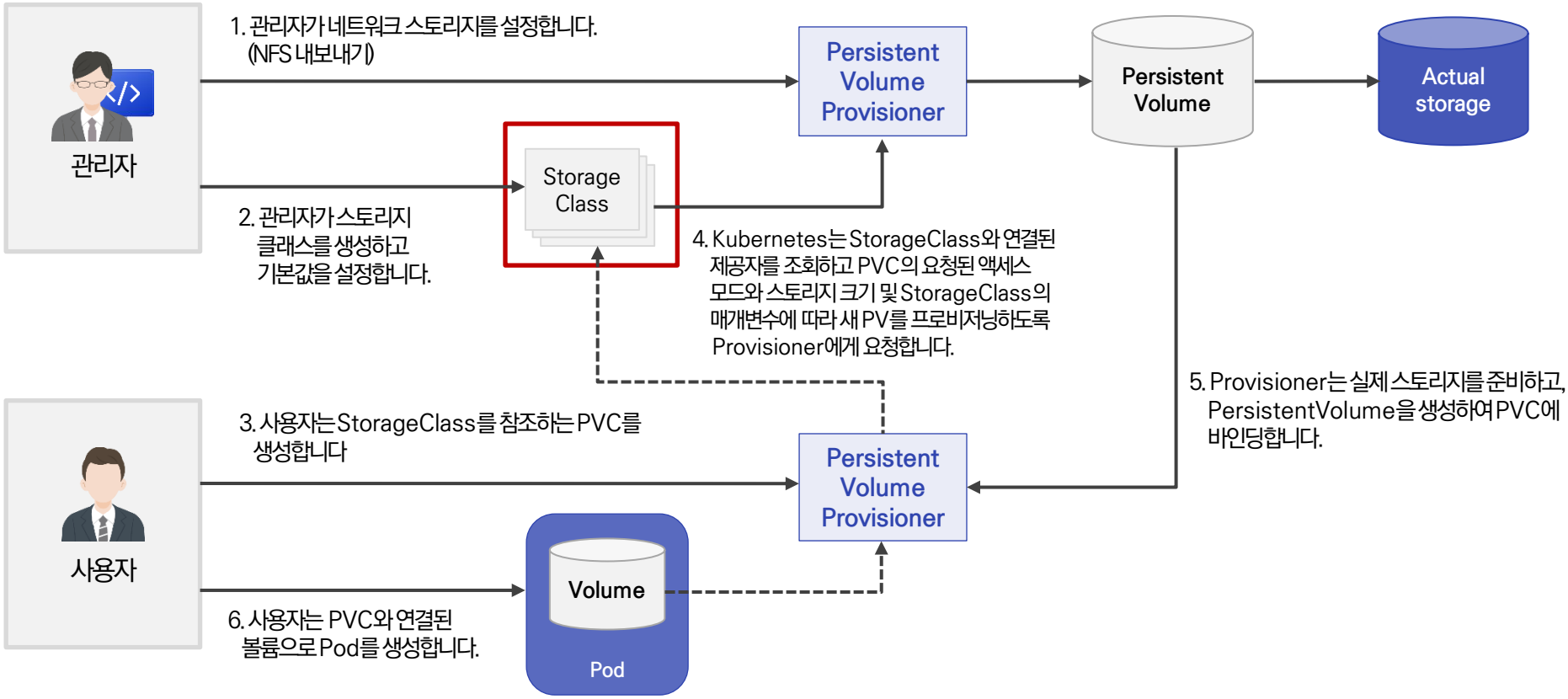


정적 볼륨 생성 예시

Kubernetes Native 어플리케이션 환경 소개

Persistent Volume

관리자는 StorageClass를 미리 생성하고, 사용자는 PVC 이용 Pod에 볼륨 할당 사용

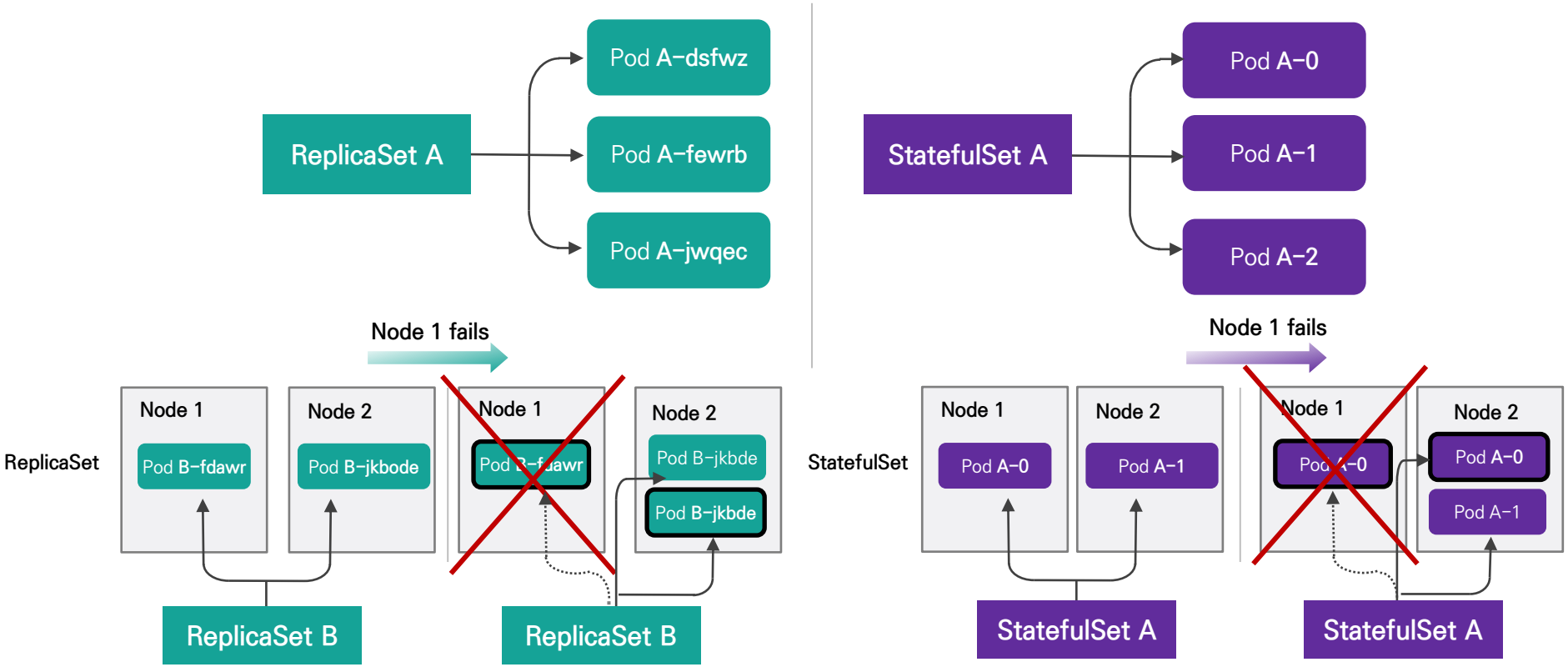


동적 볼륨 생성 예시

Kubernetes Native 어플리케이션 환경 소개

ReplicaSet

ReplicaSet은 웹과 같은 비상태 서비스, StatefulSet은 DB와 같은 상태관리 서비스

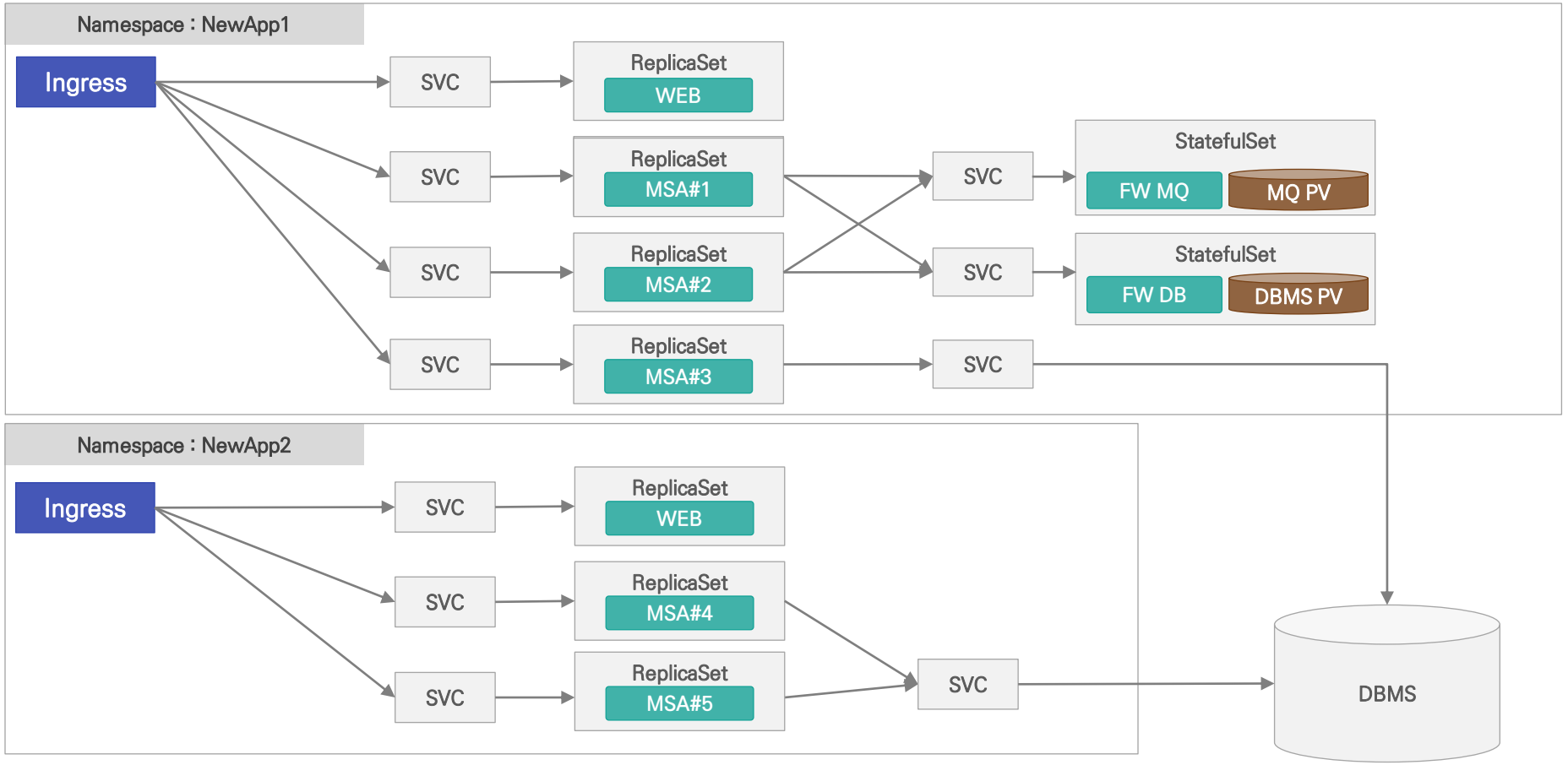


ReplicaSet은 Cattle, StatefulSet 은 Pet 과 같은 존재

Kubernetes Native 어플리케이션 환경 소개

ReplicaSet

Kubernetes 어플리케이션 배포 예시



* ReplicaSet은 Deployment로 배포

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

클라우드 네이티브 발주자 가이드

공공 클라우드 네이티브는 어떤 효과를 제공하게 되나요?

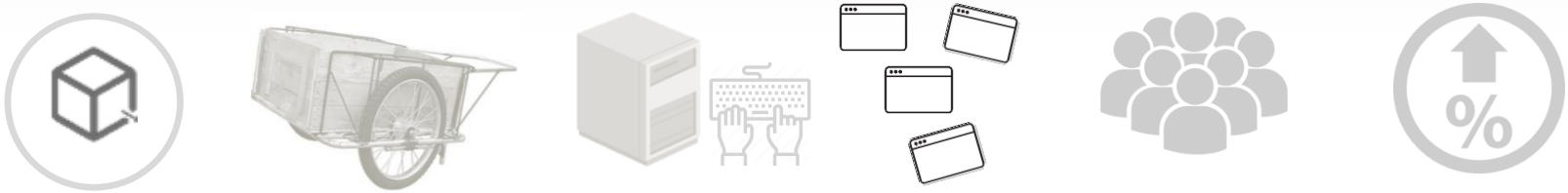
NIA 한국지능정보사회진흥원

클라우드 네이티브 기대효과

행정기관을 위한 정보화 사업 단계별 관리 및 점검가이드를 참조하여 수립한 클라우드 네이티브 개발자 프로세스입니다.

With 클라우드 네이티브 애플리케이션

민첩성	이식성	확장성	표준성	협력성	경제성
빠른 배포	컨테이너 보급	빠른 개발 및 증설 (정책대응)	플랫폼화 (소스, OS, DB등)	서로 소통하는 조직 (Agile)	서비스별 신기술 적용 (C/JAVA, DB등)
배포/테스트지연	프로그램 재설치	시스템 폭주	복잡한 환경	소통 어려움	높은 비용



Without 클라우드 네이티브 애플리케이션

클라우드 네이티브 애플리케이션 행정·공공업무 변화 내역

MSA 신속한 행정	민첩성	행정·공공기관의 **빈번한 제도 및 정책변화에 빠르게 대응할 수 있는 On-Demand 서비스**를 제공
서비스 확장성	확장성	백신예약, 마스크 예약, 연말정산, 원격근무 등 업무 폭주 시에도, **신속한 자원 증설이 가능하도록 컨테이너 기반** Scale-Out 자원환경을 제공
MSA 프로그램 이식성	이식성	공공·민간 클라우드 센터에 설치된 프로그램 파일을 **기본적으로 동일한 아키텍처의 다른 컴퓨터로 전송 및 API를 호출**
MSA 신서비스 보급	경제성	정부의 디지털 뉴딜(디지털 트랜스포메이션, 4차산업 혁명)에 변화에 능동적으로 대처, **다양한 언어로 서비스 개발할수 있도록 개발언어에 대한 선택성을 제공**
MSA 서비스 독립성	민첩성	행정·공공기관 업무담당자의 잦은 보직 변경이 있는 상황에서, **타시스템 전체 구조를 이해할 수 없는 상태에서도 전체시스템의 영향없이 작은 서비스로 개발**
CI/CD 서비스 자동화	협력성	DevOps기반 자동화를 통한 휴먼장애 방지, 각 서비스별 독립적인 개발·배포를 통하여 **행정·공공기관 운영자에게 편리한 배포 및 모니터링 환경을 제공**
MSA 플랫폼 보급확산	표준성	디지털 정부 플랫폼 기반의 손쉬운 개발 툴 등 표준화된 개발플랫폼 환경제공을 통하여 **공공 클라우드 센터의 전면전환을 효과적으로 수행**

NIA 한국지능정보사회진흥원

행정·공공 업무가 이렇게 바뀝니다.

클라우드 네이티브 기반
'포스트 코로나 시대의 디지털 정부혁신 발전계획'과
'한국판 뉴딜 종합계획'에 따른

빠른 대국민 서비스가 가능합니다

클라우드 네이티브 기반
공공 클라우드 센터 전면 전환에 따른
센터, 부서, 업무 간의

상호 운용성 제공이 가능합니다.

클라우드 네이티브 기반
마이크로 서비스 품질 강화(즉시 배포 등)에 따른

24 x 365 무중단 행정 서비스가 가능합니다.

클라우드 네이티브 기반
개발부터 운영까지 전주기 관리를 통한 (DevOps, CI/CD)

서비스 운영 자동화가 가능합니다.

감사합니다.