



Business Agility와 Digital Innovation을 위한 .

Application Modernization

ARCHITECTURE LEAD NEXT LEVEL

들어가기 전에...

동아일보

2020-08-03 (월) B04면

시스템 잘게 쪼개 민첩한 대응… 글로벌 IT 화두는 ‘암스트롱’

(AM+Strong)

기존 IT 시스템 복잡하게 얹혀
간단한 업데이트에도 전체 멈춰
“부분마취해도 될 일을 전신마취”

넷플릭스, 서버 1000개 이상 쪼개
데이터 폭증에도 안정적 서비스
LG CNS, 기상청 등 프로젝트 진행
국내외 제조업계에서도 관심 높아져

이 서비스를 제공하기 위해 시스템 개편에 나서고 있다.

‘암스트롱’의 핵심은 기존 IT 시스템을 수천 수백 개로 쪼개는 것이다. 복잡한 시스템을 여러 개의 자율적인 조직으로 분화해 각종 업데이트 요구에 실시간 대응할 수 있는 민첩성을 갖추기 위해서다. 현재는 간단한 업데이트를 위해 서버 전체를 멈춰야 하는 경우가 적지 않다. IT 업계 관계자는 “부분 마취로 가능한 시술을 위해 전신 마취를 하는 것”이라며 “클라우드 전환 이 효율적으로 작동하기 위해서라도 AM이 필수적”이라고 했다.

넷플릭스는 AM을 통해 글로벌 시장에 도약한 대표적 사례다. 넷플릭스는 2007년 심각한 데이터베이스(DB) 손상을 입고 3일간 서비스 장애를 겪은 후 10년 넘게 AM을 선도하고 있다. 현재 서버를 1000개 이상으로 쪼개는 작업을 통해 현재는 11.5초마다 새로운 요구사항을 반영하는 시스템을 구축했다. 2007년부터 최근까지 월간 시청량은 1000배, 이용자는 10배 이상 늘었지만, 안

암스트롱(AM-strong):

애플리케이션 현대화 (Application Modernization)를 의미하는 'AM'과 강하다는 뜻의 'Strong'의 합성어.
인류 최초로 딜을 맺자마자 미국의 LG 암스트롱처럼 기존 정보기술(IT) 시스템의 근간을 허물고 완전히 새로운 체계를 만들어 나가자는 IT 업계의 의지를 담은 신조어.

정보기술(IT) 시스템 쪼개기 나서는 기업들

넷플릭스	서버 1000개 이상으로 쪼개 고객의 요구 11.5초 만에 반영하는 시스템 구축
LG유플러스	기존 서버를 가입 상담, 예약 판매, 개통 등으로 분리, 업그레이드 시간 50% 이하로 감소
쿠팡	서비스 대부분을 작은 단위로 쪼개, 개발 테스트 배포까지 민첩하게 가능한 환경 구축
기상청	전국 주요 지역 기상 데이터센터를 클라우드로 분산 추진, 실시간 기상 데이터 인정적 제공

젝트를 통해 주문과 결제 서비스의 속도를 높이고 장애를 줄였다.

최근에는 제조업에서도 ‘암스트롱’의 중요성이 커지고 있다. 자동화 공장들도 라인에 생산 주문을 넣은 후 돌발 상황에 대처하는 데 상당한 시간이 걸리는데 이를 해결하지 못하면 경쟁력에 심각한 문제가 생길 수 있기 때문이다. 이미 중국 기업들은 경쟁적으로 AM에 집중 투자하면서 1, 2년 후면 사흘 안에 모든 시스템 수정이 가능한 체계를 구축할 것으로 예상되고 있다.

‘AM’은 클라우드 전환 시장이 커질수록 더 주목받는 분야가 될 것이란 전망이 나온다. 현신균 LG CNS 부사장(AM사업 총괄)은 “클라우드로 전환한 기업의 약 80%가 비대면 시대에 적응하기 힘들 정도로 비즈니스 민첩성이 부족하다는 분석들이 나온다”며 “기업이 기존 자산을 잘 활용하면서 새로운 힘을 만들어내는 ‘양손잡이 조직’으로 발전하기 위해서도 AM이 필수적”이라며 말했다.

유근형 기자 noel@donga.com

LG CNS는 LG유플러스의 온라인 가입 지원 시스템에 대한 AM 프로젝트를 최근 마무리했다. 기존에는 가입 상담, 예약 판매, 개통 등 8개 영역 중 1개 서비스를 업그레이드하려고 해도 14일 가량이 걸렸지만, 현재는 6일 이하로 시간은 절반 이상 단축했다

제는 6일 이하로 시간을 절반 이하 단축해낸다. LG CNS는 기상청의 차세대 종합기상정보 시스템의 ‘AM 프로젝트’도 진행하고 있다. 기존 기상 데이터 처리 방식을 작은 단위로 쪼개고, 전국 주요 지역 기상 데이터센터를 클라우드 시스템에 분산 구축해 기상 서비스 제공을 효율화하기 위해서다. 쿠팡, 11번가도 AM 프로

LG CNS는 기상청의 차세대 종합기상정보 시스템의 ‘AM 프로젝트’도 진행하고 있다. 기존 기상 데이터 처리 방식은 작은 단위로 쪼개고, 전국 주요 지역 기상 데이터센터를 클라우드 시스템에 분산 구축해 기상 서비스 제공을 효율화하기 위해서다

들어가기 전에...

MSA가 주목받는 이유는 빠른 서비스 출시와 상황에 따른 확장성 때문이다. 서비스별로 시스템을 구축하기 때문에 빅뱅방식보다 개발 속도가 빠르다. 특히 클라우드 환경으로 시스템 이전이 늘면서 MSA방식 효용성이 높아졌다.

임용서 한국IBM MSA부문 총괄 상무는 "MSA 도입은 확대될 것이라면서 "혁신적 변화도 경쟁력을 확대하려는 기업이 적극 도입해 MSA는 필수 요소가 됐다고 말했다.

Contents

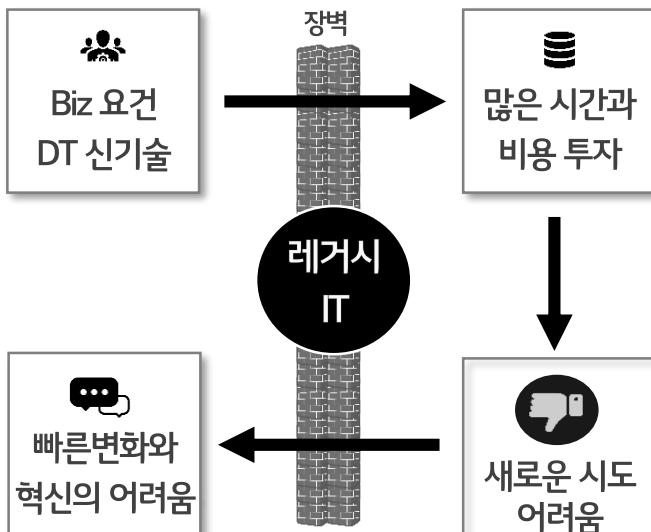
1. Application Modernization 개요
2. MSA 성공을 위한 핵심 요소
3. MSA Outer Architecture
4. LGCNS MSA-Core System 적용 및 AM체계

비즈니스 Agility를 위한 새로운 IT의 필요성

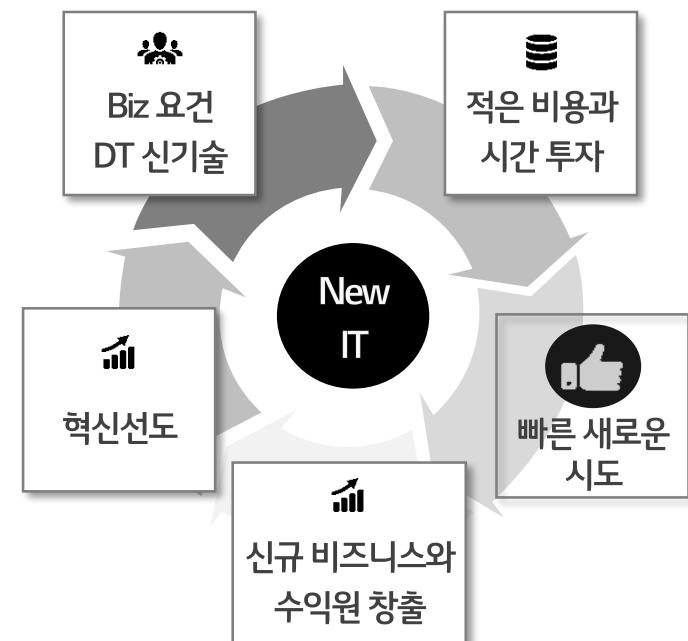
1. Application Modernization 개요

고객의 Needs를 신속하게 반영하고 혁신적인 서비스를 제공하기 위해 다양한 기술과 방법을 적용하고 있습니다. 이 때, IT는 비즈니스 신속성과 혁신을 뒷받침할 수 있는 민첩한 구조여야 합니다

비즈니스 변화에 부담이 큰 레거시 IT 환경



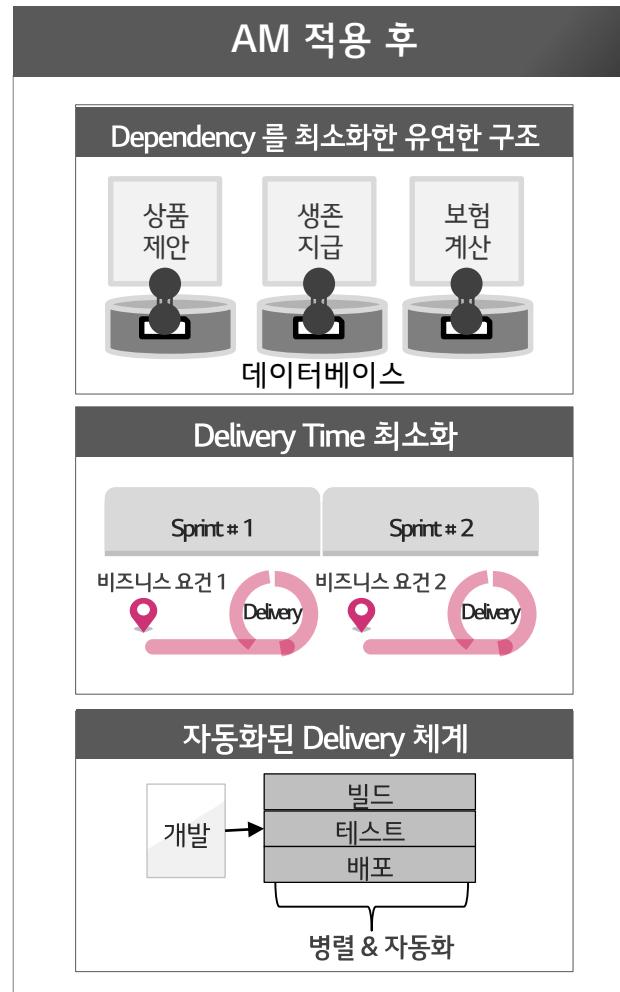
혁신을 선도하는 새로운 IT 환경



Application Modernization의 정의

1. Application Modernization 개요

비즈니스 Agility를 위해서는 Biz 요건의 신속한 반영, AI/Bigdata 등 신기술의 빠른 융합이 필요하며, 이를 위해 기존 IT의 응용구조, 개발방식, 아키텍처 변화를 추진하며 이를 “Application Modernization”으로 정의합니다



어플리케이션 구조
MSA

일하는 방식
Agile

운영환경
DevOps

Contents

1. Application Modernization 개요
2. MSA 성공을 위한 핵심 요소
3. MSA Outer Architecture
4. LGCNS MSA-Core System 적용 및 AM체계

MSA 동향

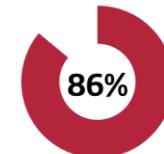
MSA 도입 의견



신규 시스템 도입을 고민하는 약 91%의 사용자들이
MicroServices 도입을 검토



92% CxO들이 앞으로 Microservice의
도입이 증가할 것으로 예측

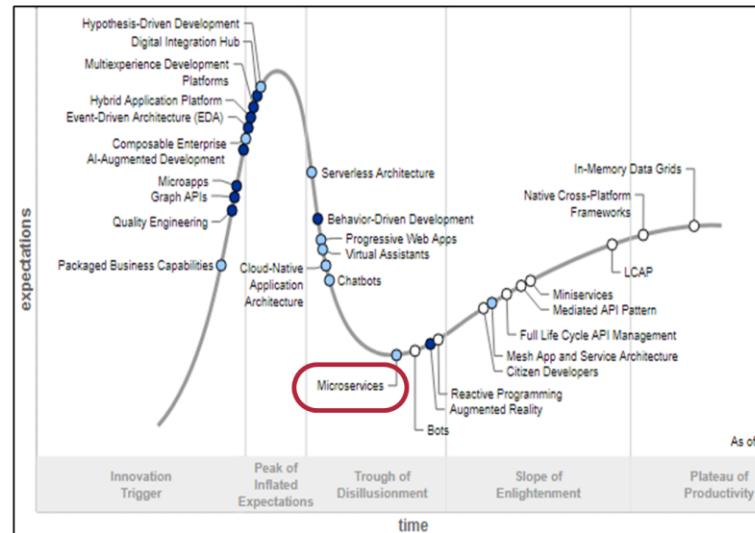


86% 사용자들이 향후 5년내 MicroServices가
기본 아키텍처가 될 것으로 예측

Source: CLOUD EXPO 2018 , Deloitte

- Cloud로의 인프라 전환 이후, 근본적인 변화를 위해 Cloud Native Application, MicroServices 기반 Application Modernization 단계로 접어들고 있음

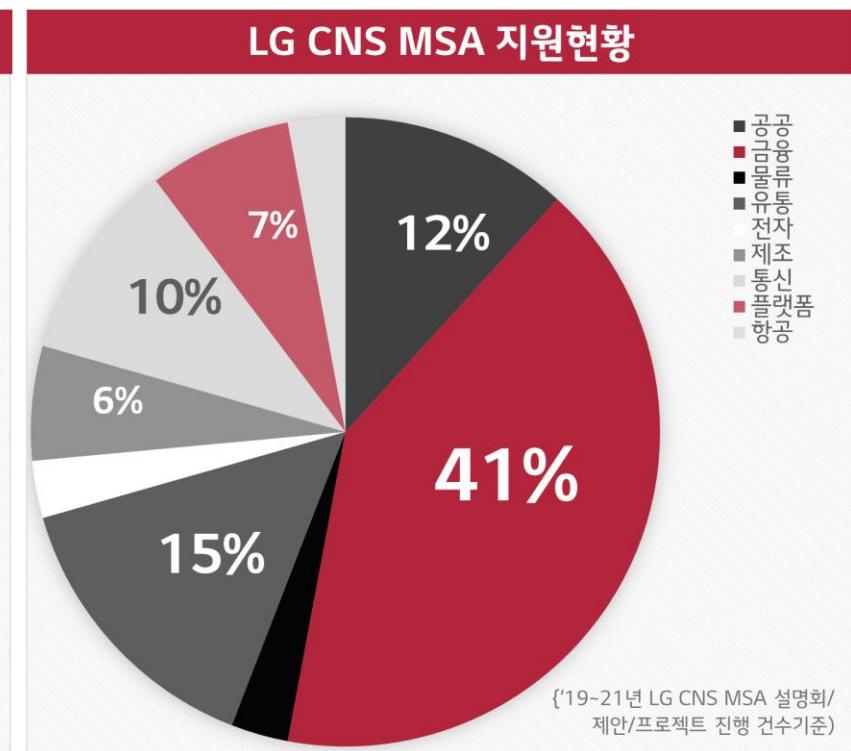
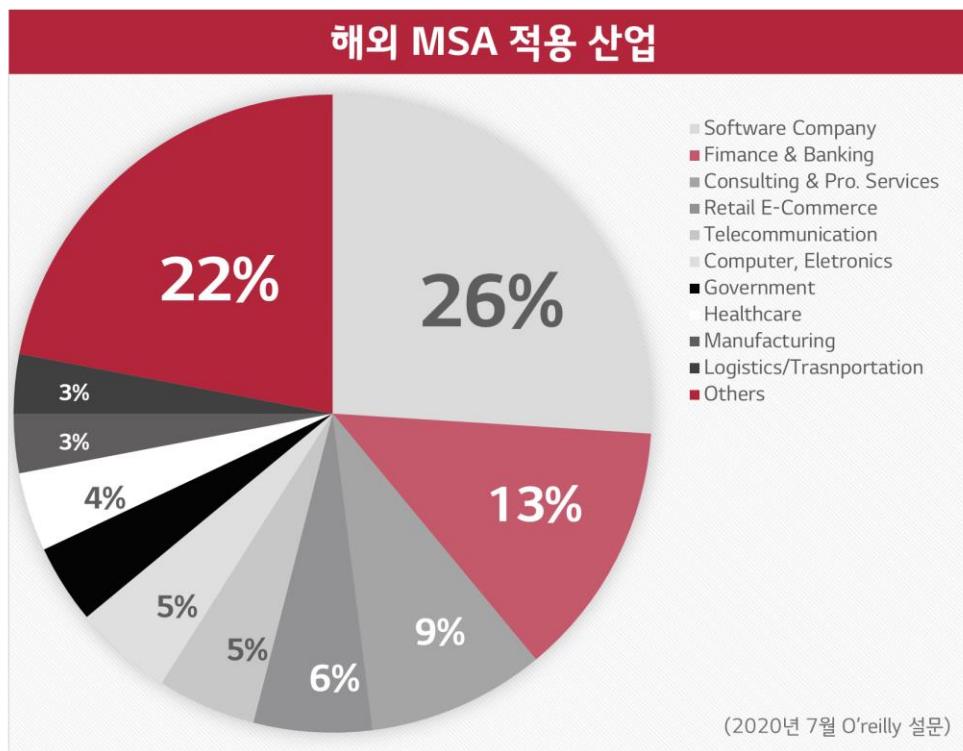
Gartner : Hype Curve



Source: Hype curve 2020, Gartner

- MicroServices 성장기에 접어들었으며(향후 2~5년 내 확산 예상) 애자일, 조직, 문화 등을 고려하지 않은 무분별한 도입과 그로 인한 도입 실패 그리고 Monolithic으로의 회기함을 견제

MSA 적용 사례



- ✓ 해외에서는 MSA를 근래에 도입하였고, 다양한 사업군에 걸쳐 MSA 적용하고 있으며(금융권 분야 13% 수준)
91%가 적용을 성공했다고 긍정적으로 답변함

- ✓ 쇼핑몰 등 대 고객 서비스 중심으로 MSA 도입 및 구축 본격화
'20년 기점으로 금융, 유통, 공공, 통신영역의 관심이 급격히 증가
은행은 비금융 채널 중심으로 Small, Agile 기반 MSA를 적용

MSA를 도입하는 현실



일단, MSA 도입이 목적

생각해 볼 문제

- 앞뒤가 맞지 않는(수긍하기 어려운) Architecture를 도입할 가능성
- 기존 팀이나 서비스 관련 팀들이 그 목적을 이해하지 못할 수 있음



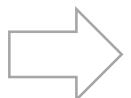
MSA 도입으로 여러 문제점
해결 기대

- 개발 시 이러한 문제/숙제는 여전히 남게 되거나 오히려 더 악화될 수 있음
- 왜 필요한지 생각을 바꾸지 않고 진행하면 오히려 실망할 수 도 있음



프로세스, 정책, 조직 변경
없이 MSA 도입

- 도입 시 최소한의 혜택만 얻게 됨
- 서비스 전환노력이 실패로 돌아갈 가능성이 높음



서비스를 어떤 수준으로
쪼개야 하는지 기준 불명확

- 많은 서비스로 분리 시 과도한 복잡성 초래, 서비스 추적, 모니터링 어려움
- 분산된 서비스 트랜잭션 처리, 데이터를 한번에 조회하기



비즈니스 민첩성과
MSA에 대한 잘못된 정의

- 적합하지 않은 업무에 MSA를 도입해 비즈니스 민첩성을 악화 할 수 있음
- 현업과 개발 조직간의 MSA 구현 목적을 위한 갭 발생

-  **MSA를 해야만 하는 이유와 구조적 문제점**
-  **Micro Service 쪼개기**
-  **반드시 고려해야 하는 Critical Design Issues 및 테스트**
-  **Outer Architecture - MSA의 기반 아키텍처**
-  **Core System에 MSA 적용하기**

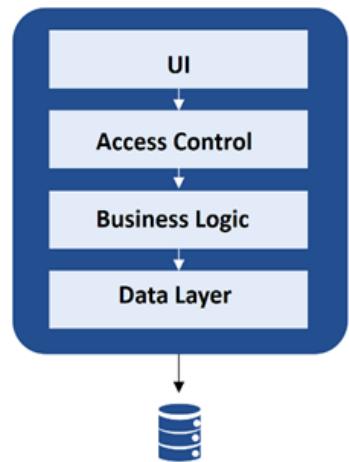
MSA를 해야만 하는 이유와 구조적 문제점 – MSA 정의

2. MSA 성공을 위한 핵심 요소

비즈니스 변화와 성장 속도가 빨라짐에 따라 IT시스템이 비즈니스 변화와 속도에 신속하게 대응하기 위하여 만들어진 개발과 운영을 포괄하는 기술입니다

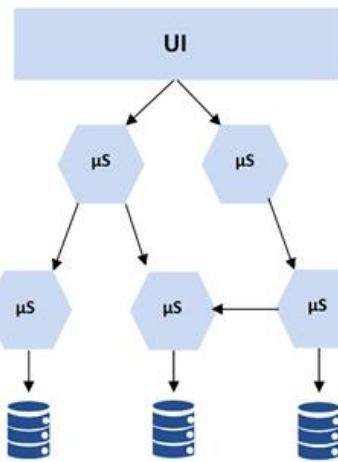
모놀리식(Monolithic) 아키텍처

- 단일체의, 한 덩어리로 뭉친
거대 단일 서비스 개발방식.
- 하나의 프로젝트도 구성되어
있으며 단일 패키지도 배포된다.



Monolithic

Mainframe, Client/Server, Web



Microservices

Cloud Native Application

마이크로서비스(Micro Service) 아키텍처

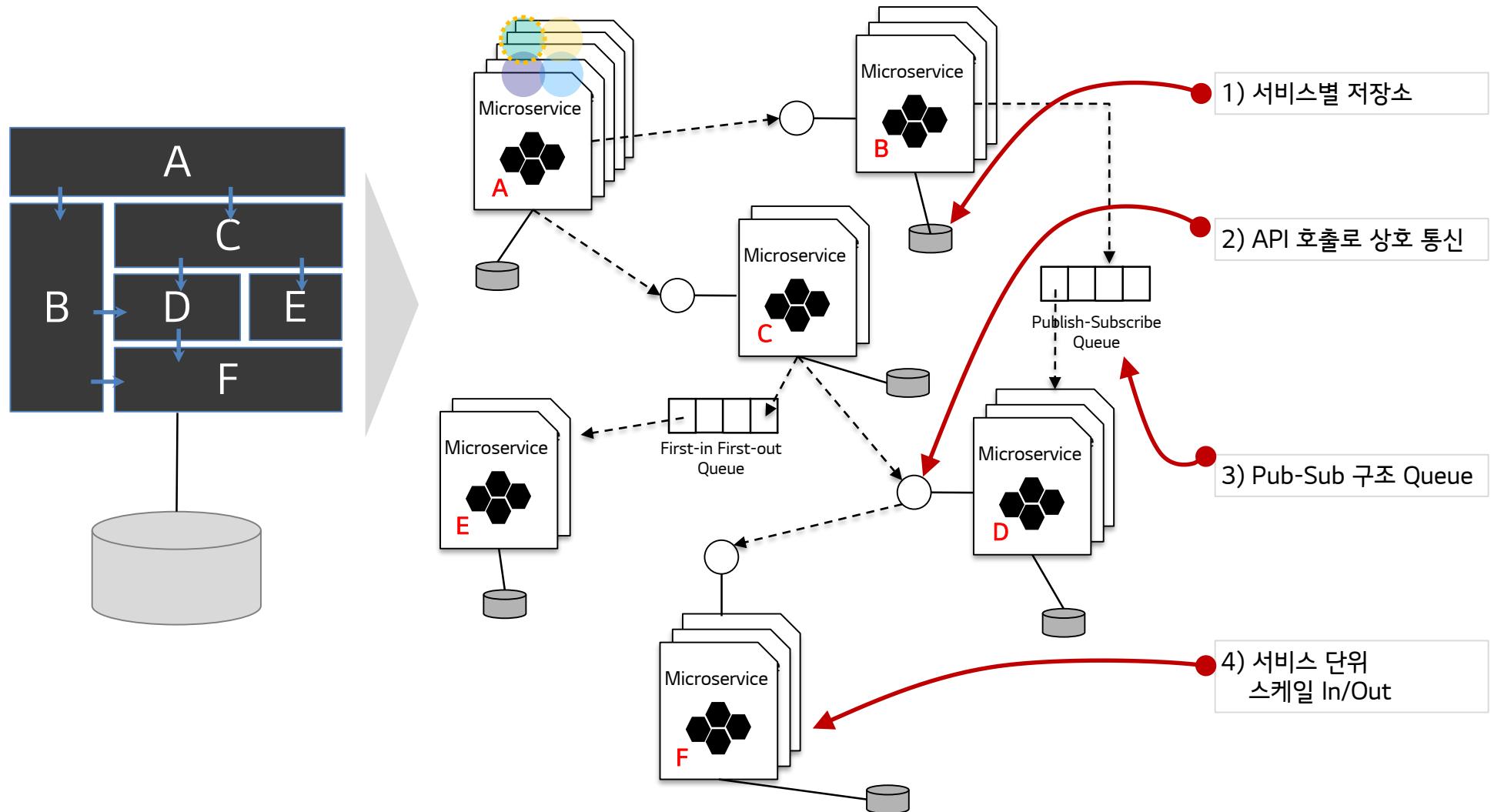
- 독립적이고 단순한 개별 서비스들도
전체의 서비스를 구성
- 단일 Application은 나누어서
작은 서비스들의 조합으로 구성

MSA is New Architecture Style

하나의 큰 애플리케이션은 여러개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처.
즉, 어플리케이션이 여러 개의 작고 독립적으로 배포가 가능한 서비스로 구성되어 있는 것을 의미합니다.

MSA를 해야만 하는 이유와 구조적 문제점 – MSA 구조

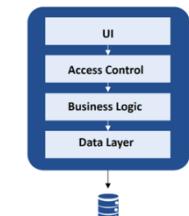
2. MSA 성공을 위한 핵심 요소



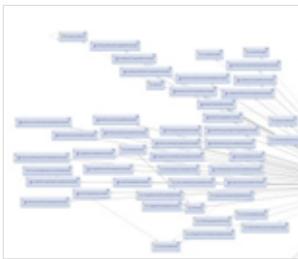
MSA를 해야만 하는 이유와 구조적 문제점 – MSA 구조

2. MSA 성공을 위한 핵심 요소

Simple Code Base



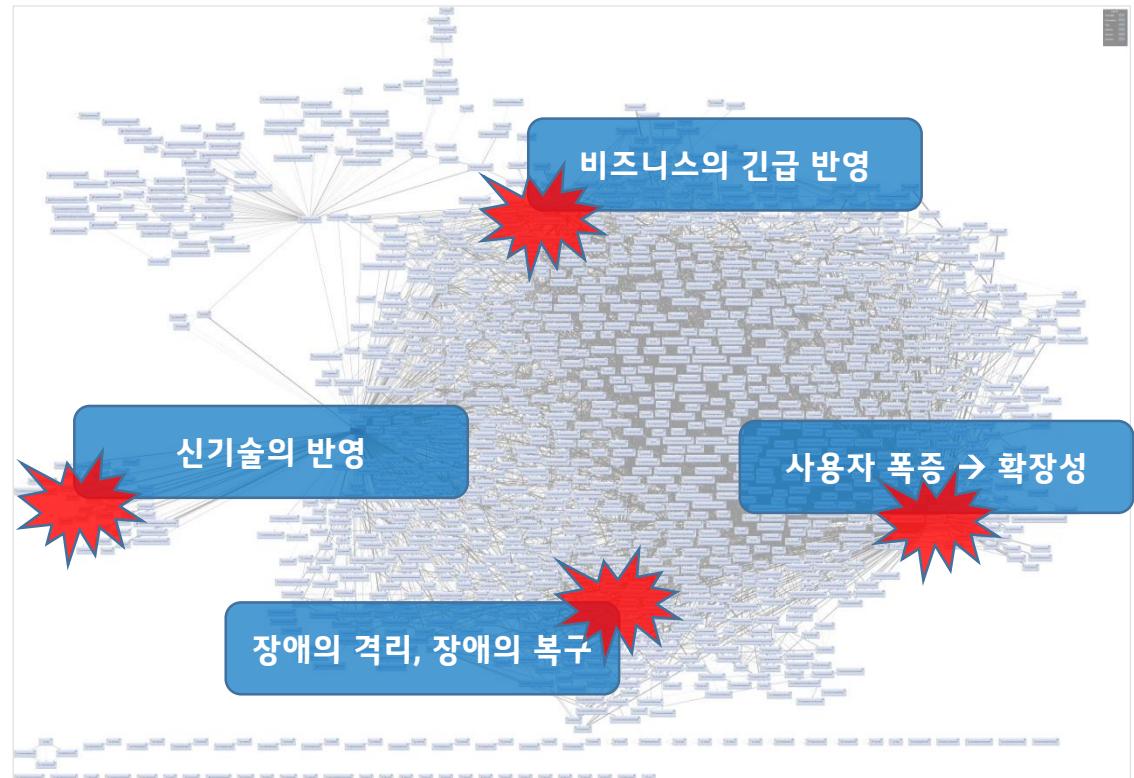
Monolithic



복잡도 증가

- 간단한 Architecture
- 단순한 기술
- 유지보수(추가/변경) 용이

Large Code Base



비즈니스의 긴급 반영

신기술의 반영

사용자 폭증 → 확장성

장애의 격리, 장애의 복구

대규모 개발자에 의한 개발/배포/유지보수의 어려움



Monolith 또는 Monolithic Application

- 사전적 의미 : 거대한, 단일체
- 시스템이 지나치게 크다는 표현으로 MSA대비 기존 방식을 의미

MSA를 해야만 하는 이유와 구조적 문제점 – MSA 구조적 문제점

2. MSA 성공을 위한 핵심 요소

관점		Monolith 대비 단점
성능	서비스간 호출 시 API 통신	<ul style="list-style-type: none">통신 포맷인 Json/Xml 메시지를 Java Object(VO 등) 데이터 모델로 변환하는 <u>네트워크 오버헤드</u> 발생네트워크를 통한 API 통신으로 <u>처리 시간이 추가 소요</u> Latency Time(대기 시간, 호출 시간)
테스트	서비스들 각각 분리, 타 서비스에 대한 종속성	<ul style="list-style-type: none">특정 시나리오 기능 테스트 시 여러 서비스에 걸쳐 테스트를 진행테스트 환경 구축, 문제 발생 시 여러 시스템을 동시에 점검해야 하는 등 <u>테스팅의 복잡도 증가</u>
트랜잭션	분산 환경으로 DBMS 내 트랜잭션 처리(Commit, Rollback) 이용 불가	<ul style="list-style-type: none">API기반의 여러 서비스를 하나의 트랜잭션으로 묶기 어려움, <u>데이터 정합성 및 보상 트랜잭션 이슈 발생</u>
데이터 조합 (Query)	서비스 별 데이터 분리 저장	<ul style="list-style-type: none">여러 서비스에 걸쳐 분산 된 데이터를 한번에 조회하기 어려움, <u>성능 저연 및 코딩 난이도 증가</u>
복잡도	분산 환경, 트랜잭션, Query 등을 위한 별도의 설계, 개발, 테스트 필요	<ul style="list-style-type: none">서비스 호출 추적, 디버깅(장애 추적), 모니터링 등의 어려움전체적인 <u>운영환경의 복잡도 증가</u>

-  **MSA를 해야만 하는 이유와 구조적 문제점**
-  **Micro Service 쪼개기**
-  **반드시 고려해야 하는 Critical Design Issues 및 테스트**
-  **Outer Architecture - MSA의 기반 아키텍처**
-  **Core System에 MSA 적용하기**

목표 설정



MSA 목표 설정



조직 구성



도입 대상 식별

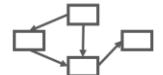
서비스 분리



이벤트스토밍 FDD DDD



후보서비스 식별



서비스 관계 정의



서비스 정의

서비스 평가/검증



후보서비스 평가



응용프로그램 역공학



서비스 시뮬레이션

서비스 설계



API 설계



동기/비동기 설계



분산 이슈 설계

LG CNS Enterprise MSA Hexagon 방법론

2. MSA 성공을 위한 핵심 요소

Microservice Discovery

후보 서비스 도출 By 비즈니스 역량

Biz 도메인 분석/후보서비스 식별

- Event Storming
- Service Relationship Diagram
- Service Definition
- 현행시스템 분석/업무기능분해/업무흐름 분석 (FDD, DDD)

후보 서비스 평가 By 평가항목

비즈니스 요건에 따른 평가/측정 항목 정의 (예시)

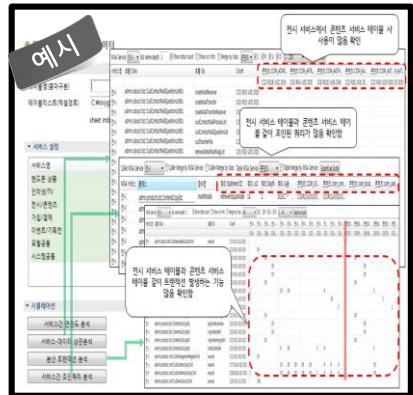
- 신속하고 독립적인 배포
- 사용량(트랜잭션) 폭증
- 장애의 격리



マイクロ서비스 시뮬레이션을 통한 feasibility 분석
(DevOn Microservice Simulator)

Microservice Simulator를 이용하여 AS-IS 소스 기반의 시뮬레이션 수행

- 서비스 의존성 분석
- 서비스 데이터 상관도 분석
- 분산 트랜잭션 분석
- 조인 쿼리 분석



Microservice Design

API First Design

- 안정적이고 견고한 API 설계를 위한 API First Design 수행

- Provider Driven Design, Consumer Driven Design 기법 사용

분산 트랜잭션 설계

API Composition 설계

동기/비동기 설계

- 마이크로서비스간 분산트랜잭션 및 보상트랜잭션 설계
- 분산 트랜잭션 디자인 패턴(Saga)을 활용한 Orchestration/Choreography 방식의 분산 트랜잭션 설계

- 성능을 고려한 여러 서비스간의 호출 메커니즘 설계
- API Composition 디자인 패턴(CQRS)을 활용한 Command와 Query의 책임을 분리하여 설계

- 서비스 독립성을 고려한 비동기 호출 설계
- 3가지 유형(통보, 요청/비동기 응답, 발행/구독) 설계

マイクロ서비스 설계 모델링 및 검증
(DevOn Modeler/Microservice Advisor)

API First Design, 패턴기반 분산 설계를 지원하고 마이크로서비스의 품질 검증

- 서비스 간 의존성
- 분산 트랜잭션 분석
- API 명세 생성

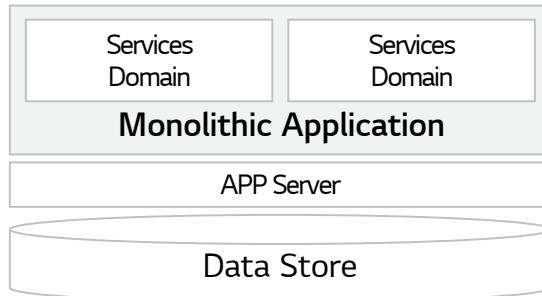


<< MSA 모델 >>

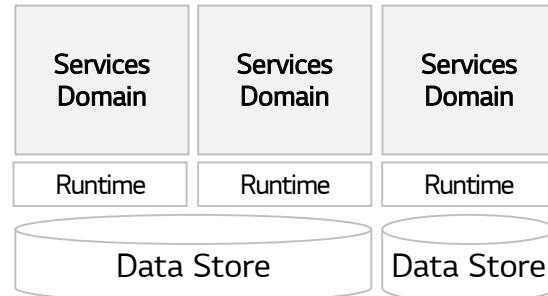
이벤트 스토밍 방식으로 서비스 쪼개기



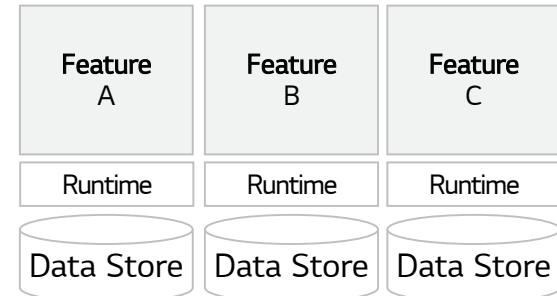
1 Monolithic(Macro)



2 Mini Service



3 Micro Service



Looser Coupling, Greater Agility

Lower Complexity, Easier to Run

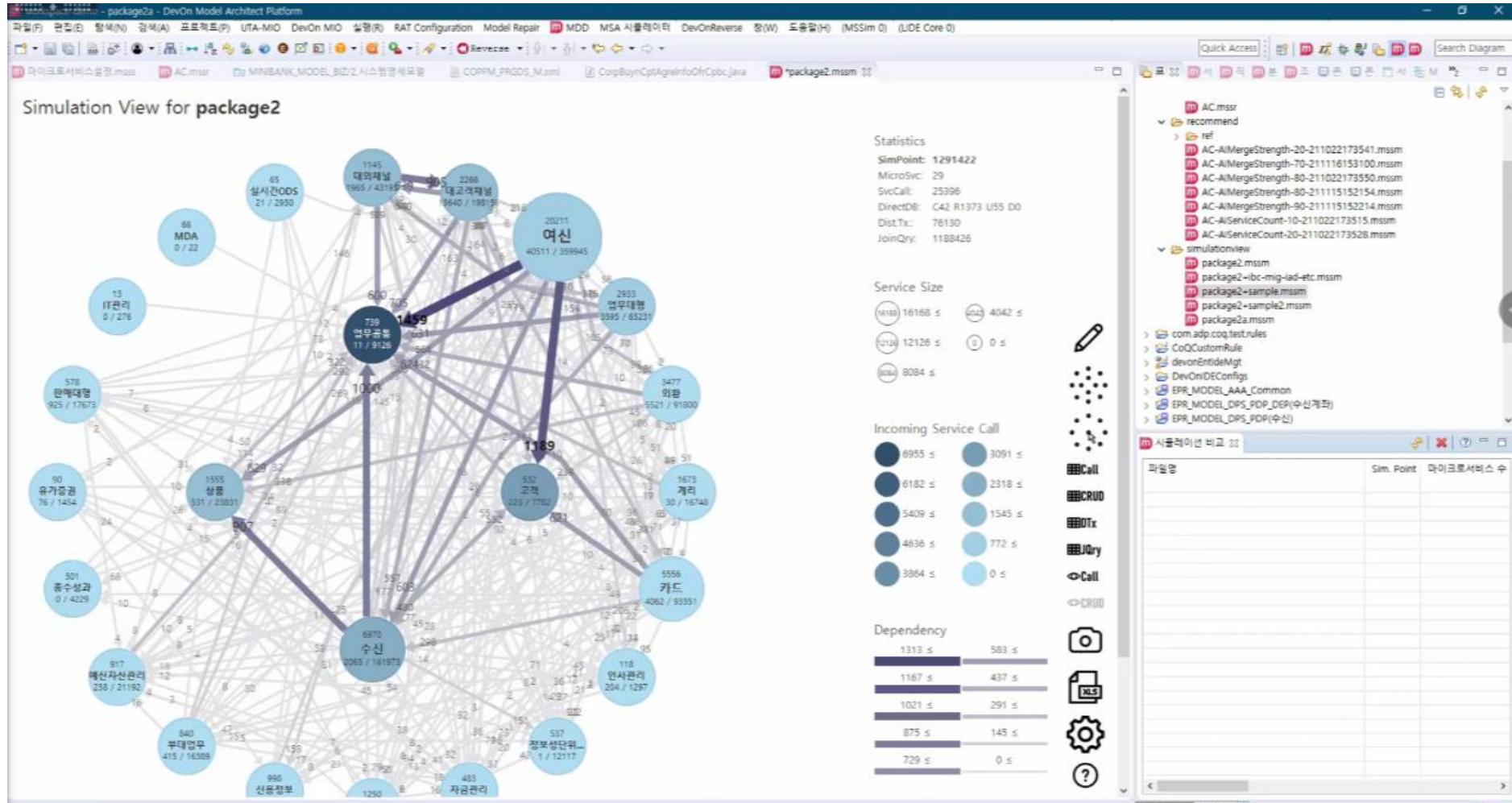
	Monolithic	Mini Service	Micro Service
특징	<ul style="list-style-type: none"> • 거대 단일 어플리케이션 • 서비스에 드화된 호출 • 전동적 기업 응용 아키텍처 	<ul style="list-style-type: none"> • 도메인에 따른 어플리케이션 다중 분리 구조 • 단일 시스템을 도메인 별로 서비스 분리 	<ul style="list-style-type: none"> • 이벤트, 기능에 따른 초소형 어플리케이션 집합 • 인터넷 기업형 응용 아키텍처
장점	<ul style="list-style-type: none"> • 직관적인 아키텍처 구성 • 고가용성 서버 환경 구축 • Trace, E2E, Debug, Monitoring 용이 	<ul style="list-style-type: none"> • 비즈니스 성격에 따른 유연한 대응 • 도메인 분리도 도메인 별 대응 가능한 구조 	<ul style="list-style-type: none"> • 비즈니스 민첩성 확보 • 개발, 배포, 확장 용이성
단점	<ul style="list-style-type: none"> • 대내외 변화 대응 및 비즈니스 민첩성 저하 	<ul style="list-style-type: none"> • 이원화된 플랫폼, 개발/운영 조직 • 마이크로서비스에 비해 상대적으로 떨어지는 비즈니스 민첩성 	<ul style="list-style-type: none"> • 복잡한 배포 및 실행 구조 • 유지보수 비용 증가

Micro Service 쪼개기

2. MSA 성공을 위한 핵심 요소

MicroServices 시뮬레이션 및 자동 추천

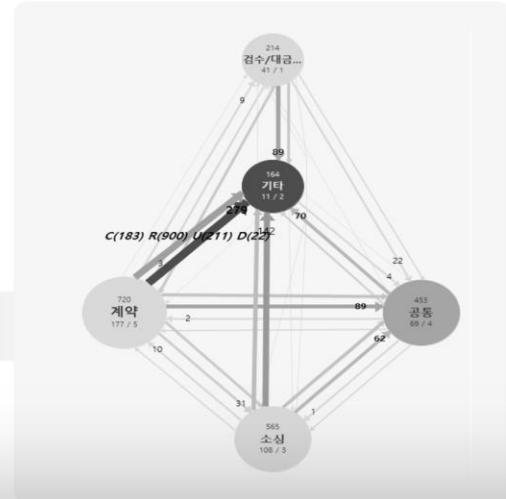
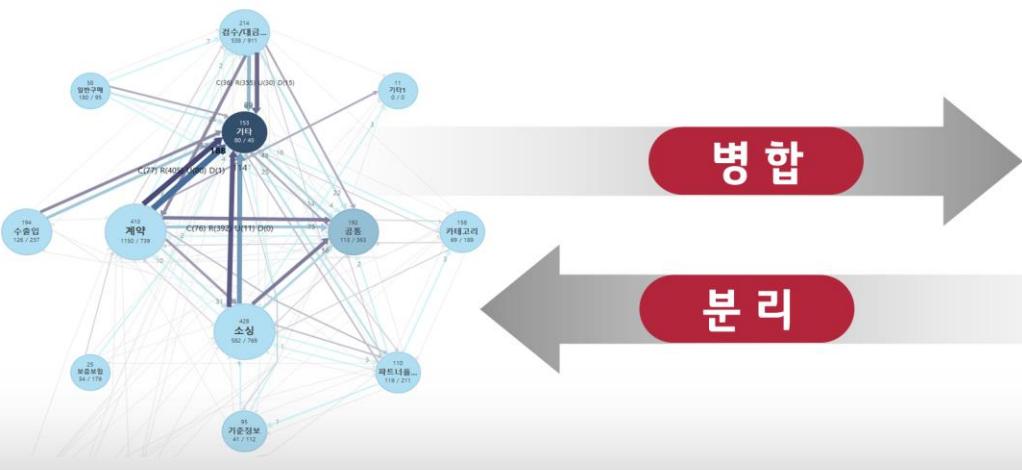
LG CNS DevOn ms Simulator



MicroServices 시뮬레이션 및 자동 추천

LG CNS DevOn ms Simulator

**서비스
분석/예측**



**서비스
병합/분리에
따른
세부 정보**

서비스간 연관도



서비스-데이터 상관관계



분산 트랜잭션



조인쿼리

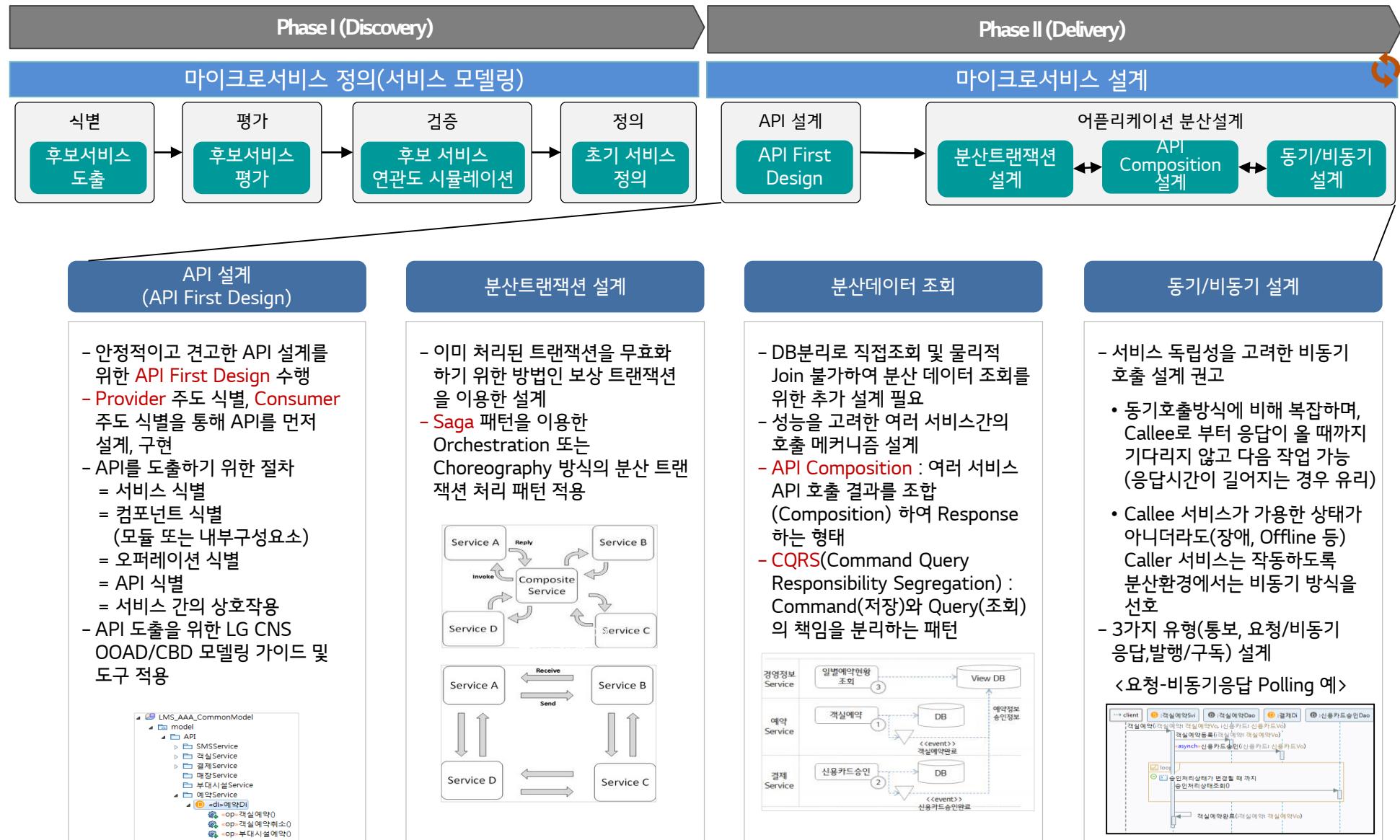


서비스 크기



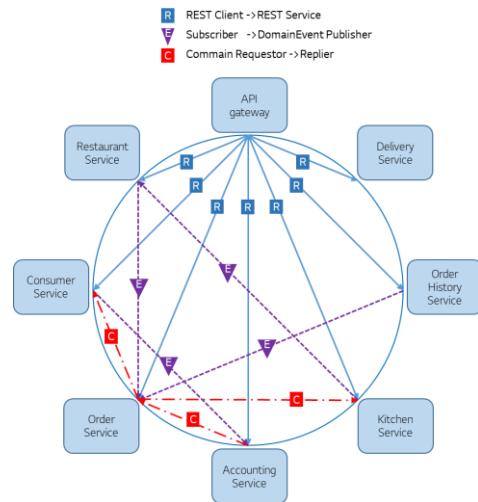
반드시 고려해야 하는 Critical Design Issues 및 테스트

2. MSA 성공을 위한 핵심 요소



반드시 고려해야 하는 Critical Design Issues 및 테스트

2. MSA 성공을 위한 핵심 요소



an example of Inter-service communication in MSA

(출처 : Microservices Pattern, Chris Richardson)

LG CNS 마이크로서비스 테스팅 전략

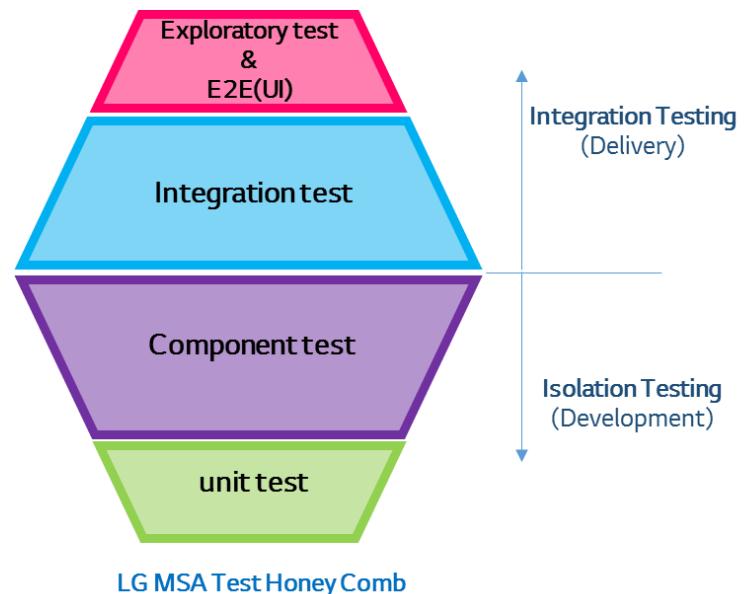
마이크로 서비스들이 상호 협력하고 연계되어야 하는 MSA의 특징을 고려하여 Component/Integration test에 집중하는

HoneyComb 테스트 전략 적용

- Automated Testing(Agility)
- Loosely-Coupled Testing(Isolation)
- Consumer-Driven Contract Testing(Integration)

마이크로서비스 테스팅 Challenges

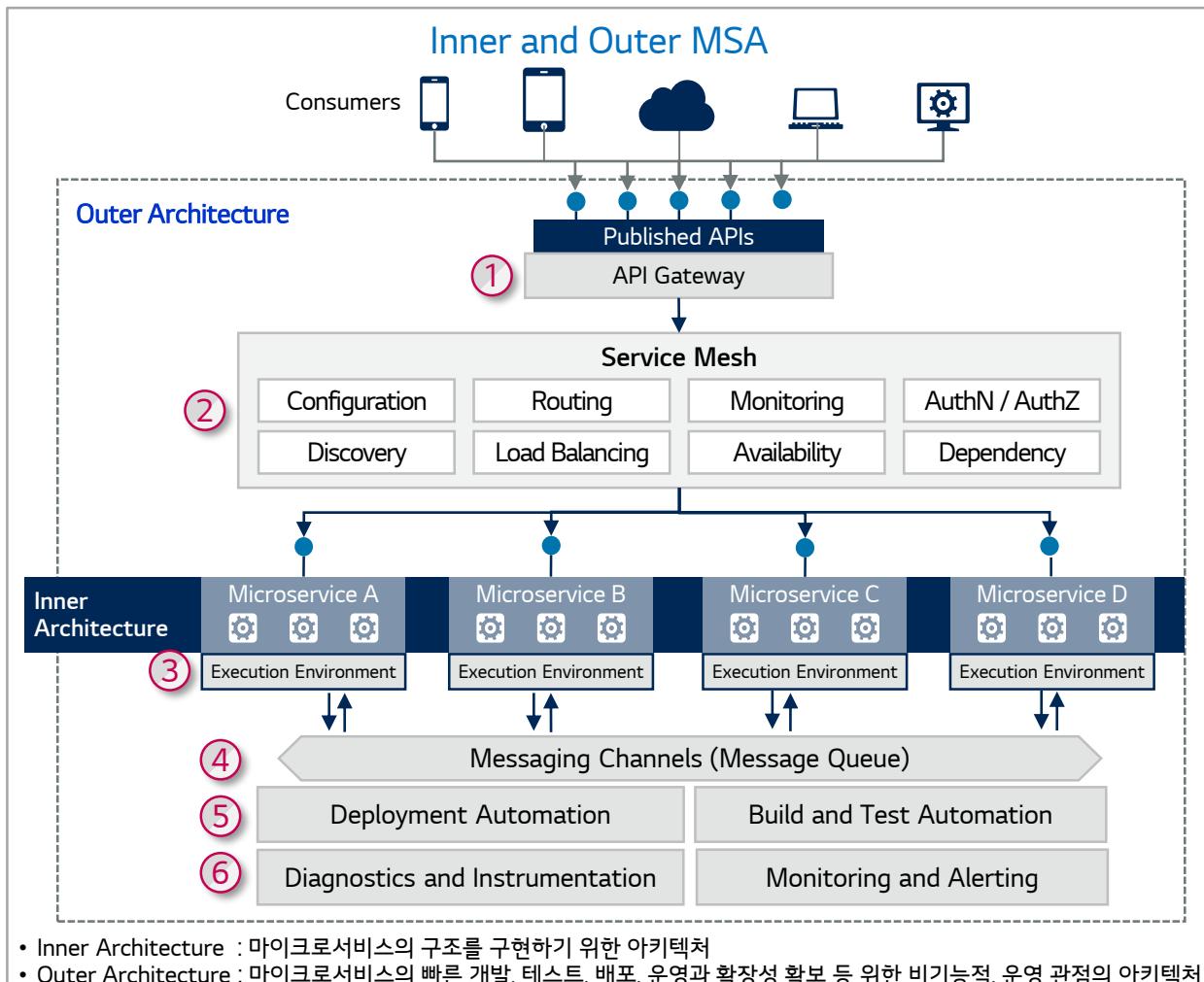
- “Biz Agility” 달성을 위한 빠른 Delivery process가 요구됨
- 모든 마이크로 서비스가 작동하는 테스트 환경 구성이 어려워짐
- API 개선이 더 잦아지고, 작은 서비스들의 상호 조합/구성으로 인한 상호연계 검증 중요성 부각
- 개별 테스트는 단순화 BUT 전체 테스트는 훨씬 복잡해짐



-  **MSA를 해야만 하는 이유와 구조적 문제점**
-  **Micro Service 쪼개기**
-  **반드시 고려해야 하는 Critical Design Issues 및 테스트**
-  **Outer Architecture - MSA의 기반 아키텍처**
-  **Core System에 MSA 적용하기**

Contents

1. Application Modernization 개요
2. MSA 성공을 위한 핵심 요소
3. MSA Outer Architecture
4. LGCNS MSA-Core System 적용 및 AM체계



① API Gateway (External Gateway)

- 외부의 API 요청을 내부 API로 Routing
- 상용 제품의 경우, Service Mesh의 기능을 포함

② Service Mesh

- 내부 서비스간 호출 서비스 제공

③ Container M'gmt

- 서비스의 실행을 위한 Container 기반 환경
(예: Docker, Kubernetes 등)

④ Backing Service

- 서비스간 비동기 통신, 이벤트 전달 등을 위한 Message Queue / MOM 기반 서비스 제공

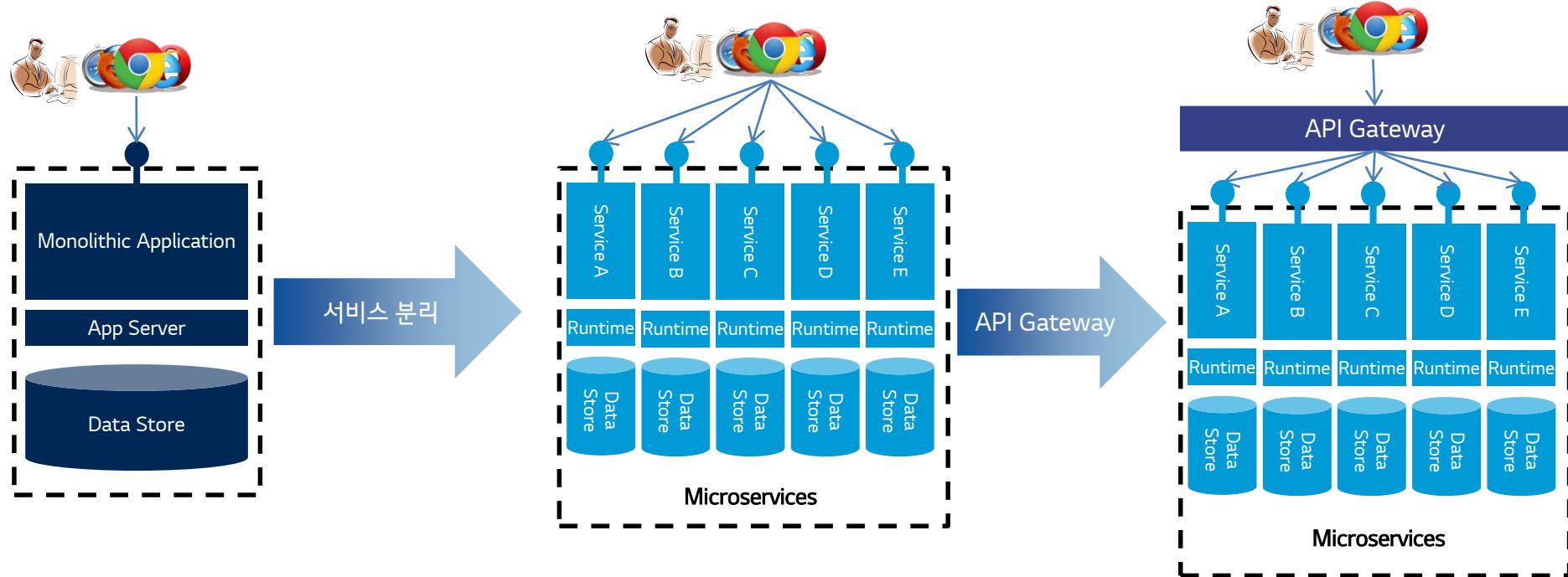
⑤ CI/CD Automation

- 개발, 운영환경 배포시까지의 자동화된 절차와 도구로써 CI/CD 제공

⑥ Telemetry (Diagnostics / Monitoring)

- 분산된 다수 서비스의 실행 상태 로그 수집 추적, 분석 도구

OSS 계열(Google Apigee, Redhat 3scale) 및 상용 계열(CA, IBM) 모두 Full life cycle API 관리를 지원하는 추세이며, 외부의 요청을 내부 API로 연계를 위한 Proxy 역할, Orchestration, 공통기능(AuthN/AuthZ 등) 제공

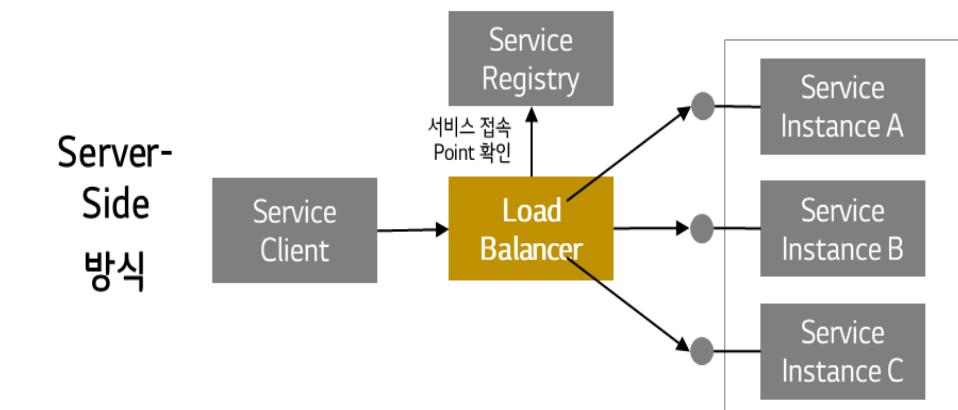
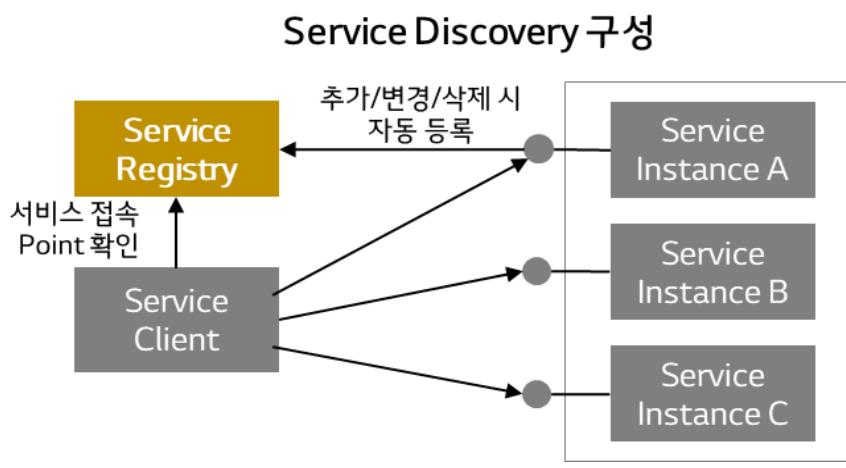
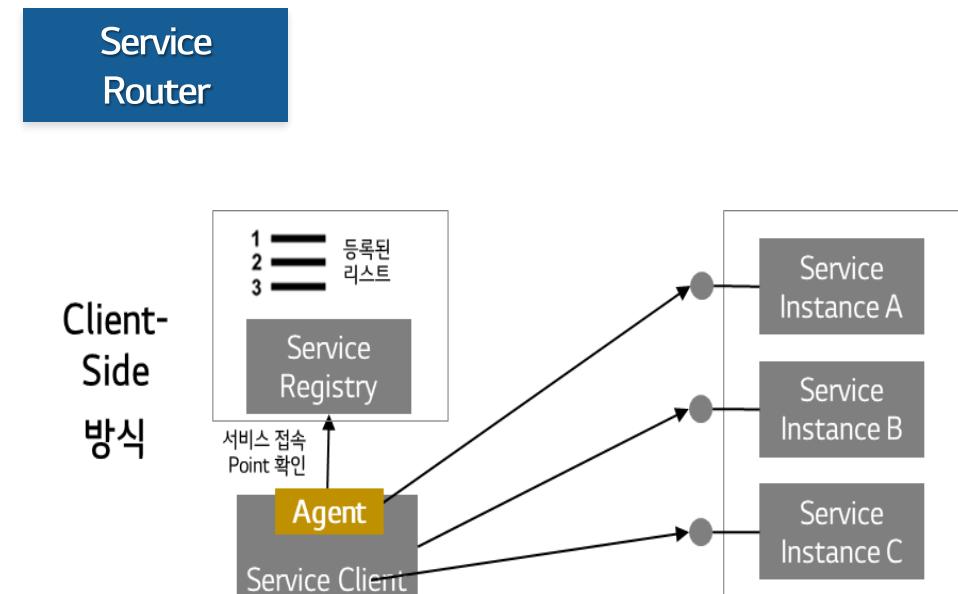
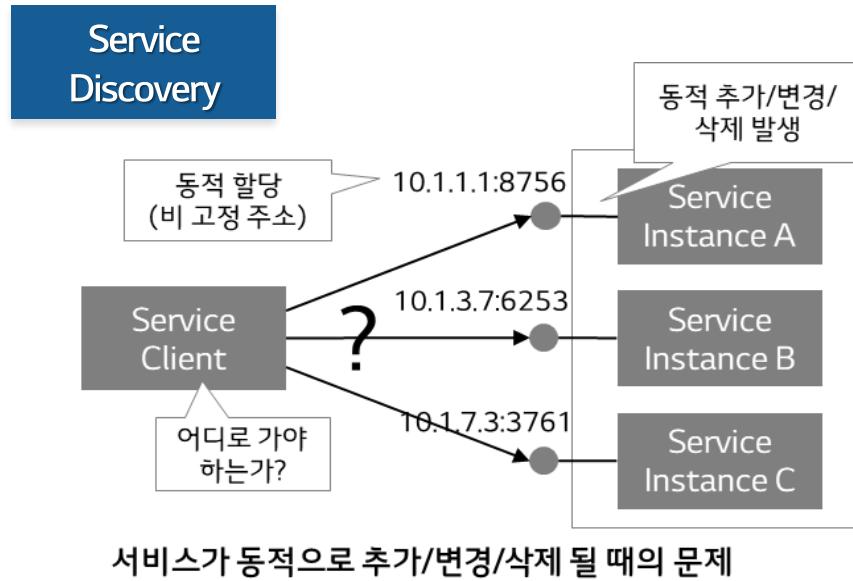


- API Gateway를 Microservices 앞에 두고 모든 Client 요청의 진입 점이 되도록 함 (단일 Endpoint 제공)
- 서비스 들을 포함한 시스템에 대한 아키텍처를 캡슐화하여, 각 Client에 맞게 적용된 API를 제공
- 공통 기능/역할 : 인증/인가 (Authentication/Authorization), 통합 모니터링, Load Balancing 부하 분산 Message Format 변환, API 호출 Logging 등

Service Mesh

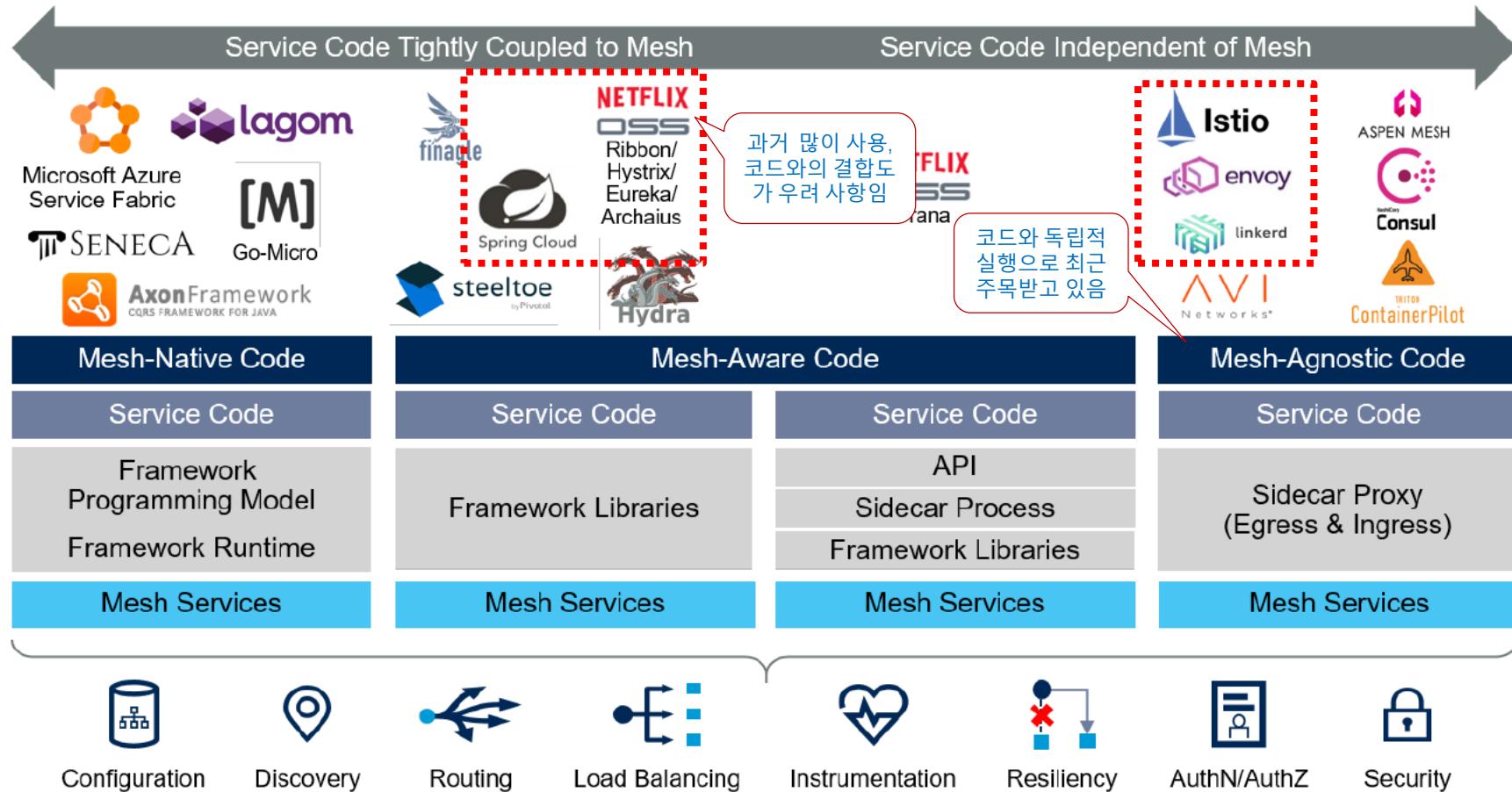
3. MSA Outer Architecture

Service Mesh는 마이크로서비스 간의 원활한 통신을 지원하기 위해 마이크로서비스 외곽에서 관리 및 운영하는 체계로 복잡한 분산 컴퓨팅 문제를 관리하여, 개발자의 부담을 줄이고 생산성 향상에 기여



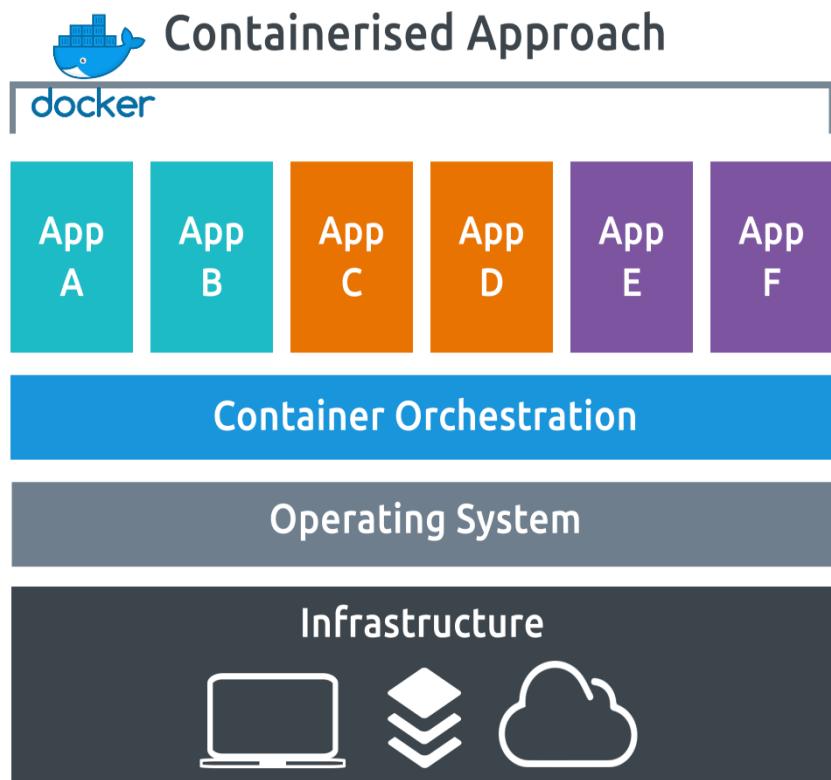
Service Mesh 는 주로 오픈소스 형태로 다양한 솔루션이 존재하며 향후 서비스 코드와의 결합도(의존도)가 적은 솔루션이 주목 받을 것으로 예상됨

A Spectrum of Service Mesh Technologies Based on Code Coupling



컨테이너화된 애플리케이션의 배포, 확장 및 관리를 Container Orchestration Tool 을 이용하여 자동화함

Container 이미지를 저장/관리 및 OSS, Managed Service를 활용한
Container Orchestration 활용 기술 확보



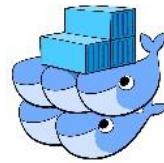
Container 이미지를 저장/관리 및 OSS, Managed Service를 활용한 Container Orchestration 기술 확보

Container Orchestration Tools



MESOS

Marathon (Mesosphere)



Docker Swarm



Nomad (HashiCorp)



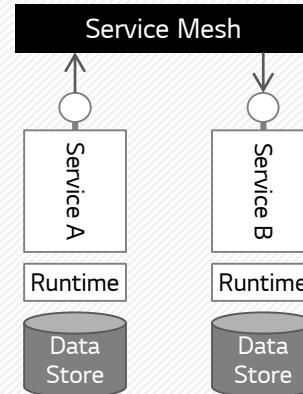
Kubernetes

- 컨테이너 자동 배치 및 복제
- 컨테이너 그룹에 대한 로드 밸런싱
- 컨테이너 장애 복구
- 클러스터 외부에 서비스 노출
- 컨테이너 추가 또는 제거로 확장 및 축소
- 컨테이너 서비스간의 인터페이스를 통한 연결 및 네트워크 포트 노출 제어

Backing Service는 Persistence 상태 또는 공유 데이터에 대한 처리를 위한 역할 제공

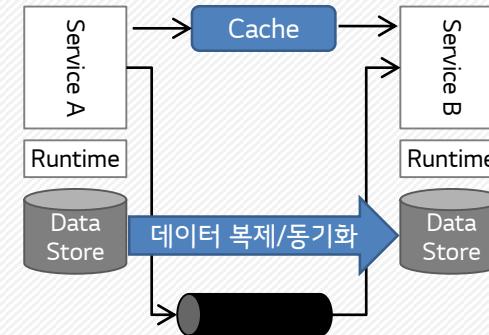
01

Service A에서 관리하는 데이터를 Service B에서 사용해야 한다면?



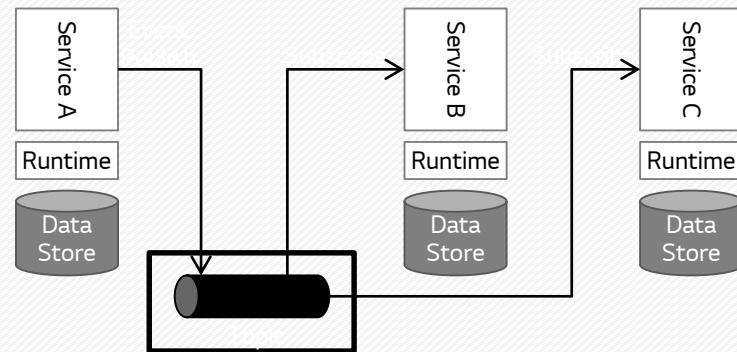
02

타서비스의 데이터 접근이 비번하며, API 호출 과다로 인한 성능 및 자원의 부담이 있다면?
CQRS 또는 데이터 공유 등이 필요하다면?



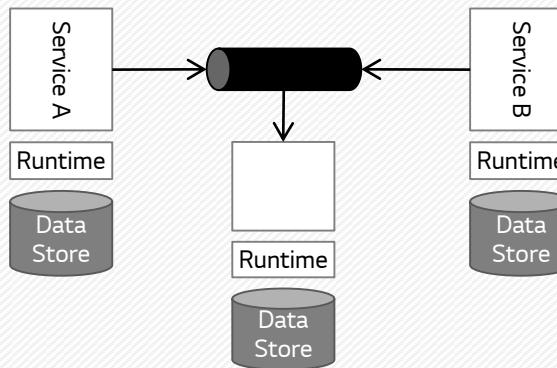
03

Service A에서 처리된 이후 비동기로 다른 서비스들의 처리가 필요하다면? (예: Choreography)



04

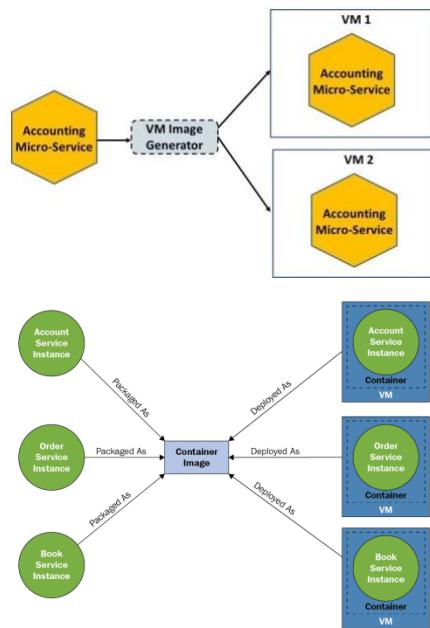
준 실시간 데이터 대상 리포팅이 필요하다면?



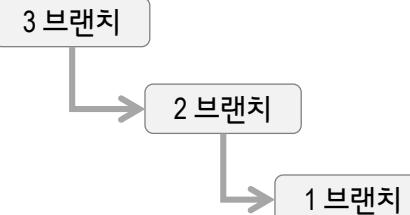
배포 패턴 선택

MSA 환경에서의 서비스 배포 패턴

- Multiple Services per Host
- Single Service per VM
- Single Service per Container
- Serverless deployment

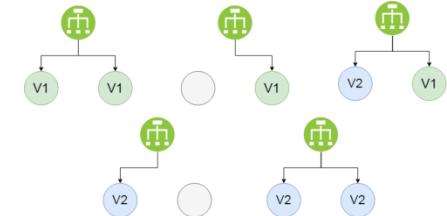


브랜치 전략 수립

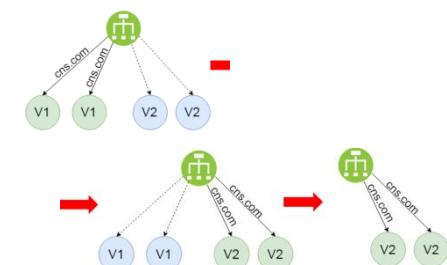


배포 기법 선택

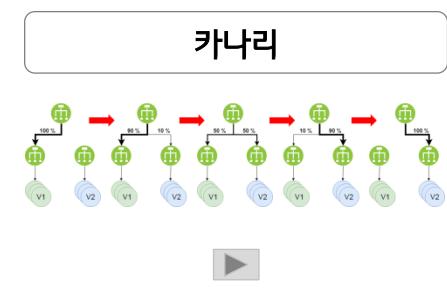
롤링 업데이트



블루/그린

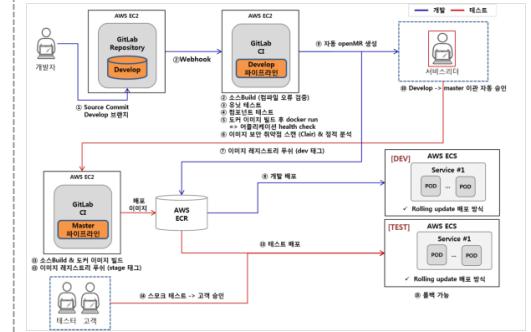


카나리



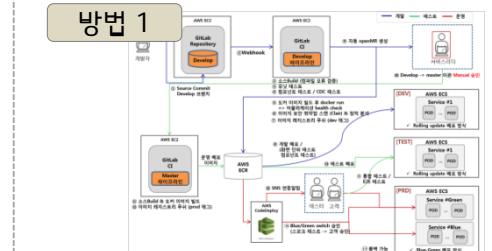
프로젝트 단계에 따른 CI/CD 프로세스

개발 단계의 CI/CD

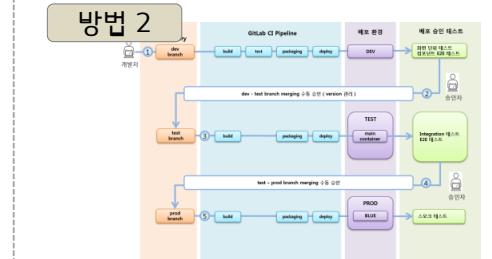


개발/운영 통합 단계의 CI/CD

방법 1



방법 2



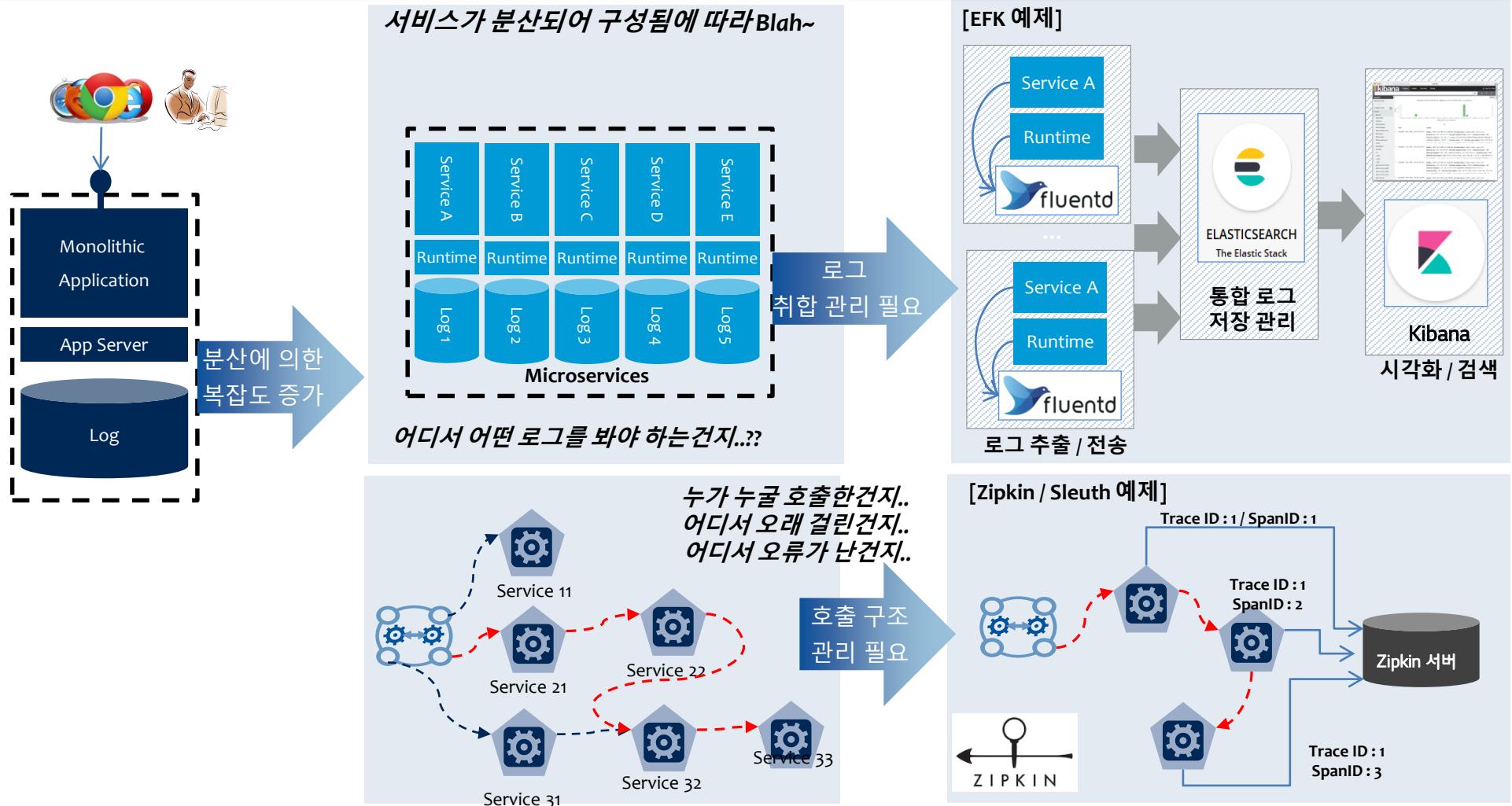
Telemetry/Observation (가시성)

3. MSA Outer Architecture

Telemetry 는 모니터링, 통보, 로깅, 추적, 분석을 위한 서비스 제공 역할 수행

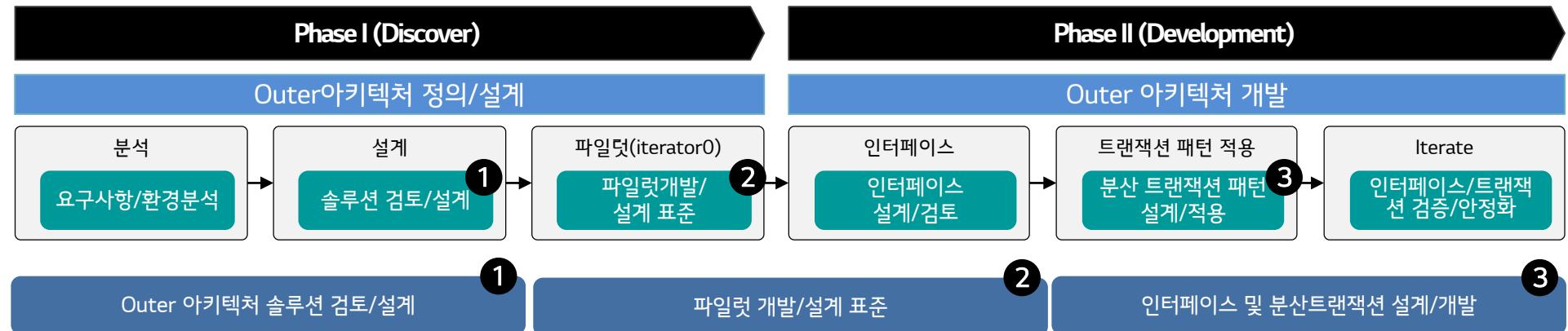


Telemetry – 로깅, 추적, 모니터링

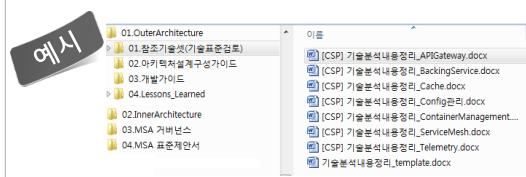


LG CNS Outer Architecture

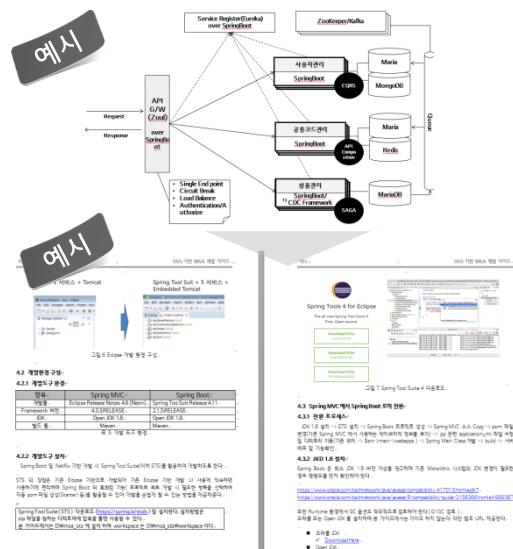
3. MSA Outer Architecture



- 분석단계 요구사항 및 환경 분석 자료를 통해 필 요 솔루션 리스트 업
 - Outer 솔루션 평선 매트릭스 분석에 의한 적합 솔루션 선정

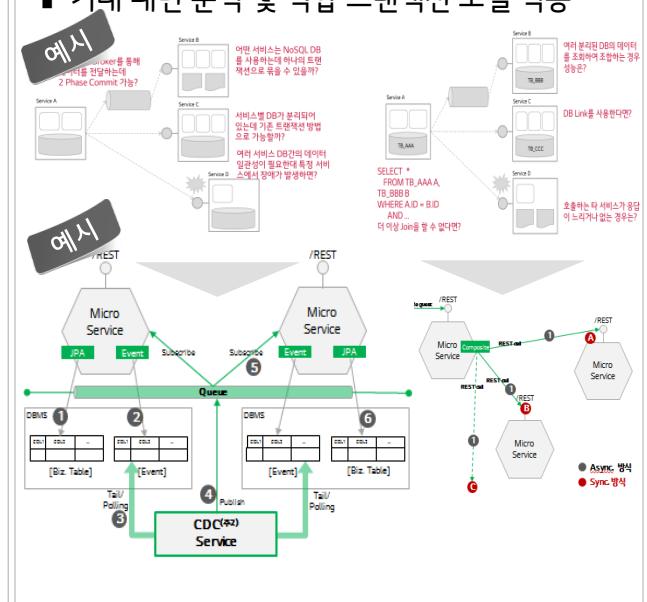


- MSA 개발 가이드 참조를 통한 파일럿 개발
 - OSS/CSP 각각 대표 솔루션 개발 가이드 참조를 통한 파일럿 개발



- Inner 아키텍처에서 나온 인터페이스 설계 및 분산 트랜잭션 설계

■ 거래 패턴 분석 및 적합 트랜잭션 모델 적용



LG CNS Outer Architecture

3. MSA Outer Architecture

구분	제작사	제품명	참조기술셋	개발가이드	운영환경				예상 /
					On Premis	AWS	Azure	Google	
WEB Svr	OSS	Apache Web Svr	●	●	✓	✓	✓	✓	
	OSS	NGINX	●	●	✓	✓	✓	✓	
WAS	OSS	SpringBoot	●	○	✓	✓	✓	✓	
	LG CNS	DevOnBoot		○	✓	✓	✓	✓	
API Gateway	Amazon	API Gateway	●	●		✓			
	Google	Apigee							
	IBM	API Connect	○						
	MS	Azure API Management	○					✓	
	Netflix OSS	Zuul	●	●	✓	✓	✓	✓	
	Red Hat	3scale							
Service Mesh	Amazon	App Mesh	●	●		✓			
	Apache	Zookeeper	●	○	✓	✓	✓	✓	
	Google	GCSM(Google Cloud Service Mesh)							
	MS	Azure Fabric Service Mesh	○					✓	
	Netflix OSS	Archaius							
	Netflix OSS	Eureka	●	○	✓	✓	✓	✓	
	OSS	gRPC							
	OSS	isto	●	○	✓	✓	✓	✓	

범례		
완료	일부제공	예정
●	◐	○

Contents

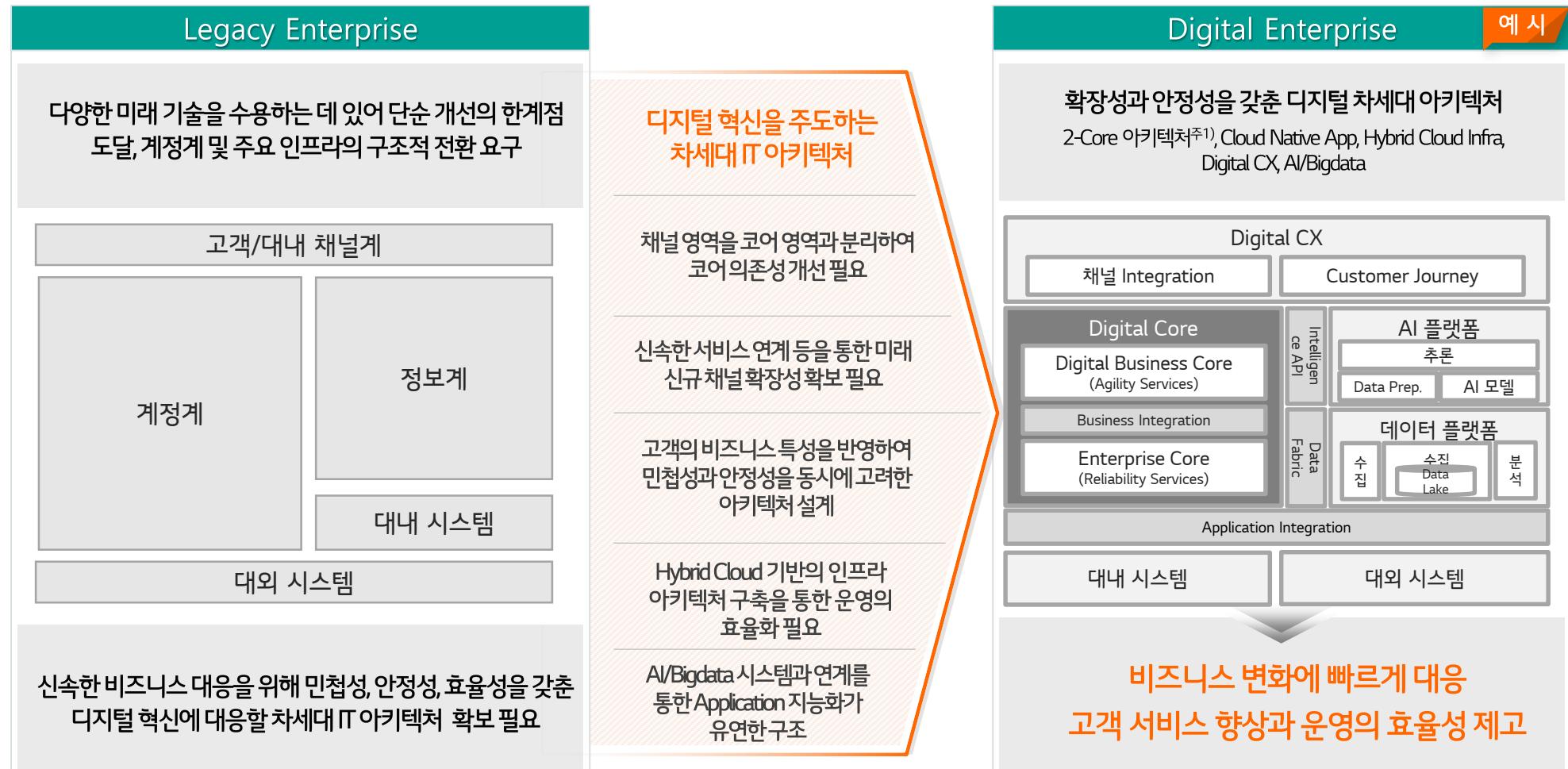
1. Application Modernization 개요
2. MSA 성공을 위한 핵심 요소
3. MSA Outer Architecture
4. LGCNS MSA-Core System 적용 및 AM체계

-  **MSA를 해야만 하는 이유와 구조적 문제점**
-  **Micro Service 쪼개기**
-  **반드시 고려해야 하는 Critical Design Issues 및 테스트**
-  **Outer Architecture - MSA의 기반 아키텍처**
-  **Core System에 MSA 적용하기**

Core System에 MSA 적용하기

4. LG CNS MSA

▶ Digital Enterprise를 위한 차세대 IT 아키텍처



주1) 비즈니스 유형에 따라 안정성과 민첩성을 추구하는 IT 아키텍처

▶ 디지털 차세대 Core System을 위한 2-Core 아키텍처

◆ 디지털 차세대 2-Core 아키텍처 ◆		
기존 차세대	Enterprise Core	Digital Business Core
• 채널별로 산재되어 있는 비즈니스 로직	신뢰성(Reliability)	민첩성(Agility)
• 전시/이벤트 등 고객 트래픽 증가에 적시 대응 부족	효율성, 안정성	고객 경험, 빠른 대응
• 지역/장애 등의 사유로 안정적이지 못한 비즈니스 코어 서비스(주문 서비스 등)	접근방향 IT-centric	Business-centric
• 시스템의 복잡성 증가로 인한 유지보수 생산성 저하	적용대상 핵심 업무(back-office)	고객 접점 업무(front-office)
• Cloud 및 디지털 기술의 적용이 어려운 시스템 구조	주기 장기(months)	단기(days, weeks)
• AI/CC, 분석 시스템과의 연계 시 구조적 한계	거버넌스 계획 중시, 승인 기반	실증적, 지속적 반복
	클라우드 On-Premise, Private Cloud	Public/Private Cloud
	서비스 크기 Coarse-grained Service	Fine-grained Service

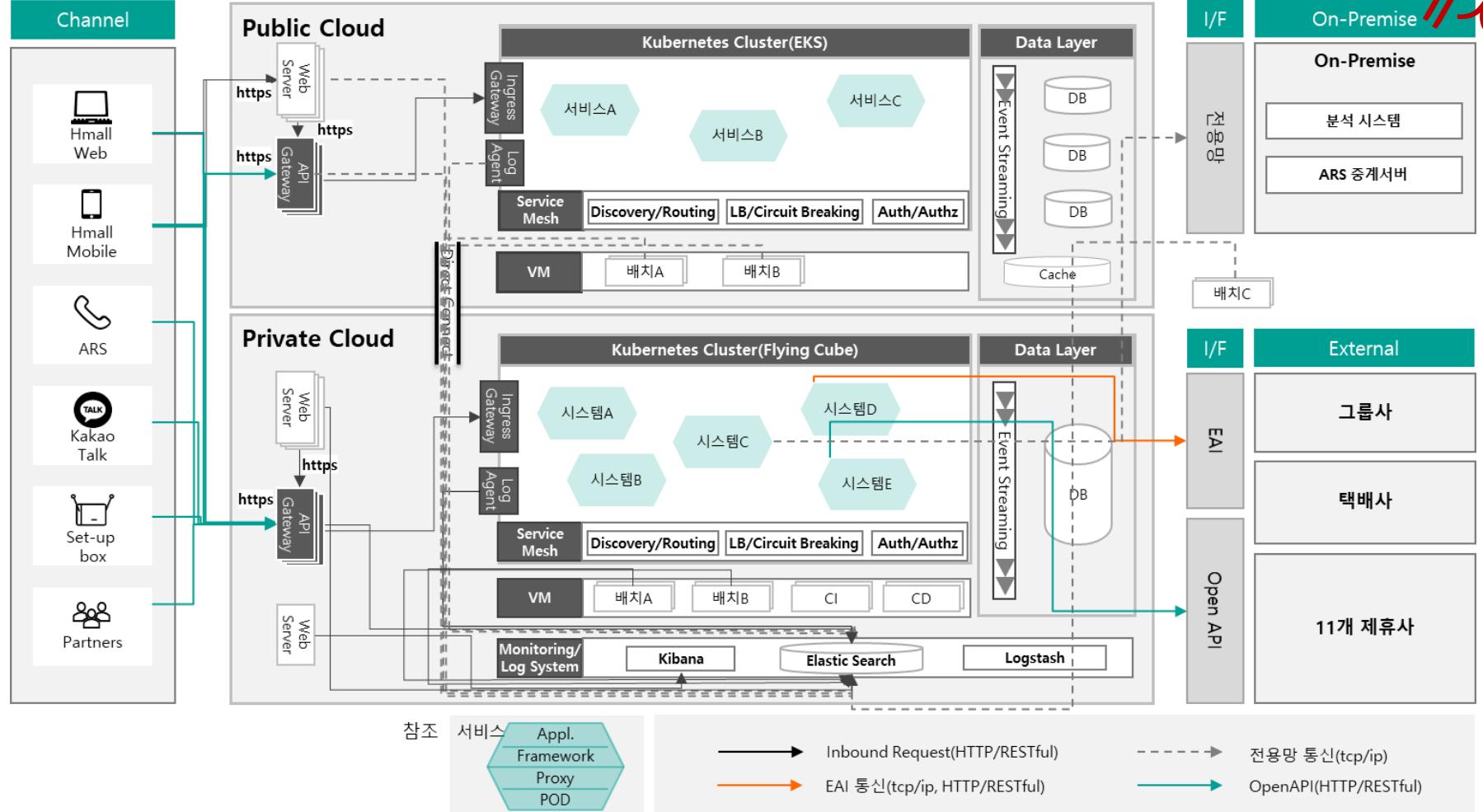
주1) Fine-grained Service : 민첩성이 필요한 작은 단위의 업무를 구현한 서비스

주2) Coarse-grained Service : 데이터 정합성 및 안정성이 필요한 중규모 이상의 업무를 구현한 서비스

Core System에 MSA 적용하기

4. LG CNS MSA

S/W 아키텍처 구성도 (예시)



구현 기술

MicroServices

일하는 방식

Agile

조직/Process/기술

DevOps

인프라/Runtime

Cloud / Container

* Enterprise
MSA
Hexagon
Methodology

MicroServices
Discovery

서비스 정의
서비스 시뮬레이션
서비스 AI 추천

MicroServices
Design & Develop

API First Design
서비스설계/개발
서비스Visualization

MicroServices
Test

Component Test
Integration Test
API Contract Test

MicroServices
Delivery

CI/CD Pipeline
테스트 자동화
인프라 프로비저닝 자동화

DevOn
MSA Suite

Event
StormingDomain
Driven DesignFunction
Decomposed
DesignAPI
GatewayService
MeshBacking
Service

Telemetry

MSA SWB & DevOn Simulator

DevOn Modeler & MSA SDM

DevOn Tester & DevOn API

Agile Practice

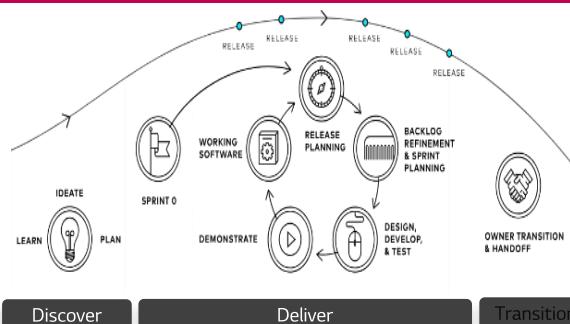
Scaled Agile

Product Backlog	Scrum Meeting
Sprint Backlog	Sprint
Sprint Review	Retro retrospective

eXtreme Programming

Pair Programming
Test Driven Development
Continuous Integration

Product Engineering Methodology



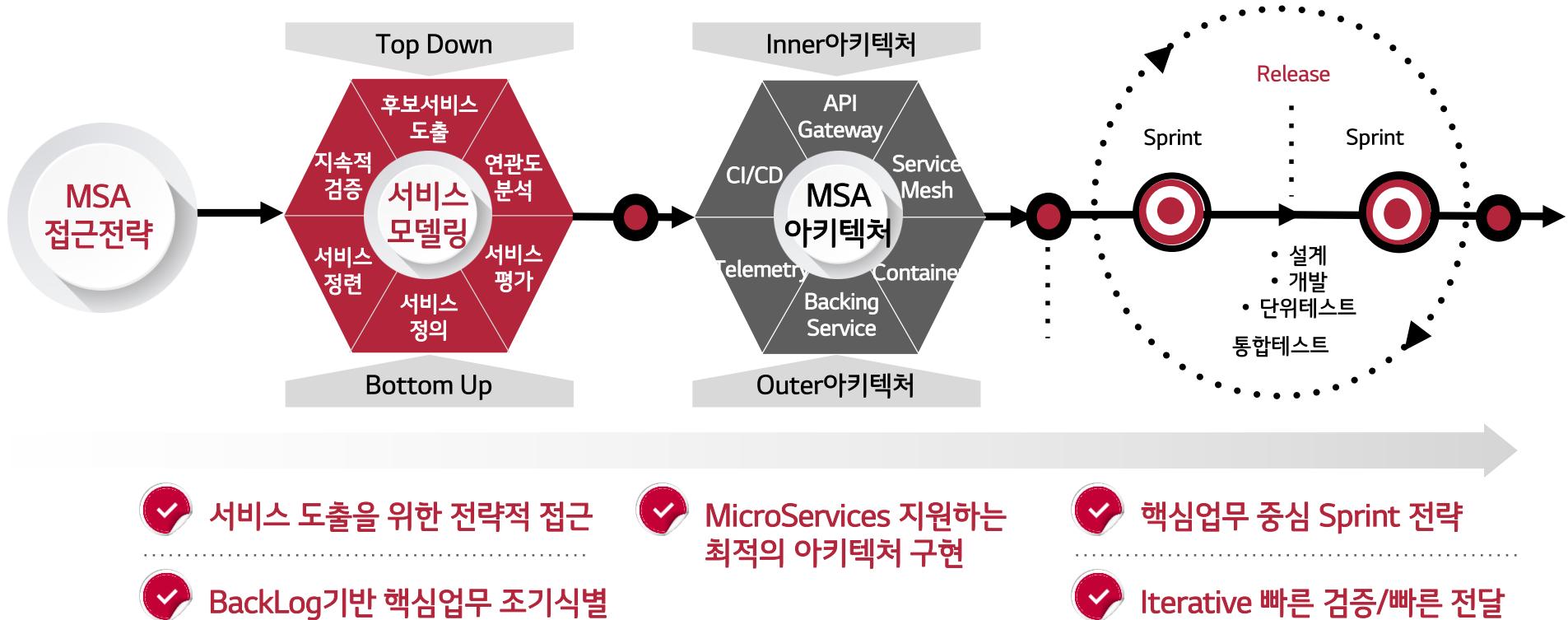
PEM
Large Program

SAFe

Program Definition
Product Discovery
Product Delivery & Pod Additions
Program Increment
Agile Release Train

Enterprise MSA Hexagon 방법론

LG CNS Way4U – MSA경로

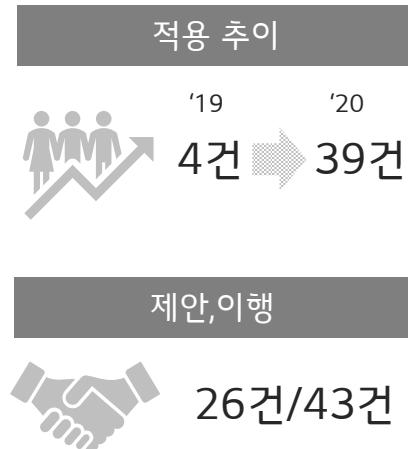
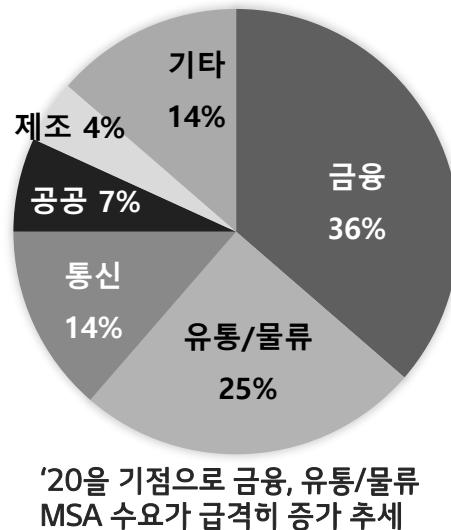




MSA 구축사례

LG CNS 사례 분석

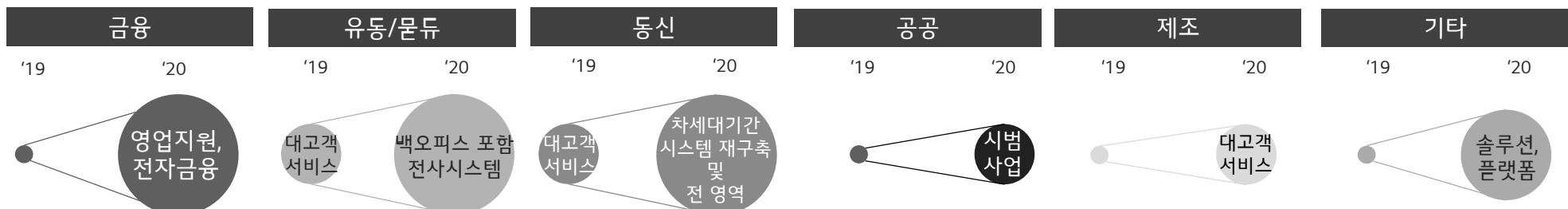
산업별 AM 적용 동향



'19~20년 LG CNS에서 MSA도 설명회/제언, 제안, 이행 프로젝트 수행 건

분류	제언	제안	사업화(이행, POC)	계
금융	8	2	6	16
유동/물류	6	3	2	11
통신		1	5	6
공공	2		1	3
제조			2	2
기타	1		5	6
계	17	6	20	43

산업별 AM 적용 현황



- 보험, 생명 중심 AM 관심 급증, 채널 위주 MSA 개발
- 은행은 비금융 업무 MSA 적용, 채널 중심 MSA 시도

- 쇼핑몰 등 대고객 채널 중심 MSA 도입 시스템 구축
- 최근 영업, 물류 영업 등 백엔드 코어도 확장
- 그룹 동신사 중심으로 AM 적용 분야를 전 시스템 영역으로 확대

- 종합기상정보시스템 MSA 기반 구축/운영
- 심사평가원 등 공공기관 사업 도입 검토

- B2C 채널 대고객 서비스 중심 진행
- MSA 기반 솔루션 개발 (LG CNS)
- MSA 도입 서비스 개발이 가능한 Platform 출시 등

Why? 왜 쪼개려고 하는 것인가?

How? 어떻게 쪼갤 것인가?

What? 어떤 기술로 구현할 것인가?

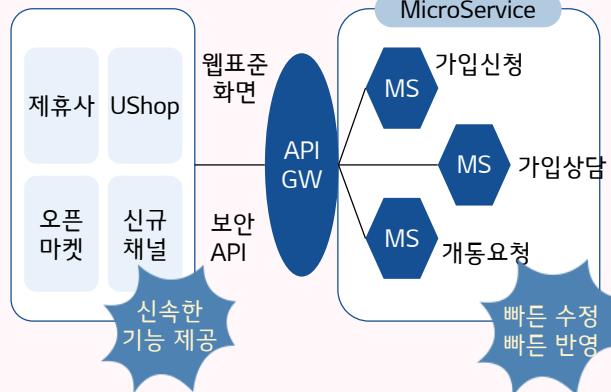
What? 어떤 방식으로 이행할 것인가?

Agility로 비즈니스 경쟁력을 확보하는 LG CNS AM

LG CNS는 기술리더십과 다양한 형태의 사업에 AM을 적용한 경험을 기반으로 기업의 비즈니스 전략을 신속하게 반영하고 경쟁력을 확보할 수 있는 Next Architecture를 구축할 수 있음

L통신사-채널 온라인 가입 지원 시스템 구축

- 온라인 영업 채널 경쟁력 강화를 위해 비즈니스 Agility 확보, 클라우드 환경으로 전환



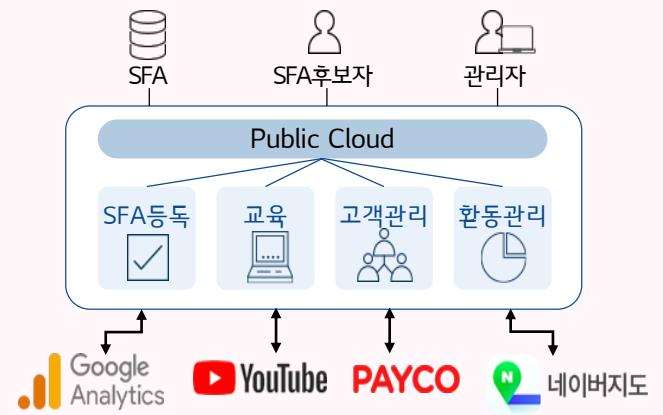
사업적 Needs 적기 대응

트래픽 증가 시에도 수용 가능

정기/긴급 배포 → 중단 없이 수시 배포

H금융사 모바일 SFA 시스템 구축

- 채널 확장 및 최신 디지털 트렌드 반영을 목표로 MSA, Agile, DevOps 등 AM 적용



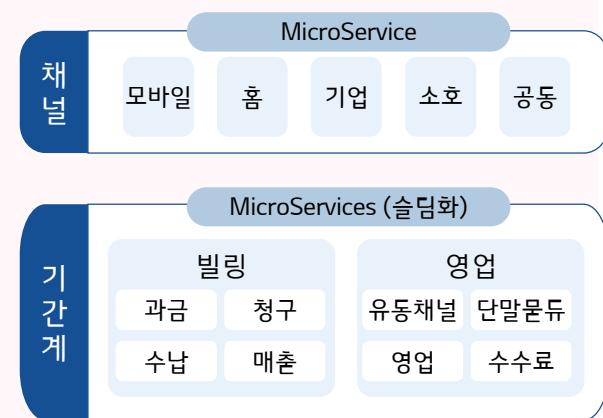
신속하고 유연한 채널 확장 및 서비스 배포

설계사의 업무 효율화 및 상향 평준화

영업 경쟁력 확보

L유통/서비스사 차세대 시스템 구축 사례

- 빠른 상품 및 서비스 출시와 연속성 확보 등을 목표로 채널과 기간계 시스템에 AM 적용



새로운 상품, 서비스 출시 시간 단축

평균 개발 시간 단축(42일→14일)

지속적인 기간계의 슬림화 유지

Case 1

Case 1 (통신)

온라인 영업채널의 경쟁력을 확보하고자 MSA 적용을 통한 비즈니스 Agility 확보, 노후화된 인프라 개선을 위한 Cloud로 전환하였고, 이를 통해 Biz 전략을 신속하게 반영함으로써 온라인 경쟁력을 확보할 수 있었습니다

배경

- 신규 영업 채널 생성시 개별화면 및 연동 모듈 개발 등 사업적 대응 어려움
- 낙후된 시스템으로 인해 신규 채널 확대 시 지연 현상 발생
- 영업 경쟁력 강화를 위한 필요기능 부재 및 불편한 UI

주요 적용 사항

1 기능 단위로 서비스 정의 및 Agile 방식 적용

이벤트 기반 스토리 도출과 평가도 마이크로서비스 후보군 도출
가입신청부터 개동에 이드는 서비스간의 상호관계 및 연계
복잡도 고려하여 최종 11개 선정

2 Public Cloud 기반 마이크로서비스 아키텍처 설계/구축

Infra 요구사항에 On-Time 대응: Public Cloud 구성, Auto Scaling
장애 확산 방지, 배포 속도 향상(컨테이너 활용)

3 UI/UX 개선을 통한 사용자 편의성 및 상담사 효율 증대

Value

사업적 Needs 적기 대응

- ✓ 독립된 서비스 체계로 고객 Needs 즉시 반영
- ✓ 업무 변경 발생 시 시스템 반영 속도 개선

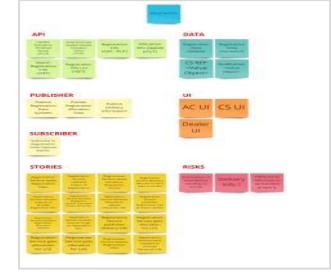
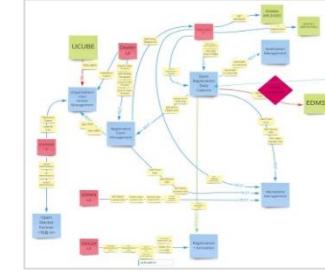
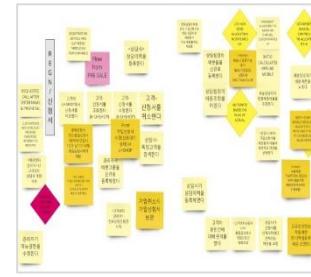
트래픽이 증가되어도 수용 가능

- ✓ 이벤트, 프로모션 발생 시 Auto Scaling
- ✓ Container Auto Scaling, 인프라 On-Time 대응

정기/긴급 →
배포 중단 없이
수시 배포

Event Storming 부터

Event Storming 방식으로 서비스 도출, Agile 방식 적용



6개월간의 서비스 구현 및 오픈



진행 방식

GOAL



1. 업무 독립성

2. 요구사항 반영 용이

3. 업무 복잡성 감소

4. 장애복구/격리

5. IT업무 효율화

6. 확장성

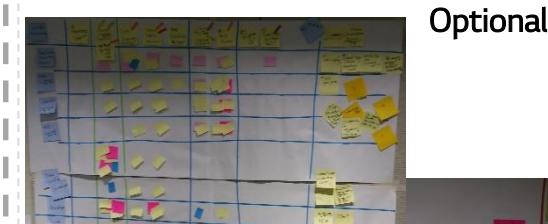
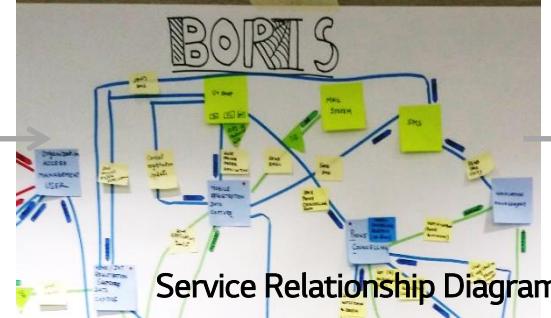
7. Data Ownership

8. Team/Org Structure

EVENT STORMING

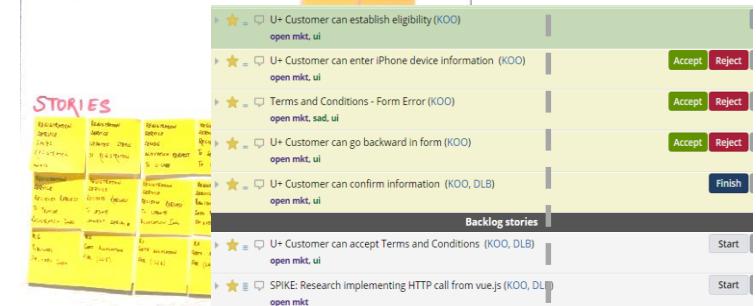
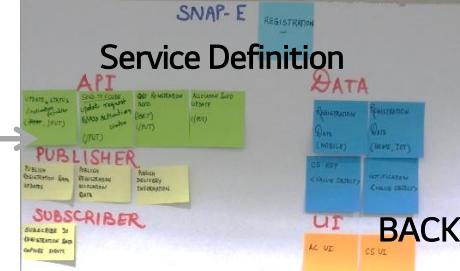


Service Relationship Diagram



Optional

Service Definition



BACKLOG/IPM

IMPLEMENTATION



Agile

Case 2

Case 2 (생명/보험)

금융계열사의 표준 모바일 플랫폼을 퍼블릭/마이크로서비스 기반으로 구축하는 금융권 최초 사례로
마이크로서비스 기반으로 서비스 모델링, 아키텍처 구현, GitLab 기반의 DevOps 기술/환경 적용하였습니다

배경

- 디지털 기반의 채널전략 및 판매 프로세스 혁신 필요
- 모바일 중심 소비활동 강세
강력한 고객 트래픽 기반의 디지털 판매채널의 등장
- 모바일 플랫폼을 통하여 디지털 혁신 및 시장 선도

주요 적용 사항

1 공통(Shared) 영역과 회사 특화된 서비스 분리

- 공동 업무에 대한 서비스 영역
- 회사별 변이 서비스 영역 분리 및 추가 채널에 대한 확장 고려
- 내부 서비스 간 연계 33개 API 도출(Sync 31개, Async 2개)

2 금융권 SI 환경 특성을 고려한 Agile 방법론 적용

- 개발/단위테스트 후 업무 중심 동합 테스트 수행하는 Iteration 구성은 동한 점증적인 오픈 및 유연하게 고객요구사항 대응

3 AWS 기반 MSA Architecture 반영

- MSA분리/경량화하여 Kubernetes/컨테이너도 서비스 구현
- CI/CD DevOps 환경을 통한 신속한 서비스 배포/관리
- AppMesh 및 X-ray, EFK Stack 서비스 활용
 - . 외부 : API Gateway(AWS)
 - . 내부 : Service Mesh(App Mesh)
 - . 거래 추적 : Cloud watch, X-Ray
 - . Message : SQS 적용

Value

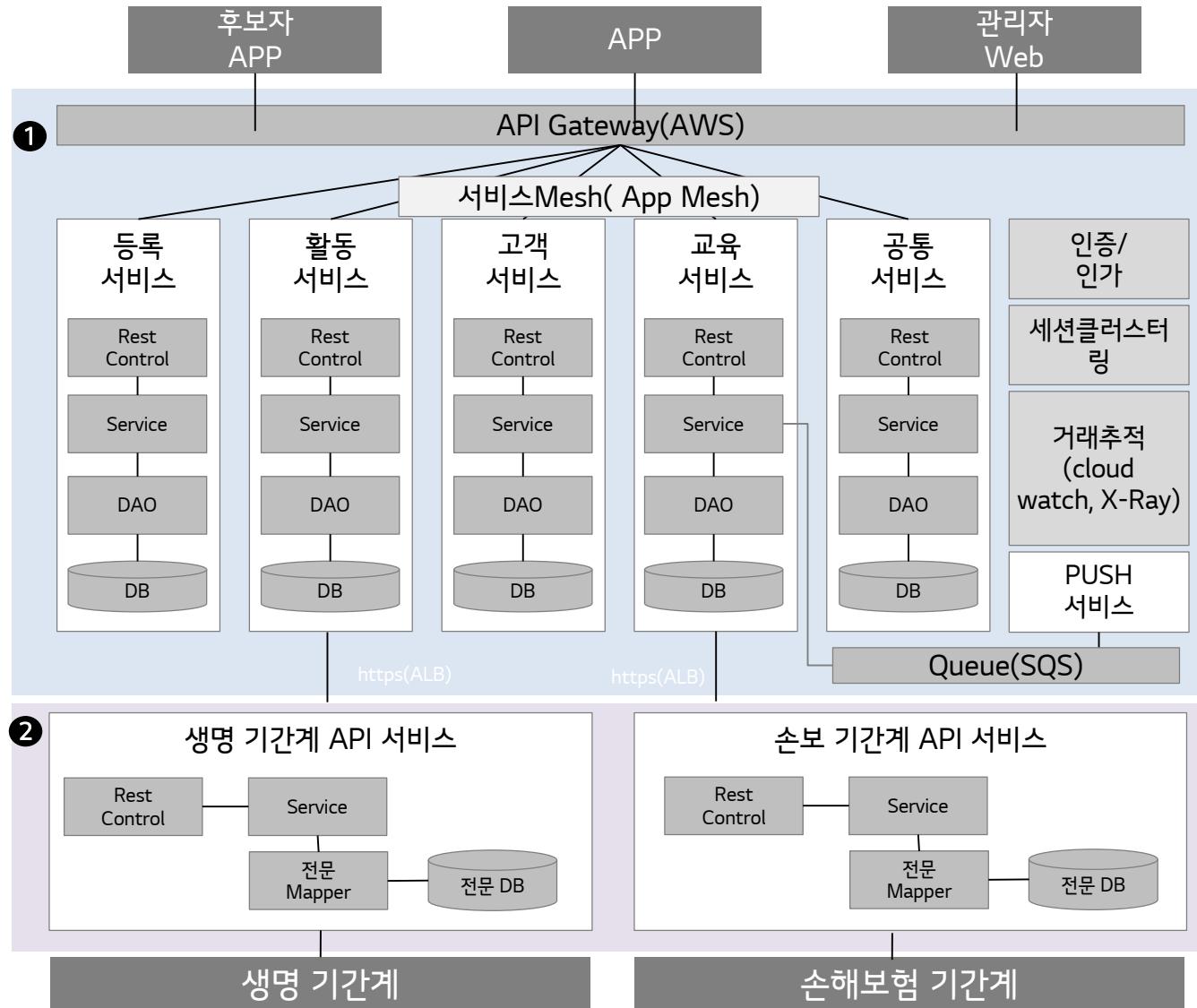
금융권 최초 사례

- 퍼블릭/마이크로서비스 기반
금융계열사의 표준 모바일 플랫폼 확보
- 기술력에 대한 확신을 고객 전달
프로젝트 추가 수주

서비스 확장 및 빠른 변화에 대응

- 채널 확장에 유연한 시스템 구조 확보
- 마이크로서비스 기반의 DevOps 기술/환경

아키텍처 구성



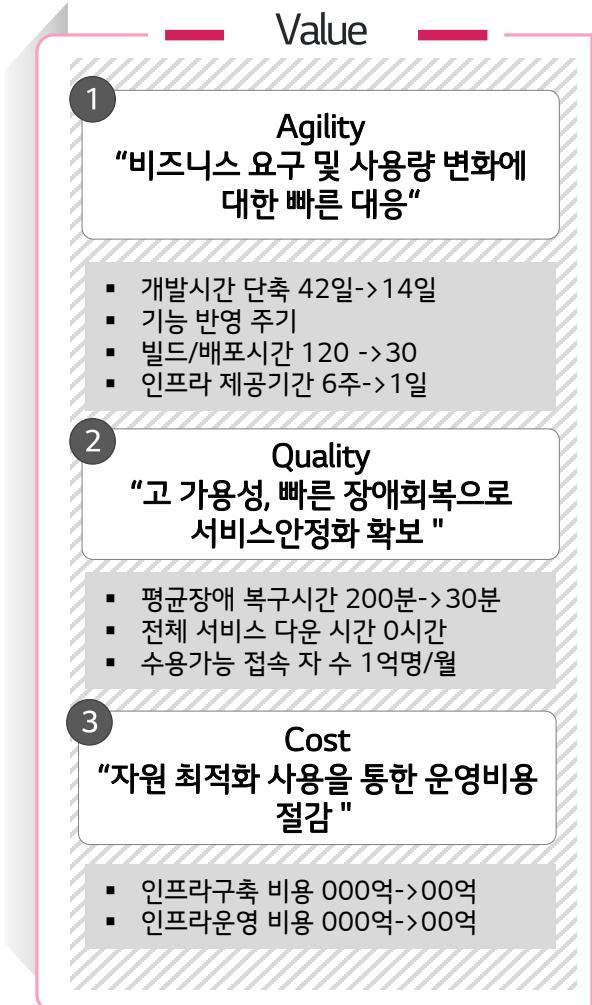
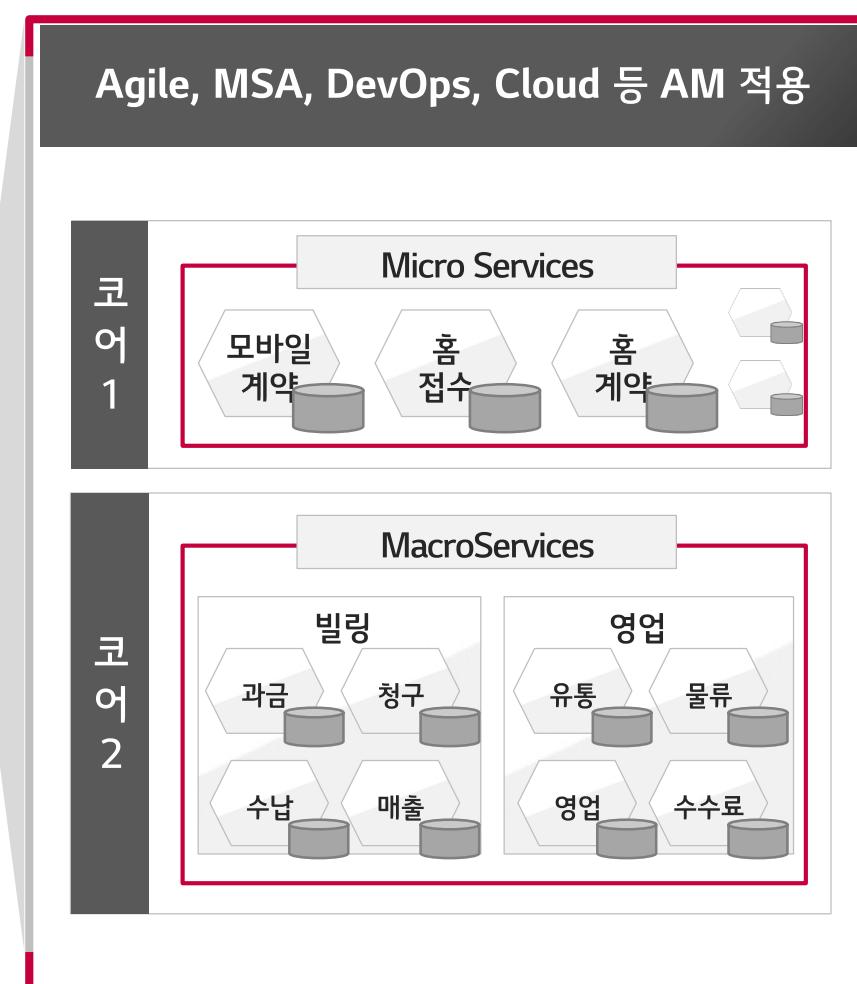
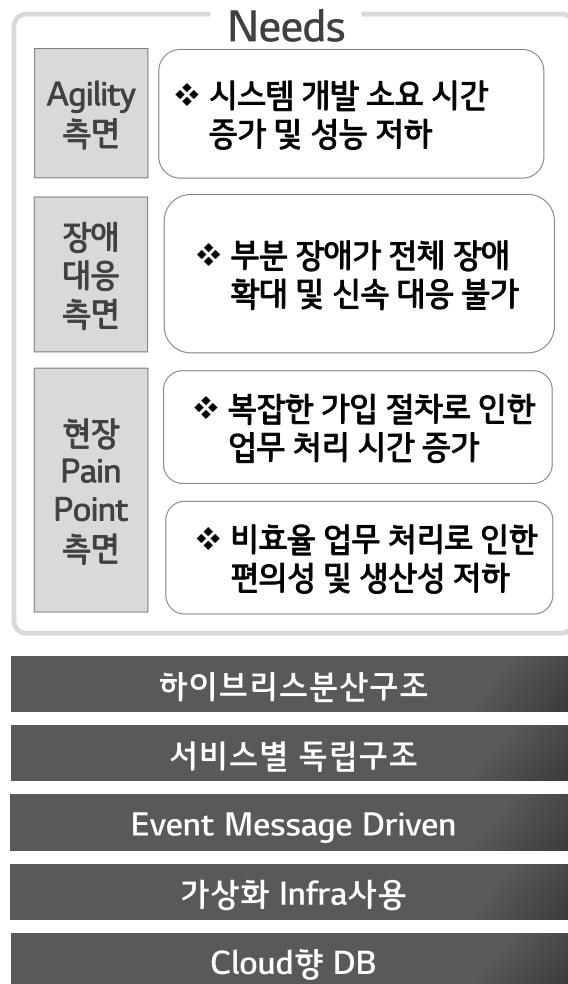
MSA 아키텍처 특징

- 공통(Shared) 영역과 회사 특화된 서비스 분리를 통한 비즈니스 확장성 확보
 - ① 공통 업무에 대한 서비스 영역
 - ② 회사별 변이에 대한 서비스 영역 분리
 - 추가 채널 혹은 회사 추가에 용이하게 구성
- AWS 기반 MSA Architecture 반영
 - Framework : DevOn Boot FW 최초 적용
 - 세션 클러스터 : ElastiCache(Redis기반)
 - 서비스 연계 :
 - . 외부 : API Gateway(AWS)
 - . 내부 : Service Mesh(App Mesh)
 - 거래 추적 : Cloud watch, X-Ray
 - Message : SQS 적용
- 서비스 간 연계 최소화 단위 서비스 구성
 - 내부 서비스 간 연계 33개 API 도출
 - (Sync 31개, Async 2개)

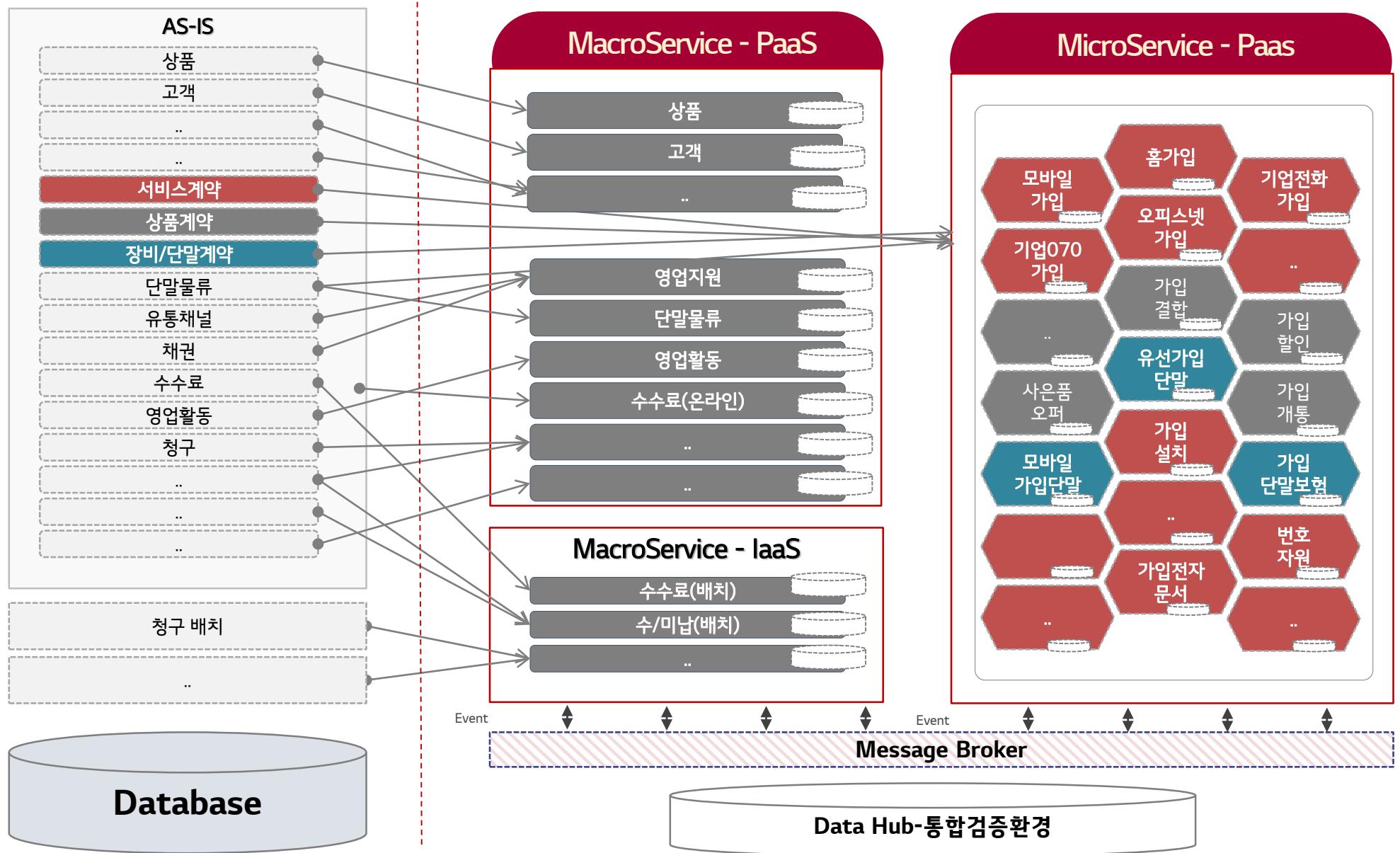
Case 3

Case 3 (통신)

L사의 차세대 시스템은 빠른 상품 및 서비스 출시, 서비스 연속성 확보 등을 목표로 채널과 기간계 시스템 전체에 Application Modernization을 적용하여 현재 프로젝트 진행하고 있습니다



Core System MSA 예시(Macro-Micro)





MSA DDD 적용사례

Application을 민첩(Agility)하게 만들려면?

복잡성을 제거하여 단순하게

관심사와 경계를 나누는 것

독립성을 확보하는 것

- Application의 복잡성은 경계 없는 Table간의 직접 참조
- Data의 관심사를 분리하고 캡슐화
- 문제해결을 위한 필요한 기능으로 추상화

- 비즈니스 업무영역(도메인)위주로 분리
- 도메인은 이벤트를 발생
- 역할계층분리(Layered Architecture)
- Aggregate Oriented Model

- 데이터의 캡슐화
- 도메인 간의 참조는 데이터가 아니라 이벤트
- 이벤트를 통한 정보 참조
- Pub/Sub 구조의 이벤트 반행/구독
- 독립적인 배포 가능
- 독립적인 DB 사용

MVC패턴

DB중심이 아닌 응용중심

Domain Model 패턴

이벤트 기반 비동기

Aggregate 설계

JPA

Domain Driven Design



DevOps Pipeline Delivery

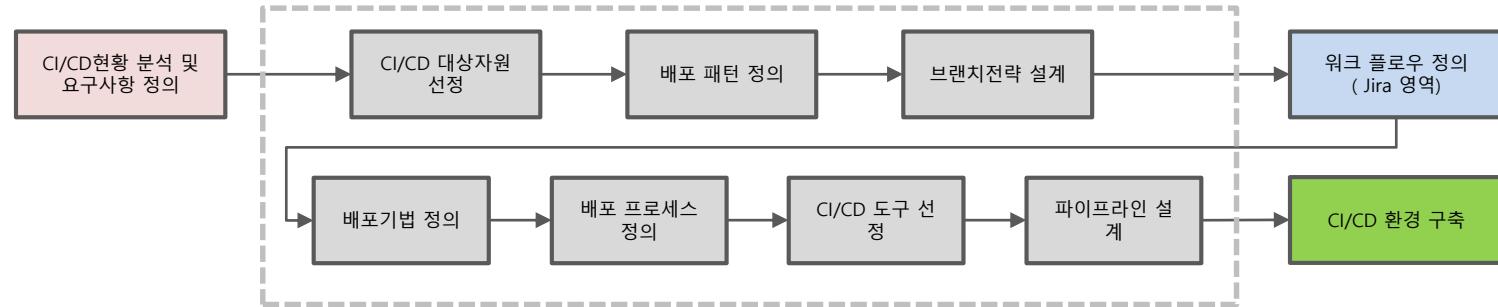
LG CNS Architecture Service Center

Hybrid Cloud MSA 프로젝트 CI/CD

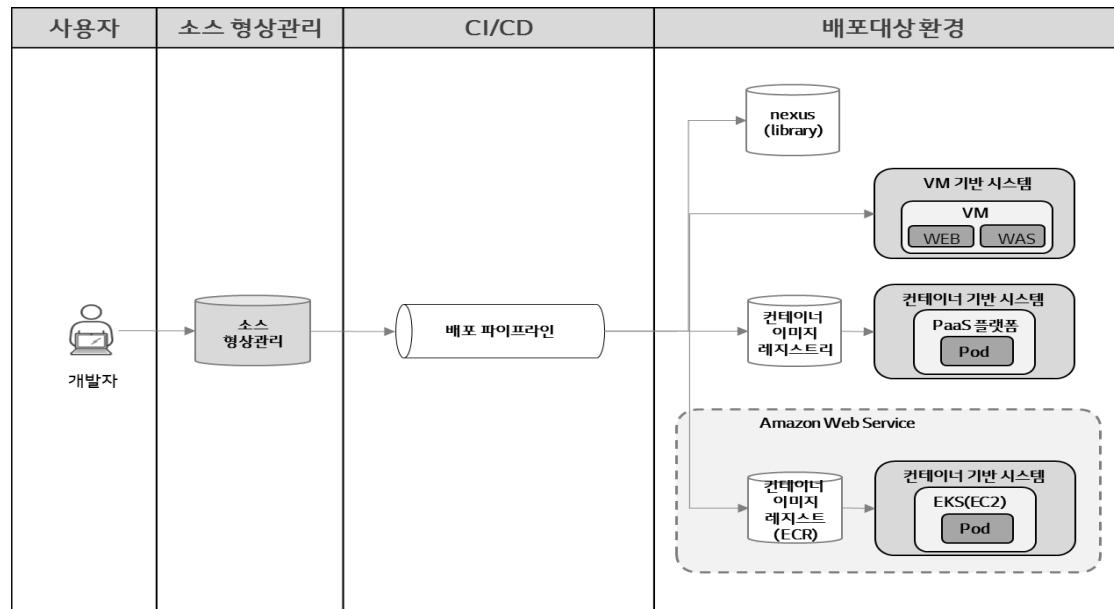
Hybrid Cloud MSA 프로젝트 CICD 사례

L사의 차세대 시스템은 Public/Private Cloud, VM 환경의 복잡한 환경으로 Build에서 Delivery까지 Continuous Integration / Continuous Deployment 환경을 OSS기반으로 최적화하여 구성함

CI/CD 구축 절차



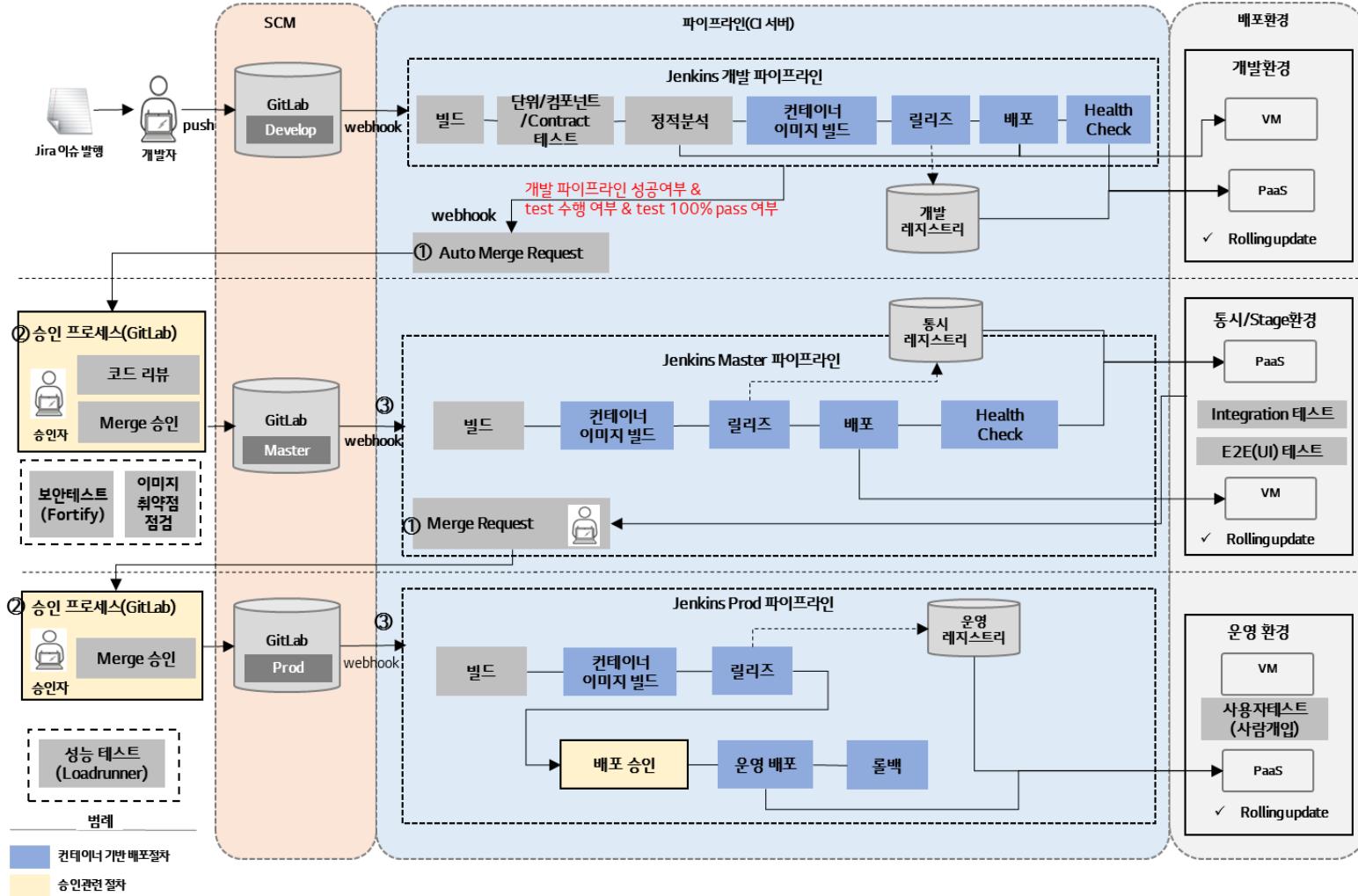
Hybrid Cloud 배포패턴



Hybrid Cloud MSA 프로젝트 CICD 사례

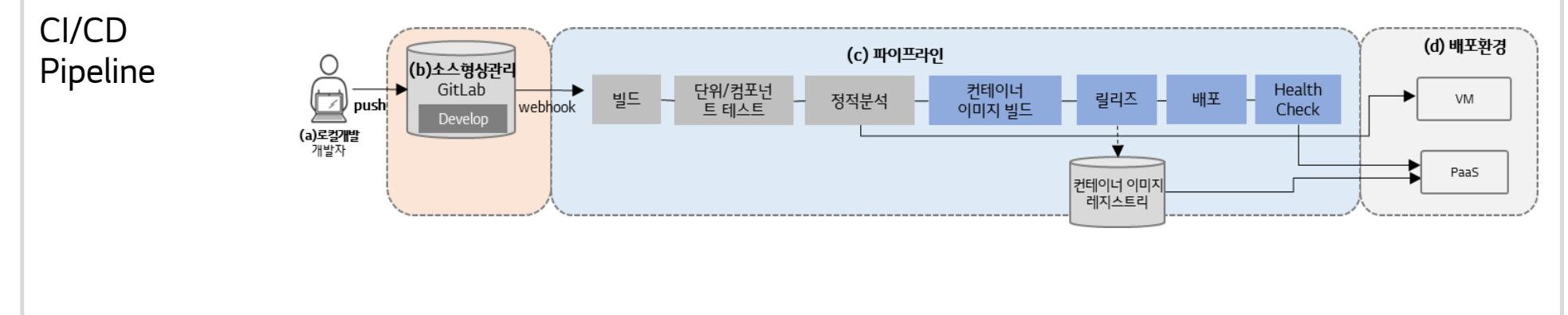
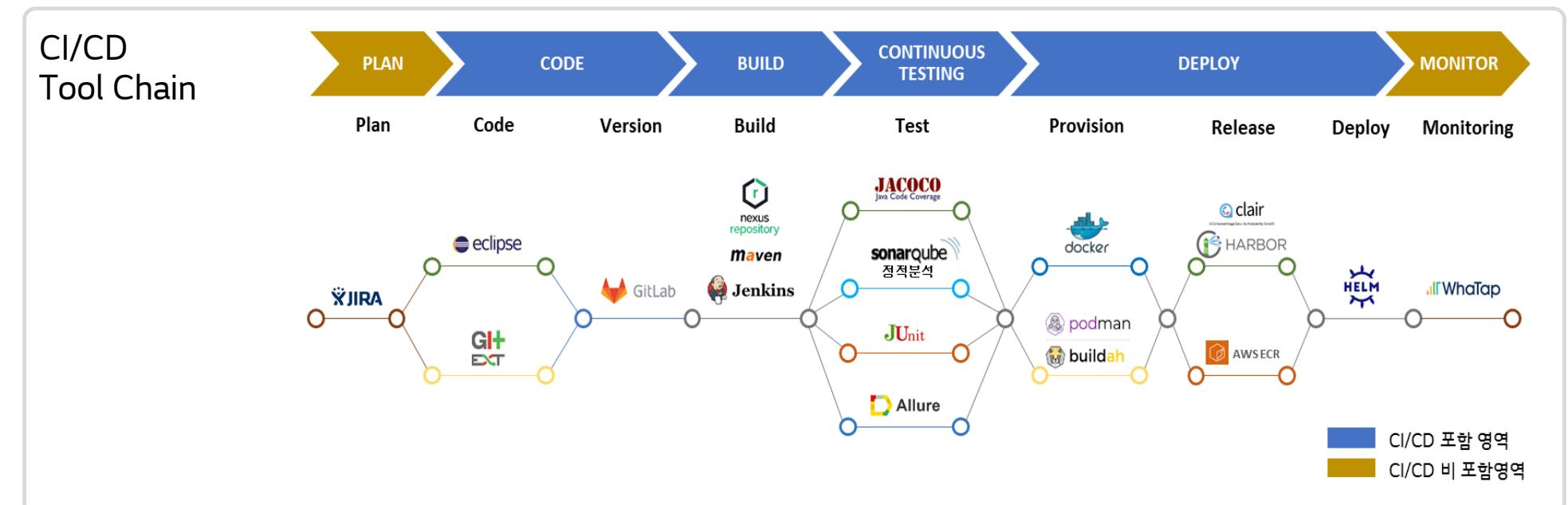
브랜치 전략은 SI는 Two 브랜치, 운영은 Three 브랜치 전략을 사용하여 Merge Request 승인을 통해 품질 관리를 강화함

브랜치 전략



Hybrid Cloud MSA 프로젝트 CICD 사례

L사의 MSA기반 멀티 배포 환경 CI/CD 자동화를 위한 최적화 된 Tool Chain 선정하고, 컨테이너/Cloud 환경 특성을 고려하여 잦은 배포에 대해 자동화 된 테스트 및 검증이 가능한 파이프라인 구축함



감사합니다