금융산업 핵심 역량 강화를 위한 코어 어플리케이션 현대화 전략

November 2023

KEARNEY

01 코어 어플리케이션 현대화의 의미와 필요성

금융 디지털 혁신과 코어 어플리케이션 현대화

코어 어플리케이션 정의

– 일상적인 금융 거래를 처리하고, 계정 및 기타 재무 기록에 대한 업데이트를 게시하는 백엔드 시스템을 의미한다. 일반적으로 금융상품 및 신용 처리 기능이 포함되며, 총계정원장 시스템 및 보고 도구에 대한 인터페이스를 포함한다.

코어 어플리케이션의 현대화

– 금융기관이 DT 혁신을 위해 코어 어플리케이션의 업무 서비스 및 프로세스와 인프라를 최신 기술과 방법론으로 업그레이드하고 변화시키는 과정을 의미한다.

코어 어플리케이션 현대화의 추진 동인

변화하는 고객과 시장의 요구에 대한 기민한 대응을 위해, 지금까지 채널 및 서비스 접점의 디지털 혁신에서 나아가, 코어 어플리케이션의 변화도 필요하다는 인식이 확산되고 있음

외부 연계 및 서비스 확장	금융 서비스에 대한 외부 핀테크/플랫폼사와의 협업 및 공동 서비스 개발 용이성
Biz 신속성/효율성 향상	신규 상품/서비스의 출시기간 단축과 非금융 컨버전스에 대한 Biz 니즈의 민첩한 대응
Cloud-First	Open innovation과 기술부채 해소 위한 기술 중립적(Agnostic) 개방형 구조 필요
고객 경험 개선	고객 경험 여정을 최우선 고려한 채널 서비스 구현 시 코어 시스템의 기술적 제약 해소
비용 효율화	시스템 트래픽 변동성에 따른 탄력적인 자원 및 인프라 활용

코어 시스템의 진화

코어 시스템은 기술 발전에 따라 진화 중이며, 최근 클라우드 기반의 기술 구조 전환이 글로벌 트렌드로 부상함

[~2000년대 중반] **폐쇄형 기술구조**

- 단말 기반의 단순한 상품을 운영하면서 24X365나 온라인 지급 결제와 같은 대고객 서비스의 안정성과 연속성을 중요시하던 시기이다.
- 대부분의 시스템이 메인 프레임 기반으로 구축되어 있으며, 프로그래밍(코딩) 중심의 기능이 개발되어 왔다. 또한, 특정 벤더의 하드웨어나 소프트웨어에 종속되어 있으며, 채널계와 대외계 인터페이스 중심으로 구축되어 있었다.

[~2010년대] **개방형 기술구조**

- 인터넷 모바일 뱅킹이라는 비대면 채널이 주요 채널로 자리매김하면서 핀테크 회사들이 등장, 기존과 다른 경쟁 구도가 생긴 시기이다.
- 유닉스 기반의 인프라 환경으로 전환을 시도하고, 코딩 중심의 기능 개발에서 더 나아가 상용 소프트웨어를 도입하려는 움직임이 점차 높아지게 되었다. 디지털 뱅킹 플랫폼을 통합 구축하는 것을 중심이 되었다.

[~2020년대] **클라우드 기술구조**

- 디지털 뱅킹에 대한 경쟁이 본격화되고 다양한 니즈가 발생하고 있는 시기이다.
- 클라우드 기반의 마이크로서비스 아키텍처(MSA)나 Low-code 코드 개발 툴과 같은 다양한 기술 요소들을 금융 시스템에 접목시키는 시도가 늘어나고 있다. 또한, Baas(Banking as a Service)와 같이 금융 상품이나 서비스들을 API 중심으로 대외 생태계에 오픈하고 이를 제휴하는 기술 구조의 움직임도 생기고 있다.

02 글로벌 선도 금융사들의 코어 시스템 현대화 추진 동향

글로벌 금융기관들의 코어 시스템 현대화 추진 목적

많은 글로벌 금융사들이 전사 차원의 디지털 혁신, 고객 서비스 향상, 신속한 상품/서비스 출시 목적으로, 코어시스템 현대화를 우선과제로 추진 중임

- 최근 가트너(Gartner) 조사에 따르면, 조사 대상 금융기관의 83%가 코어 시스템의 현대화를 추진했거나 현재 추진 중이라고 응답했다. 그리고 53%는 이러한 코어시스템의 현대화가 IT 추진 과제 중 우선순위가 높은 과제로 응답했다. 즉, 전사 차원에서 디지털 혁신의 목적으로 코어 시스템의 현대화를 이미 추진 중에 있다고 볼 수 있다.
- 조사 내용 중 코어 시스템 현대화를 추진하는 주요 동인을 살펴 보면 운영 효율화 수준 제고, 전행 차원의 디지털 혁신 지원, 대고객/파트너사 서비스 향상, 신상품/서비스 출시 기간 단축, Biz 생태계 연계 확장성 제공 등이 있다.

코어 시스템 현대화의 주요 트렌드

해외 사례를 통해 클라우드 네이티브 기술 기반의 현대화 추진이 이미 빠르게 확산되고 있음을 확인할 수 있음

Cloud-First 전략

- 기술 요소를 선정, 검토함에 있어서 클라우드 플랫폼을 우선적으로 검토하는 전략으로, Multi 클라우드 구성 등을 고려한다.
- JP Morgan Chase 의 경우, 모든 기존의 레거시 시스템들을 퍼블릭 클라우드로 옮기는 퍼블릭 클라우드 퍼스트 전략을 채택하여 실행하고 있다.
- Lloyds Banking Group은 GCP로의 마이그레이션을 진행 중이다.

디지털 뱅크 설립을 통한 신기술 검증

- 기존의 레거시 금융사들이 새로운 브랜딩 전략으로 New 디지털 뱅크를 설립하는 것을 의미한다.
- Standard Chartered 의 경우, 싱가포르에는 'Trust', 홍콩에는 'Mox'라는 브랜드로 디지털 뱅크를 설립하고 해당 디지털 뱅크들은 클라우드 네이티브 기술 요소의 시스템을 구현하여 운영 중이다.
- 이탈리아 최대 은행인 Intesa Sanpaolo는 'Isybank'라는 브랜드로, JP Morgan Chase 역시 'UK Digital bank'라는 브랜드로 디지털 뱅크를 운영하고 있다.

De-Coupled 아키텍처 지향

- 서비스 단위의 아키텍처 및 DB 분리를 추진하고 있으며, 마이크로서비스 아키텍처(MSA)나 Event-driven 아키텍처를 활용하여 기존의 레거시 시스템의 아키텍처를 변환시키고 있다.
- Santander는 'Gravity', Standard Chartered 는 'Atlas'라는 인하우스 코어 솔루션이 존재하며, 해당 솔루션들은 이러한 아키텍처 사상을 지향하여 설계되었다.

금융 솔루션 시장의 개편

- 클라우드 네이티브 솔루션을 제공하는 Player 중심으로 생태계가 개편되고 있다.
- 클라우드 네이티브 뱅킹 솔루션사는 'Thought Machine', '10x', 'Finxact' 등이 있다.

코어 어플리케이션 현대화 추진 방식

코어시스템 현대화 방식은 크게 2가지로 분류할 수 있으며, 글로벌 금융사들은 각각의 특성에 적합한 전환 방식을 선정하여 현대화를 진행 중임

1) Migration

- 기존의 레거시 코어를 뉴 코어로 바꾸는 것으로 Big Bang 방식과 Phased 방식으로 나뉜다.
- Big Bang 방식의 경우 레거시 코어를 뉴 코어로 일괄적으로 전환하기 때문에 기술적인 위험이 존재하고, 기술 경쟁력의 우위를 잃어버릴 가능성이 존재한다. 그러므로 최근에는 이 방식을 거의 활용하고 있지 않다.
- Phased 방식은 기존의 레거시를 현대화된 코어 시스템으로 전환할 때, 점진적으로 전환하는 방식이다. 그러므로 Big Bang 방식에 비해 상대적으로 위험도가 낮고, 전환 소요기간이 유연한 것이 특징이다. Lloyds Banking Group, JP Morgan Chase, Santander 등과 같은 주요 글로벌 은행이 이 방식을 채택해 현대화를 진행 중이다.

2) Greenfield

- 기존의 대형 금융사들이 디지털 은행을 새롭게 설립하면서 기존의 레거시 시스템을 바꾸지 않고 새로운 브랜드를 만드는 것을 뜻한다.
- 기존 은행과 새로운 디지털 은행이 공존하기 때문에 상대적으로 리스크가 적고 새로운 시스템을 특정 파일럿 상품/서비스 중심으로 구축하기 때문에 소요 기간도 상대적으로 적게 든다.

코어 어플리케이션 현대화 사례 ① Lloyds Banking Group

LBG는 그룹 차원의 디지털 전략 방향성 아래, 코어 뱅킹 시스템 현대화를 중심으로 하는 디지털 전략을 추진 중임

- LBG는 Technology Strategy 2.0 이라는 그룹 차원의 전략 방향성 아래 Grow, Focus, Change 3개 영역으로 분류해서 세부 디지털 전략들을 수립했다. 그 중 '코어 뱅킹 시스템 현대화 기반 상품 오퍼링 혁신'이라는 메인 전략을 진행 중에 있다.

- 코어 뱅킹 시스템의 현대화 전략의 방향성은 크게 3가지이다. 신규 코어 뱅킹 시스템 기반 트래픽 부하 완화 및 Resiliency를 확보하는 'Core Resilience', 기존 코어 뱅킹 40개 기능을 간소화하는 'CBS Simplification', 경량화된 신규 코어 뱅킹 시스템을 구축하는 'Lean Modern Core'이다. LBG는 오래된 기술 요소 기반의 뱅킹 시스템을 보유하고 있었다. 그러다 보니 변화에 대한 대응에 한계가 있고, 외부 기술의 발전 속도를 빠르게 따라가지 못하는 제약이 있었다. 이를 해결하기 위한 전략적 방향으로 2030년까지 점진적으로 전환하는 일정으로 진행 중에 있다.

코어 어플리케이션 현대화 사례 ② JP Morgan Chase

US 소매금융 CCB(Consumer & Community Banking) 부문에 대해, Deposits 2.0이라는 코어뱅킹 현대화 프로그램을 추진 중임

– JP Morgan Chase는 크게 2가지 트랙으로 진행 중이다. 영국의 경우 'UK Digital Bank'를 이미 설립하여 운영 중이며, 미국의 경우 기존의 소매금융 CCB 부문의 기존 시스템을 새로운 코어 시스템으로 전환하는 'Deposits 2.0' 프로그램을 진행하고 있다.
즉 Greenfield 방식과 Phased Migration 방식을 동시에 채택 추진하는 것이며, 미국 소매금융 migration의 경우 신규 대출 상품을 시작으로 예금 상품, 카드/대출/리워드 상품을 점진적으로 마이그레이션 할 예정이다.

코어 어플리케이션 현대화 사례 ③ Santander

Hybrid/Multi 클라우드 및 MSA 방식을 채택하여, 100% 클라우드 전환을 목표로 단계 별 추진 중임

– 기존 레거시의 클라우드 전환을 통해 다양한 효익을 이미 실현하거나 예상하고 있다. 신규 상품/서비스의 Time to Market 시간을 단축하고, 실시간으로 데이터를 분석하여 효율성을 높일 수 있다. 또한, 사용자가 데이터에 엑세스하는 속도를 향상시켜 고객 경험을 개선할 수 있고, IT 운영 비용을 절감할 수 있게 된다.

코어 어플리케이션 현대화 사례 ④ Intesa Sanpaolo

Isybank라는 디지털 은행 설립을 통해 Greenfield 방식을 채택해 구현 중이며, 이후 전행 고객 대상 점진적인 확대 추진 예정임

– 2024년까지 소규모/단순 금융 거래 리테일 고객을 대상으로 한 Isybank 설립하는 것을 목표로 하고 있다. 2024년 이후에는 고객/시장 범위를 전체 고객군 및 해외 시장으로 확대하여 기존 레거시 시스템의 migration도 병행 추진 예정이다.

03 코어 어플리케이션 현대화를 통한 변화와 효익

현대화 방식의 전환 유형

현대화 방식은 클라우드 전환 분류에 따라 구분할 수 있으며, 최근 글로벌 현대화 사례는 Rearchitect 유형에 해당함

Rehost

Rehost 방식은 상대적으로 전환 난이도가 낮은 편으로, 과거 국내 금융권에서 진행한 방식이다. 이 방식은 대부분 애플리케이션의 구조적인 큰 틀의 변화없이 인프라 중심의 변화가 이루어진다. 예를 들어, 메인 프레임을 유닉스 서버의 인프라로 교체하거나 유닉스 서버를 x86 리눅스 서버로 교체하는 것이 인프라 중심의 변화에 해당한다.

Refactor

비용 절감이나 개발 생산성을 위해 코어 프로그램의 코드와 DB 스키마 등을 최적화하여 변환하는 것으로 이 유형에 해당하는 국내 금융권 사례는 많지 않다.

Rearchitect

코어 시스템을 클라우드 기술과 아키텍처로 전환하여 개발하는 것으로 어플리케이션을 현대화(Rewrite)하는 것이 포함된다. 이 유형에는 최근 글로벌 대형 금융사들의 현대화 사례가 해당된다.

Replace

SaaS 솔루션을 도입해서 새로운 상용의 코어 솔루션으로 대체하는 방식으로 이 방식은 상대적으로 전환 효과 및 미래가치가 높다. 국내에서는 금융권의 규제로 인해 해당 방식을 사용하고 있지 않지만, 글로벌 중소형 금융사나 모바일 은행은 Replace 방식을 채택하고 있다.

코어 시스템의 현대화로 인한 변화

디지털 혁신 비즈니스 모델을 유연하게 지원할 수 있는 기술 아키텍처 설계가 필요하며, 단순 인프라의 전환이 아닌 기술/운영 관점의 근본적인 패러다임 Shift를 의미함

디지털 네이티브 은행의 Technology Stack (KEARNEY의 프레임워크)

Channels
디지털 채널(모바일/웹) 통한 고객의 금융 서비스 경험 제공

Multi-channels layer
고객 프로파일에 따른 적응형 UX1와 컨텐츠 제공 보장

Business layer
고객 라이프사이클에 따른 영업활동을 지원하기 위한 프로세스 로직 구현

Core
금융 상품의 프로세싱 기능 지원

Enterprise management
내부 시스템 연동 (재무, 회계/결산, IT, 구매/조달, 인사 등)

Analytics
전사 업무에 대한 실질적 인사이트 제공

Integration
외부 파트너 생태계의 연계 확장 지원

Security
사이버 위협으로부터의 보안, 예방, 보호 조치

Infrastructure
전사 플랫폼에 대한 클라우드 기반 솔루션 기술요소 제공

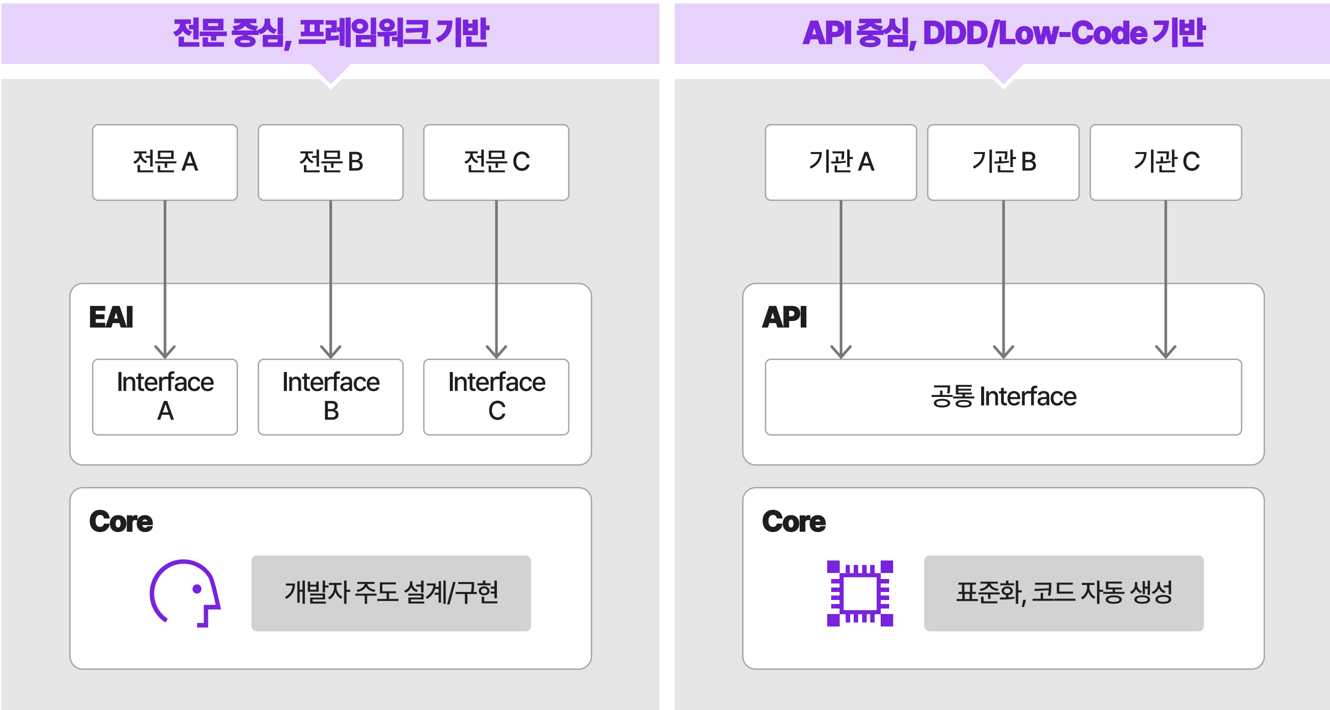
Source: Kearney

서비스 관점의 변화

비즈니스 도메인 기반 아키텍처 설계와 Low-code 개발 툴 활용으로 개발 효율성을 제고하고, 표준화된 API 방식으로 외부 서비스 연계의 확장성을 높임

- 기존의 레거시 시스템들은 대부분 전문 중심으로 EAI를 통해 외부 기관과 인터페이스를 연계하게 된다. 또한, 코어 시스템/프레임워크 개발 시, 개발자 주도로 설계 및 구현된다. 때문에 개발자 역량에 대한 의존도가 높아져 상품/서비스를 출시할 때, 많은 제약사항이 따르게 된다.

- 향후에는 API 중심, DDD(도메인 주도 설계)/Low-code 기반으로 변화할 것으로 보고 있다. 공통 인터페이스를 구현한 API 플랫폼을 두고 Open API에 따라서 대외 협력사나 기관과 빠르게 연결할 수 있는 환경을 구축할 것으로 예측하고 있다. 그렇게 되면, 개발자에 대한 의존도를 줄일 수 있으며, 개발에 대한 표준화 및 자동화를 이룰 수 있다.



- 서비스의 외부 노출/협업 시 대응 제약
- 개발자에 따른 상품 설계/구현 역량 상이

- Open API 등 미래 서비스 대응 용이
- 프로그램이 표준화된 방식으로 분석/설계

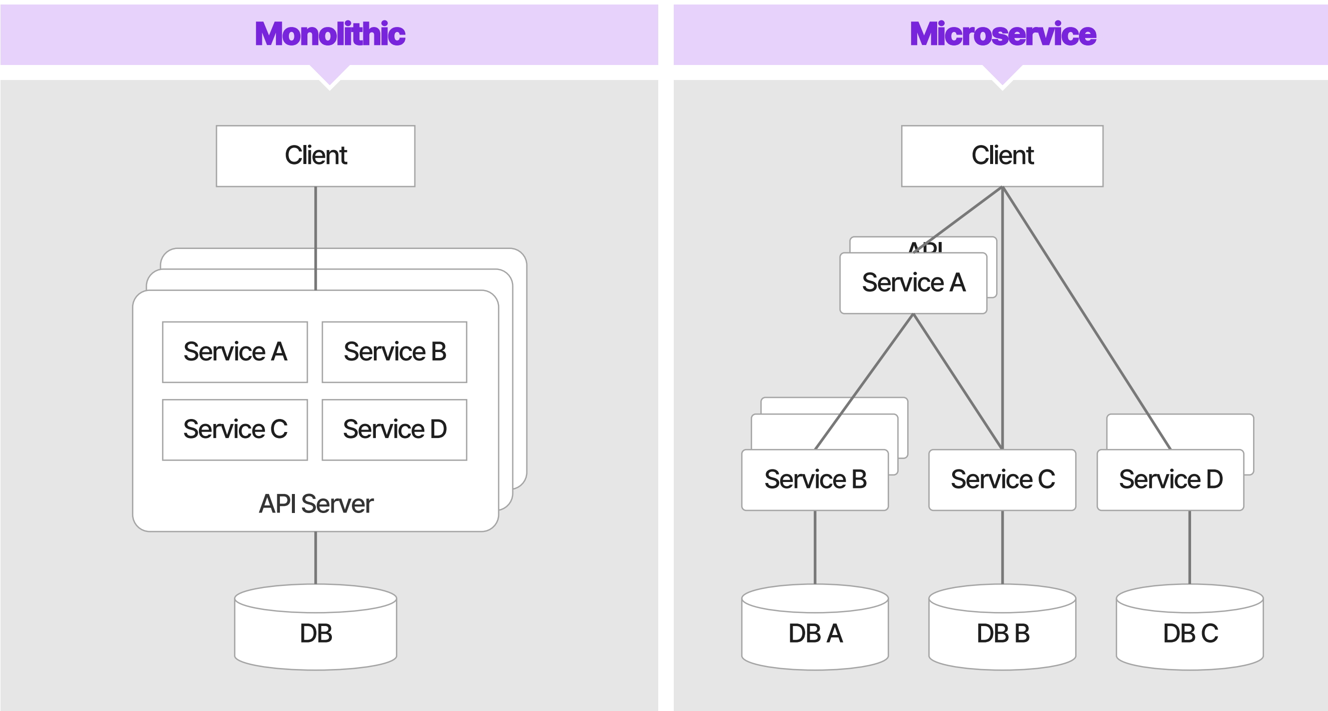
Source: Kearney Analysis

아키텍처 관점의 변화

Decoupled 아키텍처 구현을 통해 혁신 서비스의 Time-to-Market을 단축하고, Scale-out 통한 성능 향상 및 독립적 개선활동 기반 기술부채 경감 효과를 가져옴

– 기존의 전통적인 모놀리틱 아키텍처는 '안정성'이 가장 큰 장점이다. 이에 대고객 서비스를 제공하는 금융산업에 매우 적합한 환경이었다. 하지만, 어떤 특정 포인트에서 장애가 발생하거나 변화가 필요한 경우, 전체에 영향을 미칠 수 밖에 없는 구조를 갖고 있다. 또한, Scale-up 방식으로만 성능을 향상시킬 수 있기 때문에 기술적으로 고도화하기 어렵다는 이슈도 존재한다. 즉, 모놀리틱 아키텍처 환경이 과거에는 서비스의 안정성 측면에서 금융 산업에서 적합한 환경이었으나 디지털 혁신/변화가 빠르게 이루어지는 최근 상황에는 많은 제약이 존재해 마이크로서비스 아키텍처로 전환하는 시도를 진행 중이다.

– 마이크로서비스 아키텍처는 각 서비스들이 분산된 구조를 이루고 있다. 그렇기 때문에 개별 부분의 변화나 장애가 발생하더라도 전체에 영향을 주는 것을 최소화할 수 있다. 또한, 성능 향상 측면에서도 Scale-up 외 Scale-out을 통해서도 성능을 향상시킬 수 있으며, 채널계와의 변화 측면에서 멀티 채널 및 신규 미들웨어에 대한 대응이 용이하다는 장점이 있다.



– 부분의 변화가 전체에 영향을 미침
– 장애 발생 시 시스템 전체에 영향
– 성능 향상 목적의 리소스 Scaling 제약
– 채널계 변화에 따른 프로토콜 변화 대응 제약

– 각 서비스의 유기적 구성, 분산 구조
– 장애 발생 시 국지적인 영향
– 자유로운 범위의 Scale-out 적용 가능
– 멀티 채널 및 신규 미들웨어 대응 용이

Source: Kearney Analysis

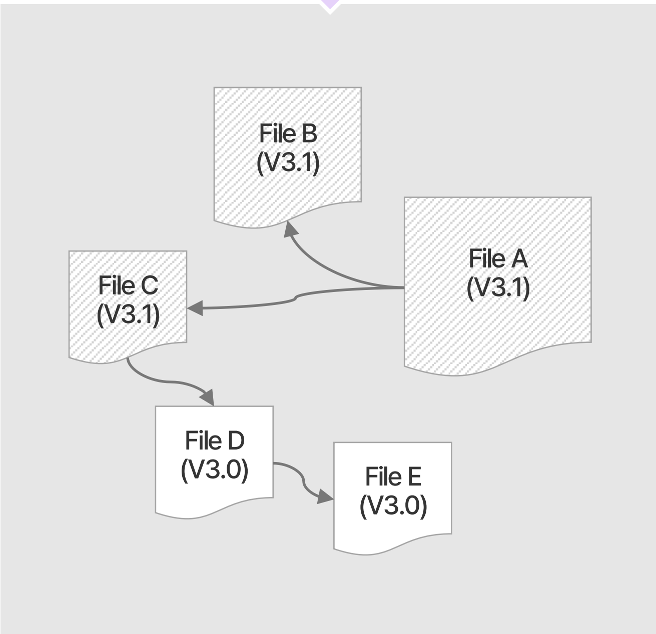
운영 관점의 변화

서비스 인스턴스 내 버전 별 이미지 관리를 통해 변경 영향도 파악이 용이하고, 장애 발생 시 Fail-over 및 Fail-back 관리 효율성을 높임

– 기존의 레거시 시스템들은 대부분 파일 단위로 버전을 관리하는 경우가 많아 버전 변경이 발생할 때마다 각 시스템에 미치는 영향도를 파악해야 하는 번거로움이 존재한다. 또한, fail-over나 fail-back 또한 관리가 어렵다는 특징이 있다.

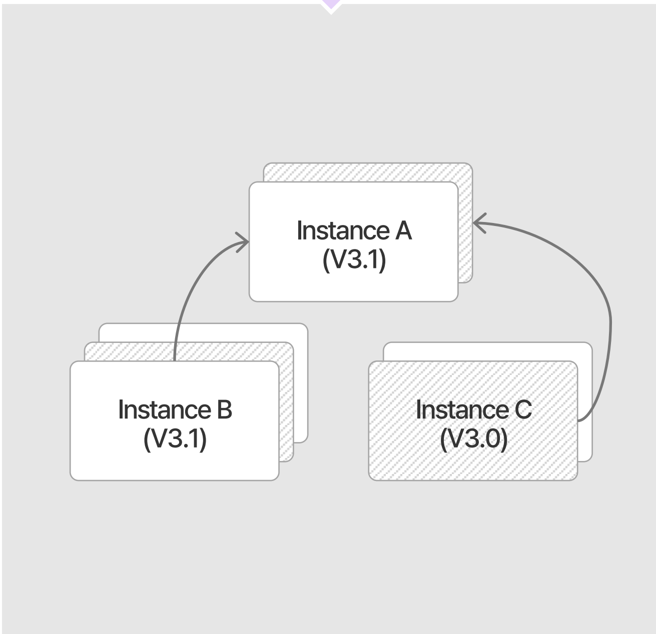
– 이에 향후 운영 관점에서 서비스 인스턴스 별로 버전을 관리하는 방식으로 변화가 필요하다. 마이크로서비스 아키텍처로 전환되고 서비스 인스턴스 별로 버전을 관리하게 되면 인스턴스 내 버전 별로 이미지화되어 관리할 수 있게 된다. 그렇게 되면, 기존 레거시 시스템에서 제약이 있었던 fail-over나 fail-back을 용이하게 관리할 수 있게 된다.

File 단위 버전 관리



– 버전 변경 발생 시 영향도 파악 제약
– Fail-over, Fail-back 관리 어려움

서비스 인스턴스 별 버전 관리



– 인스턴스 내 버전 별 이미지화되어 관리
– Fail-over, Fail-back 관리 용이

Source: Kearney Analysis

04 코어 어플리케이션 현대화 추진을 위한 주요 고려사항

주요 고려사항

국내 대형은행 사례를 통해 코어 시스템의 현대화 추진을 위한 기술적 고려사항을 식별함

기술요소 검토 및 선정	– 아키텍처 구성 Layer 별 Cloud-Native 기술요소 검토 – 3rd Party 솔루션 활용 가능성 검토 (Buy vs. Build)
어플리케이션 Deployment 아키텍처 구성	– Container 구조의 개발/배포 환경 구성 검토 – 자동 스케일링 지원, Scale-out 위한 서비스 분해 및 데이터 분리 검토
점진적 전환 시 Multi-Core 운영 방안 수립	– 검증 및 운영 전환의 단계 별 진행 검토 – 분산 데이터 간 정합성 확보 방안, Event-driven 아키텍처 검토
마이크로서비스 아키텍처 구성 방안 수립	– 도메인 주도 설계(DDD) 기반 서비스 모델링 방안 검토 – Layered Architecture 구성 방안 검토
DevOps 운영 방안 수립	– Agile 방법론 기반의 CI/CD 환경 도입 검토

구축: 코어엔진 및 Low-code 플랫폼 기반 구현

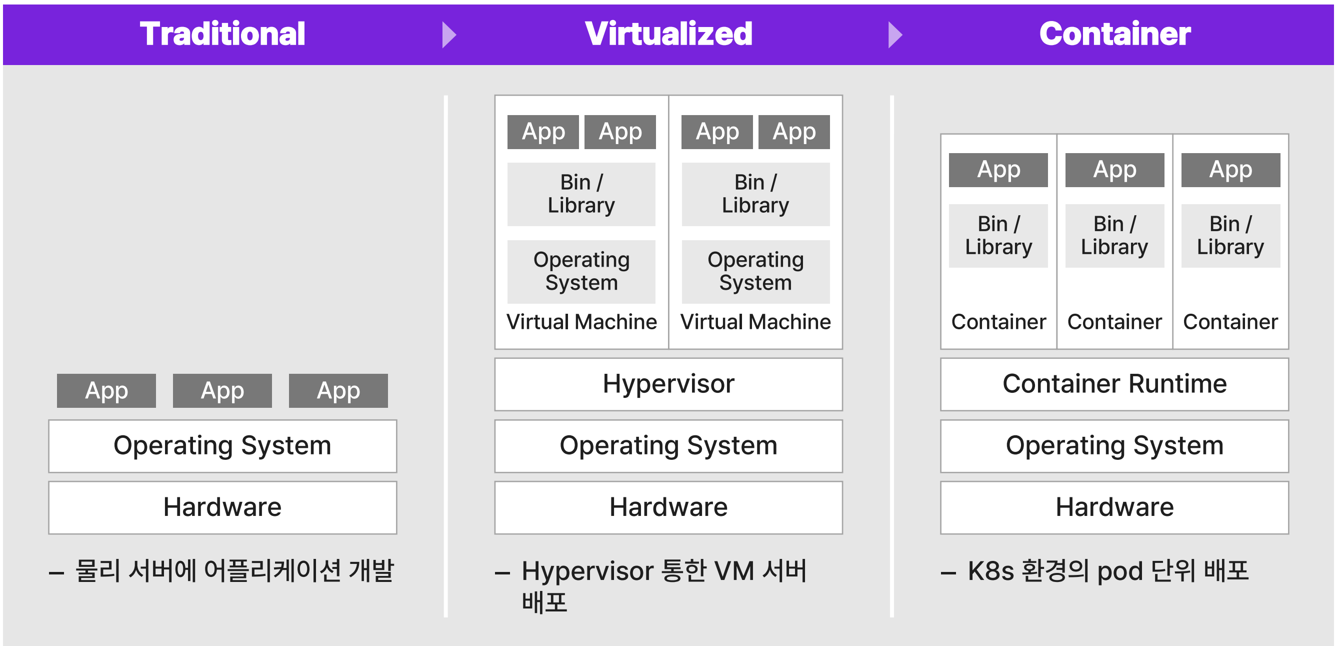
국내 금융 차세대 사업은 지금까지 In-house 개발 방식이 일반적이었으나, 최신 클라우드 기술과 솔루션 고려 시 코어엔진 등의 솔루션 도입 검토가 가능함

Full Package 방식	전사 기능/상품에 대한 패키지 제공하는 방식으로, 커스터마이징 난이도가 높다. 패키지 솔루션에 대한 종속성으로 자체적으로 기술력을 내재화하기 어렵다는 특징이 있다.
코어엔진 + In-house 개발 방식	3rd party 솔루션을 통한 경량화된 코어엔진과 금융사 특성에 맞는 상품/서비스 영역에 대해서는 인하우스로 개발하는 방식이다. 벤더에 대한 종속성은 풀패키지 방식에 비해 낮기 때문에 코어뱅킹 기술력의 내재화가 가능하다. 또한, 인하우스 개발을 병행하기 때문에 당행 특성에 맞는 커스터마이징이 가능하다.
In-house 개발 방식	모든 영역을 자체적으로 개발하기 때문에 패키지 솔루션에 대한 종속성은 없으나 넓은 개발 범위와 개발 복잡성으로 구축 난이도가 높다는 점이 특징이다.

배포: 어플리케이션 Deployment 아키텍처

기존의 서버 형태가 아닌, Docker나 K8s와 같은 환경에 pod 단위로 배포가 가능한 Container 구조로 개발환경을 구성함

– 과거의 전통적인 배포 환경은 하드웨어 물리 박스 위에 OS가 있고 그 위에 애플리케이션이 얹어지는 방식이었다. 최근 들어 하드웨어 위에 하이퍼바이저를 두고 VM 서버 상에서 애플리케이션을 배포하는 가상화 환경을 많이 구축하고 있다. 가상화 배포 환경에서 더 나아가 컨테이너 구조로 바꾸는 것을 시도하고 있다.



재구성: 마이크로서비스 아키텍처 구성

MSA 사상에 기반하여 아키텍처를 재구성하며, DDD 방법론 기반으로 서비스 도메인을 식별하여 Layered 아키텍처 구조 상에 구현함

– 마이크로서비스 아키텍처로 재구성하기 위해서는 도메인 주도 설계라는 DDD 방법론에 기반해서 서비스 도메인을 식별하고 구성하는 것이 먼저 선행되어야 한다. 다만, DDD 방법론으로 서비스 모델링을 할 때는 여러 가지 제약이 존재하는데 가장 중점적으로 해결되어야 하는 제약은 어떤 단위로 서비스를 나누고 모델링을 할 것인가에 대한 판단이다. 예를 들어, BIAN이라는 은행 산업의 표준화된 모델이 있는데 이를 참조 모델로 활용하여 은행에 특화된 상품/서비스에 맞게 변형하여 모델링하는 경우가 존재한다.

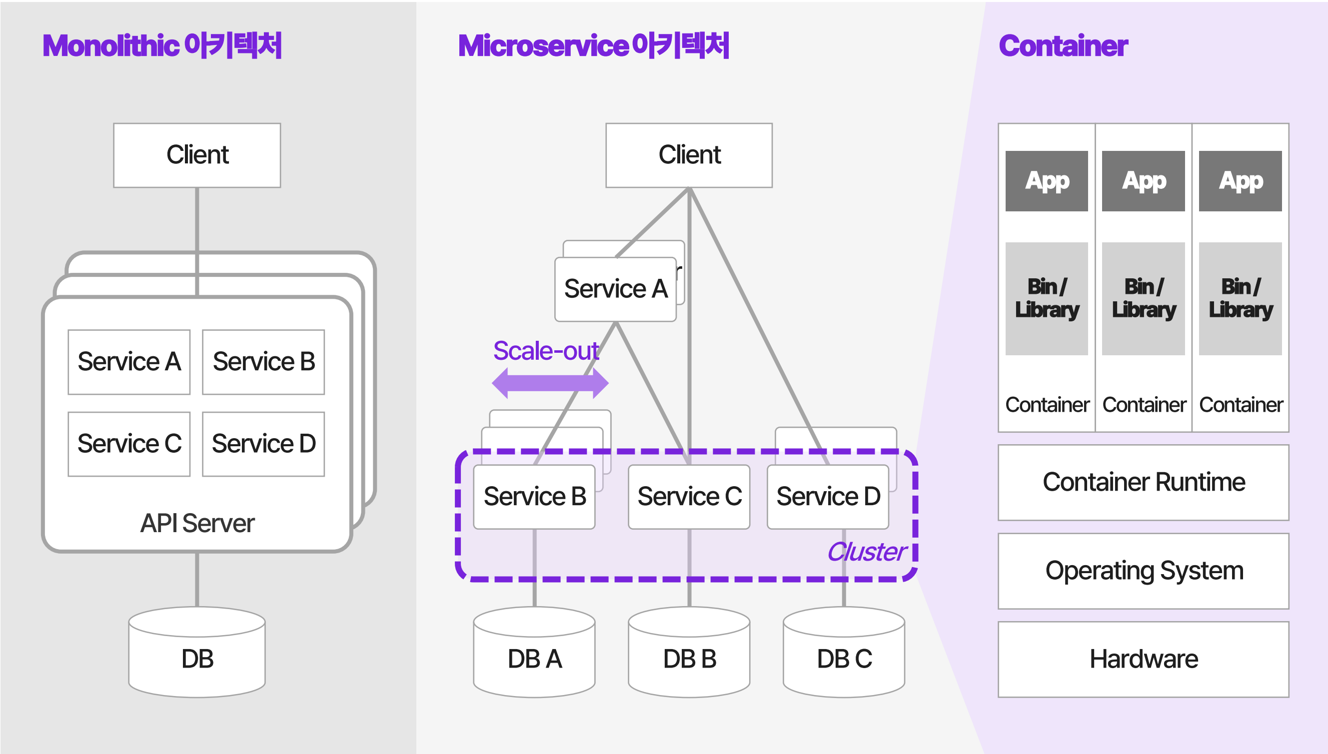
– 새로운 아키텍처를 구성할 때는 프로그램을 레이어별로 분리시키고 각 레이어별로 특정 역할을 수행하도록 한다. 이렇게 레이어를 분리하는 이유는 시스템 운영 과정에서 변경에 대한 영향을 최소화하기 위함이다.

Sources: Customer, Kearney Analysis

서비스 확장성: Container 기반 Scale-out

MSA 구성 시 Monolithic 아키텍처 대비, 컨테이너 적용을 통한 서비스 확장성(Scale-out), 시스템 경량화의 효익이 가능해짐

– 서비스별로 물리적으로 DB를 분리하는 것만이 MSA는 아니다. 물리적인 DB 분리 외에도 DB 스키마 분리와 같은 방식으로 아케텍처링할 수 있어 각 사에 맞는 적합한 방식으로 분리, 설계를 할 수 있다. 마이크로서비스 아키텍처로 바뀌면서 비즈니스나 트래픽, 데이터의 변화에 따라서 스케일 아웃을 통해 성능을 향상시킬 수 있다는 이점이 있다.



Source: Kearney

KEARNEY