클라우드

Do Not Use MSA - 마이크로서비스 아키텍처가 꼭 필요한가요?

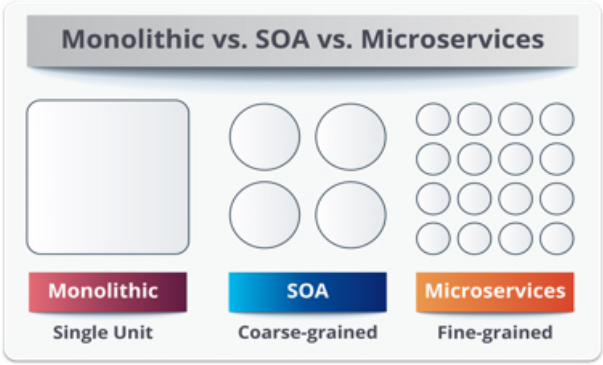
2020-03-13 박경원

1. 마이크로서비스 아키텍처 도입을 고민 중인 당신에게

IT 업계에서 마이크로서비스 아키텍처(Microservices Architecture, MSA)가 대세로 떠오르면서 엔터프라이즈 서비스 개발 프로젝트에 활발하게 채택되고 있습니다. 애플리케이션을 보다 빠르게 개발하고 성능을 지속적으로 높여 나갈 수 있다는 이유로 마이크로서비스 아키텍처가 각광받고 있는 것입니다. 시장조사업체 가트너의 아키텍처 트렌드를 살펴보면 마이크로서비스 아키텍처는 이미 성숙기에 접어들었음을 알 수 있습니다.

하지만 필자는 고객의 요구사항 대비 가용자원을 고려하여 적합한 아키텍처를 채용하는 것이 바람직하다고 생각합니다. 모든 엔터프라이즈 IT에 마이크로서비스 아키텍처를 적용하는 것은 오버 아키텍처링 또는 불필요할 수 있다고 생각하기에 이 글을 집필하게 되었습니다. 현재 기업과 IT업계 관계자들로부터 찬사를 받고 있는 마이크로서비스 아키텍처의 단점과 도입 시 고려사항을 중심으로 살펴보겠습니다.

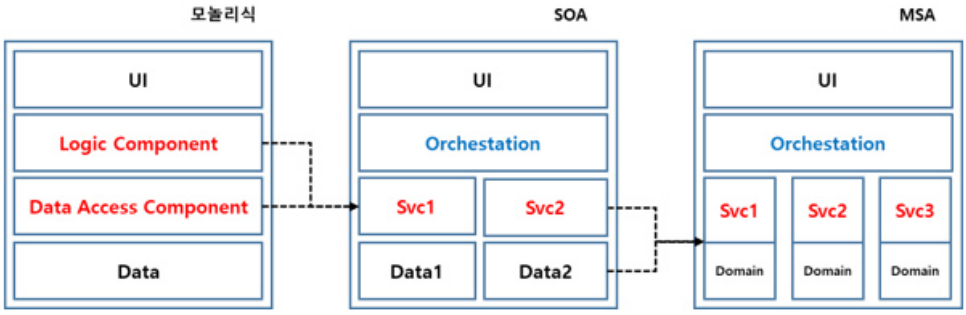
마이크로서비스 아키텍처를 언급할 때 모놀리식(Monolithic) 아키텍처와 비교를 많이 합니다. 이는 마이크로서비스 아키텍처를 돋보이게 하기 위한 방법으로 보이며 실제 둘 사이에는 여러 아키텍처들이 존재하고 있습니다. 또한 새로운 시스템을 설계할 때 모놀리식과 마이크로서비스, 둘 중 하나만을 선택하는 것도 일반적이지 않습니다. 의구심이 든다면 마이크로서비스 아키텍처가 등장한 2010년대 이전에 구축된 시스템들이 과연 모두 모놀리식을 채택했는지를 확인해 보면 될 것입니다.



[그림 1] 모놀리식 vs. SOA vs. MSA 서비스 분리 비교[1]

위의 [그림 1]을 살펴보면, 모놀리식 아키텍처 대비 서비스지향 아키텍처(Service Oriented Architecture, SOA)는 하나의 구성으로 동작하던 서비스를 단순히 몇 개의 서비스로 나눈 것이고, 마이크로서비스 아키텍처는 더 많은 서비스로 나눈 것으로 보여집니다. 마이크로서비스가 지향하는 바가 말 그대로 일련의 서비스를 더욱 잘게 나누는 분리화라고 생각한다면 이는 절반의 이해라 할 수 있습니다.

모놀리식 아키텍처가 UI, 비즈니스 로직 컴포넌트, 데이터관리 컴포넌트, 데이터들이 하나의 공간에서 밀접하게 연결되어 있다면 서비스지향 아키텍처는 공통 또는 특정 결과를 처리하기 위해 비즈니스로직 컴포넌트와 데이터관리 컴포넌트를 하나의 단일 서비스화하여 반복적인 비즈니스 동작들을 논리적이며 독립적인 형태로 구분지어 분리해 놓은 것으로 보면 됩니다. 마이크로서비스는 위의 서비스들을 더 잘게 나누기 위해 특정 도메인 바운더리, 즉 영역을 정의하고 해당 영역 안에서 필수불가결한 비즈니스 로직을 처리할 수 있도록 더 작은 단위로 서비스화합니다. 서비스지향 아키텍처에서 동일한 역할, 즉 중복된 비즈니스 로직을 제거하고 단일화했던 부분들이 다시 해당 서비스로 중복되어 재사용될 수 있으며 서비스를 나누는 기준이 하나의 도메인 영역을 중심으로 재편되는 것을 알 수 있습니다.



[그림 2] 모놀리식 vs SOA vs MSA 서비스 분리 비교[2]

위의 [그림 2]는 모놀리식 아키텍처에서 서비스지향 아키텍처로, 또는 서비스지향 아키텍처에서 마이크로서비스 아키텍처로 서비스를 정의하고 나누는 기준이 단편적으로 [그림 1]과는 다름을 보여 주고 있습니다. 이는 데이터를 바라보는 시각의 변화에 따른 당연한 수순처럼 보일 수 있습니다.
기존에 기업에서 관리하던 자원들과는 달리, 소비자들이 생산, 수정, 재생산 과정을 거쳐 쏟아내는 데이터는 정형화되기 힘든 비논리적인 데이터 집합이 되었습니다. 이를 관계형 데이터베이스로 관리하다 보니 한계에 직면하게 되었고 그 대안으로 2000년 초반부터 NoSQL이 도입되기 시작했습니다. 서비스지향 아키텍처가 만연해 있는 현시점에는 도메인의 성격과 크기, 목적에 따라 관계형 데이타베이스와 NoSQL을 채택하고 있습니다. 또한 관리해야 할 데이터 볼륨이 예측 불가능한 크기로 커질수록 MPP(Massively Parallel Processing) 데이터베이스, 일종의 데이터 웨어하우스 아키텍처로 확장되고 있습니다. 다만, 현재 기업들이 추진 중인 엔터프라이즈 서비스가 데이터 웨어하우스를 도입해야 하는지에 대해서는 고민해 볼 필요가 있습니다.

2. 마이크로서비스 아키텍처로 가는 길

[표 1]은 마이크로서비스 아키텍처의 구현 단계를 정의하고 있습니다.

	Level0 **Traditional**	**Level1** **Basic**	**Level2** **Intermediate**	**Level3** **Advanced**

	Level0 Traditional	Level1 Basic	Level2 Intermediate	Level3 Advanced
Application	Monolithic	Service Oriented Integrations	Service Oriented Applications	API Centric
Database	One Size Fit All Enterprise DB	Enterprise DB + No SQLs and Light databases	Polyglot,DBaaS	Matured Data Lake /Near Realtime Analytics
Infrastructure	Physical Machines	Virtualization	Cloud	Containers
Monitoring	Infrastructure	App & Infra Monitoring	APMs	APM & Central Log Management
Process	Waterfall	Agile and CI	CI & CD	DevOps

[표 1] A Microservices Maturity Model, from “Spring 5.0 Microservices,” 2nd Edition”by Rajesh R V (O’Reilly)

위 표를 보면, 몇 가지 의문사항이 들 수 있는데, Level 0 Traditional에서 인프라만 컨테이너향으로 변환한다면, 또는 개발 프로세스만 데브옵스(DevOps)로 간다면 모놀리식 아키텍처와 마이크로서비스 아키텍처 중 어느 것에 가깝다고 할 수 있을까요? 또는 Level 1, Level 2에서도 위와 같이 적용한다면 서비스지향 아키텍처와 마이크로서비스 아키텍처 중 무엇으로 부르는 게 맞을까요?

이러한 질문을 던지는 것은 끊임없이 발전 중인 IT서비스 트렌드를 역행하고자 하는 것은 아닙니다. 단지 현재 운영 중인 서비스를 개선하거나 새로운 서비스를 기획할 때, 기존 레거시시스템 및 데이터 연계, 현재 시점의 요구사항 그리고 예측 가능범위 내의 미래 변화까지 대응할 수 있도록 아키텍처를 비롯한 프로세스, 인프라, IT 환경에 대한 보다 폭넓고 수준높은 이해가 필요함을 시사하기 위해서입니다.

이어서 마이크로서비스 아키텍처를 도입할 경우 필수불가결하게 따라오는 쿠버네티스(Kubenetes)와 데브옵스에 대해서 논해 보겠습니다. 쿠버네티스는 2014년 구글에서 Go라는 언어로 개발하였으며 컨테이너화된 애플리케이션을 편리하게 배포 가능하고 오토스케일링, 스케줄링, 서비스 디스커버리, 로드밸런싱, 리소스 관리 등을 지원하는 컨테이너 오케스트레이션 플랫폼입니다. 기존의 베어메탈, 가상머신에서 애플리케이션을 배포하는 것과는 여러 면에서 차이점이 있는데 컨테이너향에서 가지는 장점은 분명하게 존재합니다. 간략히 비교해 보면 [표 2]와 같습니다.

구분	가상머신	컨테이너
Weight	중량화	경량화
Performance	설치된 머신에 못 미치는 성능	설치된 머신과 동일한 성능
OS	각각의 OS 위에 동작 가능	단일 호스트 OS 위에서만 동작
Virtualization	하드웨어 레벨 가상화	OS 레벨 가상화
Startup Time	수분의 시작 시간 소요	수초의 시작 시간 소요
Isolation	완전한 격리	프로세스 레벨의 격리
Security	높은 보안성	낮은 보안성
Memory	일정 수준의 메모리 자원 요구	보다 작은 메모리 자원 요구

[표 2] 가상머신과 컨테이너 비교

컨테이너향이 성능면이나 배포 시 걸리는 시간 등에서 우위를 보입니다. 가상머신은 단일머신이 주는 완전격리로 인한 보안 측면과 가상머신이 설치된 각각의 OS에서 동작 가능한 장점을 가지고 있습니다. 많은 서비스들이 컨테이너향으로 전환하고 있지만 데이터센터와 공용클라우드 환경에서는 가상머신 사용이 더욱 확산되고 있습니다. 또한 두 가상 기법은 상호부족한 점들을 채워나가며 발전하고 있습니다. 가상머신은 마이크로 가상머신으로 진화하면서 경량화와 기동시간을 줄이고 있으며 컨테이너 또한 격리성을 강화해 보안성을 높이고 있습니다.

그러면 쿠버네티스를 사용하는데 있어 해당 서비스를 어느 환경에서 이용하는 것이 효과적인지를 고민해 볼 필요가 있습니다. 분리된 서비스의 구성이 단순하며 나눠진 서비스들이 동작하는 OS 환경이 제각각이라면 쿠버네티스 환경에서 서비스들을 배포하는 것이 오히려 독이 될 수도 있을 것이기 때문입니다.

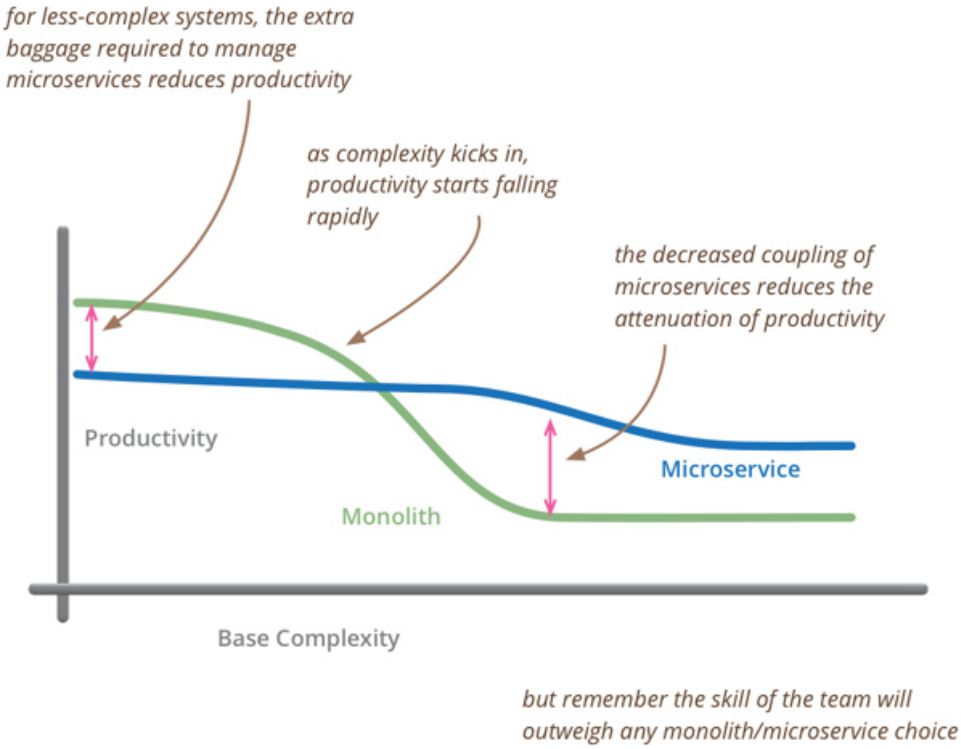
마이크로서비스 아키텍처를 언급할 때 '데브옵스(DevOps)'라는 용어를 빠뜨릴 수 없습니다. 서비스 단위가 소형화되면서 업데이트와 업그레이드가 상대적으로 용이해졌으며 이는 릴리즈 주기를 단축하는데 큰 이점이 되었습니다. 이를 보다 효과적으로 프로세스화하기 위해 데브옵스라는 개발 프로세스를 도입하는 기업이 많아졌습니다. 이름 그대로 개발(Development)과 운영(Operations)을 나누지 않고, '개발 + 테스트 + 배포 + 운영'을 동시에 할 수 있는 IT 환경 또는 문화가 등장하게 된 배경입니다.

개발 관점에서 마이크로서비스 아키텍처를 처음 접하게 되면 도메인 바운더리로 나눠진 서비스에 대해 더 많은 권한을 가질 수 있게 되어 환호할 수도 있지만 복잡도가 높아질수록 서비스에 대한 책임이 더 늘어나게 되는 것을 경험하게 됩니다. 여러 서비스들 간의 트랜잭션 처리는 이미 이슈가 아닌, 당연히 해결되어야 할 덕목으로 자리잡았으며 대표적인 해결 방안으로 Two-Phase Locking (2PL), Saga, Try-Later 같은 서비스 구성 패턴 등이 있습니다. 또한 서비스 간 통신의 홉(Hop)이 늘어날수록 응답시간 지연에 대한 처리 정책이 필요하며 요청에 대응하는 방식으로 동기 및 비동기, 모두를 고민해야 하는 상황이 발생하는 것입니다.

운영 관점에 보면 마이크로서비스 아키텍처를 통해 중앙에서 서비스들을 관리하거나 조율하지 않도록 설계했지만 여러 서비스들이 공통적으로 영향을 받거나 적용 처리가 필요한 사항들이 나타나게 됩니다. 아무래도 서비스의 바운더리를 구분지을 때 세분화한 영향도 있지만 요구사항이 바뀜에 따라 분리된 서비스에도 공통의 변경사항이 생기는 등 서비스를 효율적으로 운영하기 위한 요구사항이 끊임없이 발생하게 되는 것입니다. 참고로 쿠버네티스 환경에서는 Configmap을 통해서, 가상머신 환경에서는 Spring Cloud Config와 같은 서비스가 대안이 될 수 있습니다.

3. 당신이 고려해야 할 것은?

디자인 패턴으로 유명한 마틴 파울러는 “Don't even consider microservices unless you have a system that's too complex to manage as a monolith.”라고 말했습니다. 즉, "모놀리식으로 관리하기에 특별히 복잡한 시스템을 운영할 상황이 아니면 마이크로서비스는 고려할 필요조차 없다"고 피력했습니다. IT업계에 만연해 있는 마이크로서비스 아키텍처에 대한 무조건적인 찬사가 아닌, 필자가 생각하는 바와 어느 정도 일맥상통하는 바가 있어 소개해 드렸습니다.



[그림 3] 시스템 복잡도와 아키텍처

시스템 복잡도 단계에 따라 아키텍처 선택 시 개발 생산성에 크게 영향을 받을 수 있습니다. 특히 SW 엔지니어에게는 개발, 운영에 관한 다양한 스킬셋과 해결 방안뿐 아니라 단일 서비스가 아닌, 전체 동작에 대한 이해까지 요구될 수 있기 때문에 더욱 신중을 기하여 아키텍처를 선정해야 합니다.

지금은 각광받는 오늘의 기술이 순식간에 수명을 다해 어제의 기술이 되어버리는 시대입니다. 수도 없이 쏟아지는 트렌디한 기술 속에서 우리는 중심을 잃지 말고 진정으로 좋은 아키텍처, 좋은 기술이 무엇인지를 냉철하게 고민할 필요가 있습니다. 고객의 요구사항을 정확하게 이해하고, 해당 기술 스펙을 꿰뚫어 보며, 개발팀의 기술셋과 프로세스를 면밀히 분석하여 최적의 아키텍처와 기술을 선택하는 것이 IT 전문가가 갖추어야 할 덕목 중 하나라고 생각합니다.

마지막으로 마이크로서비스 기반 애플리케이션을 도입하거나 또는 전향을 고민하고 있다면 최소한 아래 사항을 고려해 볼 필요가 있습니다.

* 비용: 특정 서비스 아키텍처를 도입할 경우 비용을 얼마나 절감할 수 있는가?
* 개발 생산성: 마이크로서비스를 요구할 만큼 시스템 복잡도가 높은가? 또는 복잡도를 지나치게 높인 마이크로서비스가 생산성을 저해하고 있지는 않은가?
* 운영: 개발팀에게 개발과 운영을 동시에 요구할 만큼 인프라가 준비되어 있는가?

본 아티클이 마이크로서비스 기반 애플리케이션을 고민하고 있는 독자에게 간략하게 나마 전반적인 흐름을 이해하는데 도움이 될 수 있기를 기대합니다. 더 나아가 좋은 기술을 활용하여 좋은 서비스를 개발한다는 것에 대해 폭넓은 관점으로 고찰해 볼 필요가 있다는 화두를 던지고자 합니다.

 # References

[1] https://www.gartner.com/en/documents/3886164/hype-cycle-for-application-architecture-2018

[2] https://cazton.com/consulting/enterprise/software-architecture
[3] https://thenewstack.io/guide-for-2019-what-to-consider-about-vms-and-kubernetes/
[4] https://martinfowler.com/bliki/MicroservicePremium.html
[5] https://www.backblaze.com/blog/vm-vs-containers/

#MSA #MICROSERVICES #마이크로서비스 #마이크로서비스아키텍처 #아키텍처 #모놀로식 #DEVOPS #컨테이너

박경원 클라우드 전문가

에스코어㈜ 소프트웨어사업부 개발플랫폼그룹

에스코어 개발플랫폼그룹에서 DP Catalog 연구 개발을 담당하고 있습니다. 주요 연구 및 관심 분야는 Enterprise Services 관련 클라우드 플랫폼, 마이크로서비스 아키텍처(Microservices Architecture, MSA)입니다.