MSA

(Micro Service Architecture)

매시업 9기 백엔드 김다롬

목차

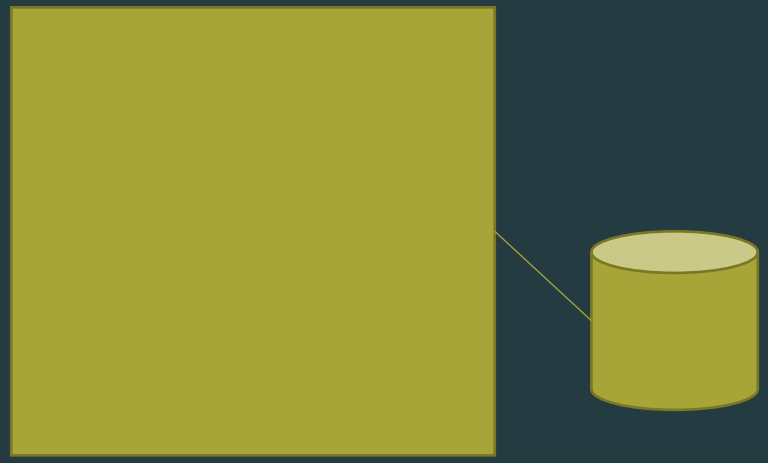
MSA 란?

- "하나의 큰 어플리케이션을 여러개의 작은 어플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍쳐“

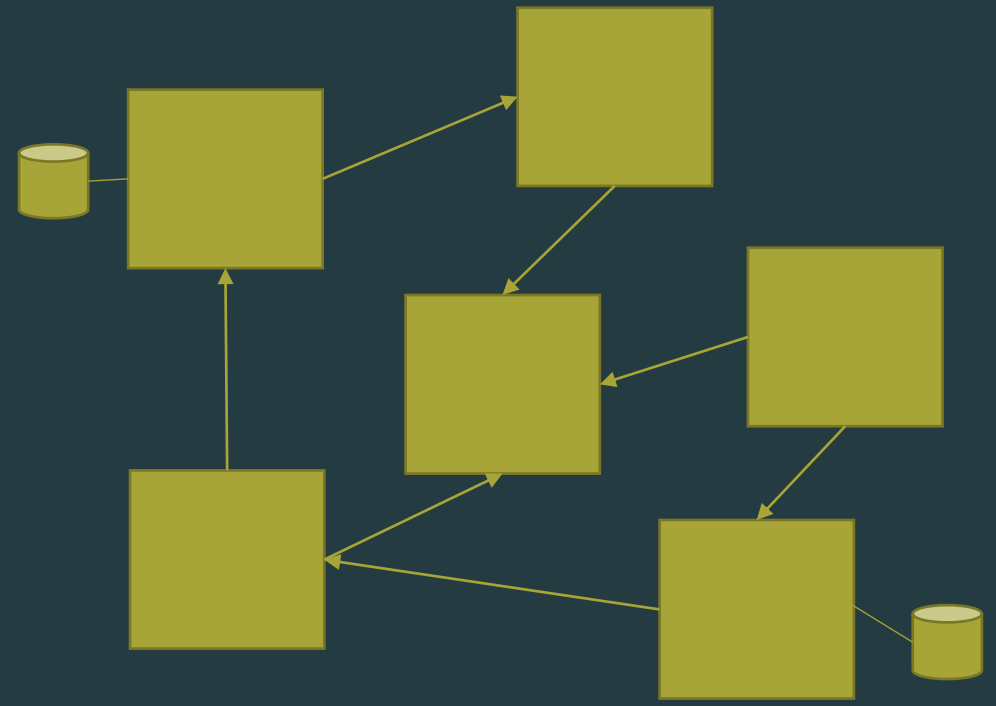
- 단일 애플리케이션 즉, 모노리스(Monolith) 애플리케이션을 작은 서비스 단위로 개발하는 방법을 MSA(Microservices Architecture)라고 한다.

MSA를 쓰는 회사들

MSA 란?



모놀리식 아키텍처



마이크로 서비스 아키텍처

MSA의 등장배경 - Monolithic Architecture

- Monolithic Architecture란, 소프트웨어의 모든 구성요소가 한 프로젝트에 통합되어있는 형태입니다.
  - 하나의 서비스 또는 어플리케이션이 하나의 거대한 아키텍처를 가짐

- 아직까지는 많은 소프트웨어가 Monolithic 형태로 구현되어 있고, 소규모 프로젝트에는 Monolithic Architecture가 훨씬 합리적
  - 간단한 Architecture이고, 유지보수가 용이하기 때문

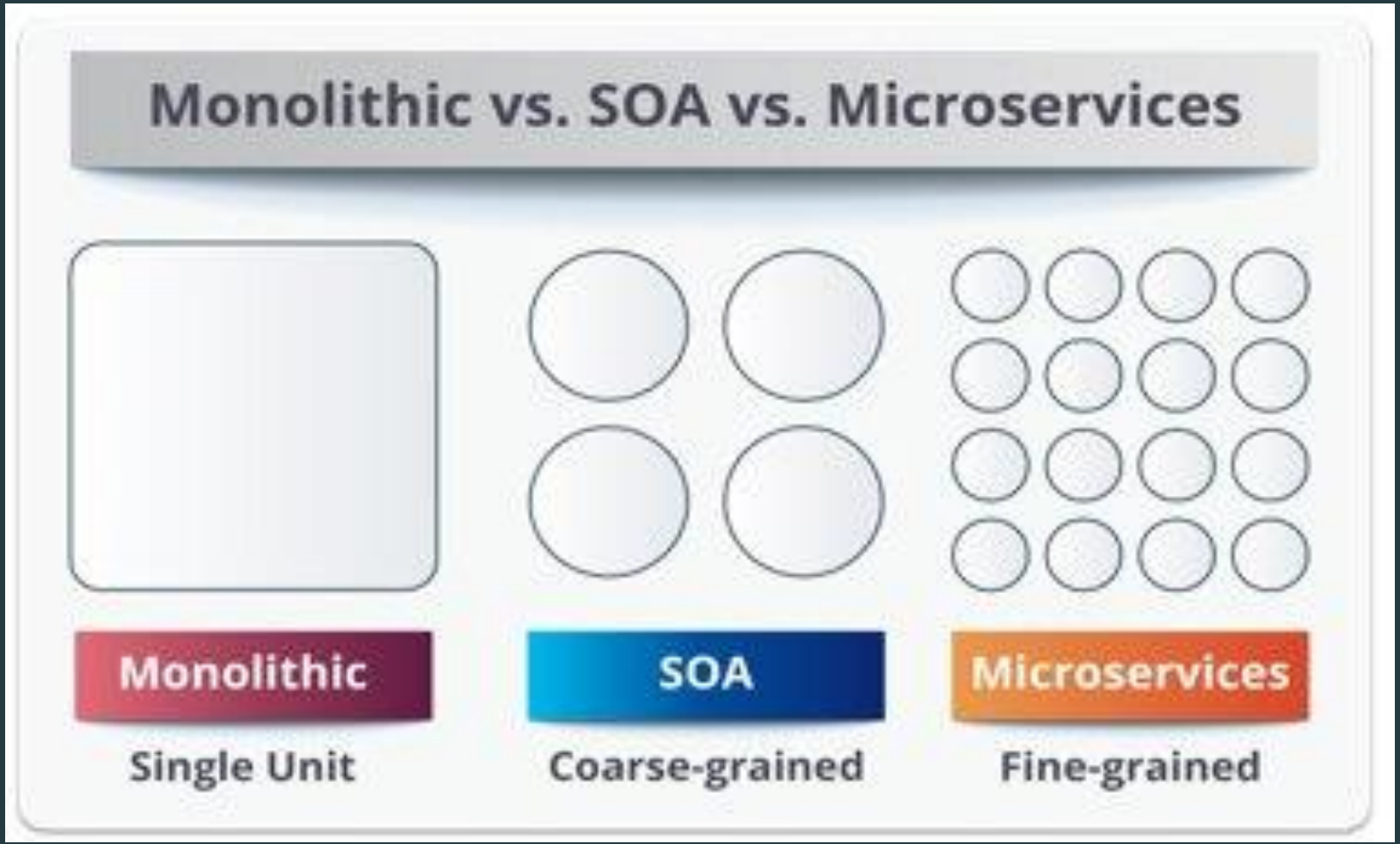
MSA의 등장배경 - Monolithic Architecture

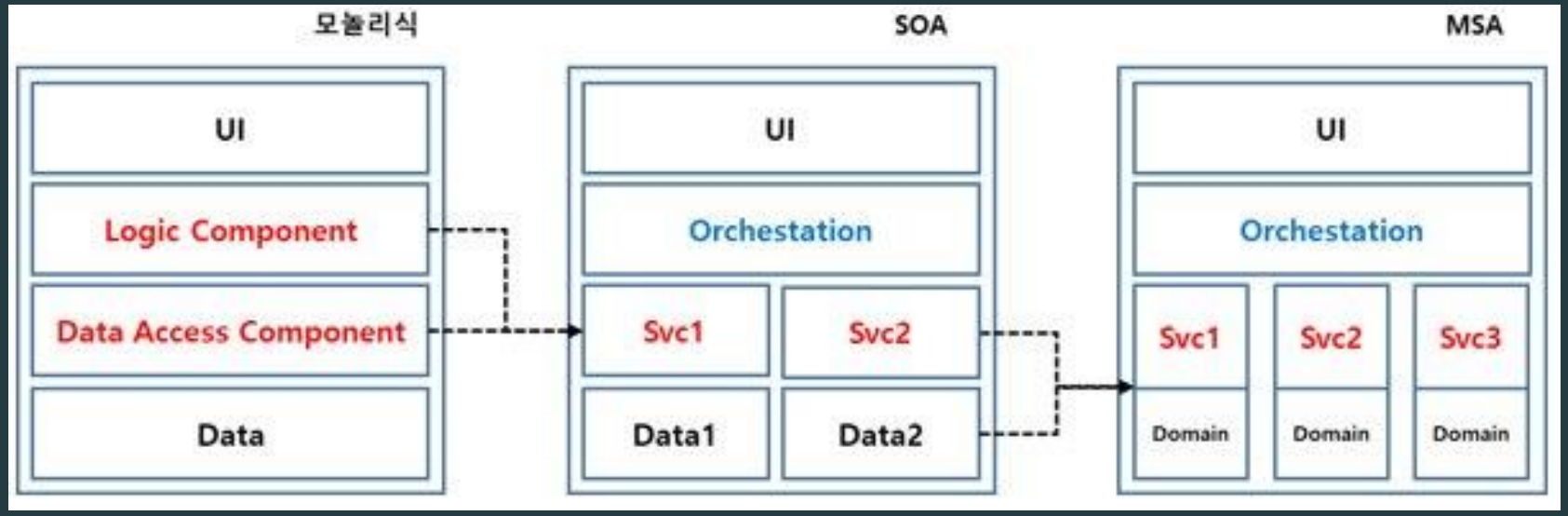
- 하지만 일정 규모 이상의 서비스, 혹은 수백명의 개발자가 투입되는 프로젝트에서 Monolithic Architecture은 뚜렷한 한계를 보인다.
  - 서비스/프로젝트가 커지면 커질수록, 영향도 파악 및 전체 시스템 구조의 파악에 어려움
  - 빌드 시간 및 테스트시간, 그리고 배포시간이 기하급수적으로 늘어남
  - 서비스를 부분적으로 scale-out하기가 힘듦
  - 부분의 장애가 전체 서비스의 장애로 이어지는 경우가 발생

MSA의 등장배경 - Micro Service의 정의

"스스로 돌아갈 수 있는 작은 서비스, 독립적 배포 가능"

- 각각의 서비스는 그 크기가 작을 뿐, 서비스 자체는 하나의 모놀리틱 아키텍쳐와 유사한 구조를 가짐
- 각각의 서비스는 독립적으로 배포가 가능해야함.
- 각각의 서비스는 다른 서비스에 대한 의존성이 최소화 되어야함
- 각 서비스는 개별 프로세스로 구동 되며, REST와 같은 가벼운 방식으로 통신되어야 함.


Monolithic vs. SOA vs. Microservices
Monolithic
SOA
Microservices
Single Unit
Coarse-grained
Fine-grained



- 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 놓았다.
- 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 있다.

특징

1. 각각 자체 프로세스에서 실행
2. 프로세스간 통신을 HTTP기반 API형식으로 통신
3. 비즈니스 기능을 중심으로 구축
4. 완전 자동화된 배포머신을 통해 독립배포
5. 중앙집중식 관리는 최소화
6. 다른 프로그래밍 언어로 개발 될 수 있음
7. 데이터 저장 기술을 사용할 수 있음

장점

전체 vs 부분

1. 장애 격리와 복구가 쉽다.
2. 비용 효율적으로 증설이 가능하다.
3. 서비스 개선 속도가 증가한다. 빠른 배포가 가능하다.
4. 생산성 향상이 될 수 있다. 코드양이 적어 쉽게 수정 가능하다.
5. 신기술 도입이 쉽다.
6. Polyglot을 적용할 수 있다.
    1. 서비스에 최적화된 개발 언어와 데이터베이스를 선택하기 쉽다.

단점

Monolithic Architecture은 단순한 아키텍쳐인데 비해
MSA는 보다 복잡한 아키텍쳐로, 전체 서비스가 커짐에 따라
그 복잡도가 기하급수적으로 늘어날 수 있습니다.

- 성능
  - 서비스 간 호출 시 API를 사용하기 때문에, 통신 비용이나, Latency(지연시간)가 그만큼 늘어나게 된다.
- 테스트 / 트랜잭션
  - 서비스가 분리되어 있기 때문에 테스트와 트랜잭션의 복잡도가 증가하고, 많은 자원을 필요
- 데이터 관리
  - 데이터가 여러 서비스에 걸쳐 분산되기 때문에 한번에 조회하기 어렵고, 데이터의 정합성 또한 관리하기 어렵다.

정합성 : 데이터가 서로 모순없이 일치해야함

고려해야 할 것

1. MSA에 적합한 서비스인가?
    1. 빠르고 잦은 배포를 필요로 하는가?
    2. 성능에 민감한 서비스인가?
    3. 분산 트랜잭션이 있는 서비스인가?
2. 데이터의 중복성을 허용해야 함
3. 배포 및 릴리즈를 자동화해야 함. 자동화 되지 않으면 운영에 굉장한 부담이 발생할 수 있다.
4. MSA 서비스에 맞는 팀이 운영되어야 함.

고려해야 할 것

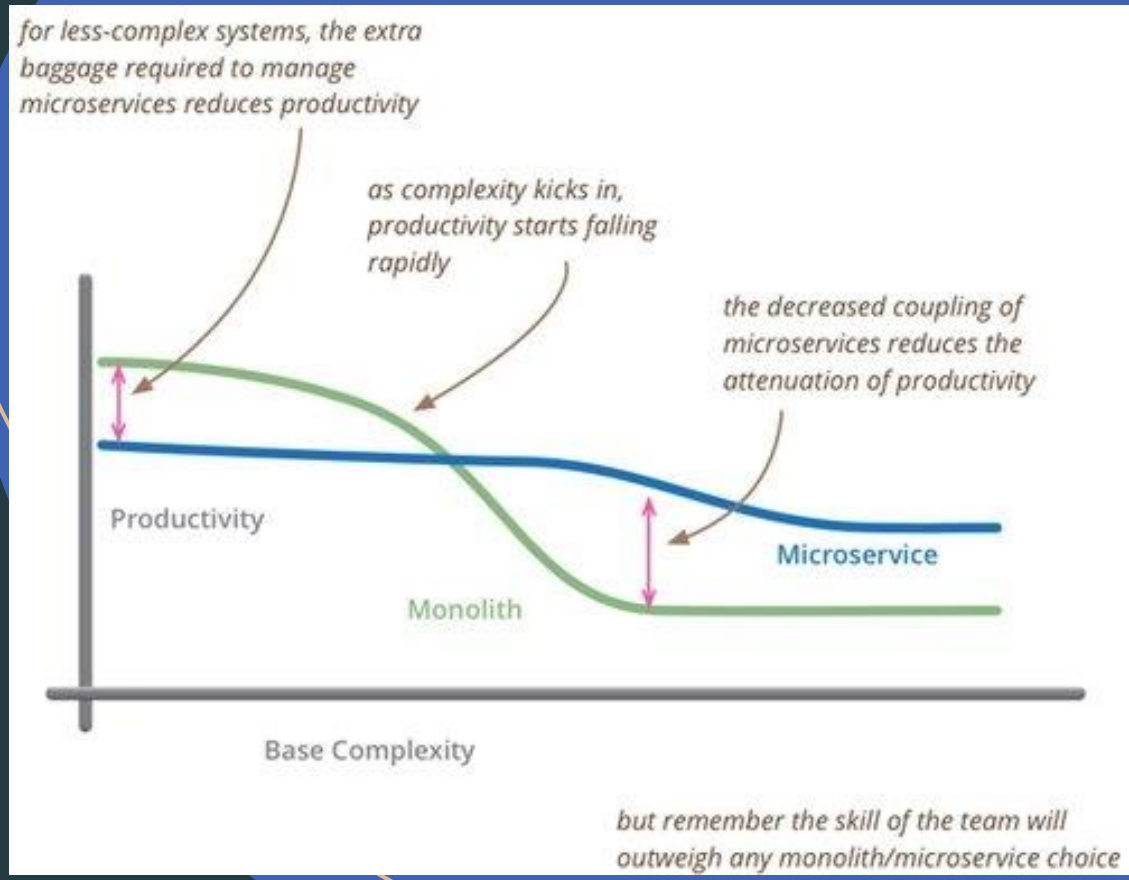
- 비용
  - 특정 서비스 아키텍처를 도입할 경우 **비용을 얼마나 절감**할 수 있는가?
- 개발 생산성
  - 마이크로서비스를 요구할 만큼 시스템 **복잡도가 높은가**? 또는 복잡도를 지나치게 높인 마이크로서비스가 생산성을 저해하고 있지는 않은가?
- 운영
  - 개발팀에게 개발과 운영을 동시에 요구할 만큼 **인프라가 준비**되어 있는가?

정리

- MSA란?
  - 규모가 큰 기존의 모놀리식 아키텍처의 한계를 극복하기 위해 나타났다.
    - 영향도 파악 및 전체 시스템 구조의 파악에 어려움
    - 빌드 시간 및 테스트시간, 그리고 배포시간이 기하급수적으로 늘어남
    - 서비스를 부분적으로 scale-out하기가 힘듦
    - 부분의 장애가 전체 서비스의 장애로 이어지는 경우가 발생
  - 작은 서비스 단위로 개발하는 방법을 MSA라고 한다. 각 서비스들은 스스로 돌아갈 수 있고, 독립적 배포 가능해야 한다.
- 장단점
  - 장애격리, 빠른 배포, 비용효율적 등
  - 통신비용, 트랜잭션 복잡도 증가, 데이터관리

정리

- 반면에,
- 고객의 요구사항 대비 가용자원을 고려하여 적합한 아키텍처를 채용하는 것이 바람직
  - 모든 엔터프라이즈 IT에 마이크로서비스 아키텍처를 적용하는 것은 오버 아키텍처링 또는 불필요할 수 있다.
- 모놀리식으로 관리하기에 특별히 복잡한 시스템을 운영할 상황이 아니면 마이크로서비스는 고려할 필요조차 없다.
- 시스템 복잡도 단계에 따라 아키텍처 선택 시 개발 생산성에 크게 영향을 받을 수 있기 때문에 신중을 기하여 아키텍처를 선정해야 합니다.



참고문서

- [MSA 제대로 이해하기 -(1) MSA의 기본 개념]
- [SK 주식회사 C&C 유튜브 | MSA?! 이거 보고 결정해! MSA의 모든것]
- [마이크로서비스 아키텍처 도입을 고민 중인 당신에게]