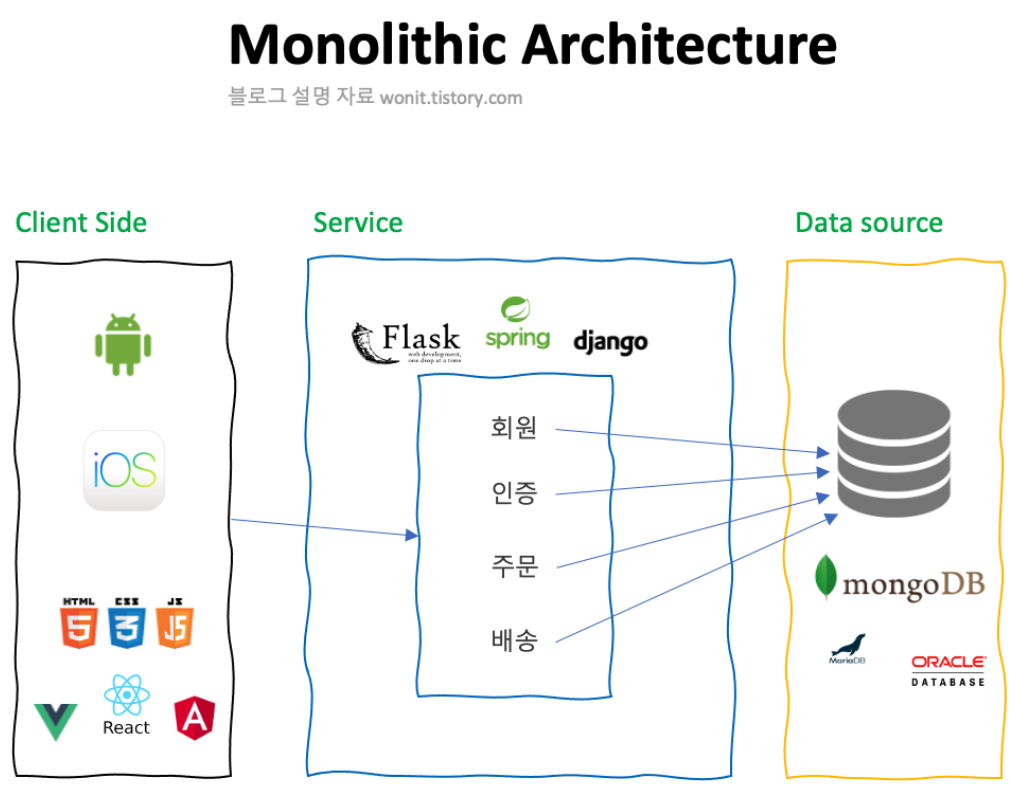
MSA란?

Due	@October 12, 2021
Tags	WEB

MSA란?

MicroService Architecture
"하나의 큰 어플리케이션을 **여러 개의 작은 어플리케이션으로** 쪼개어, **변경과 조합이 가능**하도록 만든 아키텍처"
= **독립적으로 배포 가능한 각각의 기능을 수행하는 서비스**로 구성된 프레임워크

`Monolithic Architecture`



- 소프트웨어의 모든 구성요소가 한 프로젝트에 통합되어 있는 형태

→ 프로젝트가 커진다면? **(Monolithic Architecture의 한계)**

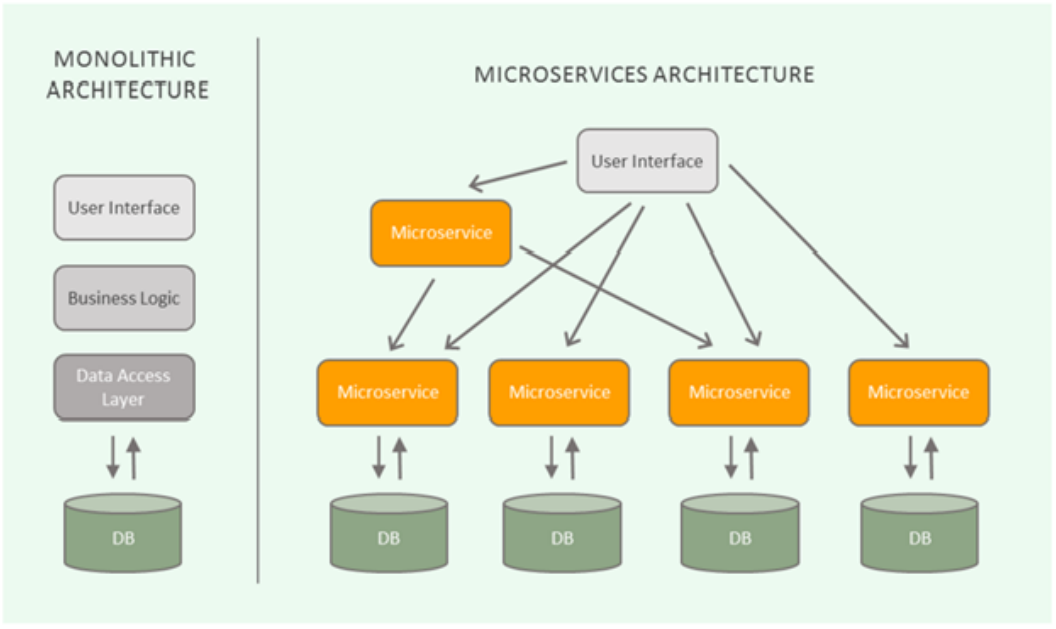
- 부분 장애가 전체 서비스의 장애로 확대될 수 있다.
  - 개발자의 잘못된 코드 배포 또는 갑작스런 트래픽 증가로 인해 성능에 문제가 생겼을 때, 서비스 전체의 장애로 확대될 수 있다.
- 부분적인 Scale-out이 어렵다. (여러 서버로 나눠 일을 처리하는 방식)
  - Monolithic Architecture에서는 사용되지 않는 다른 모든 서비스가 Scale-out 되어야 하기 때문에
- 서비스의 변경이 어렵고, 수정 시 장애의 영향도 파악이 어렵다.
  - 여러 컴포넌트가 하나의 서비스에 강하게 결합되어 있기 때문
- 배포 시간이 오래 걸린다.
  - 규모가 커지면, 작은 변경에도 높은 수준의 테스트 비용이 발생하기도 하며, 많은 사람이 하나의 시스템을 개발하여 배포하기 때문에 영향을 준다.
- 한 프레임워크와 언어에 종속적이다.

- 스프링 프레임워크를 사용할 경우, 블록체인 연동 모듈을 추가할 때 보통 node.js를 사용하는 것이 일반적이다. 하지만, 선정했던 프레임워크가 스프링이기 때문에 자바를 이용해 해당 모듈을 작성해야 한다.

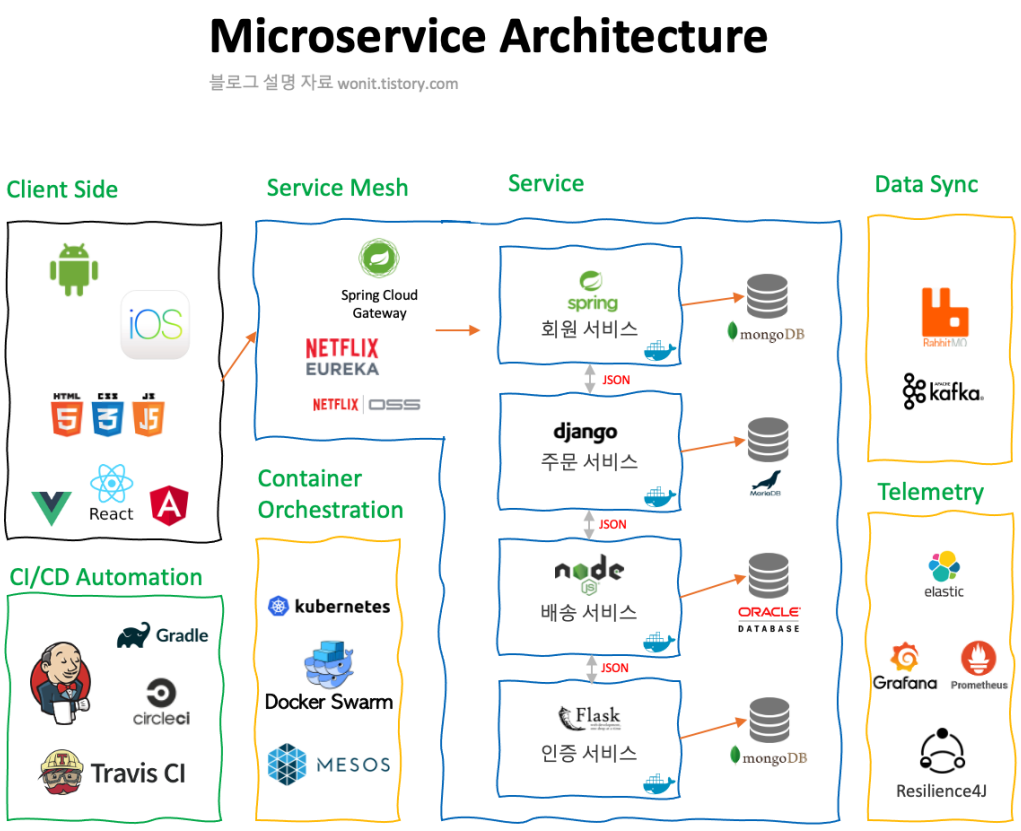
⇒ `MSA 등장`

`MSA (MicroServices Architecture)`

> ***small services, each running in its own process (스스로 돌아갈 수 있는 작은 서비스)***
> ***independently deployable (독립적 배포 가능)***



https://dzone.com/articles/what-are-microservices-actually



- MSA는 API를 통해서만 상호작용 할 수 있다.

  = 마이크로 서비스는 end-point를 API 형태로 외부에 노출하고, **실질적인 세부 사항은 모두 추상화**한다.

  = 내부의 구현 로직, 아키텍처, 프로그래밍 언어, DB, 품질 유지 체계 같은 기술적인 사항들은 모두 API에 의해 가려진다.

  ⇒ `SOA`**의 특징을 다수 공통**으로 가진다. (SOA에서 더 발전한 형태)

특징

1. 서비스들은 비즈니스 단위로 나눠져야 한다.
2. 서비스들은 최소한의 중앙 집중식 구성이 되어야 한다.
3. 서비스들은 서로 다른 언어와 DB로 구성될 수 있다.

4. 각각의 서비스들은 **HTTP API를 통해 데이터를 주고받는다.**

   → `Monolithic` : 시스템 내부의 호출, 커널 단위의 호출을 통해 데이터를 주고받음

   → `MSA` : RESTful한 API를 통해 데이터를 주고받음

⇒ 이런 특징들로 인해 변화에 민첩한 서비스를 제공할 수 있게 되는 것이다.

장점

- 어플리케이션을 구성하는 서비스 구성요소들이 모두 컨테이너에 의해 나눠지기 때문에 **최소한의 의존성**을 갖게 될 수 있다.

  ⇒ **새로운 모듈에 대한 추가가 자유롭다.**

- 서비스별로 독립적 배포가 가능하다.

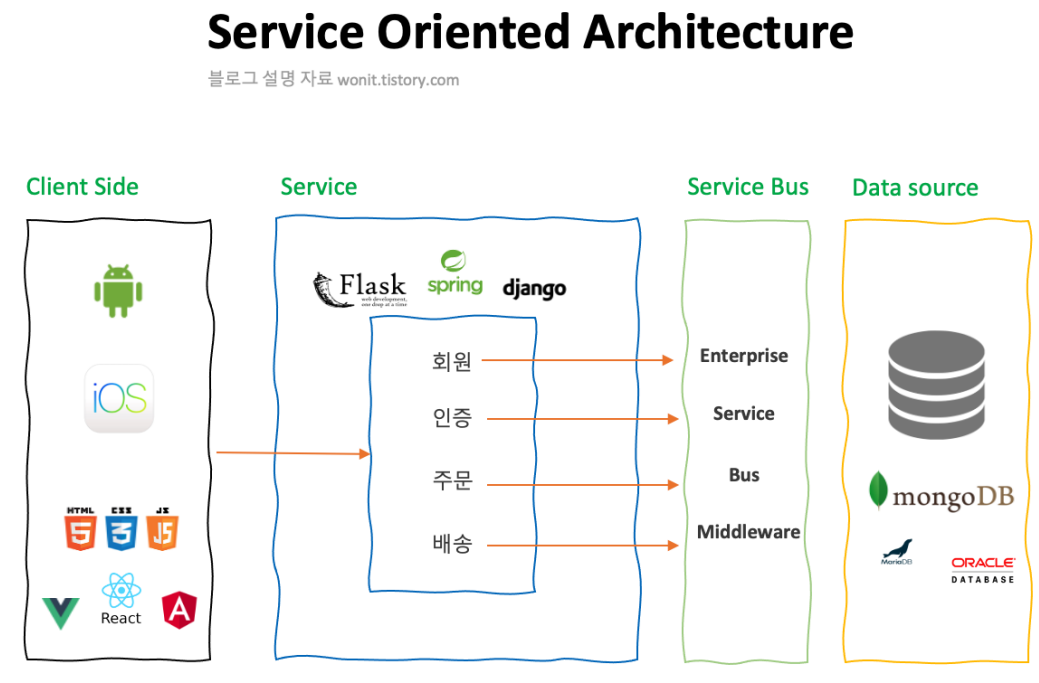
단점

- MSA를 구축하는 일이 어려움

  → `Monolithic` 서비스는 서로 내부적으로 통신했지만,

  → `Micro 서비스`는 여러 프로젝트로 나눠, 해당 프로젝트끼리 통신을 해야 하는 구조라 설정에 대한 서버를 따로 둘 정도로 복잡하게 구성

`SOA(Service Oriented Architecture)`



- 공통된 서비스를 **ESB**에 모아 사업 측면에서 **공통 기능의 집합을 통해 서비스를 제공**한다.

  > ***ESB (Enterprise Service Bus)?***
  > *서비스들을 컴포넌트화된 논리적 집합으로 묶는 미들웨어*
  > *이벤트 및 서비스에 대한 요청과 처리를 중개하여 인프라 전체 스트럭처에 분포되게 한다.*
  > *= 비즈니스 내에서 자원을 연결하고 통합*
  > *SOA의 토대가 된다.*

- **서비스?**
  - 기업의 업무에 따라 나뉘어진 하나의 소프트웨어

    = 기업의 업무 중 하나

- ESB를 이용한 SOA는 서비스 단위로 모듈을 분리 *(장점)*

  → 결합도가 낮아지는 아키텍처

  → 서비스를 지향하여 각각의 **모듈의 재사용성을 높임** + **버스 형태에 연결만 가능하다면 확장성과 유연성 증가**

- but, 하나의 DB 사용 → 끊어질 수 없는 의존성 존재 *(단점)*

  \+ 구체적이지 않은 특성 때문에 비즈니스에서의 실 성공 사례가 드물다.

⇒ `SOA` **에서 더 발전한 게** `MSA`

▼ 참고

https://wooaoe.tistory.com/57

https://velog.io/@syleemk/배민-마이크로-서비스-여행기-정리

https://byline.network/2020/12/17-108/

https://www.youtube.com/watch?v=BnS6343GTkY

https://www.samsungsds.com/kr/insights/msa.html

https://velog.io/@tedigom/MSA-제대로-이해하기-1-MSA의-기본-개념-3sk28yrv0e

https://velog.io/@tedigom/MSA-제대로-이해하기-2-MSA-Outer-Architecure

https://velog.io/@tedigom/MSA-제대로-이해하기-3API-Gateway-nvk2kf0zbj

https://velog.io/@tedigom/MSA-제대로-이해하기-4Service-Mesh-f8k317qn1b

https://velog.io/@tedigom/MSA-제대로-이해하기-5Backing-Service-lqk3b7560w

https://velog.io/@tedigom/MSA-제대로-이해하기-6Telemetry

https://martinfowler.com/articles/microservices.html

https://toma0912.tistory.com/87

https://wonit.tistory.com/487