MSA란?

2020년 6월 6일 토요일 오후 3:58

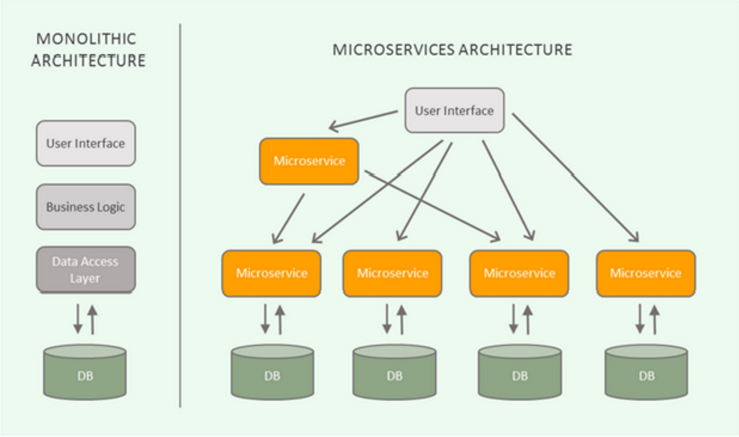
- MSA란?
  Micro Service Architecture
  하나의 큰 어플리케이션을 여러 개의 작은 어플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍쳐

- 참고 링크 : https://velog.io/@tedigom/MSA-%EC%A0%9C%EB%8C%80%EB%A1%9C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-2-MSA-Outer-Architecure

- 기존에는 Monolithic Architecture를 사용하여 모든 구성요소가 한 프로젝트에 통합되어 있는 형태였다.
  하지만 일정 규모 이상의 서비스, 수백명의 개발자가 투입되는 프로젝트에서는 큰 한계가 있다.

  a. 서비스의 구조가 커질 수록 영향도 파악이 어렵다.
  b. 빌드 시간 및 테스트시간, 배포시간이 기하급수적으로 늘어난다.
  c. 서비스를 부분적으로 scale-out하기가 힘들다.
  d. 부분의 장애가 전체 서비스의 장애로 이어지는 경우가 있다.

- MSA는 비즈니스 민첩성(Business agility)과 관련이 크다.
- Martin Folwer의 MSA 정의
  a. 스스로 돌아갈 수 있는 작은 서비스 : small services, each running in its own process
  b. 독립적 배포 가능 : independently deployable

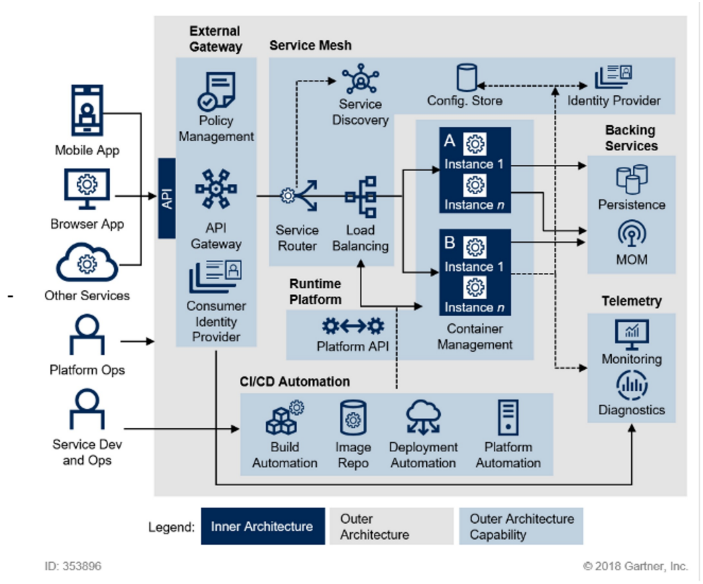


- MSA 나름대로의 정의
  a. 각각의 서비스는 크기가 작을 뿐 서비스 자체는 하나의 모놀리틱 아키텍쳐와 유사
  b. 각각의 서비스는 독립적으로 배포가 가능해야함
  c. 각각의 서비스는 다른 서비스에 대한 의존성이 최소화 되어야함
  d. 각 서비스는 개별 프로세스로 구동 되며 REST와 같은 가벼운 방식으로 통신 되어야 함.

- MSA 장점
  a. 배포(deployment) 관점
     서비스 별 개별 배포 가능(배포 시 전체 서비스의 중단이 없음
     요구사항을 신속하게 반영하여 빠르게 배포할 수 있음
  b. 확장(scaling) 관점
     특정 서비스 에 대한 확장성이 용이함
     클라우드 사용에 적합한 아키텍쳐
  c. 장애(failure) 관점
     장애가 전체 서비스로 확장될 가능성이 적음
     부분적 장애에 대한 격리가 수월함

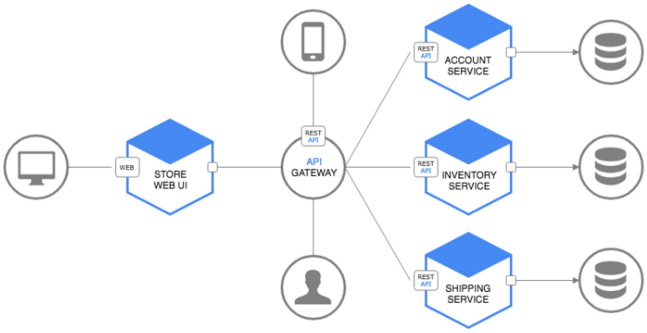
- MSA 단점
  a. 서비스가 커짐에 따라 복잡도가 기하급수적으로 늘어날 수 있다.
  b. 성능 - 서비스 간 호출 시 API를 사용하기에 통신 비용이나, Latency(하나의 패킷이 다른지점으로 보내지는 시간)가 늘어날 수 있다.
  c. 테스트 / 트랜잭션 - 서비스가 분리되어있어 테스트와 트랜잭션 복잡도가 증가하고 많은 자원을 필요로 한다.
  d. 데이터 관리 - 데이터가 여러 서비스에 걸쳐 분산되기에 한번에 조회가 어렵고 데이터의 정합성 또한 관리하기 어렵다.

- Microservices Architecture Components

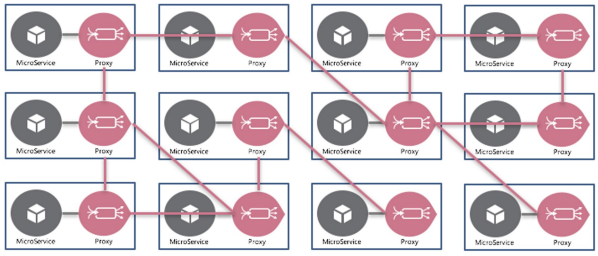
- 

- MicroService Architecture는 크게 Inner 와 Outer Architecture 두 부분으로 나뉜다.
- Inner Architecture
  내부 서비스와 관련된 architecture. 내부의 서비스를 어떻게 나누는지에 대한 설계
  a. 마이크로 서비스를 어떻게 정의할 것인가?
     쇼핑몰에서 주문하기, 장바구니담기를 같은 서비스로 넣을지 분리할 지는 비지니스나 시스템의 특성에 따라 정의됨.
     고려대상은 비즈니스 뿐만 아니라 서비스 간의 종속성, 배포 용이성, 장애대응, 운영 효율성 등 다양하다.
  b. DB Access 구조를 어떻게 설계할 것인가?
     MicroService가 사용하는 데이터는 일반적으로 일관된 API를 통해서 접근한다.
     각 마이크로 서비스에는 자체의 데이터베이스를 가질 수 있는데, 일부의 비즈니스 트랜잭션은 여러 micro service들을 걸쳐있어
     각 서비스에 연결된 데이터베이스의 정합성을 보장해 줄 수 있는 방안이 필요하다.
- Outer Architecture
  Gartner에서는 MSA의 Outer Architecture를 6개의 영역으로 분류한다.
  - External Gateway
  - Service Mesh
  - Container Management
  - Backing Services
  - Telemetry
  - CI/CD Automation

  a. External Gateway
     External Gateway는 전체 서비스 외부로부터 들어오는 접근을 내부 구조를 드러내지 않고 처리하기 위한 요소
     사용자 인증과 권한 정책관리를 수행하며, API Gateway가 가장 핵심적인 역할을 담당한다.
     API Gateway는 서버 최 앞단에 위치하며 모든 API 호출을 받아 적절한 서비스들에 메시지를 전달한다.

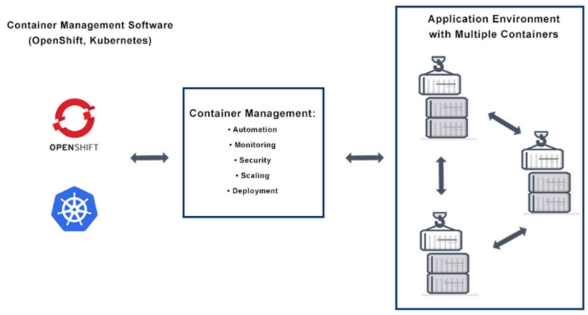


  b. Service Mesh
     Service Mesh는 마이크로 서비스 구성 요소간의 네트워크를 제어하는 역할이다.
     서비스 간에 통신을 위해서는 service discovery, service routing, 트래픽 관리 및 보안 등을 담당하는 요소가 있어야한다.



  c. Container Management
     컨테이너 기반 어플리케이션 운영은 유연성과 자율성을 가지며, 개발자가 손쉽게 접근 및 운영할 수 있는 인프라 관리 기술이기 때문에 MSA에 적합하다고 평가받고 있다.

대표적인 컨테이너 관리 환경인 Kubernetes가 Container management에 많이 사용되고 있다.
특히 AWS의 EKS, Google cloud platform의 GKE는 kubernetes를 지원하는 클라우드 서비스로, 앞으로의 어플리케이션 운영환경에 많은 변화를 가져올 것이다.



d. Backing Service
Backing Service는 어플리케이션이 실행되는 가운데 네트워크를 통해서 사용할 수 있는 모든 서비스를 말하며
MySQL등의 데이터베이스, 캐쉬, SMTP서비스 등 어플리케이션과 통신하는 Attached Resource들을 지칭하는 포괄적인 개념이다.
MSA에서의 특징적인 Backing Service들 중 하나는 Message Queue이다. MSA에서는 메시지의 송신자와 수신자가
직접 통신하지 않고 Message Queue를 활용하여 비동기적으로 통신하는 것을 지향한다.

예를 들어, MSA를 적용한 프로젝트에서 장애 발생이 일어났다고 가정해보면, 마이크로서비스 오케스트레이션이 진행되면서, 새로운 마이크로 서비스를 신규생성하고나 재생성 등의 작업을 진행하게 된다.
(오케스트레이션은 여러개의 마이크로 서비스를 묶어 새로운 서비스를 만드는 개념이다.)
만약 Message Queue를 사용하지 않는 강한 결합 구조의 경우, 여러 서비스를 걸치는 실시간 트랜잭션을 처리할 때, 하나의 서비스가 죽어버린다면 트랜잭션이 끊어지기 때문에 해당 서비스 요청을 보존할 수 없고 큰 에러가 발생하게 된다.
또한, REST 통신으로 트랜잭션 실패에 대한 처리를 구현하는 방법은 굉장히 복잡하다.
따라서 MSA에서 데이터 변경이나, 보상 트랜잭션과 관련된 처리는 Message Queue를 활용한 비동기 처리가 효율적이다.

e. Telemetry
Telemetry의 어원은 Tele(먼 거리) + metry(측정)이다. 즉, 실시간으로 먼 거리에서 원격으로 측정할 수 있다.
MSA는 상당수의 마이크로 서비스가 분산환경에서 운영되기 때문에 서비스들의 상태를 일일이 모니터링하고 이슈에 대응하는 것은 굉장히 힘들고 오랜 시간이 걸린다.
Telemetry는 서비스들을 모니터링하고, 서비스별로 발생하는 이슈들에 대응할 수 있도록 환경을 구성하는 역할을 한다.

f. CI/CD Automation
CI/CD는 어플리케이션 개발 단계를 자동화하여, 어플리케이션을 보다 짧은 주기로 고객에게 제공하는 방법이다.
지속적인 통합(Continuous Integration), 지속적인 전달(Continuous Delivery), 지속적인 배포(Continuous Deployment)가 CI/CD의 기본개념으로, 이를 자동화하는 것은 배포가 잦은 MSA 시스템에 꼭 필요한 요소 중 하나이다.