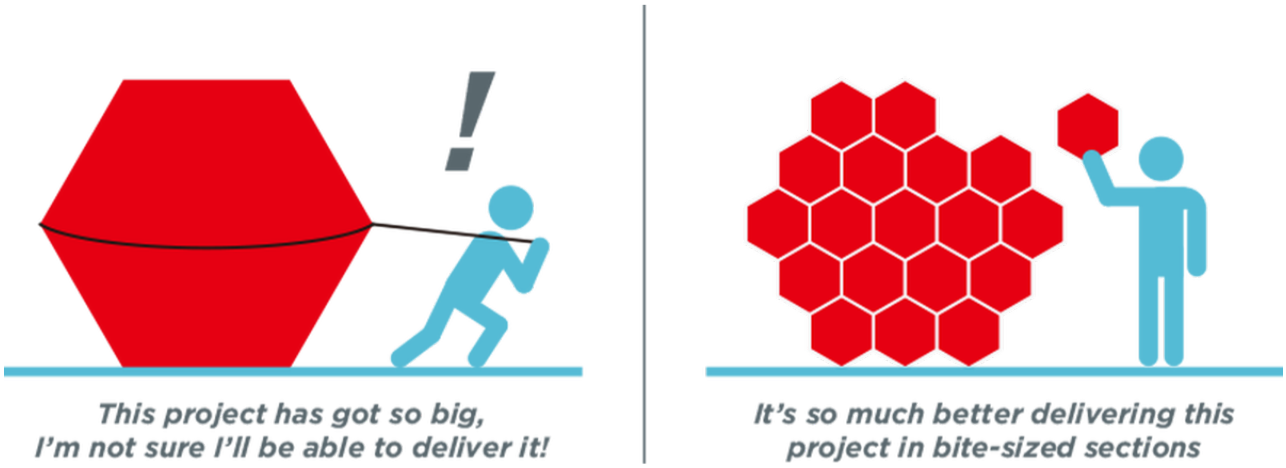
마이크로서비스 아키텍처(MSA)

*MSA*란 무엇인가 ?

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

MSA (Microservices Architecture)는 분산 컴퓨팅 아키텍처 패턴 중 하나로, 서비스 기반의 아키텍처 패턴입니다.
이를 통해 모놀리틱 아키텍처의 문제점을 해결하고, 애플리케이션의 확장성과 유연성을 개선하며, 서비스 장애가 전체 시스템에 영향을 미치지 않도록 하는 등의 이점을 제공합니다.

*MSA*의 탄생 배경

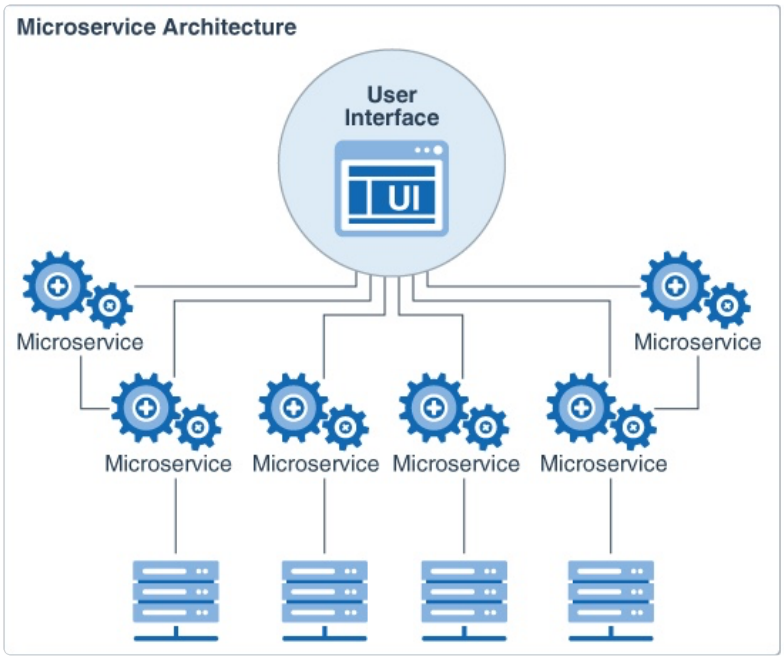


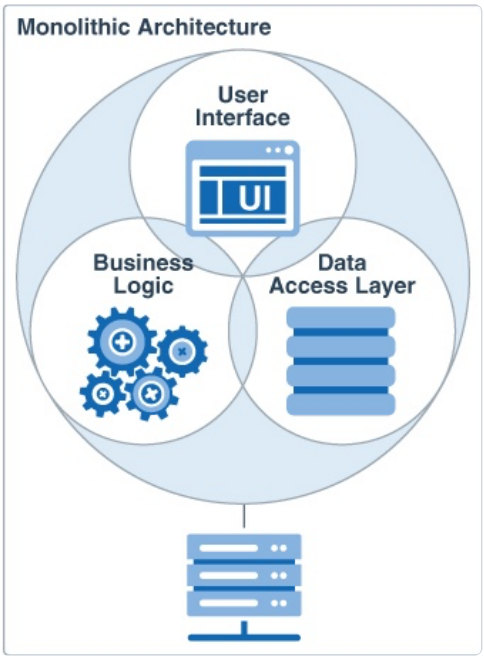
MSA(Microservices Architecture)의 탄생 배경은 기존의 모놀리식(monolithic) 아키텍처에서의 한계점을 극복하기 위한 것입니다. 모놀리식 아키텍처는 모든 서비스를 하나의 대규모 응용 프로그램으로 구현하므로, 시스템의 규모가 커질수록 유지 보수와 배포, 확장 등이 어렵고 복잡해집니다. 이에 따라, 이를 해결하기 위한 새로운 아키텍처 모델인 MSA가 등장하게 됩니다.

*Monolithic*의 한계

- 💀 전체 시스템 구조 파악의 문제
- 💀 빌드 시간 및 테스트, 배포 시간의 급증
- 💀 서비스의 특정 부분만 scale-out을 하기 어렵습니다.
- 💀 부분의 장애가 전체 서비스의 장애로 이어질 수도 있습니다.

*MSA*의 목표





🚀 **목표**

MSA는 하나의 대규모 응용 프로그램을 여러 개의 작은 서비스로 분해하므로, 각각의 서비스는 독립적으로 개발, 배포, 확장할 수 있습니다. 이를 통해 시스템의 유연성과 확장성이 높아지고, 개발 및 운영의 효율성도 향상됩니다.

*MSA*의 장단점

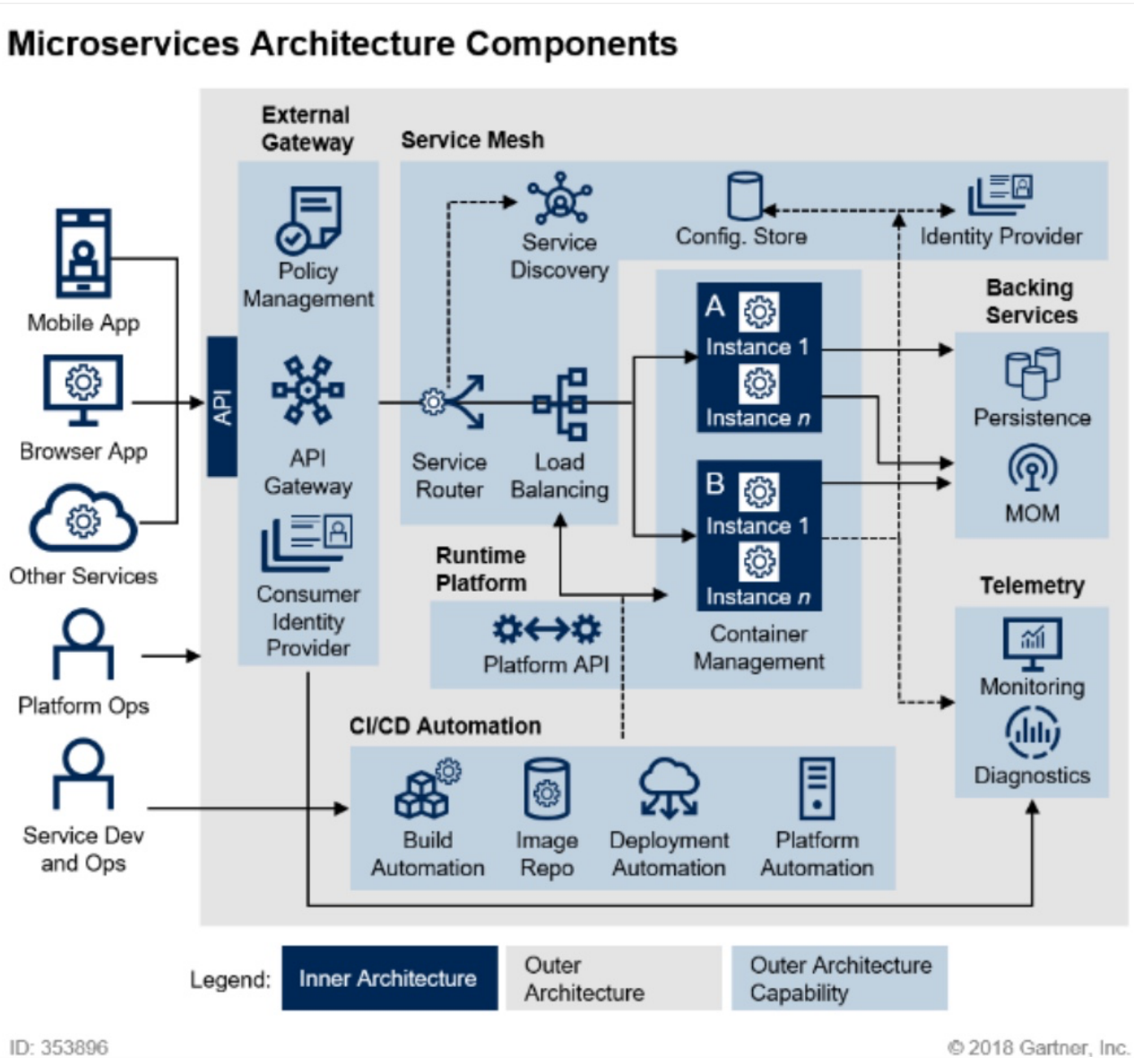
💯 **장점**

1. 서비스 별 개별 배포가 가능합니다. (배포 시 전체 서비스의 중단이 없습니다.)
2. 특정 서비스에 대한 확장성이 유리합니다. (scale-out)
3. 일부 장애가 전체 서비스로 확장될 가능성이 적습니다.

💔 **단점**

1. 서비스 간 호출 시 API를 사용하므로, 통신 비용이나 Latency에 대해 성능
2. 서비스가 분리되어 있어 테스트와 트랜잭션의 복잡도가 증가합니다.
3. 데이터가 여러서비스에 분산되어 조회하기 어렵습니다.

*MSA*를 구성하는 주요 컴포넌트



1 Service

MSA를 이루는 기본 단위이며, 작은 서비스들이 모여 전체 시스템을 구성합니다. 각 서비스는 독립적으로 실행되며, 자체적으로 데이터를 관리하고 프로세스를 실행합니다.

2 API Gateway

클라이언트가 서비스에 접근할 때 필요한 진입점 역할을 합니다. API 게이트웨이는 클라이언트 요청을 받아 해당 요청을 처리할 서비스를 찾아 전달하고, 로드밸런싱과 인증/인가 기능도 수행합니다.

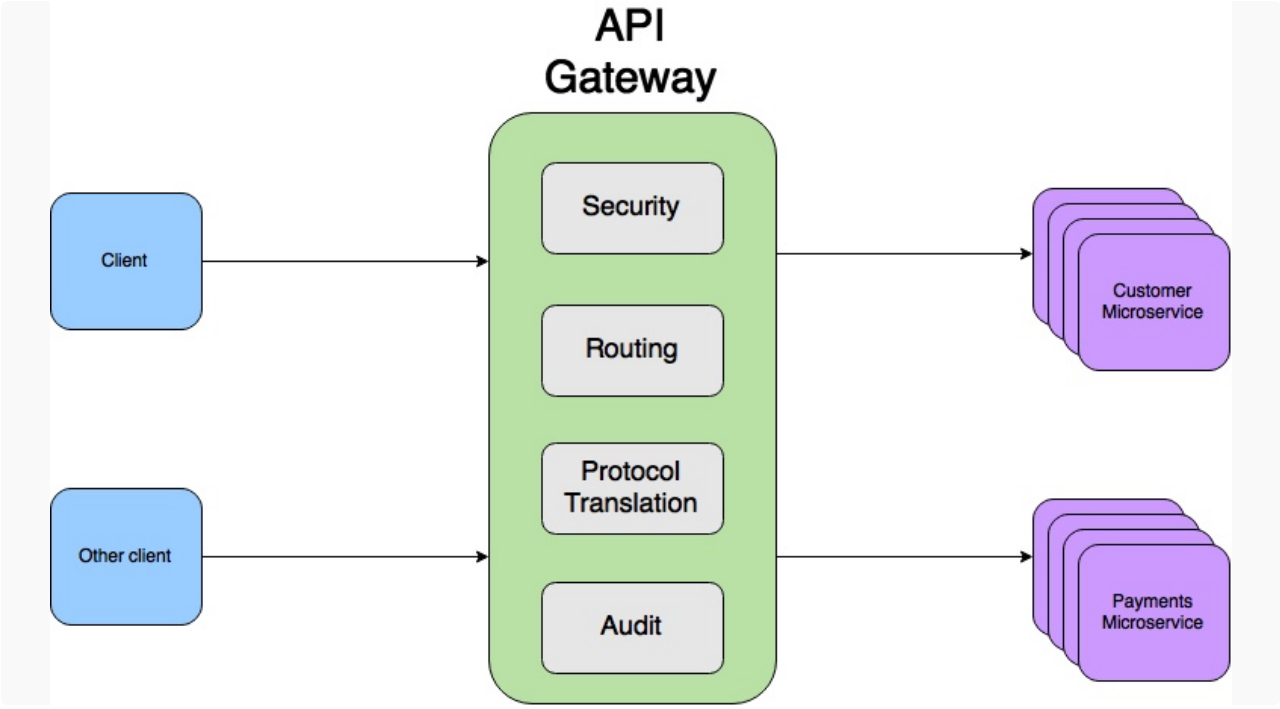
3 서비스 디스커버리

각 서비스의 위치와 상태를 추적하고 관리하는 도구입니다. 서비스 디스커버리는 각 서비스의 IP 주소와 포트 번호를 추적하며, 서비스의 동적인 추가와 삭제를 감지합니다. 이를 통해 클라이언트는 서비스의 위치와 상태를 알 수 있으며, 이를 기반으로 API 게이트웨이는 요청을 전달합니다.

4 분산 데이터 관리

MSA에서 각 서비스는 자체적으로 데이터를 관리하며, 서로 다른 서비스 간에 데이터를 공유해야 할 경우 분산 데이터 관리 기술을 사용합니다. 이를 위해 NoSQL 데이터베이스, 메시지 큐 등의 기술을 사용할 수 있습니다. 분산 데이터 관리 기술을 사용하면 데이터 일관성과 동시성을 보장할 수 있으며, 서비스 간의 의존성을 줄일 수 있습니다.

*MSA*에서의 *API Gateway*의 역할



MSA에서는 여러 개의 작은 서비스로 분리되어있기 때문에, 클라이언트에서는 각 서비스의 진입점을 알아내야 한다.

💯 *API Gateway*의 이점

1. 로드밸런싱을 통한 트래픽 분산
2. 인증/인가 일괄 처리
3. API 버전 관리를 통한 호환성 유지
4. 서비스 디스커버리 기능

🔨 대표적인 *API* 게이트웨이 도구

1. Zuul: Netflix에서 개발한 API 게이트웨이 도구로, 로드밸런싱, 인증/인가, 서비스 디스커버리 등의 기능을 제공한다.
2. Kong: 오픈소스 API 게이트웨이 도구로, 로드밸런싱, 인증/인가, API 버전 관리, 서비스 디스커버리 등의 기능을 제공한다.
3. Ambassador: Kubernetes 기반의 API 게이트웨이 도구로, 로드밸런싱, 인증/인가, 서비스 디스커버리 등의 기능을 제공한다.

MSA에서의 보안 문제🔒는 어떤 것이 있는가?

1️⃣ *MSA*에서의 보안 이슈

MSA에서는 여러 개의 작은 서비스들이 독립적으로 운영되기 때문에, 전통적인 모놀리틱 아키텍처에 비해 보안 이슈가 더 복잡해질 수 있다. 예를 들어, 분산 환경에서서 인증/인가, 데이터 무결성, 서비스 간 통신 등에 대한 보안이 필요하며, 이를 위한 적절한 보안 정책과 기술이 필요하다.

2️⃣ 분산 환경에서의 인증/인가와 권한 부여 방법

대표적으로 OAuth 2.0, OpenID Connect와 같은 프로토콜을 이용하여 각 서비스의 API를 보호하고, 권한 부여를 할 수 있다. 또한 JWT(JSON Web Token)과 같은 기술을 이용하여, 클라이언트와 서버간의 인증과 권한 부여를 할 수 있다.

3️⃣ 서비스 간의 보안 통신을 위한 암호화 방법

서비스 간의 통신을 보호하기 위해서는, 암호화된 통신을 구현해야 한다. 대표적으로 SSL/TLS, HTTPS 프로토콜 등을 이용하여 서비스 간의 통신을 암호화할 수 있다. 또한, API 게이트웨이와 서비스 간의 통신에 대해서도 보안을 강화할 수 있다.

4️⃣ *MSA*에서의 보안 모니터링과 디버깅

MSA에서는 보안 이슈가 발생할 가능성이 높기 때문에, 적절한 보안 모니터링이 필요하다. 각 서비스의 로그 및 메트릭 데이터를 수집하여, 이를 기반으로 보안 이슈를 탐지하고 대응할 수 있다. 또한, 서비스 간의 통신 데이터를 검증하고, 불법적인 요청을 차단할 수 있는 방법도 고려해야 한다. 이를 위해, API 게이트웨이와 분산 추적 및 로깅 도구를 이용하여 보안 이슈를 식별하고 대응할 수 있다.

MSA의 도입 시 고려해야 할 사항🖍️

✅ 조직 문화와 개발 방법론의 변화에 대한 대비

MSA의 도입은 기존의 모놀리식 아키텍처와는 다른 접근 방법을 필요로 합니다. 따라서 조직 문화와 개발 방법론의 변화를 대비하고, 이에 대한 교육과 지원이 필요합니다.

✅ 모놀리식 아키텍처에서의 전환 방법과 이슈

MSA는 모놀리식 아키텍처와 다른 접근 방법을 사용하므로, 기존 시스템에서의 전환 방법과 이슈를 고려해야 합니다. 예를 들어, 기존 시스템에서의 기능 분리와 서비스 분해, 데이터 일관성 유지 등을 고려해야 합니다.

✅ *MSA* 도입 시 기존 시스템과의 통합 방법과 이슈

MSA 도입 시 기존 시스템과의 통합 방법과 이슈를 고려해야 합니다. 이를 위해서는 API 게이트웨이나 이벤트 버스를 활용하여 통합할 수 있습니다. 또한 데이터 일관성과 트랜잭션 관리를 위한 방안도 고려해야 합니다.

✅ 확장성 고려

MSA의 핵심 가치 중 하나는 확장성입니다. 따라서, 서비스 확장을 고려한 아키텍처 설계와 구현, 로드밸런싱, 모니터링 등이 필요합니다.

✅ 안정성과 신뢰성

서비스의 안정성과 신뢰성은 MSA 도입 시 고려해야 할 중요한 요소입니다. 따라서, 서비스의 고가용성, 장애 대응 방안 등을 고려하고 이를 위한 모니터링과 로깅을 설정해야 합니다.

✅ **분산 시스템에서는 보안 문제**

각 서비스와 API의 보안을 고려하고, 인증과 권한 부여, 데이터 보호 등을 고려해야 합니다.