

MSA?
MFE?

조새나라



01

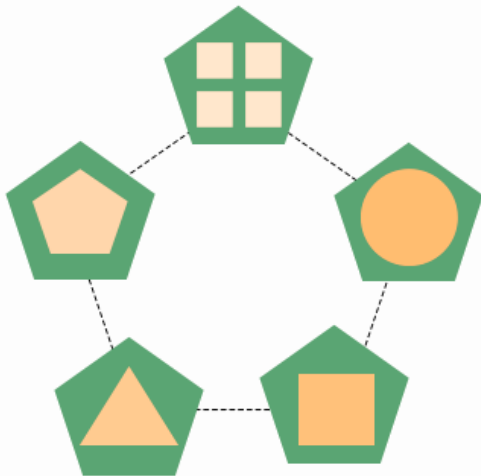
MSA

Microservice Architecture

What is MSA?

정의

마이크로서비스(microservice) 아키텍처는 애플리케이션을 느슨하게 결합된 서비스의 모임으로 구조화하는 서비스 지향 아키텍처(SOA) 스타일의 일종인 소프트웨어 개발 기법.

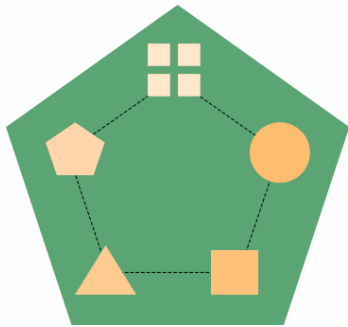


What is MSA?

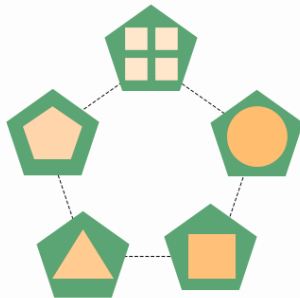
정의

모든 서비스가 강하게 연결되어있는 기존의 모노리식(Monolithic) 아키텍처의 한계를 해결하고자 독립적인 서비스 단위로 개발, 배포, 유지보수를 하도록 아키텍처를 설계하는 것.
공식적인 정의나 산업적인 합의는 없지만 공통된 철학을 공유하고 있다.

Monolith



Microservices



What is MSA?

철학

단일 책임 원칙 중시

“한 가지 일을 하되 잘하라”
(Do one thing and do it well)

서비스의 크기는 작다

하나의 기능을 수행하는데
섬세하다.

테스트 및 전개의 자동화

관리 및 운용상의 부담을 완화하며
병렬로 개발할 수 있다.

안티프래질 시스템

문화와 디자인 철학은
실패와 고장을 받아들여야 한다.

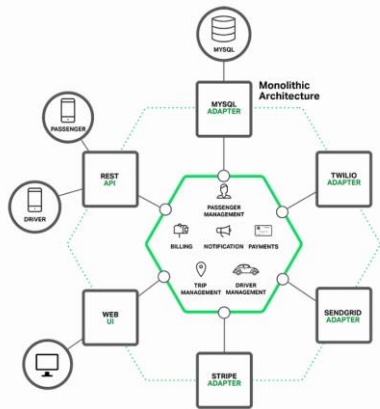
최소한이고 완전하다

각 서비스를 유연하고 회복력 있고
구성할 수 있다.

Monolithic? MSA?

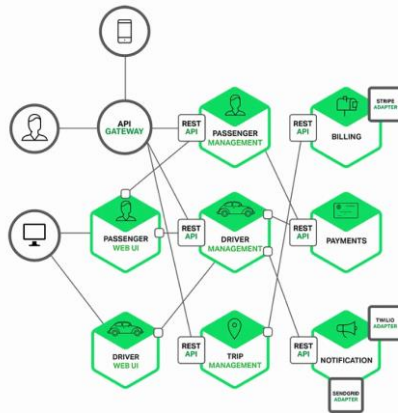
Monolithic

밀접하게 연결된 모듈들을 개발하고
하나의 결과물로 패키징하여 배포되는 형태



MSA

한 가지 일만 수행하는 모듈을
독립적으로 개발을 하고 배포되는 형태

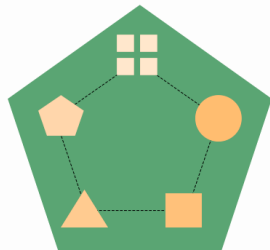


Monolithic? MSA?

Monolithic

장점 : 단순한 구조로 개발, 배포, 테스트가 쉬움
단점 : CI/CD가 어려우며 각 모듈의 밀접한
관계로 확장성과 운영, 장애 회복력이 약함

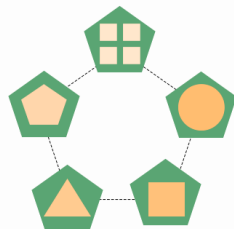
Monolith



MSA

장점 : 유연하고 확장성 있는 개발이 가능하며
CI/CD에 자유롭다
단점 : 설계의 어려움, 서비스간 정보의 장벽,
더 많은 통신 비용, 테스트와 데이터 관리의 어려움

Microservices



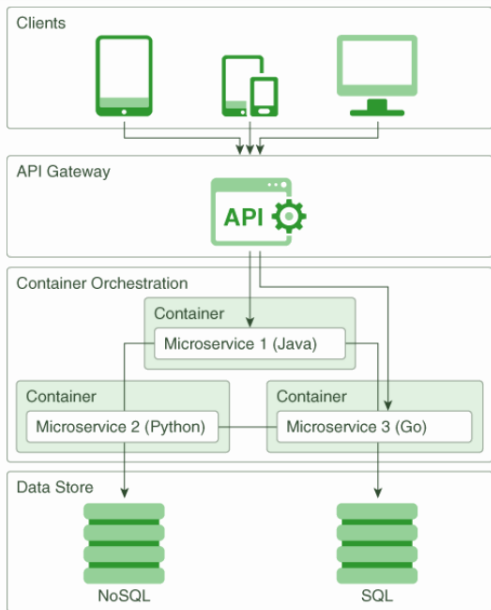
vs

Microservice Architecture

구조

공식적인 표준은 없지만
정형화된 구조는 다음과 같다.

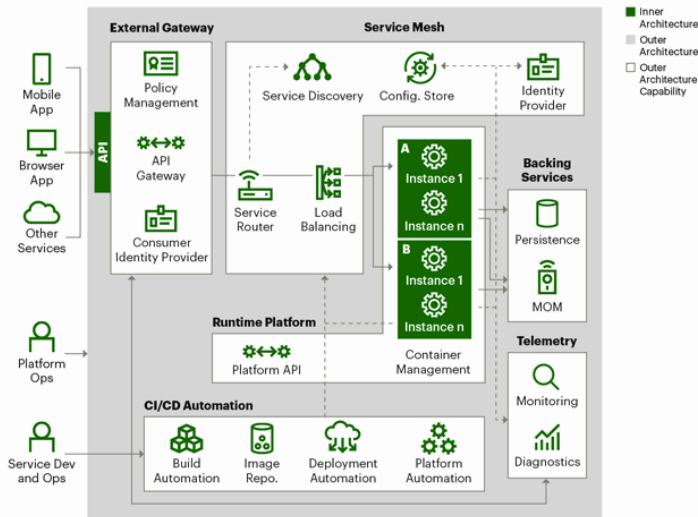
- 모든 클라이언트는 하나의 서버로 요청(API Gateway)
- 게이트웨이 서버에서 각 요청을 적절한 서비스로 라우팅
- 각 서비스들은 독립적으로 작동하며 언어와 DB에 자유로움



Microservice Architecture

컴포넌트 구조

Functional Components of a Microservice Platform



더 세밀하게 영역별로 나눈 구조는 다음과 같다

- Inner Architecture : 내부 서비스 구축에 대한 설계 영역
- Outer Architecture : 서비스의 개발, 배포, 실행 등의 영역

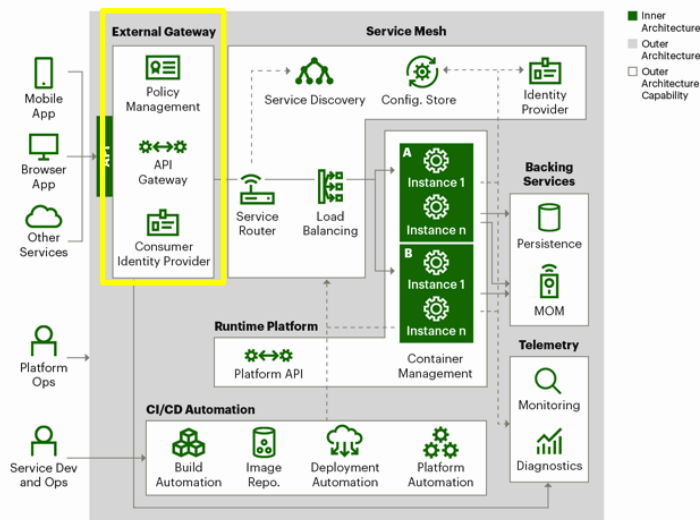
Outer Architecture는 다시 아래의 요소로 나누어 진다.

- External Gateway
- Service Mesh
- Runtime Platform
- Backing Service
- Telemetry
- CI/CD Automation

Microservice Architecture

Outer Architecture

Functional Components of a Microservice Platform



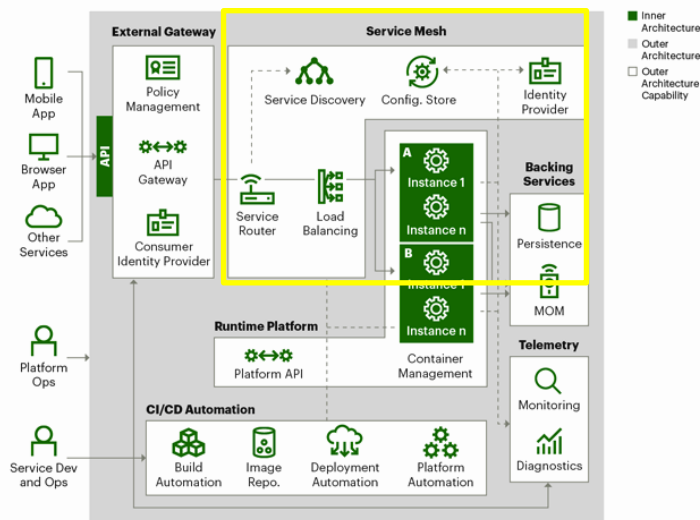
External Gateway

- 서버의 최앞단에서 외부의 모든 API 요청을 받으며 내부 구조를 드러내지 않고 처리하는 것을 목적으로 함.
- 사용자의 인증, 인가 등의 권한 정책을 관리를 처리함.
- 인증된 사용자의 명령을 처리함.

Microservice Architecture

Outer Architecture

Functional Components of a Microservice Platform



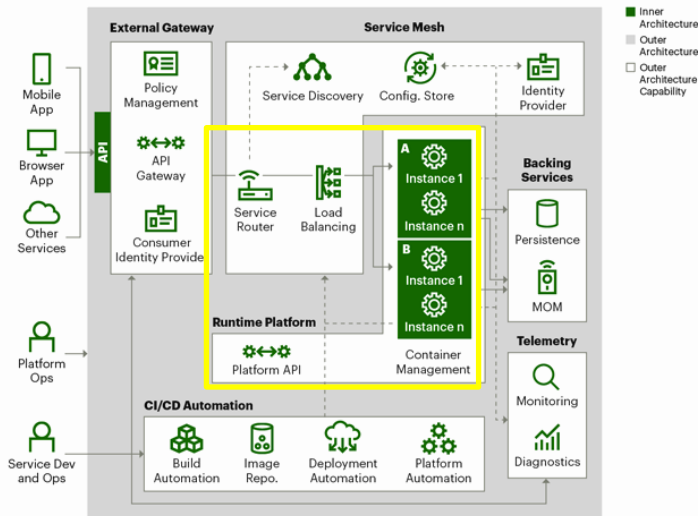
Service Mesh

- 내부 서비스 간의 통신을 담당.
- 라우팅, 트래픽 관리(로드 밸런싱), 보안 등을 처리함.

Microservice Architecture

Outer Architecture

Functional Components of a Microservice Platform



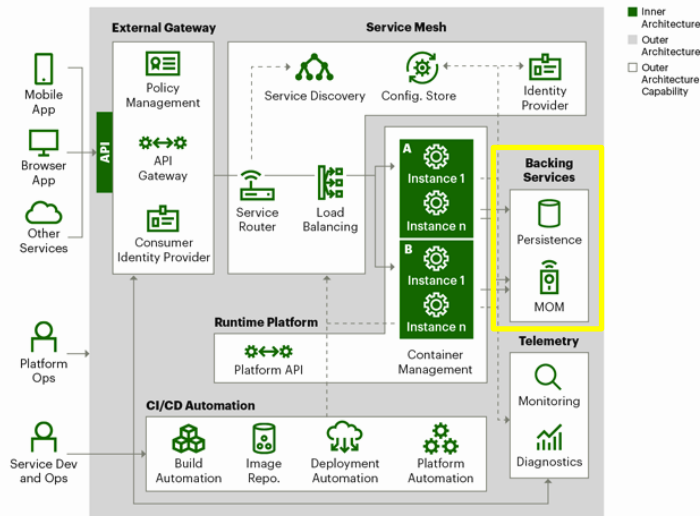
Runtime Platform

- 실제 로직이 수행되는 영역.
- 컨테이너 기반으로 서비스가 다양한 환경에 배포 및 실행될 수 있도록 환경이 구성됨. (이때 주로 도커와 쿠버네티스를 사용.)

Microservice Architecture

Outer Architecture

Functional Components of a Microservice Platform



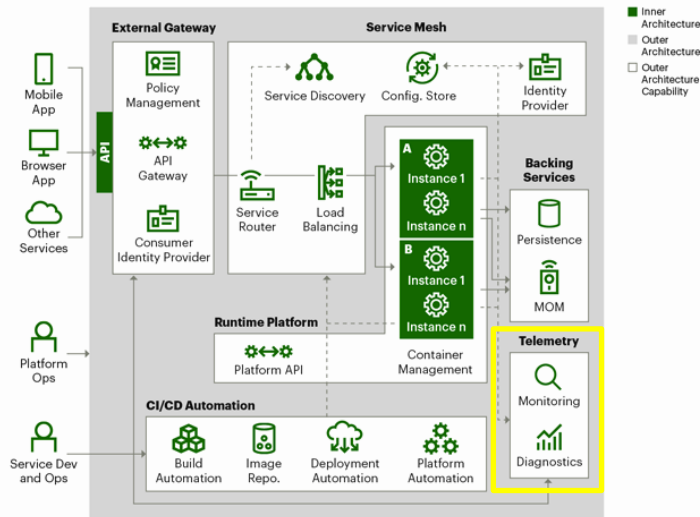
Backing Services

- 어플리케이션이 실행 중에 네트워크를 통해 사용할 수 있는 서비스
- 데이터베이스, 캐싱 시스템, SMTP 서비스 등 어플리케이션과 통신하는 attached Resource를 지칭하는 포괄적인 개념.
- 가장 특징적인 기능은 Message Queue를 활용한 비동기적 통신.

Microservice Architecture

Outer Architecture

Functional Components of a Microservice Platform



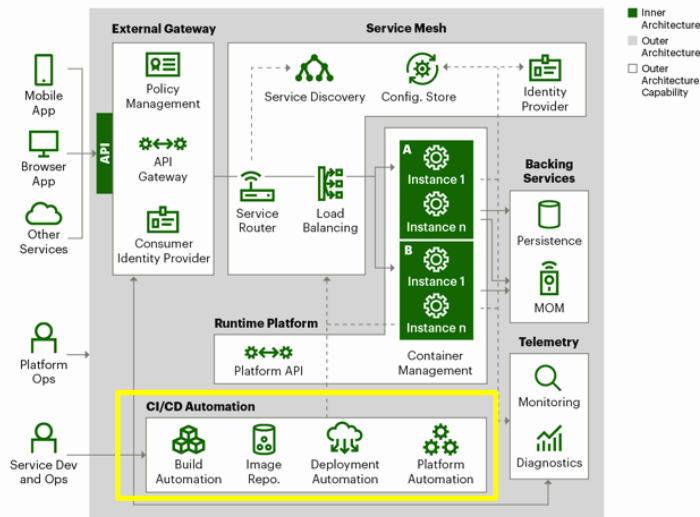
Telemetry

- Tele(먼 거리) + metry(측정).
- 각 서비스들의 상태를 모니터링하고 이슈에 대응하는 환경을 구성하는 영역.

Microservice Architecture



Outer Architecture

Functional Components of a Microservice Platform



CI/CD Automation

- CI/CD의 자동화를 통해 보다 쾌적한 MSA 환경을 제공하는 영역.
- 복잡한 구조 및 잦은 배포가 일어나는 MSA에 필수적인 영역.



02

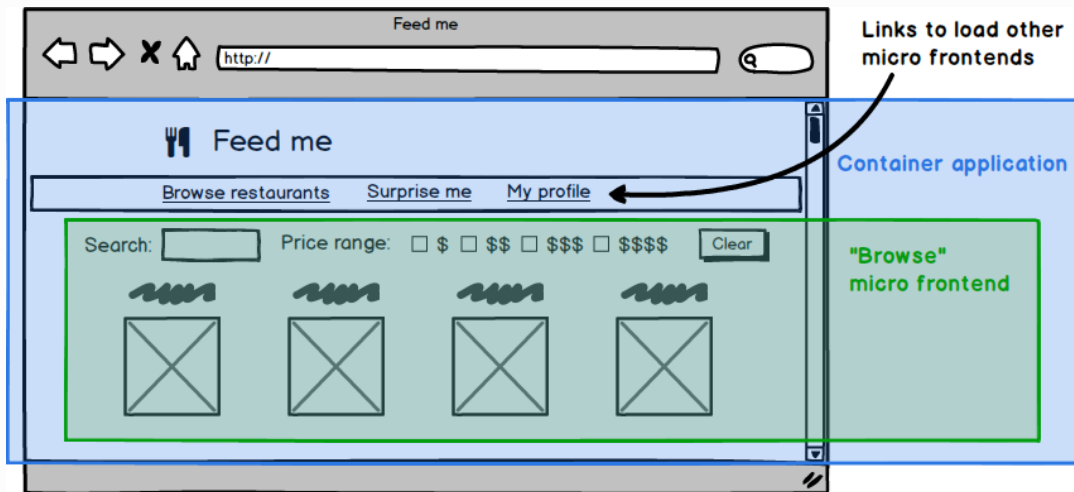
MFE

Micro-Frontend

What is MFE?

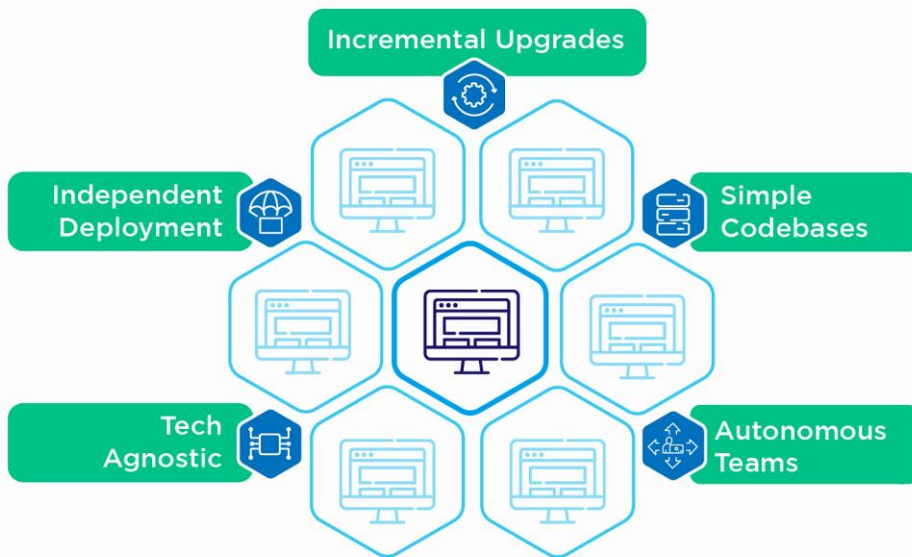
정의

프론트엔드의 단일 구조를 개별적으로 개발, 테스트, 배포할 수 있는 덩어리로 분해한 패턴.
즉, 백엔드에서 유행한 마이크로서비스(microservice)의 개념을 프론트엔드에 도입한 것이다.



What is MFE?

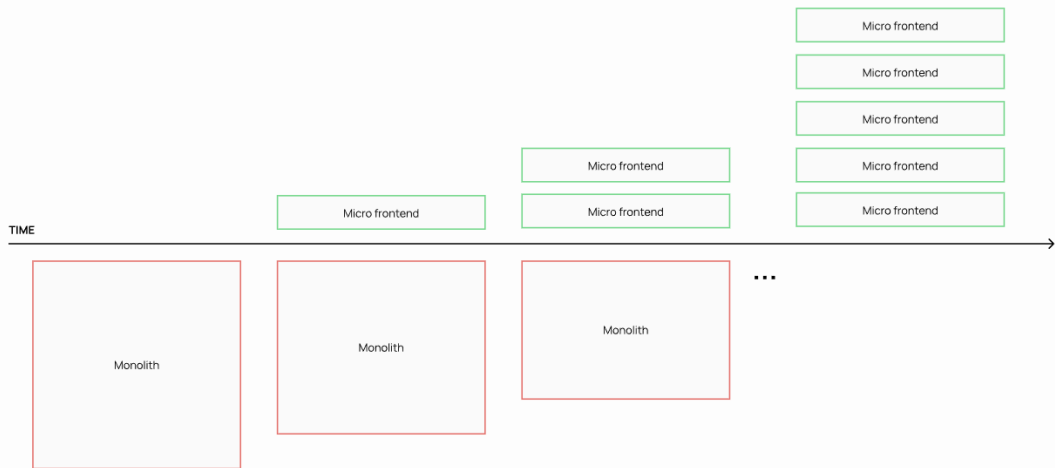
주요 이점



What is MFE?

점진적인 업그레이드

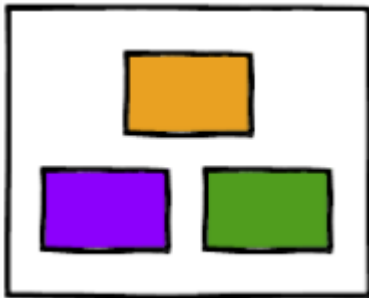
기능별로 분리가 되어 개발되기 때문에
이전보다 더 점진적인 방식으로 프론트엔드의 일부를 업그레이드 혹은 마이그레이션 가능.



What is MFE?

단순한 코드 베이스

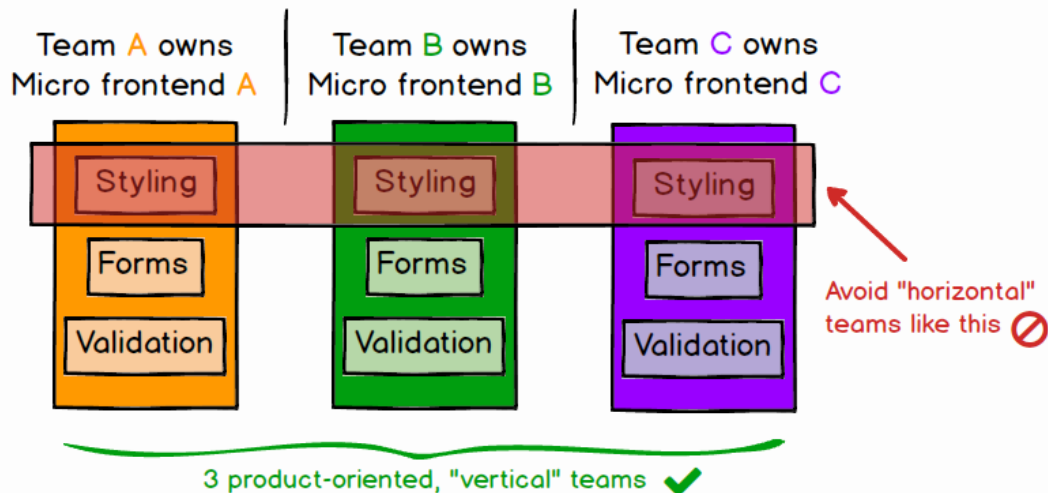
각 마이크로 프론트엔드는 독립적인 개발을 수행하기 때문에 코드 기반이 더 작고 단순함.
이는 각 구성 요소간의 의도하지 않은 결합의 가능성을 줄어들어 더 깔끔한 코드를 제공.



What is MFE?

자율적인 팀

코드 베이스와 배포 주기 등이 모두 분리되기 때문에 자율적으로 팀이 구성 가능하며
기능적으로 구성된 팀은 기술적으로 묶은 팀보다 훨씬 응집력이 높고 효율적임.

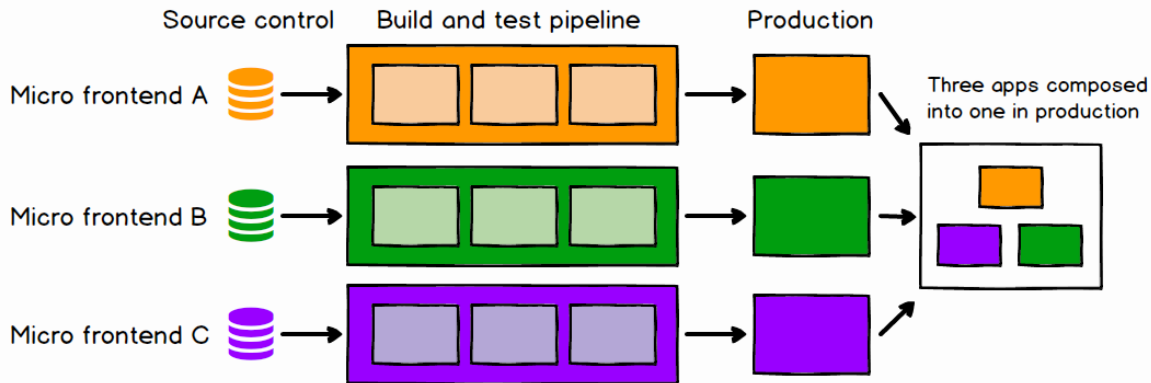


What is MFE?

독립적인 배포

마이크로서비스와 마찬가지로 독립적인 배포 가능성이 핵심.

배포의 범위가 좁아지기에 관련 위험도도 줄어듦.

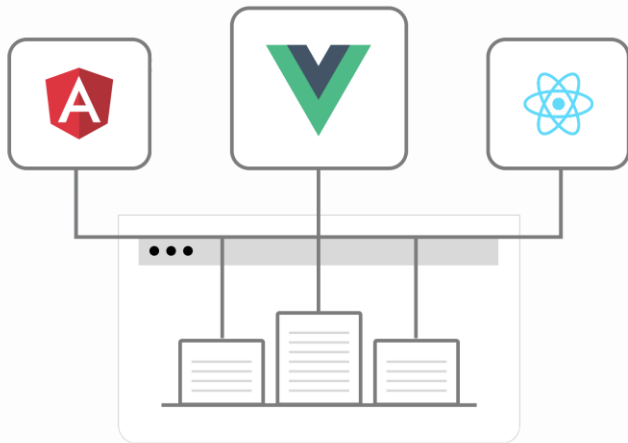


What is MFE?

기술 불가지론

기술에 구애 받지 않음.

즉, 각 팀이 마이크로 프론트엔드에 가장 적합한 기술 스택을 선택하여 유연하게 개발할 수 있음.



What is MFE?

주요 단점



What is MFE?

커진 다운로드 사이즈

독립적으로 구축된 MFE의 JS 번들은 각 MFE들과 종속된 라이브러리들이 존재하게 된다.

예를 들면 동일한 React를 쓰는 MFE 팀이더라도 독립적으로 빌드, 배포를 하기 때문에
사용자는 n번의 다운로드가 필요하다.

이를 해결하기 위해서 공통 종속성을 외부화 하는 방법이 있으나 암묵적인 합의가 필요하다.

**Bigger
Download Sizes**



What is MFE?

환경 차이

MFE는 기본적으로 다른 MFE의 환경을 고려할 필요도 없으며 기존의 프로덕션의 환경이 아닌
빈 페이지에서 독립적으로 실행할 수도 있는 것이 장점.
하지만 프로덕션 환경과 상당히 다른 환경에서 개발하는 경우
정상적으로 작동하지 않을 수 있는 위험이 존재한다.

해결책은 프로덕션 환경과 유사한 환경에서 정기적으로 통합, 테스트를 하여
최대한 빠르게 이슈를 확인하고 해결하는 것이 최선이다.



**Differences
in Environment**

What is MFE?

운영의 복잡성

백엔드의 마이크로서비스와 비슷하게 운영의 복잡성이 발생함.

아키텍처가 분산되며 필연적으로 더 많은 레포지토리, 빌드/배포 파이프라인, 서버, 도메인 등이 발생하여 관리포인트가 증가함.

이를 효율적으로 관리하기 위해선 자동화, 테스트, 최소한의 품질 보증에 대해서 확신이 있을 경우 MFE를 시작하는 것이 좋다.

Management
Complexity



What is MFE?

규정 준수 문제

독립적인 MFE들의 코드 베이스가 일관성 있는 품질을 제공하고 개발 규정을 준수할 것이라 기대하기는 어렵다.

모든 팀에서 일관성 있는 품질과 프로세스를 유지하려면 뛰어난 리더십을 지닌 리드 개발자가 필요하며 정기적으로 각 팀의 코드를 검토하고 감독해야 할 것이다.



Compliance
Issues

Micro-Frontend 구현

통합 접근법

iframe 기반 MFE

iframe내에 캡슐화하여
런타임 시 구성.

웹 컴포넌트 기반 MFE

재사용과 캡슐화가 가능한
웹 컴포넌트 형태로 구성.

Module Federation

Webpack의 Module
Federation 기능을 통한 구성.

NPM Packages

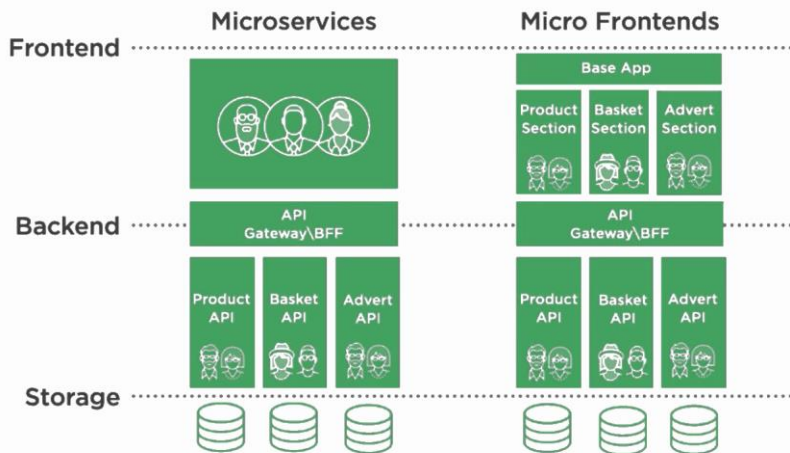
NPM 패키지를 통한
빌드 타임에서의 통합.

스크립트 로딩 MFE

<script>태그를 사용한
가장 기본적인 통합 방법.

Micro-Frontend

구조

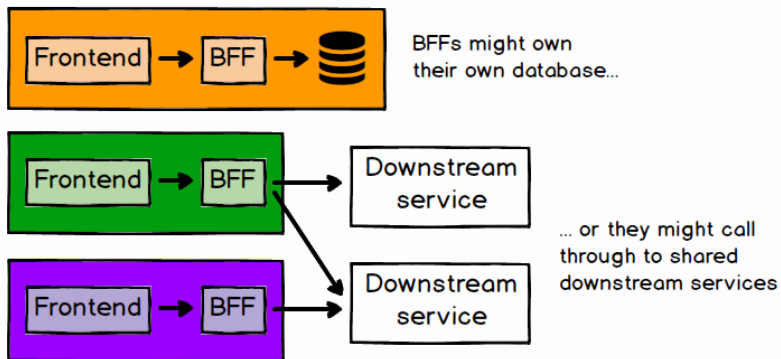


MFE역시 공식적인 표준은 없지만
정형화된 구조는 다음과 같다.

- 베이스가 되는 앱에 기능별 MFE가 구성됨
- 각 기능끼리의 통신은 최소화하거나 없어야 함
- 각 MFE의 API 요청은 API Gateway 혹은 각 BFF를 통하여 처리

Micro-Frontend

백엔드 통신



MFE에서 백엔드 통신은 개별 BFF를 두는 것을 가장 권장하고 있다.

BFF는 Backend For Frontend의 약자로서
프론트엔드를 위한 중간 서버를 구현한 것이다.

BFF는 기존의 API에서 필요한 데이터만 받는 Partial Response를
제공할 수 있기 때문에 MFE와 잘 맞는 패턴이다.

또한 BFF가 필요한 기본적인 이유는 MFE 팀이 다른 팀에서 그들을
위해서 무언가를 구축해주길 기다릴 필요가 없다는 것이다.

이는 마이크로 서비스의 철학과도 들어맞는 부분이다.

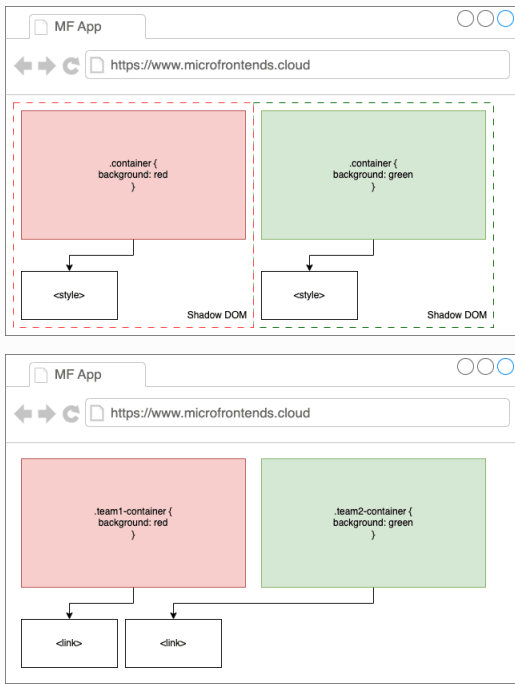
Micro-Frontend

스타일링

MFE가 가지는 큰 난관은 바로 CSS 스타일링이다.
CSS는 전역적이고 상속적이며 모듈 시스템, 네임 스페이스가 없다.

하지만 현대적인 프론트엔드 개발 환경에서는 이미 이러한 CSS를 쉽게 관리할 수 있는 개발 방법론들이 등장하였다.

대표적으로 웹 컴포넌트 기반의 Shadow DOM과 BEM과 같은 네이밍 컨벤션이 있다.



결론

마이크로 서비스는 백엔드, 프론트엔드를 가리지 않고 프로젝트가 거대해지고 복잡해짐에 따라 필요성이 커지고 있다.

이러한 강한 결합과 응집력을 줄이도록 하여 여러 문제점들을 해결하는 데 좋은 해결책임은 분명하지만 표준화 되어있지 않은 구현 방법, 장점과 함께 동반되는 새로운 단점과 같은 문제들이 존재한다.

하지만 현재 마이크로서비스는 현대적인 웹 개발 아키텍처에 표준적인 방법으로서 자리잡고 있기 때문에 조직의 적합성과 별개로 진지한 검토는 필요한 것으로 보인다.

감사합니다!

출처

<https://en.wikipedia.org/wiki/Microservices>

<https://velog.io/@juhyeon1114/Micro-Service-Architecture-%EA%B0%9C%EB%85%90-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0>

<https://martinfowler.com/articles/micro-frontends.html#IncrementalUpgrades>

<https://commercelayer.io/blog/composable-commerce-with-micro-frontends>

<https://medium.com/outreach-prague/micro-frontends-comparing-leading-frameworks-cb54cd9f7a03>

<https://www.clickittech.com/developer/microfrontends/>

<https://dev.to/florianrappl/css-in-micro-frontends-4jai>