통 합 구 현

MSA 이해

한국기술교육대학교
온라인평생교육원

학습내용

- 마이크로 서비스 아키텍처(MSA) 이해
- 거버넌스 모델
- DevOps

학습목표

- 마이크로 서비스 아키텍처(MSA)의 구조에 대해 설명할 수 있다.
- 거버넌스 모델특징을 설명 할 수 있다.
- DevOps의 정의 및 등장배경 그리고 사이클 개발자의 필요 역량을 설명 할 수 있다.

1. 마이크로 서비스 아키텍처(MSA) 이해

1) MSA란?

(1) 모노리틱 아키텍처(Monolithic Architecture)

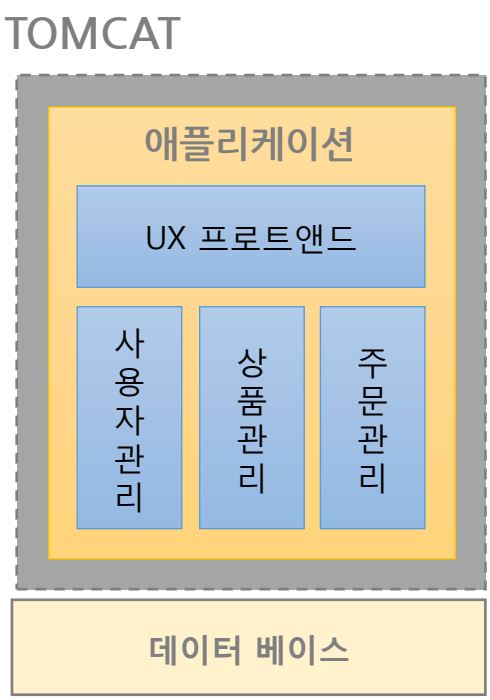
모노리틱 아키텍처

- 하나의 애플리케이션 내에 모든 로직이 들어가 있는 구조

① 전통적인 애플리케이션 개발 스타일로 테스트와 배포 용이

② 작은 크기의 애플리케이션 개발에 유리
- 규모가 큰 애플리케이션의 경우 특정 컴포넌트나 모듈의 문제가 다른 컴포넌트에 영향을 줄 수 있음

③ 특정 컴포넌트를 수정하고 재배포시 전체 애플리케이션을 재 컴파일해서 배포해야 함
- 빌드, 배포, 서버의 기동 시간의 증가 초래



1. 마이크로 서비스 아키텍처(MSA) 이해

1) MSA란?

(2) 마이크로 서비스 아키텍처(MicroService Architecture)란?

마이크로 서비스 아키텍처

- SOA에 근간을 두고 대용량 웹 서비스 개발에 맞게 변형된 아키텍처

① 서비스란 데이터부터 비즈니스 로직까지 독립적으로 상호 의존성 없이 개발된 컴포넌트 (수직적 분할)

② 서비스는 도메인(업무)의 경계에 따라 구분하는 것이 바람직함
예) 사용자관리, 상품관리, 주문관리

2) MSA 구조

(1) 마이크로 서비스 아키텍처(MSA) 구조

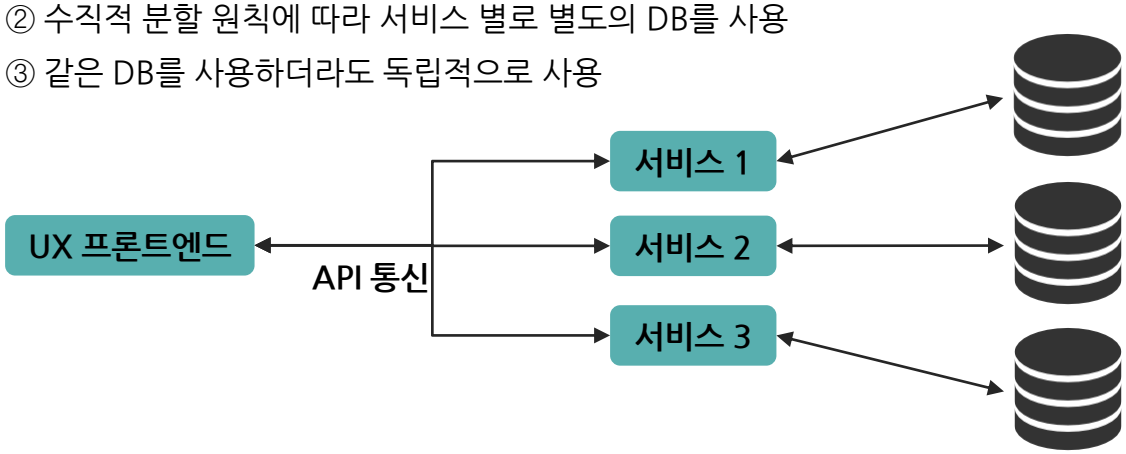
MSA

- 각 컴포넌트는 서비스라는 형태로 구현하고 API를 이용하여 타 서비스와 통신하는 구조

① 각 서비스가 독립적이므로 변경된 컴포넌트만 배포

② 수직적 분할 원칙에 따라 서비스 별로 별도의 DB를 사용

③ 같은 DB를 사용하더라도 독립적으로 사용



[모노리틱과 마이크로 서비스 아키텍처간 데이터 저장 구조 비교]

1. 마이크로 서비스 아키텍처(MSA) 이해

3) API Gateway

(1) API Gateway의 정의

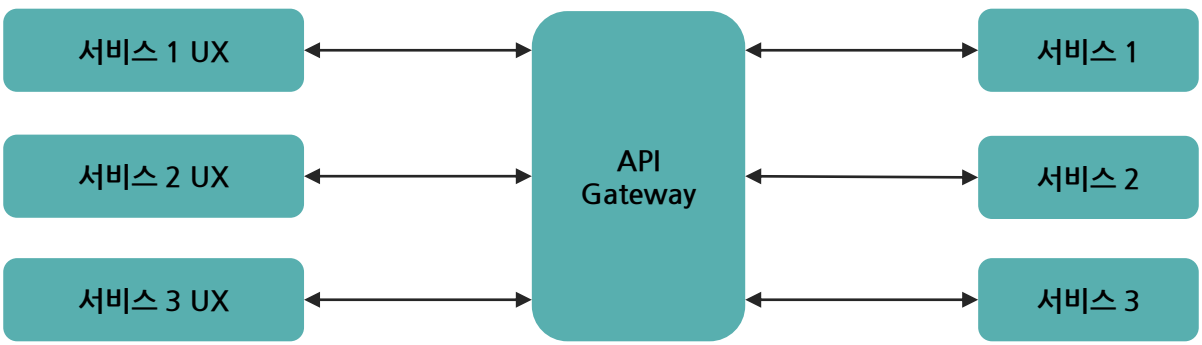
API Gateway

- 모든 API에 대한 엔드포인트를 통합하고 몇 가지 추가 기능을 제공하는 미들웨어로 SOA의 ESB(Enterprise Service Bus)의 경량화 버전

(2) API Gateway의 주요기능

① 엔드포인트 통합 및 토폴로지 정리

- API 엔드포인트(서버 URL) 문제(P2P형태의 통신)를 해결하기 위해 사용
- 중앙에 API Gateway를 서비스 버스로 사용하여 Hub & Spoke 방식으로 변환시켜 서비스간 호출을 단순화



[엔드포인트 통합 및 토폴로지 정리]

② 공통기능처리

- 여러 개의 서비스를 하나로 묶어 처리하는 개념
- 오픈API의 매시업(Mashup)과 같은 개념

1. 마이크로 서비스 아키텍처(MSA) 이해

3) API Gateway

(2) API Gateway의 주요기능

③ 공통기능처리

- 서비스 컴포넌트별 중복 개발해야 하는 기능을 API Gateway를 통해 공통기능으로 처리
- 자체 비즈니스 로직에만 집중하여 개발

④ 중재

- 서비스간 전송 메시지의 포맷을 변환하는 메시지 변환(Message Transformation), 프로토콜변환, 메시지 라우팅 등 처리 가능
- API Gateway에 부담이 될 수 있어서 높은 수준의 설계와 기술적인 노하우 필요

4) MSA의 특징 및 문제점

(1) 마이크로 서비스 아키텍처 특징

① 배포

- 각 서비스가 다른 서비스와 물리적으로 완벽하게 분리되기 때문에 변경된 서비스만 부분 재배포가 가능

② 확장성

- 서비스별 독립된 배포구조로 인해 확장성 증가
- 부하를 많이 받는 서비스 컴포넌트만 확장

③ 컨웨이의 법칙(Conway's Law)

- 소프트웨어의 구조는 그 소프트웨어를 만드는 조직의 구조와 일치
- 팀은 7~10명 구성, 각 컴포넌트를 팀에 배치
- 팀 간의 의존성을 제거해서 독립적으로 개발 수행이 가능하도록 조직구성

1. 마이크로 서비스 아키텍처(MSA) 이해

4) MSA의 특징 및 문제점

(2) MSA의 문제점

① 성능

- 서비스 간의 호출을 API 통신을 이용하기 때문에 값을 변환하는 마샬링 오버헤드와 메시지 전송 등으로 시간이 추가로 소요

② 메모리

- 서비스를 독립적으로 사용하기 때문에 그만큼 메모리 사용량이 늘어남
- 하드웨어 비용이 낮아지고, 캐싱 등으로 해결 가능

③ 테스팅

- 사용자 시나리오나 기능 테스트 시 여러 서비스에 걸쳐 테스트 해야 함
- 테스팅 환경 구축과 테스팅에 대한 복잡도 증가

④ 서비스간 트랜잭션 처리

- 모노리틱 아키텍처는 트랙잭션 처리가 용이하지만 API 기반의 여러 서비스를 하나의 트랜잭션으로 묶는 것은 불가능
- 애플리케이션 설계 단계에서 분산 트랜잭션 자체를 없애야 함 (MSA는 금융, 제조 서비스에 부적합)
- 트랜잭션을 구현해야하며 애플리케이션 수준에서 처리 (오류발생시 이를 보상하는 로직 구현)
- 복합 서비스(Composite Service)구현
  → 트랜잭션을 묶어야 하는 두 개의 시스템을 트랜잭션을 지원하는 네이트브 프로토콜을 이용해서 구현한 다음 이를 API로 노출하는 방법

2. 거버넌스 모델

1) 거버넌스 모델이란?

(1) 거버넌스 모델

거버넌스 모델

- **시스템을 개발하는 조직의 구조나 프로세스**

① 중앙 집중형 거버넌스 모델

- 중앙 집중화된 조직에서 표준화된 프로세스와 가이드를 기반으로 전체 팀을 운용하는 모델
- 유지 보수가 비교적 쉬우며 팀 간의 인원 교체가 편리
- 신기술을 도입하려면 모든 개발팀을 교육시키고 운영 준비를 해야 하기 때문에 기술에 대한 적용 민첩성이 떨어짐

② 분산형 거버넌스 모델

- 각 팀에 독립적인 프로세스와 기술 선택 권한을 주는 모델
- 유지 보수가 비교적 쉬우며 팀 간의 인원 교체가 편리
- 신기술을 도입하려면 모든 개발팀을 교육시키고 운영 준비를 해야 하기 때문에 기술에 대한 적용 민첩성이 떨어짐

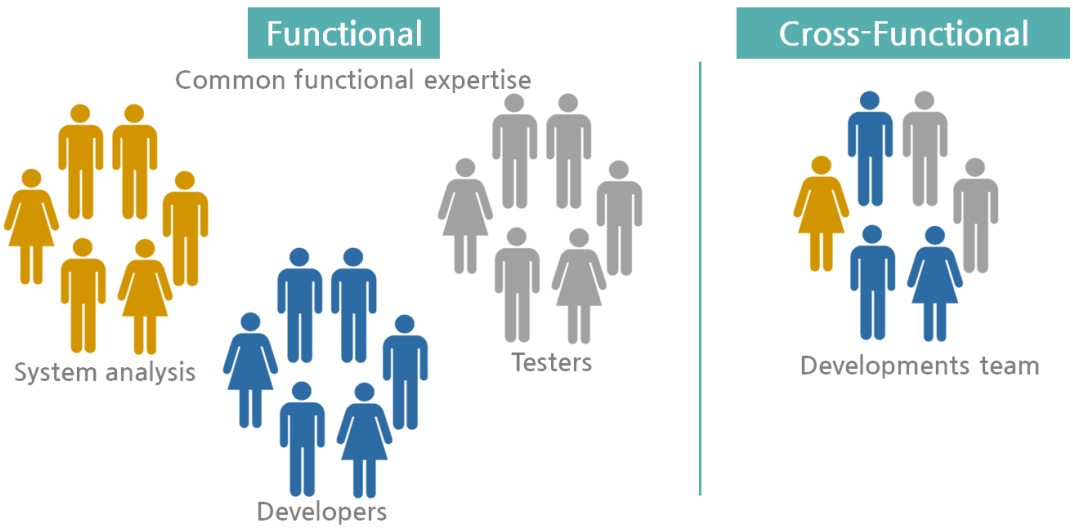
2. 거버넌스 모델

2) 분산형 거버넌스 모델의 특징

(1) 특징

① Cross Functional Team

- 기존 팀 모델 : 역할별로 팀을 구분
  → 인력운영 등 자원 관리에 유연성 부여
  → 팀간 커뮤니케이션에 많은 시간 소요
- 기획, UX, 개발, 인프라운영 등 소프트웨어 시스템을 개발하는데 필요한 모든 역할을 하나의 팀으로 구성하고 움직이는 모델
  → 서비스 기획부터 설계 개발 운영까지 다른 팀의 의존성 없이 운영 가능



② DevOps (Development + Operation)

- 개발과 운영을 하나의 조직에 합쳐놓은 구조
- 운영 시 발생하는 여러 문제점과 고객의 피드백을 빠르게 수용가능
- 좋은 모델이나 높은 수준의 팀 성숙도가 요구
  → 개발자가 인프라에 대한 설명 운영까지 담당하므로 업무 부담 증가

2. 거버넌스 모델

2) 분산형 거버넌스 모델의 특징

(1) 특징

③ Project vs. Product

- 팀 간의 인력이동이 발생하면 새로운 팀이 사용하는 표준과 프로세스에 대한 재교육이 필요해지기 때문에 팀의 영속성을 보장해주어야 함
- 기존 프로젝트는 역할 중심
  → 일정기간에 정해진 요구 사항을 구현하고, 구현 완료되면 프로젝트 구성원은 해체됨
- 분산형 거버넌스 모델은 Project 중심 보다는 Product(상품) 중심으로 팀을 운영
  → 팀은 상품에 대한 요구사항 정의, 개발 및 운영 등을 책임지고, 계속해서 상품을 개선하는 활동 지속

④ Self-organized Team

- 서비스(상품)를 기획하고, 개발, 운영하며 스스로 서비스를 발전시킬 수 있는 독립적인 수행 능력을 갖춘 팀으로 성장

⑤ Alignment

- 각 팀간의 커뮤니케이션 방법이나 프로세스 등 최소한의 표준과 기술 수준을 맞추는 과정
- 각 팀간의 역량의 차이로 개발 속도의 차이가 생길 수 있음

2. 거버넌스 모델

3) MSA 적용

(1) MSA 적용

① MSA

- 서비스의 재사용성
- 유연한 아키텍처
- 대용량 웹 서비스 지원
- 팀의 높은 성숙도 필요 (충분한 능력을 갖추지 못한 팀은 많은 시행 착오를 겪을 수 있음)

② 적용

- 모노리틱 시스템을 운영하면서 발생하는 문제점을 개선하기 위하여 마이크로 서비스 아키텍처 형태로 진화하는 것이 바람직함

3. DevOps

1) DevOps란?

(1) DevOps의 정의

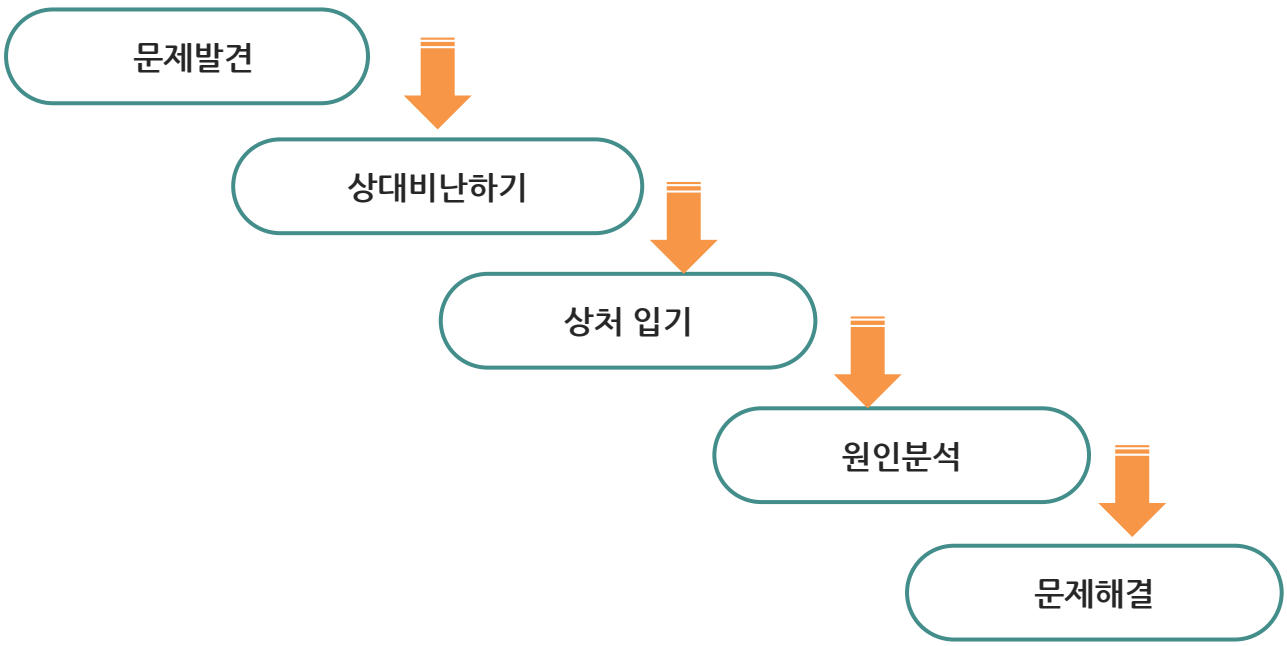
DevOps

- 개발팀에서 개발이 끝나면 시스템은 운영팀에 이관되어 운영팀이 해당 시스템을 배포하고 관리·운영
- 엔지니어가 프로그래밍과 빌드를 하고 직접 시스템 배포 및 서비스를 실행하고 사용자와 끊임없이 상호작용하면서 서비스를 개선해나가는 일련의 과정
- 협업중심의 개발 문화

(2) 분리 시 문제점

① 책임전가

- 문제 발생시 서로 책임을 전가하고 문제 해결은 지연 됨 (Fingerpointyness)



3. DevOps

1) DevOps란?

(2) 분리 시 문제점

② 운영이슈에 대한 전달

- 운영팀이 접수한 VoC(Voice of Customer)를 바탕으로 서비스 개선을 요청하지만 개발팀은 추가 업무로 간주

③ 비즈니스 변경

- 비즈니스 환경변화에 따른 새로운 요구사항이 추가되고, 이를 반영하여 추가적으로 시스템을 개발하고 배포해야 하므로 개발팀과 운영팀은 추가 업무로 간주

(3) 방법론

① 개발과 운영을 합친 방법론

- 서비스 요구사항의 신속한 반영
- 고객의 요구 사항에 민감한 소프트웨어 개발

(4) 등장배경

① 인터넷의 발전 : 지식 습득 채널의 다양화

② 오픈소스의 발전 : 수많은 개발자

③ 오픈소스 스티칭 : 오픈소스들의 조합

④ 좋은 도구들 : 개발, 빌드, 배포, 모니터링 등

⑤ 클라우드 등장 : 인프라 제공 서비스 활용

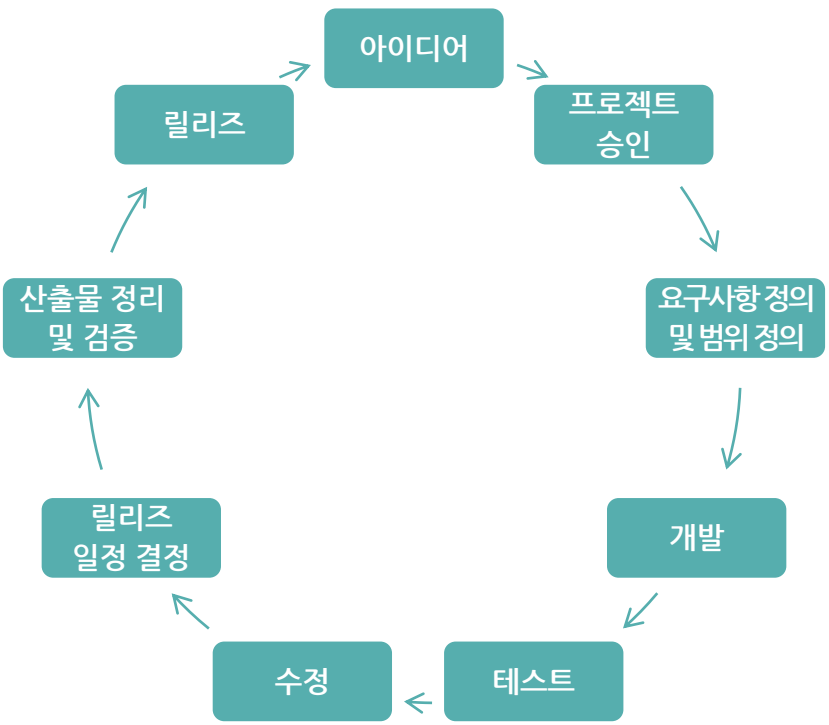
3. DevOps

1) DevOps란?

(5) DevOps 특징

- Cross Functional Team : 각기 다른 역할을 하는 팀원으로 구성
- Widely Shared Metric : 팀 전체가 공유한 서버스에 대한 공통지표
- Automating Repetitive Tasks : 반복적인 작업을 도구를 이용해 자동화
- Post-mortem : 장애 등 문제가 있을 때 처리 후 그 내용을 공유
- Regular Release : 짧은 주기의 정기적인 릴리즈를 통해 빠른 서비스 기능 개선과 고객의 Voc를 잘 반영

2) DevOps 기반의 개발 사이클



3. DevOps

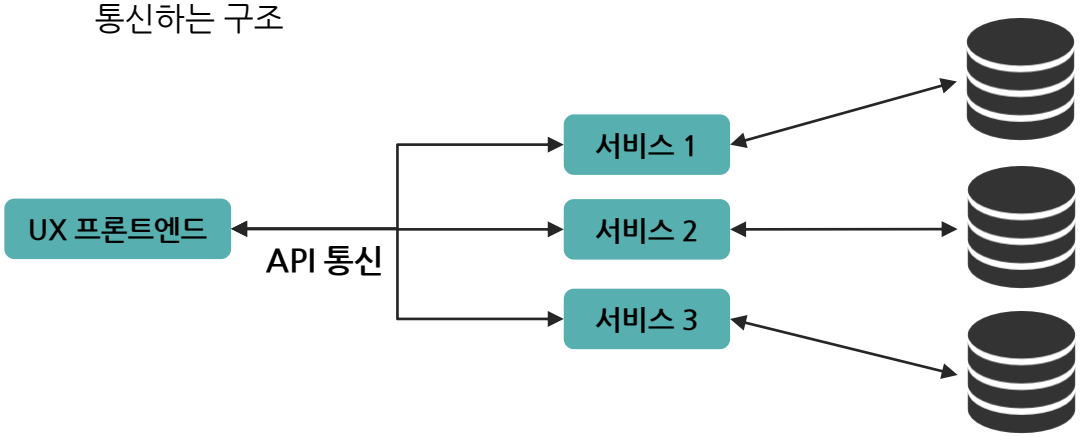
3) DevOps 개발자 필요 역량

- 기본적 개발에 필요한 코딩능력
- 다른 사람과 협업하고 의사소통 할 수 있는 능력
- 프로세스를 이해하고 재정의 할 수 있는 능력

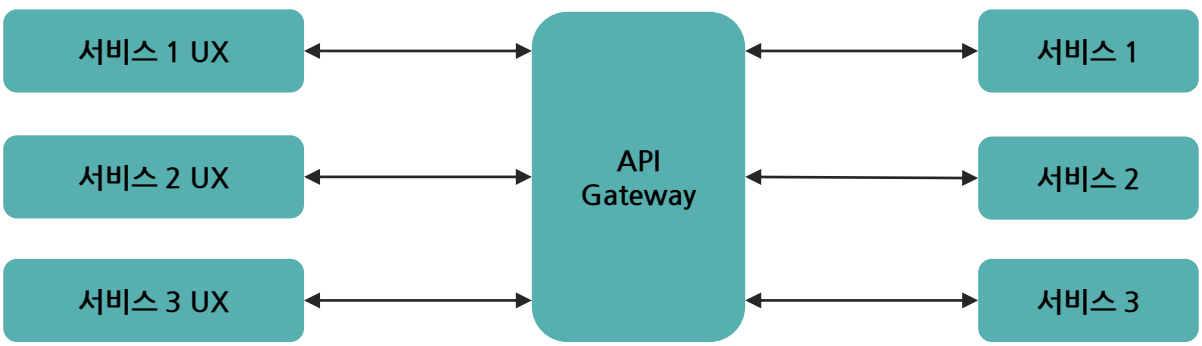
핵심정리

1. 마이크로 서비스 아키텍처(MSA)이해

- MSA란?
  - SOA에 근간을 두고 대용량 웹 서비스 개발에 맞게 변형된 아키텍처
  - 각 컴포넌트는 서비스라는 형태로 구현하고, API를 이용하여 타 서비스와 통신하는 구조



- API Gateway 개념
  - 모든 API에 대한 엔드포인트를 통합하고 몇 가지 추가 기능을 제공하는 미들웨어로 SOA의 ESB(Enterprise Service Bus)의 경량화 버전
  - API Gateway 특징 으로 엔드포인트 통합 및 토폴로지 정리, 오케스트레이션, 공통기능처리, 중재 가 있다.



핵심정리

2. 거버넌스 모델

- 각 팀에 독립적인 프로세스와 기술 선택 권한을 주는 모델 : 분산형 거버넌스 모델
- 거버넌스 모델 특징으로 Cross Functional Team, DevOps, Project vs. Product , Self-organized Team, Alignment 이 있음

3. DevOps

- 협업 중심의 개발 문화로 개발팀에서 개발이 끝나면 시스템은 운영팀에 이관되어 운영팀이 해당 시스템을 배포하고 관리·운영
- 엔지니어가 프로그래밍과 빌드를 하고 직접 시스템 배포 및 서비스를 실행하고 사용자와 끊임없이 상호작용하면서 서비스를 개선해나가는 일련의 과정
- DevOps 개발자는 기본적 개발에 필요한 코딩능력과 다른사람과 협업할 수 있는 능력 프로세스를 이해하고 재정의 할 수 있는 능력이 필요함