CONFLUENT

Data in Motion을 위한 이벤트 기반 마이크로서비스 아키텍처 소개

HyunSoo Kim

Senior Solutions Engineer, Korea SE Team Lead

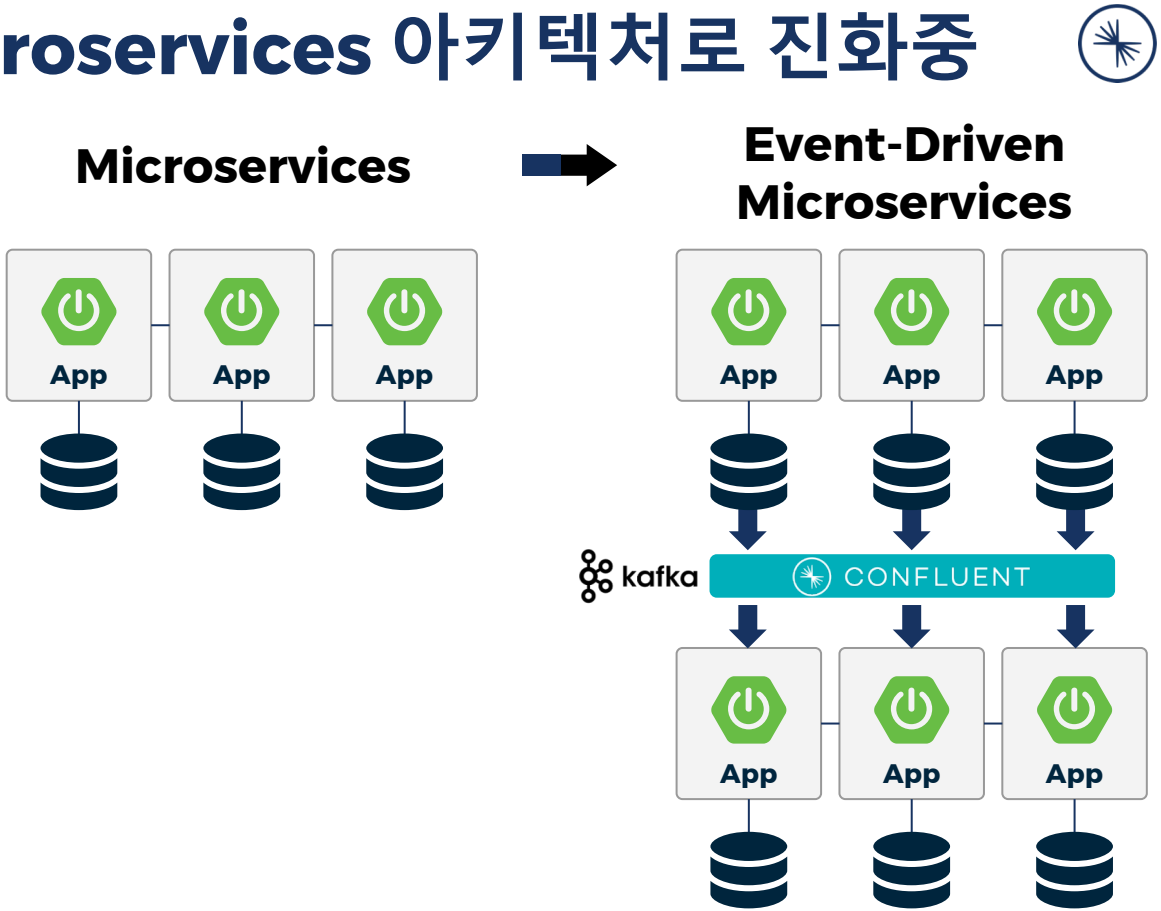
hkim@confluent.io

기업은 소프트웨어 사용자에서 벗어나서 소프트웨어가 되기 위해서 움직이고 있습니다

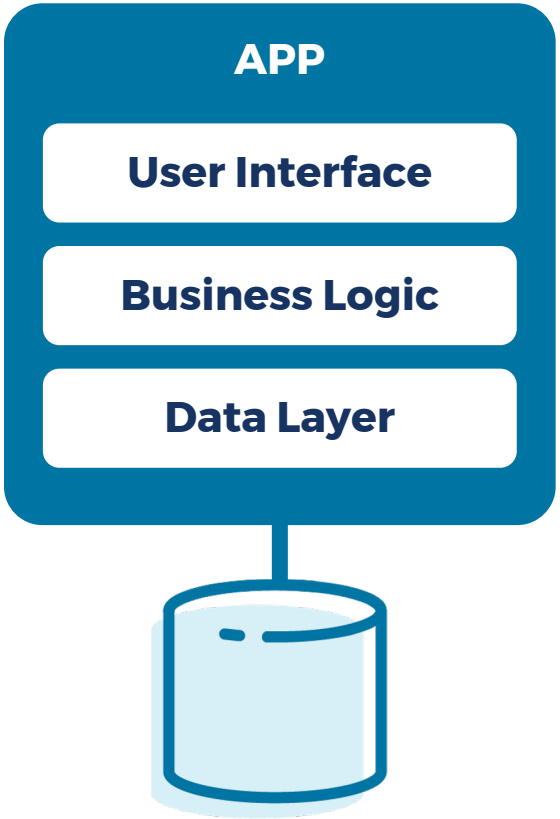
- 더 빠른 애플리케이션 개발 및 서비스 출시 시간 단축
- 팀이 독립적으로 구축할 수 있도록 지원
- 인프라 비용 및 복잡성 감소
- 서비스 간의 종속성 제거
- 더 빠르고 쉽게 서비스 확장

Application 아키텍처 관점에서의 변화를 살펴보면...

Event Driven Microservices 아키텍처로 진화중

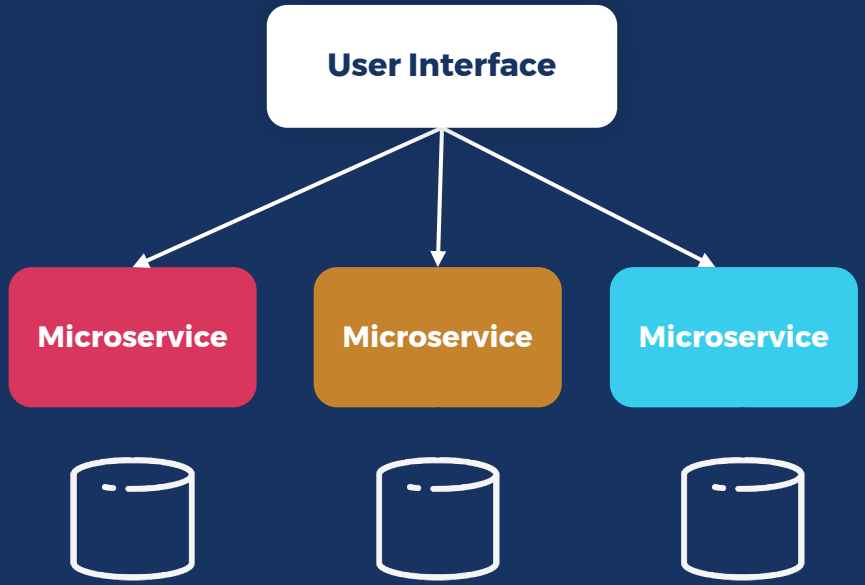


Monolith 애플리케이션



- 상호 의존적인 부분이 매우 많은 복잡한 애플리케이션
- 다른 서비스에 영향을 주지 않고 배포하기 어려움
- 열악한 개발자 경험 및 생산성 저하
- 서비스 출시 시간 지연 초래

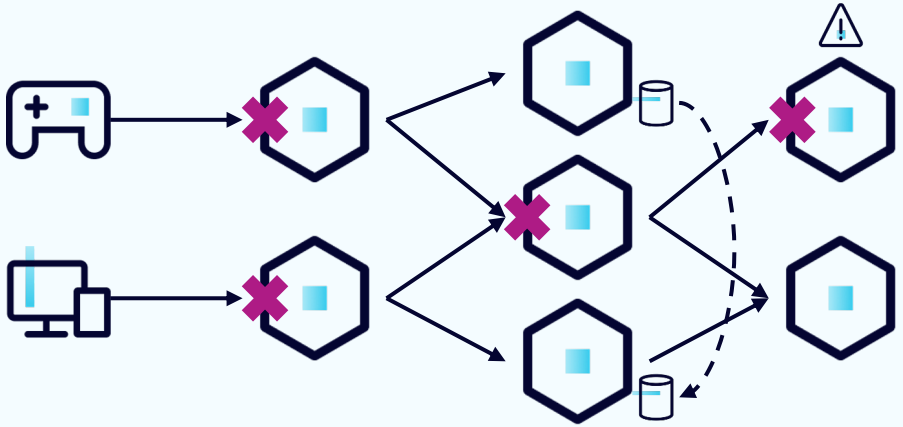
Microservice로의 진화



- 여러 개의 더 작은 단일 기능 애플리케이션
- 독립적으로 배포 및 업그레이드 가능
- 다른 프로그래밍 언어로 구축 가능
- 서비스별로 확장 및 축소
- 더 빠른 서비스 출시

기존 MSA 접근 방식의 문제점

Microservice Challenges - Orchestration



Independent Scalability (독립적인 확장성)

컴포넌트 단위로 배포 및 확장/축소 가능.
애플리케이션에 따라 개별의 라이프 사이클에 의해 서비스를 관리.
개별 확장/축소 가능 - 병목 현상이 되는 특정 서비스만의 확장에 의해 전체 처리량 향상.

Cascading Failure (연쇄 장애)

특정 서비스가 응답할 수 없게 되면 서비스에 요청하는 서비스가 응답할 수 없음.
1) 서비스 장애, 고부하로 인한 반응 속도 저하가 전체적인 정지/성능 저하로 이어짐.

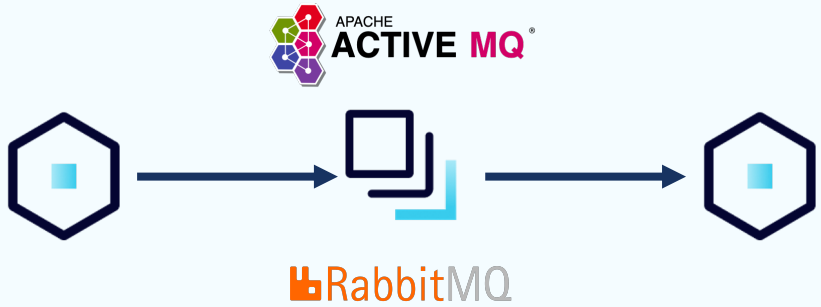
Data Consistency (데이터 일관성)

각 서비스마다 독립적인 데이터스토어가 있습니다. 서로 다른 서비스간에 데이터 무결성을 유지해야 함.
2) PC나 Saga와 같은 패턴에서는 데이터 동기 비용이 높고 처리도 복잡해짐
확장/축소하기 어려움

Microservice Challenges - Legacy Messaging

Asynchronous Communication (비동기 통신)

동기 통신 방식의 단점을 극복 가능.
마이크로서비스간에 비동기 방식의 통신을 사용하여 서비스간 종속성을 없앰.
서비스 연쇄 장애 문제를 해결 가능.



Lack of Message Persistence (메시지 유지 취약)

Queue에 저장된 데이터를 가져가면 그 데이터를 지움.
데이터 유실의 가능성이 높음.
1) 그 데이터를 필요로 하는 다른 서비스가 있으면 데이터를 다른 Queue로 복제해야 하는 문제 발생

Low Performance/High Latency (낮은 성능/높은 지연시간)

대용량의 데이터(BigData) 전송/처리를 위한 처리량 제공 불가능 - 지연시간 또한 높음.
2) 이벤트 스트리밍 처리 불가능 - 실시간 분석 불가능

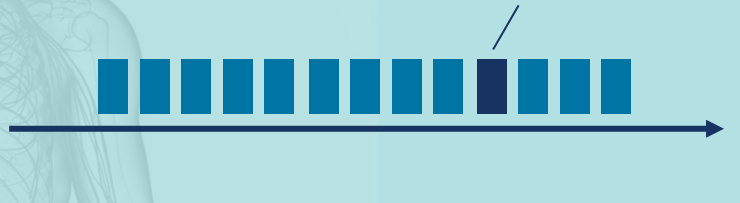
이 문제들을 해결할 수 있는 방법이

“Kafka 중심”의 이벤트 기반 마이크로서비스 아키텍처 !!!

‘Event’는 비즈니스에서 일어나는 일입니다

교통

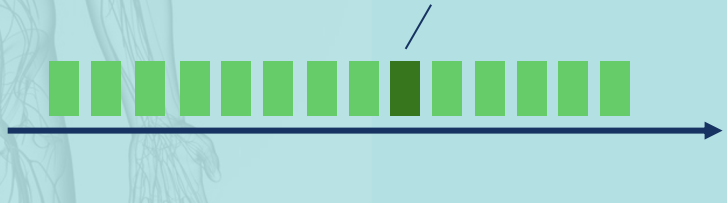
Carol의 자동차에 있는 TPMS 센서가 오전 5시 11분에 낮은 타이어 압력을 감지했습니다.



Kafka

뱅킹

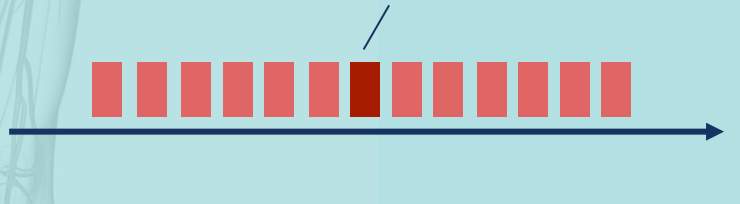
Alice는 금요일 오후 7시 34 분에 Bob에게 250달러를 보냈습니다.



Kafka

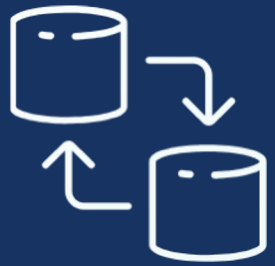
소매

Sabine의 Fujifilm 카메라 주문은 오전 9시 10분에 배송 되었습니다.



Kafka

Apache Kafka의 특징



01
이벤트 스트림을
Publish & Subscribe

02
이벤트 스트림을
Store

03
이벤트 스트림을
Process & Analyze

Apache Kafka의 성능

Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines)

Jay Kreps

April 27, 2014

- Intel Xeon 2.5 GHz processor(6 코어)
- 7200 RPM SATA 드라이버 6 개 : RAID 아닌 JBOD 로 구성
- 32 GB of RAM
- 1 Gb Ethernet
- 상기 스펙의 총 6 대 HW 사용

  3 대 – Zookeeper/Load Generator, 3 대 – Kafka Broker
- Three Producer, 3x async replication :

  2,024,032 records/sec (193.0 MB/sec)

https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

Apache Kafka의 역사

- 하루 4.5 조 개 이상의 이벤트 스트림 처리
- 하루 3,000 억 개 이상의 사용자 관련 이벤트 스트림 처리
- 기존의 Messaging Platform(예, MQ)로 처리 불가능
- 이벤트 스트림 처리를 위해 개발
- 2010년에 Apache Software Foundation 에 기부되어 오픈소스화

Apache Kafka의 등장

- 2012년 Apache Incubator 과정을 벗어나 최상위 프로젝트가 됨
- Fortune 100 기업 중 80% 이상이 Apache Kafka를 사용[1)]

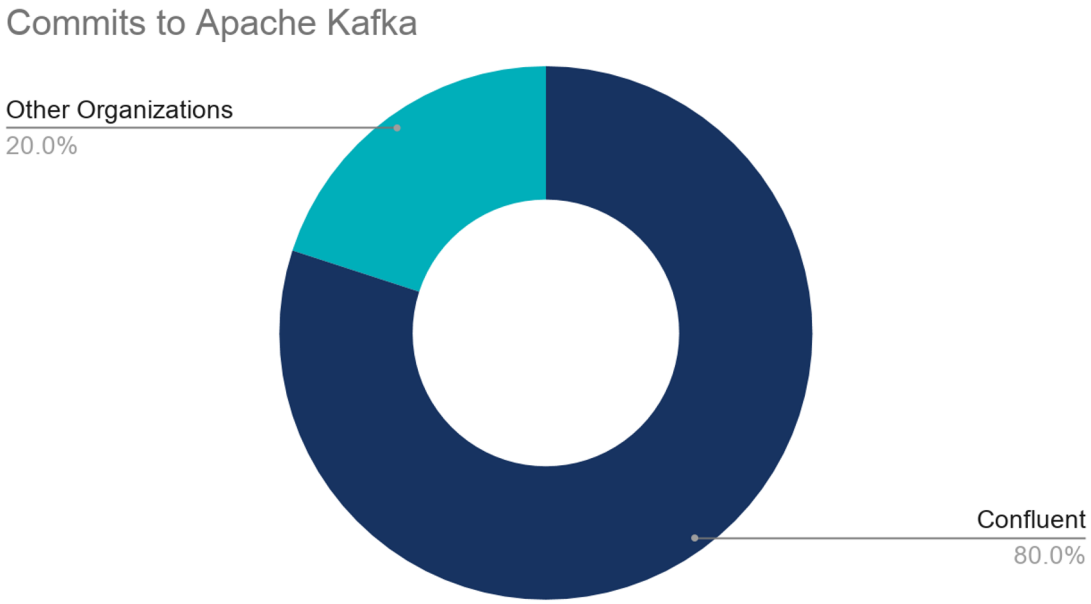
1) https://kafka.apache.org/

Apache Kafka 원조 회사

CONFLUENT

- Kafka 창시자(Jay Kreps)가 만든 회사
- 2014년 설립
- Mountain View, CA
- 2021년 6월 IPO (기업공개, Nasdaq 상장)

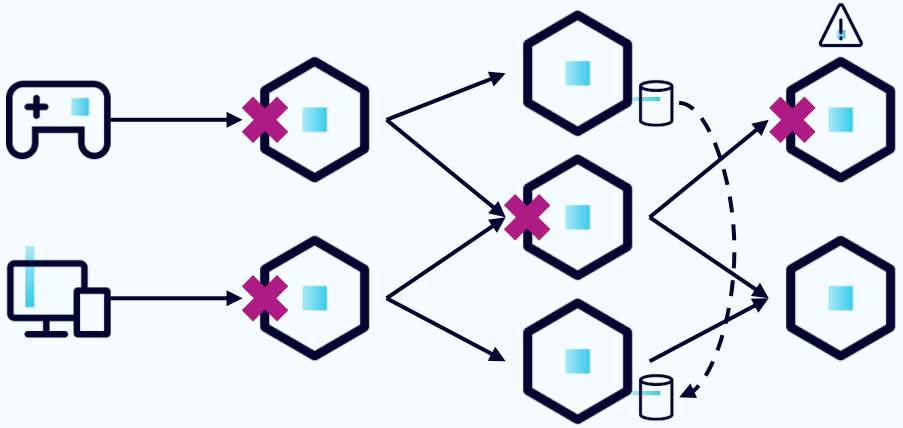
Confluent는 세계 최고의 Kafka 전문가로부터 커미터 중심의 전문성을 제공합니다



Confluent는 Apache Kafka에 대한 모든 Commit의 80 % 이상을 담당합니다

기존 MSA 접근 방식의 문제점과 해결방안

Microservice Challenges - Orchestration



Independent Scalability (독립적인 확장성)

컴포넌트 단위로 배포 및 확장/축소 가능.
애플리케이션에 따라 개별의 라이프 사이클에 의해 서비스를 관리.
개별 확장/축소 가능 - 병목 현상이 되는 특정 서비스만의 확장에 의해 전체 처리량 향상.

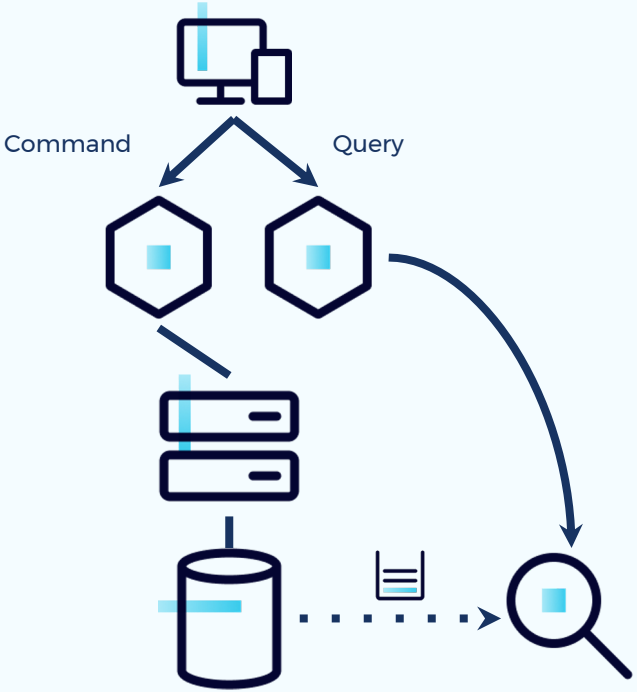
Cascading Failure (연쇄 장애)

특정 서비스가 응답할 수 없게 되면 서비스에 요청하는 서비스가 응답할 수 없음.
1) 서비스 장애, 고부하로 인한 반응 속도 저하가 전체적인 정지/성능 저하로 이어짐.

Data Consistency (데이터 일관성)

각 서비스마다 독립적인 데이터스토어가 있습니다. 서로 다른 서비스간에 데이터 무결성을 유지해야 함.
2) PC나 Saga와 같은 패턴에서는 데이터 동기 비용이 높고 처리도 복잡해짐
확장/축소하기 어려움

CQRS and Change Data Capture - Consistent Data



CQRS

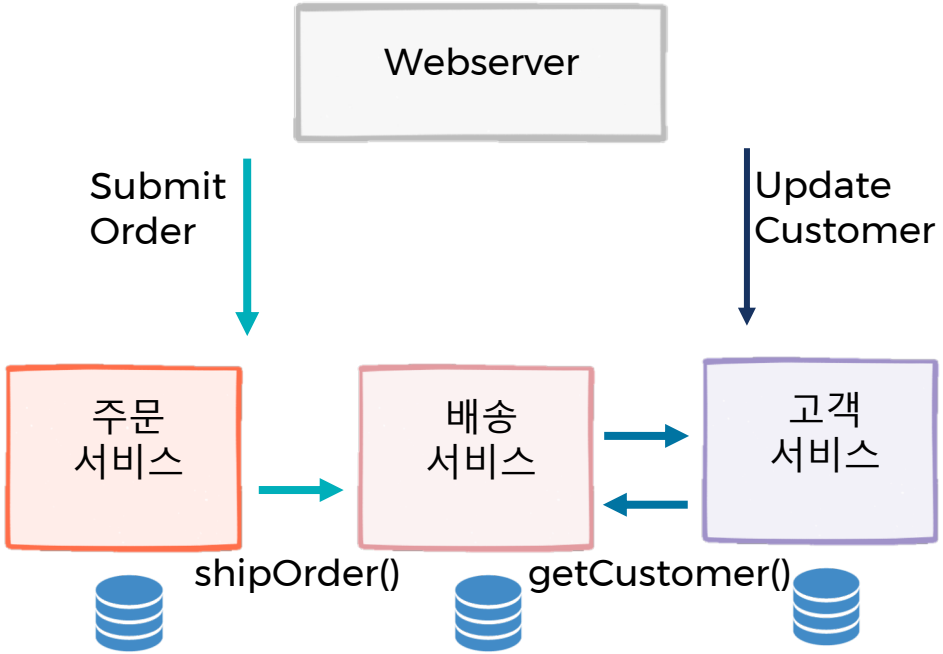
Command Query Responsibility Segregation - CRUD (Command) 및 검색 (Query) 역할과 데이터 저장소를 분리하는 방법. 기존 시스템에 영향을 최소화하면서, 병목 현상이 되기 쉬운 검색 기능을 다른 서비스로서 분리.

CDC(Change Data Capture)

데이터베이스의 변경사항을 추출/이벤트화해 제휴하는 것으로, 다른 데이터 스토어간의 데이터 무결성을 비동기로 복제하는 기법. CDC 이벤트를 유실하지 않고, 순서대로 동기할 필요가 있다.

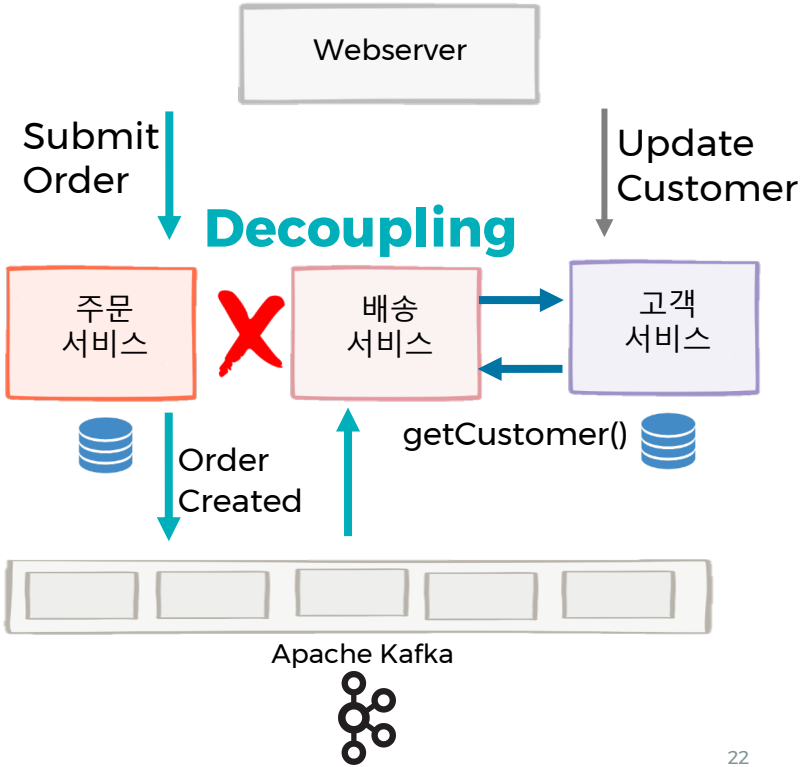
Buying an iPad (with REST)

- 주문 서비스는 배송 서비스를 호출하여 물건을 배송하도록 지시
- 배송 서비스는 배송할 주소를 조회 (고객 서비스에서)



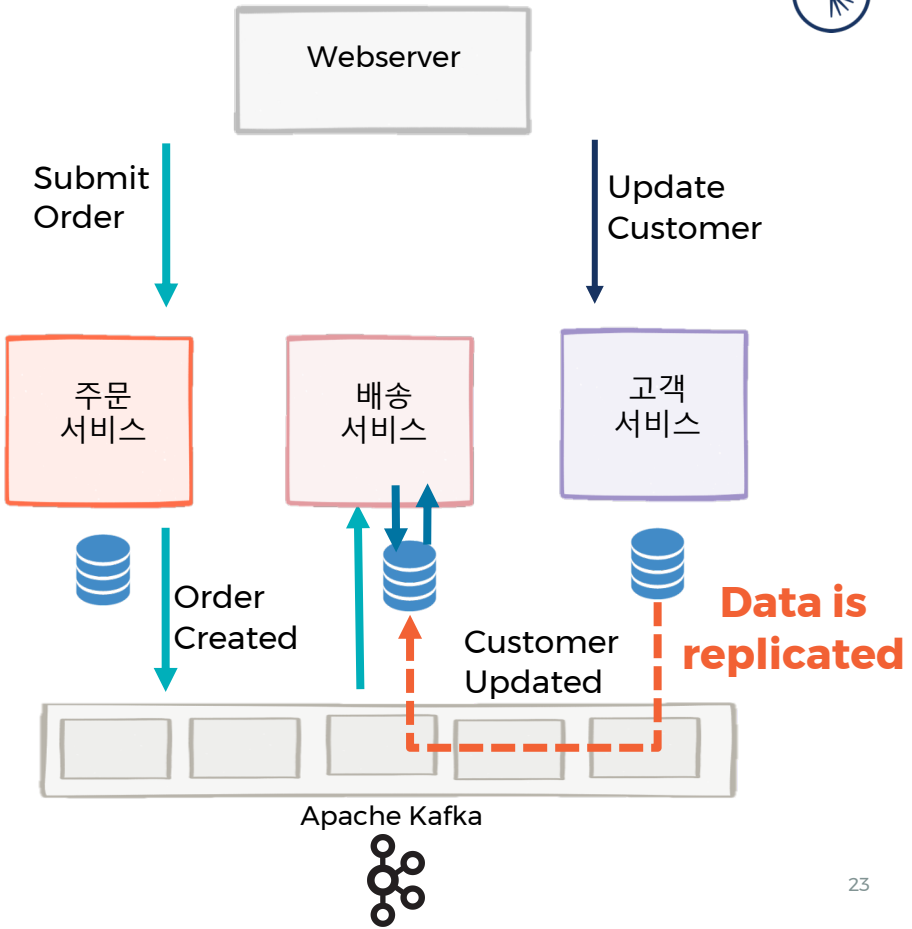
서비스 간 이벤트 기반 비동기 메시징

- 주문 서비스는 더 이상 배송 서비스(또는 기타 서비스)에 대해 알지 못함
- 주문 서비스는 주문 이벤트를 생성 및 메시지 Broker로 전송함



데이터베이스 간 CDC 기반의 데이터 복제

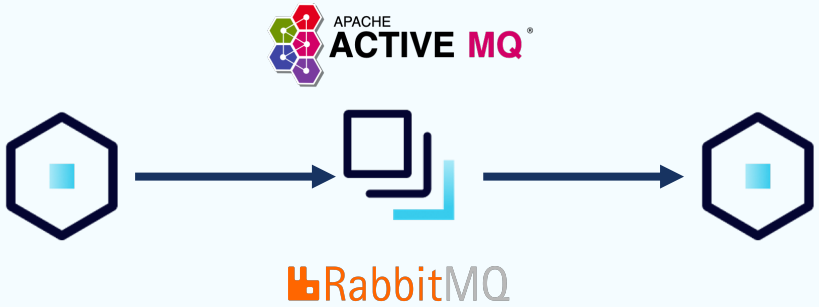
- 고객 서비스로의 직접 호출이 없음
- 대신, 고객 데이터 변경시 CDC로 추출되어 이벤트로 전송되어 배송 서비스의 DB에 복제되며 로컬에서 쿼리



Microservice Challenges - Legacy Messaging

Asynchronous Communication (비동기 통신)

동기 통신 방식의 단점을 극복 가능.
마이크로서비스간에 비동기 방식의 통신을 사용하여 서비스간 종속성을 없앰.
서비스 연쇄 장애 문제를 해결 가능.



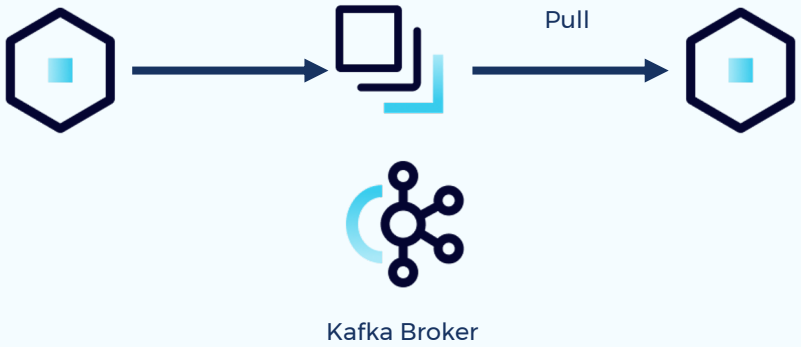
Lack of Message Persistence (메시지 유지 취약)

Queue에 저장된 데이터를 가져가면 그 데이터를 지움.
데이터 유실의 가능성이 높음.
1) 그 데이터를 필요로 하는 다른 서비스가 있으면 데이터를 다른 Queue로 복제해야 하는 문제 발생

Low Performance/High Latency (낮은 성능/높은 지연시간)

대용량의 데이터(BigData) 전송/처리를 위한 처리량 제공 불가능 - 지연시간 또한 높음.
2) 이벤트 스트리밍 처리 불가능 - 실시간 분석 불가능

Kafka-Based Durable Event Driven Architecture



Pull Model

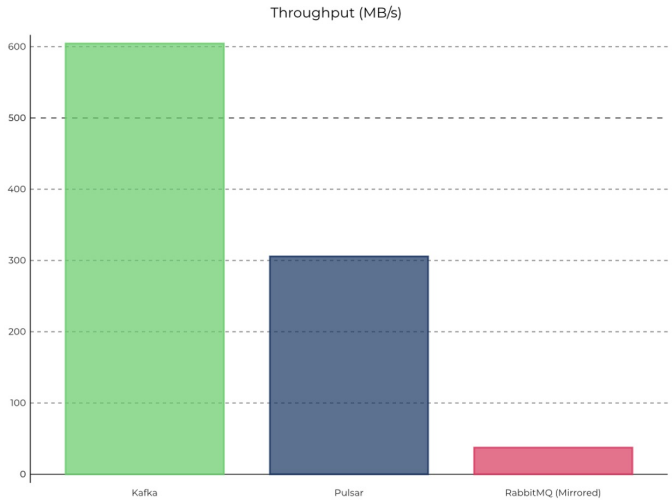
Pull 모델이기 때문에 백 프레셔가 불필요 - Consumer를 단순하게 추가하는 것으로, 다른 영향을 주지 않고 개별적으로 대응. 또한 일반적인 Message Broker와 달리 같은 스트림의 Consumer가 증가해도 Broker의 부하에 영향을 미치지 않는다.

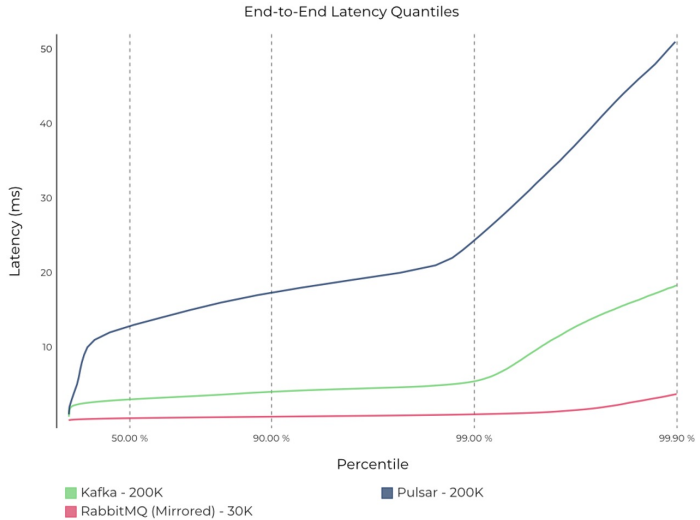
Durable Streams

Kafka Broker가 정해진 보관기간 동안 스트림 데이터를 유지할 수 있기 때문에 데이터의 유실을 일으키지 않는다. 저장된 데이터는 3중화이상으로 안전하게 보관된다.

Kafka vs RabbitMQ

Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?[1)]





1) [Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?]

Kafka vs RabbitMQ

Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?[1)]

	Kafka	Pulsar	RabbitMQ (Mirrored)
Peak Throughput (MB/s)	605 MB/s	305 MB/s	38 MB/s
P99 Latency (ms)	5 ms (200 MB/s load)	25 ms (200 MB/s load)	1ms* (reduced 30 MB/s load)

*RabbitMQ의 Latency(지연 시간)은 30MB/s보다 높은 처리량에서 크게 저하됩니다. 또한 미러링의 영향은 더 높은 처리량에서 더 커지며, 미러링 없이 클래식 queue만 사용하면 더 나은 Latency를 얻을 수는 있으나 데이터 유실 발생 가능성이 높습니다.

1) Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?

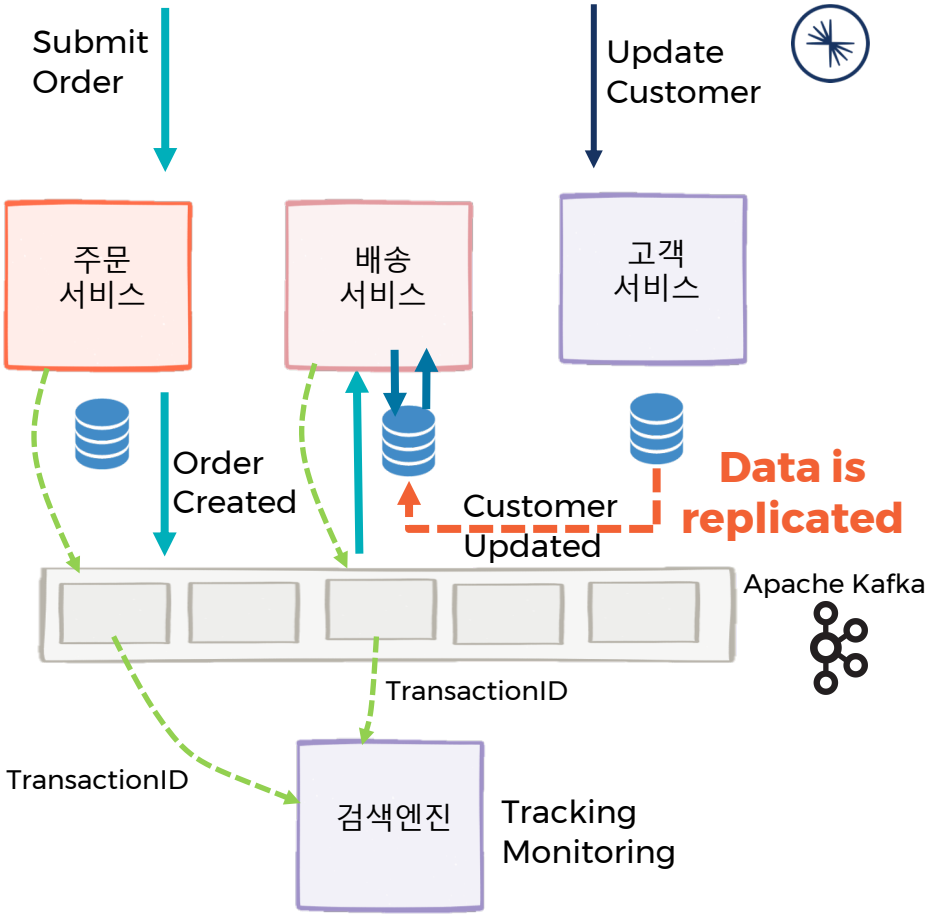
많은 고객들이 Apache Kafka를 사용하여 이벤트 기반 마이크로서비스 아키텍처를 이미 적용

- 마이크로서비스 Producer와 Consumer의 완전한 분리
- 복제를 통한 메시지 내구성 강화
- 메시지 재처리를 위한 재생(replay) 기능
- 메시지 형식 검증

- CDC 기반 DB 동기화
- 추가 오버헤드 없이 새로운 서비스를 쉽고 빠르게 추가
- 전송된 모든 메시지에 대한 중앙 집중식 보안 제어
- 메시지 형식 호환성
- 서비스 Tracking/Monitoring

서비스 Tracking

- 서비스 처리 Tracking/Monitoring
- 안전한 데이터 보관
- 검색엔진 부하 경감



Use Case – eCommerce

이벤트 기반 마이크로서비스 전환

Challenge

15년 이상 사용해 온 여러 전자상거래 플랫폼에서 지속적인 성장을 견인

Solution

Confluent를 사용하여 모놀리식 아키텍처에서 마이크로서비스 아키텍처로 마이그레이션함으로써 독립성, 유연성 및 확장성 개선

Result

- 간소화된 유지 관리 및 운영으로 생산성 향상
- 안정적이고 처리량이 많은 스트리밍
- 신속한 지원
- 시장 출시 시간 단축

"우리는 오픈 소스 Kafka를 직접 실행한 경험이 있지만 대규모 배포에서 발생하는 문제 해결을 지원하는 신뢰할 수 있는 파트너가 필요하다는 것을 깨달았습니다. Confluent는 이러한 지원을 제공할 수 있는 현지 사무소를 보유하고 있었을 뿐만 아니라 우리가 미션 크리티컬 전자상거래 플랫폼에 원했던 가장 잘 알려져 있고 검증된 솔루션을 보유하고 있었습니다."

— 이베이코리아, 이홍우 매니저

Summary

Why Apache Kafka?

개발 속도 및 출시 시간 향상

마이크로서비스 Producer와 Consumer를 완전히 분리함으로써 한 번 Publish하고 많이 Consume하며 언제든지 이벤트를 Replay

레거시 메시징 시스템의 현대화 및 오프로드

비용을 절감하고 기존 Messaging Queue 및 메인프레임에서 이벤트 스트리밍 플랫폼으로 데이터 마이그레이션

이벤트 기반 마이크로 서비스 구축

Central Point of Failure 가 없는 수천 개의 마이크로서비스를 확장하면서 수백만 개의 메시지 전달

서비스 간 종속성 제거 및 스키마 호환

스키마 유효성 검사로 스키마 호환성, 버전 제어 및 품질 보증을 유지하여 서비스 간 종속성을 제거

손쉬운 재생을 위해 단일 플랫폼에서 상태 유지

간편한 Replay을 위해 동기화된 중앙 플랫폼에서 상태를 유지하여 상황별 마이크로서비스를 구축

Why Confluent?

이벤트 스트리밍 리더쉽

성공을 보장하기 위해 이벤트 스트리밍 분야에서 검증된 리더와 협력 필수

완전한 이벤트 스트리밍 플랫폼

업계에서 가장 완벽한 이벤트 스트리밍 플랫폼 활용

Committer 중심의 전문성

진정한 원조 Kafka 전문가의 기술 지원을 통해 위험 및 다운 타임 최소화

선택의 자유

어디에나 배포하여 비즈니스에 적합한 데이터 아키텍처 구축

Confluent Solutions

Cloud-Native SaaS

Confluent Cloud

클라우드를 위해 재 설계된

Cloud-Native Apache Kafka

주요 Public Cloud에서 사용 가능

Self-Managed Software

Confluent Platform

온프레미스 설치용

Enterprise Apache Kafka

모든 플랫폼, 온 프레미스 또는 클라우드에 배포


CONFLUENT