[MSA] 이벤트 기반 아키텍처

Created	@2021년 11월 23일
Tags	micro service
스터디	개인

EDA(Event Driven Architecture)

이벤트

- 시스템에서 상태의 변화 또는 사건의 발생을 의미
- 시스템의 내부(마이크로 서비스)에서 발생될 수있고 외부(사용자)로부터 발생될 수 있다.

작동 방식

- 이벤트 생성자와 소비자로 구분
- 생성자는 이벤트를 감지해서 메시지로 해당 이벤트를 나타냄
- 생성자는 소비자와 이벤트 결과를 알 수 없음.
- **이벤트 처리 플랫폼이 이벤트를 비동기식으로 처리하는 채널을 통해 소비자로 전송**

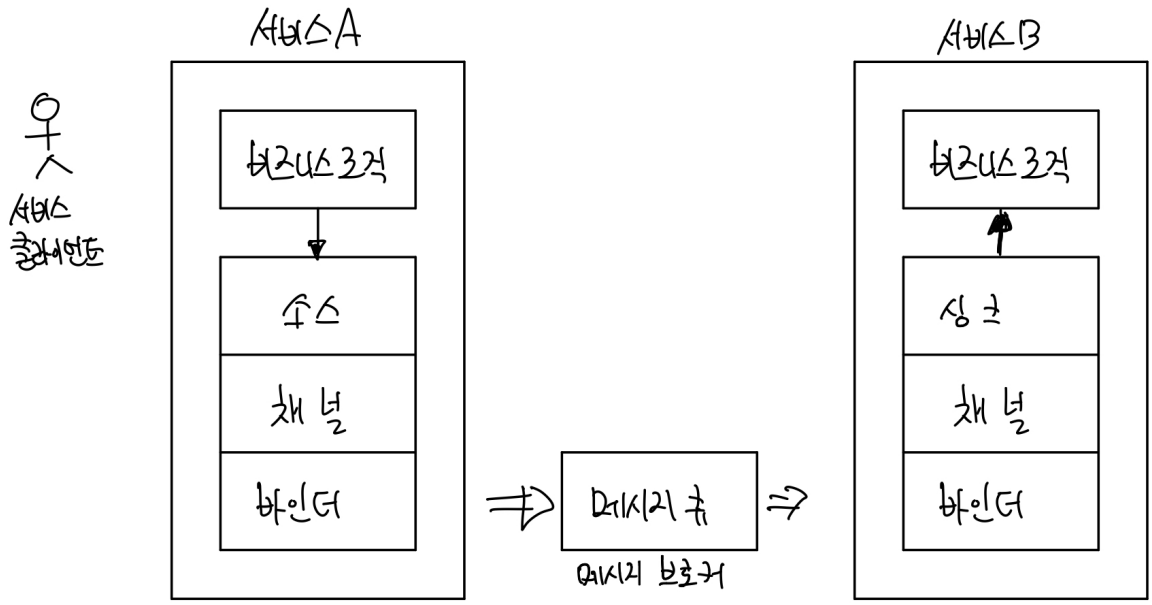
장점

- **약결합**: 서비스가 소유한 데이터를 직접 관리하는 엔드포인트만 노출함으로써 의존성 최소화
- **내구성**: 메시지 큐를 사용하여 소비자가 다운되어도 큐에 기록됨, 생성자는 계속 메시지 생성 가능
- **확장성**: 생성자는 소비자의 응답을 기다릴 필요 없음. 소비자가 느리다면 수평적 확장 가능.
- **유연성**: 생성자는 소비자를 알 수 없기 때문에 새로운 소비자 추가가 쉬움

단점

- 메시지 처리 의미론: 메시지 처리시에서 예외 상황 처리에 대해 설계해야한다.
- 메시지 가시성: 메시지를 생성한 트렌잭션을 추적해야한다.

Spring cloud stream



용어

- **소스(source)**
  - 서비스가 메시지를 발행할 준비가 되면 소스를 사용해 메시지 발행
  - 메시지를 표헌하는 POJO(Plain Old Java Object)를 전달 받는 인터페이스
  - 메시지를 받아서 직렬화(기본적으로 JSON)하고 메시지를 채널로 발행
- **채널(channel)**
  - 생산자와 소비자가 메시지를 발행하거나 소비한 후 메시지를 보관할 큐를 추상화한 것
- **바인더(binder)**
  - 특정 메시지 플랫폼과 통신
  - 스프링 클라우드 스트림의 바인더를 사용하면 플랫폼마다 별도의 api를 제공하지 않고도 메시징 사용 가능
- **싱크(sink)**
  - 싱크를 사용해서 큐에서 메시지를 받음.
  - 메시지 수신을 위해서 채널을 수신대기하고 메시지를 다시 POJO로 역직렬화

처리 과정

1. 서비스 클라이언트는 서비스를 호출, 비즈니스 로직에서 서비스A의 상태 변경
2. 소스는 메시지를 발행
3. 메시지가 채널로 발행
4. 바인더는 특정 하부 메시징 시스템과 통신
5. 메시지 브로커(메시지 큐)에 메시지 적재
6. 서비스 B가 메시지를 가져와서 처리

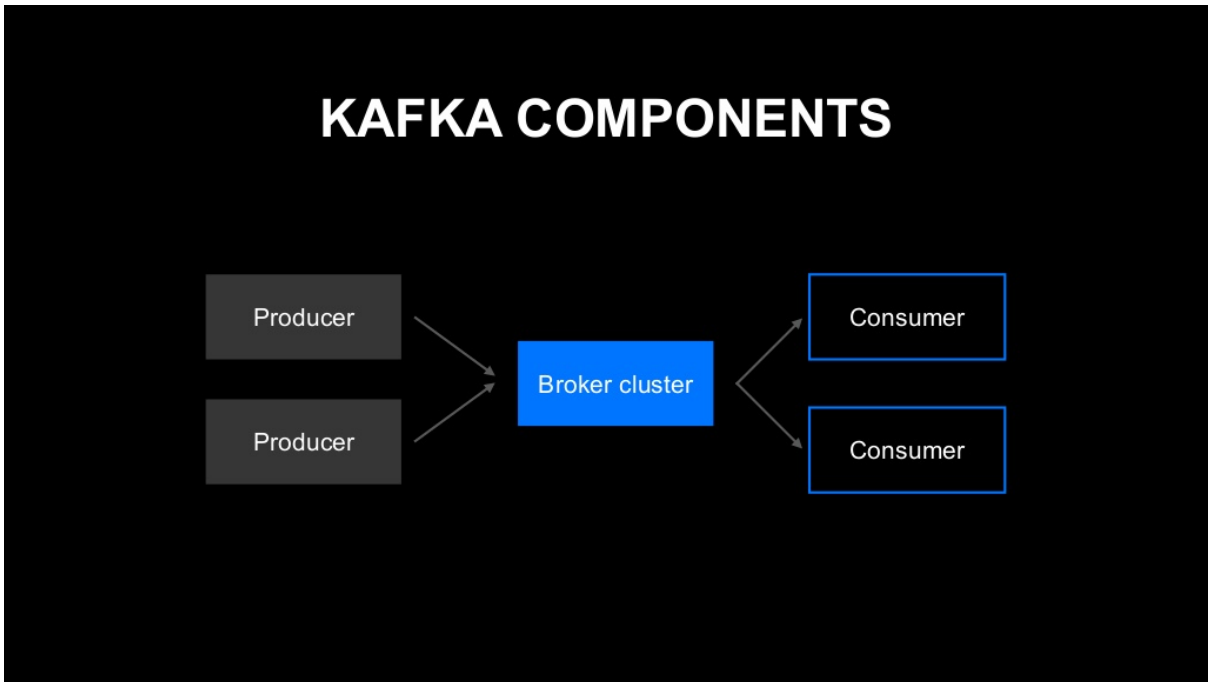
활용 방안

1. (분산) 캐싱
2. 채팅 서비스

Apache Kafka

- 쓰트리밍 데이터를 다루기 위한 미들웨어와 주변 생태계를 의미
- 높은 확장성(saclability), 가용성(avaliability)
- 데이터 영속성(persistency): kafka에 입력되면서 데이터 영속성
- Pub/Sub 모델: pulish/subscribe(생산자/소비자)를 통한 데이터 분포 지원

kafka의 3가지 컴포넌트



- **producer**: kafka에 데이터를 입력하는 클라이언트
- **broker cluster**: 여러 노드로 구서되는 클러스터로 `topic`이라는 데이터 관리 유닛을 호스팅
- **consumer**: kafka에서 데이터를 가져오는 클라이언트

kafka 사용 예시

1. 분산 큐잉 시스템
    - 특정 서비스에서 자리하는데 자원을 많이 사용해야하는 경우

      → 다른 프로세스에서 작동중인 백그라운드의 프로세서에 요청하기 위한 큐로 사용
2. 데이터 허브
    - 데이터 업데이트가 발생한 경우

      → 해당 데이터를 사용하는 다른 서비스에 전파하기 위한 허브로 사용

클러스터에 데이터를 집중 시키는 이유

1. 데이터 허브
    - 서비스가 해당 데이터를 쉽게 찾을 수 있다.
    - 서비스에서 데이터에 접근하는 수단을 통일시킬 수 있다.
    - 아키텍처를 단순하게 유지할 수 있다.
2. 운영 효율
    - 서비스와 시스템이 하나의 클러스터를 사용하면 엔지니어링 자원을 해달 클러스터에 집중 가능

      → 신뢰성과 성능 향상 가능

높인 신뢰성과 성능을 위한 조건

1. 작업 부하(workload)에서 보호 → 클러스터 공격과 같은 작업 부하가 걸리면 연관된 서비스에 영향
2. 생산자(클라이언트) 추적 가능 → 디버깅을 위해서 클라이언트 특정 가능해야함
3. 클라이언트(생산자,소비자) 사이 작업 부하 격리 → 다른 클라이언트의 작업 부하로 다른 클라이언트가 느려지면 안됨

추가 공부 필요

- Kafka는 자체 캐시 레이어가 없어서 OS별로 제공되는 페이지 캐시에 의존
  - 대용량 데이터가 애플리케이션의 메모리와 OS 페이지 캐시에 중복으로 캐시되는 것을 방지
  - kafka 클라이언트에는 배칭(batching) 기능이 기본적으로 지원

    → 여러개의 레코드를 하나의 큰 요청으로 묶어서 클러스터에 전달

    → 데이터 양과 레코드 수가 늘어나도 요청 수 증가 억제 가능 → 요청 별로 발생하는 오버헤드 방지