< Lambda architecture + 구성요소 / use case >

1. 참조: AWS Summit Seoul 2016 - Lambda를 활용한 서버없는 아키텍쳐 구현하기

[https://www.youtube.com/watch?v=trAtxluzj-g](https://www.youtube.com/watch?v=trAtxluzj-g )

거대한 리테일 회사가 데이터를 저장하기 위해 거대한 상용 DBMS를 사용

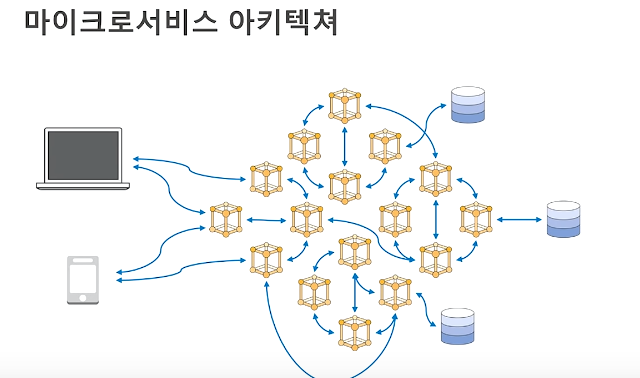
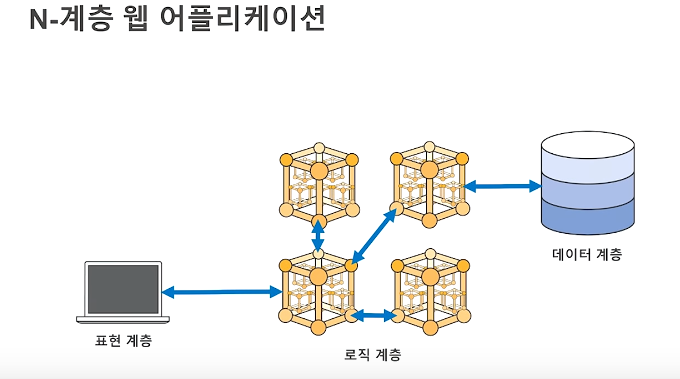
그 데이터베이스 안에 재고, 물류, 고객, 장바구니 등의 모든 정보가 들어있음

그 데이터베이스 시스템에 문제가 생긴다면?

-> 파급되는 수많은 피해(모든 것을 한 곳에, 그리고 그 한 곳의 실패 / 늘어나는 작업시간 / 운영의 어려움)

-> 길어지는 변경 주기(파격적인 사업진행 불가능 / 준비에 들어가는 수많은 시간)

서비스 지향 아키텍처 SOA (각각의 서비스별로 인터페이스를 갖고, 서비스끼리 통신하는 아키텍처)

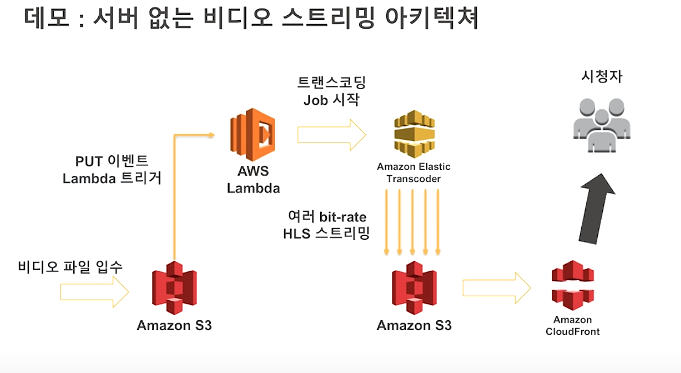
-> 서비스를 쪼개기 (N-계층 웹 어플리케이션 / 마이크로서비스 아키텍처 – 서비스를 잘게 쪼개서 서로 데이터를 공유하지 않는 단계까지 나누는 것, 작은 DB의 문제가 전체 시스템의 문제로 이어지지 않음) 

-> 하지만 여전히 공유되는 의존관계가 있다 -> 서버를 여전히 사용해야 함(서버의 성능, 문제, 용량, 변경관리…)

-> 새로운 서비스를 만들 때 여전히 서버에 의존 -> 서버 없는 아키텍처

AWS Lambda : 서버 없는, 이벤트 처리 방식의 컴퓨팅 서비스, 서버 없는 마이크로 서비스 / 실시간 데이터 처리와 유연한 백엔드 서비스 구축을 손쉽게 수행

Lambda 동작원리 : 1. 코드 업로드 2. 트리거 3. 람다실행 4. 사용요금



비디오 파일이 s3로 떨어지면 (s3는 파일이 떨어지면 이벤트가 생김) 그 이벤트를 받아서 람다를 트리거 한다.

람다가 트랜스코딩을 시작을 해서 트랜스코딩된 데이터가 s3에 떨어지면

클라우드 프론트를 통해서 시청자들한테 직접 전송

람다 사용 예제

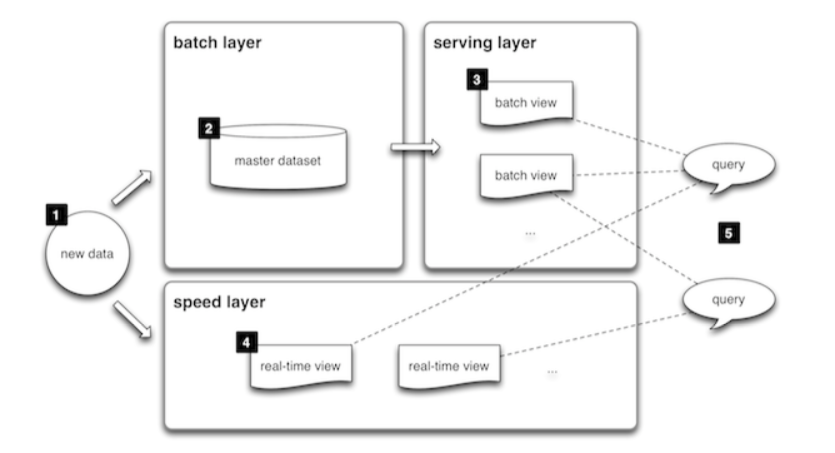
1. 데이터 처리 : 유입되는 데이터들에 특정한 상황이 발생했을 때 트리거 / 데이터의 변화, 시스템의 상태 변화 혹은 사용자의 작업내용에 반응하여 코드 수행
2. 백 엔드 : 웹, 모바일, IoT 그리고 3nd party API로부터의 요청을 처리하는 백 엔드 로직 수행
3. 시스템 제어 : AWS 자원들의 상태 및 데이터 변화에 따른 응답 및 대응 작업 수행

2. Lambda Architecture in 10 minutes or less: [https://www.youtube.com/watch?v=fPlgoTLJh38](https://www.youtube.com/watch?v=fPlgoTLJh38 )

3. 람다 아키텍처

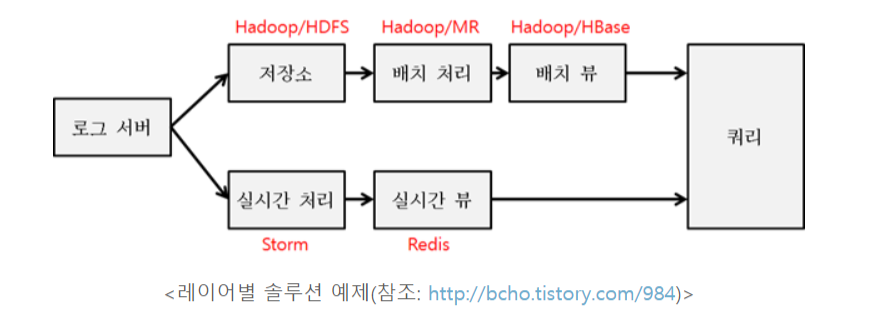
1. **Lambda architecture** is a [data-processing](https://en.wikipedia.org/wiki/Data_processing) architecture designed to handle massive quantities of data by taking advantage of both [batch](https://en.wikipedia.org/wiki/Batch_processing) and [stream-processing](https://en.wikipedia.org/wiki/Stream_processing)methods. 람다 아키텍처는 배치 및 스트림 처리 방법의 장점을 모두 활용하여 대량의 데이터를 처리하도록 설계된 데이터 처리 아키텍처입니다.  This approach to architecture attempts to balance [latency](https://en.wikipedia.org/wiki/Latency_(engineering)), [throughput](https://en.wikipedia.org/wiki/Throughput), and [fault-tolerance](https://en.wikipedia.org/wiki/Fault-tolerance) by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data. 이러한 아키텍처 접근 방식은 배치 프로세싱을 사용하여 배치 데이터의 포괄적이고 정확한 보기를 제공하는 동시에 실시간 스트림 처리를 사용하여 온라인 데이터 보기를 제공하여 지연 시간, 처리량 및 내결함성의 균형을 맞추려고 합니다.
2. (출처 : http://gyrfalcon.tistory.com/136)

실시간 분석을 지원하는 빅데이터 아키텍처, 대량의 데이터를 실시간으로 분석하기 어려우니 batch로 미리 만든 데이터와 실시간 데이터를 혼합해서 사용하는 방식이다.



데이터가 생성되면 데이터 저장소에 저장을 한다. 이 데이터는 batch로 일정주기마다 batch view를 만들어 낸다. 그리고 동일한 데이터를 실시간 데이터 처리를 통해 real-time view를 만든다.

**batch view + real-time view = 빠르지만 실시간 데이터가 반영된 분석을 할 수 있다.**



람다 아키텍처는 총 3개의 layer로 구성된다.

* batch layer : raw 데이터가 저장, batch 처리하여 batch view 생성 / batch layer인 Hadoop/HDFS, MR은 데이터를 저장하고 mapReduce로 데이터를 분석한다.



* serving layer : batch로 분석된 데이터가 저장, batch외에는 쓰기가 안됨 / serving layer는 HBase로 mapReduce로 분석한 데이터를 저장하는 NoSQL이다.

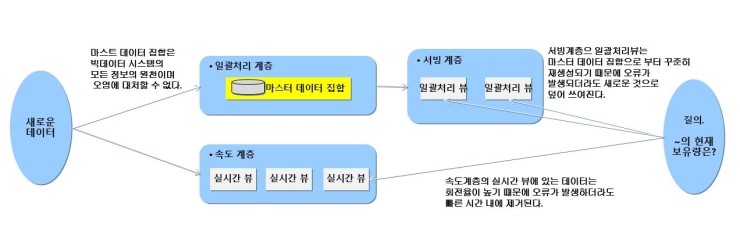


* speed layer : 실시간 데이터를 집계(real-time view) / 스트리밍 데이터를 처리하는 storm을 사용한다 = 빠른 데이터 처리가 필요하다는 뜻이다. 그리고 memory기반의 redis를 사용한다 = 빠른 속도가 필요한 솔루션을 사용했다.



여기서 batch layer에서 만든 배치 뷰 데이터와 speed layer에서 만든 실시간 뷰의 데이터가 중복되지 않게 관리하는 것이 중요하다. 이 부분은 timestamp로 해결 가능하다. 그리고 batch로 데이터가 만들어 진 후에 실시간 뷰의 데이터를 주기적으로 지워주어야 한다.

4. 람다 아키텍처 구성개념



: 마스터 데이터집합을 기반으로 놓고 (Master Dataset)

: 이 집합에 대해 최종적 일관성을 지는 뷰를 생성하는 일괄처리 영역을 그 위에 붙이고

: 뷰를 물고 질의를 처리하는 서빙계층을 그 위에 올리고

: 일괄처리 계층의 느린 속도를 커버하는 속도계층으로 보완하는 포괄적인 모델

5. API Gateway + Lambda 간단 사용 예제 (백 엔드)

(출처 : https://blog.naver.com/homs44/220751194898)

