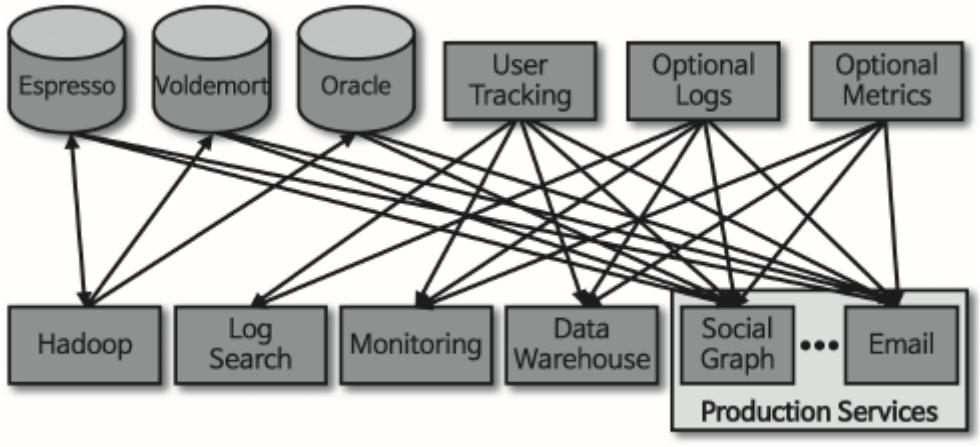
아파치카프카의 역사와미래

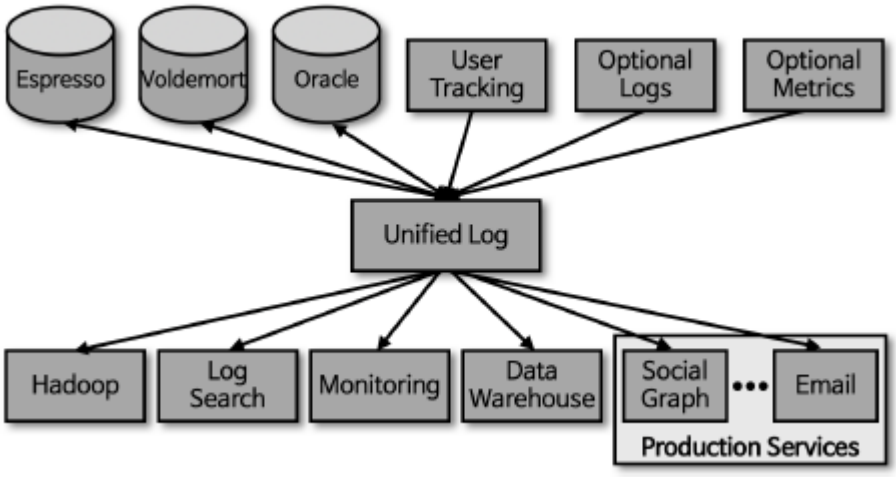
아파치 카프카의 탄생

> 아파치 카프카는 대용량의 실시간 메시지 스트리밍 플랫폼으로, 2011년 LinkedIn에서 처음 개발되었습니다. 초기에는 LinkedIn 내부에서 사용하던 것으로 시작하여 2012년에 오픈소스로 공개되었습니다.

> 데이터를 생성하고 적재하기 위해서는 데이터를 생성하는 소스 애플리케이션과 데이터가 최종 적재되는 타깃 애플리케이션을 연결해야 한다. 초기 운영 시에는 단방향 통신을 통해 소스 애플리케이션에서 타깃 애플리케이션으로 연동하는 소스코드를 작성했고 아키텍처가 복잡하지 않았으므로 운영이 힘들지 않았다. 그러나 시간이 지날수록 아키텍처는 거대해졌고 소스 애플리케이션과 타깃 애플리케이션의 개수가 점점 많아지면서 문제가 생겼다. 데이터를 전송하는 라인이 기하급수적으로 복잡해지기 시작했다.



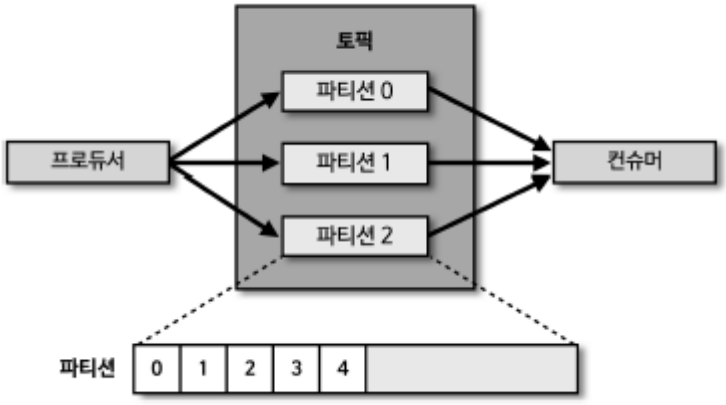
> 이를 해결하기 위해 링크드인 데이터팀에서는 다양한 메시징 플랫폼과 ETL(Extract Transform Load) 툴을 적용하여 아키텍처를 변경하려고 노력했지만 파편화된 데이터 파이프라인의 복잡도를 낮춰주는 아키텍처가 되지는 못했다.



> 링크드인 데이터팀은 신규 시스템을 만들기로 결정했고 그 결과물이 아파치 카프카(Apache Kafka)다.

→ 카프카는 각각의 애플리케이션끼리 연결하여 데이터를 처리하는 것이 아니라 한 곳에 모아 처리할 수 있도록 중앙집중화했다.

메시지 큐 구조를 그대로 살린 카프카 내부 구조



> 카프카 내부에 데이터가 저장되는 파티션의 동작은 FIFO(FirstInFirstOut)방식의 큐 자료구조와 유사하다. 큐에 데이터를 보내는 것이 ‘프로듀서’이고 큐에서 데이터를 가져가는 것이 ‘컨슈머’다.

> 프로듀서가 메시지를 보내면 파티션에 저장된다
> 파티션은 큐구조

> 컨슈머는 차례대로 데이터를 가져간다
> 파티션에 저장된 데이터는 삭제되지 않는다

아파치 카프카가 데이터 파이프라인으로 적합한 4가지 이유

빅데이터 파이프라인에 적합한 카프카의 특징 1 - 높은 처리량

> 많은 양의 데이터를 송수신할 때 맺어지는 네트워크 비용은 무시할 수 없는 규모가 된다.
> 동일한양의 데이터를 보낼 때 네트워크 통신 횟수를 최소한으로 줄인다면 동일 시간 내에 더 많은 데이터를 전송할 수 있다. 많은 양의 데이터를 묶음 단위로 처리하는 배치로 빠르게 처리할 수 있기 때문에 대용량의 실시간 로그데이터를 처리하는 데에 적합 하다. 또한, 파티션 단위를 통해 동일 목적의 데이터를 여러 파티션에 분배하고 데이터를 병렬처리할 수 있다. 파티션 개수만큼 컨슈머를 늘려서 동일 시간당 데이터 처리량을 늘리는 것

빅데이터 파이프라인에 적합한 카프카의 특징 2 - 확장성

> 데이터 파이프라인에서 데이터를 모을 때 데이터가 얼마나 들어올지는 예측하기 어렵다.

> 특정 이벤트로 인해 많은 양의 데이터가 들어오는 경우가 있다. 카프카는 이러한 가변적인 환경에서 안정적으로 확장 가능하도록 설계되었다. 데이터가 적을 때는 카프카 클러스터의 브로커를 최소한의 개수로 운영하다가 데이터가 많아지면 클러스터의 브로커 개수를 자연스럽게 늘려 스케일 아웃(scale-out)할 수있다. 반대로 데이터 개수가 적어지고 추가 서버들이 더는 필요없어지면 브로커 개수를 줄여 스케일 인(scale-in)할 수 있다. 카프카의 스케일 아웃, 스케일 인 과정은 클러스터의 무중단 운영을 지원하므로 365일 24시간 데이터를 처리해야 하는 커머스나 은행 같은 비즈니스 모델에서도 안정적인 운영이 가능하다

빅데이터 파이프라인에 적합한 카프카의 특징 3 - 영속성

> 영속성이란 데이터를 생성한 프로그램이 종료되더라도 사라지지 않은 데이터의 특성을 뜻한다.

> 카프카는 다른 메시징 플랫폼과 다르게 전송받은 데이터를 메모리에 저장하지 않고 파일 시스템에 저장한다. 파일 시스템에 데이터를 적재하고 사용하는 것은 보편적으로 느리다고 생각하겠지만, 카프카는 운영체제 레벨에서 파일 시스템을 최대한 활용하는 방법을 적용하였다.

> 운영체제에서는 파일 I/O 성능 향상을 위해 페이지 캐시(page cache) 영역을 메모리에 따로 생성하여 사용한다. 페이지 캐시 메모리 영역을 사용하여 한번 읽은 파일 내용은 메모리에 저장시켰다가 다시 사용하는 방식이기 때문에 카프카가 파일 시스템에 저장하고 데이터를 저장, 전송하더라도 처리량이 높은 것이다.

> 디스크 기반의 파일 시스템을 활용한 덕분에 브로커 애플리케이션이 장애 발생으로 인해 급작스럽게 종료되더라도 프로세스를 재시작하여 안전하게 데이터를 다시 처리할 수 있다.

빅데이터 파이프라인에 적합한 카프카의 특징 4 - 고가용성

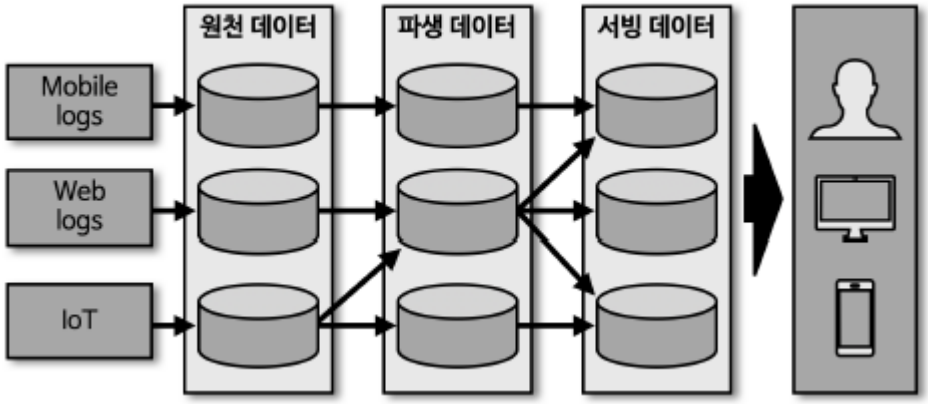
> 3개 이상의 서버들로 운영되는 카프카 클러스터는 일부 서버에 장애가 발생하더라도 무중단으로 안전하고 지속적으로 데이터를 처리할 수 있다. 클러스터로 이루어진 카프카는 데이터의 복제(replication)를 통해 고가용성의 특징을 가지게 되었다. 프로듀서로 전송받은 데이터를 여러 브로커 중 1대의 브로커에만 저장하는 것이 아니라 또 다른 브로커에도 저장하는 것이다.

> 한 브로커에 장애가 발생하더라도 복제된 데이터가 나머지 브로커에 저장되어 있으므로 저장 된 데이터를 기준으로 지속적으로 데이터 처리가 가능한 것이다.

> 이에 더하여 서버를 직접 운영하는 온프레미스(on-premise) 환경의 서버 랙 또는 퍼블릭 클라우드(public cloud)의 리전 단위 장애에도 데이터를 안전하게 복제할 수 있는 브로커 옵션들이 준비되어 있다.

빅데이터 아키텍처의 종류와 카프카의 미래

데이터 레이크 아키텍처와 카프카의 미래 - 초기 빅데이터 플랫폼

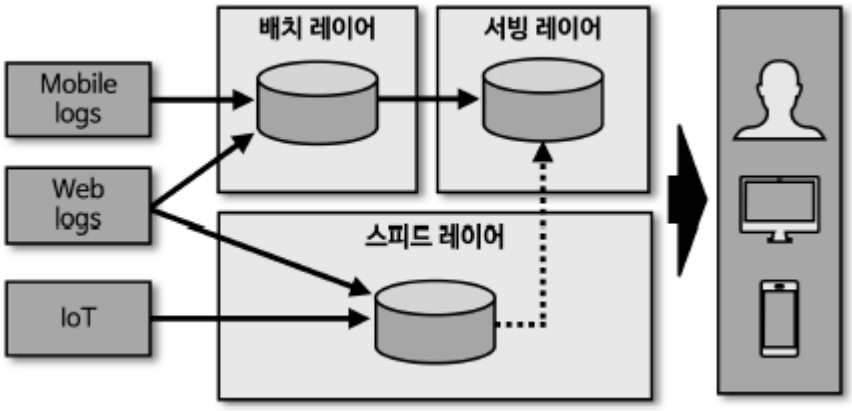


> 초기 빅데이터 플랫폼은 엔드 투 엔드로 각 서비스 애플리케이션으로부터 데이터를 배치로 모았다.

> 데이터를 배치로 모으는 구조는 유연하지 못했으며 실시간으로 생성되는 데이터들에 대한 인사이트를 서비스 애플리케이션에 빠르게 전달하지 못하는 단점이 있었다.

> 원천 데이터로부터 파생된 데이터의 히스토리를 파악하기 어려웠고 계속되는 데이터의 가공으로 인해 데이터가 파편화되면서 데이터 거버넌스(data governance: 데이터 표준 및 정책)를 지키기 어려웠다.

데이터 레이크 아키텍처와 카프카의 미래 - 람다 아키텍처



> 람다 아키텍처는 3가지 레이어로 나뉜다.
> 배치 레이어는 배치 데이터를 모아서 특정 시간, 타이밍마다 일괄처리

> 서빙 레이어는 가공된 데이터를 데이터 사용자

> 서비스 애플리케이션이 사용할 수 있도록 데이터가 저장된 공간이다

> 스피드 레이어는 서비스에서 생성되는 원천 데이터를 실시간으로 분석하는 용도로 사용한다

> 배치 데이터에 비해 낮은 지연으로 분석이 필요한 경우에 스피드 레이어를 통해 데이터를 분석한다

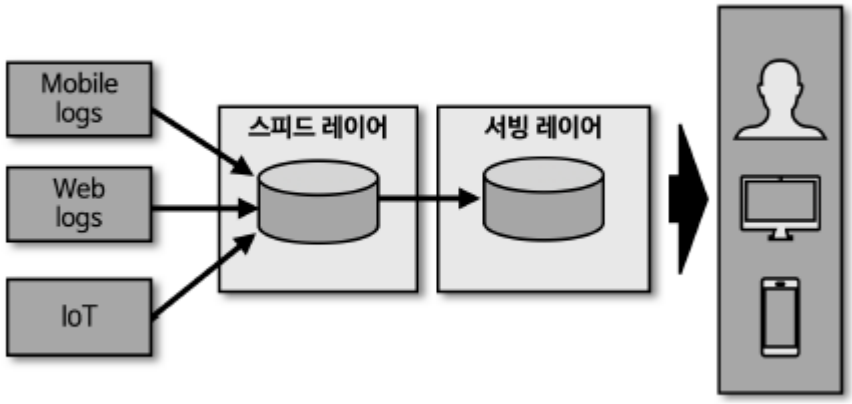
람다 아키텍처의 한계

> 람다 아키텍처는 데이터 처리 방식을 명확히 나눌 수 있었지만 레이어가 2개로 나뉘기 때문에 생기는 단점이 있다.

> 데이터를 분석, 처리하는데에 필요한 로직이 2벌로 각각의 레이어에 따로 존재해야 한다는 점과 배치 데이터와 실시간 데이터를 융합하여 처리할 때는 다소 유연하지 못한 파이프라인을 생성해야 한다는 점

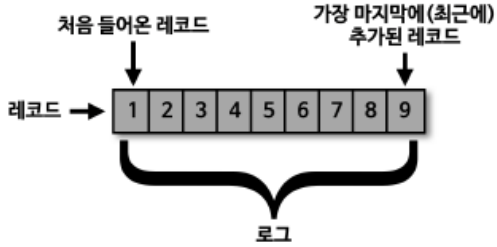
> 1개의 로직을 추상화하여 배치 레이어와 스피드 레이어에 적용하는 형태를 고안한 서밍버드가 있었지만 완벽하지 않았음

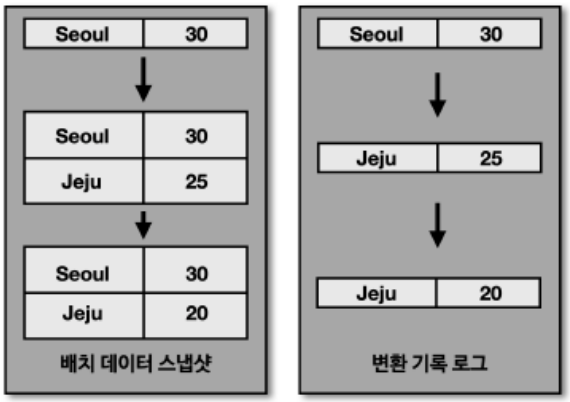
데이터 레이크 아키텍처와 카프카의 미래 - 카파 아키텍처



> 람다 아키텍처에서 단점으로 부각되었던 로직의 파편화, 디버깅, 배포, 운영 분리에 대한 이슈를 제거하기 위해 배치 레이어를 제거한 카파 아키텍처는 스피드 레이어에서 데이터를 모두 처리가 가능하다

카파 아키텍처의 활용





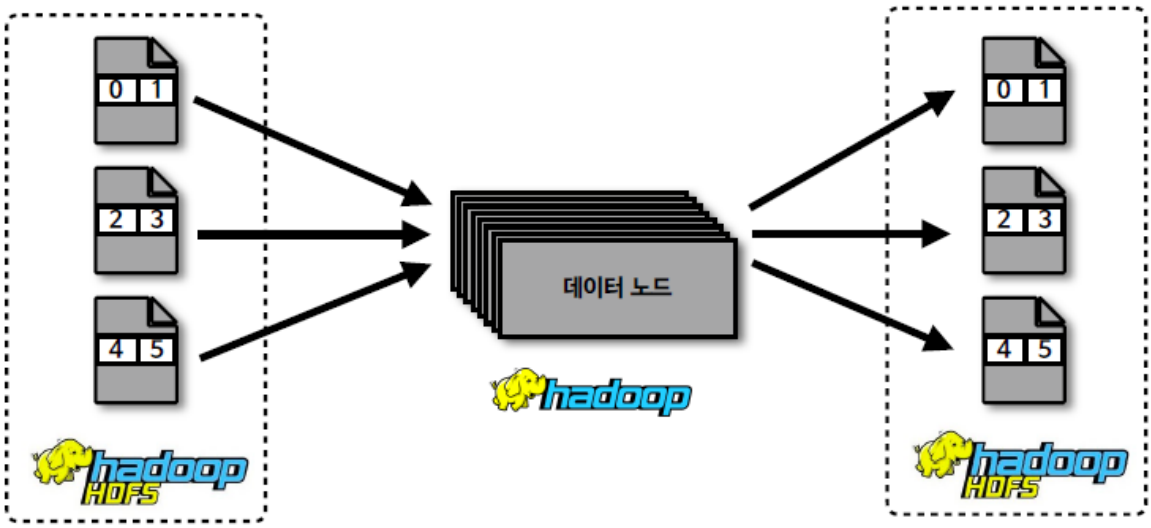
> 로그는 배치 데이터를 스트림으로 표현하기에 적합하다

> 일반적으로 데이터 플랫폼에서 배치 데이터를 표현할 때는 각 시점(시간별, 일자별 등)의 전체 데이터를 백업한 스냅샷 데이터를 뜻했다.

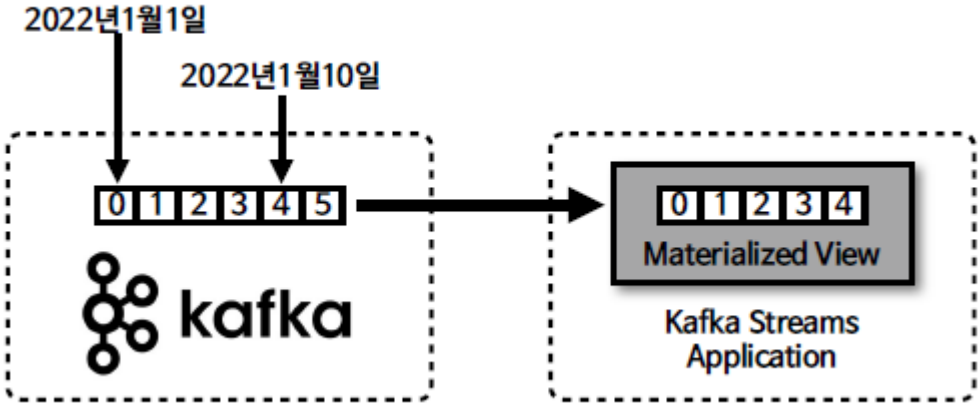
> 배치 데이터를 로그로 표현할 때는 각 시점의 배치 데이터의 변환 기록(change log)을 시간 순서대로 기록함으로써 각 시점의 모든 스냅샷 데이터를 저장하지 않고도 배치 데이터를 표현할 수 있게 되었다

배치 데이터	스트림 데이터
- 한정된(bounded) 데이터 처리 - 대규모 배치 데이터를 위한 분산 처리 수행 - 분, 시간, 일 단위 처리를 위한 지연 발생 - 복잡한 키 조인 수행	- 무한(unbounded) 데이터 처리 - 지속적으로 들어오는 데이터를 위한 분산 처리 수행 - 분 단위 이하 지연 발생 - 단순한 키 조인 수행

배치 데이터를 처리하는 방법(in 하둡)



스트림 데이터를 배치로 사용하는 방법(in 카프카)

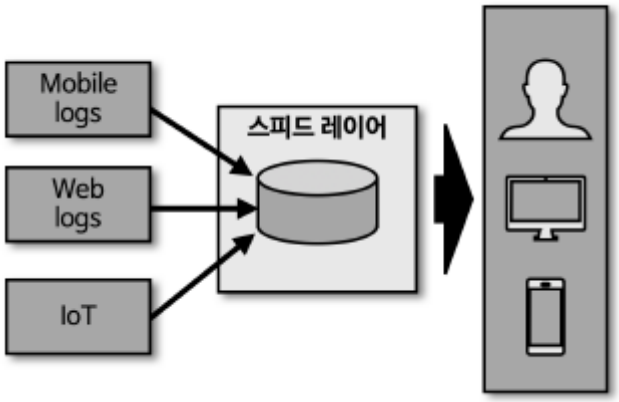


> 스트림 데이터를 배치 데이터로 사용하는 방법은 로그에 시간을 남기는 것

> 로그에 남겨진 시간을 기준으로 데이터를 처리하면 스트림으로 적재된 데이터도 배치로 처리할 수 있게 된다.

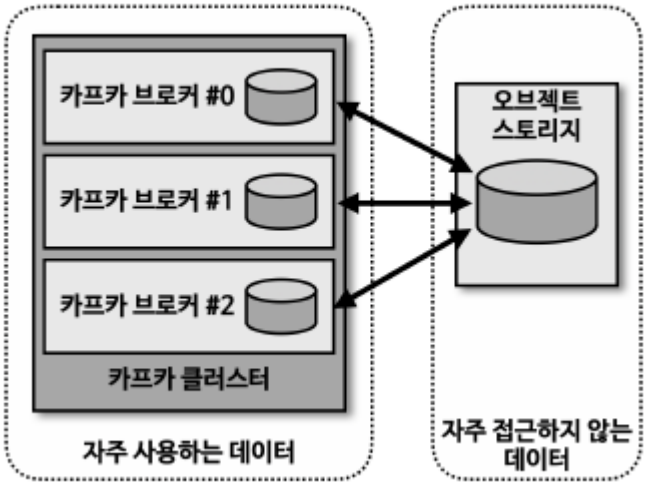
> 카프카는 로그에 시간(timestamp)을 남기기 때문에 이런 방식의 처리가 가능하다

데이터 레이크 아키텍처와 카프카의 미래 - 스트리밍 데이터 레이크



> 카파 아키텍처를 살펴보면 데이터를 사용하는 고객을 위해 스트림 데이터를 서빙레이어에 저장하는 것을 알수있다

> 스피드 레이어로 사용되는 카프카에 분석과 프로세싱을 완료한 거대한 용량의 데이터를 오랜 기간 저장하고 사용할 수 있다면 서빙 레이어는 제거되어도 된다오히려 서빙 레이어와 스피드 레이어가 이중으로 관리되는 운영 리소스를 줄일 수 있다



> 아직은 카프카를 스트리밍 데이터 레이크로 사용하기 위해 개선해야 하는 부분이 있다. 우선 자주 접근하지 않는 데이터를 굳이 비싼 자원(브로커의 메모리, 디스크)에 유지할 필요가 없다.

> 카프카 클러스터에서 자주 접근하지 않는 데이터는 오브젝트 스토리지와 같이 저렴하면서도 안전한 저장소에 옮겨 저장하고 자주 사용하는 데이터만 브로커에서 사용하는 구분 작업이 필요