Data Pipeline @KakaoStory

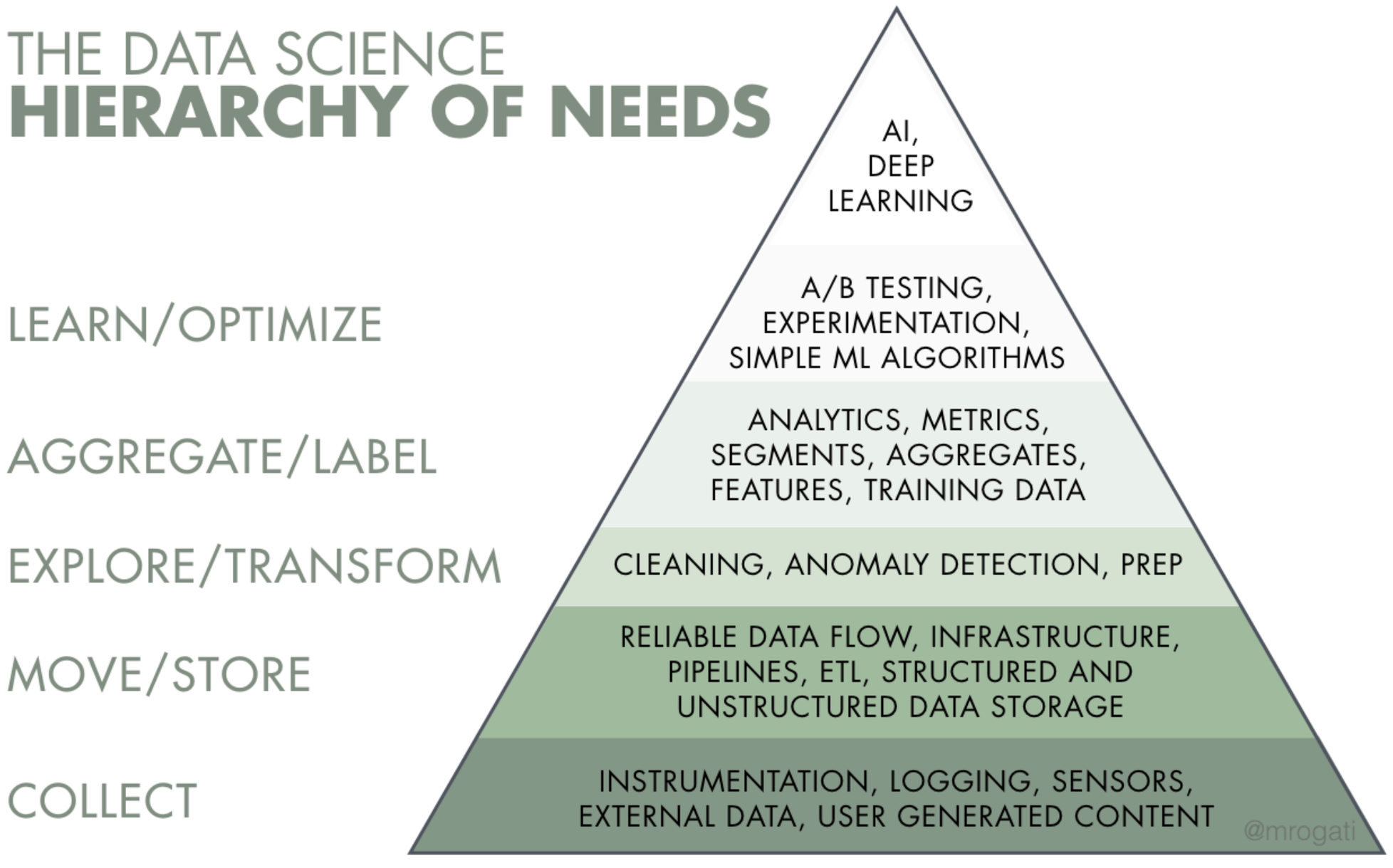
2018.09.08

Agenda

- Data Pipeline @ why
- Data Pipeline @ How
    - Lambda Architecture
    - ETL (Extract Transform Load)
    - Implementation
    - Kakaostory Data Pipeline
- Kafka topic & log
    - Simple Implementation
    - Unified Log Processing
    - Transformer & Loader
- Kafka & HDFS
- ETL case study
    - A/B Test Pipeline
- Summary

Data Pipeline @ Why

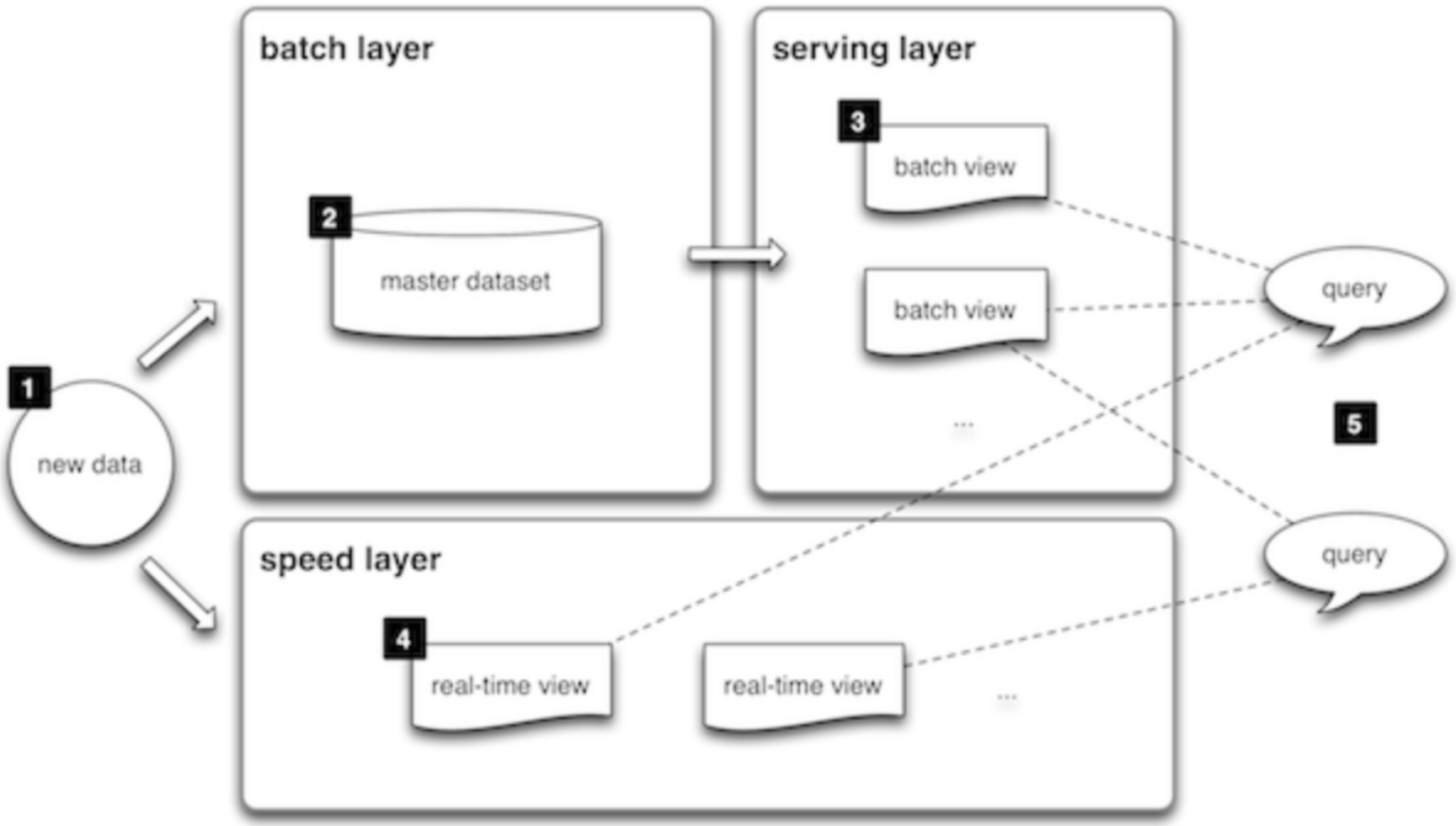
- query = function(all data)
- data pipeline makes functions



출처 : https://bit.ly/2vuripR

Data Pipeline @ How

Lambda(λ) Architecture



1. All **data** entering the system is dispatched to both the batch layer and the speed layer for processing.
2. The **batch layer** has two functions: (i) managing the master dataset (an immutable, append-only set of raw data), and (ii) to pre-compute the batch views.
3. The **serving layer** indexes the batch views so that they can be queried in low-latency, ad-hoc way.
4. The **speed layer** compensates for the high latency of updates to the serving layer and deals with recent data only.
5. Any incoming **query** can be answered by merging results from batch views and real-time views.

출처 : http://lambda-architecture.net/

Data Pipeline @ How

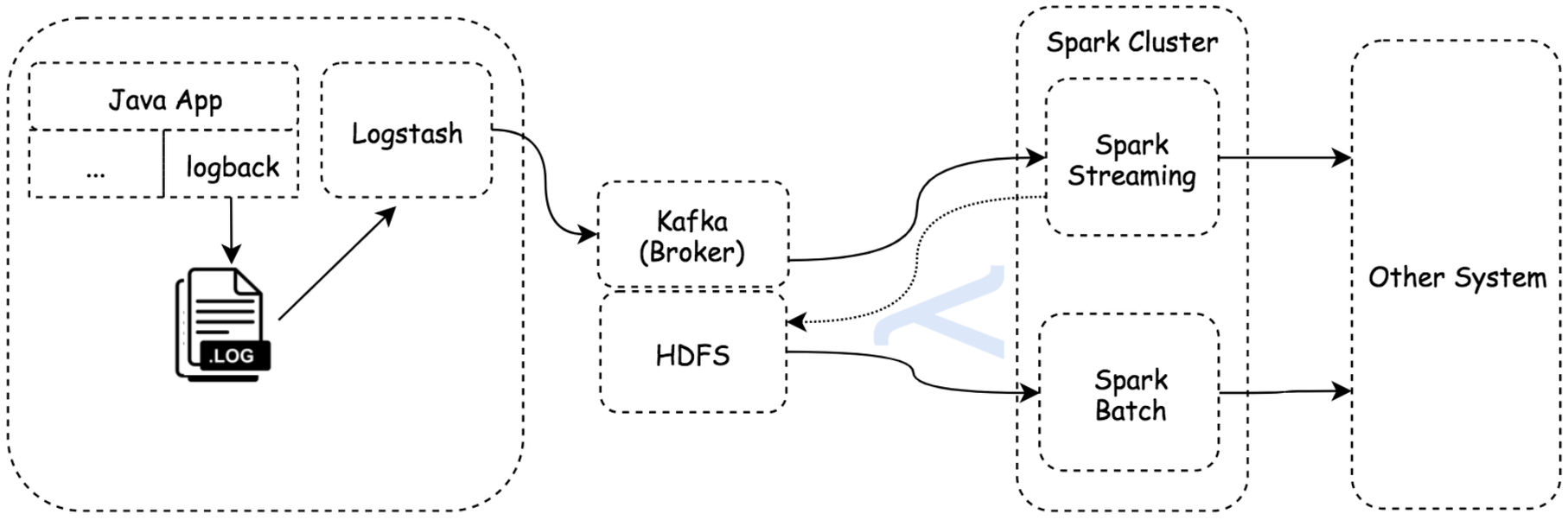
ETL (Extract Transform Load)

- E (extract)
  - 데이터 소스로 부터 데이터를 추출하는 과정
    - 실시간 처리 : ex) data reading from kafka
    - 배치 처리 : ex) data reading from HDFS
    - 개인 컴퓨터 : ex) data reading from csv file
- T (transform)
  - 입력 데이터를 최종 시스템 요구사항에 맞게 변환하는 과정
    - Cleaning, Validation, Sorting, Join, Format transform, Applying business rule, ...
- L (Load)
  - 변환된 데이터를 서비스될 최종 시스템으로 전달하는 단계
    - RDB, NoSQL, 실시간 분석 플랫폼(OLAP), ...

Data Pipeline @ How

Implementation

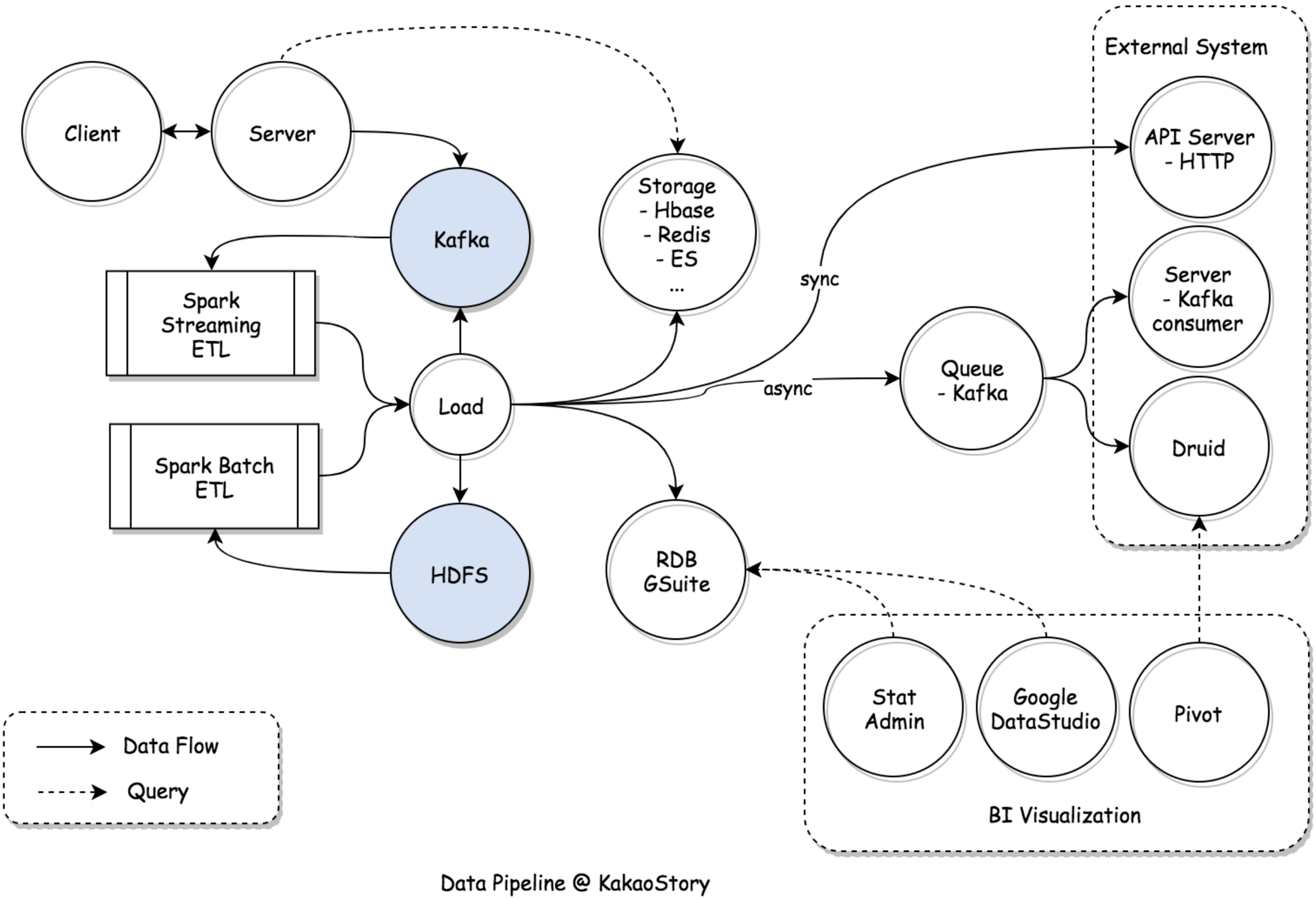
- 주요 구성요소
  - logback, logstash, kafka, hdfs, spark, scala
  - log modeling (client log, server log, ...)
- why spark
  - 데이터 처리에 필요한 리소스 scale out
  - 배치, 실시간처리, ML지원
- Scala로 커스텀 ETL 작성
  - scala learning curve. but no pain, no gain
  - 기존 java library의 활용 (xxx client library)



Data pipeline implementation

Data Pipeline @ How

Kakaostory Data Pipeline

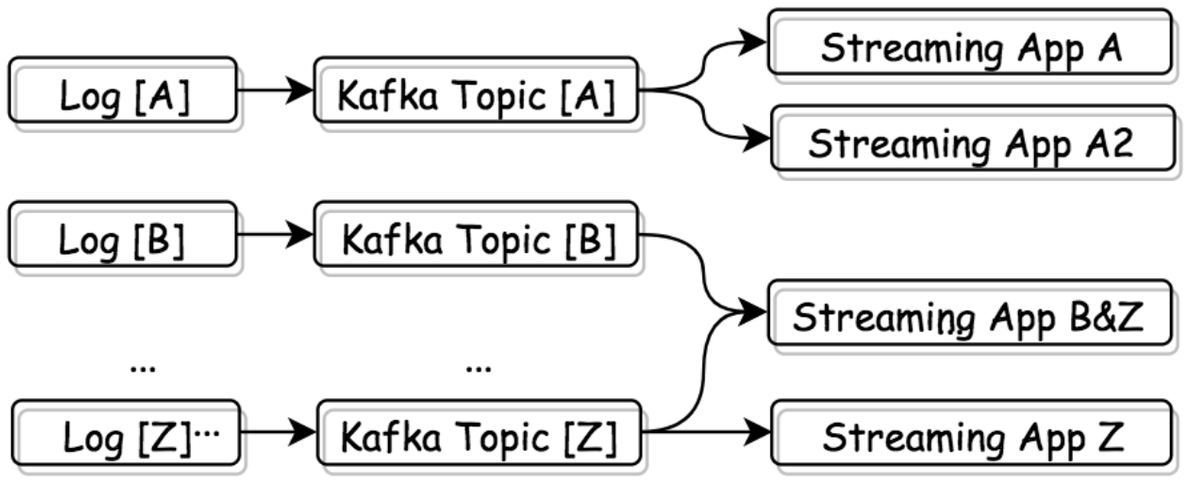


Data Pipeline @ KakaoStory

Kafka topic & Log

Simple Implementation

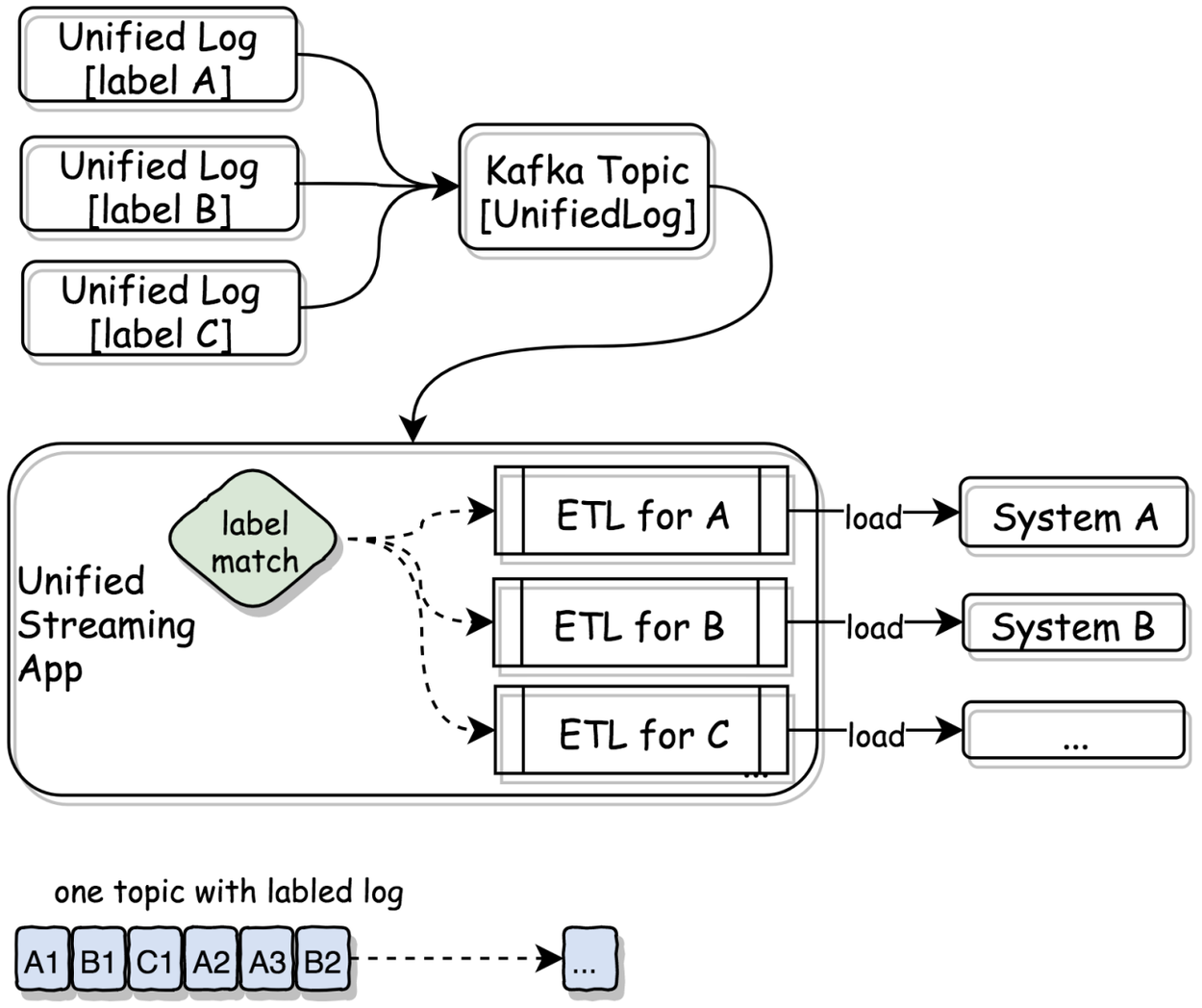
- 가장 단순한 카프카 응용
- 필요한 로그 정의 및 카프카 토픽 생성
- 장점
  - 단순한 구성
  - 앱간 간섭이 없다.
- 단점
  - ETL 작업이 많아지면서 스트리밍 앱이 선형 증가
  - 단순 ETL 작업이라고해도 1개의 프로세스 작업을 차지
  - 로그 생성시 마다 Kafka 토픽을 계속 생성
  - spark streaming의 경우 기본값으로 최소 구성하면
    - Driver (2GB) + Executor (2GB) * 2개 = 6GB
    - 6GB / App (메모리 관점에서)
- 카카오스토리 초기 구성 형태
  - ETL 작업 10개 미만, 단순하고 좋았음
  - ETL 작업 10개 이상, 작업을 늘리는게 부담이 됨



Kafka topic & Log

Unified Log Processing

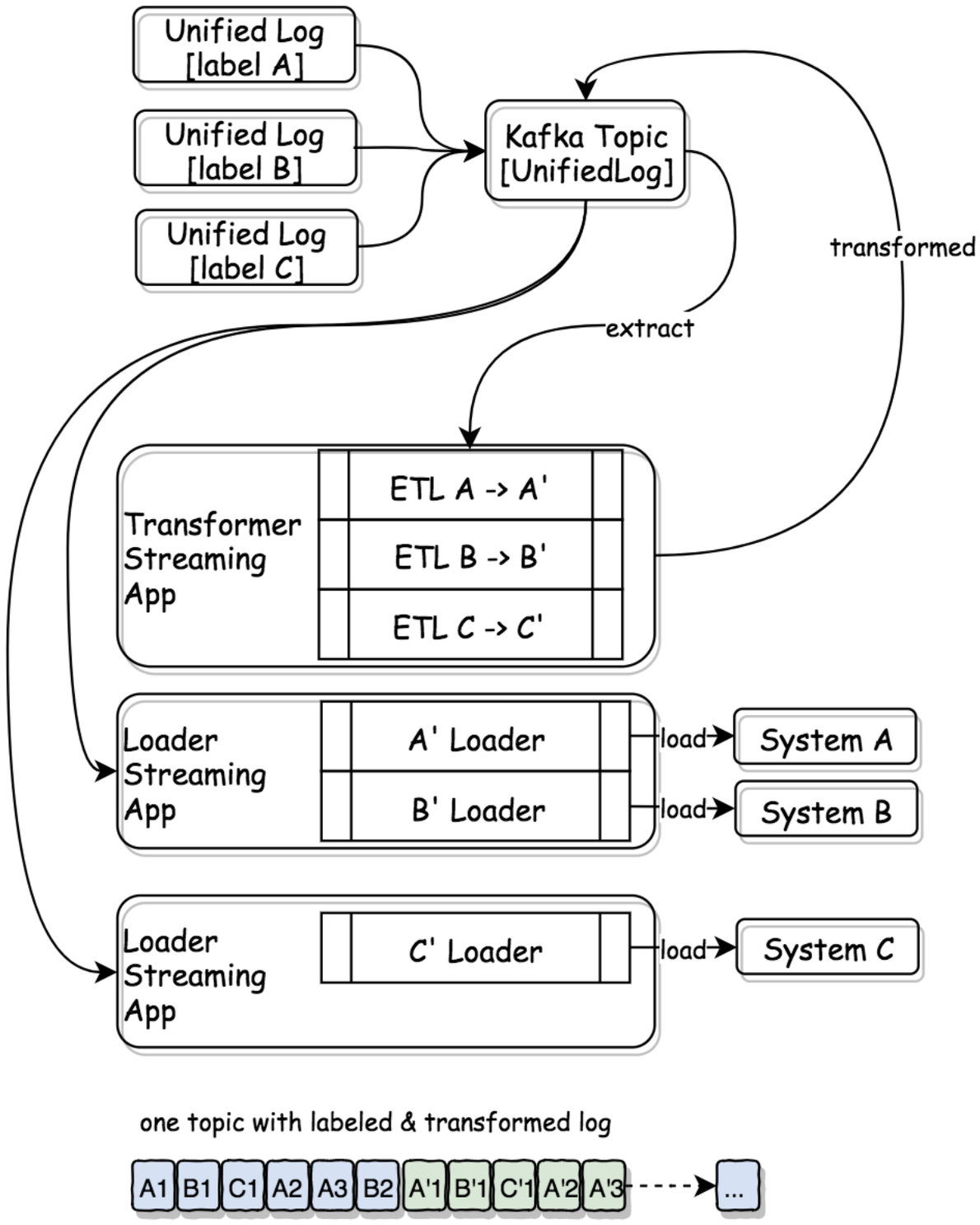
- 스트리밍 앱을 최소로 만드는 구조 고민
- 개선 (Producer)
  - 카프카 토픽은 물리적인 채널로 이용
  - 로그 종류는 로그 데이터에 레이블을 함께 전송
- 개선 (Consumer)
  - 스트리밍 앱에서 로그 레이블 별 ETL 처리 로직 분기
- 장점
  - 로그 포맷이 일원화되어 관리하기 편함
  - 로그 종류를 늘리더라도 카프카 토픽을 새로 생성할 필요가 없음
  - 최소한의 스트리밍 앱으로 ETL 작업 처리
- 단점
  - 특정 ETL 작업 지연시 전체 ETL 작업 영향
  - 작업 지연은 주로 통제할 수 없는 외부 시스템 로딩 과정에서 발생



Kafka topic & Log

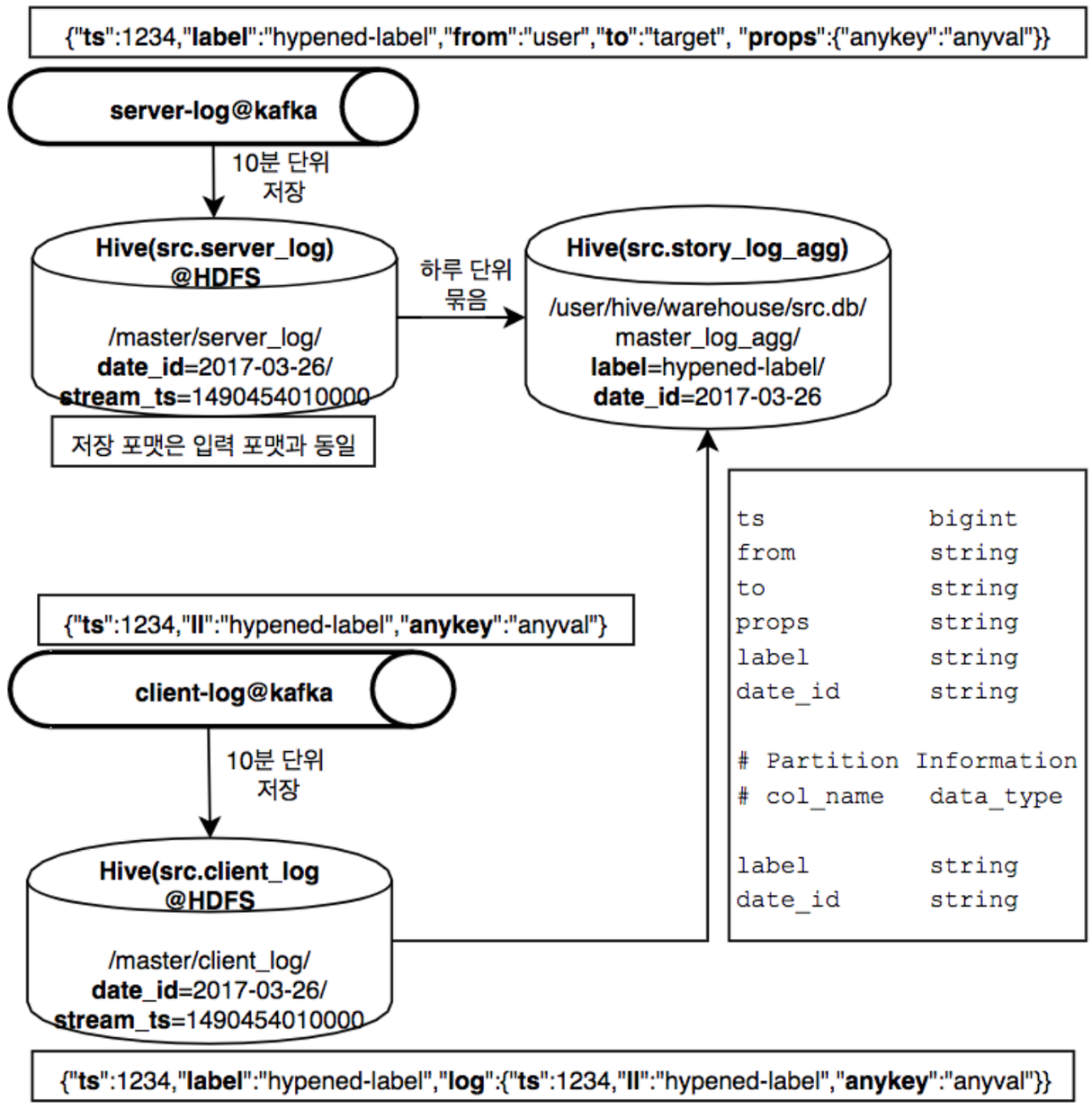
Transformer & Loader

- 개선 (Consumer)
  - 포멧 변환과 최종 로딩과정 분리
  - 포멧 변환 : Transformer
  - 로딩 : Loader
- 장점
  - 스트리밍 앱 최소 운영
  - 로더 특성에 따른 스트리밍 앱 분리
    - 특정 작업에 대한 영향도를 분리
  - 로더 로직 재사용
    - HTTP, Kafka, HBase, ...



Kafka log & HDFS

- Batch Layer 구축을 위한 사전 작업
- 마스터 로그를 배치 작업에 유용한 형태로 HDFS에 저장
  - 로그 종류(label)과 수집 날짜를 기준으로 파티셔닝
- 주로 일/시간단위 로그 취합
- 여러 로그 소스의 구분도 label을 이용하여 구분
- HDFS 로그의 활용
  - spark batch 프로그램에서 ETL 작업 수행
  - hive external table 매핑 후 hive query

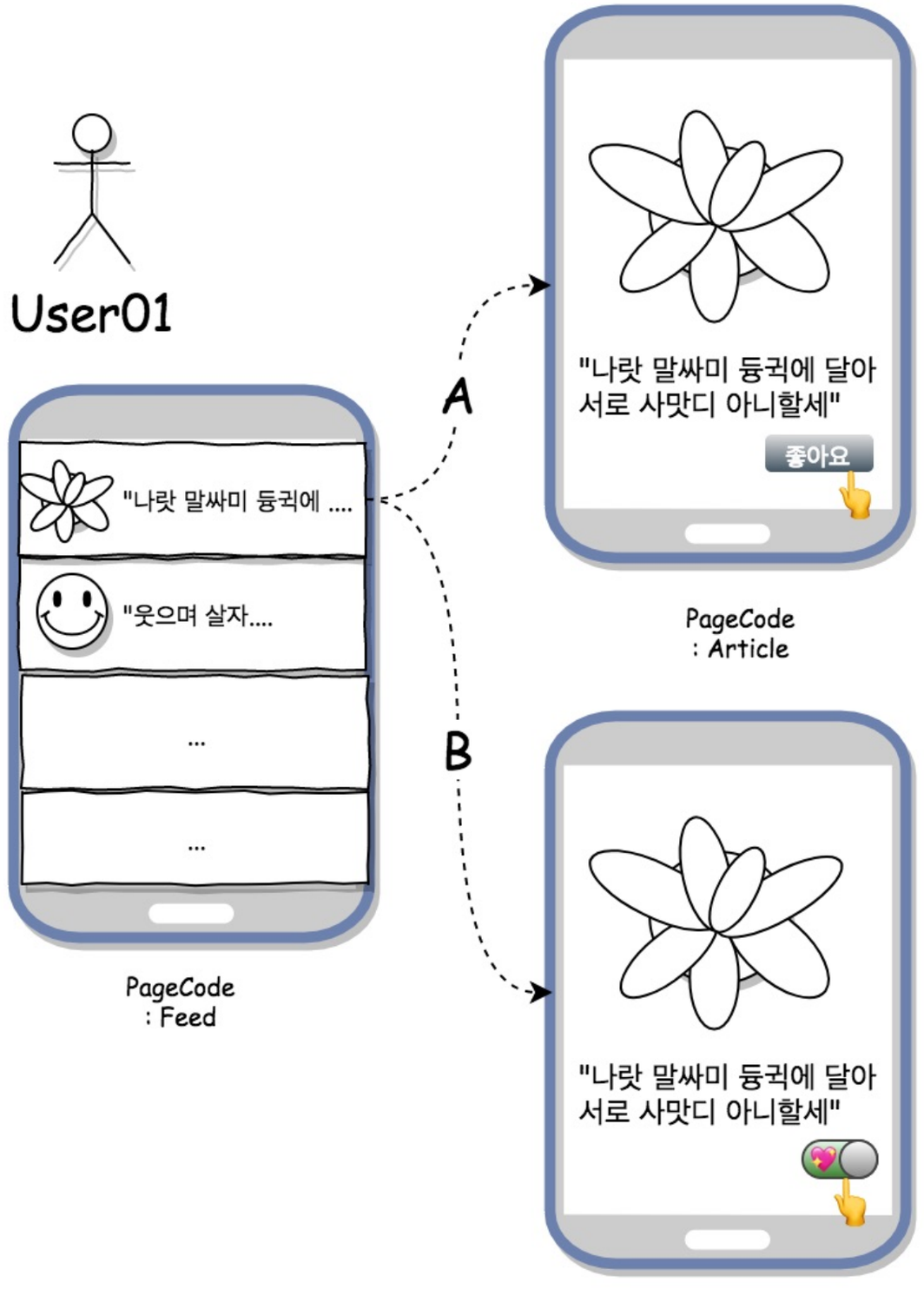


ETL case study

A/B Test Pipeline

- 문제 정의
  - 사용자는 어떤 타입의 좋아요 버튼을 선호할까?
  - 데이터 기반 의사결정 시스템
  - 아이템 노출에 대한 전환율 측정 문제

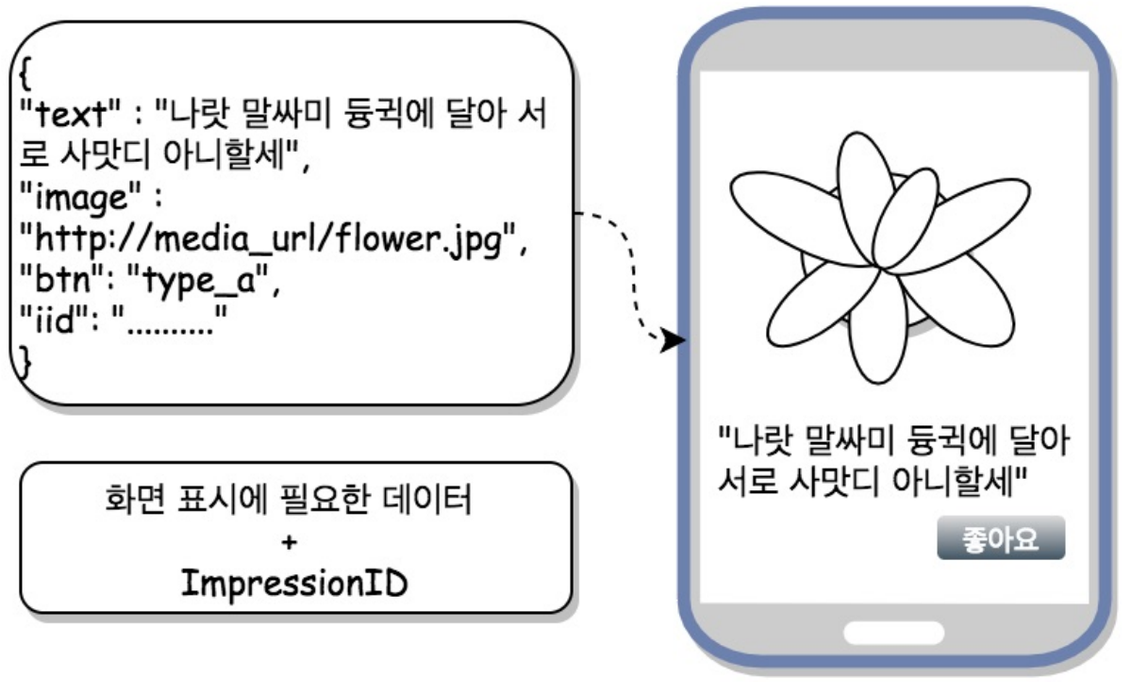
- 어떻게 만들지?
  - 사용자 이벤트는 어떻게 정의?
  - 서버와 클라이언트는 어떤 데이터를 교환할까?
  - 실험 비중은 어떻게 제어할까?
  - 아이템 전환율 측정을 위한 데이터 구조는 어떻게 가져갈까?



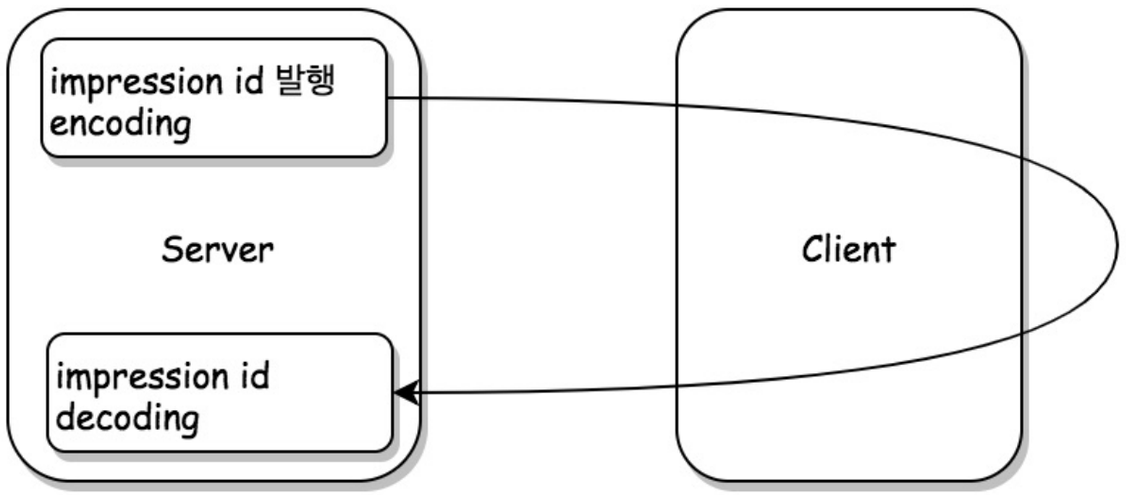
ETL case study

A/B Test Pipeline

- Data between client and server
  - 화면 표시에 필요한 데이터
  - 피드백 측정용 데이터
- Impression ID
  - 광고/추천에서 효과측정을 위해 사용하는 기법
  - Impression ID 노출 대비 클릭으로 전환율 측정
  - 클라이언트 릴리즈에 영향 받지않는 형태
  - 부가정보가 필요한 경우 확장가능한 형태
  - 클라이언트는 해당 값을 해석할 필요 없고 잘 받아서 잘 되돌려주면 됨
- Impression ID Example
  - 노출 아이템 정보 : 글 ID, 버튼 타입, 사용자 ID, 연령, ...
  - 하나의 값으로 변환 : json + base64
  - before encoding
    - { "article_id": 123, "algorithm": "text_btn", "user_id": "user01", "birth_year": "1984", ...}
  - after encoding
    - eyAiYXJ0aWNsZV9pZCI6IDEyMywgImFsZ29yaXRobSI6ICJ0ZXh0X2J0biIsICJ1c2VyX2lkIjogInVzZXIwMSIsICJiaXJ0aF95ZWFyIjogIjE5ODQiLCAuLi59Cgo



data between client and server

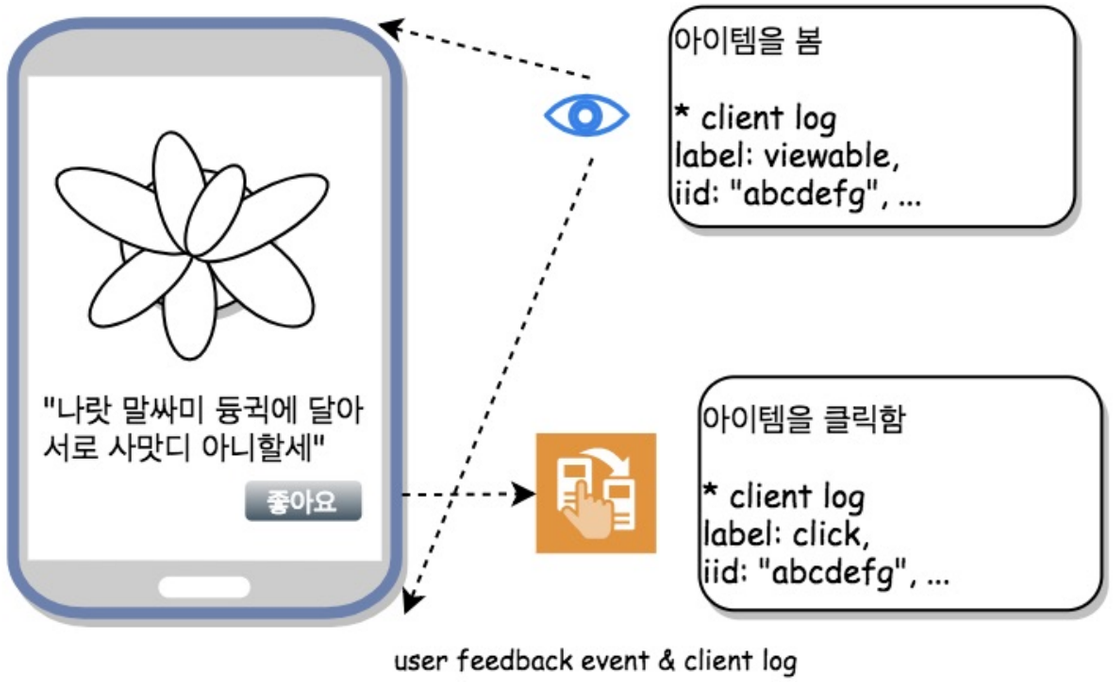


impression id for feedback

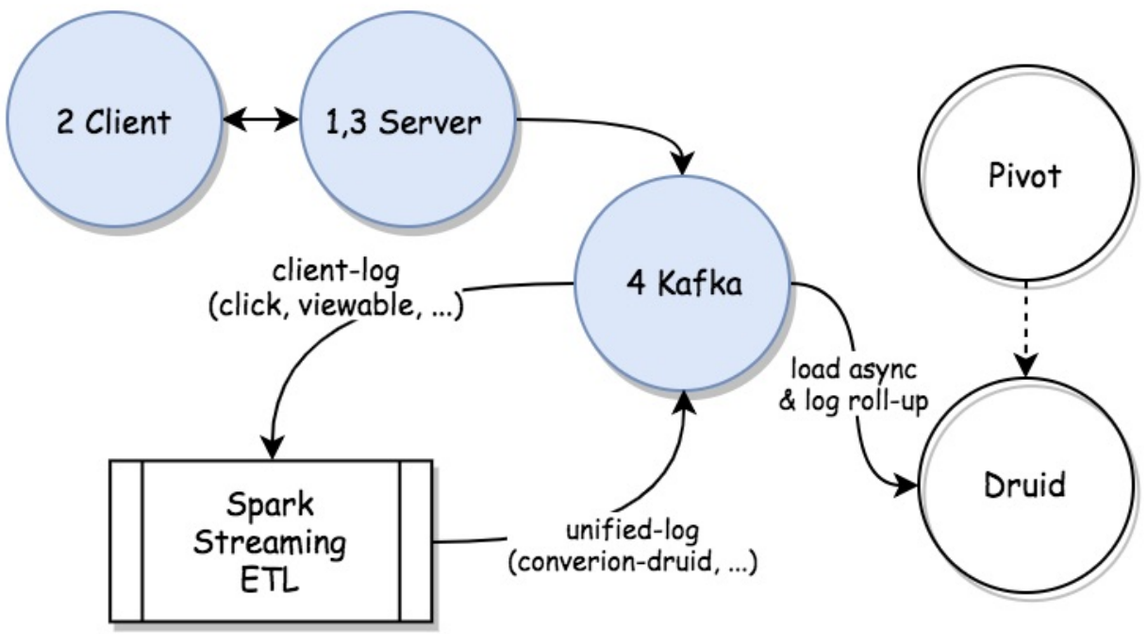
ETL case study

A/B Test Pipeline

- 서버 구현
  - 렌더링 정보와 impression id 발행
  - 50 : 50 노출 실험 가정하면
    - 호출 트래픽 기반
      - rand() < 0.5 => A type impression id 발행
      - rand() >= 0.5 => B type impression id 발행
    - 사용자 기반
      - 사용자 아이디 % 10 => A type (0~4), B type (5~9)
- 클라이언트 구현
  - 사용자 피드백을 이벤트 로그로 변환
  - 아이템의 노출
    - label=viewable, kafka topic=client-log
  - 아이템의 클릭
    - label=click, kafka topic=client-log
  - 노출 정보 매핑
    - iid(impression id)에 추가
- 다시 서버 구현
  - 클라이언트로 받은 피드백 로그를 kafka로 전송



user feedback event & client log

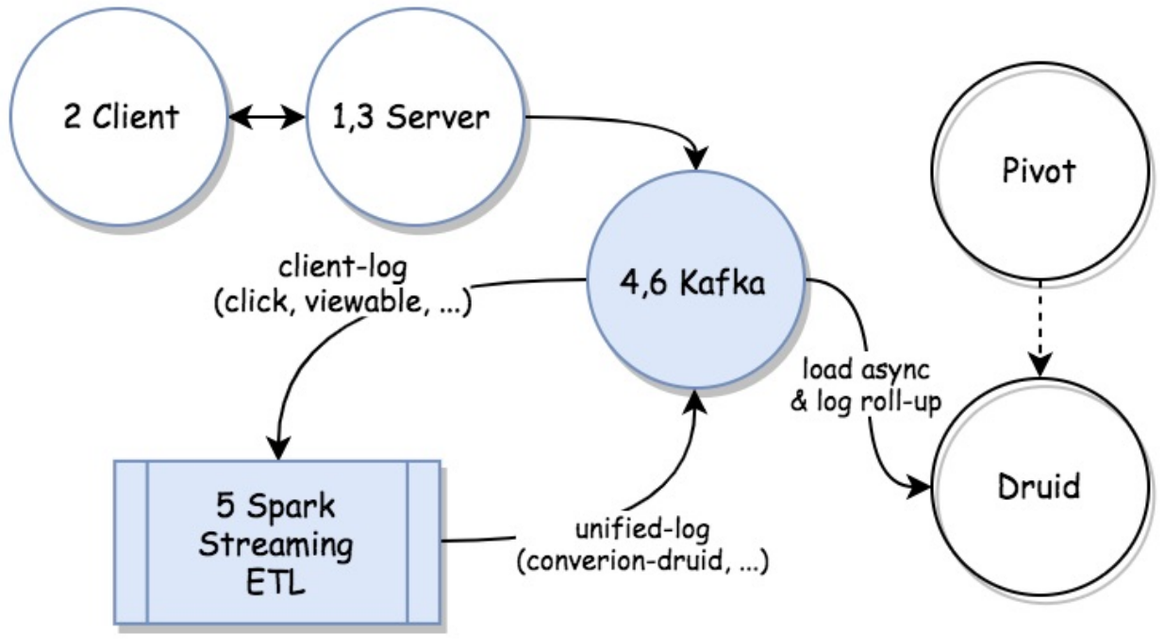


data flow @ data pipeline

ETL case study

A/B Test Pipeline

- Extract
  - client-log read from kafka
  - drop invalid log
  - decode impression id
- Transform
  - 측정가능한 형태로 변환작업
  - viewable to conversion log
    - { label: viewable, page: 123, iid: "abcdefg", ... }
    - { label: conversion, algorithm: A, impressions: 1, clicks: 0 }
  - click to conversion log
    - { label: click, page: 123, iid: "abcdefg", ... }
    - { label: conversion, algorithm: A, impressions: 0, clicks: 1 }
- Load
  - conversion log를 druid 로딩용 kafka topic으로 전송

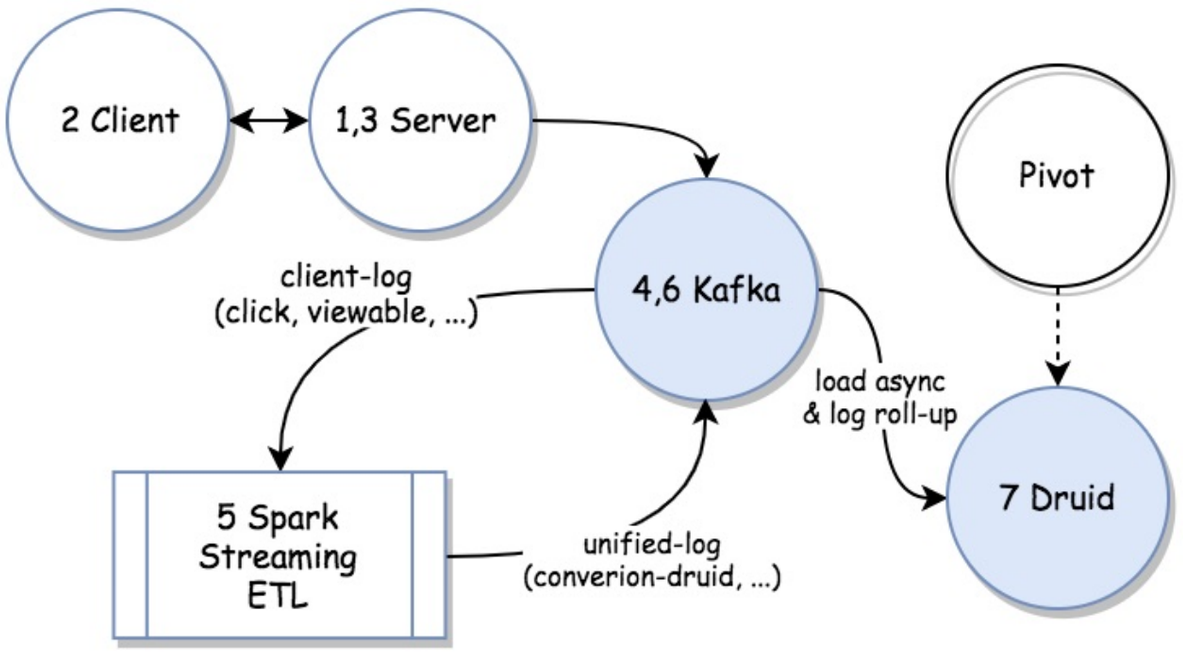


data flow @ data pipeline

ETL case study

A/B Test Pipeline

- Load from Druid
  - roll up 과정을 통해 실시간 뷰 구성
- Before Roll Up
  - { label: conversion, algorithm: A, impressions: 1, clicks: 0, ... }
  - { label: conversion, algorithm: A, impressions: 0, clicks: 1, ... }
  - { label: conversion, algorithm: A, impressions: 1, clicks: 0, ... }
  - { label: conversion, algorithm: B, impressions: 1, clicks: 0, ... }
  - { label: conversion, algorithm: B, impressions: 1, clicks: 0, ... }
- After Roll Up
  - { label: conversion, algorithm: A, impressions: 2, clicks: 1 }
  - { label: conversion, algorithm: B, impressions: 2, clicks: 0 }
- 전환율 측정
  - A : 2번 노출 1번 클릭, 1 / 2 = 0.5 => 50% 전환
  - B : 2번 노출 0번 클릭, 0 / 2 = 0 => 전환없음



data flow @ data pipeline

ETL case study

A/B Test Pipeline

- BI visualization
  - Real time View
    - Druid -> Pivot
  - Batch View
    - Daily spark batch
      - spark batch to query druid
      - load stat to google spreadsheets
- google spreadsheets -> data studio

Summary

- 지속가능한 파이프라인 구축을 위해서는?
  - 개발자간 협업 (front / backend / system / ...)
  - 통합된 로그 형식 및 Data Flow에 대한 고민은 매우 중요함
  - 지속적으로 관리 포인트는 제거
    - 스트리밍 앱의 갯수는 최소로 운영
    - 불필요한 데이터소스 및 ETL작업은 주기적으로 제거
- 비정형 데이터를 바라보는 관점 변화
  - 로그는 깨끗하지 않다.
  - 로그 누락 및 재처리 가능성을 보고 뒷단 설계
  - 로그 스키마는 계속 변한다.
- 데이터다루는 방법에 대한 끊임없는 호기심
  - 패러다임 & 용어
    - Immutable, Functional, Idempotent, Eventual consistency, ...
  - 확률 자료 구조
    - Hyperloglog, Bloomfilter, ...
  - Exploration & exploitation
    - A/B test => Thompson Sampling
  - BI
    - dimension, segment, cardinality, metric, ...
  - ML pipeline