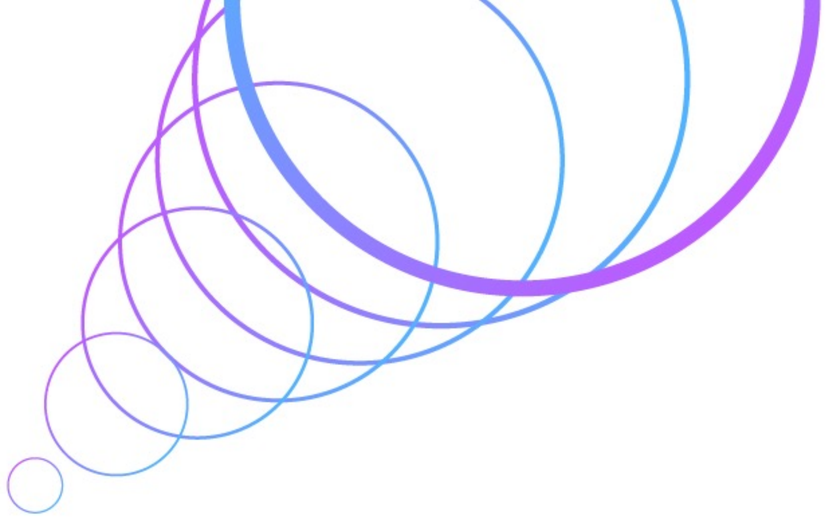


DEVIEW 2021

하루 n억개 웹툰 로그를 처리하는 Realtime Application 만들기

김시온 NAVER WEBTOON

CONTENTS

1. WEBTOONS’ BIGDATA

1.1 Introduce My Department

DEVIEW 2021

WEBTOON BIGDATA PLATFORM

- 네이버웹툰, 글로벌 웹툰(LINE WEBTOON) 등 여러 서비스의 로그 수집
- 대용량의 데이터를 처리하기 위해 Hadoop Ecosystem 운영
- 데이터 파이프라인을 구축하여 다양한 데이터를 안정적으로 서빙
- 적재된 Raw Log를 수차례 가공하여, 지표로 활용할 수 있는 집계 데이터 생성
- 분석가, 서비스팀를 위한 대시보드, BI(Business Intelligence) Tool 제공

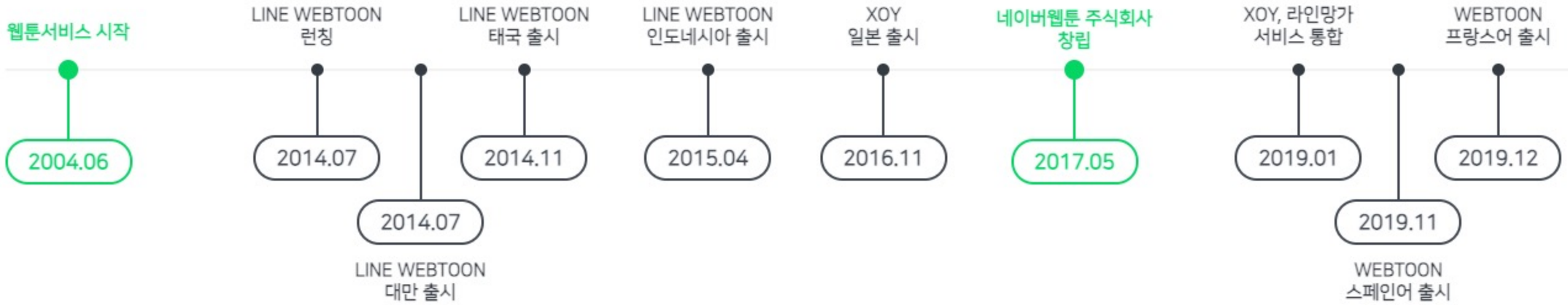
웹툰에서의 빅데이터가 낯설어요!

What is WEBTOONS’ BIGDATA

1.2 WEBTOONS' BIGDATA

WEBTOONS의 성장과 데이터

- 벌써 햇수로 17년째!
- 네이버웹툰, 시리즈, 시리즈온 등
- LINE WEBTOON : 글로벌 서비스
- 사용자의 증가, 계속 성장하는 서비스

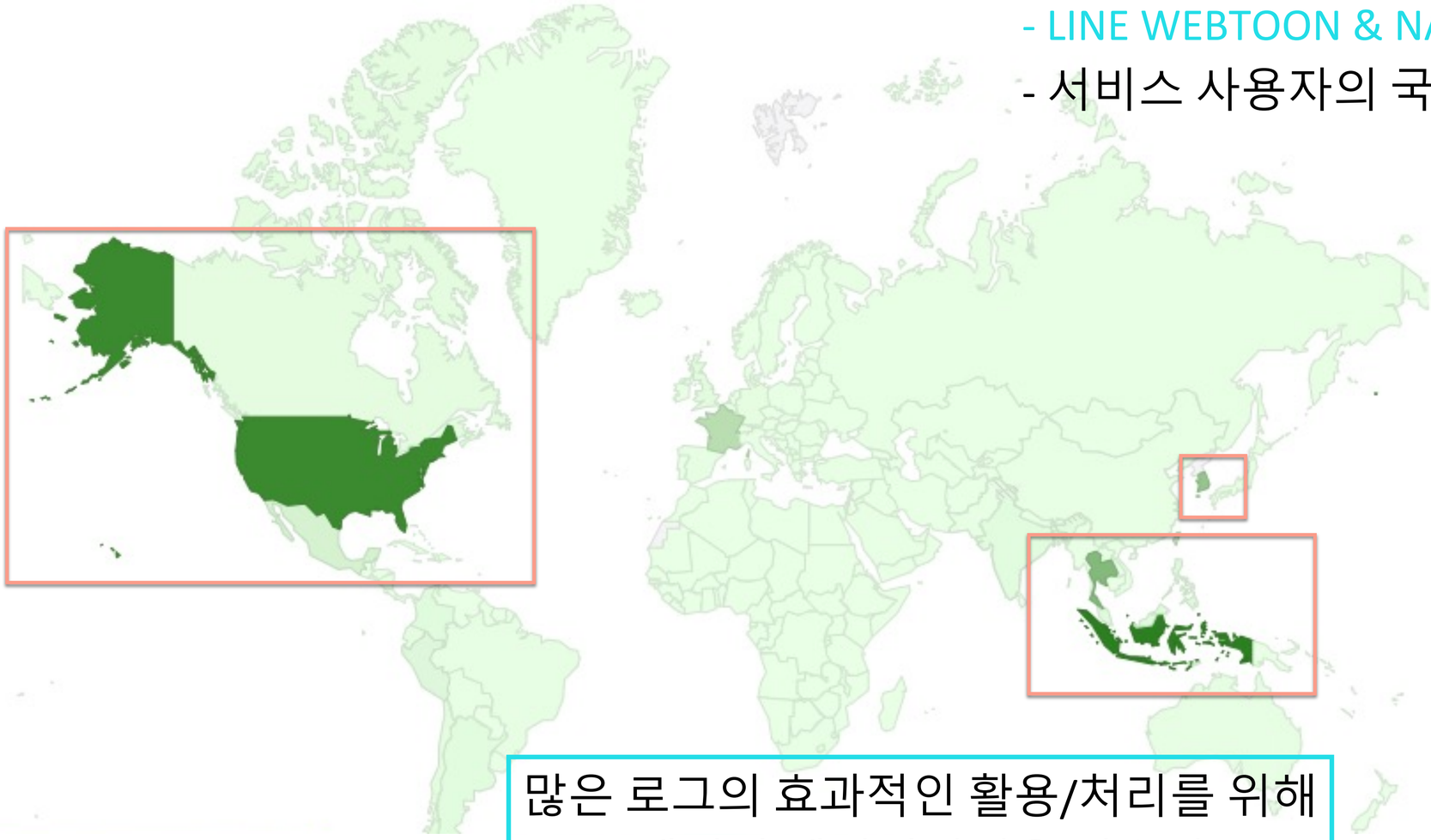


- 서비스의 전략, 사용자에게 양질의 서비스 제공을 위해
  * 데이터 수집 및 분석은 필수적

1.2 WEBTOONS' BIGDATA

Korea And Global

- LINE WEBTOON & NAVER WEBTOON GeoData
- 서비스 사용자의 국가 분포



많은 로그의 효과적인 활용/처리를 위해
데이터 엔지니어링은 필수적

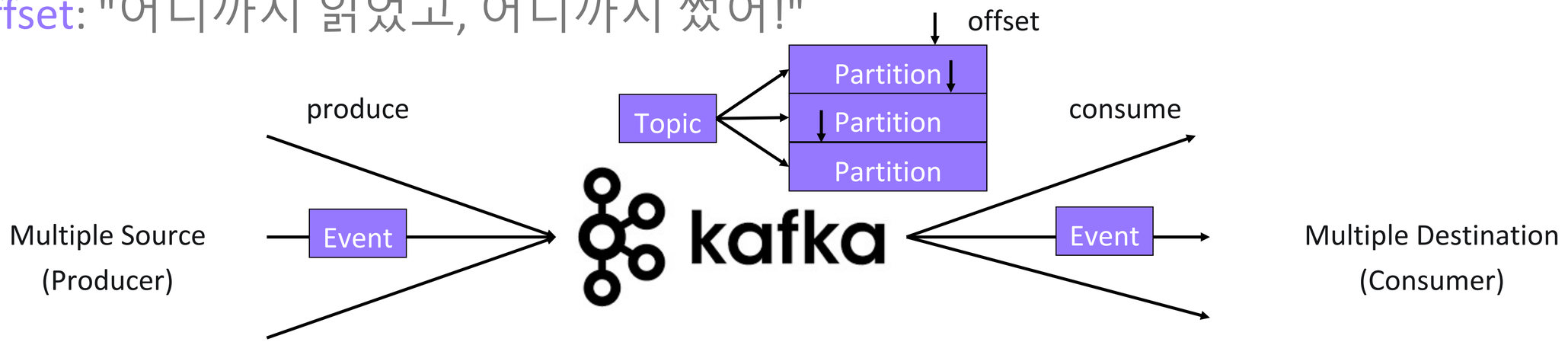
2. Kafka Streams 도입 배경

BackGrounds:
Apache Kafka와 AS-IS Pipeline

2.1 Apache Kafka

Apache Kafka: 기본 개념

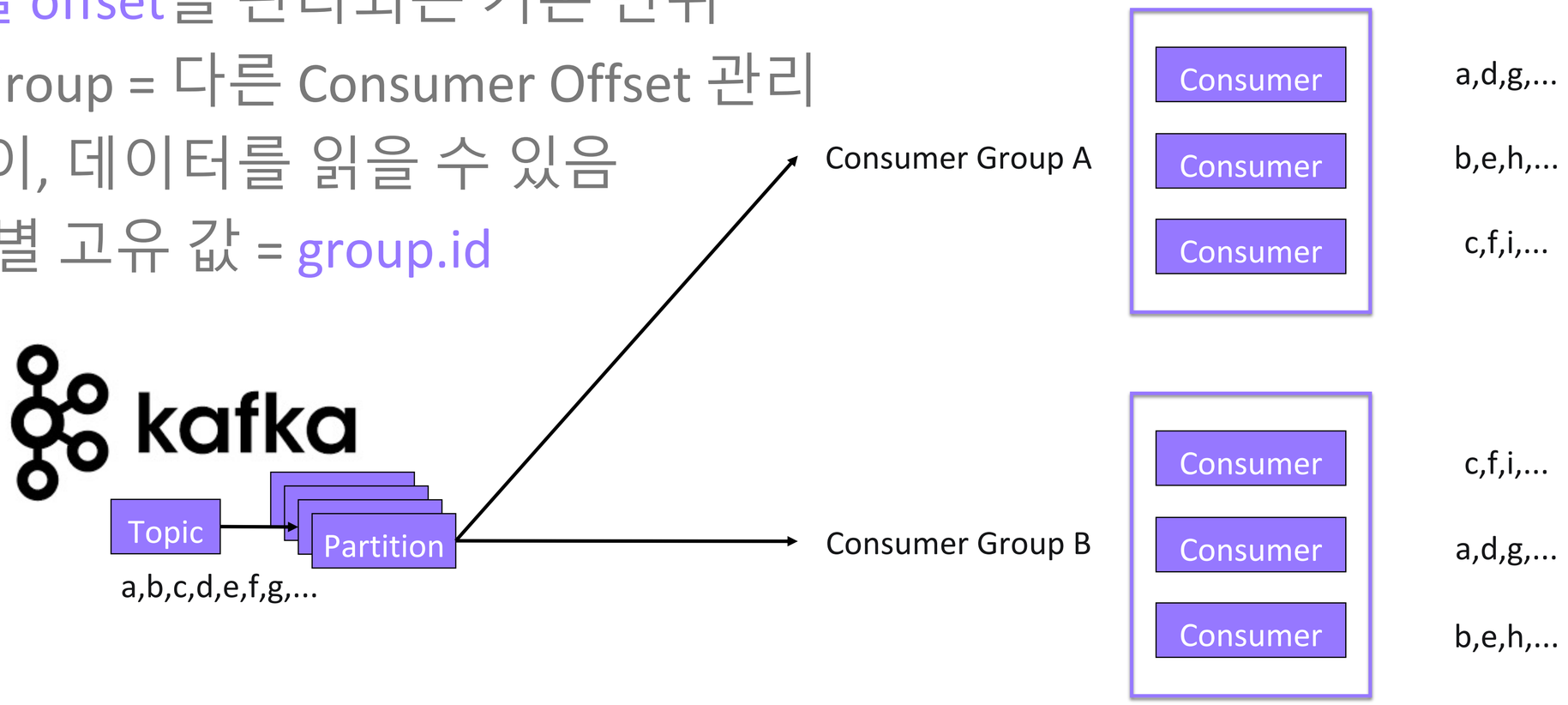
- 비동기 처리를 위한 메세징 큐
- 실시간으로 스트림을 게시, 구독, 저장 및 처리할 수 있는 분산 데이터 스트리밍 플랫폼
- 주요 개념
  * Producer & Consumer: "*Event(Message)*를 쓰는 주체, 읽는 주체"
  * Topic & Partitions : "데이터 저장소, 근데 이제 분산되어 저장된"
  * Offset: "어디까지 읽었고, 어디까지 썼어!"



2.1 Apache Kafka

Apache Kafka: Consumer Group

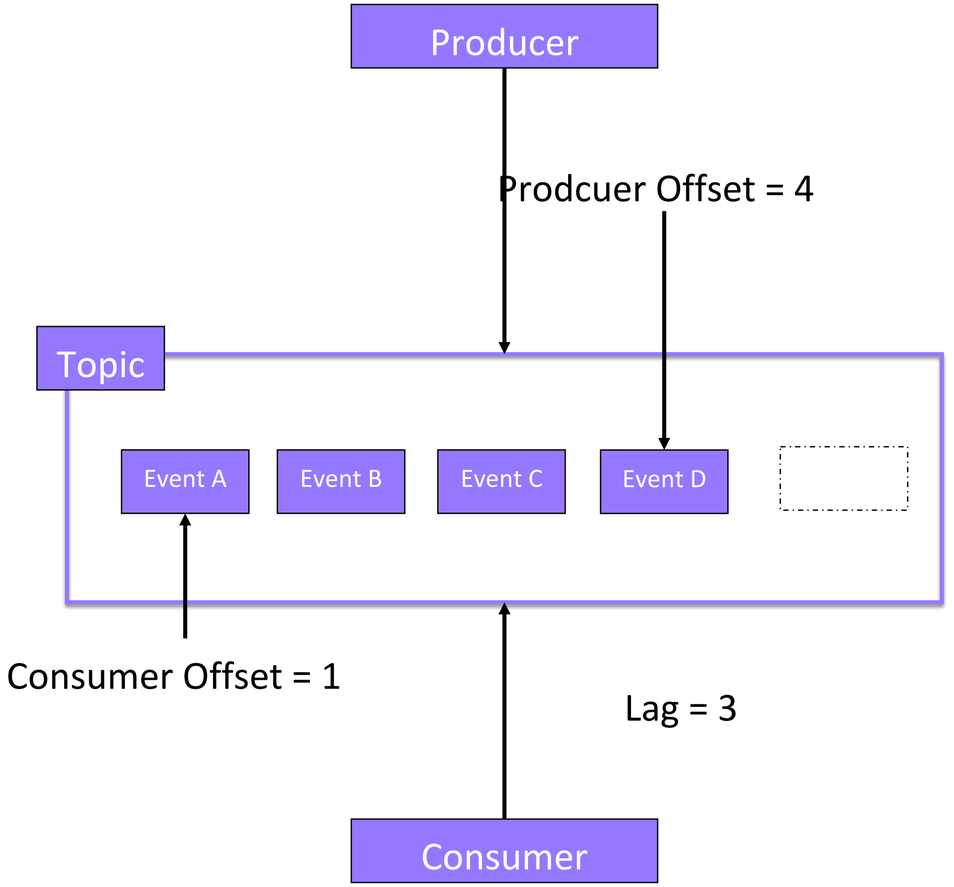
- Consumer instance를 대표하는 그룹
- 토픽 내 파티션별 offset을 관리되는 기본 단위
- 다른 Consumer Group = 다른 Consumer Offset 관리
- Group간 간섭없이, 데이터를 읽을 수 있음
- Consumer Group별 고유 값 = group.id



2.1 Apache Kafka

Apache Kafka: Lag

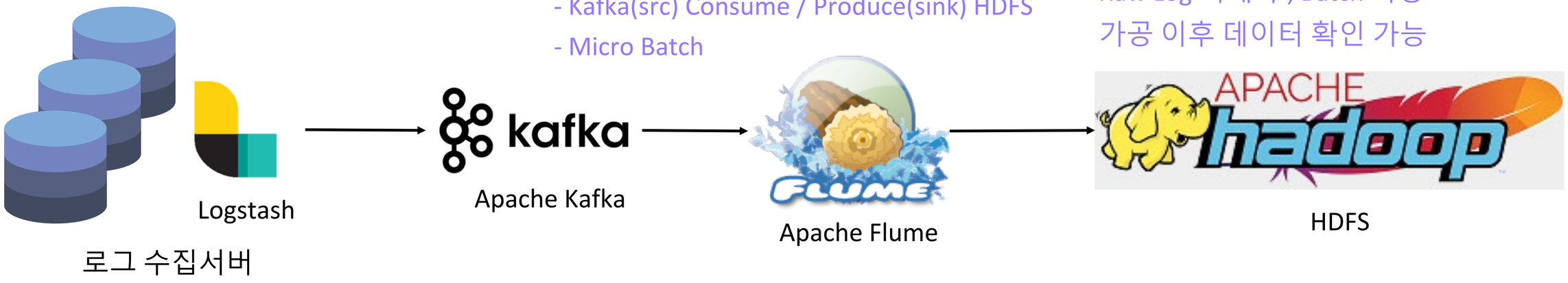
- Producer Offset – Consumer Offset
- 주로 Consumer의 상태 확인시 사용
- Lag가 높아진다
  * Produce 속도가 증가
  * Consume 속도가 감소
- Producer 상태 정상시, Consumer 이슈



2.2 AS-IS PIPELINE

BATCH PIPELINE

- 글로벌웹툰(LINE WEBTOON)의 처리 파이프라인
- Batch: 주기적(*Hourly/Daily/Weekly,...*)으로 데이터 처리
- 정해진 시간(배치 시간)을 기다려야 데이터를 조회할 수 있음
- 실시간 X, 즉각적으로 데이터 활용이 불가능함



가공된 데이터를 확인하려면 => 배치 처리 시간까지 기다려야 함

2.3 실시간 처리 니즈

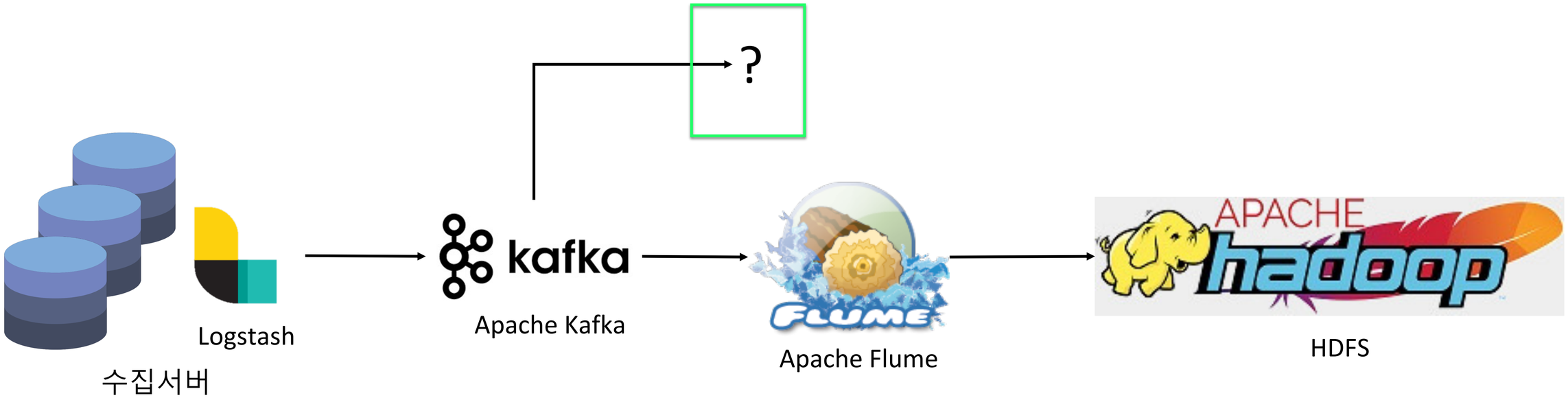
추천 데이터팀에 실시간으로 데이터를 가공/전달하기

- 실시간으로 데이터가 가공 후 전송되어야 함
- 현재 사용하고 있는 FLOW에서 처리가 불가능한 상태

2.3 실시간 처리 니즈

작업을 위한 조건 명세

- Kafka와 연동하여, 실시간으로 consume하여 가공 및 전달 하는 App
- Micro-batch 형태가 아닌, 행단위 처리(row-by-row)
- 부서내 Java/Spring 경험자가 많으므로, 이를 고려
- 추후 트래픽 증가를 고려하여 Scale out(서버 증설)이 쉽게 가능해야 함



2.4 Kafka Streams

DEVIEW 2021

Kafka Streams 정의

- kafka-streams library
- 가벼운 Client library, Java application으로 쉽게 적용이 가능함
- Kafka에 실시간으로 produce/consume
- System Dependency X (e.g. *Linux, Mac OS, windows* 구분 없이 동작)
- Row-by-Row 단위 처리를 지원
- Spring에 연동하여 사용 가능

kafka streams

2.4 Kafka Streams

DEVIEW 2021

Kafka Streams 주요 개념: KStream

- Record stream을 추상화 한 단위
- 각 데이터 레코드는 *Key, Value*형태를 가짐
- *Key, Value* Serialization
- Record는 Key로 update 되지 않음

* 예시: *alice*를 key로하는 2개의 레코드가 순차유입

```
("alice", 1) --> ("alice", 3)
```

* 최종적으로 KStream에는 2개의 레코드 존재
  - alice라는 키를 통해 update연산이 일어나지 않음. 레코드별 독자 처리

처리해야할 데이터(log)는 key별 취합 연산이 아님
=> KStream 사용 가능

실제 어떻게 사용하는지 모르겠어요!

=> DEMO Application 만들기

2.5 Streams Demo App

Demo App 만들기

- Kafka Streams를 활용하여 Java Application 만들기
- Spring Boot Framework(but Not Web APP)
- src-topic에서 데이터를 읽어, dest-topic에 대문자로 변경한 문자열을 produce
- Demo App link!
  * https://github.com/Wshid/DEVIEW-2021-KAFKA-STREAM

2.5 Streams Demo App

Demo App 구동 및 테스트

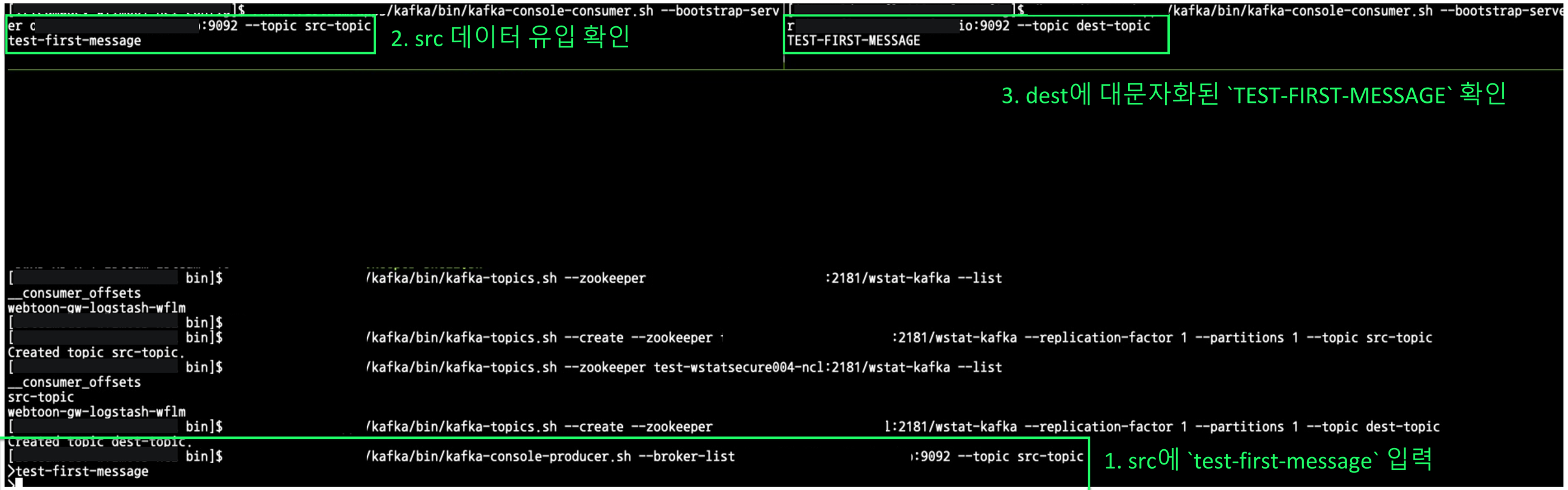
- Apache Kafka 설치 이후, src-topic/dest-topic 생성
- Demo app maven build 및 app 실행(*mvn clean package, java –jar ...*)
- kafka 설치시 포함된 console-producer, console-consumer 사용
  * 문자열 유입 및 토픽 내 데이터 확인

```
# src-topic produce, 임의의 문자열 유입
$KAFKA_HOME/bin/kafka-console-producer.sh --broker-list test-kafka:9092 --topic src-topic

# src-topic consume, src-topic 데이터 확인
$KAFKA_HOME/apps/kafka/bin/kafka-console-consumer.sh --bootstrap-server test-kafka:9092 --topic src-topic

# dest-topic consume, demo-app에 대해 가공된 데이터 확인
$KAFKA_HOME/apps/kafka/bin/kafka-console-consumer.sh --bootstrap-server test-kafka:9092 --topic dest-topic
```

2.5 Streams Demo App

Demo App 구동 및 테스트

- Demo App을 통해 대문자화 된 문자열 확인



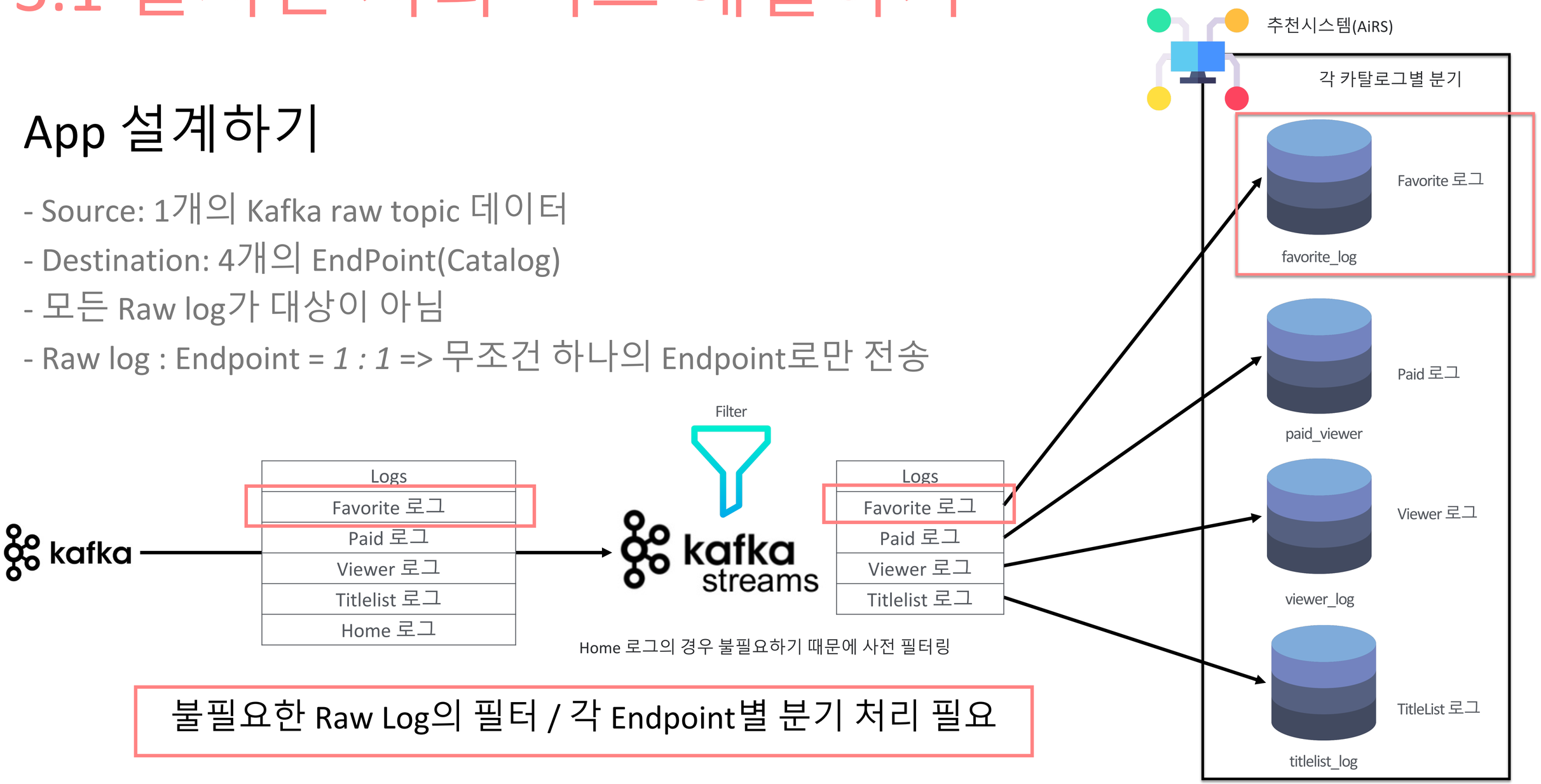
3. Streams App 개발하기

추천 데이터팀에
실시간으로 데이터를 가공/전달하기

3.1 실시간 처리 니즈 해결하기

App 설계하기

- Source: 1개의 Kafka raw topic 데이터
- Destination: 4개의 EndPoint(Catalog)
- 모든 Raw log가 대상이 아님
- Raw log : Endpoint = *1 : 1* => 무조건 하나의 Endpoint로만 전송



Home 로그의 경우 불필요하기 때문에 사전 필터링

불필요한 Raw Log의 필터 / 각 Endpoint별 분기 처리 필요

3.1 실시간 처리 니즈 해결하기

주요 메서드: Filter & Branch

KStream::filter

- Raw log, 사용되지 않는 로그 사전 필터링
  * *key,value* 형태의 인자, boolean을 리턴하는 Predicate를 인수로 받음
    * *KStream<K, V> filter(Predicate<? super K, ? super V> var1);*

KStream::branch

* *KStream : subKstream = 1 : n*
  * subKstream 조건별 배열 Index로 분기, *KStream[]* 형태 반환
    * *KStream -> subKStream[0], subKStream[1], subKStream[2], ...*
  * 조건의 순서 = 우선순위
  * 모든 *case*에 *break*가 존재하는 *switch-case*와 유사

3.1 실시간 처리 니즈 해결하기

DEVIEW 2021

KStream::filter

- Raw log의 KStream을 받아, 사전 정의된 *filterRaw* 메서드 사용

```
@Override
public KStream<String, GwLogServiceVO> process(KStream<String, String> stream) {
    try {
        return stream
            .filter((key, value) -> filterLogic.filterRaw(value)) // Raw filter
            .mapValues(this::translateGwGakRaw)
            .filter((key, value) -> v
            .mapValues(this::translat
            .filter((key, value) -> value != null);

    } catch (Exception e) {
        log.error("Error occurred GwProcessor", e);
    }
    return null;

}
```

- filter가 정의된 서비스의 메서드를 활용
- 불필요한 로그가 후처리되지 않도록 필터링

3.1 실시간 처리 니즈 해결하기

DEVIEW 2021

KStream::branch

- 하나의 KStream에서 subKStream으로 분기

```
@Override
public void process(KStream<String, String> stream) {

    try {
        KStream<String, GwLogServiceVO>[] branches =
            gwRawLogTransformer.process(stream)
                .branch(
                    (key, value) -> (filterLogic.filterViewerLog(value.getTitleNo(), value.getEpisodeNo(), value.getRawType(), value.getRequest())),
                    (key, value) -> (filterLogic.filterPaidLog(value.getTitleNo(), value.getEpisodeNo(), value.getRequest())),
                    (key, value) -> (filterLogic.filterFavoriteLog(value.getRequest())),
                    (key, value) -> (filterLogic.filterTitlelistLog(value.getTitleNo(), value.getEpisodeNo(), value.getRequest(), parsingLogic.getAirsSessionId(value.getRequest()))),
                    (key, value) -> true
                );

        branches[BRANCH_VIEWER].foreach((key, value) -> this.processUserViewer(value));
        branches[BRANCH_PAID].foreach((key, value) -> this.processPaidViewer(value));
        branches[BRANCH_FAVORITE].foreach((key, value) -> this.processFavorite(value));
        branches[BRANCH_TITLELIST].foreach((key, value) -> this.processTitlelist(value));
        branches[BRANCH_DEFAULT].foreach((key, value) -> {
            log.debug("Not Matched AiRS key : {}, value : {}", key, value);
        });
    } catch (Exception e) {
        log.error("Cannot branch AirsProcessor, {}", e);
    }

}
```

각 subKstream 조건 정의. KStream[] 형태 return

- KStream[] 형태로 분리된 subKStream을 활용
- subKStream 인덱스별 각 개별 비즈니스 로직 수행

실시간으로 정상적인

(하나의 앱에서) 로그 가공/전달

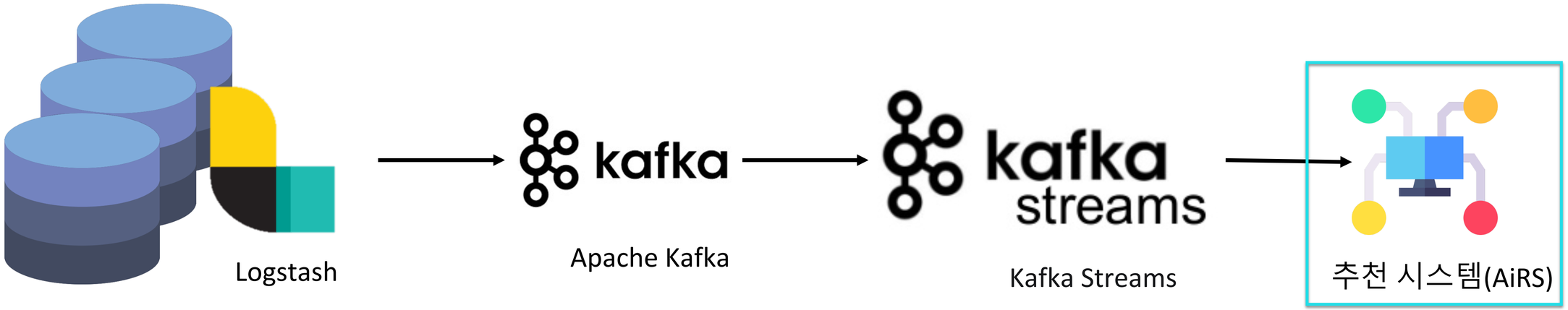
4. Streams App 확장하기

실시간 로그 처리 니즈의 증가?

4.1 Multiple Application Architecture

여러 Stream 니즈 처리하기

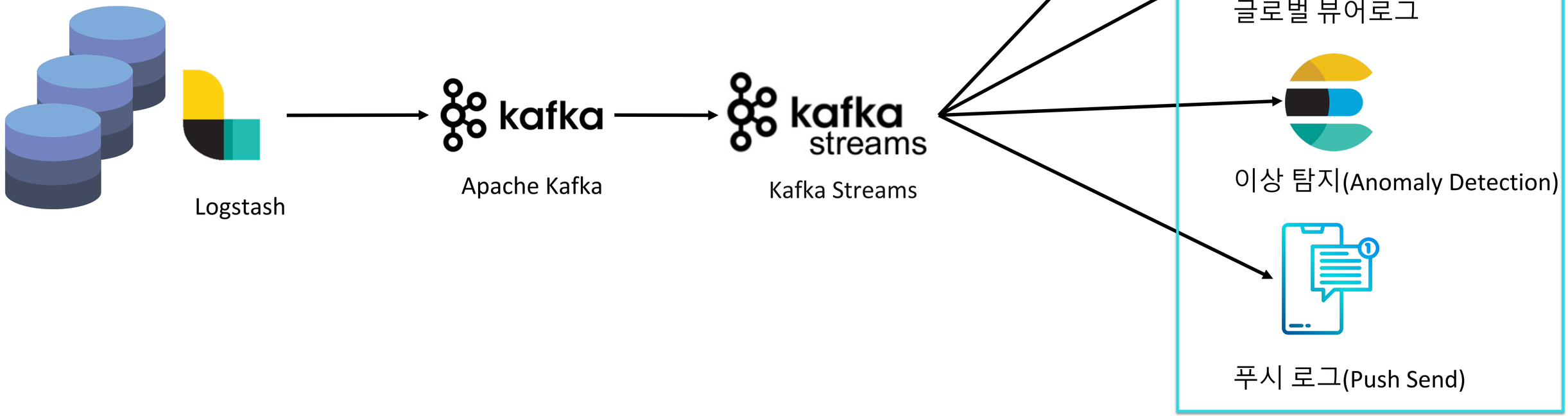
- 실시간으로 처리해야할 다양한 니즈 유입
  - 비정상 로그 탐지, 조회수 관련 로그 전송, 푸시 발송 로그, …
- 기존 Realtime FLOW(AS-IS)
  - 하나의 App만 실행 가능한 구조



4.1 Multiple Application Architecture

여러 Stream 니즈 처리하기

- 성격이 다른 Multiple Sub Application으로 구조 개선 필요
- 개선 Realtime FLOW(TO-BE)



하나의 Streams App에서
Multiple Sub Streams App로 구성하기

4.1 Multiple Application Architecture

DEVIEW 2021

Multiple Application Architecutre 요구 사항

- 서로 다른 니즈 처리를 위해, 여러 App(Sub-App)별로 개별 동작 필요
- 동일한 Source topic consume
- Source topic의 하나(1)의 Raw log는 다수의 App(n)에서 자유롭게 처리 가능
  * App별로 raw log의 offset을 서로 구애받지 않고, 독자적으로 처리

동일한 Topic Consume + App별 다른 Offset 처리
=> Consumer Group 분리 필요

4.1 Multiple Application Architecture

DEVIEW 2021

Consumer Group과 application.id

- Kafka *group.id* = Kafka Streams *application.id*
  * Kafka group.id = Consumer group의 고유 값
- 한 Topic에 대해 *application.id*가 다르면, consume되는 offset도 다름

application.id

(Required) The application ID. Each stream processing application must have a unique ID. The same ID must be given to all instances of the application. It is recommended to use only alphanumeric characters, `.` (dot), `-` (hyphen), and `_` (underscore). Examples: `"hello_world"`, `"hello_world-v1.0.0"`

This ID is used in the following places to isolate resources used by the application from others:

- As the default Kafka consumer and producer `client.id` prefix
- As the Kafka consumer `group.id` for coordination
- As the name of the subdirectory in the state directory (`state.dir`)
- As the prefix of internal Kafka topic names

Sub App별로 Raw Log를 활용하기 위해
=> application.id를 구분하기

4.1 Multiple Application Architecture

DEVIEW 2021

WORK #1: application.id 분리 설정

- 모든 App에서 동일한 Kafka 관련 설정, *application.id*만 다르게
- 공통점 관리: Kafka 공통 설정 *Map*(변수) 생성
- 각 Sub-app별로 *Map* load, *application.id* 추가
  * 이후 *Bean*으로 생성하여 *StreamsBuilder*로 사용

공통 설정 Map 정의(Kafka의 기본 설정)

```
// KstreamCommon 메서드
public Map<String, Object> kStreamsConfigProps() {
    Map<String, Object> props = new HashMap<>();
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getBootstrapServers());
    props.put(StreamsConfig.STATE_DIR_CONFIG, statePath);
    props.put(StreamsConfig.METRICS_RECORDING_LEVEL_CONFIG, metricsRecordingLevel);
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
```

sub-app별 application.id 추가

```
@Bean
public StreamsBuilderFactoryBean airsStreamsBuilderFactoryBean() {
    Map<String, Object> config = streamsBuilderConfig.kStreamsConfigProps();
    config.put(StreamsConfig.APPLICATION_ID_CONFIG, applicationIdAirs);
    return new StreamsBuilderFactoryBean(new KafkaStreamsConfiguration(config));
}
```

sub-app별 application.id 추가

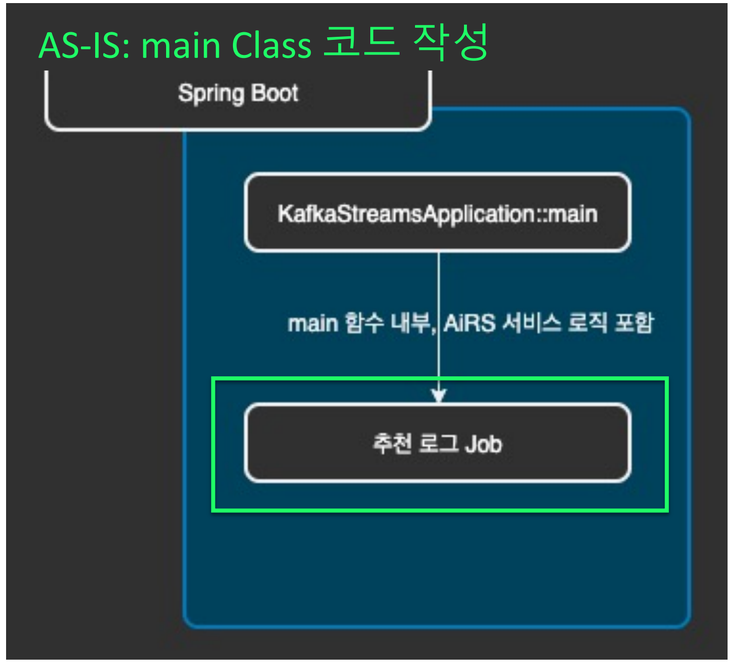
```
@Bean
@Qualifier("gwStreamsBuilderFactoryBean")
public StreamsBuilderFactoryBean gwStreamsBuilderFactoryBean() {
    Map<String, Object> config = streamsBuilderConfig.kStreamsConfigProps();
    config.put(StreamsConfig.APPLICATION_ID_CONFIG, applicationIdGw);
    return new StreamsBuilderFactoryBean(new KafkaStreamsConfiguration(config));
}
```

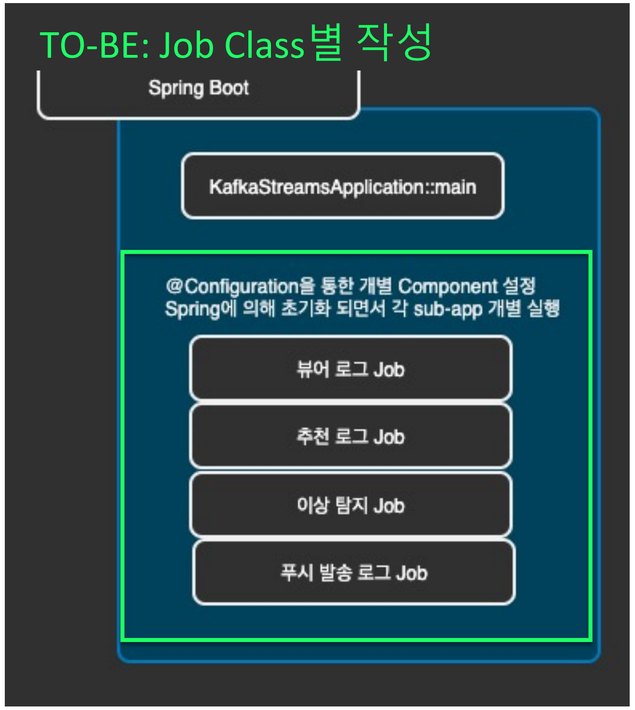
4.1 Multiple Application Architecture

DEVIEW 2021

WORK #2: Architecture Refactoring

- AS-IS: 하나의 App, main Class에 모든 코드 작성
- TO-BE: 여러개의 App, Job Class 분리
- 각 Job 에서 App 구동시 필요한 요소만 개별 관리
  * 각 App에 의존적인 내용만 Job에 정의
  * *Variables, Beans, Service Logic, ...*





4.1 Multiple Application Architecture

DEVIEW 2021

WORK #2: Architecture Refactoring

- App별 Job Class 정의
- 내부 *@Value*, *@Bean*, Service Logic 정의

```
@ConditionalOnExpression("'${jobs}'.contains('GwJob')")
public class GwJob {

    @Autowired
    StreamsBuilderConfig streamsBuilderConfig;

    @Value("${streaming.application.gw.application.id}")
    private String applicationIdGw;
    @Value("${source.stat.topics.gw.gak-raw}")
    private String gwGakRawTopicName;

    @Bean
    @Qualifier("gwStreamsBuilderFactoryBean")
    public StreamsBuilderFactoryBean gwStreamsBuilderFactoryBean() {
        Map<String, Object> config = streamsBuilderConfig.kStreamsConfigProps();
        config.put(StreamsConfig.APPLICATION_ID_CONFIG, applicationIdGw);
        return new StreamsBuilderFactoryBean(new KafkaStreamsConfiguration(config));
    }
```

```
public class AirsJob {

    @Autowired
    StreamsBuilderConfig streamsBuilderConfig;

    @Value("${streaming.application.airs.application.id}")
    private String applicationIdAirs;
    @Value("${source.stat.topics.gw.gak-raw}")
    private String gwGakRawTopicName;

    @Bean
    public StreamsBuilderFactoryBean airsStreamsBuilderFactoryBean() {
        Map<String, Object> config = streamsBuilderConfig.kStreamsConfigProps();
        config.put(StreamsConfig.APPLICATION_ID_CONFIG, applicationIdAirs);
        return new StreamsBuilderFactoryBean(new KafkaStreamsConfiguration(config));
    }
```

Sub App별로 의존적인 요소만 별도 관리 => 유지보수성 확보

4.2 Scale out

Scale out의 필요성

- 웹툰의 성장 -> 로그의 유입 증가
- 실시간 처리 니즈, Sub-app의 증가 -> 사용하는 리소스 증가
- 성능 리팩토링 진행 후에도, 리소스 한계 도달

  * *Cpu rate, Load average, Memory, Network bandwith..*

- Scale-out(서버 증설)필요
- Micro Service가 아닌 Monolithic Service의 Scale out

4.2 Scale out

Scale out 방법

- 신규 서버에서 Streams app 수행하면, 쉽게 Consumer Instance 할당 가능
- But, *num.stream.threads*의 유의
  * Stream app에서 task를 처리할 때 사용할 thread 수(default: 1)
  * Task 수는 Partitions 수에 의존적
  * Thread : Task(Partition) = 1 : n
  * Recommended settings: *Total tasks(partitions) / Stream server counts*
    * Thread를 가장 최적으로 활용
- Scale out이후, 각 Stream서버별로 partition 할당 여부 확인

4.2 Scale out

Scale out 방법

* Stream 서버 구동시, 자동으로 consumer group에 포함되어 partition을 할당

```
    8, webtoon-gw-logstash-8, webtoon-gw-logstash-6
5297 2021-08-19 13:56:12 [INFO ](StreamThread.java:221) stream-thread [gw-gak-log-realtime-2cdcabc2-0f09-4569-a6af-e5dcb33ca0f0-StreamThread-1] State transition from STARTING to PARTITIONS_ASSIGNED
5298 2021-08-19 13:56:12 [INFO ](StreamsRebalanceListener.java:88) stream-thread [gw-gak-log-realtime-2cdcabc2-0f09-4569-a6af-e5dcb33ca0f0-StreamThread-1] partition assignment took 26 ms.
5299         currently assigned active tasks: [0_16, 0_0, 0_18, 0_2, 0_4, 0_6, 0_8, 0_10, 0_12, 0_14]
5300         currently assigned standby tasks: []
5301         revoked active tasks: []
5302         revoked standby tasks: []
```

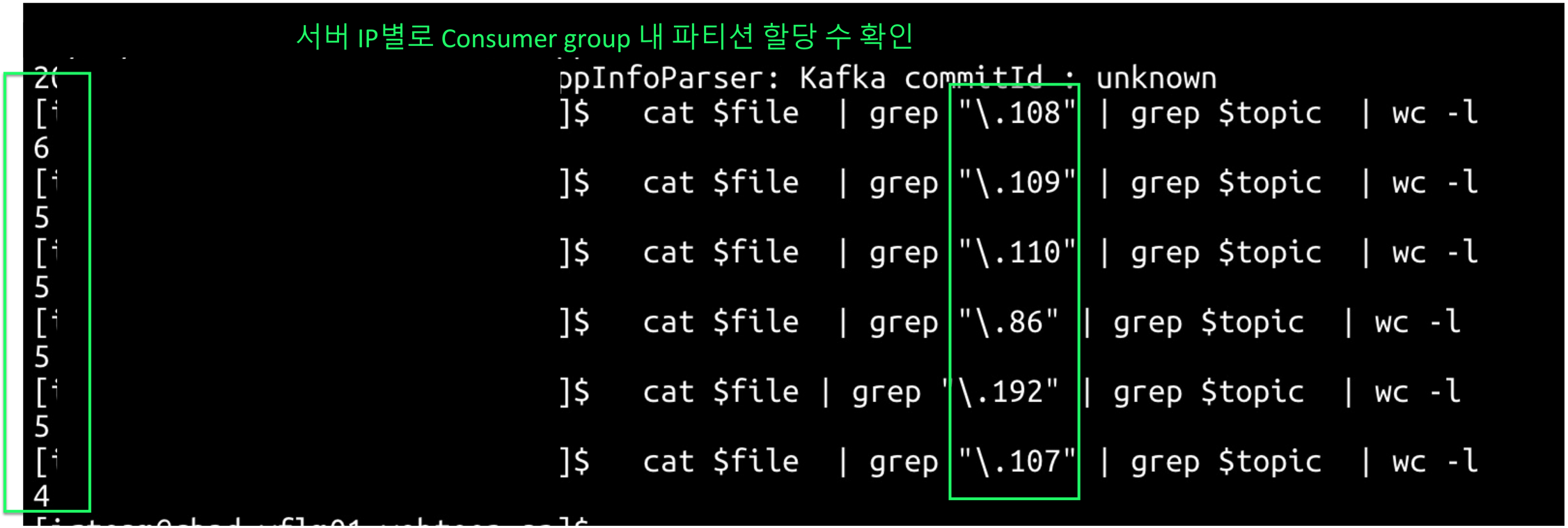
* Scale-out 이후, 각 Stream 서버별 파티션 할당을 확인하는 스크립트 (*kafka-consumer-groups.sh* 활용)

```
topic ="foo-topic"
file="consumer_group_lags.log"
consumer_group="test-group"
$KAFKA_HOME/bin/kafka-consumer-groups.sh --bootstrap-server  test-kafka:9092 –describe –group $consumer_group >  $file
cat $file  | grep "\.108" | grep $topic  | wc -l
cat $file  | grep "\.109" | grep $topic  | wc -l
cat $file  | grep "\.110" | grep $topic  | wc -l
```

4.2 Scale out

Scale out 방법

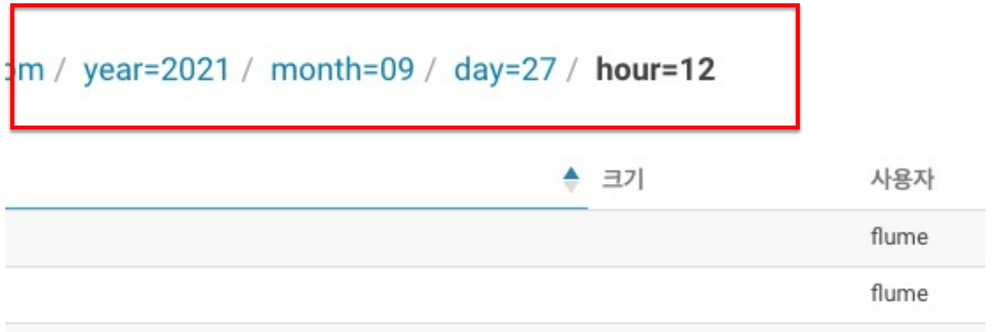
* 스크립트 구동 이후, 서버별로 할당된 파티션의 수를 집계



4.3 Realtime Pipeline

Realtime Pipeline 처리 필요성

- 기존 Batch Pipeline(AS-IS) 처리시 문제
  * Log를 *year, month, day, hour*의 디렉터리 경로로 적재
  * 디렉터리 경로 = Log의 Flume 도착 시간
  * Log가 Flume에 지연 유입 => 경로 잘못 지정

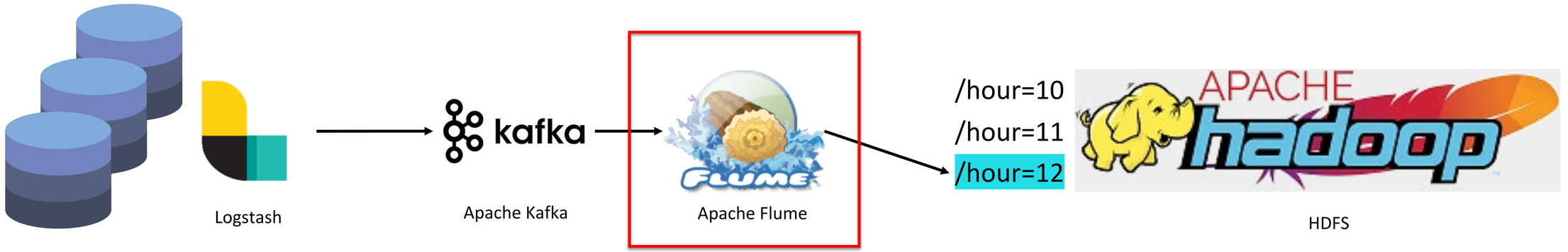


* 로그의 지연 유입 예시

1. *2021.09.27. 11:59.59* 데이터 유입

2. Flume 도착시 *12:00:00*

3. Flume 도착시 *12*시이므로, hour=12로 로그 적재



Logstash | Apache Kafka | Apache Flume | HDFS

- 해결방법: Log를 Flume 도착 시간이 아닌 Log의 *time*필드 활용하기
  - But, *Tsv* Format Log, 정확한 *time*필드를 가져올 수 없음

*time*필드로 분류하려면

log의 스키마 매핑이 필요

Raw Log를 객체 매핑할 수 있는

Custom Java Library + Kafka Streams를 사용

4.3 Realtime Pipeline

Custom Java Library + Kafka Streams를 활용한 실시간 처리

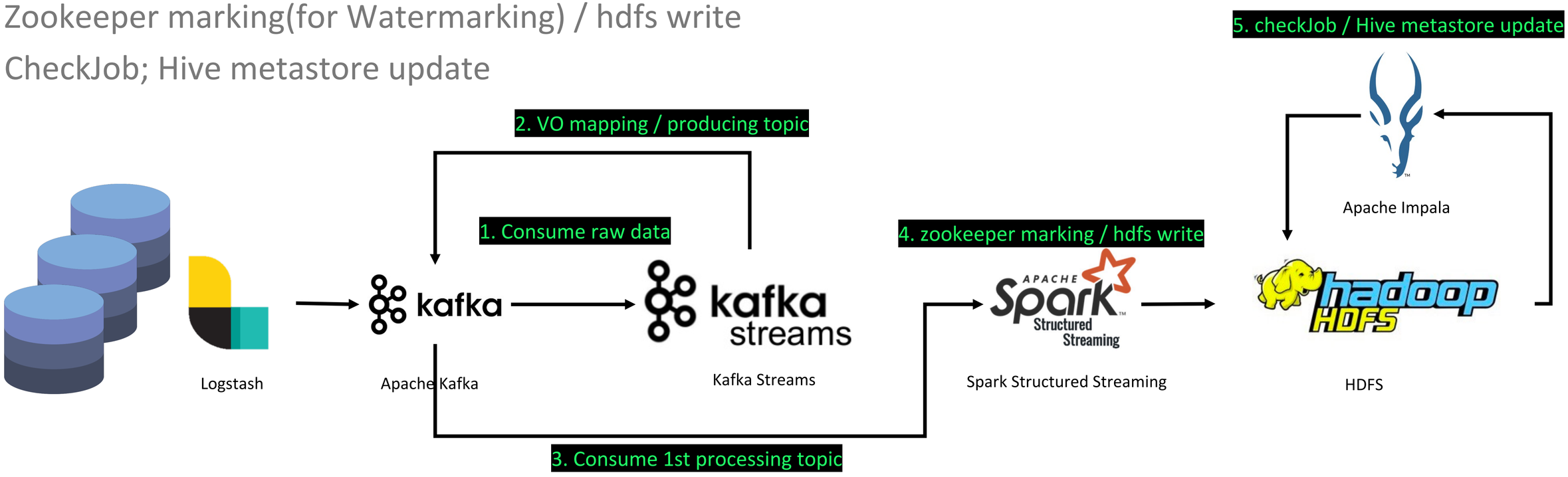
- Custom Java library, 해당 로그의 스키마 관리
  => Tsv raw log -> Java 객체 매핑
  => *time*필드 추출 가능
- Log의 도착 시간 X, log *time*필드로 디렉터리 경로 지정 -> 지연 문제 해결
- Kafka Streams, 실시간으로 가공 데이터 생성 가능
  => Batch 처리보다 빠른 시점에 가공된 데이터 확인 가능

기존 로그 지연 문제 해결
+ 가공된 데이터를 실시간으로 확인 가능

4.3 Realtime Pipeline

실시간 처리 Flow: Realtime Pipeline

1. Consume Raw Data
2. VO mapping / 1st producing topic
3. Consume 1st processing topic
4. Zookeeper marking(for Watermarking) / hdfs write
5. CheckJob; Hive metastore update



DEVIEW 2021

5. Trouble Shooting

5.1 num.stream.threads

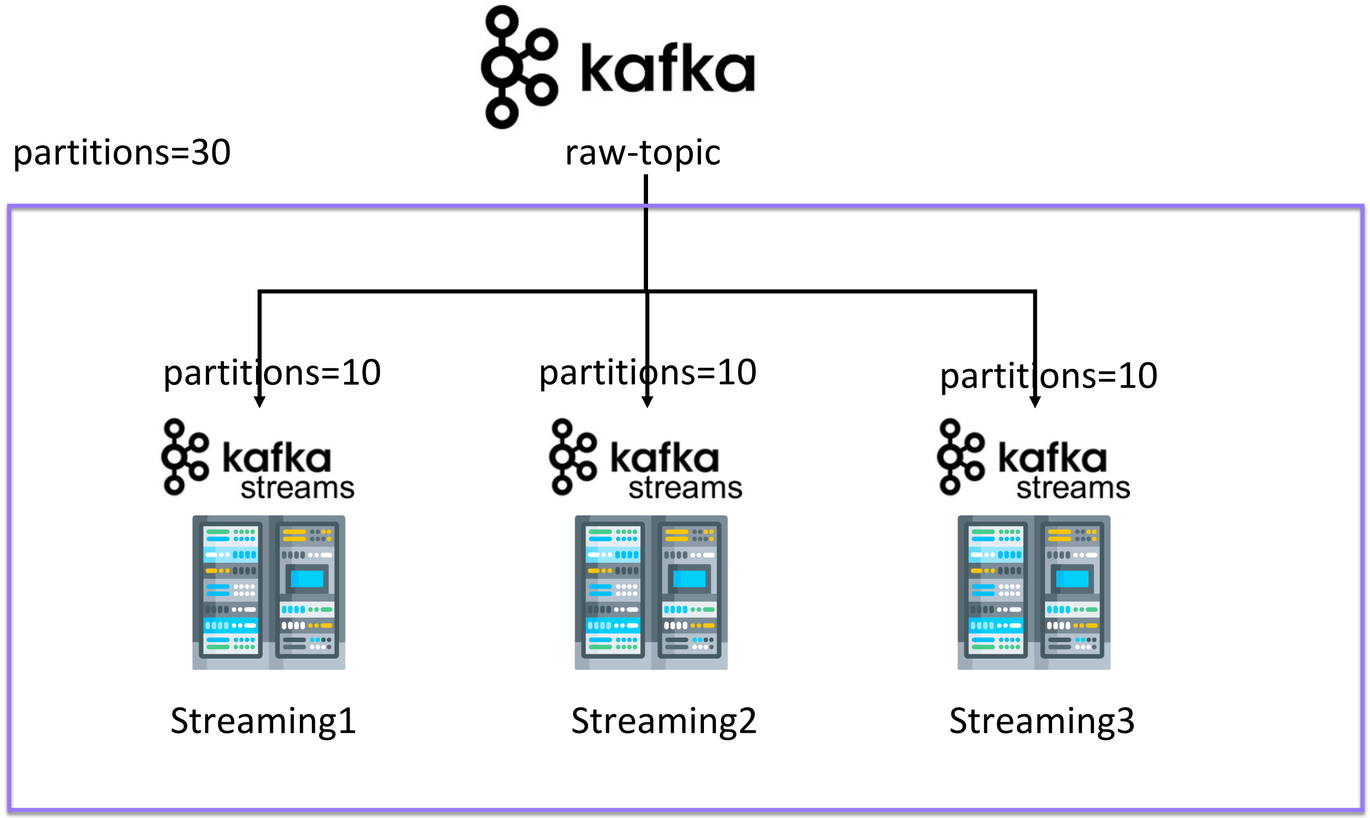
문제 상황

- Scale out을 위한 서버 투입 후 배포시 문제 발생
- 특정 Stream 서버에 Partition이 할당 되지 않음
- 모든 서버에 Partiton이 분배되지 않으면서, Scale out이 되지 않는 상황

5.1 num.stream.threads

문제 상세

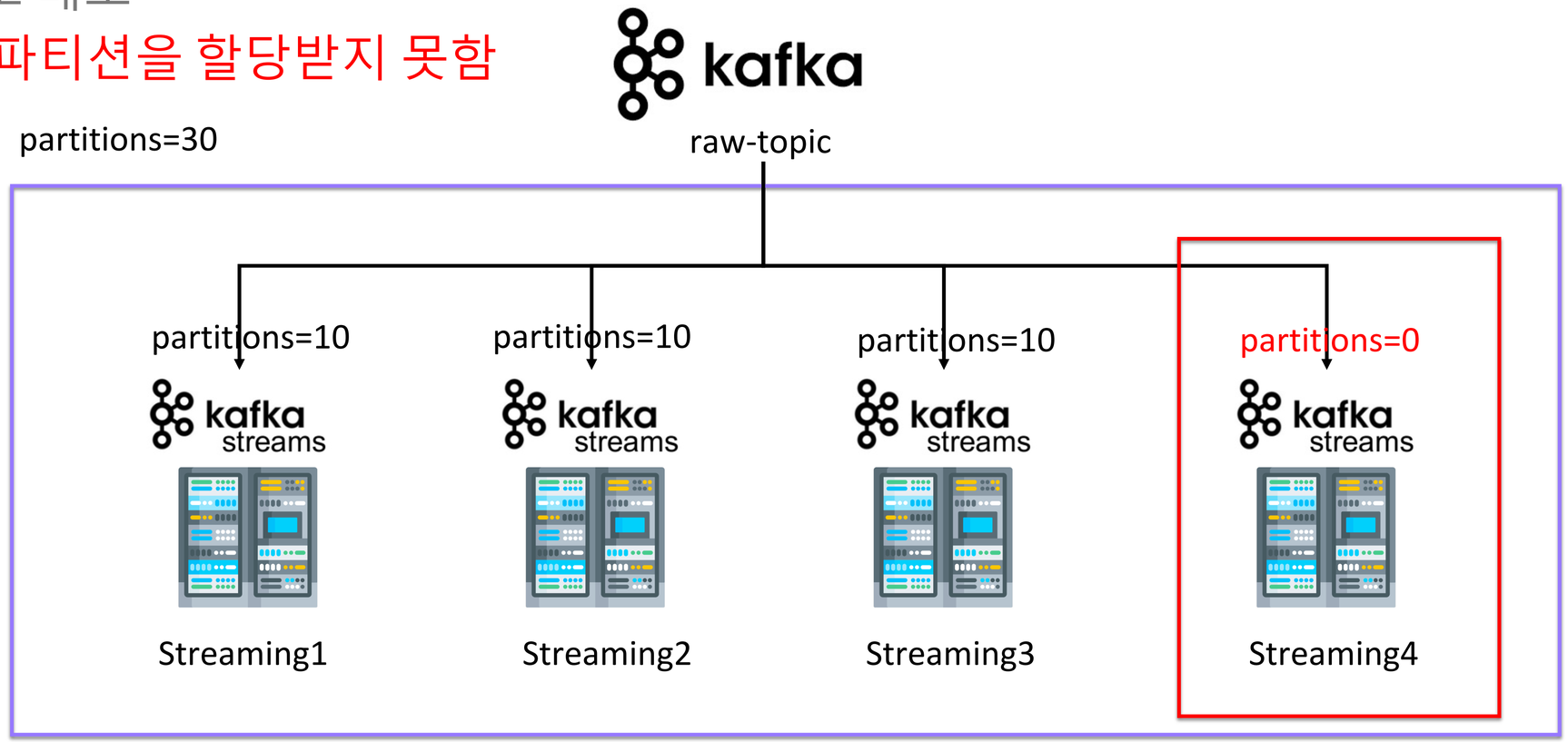
- 기존 3대의 서버로, 30개의 파티션의 topic을 consume하는 상황



5.1 num.stream.threads

문제 상세

- 신규 서버 1대 투입
- 1, 2, 3, 4 순서로 배포
- 4번 서버에서 파티션을 할당받지 못함



배포의 마지막 서버에서
파티션을 제대로 할당 받지 못함

5.1 num.stream.threads

문제 해결

- num.stream.threads 옵션 문제(*num.stream.threads=10*)
  * Stream app에서 task를 처리할 때 사용할 thread 수(default: 1)
  * Threads : Task(Partitions) = *1 : n*
- 문제 상황: Total Server threads (10 *4) > Topic partitions(10 * 3)
- 먼저 배포된 *3*대의 서버, 각 threads=10, 서버별 10개 이상의 partition 처리 가능
  * Stream 서버 *3*대, *30*개의 partition 모두 할당
- 마지막 서버 배포시, 처리할 잔여 partition이 없어, 할당받지 못함

num.stream.threads

This specifies the number of stream threads in an instance of the Kafka Streams application. The stream processing code runs in these thread. For more information about Kafka Streams threading model, see Threading Model.

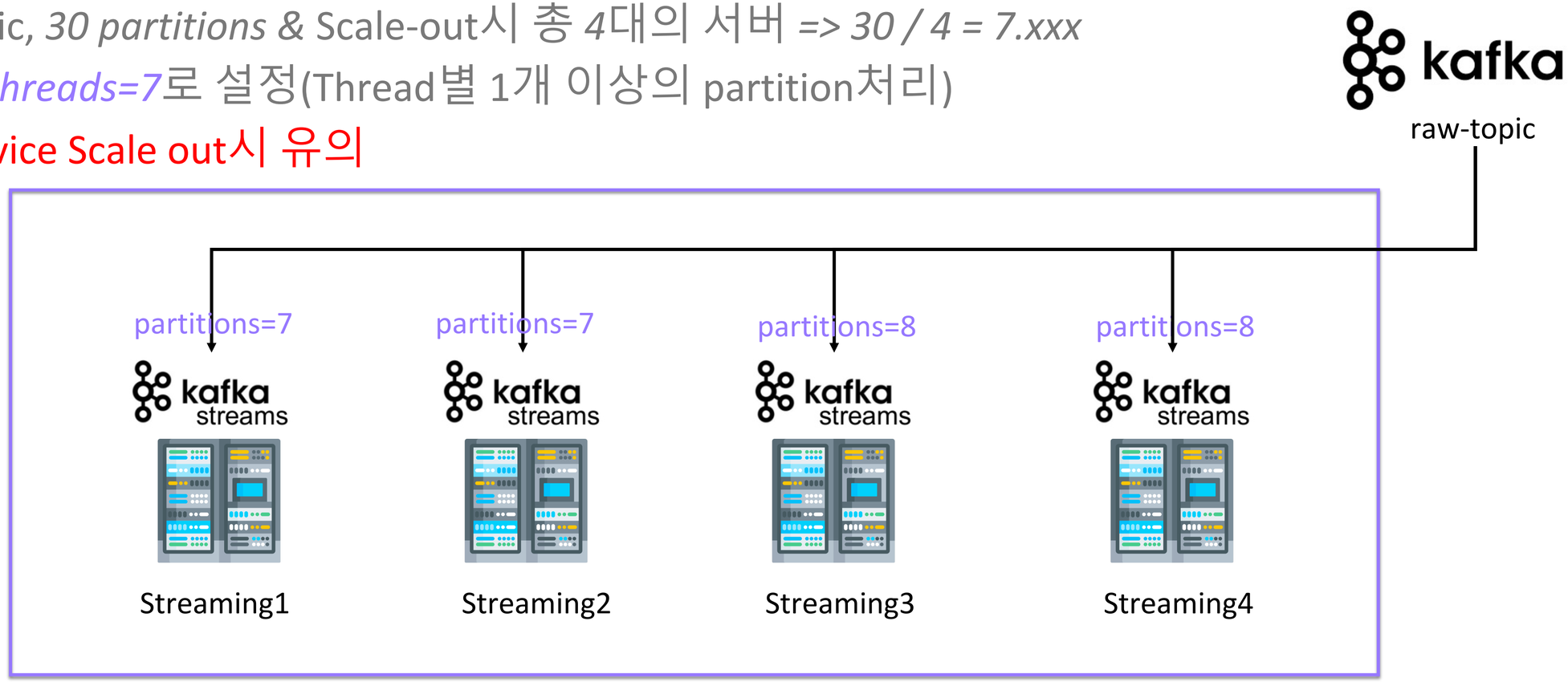
```
kafka:
  bootstrap-servers: test-kafka:9092, test-kafka:9092
  metrics-recording-level: INFO
  auto-offset-reset: latest
  num-stream-threads: 10
  commit-interval-ms: 5000
```

각 서버에 적용된 설정(10)

5.1 num.stream.threads

문제 해결

- *num.stream.threads = Total tasks(partitions) / Stream server counts*
  * e.g. 대상 Topic, *30 partitions* & Scale-out시 총 *4*대의 서버 *=> 30 / 4 = 7.xxx*
  * *num.stream.threads=7*로 설정(Thread별 1개 이상의 partition처리)
- Monolithic Service Scale out시 유의



5.2 Kafka Consumer Lag

DEVIEW 2021

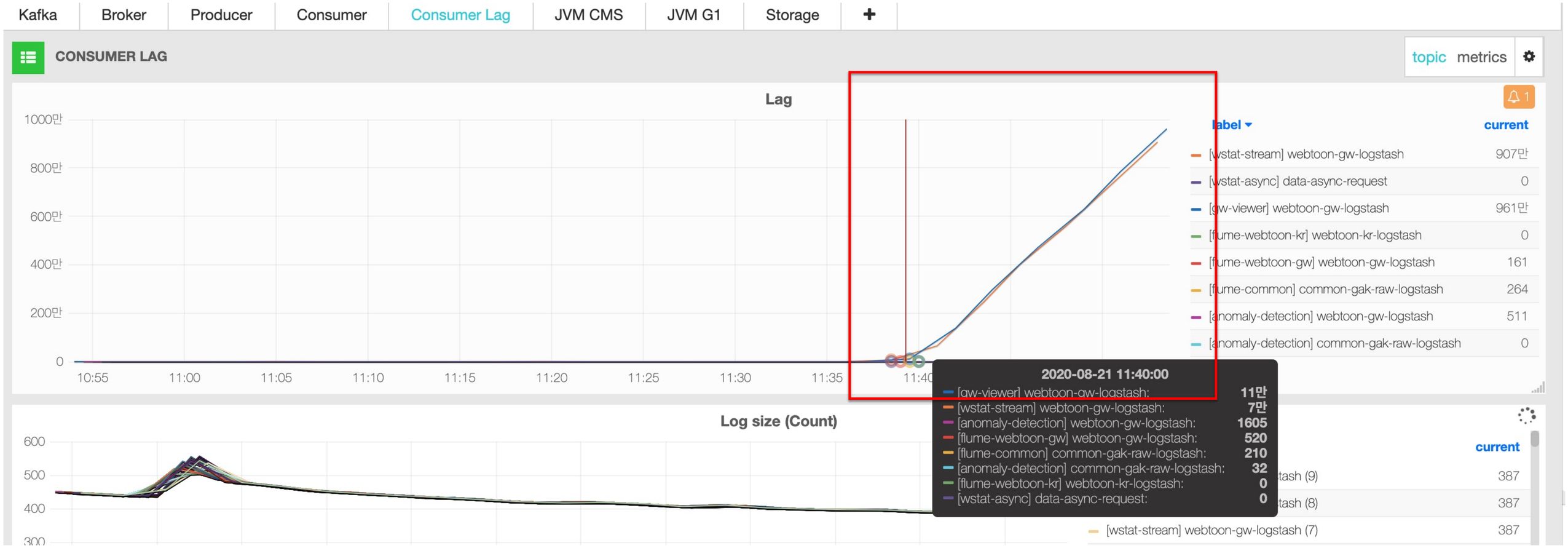
문제 상황

- 특정 Consumer Group의 Lag가 급속도로 증가
- Lag가 증가하기 시작한 시점의 서버별 상태 확인 필요
- 문제 트래킹 경험 공유

5.2 Kafka Consumer Lag

문제 상황

- 11:40분경부터 특정 Consumer Group의 Lag가 증가하는 상황

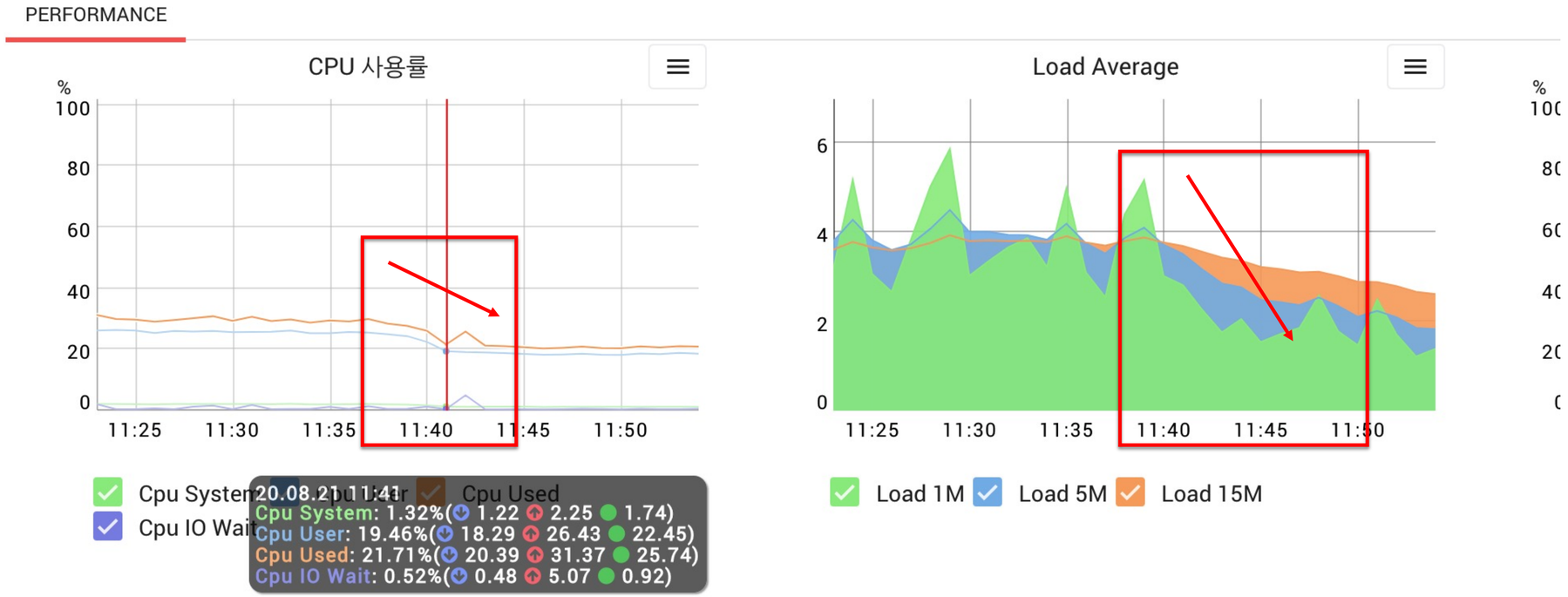


5.2 Kafka Consumer Lag

문제 추적 – 서버 상태 확인

- cpu & load average 등 지표 감소

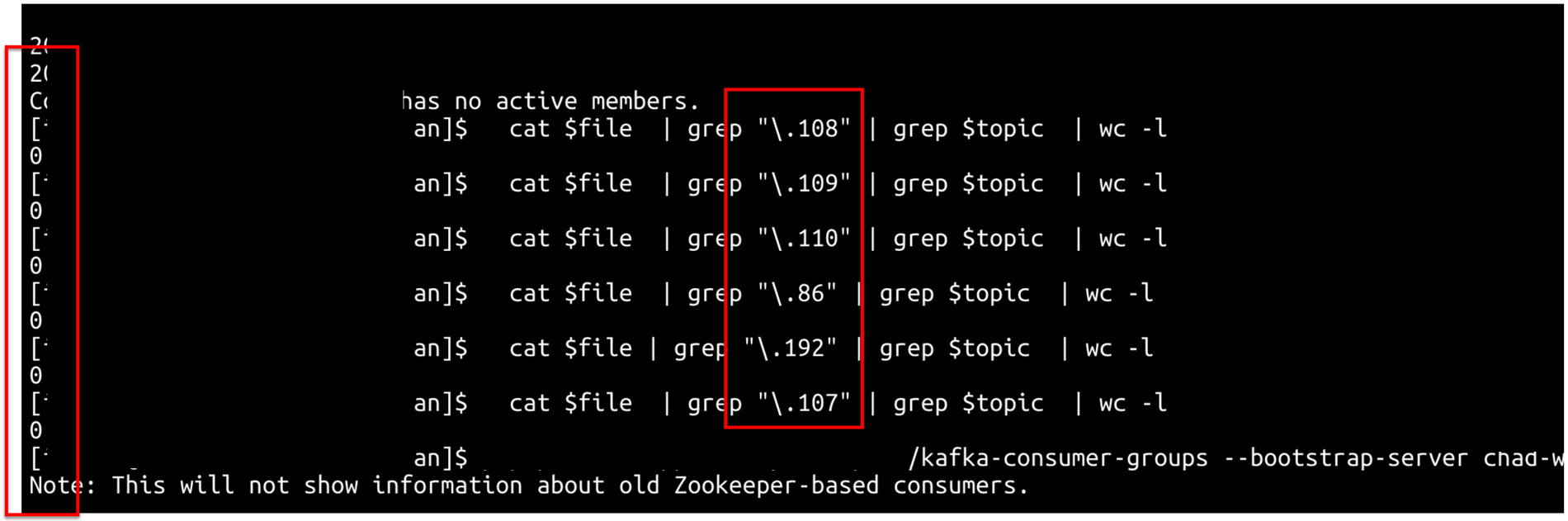
  => Kafka Streams 일부 sub-app에서 문제 발생 추측



5.2 Kafka Consumer Lag

문제 추적 – Consumer Group 상태 확인

- Consumer group내 Stream 서버별 파티션 할당 확인 스크립트 수행(CHAP.4.2. Scale out)



- 모든 서버에서 파티션 수가 모두 0으로 집계
- 해당 Sub-app이 down되었음을 인지

5.2 Kafka Consumer Lag

DEVIEW 2021

문제 추적 – 서버별 로그 확인

- 비정상 로그 유입으로 인해, Sub-app의 shutdown 로그 확인
- ERROR 발생으로 App Down
- App 재실행시에도 동일한 문제 발생

```
2020-08-21 11:40:21 [INFO ](LogContext.java:346) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] State transition from PARTITIONS_REVOKED to PARTITIONS_A
SSIGNED
2020-08-21 11:40:21 [INFO ](LogContext.java:351) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] partition assignment took 14 ms.
        current active tasks: [0_1, 0_2, 0_4, 0_6, 0_8, 0_9, 0_11, 0_13, 0_15, 0_19, 0_21, 0_23, 0_24, 0_27, 0_29]
        current standby tasks: []
        previous active tasks: [0_1, 0_2, 0_19, 0_4, 0_21, 0_6, 0_9, 0_29]
2020-08-21 11:40:21 [INFO ](LogContext.java:346) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] State transition from PARTITIONS_ASSIGNED to RUNNING
2020-08-21 11:40:22 [ERROR](LogContext.java:301) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] Failed to process stream task 0_15 due to the following
error:
2020-08-21 11:40:22 [ERROR](LogContext.java:306) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] Encountered the following error during processing:
2020-08-21 11:40:22 [INFO ](LogContext.java:346) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] State transition from RUNNING to PENDING_SHUTDOWN
2020-08-21 11:40:22 [INFO ](LogContext.java:336) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] Shutting down
2020-08-21 11:40:22 [INFO ](LogContext.java:341) [Producer clientId=gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1-producer] Closing the Kafka producer with timeoutMill
is = 9223372036854775807 ms.
2020-08-21 11:40:22 [INFO ](LogContext.java:346) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] State transition from PENDING_SHUTDOWN to DEAD
2020-08-21 11:40:22 [INFO ](LogContext.java:346) stream-client [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb]State transition from REBALANCING to ERROR
2020-08-21 11:40:22 [WARN ](LogContext.java:236) stream-client [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb]All stream threads have died. The instance will be in error state and sh
ould be closed.
2020-08-21 11:40:22 [INFO ](LogContext.java:336) stream-thread [gw-viewer-c4ae63d0-cd1c-4472-a97a-bdd719e14dbb-StreamThread-1] Shutdown complete
                                                                                                                                                1079,1        Bot
```

5.2 Kafka Consumer Lag

문제 해결

- SPoF(Single Point of Failure) Fix : 비정상 로그 유입에 대한 단일 실패 지점 존재
- 관련 예외 처리 및 배포 진행
- 이후 정상적으로 각 서버별 파티션 할당 확인

```
20/08/21 13:25:57 INFO utils.AppInfoParser: Kafka commitId : unknown
[                                an]$   cat $file  | grep "\.108" | grep $topic  | wc -l
6
[                                an]$   cat $file  | grep "\.109" | grep $topic  | wc -l
5
[                                an]$   cat $file  | grep "\.110" | grep $topic  | wc -l
5
[                                an]$   cat $file  | grep "\.86" | grep $topic  | wc -l
5
[                                an]$   cat $file | grep "\.192" | grep $topic  | wc -l
5
[                                an]$   cat $file  | grep "\.107" | grep $topic  | wc -l
4
[                                an]$
```

5.3 Consumer Resource Balancing

DEVIEW 2021

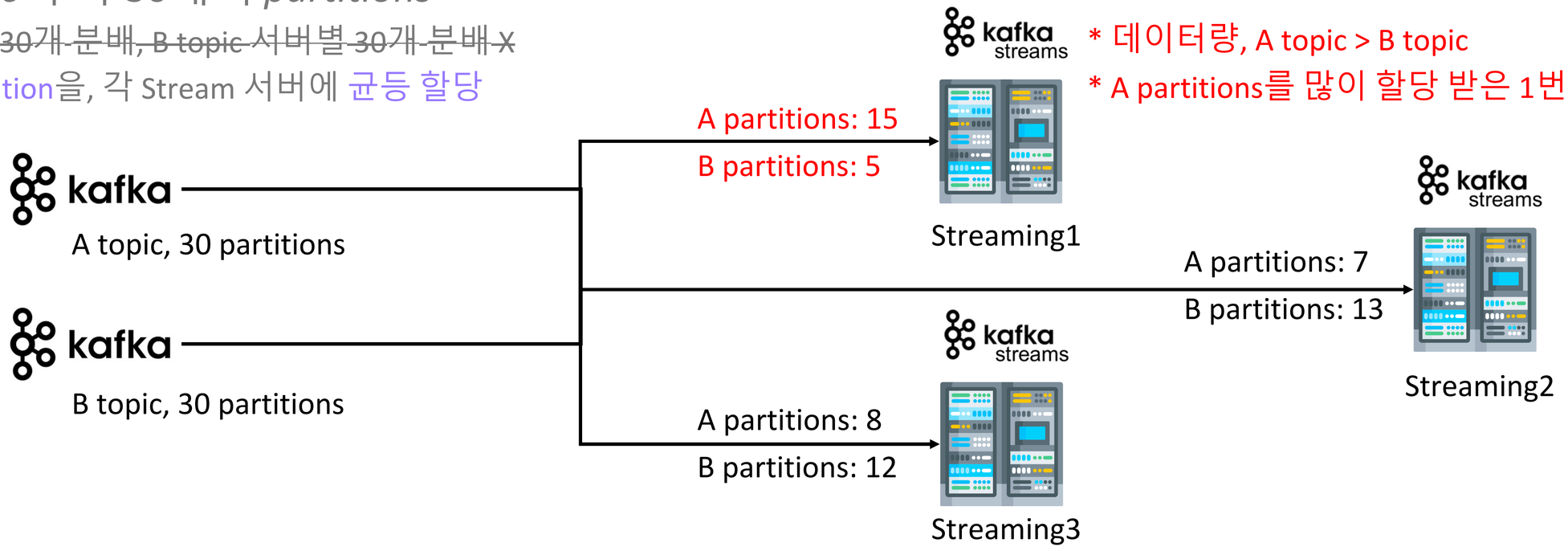
문제 상황

- 특정 서버에서만 Load Average / CPU가 높게 측정
- PeakTime이 아닌, 평균적으로 계속 리소스 사용량이 높게 유지
- Load Average가 높게 측정 => 실제 처리량이 높음

5.3 Consumer Resource Balancing

문제 파악

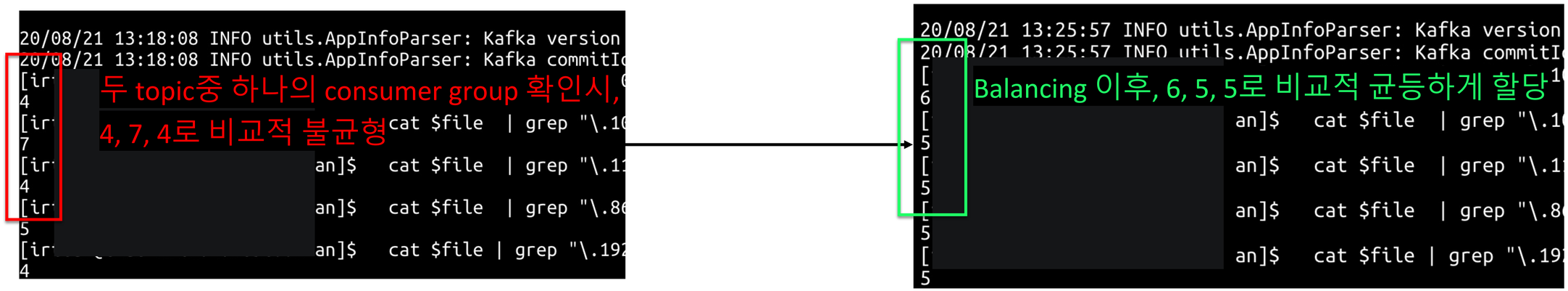
- 하나의 Sub-app에서 토픽 2개를 consume => 동일한 *appplication.id*, 2개의 topic consume
- StreamPartitionAssignor; Multiple topic consume, 각 topic별 partition 분배 X
- 전체 Topic의 Partition을 전체 Stream 서버에 균등 분배
- e.g. *A, B topic*의 각 30개의 *partitions*
  * ~~A topic 서버별 30개 분배, B topic 서버별 30개 분배 X~~
  * 총 60개의 Partition을, 각 Stream 서버에 균등 할당



5.3 Consumer Resource Balancing

문제 해결

- 하나의 *application.id*로 여러 Topic을 consume & Topic별 데이터량 차이가 큰 상황
- Consumer Resource Balancing 필요
- 여러 Topic Consume, Topic별 데이터량(무게) 차이가 클 경우,
  * 토픽별로 partition이 Stream 서버별 유사하게 분배되었는지 확인
- 이슈 발생시, 가장 불균형한 두 노드 재배포



- PartitionAssignor 정책을 새로 정의 / Topic별 개별 Sub-app구성

6. Conclusion

6.1 Future Tasks

GROW-UP

실시간 지표 제공

- 실시간으로 가공하여 적재하는 데이터를 분석가들에게 지표화하여 제공
- 기존 1차 가공 외 2,3차와 같은 지표성 데이터 제공
- 보다 기민하게 데이터 분석 요소로 활용 가능
- BI Tool을 연동하여 다양한 인사이트를 얻을 수 있음

KStream, KTable, Join

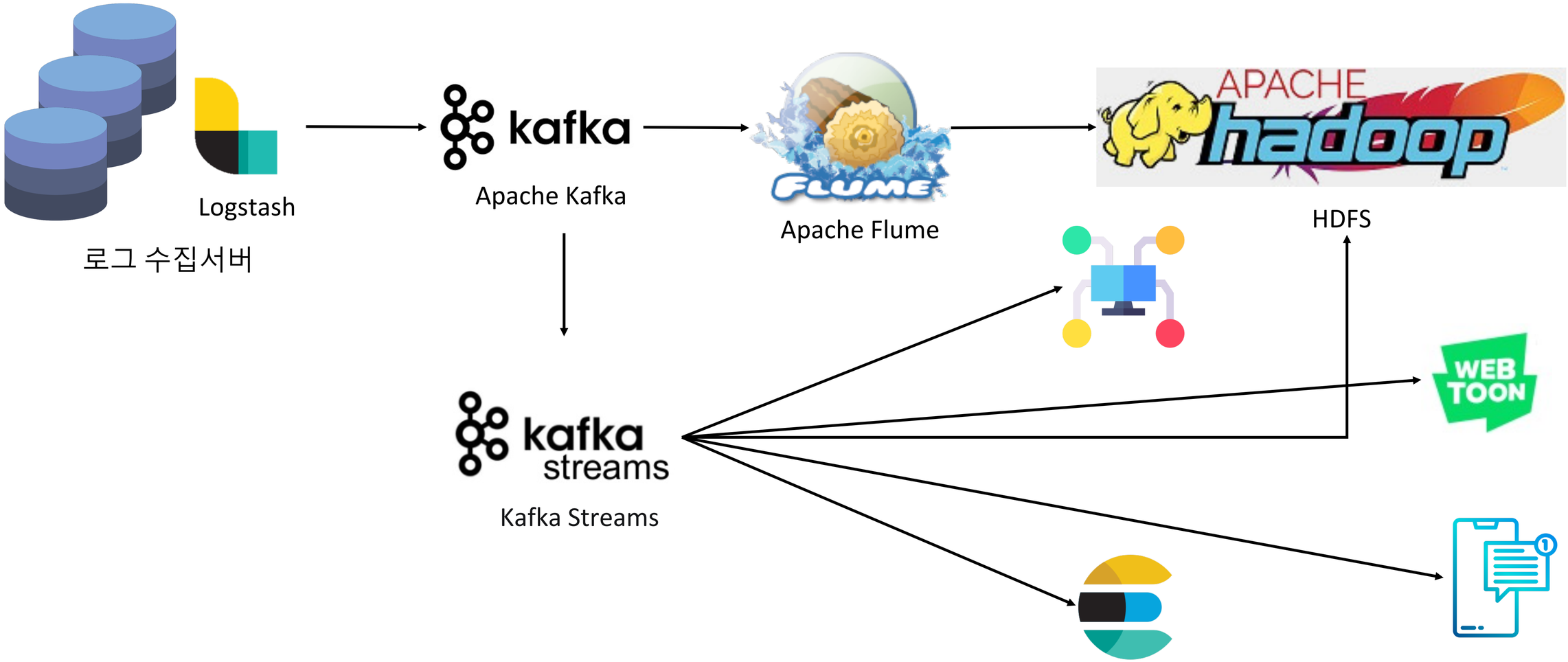
- 상태 기반의 처리 추가
  * e.g. *key 기준 update, 패턴 지표화, ...*
- 실시간 처리되는 데이터에 메타 데이터와 같은 다양한 데이터 Join
- 행동 기반의 데이터 처리 가능

6.2 Summary

AS-IS: Batch 처리만 가능했던 Pipeline

6.2 Summary

TO-BE: 실시간 처리도 가능한 Pipeline



6.2 Summary

DEVIEW 2021

Keywords

- WEBTOONS' BIGDATA
- Kafka; Producer, Consumer, Topic, ...
- Consumer & Lags
- Kafka Streams with Spring boot
- Multiple Streams Applications, *application.id*
- Monolithic Service Scale out
- Trouble Shooting

Reference

* https://www.redhat.com/ko/topics/integration/what-is-apache-kafka
* https://www.popit.kr/kafka-%EC%9A%B4%EC%98%81%EC%9E%90%EA%B0%80-%EB%A7%90%ED%95%98%EB%8A%94-%EC%B2%98%EC%9D%8C-%EC%A0%91%ED%95%98%EB%8A%94-kafka/
* https://docs.confluent.io/platform/current/streams/concepts.html#kstream
* https://kafka.apache.org/21/javadoc/org/apache/kafka/streams/kstream/KStream.html#branch-org.apache.kafka.streams.kstream.Predicate...-
* https://www.popit.kr/kafka-consumer-group/
* https://jaceklaskowski.gitbooks.io/mastering-kafka-streams/content/kafka-streams-internals-StreamThread.html
* https://kafka.apache.org/10/documentation/streams/developer-guide/config-streams.html#num-stream-threads
* https://velog.io/@ehdrms2034/%EC%B9%B4%ED%94%84%EC%B9%B4-%EC%8A%A4%ED%8A%B8%EB%A6%BC%EC%A6%88-DSL-%EA%B0%9C%EB%85%90
* https://medium.com/streamthoughts/understanding-kafka-partition-assignment-strategies-and-how-to-write-your-own-custom-assignor-ebeda1fc06f3

Thank You