DEVIEW 2020

네이버 페이 배송 EDA 전환기

이명현 NAVER
윤지수 NAVER FINANCIAL

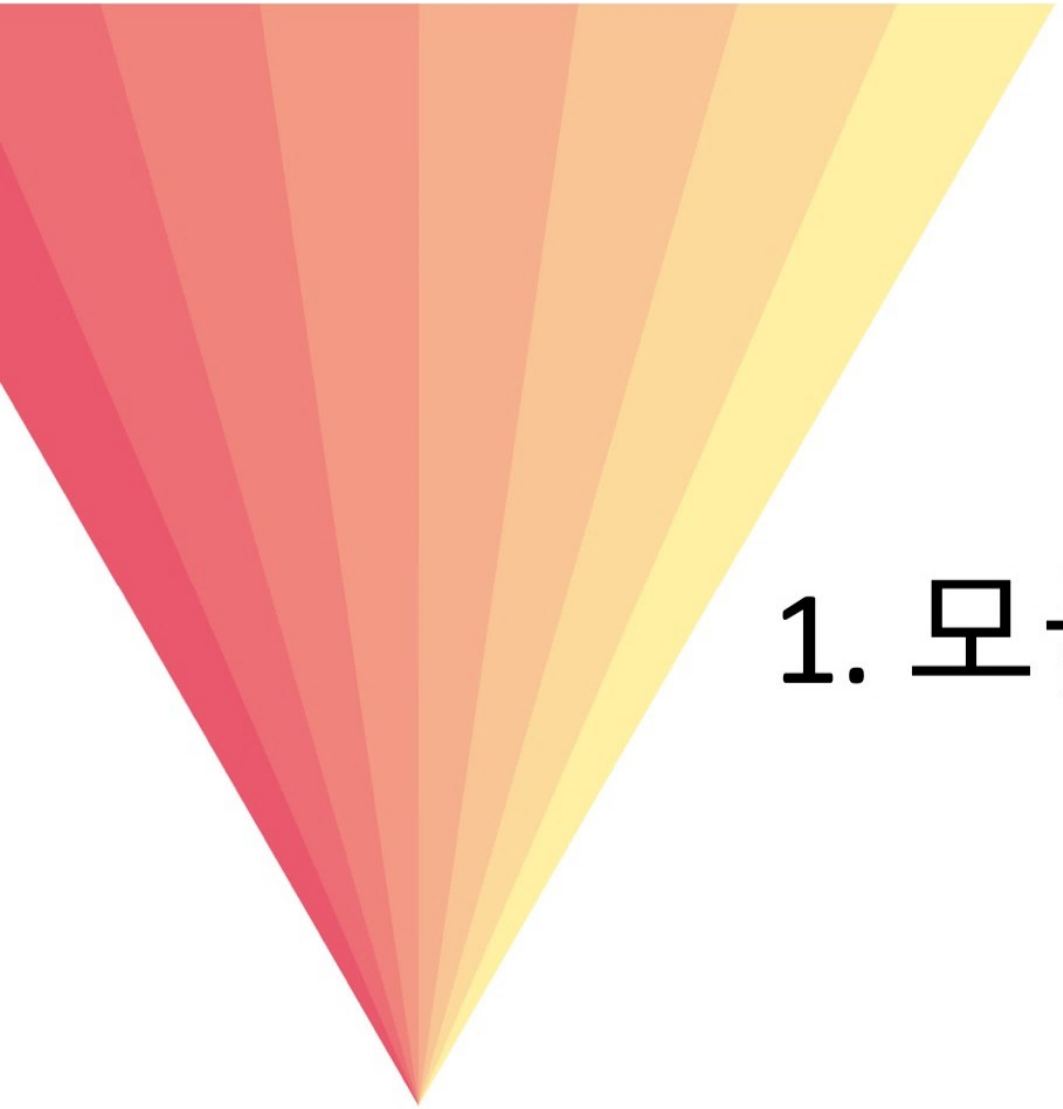
Plasma 프로젝트

- 네이버 페이 아키텍처 개선 프로젝트
- 주문형 페이 개발자 + 서비스 플랫폼 개발자
- 배송 모듈 분리 이야기

프로젝트 이야기 : 페이 왕국 🏰 주문 가문 👨‍👨‍👦
배송 용사 독립기🦸

아키텍처 & 기술 : 네이버 페이 배송 EDA 전환기

CONTENTS



DEVIEW 2020

1. 모듈 분리

1.1 주문형 페이에서 배송의 역할

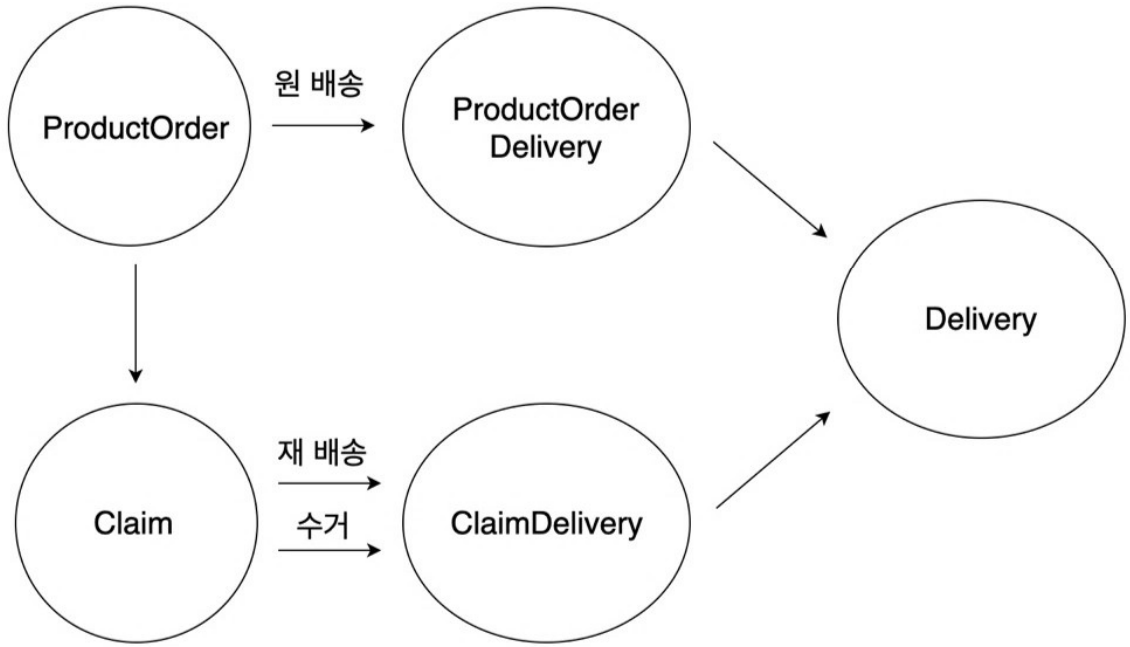
DEVIEW 2020

주문 도메인의 바깥 연결 관계

- 판매자의 상품 발송 등록
- 교환 / 반품 상품 수거 등록
- 교환 상품 재 배송 등록

주요 비즈니스

- 판매자 발송 처리
- 배송 상태 트래킹 연동

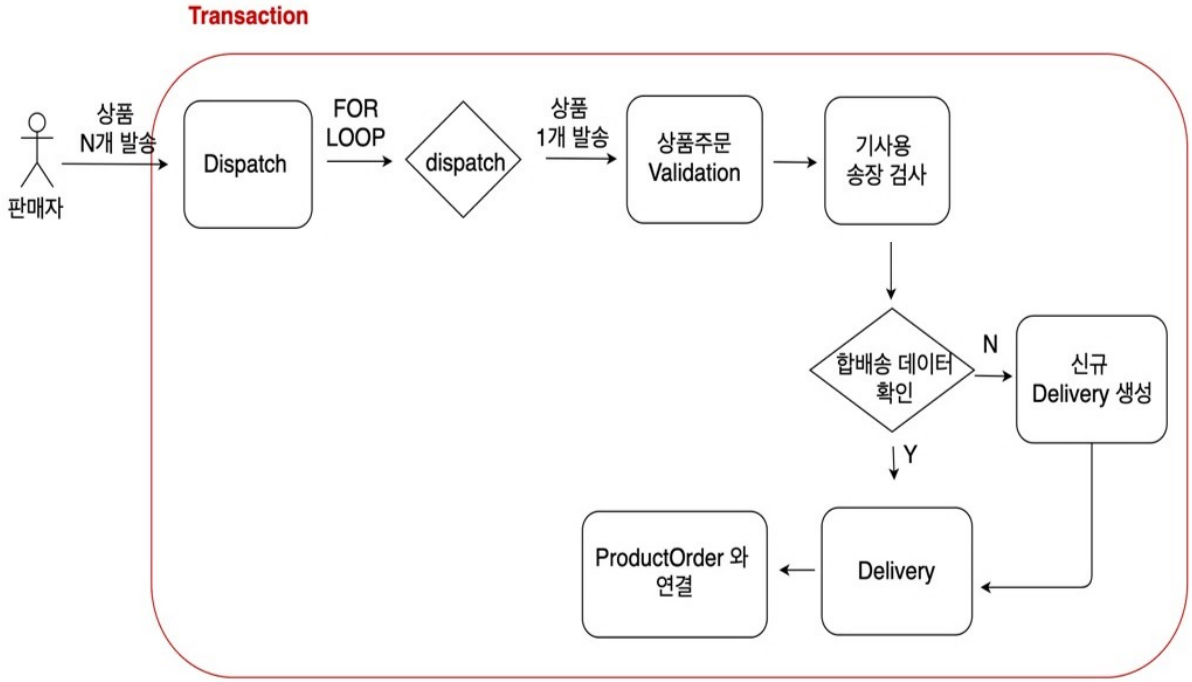


1.1 판매자 상품 발송 등록

DEVIEW 2020

판매자 Action 단위 강한 일관성

- N 개 전체가 하나의 Transaction
- 주문 – 배송 코드 강 결합

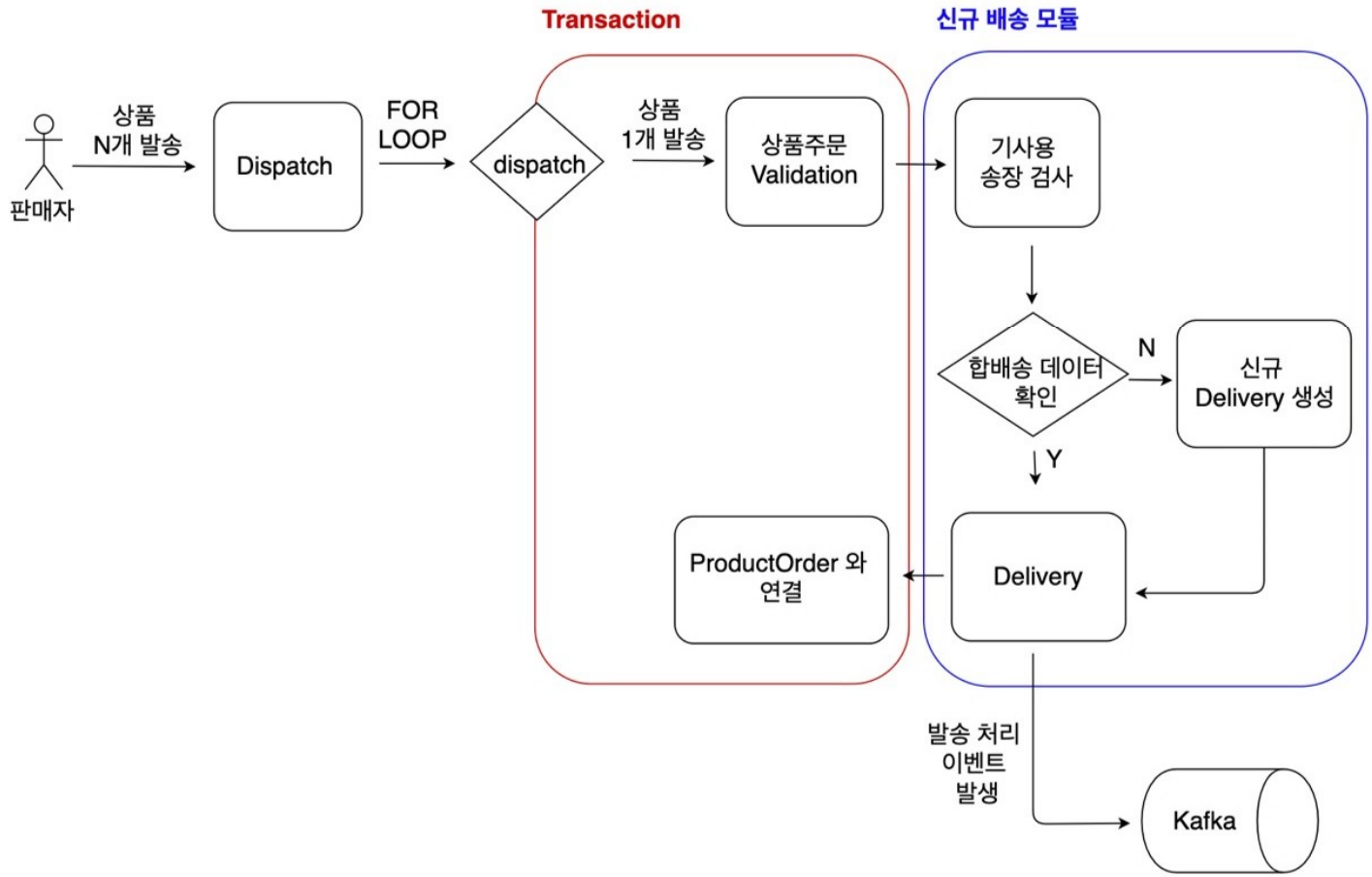


1.1 판매자 상품 발송 등록

DEVIEW 2020

서비스 분리

- 비즈니스 로직 분리
- 저장소 분리
- Transaction 분리
- 이벤트 발행

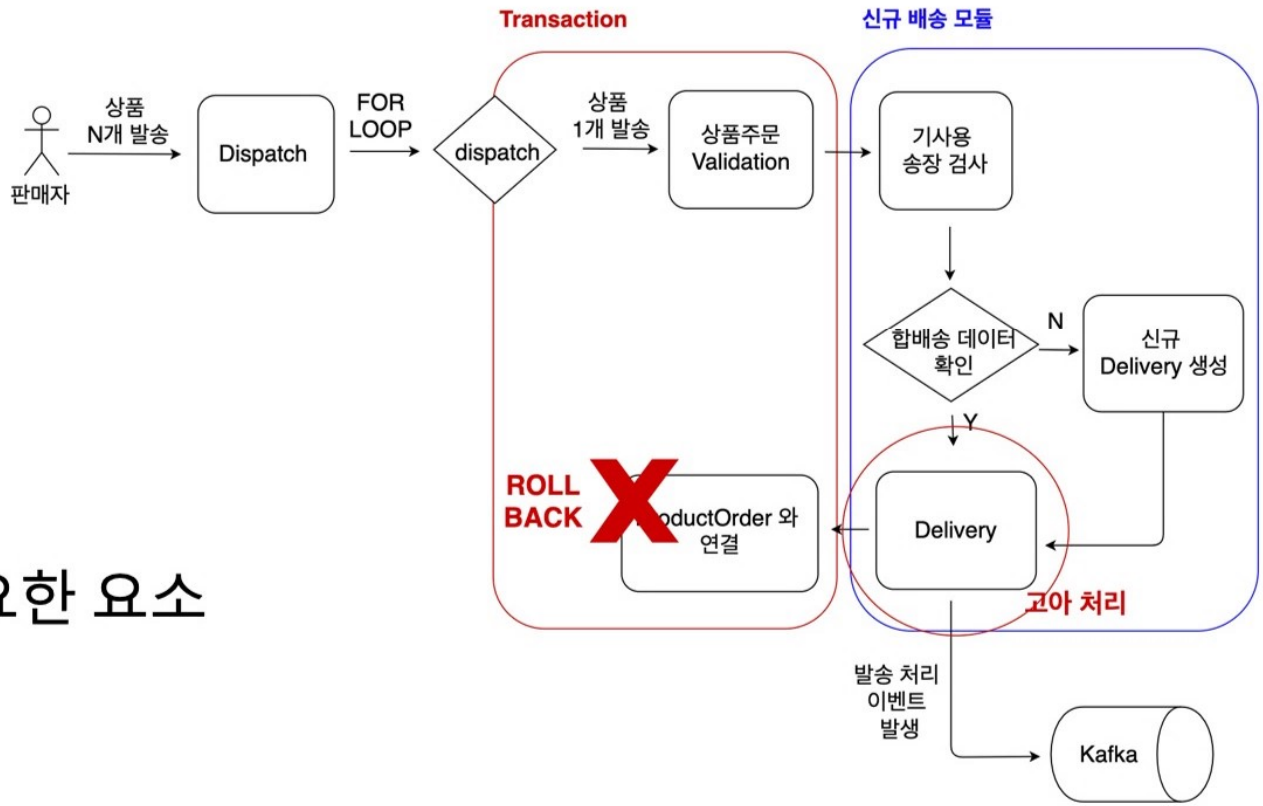


1.1 판매자 상품 발송 등록

DEVIEW 2020

Transaction 실패

- 상품주문 -> 연결정보 롤백
- 생성된 배송 정보는 고아 처리



일관성 처리는
모듈 경계 식별 및 분리 결정에 중요한 요소

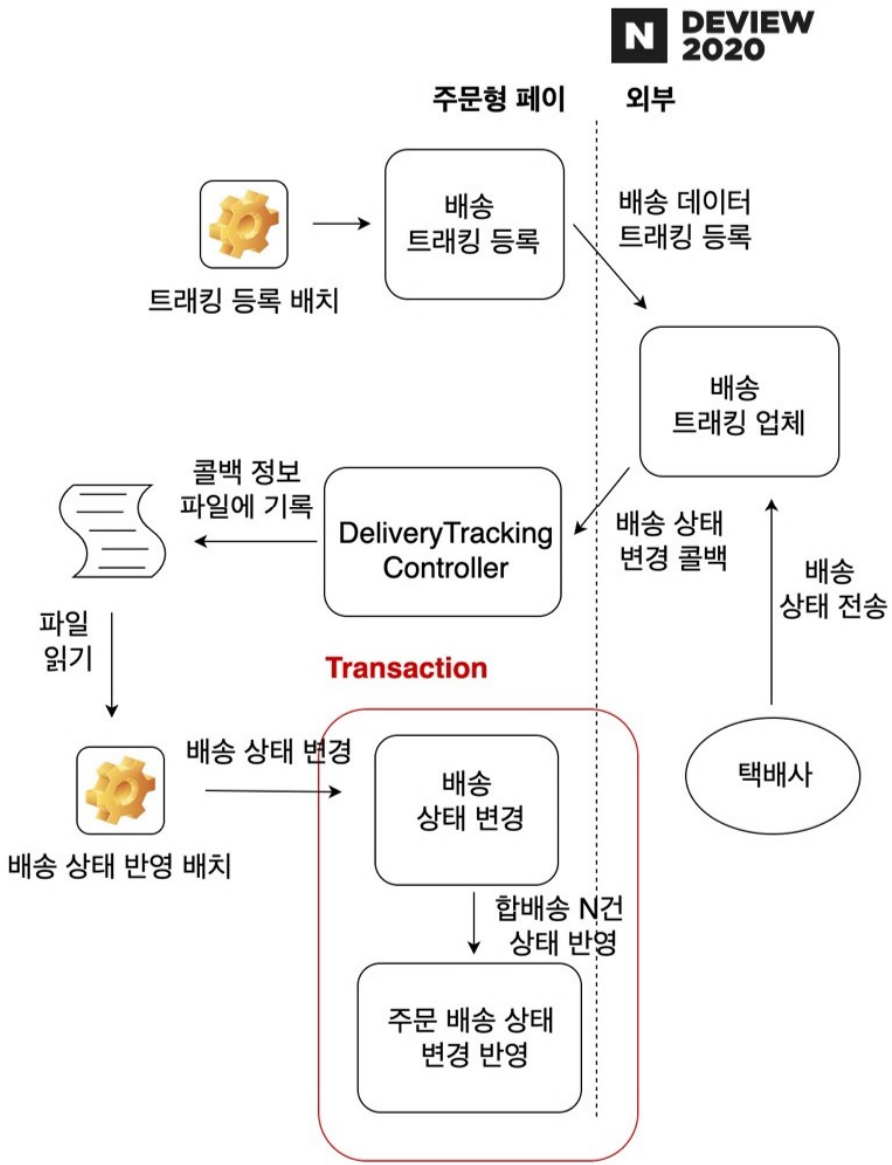
1.2 배송 상태 트래킹

배송 트래킹 등록

- 발송 처리 된 배송 정보
- 외부 트래킹 서비스에 등록

트래킹 콜백

- 택배사 배송 상태 변경에 따른 트래킹 콜백
- 배치로 배송 상태 변경 반영
- 배송 상태 변경 + 주문 배송 상태 변경 반영 (Transaction)

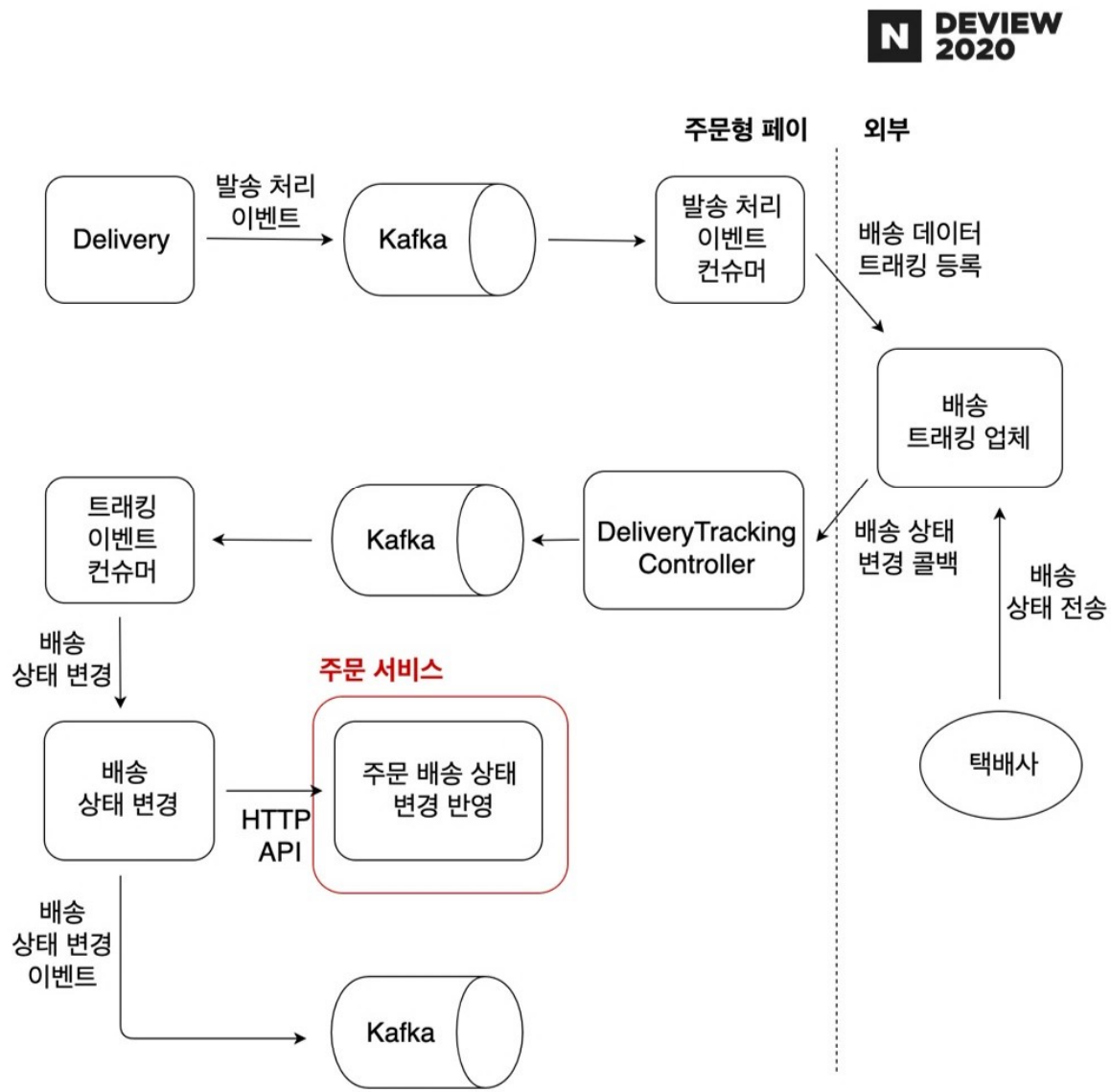


1.2 배송 상태 트래킹

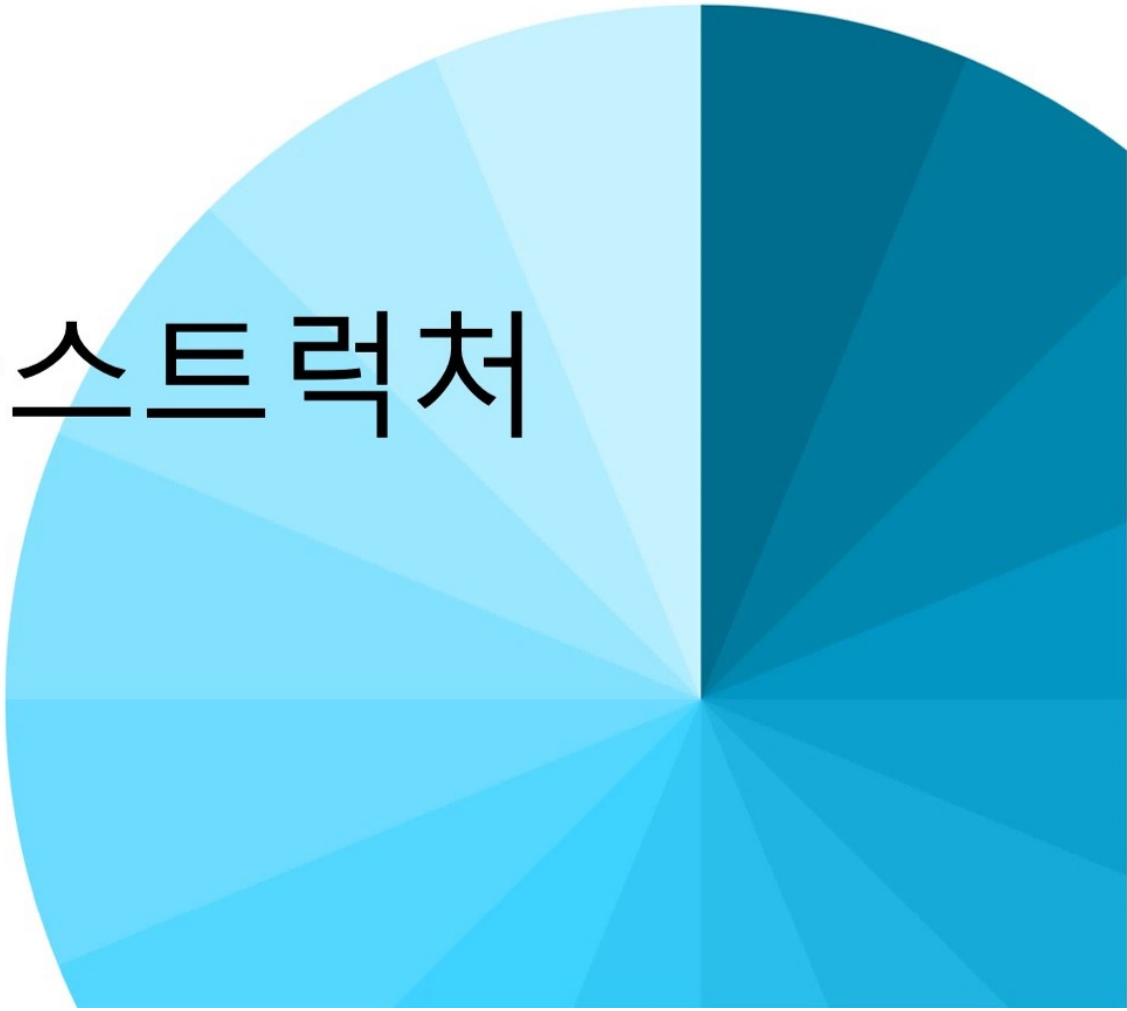
Event Processing

- 배치 -> 이벤트 프로세싱
- 실시간 처리 증가
- 병렬 처리 용이
- Transaction 분리

주문 모듈 개선시
주문 변경 반영까지
이벤트 프로세스 처리



2. 인프라스트럭처



2.1 컨테이너

DEVIEW 2020

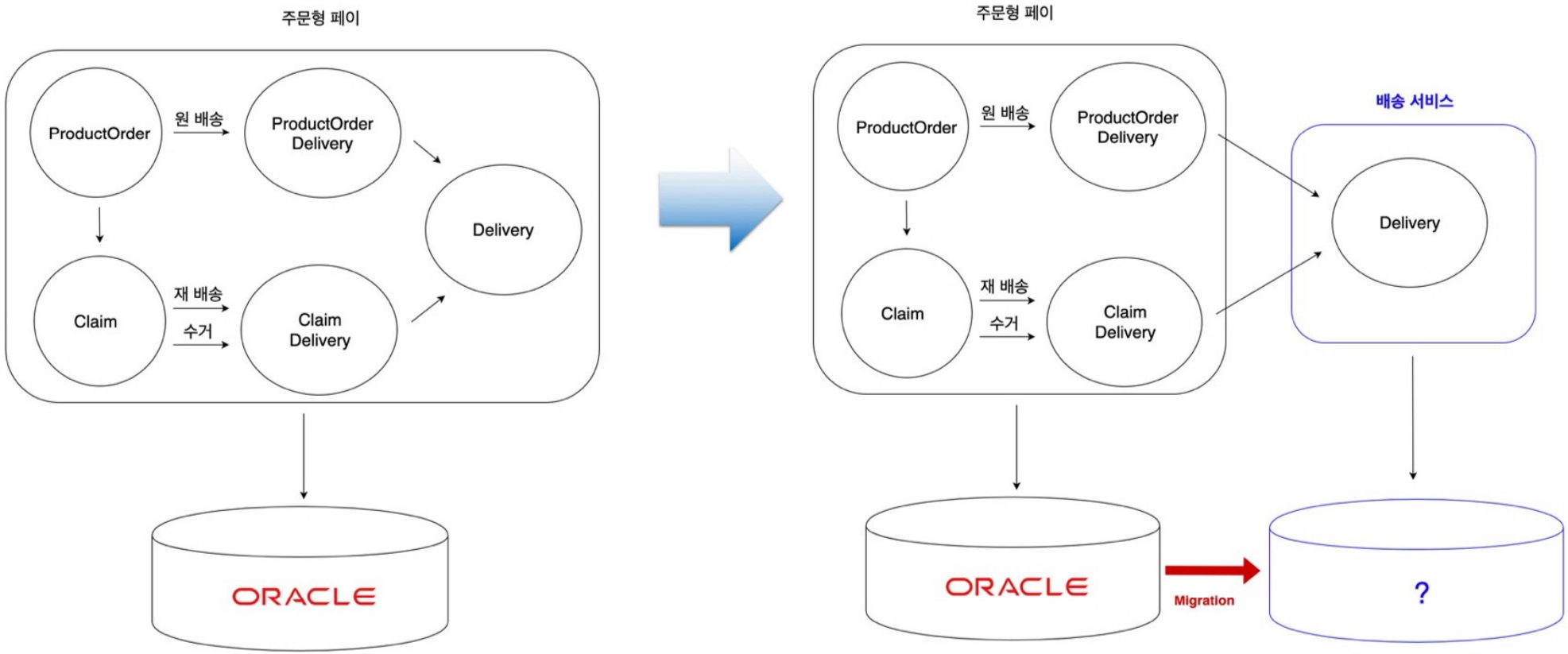
NAVER Container Cluster

- K8S 기반 사내 컨테이너 플랫폼
- Scale Out
- 리소스 효율화

2.2 데이터베이스

DEVIEW 2020

모듈 분리와 함께 신규 저장소 결정



2.2 데이터베이스

DEVIEW 2020

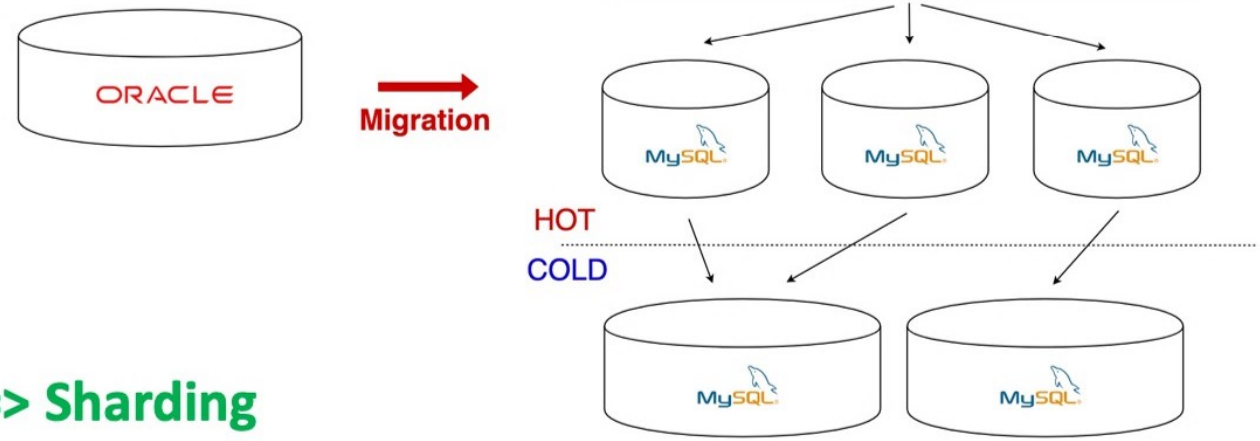
저장소 선택시 고려된 점

- Scale Out 가능한 분산 저장소
- 데이터 저장 및 활용 효율화
- 마이그레이션 및 실시간 동기화 용이

2.2 데이터베이스

nBase-T

- 분산 데이터베이스
- **NAVER** 사내 솔루션



- Scale Out 가능한 분산 저장소 **=> Sharding**
- 데이터 저장 및 활용 효율화 **=> Tiering**
- 마이그레이션 및 실시간 동기화 용이 **=> RDBMS**

2.3 nBase-ARC

DEVIEW 2020

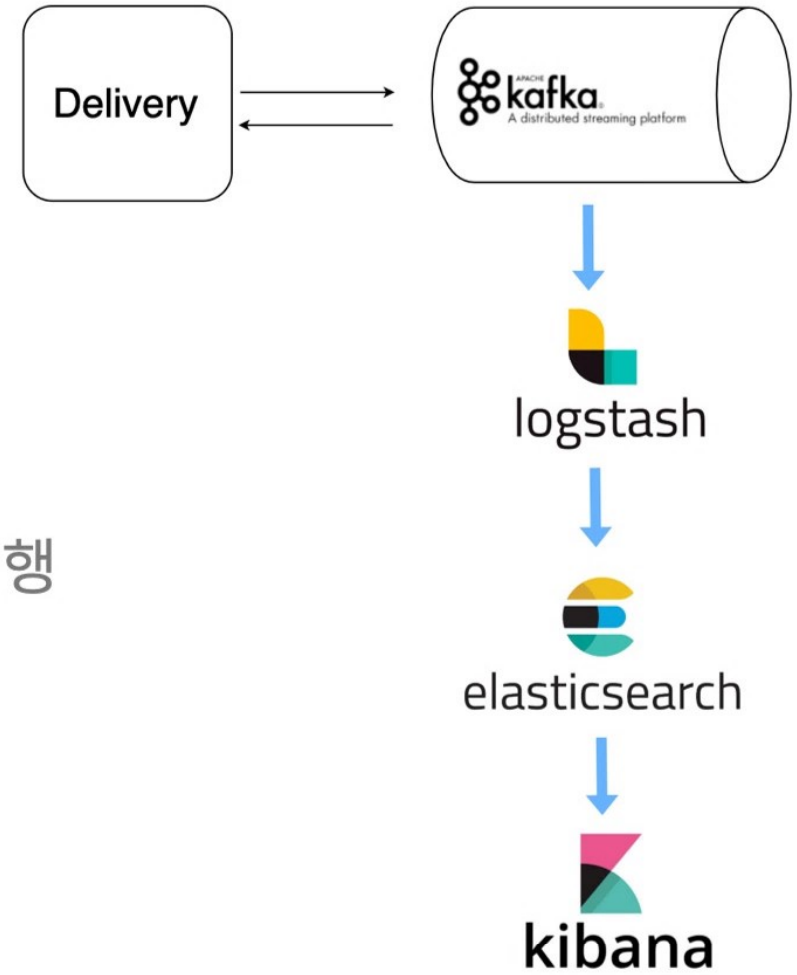
- **NAVER** Autonomous Redis Cluster DB
- Redis 기반 분산 저장 플랫폼

활용

- Global Index 저장 및 관리
- Global Cache
- 중복 처리 방어

2.4 Kafka + ELK

- Kafka Farm
- Elasticsearch Farm (ELK Stack)



활용

- 배송과 관련된 모든 변경 이벤트는 Kafka 로 발행
- 수집된 이벤트 분석 및 디버깅에 활용

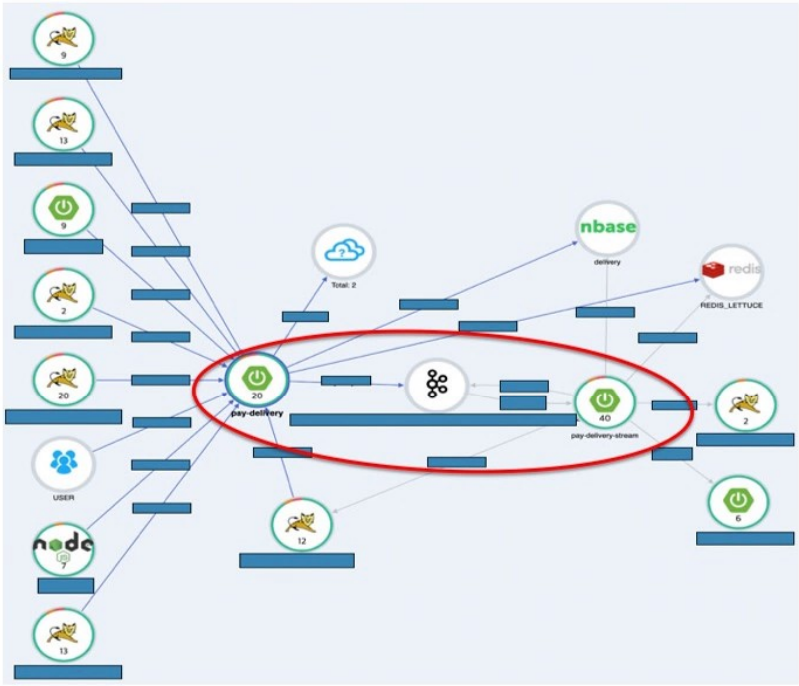
3. 아키텍처

3.1 Application 구성

DEVIEW 2020

인스턴스 분리 구성

- API Server : HTTP API 요청 처리
- Consumer Server : Kafka Consumer Message 처리



3.1 Application 구성

DEVIEW 2020

API Server 와 Consumer Server 를 분리해서 운영

- 트래픽 유입 경로가 다르다
- Consumer Server 는 BLUE / GREEN 배포가 어렵다
- GRACEFUL SHUTDOWN 방법이 다르다
- 리소스를 활용하는 방식이 다르다
- 장애 격리가 필요하다
- 배포 이유와 시점이 다르다

- API Server 는 요청 처리에 즉시성을 요구하고 Consumer 는 지연을 허용한다

3.2 Message Spec

DEVIEW 2020

CloudEvents (CNCF Incubating projects)

- 일관된 포맷으로 Message 를 관리할 필요가 있다
- https://cloudevents.io/

Example

The following example shows a CloudEvent serialized as JSON:

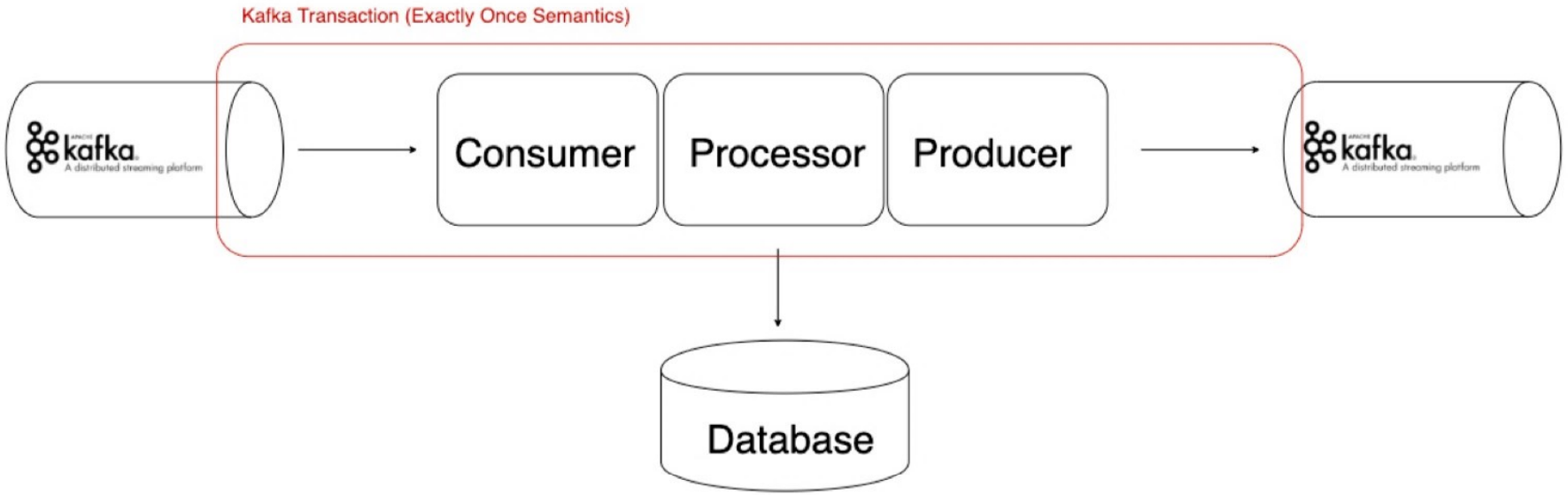
```
{
    "specversion" : "1.0",
    "type" : "com.github.pull.create",
    "source" : "https://github.com/cloudevents/spec/pull",
    "subject" : "123",
    "id" : "A234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "text/xml",
    "data" : "<much wow=\"xml\"/>"
}
```

3.3 Kafka Exactly Once Semantics

DEVIEW 2020

Consumer – Processor – Producer 처리를 한번만 유효하게 하는 방법

- 실패에 따라 여러번 재시도할 수 있어야 한다
- Processor 는 멱등성(Idempotent)을 가져야 한다

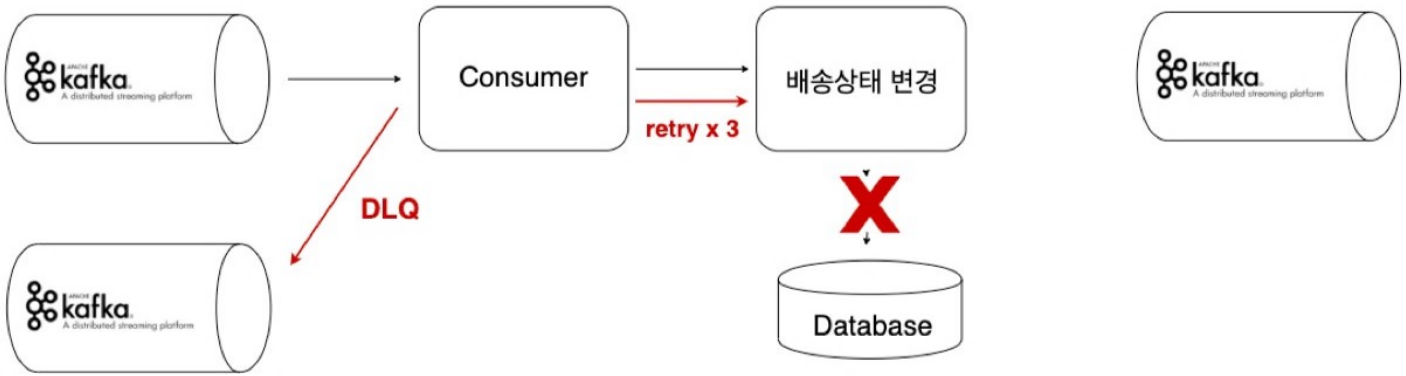


3.4 Dead letter queue

DEVIEW 2020

처리 실패한 Message 는 DLQ 로 발행한다

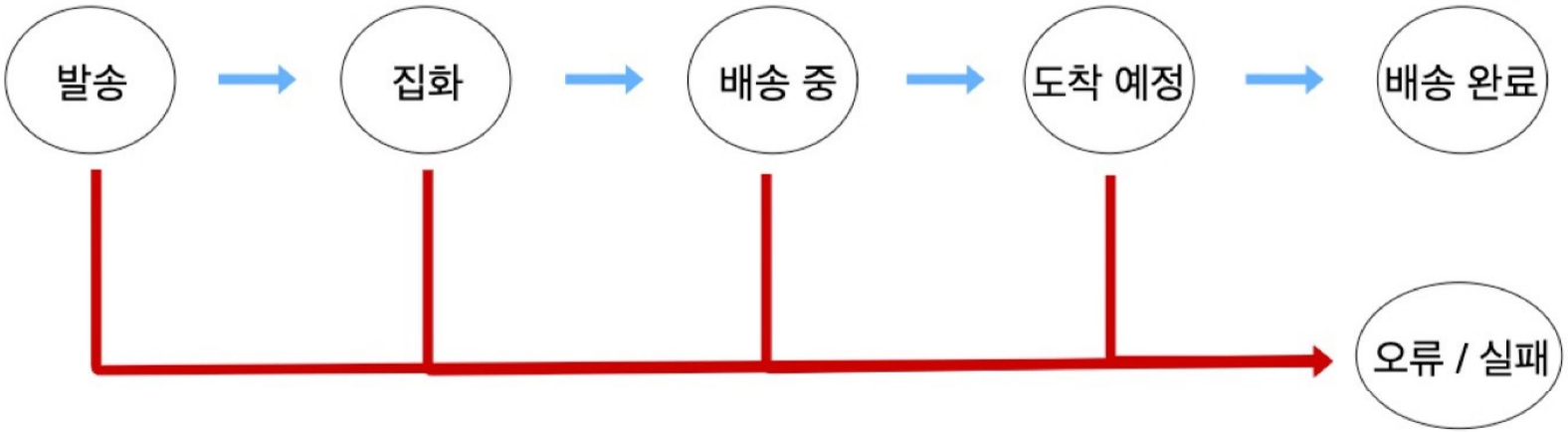
- Consumer 는 3번 재시도 한다 (재시도 간격 1초)
- DLQ 로 보내서 Message 가 막히지 않도록 한다
- DLQ 에 발행된 Message 는 원인 확인 후 재처리 하거나 Message 를 버린다.
- 재처리를 위해서 Processor 는 멱등해야 한다
- Message 순서 변경에 대한 대응이 필요하다



3.5 Idempotent

프로세싱 간 Message 누락 및 문제 발생시 재시도를 통해 최종 상태를 맞출 수 있다

- 배송 상태 변경은 반복적으로 재시도 될 수 있다
- 배송 상태가 역행되면 안된다
- 배송 상태가 최종 상태(배송완료 / 오류 / 실패)에 도달하면, 추가 요청은 처리하지 않는다

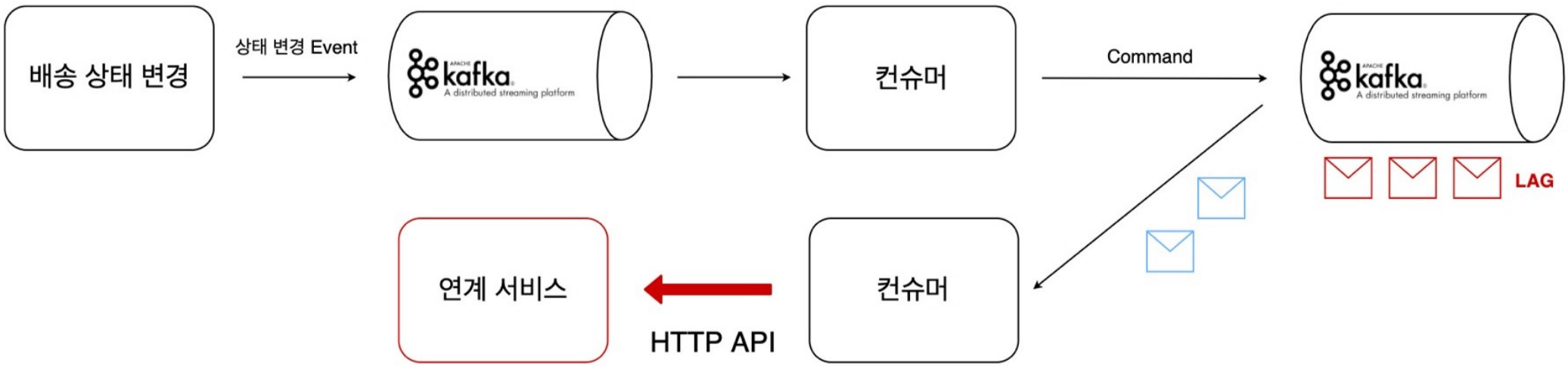


3.6 Message Flow 모니터링 및 트래픽 제어

DEVIEW 2020

Message 처리 구간 단위로 처리량을 조절할 수 있다

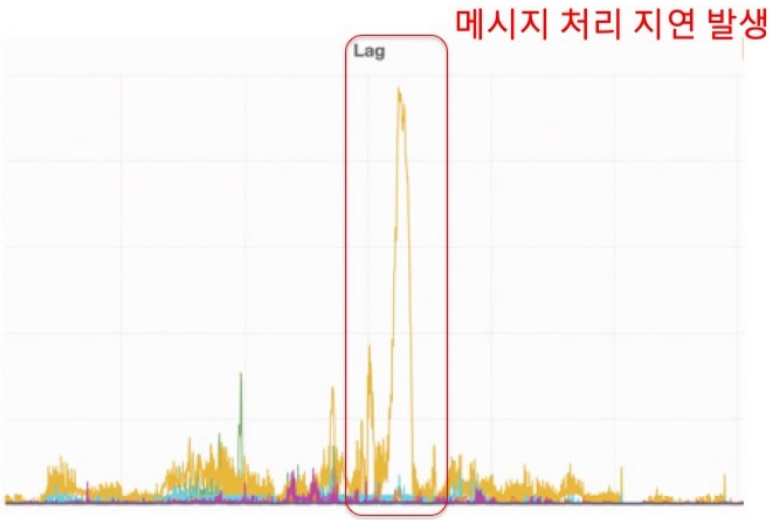
- 상태 변경 이벤트 발생량이 Message 처리량 보다 크면 지연(LAG) 이 발생한다



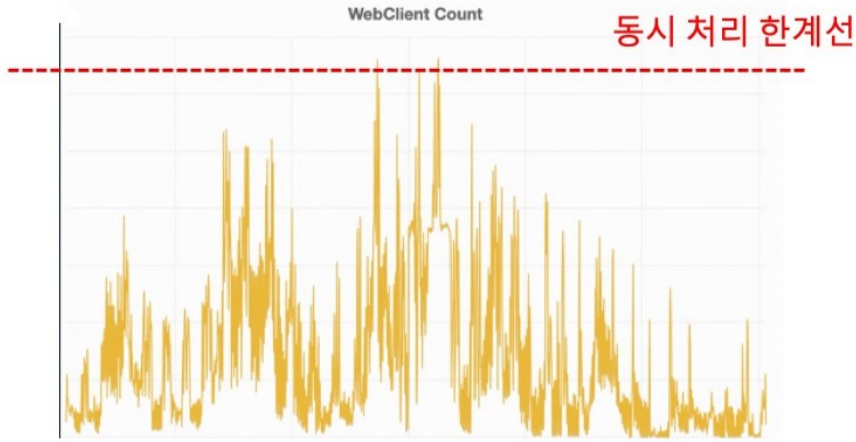
3.6 Message Flow 모니터링 및 트래픽 제어

순간적인 트래픽 폭증이 연계 서비스를 죽이지 않도록 조절한다

- Partition 을 늘려서 동시 처리량을 늘릴 것인가?
- 연계 서비스 가 늘어난 트래픽을 처리 가능한 수준인가?
- 어느 정도의 지연이 허용 가능하며, 어느 구간의 지연이 리스크가 적은가?



Consumer Group Lag



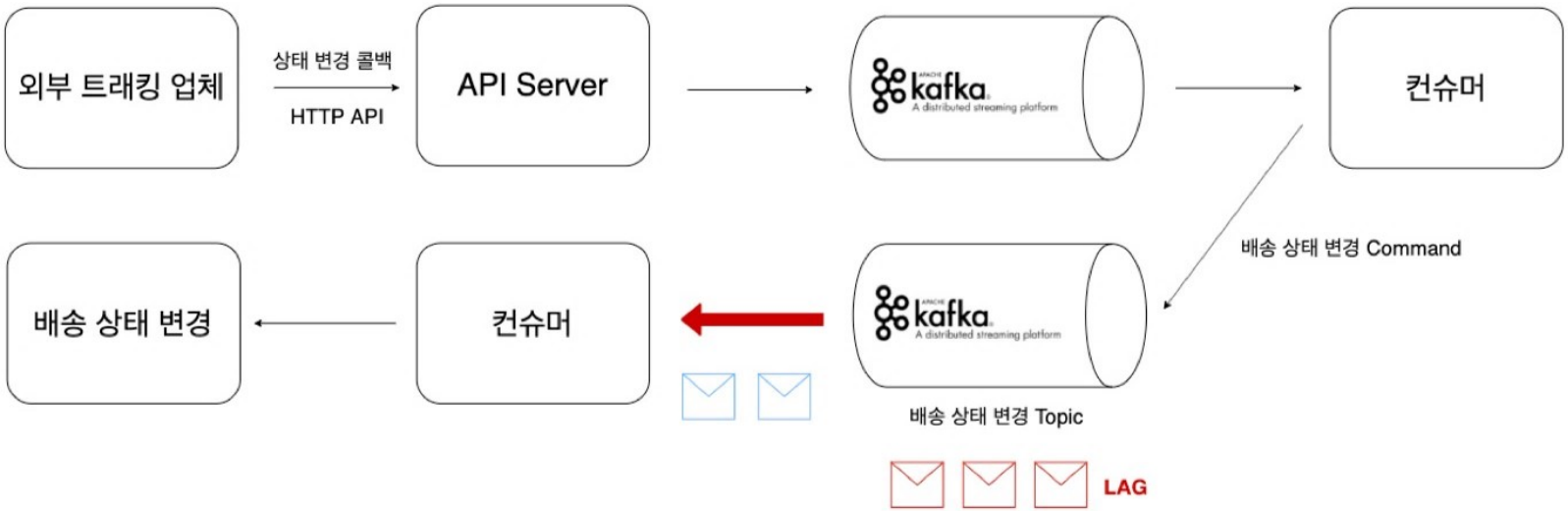
외부 연계 요청 Count

3.6 Message Flow 모니터링 및 트래픽 제어

DEVIEW 2020

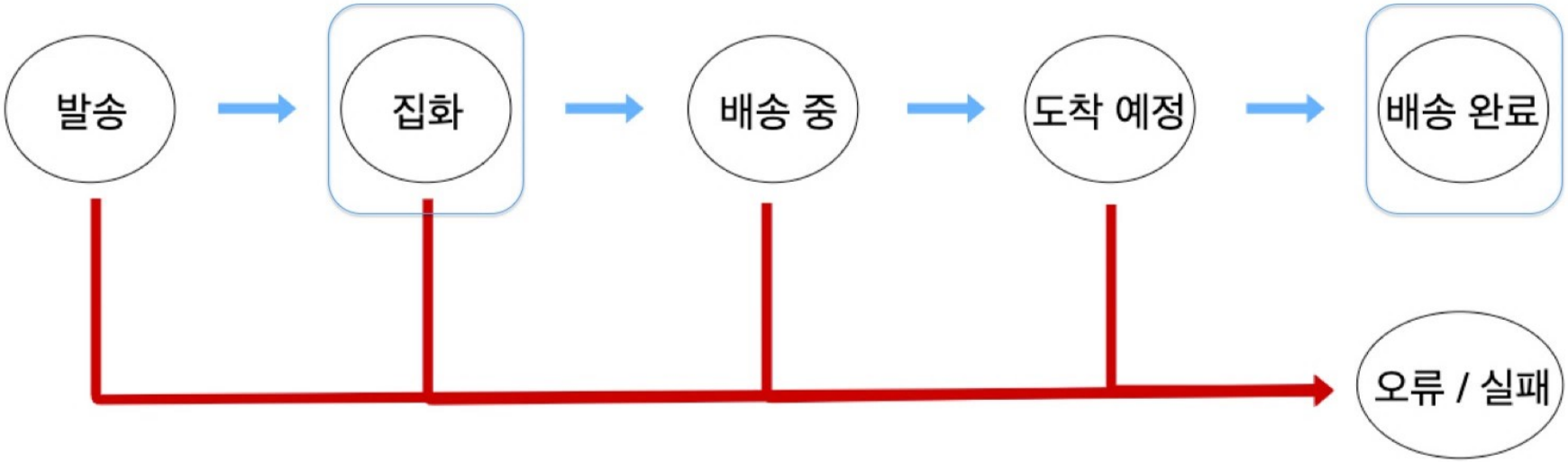
효과적으로 Message 지연 최소화 하기

- 도메인 규칙을 활용하여 Message 처리를 효율적으로 할 수 있다
- 가능하다면 도메인 규칙을 변경하여 문제를 해결하는 것도 방법이다



3.6 Message Flow 모니터링 및 트래픽 제어

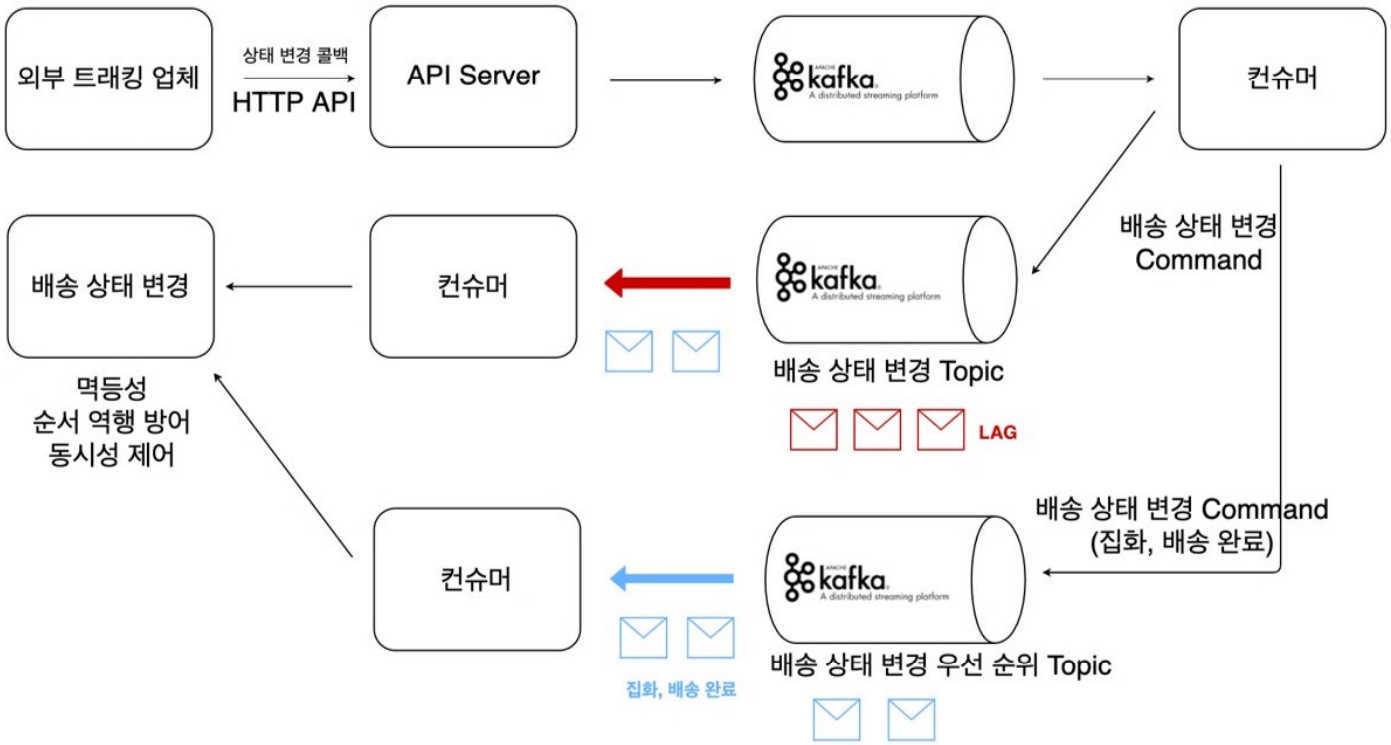
비즈니스에서 더 중요한 **"집화"** , **"배송 완료"** 상태 변경을 우선 처리한다



3.6 Message Flow 모니터링 및 트래픽 제어

지연이 발생할 때 우선 순위 Queue 로 “집화”, “배송 완료” 상태를 분기 발행한다

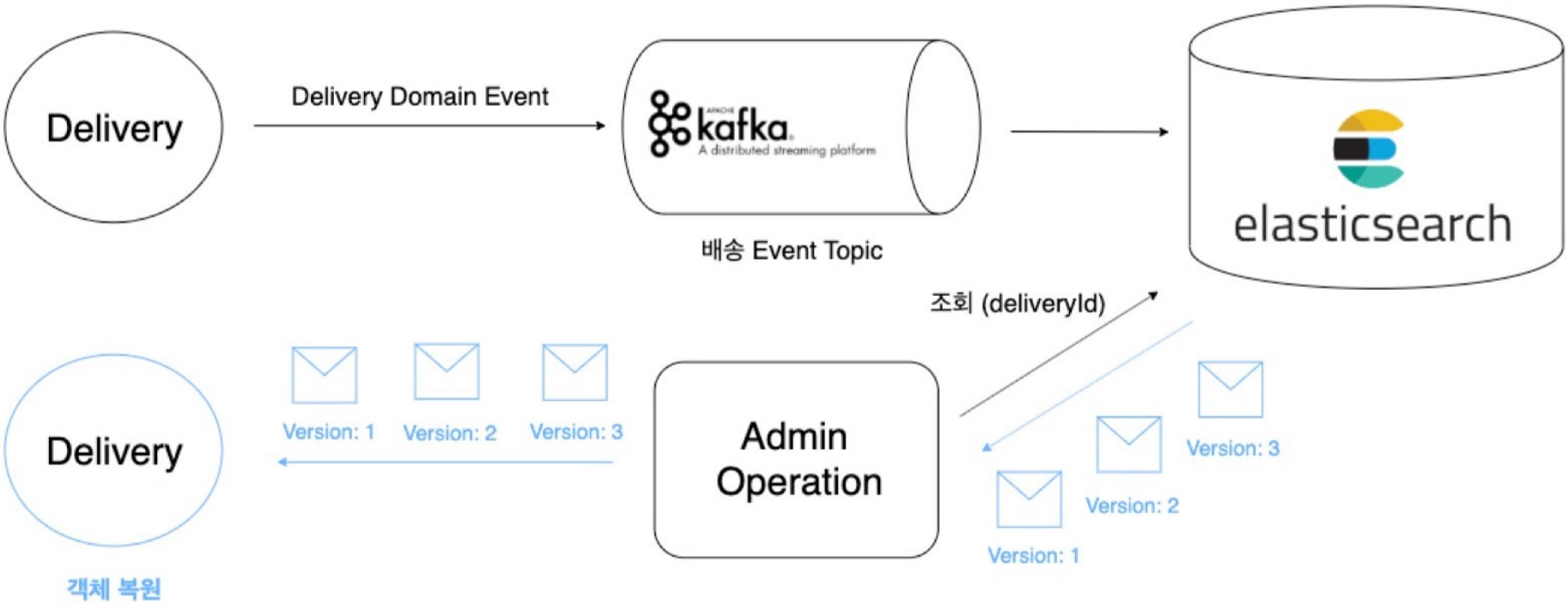
- “배송 상태 변경” 프로세스는 **동시성 제어** 및 **상태 순서 역행 방어** 가 되어야 한다.



3.7 Message 분석 및 디버깅

데이터의 변경에 대한 Domain Event 로 데이터 분석 및 디버깅 가능

- EventSourcing 기법을 사용한 도메인 객체 복원



DEVIEW 2020

4. 전환 과정

4.1 마이그레이션

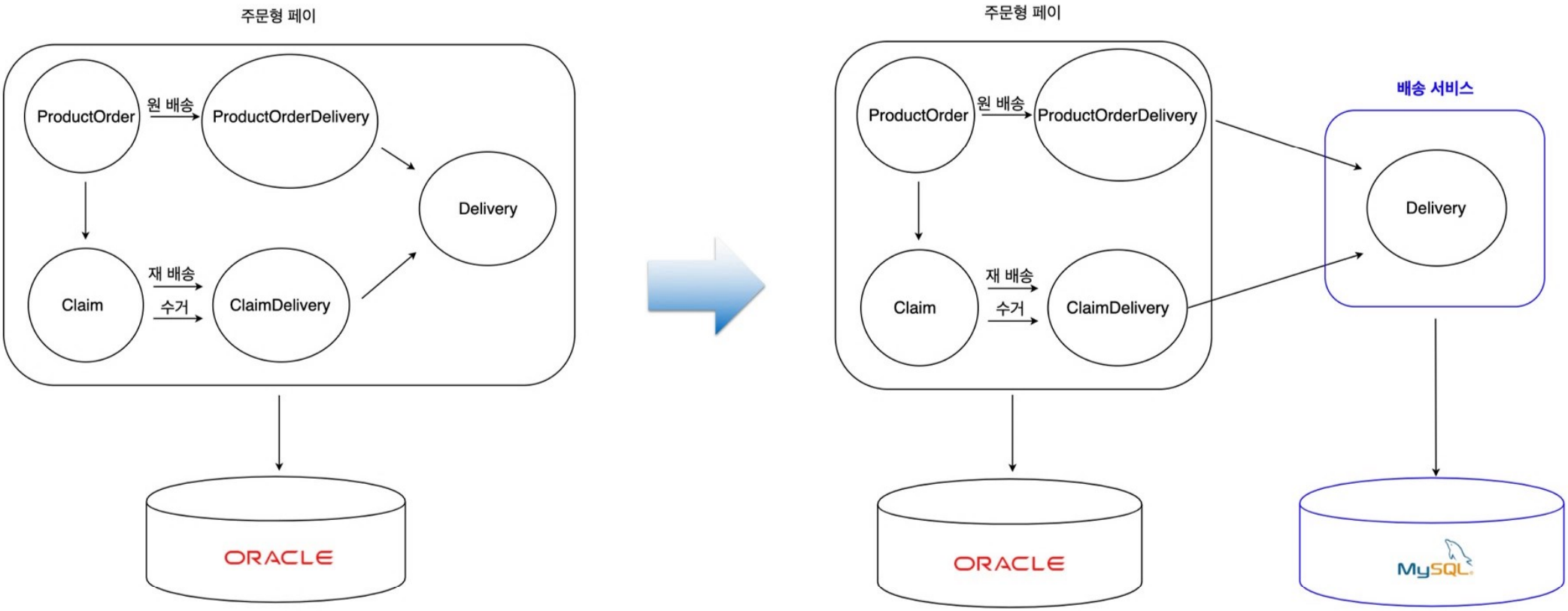
DEVIEW 2020

분산 필요성

- Monolithic Service 의 과대화
- DB(Oracle) 부하 증가
- Oracle Transaction 에 의존
  - Oracle Scale-Up -> Service 중단
- Service가 커지며 연관관계가 증가
  - 신속한 의사결정 방해

4.1 마이그레이션

배송 데이터의 모든 CRUD 작업은 nBase-T(MySQL) 에서 실행



4.1 마이그레이션

개발 / 운영의 DB 구조 및 Table 속성이 동일한가?

- Default Value, Not Null, Column Type, Column Size, …

배송 도메인과 연관되는 Table 이 맞는가?

- 배송비 관리 Table 은 배송 도메인과는 무관

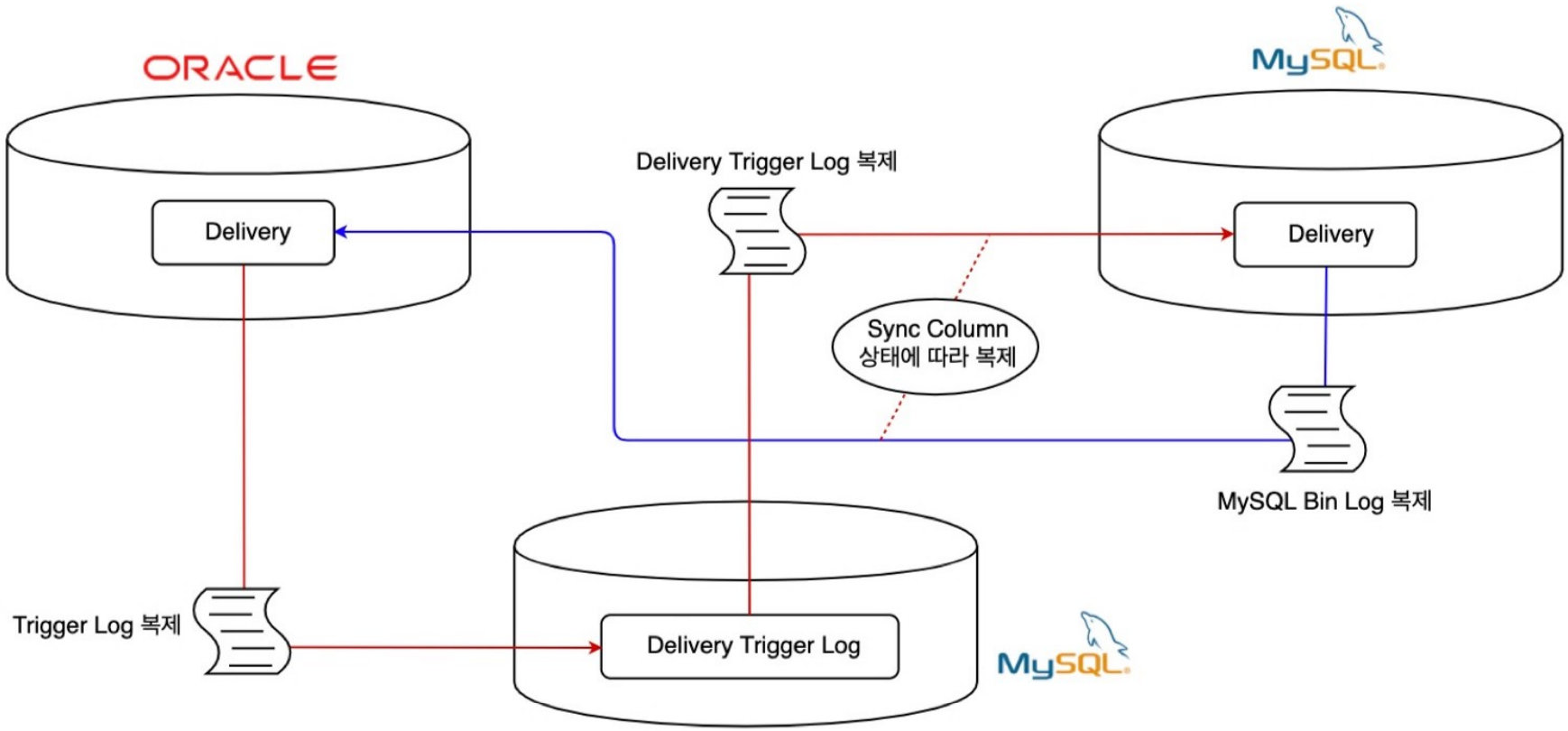
제거 가능한 Table, Column 인가?

- 사용되지 않거나 대체 가능한 Table, Column 전환 시 제거
- 히스토리성 데이터들은 ES 를 통해 별도 관리 가능

4.1 마이그레이션

DEVIEW 2020

이기종 DB 양방향 실시간 복제



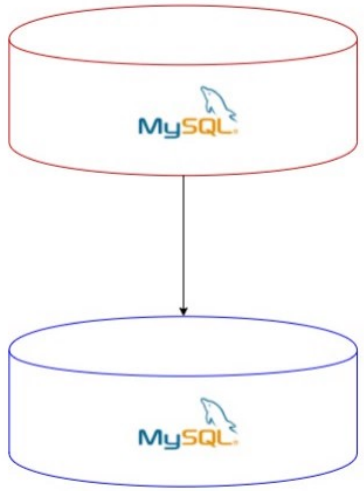
4.1 마이그레이션

DEVIEW 2020

Tiering

- 배송 완료 마지막 상태가 존재
- 배송 완료 된 데이터는 자주 조회되지 않음
- 특정 기간(1년)을 기준으로 데이터를 분산
- HOT / COD 는 각 다른 인프라 장비로 구축

Hot: 1년 이내의 배송 데이터

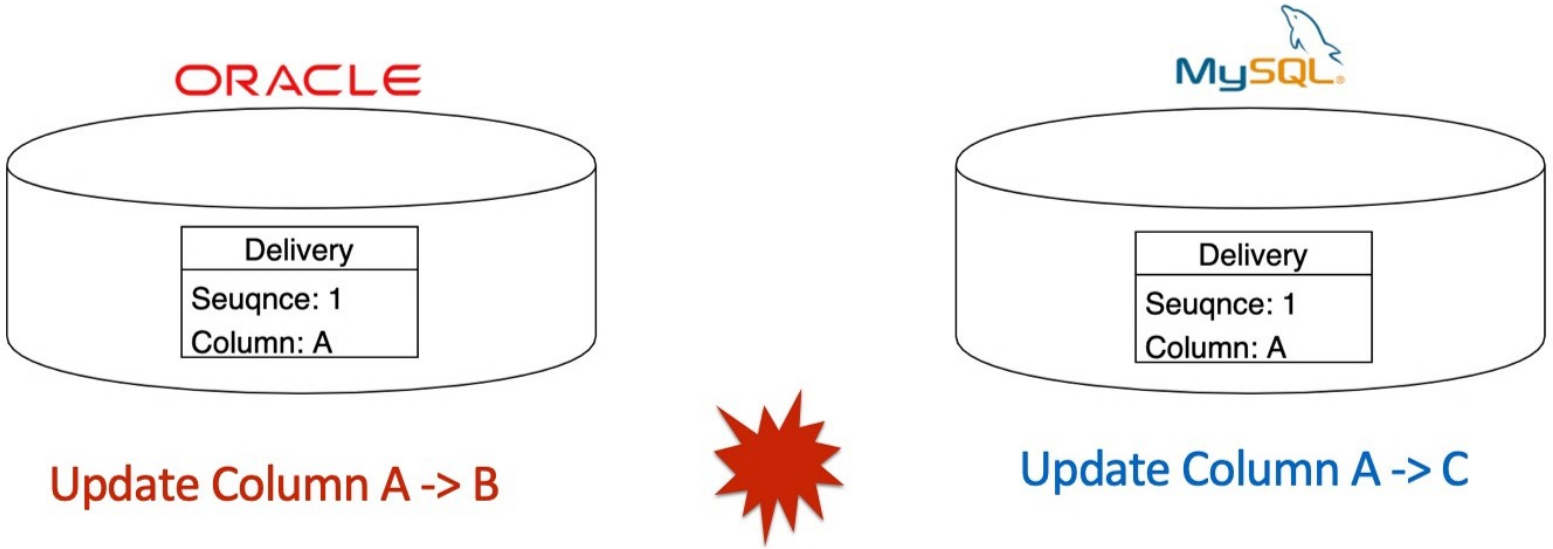


Cold: 1년 이상의 배송 데이터

4.2 전환 제약사항

DEVIEW 2020

Dual Write



동시에 동일한 Sequence 의 데이터를 Update 한다면,
Column 값은 B 와 C 중 어느 것이 되어야 하는가?

4.3 기본 전환 전략

DEVIEW 2020

기본 전략

- 최대한 기능을 분리해서
- Write 보다는 Read 기능부터
- 롤백이 쉬운 기능부터
- 의존성이 낮은 기능

4.4 전환

DEVIEW 2020

Delivery Lazy Loading 제거

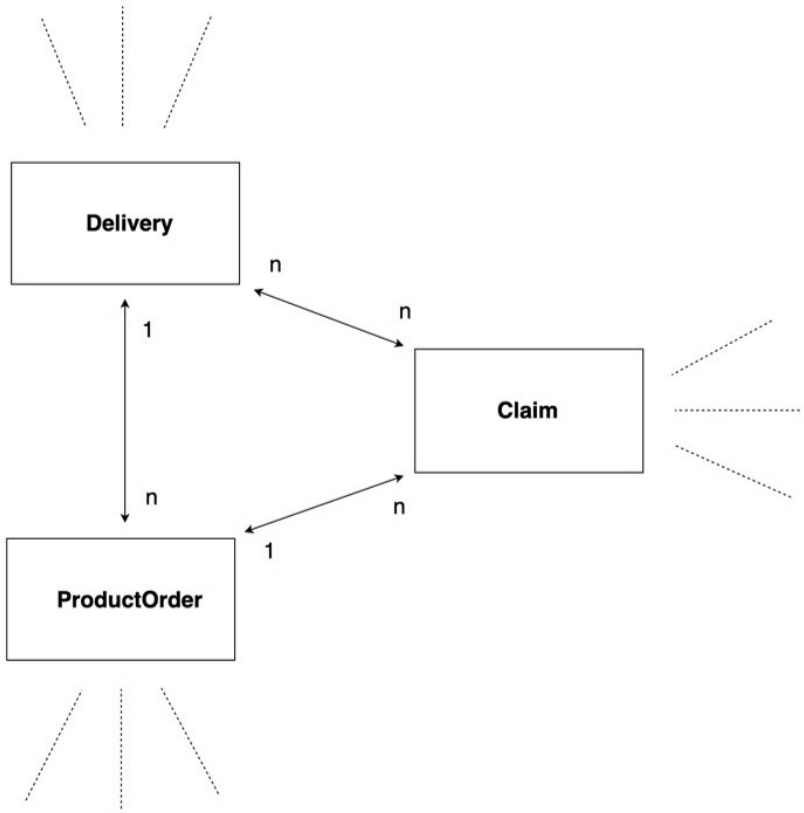
Delivery Table Join 제거

Transaction Scope 조정

- N -> 1 개 단위로 Transaction 처리

Unreached 코드, Deprecated 코드 제거

- 분산되어 있던 정책의 단일화



4.4 전환

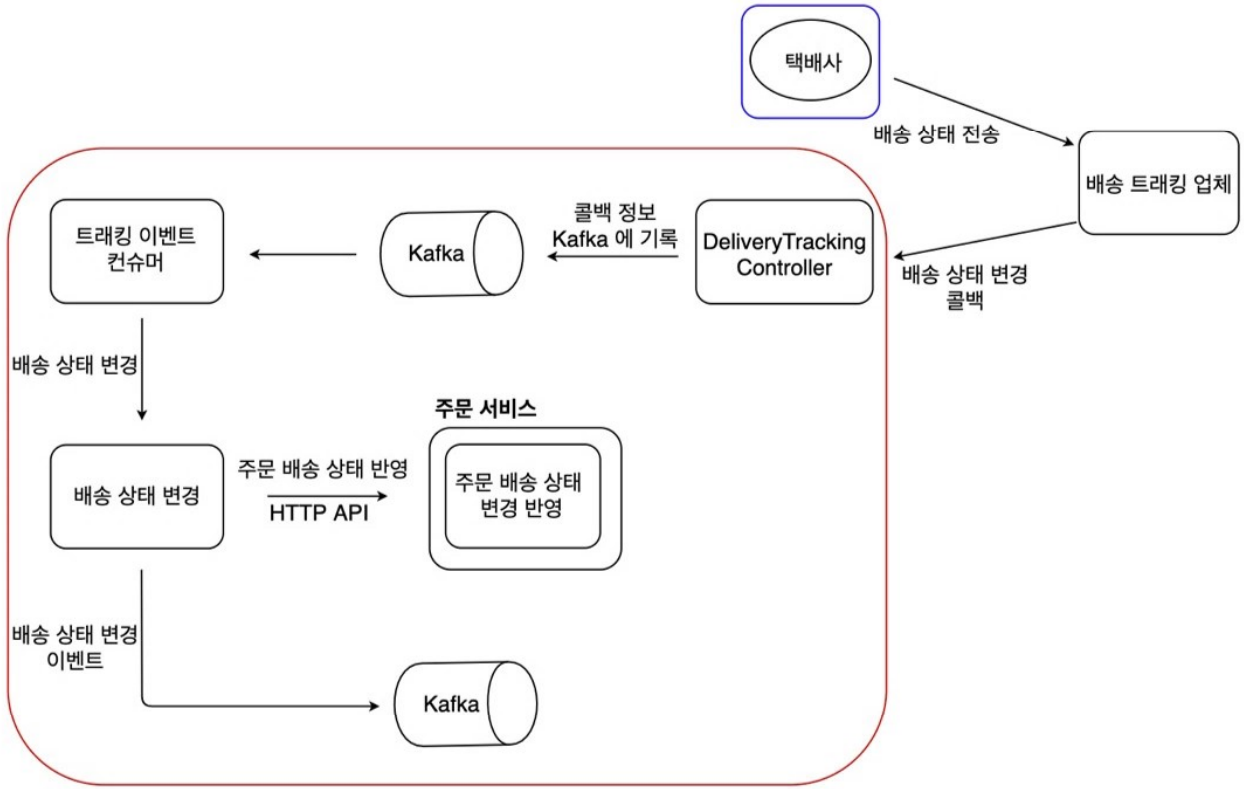
DEVIEW 2020

택배사

- 의존성이 낮으며 비교적 단순한 PK 위주의 조회 기능
- 새로운 기술 학습
- API 모듈 전환 전략 검증

배송 트래킹

- 비실시간성 데이터 (즉시성 X)
- Consumer 모듈 전환 전략 검증

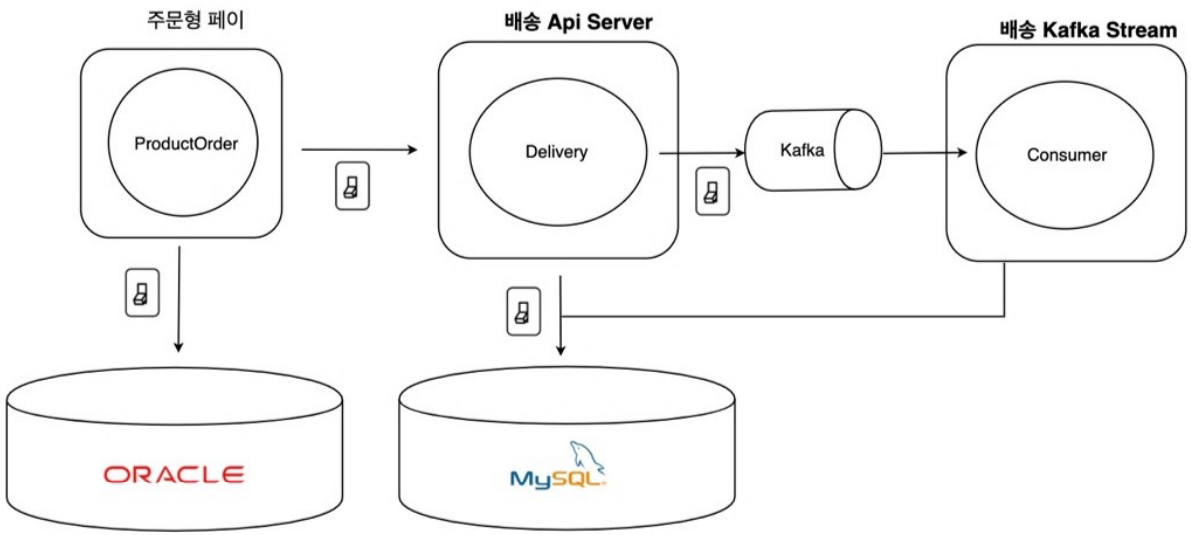


4.4 전환

DEVIEW 2020

Switch & Dry Run

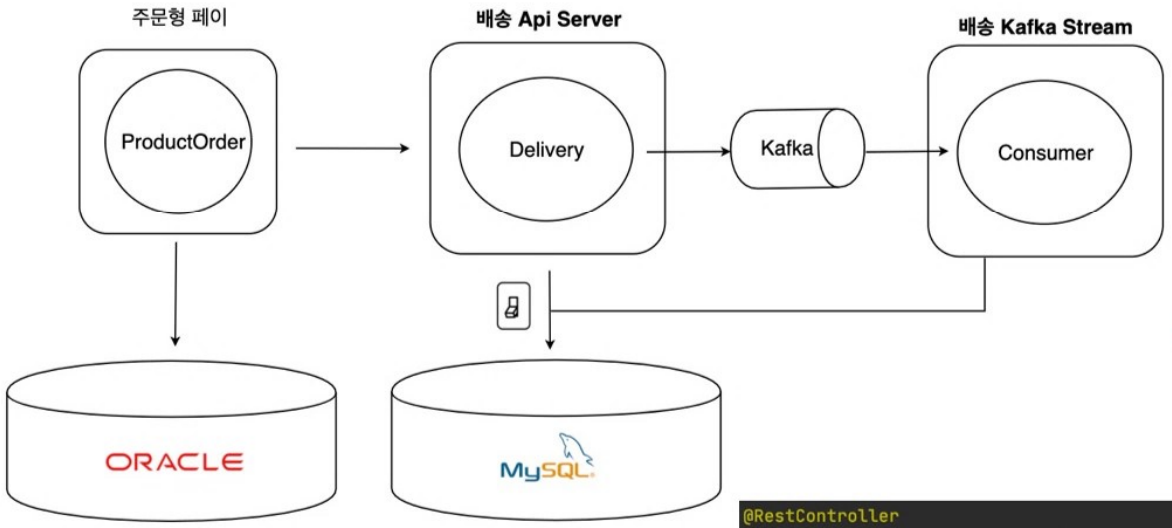
- 배포없이 신속한 롤백
- 필요한 기능별로 전환 가능
- 데이터 변경을 제외한 로직 실행 / 검증



4.4 전환

DEVIEW 2020

Switch & Dry Run

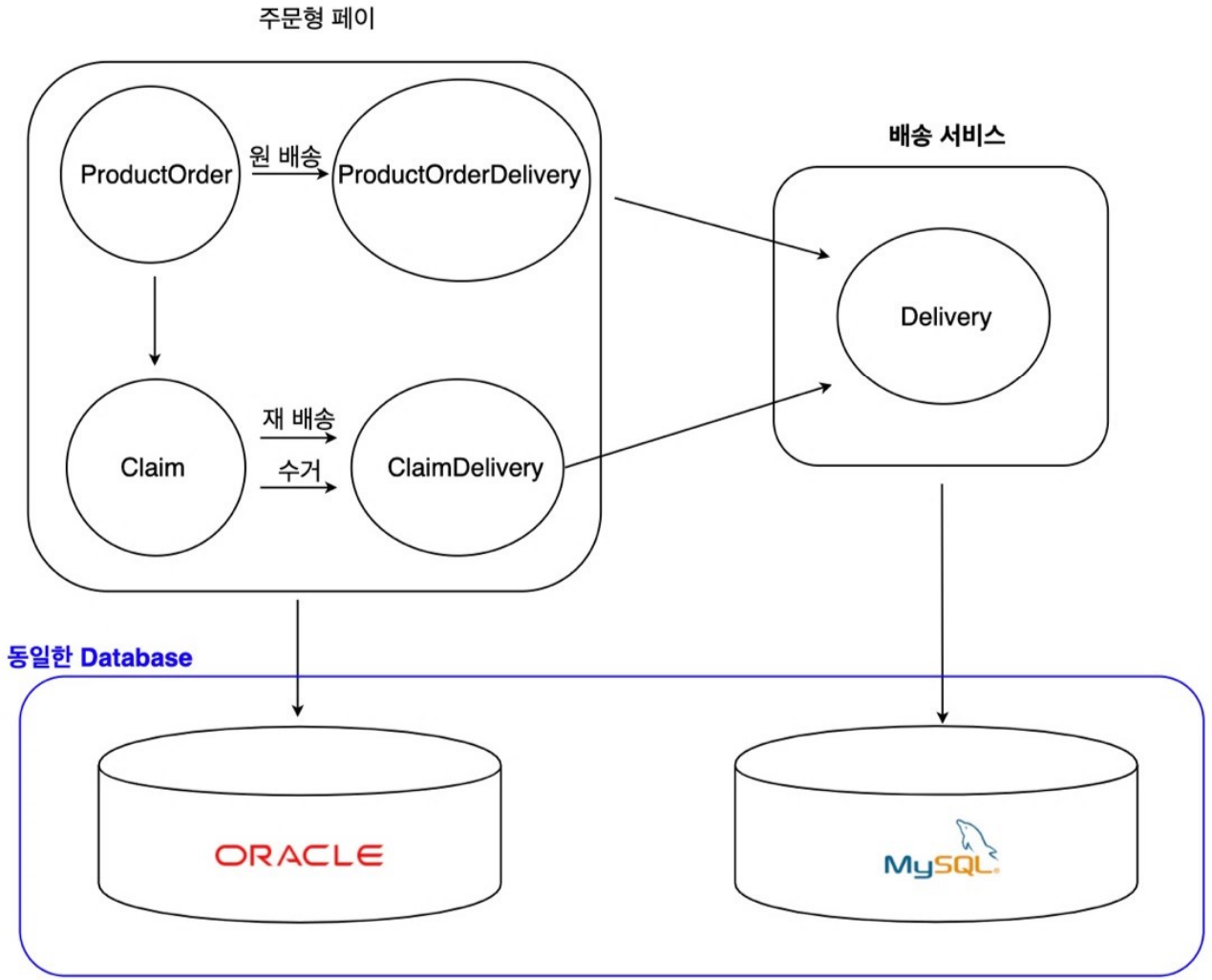



```
// 원 발송처리 기능은 이전에 완료
if (this.switchService.isOnDispatchRegularDelivery()) {
    try {
        Delivery remoteDelivery = this.deliveryService.dispatchRegularDelivery(
            form, activeDelivery.getSenderAddress(), activeDelivery.getReceiverAddress());
        this.deliveryService.compareBothPartial(activeDelivery, remoteDelivery);
    } catch (Exception ex) {
        NEXT_DELIVERY_LOGGER.error(OrderNelo2LogFormatType.배송_NEXT_DB.getLogText(
                ...params: "[DELIVERY ADAPTER] dispatchPService dispatch regularDelivery remote occurred error.",
            activeDelivery.getId()), ex);
    }
}
```

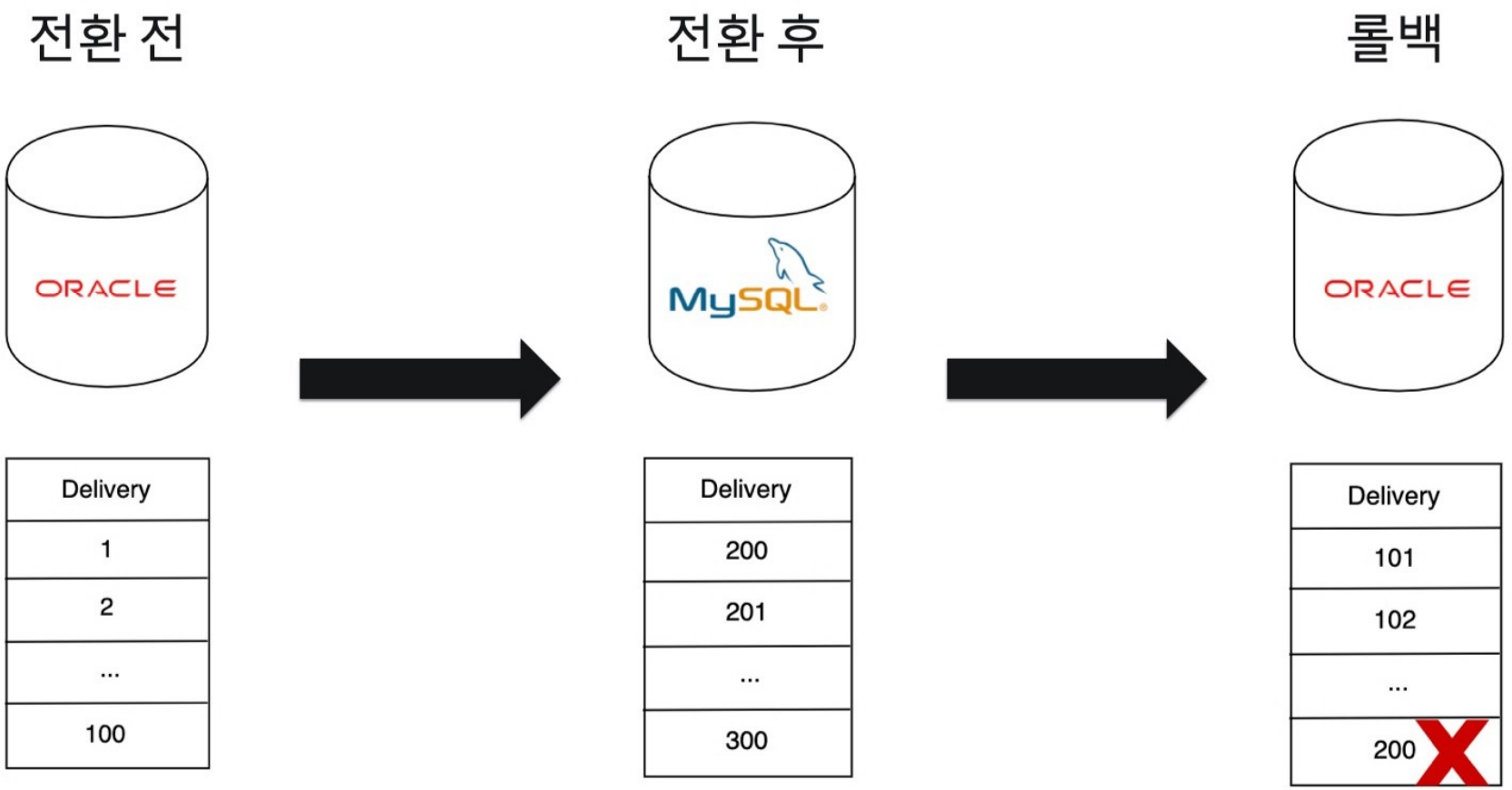
```
@RestController
@RequestMapping( ...value: "/switch")
class SwitchApiController(private val switch: Switch) {

    @GetMapping( ...value: "/deliveryWrite")
    fun getDeliveryWrite(): SwitchValue =
        SwitchValue(this.switch.isDeliveryWrite())

    @PutMapping( ...value: "/deliveryWrite")
    fun putDeliveryWrite(@RequestBody value: SwitchValue): SwitchValue {
        this.switch.setDeliveryWrite(value.value)
        return SwitchValue(this.switch.isDeliveryWrite())
    }
```

```
@Component
class DeliveryEventPublishRepository(
    private val repository: DeliveryRepository,
    private val eventPublishTemplate: EventPublishRepositoryTemplate,
    private val switch: Switch
) {
    private val logger = logger()

    fun insert(delivery: Delivery): Delivery =
        if (this.switch.isDeliveryWrite()) {
            this.eventPublishTemplate.save(delivery) { it: Delivery!
                this.repository.insert(it)
            }
        } else {
            this.eventPublishTemplate.publishEvents(delivery)
            this.validateAndLog(delivery)
            delivery
        }
```

4.5 DB Sequence



4.5 DB Sequence



4.5 DB Sequence

전환 전

Delivery
1
2
...
100

Delivery Api 요청시

Oracle 에서 Delivery Table

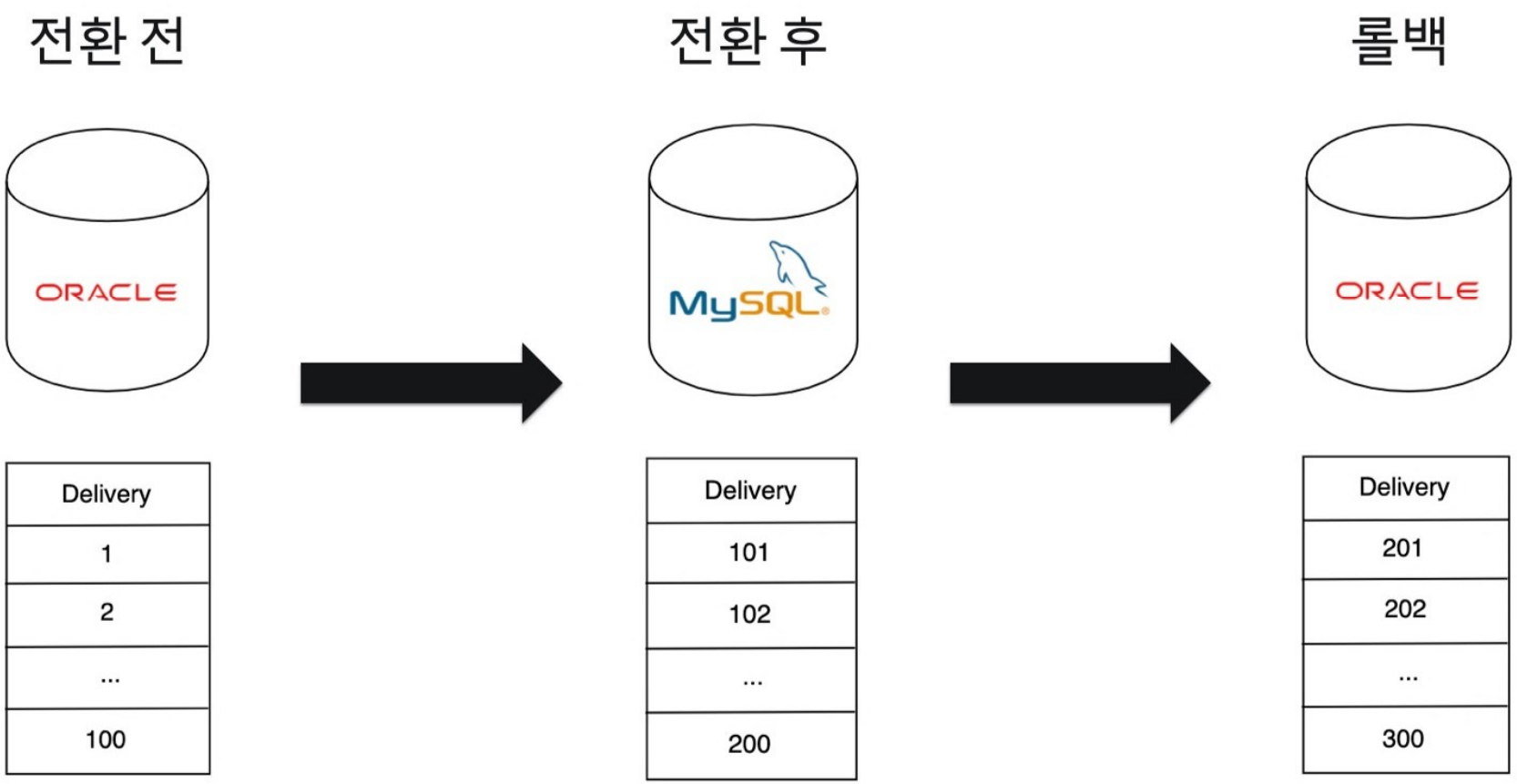
채번 후 함께 전달

101
102
...
200

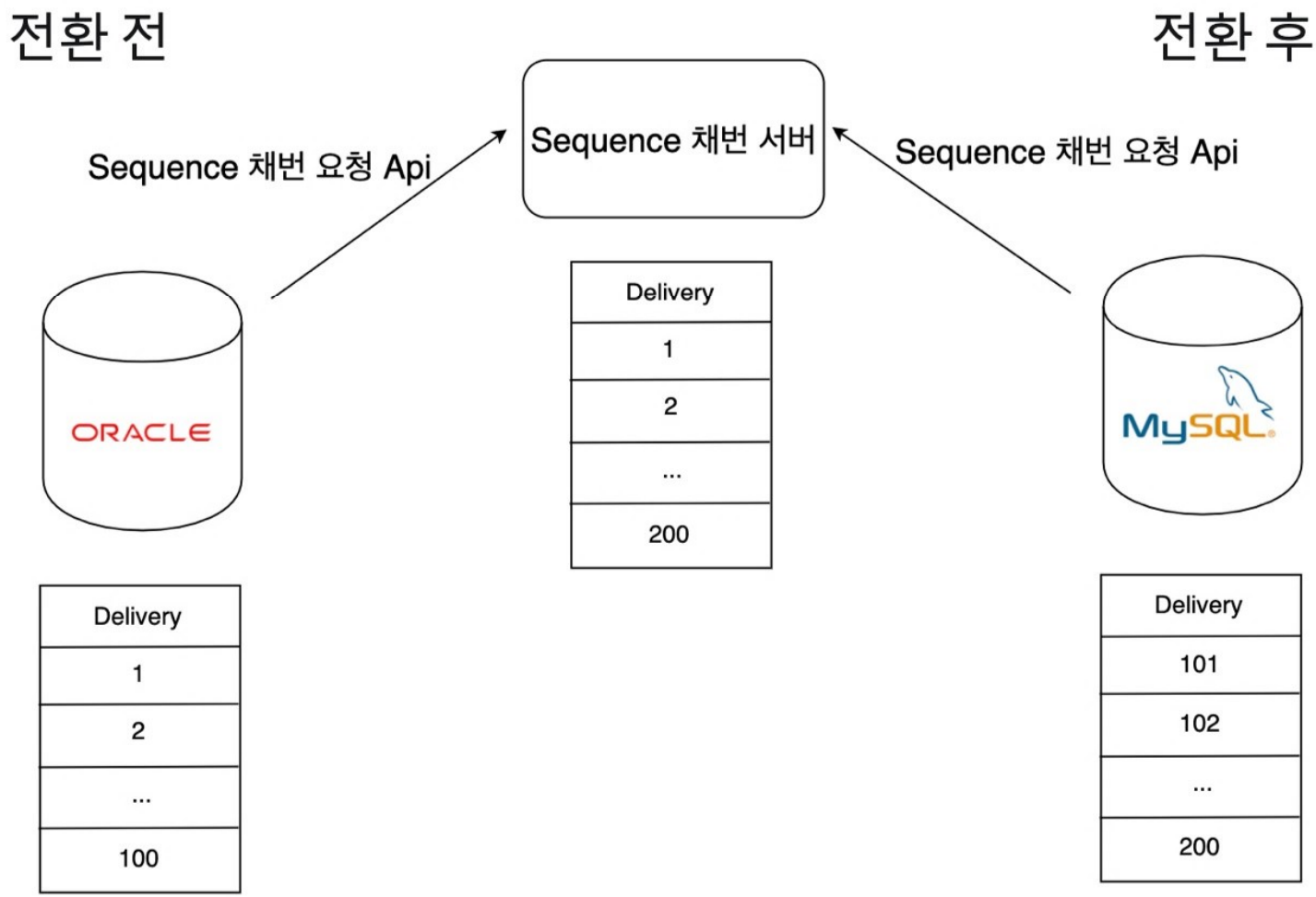
전환 후

Delivery
101
102
...
200

4.5 DB Sequence

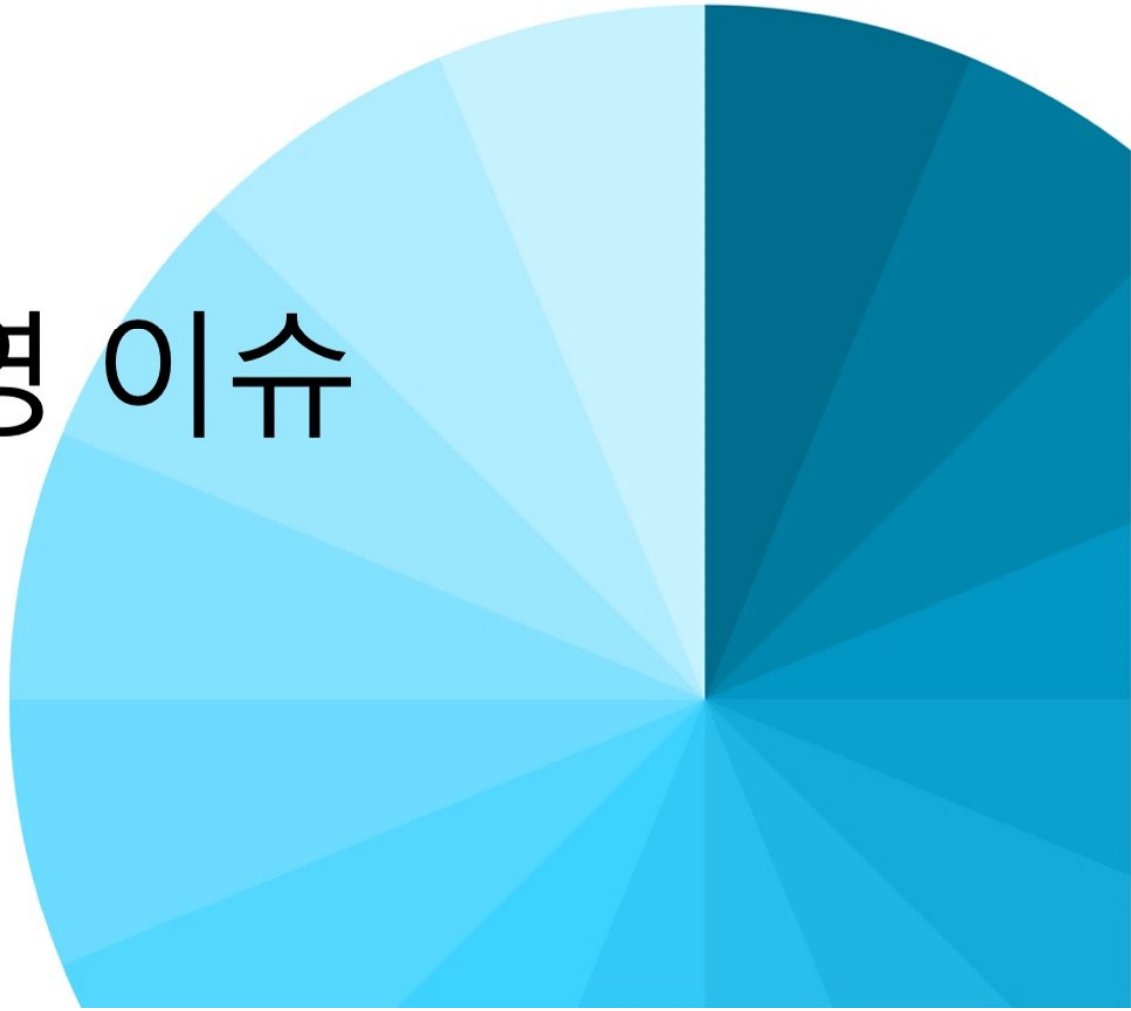


4.5 DB Sequence



DEVIEW 2020

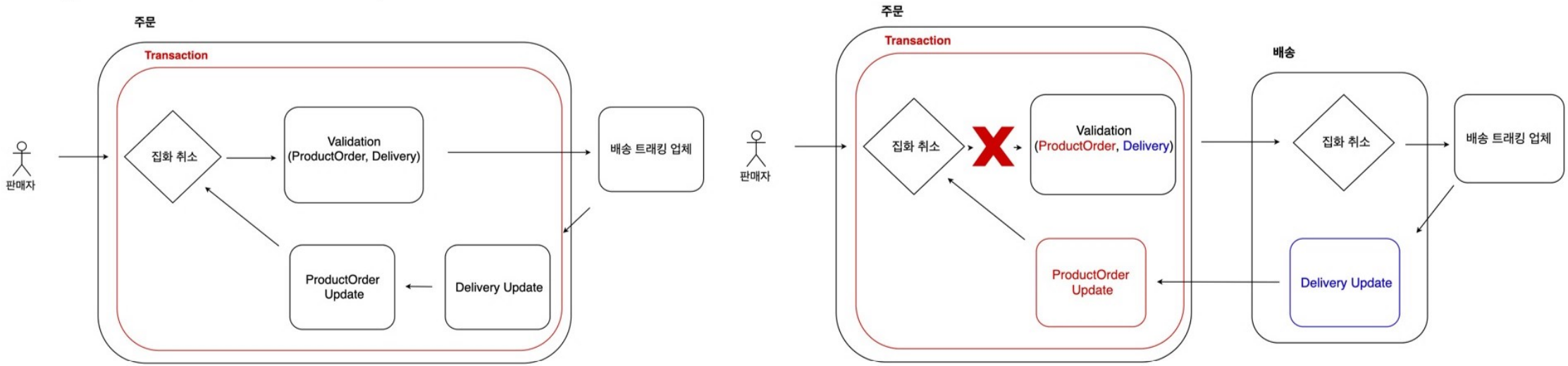
5. 운영 이슈



5.1 Transaction 분리 이슈

DEVIEW 2020

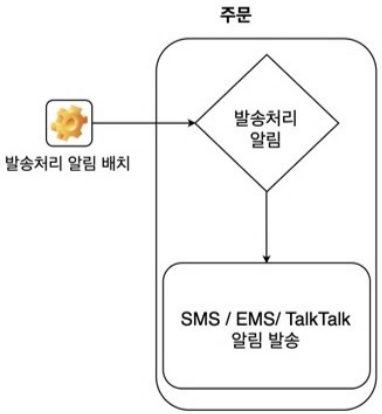
주문에서 배송 API 요청이 Timeout 으로 실패, 트래킹 업체 통신은 성공
Delivery 상태만 Update
판매자 집화 취소 재시도시 Delivery 가 Update 되어 Validation 실패
집화 취소 Validation 에서 배송 상태가 “집화 취소” 라도 통과할 수 있게 변경
(멱등성 보장 필요)

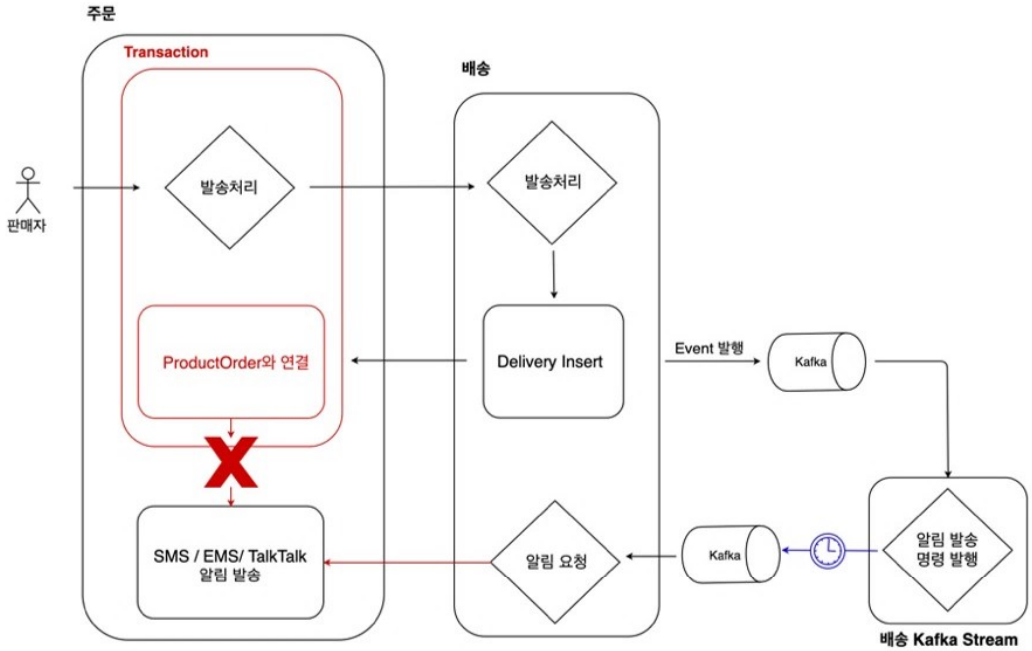


5.2 주문 Transaction 종료와 배송 Consume 시간차 이슈

DEVIEW 2020

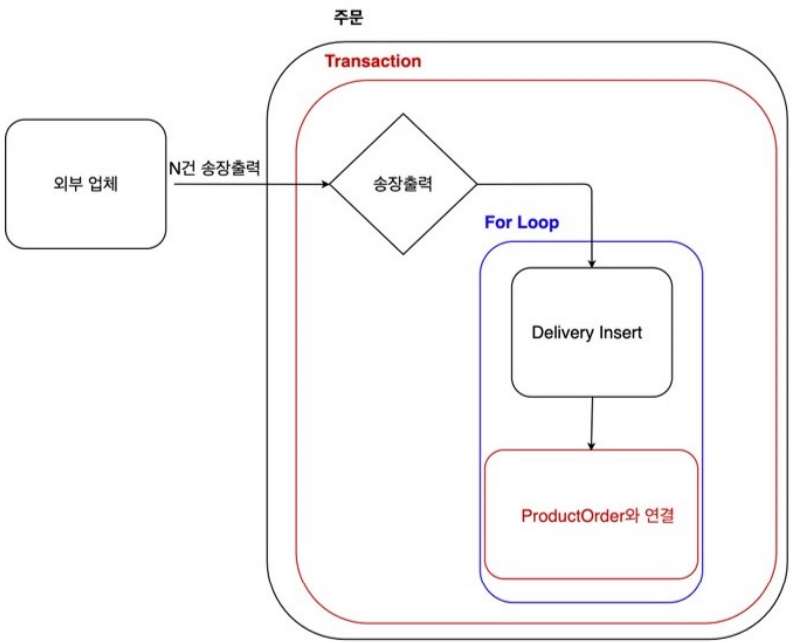
Delivery Insert 시 ProductOrder 연결, Event 발행 함께 수행
Event 발행이 먼저 수행되고, 이후 로직에 연결된 ProductOrder 가 필요
주문 Transaction 이 아직 실행 중이어서 ProductOrder 조회 실패
일정 시간 이후 Event 를 발행하는 Timer Handler 추가

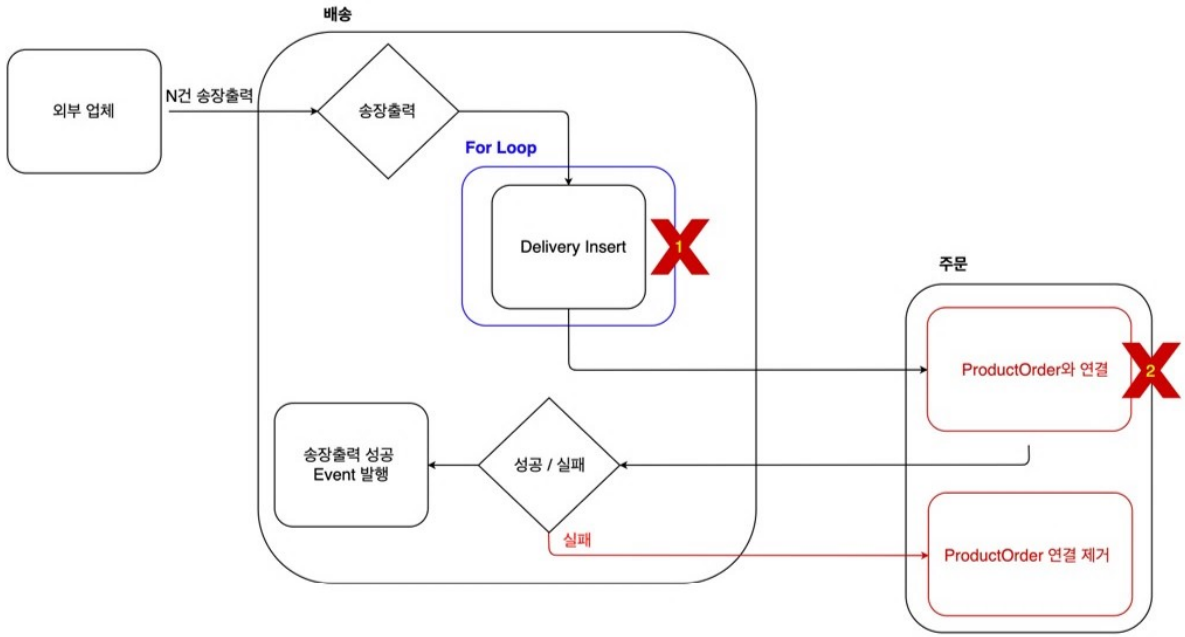




5.3 N -> 1건 단위 Transaction 변경 이슈

N건의 송장출력에 대한 최종 응답은 N건의 전체 성공 / 실패 여부
N건의 Delivery Insert 도중 실패시 고아 데이터 생성
ProductOrder 연결 API 요청 실패시 일부만 ProductOrder 와 연결 될 가능성
API 요청 실패시 ProductOrder 와 연결 제거 요청





5.4 Kafka Lag 해소

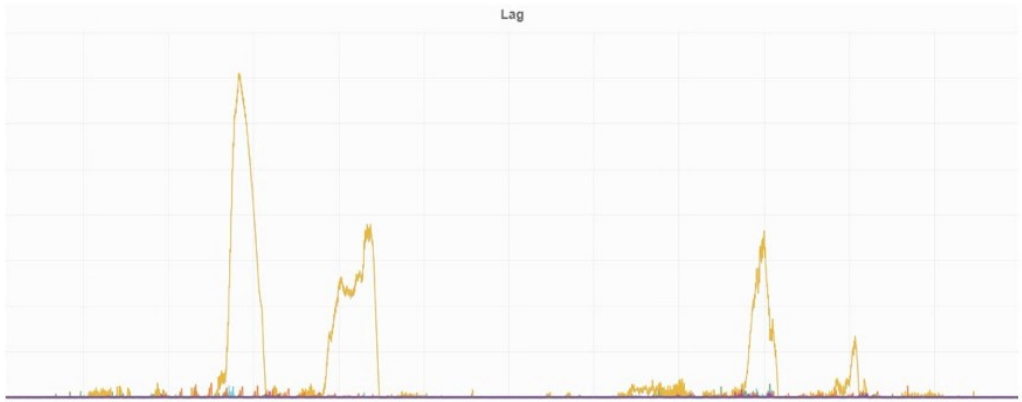
DEVIEW 2020

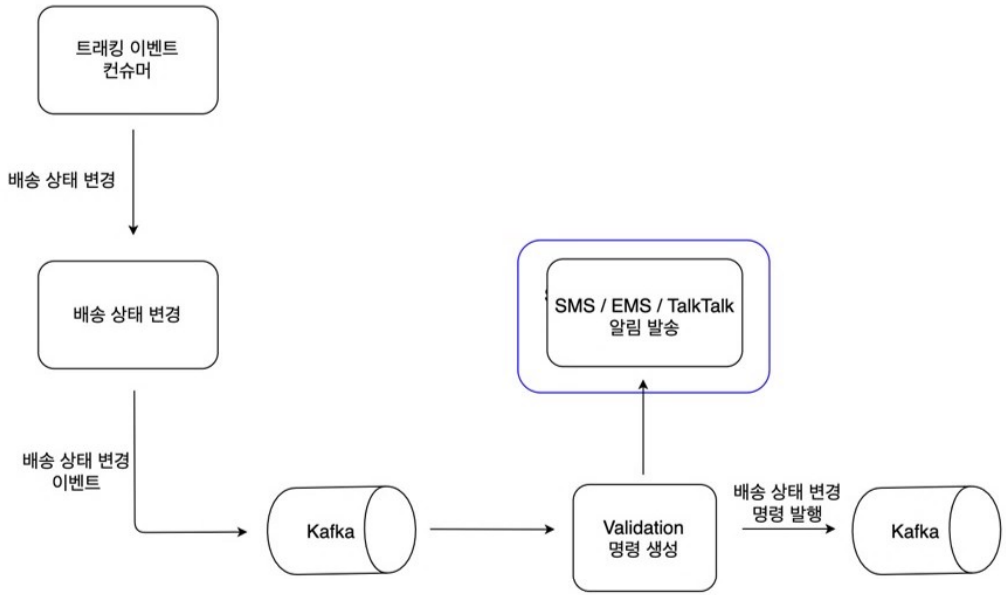
트래픽 증가로 인한 Kafka Lag 발생

- Consume 한 메시지 처리량보다 적은 연계 서비스 처리량으로 인한 Lag

Partition 을 늘려 병렬 처리량 증가 가능

- 연계 서비스 처리량 한계치



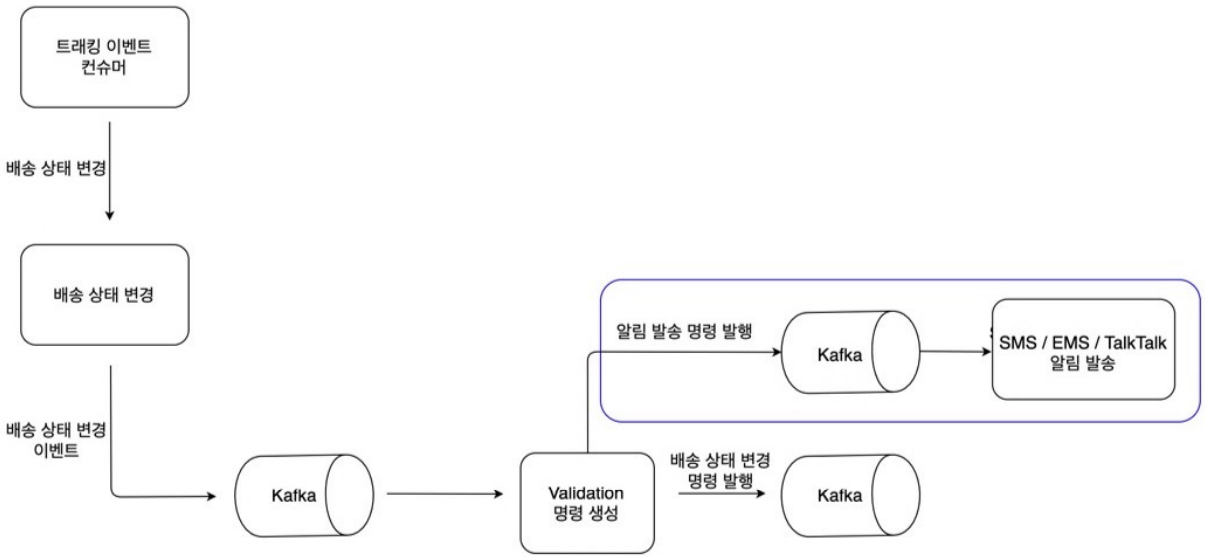


5.4 Kafka Lag 해소

DEVIEW 2020

트래픽 증가로 인한 Kafka Lag 발생

- 연계 서비스 호출 명령을 발행하는 Topic 생성
- 배송 상태 변경 명령 발행 처리는 연계 서비스 처리량과 독립
- 연계 서비스 처리량에 맞게 트래픽 조절



5.4 Kafka Lag 해소

Batch Consumer

- Consumer Listener 가 메시지를 List 단위로 처리

```
inbound-delivery-send-app-push-command:
  group: pay-delivery-send-app-push-command
  destination: delivery-send-app-push-command
  consumer:
    concurrency: 2
    header-mode: none
    auto-startup: false
    batch-mode: true
```

Kafka Transaction 미사용 설정

- List 자체가 Transaction 으로 묶임
- 중간에 처리 실패시 부분 성공 처리 불가

BatchListenerFailedException 에 실패한 index 전달

- Index -1 메시지까지 Offset Commit

```
@StreamListener(
    target = INBOUND_DELIVERY_SEND_APP_PUSH_COMMAND_CHANNEL
)
fun onDeliverySendAppPushCommand(commands: List<Message>) {
    commands.forEachIndexed { index, command ->
        try {
            when (command) {
                is RegularDeliveryDispatchSendAppPush ->
                    this.deliverySendAppPushService.regularDeliveryDispatchSendAppPush(command)
                is TrackingStatusSendAppPush ->
                    this.deliverySendAppPushService.trackingSendAppPush(command)
                else ->
                    logger.error("[SEND APP PUSH] This command can not send app push. command: $command")
            }
        } catch (ex: Exception) {
            val throwException = when (ex) {
                is ConstraintViolationException -> {
                    logger.error(ex.message, ex)
                    KafkaConsumerNotRecoverableException(ex)
                }
                else -> {
                    logger.error(ex.message, ex)
                    ex
                }
            }

            throw BatchListenerFailedException(throwException.message, throwException, index)
        }
    }
}
```

5.4 Kafka Lag 해소

DEVIEW 2020

Consumer Poll 횟수 감소

- 잦은 Poll, Offset Commit 등은 Offset Fetch 를 증가시키고 Broker CPU 사용량 증가

max.poll.records 설정을 통해 Poll 횟수 감소

- 너무 높게 잡으면 Rebalance 시간이 증가

메시지가 자주 발행되는 Topic 이 아닌 경우 IdleBetweenPolls 설정

- Consume 할 데이터가 없는 경우 반복적인 Poll 감소

```yaml
bindings:
  inbound-delivery-event:
    consumer:
      enable-dlq: true
      dlq-name: delivery-event-dlq
      startOffset: latest
      configuration:
        client.id: delivery-event-${random.uuid}
        max.poll.records: 3000
```

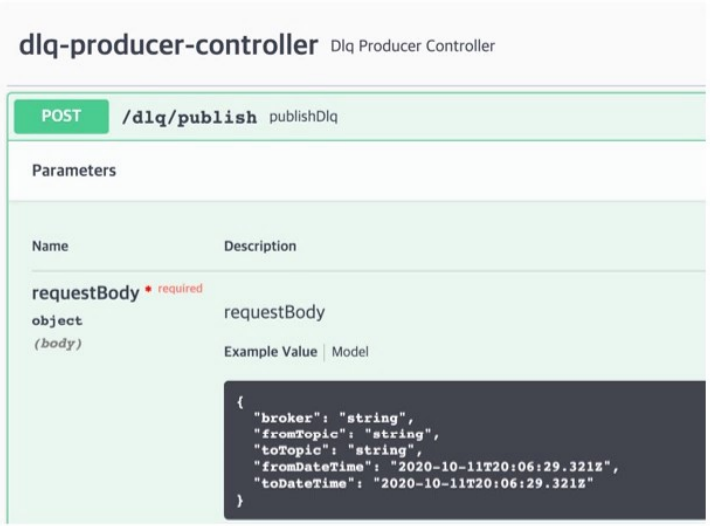
```kotlin
@Bean
fun listenerContainerCustomizer(
    applicationContext: ApplicationContext
): ListenerContainerCustomizer<AbstractMessageListenerContainer<*, *>> {
    return ListenerContainerCustomizer { container: AbstractMessageListenerContainer<*, *>, topic, group ->
        container.containerProperties.idleBetweenPolls = 100
```

5.5 보정

DEVIEW 2020

API 를 통해 Kafka DLQ 에 있는 Event 들을 재처리

- 멱등한 Event 처리 방식 필요
- Event 순서 역행에 대한 방어 필요



Kafka 에 없는 Event 는 Elasticsearch 에서 조회 후 Kafka 로 재발행

DB 상태 직접 변경

- 코드 로직을 수행하지 않기 때문에 위험하며 도메인 지식 필요

분산이 되는 만큼 불확실성이 커진다.

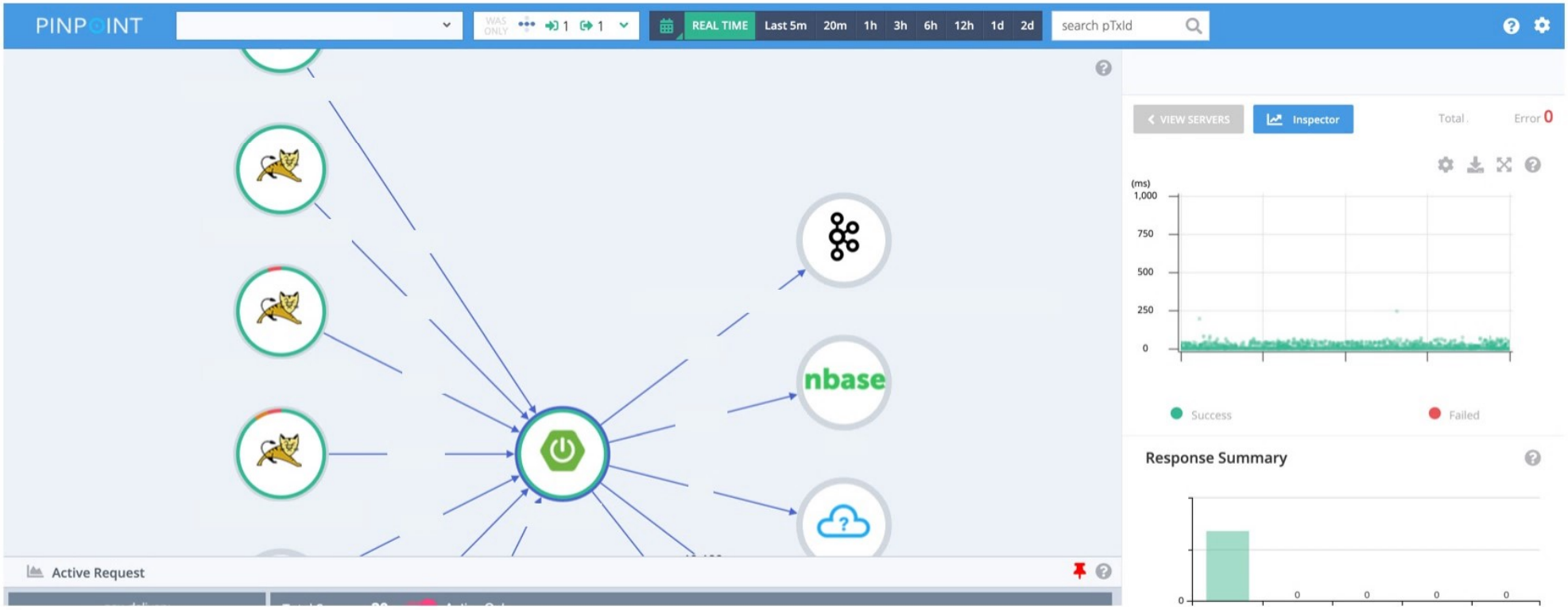
불확실성을 어떻게 보완할지가 가장 중요하며,
도메인이 가지는 특징에 맞는 아키텍처를 설계해야한다.

6. 도구 활용

6.1 Pinpoint

DEVIEW 2020

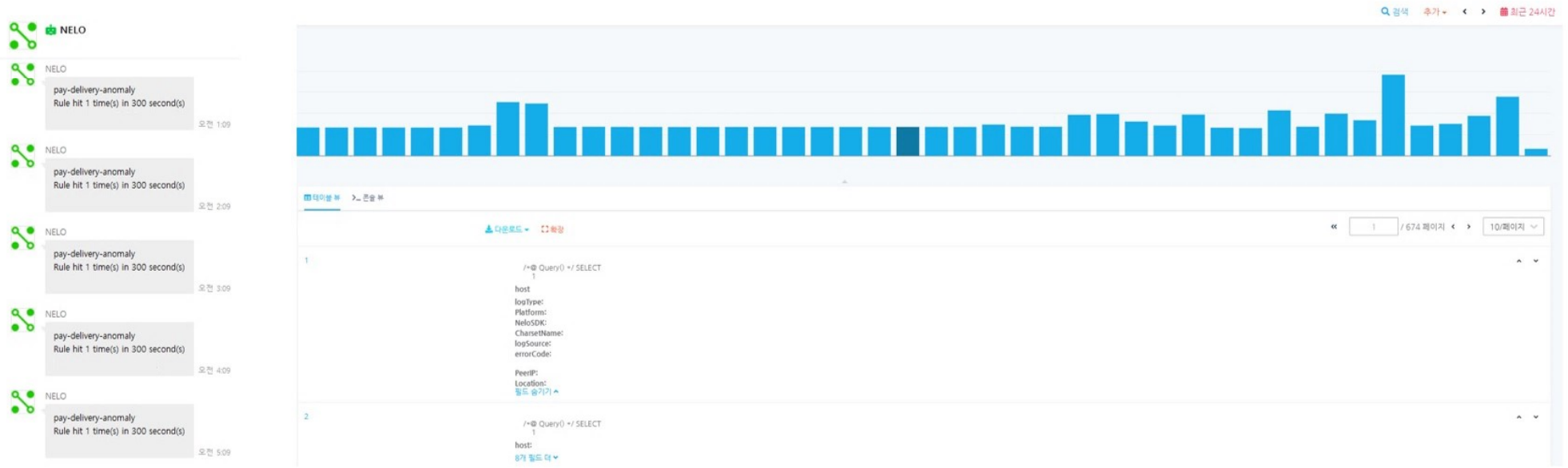
전반적인 트래픽 상황
요청 / 응답 처리속도 확인



6.2 NELO (로그 수집)

DEVIEW 2020

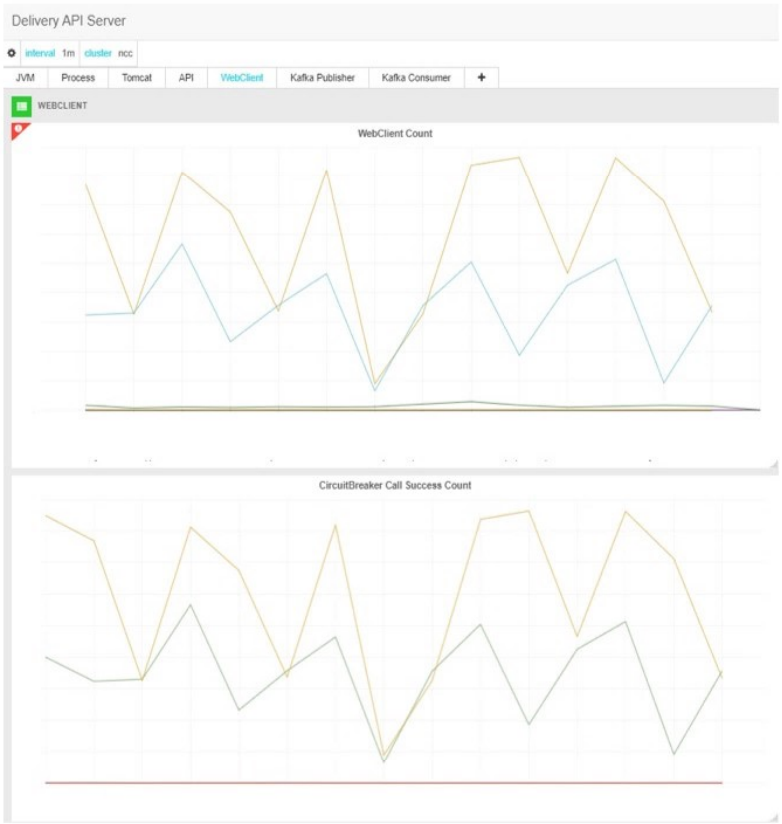
요청 ID, Exception Message 등 이슈 추적에 필요한 로깅
예외 발생시 발림

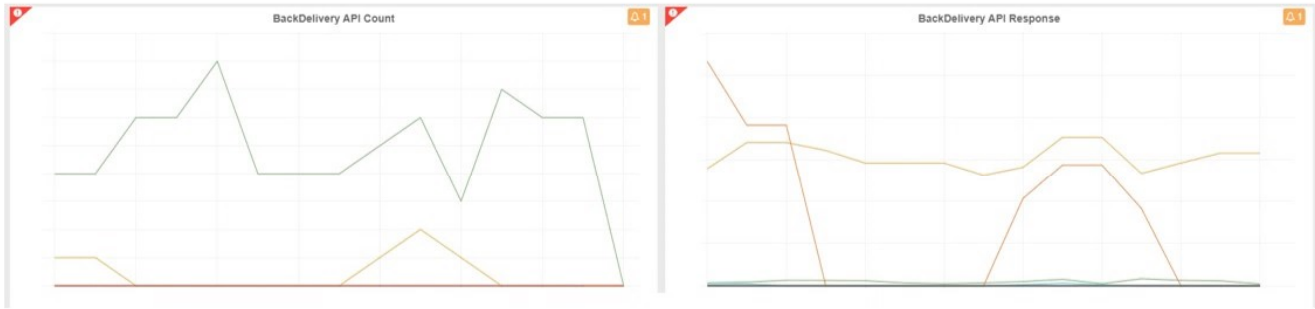


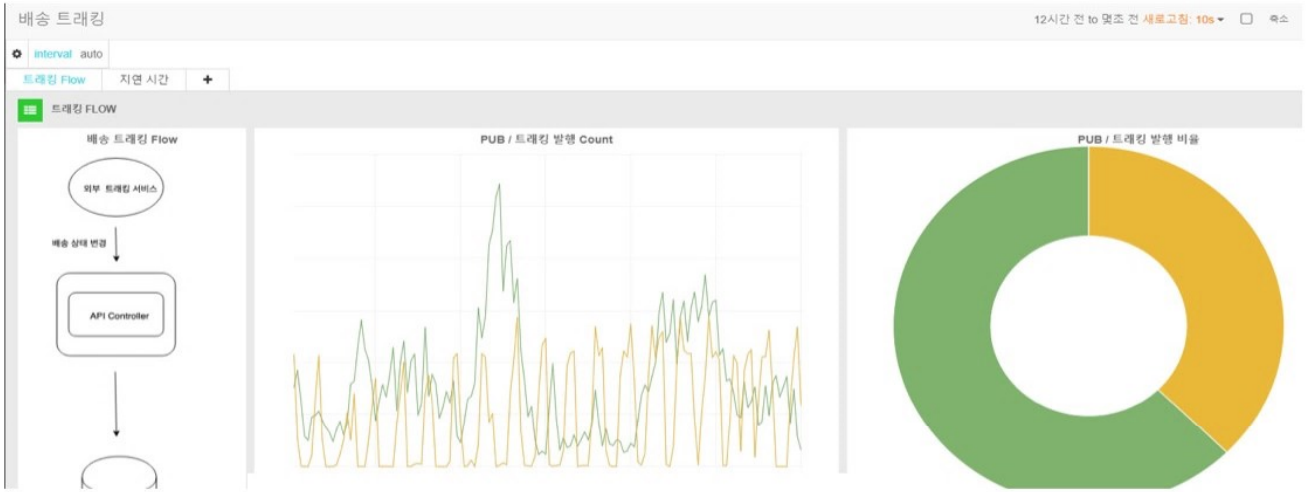
6.3 NPOT (Metric 수집)

DEVIEW 2020

특정 API, Kafka WebClient 등의 호출 상황
임계 수치를 통한 알림



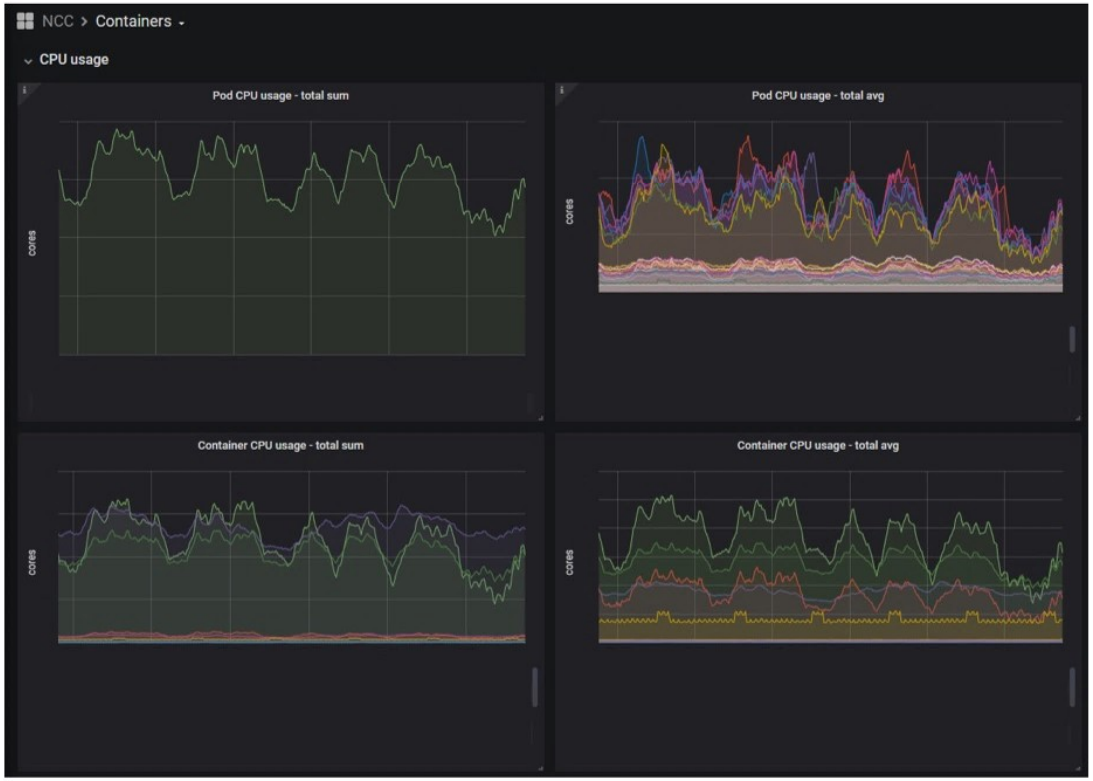


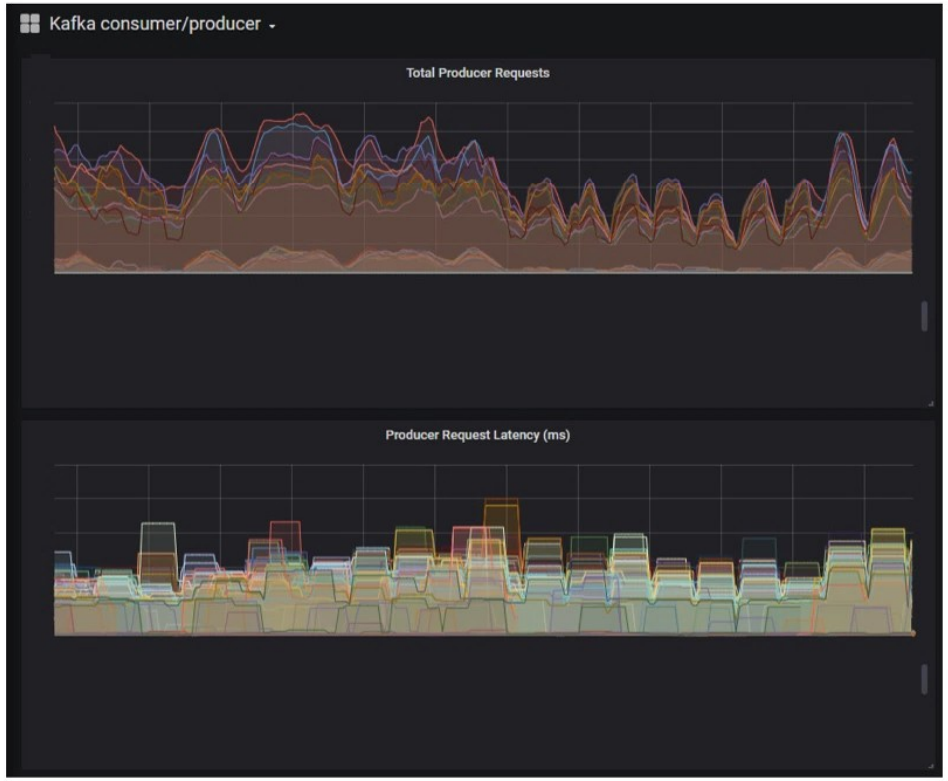


6.4 Grafana

DEVIEW 2020

Container 의 리소스 상황



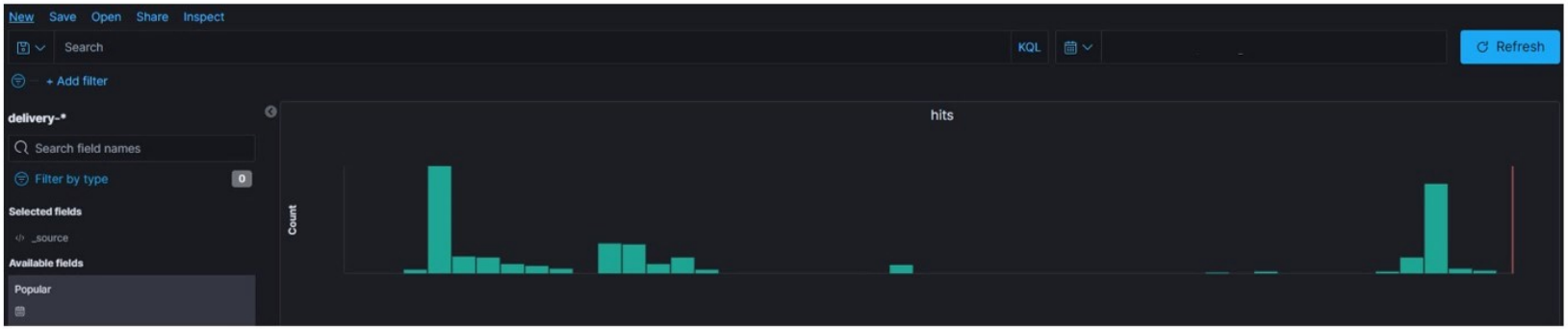


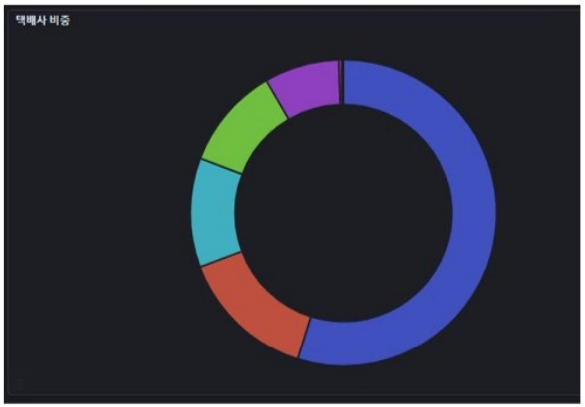
6.5 ELK(Elasticsearch Logstash Kibana)

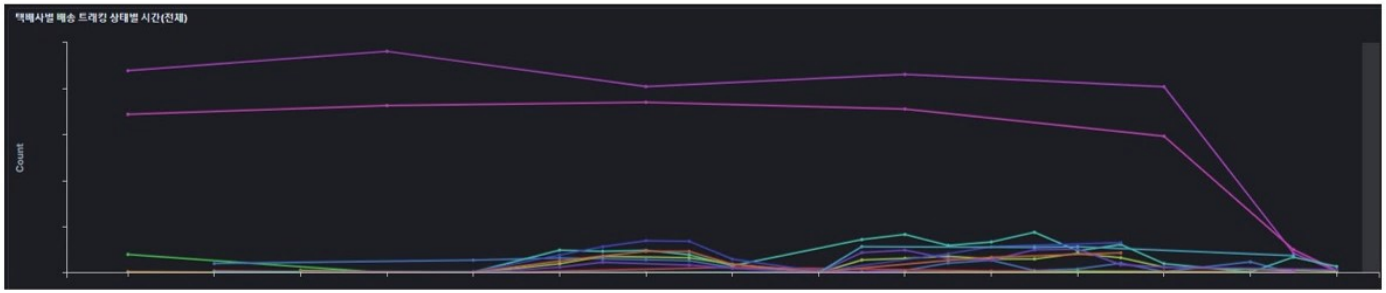
DEVIEW 2020

발행된 Event 모니터링

Event 를 활용하여 데이터 분석







Q & A

Thank You