DEVIEW
2019

스케일아웃없이 순간 급증하는 주문 처리하기 (Microservice with Kafka)

김 태 경
11번가 / Platform Engineering Team

CONTENTS

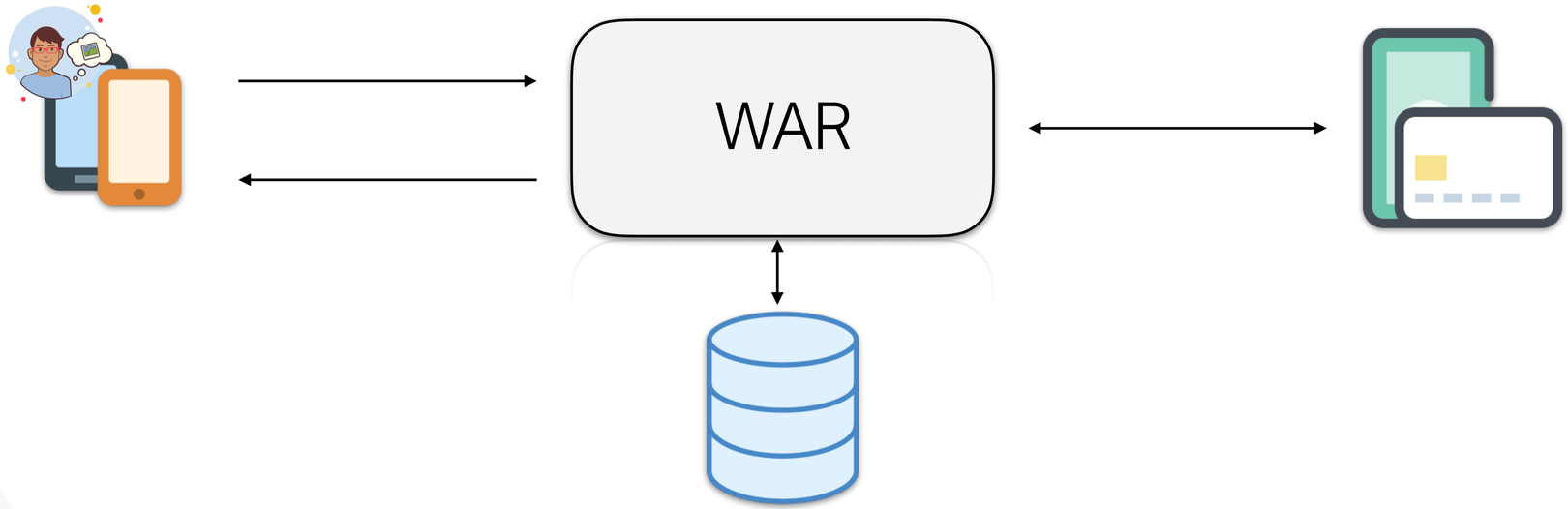
DEVIEW
2019

DEVIEW
2019

1. Microservice로의 전환

1.1 - 2016년 11번가

Monolithic Architecture



1.1 - 2016년 11번가

- 2008년 서비스 시작
- 200명 이상의 개발자
- 23,054 라인의 파일
- 10,000 라인이 넘는 23개 파일

그래서 …

3:00 AM 배포를 위한 출근

배포 시간만 수시간, 실패 시 토요일 새벽 재시도

1.2 - 2017년 MSA 으로의 전환

DEVIEW 2019

개발 가이드

Table of Contents

Vine Platform

1. Vine Platform Overview

11번가는 증가하는 거래량을 수용하고 변경에 빠르게 대응해야 한다는 과제를 가지고 있습니다.
하지만 기존 11번가의 Monolithic Architecture 는 이러한 요구사항을 충족시키는데 한계가 있습니다. 이에 대한 방안으로 Microservices Architecture 가 제시되었습니다.

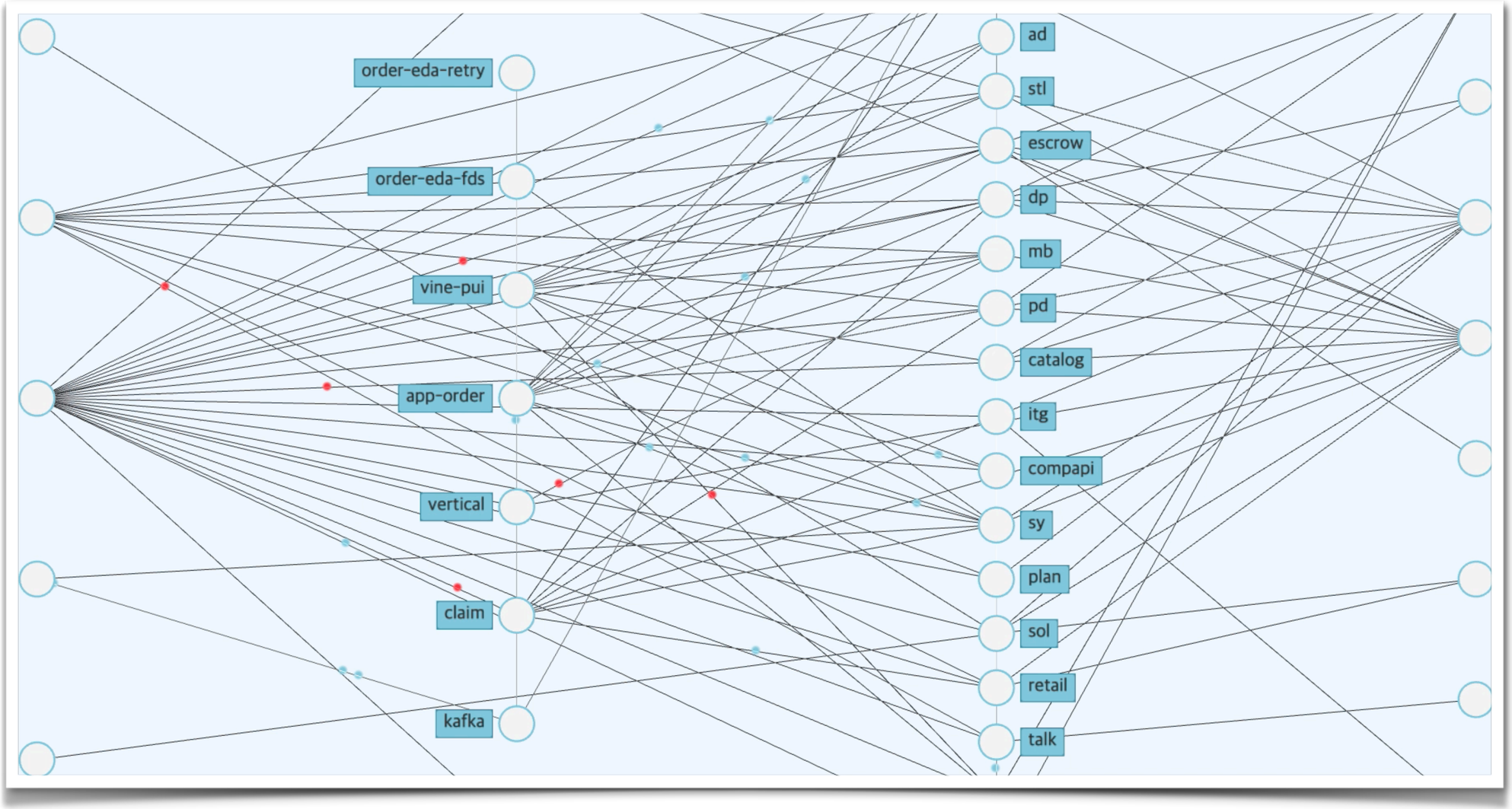
새로운 아키텍쳐 아래에서 개발된 API 서버들을 Vine API Server라고 부르며, 이 서버들이 동작하기 위한 플랫폼을 Vine Platform이라고 부르고 있습니다. 본 문서에서는 Vine API Platform에 대한 개요, 구성과 개발/운영에 필요한 필수 정보를 제공합니다.

개발 환경

- SpringBoot 2.1.8
- Cloud Greenwich.SR3
- Spring Cloud Netflix (Zuul, Hystrix, Eureka)
- Spring Cloud gateway
- ~~Zuul2~~ , ~~Resilience4j~~

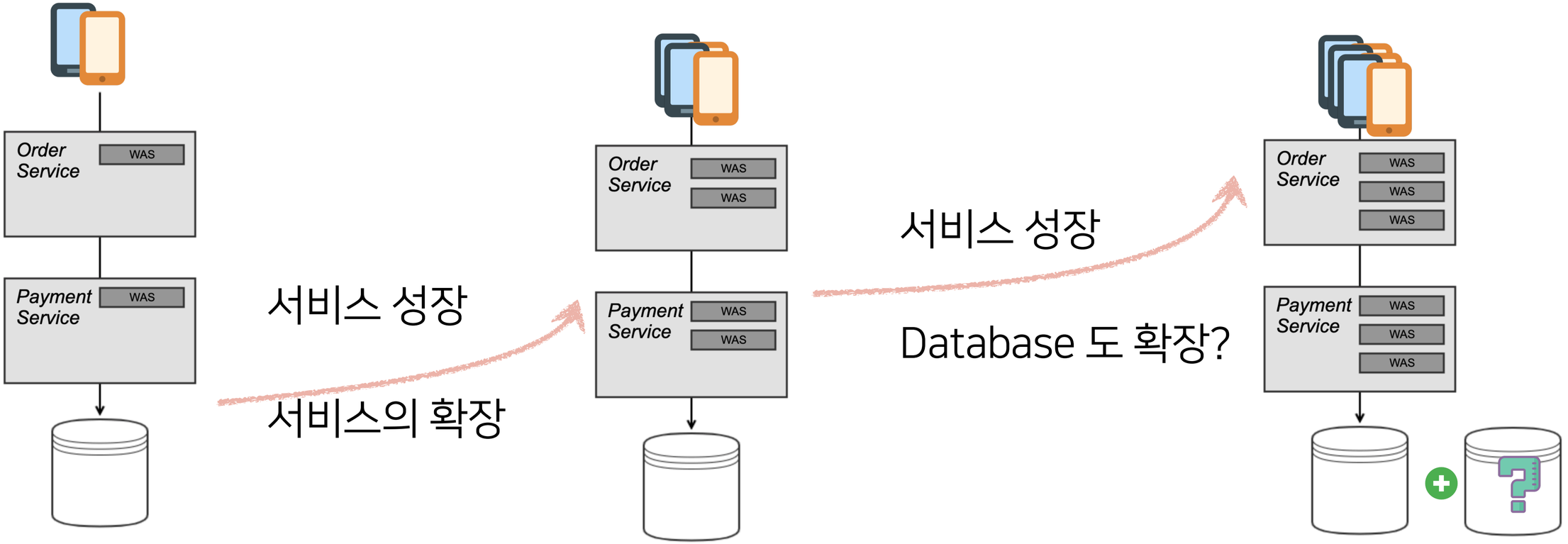
1.2 - 2017년 MSA 로의 전환 (a.k.a. VINE)

DEVIEW 2019



1.3 - 서비스의 성장과 새로운 문제의 출현

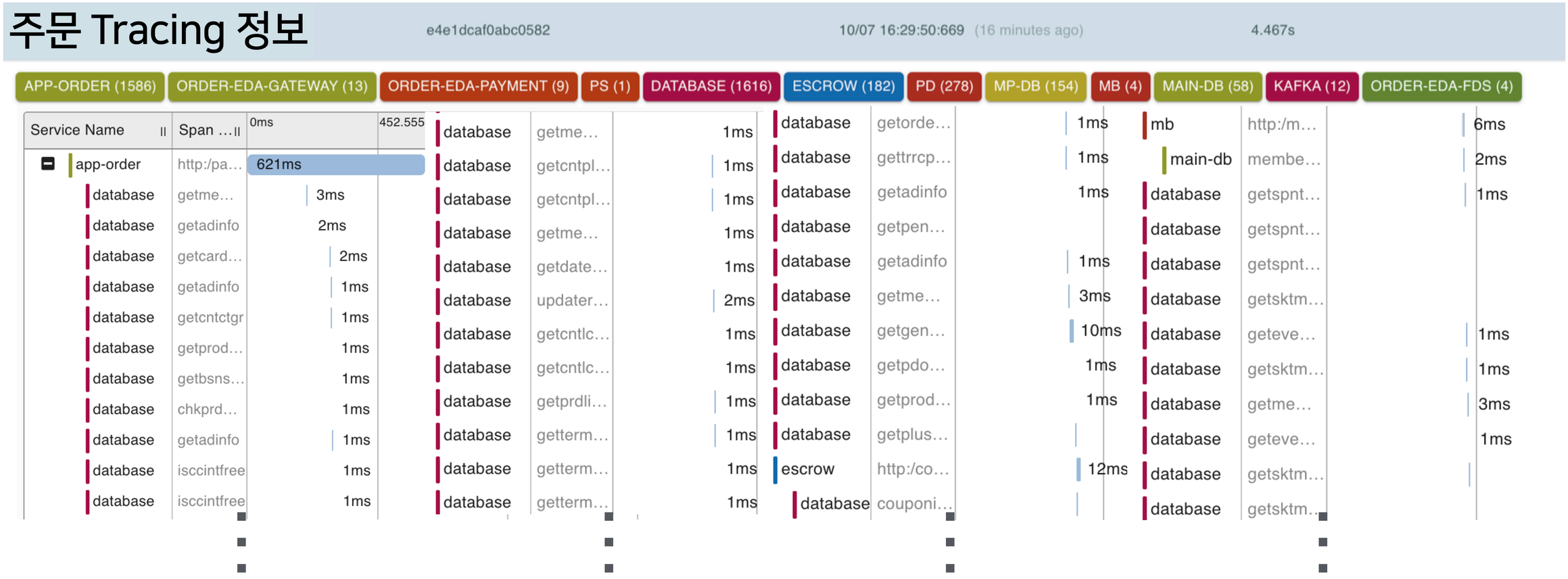
DEVIEW 2019



2. 문제의 분석과 해결 방법

2.1 - 주문,결제의 데이터베이스 부하

DEVIEW 2019



2.1 - 주문,결제의 데이터베이스 부하

DEVIEW 2019

주문 요청

주문 저장

결제 처리

이상 결제 체크

결제 진행 중입니다.

잠시만 기다려주세요.

Push/메일

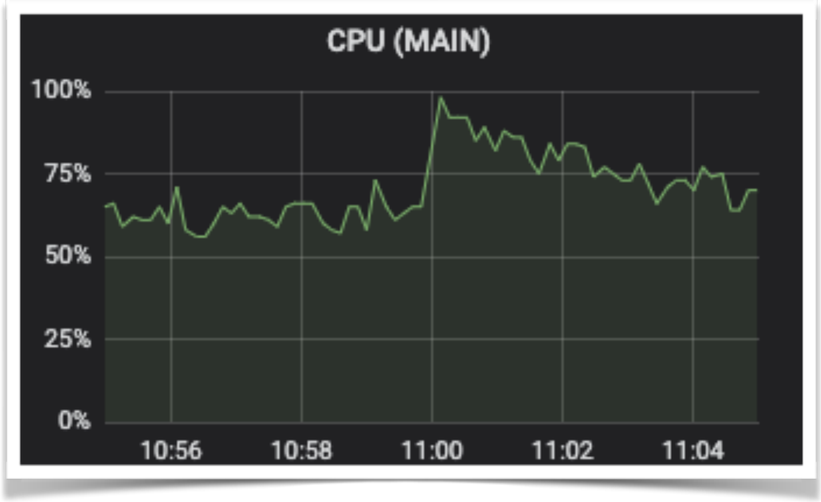
계정 Update

장바구니

주문 결과

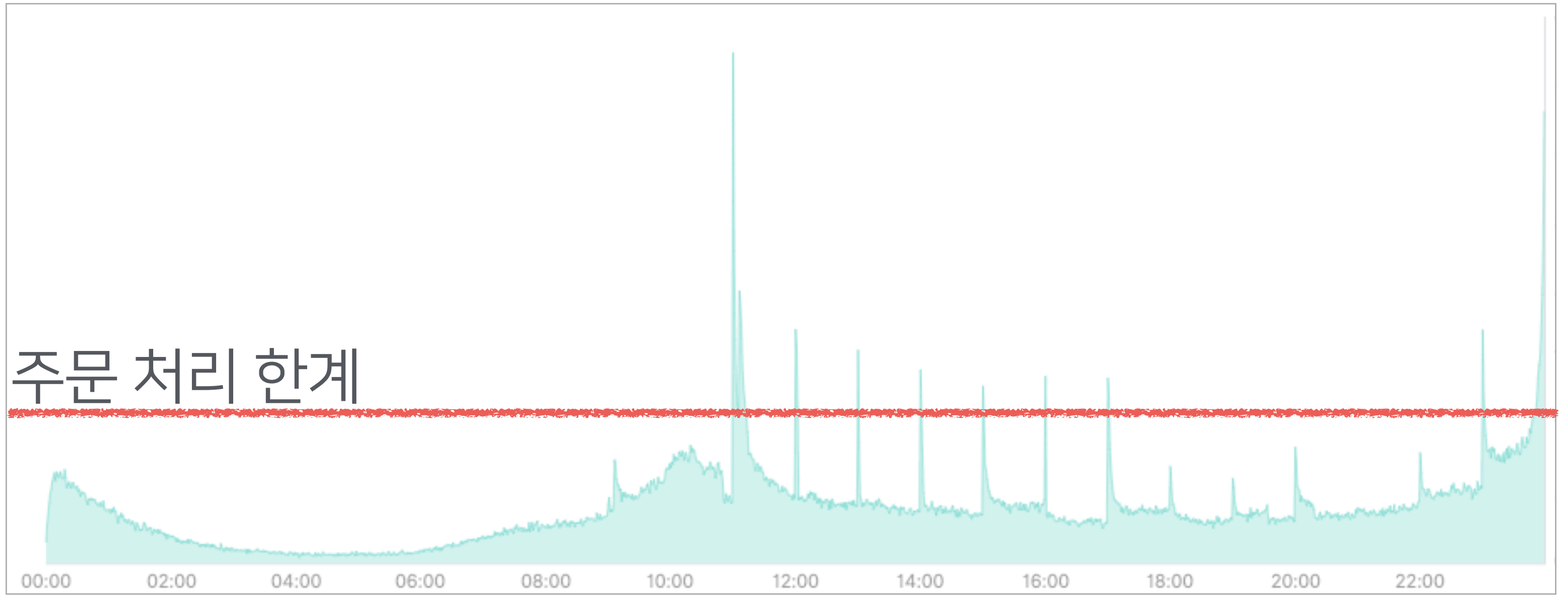
주문이 완료 되었습니다.

주문내역/정보를 확인하시기 바랍니다.



2.2 - 데이터베이스의 병목 해결 방법

DEVIEW 2019

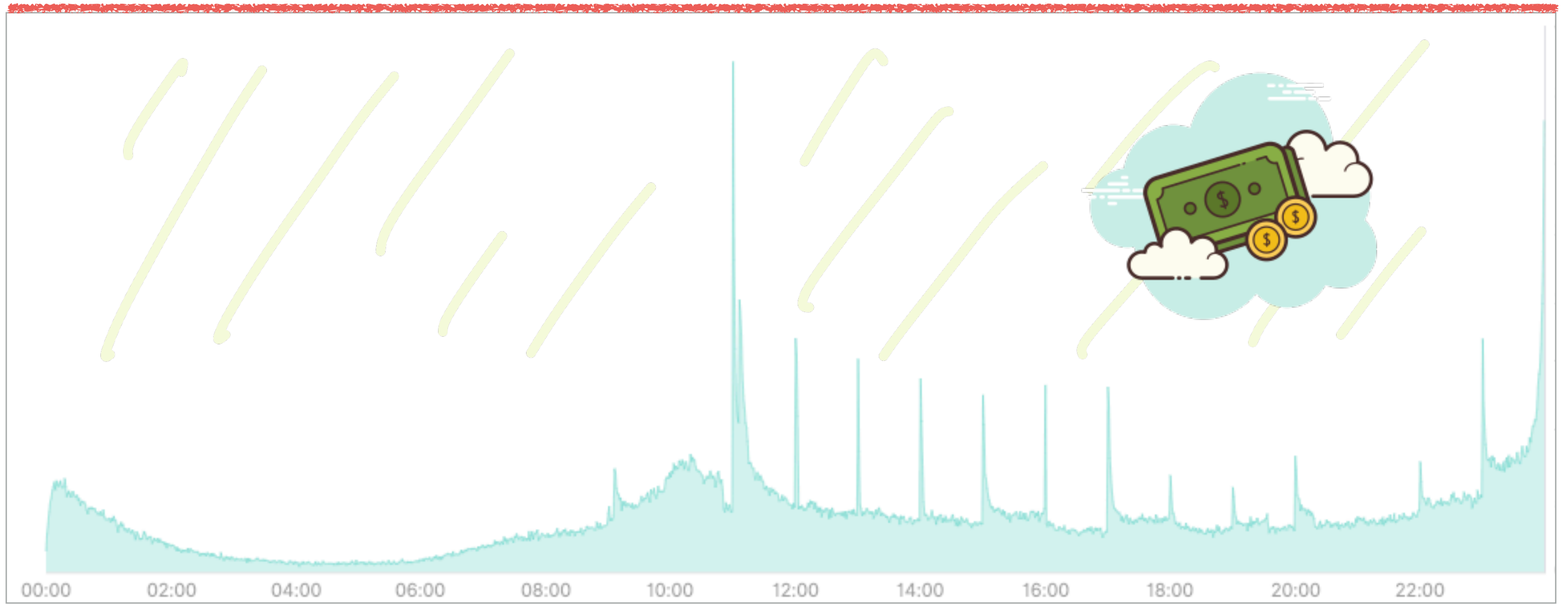


일일 주문량

2.2 - 데이터베이스의 병목 해결 방법

DEVIEW 2019

주문 처리 한계



일일 주문량

효율의 극대화

1. 쿼리 튜닝
2. Application 최적화

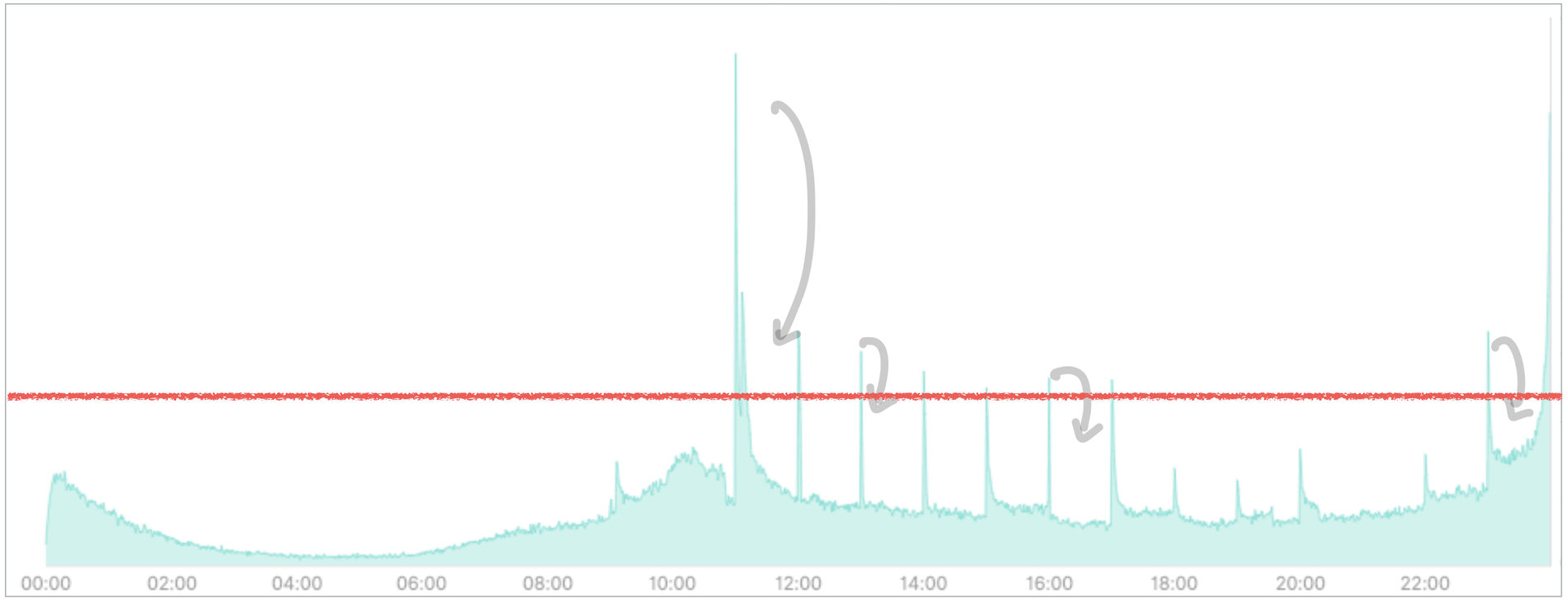
물리적 증설

1. Scale-up
2. RAC
3. DB 분리

2.2 - 데이터베이스의 병목 해결 방법

DEVIEW
2019

올릴수 없으면 선을 넘는 녀석들(주문)을 선 밑으로 내려오게 한다

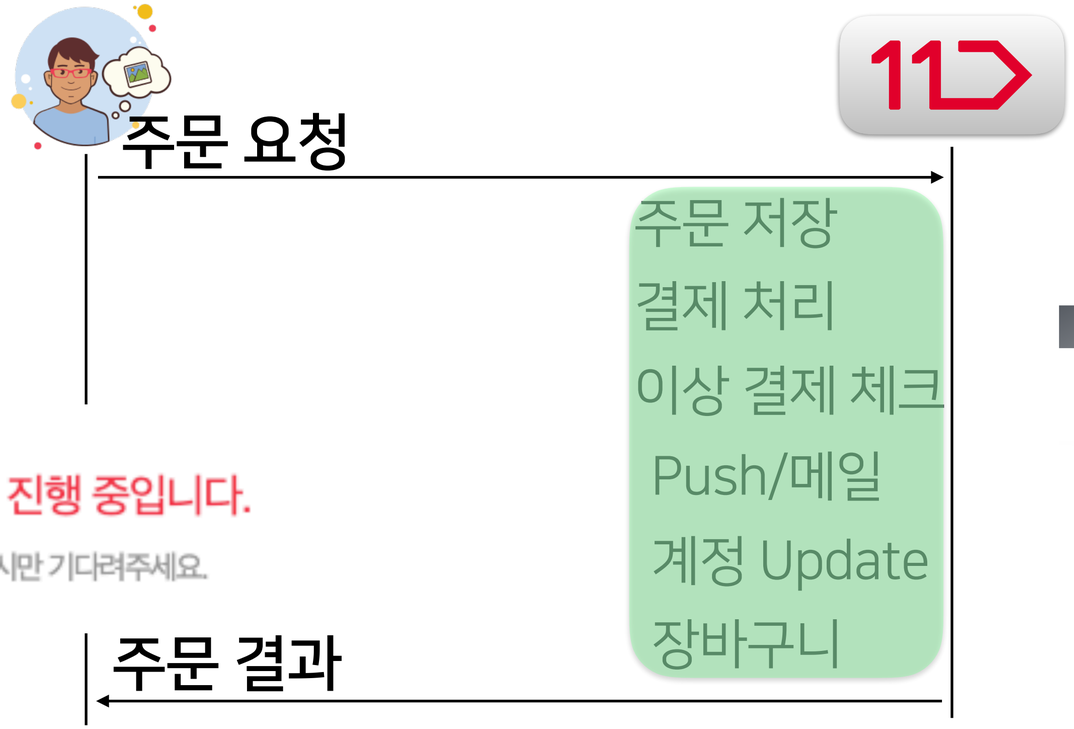


일일 주문량

How ?

2.3 - 트랜잭션의 분리, 구간별 아키텍쳐의 분리

DEVIEW 2019

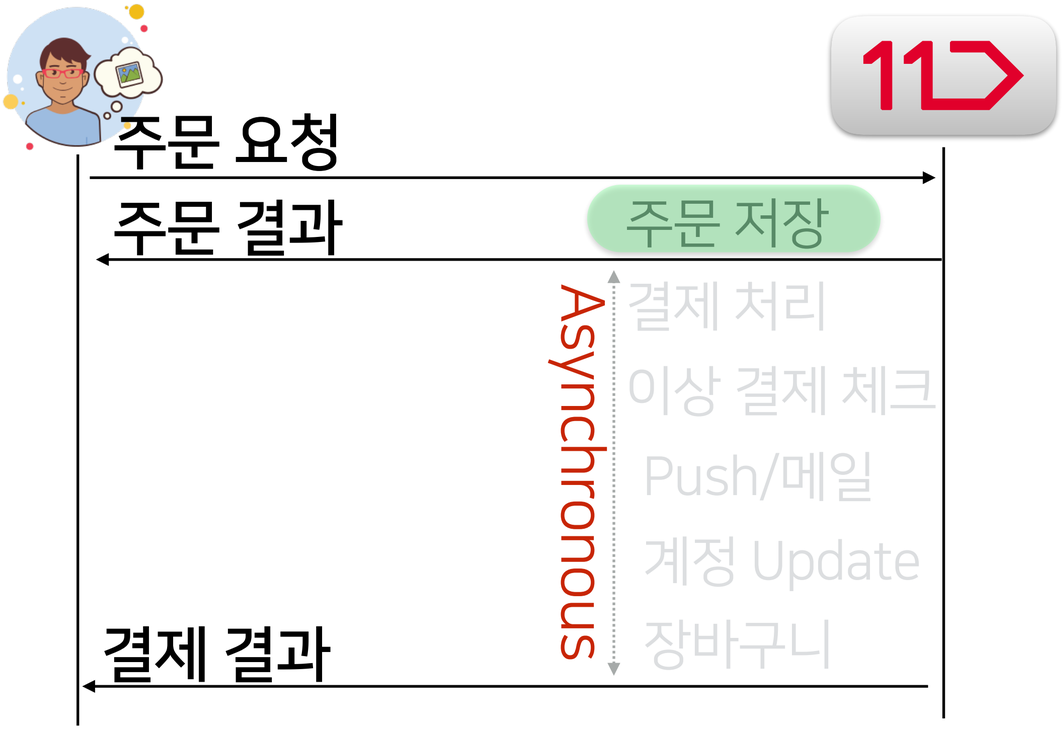


결제 진행 중입니다.

잠시만 기다려주세요.

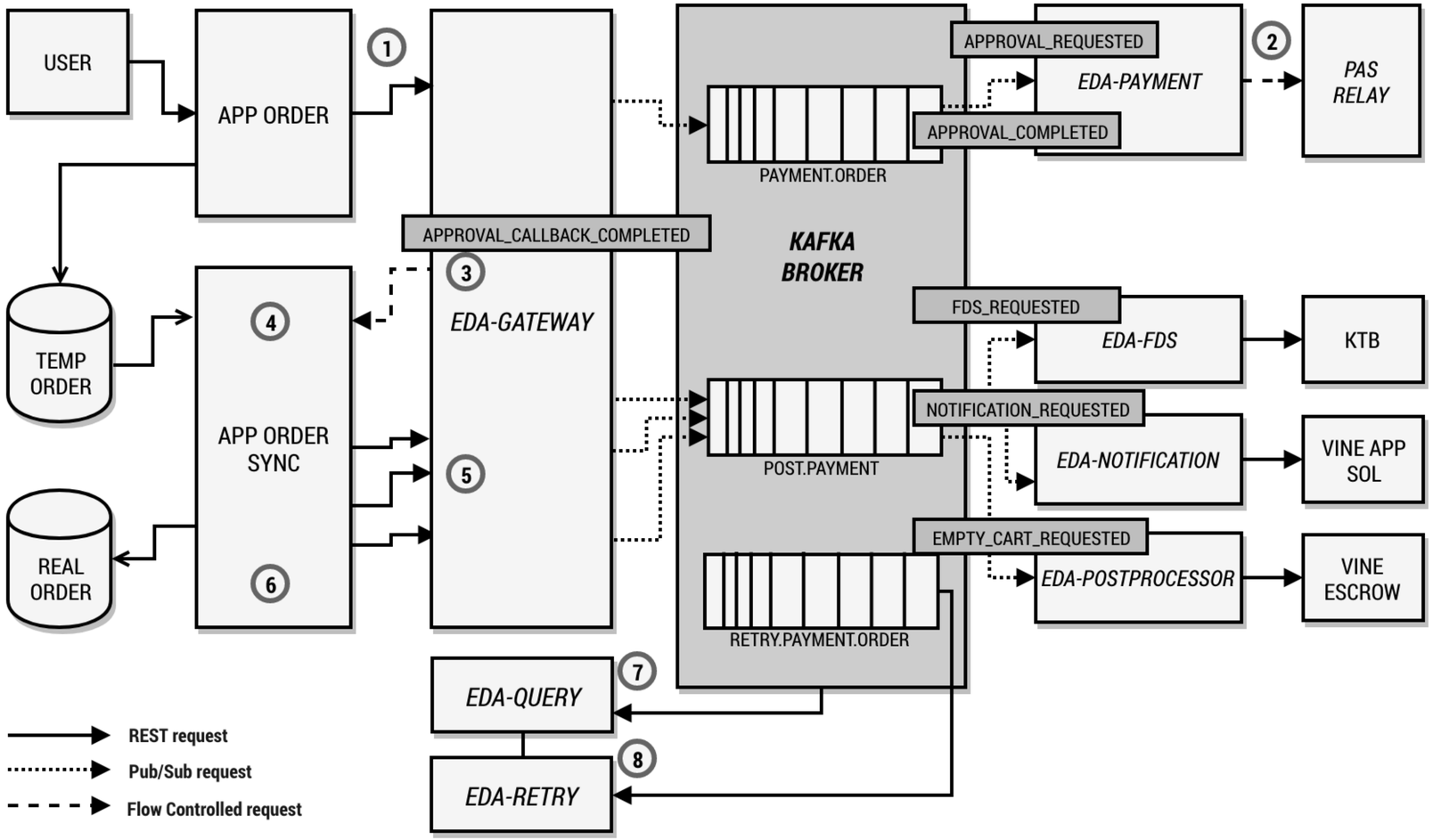
주문이 완료 되었습니다.

주문내역/정보를 확인하시기 바랍니다.



2.4 - 11번가 주문/결제 아키텍쳐

DEVIEW 2019

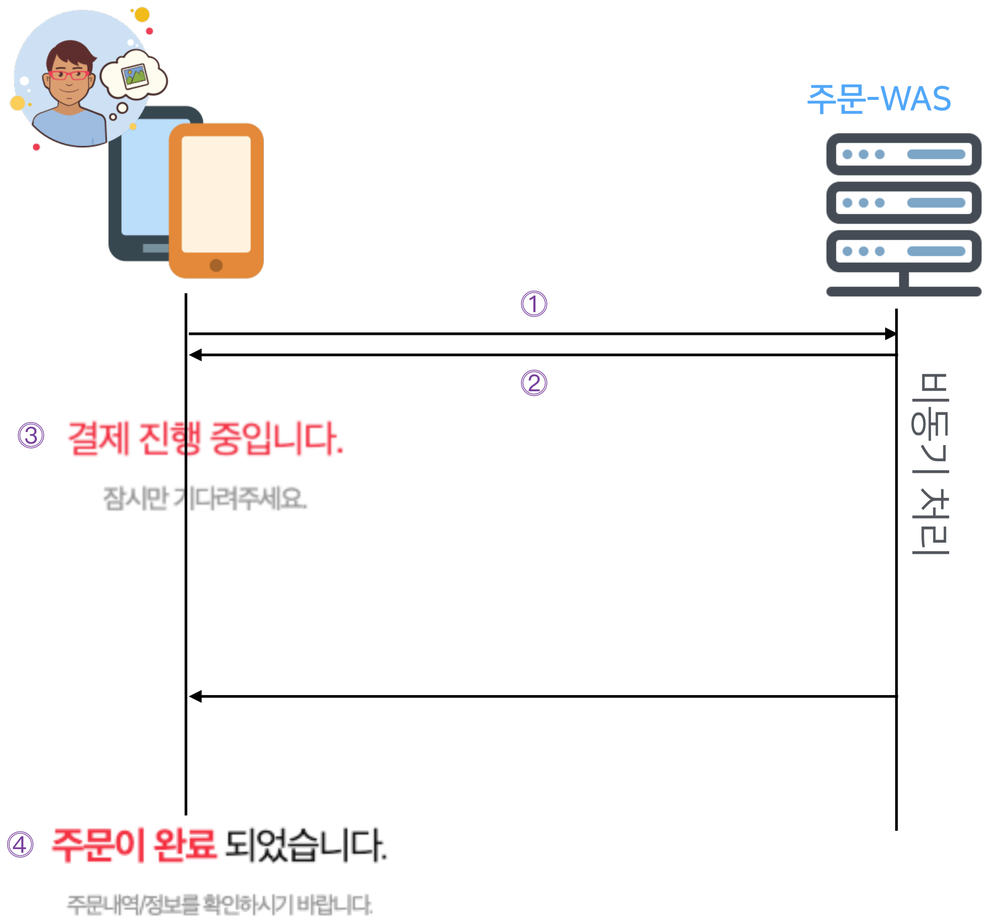


1. AppOrder 주문 결제승인요청 호출
2. PAS 결제 승인 요청 w/flow_control
3. AppOrderSync 에 콜백 요청 w/flow_control
4. 콜백 후 *가주문* -> *진주문* 전환작업 시작
5. FDS, Mail,SMS, 장바구니 삭제 와 같은 비동기 작업
6. 진주문 처리 완료, 사용자 주문 완료
7. Event를 Aggregate 하여 조회 API 제공
8. 실패한 Callback에 대해서는 재시도

- 진주문 실패시 결제 취소는 이전처럼 직접 호출

2.4 - 11번가 주문/결제 아키텍쳐

DEVIEW
2019



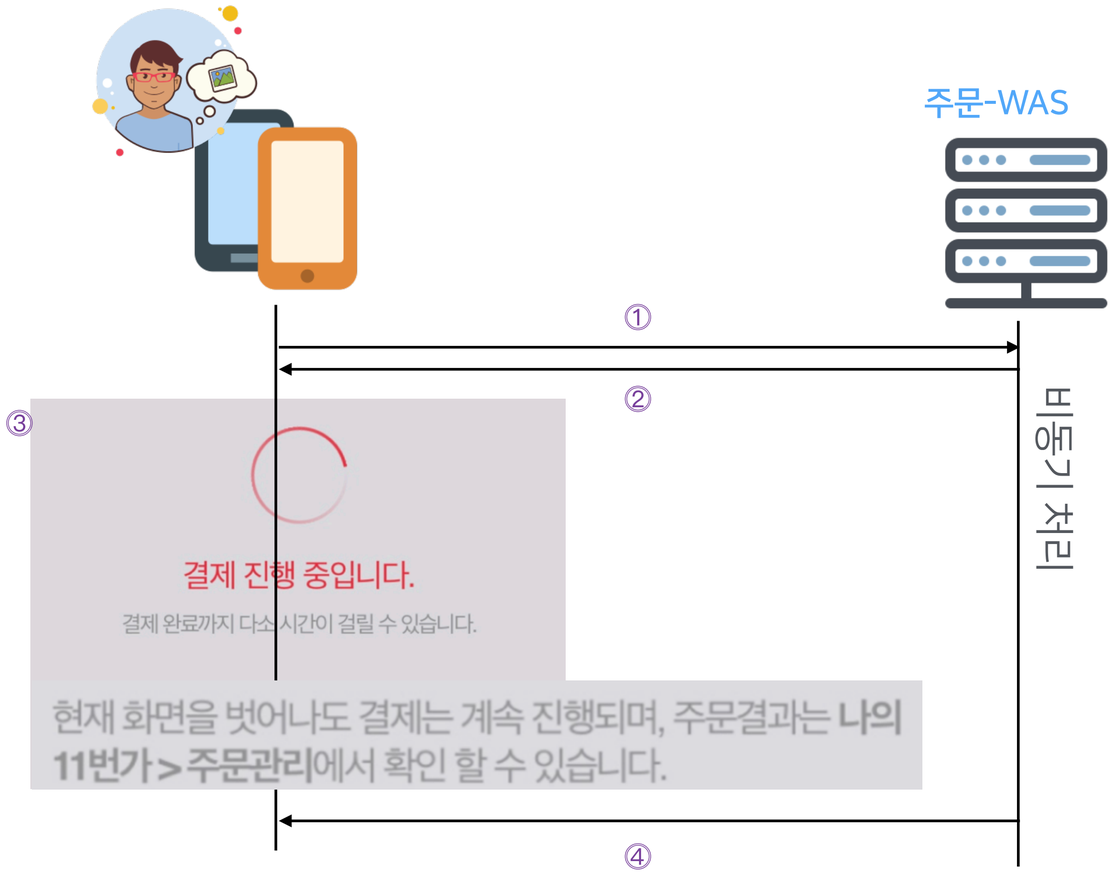
#일반적인 상황

사용자가 주문을 하더라도 결제가 즉시 처리 된다

변화된 것은 없다

2.4 - 11번가 주문/결제 아키텍쳐

DEVIEW 2019



선착순 결제 이벤트 상황 (처리 한계 넘을 경우)

① 사용자가 '주문' 을 함

② 카프카에 저장후 즉시 Okay 응답

③ 비동기로 결제 진행

결제 중에도 현재 화면을 벗어 나도 괜찮다는 가이드

④ 결제 결과 통지

주문의 대기시간이 사라지고,

결제 승인까지 화면을 보고 있을 필요가 없다

DEVIEW
2019

3. Kafka 기반의 서비스 구현

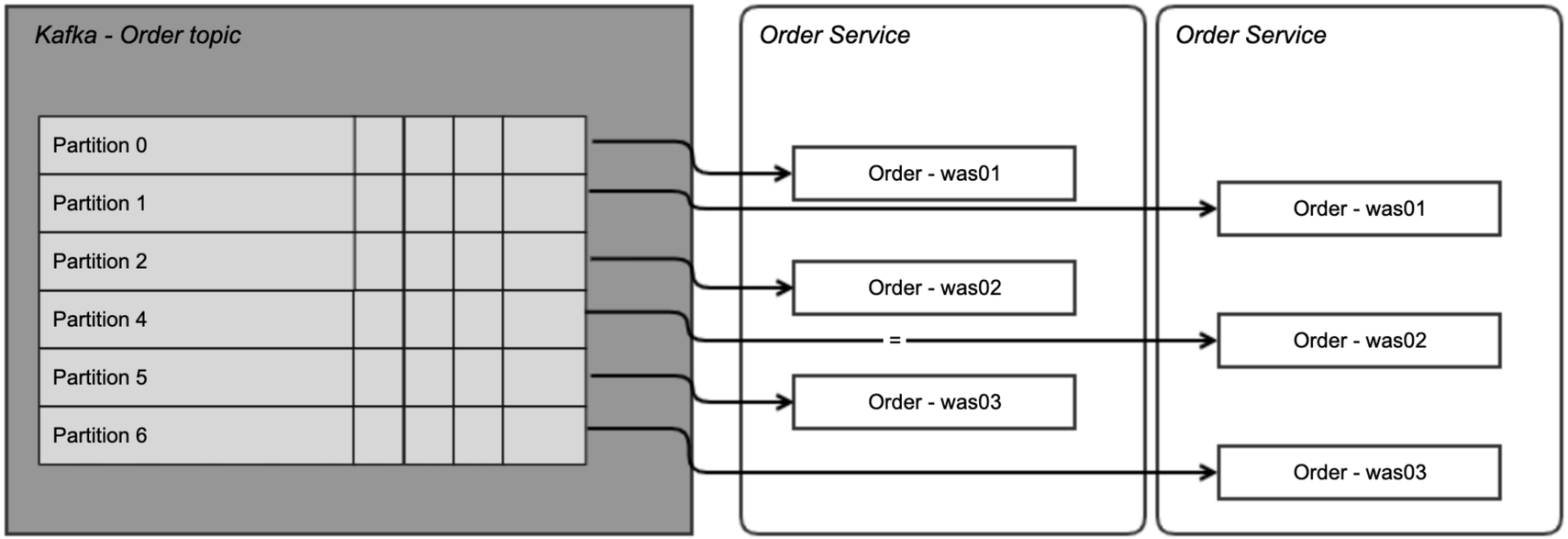
1. Consumer Group

   Consumer 들의 집합, 여러개의 Partition 으로 이루어진 토픽을 Partition 별로 담당해서 메시지를 처리한다. 메시지는 Consumer Group 내에서는 단 한번 처리된다

2. Rebalance

   Consumer Group 에 Consumer 가 추가, 삭제되면 처리하는 Partition을 재조정(Reassign)하는 작업
   Kafka 1.X 에서는 이 작업이 일어나면 전체 Consumer가 메시지 Polling을 순간 중단한다

3.1 - Consumer Application 배포 방법

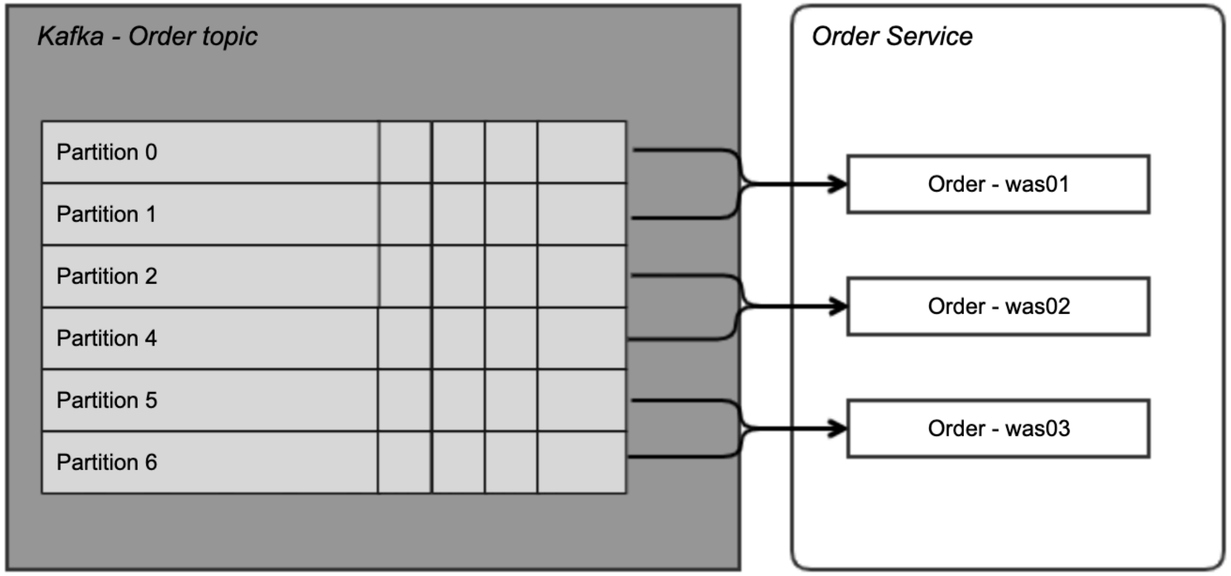


Consumer를 포함하는 서비스의 Blue/Green 배포

- Blue Green 배포 위해 한 세트를 준비 하는 순간 - 파티션 할당 후 동작
- 파티션 개수의 2배를 준비 ?

3.1 - Consumer Application 배포 방법

DEVIEW
2019

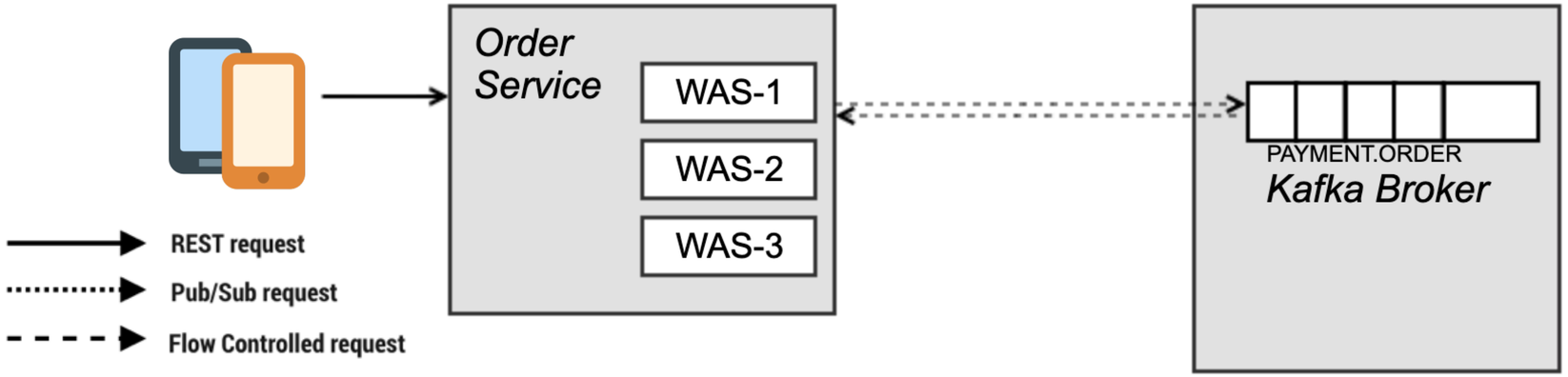


Consumer를 포함하는 서비스의 Rolling Deploy

-1번을 Down 하면 파티션 재분배 -> 전체 서비스 순단

-2번 배포… 3번 배포…

3.1 - Consumer Application 배포 방법

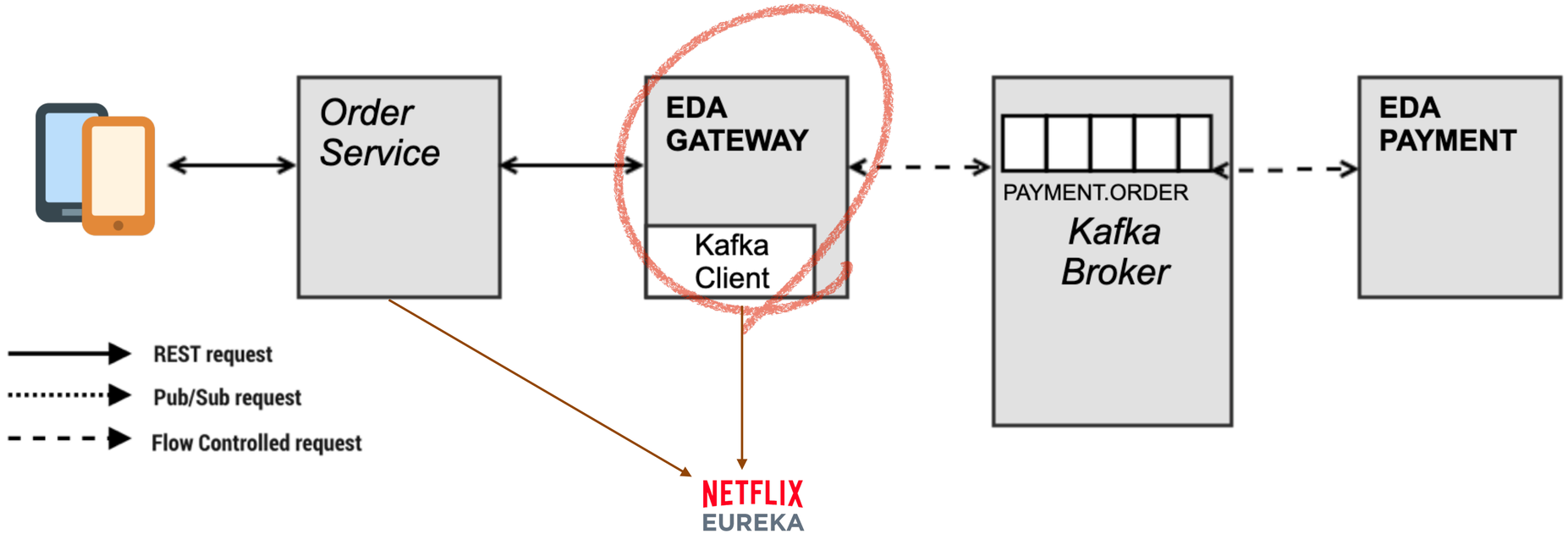


1. 서비스를 쉽게 배포 할 수 없다
2. WAS-1의 장애 WAS-2, WAS-3 까지 공동 장애 (리밸런스에 의한 순간 멈춤 현상)
3. 서비스 개발장에게 Kafka 관련 지식이 필요함
4. 인프라 계층과 의존성 : Kafka Version, Configuration, Broker …

3.1 - Consumer Application 배포 방법

DEVIEW 2019

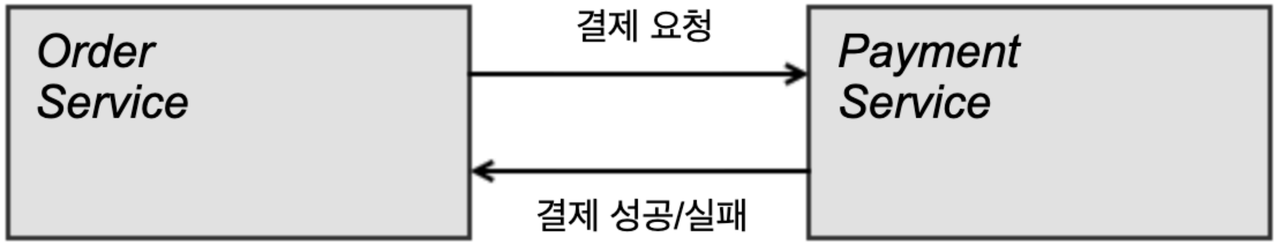
Kafka Client (Consumer) 모듈을 분리 하여 배포에 어려움이 없게 함



	Order Service	EDA Gateway
배포주기	하루 수회	가끔
Kafka	독립적	의존적
Type	Eureka/HTTP	Message/Avro

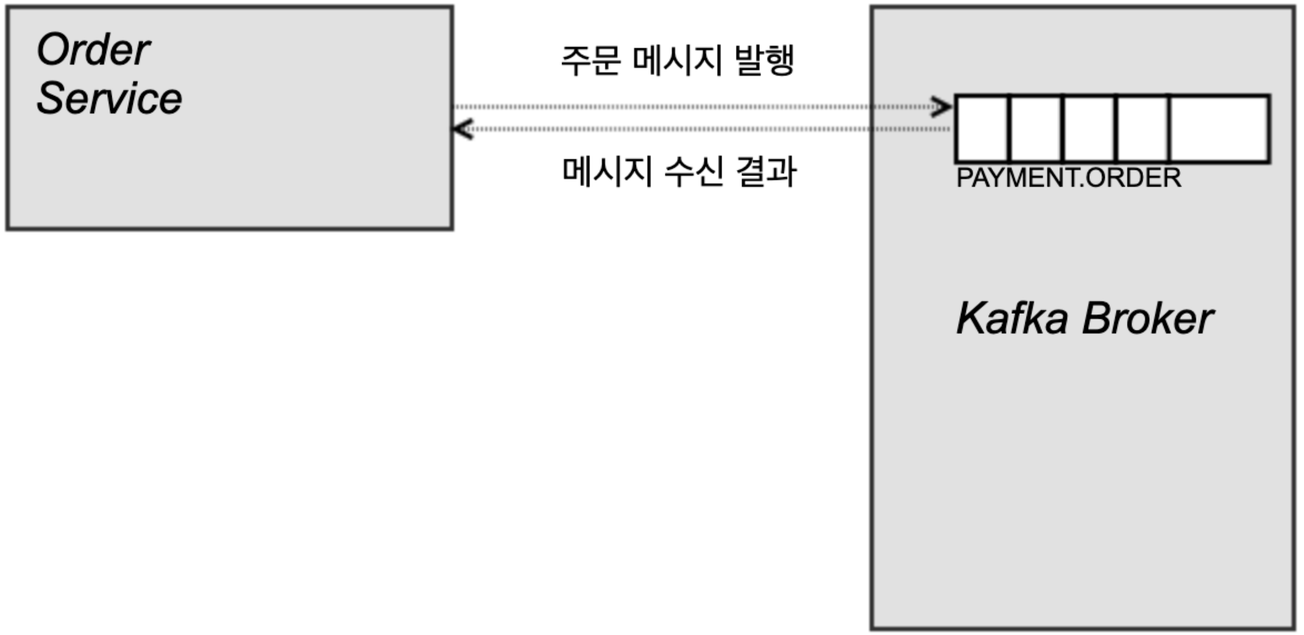
3.2 - 응답 메시지의 처리

REST



결제 요청을 보내면
처리결과를 받을 수 있다

Event
Driven

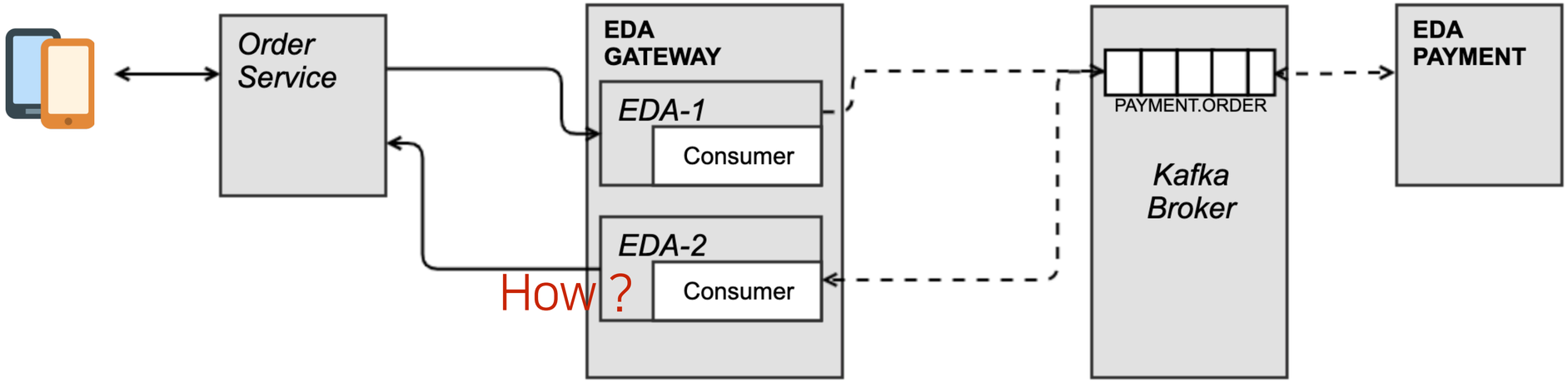


결제 요청을 보내면
카프카에 저장이 된 것을 안다

결제가 되는지 안되는지
언제 처리되는지도 모른다

3.2 - 응답 메시지의 처리

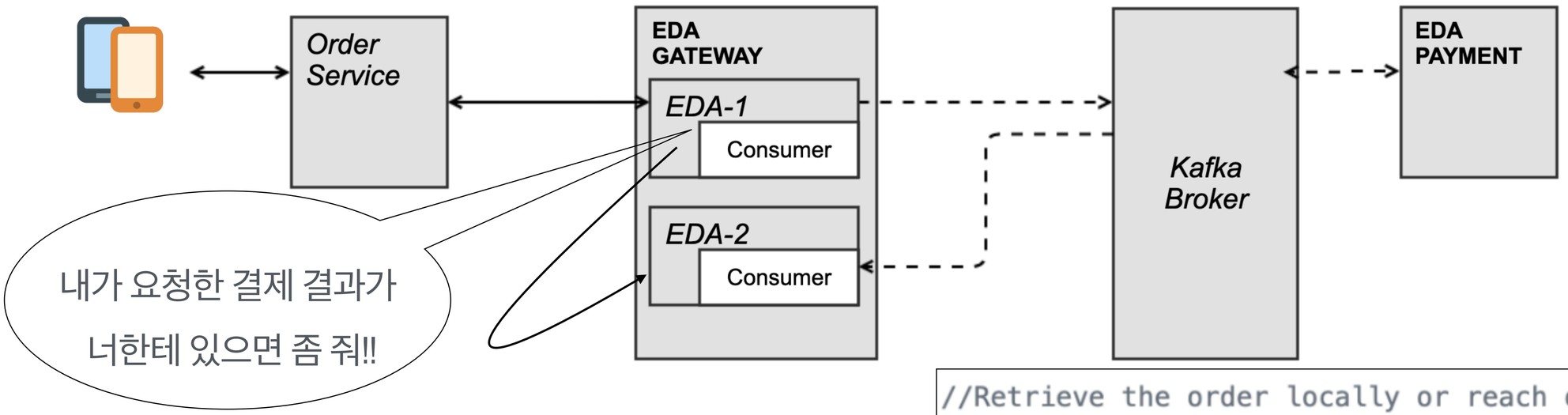
REST로 1번 서버에게 '결제 메시지'를 보냈지만, 응답 메시지는 2번 서버에 있다 ?



Message Send : murmur2(KEY) % Partition Number 로 저장 위치가 결정

Message Receive : partition assign 에 의존

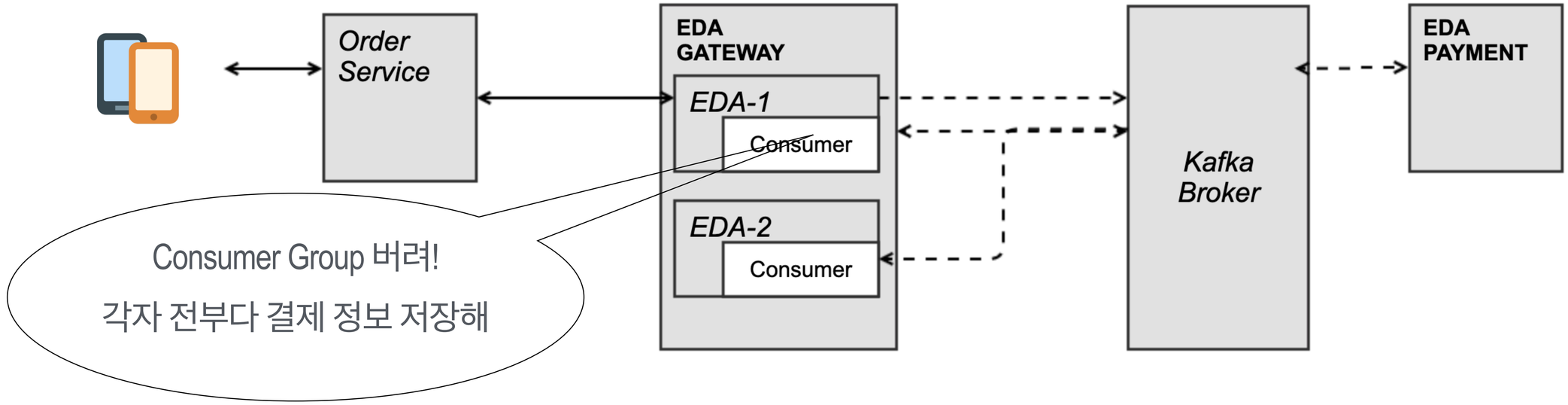
3.2 - 응답 메시지의 처리 방법#1



⚠ **결과 메시지가 있을 것 같은 옆 서버에 요청해서 받아온다**

```
//Retrieve the order locally or reach out to a different
if (thisHost(hostForKey)) {
  fetchLocal(id, asyncResponse,
      (k, v) -> (v.getState() == OrderState.VALIDATED ||
} else {
  fetchFromOtherHost(new Paths(hostForKey.getHost(), host
      asyncResponse, timeout);
}
```

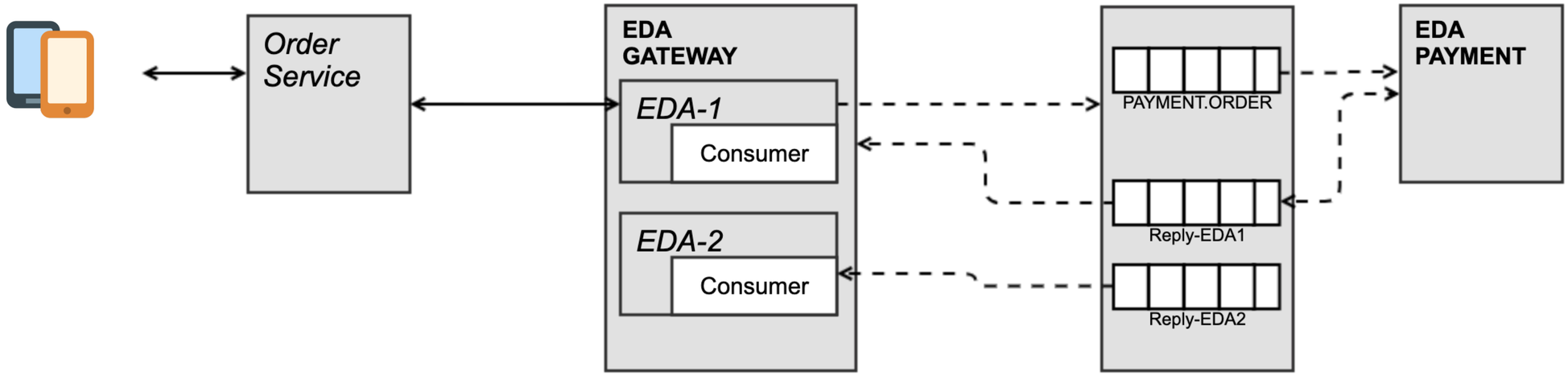
3.2 - 응답 메시지의 처리 방법#2



⚠ **Assign 된 Partition 뿐 아니라, 모든 Message 를 수신**

즉, 실제 처리해야 하는 양보다 몇배의 메시지를 처리해야한다

3.2 - 응답 메시지의 처리 방법#3

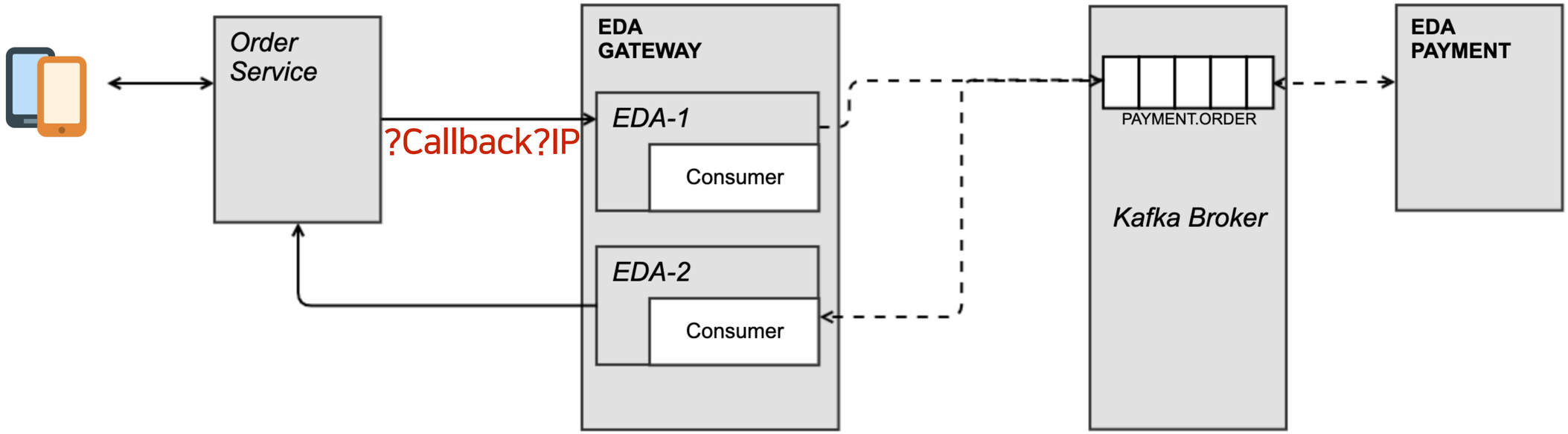


⚠ **Consumer 담당 Reply-Topic을 할당 한다**

- reply topic이 scale out의 병목
- topic 은 값싼 자원이 아니다
- topic에 데이터는 남았는데 담당 서버가 죽으면 누가 처리? (reassign issue)

3.2 - 응답 메시지의 처리 Solution

결과를 기다리지 않고, Callback IP를 통해 '통보를 받는' 형식으로 처리



3.3 - Backpressure

DEVIEW 2019

1. Tutorial

```
@KafkaListener
public void listen(ConsumerRecord<String, PaymentMessage> record) {
    sleep( seconds: 1_000);
    messageHandler.handleMessage(record.value());
}
```

- Polling
- 메시지의 처리가 끝나야, 다시 메시지를 Polling

처리 속도 : 1 TPS

3.3 - Backpressure

2. Multiple Consumer

```
@KafkaListener //@concurrency:4
public void listen(ConsumerRecord<String, PaymentMessage> record) {
    sleep( seconds: 1_000);
    messageHandler.handleMessage(record.value());
}
```

- 성능이 Partition 수에 의존적
- Partition 의 증가 -> 카프카 브로커, 서비스 Heap Memory 등에 영향을 준다
- **Rebalance / 순간 중단 시간 증가 (*)**

처리 속도 : ConsumerSize TPS

3.3 - Backpressure

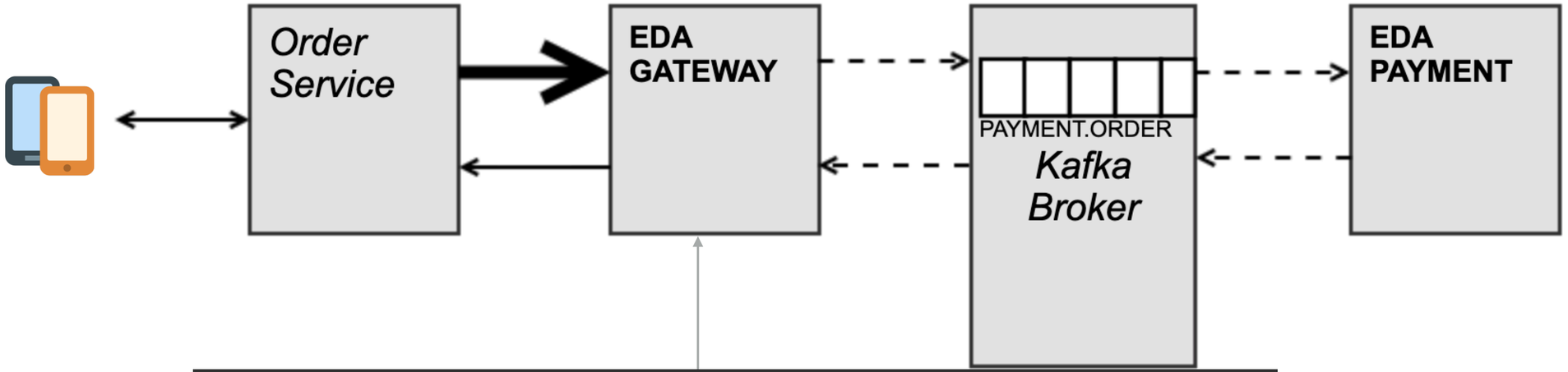
3. Consumer + ThreadPool

```
// tp = Executors.newFixedThreadPool(10);
@KafkaListener
public void listen(ConsumerRecord<String, PaymentMessage> record) {
    tp.submit(() -> messageHandler.handleMessage(record.value()));
}
```

- ThreadPool Size 변경을 통해 메시지 트래픽 제어 가능
- Partition 수에 덜 의존적임

gooood~!

3.3 - Backpressure

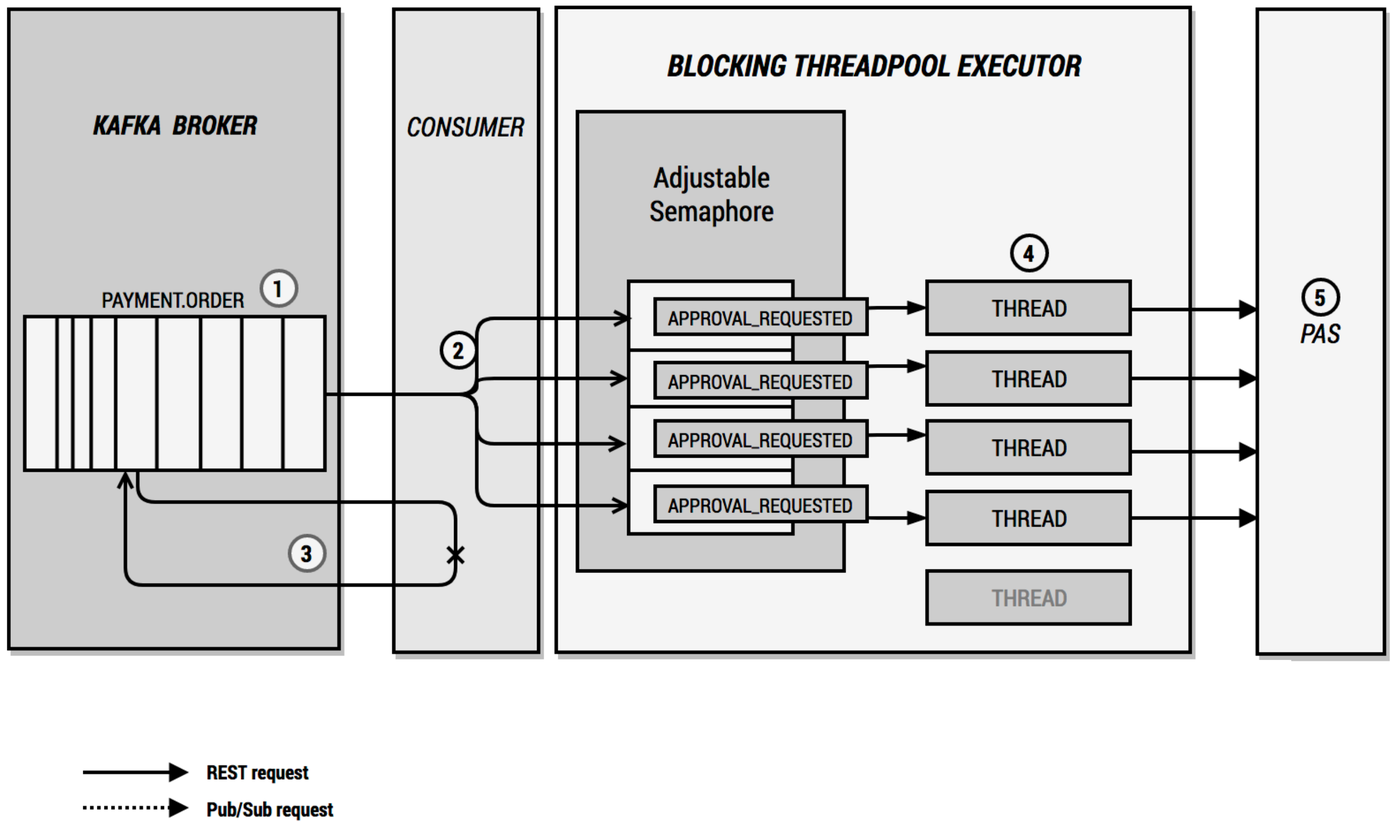



```
// tp = Executors.newFixedThreadPool(10);
@KafkaListener
public void listen(ConsumerRecord<String, PaymentMessage> record) {
    tp.submit(() -> messageHandler.handleMessage(record.value()));
}
```

ThreadPool 에서 수행되는 Task가 오래 걸리면 Consumer.poll()을 호출 안함

장애라고 판단되어 해당 인스턴스는 Consumer Group 에서 제거

3.3 - Backpressure



1. '승인 요청'메시지를 토픽에 적재
2. Semaphore 값에 따라 '승인 요청' 메시지 처리
   처리 완료시 Semaphore 반환
3. 허용하는 갯수 (max permit) 이상의 메시지는
   Block, 이 값은 Runtime에 조절 가능
   Offset을 되돌려 Poll Interval 을 짧게 보장함
4. Thread 갯수는 조절하지만 Queue가
   무한히 쌓이므로 Semaphore를 통해 조절
5. 아무리 주문 요청이 많아도
   처리 가능한 수준의 결제만 처리

3.3 - Backpressure

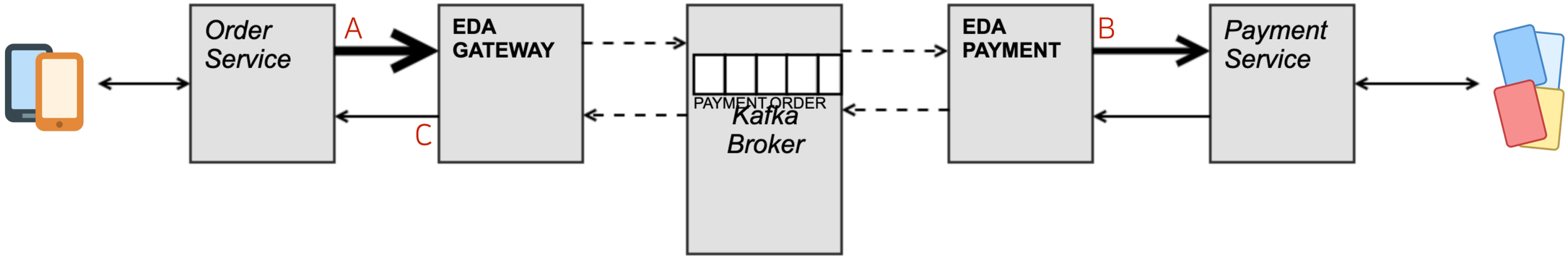

```
lic class BlockingThreadPoolExecutor extends ThreadPoolExecutor implements DisposableBean
 @Override
 public void execute(Runnable command) {
     blockUntilAcquireSemaphore();
     try {...} catch (RejectedExecutionException ex) {
         logger.error("Implementation Error - Rejection should not occur.", ex);
         semaphore.release();
         throw ex;
     }
 }
 private void blockUntilAcquireSemaphore() {
     boolean acquire;
     try {
         acquire = semaphore.tryAcquire(acquireTimeout, TimeUnit.MILLISECONDS);
     } catch (InterruptedException e) {...}
     if (!acquire) {...}
 }
```

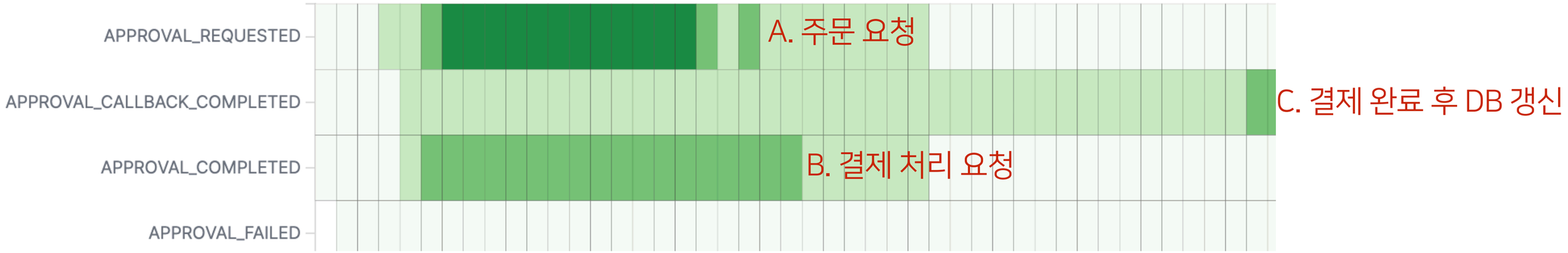
1. ThreadPoolExecutor 의 확장
2. Metric 수집
3. EndPoint 추가

```
//micrometer's metric
private void registerFailedCountAcquiringSemaphoreMetric(String
    FunctionCounter.builder( name: "executor.failedAcquiringSema
            .tags(tags.and("name", beanName))
            .description("indicates failure count of acquiring sem
            .register(registry);
}
```

```
@RestControllerEndpoint(id = "vine-executors")
public class BlockingExecutorsEndpoint {
    @PutMapping("/{name}/size")
    public ResponseEntity<Void> adjust(@PathVariab
        registrar.adjustMaxPermits(name, parameter
        return ResponseEntity.noContent().build();
    }
```

3.3 - Backpressure

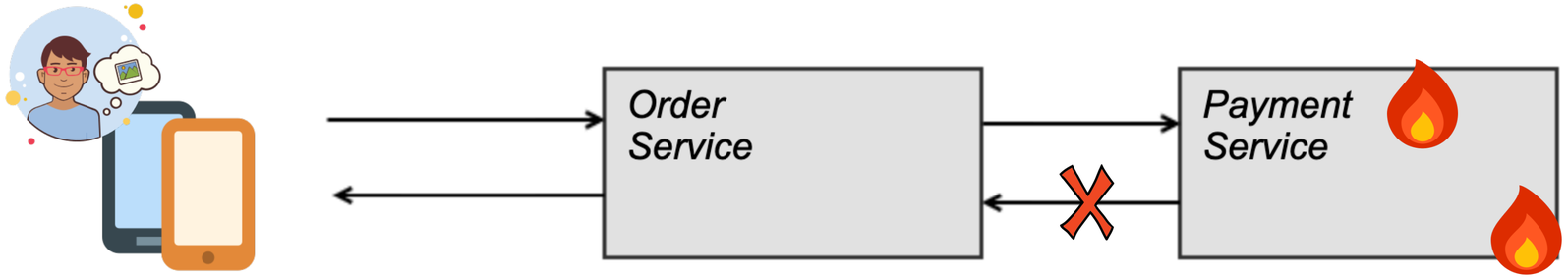




3.4 - 메시지의 유실과 감지

DEVIEW 2019

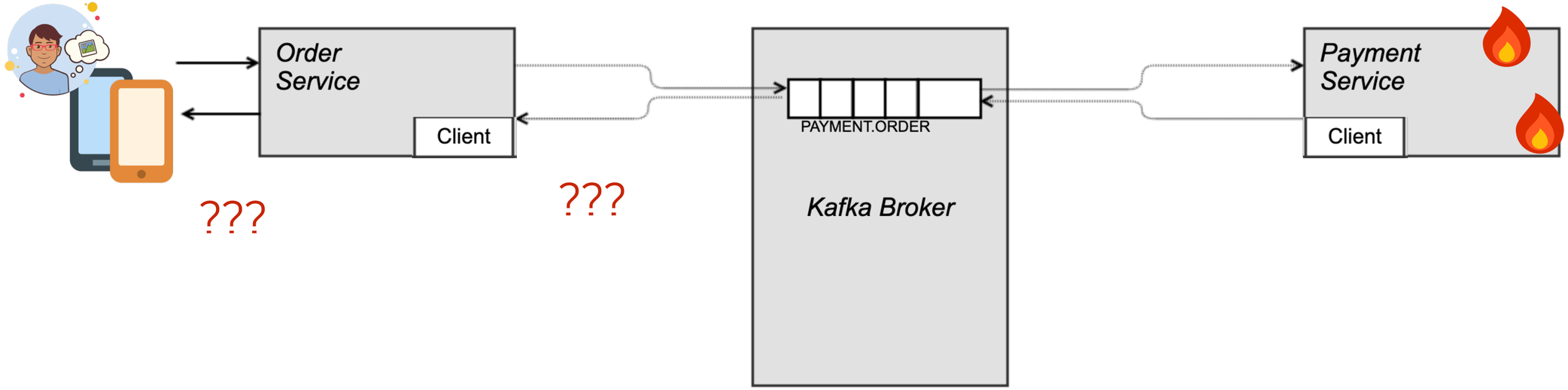
MSA - 타임아웃 이내에 요청의 실패/성공을 알 수 있다



- Read Timeout
- Connection Refused
- Circuit Open

3.4 - 메시지의 유실과 감지

Event Driven - 메시지가 유실 된 것인지, 늦게 처리되는지 알 수가 없다



- 최초의 Request 의 응답은 '카프카에 메시지를 넣었다'에 대한 응답
- 중간에 메시지가 유실 된 것인지, 결제가 한 시간째 진행중인지 알 수 없다

3.4 - 메시지의 유실과 감지

DEVIEW 2019

‘주문번호’를 Key로 하여 5분간 TimeWindow를 생성 요청, 결과의 메시지를 모니터링


```
  "aggs": {
    "enumsum": {
      "sum": {
        "script": {
          "inline": "def path = doc['type'].value;
                     "if (path != null && path.equals(\"APPROVAL_REQUESTED\") == true) {
                     "    return 1;
                     "}
                     "return 10;\n",
          "lang": "painless"
    "min_bucket_selector": {
      "bucket_selector": {
        "buckets_path": {
          "count": "enumsum"
        "script": {
          "source": "params.count <= 2"

"query.bool.filter" : {
    now-10m ~ now-5m && type:'APPROVAL_REQUESTED'
    now-10m ~ now    && type:'APPROVAL_FAILED or COMPLETED'
```

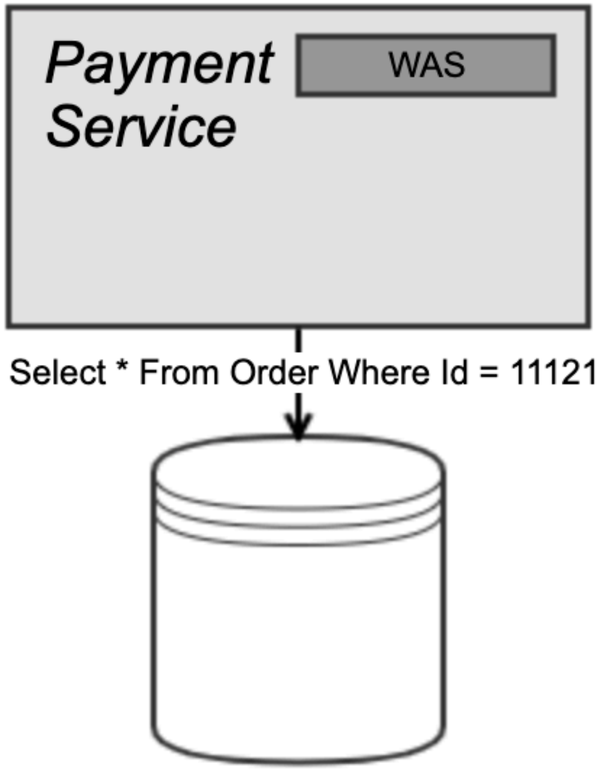
Why Kafka ?

Database 를 Online Transaction 에서 배제하고 싶다 (회원, 제품…)

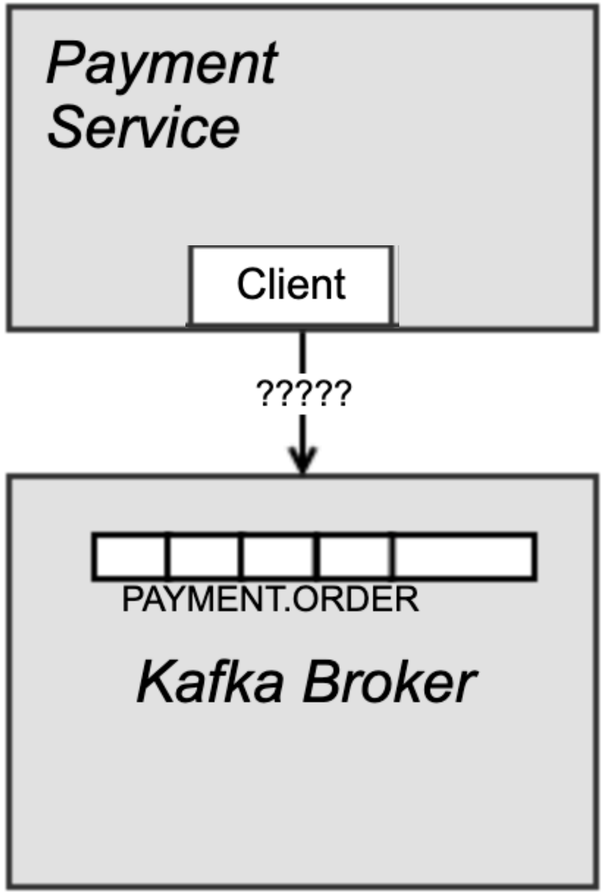
Kafka를 Source of Truth 로 정의하고
Kafka Message를 Materialized View 로 구축하여 Database View 로 활용

3.5 - Materialized View & StateStore

ORACLE



KAFKA



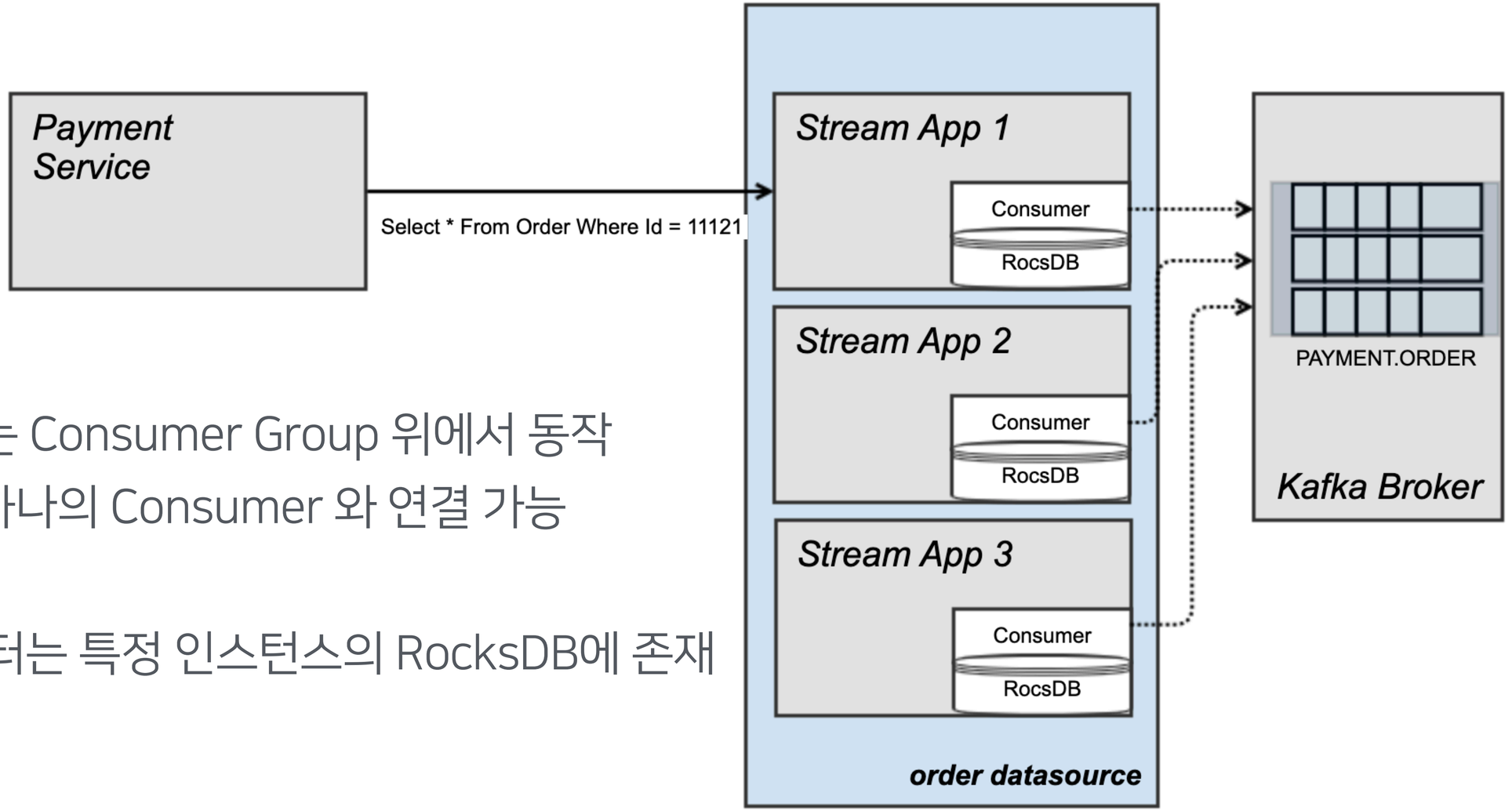
3.5 - Materialized View & StateStore

Interactive Query 를 위한 매개체

- Materialized View
- StateStore : Kafka Stream에서의 M/View 생성방법

3.5 - Materialized View & StateStore

DEVIEW 2019

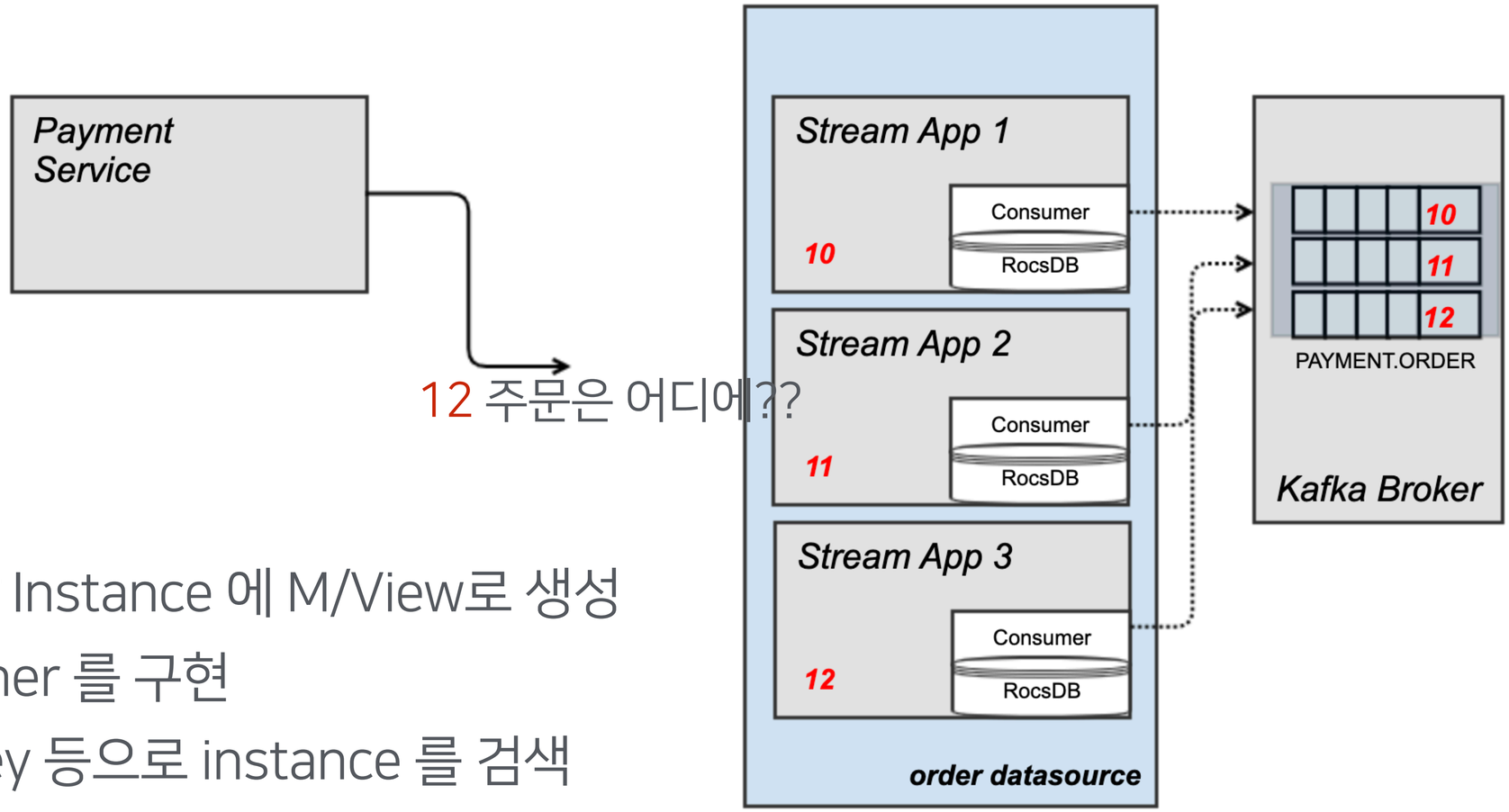


StateStore 는 Consumer Group 위에서 동작

파티션은 단 하나의 Consumer 와 연결 가능

즉, 특정 데이터는 특정 인스턴스의 RocksDB에 존재

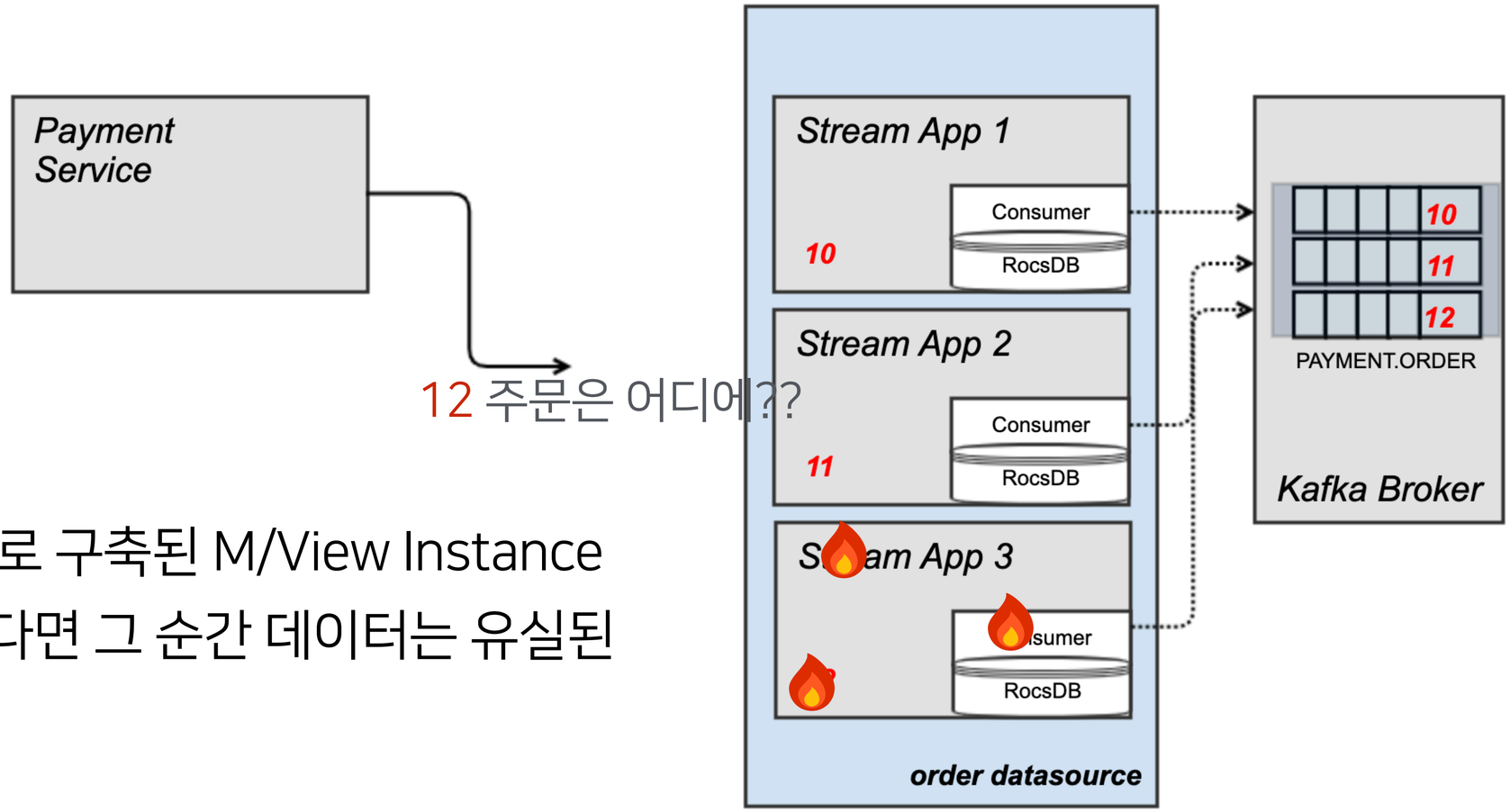
3.6 - StateStore : Query Issue



방법

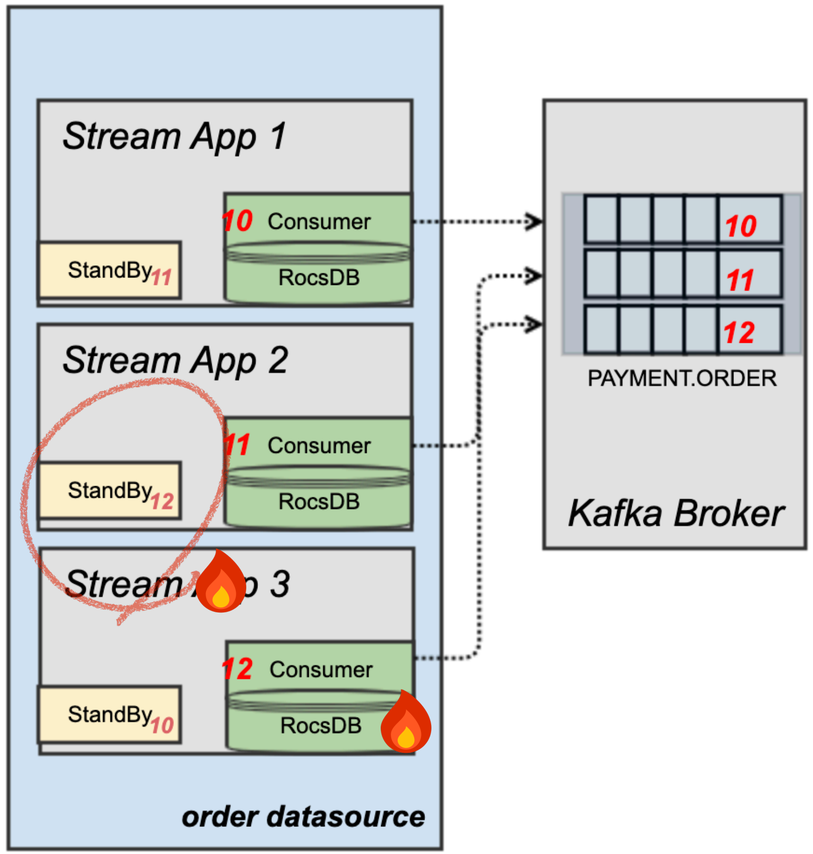
- 모든 데이터를 각 Instance 에 M/View로 생성
- DefaultPartitioner 를 구현
- metadataForKey 등으로 instance 를 검색

3.6 - StateStore : HA Issue

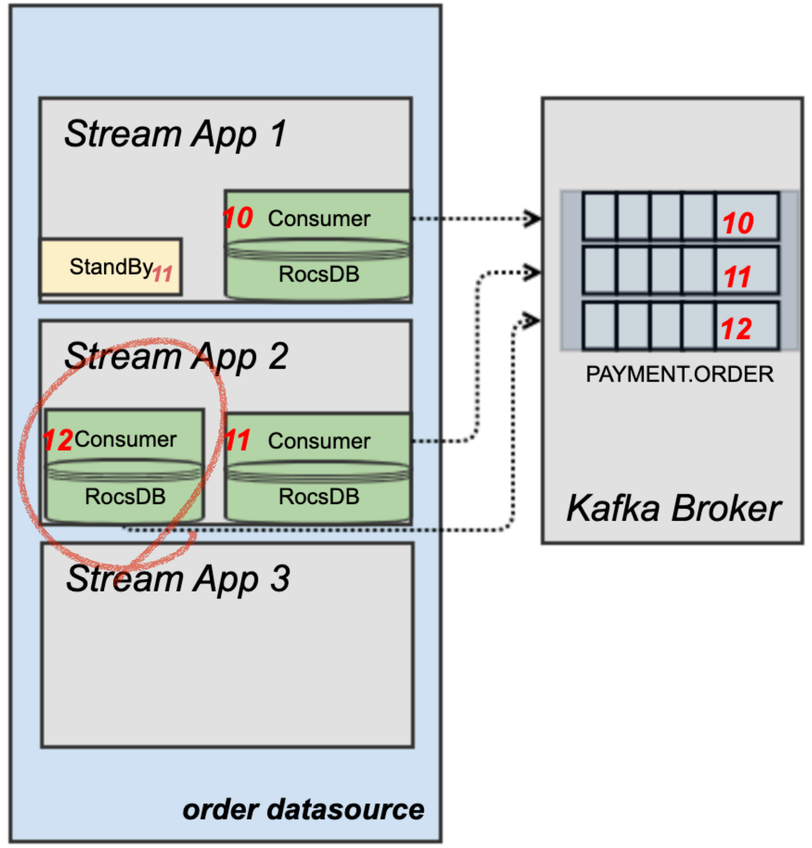


Kafka Stream 으로 구축된 M/View Instance 중 장애가 발생 한다면 그 순간 데이터는 유실된 상황과 동일하다

3.6 - StateStore : HA Issue



StandBy 가 Active 로
변경 되기까지 수 분 동안
데이터의 1/3은 Query 불가



3.7 - StateStore & Streams 활용

DEVIEW 2019

1) Window StateStore 기반의
주문, 가격 이상징후 탐색

```
private WindowBytesStoreSupplier windowBytesStoreSupplier() {
    return Stores.persistentWindowStore(WINDOW_STORE_NAME, R
                                        numSegments: 2, WIND
}

private TimeWindows timeWindow() {
    // 1분 간격으로 단어를 count 한다
    return TimeWindows.of(WINDOW_SIZE);
}
```

2) Streams 를 이용하여
SMS Application 작성

```
@Component
public class NotificationSmsStream extends AbstractNotificationStream {
    @Override
    void addProcessor(StreamsBuilder builder) {
        KStream<String, PostPaymentMessageBin> paymentCompletedStream = builder
                .stream(getStreamProperties().getSourceTopic(),
                        with(messageBinSerde.getKey(), messageBinSerde.getValue()))
                .filter((k, v) -> SEND_SMS_REQUESTED.equals(v.getEventType()));
        paymentCompletedStream
                .map(command)
                .to(getStreamProperties().getSinkTopic(),
                        Produced.with(messageBinSerde.getKey(), messageBinSerde.getValue()));
    }
```

3) 실패한 주문의 재결제를 위한 App

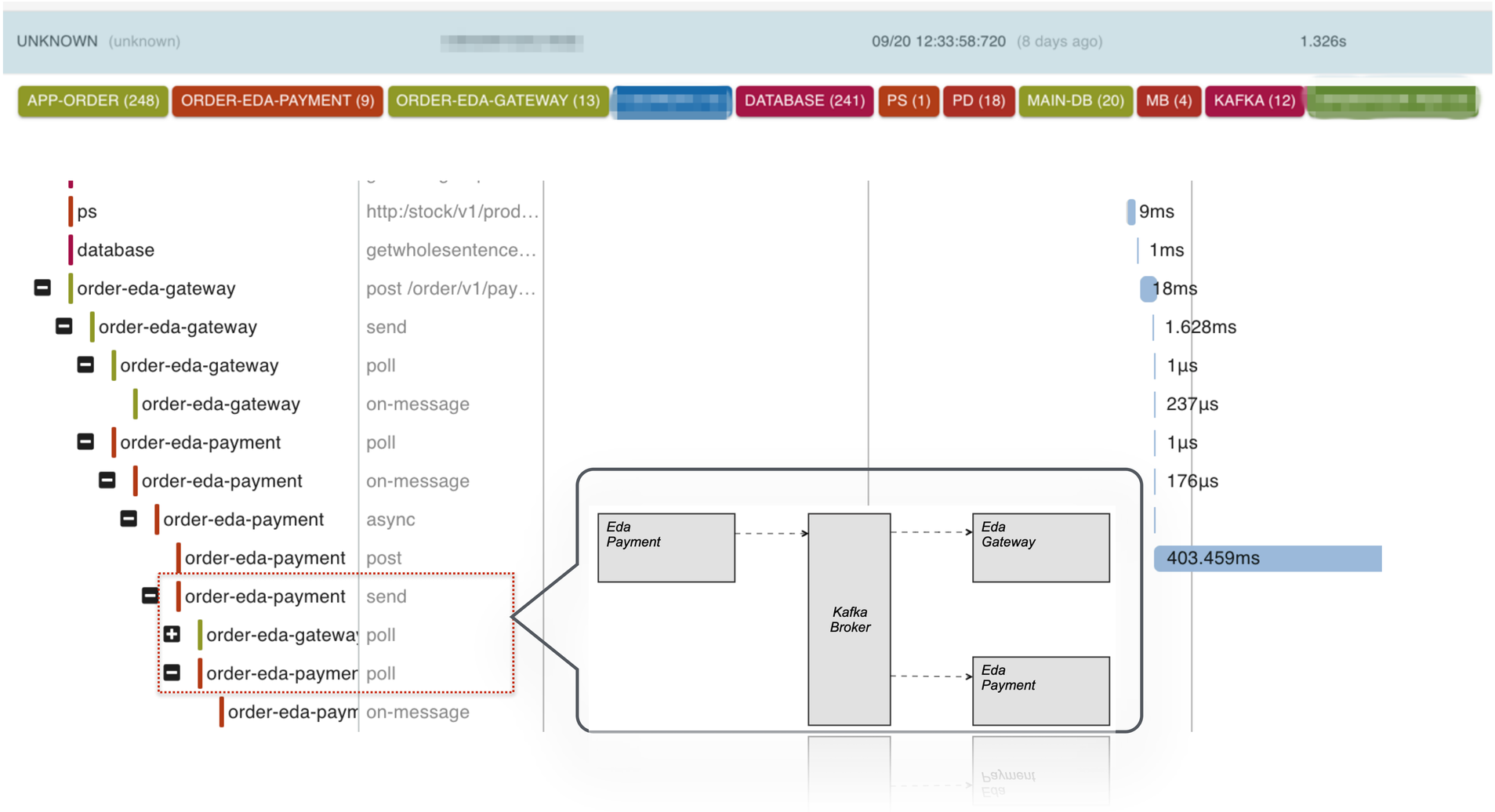
4) 주문 상태 확인을 위한 View

DEVIEW
2019

4.새로운 시스템을 위한 모니터링 개선

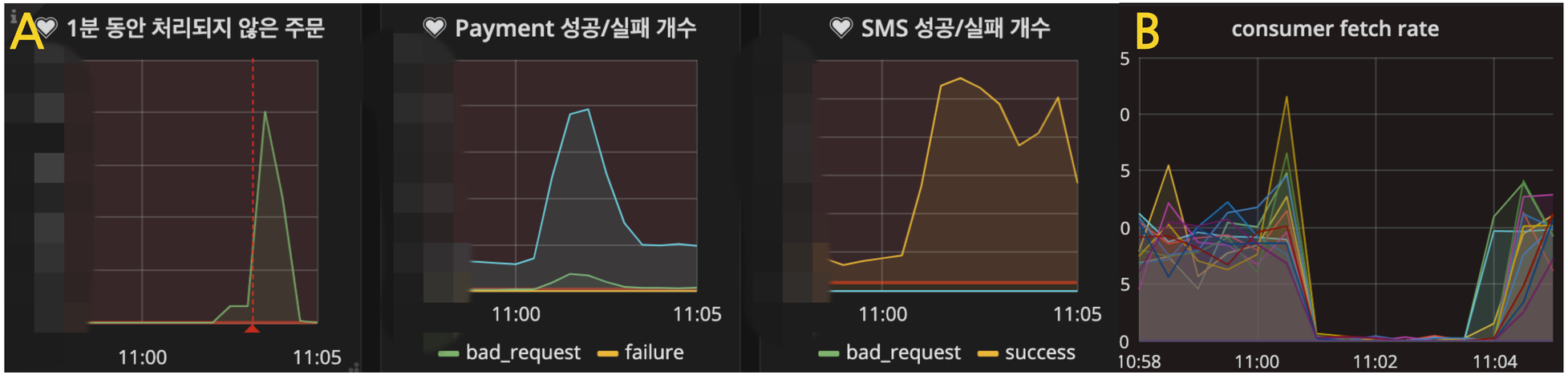
4.1 - Tracing : Zipkin Brave

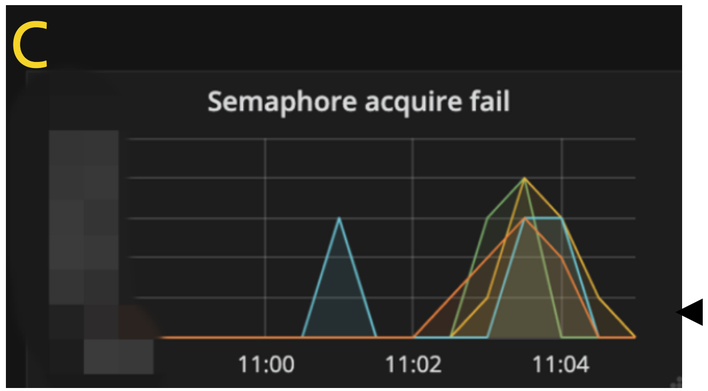
DEVIEW 2019



4.2 - Custom Metric : Prometheus

DEVIEW 2019

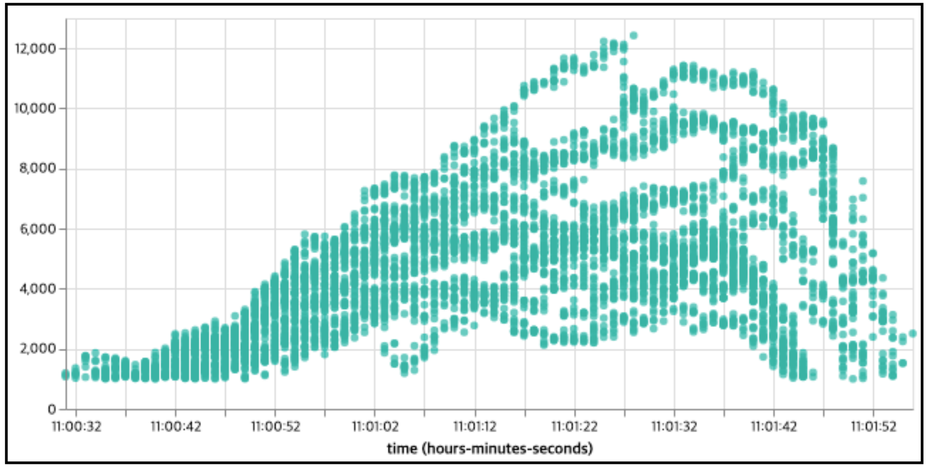


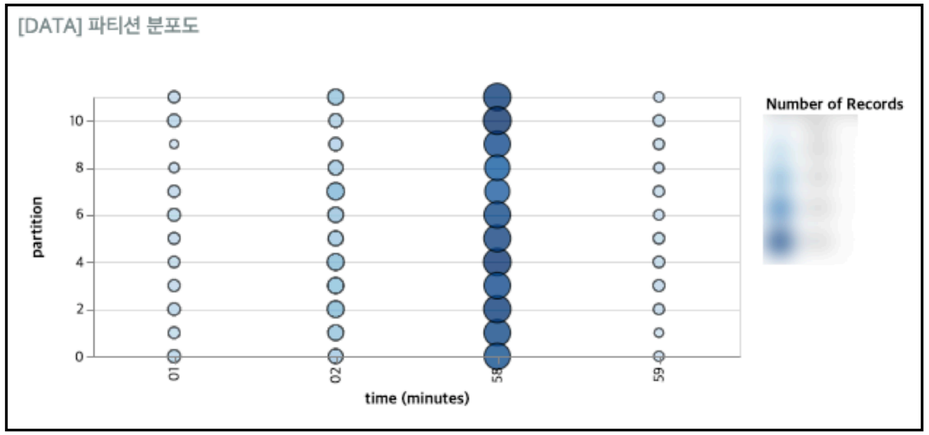



```
//micrometer's metric
private void registerFailedCountAcquiringSemaphoreMetric(String
    FunctionCounter.builder( name: "executor.failedAcquiringSema
            .tags(tags.and("name", beanName))
            .description("indicates failure count of acquiring sem
            .register(registry);
}
```

4.3 - Message : Kibana, Vega

DEVIEW 2019





Topic-Partition 별 Lag

[WARN]Topic : payment.order, Partition : 3, Owner : , lag : 206, Complete : 1
[WARN]Topic : payment.order, Partition : 4, Owner : , lag : 246, Complete : 1
[WARN]Topic : payment.order, Partition : 5, Owner : , lag : 244, Complete : 1
[WARN]Topic : payment.order, Partition : 7, Owner : , lag : 276, Complete : 1
[WARN]Topic : payment.order, Partition : 8, Owner : , lag : 224, Complete : 1

[Alerting] Semaphore Failure alert

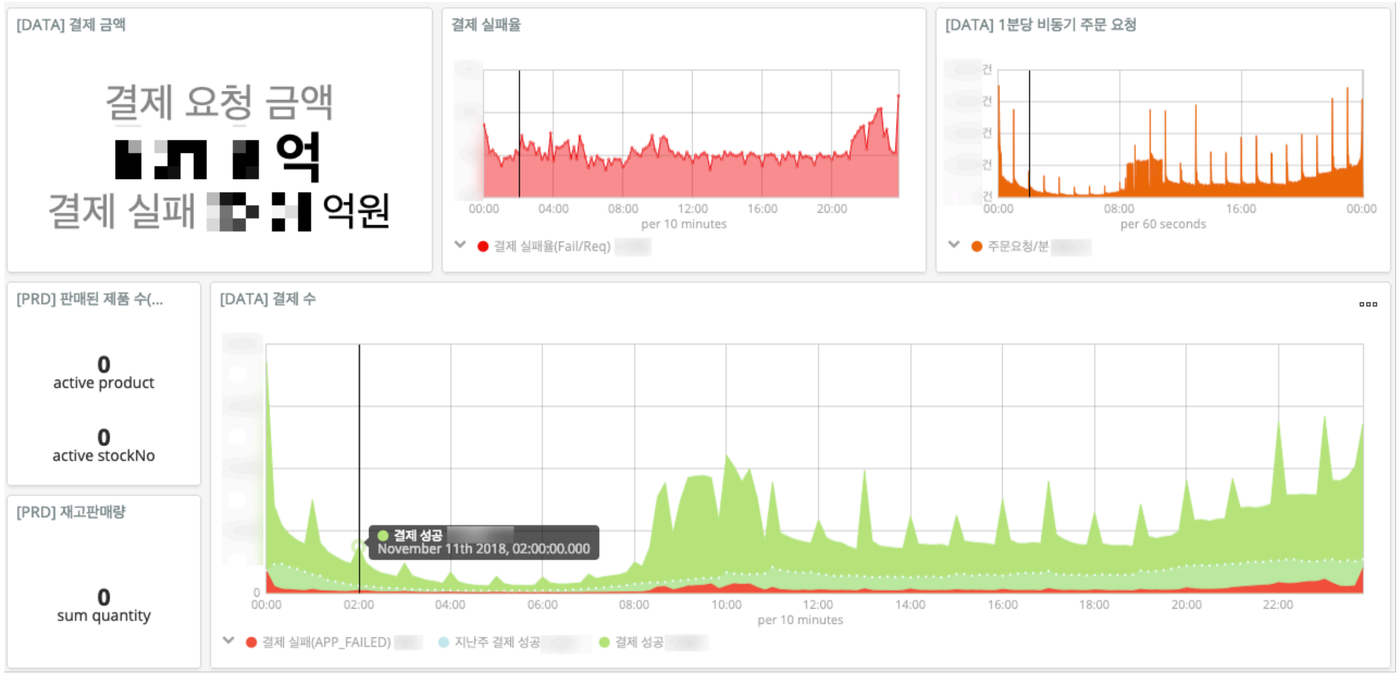
세마포어 획득 실패 증가 감지 (LAG, 메시지 처리 지연 여부 확인)

[Alerting] Consumer Join alert

컨슈머 그룹 리밸런싱 감지됨 (배포 상황인 경우는 무시)

4.4 - Data visualization

DEVIEW 2019



구카드사용불가 (1588-4500)
구카드, 신규수령카드 이용요망
1회 사용한도액 초과 한도초과 탈회/해지 카드
거래정지카드(B/L)
거래정지카드 도난, 분실카드 유효기간경과카드
거래거절 거래정지카드
신규수령카드 사용요망

[ORDER] 제품 판매 순위

제품ID	제품명	가격	주문 횟수	주문 수량	MaxQ
4069	멸균우유 125ml 200ml 24팩 멸균우유	19,100			5
4069	24팩 멸균우유	19,100			4
4365	[타임딜]	11,000			4
7435	미역국	15,300			4

DEVIEW
2019

5. 안정적인 서비스를 위하여

Reactive System

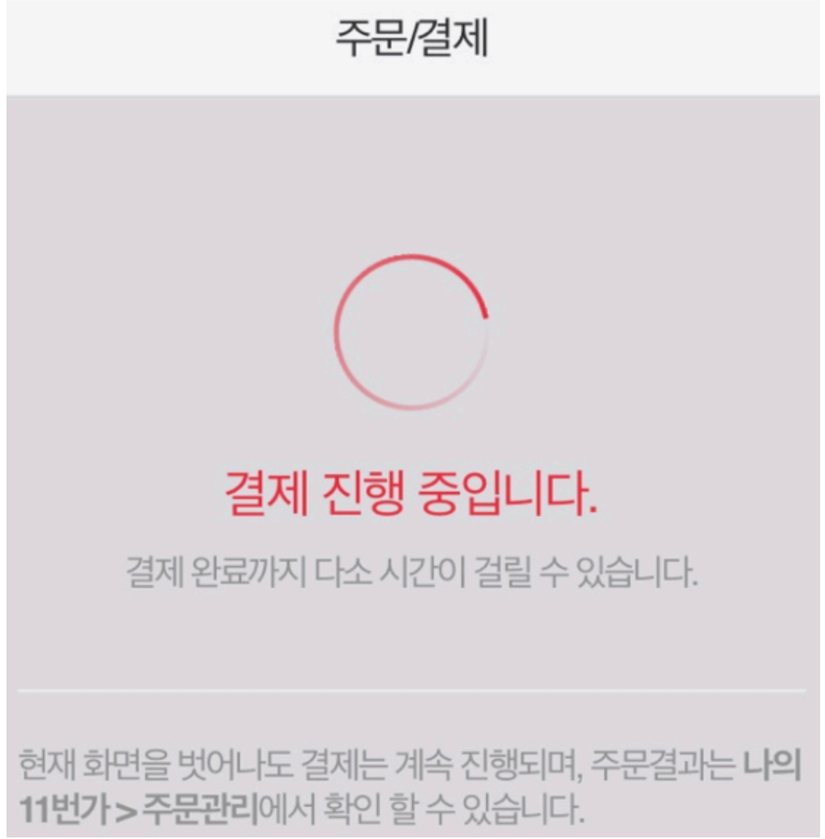
The Reactive Manifesto

Published on September 16 2014. (v2.0)

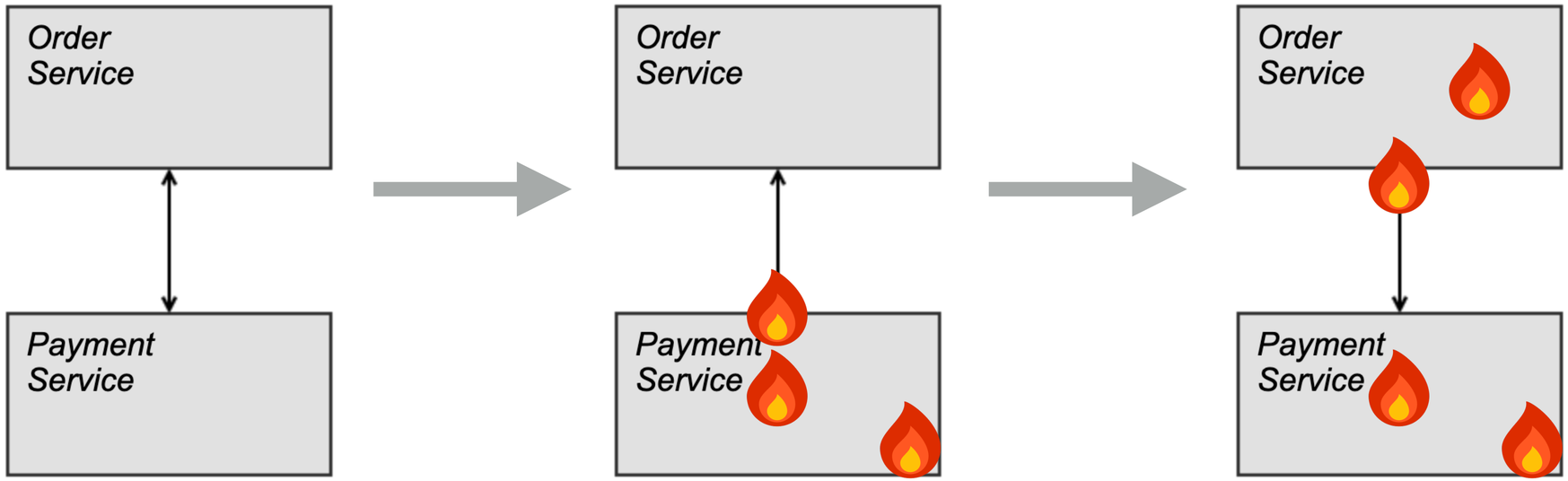
- Message Driven : Rely on asynchronous message-passing
- Resilient : System stays responsive in the face of failure.
- Elastic : System stays responsive under varying workload
- Responsive : System responds in a timely manner if at all possible

5.1 - Responsive (응답성)

DEVIEW
2019



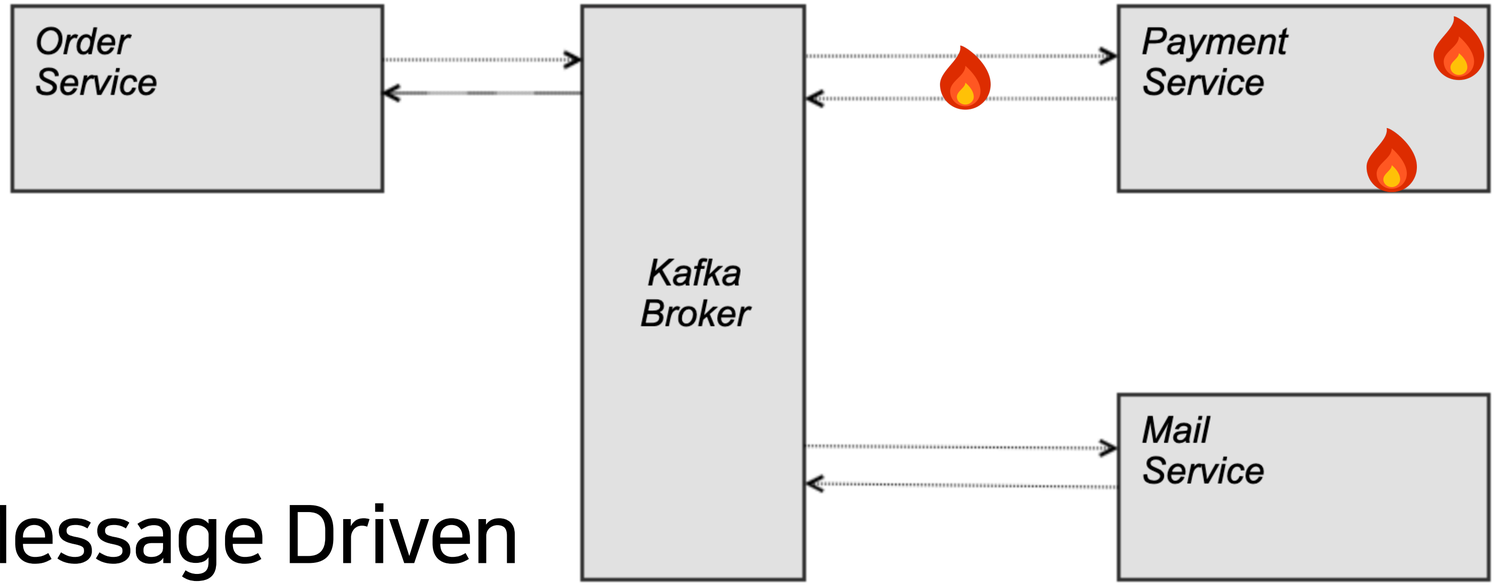
5.2 - Isolation (분리)



MSA

- 물리적 분리의 이점 : 독립 개발 환경, 배포 주기
- 하지만 사용자 관점에서는 하나의 서비스

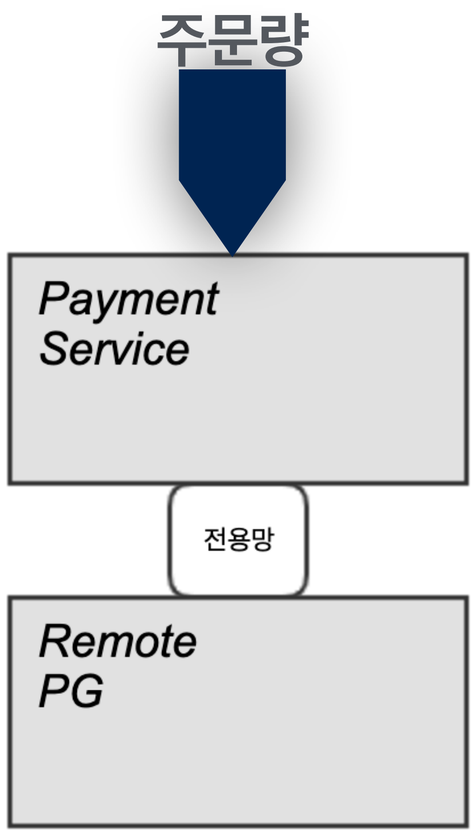
5.2 - Isolation (분리)

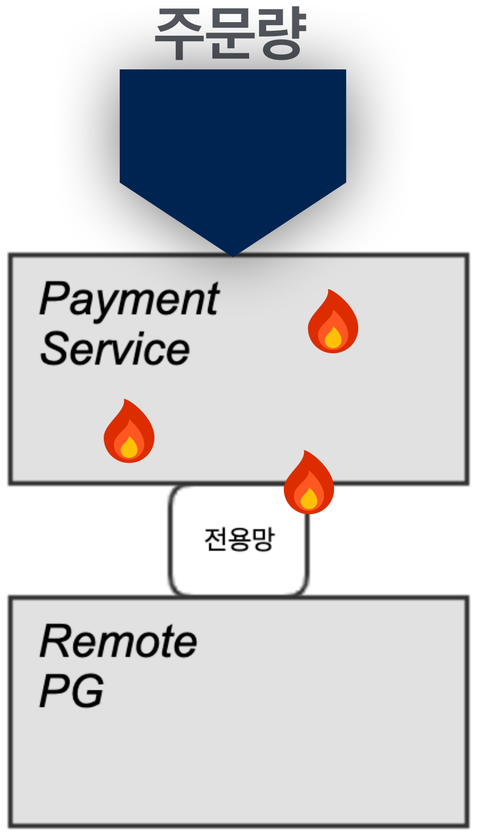


Message Driven

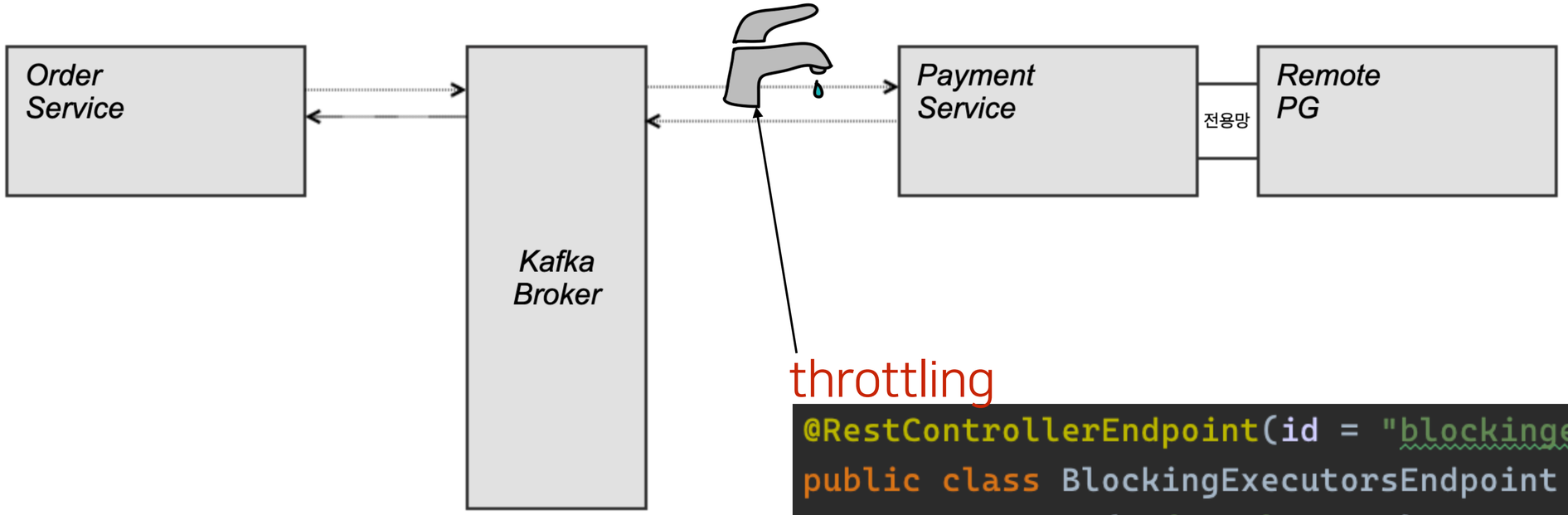
- 물리적 분리의 이점 : 독립 개발 환경, 배포 주기
- 논리적으로 분리
- Resilient 의 기본이 되다

5.3 - Back-pressure (배압)





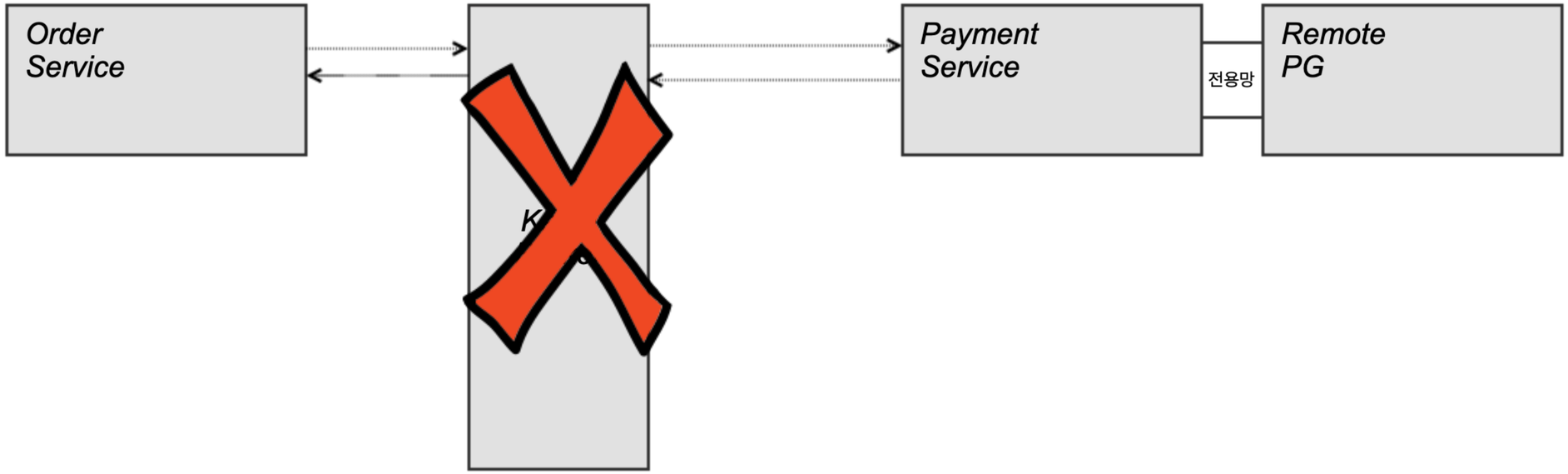
5.3 - Back-pressure (배압)




```
@RestControllerEndpoint(id = "blockingexecutors")
public class BlockingExecutorsEndpoint {
    @PutMapping("/{name}/size")
```

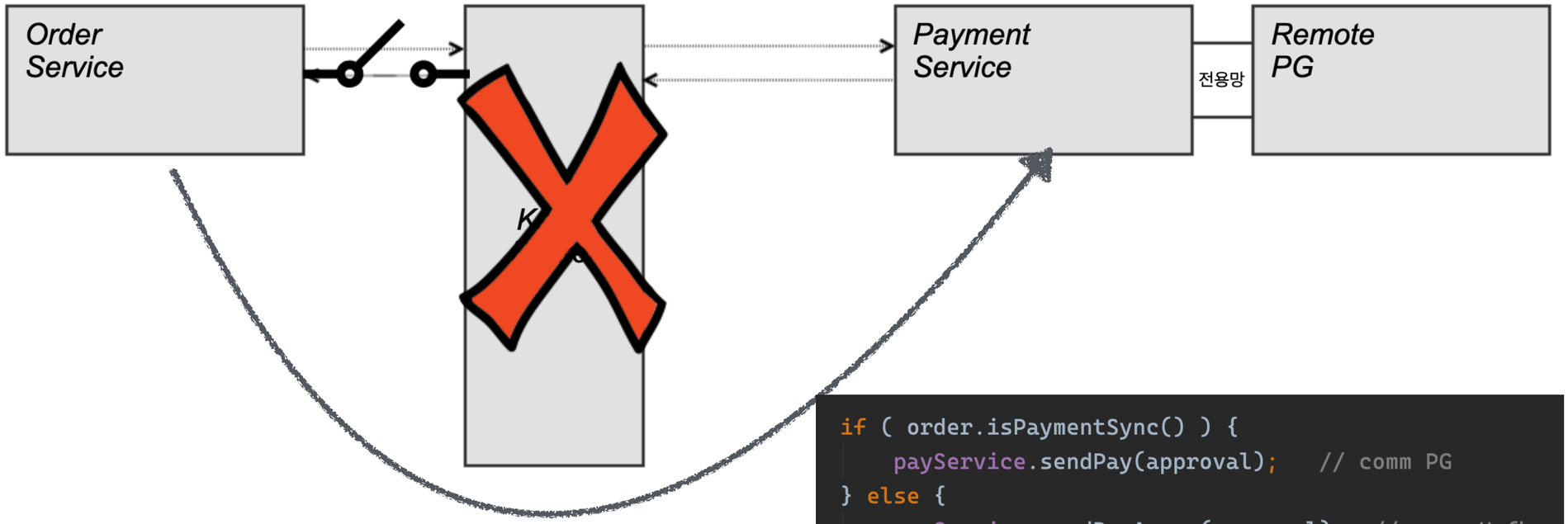
5.4 - Resilient (탄력성) 을 위한 Plan B

DEVIEW
2019



5.4 - Resilient (탄력성) 을 위한 Plan B

DEVIEW 2019




```
if ( order.isPaymentSync() ) {
    payService.sendPay(approval);    // comm PG
} else {
    payService.sendPayAsync(approval);  // comm Kafka
}
```

5.5 - Arts of test code (Human Error)

DEVIEW 2019

1) 속성 변경 체크

```
@Test
@DisplayName("topic name 바뀌면 안됨")
public void should_not_change_topic_name() {
    assertAll(() -> assertEquals( expected: "pos
        () -> assertEquals( expected: "payment.o
        () -> assertEquals( expected: "retry.pay
}
@Configuration
@EnableConfigurationProperties(GatewayProperti
static class Config { }
```

2) OpenSource 버전 체크

```
build.dependsOn {
    assert file("./src/patch/java/org/springframework/cloud/netflix/ribbon/apache
            "$springCloudVersion".contains("Dalston"): "Dalston 전용 Patch이므로 삭제
}
```

5.5 - Arts of test code (Study Test)

DEVIEW 2019

3) 자바 기본의 깊은 이해를 위해서

```java
/**
 * Java 에서 기본 제공하는 Semaphore 의 기능을 검증하는 테스트입니다
 */
public class SemaphoreVerificationTest {
```

4) 오픈 소스의 기능 검증 & 업그레이드 안정성

```java
/**
 * 이 테스트의 목적은 Spring Retry 학습 입니다.
 * Spring Retry에 대한 내용 공식문서 혹은 아래 링크에 있는 Spring Retry.pdf 파일을
 * http://wiki.11stcorp.com/display/DevInno/Dev+Innovation+Seminar+-+
 */
public class RetryTemplateStudyTest {
    @Test
    void 콜백메소드_성공시_재시도_하지않음() {
        int maxAttempts = 3;
        SimpleRetryPolicy policy = new SimpleRetryPolicy(maxAttempts)
        template.setRetryPolicy(policy);
        assertRetryExpectedTimes(template,  expectedRetryCount: 1, contex
    }
    private <T, E extends RuntimeException> T assertRetryExpectedTime
        return this.assertRetryExpectedTimes(template, expectedRetryC
    }
```

5.6 - Chaos engineering

DEVIEW
2019

1. 정상 & 비정상을 확인 할 수 있는 모니터링 구축
2. 예상되는 장애 지점과 시나리오를 작성
3. **실서비스를 대상**
4. 직접 사용하고, 고객의 VoC 까지 확인
5. 회고와 개선
6. Level Up 과 강한 멘탈 획득

5.6 - Chaos engineering

DEVIEW 2019

eda-payment01 강제 종료 이후 partition process pending

- 7시 21분 0초 경 kill -9 으로 해당 서버 종료
  - 중간 내용 생략
- 정리
  - 서버 종료이후 해당 서버가 담당하던 partition (1,5,10)은 약 10초간 데이터 처리를 하지 못함 - 다른 파티션은 계속 동작중이었음
  - session_timeout(디폴트:10초) 에 의해서 eda-payment02에서 파티션 리밸런싱 작업이 일어남 시각 : 07:21:10.188
  - 서버가 담당하고 있던 partition 은 전체적으로 변경됨 테스트 전후 Partition 할당 변화
- 결론
  - EDA-PAYMENT 서버군 중 한대가 예상치 못하게 종료가 된다면 예상한대로 약 10초정도 1/3 정도의 주문 진행이 멈춘다

예상된 장애를 확인하고,
정확한 장애 지속시간 확인

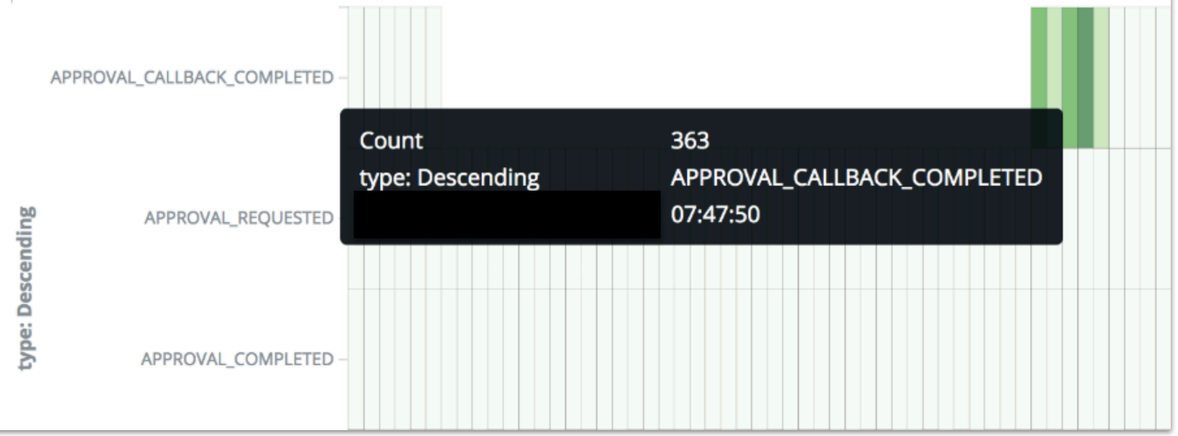
- 서버 1대 장애로 10초간의 장애
  그리고 리밸런스에 의한 2초 장애

5.6 - Chaos engineering

DEVIEW 2019

EDA → Order Callback 을 완전히 중단하는 스위치 On

- 시나리오
  - Database 가 CPU 가 부하가 걸렸을 때, 결제 결과 Insert Query를 잠시 중단
  - 이후, CPU 부하가 내려가는 것을 확인하고 결제 정보를 DB에 저장 하도록 스위치 복구
- 적용 전 후
  - 메시지 분포도



장애시 행동강령 숙지

-해야 할 행동과 결과를 몸이 기억

-사건의 전,후의 Metric 을 체크

5.6 - Chaos engineering

DEVIEW 2019

우병훈(Byunghun Woo)/Platform Engineering팀/11ST

ribbon AvailabilityFilteringRule 이 적용되어 있습니다. 대상 장애 서버에 3번 이상 연속 실패하고 ribbon 리스트에서 임시 제거된다고 했을 때 app-order 인스턴스 * 연속 3번 실패 = xxx번 실패 가능할듯 한데 일단 전체 실패건수는 이론적인 실패횟수 이내이고, 초당 3tps 정도라고 들어서 트래픽이 낮아서 빨리 안빠진것 같습니다. 실제로는 health update 안올라와서 eureka 에서 out_of_service 된 것이 동기화 되는 시간만큼 진행되어 eureka 동기화에 의해 장애 서버가 제거된 것이 아닐까요?

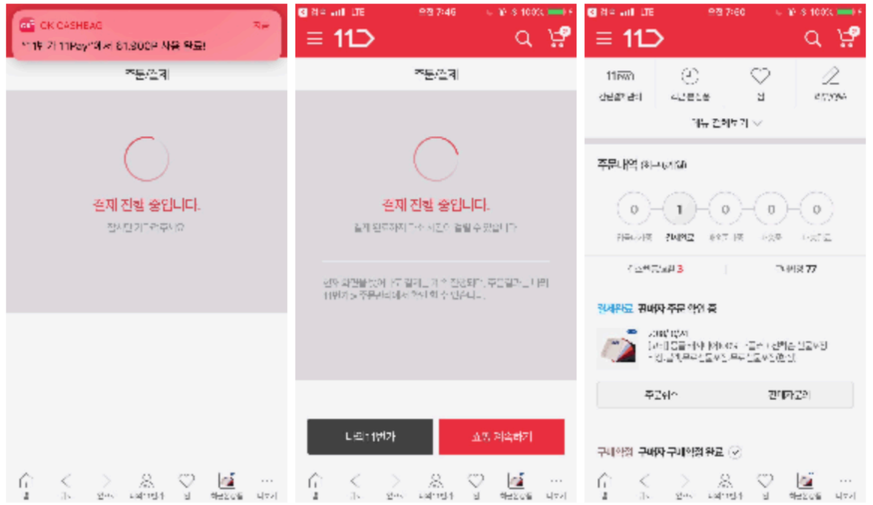
A. 예상치 못한 이슈의 발견

- 주문트래픽이 미미할 때 이슈 확인

B. 직접 사용하고, 장애를 경험하다

-VoC 대응 방안 논의

Callback을 중단했을 경우 아이폰 앱의 화면

- **Callback Pause 인 7시 44분 LTE - iOS 앱으로 주문**
- **11PAY 3천원 쿠폰 + 전액 OCB로 결제 요청**
- 

- **그림1) 7:44분 결제 요청**
- **그림2) 7:45분 결제 요청 후 1분 대기 중, 이부 앱을 종료함**
- **그림3) 7:47분 Callback Pause 해제, 이후 7:50분 앱을 들어가서 주문 상태를 본 결과**
- **특이사항)**

5.7 - 상용 값과 메시지의 흐름

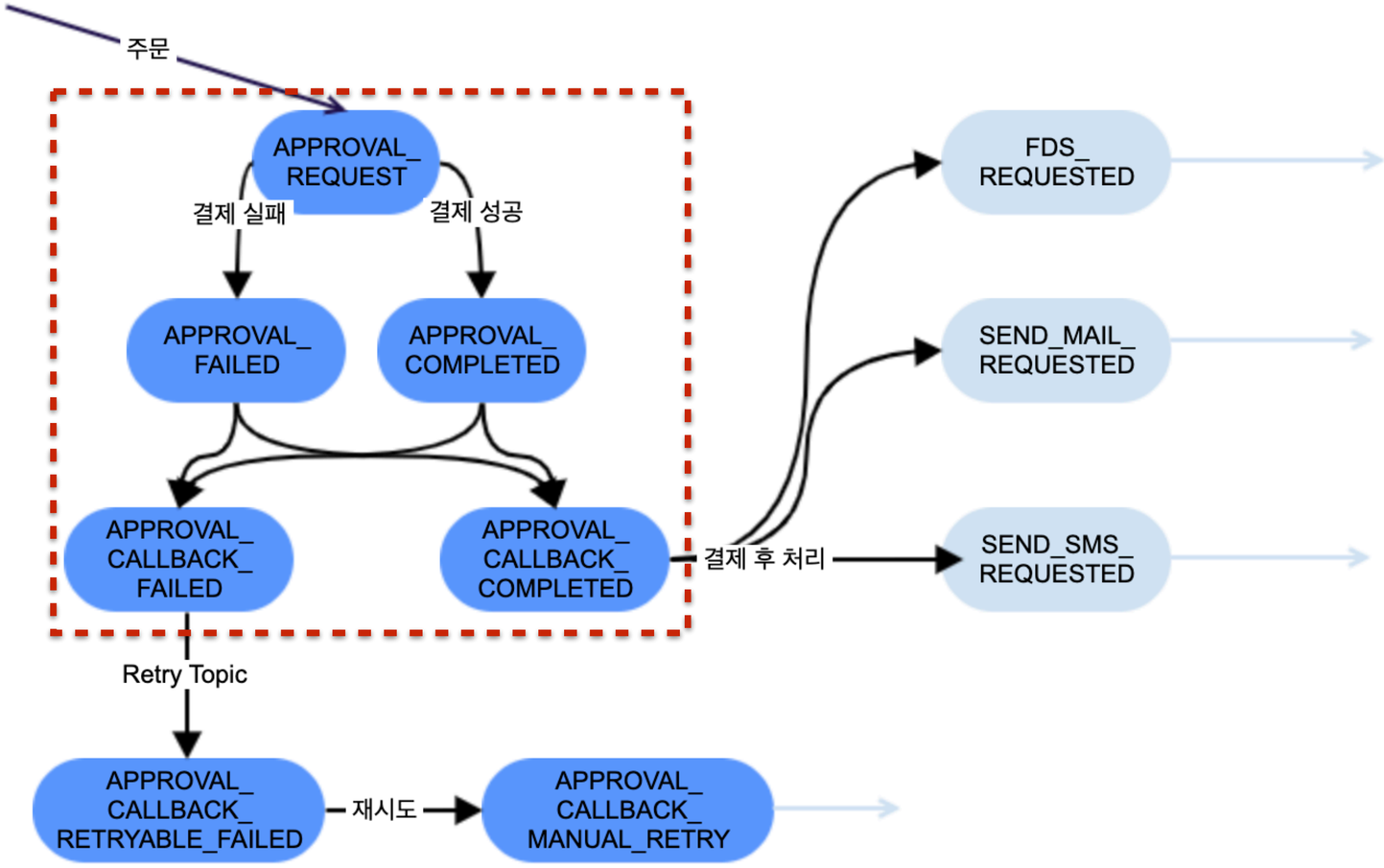
DEVIEW 2019

PAYMENT.ORDER

- Partition:12, Replica:3, ISR:2
- ACKS : ALL

Application Configuration

- Confluent's KafkaAvroSerializer
- Confluent's SchemaRegistry
- DefaultPartitioner
- a/commit : false (mode - record)
- max-poll-records: 1



DEVIEW
2019

Q & A

DEVIEW
2019

Thank You