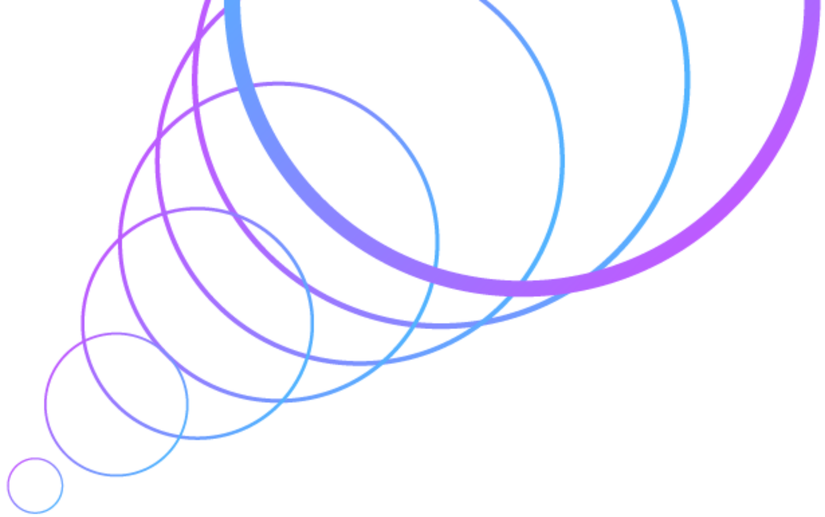


DEVIEW 2021

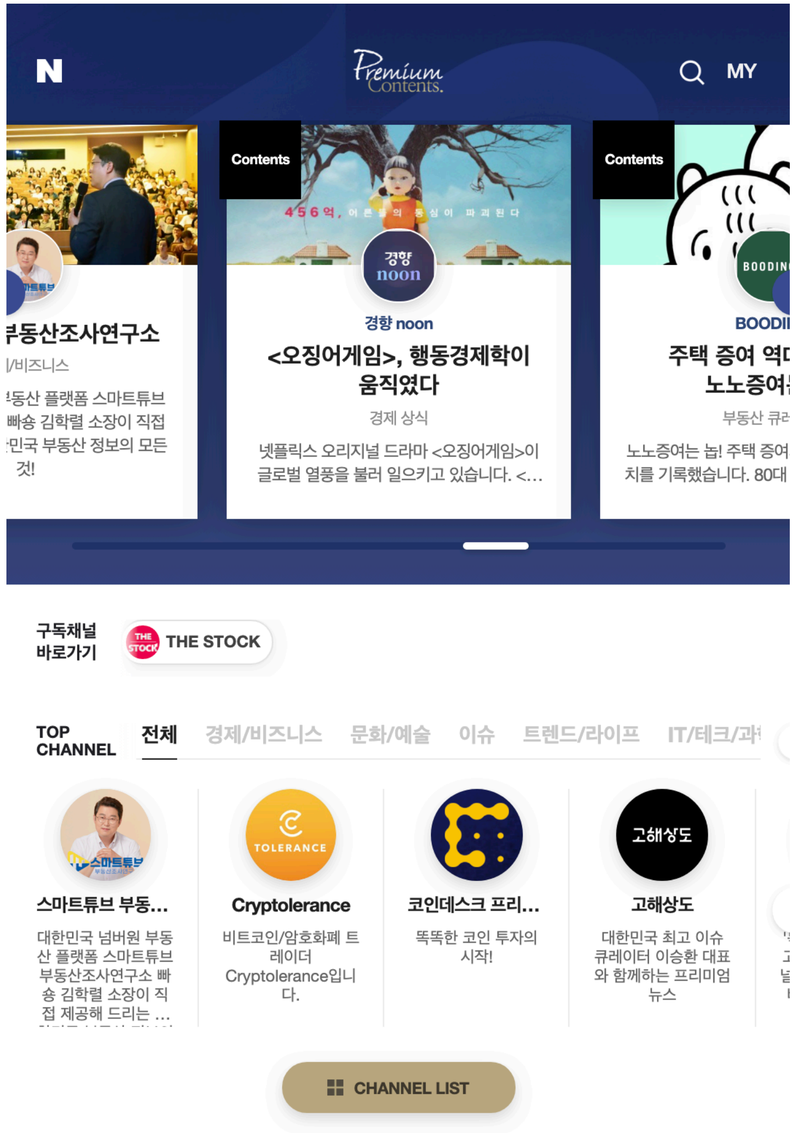
썰 푼다! 주니어 개발자의 프리미엄 콘텐츠 Hub 서버 개발기

이상민 NAVER

프리미엄 콘텐츠 - Preview

창작자를 위한 유료 구독 플랫폼 `네이버 프리미엄 콘텐츠`

- 콘텐츠 향 스마트스토어
- 콘텐츠 생산에서 발행, 판매, 통계, 정산 등 콘텐츠 판매에 필요한 기능 제공



프리미엄 콘텐츠 - Preview

DEVIEW 2021

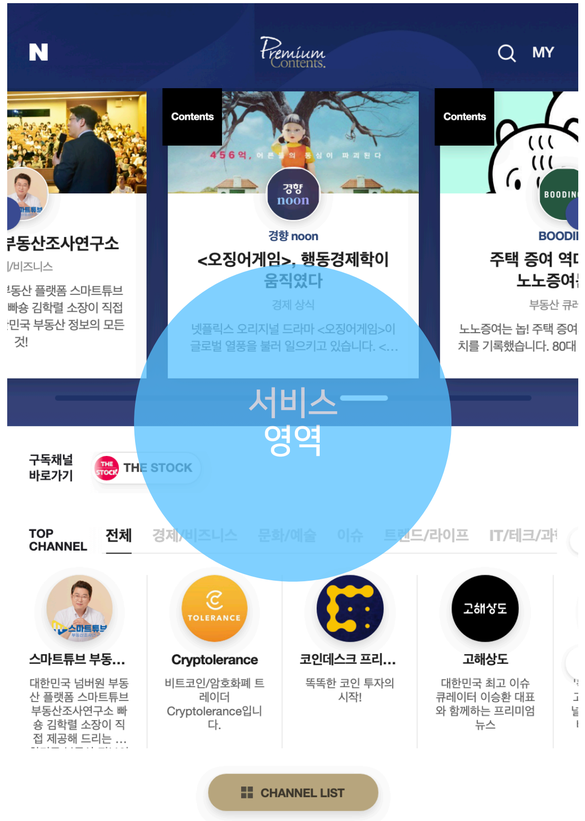
각 영역별 독립적 구성

스튜디오 영역

유료화/상품 영역

콘텐츠 Hub 영역

채널/가입자 영역



서비스 영역

Smart Editor

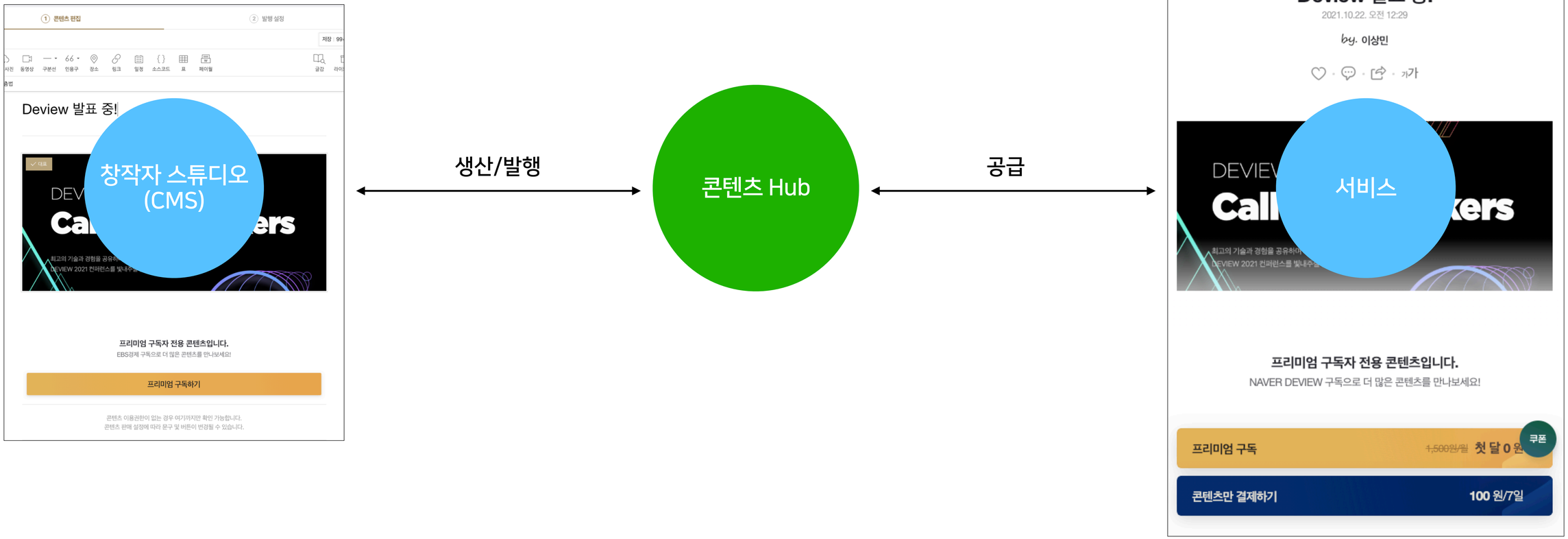
CONTENTS

1.프리미엄 콘텐츠 Hub

1.1 프리미엄 콘텐츠 Hub

프리미엄 콘텐츠 Hub

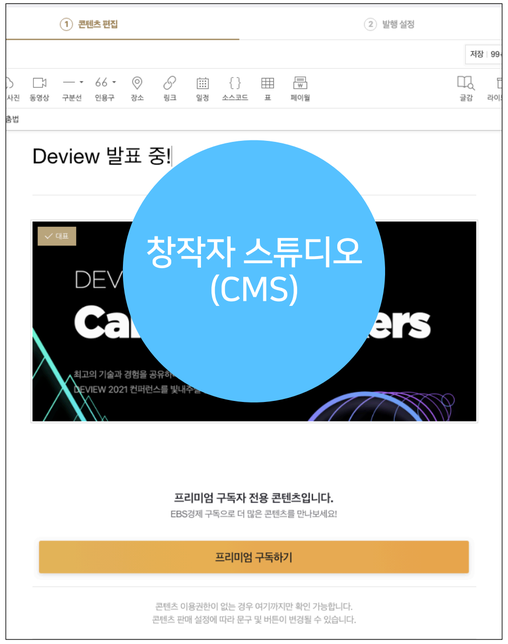
- 유료 콘텐츠 생산/발행/공급 플랫폼



1.1 프리미엄 콘텐츠 Hub

프리미엄 콘텐츠 Hub

- 유료 콘텐츠 생산/발행/공급 플랫폼

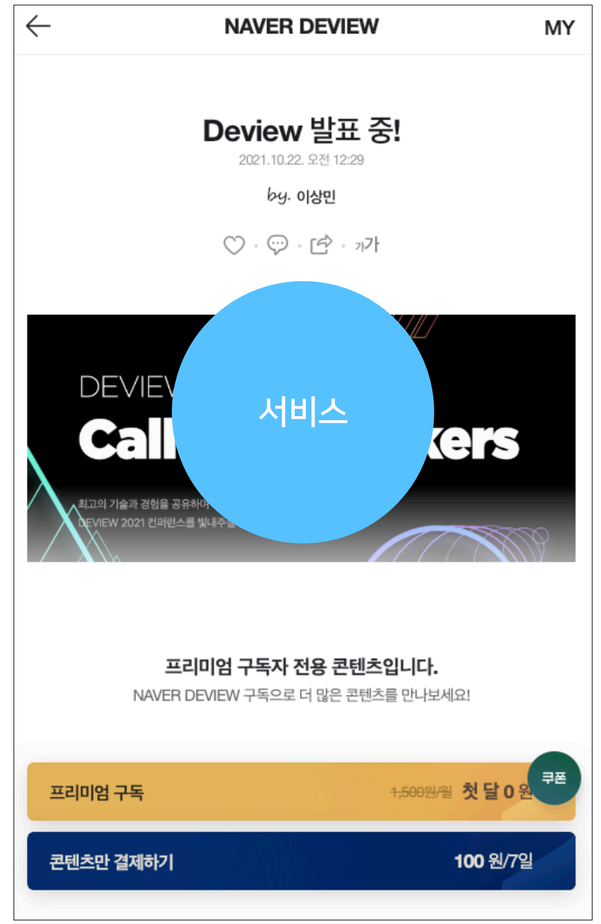


창작자 스튜디오 (CMS)

생산/발행

콘텐츠 Hub

공급



서비스

연동

Smart Editor

오디오 플랫폼

동영상 플랫폼

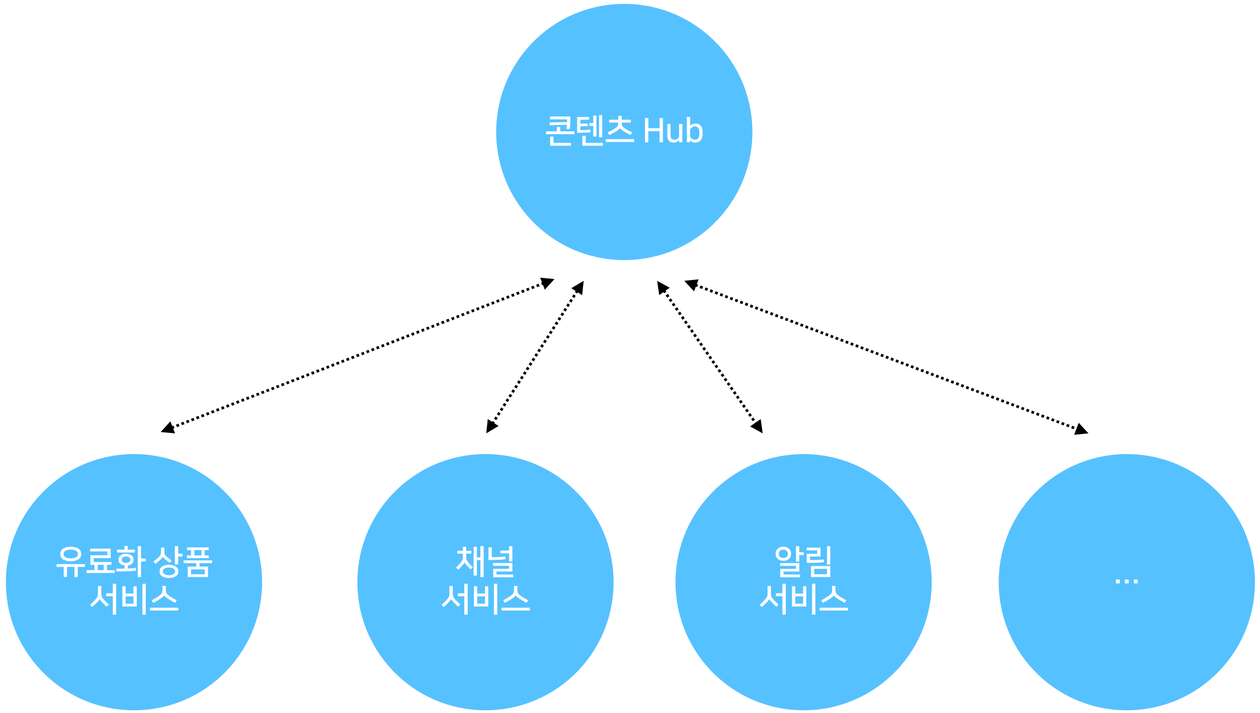
연관 플랫폼
...

1.1 프리미엄 콘텐츠 Hub

DEVIEW 2021

Microservice Architecture

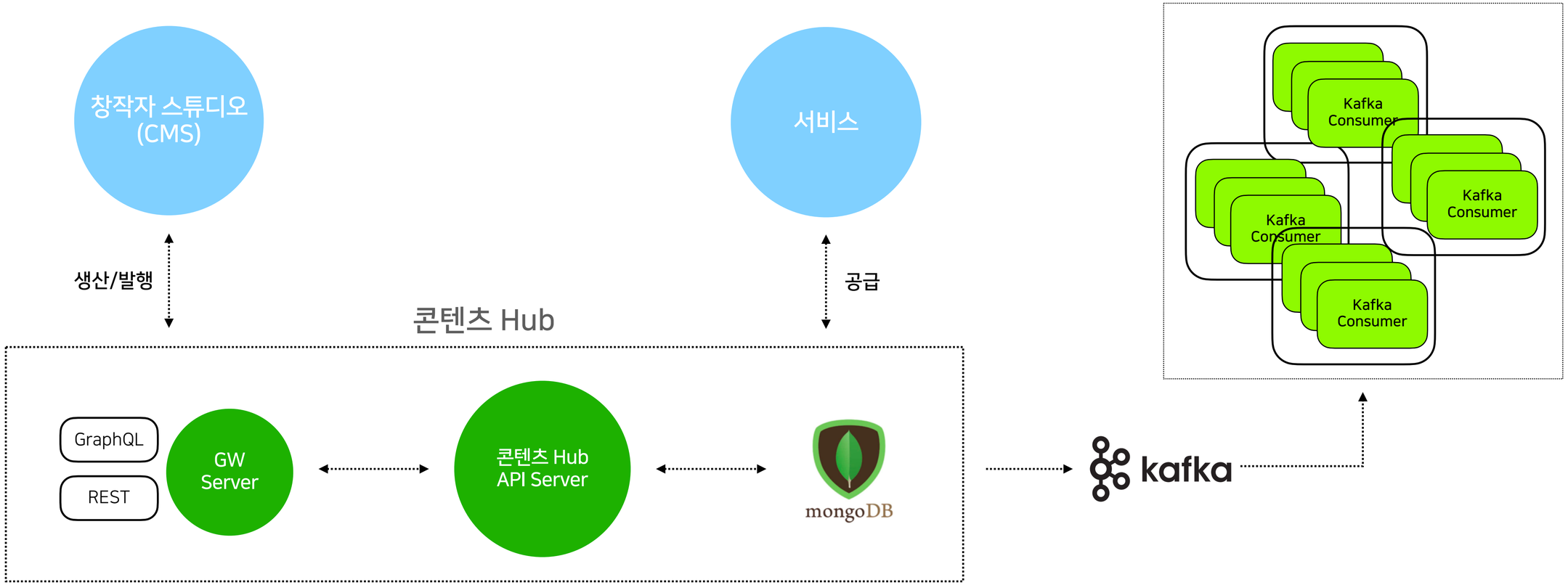
- 채널 Service
- 유료화(상품) Service
- 알림 Service
- ...



1.2 프리미엄 콘텐츠 Hub 구성

DEVIEW 2021

구성



DEVIEW 2021

2.Reactor로 보는 콘텐츠 Hub.SSUL

2.1 API 서버 기술 스택

DEVIEW 2021

프리미엄 콘텐츠 Hub

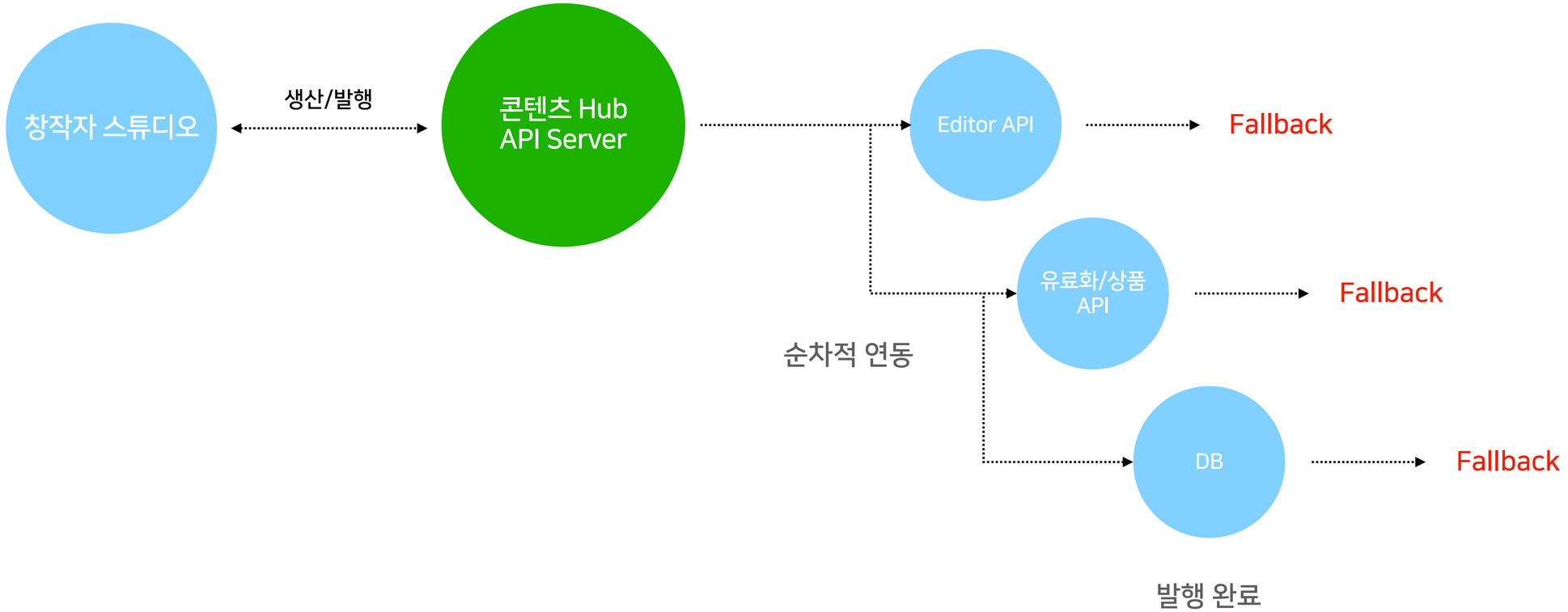
- Spring Boot 2.3.3/JDK11
- Spring Webflux/Reactor

2.2 Fallback Publisher

DEVIEW 2021

프리미엄 콘텐츠 Hub

- 타 서비스/플랫폼과 순차적 연동 빈도가 높음
- Step 별 에러 및 후처리 격리가 필요

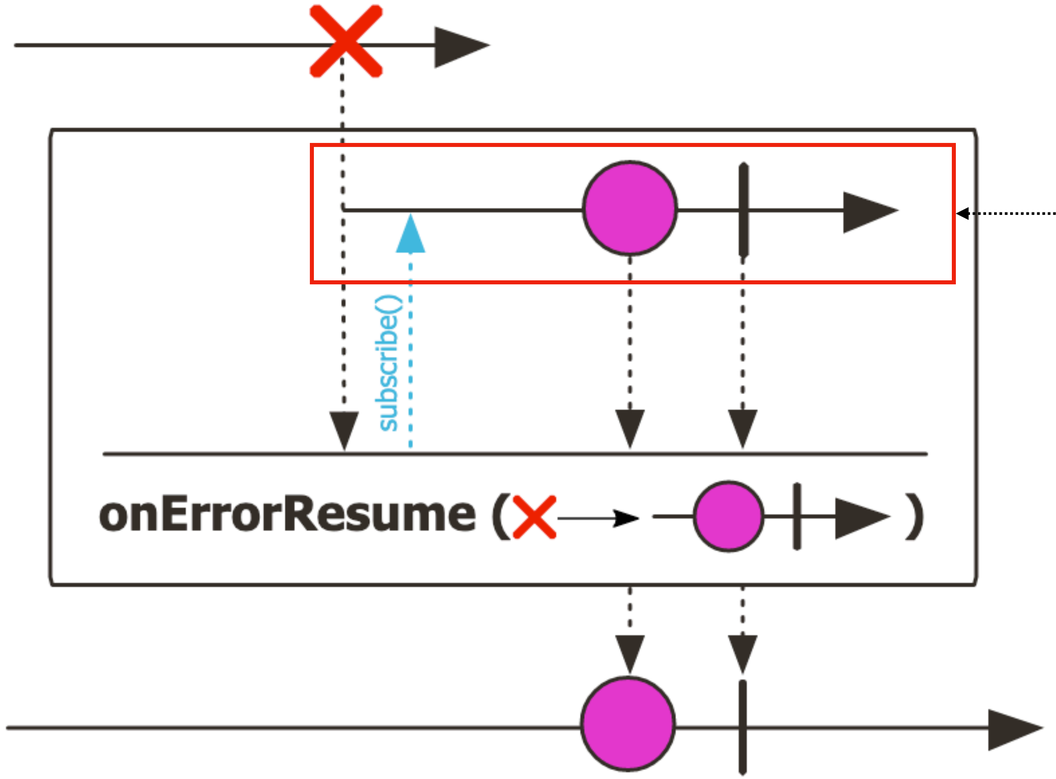


2.2 Fallback Publisher

DEVIEW 2021

onErrorResume

```
onErrorResume(Function<? super Throwable,? extends Mono<? extends T>> fallback)
```
Subscribe to a fallback publisher when any error occurs, using a function to choose the fallback depending on the error.



Fallback Publisher

에러 발생시 publisher 대체

projectreactor.io

2.2 Fallback Publisher

DEVIEW 2021

단계별 Fallback 처리 - onErrorResume

```
@Override
public Mono<Content> postProcess(ContentProductInfo contentProductInfo) {
    Content content = contentProductInfo.getContent();
    ProductInfo productInfo = contentProductInfo.getProductInfo();
    Requester requester = contentProductInfo.getRequester();
    String contentId = content.get_id();

    return productProcessor.registerProduct(content, productInfo, requester)
            .onErrorResume(t -> onProductRegisterFailed(content) Step 1
                    .then(Mono.error(new ProductRegisterException(contentId, t)))
            )
            .map(content::withSalesDatetime)
            .flatMap(c -> contentsRepository.insert(c)
                    .onErrorResume(t -> onInsertContentFailed(content) Step 2
                            .then(Mono.error(new InsertContentException(contentId, t)))
                    );
}
```

```
private Mono<Content> onProductRegisterFailed(Content content) {
    content = content.toBuilder()
            .tempDatetime(LocalDateTime.now())
            .publishDatetime(null)
            .registerDatetime(null)
            .modifyDatetime(null)
            .build();

    return tempContentsRepository.insert(content)
            .doOnSuccess(c -> log.warn("[onProductRegisterFailed-{}] 
}
```

```
private Mono<ProductResponse> onInsertContentFailed(Content content) {
    String contentId = content.get_id();
    CpType cpType = content.getCpType();
    String cpName = content.getCpName();
    String subId = content.getSubId();

    return productProcessor.removeRollback(cpType, cpName, subId, conter
            .doOnSuccess(c -> log.warn("[onInsertContentFailed] {} 컨텐츠
}
```

1. 상품 등록 (타 서비스 연동)
   - Step 1 - Fallback: 임시저장 콘텐츠로 전환
2. DB 저장
   - Step 2 - Fallback: 상품 롤백

Step 별 Fallback 격리 실행

2.2 Fallback Publisher

단계별 Fallback 처리 네이밍 - on{실패한 행위}

```java
@Override
public Mono<Content> postProcess(ContentProductInfo contentProductInfo) {
    Content content = contentProductInfo.getContent();
    ProductInfo productInfo = contentProductInfo.getProductInfo();
    Requester requester = contentProductInfo.getRequester();
    String contentId = content.get_id();

    return productProcessor.registerProduct(content, productInfo, requester)
            .onErrorResume(t -> onProductRegisterFailed(content) Step 1
                    .then(Mono.error(new ProductRegisterException(contentId, t)))
            )
            .map(content::withSalesDatetime)
            .flatMap(c -> contentsRepository.insert(c)
                    .onErrorResume(t -> onInsertContentFailed(content) Step 2
                            .then(Mono.error(new InsertContentException(contentId, t)))
                    );
}
```

```java
private Mono<Content> onProductRegisterFailed(Content content) {
    content = content.toBuilder()
            .tempDatetime(LocalDateTime.now())
            .publishDatetime(null)
            .registerDatetime(null)
            .modifyDatetime(null)
            .build();

    return tempContentsRepository.insert(content)
            .doOnSuccess(c -> log.warn("[onProductRegisterFailed-{}] 
}
```

```java
private Mono<ProductResponse> onInsertContentFailed(Content content) {
    String contentId = content.get_id();
    CpType cpType = content.getCpType();
    String cpName = content.getCpName();
    String subId = content.getSubId();

    return productProcessor.removeRollback(cpType, cpName, subId, conter
            .doOnSuccess(c -> log.warn("[onInsertContentFailed] {} 컨텐츠
}
```

실패한 행위에 중점을 둘 것인가?

실행할 행위에 중점을 둘 것인가?

실패한 행위

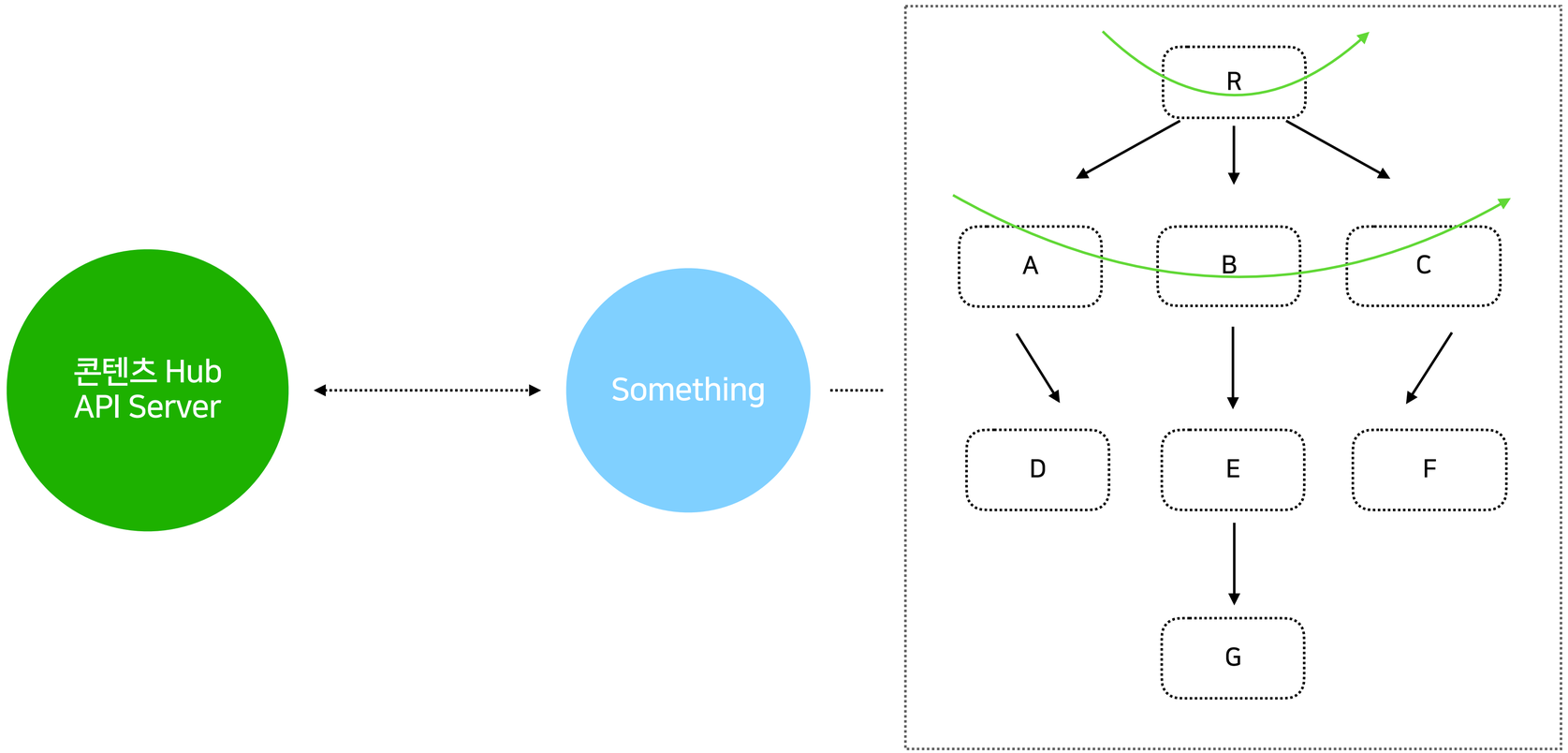
네이밍 -> on{실패한 행위}

- 필요한 경우 동일한 니즈에 대해 재사용 가능
- 코드가 곧 매뉴얼 -> flow 파악 용이

2.3 그래프 순회

프리미엄 콘텐츠 Hub

- 그래프를 생성하고 순회가 필요한 케이스



2.3 그래프 순회

DEVIEW 2021

Flux/Mono.expand

- Reculsively, Breadth-first 순회

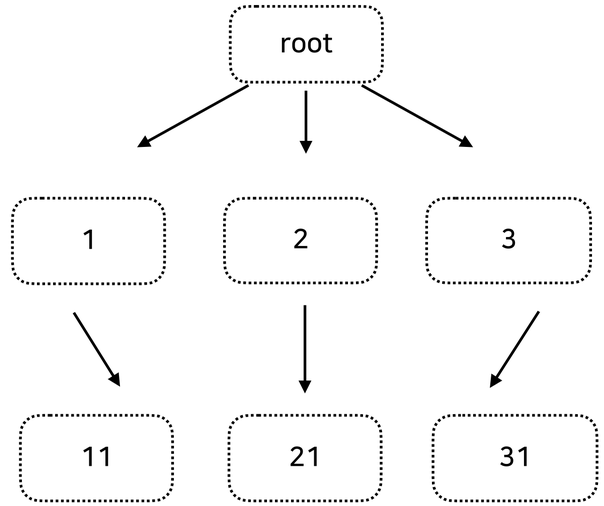
```
expand(Function<? super T,? extends Publisher<? extends T>> expander)
```

Recursively expand elements into a graph and emit all the resulting element using a breadth-first traversal strategy.

```
Node sampleNodes() {
    return new Node( name: "root",
                new Node( name: "1",
                        new Node( name: "11")),
                new Node( name: "2",
                        new Node( name: "21")),
                new Node( name: "3",
                        new Node( name: "31"))
    );
}

static class Node {
    final String name;
    final List<Node> children;

    Node(String name, Node... nodes) {
        this.name = name;
        this.children = new ArrayList<>();
        children.addAll(Arrays.asList(nodes));
    }
}
```



projectreactor.io

2.3 그래프 순회

DEVIEW 2021

Flux/Mono.expand

- Reculsively, Breadth-first 순회

```
expand(Function<? super T,? extends Publisher<? extends T>> expander)
```
Recursively expand elements into a graph and emit all the resulting element using a breadth-first traversal strategy.

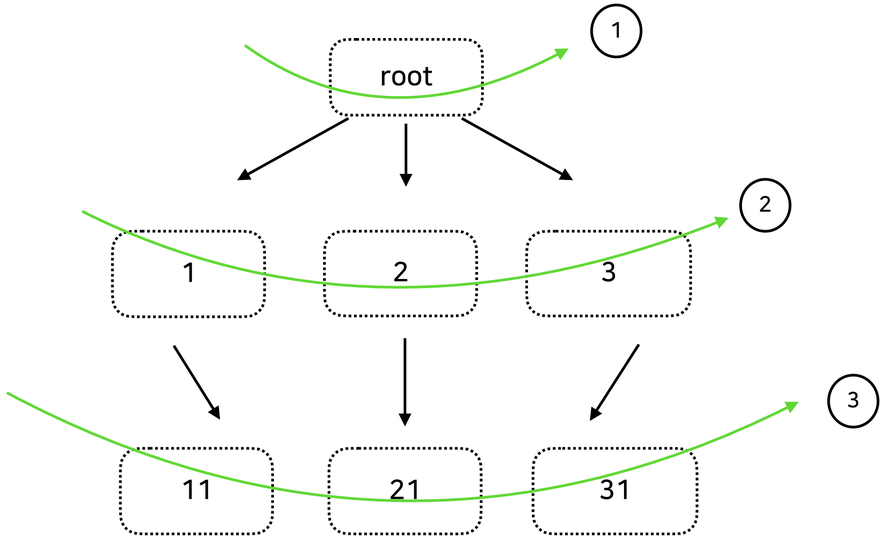
```
@Test
public void expandTest() {
    Node root = sampleNodes();
    Flux<String> expandFlux = Flux.just(root)
            .expand(v -> Flux.fromIterable(v.children))
            .map(v -> v.name);

    StepVerifier.create(expandFlux) StepVerifier.FirstStep<String>
            .expectNext("root") StepVerifier.Step<String>
            .expectNext("1", "2", "3")
            .expectNext("11", "21", "31")
            .verifyComplete();
}

Node sampleNodes() {
    return new Node( name: "root",
                new Node( name: "1",
                        new Node( name: "11")),
                new Node( name: "2",
                        new Node( name: "21")),
                new Node( name: "3",
                        new Node( name: "31"))
    );
}
```

자식 노드 리턴

너비 우선 순회 검증



projectreactor.io

2.3 그래프 순회

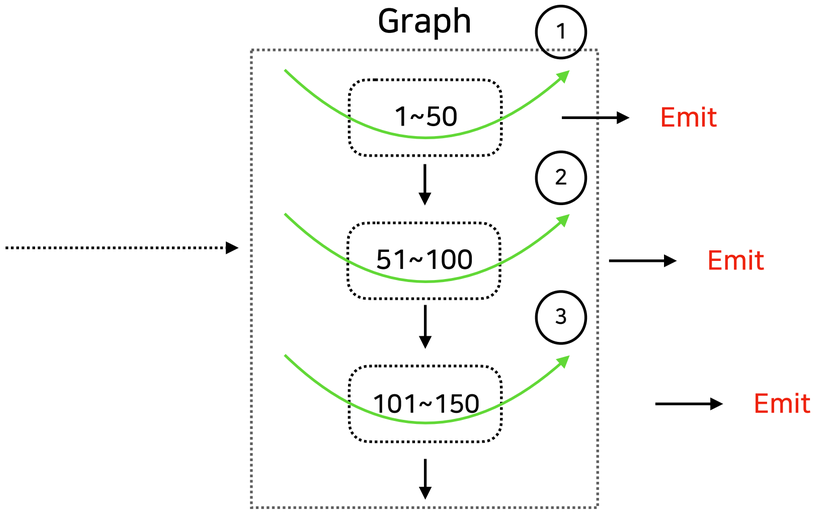
DEVIEW 2021

외부 API 페이징 요청

```
expand(Function<? super T,? extends Publisher<? extends T>> expander)
Recursively expand elements into a graph and emit all the resulting element using a breadth-first traversal strategy.
```

```
public Flux<SpubsChannel> findInvalidChannel0(int limit) {
    AtomicInteger page = new AtomicInteger( initialValue: 0);
    return spubsCpRepository.findChannels(page.get(), limit) Mono<SpubsChannelsResponse>
            .expand(response -> {
                List<SpubsChannel> channels = response.getContent();
                if (CollectionUtils.isEmpty(channels)) {
                    return Mono.empty();
                }
                return spubsCpRepository.findChannels(page.incrementAndGet(), limit);
            }) Flux<SpubsChannelsResponse>
            .map(SpubsChannelsResponse::getContent) Flux<List<SpubsChannel>>
            .flatMapIterable(channels -> channels) Flux<SpubsChannel>
            .filter(SpubsChannel::isNotValidChannel);
}
```

자유변수 변경이 필요



2.3 그래프 순회

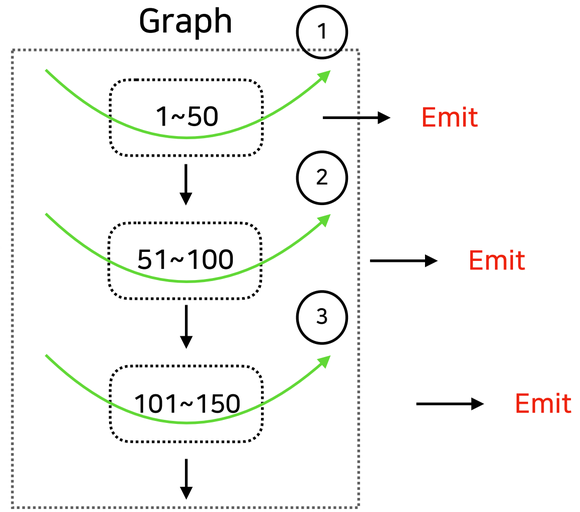
DEVIEW 2021

외부 API 페이징 요청

```
expand(Function<? super T,? extends Publisher<? extends T>> expander)
Recursively expand elements into a graph and emit all the resulting element using a breadth-first traversal strategy.
```

```
public Flux<SpubsChannel> findInvalidChannel1(int limit) {
    return spubsCpRepository.findChannels( page: 0, limit) Mono<SpubsChannelsResponse>
            .expand(response -> {
                List<SpubsChannel> channels = response.getContent();
                if (CollectionUtils.isEmpty(channels)) {
                    return Mono.empty();
                }
                return spubsCpRepository.findChannels(response.getNextPage(), limit);
            }) Flux<SpubsChannelsResponse>
            .map(SpubsChannelsResponse::getContent) Flux<List<SpubsChannel>>
            .flatMapIterable(channels -> channels) Flux<SpubsChannel>
            .filter(SpubsChannel::isNotValidChannel);
}
```

Next Page 추출이 가능한 경우



better! 각 상황에 맞게 사용

projectreactor.io

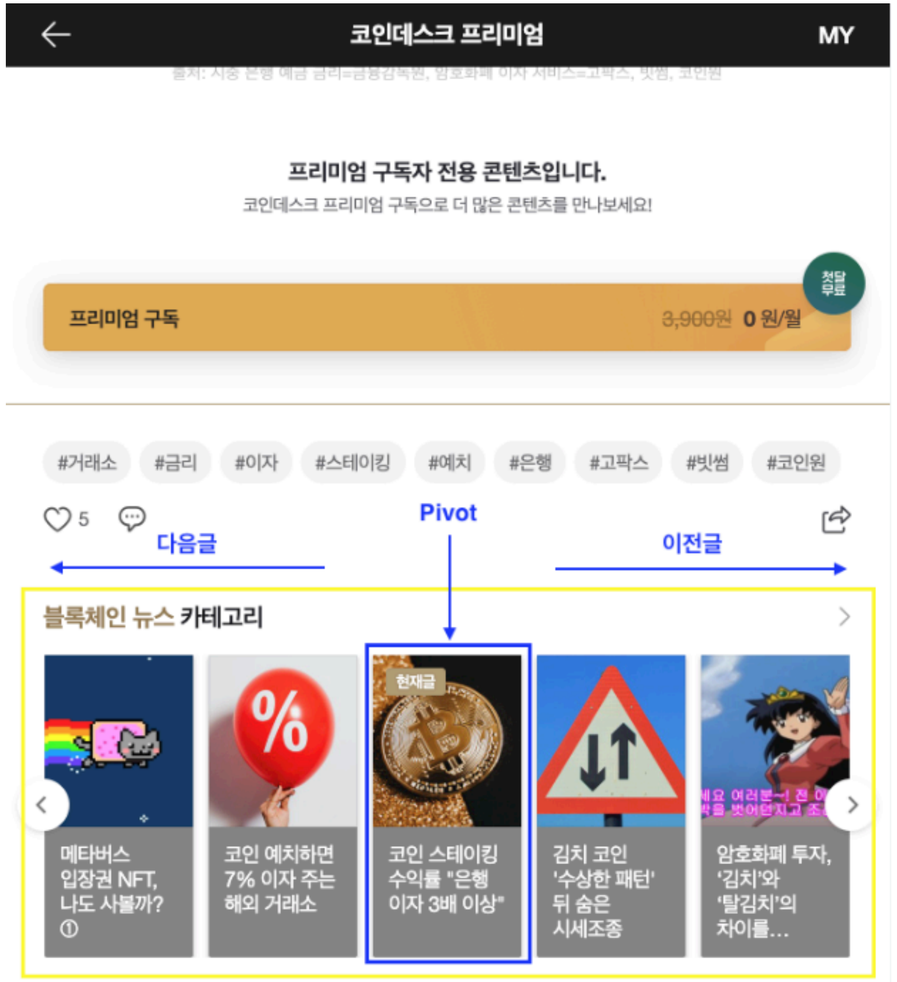
2.3 그래프 순회

이전 글/다음 글 스펙

- 콘텐츠 엔드 하단 -> `동일 카테고리의 콘텐츠 목록` 노출
- 현재 콘텐츠를 가운데 원소로 포함하여 다음 글/이전 글 노출

총 N개 노출

e.g.) N이 5인 경우 -> 다음 글 2개 / 현재 글 / 이전 글 2개



2.3 그래프 순회

DEVIEW 2021

이전 글/다음 글 스펙

- 콘텐츠 엔드 하단 ->`동일 카테고리의 콘텐츠 목록` 노출
- 현재 콘텐츠를 가운데 원소로 포함하여 다음 글/이전 글 노출

노출 Flow

1. 다음 글 N개, 이전 글 N개
2. Reduce
3. 현재 콘텐츠를 기준으로 breadth-first 순회
4. N에 도달한 경우 순회 중지

```
return Flux.mergeSequential(nextContents, previousContents) Flux<Content>
        .distinct(Content::get_id)
        .reduce(new ArrayList<>(), new Reducer<Content>().listReducer) Mono<List
        .map(contents -> new BreadthFirstExpander(contents, contentId, limit))
        .flatMap(BreadthFirstExpander::traverse) Mono<List<Content>>
        .map(contents -> ContentsResponse.builder()
                .contents(contents)
                .build()
        );
```

Application Join

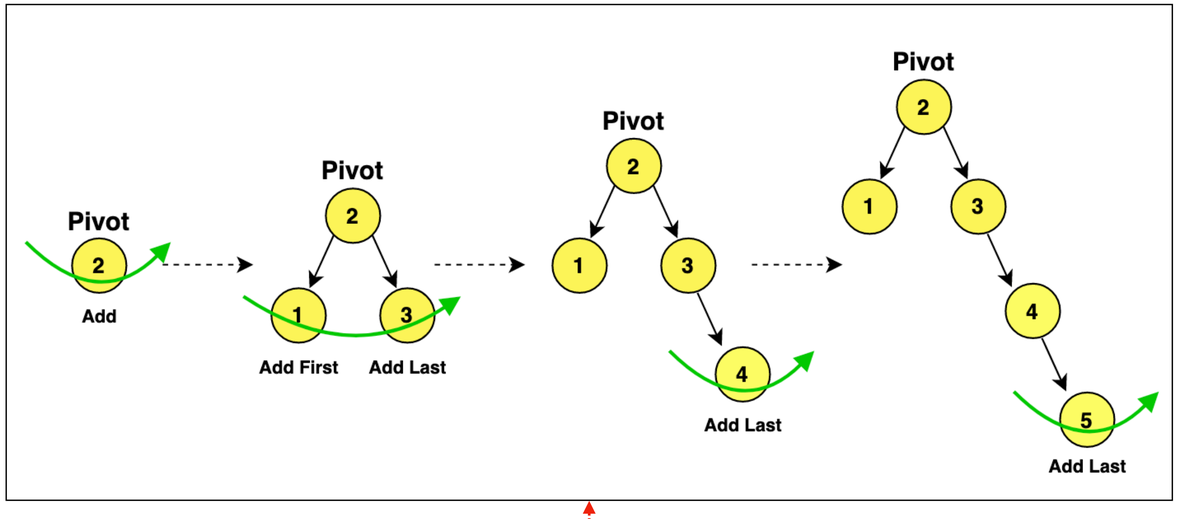
2.3 그래프 순회

DEVIEW 2021

이전 글/다음 글 스펙

- 콘텐츠 엔드 하단 ->`동일 카테고리의 콘텐츠 목록` 노출
- 현재 콘텐츠를 가운데 원소로 포함하여 다음 글/이전 글 노출

Graph




```
public Mono<List<Content>> traverse() {
    int index = pivotIndex(contents, contentId);
    LinkedList<Content> result = new LinkedList<>();
    result.add(contents.get(index));
    visited.add(index);

    return Mono.just(index)
            .expand(idx -> {
                if (result.size() == limit || result.size() == contents.size()) {
                    return Mono.empty();
                }

                Mono<Integer> left = Mono.empty();
                Mono<Integer> right = Mono.empty();

                int leftIdx = idx - 1;
                int rightIdx = idx + 1;

                if (leftIdx >= 0 && !visited.contains(leftIdx)) {
                    left = Mono.just(leftIdx);
                    result.addFirst(contents.get(leftIdx));
                }

                if (rightIdx < contents.size() && !visited.contains(rightIdx)) {
                    right = Mono.just(rightIdx);
                    result.addLast(contents.get(rightIdx));
                }

                visited.add(idx);

                return Flux.mergeSequential(left, right)
                        .filter(next -> !visited.contains(next));
            })
            .then(Mono.just(result));
}
```

2.3 그래프 순회

DEVIEW 2021

Flux/Mono.expandDeep

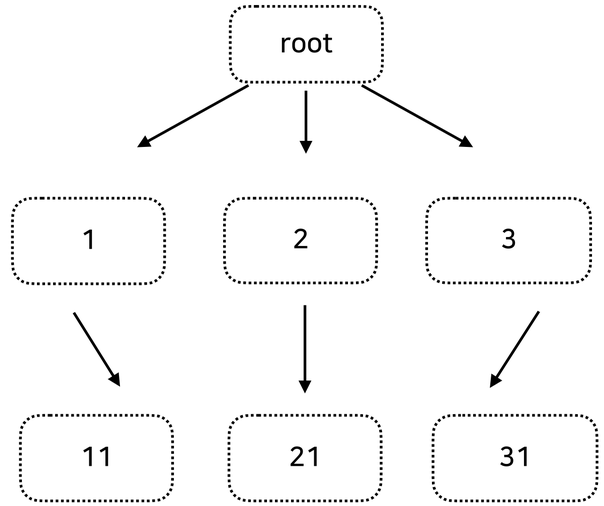
- Reculsively, Depth-first 순회

```
expandDeep(Function<? super T,? extends Publisher<? extends T>> expander)
Recursively expand elements into a graph and emit all the resulting element, in a depth-first traversal order.
```

```
Node sampleNodes() {
    return new Node( name: "root",
                new Node( name: "1",
                        new Node( name: "11")),
                new Node( name: "2",
                        new Node( name: "21")),
                new Node( name: "3",
                        new Node( name: "31"))
    );
}

static class Node {
    final String name;
    final List<Node> children;

    Node(String name, Node... nodes) {
        this.name = name;
        this.children = new ArrayList<>();
        children.addAll(Arrays.asList(nodes));
    }
}
```



projectreactor.io

2.3 그래프 순회

DEVIEW 2021

Flux/Mono.expandDeep

- Reculsively, Depth-first 순회

```
expandDeep(Function<? super T,? extends Publisher<? extends T>> expander)
```
Recursively expand elements into a graph and emit all the resulting element, in a depth-first traversal order.

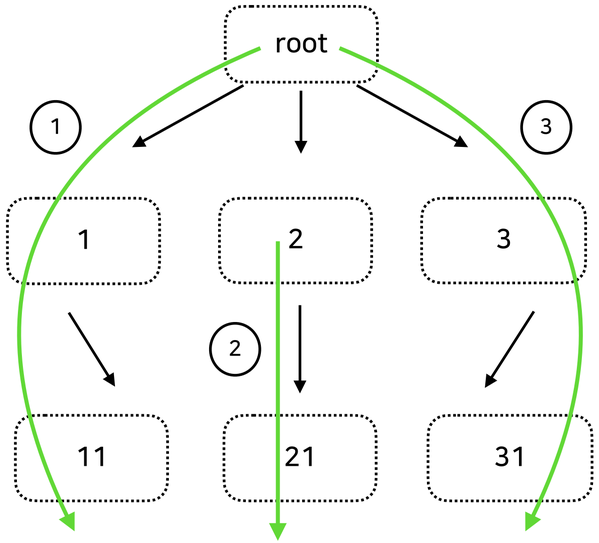
```java
@Test
public void expandDeepTest() {
    Node root = sampleNodes();
    Flux<String> expandDeepFlux = Flux.just(root)
            .expandDeep(v -> Flux.fromIterable(v.children))
            .map(v -> v.name);

    StepVerifier.create(expandDeepFlux) StepVerifier.FirstStep<String>
            .expectNext("root") StepVerifier.Step<String>
            .expectNext("1", "11")
            .expectNext("2", "21")
            .expectNext("3", "31")
            .verifyComplete();
}

Node sampleNodes() {
    return new Node( name: "root",
            new Node( name: "1",
                    new Node( name: "11")),
            new Node( name: "2",
                    new Node( name: "21")),
            new Node( name: "3",
                    new Node( name: "31"))
    );
}
```

자식 노드 리턴

깊이 우선 순회 검증



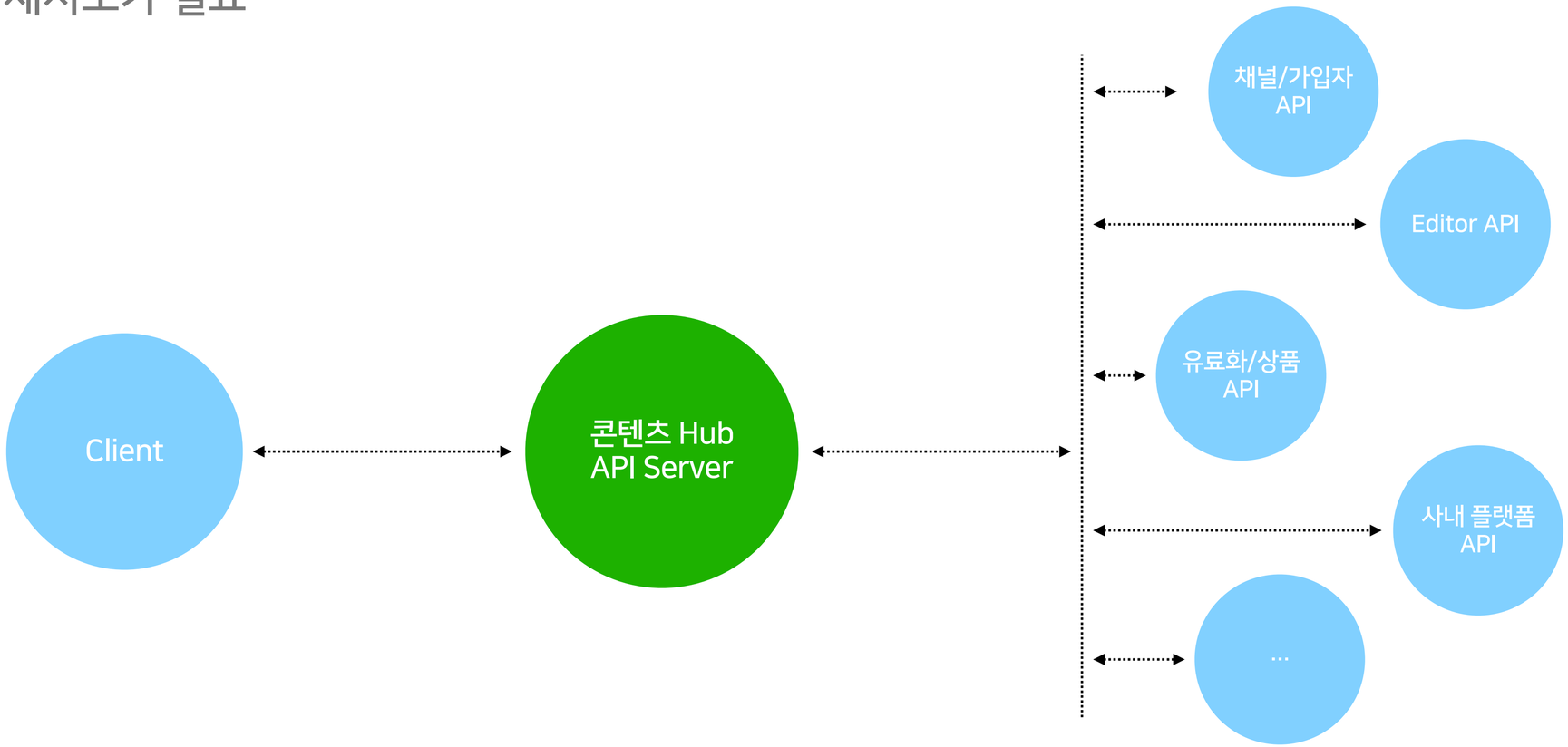
projectreactor.io

2.4 재시도 전략

DEVIEW 2021

프리미엄 콘텐츠 Hub

- 빈도 높은 외부 API 연동
- 배포/장애/네트워크 이슈 등 API 연동 실패
- 적절한 재시도가 필요



2.4 재시도 전략

DEVIEW 2021

Flux/Mono.retryWhen

Mono<T>

reactor.util.retry

Class Retry

java.lang.Object

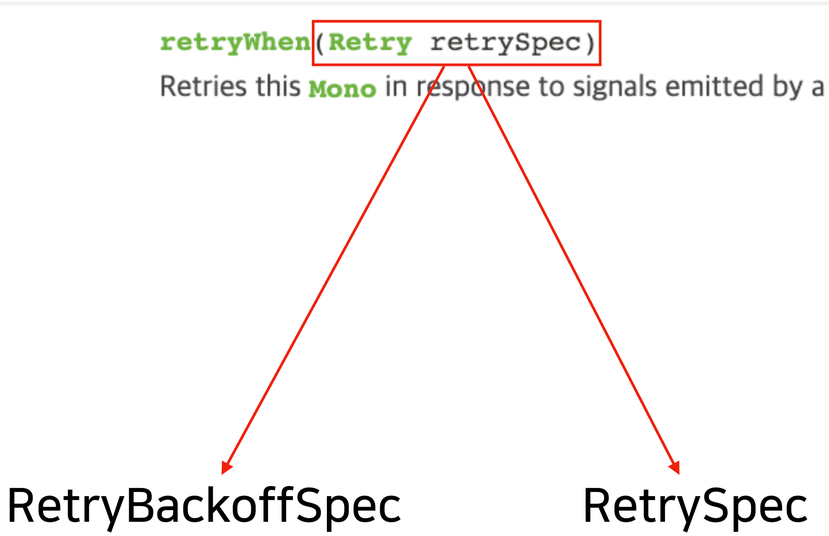
reactor.util.retry.Retry

Direct Known Subclasses:

RetryBackoffSpec, RetrySpec

retryWhen(Retry retrySpec)

Retries this **Mono** in response to signals emitted by a companion **Publisher**.



2.4 재시도 전략

DEVIEW 2021

RetryBackoffSpec

`static RetryBackoffSpec`	**`backoff`**`(long maxAttempts, Duration minBackoff)` A **`RetryBackoffSpec`** preconfigured for exponential backoff strategy with jitter, given a maximum number of retry attempts and a minimum **`Duration`** for the backoff.
`static RetryBackoffSpec`	**`fixedDelay`**`(long maxAttempts, Duration fixedDelay)` A **`RetryBackoffSpec`** preconfigured for fixed delays (min backoff equals max backoff, no jitter), given a maximum number of retry attempts and the fixed **`Duration`** for the backoff.

projectreactor.io

Jitter

3. 데이터 통신(패킷교환 등)에서 말하는 지터

- 패킷 지연(Delay)이 일정하지 않고, 수시로 변하고, 패킷 간의 간격이 일정하지 않는 현상
  - 이는 지연변이 라는 용어로 더 잘 알려져 있음

[정보통신용어기술해설] http://www.ktword.co.kr/test/view/view.php?m_temp1=991

RetrySpec

`static RetrySpec`	**`indefinitely`**`()` A **`RetrySpec`** preconfigured for the most simplistic retry strategy: retry immediately and indefinitely (similar to **`Flux.retry()`**).
`static RetrySpec`	**`max`**`(long max)` A **`RetrySpec`** preconfigured for a simple strategy with maximum number of retry attempts.
`static RetrySpec`	**`maxInARow`**`(long maxInARow)` A **`RetrySpec`** preconfigured for a simple strategy with maximum number of retry attempts over subsequent transient errors.

simple

projectreactor.io

2.4 재시도 전략

DEVIEW 2021

RetryBackoffSpec

`static RetryBackoffSpec backoff(long maxAttempts, Duration minBackoff)`

A `RetryBackoffSpec` preconfigured for exponential backoff strategy with jitter, given a maximum number of retry attempts and a minimum `Duration` for the backoff.

`static RetryBackoffSpec fixedDelay(long maxAttempts, Duration fixedDelay)`

A `RetryBackoffSpec` preconfigured for fixed delays (min backoff equals max backoff, no jitter), given a maximum number of retry attempts and the fixed `Duration` for the backoff.

projectreactor.io

Jitter

3. 데이터 통신(패킷교환 등)에서 말하는 지터

- 패킷 지연(Delay)이 일정하지 않고, 수시로 변하고, 패킷 간의 간격이 일정하지 않는 현상
  - 이는 지연변이 라는 용어로 더 잘 알려져 있음

[정보통신용어기술해설] http://www.ktword.co.kr/test/view/view.php?m_temp1=991

RetrySpec

simple

`static RetrySpec`	`indefinitely()` A `RetrySpec` preconfigured for the most simplistic retry strategy: retry immediately and indefinitely (similar to `Flux.retry()`).
`static RetrySpec`	`max(long max)` A `RetrySpec` preconfigured for a simple strategy with maximum number of retry attempts.
`static RetrySpec`	`maxInARow(long maxInARow)` A `RetrySpec` preconfigured for a simple strategy with maximum number of retry attempts over subsequent transient errors.

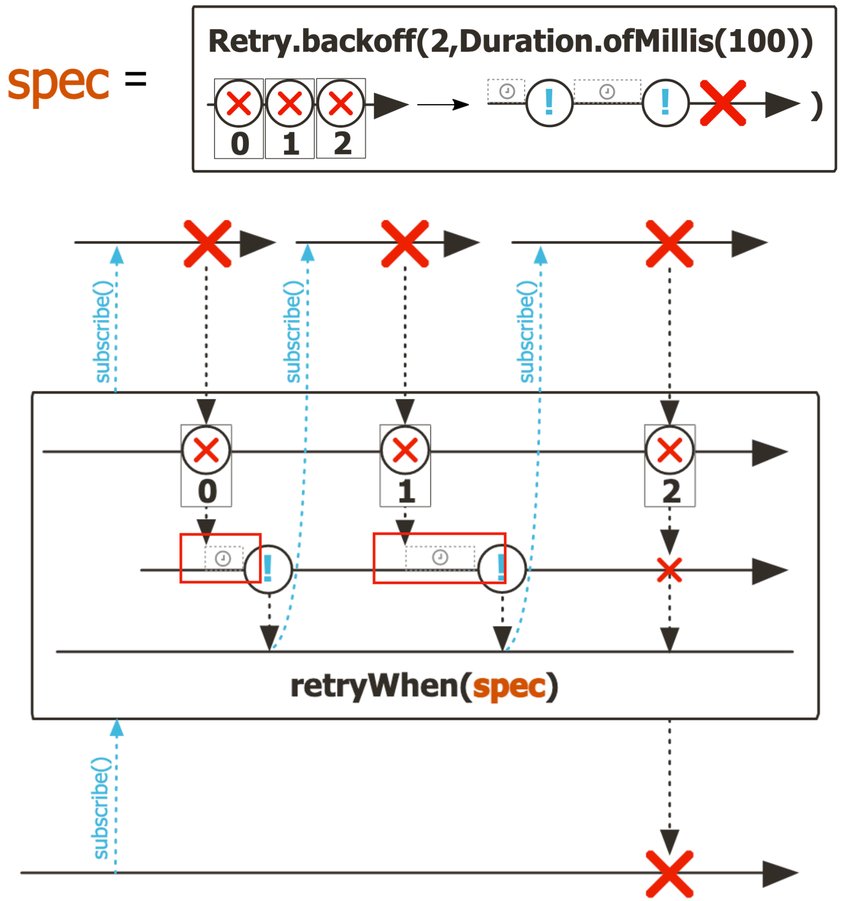
projectreactor.io

2.4 재시도 전략

DEVIEW 2021

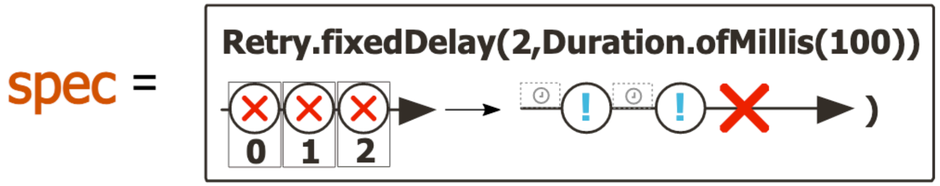
Retry.backoff

- 점진적으로 재시도 간격을 늘린다 with Jitter



Retry.fixedDelay

- 재시도 간격 고정



2.4 재시도 전략

DEVIEW 2021

콘텐츠 Hub의 재시도 전략 - Default: fixedDelay

- Client 별 timeout 설정이 각기 다르기에 Backoff 설정이 무의미해질 가능성이 있음
- Retry에 실패하더라도 client에게 빠른 응답을 보장해야 함
- 배치작업에서 backoff 사용

```
return Mono.just(content)
        .map(contentIdsGenerator::generate)
        .flatMap(c -> productProcessor.registerProduct(c, productInfo, requester)
                .map(content::withSalesDatetime)
        )
        .retryWhen(Retry.fixedDelay(maxAttempts, Duration.ofMillis(100))
                .filter(t -> t instanceof ProductIdDuplicateException)
                .doBeforeRetry(signal -> doSomething())
                .doAfterRetry(signal -> doSomething())
                .onRetryExhaustedThrow((backOffSpec, signal) -> new ProductRegisterException(signal.failure()))
        )
        .flatMap(c -> contentsRepository.insert(c)
                .onErrorResume(t -> onInsertContentFailed(c).then(Mono.error(t)))
        )
```

- Retry..Spec#doAfterRetry/Asnc(..)
- Retry..Spec#doBeforeRetry/Asnc(..)
- Retry..Spec#onRetryExhaustedThrow(..)
- Retry..Spec#filter(..)
- Retry..Spec#jitter (..)
- ...

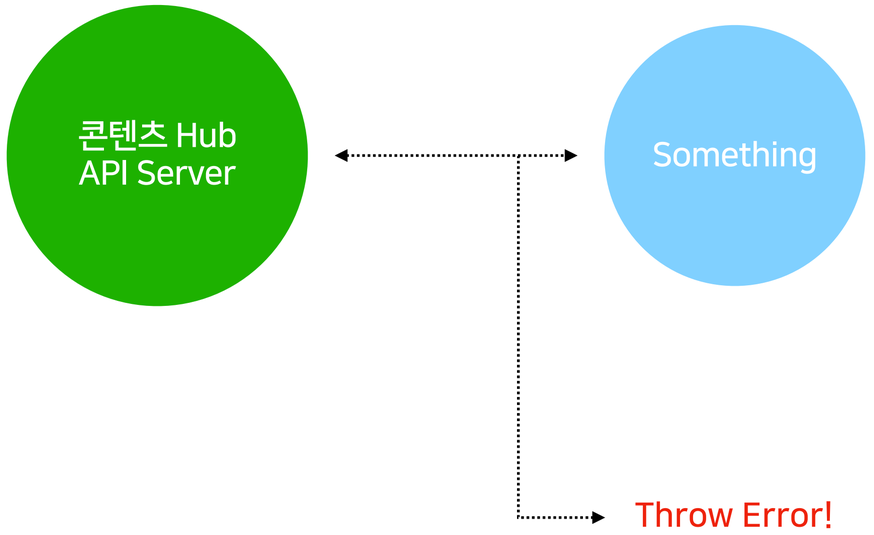
2.5 검증 및 에러 throw

DEVIEW 2021

프리미엄 콘텐츠 Hub

- 프로그래밍 과정에서 데이터를 검증하거나, 에러를 발생시켜야 하는 경우가 빈번
- e.g.) size check / 데이터 상태 체크 등

1. Flux/Mono.handle
2. Flux.concatMap
3. Flux/Mono.flatMap



2.5.1 Flux/Mono.handle

DEVIEW 2021

Flux/Mono.handle

```
public final <R> Mono<R> handle(BiConsumer<? super T,SynchronousSink<R>> handler)
```

```
public interface SynchronousSink<T>
```

Interface to produce synchronously "one signal" to an underlying `Subscriber`.

At most one `next(T)` call and/or one `complete()` or `error(Throwable)` should be called per invocation of the generator function.

2.5.1 Flux/Mono.handle

DEVIEW 2021

Flux/Mono.handle - SynchronousSink

```
AtomicInteger atomicInteger = new AtomicInteger( initialValue: 0);
Flux<Integer> integerFlux = Flux.generate(sink -> {
    sink.next(atomicInteger.getAndIncrement());
    sink.next(atomicInteger.getAndIncrement());
    if (atomicInteger.get() == 10) {
        sink.complete();
    }
});

integerFlux.subscribe(new BaseSubscriber<>() {
    @Override
    protected void hookOnSubscribe(Subscription subscription) {
        request( n: 1);
    }
});
```

SynchronousSink는 동기적으로 한 번에 하나의 데이터만 생성한다.

Generator 함수 내에서

- next/complete/error 모두 최대 1번 호출
- next/complete/error 혼합 호출 불가

- void next(T t)
- void complete()
- void error(Throwable e)

```
reactor.core.Exceptions$ErrorCallbackNotImplemented: java.lang.IllegalStateException: More than one call to onNext
    Caused by:
    java.lang.IllegalStateException: More than one call to onNext
        at reactor.core.publisher.FluxGenerate$GenerateSubscription.next(FluxGenerate.java:157)
```

2.5.1 Flux/Mono.handle

DEVIEW 2021

Flux/Mono.map vs Flux/Mono.handle

```
Flux<ChannelInfo> channelMapFlux = findChannels(CpType.Premium)
        .map(channel -> {
            if (!isValid(channel)) {
                // ignore 불가능
                return new ChannelInfo();
            }
            else {
                ChannelInfo channelInfo = channel.getChannelInfo();
                if (isValidState(channelInfo.getSvcState())) {
                    // error
                    throw new RuntimeException();
                }
                // emit item
                return channel.getChannelInfo();
            }
        });
```

Error

Emit

```
Flux<ChannelInfo> channelHandleFlux = findChannels(CpType.Premium)
        .handle((channel, channelSynchronousSink) -> {
            if (!isValid(channel)) {
                // ignore element 가능
            }
            else {
                ChannelInfo channelInfo = channel.getChannelInfo();
                if (isValidState(channelInfo.getSvcState())) {
                    // error
                    channelSynchronousSink.error(new RuntimeException());
                }
                // emit item
                channelSynchronousSink.next(channelInfo);
            }
        });
```

Skip 가능

	map	Handle	비고
Element ignore	X	O	- map은 1대1 매핑 - handle은 skip 가능
Error	throw new ~	sink.error(Throwable t)	- 모두 onError로 전파 O - map은 unchecked exception only
Emit item	return t	sink.next(T t)	
Mono/Flux 지원 여부	모두 지원	모두 지원	

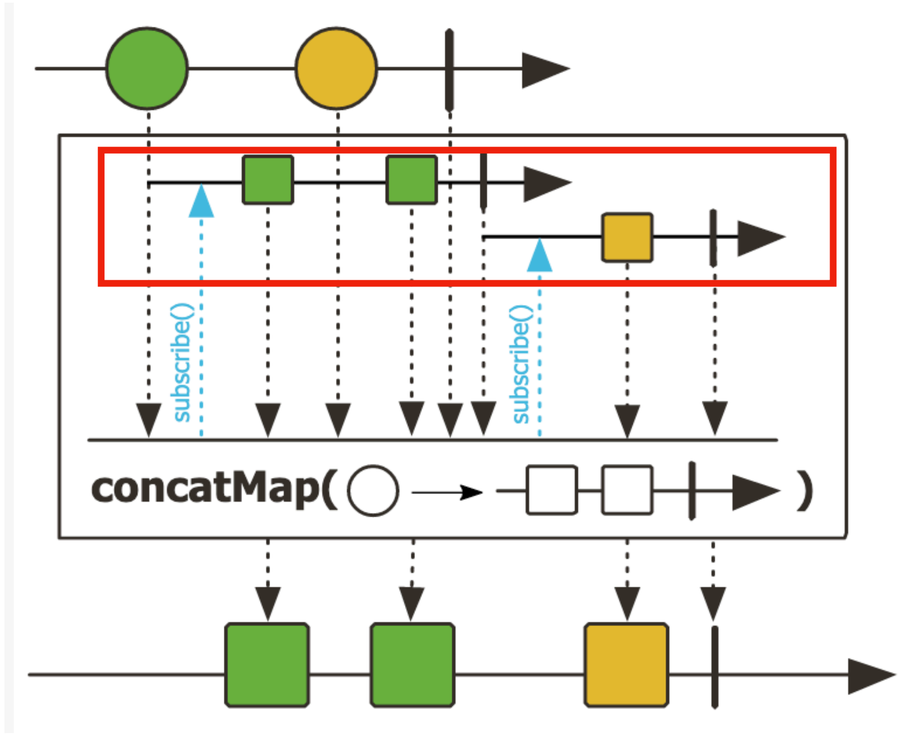
Synchronous 처리 시, 더 복잡한 조건의 element processing이 필요한 경우에 handle

2.5.2 Flux.concatMap

DEVIEW 2021

Flux.concatMap

```
public final <V> Flux<V> concatMap(Function<? super T,? extends Publisher<? extends V>> mapper)
```



Inner Publisher의 complete 이후 next generate

Asynchronous handling of a single stream at a time

- concatMap은 asynchronous를 유지하며, 마치 map과 비슷하게 활용 가능
  (데이터 매핑/변환 관점이 아닌 순차적 처리 측면)
- Flux 처리시 map -> concatMap으로 대체 가능

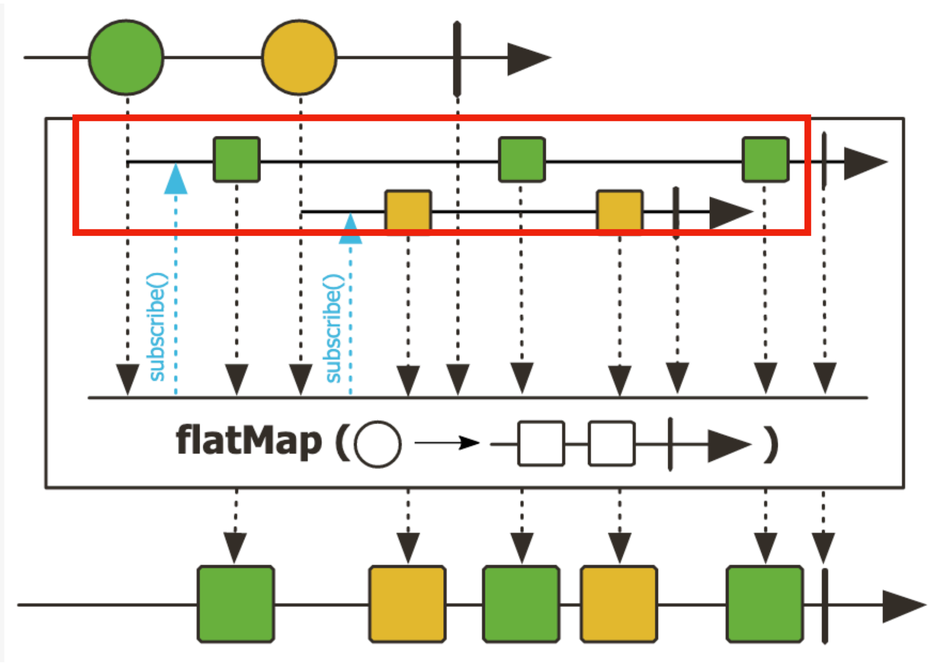
```
Flux<ChannelInfo> channelConcatMapFlux = findChannels(CpType.Premium)
        .concatMap(channel -> {
            if (!isValid(channel)) {
                return Mono.empty();                          Empty
            }
            else {
                ChannelInfo channelInfo = channel.getChannelInfo();
                if (isValidState(channelInfo.getSvcState())) {
                    // error
                    return Mono.error(new RuntimeException());   Error
                }
                // emit item
                return Mono.just(channelInfo);                   Emit
            }
        });
```

projectreactor.io

2.5.3 Flux/Mono.flatMap

DEVIEW 2021

Flux/Mono.flatMap



Inner Publisher의 complete를 기다리지 않음

flatMap은 asynchronous / 여러 stream을 merge 하도록 설계

- Sub stream asynchronous handling
- 내부적으로 추가적인 메모리 사용
  (e.g. stream 별 Queue 할당 등)

Scalar stream/error 를 던지기 위해 flatMap을 사용하지 않아도 됨.
-> concatMap으로 대체 가능

```
Flux<ChannelInfo> channelFlatMapFlux = findChannels(CpType.Premium)
        .flatMap(channel -> {
            if (!isValid(channel)) {
                return Mono.empty();                          Empty
            }
            else {
                ChannelInfo channelInfo = channel.getChannelInfo();
                if (isValidState(channelInfo.getSvcState())) {
                    // error
                    return Mono.error(new RuntimeException());   Error
                }
                // emit item
                return Mono.just(channelInfo);                   Emit
            }
        });
                scalar stream
```

projectreactor.io

2.5.4 검증 및 에러 throw 정리

DEVIEW 2021

Flux/Mono.handle

- emit/error/ignore 모두 필요한 복잡한 케이스를 컨트롤하는 경우

Flux.concatMap

- Asynchronous를 유지해야 하는 경우
- Flux chaining에서 map을 사용하고 있는 경우 대체 가능
- flatMap으로 scalar stream / error 를 반환하는 경우 대체 가능

Flux/Mono.flatMap

- Asynchronous를 유지하며 1:N 변환 혹은 flatten이 필요한 경우
- 이미 사용하고 있는 경우

Flux/Mono.map

- Mono chaining에서 간단한 매핑 혹은 error 를 반환하는 경우
- 이미 사용하고 있는 경우

2.5.4 검증 및 에러 throw 정리

DEVIEW 2021

시간이 소요되는 작업 (e.g. 외부 API/DB 연동)은?

- 항상 Mono/Flux 타입을 반환
- block() 사용 X
- 사용처에서 flatMap으로 flatten 시켜줘야 함
  (Mono는 concatMap을 지원하지 않으니, 일반적으로 flatMap 사용)

2.5.4 검증 및 에러 throw 정리

DEVIEW 2021

시간이 소요되는 작업 (e.g. 외부 API/DB 연동)은?

- 항상 Mono/Flux 타입을 반환
- block() 사용 X
- 사용처에서 flatMap으로 flatten 시켜줘야 함

  (Mono는 concatMap을 지원하지 않으니, 일반적으로 flatMap 사용)

시간이 소요되는 작업에 map 연산자를 사용하고 있는가?

- No, 단순 매핑 및 검증에만 사용

2.5.4 검증 및 에러 throw 정리

DEVIEW 2021

시간이 소요되는 작업 (e.g. 외부 API/DB 연동)은?
- 항상 Mono/Flux 타입을 반환
- block() 사용 X
- 사용처에서 flatMap으로 flatten 시켜줘야 함

  (Mono는 concatMap을 지원하지 않으니, 일반적으로 flatMap 사용)

시간이 소요되는 작업에 map 연산자를 사용하고 있는가?
- No, 단순 매핑 및 검증에만 사용

그렇다면 타이트하게 연산자를 가져가는 정책이 성능상 많은 이점을 주는가?

Reactor의 다양하고 어려운 연산자

실제 병목 구간이 아닐 가능성이 높다.

선택과 집중

자유도를 높이고 더 영향도가 높은 곳에 집중

2.6 리액터 테스트

DEVIEW 2021

프리미엄 콘텐츠 Hub

- StepVerifier
- PublisherProbe
- TestPublisher

2.6.1 StepVerifier

DEVIEW 2021

StepVerifier

- Publisher sequence를 테스트하는 declarative way
- 데이터 스트림/예외/완료 검증
- 가장 높은 빈도로 사용됨

2.6.2 PublisherProbe

PublisherProbe

- Control flow와 관련된 테스트를 위해 Mono/Flux를 쉽게 얻을 수 있는 test utility
- subscribe/cancel/request event 캡처 및 검증 가능
- 정적 팩토리 메소드 제공
  - PublisherProbe.of(Publisher<? extends T> source)
  - PublisherProbe.empty()

subscribe/cancel/request Capturing

long	**subscribeCount**()
boolean	**wasCancelled**()
boolean	**wasRequested**()
boolean	**wasSubscribed**()

Publisher 추적 가능

assert{~}

assertWasCancelled()
Check that the probe was cancelled at least once, or throw an **AssertionError**.

assertWasNotCancelled()
Check that the probe was never cancelled, or throw an **AssertionError**.

assertWasNotRequested()
Check that the probe was never requested, or throw an **AssertionError**.

assertWasNotSubscribed()
Check that the probe was never subscribed to, or throw an **AssertionError**.

assertWasRequested()
Check that the probe was requested at least once, or throw an **AssertionError**.

assertWasSubscribed()
Check that the probe was subscribed to at least once, or throw an **AssertionError**.

2.6.2 PublisherProbe - retry 검증

DEVIEW 2021

```
@Test
void 예약발행_상품아이디_중복인경우_재요청_테스트() {
    CpType cpType = CpType.Premium;
    LocalDateTime dateTime = LocalDateTime.of( year: 2021,  month: 10,  dayOfMonth: 10,  hour: 10,  minute: 10,  second: 10,  nanoOfSecond: 100);
    Content content = Content._builder()._id(test)
            .cpType(cpType).cpName(test).subId(test)
            .title(test).naverId(test)
            .type(ContentType.TEXT)
            .registerDatetime(dateTime)
            .publishDatetime(dateTime.plusDays(1))
            .productType(ProductType.TICKET)
            .build();

    PublisherProbe<ProductResponse> probe = PublisherProbe.of(
            Mono.error(new ProductIdDuplicateException())
    );

    when(productProcessor.registerProduct(any(), any(), any()))
            .thenReturn(probe.mono());

    ReservedContentPublisher publisher = new ReservedContentPublisher(contentsRepository,
            productProcessor,
            contentIdsGenerator
    );

    StepVerifier.create(publisher.publish(content, new ProductInfo(), Requester.builder()
            .naverId(test)
            .build())
    ).expectError(RuntimeException.class).verify();

    probe.assertWasRequested();
    probe.assertWasSubscribed();

    assertThat(probe.subscribeCount()).isEqualTo(ReservedContentPublisher.maxAttempts + 1);
}
```

1. 상품 아이디 중복시 N번 retry 로직을 포함

2.6.2 PublisherProbe - retry 검증

DEVIEW 2021

```
@Test
void 예약발행_상품아이디_중복인경우_재요청_테스트() {
    CpType cpType = CpType.Premium;
    LocalDateTime dateTime = LocalDateTime.of( year: 2021,  month: 10,  dayOfMonth: 10,  hour: 10,  minute: 10,  second: 10,  nanoOfSecond: 100);
    Content content = Content.builder()._id(test)
            .cpType(cpType).cpName(test).subId(test)
            .title(test).naverId(test)
            .type(ContentType.TEXT)
            .registerDatetime(dateTime)
            .publishDatetime(dateTime.plusDays(1))
            .productType(ProductType.TICKET)
            .build();

    PublisherProbe<ProductResponse> probe = PublisherProbe.of(
            Mono.error(new ProductIdDuplicateException())
    );

    when(productProcessor.registerProduct(any(), any(), any()))
            .thenReturn(probe.mono());

    ReservedContentPublisher publisher = new ReservedContentPublisher(contentsRepository,
            productProcessor,
            contentIdsGenerator
    );

    StepVerifier.create(publisher.publish(content, new ProductInfo(), Requester.builder()
            .naverId(test)
            .build())
    ).expectError(RuntimeException.class).verify();

    probe.assertWasRequested();
    probe.assertWasSubscribed();

    assertThat(probe.subscribeCount()).isEqualTo(ReservedContentPublisher.maxAttempts + 1);
}
```

2. 상품등록(registerProduct)은
정의 된 Publisher를 반환하도록 설정

1. 상품 아이디 중복시 N번 retry 로직을 포함

2.6.2 PublisherProbe - retry 검증

DEVIEW 2021

```
@Test
void 예약발행_상품아이디_중복인경우_재요청_테스트() {
    CpType cpType = CpType.Premium;
    LocalDateTime dateTime = LocalDateTime.of( year: 2021,  month: 10,  dayOfMonth: 10,  hour: 10,  minute: 10,  second: 10,  nanoOfSecond: 100);
    Content content = Content.builder()._id(test)
            .cpType(cpType).cpName(test).subId(test)
            .title(test).naverId(test)
            .type(ContentType.TEXT)
            .registerDatetime(dateTime)
            .publishDatetime(dateTime.plusDays(1))
            .productType(ProductType.TICKET)
            .build();

    PublisherProbe<ProductResponse> probe = PublisherProbe.of(
            Mono.error(new ProductIdDuplicateException())
    );

    when(productProcessor.registerProduct(any(), any(), any()))
            .thenReturn(probe.mono());

    ReservedContentPublisher publisher = new ReservedContentPublisher(contentsRepository,
            productProcessor,
            contentIdsGenerator
    );

    StepVerifier.create(publisher.publish(content, new ProductInfo(), Requester.builder()
            .naverId(test)
            .build())
    ).expectError(RuntimeException.class).verify();

    probe.assertWasRequested();
    probe.assertWasSubscribed();

    assertThat(probe.subscribeCount()).isEqualTo(ReservedContentPublisher.maxAttempts + 1);
}
```

2. 상품등록(registerProduct)은 정의 된 Publisher를 반환하도록 설정

1. 상품 아이디 중복시 N번 retry 로직을 포함

3. probe requested / subscribed 검증

4. probe subscribe count 검증
- probe.subscribeCount() == 원본 요청 + retry 요청

Mockito
counting method invocations
대체 가능

- Mockito.times(n)

2.6.3 TestPublisher

DEVIEW 2021

TestPublisher

- 직접 데이터 조작이 가능한 Publisher 생성
- 테스트 목적으로 onNext/onComplete/onError 트리거 가능
- Reactive Streams의 rule 위반 가능 (e.g. null 허용)
- Hot / cold 주의해서 사용
- Implements PublisherProbe

- TestPublisher<T> emit(T... values)
- TestPublisher<T> error(Throwable t)
- TestPublisher<T> next(T value)
- TestPublisher<T> next(T first, T... rest)

```
static <T> TestPublisher<T> create()
```
Create a standard hot **`TestPublisher`**.

```
static <T> TestPublisher<T> createCold()
```
Create a cold **`TestPublisher`**, which can be subscribed to by multiple subscribers.

2.6.3 TestPublisher

DEVIEW 2021

PublisherProbe.empty()

```
@Test
public void PublisherProbe_test() {
    PublisherProbe<Integer> publisher = PublisherProbe.empty();

    StepVerifier.create(toTest(publisher.flux())) StepVerifier.FirstStep<Integer>
            .expectNext(1) StepVerifier.Step<Integer>
            .expectNext(2)
            .expectNext(3)
            .verifyComplete();

    publisher.assertWasSubscribed();
}

public Flux<Integer> toTest(Flux<Integer> integerFlux) {
    return integerFlux.concatMap(i -> Mono.just(i + 1))
            .switchIfEmpty(Flux.fromIterable(Arrays.asList(1, 2, 3)));
}
```

통과

통과

테스트 통과

2.6.3 TestPublisher

TestPublisher.create()

```java
@Test
public void TestPublisher_test() {
    TestPublisher<Integer> publisher = TestPublisher.<Integer>create().emit();

    StepVerifier.create(toTest(publisher.flux())) StepVerifier.FirstStep<Integer>
            .expectNext(1) StepVerifier.Step<Integer>
            .expectNext(2)
            .expectNext(3)
            .verifyComplete();

    publisher.assertWasSubscribed();
}

public Flux<Integer> toTest(Flux<Integer> integerFlux) {
    return integerFlux.concatMap(i -> Mono.just(i + 1))
            .switchIfEmpty(Flux.fromIterable(Arrays.asList(1, 2, 3)));
}
```

통과

실패

테스트 실패

```
PublisherProbe should have been subscribed but it wasn't
```

2.6.3 TestPublisher

TestPublisher.createCold()

```
@Test
public void TestPublisher_test() {
    TestPublisher<Integer> publisher = TestPublisher.<Integer>createCold().emit();

    StepVerifier.create(toTest(publisher.flux())) StepVerifier.FirstStep<Integer>
            .expectNext(1) StepVerifier.Step<Integer>
            .expectNext(2)
            .expectNext(3)
            .verifyComplete();

    publisher.assertWasSubscribed();
}
```

→ 테스트 통과

TestPublisher.create() - hot publisher를 생성

테스트가 통과하려면 createCold()로 변경

TestPublisher는 주의를 필요로 하기에
Reactive Streams의 rule을 위반하려는 것이 아니라면

PublisherProbe를 사용하는 것이 좋은 선택

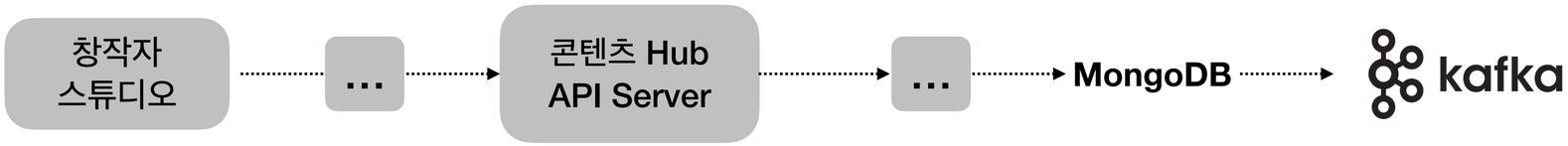
3. Kafka로 보는 콘텐츠 Hub.SSUL

3.1 콘텐츠 발 이벤트

DEVIEW 2021

콘텐츠 관련 모든 C/U/D 이벤트는 kafka로 publish

- 콘텐츠 히스토리 저장
- 알림/뉴스레터 전송
- 캐시 갱신
- 카운트 갱신
- 타 플랫폼 연동
- …



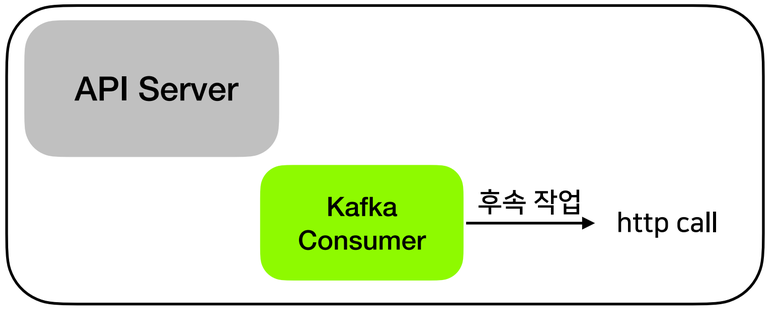
Code base publish / Mongodb Change Streams

3.1 콘텐츠 발 이벤트

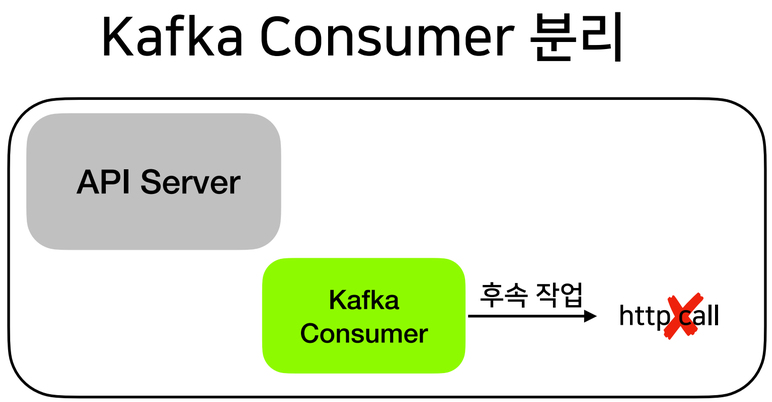
API Server / Consumer 분리 구성

- 이벤트별 독립적 처리를 필요로 하는 경우
- 기능/배포 격리가 필요한 경우

Kafka Consumer 분리



3.1 콘텐츠 발 이벤트



3.1 콘텐츠 발 이벤트

DEVIEW 2021

Basic 한 접근

N번 Retry 이후 commit -> 해당 이벤트 skip

- 이벤트 유실 대응 중요도가 낮은 Kafka Consumer
- 이벤트 순서에 민감한 Kafka Consumer

e.g.) 캐시 갱신

단점 - 처리 속도 저하

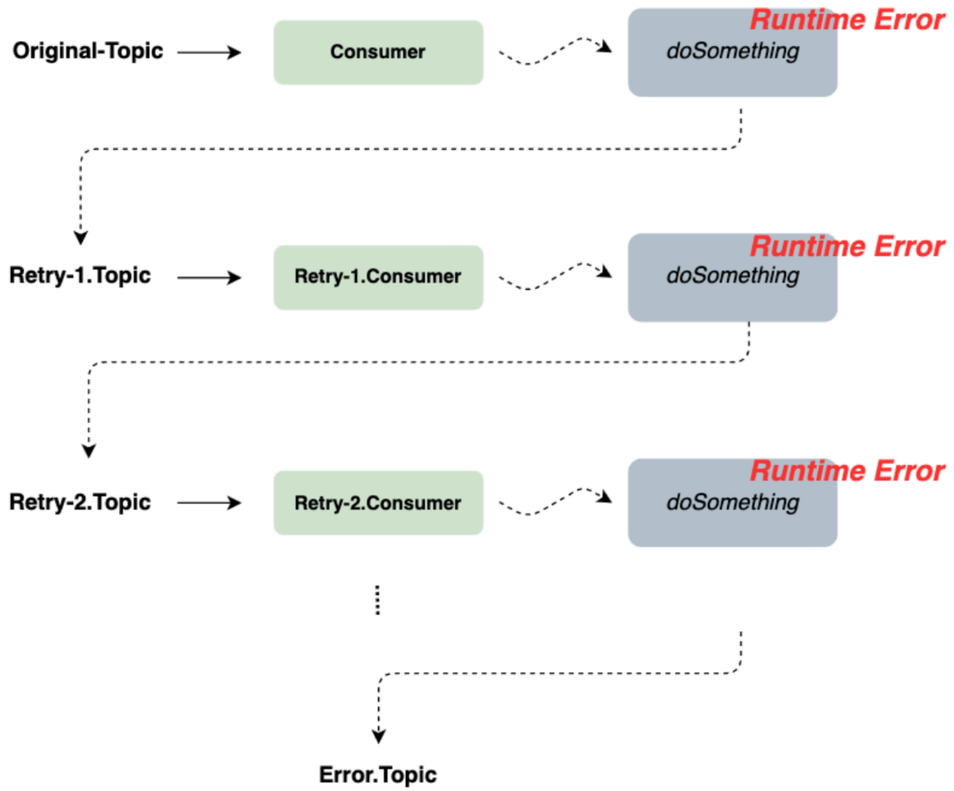
3.2 Single App 기반 Retry/DLT pipeline

DEVIEW 2021

Single App 기반 Retry/DLT pipeline 개발

Retry topic / Dead letter topic

- 런타임 에러시 유연한 재처리를 위해 개발
- Retry topic ~ DLT 까지의 데이터 파이프라인 구성



런타임 중 동적으로 Retry Kafka Endpoint (Listener)를 등록 시켜
Single 애플리케이션 기반으로 메인 컨슈머 및 리트라이 컨슈머를 모두 구동

3.2 Single App 기반 Retry/DLT pipeline

DEVIEW 2021

Usage

- 설정 기반 config
- 메인 컨슈머 / 리트라이 컨슈머가 서로 다른 메소드 사용 가능

KafkaDynamicEndpoint Processor 사용 가이드

1. AbstractRetryJsonValueListener 상속

예제

```
public class ContentHistoryListener extends AbstractRetryJsonValueListener<ContentMessage>
        private static final String GROUP_ID = "content-history-processor";

        public ContentHistoryListener() {
                super(GROUP_ID, ContentMessage.class);
        }

        @KafkaListener(
                topics = "${media.kafka.topic.sps-contents}",
                groupId = GROUP_ID,
                containerFactory = "contentHistoryListenerContainerFactory"
        )
        public void consume(@NonNull @Payload ContentMessage message, @Headers Map<String,
                doSomething();
        }
```

2. `dynamic-kafka` config 정의

```
dynamic-kafka:
  enable: true
  default: # default consumer
    classPath: com.naver.media.kafka.scs.premium.history.ContentHistoryListener
    methodName: consume
    containerFactory: contentHistoryListenerContainerFactory
    retry: 2
    interval: 5m # 현재 미지원
```

3.2 Single App 기반 Retry/DLT pipeline

DEVIEW 2021

Consumer group 별 retry/dead-letter topic format

Retry / Dead-Letter Topic Format

Retry Topic Format	Error Topic Format
`retry-{N}.{original-topic}.{groupId}`	`error.{original-topic}.{groupId}`

예시

- 기준 topic: **premium-contents**
- 기준 groupId: **premium-listener**
- `dynamic-kafka.default.retry=3`

대상	Topic Name
Retry-Topic	retry-1.premium-contents.premium-listener
Retry-Topic	retry-2.premium-contents.premium-listener
Retry-Topic	retry-3.premium-contents.premium-listener
Dead-Letter-Topic	error.premium-contents.premium-listener

3.2 Single App 기반 Retry/DLT pipeline

DEVIEW 2021

Retry Consumer에서 원본 메시지 timestamp 활용

- DeadLetterRecoverer 커스텀 구현을 통해 원본 메시지의 timestamp 헤더 추가
- 추후 작업에 따라 순서를 따질 수 있음
  - e.g.) timestamp를 기반으로 key를 생성하여 이후 데이터를 저장하는 case
  - e.g.) last timestamp를 이후 데이터에 포함하는 case

원본 메시지의 timestamp

- 원본 메시지의 produced timestamp는 메시지 헤더의 `dynamic-kafka-original-timestamp` 를 통해 접근
- `dynamic-kafka-original-timestamp` 는 `Long` type의 byte buffer로 이루어져 있기에 변환해서 사용

```
@KafkaListener(
        topics = "${media.kafka.topic.sps-contents}",
        groupId = GROUP_ID,
        containerFactory = "contentHistoryListenerContainerFactory"
)
public void consume(@NonNull @Payload PremiumMessage<Content> message, ConsumerRecordMetadata metadata,
        @Headers Map<String, Object> headers) {
    long timestamp = metadata.timestamp();

    if (headers.containsKey( o: "dynamic-kafka-original-timestamp")) {
        byte[] b = (byte[])headers.get("dynamic-kafka-original-timestamp");
        ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES);
        buffer.put(b);
        buffer.flip();
        timestamp = buffer.getLong();
    }
```

3.2 Single App 기반 Retry/DLT pipeline

DEVIEW 2021

적용 사례 - 컨텐츠 변경 이력을 쌓는 경우

Topics

Show 10 entries

Search: psContents-complete

Topic	# Partitions	# Brokers	Brokers Spread %	Brokers Skew %	Brokers Leader Skew %	# Replicas	Under Replicated %	Producer Message/Sec	Summed Recent Offsets
media-data-spsContents-complete-changed	1	3	60	0	0	3	0	0.00	32,626
retry-1.media-data-spsContents-complete-changed.content-history-processor	1	3	60	0	0	3	0	0.00	100
retry-2.media-data-spsContents-complete-changed.content-history-processor	1	3	60	0	0	3	0	0.00	93
error.media-data-spsContents-complete-changed.content-history-processor	1	3	60	0	0	3	0	0.00	75

- 메시지 처리 속도 증가
- 메인 컨슈머에서 Retry 로직 없이 API 연동 에러 대응 가능
- 처리에 최종 실패한 이벤트는 error topic으로 수렴
- 원본 메시지의 timestamp 기반으로 db key를 생성 -> 순서가 변경되어도 이슈 없음

3.3 복구

DEVIEW 2021

복구 flow

- 필요한 경우 DLT(error) topic 대상으로 재처리 consumer run
- DB 수동 변경

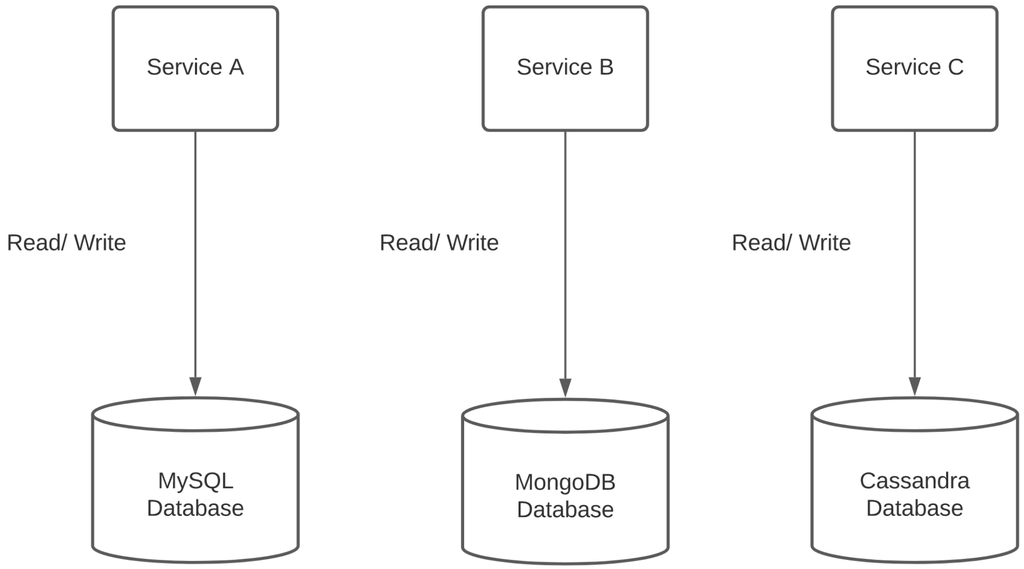
서비스 비노출 데이터 or 노출 영향이 적은 데이터
-> 뒤늦게 발견되거나, 작업에 시간이 소요되어도 OK

3.3 복구

DEVIEW 2021

우리는.. Microservice per Database

- Polyglot Database
- 서비스 간 순차적 write/update flow 빈번



https://www.baeldung.com/cs/microservices-db-design

3.3 복구

복구 flow

- 필요한 경우 DLT(error) topic 대상으로 재처리 consumer run
- DB 수동 변경

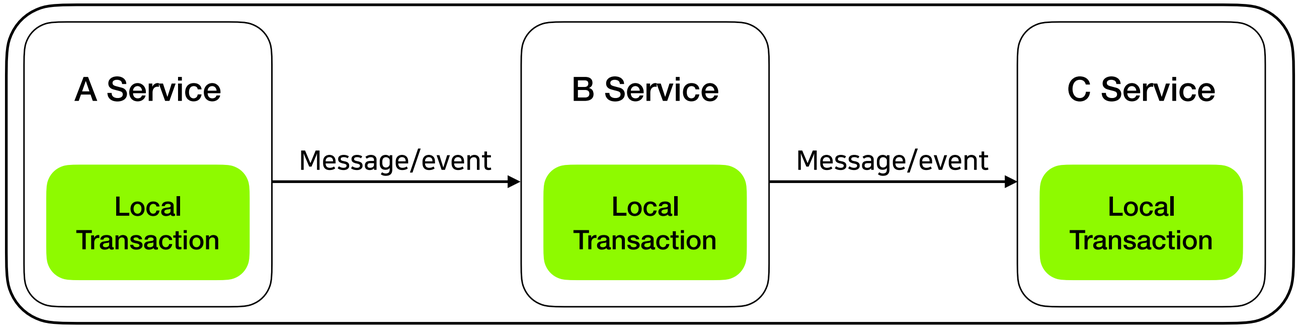
서비스 비노출 데이터 or 노출 영향이 적은 데이터
-> 뒤늦게 발견되거나, 작업에 시간이 소요되어도 영향이 작다.

타 Part의 Microservice 와 연동하며 데이터 상태를 변경하는 경우는?

- 빠른 복구가 필요함
- 서비스 간 Data Consistency를 높게 유지할 필요가 있음

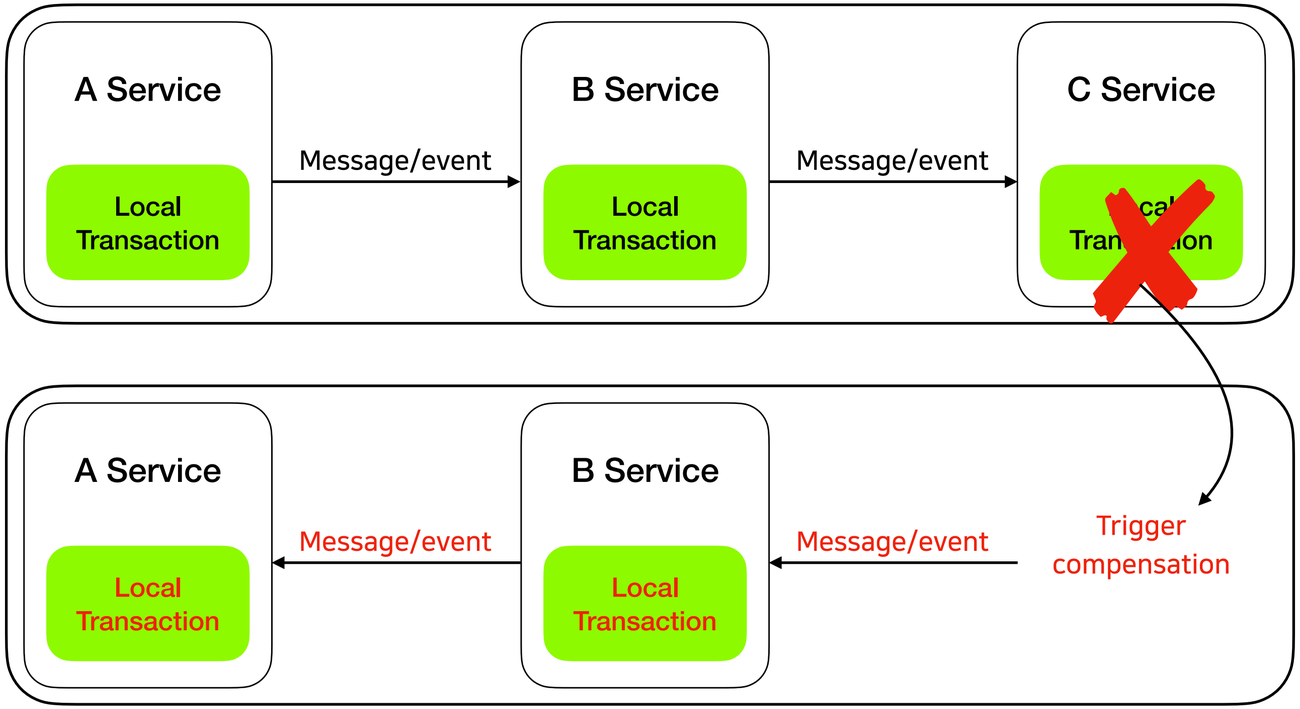
3.4 Saga

- Sequence of local transactions
- 각 local transaction이 완료된 후 이벤트/메시지 발행



3.4 Saga

- Sequence of local transactions
- 각 local transaction이 완료된 후 이벤트/메시지 발행
- Compensation 이벤트/메시지
- Data consistency

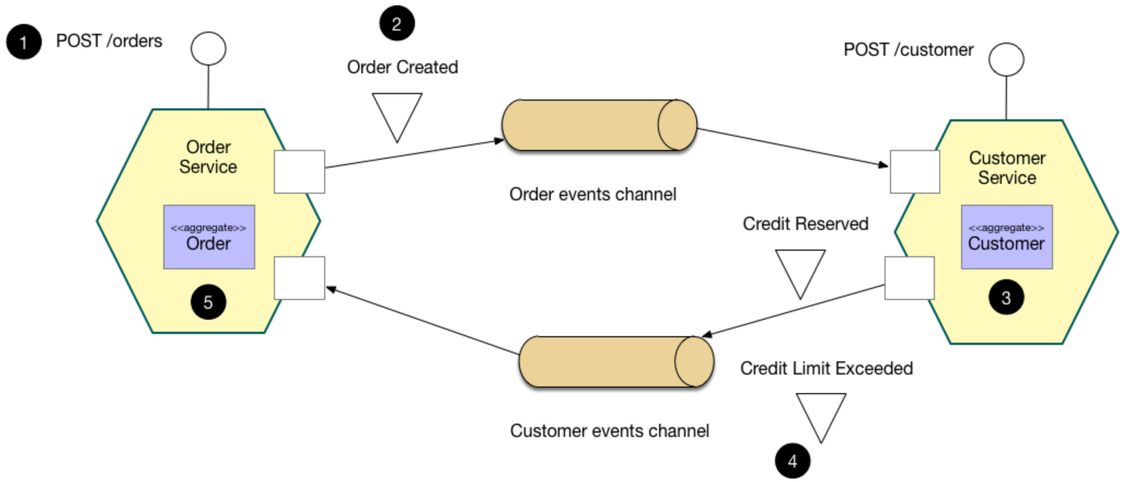


3.4 Saga

DEVIEW 2021

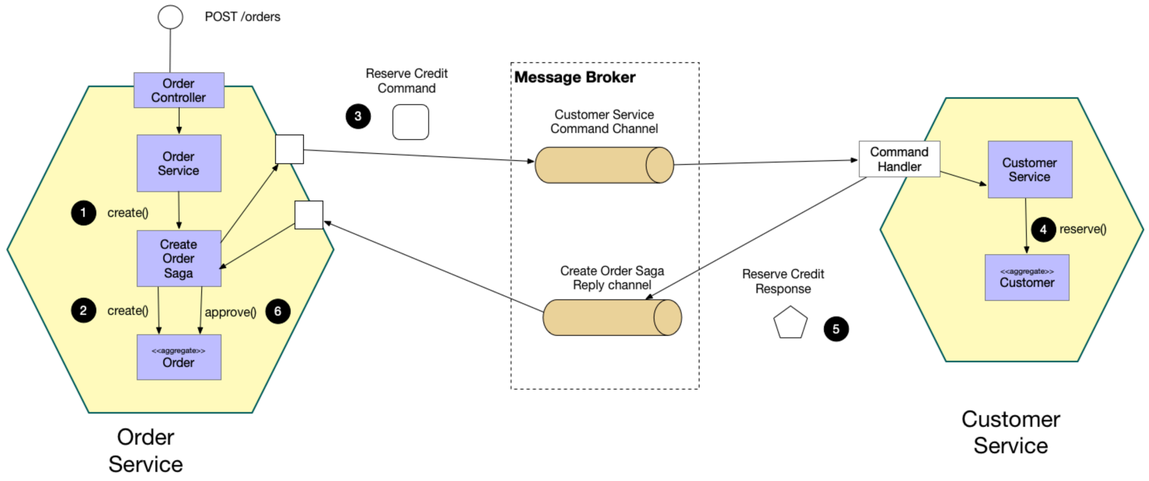
Choreography-based

- 각 local transaction이 타 서비스의 local transaction trigger
- 적은 서비스 및 간단한 연동에 적합
- 구현이 비교적 쉽지만, 트랜잭션 추적이 어려움



Orchestration-based

- Local transactions을 중앙에서 관리하는 Orchestrator가 존재
- 많은 서비스가 포함된 복잡한 연동에 적합
- Transaction의 중앙 집중화 But 복잡해질 수 있음

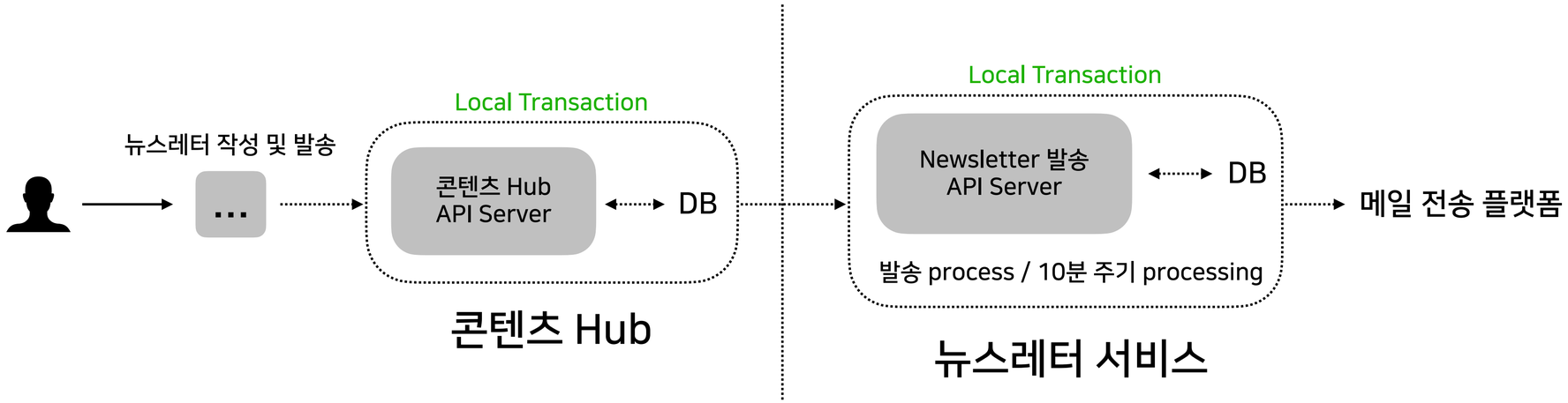


https://microservices.io/patterns/data/saga.html

3.4 Saga

적용 사례 - 콘텐츠 뉴스레터 발송

- 뉴스레터 서비스는 요청 정보를 DB에 먼저 쌓고, 10분 주기 배치로 발송
- 비동기 실행이기에, 발송 완료 응답은 즉각 받을 수 없음
- Data Consistency가 지켜져야 함
  - e.g.) 실제 발송이 완료되었음에도 '발송 중' 상태를 가지면 안 됨
  - e.g.) 실제 발송이 실패했음에도 '발송 중' 상태를 가지면 안 됨



3.4 Saga

요청 및 응답 유실 최소화

Data Consistency를 지원

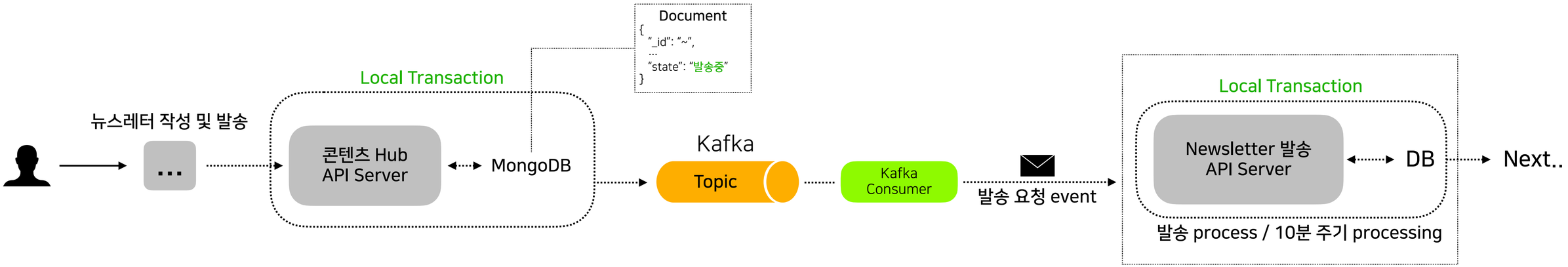
적은 서비스 간 연동

Choreography-based

3.4 Saga

콘텐츠 뉴스레터 발송 요청

- 서비스 간 long term 비동기 연동

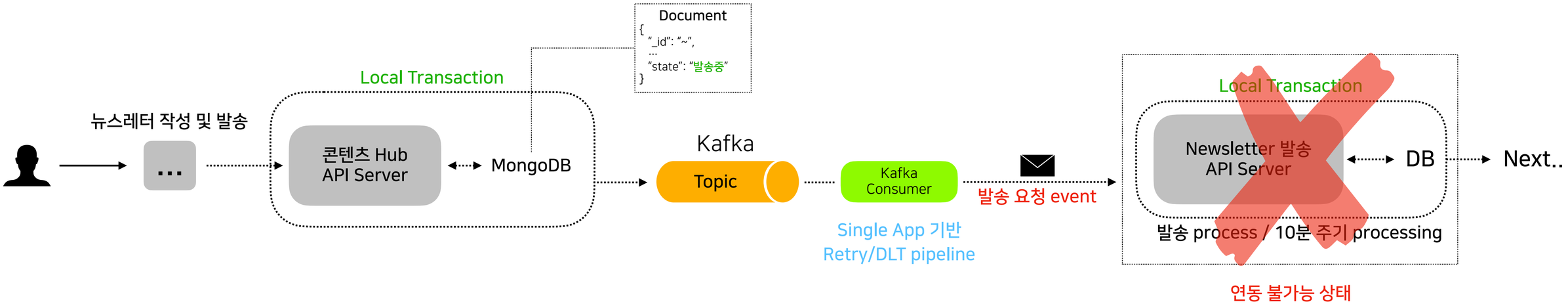


3.4 Saga

DEVIEW 2021

콘텐츠 뉴스레터 발송 요청 실패 case

- Newsletter server는 rest/http endpoint만 지원
- Retry/DLT pipleline 적용
- 서비스 연동 불가 시 Retry/DLT topic으로 pipe
- 연동 가능 시점에 재처리



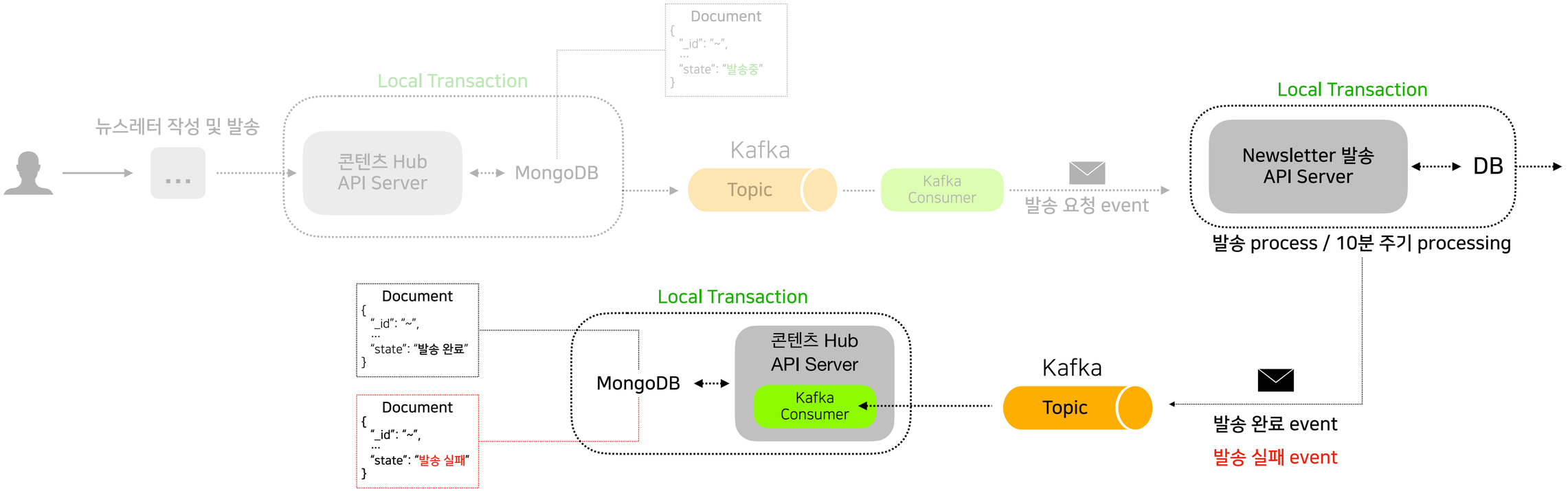
장애, 배포, 인프라 이슈에도 요청 유실 최소화

3.4 Saga

DEVIEW 2021

콘텐츠 뉴스레터 발송 응답 처리

- 서비스 간 long term 비동기 연동

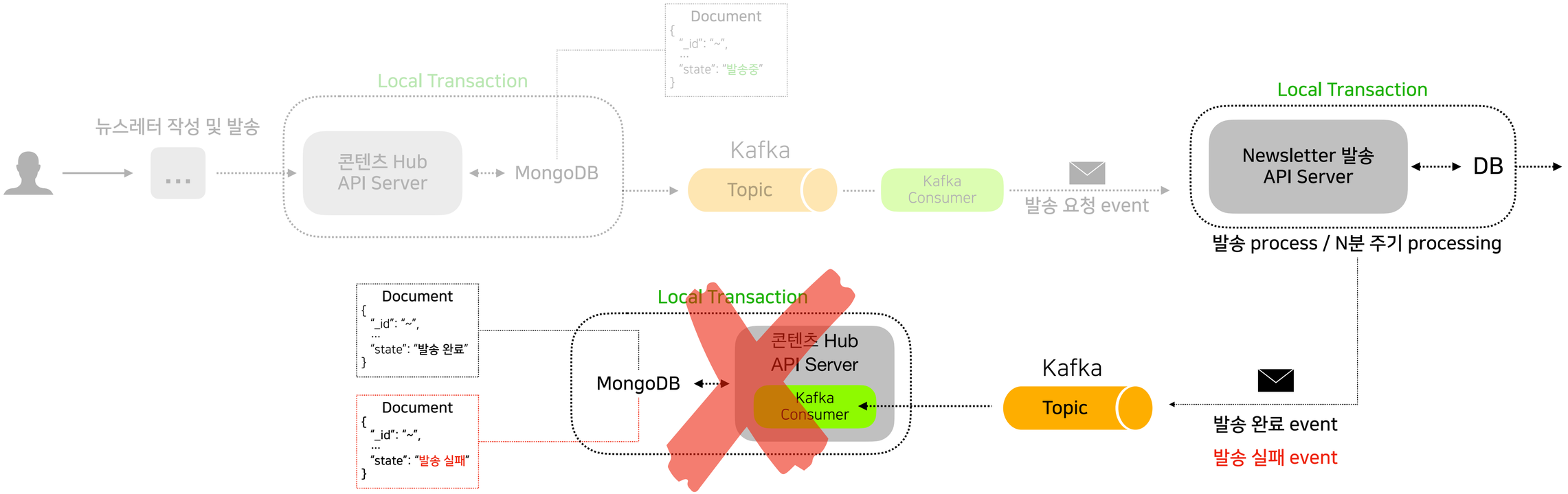


3.4 Saga

DEVIEW 2021

콘텐츠 뉴스레터 발송 응답 처리 실패 case

- Server 장애 시 consumer down / group에서 제거
- 복구 시 이전 offset 기준 자동 재처리 진행



3.4 Saga

DEVIEW 2021

Always Saga?

예시처럼 Service 간 연동에 지연이 존재하는 경우만 적용

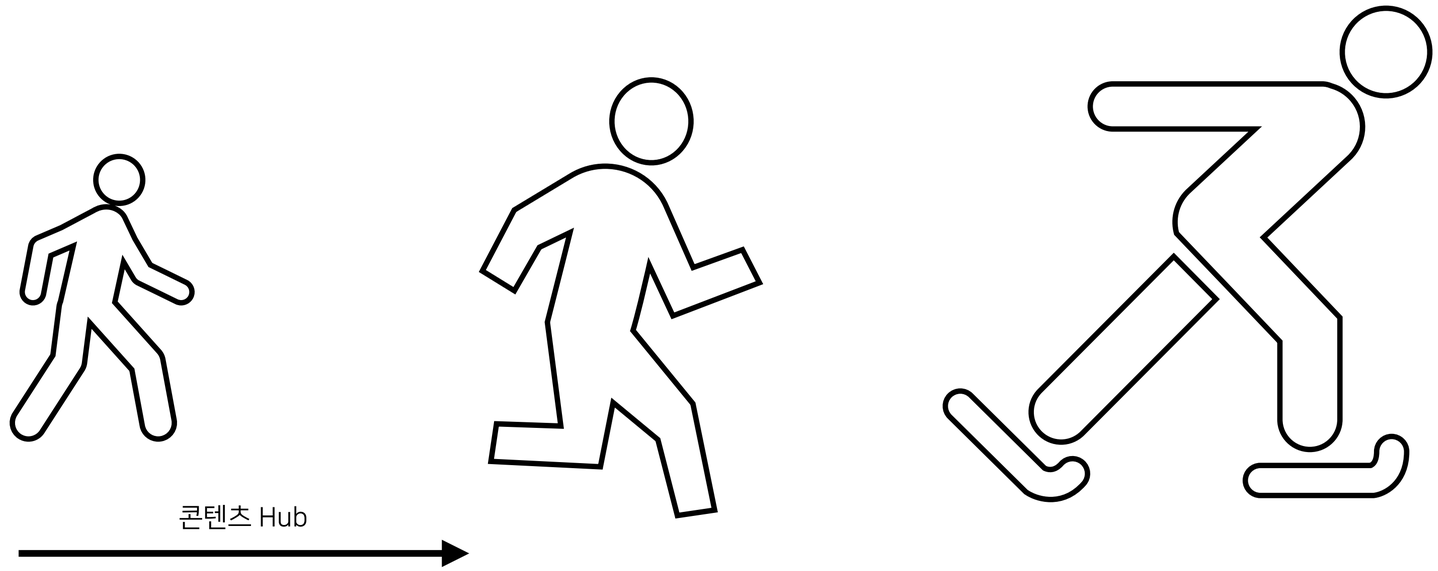
현재 대부분의 경우 불필요

각 상황별 특징에 따라 구성하는 것이 중요

4. 향후 계획

4.1 콘텐츠 Hub는 성장 ing

- Change Data Capture
- Kafka Consumer Monitoring
- MongoDB 복구 flow
- ...



Thank You