(곧) 백엔드 개발자의 What Why How

얕게 알아보는 메시징 시스템

발표에서 얻을 수 있는 것

- (응애) 새내기

- (곧) 프론트앤드 개발자

- (곧) 백엔드 개발자

(곧) 개발자의 새내기 시절

(응애 새내기)

개발자가 되겠어!

출처: 구글이미지

가장 먼저,

SAMSUNG SDS if(kakao)dev

생각 하는 방식을 바꿔야 한다 

What Why How

- 직면한 문제
- A 기술을 사용/개발한 이유
- A 기술이 무엇인지
- 어떻게 해당 기술이 동작하는지

물론, ~~이상적인 생각 방식~~

출처: 구글이미지

목표 설정하고
공부하는 데
적용시켜 봄

(곧) 백엔드 개발자

채용공고 👀

- 대용량 트래픽 / 데이터 처리할 수 있는 서비스

- Java / Kotlin
- Spring Framework
- Kafka, MQ 등 메시지 플랫폼

- Spring Cloud
- MSA

제 관심사와 목표와 관련해서

Kafka, MQ 등
메시징 시스템이 무엇인지

왜 해당 기술을 사용하는지

어떻게 동작 & 사용하는지

얕게 살펴보는 # 메시징 시스템

Apache kafka

메시징

- 메시지를 서로 **비동기적**으로 주고 받는 통신 방식

Ex) *메시지 브로커*가 있는 아키텍처, *Pub-Sub*기반의 채널

메시지 브로커

- 모든 메시지가 지나는 인프라 서비스
- 송신자의 메시지 프로토콜 형식으로부터의 메시지를 수신자의 메시지 프로토콜 형식으로 변환
- 메시지 채널 구현방식은 메시지 브로커마다 다름
  - 메시지 프로토콜 : AMQP, JMS
  - 메시징 방식 : Point – to – Point , Pub/Sub

익스체인지, 큐

토픽

큐, 토픽

큐

스트림

(브로커 기반)

메시징 시스템 왜 사용하는가?

- 대용량 트래픽 / 데이터 처리할 수 있는 서비스

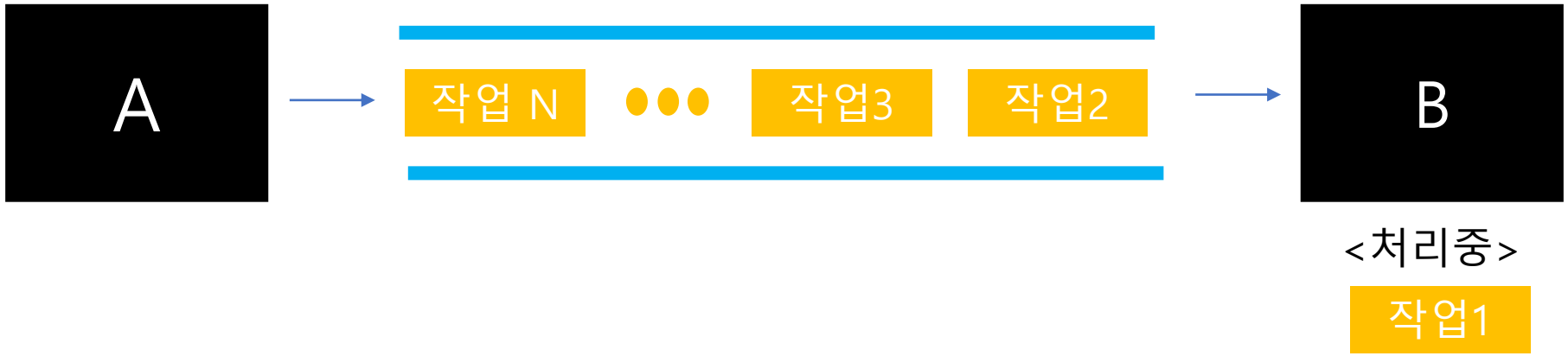
- Java / Kotlin
- Spring Framework
- Kafka, MQ 등 메시지 플랫폼

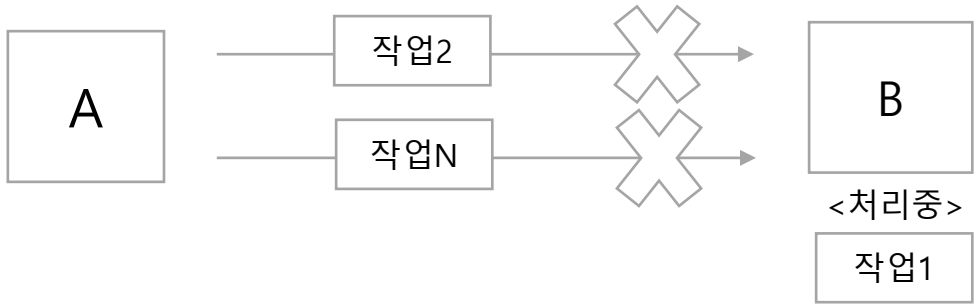
- Spring Cloud
- MSA

- 서비스 간의 비동기 통신이 가능 (결합도 낮아짐)
  - 적절한 채널에 메시지를 보내기만 하면 됨.
  - 해당 메시지를 어떤 위치의 서비스 인스턴스가 사용하는지 몰라도 됨.

- *처리 가능한 시점까지 메시지 버퍼링 가능*
- *손쉽게 확장 가능*
  - 고려해야 할 사항

EX) 처리 가능한 시점까지 메시지 버퍼링

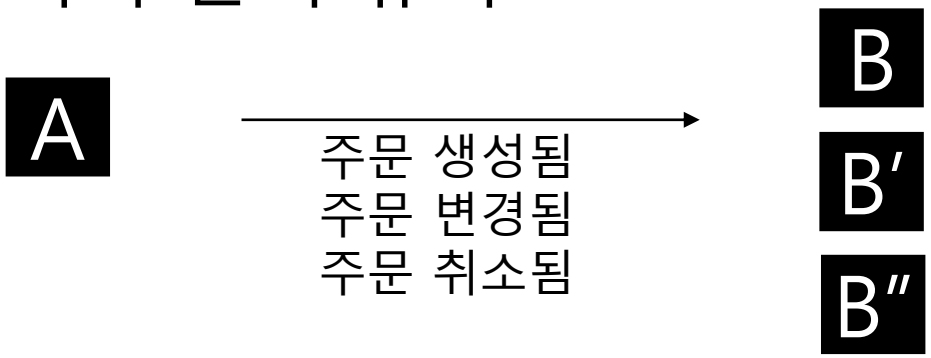




EX) 확장하는 경우 고려해야 할 사항

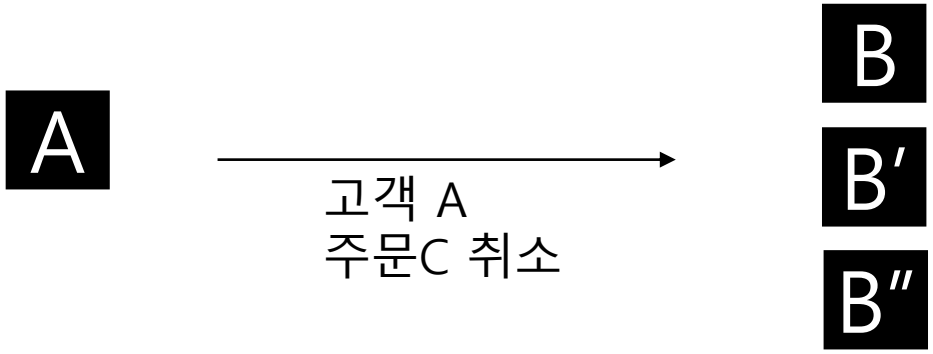
손쉽게 확장 가능

- 메시지 순서 유지



동일한 주문인지 판단해서,
이들은 순서대로 처리되도록

- 중복 메시지 걸러내기



환불이 한 번만 되야 함
고객 A의
주문 C 취소에 대한
처리는 한 번만 일어나도록

What Why는 앞에서 알아봄

그렇다면, “How : 어떻게 사용하는지” Apache Kafka로 살펴보자

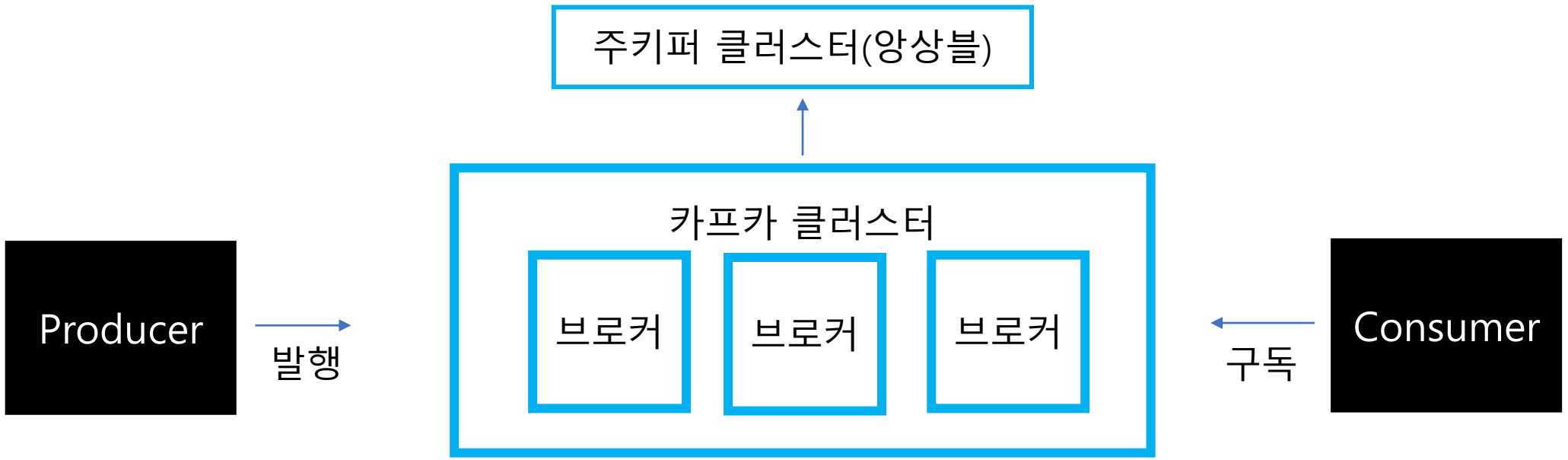
Apache Kafka : 배경 & 특징

- Linkedin이 “실시간 데이터 피드를 관리하기 위해” “통일된, 높은 처리량, 낮은 지연시간을 플랫폼” 개발
- 2011년 초 오픈소스화 (Github 링크)

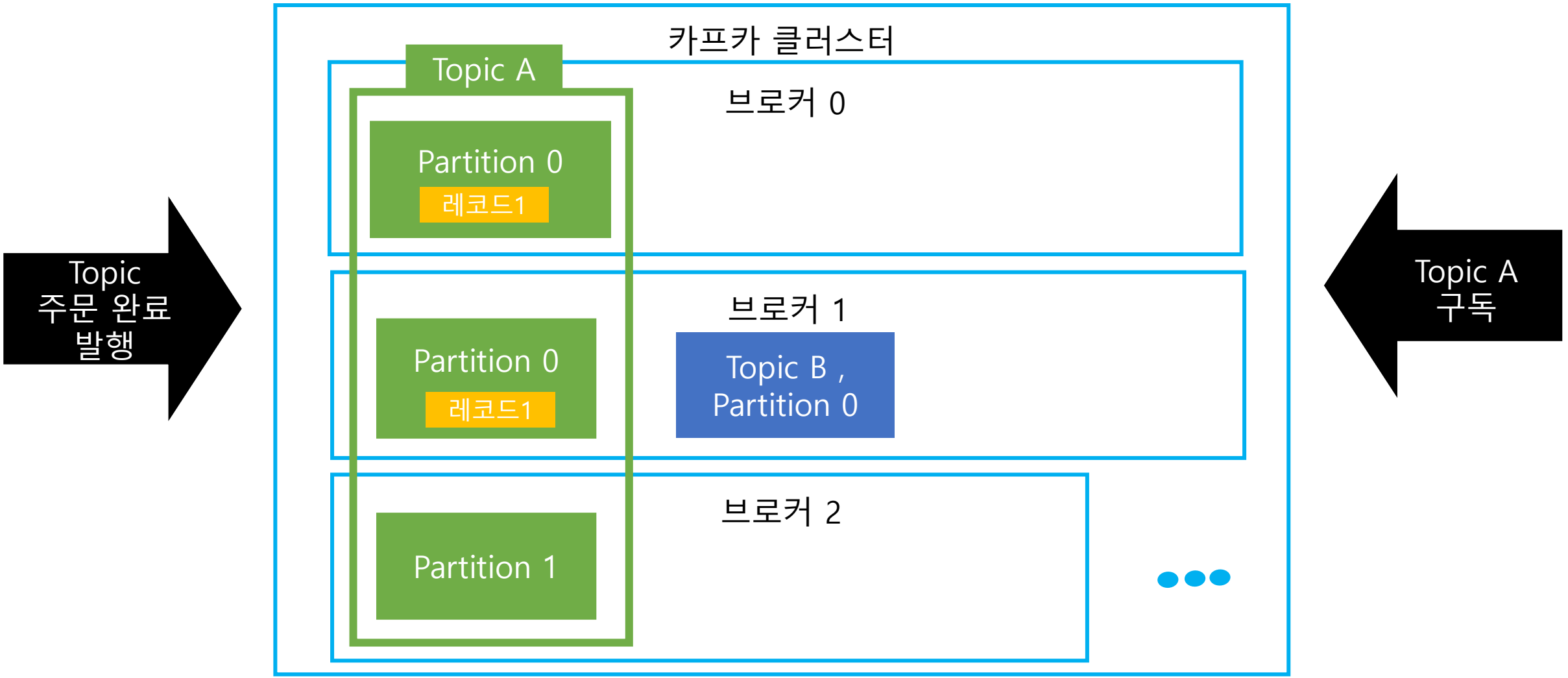
- 메시지 브로커 (처리 후, 즉시, 빠른 시간 내 삭제)

- 이벤트를 하나만 두고, 인덱스를 통해 관리
- 이벤트를 보존할 수 있음
  - 장애가 일어난 부분부터 재처리할 수 있음
  - 이벤트 기반 마이크로서비스 아키텍처

Apache Kafka : 기본 구조



Apache Kafka : 토픽 & 파티션



실습 : docker 이미지 다운로드

```
$ docker-compose version

$ docker-compose –f docker-compose.yml up –d

$ docker ps
```

```
CONTAINER ID   IMAGE                              COMMAND                  CREATED         STATUS         PORTS                                          NAMES
69ea03cb786a   confluentinc/cp-kafka:latest       "/etc/confluent/dock…"   5 minutes ago   Up 5 minutes   9092/tcp, 0.0.0.0:29092->29092/tcp             kafka-ex-kafka-1
17e189b714f2   confluentinc/cp-zookeeper:latest   "/etc/confluent/dock…"   5 minutes ago   Up 5 minutes   2888/tcp, 3888/tcp, 0.0.0.0:22181->2181/tcp    kafka-ex-zookeeper-1
```

실습 : topic 생성 (topic-a)

(1) Topic 생성

```
$ docker-compose exec kafka kafka-topics --create --topic topic-a
--bootstrap-server kafka:9092 --replication-factor 1 --partitions 1
```

```
Created topic topic-a.
```

topic-a 생성 확인

```
$ docker-compose exec kafka kafka-topics --describe --topic topic-a
 --bootstrap-server kafka:9092
```

```
Topic: topic-a  TopicId: uK_2dmnHS9iRK-bkLBAR1Q PartitionCount: 1       ReplicationFactor: 1    Configs:
        Topic: topic-a  Partition: 0    Leader: 1       Replicas: 1     Isr: 1
```

실습 : 컨슈머/프로듀서 실행하기

```
$ docker-compose exec kafka bash
```

(2) 컨슈머 실행

```
[appuser@]$ kafka-console-consumer --topic topic-a
--bootstrap-server kafka:9092
```

(3) 프로듀서 실행

```
[appuser@]$ kafka-console-producer --topic topic-a
--broker-list kafka:9092
```

실습 : 결과 확인

(4) 메시지 발행 후, 확인

```
[appuser@69ea03cb786a ~]$ kafka-console-producer --topic topic-a --broker-list kafka:9092
>Hello
>From Producer
>Topic A 입니다
>
```

```
[appuser@69ea03cb786a ~]$ kafka-console-consumer --topic topic-a --bootstrap-server kafka:9092
Hello
From Producer
Topic A 입니다
```

```
[appuser@69ea03cb786a ~]$ kafka-console-consumer --topic topic-b --bootstrap-server kafka:9092
```

출처

- 트래픽 많은 회사는 다 카프카를 씁니다.

https://www.youtube.com/watch?v=PJMvKJrkLpE

- 메시지 브로커

https://www.ibm.com/kr-ko/cloud/learn/message-brokers

- 아파치 카프카 애플리케이션 프로그래밍
- 마이크로서비스 패턴
- Docker Compose 를 이용하여 Single Broker 구성하기

https://devocean.sk.com/blog/techBoardDetail.do?ID=164007

감사합니다.