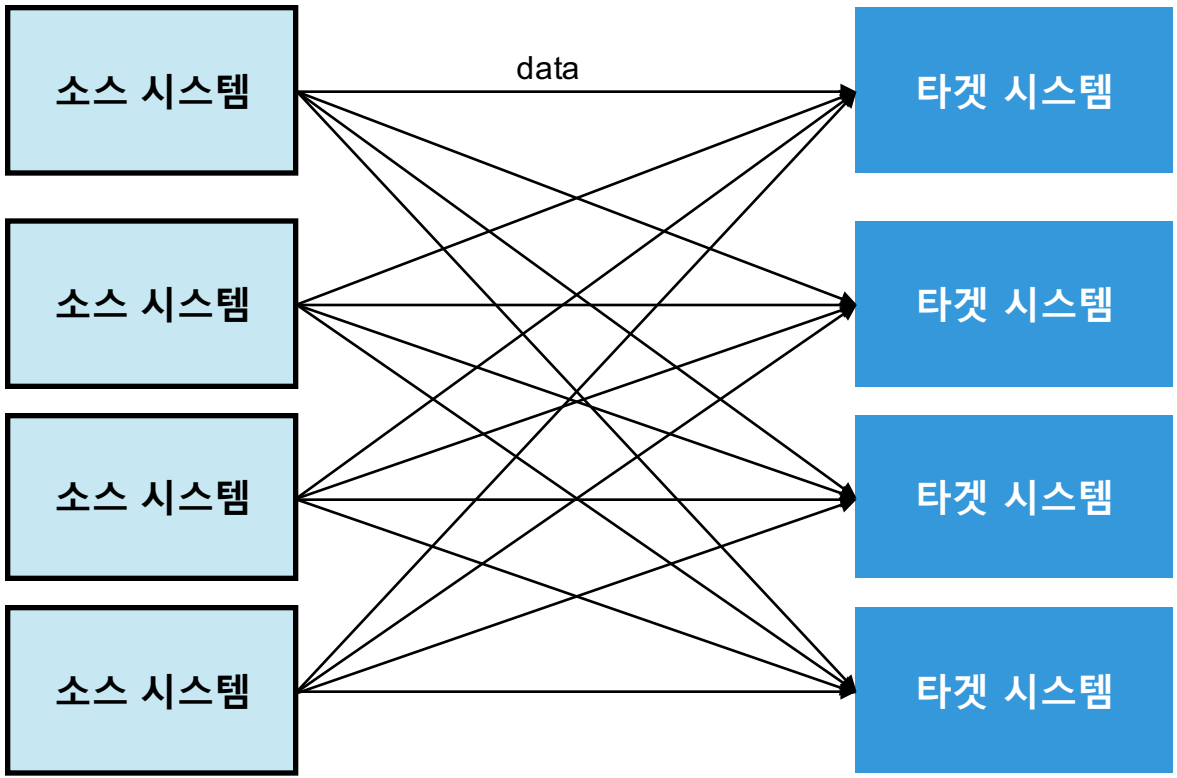
Message Queue (Kafka)

uEngine

Table of content

카프카의 탄생 배경

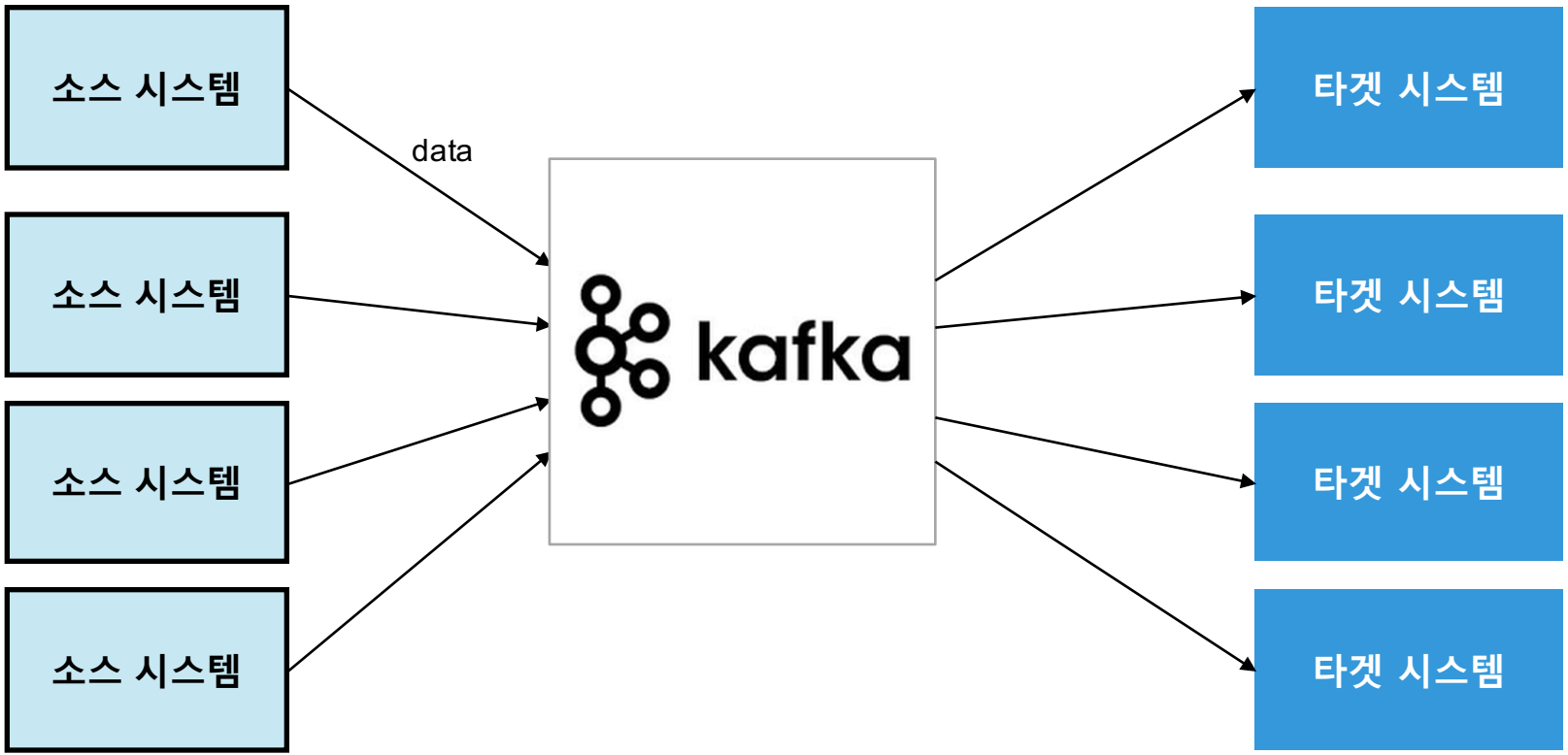


- 소스 시스템과 타겟시스템이 많아지면서 복잡

- 어려운 이유?
- 프로토콜(http, TCP, RPC, FTP, JDBC ..)
- 데이터 포맷(json, binary, csv, xml)
- 데이터 스키마 다양함 (각 시스템별로)

- 각각의 시스템이 데이터 포맷과 처리하는 방식이 달라서 확장 및 통합 이 어려워짐

카프카의 탄생 배경



카프카의 탄생 배경

- 데이터 스트림과 시스템을 구분
- 모든 데이터/이벤트를 카프카를 중심으로 연결
- 소스시스템에서는 카프카가 제공하는 표준 포맷으로 데이터를 전송(Pub)만 하고 타겟시스템에서는 데이터를 수신(Sub) 하도록 구성

Apache Kafka 를 왜 쓸까요?

- 2010 년에 링크드인에서 개발
- 2012 년에 아파치 오픈소스로 등록됨
- 2014 년에 링크드인 개발자들이 나와서 컨플루언트라는 회사를 창립해 카프카를 계속 발전시킴
- 확장 (Distributed) , 유연 (Resilient), 장애에 강한 (fault tolerant) 아키텍처
- 수평 확장 가능
  - 100 개의 Broker 까지 확장
  - 1초에 100만개의 메세지
  - 무중단 확장 가능
- 높은 퍼포먼스 (10ms 미만의 지연) – real time

Apache Kafka 를 어떻게 쓸까요?

- 메세징 시스템
- Activity Tracking
- 모니터링 데이터 수집 (Gather metrics)
- 로그 수집 (Log Aggregation)
- Stream Processing
- 시스템간의 종속성 분리 (De-coupling of System Dependencies)
- Spark, Storm, Flink, Hadoop 등 많은 빅데이터 기술과 통합하여 사용

- https://kafka.apache.org/uses

Apache Kafka 를 어떻게 쓸까요?

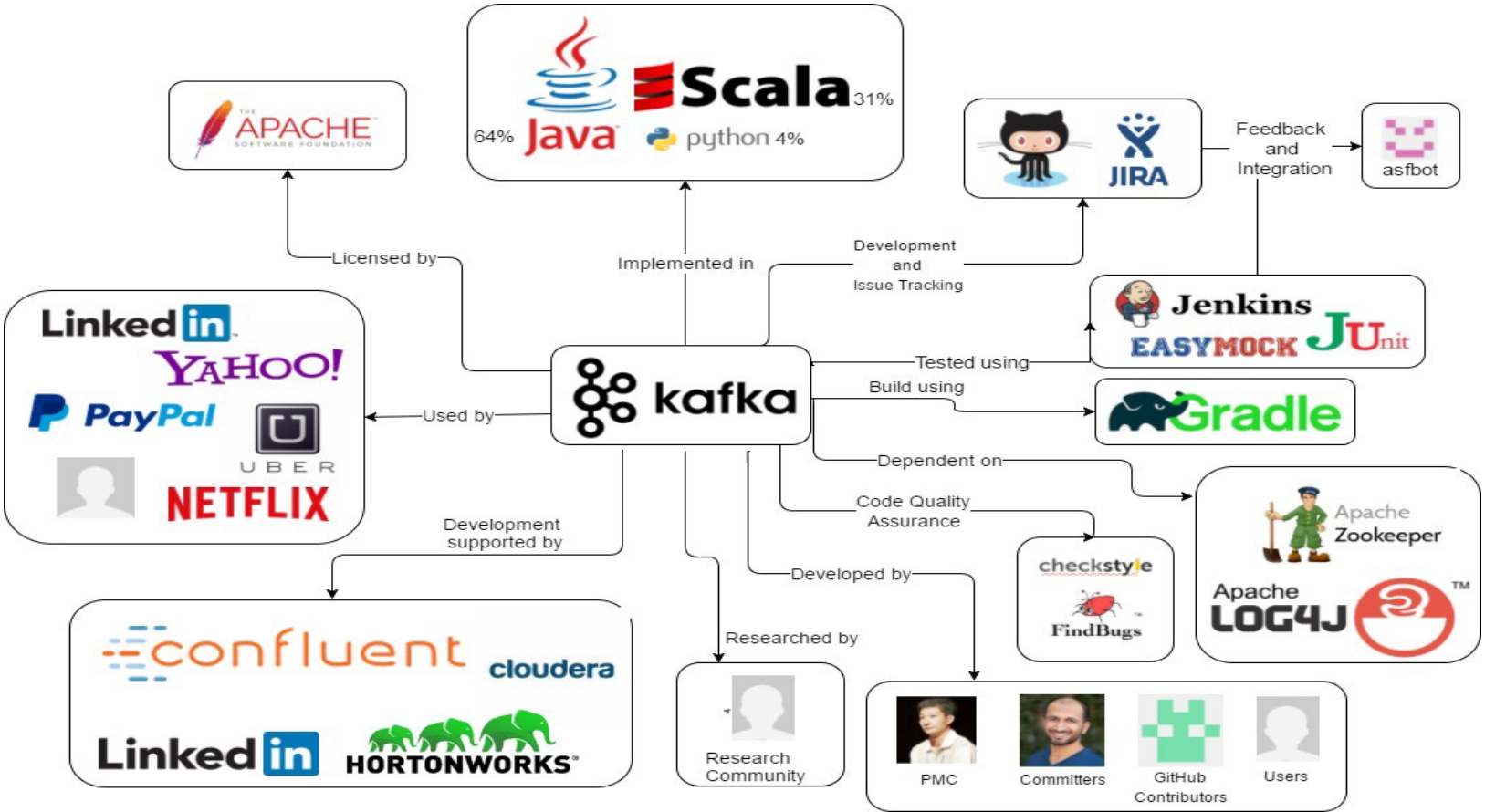


Table of content

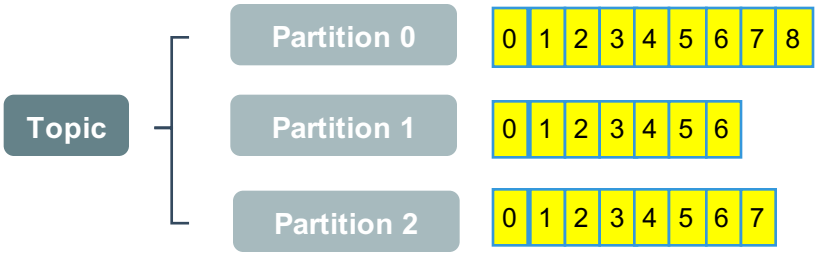
Topics, Partitions, Offsets

- 토픽
  - 특정 데이터 스트림, 즉 메세지를 구분 하는 통로
  - 하나의 카프카에 여러 메세지가 뒤섞여서 오가는데, 거기서 토픽으로 구분하여 원하는 메세지를 찾는 방식
  - 데이터베이스의 테이블, 카톡의 단톡방 과 비슷한 개념
  - 토픽은 여러개 만들수 있고, 이름으로 구분됨

  - 토픽 이름을 어떻게 만들어야 하나?
  - https://riccomini.name/how-paint-bike-shed-kafka-topic-naming-conventions
  - https://devshawn.com/blog/apache-kafka-topic-naming-conventions/
  - <project>.<product>.<event-name>
  - services, consumers, or producers 등과 연결하여 만들지 않기를 추천함

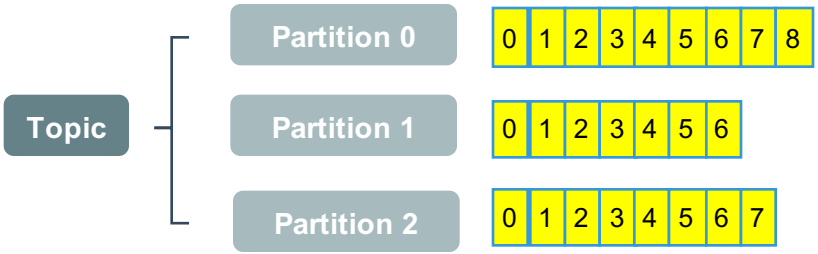
Topics, Partitions, Offsets

- 파티션
  - 토픽은 파티션을 나눌수 있음.
  - 토픽당 데이터를 분산 처리하는 단위
  - 병렬 처리와 많은 양의 데이터 처리를 위해서 파티션을 늘리수 있음
  - 늘리기만 하고 줄이는 것은 안됨
  - 각각의 파티션은 0부터 1,2 이렇게 증가하게됨
  - 파티션 별로 증가되는 아이디 , 즉 offset 라는 값을 가지게 됨



Topics, Partitions, Offsets

- 토픽과 파티션
  - offset 은 파티션에서만 의미가 있음.
    - 파티션 3개가 있다면 각각의 파티션의 offset 0번은 은 다른 데이터를 가지게됨
  - 파티션 안에서는 데이터의 순서가 보장
    - 파티션이 여러개면 1번과 2번 파티션에 데이터가 어느게 먼저 올지 알수 없음
  - 파티션에 데이터가 한번 쓰여지면 변경이 안됨
  - key 를 주지 않으면 어느 파티션에 데이터가 들어갈지 모름

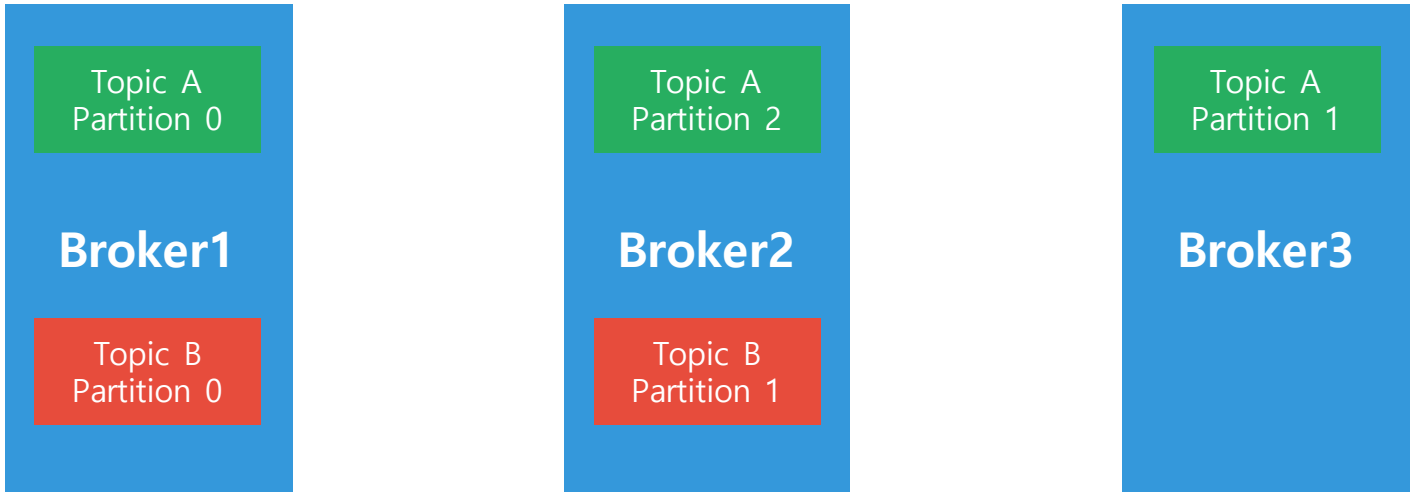


Broker

- 카프카를 클러스터로 구성하였을때 각각의 서버를 Broker 라고 부름
- 각각의 브로커는 토픽 파티션을 가짐
- 어떤 브로커에 접속해도 전체 카프카 클러스터에 접속 가능
- 브로커는 3개로 시작하여 100개까지 늘어날 수 있음

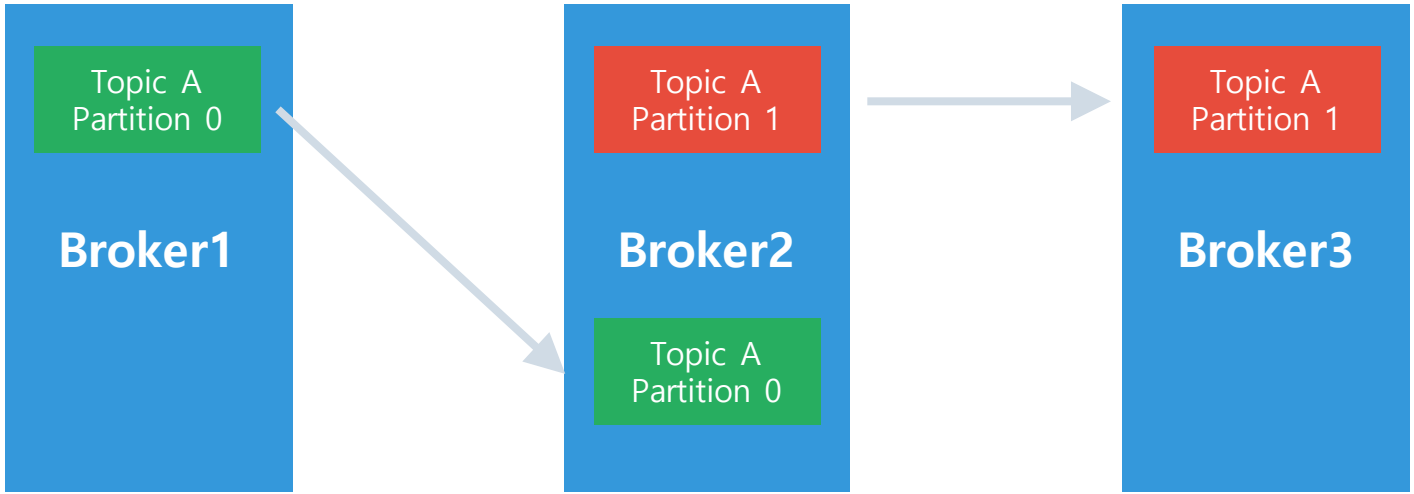
Broker

- 만약 Topic A 의 파티션이 3 개라면?
- 만약 Topic B 의 파티션이 2 개라면?
  - Broker 3 은 Topic B 에 대한 데이터를 가지고 있지 않다.



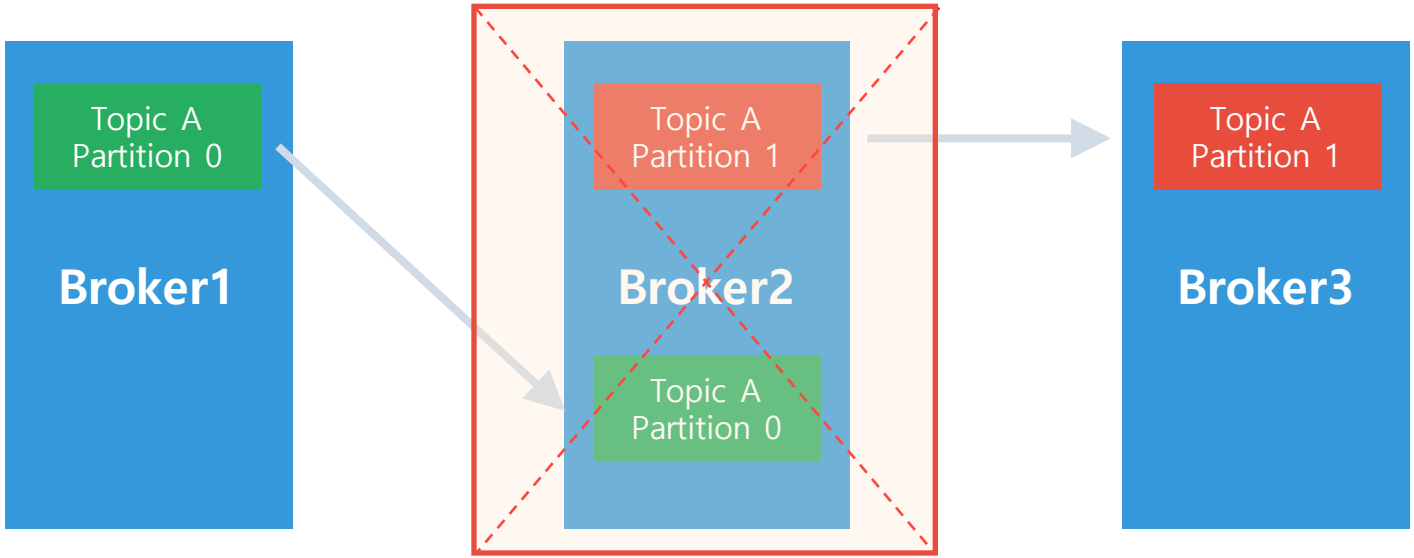
Topic replication factor

- 토픽을 이루는 파티션을 복제하는 기능
- Topic A 의 파티션이 2 개, replication factor 가 2 라면?
- replication factor 는 2 이상 하는 것이 좋음 (중요데이터는 3)



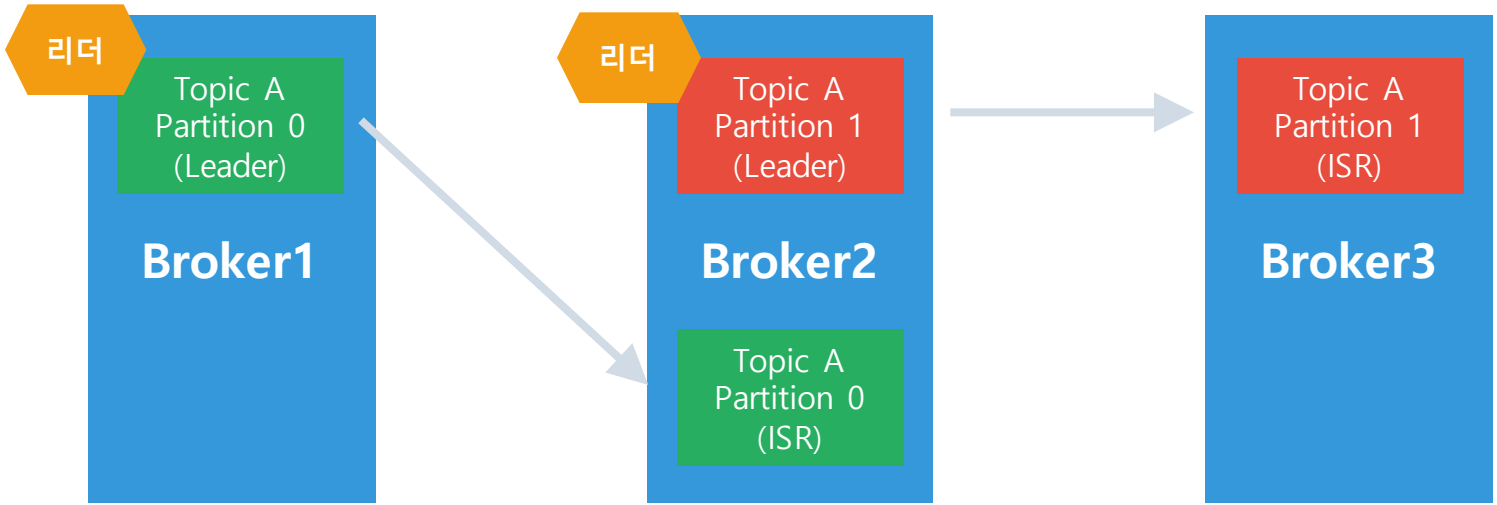
Topic replication factor

- 만약 Broker 2 가 장애가 발생해도 토픽에 정상 데이터가 수신됨
- 만약 토픽의 데이터가 3GB 면, replication factor 를 3으로 주었을때 9GB 의 데이터가 됨



파티션의 Leader

- 파티션의 리더만 데이터를 읽고, 쓰기가 가능
- 나머지 파티션은 ISR(In Sync Replica) 역할을 하게됨
- ISR 은 리더의 데이터를 체크하여 복제하고, 리더가 장애시 리더역할을 함



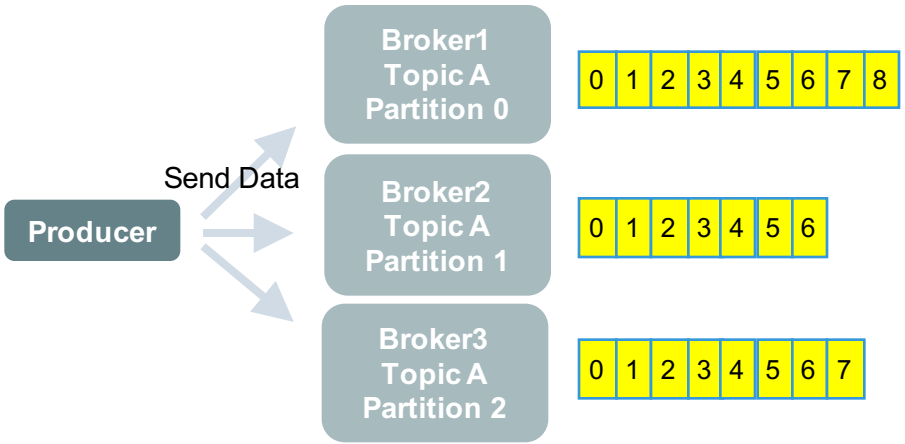
클러스터에서 Leader 재조정 – broker-0 삭제시

```
Topic:f9Topic   PartitionCount:9        ReplicationFactor:3     Configs:
        Topic: f9Topic  Partition: 0    Leader: 0       Replicas: 0,1,2 Isr: 1,2,0
        Topic: f9Topic  Partition: 1    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: f9Topic  Partition: 2    Leader: 2       Replicas: 2,0,1 Isr: 1,2,0
        Topic: f9Topic  Partition: 3    Leader: 0       Replicas: 0,2,1 Isr: 1,2,0
        Topic: f9Topic  Partition: 4    Leader: 1       Replicas: 1,0,2 Isr: 1,2,0
        Topic: f9Topic  Partition: 5    Leader: 2       Replicas: 2,1,0 Isr: 1,2,0
        Topic: f9Topic  Partition: 6    Leader: 0       Replicas: 0,1,2 Isr: 1,2,0
        Topic: f9Topic  Partition: 7    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: f9Topic  Partition: 8    Leader: 2       Replicas: 2,0,1 Isr: 1,2,0
```

```
Topic:f9Topic   PartitionCount:9        ReplicationFactor:3     Configs:
        Topic: f9Topic  Partition: 0    Leader: 1       Replicas: 0,1,2 Isr: 1,2,0
        Topic: f9Topic  Partition: 1    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: f9Topic  Partition: 2    Leader: 2       Replicas: 2,0,1 Isr: 1,2,0
        Topic: f9Topic  Partition: 3    Leader: 2       Replicas: 0,2,1 Isr: 1,2,0
        Topic: f9Topic  Partition: 4    Leader: 1       Replicas: 1,0,2 Isr: 1,2,0
        Topic: f9Topic  Partition: 5    Leader: 2       Replicas: 2,1,0 Isr: 1,2,0
        Topic: f9Topic  Partition: 6    Leader: 1       Replicas: 0,1,2 Isr: 1,2,0
        Topic: f9Topic  Partition: 7    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: f9Topic  Partition: 8    Leader: 2       Replicas: 2,0,1 Isr: 1,2,0
```

Producer

- 메세지를 생산(produce) 하여 토픽으로 메세지를 보내는 애플리케이션, 서버등을 Producer 라고 부른다.
- 메세지를 전송할때 Key 옵션을 줄 수있는데, Key 값을 설정하여 특정 파티션에 메세지를 보낼수 있다.
- Key 옵션을 주지 않을때는 파티션에 라운드로빈 방식으로 균등하게 메세지를 보낸다.

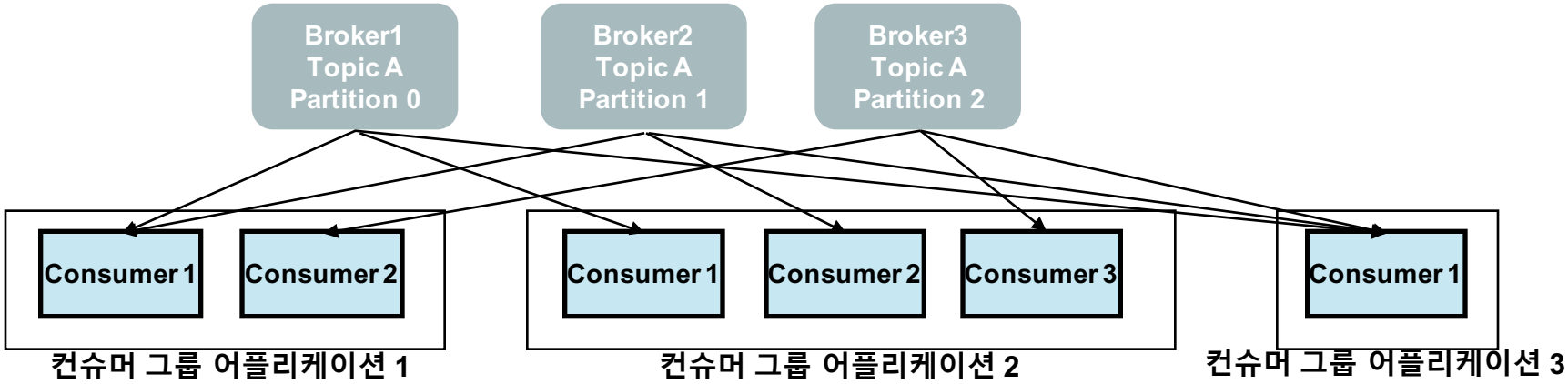


Producer

- 메세지 손실 가능성이 높지만 빠른 전송이 필요한 경우
  - acks=0 프로듀서는 응답을 기다리지 않는다.
- 메세지 손실 가능성이 적고, 적당한 속도의 전송이 필요한 경우
  - acks=1 프로듀서는 리더의 응답만 체크 한다. (default)
- 전송 속도는 느리지만 메세지 손실이 없어야 하는 경우
  - acks=all 프로듀서는 리더와 ISR 의 모든 응답을 체크한다.
  - Replication 이 없는 경우라면 acks=1 과 동일하다.

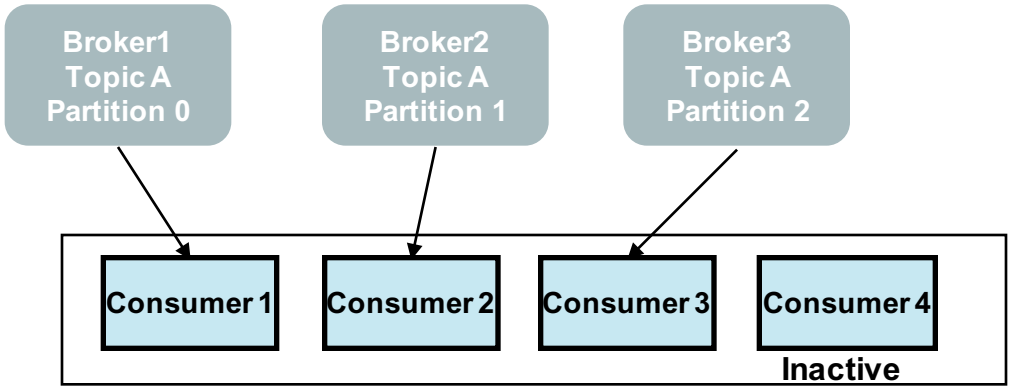
Consumer

- 토픽 이름으로 저장된 메세지를 가져가는 애플리케이션, 서버등을 Consumer 라고 부른다.
- Consumer 들의 집합을 Consumer Group 이라고 부르며, 카프카는 Consumer Group 단위로 데이터를 처리한다.
- Consumer Group 안의 각각의 Consumer 는 파티션의 데이터를 읽는다.



Consumer

- 만약 Consumer 갯수가 파티션보다 많으면, 매칭되지 않은 Consumer 는 작동 안하게 된다.
- 가장 좋은 방법은 파티션 수와 컨슈머 수를 일치 시키는 것이다.



Consumer Offset

- 카프카는 컨슈머 그룹이 어디까지 메세지를 가져갔는지 Offset 을 체크한다.
- 컨슈머 그룹은 메세지를 가져간 후에 offset 을 Commit 한다.
- 카프카는 내부적으로 별도의 토픽으로 커밋된 정보를 저장한다.
  __consumer_offset 형식으로 저장됨
- 만약 컨슈머가 죽었을때, 해당 offset 정보를 읽어서 어디부터 데이터를 읽을지 파악하고, offset 다음부터 메세지를 수신한다.

Zookeeper

- 분산 애플리케이션 코디네이터
- 주키퍼는 브로커를 관리한다.
- 파티션의 리더 선출을 도와준다.
- 카프카의 변경에 대하여 알림을 준다 (토픽생성, 브로커 die, 브로커 up, 토픽 삭제)
- **카프카는 주키퍼 없이 동작 못함**
- 주키퍼 클러스터는 홀수로 동작함 (leader/follower 선출때문에 짝수로 동작 못함)
- 주키퍼 leader 는 write , follower 는 read 작업을 함

Table of content

카프카 설치

- 아래 링크에 가서 카프카 설치와 실행
- http://msaschool.io/#/%EC%84%A4%EA%B3%84--%EA%B5%AC%ED%98%84--%EC%9A%B4%EC%98%81%EB%8B%A8%EA%B3%84/04_%EA%B5%AC%ED%98%84/10_%EC%9D%B4%EB%B2%A4%ED%8A%B8%EA%B8%B0%EB%B0%98%20%EB%A9%94%EC%84%B8%EC%A7%80%20%EC%B1%84%EB%84%90

Table of content

CLI - topic

- kafka-topics
- kafka-topics --zookeeper 127.0.0.1:2181 --list
- kafka-topics --bootstrap-server 127.0.0.1:9092 --list
- kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --create
- kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --create --partitions 3
- kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --create --partitions 3 --replication-factor 2
- kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --create --partitions 3 --replication-factor 1
- kafka-topics --zookeeper 127.0.0.1:2181 --list
- kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --describe
- kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --delete
- (현재 버전에서는 윈도우에서 토픽 삭제시 에러가 발생 – 지우지 마세요)

이전 버전의 kafka 에서는 --zookeeper 로 호출하였으나 최신버전에서는 --bootstrap-server 로 호출

CLI - producer

- kafka-console-producer
- 만약 토픽 삭제 하였다면 토픽 다시 생성
  - kafka-topics --zookeeper 127.0.0.1:2181 --topic first_topic --create --partitions 3 --replication-factor 1
- kafka-console-producer --bootstrap-server 127.0.0.1:9092 --topic first_topic
- kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic --producer-property acks=all
- kafka-console-producer --broker-list 127.0.0.1:9092 --topic new_topic
- kafka-topics --zookeeper 127.0.0.1:2181 --list
- kafka-topics --zookeeper 127.0.0.1:2181 --topic new_topic --describe
- vi config/server.properties
  - num.partitions=3 으로 변경
  - 카프카 재시작
- kafka-console-producer --broker-list 127.0.0.1:9092 --topic new_topic_2
- kafka-topics --zookeeper 127.0.0.1:2181 --list
- kafka-topics --zookeeper 127.0.0.1:2181 --topic new_topic_2 --describe

CLI - consumer

- kafka-console-consumer
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic
- ## 새로운 터미널 open
- kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --from-beginning

Consumer 는 새로
추가되는 메세지만 받음

- ## 컨슈머 그룹별로 데이터 수신
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --group order
- ## first_topic 에 새로운 데이터 쓰기
- kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic
- ## 새로운 터미널을 열어서 같은 그룹에 메세지가 1개만 받는것 확인
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --group order
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --group product --from-beginning

파티션이 3이면 3개의
컨슈머 그룹에 나누어서
메세지를 수신함

새로운 컨슈머 그룹으로 조회시 처음에는 모든 메세지 수신, 두번째는 수신된건 안받음

CLI - consumer-groups

- kafka-consumer-groups
- kafka-consumer-groups --bootstrap-server localhost:9092 --list
- kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group order
- kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group product

- ## Lag 은 해당 그룹이 아직 못받은 메세지를 나타낸다.

- ## 컨슈머 그룹을 열어 놓은 상태에서 describe 해보기
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --group order
- kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group order

CLI – offset reset

- kafka-consumer-groups
- kafka-consumer-groups --bootstrap-server localhost:9092 --reset-offsets --to-earliest
- kafka-consumer-groups --bootstrap-server localhost:9092 --reset-offsets --to-earliest --execute
- kafka-consumer-groups --bootstrap-server localhost:9092 --reset-offsets --to-earliest --execute --group product --topic first_topic
- ## 토픽의 특정 그룹의 offset 을 reset 시킨 후 조회
- kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group product
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --group product
- ## 특정 값 만큼만 뒤로 보내기
- kafka-consumer-groups --bootstrap-server localhost:9092 --reset-offsets --shift-by -2 --group product --topic first_topic –execute
- ## [참고] key 값을 기준으로 메세지 publish
- kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic --property parse.key=true --property key.separator=,
- kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --from-beginning --property print.key=true --property key.separator=, --group product

Table of content

Use Producer API

- 샘플 소스코드 다운로드
- git clone https://github.com/kimscott/kafka-example.git

- kafka-producer-basic 프로젝트의 pom.xml 확인
- kafka-clients 디펜던시를 받아야 하지만, spring-kafka 가 해당 라이브러리를 포함함
- [실습]
- kafka-clients 라이브러리를 사용하여 메세지 publish 하기
- spring-kafka 라이브러리를 사용하여 메세지 publish 하기
- Producer Callback 사용
- Producer Key 사용

Use Consumer API

- kafka-consumer-basic 프로젝트
- [실습]
- kafka-clients 라이브러리를 사용하여 메세지 consume 하기
- spring-kafka 라이브러리 - *@KafkaListener* 를 사용하여 메세지 consume 하기
- 커슈머 어플리케이션의 포트를 변경하여 여러개의 서비스를 올린 후, 파티션 재설정 확인
- 특정 파티션의 offset 을 찾아서 (seek) 해당 offset 부터 메세지 가져오기

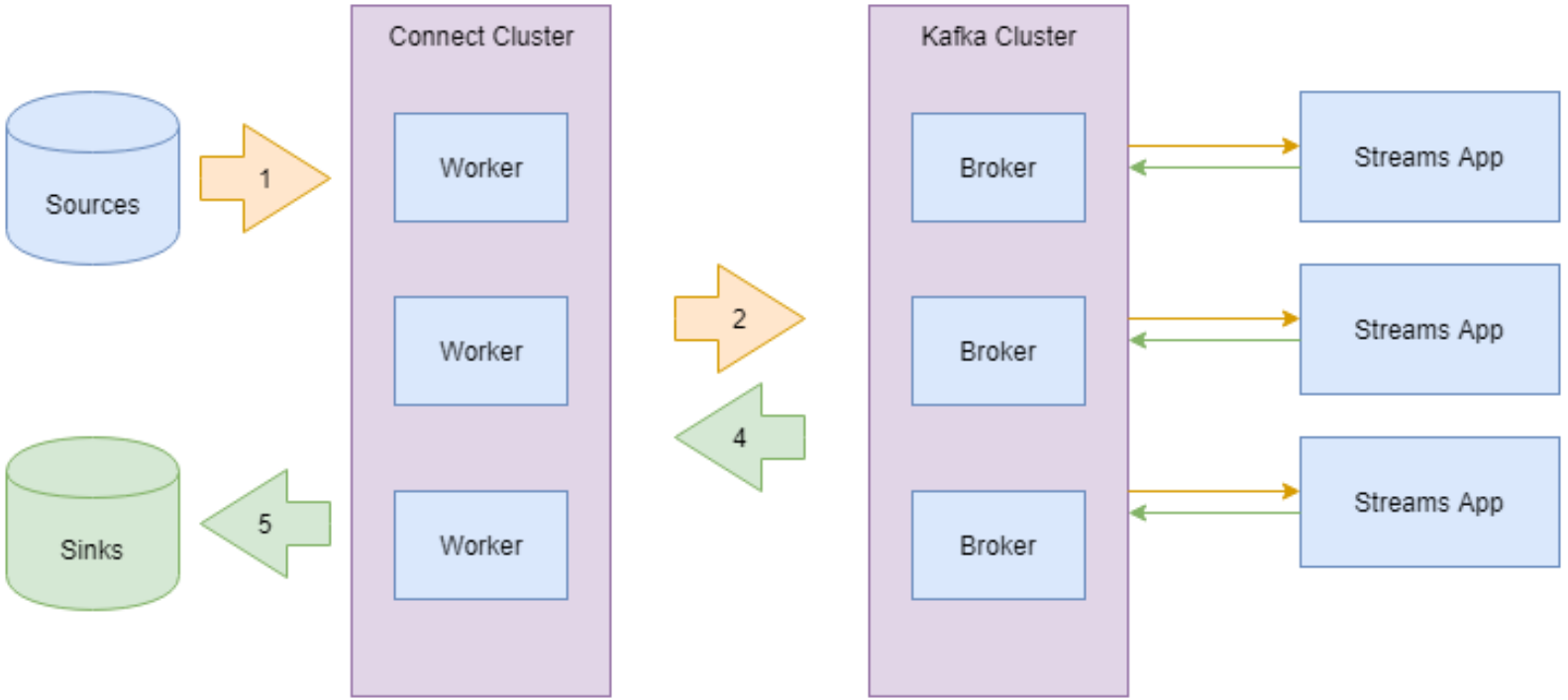
Table of content

Kafka Connect - 나오게 된 이유

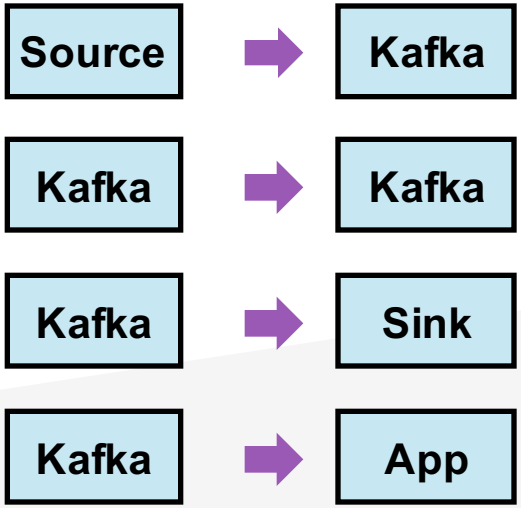
- SNS (twitter , facebook 등) 에서 실시간 데이터를 가져오고 싶은데, 누군가 이미 만들어서 카프카로 보내고 있진 않을까?
- 데이터베이스(MongoDB, PostgreSQL 등) 에서 데이터를 가져와서 처리하고 싶은데, 누군가 이미 만들어서 카프카로 보내고 있진 않을까?
- 카프카의 데이터를 데이터저장소 (S3, HDFS등) 나 검색엔진 (Elastic search 등) 으로 보내고 싶은데, 이 생각을 나만 했을까?
- 카프카 커넥트는 누군가 만든 코드와 커넥터 (connector) 를 재사용 하여 같이 쓰자라는 개념으로 나오게됨.
- Kafka 0.9 버전 (2015 년) 부터 가능

- **다른 소스 데이터로부터 카프카를 통해서 실시간 메세지를 받아오는 방법**
- (예: 트위터 데이터, 스파크, 빅데이터, iot 등등)

Kafka Connect and Kafka Streams



Kafka Connect and Streams



Producer API

Consumer API, Producer API

Consumer API

Consumer API

Kafka Connect Source

Kafka Streams

Kafka Connect Sink

Kafka 로 데이터를 가져오고,
변환하고, 보내기가 쉬워짐

Kafka Connect – 사용법

- https://www.confluent.io/hub/
- 각종 커넥터를 다운받을 수 있음
- 다운 받은 jar 파일들 (플러그인) 경로를 config/connect-standalone.properties 파일의 plugin.path 에 명시

- bin/connect-standalone.sh config/connect-standalone.properties
- 명령으로 커넥터 실행
- 실행 후 kafka Topic 에 데이터가 들어 오는지 확인

Kafka Streams

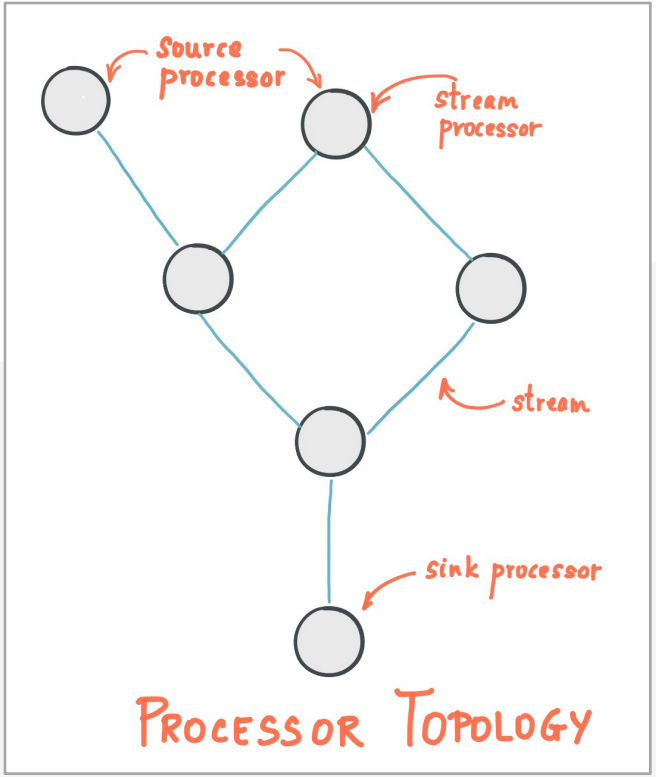
- Easy data processing and transformation library within kafka

- 카프카는 대규모 메세지를 저장하고 빠르게 처리하기 위해 만들어 졌다.
- 점차적으로 연속된 메세지 스트림을 처리하는데도 사용되기 시작함.
- 카프카의 Producer API , Consumer API 는 개별동작 하고, low level API 며 개발자 친화적이지 않아서, 새로운 Streams API 를 생성함
- Kafka 1.0 버전 (2016 년) 부터 가능

Kafka Streams

- **simple and lightweight client library**
- **no external dependencies on systems other than Apache Kafka itself**
- Supports **fault-tolerant local state**
- Supports **exactly-once**
- **one-record-at-a-time processing (no batching)**

- 처리 노드의 토폴로지를 정의 (각 노드가 1개의 토픽)
- 실시간으로 두개의 토픽 데이터를 합칠수도 있음
- 그래프 처럼 그려진 토폴로지를 따라서 작동하는 방식



- 토폴로지(topology): 전체 시스템의 부분을 배열하고 서로 연결하는 방법

Table of content

Use Streams API

- 샘플 소스코드 다운로드
- git clone https://github.com/kimscott/kafka-example.git

- Kafka-stream 과 spring-stream-kafka 를 사용하는 두개의 프로젝트로 구성됨 (두 개의 기능은 같음)

Use Streams API

- [실습]
- *1. input_topic 과 output_topic 토픽을 생성 합니다.*
  - *kafka-topics --bootstrap-server 127.0.0.1:9092 --topic input_topic --create --partitions 3 --replication-factor 1*
  - *kafka-topics --bootstrap-server 127.0.0.1:9092 --topic output_topic --create --partitions 3 --replication-factor 1*
  - *kafka-topics --bootstrap-server 127.0.0.1:9092 --list*
- *2. output_topic 토픽을 consume 합니다.*
  - *kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic output_topic --group output --from-beginning*
- *3. kafka-producer-basic 에 있는 SimpleJsonMessageProduce 클래스의 메인 메서드를 실행하여 이벤트를 발송합니다.*
- *4. kafka-streams-basic 프로젝트의 StreamFilterByEvent 클래스의 메인 메서드를 실행합니다.*

여지껏 배운 Kafka API 를 어떻게 쓰는게 좋을까?

- Kafka Producer API
  - 사용하기 간단하고, 비동기방식으로 사용할수 있는 장점이 있다.
  - 데이터 스트림을 직접 배출하는 시스템에 적합하다. 예) 로그, IOT
- Kafka Connect Source API
  - 데이터베이스로 부터 변화된 값만 읽어서 카프카에 적재 하고싶을때 사용할 수 있다.
  - 이미 구현된 커넥트를 실행만 시키면 데이터를 수집한다. 예) CDC, twitter, mongoDB
- Kafka Consumer API
  - 데이터 스트림을 읽어서 실시간으로 작업을 할때 사용. 예) 알람 발송, 이메일 발송 등
- Kafka Connect Sink API
  - 카프카로 부터 데이터베이스에 값을 넣을때 사용할수 있다.
  - 예) Kafka to S3, Kafka to HDFS, Kafka to PostgreSQL, Kafka to MongoDB
- Kafka Streams API
  - 카프카로부터 데이터를 받아서 처리 후 다시 카프카로 보낼때 주로 사용
  - 스트림 데이터를 원하는 데로 분리하고, 뭉치고, 가공이 가능
  - KSQL 같은 기능을 추가하여, 데이터 스트림의 내용을 쿼리형식으로 조회하여 수집이 가능함

Table of content

토픽 설정 변경

- 대부분의 토픽은 기본 설정은 Broker 에 존재 한다 (server.properties)
- 특별한 설정이 필요할때 토픽의 설정을 변경하여 사용 할 수 있다.
  - Replication Factor 조절
  - Partitions 사이즈 조절
  - 메세지 저장 기간 변경
  - 메세지 저장 사이즈 변경
  - 메세지 압축

- 자세한 설정 항목은 아래 공식 문서 활용
- http://kafka.apache.org/documentation.html#brokerconfigs

토픽 설정 변경 방법

- kafka-topics --bootstrap-server localhost:9092 --list
- kafka-topics --bootstrap-server localhost:9092 --create --topic config-test --partitions 1 --replication-factor 1
- kafka-topics --bootstrap-server localhost:9092 --alter --topic config-test --partitions 3
- kafka-topics --bootstrap-server localhost:9092 --alter --topic config-test --partitions 1
- kafka-topics --bootstrap-server localhost:9092 --describe --topic config-test

토픽 설정 변경 방법

- kafka-configs
- kafka-configs --bootstrap-server localhost:9092 --entity-type topics --entity-name config-test --describe
- kafka-configs --bootstrap-server localhost:9092 --entity-type topics --entity-name config-test --add-config retention.ms=10000 --alter
- ## log.retention.check.interval.ms=300000 (로그 보유 정책을 확인하는 시간이 기본 5분으로 되어있어서 로그 보유 기간을 10초로 주어도 맥시멈 5분이 걸릴 수 있다.)
- kafka-configs --bootstrap-server localhost:9092 --entity-type topics --entity-name config-test --delete-config retention.ms --alter

- ## 토픽을 생성하는 시점에 config 를 줄 수 있다. 다만 변경은 kafka-configs 를 통해서 사용하도록 가이드됨
- kafka-topics --bootstrap-server localhost:9092 --topic config-test1 --config retention.ms=10000 --create --partitions 1 --replication-factor 1
- kafka-topics --zookeeper localhost:2181 --topic config-test1 --config retention.ms=10000 --alter

Partitions and Segments

- 카프카는 삭제해야할 파일들을 쉽게 찾기 위하여 Segments 단위로 데이터를 저장한다.
- Partitions 에 데이터가 들어왔을때 Segments (file) 로 저장이 된다.

- 하나의 Segment 파일만 Active 이고 해당 파일에만 쓰기가 가능하다.
- 세그먼트는 인덱스 파일(.index)과 로그 파일(.log)로 구성된다
- 최대 세그먼트 크기 : log.segment.bytes (기본 1GB – server.properties)
- 세그먼트 저장 기간 : log.retention.hours(ms,minutes) =168 (기본 7일)

```
    4096 Jun 22 00:50 .
    4096 Jun 23 05:29 ..
10485760 Jun 22 05:30 00000000000000337954.index
 3343242 Jun 22 05:30 00000000000000337954.log
10485756 Jun 22 05:30 00000000000000337954.timeindex
```

Partitions and Segments

- 만약 log.segment.bytes (default 1GB) 를 줄이면 어떤일이 발생할까?
  - 파티션 별로 더 많은 Segments 가 발생함
  - 로그를 압축하는 행위 (Log Compaction : 기존 Active Segment 를 압축하고, 새로운 Active segment 파일 생성) 가 자주 발생함.
  - 카프카가 많은 Segment 파일을 열어야 해서 Too Many Open Files 에러가 발생 할 수 있음.
- 보통 하루에 log.segment.bytes 의 로그가 쌓인다면 매우 좋다고 추천됨

Log Cleanup

- Kafka 의 로그를 처리하는 컨셉은 log.cleanup.policy 로 처리 한다.
- log.cleanup.policy 는 두가지 옵션을 가짐
- log.cleanup.policy=delete (default)
  - 기본 설정에 의해서 모든 데이터를 삭제함
- log.cleanup.policy=compact
  - Key 값에 의하여 가장 최근 값만 토픽에 저장한다.

- # 대표적인 compact 토픽
- kafka-topics --bootstrap-server localhost:9092 --topic __consumer_offsets –describe

- Log cleanup 이 일어나는 주기는 segment 설정 크기에 따라 다르다.
- Segment 저장 크기를 작게 하면 Log cleanup 이 자주 일어나고, Log cleanup 은 CPU 와 RAM 리소스를 사용하니, 너무 자주 발생 시키는것은 안좋다.
- log.cleaner.backoff.ms (기본 15초) 에 의해서 체크됨

Log Compaction

- log.cleanup.policy=compact
- Key 값에 의하여 최근 데이터만 보여줌
- 컨슈머가 모든 데이터를 읽고 싶지 않고, 현재 상태값으로 부터 시작하고 싶을때 유용함

- Segment 에 데이터가 커밋 되었을때만 작동하기 때문에, 만약 계속 데이터를 consume 하고 있다면 중복된 데이터가 보여 질 수 있다.

- segment.ms : 엑티브 세그먼트를 닫는 시간 (default 7일)
- segment.byte : 멕시멈 세그먼트 사이즈
- min.cleanable.dirty.ratio : 로그 압축기가 로그 정리를 시도하는 빈도수를 제어, 비율이 높을수록 클리닝 하는 횟수는 적고, 효율적이지만 로그 공간이 낭비됨 (0.5 default : 50%)

Log Compaction - 실습

- kafka-topics --bootstrap-server localhost:9092 --create --topic bank-account --partitions 1 --replication-factor 1 --config cleanup.policy=compact --config segment.ms=5000 --config min.cleanable.dirty.ratio=0.001
- kafka-topics --bootstrap-server localhost:9092 --describe --topic bank-account
- kafka-console-consumer --bootstrap-server localhost:9092 --topic bank-account --from-beginning --property print.key=true --property key.separator=,
- kafka-console-producer --bootstrap-server localhost:9092 --topic bank-account --property parse.key=true --property key.separator=,
  - kim, 50000 won
  - lee, 30300 won
  - park, 20000 won
  - seo, 100000 won
  - kim, 30000 won
  - sin, 200000 won