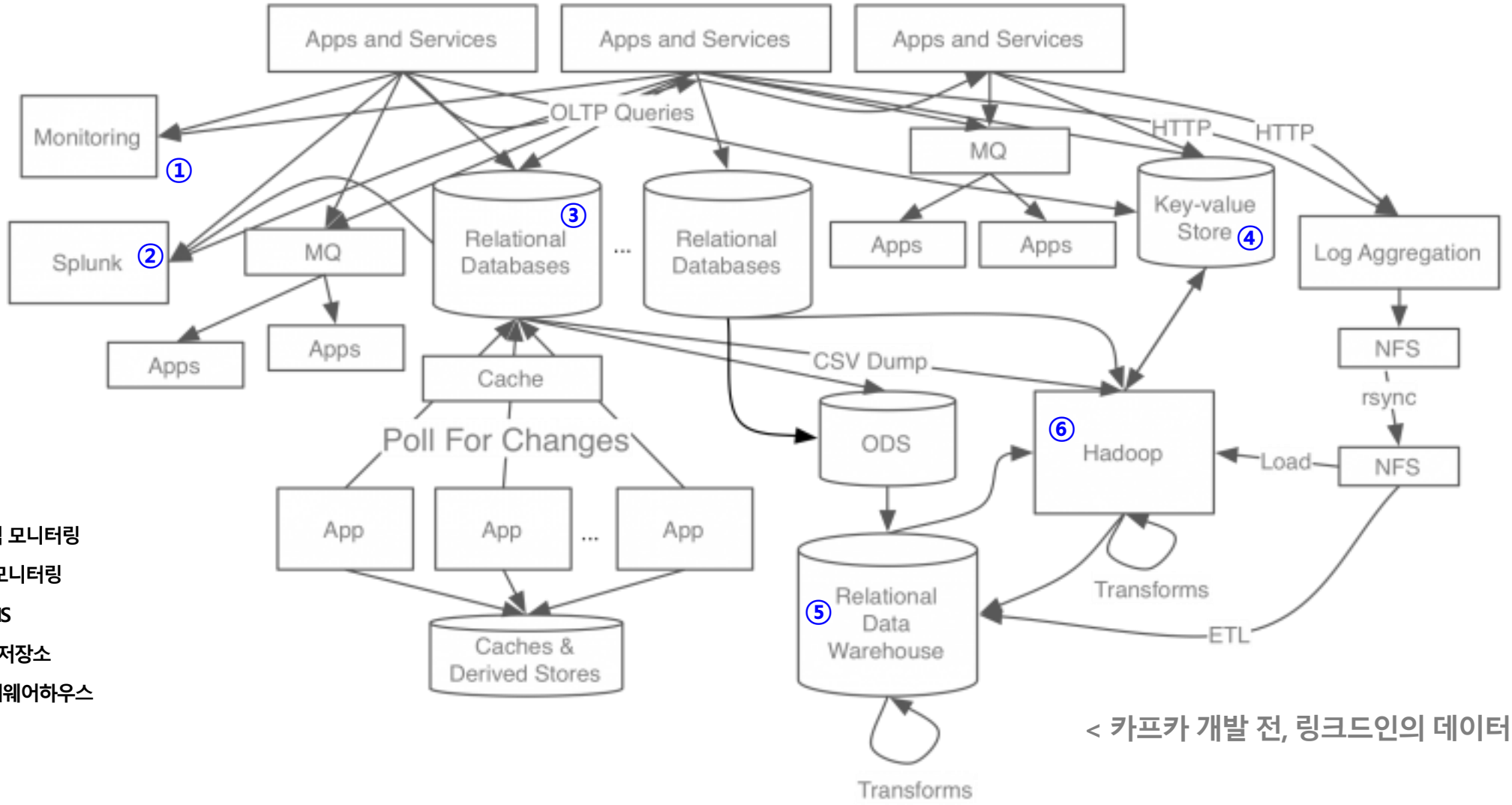
인공지능(AI)

KAFKA 구축

카프카 개요

- 대용량, 대규모 메시지 데이터를 빠르게 처리하도록 개발된 **메시징 플랫폼**
- 빅데이터 분석 시, 여러 스토리지와 분석 시스템에 데이터를 연결하기 위한 도구로 각광받고 있음
- 카프카 이용 기업
  - 해외 : 넷플릭스, 에어비앤비, 마이크로소프트 등 다수
  - 국내 : 네이버, 카카오 등 다수

카프카 탄생 배경



① 매트릭 모니터링

② 로그 모니터링

③ RDBMS

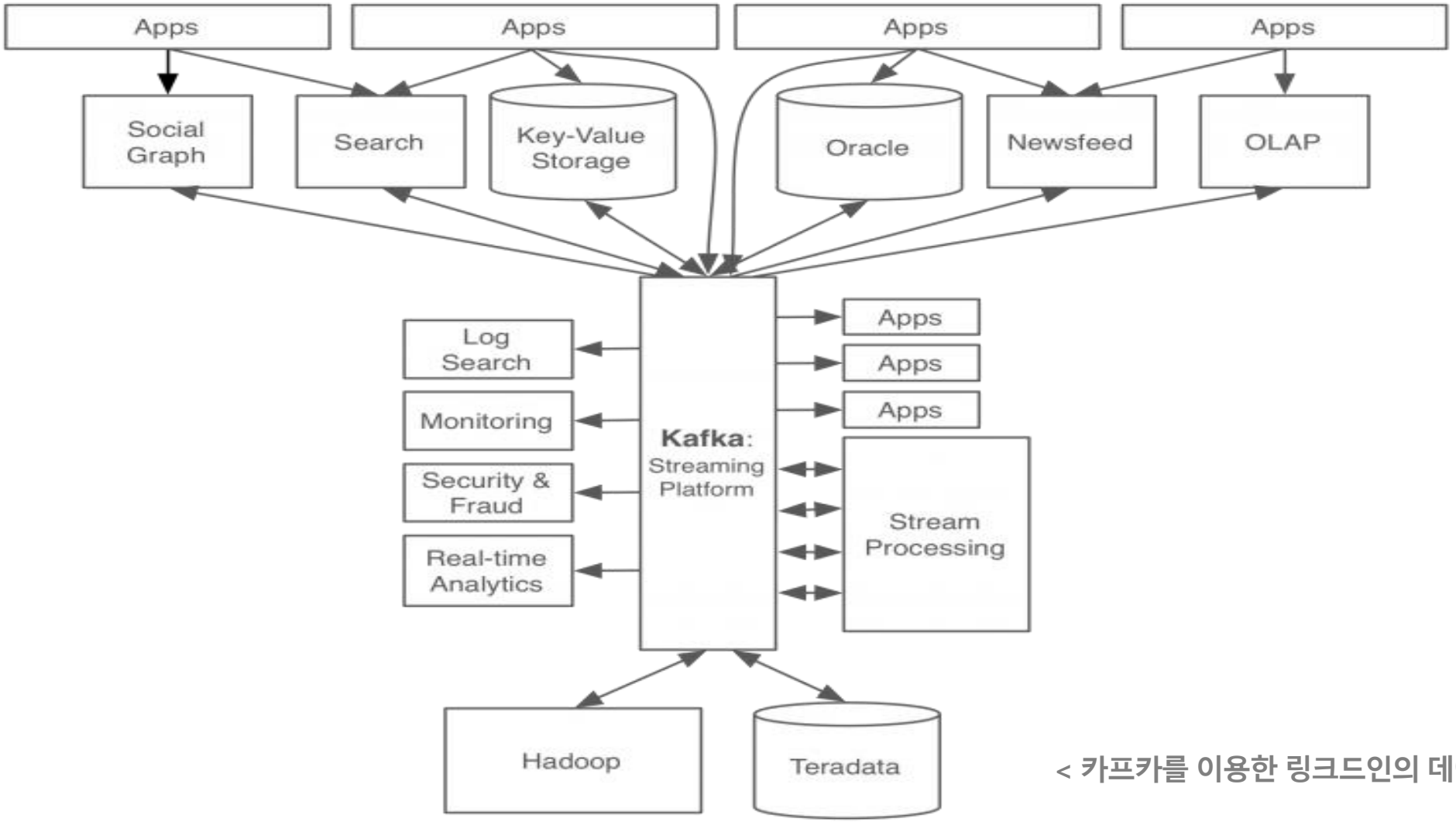
④ 키/값 저장소

⑤ 데이터웨어하우스

⑥ 하둡

< 카프카 개발 전, 링크드인의 데이터 처리 시스템 >

카프카 탄생 배경



< 카프카를 이용한 링크드인의 데이터 처리 시스템 >

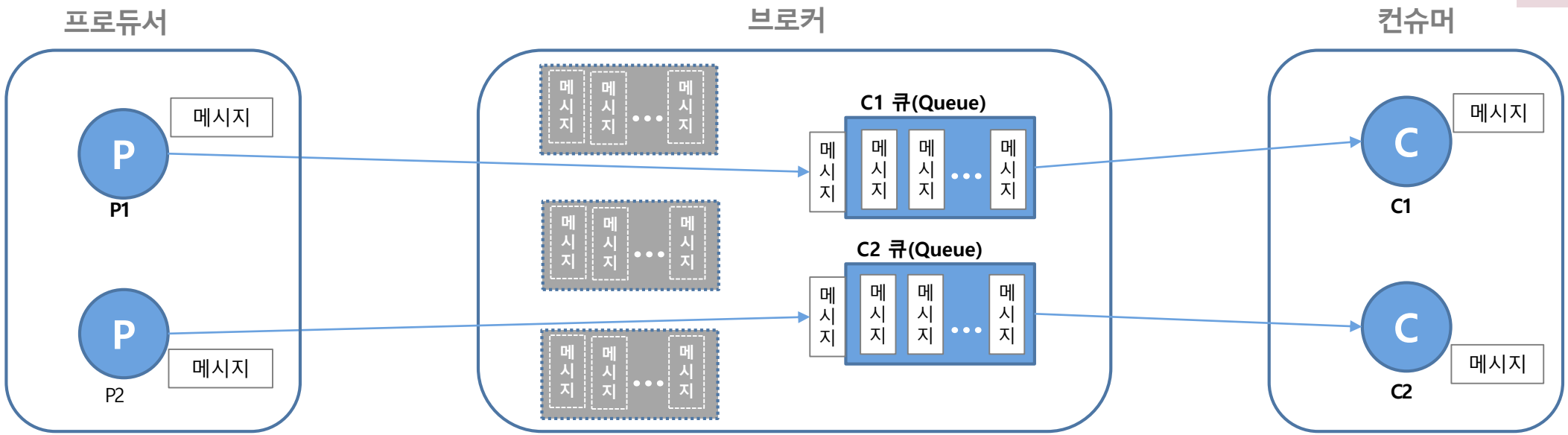
기본 개념

- 카프카는 비동기 처리를 위한 메시징 큐의 한 종류
- 프로듀서와 컨슈머로 구분 (메일과 비교)

중앙에 메시징 시스템 서버를 두고
메시지를 보내고(Publish) 받는
(Subscribe) 형태

➔ **펍(Pub)/섭(Sub) 모델**

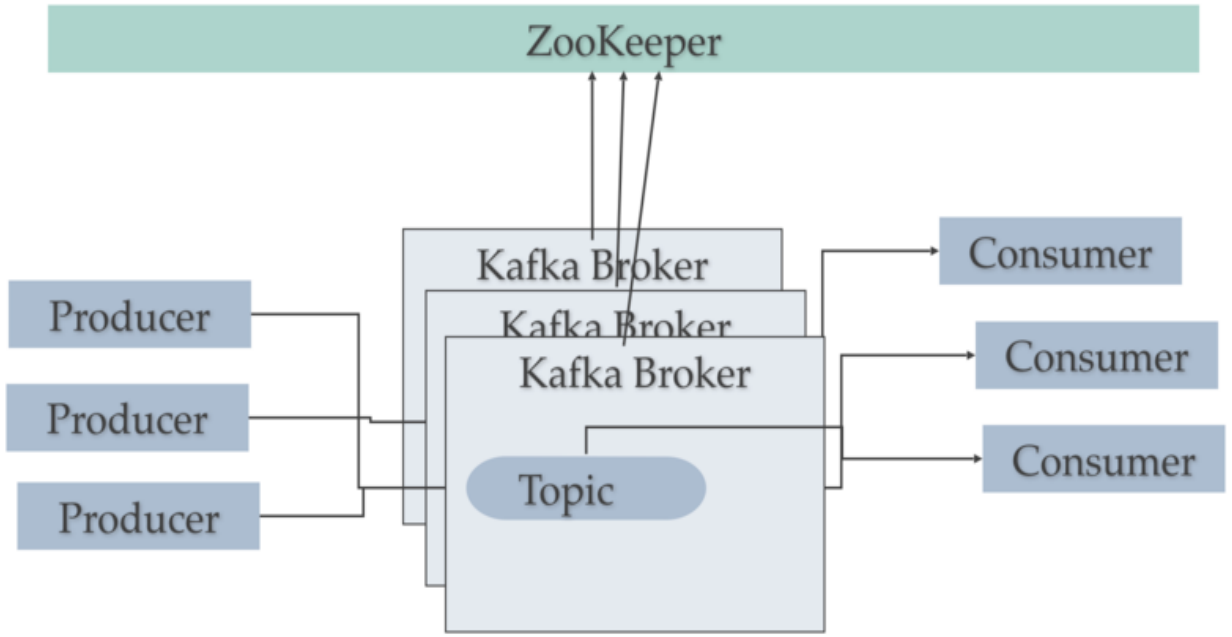
카프카 동작 방식



- 메시지 교환 전달의 신뢰성 관리 및 교환기 기능을 프로듀서와 컨슈머에서 처리
- 카프카(브로커)에서는 **메시징 전달** 기능에만 집중
  ➔ 데이터 파이프라인 역할(데이터 입출력 중계 버퍼)

- 메시지 전달과정
  1. 프로듀서는 새로운 메시지를 카프카 브로커로 전송
  2. 프로듀서가 전송한 메시지는 컨슈머 큐(토픽)에 도착하여 저장
  3. 컨슈머는 카프카 브로커에 접속, 새로운 메시지를 확인하여 가져감
  4. 브로커에는 토픽(Topic)이라는 식별자를 이용해 토픽 단위로 메시지들 저장 됨

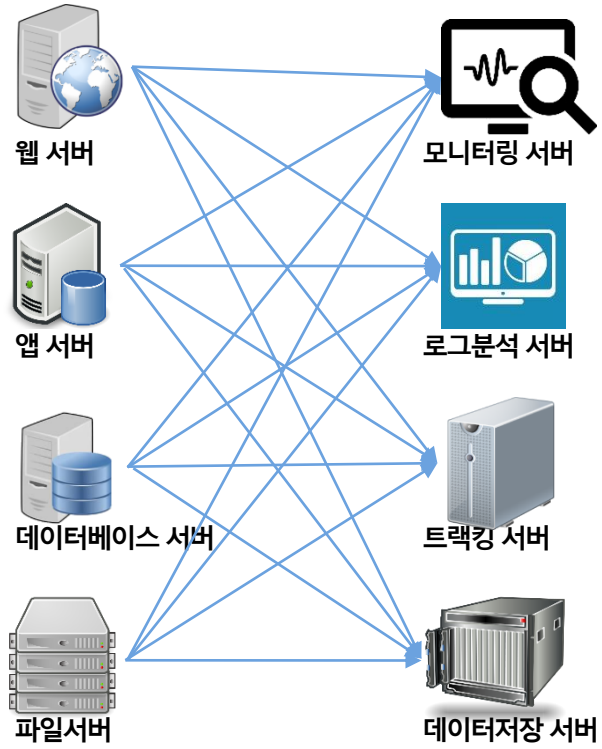
카프카 동작 방식



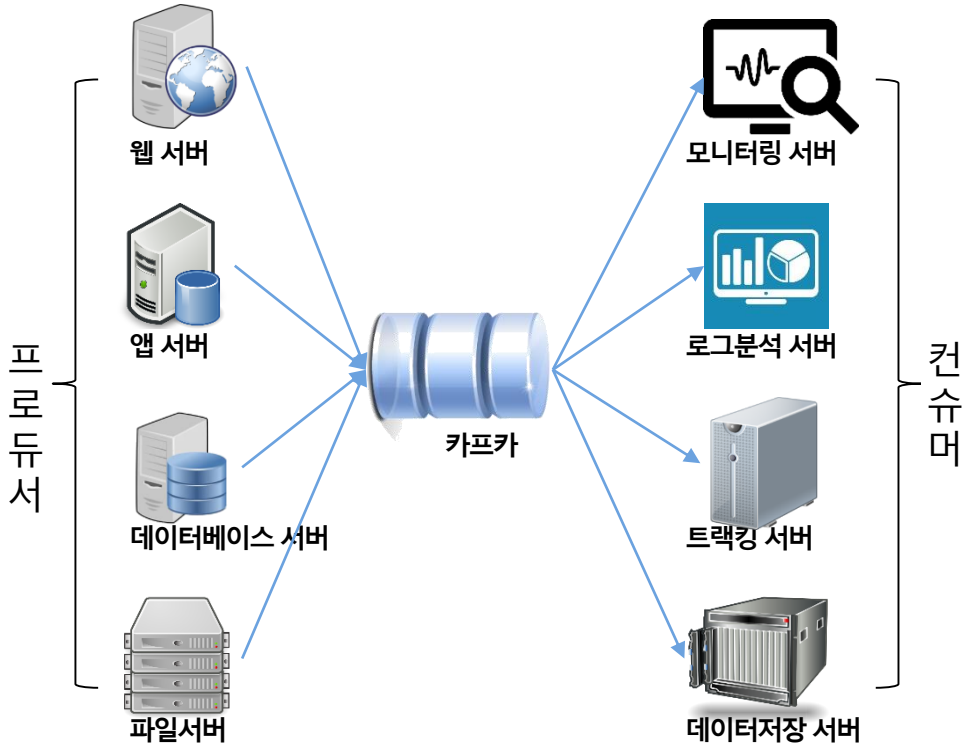
- 주키퍼(Zookeeper)는 분산 애플리케이션을 위한 코디네이션 시스템(분산 애플리케이션 관리)
- 카프카는 주키퍼를 통해 카프카 브로커의 클러스터 구성 및 상태 등을 관리
- 카프카 클러스터 각 노드는 브로커나 토픽의 추가/삭제 시점 등을 주피커를 통해 알 수 있음
- 주키퍼는 과반수 방식에 의해 운영되며 최소 3대 이상, 홀수 개로 주피커 클러스터(앙상블)를 구성해야 함

카프카 특징

- 프로듀서와 컨슈머의 분리



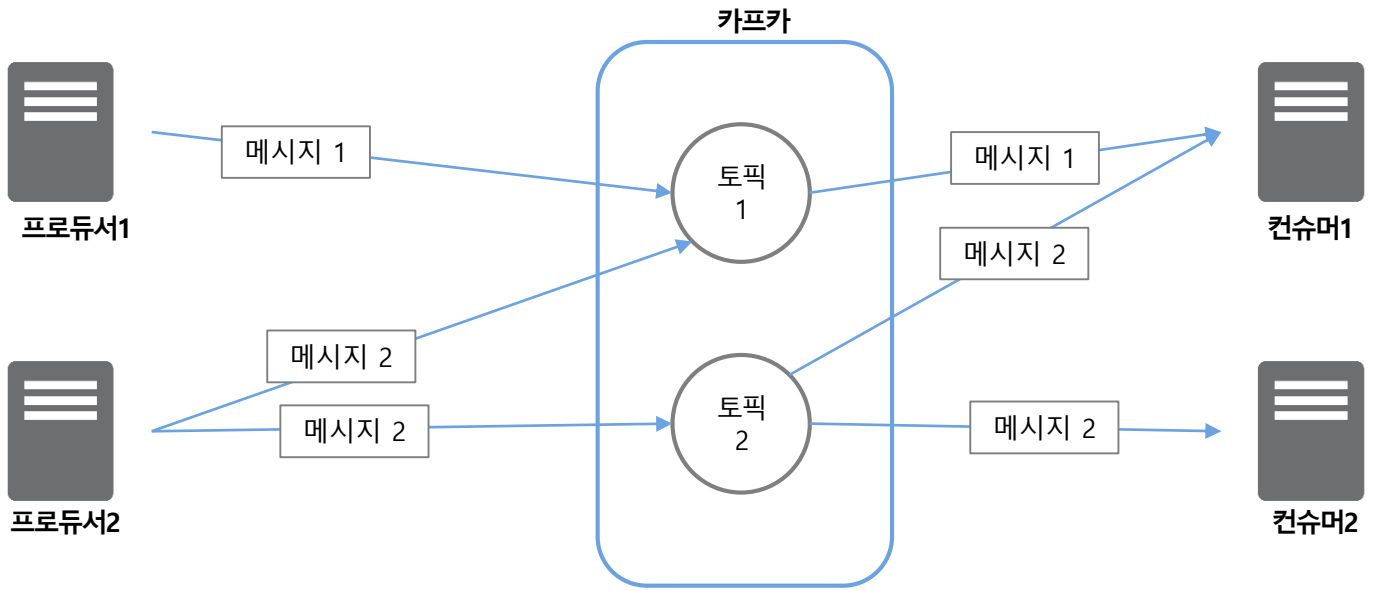
웹 서비스를 제공하는 기업에서
카프카를 사용하지 않은 구성



웹 서비스를 제공하는 기업에서
카프카를 이용한 펍/섭 구성

카프카 특징

- 멀티 프로듀서, 멀티 컨슈머
  - 하나의 토픽에 여러 프로듀서 또는 컨슈머가 접근 가능
  - 하나의 프로듀서가 여러 개의 토픽에 메시지 전달 가능
  - 하나의 컨슈머가 여러 개의 토픽에서 메시지 가져갈 수 있음

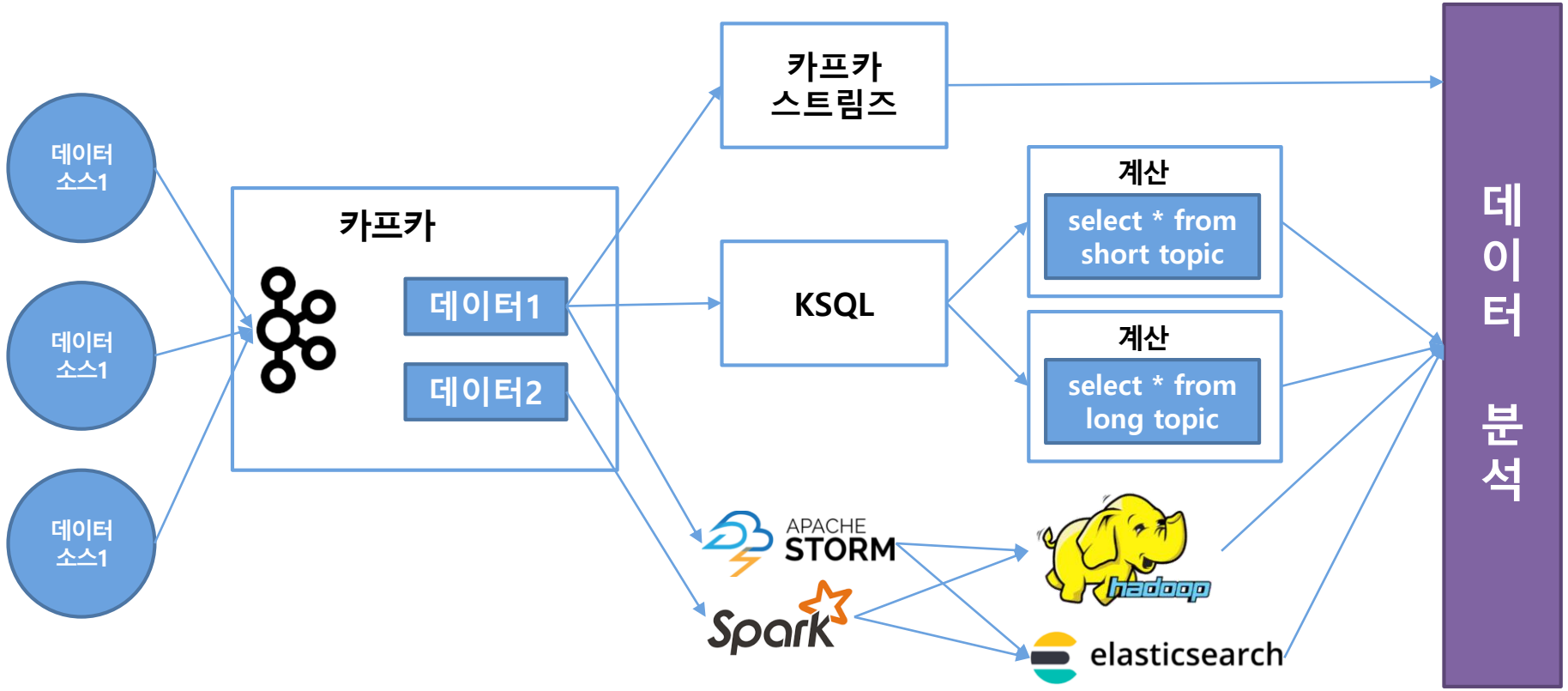


카프카 특징

- 디스크에 메시지 저장
  - 일반적인 메시징 시스템들은 컨슈머가 메시지 읽어가면 큐에서 메시지를 삭제함
  - 카프카는 컨슈머가 메시지를 읽어가더라도 미리 정해진 보관 주기(기본 7일) 동안 디스크에 메시지를 저장
  - 트래픽이 일시적으로 폭주하여 컨슈머 처리가 늦어지더라도 카프카의 디스크에 안전하게 보관
    → 컨슈머는 메시지 손실 없이 메시지를 가져갈 수 있음
  - 컨슈머에 오류 발생 시, 컨슈머를 잠시 중단하고 오류 해결 후 다시 실행 가능
- 확장성
  - 주피커-카프카 클러스터는 3대의 브로커로 시작해 수십 대의 브로커로 확장 가능
  - 확장 작업은 카프카 서비스의 중단 없이 온라인 상태에서 작업 가능
- 성능
  - 카프카는 내부적으로 분산 처리, 배치 처리 등 다양한 기법을 사용
  - 2015년 8월 기준으로 링크드인에서는 1조개의 메시지를 생산하고 카프카를 이용해 하루에 1페타바이트 이상의 데이터를 처리함

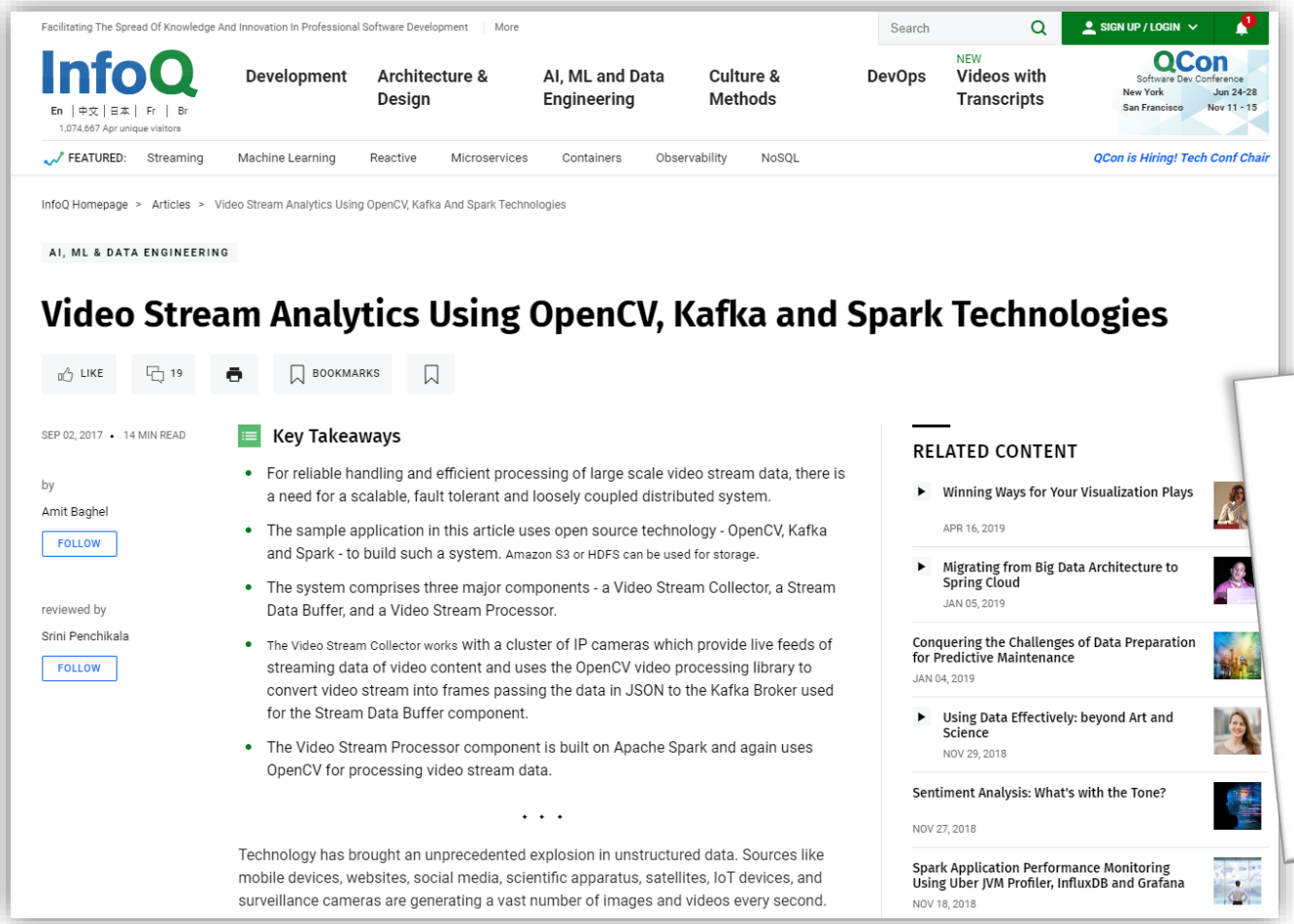
카프카의 확장과 발전

- 카프카 스트림즈, KSQL 등의 실시간 분석 시스템으로 진화
- 기존 아파치 스톰이나, 스파크, 하둡을 활용한 배치 분석 플랫폼과 연결 가능



카프카의 확장과 발전

- 영상 자료의 실시간 전송 및 분석에 활용



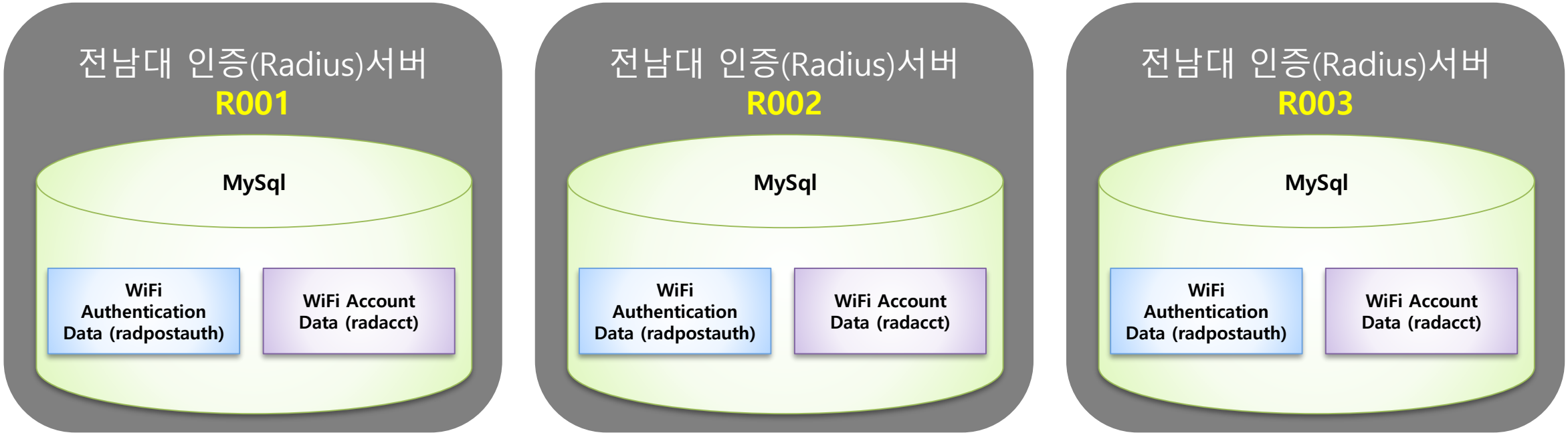
- object tracking,
- motion detection,
- face recognition,
- gesture recognition,
- augmented reality, and
- image segmentation.

카프카 용어 정리

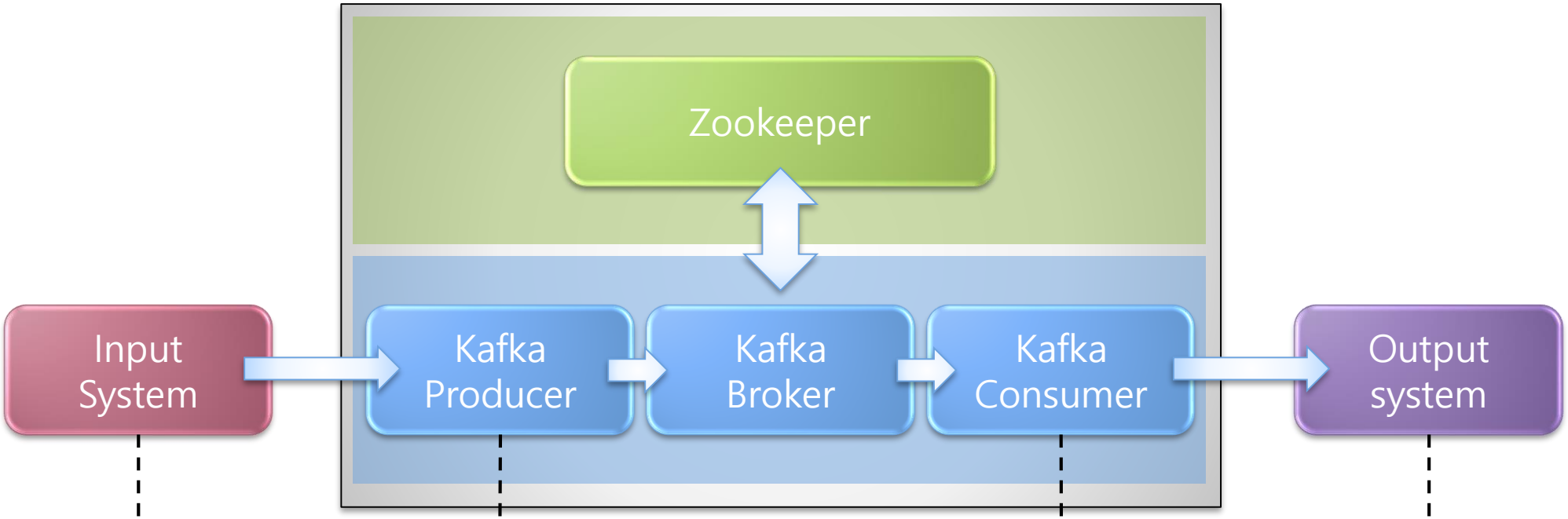
- **카프카(Kafka)** : 아파치 프로젝트 애플리케이션 이름, 클러스터 구성이 가능하여, 카프카 클러스터라 함
- **브로커(Broker)** : 카프카 애플리케이션이 설치되어 있는 서버 또는 노드
- **주키퍼(Zookeeper)** : 카프카 클러스터를 관리하기 위한 코디네이션 시스템, 최소 3개 이상 홀수 개의 클러스터로 구성되며 주키퍼 클러스터를 앙상블(ensemble)이라 함
- **토픽(Topic)** : 프로듀서와 컨슈머들이 카프카로 보낸 자신들의 메시지를 구분하기 위한 네임 (DB에서 Table과 비슷한 역할)
- **파티션(Partition)** : 토픽을 나누는 단위, 병렬 처리를 위해 토픽을 나눌 수 있고, 많은 양의 메시지 처리를 위해 파티션 수를 늘릴 수도 있음
- **프로듀서(Producer)** : 메시지를 생산하여 브로커의 토픽 이름으로 보내는 서버 또는 애플리케이션
- **컨슈머(Consumer)** : 브로커의 토픽 이름으로 저장된 메시지를 가져가는 서버 또는 애플리케이션

카프카 전송 실전 예(무선랜 인증 데이터)

전송하고자 하는 원본 데이터 (전남대 인증 서버에서 처리된 데이터, MySql)

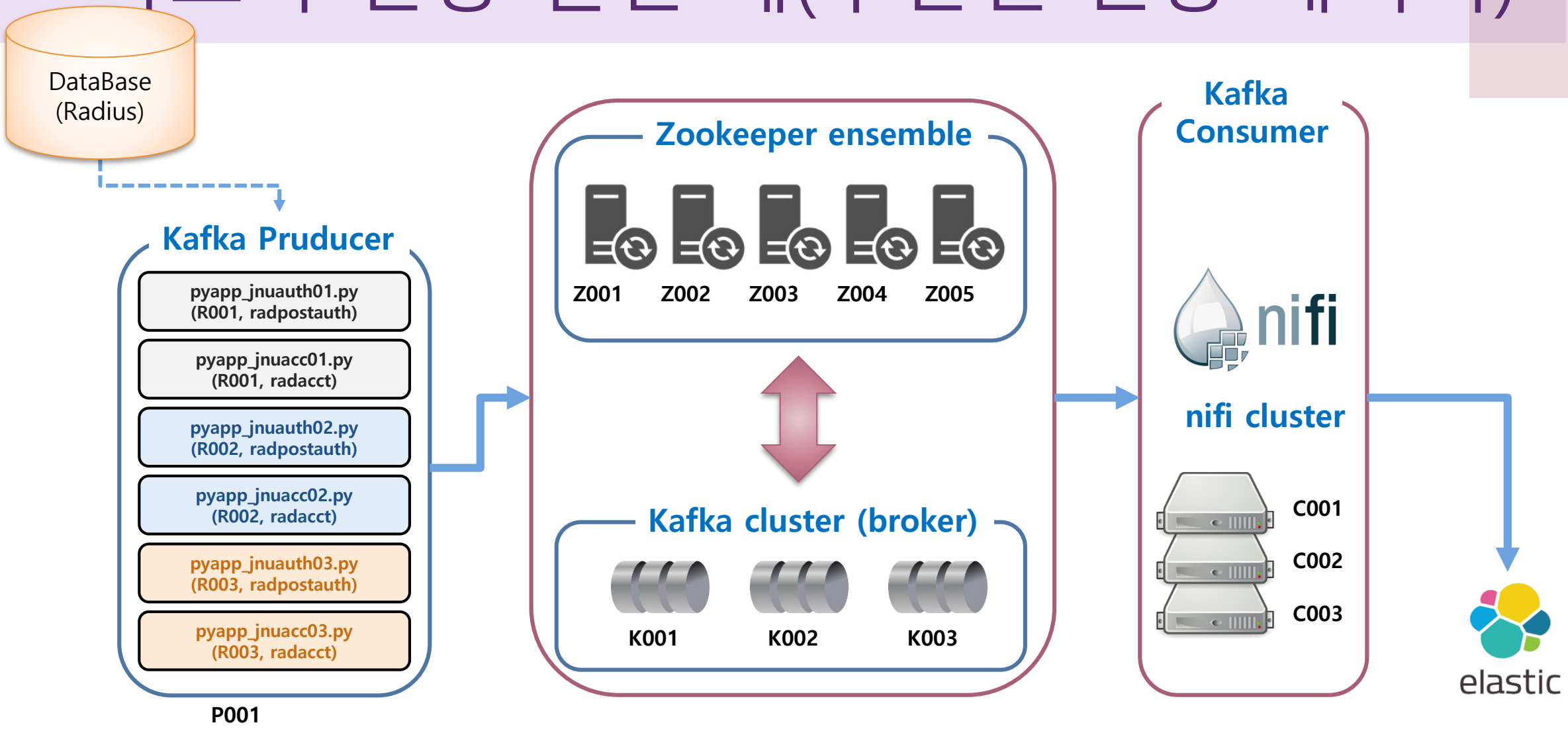


카프카 전송 실전 예(무선랜 인증 데이터)



- **무선랜인증 데이터(Radius)**
  → MySql
- **Python Application**
- **Apache NiFi Application**
  → 데이터 흐름 자동화 프로그램, 여러 종류의 시스템과 통신할 수 있고 시각적으로 보고, 수정 가능
- **Elasticsearch**
  → 분산형 검색 및 분석 엔진

카프카 전송 실전 예(무선랜 인증 데이터)



카프카 전송 실전 예(무선랜 인증 데이터)

Python을 이용한
Kafka Producer Application

root@elk9: /home/jnu

GNU nano 2.5.3 File: /usr/local/kafkaproducer/pyapp_jnuauth03.py

```
        conn.close()

def send_produced_messages(message):
    producer = KafkaProducer(acks=1, compression_type='gzip', bootstrap_servers='                                    ')
    future = producer.send('jnu_wifi_auth', message)
    result = future.get(timeout=10)


def main():

    rows = get_db()
    file_data = OrderedDict()

    sql="update radpostauth set kafkasend='Y' where id = %s;"

    if rows is not None:
        for r in rows:
            file_data["id"] = r[0]
            file_data["username"] = r[1]
            file_data["pass"] = r[2]
            file_data["reply"] = r[3]
            file_data["authdate"] = r[4].strftime('%Y-%m-%d %H:%M:%S')
            file_data["authdate_v20190503"] = r[4].isoformat()
            file_data["requestipaddress"] = ''
            file_data["calledstationid"] = ''
            file_data["callingstationid"] = ''
            file_data["framedipaddress"] = ''
            file_data["arubaessidname"] = ''
            file_data["arubalocationid"] = ''
            file_data["server"] = '168.131.33.63'

#            print(json.dumps(file_data).encode('utf-8'))

            cursor = db_ins.cursor()
            cursor.execute(sql,(r[0],))
            db_ins.commit()

            send_produced_messages(json.dumps(file_data).encode('utf-8'))


if __name__ == "__main__":
    now = int(datetime.now().strftime("%H%M"))
    if now < 200 or now >= 300:
        main()
```

^G Get Help ^O
^X Exit ^R

```
def send_produced_messages(message):
    producer = KafkaProducer(acks=1, compression_type='gzip', bootstrap_servers='                                    ')
    future = producer.send('jnu_wifi_auth', message)
    result = future.get(timeout=10)
```

카프카 전송 실전 예(무선랜 인증 데이터)

NiFi를 이용한 Kafka Consumer Application