카프카 활용

비동기 시스템 구성

비동기 처리를 위해 활용되는 **QUEUE**

- 배치
  - FILE
  - DB
- 준 실시간 (NEAR REAL TIME)
  - REDIS
  - **RABBITMQ**
  - **KAFKA**

비동기 처리를 위해 활용되는 **QUEUE**

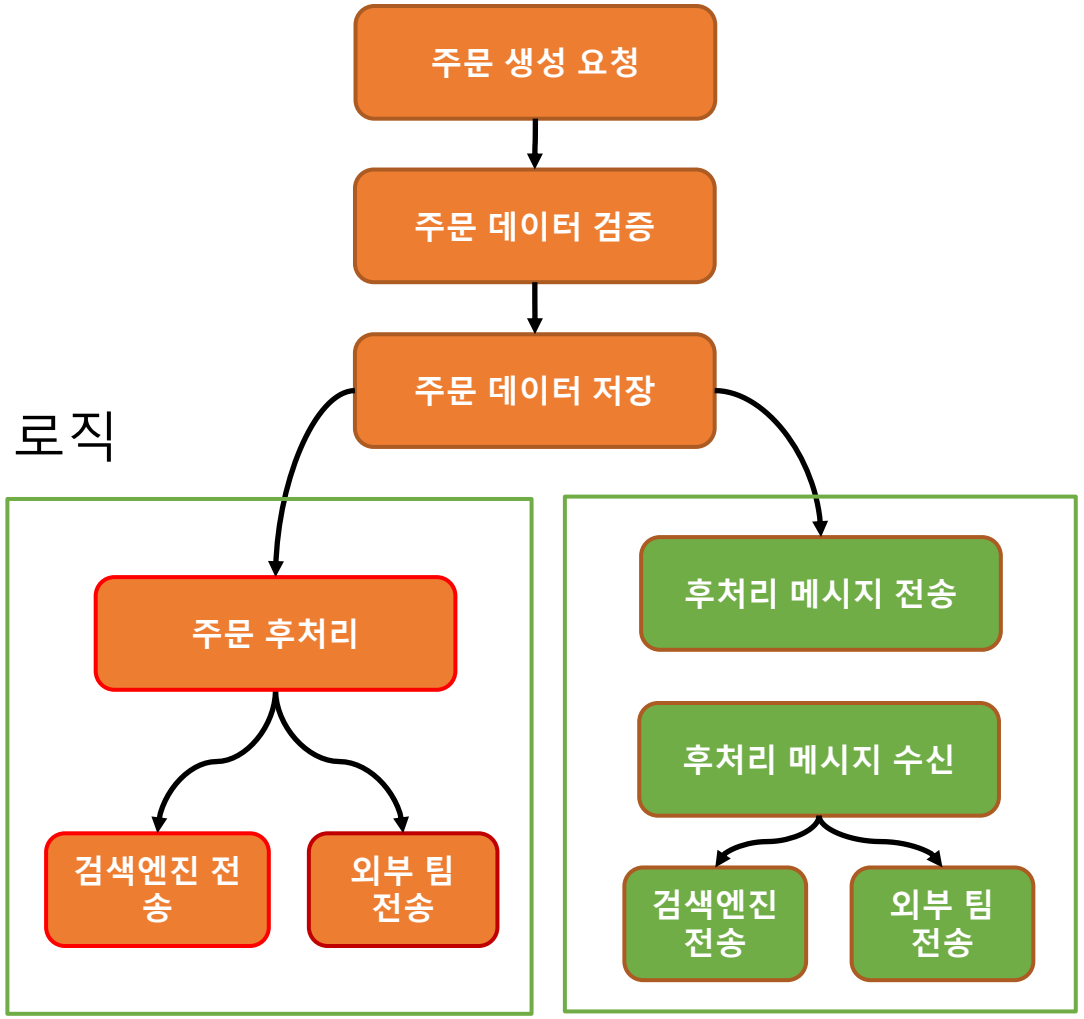
- KAFKA
  - 장점
    - 대용량 트래픽 메시지를 처리하는 구조에 적합
  - 단점
    - CONSUMER THREAD 수가 PARTITION 수에 종속적
    - CONSUMER 에서의 재처리 어려움
      - 재처리 시작점을 설정하는게 사실상 어려움
      - 메시지 마다 재처리 되는 타이밍을 설정 할 수 없음

비동기 처리를 위해 활용되는 **QUEUE**

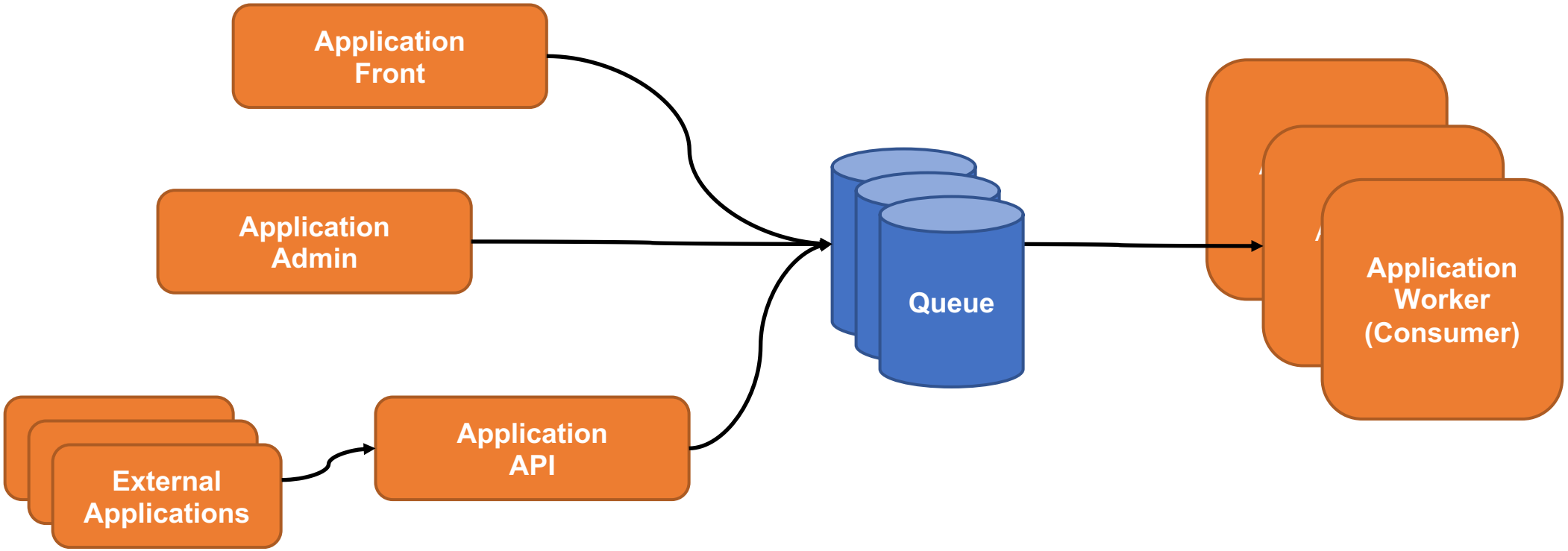
- RabbitMQ
  - 장점
    - Consumer 의 재처리를 위한 여러가지 기능들이 제공됨
    - Consumer 는 제한없이 추가 될 수 있음
  - 단점
    - 메시지가 분산되어 저장되는 구조가 아니므로 대용량을 처리하는데 한계가 있음
    - 여러 Consumer 가 추가되면 메시지 처리 순서를 보장할 수 없음

비동기로 처리하기에 적절한 로직

- 지연처리(응답)을 허용하고 실패가 중요 요소가 아닌 로직
  - 주문 저장 후,
    - 처리 결과를 ELASTICSEARCH 로 전송
    - 처리 결과를 외부 여러 팀으로 전송

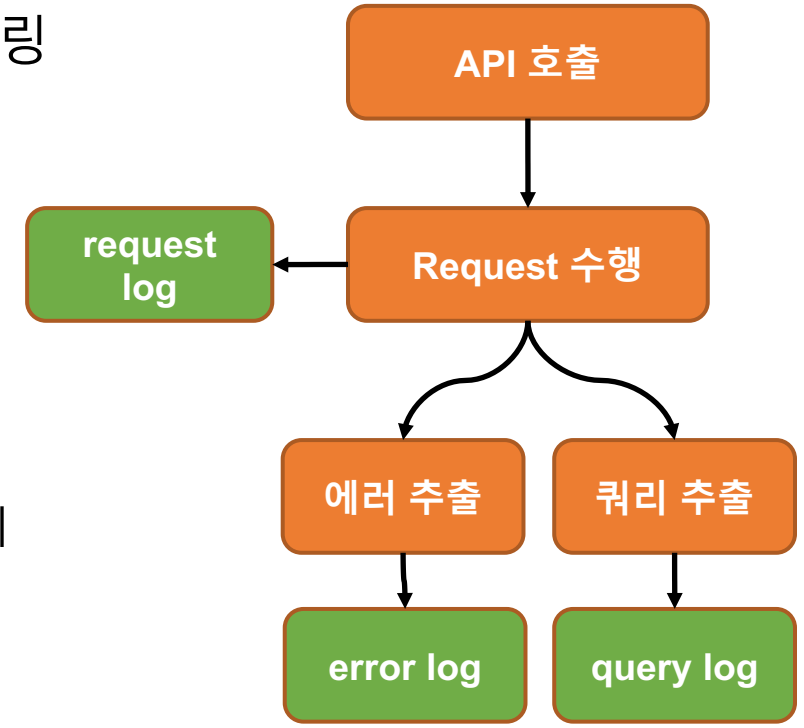


비동기 처리를 위한 일반적인 구조

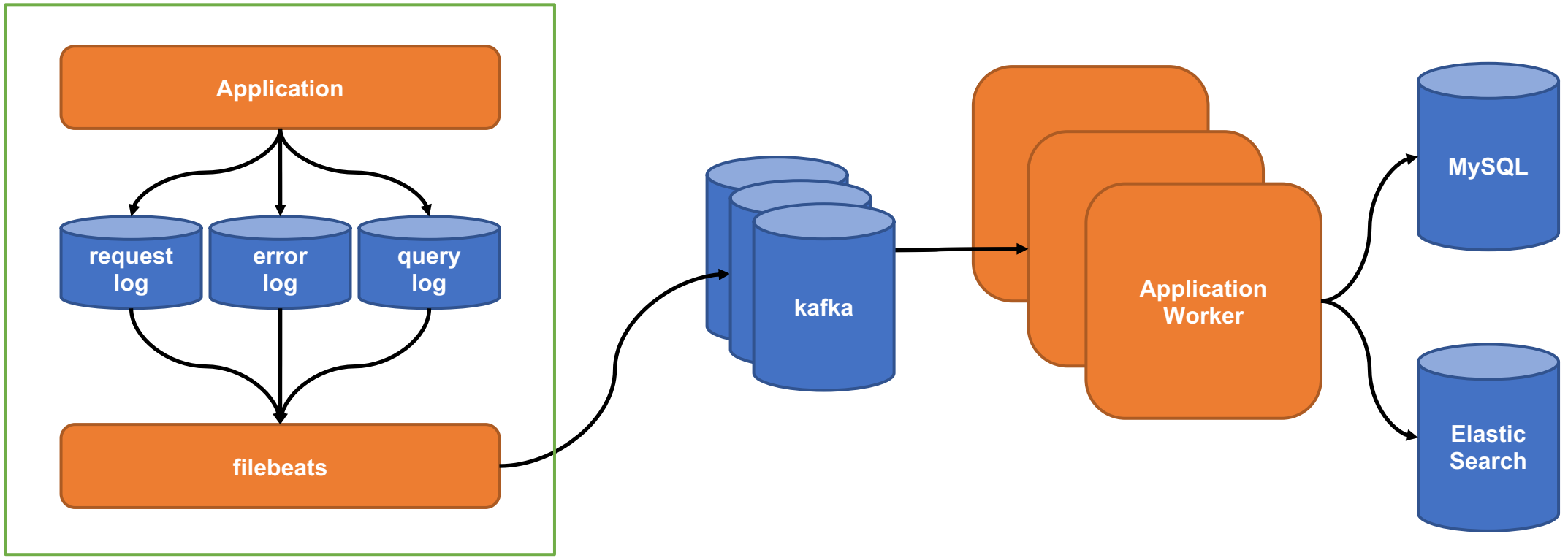


적용 #1 로그 적재

- 여러 APPLICATION 에서 발생된 로그를 한곳으로 수집 후 모니터링
  - REQUEST LOG
    - INPUT/OUTPUT 을 포함
  - ERROR LOG
    - 로직 수행도중 발생된 에러 혹은 정보성 로그
  - QUERY LOG
    - 수행된 모든 쿼리 수집 혹은 특정 THRESHOLD 이상 소요된 SLOW 쿼리



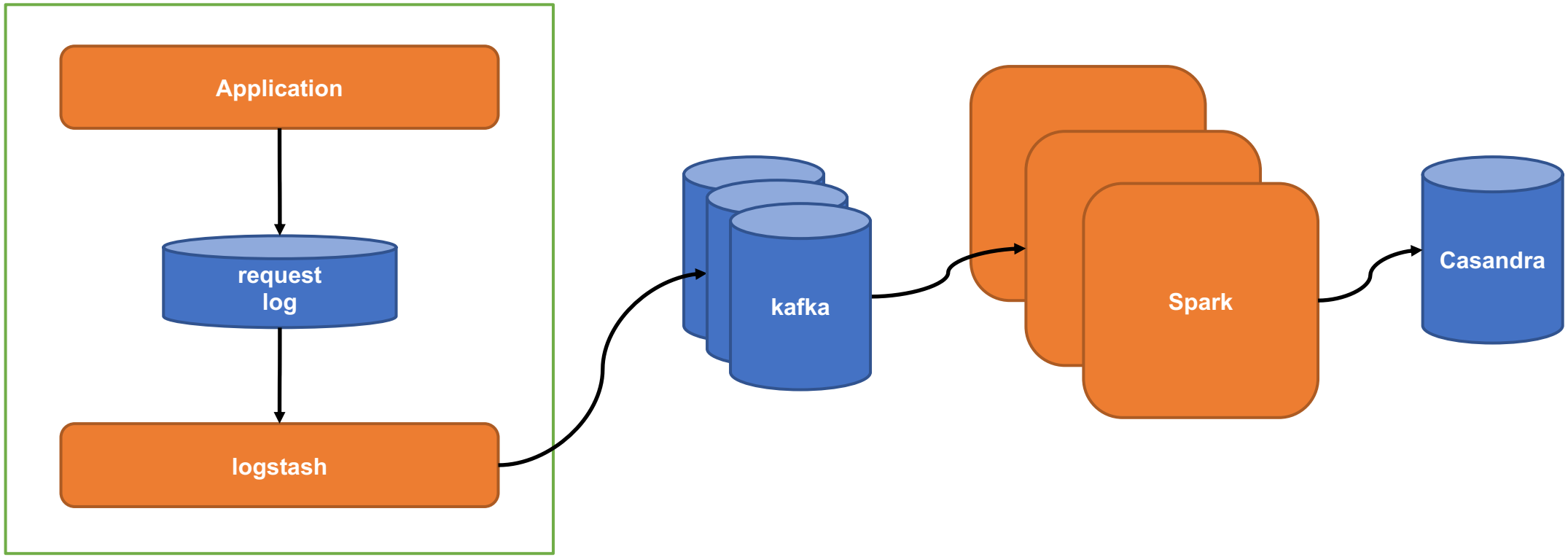
적용 **#1** 로그 적재


Application
request log
error log
query log
filebeats
kafka
Application Worker
MySQL
Elastic Search

적용 **#2** 스토리 홈 방문수 카운팅

- 개별 스토리 홈의 일간 방문수(PV)
  - 일간 수백만 명의 사용자가 방문하는 스토리 홈의 방문수를 지연 없이 반영
  - PV COUNTER STORAGE에 문제가 생겨도
    - 스토리 홈 기능에 영향 최소화
    - 문제가 된 데이터는 복구 가능

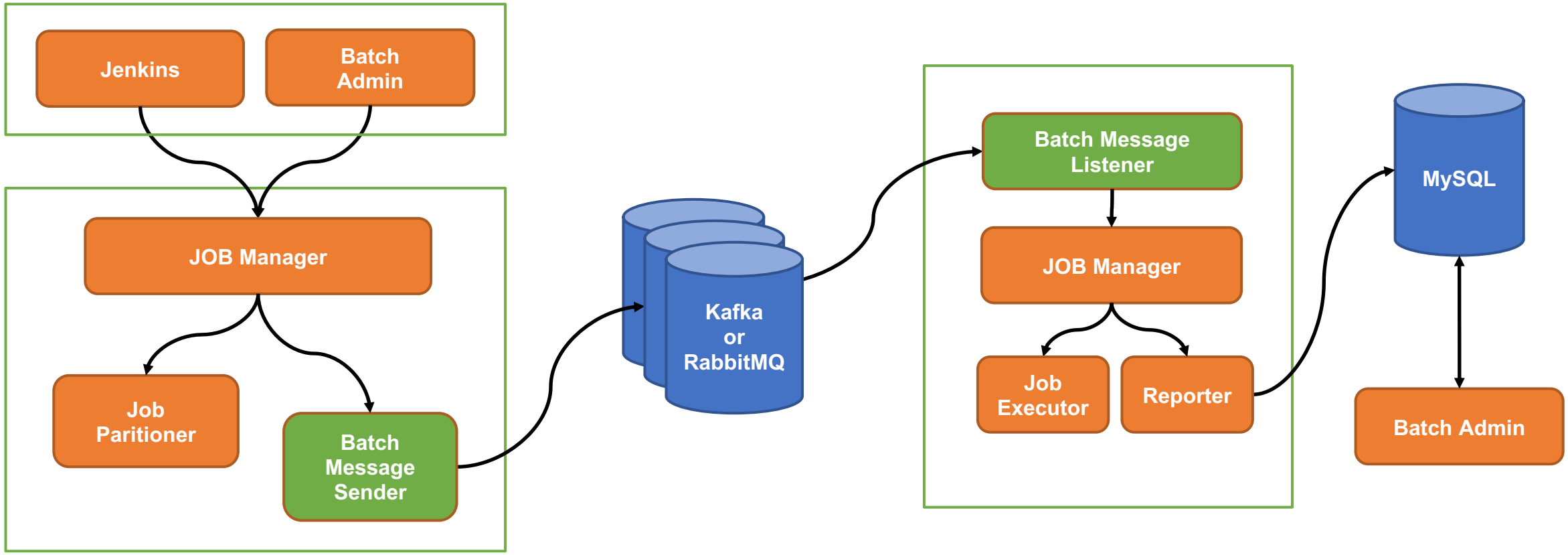
적용 **#2** 스토리 홈 방문수 카운팅



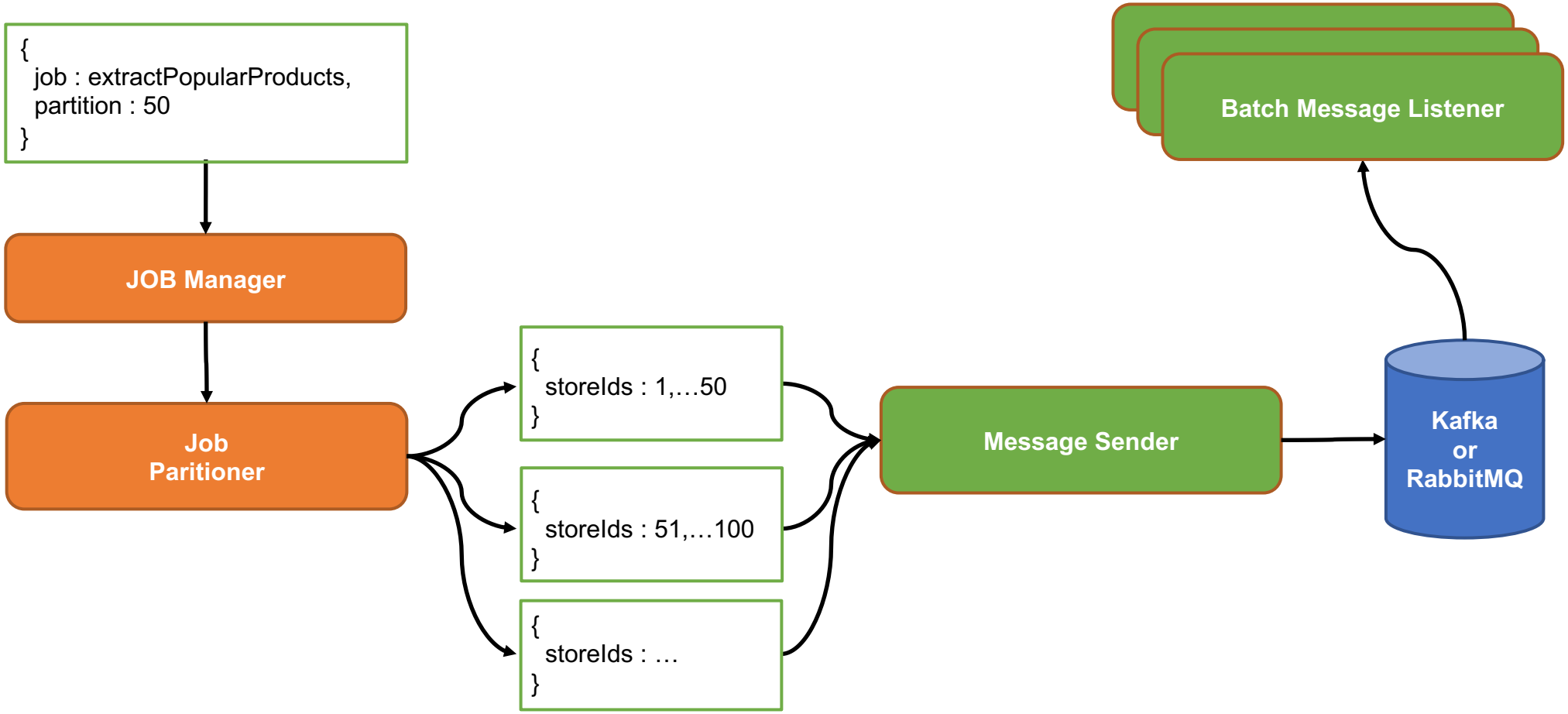
적용 **#3** 배치

- 주기적으로 “톡 스토어”에 입점한 모든 스토어를 대상으로 각 스토어의 인기상품 상품리스트를 추출하여 외부 팀에 전달
  - “톡 스토어”에 입점한 스토어들은 빠른 속도로 증가하고 개별 스토어의 누적 상품 수도 증가
    - 확장이 용이한 구조 필요

적용 **#3** 배치


Jenkins
Batch Admin
JOB Manager
Job Paritioner
Batch Message Sender
Kafka or RabbitMQ
Batch Message Listener
JOB Manager
Job Executor
Reporter
MySQL
Batch Admin

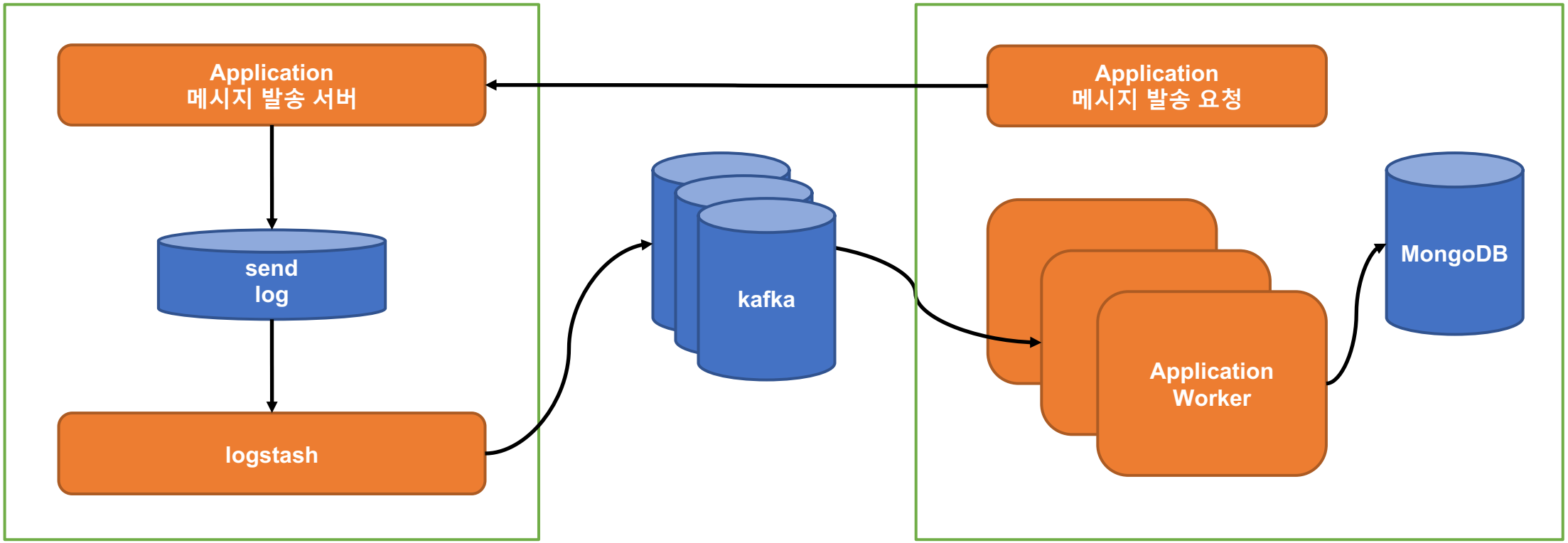
적용 **#3** 배치


{
job : extractPopularProducts,
partition : 50
}
JOB Manager
Job
Paritioner
{
storeIds : 1,…50
}
{
storeIds : 51,…100
}
{
storeIds : …
}
Message Sender
Batch Message Listener
Kafka
or
RabbitMQ

적용 #4 메시지 발송 결과 수신

- 연동 부서의 시스템 으로 메시지 전송을 요청하고
  전송 결과를 받아 별도의 저장소에 저장해 전송 결과를 조회함

적용 #4 메시지 발송 결과 수신



QUEUE 를 활용한 비동기 처리시 고민되는 부분

- QUEUE 로 메시지를 전송하는 방법
- QUEUE 의 메시지를 재처리 하는 방법
- QUEUE 혹은 APPLICATION 장애에 의한 메시지 유실 방지

QUEUE 로 메시지를 전송하는 방법

- 메시지를 로직에서 직접 전송
  - **ASYNC**
  - SYNC
- 메시지는 로그로 남기고 외부 솔루션을 통해 전송
  - LOGSTASH
  - **FILEBEAT**
  - **FLUENTD**
- 두 방법 혼합
  - 정상적인 경우는 로직 내에서 잔송하고, QUEUE 에 문제가 생긴 경우는 로그를 활용

QUEUE 의 메시지를 재처리 하는 방법

- 실패한 메시지를 메인 큐로 재 전송
  - 재처리 가능한 최대 횟수와 재시도 INTERVAL 설정
  - 최대 횟수 재처리 시도 후에도 실패하는 경우에 대한 처리 추가
    - 버림
    - 수동 재처리
- 실패한 메시지 처리를 위한 별도 큐 추가
  - 메인 큐
  - 재처리 큐(재처리를 위한 INTERVAL 종류에 하나 이상으로 구성)
  - 실패 큐

QUEUE 의 메시지를 재처리 하는 방법

- 대용량 트레픽 메시지 처리가 아니라면, RABBITMQ 를 통해 재처리 되도록 구성
  - DEAD LETTER 와 TTL 활용
    - QUEUE-ORIGINAL
    - QUEUE-RETRY
    - QUEUE-FAILED
- KAFKA 를 사용해야하는 구성이라면
  - 재처리를 위한 TOPIC 을 별도로 구성하는 구조
    - HTTPS://BLOG.PRAGMATISTS.COM/RETRYING-CONSUMER-ARCHITECTURE-IN-THE-APACHE-KAFKA-939AC4CB851A
  - 재처리를 위한 부분까지 KAFKA 에 의존적으로 구성하기보다는 보조 솔루션을 활용

QUEUE 장애 대응

- 메시지 전송
  - 특정 임계치 이상의 오류가 발생한 경우, 일정시간동안 전송을 멈춤(CIRCUIT BREAK)
  - 전송이 필요한 메시지들은 계속 유입이 되는 상태이므로 어딘가에 저장을 해두고 QUEUE 장애가 해결되면 전송되도록 구성
    - FILE, REDIS, DB 등 여러가지 대안이 사용될 수 있지만 FILE 기반 추천

배포 시점에서의 메시지 송/수신(처리) 보장

- SPRING 을 활용하여 서버가 구동되는 경우,
  - KILL을 통해 서버를 종료하게 되면 등록된 빈의 DESTROY 호출 순서에 의해 메시지 송신 혹은 수신된 메시지 처리가 정상적으로 되지 않고 종료 될 수 있음
- 서버 종료 시점에 모든 메시지 송/수신 처리가 보장되도록
  - GRACEFUL SHUTDOWN 구현 필요

배포 시점에서의 메시지 송/수신(처리) 보장

- 메시지 송신
  - 서버가 종료되는 시점에 모든 요청이 처리되었음이 보장 되어야함
    - 요청된 REQUEST 가 처리되는 도중에 서버 포트가 내려가게 되면 해당 요청은 실패
    - 요청된 REQUEST 가 READ 라면 재처리가 가능하지만, WRITE 라면 재처리 어려움

배포 시점에서의 메시지 송/수신(처리) 보장

- 메시지 수신(처리)
  - 외부 리소스(DB, HTTPCLIENT 등)가 종료되기전에 KAFKA, RABBITMQ 등의 CONSUMER 를 먼저 종료
    - SPRING 에 등록된 빈들의 DESTROY 순서가 의도대로 제어하기 어렵기 때문에
    - CONSUMER 가 메시지를 처리하는 도중에 외부 리소스가 먼저 종료되었다면
    - 메시지 처리가 실패할 수 있음

배포 시점에서의 메시지 송/수신(처리) 보장