MSA 아키텍쳐

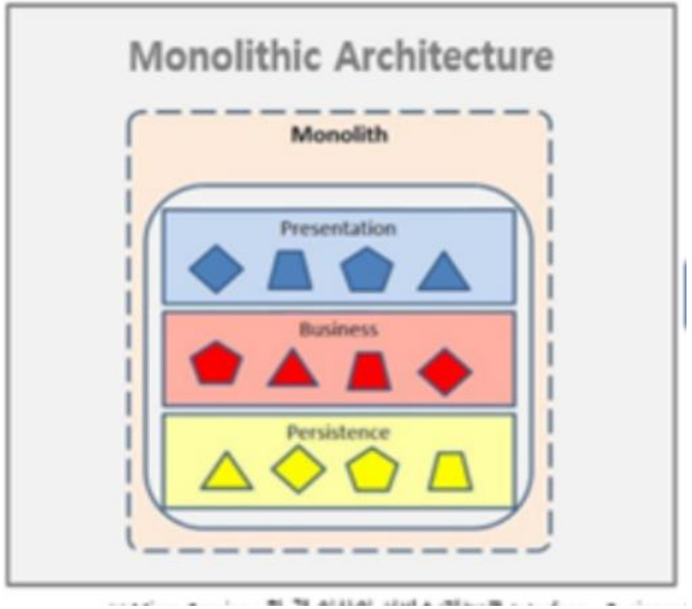
CONTENTS

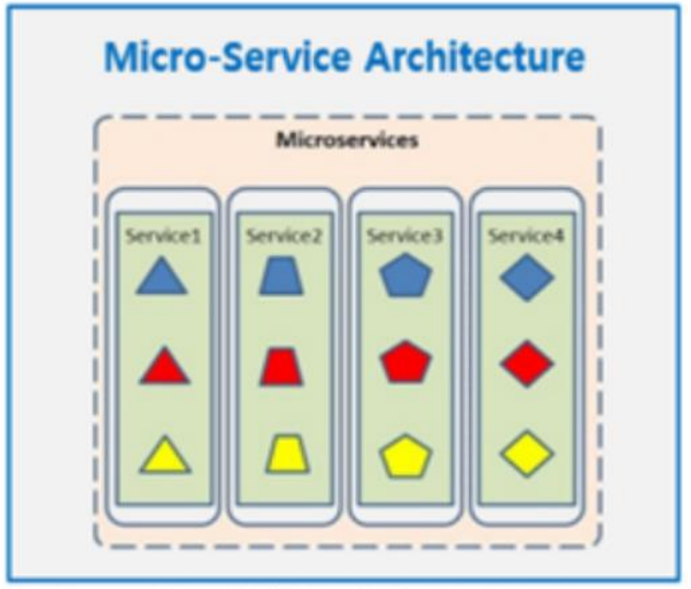
MSA 아키텍쳐

MSA 소개 : 정의

MSA(Micro Service Architecture)란?

- 단일 프로그램을 각 컴포넌트 별로 나누어 작은 서비스의 조합으로 구축하는 방법
- 작고, 독립적으로 배포 가능한 각각의 기능을 수행하는 서비스로 구성되어 단일 사업 영역에 초점을 둠





모든 업무 로직이 하나의 애플리케이션 형태로 패키징

기능 중심의 서비스 단위로 개발 및 배포

MSA 아키텍쳐

MSA 소개 : 등장 배경

MSA 등장 배경 : Monolithic 개발 방식의 한계

1. 부분 장애가 전체 서비스의 장애로 확대
   - 잘못된 코드 배포 또는 갑작스런 트래픽 증가로 인해 성능에 문제가 생겼을 때, 서비스 전체의 장애로 확대

2. 부분적인 Scale-out이 어렵다
   - 어플리케이션 전체 Scale out이 진행되어야 함

3. 서비스의 변경이 어렵고, 수정 시 장애의 영향도 파악이 힘들다.
   - 여러 컴포넌트가 하나의 서비스에 강하게 결합되어 있기 때문에 수정에 대한 영향도 파악이 힘들다

4. 배포 시간이 오래 걸린다.
   - 규모가 커짐에 따라 작은 변경에도 높은 수준의 테스트 비용이 발생 및 타 서비스와의 영향도 파악의 어려움

5. 하나의 Framework와 언어에 종속적이다.
   - 서비스 별 강점이 있는 언어 및 환경 있음에도 하나의 개발 환경에서 전체 서비스 개발

MSA 아키텍쳐

MSA 소개 : MSA 장단점

장점

1. Antifragile (반취약성)
   - 부분의 장애가 전체의 장애로 이어지지 X
   - 빠른 장애 파악 및 대응

2. 서비스별로 독립적 배포가 가능
   - 빠른 개발 및 유지보수 용이
   - 가벼워진 CI/CD 배포 환경

3. 개별적인 scale-out
   - 메모리, CPU 등 자원에 있어 효율적 사용

4.서비스별 개발 환경 설정
   - 기능에 특화된 개발 언어 / DB 사용

단점

1. 복잡한 어플리케이션 구조
   - 어플리케이션이 커질 수록 서비스간 종속성, 연결 관계 파악의 어려움
   - 연결 관계에 따라 장애 전파 가능성 존재

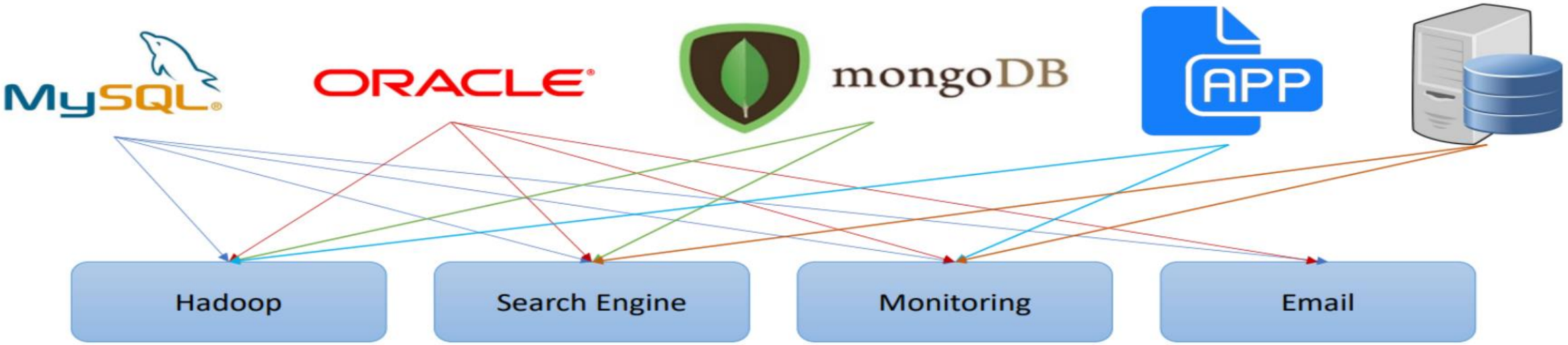
2. 서비스 성능 저하 가능성
   - 분산 시스템의 서비스 간 네트워크 통신 증가

3. 트랜잭션 처리 등 필요 로직의 증가
   - 통신의 장애와 서버의 부하 시의 별도 transaction 유지 방법 필요

=> MSA 전환을 위해선 복잡도 / 서비스 관계 /성능에 관한 고려 필요

MSA 아키텍쳐

MSA 통신: KAFKA 등장 배경



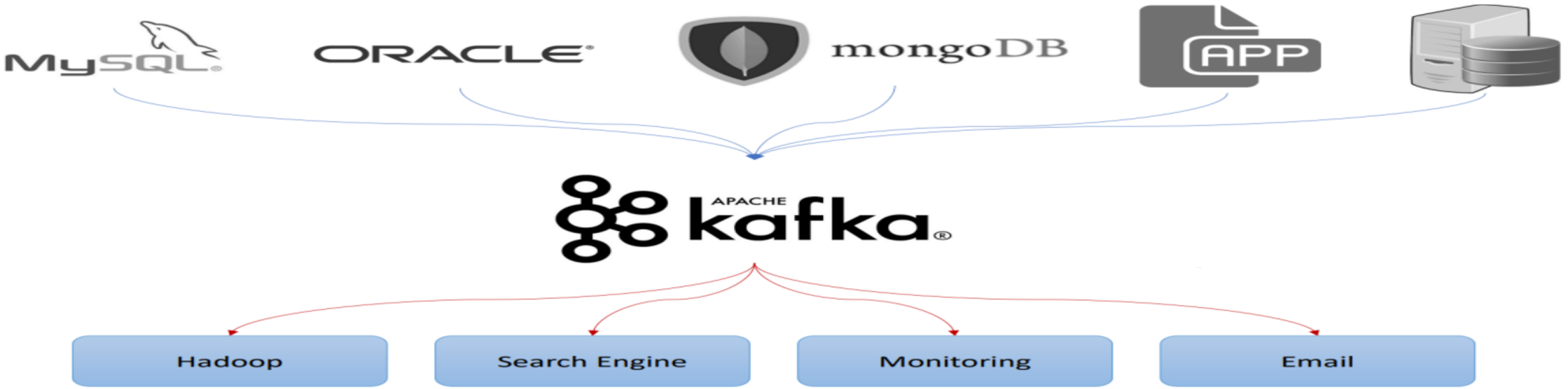
- MSA 환경에서의 DB 및 컴포넌트 다분화 = > DB와 서비스간 연결 관계 복잡도 증가
- 연결된 서비스끼리의 데이터 전송 / 입력 형식 조율 필요
- 기존의 End-To-End 방식으로는 서비스 확장에 따른 복잡성 해결 불가

=> 모든 시스템으로의 데이터 전송이 가능한 전송 모델의 필요성 증가

=> MOM(Message Oriented Middleware) 구조의 데이터 전송 모델 등장

MSA 아키텍쳐

MSA 통신: KAFKA 등장 배경



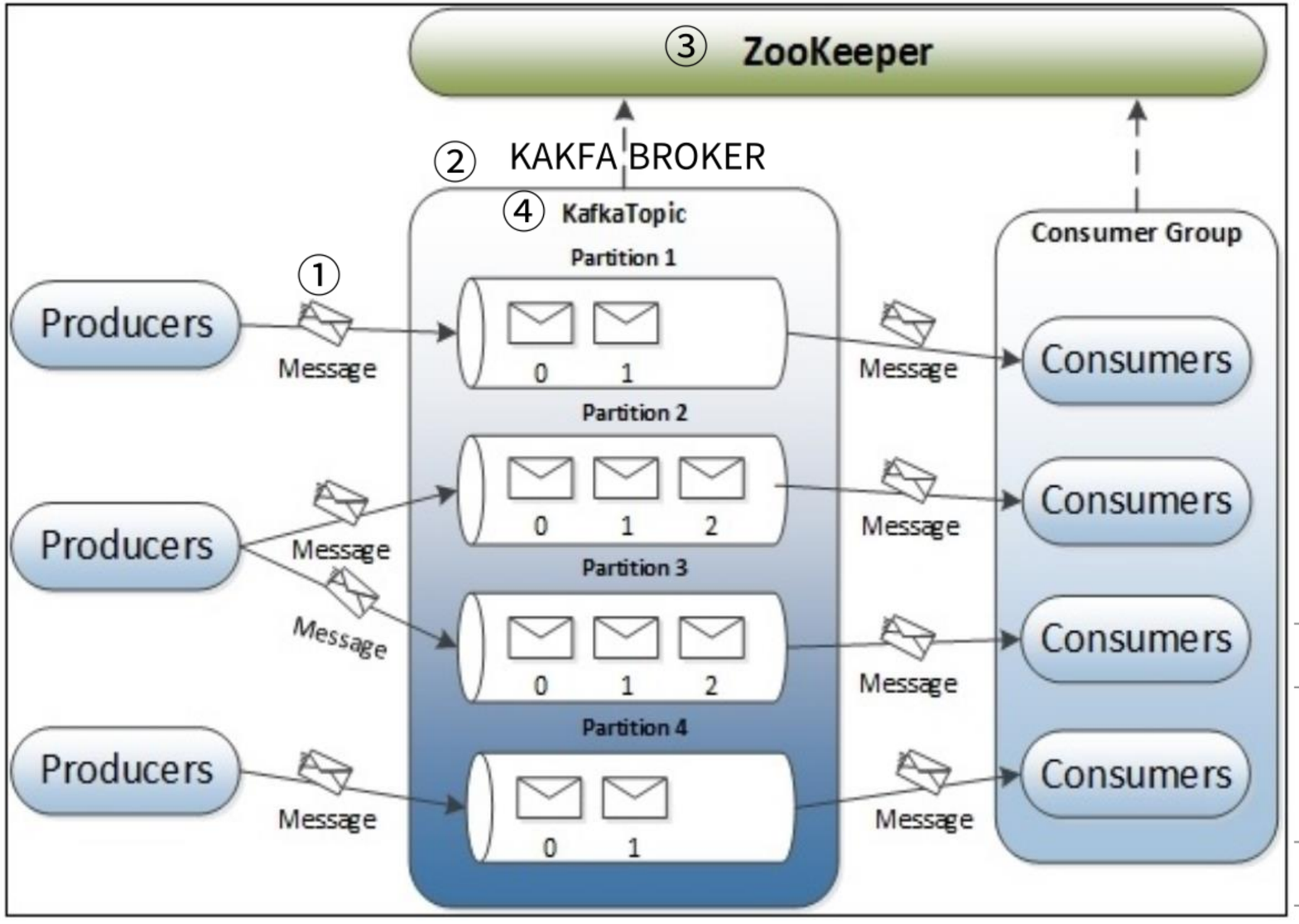
- Producer/Consumer 분리 , 중앙 시스템이 데이터 전송 관리
- 메시지를 여러 Consumer에게 허용

↓

- Producer/Consumer 분리 , 중앙 시스템이 데이터 전송 관리
- 데이터 전송 / 입력 형식을 중앙 서버에 맞춰 서비스 별로 구현 가능
- 메시지를 여러 Consumer에게 허용함에 따라 서비스 확장 용이

MSA 아키텍쳐

MSA 통신: KAFKA 아키텍쳐



① 메시지 (Event)

- Producer와 Consumer가 메시지를 주고 받는 단위

② KAFKA BROKER

- Kafka 서버
- 메시지를 주고 받을 때 저장되는 공간
- 메시지를 받아 처리하는 Leader Broker와
  데이터를 복제 저장 하는 Follwer Broker 구성

③ Zookeeper

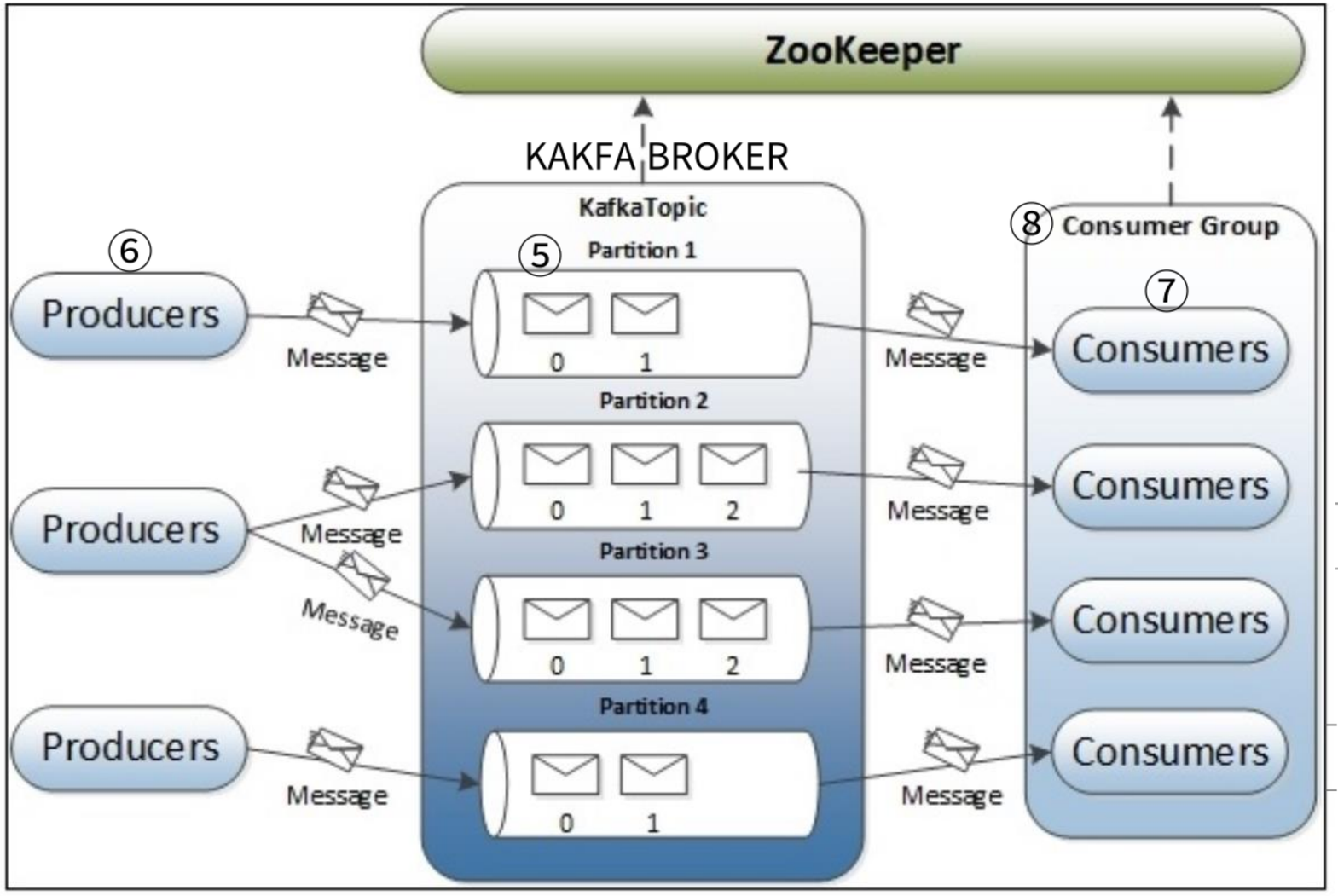
- Broker을 중재하는 Controller
- 메타데이터 저장 / 리더 브로커 선출

④ Topic

- 메시지가 쓰이는 곳
- 같은 Topic을 가진 메시지들의 처리 목적

MSA 아키텍쳐

MSA 통신: KAFKA 아키텍쳐



⑤ Partition

- 여러 Broker에 분산된 Topic
- 메시지의 key값에 따라 분산 처리

⑥ Producer

- 이벤트 게시하는 클라이언트

⑦ Consumer

- Topic을 구독하고 이벤트 처리하는 클라이언트
- 지정된 partition에서 메시지 구독

⑧ Consumer Group

- 하나의 Topic을 구독하는 여러 Consumer 집합
- 특정 Consumer 불능 시, Reblance 작업을 통해 담당 partition 재지정

MSA 아키텍쳐

MSA 통신: KAFKA 특징

확장성

- Topic을 구독하는 Consumer 추가 용이

- 메시지 보관 / 처리하는 Broker 서버의 탄력적 운용 가능

- 단, consumer별 지정 partition을 재조정 시에는 실시간 처리 불가

내구성

- broker 서버의 replication 지원을 통한 데이터 유실 위험 낮춤

- 데이터 처리를 담당하는 Leader broker의 장애 시, 다른 leader 선출을 통한 장애 예방

Producer 중심 메시징

- Producer와 Consumer는 비동기 통신

- Consumer가 메시지 받는 것과 무관하게 Producer는 메시지 생산

- Consumer가 전달 받았음을 보장하기 위해서는 복잡한 코드 필요

높은 처리율

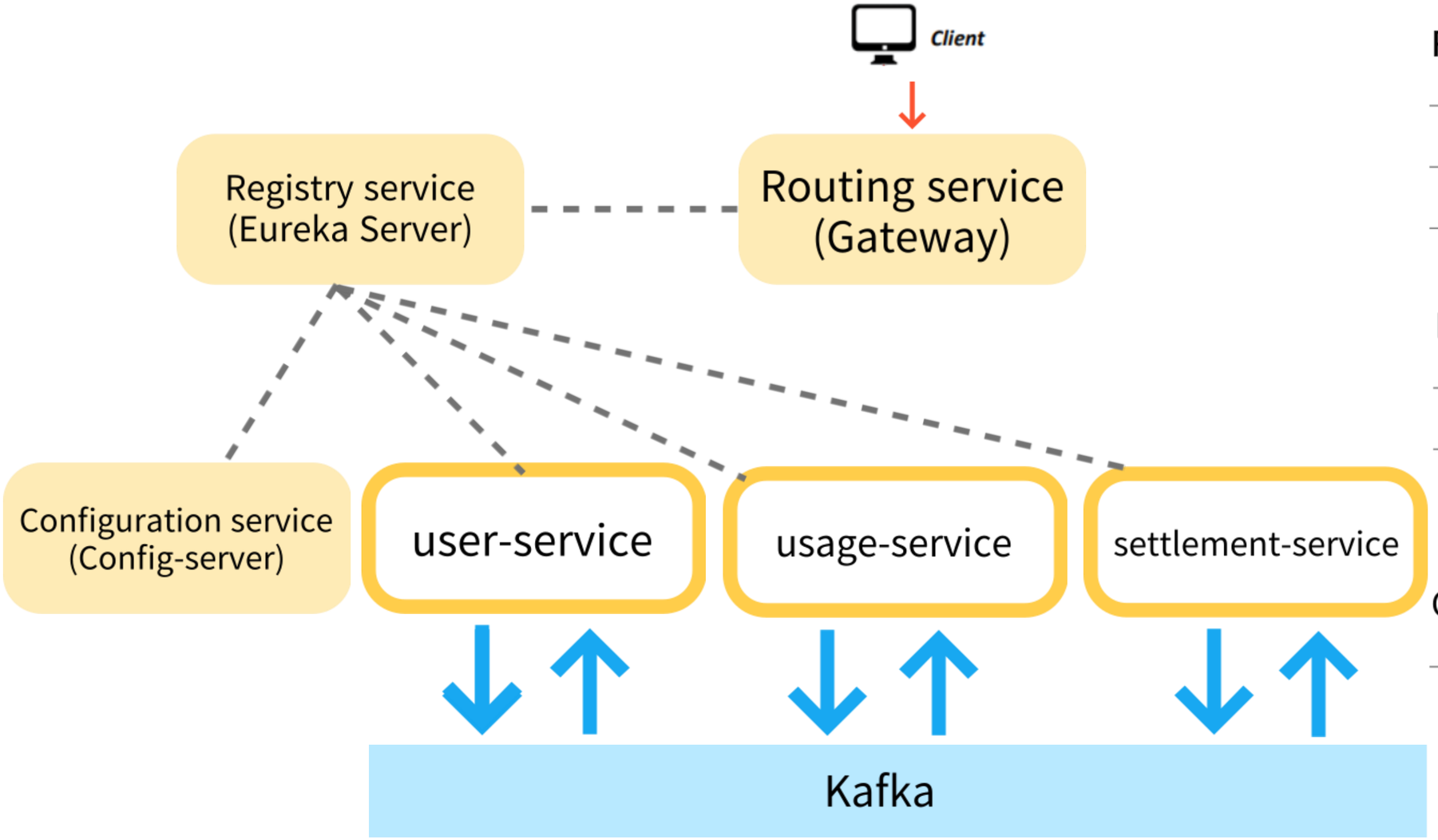
- Low Latency (낮은 지연)

- High Throughput (빠른 전달)

- 대규모 메시지 처리

MSA 실습

프로젝트 소개



Registry service(Eureka Server)

- 모든 Eureka Client 서버가 등록
- Client들은 고유 이름으로 등록
- Client들은 중앙 서버를 통해서 서로 통신

Routing service(Gateway)

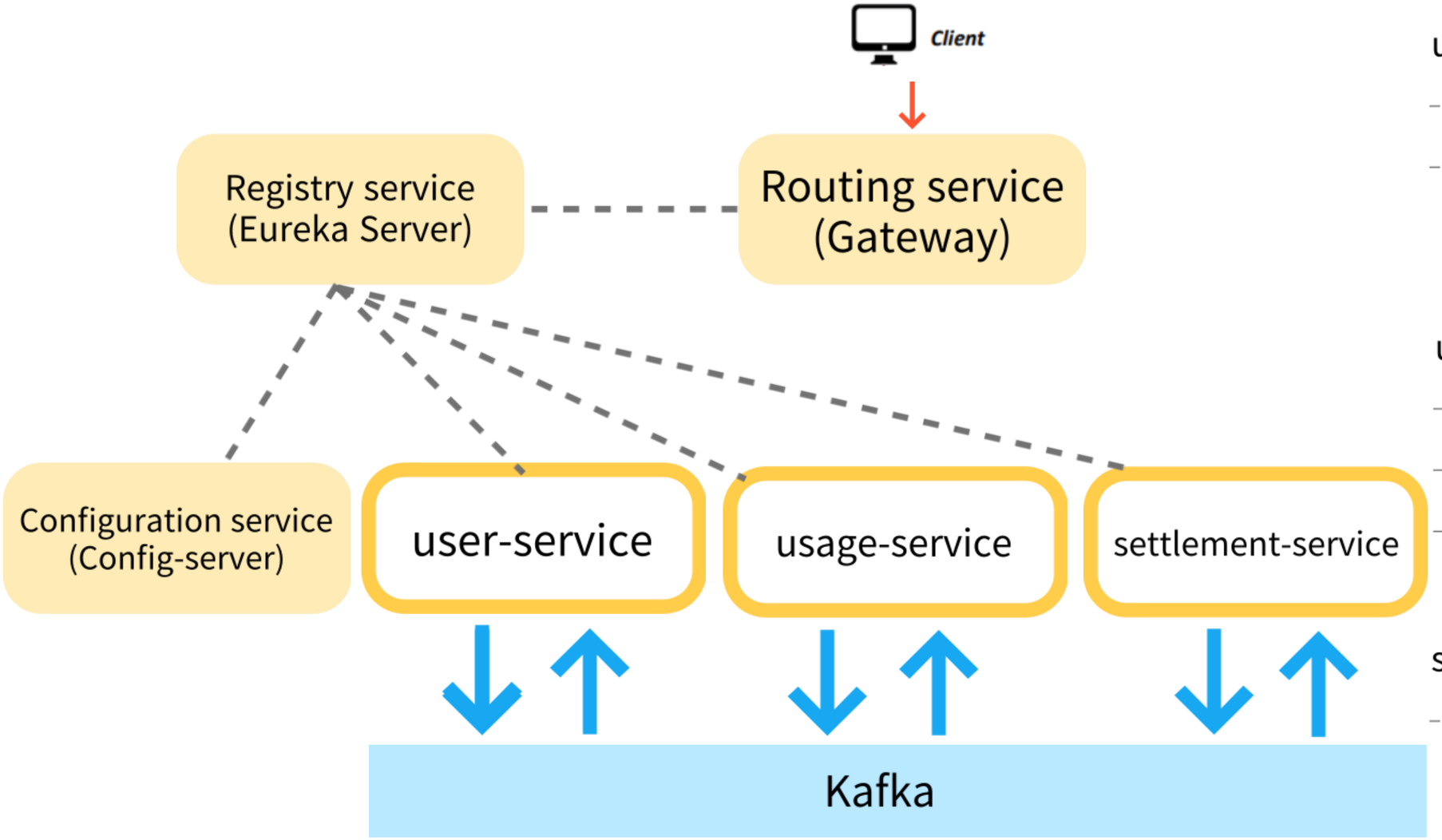
- 사용자의 요청을 받는 영역
- url 과 맵팽되는 eureka client 이름을 Eureka server에서 찾아 요청을 전달

Configuration service(Config-server)

- 서비스들의 설정 정보를 중앙 관리

MSA 실습

프로젝트 소개



user-service

- 사용자 가입 / 요금제 가입
- 사용자가 요금제 가입 시, 정산 서비스에 사용자, 요율 정보 전달

usage-service

- 사용자의 사용 내역 기록
- 내역이 추가될 때마다 정산 서비스 갱신
- kafka connect로 사용 내역 백업

settlement-service

- 사용자의 총 사용 금액 조회