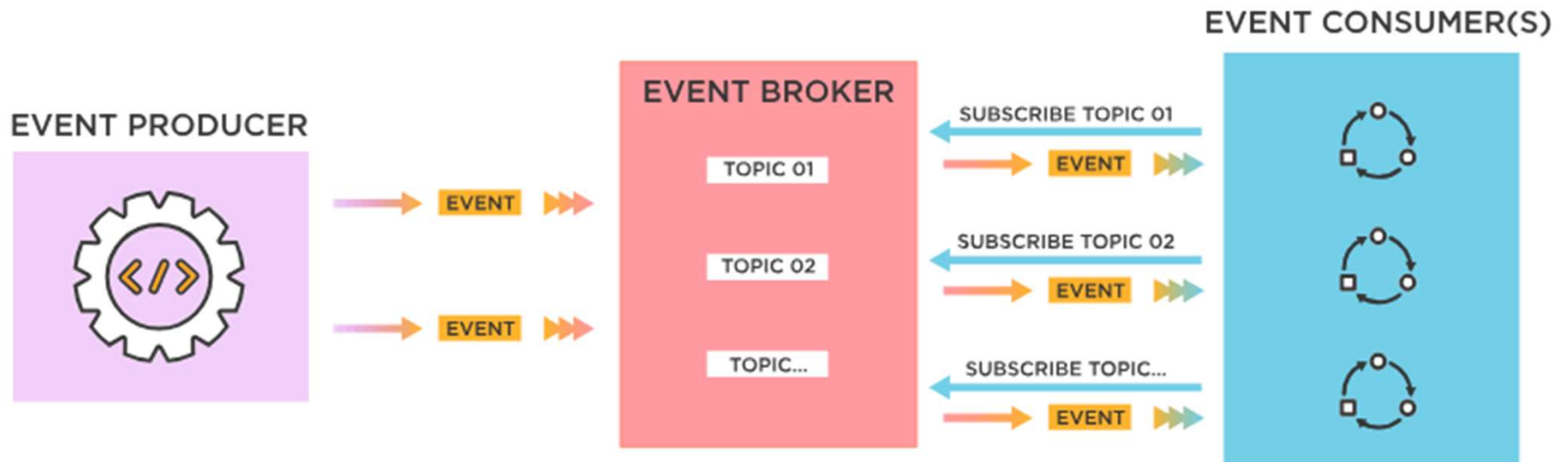


# **Event Driven Architecture**

## **적용하기**

# Event Driven Architecture

- 분산된 시스템에서 이벤트를 발행하고 구독자는 발행된 이벤트를 처리하는 방식의 아키텍처
- 이벤트는 상태 변경을 의미. 예를 들어, 회원가입, 회원정보수정, 주문, 주문취소 등이 해당
- 발행자와 구독자 간의 느슨한 결합을 촉진. 이로 인해 시스템 확장, 업데이트, 독립적인 배포가 용이



# Event Driven Architecture 적용 사례

WOOWACON 2023

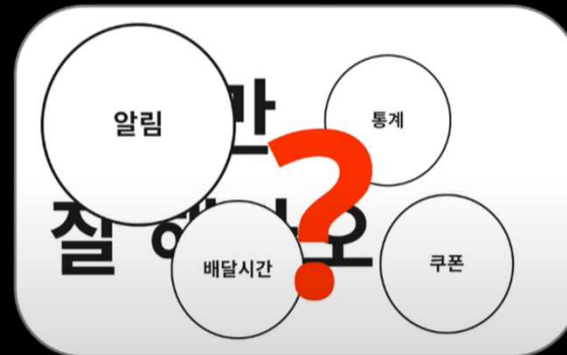


## 왜 이벤트 기반 아키텍처를 선택했는가?

WOOWACON 2023

배달 시스템의 복잡도 증가

배달



- 배달만 잘 수행하기를 기대했지만...
- 점점 더 커져만 가는 배달

## Event Driven Architecture 적용 사례 (계속)

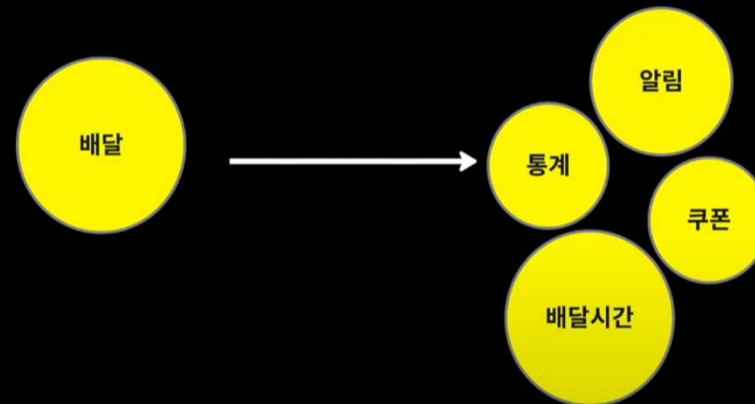
WOOWACON 2023



### 왜 이벤트 기반 아키텍처를 선택했는가?

WOOWACON 2023

강한 일관성



- 배달이 변경되었을 때, 관련 기능도 **“동시에”** 반영되어야 한다.

## Event Driven Architecture 적용 사례 (계속)

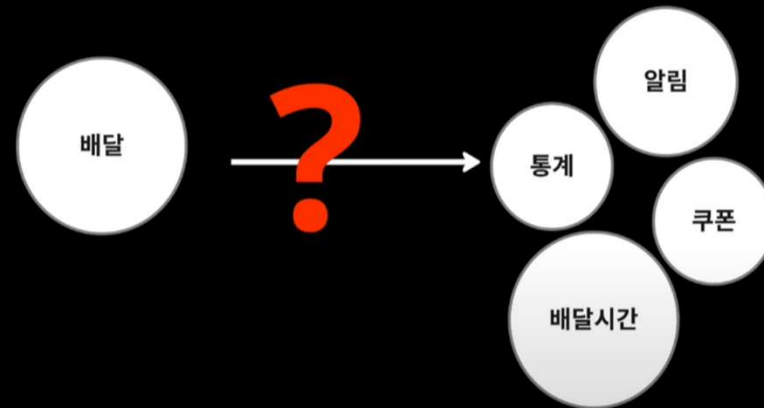
WOOWACON 2023



### 왜 이벤트 기반 아키텍처를 선택했는가?

WOOWACON 2023

시스템을 분리하자



- **배달은 배달만 잘 수행하자**
- **대부분 기능은 배달과 강한 일관성을 필요로하지 않는다.**

강한 일관성 필요 예시 : 항공편의 좌석 예약 (동기식 사용 必)  
약한 일관성 필요 예시 : 배달이 늦을 시 쿠폰 발급 (비동기식 사용해도 무관)  
→ 다음을 잘 판단하여 기술 적용 必

## Event Driven Architecture 적용 사례 (계속)

WOOWACON 2023



왜 이벤트 기반 아키텍처를 선택했는가?

WOOWACON 2023

구현을 하려 보니...

**이벤트**는 어떤 정보를  
가지고 있어야 할까?

## Event Driven Architecture 적용 사례 (계속)

WOOWACON 2023



### 이벤트의 구성요소

WOOWACON 2023

행위를 표현하는 방법

대상

행동

정보

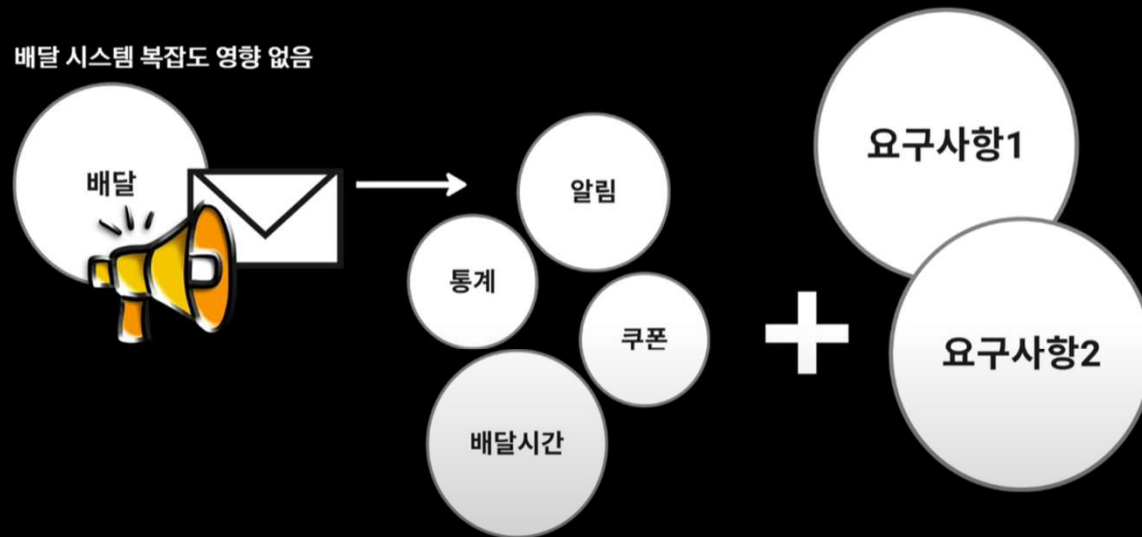
시간



## 이벤트 기반 아키텍처 적용 후

WOOWACON 2023

배달 시스템 복잡도 영향 없음







## 이벤트 기반 아키텍처 적용 후

WOOWACON 2023

이벤트 데이터 무분별한 추가 주의



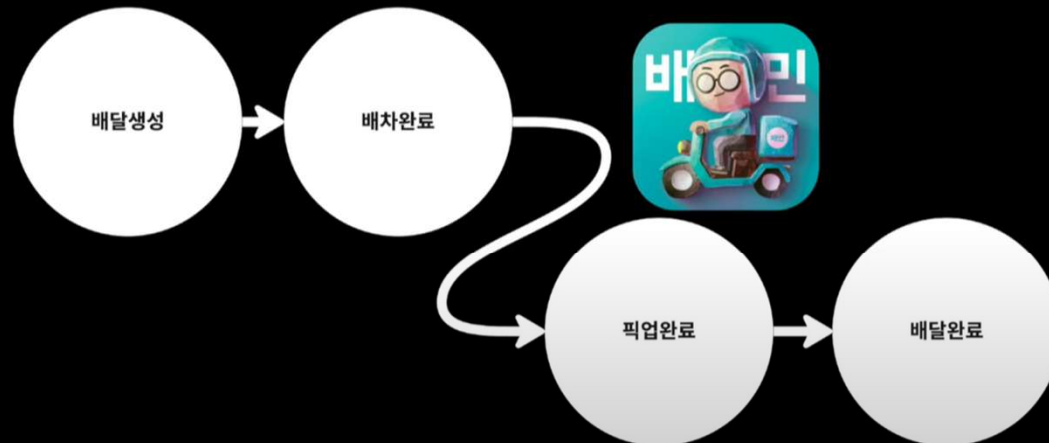
- 행위자 기반의 데이터 정의 필요
- 소비자 요구사항에 대한 무분별한 데이터 추가 주의



## 이벤트 기반 아키텍처 적용 후

WOOWACON 2023

이벤트의 순서가 중요하다



모든 경우에 Event Driven 방법을 사용하는 것은 아님  
ex. N 개의 서비스를 하나의 트랜잭션으로 처리하는 경우 (동기식 처리를 필요로 하는 경우)

# Kafka

## - Kafka는

- 대용량 데이터 스트림을 안정적으로 처리하고 관리하기 위한 분산 스트리밍 플랫폼
- Pub-Sub 모델의 메시지 큐 형태로 동작한다.

## - Kafka 용어

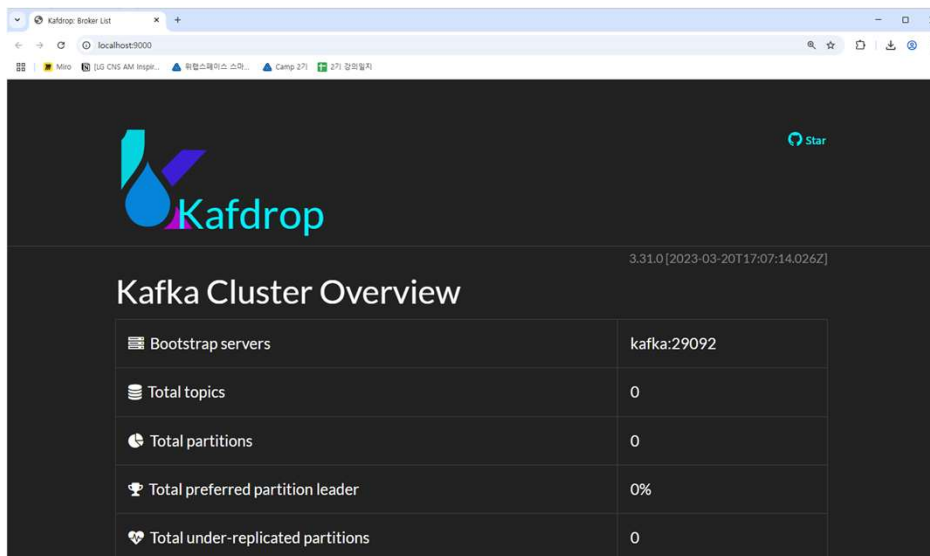
- 브로커(Broker): 아파치 카프카 애플리케이션이 설치되어 있는 서버 또는 노드
- 토픽 (Topic) : 프로듀서가 컨슈머로 보낸 자신들의 메시지를 구분하기 위한 고유의 이름. 이벤트 이름
- 프로듀서 (Producer) : 메시지를 생산하여 브로커의 토픽 이름으로 보내는 애플리케이션 이벤트 송신 측
- 컨슈머 (Consumer) : 브로커의 토픽을 구독하여 저장된 메시지를 가져가서 처리하는 애플리케이션 이벤트 수신 측

## - Kafka 특징

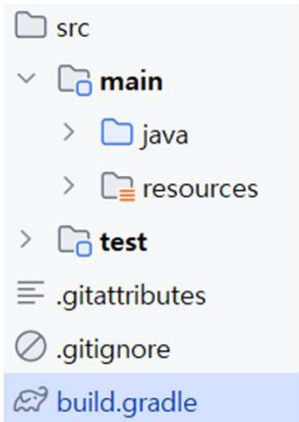
- 전송 보장 (보장성을 위해 같은 이벤트가 2번 오는 경우가 있음. 개별 처리 필요한 경우 존재.)
- 순서 보장
- 디스크에 메시지를 저장하여 영속성 보장 (기간 지정)

# Kafka 설치

- Kafka docker-compose.yml 파일을 C:/Server/kafka 디렉토리에 다운로드
- C:/Server/kafka 디렉토리에서 docker compose up -d 명령어 실행  
C:\Server\kafka> docker compose up -d
- 브라우저에서 localhost:9000 접속하여 설치 확인



## 프로젝트 의존성 추가 – user, post, alim 프로젝트



```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.cloud:spring-cloud-starter-loadbalancer'  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'  
    implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'  
    implementation 'org.springframework.boot:spring-boot-starter-validation'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    runtimeOnly 'com.mysql:mysql-connector-j:8.4.0'  
  
    implementation 'org.springframework.kafka:spring-kafka'  
    testImplementation 'org.springframework.kafka:spring-kafka-test'  
  
    implementation 'io.jsonwebtoken:jjwt-api:0.12.5'  
    runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.12.5'  
    runtimeOnly 'io.jsonwebtoken:jjwt-gson:0.12.5'  
  
    compileOnly 'org.projectlombok:lombok'  
    annotationProcessor 'org.projectlombok:lombok'  
  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'  
}
```

## 프로젝트 설정 추가 – user, post, alim 프로

가  
commit  
가

+

+

+

```
spring:  
  application:  
    name: backend-user
```

```
kafka:
```

```
  listener:
```

```
    ack-mode: manual_immediate
```

Consumer가 메시지 처리에 대한 ack을 어떻게 할지에 대한 방식을 정의  
(전송 보장을 위한 ACK. (ex) TCP 통신)

```
  consumer:
```

```
    group-id: ${spring.application.name}
```

```
    key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
```

```
    value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
```

```
    enable-auto-commit: false
```

```
    auto-offset-reset: latest
```

새로운 서비스로 시작 시, offset을 "마지막" 이벤트으로 설정

```
    max-poll-records: 10
```

```
  properties:
```

```
    spring.json.trusted.packages: "*" 어플리케이션 내의 신뢰 가능한 패키지 지정
```

```
    spring.json.use.type.headers: false # 헤더의 타입 정보 무시
```

```
  producer:
```

```
    key-serializer: org.apache.kafka.common.serialization.StringSerializer
```

```
    value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
```

```
  properties:
```

```
    spring.json.add.type.headers: false # 타입 헤더 추가 비활성화
```

application.yml

## 프로젝트 설정 추가 – user, post, alim 프로젝트 (계속)

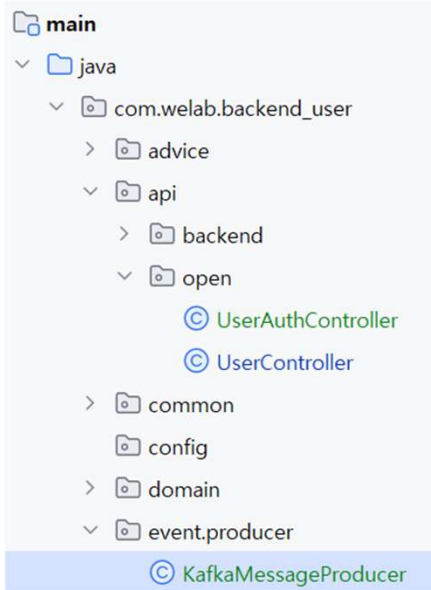
```
server:
  port: 8081

spring:
  datasource:
    url: jdbc:mysql://localhost:13306/user?serverTimezone=UTC&useSSL=true&autoReconnect=true&useUnicode=true&characterEncoding=utf-8
    username: user
    password: 1234
    driver-class-name: com.mysql.cj.jdbc.Driver
  hikari:
    connection-test-query: SELECT 1 # HikariCP 유효성 검사 추가
    validation-timeout: 5000
  jpa:
    hibernate:
      ddl-auto: create # 오직 테스트 환경에서만
      generate-ddl: true # 오직 테스트 환경에서만
      show-sql: true
      open-in-view: false
  kafka:
    bootstrap-servers: localhost:9092
```

...

application-local.yml

# User 프로젝트 코드 작성 - KafkaMessageProducer

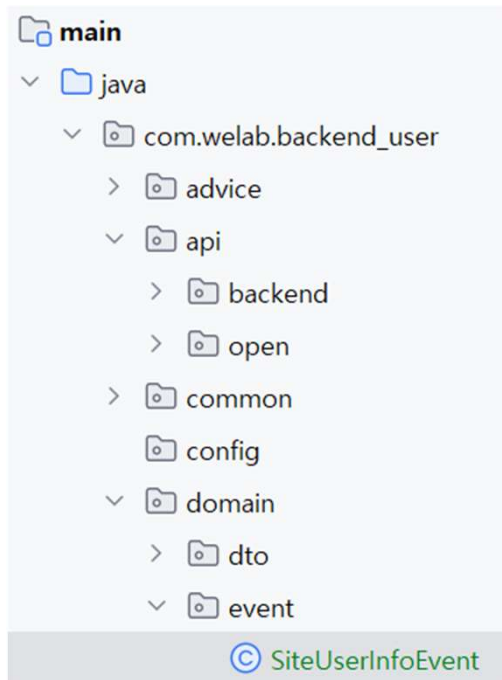


```
@Slf4j
@Service
@RequiredArgsConstructor
public class KafkaMessageProducer {
    private final KafkaTemplate<String, Object> kafkaTemplate;

    public void send(String topic, Object message) {
        kafkaTemplate.send(topic, message);
    }
}
```



# User 프로젝트 코드 작성 - SiteUserInfoEvent



@Getter

@Setter

public class SiteUserInfoEvent {

public static final String Topic = "userinfo"; 이벤트 명

private String action; 이벤트 구성 요소 : 행동

private String userId;

이벤트 구성 요소 : 정보

private String phoneNumber;

private LocalDateTime eventTime; 이벤트 구성 요소 : 시간

```
public static SiteUserInfoEvent fromEntity(String action, SiteUser siteUser) {  
    SiteUserInfoEvent message = new SiteUserInfoEvent();
```

```
    message.action = action;
```

```
    message.userId = siteUser.getUserId();
```

```
    message.phoneNumber = siteUser.getPhoneNumber();
```

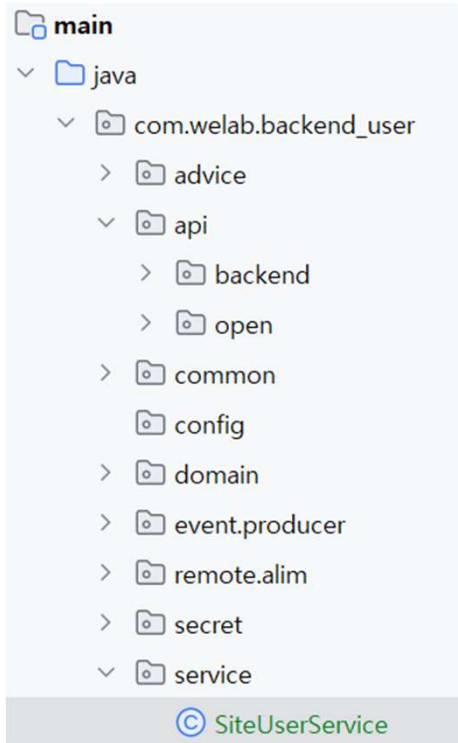
```
    message.eventTime = LocalDateTime.now();
```

```
    return message;
```

```
}
```

```
}
```

# User 프로젝트 코드 작성 – SiteUserService 수정



```
@Transactional
public void registerUser(SiteUserRegisterDto registerDto) {
    SiteUser siteUser = registerDto.toEntity();


    siteUserRepository.save(siteUser);

    SiteUserInfoEvent message = SiteUserInfoEvent.fromEntity("Create", siteUser);


    kafkaMessageProducer.send(SiteUserInfoEvent.Topic, message);
}
```

API 호출 대신 Kafka로 변경

# Kafdrop 이용한 메시지 전송 확인

Star

## Topic: userinfo

 View Messages

Delete topic

### Overview

# of partitions	1
Preferred replicas	100%
Under-replicated partitions	0
Total size	0
Total available messages	0

### Configuration

No topic-specific configuration

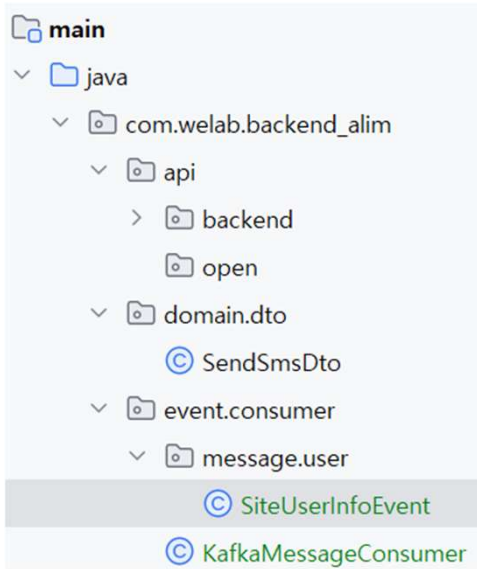
### Partition Detail

Partition	First Offset	Last Offset	Size	Leader Node	Replica Nodes	In-sync Replica Nodes	Offline Replica Nodes	Preferred Leader	Under-replicated
0	0	0	0	0	0	0		Yes	No

### Consumers

Group ID	Combined Lag
----------	--------------

# Alim 프로젝트 코드 작성 - SiteUserInfoEvent



```
@Getter
@Setter
public class SiteUserInfoEvent {
    public static final String Topic = "userinfo"; 이벤트 명

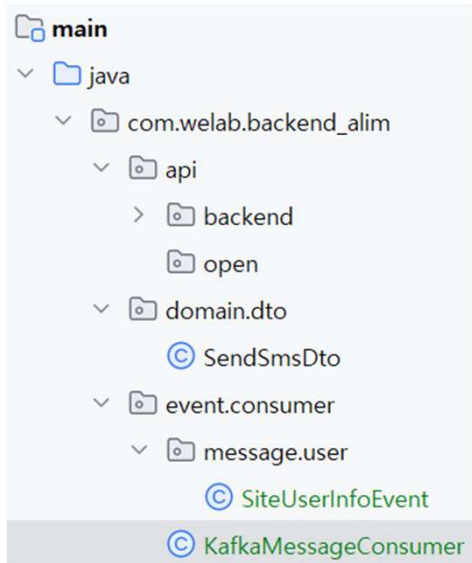
    private String action; 이벤트 행동

    private String userId; 이벤트 정보

    private String phoneNumber;

    private LocalDateTime eventTime; 이벤트 시간
}
```

# Alim 프로젝트 코드 작성 - KafkaMessageConsumer



```
@Slf4j
@Service
@RequiredArgsConstructor
public class KafkaMessageConsumer {
    @KafkaListener(
        topics = SiteUserInfoEvent.Topic,
        properties = {
            JsonSerializer.VALUE_DEFAULT_TYPE
            + ":com.welab.backend_alim.event.consumer.message.user.SiteUserInfoEvent"
        })
    void handleSiteUserInfoEvent(SiteUserInfoEvent event, Acknowledgment ack) {
        log.info("SiteUserInfoEvent 처리. userId={}", event.getUserId());

        ack.acknowledge();
    }
}
```

Deserialize ,

ACK