

전자정부 표준프레임워크

MSA 템플릿

스프링 클라우드 스트림을 사용한

이벤트 기반 아키텍처



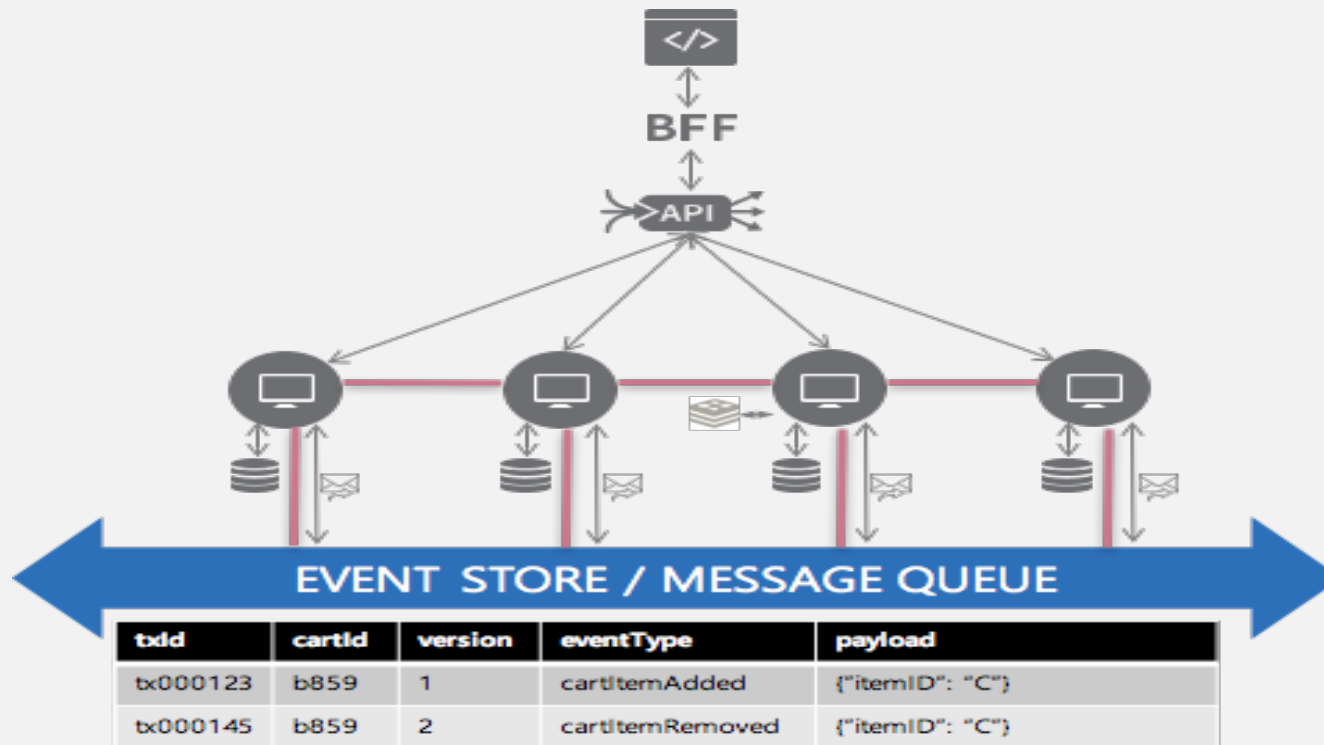
Contents

1. _ Event Driven Architecture
2. _ Event Driven MSA 템플릿
3. _ Event Stream 구현



Event Driven Architecture

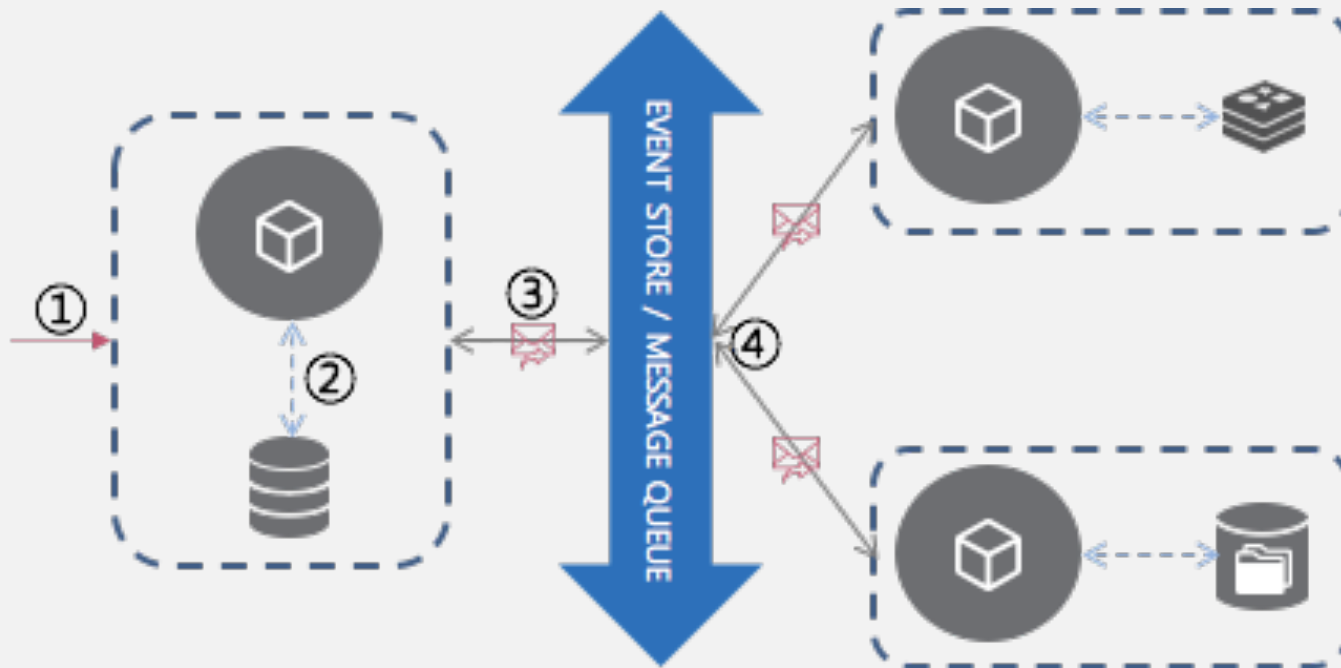
분산된 시스템 간 이벤트를 생성, 발행하고 발행된 이벤트를 필요로 하는 수신자에게 전송하고 이벤트를 수신한 수신자가 이벤트를 처리하는 형태의 시스템 아키텍처



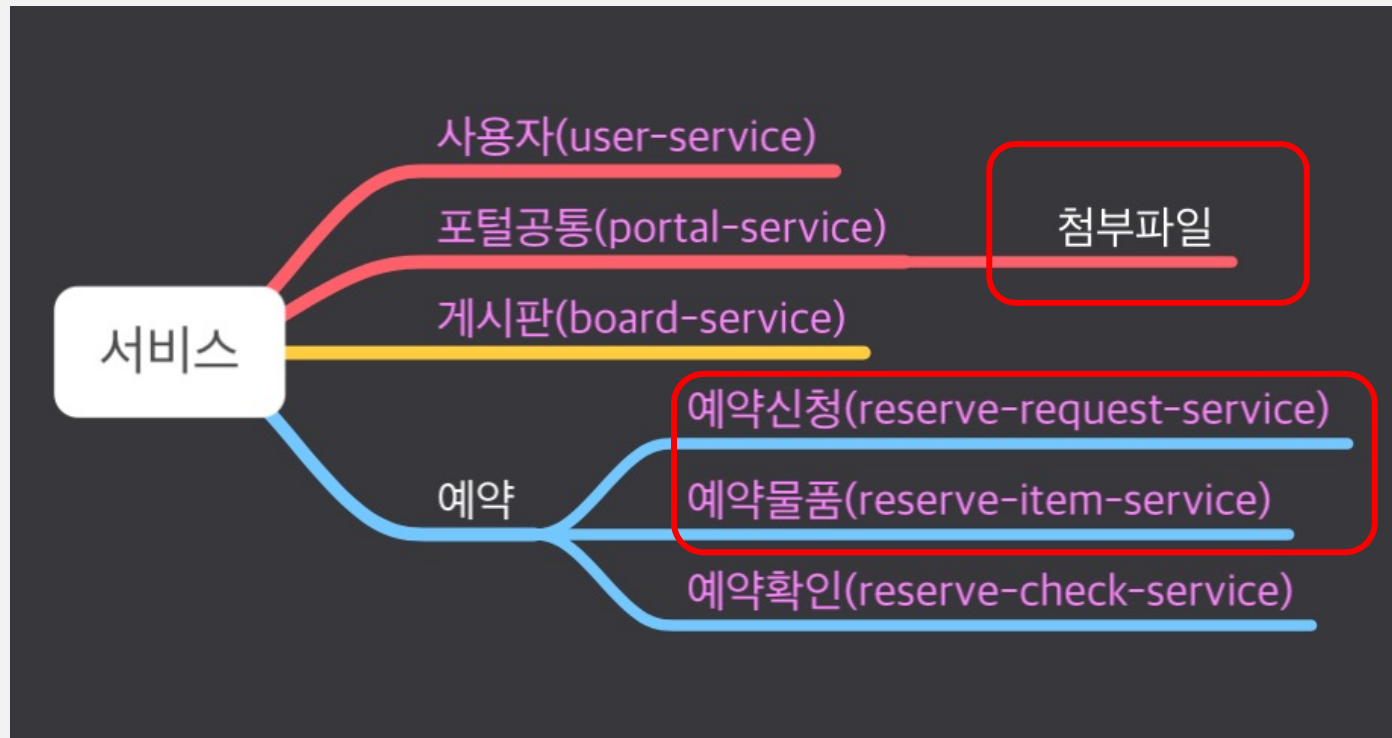
<https://jaehun2841.github.io/2019/06/23/2019-06-23-event-driven-architecture>

Event Driven MicroService

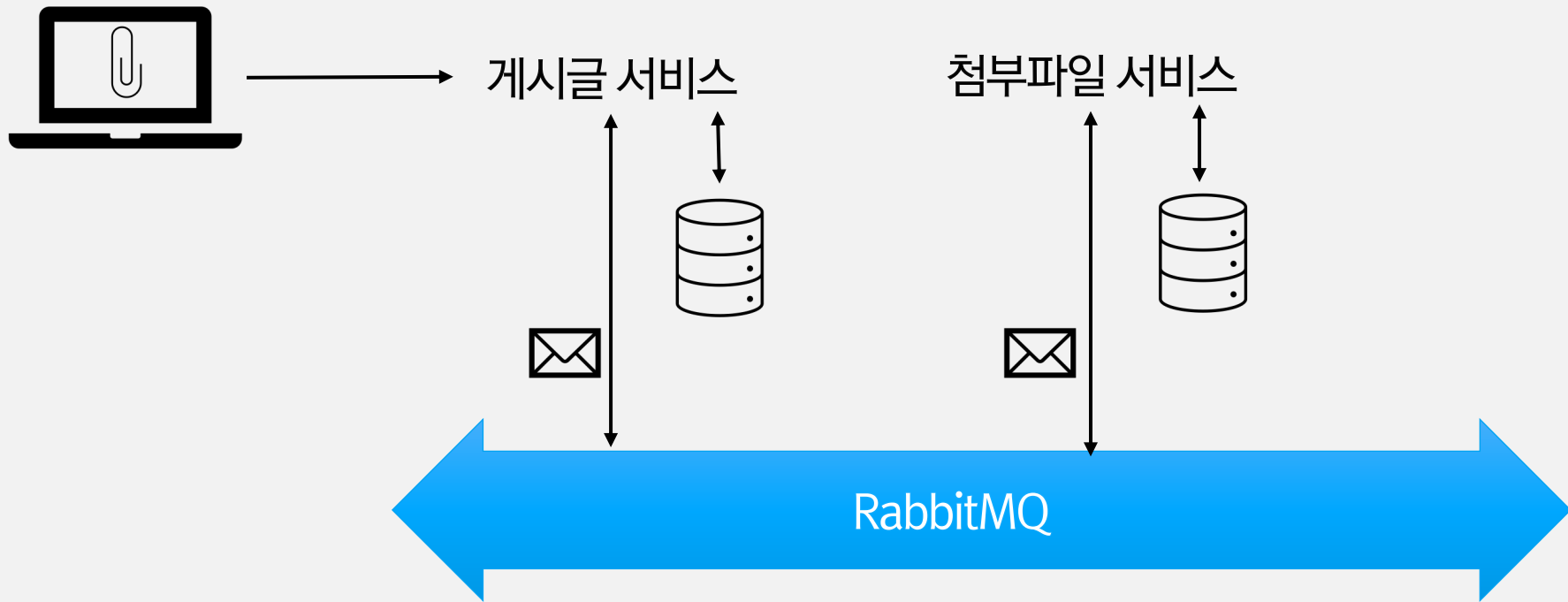
MSA가 적용된 시스템에서 이벤트 발생시 해당 이벤트 로그를 보관하고 이를 기반으로 동작하며, 비동기 통신을 통해 시스템 내 통합(integration)을 수행하는 아키텍처.



● 예약 서비스, 첨부파일 Entity 등록

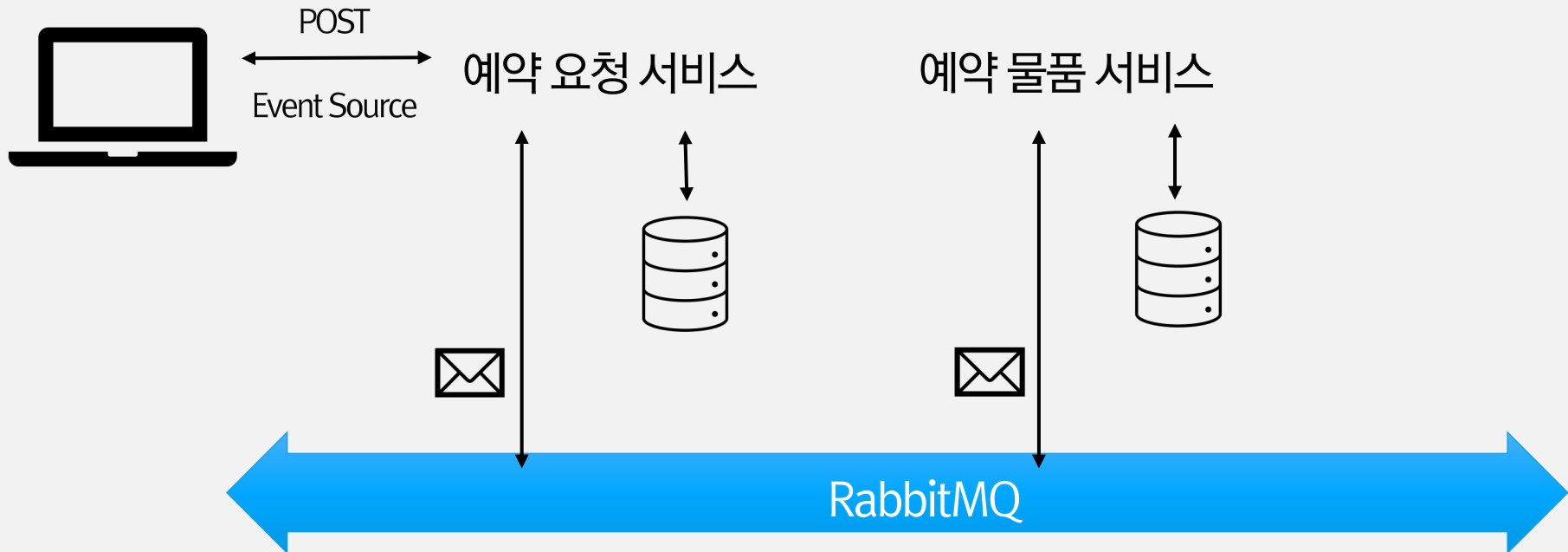


Business Flow – 첨부파일 Entity 정보 update



exchange	queue	payload
attachment-entity.topic	attachment-entity.topic	{entity: post, entityId: 1}

Business Flow – 실시간 예약 시 예약 물품 재고 변경

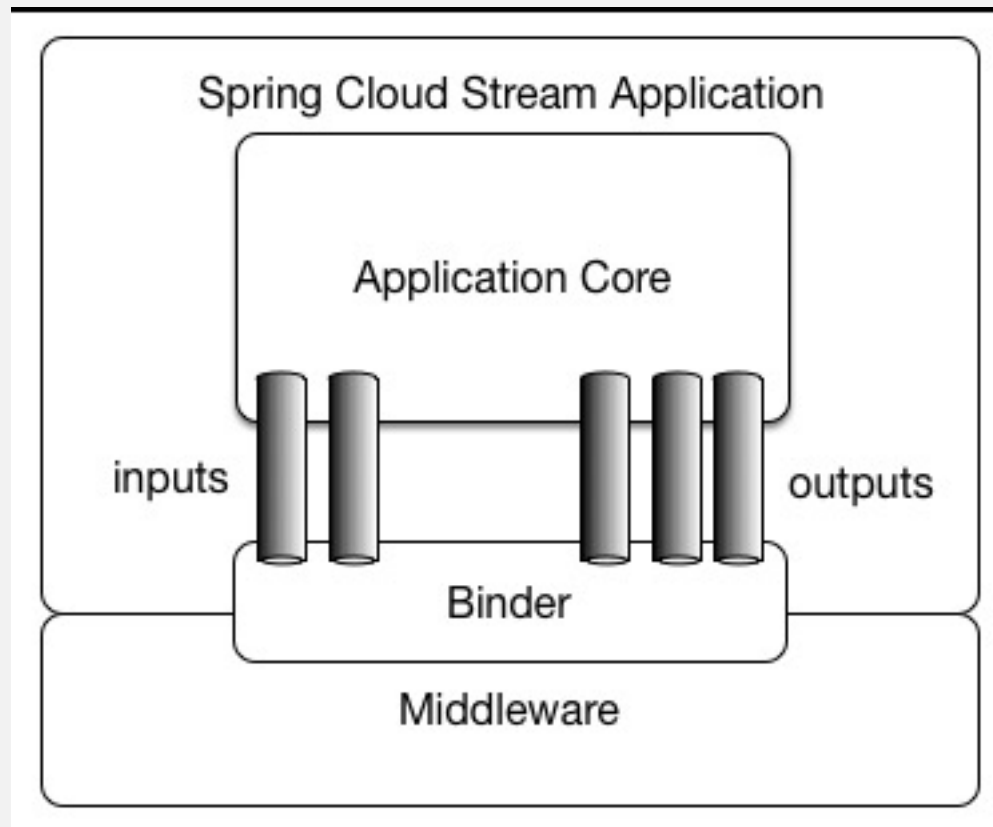


exchange	queue	payload
reserve-request.topic	reserve-request.topic	{itemId:1, reserveld:reserve_1, qty:20}
Inventory-updated.topic	Inventory-updated.topic	{reserveld:reserve_1, isSuccess:true}
success-or-not.direct	reserve_1	true

3. Event Stream 구현

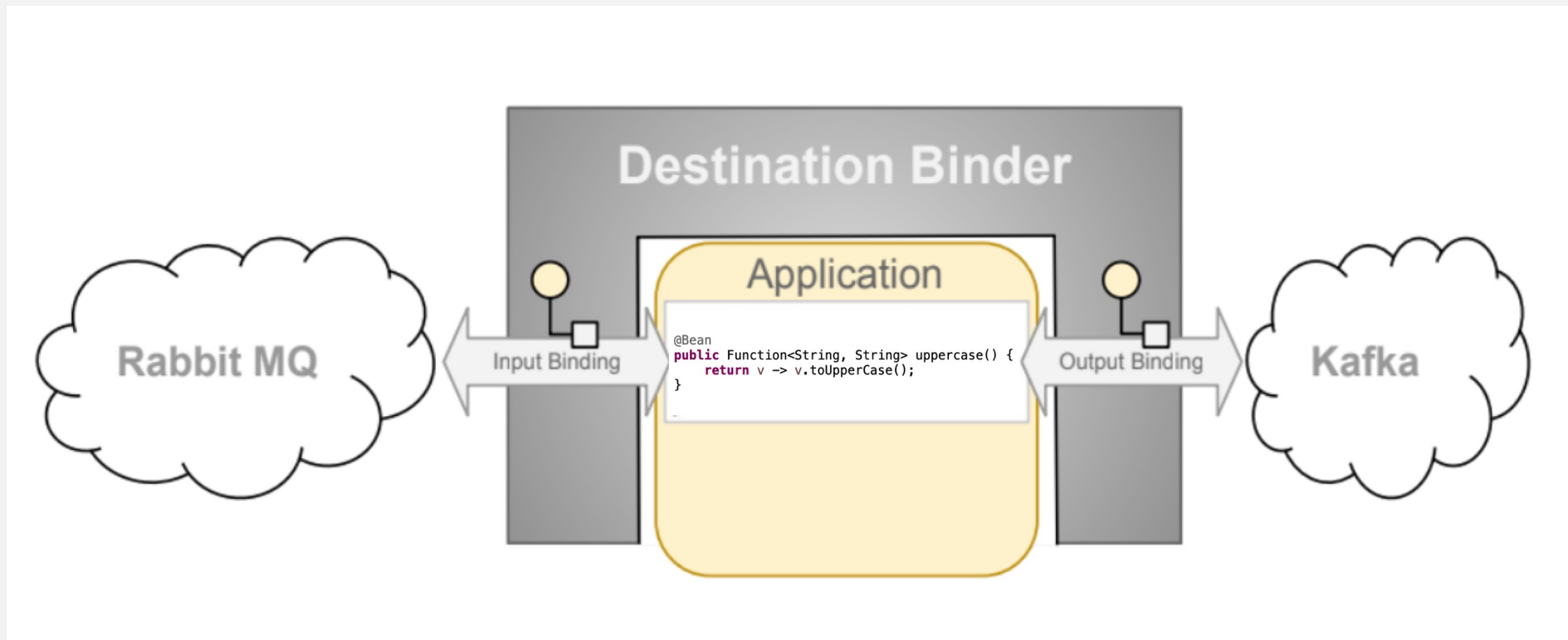
Spring Cloud Stream

- 외부 시스템에 연결할 수 있는 애플리케이션을 신속하게 구축할 수 있는 경량의 마이크로 서비스 프레임 워크
- Apache Kafka 또는 RabbitMQ 등을 사용하여 Spring Boot 어플리케이션과 메시지를 보내고 받음
- 이벤트 중심 마이크로 서비스를 구축하기 위한 프레임워크



<https://docs.spring.io/spring-cloud-stream/docs/current/reference/html/spring-cloud-stream.html#spring-cloud-stream-reference>

Spring Cloud Stream – Main Concept



<https://docs.spring.io/spring-cloud-stream/docs/current/reference/html/spring-cloud-stream.html#spring-cloud-stream-reference>

Spring Cloud Stream – v3.1.3

build.gradle

```
dependencies {  
    //messaging  
    implementation 'org.springframework.cloud:spring-cloud-stream '  
    implementation 'org.springframework.cloud:spring-cloud-stream-binder-rabbit '  
    implementation 'org.springframework.boot:spring-boot-starter-amqp' //Dynamic Queue 생성 시  
}
```

```
spring:  
  cloud:  
    bus:  
      destination: springCloudBus  
  stream:  
    function: # Consumer, Supplier, Function 타입으로 정의한 Bean name  
      definition: inventoryUpdated;busConsumer  
    bindings:  
      busConsumer-in-0: # Spring Cloud Bus에 대한 consumer binding  
        destination: ${spring.cloud.bus.destination}  
      reserveRequest-out-0: # 예약 요청후 물품 재고업데이트 이벤트에 대한 producer binding  
        destination: reserve-request.topic # exchange name  
        group: reserved  
      inventoryUpdated-in-0: # 예약 요청후 물품 재고업데이트 결과에 이벤트에 대한 consumer binding  
        destination: inventory-updated.topic # exchange name  
        group: reserved
```

예약 신청 서비스 Producer 및 Dynamic Queue 생성

ReserveService.java

```
...
private final StreamBridge streamBridge;
private final AmqpAdmin amqpAdmin;
...
public Mono<ReserveResponseDto> saveForEvent(ReserveSaveRequestDto saveRequestDto) {
    return create(saveRequestDto)
        .flatMap(reserveResponseDto ->
            Mono.fromCallable(() -> {
                //예약 저장 후 해당 id로 queue 생성
                Exchange ex = ExchangeBuilder.directExchange(GlobalConstant.SUCCESS_OR_NOT_EX_NAME)
                    .durable(true).build();
                amqpAdmin.declareExchange(ex);

                Queue queue = QueueBuilder.durable(reserveResponseDto.getReserveld()).build();
                amqpAdmin.declareQueue(queue);

                Binding binding = BindingBuilder.bind(queue)
                    .to(ex)
                    .with(reserveResponseDto.getReserveld())
                    .noargs();
                amqpAdmin.declareBinding(binding);

                log.info("Biding successfully created");

                streamBridge.send("reserveRequest-out-0", reserveResponseDto);

                return reserveResponseDto;
            }).subscribeOn(Schedulers.boundedElastic())
        );
}
...
```

예약 신청 서비스 Consumer & Direct Message 발행

ReserveEventConfig.java

```
@Bean
public Consumer<Message<RequestMessage>> inventoryUpdated() {
    return message -> {
        log.info("receive message: {}, headers: {}", message.getPayload(), message.getHeaders());
        if (message.getPayload().getIsItemUpdated()) {
            reserveService.updateStatus(message.getPayload().getReserveld(), ReserveStatus.APPROVE).subscribe();
        } else {
            reserveService.delete(message.getPayload().getReserveld()).subscribe();
        }
    }

    RabbitTemplate rabbitTemplate = rabbitTemplate(connectionFactory);
    rabbitTemplate.convertAndSend(GlobalConstant.SUCCESS_OR_NOT_EX_NAME,
        message.getPayload().getReserveld(), message.getPayload().getIsItemUpdated());
};

@Bean
public RabbitTemplate rabbitTemplate(final ConnectionFactory connectionFactory) {
    final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
    rabbitTemplate.setMessageConverter(messageConverter());
    return rabbitTemplate;
}

@Bean
public Jackson2JsonMessageConverter messageConverter() {
    return new Jackson2JsonMessageConverter();
}
```

예약 신청 결과 Server Sent Events

ReserveApiController.java

```
@CrossOrigin()
@GetMapping(value = "/api/v1/requests/direct/{reserveld}", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
public Flux<String> receiveReservationResult(@PathVariable String reserveld) {
    MessageListenerContainer mlc = messageListenerContainerFactory.createMessageListenerContainer(reserveld);
    Flux<String> f = Flux.create(emitter -> {
        mlc.setupMessageListener((MessageListener) m -> {
            String qname = m.getMessageProperties().getConsumerQueue();
            log.info("message received, queue={}", qname);

            if (emitter.isCancelled()) {
                log.info("cancelled, queue={}", qname);
                mlc.stop();
                return;
            }

            String payload = new String(m.getBody());
            log.info("message data = {}", payload);
            emitter.next(payload);

            log.info("message sent to client, queue={}", qname);
        });
    });
}
```

...계속

예약 신청 결과 Direct Message 수신 API

ReserveApiController.java

...계속

```
emitter.onRequest(v -> {
    log.info("starting container, queue={}", reserveld);
    mlc.start();
});

emitter.onDispose() -> {
    log.info("on dispose, queue={}", reserveld);
    mlc.stop();
    amqpAdmin.deleteQueue(reserveld);
});

log.info("container started, queue={}", reserveld);
});

return Flux.interval(Duration.ofSeconds(5))
    .map(v -> {
        log.info("sending keepalive message...");
        return "no news is good news";
    })
    .mergeWith(f)
    .delayElements(Duration.ofSeconds(5));
}
```

3. Event Stream 구현

프론트엔드 Event Source

ReserveEventSource.tsx

```
useEffect(() => {
  let eventSource: EventSource = null
  if (data) {
    eventSource =
      new EventSource(`${SERVER_API_URL}${reserveService.requestApiUrl}/direct/${data.reserveId}`)

    eventSource.onmessage = event => {
      if (event.data !== 'no news is good news') {
        setSuccess(event.data)
        eventSource.close()
      }
    }
    eventSource.onerror = err => {
      console.error('EventSource failed:', err)
      eventSource.close()
    }
  }

  return () => {
    if (eventSource) {
      eventSource.close()
      eventSource = null
    }
  }
}, [data])
```