

Event Driven MicroService Hands-On

V.1.0.0

2018.10



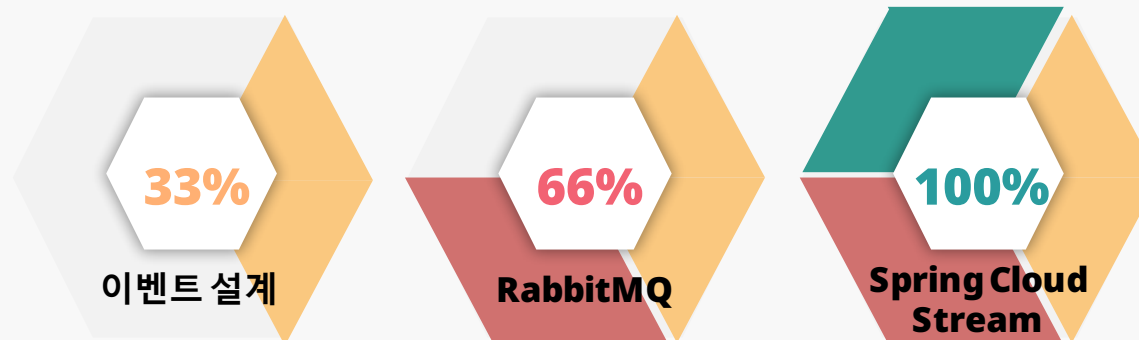
Copyright©2018 by SK TECH TRAINING CENTER All rights reserved.



Table of Contents

01 | Awesome-shoping 샘플 소개

02 | Awesome-shoping 구현 사전 준비



03 | EDM 구현 따라하기

Part 01. Awesome-shoping 샘플 소개



Awesome-shopping 개발 환경 세팅

Awesome-shopping 개발환경을 구축합니다.

- 시스템 요구 사항

Name	Servlet	Java	Apache Maven
Tomcat 8.5	3.1	Java 8	3.2 +

- 통합 개발 환경(IDE)

- Spring Tool Suite 3.9.2.RELEASE : <https://spring.io/tools/>
- JDK 8 : <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

- Docker 설치

OS	링크
Windows10	https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows
Windows7	https://docs.docker.com/toolbox/overview/#whats-in-the-box
Mac	https://docs.docker.com/docker-for-mac/install/

Awesome-shopping 개발 환경 세팅

Awesome-shopping 프로젝트를 Git에서 Clone 받습니다.

Git 주소

- <https://github.com/cloudz-labs/awesome-shopping-handson.git>

Awesome-shoping 개발 환경 세팅

Awesome-shoping 프로젝트에 K8S 상 RabbitMQ의 port 정보를 변경합니다.

```
spring:
  cloud:
    stream:
      binders:
        rabbitmq:
          type: rabbit
          environment:
            spring:
              rabbitmq:
                host: 169.56.106.154
                port: {개별로 전달받은 port 정보 기입}
                username: labs
                password: awesome-rabbitmq
```

Awesome-shopping 개발 환경 세팅

Awesome-shopping 프로젝트에 h2-console에 접속하여 data의 변경사항을 확인할 수 있습니다.

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:mediadb

User Name: sa

Password:

Connect

Test Connection

jdbc:h2:mem:mediadb
 + ACCOUNTS
 + CATEGORIES
 + CONTENTS
 + EPISODES
 + PROFILES
 + PROMOTIONS
 + INFORMATION_SCHEMA
 + Sequences
 + Users
 H2 1.4.196 (2017-06-10)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM CATEGORIES |

SELECT * FROM CATEGORIES;

ID	NAME
action	액션 모험
comedy	코미디
drama	드라마 영화
family_kids	가족/아동 영화
horror	공포 영화
korean	한국 영화
romance	로맨스 영화
sf_fantasy	SF 및 판타지 영화
tv	TV 프로그램
winner	수상작

(10 rows, 2 ms)

Awesome-shoping 업무 정의

Awesome-shoping의 업무 list up 및 시각화 자료를 확인합니다.

기능 List

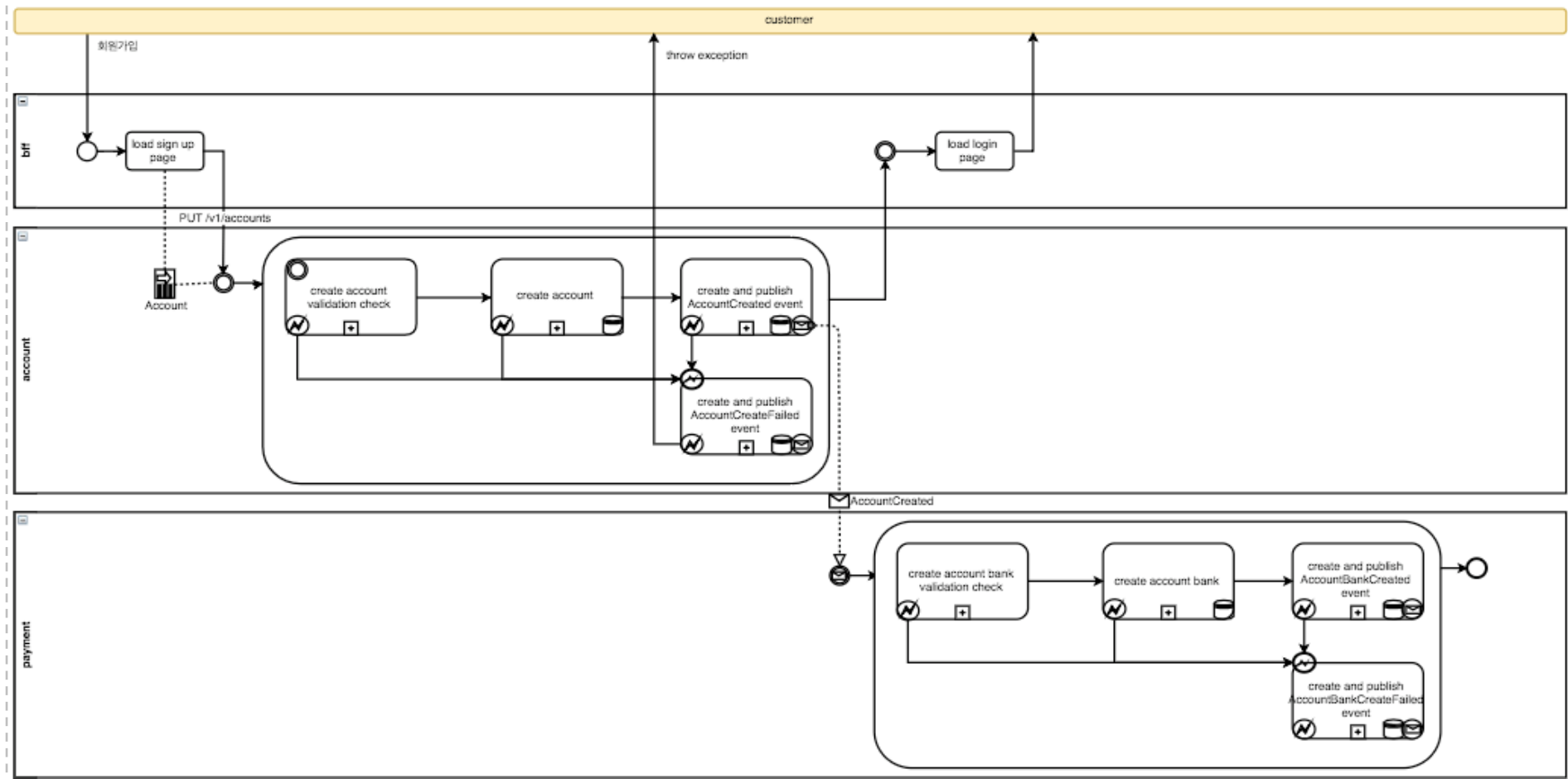
- 회원가입
- 로그인
- 메인 조회
- 카테고리 별 상품 조회
- 할인 중 상품 조회
- 상품 단건 상세 조회
- 장바구니 조회
- 장바구니 추가
- 장바구니 삭제
- 주문하기

STEP 02. 업무 정의

[BACKUP]Awesome-shopping 업무 정의

Awesome-shopping의 회원가입 이벤트 시각화 예시를 확인합니다.

회원가입



Awesome-shopping 구현 Spec

Awesome-shopping의 구현 Spec을 확인합니다.

Implements Spec

- **이벤트 설계**
 - BPMN 2.0 표기법
- **Backing Service**
 - rabbitMQ
 - h2
- **Application**
 - MVC + Pub/Sub Package Structure
 - Spring Boot 2.0.3
 - Dependency
 - spring cloud stream
 - aop
 - web
 - mybatis
 - rabbitMQ
 - h2

Insight - 이벤트 설계/구현

지금까지의 내용을 바탕으로 EDM을 구현하기 위해 기존 방식에서 추가로 이벤트 설계를 고려해야 합니다.

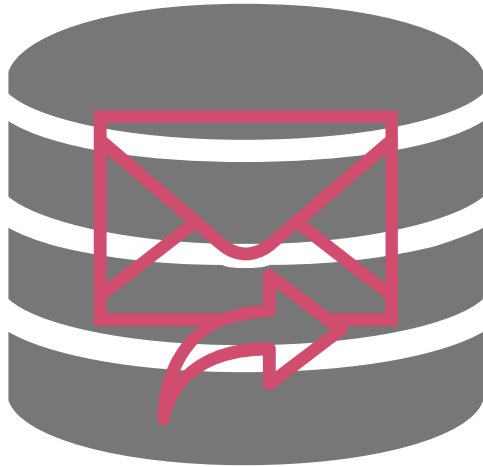


업무에 기반한 의미있는 이벤트 설계

- 정상 시나리오에서 **이벤트 흐름** 도출
- 장애 발생시 **retry/rollback 여부**에 따른 이벤트 흐름 도출
- 이벤트 흐름 **시각화** 방법
- Scaling된 Application 간 경쟁적인 event msg 구독

Insight - 이벤트 스토어 설계/구현

지금까지의 내용을 바탕으로 EDM을 구현하기 위해 기존 방식에서 추가로 고려해야 하는 사항들이 있습니다.



이벤트 스토어 설계/구현

- Application + Backing Service
- 요구사항 별 별도의 뷰 제공
- 메세지 큐 기능
- exactly once 보장
- dlq 기능
- 고가용성
- 내결함성
- throughput
- event message persistent
- retry 설정
- time out 설정
- 이벤트 라우팅 룰/전략
- 확장성

유용한 Backing Service

- NoSQL
- 관계형 Database
- RabbitMQ
- Kafka
- Geode
- CSP – MQ 서비스
- 기타 등등

Insight – Application 설계/구현

지금까지의 내용을 바탕으로 EDM을 구현하기 위해 기존 방식에서 추가로 고려해야 하는 사항들이 있습니다.



Application

•mvc + pub/sub

- project package 구조 정의
- event 보관/발행을 위한 기능 설계/구현

•event msg 설계/구현

- Application 별 event type 도출
- event type 에 대응되는 fail event type도출
- event 먹등 처리 고려

•이벤트 생성+발행 logic 설계/구현

- 데이터의 생성, 변경, 삭제와 이벤트 생성+발행의 일관성 유지 필요

•보상 트랜잭션

- 오류 발생시 fail 이벤트 생성+발행 logic 설계/구현
- transactionId 생성, 전달 방법 설계/구현

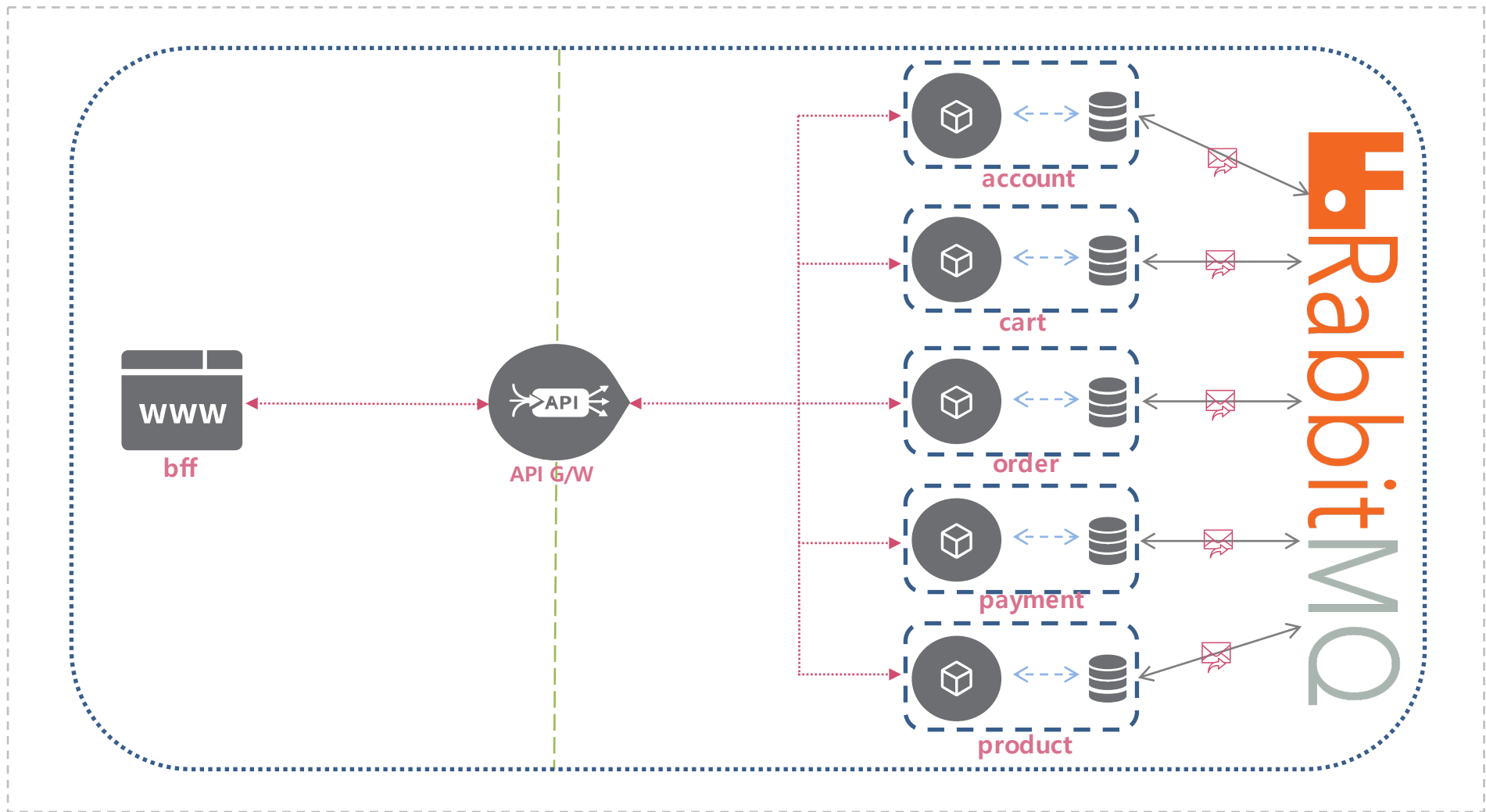
[Backup]Requirements Coverage

Event Driven MicroService 의 새로운 요구사항과 달성 방법

요구사항	달성 방법
기능별 별도의 뷰 제공	 
메세지 큐 기능	
Exactly Once 보장	 
Dead letter queue	 
Event Message Persistent	
Retry	 
TimeOut	  
TTL	  
Event Routing Rule	  
Event flow 도출	
Event Message	 
Event Store	   
보상 트랜잭션	   

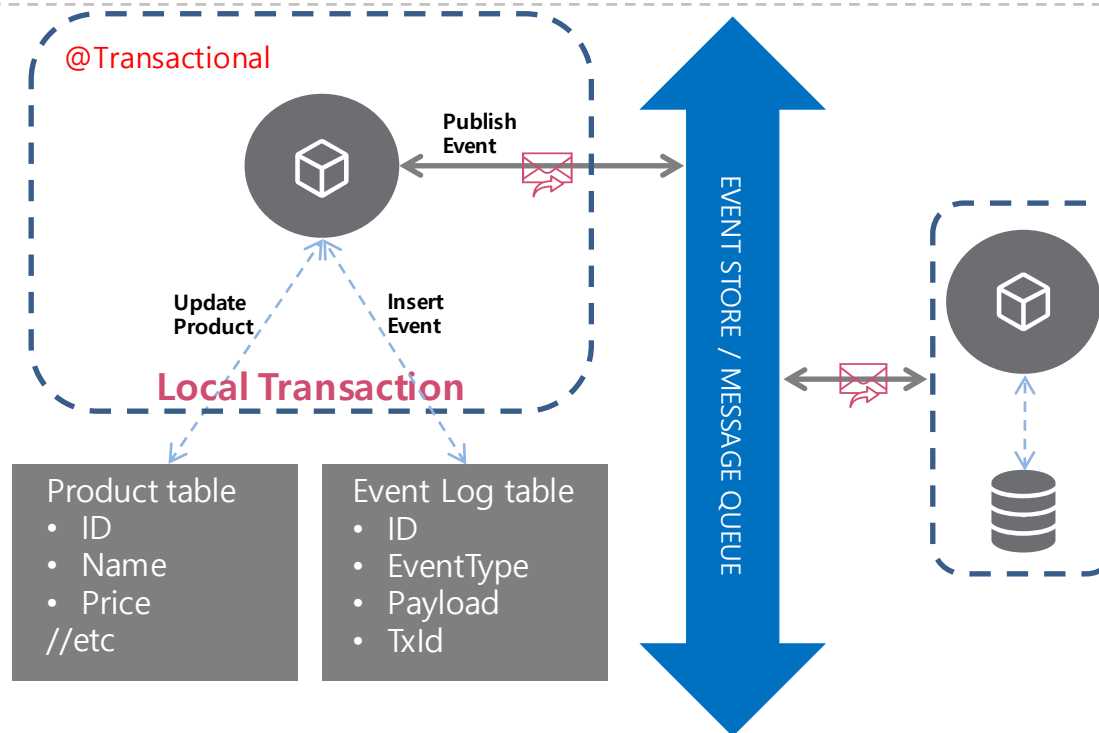
Awesome-shopping Outer 구성

Awesome-shopping의 구성을 확인합니다.



Awesome-shopping Outer 구성

MVC + PubSub /w Local Transaction 에 대해 알아봅니다.



- **MVC + PubSub /w Local Transaction**

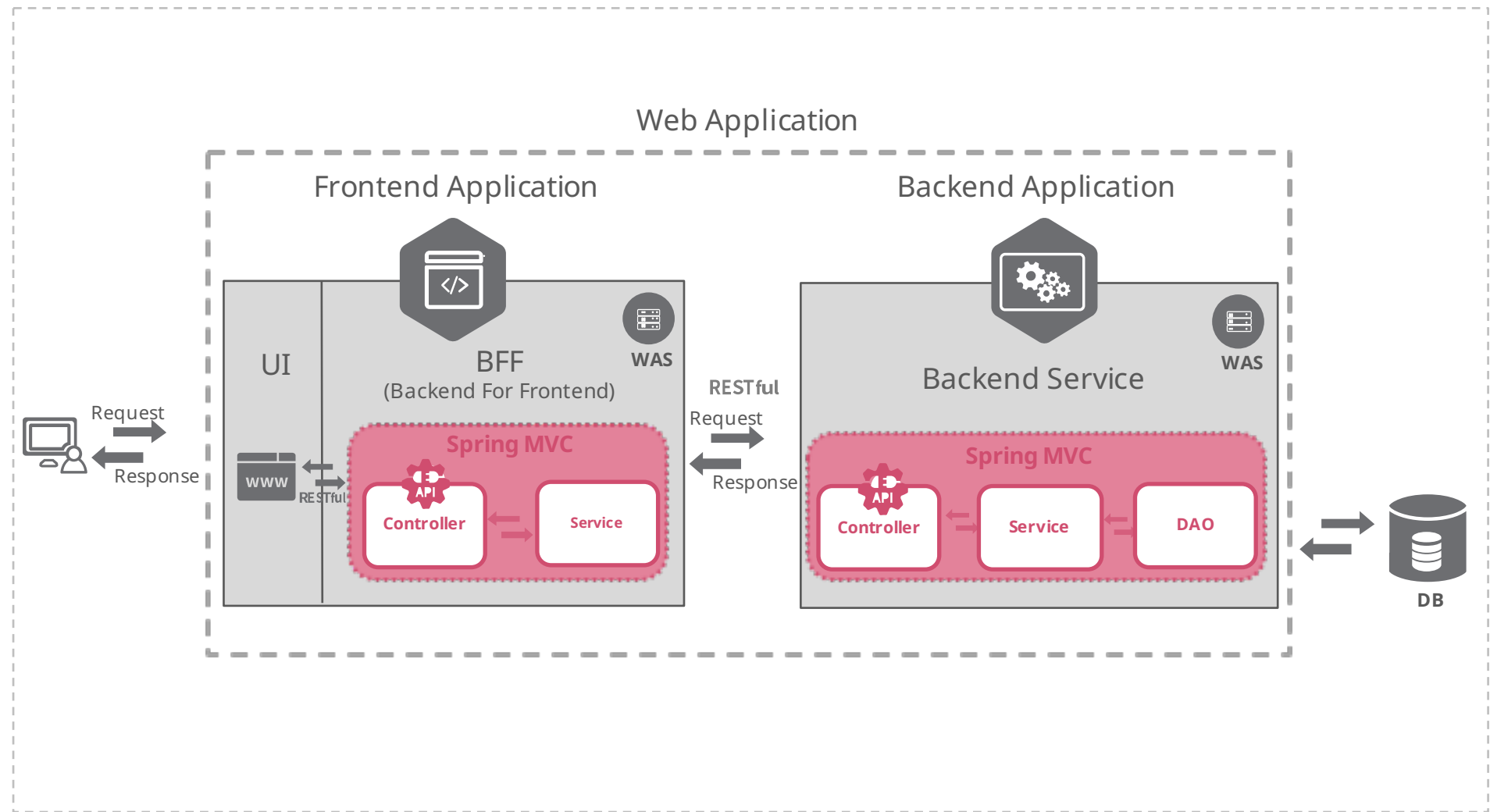
- 로컬 디비 내 이벤트 table 생성
- 이벤트 발생시 로컬 디비에 이벤트 보관
- 보관과 동시에 이벤트를 발행
- 보관과 발행 사이 트랜잭션 처리

- **이유**

- 변경사항이 적다.
- 데이터 변경, 이벤트 보관, 발행 간 Transaction 처리가 쉽다
- 새로운 기술 적용이 적다.
- 이해가 쉽다.

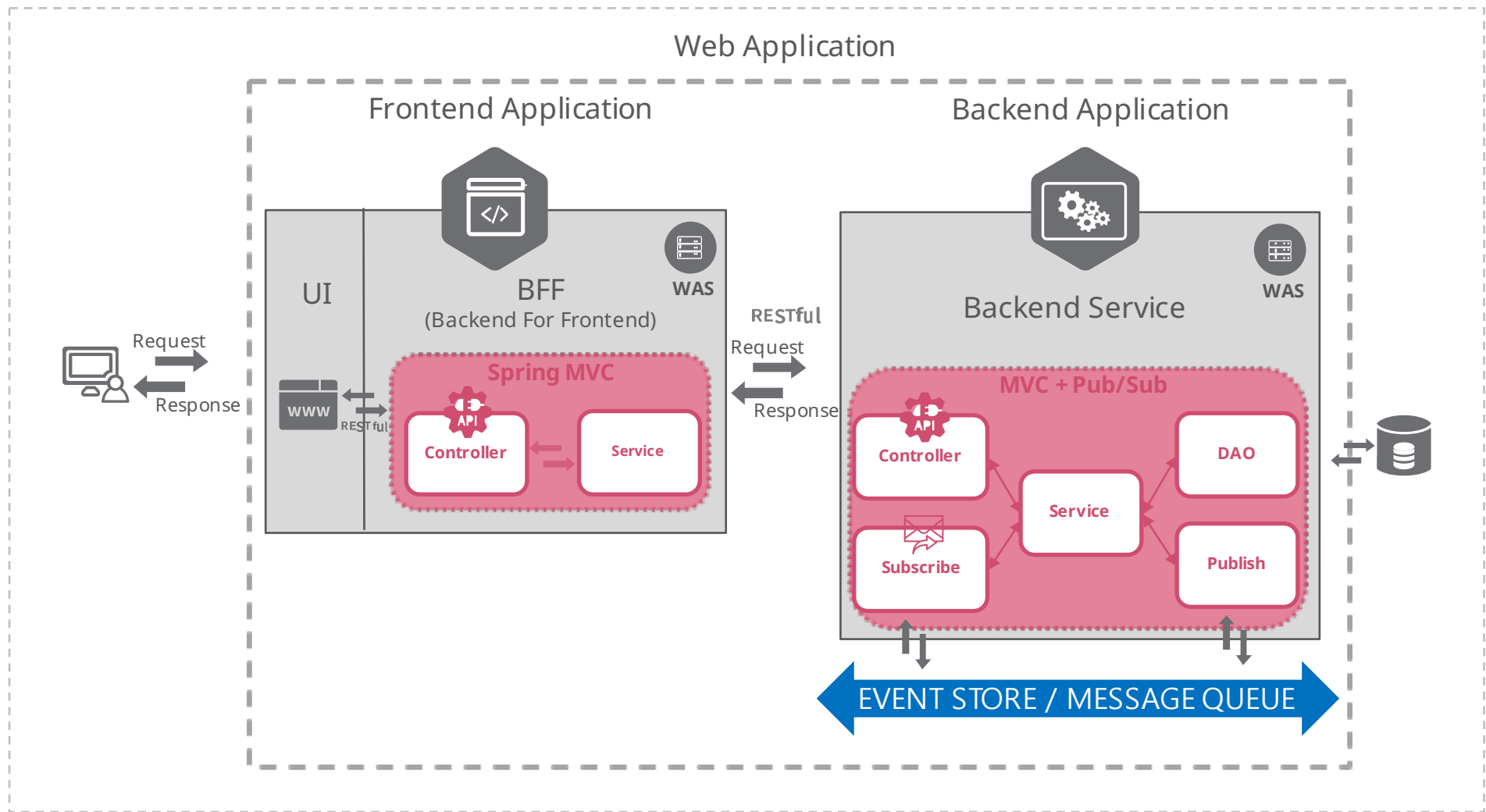
Awesome-shopping Inner 구성

MVC가 적용된 어플리케이션의 Inner 구성을 알아봅시다.



Awesome-shopping Inner 구성

MVC + Pub/Sub이 적용된 어플리케이션의 Inner 구성을 알아봅시다.



Awesome-shopping Inner 구성

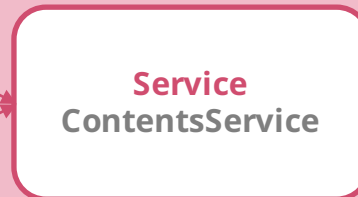
MVC + Pub/Sub이 적용된 어플리케이션의 Layer 상세 내용을 알아봅시다.

- 웹 애플리케이션 요청을 받는 곳
- HTTP 요청과 응답에 대한 처리
- REST API 작성



- 관심있는 이벤트 구독을 하는 곳
- 이벤트 구독 및 메세지 수신
- @StreamListener 작성

- Controller부터 호출됨
- 비즈니스 로직, Transaction 처리



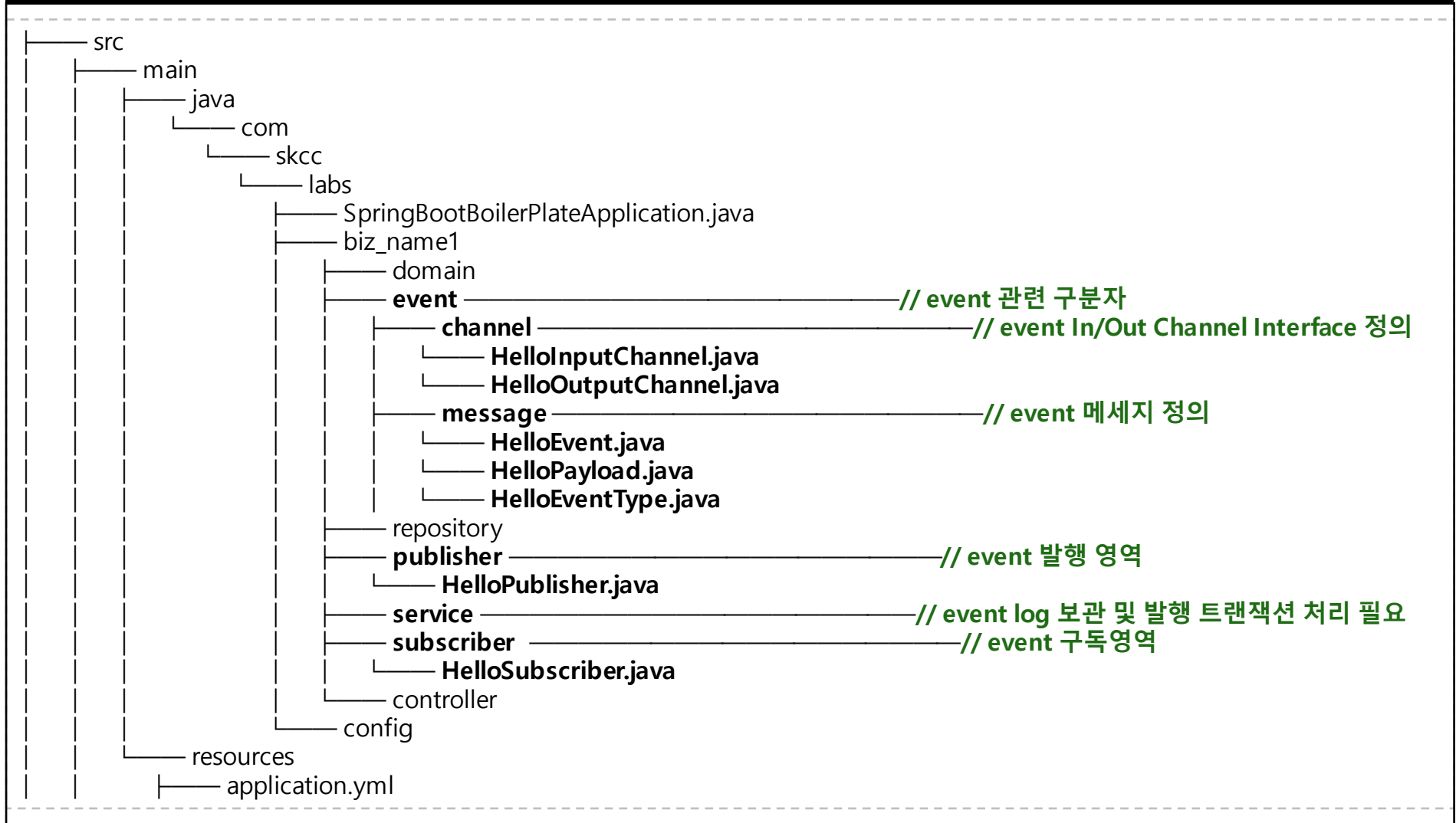
- Data Access Object
- 데이터소스에 접근
- 데이터 CRUD 처리



- 이벤트 메세지 발행

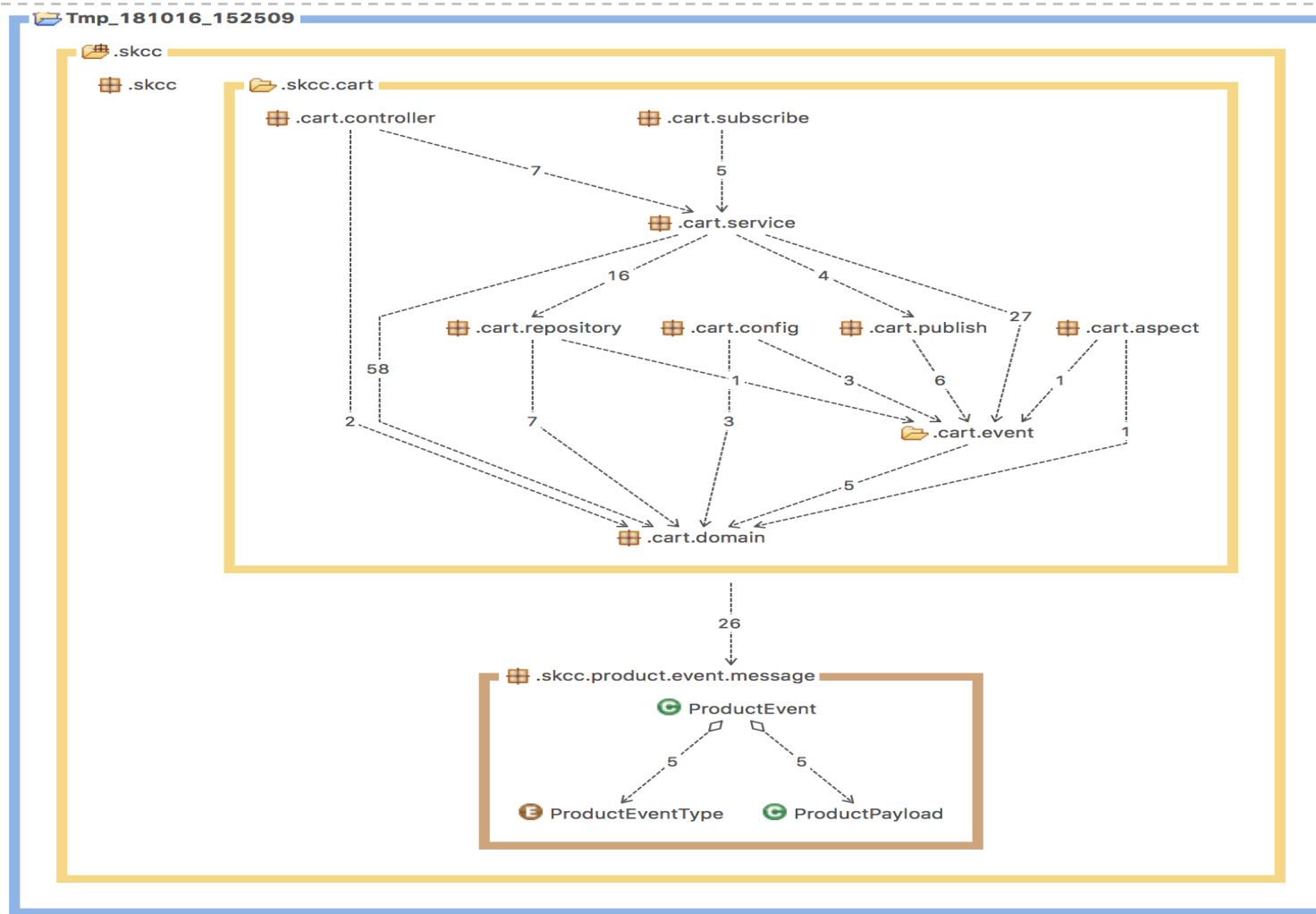
Awesome-shopping MVC + Pub/Sub Package 구성

MVC + Pub/Sub이 적용된 어플리케이션의 Package 구성을 알아봅시다.



Awesome-shopping MVC + Pub/Sub Package Dependency

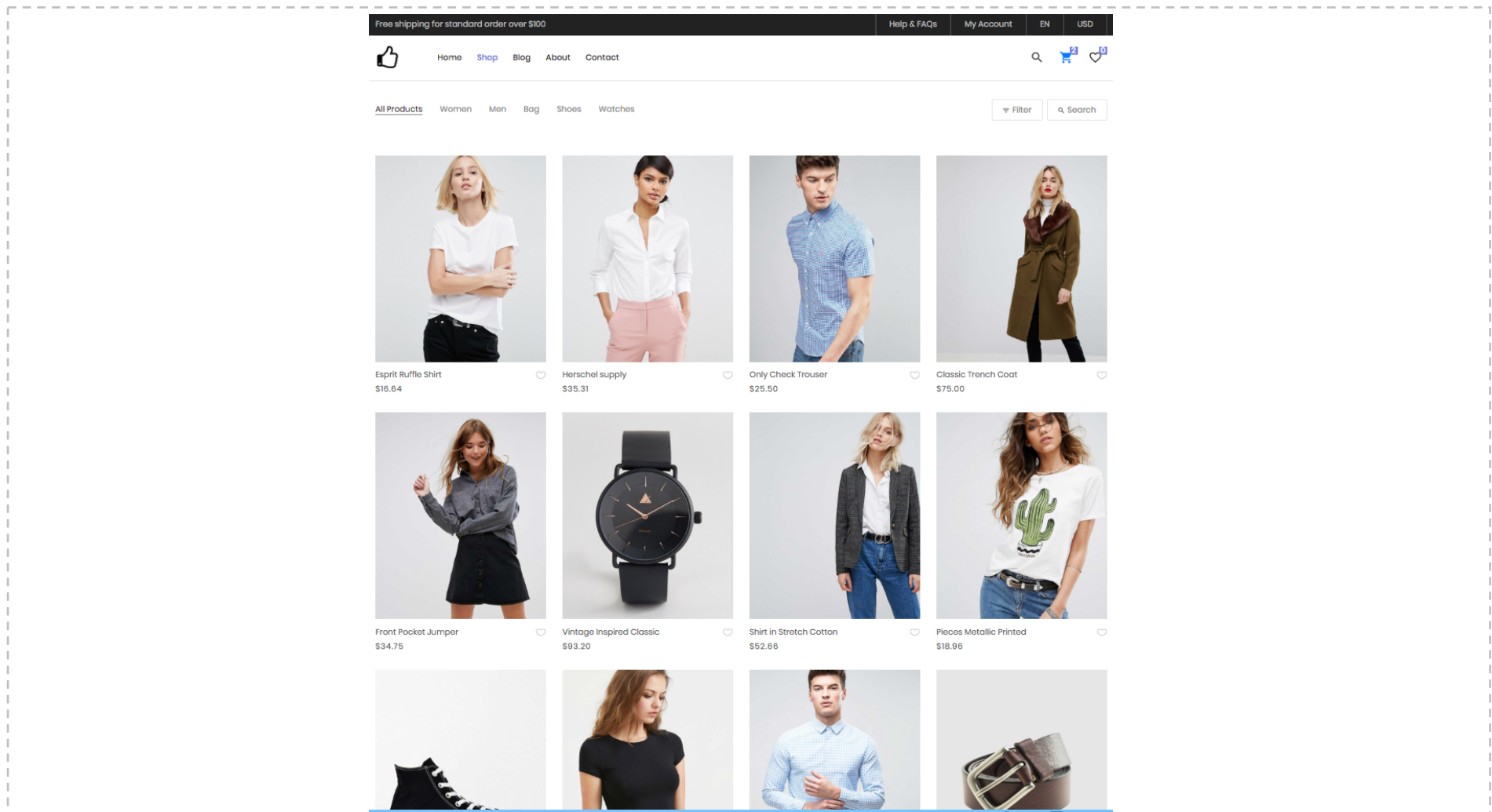
MVC + Pub/Sub이 적용된 어플리케이션의 Package Dependency를 알아봅시다.



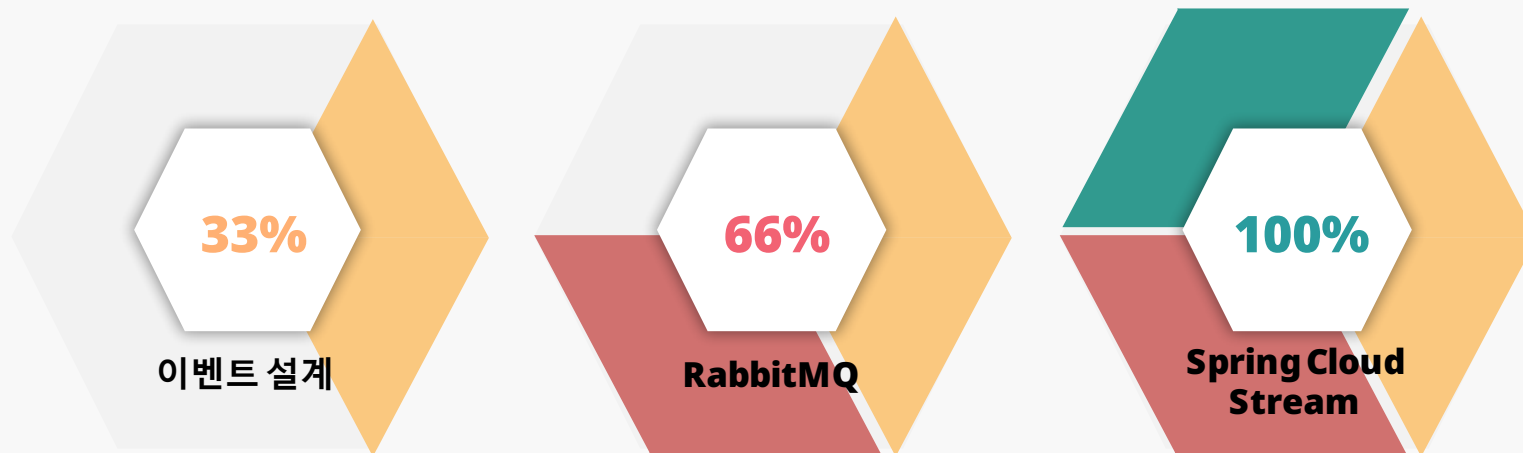
STEP 04. 구성

Awesome-shopping 구현

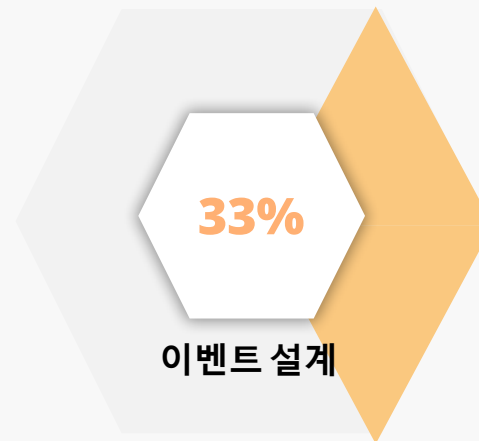
Awesome-shopping을 구현한다면 이런 모습일 것입니다.



PART 02. Awesome-shopping 구현 사전 준비

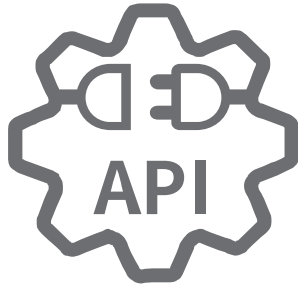


PART 02-1. 이벤트 설계



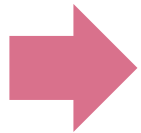
이벤트 설계란?

설계/분석 단계에서 **API**와 마찬가지로 **Event** 도 **list up**, **flow** 정의, 시각화, **Spec** 정의 등의 과정이 필요합니다.



- **REST API vs Event**

- 생성 -> 발행 -> 전달 -> 수신 -> 동작
- 기능 제공 서비스에서 노출
- 요청을 받으면 기능을 수행, 필요시 새로운 요청 전송
- idendity, In/Out Param, 의미 있는 이벤트 정의 등 필요
- **이벤트 설계는 API 설계와 동급**
- **협의를 필요한 영역**



domain 간 이벤트 flow 및 발생,발행,구독 과정을 BPMN 표기법으로 시각화

이벤트 flow

업무 흐름을 기반으로 이벤트를 도출하는 방식이 필요합니다. BPMN 등 업무를 기반으로 이벤트를 도출하는 방법을 활용할 수 있습니다. 이벤트는 Pub/Sub 구조로 전달됩니다. 서비스를 기준으로 이벤트의 발행,구독 흐름을 파악하는 것이 필요합니다.



- **BPMN**(Business Process Model & Notation)
 - UML을 단순화 시킨 새로운 모델링 표준
 - 비즈니스 프로세스 모델링 표준
 - 업무 기능을 기반으로 API, domain, Message, DB access 등 도출 가능
 - BPMN tool에서 모델링 -> 코드 생성까지 가능
 - -> **BPMN 2.0**의 표기법을 통한 이벤트 시각화

[BACKUP]BPMN 2.0 표기법

이벤트 흐름 파악 및 시각화를 위한 BPMN 2.0 표기법을 소개합니다.

Link : <https://www.bpmnquickguide.com/view-bpmn-quick-guide/>

Activity

- 실행될 수 있는 업무 단위, Process, Sub-Process, Task로 구분
- 주로 Sub-Process, Task 사용

Task

- 더 이상 쪼갤 수 없는 최소 업무 단위



Task

Sub-Process

- 내부에 다른 프로세스를 포함하는 프로세스



Sub Process

[BACKUP]BPMN 2.0 표기법

이벤트 흐름 파악 및 시각화를 위한 BPMN 2.0 표기법을 소개합니다.

Link : <https://www.bpmnquickguide.com/view-bpmn-quick-guide/>

EVENT(BPMN 용어)

- 프로세스 실행에 영향을 주는 사건, 행위
- 프로세스 간 comm 및 조건에 따른 프로세스 실행에 사용

Start Event [Start Event](#)

- 프로세스 시작을 의미, 어떤 순서도와도 연결될 수 없다.

Intermediate Event [Intermediate Event](#)

- 프로세스 중간에 일어나는 사건, 행위

End Event [End Event](#)

- 프로세스 종료를 의미

다양한 이벤트 표시



Throw - Message Intermediate Event



Timer Intermediate Event



Interrupting - Error Start Event



Interrupting - Compensation Start Event

[BACKUP]BPMN 2.0 표기법

이벤트 흐름 파악 및 시각화를 위한 BPMN 2.0 표기법을 소개합니다.

Link : <https://www.bpmnquickguide.com/view-bpmn-quick-guide/>

Gateway

- 프로세스 진행 중 조건에 따른 분기 및 통합을 표현하는 것
- 프로세스 의사결정의 순간을 표시

Exclusive Gateway

- 여러가지 결과가 산출될 수 있는 분기점
- 데이터를 기반으로 조건에 따라 분기



Exclusive Gateway - with Marker

Parallel Gateway

- 나뉘어져서 실행 후 merge -> sync 동작
- 분기점에서 나뉜 프로세스는 동시에 시작



Parallel Gateway

[BACKUP]BPMN 2.0 표기법

이벤트 흐름 파악 및 시각화를 위한 BPMN 2.0 표기법을 소개합니다.

Link : <https://www.bpmnquickguide.com/view-bpmn-quick-guide/>

Swimlane

- Pool 내 역할, 임무에 따라 세분화된 그룹
- 그룹 간 프로세스의 상호작용이 도표화되는 영역
- MSA에서 domain, 서비스를 표현하기 적합



Lane

Pool

- Swimlane을 묶어서 표현한 개념
- Pool간 구별은 해당 프로세스에 참여하는 구별할 수 있는 조직을 의미
- MSA에서 시스템 구분을 표현하기에 적합



Pool


[BACKUP]BPMN 2.0 표기법

이벤트 흐름 파악 및 시각화를 위한 BPMN 2.0 표기법을 소개합니다.

Link : <https://www.bpmnquickguide.com/view-bpmn-quick-guide/>

Flow Object

- 프로세스 안 요소 간의 관계를 묘사

Sequence Flow  Sequence Flow

- 시간의 흐름에 따른 프로세스의 진행
- 시작되는 요소와 받는 요소가 반드시 필요
- 동일한 Swimlane 내에서 사용

Message Flow  Message Flow

- 서로 다른 참여자 사이의 메세지 흐름 표시
- 동일한 Swimlane 내에서 사용 불가

```

graph LR
    Start(( )) --> Reserve[reserve goods]
    Reserve --> Process[process payment]
    Process --> Prepare[prepare shipment]
    Prepare --> Shipment((shipment prepared))
    Shipment --> Next[...next steps...]
    Next --> End((order processed))

    Reserve --> Remove[remove reservation]
    Remove -.-> Process

    Process -- error occurred --> Reverse[reverse payment]
    Reverse --> Process

    Process --> Inform[inform customer]
    Inform --> Diamond{ }
    Diamond --> TwoDays((2 days))
    TwoDays --> Cancelled1((order cancelled))

    Diamond --> Updated[payment data updated]
    Updated --> Cancelled2((order cancelled))

    Inform --> Cancelled3((order cancelled))

    Start --> Requested1((cancellation requested))
    Requested1 --> Cancelled4((order cancelled))

    Start --> Requested2((cancellation requested))
    Requested2 --> Refusal[send refusal information]
    Refusal --> Refused((cancellation refused))
  
```


이벤트 list up

발행 서비스는 이벤트 구독 서비스를 고려하지 않고 이벤트를 발행합니다. 구독 서비스는 시스템 내 이벤트의 **list** 를 파악해서 관심있는 이벤트를 구독할 수 있습니다.

(Example)

- Publish

서비스명	이벤트명	의미
account	accountCreated	계정 생성
account	accountEdited	계정 정보 변경
account	accountCreateFailed	계정 생성 실패
account	accountEditFailed	계정 정보 변경 실패

- Subscribe

서비스명	이벤트명	사유	To do	의미
order	productSubtracteFailed	rollback	주문 취소	상품 확보 실패
order	paymentCreateFailed	rollback	주문 취소	결제 내역 생성 실패
order	paymentCreated	biz. flow		결제 생성, 결제 데이터 동기화 필요
order	paymentPaid	biz. flow		결제 완료, 결제 데이터 동기화 필요

이벤트 spec

이벤트 객체는 비동기 메시지 큐 방식으로 시스템 내 통합을 수행합니다. 시스템 내 공통으로 인식될 수 있는 이벤트 객체의 spec을 정의할 필요가 있습니다.

- Publish

Spec 명	의미
서비스명	이벤트 발행 주체 서비스
이벤트명	의미 있는 이벤트 식별자

- Subscribe

Spec 명	의미
서비스명	이벤트 발행 주체 서비스
이벤트명	의미 있는 이벤트 식별자
사유	구독 사유
To do	이벤트 구독 후 수행 업무



이벤트 시각화를 바탕으로 기능별 이벤트 발행/구독 구현을 위한 list up 및 Spec 정의

이벤트 message

의미있는 단위의 이벤트를 전달하기 위한 이벤트 메세지 구성이 필요합니다. 이벤트 메세지는 협의를 통해 발행,구독 서비스 모두에서 사용가능한 형식으로 정의합니다.

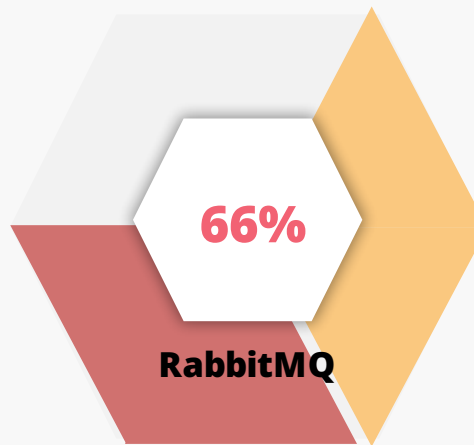
(Example)

id	domain	domainId	eventType	Payload	txId
1	account	10002156	CartAdded	{"username": " abcd@google.com ", "name": "jisang", "mobile": "01012345678", "scope": "customer"}	tx123123

- 이벤트 메세지

column	의미
id	이벤트 메세지를 식별하는 key 값
domain	이벤트 발행 도메인
domainId	이벤트 발행에 연관된 도메인 객체의 id
eventType	이벤트 유형. 의미 있는 이름 작성 필요
payLoad	데이터 생성, 변경, 삭제 데이터. Before/After 혹은 변경사항을 전달
txId	기능 수행에 연관된 이력을 조회하기 위한 key 값

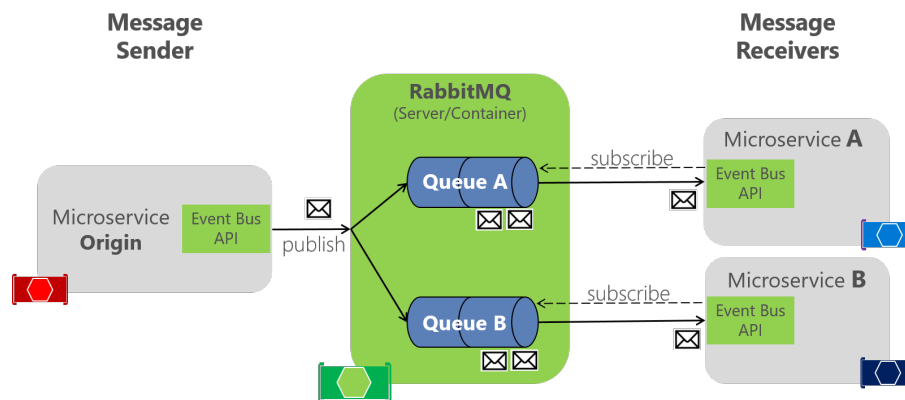
PART 02-2. RabbitMQ



RabbitMQ 란?

비동기 메시지 통신(RabbitMQ)을 통한 시스템 내 통합을 수행합니다.

<https://www.rabbitmq.com/getstarted.html>

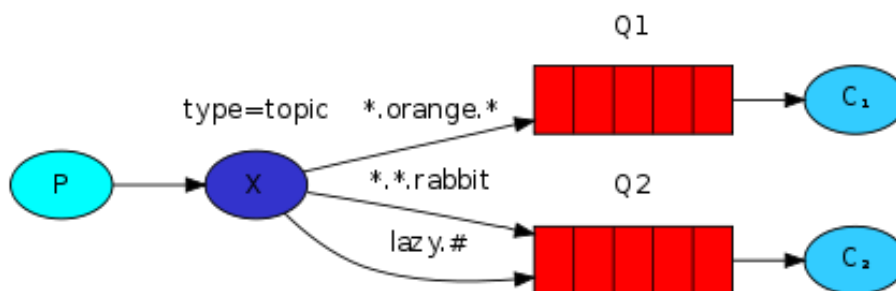


RabbitMQ 란?

- AMQP 프로토콜을 구현한 메시지 큐

주요 용어

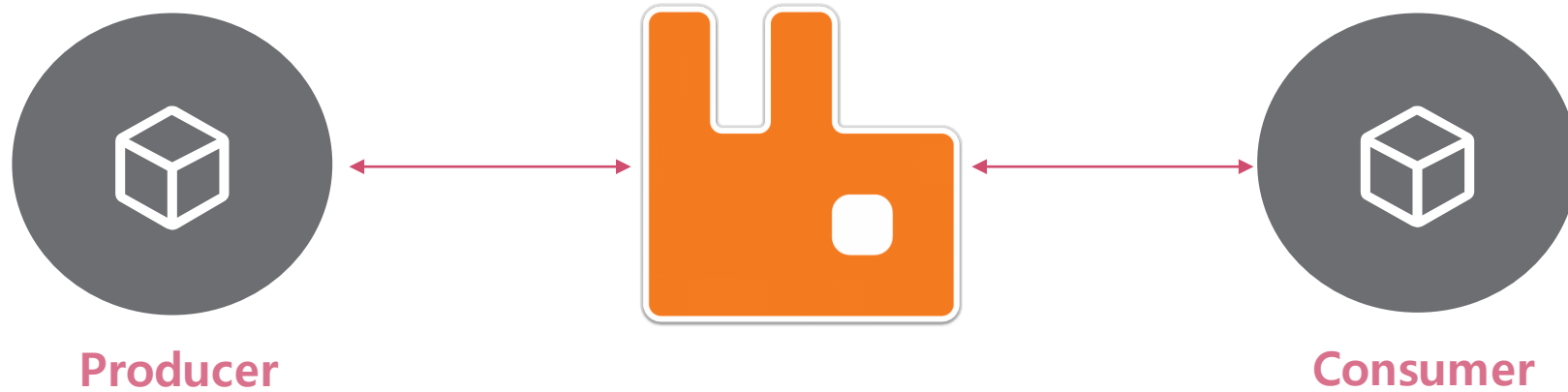
- Producer
- Consumer
- Exchange
- Binding
- Queue
- Dead letter queue / TTL



➡ 샘플은 Topic Exchange 방식을 적용

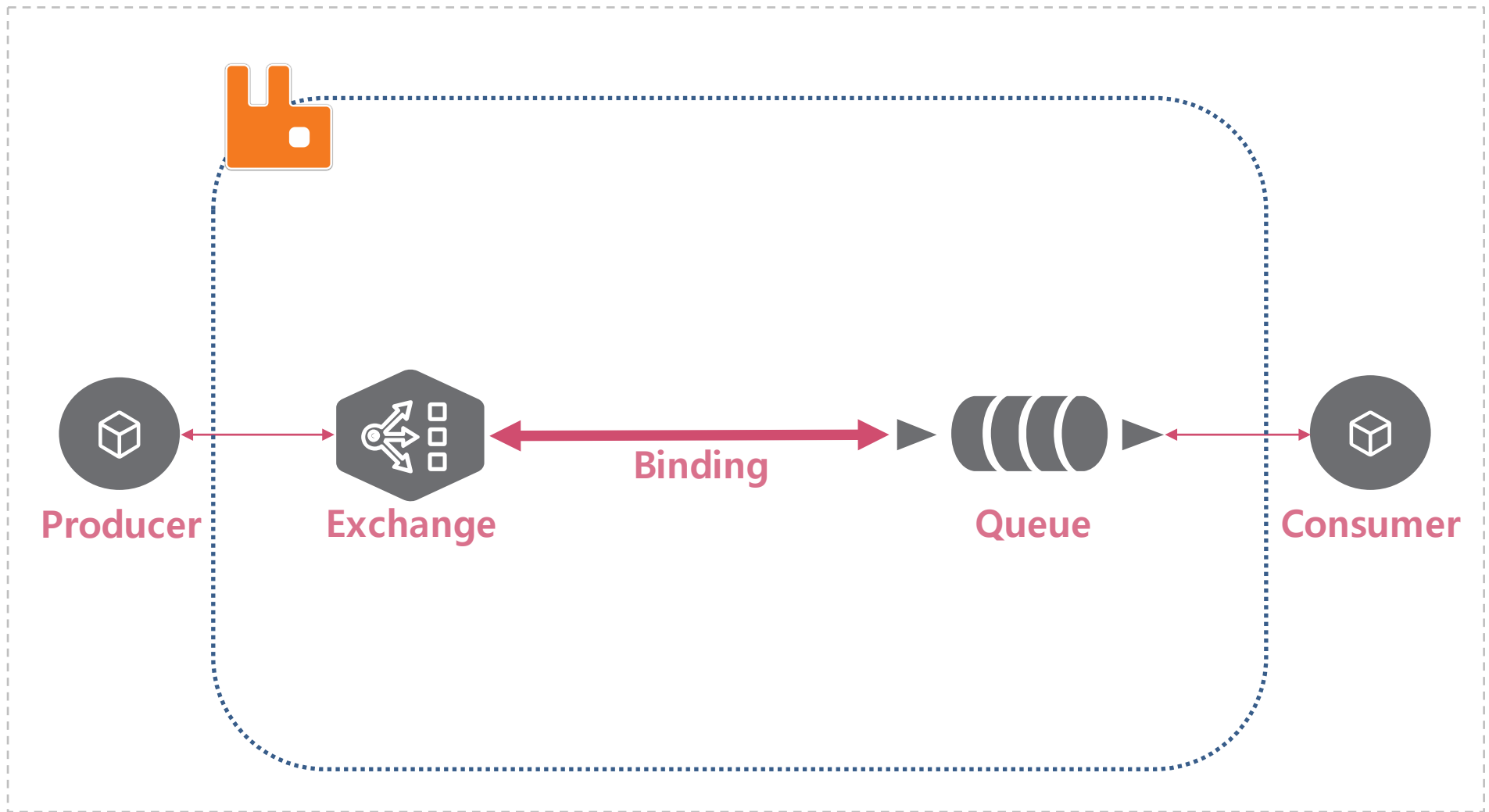
RabbitMQ란?

RabbitMQ의 Producer, Consumer에 대해 알아봅니다.



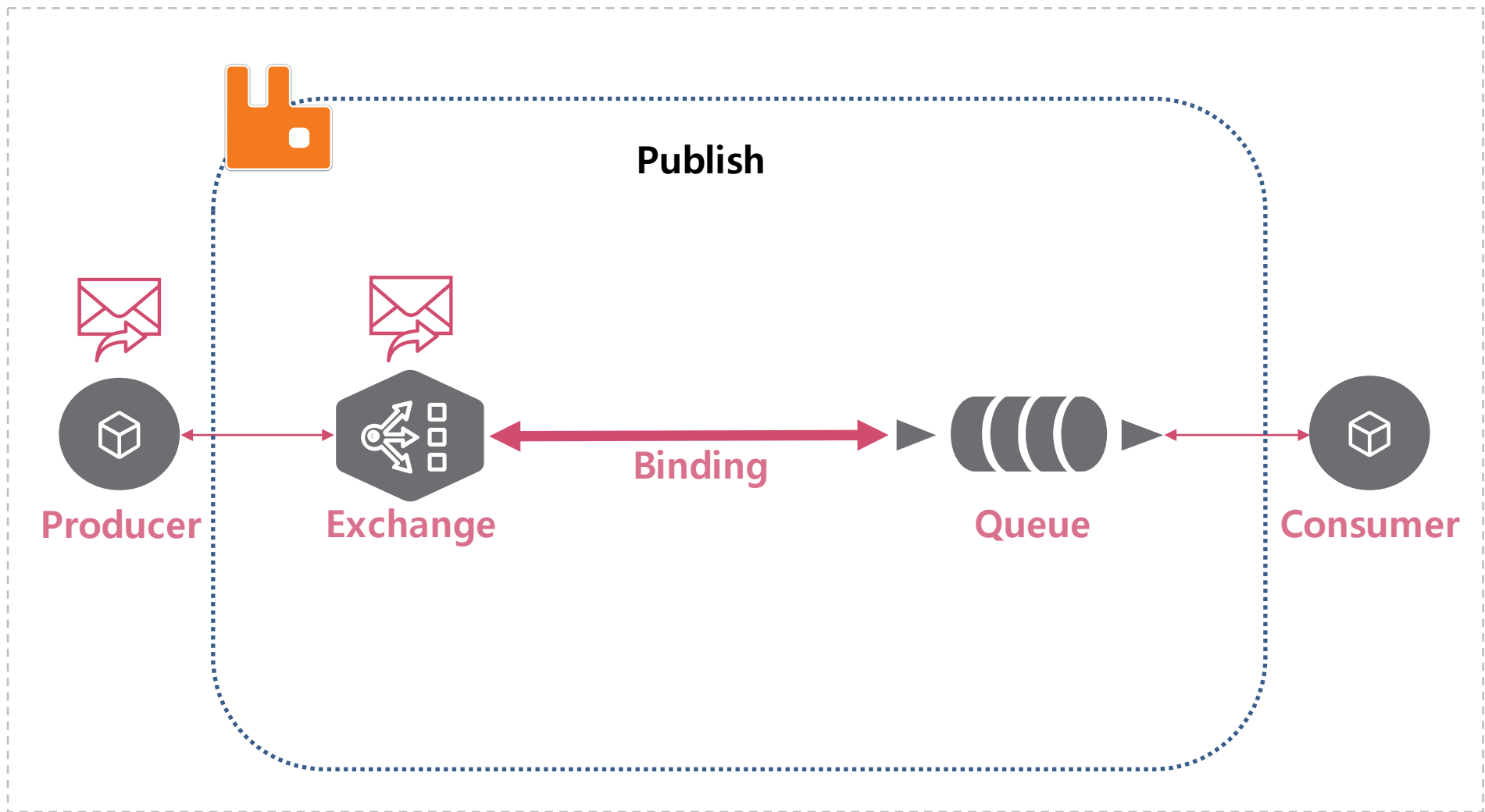
RabbitMQ 란?

RabbitMQ의 Queue, Exchange, Binding에 대해 알아봅니다.



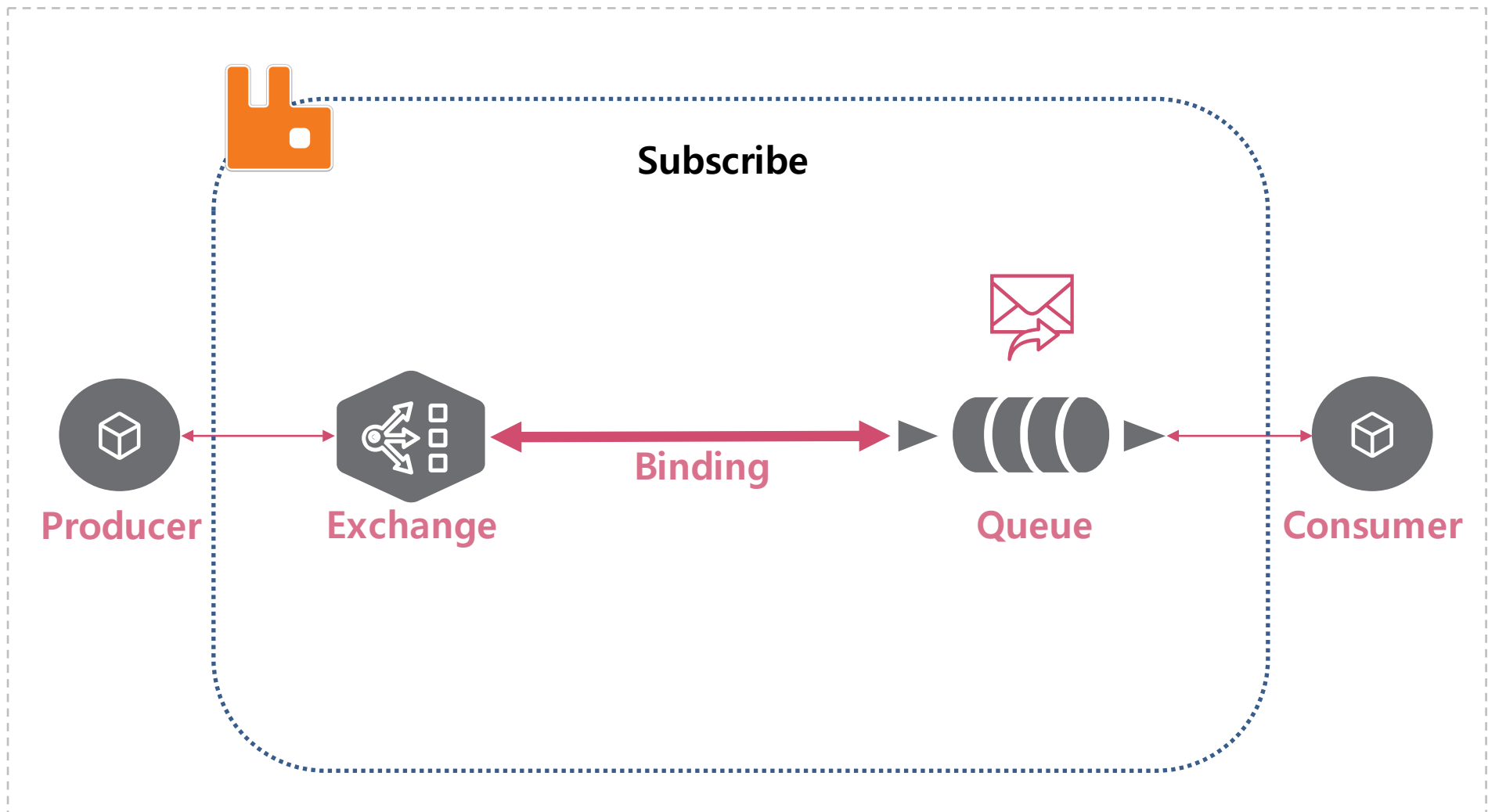
RabbitMQ 란?

RabbitMQ에서 **one-to-one** 메세지 발행에 대해 알아봅니다.



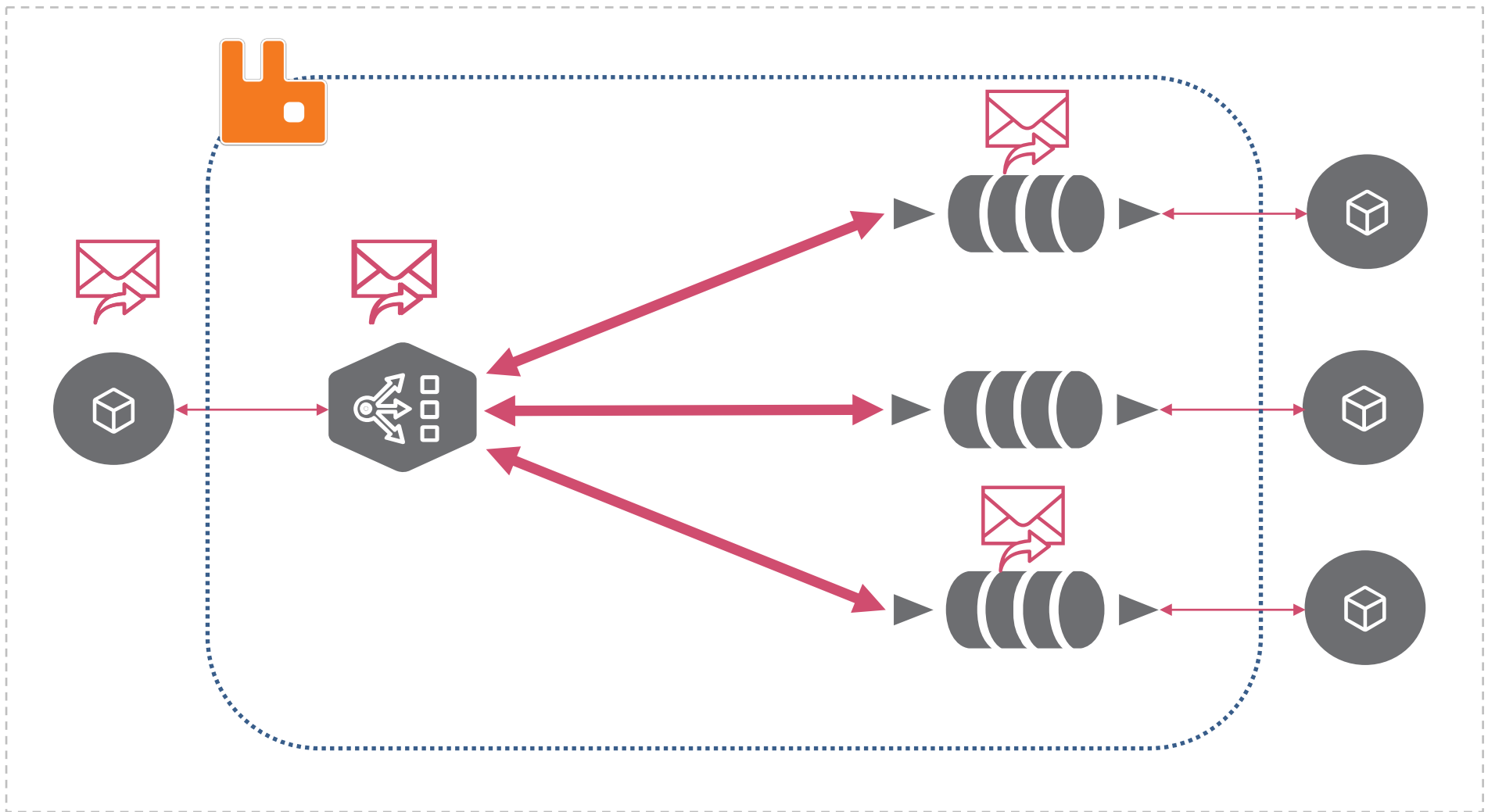
RabbitMQ 란?

RabbitMQ에서 **one-to-one** 메세지 구독에 대해 알아봅니다.



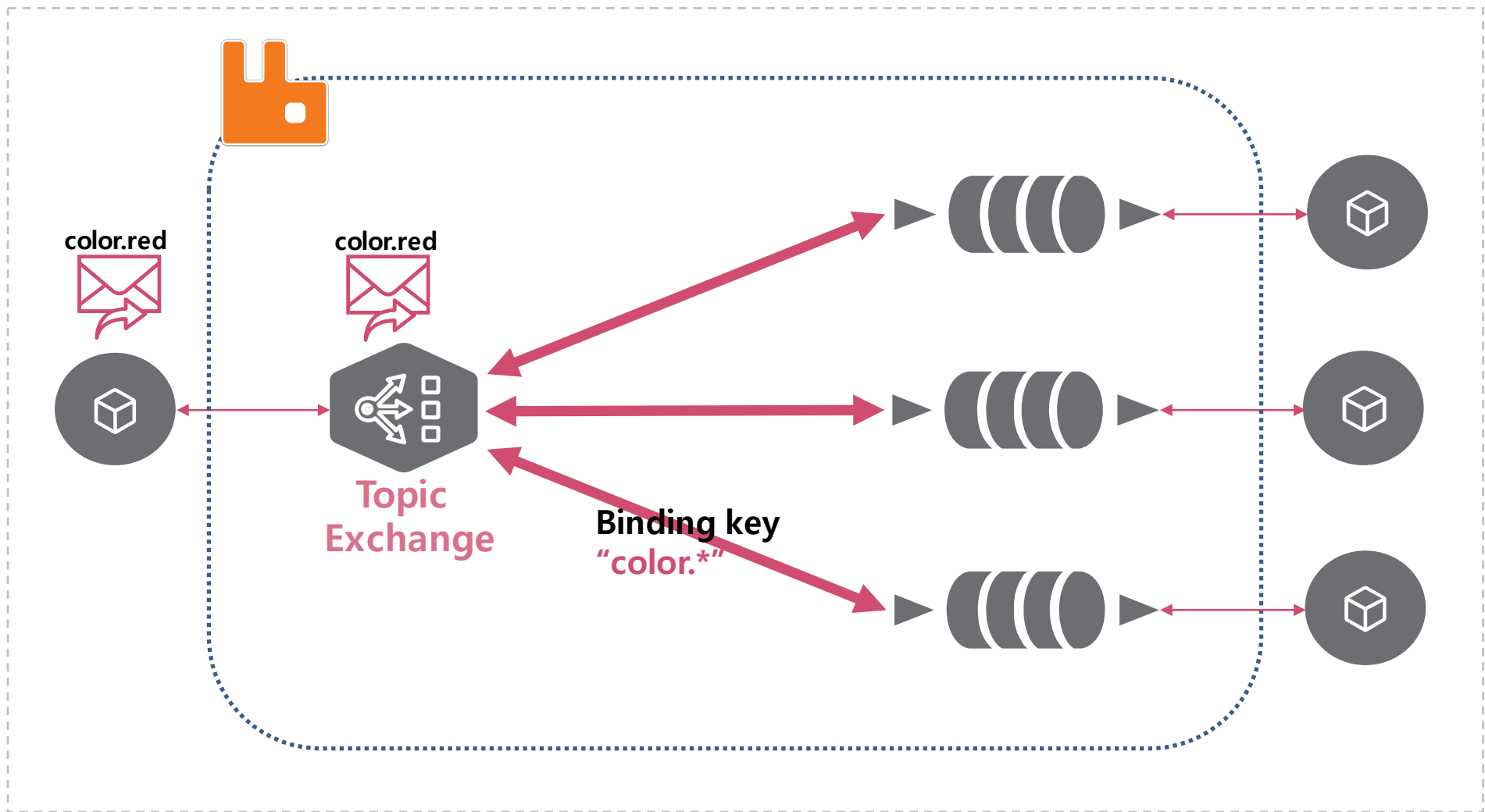
RabbitMQ란?

RabbitMQ에서 one-to-many 메세지 전달에 대해 알아봅니다.



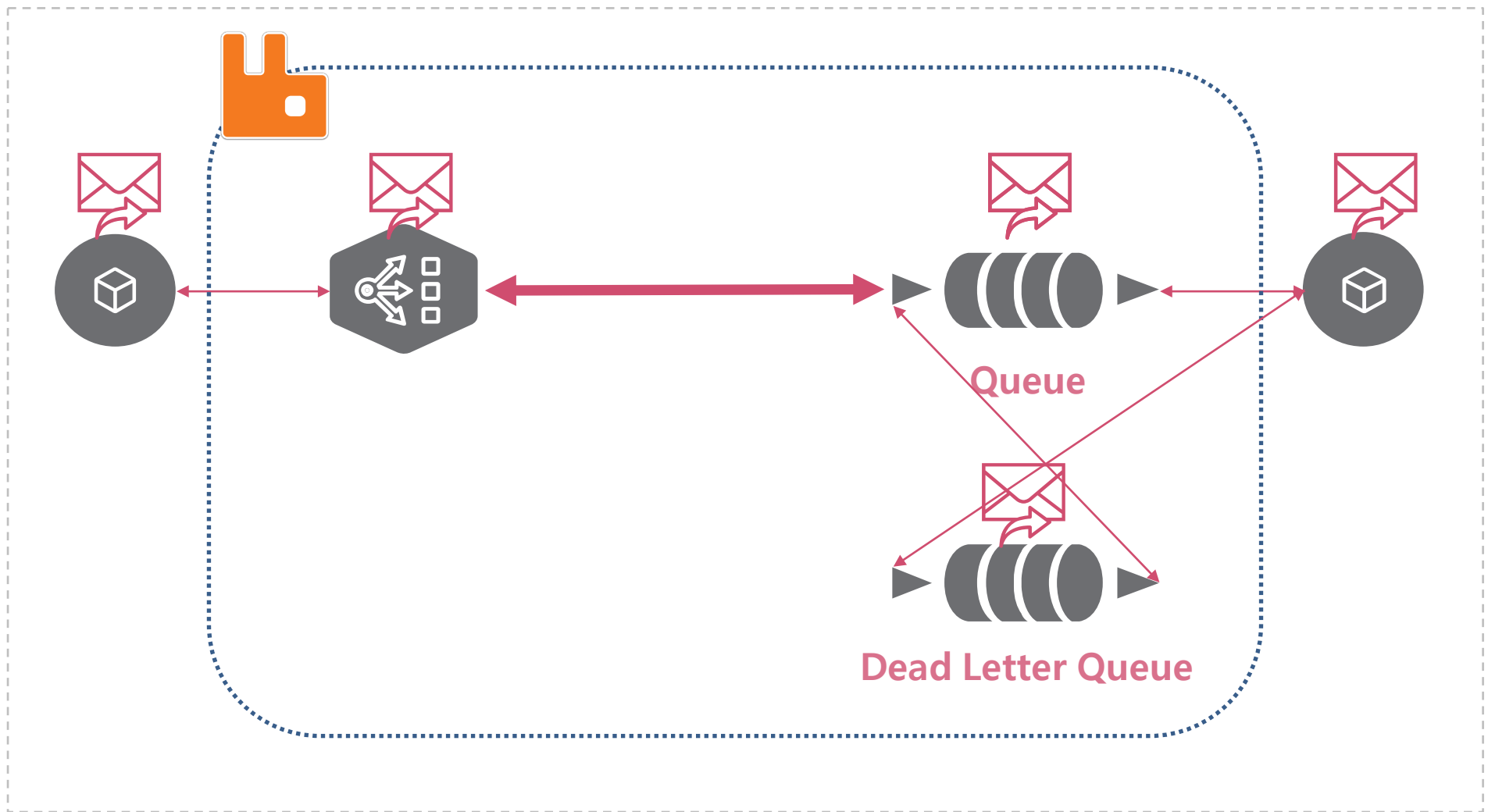
RabbitMQ란?

RabbitMQ의 Topic Exchange의 동작에 대해 알아봅니다.

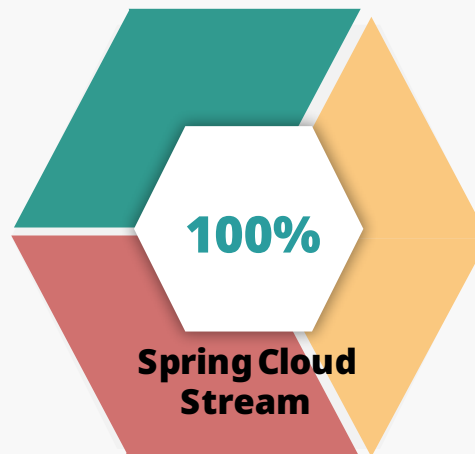


RabbitMQ란?

RabbitMQ의 Dead Letter Queue에 대해 알아봅시다.



PART 02-3. Spring Cloud Stream

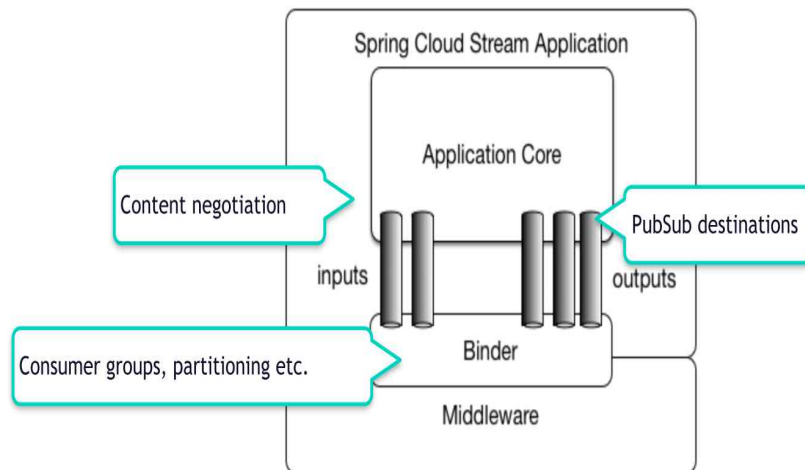


Spring Cloud Stream

이벤트/메세지 기반 Architecture를 구현하는 Framework인 Spring Cloud Stream 을 알아봅니다.

<https://cloud.spring.io/spring-cloud-stream/>

Spring Cloud Stream



Spring Cloud Stream

- 메세지 기반 Application 구현을 위한 F/W
- Application 레벨에서 메세지 큐와 연계를 위한 추상화 제공
- Spring Integration + Spring Boot
- 현재 RabbitMQ / Kafka 지원

주요 용어

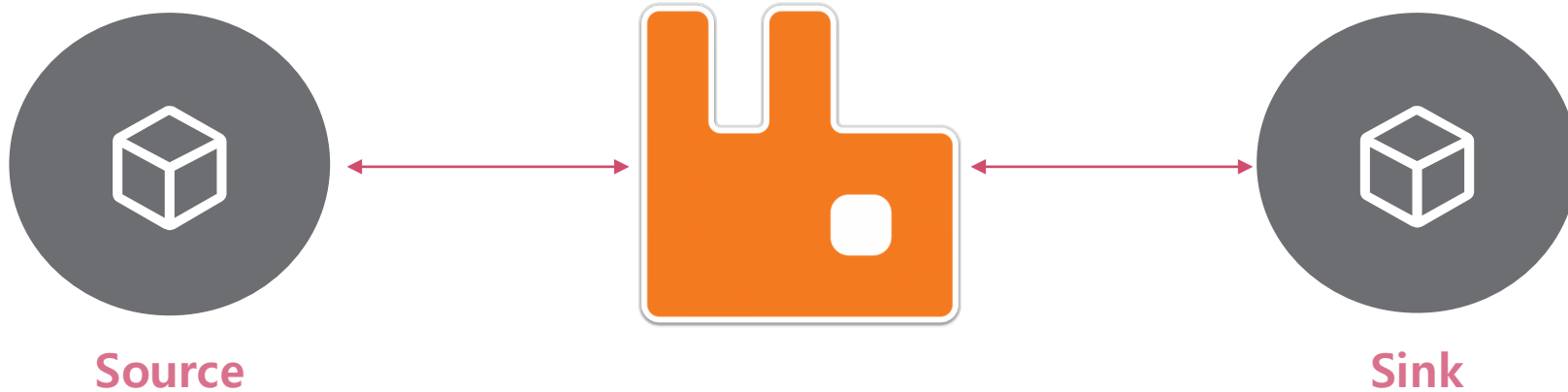
- Source
- Sink
- Binder
- Binding

why?

- serialize/deserialize 구현에 신경 쓸 필요없다.
- 추상화 → 메세지 브로커의 내부 구성 이해 필요 X
- 설정 + 어노테이션 기반
- 파티셔닝 지원
- Consumer 그룹 기능 지원

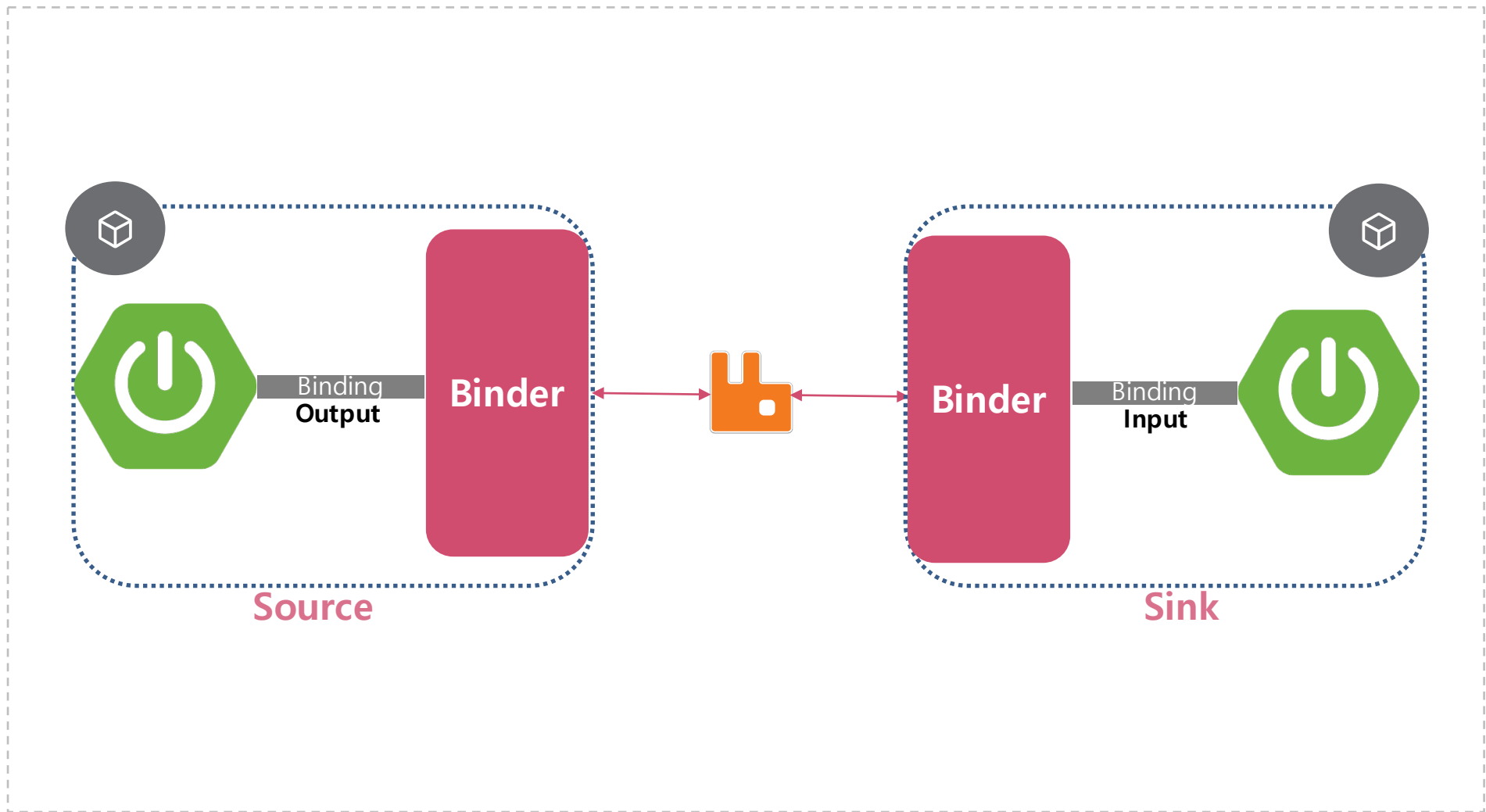
Spring Cloud Stream

Spring Cloud Stream 의 Source, Sink에 대해 알아봅시다.



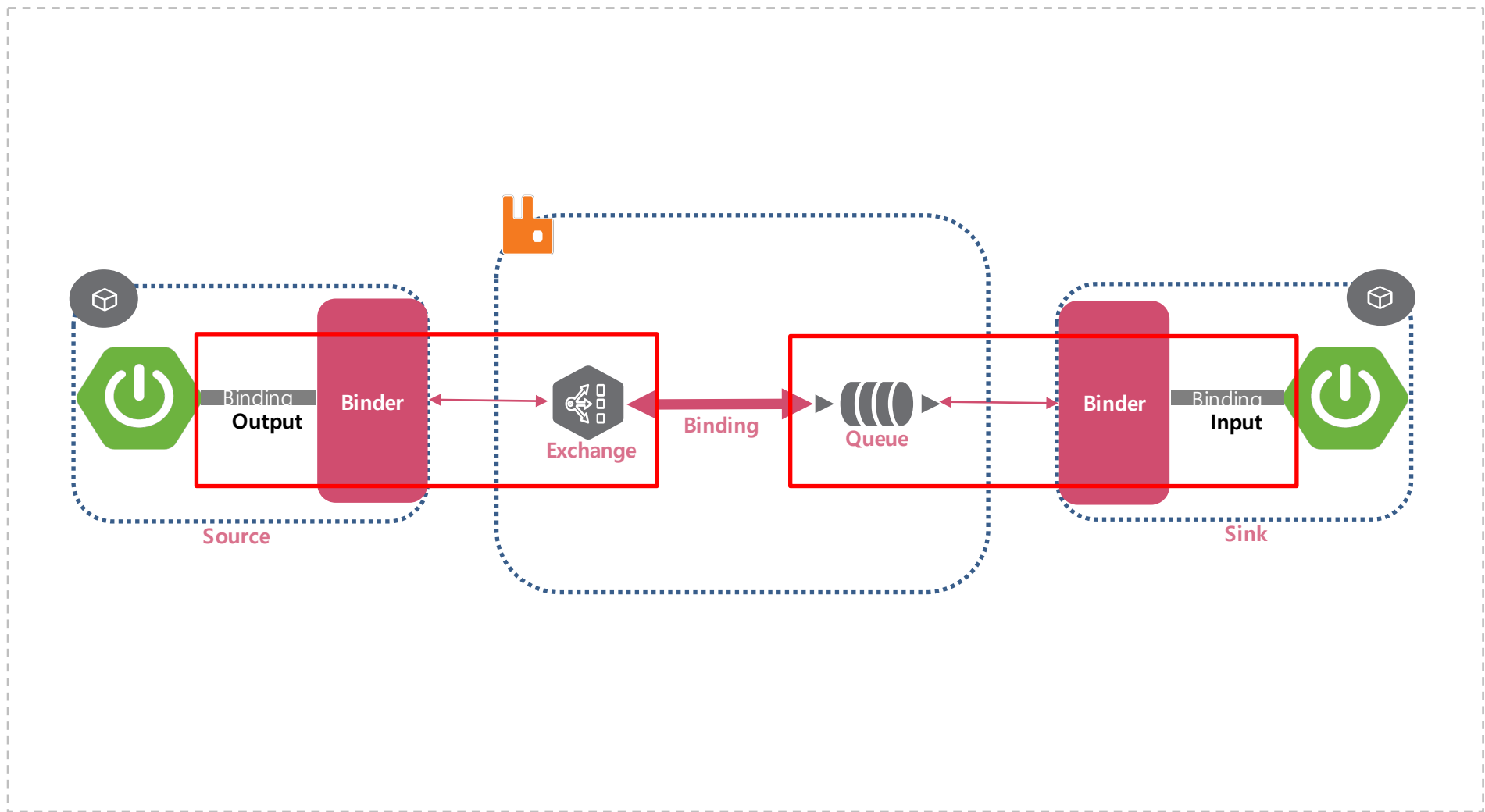
Spring Cloud Stream

Spring Cloud Stream의 Binder와 Binding에 대해 알아봅시다.



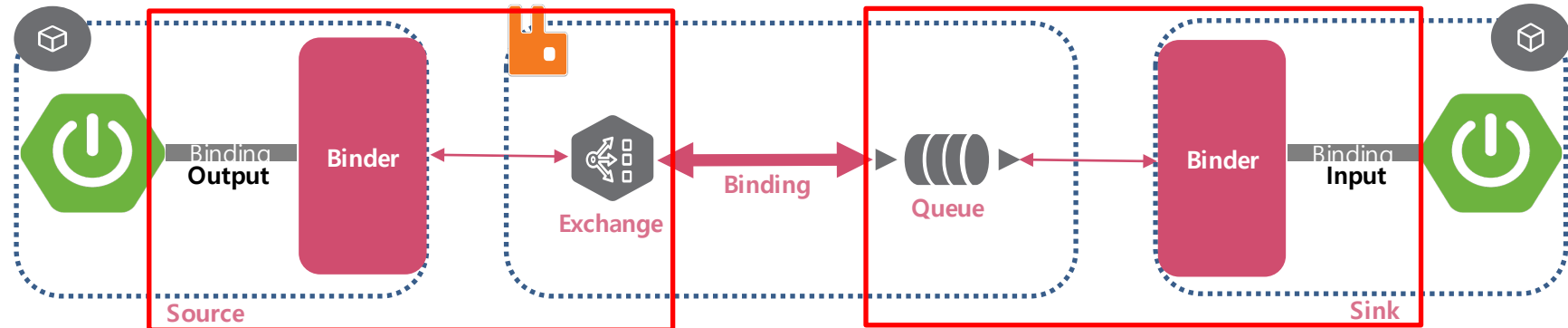
Spring Cloud Stream

Spring Cloud Stream과 RabbitMQ의 컴포넌트를 도식화합니다.



Spring Cloud Stream

Spring Cloud Stream의 설정과 RabbitMQ의 컴포넌트가 연계되는 것을 확인합니다.



```
spring:
  cloud:
    stream:
      binders:
        rabbitmq:
          type: rabbit
          environment:
            spring:
              rabbitmq:
                host: 169.56.106.154
                port: 32598
                username: labs
                password: awesome-rabbitmq
```

```
bindings:
  Input:
    destination: Exchange #Queue이름 관련
    group: cart.productAmountAdded #Queue이름 관련
    content-type: application/json
  Output:
    destination: Exchange
    content-type: application/json
  rabbit:
    bindings:
      Input:
        consumer:
          autoBindDlq: true
          bindingRoutingKey: cart.productAmountAdded.#
      Output:
        producer:
          routing-key-expression: headers['routeTo']
```

PART 03. EDM 구현 따라하기



STEP 01. EDM 설계 따라하기

awesome-payment-service의 기능을 구현해 정상적으로 주문 처리를 진행해보겠습니다.

이벤트/메세지 기반 Architecture를 구현하는 Framework인 Spring Cloud Stream 을 알아봅니다.

The screenshot shows a web application interface. At the top, there is a navigation bar with a thumbs-up icon, links for Home, Shop, Blog, About, and Contact, and a search bar. A shopping cart icon with a red box around it and a heart icon are also present. Below the navigation bar is a section titled "PRODUCT OVERVIEW". Under this section, there are tabs for "All Products", "Women", "Men", "Bag", "Shoes", and "Watches". To the right of these tabs are "Filter" and "Search" buttons. Below the tabs, there are four product cards, each with an image, a title, and a price:

- Esprit Ruffie Shirt 76000
- Herschel supply 90000
- Only Check Trouser 76000
- Classic Trench Coat 160000

Below the product cards, there is a dashed red line. Below the dashed red line, there is a browser address bar showing "localhost:8081/shopping-cart". Below the address bar, there is a section titled "Whitelabel Error Page". To the left of this section is a large red arrow pointing right. The text in the "Whitelabel Error Page" section reads:

This application has no explicit mapping for /error, so you are seeing this as a fallback.

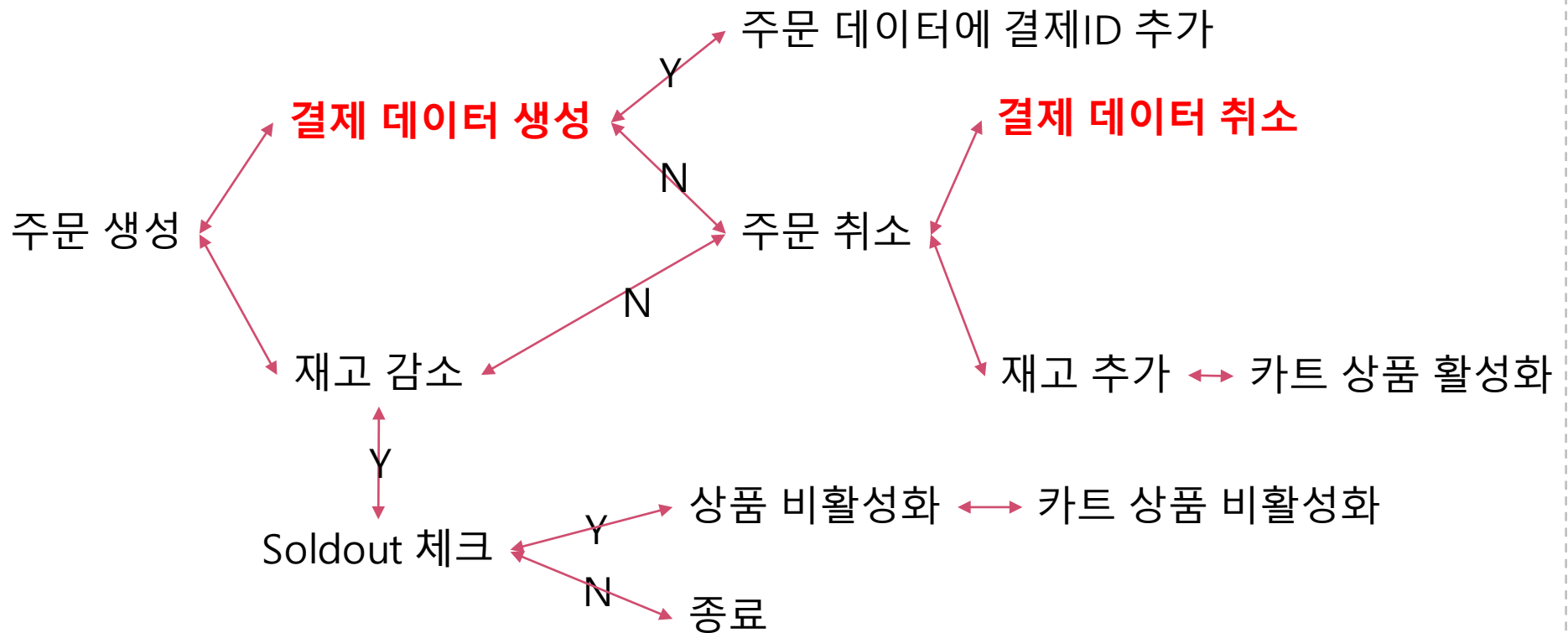
Mon Oct 22 18:43:27 KST 2018
There was an unexpected error (type=Internal Server Error, status=500).
500 null

Awesome-Shopping 이벤트 설계 따라하기

Awesome-shopping의 기능 중 주문하기를 BPMN 2.0 표기법에 따라 구성해보겠습니다.

주문하기 기능

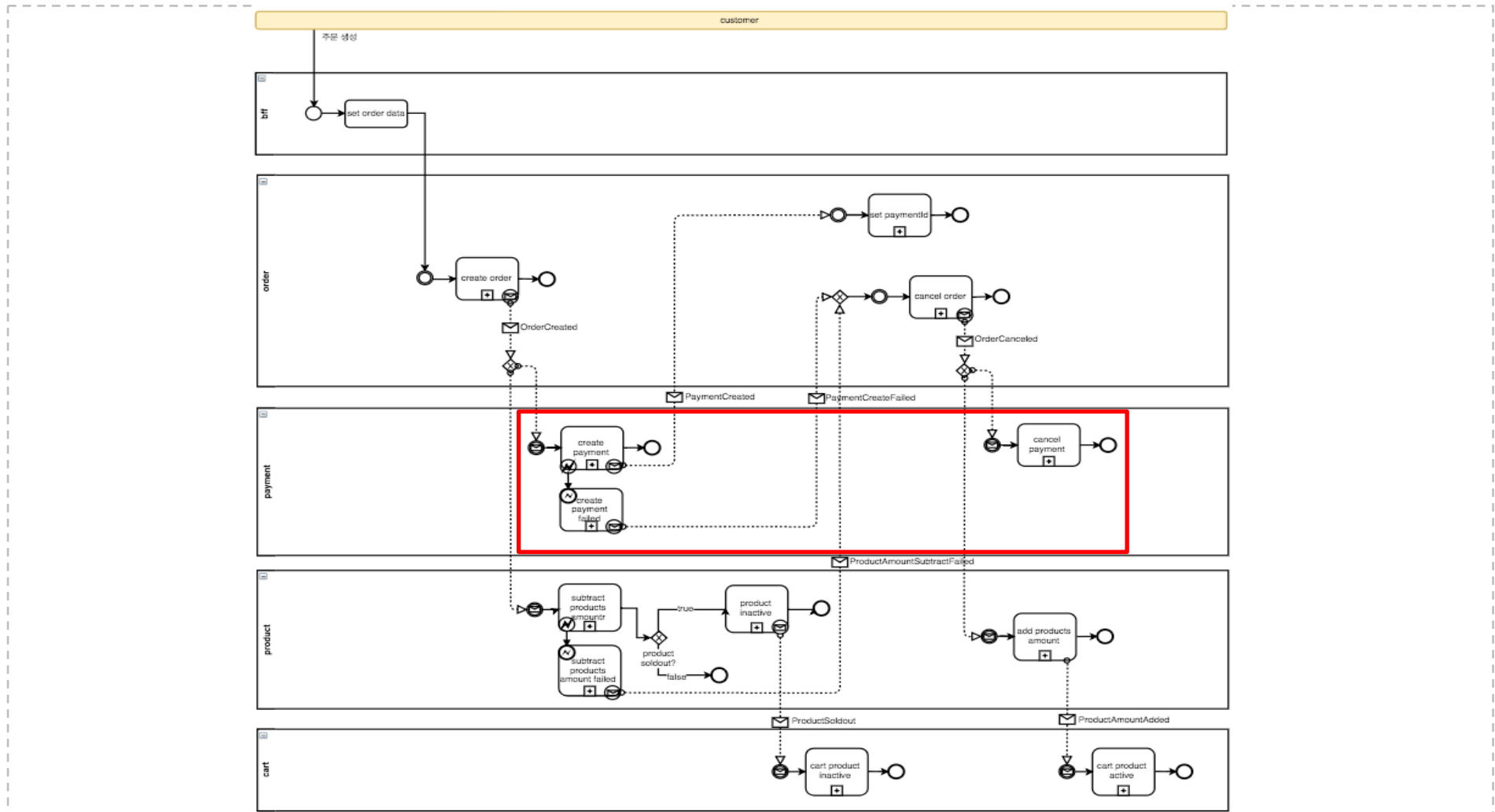
awesome-payment-service 기능



STEP01. EDM 설계 따라하기

Awesome-Shopping 이벤트 설계 따라하기

Awesome-shoping의 기능 중 주문하기 중 awesome-payment-service의 이벤트 설계 과정을 따라해보겠습니다.



Awesome-Shopping 이벤트 설계 따라하기

BPMN 2.0 표기법에 따라 구성된 awesome-payment-service의 이벤트를 list up 합니다.

awesome-payment-service list up

- Publish

서비스명	이벤트명	의미
payment	PaymentCreated	결제 생성
payment	PaymentCreateFailed	결제 생성 실패
payment	PaymentCanceled	결제 취소
payment	PaymentCancelFailed	결제 취소 실패

- Subscribe

서비스명	이벤트명	사유	의미	To do
payment	OrderCreated	biz. flow	주문 생성시 확보한 결제 데이터 생성	결제 생성
payment	OrderCanceled	rollback	재고 확보 부족 또는 결제 데이터 생성 오류로 인한 주문 취소	결제 취소

Awesome-Shopping 이벤트 설계 따라하기

awesome-payment-service의 list up 된 이벤트를 기준으로 이벤트 메시지를 구성합니다. Payload는 domain object를 기준으로 구성합니다.

awesome-payment-service 이벤트 메시지

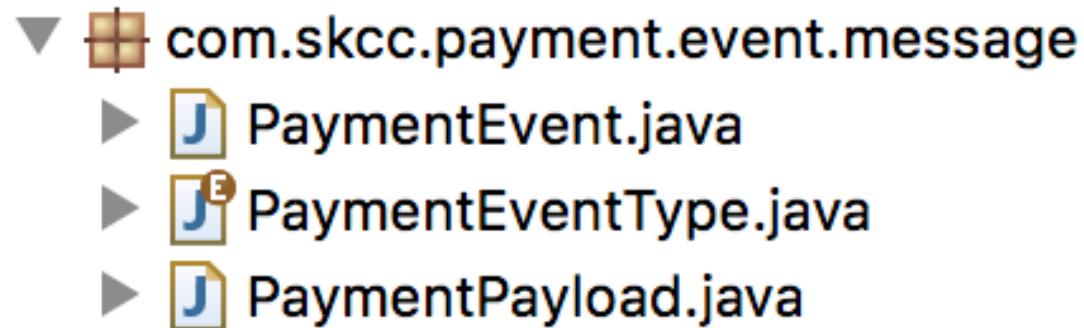
id	domain	domainId	eventType	payload	txId
1	payment	1	PaymentCreated ,PaymentCreateFailed ,PaymentCanceled ,PaymentCancelFailed	{ long id; long accountId; long orderId; string paymentMethod; string paymentDetail1; string paymentDetail2; string paymentDetail3; long price; string paid; string active; }	tx123123

Awesome-Shopping 이벤트 설계 따라하기

awesome-payment-service에 이벤트 메시지를 구현합니다.

1. `com.skcc.payment.event.message` 패키지를 추가합니다.

2. `PaymentEvent`, `PaymentPayload` class 및 `PaymentEventType` enum을 추가합니다.



Awesome-Shopping 이벤트 설계 따라하기

awesome-payment-service에 이벤트 메시지를 구현합니다.

3. PaymentEvent를 구현합니다.

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PaymentEvent {

    private long id;
    private String domain;
    private long paymentId;
    private PaymentEventType eventType;
    private PaymentPayload payload;
    private String txId;
    private String createdAt;

}
```

Awesome-Shopping 이벤트 설계 따라하기

awesome-payment-service에 이벤트 메시지를 구현합니다.

4. PaymentEventType을 구현합니다.

```
public enum PaymentEventType {  
  
    PaymentCreated  
    , PaymentCanceled  
    , PaymentCreateFailed  
    , PaymentCancelFailed  
  
}
```

Awesome-Shopping 이벤트 설계 따라하기

awesome-payment-service에 이벤트 메시지를 구현합니다.

5. PaymentPayload를 구현합니다.

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PaymentPayload {

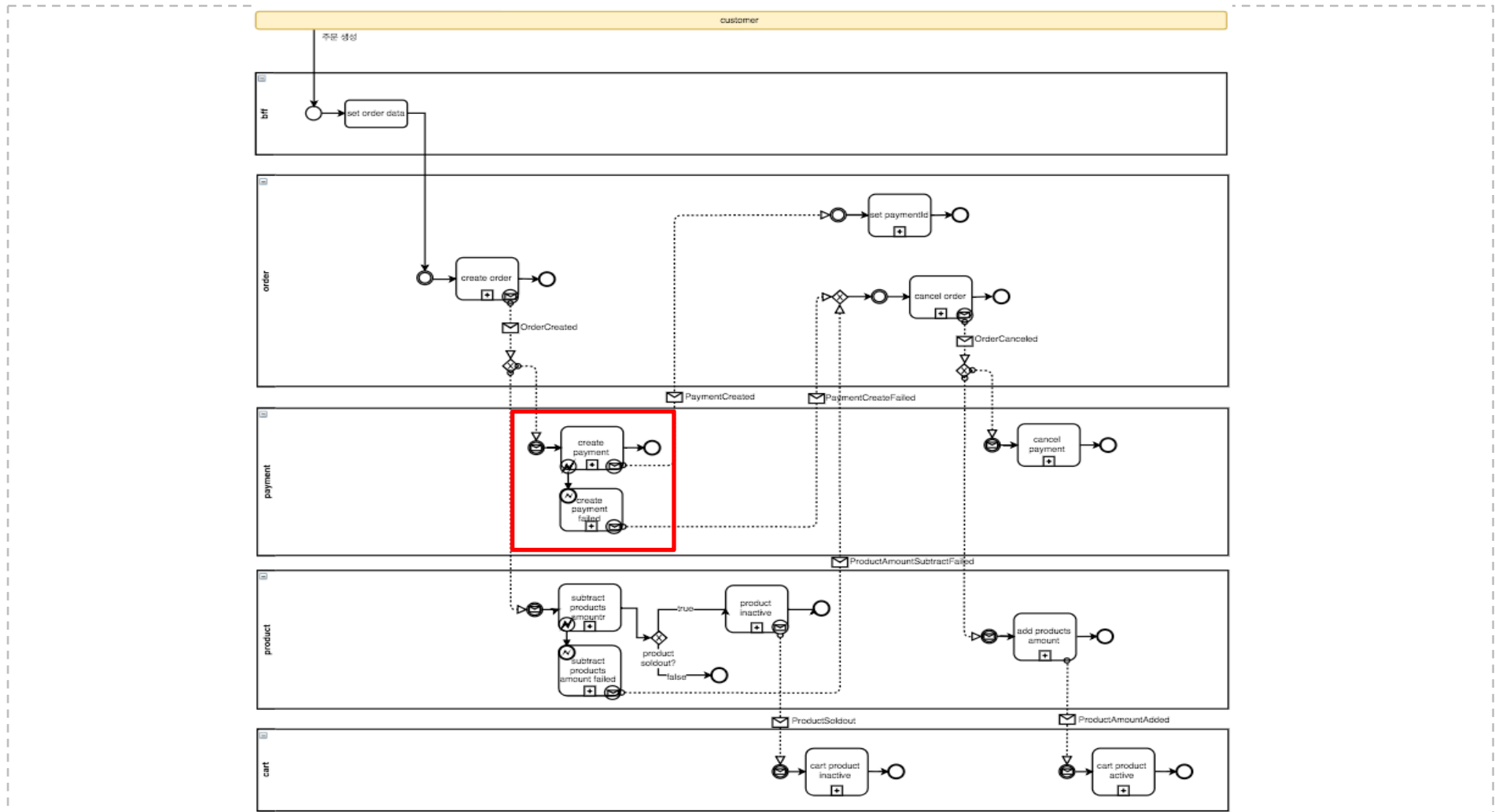
    private long id;
    private long accountId;
    private long orderId;
    private String paymentMethod;
    private String paymentDetail1;
    private String paymentDetail2;
    private String paymentDetail3;
    private long price;
    private String paid;
    private String active;

}
```

STEP02. EDM 구현 따라하기

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

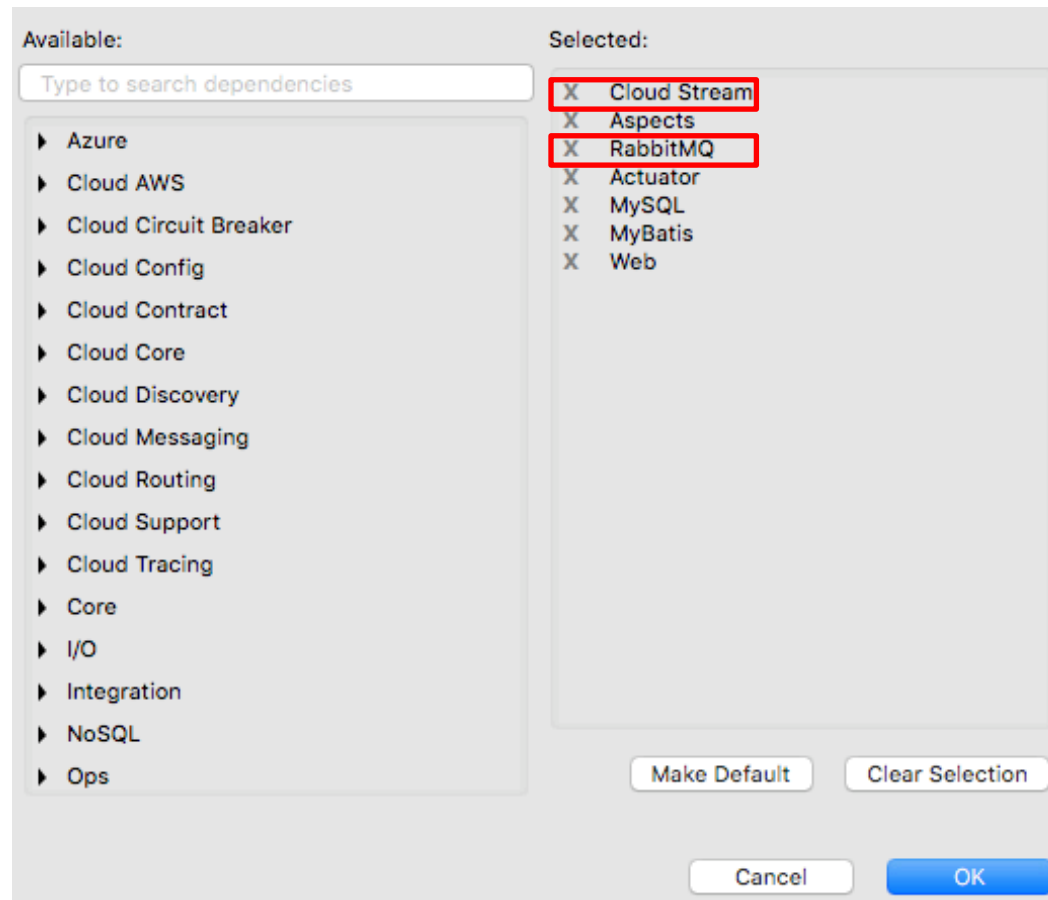
Awesome-payment-service 이벤트 시각화를 통해 구현이 필요한 이벤트 흐름을 확인합니다.



STEP 02. EDM 구현 따라하기

Spring Cloud Stream + RabbitMQ를 적용해 awesome-payment-service에 이벤트 관련 추가 기능을 구현합니다.

1. Spring cloud stream, rabbitmq dependency를 추가합니다.



awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

2. com.skcc.payment.event.channel 패키지에 PaymentInputChannelInterface를 구현합니다.

```
public interface PaymentInputChannel {  
  
    String orderCreated = "orderCreated";  
  
    @Input(PaymentInputChannel.orderCreated)  
    SubscribableChannel orderCreated();  
  
}
```

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

3. com.skcc.payment.event.channel 패키지에 PaymentOutputChannel Interface를 구현합니다.

```
public interface PaymentOutputChannel {  
  
    String paymentOutput = "paymentOutput";  
  
    @Output(PaymentOutputChannel.paymentOutput)  
    MessageChannel getMessageChannel();  
  
}
```


awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

4. application.yml에 OrderCreated 관련 설정을 추가합니다.

```
spring:
  cloud:
    stream:
      bindings:
        orderCreated:
          destination: orderExchange
          group: payment.orderCreated
          content-type: application/json
      paymentOutput:
        destination: paymentExchange
        content-type: application/json
```

Continue...

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

4. application.yml에 OrderCreated 관련 설정을 추가합니다.

Continue...

```
spring:
  cloud:
    stream:
      rabbit:
        bindings:
          orderCreated:
            consumer:
              autoBindDlq: true
              bindingRoutingKey: order.OrderCreated.#
          paymentOutput:
            producer:
              routing-key-expression: headers['routeTo']
```

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

5. com.skcc.payment.subscribe 패키지에 이벤트 구독을 위한 PaymentSubscribe class를 구현합니다.

```
@Component
@EnableBinding(PaymentInputChannel.class)
public class PaymentSubscribe {

    중략...

    @StreamListener(PaymentInputChannel.orderCreated)
    public void receiveOrderCreatedEvent(OrderEvent orderEvent) {
        this.paymentService.createPaymentAndCreatePublishEvent(orderEvent);
    }
}
```

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

6. com.skcc.payment.publish 패키지에 이벤트 발행을 위한 PaymentPublish class를 구현합니다.

```
@Component
@EnableBinding(PaymentOutputChannel.class)
public class PaymentPublish {

    중략 ...

    public void send(PaymentEvent paymentEvent) {
        this.paymentOutputChannel.getMessageChannel().send(
            MessageBuilder.withPayload(paymentEvent).setHeader(
                "routeTo", domain + "." + paymentEvent.getEventType()).build());
    }
}
```

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

6. com.skcc.payment.repository 패키지에 이벤트 발행을 위한 PaymentMapper를 구현합니다.

```
@Mapper
public interface PaymentMapper {
    중략...

    public void createPaymentEvent(PaymentEvent paymentEvent);
}
```

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

7. src/main/resources/mappers/PaymentMapper.xml을 구현합니다.

```
<insert id="createPaymentEvent" parameterMap="PaymentEvent">
    insert into payment_events (id, domain, paymentId, eventType, payload, txId,
createdAt)
    values(#{id}, #{domain}, #{paymentId}, #{eventType}
        , #{payload, typeHandler = com.skcc.payment.config.PaymentPayloadJsonTypeHandler}
        , #{txId}, NOW())
</insert>
```

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

8. com.skcc.payment.service 패키지에 PaymentService를 구현합니다.

```
@Service
public class PaymentService {

    public boolean createPaymentAndCreatePublishEvent(OrderEvent orderEvent) {
        boolean result = false;
        Payment payment = this.convertOrderEventToPayment(orderEvent);
        payment.setPaid("unpaid");
        try {
            this.paymentService.createPaymentAndCreatePublishPaymentCreatedEvent(orderEvent.getTxId(), payment);
            result = true;
        } catch (Exception e) {
            try {
                result = false;
                e.printStackTrace();
                this.paymentService.createPublishPaymentCreateFailedEvent(orderEvent.getTxId(), payment);
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
        return result;
    }

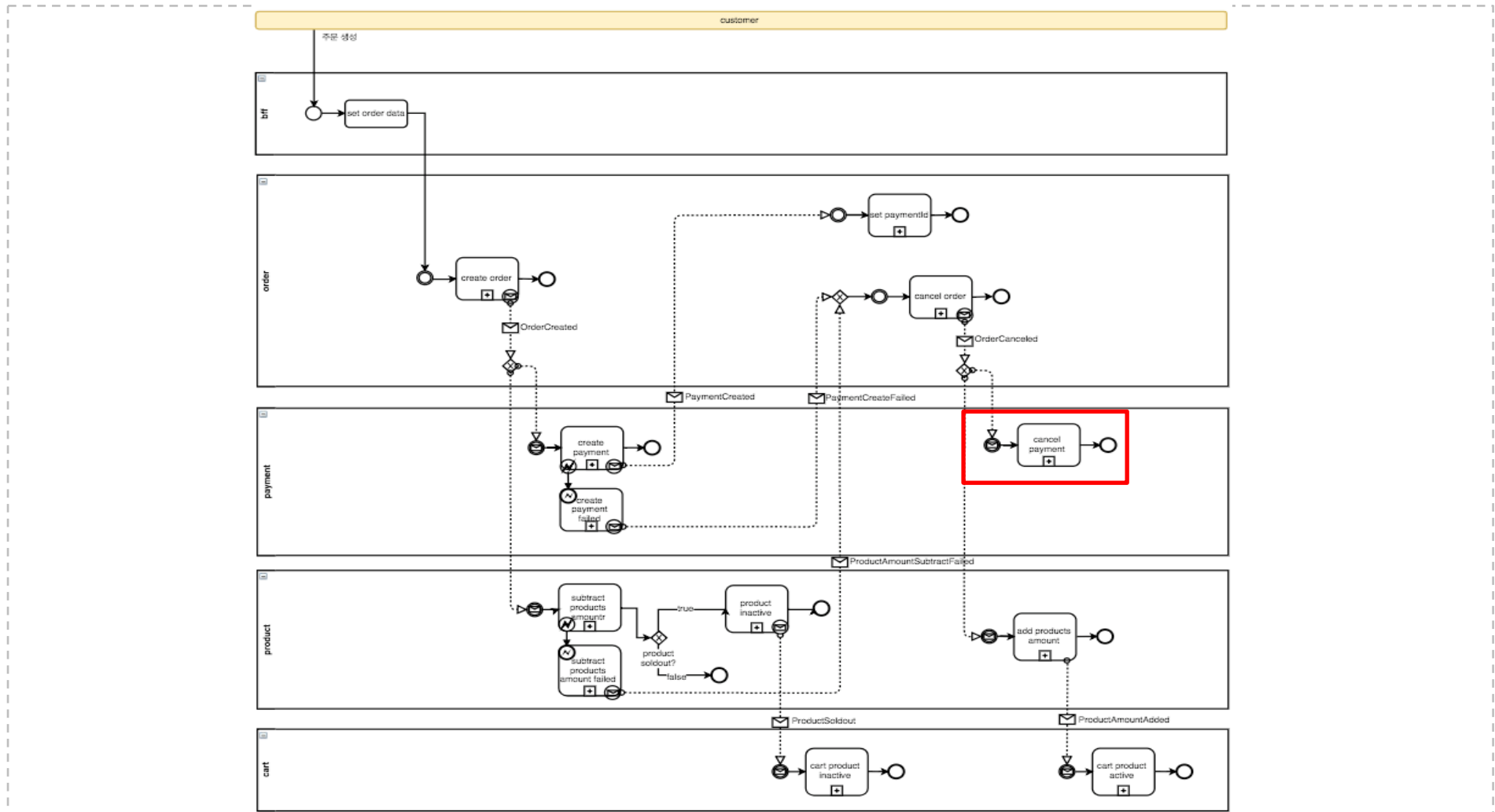
    @Transactional
    public void createPaymentAndCreatePublishPaymentCreatedEvent(String txId, Payment payment) throws Exception {
        this.createPaymentValidationCheck(payment);
        Payment resultPayment = this.createPayment(payment);
        this.createPublishPaymentEvent(txId, resultPayment, PaymentEventType.PaymentCreated);
    }

    @Transactional
    public void createPublishPaymentCreateFailedEvent(String txId, Payment payment) throws Exception {
        this.createPublishPaymentEvent(txId, payment, PaymentEventType.PaymentCreateFailed);
    }
}
```

STEP02. EDM 구현 따라하기

awesome-payment-service에 OrderCanceled 이벤트 관련 기능을 구현합니다.

Awesome-payment-service 이벤트 시각화를 통해 구현이 필요한 이벤트 흐름을 확인합니다.



awesome-payment-service에 OrderCanceled 이벤트 관련 기능을 구현합니다.

9. com.skcc.payment.event.channel 패키지에 PaymentInputChannelInterface를 구현합니다.

```
public interface PaymentInputChannel {  
  
    String orderCanceled = "orderCanceled";  
  
    @Input(PaymentInputChannel.orderCanceled)  
    SubscribableChannel orderCanceled();  
  
}
```

awesome-payment-service에 OrderCanceled 이벤트 관련 기능을 구현합니다.

10. application.yml에 OrderCreated 관련 설정을 추가합니다.

```
spring:
  cloud:
    stream:
      bindings:
        orderCanceled:
          destination: orderExchange
          group: payment.orderCanceled
          content-type: application/json
      rabbit:
        bindings:
          orderCanceled:
            consumer:
              autoBindDlq: true
              bindingRoutingKey: order.OrderCanceled.#
```

awesome-payment-service에 OrderCanceled 이벤트 관련 기능을 구현합니다.

11. com.skcc.payment.subscribe 에 이벤트 구독을 위한 PaymentSubscribe class를 구현합니다.

```
@Component
@EnableBinding(PaymentInputChannel.class)
public class PaymentSubscribe {

    중략 ...

    @StreamListener(PaymentInputChannel.orderCanceled)
    public void receiveOrderCanceledEvent(OrderEvent orderEvent) {
        this.paymentService.cancelPaymentAndCreatePublishEvent(orderEvent);
    }
}
```

awesome-payment-service에 OrderCanceled 이벤트 관련 기능을 구현합니다.

12. com.skcc.payment.service 패키지에 PaymentService를 구현합니다.

```
@Service
public class PaymentService {

    public boolean cancelPaymentAndCreatePublishEvent(OrderEvent orderEvent) {
        boolean result = false;
        String txId = orderEvent.getTxId();
        Payment payment = this.findPaymentByOrderId(orderEvent.getOrderId());
        if(payment == null)
            return result;
        try {
            this.paymentService.cancelPaymentAndCreatePublishPaymentCanceledEvent(txId, payment);
            result = true;
        } catch(Exception e) {
            try {
                result = false;
                e.printStackTrace();
                this.paymentService.createPublishPaymentCancelFailedEvent(txId, payment);
            } catch(Exception e1) {
                e1.printStackTrace();
            }
        }
        return result;
    }

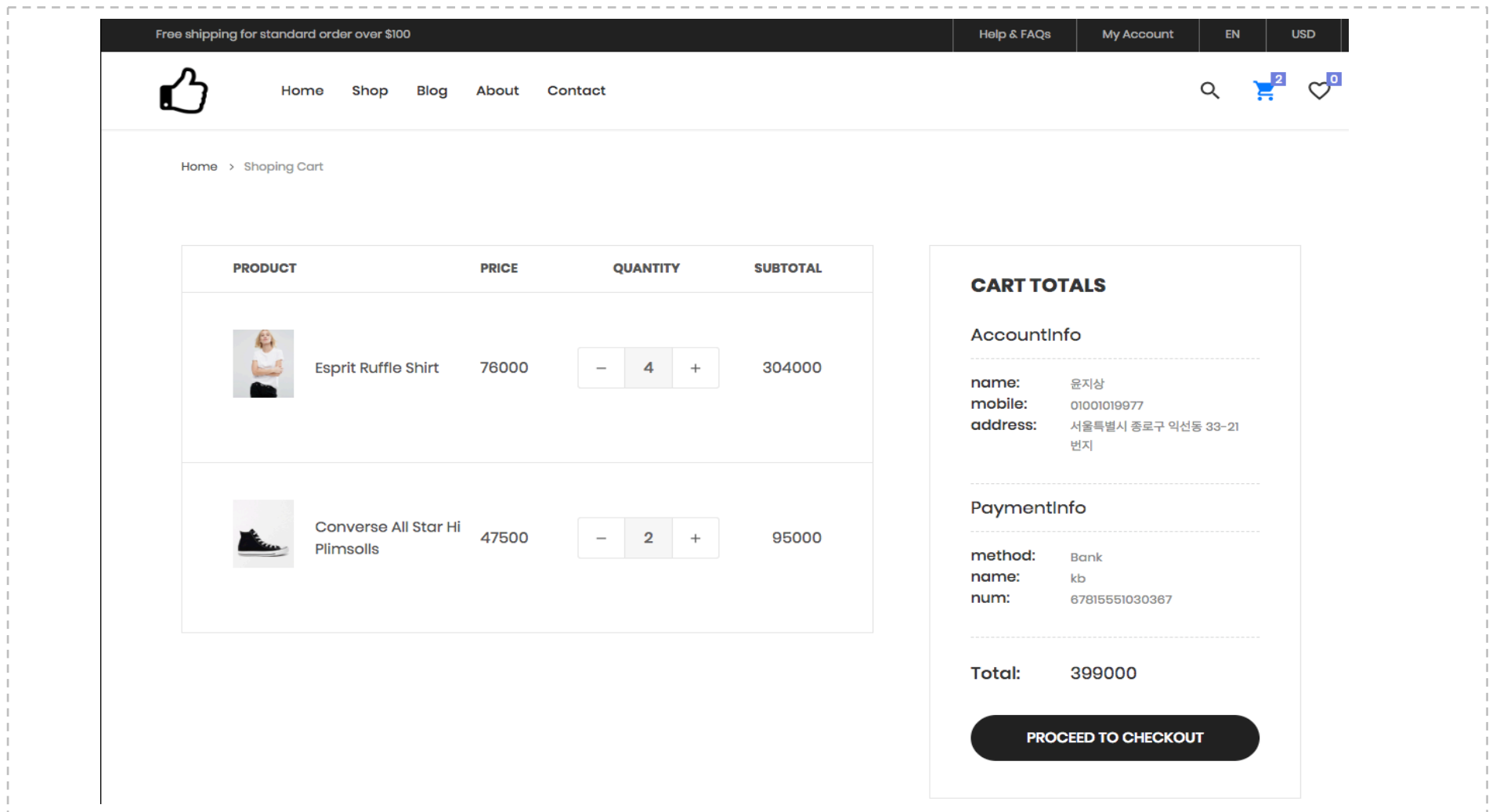
    @Transactional
    public void cancelPaymentAndCreatePublishPaymentCanceledEvent(String txId, Payment payment) throws Exception{
        this.cancelPaymentValidationCheck(payment);
        this.cancelPayment(payment);
        this.createPublishPaymentEvent(txId, payment, PaymentEventType.PaymentCanceled);
    }

    @Transactional
    public void createPublishPaymentCancelFailedEvent(String txId, Payment payment) throws Exception{
        this.createPublishPaymentEvent(txId, payment, PaymentEventType.PaymentCancelFailed);
    }
}
```

STEP 02. EDM 구현 따라하기

awesome-payment-service에 OrderCreated 이벤트 관련 기능을 구현합니다.

13. 완성된 화면을 확인합니다.



Event Driven MicroService 적용시 고려사항

Event Driven MicroService를 적용할 때 고려사항에 대해 알아보겠습니다.

- **Tooling**
 - 이벤트 설계 후 이벤트 메시지 생성, 이벤트 list up, enable 관리 등 기능 수행
- **이벤트 설계**
 - 기존 MVC에 Pub/Sub 이 추가된 방식 -> 적절한 이벤트 설계 방법 필요
- **테스트**
 - Event Driven 적용 후 테스트 시나리오, 성능 측정 등 고려 필요
- **이벤트 모니터링**
 - 메시지 스트림 모니터링 및 Failed 이벤트 모니터링 여부 결정 등
- **고가용 구성**
 - 메시지 큐를 사용하는 환경에서 기존에 추가되는 Backing Service 구성에 따른 고려 필요
- **어플리케이션 흐름**
 - 비동기 방식으로 시스템 내 통합을 수행하기 때문에 하나의 기능을 수행하기 위해 앞단에서 필요한 데이터들을 모두 받을 수 있게 재구성 필요

감사합니다

