Event-Storming based MSA training

Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

Target Domain : 온라인 쇼핑몰 (12번가)

- **상호명 : 12번가**

- **Vision & Mission**

  “고객으로부터 신뢰받는 최고의 커머스 포털”

  - 우리는 세상 모든 것을 판매하는 쇼핑의 관문으로의 진화를 꿈꿉니다. 고객이 원하는 모든 유형의 상품과 서비스를 만날 수 있는 곳, 바로 12번가입니다.

  - 우리는 우리의 기술로 탐색부터 구매까지, 쇼핑의 모든 과정에서 고객에게 최적의 솔루션을 제공할 수 있을 것이라 믿습니다. 쇼핑에 관한 절대적인 편의를 제공하는 곳, 바로 12번가입니다.

  - 우리는 고객의 슬기로운 쇼핑 생활을 위해 끊임없이 노력하고 있습니다. 풍성한 콘텐츠로 쇼핑 그 이상의 재미와 감동을 선사하는 곳, 바로 12번가입니다.

12번가 User Stories

- MD가 상품정보(상품ID, 상품명, 상품가격, 재고수량, 설명, 추천ID)를 등록/ 수정/ 삭제한다.
- 고객이 주문과 취소를 함에 따라서 상품 재고량이 변경된다.
- 고객이 상품명으로 조회 시, 상품정보가 조회 된다.
- 시스템이 상품을 출력할 때, 추천 상품들이 동시에 출력된다.
- 품절된 상품에 대해서는 구매 대기가 가능하다.
- 품절된 상품이 재 입고될 때, 구매 대기 목록에 있는 고객에게 알림이 발송된다.
- 고객이 상품관련 질문을 올리면 MD가 회답한다.
- 고객이 상품을 장바구니에 담는다.
- 고객이 주문할 때, 상품과 구매개수 및 옵션을 선택한다.
- 고객이 주문내역 변경(수정, 취소)이 가능하다.
- 주문상태가 변경될 때마다 고객에게 알림이 가야 한다.
- “카드결제” 결제 시 결제대행사가 한도체크 및 결제행위를 실행된다.
- 주문취소가 완료되면 결제된 금액은 자동으로 환불된다.
- 시스템은 결제 완료된 주문의 배송 정보를 물류사에 전달한다.
- 배송정보에는 상품ID, 주문정보, 고객정보 등의 내용이 포함된다.
- 고객이 배송중인 상품에 대해 배송현황 조회가 가능하다.
- 고객이 입력한 배송관련 질의사항은 배송팀에서 대응한다.
- 시스템은 고객의 배송완료 또는 반품완료 시, 고객에게 상태 알림을 제공한다.
- 주문취소는 주문상품 반품, 환불완료가 되면 최종 주문취소가 완료된다.
- 고객의 상품관련 컴플레인은 상품관리팀에게 연결이 된다.
- 구매자에 대해, 잦은 주문 (승인 및 취소) 반복 시 블랙리스트로 등록되고, 등록된 사용자는 주문 시에 대한 규제를 가한다.
- 판매자에 대해, 물품 배송이 자주 지연될 경우(기간 내 몇 회 이상), 잘못된 정보로 홍보할 경우(기간 내 몇 건 이상), 고객 불만 사항이 많을 경우(기간 내 몇 건 이상), 판매자가 블랙리스트로 될 경우 상품 등록 시 규제를 가한다.
- 모든 비즈니스 메일은 마케팅 관리팀에서 발송을 담당한다.

Teaming & KPI Definition

Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

AS-IS: Pain-points

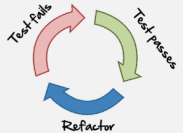
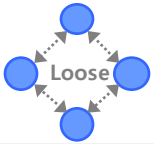
A 사 의료분야 SaaS 운영

- 서비스 업그래이드가 수시로 요청이 들어와 거의 매일 야근중. 개발자 행복지수가 매우 낮음.
- 한팀의 반영이 전체팀의 반영에 영향을 주어 거의 매일 야근해야 함. 행복지수 낮음을 토로함.
- 테넌트별 다형성 지원을 제대로 하지 못하여 가입고객이 늘 때마다 전체 관리 비용이 급수로 올라가는 한계에 봉착함
- 자체 IDC를 구성하여 하드웨어, 미들웨어 구성을 직접해야 하는 비용문제.

B 사 제조분야 SaaS 운영

- 운영팀과 개발팀이 분리되어 개발팀의 반영을 운영팀이 거부하는 사례 발생
- 개발팀은 새로운 요건을 개발했으나, 이로인해 발생하는 오류가 두려워 배포를 꺼려함
- 현재 미국, 일본, 유럽 등 수요가 늘어나는 상황이나, 상기한 문제로 신규 고객의 요구사항을 받아들이지 못하는 상황
- 수동 운영의 문제로, SLA 준수가 되지 못하여 고객 클레임이 높은편
- 기존 모놀로씩 아키텍처의 한계로 장기적인 발전의 한계에 봉착

To be Agile

Delivery & Requirement management	**MVP**(Minimum Viable Product) & **Incremental delivery**	
S/W Engineering	**Test driven Development & Continuous Refactoring**	
Technical Design	**Loosely Coupled Architecture**	
Process & Collaboration	**Continuously Improving**	

DevOps: Issues

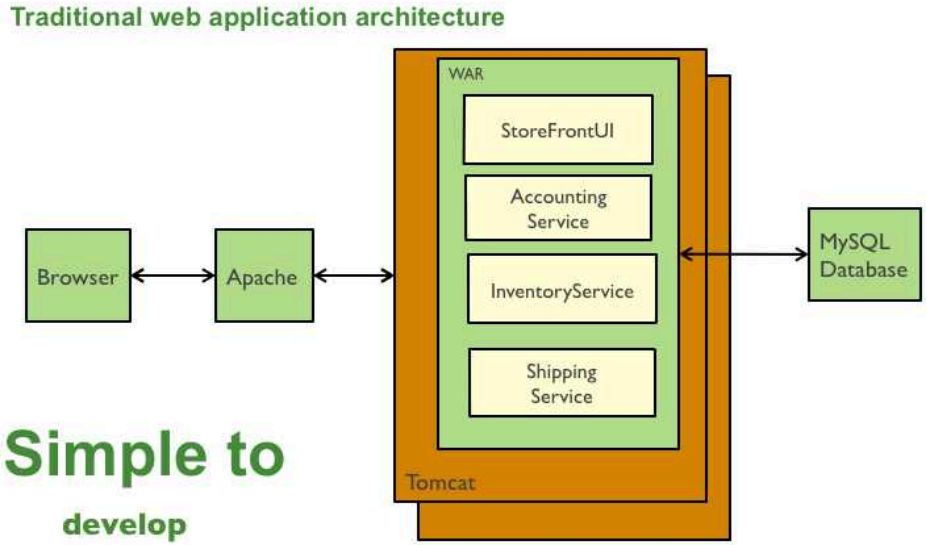
Continuous Delivery

Amazon, Google, Netflix, Facebook, Twitter는 얼마나 자주 배포할까요?

Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	Minutes	High	High
Google	5,500 / day	Minutes	High	High
Netflix	500 / day	Minutes	High	High
Facebook	1 / day	Hours	High	High
Twitter	3 / week	Hours	High	High
Typical enterprise	Once every 9 months	Months or quarters	Low / Medium	Low / Medium

출처: 도서 The Phoenix Project

Monolithic Architecture



→ 모든 서비스가 한번에 재배포
→ 한팀의 반영을 위하여 모든 팀이 대기
→ 지속적 딜리버리가 어려워

Jeff Bezos Mandate

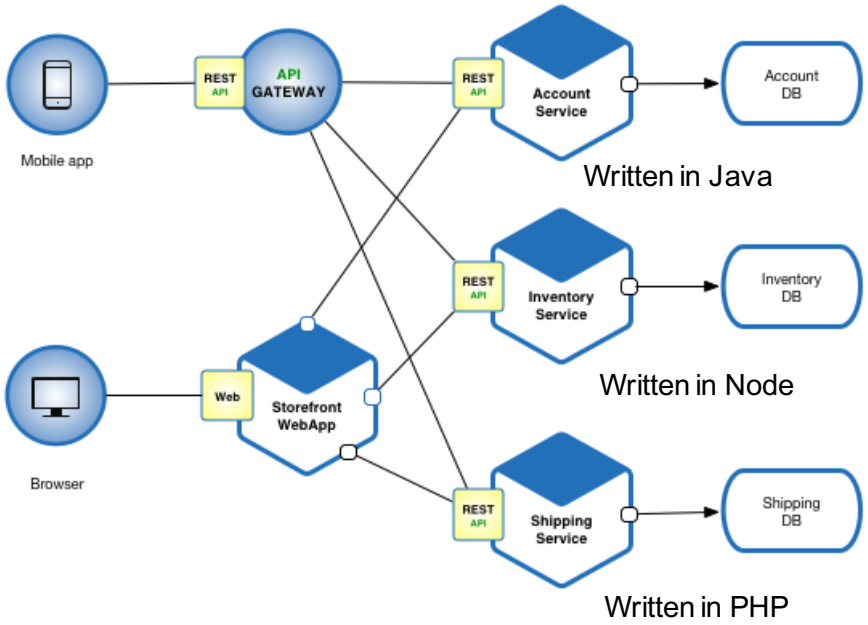
1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. **There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.**

   …
4. The team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
5. Anyone who doesn't do this will be fired.

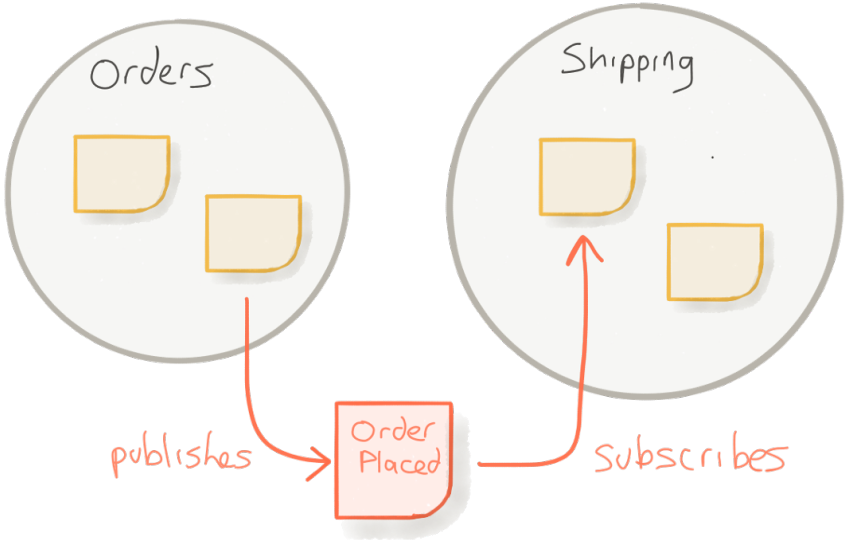
Approach #1: Micro Service Architecture

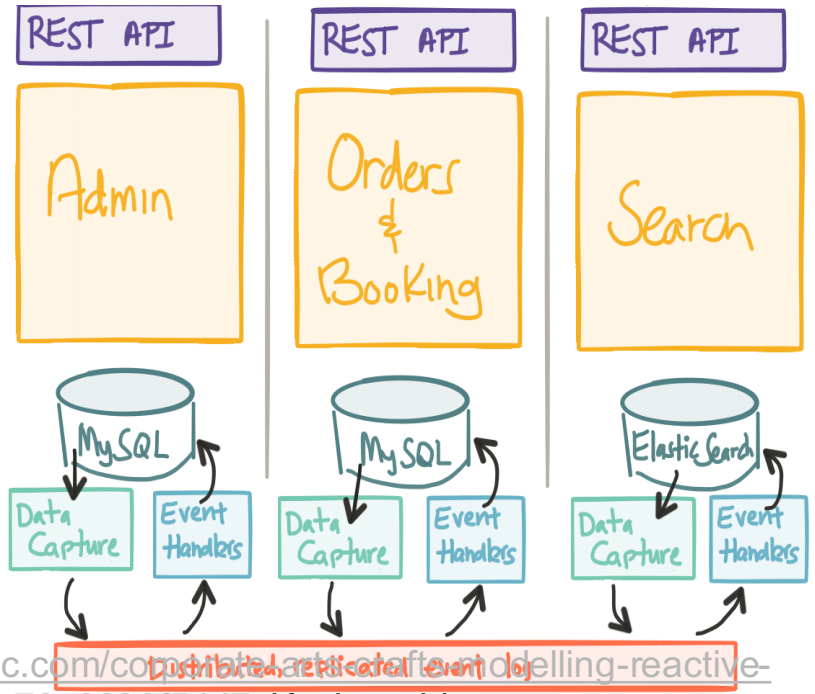
Contract based, Polyglot Programming

→ Separation of Concerns, Parallel Development, Easy Outsourcing



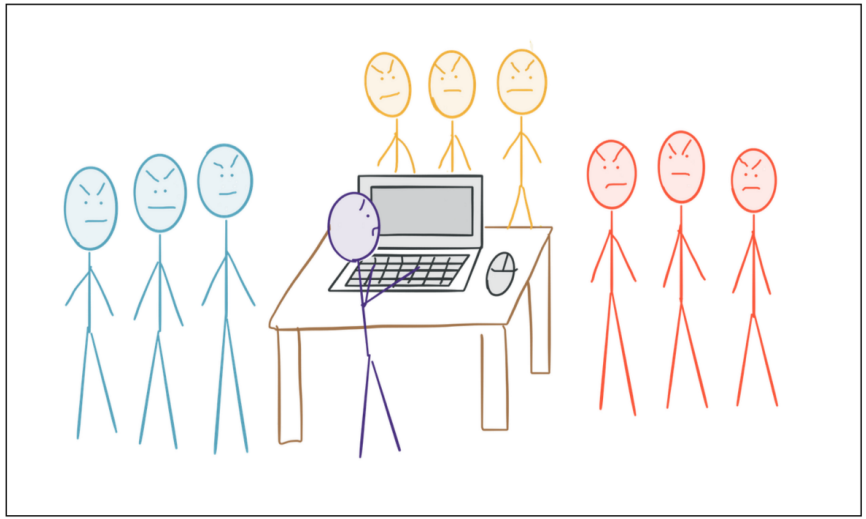
Approach #2: Event Driven Architecture

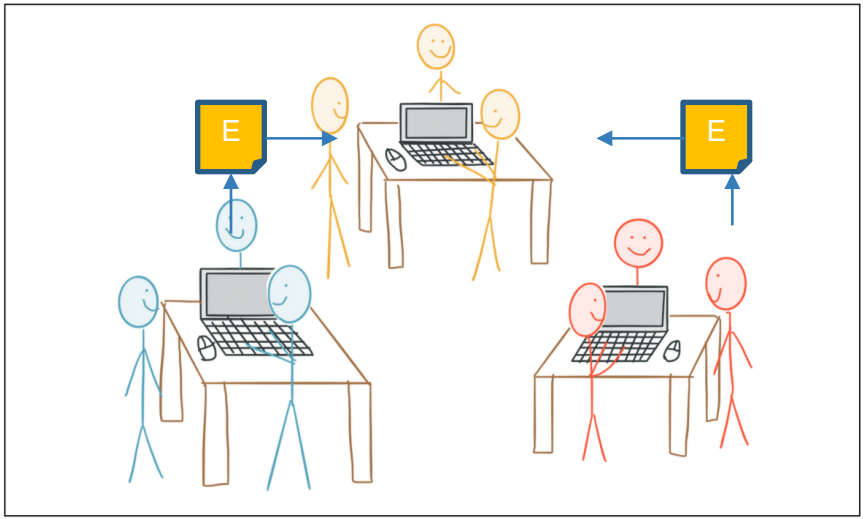




Source: https://blog.redelastic.com/compositional-artefacts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

Approach #2: Event Driven Architecture





Monolith:

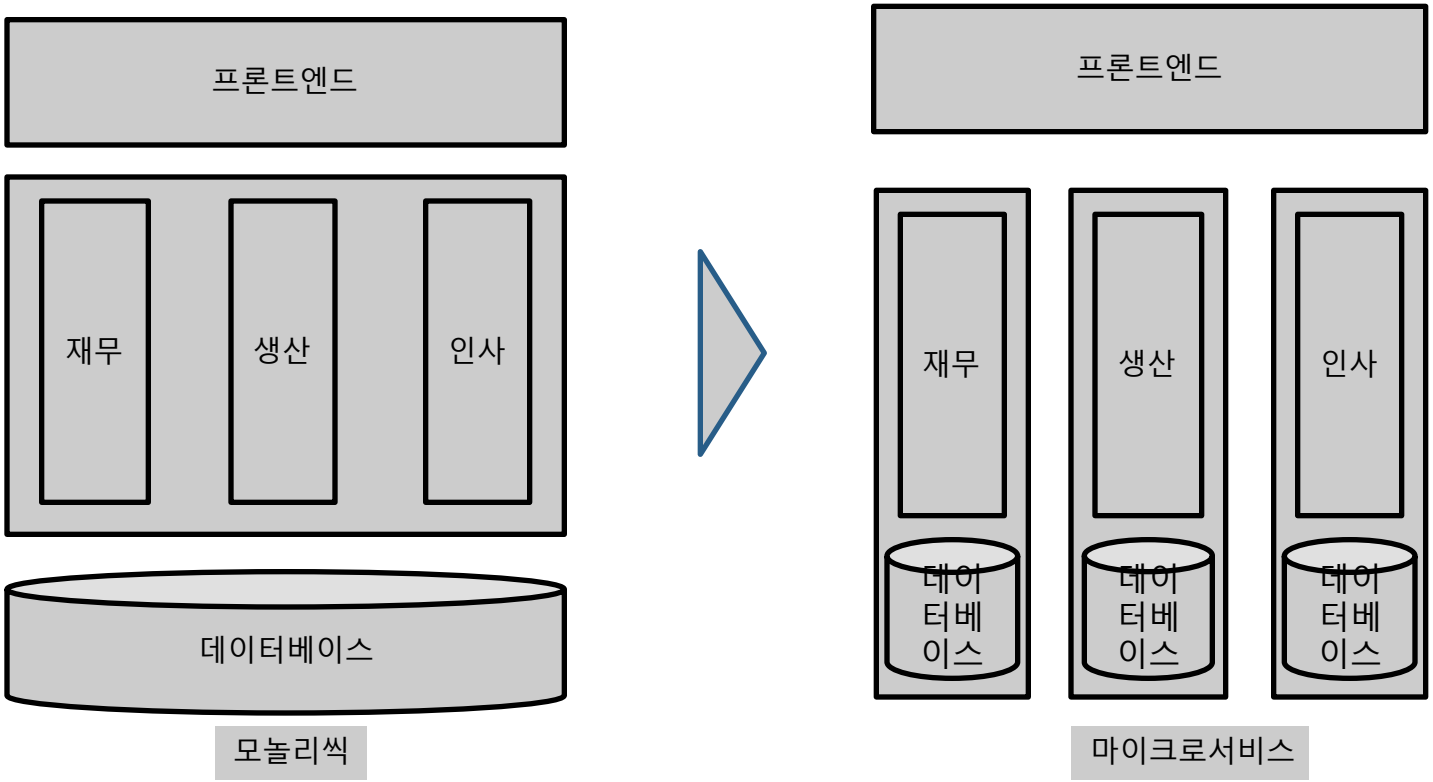
- With Canonical Model
- (“Rule Them All” model)

Reactive Microservices:

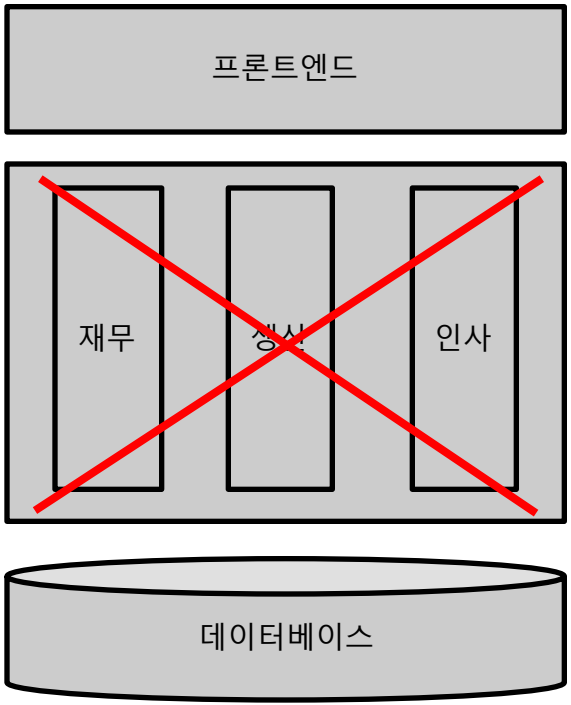
- Autonomously designed Model(Document)
- Polyglot Persistence

Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

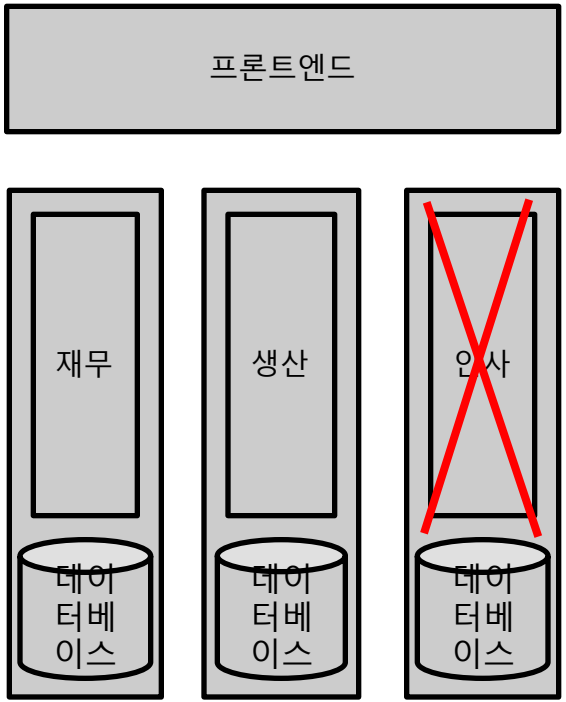
적용효과


프론트엔드
재무
생산
인사
데이터베이스
모놀리씩
프론트엔드
재무
생산
인사
데이터베이스
데이터베이스
데이터베이스
마이크로서비스

효과 – 장애 최소화

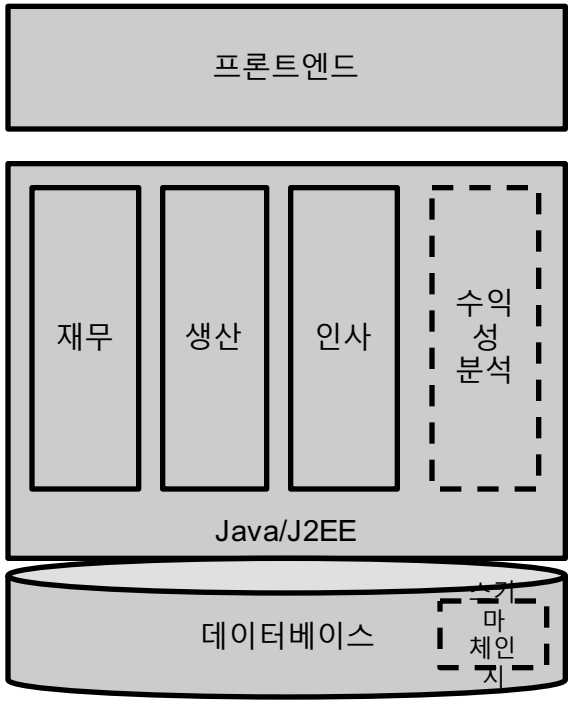


장애가 전파되며, 수동 복구로 장애 지연

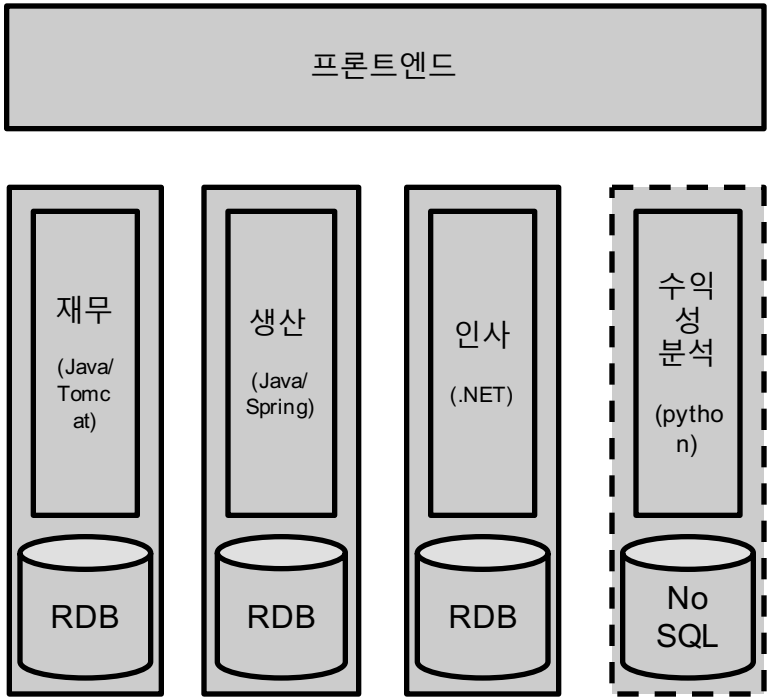


장애가 격리되며, 자동으로 복구됨(이전버전)

효과 – 기능 확장

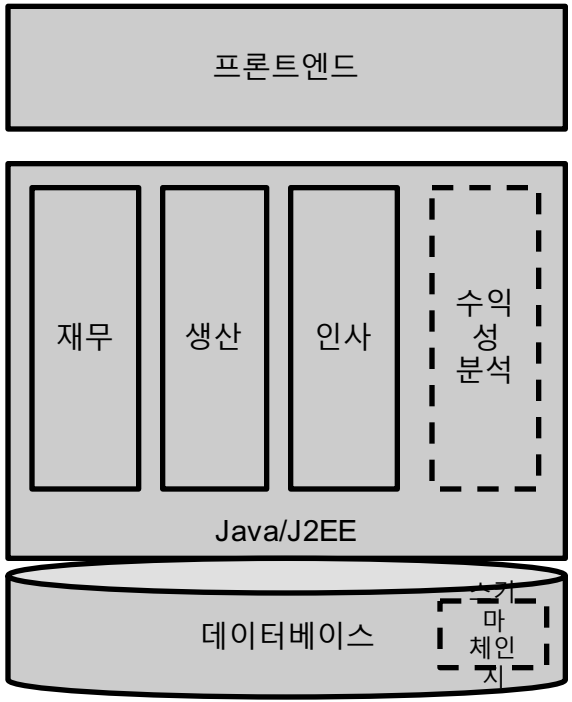


기존 코드의 수정 → Big Impact

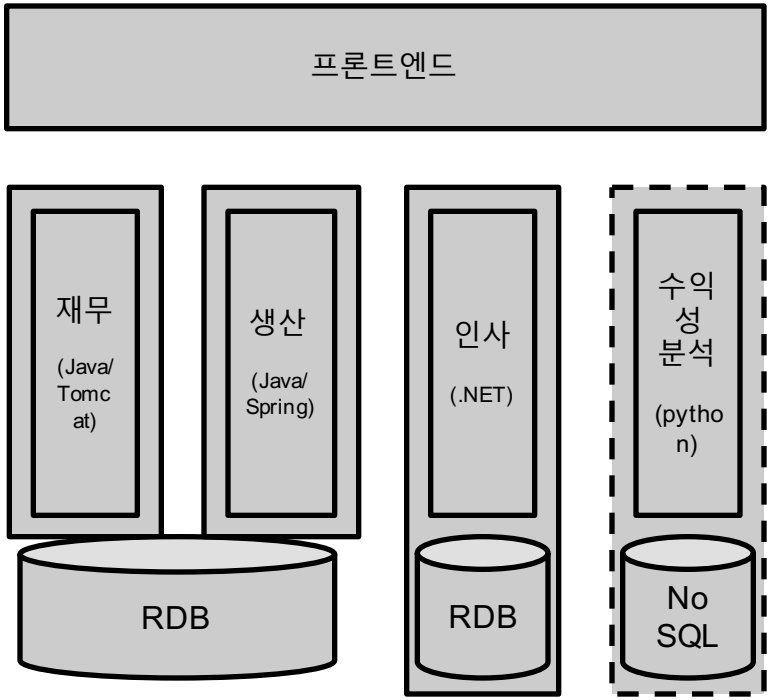


신규 마이크로서비스의 추가 → Minimal Impact
자율적 기술선택(플랫폼 & 데이터레이아웃)

효과 – 기능 확장

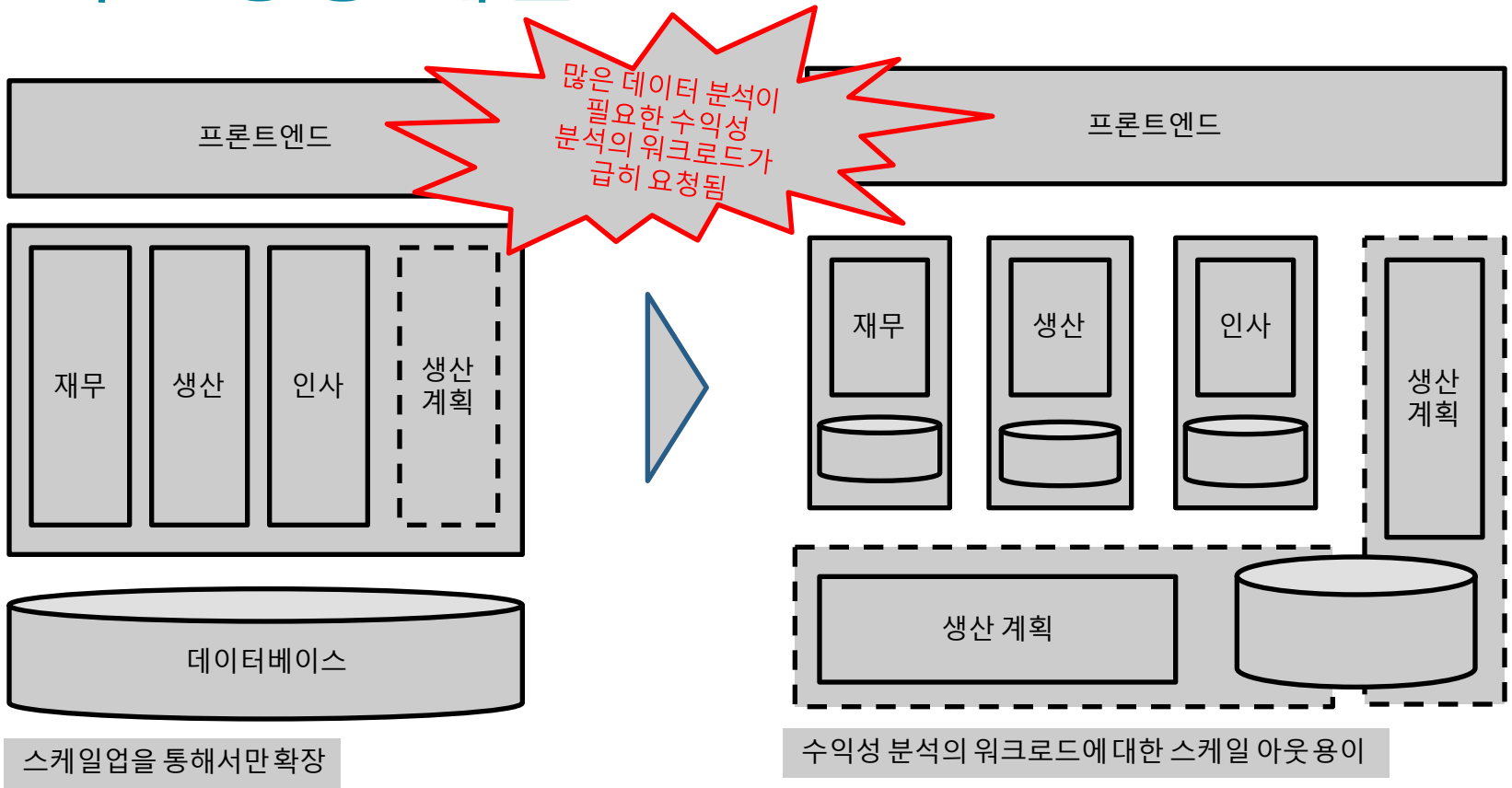


기존 코드의 수정 → Big Impact



신규 마이크로서비스의 추가 → Minimal Impact
자율적 기술선택(플랫폼 & 데이터레이아웃)

효과 – 성능 배분


많은 데이터 분석이
필요한 수익성
분석의 워크로드가
급히 요청됨
프론트엔드
재무
생산
인사
생산
계획
데이터베이스
스케일업을 통해서만 확장
프론트엔드
재무
생산
인사
생산
계획
생산 계획
수익성 분석의 워크로드에 대한 스케일 아웃 용이

Micro Service Architecture

- 변경된 서비스만 재배포

→ Side effect 최소화

- 자율성

→ 각 서비스에 대한 자유로운 언어, 아키텍처, 아웃소싱 용이

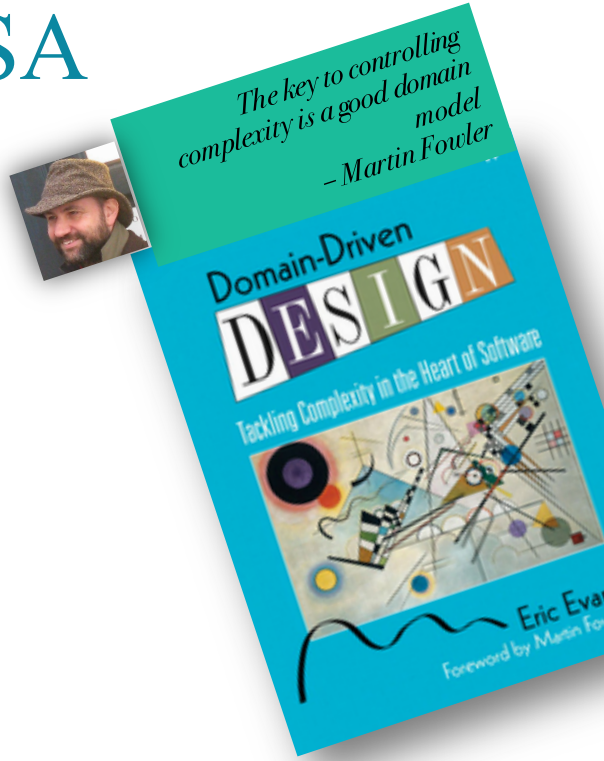
- 병렬 개발, 타임 투 마켓, 린 개발

Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

Domain-Driven Design & MSA



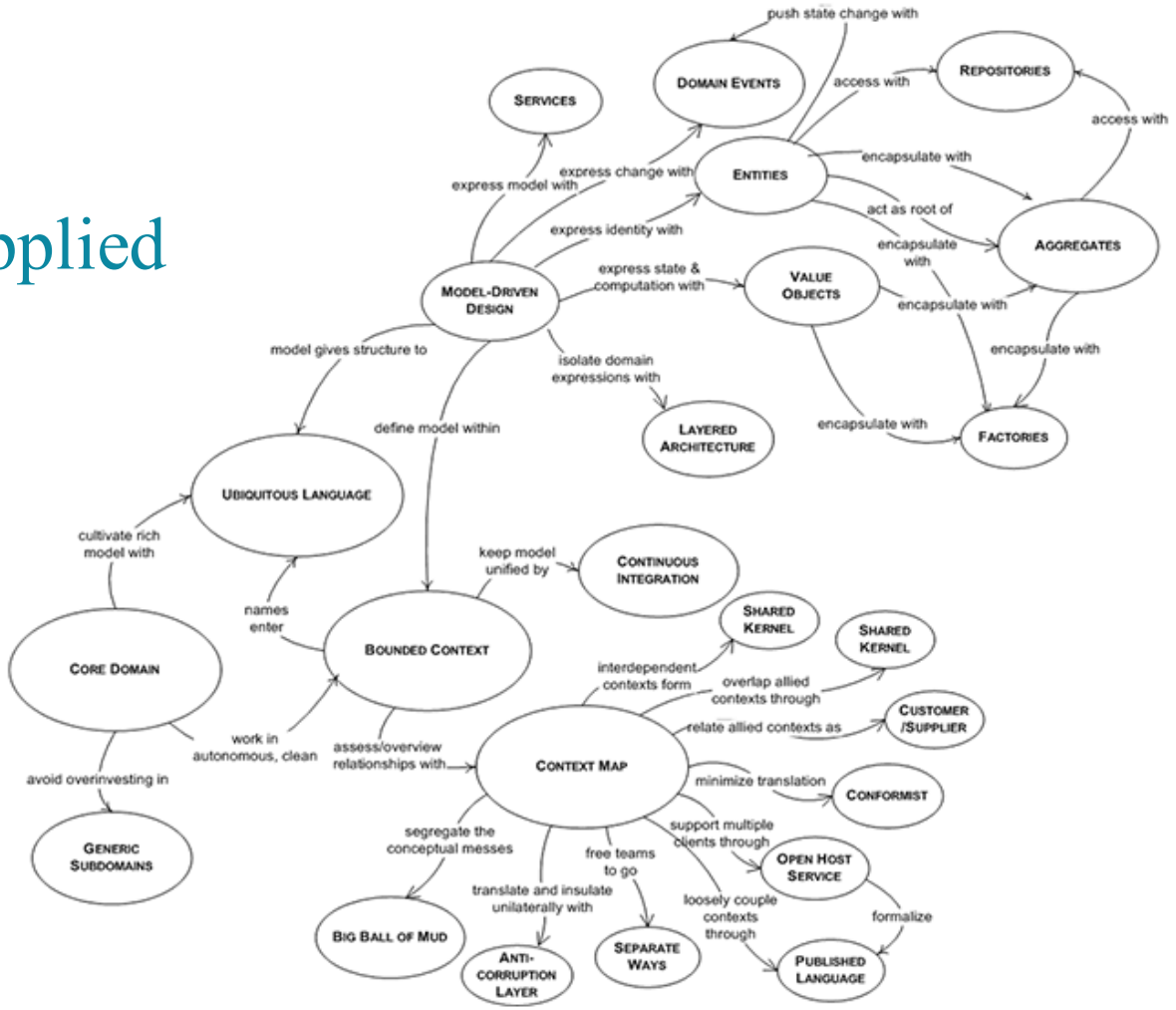
DDD for MSA

- Bounded Context → 어떤 단위가 마이크로 서비스가 될 수 있는가?
- Context Mapping → 서비스를 결합 할 때는 어떻게 결합할 것인가?
- Domain Events → Event-driven Architecture 도출

Applicable Practices:

ㄱ. 이벤트 스토밍과 BPMN: 이벤트, 커맨드, 액터, 어그리게이트 식별
ㄴ. UML 과 컨텍스트 맵: 보편언어, 바운디드 컨텍스트 식별

DDD Pattern Applied

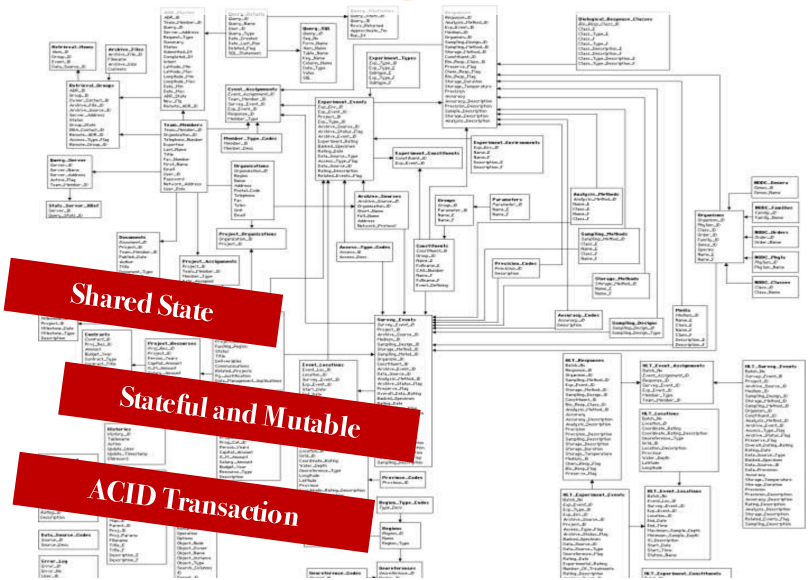


DDD: Domain Object Classifications

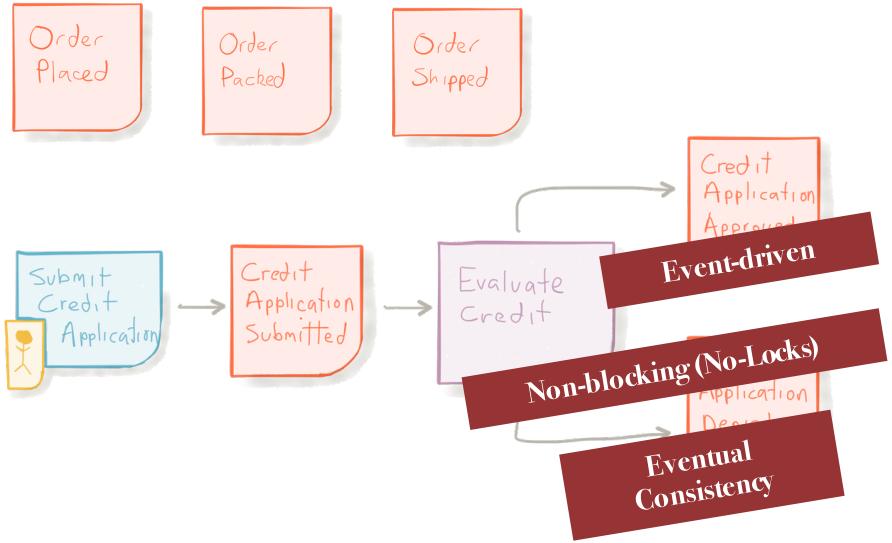
Classification	Description	Example
Domain Event	▪ Captures the memory of something interesting which affects the domain	▪ I go to Babur's for a meal on Tuesday, and pay by credit card. This might be modeled as an event, whose event type is 'make purchase', whose subject is my credit card, and whose occurred date is Tuesday. If Babur's uses an old manual system and doesn't transmit the transaction until Friday, the noticed date would be Friday.
Entity and Aggregate	▪ Objects that have a distinct identity that runs through time and different representations.	▪ Usually big things like Customer, Ship, Rental Agreement.
Service	▪ A standalone operation within the context of your domain. A **Service Object** collects one or more services into an object. Typically you will have only one instance of each service object type within your execution context.	▪ Services are usually accesses to external resources like Database Connection, Messaging Gateway, Repository, Product Factory.

Event Storming: DDD 를 쉽게하는 방법

- **이벤트스토밍은** 시스템에서 발생하는 **이벤트를중심 (Event-First) 으로** 분석하는 기법으로 특히 Non-blocking, Event-driven 한 **MSA 기반 시스템을 분석에서 개발까지 필요한 도메인에 대한 탁월하게 빠른 이해를 도모**하는데 유리하다.
- 기존의 유즈케이스나 클래스 다이어그래밍 방식은 고객 인터뷰나 엔티티 구조를 인지하는 방식과 다르게 별다른 **사전 훈련된 지식과 도구 없이 진행할 수 있다.**
- 진행과정은 참여자 워크숍 방식의 방법론으로 결과는 **스티키 노트를 벽에 붙힌 것으로 결과가 남으며, 오랜지색 스티키 노트들의 연결로 비즈니스 프로세스가 도출되며** 이들을 이후 BPMN과 UML 등으로 정재하여 전환할 수 있다.



VS.



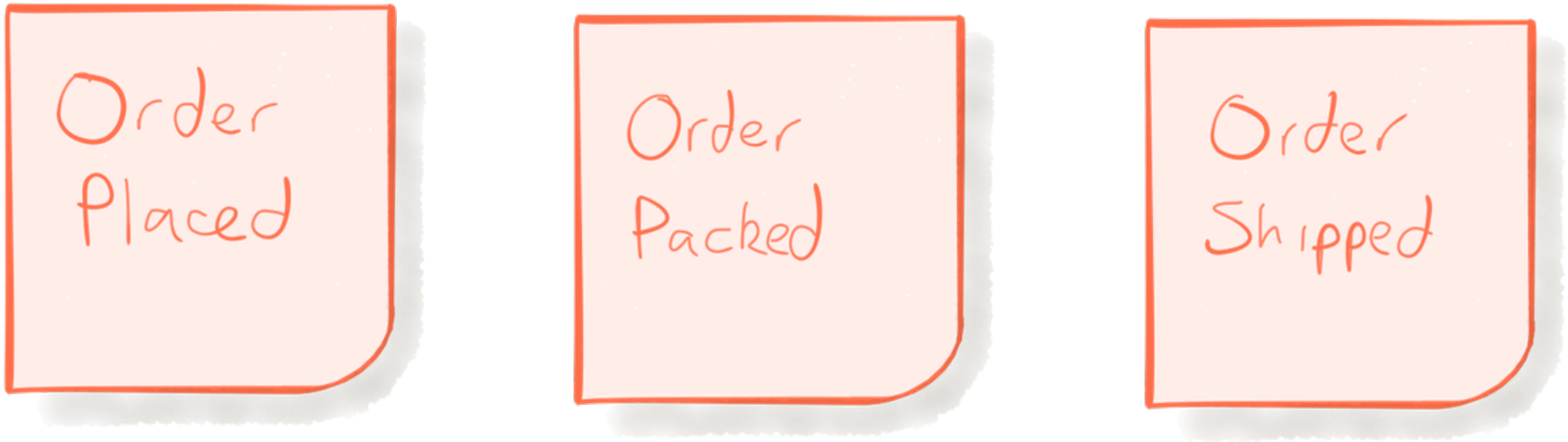
What's an Event?

An Event message is non-prescriptive of what should happen in other services.

Events always carry a name in its past-tense form:
OrderWasAccepted, OrderHasShipped, CustomerWasReimbursed

Other qualities
- Immutable, i.e. content cannot be changed
- Always carries the ID of the Business Object it relates to
- An event can and will typically will be published to multiple consumers.
  - The publisher of the event does not know who the recipients are

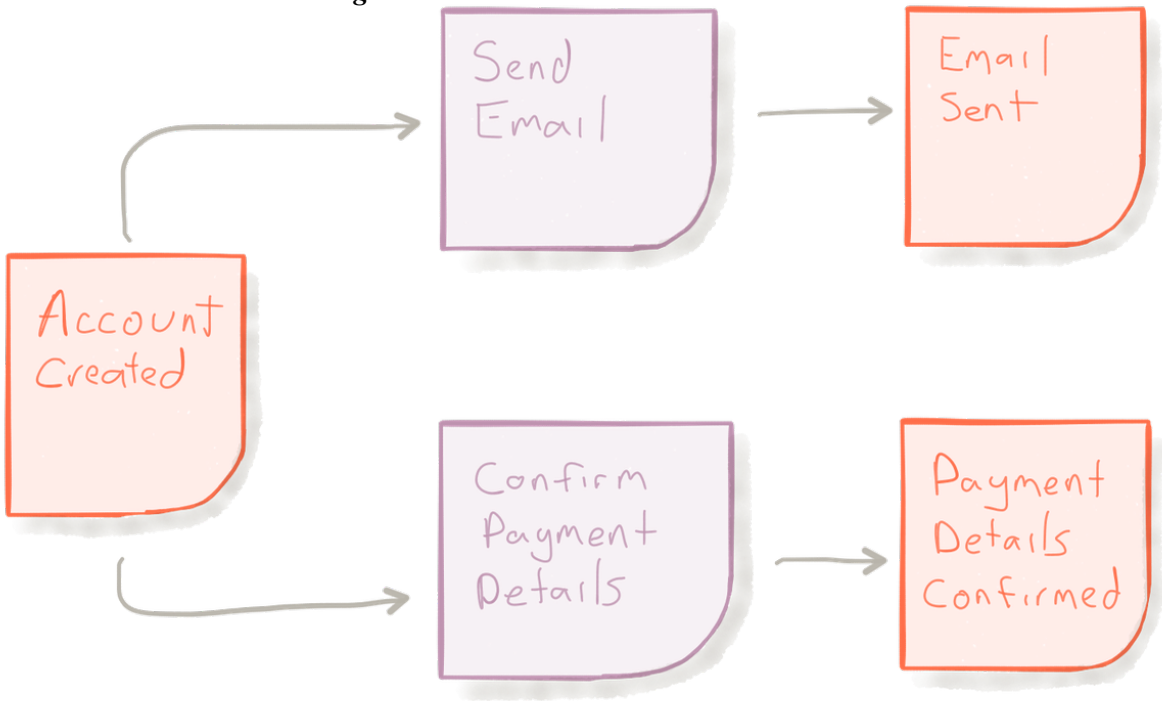
Event Flows: Timeline-based



Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

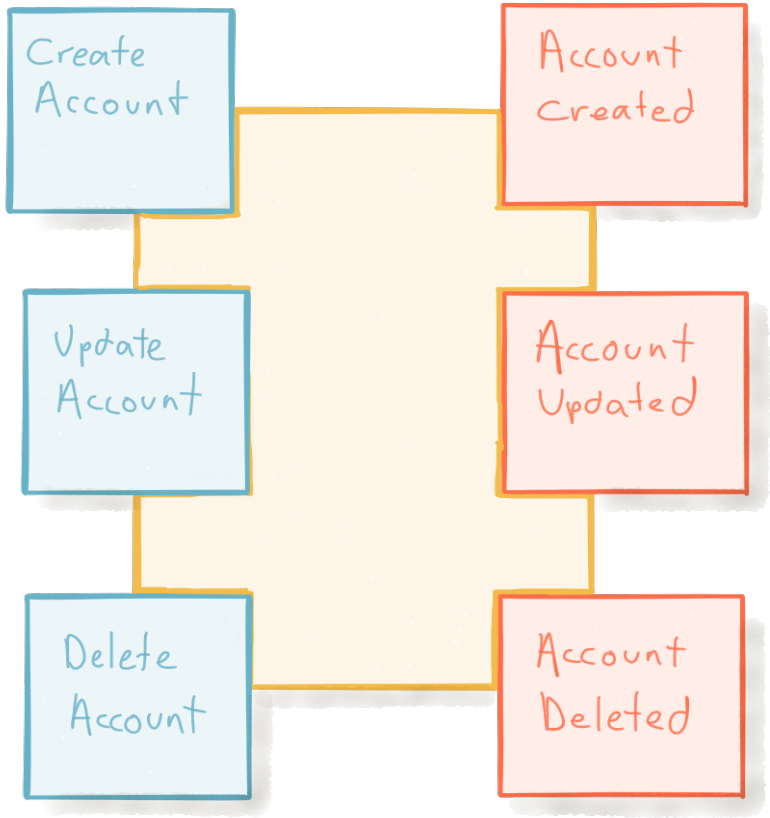
Policy: Reactions triggered by Event

*"Whenever a **new user account is created** we will **send her an acknowledgement by email**".*



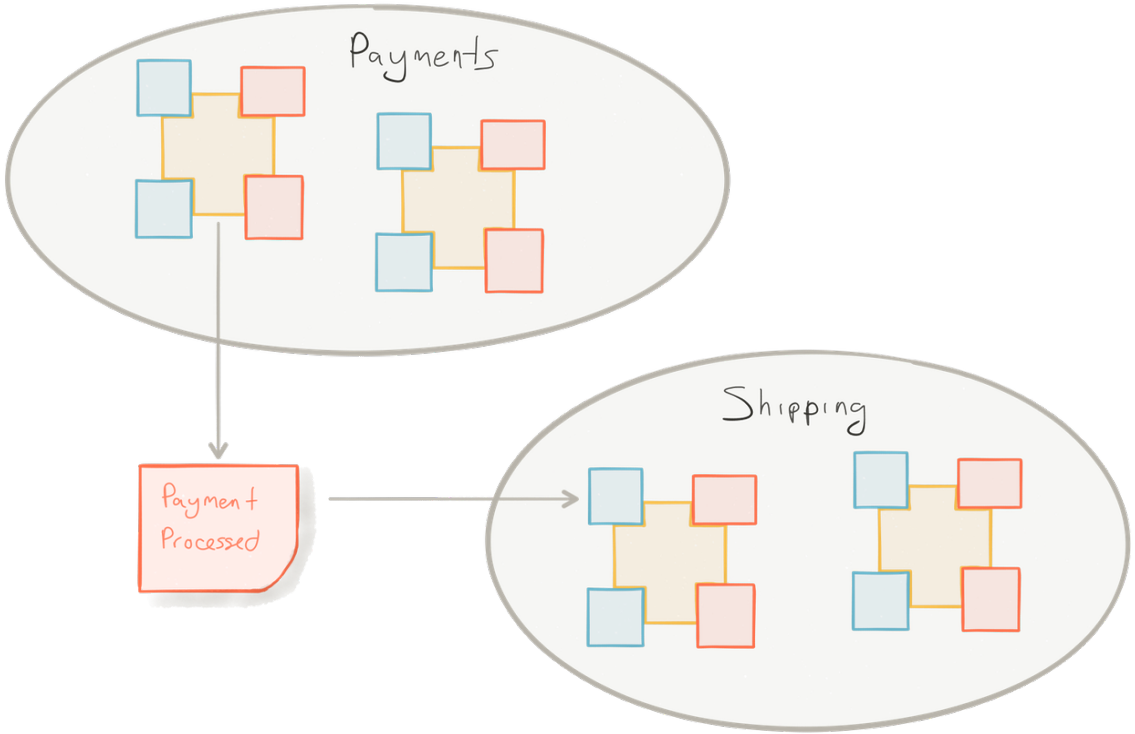
Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

Decomposing Mi-Svc
: Finding Aggregate



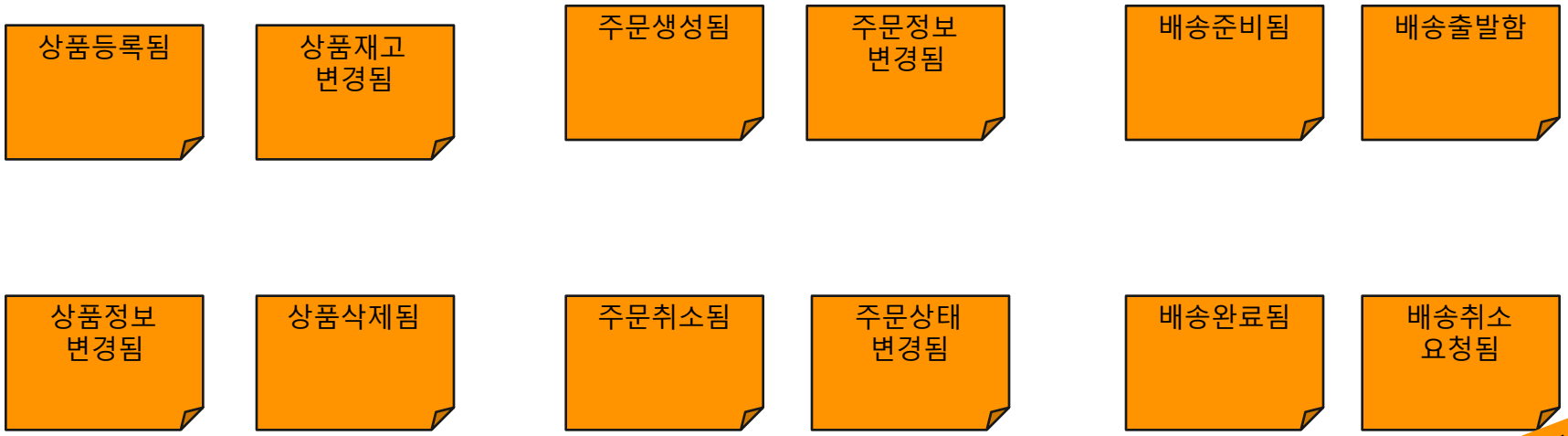
Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

Decomposing Mi-Svc
: Bounded Contexts



Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

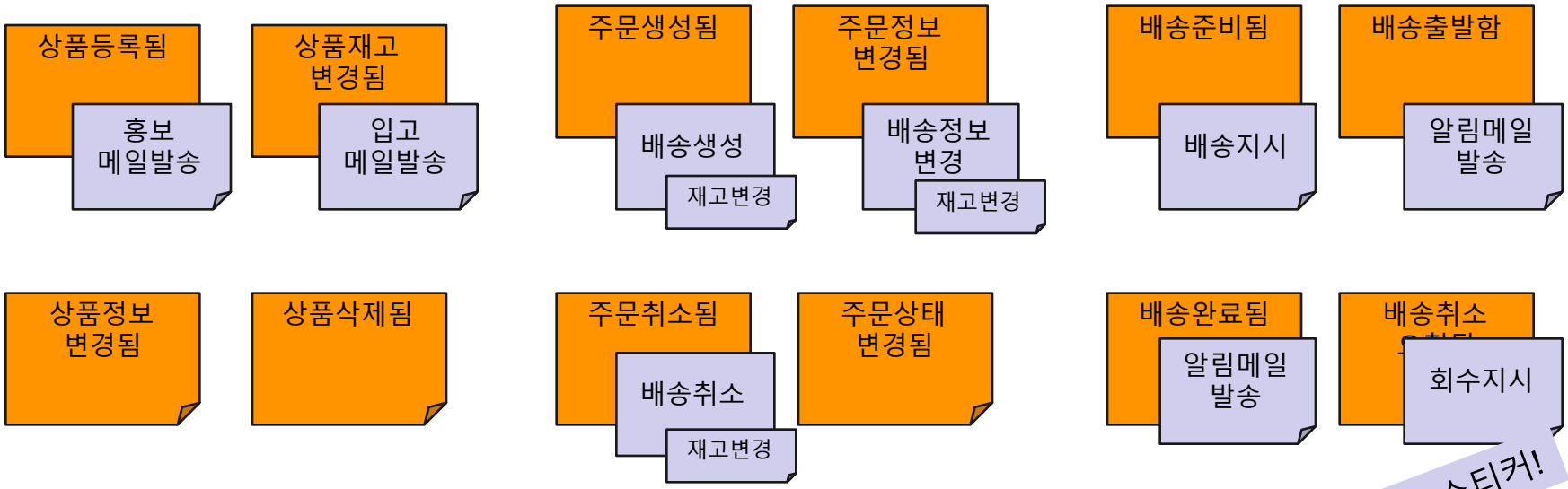
Lab Time – Event 들을 먼저 도출



우리 서비스에는 어떤 비즈니스 이벤트들이 발생하는가?
현업이 사용하는 용어를 그대로 사용 (Ubiquitous Language)
용어의 namespace 를 구지 나누려는 노력을 하지 않음

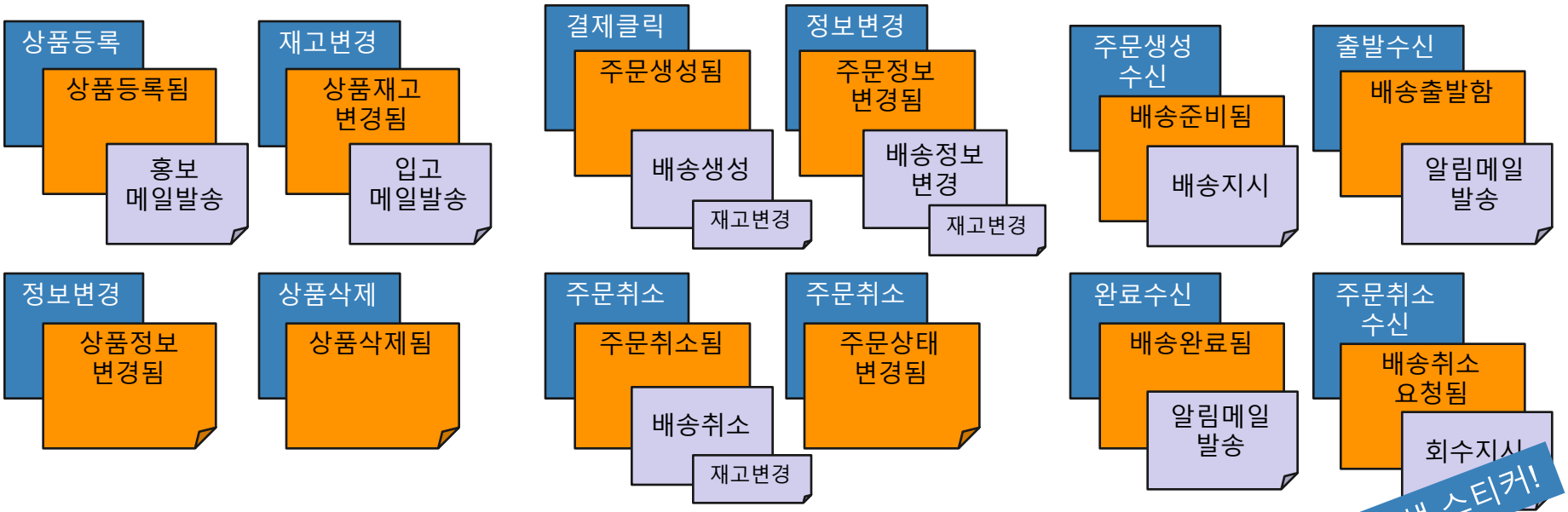
이벤트: 오랜지 스티커!

Lab Time – Policy 도출



어떤 이벤트에 이어서 곧바로 항상 발생해야하는 업무규칙
구현상에서는 이벤트의 Publish에 따라 벌어지는 이후의 프로세스가 자동으로 트리거 되게 함

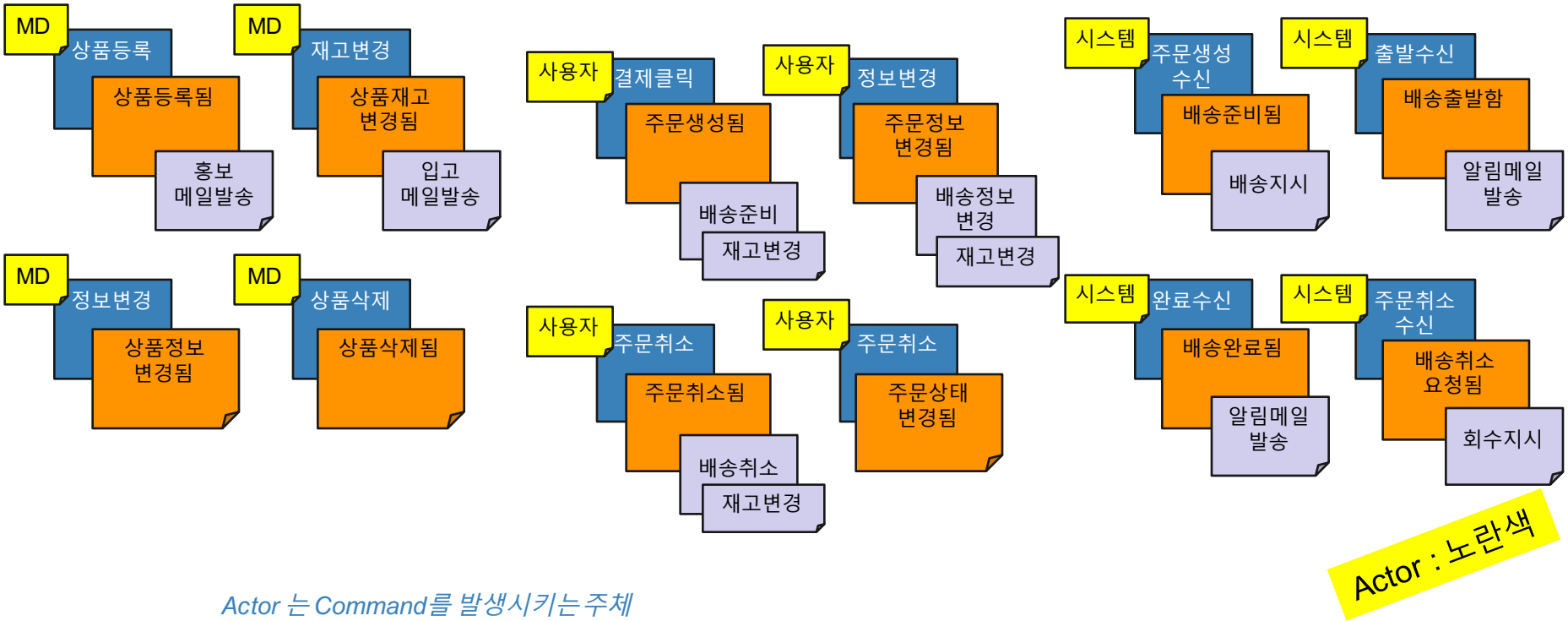
Lab Time – Command 도출 (쓰기행위)



Event를 발생시키는 명령은 무엇인가? UI를 통해? or 시간도래? or 다른 이벤트에 의해?

Command 는 어떠한 상태의 변화를 일으키는 서비스를 말함.
** Command 라는 용어는 CQRS 에서 유래했으며, 쓰기서비스인 Command 와 읽기행위인 Query는 구분됨.*

Lab Time – Command Actor 식별

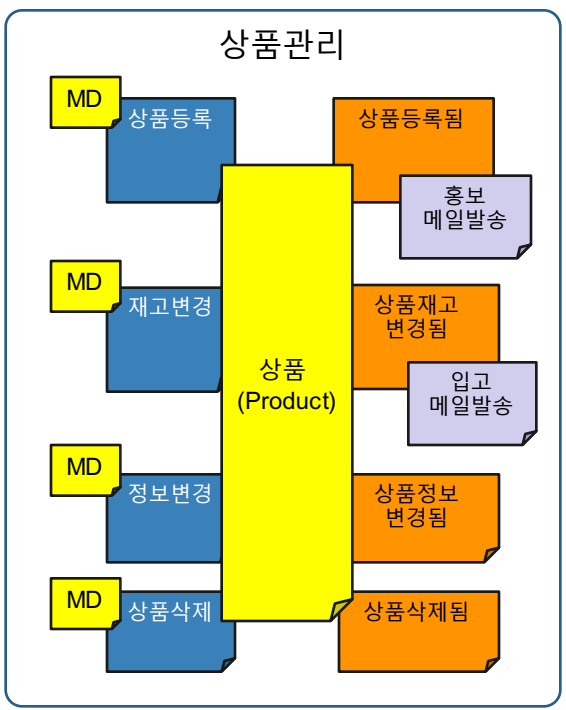


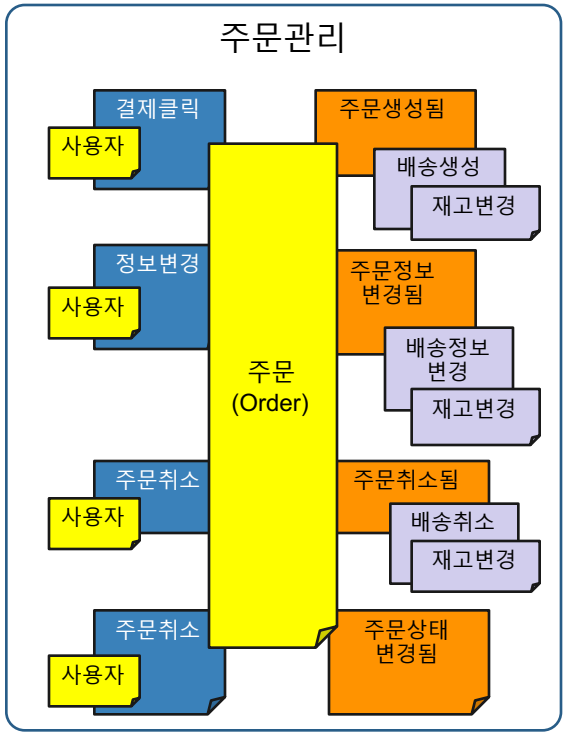
Actor 는 Command를 발생시키는 주체
담당자 또는 시스템 (외부 (Extemal) 또는 내부) 이될 수 있음

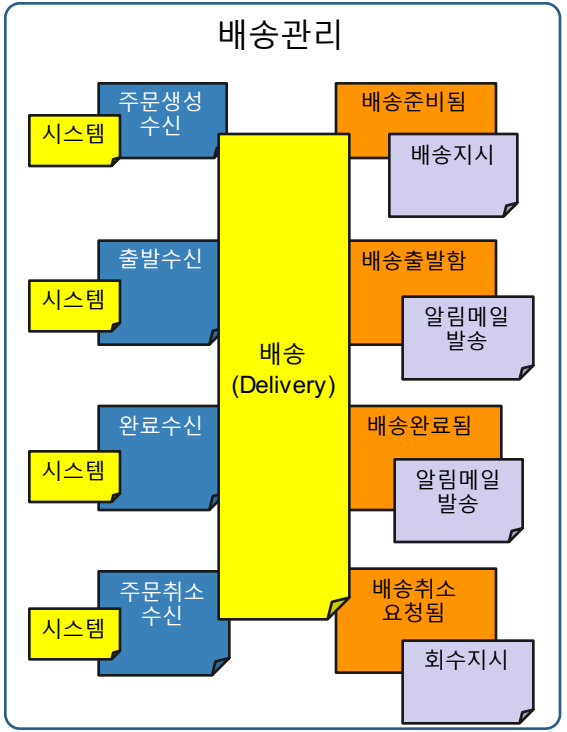
Lab Time – Aggregate 도출 (데이터모델)

도메인 이벤트가 발생하는데는, 어떠한 도메인 객체의 변화가 발생했기 때문이다.
하나의 ACID 한 트랜잭션에 묶여 변화되어야 할 객체의 묶음을 도출하고, 그것들을 커맨드, 이벤트와 함께 묶는다.

Lab Time – Bounded Context 도출







개념적:

- *Bounded Context 는 동일한 문맥으로 효율적으로 업무 용어(도메인 클래스)를 사용할 수 있는 객체범위를 뜻한다. 하나의 BC 는 하나이상의 어그리게잇을 원소로 구성될 수 있다. BC를 Microservice 구성단위로 정하게 되면 이를 담당한 팀 내의 커뮤니케이션이 효율화된다.*

구현관점:

- *어그리게잇은 ACID 트랜잭션 범위이기도 하기때문에 이를 더 쪼개서 BC 를 구성할 수는 없다. BC 내의 어그리게잇들에 대한 보존(Persistence)은 아마도 그 목적에 맞는 최적화된 데이터모델을 독립적으로 설계하고 구현할 수 있다.*

12번가 Shopping Mall Bounded Context

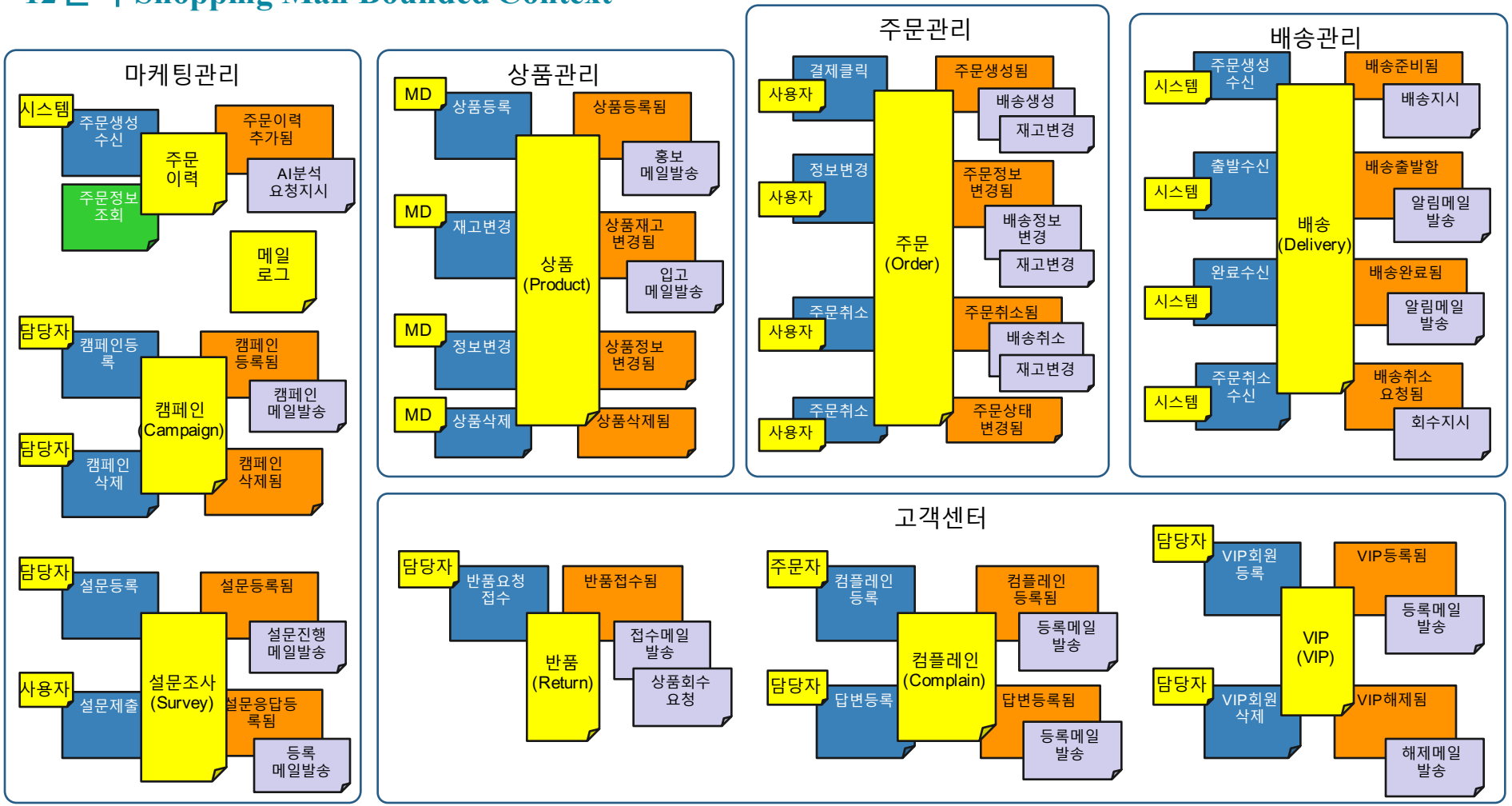


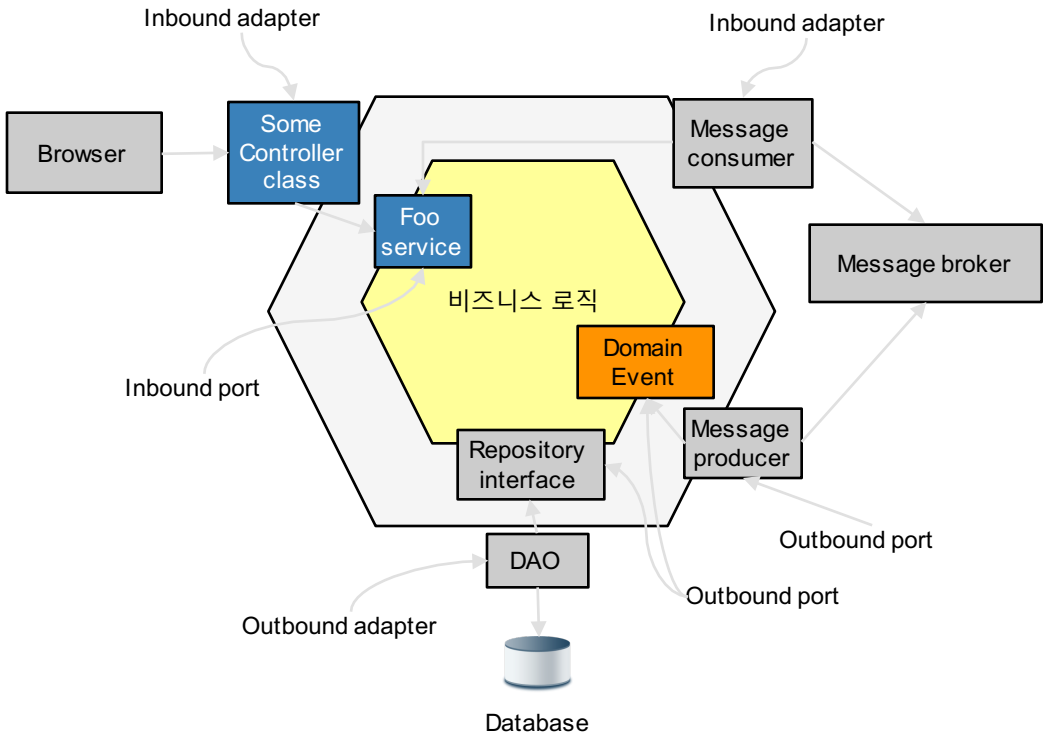
Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

Microservice Implementation Pattern (1)

- Hexagonal Architecture
  - Create your service to be independent of either UI or database and to provide adapters for different input/output sources such as GUI, DB, test harness, RESTful resource, etc.
  - Implement the publish-and-subscribe messaging pattern: As events arrive at a port, an adapter (a.k.a. service agent) converts it into a procedure call or message and passes it to the application. When the application has something to publish, it does it through a port to an adapter, which creates the appropriate signals needed by the receiver. (https://alistair.cockburn.us/Hexagonal+architecture)



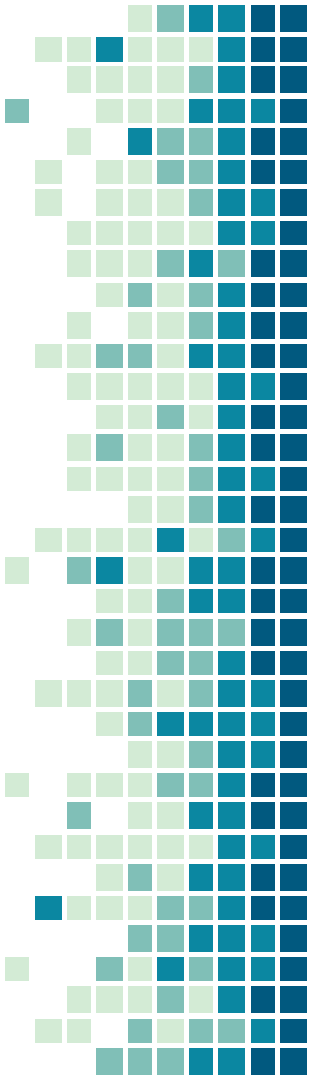
Microservice Implementation Pattern (2)

- HATEOAS (Hypertext As The Engine Of Application State)
  - HATEOAS is deemed the highest maturity level of REST. (https://martinfowler.com/articles/richardsonMaturityModel.html)
  - In the HATEOAS architecture, a client enters a REST application through a specific URL, and all future actions the client may take are discovered within resource representations returned from the server.
  - This self-contained discoverability can be a drawback for most service consumers who prefer API documentation.

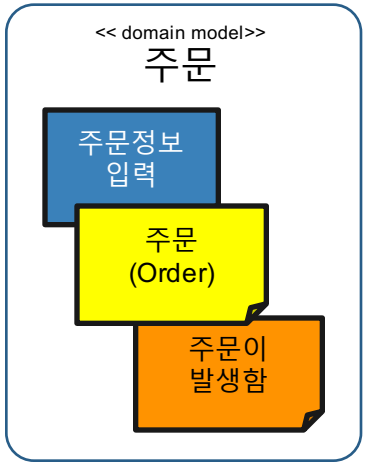
```
GET .../followers/ids.json?cursor=-1&screen_name=josdirksen

{
 "previous_cursor": 0,
 "id": {
   "name": "John Smit",
   "id": "12345678"
   "links" : [
   { "type: "application/vnd.twitter.com.user",
     "rel": "User info",
     "href": "https://.../user/12345678"},
   { "type": "application/vnd.twitter.com.user.follow",
     "rel": "Follow user",
     "href": "https://.../friendship/12345678"}
   ] // and add other options: tweet to, send direct message,
     // block, report for spam, add or remove from list
  }  // This is how you create a self-describing API.
}
```

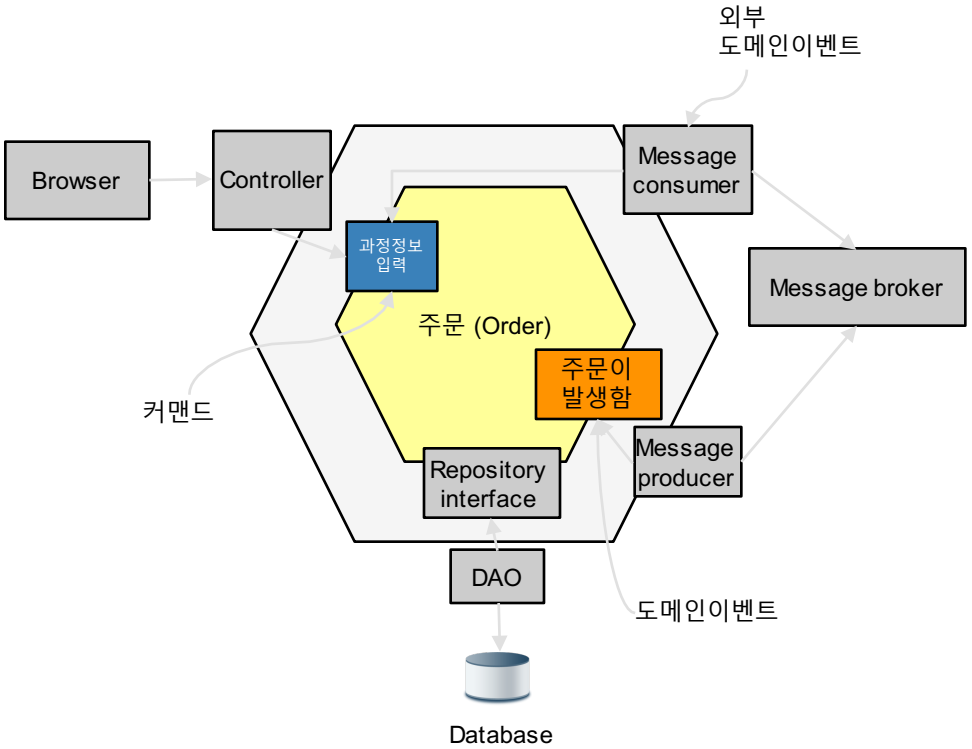
Service Implementation

- You have Powerful Tool:
  Domain-Driven Design and Spring Boot / Spring Data REST
- Domain Classes : Entity or Value Object
- Resources can be bound to Repositories → Full HATEOAS service can be generated!
- Services can be implemented with Resource model firstly
- Low-level JAX-RS can be used if not applicable above

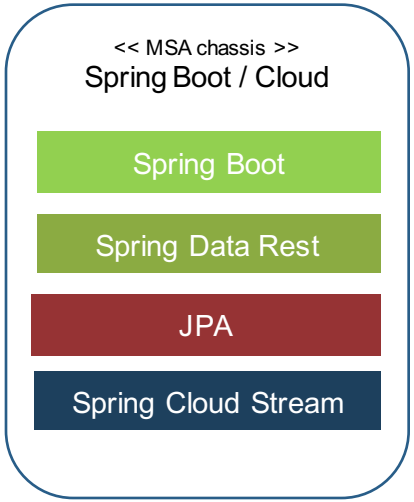


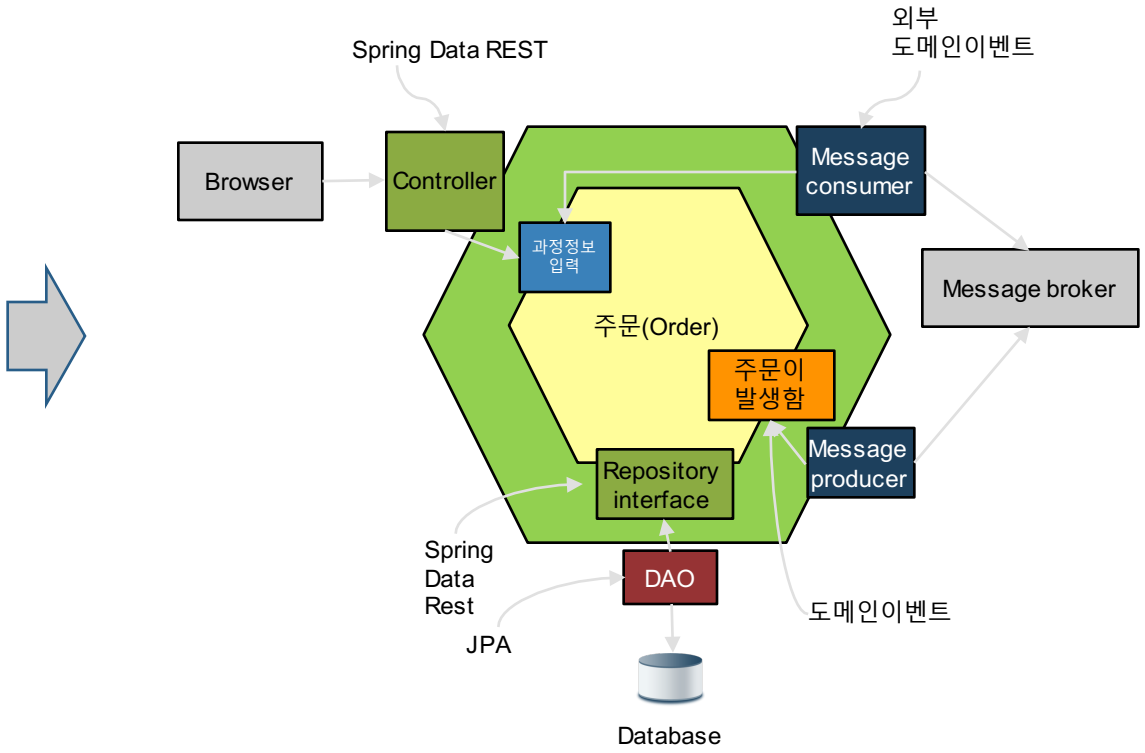
Step 1: Applying Hexagonal Architecture



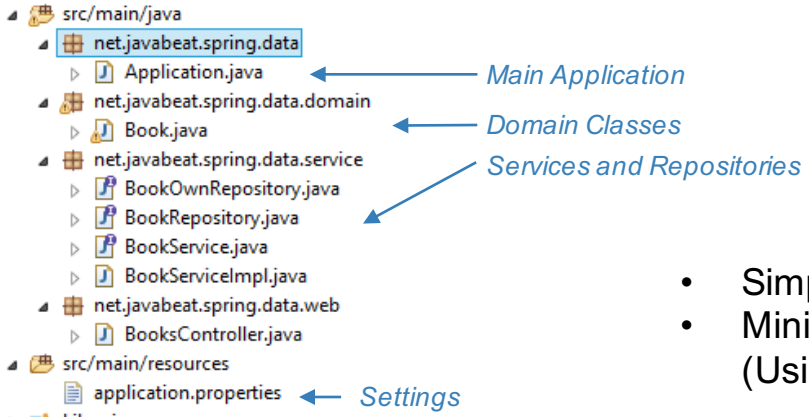


Step 2: Applying MSA Chassis





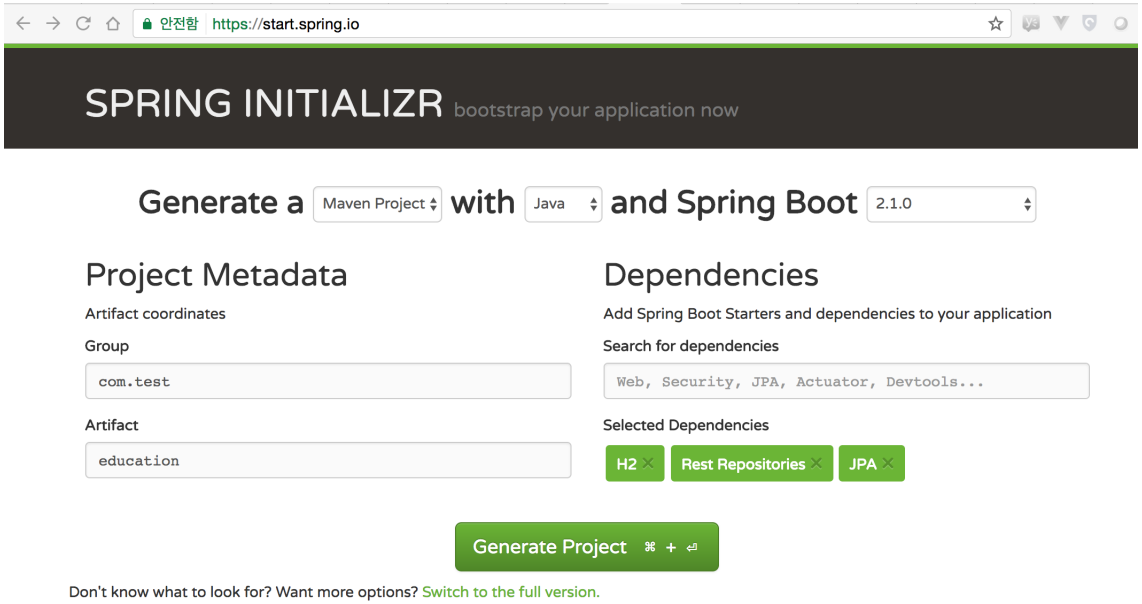
Spring Boot: A MSA Chassis



- Simple Java (POJO)
- Minimal Understanding Of Frameworks and Platforms (Using Annotations)

- No Server-side GUI rendering
  (Only Exposes REST Service)
- No WAS deployment required
  (Code is server; Tomcat embedded)
- No XML-based Configuration

Lab: Create a Spring boot application



Go to start.spring.io
Set metadata:
- Group: com.example
- Artifact: 12st-mall
- Dependencies:
  - H2
  - Rest Repositories
  - JPA

Press “Generate Project”
Extract the downloaded zip file
Build the project:
./mvnw spring-boot:run

Port 충돌시:
./mvnw spring-boot:run -Dserver.port=8081

Running Spring Boot Application

```
$ mvn spring-boot:run      # 혹은 ./mvnw spring-boot:run

[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building ….
[INFO] ------------------------------------------------------------------------
[INFO] Downloading:
https://repo.maven.apache.org/maven2/org/springframework/security/spring-security-
core/maven-metadata.xml
[INFO] Downloading:
https://oss.sonatype.org/content/repositories/snapshots/org/springframework/security/spring
-security-core/maven-metadata.xml
[INFO] Downloading: https://repo.spring.io/libs-release
:
```

Started Application in 11.691 seconds (JVM running for 14.505)

```
Source code:
$ git clone https://github.com/uengine-oss/software-modeling-class-management-monolith.git
```

Test Generated Services

```
$ http localhost:8080
HTTP/1.1 200
Content-Type: application/hal+json;charset=UTF-8
Date: Wed, 05 Dec 2018 04:20:52 GMT
Transfer-Encoding: chunked

{
    "_links": {
        "profile": {
            "href": "http://localhost:8080/profile"
        }
    }
}
```

Aggregate → Entity Class 작성


```
@Entity
public class Product {

    @Id
    @GeneratedValue
    private Long id;

    String name;
    int price;
    int stock;


    @OneToMany( cascade =
CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "order")
    List<Orde> orders;
}
```


```
@Entity
public class Order {

    @Id
    @GeneratedValue
    private Long id;
    private Long productId;
    private String productName;
    private int quantity;
    private int price;
    private String customerName;
    private String customerAddr;

     @ManyToOne
@JoinColumn(name="productId")
    Product product;
}
```


```
@Entity
public class Delivery {

    @Id @GeneratedValue
    private Long deliveryId;
    private Long orderId;
    private String customerName;
    private String deliveryAddress;
    private String deliveryState;

    @OneToOne
    Order order;
}
```

Commands → Service Object* 와 API의 생성

Repository Pattern 으로 생성된 것 사용할 수 있는 경우

주문과 배송 API 는 Order Repository 에서 생성된 CRUD actions 들 중 POST Service 를 그대로 사용할 수 있으므로 별도 구현할 필요 없다. 마찬가지 주문서 수정, 삭제도 각각 PATCH, DELETE 로 API 생성

전체의 약 7~80% 커버


```
public interface OrderRepository extends PagingAndSortingRepository<Course, Long> {
    List<Course> findByOrderId(@Param("orderId") Long OrderId);
}
```

Repository 에서 생성된 API 를 사용 못하는 경우, 별도 Spring MVC Service 로 생성

배송일정 등을 위한 백엔드 API 는 Order Repository 에 생성된 PUT-POST-PATCH-DELETE 중 적합한 것이 없으므로 별도 추가 action URI를 만드는 것이 적합

나머지 2~30% 수준


```
@Service
public interface ProductService {
    @RequestMapping(method = RequestMethod.GET, path="/calendar/{deliveryId}/{date}")
    Resources getProduct(@PathVariable("deliveryId") Long deliveryId, @PathVariable("date")
String date);
}
```

Main Class

```
@SpringBootApplication
public class Application
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Test Generated Services

```
$ http localhost:8081
{
    "_links": {
        "products": {
            "href": "http://localhost:8081/products{?page,size,sort}",
            "templated": true
        }
}
$ http localhost:8082
{"_links": {
        "orders": {
            "href": "http://localhost:8082/orders{?page,size,sort}",
            "templated": true
        }
}
$ http localhost:8083
{"_links": {
        "deliverys": {
            "href": "http://localhost:8083/deliverys{?page,size,sort}",
            "templated": true
        } }
}
```

Create new Order

```
$ http localhost:8082/orders productId=1 productName="TV" quantity=1
customerName="홍길동" customerAddr="서울시"
```


```
{
    "_links": {
    "order": {
        "href": "http://localhost:8082/orders/1"
    },
    "self": {
        "href": "http://localhost:8082/orders/1"
    }
},
    "customerAddr": "서울시",
        "customerName": "홍길동",
        "price": 0,
        "productId": 1,
        "productName": "TV",
        "quantity": 1
}
```


Create a delivery of the order

```
$ http POST localhost:8083/deliverys deliveryState =“DeliveryStar
orderId ="http://localhost:8082/orders/1"
```

```
$ http "http://localhost:8083/deliverys/1/order"
```

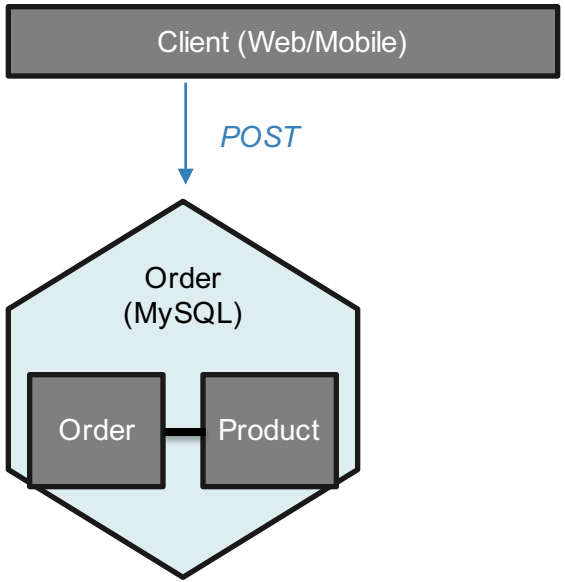

```
{
  "_links": {
    "self": {
      "href": "http://localhost:8082/orders/1"
    }
  },
  "customerAddr": "서울시",
  "customerName": "홍길동",
  "price": 0,
  "productId": 1,
  "productName": "TV",
  "quantity": 1
}
```


URI of the SOA delivery

Delivery is aggregation root

Monolothic 에서의 분산 트랜잭션 – 이슈없음

서비스구성 (모놀로씩)



조회화면

Consistent

In-consistent

주문량: 0 재고량: 10

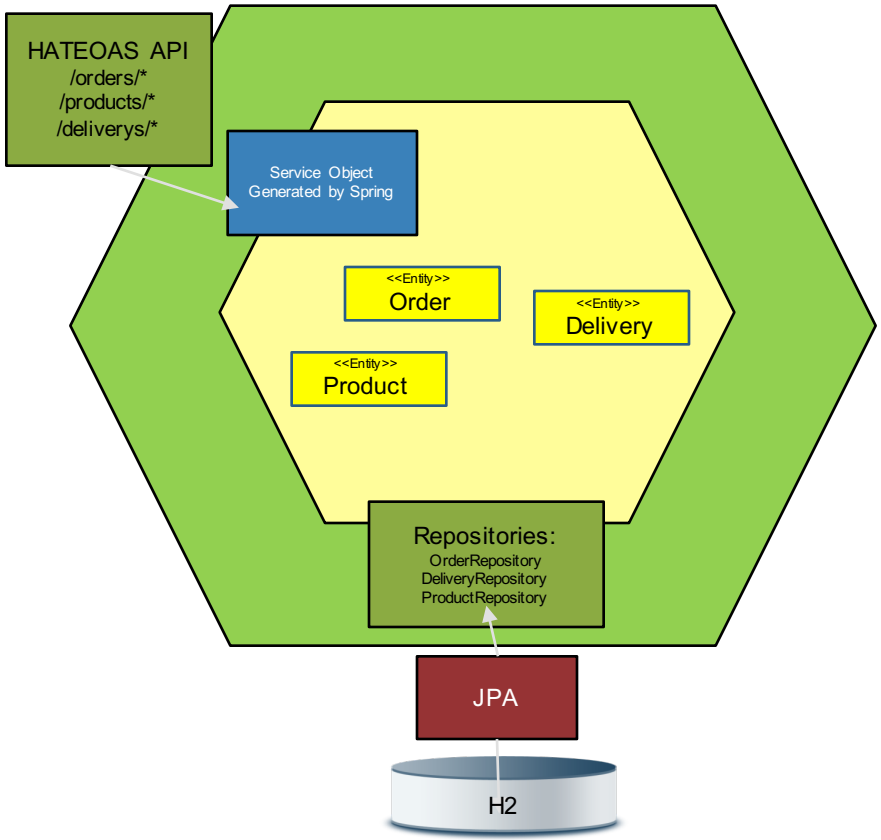
주문량: 1 재고량: 9

주문량: 2 재고량: 8

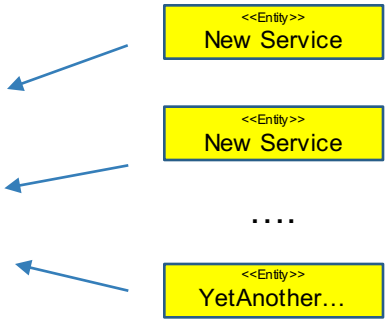
항상 일치함

But,
Tight-Coupling
Shared Database
Single DB
Architecture....

Nice but Too big (monolith)



Plan to add…



Requirements / Check

Category	Requirement	Solution
Maintainability & Scalability	24*365 Zero Down time	Self-Healing, Horizontally Scale
	Minimal Dev-Ops, Dev-Dev team side effects in deploy time (more than 100 deploys per day)	Loose coupled design (HATEOAS API, Service Decomposition), PubSub
		Auto Provisioning (by docker containerizing)
		DevOps CI/CD
Scalability & Reusability	Multi-channel Support	Responsive Web, Chat Bot
	Dynamic Service Composition	Dynamic Discovery/Composition, BPM
Usability & Performance	Fast browsing, Single Page, DDoS protection, Inter-microservice integration	Client-side UI rendering, API Gateway, Circuit Breaker, Event-driven Arch.
Security	For all services including 3rd-party ACL	Access Token, IAM

HATEOAS API

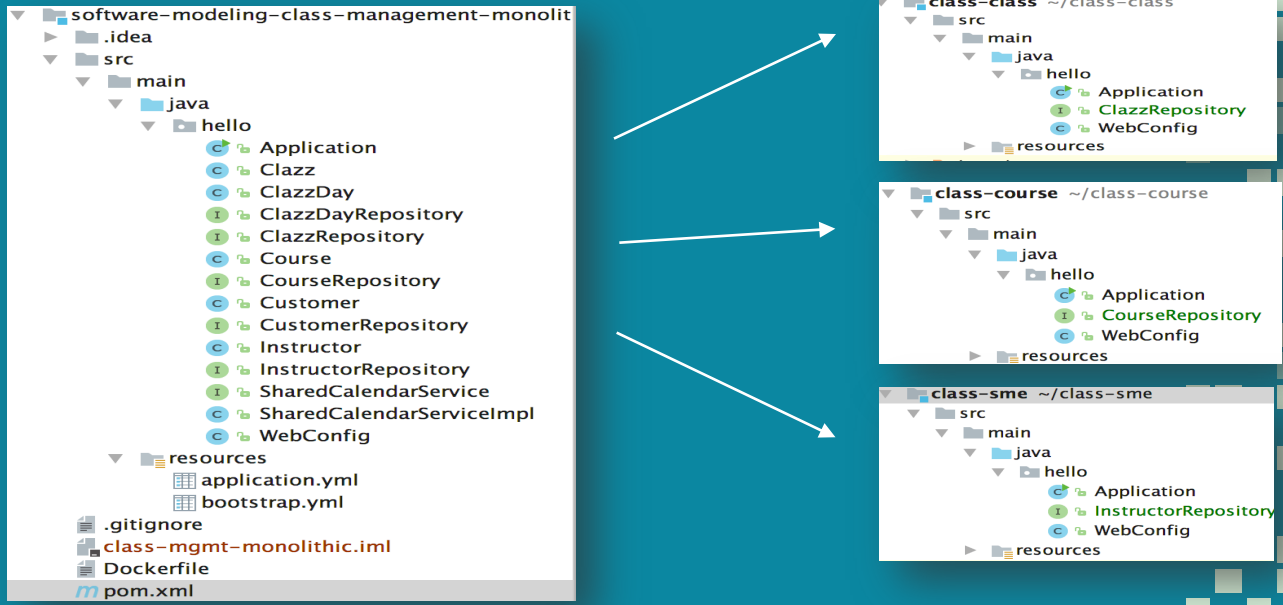
Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

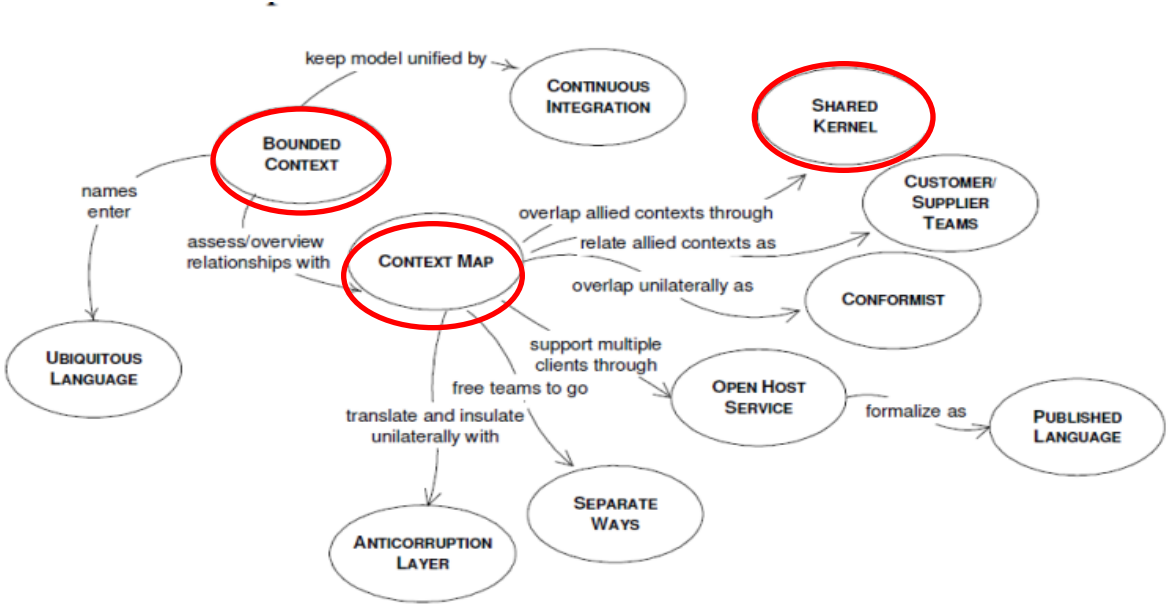
1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

“From Monolith to Microservices”

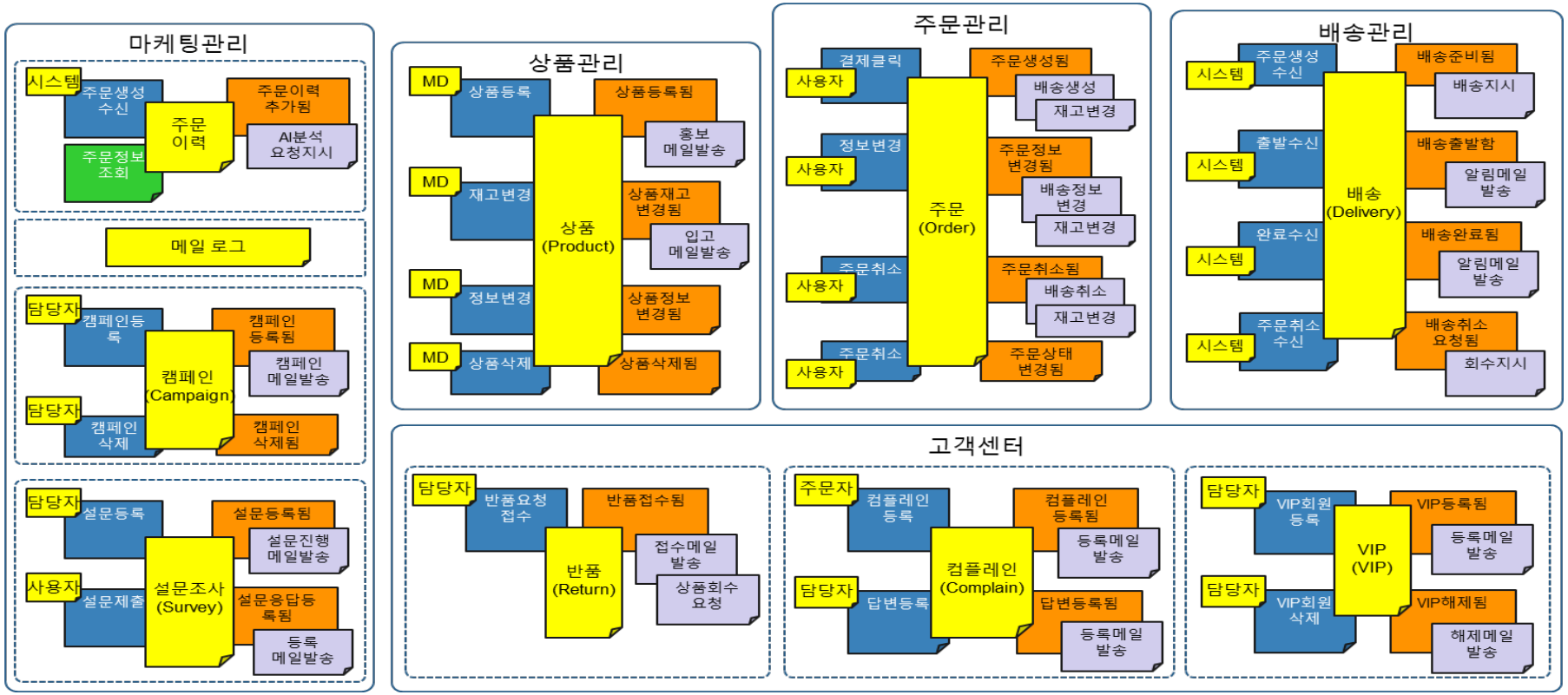
https://github.com/jinyoung/software-modeling-class-management-monolith
→
https://github.com/jinyoung/software-modeling-class-management-msa



DDD Pattern Applied

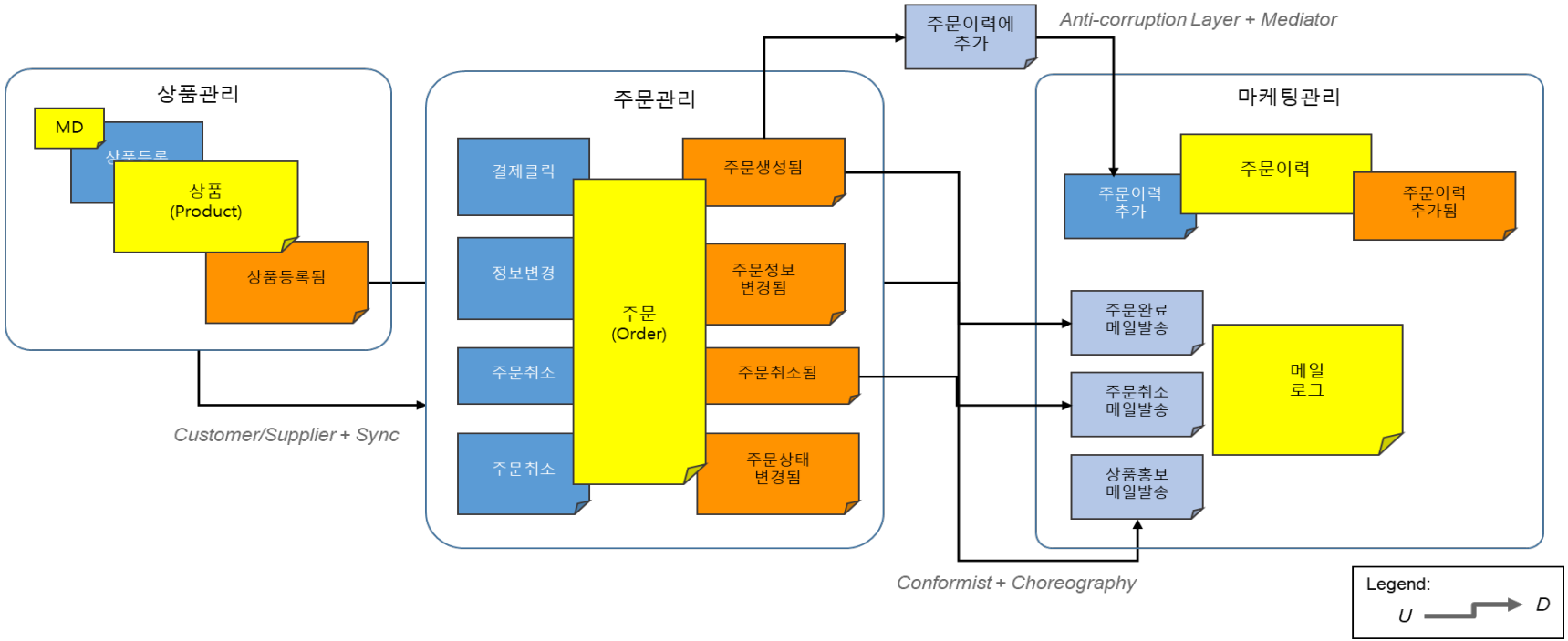


Lab Time – 마이크로서비스 도출



- *마케팅 관리와 고객센터 팀 내에는 서로 간의 간섭이 전혀 없는 복수 개의 어그리게잇으로 구성되어 있어 더 작은 마이크로서비스로 충분히 도출 가능함.*

Lab Time – Context Map 도출



개념적:

- *BC 간의 정보참조의 릴레이션, 혹은 이벤트가 발생한 이후의 동반된 행위의 호출의 관계를 선으로 표시함. Upstream → Downstream 으로 정보가 내려가게 됨.*

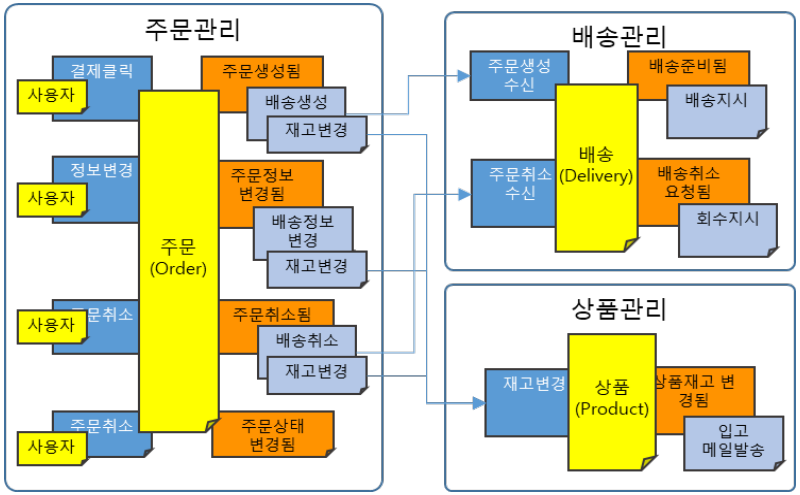
구현관점:

- *자치적으로 구현설계가 이루어질 수 있는 BC 간에 커뮤니케이션을 위해서는 데이터의 적절한 변환이나 표준화, Pub/Sub 등의 커뮤니케이션 방법 및 데이터 전환에 대한 이슈가 발생하게 됨. 해당 이슈를 적절한 패턴을 선택하여 실제 연동을 위한 Adapter 를 개발해야 함.*

Lab Time – Topology Consideration

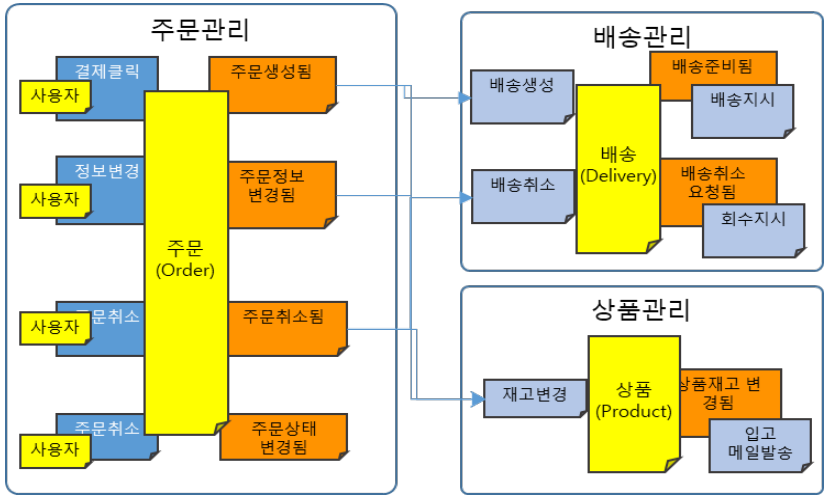
라이락 스티커 (Policy) 를 어디에 둘 것인가?
- Orchestration or Choreography? Or Mediation?

Orchestration



Originator should know how to handle the policy
→ Coupling is High

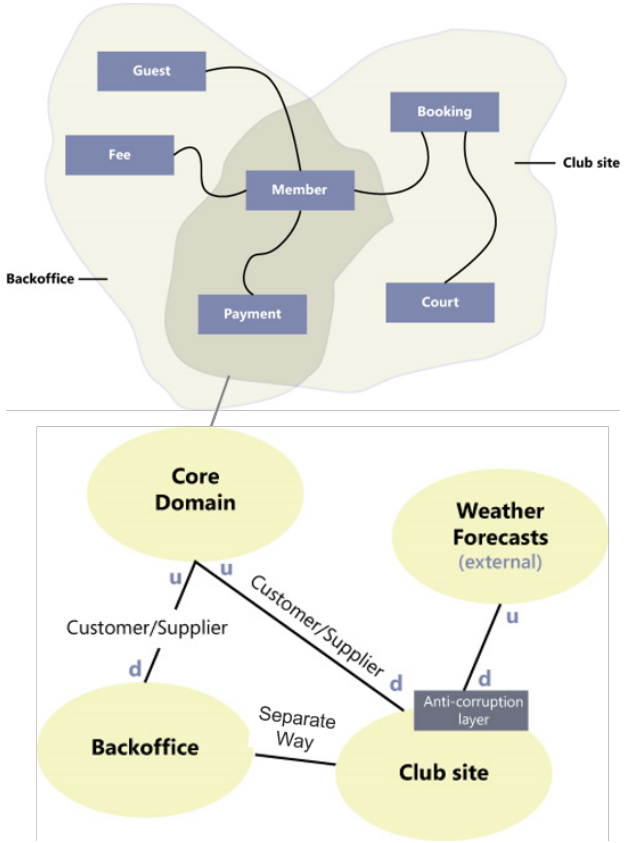
Choreography



More autonomy to handle the policy
→ Low-Coupling
→ Easy to add new policies

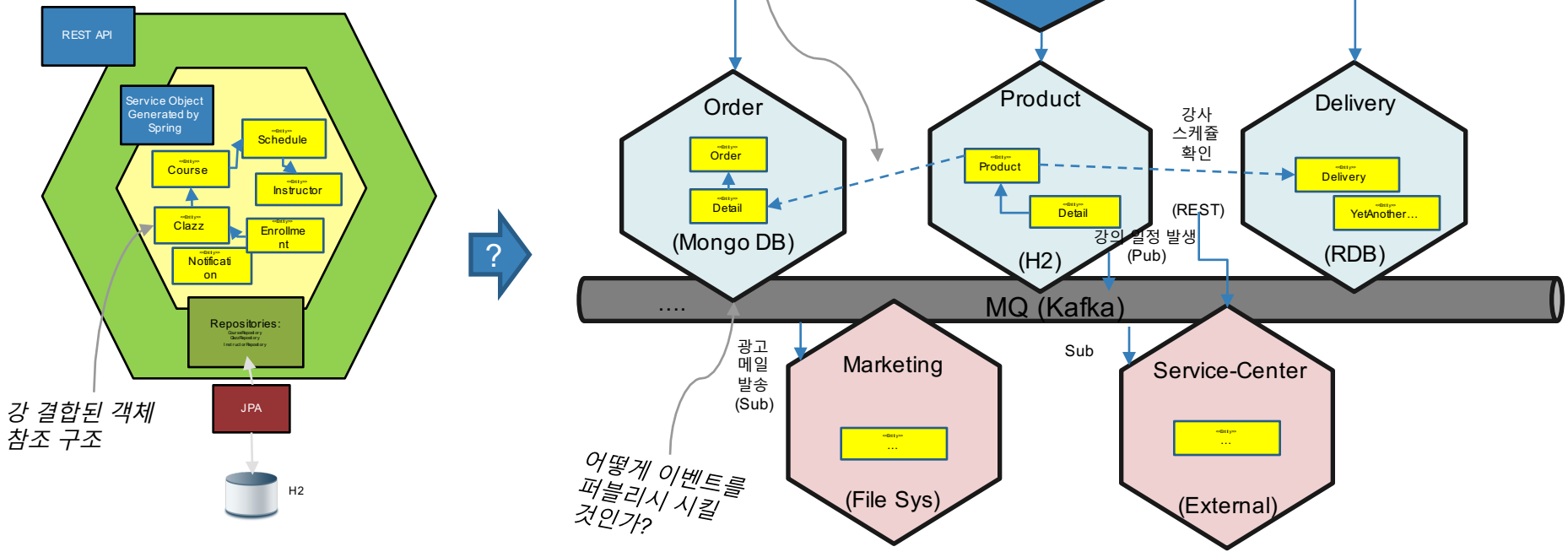
Ref: Relation types between services

- **Shared Kernel**
  - Designate a subset of the domain model that the two teams agree to share.
- **Customer/Supplier**
  - Make the downstream team play the customer role to the upstream team.
- **Conformist**
  - Eliminate the complexity of translation between bounded contexts by slavishly adhering to the model of the upstream team.
- **Separate Ways**
  - Declare a bounded context to have no connection to the others at all. The features can still be organized in middleware or the UI layer.
- **Open Host Service**
  - Open the protocol so that all who need to integrate with you can use it. Other teams are forced to learn the particular dialect used by the host team. In some situations, using a well-known published language as the interchange model can reduce coupling and ease understanding.



Eric Evans, Domain-Driven Design, 2003.

Issues in transforming Monolith to Microservices



Solutions – 객체의 분리

- 참조해야 할 도메인 클래스들의 공통화
  → Shared Kernel, Maven Repo 등을 통한 공통 모델의 공유
  : 동일 언어인 경우 유리, 디펜던시 발생
  → 혹은 중복 모델 개발 : Polyglot 인 경우 유리

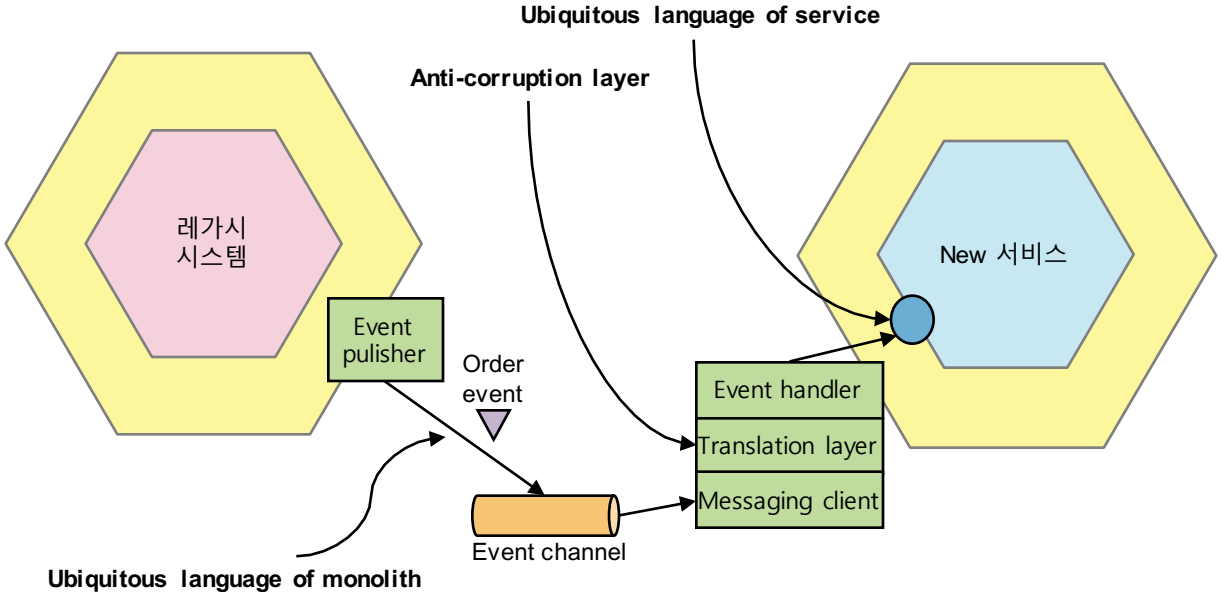
- Entity 분리에 따른 원격 객체 (어그리게잇) 참조
  → Primary Key 를 통해서만 참조
  → HATEOAS link 를 이용

Solutions – API 통합

- API Gateway
  → 진입점의 통일
  → Path-based Routing (기존에 REST 로 된경우 가능)

- Service Registry
  → API Gateway 가 클러스터 내의 인스턴스를 찾아가는 맵

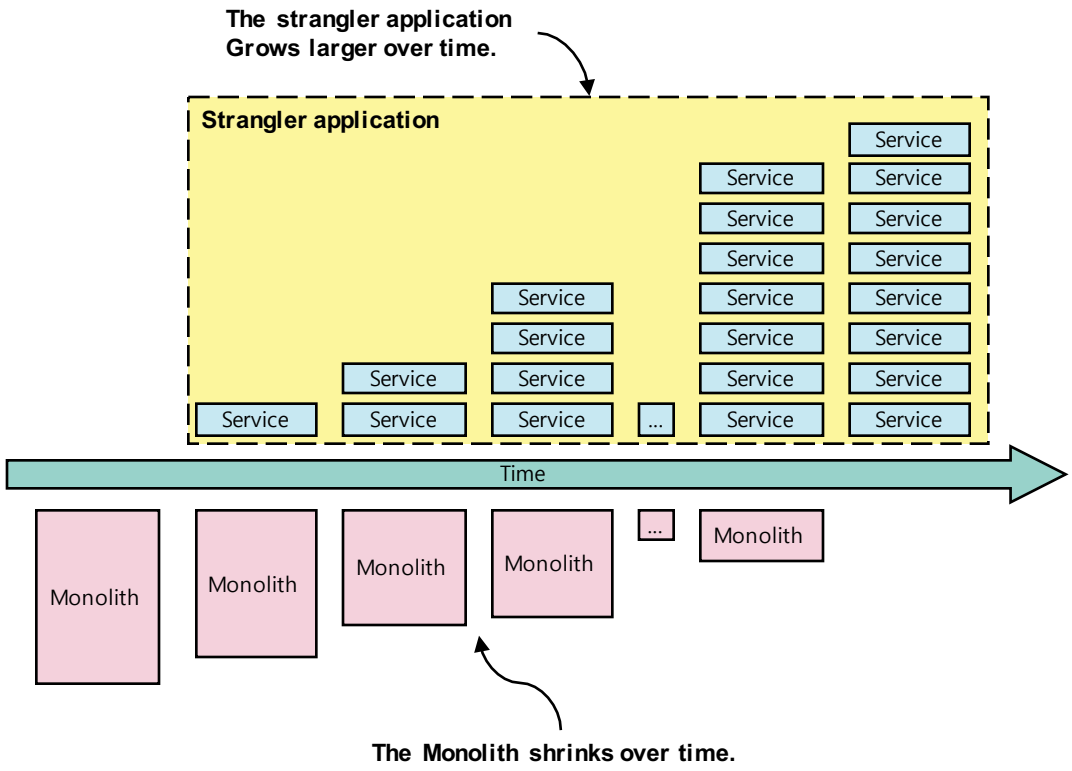
Solutions – 이벤트의 퍼블리시



- **Aggregate 내 코드 주입**
  → 호출 코드 직접 주입, Hexagonal Architecture 의 손상
  → JPA 의 Lifecycle Annotation 사용
- **CDC (Change Data Capturing) 기능 사용**
  → DB 의 Change Log 를 Listening, Event 자동퍼블리시 하는 툴
  → Debezium, Eventuate Tram 등이 존재

Modified from Microservices Patterns, Chris Richardson, Manning, 2018

Legacy Transformation – Strangler Pattern



- Strangler 패턴으로 레가시의 모노리스 서비스가 마이크로서비스로 점진적 대체를 통한 Biz 임팩트 최소화를 통한 구조적 변화
- 기존에서 분리된 서비스 영역이 기존 모노리스와 연동 될 수 있도록 해주는 것이 필요.
- 그 방법은 앞서 동기/ 비동기 방식이 채용될 수 있음
- 동기 – 레거시로 하여금 기존 소스코드 수정 요인이 높음, 따라서 이벤트 기반 비동기 연동 (CQRS) 추천

Modified from Microservices Patterns, Chris Richardson, Manning, 2018

마이크로 서비스간의 서열과 역학관계

무엇을 우선적으로 챙길 것인가?

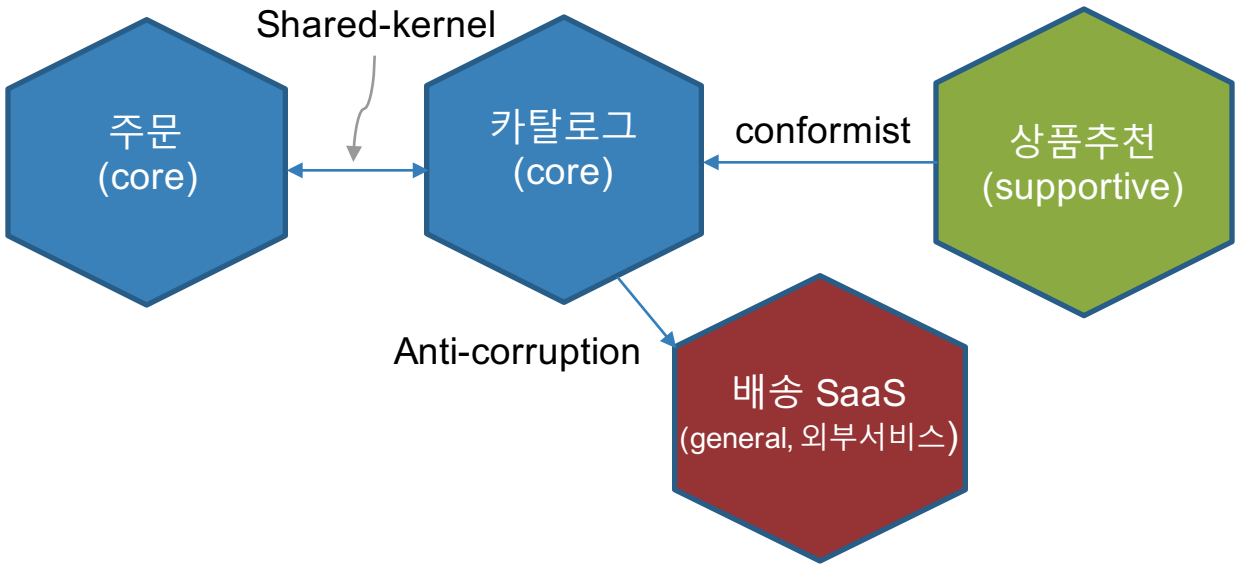
1순위: Core Domain
버릴 수 없는. 이 기능이 제공되지 않으면 회사가 망하는.
예) 쇼핑몰 시스템에서 주문, 카탈로그 서비스 등

2순위: Supportive Domain
기업의 핵심 경쟁력이 아닌, 직접 운영해도 좋지만 상황에 따라 아웃소싱 가능한. 시스템 리소스가 모자라면 외부서비스를 빌려쓰는 것을 고려할만한
예) 재고관리, 배송, 회원관리 서비스 등

3순위: General Domain
SaaS 등을 사용하는게 더 저렴한, 기업 경쟁력과는 완전 무관한
예) 결재, 빌링 서비스 등

마이크로 서비스간의 서열과 역학관계

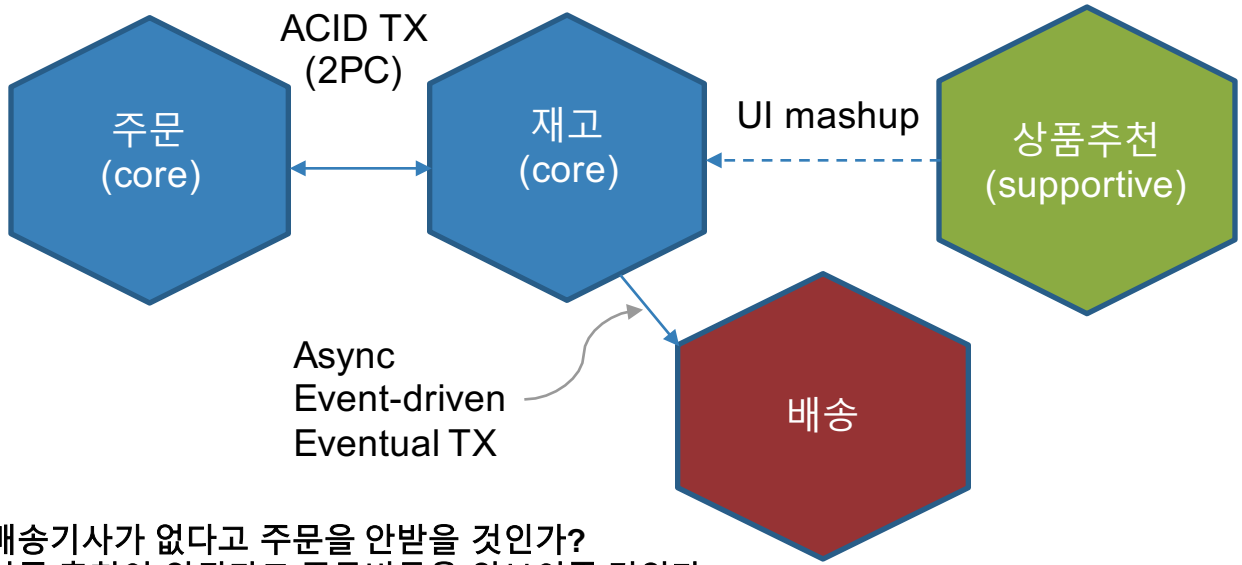
어느 마이크로 서비스의 인터페이스를 더 중요하게 관리할 것인가?



Core Domain 간 (높은 서열끼리) 에는 **Shared-kernel** 도 허용가능하다. 하지만, 중요도가 낮은 서비스를 위해 높은서열의 서비스가 인터페이스를 맞추는 경우는 없을 것이다.

마이크로 서비스간의 서열과 역학관계

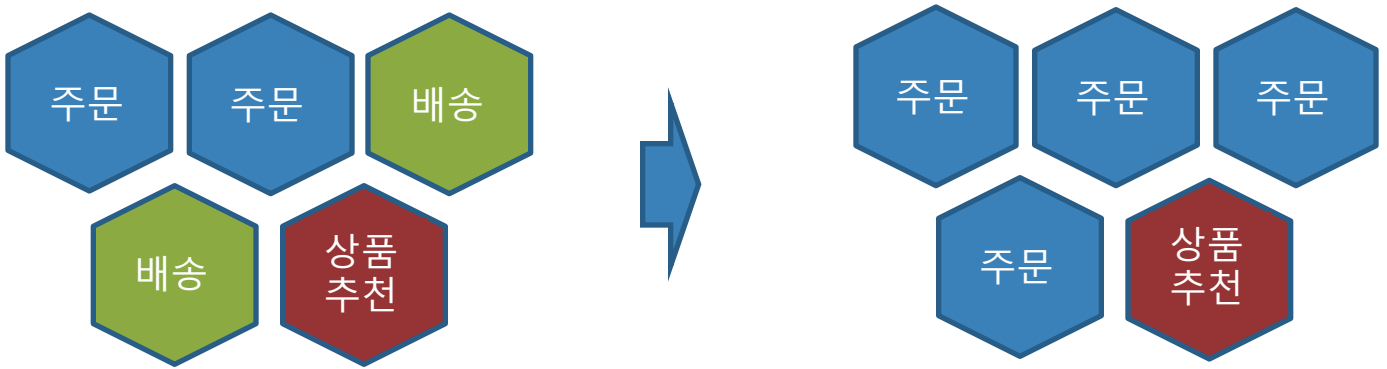
마이크로서비스간 트랜잭션의 묶음을 어떻게 할 것인가?



배송기사가 없다고 주문을 안받을 것인가?
상품 추천이 안된다고 주문버튼을 안보여줄 것인가?
Core Domain 간에는 강결합이 요구되는 경우가 생길 수 있다. 하지만 우선순위가 떨어지는 비즈니스 기능을 위해 강한 트랜잭션을 연결할 이유는 하등에 없다.

마이크로 서비스간의 서열과 역학관계

손님이 많이 와서 집이 좁아졌다.. 무엇을 우선적으로 줄일 것인가?



배송서비스는 야간에 올라와서 후에 처리되어도 된다.
주문이 몰려드는 시간에 주문을 못받으면 배송은 의미가 없다

Requirements / Check

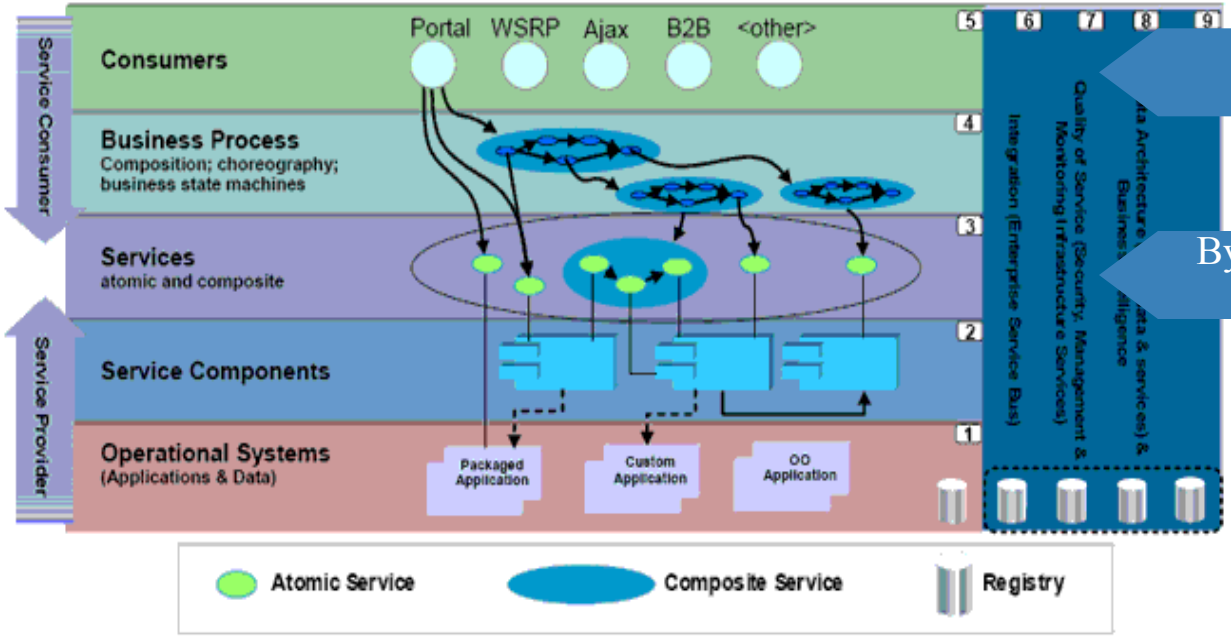
Category	Requirement	Solution
Maintainability & Scalability	24*365 Zero Down time	Self-Healing, Horizontally Scale
	Minimal Dev-Ops, Dev-Dev team side effects in deploy time (more than 100 deploys per day)	~~Loose coupled design (HATEOAS API, Service Decomposition)~~ Auto Provisioning (by docker containerizing)
		DevOps CI/CD
Scalability & Reusability	Multi-channel Support	Responsive Web, Chat Bot
	Dynamic Service Composition	Dynamic Discovery/Composition, BPM
Usability & Performance	Fast browsing, Single Page, DDoS protection, Inter-microservice integration	Client-side UI rendering, API Gateway, Circuit Breaker, Event-driven Arch.
Security	For all services including 3rd-party ACL	Access Token, IAM

Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

Integration Patterns



By UI

By Request-Response

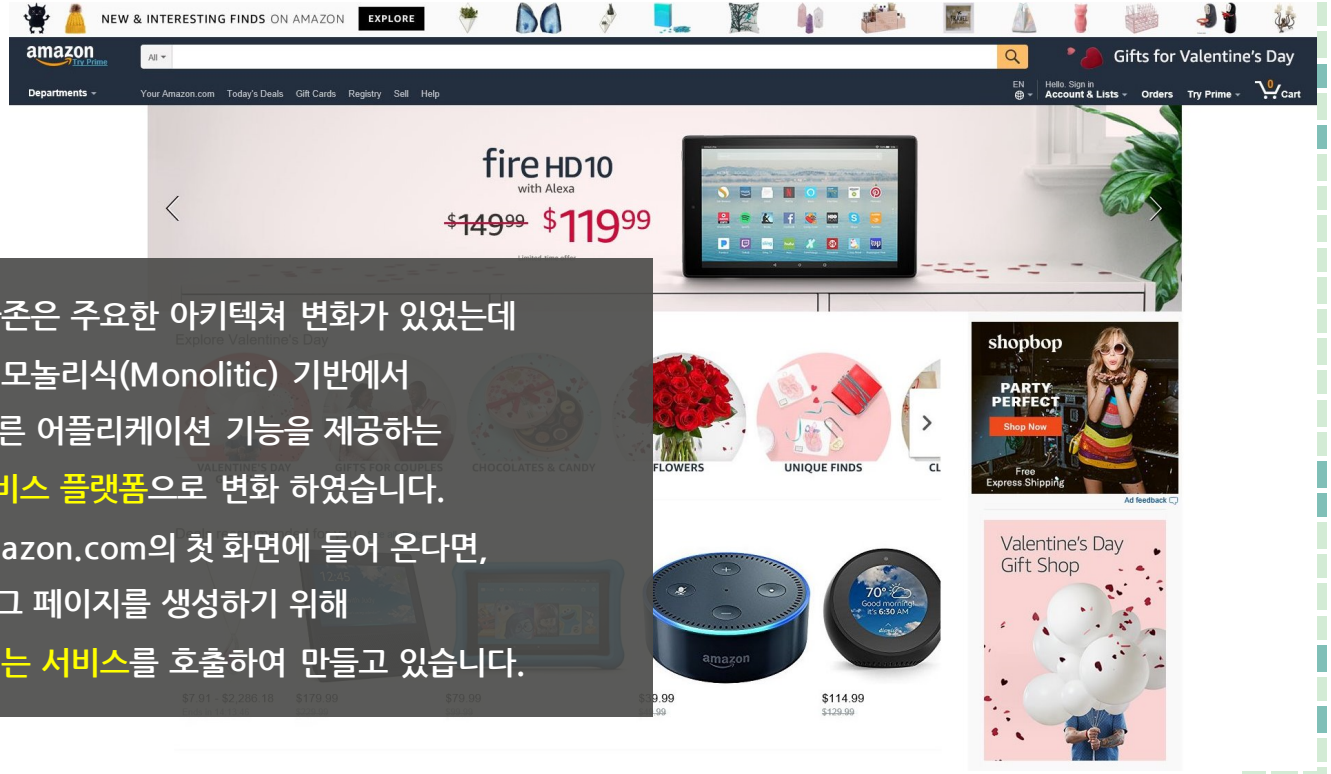
By Event-driven Architecture

Service Composition with User Interface

- Extended Role of Front-end in MSA Architecture: Service Aggregation
- Why MVVM?
- W3C Web Components Standard – Domain HTML Tags
- Implementation: Polymer and VueJS
- Another: ReactJS and Angular2
- Micro-service Mashups with Domain Tags: i.e. IBM bluetags
- Cross-Origin Resource Sharing
- API Gateway (Netflix Zuul)

https://www.youtube.com/watch?v=djQh8XKRzRg
https://github.com/IBM-Cloud/bluetag

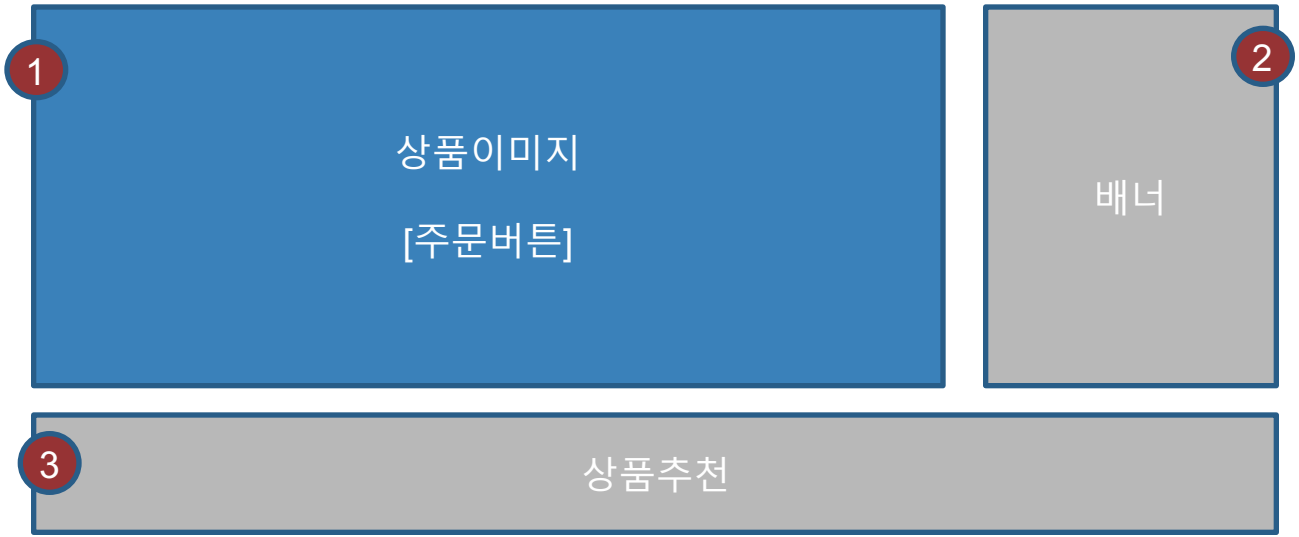
아마존 닷컴은 다양한 서비스 제공과 효율적인 운영 환경 전환을 위해 분산 서비스 플랫폼으로 전환 하였습니다.



2001년 아마존은 주요한 아키텍쳐 변화가 있었는데
기존 모놀리식(Monolitic) 기반에서
서로 다른 어플리케이션 기능을 제공하는
분산 서비스 플랫폼으로 변화 하였습니다.
현재의 Amazon.com의 첫 화면에 들어 온다면,
그 페이지를 생성하기 위해
100여개가 넘는 서비스를 호출하여 만들고 있습니다.

Client-side Rendering : 장애 전파 회피

다음중 가능한 빨리 로딩되어야 하고, 문제가 없어야 할 화면 영역은?

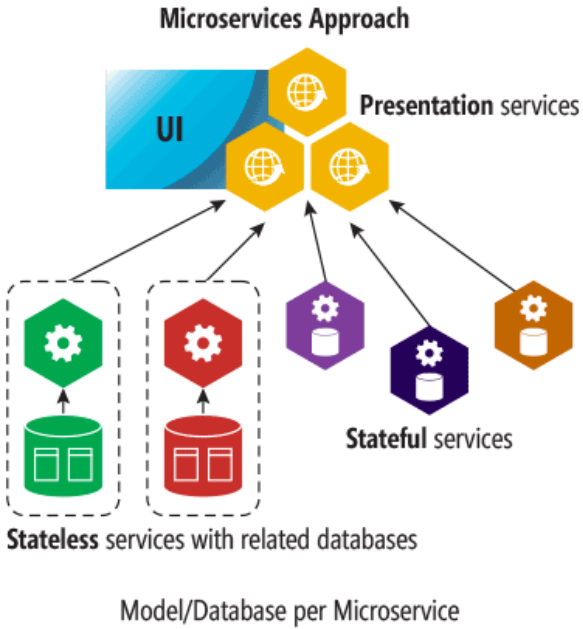


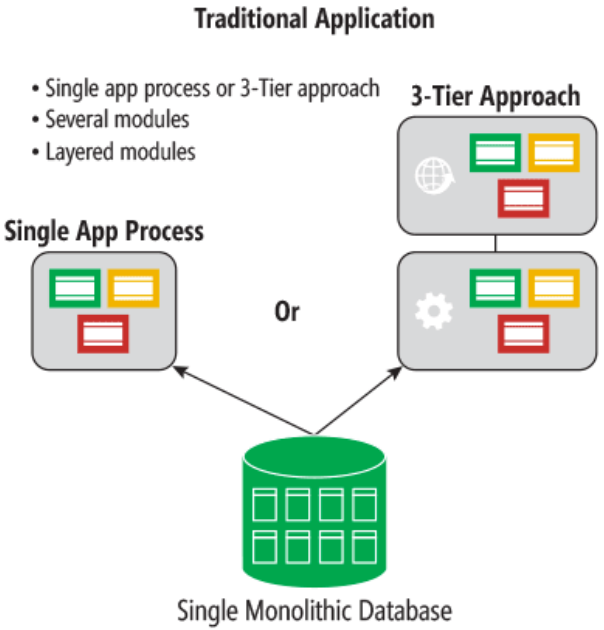
Server-side Rendering 은 모든 화면의 콘텐츠가 도달해야만 화면을 보여줄 수 있지만, **Client-side Rendering** 은 먼저 데이터가 도달한 화면부터 우선적으로 표출할 수 있다.

광고배너가 나가지 못한다고 주문을 안받을 것인가?
그걸 원하지 않는다면 **AJAX / MVVM** 같은 **Client-side Rendering** 기술에 주목하자

Microservice Integration with by UI

- 서비스의 통합을 위하여 기존에 Join SQL 등을 사용하지 않고 프론트-엔드 기술이나 API Gateway 를 통하여 서비스 간 데이터를 통합함
- 프론트엔드에서 데이터를 통합하기 위한 접근 방법으로는 W3C 의 Web Components 기법과 MVVM 그리고 REST API 전용 스크립트가 유용함





W3C Web Components

- Custom elements
- HTML imports
- Template
- Shadow DOM

```
<style>
   style for A
   style for B
   style for C
</style>

<html>
   element for A
   element for B
   element for C
</html>

<script>
    script for A
    script for B
    script for C
</script>
```

→

#index.html

```
<A> <A>
<B> <B> <B>
<C>
```

#A.html

```
<style>
   style for A
</style>

<html>
  element for A
</html>

<script>
   script for A
</script>
```

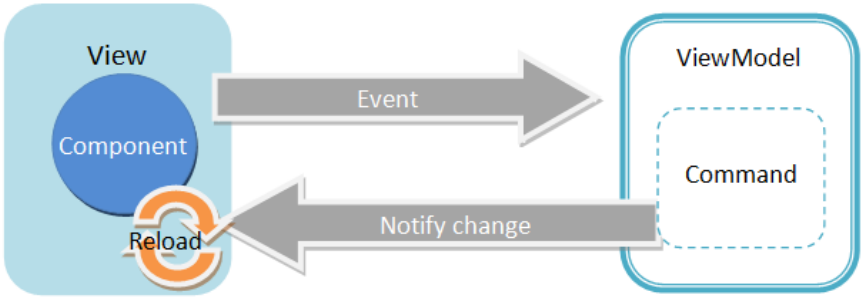
#B.html

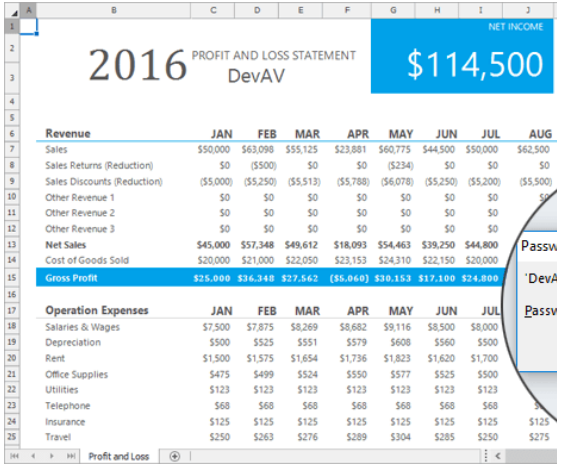
```
<style>
   style for B
</style>

<html>
  element for B
</html>

<script>
   script for B
</script>
```

#C.html

```
<style>
   style for C
</style>

<html>
  element for C
</html>

<script>
   script for C
</script>
```

MVVM





2016 PROFIT AND LOSS STATEMENT DevAV

NET INCOME $114,500

Revenue	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG
Sales	$50,000	$63,098	$55,125	$23,881	$60,775	$44,500	$50,000	$62,500
Sales Returns (Reduction)	$0	($500)	$0	$0	($234)	$0	$0	$0
Sales Discounts (Reduction)	($5,000)	($5,250)	($5,513)	($5,788)	($6,078)	($5,250)	($5,200)	($5,500)
Other Revenue 1	$0	$0	$0	$0	$0	$0	$0	
Other Revenue 2	$0	$0	$0	$0	$0	$0	$0	
Other Revenue 3	$0	$0	$0	$0	$0	$0	$0	
Net Sales	**$45,000**	**$57,348**	**$49,612**	**$18,093**	**$54,463**	**$39,250**	**$44,800**	
Cost of Goods Sold	$20,000	$21,000	$22,050	$23,153	$24,310	$22,150	$20,000	
Gross Profit	**$25,000**	**$36,348**	**$27,562**	**($5,060)**	**$30,153**	**$17,100**	**$24,800**	

Operation Expenses	JAN	FEB	MAR	APR	MAY	JUN	JUL
Salaries & Wages	$7,500	$7,875	$8,269	$8,682	$9,116	$8,500	$8,000
Depreciation	$500	$525	$551	$579	$608	$560	$500
Rent	$1,500	$1,575	$1,654	$1,736	$1,823	$1,620	$1,700
Office Supplies	$475	$499	$524	$550	$577	$525	$500
Utilities	$123	$123	$123	$123	$123	$123	$123
Telephone	$68	$68	$68	$68	$68	$68	$68
Insurance	$125	$125	$125	$125	$125	$125	$125
Travel	$250	$263	$276	$289	$304	$285	$250

Profit and Loss

Frameworks supporting Web Components

Polymer

VueJS

- Provides Cohesive Component Model
- Component Composition by HTML markup
- Dynamic Data Binding
- Responsive Web by Material Design
- Standard

** 비표준: Angular JS, React

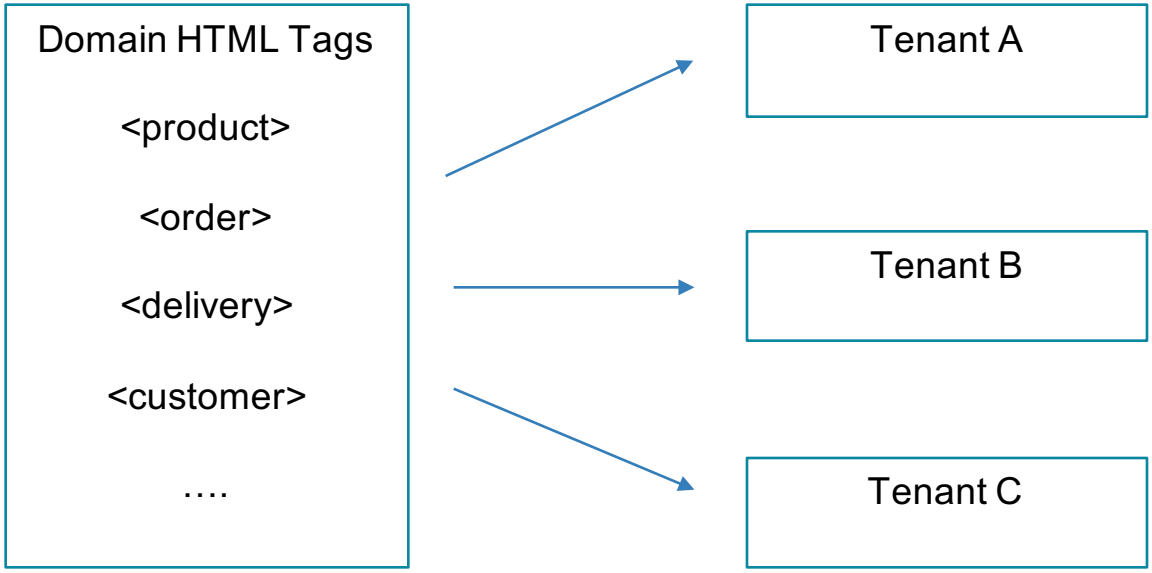
Custom tag for Domain Class: <product> tag

```
<template>
  <div>
    <h1>Product Management</h1>

    <product v-model=”products[index]" v-for='(product, index) in products'
@change="updateProduct" @remove="removeProduct"
@classes="jumpToProducts"></product>

    <h2>Add New Product</h2>
    <product v-model="newProduct" editMode=true isNew=true
@add="addProduct"></product>

  </div>
</template>
```

Data binding to Domain Class Tags

```
<product v-model=”products[index]" v-for='(product, index) in products'
@change="updateProduct" @remove="removeProduct"
@classes="jumpToProducts"></product>


<script>
var products = [];
var me = this;

backend.$bind(”products", products );

 products.$load().then(function(products ){
    me. products = products ; // set the view's data with the loaded data obtained from backend.
});
</script>
```

Tenant UI Customization (Composition) by Tag composition



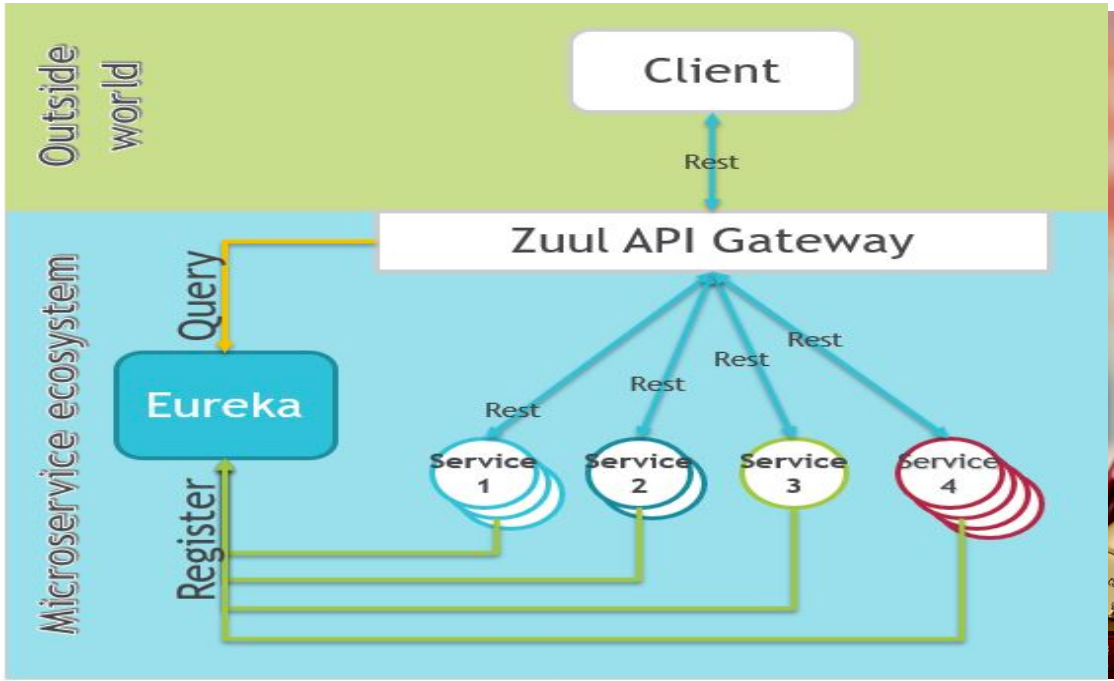
Oops! CORS Exception?

Accessing Multiple Microservices from outside

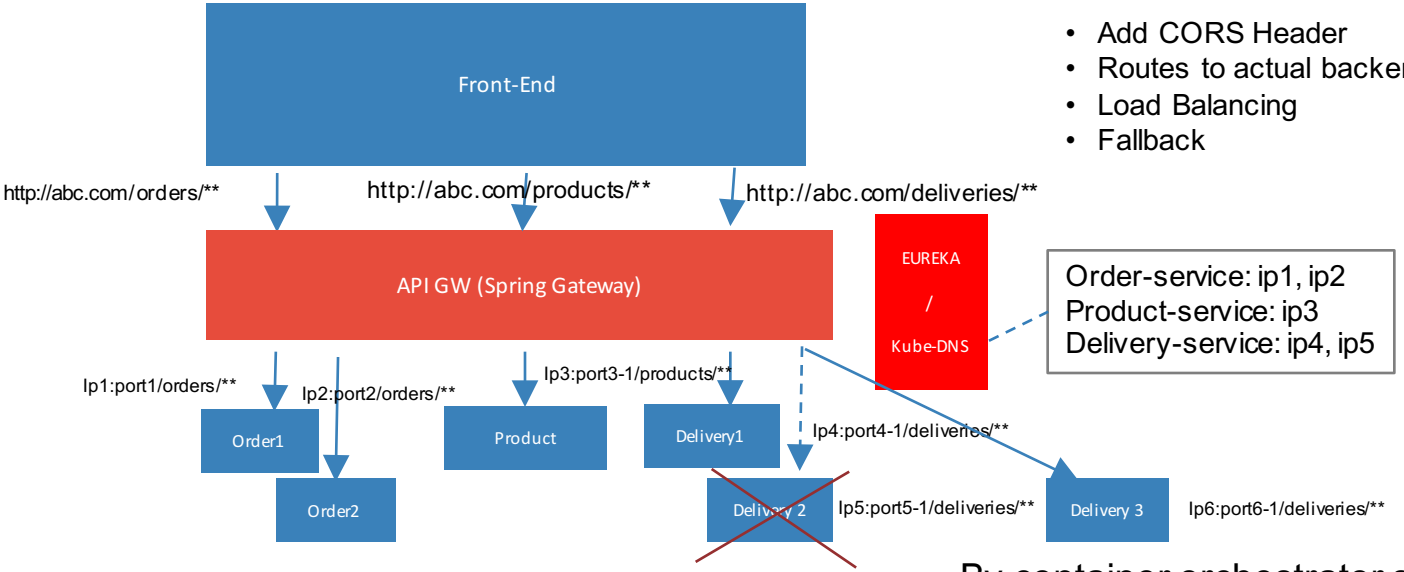
API Gateway: Edge Service

Acting like “Skin” to access our services :

- Re-Routes to multiple services
- Allows CORS
- Checks ACLs
- Prevent DDOS etc.



API Gateway:
Routing, Securing and Load-balancing



- Add CORS Header
- Routes to actual backend service
- Load Balancing
- Fallback

By container orchestrator something

Configuring Spring Gateway

#application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id: product
          uri: http://product:8080
          predicates:
            - Path=/product/**
        - id: order
          uri: http://order:8080
          predicates:
            - Path=/order/**
        - id: delivery
          uri: http://delivery:8080
          predicates:
            - Path=/deliveries/**
```

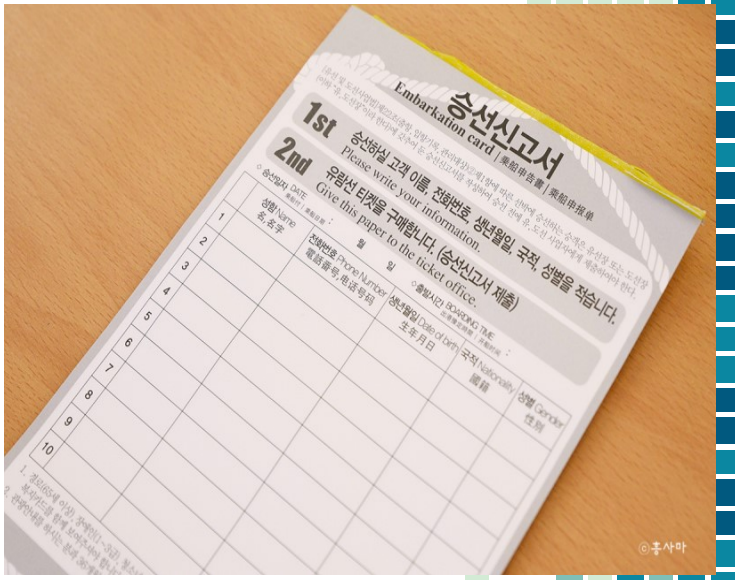
Service Registration: EUREKA or Kube-DNS

spring Eureka

HOME LAST 1000 SINCE STARTUP

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
BPM	n/a (1)	(1)	UP (1) - 192.168.0.47:bpm:8090
DEFINITION	n/a (2)	(2)	UP (2) - 192.168.0.47:definition:8091 , 192.168.0.47:definition:8089
ZUUL-ROUTER	n/a (1)	(1)	UP (1) - 192.168.0.47:zuul-router:8080

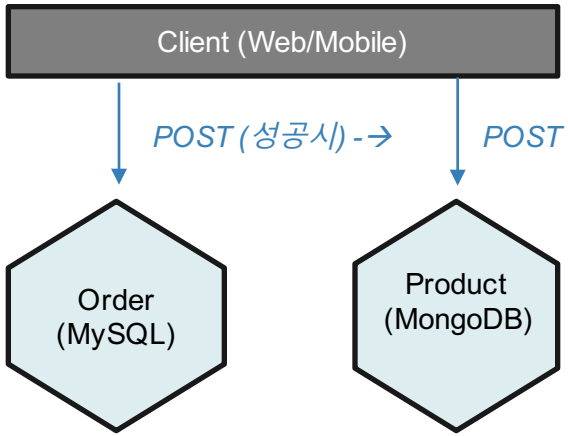


OAuth2 Authorization Server implemented by Spring Security

- https://github.com/Baeldung/spring-security-oauth
- https://www.keycloak.org/
- https://github.com/TheOpenCloudEngine/uEngine-cloud/tree/master/uengine-cloud-iam

Transaction Problem in UI composition

서비스구성 (클라이언트가 순차 호출)



조회화면

Consistent

In-consistent

주문량: 0 재고량: 10

주문량 : 1 재고량 : 10

주문량 : 2 재고량 : 9

주문량 : 3 재고량 : 8

둘다 순조롭게 호출 성공시 일치

불일치 영원히 생길 수 있음!!

Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

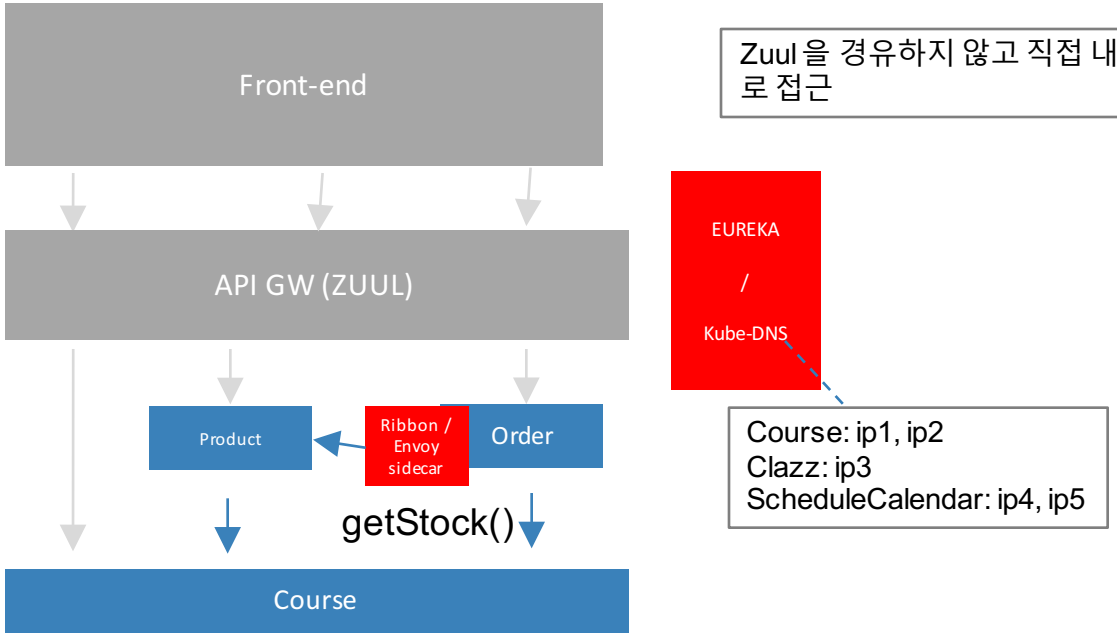
Service Composition by Request-Reponse

- Inter-microservices call requires client-side discovery and load-balancing
- Netflix Ribbon and Eureka
- Hiding the transportation layer: Spring Feign library and JAX-RS
- Circuit Breaker Patterns

서비스의 호출 방식 2가지

외부에서	마이크로 서비스 간 (inter-microservice call)
AJAX 로 REST 호출	HTTPClient 로 REST 호출 혹은 Event 로 pub/sub
Hybind 를 사용, 자바스크립트객체와 Resource 바인딩	Feign 을 사용, @Autowired 로 proxy 객체리턴 혹은 MOM (Apache Kafka), Spring Cloud Stream
Zuul 을 경유하여 Eureka 에서 발견된 path 에 따른 서비스가 라우팅	Ribbon 이 Eureka 를 통하여 직접 마이크로 서비스가 로드밸런싱 된 대상을 선택

Inter-microservices Call – Request/Response



FeignClient

---- url 을 기반으로 Kube-DNS 를 찾아감

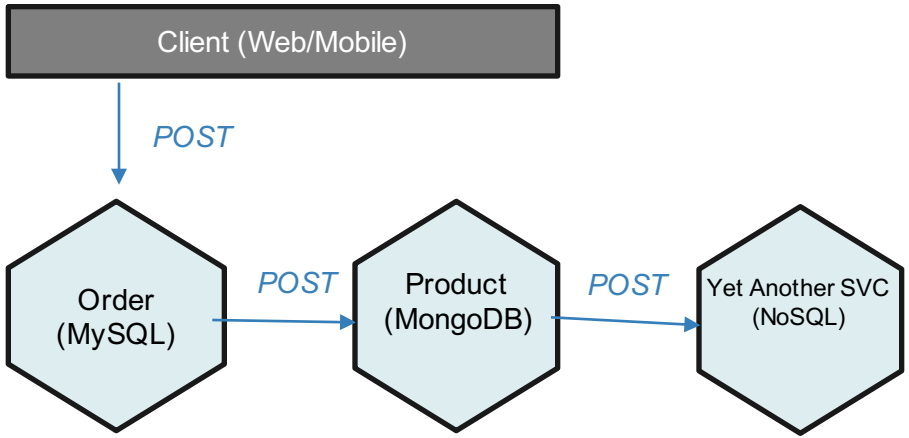
```
@FeignClient(name = ”product”, url=“http://product:8080”)
public interface ProductService {

 @RequestMapping(method  = RequestMethod.GET, path="/products/{productId}")
  Product getProduct(@PathVariable(”productId") Long productId);

}
```

Consumer Code: Order

```
@Entity
public class Order{

    @Autowired
    ProductService ProductService;

    @PrePersist
    public void beforeSave() {

            Product product = ProductService.getProduct(getProductId());

            if(product.getStock() < getQty()){
                throw new RuntimeException("No Available stock!");
            }

    }
}
```

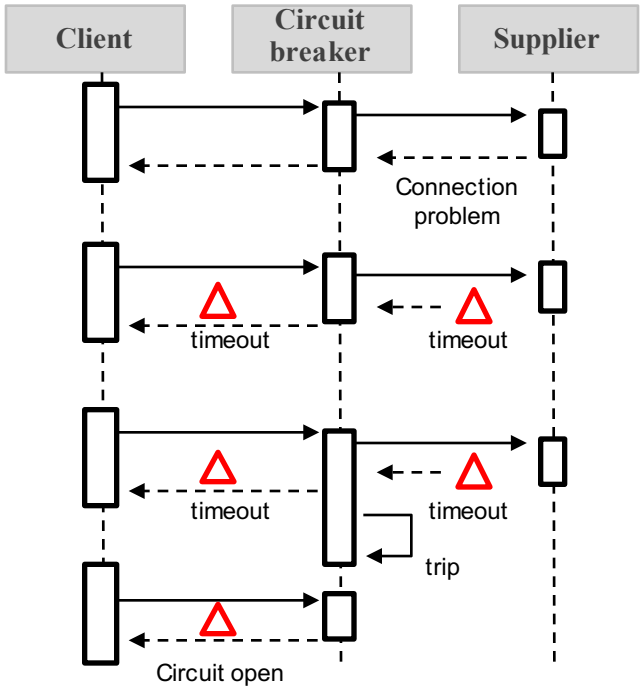
Issues in Request-Response model

서비스구성 (Request-Response, Sync 호출)



- 성능저하
- 장애전파
- 트랜잭션 처리를 위한 2PC 구현

장애전파 차단: 서킷브레이커 패턴



장애 감지 시, 차단기 작동(Open)

↓

일시 동안 Fallback으로 서비스 대체

↓

일부 트래픽으로 서비스 정상여부 확인

↓

정상 확인 시, 차단기 해제(Close)

Service Composition by Event-driven Model

- Event-driven Approach
- ACID Tx. vs Eventual Tx.
- Business Transaction / Compensation (Saga) Pattern

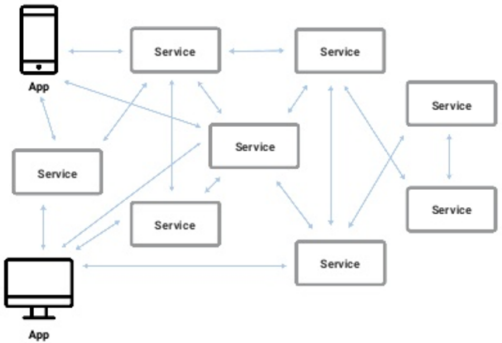
Microservice Integration with Event-driven Architecture

Request-Response Applications

Deterministic

Rigid

Tight coupling



confluent

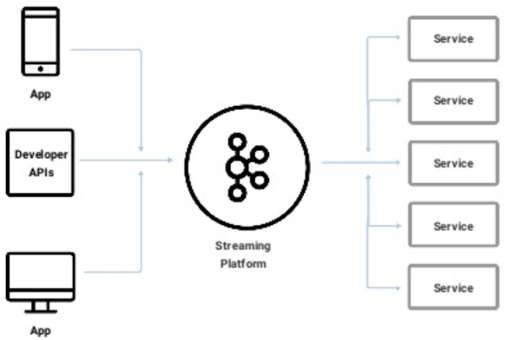
Vs.

Event-Driven Applications

Responsive

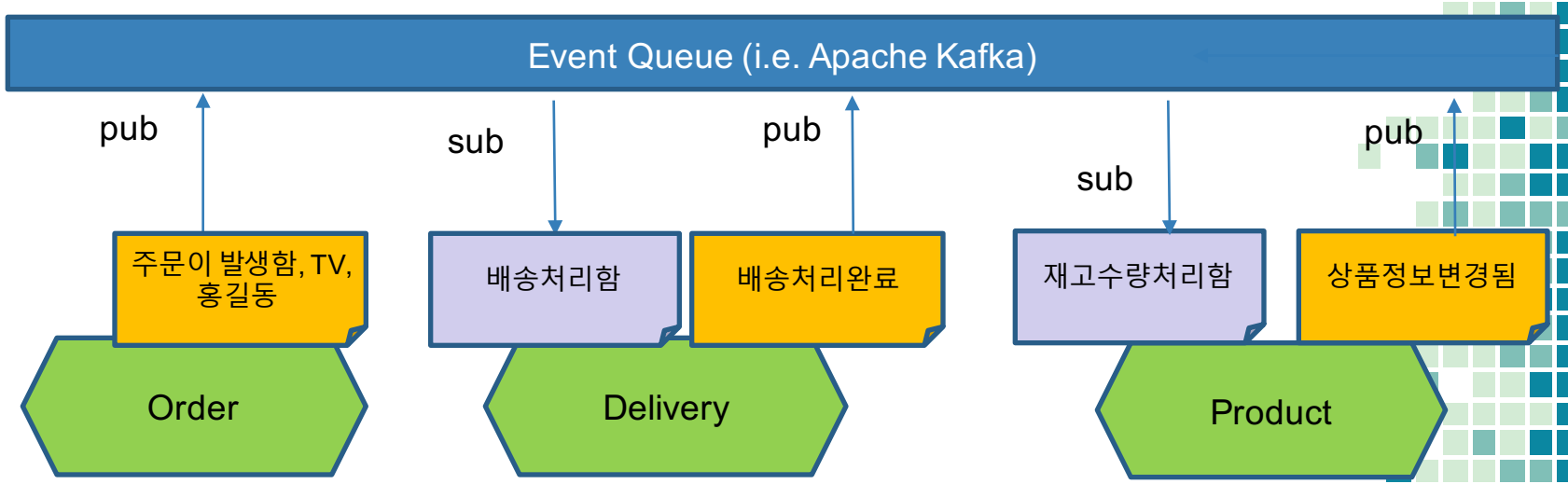
Flexible

Extensible



confluent

Inter-microservice Call - Event Publish / Subscribe (PubSub)



Event Publisher : Order

```
@Entity
public class Order {
 …

    @PostPersist
    private void publishOrderPlaced() {
        OrderPlaced orderPlaced = new OrderPlaced();

        orderPlaced.setOrderId(id);
              …

        kafka.send(orderPlaced);
    }
}
```

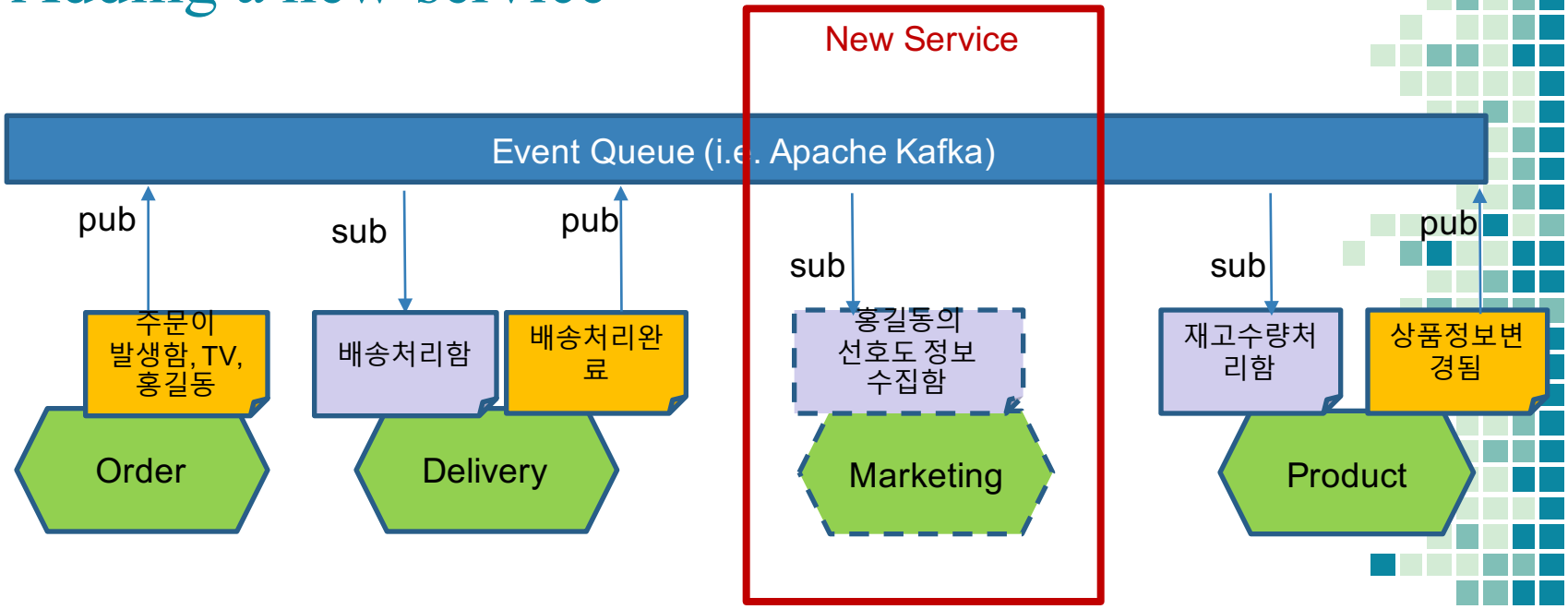
Event Consumer: Delivery

```
@KafkaListener(topics = "shopping")
   public void onOrderPlaced(…) {

      Delivery delivery = new Delivery();
      delivery.setOrderId(orderPlaced.getOrderId());
      delivery.setDeliveryAddress(orderPlaced.getCustomerAddr());
      delivery.setCustomerName(orderPlaced.getCustomerName());
      delivery.setDeliveryState(DeliveryStarted.class.getSimpleName());

      deliveryRepository.save(delivery);

      Kafka.send(new OrderShippped(delivery));  //triggers new event
   }
```

Event Consumer: Product

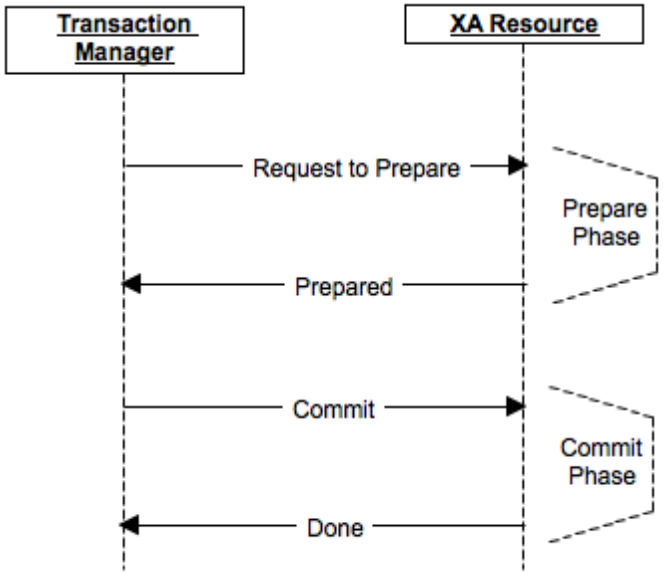
```
@KafkaListener(topics = ”shopping")
public void onOrderShipped(…) {

    Product product = productRepository.findById(orderShipped.getProductId());
    product.setStock(product.getStock()  – 1);
    productRepository.save(product);

}
```

Adding a new service

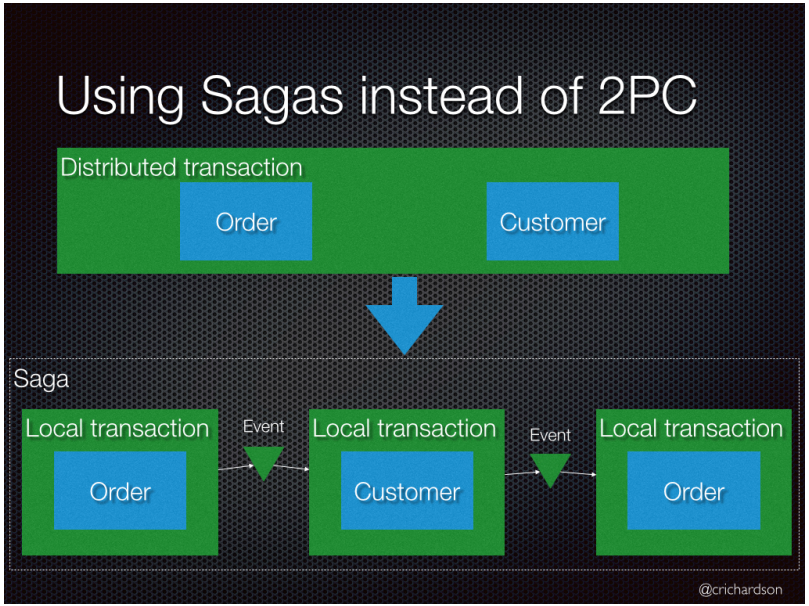


Distributed Transaction Issues

Atomic Transaction / 2PC

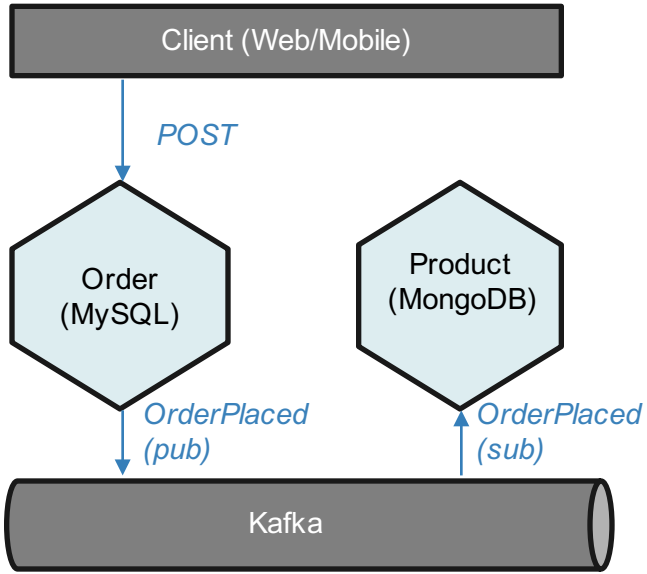


Eventual Transaction / compensation



Eventual TX를 통한 분산 트랜잭션 처리

서비스구성 (Eventual TX)



조회화면

Consistent

In-consistent

주문량: 0 재고량: 10

주문량: 1 재고량: 10

주문량 : 1 재고량: 9

주문량 : 2 재고량: 9

잠깐 불일치

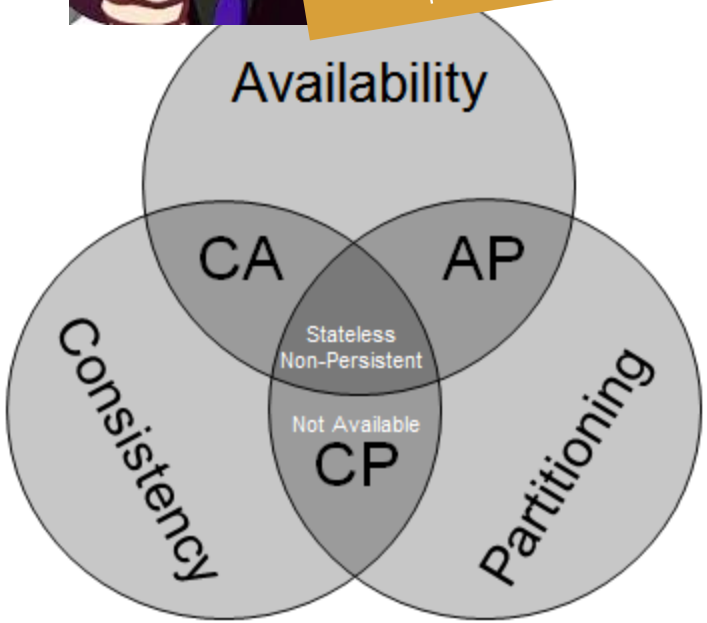
주문량: 2 재고량: 8

결국 일치

여기서 선택의 길을 만남

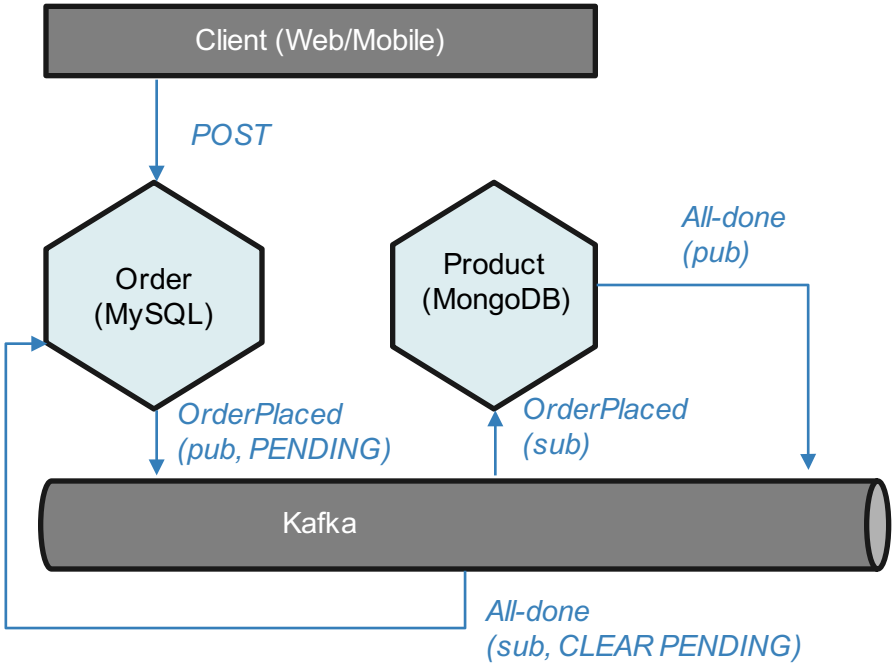
- 내 서비스에서 데이터 불일치가 얼마나 미션 크리티컬 한가?
- 크리티컬 하다고 응답한다면, 내 서비스의 성능과 확장성에 비하여 크리티컬 한가?
- 성능과 확장성 보다 크리티컬 하다면, 성능과 확장성을 포기할 수 있는가?
- *성공한 내 서비스의 경쟁자들은 무엇을 포기했는가?*
- ➔ 강력한 의사결정 필요 (무엇으로 태어날 것인가…)

미안하지만…
하나만 선택해!!!



Eventual TX – 불일치가 Mission Critical 한 경우

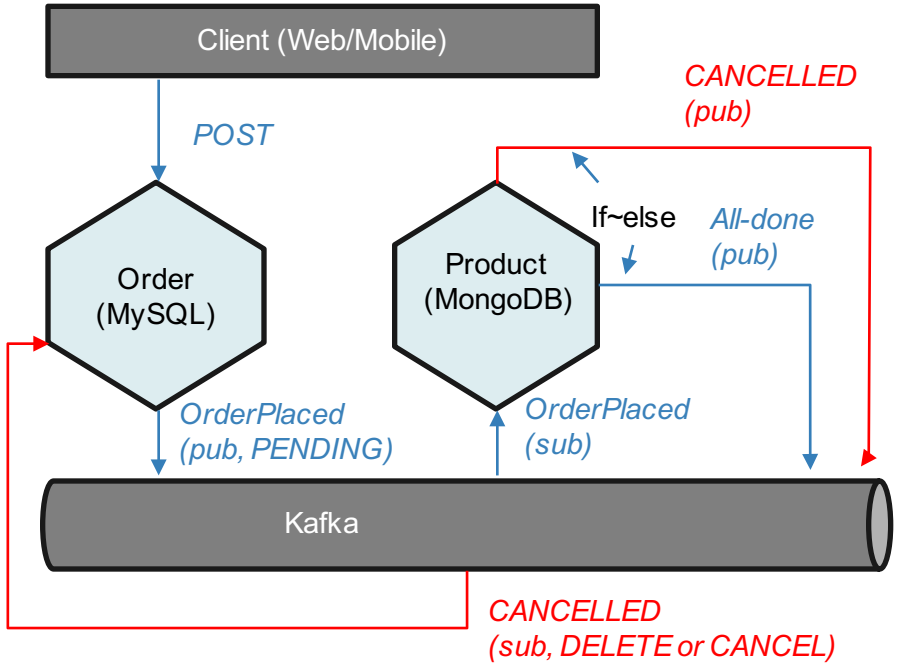
서비스구성 (Eventual TX)



조회화면

Eventual TX – Rollback (Saga Pattern)

서비스구성 (Eventual TX)



조회화면

Consistent

In-consistent

주문량 : 0 재고량 : 10

주문량 : 1 재고량 : 10 (PENDING)

주문량 : 1 재고량 : 9

주문량 : 2 재고량 : 9 (PENDING)

주문량 : 2 재고량 : 8 → 취소됨

주문량 : 2 재고량 : 9

복구 & 결국 일치

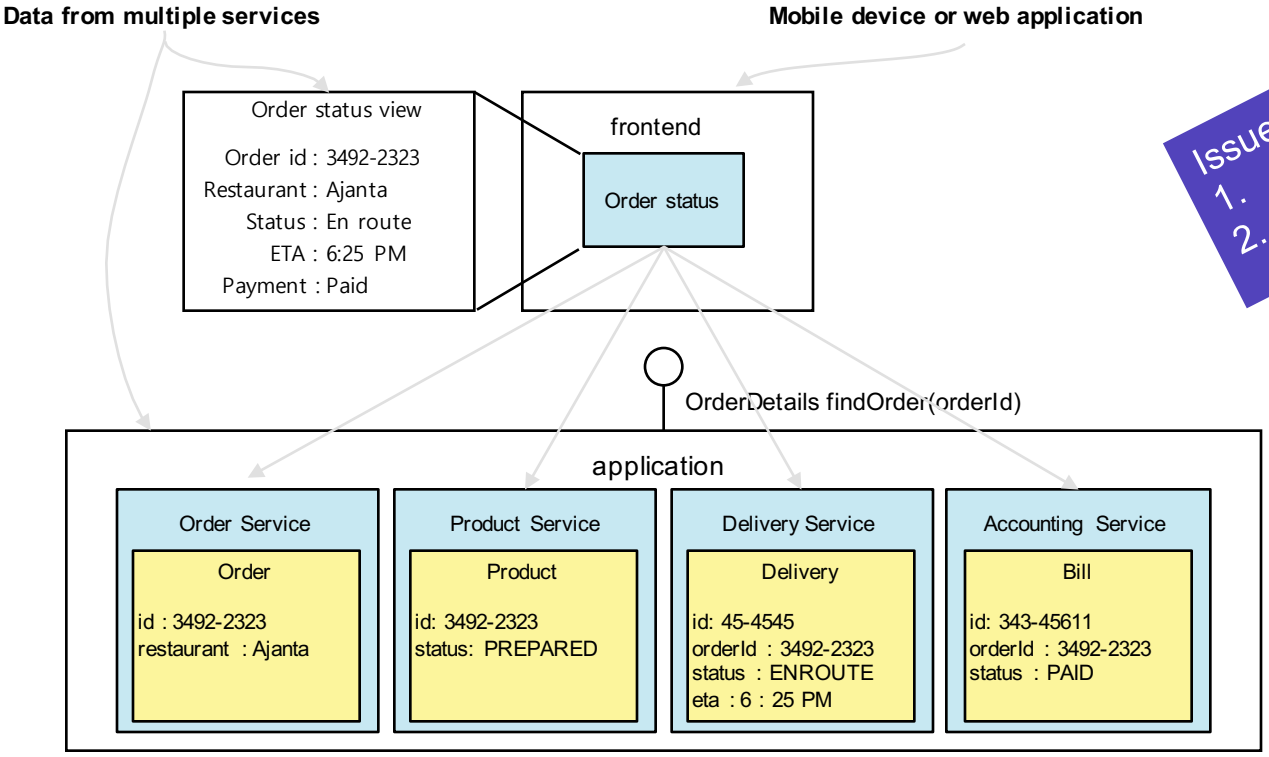
Resources on Sagas

- http://eventuate.io/
- https://vladmihalcea.com/how-to-extract-change-data-events-from-mysql-to-kafka-using-debezium/?fbclid=IwAR33Spb4jPBNI6VNHuCxdu_BxpWdzOLzMvbCtHHvJrRmJPfiEoXwM1qWYBs

Data Query (Projection) in MSA

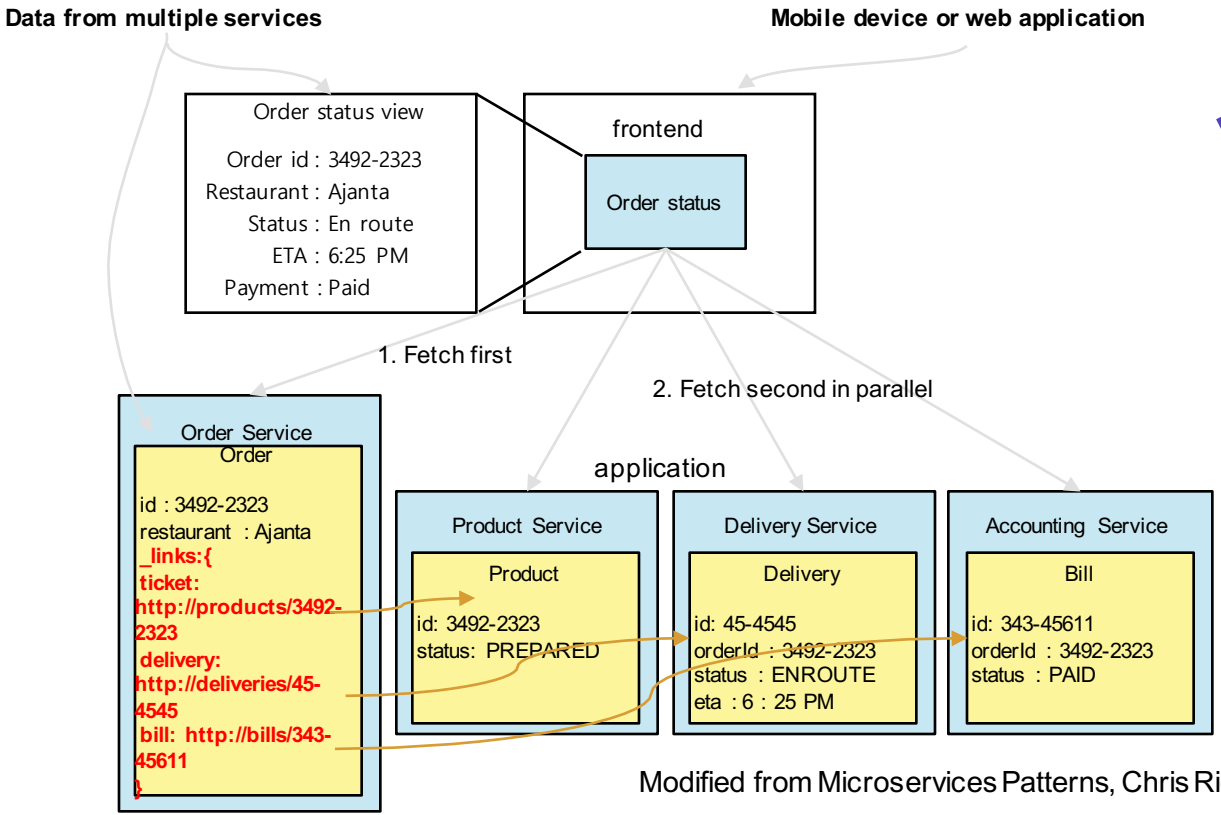
- Issues: Existing Join Queries and Performance Issues
- Composite Services or GraphQL
- Event Sourcing and CQRS

분산서비스에서의 Data Projection Issue



Modified from Microservices Patterns, Chris Richardson, Manning, 2018

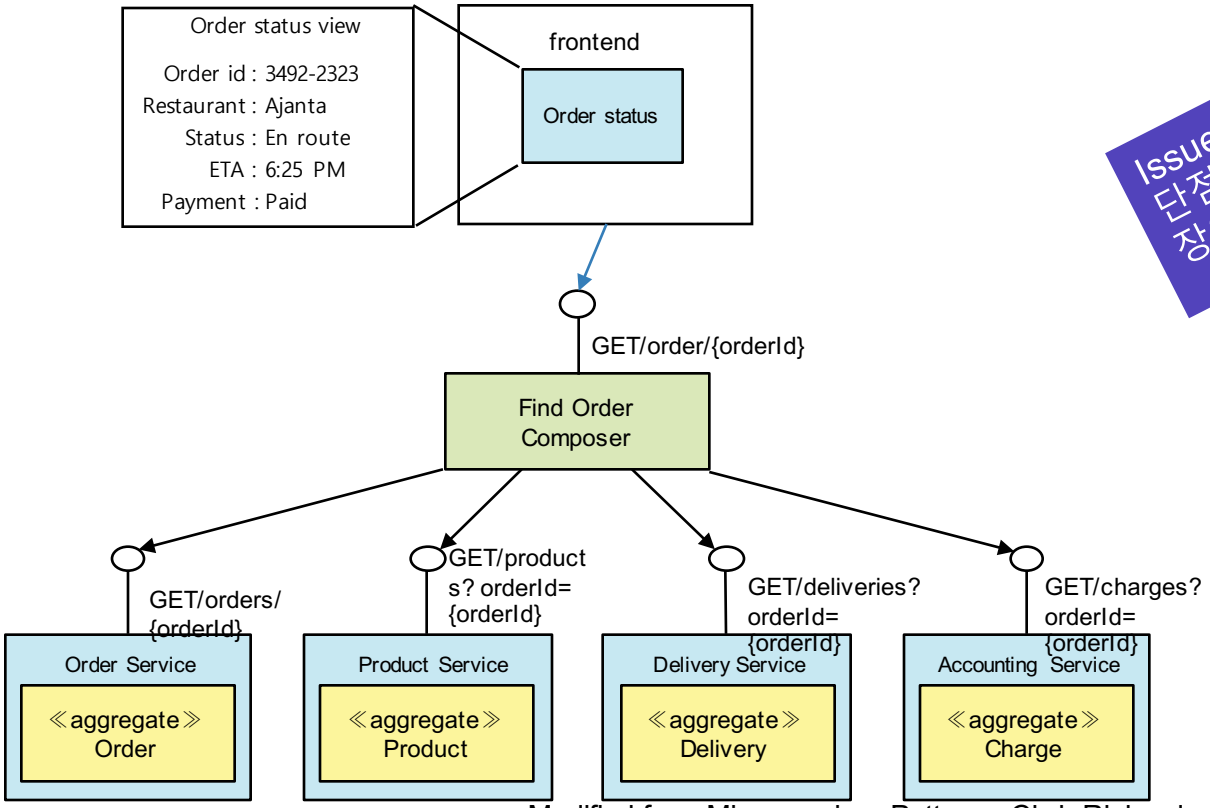
Data Projection by UI and HATEOAS



Issues:
단점: UI 개발
복잡성과 성능

Modified from Microservices Patterns, Chris Richardson, Manning, 2018

Data Projection by Composite Service or GraphQL



Issues:
단점: 성능(속도), 장애전파

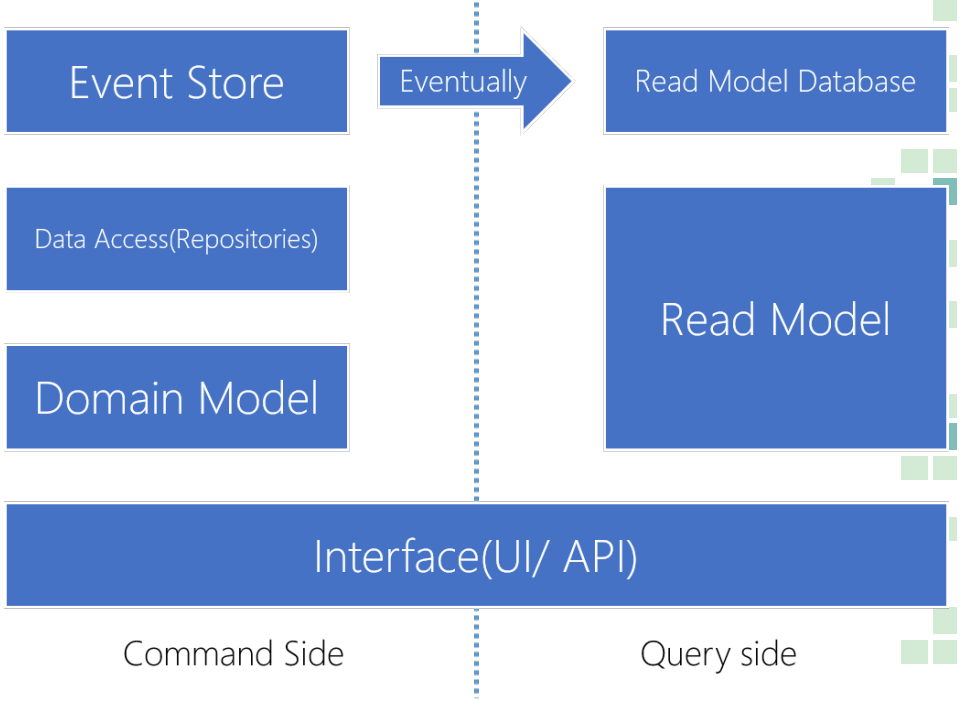
Modified from Microservices Patterns, Chris Richardson, Manning, 2018

생각을 다르게 하자
: CQRS and Event-Sourcing

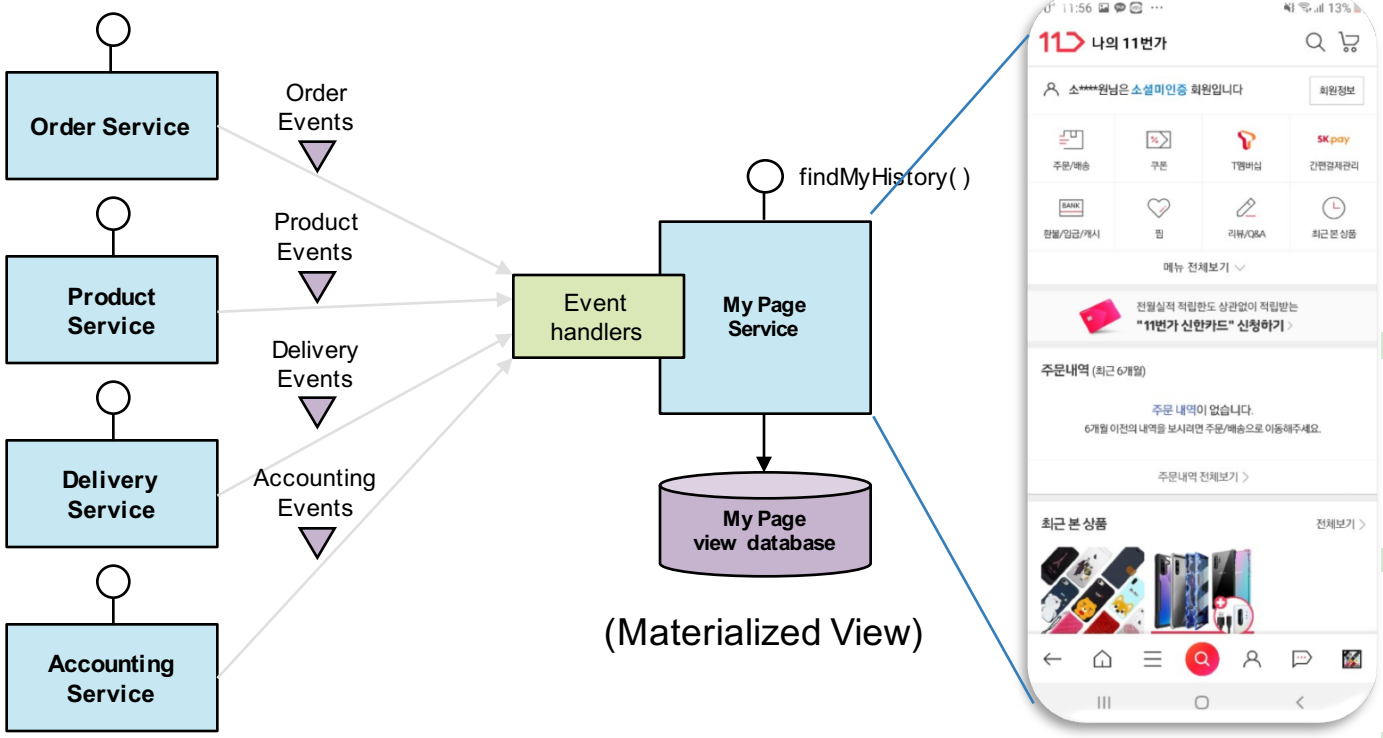
- CQRS 패턴
- 읽기전용 DB와 쓰기 DB를 분리함으로써 Optimistic Locking 구현
- Query 뷰를 다양하게 구성하여 여러 MSA 서비스 목적에 맞추어 각 서비스의 리드모델에 부합
- Polyglot Persistence

https://justhackem.wordpress.com/2016/09/17/what-is-cqrs/

https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-1-richardson



CQRS 의 확장 : Multiple Event Sources



Modified from Microservices Patterns, Chris Richardson, Manning, 2018

Requirements / Check

Category	Requirement	Solution
Maintainability & Scalability	24*365 Zero Down time	Self-Healing, Horizontally Scale
	Minimal Dev-Ops, Dev-Dev team side effects in deploy time (more than 100 deploys per day)	Loose coupled design (HATEOAS API, Service Decomposition), PubSub
		Auto Provisioning (by docker containerizing)
		DevOps CI/CD
Scalability & Reusability	Multi-channel Support	Responsive Web, Chat Bot
	Dynamic Service Composition	Dynamic Discovery/Composition, BPM
Usability & Performance	Fast browsing, Single Page, DDoS protection, Inter-microservice integration	Client-side UI rendering, API Gateway, Circuit Breaker, Event-driven Arch.
Security	For all services including 3rd-party ACL	Access Token, IAM

Table of Content

Microservice and SOA Programming+Microservice-Based DevOps Project

1. Review for the Domain Problem: A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices
6. Front-end Development in MSA
7. Service Composition with Request-Response and Event-driven
8. Implementing DevOps Environment with Kubernetes, Istio and Gitlab

DevOps

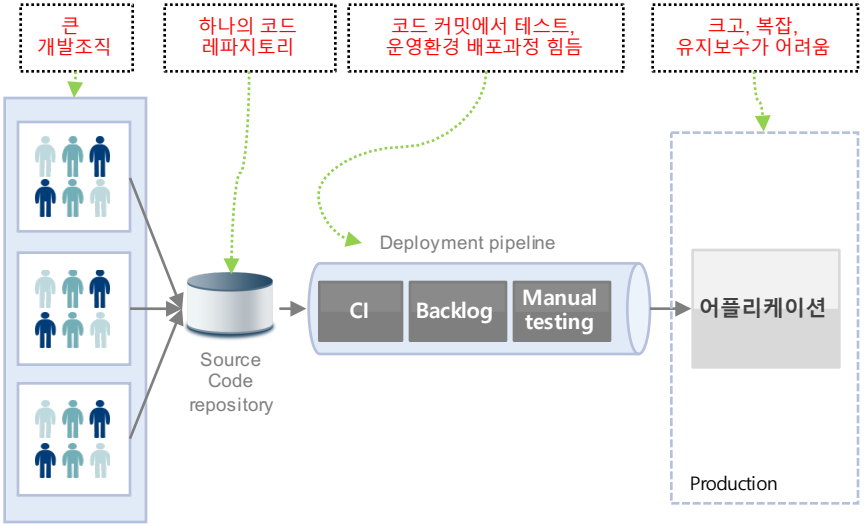
- DevOps Process and Tools
- Deploy Strategies
- Containers and Orchestrators
- Cloud Foundry, Bluemix, and Kubernetes

https://www.youtube.com/watch?v=_I94-tJlovg

Process Change: 열차말고 택시를 타라!

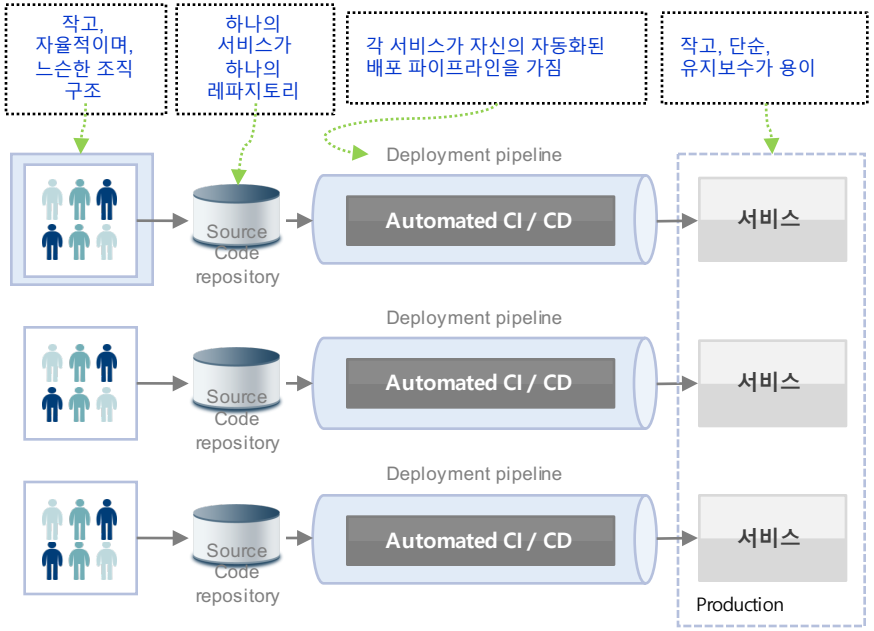
모노리식 개발 및 운영환경

- 모노리식 환경하에서는 큰 개발팀이 하나의 소스코드 레파지토리에 변경 사항을 커밋하므로 코드간 상호 의존도가 높아 신규 개발 및 변경이 어려움
- 작은 변경에도 전제를 다시 테스트/ 배포하는 구조이므로 통합 스케줄에 맞춘 파이프라인을 적용하기가 어렵고 Delivery 시간이 과다 소요



마이크로서비스 개발 및 운영환경

- 작고 분화된 조직에서 서비스를 작은 크기로 나누어 개발하므로 해당 비즈니스 로직에만 집중하게 되어 개발 생산성 향상
- 연관된 마이크로서비스만 테스트를 수행하므로 개발/테스트/배포 일정이 대폭 축소

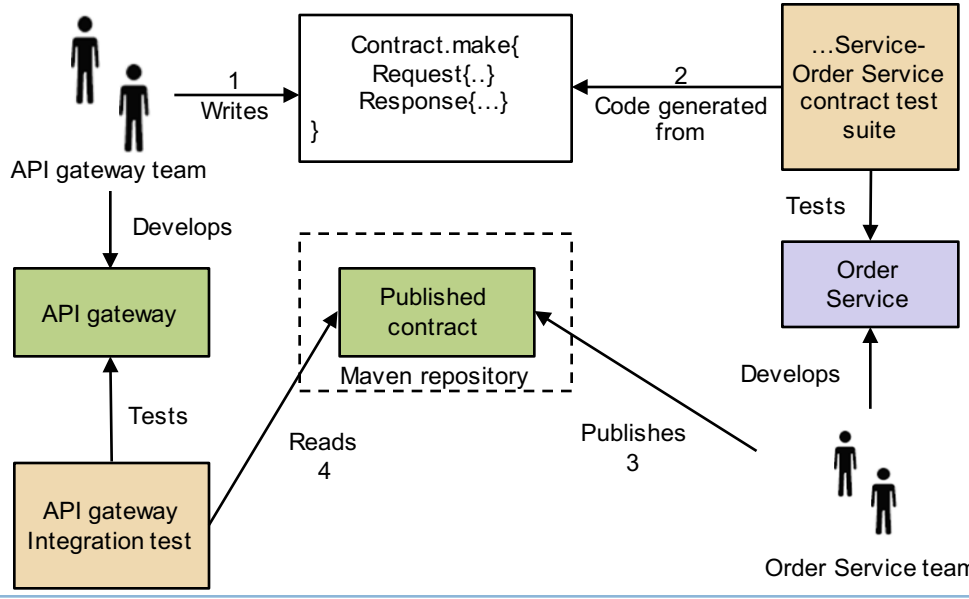


Modified from Microservices Patterns, Chris Richardson, Manning, 2018

Process Change: Consumer-driven Test

MSA 서비스의 테스트

- 컨트랙트 테스트는 서비스 수요자가 주도하여 테스트를 작성한다. 수요자가 테스트를 작성하여 제공자의 레포지토리에 Pull-Request 하면, 제공자가 이를 Accept 한후에 제공자측에서 테스트가 생성되어 테스트가 벌어진다.

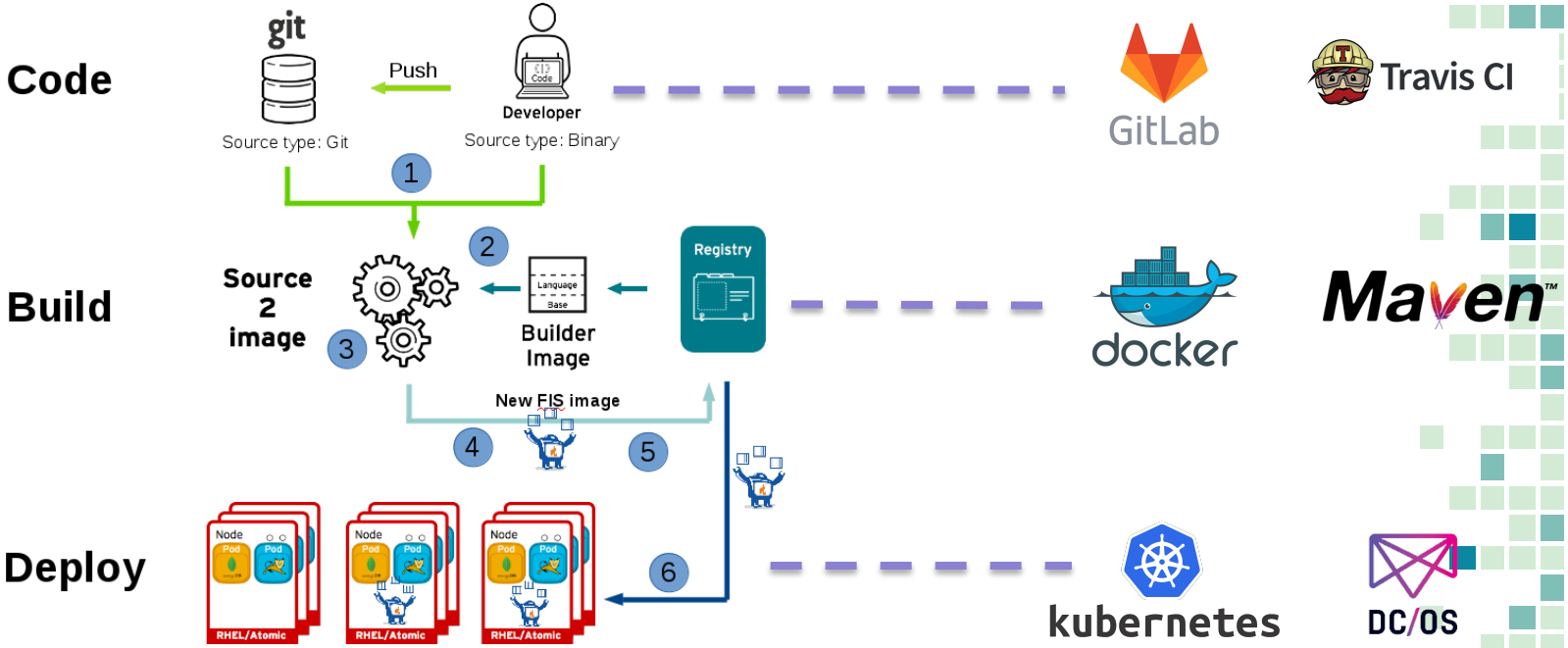


특징

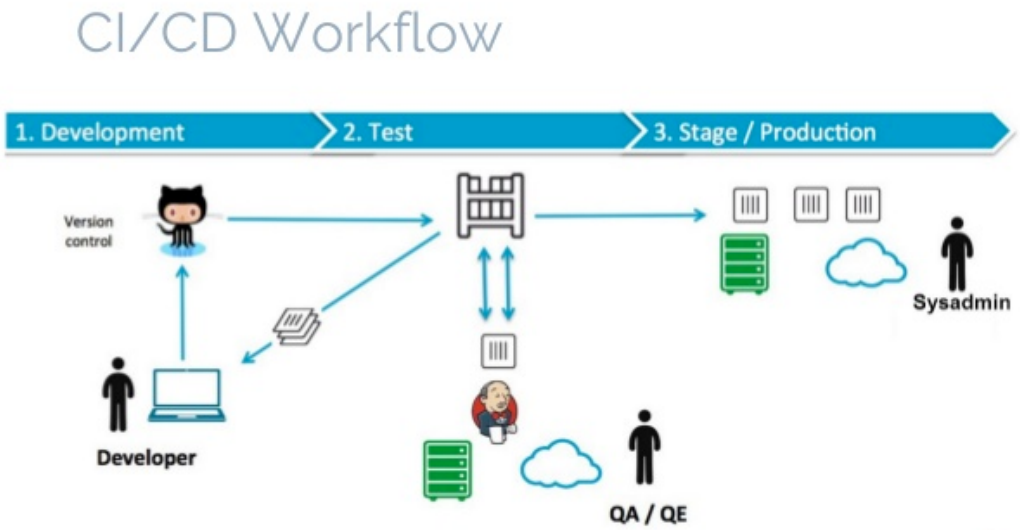
- MSA 테스트에는 Contract-based Test가 특징적
- API Consumer가 먼저 테스트 작성
- API Provider가 Pull Request 수신 후 테스트는 자동으로 생성됨
- Consumer 측에 Mock 객체가 자동생성 병렬개발이 가능해짐
- Spring Cloud Contract가 이를 제공
- Provider의 일방적인 버전업에 따른 하위 호환성 위배의 원천적 방지

Modified from Microservices Patterns, Chris Richardson, Manning, 2018

DevOps toolchain

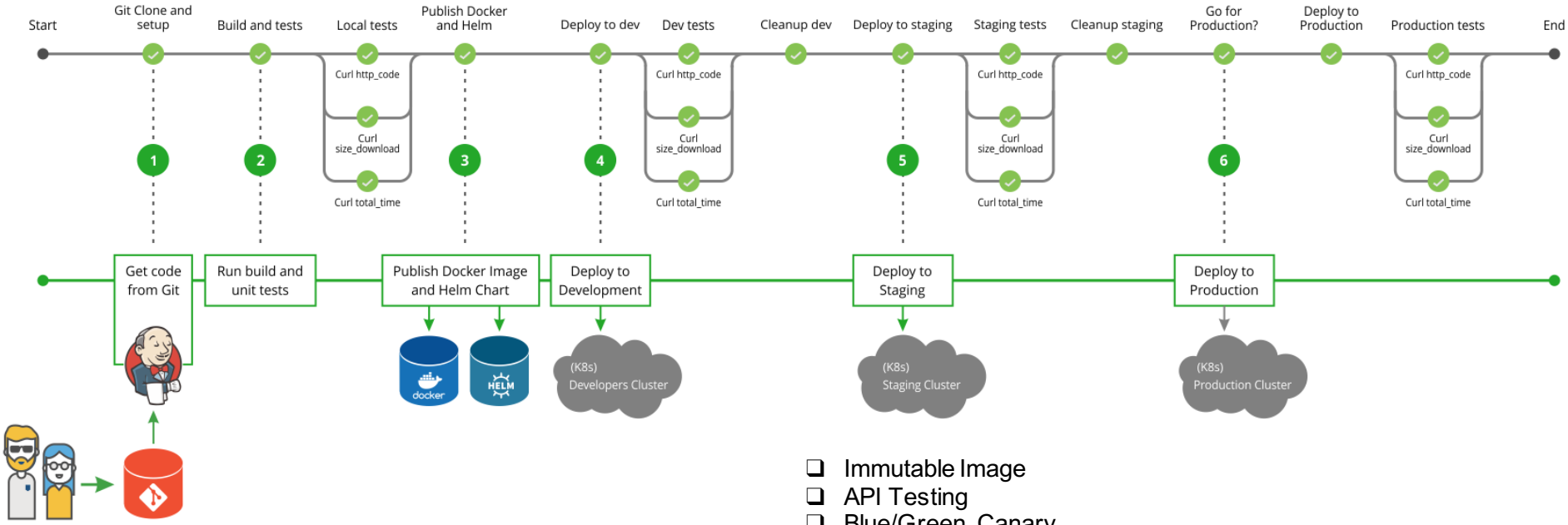


CI/CD Tools – conventional CI/CD



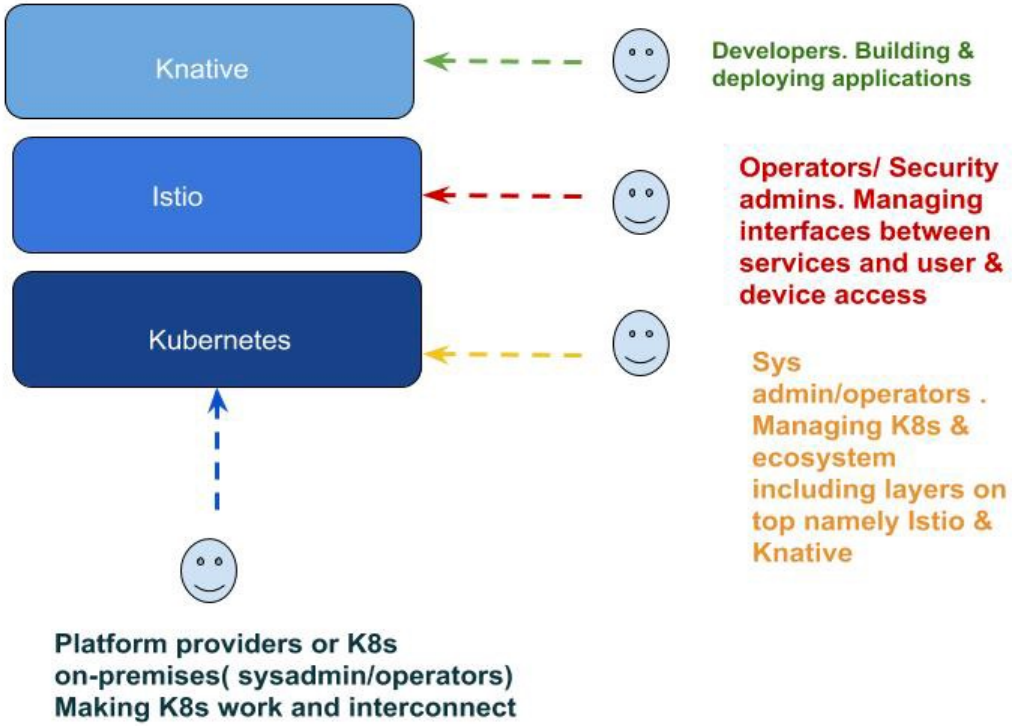
- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

CI/CD Tools – Container-based



- ❑ Immutable Image
- ❑ API Testing
- ❑ Blue/Green, Canary
- ❑ 도구: Jenkins + Docker + Spinnaker + Helm + Kubernetes
  → 매우복잡

CI/CD Tools - Serverless



- Infrastructure As A Code
- Application Configuration 과 Infra Configuration을 하나의 설정에 통합
- 단일 도구

비교

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Container Solutions

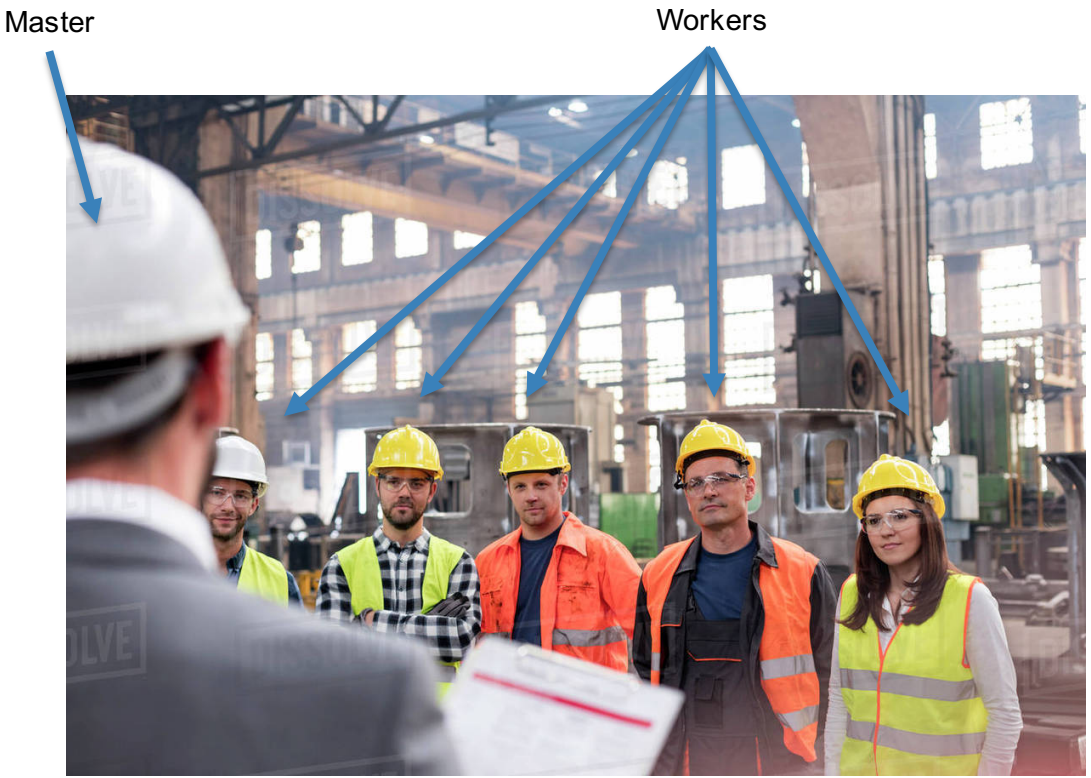
Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

DevOps

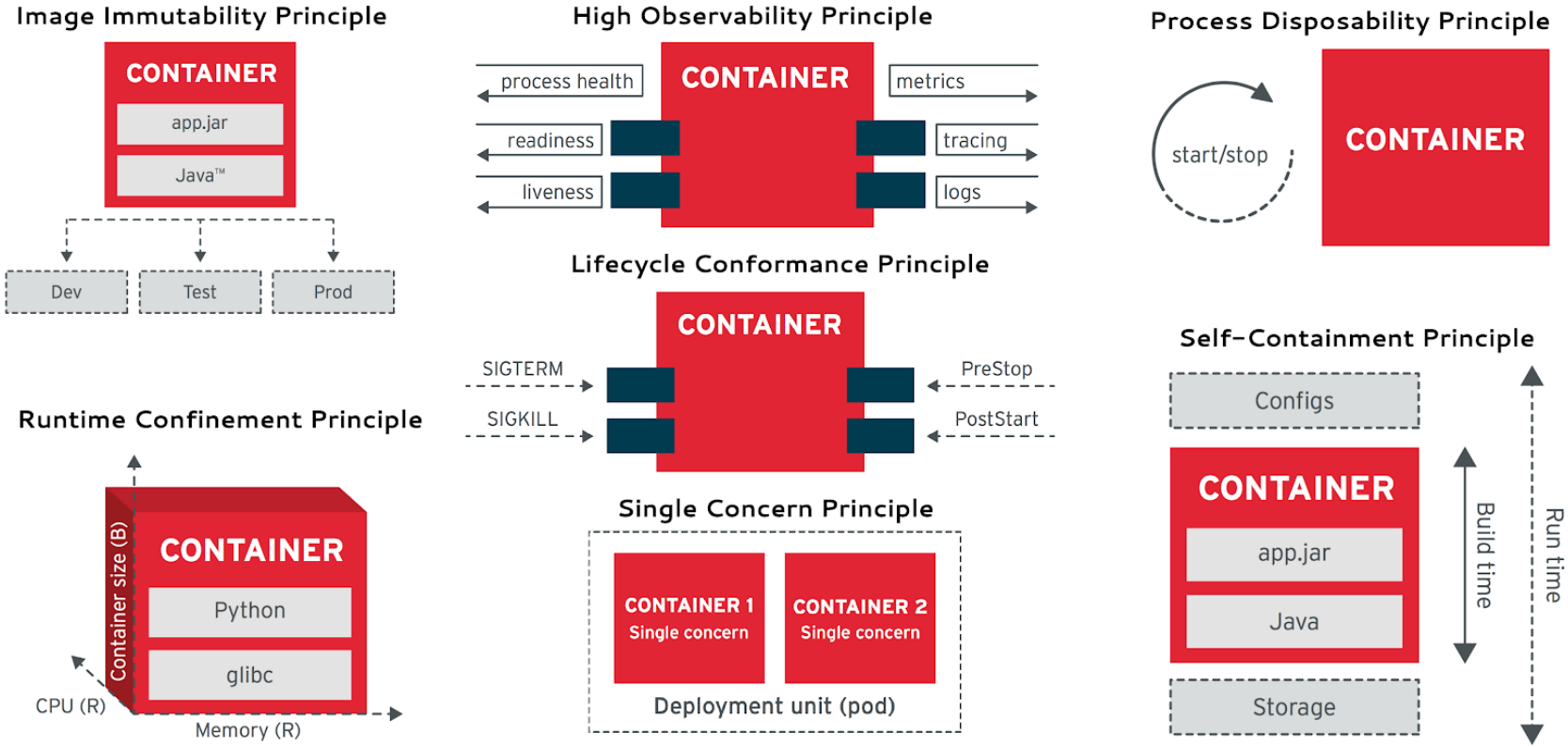
DevOps Platforms

Container	Workload Distribution Engine (Container Orchestrator)	PaaS
• Docker	• Kubernetes • Docker SWARM (toy) • Mesos Marathon (DCOS)	• Google Compute Engine • Redhat Open Shift • IBM Bluemix • Amazon ECS
• Warden(Garden)	• Cloud Foundry	▪ IBM Bluemix ▪ Heroku ▪ GE's Predix ▪ Pivotal Web Services
• Hypervisor	• CF version 1 • Engineyard….	• Amazon Beanstalk

Kubernetes

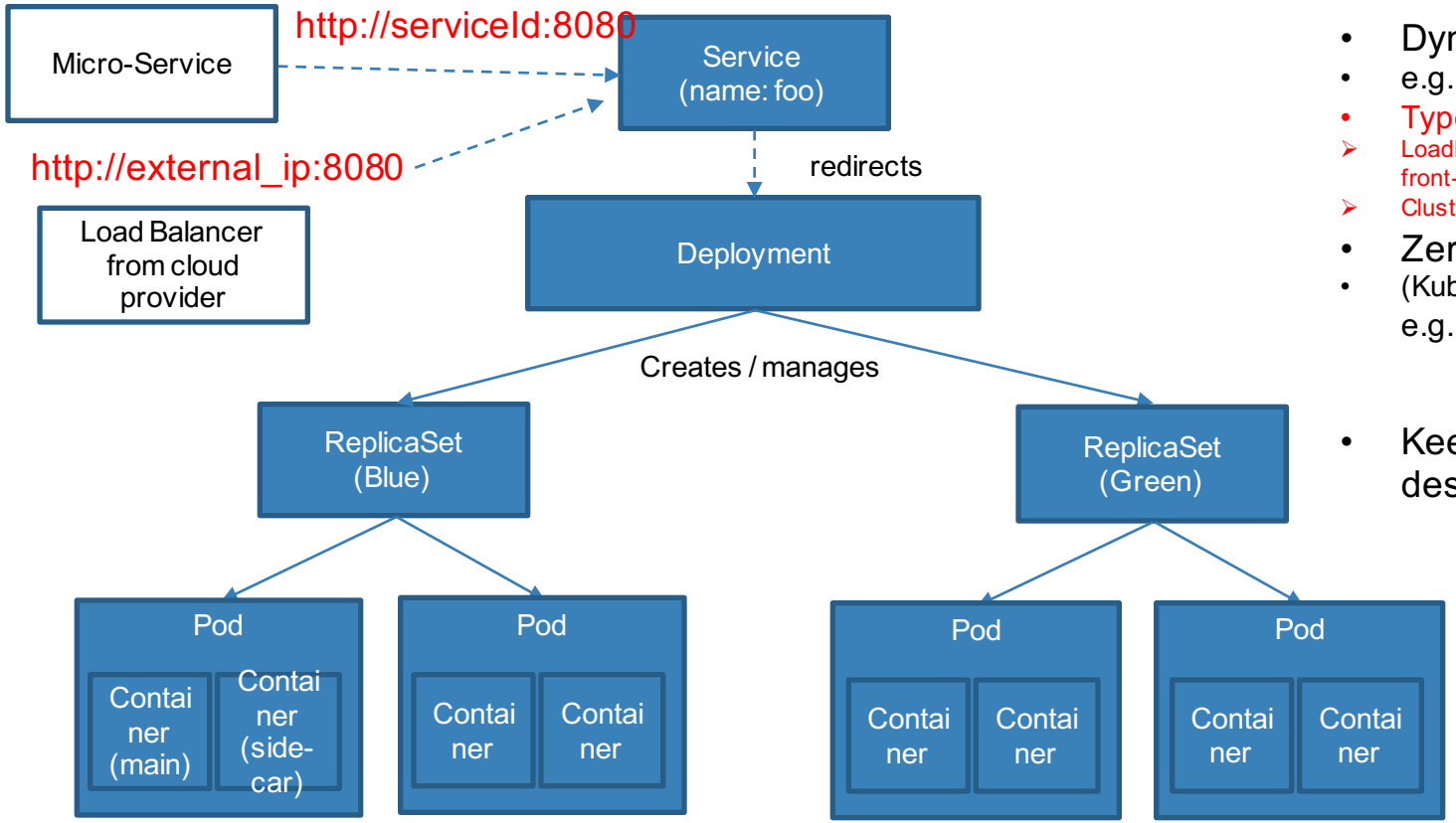


Container-based Application Design



https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIIkCIwpaMSL5mmW7VMaQb6hRV8hkd38

Kubernetes Object Model

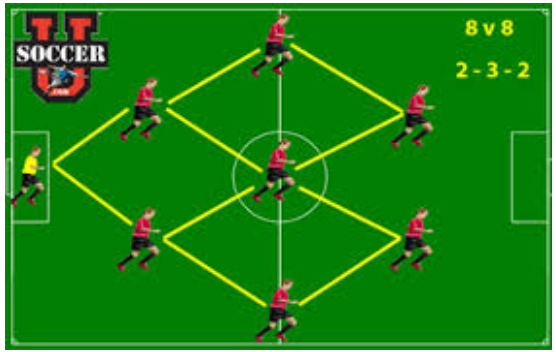


Responsible for:

- Dynamic Service Binding
- e.g. http://foo:8080
- Type:
  - LoadBalancer e.g. Ingress (API GW) or front-end
  - ClusterIP e.g. 내부 마이크로 서비스들
- Zero-down time deployment
- (Kubernetes default is rolling-update) e.g. Blue / Green
- Keep replica count as desired (replicas=2)
- Service Hosting

Declaration based configuration

> Desired state



> my-app.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
…
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
---
apiVersion: apps/v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  labels:
    app: backend
spec:
  replicas: 3
…
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: backend:latest
        ports:
        - containerPort: 8080
---
apiVersion: apps/v1
kind: Service
metadata:
  name: backend-service
  labels:
    app: backend
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db-deployment
  labels:
    app: db
spec:
  replicas: 2
…
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: db
        image: mongo
        ports:
        - containerPort: 27017
---
apiVersion: apps/v1
kind: Service
metadata:
  name: mongo-service
  labels:
    app: mongo
```

Reference MSA Projects

- Online Commerce Example:
  https://github.com/TheOpenCloudEngine/uEngine-cloud/wiki

- Commercial Product: uEngine 5 BPMS:
  https://github.com/uengine-oss/uEngine5-base

- CQRS and Spring 2.0:
  http://www.kennybastani.com/2016/04/event-sourcing-microservices-spring-cloud.html

THANKS!

Any Question?

You can find me at:
jyjang@uengine.org
https://github.com/jinyoung
https://github.com/TheOpenCloudEngine