

Developing Cloud Native Applications

3rd Jan 2020, ver2.0



Agile 의 정의

Planning is important!

- 계획 중심
- 실패는 나쁜 것
- 비용 중심 사고

What Most People Think



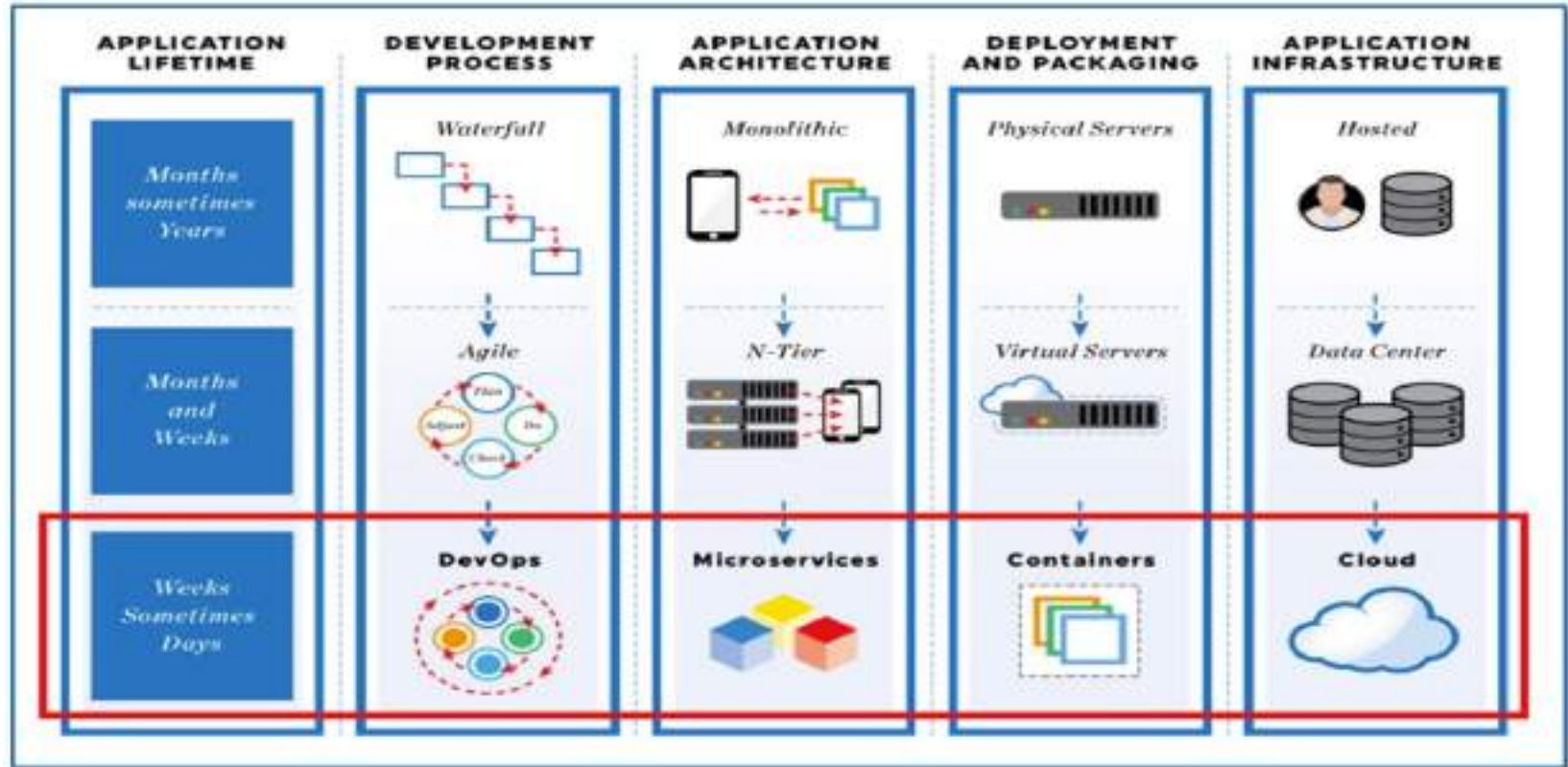
What Successful People Know



Agility is important

- 성장 마인드셋
- Fail Cheap, Fail Fast, Fail Often
- 고객 중심 사고

Agile 에 필요한 것들



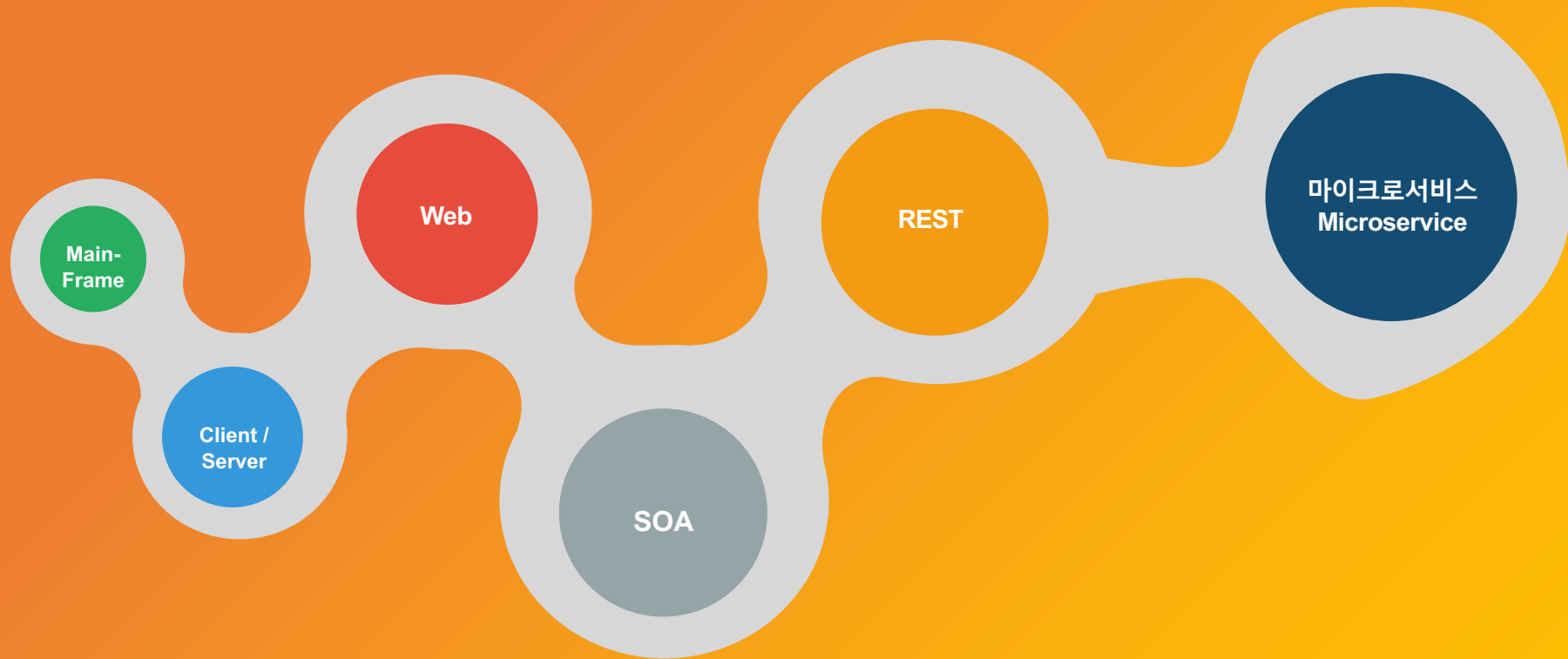
제프 베조스의 의무사항

”



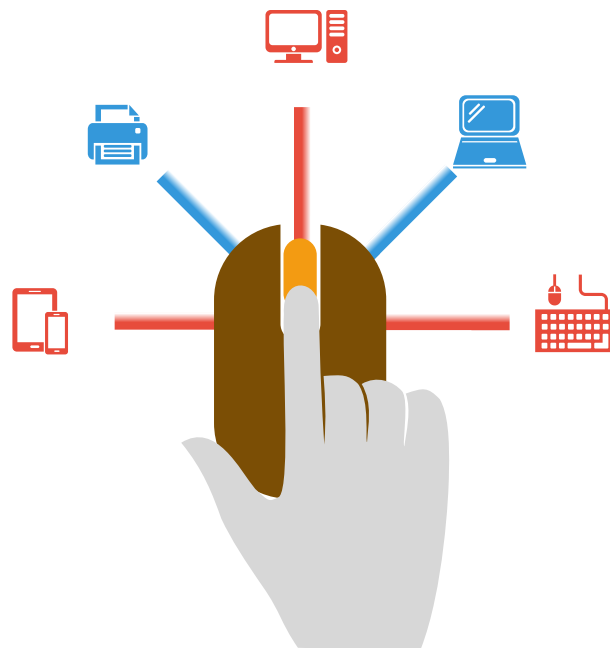
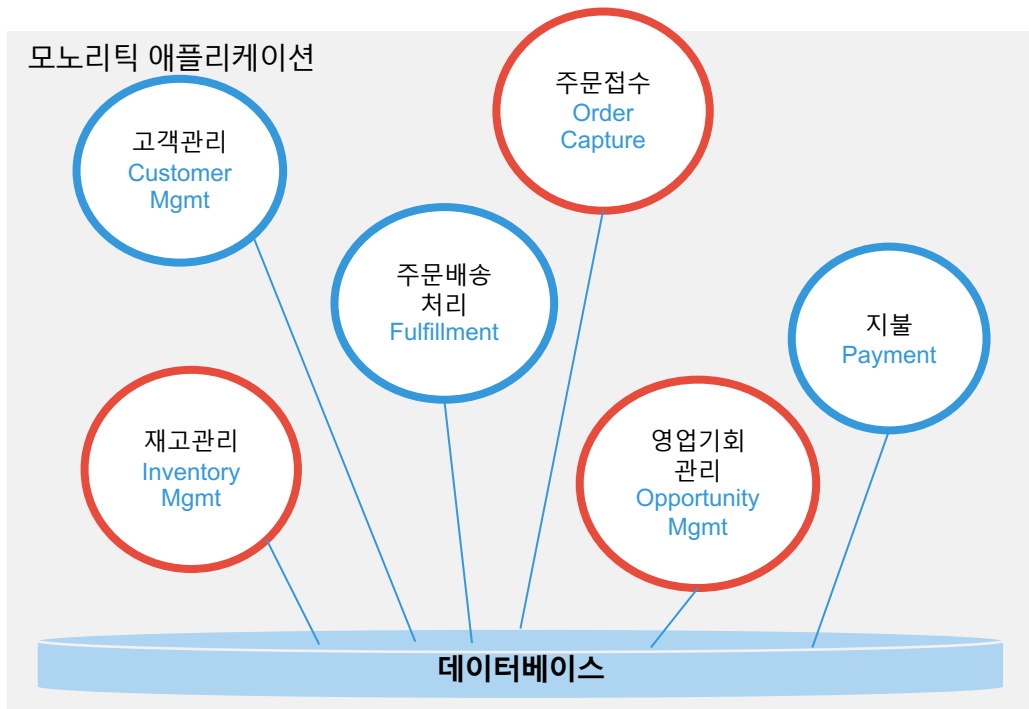
1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. 다음과 같은 그 어떠한 직접적 서비스간의 연동은 허용하지 않겠다:
no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors 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.

소프트웨어 아키텍처의 성장 여정



첫번째, 마이크로서비스가 아닌 것: 모노리틱 아키텍처 (A Monolithic Architecture)

An Enterprise Application or Suite



모노리틱 아키텍처의 분석

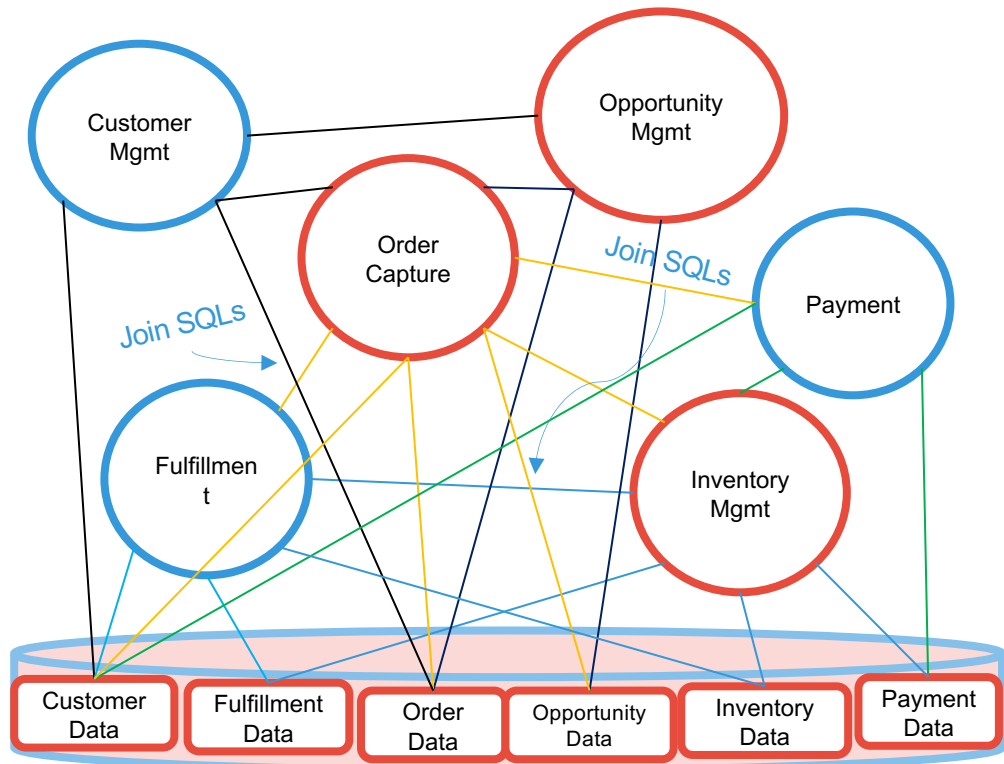
상호 데이터 참조가 용이하여 빨리 개발하기 위한 초기 아키텍처로 적합

그러나, 쉬운 상호연동은 상호의존성을 높힘
But, ease of interaction results in many inter-dependencies

시간이 갈수록, 커플링(의존성)은 강해지고 강해짐
Over time, coupling becomes tighter and tighter

하나의 컴포넌트를 수정하는 일은...
e.g. Order Data Table 의 field 변경

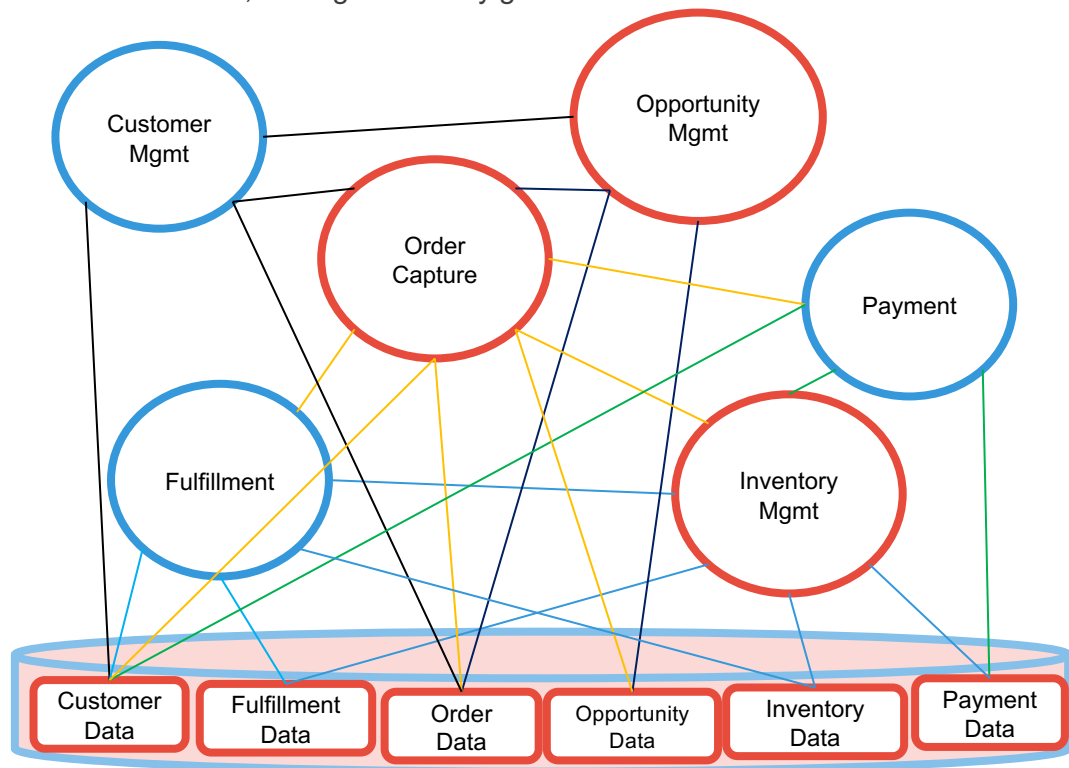
많은 다른 컴포넌트의 수정을 동반하게 됨
e.g. 다른 모듈의 Join SQL 들



모노리틱 아키텍처의 단점

Drawbacks of a Monolithic Architecture

Size matters, costs grow as they grow



코드량이 방대함 (오류발생시 바닷가에서 바늘 찾기)
Large code base

개발환경이 무거움 (IDE, WAS 등)
Overloaded IDE and development environment
(Web container, etc.)

하나의 변경이 나머지의 모든 재배포를 유발함
Deployment of any change requires redeploying
everything

선별적 확장이 용이하지 않음
Only scales in one dimension

하나의 기술 스택 만을 선택 가능 e.g. Java 1.8 + Oracle
You are committed to the technology stack

새로운 개발팀을 추가하는데 어려움이 있음
Becomes an obstacle to scaling development

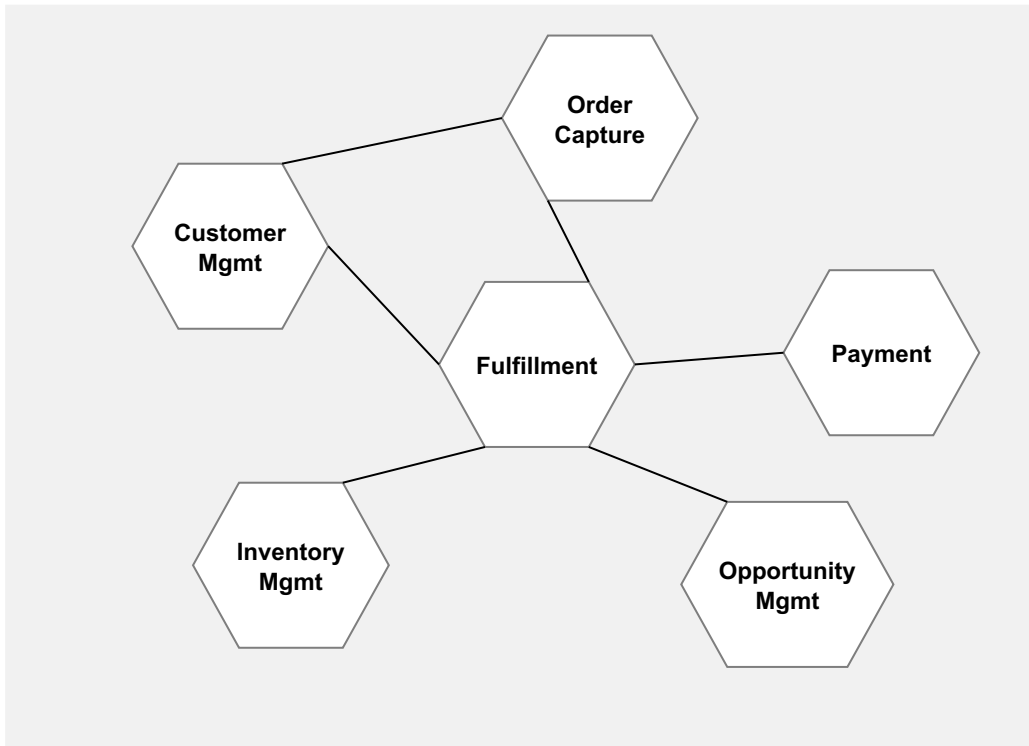
반면: A Microservice Architecture

Service-oriented architecture of loosely coupled elements with bounded contexts

모든 기능을 각각의 배포 스택으로
분리

Break each function into separate
deployment stacks

- Separate database
- Separate Servers running any technology
- Local or wide-area network



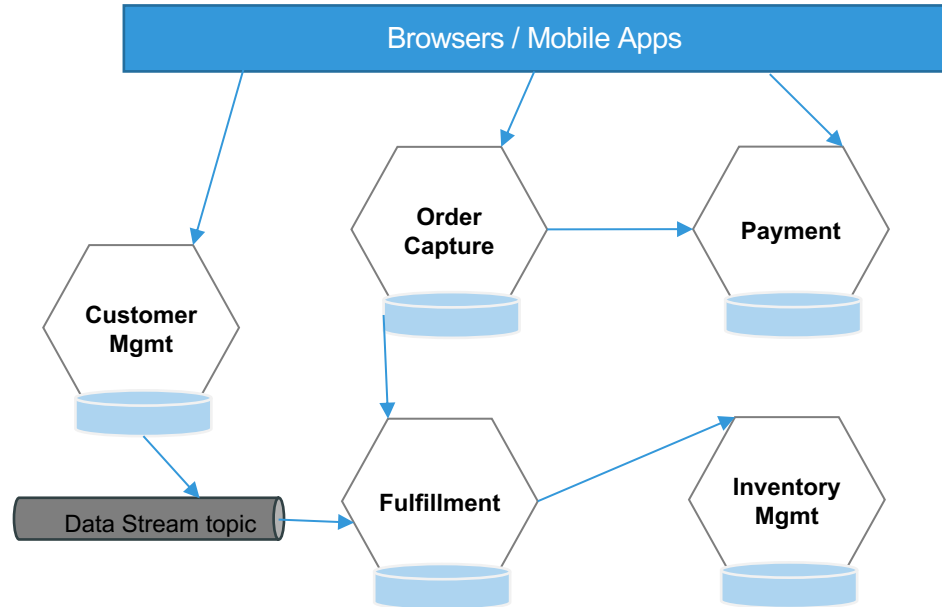
Microservice Architecture

Isolation is the name of the game

Service-oriented architecture

Loosely coupled elements

- Interaction only through HTTP/REST
- Or asynchronous streaming / messaging



Principles of Microservices

Isolation is the name of the game

독립성과 자치성을 코드의 재사용성 보다 높게 본다

Independence and autonomy are more important than code re-usability

마이크로 서비스는 코드와 데이터를 공유하지 않는다

Microservices should not share code or data

불필요한 서비스와 소프트웨어 컴포넌트(라이브러리) 간의 커플링을 피한다

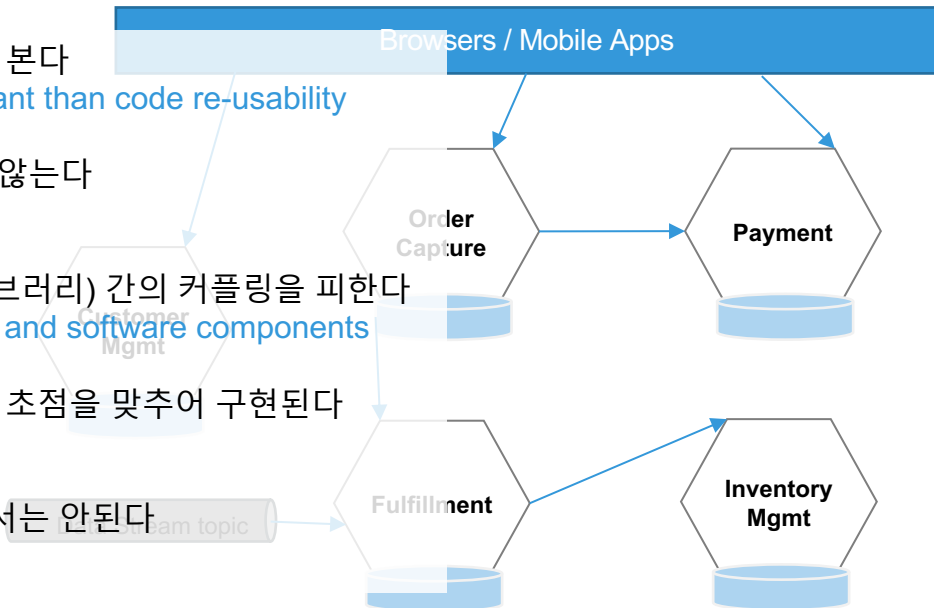
Avoid unnecessary coupling between services and software components

각 마이크로서비스는 각자의 단 하나의 기능에 초점을 맞추어 구현된다

Single responsibility

운영시에는 SPOF (단일 실패 지점) 을 만들어서는 안된다

There should be no single point of failure



HTTP/REST 를 이용한 약결합 연동

(Loosely-Coupled Interaction via HTTP/REST)

REST APIs must be stable and hide internals

클라이언트-서버 REST API 는 내부구현을 숨길 수 있으면서 연동

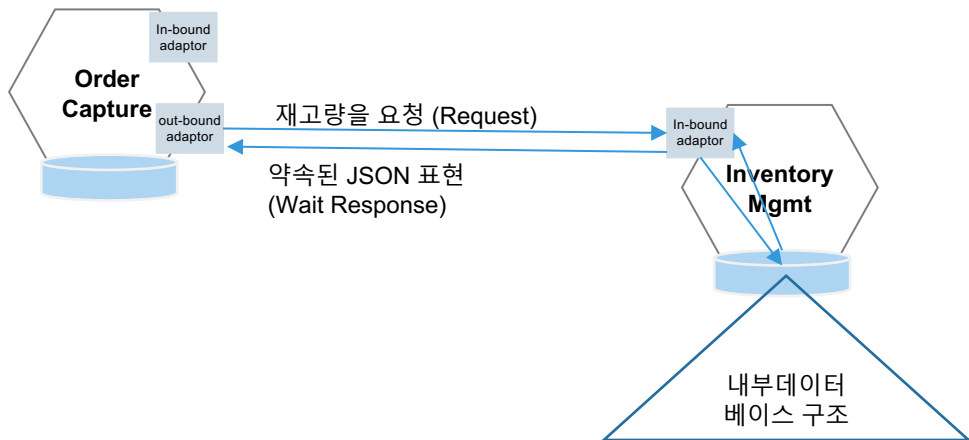
Client-oriented REST APIs hide the internal implementation of the service

동기식 연동 (Synchronous, Request-Response)

SOAP 에 비하여 REST 는 API 정의 및 관리 비용이 낮으나, 각 클라이언트에 대한 요청에 충분한 API 를 정의하고 관리하는 비용이 높아질 수 있음

API 는 하위호환성을 유지하기 위하여 추가적인 수정만을 허용

Only additive changes allowed



→ 빌드타임에서만 약결합을 제공함

비동기 메시지를 통한 약결합 연동

Loosely-Coupled Interaction via Asynchronous Messaging

Data streaming can decouple database with shared data

상대의 수신을 기다리지 않는 비동기식 메시징
Asynchronous messaging(streaming)

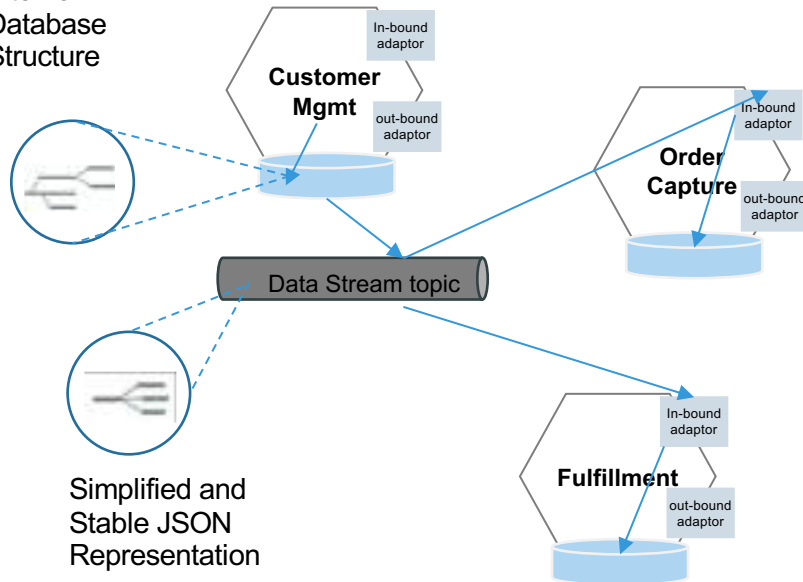
- Publish-Subscribe
- 준실시간으로 동작함
- **Decouples “timing” = Non-blocking**

Data streams hide the internal implementation of the service

Client ignore parts of the stream they don't understand

➔ 런타임에서도 약결합을 제공함

Internal Database Structure



Simplified and Stable JSON Representation

Microservice Architecture benefits

Smaller is better

작은 코드량은 이해하기 쉽고, 오류를 찾기 쉽다
(버그가 살기 힘든 공간 = 버그가 숨을 공간이 적다)

Smaller, independent code base is easier to understand

수정된 서비스에 대한 국지적 단위 디플로이와 테스트가 용이해진다

Simpler deployment and testing of just the changed service

장애격리가 좋아진다

Improved fault isolation

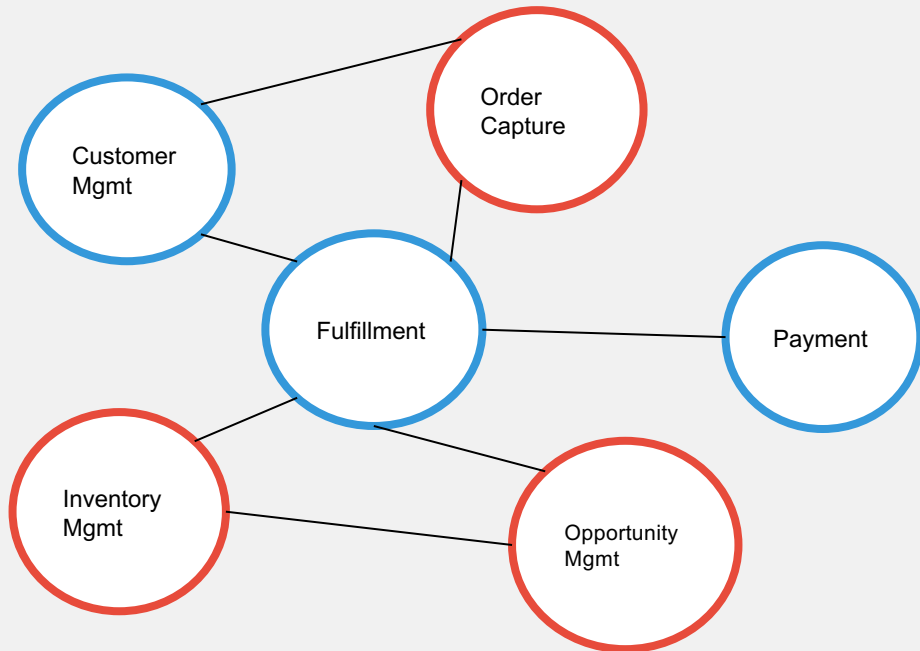
여러가지 혼재된 기술 스택 (신기술)을 사용하기 쉽다

Not committed to one technology stack

개발환경을 구축하기 쉽고 빨리 기동된다

IDE and app startup are faster

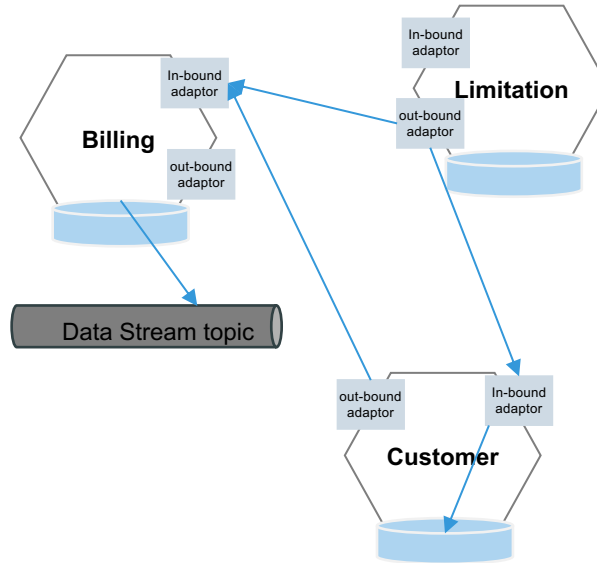
Monolithic



Microservice Architecture

✓ Updating one service doesn't require changing others

✓ Ability to upgrade the tech stack (HW/SW/DBMS/NW) Of each service independently



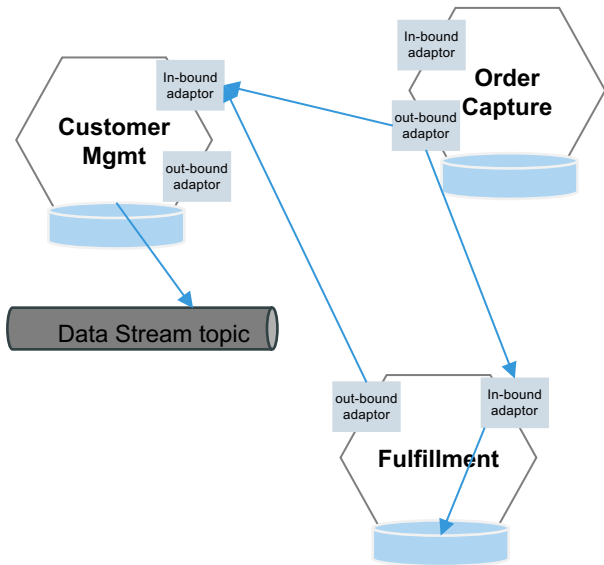
✓ Good fault isolation

✓ Smaller, simpler code base

✓ Options for scaling

Drawbacks to a Microservice Architecture

Distributed computing adds complexity and slow down initial development



개발도구들이 아직 최적화되지 않았다

Dev tools not optimized for distributed services

전체적인 테스트가 복잡해진다

Testing can be more complicated

운영과 디플로이가 복잡해져
쿠버네티스 와 같은 DevOps
환경이 필수적이다

Deployment and operations are more complex

서비스를 어떻게 쪼갤 것인가?

Where/How to decompose the services?

서비스간 연동을 통해서만 구현하는
비용의 상승

Inter-service communication complicates development

분산트랜잭션 (데이터 일관성 등)을
어떻게 보장할 것인가?

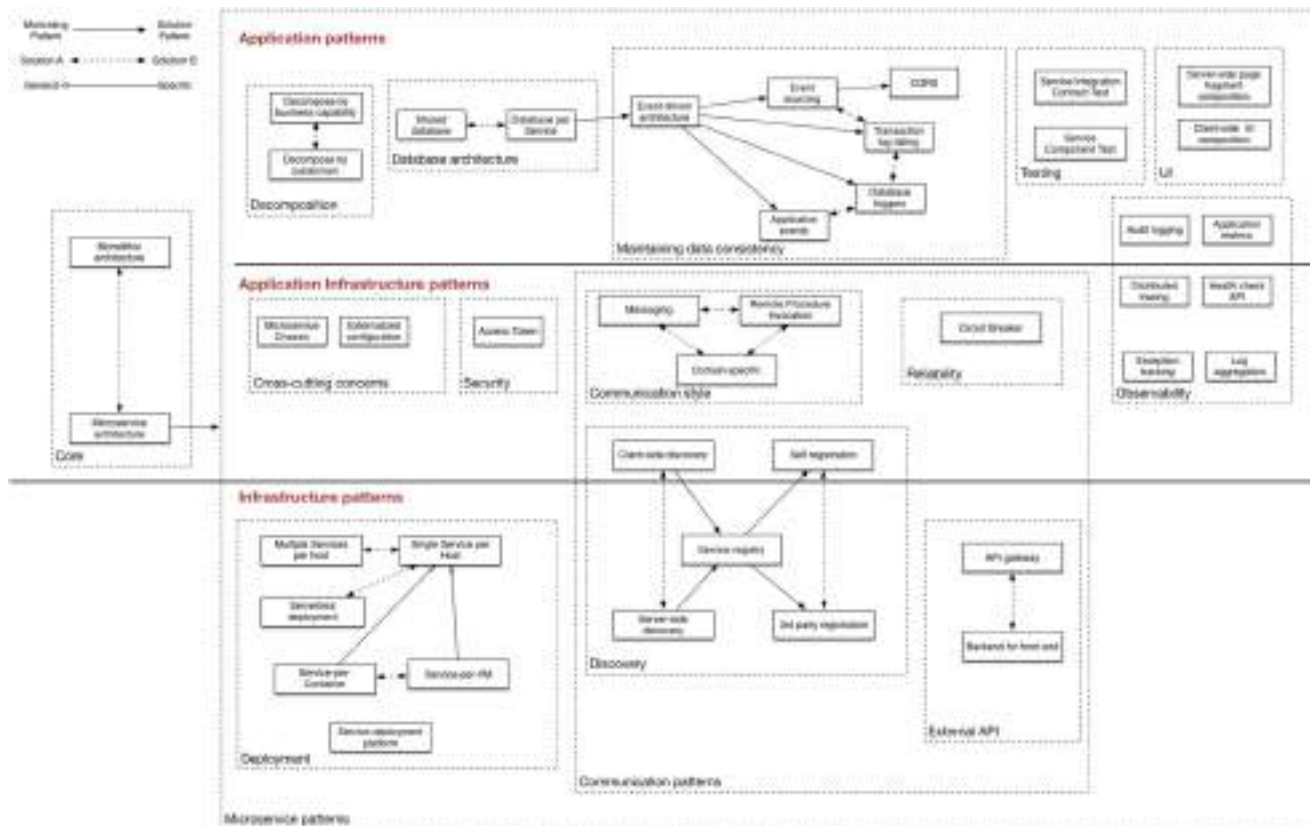
Maintaining consistency with distributed transactions is hard

서비스 개수가 많아진 상황의
보안처리를 어떻게 할 것인가?

What about inter-service security?
Identity management?

해결책: MSA 디자인 패턴

- microservices.io



마이크로 서비스 전환 사례



Video & Broadcasting



Mobile Apps and
Serverless Microservices



Pure Play Video OTT- A



From Monolithic to
Microservices



Gaming Platform



IoT Service



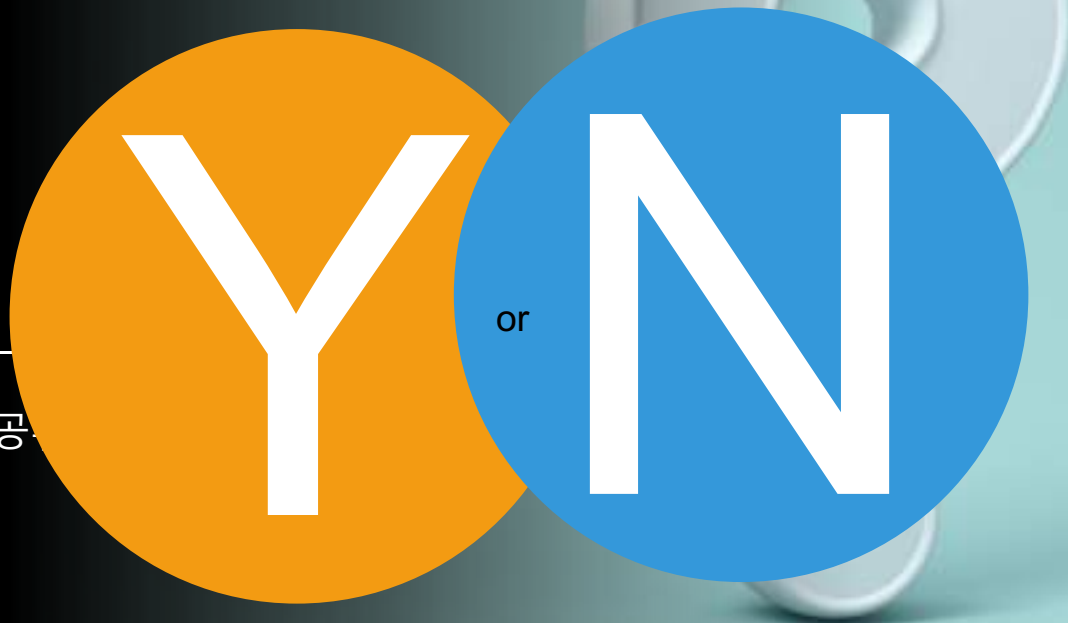
Serverless Microservices

당근마켓

<https://youtu.be/mLlthm96u2Q?t=50>

Quiz

마이크로서비스간에 코드를 공유하는 것은 좋은 사례이다



Quiz

마이크로서비스가 실패할 때는 독립적으로 실패해야
한다

Y

or

N



Quiz

• 왜 단일 Database 접근은 마이크로서비스에서 안티-패턴 (하지 말아야 할 접근) 인가?

1. 마이크로서비스는 관계형 데이터베이스 (R-DBMS) 와 호환되지 않기 때문이다
2. 하나의 데이터베이스만 사용했을 때는 이것이 실패단일지점 (Single Point of Failure)가 될 수 있기 때문이다
3. 단일 데이터베이스 시스템은 보통 큰 시스템을 만들 때만 쓰이기 때문이다.

목표수준수립과 비용



늘어나는고객수를
처리하기 위한 확장

- Core/Supporting 분리
- 컨테이너 적용
- 팀의 확장
- 팀 자율성



팀자율성 개선

- 팀수준 (KPI or Business Capability) 수준의 분리



새로운 언어와
기술의 도입

- 컨테이너 기술



시스템 작동 중단
시간 감소

- Core/Supporting 의 분리
- 무정지 재배포
- 컨테이너 기술

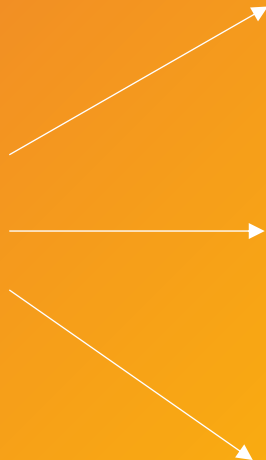
Table of content

Microservice and
Event-storming-Based
DevOps Project

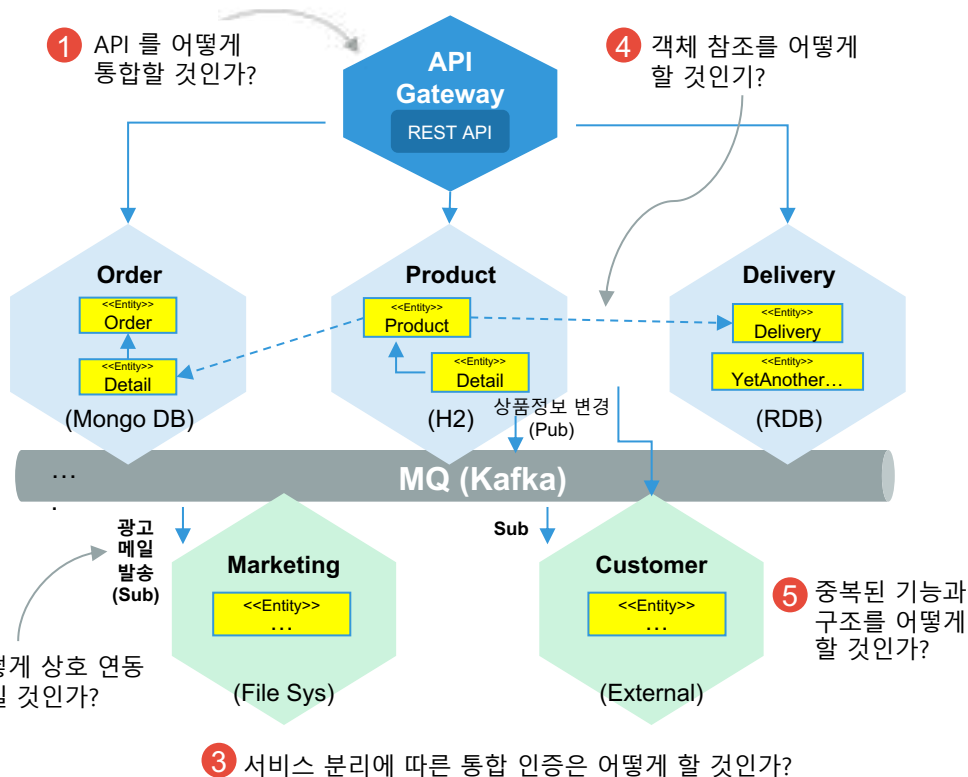
1. The Domain Problem : A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices ✓
6. Front-end Development in MSA
7. Service Composition with Request-Response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

“ From Monolith to Microservices

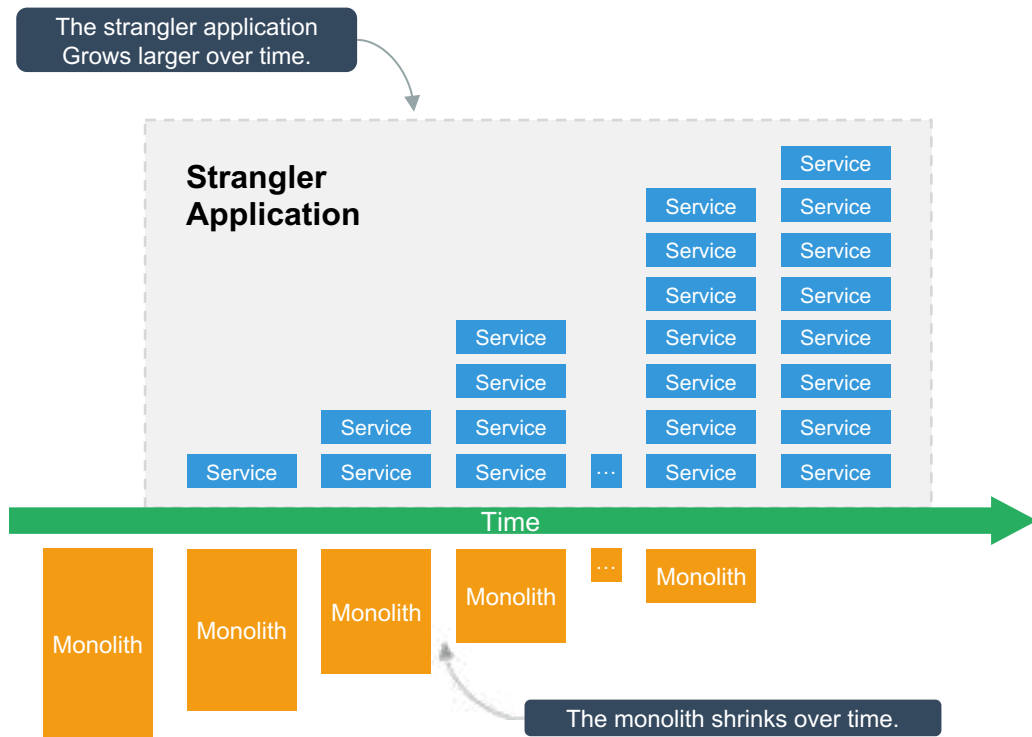
<https://github.com/event-storming>



Issues in transforming Monolith to Microservices



Legacy Transformation – Strangler Pattern



- Strangler 패턴으로 레가시의 모노리스 서비스가 마이크로 서비스로 점진적 대체를 통한 Biz 임팩트 최소화를 통한 구조적 변화
- 기존에서 분리된 서비스 영역이 기존 모노리스와 연동 될 수 있도록 해주는 것이 필요

1. API 를 어떻게 통합할 것인가?

- API Gateway

진입점의 통일

URI Path-based Routing (기존에 REST 로 된경우 가능)

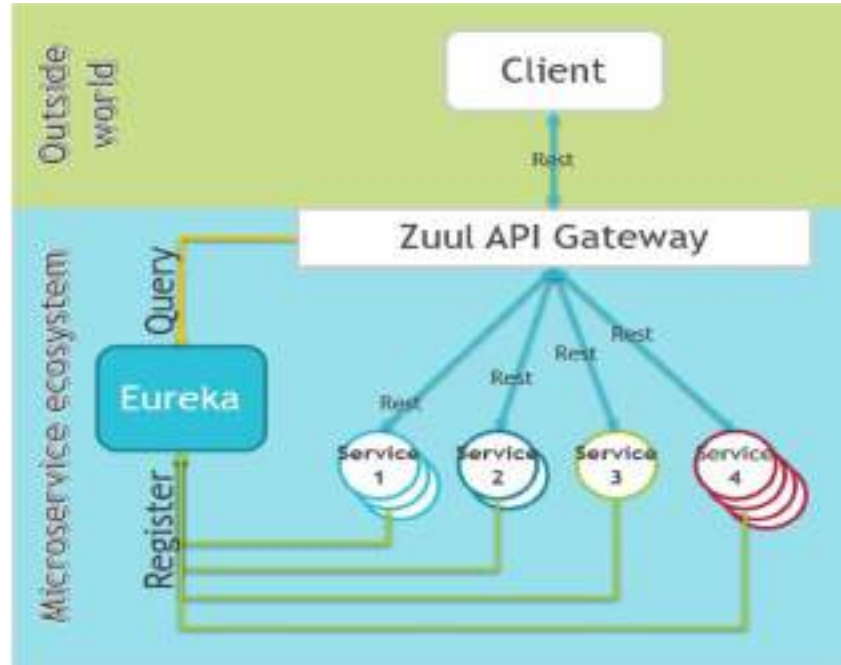
- Service Registry

API Gateway 가 클러스터 내의 인스턴스를 찾아가는 맵

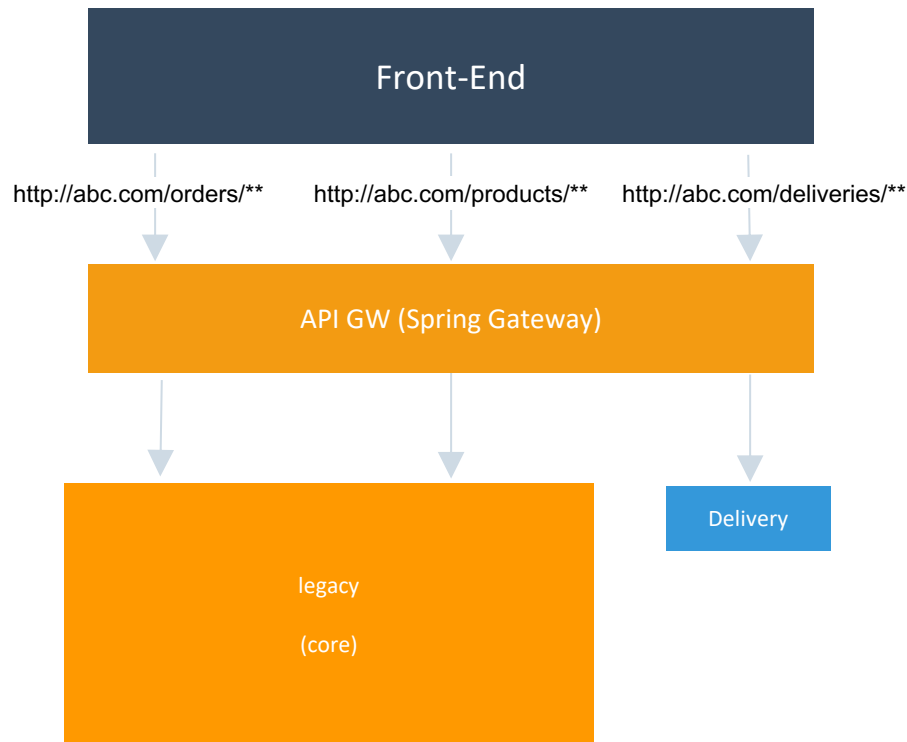
API Gateway : Edge Service

Acting like “Skin” to access our services :

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



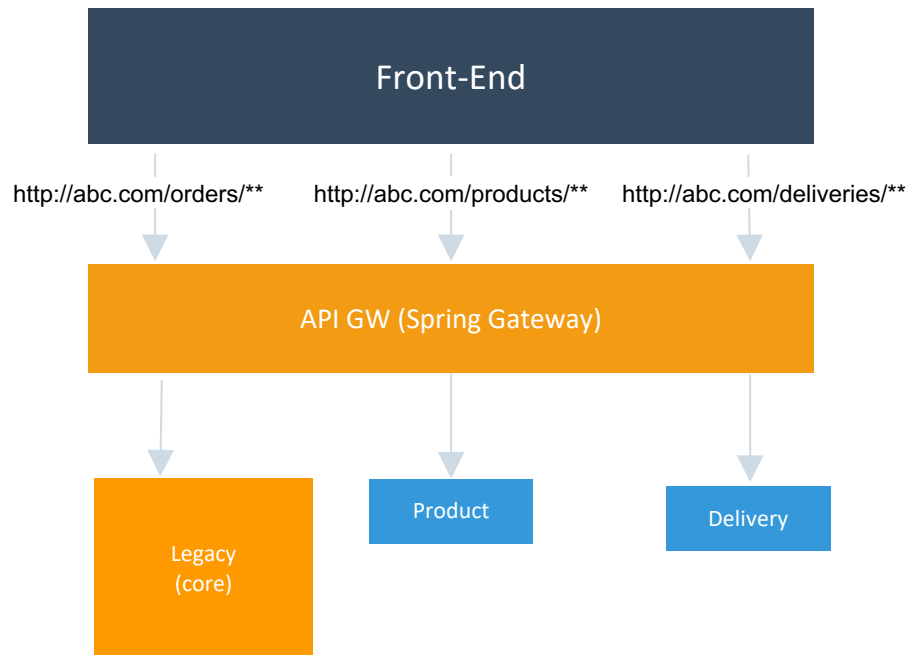
API Gateway : Strangler



#application.yml

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

API Gateway : Strangler



#application.yml

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

Service Registry: EUREKA or Kube-DNS



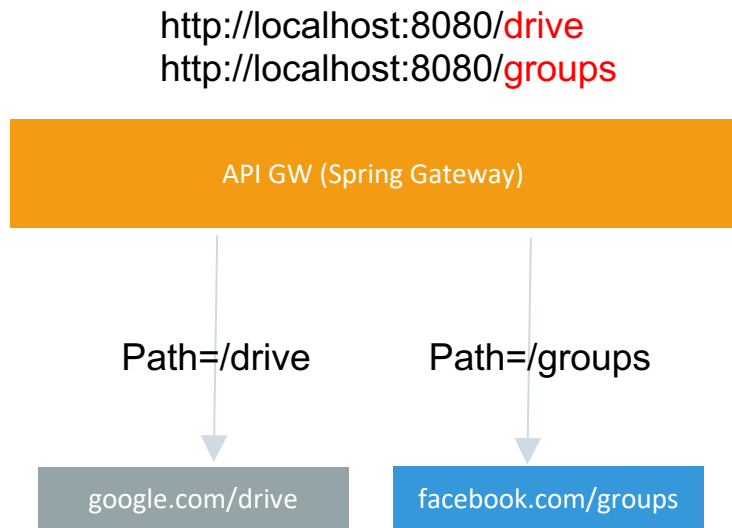
The screenshot shows the Spring Eureka web interface. At the top, there is a header with the 'spring Eureka' logo on the left, and 'HOME' and 'LAST 1000 SINCE STARTUP' links on the right. Below the header, the text 'Instances currently registered with Eureka:' is displayed. Underneath this text is a table with four columns: 'Application', 'AMIs', 'Availability Zones', and 'Status'. The table contains three rows of data:

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:8081, 192.168.0.47 definition:8086
ZUUL-ROUTER	n/a (1)	(1)	UP (1) - 192.168.0.47 zuul-router:8080



Lab: API Gateway – Path-based routing (2/3)

- Route : 게이트웨이의 기본 블록 구성이다. ID, Url , Predicate, Collection of Fillers 로 이루어져 있다. Predicate 가 일치할 때 Route 가 true 로 인식을 한다
- Predicate : 조건부 – 헤더, 파라미터등의 http 요청을 매치

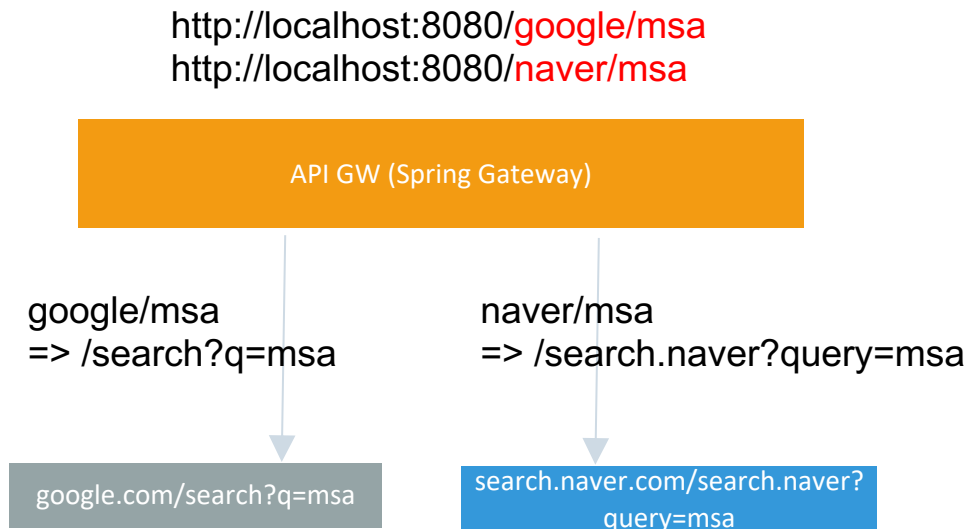


spring:
cloud:
gateway:
routes:

- **id:** google
uri: http://google.com
predicates:
 - Path=/drive
- **id:** facebook
uri: http://facebook.com
predicates:
 - Path=/groups

Lab: API Gateway – Filter (3/3)

- Filter : requests and responses 를 downstream 으로 요청을 보내기 전후에 수정이 가능하도록 구성
- URL 에 따라 다른 검색엔진에서 'msa' 검색

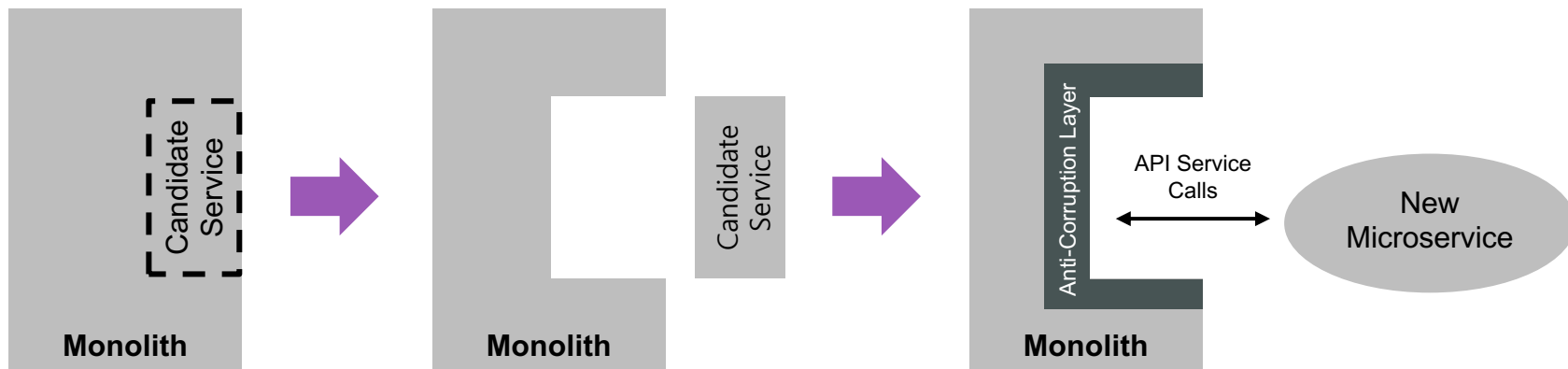


```
spring:
  cloud:
    gateway:
      routes:
        - id: google
          uri: http://google.com
          predicates:
            - Path=/google/{param}
          filters:
            - RewritePath=/google(?<segment>/?.*),/search
            - AddRequestParameter=q,{param}
```

2. 어떻게 (다시) 상호 연동시킬 것인가?

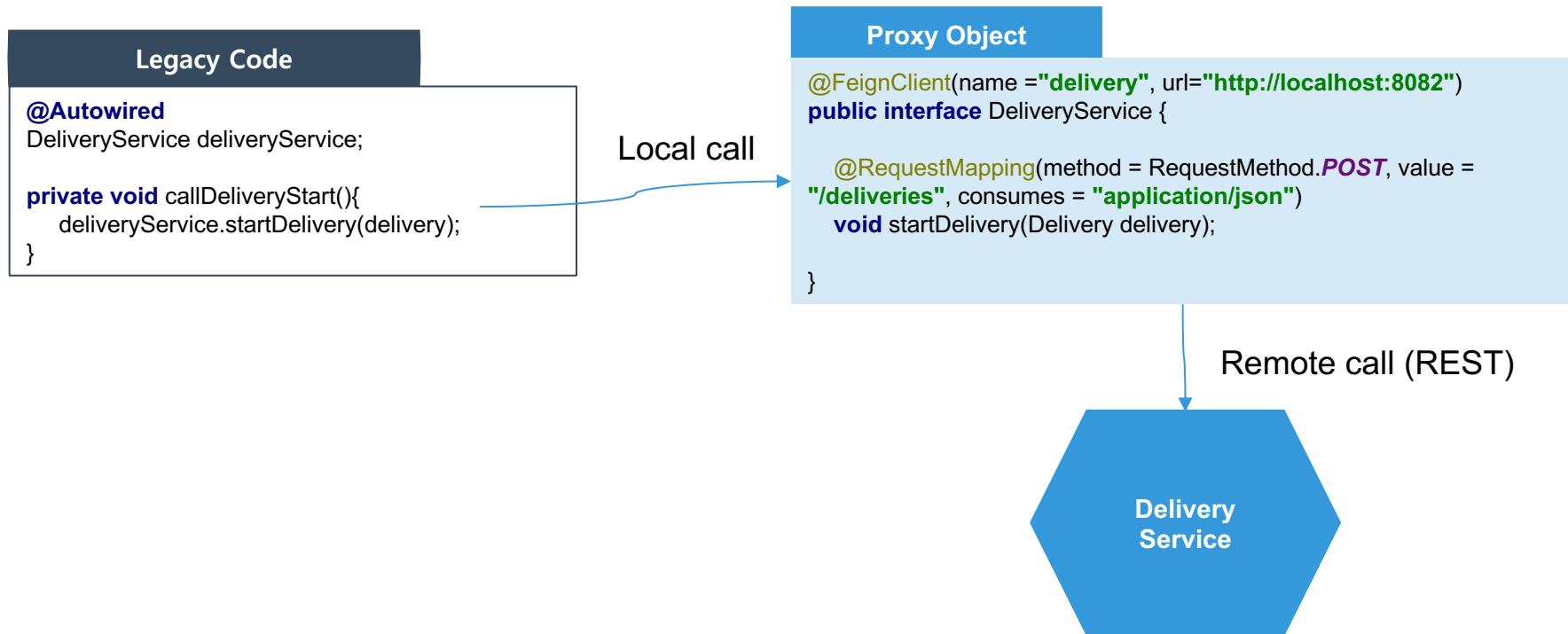
- Find the seams and replace with proxy
기존 연계되던 이음매 객체 (interface) 를 찾아, 그에 상응하는 Façade, Proxy 로 대체
- Event Shunting
레가시에서 이벤트를 퍼블리시하도록 전환, 신규 서비스가 주요 비즈니스 이벤트에 대하여 반응하도록 처리

Find the seams and replace with proxy

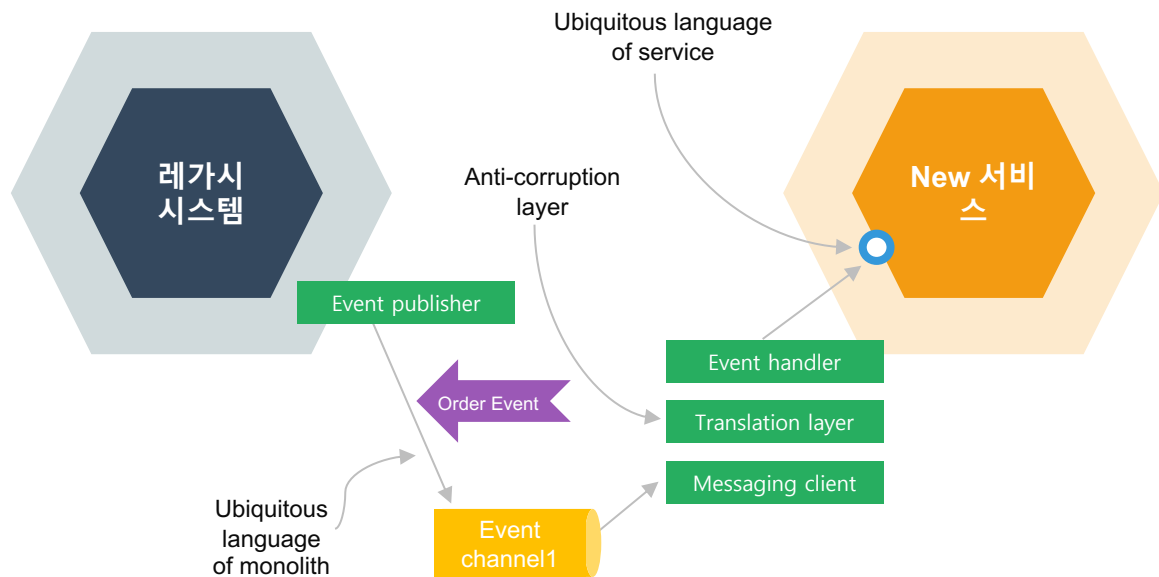


- Determine candidate service from Monolith and strangle it out
- Convert candidate service to Microservice and Add Anti-Corruption Layer to enable communication with Monolith

Proxy using FeignClient



Event Shunting



- **Aggregate 내 코드 주입**

호출 코드 직접 주입, Hexagonal Architecture의 손상
JPA의 Lifecycle Annotation 사용

- **CDC (Change Data Capturing) 기능 사용**

DB의 Change Log를 Listening, Event 자동퍼블리시 하는 툴
Debezium, Eventuate Tram 등이 존재

Lab: Proxy with FeignClient (2)

- 기본 소스코드 다운로드
 - git clone <https://github.com/event-storming/monolith.git>
 - git clone https://github.com/event-storming/reqres_delivery.git
- Monolith 의 Local 호출을 Proxy 호출로 전환 과정
 - DeliveryServiceImpl.java 파일의 내용을 모두 주석처리
 - pom.xml 에 feignclient 디펜던시 추가
 - Application.java 에 @EnableFeignClients 를 선언
 - DeliveryService.java 파일에 feign-client 추가
(참고 - https://github.com/event-storming/reqres_orders/blob/master/src/main/java/com/example/template/DeliveryService.java)
 - @FeignClient(name = "**delivery**", url="http://localhost:8082")
- 기존의 local 객체 참조를 ID 참조로 전환
 - Monolith 프로젝트의 Order.java 와 Delivery.java 의 @OneToOne 관계 제거
 - delivery.setOrder(**this**); 를 delivery.setOrderId(this.getId()); 로 변경

Lab: Proxy with FeignClient (3)

- 주문 하기
http localhost:8088/orders productId=1 quantity=3 customerId="1@uengine.org" customerName="홍길동" customerAddr="서울시"
- 배송확인
http http://localhost:8088/deliveries
http http://localhost:8082/deliveries
- 주문 후 변경된 상품 수량 확인
http <http://localhost:8088/orders/1/product>
- 프로젝트 원복
git reset --hard

3. 서비스 분리에 따른 통합인증은 어떻게 할 것인가?



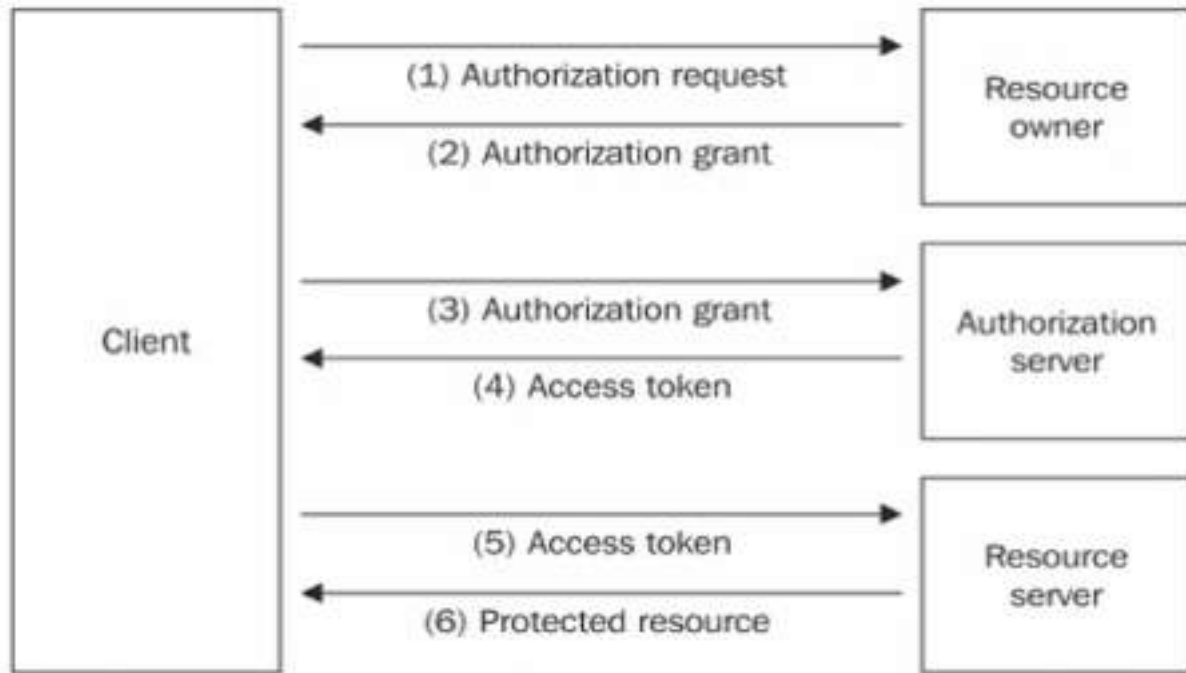
- **OAuth2.0**

웹, 모바일 어플리케이션에서 타사의 API 권한 획득을 위한 프로토콜
Google, facebook 등을 통한 인증 위임

- **JWT(Json Web Token) 토큰**

Header, Claim Set, Signature로 구성
요청 헤더에 Authorization 값을 담아서
서버로 송신

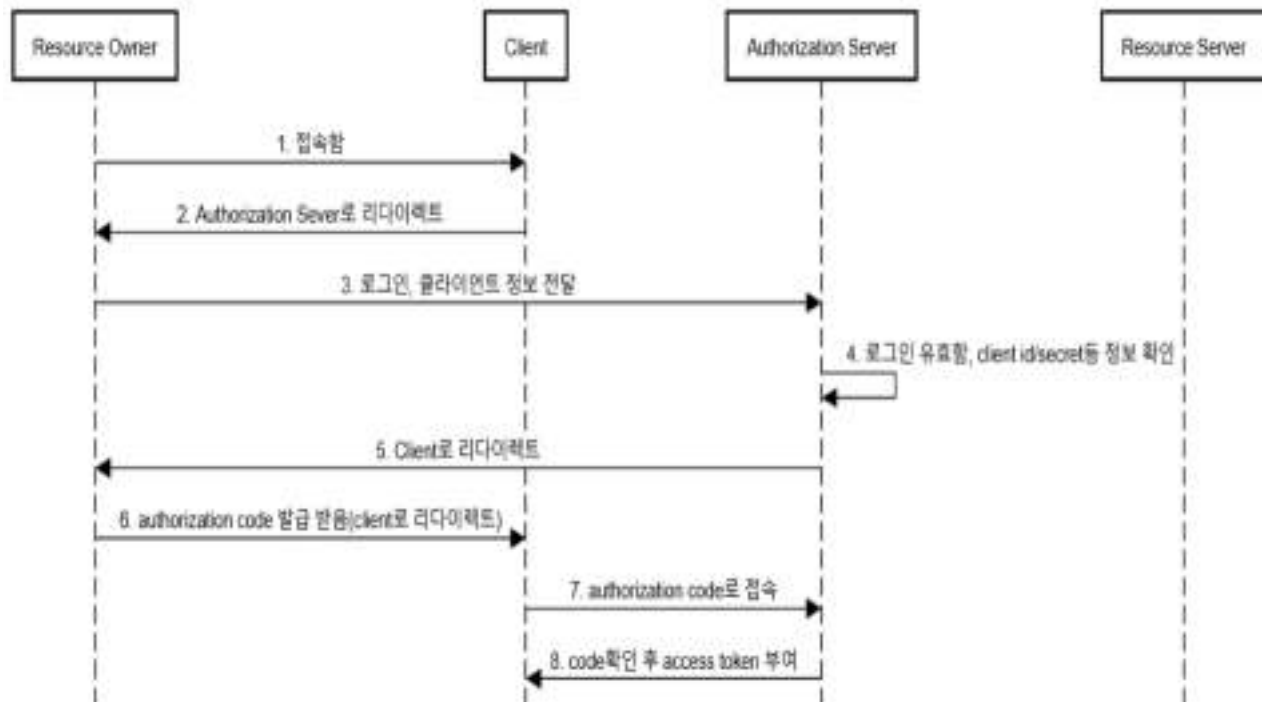
OAuth 2.0: Authorization Flow



1. 클라이언트가 자원 소유자에게 권한 요청
1. 자원 소유자가 권한을 허가시, 클라이언트는 권한 증서를 발급받음
1. 클라이언트는 권한증서를 가지고 토큰을 권한 서버에 요청
1. 권한증서의 유효성을 체크하고 토큰을 발급해줌
1. 클라이언트는 토큰을 사용하여 자원 요청
1. 토큰 유효성 확인후 요청 처리

OAuth 2.0: Grant Type – Authorization code (1/4)

Authorization Code Grant Type



1. 구글, 페이스북, 카카오등 유저 정보가 다른 시스템에 있을때 사용하는 방식

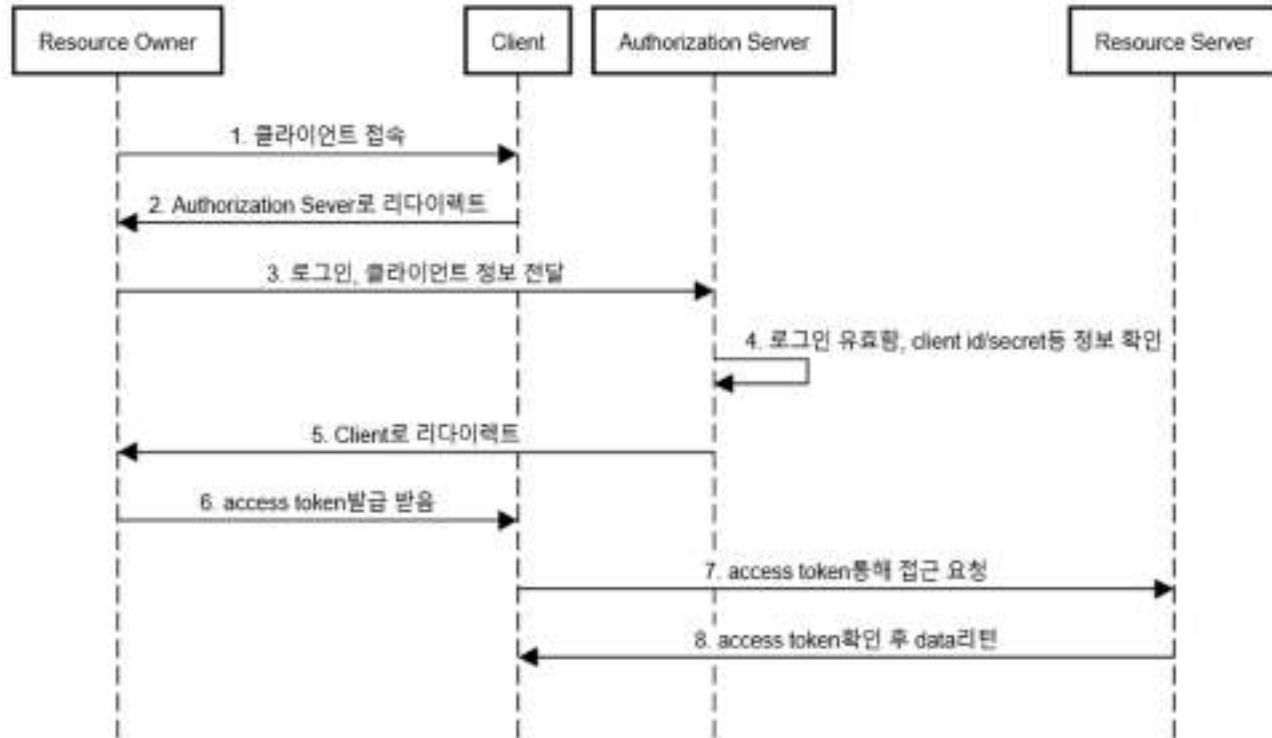
1. 어플리케이션이 인증서버에 요청해 브라우저를 열어서 사용자가 인증을 진행하게 하는 방식으로 사용

1. 토큰요청시 코드를 요청하는 단계가 있어서 보안에 효과적

1. 가장 복잡하지만, 가장 많이 쓰임 > 개발자만 고생하면됨

OAuth 2.0: Grant Type – Implicit (2/4)

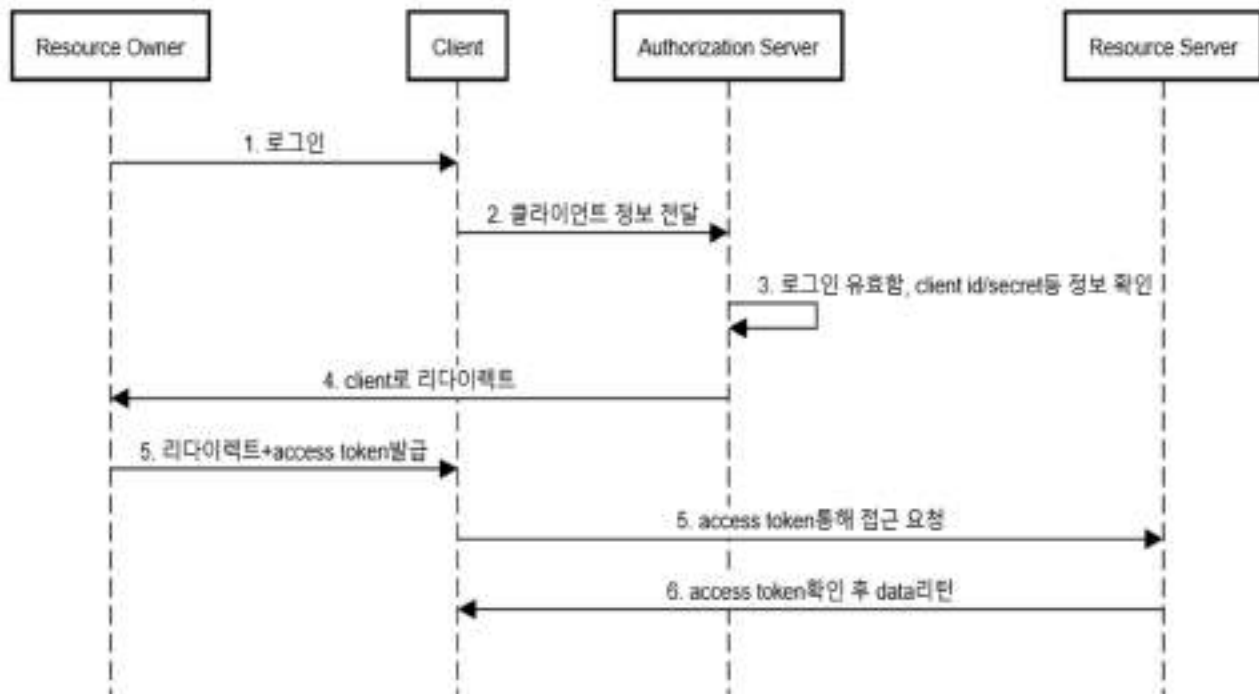
Implicit Grant Type



- 1. Authorization_code 방식에서 코드를 요청하는 프로세스를 제거하고, 바로 토큰을 return
- 1. Javascript 로 동작하는 SPA(Single Page Application) 에서 사용하기 위해 만들어 졌으나 권장하지 않음
- 1. 신뢰성 있는 앱 또는 단말기에서 사용
- 1. 외부에 있는 Oauth 서버가 cors 를 지원하지 않을때 사용

OAuth 2.0: Resource Owner Credential (3/4)

Resource Owner Credential Grant Type(password grant)



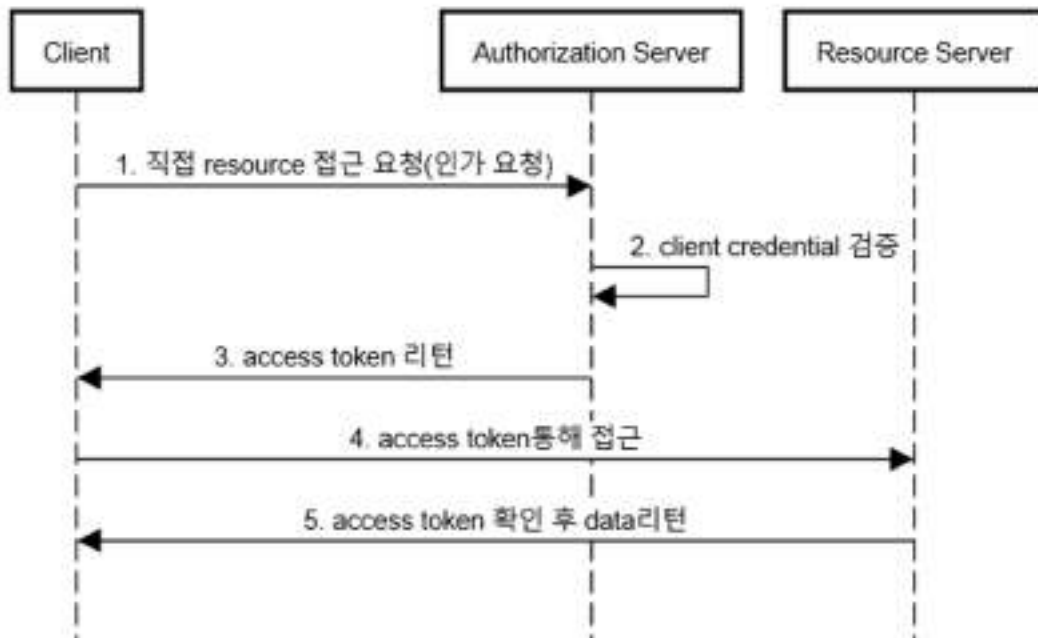
1. 타사의 인증 서버를 사용하지 않고, 자신의 서비스에 인증시 사용 (자신이 유저정보를 가지고있고, 내 서비스만 인증할때)

1. OAuth 2.0 의 가장 간단한 인증 중 하나

1. 전통적으로 이름과 비밀번호로 인증

OAuth 2.0: Client Credential (4/4)

Client Credential Grant Type



1. 사용자가 아닌 응용프로그램 (client) 이 인증을 요청할때 사용
(Resource Owner = Client)

1. 접근 권한이 한정되어있는 프로그램 사용시 활용

1. 신뢰성이 높은 관리자용 Desktop App 이나 Mobile App 에서 사용

Lab: OAuth Authorization (1/3)

- Local 환경에 Gateway, Oauth, UI 프로젝트 다운로드
- git clone <https://github.com/event-storming/oauth.git>
- git clone <https://github.com/event-storming/gateway.git>
- git clone <https://github.com/event-storming/ui.git>

- gateway, oauth
 - mvn spring-boot:run
- ui
 - npm install
 - npm run serve

- localhost:8080 접속

Lab: OAuth Based Authorization (2/3)

- UI 에서 토큰 위치 확인
 - localStorage
 - localStorage.accessToken
 - localStorage.getItem("accessToken")
 - <https://jwt.io/>
- API Call through gateway
 - http localhost:8088
 - http localhost:8088/orders "Authorization: Bearer \$TOKEN"
 - curl localhost:8088/orders --header "Authorization: Bearer \$TOKEN"
- JWT
 - 필요한 정보를 Token body 에 저장하여 사용자가 가지고 있고, 증명처럼 사용, header 에 실어 서버에 요청
 - 토큰을 변조 하더라도, SECRET_KEY 가 없으면 복호화를 못함
 - 참고 : <https://brownbears.tistory.com/440>

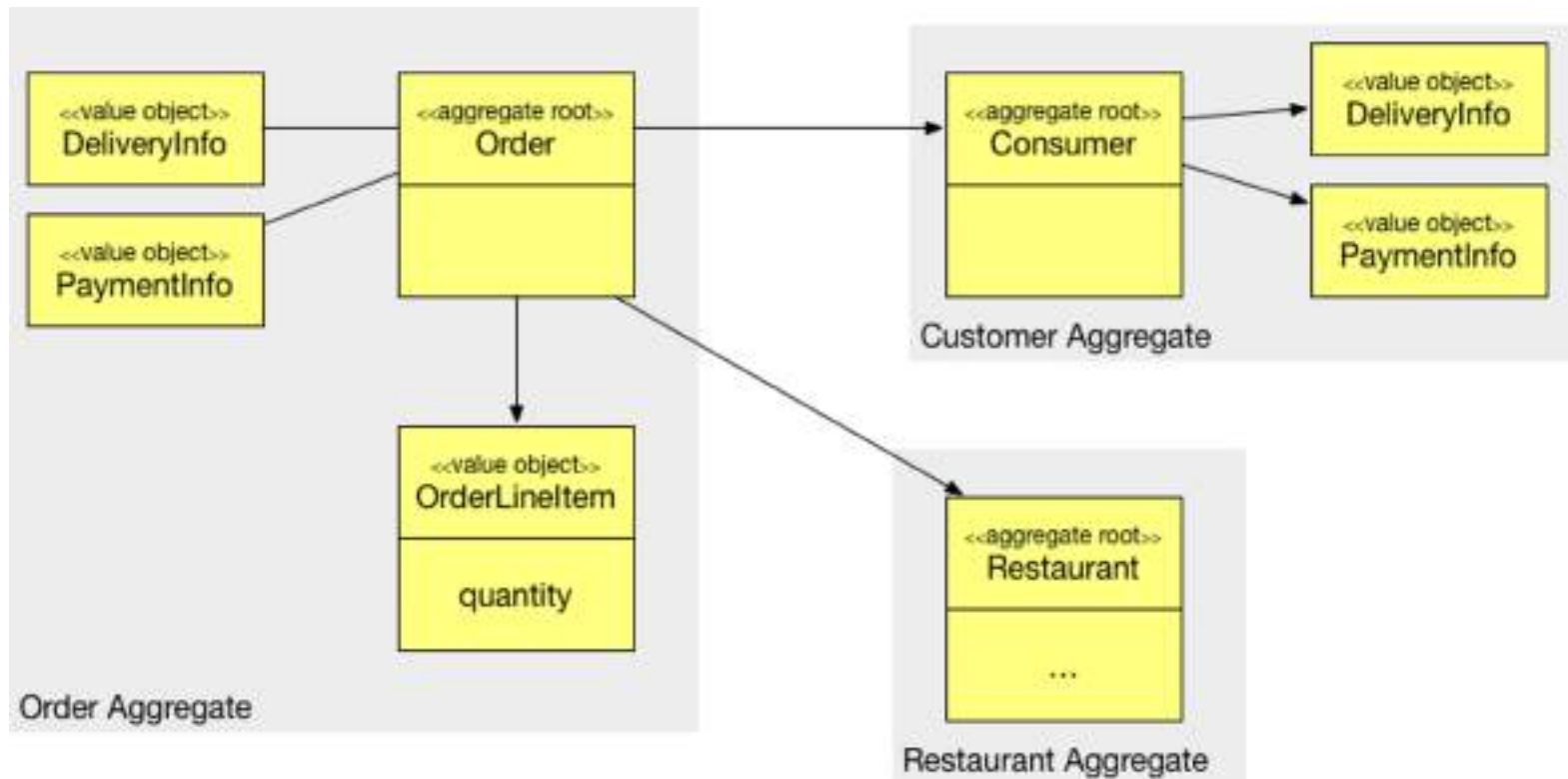
Lab: OAuth Based Authorization (3/3)

- 인증 서버에 토큰 요청- password grant
 - `http --form POST localhost:8090/oauth/token "Authorization: Basic dWVud2luZS1jbGllbnQ6dWVud2luZS1zZW5yZXQ=" grant_type=password username=1@uengine.org password=1`
- 인증 서버에 토큰 요청- client_credentials grant
 - `http --form POST localhost:8090/oauth/token "Authorization: Basic dWVud2luZS1jbGllbnQ6dWVud2luZS1zZW5yZXQ=" grant_type=client_credentials`

4. 객체 참조를 어떻게 할 것인가?

- 직접적 메모리 기반 객체 참조나 통합 DB 내의 Primary key 로는 불가능
- 분리된 Aggregate 내부의 Entity 간에는 Key 값으로만 연결
 - Primary Key 를 이용한 RESTful URI (Universal Resource Identifier) 로 연결
 - HATEOAS link 를 이용하여 JSON 내에 포함

Aggregate Root를 통한 객체 참조



URI 를 통한 객체 참조

- **HATEOAS** (Hypertext As The Engine Of Application State)

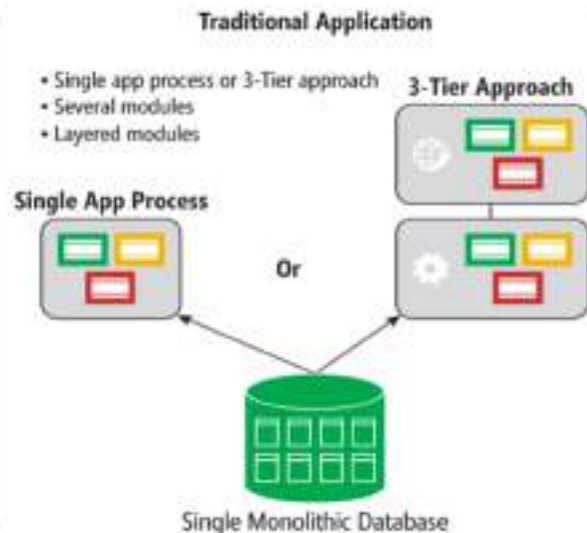
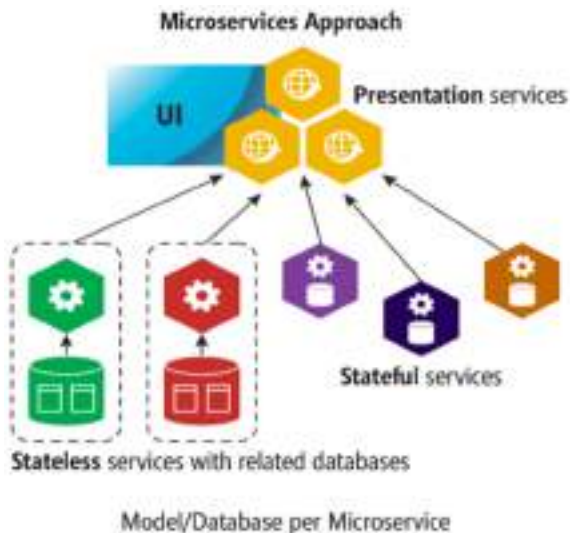
- HATEOAS is deemed the highest maturity level of REST.
(<https://martinfowler.com/articles/richardsonMaturityModel.html>)
- In the HATEOAS architecture, a client enters a REST application through a specific URL, and all future actions the client may take are discovered within resource representations 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.
}
```

Microservice Integration with by UI

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



5. 중복된 기능과 데이터를 어떻게 할 것인가?

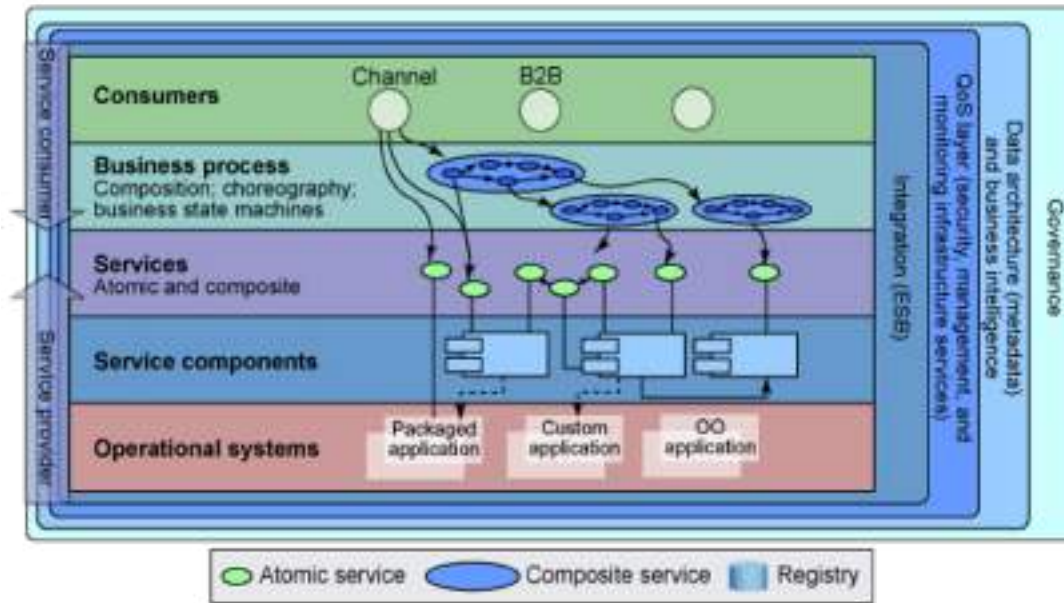
- 재사용하지 않고 중복 구현하는 것이 MSA스러운 것
재사용 통한 경제성(SOA사상, 디펜던시발생)과 자율적 창발 (낮은 간섭과 빠른 출시)의 트레이드 오프에서 후자의 전략을 선택
Utility service X, DRY(Don't Repeat Yourself) 룰 X
Polyglot Persistency
- 예외상황 : 데이터 참조에 intensive 한 서비스 (인증정보 등)는 Utility 서비스 성격으로 구현 가능함

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. 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

Integration Patterns



By UI

By Request-Response

By Event-driven Architecture

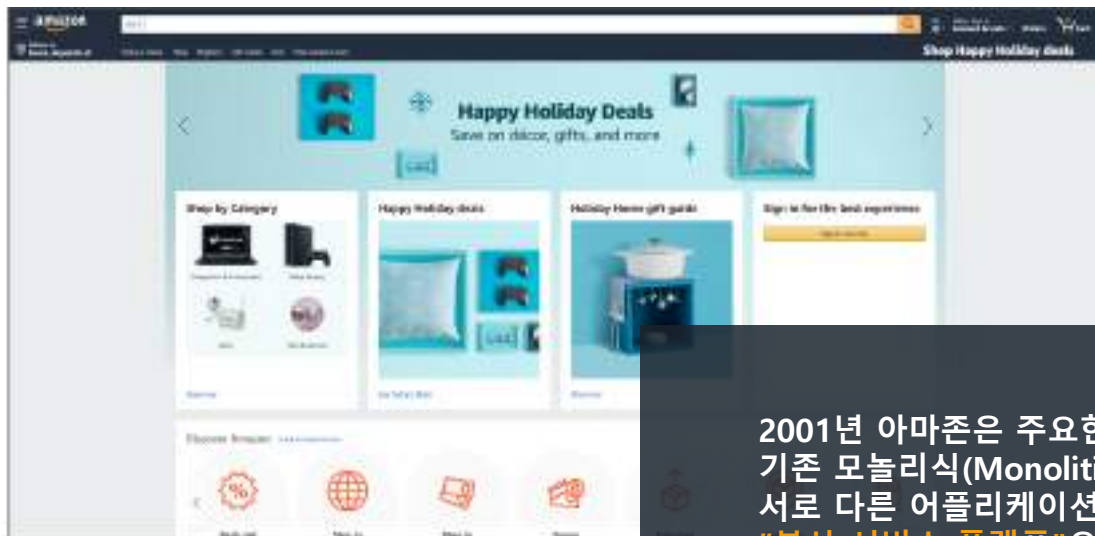


Service Composition with User Interface

- 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)



- 아마존닷컴은 다양한 서비스 제공과 효율적인 운영 환경 전환을 위해 분산 서비스 플랫폼으로 전환 하였습니다.

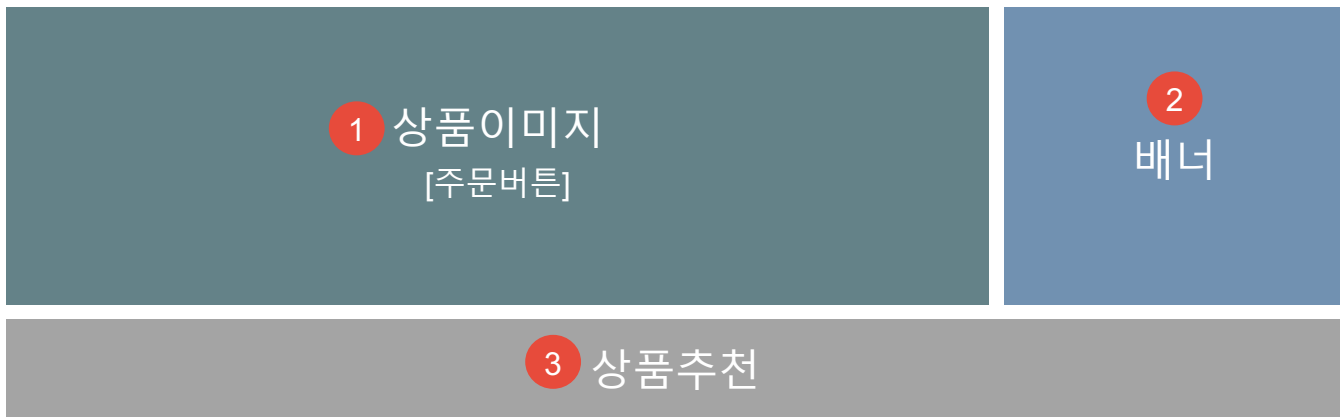


2001년 아마존은 주요한 아키텍처 변화가 있었는데 기존 모놀리식(Monolithic) 기반에서 서로 다른 어플리케이션 기능을 제공하는 **"분산 서비스 플랫폼"**으로 변화 하였습니다.

현재의 Amazon.com의 첫 화면에 들어 온다면, 그 페이지를 생성하기 위해 **100여개가 넘는 서비스**를 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?

W3C Web Components

- Dynamically composed at client-side
- Custom elements
- HTML imports
- Template
- Shadow DOM

```
<style>  
  style for product  
  style for order  
  style for delivery  
</style>
```

```
<html>  
  markup for product  
  markup for order  
  markup for delivery  
</html>
```

```
<script>  
  script for product  
  script for order  
  script for delivery  
</script>
```



```
# main  
<product> <product>  
<order> <order> <order>  
<delivery>
```

```
# product  
<style>  
  style for product  
</style>  
  
<html>  
  markup for  
  product  
</html>  
  
<script>  
  script for product  
</script>
```

```
# order  
<style>  
  style for order  
</style>  
  
<html>  
  markup for order  
</html>  
  
<script>  
  script for B  
</script>
```

```
# delivery  
<style>  
  style for delivery  
</style>  
  
<html>  
  markup for  
  delivery  
</html>  
  
<script>  
  script for  
  delivery  
</script>
```

Almost all the frontend frameworks support Web Components



Polymer



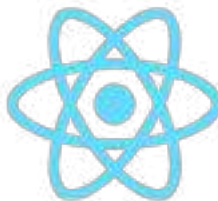
VueJS



AngularJS



ember



React

Custom tag for Domain Class: <product> tag

```
<table>
  <tr
    v-for="(item, index) in props.items"
    :key="item.id"
  >
    <product
      v-model="props.items[index]"
      @inputBuy="showBuy"
      @inputEdit="showEdit"
    ></product>

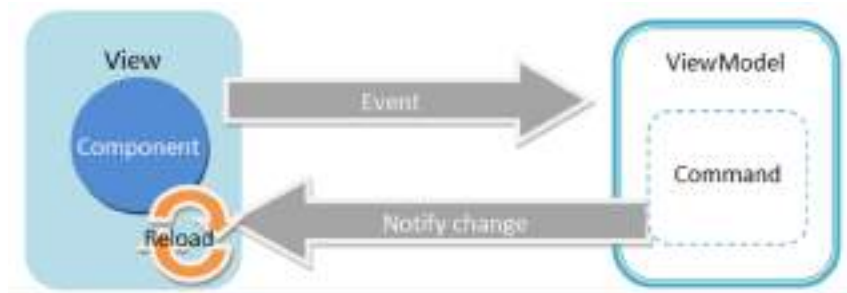
  </tr>
</table>
```

Data binding to Domain Tags

```
<product
  v-model="props.items[index]"
  @inputBuy="showBuy"
  @inputEdit="showEdit"
></product>

<script>
  methods: {
    getProdList() {
      var me = this
      me.$http.get(`${API_HOST}/products`).then(function (e) {
        me.items = e.data._embedded.products;
      })
    }
  }
</script>
```

MVVM



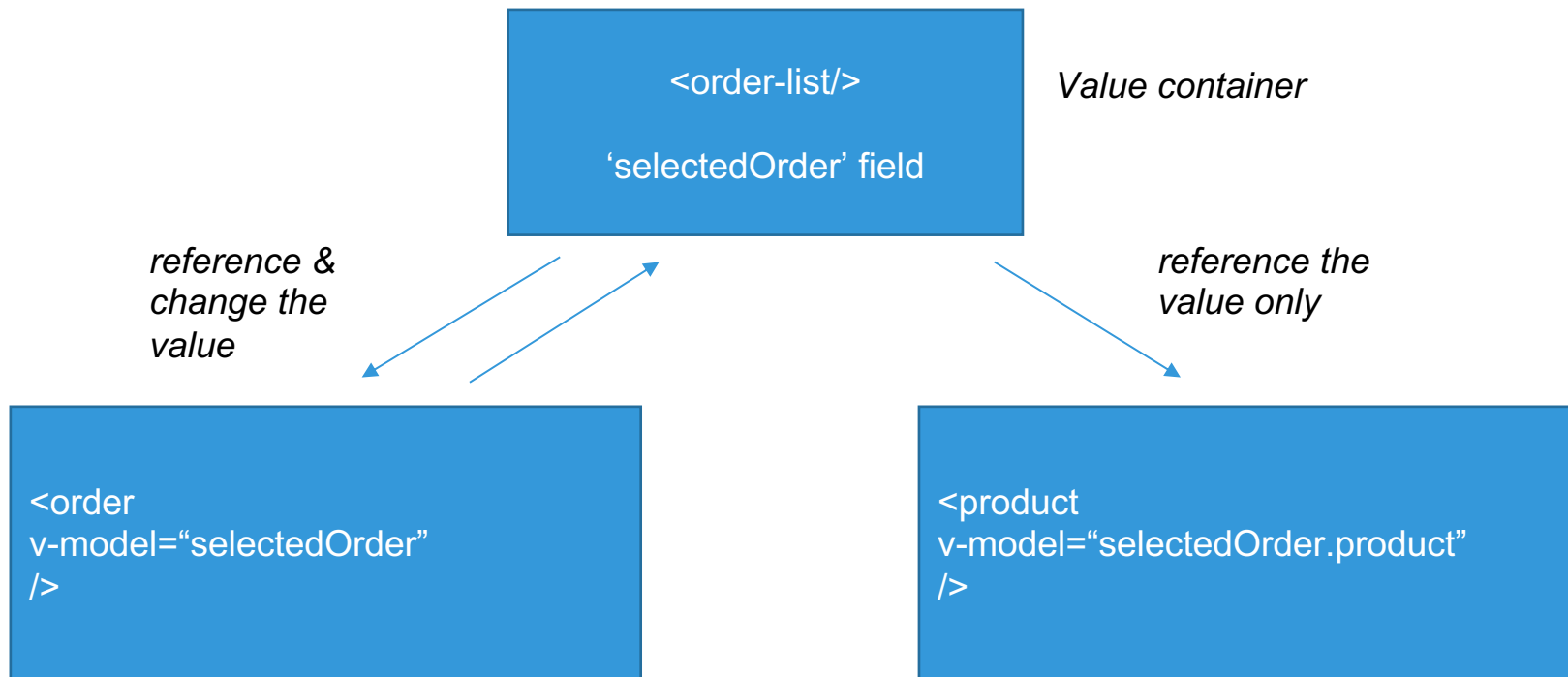
2016 PROFIT AND LOSS STATEMENT
DevAV

NET INCOME \$114,500

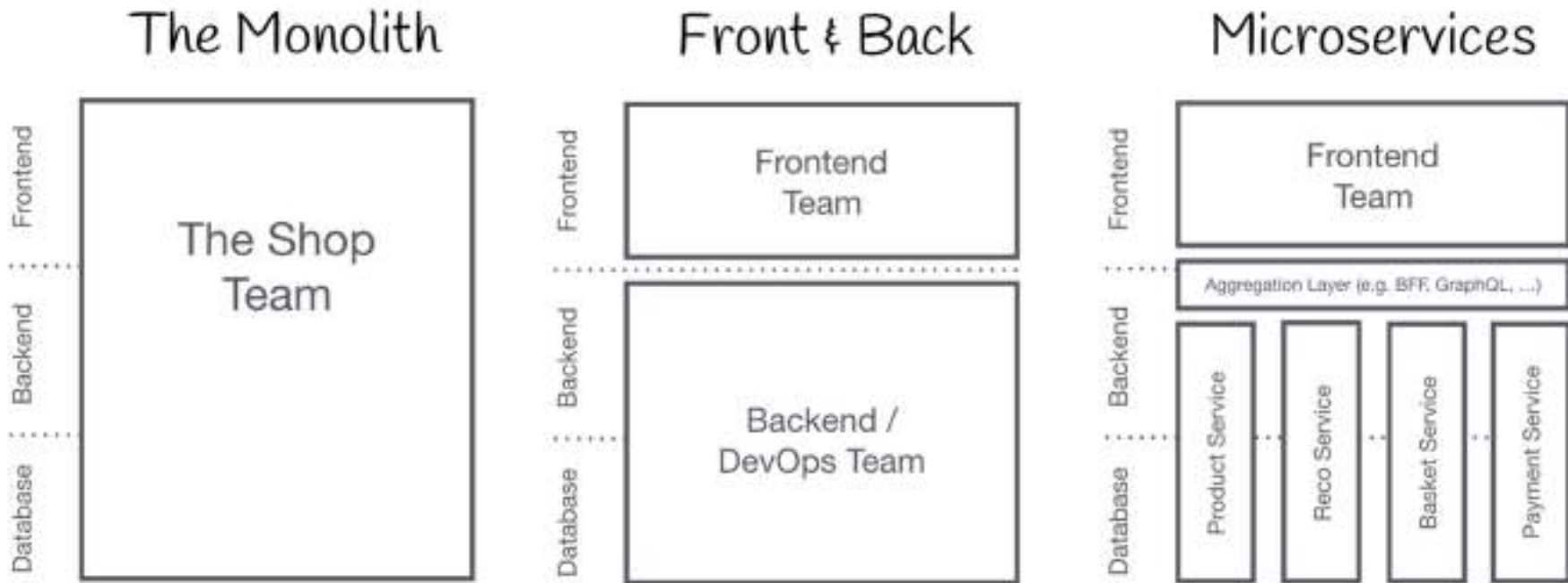
Revenue	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG
Sales	\$20,000	\$60,000	\$22,722	\$22,881	\$80,175	\$44,800	\$50,000	\$62,500
Sales Returns (Reduction)	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Sales Discounts (Reduction)	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Other Revenue 1	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Other Revenue 2	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Other Revenue 3	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0
Net Sales	\$20,000	\$60,000	\$22,722	\$22,881	\$80,175	\$44,800	\$50,000	\$62,500
Cost of Goods Sold	\$20,000	\$20,000	\$20,000	\$20,000	\$20,000	\$20,000	\$20,000	\$20,000
Gross Profit	\$0	\$40,000	\$2,722	\$2,881	\$60,175	\$24,800	\$30,000	\$42,500
Operation Expenses	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG
Salaries & Wages	\$7,000	\$7,075	\$6,399	\$6,900	\$8,175	\$8,200	\$8,000	\$8,000
Depreciation	\$800	\$800	\$800	\$870	\$800	\$800	\$800	\$800
Rent	\$1,000	\$1,070	\$1,034	\$1,234	\$1,000	\$1,000	\$1,000	\$1,000
Office Supplies	\$475	\$489	\$624	\$600	\$677	\$675	\$675	\$675
Utilities	\$125	\$125	\$125	\$125	\$125	\$125	\$125	\$125
Telephone	\$80	\$80	\$80	\$80	\$80	\$80	\$80	\$80
Insurance	\$125	\$125	\$125	\$125	\$125	\$125	\$125	\$125
Taxes	\$200	\$200	\$270	\$289	\$284	\$289	\$289	\$270

Profit and Loss

Interaction between Domain Tags

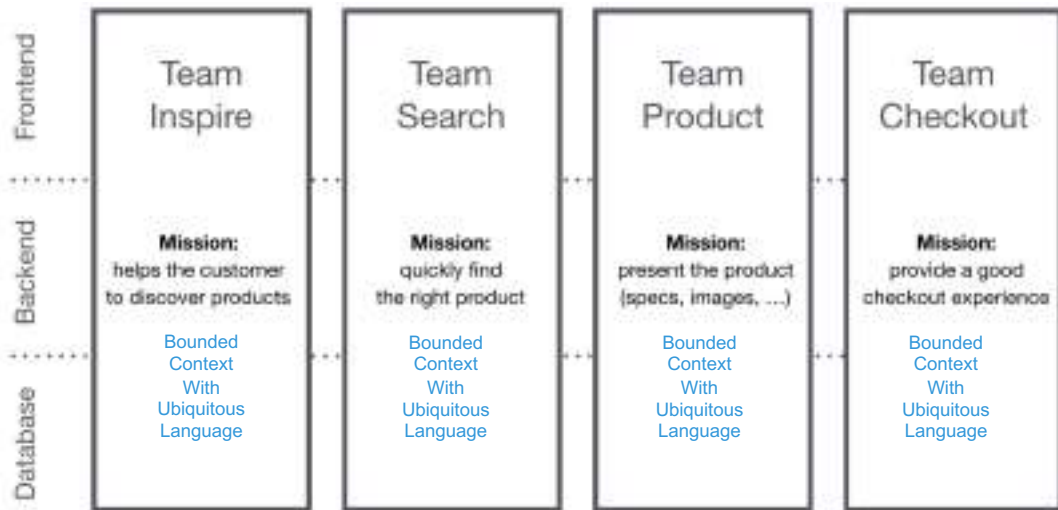


Monolith-Frontend



<https://micro-frontends.org/>

Micro-Frontends



- 팀간 분리된 컴포넌트 기반 개발과 통합
- 팀간 배포 독립성
- 팀간 기술 독립성 (다종의 UI-framework 혼합)
- 장애 격리 (하나의 컴포넌트내의 자바스크립트 엔진이 죽어도 다른 컴포넌트가 영향 안받아야)
- 이벤트 채널을 통한 컴포넌트간 연동

<https://www.martinfowler.com/articles/micro-frontends.html>

<https://micro-frontends.org/>


Lab: Multiple frontend frameworks on a single page

What is your name?

Enter your name above then click the button below to tell the components.

Tell the components


Angular



Hello James from your friendly Angular component.

Say hello

React



Hello James from your friendly React component.

Say hello

While they provide:

- Autonomously deployable
- Communicate each other

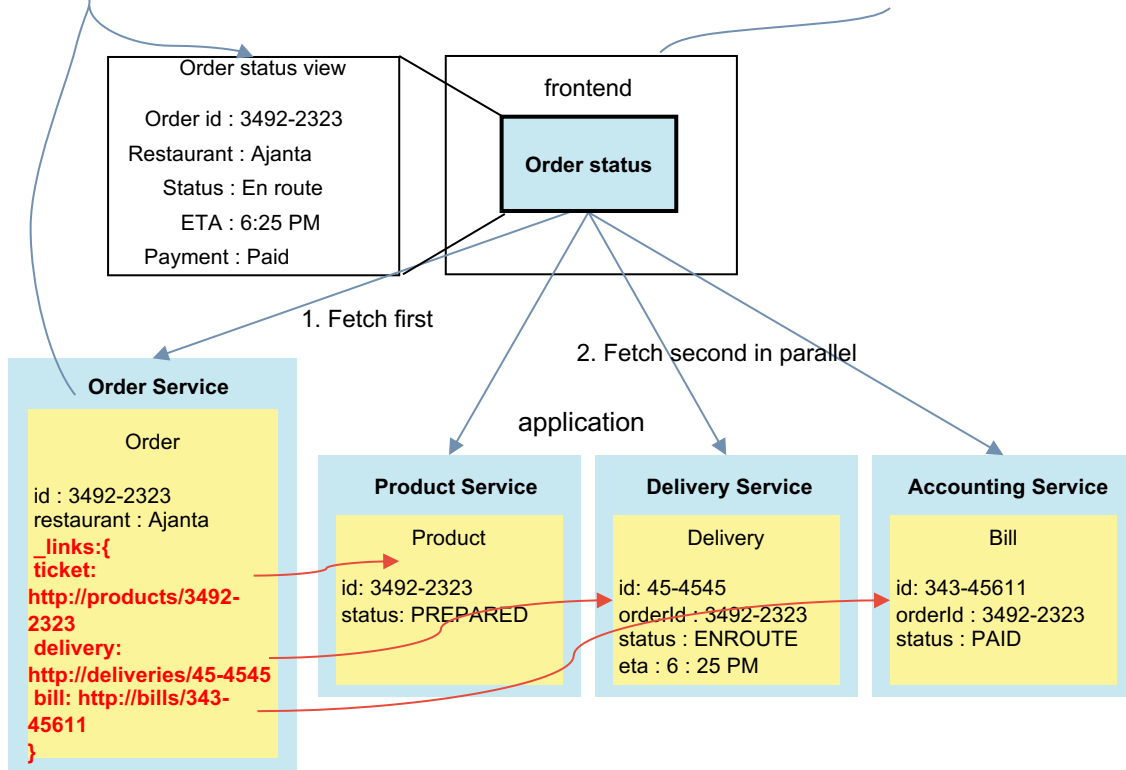
<https://medium.com/javascript-in-plain-english/create-micro-frontends-using-web-components-with-support-for-angular-and-react-2d6db18f557a>

Read-Side using UI:

Data Projection by UI and HATEOAS

Data from multiple services

Mobile device or web application



Issues :

- 단점 : UI 개발 복잡성과 성능

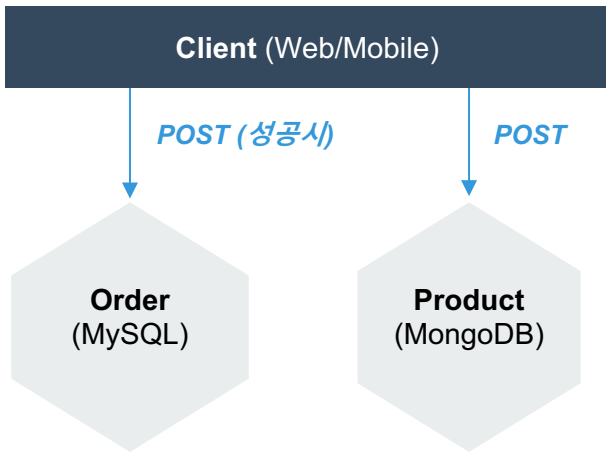
Write-Side using UI:

Distributed Transaction Problem in data mutation by UI composition

In-consistent

Consistent

- 서비스구성 (클라이언트가 순차 호출)



- 조회화면

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!!

Integration Strategies Compared

	UI	Req-Resp	Pub-Sub
읽기	UI Composition + HATEOAS		
쓰기	Not Recommended		

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. 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



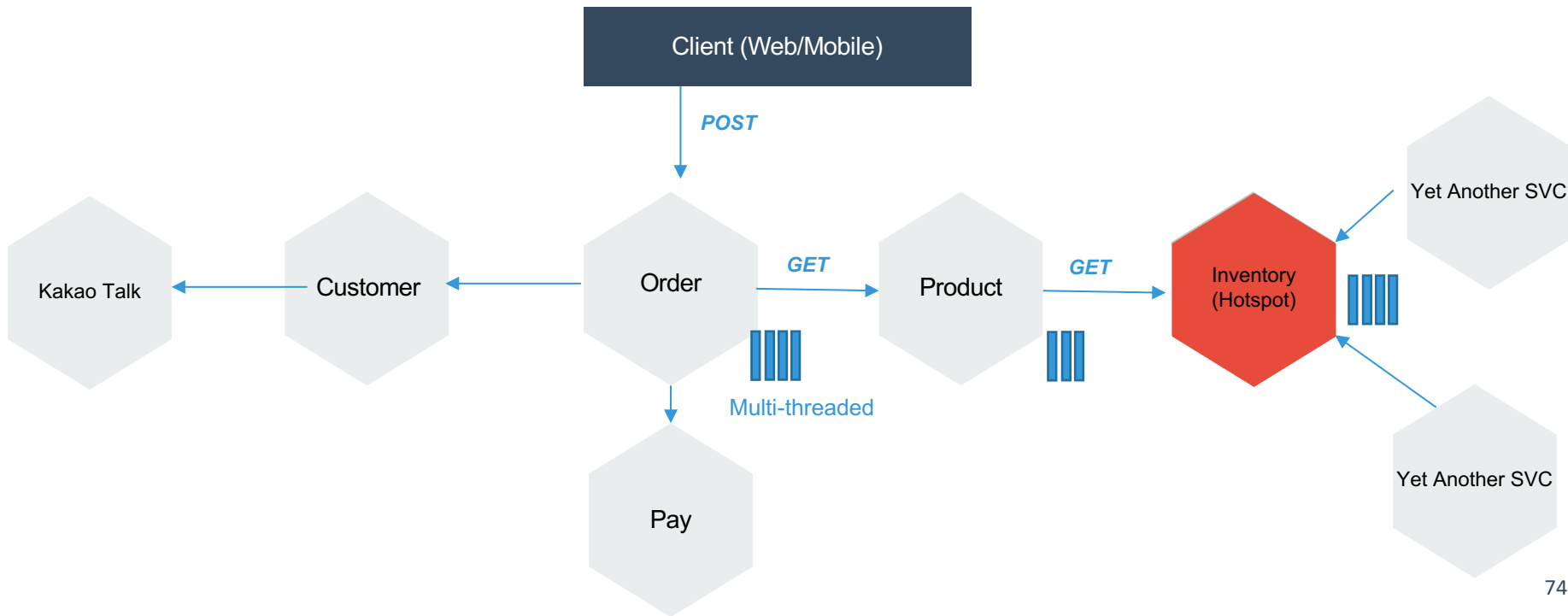
Service Integration by Request-Response

- 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

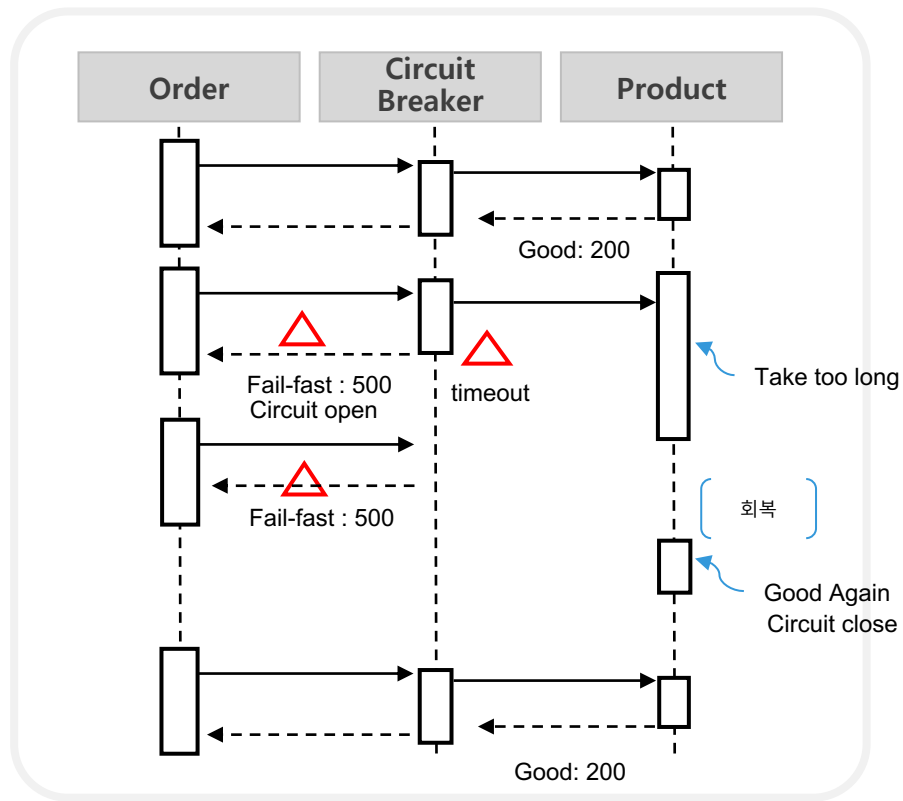
Data Projection in Request-Response model

서비스구성 (Request-Response, Sync 호출)

- 성능저하
- 장애전파



장애전파 차단: 서킷브레이커 패턴



하나의 트랜잭션이 가능한 블로킹이 발생하지 않도록 미리 차단, Fail-Fast 전략

□ 메모리 사용 폭주 막음

장애 감지 시, 차단기 작동(Open)



일시 동안 Fallback으로 서비스 대체



일부 트래픽으로 서비스 정상여부 확인



정상 확인 시, 차단기 해제(Close)

Lab Time: Circuit breaker using istio

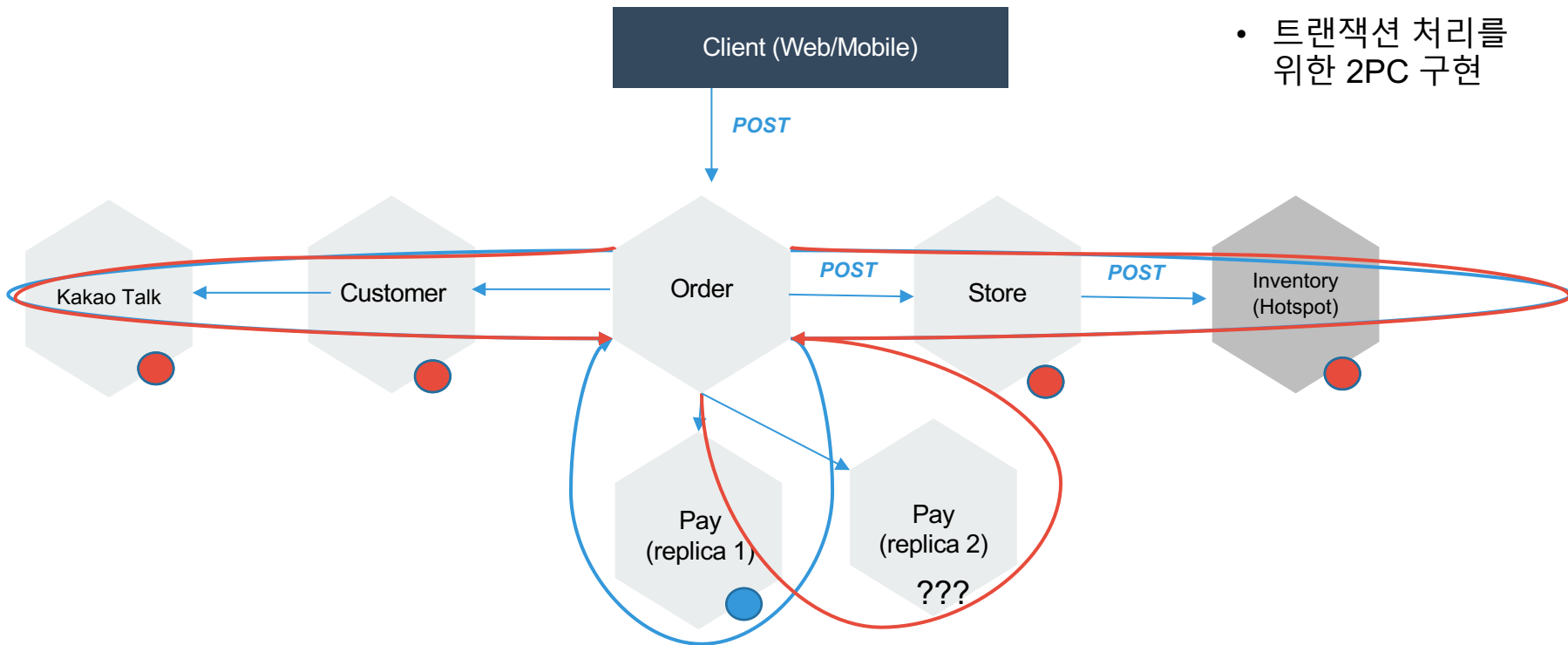
- product 서비스에 서킷 브레이커 적용
 - **connectionPool** :
지정된 서비스에 connections를 제한하여 가능 용량 이상의 트래픽 증가에 따른 서비스 Pending 상태를 막도록 *circuit breaker*를 작동시키는 방법
 - **outlierDetection** :
오류 발생하거나 응답이 없는 인스턴스를 탐지하여 *circuit breaker*를 작동시키는 방법
- 부하를 주어서 지정된 커넥션 만큼 차단 확인
 - `siege -c2 -t10S -v --content-type "application/json" 'http://orders:8080/orders POST {"productId":2,"quantity":1}'`
- 삭제
 - `kubectl delete dr --all`

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: delivery
spec:
 host: delivery
 trafficPolicy:
 connectionPool:
 tcp:
 maxConnections: 1
 http:
 http1MaxPendingRequests: 1
 maxRequestsPerConnection: 1
 outlierDetection:
 consecutiveErrors: 5
 interval: 1s
 baseEjectionTime: 30s
 maxEjectionPercent: 100

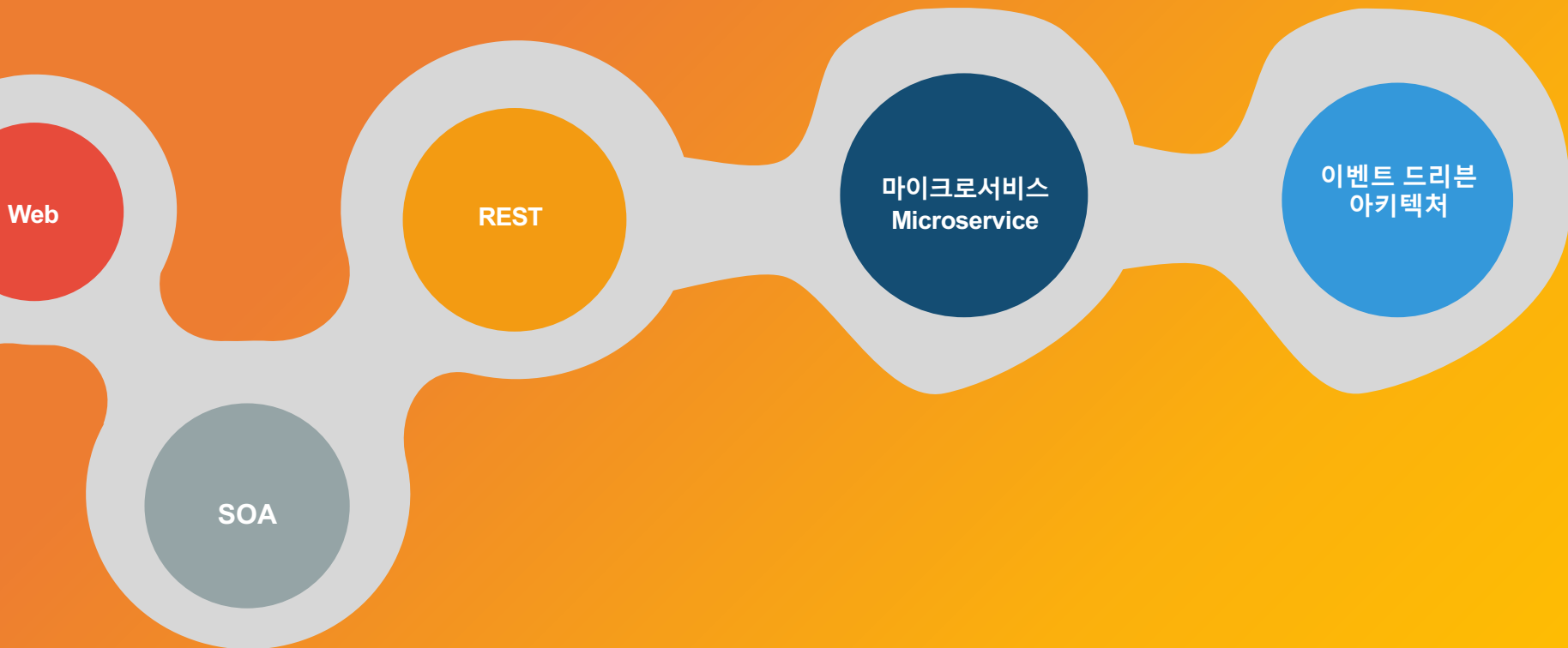
Write-side using Req-Resp: 2-Phase Commit

서비스구성 (Request-Response, Sync 호출)

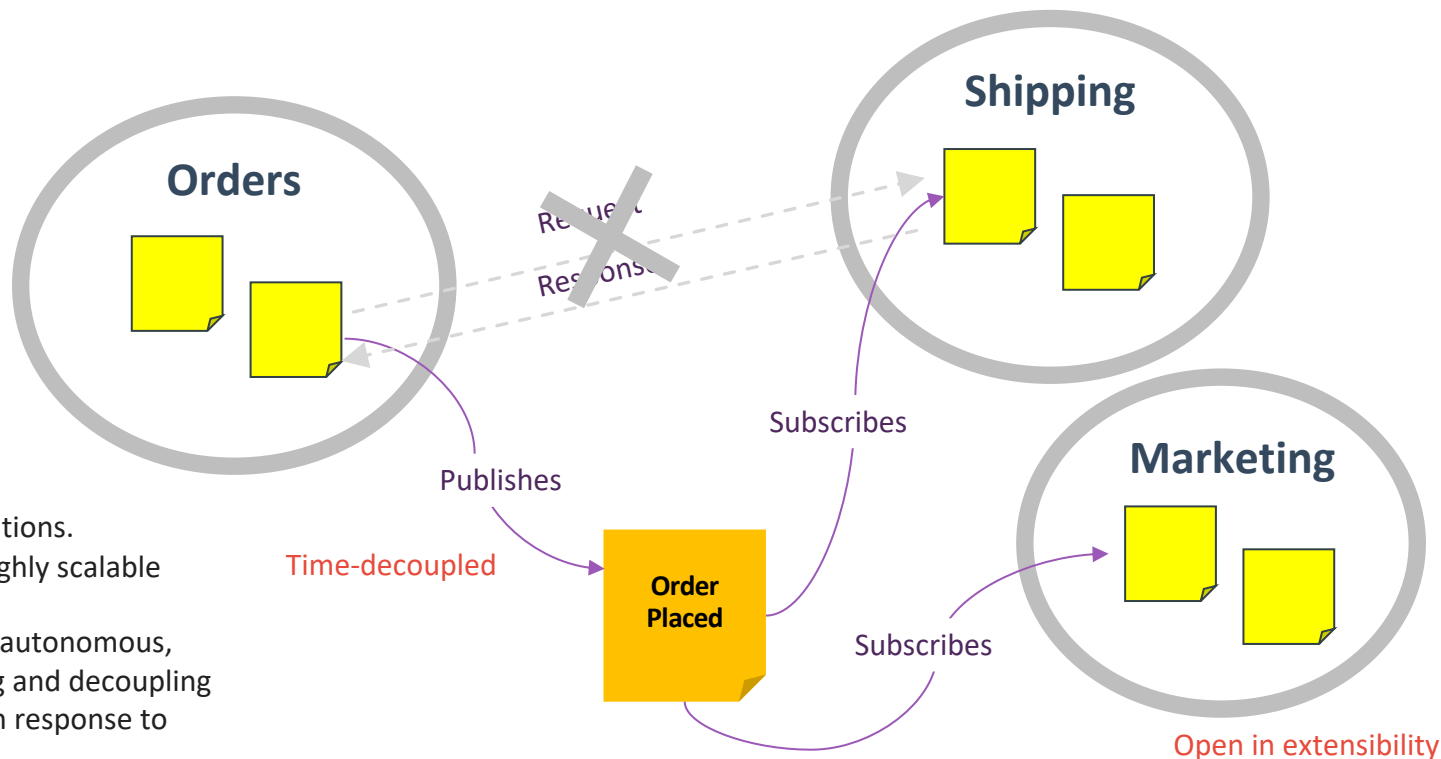
- 성능저하
- 장애전파
- 트랜잭션 처리를 위한 2PC 구현



소프트웨어 아키텍처의 성장 여정



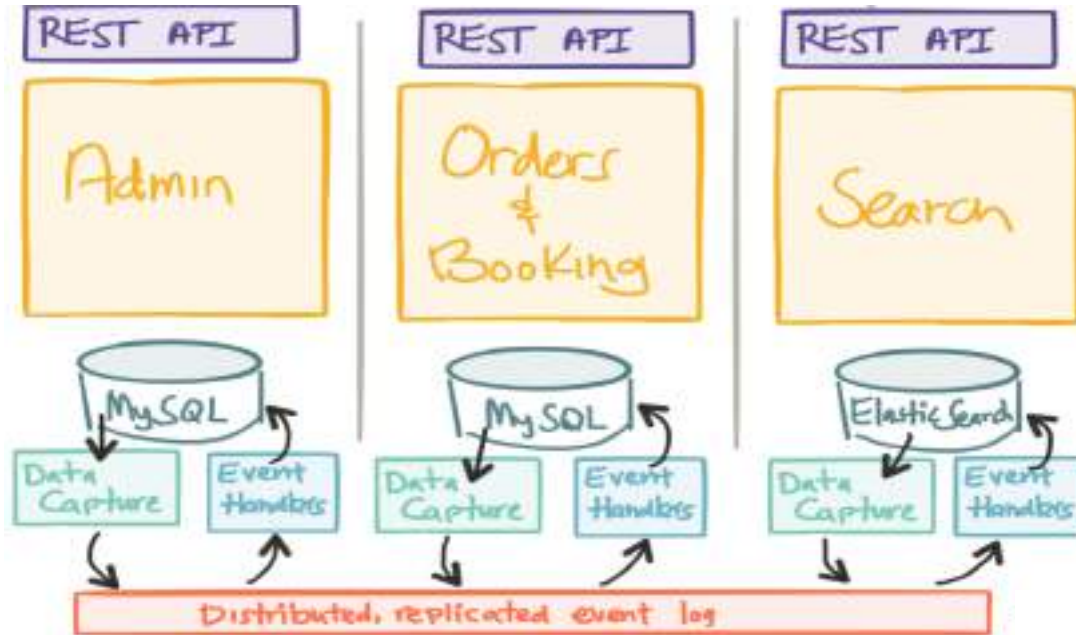
Request-Response 의 한계점과 EDA



- No point-to-point integrations.
- High performance and highly scalable systems.
- Components can remain autonomous, being capable of coupling and decoupling into different networks in response to different events.
- Advanced analytics can be done using complex event processing.

Approach #2 : Event Driven Architecture

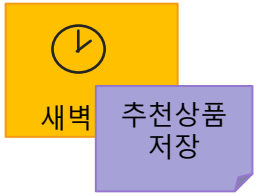
Polyglot-persistence





새로운 기능
= 신규 팀의
합류

마케팅팀

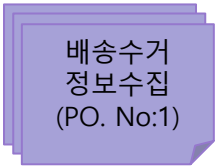
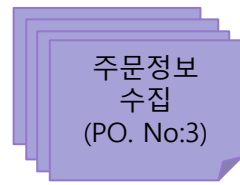


File-System

Id	user	category	cnt
1	홍길동	Electronic	5
2	아무개	Camping	2

Data Join
기능 요구

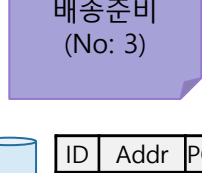
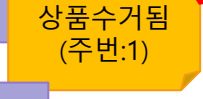
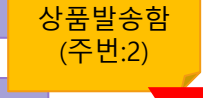
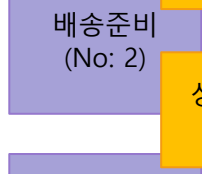
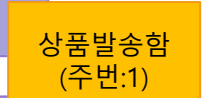
마이페이지 - CQRS



HTML

id	User	Item	Qty	prc	o-stat	d-stat
001	홍길동	TV	5	40,000	cancel	Returned
002	아무개	TV	5	40,000	order	Delivery Started
003	아무개	RADIO	2	20,000	order	

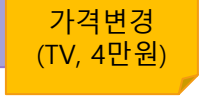
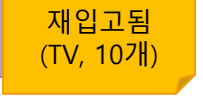
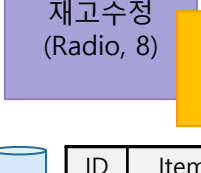
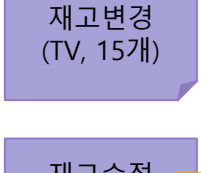
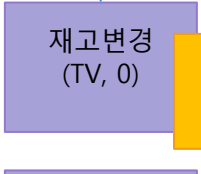
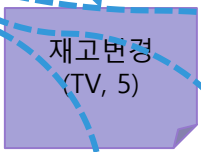
배송팀



Redis

ID	Addr	PO-id	Qty	Status
1	강동구	1	5	Returned
2	강서구	2	5	Started
3	강서구	3	2	prepare

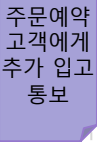
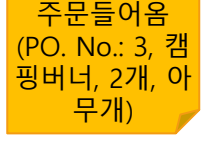
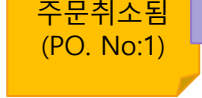
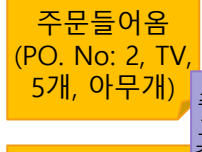
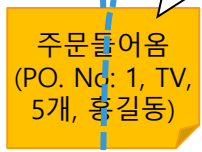
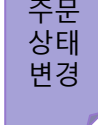
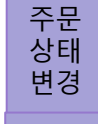
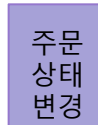
상품팀



MongoDB

ID	Item	Stock
1	TV	15
2	Radio	8

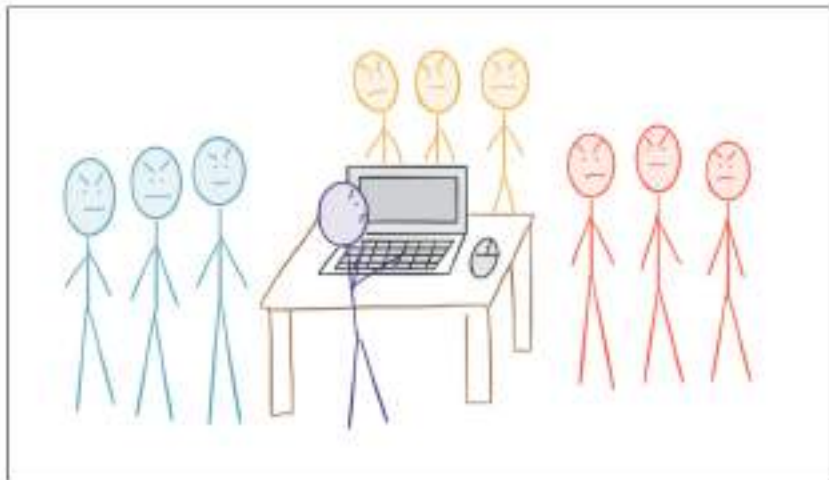
주문팀



MySQL

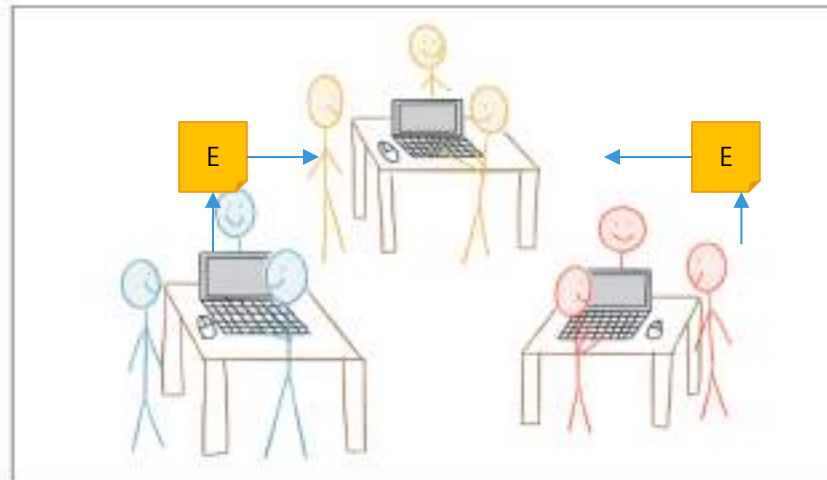
ID	User	Item	Qty	status
1	홍길동	TV	5	Returned
2	아무개	TV	5	Delivery Started
3	아무개	버너	2	ordered

Event Driven MSA Architecture



Monolith:

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



Reactive Microservices:

- Autonomously designed Model(Document)
- Polyglot Persistence

Case Study – 11번가

<https://tv.naver.com/v/11212897>

Integration Strategies Compared

	UI	Req-Resp	Pub-Sub
읽기	UI Composition + HATEOAS	GET Request + Circuit Breaker	
쓰기	Not Recommended	2PC Not Recommended	

Event Publisher : Order

@Entity

public class Order {

...

@PostPersist // 주문이 저장된 후에

private void publishOrderPlaced() {

OrderPlaced orderPlaced = **new** OrderPlaced(); // 주문이 들어온 사실을 이벤트로 작성

orderPlaced.setOrderId(**id**);

...

orderPlaced.publish(); // 메시지 큐에 주문이 들어왔음을 신고

}

}

Event Consumer : Delivery

```
@KafkaListener(topics = "shopping") // shopping 토픽의 이벤트를 수신  
public void onOrderPlaced(OrderPlaced orderPlaced) { // OrderPlaced 이  
벤트가 들어오면..
```

```
    deliveryService.start(orderPlaced.getId());  
  
}
```

Event Consumer : Product

```
@KafkaListener(topics = "shopping") // 쇼핑 토픽을 수신
public void onOrderPlaced(...) { // 주문이 들어오는 이벤트가 오면

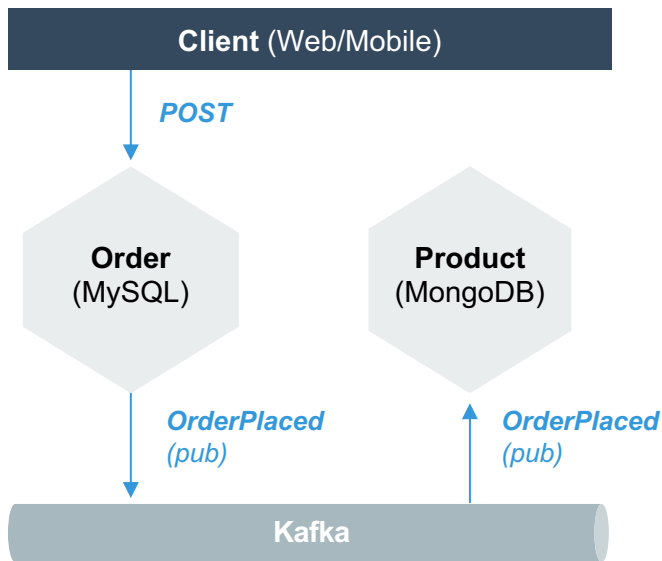
    // 해당 상품의 재고량을 주문량 만큼 줄여서 저장
    Product product =
productRepository.findById(orderPlaced.getProductId()).get();
    product.setStock(product.getStock() - orderPlaced.getQuantity());

    productRepository.save(product);

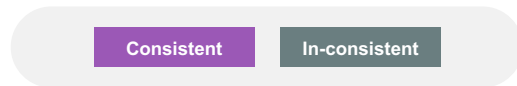
}
```

Eventual Consistency를 통한 분산 트랜잭션 처리

- 서비스구성 (Eventual TX)



- 조회화면



주문량 : 0 , 재고량 : 10
주문량 : 1 , 재고량 : 10
주문량 : 1 , 재고량 : 9
주문량 : 2 , 재고량 : 9
주문량 : 2 , 재고량 : 8

잠깐 불일치

결국 일치

여기서 선택의 길을 만남...

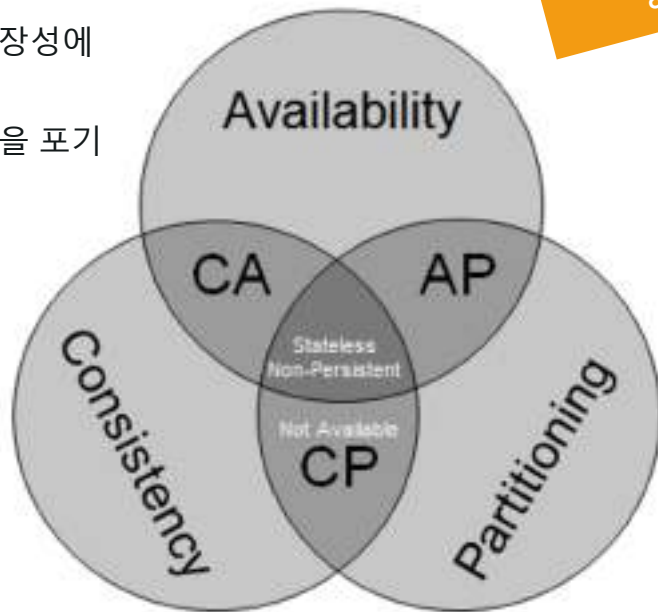


미안하지만...
하나만 선택해!!!

- 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?

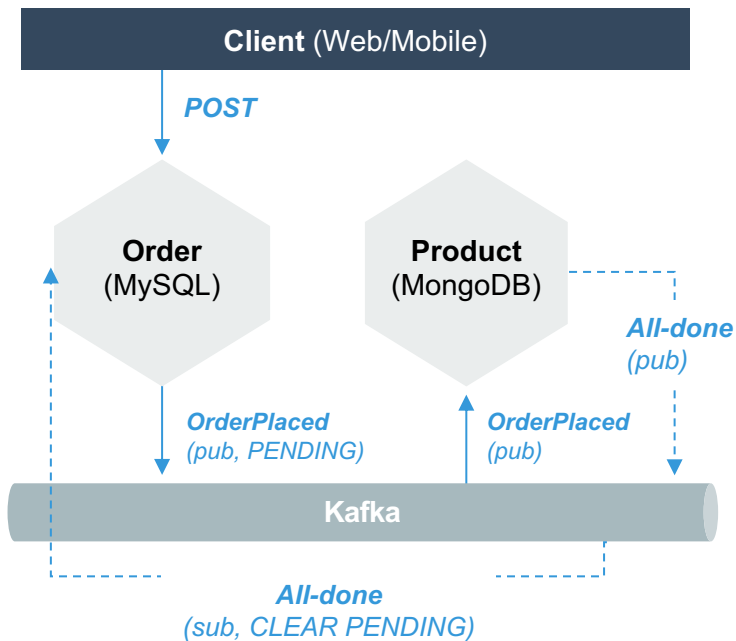
“성공한 내 서비스의 경쟁자들은
무엇을 포기했는가?”

❑ 강력한 의사결정 필요
(무엇으로 태어날 것인가...)

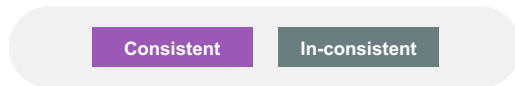


Eventual TX – 불일치가 Mission Critical 한 경우

- 서비스구성 (Eventual TX)



- 조회화면



주문량 : 0 , 재고량 : 10
주문량 : 1 , 재고량 : 10 (PENDING)
주문량 : 1 , 재고량 : 9
주문량 : 2 , 재고량 : 9 (PENDING)
주문량 : 2 , 재고량 : 8

불일치 할 수
있음을 표시

결국 일치

What is the Saga Pattern?

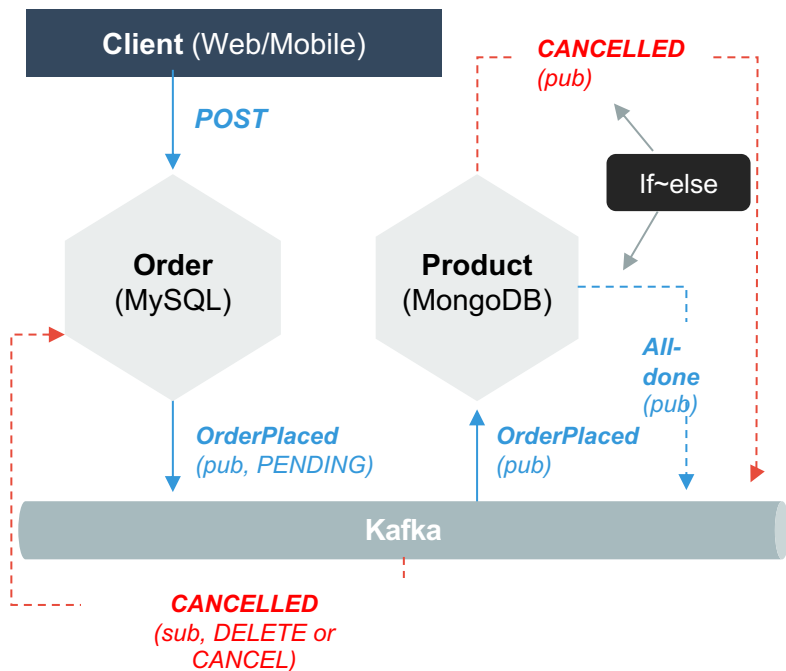
The Saga Pattern is a design pattern that provides a solution for implementing transactions in the form of **sagas** that span across two or more microservices.

A **saga** can be defined as a sequence of local transactions. Where each participating microservice executes one or more local transactions, and then publish an event that is used to trigger the next transaction in a saga that resides in another participating microservice.

When one of the transactions in the sequence fails, the saga executes a series of **compensating transactions** to undo the changes that were made by the preceding transactions.

Eventual TX – Rollback (Saga Pattern)

- 서비스구성 (Eventual TX)



- 조회화면

Consistent In-consistent

주문량 : 9 , 재고량 : 2
주문량 : 9 , 재고량 : 2 (PENDING)
주문량 : 10 , 재고량 : 1
주문량 : 11 , 재고량 : 1 (PENDING)
주문량 : 10 , 재고량 : 0 취소됨

복구 &
결국 일치

Integration Strategies Compared

	UI	Req-Resp	Pub-Sub
읽기	UI Composition + HATEOAS	GET Request + Circuit Breaker	
쓰기	Not Recommended	2PC Not Recommended	Saga (Eventual Consistency)



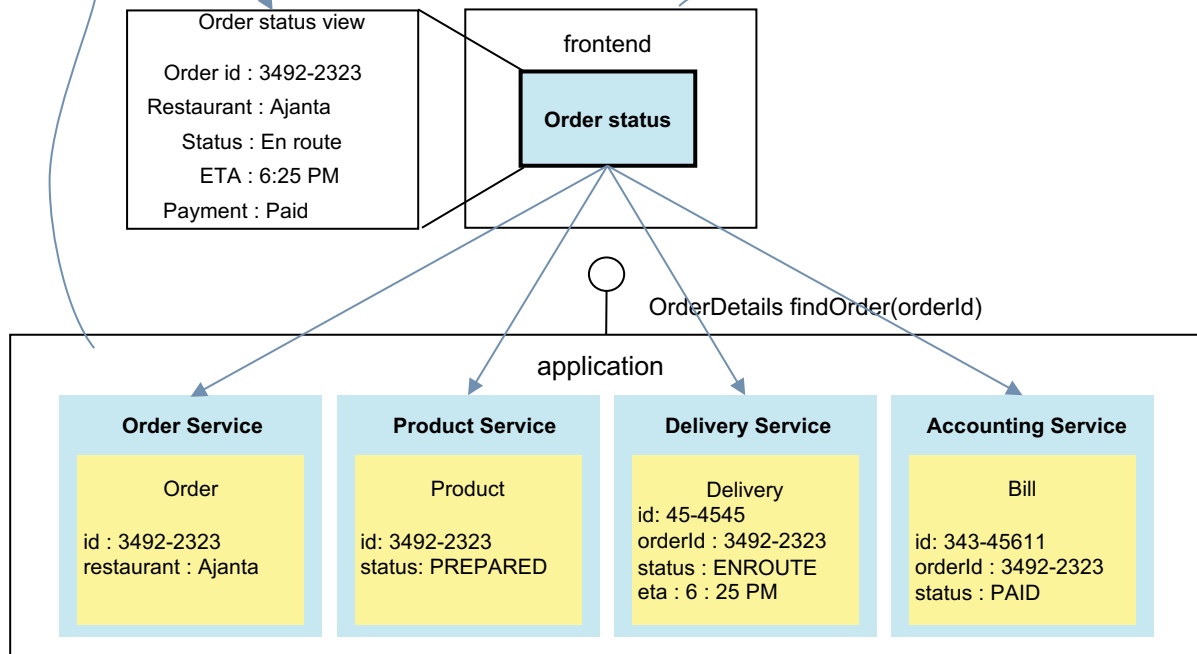
Data Query (Projection) in MSA

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

분산서비스에서의 Data Projection Issue

Data from multiple services

Mobile device or web application

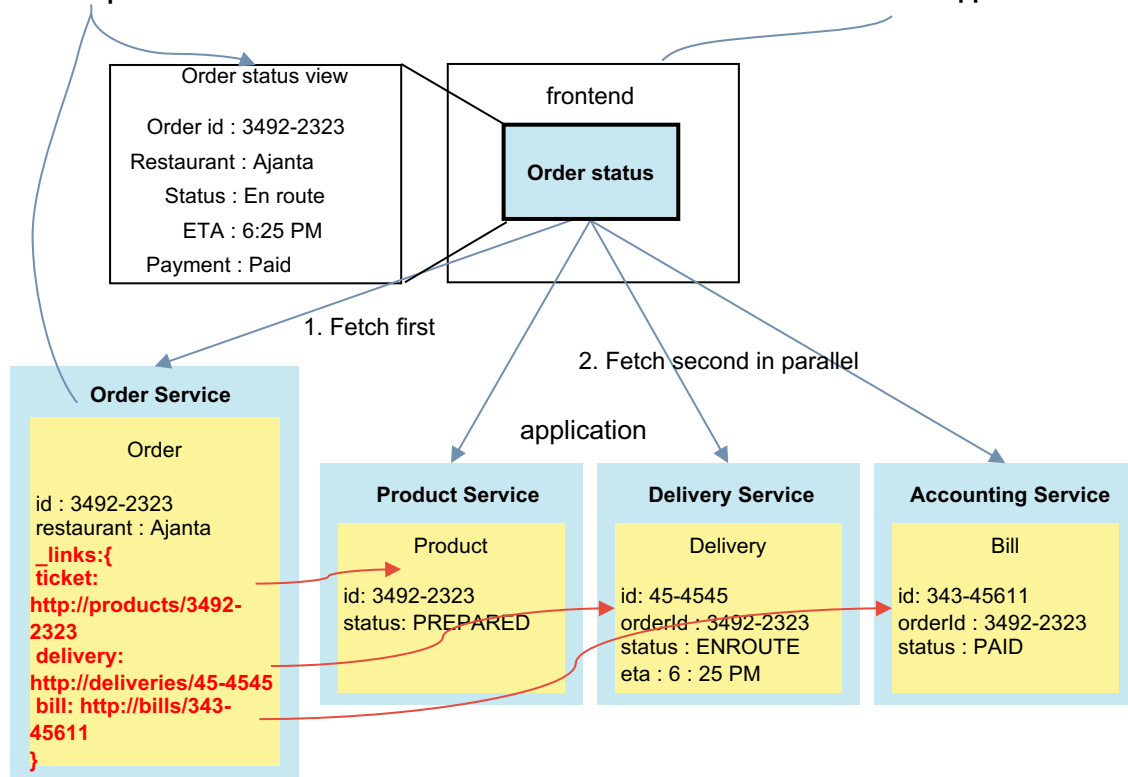


Issues :
1. 성능/속도
2. 데이터량

Data Projection by UI and HATEOAS

Data from multiple services

Mobile device or web application



Issues :

- 단점 : UI 개발 복잡성과 성능

Lab Time: Data Projection by UI and HATEOAS (1/2)

// 주문정보 조회

```
$http.get(`http://orders:8080/orders/search/findByCustomerId?customerId=고객ID`)
```

```
.then(function (orderResult) {
```

// 주문 정보에 해당하는 배송정보 조회

```
orderResult.forEach(function (orderItem, orderIndex) {
```

```
  $http.get(orderItem._links.delivery.href)
```

```
  .then(function (deliveryResult) {
```

// 주문과 배송정보를 조합

```
  })
```

```
})
```

```
})
```

<https://github.com/event-storming/ui/blob/master/src/components/order/OrderListMashup.vue>

Lab Time: Data Projection by UI and HATEOAS (2/2)

- 주문을 여러건 한다.
- Chrome 의 개발자 모드 > Network 탭을 연다.
- Menu 의 My page (UI Mash Up) 을 열어서 API 날라가는 모습을 확인한다.

주문 내역

주문 번호	주문상품	구매수량	결제금액	주문상태	주문상세	주문취소
1	MASK	1	20000	DeliveryCompleted	OrderPlaced	주문취소
2	NOTEBOOK	1	30000	DeliveryCompleted	OrderPlaced	주문취소
3	TV	1	10000	DeliveryCompleted	OrderPlaced	주문취소
4	TV	1	10000	DeliveryCompleted	OrderPlaced	주문취소
5	TV	1	10000	DeliveryCompleted	OrderPlaced	주문취소

1@uengine.org

findByCustomerId?customerId=1@uengine.org

1@uengine.org

findByCustomerId?customerId=1@uengine.org

findByOrderIdOrderByDeliveryIdDesc?orderId=1

findByOrderIdOrderByDeliveryIdDesc?orderId=2

findByOrderIdOrderByDeliveryIdDesc?orderId=3

findByOrderIdOrderByDeliveryIdDesc?orderId=4

findByOrderIdOrderByDeliveryIdDesc?orderId=5

findByOrderIdOrderByDeliveryIdDesc?orderId=1

findByOrderIdOrderByDeliveryIdDesc?orderId=4

findByOrderIdOrderByDeliveryIdDesc?orderId=2

findByOrderIdOrderByDeliveryIdDesc?orderId=3

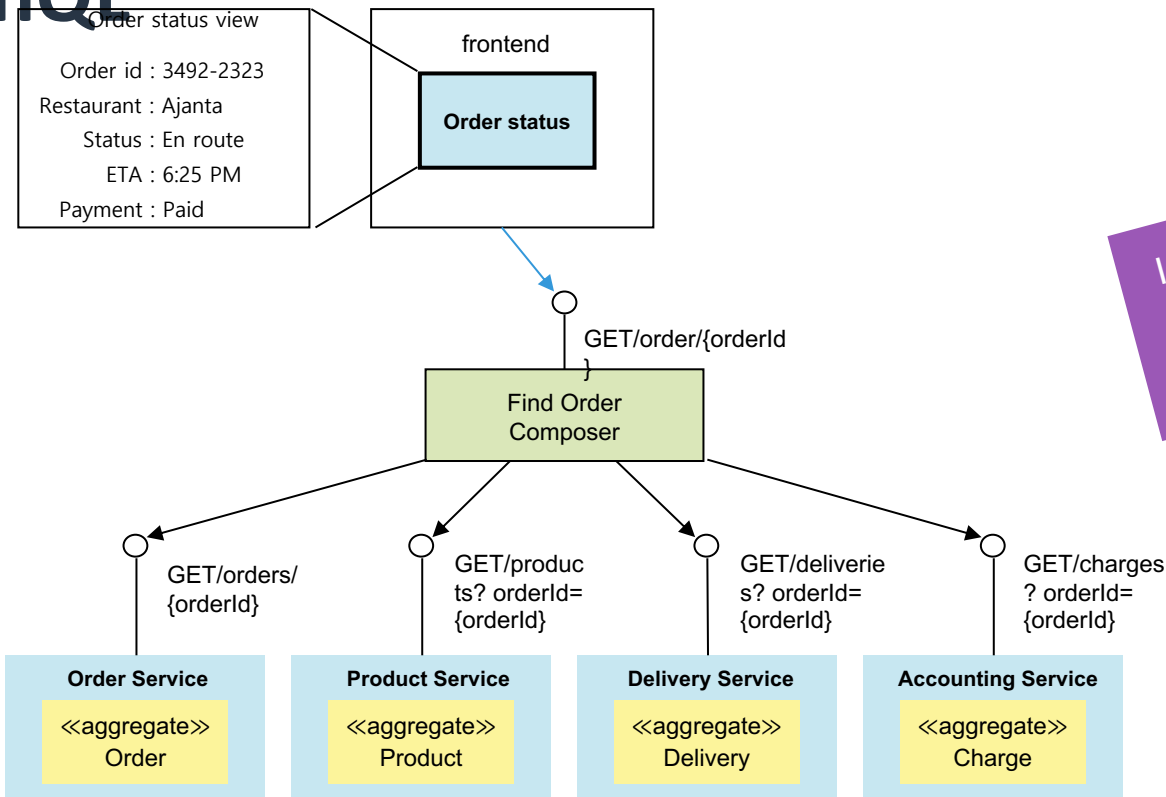
findByOrderIdOrderByDeliveryIdDesc?orderId=5

* _embedded: {,_,...}

* _links: {self: {href: "http://localhost:8081/...}}

하위 데이터를
HATEOAS 로
유지시 UI 개발
이 편리

Data Projection by Composite Service or GraphQL



Issues :

- 단점 : 성능(속도), 장애전파

Lab Time: Data Projection by Composite Service (1/3)

- 목표 : 사용자의 주문이력과 각 주문에 대한 상세 정보 (주문, 상품, 배송) 를 조합하여 사용자에게 보여준다.
- 작업 순서
 - 데이터를 합성할 신규 서비스 생성
 - 주문서비스의 주문 이력 API 를 호출한다.
 - 주문건에 대한 상품서비스의 상품정보 API 호출한다.
 - 주문건에 대한 배송서비스의 배송정보 API 를 호출한다.
 - 호출 결과들을 모아서 보여주고자 하는 데이터를 만들어서 return 한다.

Lab Time: Data Projection by Composite Service (2/3)

```
CompletableFuture<List<OrderInfo>> orderListCF = CompletableFuture.supplyAsync(() -> {  
    // Call OrderService API  
}).thenCompose(orderListObject -> CompletableFuture.supplyAsync(() -> {  
    // 조회된 주문별로 다른 서비스 호출  
  
    CompletableFuture<Product> productInfoCF = CompletableFuture.supplyAsync(() -> {  
        // Call product Service API  
    });  
  
    CompletableFuture<Delivery> deliveryInfoCF = CompletableFuture.supplyAsync(() -> {  
        // Call product Service API ( HATEOAS API 호출 )  
    });  
  
    // 모든 작업이 끝나기를 기다린다.  
    CompletableFuture.allOf(productInfoCF, deliveryInfoCF).join();  
  
    // 데이터를 조합한다  
    OrderInfo orderInfo = new OrderInfo(order, productInfoCF.get(), deliveryInfoCF.get());  
});
```

https://github.com/event-storming/composite_service/blob/master/src/main/java/com/example/template/CompositeService.java

Lab Time: Data Projection by Composite Service (3/3)

- 참고 코드 실행

- git clone https://github.com/event-storming/composite_service.git
- cd composite_service
- mvn spring-boot:run

- 주문 여러건 하기

- http localhost:8081/orders productId=2 quantity=1 customerId="1@uengine.org" customerName="홍길동" customerAddr="서울시"

- 호출해보기

- http localhost:8088/composite/orders/1@uengine.org
- Thread 부분을 주석을 해제 하고 서비스 재시작 후 호출

- 단점 확인해보기

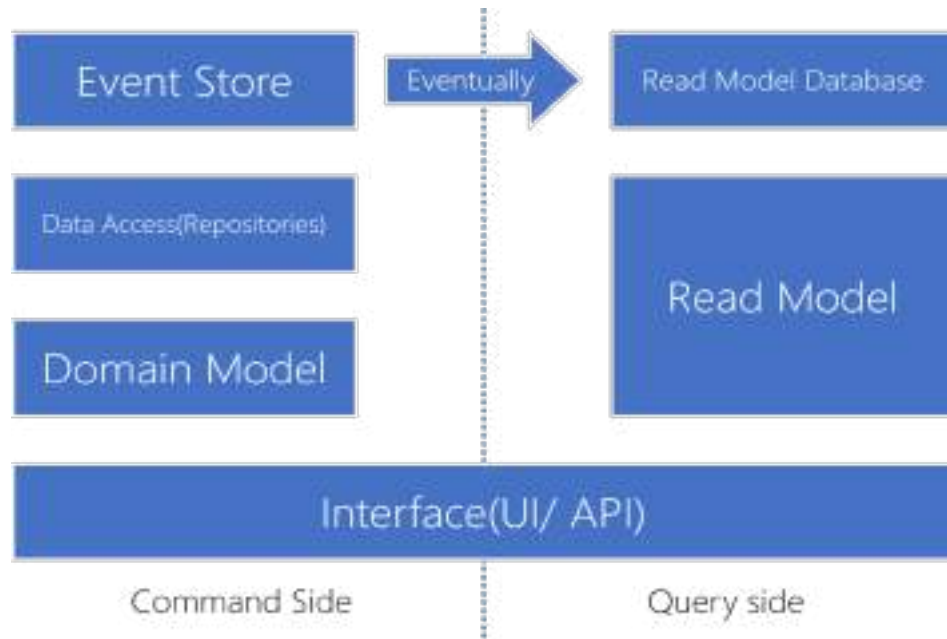
- 주문, 배송, 상품 서비스가 모두 가동중이어야 데이터 조회가 됨
- 주문이력이 많을시에 모든 데이터를 조회 하기때문에 시간이 많이 걸림
- 각 호출 API 별로 return 되는 data 를 알고 있어야 함 (각 서비스에서 변경시 잦은 변경 요청)

생각을 다르게 하자 : CQRS and Event-Sourcing

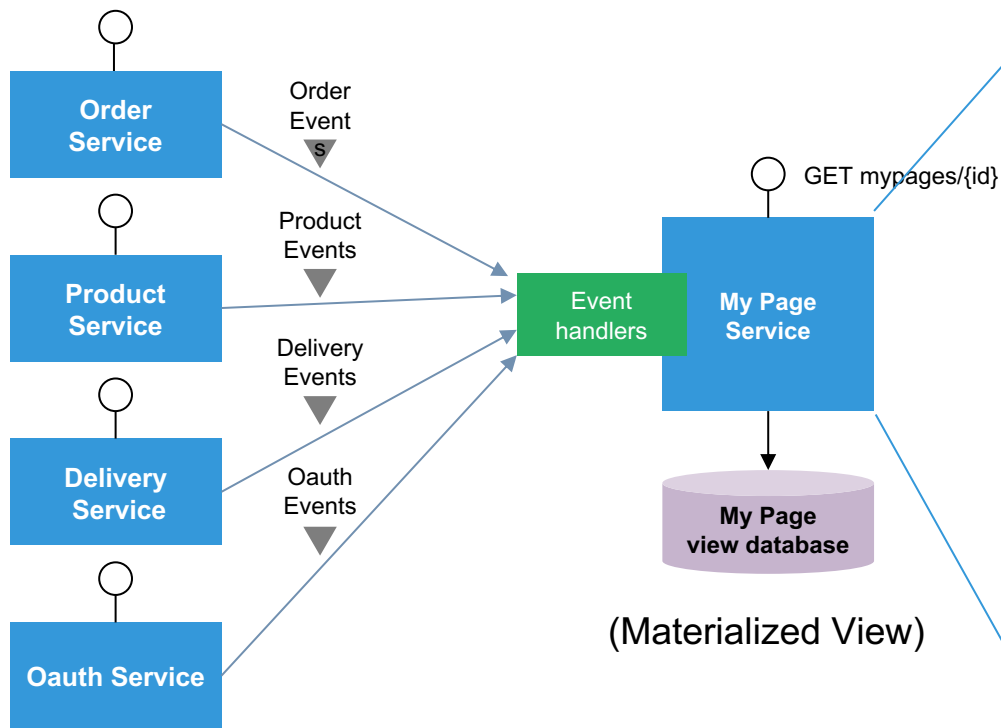
- 데이터 원본에서 매번 가져오지 말고, 취합된 별도의 뷰를 미리 만들어놓자
- Command (Write) / Query (Read) Responsibility Segregation
- 읽기전용 DB와 쓰기 DB를 분리함으로 써 빠른 읽기 구현
- Query 뷰를 다양하게 구성하여 여러 MSA 서비스 목적에 맞추어 각 서비스의 Polyglot Persistence 구현

<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



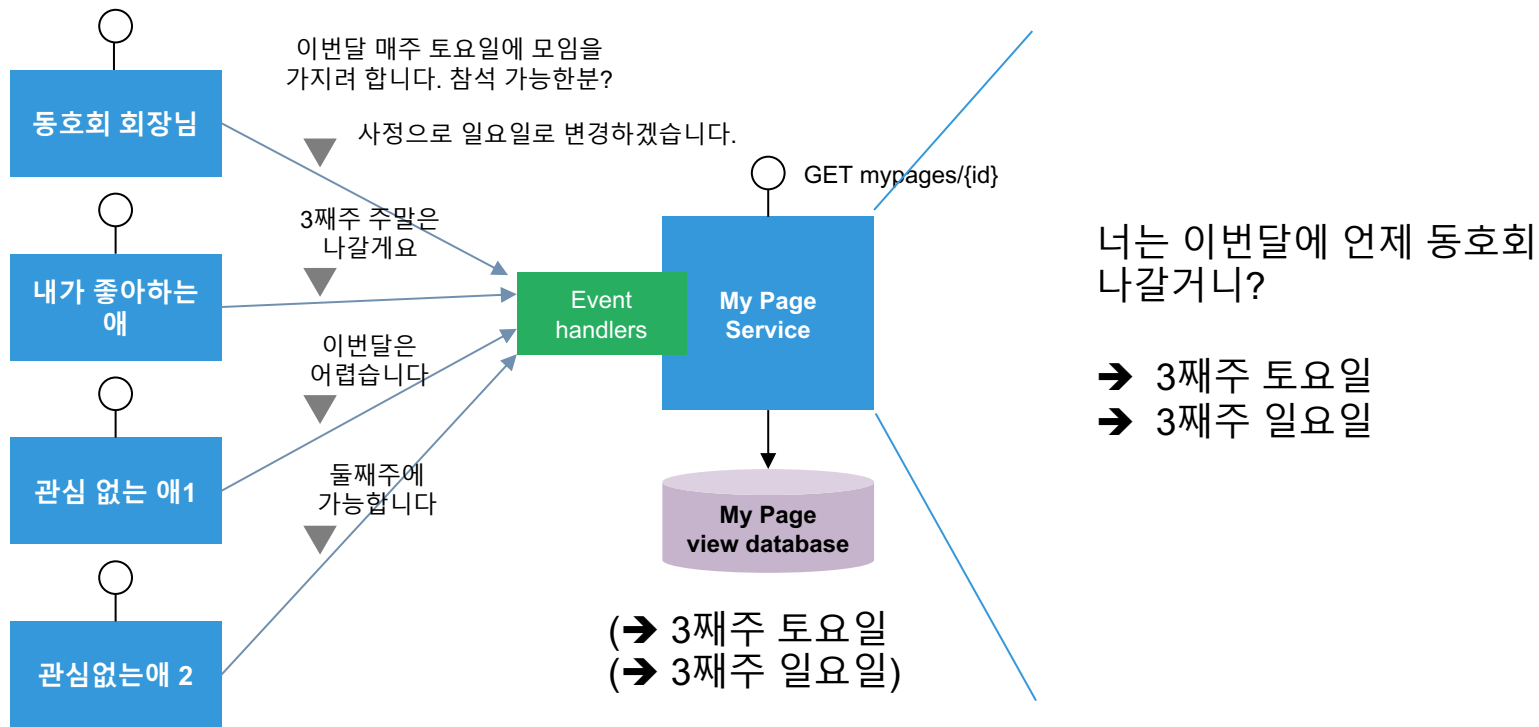
Issues :

- 장점 : 성능, 장애격리
- 단점 : 다이나믹 쿼리는 불가

The screenshot shows a web application interface. At the top, there's a header with '12 STORE'. Below it, there's a section for '유저 정보' (User Information) with the name '유연진' and some details. Below that, there's a section for '주문 내역' (Order History) with a table of orders. The table has columns for 'Order No.', 'Product Name', 'Quantity', 'Price', 'Status', 'Delivery', and 'Action'. There are 5 rows of data. Each row has a green button labeled '주문 취소' (Cancel Order) and a link labeled '주문자 정보' (Orderer Information). At the bottom, there's a pagination bar showing 'Items per page: 10' and '1 of 5'.

Order No.	Product Name	Quantity	Price	Status	Delivery	Action
1	BOOK	10000	1	주문 완료	배송완료	주문자 정보
2	CLOCK	30000	1	주문 완료	배송완료	주문자 정보
3	CLOCK	30000	1	주문 완료	배송완료	주문자 정보
4	BOOK	40000	1	주문 완료	배송완료	주문자 정보
5	BOOK	60000	1	주문 완료	배송완료	주문자 정보

CQRS 의 확장 : Multiple Event Sources



Integration Strategies Compared

	UI	Req-Resp	Pub-Sub
읽기	UI Composition + HATEOAS	GET Request + Circuit Breaker	CQRS
쓰기	Not Recommended	2PC Not Recommended	Saga (Eventual Consistency)

다음중 Saga 패턴이 해소하고자 하는 것은?

1. 커맨드와 쿼리의 역할 분리를 통한 성능개선
2. 하나 이상의 마이크로서비스에 대한 분산 트랜잭션 처리
3. 마이크로 서비스의 데이터베이스 구현



Quiz

- 다음중 CRUD 오퍼레이션을 Command와 Query 로 잘 묶은 것은? (CRUD = Create, Read, Update, Delete)
 - Command: Create, Read, Delete | Query: Update
 - Command: Create, Update, Delete | Query: Read
 - Command: Create, Update | Query: Read, Delete
 - Command: Update, Delete | Query: Read, Create



Quiz

- Event Sourcing 과 CQRS 를 적용했을때 얻을 수 있는 장점은?
 1. 읽기 행위 전용과 쓰기 전용의 데이터베이스를 2개 이상으로 나눌 수 있게 한다.
 2. 도메인 이벤트를 저장하고 전 시스템의 전이상태를 보존하여 데이터베이스 시스템의 실패시에 데이터를 복구 할 수 있다.
 3. 이벤트 스토어를 리플레이시켜서 어떤 데이터이든 상태를 재생성할 수 있다.
 4. 1,2,3 모두 옳다
 5. 1,2,3 모두 틀렸다

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. 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

DDD Patterns



Domain-Driven Design & MSA

DDD for MSA

- **Bounded Context 와 Ubiquitous Language**
 - 어떤 단위로 마이크로 서비스를 쪼개면 좋은가?
- **Context Mapping**
 - 서비스를 어떻게 결합할 것인가?
- **Domain Events**
 - 어떤 비즈니스 이벤트에 의하여 마이크로 서비스들이 상호 반응하는가?



*The key to controlling complexity is a
good domain model
— Martin Fowler*



분리관점 - 어떤 칼로 쪼갤 것인가?



도메인
관점

업무 중요도와
자치성

--> core/supporting
KPI
→ 마이크로서비스에서
일순위 분리 관점



기술 관점

기술 아키텍처가 상이한
모듈

→ 데이터베이스, 플랫폼, 언어가
상이한



트랜잭션 관점

트랜잭션 모형이
다른

→ ACID or Eventual
Consistency?



유지관리관
점

변경시 동반되어
수정되어야 하는
영역



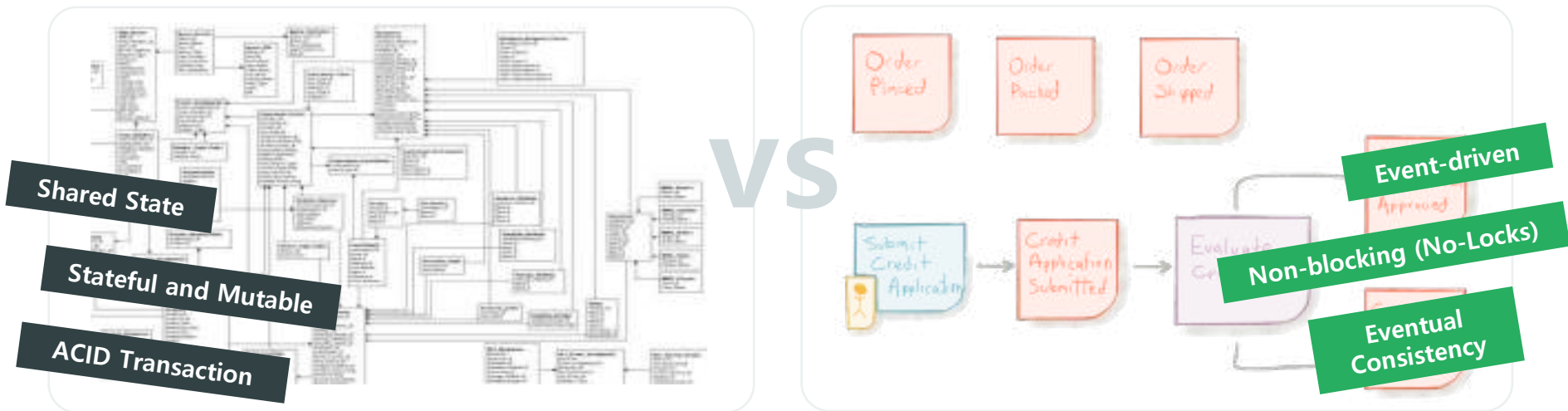
재사용 관점

유틸리티 성격의
서비스

- But,
마이크로서비스에서는
후순위 분리 관점

Event Storming : DDD 를 쉽게 하는 방법

- 이벤트스토밍은 시스템에서 발생하는 이벤트를 중심 (Event-First) 으로 분석하는 기법으로 특히 Non-blocking, Event-driven 한 MSA 기반 시스템을 분석에서 개발까지 필요한 도메인에 대한 탁월하게 빠른 이해를 도모하는데 유리하다.
- 기존의 유즈케이스나 클래스 다이어그램 방식과 다르게 별다른 사전 훈련된 지식과 도구 없이 진행할 수 있다.
- 진행과정은 참여자 워크숍 방식의 방법론으로 결과는 스티키 노트를 벽에 붙힌 것으로 결과가 남으며, 오렌지색 스티키 노트들의 연결로 비즈니스 프로세스가 도출되며 이들을 이후 BPMN과 UML 등으로 정제하여 전환할 수 있다.



Event Storming : Prepare

People

- 모든 팀 및 프로세스, UI, DB, 아키텍처를 책임질 팀당 2명 이상 구성
- 이벤트스토밍 전문가
퍼실리테이터가 초기에는 필요할 수 있음

Tools

- 큰 종이 시트 및 종이 시트를 여러 장 붙일 수 있는 충분한 벽이 있는 넓은 공간
- 여러 종류의 색깔 스티커, 검은 색 펜, 검은색 or 파란색 테이프
- 서서 하는 방식으로 의자 필요 없음.



Type of stickers

Domain
Event
(Orange)

발생한 사실, 결과, Output

도메인 전문가가 정의
이벤트 퍼블리싱



사용자, 페르소나, 스테이크 홀더

유저 인터페이스를 통해 데이터를
소비하고 명령을 실행하여 시스템
과 상호 작용

Definition

정의

도메인에 대한 용어 등의
설명, 기술

Command
(Sky Blue)

의사결정, Input, API, UI 버튼

현재형으로 작성,
행동, 결정 등의 값들에 대한
UI 혹은 API

Aggregate
(Yellow)

구현체 덩어리, 시스템, 모듈

비즈니스 로직 처리의 도메인 객체
덩어리. 서로 연결된 하나 이상의
엔터티 및 value objects의 집합체

Read
Model
(Green)

의사결정에 필요한 자료, View, UI

행위와 결정을 하기 위하여
유저가 참고하는 데이터, 데이터 프로
젝션이 필요 : CQRS 등으로 수집

External
System
(Pink)

외부 시스템

시스템 호출을 암시 (REST)

Policy
(Lilac)

정책, 반응(Reaction)

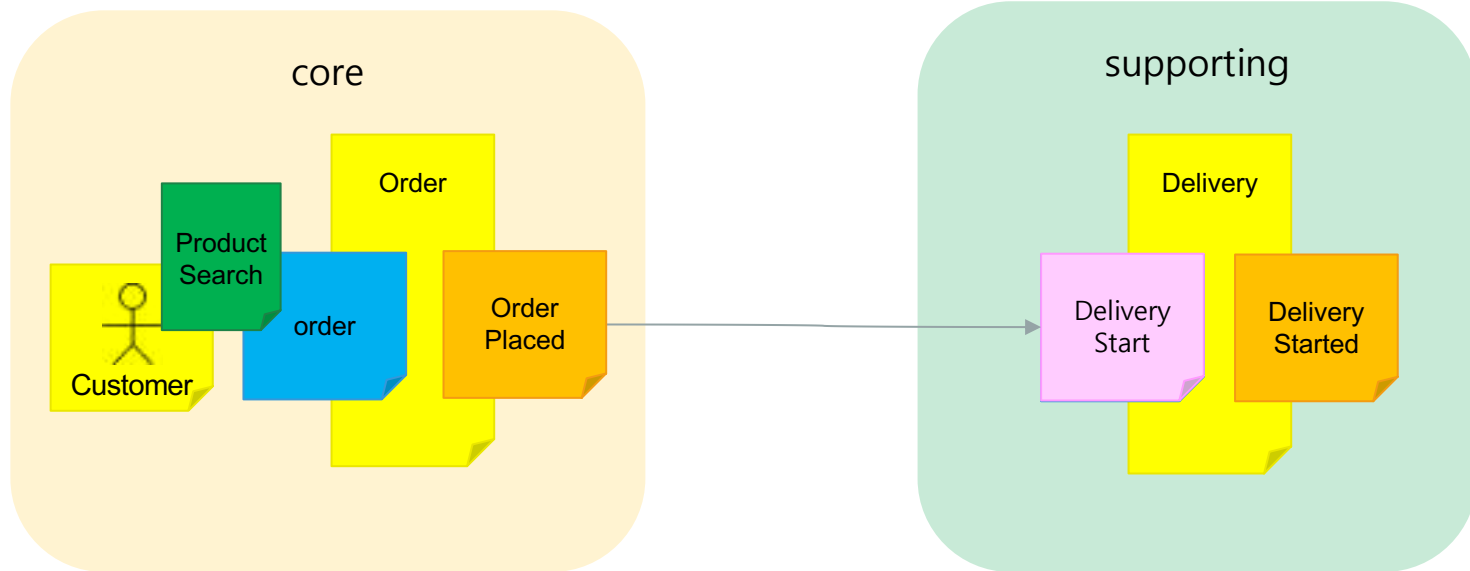
이벤트에 대한 반응
(서브스크라이브)
비즈니스 룰 엔진 등

Comment
Or
Question
(Purple)

의견 또는 질문

추기적인 내용 입력,
예측되는 Risk

Examples



Firstly, Event Discovery

PO가 주도
(업무전문가)



Account
Created

Email
Sent

Payment
Details
Confirmed

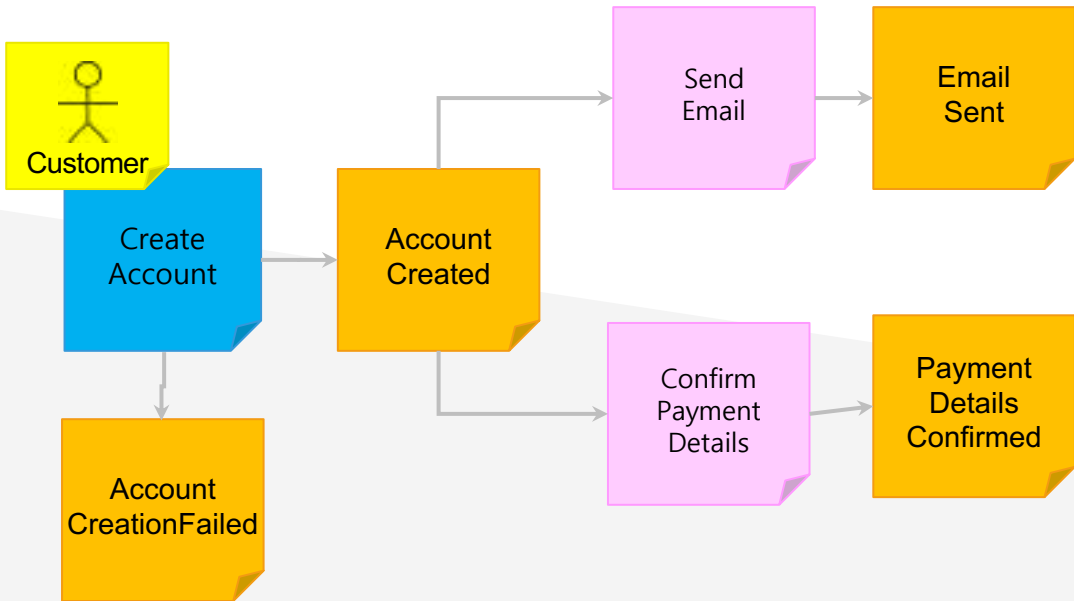
- 오렌지(주황색) 스티커 사용
- 각 도메인 전문가들이 개별 도메인 이벤트 목록 작성
- 이벤트는 도메인 전문가와 비즈니스 관계자 서로 이해할 수 있는 의미 있는 방식(유비쿼터스 랭귀지)으로 표현하고 동사의 과거형 (p.p.)으로 표현한다.
- 시작 및 종료 이벤트를 식별하고 스티커를 붙일 벽의 시작과 끝의 타임라인에 배치
- 이벤트를 페르소나와 관련시키는 방법에 대해 논의
- 중복된 이벤트를 발견하면 중복된 이벤트를 벽에서 제거
- 불분명한 경우 다른 색상의 스티커 메모를 사용하여 질문이나 의견을 추가(빨간색 스티커)
- 이벤트에 대해서 동사를 과거 시제로 입력하고 다른 이벤트와 명확하게 구분되는 용어를 사용

Command, Policy, Actor 도출

PO + UI/UX 가 주도
(업무전문가)



- 하늘색 or 라일락 색의 스티커 사용
- Command 는 사용자의 의사결정에 의하여 (UI) 결과이벤트에 이르게 하는 Input
- Command 는 어떠한 상태의 변화를 일으키는 서비스*
- Policy는 이벤트가 발생한 후 발생하는 반응형 논리 (Input)
- Policy는 결과로서 Event 를 트리거 할 수 있고, 다른 Command를 호출 할 수도 있음
- Policy는 시스템에 의하여 자동화 될 수 있다.



* Command 라는 용어는 CQRS 에서 유래했으며, 쓰기서비스인 Command 와 읽기행위인 Query는 구분됨.

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

Aggregate 도출

- 노란색 스티커 사용
- 같은 Entity를 사용하는 연관 있는 도메인 이벤트들의 집합
- 관련 데이터 (Entity 및 value objects)뿐만 아니라 해당 Aggregates의 Life Cycle에 의해 연결된 작업(Command)으로 구성

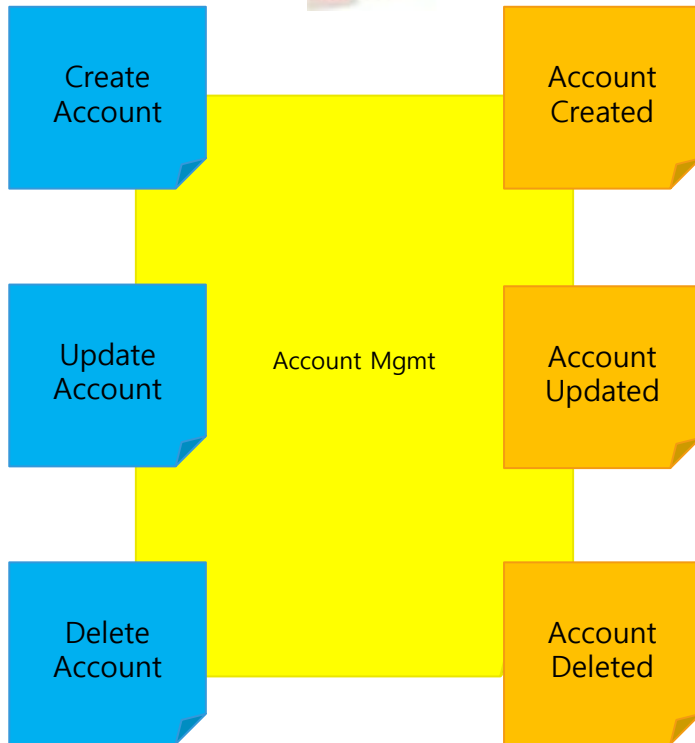
도메인 이벤트가 발생하는 데는, 어떠한 도메인 객체의 변화가 발생했기 때문이다.
하나의 ACID 한 트랜잭션에 묶여 변화되어야 할 객체의 묶음을 도출하고, 그것들을 커맨드, 이벤트와 함께 묶는다.

아키텍트/개발자/DBA
주도



Inputs

Outputs



Bounded Contexts 도출

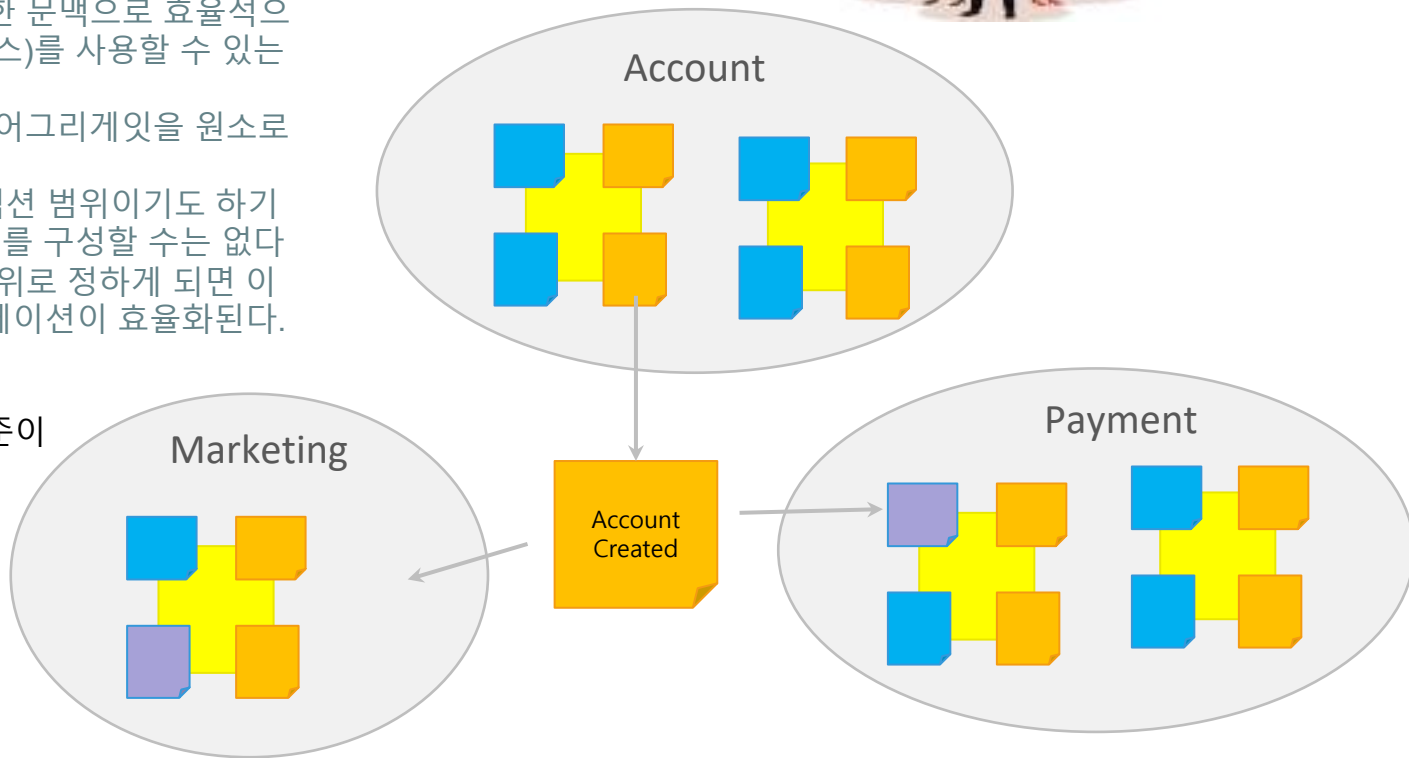
다같이



- Bounded Context 는 동일한 문맥으로 효율적으로 업무 용어 (도메인 클래스)를 사용할 수 있는 범위를 뜻한다.
- 하나의 BC 는 하나이상의 어그리게잇을 원소로 구성될 수 있다.
- 어그리게잇은 ACID 트랜잭션 범위이기도 하기 때문에 이를 더 쪼개서 BC 를 구성할 수는 없다
- BC를 Microservice 구성단위로 정하게 되면 이를 담당하는 팀 내의 커뮤니케이션이 효율화된다.

찾는 방식은 여러가지 기준이 있다

- Domain Boundary
- Transaction Boundary
- Technical Stack
- ...



Lab Time – 12 번가 예제 – 최초 Event Discovery

- Customers search products and clicks the order button to placed an order.
- When an order is placed, the delivery team prepares for delivery of the product and the delivery will start.
- When delivery has been started, the product inventory is decreased.
- Customers can cancel their orders.
- When an order has been canceled, delivery belongs to the order must be canceled as well.
- When delivery has been canceled, the product inventory is increased.

OrderPlaced

DeliveryStarted



InventoryDecreased

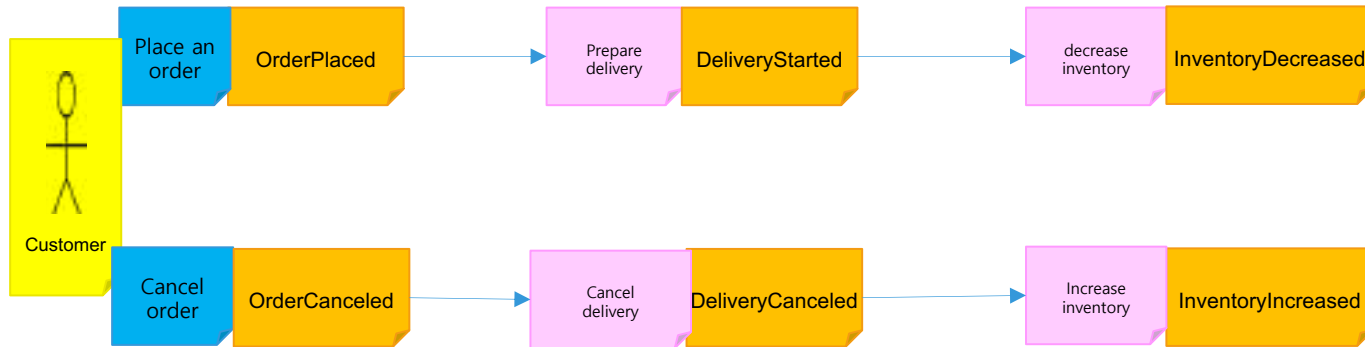
OrderCanceled

DeliveryCanceled

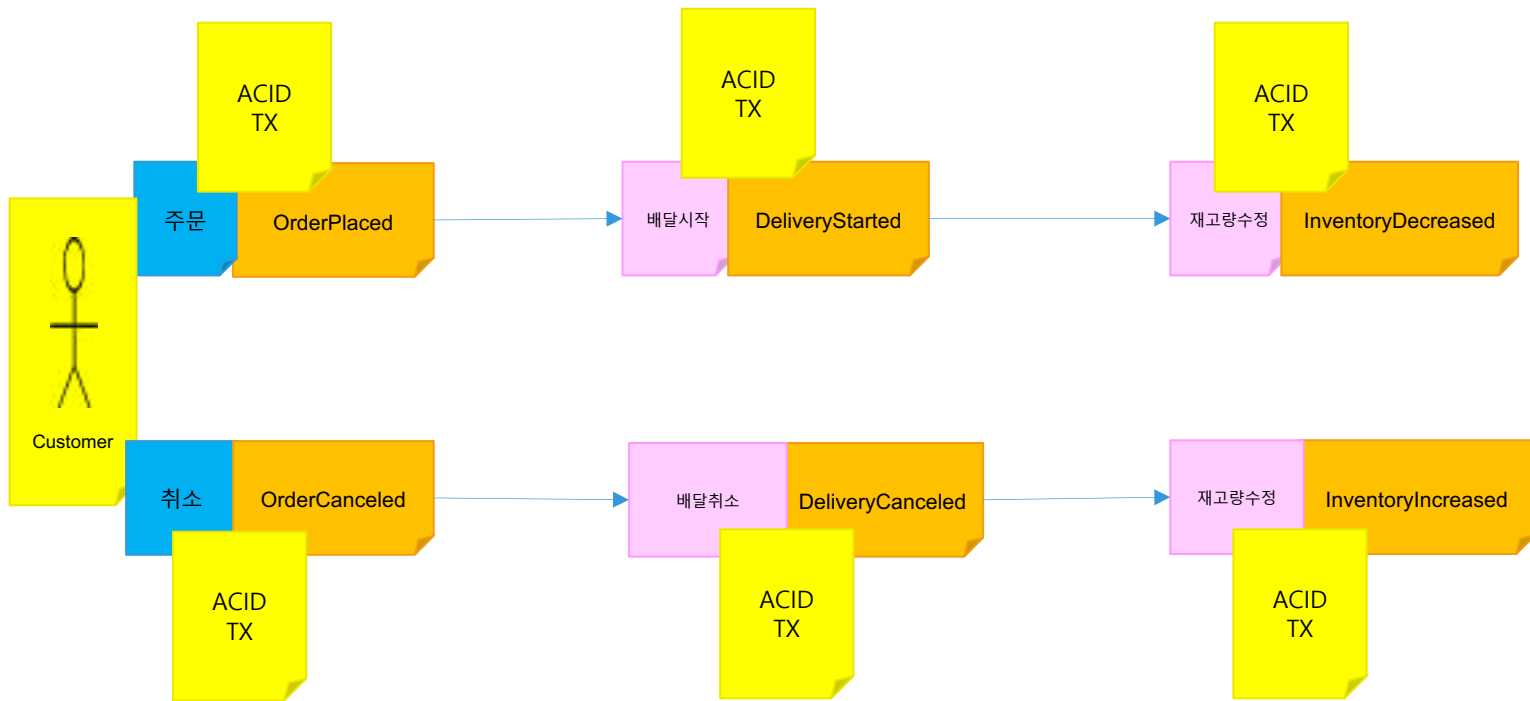
InventoryIncreased

Lab Time – Command, Policy, Actor

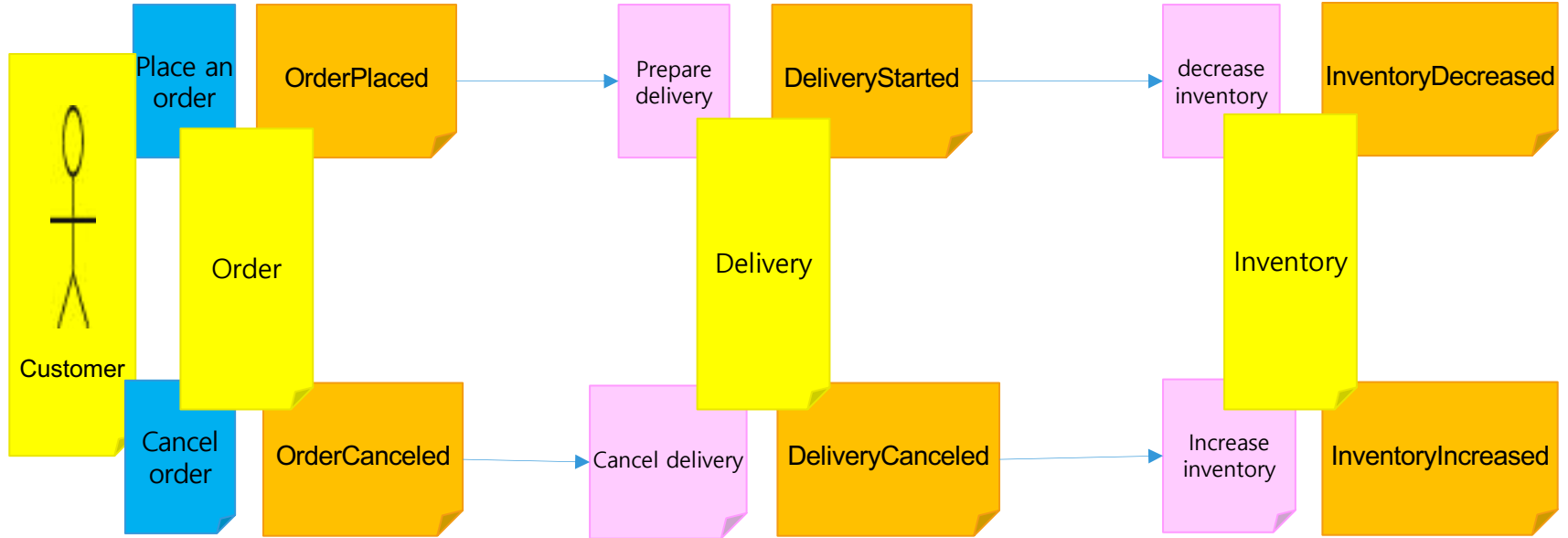
-  **Customers** search products and clicks the order button to [placed an order](#).
- When an [order is placed](#), the [delivery team prepares for delivery](#) of the product and [the delivery will start](#).
- [When delivery has been started, the product inventory is decreased](#).
-  **Customers** can [cancel their orders](#).
- [When an order has been canceled, delivery belongs to the order must be canceled as well](#).
- [When delivery has been canceled, the product inventory is increased](#).



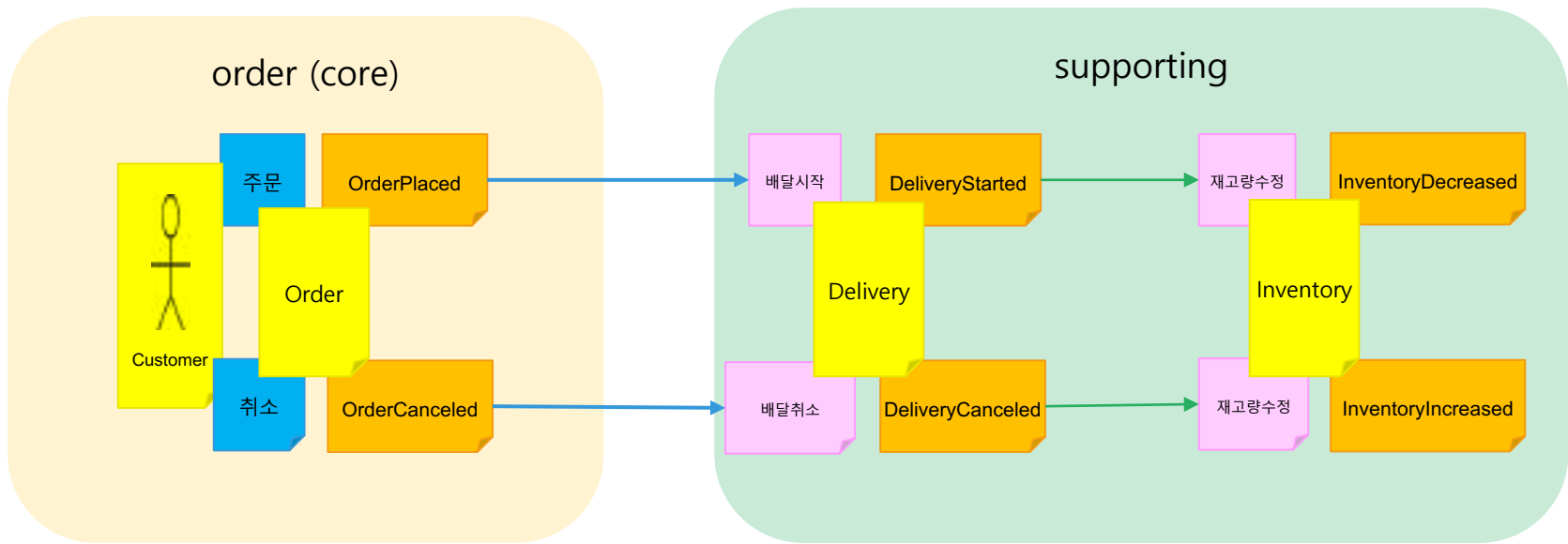
Lab Time – Aggregate



Lab Time – Aggregate (2)



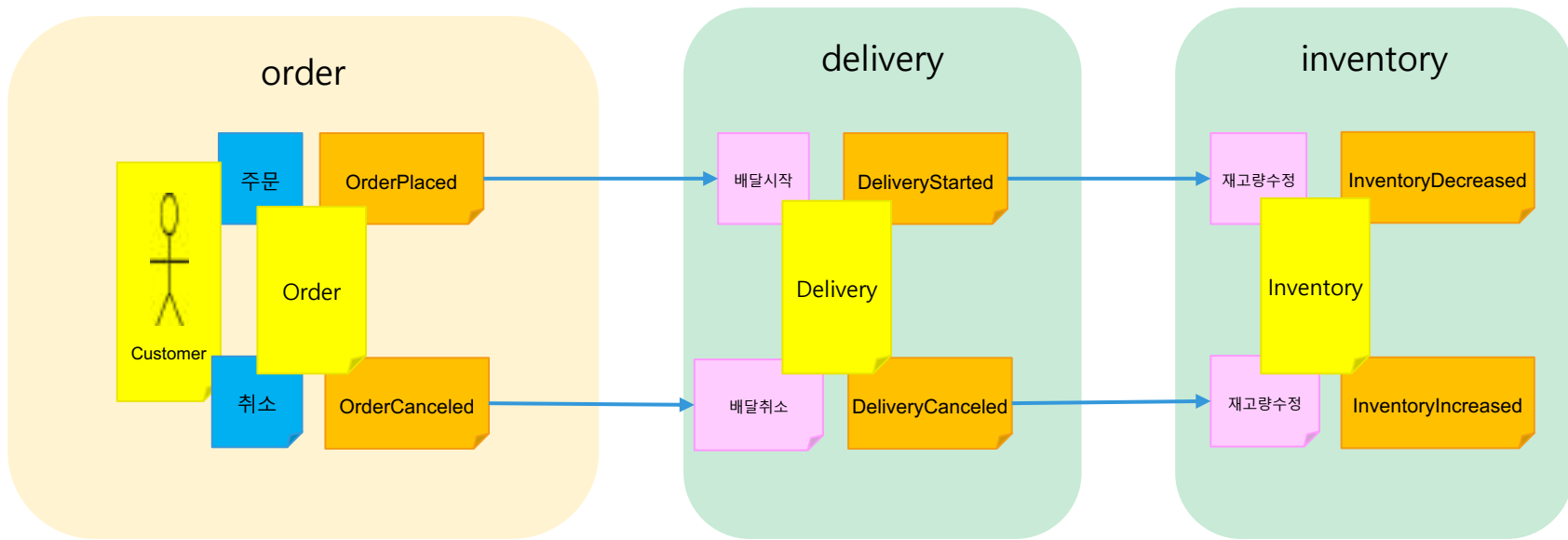
Lab Time – Bounded Context 분리



—————→
PubSub via Event Stream
(e.g. Kafka, Axon Server)

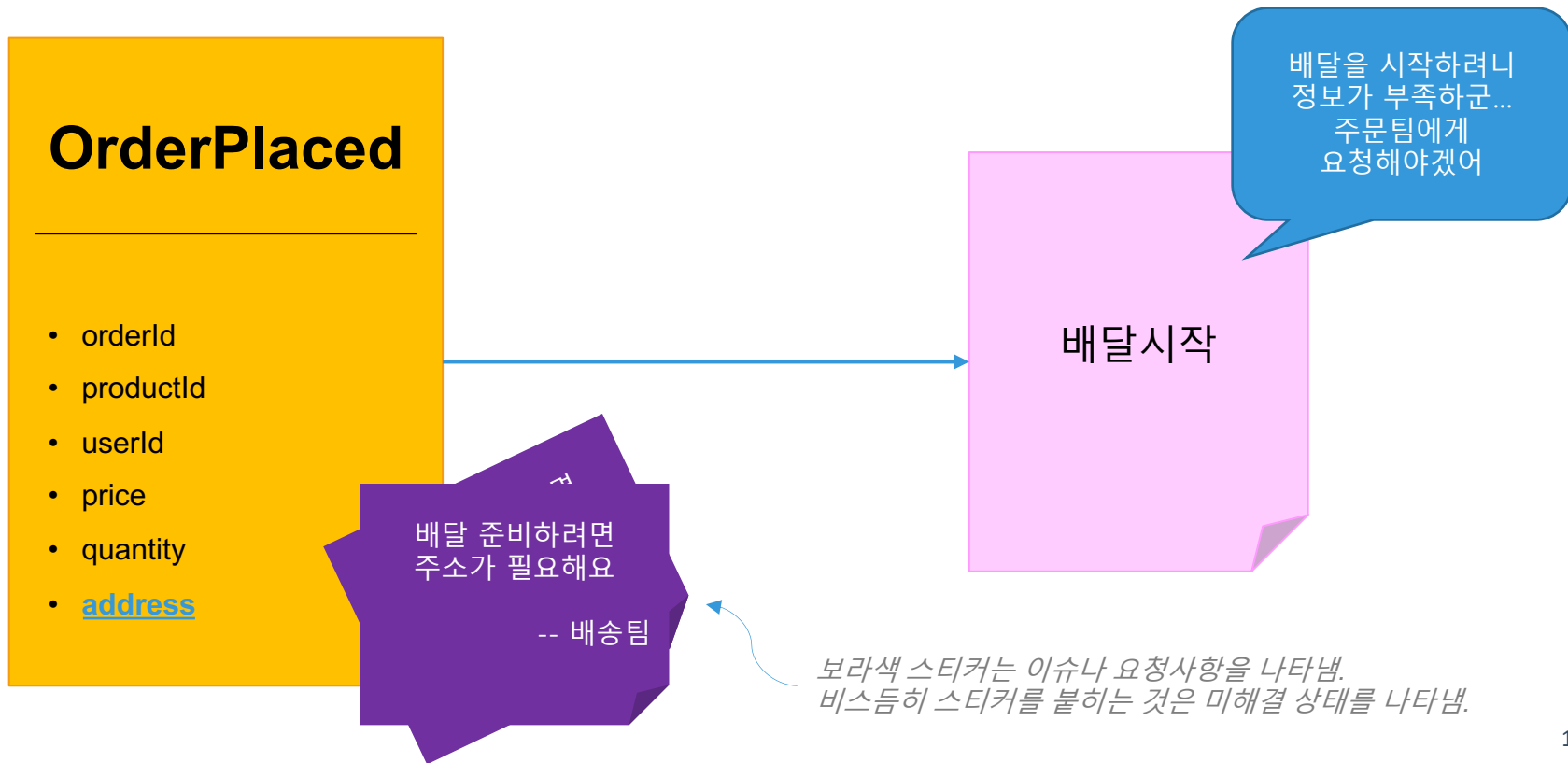
—————→
PubSub via Local Event Design Pattern
(e.g. White-board Pattern)

Lab Time – Bounded Context 분리 (다음스프린트)

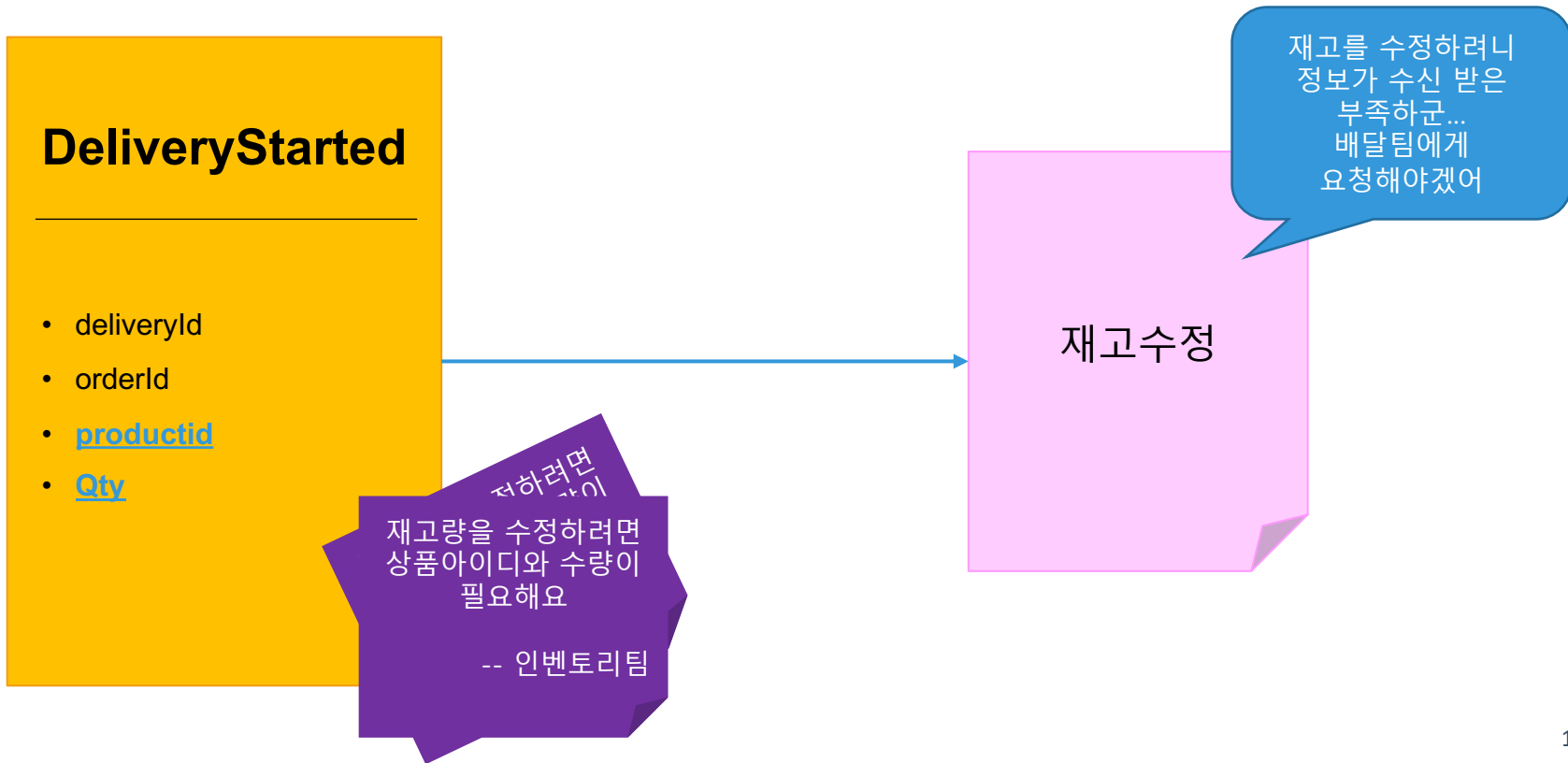


→
PubSub via Event Stream
(e.g. Kafka, Axon Server)

Lab Time – 도메인 이벤트의 속성 선언



Lab Time – 도메인 이벤트의 속성 선언(2)



Quiz. 다음중 도메인 이벤트로 부적절한 것은?

O

상품주문이 발생함

다른 팀에서 관심 가질만함

상품이 입고됨

다른 팀에서 관심 가질만함

상품정보 변경됨

적당한 사이즈임

X

상품주문

It's a command

상품을 조회함

It doesn't make any state change

JPA 를 통해 상품 정보를 Update 함

Too technical and too fine-grained

Be business level

Nobody will be interested in this event

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. 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

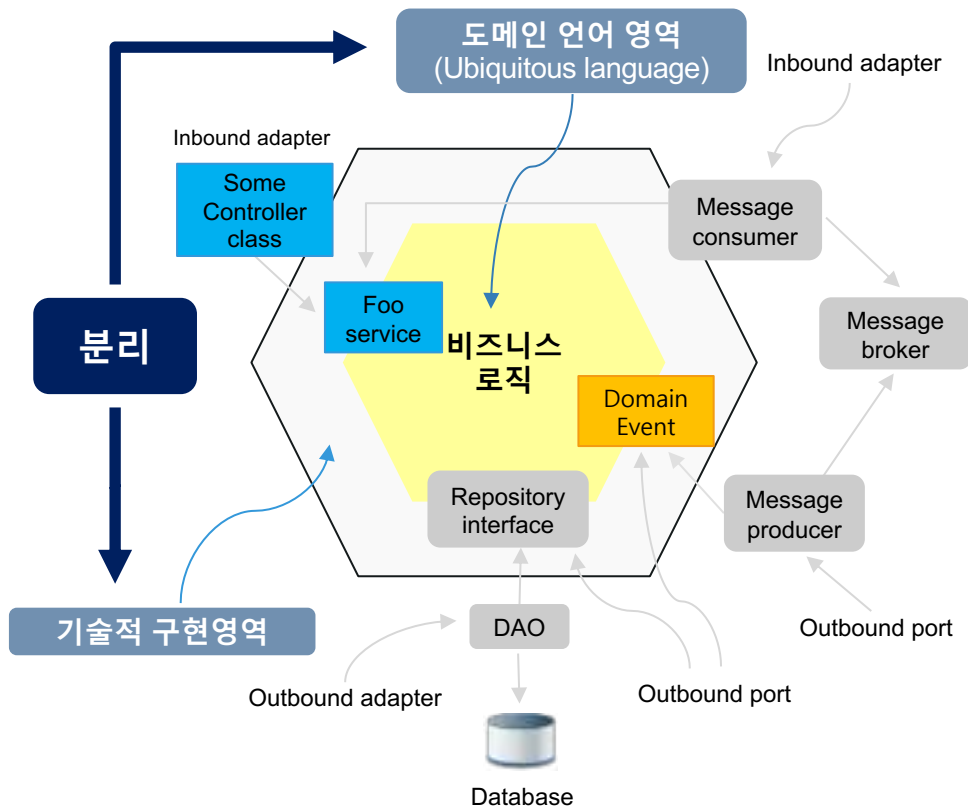
Microservice Implementation Pattern (1)

- 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>



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

이벤트 스토밍 결과에서 구현 기술 연동

<u>요소</u>	<u>구현체</u>	<u>미들웨어/프레임워크</u>
이벤트 (Domain Event)	<ul style="list-style-type: none">• 도메인 이벤트 클래스 (POJO)	<ul style="list-style-type: none">• 카프카 퍼블리시• Rabbit MQ
커맨드 (Command)	<ul style="list-style-type: none">• 서비스 객체• 레포지토리 객체 (PagingAndSortingRepository)	<ul style="list-style-type: none">• Spring Data REST• RESTeasy
결합물 (Aggregate)	<ul style="list-style-type: none">• 엔티티 클래스 (ORM)• 도메인 모델	<ul style="list-style-type: none">• JPA Entity• Value Objects
정책 (Policy)	<ul style="list-style-type: none">• 엔티티 클래스의 Hook에 정책 호출 구문 주입• CDC 를 통한 이벤트 후크	<ul style="list-style-type: none">• 카프카 Event Listening Code• JPA Lifecycle Hook• CDC (Change Data Capturing)
바운디드 컨텍스트	<ul style="list-style-type: none">• 마이크로 서비스 (후보)	<ul style="list-style-type: none">• Spring Boot

분석/설계 (Sticker)

Order

- orderId
- productId
- userId
- price
- quantity
- telephone



Aggregate Root

```
@Entity
public class Order{

    @Id Long id;
    Long productId;
    String customerId;
    Money price;
    int quantity;

    ... setter/getters ...

}
```

Aggregate Members

```
@Entity
public class OrderDetail {

    ....

    ... setter/getters ...

}
```

```
//Value Objects
public class Money {

    ....

    ... setter/getters ...

}
```

주문
(POST)

주문수정
(PATCH)

주문삭제
(DELETE)

Command CRUD 에 해당하면? Repository Pattern 으로 자동생성

```
public interface OrderRepository extends ↓       ↓  
    PagingAndSortingRepository<Order, Long>{  
    // 비워둠  
}
```

Command Repository Pattern 이 안될시에 MVC 패턴으로 구현

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


분석/설계 (Sticker)

OrderPlaced

- orderId
- productId
- userId
- price
- quantity
- ~~Telephone~~
- timestamp



개발 (POJO)

```
public class OrderPlaced{  
  
    Long orderId;  
    Long productId;  
    String userId;  
    double price;  
    int quantity;  
  
    ... setter/getters ...  
  
}
```



실행 (JSON)

```
{  
  type: "OrderPlaced",  
  name: "캠핑의자",  
  userId : "1@uengine.org",  
  orderId: 12345,  
  price: 100  
  quantity : 10  
}
```

Order

- orderId
- productId
- userId
- price
- quantity
- Telephone

- publishOrderPlaced()



Event POJO Class 와 이벤트 발사로직

```
@Entity
public class Order {
    ...

    @PostPersist // 주문이 저장된 후에
    private void publishOrderPlaced() {
        OrderPlaced orderPlaced = new OrderPlaced(); // 주문이 들어온 사실을
        이벤트로 작성

        orderPlaced.setOrderId(id);
        ...

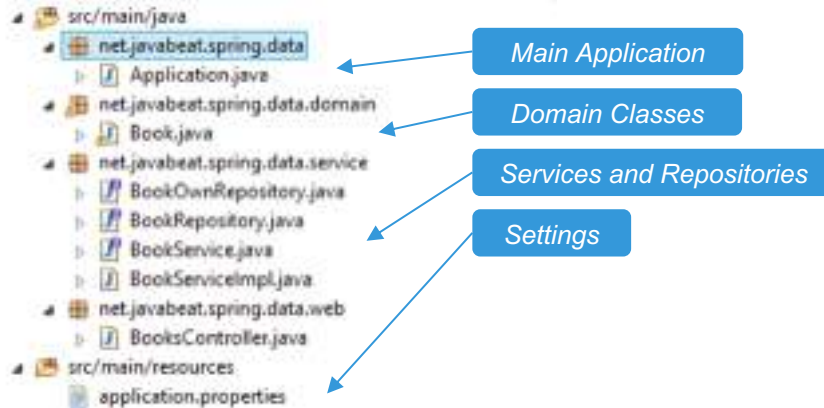
        orderPlaced.publish(); // 메시지 큐에 주문이 들어왔음을 신고
    }
}
```

(Whenever OrderPlaced)

배송을 준비함

```
@KafkaListener(topics = "shopping")  
public void onOrderPlaced(OrderPlaced orderPlaced)  
  
    deliveryService.start(orderPlaced.getId());  
}
```

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

The screenshot shows the start.spring.io web application. The browser address bar displays 'start.spring.io'. The page has a sidebar on the left with the following sections: 'Spring initializer bootstrap your application', 'Project' (with 'Maven Project' selected), 'Language' (with 'Java' selected), 'Spring Boot' (with '2.2.0.M6', '2.2.0 (SNAPSHOT)', '2.1.6 (SNAPSHOT)', and '2.1.6' selected), 'Project Metadata' (with 'Group' set to 'com.12st', 'Artifact' set to 'order', and 'Options' expanded), and 'Dependencies' (with '0' selected). The main area shows 'Search dependencies to add' with results for 'H2 Database', 'Rest Repositories', and 'Spring Data JPA'. At the bottom, there are buttons for 'Generate the project' and 'Explore the project'.

Go to start.spring.io

Set metadata:

- Group: com.12st
- Artifact: order
- 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)
```

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"  
    }  
  }  
}
```



HTTP = success



HATEOAS links

Aggregate Root ? Entity Class 작성

상품

```
@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<Order> orders;
}
```

주문

```
@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;
}
```

배송

```
@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 API Implementation by Spring Data REST Repository

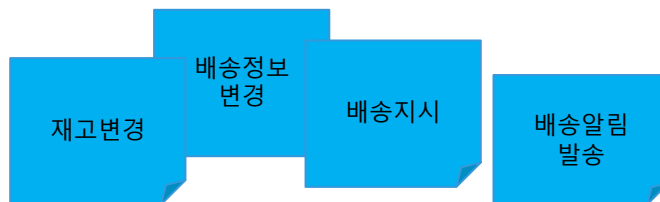
비즈니스 객체 (명사) 에서 추출 가능한 CRUD 의 경우



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

비즈니스 프로세스 (동사) 에서 추출 가능한 경우

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



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

Main Class

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

Test Generated Services

```
$ http localhost:8088
{
  "_links": {
    "deliveries": {
      "href": "http://localhost:8088/deliveries{?page,size,sort}",
      "templated": true
    },
    "orders": {
      "href": "http://localhost:8088/orders{?page,size,sort}",
      "templated": true
    },
    "products": {
      "href": "http://localhost:8088/products{?page,size,sort}",
      "templated": true
    },
    "profile": {
      "href": "http://localhost:8088/profile"
    }
  }
}
```



Create a new Order

\$ http localhost:888/orders productId=1 quantity=3 customerId="1@uengine.org"
customerName="홍길동" customerAddr="서울시"

```
{
  "_links": {
    "delivery": {
      "href": "http://localhost:8088/orders/1/delivery"
    },
    "order": {
      "href": "http://localhost:8088/orders/1"
    },
    "product": {
      "href": "http://localhost:8088/orders/1/product"
    },
    "self": {
      "href": "http://localhost:8088/orders/1"
    }
  },
  "customerAddr": "서울시",
  "customerId": "1@uengine.org",
  "customerName": "홍길동",
  "price": 10000,
  "productId": 1,
  "productName": "TV",
  "quantity": 3,
  "state": "OrderPlaced"
}
```



URI of the new order

Create a delivery for the order

\$ http **http://localhost:8088/orders/1/delivery**

```
{
  "_links": {
    "delivery": {
      "href": "http://localhost:8088/deliveries/1"
    },
    "order": {
      "href": "http://localhost:8088/deliveries/1/order"
    },
    "self": {
      "href": "http://localhost:8088/deliveries/1"
    }
  },
  "customerId": "1@uengine.org",
  "customerName": "홍길동",
  "deliveryAddress": "서울시",
  "deliveryState": "DeliveryStarted",
  "productName": "TV",
  "quantity": 3
}
```



URI of the delivery service



Quiz

스프링 부트/클라우드에 대한 설명 중 옳은 것은?

1. WAS 가 필요없이 자체로서 웹서버 기능을 포함한다
2. 구성의 단순성을 위하여 복잡한 설정을 최대한 배제하였다
3. 마이크로서비스를 구현하기 용이한 디자인 패턴들을 다양하게 제공하고 있다
4. 1,2,3 모두 옳다
5. 1,2,3 모두 틀렸다

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. 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 ✓



DevOps

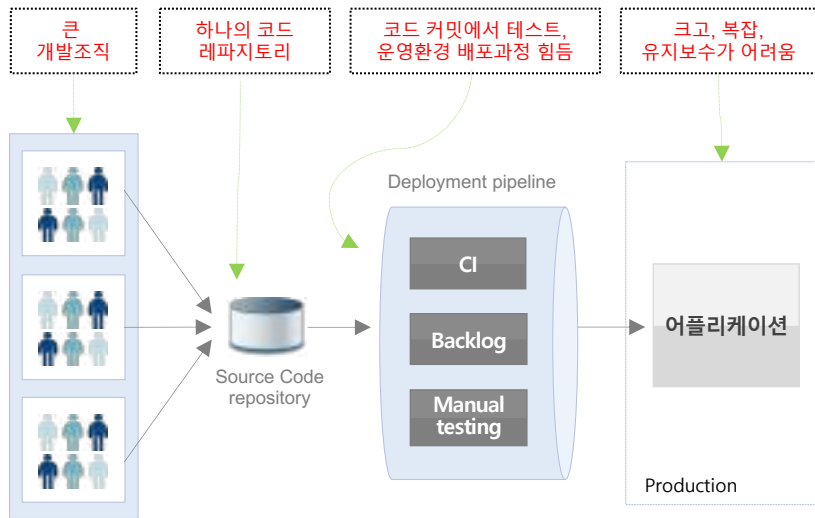
- DevOps Process and Tools
- Deploy Strategies
- Containers and Orchestrators
- Google Cloud Platform

https://www.youtube.com/watch?v=_l94-tJlovq

Process Change : 열차말고 택시를 타라!

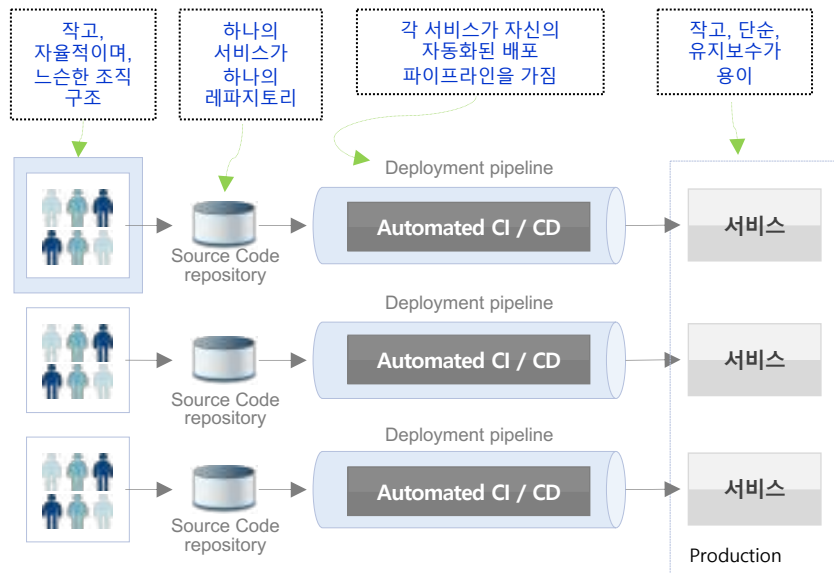
모노리식 개발 및 운영환경

- 모노리식 환경하에서는 큰 개발팀이 하나의 소스코드 레파지토리에 변경 사항을 커밋하므로 코드간 상호 의존도가 높아 신규 개발 및 변경이 어려움
- 작은 변경에도 전체를 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

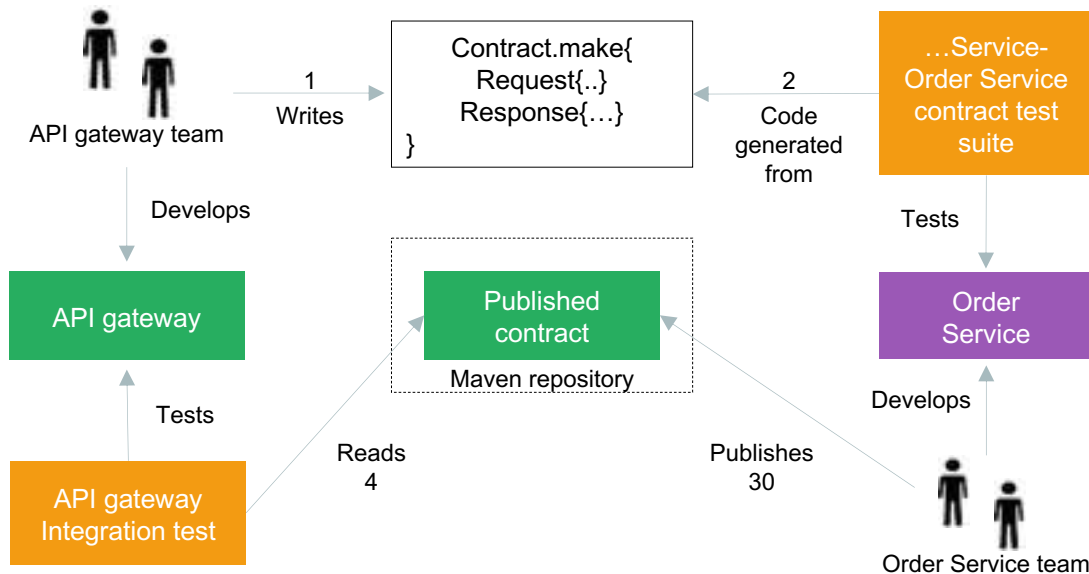


Process Change : Consumer-driven Test

MSA 서비스의 테스트

컨트랙트 테스트는 서비스 수요자가 주도하여 테스트를 작성한다.

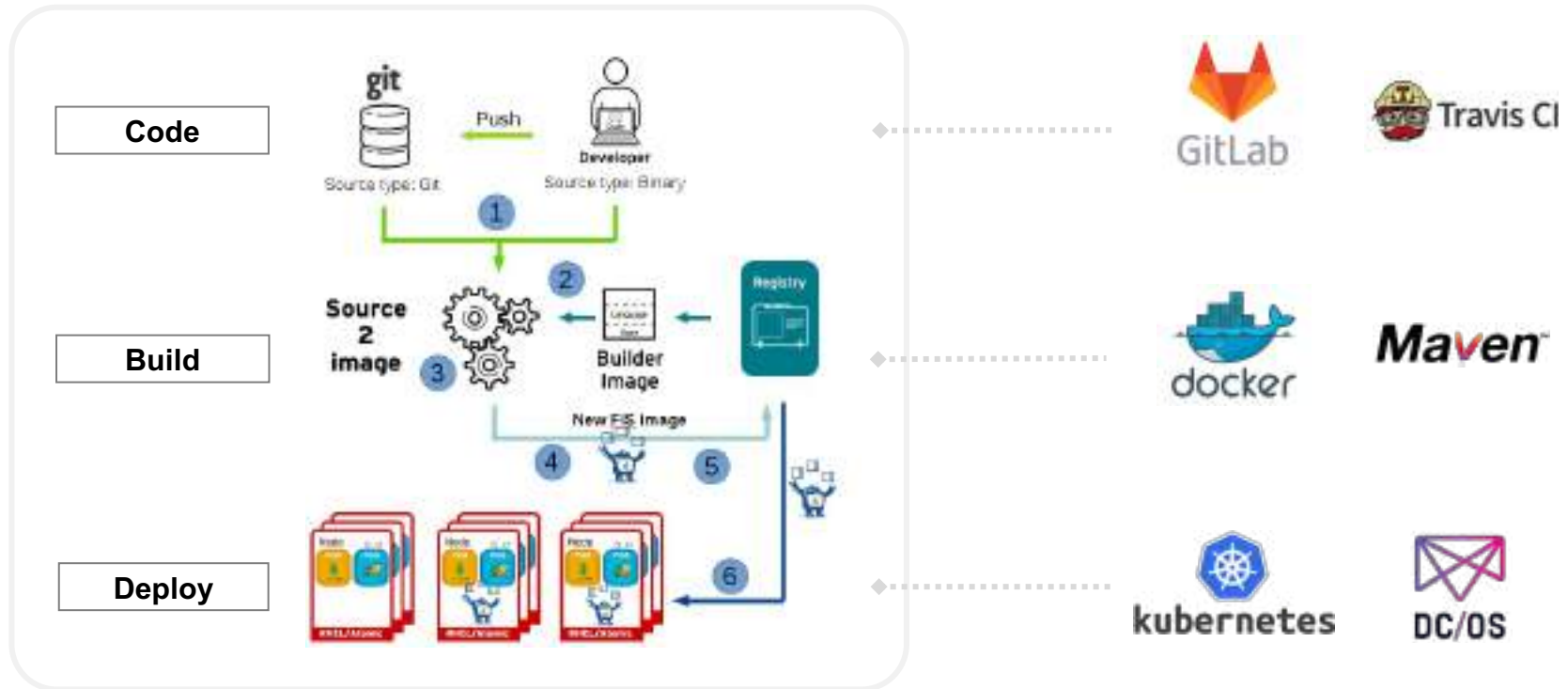
수요자가 테스트를 작성하여 제공자의 레포지토리에 Pull-Request 하면, 제공자가 이를 Accept 한후에 제공자측에서 테스트가 생성되어 테스트가 벌어진다.



특징

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

DevOps toolchain



Facebook

2. 배포 주기

- 매일 마이너 업데이트
- 메이저 업데이트 (매주 화요일 오후)



3. Deployment Pipeline

출처: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6449236>

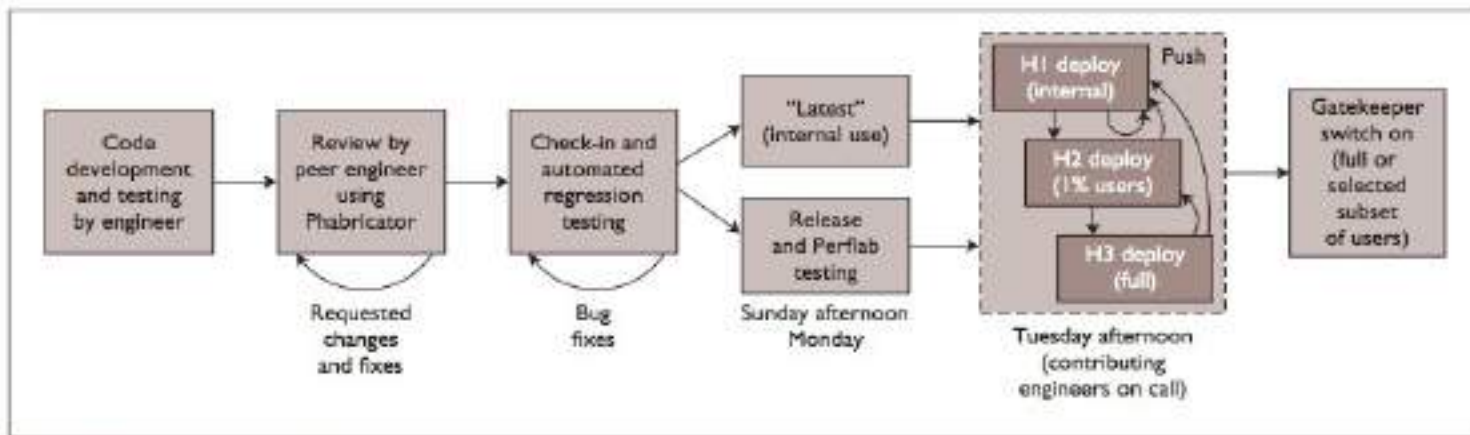


Figure 8. The Facebook deployment pipeline. Multiple controls exist over new code.

Netflix

4. Canary를 통해서 확신 갖기

- Canary란? 실제 production 환경에서 small subset에서 새로운 코드를 돌려 보고 옛날 코드와 비교해서 새로운 코드가 어떻게 돌아가는 지 보는 것
- Canary 분석 작업(HTTP status code, response time, 실행수, load avg 등이 옛날 코드랑 새로운 코드랑 비교해서 어떻게 다른 지 확인하는 것)은 1000개 이상의 metric을 비교해서 100점 만점에 점수를 주고 일정 점수일 경우만 배포할 수 있음.



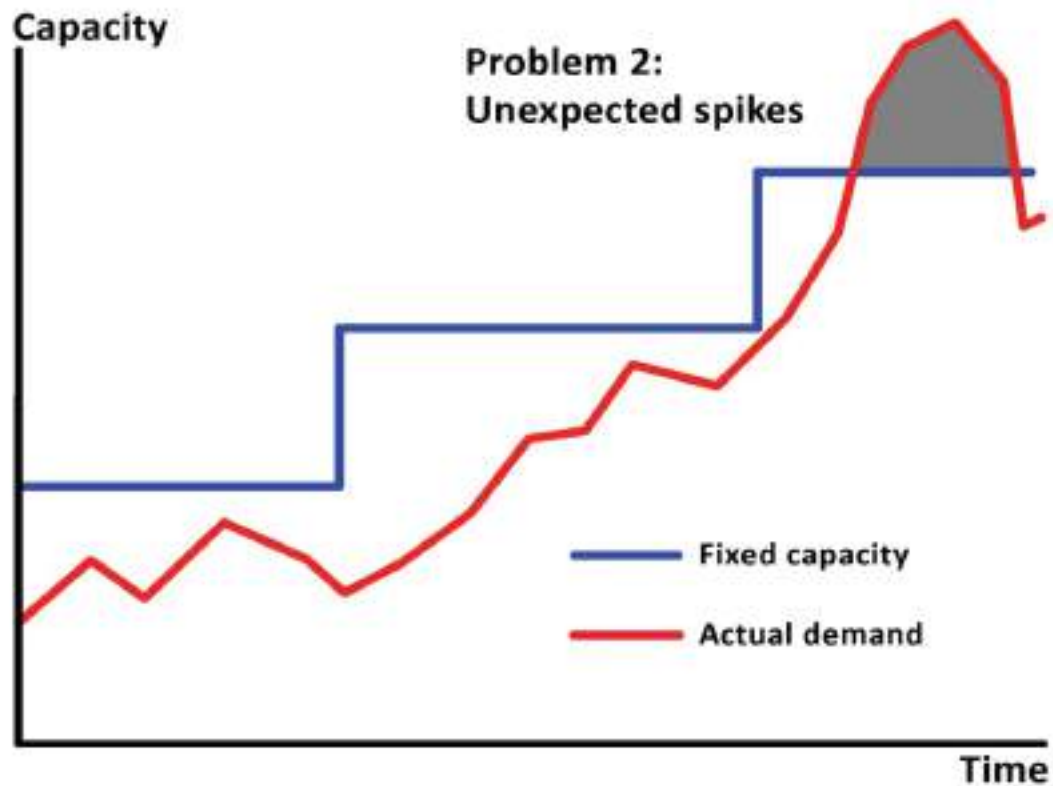
Canary Score

AMI Name: ami-4219582b Date: Tue Aug 13 23:31:27 UTC 2013 -1 Hours

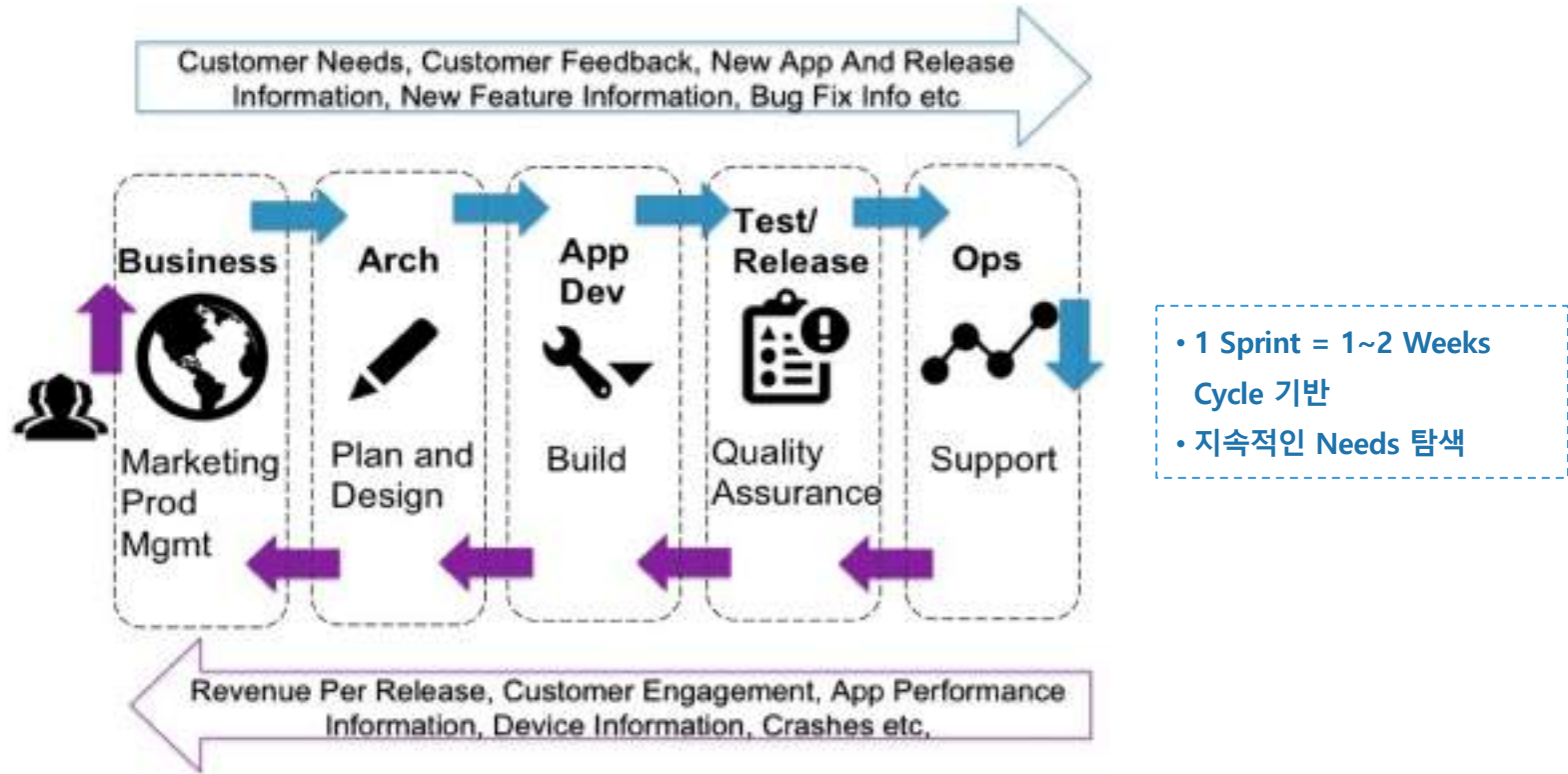


출처: <http://techblog.netflix.com/2013/08/deploying-netflix-api.html>

Managing Scalability



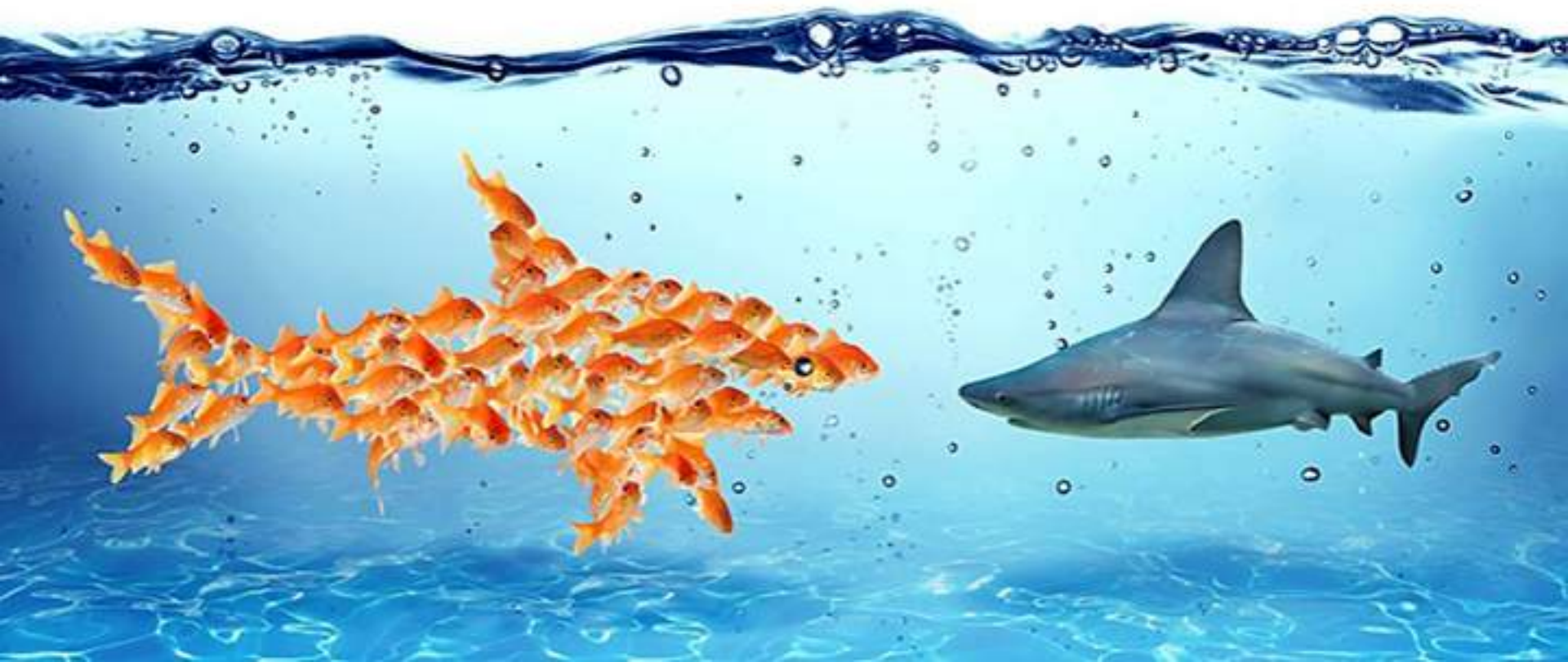
Approach #3: BizDevOps Process & Infra



“

서비스를 죽지 않게 하려면...

”



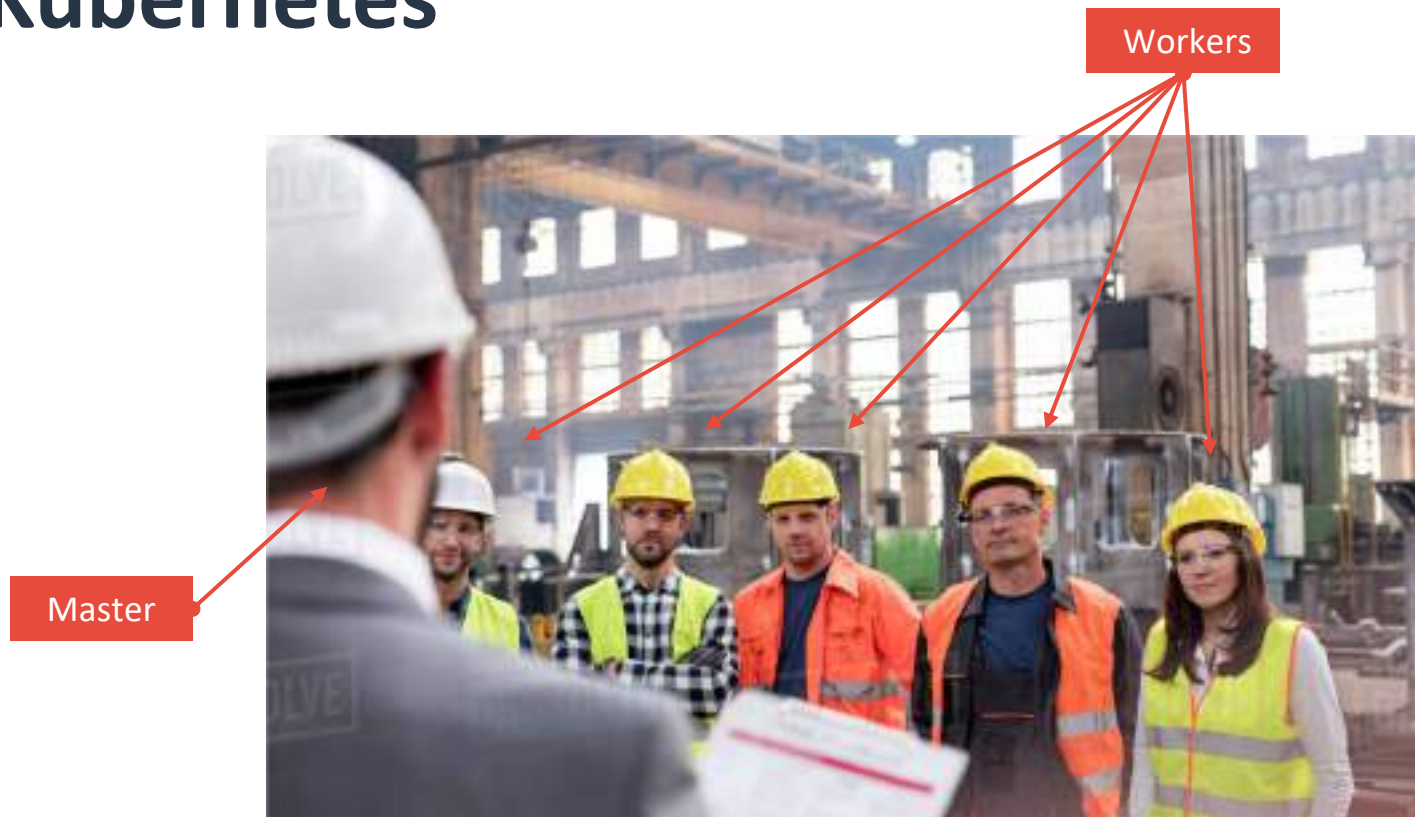
“

항상성이 (Self-Healing) 자동으로 유지되면 어떨까?

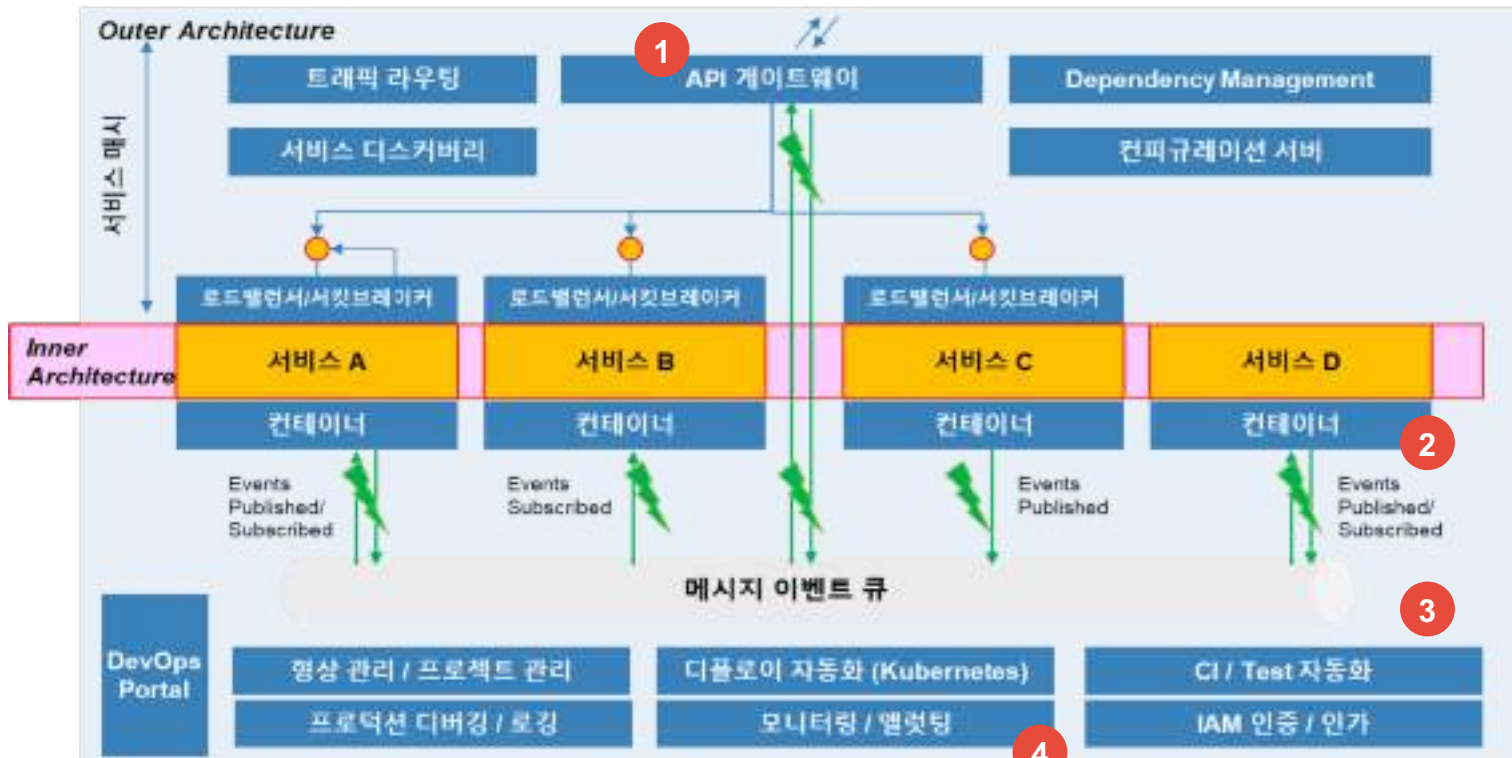
”



Kubernetes

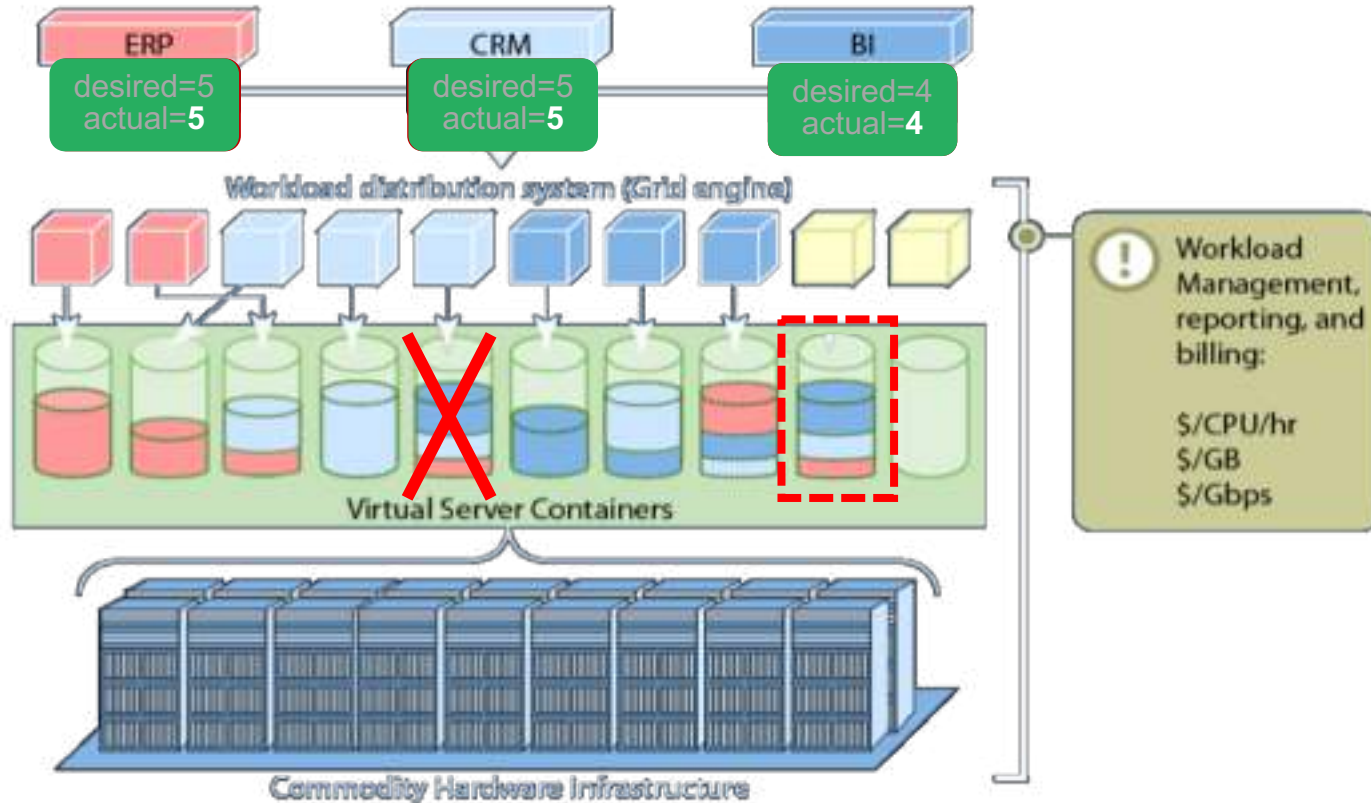


Outer & Inner Architecture

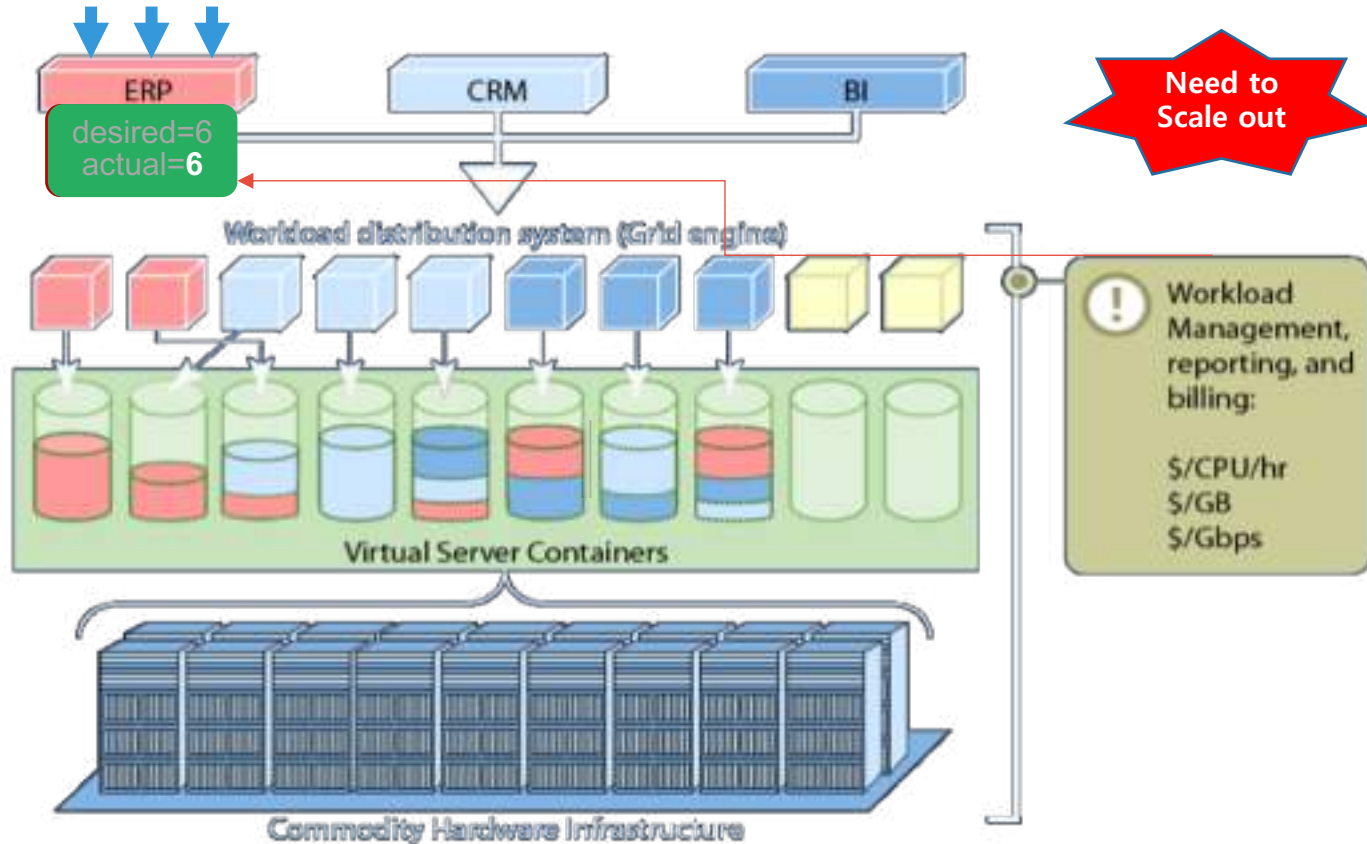


1. API Gateway : API를 사용하여 통신하는 모든 서비스들의 인증 및 제어, 트래픽 모니터링 등 수행
2. Managed Container System : 서비스들의 중앙 관리 및 인증, 라우팅등의 기능 수행
3. Backing Services : 스토리지(Block, Object), Cache, Message Queue 등의 기능 영역
4. Observability Services : 서비스들의 이벤트 모니터링 및 알림, 로그 취합, 진단 등 수행

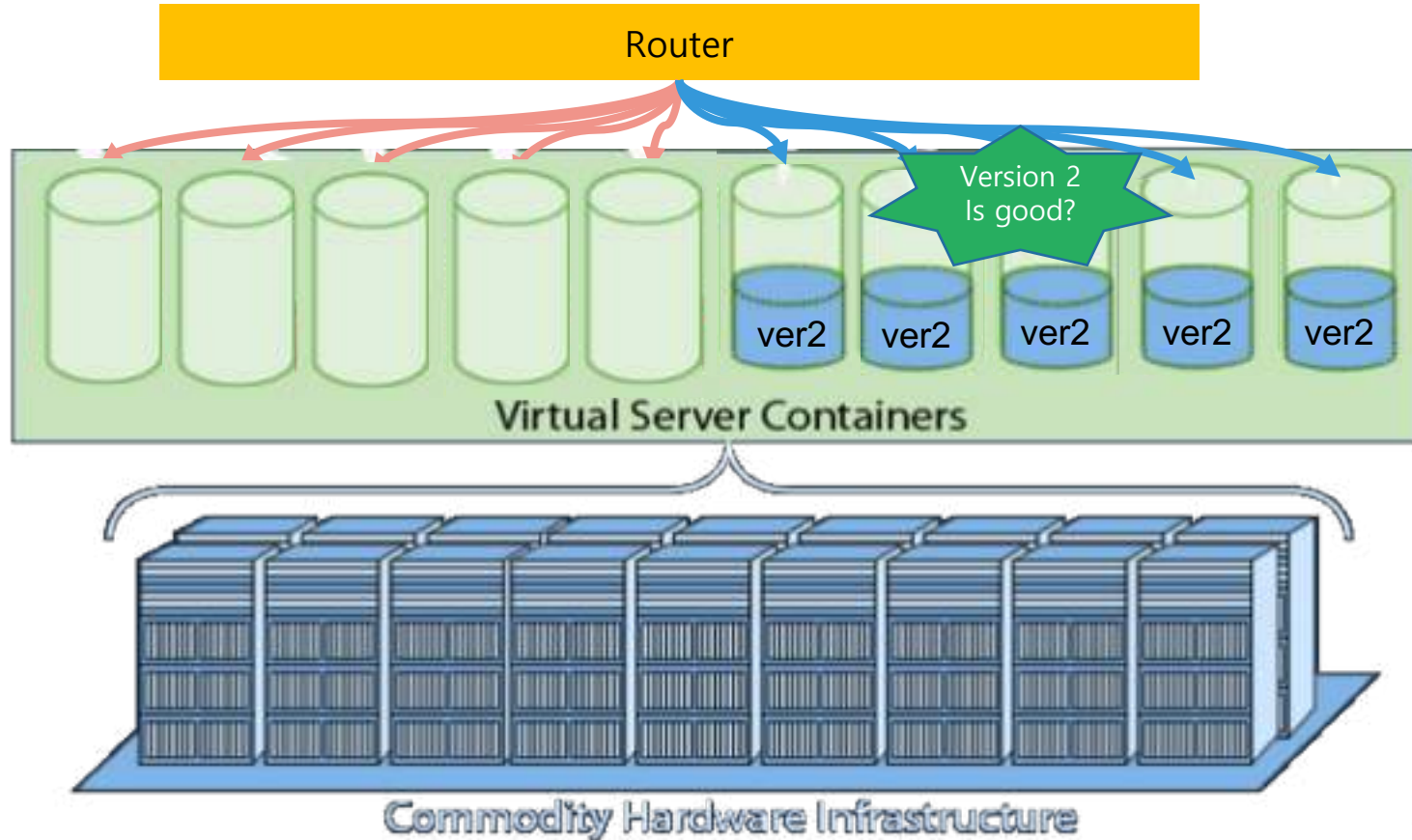
Container Orchestration – Self Healing Mechanism



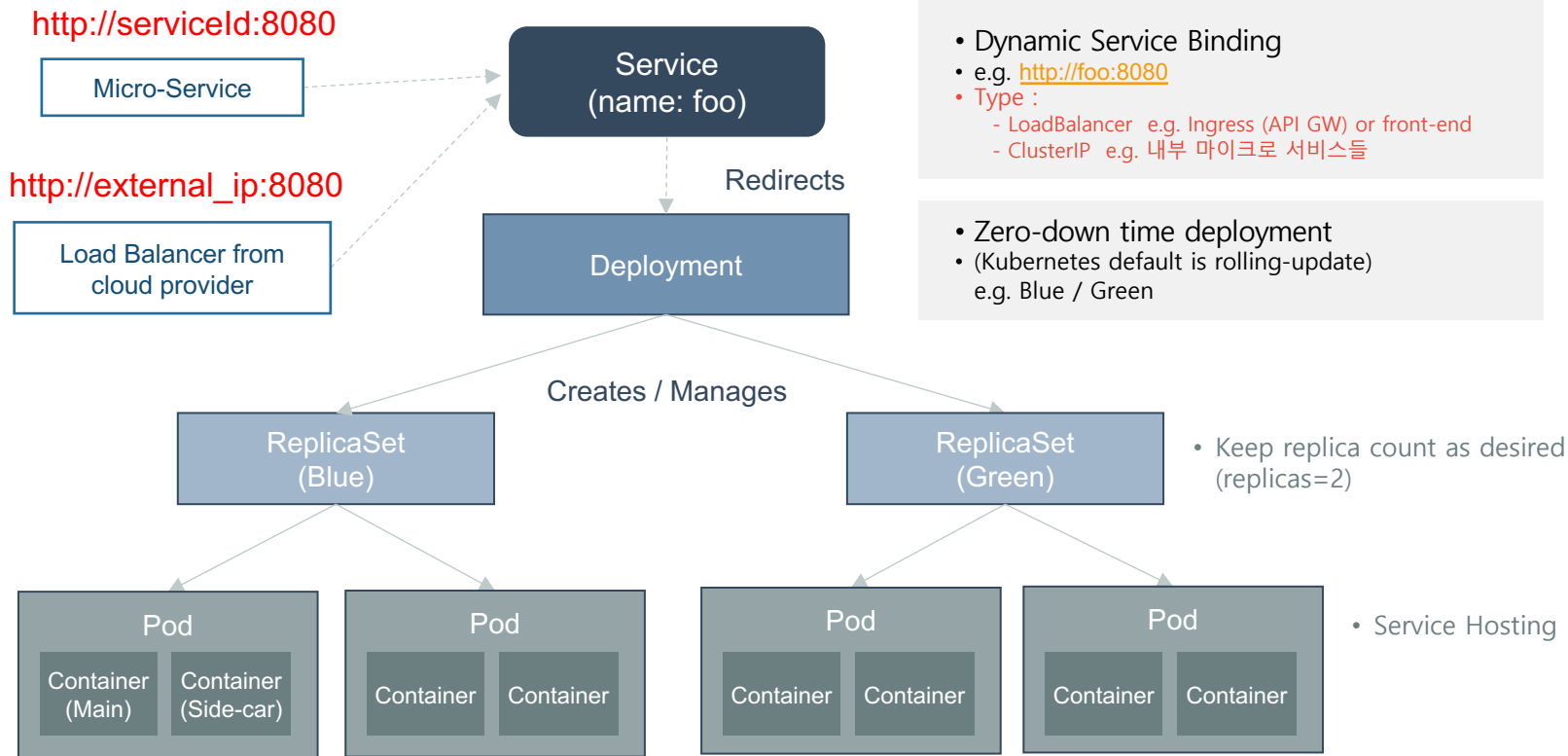
Container Orchestration – Scale out



Container Orchestrator – Zero Down Time Deploy



Kubernetes Object Model



Declaration based configuration

```
> my-app.yml
```

> Desired state



```
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
```

```
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
```

```
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
```


THANKS!

Any Question?

You can find me at:

jyjang@uengine.org
<https://github.com/jinyoung>
<https://github.com/TheOpenCloudEngine>

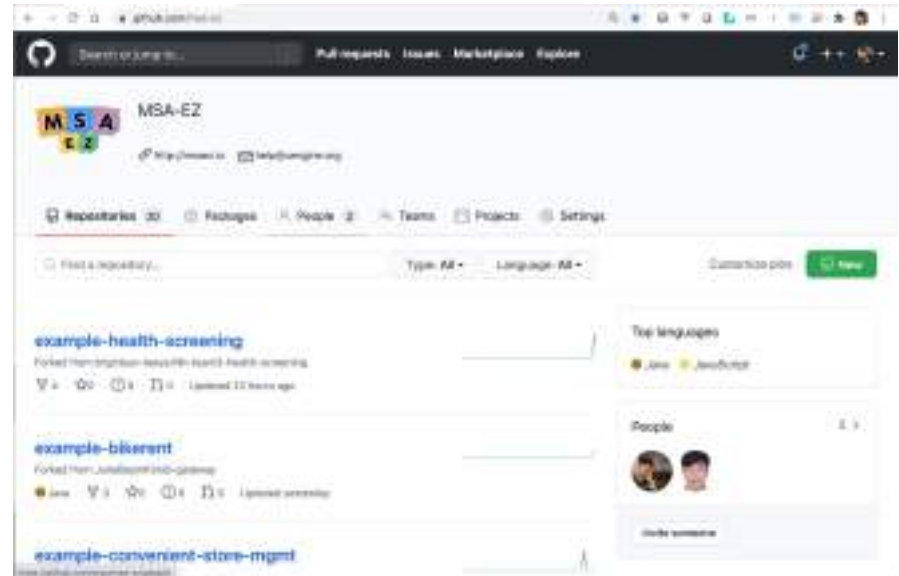
Recommended Books

- **Overall MSA Design patterns:**
<https://www.manning.com/books/microservices-patterns>
- **Microservice decomposition strategy:**
 - DDD distilled: <https://www.oreilly.com/library/view/domain-driven-design-distilled/9780134434964/>
 - Event Storming: https://leanpub.com/introducing_eventstorming
- **API design and REST:**
<http://pepa.holla.cz/wp-content/uploads/2016/01/REST-in-Practice.pdf>
- **Database Design in MSA:**
 - Lightly:
https://www.confluent.io/wp-content/uploads/2016/08/Making_Sense_of_Stream_Processing_Confluent_1.pdf
 - Deep dive:
https://dataintensive.net/?fbclid=IwAR3OSWkhqRjLI9gBoMpbsk-QGxeLpTYVXIjVCSaw_A5eYrBDc0piKSm4pMM

github.com/msa-ez

Reference MSA Projects

github.com/msa-ez



MSA School

msaschool.io

The screenshot displays the MSA School website interface. The header includes the MSA School logo and navigation links: Home, Course List, Search Course/Category, My Course, and My Profile. A sidebar on the left lists categories: MSA School 소개, MSA 소개란, MSA 플랫폼, MSA 제품/서비스 소개, MSA 소개, and MSA 소개 영상. The main content area features a large banner for 'Biz Dev Ops' with the text '협난한 MSA 구축 여정의 길라잡이' (A guide for the journey of building MSA). Below the banner, there are three circular icons representing different aspects of the course: 'BizDevOps 전 과정 지원' (Full support for the BizDevOps process), '실전 MSA 코드 활용' (Practical MSA code usage), and '논쟁적에 맞춘 특화된 교육' (Specialized education tailored to the debate). To the right of the banner, there is a section titled '가독성이 높고 이해하기 쉬운 BizDevOps 학습을 위한 1인 1책' (A one-person-one-book learning for BizDevOps that is easy to read and understand). Below this, there is a paragraph about MSA School's mission to provide high-quality, practical MSA education. At the bottom, there is a note about the MSA School's commitment to providing the best MSA education.

MSA School 소개

MSA 소개란

MSA 플랫폼

MSA 제품/서비스 소개

MSA 소개

MSA 소개 영상

협난한 MSA 구축 여정의 길라잡이

Biz Dev Ops

가독성이 높고 이해하기 쉬운 BizDevOps 학습을 위한 1인 1책

MSA School 소개

MSA School은 MSA 구축을 위한 최고의 교육 플랫폼을 제공하는 것을 목표로 하고 있습니다. MSA School은 MSA 구축을 위한 최고의 교육 플랫폼을 제공하는 것을 목표로 하고 있습니다. MSA School은 MSA 구축을 위한 최고의 교육 플랫폼을 제공하는 것을 목표로 하고 있습니다.

MSA School은 MSA 구축을 위한 최고의 교육 플랫폼을 제공하는 것을 목표로 하고 있습니다. MSA School은 MSA 구축을 위한 최고의 교육 플랫폼을 제공하는 것을 목표로 하고 있습니다. MSA School은 MSA 구축을 위한 최고의 교육 플랫폼을 제공하는 것을 목표로 하고 있습니다.