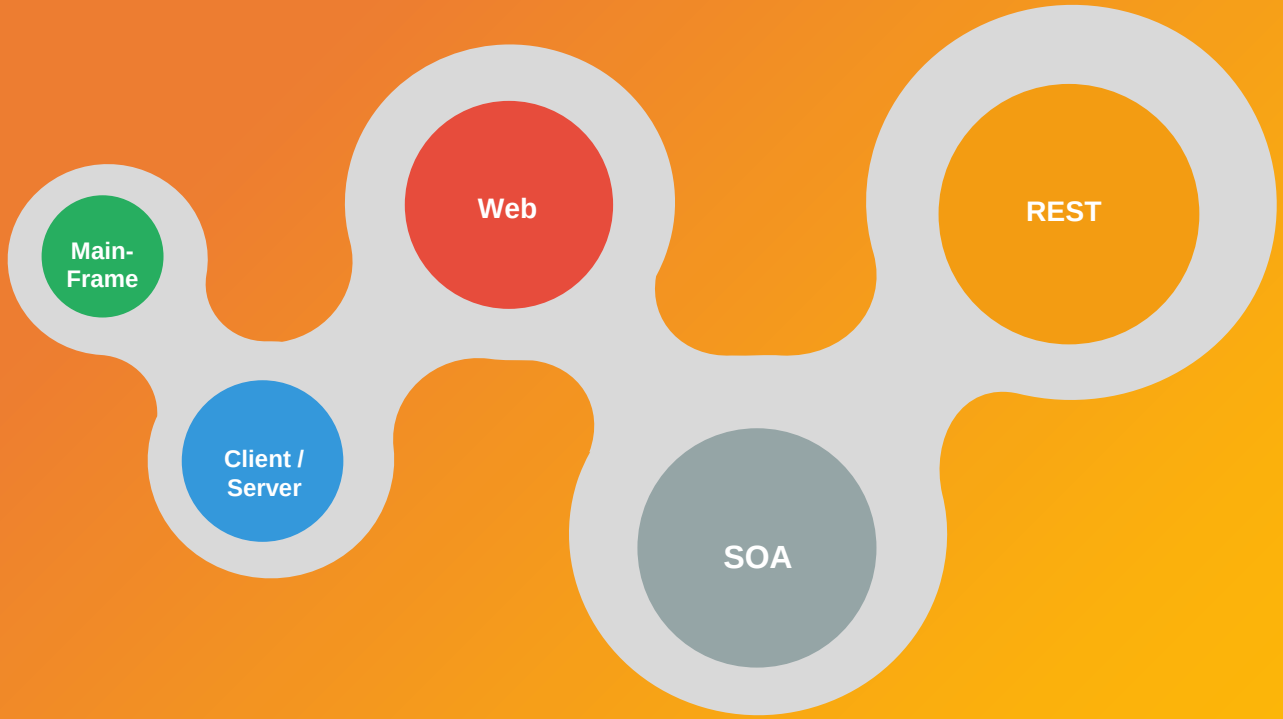
Developing Cloud Native Applications

3rd Jan 2020, ver2.0

uEngine

Software System Architecture Journey



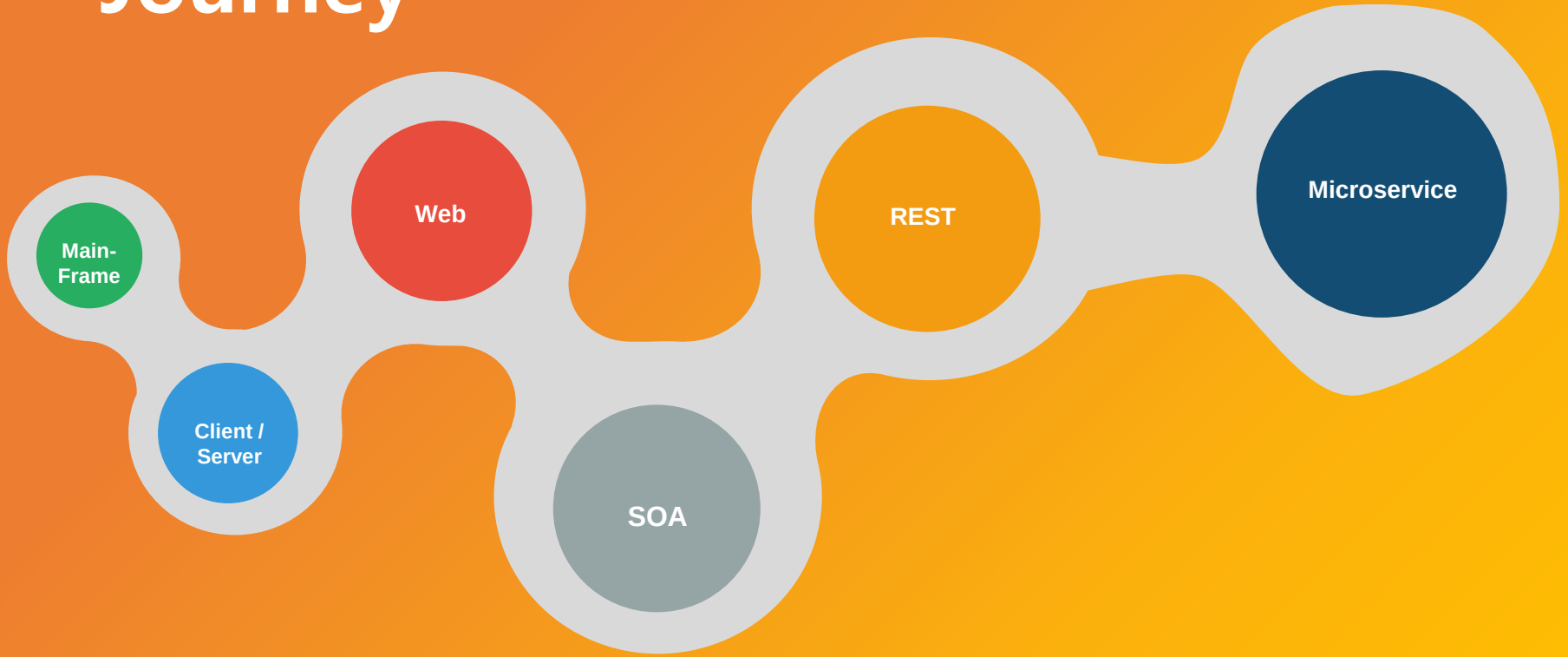
Agile delivery

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

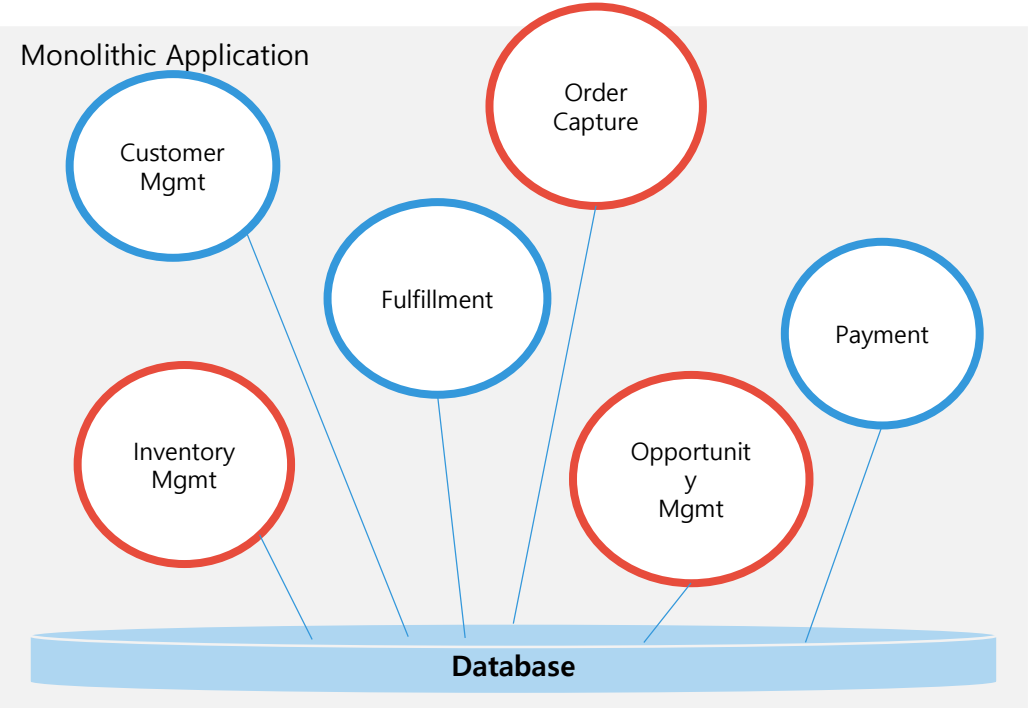
출처: 도서 The Phoenix Project

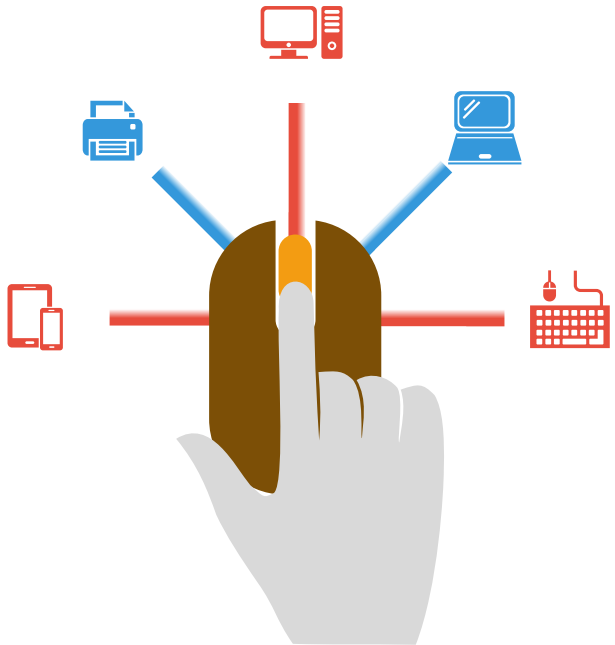
Software System Architecture Journey



First, What it is Not: A Monolithic Architecture

An Enterprise Application or Suite





Anatomy of a Monolithic Architecture

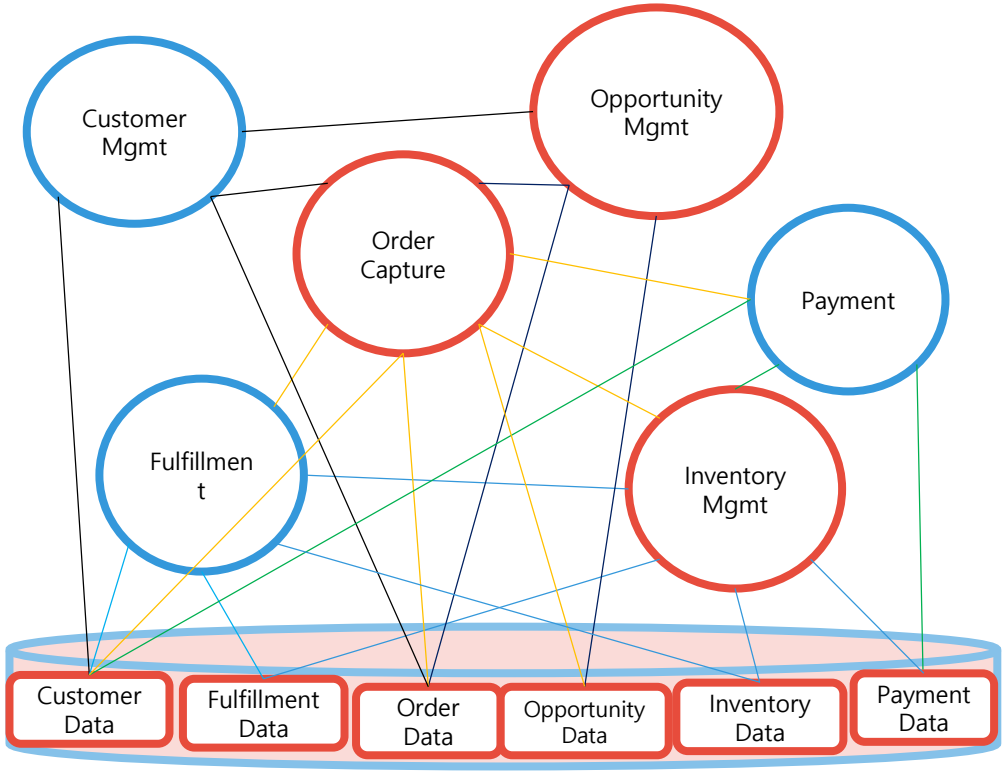
Good initially because interaction is easy

But, ease of interaction results in many inter-dependencies

Over time, coupling becomes tighter and tighter

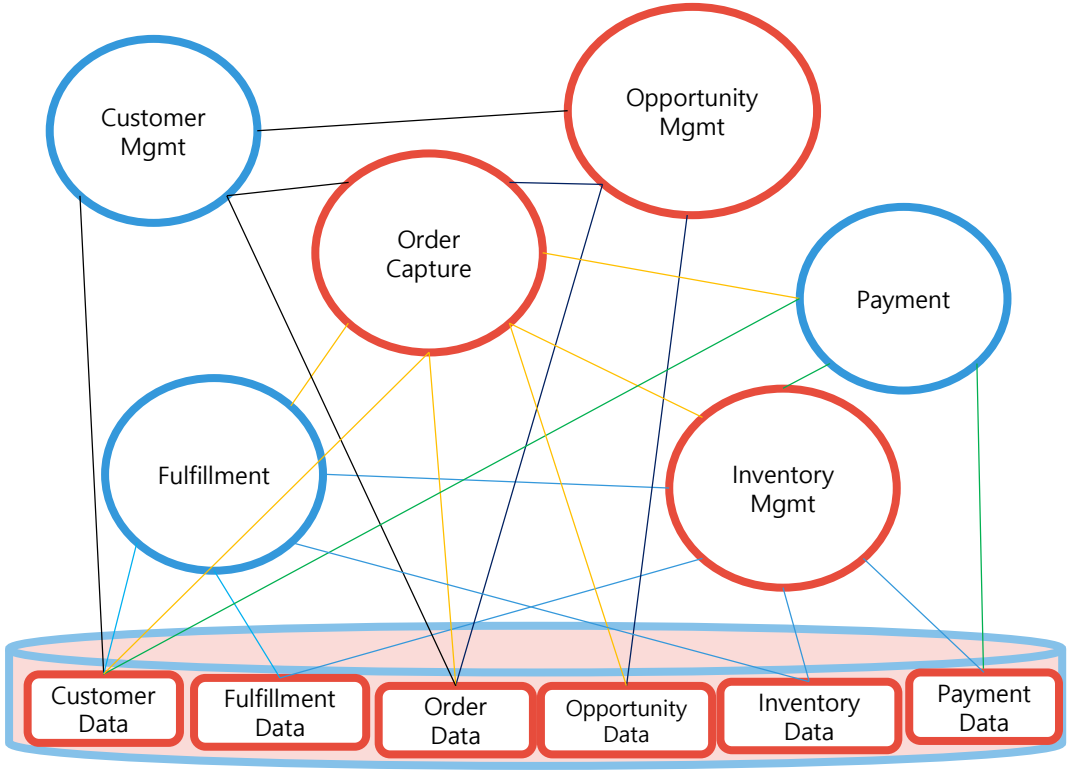
Change to one Component…

….require changes in many



Drawbacks of a Monolithic Architecture

Size matters, costs grow as they grow



Large code base

Overloaded IDE(Integrated development environment), web container, etc.

Deployment of any change requires redeploying everything

Only scales in one dimension

You are committed to the technology stack

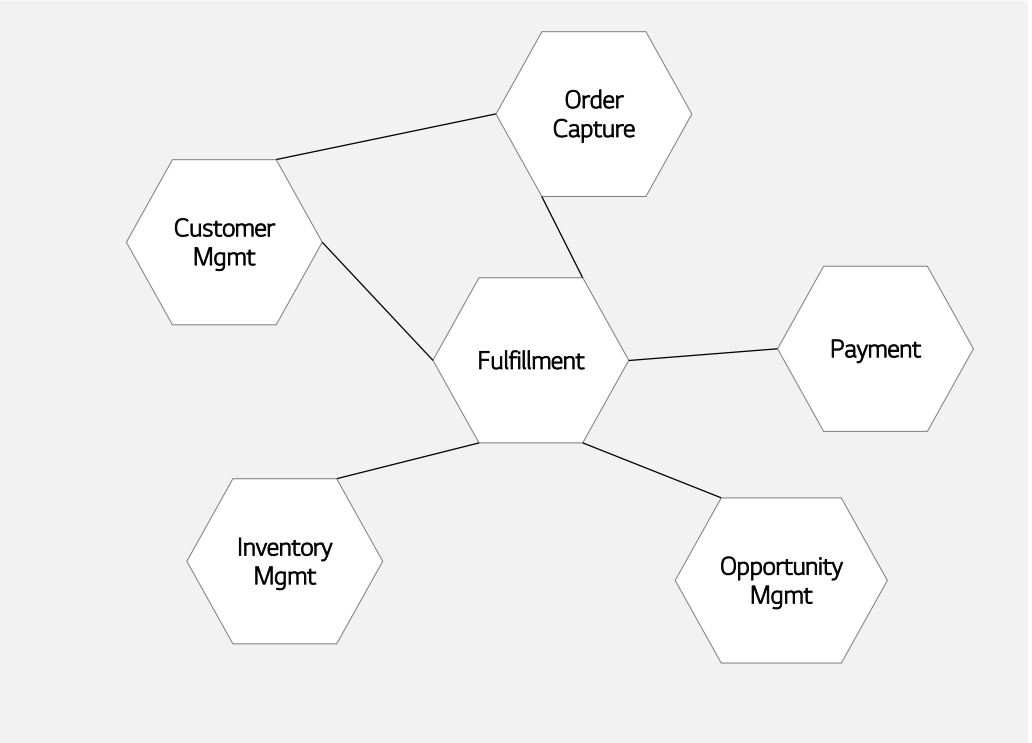
Becomes an obstacle to scaling development

In Contrast: A Microservice Architecture

Service-oriented architecture of loosely coupled elements with bounded contexts

Break each function into
separate deployment stacks

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



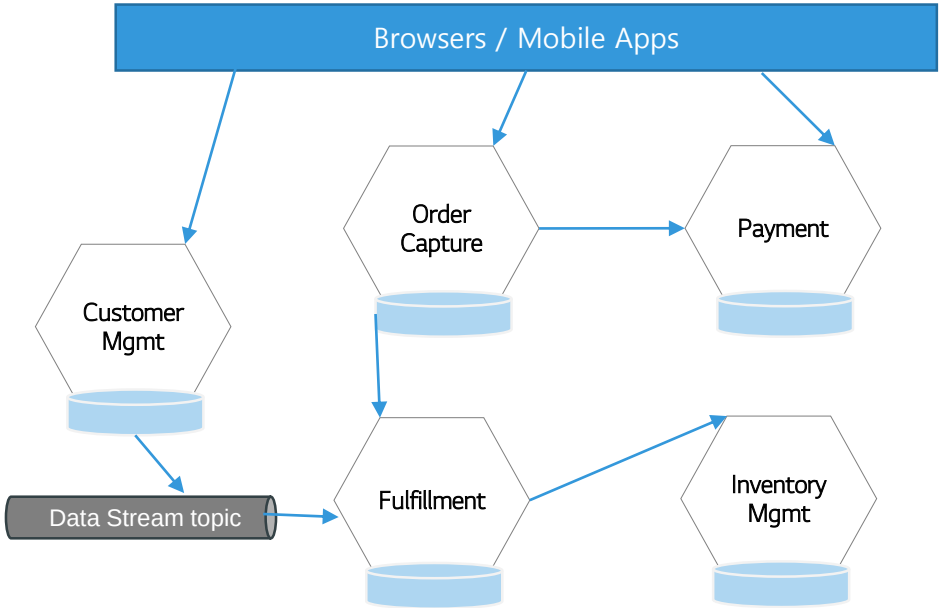
Anatomy of Microservice Architecture

Isolation is the name of the game

Service-oriented architecture

Loosely coupled elements

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



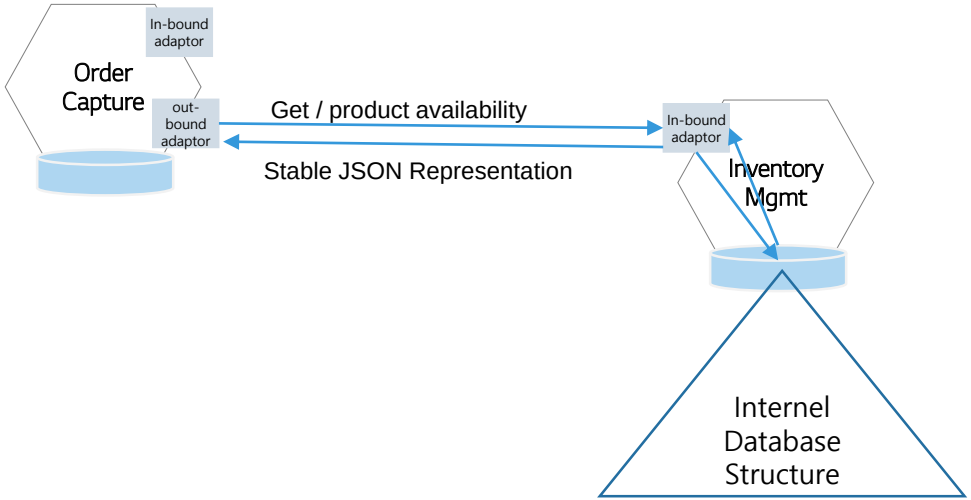
Loosely-Coupled Interaction via HTTP/REST

REST APIs must be stable and hide internals

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

Client ignore parts of representations they don't understand

Only additive changes allowed



Loosely-Coupled Interaction via Asynchronous Messaging

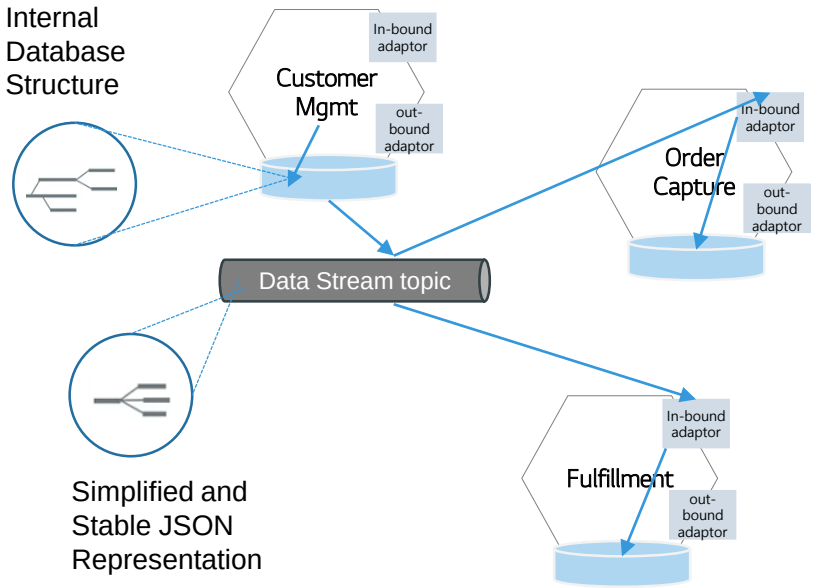
Data streaming can decouple database with shared data

Asynchronous messaging(streaming)

- Operation in near real-time
- Decouples **“timing”**

Data streams hide the internal implementation of the service

Client ignore parts of the stream they don’t understand



Microservice Architecture benefits

Smaller is better

Smaller, independent code base is easier to understand

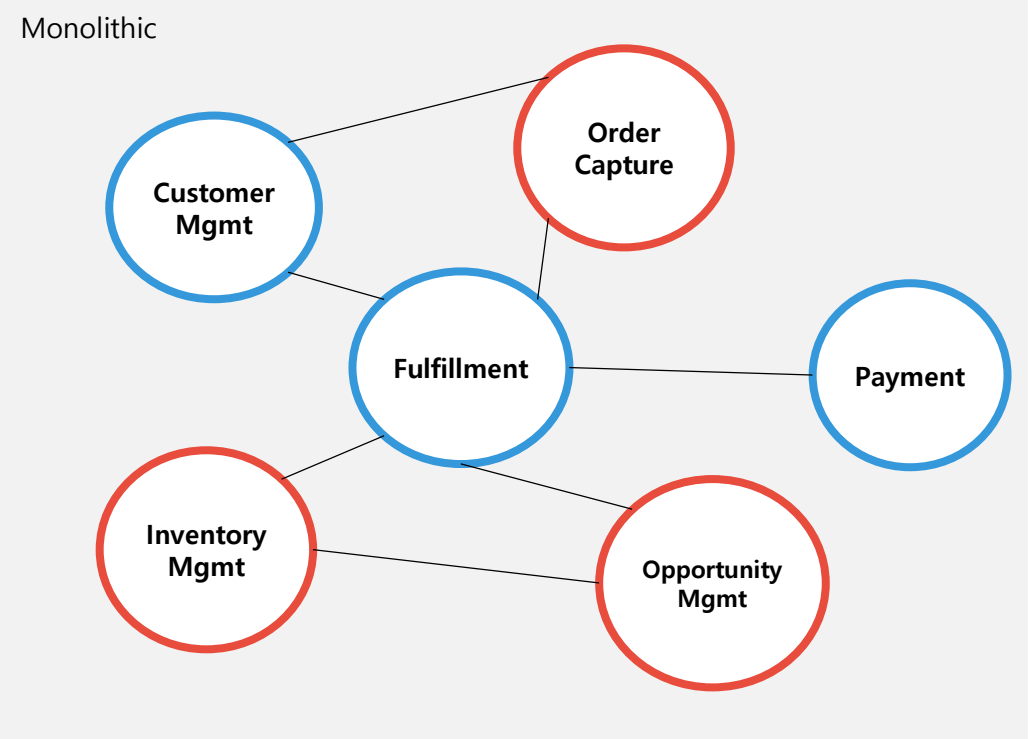
IDE and app startup are faster

Simpler deployment and testing of just the changed service

Improved fault isolation

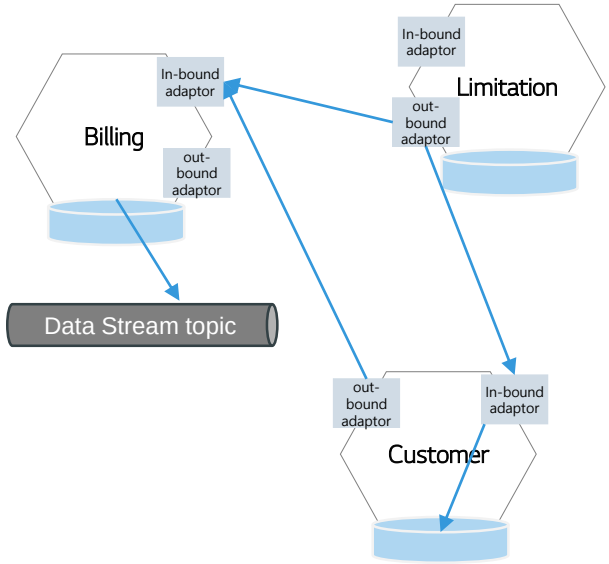
Not committed to one technology stack

Testing can be automated since all features are exposed as services



Microservice Architecture

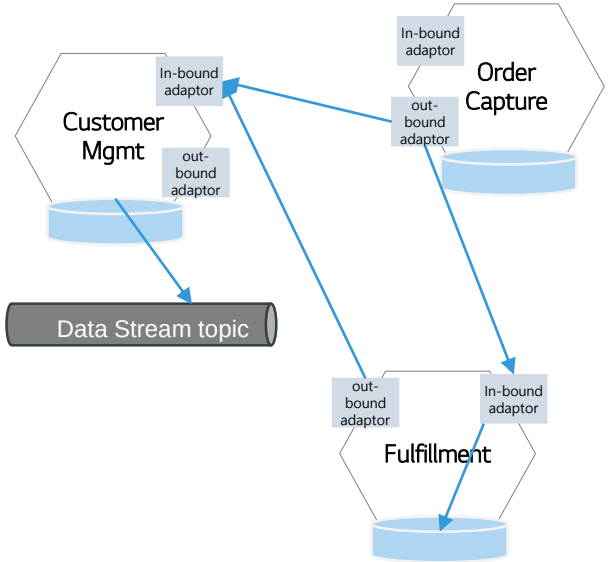
- Updating one service doesn't require changing others
- Ability to upgrade the tech stack (HW/SW/DBMS/NW) Of each service independently



- Good fault isolation
- Smaller, simpler code base
- Options for scaling

Drawbacks to a Microservice Architecture

Distributed computing adds complexity and slow down initial development



Dev tools not optimized for distributed services

Testing can be more complicated

Deployment and operations are more complex

Where/How to decompose the services?

Inter-service communication complicates development

- Potentially unreliable connections require fault tolerance

Maintaining consistency with distributed transactions is hard

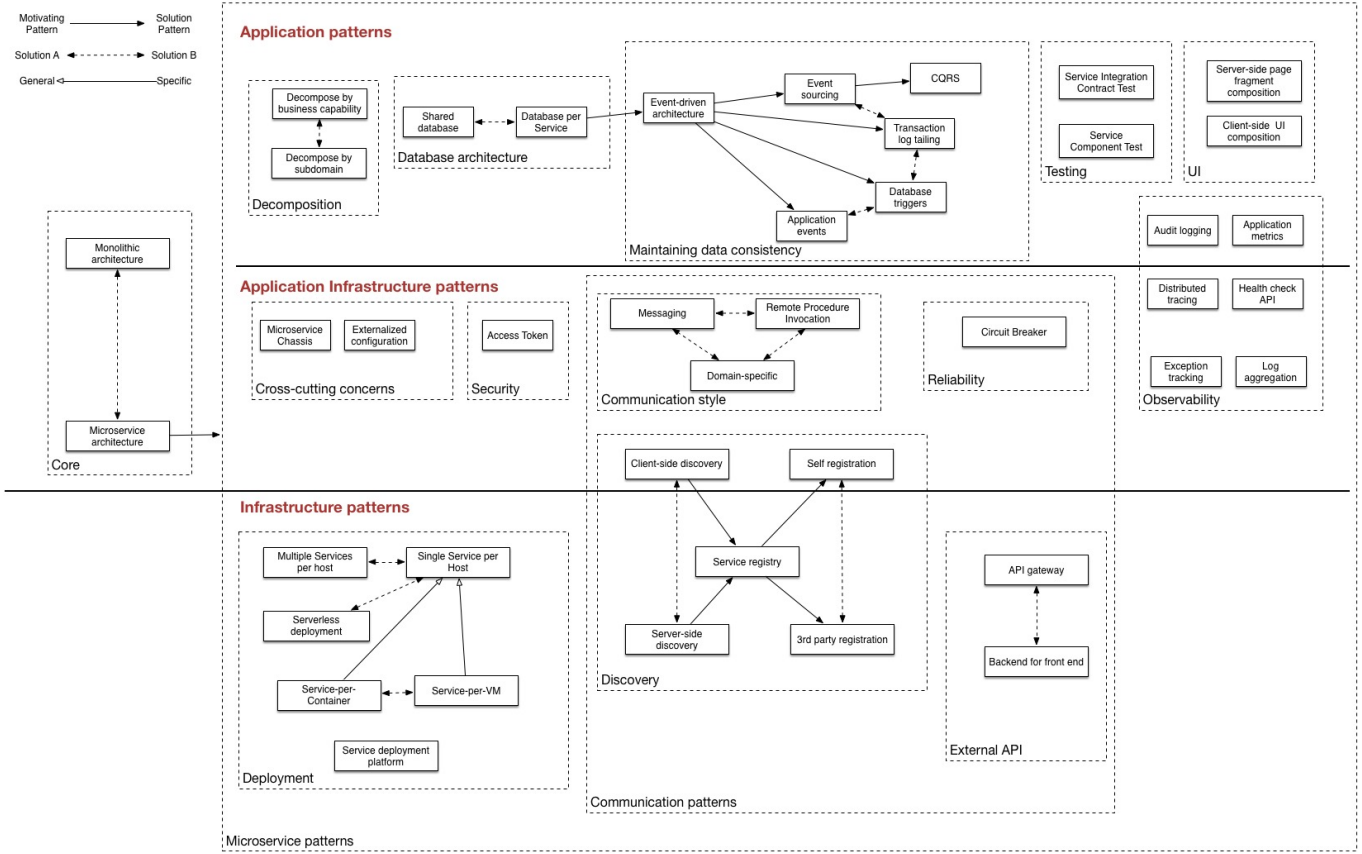
What about inter-service security? Identity management?

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

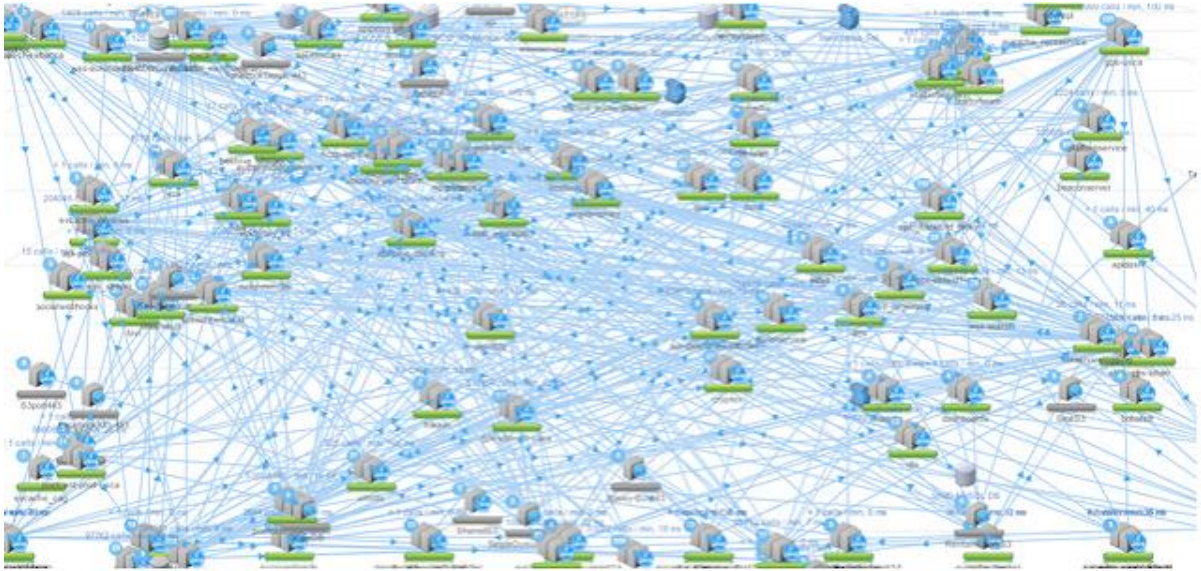
Micro-Service Architecture Patterns

- microservices.io



Netflix

넷플릭스는 수백개의 마이크로서비스를 운영하고 있는 것으로 유명하다. 각 마이크로서비스간에는 REST 방식의 호출을 통하여 연동되며 이들간의 자동화된 식별과 동적 연동을 위하여 자체적인 플랫폼을 구축했고 이를 Netflix OSS 라는 이름으로 오픈소스화 하였다.



마이크로 서비스 전환 사례

Video & Broadcasting

Mobile Apps and
Serverless Microservices

Pure Play Video OTT- A

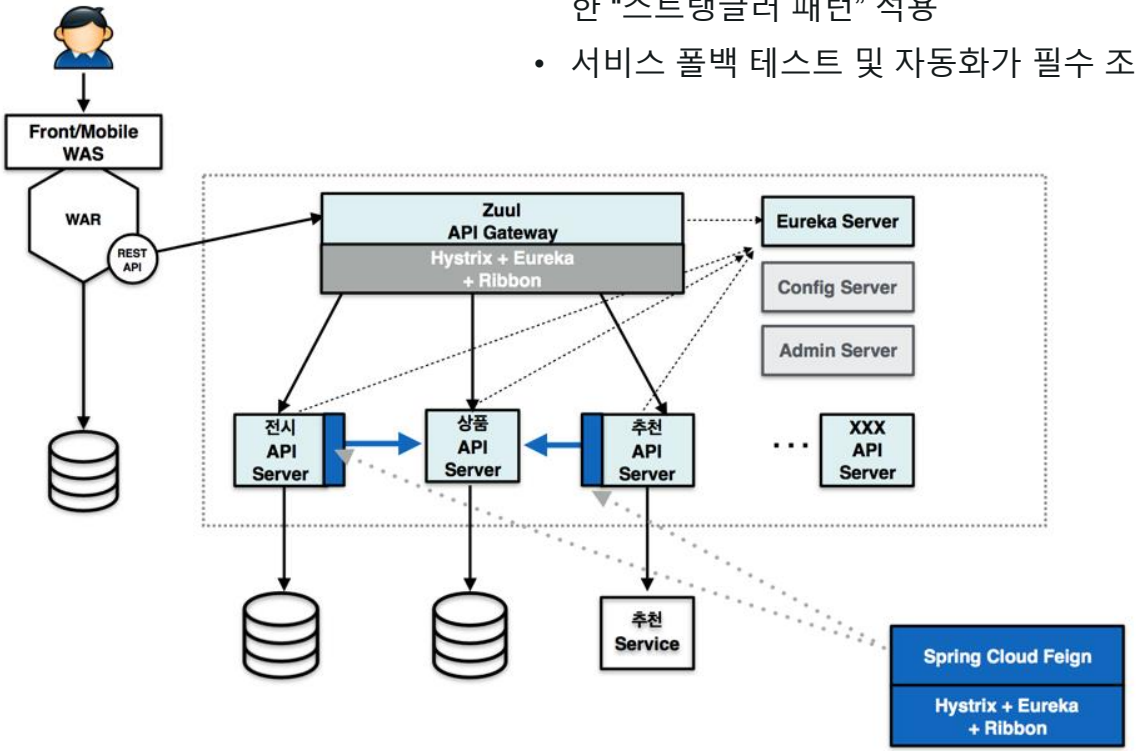
From Monolithic to
Microservices

Gaming Platform

IoT Service

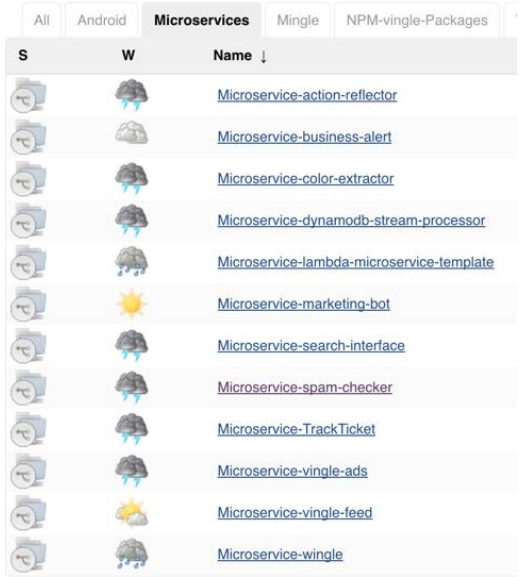
Serverless Microservices

- 가장 작고 독립적인 서비스로 부터 마이그래이션
- 순차적 레거시의 마이크로서비스 전환을 위한 API GW 적용한 “스트랭글러 패턴” 적용
- 서비스 폴백 테스트 및 자동화가 필수 조건



vingle

- 새로 추가/변경 필요한 기능부터 분리
  - 안정된 레거시는 가능한 손대지 않음
  - 적은 리스크 업무 영역부터
  - 기술보다 노하우와 팀의 문화 정립 우선시
- Business Domain 단위 서비스구성
  - 팀단위 분리 X, 기술적 단위 분리 X
  - 비즈니스 서브 도메인 단위로의 구성
    → Separation of Concerns
  - 응집도
- 서비스 존재 목적은 재사용되어지는 것
  - 표준화된 API I/F
  - 자동화된 문서화



Tip: monolithic and MSA

	monolithic	MSA
Aggregation (데이터 통합)	Backend 가 주도	Front 가 주도
Database	통합 데이터베이스	서비스 별 데이터베이스
필수 환경	WAS	DevOps, PaaS (Grid Engine)
서비스 굵기	업무 비즈니스 기능별	구현 팀별, 10000 라인 이하로?, 관심사 별
Front 기술	JSP, Struts 등 Server-side rendering	MVVM, AJAX 등 Client-side rendering
Container / Packaging	WAS / WAR	Spring-Boot, Docker

Comparison with SOA

	Traditional SOA	Microservices
Messaging type	Smart, but dependency-laden ESB	Dumb, fast messaging (as with Apache Kafka)
Programming style	Imperative model	Reactive actor programming model that echoes agent-based systems
Lines of code per service	Hundreds or thousands of lines of code	100 or fewer lines of code
State	Stateful	Stateless
Messaging type	Synchronous: wait to connect	Asynchronous: publish and subscribe
Databases	Large relational databases	NoSQL or micro-SQL databases blended with conventional databases
Code type	Procedural	Functional
Means of evolution	Each big service evolves	Each small service is immutable and can be abandoned or ignored
Means of systemic change	Modify the monolith	Create a new service
Means of scaling	Optimize the monolith	Add more powerful services and cluster by activity
System-level awareness	Less aware and event driven	More aware and event driven

Source: pwc

MSA Trade-offs (from Martin Fowler)

Microservices provide benefits…

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.

- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.

- Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

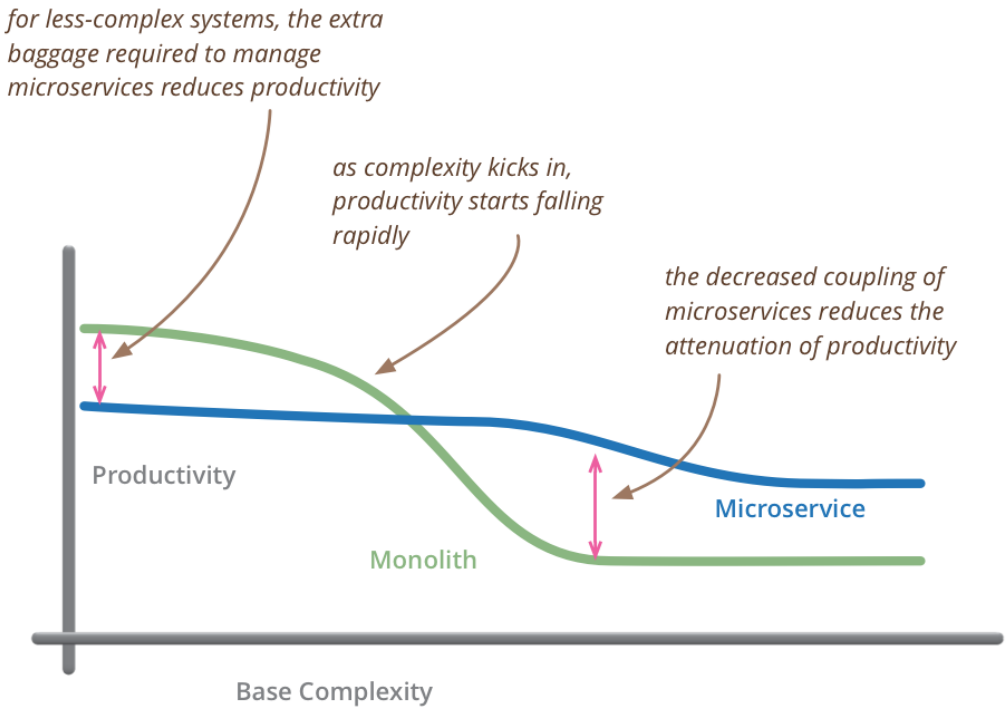
…but come with costs

- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.

- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.

- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

MSA Trade-off



but remember the skill of the team will outweigh any monolith/microservice choice

Tip: 10 Attributes of Cloud Native Applications

1. Packaged as lightweight containers
2. Developed with best-of-breed languages and frameworks
3. Designed as loosely coupled microservices
4. Centered around APIs for interaction and collaboration
5. Architected with a clean separation of stateless and stateful services
6. Isolated from server and operating system dependencies
7. Deployed on self-service, elastic, cloud infrastructure
8. Managed through agile DevOps processes
9. Automated capabilities
10. Defined, policy-driven resource allocation

https://thenewstack.io/10-key-attributes-of-cloud-native-applications/

Table of content

Microservice and
Event-storming-Based
DevOps Project

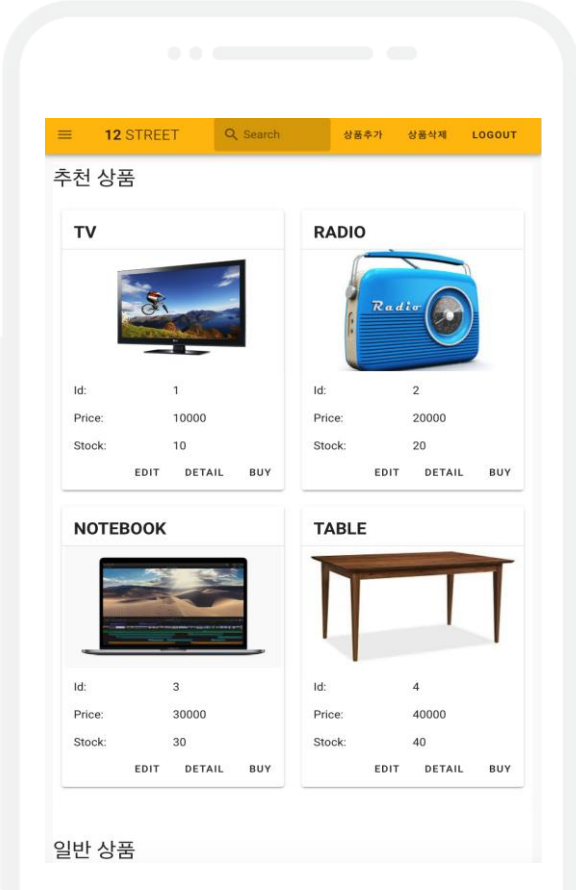
UEngine

Target Domain

: Online Shopping Mall (12 STREET)

- Vision & Mission

Service resiliency	24시간 365일 접속과 주문이 가능 : 자동화된 회복, 장애전파최소, 무정지 재배포
Customer responsiveness	다양한 고객 기능 요구사항의 탐색과 반영
Scalability	조직, 기능 및 데이터의 확장에 열려 있는 아키텍처 : Feature-driven-development



Organization & KPI Definition - 창업시기

- Multiple Concerns Single Organization
- Horizontally aligned

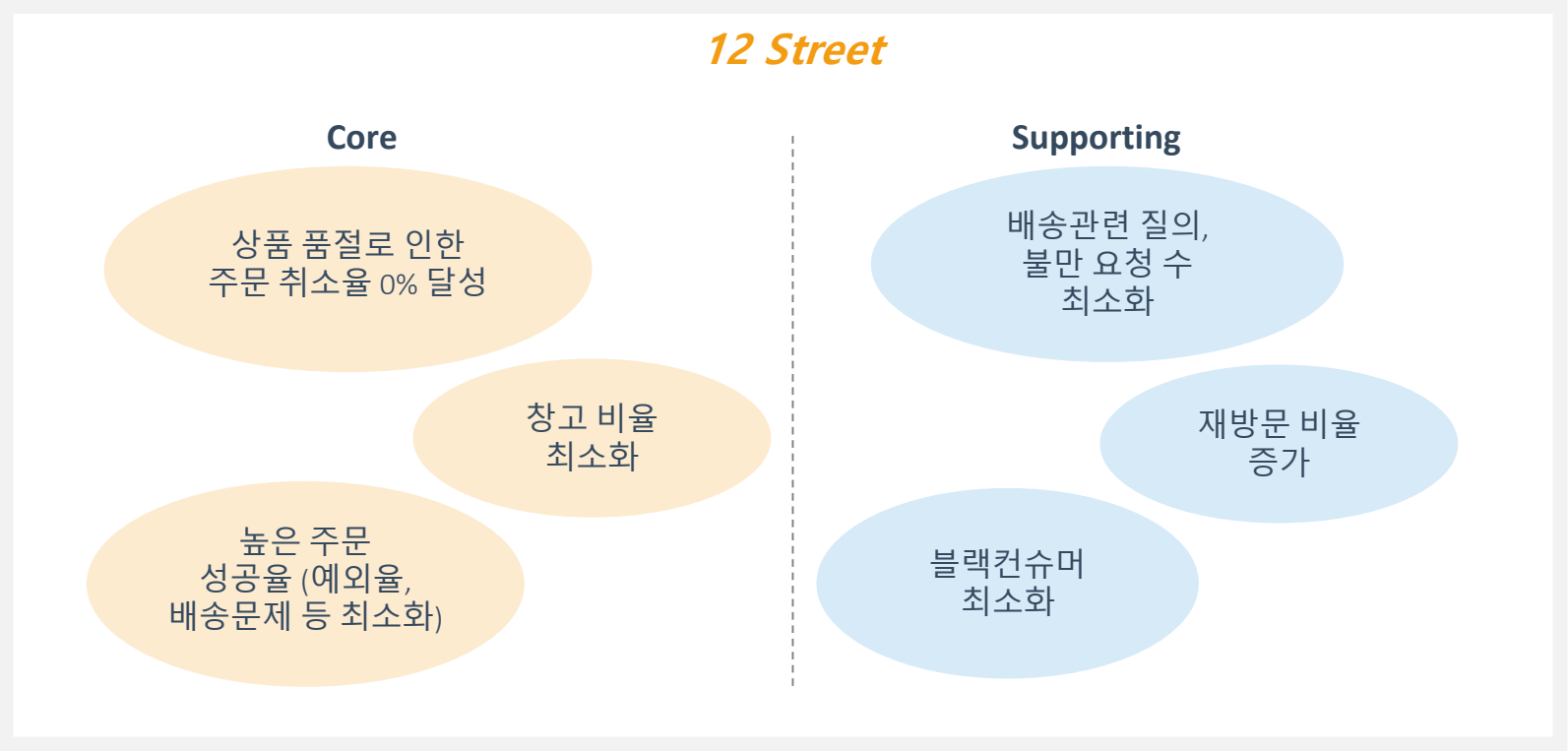
User Stories

1. 고객이 주문할 상품과 개수를 선택하여 주문버튼을 클릭한다.
2. 주문이 벌어지면 배송팀은 해당 상품에 대한 배송을 준비한다.
3. 주문과 취소가 완료됨에 따라서 상품 관리팀이 관리하는 재고량이 변경(+/-)된다.
4. 품절 상품에 대해서도 고객이 구매 가능하며, 상품이 입고된 후, 이를 구매 대기 고객에게 인폼하고 처리(재 입고시 지연 배송, 주문 취소) 한다.

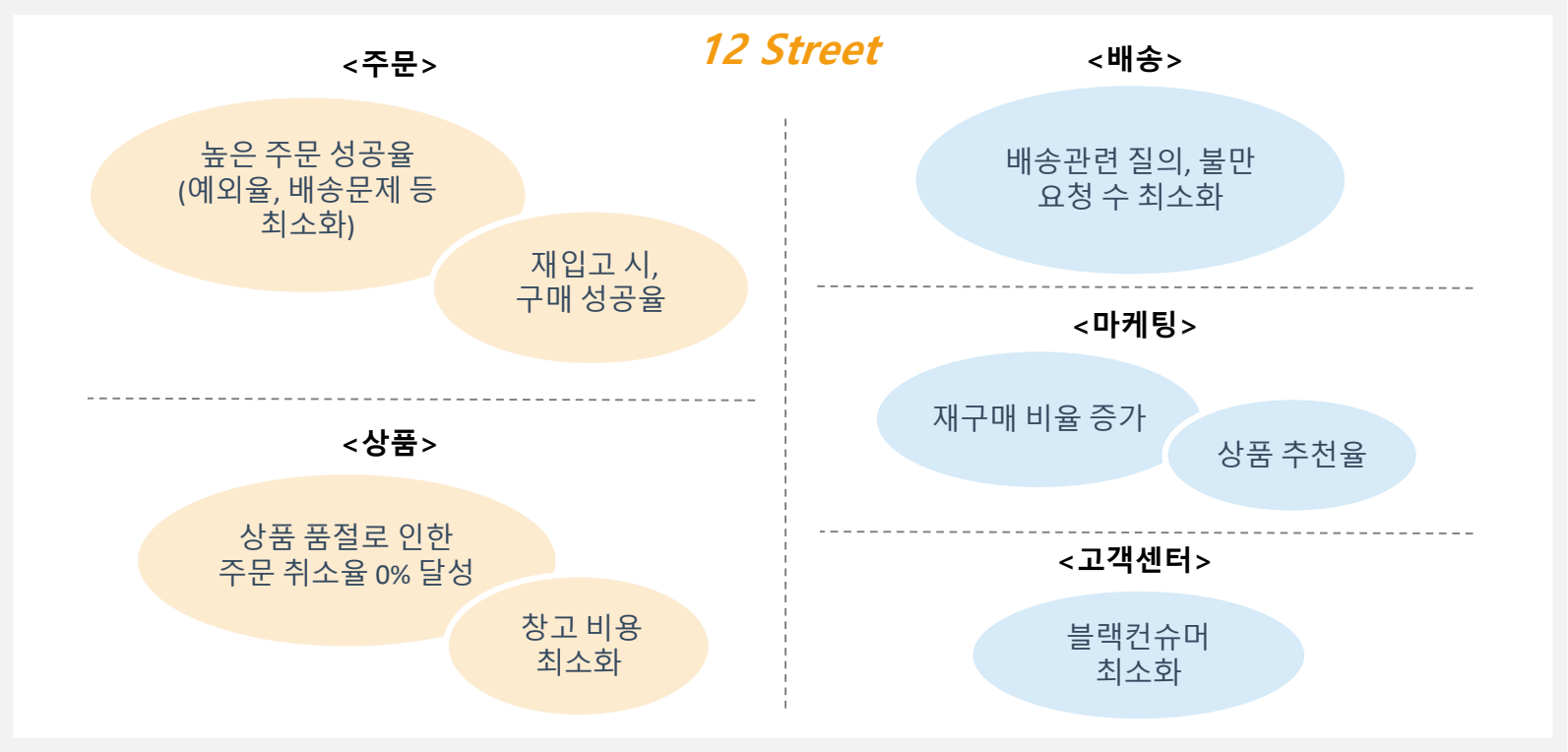
1. The customer enters the number of products to order and clicks the order button.
2. When an order is placed, the delivery team arranges for delivery of the product.
3. As the order or cancellation is completed, the product stock is changed(+/-).
4. Customers can also purchase products that are out of stock and This can be resolved after registering products later.

"책임소재가 불분명한 요구사항"

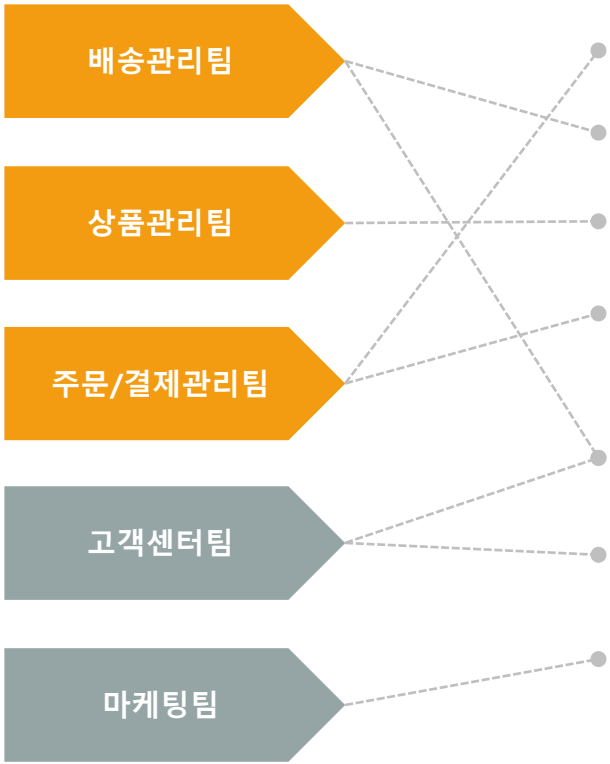
Separate Core Domain from Supporting Domain



Separation of Concerns - 회사의 성장



신설된 User Stories 및 Concerns Mapping



1. The customer enters the number of products to order and clicks the order button.
2. When an order is placed, the delivery team arranges for delivery of the product.
3. As the order or cancellation is completed, the product stock is changed(+/-).
4. Customers can also purchase products that are out of stock and This can be resolved after registering products later

5. A notification is sent to the customer whenever the delivery status changes.
6. Customers who repeat orders and cancellations frequently are blacklisted, and customers with high order purchase rates are whitelisted.
7. The marketing team mails recommendation products tailored to the customers at 9 am daily.

Table of content

Microservice and
Event-storming-Based
DevOps Project

UEngine

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.

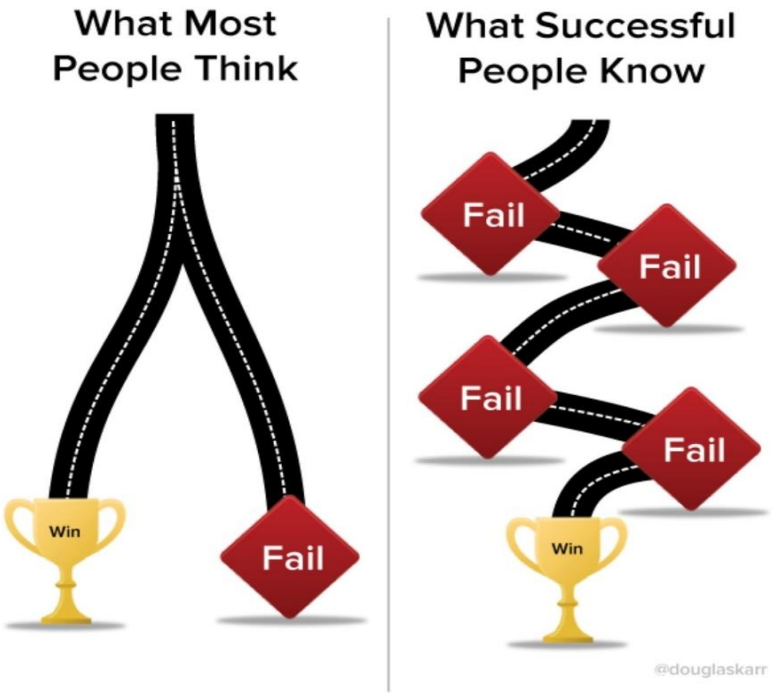
B 사 제조분야 SaaS 운영

- 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

What is Agile?

Planning is important!

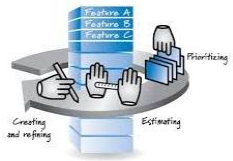
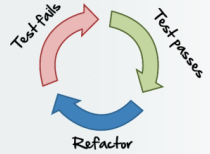
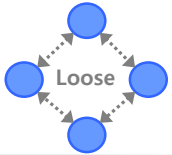
- Fail is bad
- Cost-driven



Agility is important

- Fail Cheap, Fail Fast, Fail Often
- Customer-driven

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**	

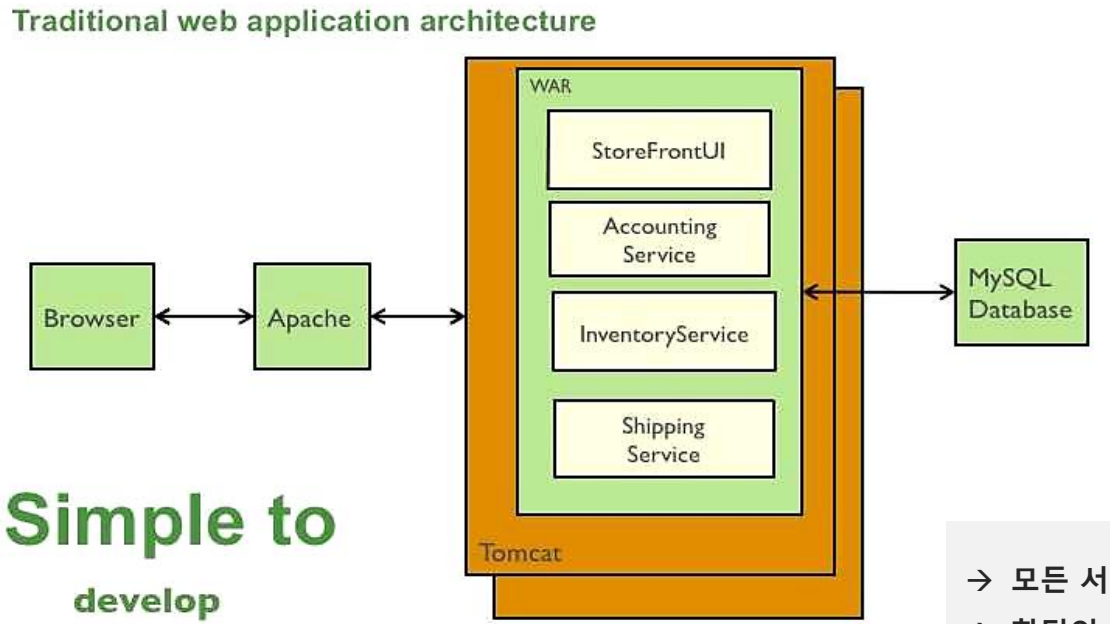
Continuous Delivery

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

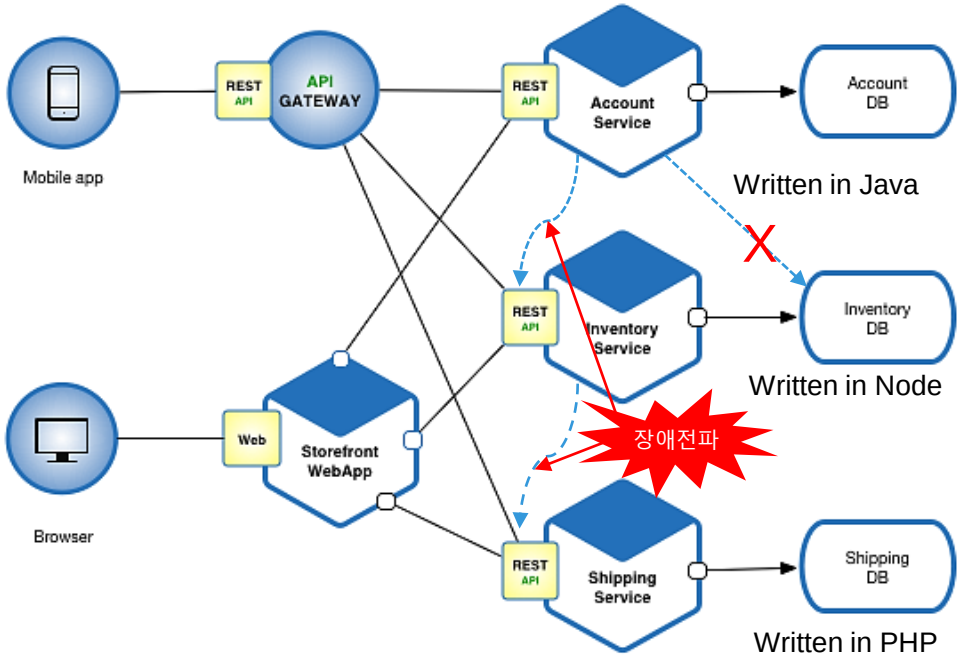
Jeff Bezos Mandate”

1. All teams will henceforth expose their data andfunctionality 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

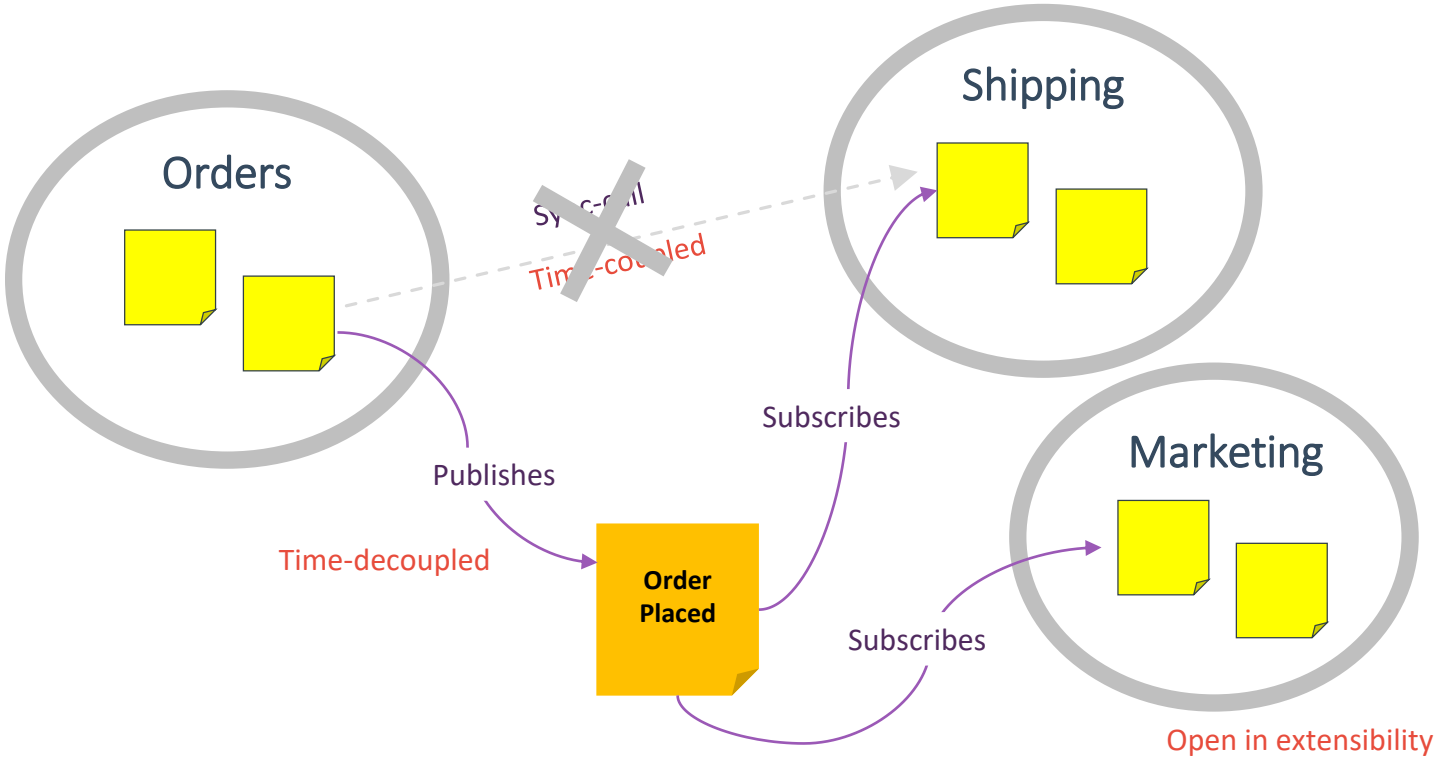
Contract based, Polyglot Programming
→ Separation of Concerns, Parallel Development, Easy Outsourcing



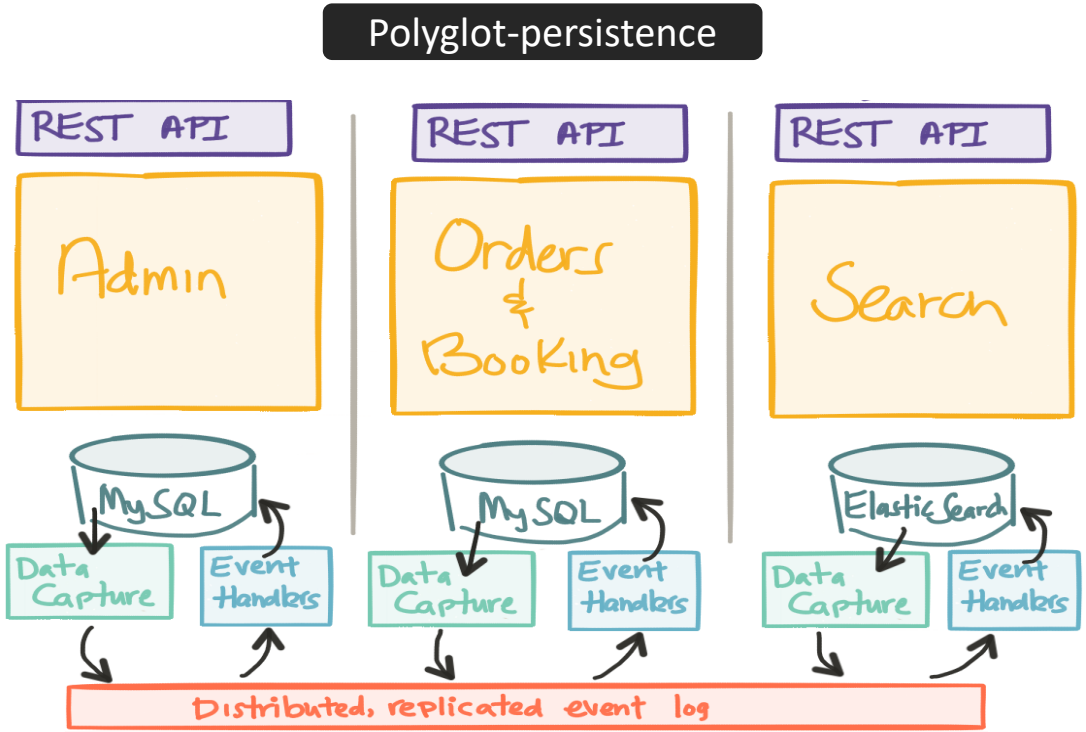
[Limitation]

1. Code Coupling 해소
   But, Time Coupling 잔존
2. 서비스 Blocking 가능성 내재
3. 장애 전파 우려
4. Point to Point 연결에 따른 복잡한 스파게티 네트워크

Approach #2 : Event Driven Architecture



Approach #2 : Event Driven Architecture



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

주문팀

주문 상태 변경

주문 상태 변경

주문 상태 변경

주문들어옴 (PO. No: 1, TV, 5개, 홍길동)

주문들어옴 (PO. No: 2, TV, 5개, 아무개)

주문취소됨 (PO. No:1)

주문들어옴 (PO. No.: 3, Radio, 2개, 아무개)

MySQL

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

상품팀

재고변경 (No:1, TV, 5)

재고변경 (No:1, TV, 0)

재입고됨 (ID:1, TV, 10개)

재고변경 (No:1, TV, 15개)

재고수정 (No:2, Radio, 8)

MongoDB

ID	Item	Stock
1	TV	15
2	Radio	8

배송팀

배송준비 (No: 1)

상품발송함 (주번:1)

배송준비 (No: 2)

상품발송함 (주번:2)

배송수거 (No: 3)

상품수거됨 (주번:1)

배송준비 (No: 3)

Redis

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

마케팅팀

선호도수집

새벽

추천상품 저장

File-System

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

오전

추천상품 이메일 발송

마이페이지

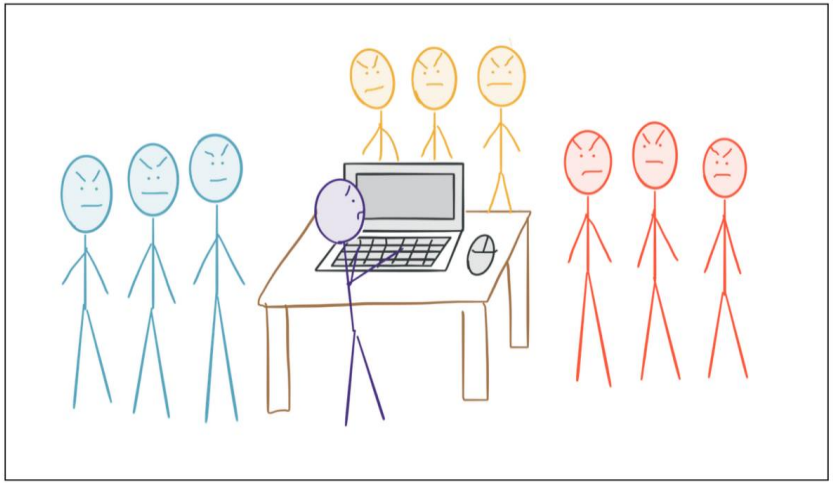
주문정보 수집 (PO. No:3)

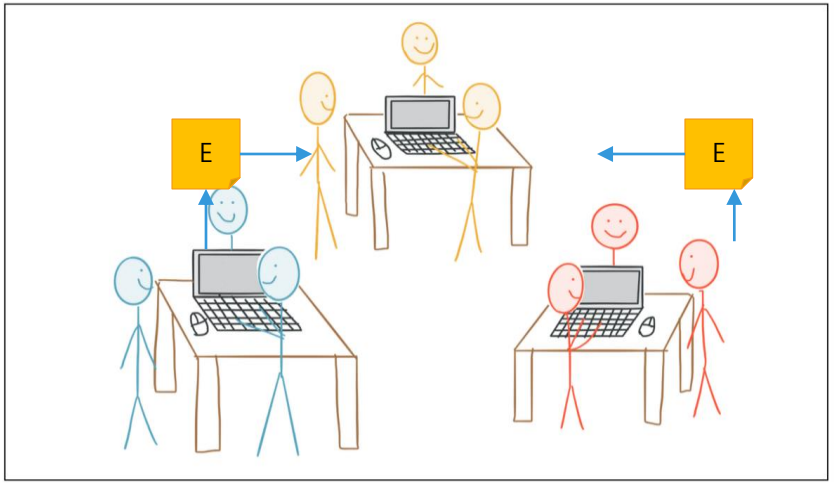
배송수거 정보수집 (PO. No:1)

HTML

id	User	Item	Qty	prc	o-stat	d-stat
001	홍길동	TV	5	50,000	cancel	Returned
002	아무개	TV	5	50,000	order	Delivery Started
003	아무개	RADIO	2	20,000	order	

Event Driven MSA Architecture





Monolith:

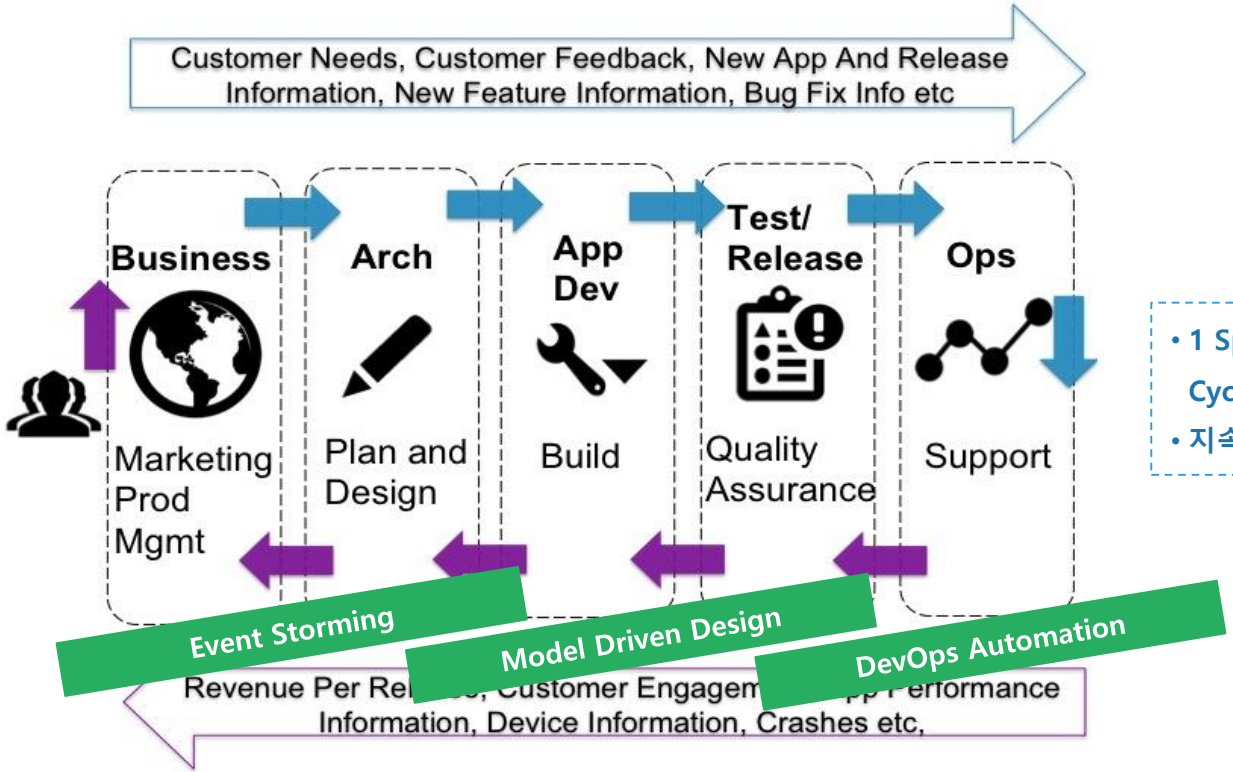
- With Canonical Model
- (“Rule Them All” model)

Reactive Microservices:

- Autonomously designed Model(Document)
- Polyglot Persistence

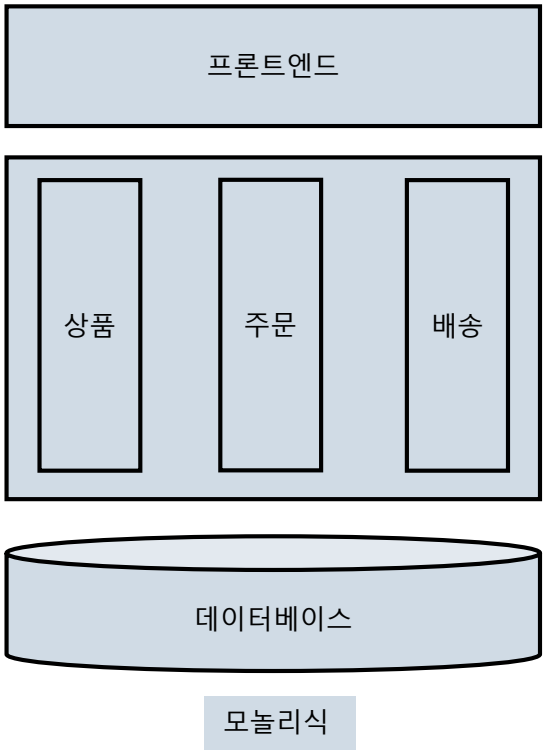
Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

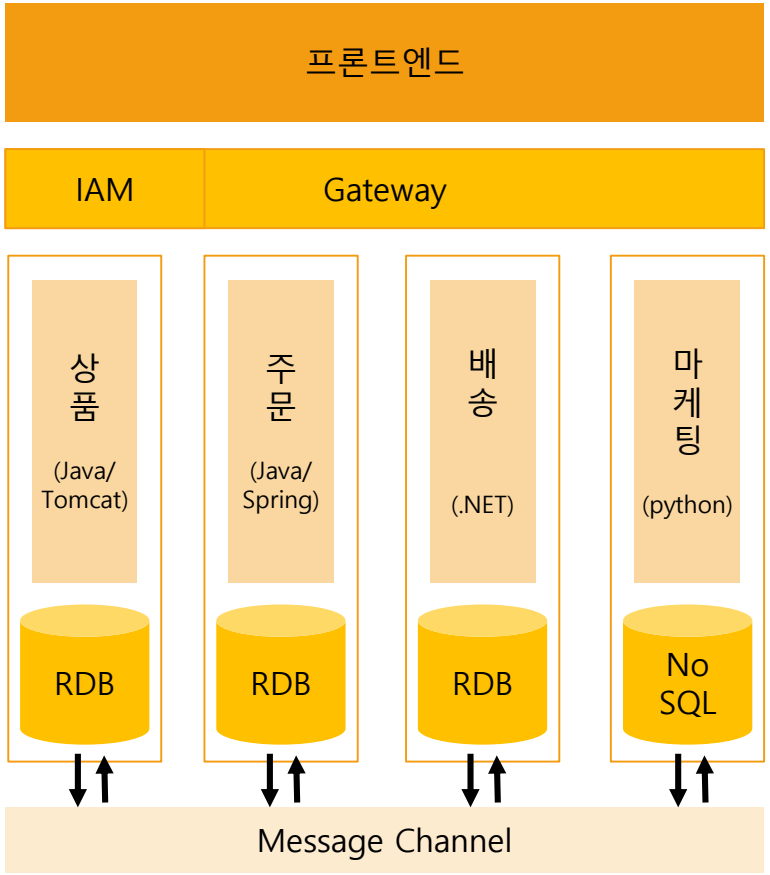
Approach #3: BizDevOps Process



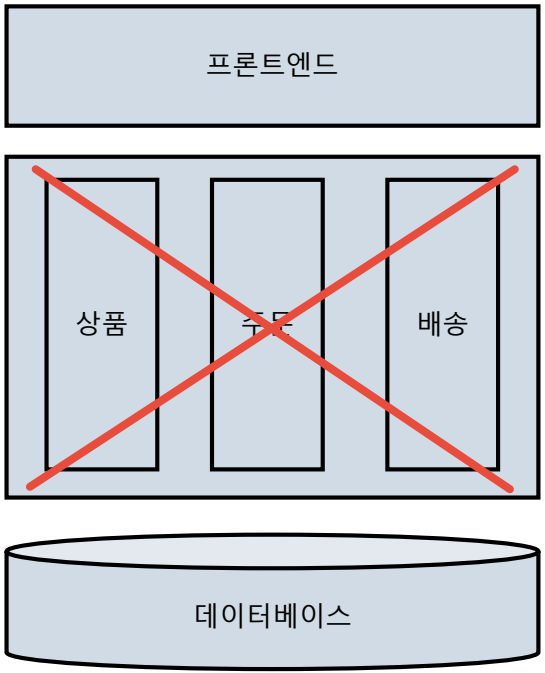
- 1 Sprint = 1~2 Weeks Cycle 기반
- 지속적인 Needs 탐색

적용 아키텍처

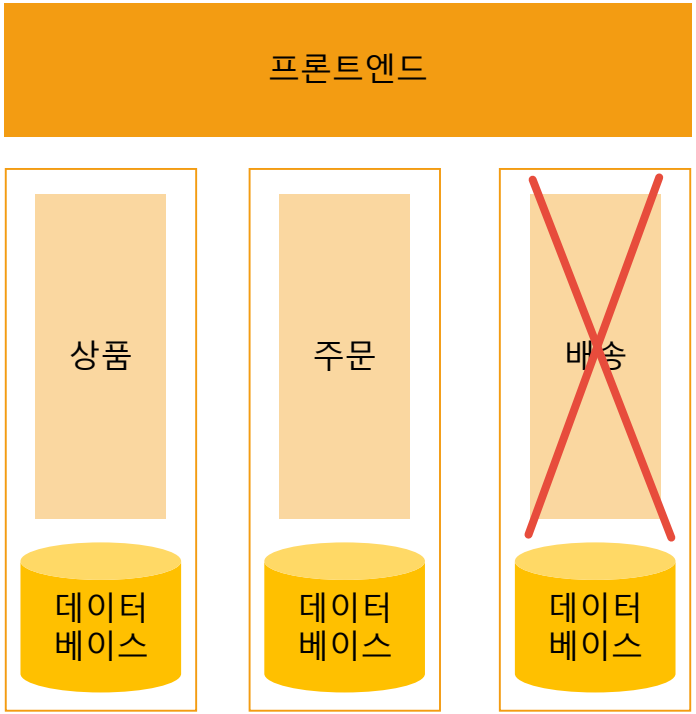

프론트엔드
상품
주문
배송
데이터베이스
모놀리식


프론트엔드
IAM
Gateway
상품
(Java/
Tomcat)
주문
(Java/
Spring)
배송
(.NET)
마케팅
(python)
RDB
RDB
RDB
No SQL
Message Channel

효과 – 장애 최소화

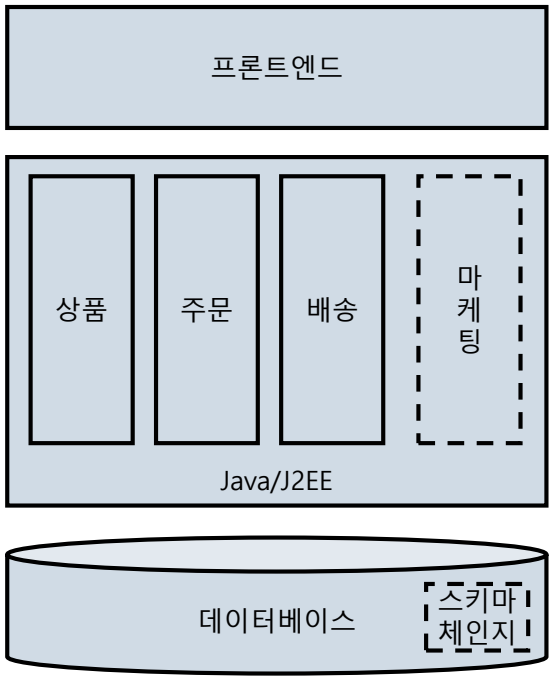


장애가 전파되며, 수동 복구로 장애 지연

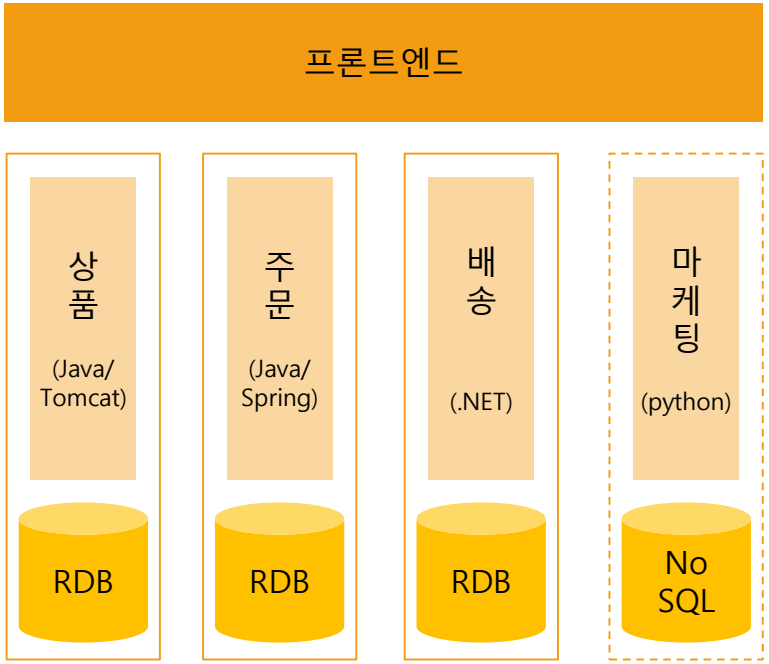


장애가 격리되며, 자동으로 복구됨(이전버전)

효과 – 기능 확장

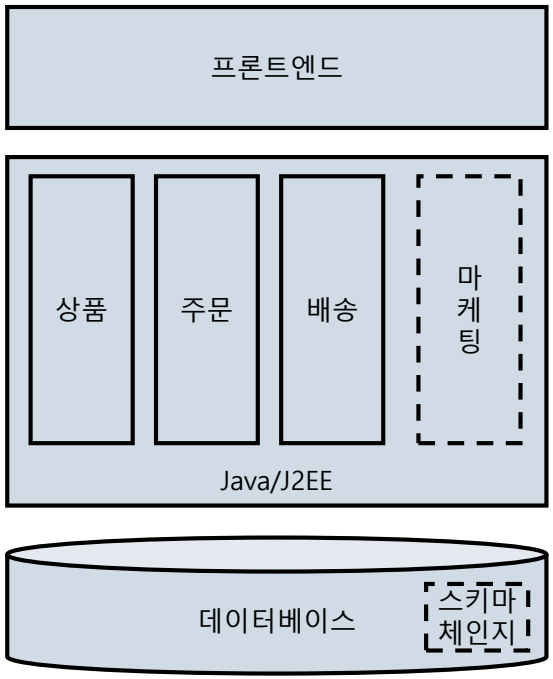


기존 코드의 수정 → Big Impact

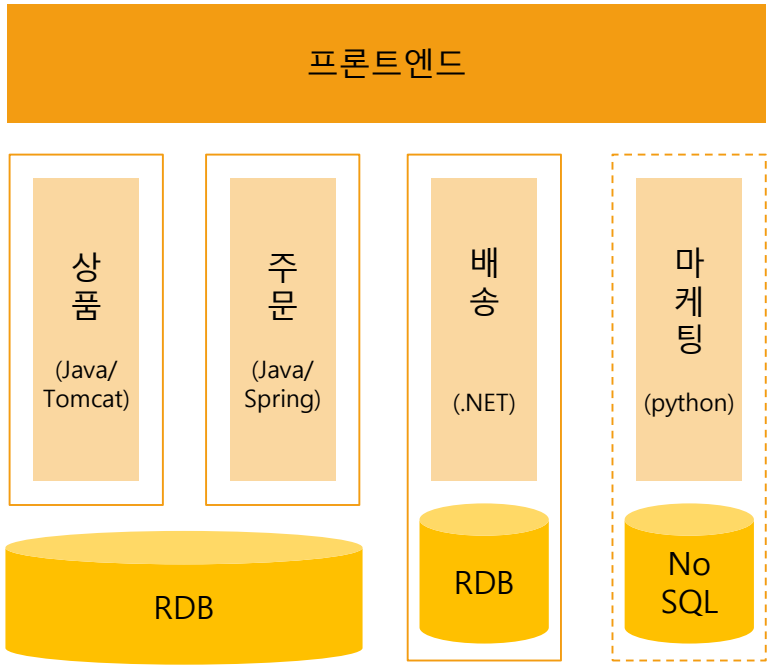


신규 마이크로서비스의 추가 → Minimal Impact
자율적 기술선택(플랫폼 & 데이터레이아웃)

효과 – 기능 확장

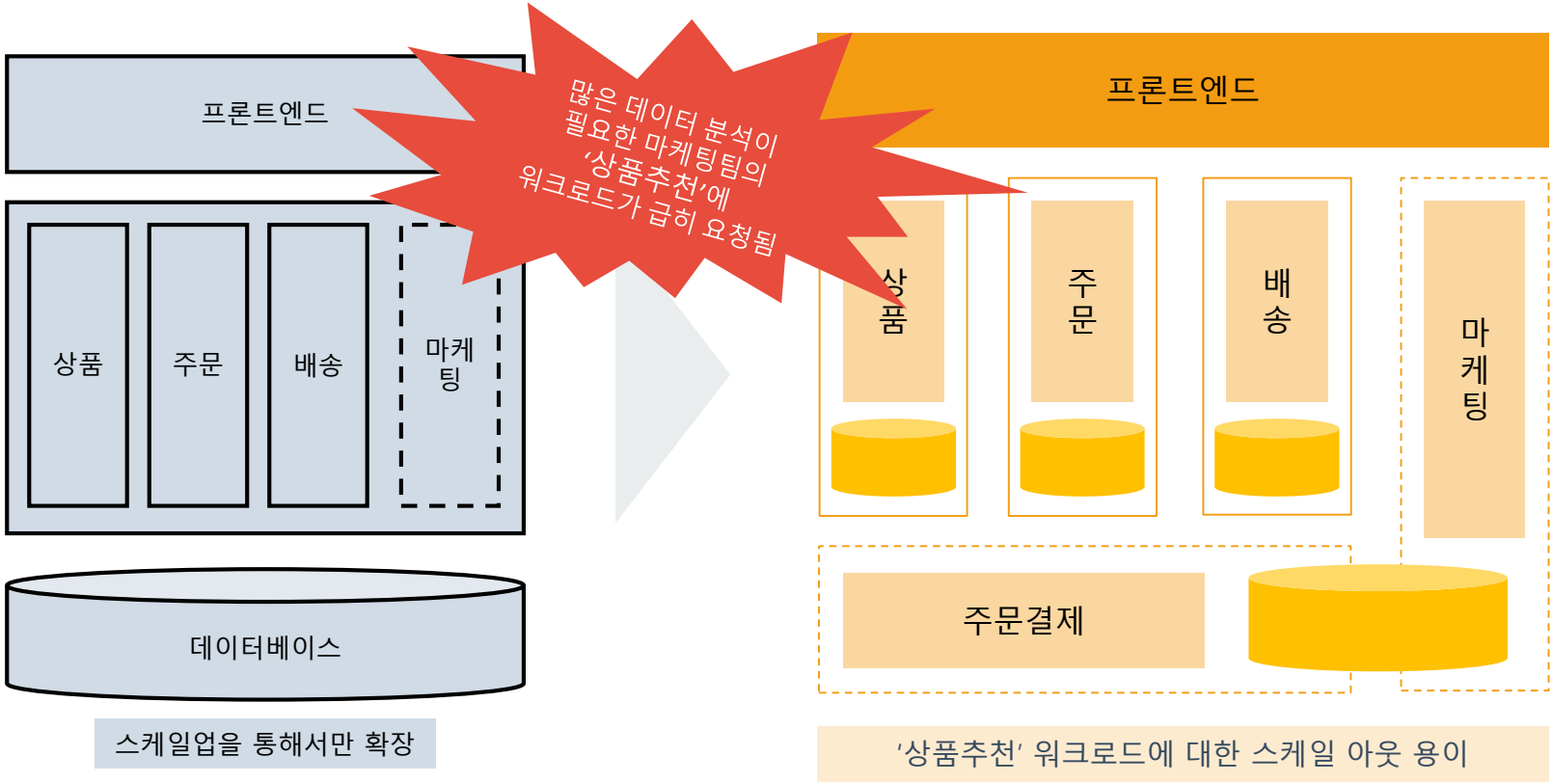


기존 코드의 수정 → Big Impact



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

효과 – 성능 배분



스케일업을 통해서만 확장

'상품추천' 워크로드에 대한 스케일 아웃 용이

Table of content

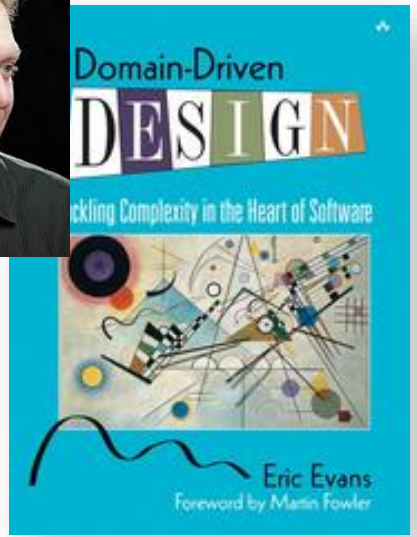
Microservice and
Event-storming-Based
DevOps Project

UEngine

DDD Patterns

Domain-Driven Design & MSA

DDD for MSA



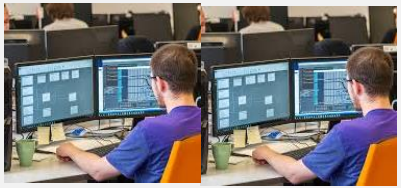
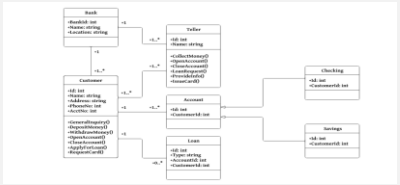
- **Bounded Context 와 Ubiquitous Language**
  → 어떤 단위로 마이크로 서비스를 쪼개면 좋은가?
- **Context Mapping**
  → 서비스를 어떻게 결합할 것인가?
- **Domain Events**
  → 어떤 비즈니스 이벤트에 의하여 마이크로 서비스들이 상호 반응하는가?

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

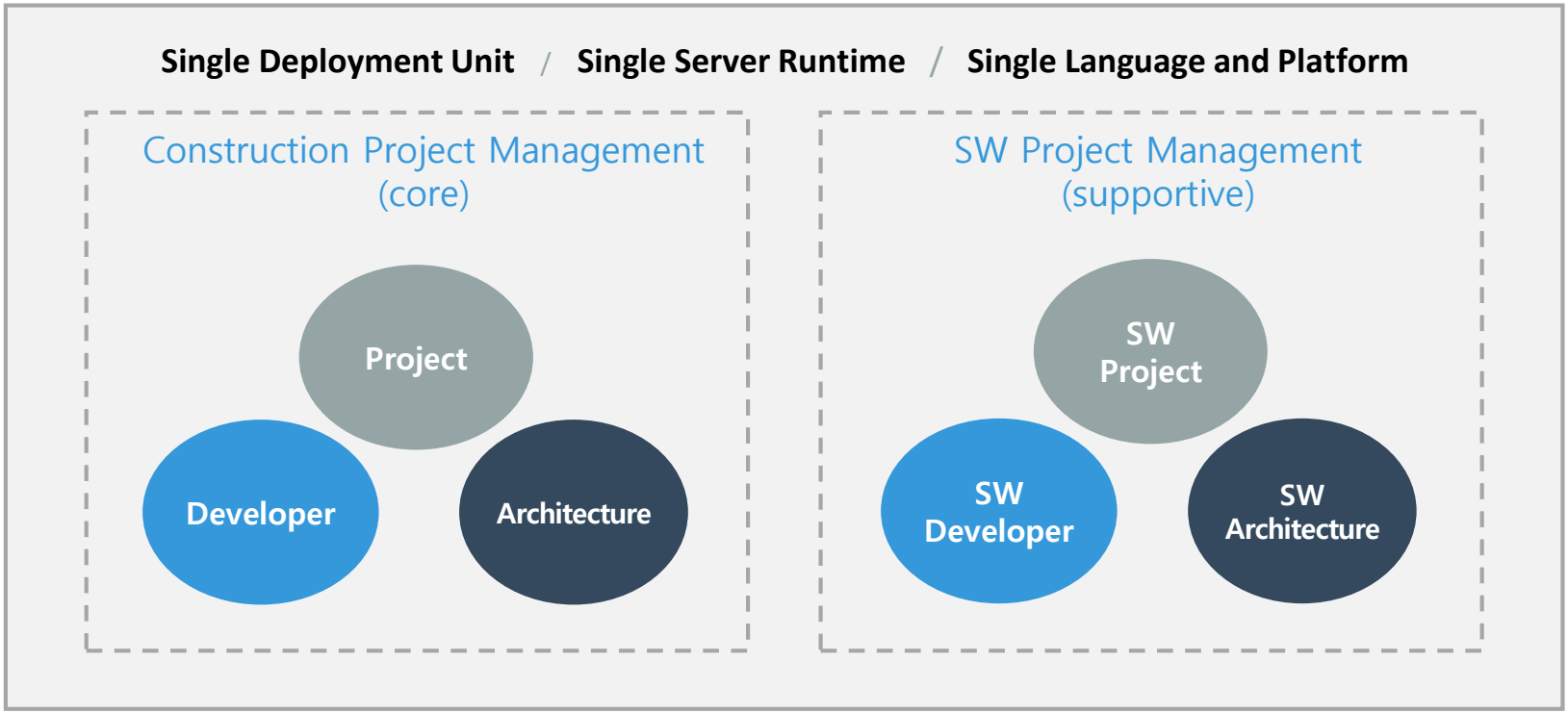
Bounded Context
(한정된 맥락)

Ubiquitous Language
(도메인 언어)

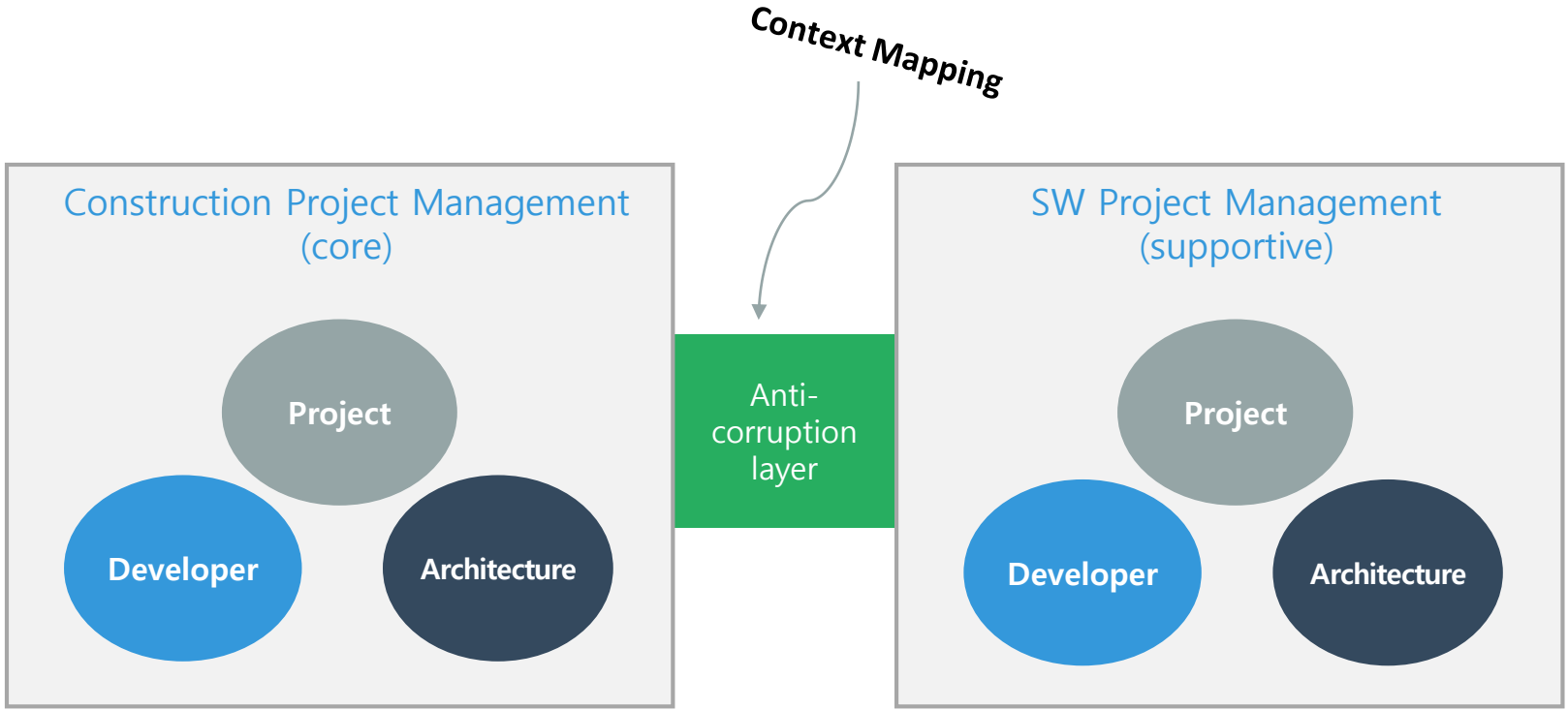
Bounded Context and Ubiquitous Language

	SW	CONSTRUCTION
A Project		
An Architecture		
A Developer		

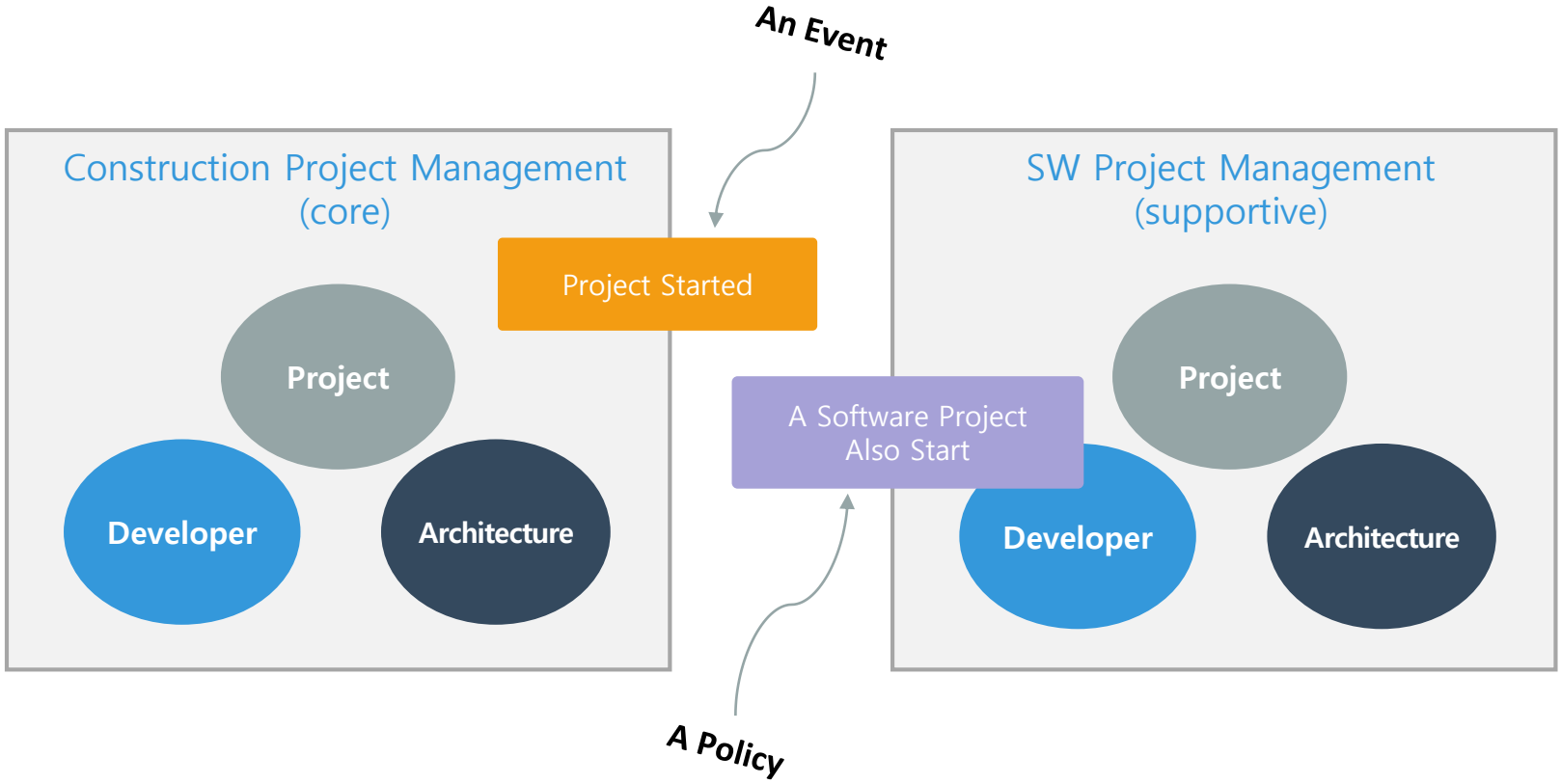
In Monolith :


Single Deployment Unit / Single Server Runtime / Single Language and Platform
Construction Project Management (core)
Project
Developer
Architecture
SW Project Management (supportive)
SW Project
SW Developer
SW Architecture

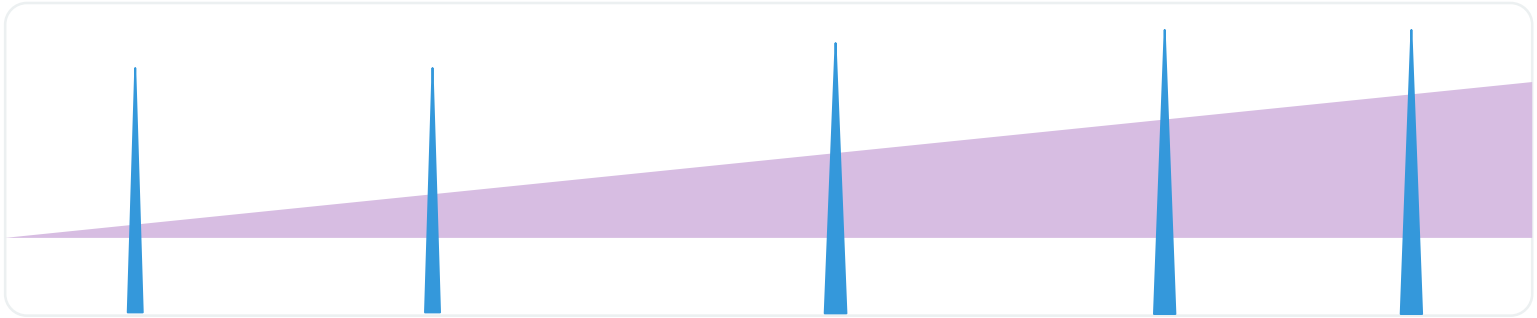
In Microservices :



In Microservices :



5개의 눈금: 어느 굵기로 쪼갤 것인가? 그것이 문제로다...



Aggregate

최소 사이즈 – ACID 가 준수되어야 하는 원자적 단위
- 유연성이 높은

Bounded context

커뮤니케이션이 원활한 단위 – Separation of Concerns 이 좋은

Business capability

비즈니스적 성과 측정과 피버팅이 용이한 단위 – 팀단위와 연결하기 좋은

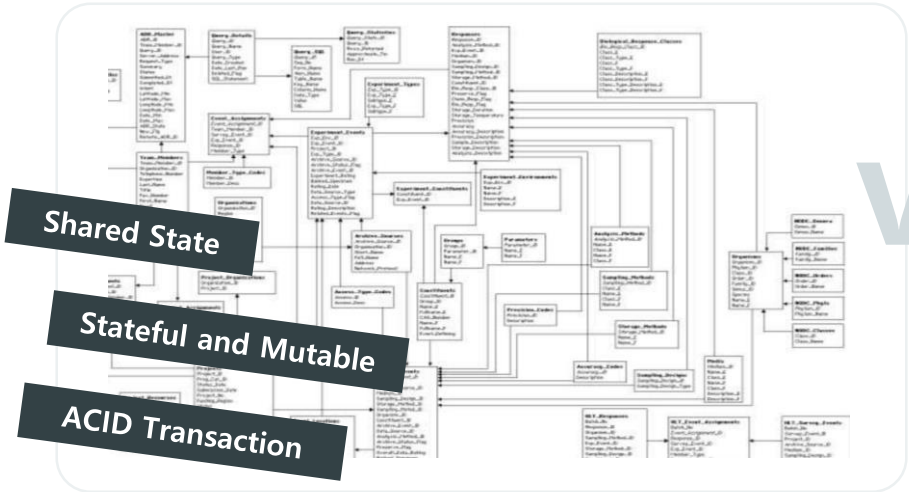
Mini

구현이 용이한 단위
- 점진적 MSA 전환이 용이한 단위

Monolith

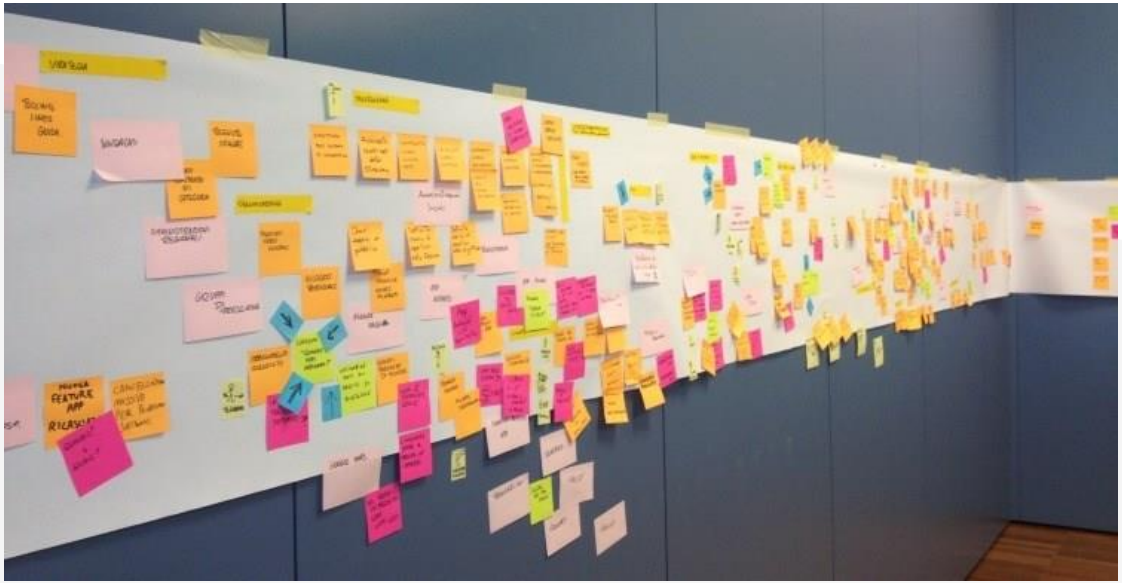
Event Storming : DDD 를 쉽게 하는 방법

- **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 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 도출**되며 이들을 이후 BPMN과 UML 등으로 정재하여 전환할 수 있다.



Event Storming : Prepare

- 3팀 이상으로의 구성 및 팀당 최소 2명 이상 구성
- 큰 종이 시트 및 종이 시트를 여러 장 붙일 수 있는 충분한 벽이 있는 넓은 공간
- 여러 종류의 색깔 스티커, 검은색 펜, 검은색 or 파란색 테이프
- 서서 하는 방식으로 의자 필요 없음.



Type of stickers

Domain Event (Orange)

P.P 형태의 동사

도메인 전문가가 정의
이벤트 퍼블리싱

사용자, 페르소나, 스테이크 홀더

유저 인터페이스를 통해 데이터를
소비하고 명령을 실행하여
시스템과 상호 작용

Definition

정의

도메인에 대한 용어 등의
설명,기술

Command (Sky Blue)

명령

현재형으로 작성,
행동, 결정 등의 값들에 대한
정의 UI 혹은 API

Aggregate (Yellow)

집합체

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

Read Model (Green)

리드모델

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

External System (Pink)

외부 시스템

시스템 호출을 암시 (REST)

Policy (Lilac)

업무정책

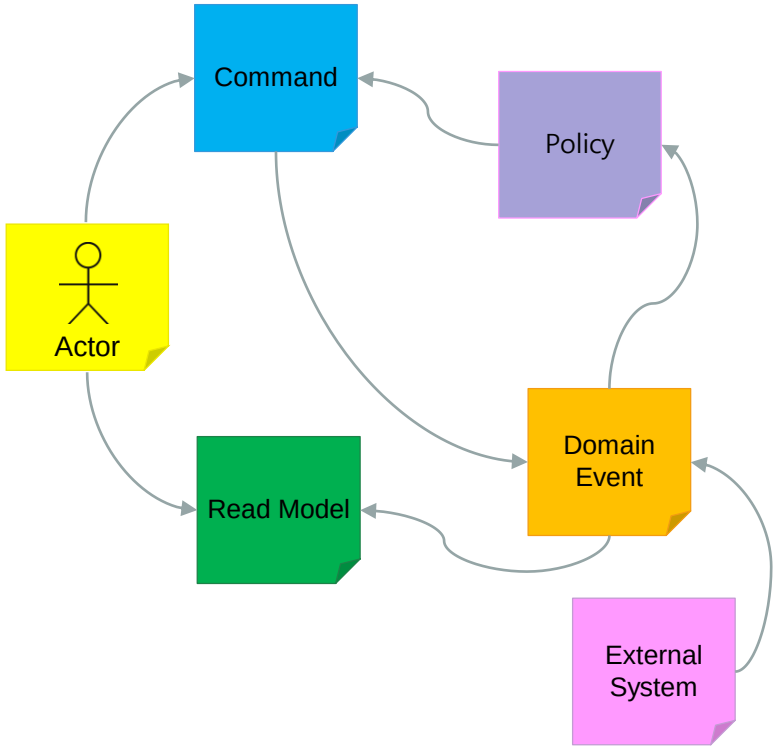
이벤트에 대한 반응
(서브스크라이브)
비즈니스 룰 엔진 등

Comment Or Question (Purple)

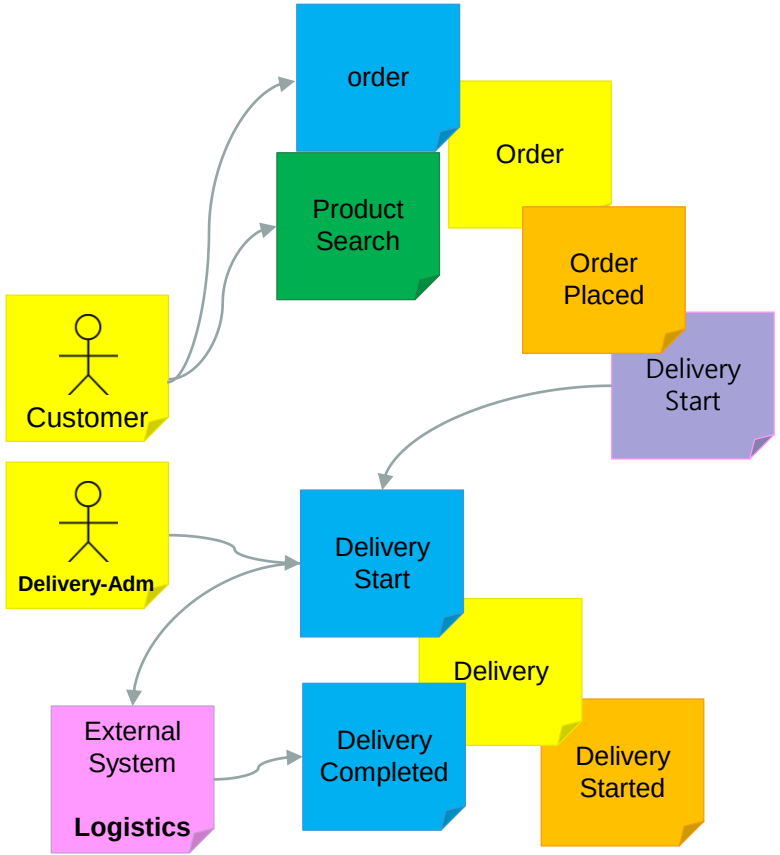
의견 또는 질문

추기적인 네용 입력,
예측되는 Risk

Examples



[Meta Model]



[12st Sample]

Firstly, Event Discovery

- 오렌지(주황색) 스티커 사용
- 각 도메인 전문가들이 개별 도메인 이벤트 목록 작성
- 이벤트는 도메인 전문가와 비즈니스 관계자 서로 이해할 수 있는 의미 있는 방식으로 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

잘 도출된 이벤트와 아닌 경우

O

다른 팀에서 관심 가질만함

다른 팀에서 관심 가질만함

적당한 사이즈임

X

It's a command

It doesn't make any state change

Too technical and too fine-grained

Be business level

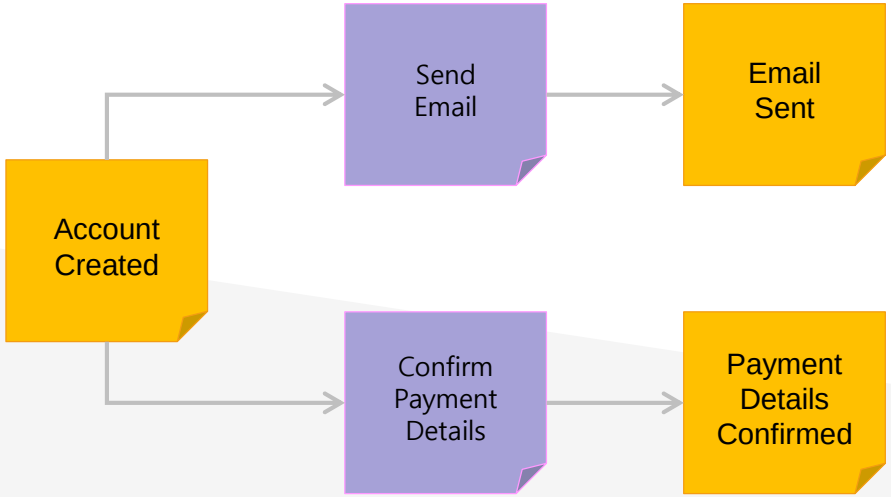
Nobody will be interested in this event

Command 와 Actor, UI 도출

- Command는 파란색 스티커 메모를 사용
- 도메인 분석에서 시스템 설계의 첫 단계
- 관심있는 비즈니스 프로세스를 구현하는 시스템을 구축하려면 이러한 이벤트가 발생하는 방식에 대한 질문 등으로 정의한다.
- 목표는 이벤트가 효과를 기록하는 원인을 찾는 것
- 예상되는 이벤트 트리거 유형의 예
  - Operator 결정을 내리고 명령을 내릴 시
  - 외부 시스템 또는 센서가 자극될 시
  - 정해진 시간 경과 시
- 마이크로 서비스 구현에서 API가 될 수 있다.
- Command를 사용하는 인간 혹은 주체는 노란색 스티커 메모를 사용한다.

Policy 도출

- 라일락 색의 스티커 사용
- Policy는 이벤트가 발생한 후 발생하는 반응형 논리
- Policy는 다른 Command를 트리거
- Policy는 process 같이 사람이 행하는 수동 동작 및 자동화 될 수 있다.



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

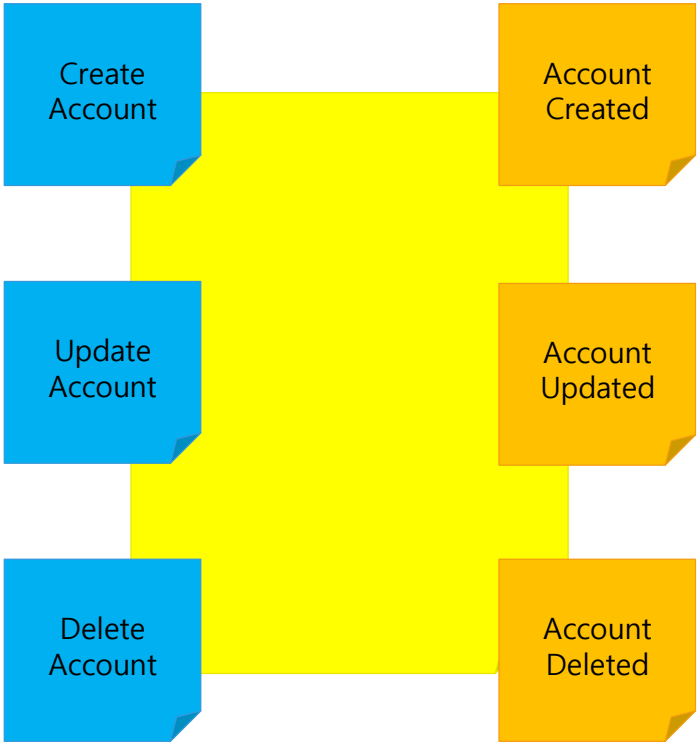
Read Model 도출

- 녹색 스티커 사용
- 이벤트를 생성하기 위해 Command을 실행하는데 필요한 데이터의 이해
- The Data needed in order to make that decision
  사용자의 의사 결정 과정을 지원하는 도구로 사용됨
- 각 Command 및 Event에 대해 필요한 속성 및 데이터 요소에 대한 설명

" Read Model Derivation "

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

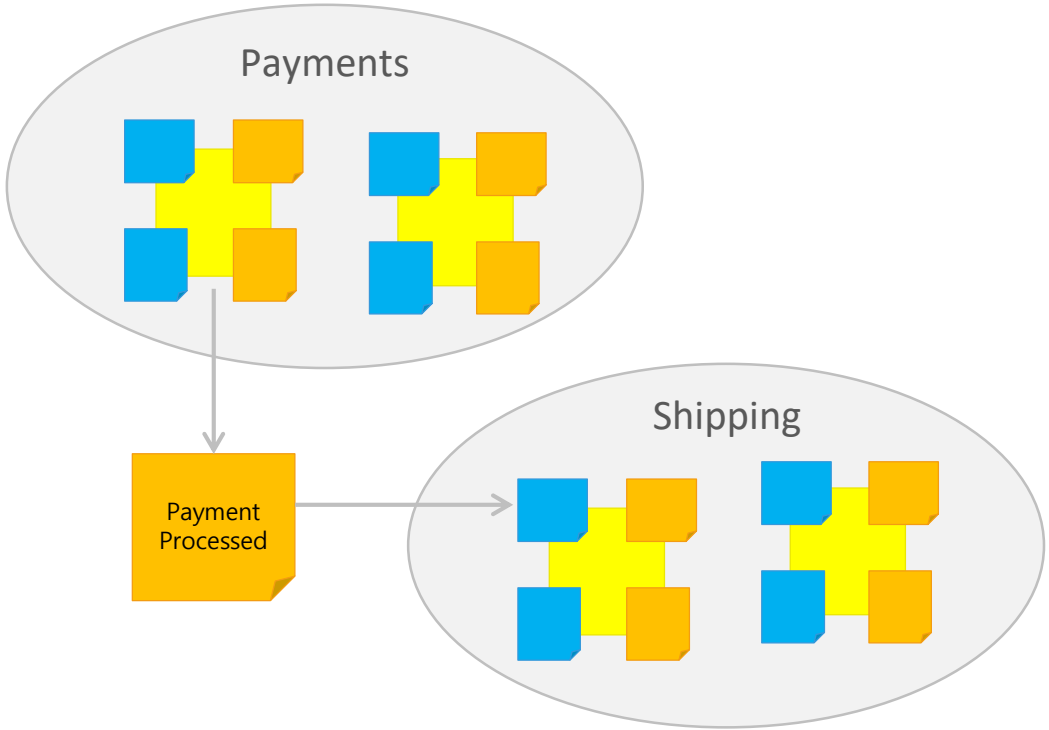


Bounded Contexts 도출

- 이벤트의 내용을 정의하고 시스템의 경계를 구분

- 찾는 방식은 두가지 유형으로 구분되어진다

  - Time Boundary
  - Subject Boundary



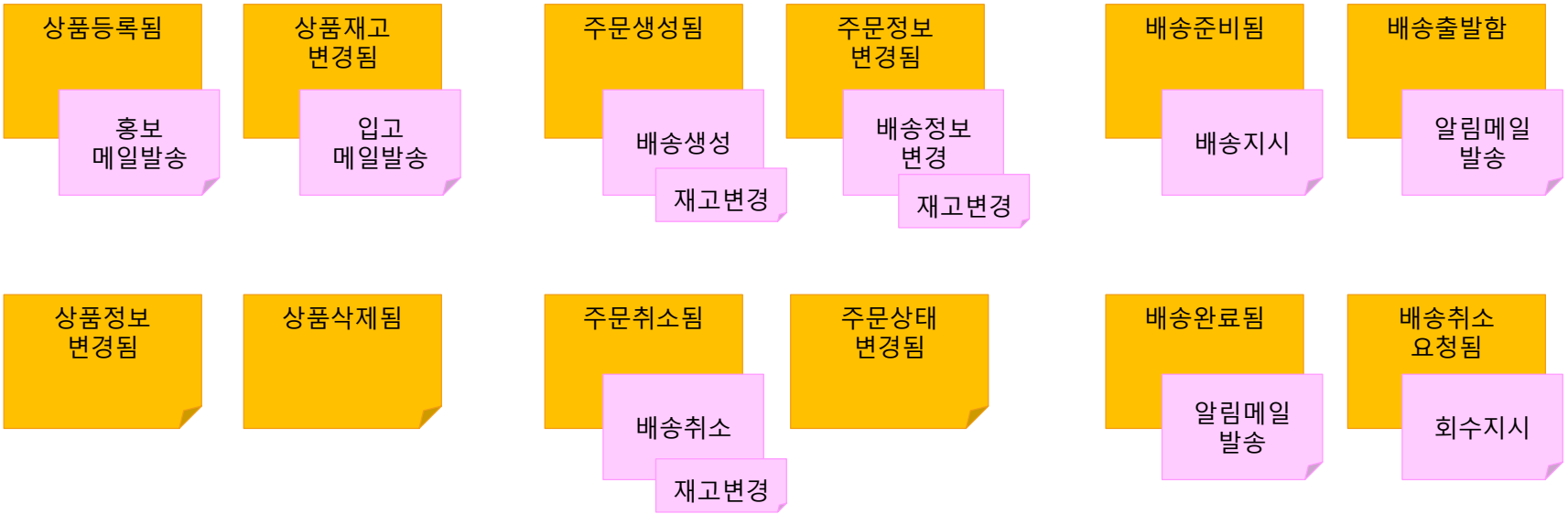
Source: https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7, Kevin webber

Lab Time – Event 들을 먼저 도출

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

이벤트 : 오렌지 스티커!

Lab Time – Policy 도출



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

규칙 : 라일락 스티커!

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 (External) 또는 내부)이 될 수 있음

Lab Time – Aggregate 도출

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.

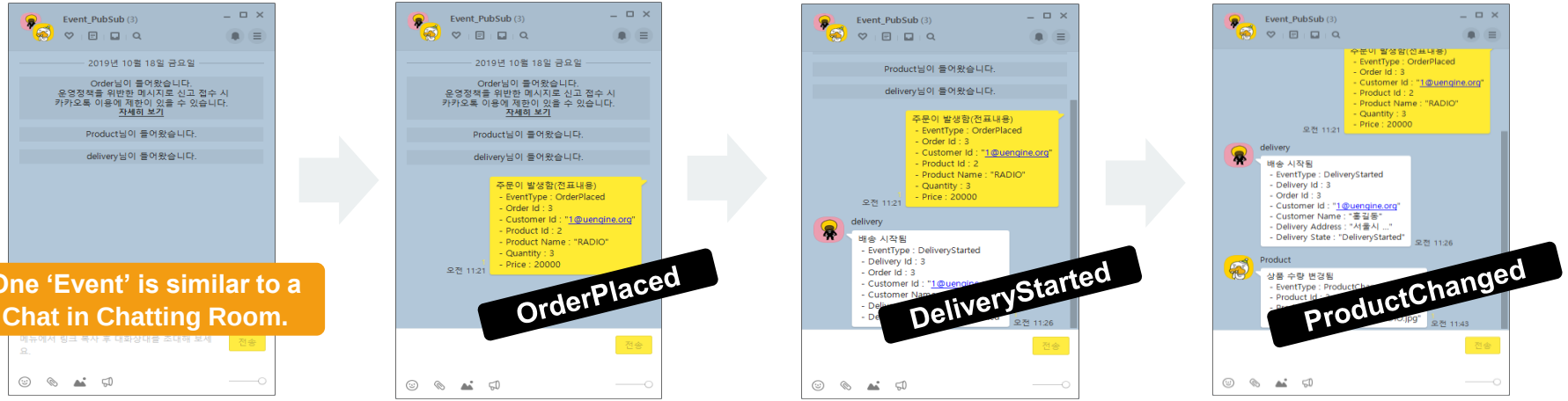
Lab Time – Bounded Context 도출

- ***개념적:***
  Bounded Context 는 동일한 문맥으로 효율적으로 업무 용어 (도메인 클래스)를 사용할 수 있는 객체 범위를 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

REAL CASE

12st Microservices Event Sample :



One 'Event' is similar to a Chat in Chatting Room.

Chatting Room

```
>
>{"eventType":"OrderPlaced","timestamp":"20190916151922","stateMessage":"Order Placed","productId":2,"orderId":3,"productName":"RADIO","quantity":3,"price":20000,"customerId":"1@uengine.org","customerName":"Hong Gil Dong","customerAddr":"Seoul"}
>
```

```
r.bat --broker-list http://localhost:9092 --topic eventTopic
>{"eventType":"DeliveryStarted","timestamp":"20190916151922","stateMessage":"Delivery Started","deliveryId":3,"orderId":3,"customerId":"1@uengine.org","customerName":"Hong Gil Dong","deliveryAddress":"Seoul ...","deliveryState":"DeliveryStarted"}
>
```

```
12-2.1.0₩bin₩windows>kafka-console-producer.bat --broker-list http://localhost:9092 --topic eventTopic
>{"eventType":"ProductChanged","timestamp":"20190916151922","stateMessage":"Product Changed","productId":2,"productName":"RADIO","productPrice":20000,"productStock":14,"imageUrl":"/goods/img/RADIO.jpg"}
>
```


```
{"eventType":"OrderPlaced","timestamp":"20190916151922","stateMessage":"Order Placed","productId":2,"orderId":3,"productName":"RADIO","quantity":3,"price":20000,"customerId":"1@uengine.org","customerName":"Hong Gil Dong","customerAddr":"Seoul"}
{"eventType":"DeliveryStarted","timestamp":"20190916151922","stateMessage":"Delivery Started","deliveryId":3,"orderId":3,"customerId":"1@uengine.org","customerName":"Hong Gil Dong","deliveryAddress":"Seoul ...","deliveryState":"DeliveryStarted"}
{"eventType":"ProductChanged","timestamp":"20190916151922","stateMessage":"Product Changed","productId":2,"productName":"RADIO","productPrice":20000,"productStock":14,"imageUrl":"/goods/img/RADIO.jpg"}
```

Table of content

Microservice and
Event-storming-Based
DevOps Project

UEngine

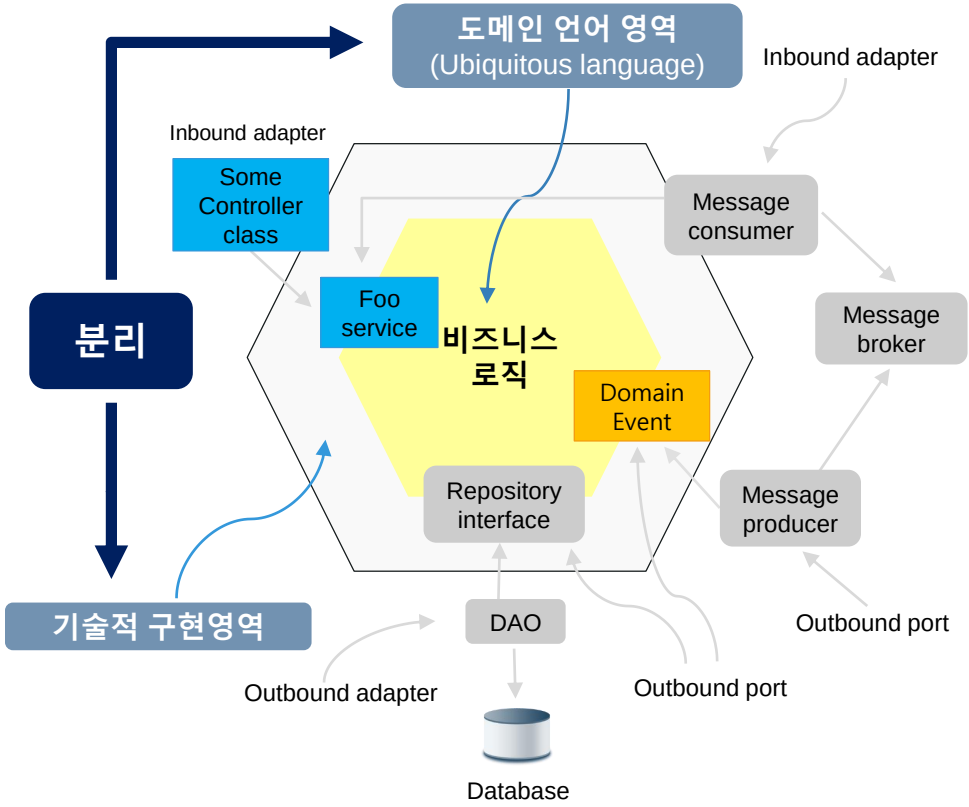
Microservice Implementation Pattern (1)

- **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.

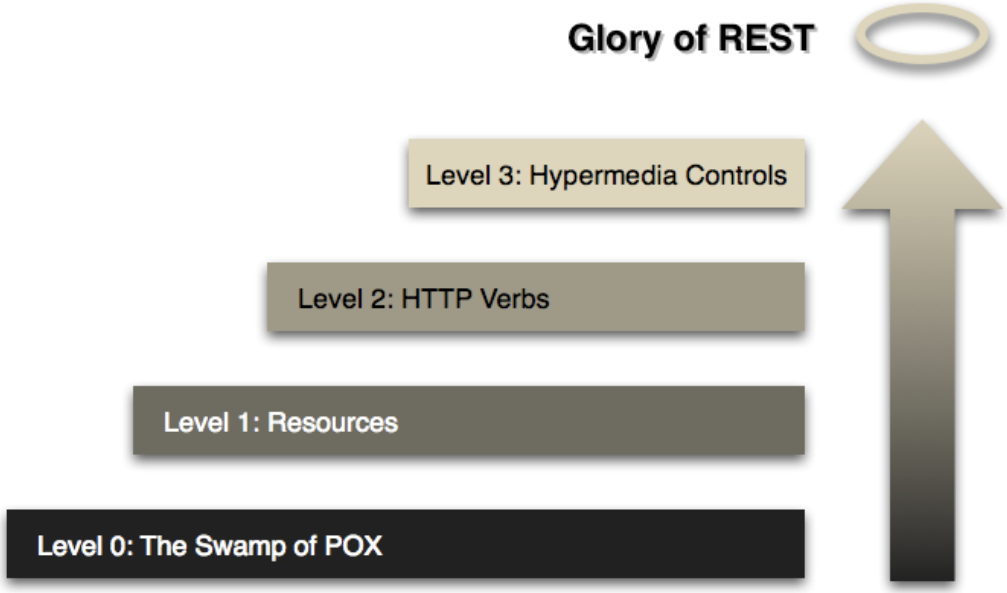
https://alistair.cockburn.us/Hexagonal+architecture



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

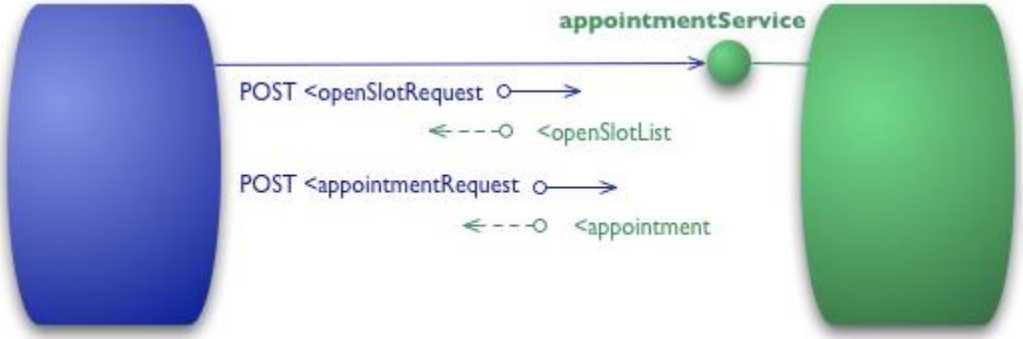
Exposing Service: REST Maturity Model



82

Level 0: Swamp of POX

- Use as a RPC, returns full serialized document

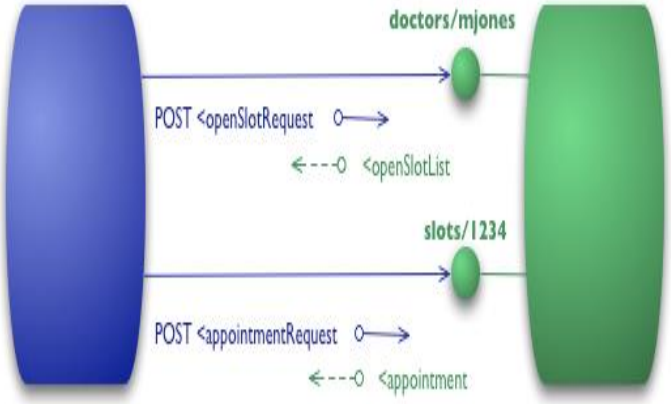



```
<openSlotList>
 <slot start = "1400" end = "1450">
 <doctor id = "mjones"/>
</slot>
<slot start = "1600" end = "1650">
<doctor id = "mjones"/> </slot> </openSlotList>
```


83

Level 1: Resources

Level 1 tackles the question of handling complexity by using divide and conquer, breaking a large service endpoint down into multiple resources.




```
<openSlotList>
  <slot _link = ”http://openslots/1234"/>
  <slot _link = ”http://openslots/1235"/>
</openSlotList>
```

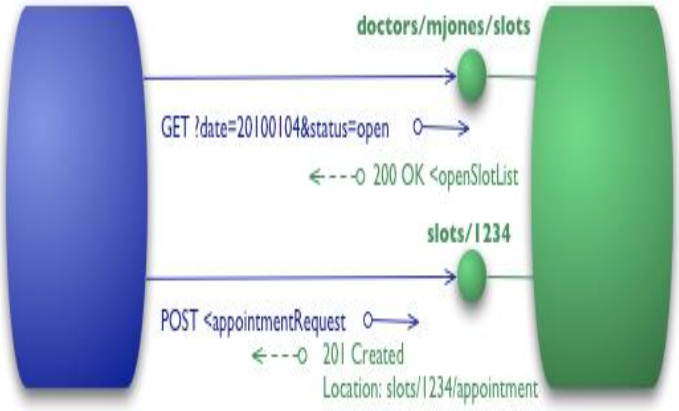
```
# http://openslots/1234
<slot start = "1400" end = "1450">
 <doctor id = "mjones"/>
</slot>
```

```
# http://openslots/1235
<slot start = "1600" end = "1650">
<doctor id = "mjones"/> </slot>
```


84

Level 2: HTTP Verbs

Level 2 introduces a standard set of verbs so that we handle similar situations in the same way, removing unnecessary variation.



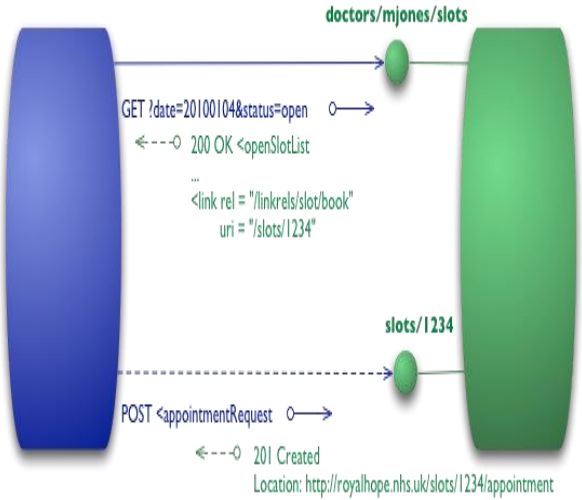
Operation	HTTP / REST
Create	PUT / **POST**
Read (Retrieve)	**GET**
Update (Modify)	**PUT** / PATCH
Delete (Destroy)	**DELETE**

```
# http://openslots/1234
<slot start = "1400" end = "1450">
 <doctor id = "mjones"/>
</slot>
```


85

Level 3: Hypermedia Controls

Level 3 introduces discoverability, providing a way of making a protocol more self-documenting.




```
<appointment>
<slot id = "1234" doctor = "mjones" start = "1400" end =
"1450"/> <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel" uri =
"/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest" uri =
"/slots/1234/appointment/tests"/>
  <link rel = "self" uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime" uri =
"/doctors/mjones/slots?date=20100104@status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo" uri =
"/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help" uri = "/help/appointment"/>
</appointment>
```


86

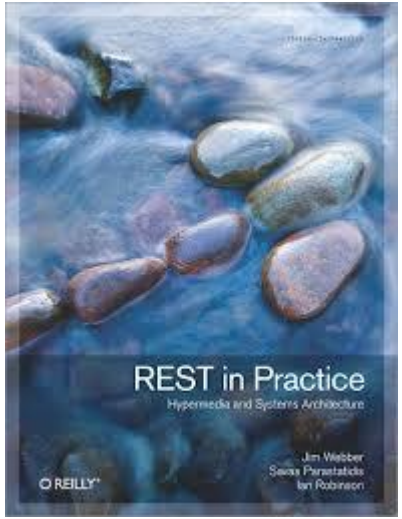
Level 3: In another words, HATEOAS

- Hypermedia As The Engine Of Application State
- A RESTful API can be compared to a website. As a user, I only know the root URL of a website. Once I type in the URL (or click on the link) all further paths and actions are defined by other links. Those links may change at any moment, but as a user, all I need to know is the root URL and I'm still able to use the website.

87

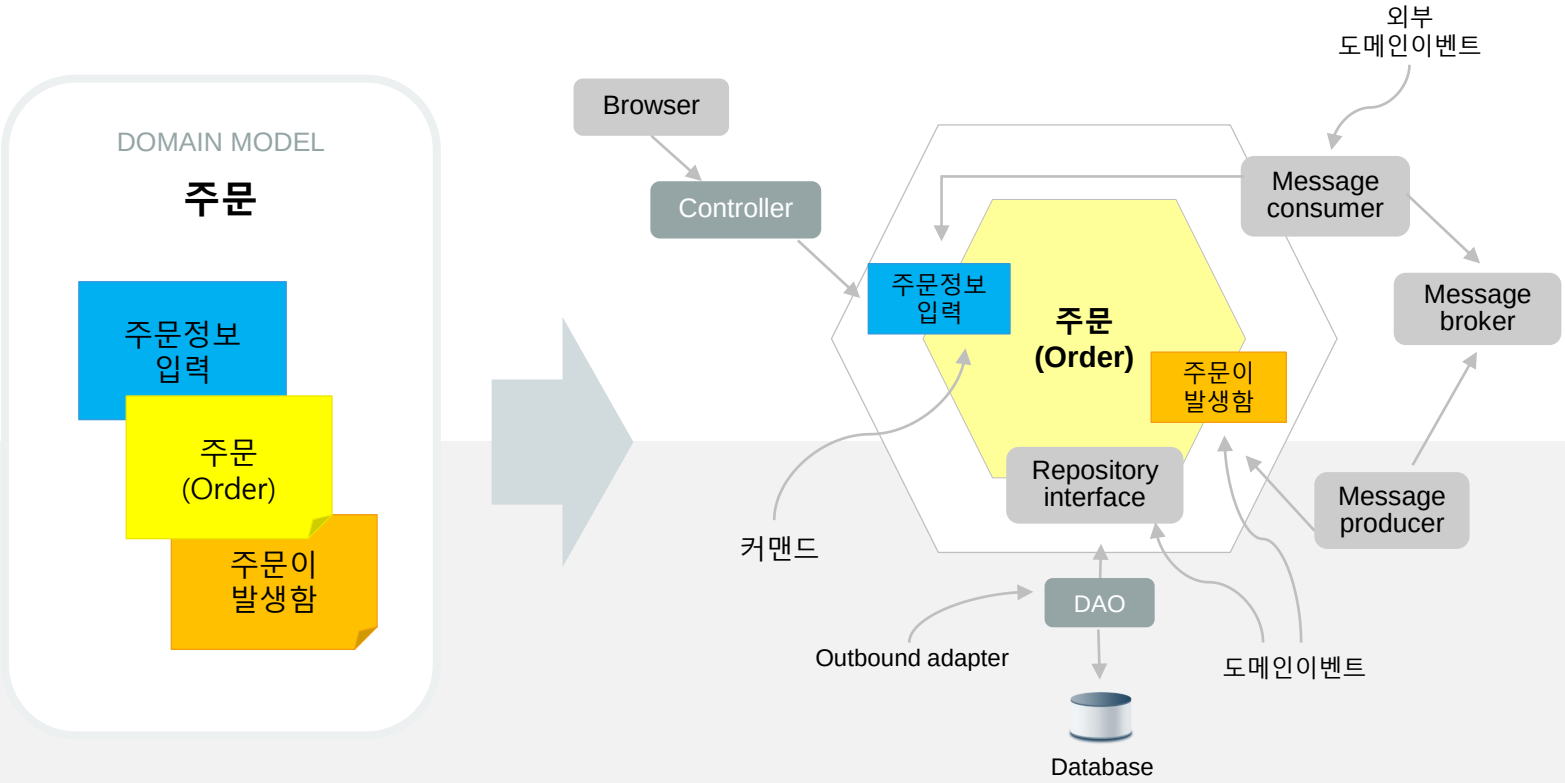
Recommended Book

- REST in Practice
- Domain Driven Design Quickly



88

Applying Hexagonal Architecture

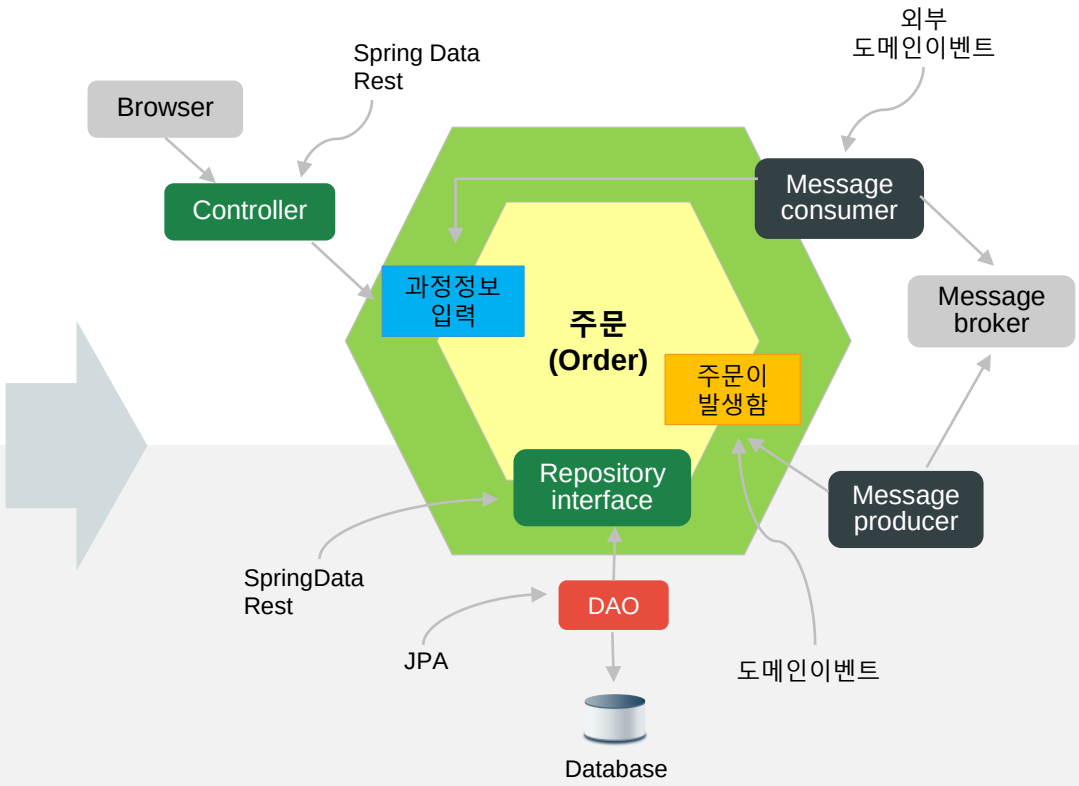


Applying MSA Chassis

DOMAIN MODEL

Spring Boot / Cloud

- Spring Boot
- Spring Data Rest
- JPA
- Spring Cloud Stream



이벤트 스토밍 결과에서 구현 기술 연동

요소	구현체	미들웨어/프레임워크
이벤트 (Domain Event)	• 도메인 이벤트 클래스 (POJO)	• 카프카 퍼블리시 • Rabbit MQ
커맨드 (Command)	• 서비스 객체 • 레포지토리 객체 (PagingAndSortingRepository)	• Spring Data REST • RESTeasy
결합물 (Aggregate)	• 엔티티 클래스 (ORM) • 도메인 모델	• JPA Entity • Value Objects
정책 (Policy)	• 엔티티 클래스의 Hook에 정책 호출 구문 주입 • CDC 를 통한 이벤트 후크	• 카프카 Event Listening Code • JPA Lifecycle Hook • CDC (Change Data Capturing)
바운디드 컨텍스트	• 마이크로 서비스 (후보)	• Spring Boot

분석/설계 (Sticker)

Order

- orderId
- name
- userId
- price
- quantity

Aggregate → Domain Model

```
@Entity
public class Order{

  Long id;
  String name;
  String customerId;
  double price;
  int quantity;

… setter/getters …

}
```

```
@Entity
public class OrderDetail {

….

… setter/getters …

}
```

주문
(POST)

주문수정
(PATCH)

주문삭제
(DELETE)

Command → CRUD 에 해당하면? Repository Pattern 으로 자동생성

```
public interface OrderRepository extends
        PagingAndSortingRepository<Order, Long>{


}
```

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
- name
- userId
- price
- quantity

개발 (POJO)

```
public class OrderPlaced{

  Long orderId;
  String name;
  String userId;
  double price;
  int quantity;

… setter/getters …

}
```

실행 (JSON)

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

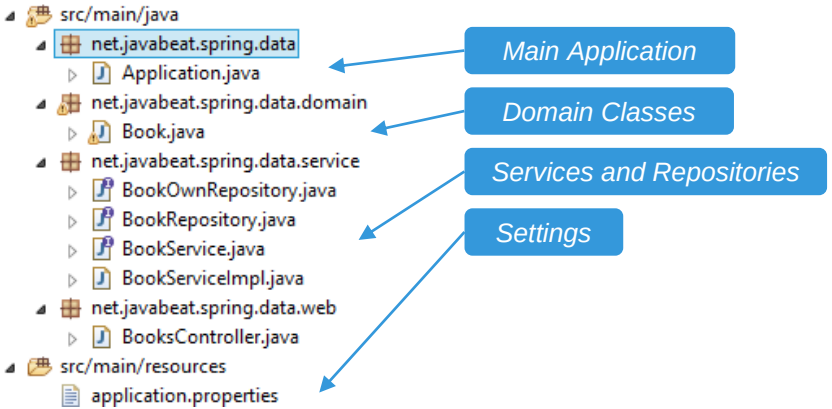
OrderPlaced

- orderId
- name
- userId
- price
- quantity

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);
          …

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

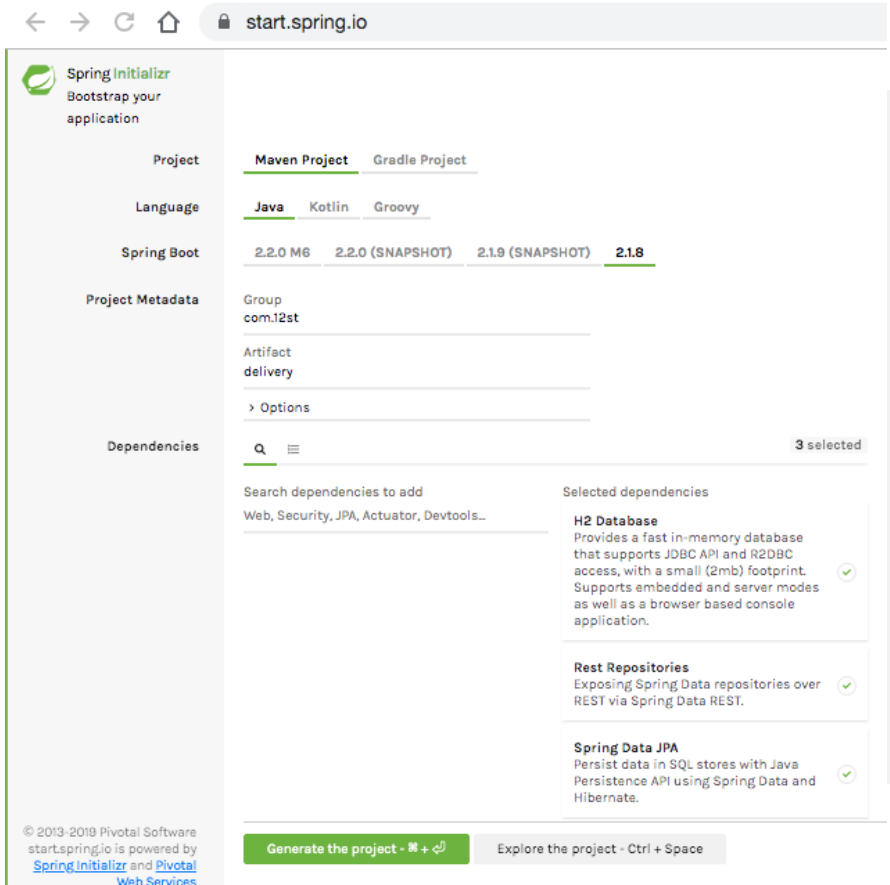
Spring Boot : A MSA Chassis



- 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.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

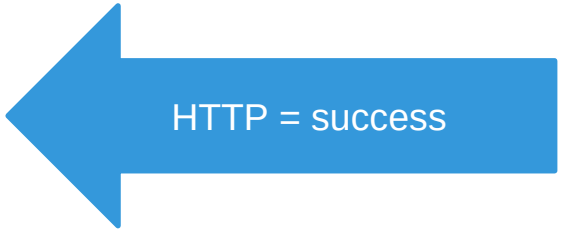
```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building ….
[INFO] ------------------------------------------------------------------------
[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"
        }
    }
}
```



Aggregate → 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<Orde> orders;
}
```

주문

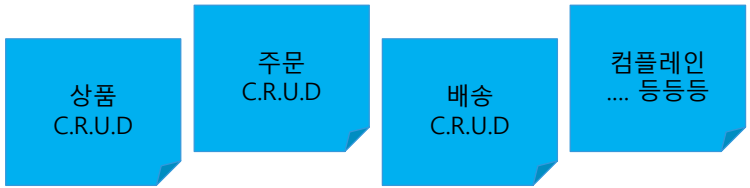
```
@Entity
public class Order {

    @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

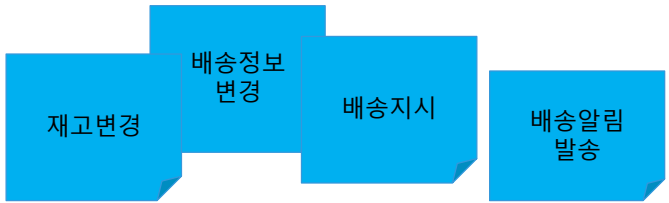
비즈니스 객체(명사)에서 추출 가능한 CRUD 의 경우




```
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);
    }
}
```

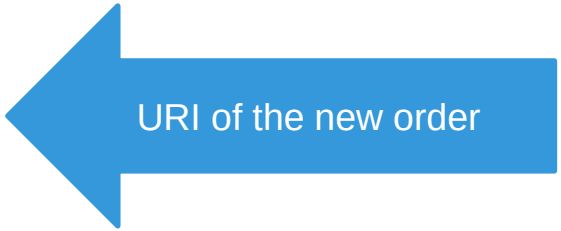
Test Generated Services

```
$ 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 poductId=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"
}
```

Create a delivery for the order

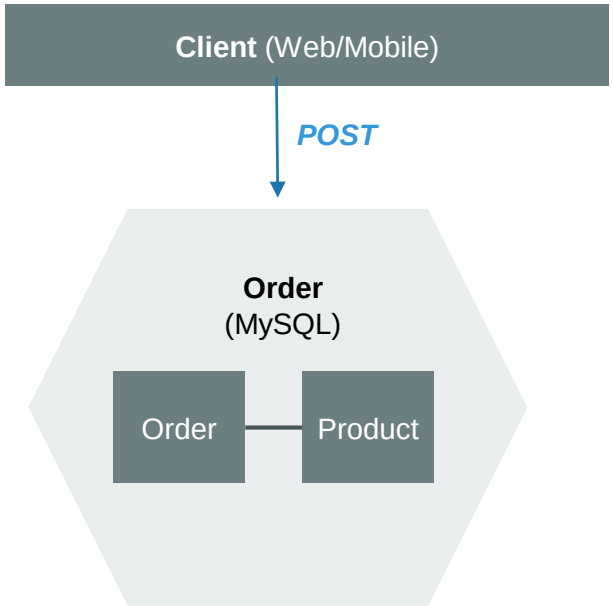
$ http **http://localhost:8088/orders/1/delivery**


```
{
"_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
}
```

Monolithic 에서의 분산 트랜잭션 – 이슈없음

In-consistent | Consistent

- 서비스구성 (모놀로씩)



- 조회화면

주문량 : 0 , 재고량 : 10

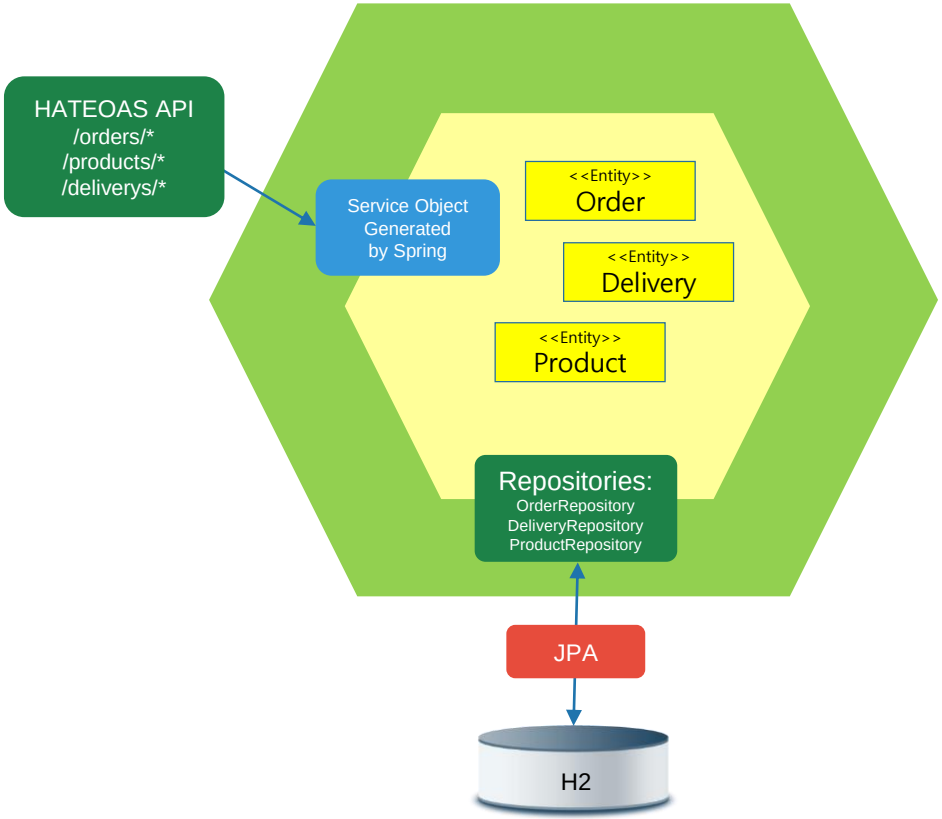
주문량 : 1 , 재고량 : 9

주문량 : 2 , 재고량 : 8

항상 일치함

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

Nice but Too big (monolith)



Plan to add…

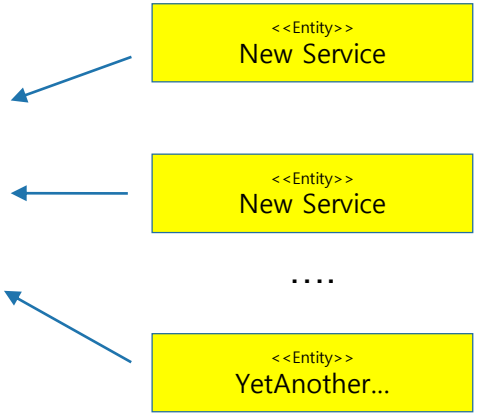
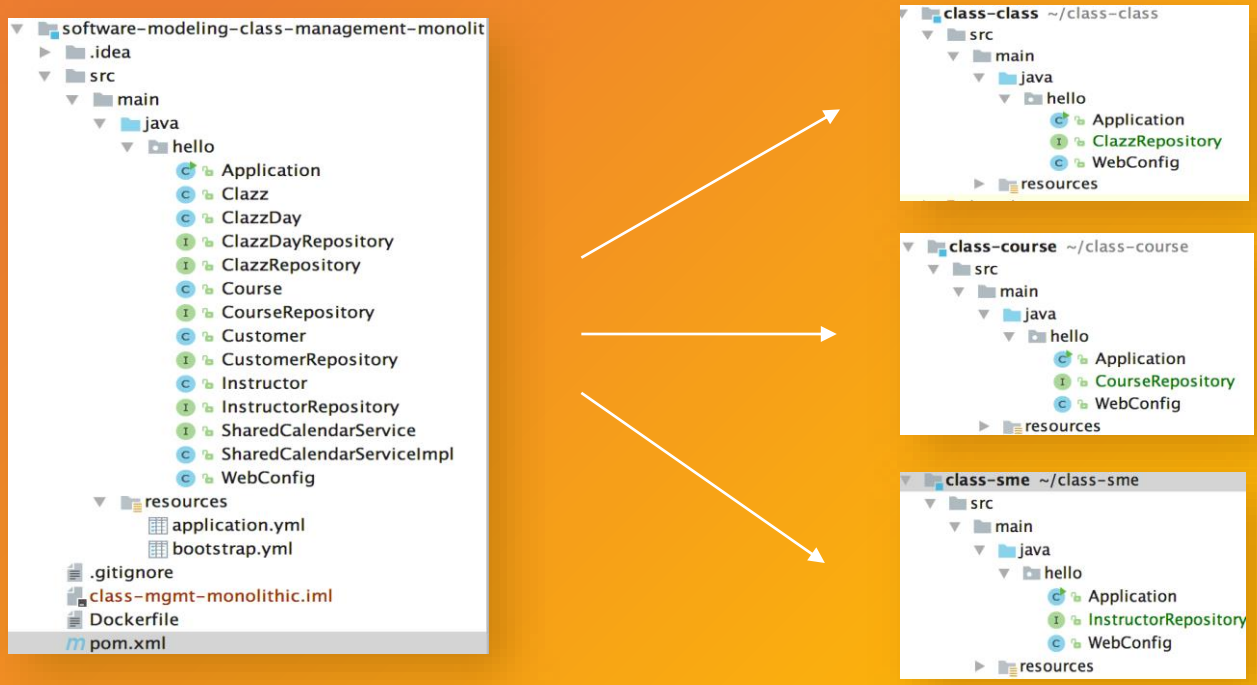


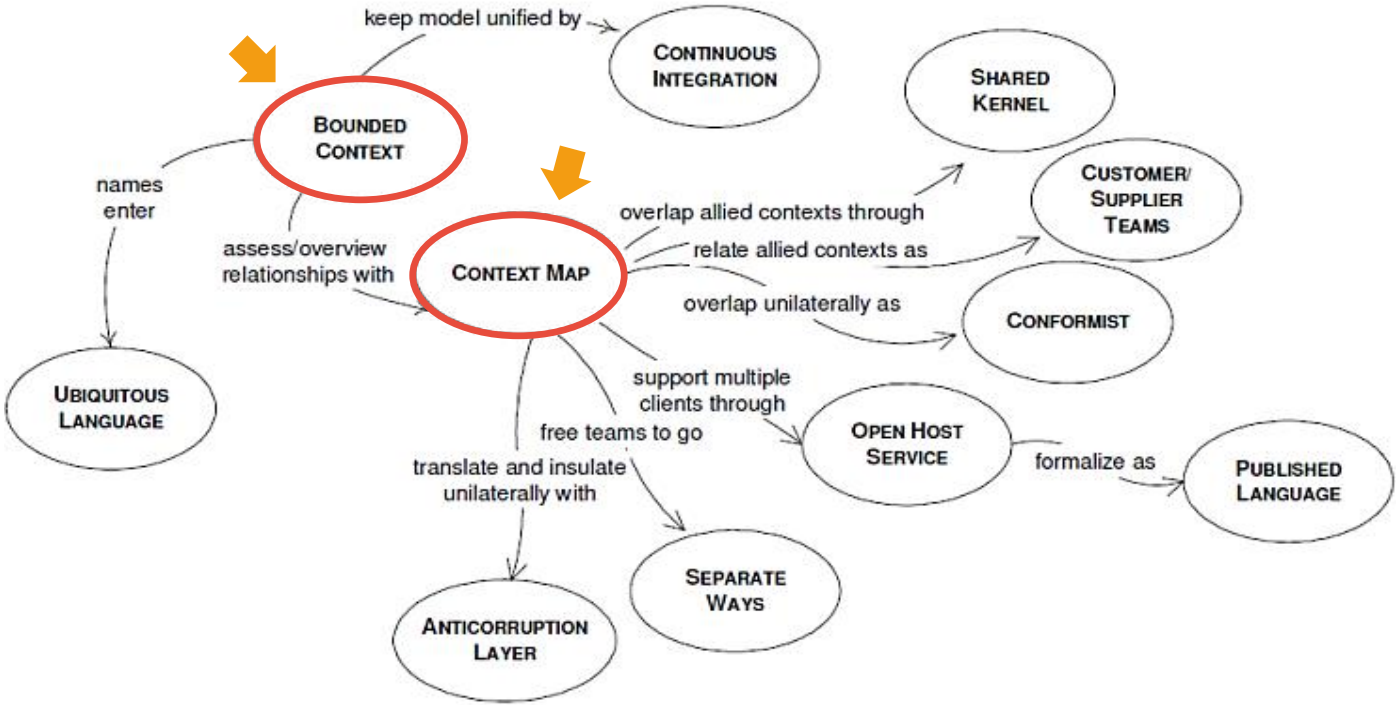
Table of content

From Monolith to Microservices

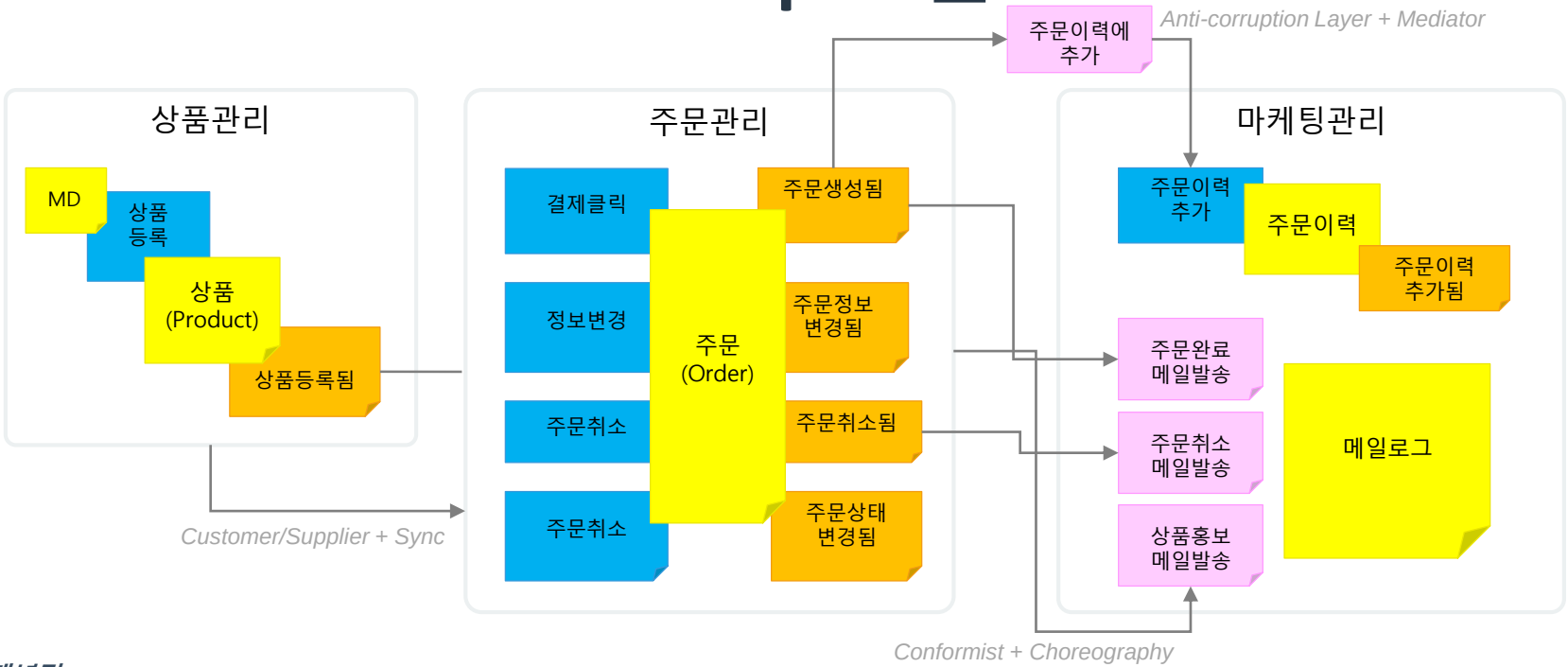
https://github.com/event-storming



DDD Pattern Applied



Lab Time – Context Map 도출

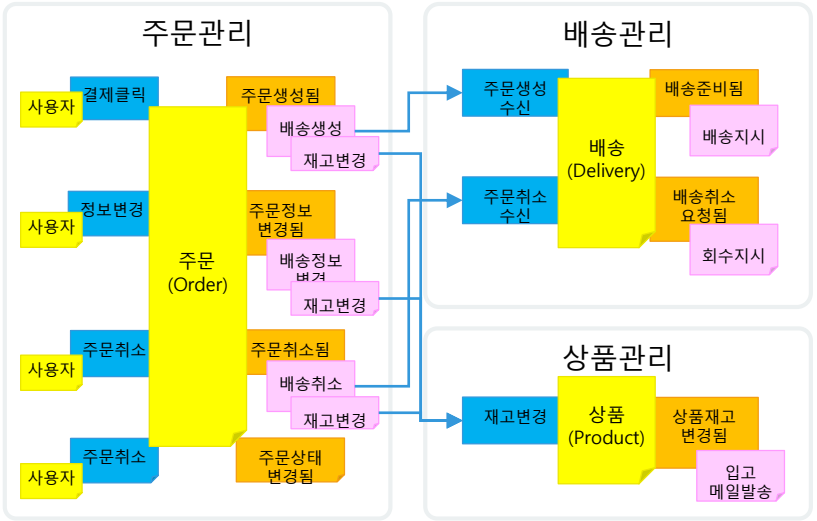


- ***개념적:***
  BC 간의 정보참조의 릴레이션, 혹은 이벤트가 발생한 이후의 동반된 행위의 호출의 관계를 선으로 표시함. Upstream → Downstream 으로 정보가 내려가게 됨.
- ***구현관점:***
  자치적으로 구현설계가 이루어질 수 있는 BC 간에 커뮤니케이션을 위해서는 데이터의 적절한 변환이나 표준화, Pub/Sub 등의 커뮤니케이션 방법 및 데이터 전환에 대한 이슈가 발생하게 됨. 해당 이슈를 적절한 패턴을 선택하여 실제 연동을 위한 Adapter 를 개발해야 함.

Lab Time – Topology Consideration

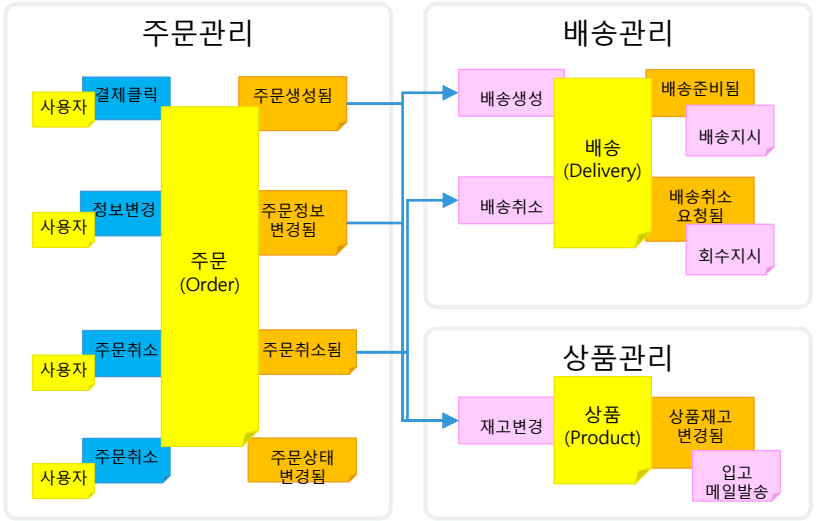
라이락 스티커 (Policy) 를 어디에 둘 것인가? - Orchestration or Choreography? Or Mediation?

Orchestration



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

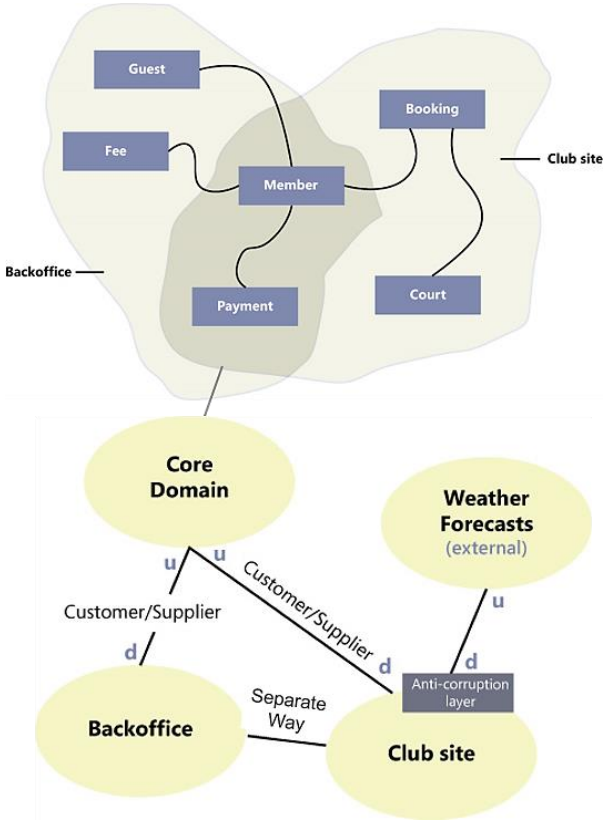
Choreography



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 feat ures 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 t eams 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 c an reduce coupling and ease understanding.



Eric Evans, Domain-Driven Design, 2003.

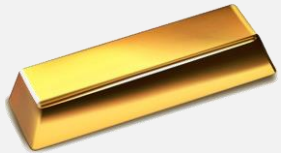
마이크로 서비스간의 서열과 역학관계

" 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

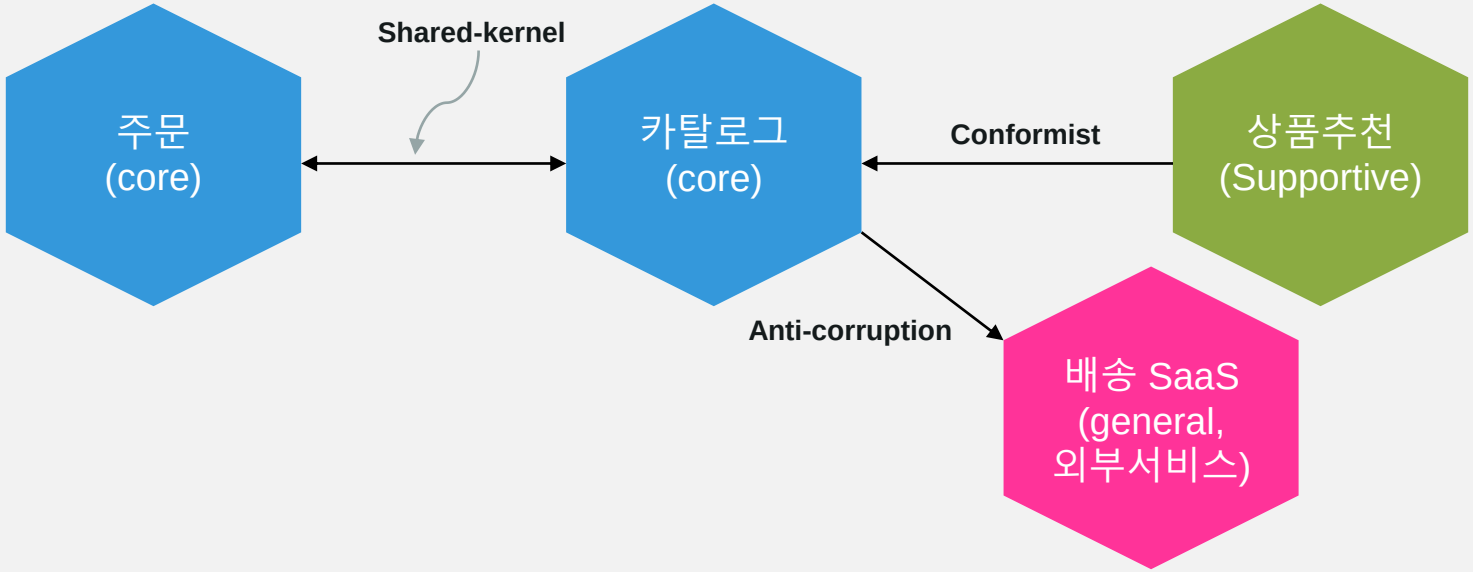
3순위 : General Domain

- SaaS 등을 사용하는게 더 저렴한, 기업 경쟁력과는 완전 무관한

예) 결재, 빌링 서비스 등

마이크로 서비스간의 서열과 역학관계

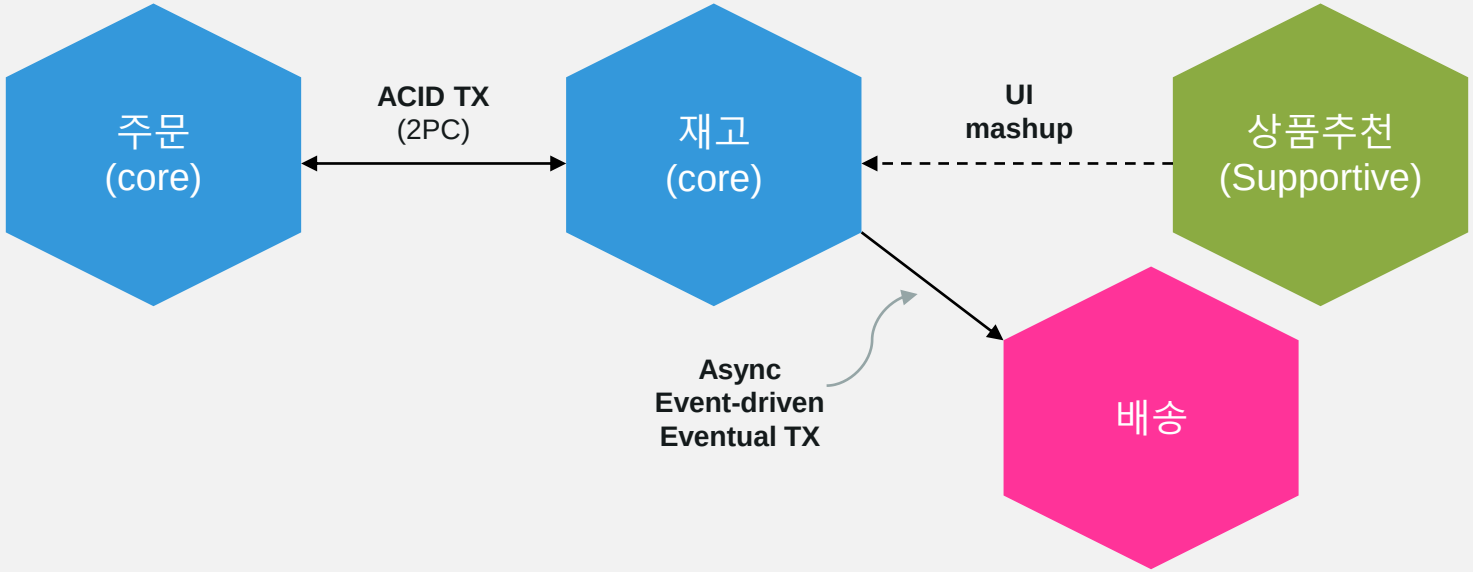
" 어느 마이크로 서비스의 인터페이스를 더 중요하게 관리할 것인가? "



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

마이크로 서비스간의 서열과 역학관계

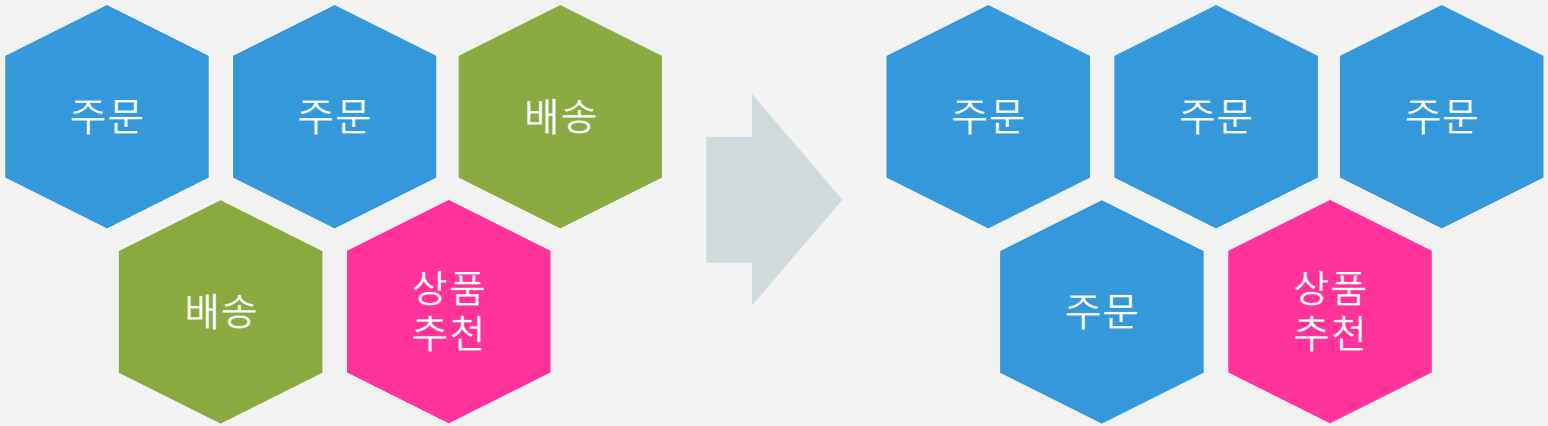
" 마이크로서비스간 트랜잭션의 묶음을 어떻게 할 것인가? "



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.

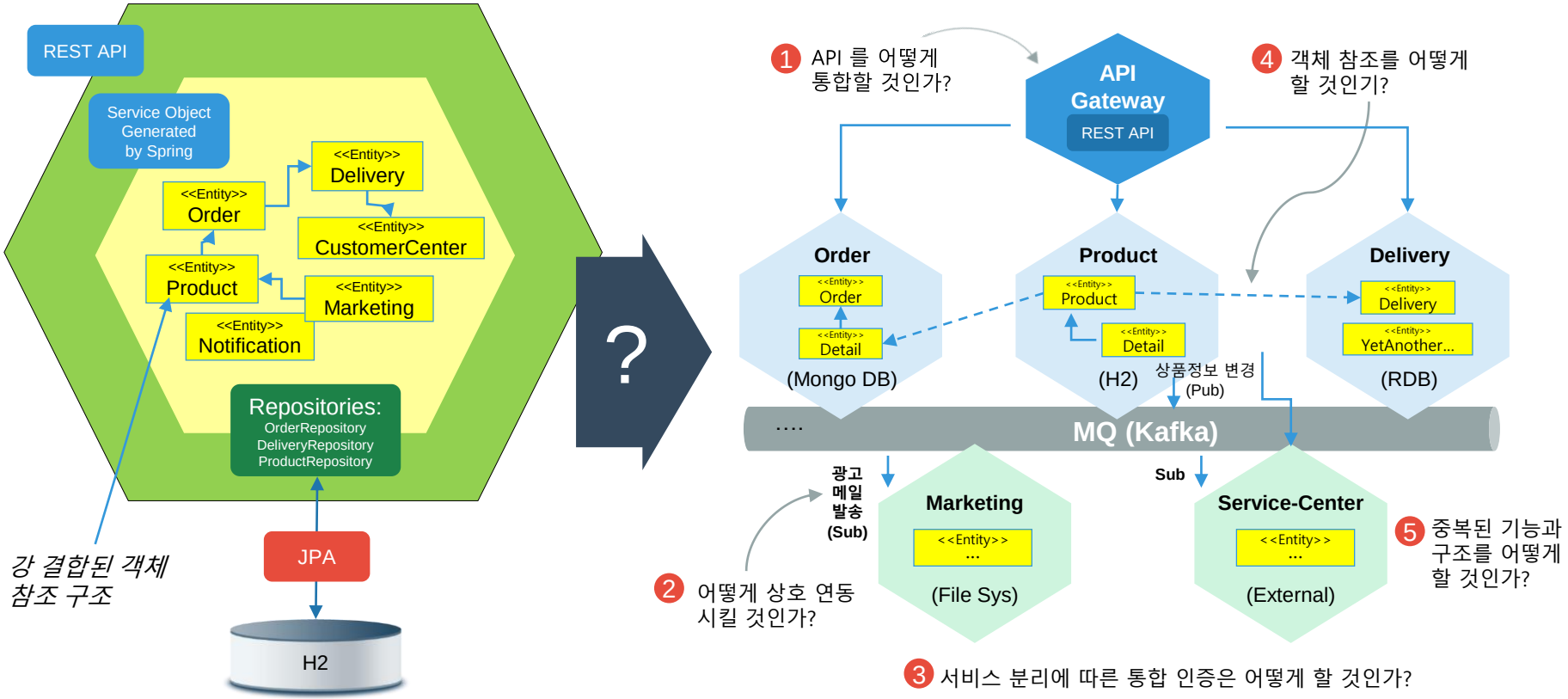
마이크로 서비스간의 서열과 역학관계

"손님이 많이 와서 집이 좁아졌다.. 무엇을 우선적으로 줄일 것인가?"



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

Issues in transforming Monolith to Microservices



1. API 를 어떻게 통합할 것인가?

- API Gateway

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

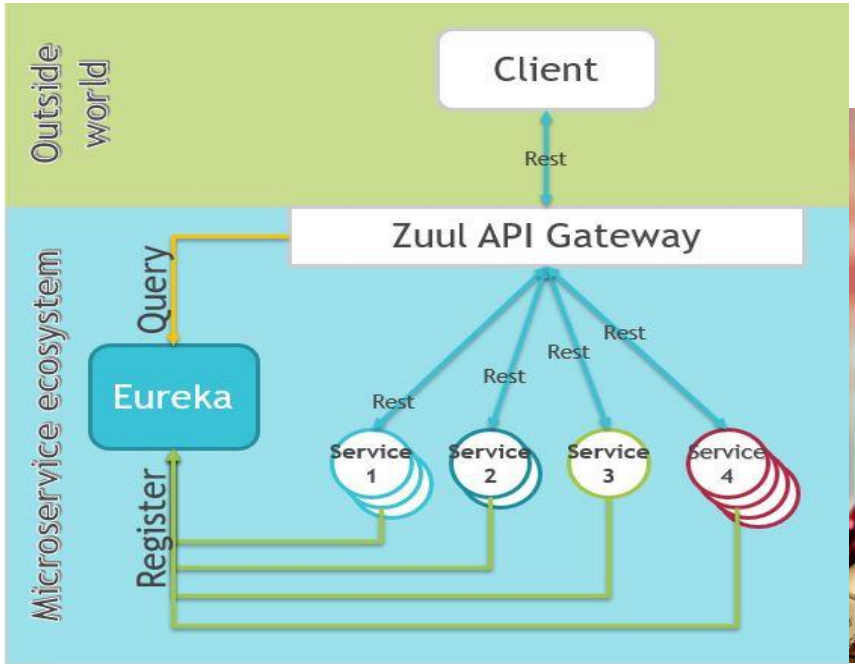
- Service Registry

  → API Gateway 가 클러스터 내의 인스턴스를 찾아가는 맵

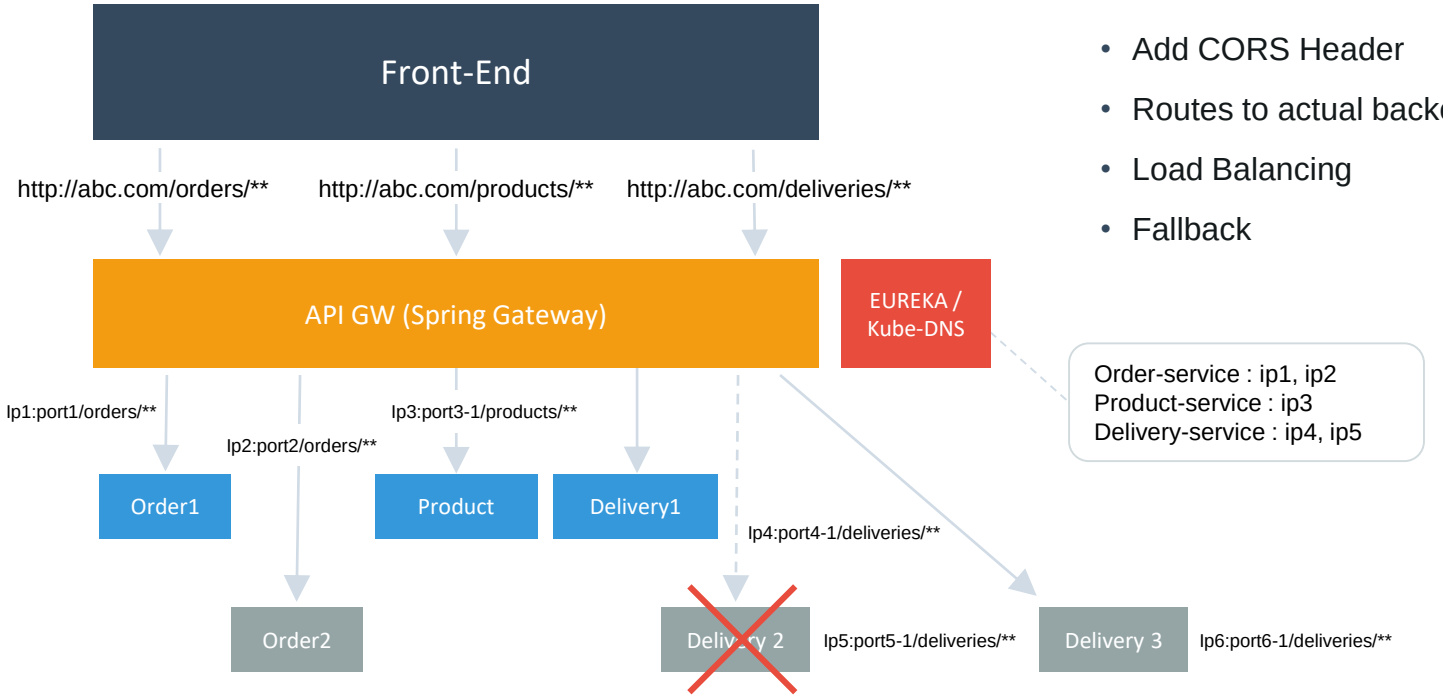
API Gateway : Edge Service

Acting like “Skin” to access our services :

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



API Gateway : Routing, Securing and Load-balancing



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

By container orchestrator something

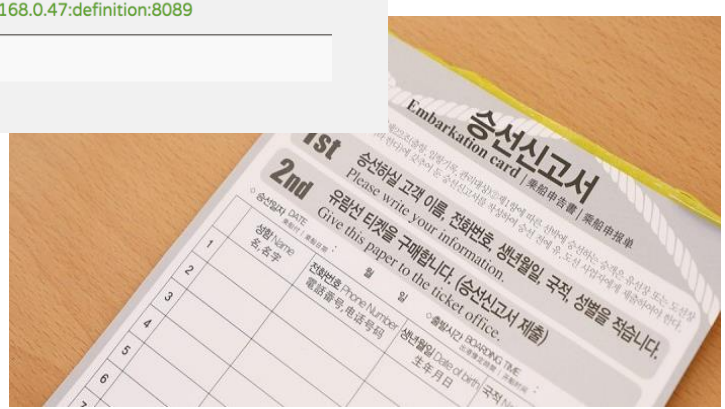
Service Registry: EUREKA or Kube-DNS

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



Configuring Spring Gateway

#application.yml

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


#application.yml

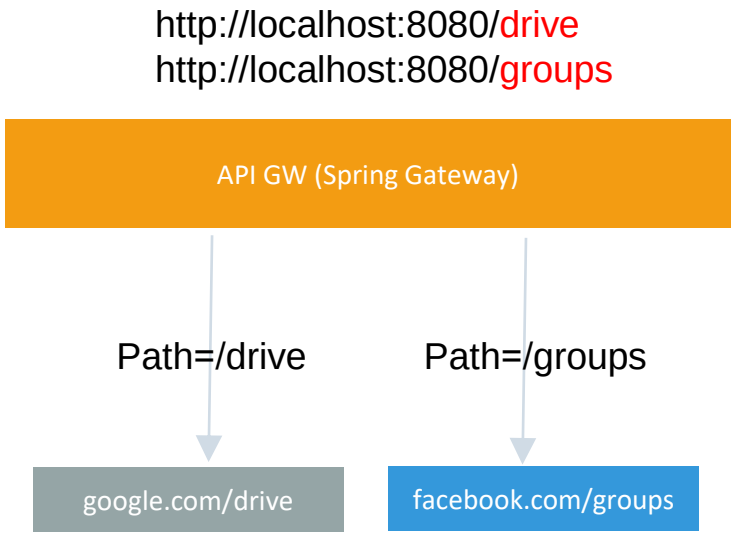
```
spring:
  cloud:
    gateway:
      routes:
        - id: product
          uri: http://legacy:8080
          predicates:
            - Path=/product/**
        - id: order
          uri: http://order:8080
          predicates:
            - Path=/order/**
        - id: delivery
          uri: http://delivery:8080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

Lab: API Gateway (1/3)

- https://start.spring.io/ 에서 ‘gateway’ dependencies 로 추가
- https://spring.io/projects/spring-cloud 에서 spring-boot 와 spring-cloud 디펜던시 확인
- mvn spring-boot:run
  - port 8080, netty 서버

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

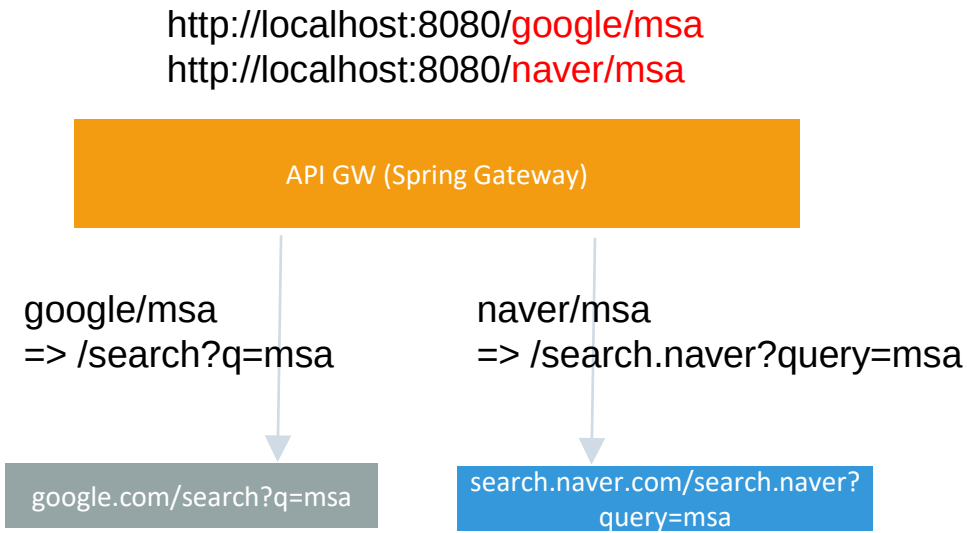
- Route : 게이트웨이의 기본 블록 구성이다. ID, Url , Predicate, Collection of Fillers 로 이루어 져 있다. Predicate 가 일치할 때 Route 가 true 로 인식을 한다
- Predicate : 조건부 – 헤더, 파라미터등의 http 요청을 매치




```yaml
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



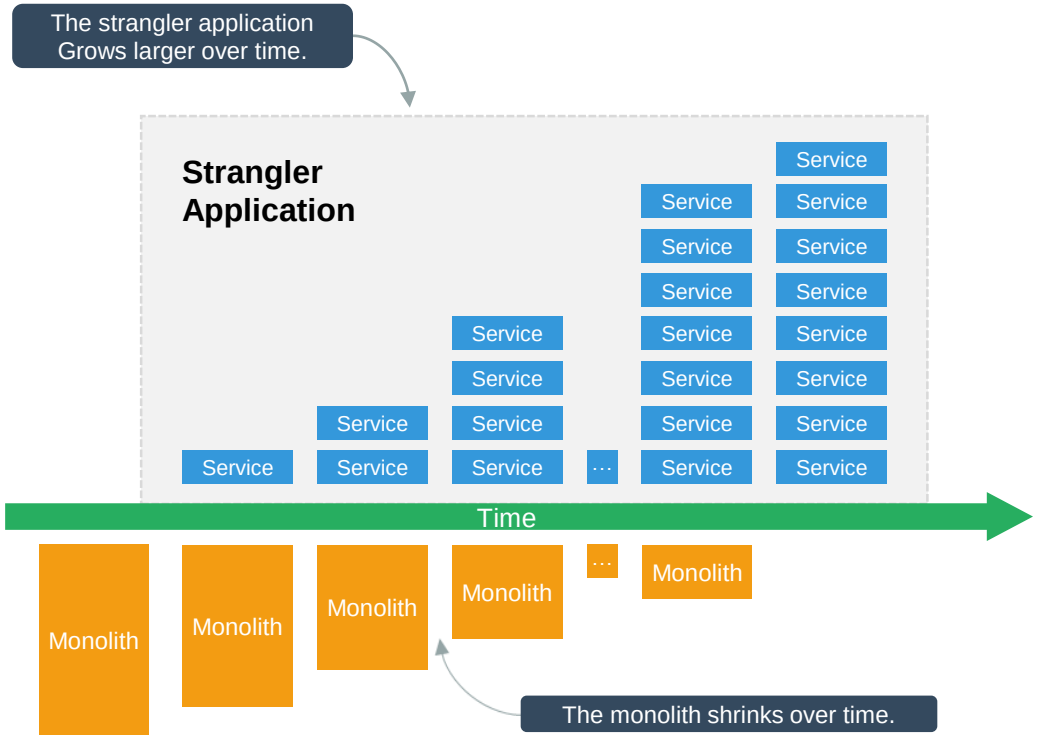

```
spring:
  cloud:
    gateway:
      routes:
      - id: google
        uri: http://google.com
        predicates:
          - Path=/google/{param}
        filters:
          - RewritePath=/google(?<segment>/?.*),/search
          - AddRequestParameter=q,{param}
```

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

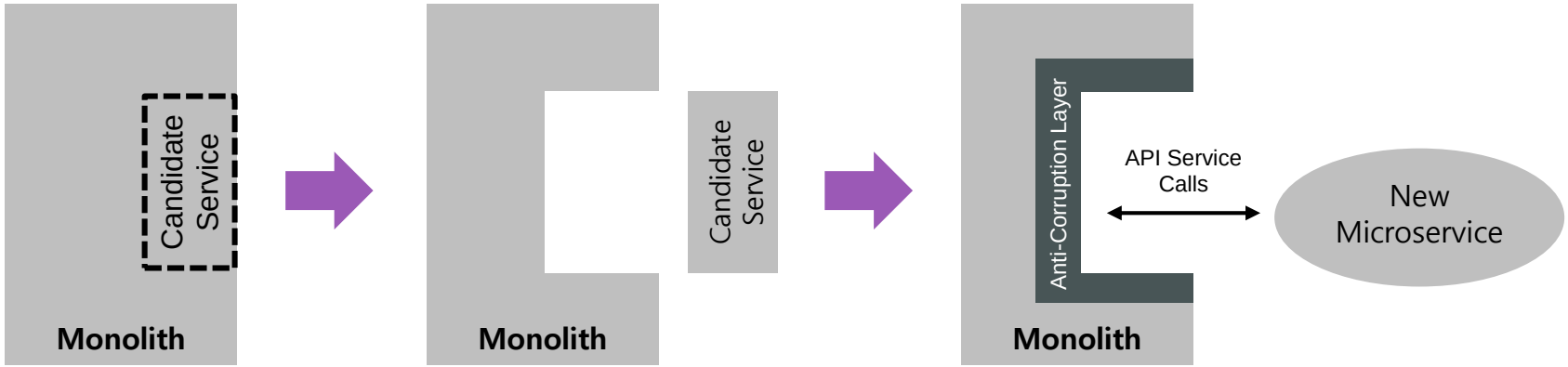
- Event Shunting
  → 레가시에서 이벤트를 퍼블리시하도록 전환, 신규 서비스가 주요 비즈니스 이벤트에 대하여 반응하도록 처리

Legacy Transformation – Strangler Pattern



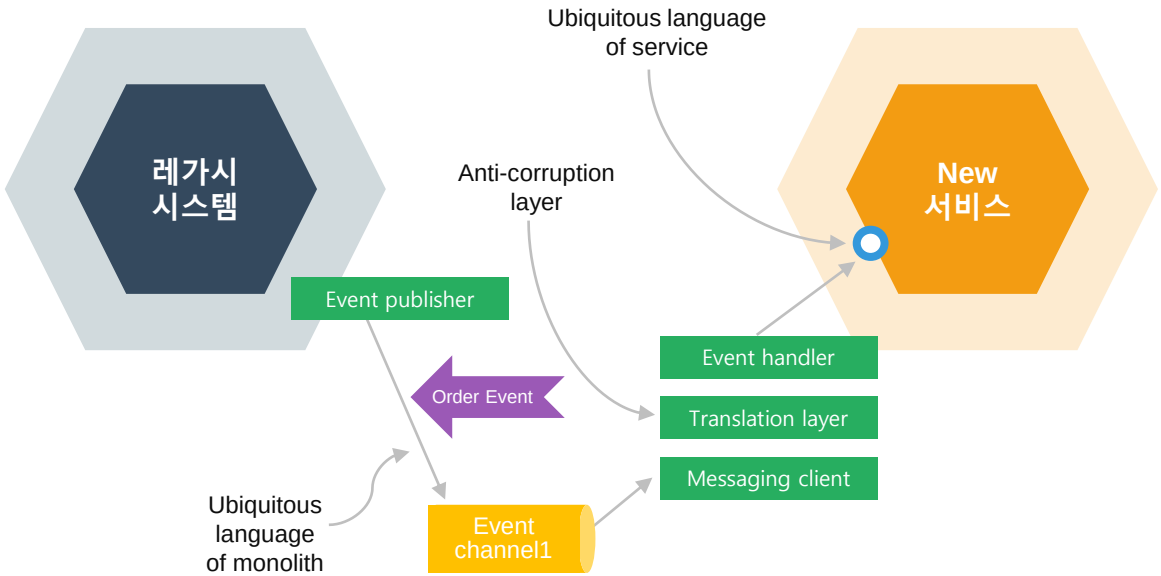
- Strangler 패턴으로 레가시의 모노리스 서비스가 마이크로 서비스로 점진적 대체를 통한 Biz 임팩트 최소화를 통한 구조적 변화
- 기존에서 분리된 서비스 영역이 기존 모노리스와 연동 될 수 있도록 해주는 것이 필요
- 그 방법은 앞서 동기/ 비동기 방식이 채용될 수 있음
- 동기 – 레거시로 하여금 기존 소스코드 수정 요인이 높음, 따라서 이벤트 기반 비동기 연동 (CQRS) 추천

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

Event Shunting



- **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

Lab: Proxy with FeignClient (1)

기존 연결부를 FeignClient 로 만든 Stub 으로 대체

Legacy

```
private void callDeliveryStart(){
    DeliveryService deliveryService = Application.applicationContext.getBean(DeliveryService.class);
    deliveryService.startDelivery(delivery);
}
```

Proxy Object

```
@FeignClient(name ="delivery", url="${api.url.delivery}")
public interface DeliveryService {

    @RequestMapping(method = RequestMethod.POST, value =
"/deliveries", consumes = "application/json")
    void startDelivery(Delivery delivery);

}
```

Local Object

```
public class DeliveryServiceImpl implements DeliveryService{

    public void startDelivery(Delivery delivery){
        deliveryRepository.save(delivery);
    }
}
```

기존 하나로 묶여있던 delivery 관련 로직 (구현코드들)을 Delivery Service 로 옮김

Delivery Service

```
public class DeliveryServiceImpl implements DeliveryService{

    public void startDelivery(Delivery delivery){
        deliveryRepository.save(delivery);
    }
}
```

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?

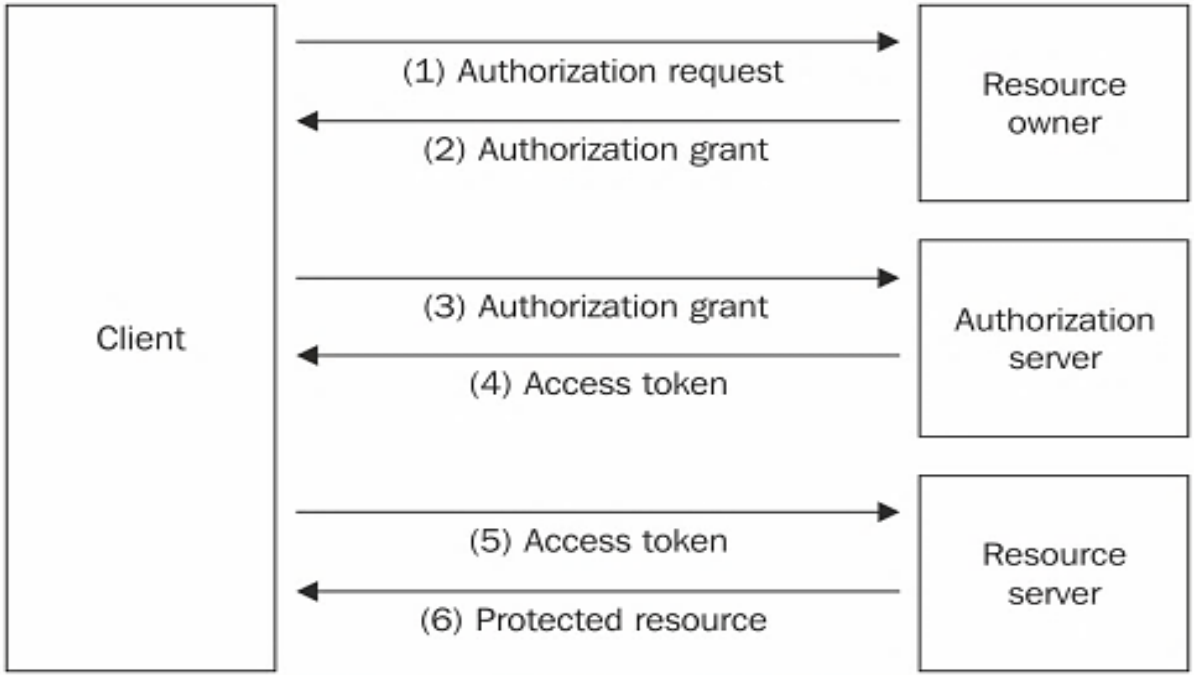
- **OAuth2.0**

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

- **JWT(Json Web Token) 토큰**

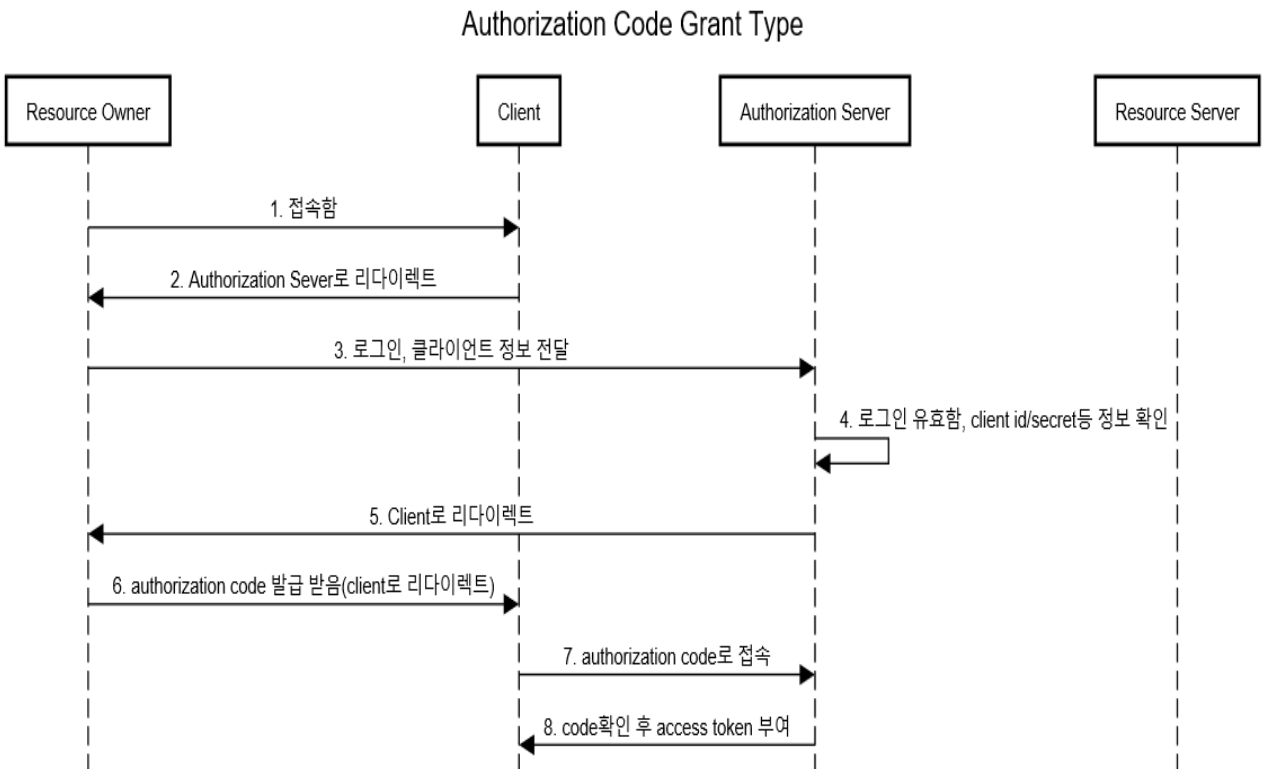
  → Header, Claim Set, Signature로 구성
  → 요청 헤더에 Authorization 값을 담아서 서버로 송신

OAuth 2.0: Authorization Flow



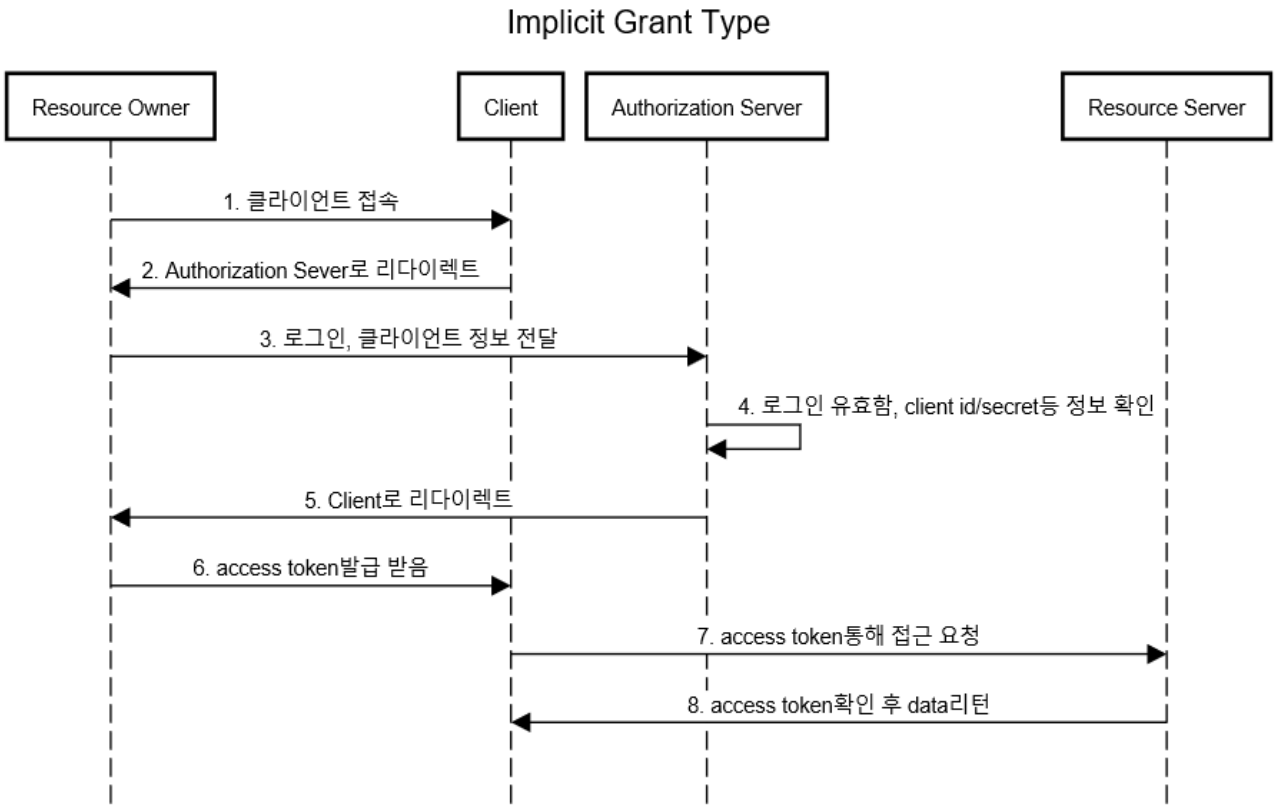
1. 클라이언트가 자원 소유자에게 권한 요청
2. 자원 소유자가 권한을 허가시, 클라이언트는 권한 증서를 발급받음
3. 클라이언트는 권한증서를 가지고 토큰을 권한 서버에 요청
4. 권한증서의 유효성을 체크하고 토큰을 발급해줌
5. 클라이언트는 토큰을 사용하여 자원 요청
6. 토큰 유효성 확인후 요청 처리

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



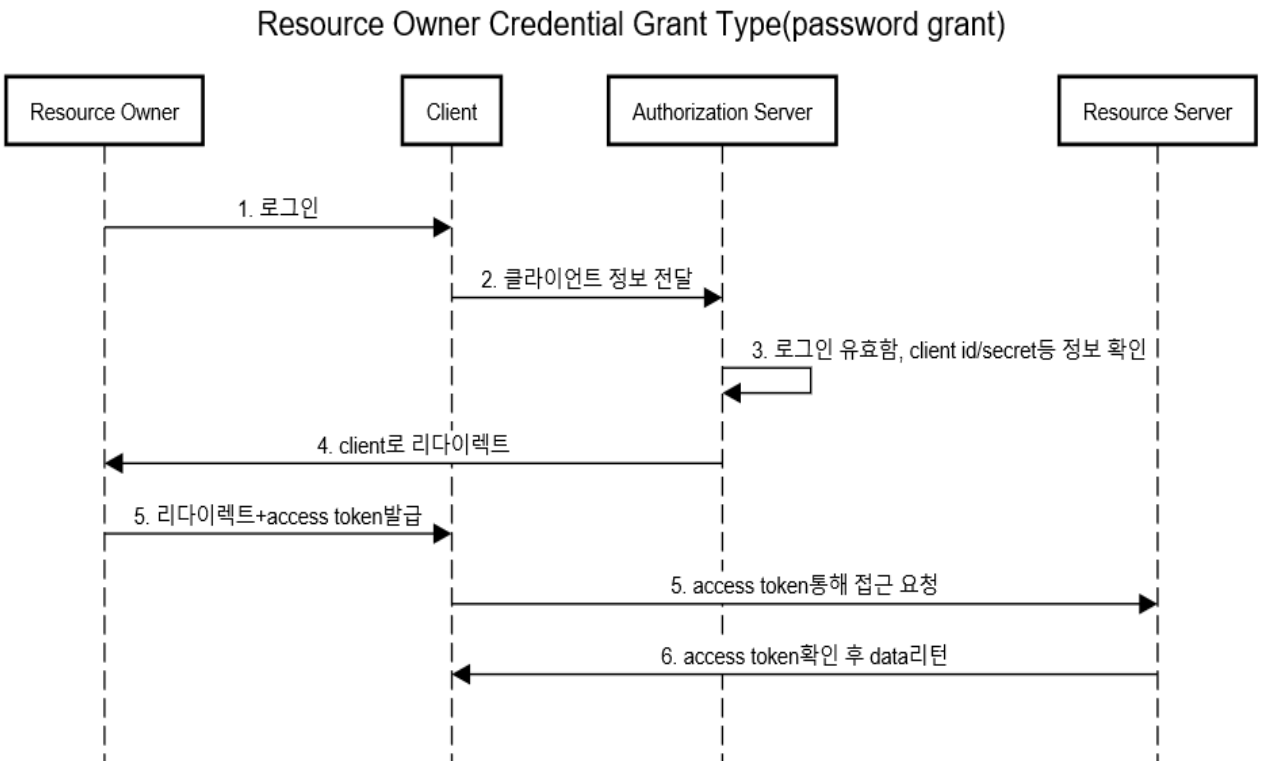
1. 구글,페이스북,카카오등 유저 정보가 다른 시스템에 있을때 사용하는 방식
2. 어플리케이션이 인증서버에 요청해 브라우저를 열어서 사용자가 인증을 진행하게 하는 방식으로 사용
3. 토큰요청시 코드를 요청하는 단계가 있어서 보안에 효과적
4. 가장 복잡하지만, 가장 많이 쓰임 > 개발자만 고생하면됨

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



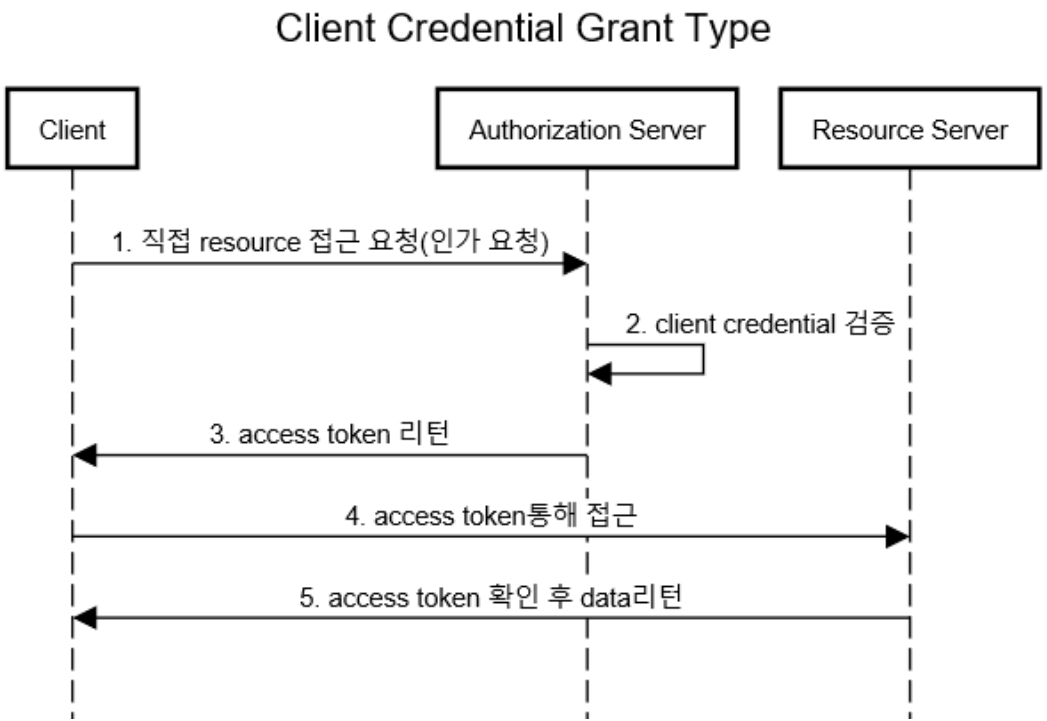
1. Authorization_code 방식에서 코드를 요청하는 프로세스를 제거하고, 바로 토큰을 return
2. Javascript 로 동작하는 SPA(Single Page Application)에서 사용하기 위해 만들어 졌으나 권장하지 않음
3. 신뢰성 있는 앱 또는 단말기에서 사용
4. 외부에 있는 Oauth 서버가 cors 를 지원하지 않을때 사용

OAuth 2.0: Resource Owner Credential (3/4)



1. 타사의 인증 서버를 사용하지 않고, 자신의 서비스에 인증시 사용 (자신이 유저정보를 가지고있고, 내 서비스만 인증할때)
2. Oauth 2.0 의 가장 간단한 인증중 하나
3. 전통적으로 이름과 비밀번호로 인증

OAuth 2.0: Client Credential (4/4)



1. 사용자가 아닌 응용프로그램 (client) 이 인증을 요청할때 사용
   (Resource Owner = Client)
2. 접근 권한이 한정되어있는 프로그램 사용시 활용
3. 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)

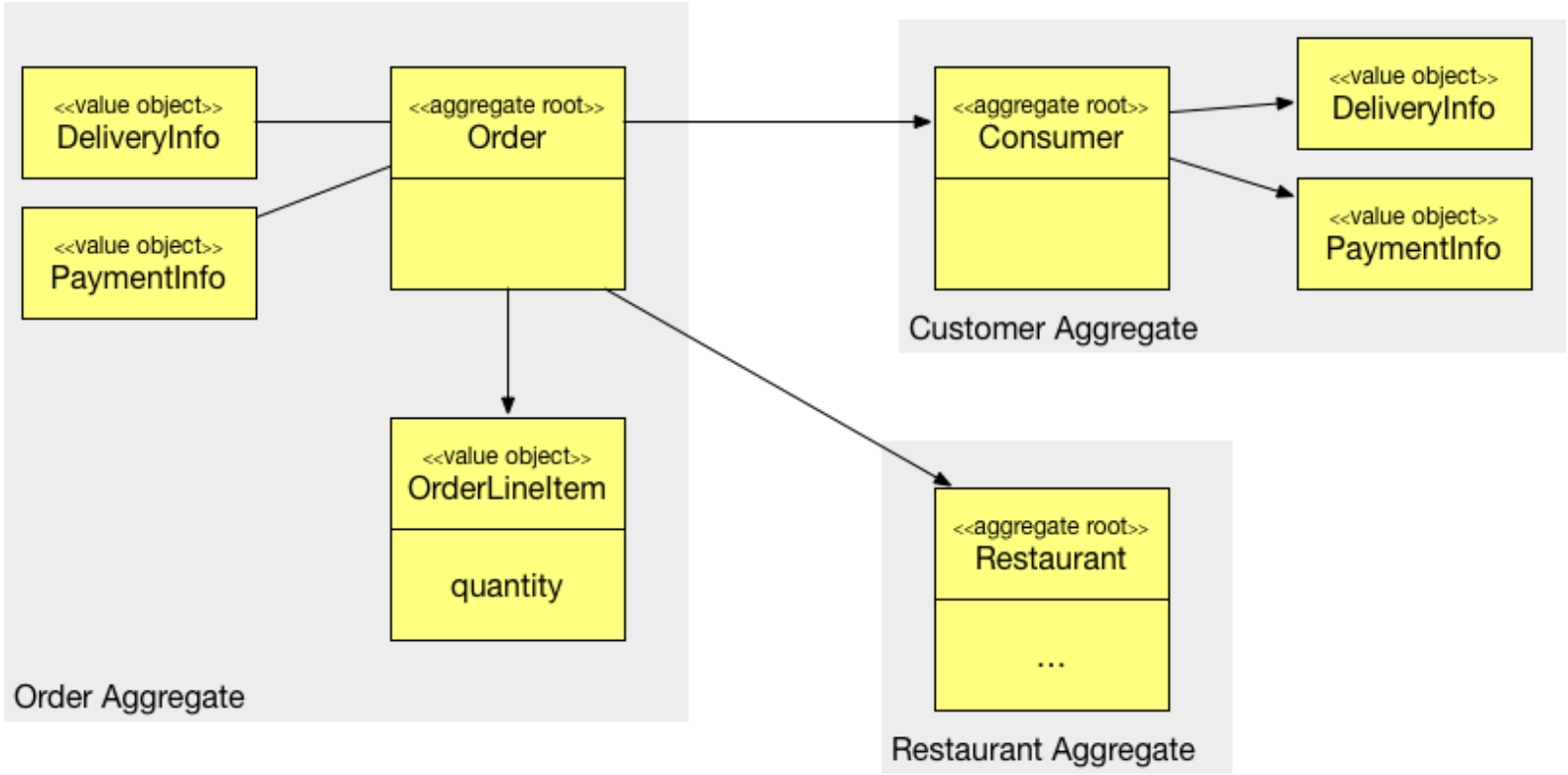
- 인증 서버에 토큰 요청- password grant
  - http --form POST localhost:8090/oauth/token "Authorization: Basic dWVuZ2luZS1jbGllbnQ6dWVuZ2luZS1zZWNyZXQ=" grant_type=password username=1@uengine.org password=1
- 인증 서버에 토큰 요청- client_credentials grant
  - http --form POST localhost:8090/oauth/token "Authorization: Basic dWVuZ2luZS1jbGllbnQ6dWVuZ2luZS1zZWNyZXQ=" grant_type=client_credentials

4. 객체 참조를 어떻게 할 것인가?

- 직접적 메모리 기반 객체 참조는 불가능

- 분리된 Aggregate 내부의 Entity 간에는 Key 값으로만 연결
  → Primary Key 를 통해서만 참조
  → HATEOAS link 를 이용

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.
}
```

5. 중복된 기능과 데이터를 어떻게 할 것인가?

- 재사용하지 않고 중복 구현하는 것이 MSA 스러운 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

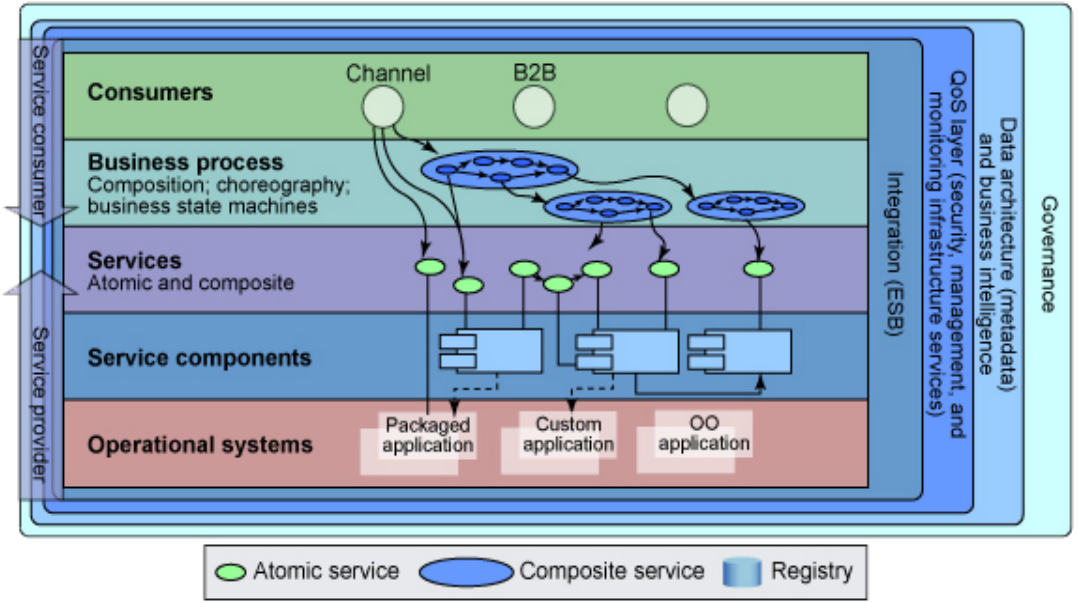
- 예외상황 : 데이터 참조에 intensive 한 서비스 (인증정보 등)는 Utility 서비스 성격으로 구현 가능함

Table of content

Microservice and
Event-storming-Based
DevOps Project

UEngine

Integration Patterns



By UI

By Request-Response

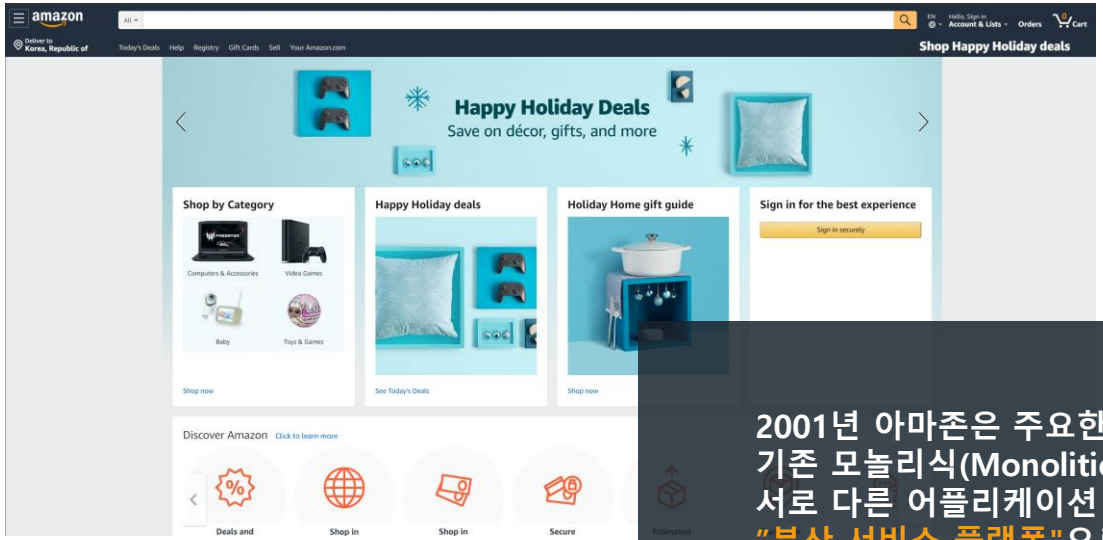
By Event-driven Architecture

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.



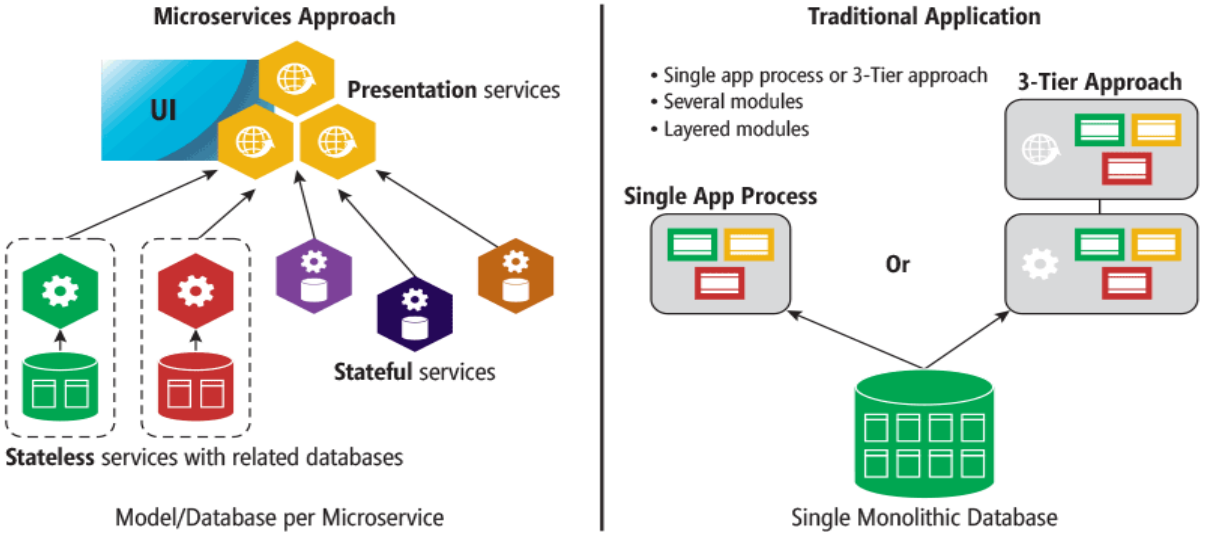
amazon

2001년 아마존은 주요한 아키텍쳐 변화가 있었는데
기존 모놀리식(Monolitic) 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 서비스를 호출하여 만들고 있습니다.

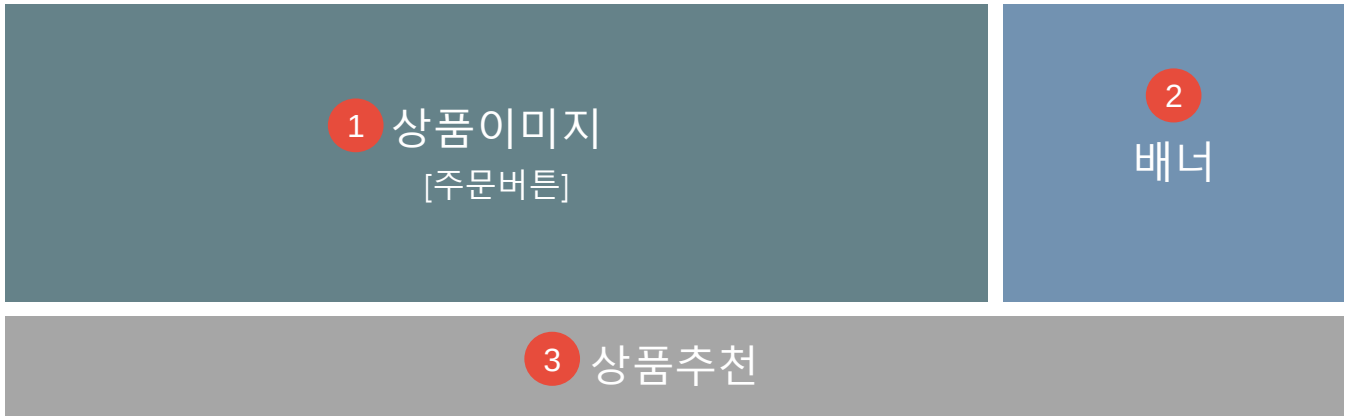
Microservice Integration with by UI

- 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



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.

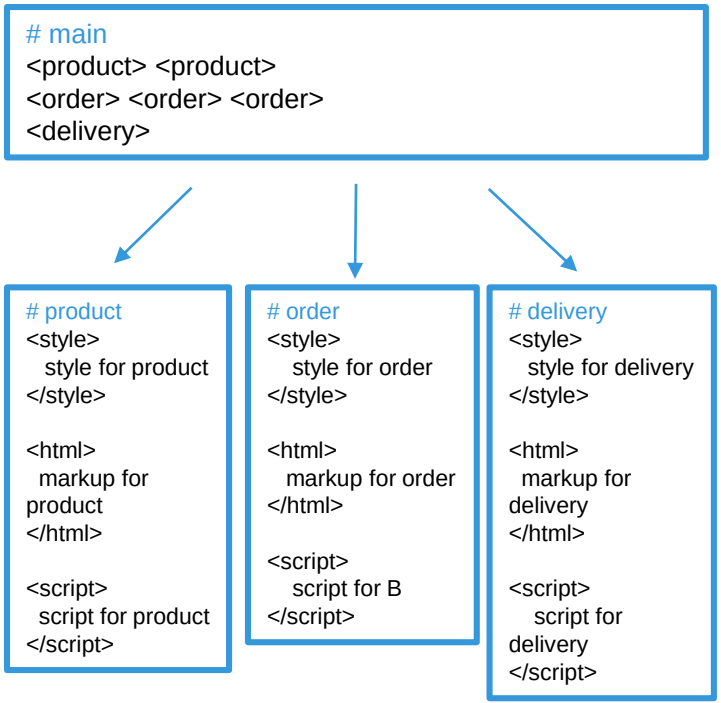
광고배너가 나가지 못한다고 주문을 안받을 것인가?

그걸 원하지 않는다면 **AJAX / MVVM 같은 Client-side Rendering 기술에 주목하자.**

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>
```



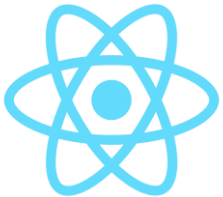
Almost all the frontend frameworks support Web Components

Polymer

VueJS

AngularJS

ember



React

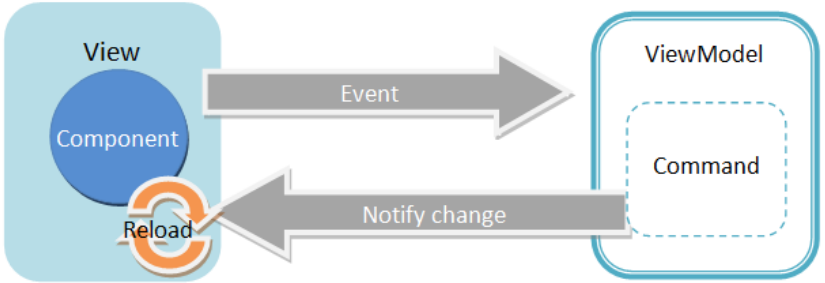
Custom tag for Domain Class: <product> tag

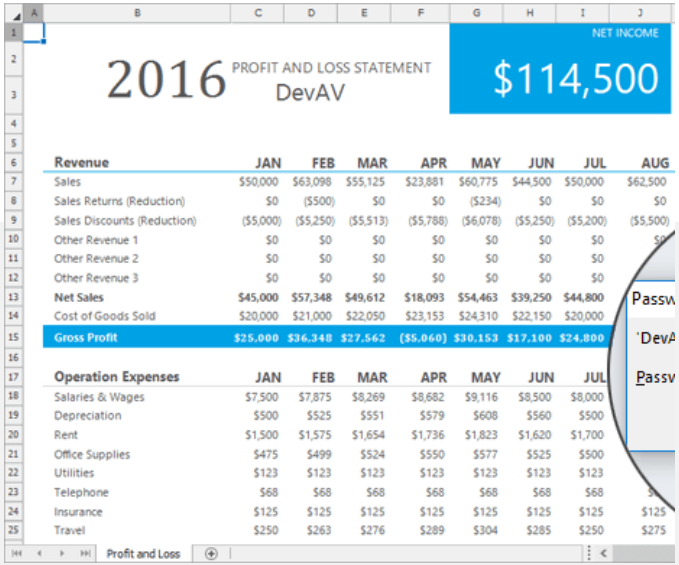
```
<table>
   <tr
         v-for="(item, index) in props.items"
         :key="item.id"
   >
      <product
            v-model="props.items[index]"
            @inputBuy="showBuy"
            @inputEdit="showEdit"
      ></product>

   </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

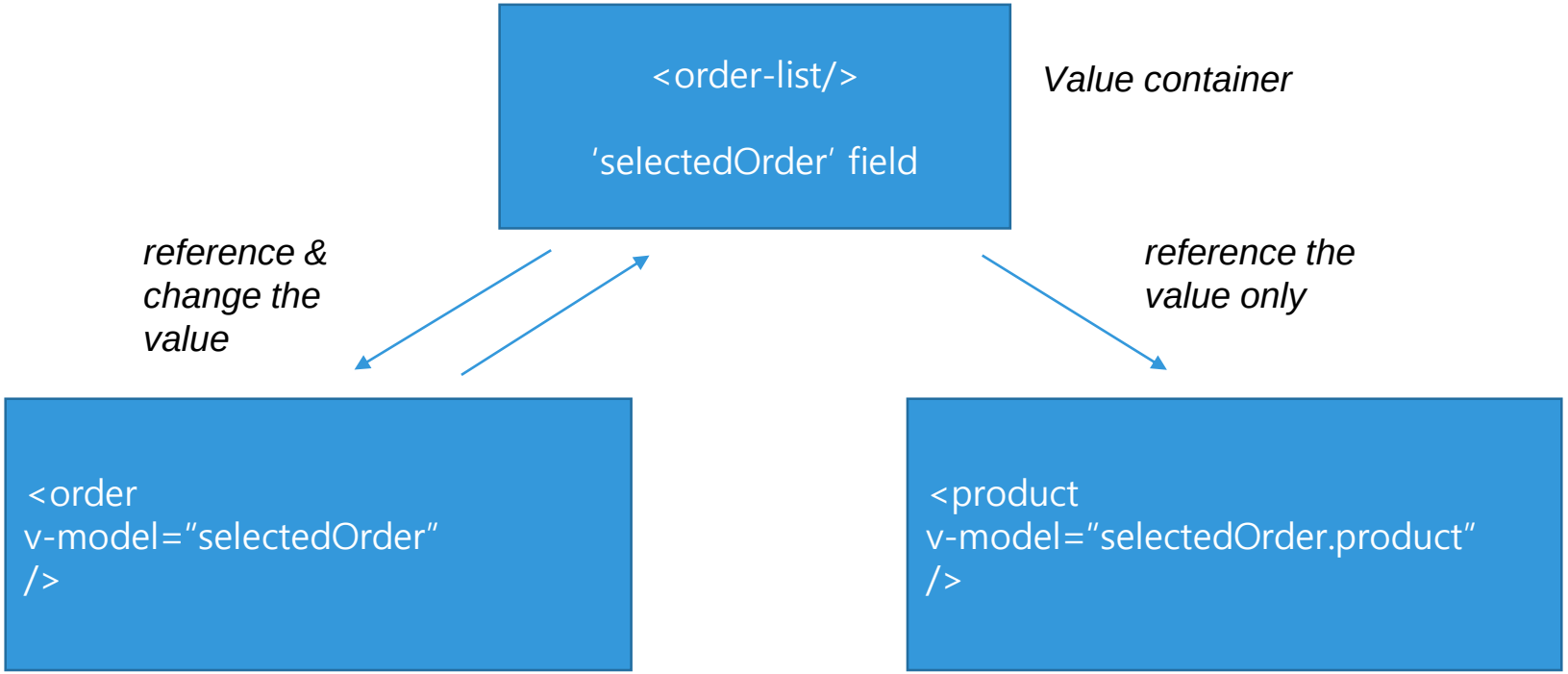
NET INCOME $114,500

Revenue	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG
Sales	$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	$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

Interaction between Domain Tags



Lab time: Shopping Mall UI 만들기

기능
- 상품 카탈로그
- 검색
- 주문

비기능
- SPA (Single Page App)
- Responsive Web
- Front-end Data Aggregation (AJAX)

159

Why Vue.js

Angular and React are not a STANDARD

While Polymer is heavy and needs Polyfill

1. Fast (compared with React and Angular)
2. Light (low dependencies)
3. Easy migration (easy to learn and easy to embrace existing libraries such as j Query or libraries)

Data binding

```
<div id="demo">
  <p>{{message}}</p>
  <input v-model="message">
</div>
```

```
<a v-bind:href="url"></a>
```

Model-UI binding

v-model

```
<span>Message is: {{ message }}</span>
<br>
<input type="text" v-model="message" placeholder="edit me">
```

Conditional Element

v-if：conditional element

v-else

```
<h1 v-if="ok">Yes</h1>
<h1 v-else>No</h1>
```

Loop for collection

v-for

```
<ul id="example-1">
 <li v-for="item in items">
  {{ item.message }}
 </li>
</ul>
```

```
var example1 = new Vue({
 el: '#example-1',
 data: {
  items: [
   { message: 'Foo' },
   { message: 'Bar' }
  ]
 }
})
```

Event Handling

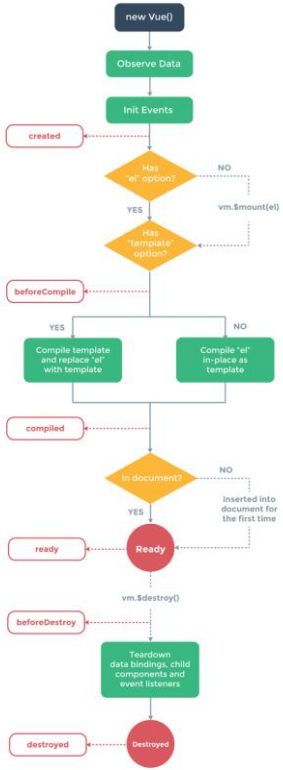
v-on

```
<div id="example">
 <button v-on:click="greet">Greet</button>
</div>
```

```
var vm = new Vue({
 el: '#example',
 data: {
  name: 'Vue.js'
 },
 // 在 `methods` 对象中定义方法
 methods: {
  greet: function (event) {
   // 方法内 `this` 指向 vm
   alert('Hello ' + this.name + '!')
   // `event` 是原生 DOM 事件
   alert(event.target.tagName)
  }
 }
})

// 也可以在 JavaScript 代码中调用方法
vm.greet() // -> 'Hello Vue.js!'
```

Life cycle

- init
- created
- beforeCompile
- compiled
- ready
- attached
- detached
- beforeDestroy
- destroyed



User defined Component (Your own HTML Tag)

```
Vue.component('child', {
 // public attributes, properties
 props: ['msg'],
 template: '<span>{{ msg }}</span>’
 data: function(){// private attributes, properties
   return {a:’aaa’, b:’bbb’}
 },
 // behaviors
 methods:{
 }
})
```

```
<child msg="hello!"></child>
```

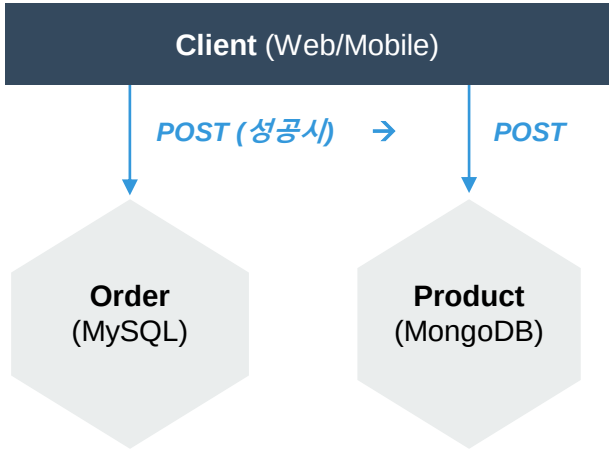
One File Component - .vue

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
module.exports =  {
  data: function () {
    return {
          msg: 'Hello World!'
    }
  }
}
</script>

<style scoped>
h1 {
  color: #42b983;
}
</style>
```

Transaction Problem in data mutation by UI composition

In-consistent Consistent

- 서비스구성 (클라이언트가 순차 호출)



- 조회화면

주문량 : 0 , 재고량 : 10

주문량 : 1 , 재고량 : 10

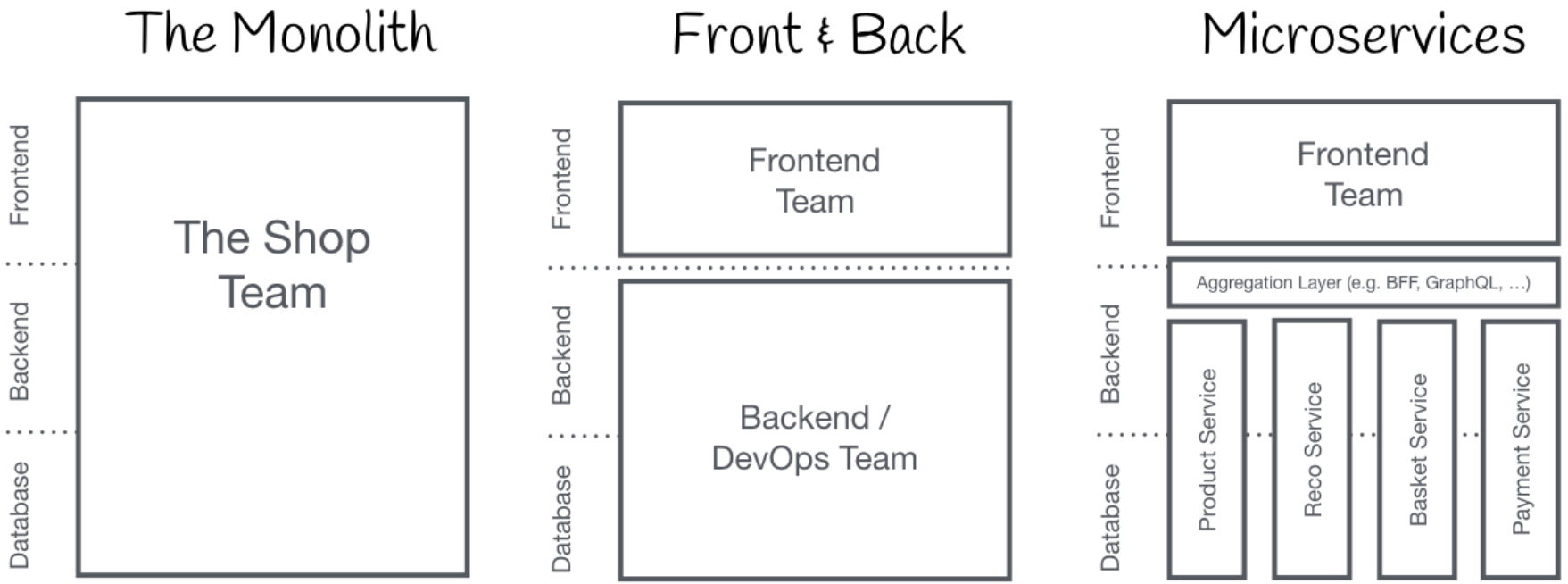
주문량 : 2 , 재고량 : 9

주문량 : 3 , 재고량 : 8

둘다 순조롭게 호출 성공시 일치

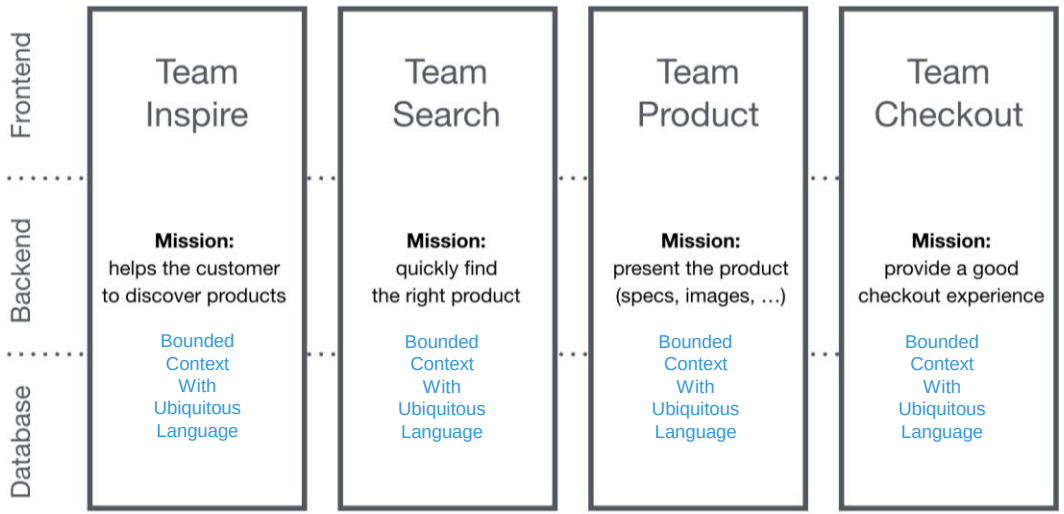
불일치 영원히 생길 수 있음!!

Monolith-Frontend



https://micro-frontends.org/

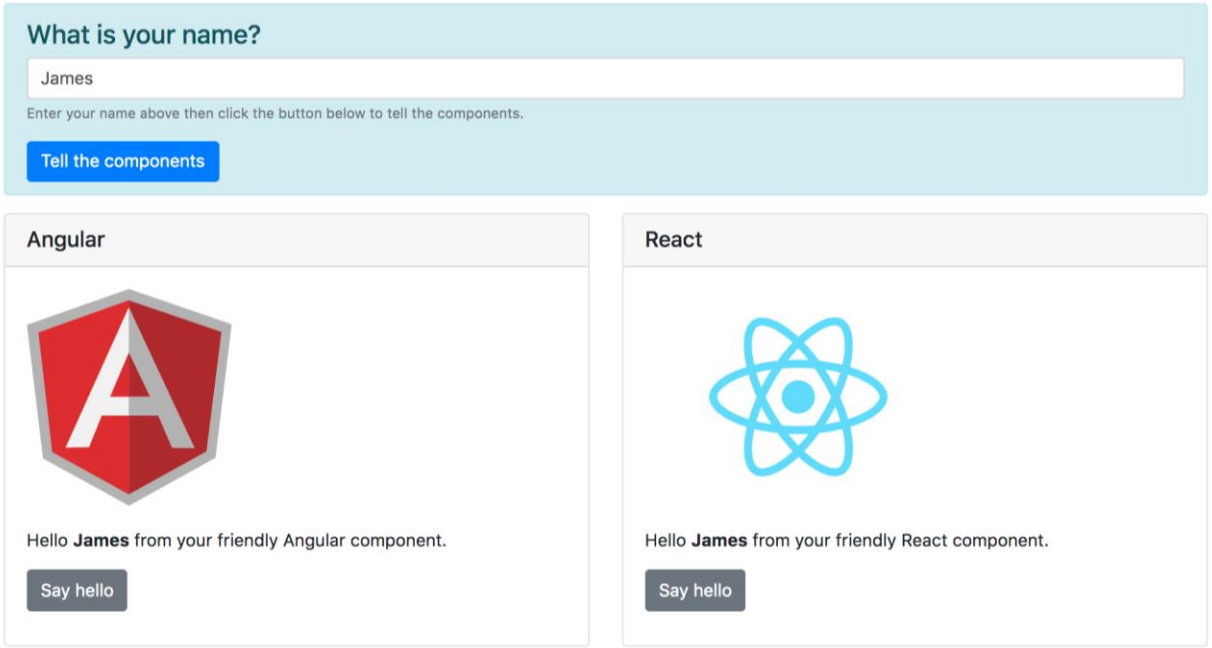
Micro-Frontends



- 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



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

Table of content

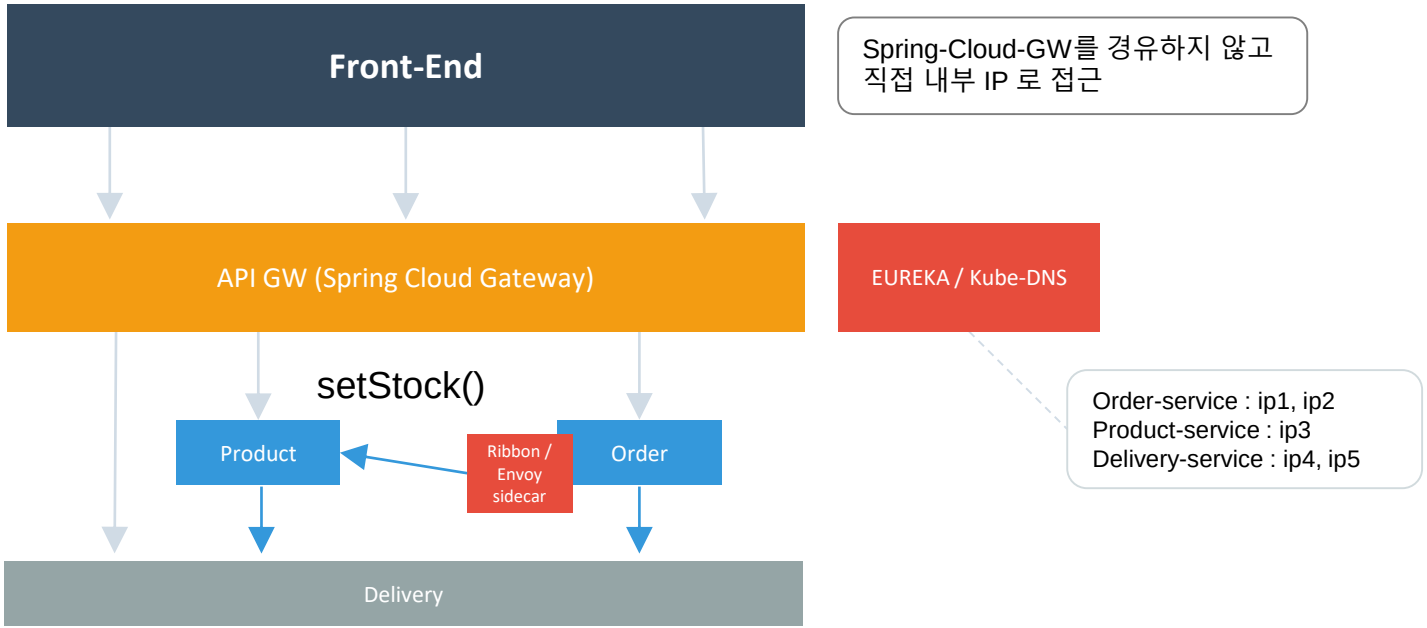
Microservice and
Event-storming-Based
DevOps Project

UEngine

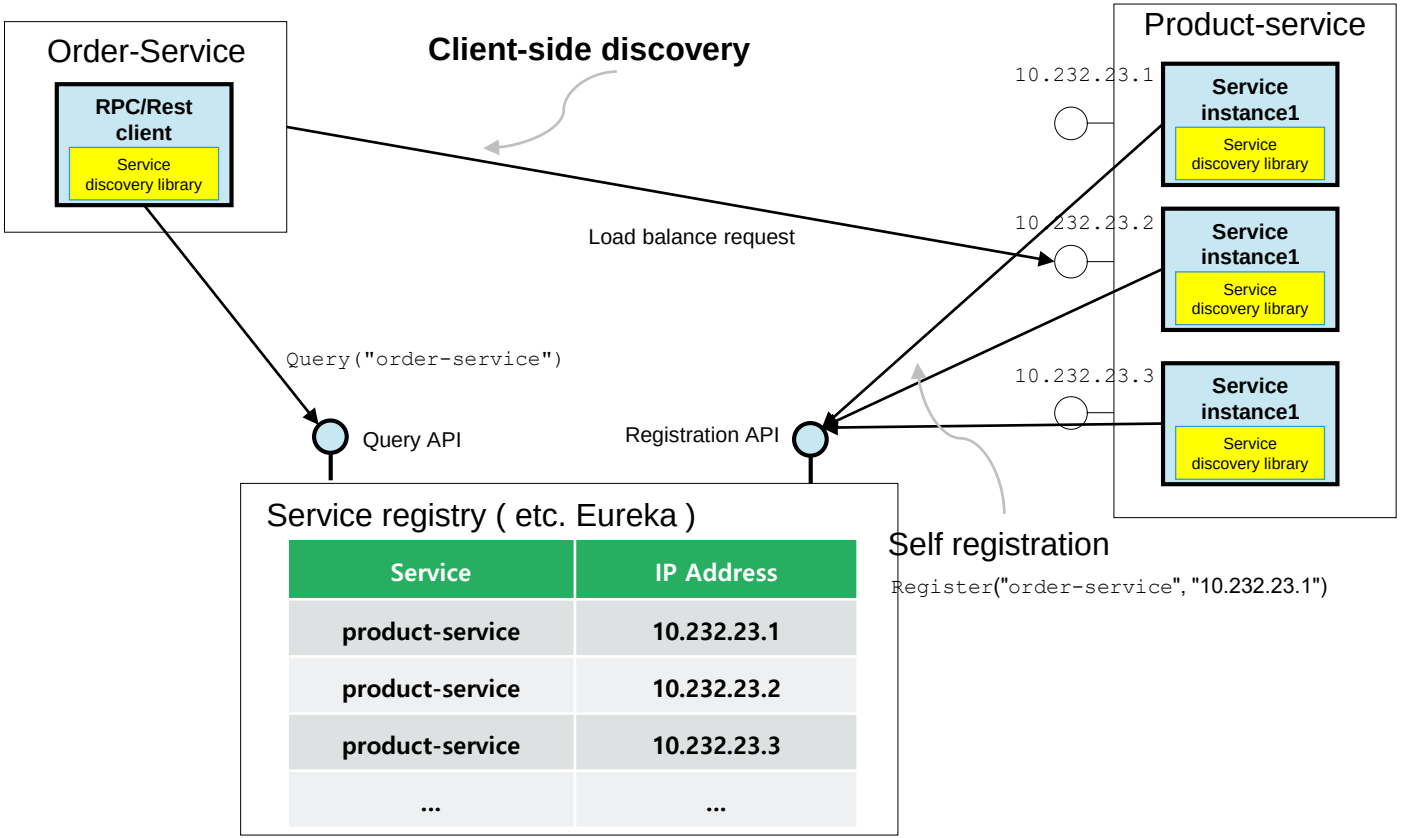
Service Integration by Request-Response

- 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

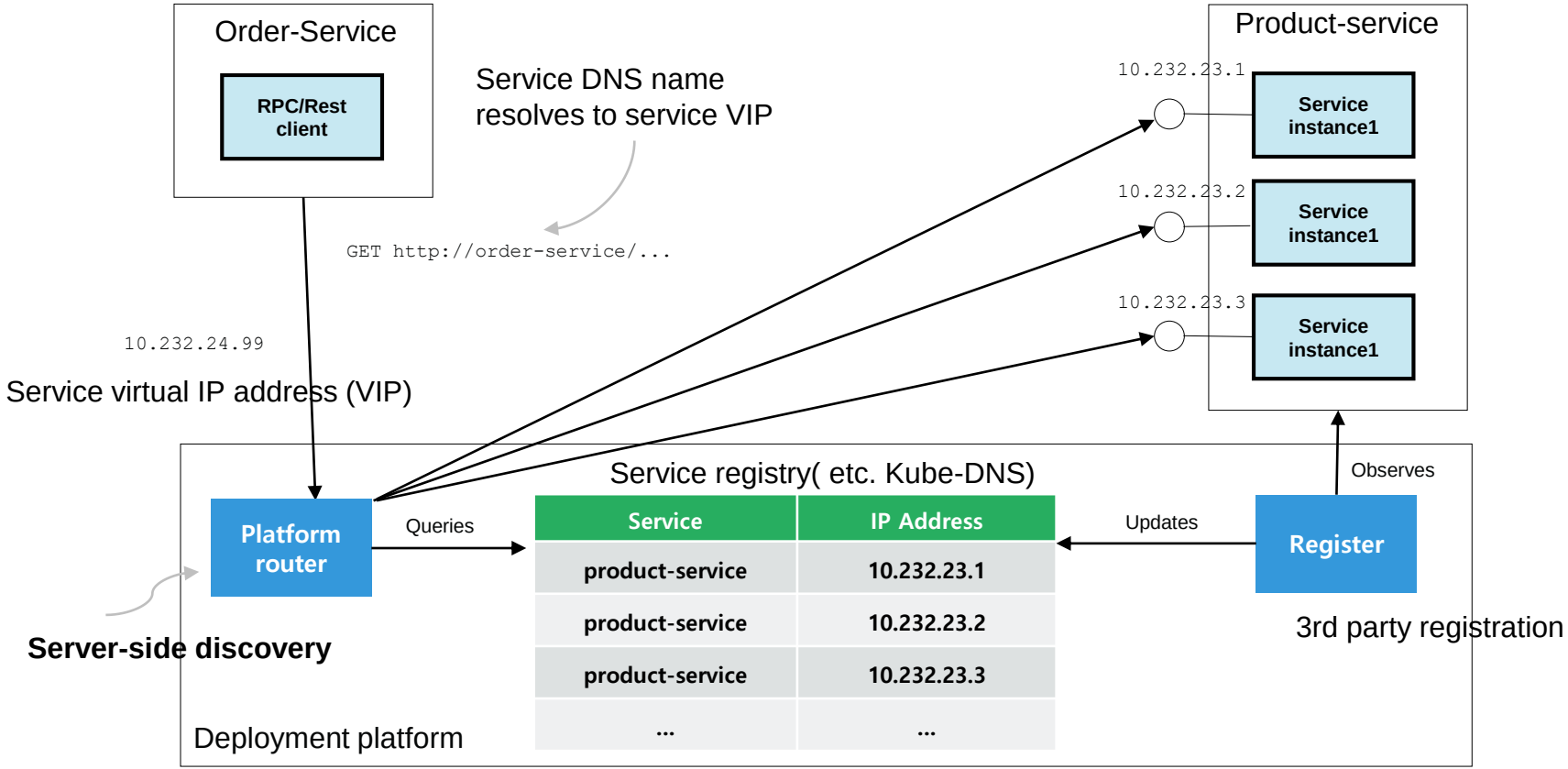
Inter-microservices Call – Request/Response



Service Discovery - APPLICATION-LEVEL



Service Discovery - PLATFORM-PROVIDED

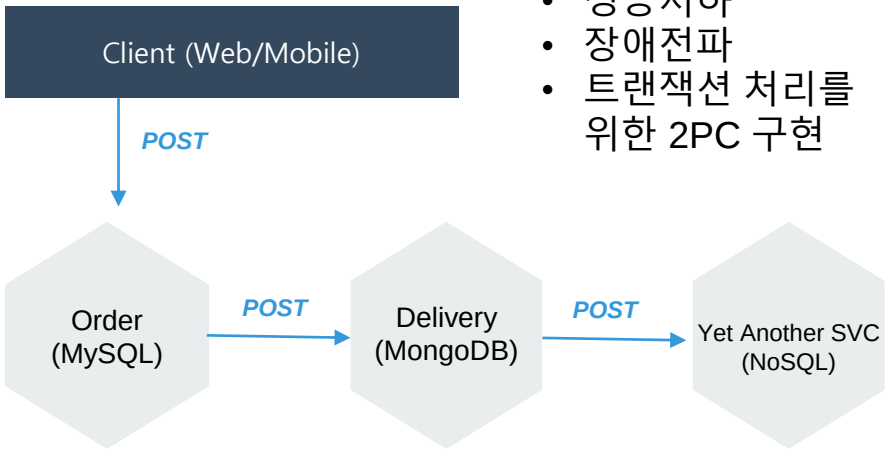


Lab Time: Service Discovery Kube-DNS

- kubectl scale deploy orders --replicas=2
- 각 replica 의 IP 를 메모
  - kubectl get po -o wide
- httpie 를 클러스터 내에서 실행
  - Workflowy 에 httpie 검색 후 설치
  - kubectl exec -it httpie bin/bash
- http http://orders:8080 productId=1 customerId="1@uengine.org" quantity=1
  (memory DB 라서 2개의 instance 중 1개만 주문등록됨)
- 직접 IP 호출
  - http http://[ip1]:8080
  - http http://[ip2]:8080
- 서비스 레지스트리를 통한 Round-robin 호출
  - http http://orders:8080

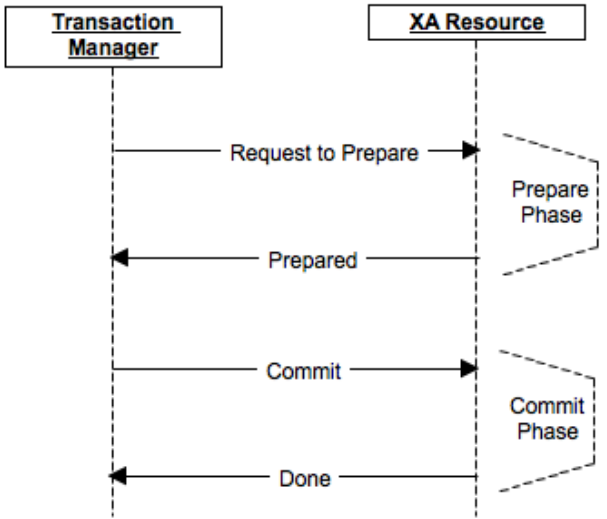
Issues in Request-Response model

서비스구성 (Request-Response, Sync 호출)

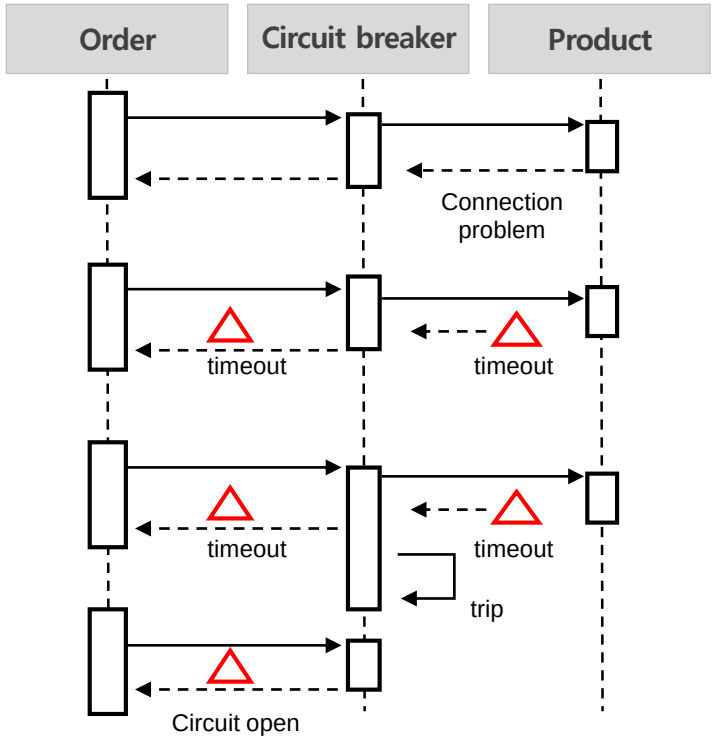


- 성능저하
- 장애전파
- 트랜잭션 처리를 위한 2PC 구현

Atomic Transaction / 2PC



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

- 프로젝트를 클러스터 배포
  - reqres_orders, reqres_products, reqres_delivery

- Install istio , httpie, siege
  - Workflowy “istio 설치” 검색
  - “httpie, siege 툴 설치” 검색

- 각 서비스를 istio 로 deploy
  - kubectl get deploy orders -o yaml > order_deploy.yaml
  - kubectl apply -f <(istioctl kube-inject -f order_deploy.yaml)
  - kubectl get deploy products -o yaml > products_deploy.yaml
  - kubectl apply -f <(istioctl kube-inject -f products_deploy.yaml)
  - kubectl get deploy delivery -o yaml > delivery_deploy.yaml
  - kubectl apply -f <(istioctl kube-inject -f delivery_deploy.yaml)

- 정상적으로 배포 확인
  - Kubectl get po
  - READY 가 2/2 가 되어야함

Lab Time: Circuit breaker using istio

- Httpie pod 접속
  - kubectl exec -it httpie bin/bash
- 상품 주문에 부하를 주기
  - siege -c2 –t10S -v --content-type "application/json" 'http://orders:8080/orders POST {"productId":2,"quantity":1}'
- 혹시 상품 수량이 부족하여 500 에러가 떨어질때
  - http PATCH products:8080/products/2 stock=99999999

Lab Time: Circuit breaker using istio

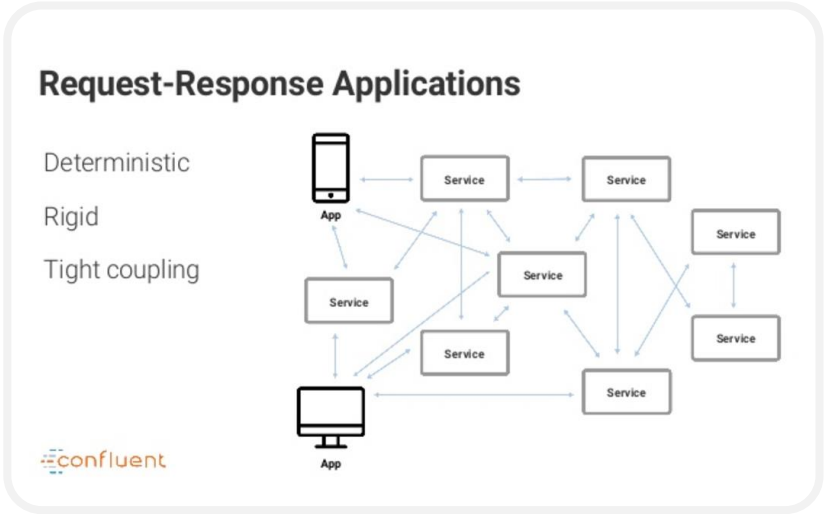
- product 서비스에 서킷 브레이커 적용
  - connectionPool :
    지정된 서비스에 connections를 제한하여 가능 용량 이상의 트래픽 증가에 따른 서비스 Pending 상태를 막도록 *circuit break* 를 작동시키는 방법
  - outlierDetection :
    오류 발생하거나 응답이 없는 인스턴스를 탐지하여 *circuit break* 를 작동시키는 방법
- 부하를 주어서 지정된 커넥션 만큼 차단 확인
  - siege -c2 -t10S -v --content-type "application/json" '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
```

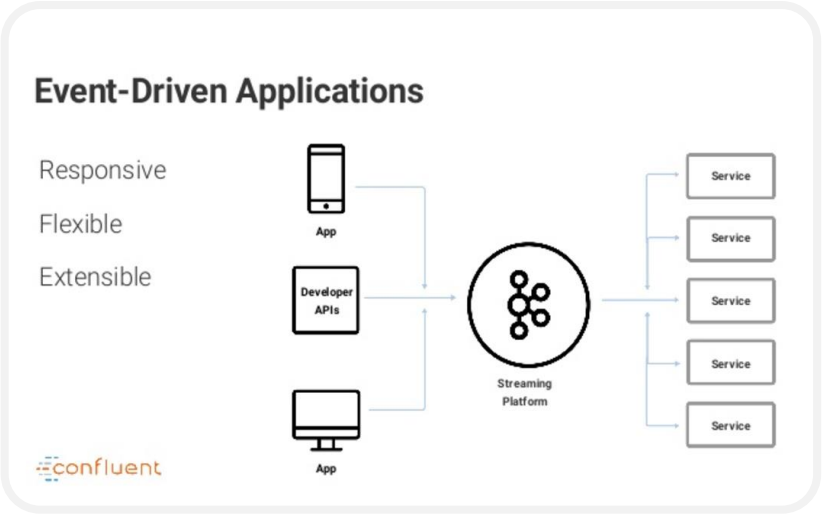
Service Integration by Event-driven Model

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

Microservice Integration with Event-driven Architecture



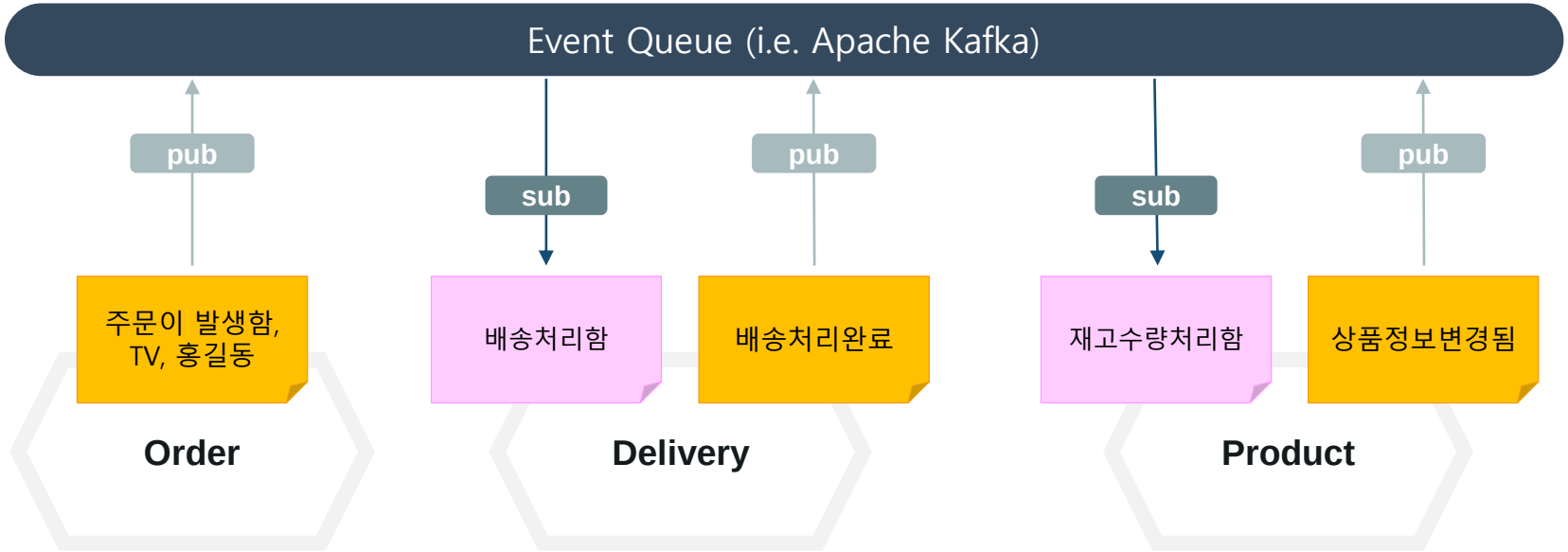
VS



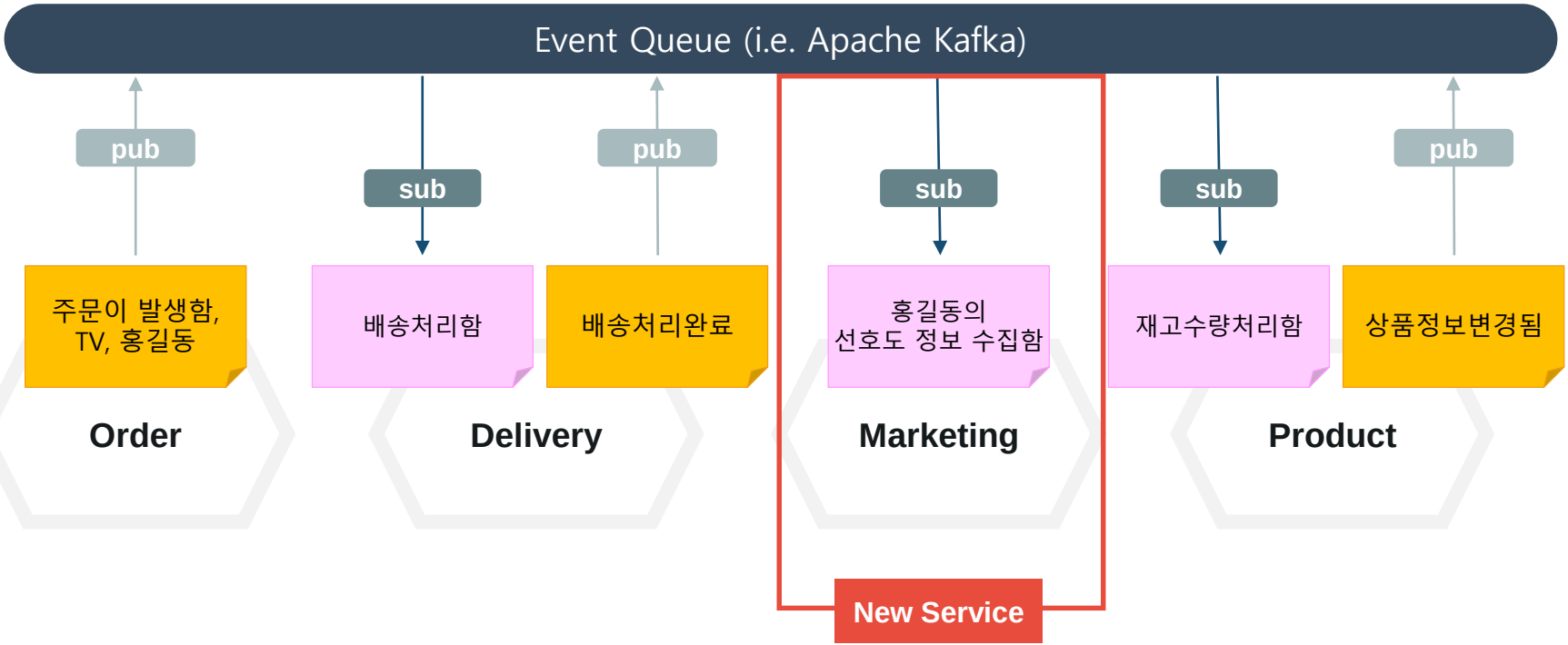
- Point to Point → Spagetti network
- Blocking Model
- System fault spread

- Broadcasting
- Non-Blocking Model
- System fault isolation

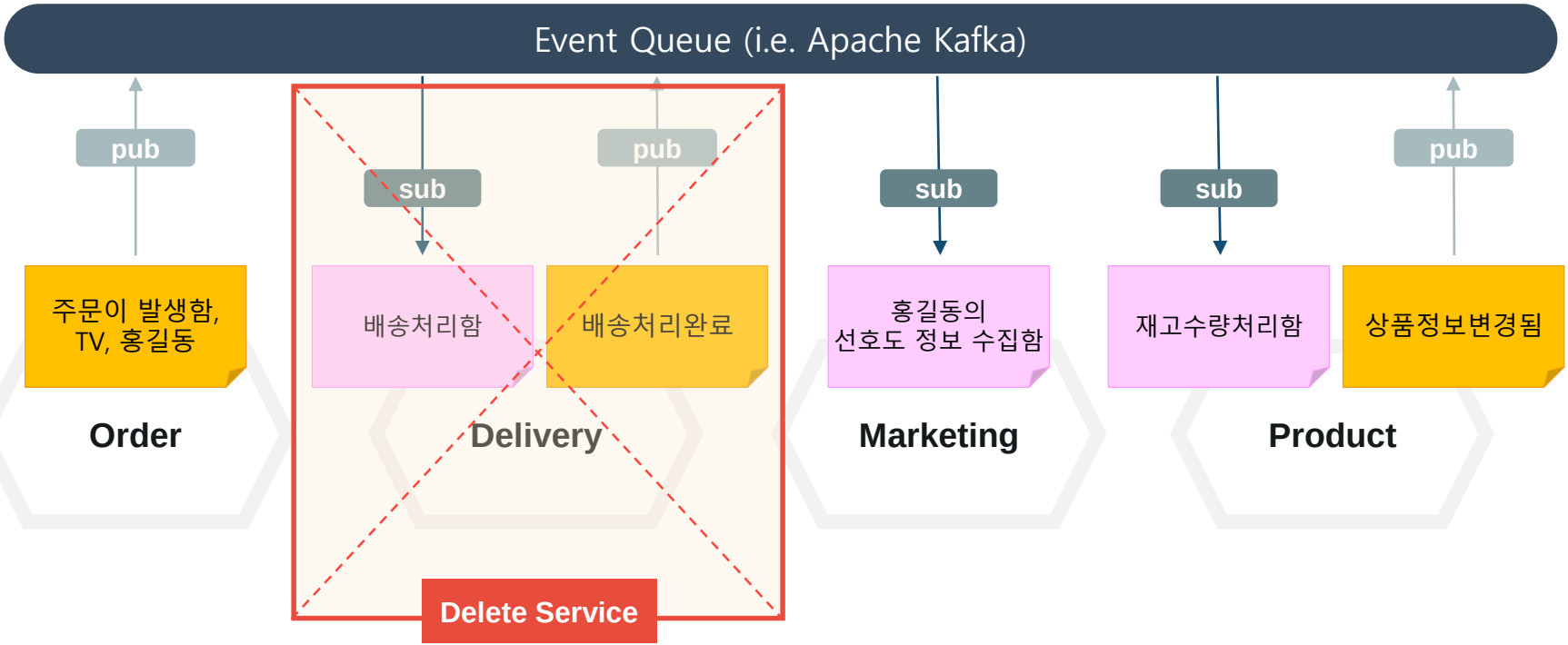
Inter-microservice Call - Event Publish / Subscribe (PubSub)



Adding a new service



Removing a service



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);
            …

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

Event Consumer : Delivery

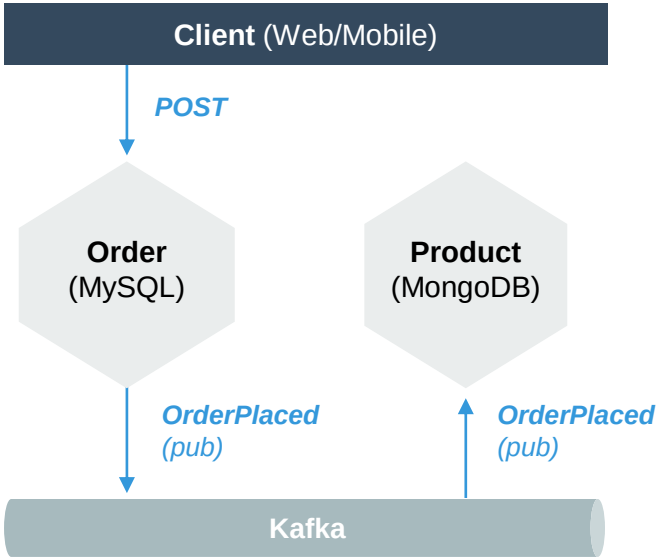
```
@KafkaListener(topics = ”shopping")   // shopping 토픽의 이벤트를 수신
public void onOrderPlaced(…) {   // OrderPlaced 이벤트가 들어오면..

    deliveryService.start(order);  // 해당 주문건의 배송 준비를 시작

    Kafka.send(new OrderShipped(delivery));  // 배송 준비 한 후에는 주문배송처리 됨을 메시지 큐에 신고
}
```

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 TX를 통한 분산 트랜잭션 처리

- 서비스구성 (Eventual TX)



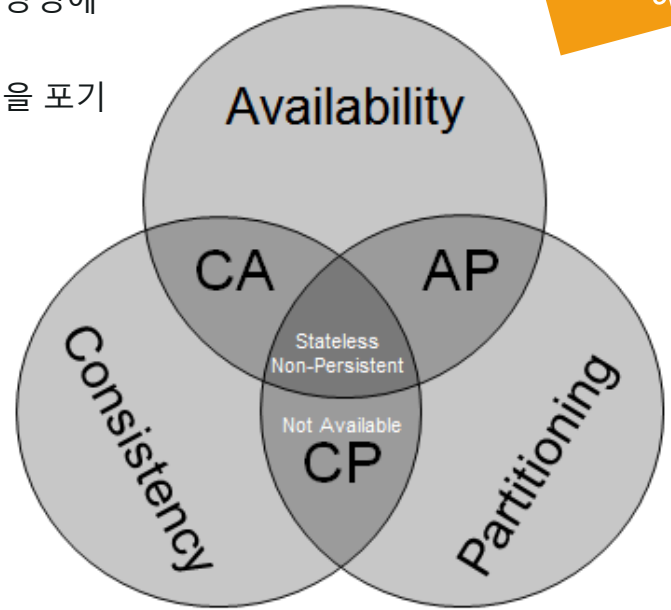
- 조회화면

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?

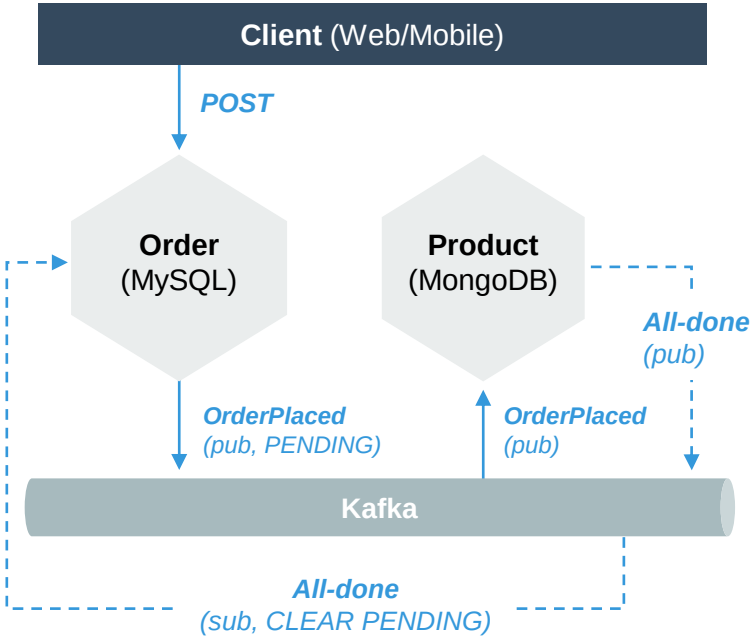
"성공한 내 서비스의 경쟁자들은 무엇을 포기했는가?"

➔ 강력한 의사결정 필요
(무엇으로 태어날 것인가...)



Eventual TX – 불일치가 Mission Critical 한 경우

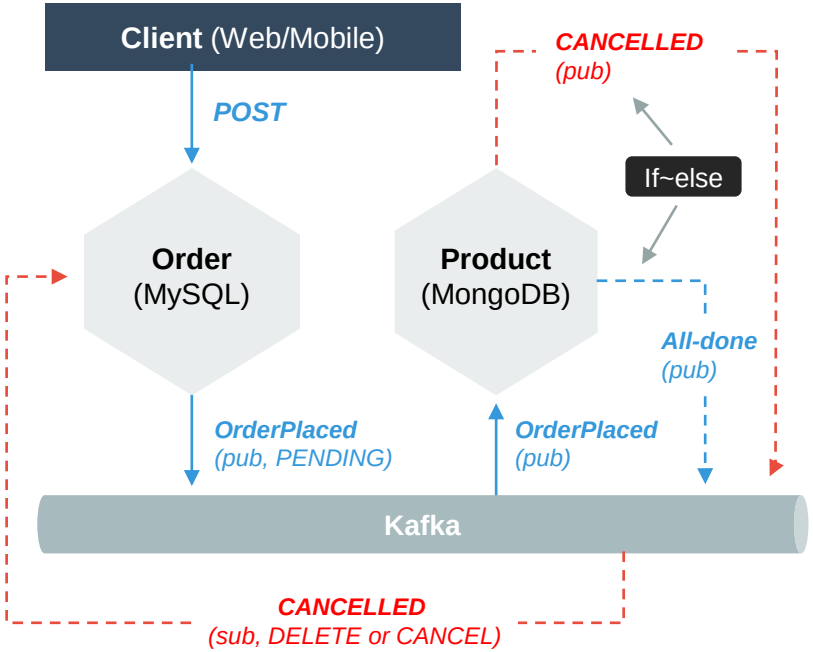
- 서비스구성 (Eventual TX)



- 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

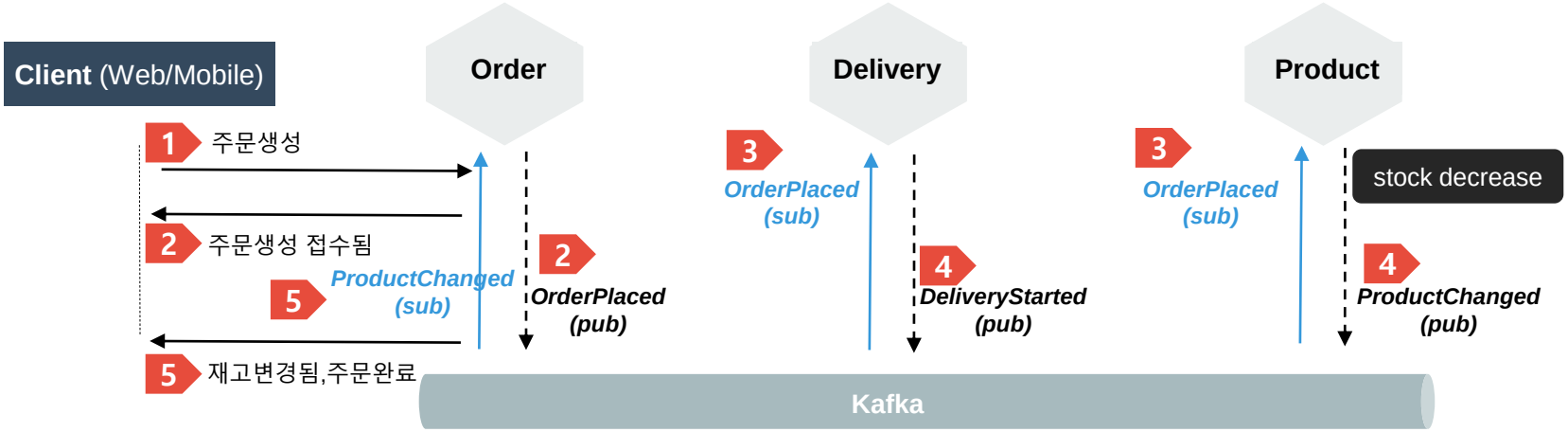
주문량 : 2 , 재고량 : 9 (PENDING)

주문량 : 2 , 재고량 : 8 → **취소됨**

주문량 : 2 , 재고량 : 9

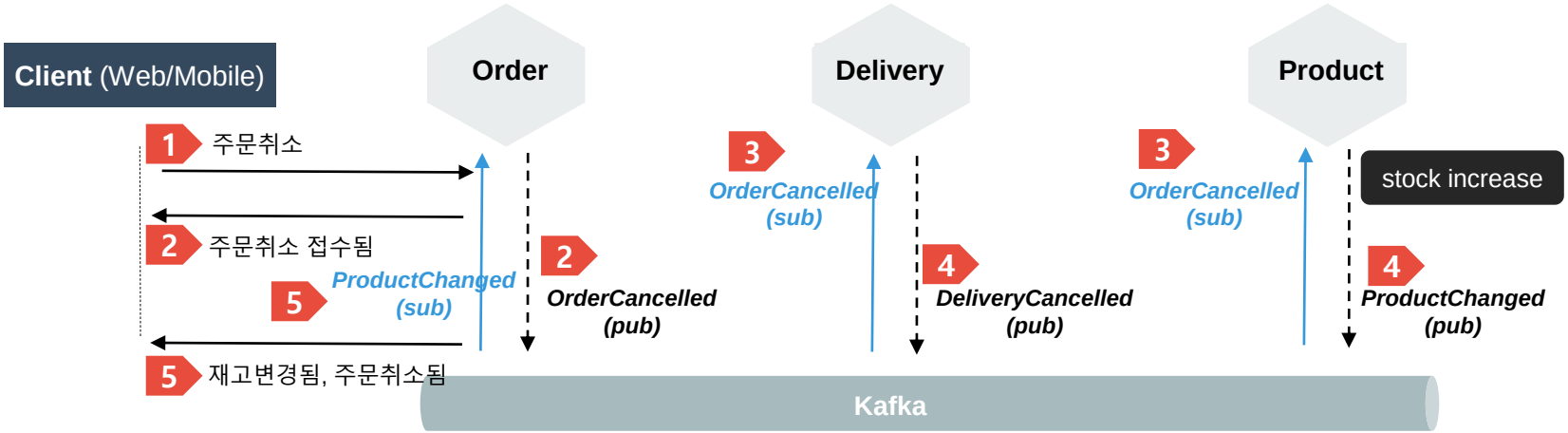
복구 & 결국 일치

Lab Time: Eventual TX – Saga Pattern (1/5)



- 확인 방법
  - 3개의 서비스 시작 order, product, delivery
  - 초기 재고량 확인
    - order svc -> http localhost:8081/products/1
    - product svc -> http localhost:8085/products/1
  - 주문 시작 API
    - http localhost:8081/orders productId=1 quantity=1 customerId="1@uengine.org" customerName="홍길동" customerAddr="서울시"
  - 이벤트 조회 topic > eventTopic
    - /usr/local/bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic eventTopic --from-beginning
  - 재고량 확인

Lab Time: Eventual TX – Saga Pattern (2/5)



- 확인 방법
  - 주문 취소 API
    - http PATCH localhost:8081/orders/1 state=OrderCancelled
  - 이벤트 조회 topic > eventTopic
    - /usr/local/bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic eventTopic --from-beginning
  - 재고량 확인

Lab Time: Eventual TX – Saga Pattern (3/5)

- Quiz

- Event Driven 으로 시스템을 구현할때
  재고량 체크는 어느 서비스에서 하는게 좋을까?
  1. 주문서비스 2. 상품서비스

- 현재 github/event-storming 코드는 주문서비스에서 재고량을 체크
  이것을 상품서비스에서 재고량을 체크하도록 변경해보자

  - 주문 서비스에서 우선적으로 주문을 받고 (재고량 체크 X)
    상품 서비스에서 주문시 재고량을 파악해서, 주문을 취소하는 로직을 구현

Lab Time: Eventual TX – Saga Pattern (4/5)

- Request/Response 방식이라면?
  - 상품 서비스 에서 주문 서비스 에게 해당 상품의 주문을 취소하는 API 호출
  - 주문 서비스 에서 배송 서비스 에게 취소된 주문 건에 배송을 취소하는 API 호출
  - 구매한 사용자에게 양해 메일 발송 – optional

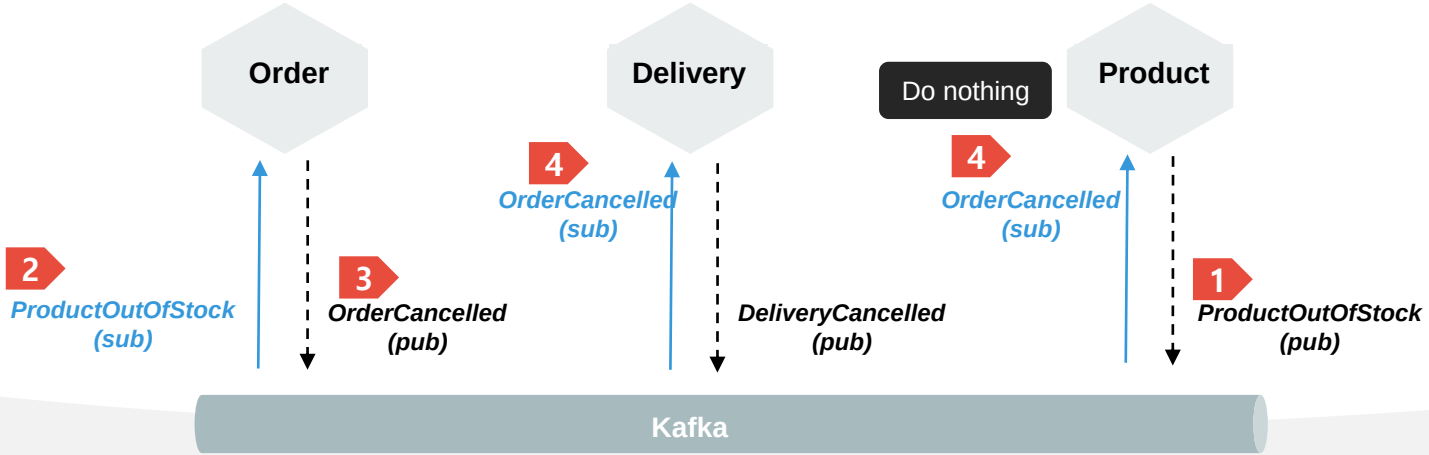
- Event Driven 방식이라면?
  - 상품 서비스에서는 ProductOutOfStock 이벤트를 발생함
  - 주문 서비스에서는 해당 이벤트를 받아서 주문을 취소함
  - 상품 서비스에서는 OrderCancelled 이벤트시 OutOfStock 상황이면 stock increase 를 안함

- 사전준비
  - 주문 서비스에서 기존의 재고량 체크하는 로직을 주석 처리함
    - order 서비스 Order.java > throw new OrderException("No Available stock!"); 주석
  - 상품 서비스에서 재고량을 체크함
    - product 서비스 ProductService.java > @StreamListener 부분에 아래처럼 변경

```
if( product.getStock() < 0 ){
    System.out.println("productOutOfStock 이벤트 발생");
}else{
    productRepository.save(product);
}
```

Lab Time: Eventual TX – Saga Pattern (5/5)

- https://github.com/event-storming/products/ 의 **saga-rollback** 브랜치 확인
- https://github.com/event-storming/orders/ 의 **saga-rollback** 브랜치 확인



작업 방향

1. product 서비스에서 ProductOutOfStock 이벤트 클래스 생성
2. ProductService.java 에서 ProductOutOfStock 이벤트 발송
3. order 서비스 > Orderservice.java 에서 ProductOutOfStock 이벤트를 리슨
4. ProductOutOfStock 발생시 OrderCancelled 이벤트를 발송
5. ProductService.java 에서 OrderCancelled 이벤트 받았을때, ProductOutOfStock 상황일때 재고량을 늘리지 않는다.

Resources on Sagas

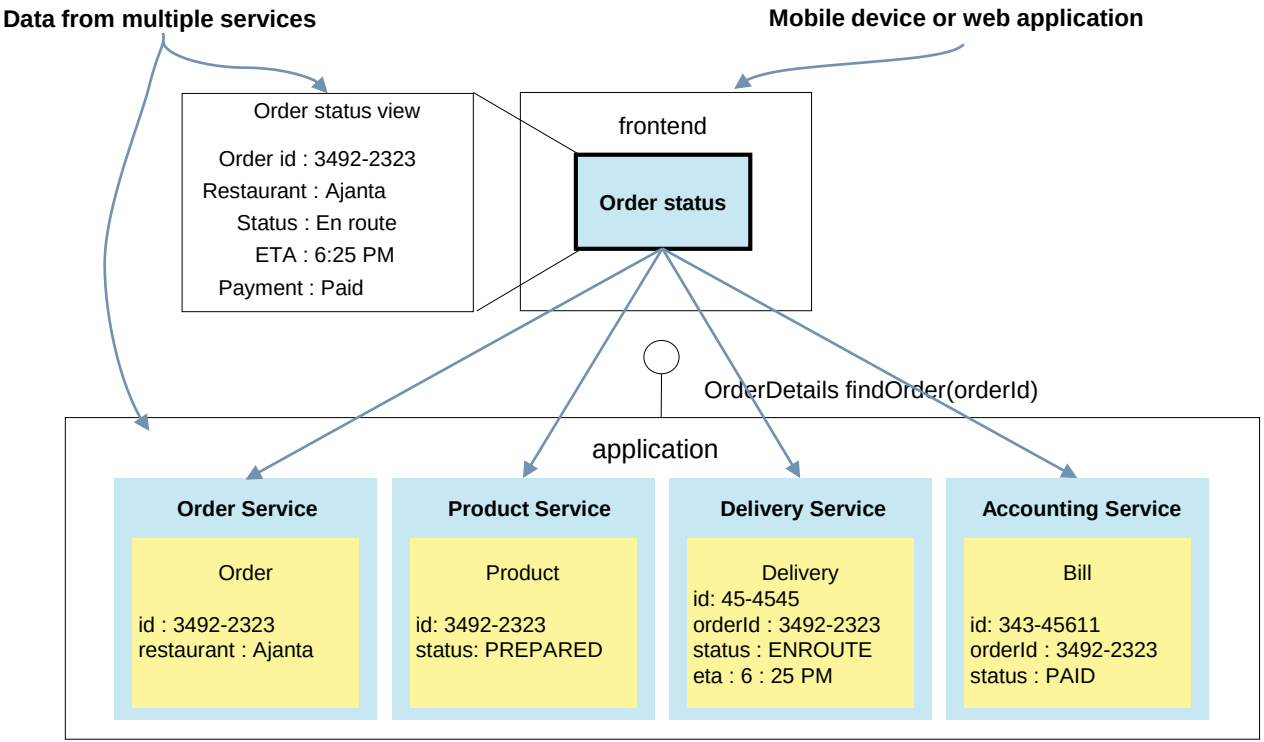
- http://eventuate.io/

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

Data Query (Projection) in MSA

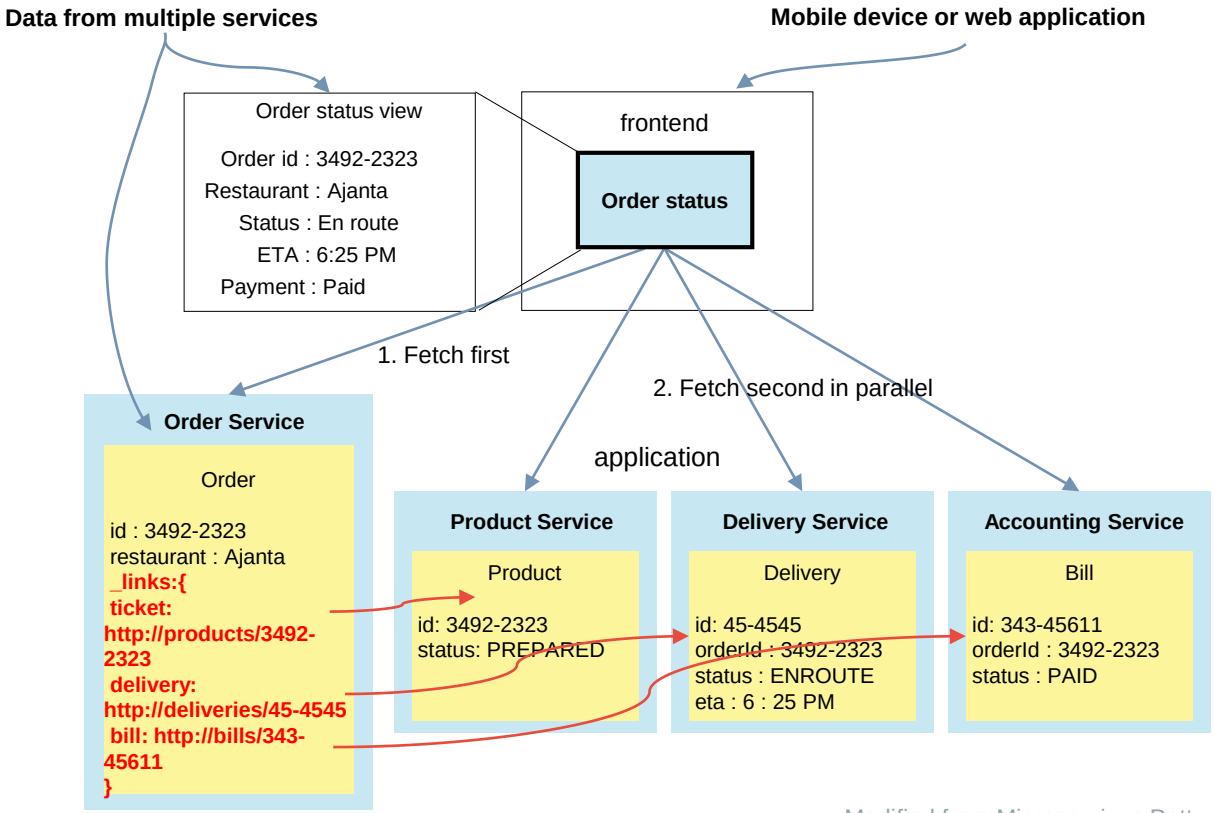
- Issues: Existing Join Queries and Performance Issues
- Composite Services or GraphQL
- Event Sourcing and CQRS

분산서비스에서의 Data Projection Issue



Issues :
1. 성능/속도
2. 데이터량

Data Projection by UI and HATEOAS



Issues :
- 단점 : UI 개발 복잡성과 성능

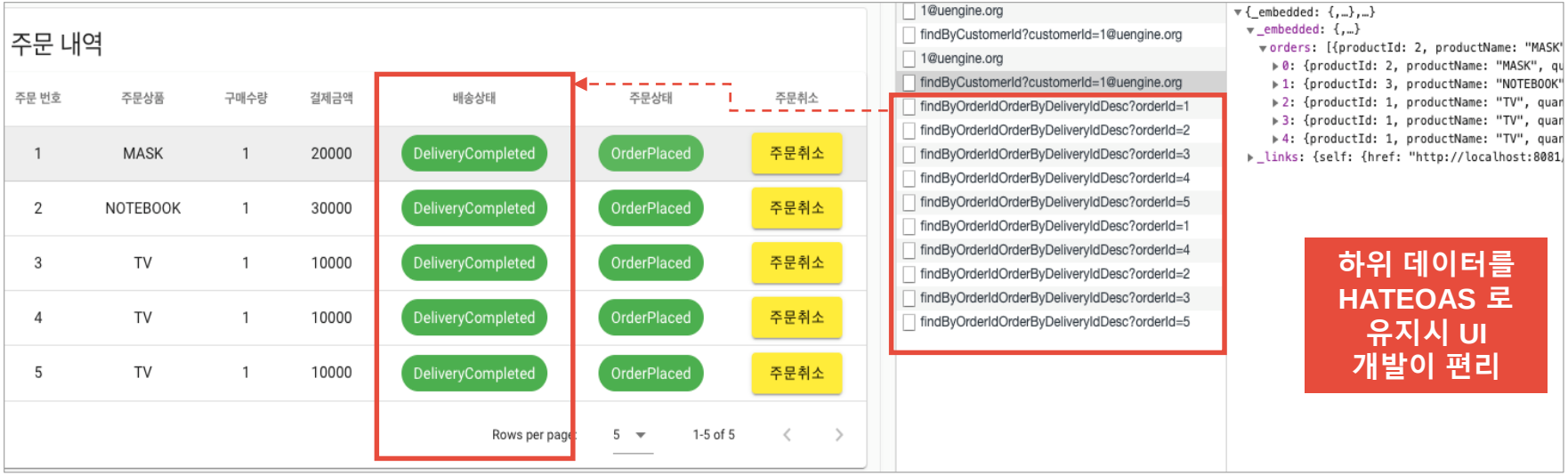
Lab Time: Data Projection by UI and HATEOAS (1/2)

```
// 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
        })
    })

})
```

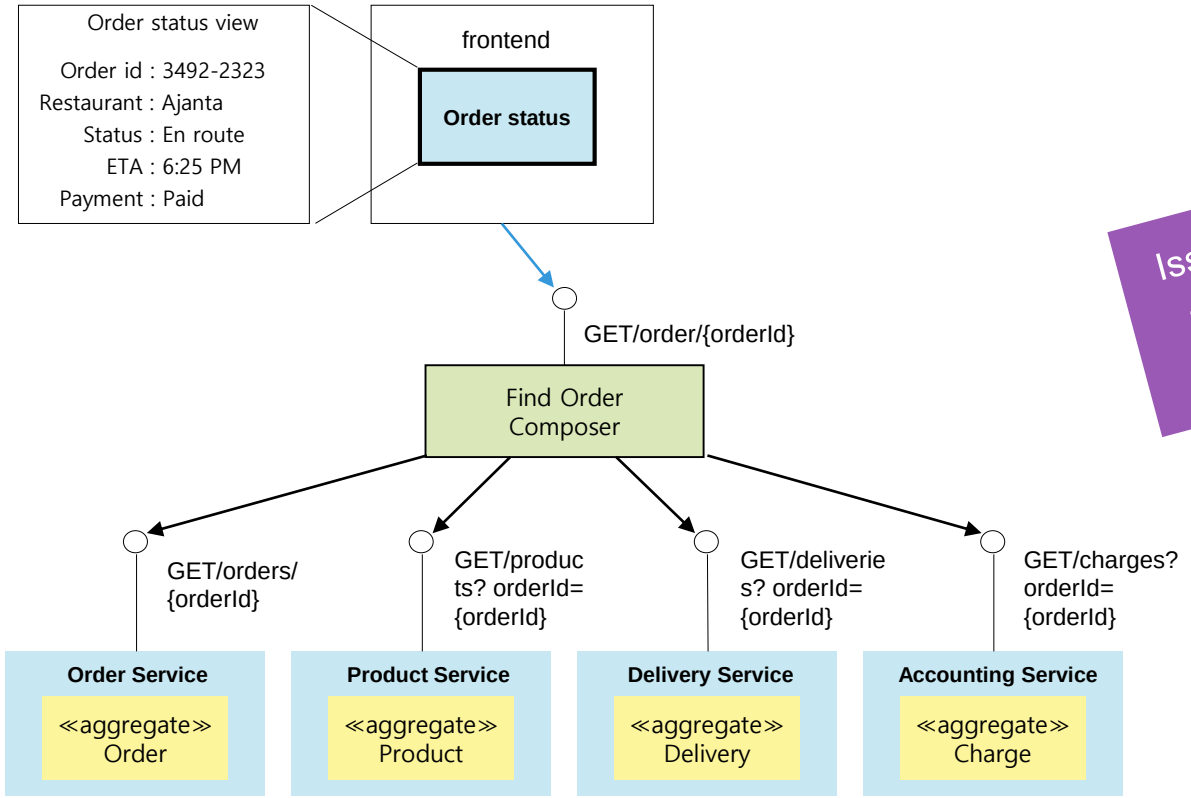
https://github.com/event-storming/ui/blob/master/src/components/order/OrderListMashup.vue

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

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



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());

});
```

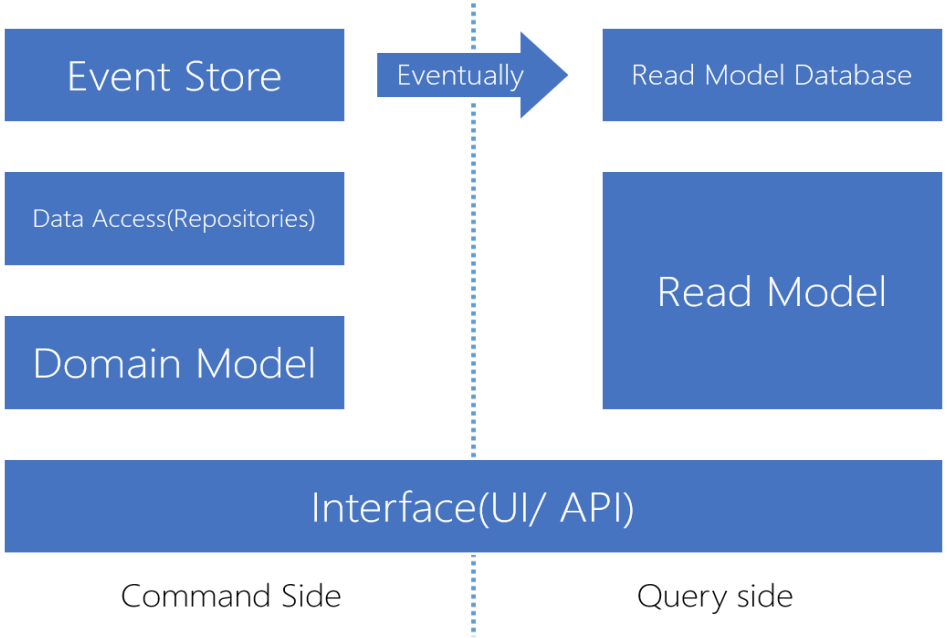
https://github.com/event-storming/composite_service/blob/master/src/main/java/com/example/template/CompositeService.java

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

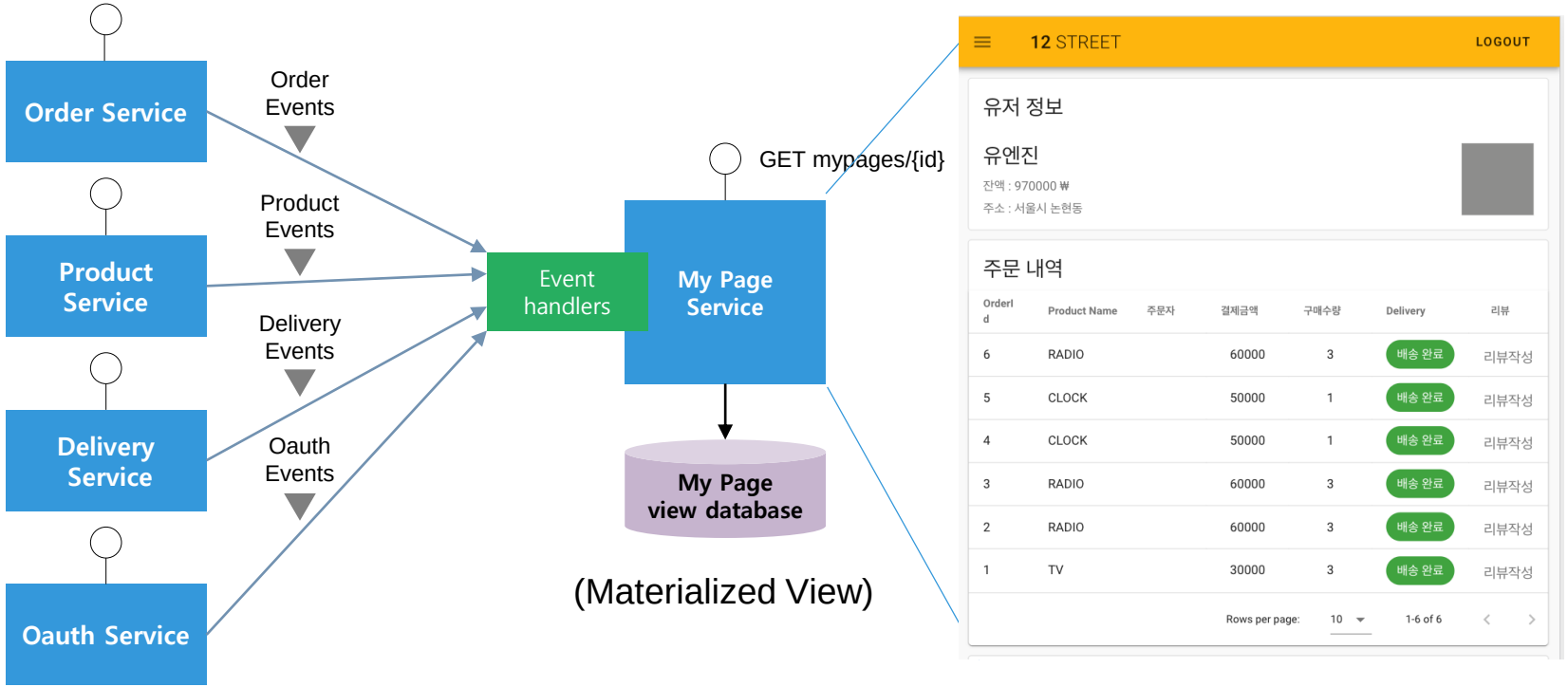
- CQRS 패턴
- 읽기전용 DB와 쓰기 DB를 분리함으로써 Optimistic Locking 구현
- Query 뷰를 다양하게 구성하여 여러 MSA 서비스 목적에 맞추어 각 서비스의 리드모델에 부합
- Polyglot Persistence



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



CQRS 의 확장 : Multiple Event Sources

```
@StreamListener(KafkaProcessor.INPUT)
public void onMessage(@Payload String message) {
        /**
         * 배송 시작 이벤트
         */
        if( event.getEventType().equals(DeliveryStarted.class.getSimpleName())){
            …..
            orderHistoryRepository.save(orderHistory);
        }
        /**
         * 배송 취소 이벤트
         */
        else if( event.getEventType().equals(DeliveryCancelled.class.getSimpleName())){
            …..
            orderHistoryRepository.save(orderHistory);
        }
        /**
         * 주문 생성 이벤트
         */
        else if( event.getEventType().equals(OrderPlaced.class.getSimpleName())) {
            …..
            orderHistoryRepository.save(orderHistory);
        /**
         * 주문 취소 이벤트
         */
        else if( event.getEventType().equals(OrderCancelled.class.getSimpleName()))
             …..
            orderHistoryRepository.save(orderHistory);
}
```

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

Table of content

Microservice and
Event-storming-Based
DevOps Project

UEngine

Lab Time: Application 에 설정값 주입 (1/3)

- git clone https://github.com/event-storming/monolith.git

- application.yaml
  - superuser.userId *: ${_SUPER_ID:admin123}*
  - superuser.userId *: ${환경변수명:기본값}*

- 사용
  - @value("${superuser.userId}")

  - Environment env = Application.*applicationContext*.getEnvironment();
    String datasourceUrl = env.getProperty("superuser.userId")

- 브라우저에서 실행 하여 결과값 확인
  - mvn spring-boot:run
  - http://localhost:8088/env

Lab Time: Application 에 설정값 주입 (2/3)

- MAVEN 실행시
  - export _SUPER_UID=prod_admin
  - mvn spring-boot:run
  - mvn spring-boot:run -Dspring-boot.run.arguments=--superuser.userId=dev_admin
  - mvn spring-boot:run -Dsuperuser.userId=dev_admin
- Docker 실행시
  - docker run –e _SUPER_UID=docker_admin ..

Lab Time: Application 에 설정값 주입 (3/3)

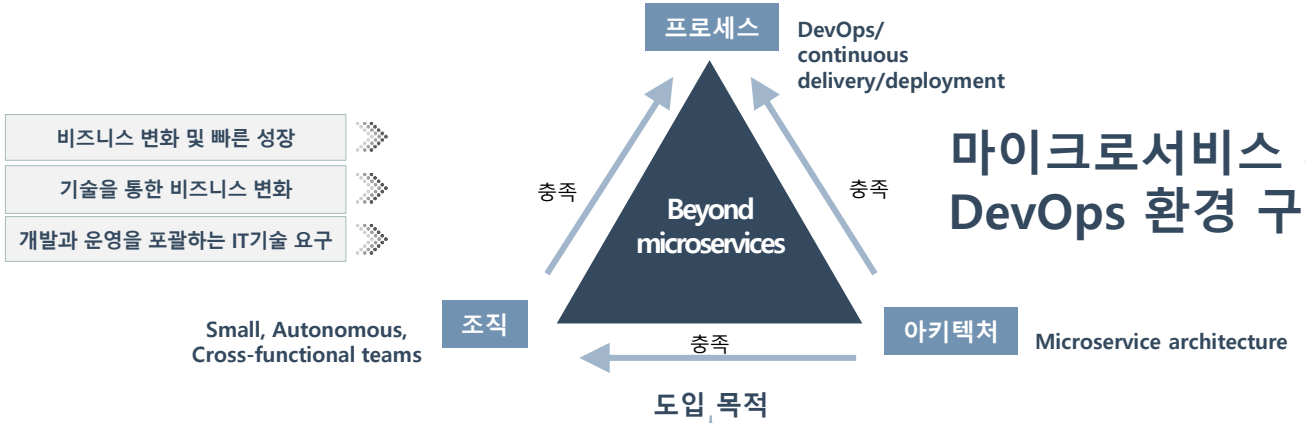
- Kubernetes ConfigMap 을 활용하여 개발/운영 설정 나누기
  - 개발 클러스터 - configmap 생성
    - kubectl create configmap admin-config --from-literal=adminUser=kube_dev_admin
  - 운영 클러스터 - configmap 생성
    - kubectl create configmap admin-config --from-literal=adminUser=kube_prod_admin
  - 배포 deployment.yaml 에 env 값 설정

```
env:
  - name: _SUPER_UID
    valueFrom:
      configMapKeyRef:
        name: admin-config
        key: adminUser
```

Appendix: MSA 프로젝트 관리와 조직 변화 관리

MSA 프로젝트 목표 수립

마이크로 서비스 아키텍처는 하나 또는 소수의 큰 시스템을 작은 크기의 서비스로 나누고, 서비스 별로 개별적인 데이터 저장소를 소유하고 독립된 실행 환경을 구축함으로써 서비스 단위로 독립적인 개발, 빌드, 테스트, 배포 모니터링, 라우팅, 확장이 가능한 아키텍처 입니다.



마이크로서비스 기반 DevOps 환경 구축

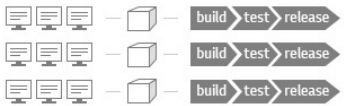
도입 목적

개발 생산성

- 서비스를 충분히 작은 크기로 나누어 독립적으로 개발하고 마이크로서비스간에는 API 규약에 따라 상호 연계
- 개발자는 비즈니스 로직에만 집중 가능
- 테스트 시에도 연관 마이크로서비스만 테스트를 수행하므로 배포 일정 축소

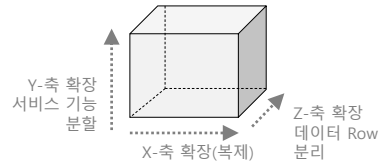
배포 유연성

- 마이크로서비스 아키텍처는 개별 서비스 단위로 개발/패키징/빌드/테스트/배포가 가능하므로 각 서비스를 각자의 로드 맵에 따라 개발 배포



확장을 통한 안정성

- 시스템의 기능을 분할하여, 서비스 단위 별로 트래픽에 대해 자율적 확장을 통한 성능 향상



Specific Goal Setting – MSA Maturity Levels

	Early	Inception	Expanding	Mature
기능분해	비즈니스 역량은 도출한 비즈니스 기능과 유즈케이스를 단위로 하여 분리 가능	뭉뚱그리지 말고 확실히 분해해라. 추출 된 각 유스 케이스와 인터페이스를 통해 액세스 할 데이터에 대해 잘 정의 된 인터페이스를 가짐.	컨텍스트매핑의 결과로 도출된 범위가 한정된 문맥 (Bounded context) 을 기준으로 분해한다. ubiquitous language 가 다른 bounded context간의 커뮤니케이션시 Anti-corruption layer를 통해 연동	이벤트 기반으로 식별된 도메인 영역, 머털 럴라이즈드 뷰, 읽기/쓰기 (커맨드) 를 위한 분리된 별도의 모델 (CQRS)
데이터	서비스간 스키마를 공유한다. 2 PC 를 사용할 수 있다. ACID 기반의 트랜잭션을 유지한다. Canonical Data Model 를 지향한다	각각의 서비스는 자신만의 데이터베이스를 가짐. 서비스들과 다중 엔터프라이즈 데이터 저장소간의 트랜젝션이 적은 조정으로 이루어짐.	-완전히 분산 된 데이터 관리.RDBMS, Search index. Document DB, Graph DB 등등 데이터를 해당 노드에 도달할 때까지는 데이터에 일관성이 없는 상태이나 일정 시간이 지나면, 다시 Consistency를 충족	이벤트 기반 데이터 관리, 이벤트 소싱 및 커맨드 쿼리
테스팅	통합, 회귀, 시스템 통합 테스트(SIT: System Integration Testing), 사용자 인수 검사(AUT: User Acceptance Testing)를 위한 부분 자동화된 유닛테스팅	Junit, Integration, Functional, SIT, UAT, Regression, Performance 의 완전한 자동 테스팅	component 테스트, A/B 테스트, 실패테스트(Chaos Monkey)	컨트랙트 테스트, 컨슈머 주도 계약, 테스트 데이터별 유저 퍼소나(persona)와 여정(journey)을 활용한 E2E 테스트
인프라 스트럭쳐	지속적인 빌드, 지속적인 통합 운용	지속적 딜리버리와 배포, 로그의 중앙 집중화	컨테이너 사용(도커), 컨테이너 지휘자(k8s), 외부 구성(유레카, 주키퍼)	자동 프로비저닝을 갖춘 PaaS기반 솔루션
배포	설치 스크립트 구동, 호스트 당 멀티 서비스 인스턴스	VM 당 하나의 서비스 인스턴스 클라이언트 사이드 로드벨런싱 서버사이드 로드벨런싱	컨테이너 당 하나의 서비스 인스턴스이고 변경 불가능한 서버, blue/green 배포	멀티 클라우드 및 멀티 데이터 센터 지원
모니터링	SPLUNK와 함께 사용되는 APM 툴(App dynamics)	LogStash, Elastic Search, Sensu, Kibana And Graphite 를 사용한 중앙집중 로깅	Docker daemon 사용하여 통계 및 상태정보 수집하고 이를 third party 모니터링 툴인 Circonus, App Dynamics. 에 전달	종합 트랜잭션, 추적 서비스 지원
거버넌스	IT와 운영부분 최고 경영진의 의사 결정을 통해 중앙화 된 거버넌스 제공	아키텍처 및 배포 결정을 담당하는 모노리스 및 마이크로서비스 팀 직원들과 공유 된 관리 모델	팀별 완전히 분산된 거버넌스, 높은 자율성, 높은 책임과 중앙 통제적 프로세스 중심 접근 방식으로로부터 벗어난다. 각 팀은 열차를 기다리지 않고 택시를 탄다! 팀별 배포 파이프라인.	다수의 자율팀간 조율에 필요한 일종의 중앙화된 관리
팀구조	개발,QA,릴리즈, 운영이 분리된 하나의 기능 팀	공유된 서비스 모델로 팀 공동 작업 내부 소스 공개	Product Team(Prod mgr, UX, Dev, QA, dbAdmin) and Platform team(Sys Admin, Net Admin, SAN Admin), 2 Pizza team.	업무 기능별 혹은 도메인별 팀들이 모든 관점에서 책임을 수반. "네가 구축한 것은 네가 운영한다."
구조	ESB(Enterprise Service Bus)에서 실행되는 기본 SOA기반 서비스, 여전히 단일 앱이지만 모듈화	마이크로서비스를 사용하여 개발 된 새로운 기능을 갖춘 모노리스 및 마이크로서비스의 하이브리드	마이크로 서비스를 사용하여 가벼운 미들웨어 통합 레이어를 접하는 API 게이트웨이	AWS 람다와 같은 이벤트 기반의 단일 목적을 위한 전용의 서비스

https://www.linkedin.com/pulse/microservice-maturity-model-enterprise-adoption-rishi-singh/

MSA 서비스 전환 투자 우선순위 식별

* 50% 이하 : MSA 부적합, 50% ~ 70% : 자체 검토, 70% 이상 : MSA 전환 대상

평가항목	측정항목	측정내용	측정방법	작성방법	기준 데이터 (예시)
비즈니스 확장성	업무변경량	이전 변경 요청 건수 확인하여 변경 많은 업무 확인	신규 기능/기능 변경 CSR 건수	1. 업무 변경 빈도가 높다고 판단할 수 있는 CSR건수 기준 정의 2. CSR 건수와 인터뷰 결과를 종합하여 변경빈도 높은 업무 Y로 표기	100건 / 월
		업무변경 현황 및 향후 업무변경 계획 확인	현업 인터뷰, 시스템운영자 인터뷰		
	확장가능성	채널 확장, 신규 업무/기능 도입, 신기술 적용 계획 확인	현업 인터뷰	1. 확장 가능성과 관련된 인터뷰 내용을 자유 기술로 작성 2. 계획이 확인되면 Y로 표기	정성적 판단
장애대응	트랜잭션발생량	시스템, DB 부하분석을 통해 트랜잭션 발생이 집중되는 업무 확인	현행 시스템 트랜잭션 발생량 측정	1. Read와 Write로 나누어 트랜잭션 발생량 측정하여 트랜잭션 발생량이 많다고 판단할 수 있는 기준 정의 2. 1번 기준에 따라 트랜잭션양이 많은 업무에 대해 Y로 표기	월 평균 50 TPS 이상
	데이터 용량	데이터 용량이 많아 부하와 성능에 영향을 줄 수 있는지 업무 확인	현행 시스템 기준 핵심 엔터티 데이터 용량 측정	1. 성능에 영향을 줄 수 있는 데이터량 기준 정의 2. 1번 기준 이상인 업무 Y로 표기	500GB 이상
	데이터 증가량	현행 시스템에 있는 업무 중 데이터 증가량이 많아 부하와 성능에 영향을 줄 수 있는 업무 확인	현행 시스템 기준 핵심 엔터티 데이터 증가량 측정	1. 성능에 영향을 줄 수 있는 데이터 증가량 기준 정의 2. 1번 기준 이상인 업무와 인터뷰 결과를 종합하여 Y로 표기	30GB 이상 / 월
		현해 시스템에 없는 업무 중 데이터 증가량이 많은 업무 확인	현업 인터뷰		
	비즈니스 영향	장애 발생 시 매출, 신뢰도 등 비즈니스적으로 손실이나 타격을 줄 수 있는 업무 확인	현업 인터뷰	1. 인터뷰 결과를 종합하여 비즈니스적으로 손실이나 타격을 주는 업무 Y로 표기	정성적 판단
	트랜잭션 피크 발생시점	트랜잭션 발생이 특정시점에 집중되는 업무 확인	현행 시스템 트랜잭션 분석	1. 부하와 성능에 영향을 줄 수 있는 트랜젝션이 발생하는 업무 확인하고 트랜잭션 발생 시점에 대해 자유 기술 2. 특정 트랜잭션이 발생하는 업무가 있으면 Y로 표기	순간부하 150TPS 이상
배포용이성	소스 라인 수 / 용량	소스 파일 라인 수 와 용량의 크기로 배포에 용이한 크기인지 확인	현행 시스템 기준 측정	1. 배포하기에 적합한 크기 기준을 정의 2. 1번 기준 이상의 크기인 업무인 경우 Y로 표기	소스 : 10,000 라인 이상 테이블 : 50개 이상
	테이블 개수	업무와 관련 있는 테이블 개수로 배포에 용이한 크기인지 확인	현행 시스템 기준 측정		
요구사항 요청 조직 독립성	요구사항 요청 조직 수	요구사항을 요청할 예상 조직 수가 2개 이상인지 확인	현업 인터뷰	2개 이상 조직인 경우 Y로 표기	2개 이상
운영 조직 독립성	운영 조직 수	시스템을 운영할 예상 조직 수가 1개 이상인지 확인	시스템 운영자 인터뷰	2개 이상 조직인 경우 Y로 표기	2개 이상

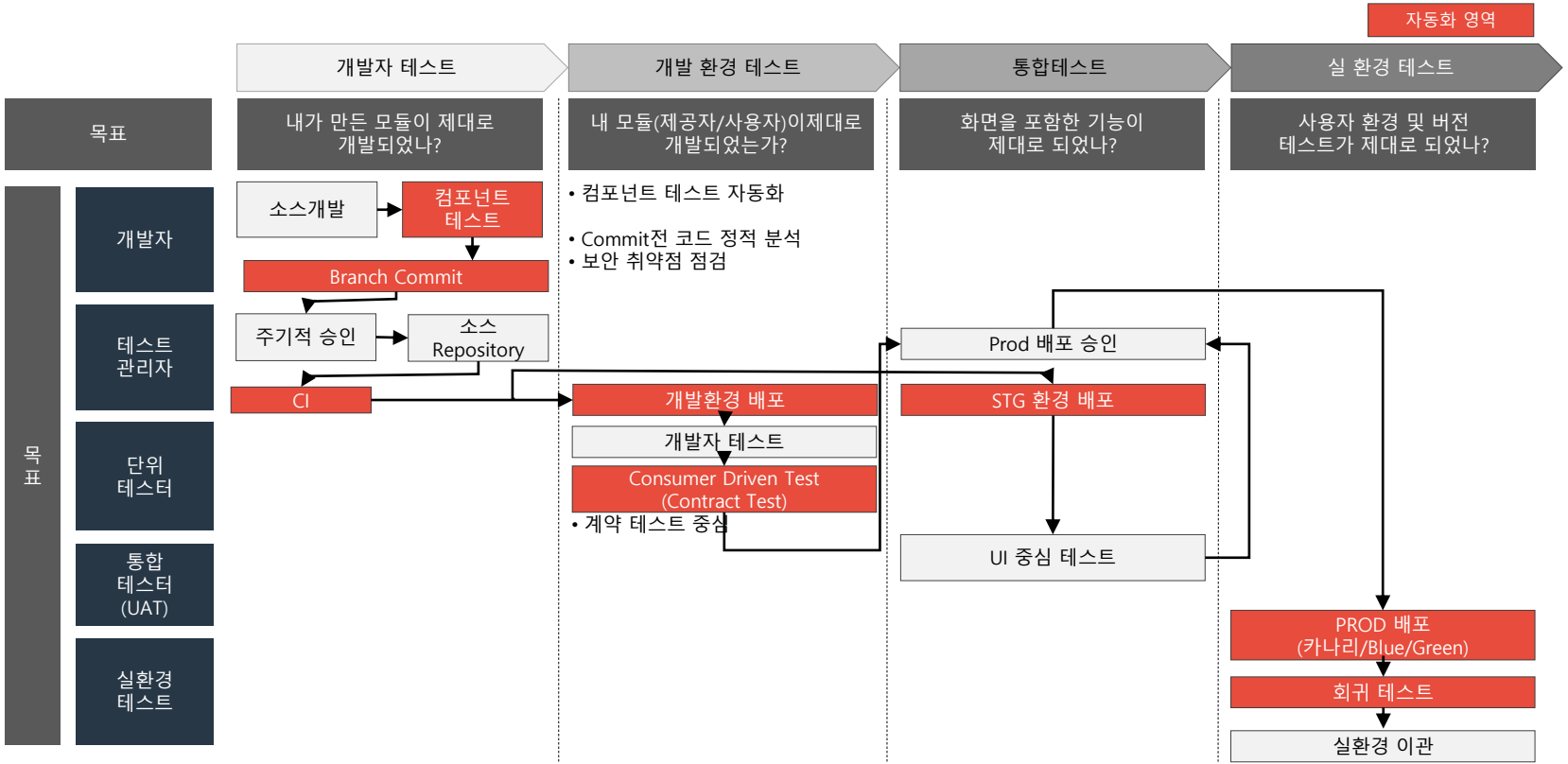
MSA 적용 전술 수립

아키텍처 패턴	체크 포인트	회피 패턴
Circuit Breaker	• 성능저하를 막아주나, Fail-fast 전략으로 사용자 경험이 나빠질 수 있음 • 적용대상이 비동기, 이벤트 기반으로 처리가능하다면 그 기반으로 전환	Saga
Database per service	• ACID 트랜잭션 비용을 포기 • ACID 트랜잭션 비용 포기가 불가하다면, Shared database 로 처리해야 하거나 Semantic Locking 통한 Eventual Transaction 상의 Lock 을 구현해야 함	Schema Per Service (Anti)
Service Registry	• 서비스 레지스트리의 유형 선택: API 기반(Eureka), DNS 기반(Kube-dns)	Saga, CQRS
Saga	• 하나이상의 마이크로서비스를 걸친 트랜잭션이 필요한 경우 & Database per service 패턴을 적용했을때 유효함 • 마이크로 서비스간 프로세스 실행시간이 상대적으로 길거나 예측하기 힘든 경우 (e.g. 결재), 비용이 높은 경우 (2PC 를 사용하기 힘든 상황)	Circuit Breaker
CQRS	• 하나 이상의 마이크로서비스에서 추출한 데이터로 뷰를 구성해야 하는경우 • 잦고 빠른 마이크로서비스 내에서의 Read 가 발생하는 경우에 사용	HATEOAS
Event Sourcing	• 이벤트 소싱은 비용이 높기 때문에 다음의 요구사항이 존재하는지 확인 필요: Undo 기능 등의 요구가 향후 생길 수 있는가?, 마이크로 서비스간의 폴리글랏 퍼시스턴스 요구?, 기능의 추가 잦음 • 이벤트 소싱에서의 이벤트는 Append only 이기 때문에 데이터의 Diff 의 정보를 충실히 포함해야 함	
Backends for frontends	• BFF 는 매번 composite service 를 구현해야 하기 때문에 관련한 frontend 의 유형이 매우 다양한 경우는 가능한 API Gateway 의 기능을 사용하거나 RESTful, HATEOAS 를 사용 권고	API Gateway
API Gateway	• API Gateway 의 유형이 다양하기 때문에 해당 기능과 역할에 따라, Service Mesh 혹은 기존 EAI (Camel library) 등에서 처리해야 하는 경우 발생할 수 있음.	BFF
Client-side UI Composition	• Server-side Rendering 은 Microservice 의 장점을 희석하므로 가능한 MVVM 기반 Client-side Rendering 을 적용해야 함	Server-side rendering (Anti)

MSA 성능 테스트 방안

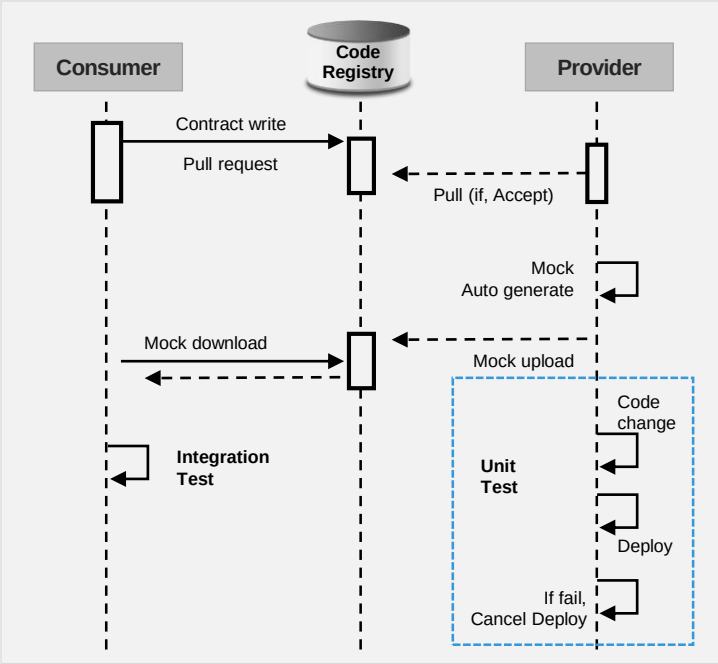
비기능 항목		품질지수 기준	비기능 성능지표 측정 방안	목표
가용성 (Availability)	가용률	가용률 측정	• 컨테이너 가용률(URL 접근성) 측정 • 측정 대상 : 컨데이너 기반의 어플리케이션 • 측정 기간 : 7 일 X 24시간	• 99.999% 이상 (Five Nines)
응답성 (Responsiveness)	응답시간	응답시간 측정	• 컨테이너 배포 시, 평균 응답시간 측정	• 평균 3초 이내
확장성 (Scalability)	확장성	리소스 확장 확인	• 컨테이너 자동/ 수동 확장 및 부하분산 처리 여부 확인	• 정상 동작
		확장요청 처리시간	• 컨테이너 확장요청 식별 시점부터 확장이 완료되기까지 소요된 시간 측정	• 평균 1분 이내
신뢰성 (Reliability)	서비스 회복시간	서비스 회복시간 측정	• 웹서버 장애(삭제) 발생 시, 서비스 회복 시간 측정	• 평균 1분 이내
			• DB 백업 파일 복구를 통한 평균 서비스 회복시간 측정	• 평균 10분 이내
	백업 준수율	백업 및 복구 기능	• 백업 및 복구 기능의 정상 동작 여부 확인	• 정상 동작

MSA 테스트 프로세스


자동화 영역
개발자 테스트
개발 환경 테스트
통합테스트
실 환경 테스트
목표
내가 만든 모듈이 제대로 개발되었나?
내 모듈(제공자/사용자)이제대로 개발되었는가?
화면을 포함한 기능이 제대로 되었나?
사용자 환경 및 버전 테스트가 제대로 되었나?
목표
개발자
테스트 관리자
단위 테스터
통합 테스터 (UAT)
실환경 테스트
소스개발
컴포넌트 테스트
Branch Commit
주기적 승인
소스 Repository
CI
• 컴포넌트 테스트 자동화
• Commit전 코드 정적 분석
• 보안 취약점 점검
개발환경 배포
개발자 테스트
Consumer Driven Test (Contract Test)
• 계약 테스트 중심
Prod 배포 승인
STG 환경 배포
UI 중심 테스트
PROD 배포 (카나리/Blue/Green)
회귀 테스트
실환경 이관

MSA 계약 테스트 방안

Consumer Driven Contracts Test 흐름도



✓ **DSL Contract 예제**

```
Contract.make {
  request {
    url "/check"
    method POST()
    body (
       age: value(regex('[2-9][0-9]')))
    headers {
      contentType(applicationJson()) }}
  response {
    status 200
    body("""
      {"status": "OK" }
"""")
}
```

```
Contract.make {
  description "should return person by id=1":
  request {
    url "/person/1"
    method GET()
  response {
    status OK()
    headers {
      contentType applicationJson()
    body(
      id: 1,
      name: "foo",
    )
  }
}
```

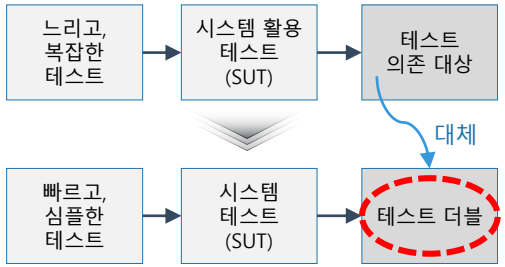
상세 설명

- 마이크로서비스 테스트는 Contract-based Test가 특징
- API Consumer가 먼저 테스트 작성
- API Provider가 Pull Request 수신 후 테스트 수행 및 Consumer Mock 객체가 자동 생성
- API Consumer는 생성된 Mock 객체를 통해 Stub 서버를 자동 생성하여 테스트 케이스 수행
  (Spring의 경우, 내부적으로 Wire mock 서버가 실행되어 Contract API 노출)
- **이를 통해 Provider의 일방적인 버전업에 따른 하위 호환성 위배를 원천적 방지**

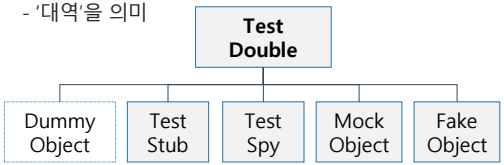
MSA 단위 기능 테스트 방안

단위(Unit) 테스트

- 소스코드의 특정 모듈이 의도된 대로 작동하는지에 대한 검사로 MSA에선 테스트 더블(Mock 객체)을 활용하여 테스트



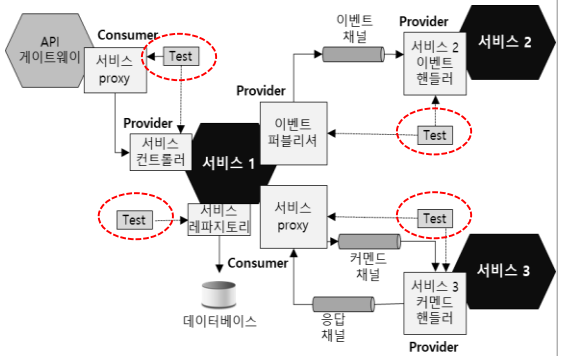
- 테스트 더블이란?
  - '대역'을 의미



출처 : http://xunitpatterns.com/TestDouble.html

서비스(통합) 테스트

- 배포 완료된 마이크로서비스에 대해 수행
- 단위 테스트의 상위 레이어에 위치하며, 구체적인 로직보다는 실제 네트워크 호출이나 데이터베이스 호출을 포함



- 서비스 테스트 전략의 일환으로, 모든 서비스를 테스트 하는 대신, 서비스간 통신을 담당하는 각 어댑터를 테스트 하는 방안도 고려

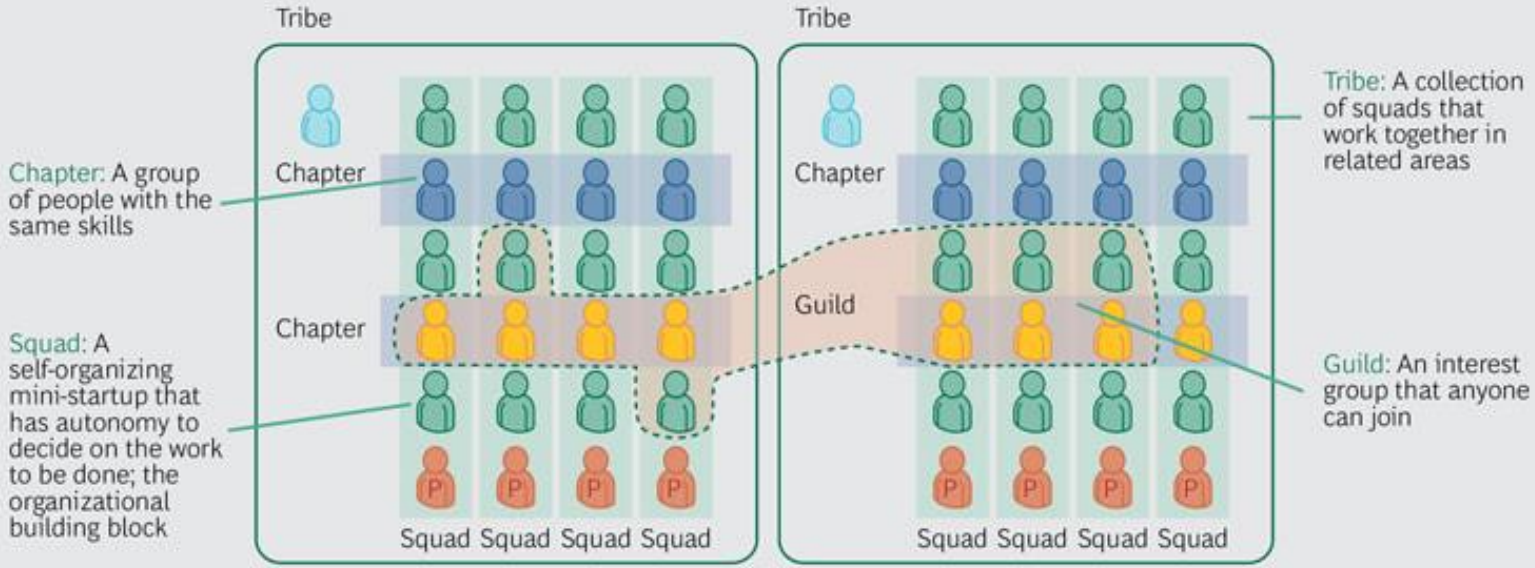
UI(E2E) 테스트

- 시스템의 모든 컴포넌트가 예정된 방식으로 잘 동작하는지를 확인하는 테스트
- 모든 서비스에 존재하는 엔드 포인트에 대한 부하 및 성능 테스트도 고려

- 거의 수행하진 않으나, BDD(Behaviour-Driven Development)를 기반으로 레코딩된 테스트를 수행하는 방안도 고려

- BDD(Behaviour-Driven Development) 란?
  - TDD와 유사하나, TDD는 테스트 자체에 집중하여 개발하는 반면, BDD는 비즈니스 요구사항에 집중하여 테스트 케이스를 개발

조직 전환 방안 – TO-BE



Source: Spotify

사례1 – ING BANK

This way of working is based on 8 important principles...

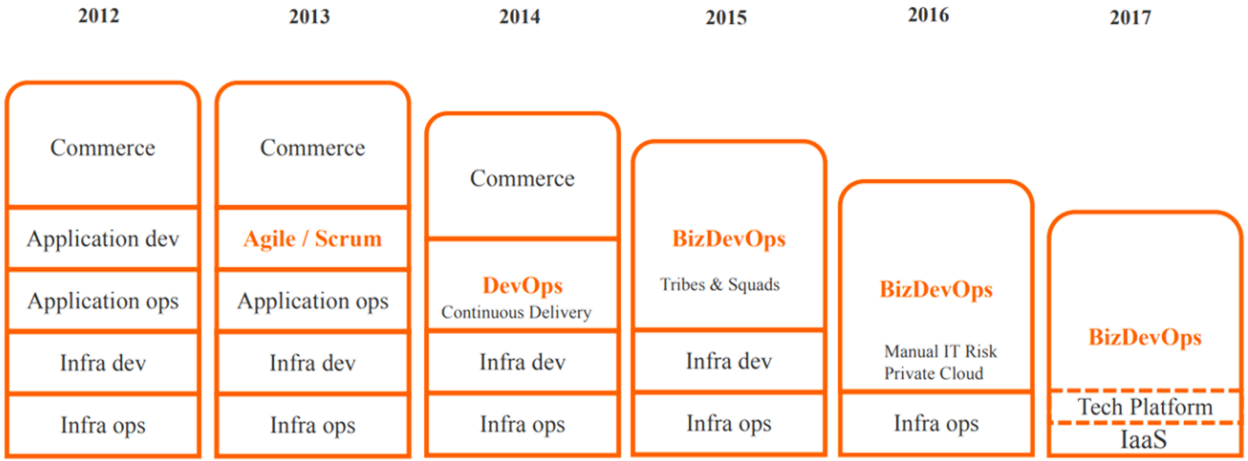
Principles

Inspired by the Agile methodology, eight principles are at the heart of our Way of Working:

1. We work in high **performing** teams
2. We **empower** teams
3. We care about talent and **craftmanship**
4. We continously **learn** from customers and apply learning to improve
5. We set **priorities** with the big picture in mind
6. We are consistent in our **organisational** design and way of working
7. We organise for **simplicity**
8. We **re-use** instead of reinvent

사례1 – ING BANK

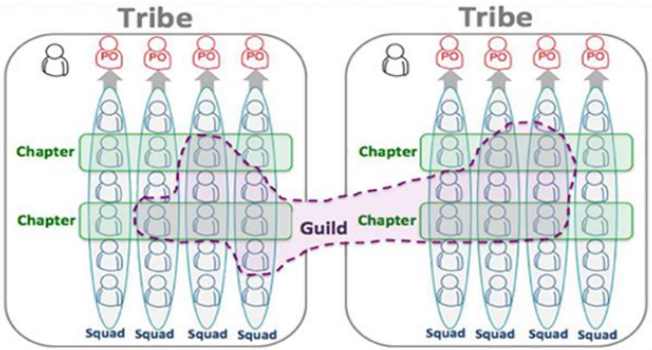
In the past five years, ING has been reorganizing for speed and skill. Roles and responsibilities have shifted radically



Engineer: From single discipline to **full stack engineers**: designing, coding, test engineering, infra engineering, etc
Product Owner: From writing PIDs to product vision and backlog to **end to end bizdevops responsibility**
IT Manager: from delivery manager to perhaps the most differntiating role: **skill and competency coach**.

사례1 – ING BANK

And this is how we organize ourselves

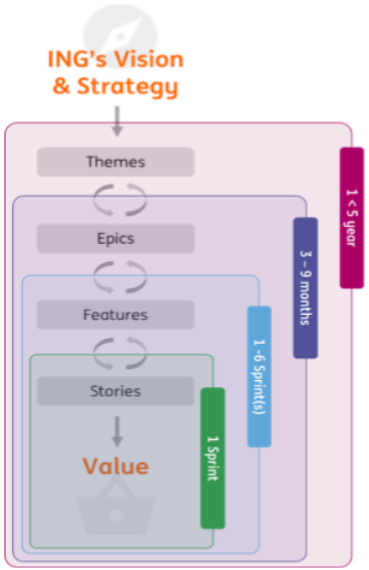


- **Squads**, are high performing "stable" teams who provide the execution power to the One WoW
- A **Tribe** is a collection of Squads organized around the same purpose
- **Chapters** are formal "groups" of members with the same background / expertise deciding on how things need to be done within a Tribe, regarding their area of expertise
- A **Guild** is a group of experts or community, which can be set up across Tribes (and even across countries) typically based on a shared interest in a technology or product / service

Squad over I, Tribe over Squad, Company over Tribe, Customer over Company

사례1 – ING BANK

These are the fundamentals of One Way of Working Agile

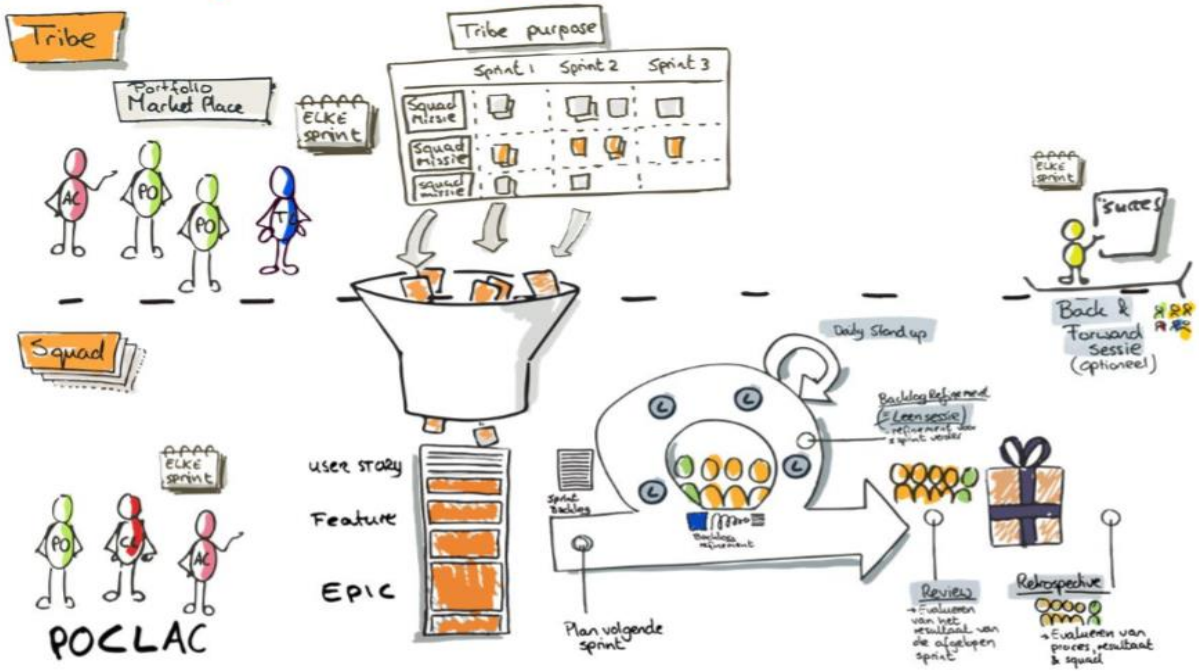


- A **Theme** connects the dots between Strategy and execution. Hence, a Theme represents an (investment) area that supports the strategic vision. Themes are representing one to five year(s).
- A Theme is broken down in multiple **Epics**: large scale bank initiatives that will be delivered in one to three quarters. At the Epic level, a.o. the benefits should be defined.
- An Epic can be broken down in one or more **Features**. A Feature reflects **part of the value** (both functional and non-functional) for a stakeholder that could be delivered within multiple (typically one to six) Sprints.
- Features have to be refined to **Story** level. Stories are explicitly defined by teams themselves. They have to be small enough to be picked up by one team and delivered within one Sprint.

A structured way to manage the demand and focus on the minimum viable products

사례1 – ING BANK

Our way of working in the new organisation



Recommended Readings

1. **Overall MSA Design patterns:**
   https://www.manning.com/books/microservices-patterns

2. **Microservice decomposition strategy:**
   - DDD distilled: https://www.oreilly.com/library/view/domain-driven-design-distilled/9780134434964/
   - Event Storming: https://leanpub.com/introducing_eventstorming

3. **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

1. **API design and REST:**
   http://pepa.holla.cz/wp-cont…/…/2016/01/REST-in-Practice.pdf

THANKS!

Any Question?

You can find me at:

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