

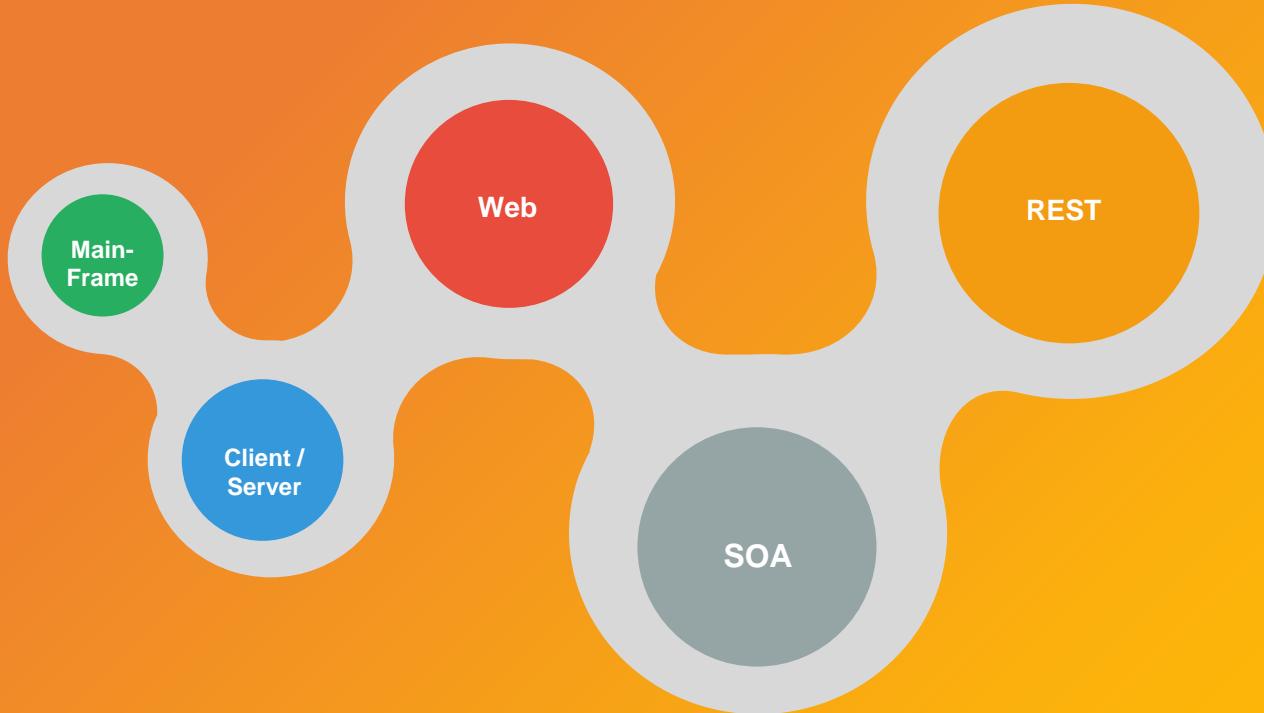
Developing Cloud Native Applications

3rd Jan 2020, ver2.0



Copyright © 2020. Jinyoung Jang & uEngine-solutions All rights reserved.

소프트웨어 아키텍처의 성장 여정



AS-IS: Pain-points

A 사 의료분야 SaaS 운영

- 서비스 업그레이드가 수시로 요청이 들어와 거의 매일 야근중. 개발자 행복지수가 매우 낮음.
- 테넌트별 다형성 지원을 제대로 하지 못하여 가입 고객이 늘 때마다 전체 관리 비용이 급수로 올라가는 한계에 봉착함
- 자체 IDC를 구성하여 하드웨어, 미들웨어 구성을 직접해야 하는 비용문제.

- 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 모놀로직 아키텍처의 한계로 장기적인 발전의 한계에 봉착

B 사 제조분야 SaaS 운영

Agile Delivery

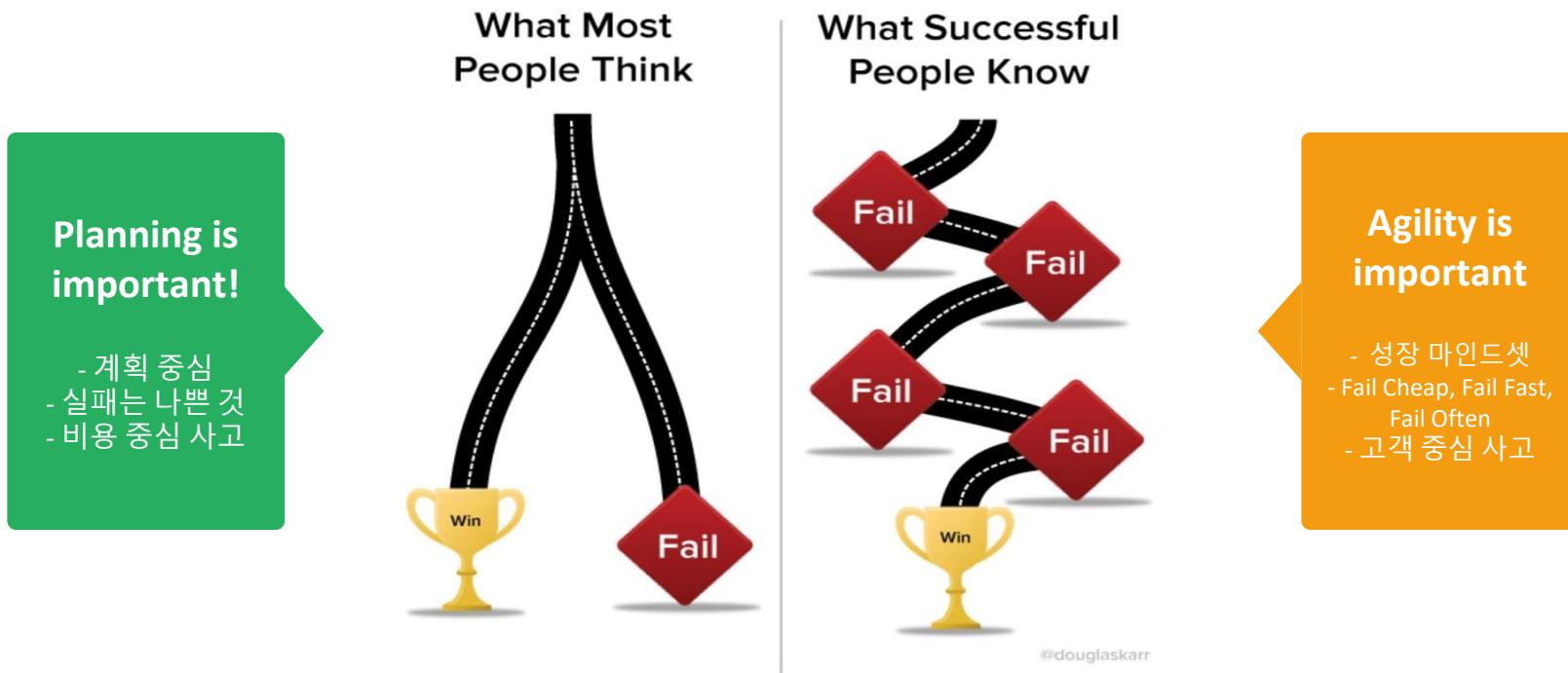
Amazon, Google, Netflix, Facebook, Twitter는 얼마나 자주 배포할까요?



Company	배포 주기 Deploy Frequency	배포 지연 시간 Deploy Lead Time	안정성 Reliability	고객 요구 응답성 Customer Responsiveness
아마존	23,000 / 일	몇 분 (Minutes)	높음	높음
구글	5,500 / 일	몇 분 (Minutes)	높음	높음
넷플릭스	500 / 일	몇 분 (Minutes)	높음	높음
페이스북	1 / 일	몇시간 (Hours)	높음	높음
트위터	3 / 주	몇시간 (Hours)	높음	높음
일반회사	9개월 주기	수개월~수분기별	낮음 / 보통	낮음 / 보통

출처: 도서 The Phoenix Project

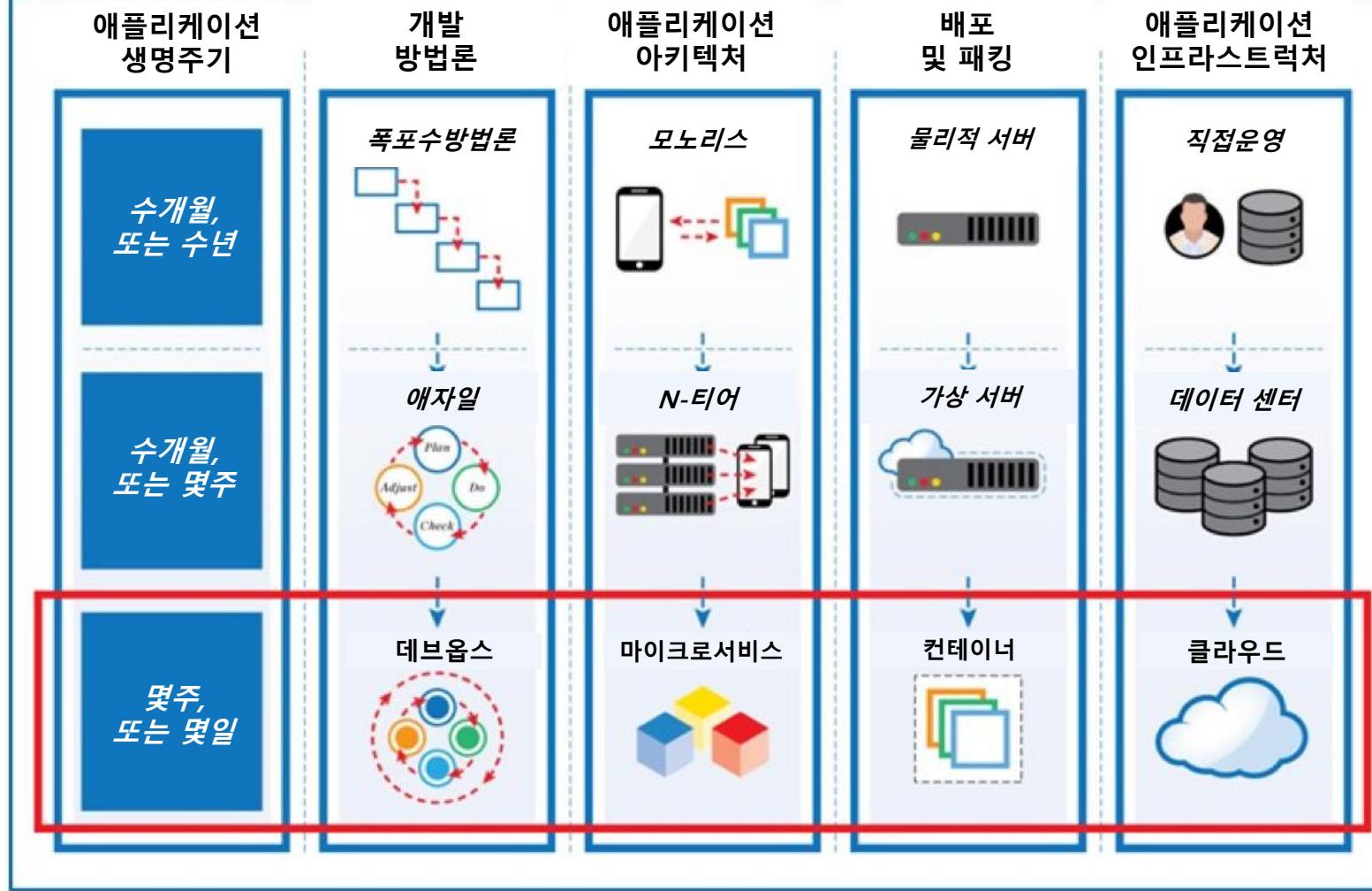
Agile 의 정의



Agility is important

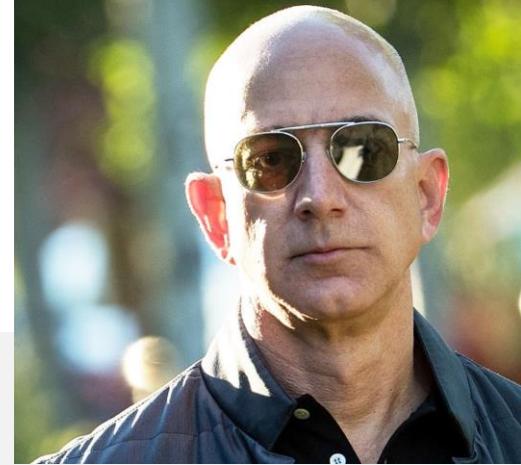
- 성장 마인드셋
- Fail Cheap, Fail Fast, Fail Often
- 고객 중심 사고

애자일에 필요한 것들



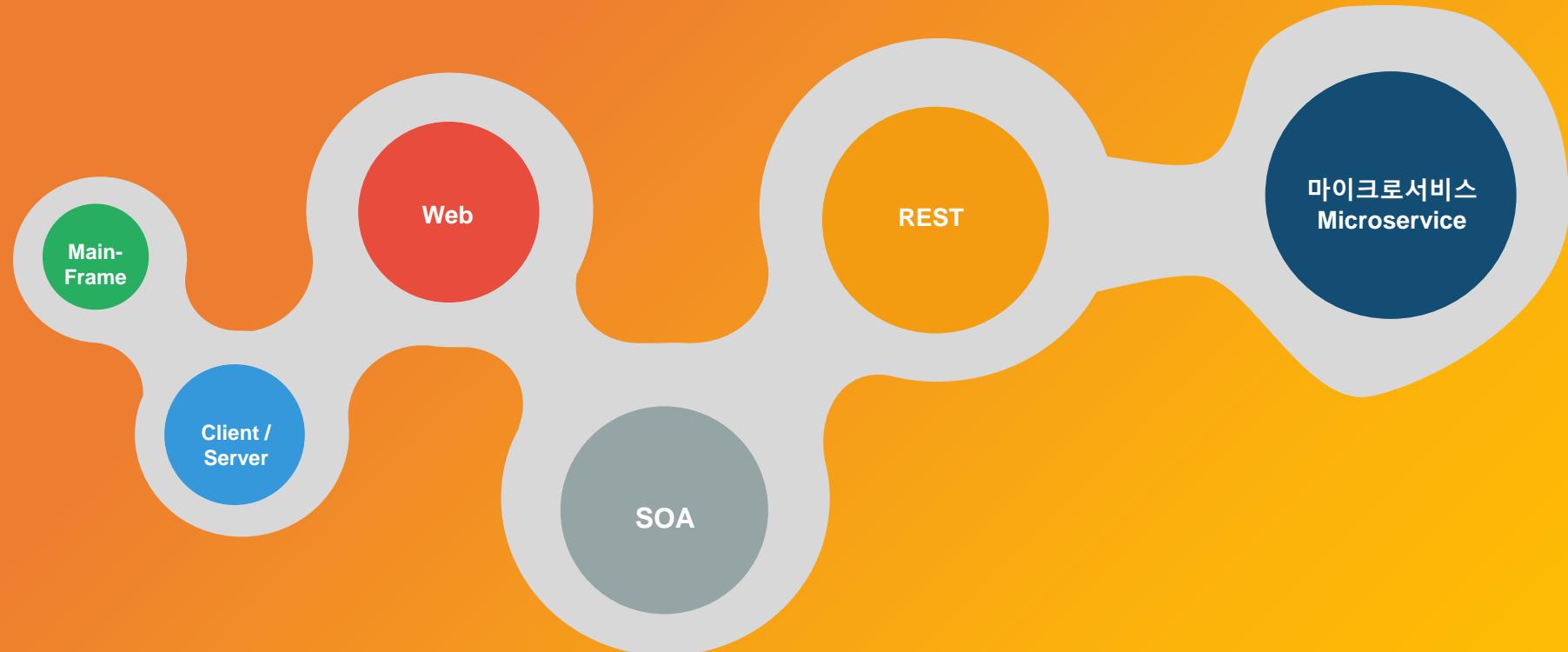
제프베조스의 의무사항

”



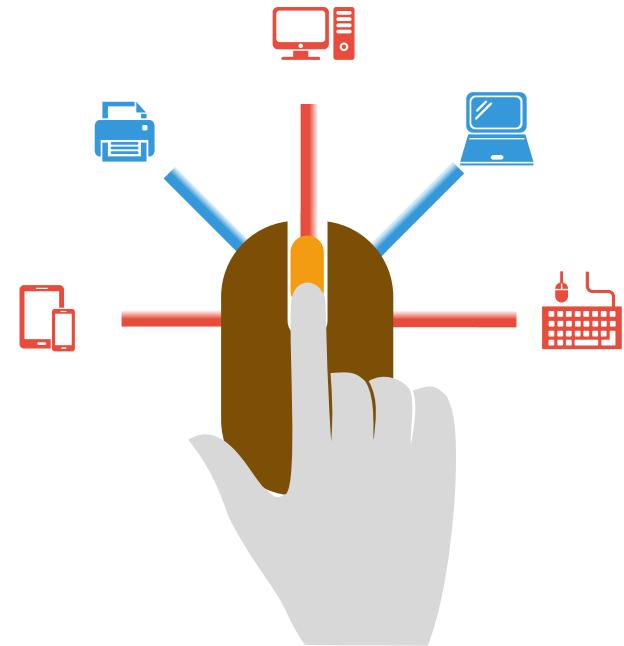
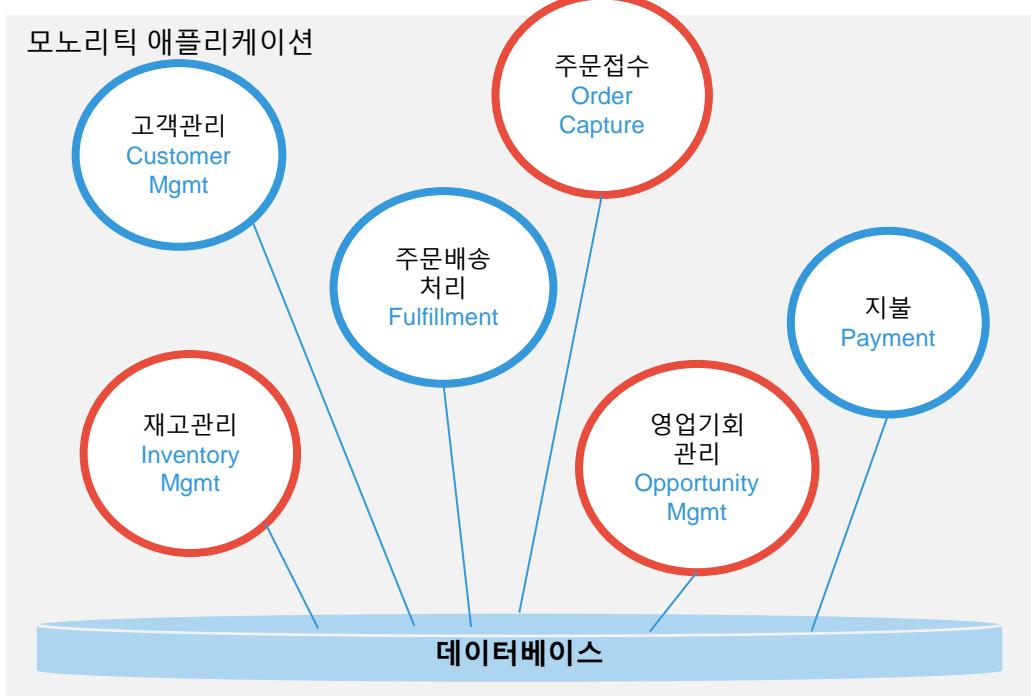
1. All teams will henceforth expose their data andfunctionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. 다음과 같은 그 어떠한 직접적 서비스간의 연동은 허용하지 않겠다:
no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
...
4. The team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
5. Anyone who doesn't do this will be fired.

소프트웨어 아키텍처의 성장 여정



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

An Enterprise Application or Suite



모노리티크 아키텍처의 분석

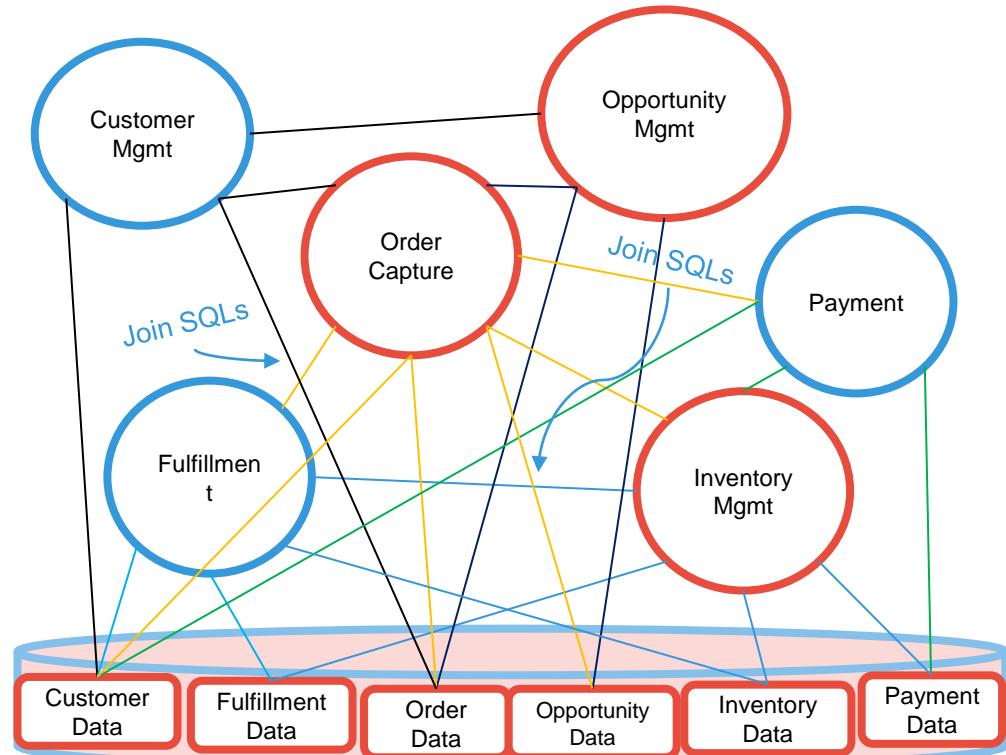
상호 데이터 참조가 용이하여 빨리 개발하기 위한 초기 아키텍처로 적합

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

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

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

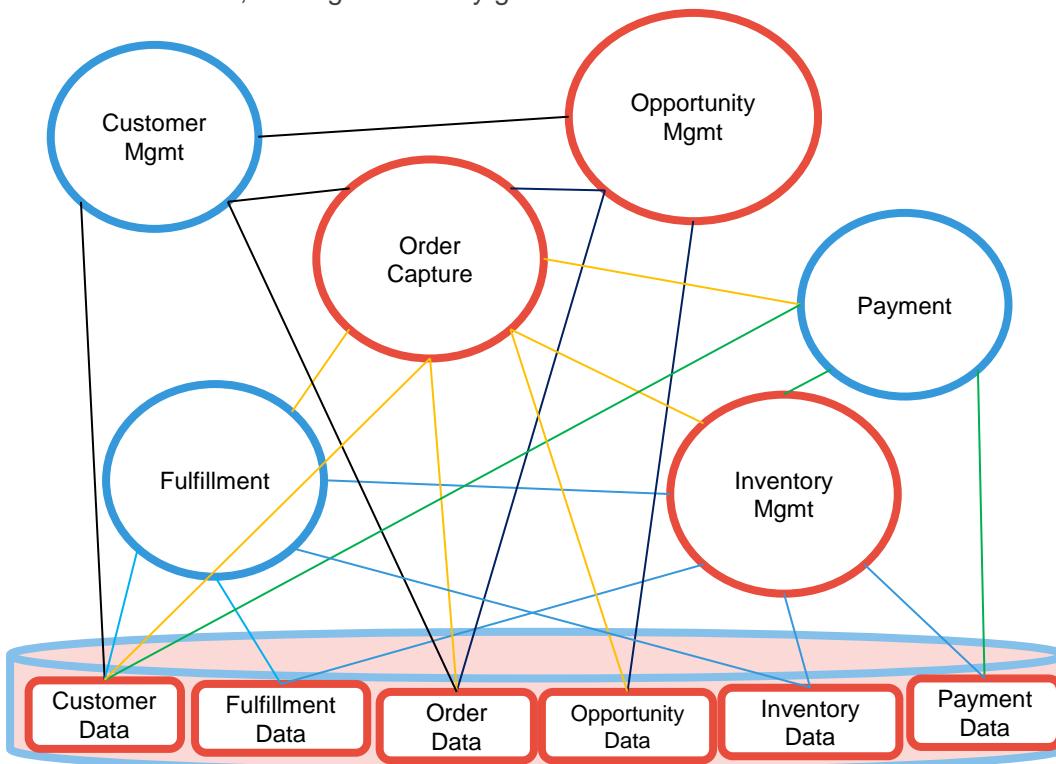
많은 다른 컴포넌트의 수정을 동반하게 됨
e.g. 다른 모듈의 Join SQL 들



모노리틱 아키텍처의 단점

Drawbacks of a Monolithic Architecture

Size matters, costs grow as they grow



코드량이 방대함 (오류발생시 바닷가에서 바늘 찾기)
Large code base

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

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

선별적 확장이 용이하지 않음
Only scales in one dimension

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

새로운 개발팀을 추가하는데 어려움이 있음
Becomes an obstacle to scaling development

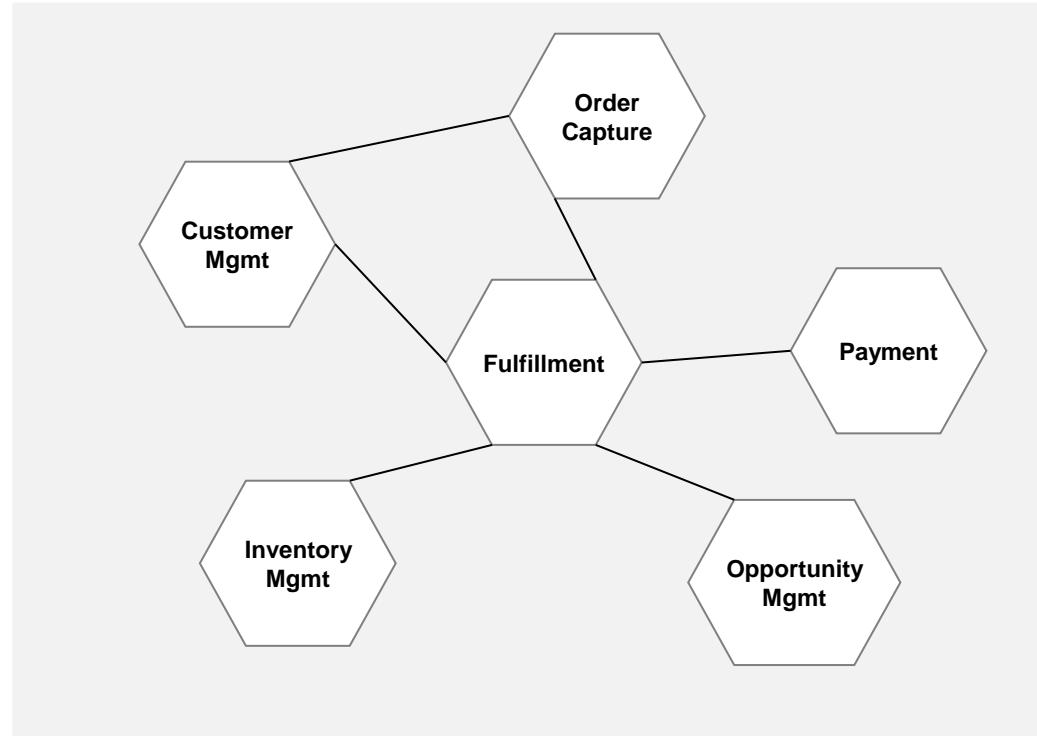
반면: A Microservice Architecture

Service-oriented architecture of loosely coupled elements with bounded contexts

모든 기능을 각각의 배포 스택으로
분리

Break each function into separate
deployment stacks

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



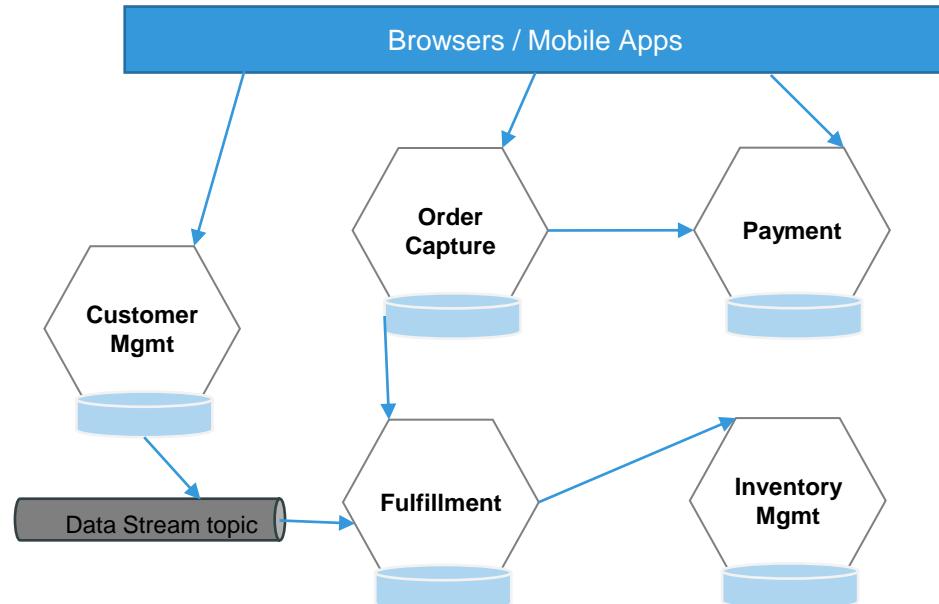
Microservice Architecture

Isolation is the name of the game

Service-oriented architecture

Loosely coupled elements

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



Principles of Microservices

Isolation is the name of the game

독립성과 자치성을 코드의 재사용성 보다 높게 본다

Independence and autonomy are more important than code re-usability

マイクロ 서비스는 코드와 데이터를 공유하지 않는다

Microservices should not share code or data

불필요한 서비스와 소프트웨어 컴포넌트(라이브러리) 간의 커플링을 피한다

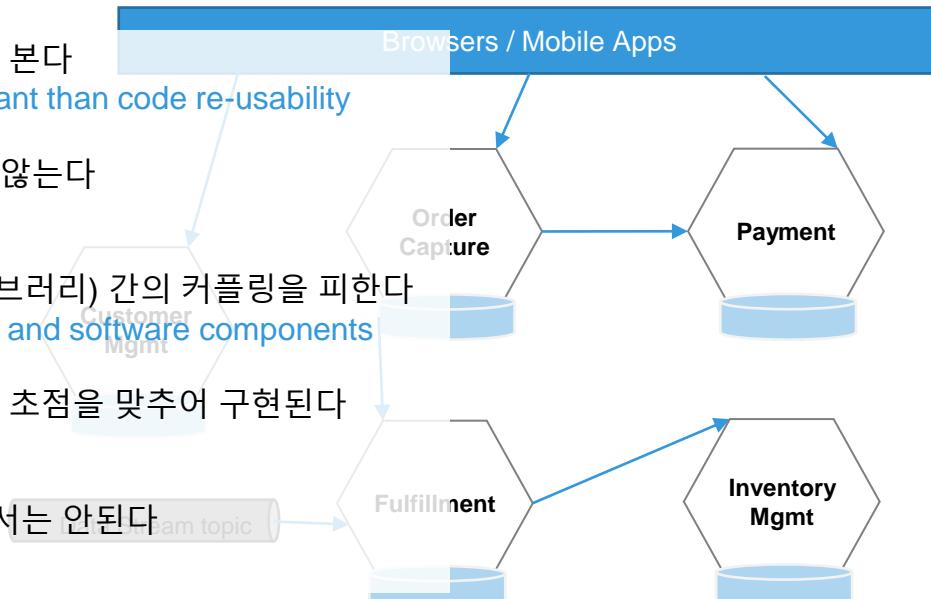
Avoid unnecessary coupling between services and software components

각 마이크로서비스는 각자의 단 하나의 기능에 초점을 맞추어 구현된다

Single responsibility

운영시에는 SPOF (단일 실패 지점) 을 만들어서는 안된다

There should be no single point of failure



HTTP/REST 를 이용한 약결합 연동 (Loosely-Coupled Interaction via HTTP/REST)

REST APIs must be stable and hide internals

클라이언트-서버 REST API 는 내부구현을 숨길 수 있으면서 연동

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

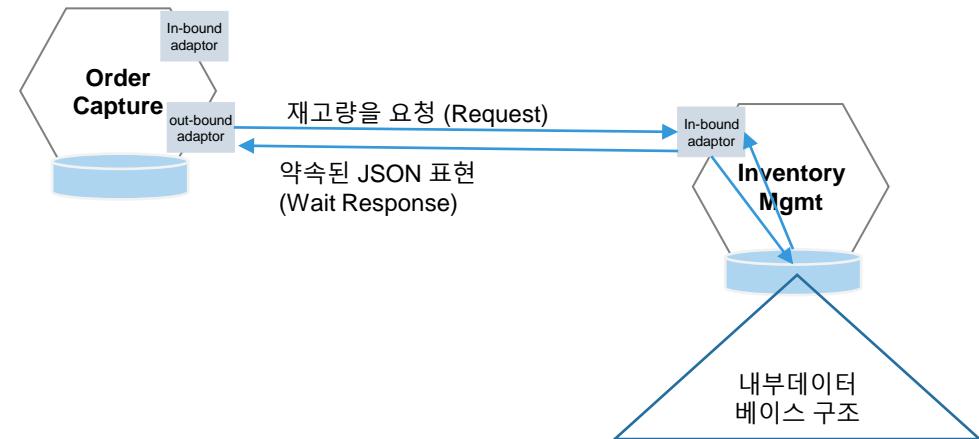
동기식 연동 (Synchronous, Request-Response)

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

API 는 하위호환성을 유지하기 위하여 추가적인 수정만을 허용

Only additive changes allowed

➔ 빌드타임에서만 약결합을 제공함



비동기 메시징을 통한 약결합 연동

Loosely-Coupled Interaction via Asynchronous Messaging

Data streaming can decouple database with shared data

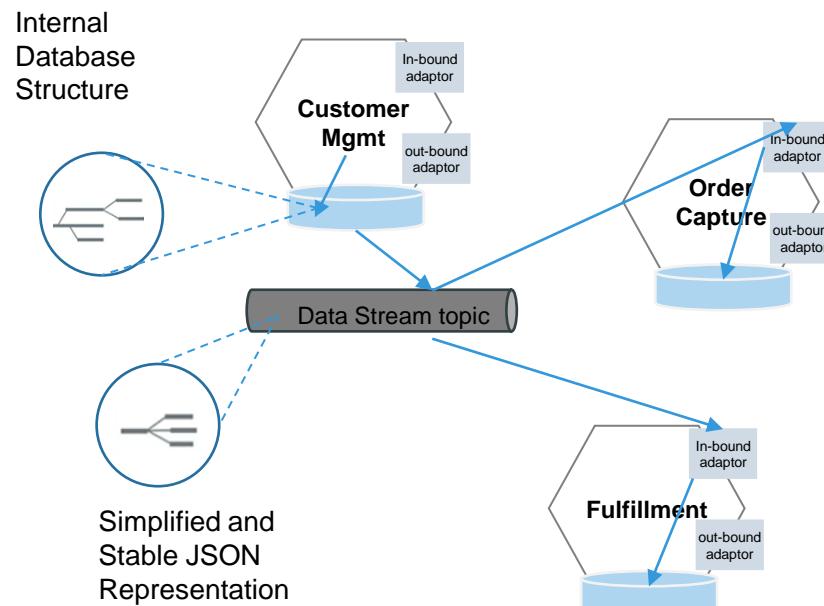
상대의 수신을 기다리지 않는 비동기식 메시징
Asynchronous messaging(streaming)

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

Data streams hide the internal implementation of the service

Client ignore parts of the stream they don't understand

→ 런타임에서도 약결합을 제공함



Microservice Architecture benefits

Smaller is better

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

Smaller, independent code base is easier to understand

수정된 서비스에 대한 국지적 단위 디플로이와 테스팅이 용이해진다

Simpler deployment and testing of just the changed service

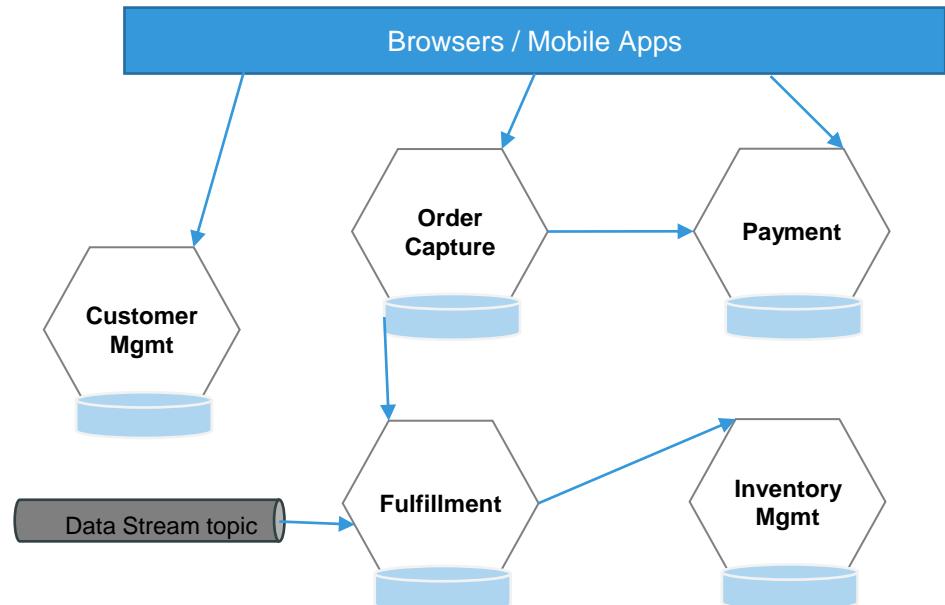
장애격리가 좋아진다

Improved fault isolation

여러가지 혼재된 기술 스택 (신기술)을 사용하기 쉽다
Not committed to one technology stack

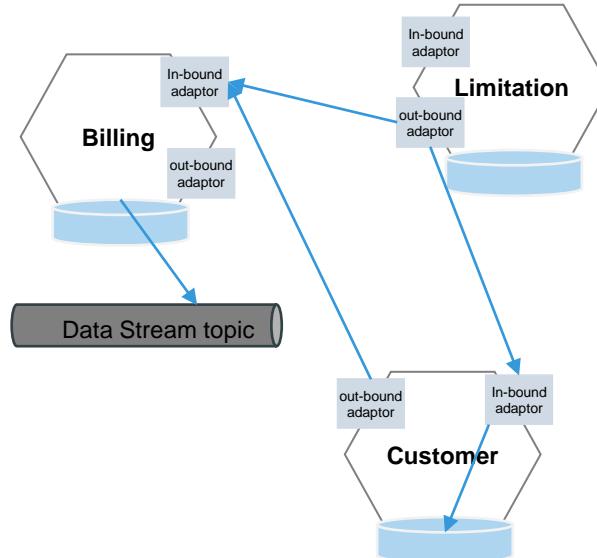
개발환경을 구축하기 쉽고 빨리 기동된다

IDE and app startup are faster



Microservice Architecture

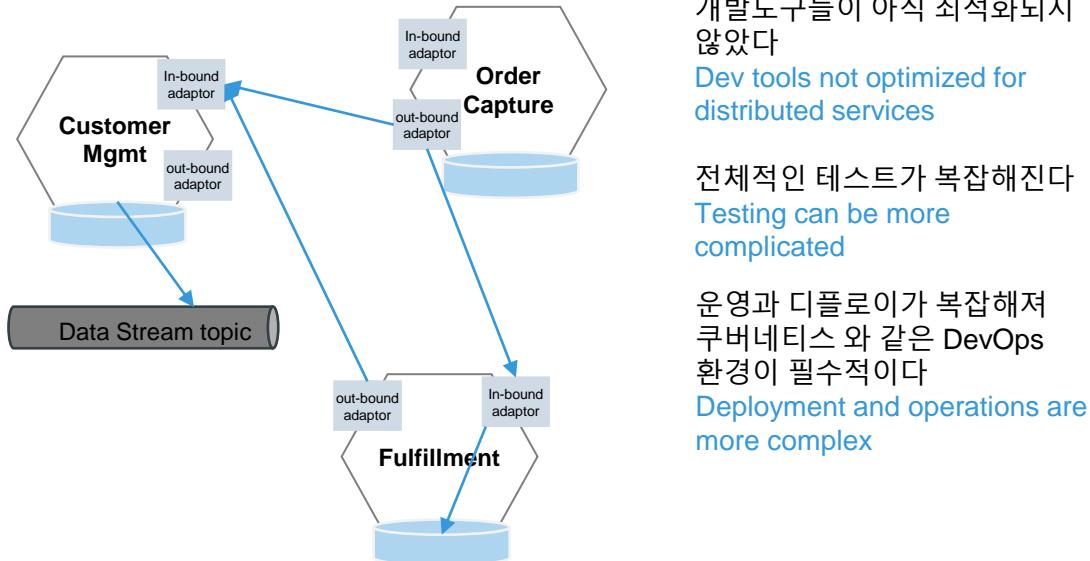
- Updating one service doesn't require changing others
- Ability to upgrade the tech stack (HW/SW/DBMS/NW) Of each service independently



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

Drawbacks to a Microservice Architecture

Distributed computing adds complexity and slow down initial development



개발 도구들이 아직 최적화되지 않았다
Dev tools not optimized for distributed services

전체적인 테스트가 복잡해진다
Testing can be more complicated

운영과 디플로이가 복잡해져 쿠버네티스와 같은 DevOps 환경이 필수적이다
Deployment and operations are more complex

서비스를 어떻게 쪼갤 것인가?
Where/How to decompose the services?

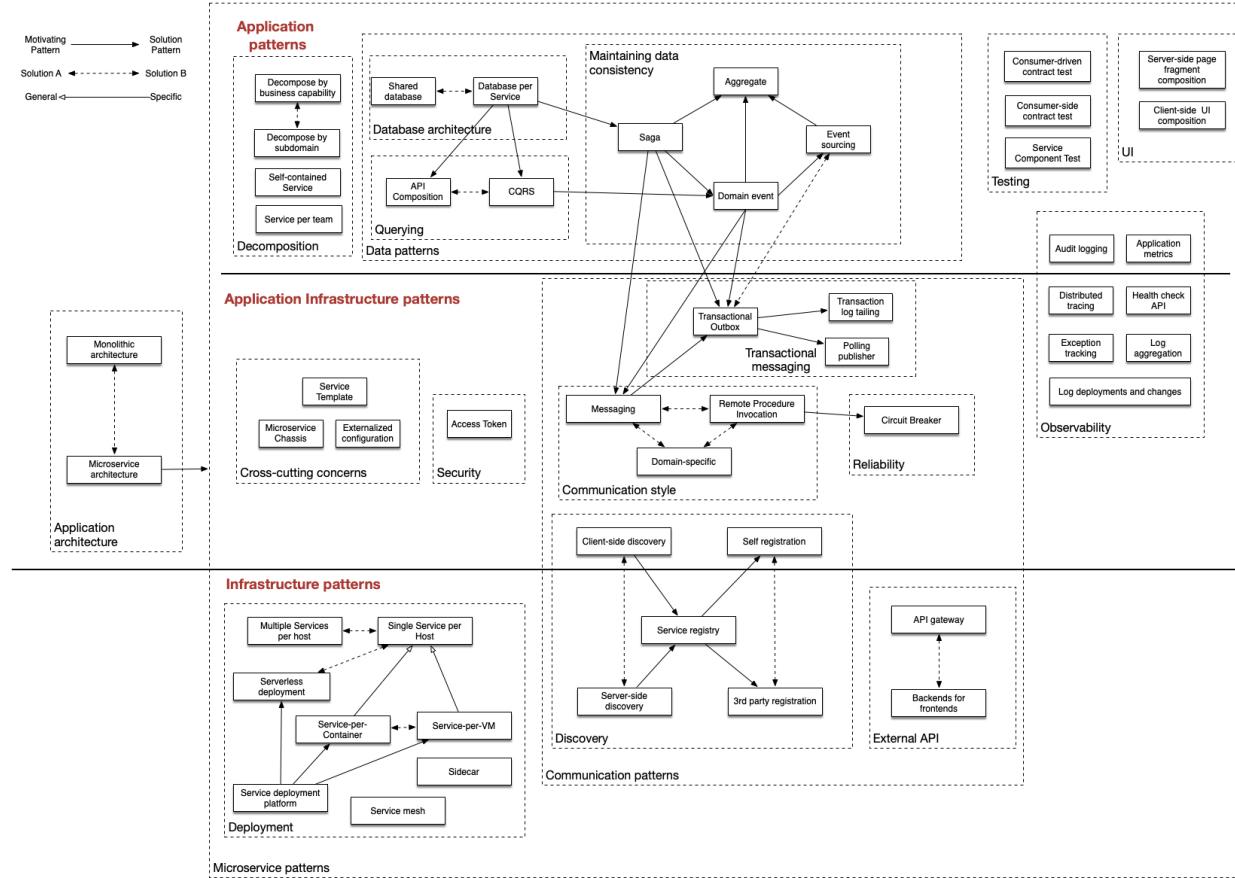
서비스 간 연동을 통해서만 구현하는 비용의 상승
Inter-service communication complicates development

분산 트랜잭션 (데이터 일관성 등)을 어떻게 보장할 것인가?
Maintaining consistency with distributed transactions is hard

서비스 개수가 많아진 상황의 보안 처리를 어떻게 할 것인가?
What about inter-service security?
Identity management?

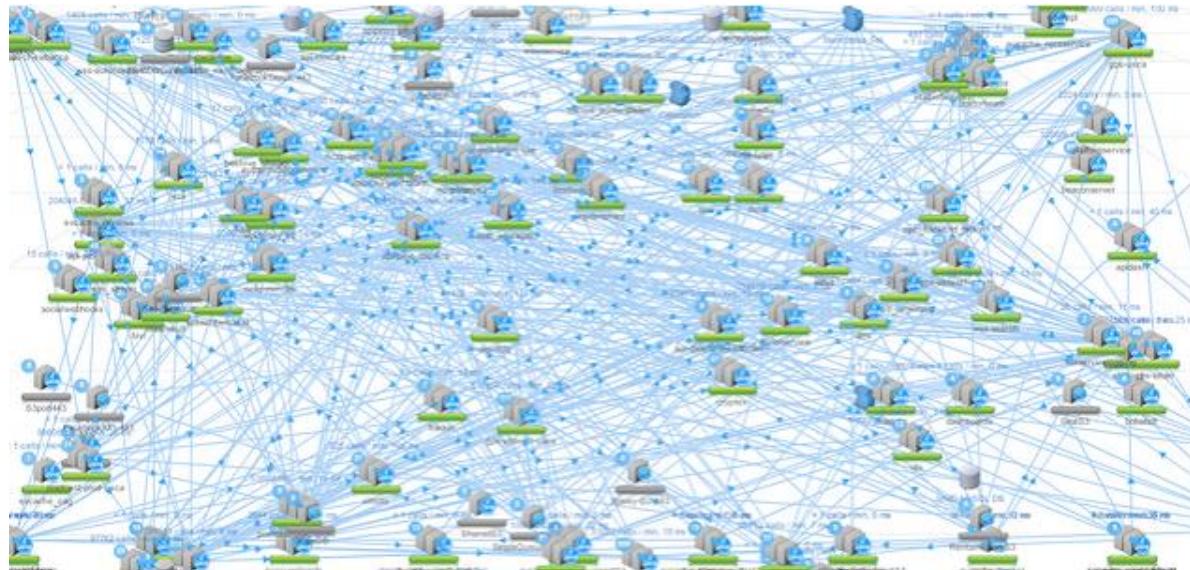
해결책: MSA 디자인 패턴

- microservices.io



넷플릭스 OSS

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



マイクロ サービス 轉換 사례



Mobile Apps and
Serverless Microservices



Pure Play Video OTT- A



Video & Broadcasting



From Monolithic to
Microservices



Gaming Platform



IoT Service

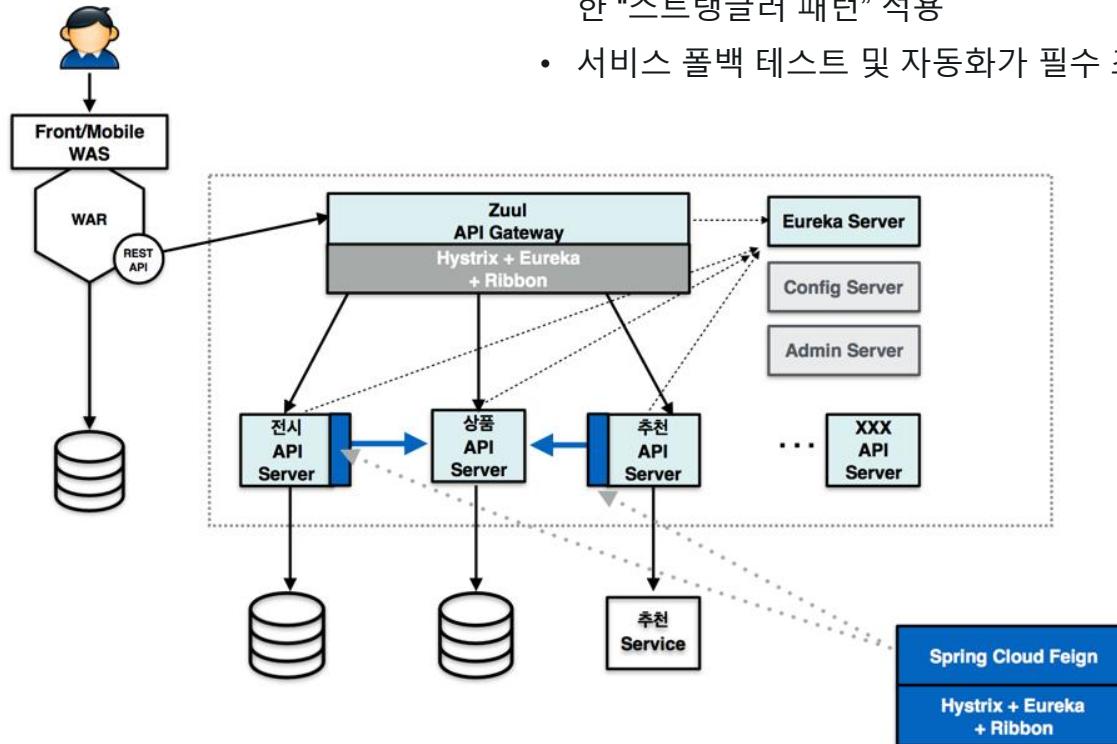


Serverless Microservices

당근마켓

<https://youtu.be/mLithm96u2Q?t=50>

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





- 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

All	Android	Microservices	Mingle	NPM-vingle-Packages
S	W	Name ↓		
		Microservice-action-reflector		
		Microservice-business-alert		
		Microservice-color-extractor		
		Microservice-dynamodb-stream-processor		
		Microservice-lambda-microservice-template		
		Microservice-marketing-bot		
		Microservice-search-interface		
		Microservice-spam-checker		
		Microservice-TrackTicket		
		Microservice-vingle-ads		
		Microservice-vingle-feed		
		Microservice-wingle		

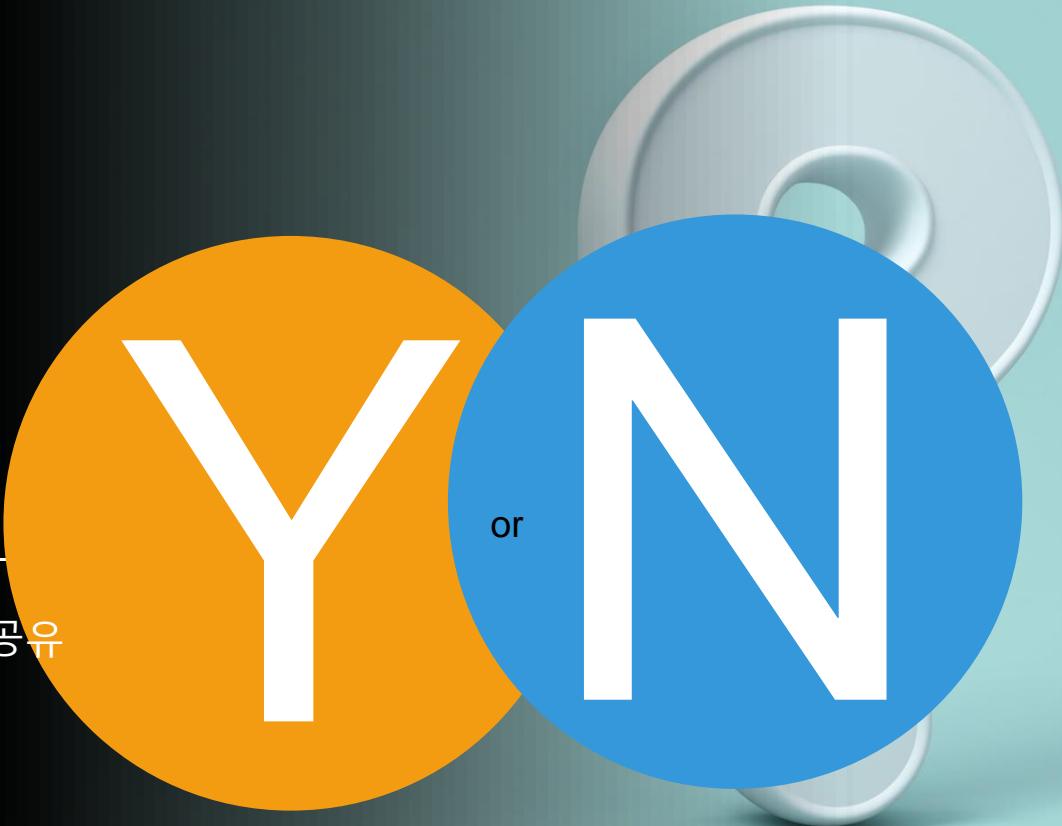
Tip: 10 Attributes of Cloud Native Applications

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/>

Quiz

마이크로서비스간에 코드를 공유
하는 것은 좋은 사례이다



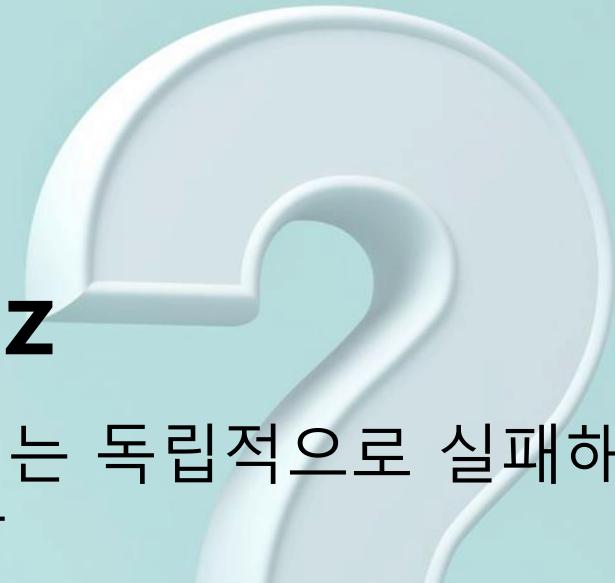
Quiz

마이크로서비스가 실패할 때는 독립적으로 실패해야 한다

Y

or

N





Quiz

- 왜 단일 Database 접근은 마이크로서비스에서 안티-패턴 (하지 말아야 할 접근) 인가?
- 1. 마이크로서비스는 관계형 데이터베이스 (R-DBMS) 와 호환되지 않기 때문이다
- 2. 하나의 데이터베이스만 사용했을 때는 이것이 실 패단일지점 (Single Point of Failure)가 될 수 있기 때문이다
- 3. 단일 데이터베이스 시스템은 보통 큰 시스템을 만들 때만 쓰이기 때문이다.

Table of content

Microservice and
Event-storming-Based
DevOps Project

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

Target Domain

: Online Shopping Mall (12 STREET)

- Vision & Mission



Service Resiliency

24시간 365일 접속과 주문이 가능
: 자동화된 회복, 장애전파최소, 무정지 재배포



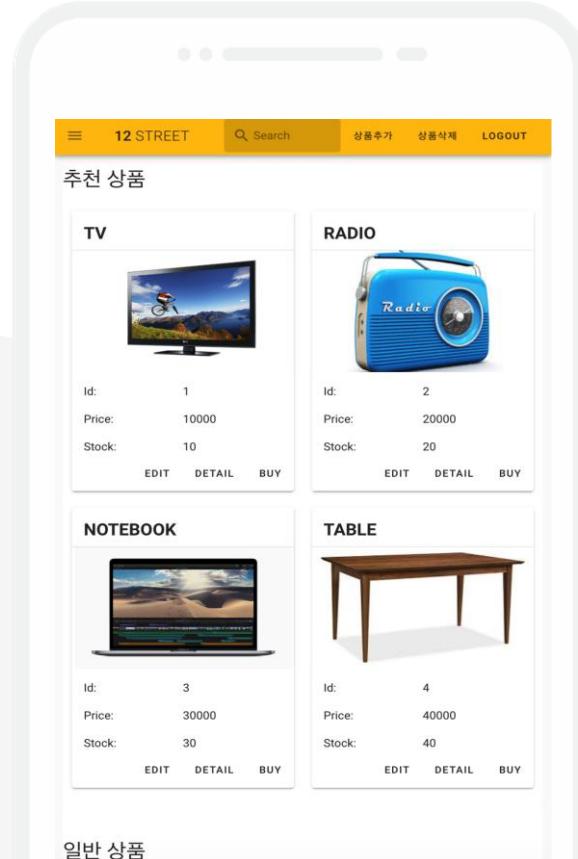
Customer Responsiveness

다양한 고객 기능 요구사항의 탐색과 반영



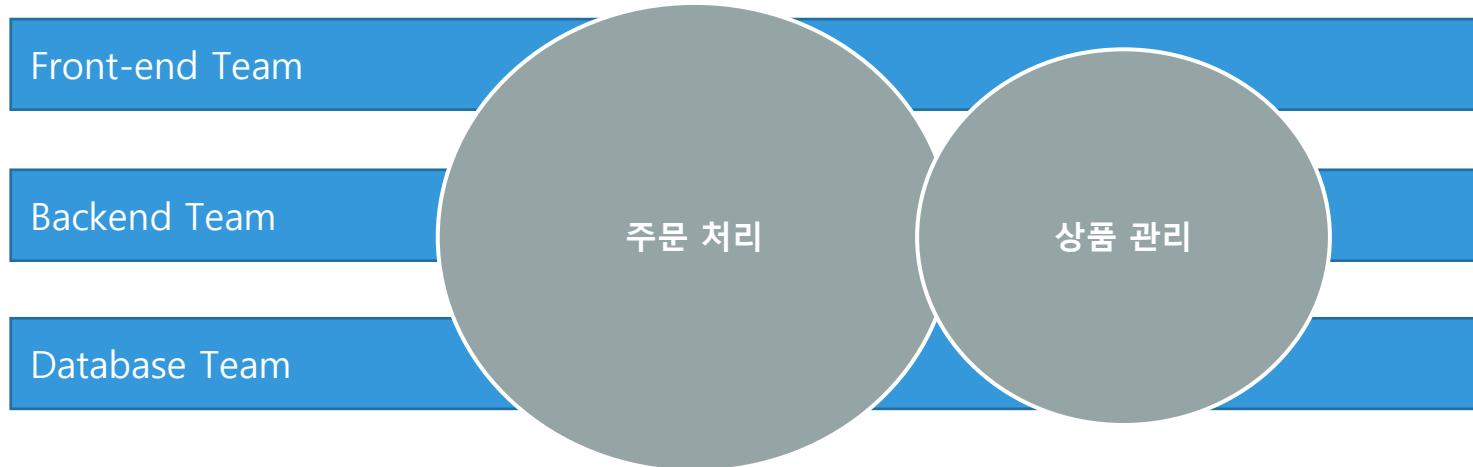
Scalability

조직, 기능 및 데이터의 확장에 열려 있는 아키텍처
: Feature-driven-development



Organization & KPI Definition - 창업시기

12 Street Team

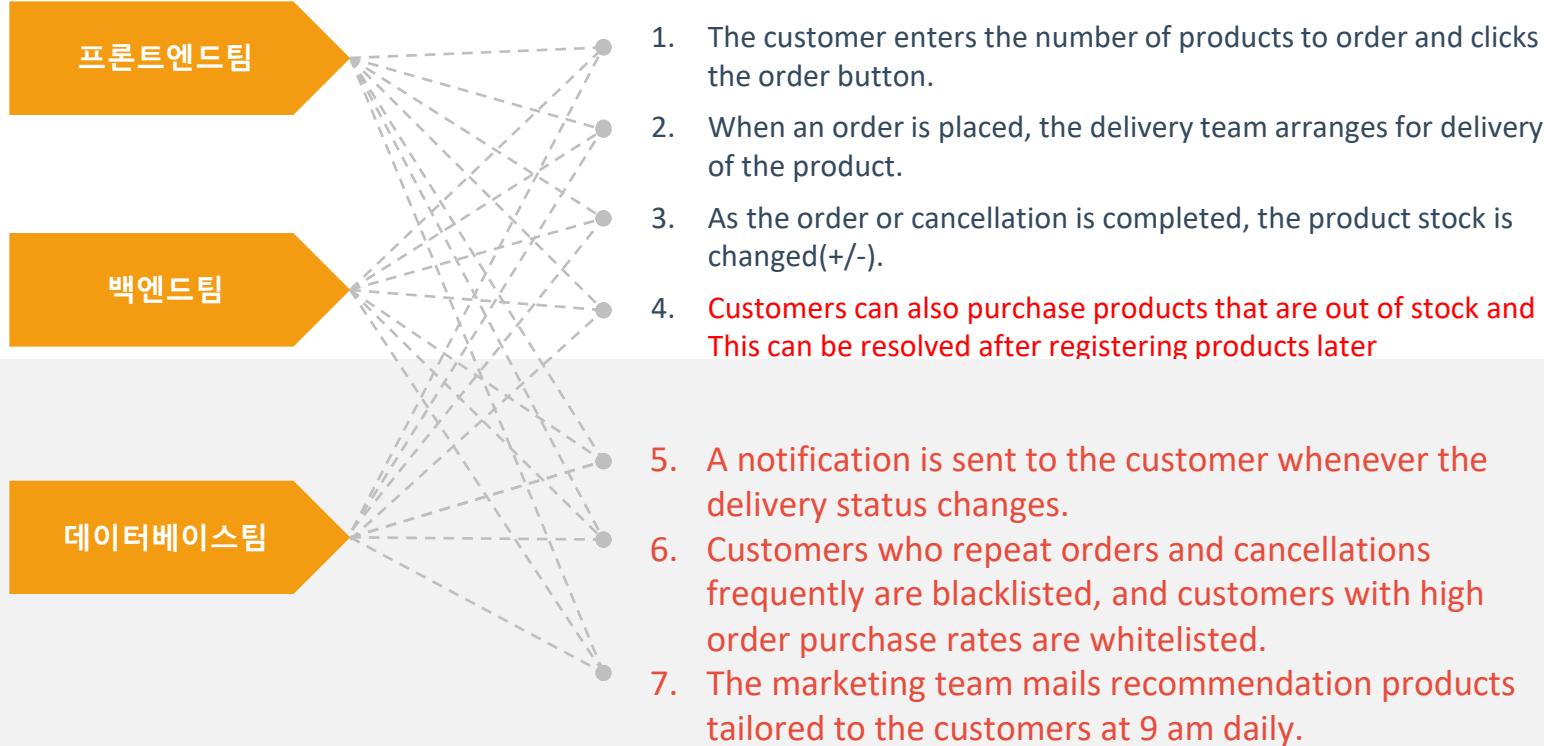


- Multiple Concerns Single Organization → no problem
- Horizontally aligned

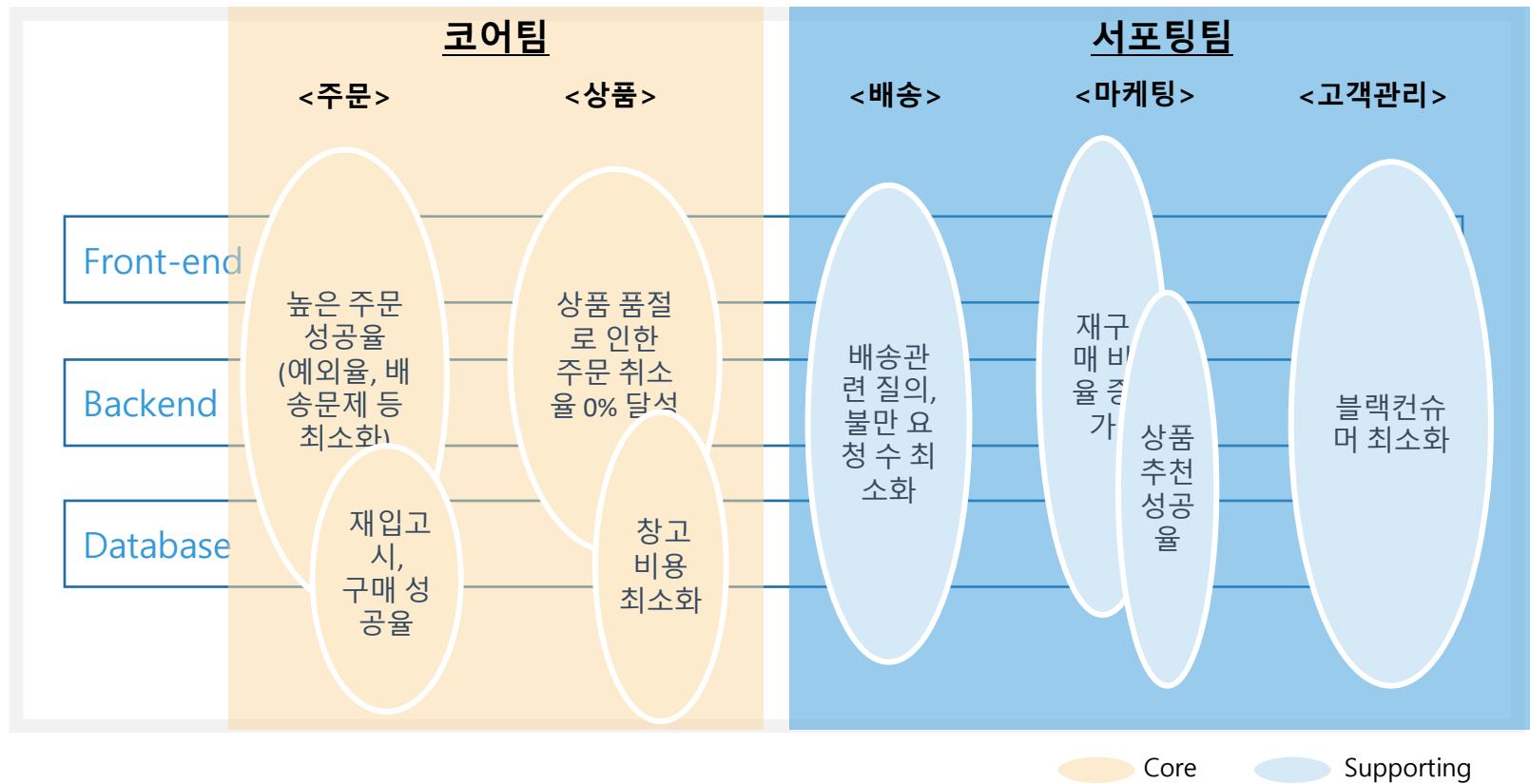
회사의 성장 – 너무 많은 Concerns



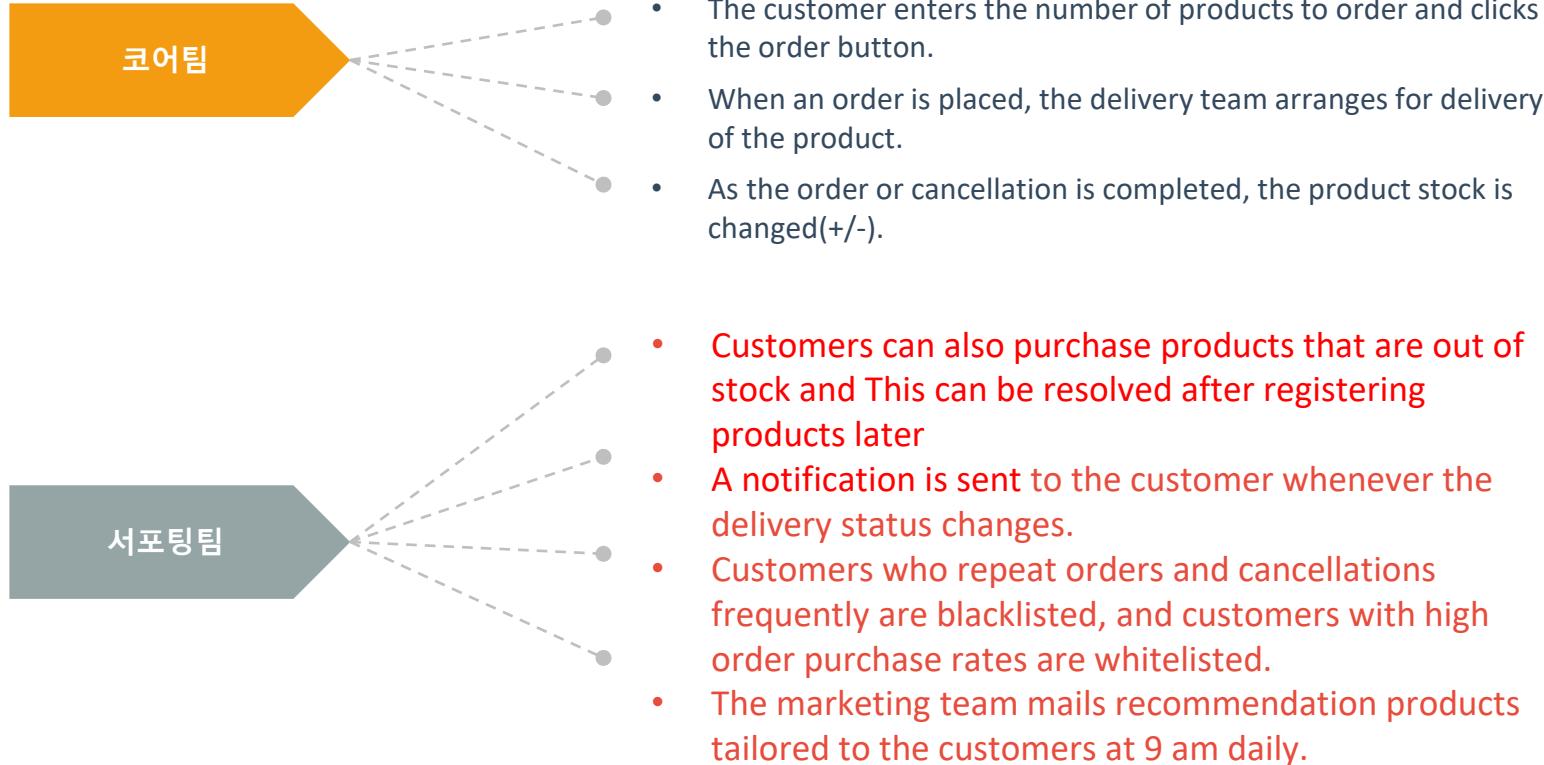
User Stories 와 책임소재 → 너무 많은 Concerns



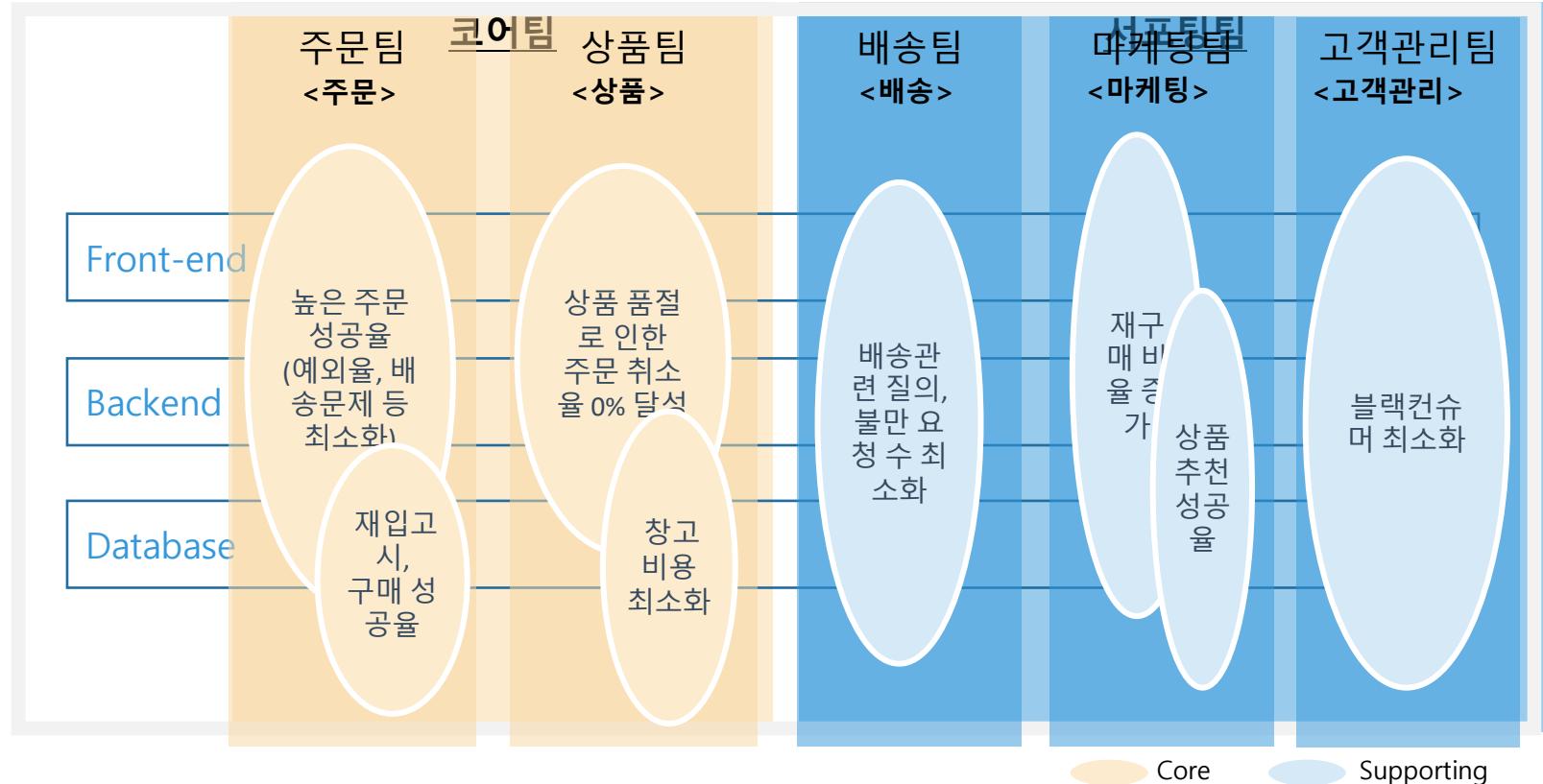
회사의 성장 - Separation of Concerns



책임소재 - 중요한 것과 덜 중요한 것의 분리



회사의 성장 – 자치성과 동기부여



책임소재 – 자치성과 동기부여



- The customer enters the number of products to order and clicks the order button.
- When an order is placed, the delivery team arranges for delivery of the product.
- As the order or cancellation is completed, the product stock is changed(+/-).
- Customers can also purchase products that are out of stock and This can be resolved after registering products later
- A notification is sent to the customer whenever the delivery status changes.
- Customers who repeat orders and cancellations frequently are blacklisted, and customers with high order purchase rates are whitelisted.
- The marketing team mails recommendation products tailored to the customers at 9 am daily.

Table of content

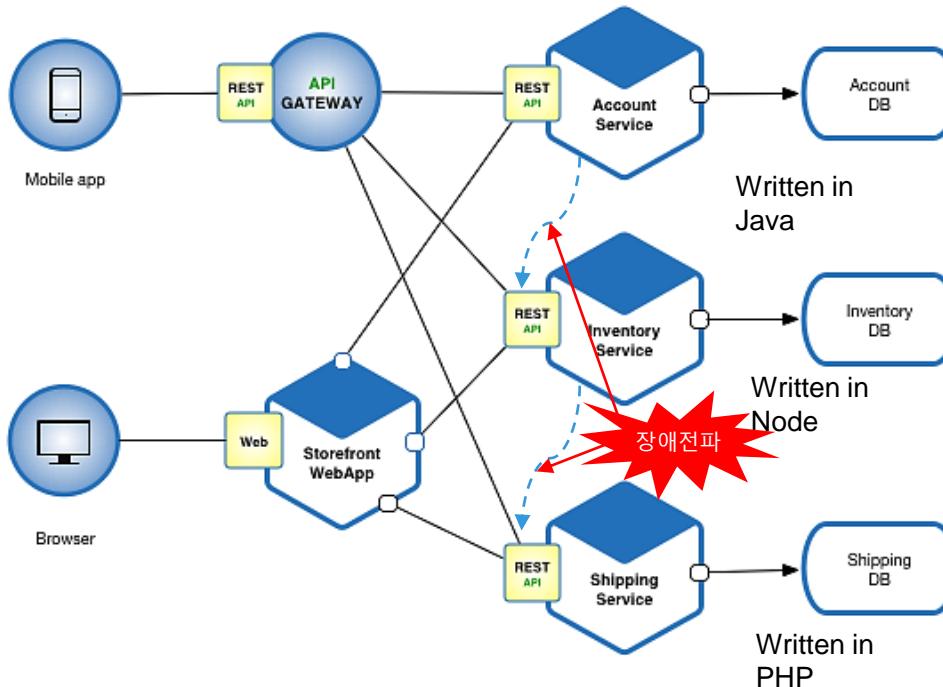
Microservice and
Event-storming-Based
DevOps Project

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

Approach #1 : Micro Service Architecture

Contract based, Polyglot Programming

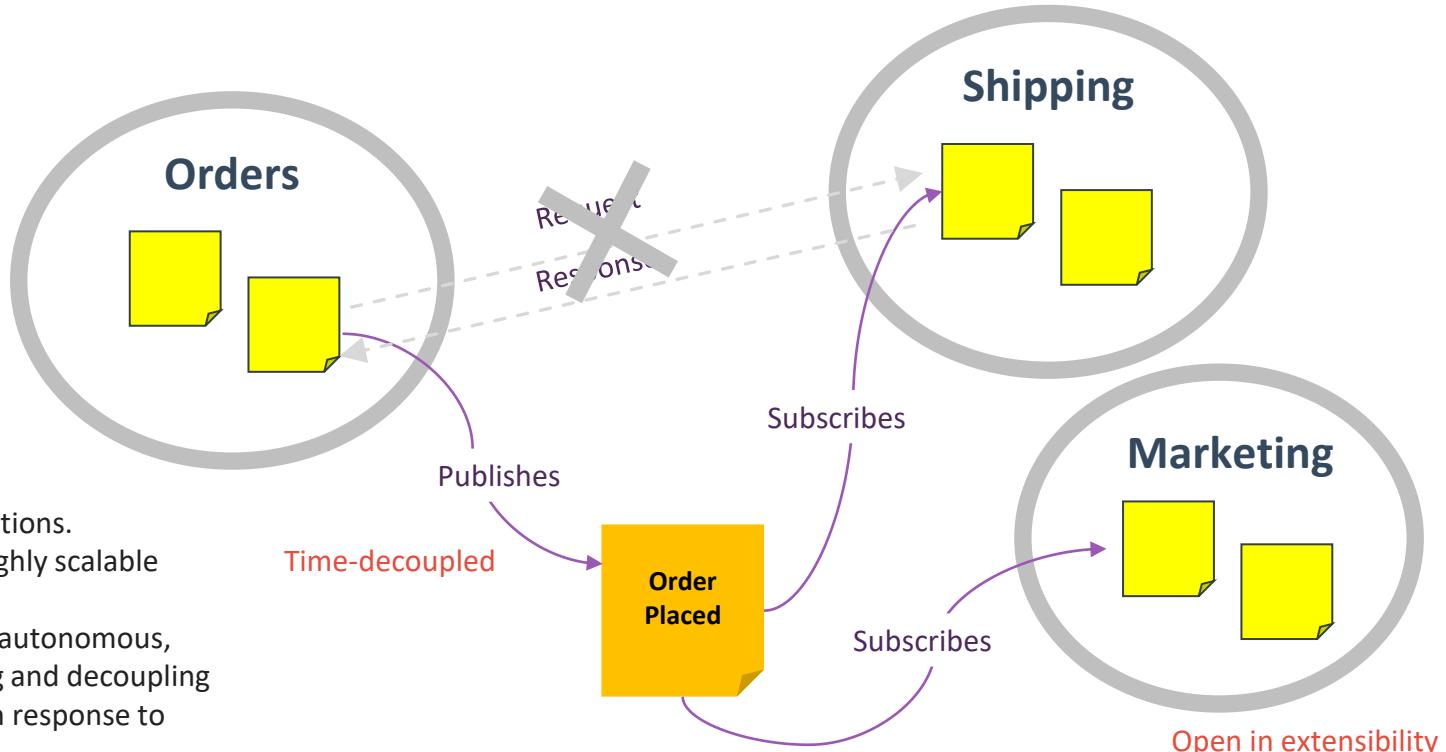
- Separation of Concerns, Parallel Development, Easy Outsourcing



[Limitation]

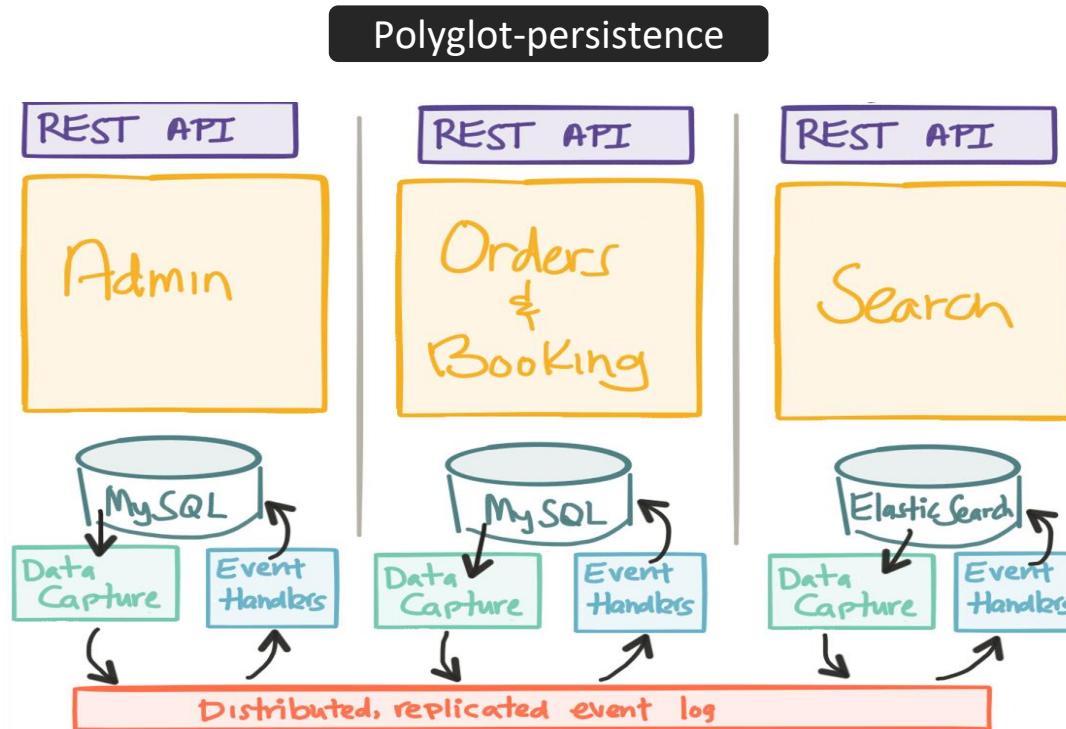
1. Code Coupling 해소
But, Time Coupling 잔존
2. 서비스 Blocking 가능성 내재
3. 장애 전파 우려
4. Point to Point 연결에 따른 복잡한 스파게티 네트워크

Approach #2 : Event Driven Architecture

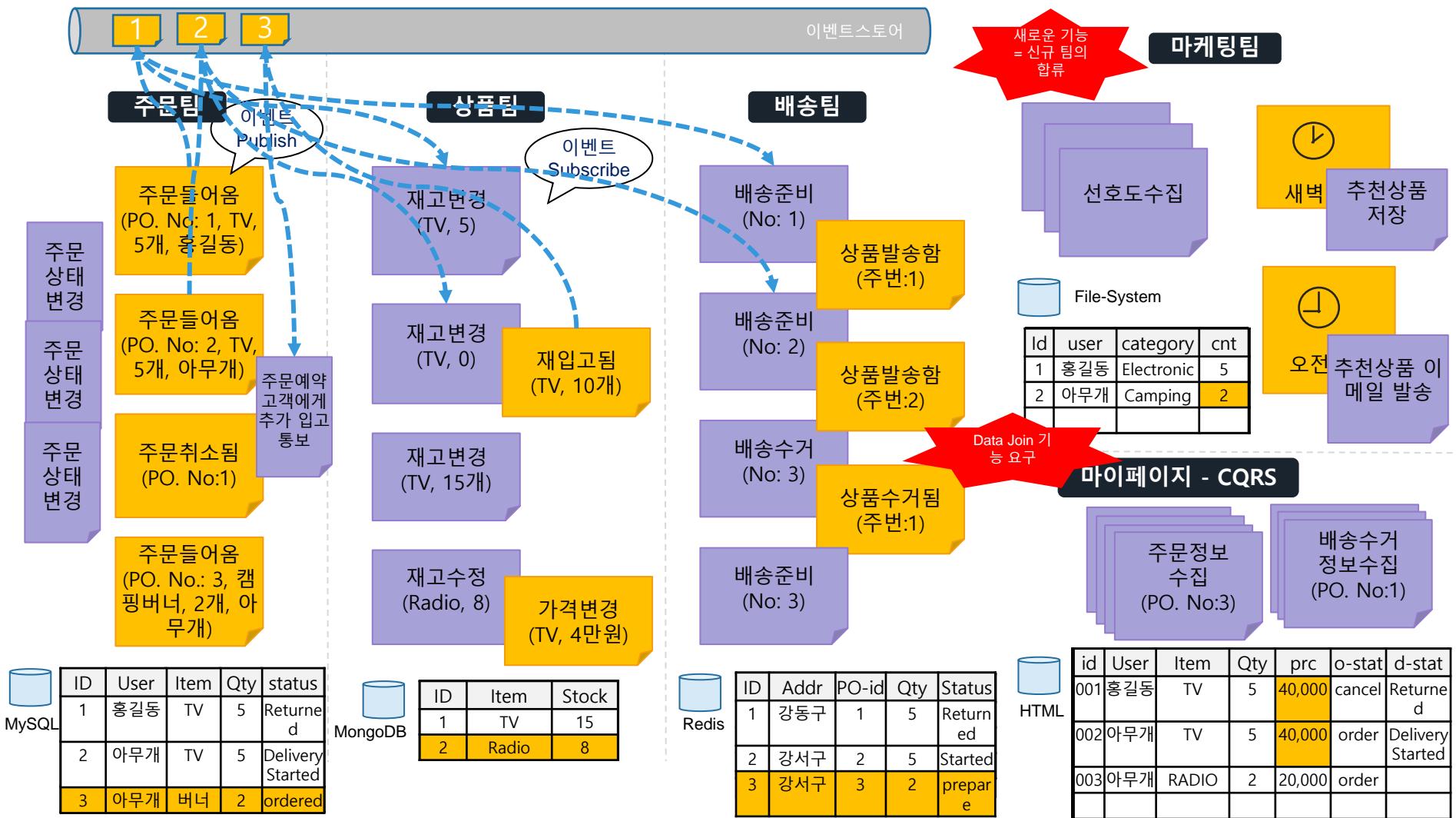


- No point-to-point integrations.
- High performance and highly scalable systems.
- Components can remain autonomous, being capable of coupling and decoupling into different networks in response to different events.
- Advanced analytics can be done using complex event processing.

Approach #2 : Event Driven Architecture



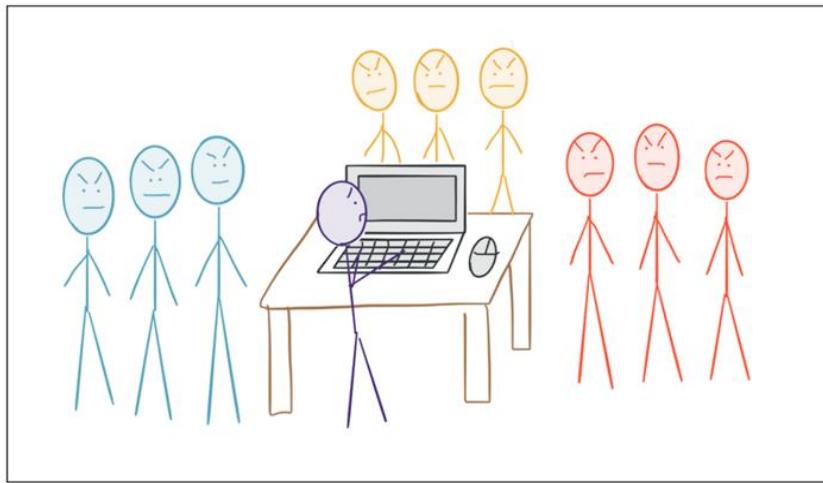
Source: <https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7>, Kevin webber



Benefits of Event-Sourcing?

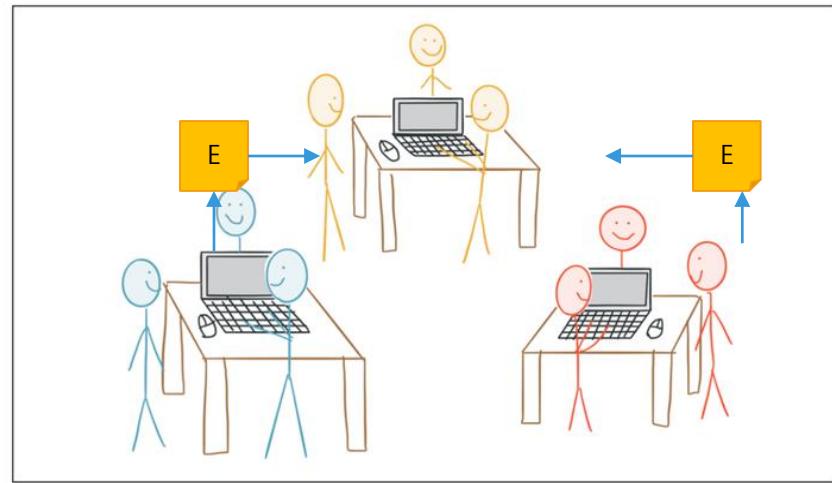
- The event store provides a complete log of every state change, effectively creating an audit trail of the entire system.
- The state of any object can be recreated by replaying the event store.
- It removes the need to map relational tables to objects.
- An event store can feed data into multiple read databases.
- In the case of failure, the event store can be used to restore read databases.

Event Driven MSA Architecture



Monolith:

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



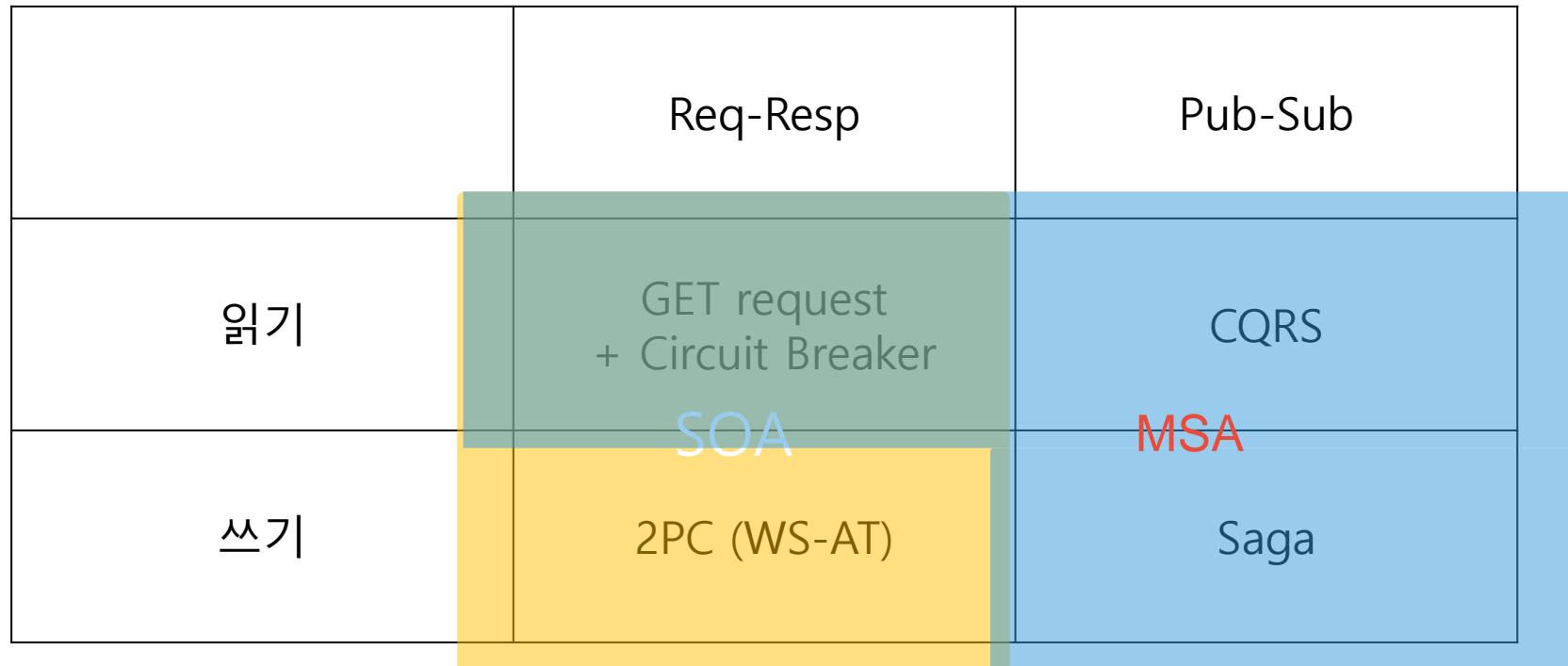
Reactive Microservices:

- Autonomously designed Model(Document)
- Polyglot Persistence

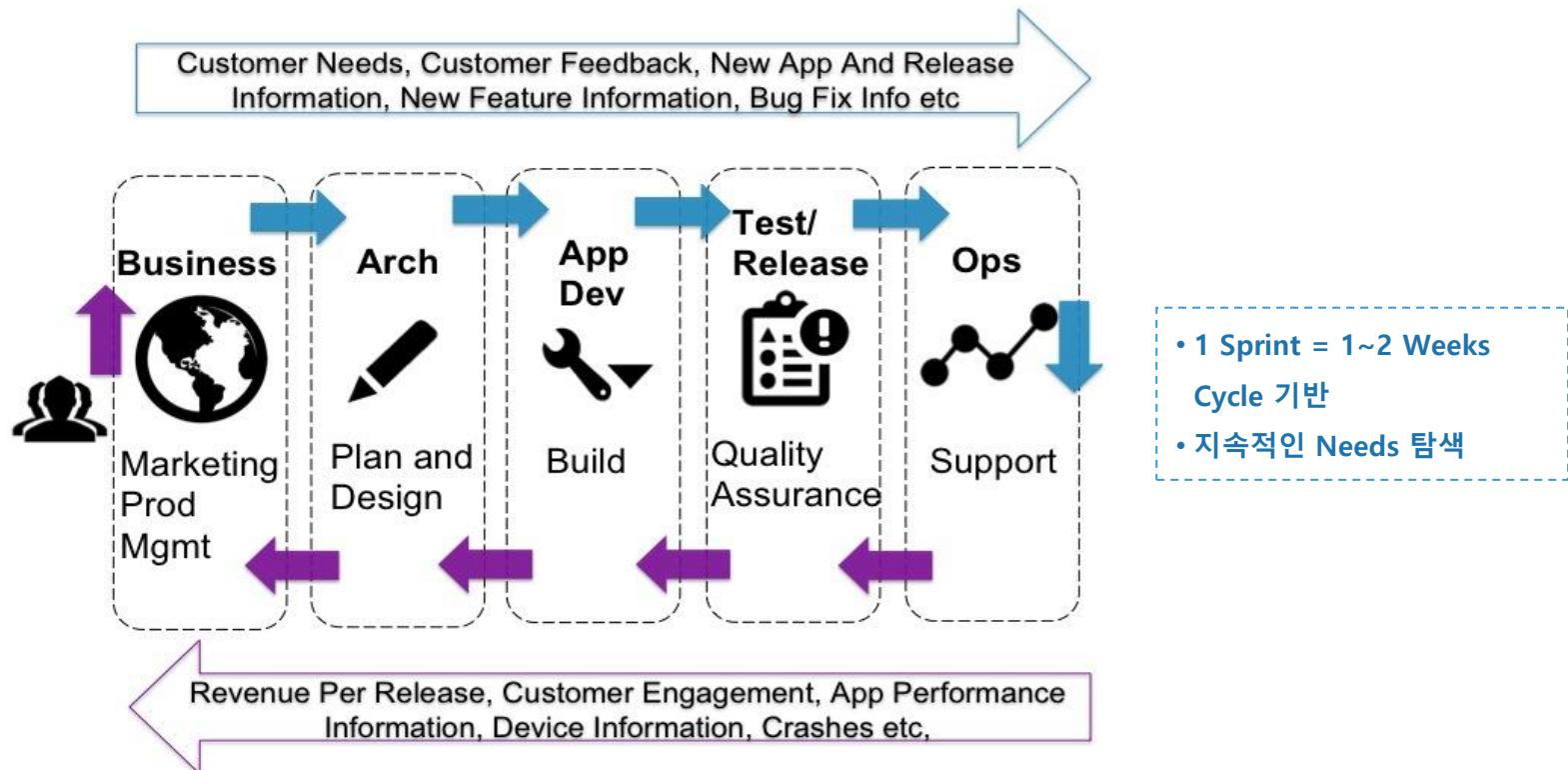
Case Study – 11번가

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

Integration Strategies Compared



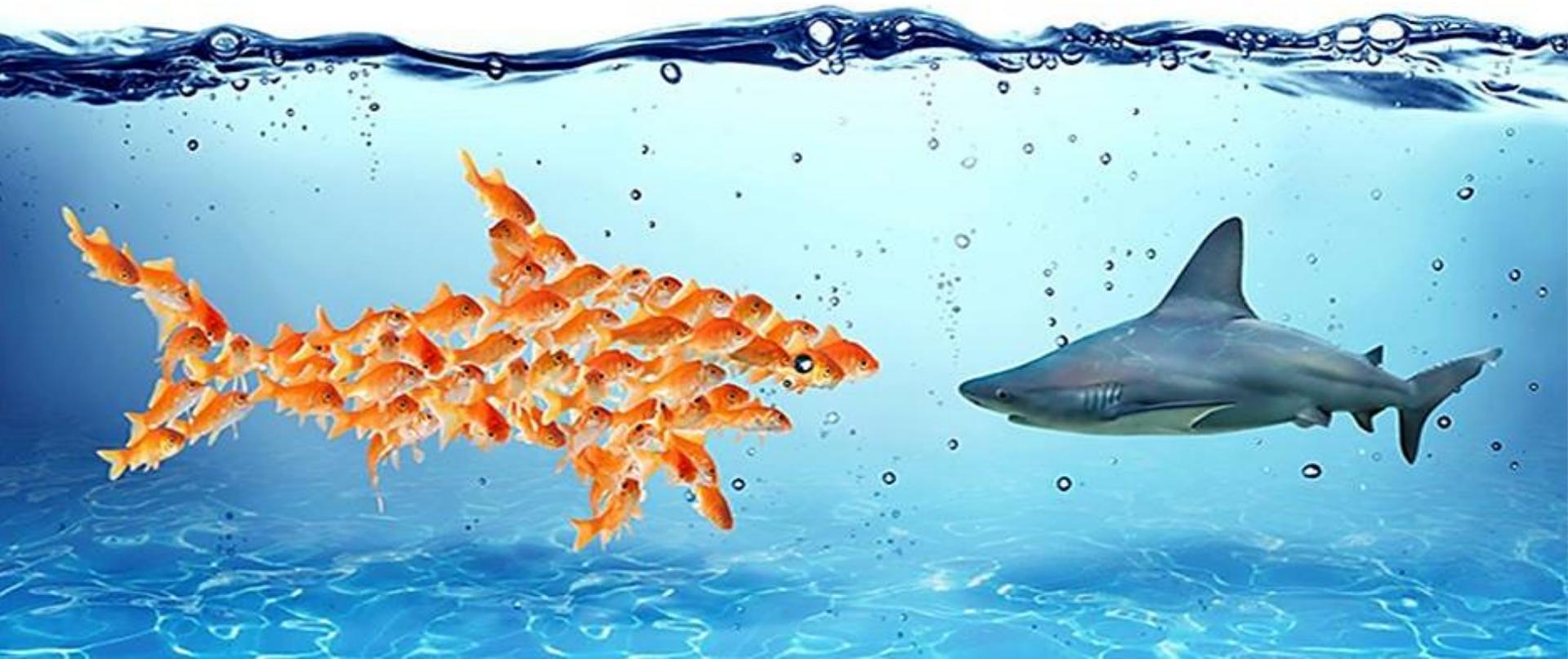
Approach #3: BizDevOps Process & Infra



“

서비스를 죽지 않게 하려면...

”



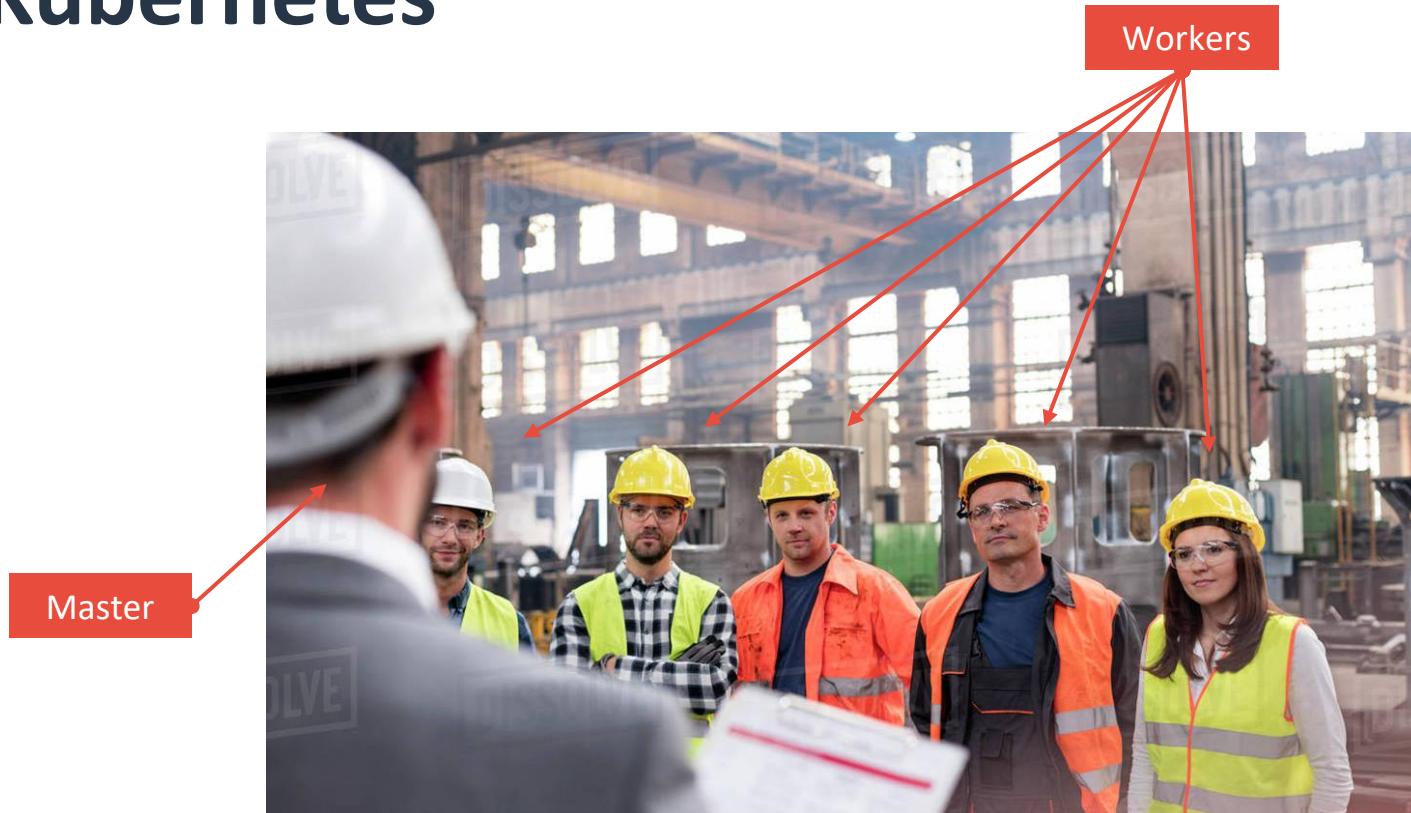
“

항상성이 (Self-Healing) 자동으로 유지되면 어떨까?

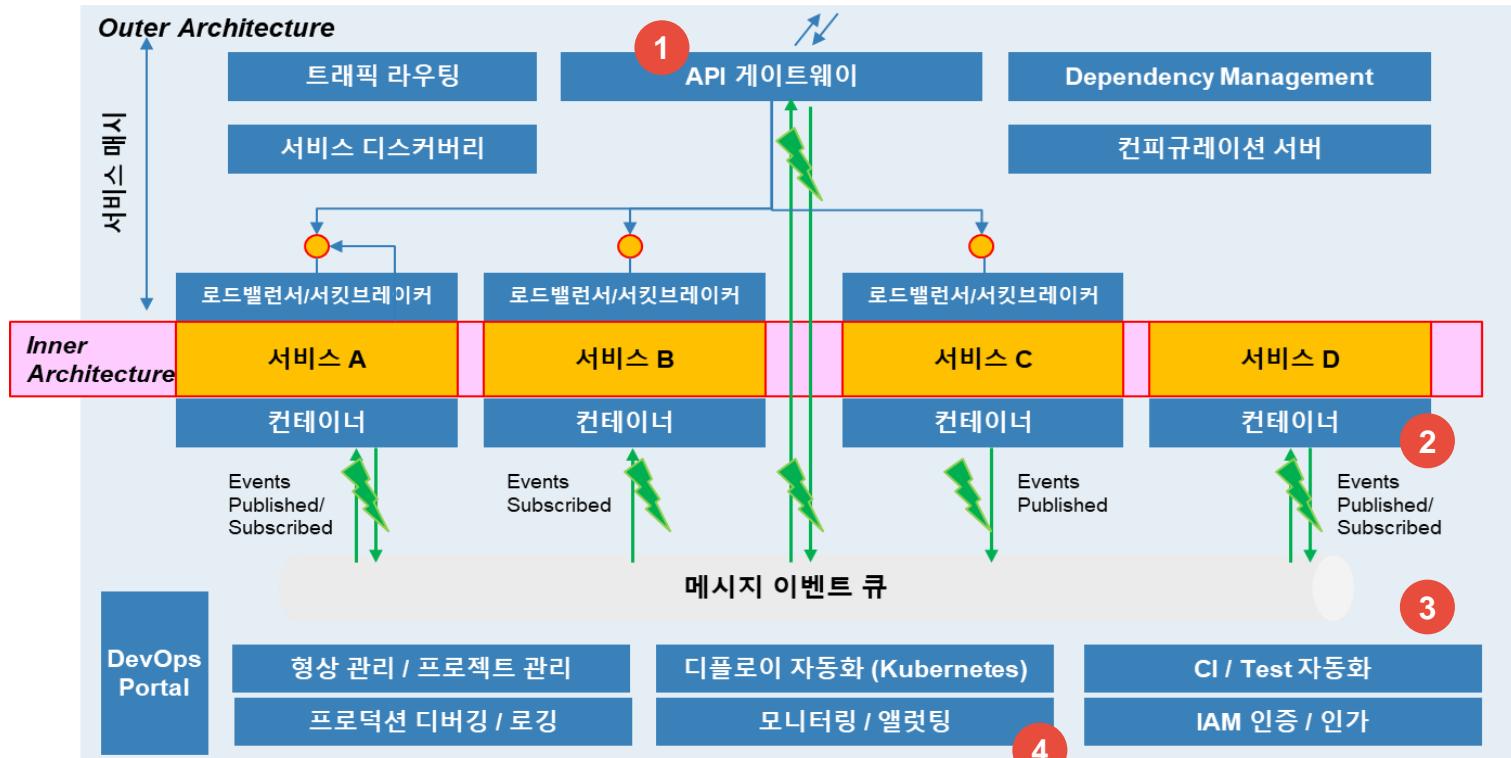
”



Kubernetes

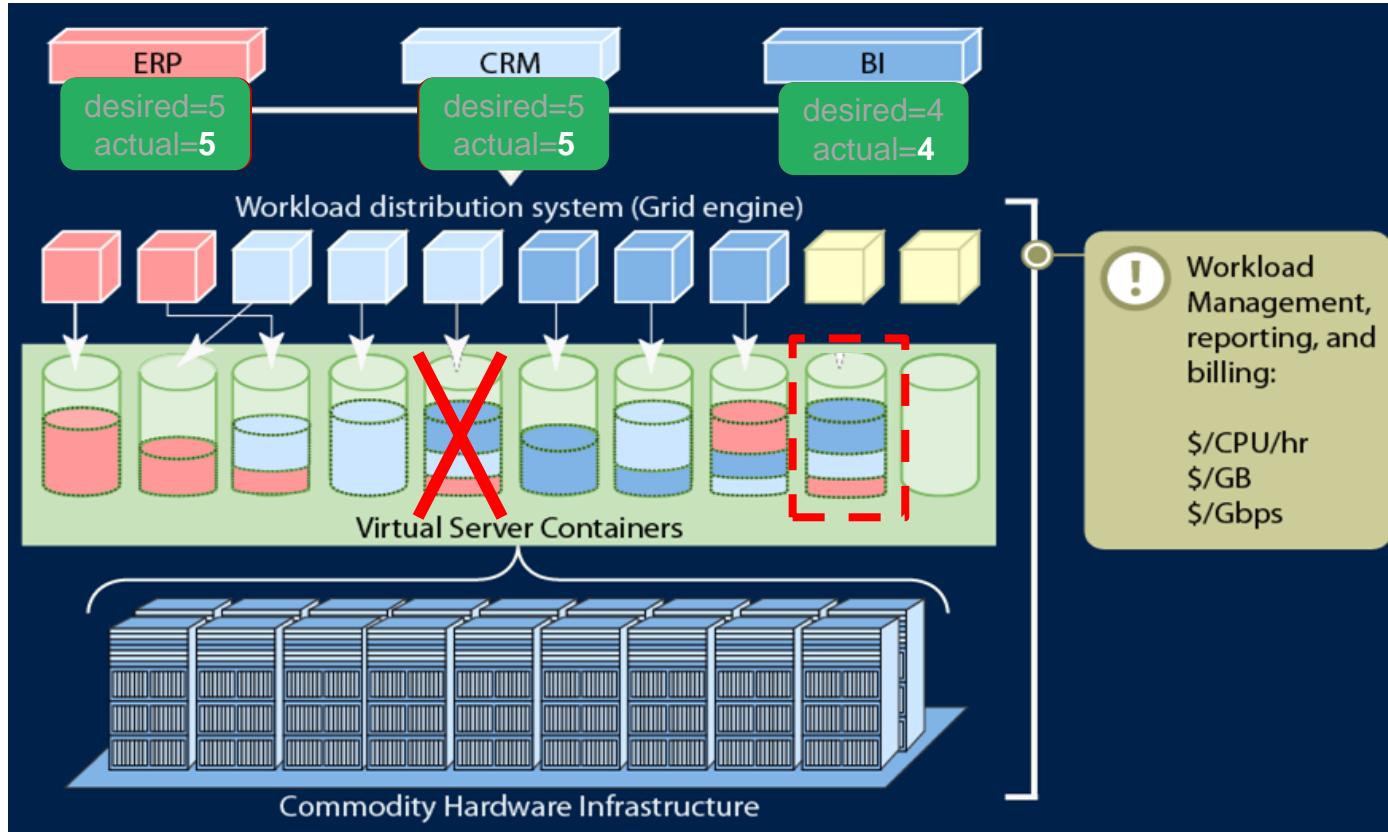


Outer & Inner Architecture

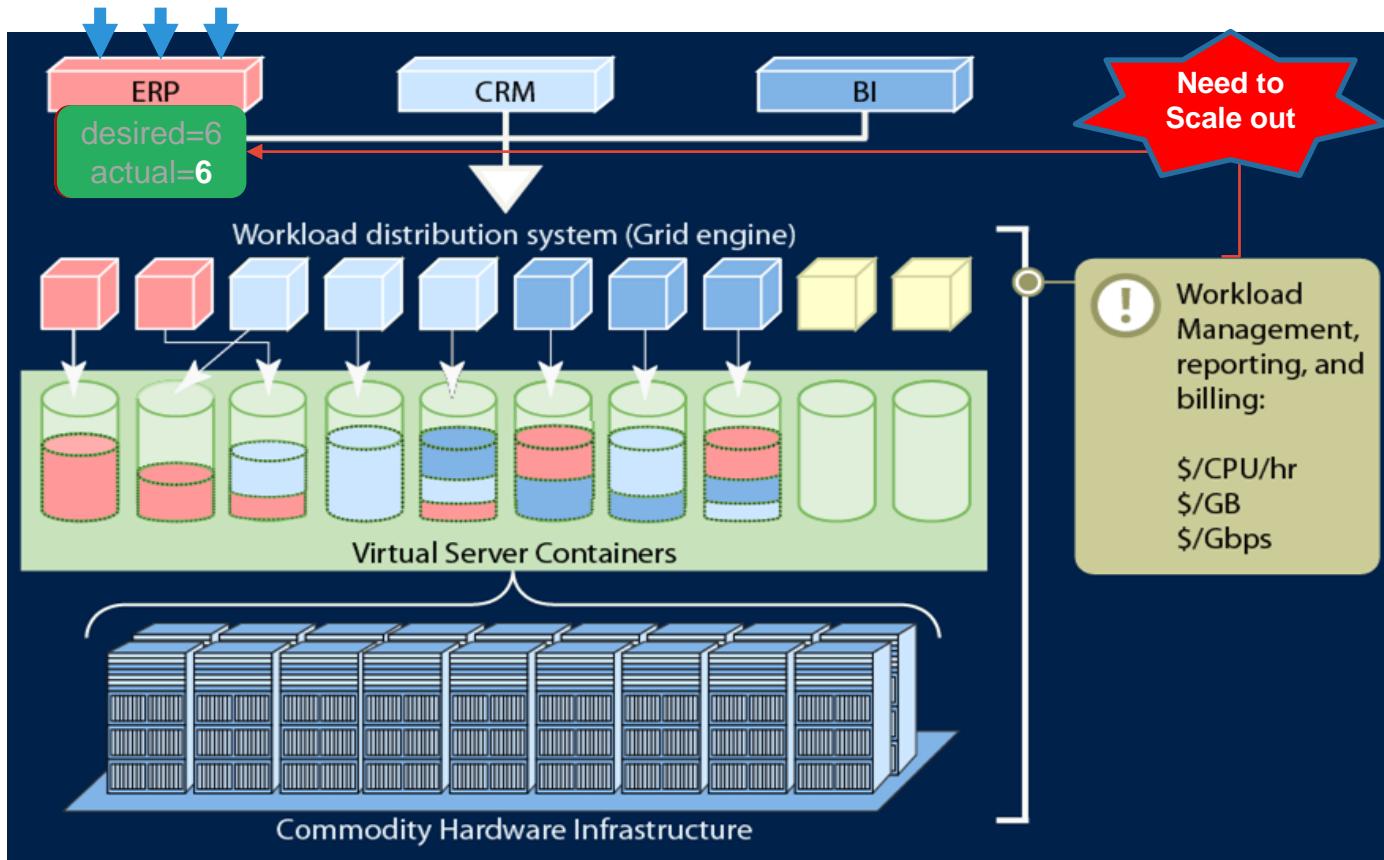


1. API Gateway : API를 사용하여 통신하는 모든 서비스들의 인증 및 제어, 트래픽 모니터링 등 수행
2. Managed Container System : 서비스들의 중앙 관리 및 인증, 라우팅등의 기능 수행
3. Backing Services : 스토리지(Block, Object), Cache, Message Queue 등의 기능 영역
4. Observability Services : 서비스들의 이벤트 모니터링 및 알림, 로그 취합, 진단 등 수행

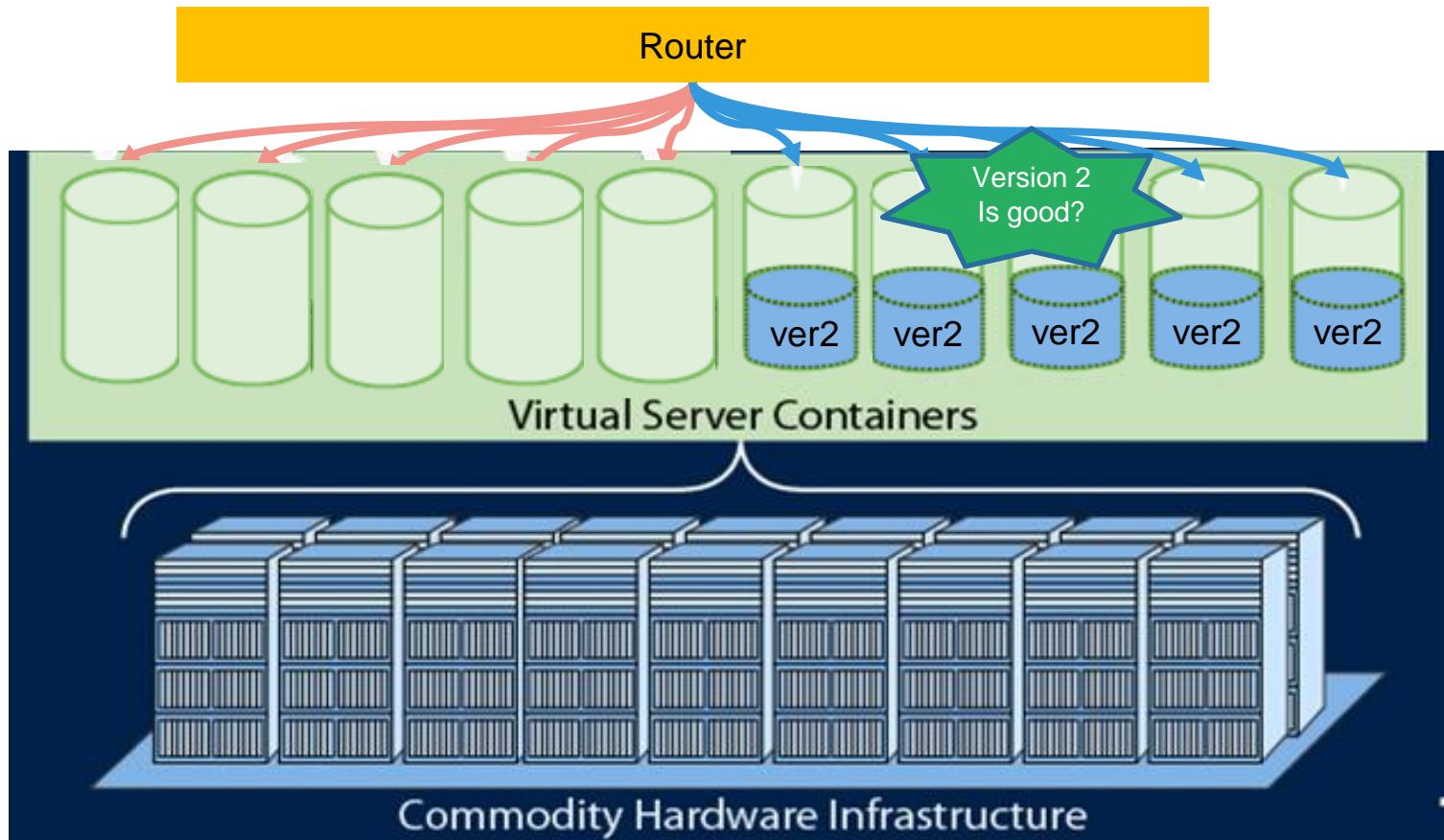
Container Orchestration – Self Healing Mechanism



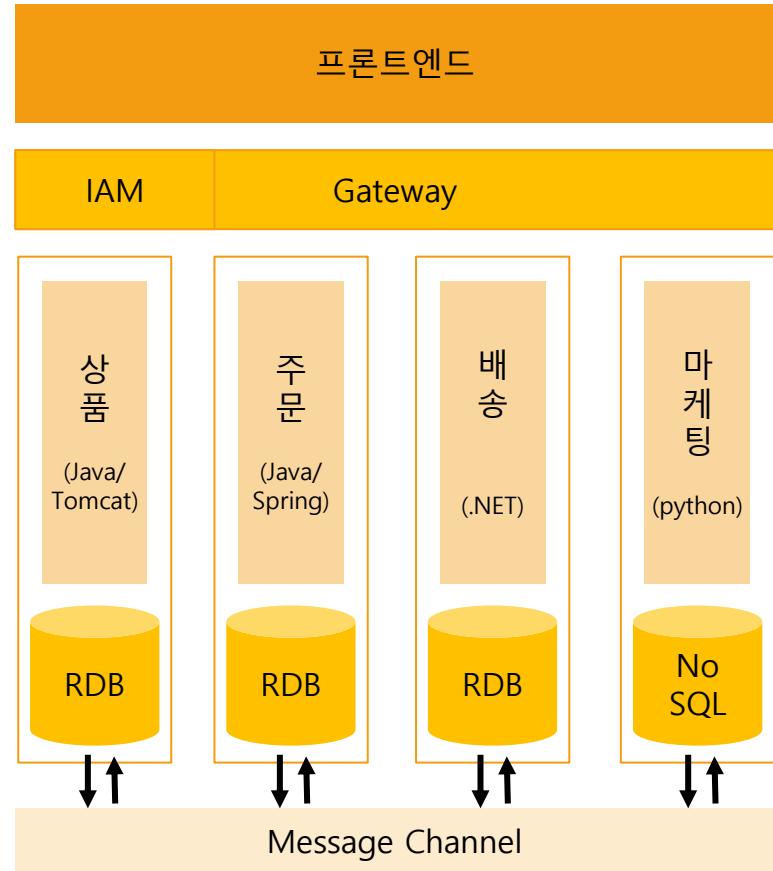
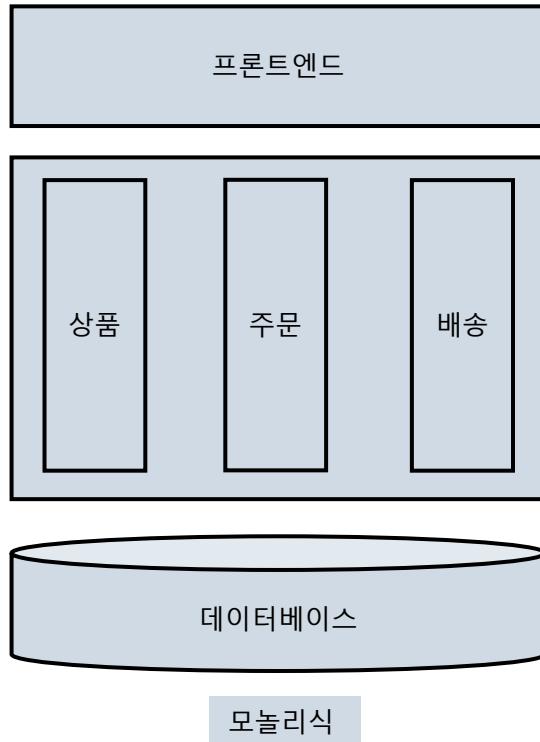
Container Orchestration – Scale out



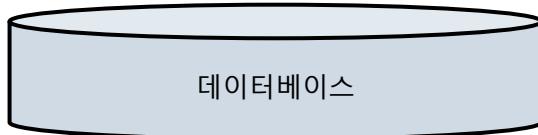
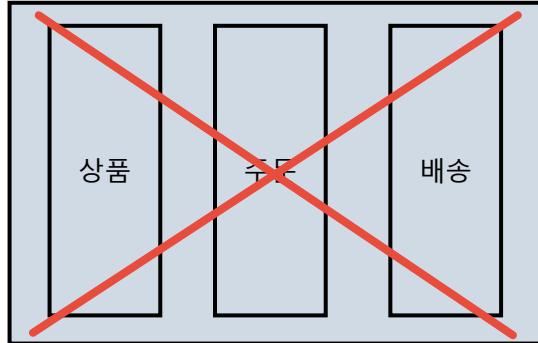
Container Orchestrator – Zero Down Time Deploy



적용 아키텍처



효과 - 장애 최소화

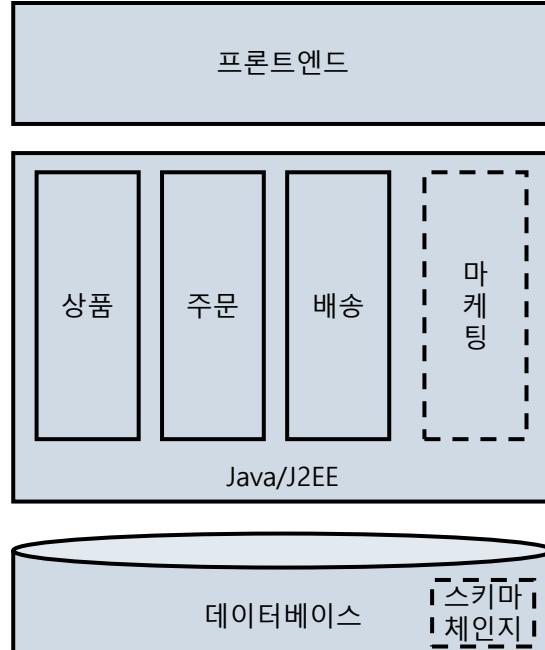


장애가 전파되며, 수동 복구로 장애 지연

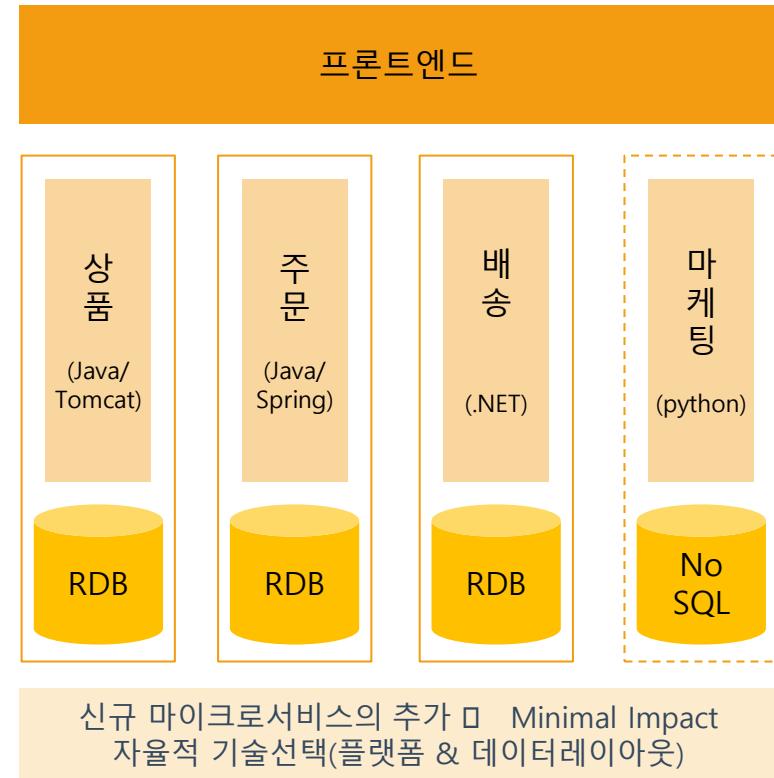


장애가 격리되며, 자동으로 복구됨(이전버전)

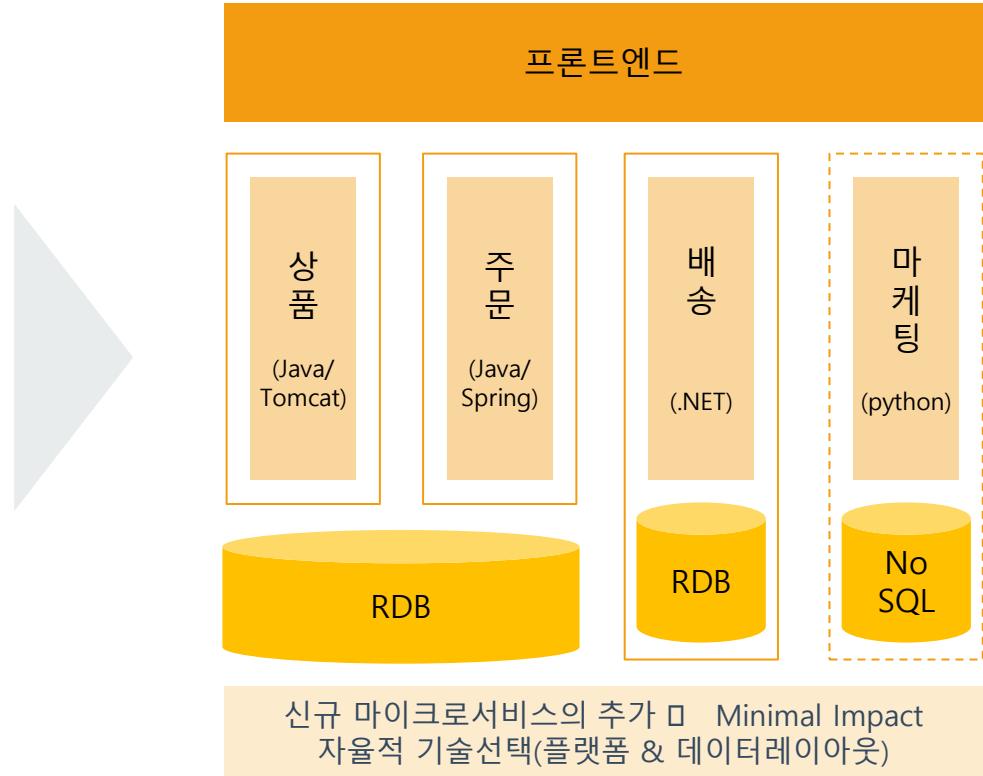
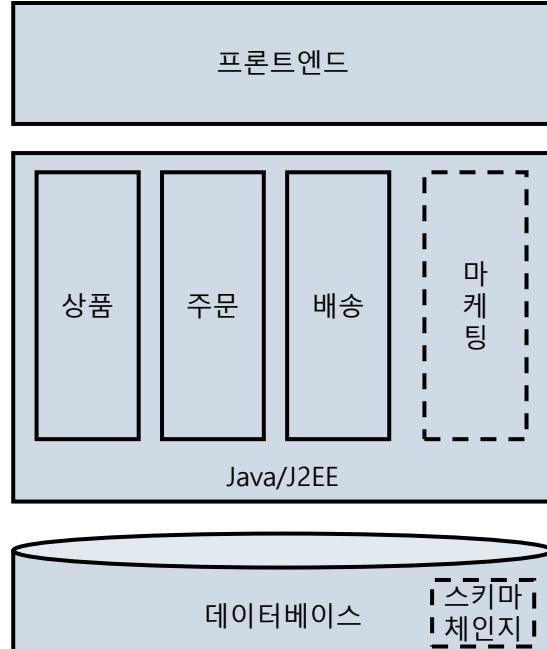
효과 - 기능 확장



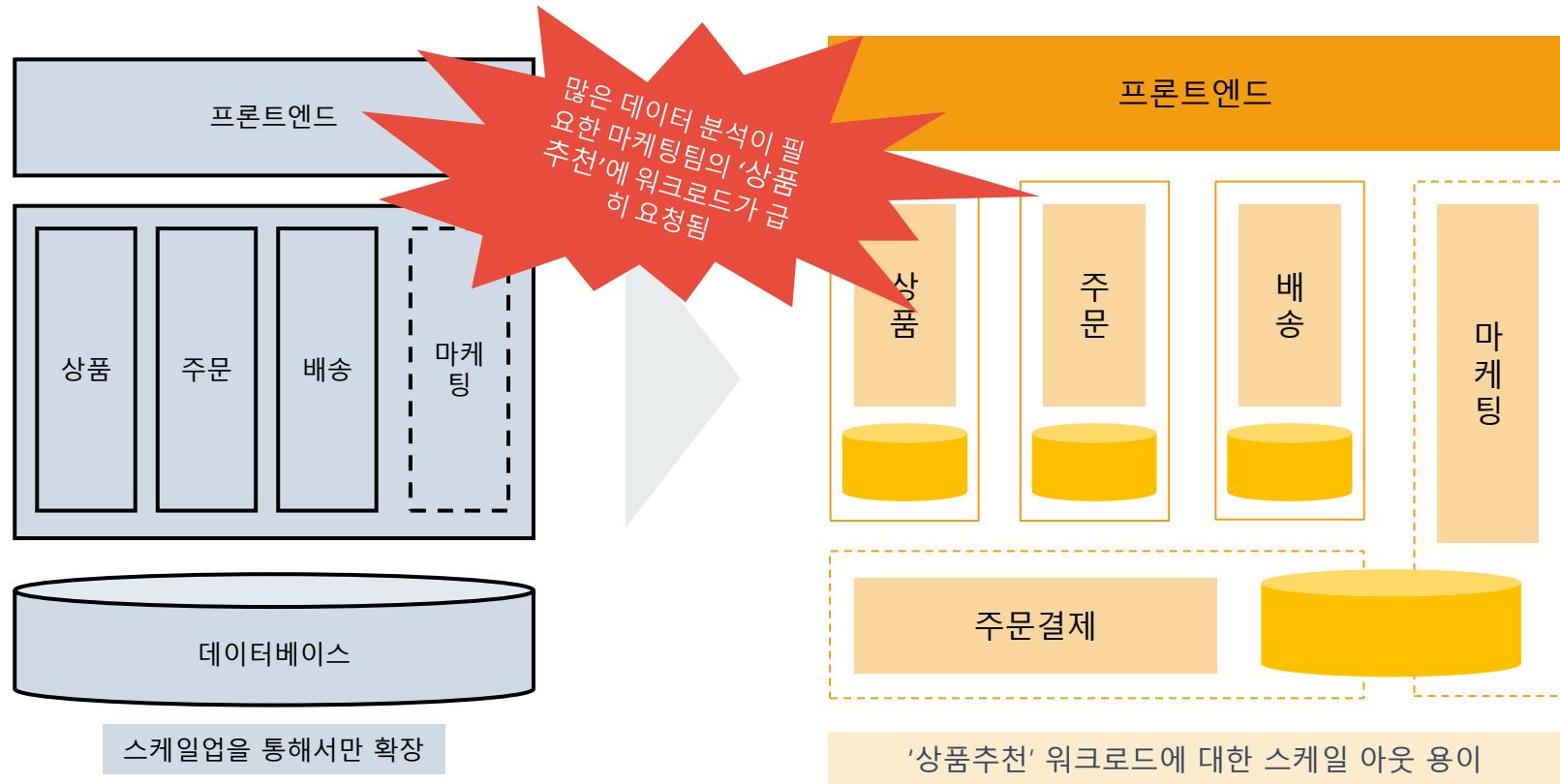
기존 코드의 수정 □ Big Impact



효과 - 기능 확장



효과 - 성능 배분



목표수준수립과 비용



늘어나는고객수를
처리하기 위한 확장

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



팀자율성 개선

- 팀수준 (KPI or Business Capability) 수준의 분리



새로운 언어와
기술의 도입

- 컨테이너 기술



시스템 작동 중단
시간 감소

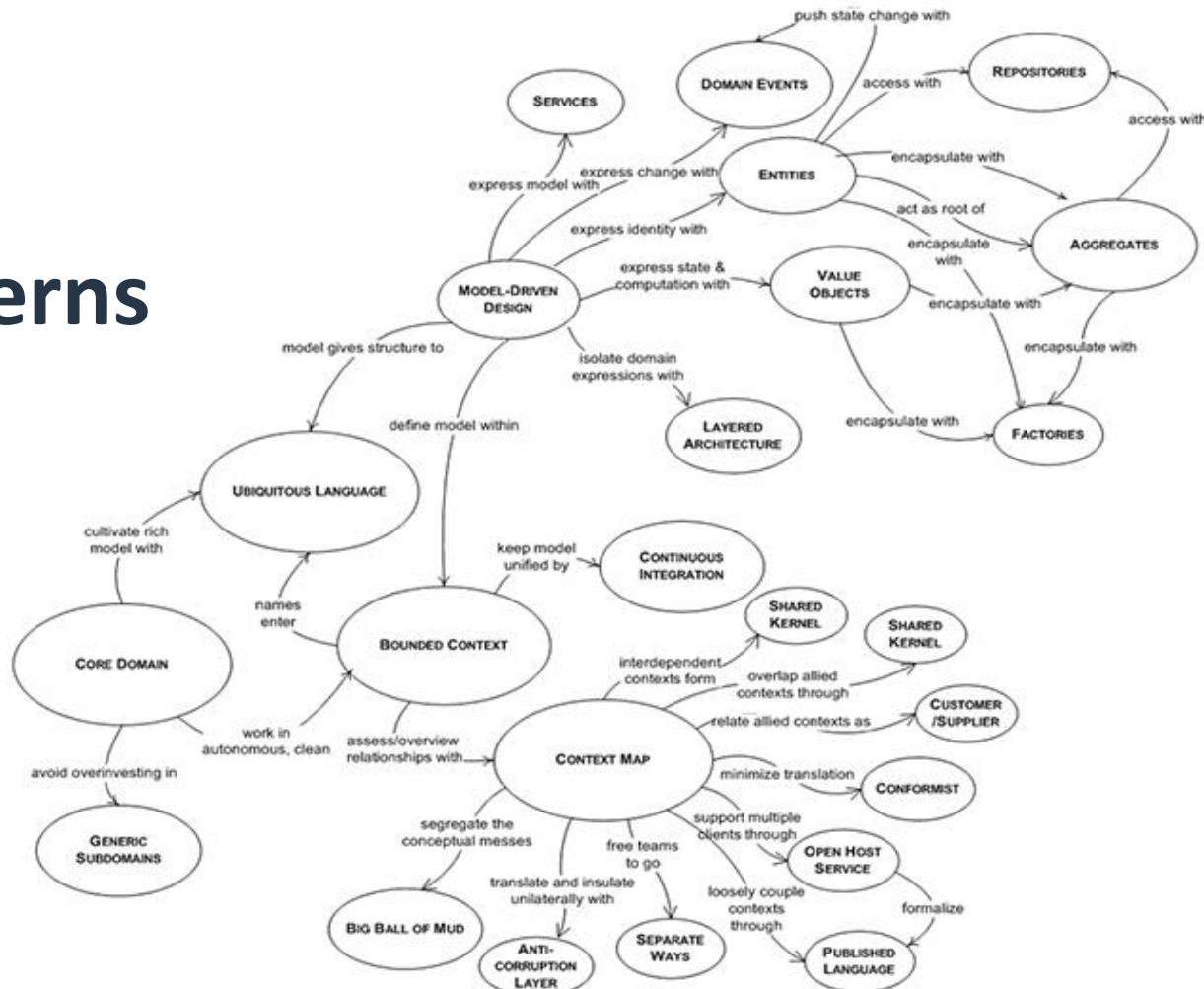
- Core/Supporting 의 분리
- 무정지 재배포
- 컨테이너 기술

Table of content

Microservice and
Event-storming-Based
DevOps Project

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

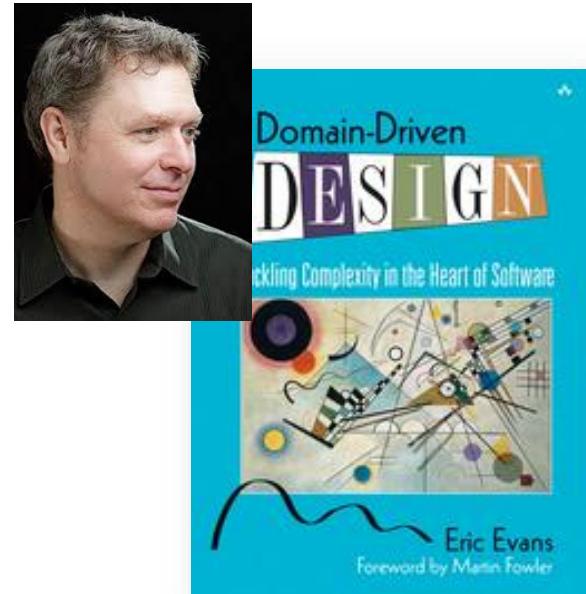
DDD Patterns



Domain-Driven Design & MSA

DDD for MSA

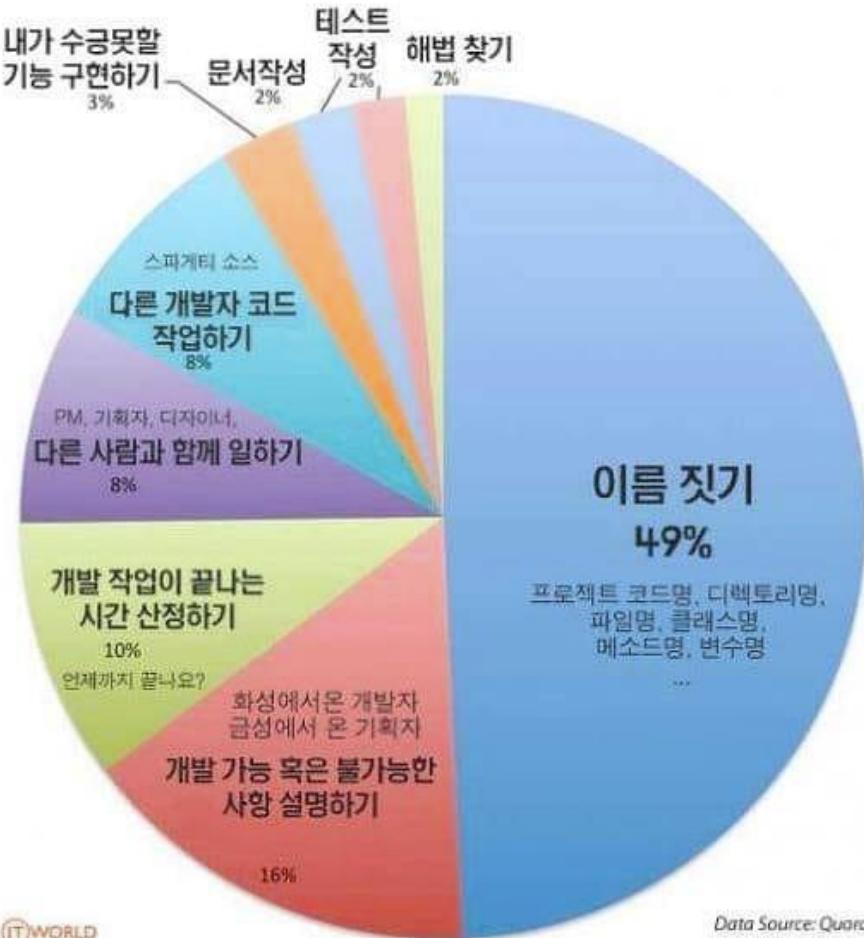
- **Bounded Context 와 Ubiquitous Language**
 - 어떤 단위로 마이크로 서비스를 쪼개면 좋은가?
- **Context Mapping**
 - 서비스를 어떻게 결합할 것인가?
- **Domain Events**
 - 어떤 비즈니스 이벤트에 의하여 마이크로 서비스들이 상호 반응하는가?



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



프로그래머가 가장 힘들어하는 일은?



Bounded Context

(한정된 맥락)

Ubiquitous Language
(도메인 언어)



어느편이 이해하기 쉬운가요?

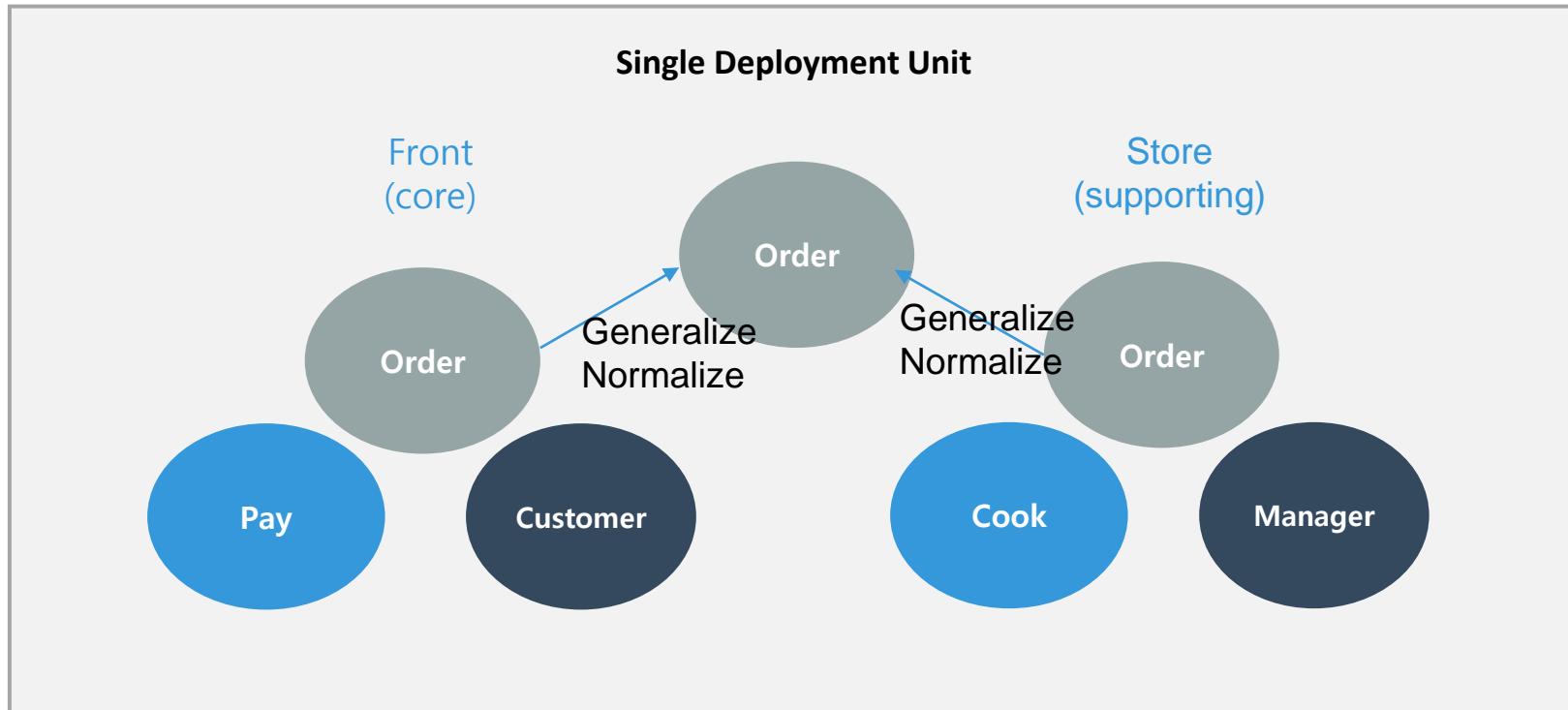
```
// 개발자1  
var currTS = new Date().getTime(); // 현재 타임스탬프  
var timeStamp1 = currTS - 60000 * 60; // 1시간 전  
var timeStamp2 = currTS - 60000 * 10; // 10분 전
```

```
// 개발자2  
var now = new Date().getTime();  
var beforeAnHour = now - 60000 * 60;  
var before10min = now - 60000 * 10;
```

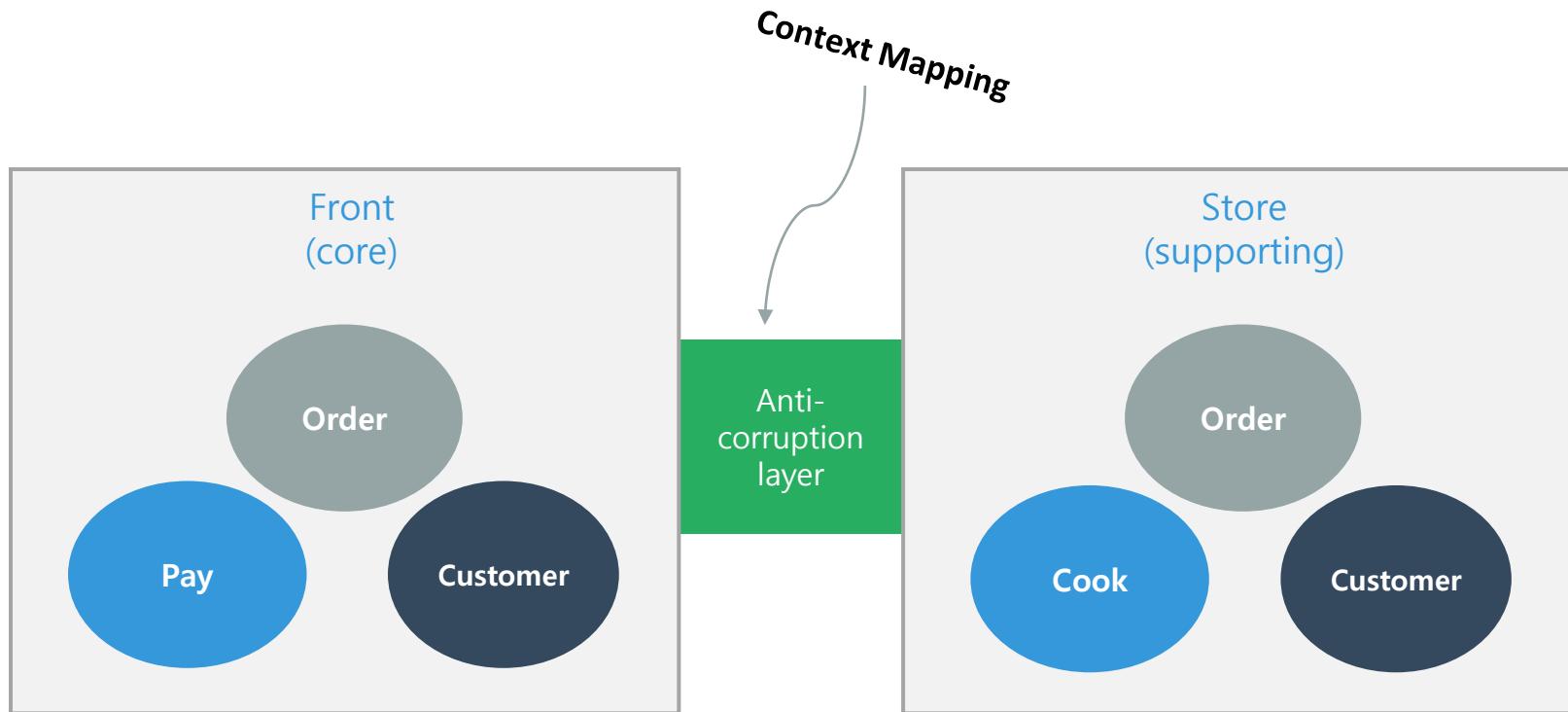
Bounded Context and Ubiquitous Language

	SW	CONSTRUCTION
A Project		
An Architecture		
A Developer		

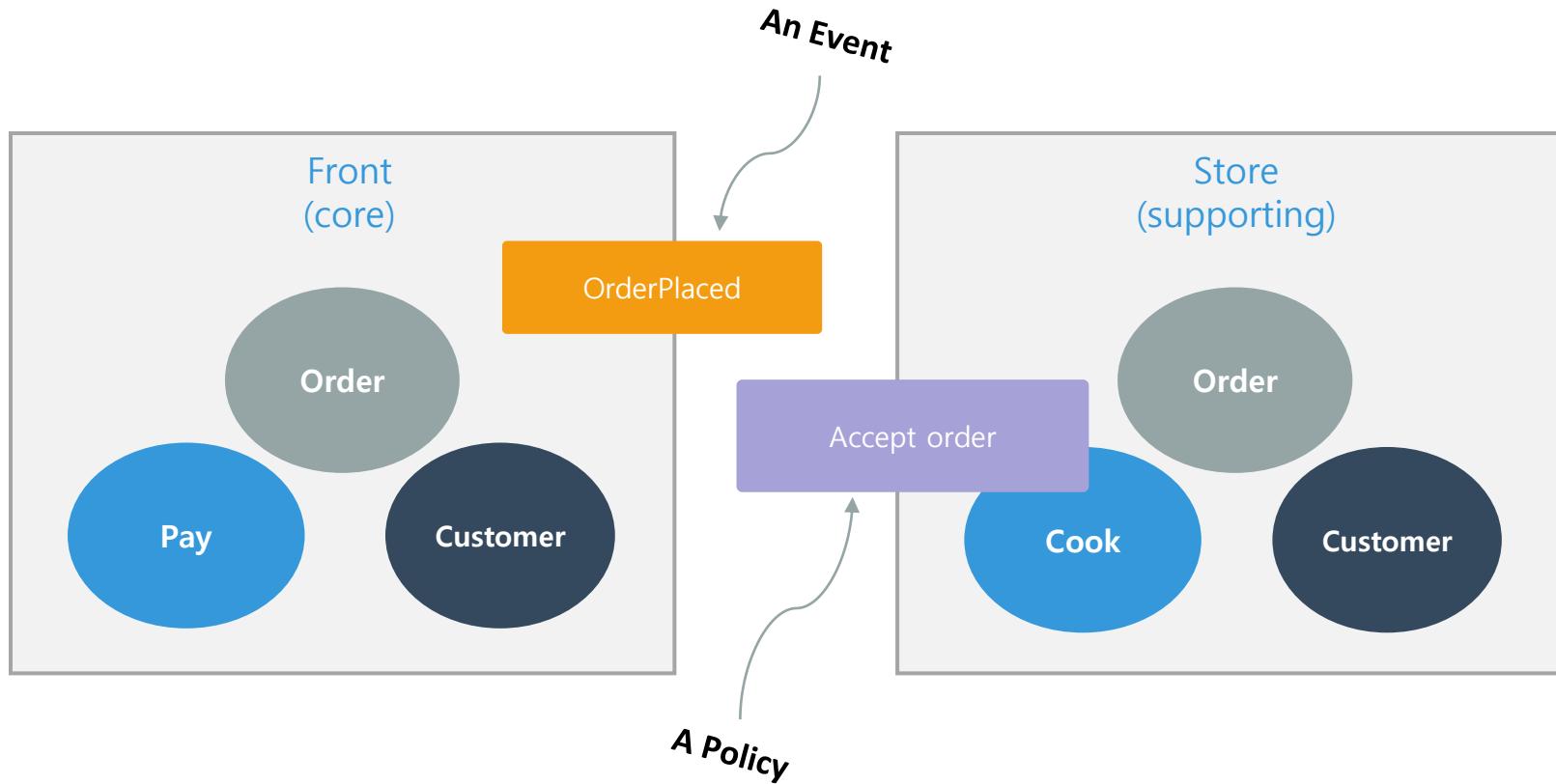
In Monolith :



In Microservices :



In Microservices :



マイクロ 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 : Supporting 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

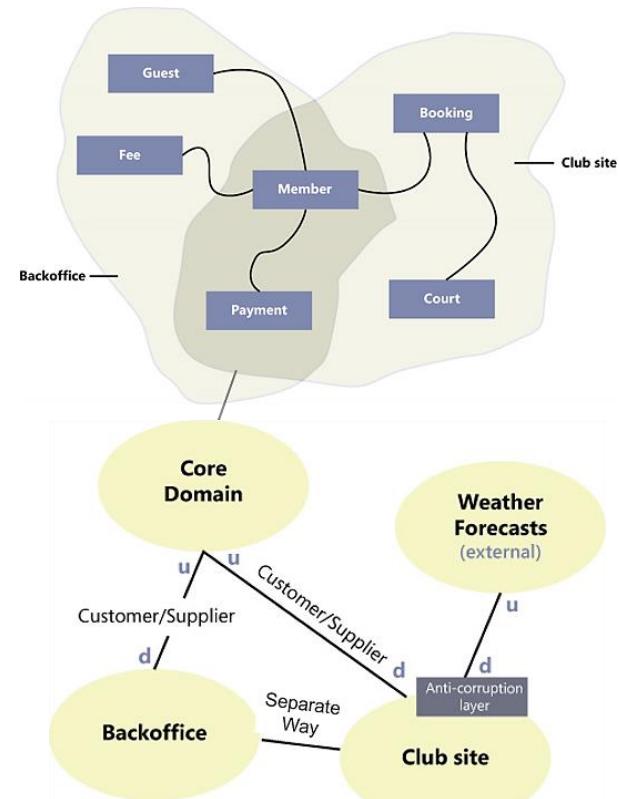


3순위 : General Domain

- SaaS 등을 사용하는게 더 저렴한, 기업 경쟁력과는 완전 무관한
예) 결재, 빌링 서비스 등

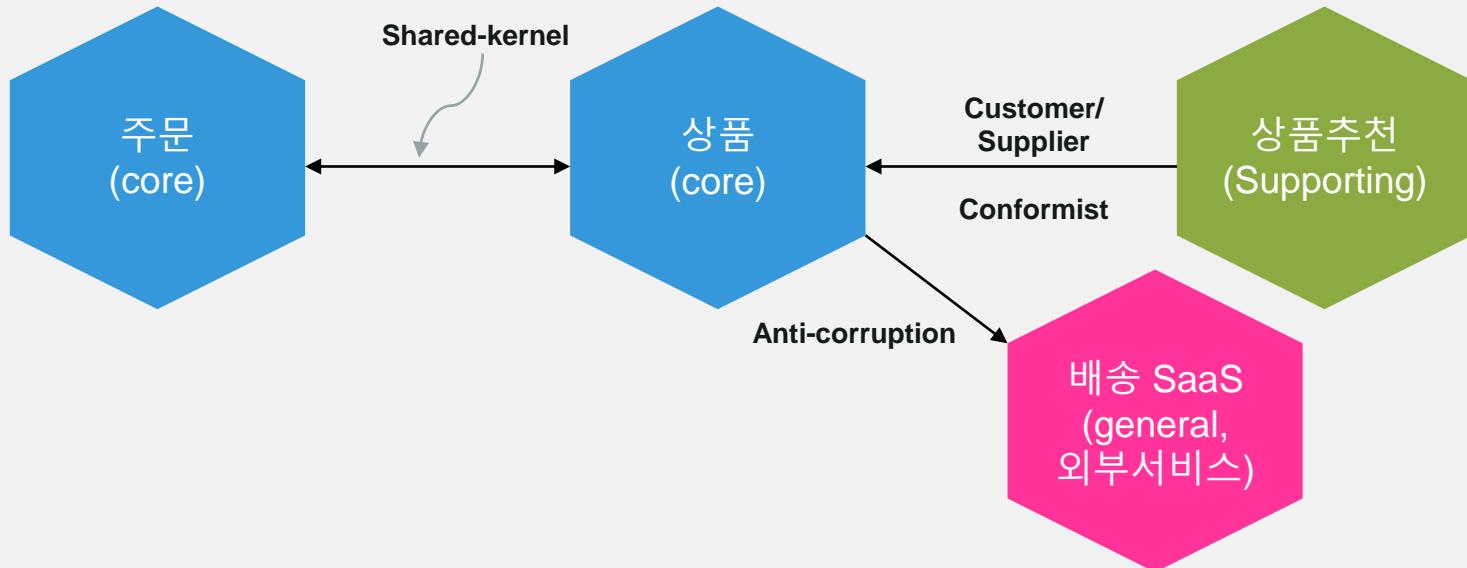
Ref : Relation types between services

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



マイクロ 서비스간의 서열과 역학관계

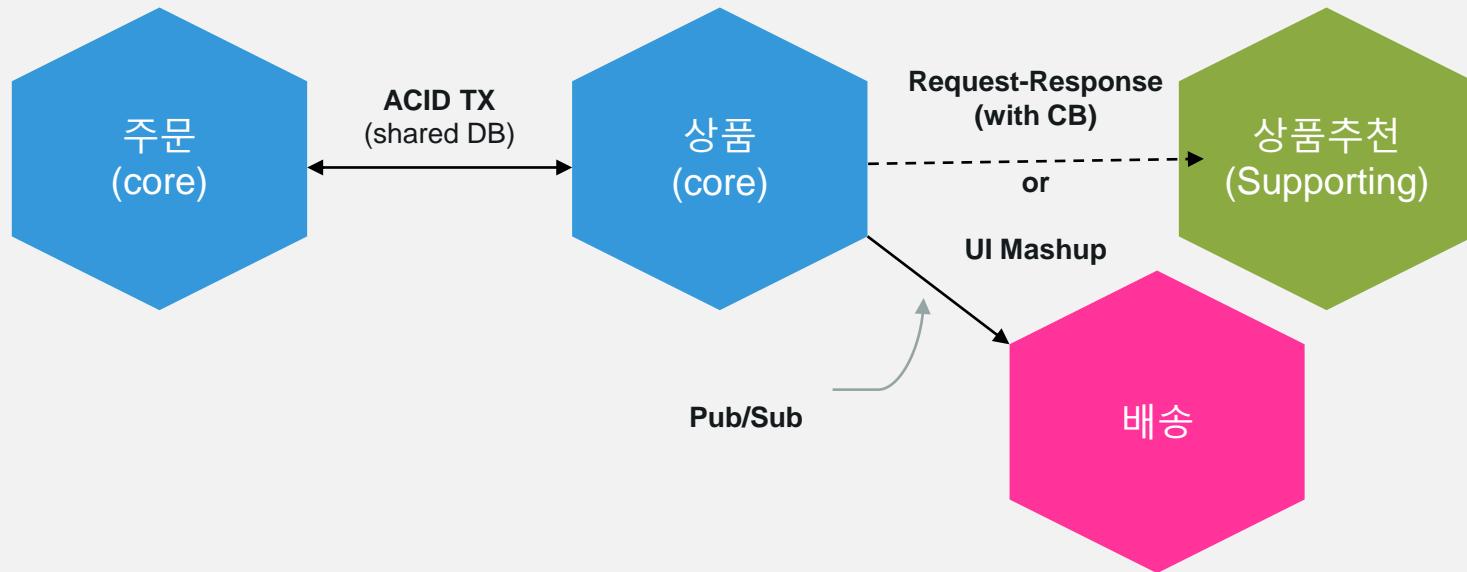
“ 어느 마이크로 서비스의 인터페이스를 더 중요하게 관리할 것인가? ”



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

マイクロ 서비스간의 서열과 역학관계

“ 마이크로서비스간 트랜잭션의 묶음을 어떻게 할 것인가? ”



배송기사가 없다고 주문을 안받을 것인가? 상품 추천이 안된다면 주문버튼을 안보여줄 것인가?

Core Domain 간에는 강결합이 요구되는 경우가 생길 수 있다.

하지만 우선순위가 떨어지는 비즈니스 기능을 위해 강한 트랜잭션을 연결할 이유는 하등에 없다.

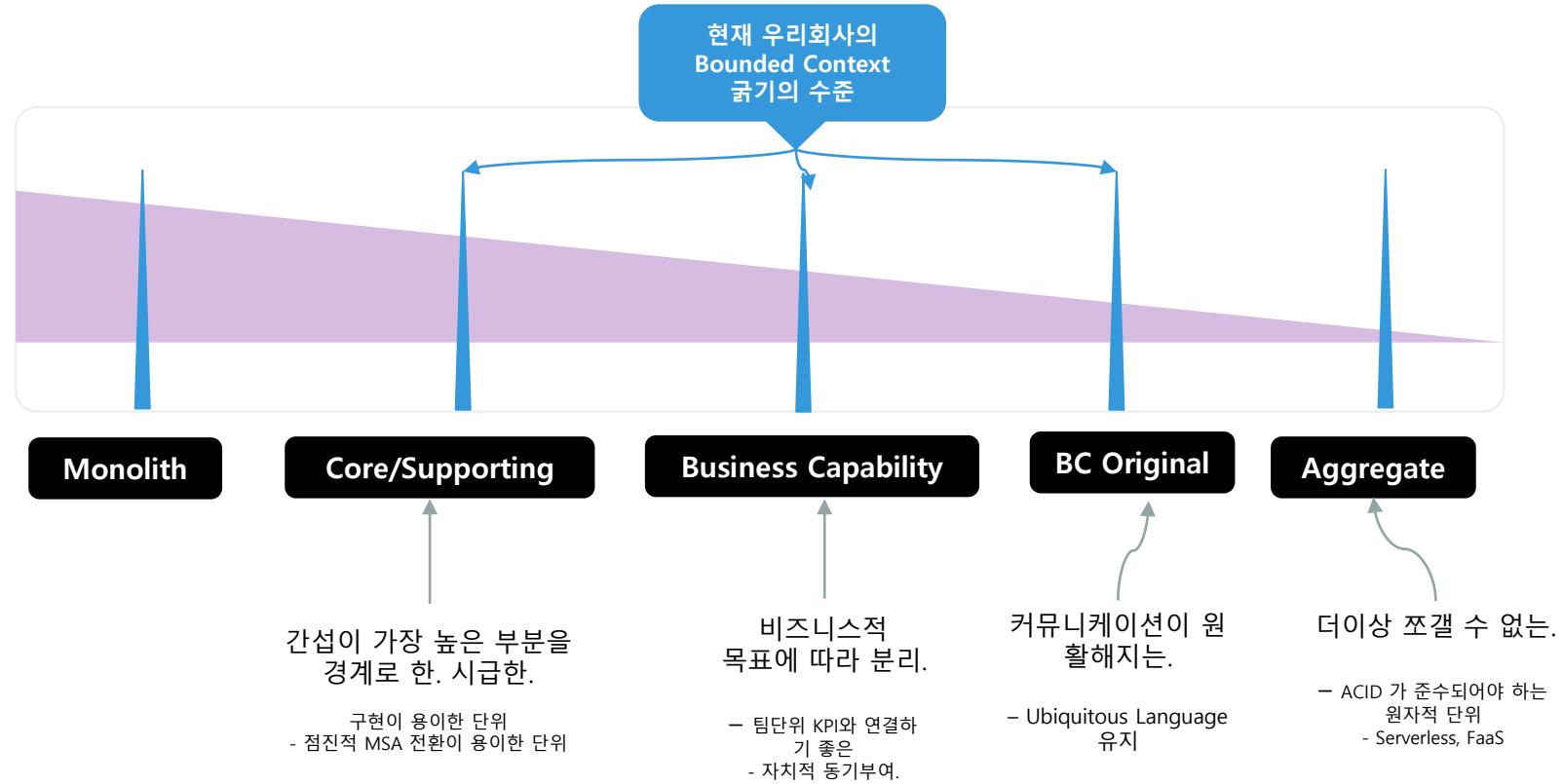
マイクロ 서비스간의 서열과 역학관계

“ 손님이 많이 와서 집이 좁아졌다.. 무엇을 우선적으로 줄일 것인가? ”



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

분리수준 - 어느 굽기로 쪼갤 것인가?



분리관점 - 어떤 칼로 쪼갤 것인가?



도메인 관점

업무 중요도와
자치성

- > core/supporting KPI
- 마이크로서비스에서 일순위 분리 관점

기술 관점

기술 아키텍처가 상이한
모듈

- 데이터베이스, 플랫폼, 언어가 상이한

트랜잭션 관점

트랜잭션 모형이
다른

- ACID or Eventual Consistency?

유지관리관 점

변경시 동반되어
수정되어야 하는
영역

재사용 관점

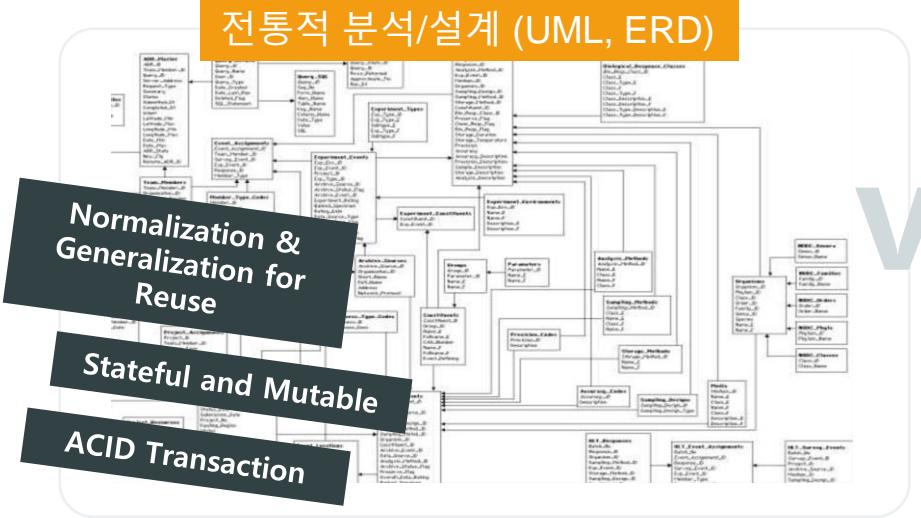
유ти리티 성격의
서비스

- But,
마이크로서비스에서는
후순위 분리 관점

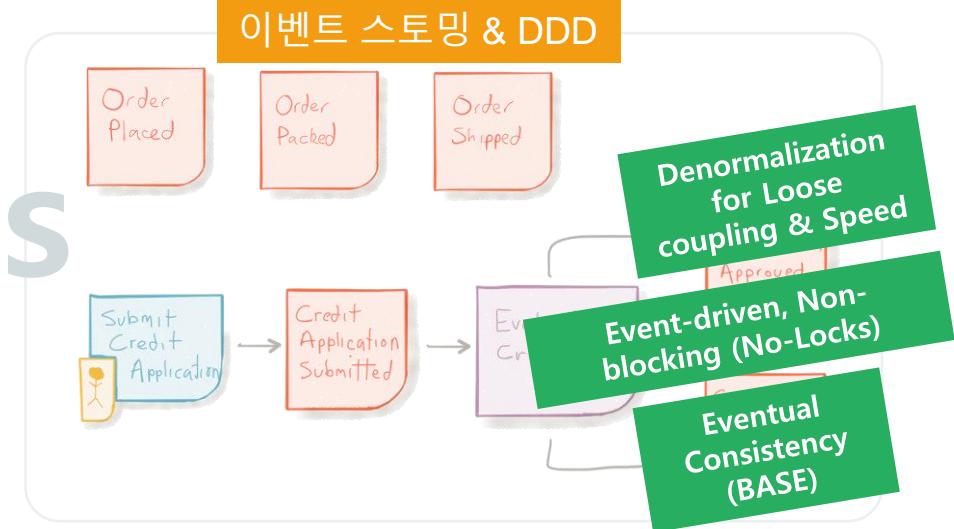
Event Storming : DDD 를 쉽게 하는 방법

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

전통적 분석/설계 (UML, ERD)

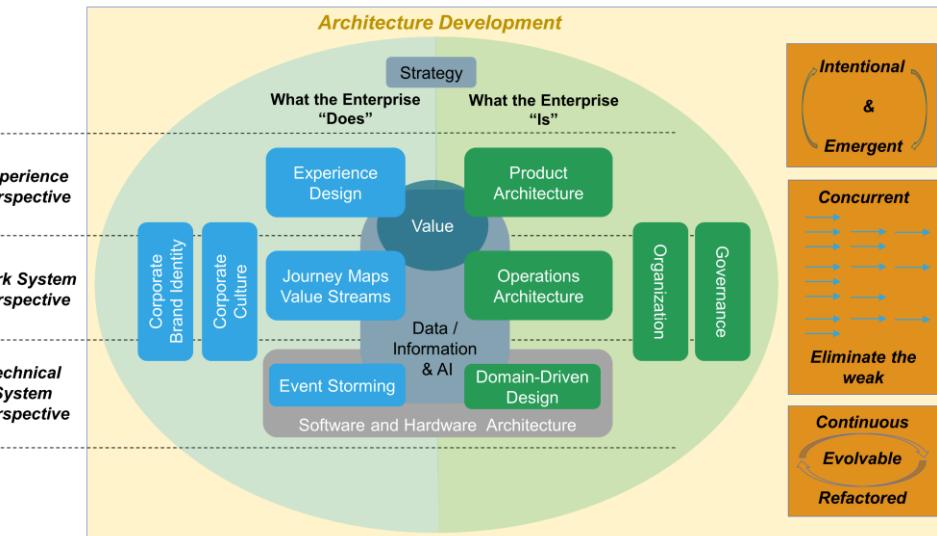


이벤트 스토밍 & DDD



Event Storming 의 위상

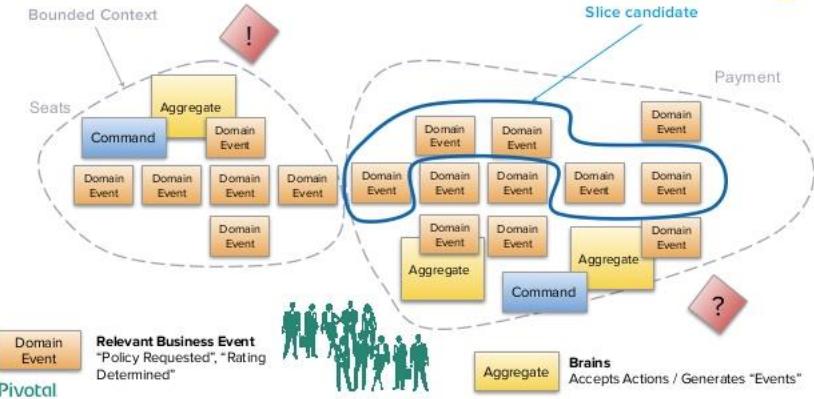
The Open Group 의 OAA 의 빌딩블록



<https://pubs.opengroup.org/architecture/o-aa-standard/#Introduction>

Pivotal 의 AppTX 방법론

Event Storming



<https://www.slideshare.net/Pivotal/pivotal-s-secret-sauce>

IBM 의 Garage 방법론... etc

Event Storming : Prepare

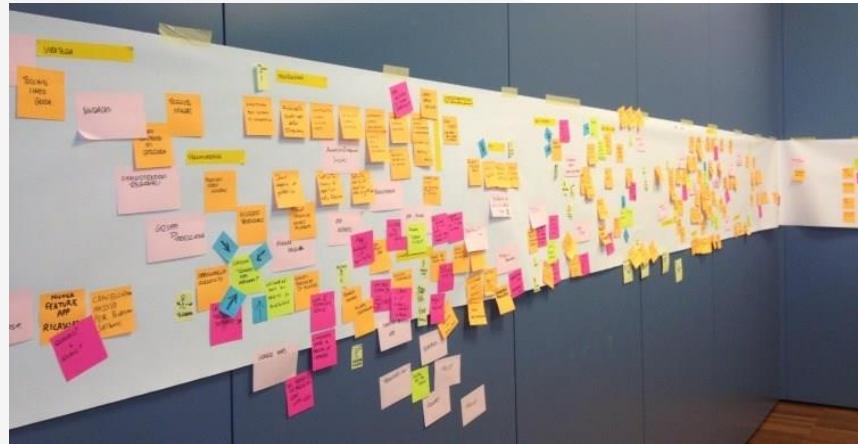
People

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

Tools

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

퍼실리레이터 (옵션)



Type of stickers

Domain Event
(Orange)

발생한 사실, 결과, Output

도메인 전문가가 정의
이벤트 퍼블리싱



사용자, 페르소나, 스테이크 홀더

유저 인터페이스를 통해 데이터를
소비하고 명령을 실행하여 시스템
과 상호 작용

Definition

정의

도메인에 대한 용어 등의
설명, 기술

Command
(Sky Blue)

의사결정, Input, API, UI 버튼

현재형으로 작성,
행동, 결정 등의 값들에 대한
UI 혹은 API

Aggregate
(Yellow)

구현체 덩어리, 시스템, 모듈

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

Read Model
(Green)

의사결정에 필요한 자료, View, UI

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

External System
(Pink)

외부 시스템

시스템 호출을 암시 (REST)

Policy
(Lilac)

정책, 반응(Reaction)

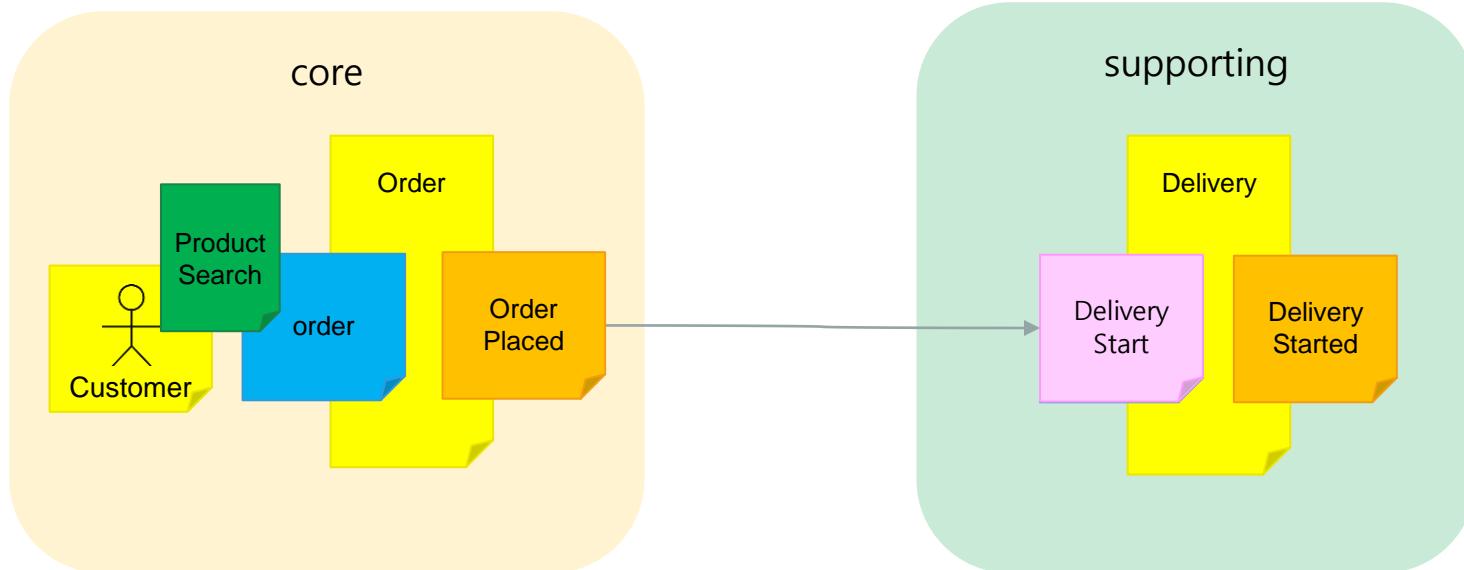
이벤트에 대한 반응
(서브스크라이브)
비즈니스 룰 엔진 등

Comment
Or
Question
(Purple)

의견 또는 질문

추가적인 내용 입력,
예측되는 Risk

Examples



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 방식(유비쿼터스 랭귀지)으로 표현하고 동사의 과거형 (p.p.)으로 표현한다.
- 시작 및 종료 이벤트를 식별하고 스티커를 불일 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

PO 가 주도
(업무전문가)



Account
Created

Email
Sent

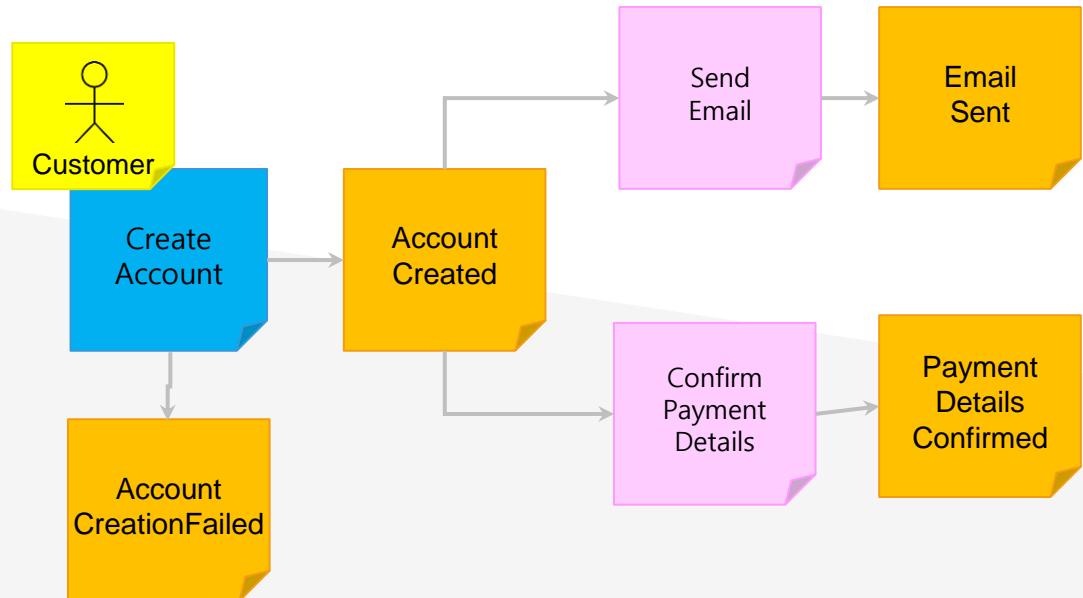
Payment
Details
Confirmed

Command, Policy, Actor 도출

PO + UI/UX 가 주도
(업무전문가)



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



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

* Command라는 용어는 CQRS에서 유래했으며,
쓰기서비스인 Command와 읽기행위인 Query는 구분됨.

Aggregate 도출

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

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

아키텍트/개발자/DBA
주도

Inputs



Outputs

Create Account

Account Created

Update Account

Account Updated

Delete Account

Account Deleted

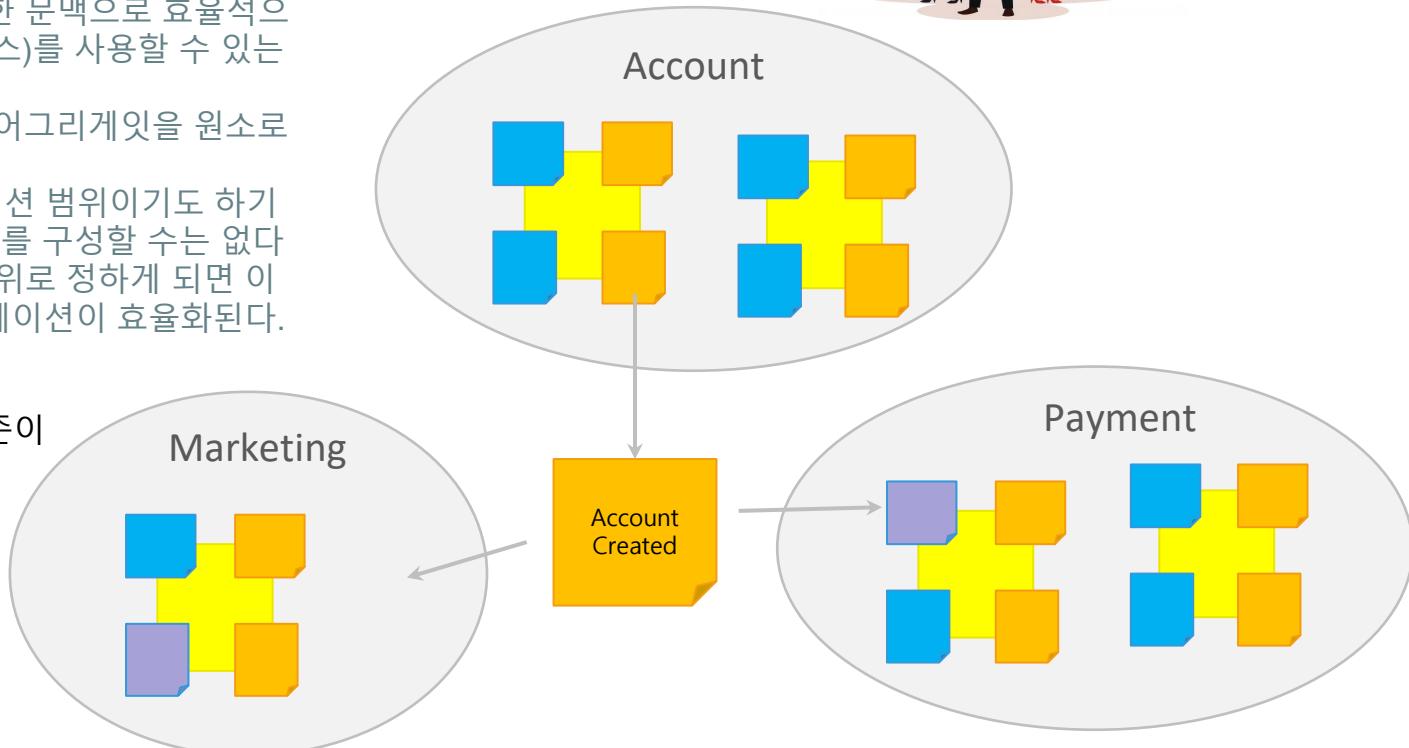
Account Mgmt

Bounded Contexts 도출

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

찾는 방식은 여러가지 기준이 있다

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

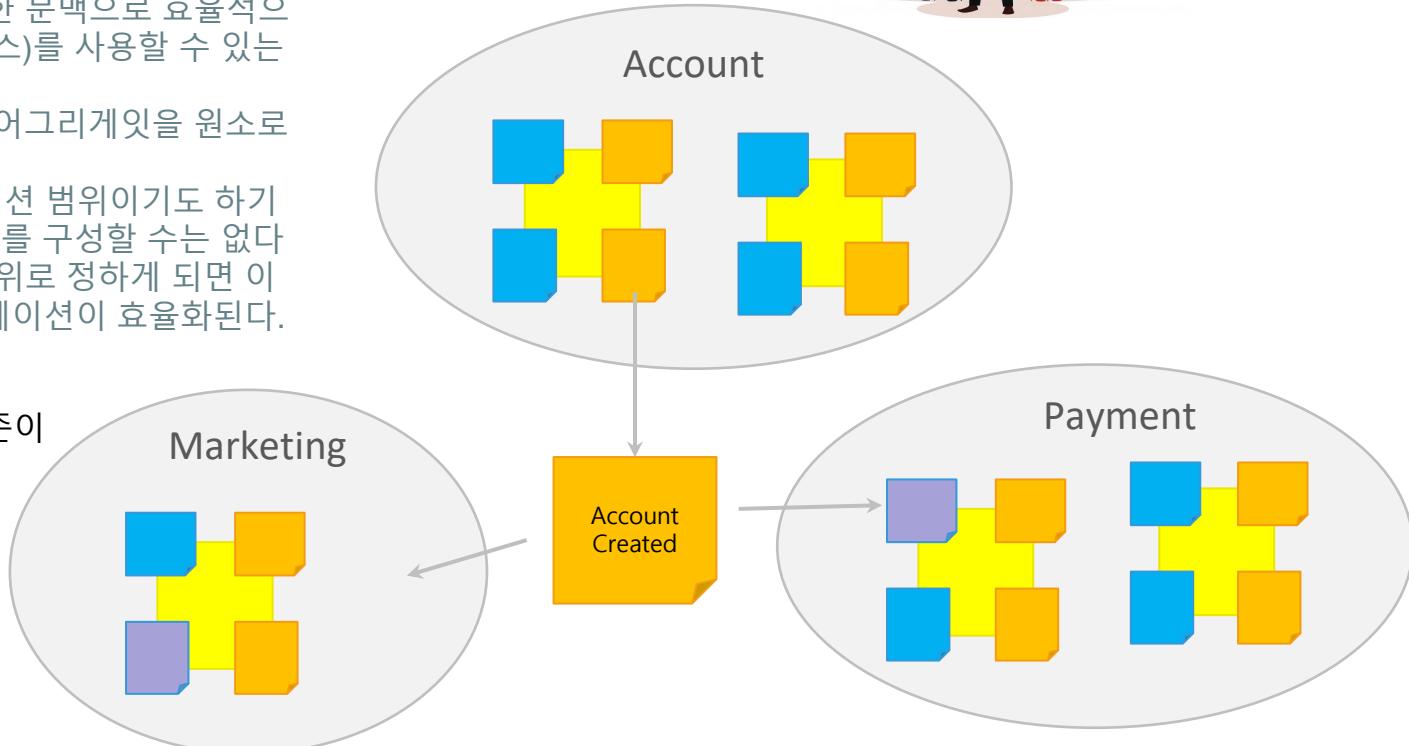


Bounded Contexts 도출

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

찾는 방식은 여러가지 기준이 있다

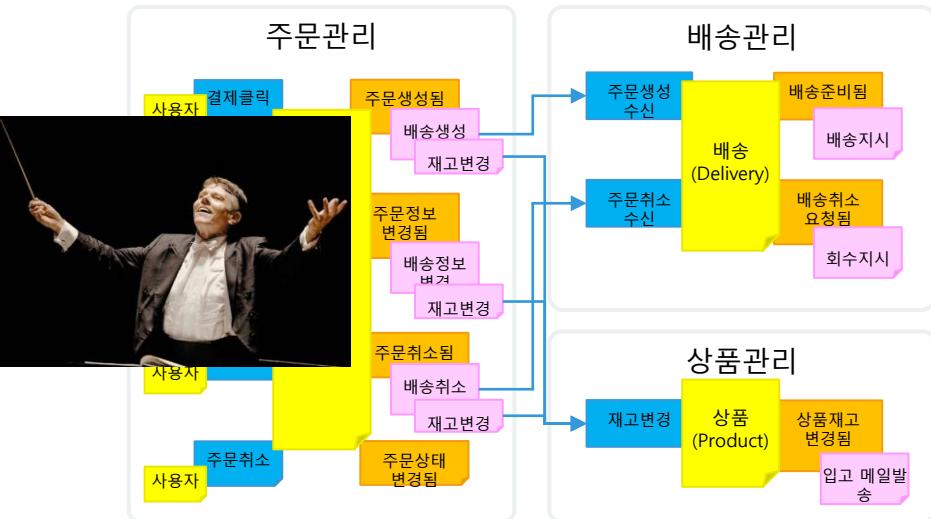
- Domain Boundary
- Transaction Boundary
- Technical Stack
- ...



Context Mapping

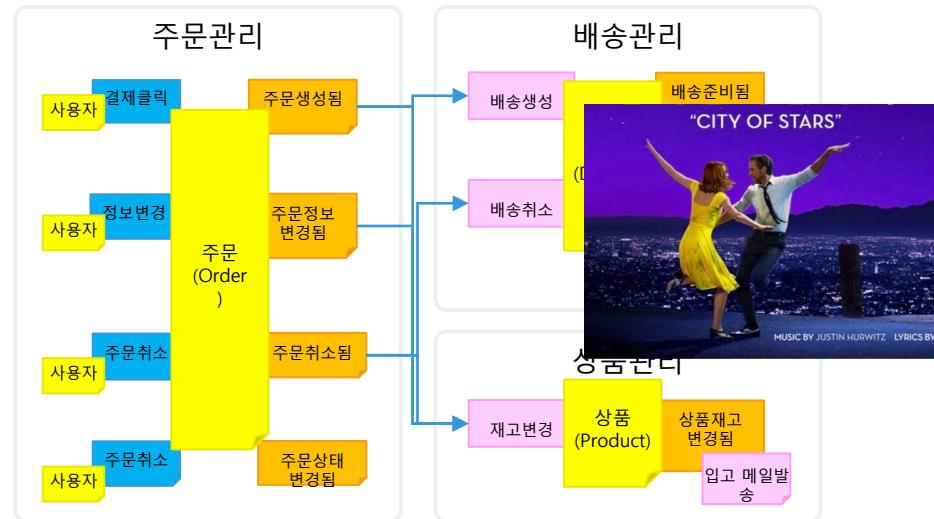
라이락 스티커 (Policy) 를 어디에 둘 것인가? - Orchestration or Choreography? Or Mediation?

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

Lab Time – 12 번가 예제 – 최초 Event Discovery

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

OrderPlaced

DeliveryStarted

InventoryDecreased

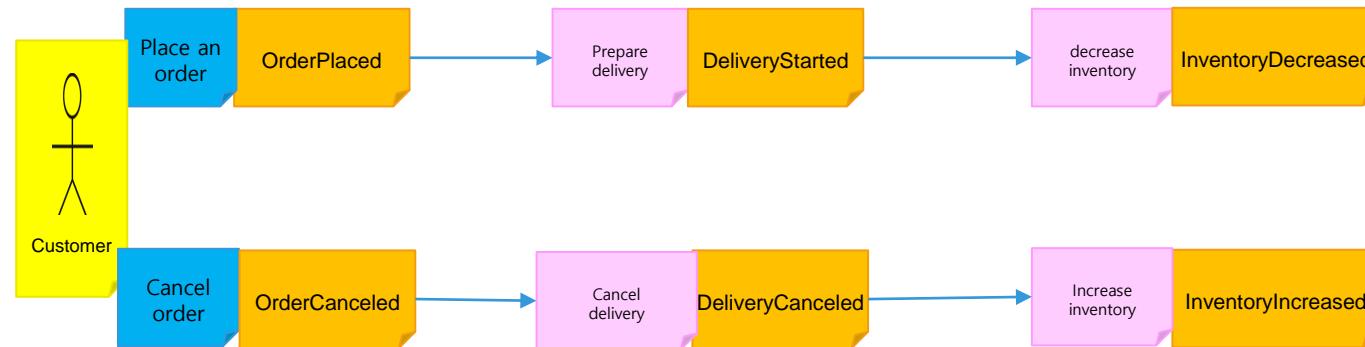
OrderCanceled

DeliveryCanceled

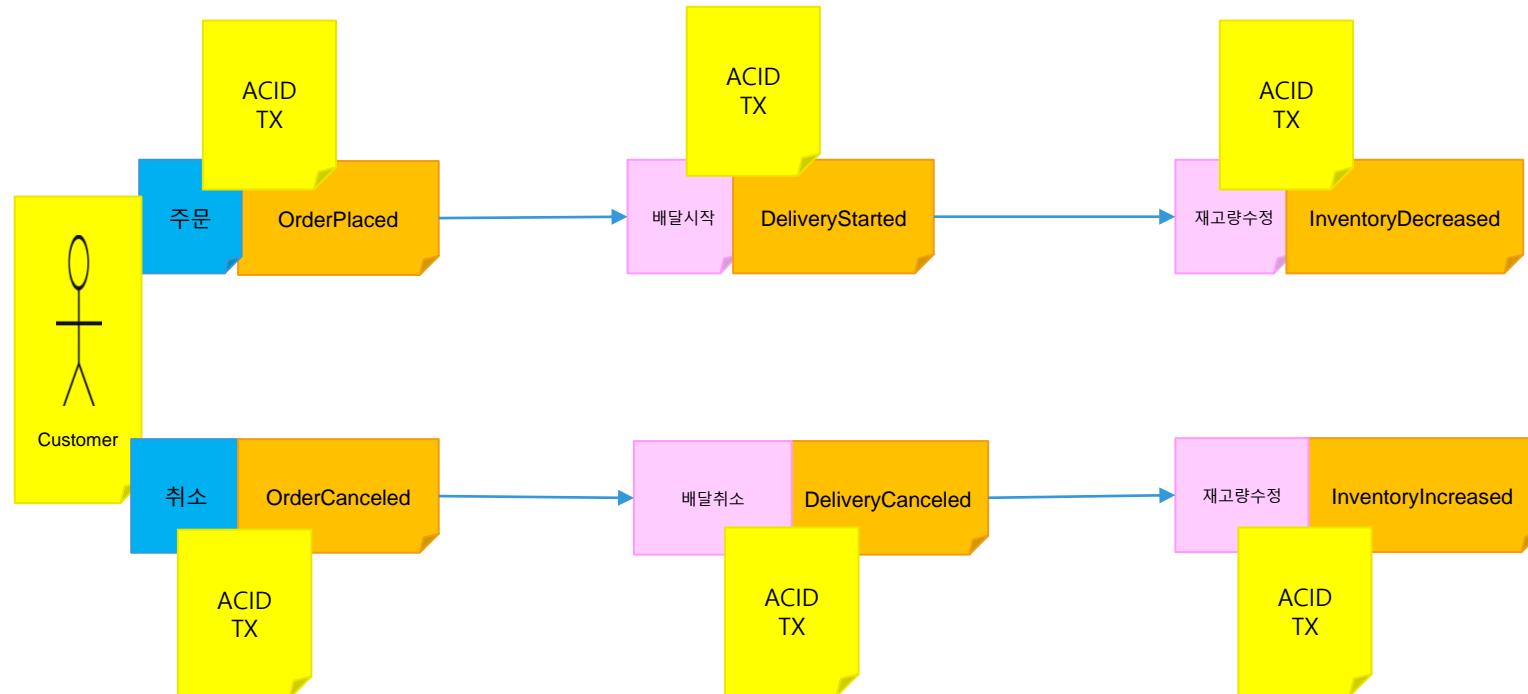
InventoryIncreased

Lab Time – Command, Policy, Actor

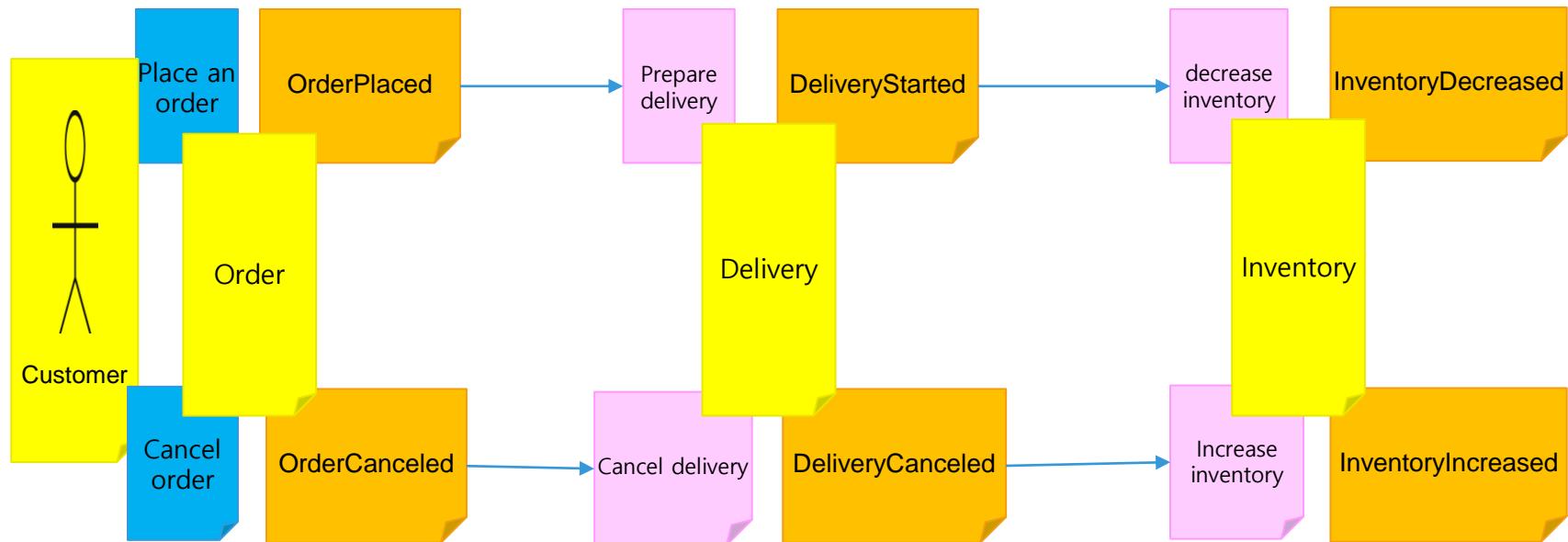
- **Customer** search products and clicks the order button to placed an order.
- When an order is placed, the delivery team prepares for delivery of the product and the delivery will start.
- When delivery has been started, the product inventory is decreased.
- **Customer** can cancel their orders.
-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



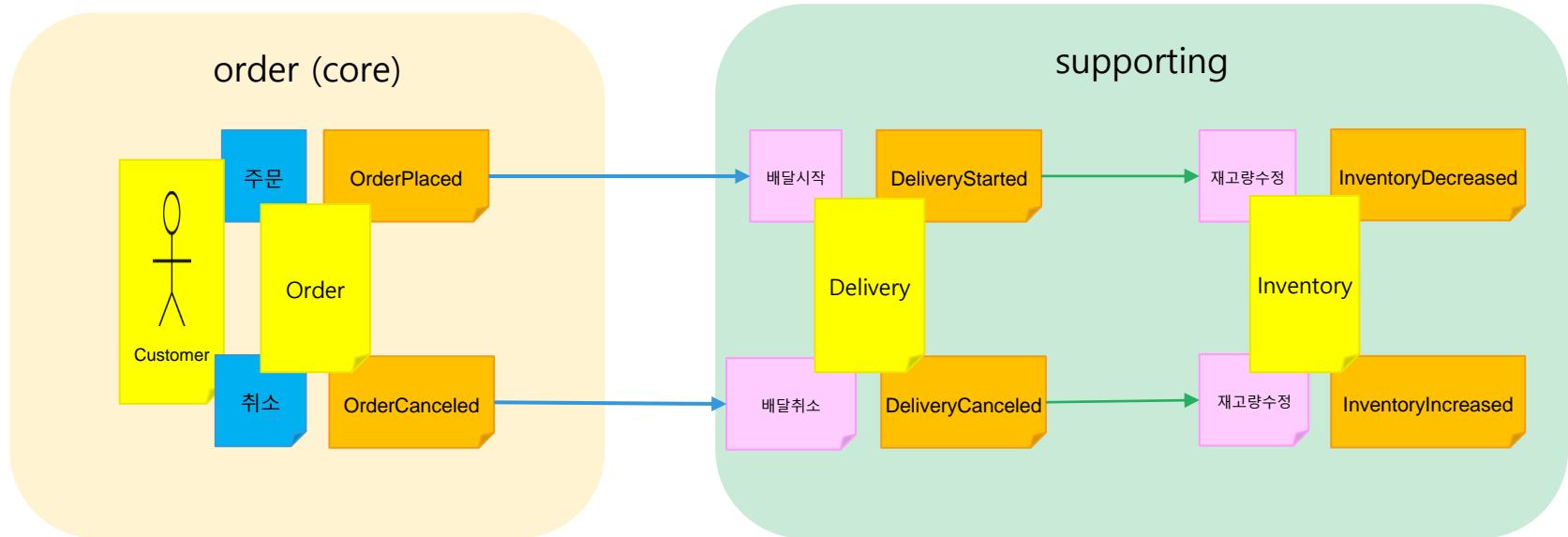
Lab Time – Aggregate



Lab Time – Aggregate (2)



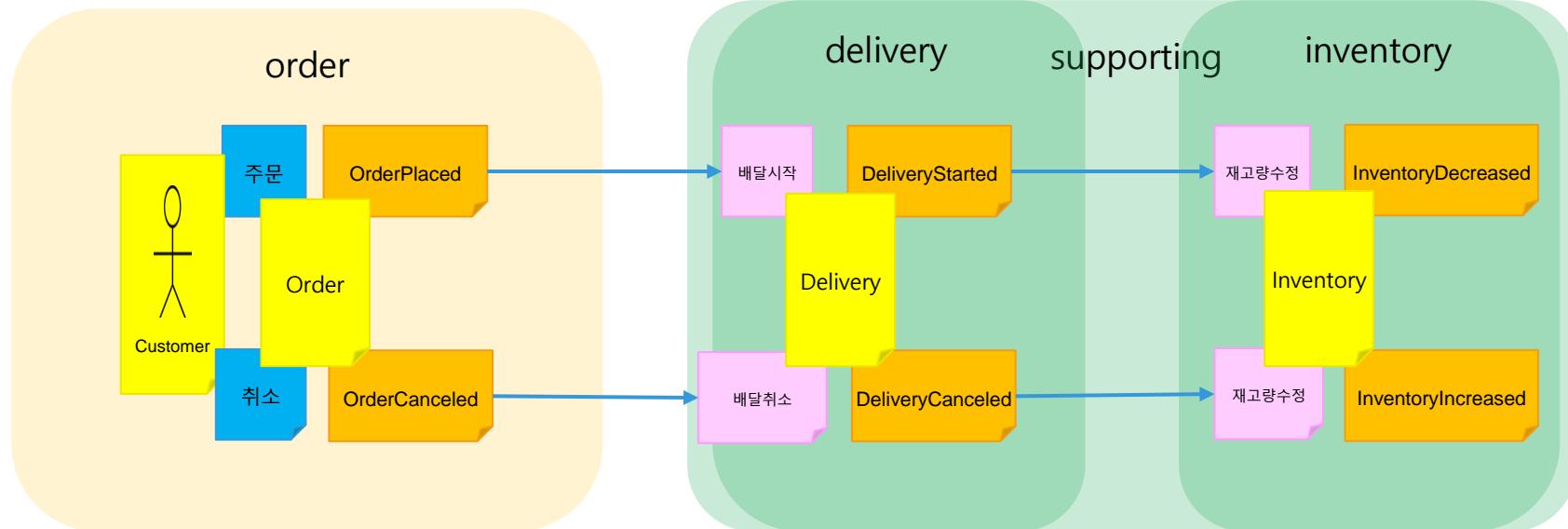
Lab Time – Bounded Context 분리



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

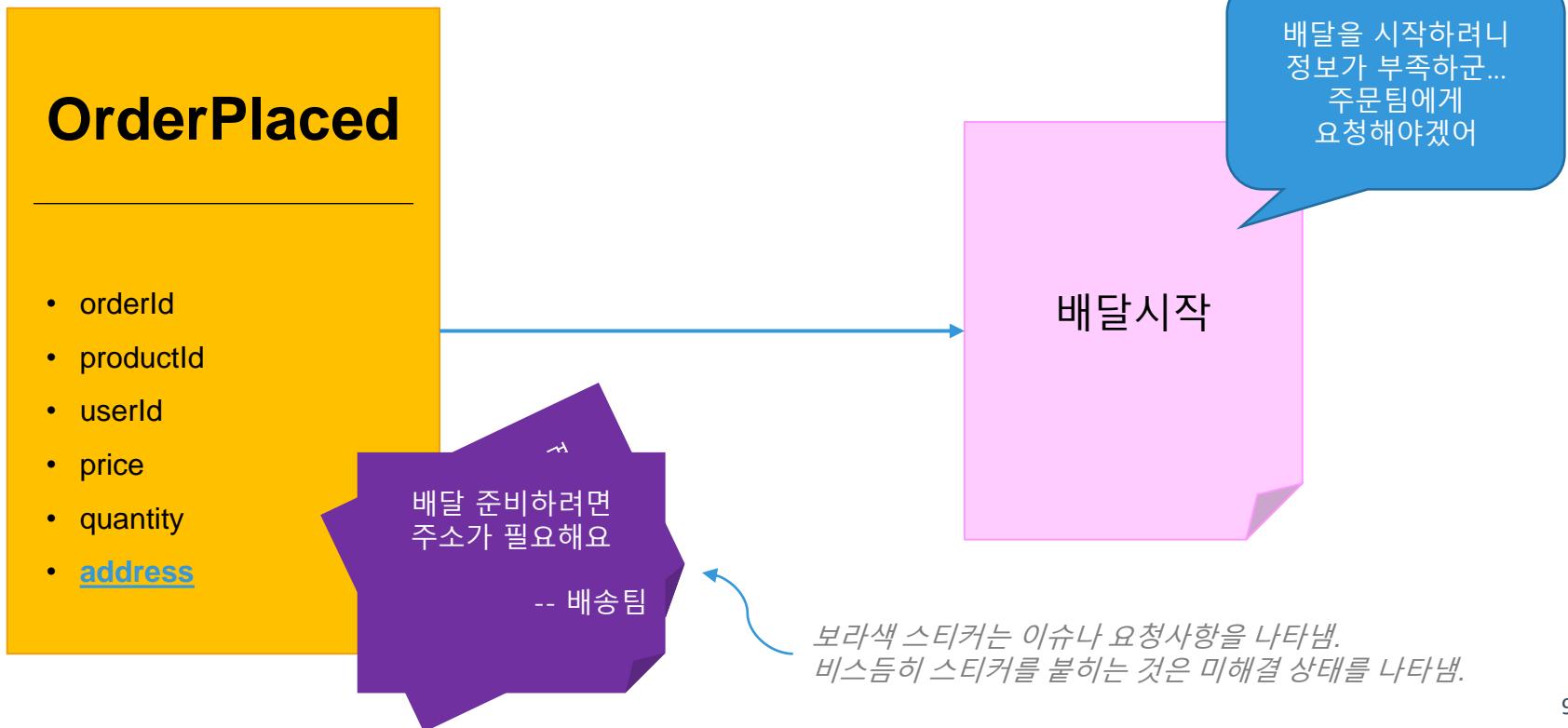
PubSub via Local Event Design Pattern
(e.g. White-board Pattern)

Lab Time – Bounded Context 분리 (다음스프린트)

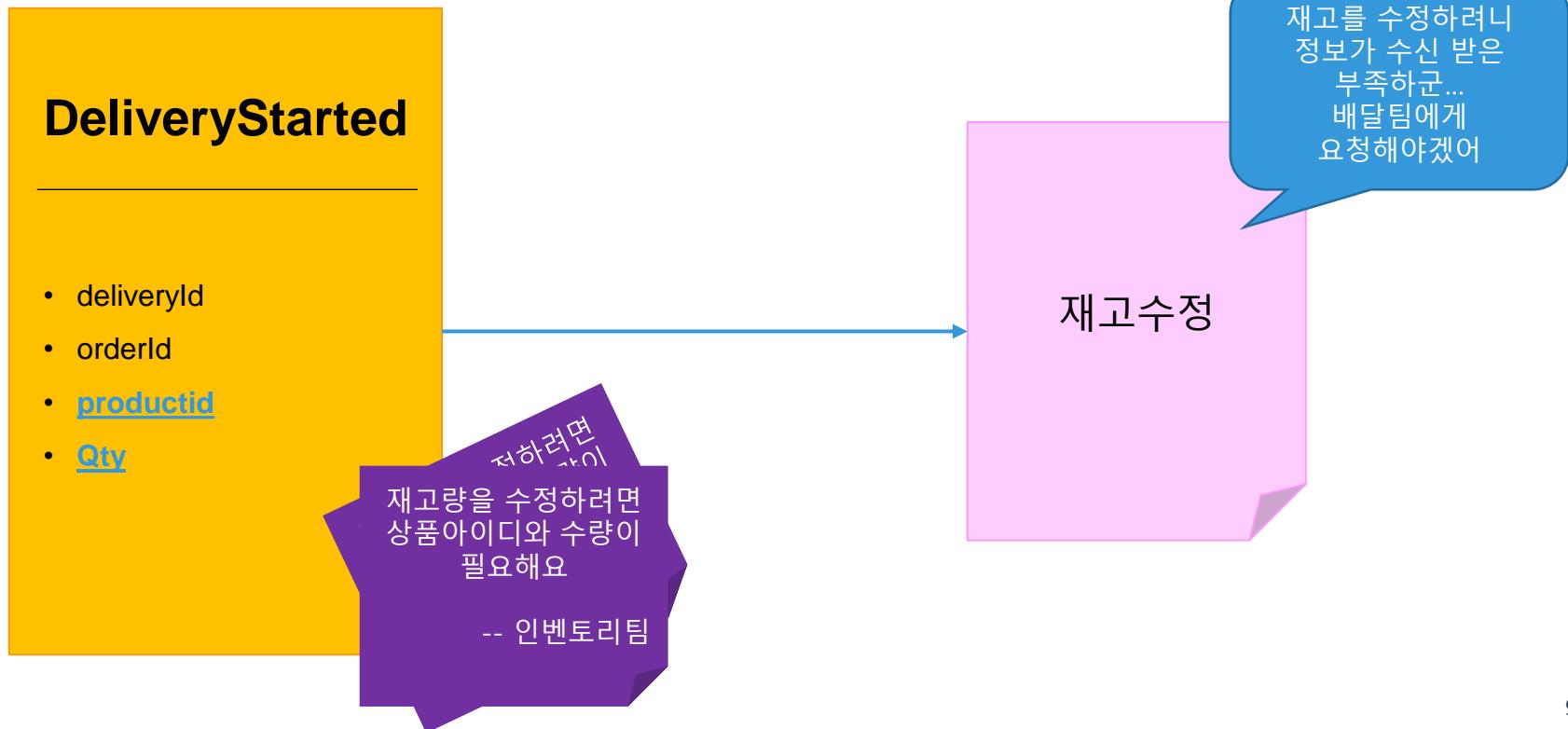


PubSub via Event Stream
(e.g. Kafka, Axon Server)

Lab Time – 도메인 이벤트의 속성 선언



Lab Time – 도메인 이벤트의 속성 선언(2)



Quiz. 다음중 도메인 이벤트로 부적절한 것은?

O

상품주문이 발
생함

상품이
입고됨

상품정보 변경
됨

다른 팀에서 관
심 가질만함

다른 팀에서 관
심 가질만함

적당한 사이즈
임

X

상품주문

상품을 조회함

JPA 를 통해 상
품 정보를
Update 함

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

Tools

- www.msaez.io
- <https://miro.com>

Table of content

Microservice and
Event-storming-Based
DevOps Project

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

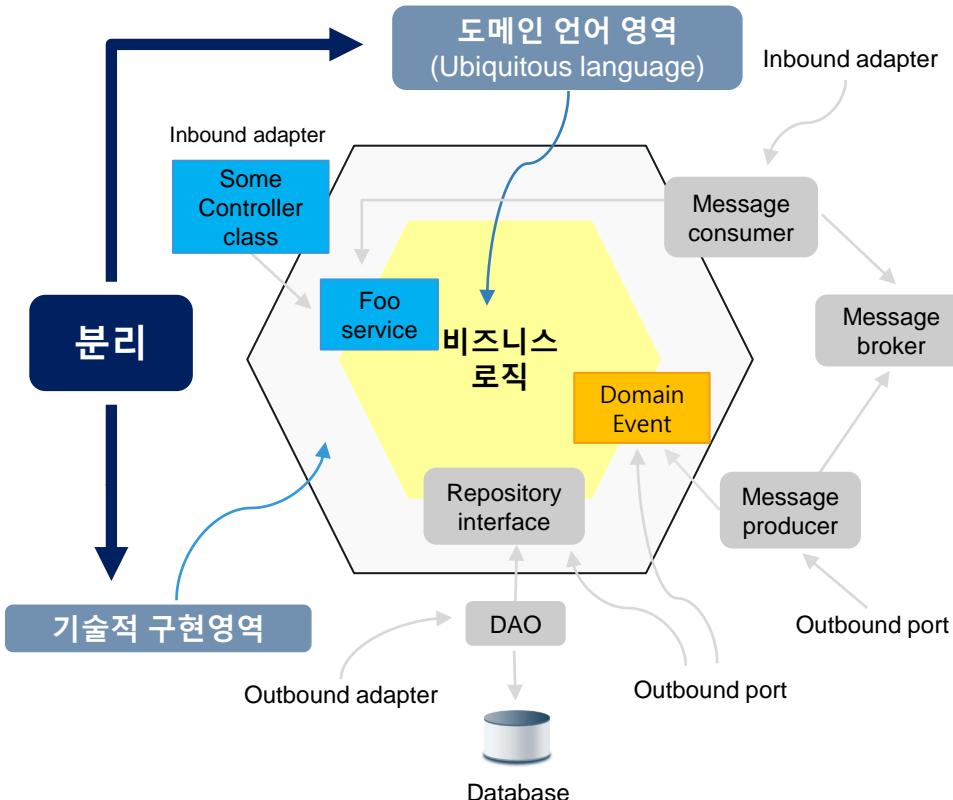
Microservice Implementation Pattern (1)

- **Hexagonal Architecture**

Create your service to be independent of either UI or database and to provide adapters for different input/output sources such as GUI, DB, test harness, RESTful resource, etc.

Implement the publish-and-subscribe messaging pattern: As events arrive at a port, an adapter (a.k.a. service agent) converts it into a procedure call or message and passes it to the application. When the application has something to publish, it does it through a port to an adapter, which creates the appropriate signals needed by the receiver.

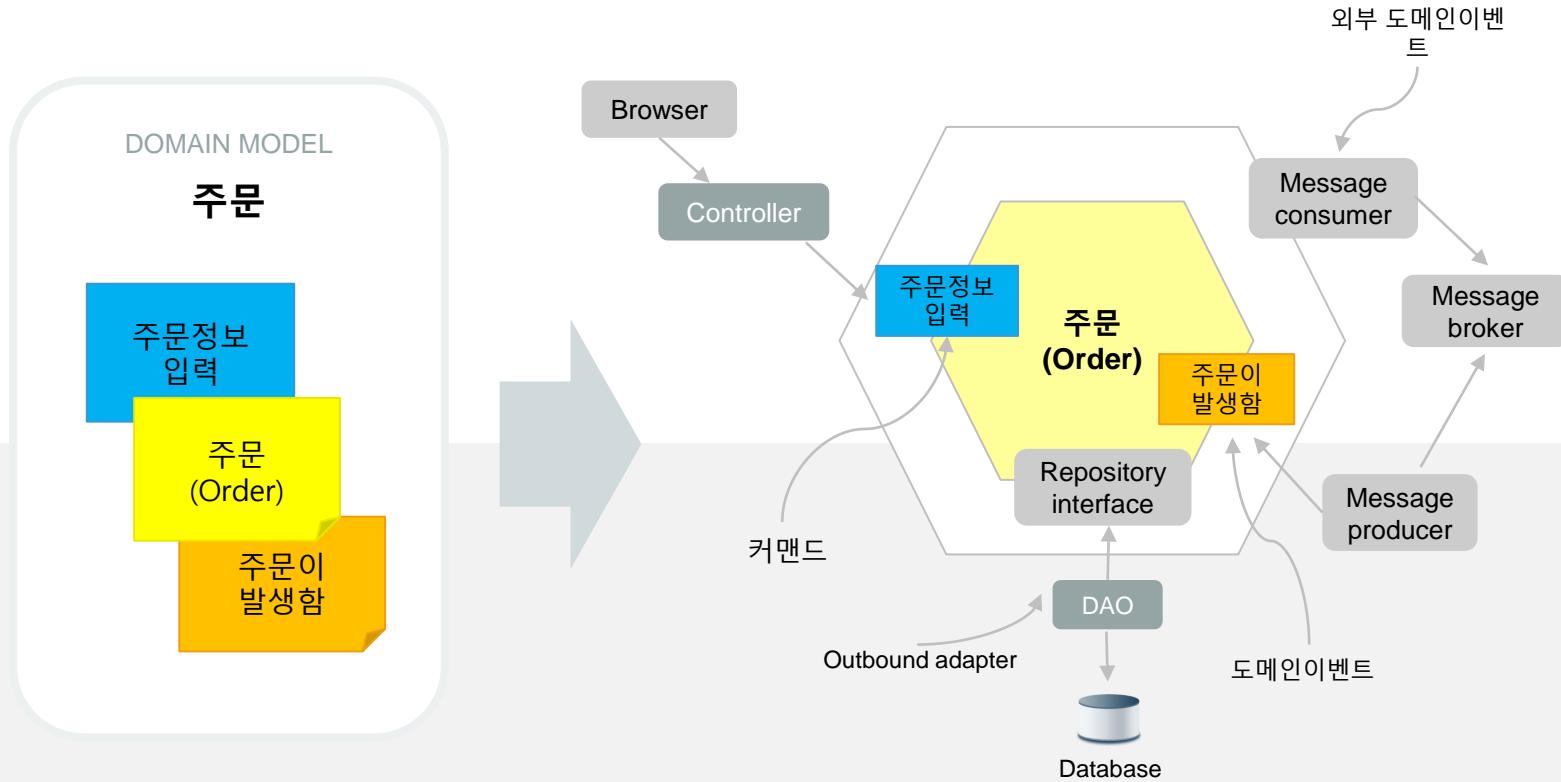
<https://alistair.cockburn.us/Hexagonal+architecture>



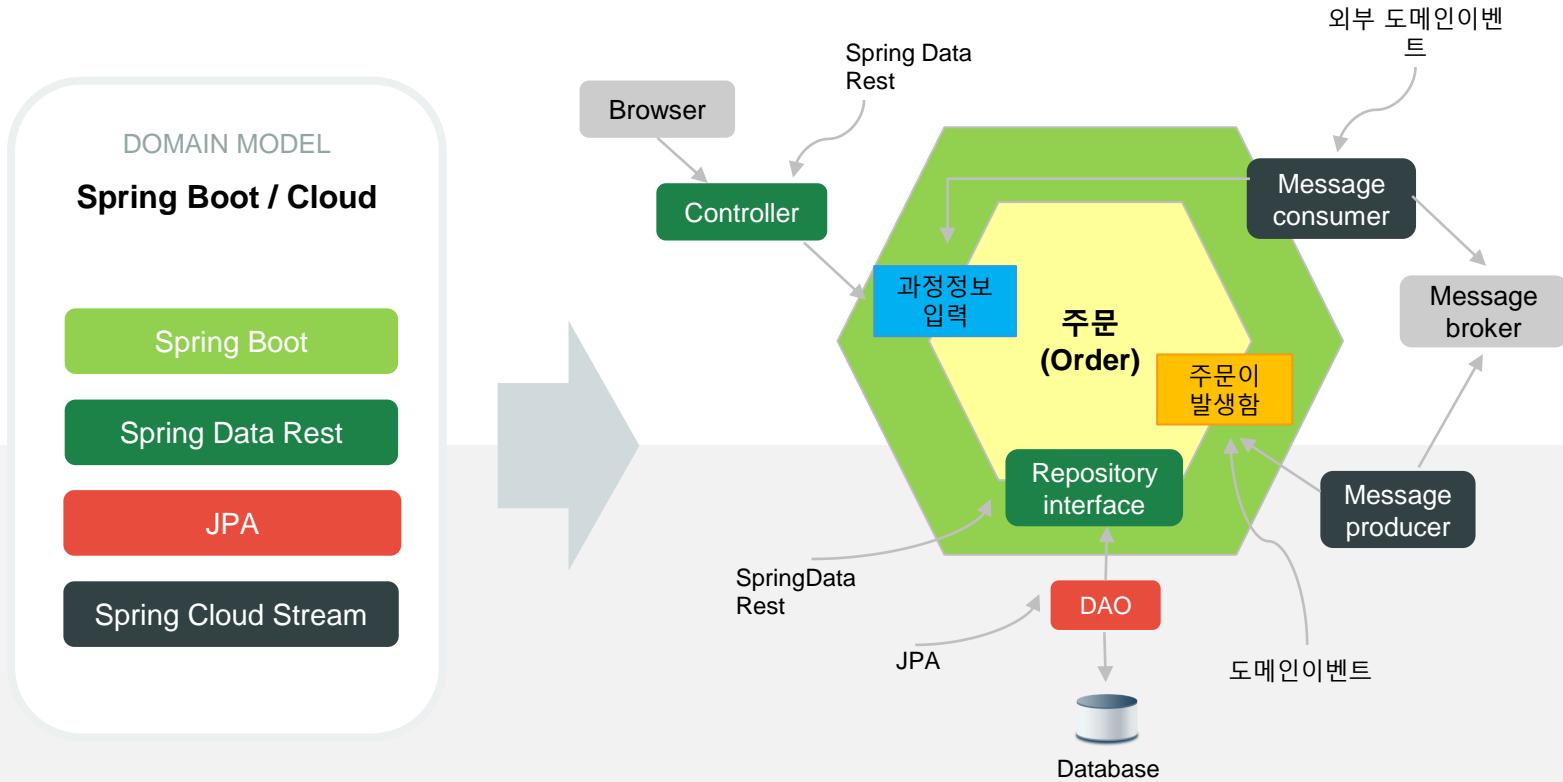
Service Implementation

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

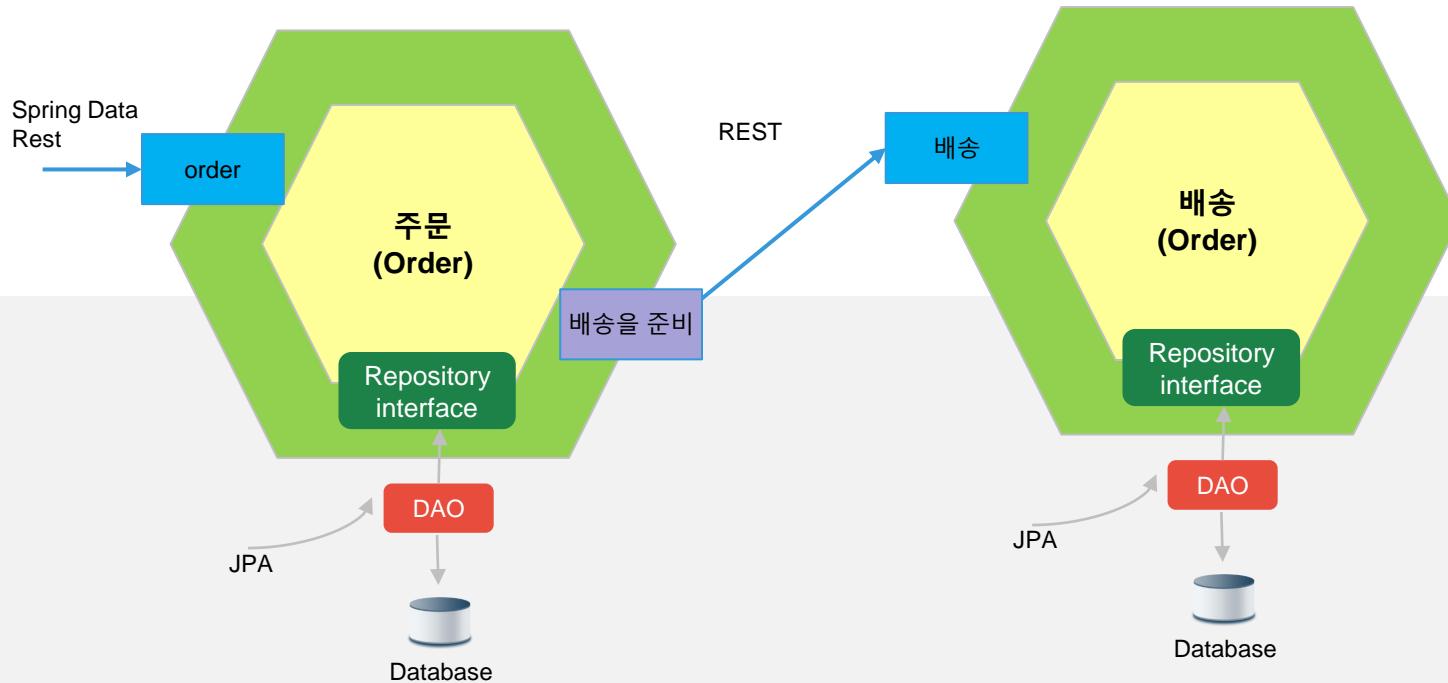
Applying Hexagonal Architecture



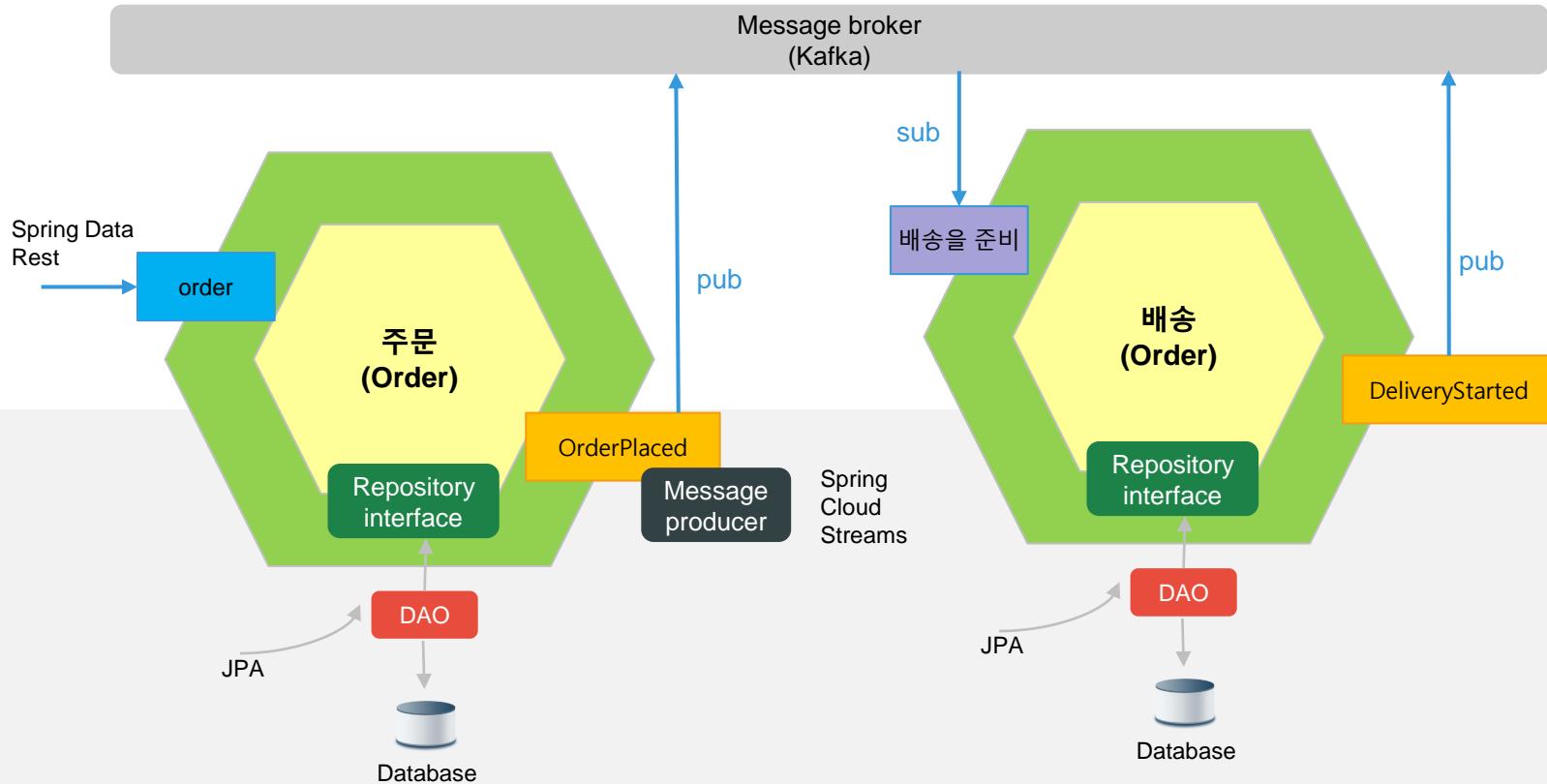
Applying MSA Chassis



Applying MSA Chassis



Applying MSA Chassis



이벤트 스토밍 결과에서 구현 기술 연동

<u>요소</u>	<u>구현체</u>	<u>미들웨어/프레임워크</u>
이벤트 (Domain Event)	<ul style="list-style-type: none">도메인 이벤트 클래스 (POJO)	<ul style="list-style-type: none">카프카 퍼블리시Rabbit MQ
커맨드 (Command)	<ul style="list-style-type: none">서비스 객체레포지토리 객체 (PagingAndSortingRepository)	<ul style="list-style-type: none">Spring Data RESTRESTEasy
결합물 (Aggregate)	<ul style="list-style-type: none">엔티티 클래스 (ORM)도메인 모델	<ul style="list-style-type: none">JPA EntityValue Objects
정책 (Policy)	<ul style="list-style-type: none">엔티티 클래스의 Hook에 정책 호출 구문 주입CDC 를 통한 이벤트 후크	<ul style="list-style-type: none">카프카 Event Listening CodeJPA Lifecycle HookCDC (Change Data Capturing)
바운디드 컨텍스트	<ul style="list-style-type: none">마이크로 서비스 (후보)	<ul style="list-style-type: none">Spring Boot

분석/설계 (Sticker)

Aggregate Root

Order

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



```
@Entity  
public class Order{  
  
    @Id Long id;  
    Long productId;  
    String customerId;  
    Money price;  
    int quantity;  
  
    ... setter/getters ...  
  
}
```

Aggregate Members

```
@Entity  
public class OrderDetail {  
  
    ....  
  
    ... setter/getters ...  
  
}
```

```
//Value Objects  
public class Money {  
  
    ....  
  
    ... setter/getters ...  
  
}
```

주문
(POST)

주문수정
(PATCH)

주문삭제
(DELETE)

Command □ CRUD에 해당하면? Repository Pattern으로 자동생성

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

Command □ Repository Pattern이 안될시에 MVC 패턴으로 구현

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

분석/설계 (Sticker)

OrderPlaced

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

개발 (POJO)

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

실행 (JSON)

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

카
프
카



Order

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

- publishOrderPlaced()



Event □ POJO Class 와 이벤트 발사로

```
@Entity
public class Order {
    ...
    @PostPersist // 주문이 저장된 후에
    private void publishOrderPlaced() {
        OrderPlaced orderPlaced = new OrderPlaced(); // 주문이 들어온 사실을
        이벤트로 작성
        orderPlaced.setOrderId(id);
        ...
        orderPlaced.publish(); // 메시지 큐에 주문이 들어왔음을 신고
    }
}
```

(Whenever OrderPlaced)

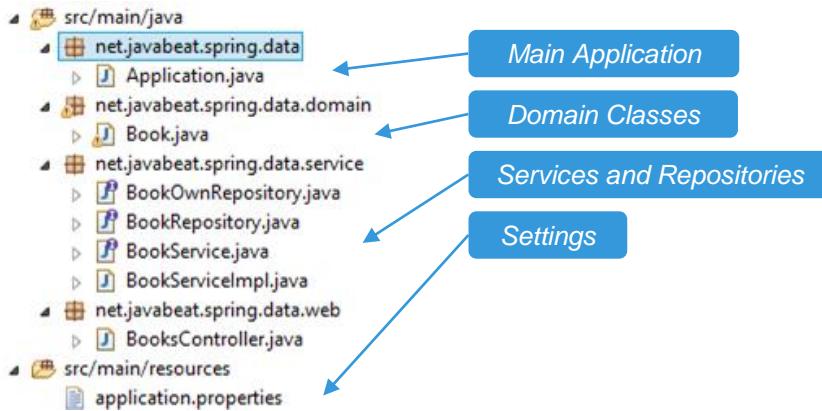
배송을 준비함

```
@KafkaListener(topics = "shopping")
public void onOrderPlaced(OrderPlaced orderPlaced)

    deliveryService.start(orderPlaced.getOrderId());

}
```

Spring Boot : A MSA Chassis



- Simple Java (POJO)
- Minimal Understanding Of Frameworks and Platforms (Using Annotations)
- No Server-side GUI rendering (Only Exposes REST Service)
- No WAS deployment required (Code is server; Tomcat embedded)
- No XML-based Configuration



Lab : Create a Spring boot application

The screenshot shows the Spring Initializr web interface at start.spring.io. The configuration is as follows:

- Project:** Maven Project
- Language:** Java
- Spring Boot:** 2.1.8 (selected)
- Project Metadata:** Group: com.12st, Artifact: delivery
- Dependencies:** H2 Database, Rest Repositories, Spring Data JPA

At the bottom, there are buttons for "Generate the project - ⌘ + ⌂" and "Explore the project - Ctrl + Space".

Go to start.spring.io

Set metadata:

- Group: com.12st
- Artifact: order
- Dependencies:
 - H2
 - Rest Repositories
 - JPA

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

Port 충돌시:

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

Running Spring Boot Application

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

```
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] -----
[INFO] Building ....
[INFO] -----
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/springframework/security/spring-security-core/maven-metadata.xml
[INFO] Downloading: https://oss.sonatype.org/content/repositories/snapshots/org/springframework/security/spring-security-core/maven-metadata.xml
[INFO] Downloading: https://repo.spring.io/libs-release
:
Started Application in 11.691 seconds (JVM running for 14.505)
```

Test Generated Services

```
$ http localhost:8080
```

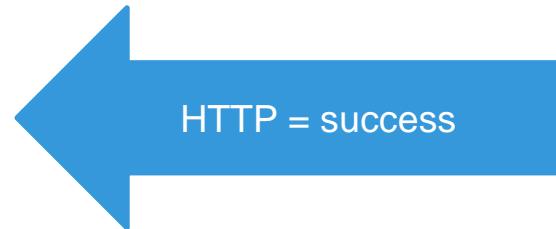
HTTP/1.1 200

Content-Type: application/hal+json; charset=UTF-8

Date: Wed, 05 Dec 2018 04:20:52 GMT

Transfer-Encoding: chunked

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



HTTP = success



HATEOAS links

Aggregate Root ℹ Entity Class 작성

상품

```
@Entity  
public class Product {  
  
    @Id  
    @GeneratedValue  
    private Long id;  
  
    String name;  
    int price;  
    int stock;  
  
    @OneToMany( cascade =  
    CascadeType.ALL, fetch =  
    FetchType.EAGER, mappedBy = "order")  
    List<Order> orders;  
}
```

주문

```
@Entity  
public class Order {  
  
    @Id  
    @GeneratedValue  
    private Long id;  
    private Long productId;  
    private String productName;  
    private int quantity;  
    private int price;  
    private String customerName;  
    private String customerAddr;  
  
    @ManyToOne  
    @JoinColumn(name="productId")  
    Product product;  
}
```

배송

```
@Entity  
public class Delivery {  
  
    @Id @GeneratedValue  
    private Long deliveryId;  
    private Long orderId;  
    private String customerName;  
    private String deliveryAddress;  
    private String deliveryState;  
  
    @OneToOne  
    Order order;  
}
```

Commands ⚡ API Implementation by Spring Data REST Repository

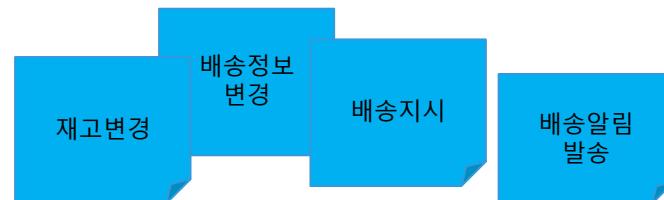
비즈니스 객체 (명사)에서 추출 가능한 CRUD의 경우



```
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 API는
Order Repository에 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"
    }
  }
}
```

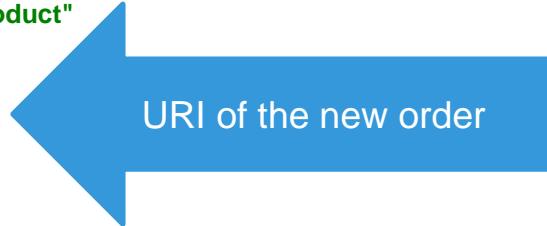


HATEOAS links

Create a new Order

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

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



URI of the new order

Create a delivery for the order

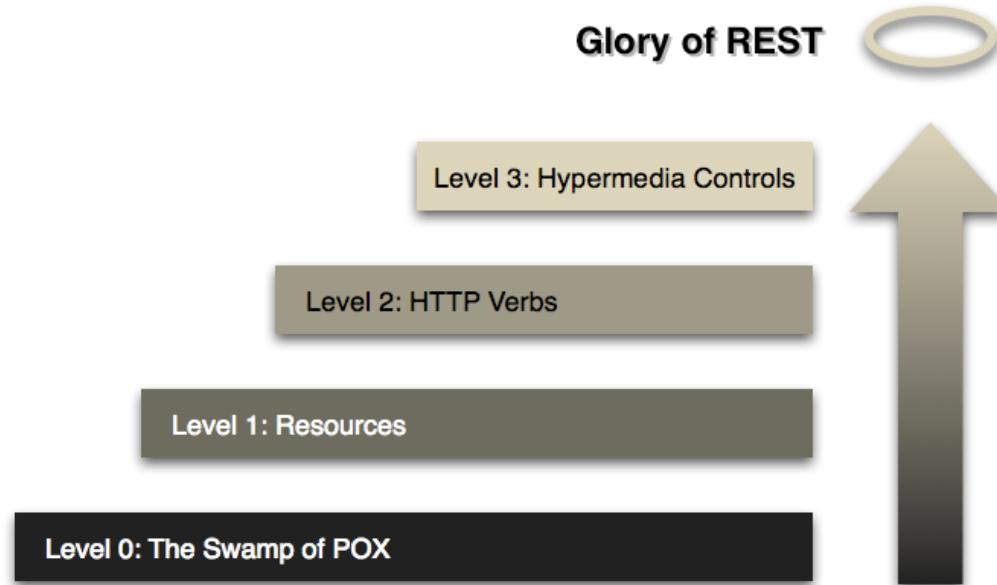
```
$ http http://localhost:8088/orders/1/delivery
```

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



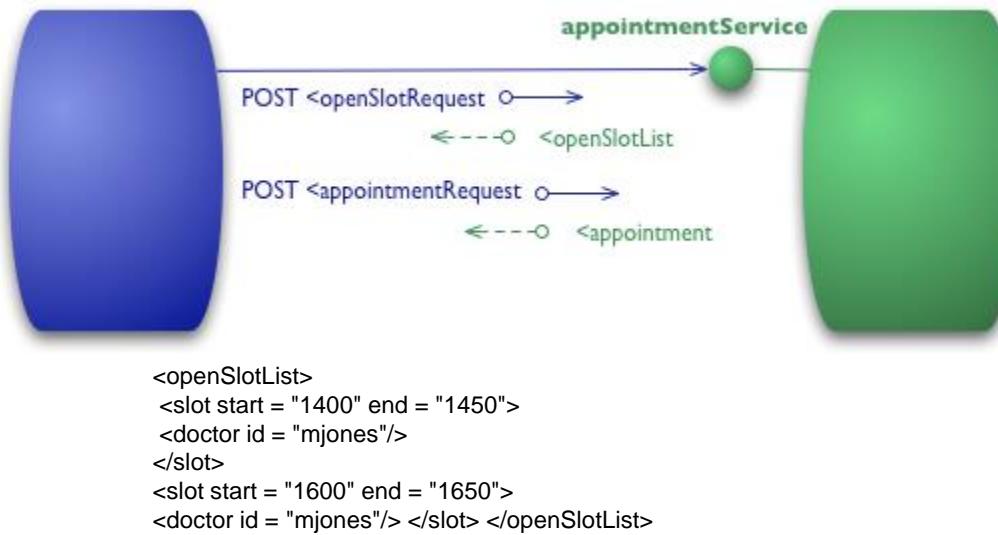
URI of the delivery service

Tip: REST Maturity Model



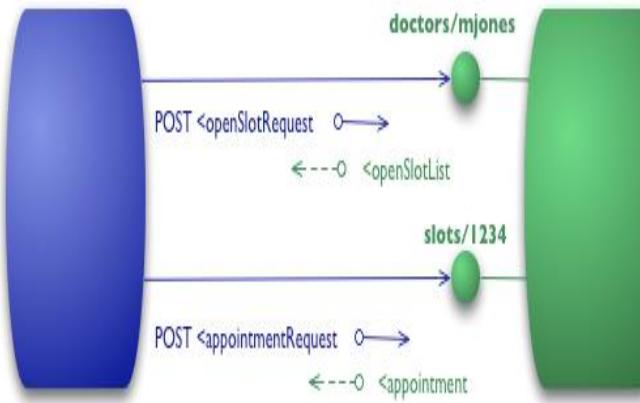
Level 0: Swamp of POX

- Use as a RPC, returns full serialized document



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.



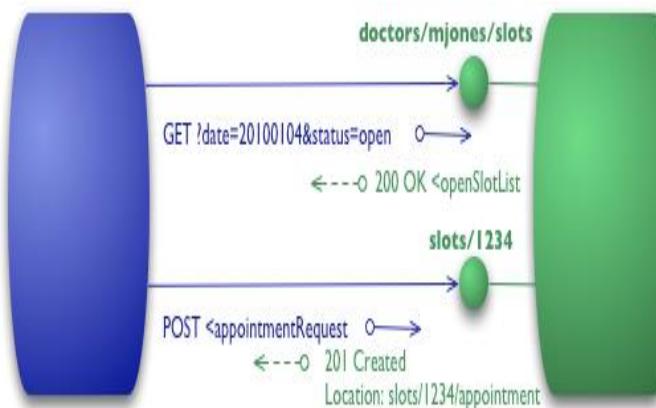
```
<openSlotList>
  <slot _link = "http://openslots/1234"/>
  <slot _link = "http://openslots/1235"/>
</openSlotList>
```

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

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

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.

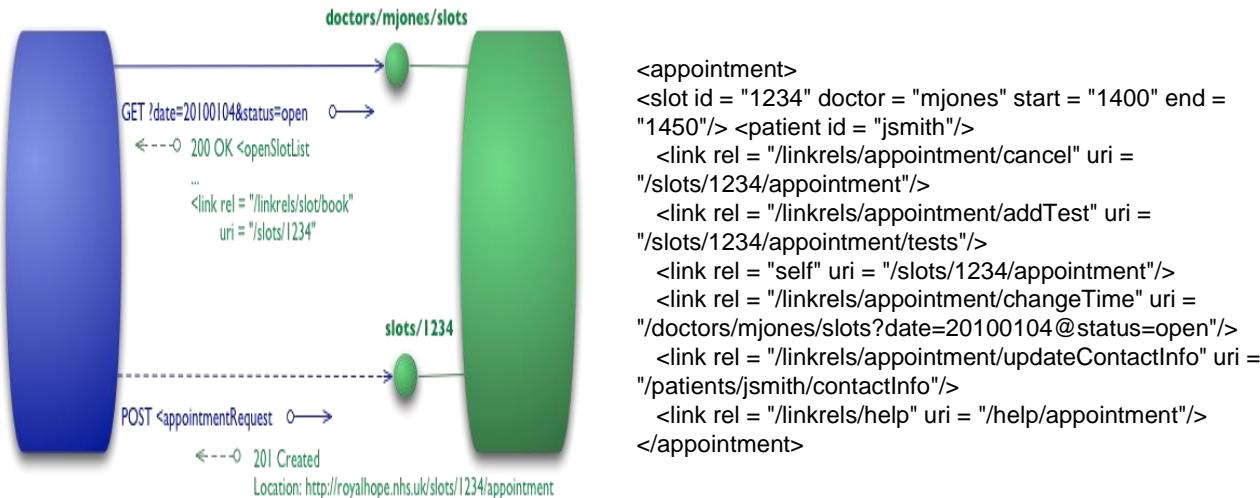


Operation	HTTP / REST
Create	PUT / POST
Read (Retrieve)	GET
Update (Modify)	PUT / PATCH
Delete (Destroy)	DELETE

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

Level 3: Hypermedia Controls

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

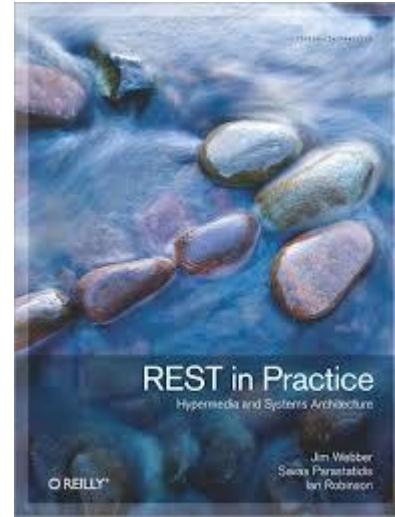


Level 3: In another words, HATEOAS

- 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.

Recommended Book

- REST in Practice
- Domain Driven Design Quickly





Quiz

스프링 부트/클라우드에 대한 설명 중 옳은 것은?

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

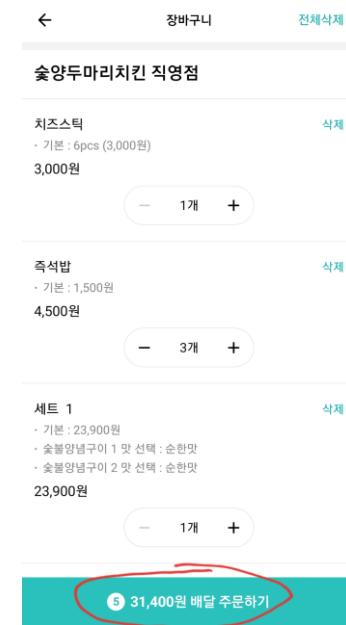
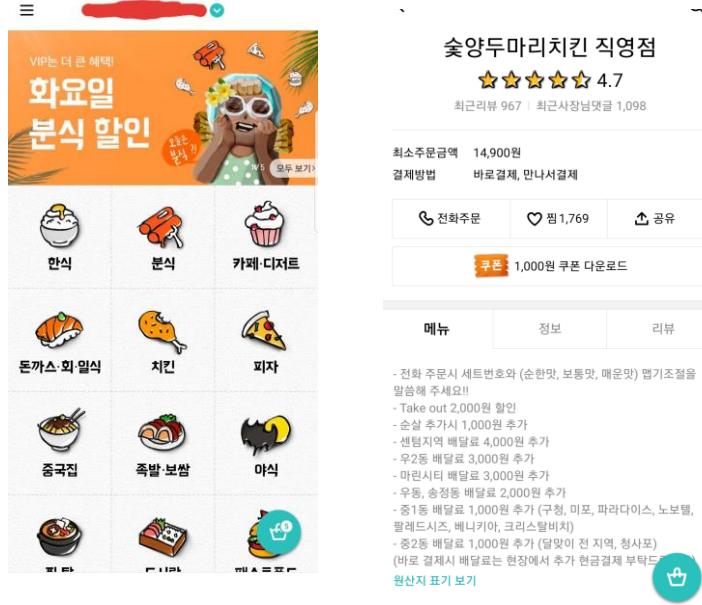
실습 예제

Food Delivery

<https://github.com/msa-ez/example-food-delivery>

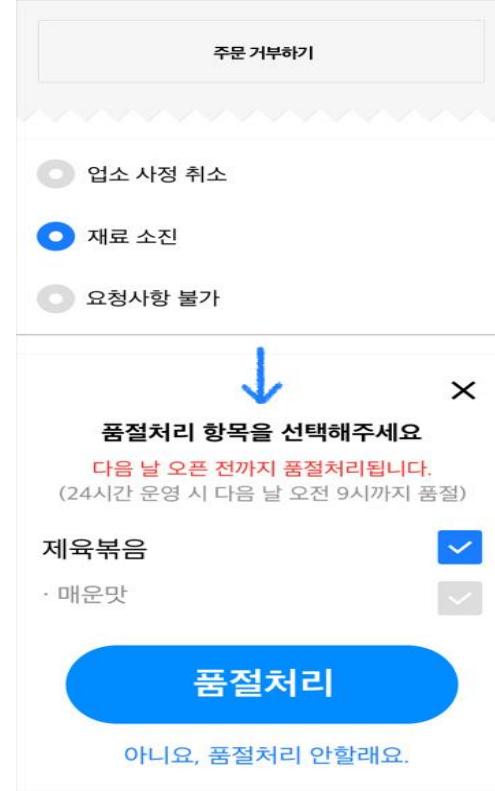
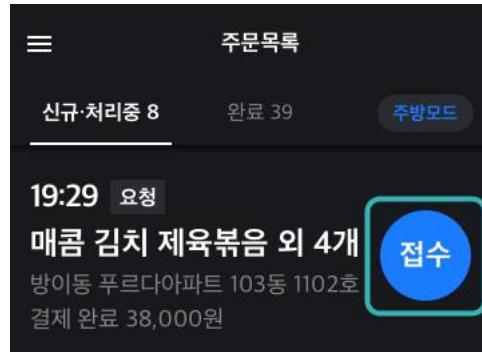
Context: Food Delivery App – Front

<https://github.com/msa-ez/example-food-delivery>



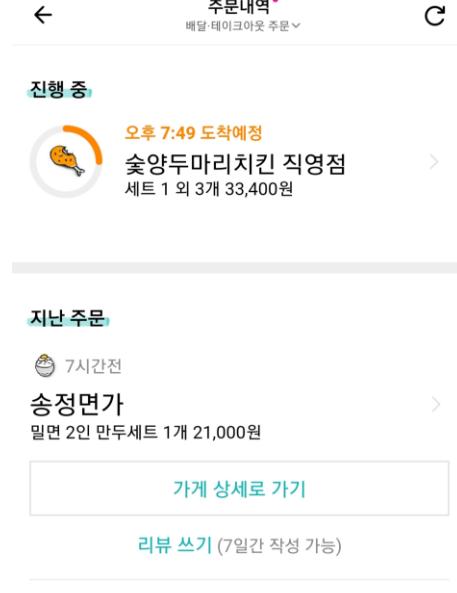
Context: Food Delivery App - Store

<https://github.com/msa-ez/example-food-delivery>



Context: Food Delivery App - Customer

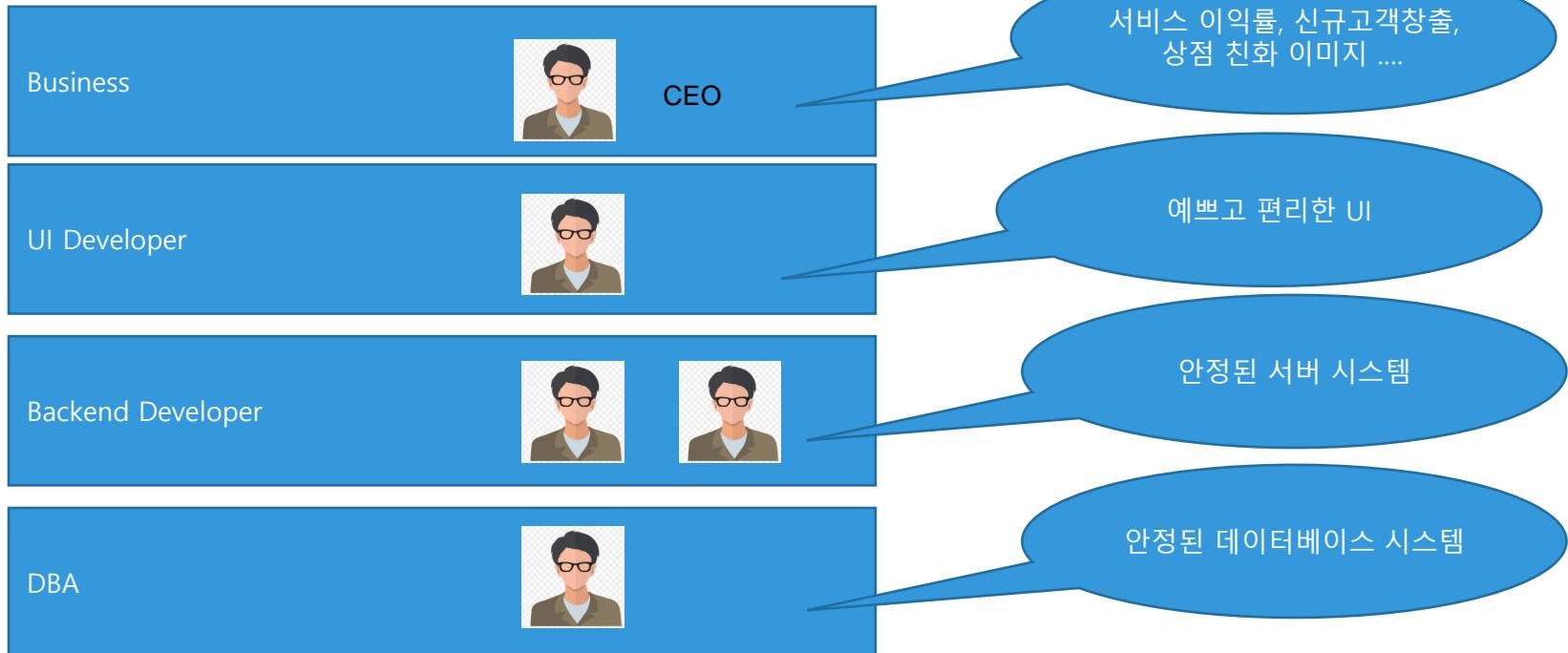
<https://github.com/msa-ez/example-food-delivery>



시나리오

1. 고객이 메뉴를 선택하여 주문한다
2. 고객이 결제한다
3. 주문이 되면 주문 내역이 입점상점주인에게 전달된다
4. 상점주인이 확인하여 요리해서 배달 출발한다
5. 고객이 주문을 취소할 수 있다
6. 주문이 취소되면 배달이 취소된다
7. 고객이 주문상태를 중간중간 조회한다
8. 주문상태가 바뀔 때 마다 카톡으로 알림을 보낸다

창업시기 조직구조 – Horizontal



조직구조 – Vertical



CEO



CTO



24시간
주문/결제

입점상인의
편익대변

고객만족도
향상

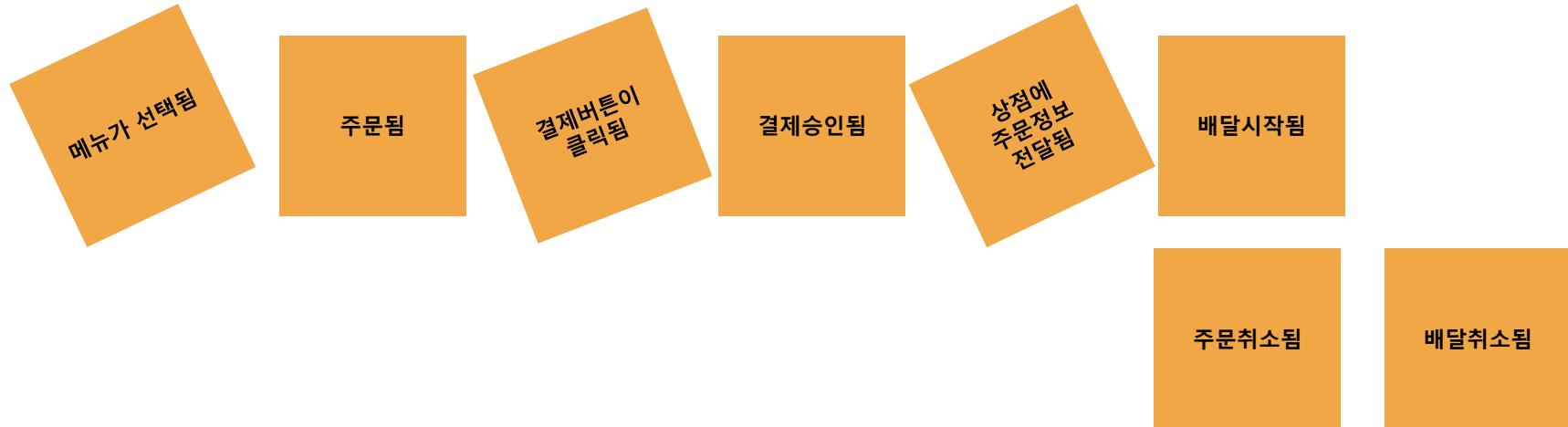
비기능적 요구사항

1. 트랜잭션
 1. 결제가 되지 않은 주문건은 아예 거래가 성립되지 않아야 한다
2. 장애격리
 1. 상점관리 기능이 수행되지 않더라도 주문은 365일 24시간 받을 수 있어야 한다 → Async (event-driven), Eventual Consistency
 2. 결제시스템이 과중되면 사용자를 잠시동안 받지 않고 결제를 잠시후에 하도록 유도한다 → Circuit breaker, fallback
3. 성능
 1. 고객이 자주 상점관리에서 확인할 수 있는 배달상태를 주문시스템(프론트엔드)에서 확인할 수 있어야 한다 → CQRS
 2. 배달상태가 바뀔때마다 카톡 등으로 알림을 줄 수 있어야 한다 → Event driven

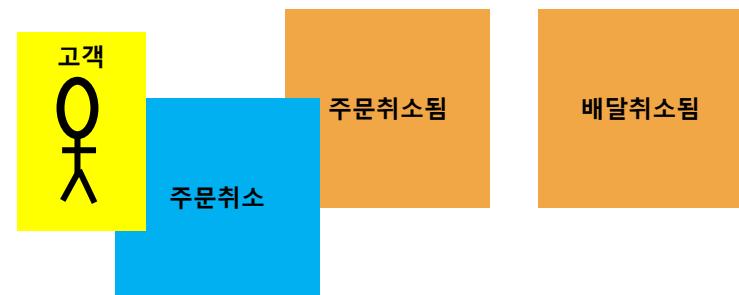
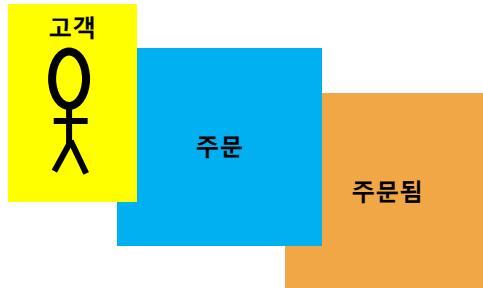
이벤트스토밍 - Event



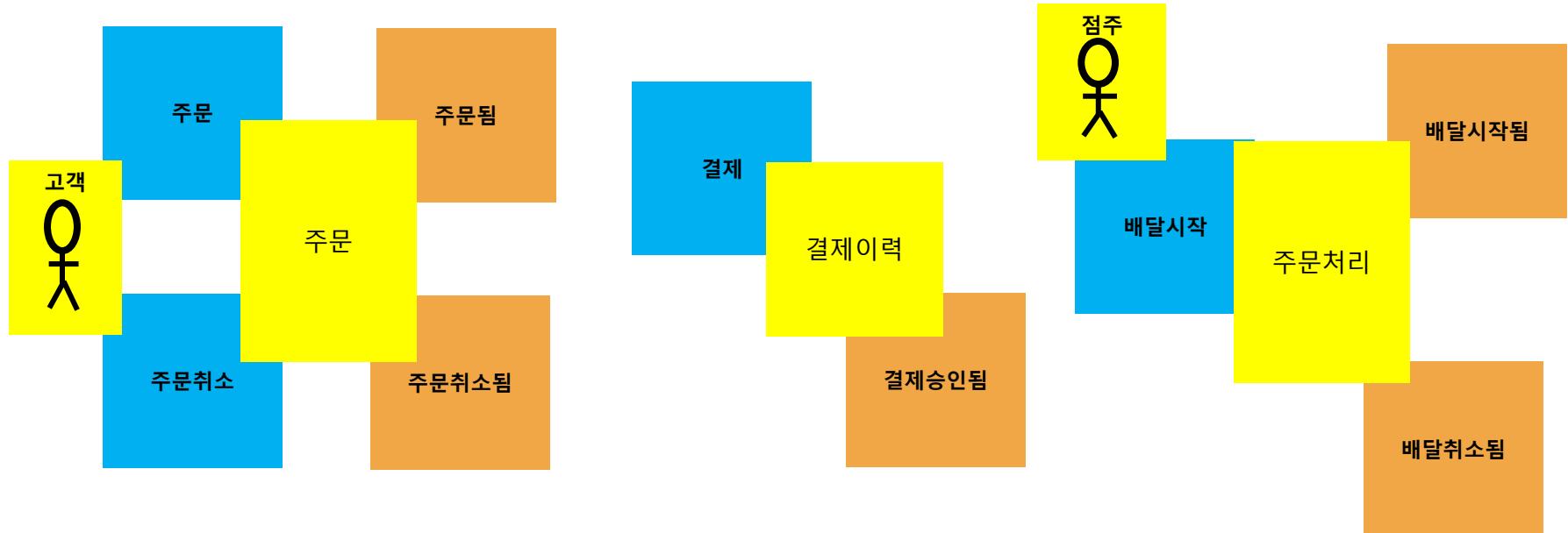
이벤트스토밍 – 비적격 이벤트 제거



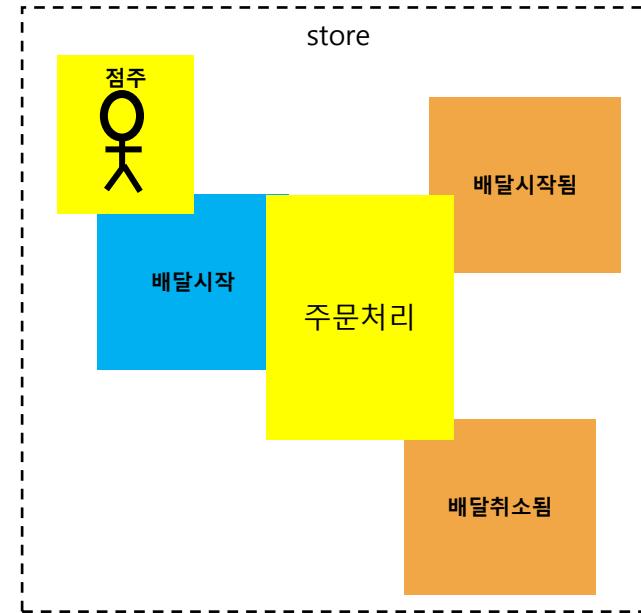
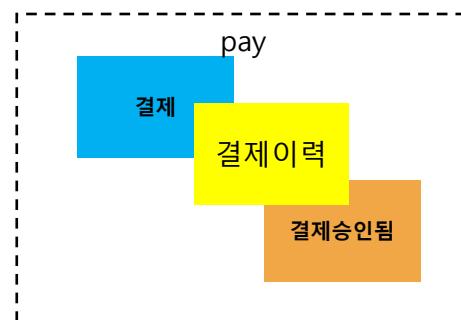
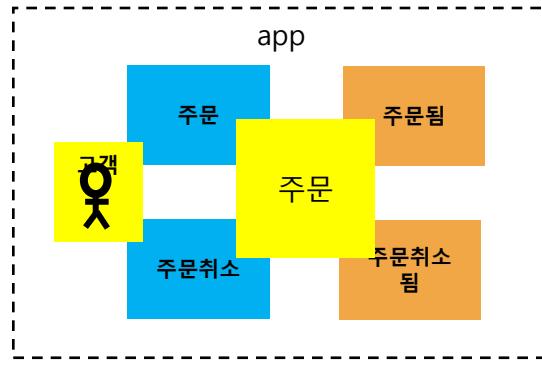
이벤트스토밍 – Actor, Command



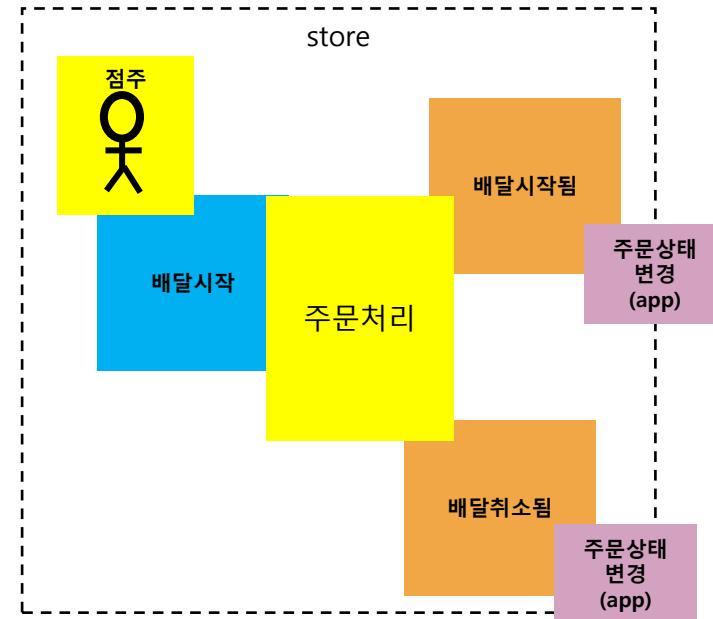
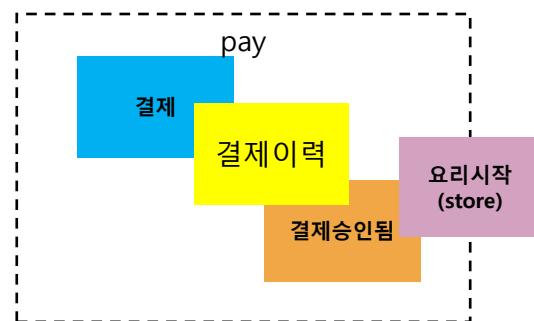
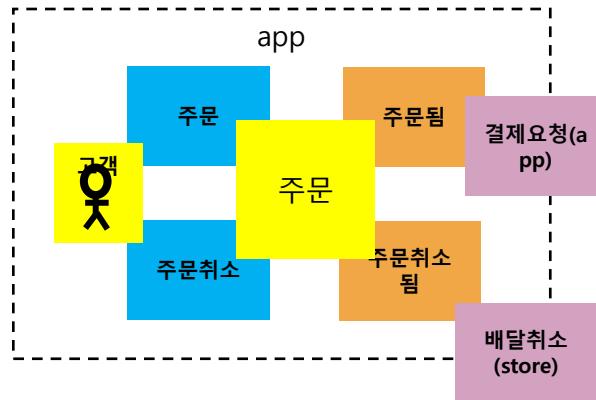
이벤트스토밍 – Aggregate



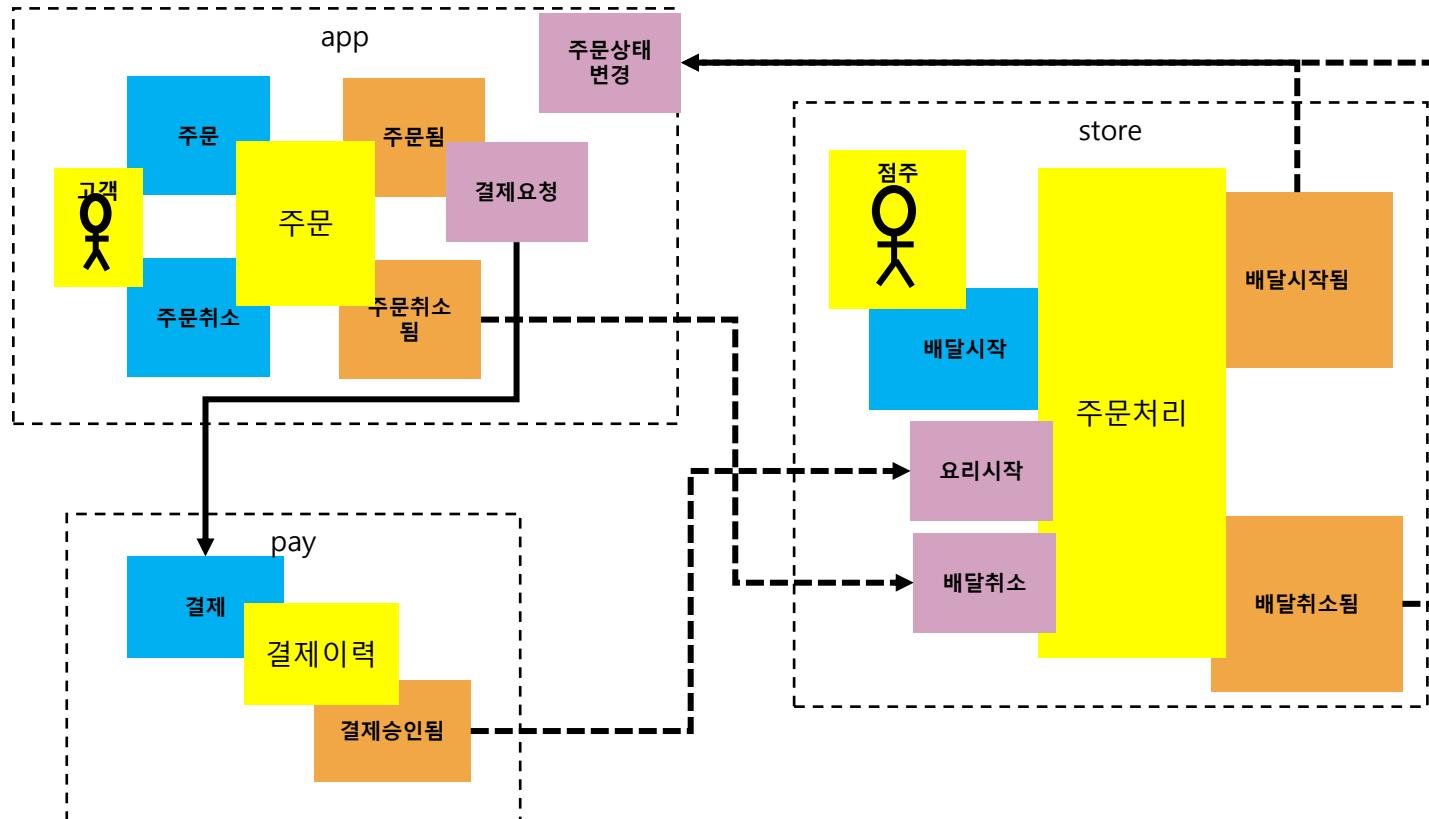
이벤트스토밍 – Bounded Context



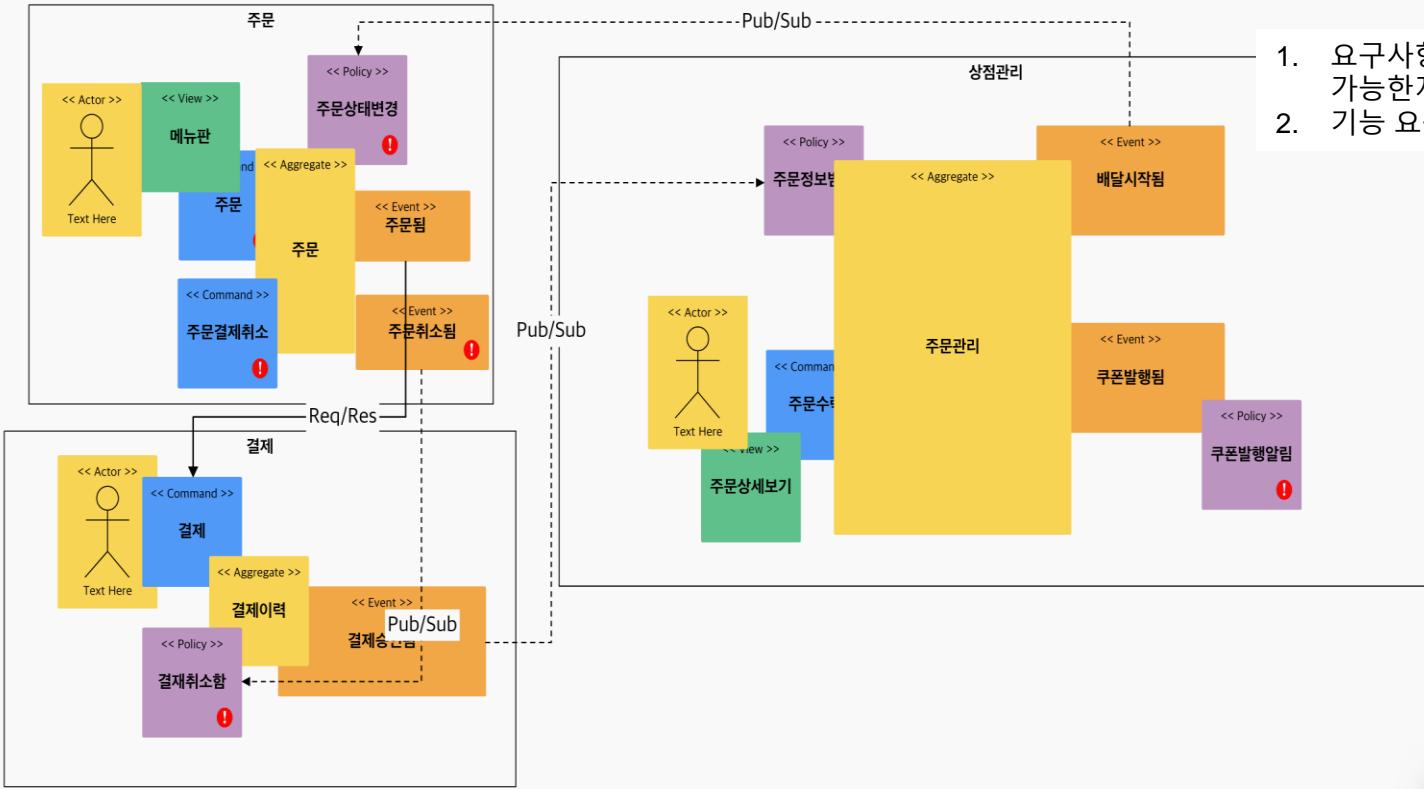
이벤트스토밍 – Policy (괄호 - 수행주체)



이벤트스토밍 – Policy 를 수행주체로 이동



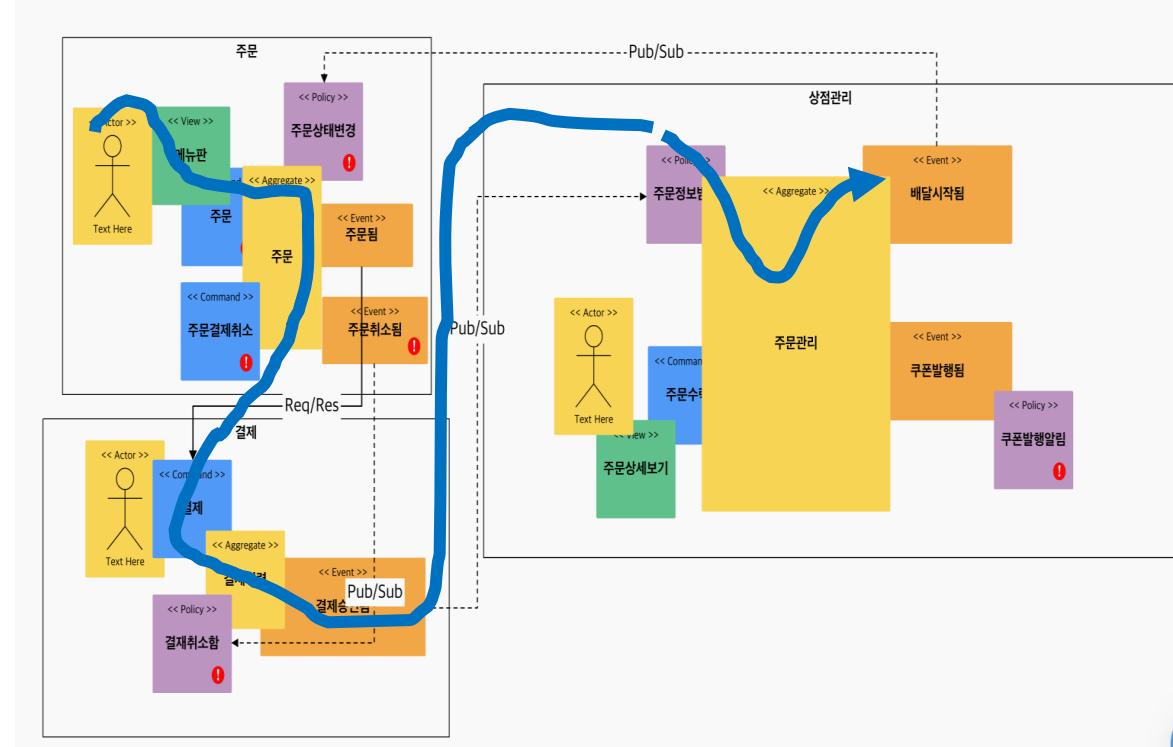
기능 요구사항 coverage



1. 요구사항별로 모든 나레이션이나 가능한지 검증함
2. 기능 요구사항별로 패스 표시

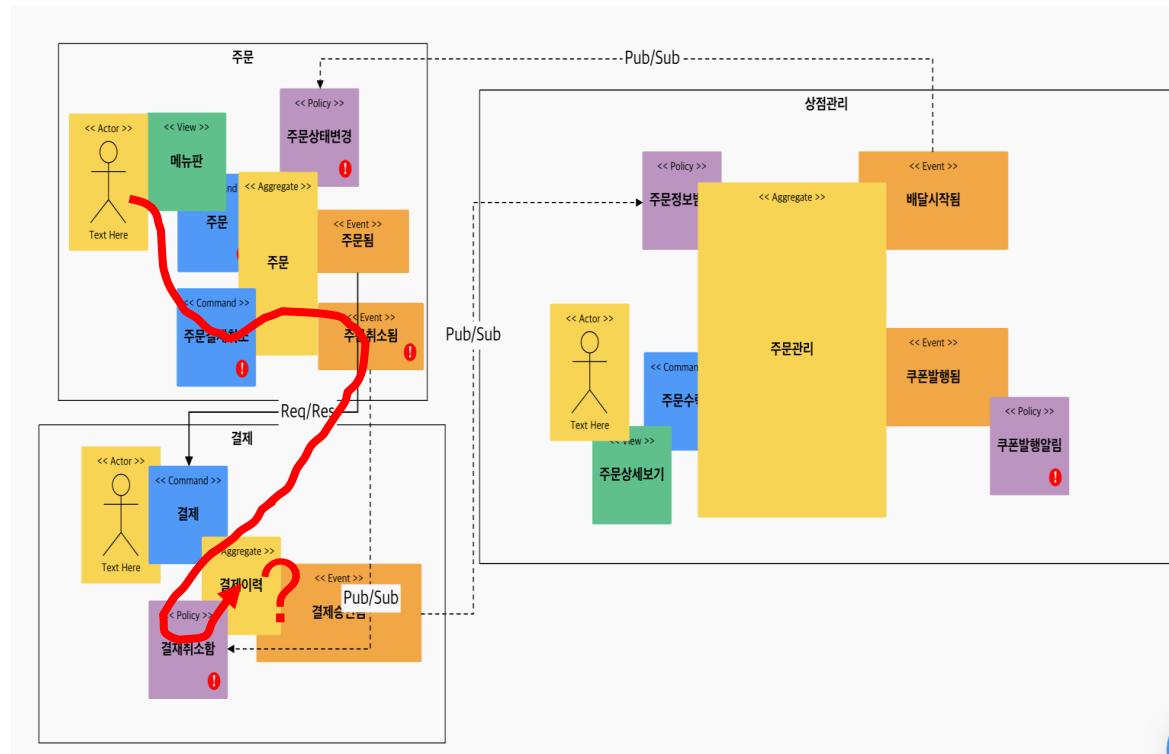
시나리오 Coverage Check (1)

1. 고객이 메뉴를 선택하여 주문 한다
2. 고객이 결제한다
3. 주문이 되면 주문 내역이 입점상점주인에게 전달된다
4. 상점주인이 확인하여 요리해서 배달 출발한다



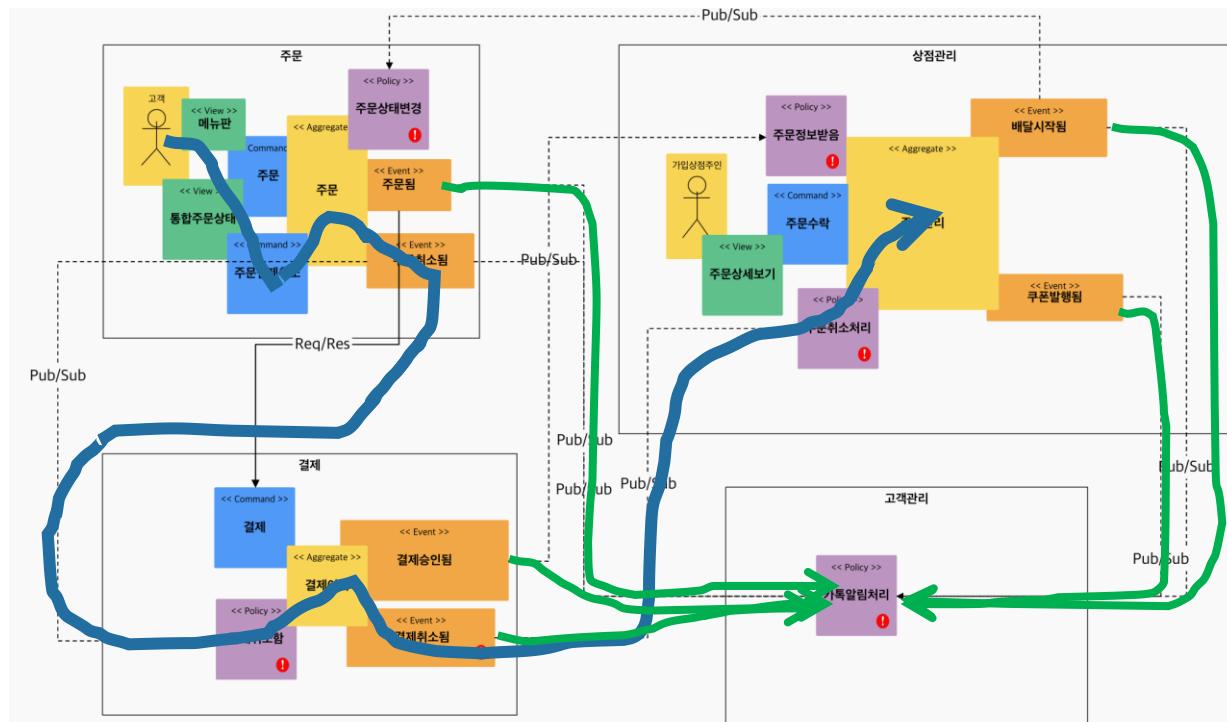
시나리오 Coverage Check (2)

5. 고객이 주문을 취소할 수 있다
6. 주문이 취소되면 배달이 취소된다 → 배달취소는?
7. 고객이 주문상태를 중간 중간 조회한다
8. 주문상태가 바뀔 때마다 카톡으로 알림을 보낸다
→ ?



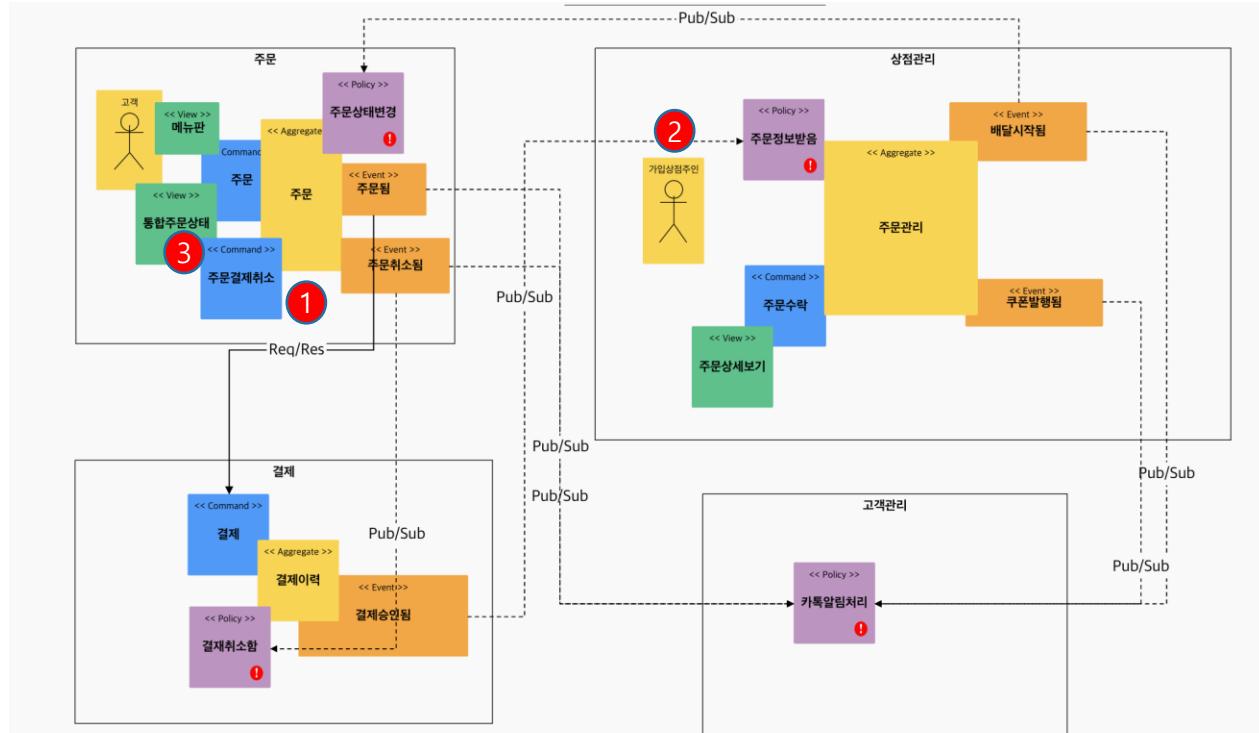
모델 업그레이드 - 요구사항 커버 확인

5. 고객이 주문을 취소할 수 있다
6. 주문이 취소되면 배달이 취소된다
7. 고객이 주문상태를 중간중간 조회한다
8. 주문상태가 바뀔 때마다 카톡으로 알림을 보낸다



비기능 요구사항 coverage

1. 주문에 대해서는 결제가 처리되어야만 주문 처리하고 장애격리를 위해 CB를 설치함
(트랜잭션 > 1, 장애격리 > 2)
2. 결제승인 이벤트를 수신하여 상점의 주문정보 변경을 수행함
(장애전파 > 1)
3. 상점의 배달관련 이벤트를 주문에서 수신하여 View Table 을 구성
(CQRS)
(성능 > 1)



헥사고날 아키텍처

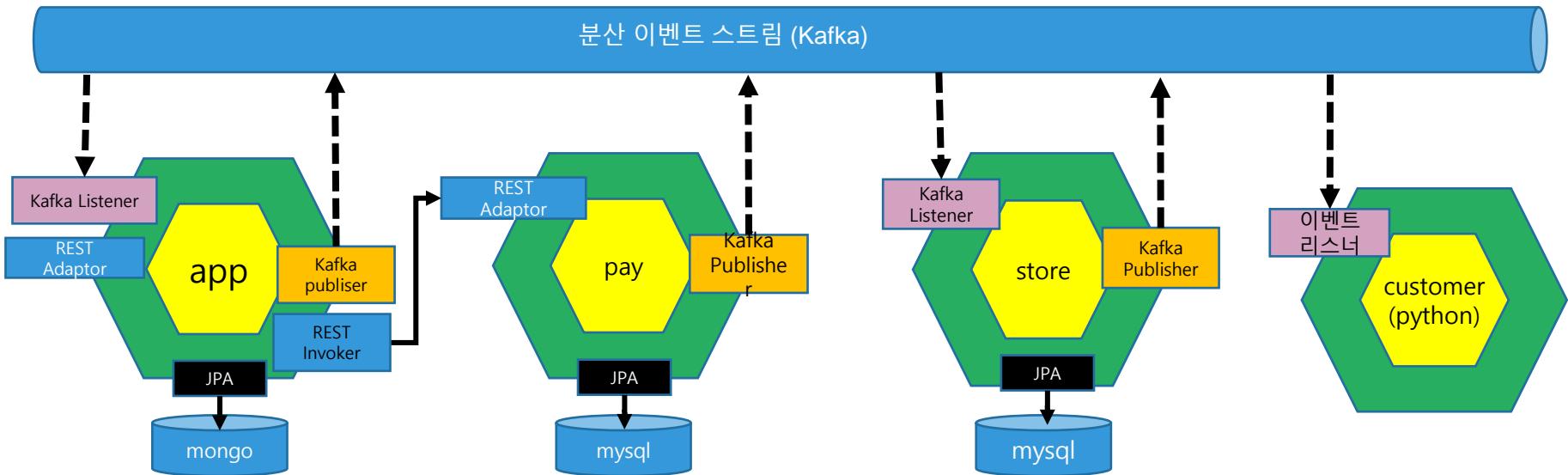


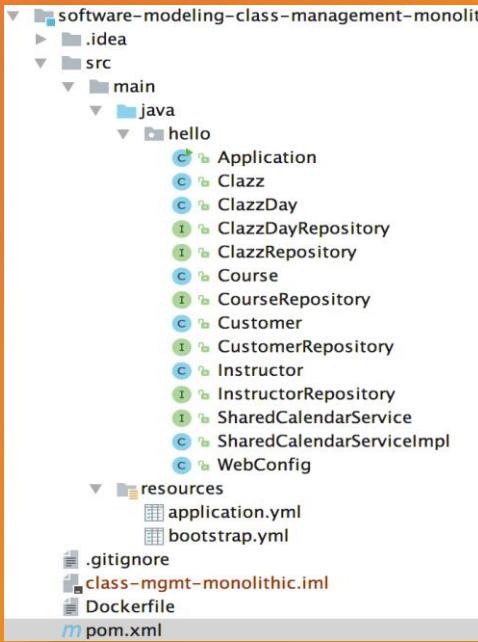
Table of content

Microservice and
Event-storming-Based
DevOps Project

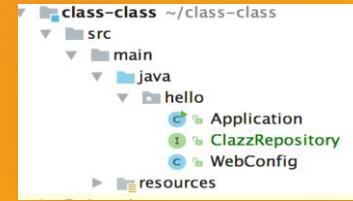
1. The Domain Problem : A Commerce Shopping Mall
2. Architecture and Approach Overview
3. Domain Analysis with DDD and Event Storming
4. Service Implementation with Spring Boot and Netflix OSS
5. Monolith to Microservices 
6. Front-end Development in MSA
7. Service Composition with Request-Response and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

From Monolith to Microservices

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



A screenshot of a code editor showing a monolithic Java application structure. The project is named "software-modeling-class-management-monolith". It contains a ".idea" folder, a "src" folder with a "main" package containing a "java" directory with a "hello" subdirectory, and a "resources" directory. The "hello" directory contains several Java files: Application, Clazz, ClazzDay, ClazzDayRepository, ClazzRepository, Course, CourseRepository, Customer, CustomerRepository, Instructor, InstructorRepository, SharedCalendarService, SharedCalendarServiceImpl, and WebConfig. The "resources" directory contains application.yml and bootstrap.yml files. At the bottom, there are .gitignore, class-mgmt-monolithic.iml, Dockerfile, and pom.xml files.



A screenshot of a code editor showing a microservice structure named "class-class". It has a "src" folder with a "main" package containing a "java" directory with a "hello" subdirectory. The "hello" directory contains Application, ClazzRepository, and WebConfig files. There is also a "resources" directory.

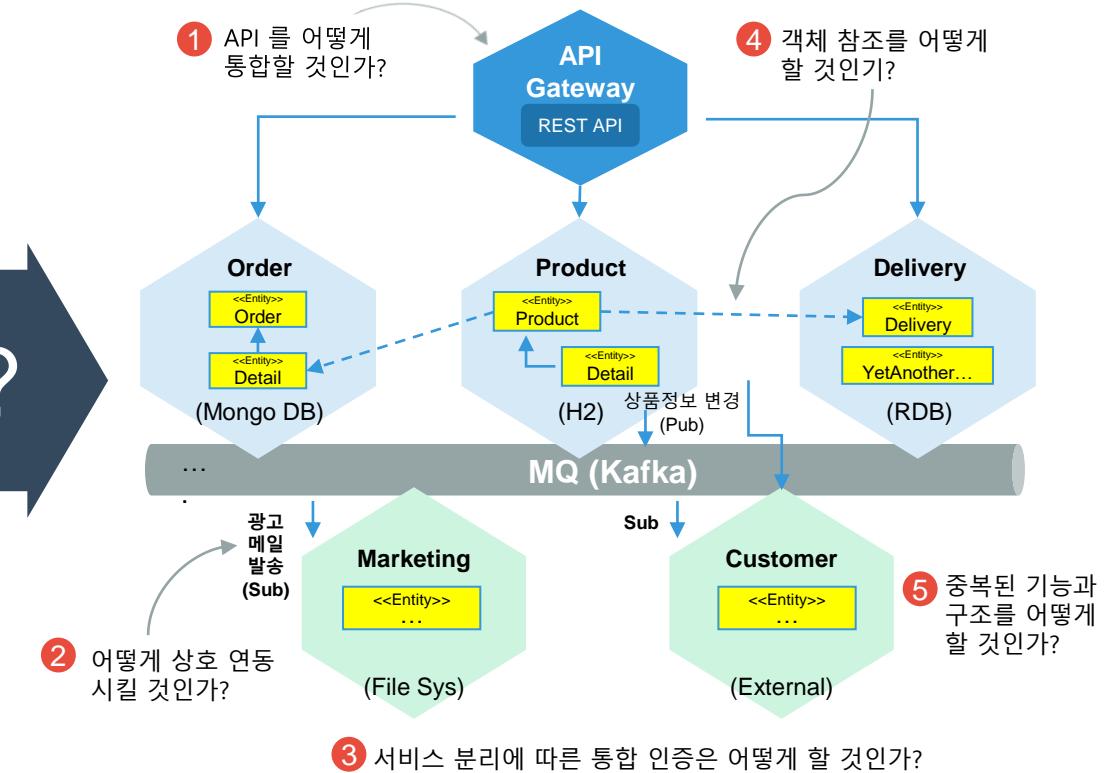
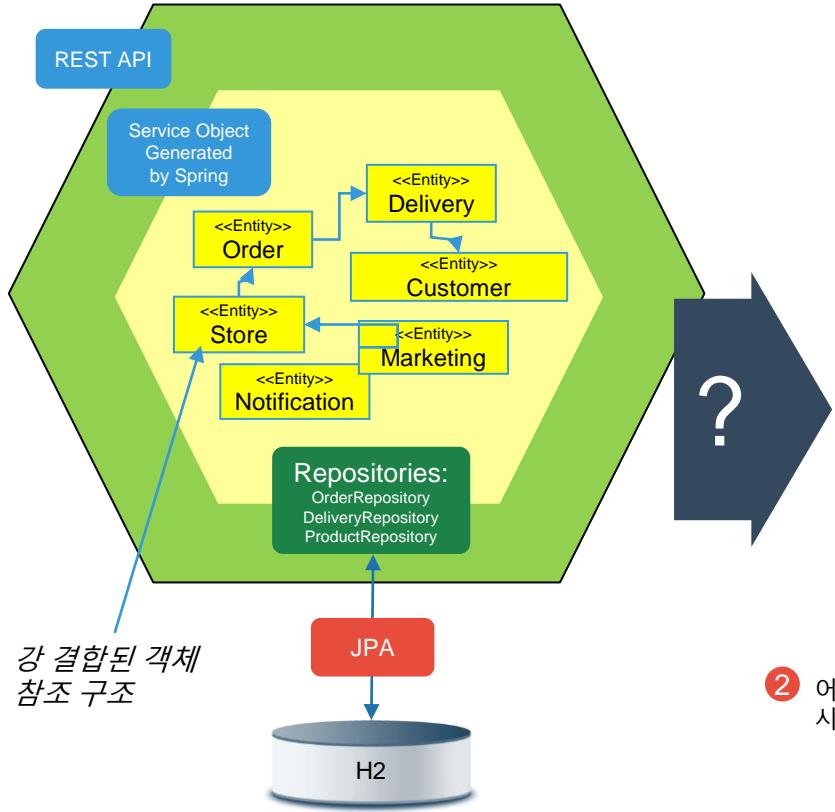


A screenshot of a code editor showing a microservice structure named "class-course". It has a "src" folder with a "main" package containing a "java" directory with a "hello" subdirectory. The "hello" directory contains Application, CourseRepository, and WebConfig files. There is also a "resources" directory.

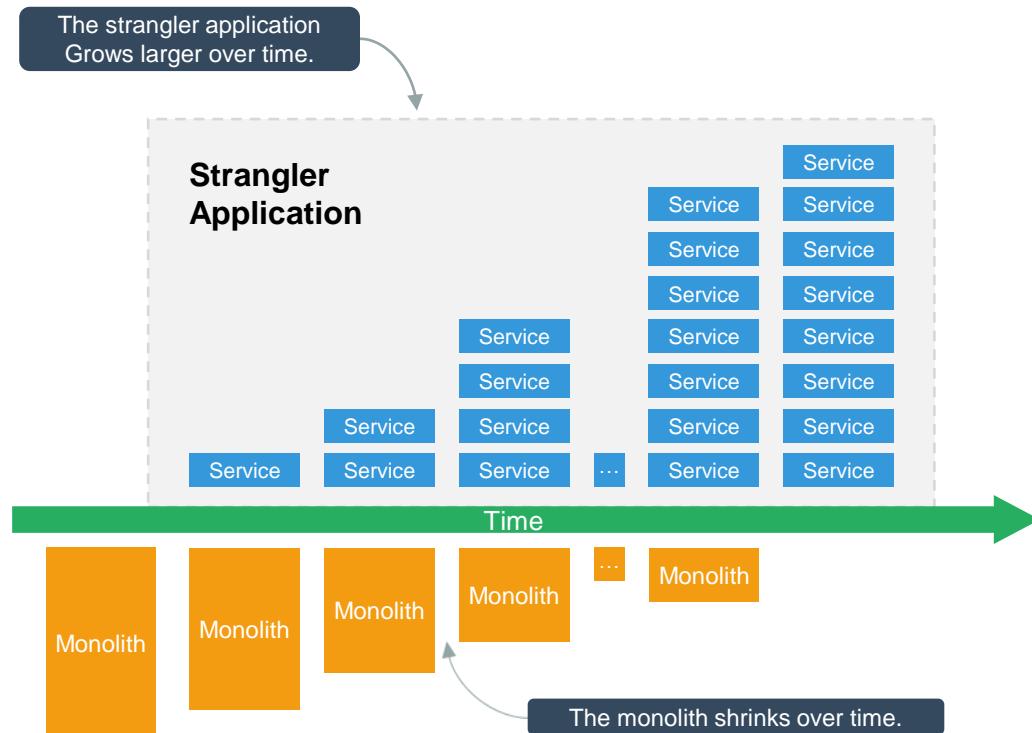


A screenshot of a code editor showing a microservice structure named "class-sme". It has a "src" folder with a "main" package containing a "java" directory with a "hello" subdirectory. The "hello" directory contains Application, InstructorRepository, and WebConfig files. There is also a "resources" directory.

Issues in transforming Monolith to Microservices



Legacy Transformation – Strangler Pattern



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

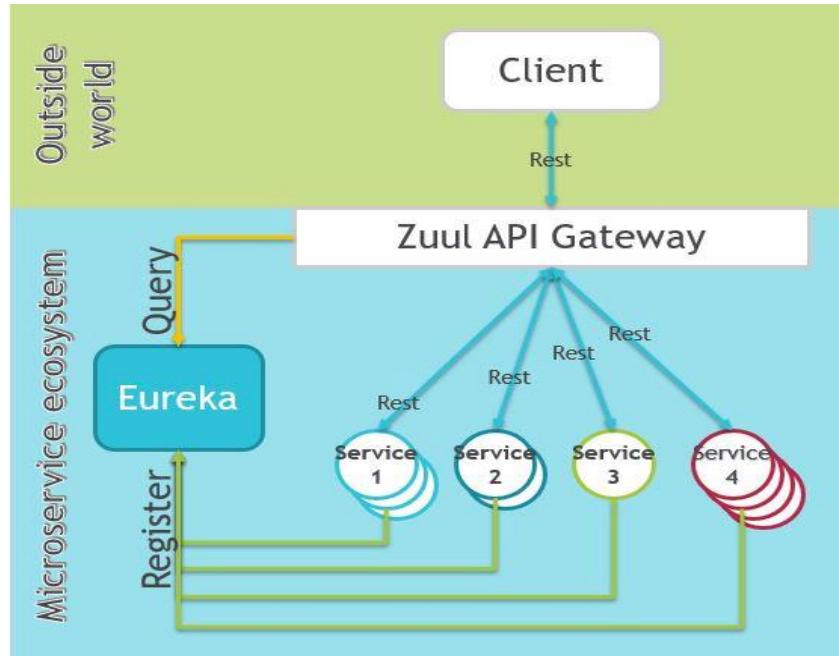
1. API 를 어떻게 통합할 것인가?

- API Gateway
 - 진입점의 통일
 - URI Path-based Routing (기존에 REST 로 된경우 가능)
- Service Registry
 - API Gateway 가 클러스터 내의 인스턴스를 찾아가는 맵

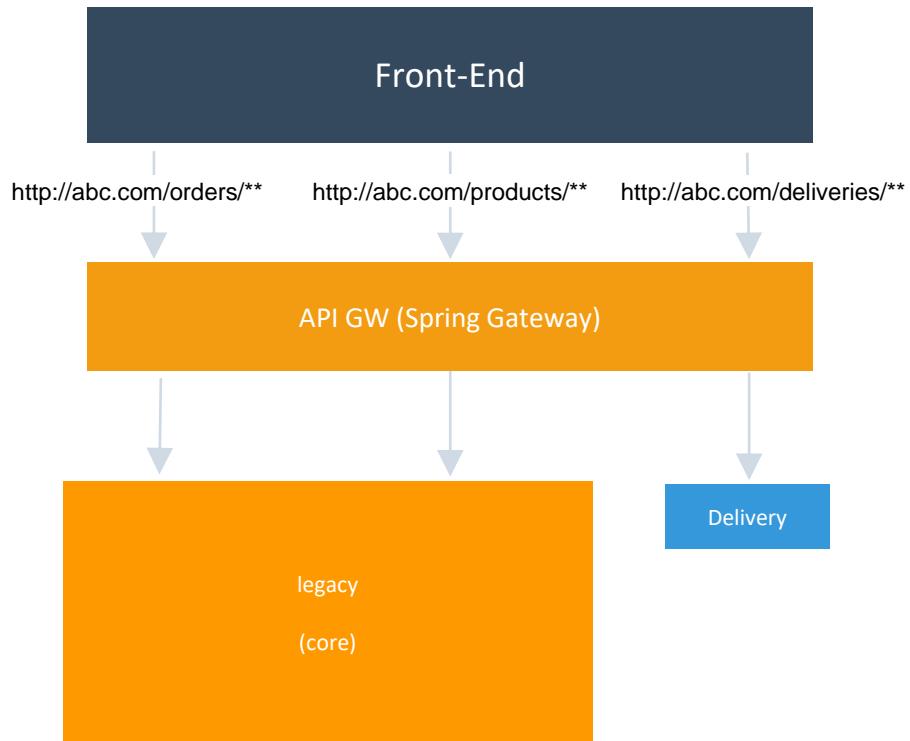
API Gateway : Edge Service

Acting like “Skin” to access our services :

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



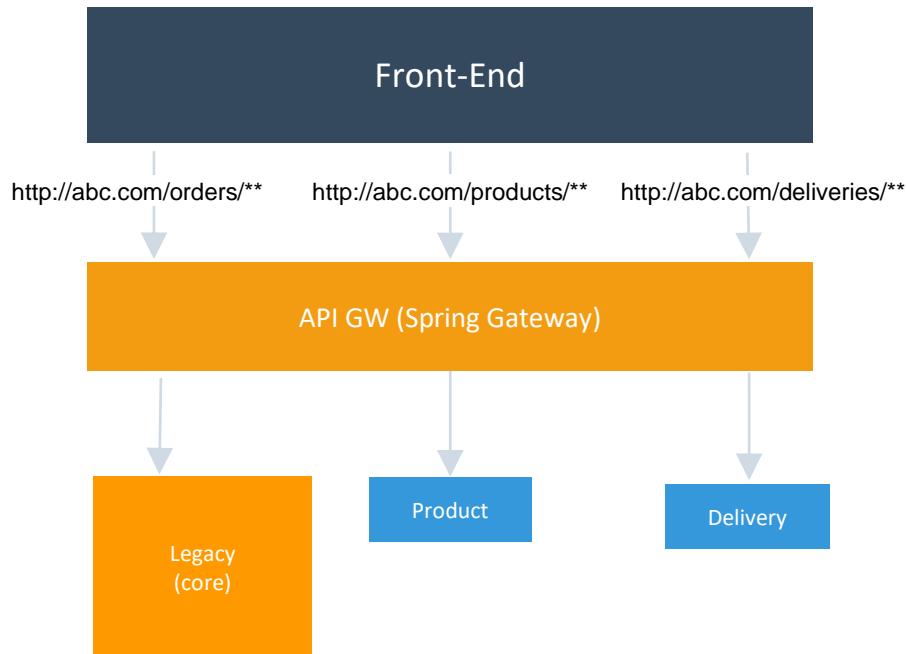
API Gateway : Strangler



#application.yml

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

API Gateway : Strangler



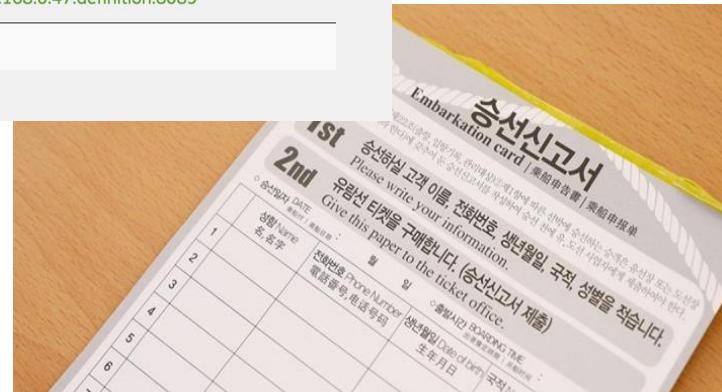
#application.yml

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

Service Registry: EUREKA or Kube-DNS

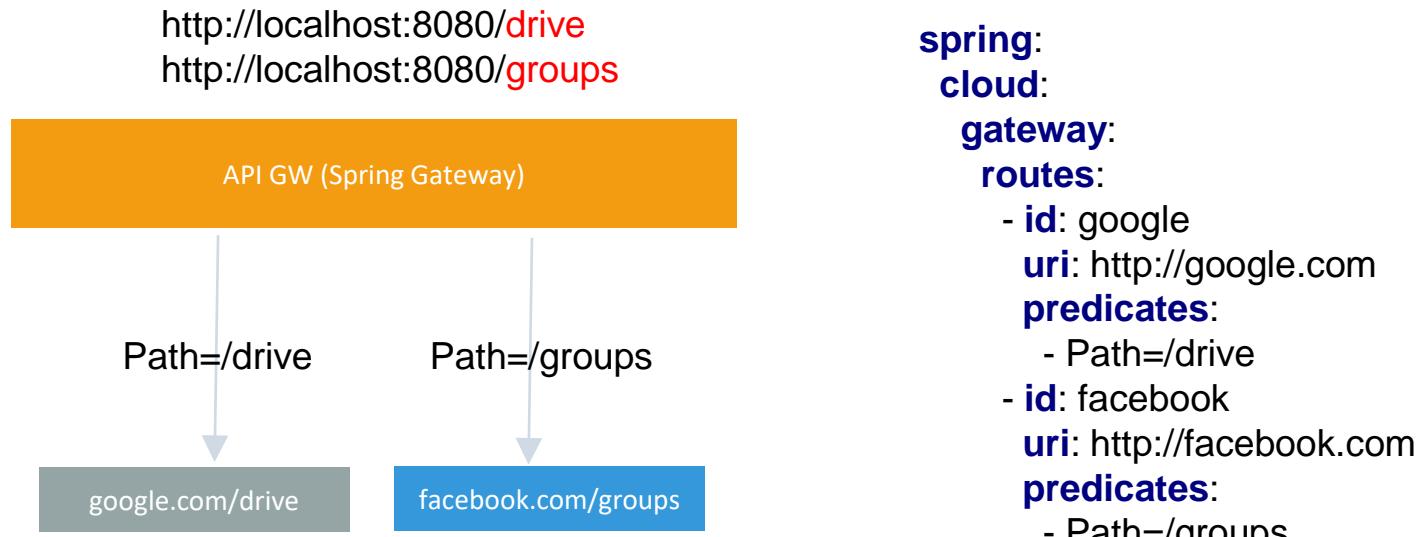
The screenshot shows the Spring Eureka service registry interface. At the top, there's a dark header with the "spring Eureka" logo on the left and "HOME" and "LAST 1000 SINCE STARTUP" links on the right. Below the header, the main content area has a title "Instances currently registered with Eureka". A table lists three registered instances:

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



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

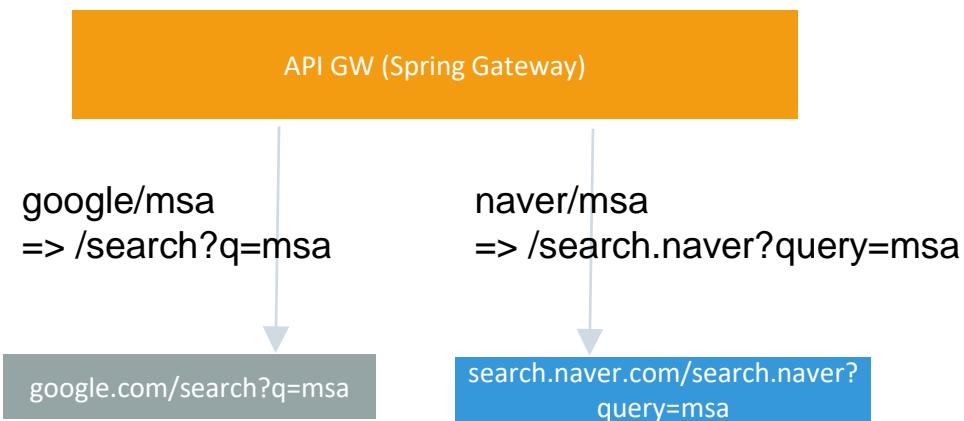
- Route : 게이트웨이의 기본 블록 구성이다. ID, Url , Predicate, Collection of Fillers 로 이루어져 있다. Predicate 가 일치할 때 Route 가 true 로 인식을 한다
- Predicate : 조건부 – 헤더, 파라미터등의 http 요청을 매치



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

http://localhost:8080/google/msa
http://localhost:8080/naver/msa

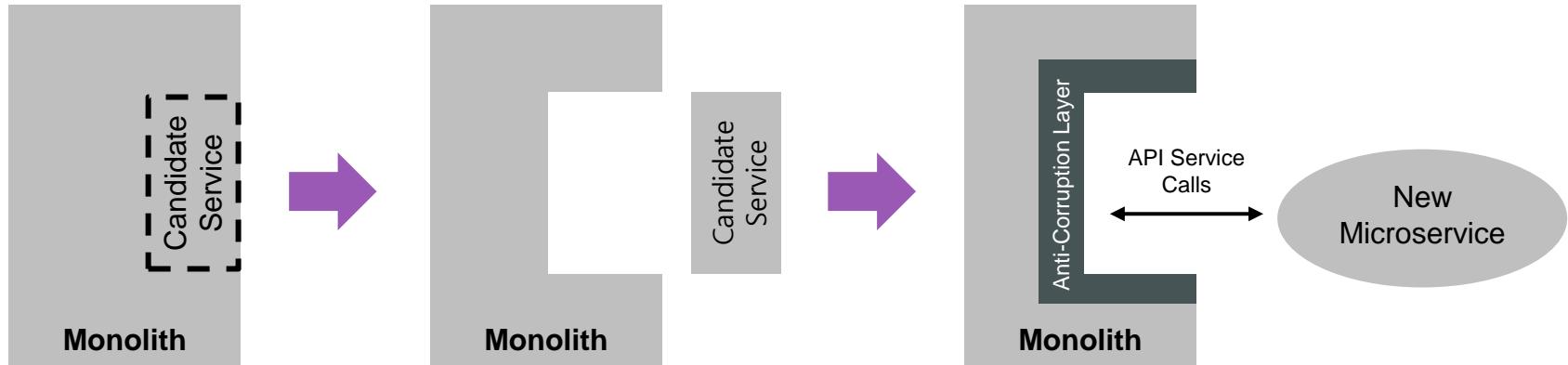


```
spring:  
cloud:  
gateway:  
routes:  
- id: google  
uri: http://google.com  
predicates:  
- Path=/google/{param}  
filters:  
- RewritePath=/google(?:<segment>/?.*)/,/search  
- AddRequestParameter=q,{param}
```

2. 어떻게 (다시) 상호 연동시킬 것인가?

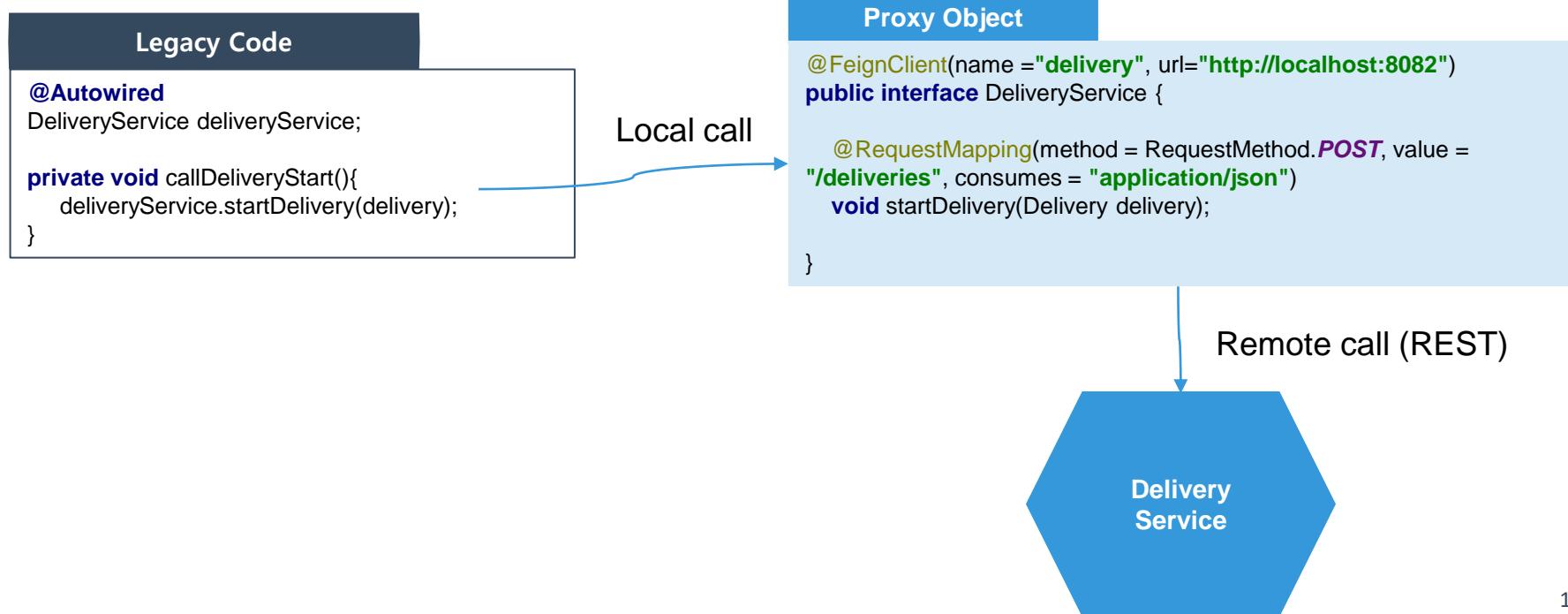
- Find the seams and replace with proxy
→ 기존 연계되던 이음매 객체 (interface) 를 찾아, 그에 상응하는 Façade, Proxy 로 대체
- Event Shunting
→ 레가시에서 이벤트를 퍼블리시하도록 전환, 신규 서비스가 주요 비즈니스 이벤트에 대하여 반응하도록 처리

Find the seams and replace with proxy

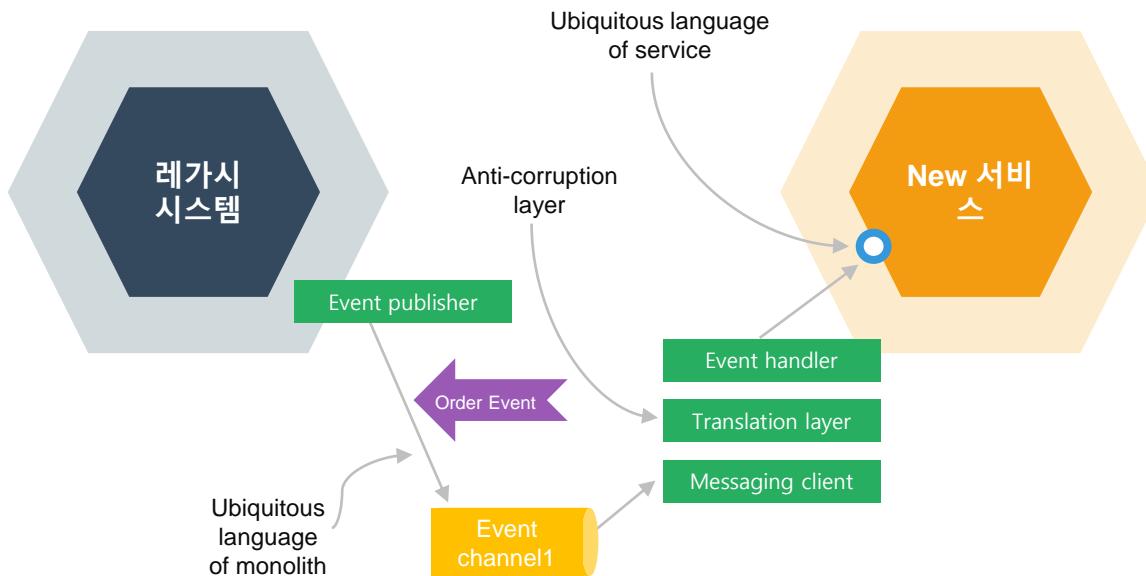


- Determine candidate service from Monolith and strangle it out
- Convert candidate service to Microservice and Add Anti-Corruption Layer to enable communication with Monolith

Proxy using FeignClient



Event Shunting



- **Aggregate 내 코드 주입**

- 호출 코드 직접 주입, Hexagonal 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

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?



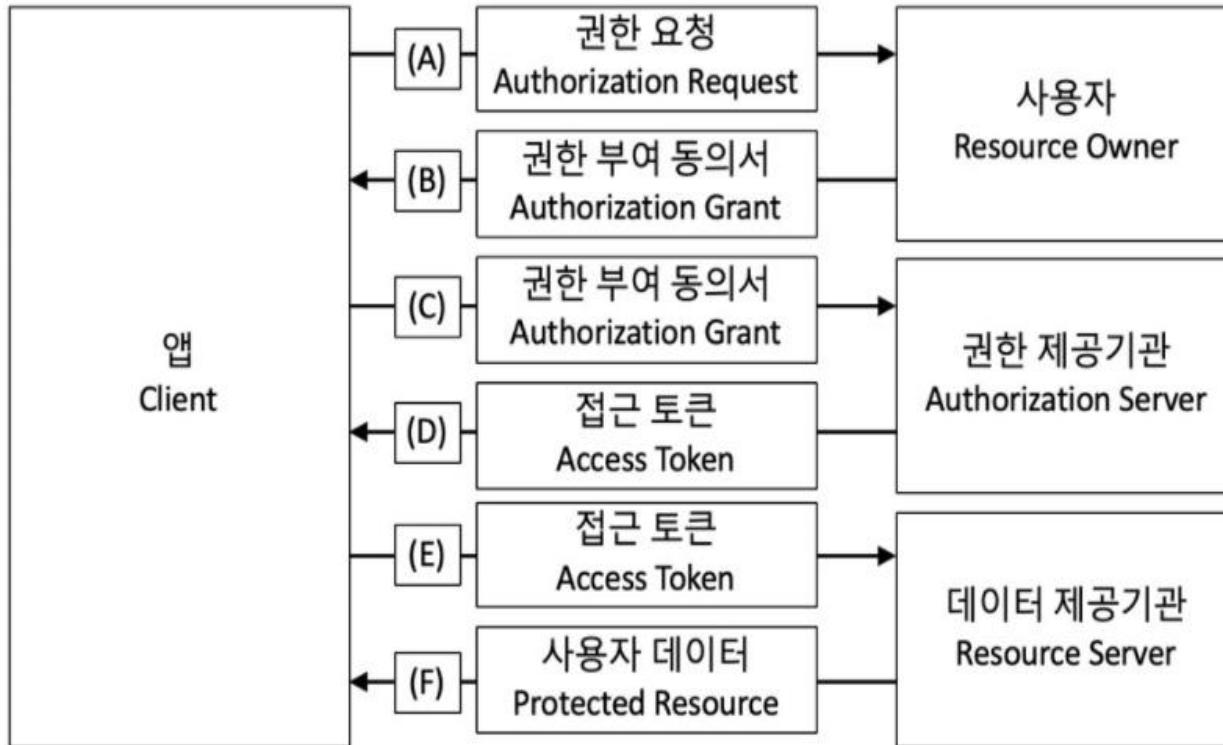
- OAuth2.0

- 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

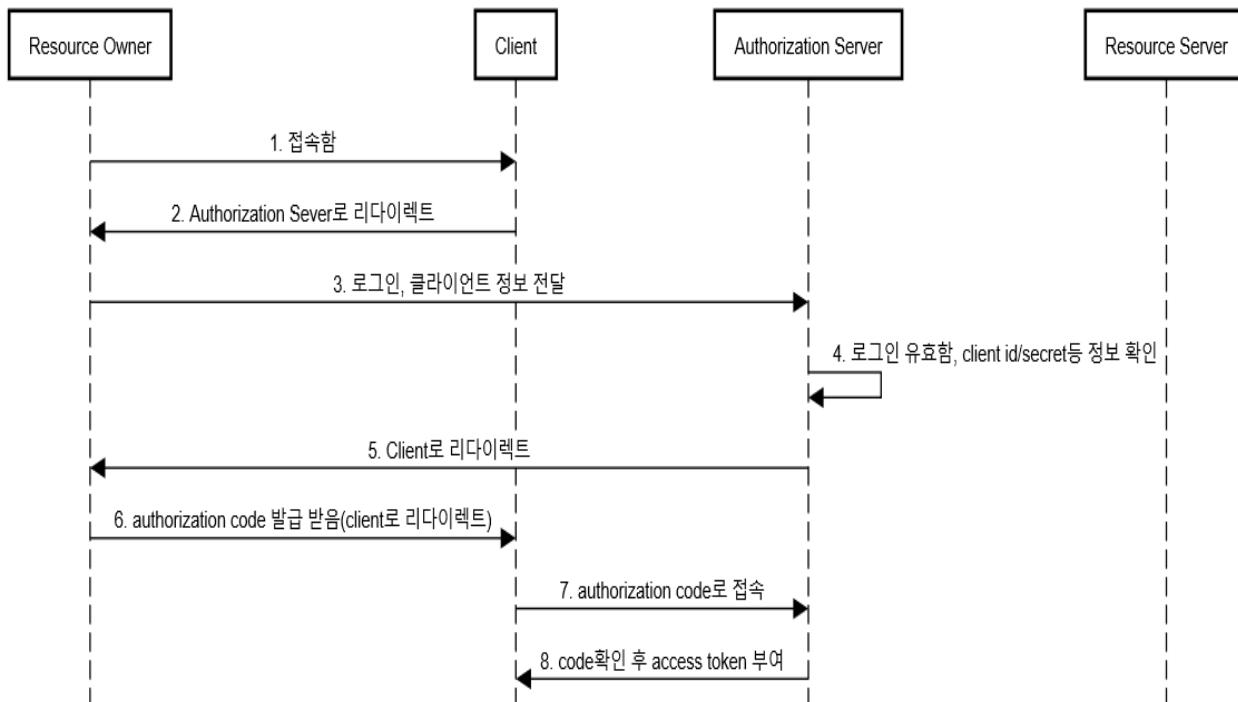
OAuth 2.0: Authorization Flow



출처: <http://www.2e.co.kr/>

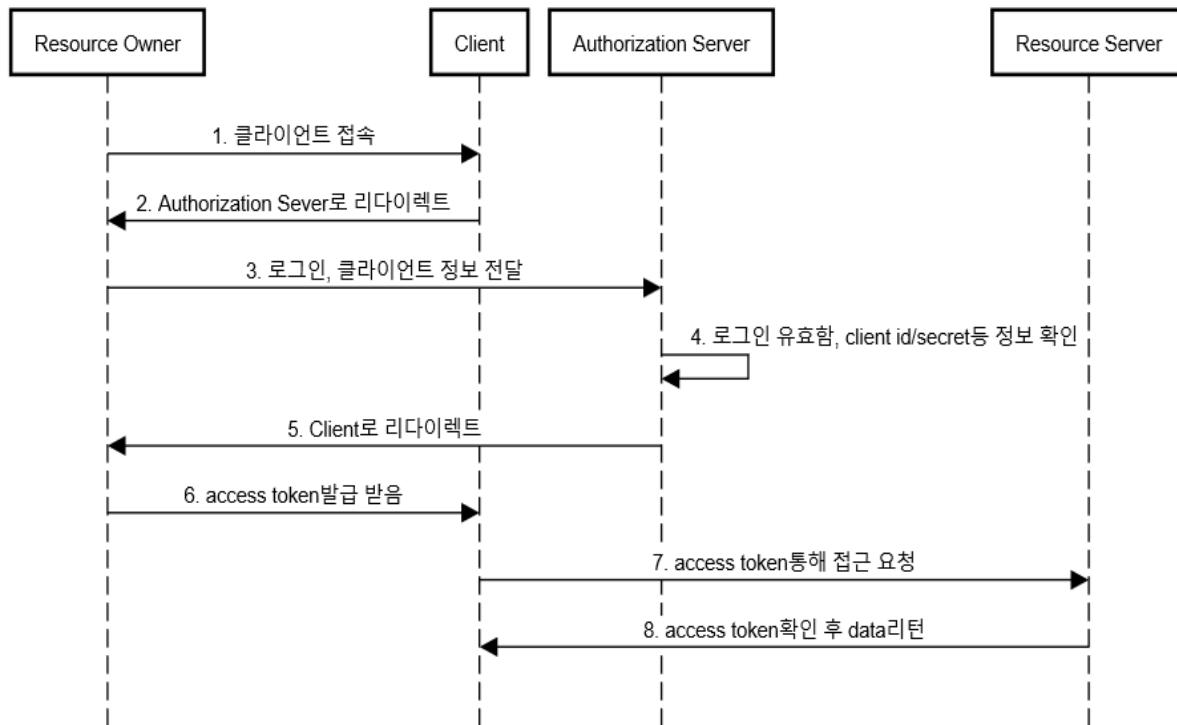
- A. 클라이언트가 자원 소유자에게 권한 요청
- B. 자원 소유자가 권한을 허가시, 클라이언트는 권한 증서를 발급받음
- C. 클라이언트는 권한증서를 가지고 토큰을 권한 서버에 요청
- D. 권한증서의 유효성을 체크하고 토큰을 발급해줌
- E. 클라이언트는 토큰을 사용하여 자원 요청
- F. 토큰 유효성 확인후 요청 처리

Grant Type #1 - Authorization Code



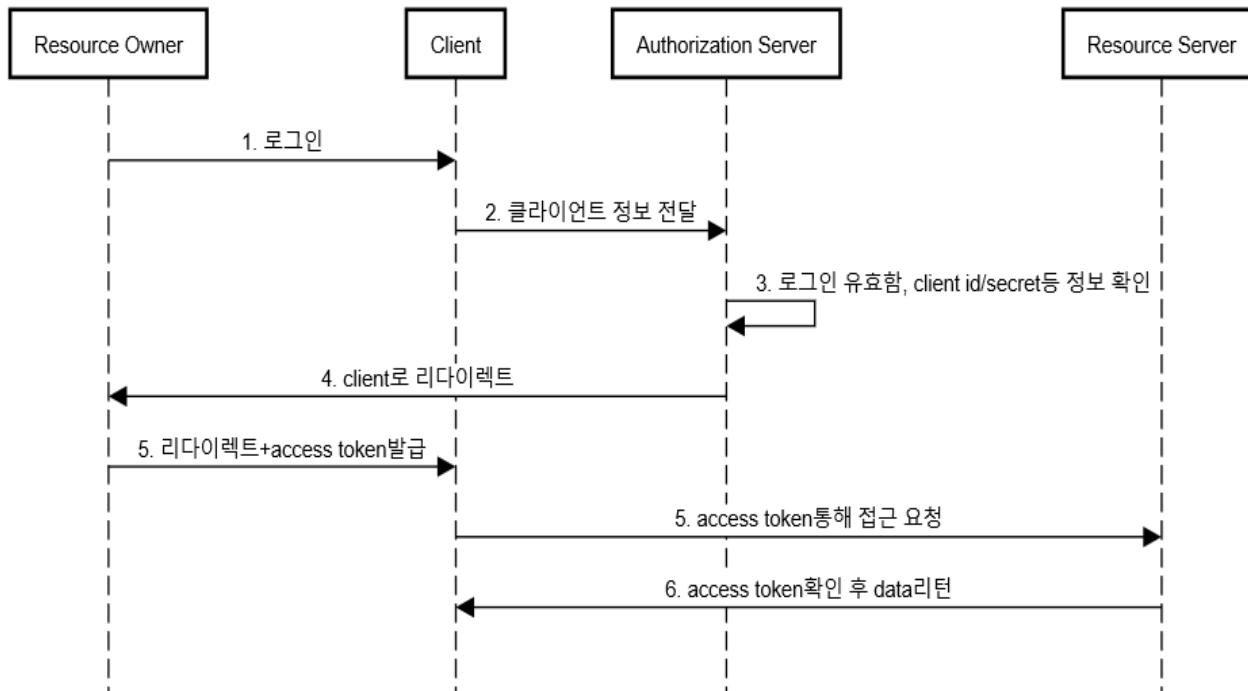
- ✓ 구글, 페이스북, 카카오 등 유저 정보가 다른 시스템에 있을 때 사용하는 방식 (Server-Side 어플리케이션)
- ✓ 어플리케이션이 인증서버에 요청해 브라우저를 열어서 사용자가 인증을 진행하게 하는 방식으로 사용
- ✓ 토큰요청시 코드를 요청하는 단계가 있어서 보안에 효과적
- ✓ 가장 복잡하지만, 가장 많이 쓰임 > 개발자만 고생하면됨

Grant Type #2 - Implicit (암묵적 유형)



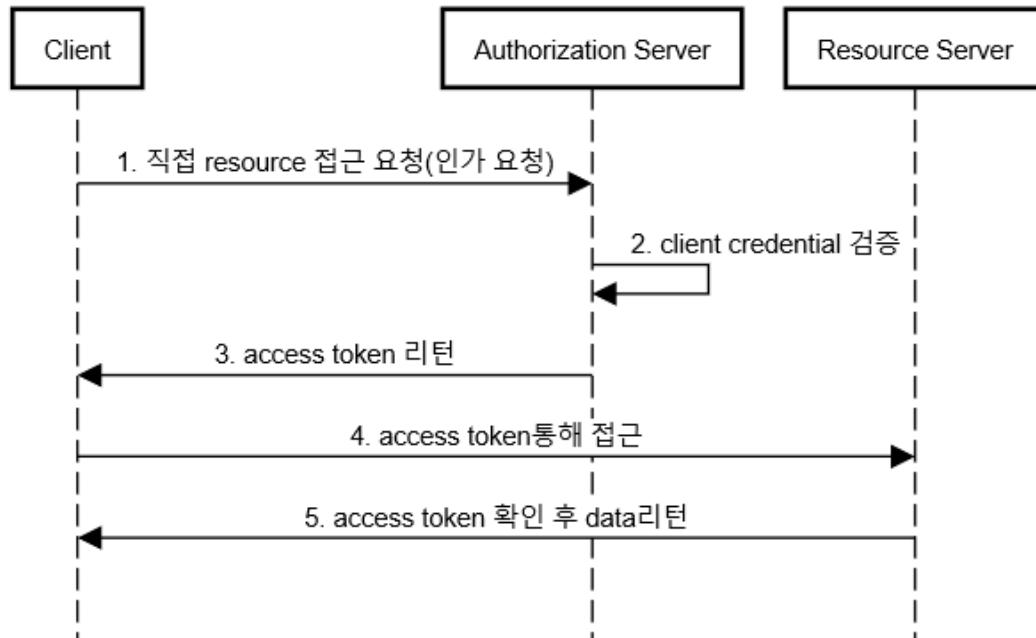
- ✓ Authorization_code 방식에서 코드를 요청하는 프로세스를 제거하고, 바로 토큰을 리턴 (Mobile Apps)
- ✓ Javascript로 동작하는 SPA(Single Page Application)에서 사용하기 위해 만들어 졌으나 권장하지 않음
- ✓ 신뢰성 있는 앱 또는 단말기에서 사용
- ✓ 외부에 있는 Oauth 서버가 CORS를 지원하지 않을 때 사용

Grant Type #3 - Resource Owner Password Credentials



- ✓ 타사의 인증 서버를 사용하지 않고, 자신의 서비스에 인증시 사용
(우리가 유저정보를 가지고 있고, 우리 서비스를 인증할 때)
- ✓ OAuth 2.0 의 가장 간단한 인증중 하나
- ✓ 전통적으로 이름과 비밀번호로 인증

Grant Type #4 - Client Credentials



- ✓ 사용자가 아닌 응용프로그램 (client) 이 인증을 요청할때 사용
- ✓ Resource Owner = Client
- ✓ 접근 권한이 한정되어있는 프로그램 사용시 활용
- ✓ 신뢰성이 높은 관리자용 Desktop App이나 Mobile App에서 사용

JWT(Json Web Token)



- JWT는 세 파트로 나누어지며, 각 파트는 점로 구분하여 xxxx.yyyyy.zzzzz 식으로 표현되며, 순서대로 헤더 (Header), 페이로드 (Payload), 서명 (Signature)으로 URL-Safe 한 Base64url 인코딩 사용
- Header는 토큰의 타입과 해시 암호화 알고리즘으로 구성
- 첫째는 토큰의 유형 (JWT)을, 두 번째는 HMAC, SHA256 또는 RSA와 같은 해시 알고리즘 표시
- Payload는 토큰에 담을 클레임(claim) 정보를 포함
- Payload 에 담는 정보의 한 '조각' 을 클레임이라고 부르고, 이는 name / value 의 한 쌍으로 토큰에는 여러개의 클레임 저장
- 마지막으로 Signature는 secret key를 포함하여 암호화

JWT(Json Web Token)

Strength

- URL 파라미터와 헤더로 사용
- 수평 스케일이 용이
- 디버깅 및 관리가 용이
- 트래픽 대한 부담이 낮음
- REST 서비스로 제공 가능
- 내장된 만료
- 독립적인 JWT

Weakness

- 클라이언트에 저장되어 위/변조의 위험성 노출
- 클레임 정보가 늘어나면 토큰이 비대해질 가능성
- Stateless 환경에서 모든 요청에 대해 전송되므로 트래픽 영향



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)

- UI에서 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 body에 저장하여 사용자가 가지고 있고, 증명처럼 사용, header에 실어 서버에 요청
 - 토큰을 변조 하더라도, SECRET_KEY가 없으면 복호화를 못함
 - 참고 : <https://brownbears.tistory.com/440>

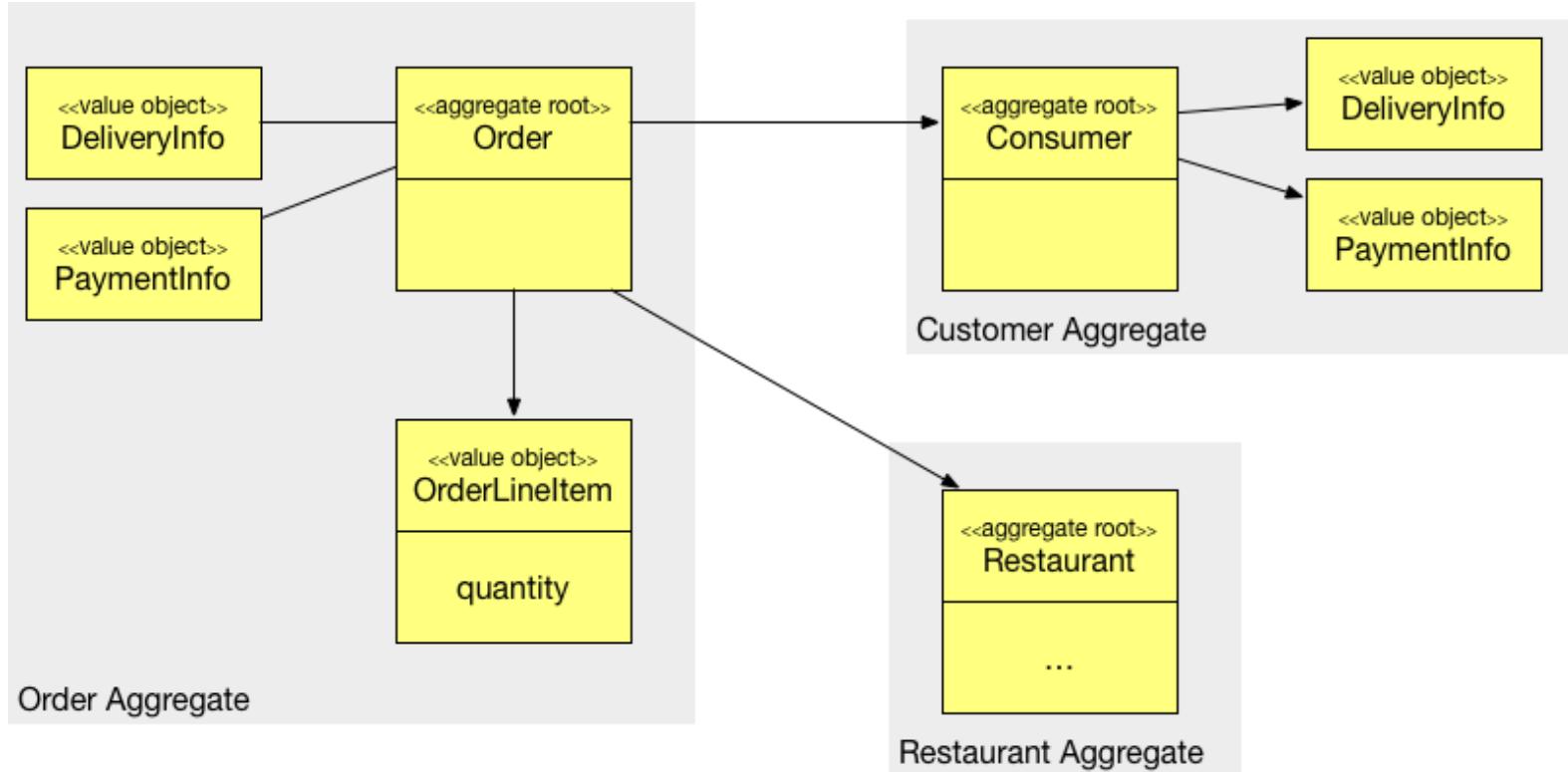
Lab: OAuth Based Authorization (3/3)

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

Aggregate Root를 통한 객체 참조



URI 를 통한 객체 참조

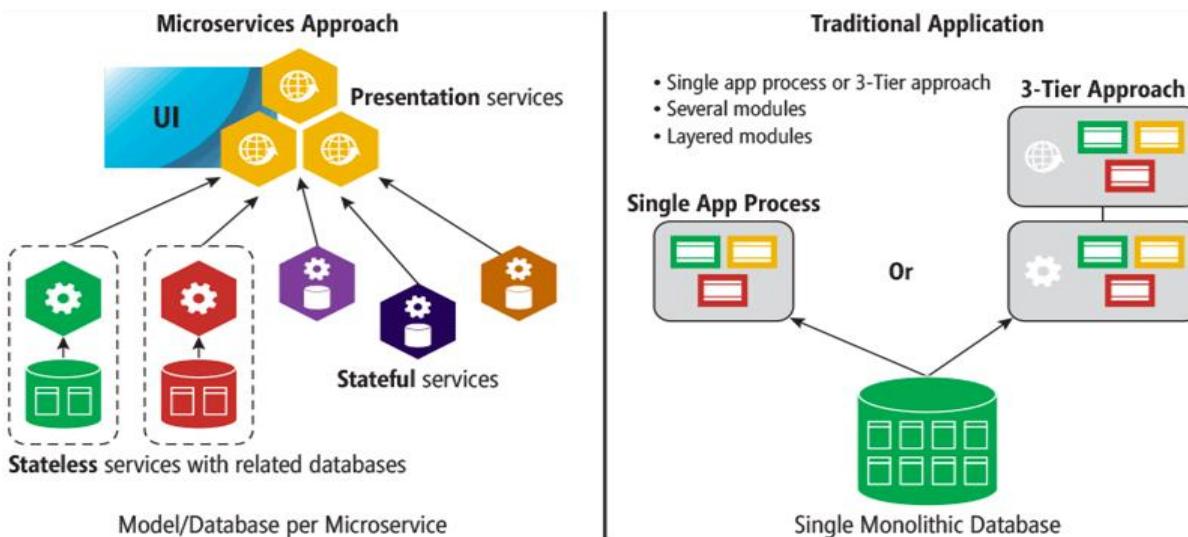
- **HATEOAS** (Hypertext As The Engine Of Application State)

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

```
GET .../followers/ids.json?cursor=-1&screen_name=josdirksen
{
  "previous_cursor": 0,
  "id": {
    "name": "John Smit",
    "id": "12345678"
    "links" : [
      { "type": "application/vnd.twitter.com.user",
        "rel": "User info",
        "href": "https://.../user/12345678" },
      { "type": "application/vnd.twitter.com.user.follow",
        "rel": "Follow user",
        "href": "https://.../friendship/12345678" }
    ] // and add other options: tweet to, send direct message,
    // block, report for spam, add or remove from list
  } // This is how you create a self-describing API.
}
```

Microservice Integration with UI

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



5. 중복된 기능과 데이터를 어떻게 할 것인가?

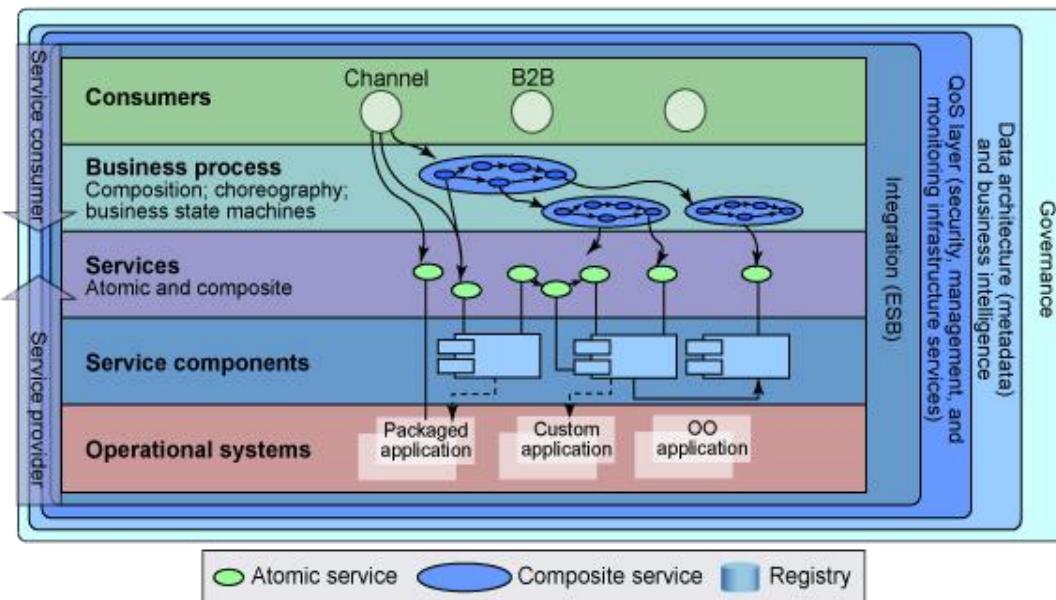
- 재사용하지 않고 중복 구현하는 것이 MSA 스러운 것
 - 재사용 통한 경제성(SOA사상, 디펜던시발생)과 자율적 창발 (낮은 간섭과 빠른 출시)의 트레이드 오프에서 후자의 전략을 선택
 - Utility service X, DRY(Don't Repeat Yourself) 를 X
 - Polyglot Persistence
- 예외상황 : 데이터 참조에 intensive 한 서비스 (인증정보 등)는 Utility 서비스 성격으로 구현 가능함

Table of content

Microservice and
Event-storming-Based
DevOps Project

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

Integration Patterns



By UI

By Request-Response

By Event-driven Architecture

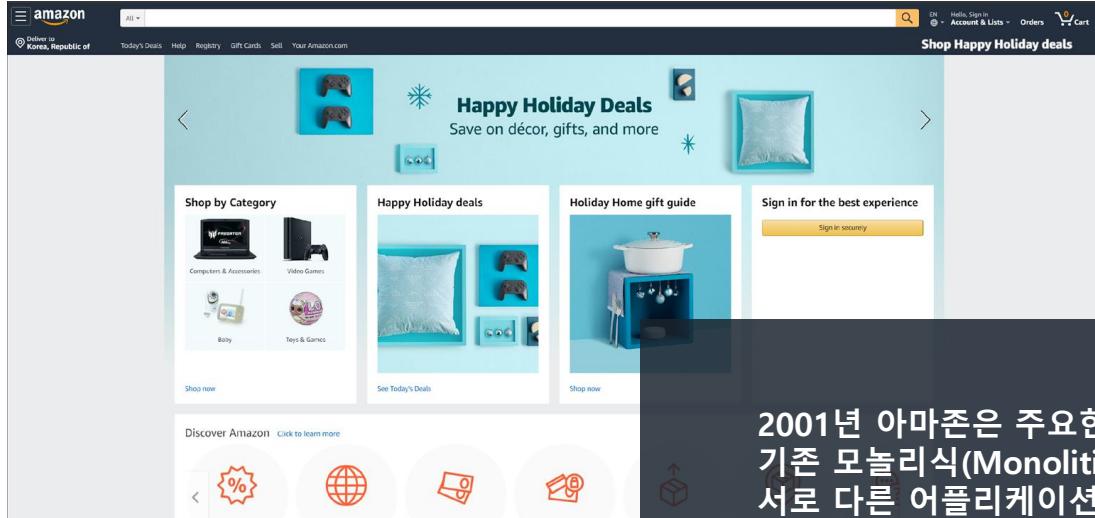
“

Service Composition with User Interface

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



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

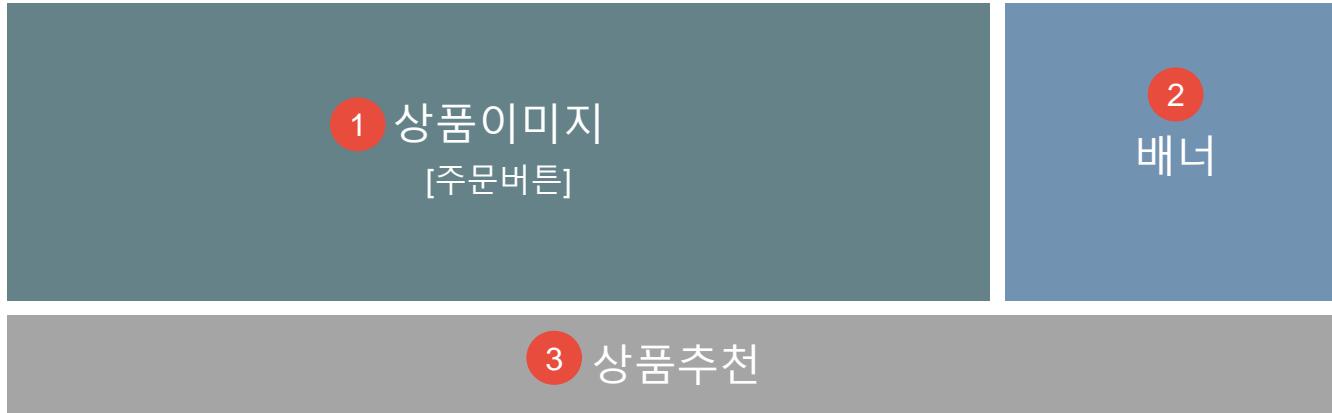


2001년 아마존은 주요한 아키텍쳐 변화가 있었는데
기존 모놀리식(Monolithic) 기반에서
서로 다른 어플리케이션 기능을 제공하는
“분산 서비스 플랫폼”으로 변화 하였습니다.

현재의 Amazon.com의 첫 화면에 들어 온다면,
그 페이지를 생성하기 위해
100여개가 넘는 서비스를 호출하여 만들고 있습니다.

Client-side Rendering : 장애 전파 회피

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



Server-side Rendering 은 모든 화면의 콘텐츠가
도달해야만 화면을 보여줄 수 있지만,
Client-side Rendering 은 먼저 데이터가
도달한 화면부터 우선적으로 표출할 수 있다.
광고배너가 나가지 못한다고 주문을 안받을 것인가?

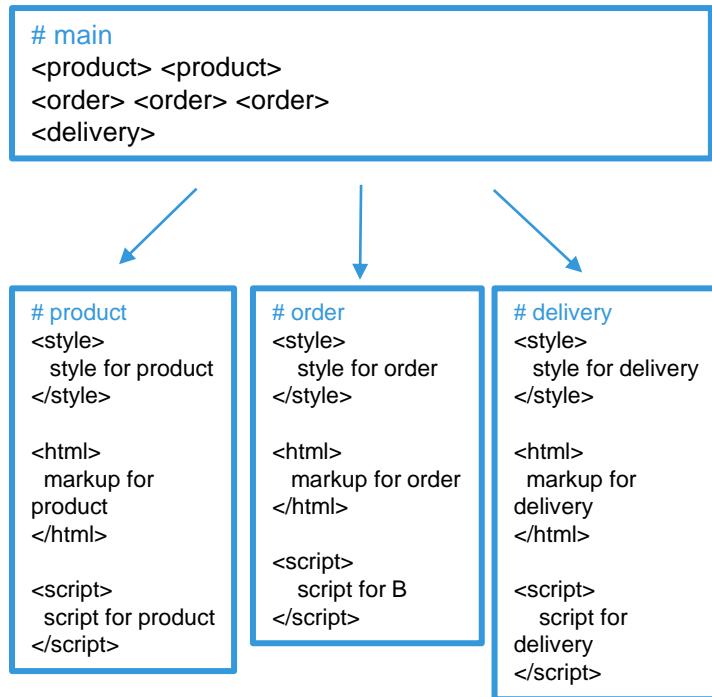
W3C Web Components

- 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



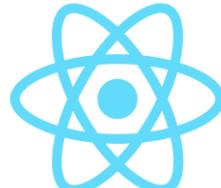
VueJS



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>

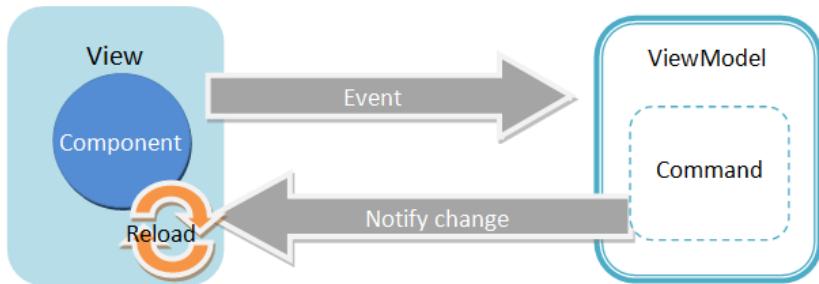
  </tr>
</table>
```

Data binding to Domain Tags

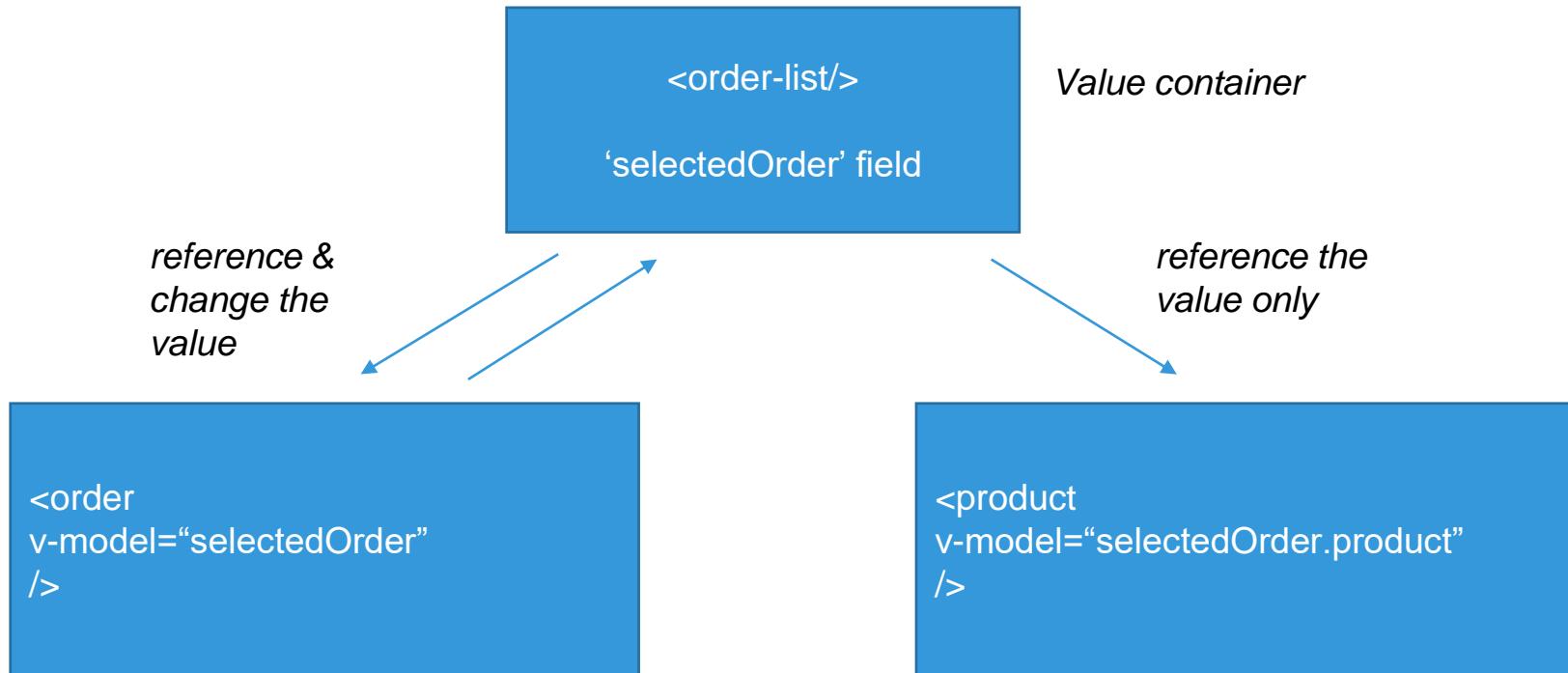
```
<product
    v-model="props.items[index]"
    @inputBuy="showBuy"
    @inputEdit="showEdit"
></product>

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

MVVM



Interaction between Domain Tags

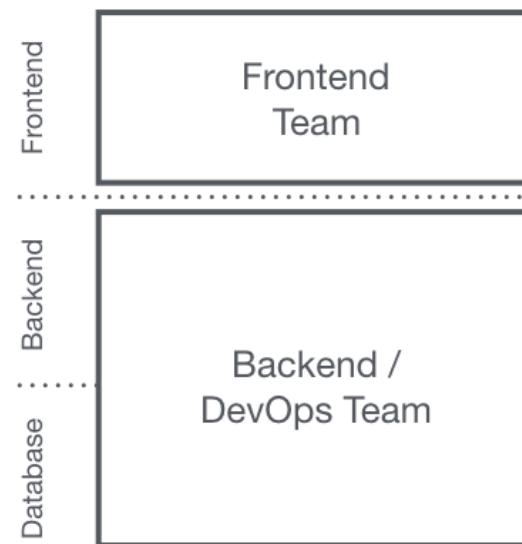


Monolith-Frontend

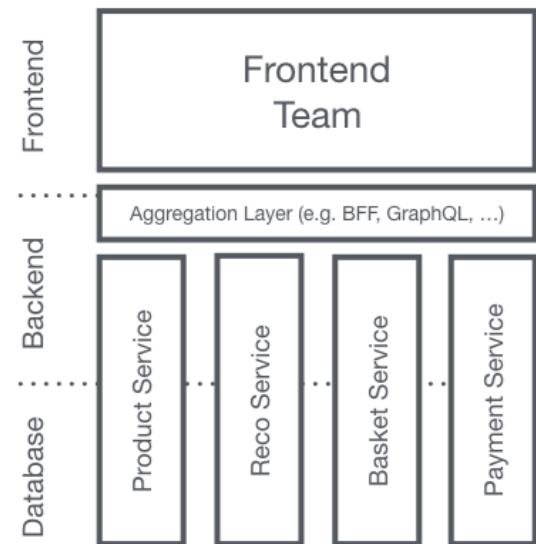
The Monolith



Front & Back

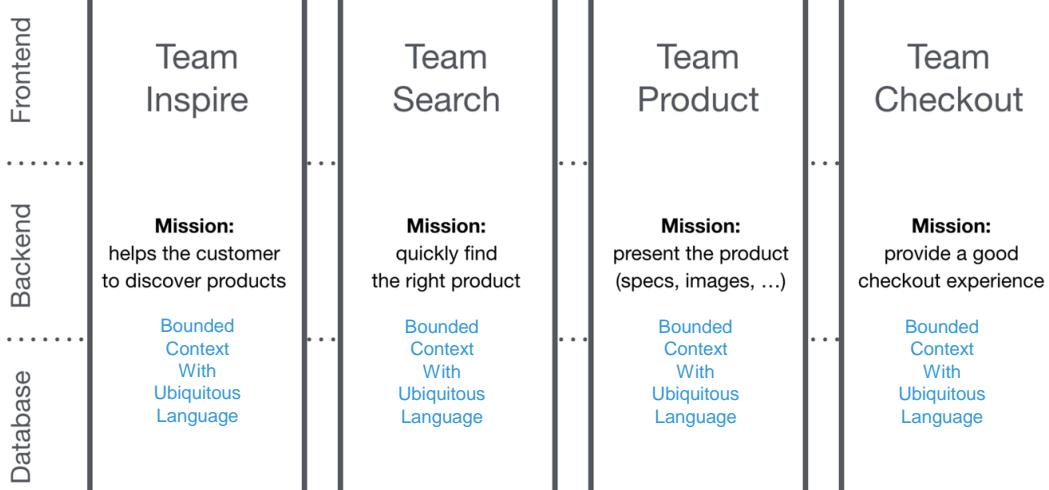


Microservices



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

Micro-Frontends



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

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

Lab: Multiple frontend frameworks on a single page

What is your name?

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

Tell the components

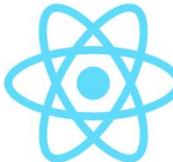
Angular



Hello **James** from your friendly Angular component.

Say hello

React



Hello **James** from your friendly React component.

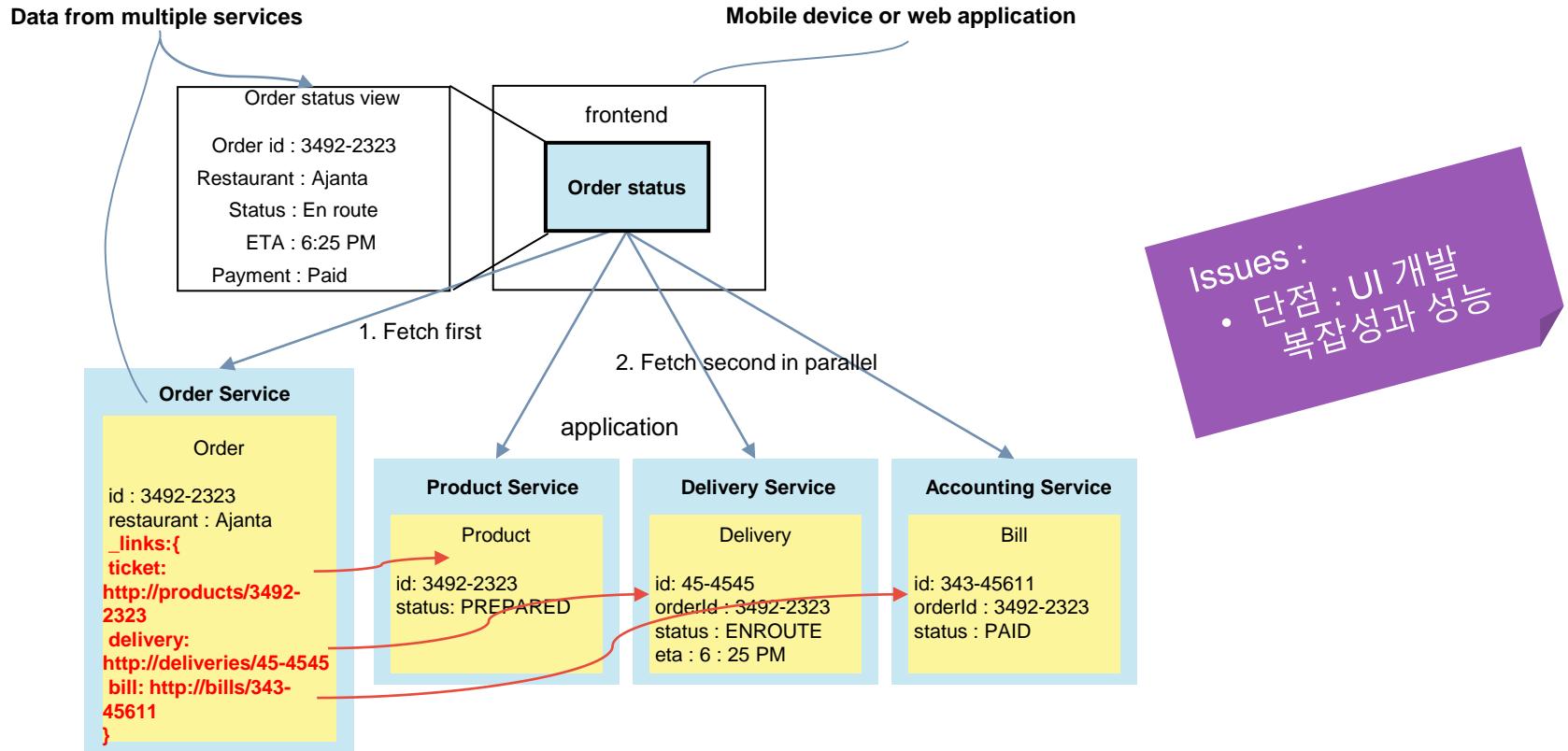
Say hello

While they provide:

- Autonomously deployable
- Communicate each other

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

Read-Side using UI: Data Projection by UI and HATEOAS



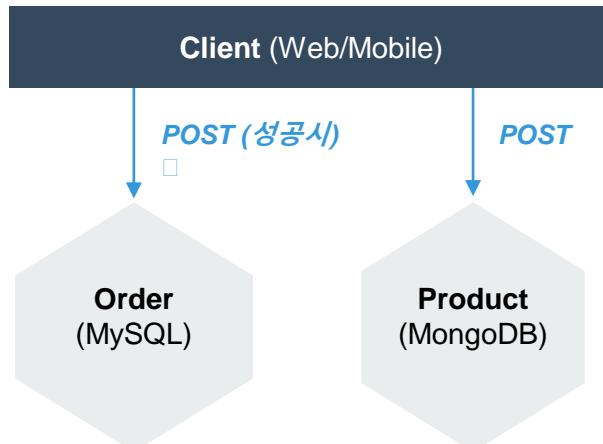
Write-Side using UI:

Distributed Transaction Problem in data mutation by UI composition

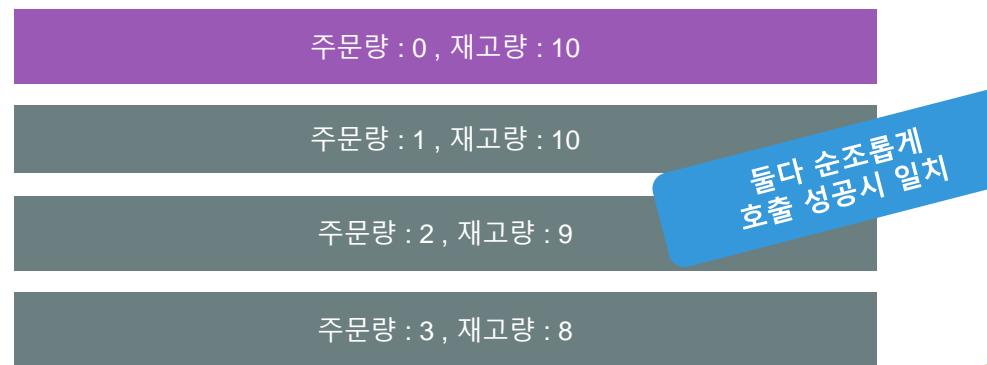
In-consistent

Consistent

- 서비스구성 (클라이언트가 순차 호출)



- 조회화면



불일치 영원히
생길 수 있음!!

Integration Strategies Compared

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

Table of content

Microservice and
Event-storming-Based
DevOps Project

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

“

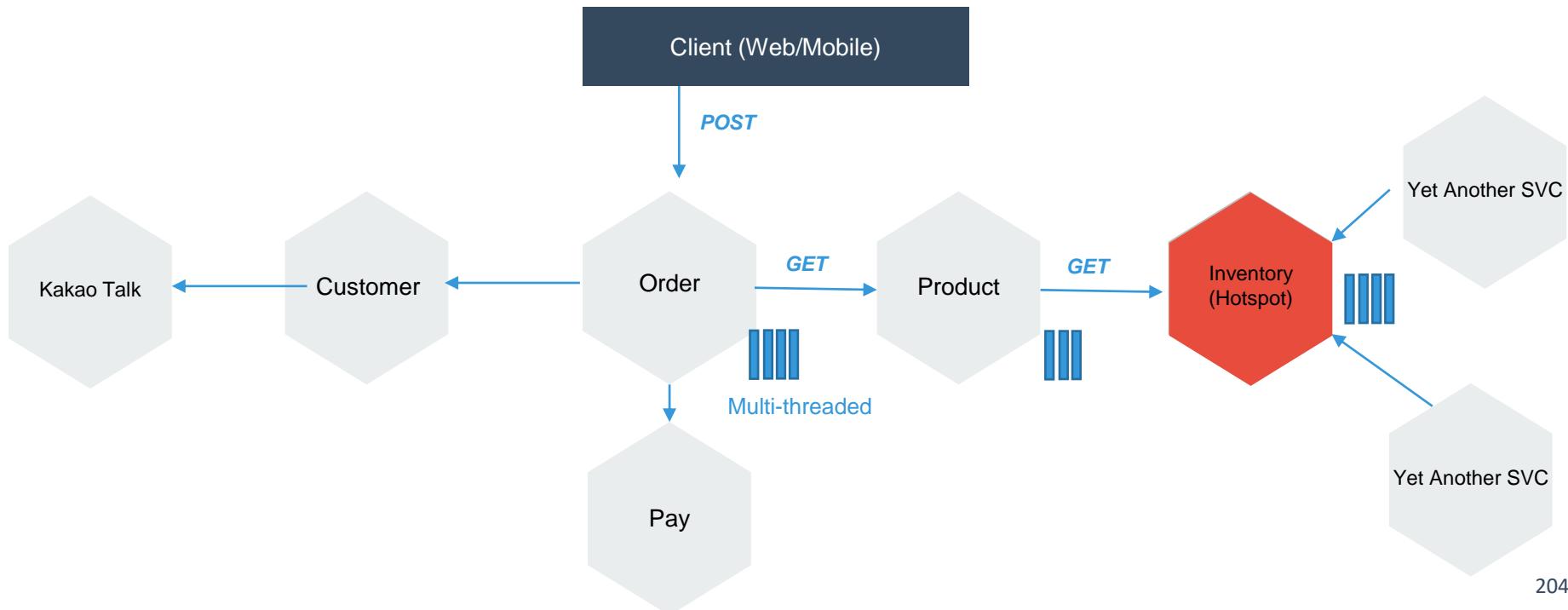
Service Integration by Request-Response

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

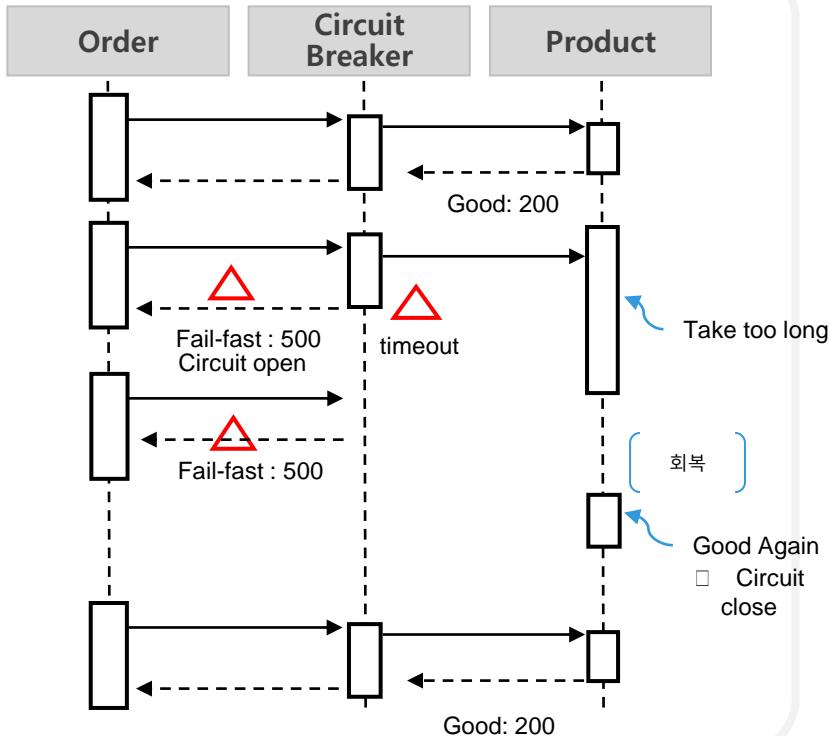
Data Projection in Request-Response model

서비스구성 (Request-Response, Sync 호출)

- 성능저하
- 장애전파



장애전파 차단: 서킷브레이커 패턴



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

- 메모리 사용 폭주 막음

장애 감지 시, 차단기 작동(Open)

일시 동안 Fallback으로 서비스 대체

일부 트래픽으로 서비스 정상여부 확인

정상 확인 시, 차단기 해제(Close)

Lab Time: Circuit breaker using istio

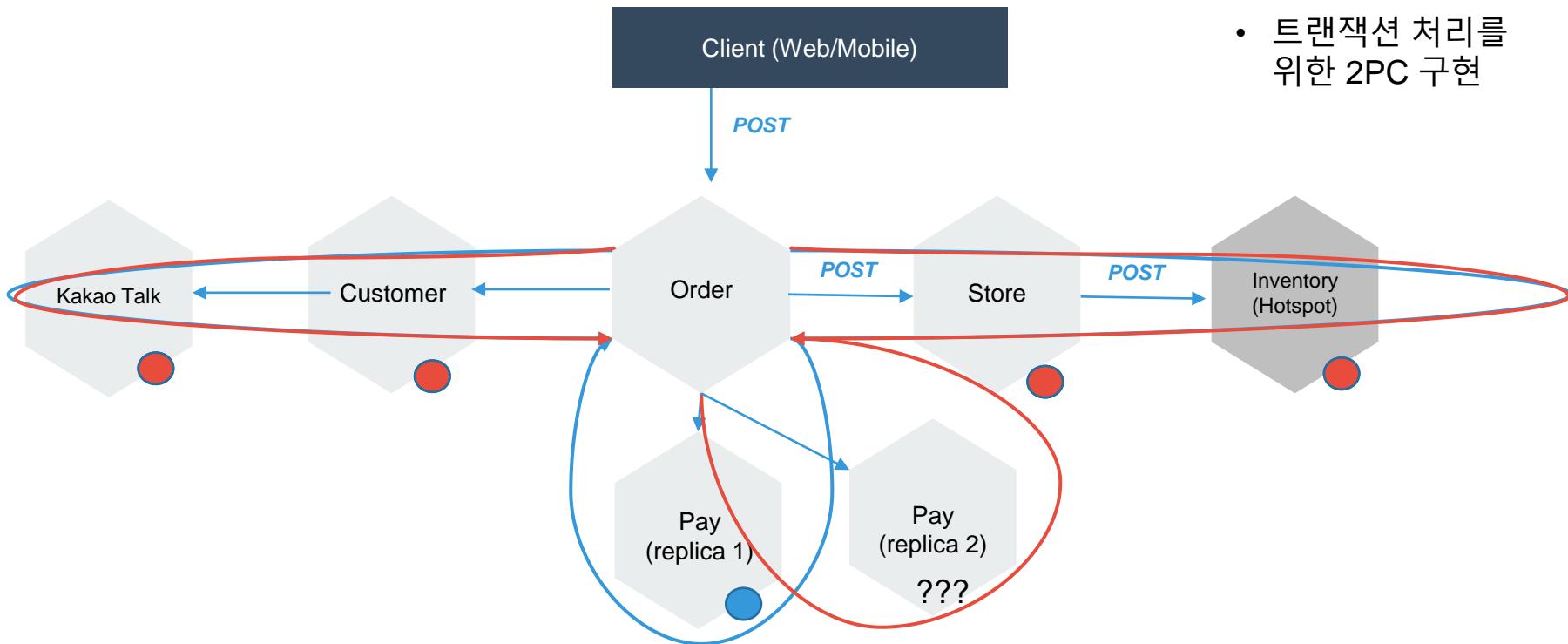
- product 서비스에 서킷 브레이커 적용
 - connectionPool :
지정된 서비스에 connections를 제한하여 가능 용량 이상의 트래픽 증가에 따른 서비스 Pending 상태를 막도록 *circuit breaker*를 작동시키는 방법
 - outlierDetection :
오류 발생하거나 응답이 없는 인스턴스를 탐지하여 *circuit breaker*를 작동시키는 방법
- 부하를 주어서 지정된 커넥션 만큼 차단 확인
 - siege -c2 -t10S -v --content-type "application/json"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

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

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

서비스구성 (Request-Response, Sync 호출)

- 성능저하
- 장애전파
- 트랜잭션 처리를 위한 2PC 구현



Integration Strategies Compared

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

“

Service Integration by Event-driven Model

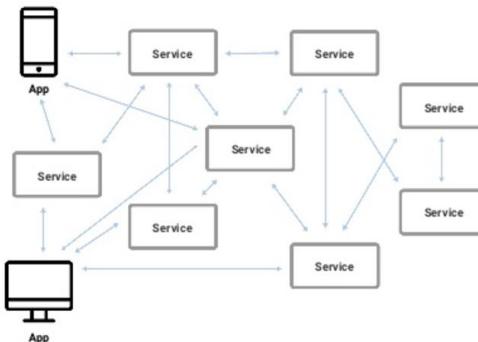
- Event-driven Approach
- ACID Tx. vs Eventual Tx.
- Business Transaction / Compensation (Saga) Pattern

Microservice Integration with Event-driven Architecture

Request-Response Applications

Deterministic
Rigid
Tight coupling

confluent

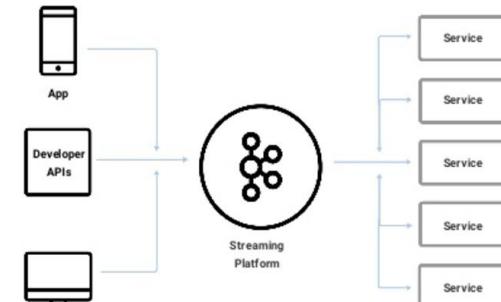


VS

Event-Driven Applications

Responsive
Flexible
Extensible

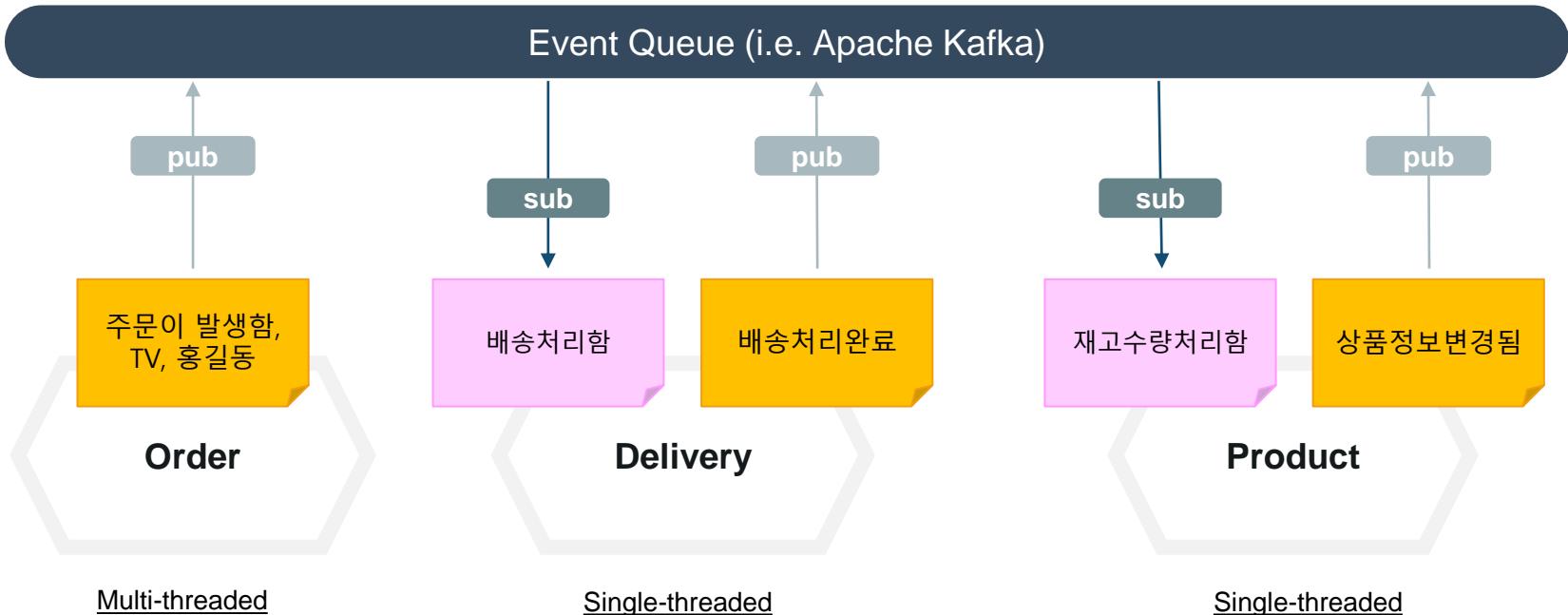
confluent



- Point to Point Spaghetti network
- Blocking Model
- System fault spread

- Broadcasting
- Non-Blocking Model
- System fault isolation

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



Event Publisher : Order

```
@Entity  
public class Order {  
    ...  
  
    @PostPersist // 주문이 저장된 후에  
    private void publishOrderPlaced() {  
        OrderPlaced orderPlaced = new OrderPlaced(); // 주문이 들어온 사실  
        을 이벤트로 작성  
  
        orderPlaced.setOrderId(id);  
        ...  
  
        orderPlaced.publish(); // 메시지 큐에 주문이 들어왔음을 신고  
    }  
}
```

Event Consumer : Delivery

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

```
    deliveryService.start(orderPlaced.getOrderId());
```

```
}
```

Event Consumer : Product

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

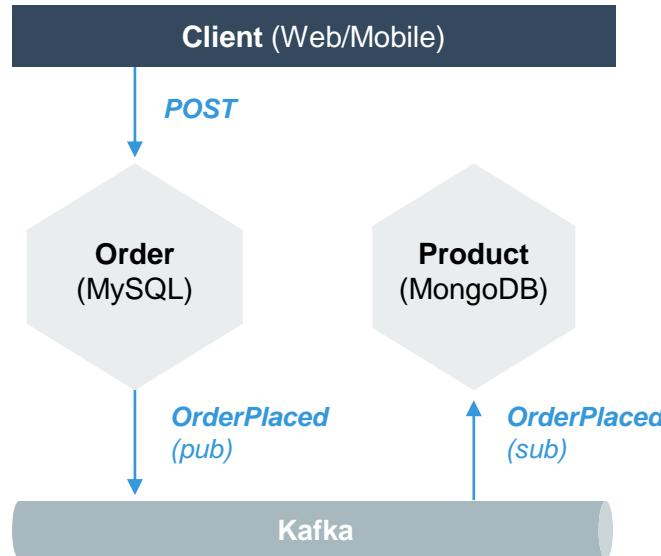
    // 해당 상품의 재고량을 주문량 만큼 줄여서 저장
    Product product =
        productRepository.findById(orderPlaced.getProductId()).get();
    product.setStock(product.getStock() - orderPlaced.getQuantity());

    productRepository.save(product);

}
```

Eventual Consistency를 통한 분산 트랜잭션 처리

- 서비스구성 (Eventual TX)



- 조회화면

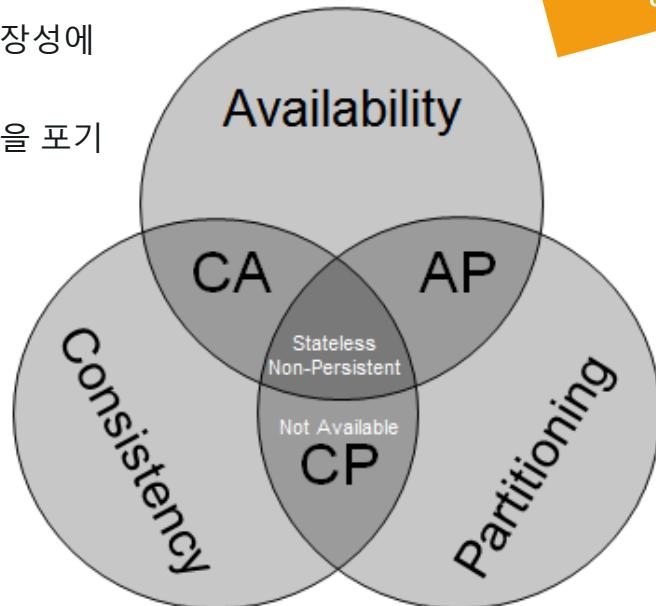


여기서 선택의 길을 만남...

- 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 포기 할 수 있는가?

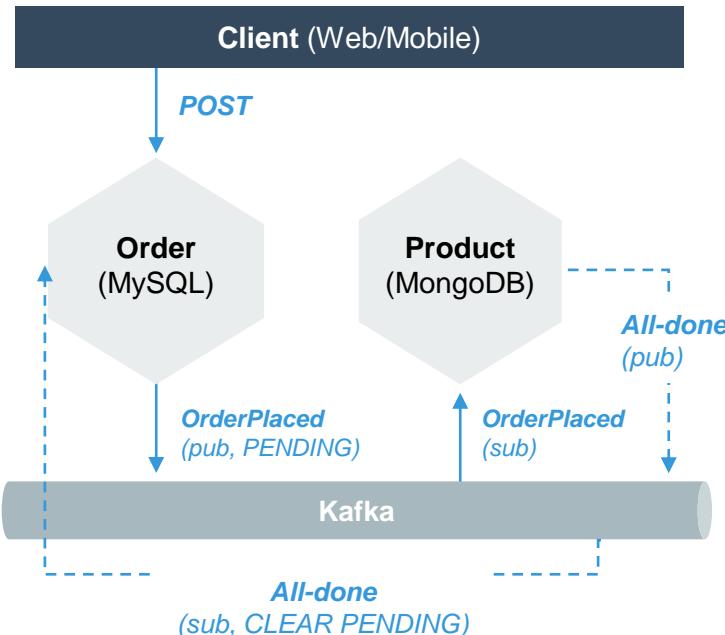
“성공한 내 서비스의 경쟁자들은 무엇을 포기했는가?”

- ▣ 강력한 의사결정 필요
(무엇으로 태어날 것인가...)



Eventual TX – 불일치가 Mission Critical 한 경우

- 서비스구성 (Eventual TX)



- 조회화면

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

불일치 할 수
있음을 표시

결국 일치

What is the Saga Pattern?

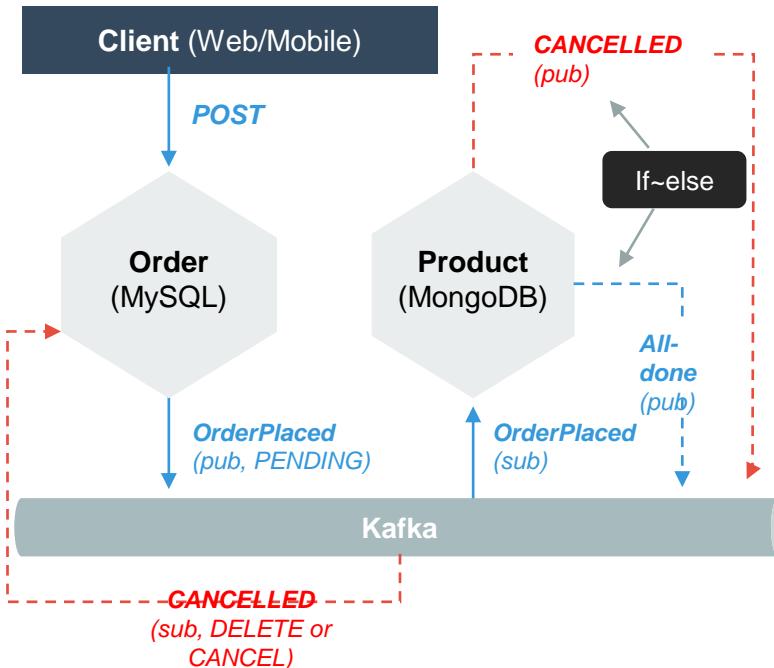
The **Saga Pattern** is a design pattern that provides a solution for implementing transactions in the form of **sagas** that span across two or more microservices.

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

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

Eventual TX – Rollback (Saga Pattern)

- 서비스구성 (Eventual TX)



- 조회화면



복구 &
결국 일치

Integration Strategies Compared

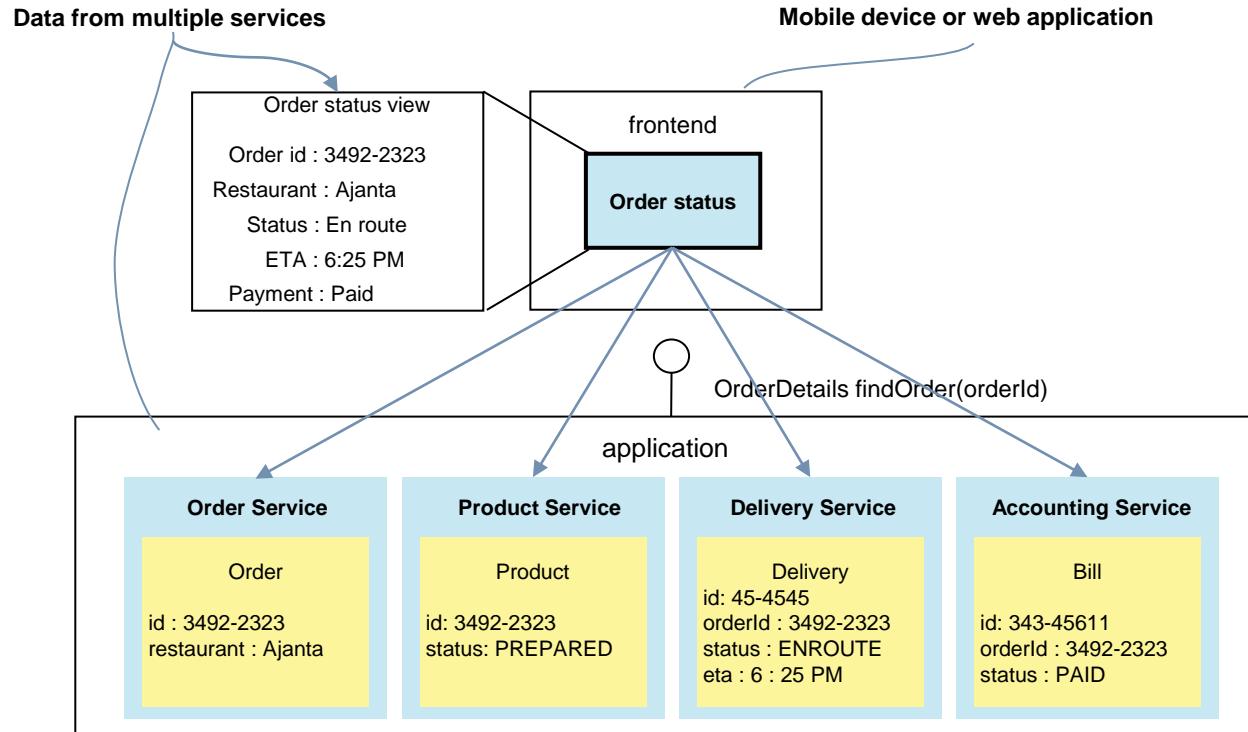
	UI	Req-Resp	Pub-Sub
읽기	UI Composition + HATEOAS	GET Request + Circuit Breaker	
쓰기	Not Recommended	2PC Not Recommended	Saga

“

Data Query (Projection) in MSA

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

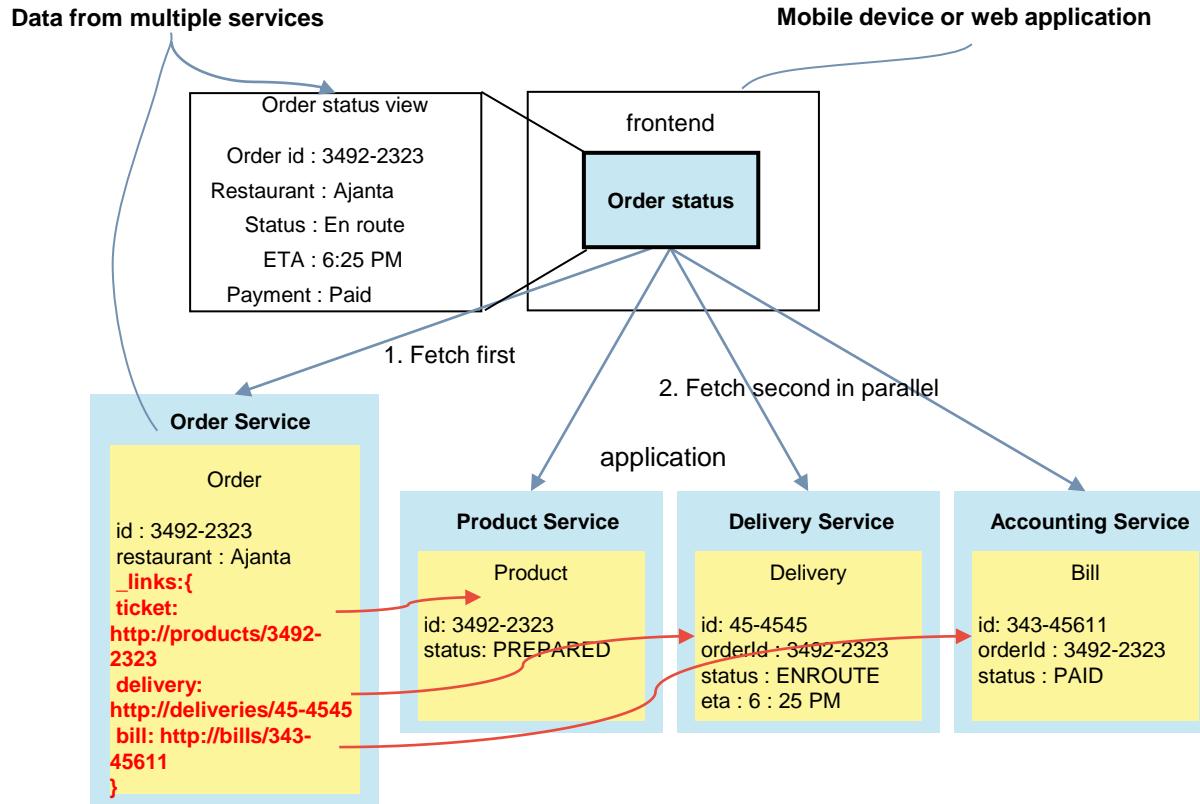
분산서비스에서의 Data Projection Issue



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)

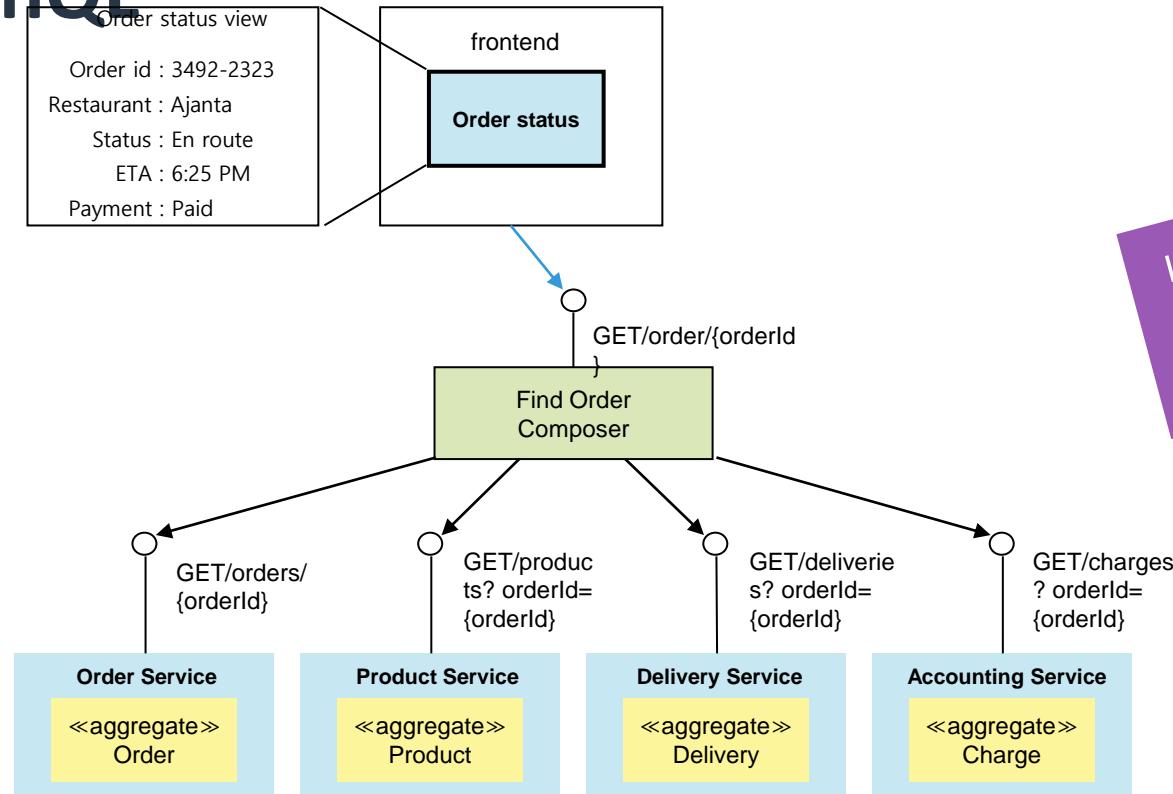
- 주문을 여러건 한다.
- Chrome의 개발자 모드 > Network 템을 연다.
- Menu의 My page (UI Mash Up)을 열어서 API 날라가는 모습을 확인한다.

주문 내역				
주문 번호	주문상품	구매수량	결제금액	배송상태
1	MASK	1	20000	DeliveryCompleted
2	NOTEBOOK	1	30000	DeliveryCompleted
3	TV	1	10000	DeliveryCompleted
4	TV	1	10000	DeliveryCompleted
5	TV	1	10000	DeliveryCompleted

```
{  
  "_embedded": {_,_,_}  
  "_embedded": {_,_}  
  "orders": [{  
    "productId": 2, "productName": "MASK",  
    "orderId": 1  
  }, {  
    "productId": 3, "productName": "NOTEBOOK",  
    "orderId": 2  
  }, {  
    "productId": 1, "productName": "TV", "orderId": 3  
  }, {  
    "productId": 1, "productName": "TV", "orderId": 4  
  }, {  
    "productId": 1, "productName": "TV", "orderId": 5  
  }]  
  "_links": {  
    "self": {  
      "href": "http://localhost:8081/  
    }  
  }  
}
```

하위 데이터를 HATEOAS로 유지시 UI 개발이 편리

Data Projection by Composite Service or GraphQL



Issues :

- 단점 : 성능(속도), 장애전파

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

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

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

```
CompletableFuture<List<OrderInfo>> orderListCF = CompletableFuture.supplyAsync(() -> {
    // Call OrderService API
}).thenCompose(orderListObject -> CompletableFuture.supplyAsync(() -> {
    // 조회된 주문별로 다른 서비스 호출

    CompletableFuture<Product> productInfoCF = CompletableFuture.supplyAsync(() -> {
        // Call product Service API
    });

    CompletableFuture<Delivery> deliveryInfoCF = CompletableFuture.supplyAsync(() -> {
        // Call product Service API ( HATEOAS API 호출 )
    });

    // 모든 작업이 끝나기를 기다린다.
    CompletableFuture.allOf(productInfoCF, deliveryInfoCF).join();

    // 데이터를 조합한다
    OrderInfo orderInfo = new OrderInfo(order, productInfoCF.get(), deliveryInfoCF.get());
});
```

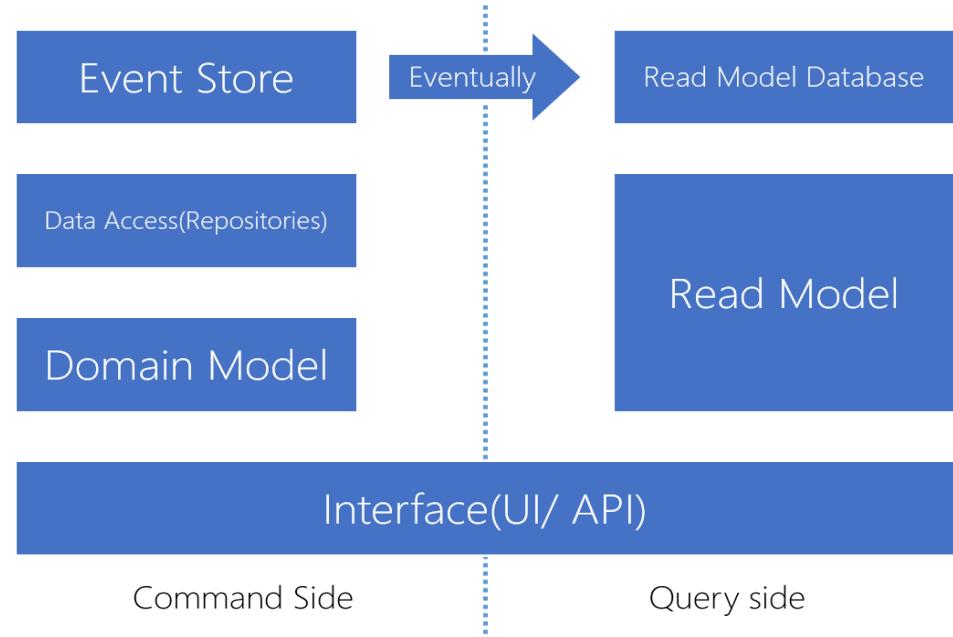
https://github.com/event-storming/composite_service/blob/master/src/main/java/com/example/template/CompositeService.java

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

- 참고 코드 실행
 - git clone https://github.com/event-storming/composite_service.git
 - cd composite_service
 - mvn spring-boot:run
- 주문 여러건 하기
 - http localhost:8081/orders productId=2 quantity=1 customerId="1@uengine.org" customerName="홍길동" customerAddr="서울시"
- 호출해보기
 - http localhost:8088/composite/orders/1@uengine.org
 - Thread 부분을 주석을 해제하고 서비스 재시작 후 호출
- 단점 확인해보기
 - 주문, 배송, 상품 서비스가 모두 가동중이어야 데이터 조회가 됨
 - 주문이력이 많을시에 모든 데이터를 조회하기 때문에 시간이 많이 걸림
 - 각 호출 API 별로 return 되는 data 를 알고 있어야 함 (각 서비스에서 변경시 잦은 변경 요청)

생각을 다르게 하자 : CQRS and Event-Sourcing

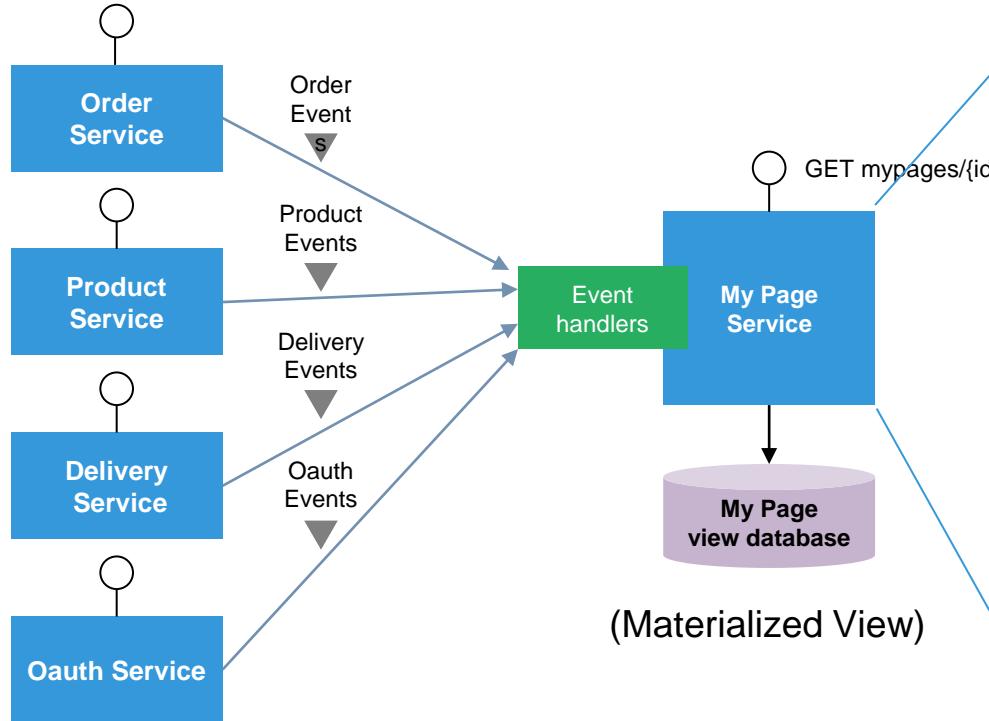
- 데이터 원본에서 매번 가져오지 말고, 취합된 별도의 뷰를 미리 만들어놓자
- Command (Write) / Query (Read) Responsibility Segregation
- 읽기전용 DB와 쓰기 DB를 분리함으로써 빠른 읽기 구현
- Query 뷰를 다양하게 구성하여 여러 MSA 서비스 목적에 맞추어 각 서비스의 Polyglot Persistence 구현



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

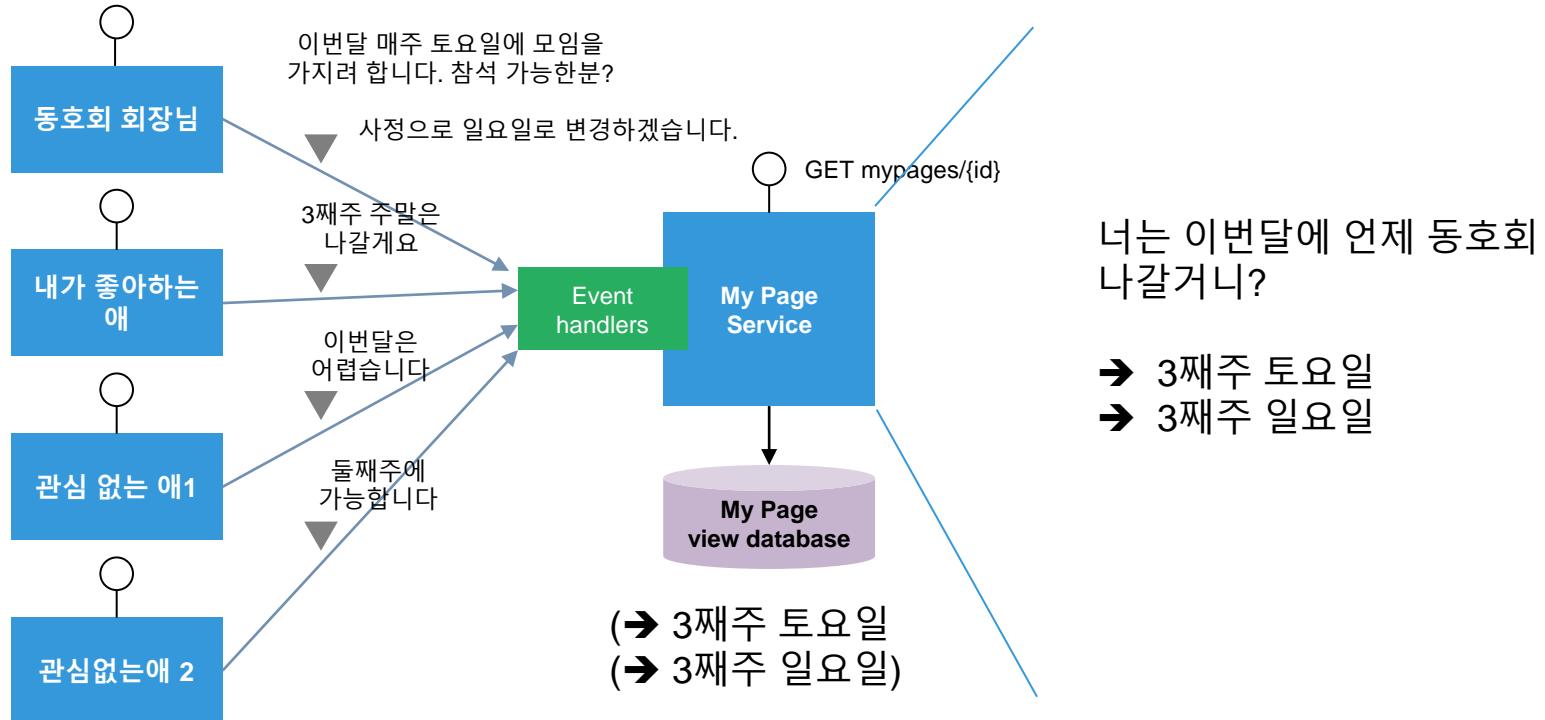
<https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-1-richardson>

CQRS 의 확장 : Multiple Event Sources



- Issues :
- 장점 : 성능, 장애 격리
 - 단점 : 다이나믹 쿼리는 불가

CQRS 의 확장 : Multiple Event Sources

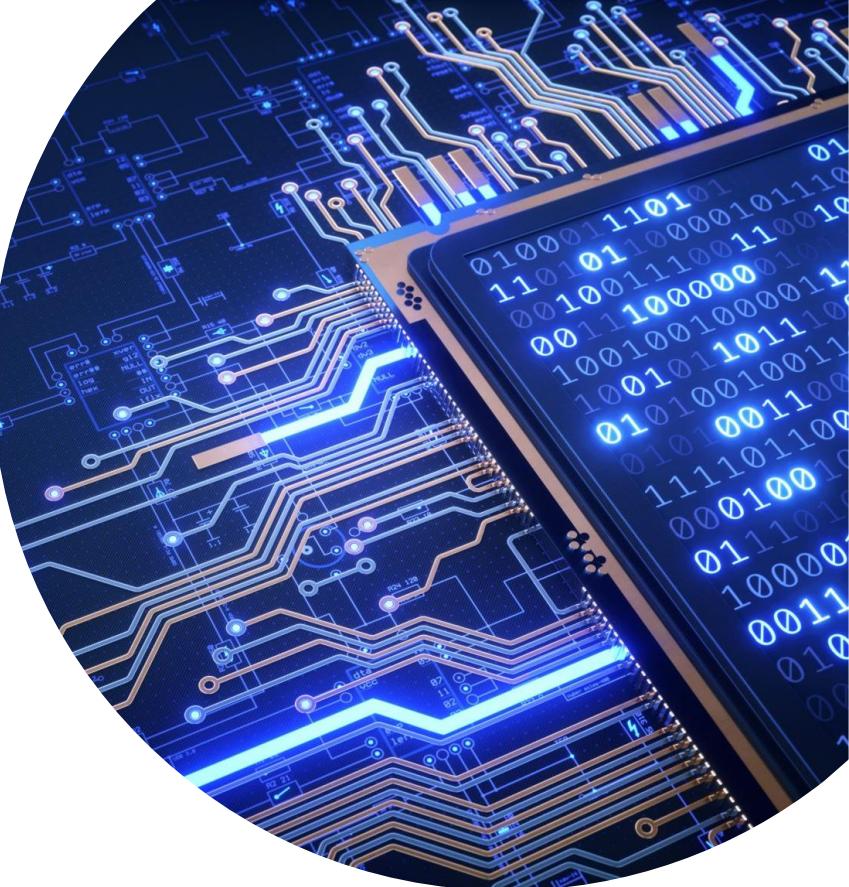


Integration Strategies Compared

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

다음 중 Saga 패턴이 해소하고자 하는 것은?

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



Quiz

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

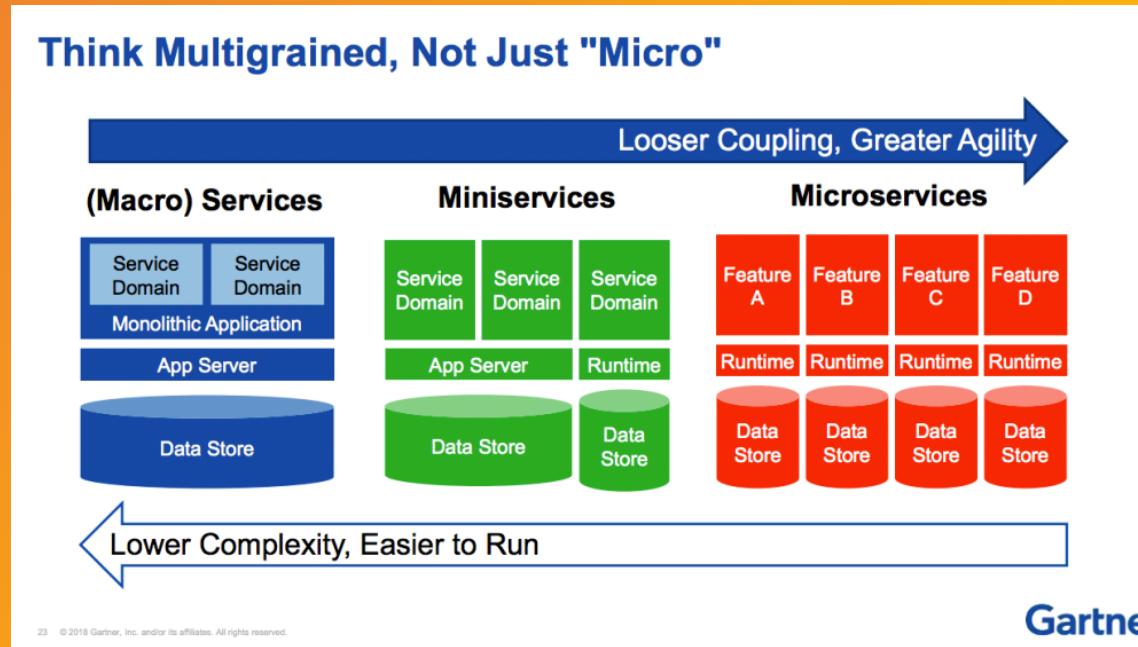


Quiz

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

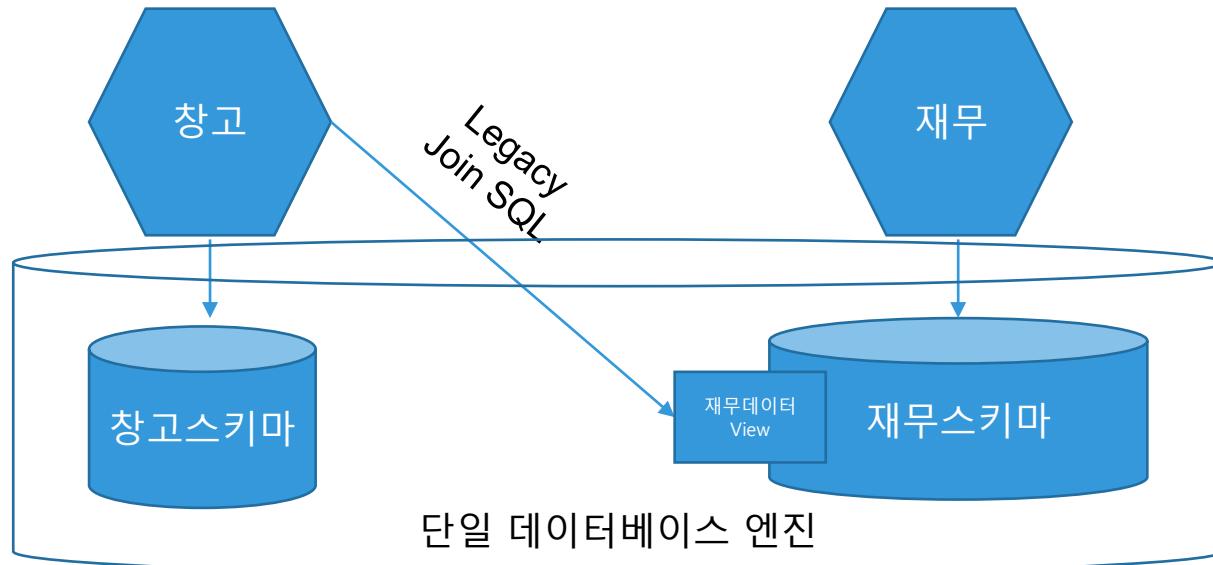
“ Tip: Mini-Service 접근을 통한 중도적 전략

- 본 챕터에서는 지금까지 배워본 정통 MSA 와 EDA의 기법을 적용하지 않고 빠른 시간 내에 MSA의 효과를 기대할 수 있는 예외 전략들을 알아봅니다.
- 이 방법들은 몇가지 제한적인 한계점이 존재하며 장기적으로는 지금 까지 배운 정공법을 통한 개선을 필요로 하지만 빠르게 비즈니스 적 효과를 거둘 수 있습니다.
- 가트너에서는 이러한 접근을 “미니 서비스”라고 부르며, MSA 아키텍처로의 전환기에 일시적으로 활용 할 수 있는 접근으로만 추천하고 있습니다.



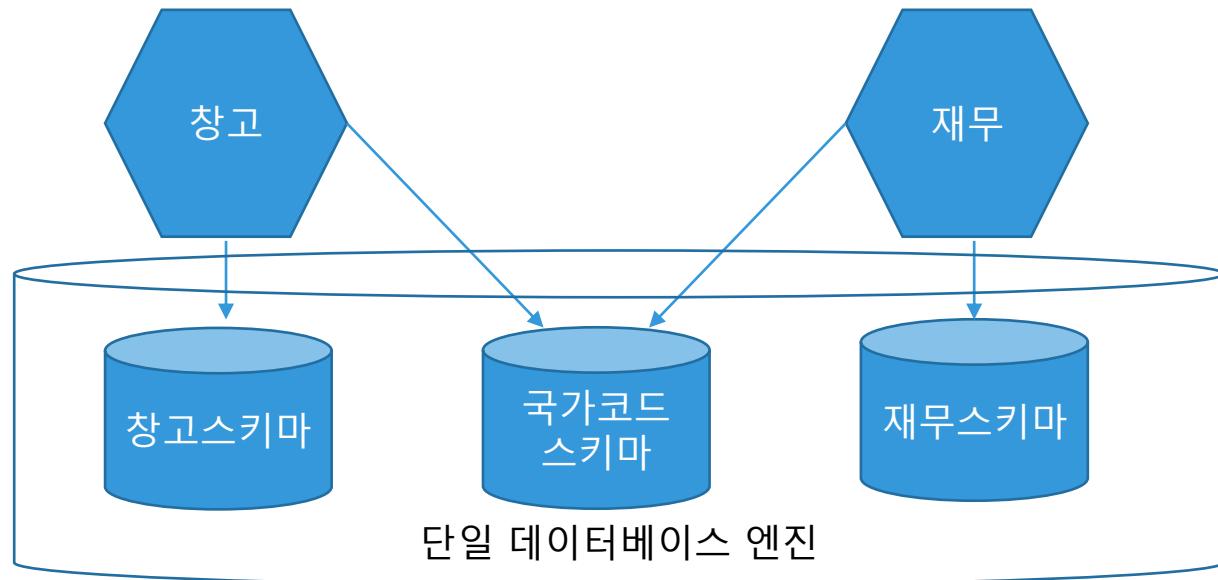
Shared Database but Schema Per Service:

- 기존 레거시가 데이터베이스 참조에 매우 의존적으로 구현된 경우 (Join SQL) 적합
- 마이크로서비스 초기에는 공유된 데이터베이스 엔진에 두되, 엄격한 스키마 분리를 통해 스키마 관리 권한과 쓰기 권한을 분리
- 기존 SQL은 뷰를 통한 (인터페이스) 접근만을 허용하여 데이터베이스 내부 구조에 대한 익닉을 도모할 수 있다 (View를 제공하는 측에서 View에 맞춘 쿼리만 변경)

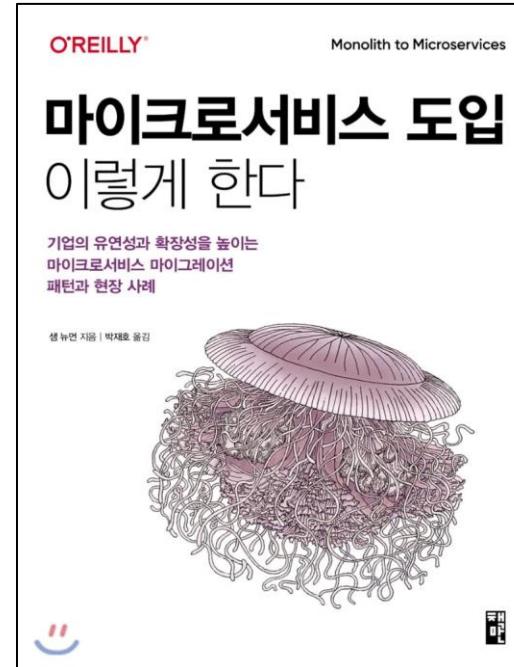
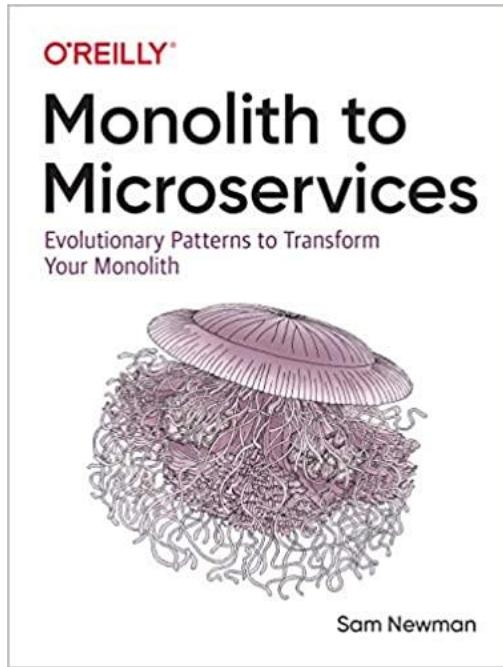


공통코드 성격의 데이터 참조

- E.g. 국가코드, 사용자 정보 등의 공통적으로 모두 사용해야 할 정보의 접근
- 공유된 접근 스키마 or 서비스를 통하여 구현



참고도서



Comparison of Inter-Mi-Svc Comm. Strategies

Integration Pattern / Tool	Database (Schema per Service)	UI	Req-Resp	Pub-Sub
Reading (Query)	View	UI Composition + HATEOAS	API Composition HTTP Request + Circuit Breaker	CQRS
Writing	Trigger	Not Recommended	2PC	Saga (Eventual Consistency)

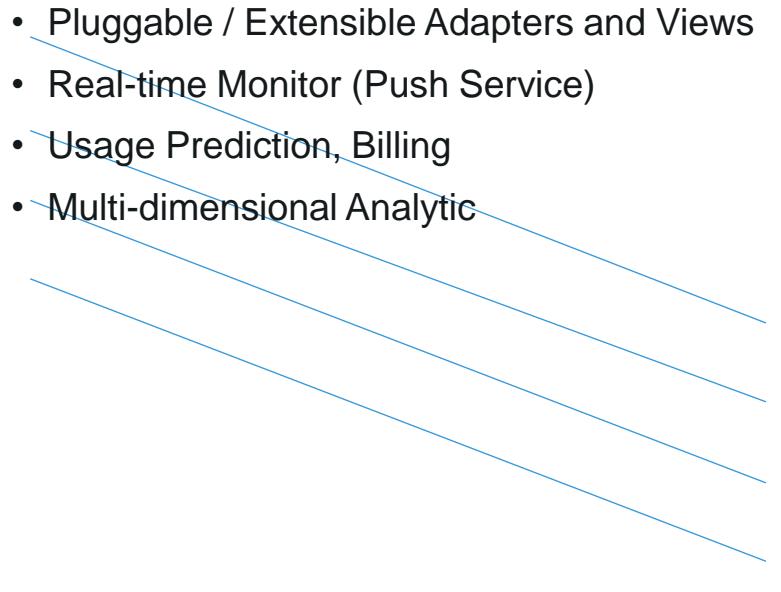
Mini Service **SOA** **MSA**

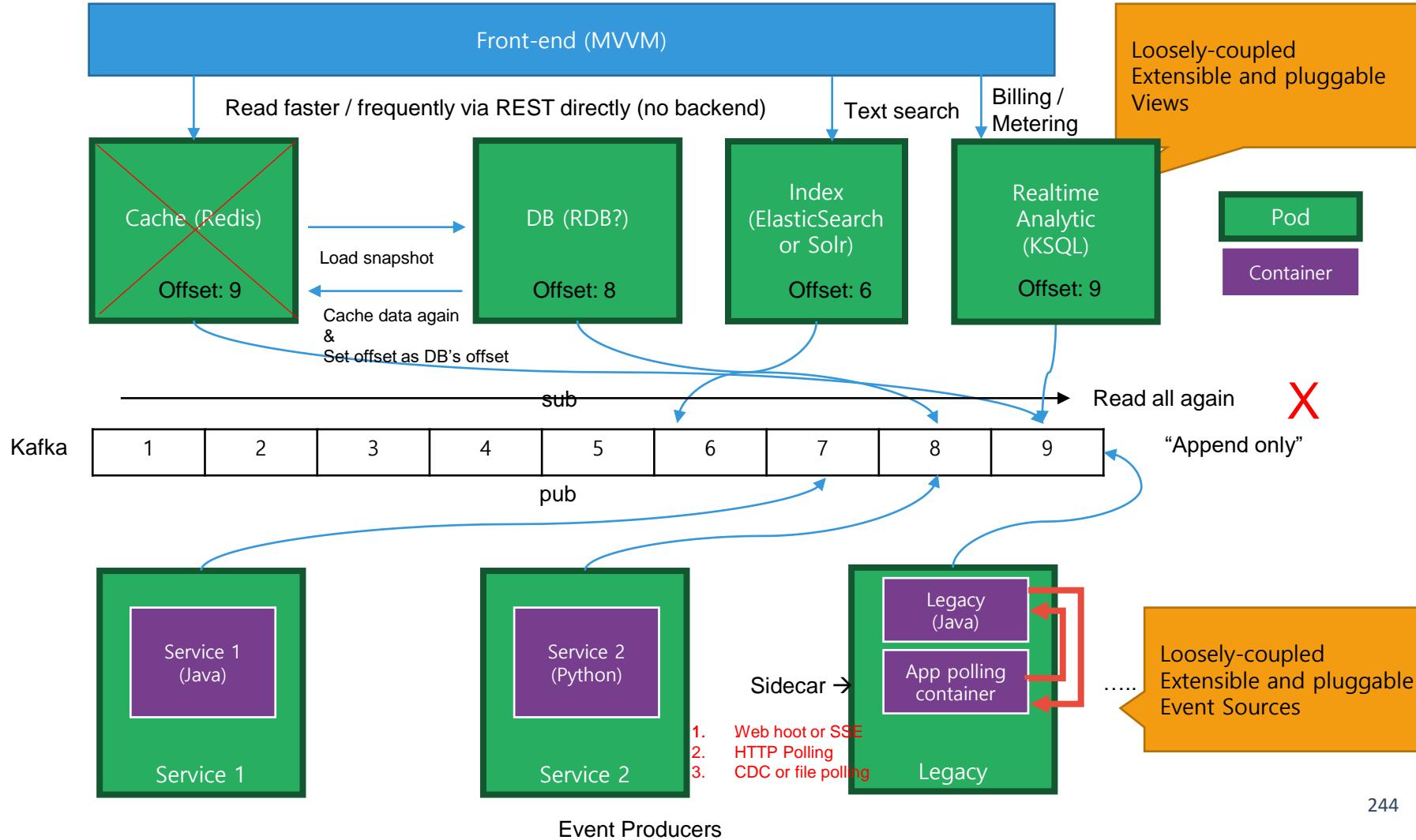
“

Case Study:

Event Sourcing 기반 예금 입출금

Design Principle

- Pluggable / Extensible Adapters and Views
 - Real-time Monitor (Push Service)
 - Usage Prediction, Billing
 - Multi-dimensional Analytic
- 
- Event-driven Microservices
 - Server-sent event
 - Stream Processing
 - Multiple Views / Polyglot Persistence



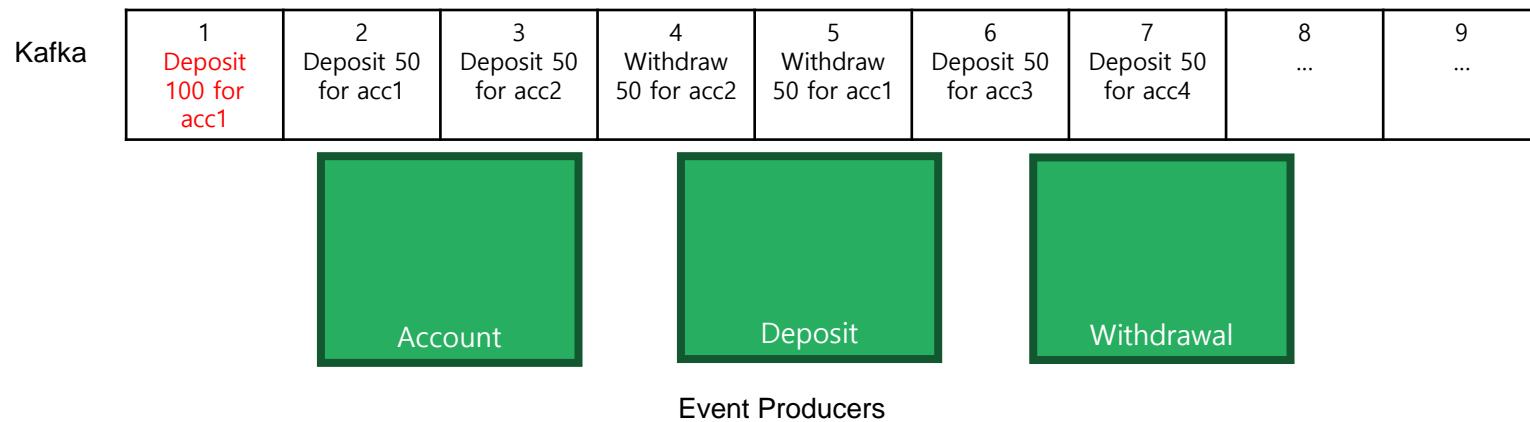
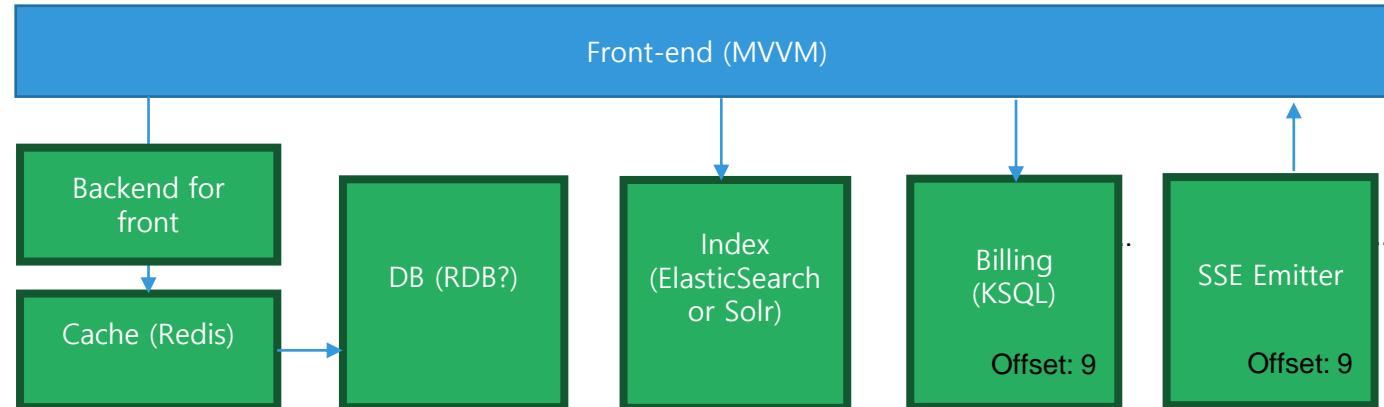
Event Formatting: Event Sourcing

Kafka

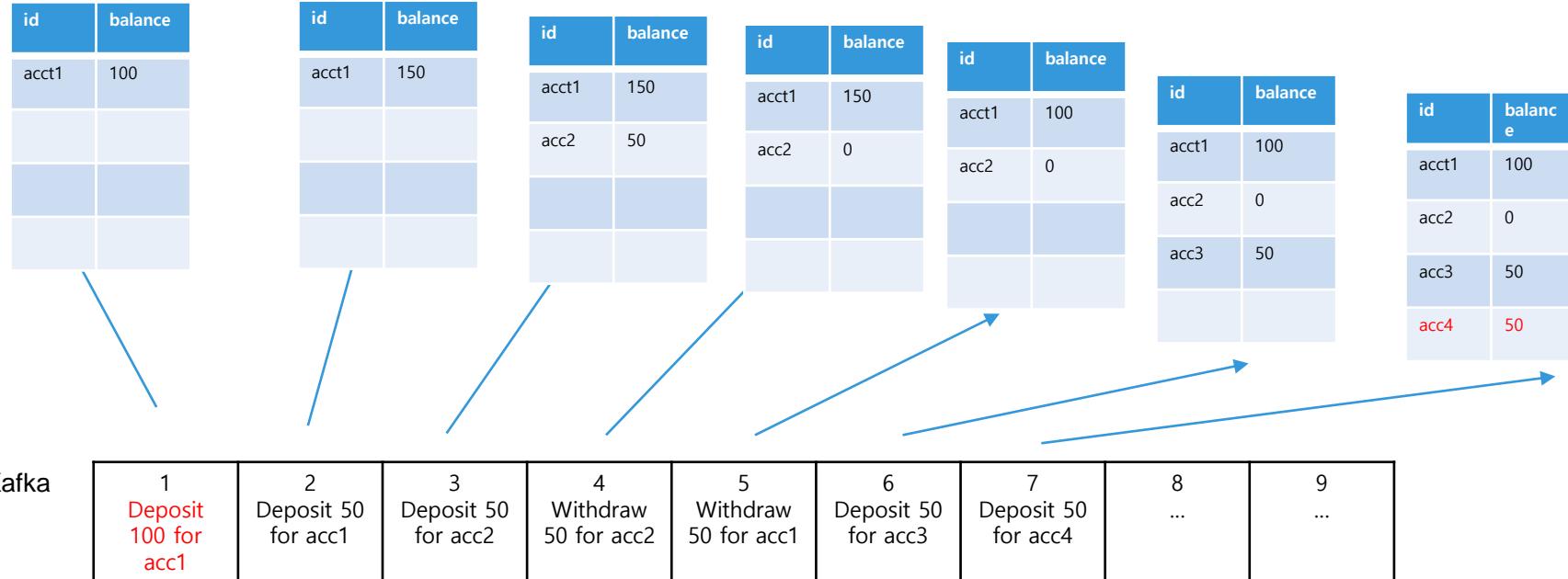
1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Append the “Diff” only

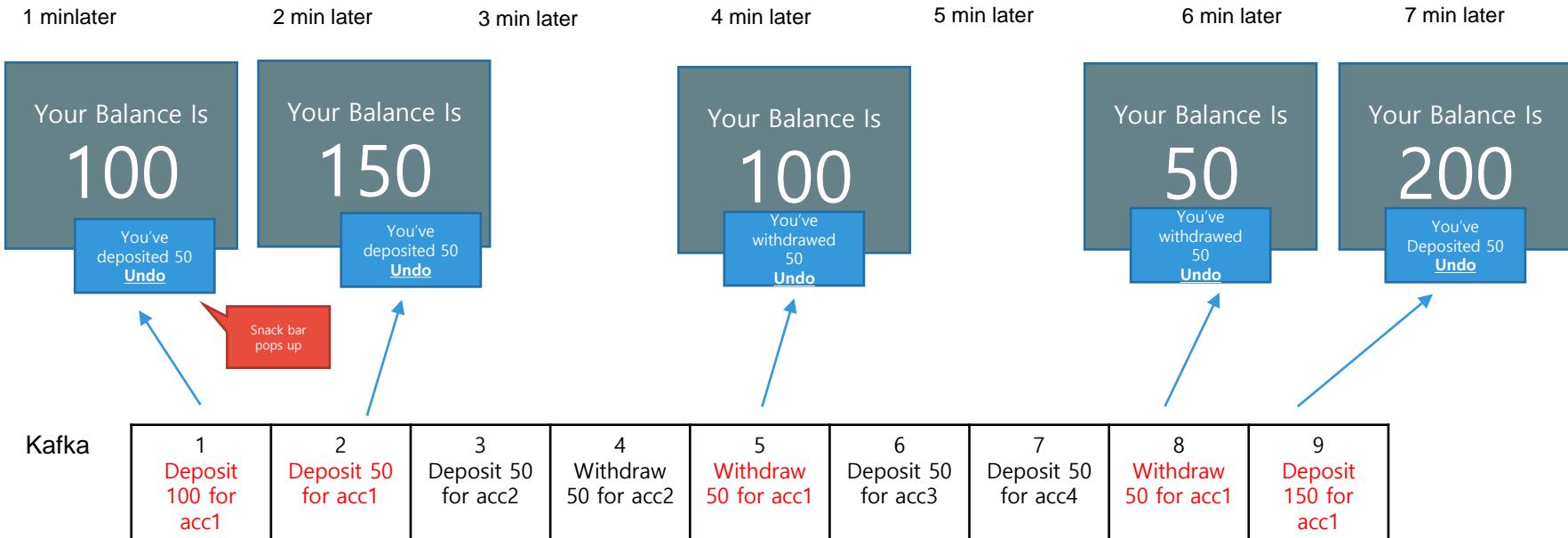
- 1: key: account1, value: { action: Deposit, amount: 100 }
- 2: key: account1, value: { action: Deposit, amount: 50 }
- 3: key: account2, value: { action: Deposit, amount: 50 }
- 4: key: account2, value: { action: Withdraw, amount: 50 }
- 5: key: account1, value: { action: Withdraw, amount: 50 }
- 6: key: account3, value: { action: Deposit, amount: 100 }
- 7: key: account4, value: { action: Deposit, amount: 50 }
-



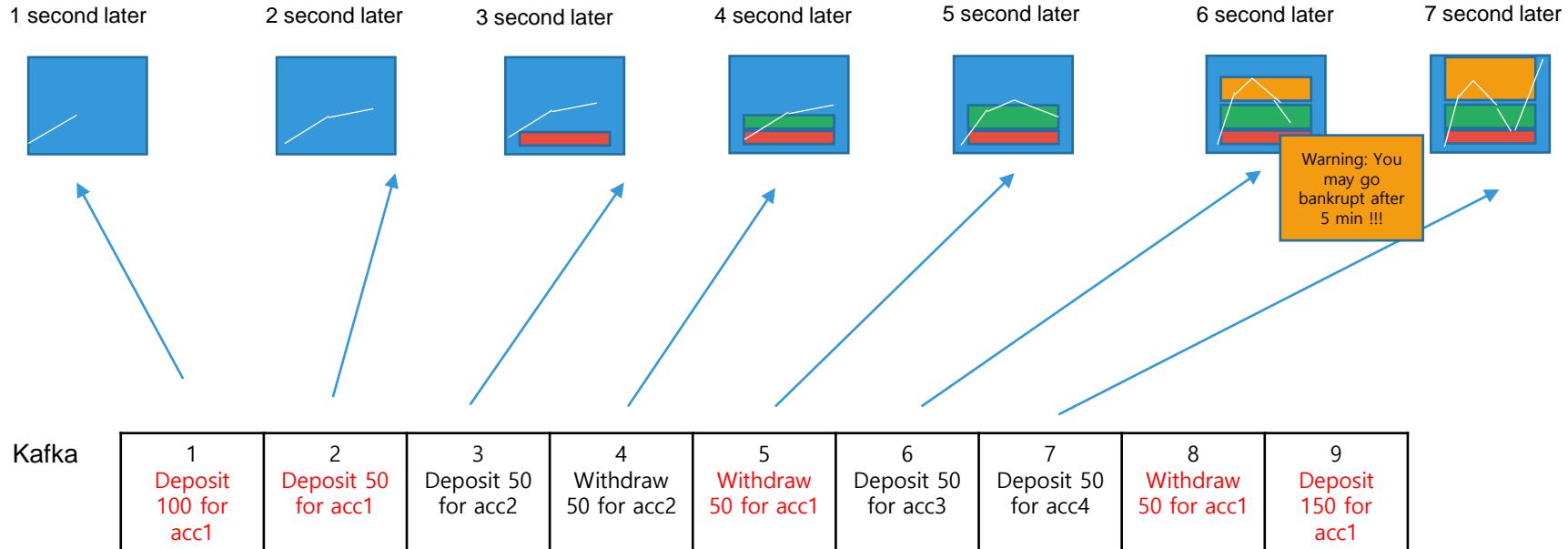
Aggregate view in Database



View in Front-end (Push service)



View in Realtime Analytic (Early Warning)



** Using KSQL or Kstreams:

“

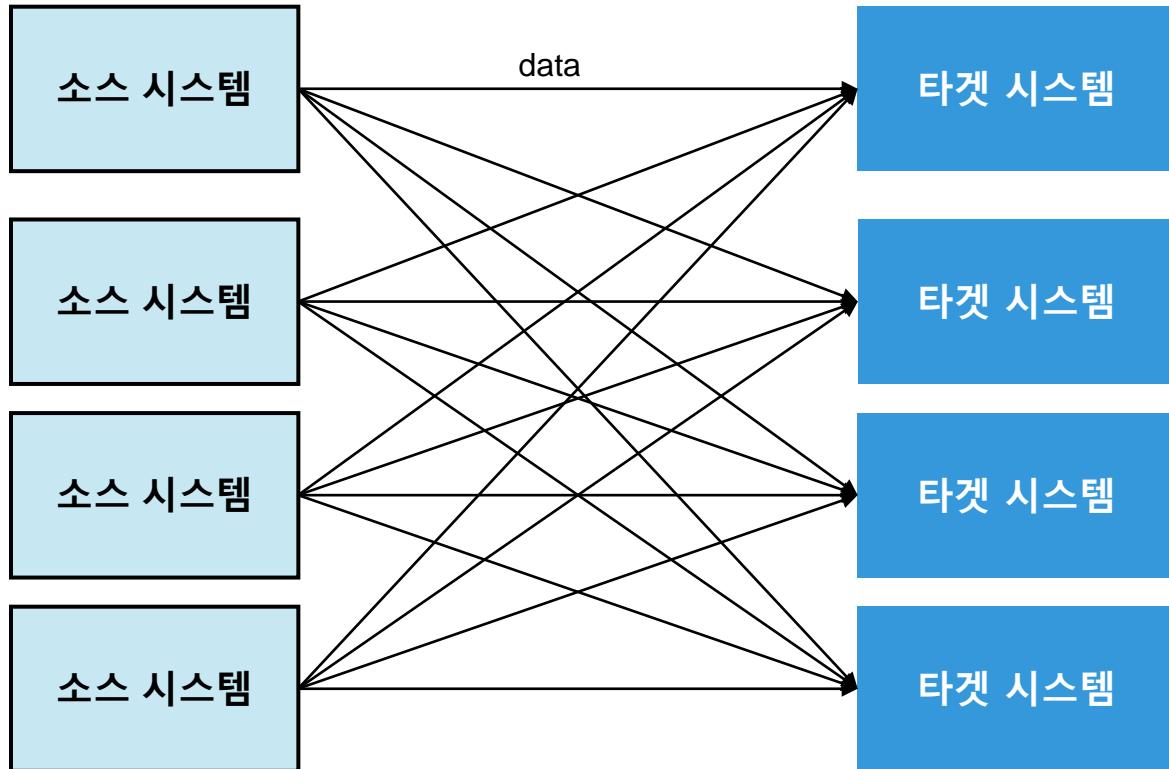
Apache-Kafka

- 탄생배경
- 구성요소: Broker, Zookeeper
- 관리객체: Topic, Partition, Producer, Consumer
- Scaling Kafka

Apache Kafka

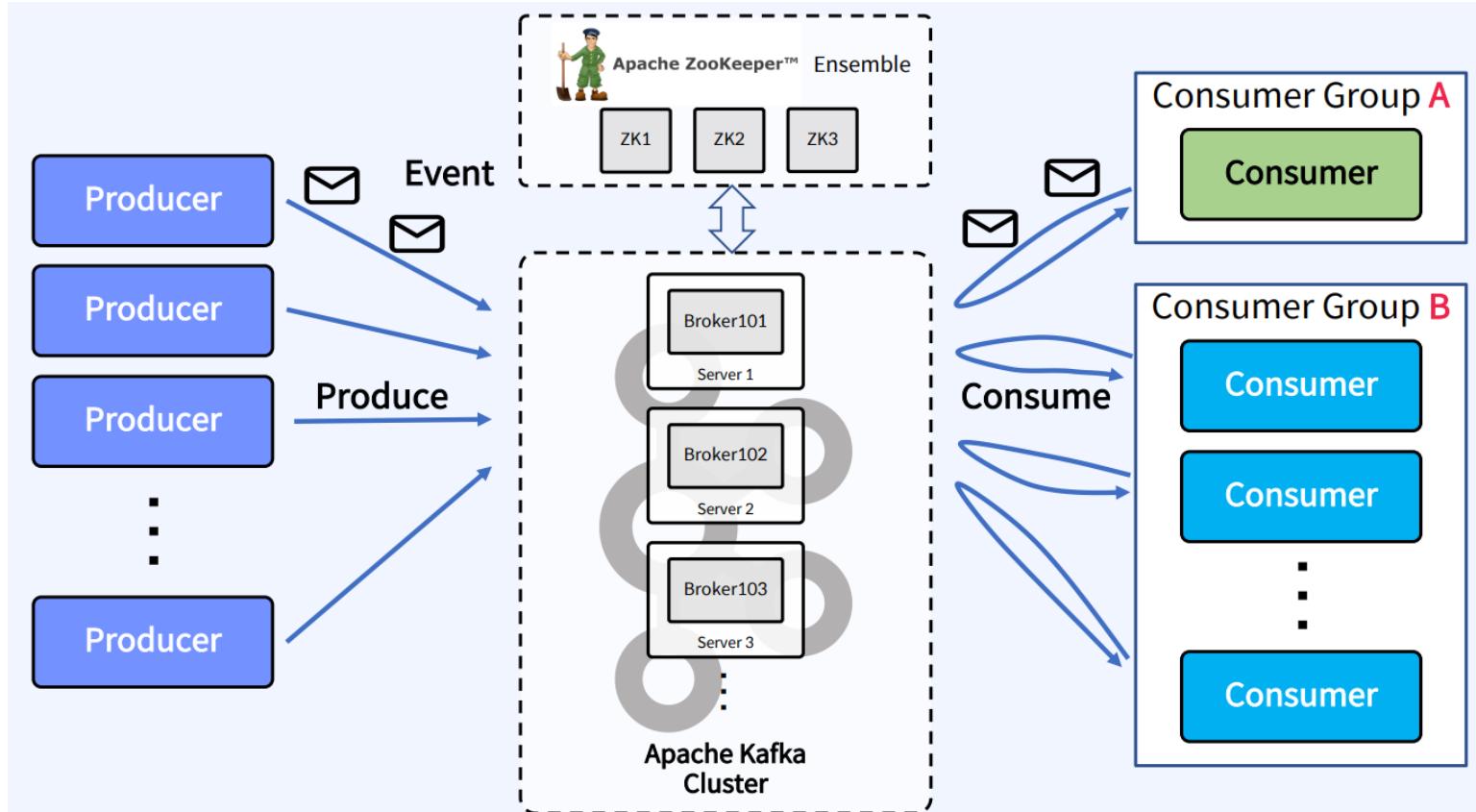
- 2010년에 링크드인에서 개발
- 2012년에 아파치 오픈소스로 등록됨
- 2014년에 링크드인 개발자들이 나와서 컨플루언트라는 회사를 창립해 카프카를 계속 발전시킴
- 확장 (Distributed), 유연 (Resilient), 장애에 강한 (fault tolerant) 아키텍처
- 수평 확장 가능
 - 100개의 Broker 까지 확장
 - 1초에 100만개의 메세지
 - 무중단 확장 가능
- 높은 퍼포먼스 (10ms 미만의 지연) – real time

탄생 배경



- 소스 시스템과 타겟시스템이 많아 지면서 복잡
- 어려운 이유?
 - 프로토콜(http, TCP, RPC, FTP, JDBC ..)
 - 데이터 포맷(json, binary, csv, xml)
 - 데이터 스키마 다양함 (각 시스템별로)
- 각각의 시스템이 데이터 포맷과 처리하는 방식이 달라서 확장 및 통합 이 어려워짐

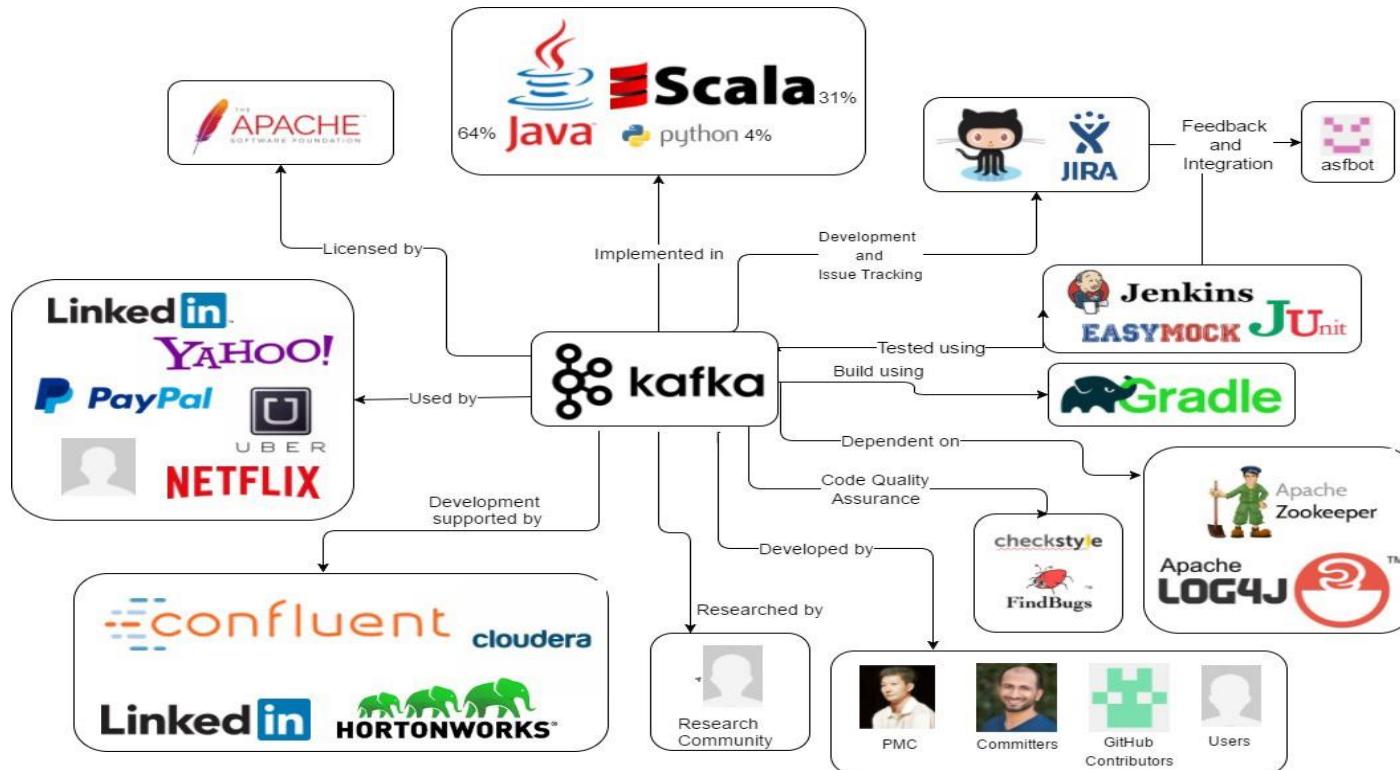
카프카 구성도



Apache Kafka 를 어떻게 쓸까요?

- 메세징 시스템
- Activity Tracking
- 모니터링 데이터 수집 (Gather metrics)
- 로그 수집 (Log Aggregation)
- Stream Processing
- 시스템간의 종속성 분리 (De-coupling of System Dependencies)
- Spark, Storm, Flink, Hadoop 등 많은 빅데이터 기술과 통합하여 사용
- <https://kafka.apache.org/uses>

Apache Kafka 를 어떻게 쓸까요?



Topics, Partitions, Offsets

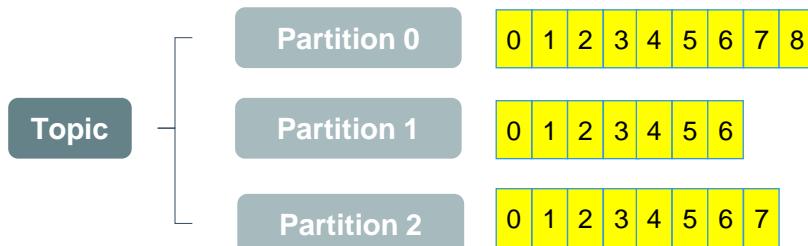
- 토픽

- 특정 데이터 스트림, 즉 메세지를 구분 하는 통로
- 하나의 카프카에 여러 메세지가 뒤섞여서 오가는데, 거기서 토픽으로 구분하여 원하는 메세지를 찾는 방식
- 데이터베이스의 테이블, 카톡의 단톡방 과 비슷한 개념
- 토픽은 여러개 만들수 있고, 이름으로 구분됨
- 토픽 이름을 어떻게 만들어야 하나?
 - <https://riccomini.name/how-paint-bike-shed-kafka-topic-naming-conventions>
 - <https://devshawn.com/blog/apache-kafka-topic-naming-conventions/>
 - <project>.<product>.<event-name>
 - services, consumers, or producers 등과 연결하여 만들지 않기를 추천함

Topics, Partitions, Offsets

- 파티션

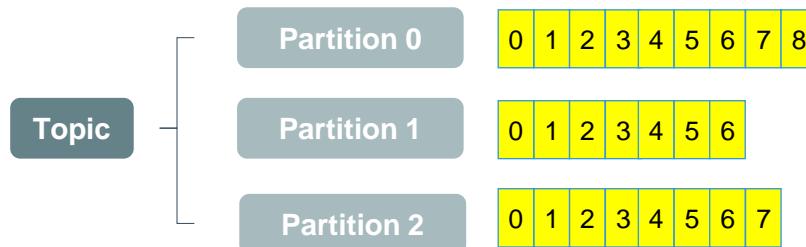
- 토픽은 파티션을 나눌 수 있음
- 토픽당 데이터를 분산 처리하는 단위
- 병렬 처리와 많은 양의 데이터 처리를 위해서 파티션을 늘리 수 있음
- **늘리기만 하고 줄이는 것은 안됨**
- 각각의 파티션은 0부터 1, 2, ... 의 Index를 가짐
- 파티션 별로 증가되는 아이디, 즉 Offset라는 값을 가지게 됨



Topics, Partitions, Offsets

- 토픽과 파티션

- Offset 은 파티션에서만 의미가 있음
 - 파티션 3개가 있다면 각각의 파티션의 offset 0번은 다른 데이터를 가지게됨
- 파티션 안에서는 데이터의 순서가 보장
 - 파티션이 복수개 일때, 1번과 2번 파티션에 적재되는 데이터의 순서성 보장이 안됨
 - 파티션에 데이터가 한번 쓰여지면 변경이 안됨
 - key 를 주지 않으면 어느 파티션에 데이터가 들어갈지 모름



Broker

- 카프카를 클러스터로 구성하였을 때, 각각의 서버를 Broker 라고 부름
- 각각의 브로커는 토픽 파티션을 가짐
- 어떤 브로커에 접속해도 전체 카프카 클러스터에 접속 가능
- 브로커는 3개로 시작하여 100개까지 늘어날 수 있음

Broker1

Broker2

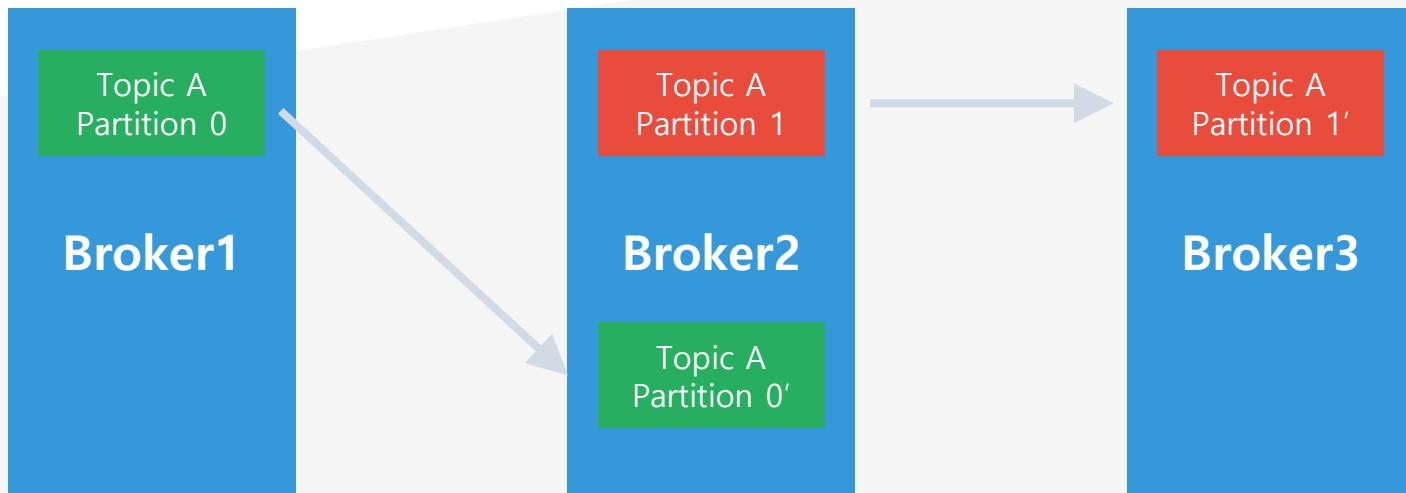
Broker3

Zookeeper

- 분산 애플리케이션 코디네이터
- 주키퍼는 브로커를 관리한다.
- 파티션의 리더 선출을 도와준다.
- 카프카의 변경에 대하여 알림을 준다 (토픽생성, 브로커 die, 브로커 up, 토픽 삭제)
- **카프카는 주키퍼 없이 동작 못함**
- 주키퍼 클러스터는 홀수로 동작함 (Leader/Follower 선출때문에 짝수로 동작 못함)
- 주키퍼 Leader 는 write, Follower 는 read 작업을 함

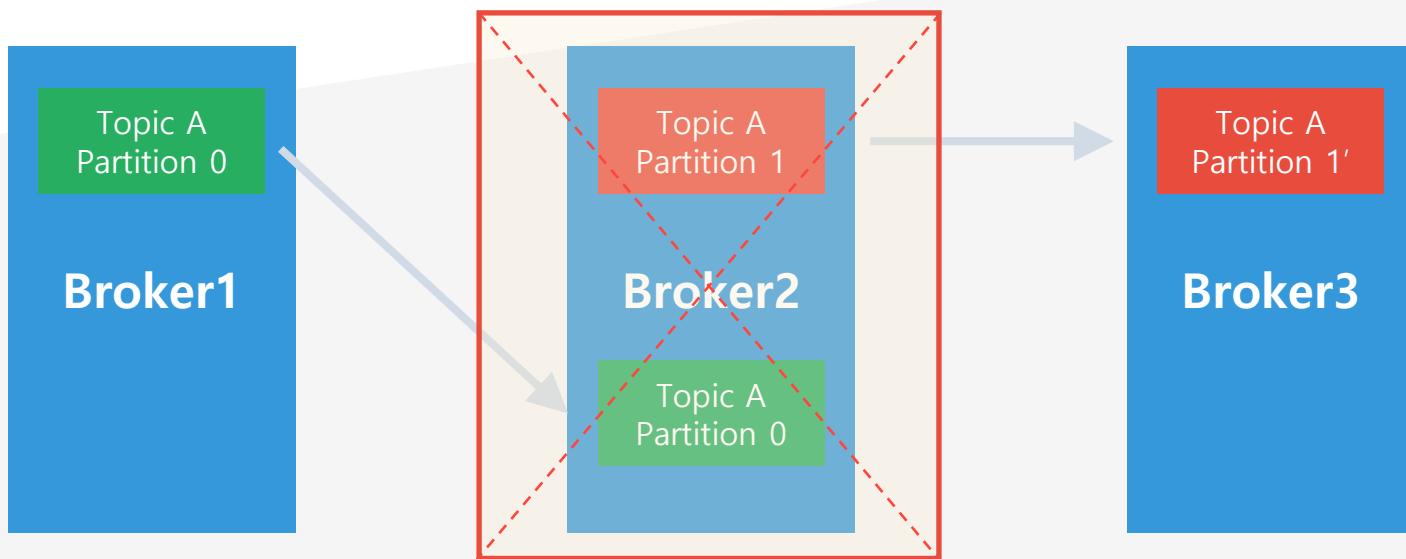
Topic Replication factor

- 토픽을 이루는 파티션을 복제하는 기능
- Topic A 의 파티션이 2 개, Replication factor 가 2 라면?



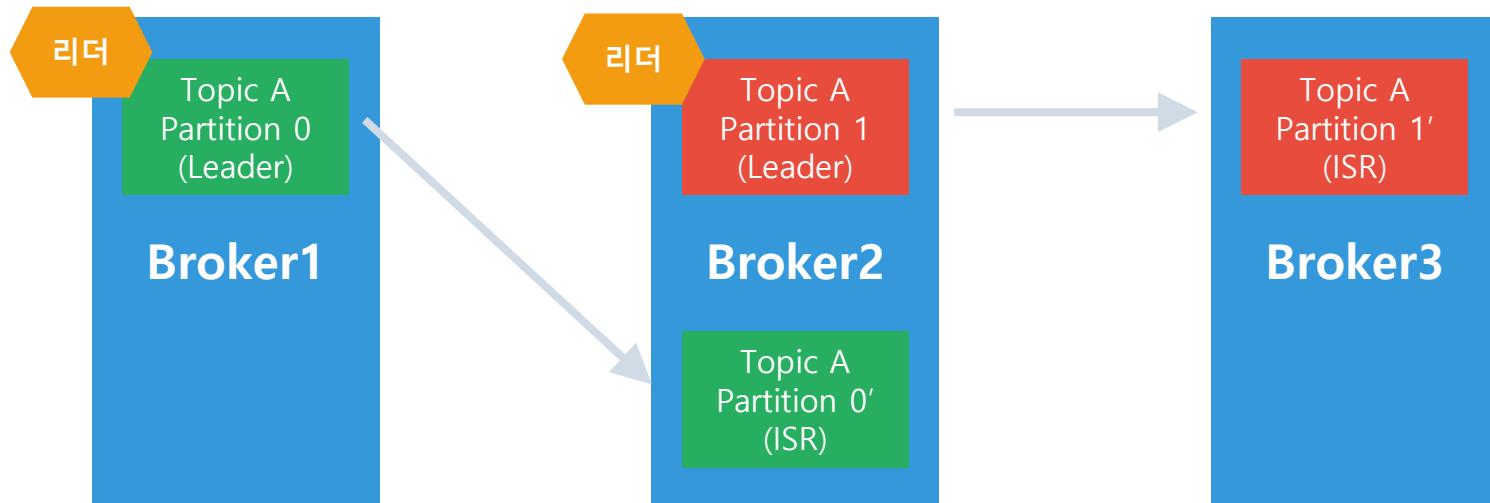
Topic Replication factor

- 만약 Broker 2 가 장애가 발생해도 토픽에 정상 데이터가 수신됨
- 만약 토픽의 데이터가 3GB 면, Replication factor가 2일 경우, 6GB 의 데이터 용량



파티션의 Leader

- 파티션의 리더만 데이터를 읽고, 쓰기가 가능
- 나머지 파티션은 ISR(In Sync Replica) 역할을 하게됨
- ISR은 리더의 데이터를 체크하여 복제하고, 리더가 장애시 리더 역할을 함



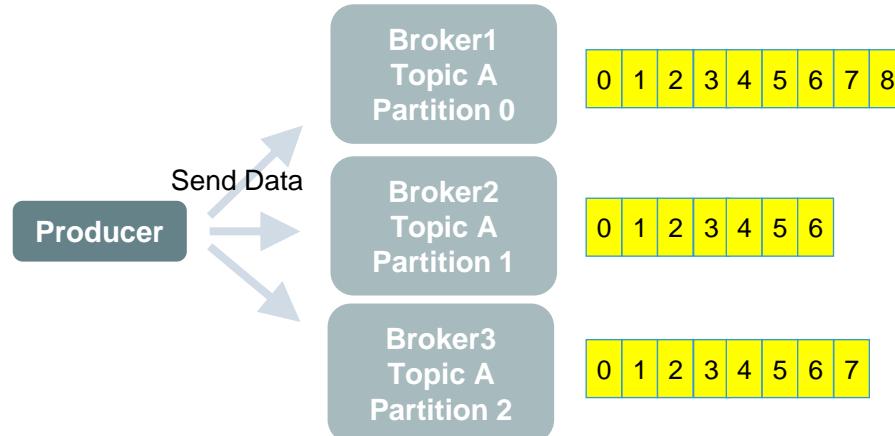
Broker-0 장애 = Leader 재선출

Topic:f9Topic	PartitionCount:9	ReplicationFactor:3	Configs:
Topic: f9Topic	Partition: 0	Leader: 0	Replicas: 0,1,2 Isr: 1,2,0
Topic: f9Topic	Partition: 1	Leader: 1	Replicas: 1,2,0 Isr: 1,2,0
Topic: f9Topic	Partition: 2	Leader: 2	Replicas: 2,0,1 Isr: 1,2,0
Topic: f9Topic	Partition: 3	Leader: 0	Replicas: 0,2,1 Isr: 1,2,0
Topic: f9Topic	Partition: 4	Leader: 1	Replicas: 1,0,2 Isr: 1,2,0
Topic: f9Topic	Partition: 5	Leader: 2	Replicas: 2,1,0 Isr: 1,2,0
Topic: f9Topic	Partition: 6	Leader: 0	Replicas: 0,1,2 Isr: 1,2,0
Topic: f9Topic	Partition: 7	Leader: 1	Replicas: 1,2,0 Isr: 1,2,0
Topic: f9Topic	Partition: 8	Leader: 2	Replicas: 2,0,1 Isr: 1,2,0

Topic:f9Topic	PartitionCount:9	ReplicationFactor:3	Configs:
Topic: f9Topic	Partition: 0	Leader: 1	Replicas: 0,1,2 Isr: 1,2,0
Topic: f9Topic	Partition: 1	Leader: 1	Replicas: 1,2,0 Isr: 1,2,0
Topic: f9Topic	Partition: 2	Leader: 2	Replicas: 2,0,1 Isr: 1,2,0
Topic: f9Topic	Partition: 3	Leader: 2	Replicas: 0,2,1 Isr: 1,2,0
Topic: f9Topic	Partition: 4	Leader: 1	Replicas: 1,0,2 Isr: 1,2,0
Topic: f9Topic	Partition: 5	Leader: 2	Replicas: 2,1,0 Isr: 1,2,0
Topic: f9Topic	Partition: 6	Leader: 1	Replicas: 0,1,2 Isr: 1,2,0
Topic: f9Topic	Partition: 7	Leader: 1	Replicas: 1,2,0 Isr: 1,2,0
Topic: f9Topic	Partition: 8	Leader: 2	Replicas: 2,0,1 Isr: 1,2,0

Producer

- 메세지를 생산(produce) 하여 토픽으로 메세지를 보내는 애플리케이션, 서버등을 Producer 라고 부른다.
- 메세지를 전송할때 Key 옵션을 줄 수 있는데, Key 값을 설정하여 특정 파티션에 메세지를 보낼수 있다.
- Key 옵션을 주지 않을때는 파티션에 라운드Robin 방식으로 균등하게 메세지를 낸다.

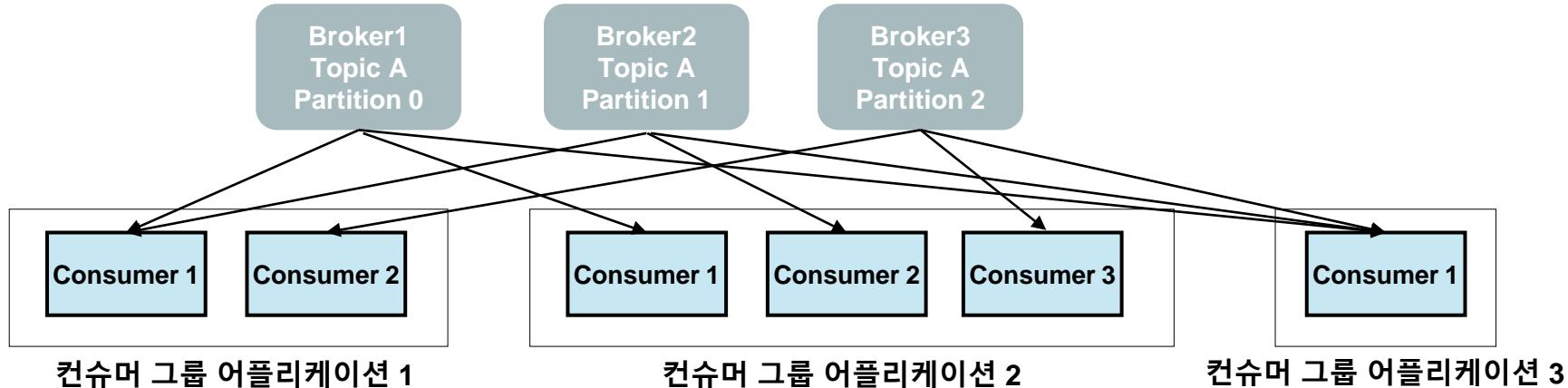


Producer

- 메세지 손실 가능성이 높지만 빠른 전송이 필요한 경우
 - acks=0 프로듀서는 응답을 기다리지 않는다.
- 메세지 손실 가능성이 적고, 적당한 속도의 전송이 필요한 경우
 - acks=1 프로듀서는 리더의 응답만 체크 한다. (default)
- 전송 속도는 느리지만 메세지 손실이 없어야 하는 경우
 - acks=all 프로듀서는 리더와 ISR 의 모든 응답을 체크한다.
 - Replication 이 없는 경우라면 acks=1 과 동일하다.

Consumer

- 토픽 이름으로 저장된 메세지를 가져가는 앱, 서버등을 Consumer 라고 부른다.
- Consumer 들의 집합을 Consumer Group 이라고 부르며, 카프카는 Consumer Group 단위로 데이터를 처리한다.
- Consumer Group 안의 각각의 Consumer 는 파티션의 데이터를 읽는다.

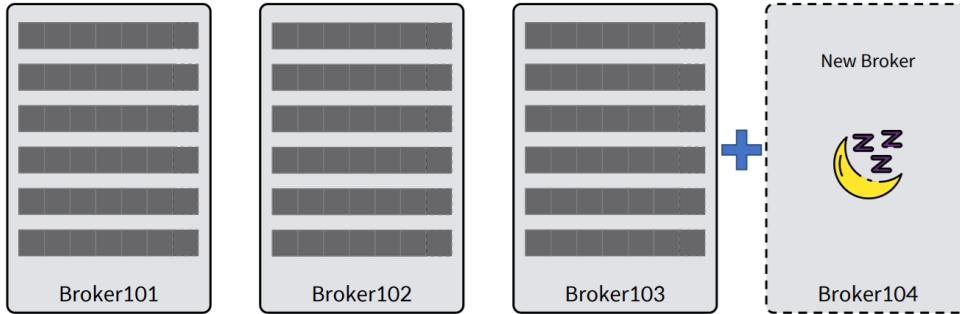


Consumer Offset

- 카프카는 컨슈머 그룹이 어디까지 메세지를 가져갔는지 Offset 을 체크한다.
- 컨슈머 그룹은 메세지를 가져간 후에 offset 을 Commit 한다.
- 카프카는 내부적으로 별도의 토픽으로 커밋된 정보를 저장한다.
 __consumer_offset 형식으로 저장됨
- 만약 컨슈머가 죽었을때, 해당 offset 정보를 읽어서 어디부터 데이터를 읽을지 파악하고, offset 다음부터 메세지를 수신한다.

Kafka Scaling - Broker

- 카프카 운영시, 모든 브로커의 리소스 사용률이 현저히 높을 경우, 새로운 브로커를 추가한다.
 - 고유 ID를 부여한 새로운 Broker를 추가 후 Zookeeper Ensemble에 연결해 준다.
 - 추가 후 kafka-reassign-partitions 명령으로 Topic 별로 리밸런싱을 직접 실행



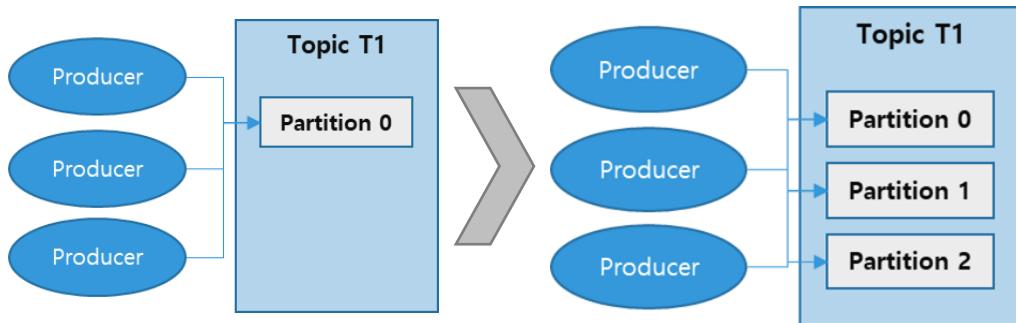
```
# Broker에 파티션 재할당  
kafka-reassign-partitions.sh \  
--bootstrap-server kafka1:9092 --zookeeper zk1:2181 \  
--reassignment-json-file reassignment.json \  
--execute
```

```
# 파티션 재할당 JSON 생성  
{  
    "partitions":  
    [  
        { "topic": "foo", "partition": 0, "replicas": [4,1,2] },  
        { "topic": "foo", "partition": 1, "replicas": [1,2,3] },  
        { "topic": "foo", "partition": 2, "replicas": [2,3,4] }  
    ],  
    "version":1  
}
```

실행명령

Kafka Scaling - Partition

- 카프카 운영시, 메시지 저장에 병목이 발생할 경우, 해당 토픽의 파티션을 증가한다.
 - 파티션 수가 많을수록 파일 핸들러가 증가하므로 적절한 개수의 파티션 설정 필요
 - 파티션은 증가만 되고 줄이는 방법이 없으므로 목표 처리량의 기준설정 필요
 - 컨슈머 그룹내의 컨슈머 개수와 동등한 파티션 개수 설정 권고



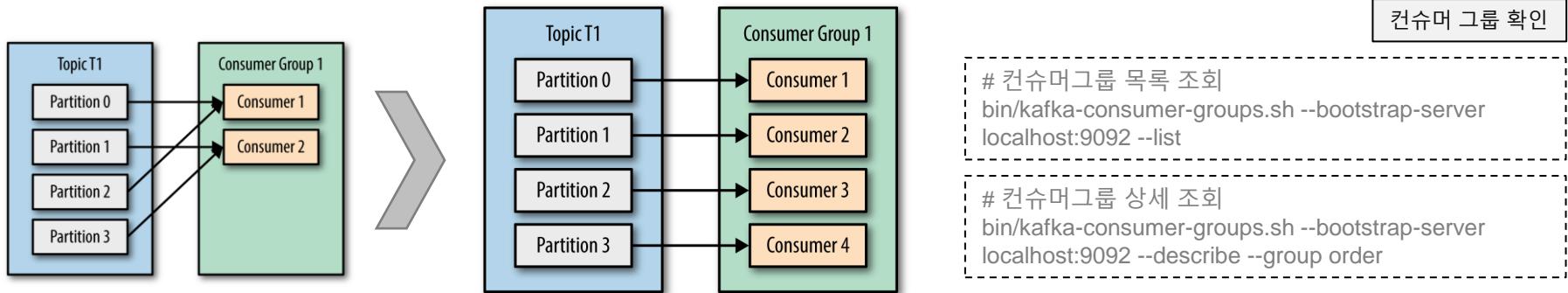
실행명령

```
# 토픽 조회  
bin/kafka-topics.sh --bootstrap-server localhost:9092 --  
describe --topic 토픽이름
```

```
# 토픽 파티션 증가  
bin/kafka-topics.sh --bootstrap-server localhost:9092 --  
topic 토픽이름 --alter --partitions 3
```

Kafka Scaling - Consumer Group

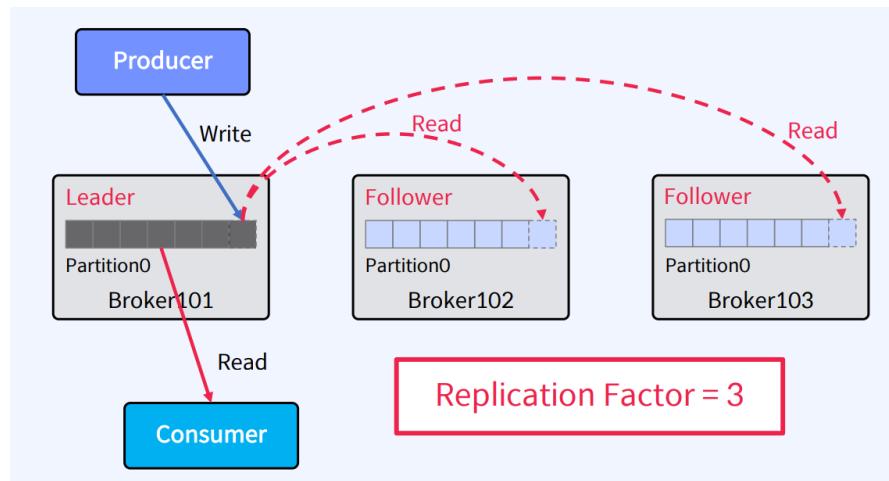
- 카프카 운영시, Message Lagging이 높을 경우, 처리를 담당하는 컨슈머를 증가한다.
 - 컨슈머 그룹별로 그룹 코디네이터(Group Coordinator)가 각 컨슈머 그룹을 관리
 - 컨슈머는 폴링(polling)하거나 커밋할 때 Heartbeat 메시지를 그룹 코디네이터에게 전달
 - 컨슈머 수가 변경되면 그룹내에서 소유권 이관작업이 일어나며 이를 리밸런싱이라 함
 - 그룹 코디네이터가 일정 기간(session.timeout.ms) 컨슈머의 하트비트를 받지 못하면, 해당 컨슈머를 장애로 판단하고 리밸런싱을 수행



Kafka Scaling - Replication Factor

- 카프카 운영시, 리더 파티션 장애를 대비해 파티션을 복제해 운영한다.
 - 리터 파티션은 읽기/쓰기를 수행하고, 팔로워 파티션은 리플리케이션만 수행
 - 리더 파티션 브로커에 장애가 발생한 경우, 팔로워가 새로운 리더가 되어 읽기/쓰기 수행
 - 리플리카 개수만큼의 저장소가 소모됨

리플리케이션 변경



```
# 리플리케이션 조회  
/usr/local/kafka/bin/kafka-topics.sh \  
--zookeeper zk1:2181 \  
--topic foo --describe
```

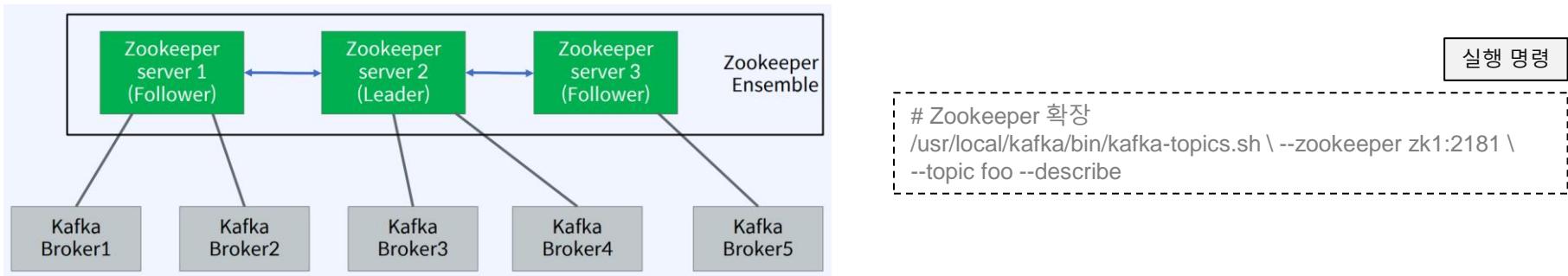
```
{ # RF 변경 JSON 생성
```

```
"partitions":  
[  
    { "topic": "foo", "partition": 0, "replicas": [1,2,3] }  
],  
"version":1  
}
```

```
# JSON 실행  
kafka-reassign-partitions.sh \  
--bootstrap-server kafka1:9092 --zookeeper zk1:2181 \  
--reassignment-json-file reassignment.json \  
--execute
```

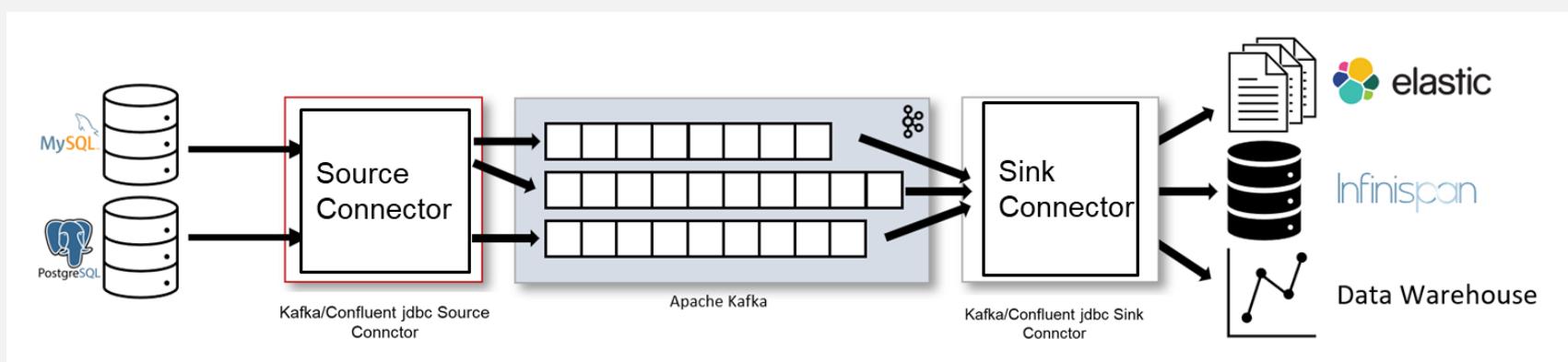
Kafka Scaling - Zookeeper

- Zookeeper는 Broker를 관리(Broker들의 목록/ 설정을 관리)하는 소프트웨어
 - Zookeeper는 변경사항에 대해 Kafka에 알린다.
 - Zookeeper를 사용해 멀티 Kafka Broker들 간의 정보(변경사항 포함) 공유 및 동기화 수행
 - (2022년에 Zookeeper를 제거한 정식 버전 출시 예정중)



Kafka Connect

- Kafka Connect는 Producer와 Consumer를 사용해 데이터 파이프라인을 생성해 준다.
- Connect와 Source Connector를 사용해 Kafka Broker로 데이터를 보내고, Connect와 Sink Connector를 사용해 Kafka에 담긴 데이터를 타겟 DB에 저장한다.



Query-Based CDC

변화를 감지하기 위한 소스 스키마에 특정 컬럼이 요구되며, Delete 는 동기화되지 않는다.

Log-Based CDC

데이터베이스를 트랜잭션 Log 레벨에서 풀링하므로 DB부하가 적으며 레이턴시가 낮다.

Kafka Connect 용어 정의

- Connect: Connector를 동작하게 하는 프로세서(서버)
- Connector: Data Source(DB)의 데이터를 처리하는 소스가 들어있는 jar파일
 - Source Connector: data source에 담긴 데이터를 topic에 담는 역할(Producer)을 하는 connector
 - Sink Connector: topic에 담긴 데이터를 특정 data source로 보내는 역할(Consumer 역할)을 하는 connector
- Dead Letter Queue – Connect에서 Connector 오류를 처리하는 방법
- 단일 모드(Standalone): 하나의 Connect만 사용하는 모드
- 분산 모드(Distributed): 여러개의 Connect를 한 클러스트로 묶어서 사용하는 모드
 - 특정 Connect가 장애가 발생해도 나머지 Connect가 대신 처리하도록 함

Query-based CDC (1/2)

- Kafka Connect 서버에 REST api를 통해 Sink/Source Connector를 등록한다.
- Source Connector 예제

```
curl -i -X POST -H "Accept:application/json" \
-H "Content-Type:application/json" http://{KAFKA CONNECT URL}:8083/connectors/ \
-d '{
  "name": "jdbc-source-connect-new",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "connection.url": "jdbc:oracle:thin:@{DB URL}:{DB PORT}:{DB NAME}",
    "connection.user": "{USER NAME}",
    "connection.password": "{PASSWORD}",
    "mode": "timestamp",
    "timestamp.column.name": "{TIMESTAMP COLUMN NAME}",
    "schema.pattern" : "{SCHEMA NAME}",
    "topic.prefix" : "{TOPIC PREFIX}",
    "table.whitelist" : "{TABLE NAME}",
    "tasks.max" : "1",
    "dialect.name": "OracleDatabaseDialect",
    "db.timezone": "Asia/Seoul"
  }
}'
```

Oracle JDBC Sample

Query-based CDC (2/2)

- Sink Connector 예제

```
curl -i -X POST -H "Accept:application/json" \
-H "Content-Type:application/json" http://localhost:8083/connectors/ \
-d '{
  "name": "postgresql-jdbc-sink-connect",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "connection.url": "jdbc:postgresql://{{DB_URL}}/{{DB_NAME}}",
    "database.history.kafka.bootstrap.servers": "127.0.0.1:9092",
    "connection.user": "{USERNAME}",
    "connection.password": "{PASSWORD}",
    "auto.create": "false",
    "auto.evolve": "true",
    "delete.enabled": "false",
    "tasks.max": "1",
    "topics": "{{TABLE_NAME}}",
    "database.serverTimezone": "Asia/Seoul"
  }
}'
```

PostgreSQL JDBC Sample

Log-based CDC (1/2)

- Log-Based CDC는 각 DB 벤더사의 connector를 이용하거나 debezium의 Open Source인 debezium Connector를 사용한다.

```
curl -i -X POST -H "Accept:application/json" \
-H "Content-Type:application/json" http://[KAFKA CONNECT URL]:8083/connectors/ \
-d '{
  "name": "{CONNECTOR NAME}",
  "config": {
    "connector.class" : "io.debezium.connector.oracle.OracleConnector",
    "tasks.max" : "1",
    "database.server.name" : "{LOGICAL DB NAME}",
    "database.hostname" : "[DB URL]",
    "database.port" : "[DB PORT]",
    "database.user" : "{USERNAME}",
    "database.password" : "{PASSWORD}",
    "database.dbname" : "[DB NAME]",
    "database.server.id": "41",
    "table.include.list" : "{SCHEMA}.{TABLE NAME}",
    "database.history.kafka.bootstrap.servers": "[KAKFA BROKER URL]:9092",
    "database.history.kafka.topic": "schema-changes.rentqa",
    "include.schema.changes": "true",
    "transforms": "route",
    "transforms.route.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.route.regex": "([^.]+)\\.([^.]+)\\.([^.]+)",
    "transforms.route.replacement": "$3",
    "database.connectionTimeZone": "Asia/Seoul",
    "database.connection.adapter": "logminer"
  }
}'
```

Oracle Log기반 CDC Source Connector

Log-based CDC (2/2)

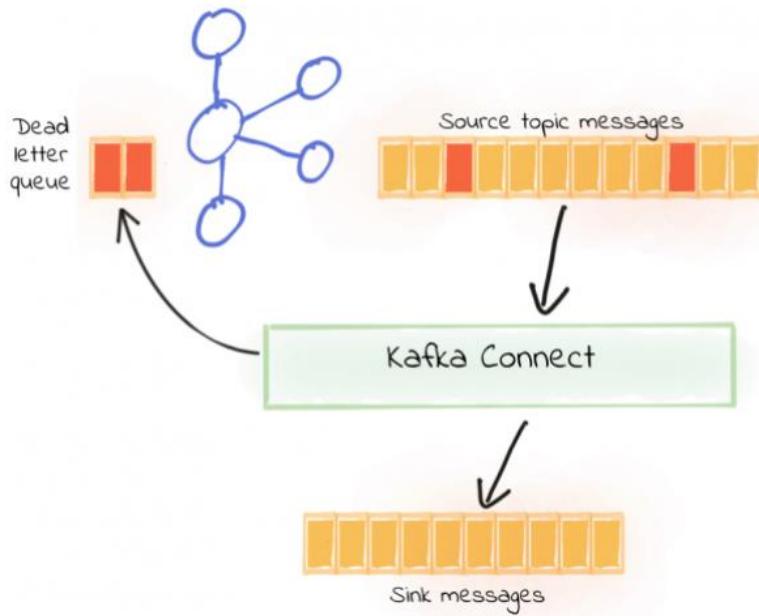
- Sink Connector 예제

```
curl -i -X POST -H "Accept:application/json" \
-H "Content-Type:application/json" http://localhost:8083/connectors/ \
-d '{
  "name": "jdbc-sink-connect-cluster",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "connection.url": "jdbc:oracle:thin:@{DB_URL}/{DB_NAME}",
    "database.history.kafka.bootstrap.servers": "127.0.0.1:9092",
    "connection.user": "{USERNAME}",
    "connection.password": "{PASSWORD}",
    "auto.create": "true",
    "auto.evolve": "false",
    "delete.enabled": "true",
    "tasks.max": "1",
    "topics": "{TARGET TABLE NAME}",
    "pk.fields": "{PK COLUMN NAME}",
    "pk.mode": "record_key",
    "insert.mode": "upsert",
    "transforms": "unwrap", "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.unwrap.drop.tombstones": false,
    "database.serverTimezone": "Asia/Seoul"
  }
}'
```

Oracle Log기반 CDC Sink Connector

Kafka Connect DLQ(Dead Letter Queue)

- Kafka Connect 설정



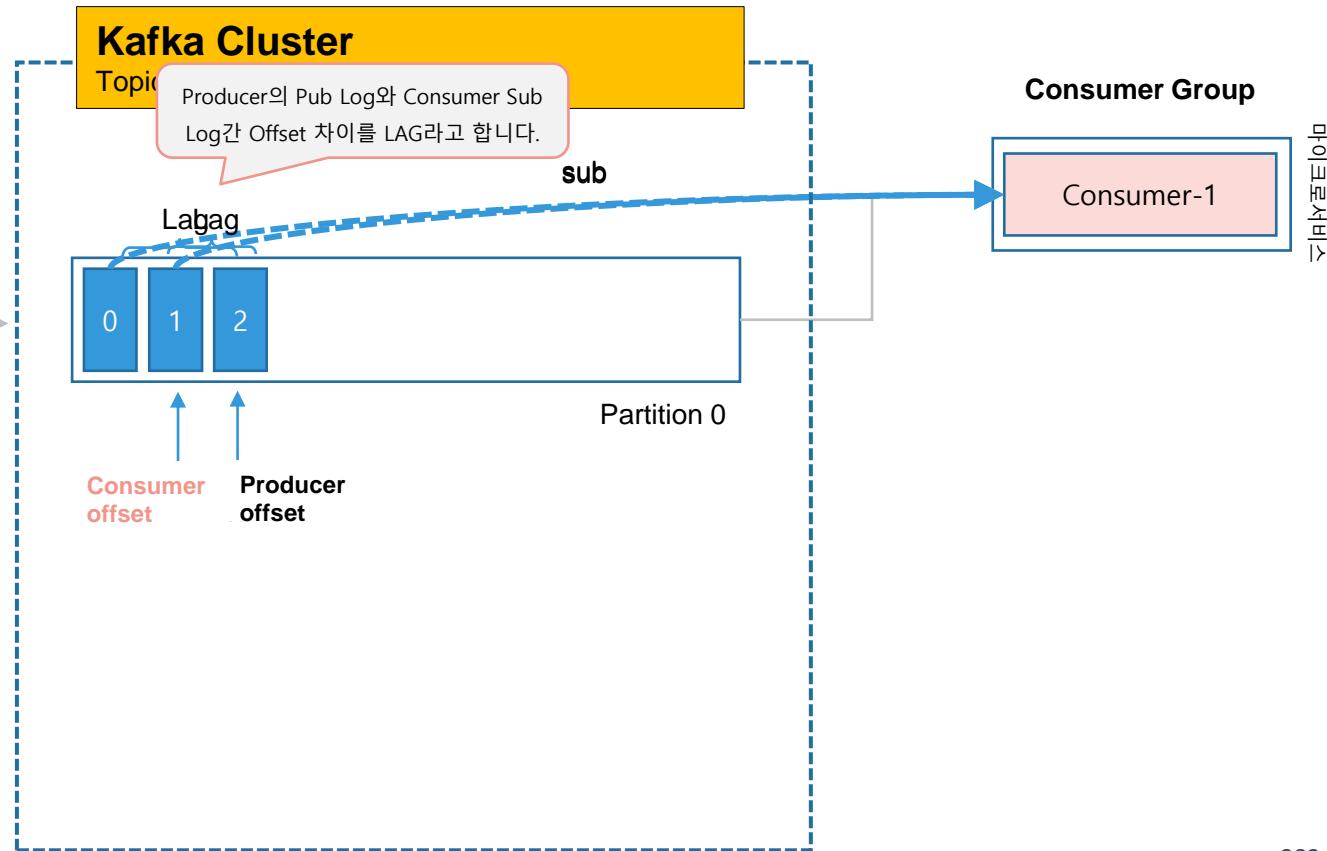
실행 명령

```
errors.tolerance = all  
errors.deadletterqueue.topic.name = {토픽}  
  
// 유효하지 않다고 판단된 레코드의 메타데이터 header에 저장  
errors.deadletterqueue.context.headers.enable = true  
  
// 로그 파일에 유효하지 않는 레코드 기록  
errors.log.enable = true  
errors.log.include.messages = true
```

Message Flow in Kafka

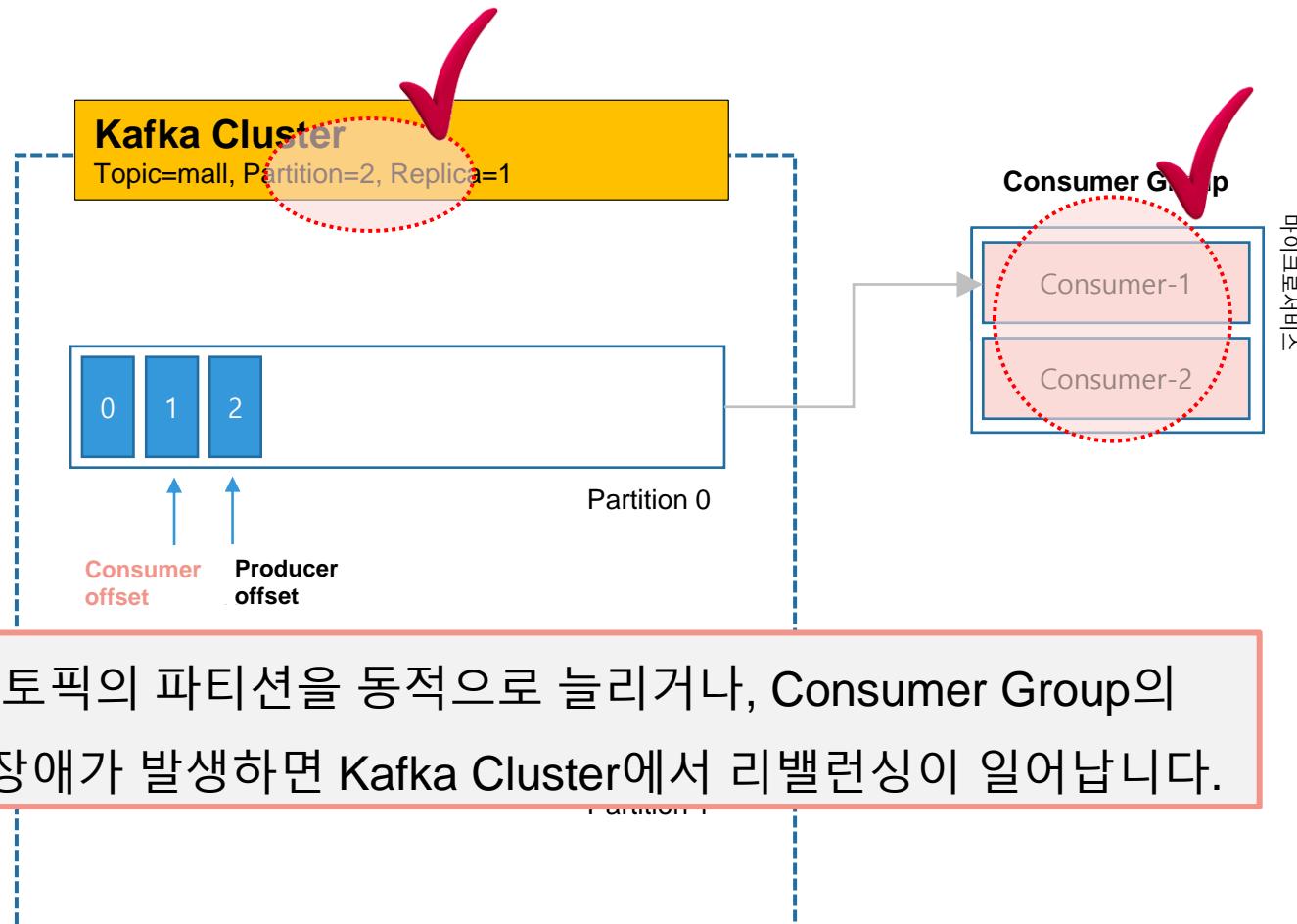
Kafka Basic

마이크로서비스 아키텍처



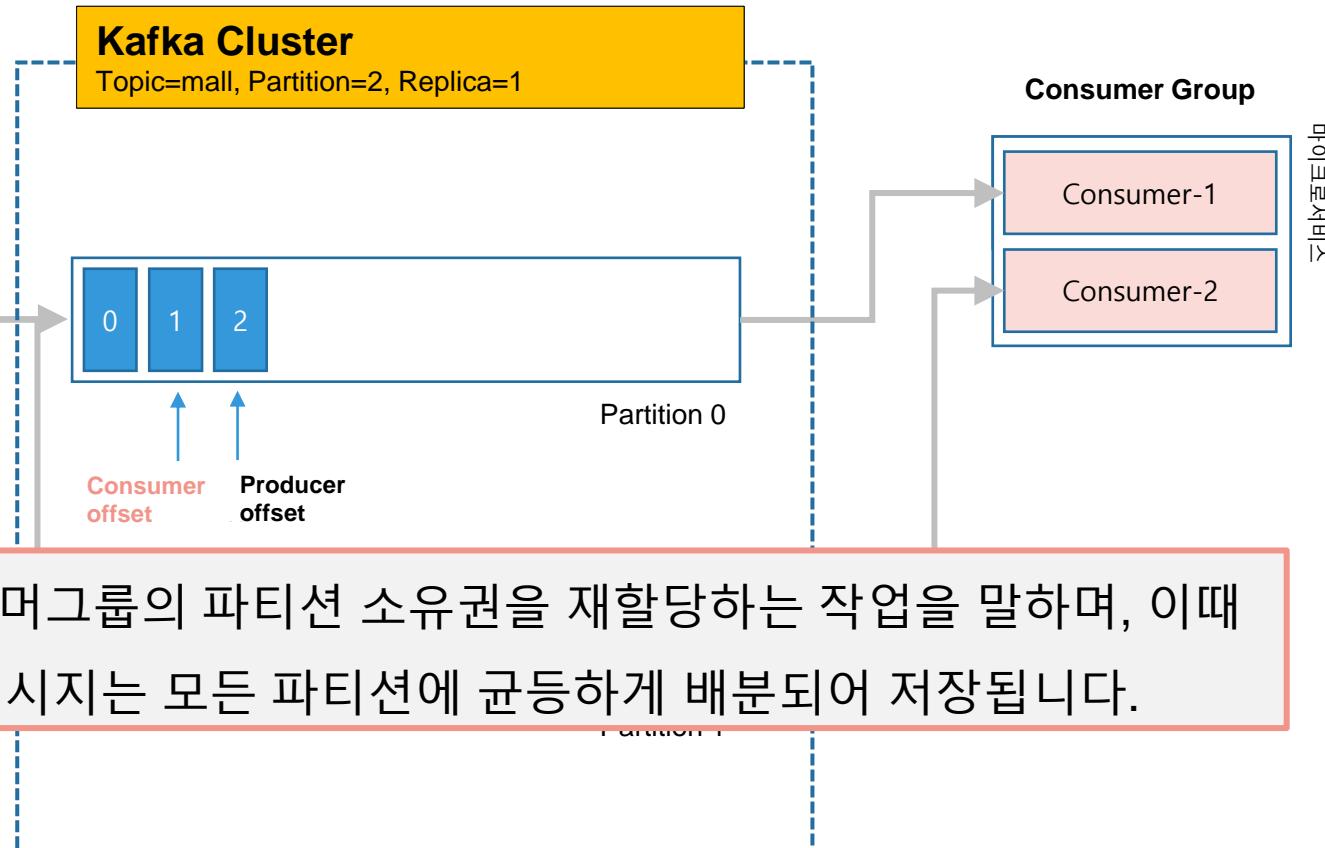
Rebalancing

마이크로서비스
설계 원칙

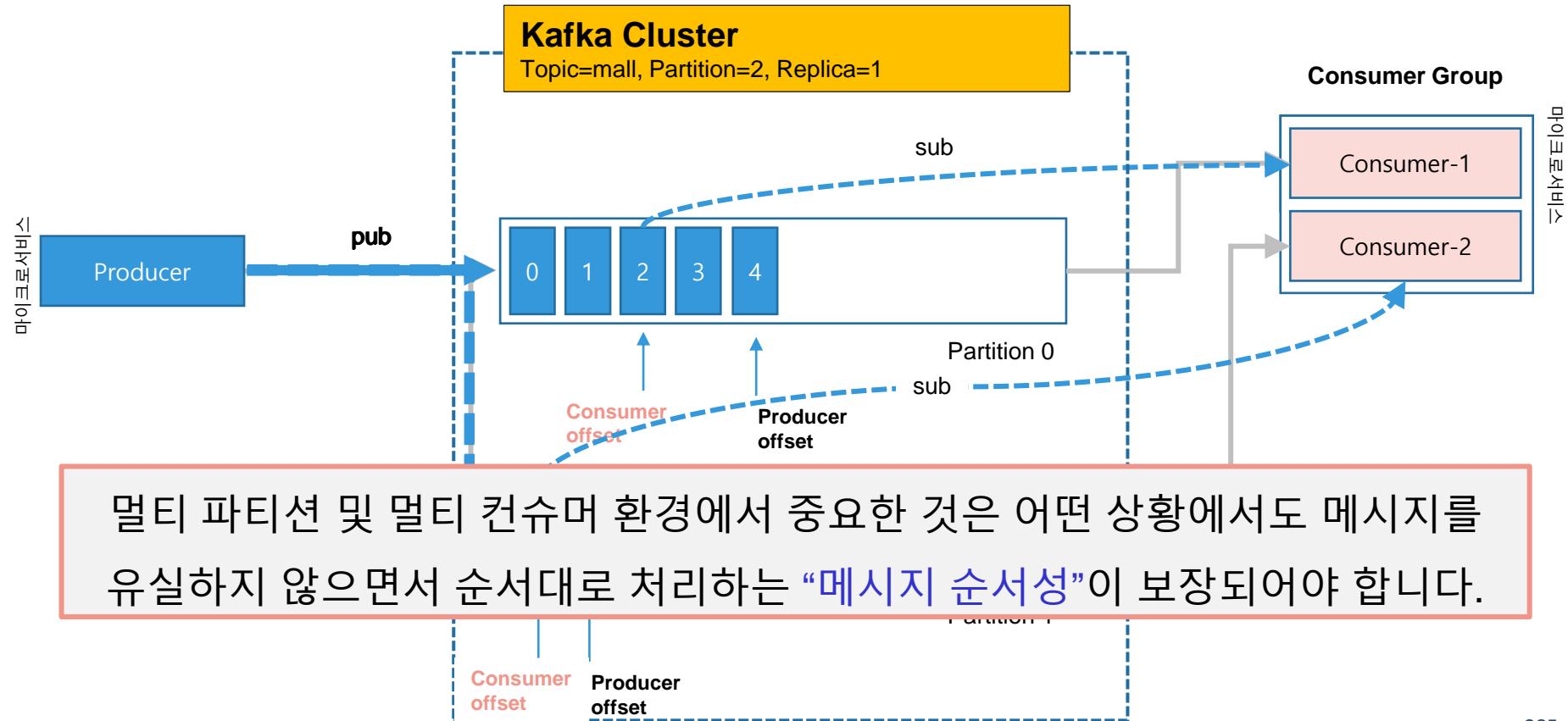


Rebalancing

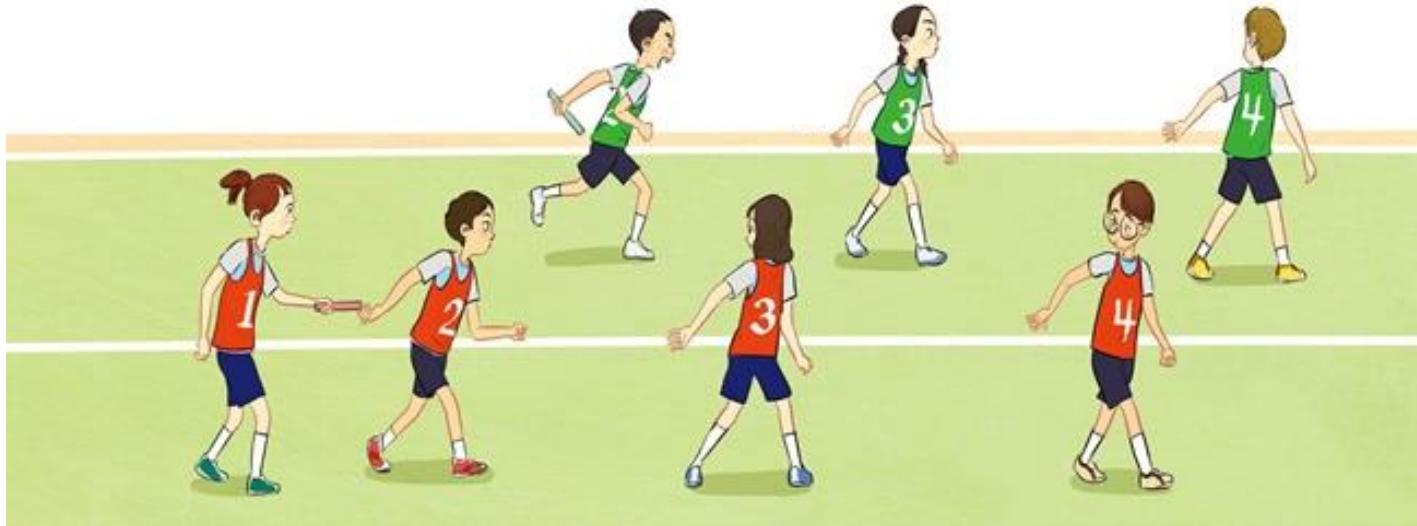
마이크로서비스
설계 원칙



Multi-Partition



Message Orderliness



이는 마치, 400계주 이어달리기처럼 카프카 메시지들은 지정된 트랙(파티션)을
유지하며, 메시지간 순서성 보장을 위한 바톤(Key)을 주고 받아야 합니다.

Message Orderliness

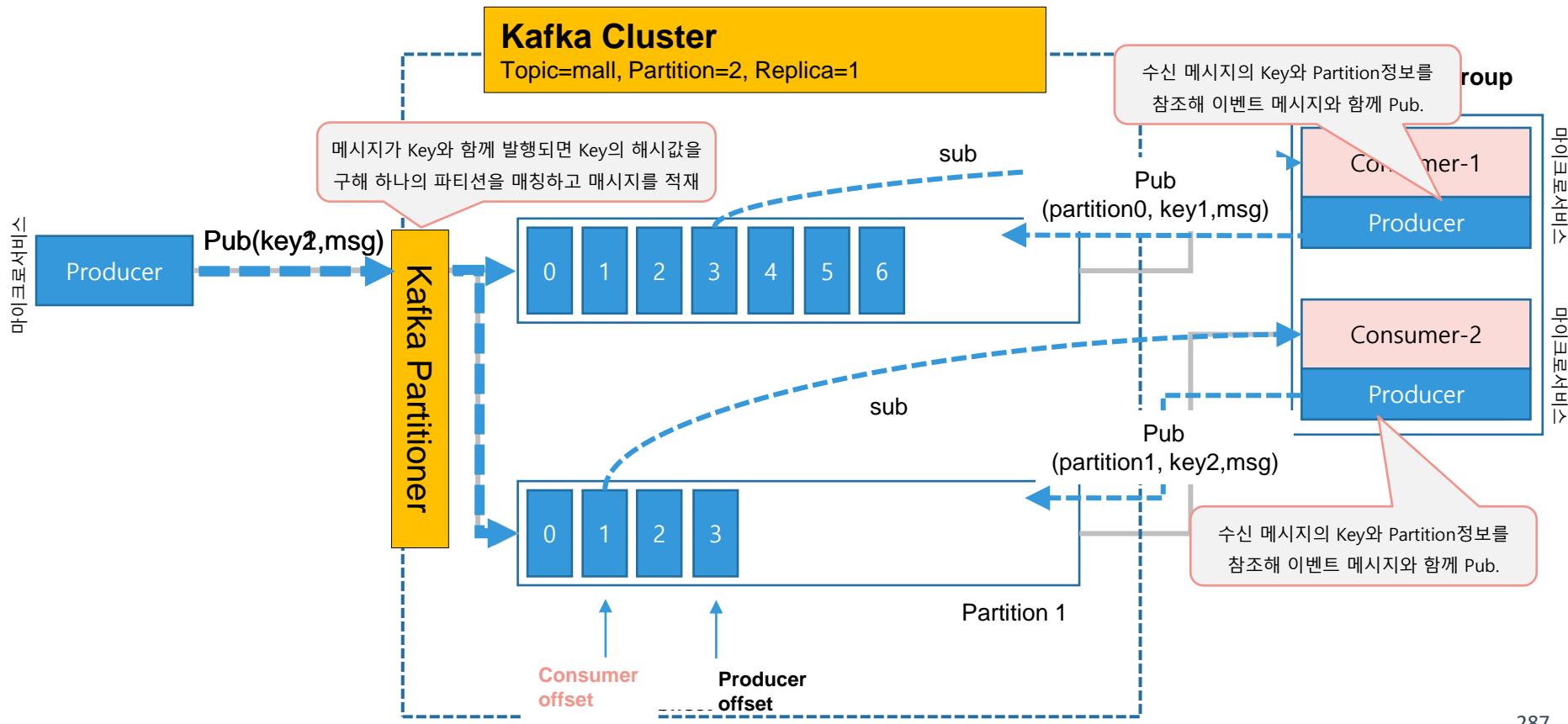


Table of content

Microservice and
Event-storming-Based
DevOps Project

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

“

DevOps

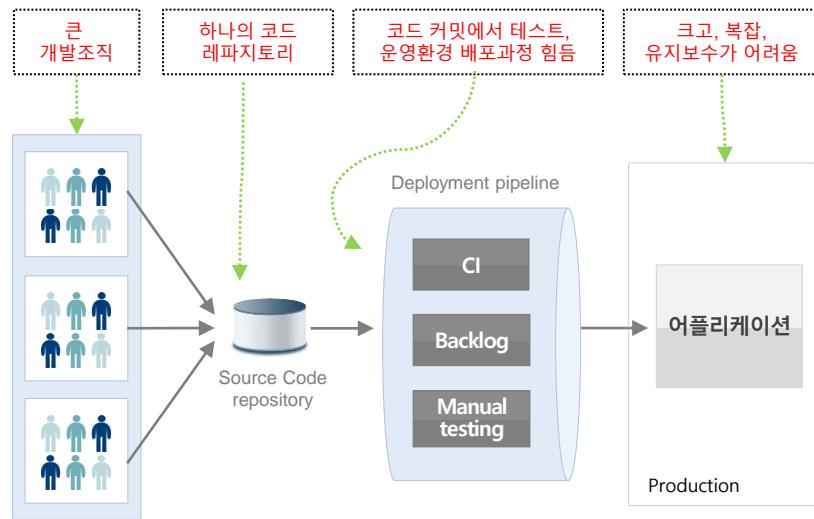
- DevOps Process and Tools
- Deploy Strategies
- Containers and Orchestrators
- Google Cloud Platform

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

Process Change : 열차말고 택시를 타라!

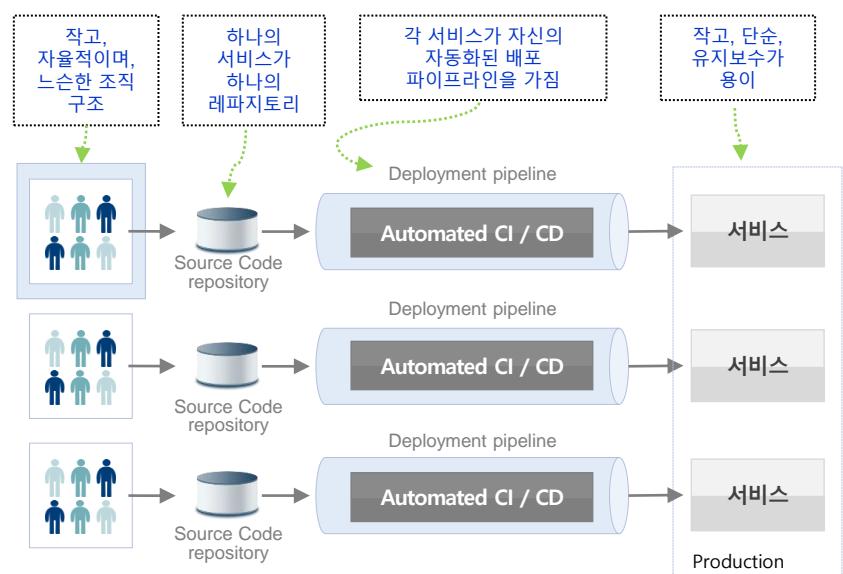
모노리식 개발 및 운영환경

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



マイ크로서비스 개발 및 운영환경

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

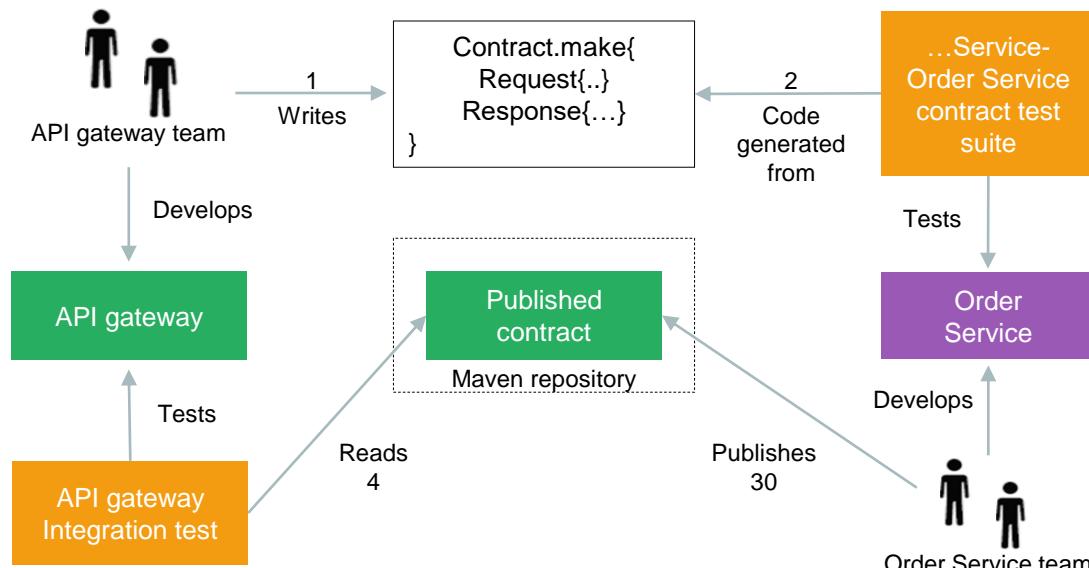


Process Change : Consumer-driven Test

MSA 서비스의 테스트

컨트랙트 테스트는 서비스 수요자가 주도하여 테스트를 작성한다.

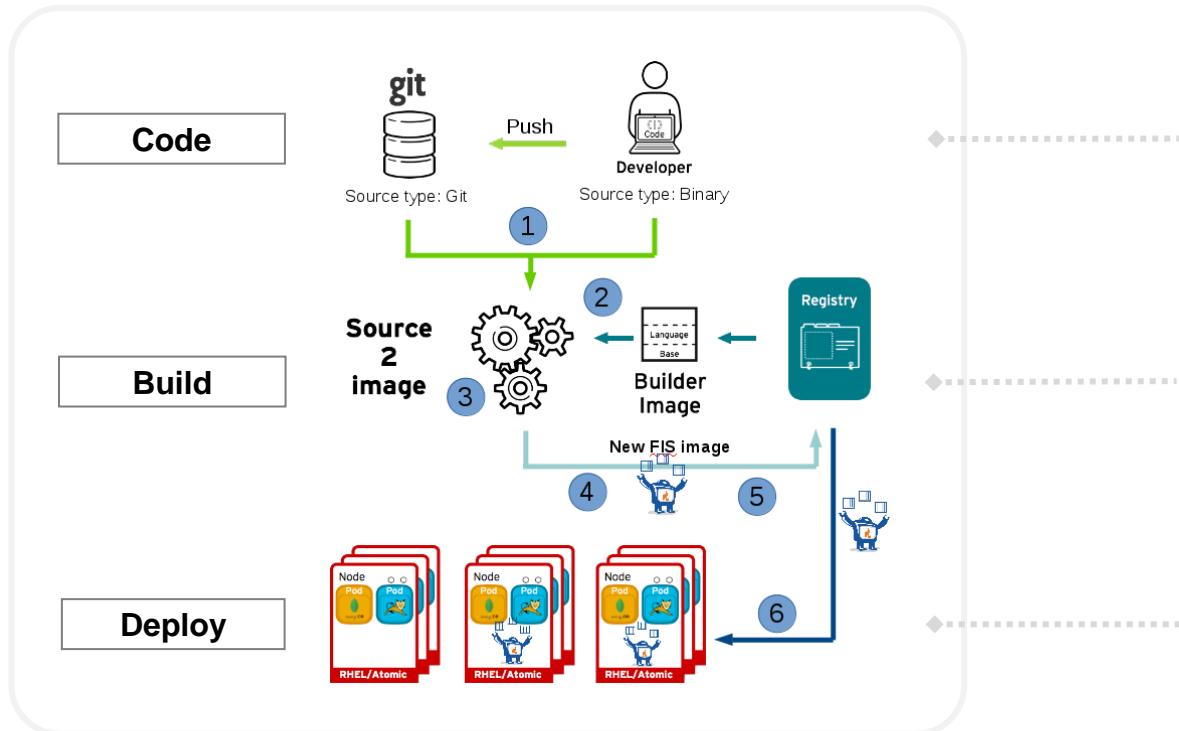
수요자가 테스트를 작성하여 제공자의 레포지토리에 Pull-Request 하면, 제공자가 이를 Accept 한후에 제공자측에서 테스트가 생성되어 테스트가 벌어진다.



특징

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

DevOps toolchain



Supporting Continuous Delivery

Facebook

2. 배포 주기

- 매일 마이너 업데이트
- 메이저 업데이트 (매주 화요일 오후)



3. Deployment Pipeline

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

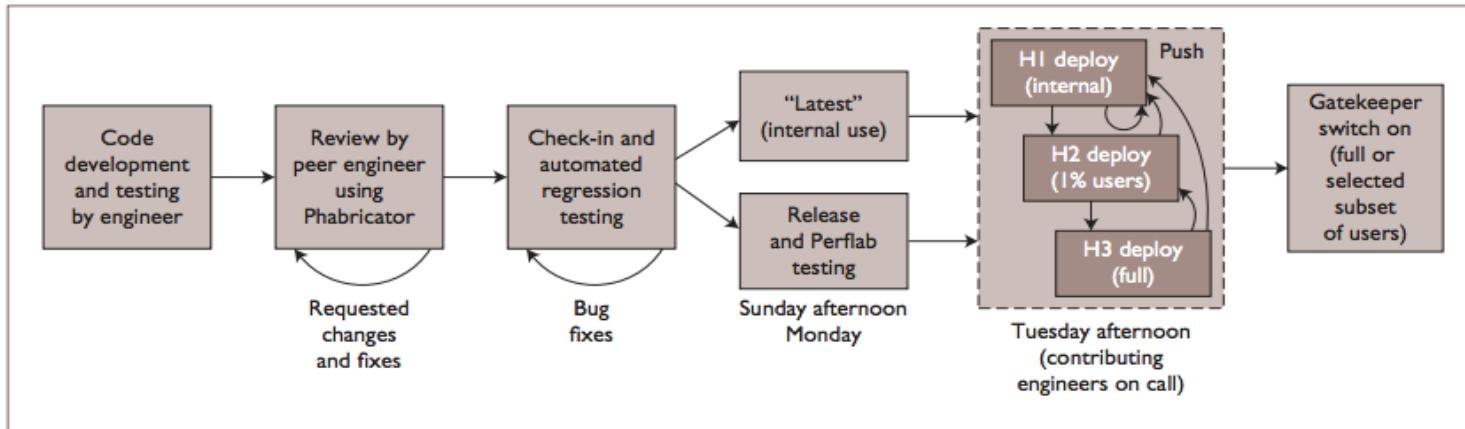


Figure 8. The Facebook deployment pipeline. Multiple controls exist over new code.

Supporting Continuous Delivery

Netflix

4. Canary를 통해서 확신 갖기

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



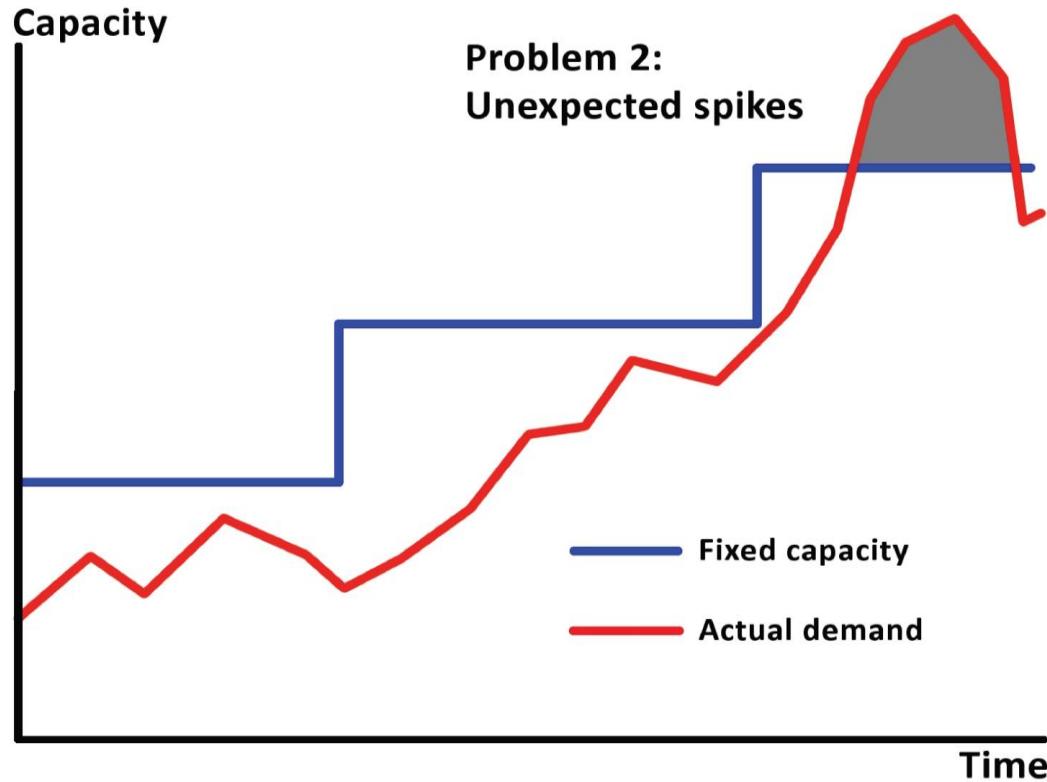
Canary Score

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



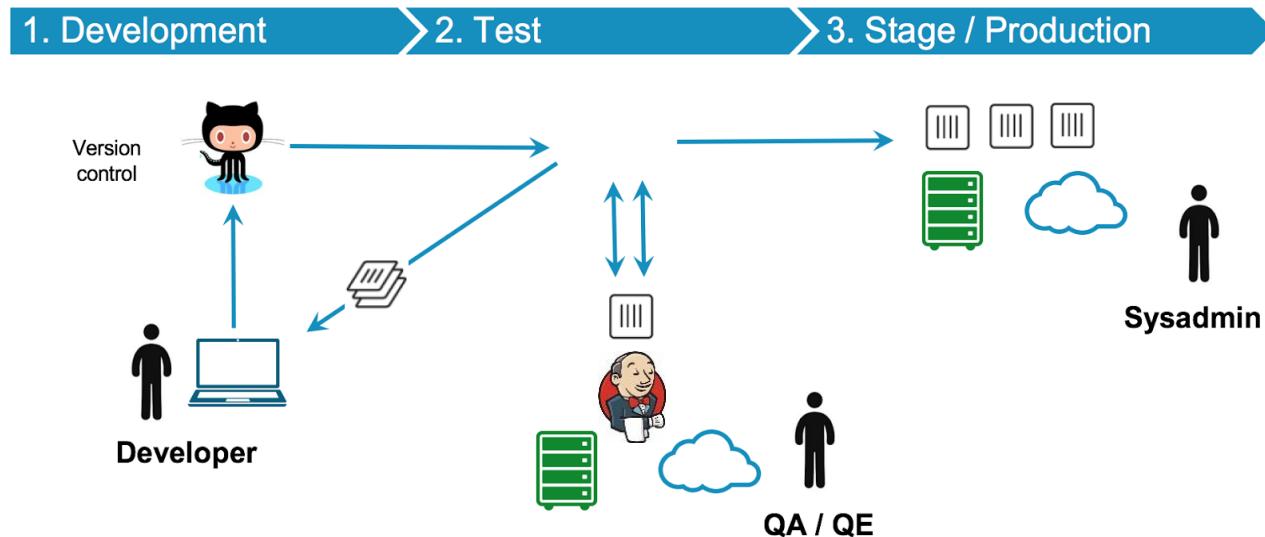
출처: <http://techblog.netflix.com/2013/08/deploying-netflix-api.html>

Managing Scalability



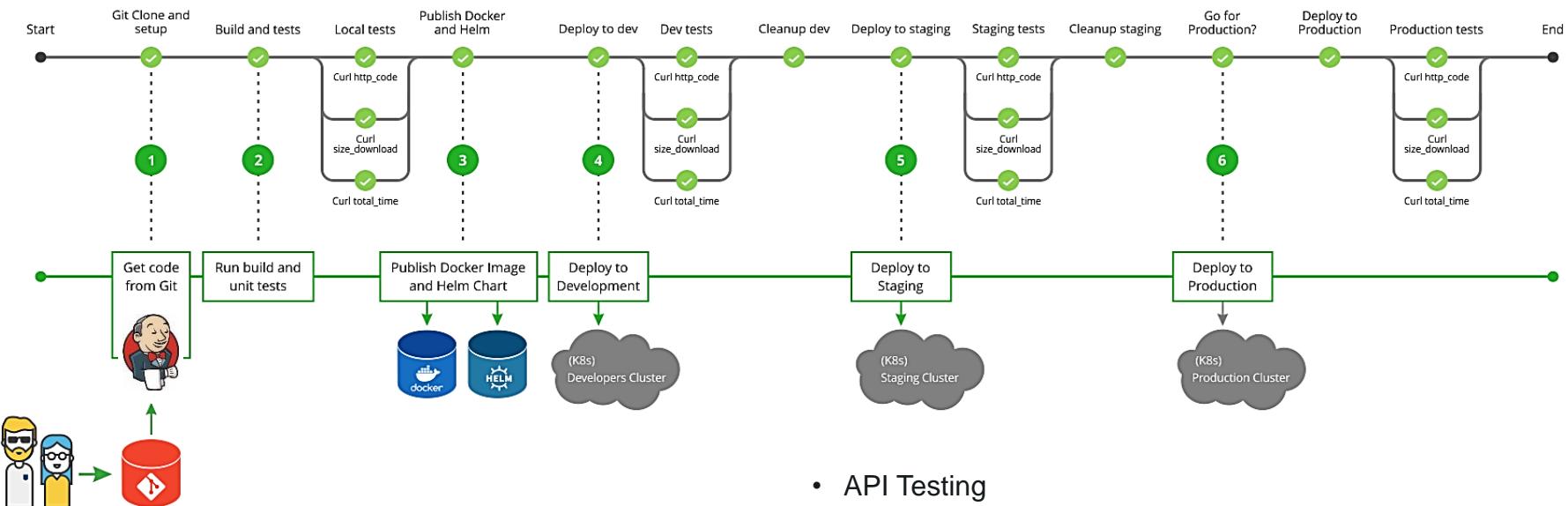
CI/CD Tools – conventional CI/CD

CI /CD Workflow



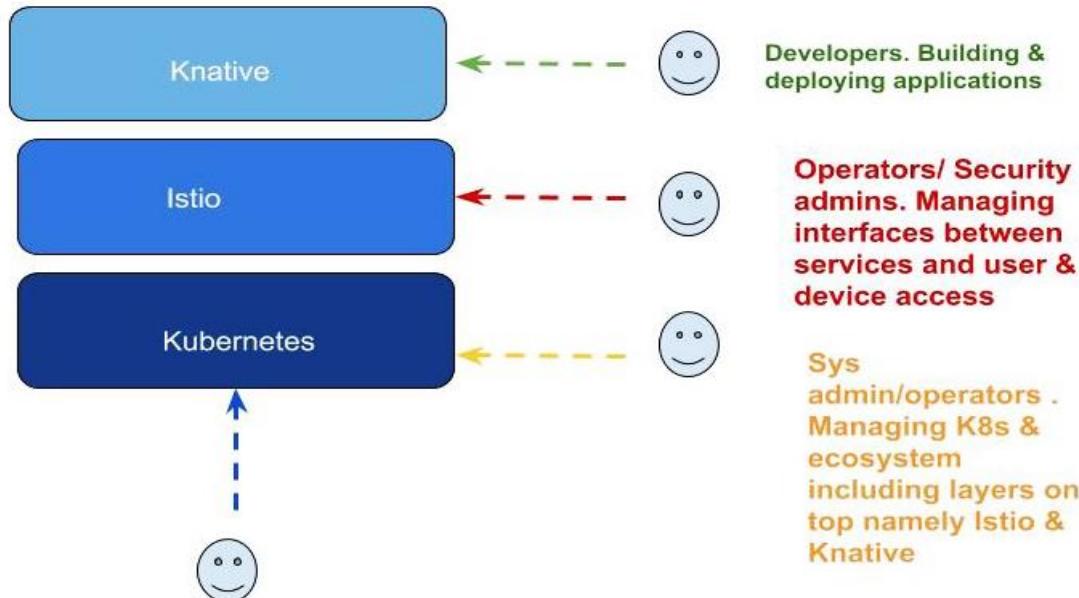
- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

CI/CD Tools – Container-based



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

CI/CD Tools - Serverless



Platform providers or K8s
on-premises(sysadmin/operators)
Making K8s work and interconnect

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

배포 전략별 비교

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

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

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



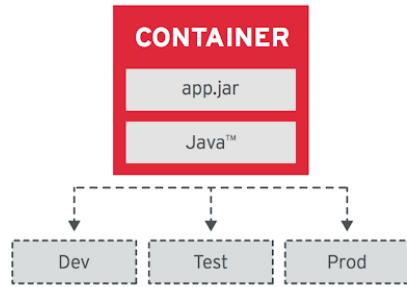
Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■ □ □	■ ■ ■	■ ■ ■	□ □ □
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■ □ □	■ ■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

DevOps Platforms

Container	Workload Distribution Engine (Container Orchestrator)	PaaS
<ul style="list-style-type: none">Docker	<ul style="list-style-type: none">KubernetesDocker SWARM(toy)Mesos Marathon(DCOS)Cloud Foundry	<ul style="list-style-type: none">Google Cloud PlatformRedhat Open ShiftAmazon EKSIBM Bluemix
<ul style="list-style-type: none">Warden(Garden)		<ul style="list-style-type: none">HerokuGE's Predix
		<ul style="list-style-type: none">Pivotal Web Services
<ul style="list-style-type: none">Hypervisor	<ul style="list-style-type: none">CF version 1Engine yard....	<ul style="list-style-type: none">Amazon Beanstalk

Container-based Application Design

Image Immutability Principle



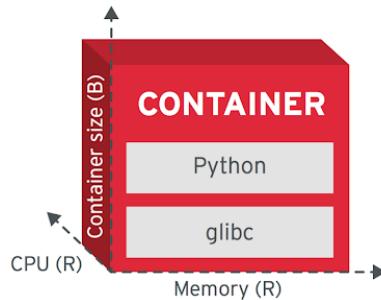
High Observability Principle



Process Disposability Principle



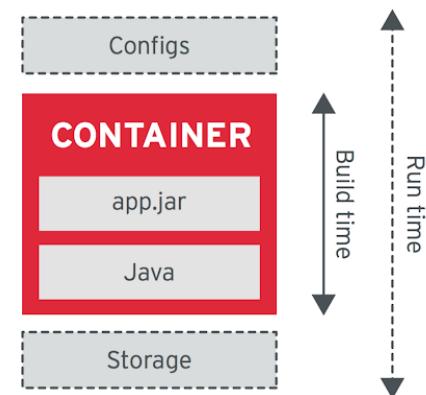
Runtime Confinement Principle



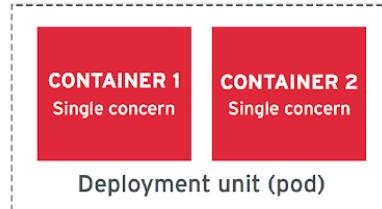
Lifecycle Conformance Principle



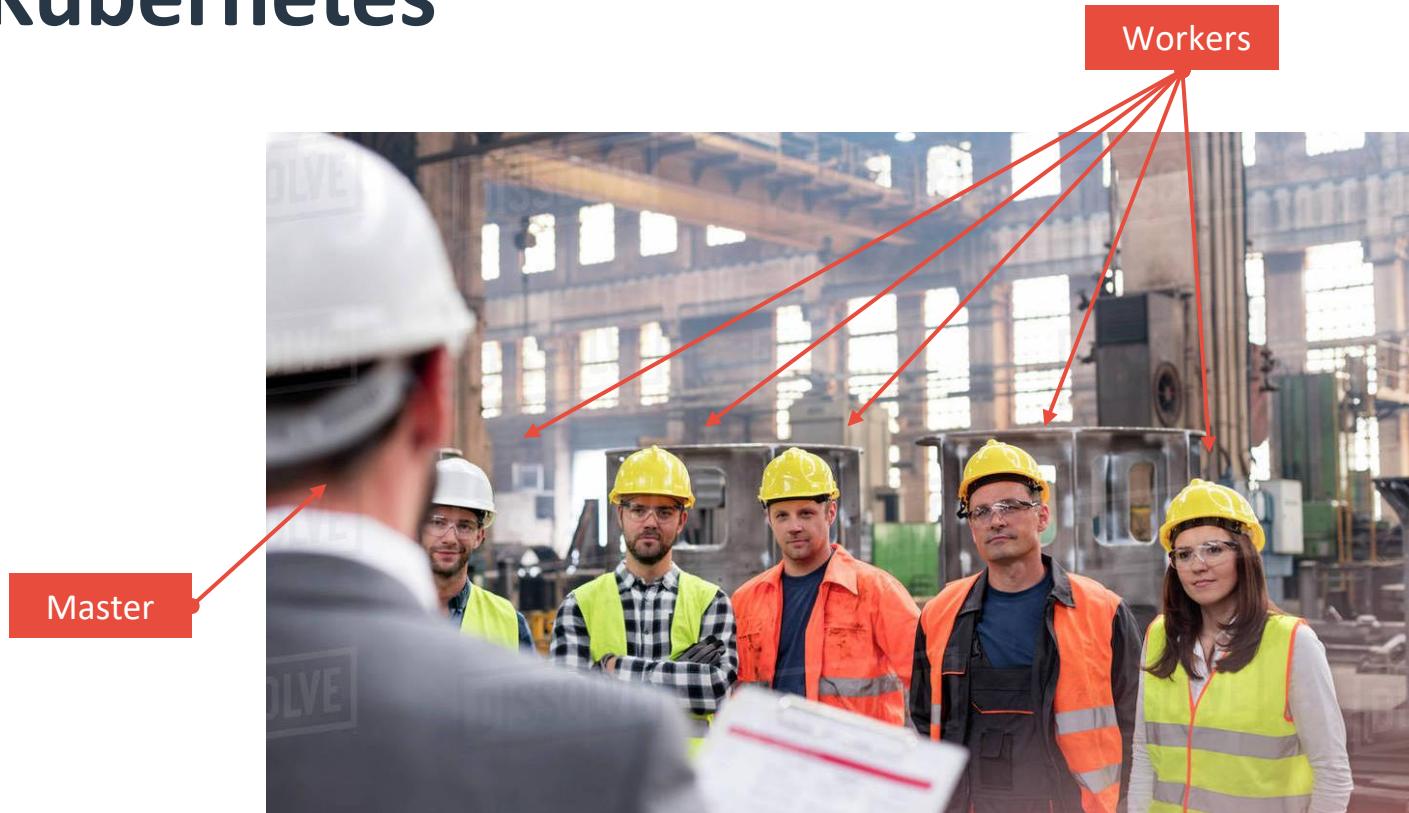
Self-Containment Principle



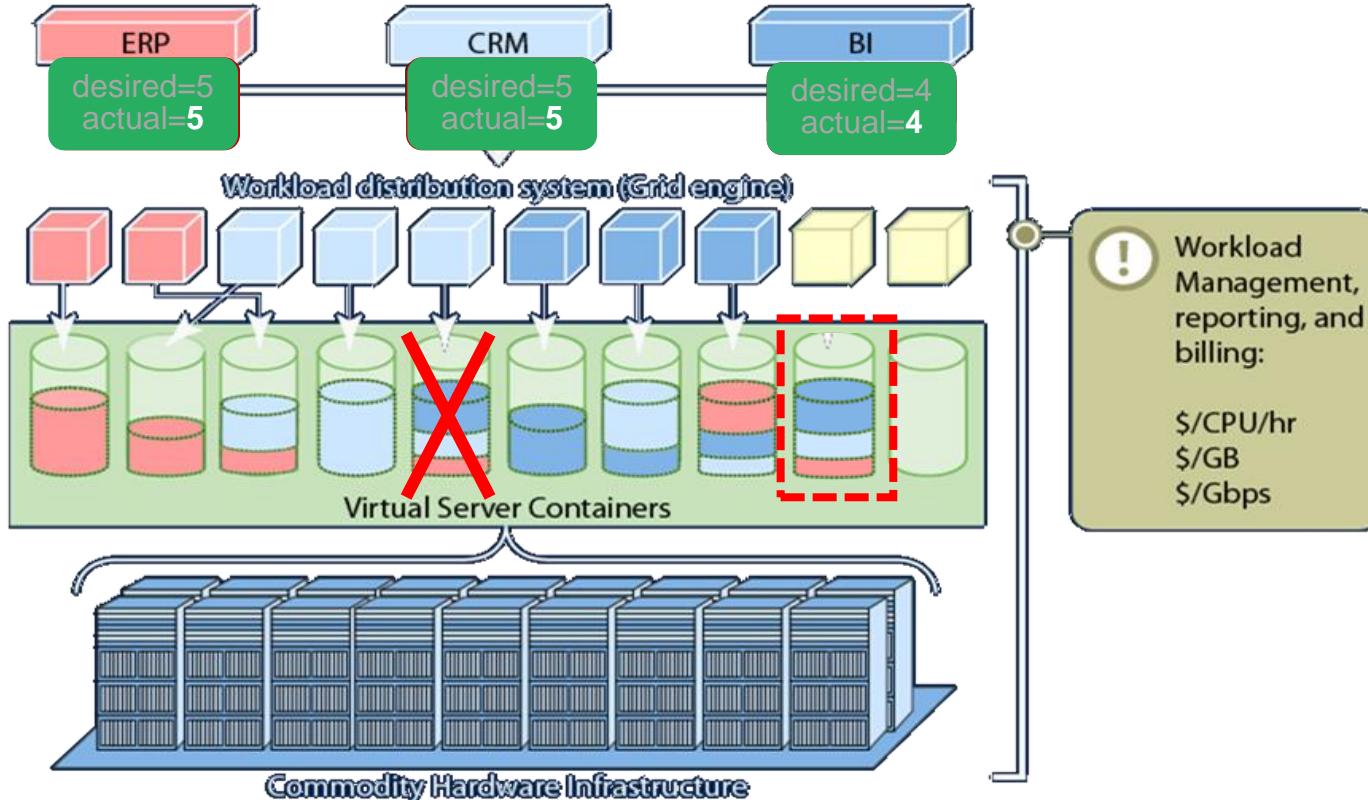
Single Concern Principle



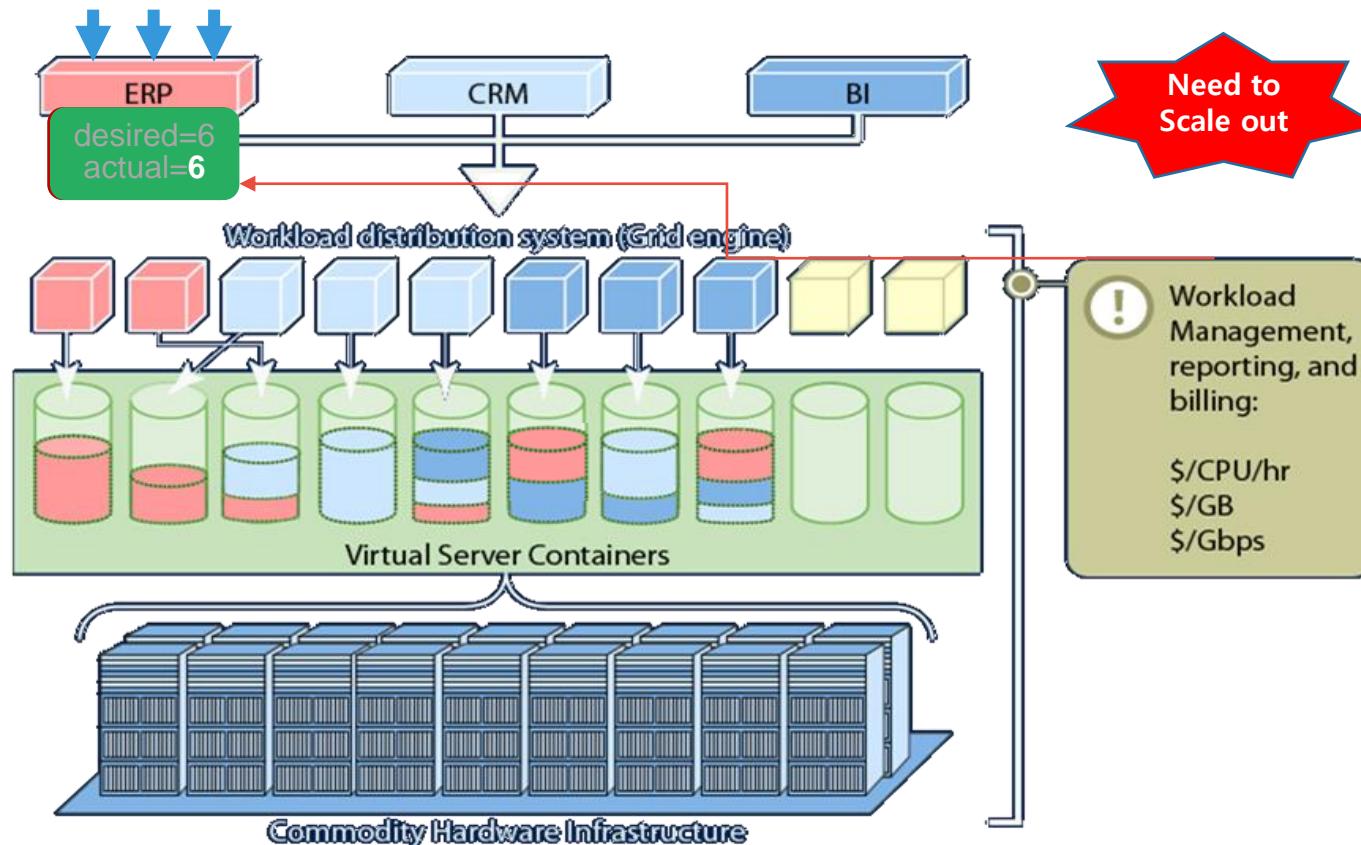
Kubernetes



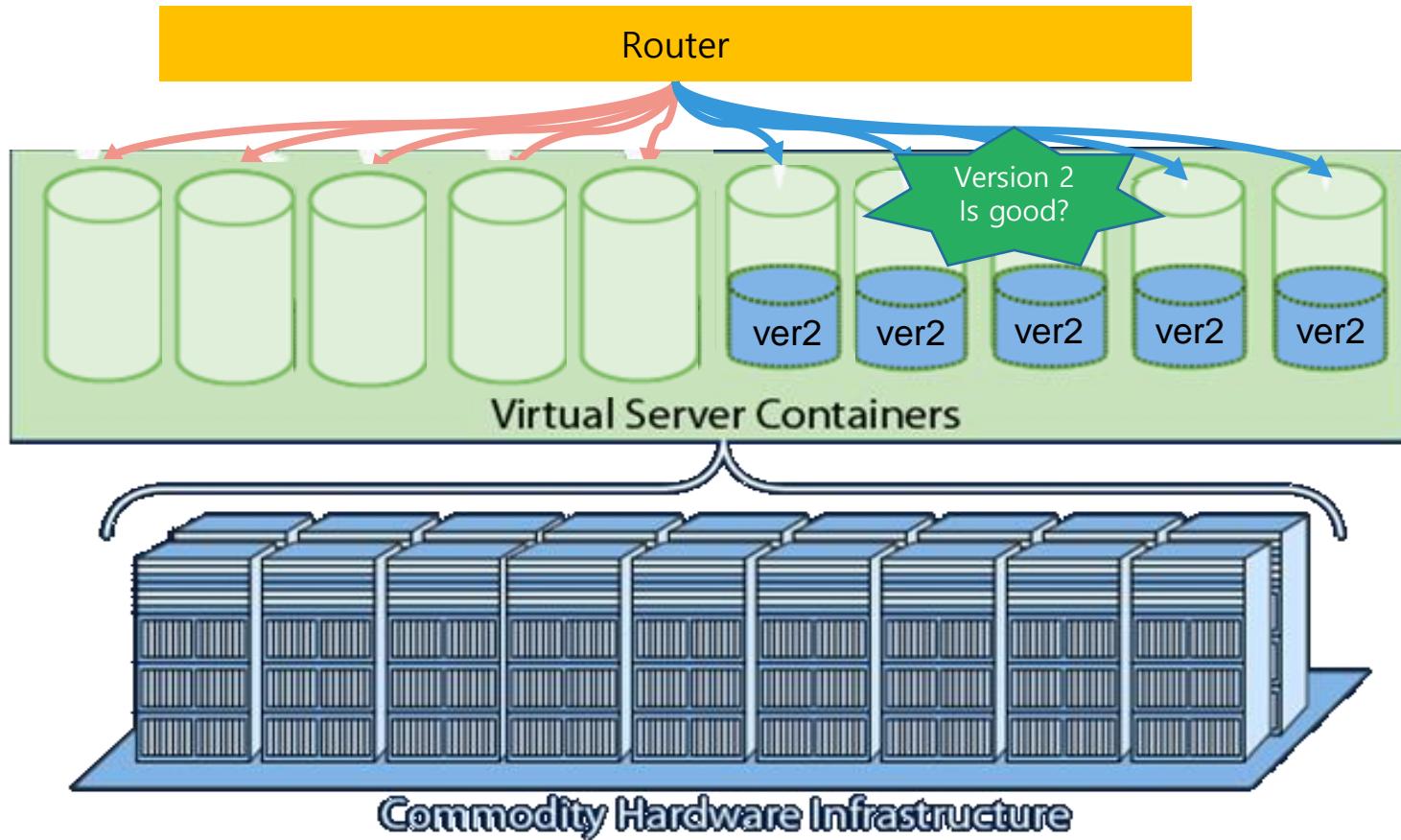
Container Orchestration – Self Healing Mechanism



Container Orchestration – Scale out



Container Orchestrator – Zero Down Time Deploy



Kubernetes Object Model

http://serviceld:8080

Micro-Service

Service
(name: foo)

http://external_ip:8080

Load Balancer from
cloud provider

Redirects

Deployment

- Dynamic Service Binding

- e.g. <http://foo:8080>

- Type :

- LoadBalancer e.g. Ingress (API GW) or front-end
- ClusterIP e.g. 내부 마이크로 서비스들

- Zero-down time deployment

- (Kubernetes default is rolling-update)
e.g. Blue / Green

Creates / Manages

ReplicaSet
(Blue)

ReplicaSet
(Green)

- Keep replica count as desired
(replicas=2)

Pod

Container
(Main)

Container
(Side-car)

Pod

Container

Container

Pod

Container

Container

Pod

Container

Container

- Service Hosting

Declaration based configuration

> Desired state



> my-app.yml

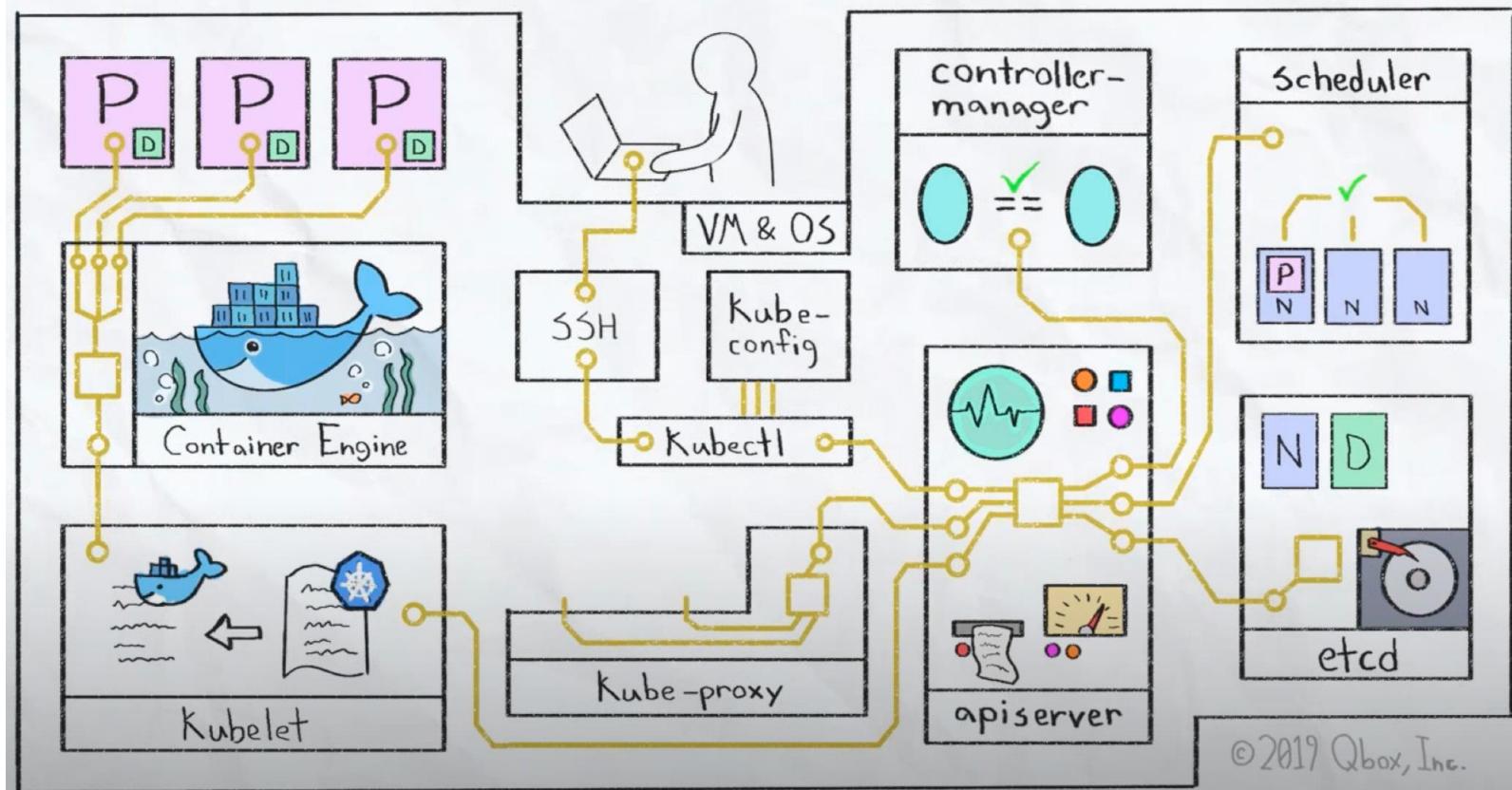
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
...
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
          - containerPort: 80
---
apiVersion: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

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

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

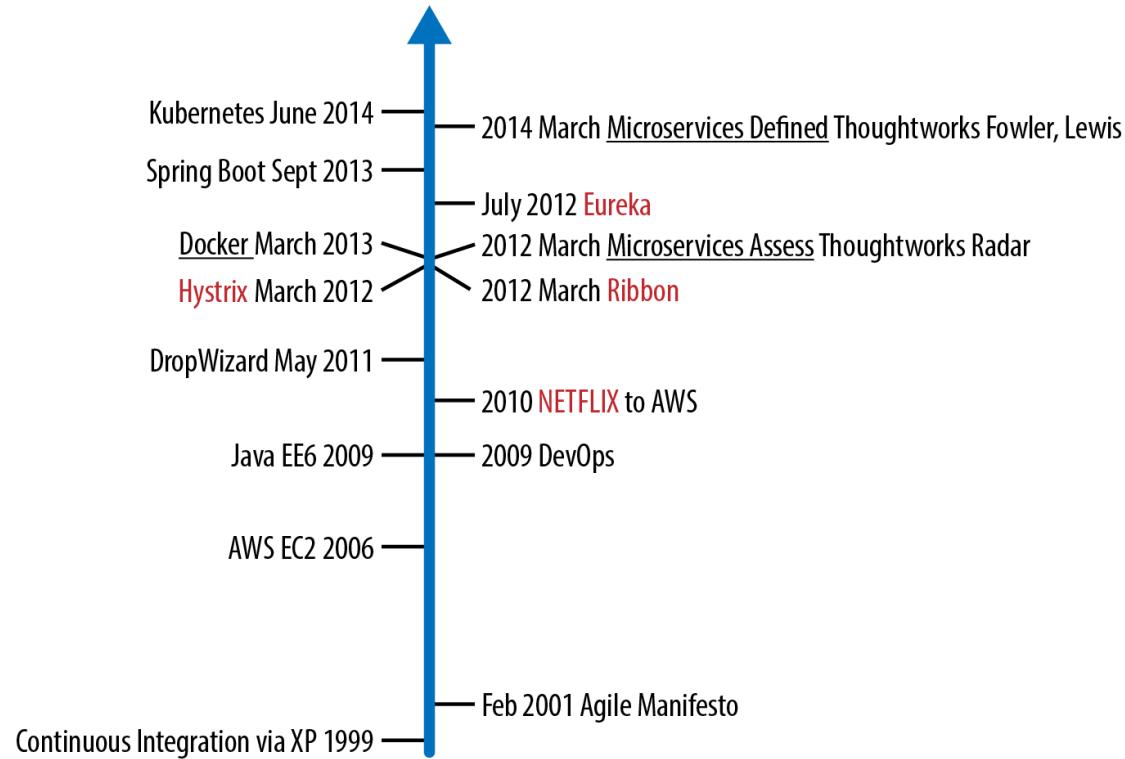
```
apiVersion: apps/v1
kind: Service
metadata:
  name: mongo-service
  labels:
    app: mongo
```

Anatomy of Kubernetes



Service Mesh

Microservices timeline



K8S vs Spring Cloud vs Istio

Capabilities	Spring Cloud	Istio	Kubernetes	Spring Cloud with Kubernetes & Istio on IaaS+
DevOps Experience				Self service, multi-environment capabilities
Auto Scaling & Self Healing				Pod/Cluster Autoscaler, HealthIndicator, Scheduler, Pool Ejection
Deployment & Scheduling				Deployment strategy, DarkLaunch, AB, Canary, Scheduler Strategy
Resilience & Fault Tolerance				HealthIndicator, Hystrix, HealthCheck, Process Check, Circuit Breaker/Timeout/Retry
Distributed Tracing				Zipkin, Jaeger
Centralized Metrics				Heapster, Prometheus, Grafana
Centralized Logging				EFK
API Gateway				Zuul, Traffic Control, Egress
Load Balancing				Ribbon, Service, Envoy
Chaos Engineering				Chaos Monkey for Spring Boot, Chaos Toolkit Kubernetes, Envoy
Service Discovery				Service
Configuration Management				Externalized Configuration, ConfigMaps, Secrets
Application Packaging				Spring Boot Maven/Gradle plugin
Job Management				Spring Batch, Scheduled Jobs
Process Isolation				Docker, Pods, Envoy
Environment Management				Namespaces, Authorization
Resource Management				CPU and Memory Limits, Namespace resource quotas
Operating System		IaaS+		Ubuntu, Atomic, ...
Virtualization				VMWare, Openstack, ...
Hardware, Storage, Networking				AWS, GCP, Azure, ...

Advanced Routing & Deployment Strategy

- Ingress / Egress Gateway
- Canary Deploy
 - 특정 유저의 신상, 지역, 권한, 접근 단말에 따른 다른 버전의 노출
- AB Testing / Shadow Deploy (Dark Launch)
 - 신규 버전의 오류 노출 없는 실질적 테스트

Advanced Resilience

- Retry (Kubernetes X, Spring Cloud O)
 - 서비스간 호출의 실패에 대한 재시도
- Circuit Breaking (Spring Cloud O) / Rate Limiting
 - 인스턴스의 보호
 - 전체 서비스 장애 차단
- Pool Ejection (Spring Cloud / Eureka)
 - 죽은 인스턴스의 제외
- Circuit Breaking + Pool Ejection + Retry = High Resilience

Advanced Security

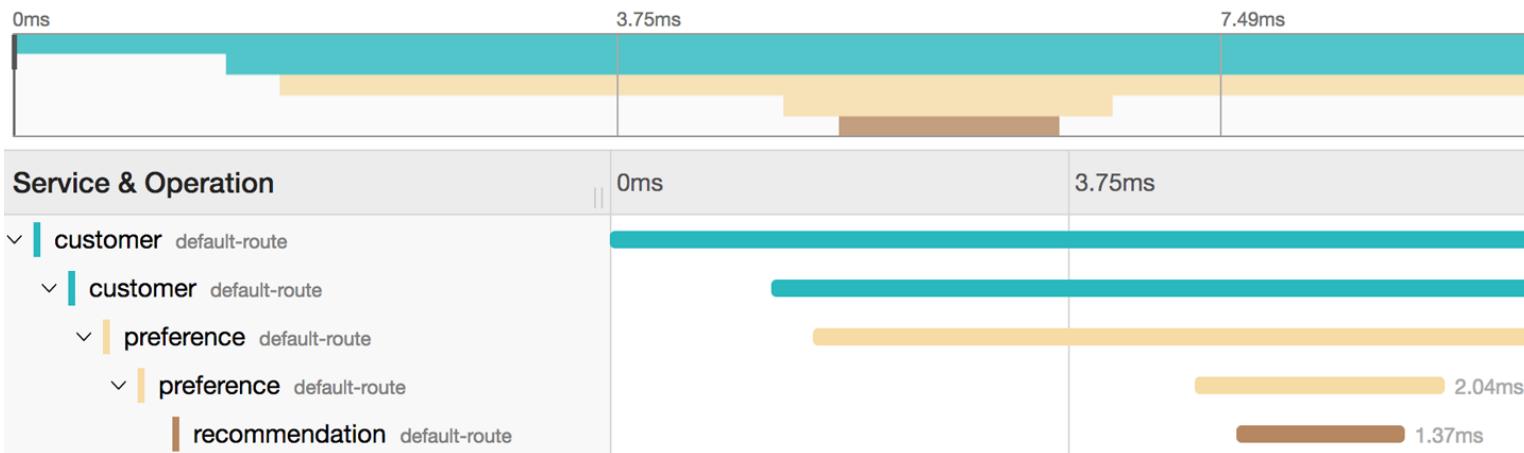
- TLS based Inter-Mi-services Communication
 - By Auth (신규모듈)
- Whitelist and Blacklist

Advanced Observability

- Distributed Tracing and Measure
 - 서비스간 호출의 내용 기록

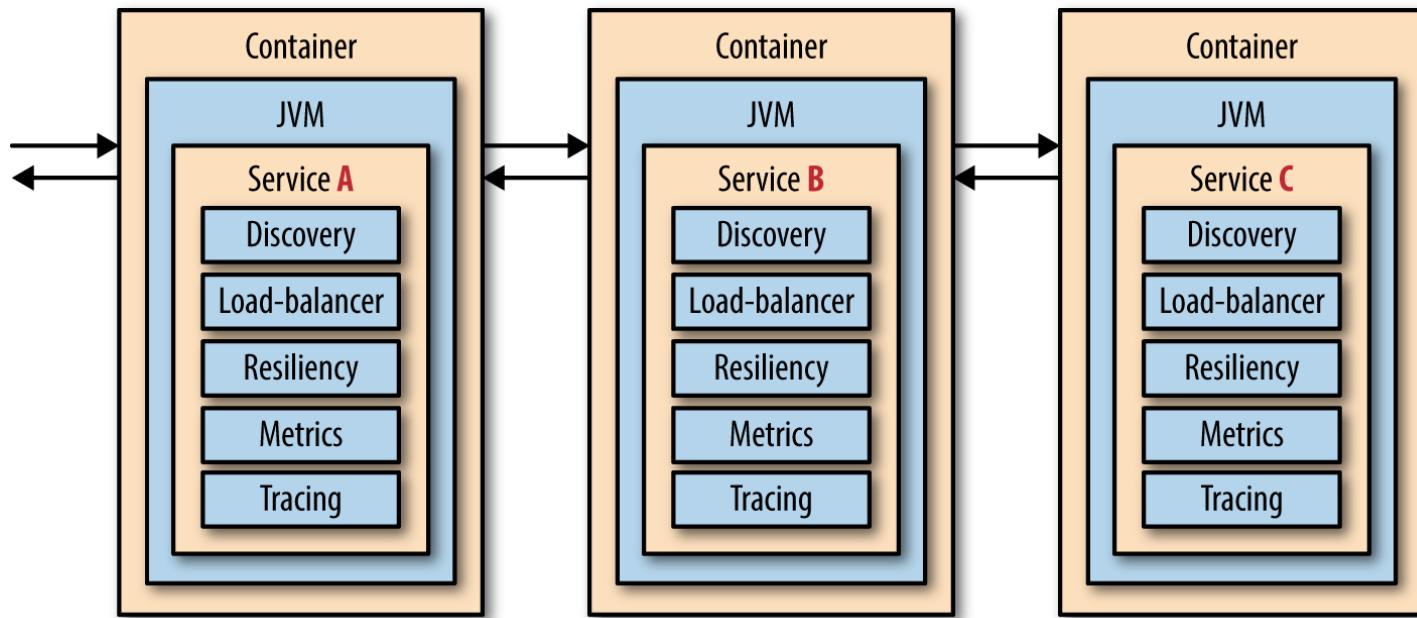
▼ customer: default-route

Trace Start: February 28, 2018 7:46 PM | Duration: 14.99ms | Services: 3 | Depth: 5 | Total Spans: 5

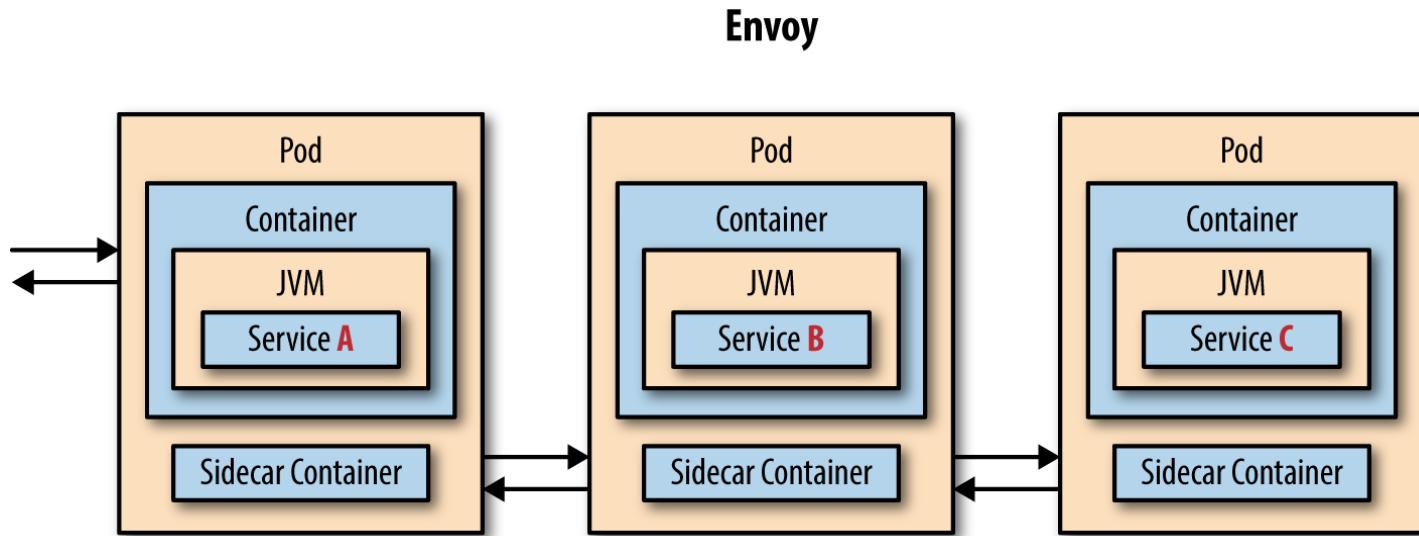


Spring Cloud + Netflix

Before Istio



After Istio (on K8S)



좋은점:

1. L7 레이어를 사용, 성능이 높음
2. Code 변경 없이 Cross-cutting 이슈를 다루어줌
3. Polyglot 다양한 언어에 무관하게 적용 가능
4. Main 서비스의 재배포 없이 Sidecar 를 관리 가능함

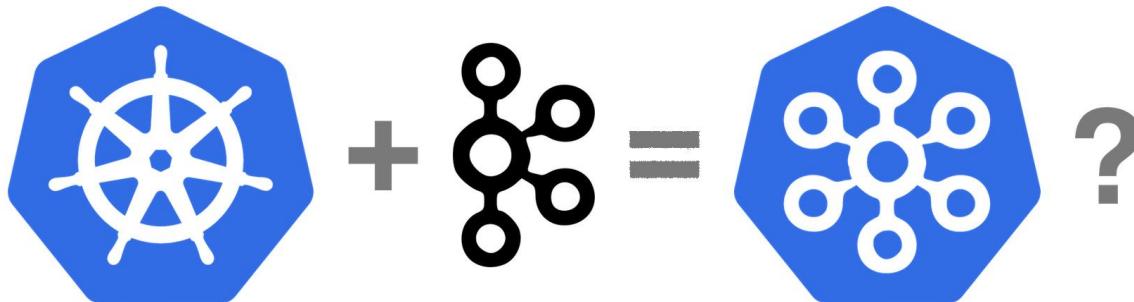
MSA 구현 기술들

- 1세대 MSA (Spring Cloud / Netflix OSS) 코드수정 ✓
- 2세대 Service Mesh (Kubernetes / Istio) 코드수정 X
- 3세대 Event Driven MSA (Serverless / Kafka) 간섭최소화

MSA 구현 기술들

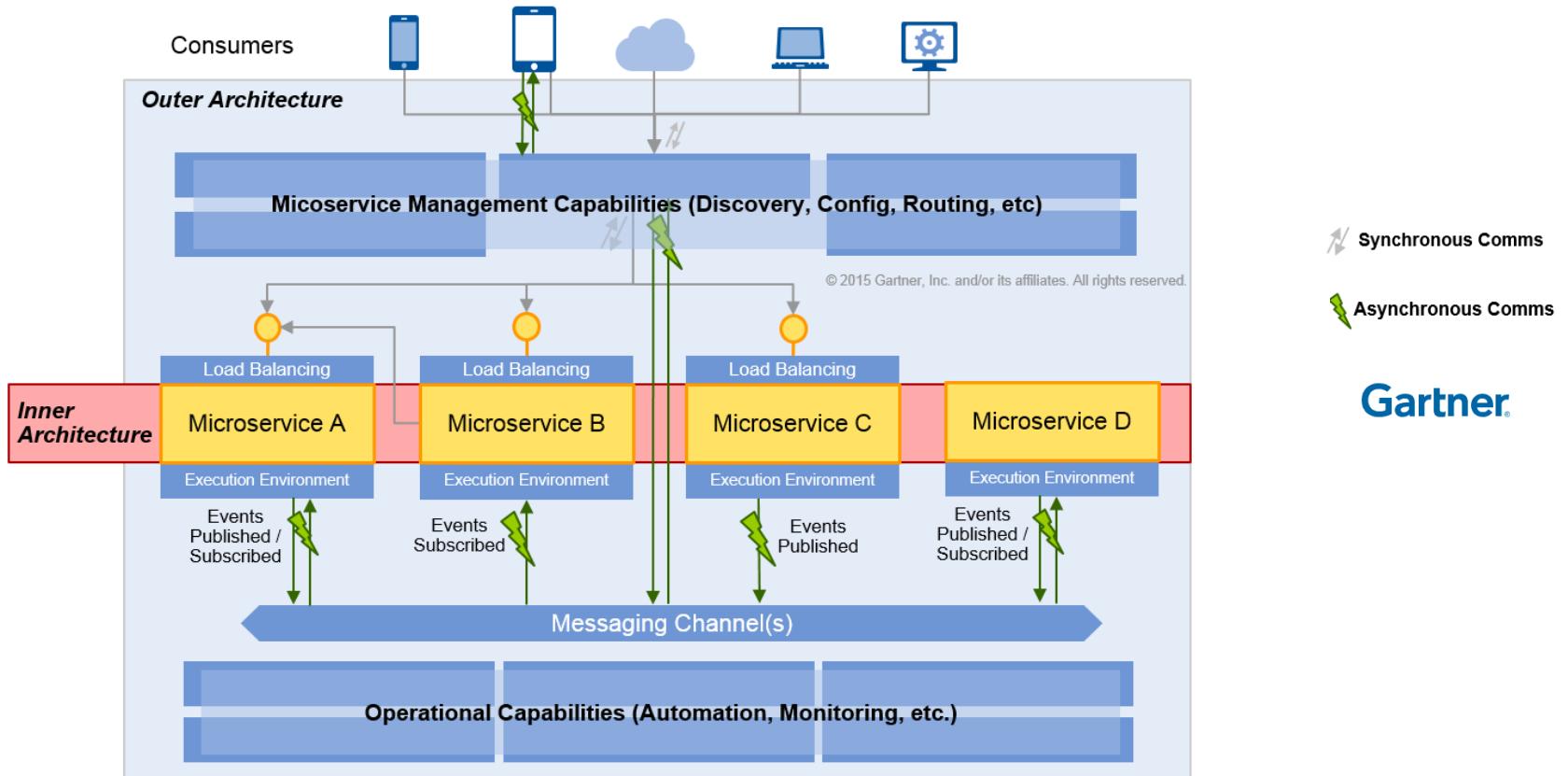
비교	Spring Cloud / Netflix OSS	Kubernetes / Istio	Event Driven MSA (Kafka / K-Native)
제공기능	API GW, Service Registry, Circuit Breaker ...	Covers all of Netflix OSS + Canary Deploy, Dark Launch ...	PubSub, Materialized View, Realtime Streaming Processing
코드 변경	필요 (only Java)	불필요 (Polyglot)	N/A (with CDC)
서비스 간 커플링	Loosely-Coupled	Loosely-Coupled	가장 Loosely-coupled
데이터 프로젝션 성능	느림 (Blocked, Request-Response)	느림 (Blocked, Request-Response)	빠름 (Non-blocking, Event Sourcing)

Kubernetes + Istio + Kafka: → High Availability, High Business Continuity



- Self-Healing
- ZDT Deploy
- Auto Scale
- Non-blocking
- Time-Decoupled
- Pluggable subscriber

Reference Cloud Native Architecture

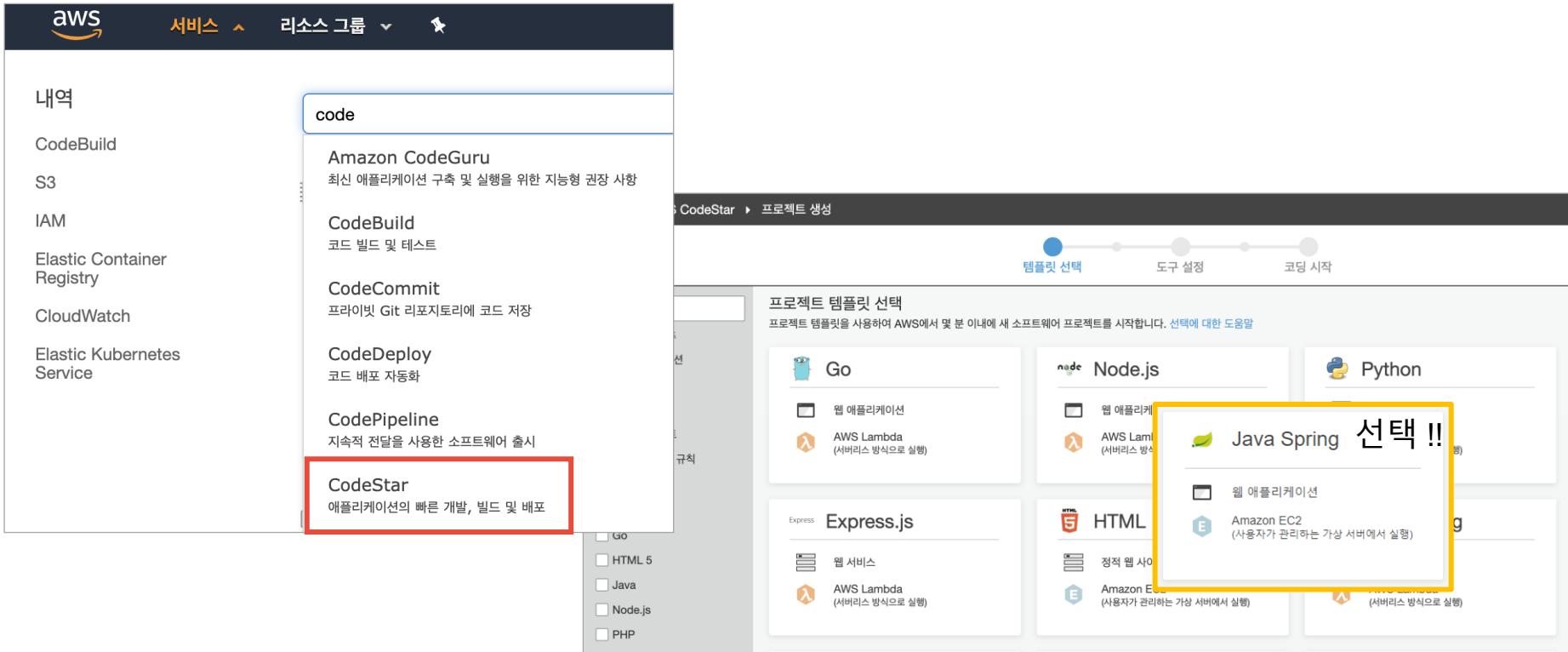


“ DevOps : Amazon CI/CD Platforms

- AWS CodeStar
- AWS CodeBuild

AWS DevOps – CodeStar

- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구



Aws DevOps - CodeStar

- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구

The image shows the AWS CodeStar project creation interface and its resulting dashboard. On the left, the 'Project Information' step of the wizard is shown. It includes fields for 'Project Name' (sample-codestar), 'Project ID' (sample-codestar), and 'Repository Type'. A yellow box highlights the 'Repository Type' field with the text 'user00-codestar 입력 !!'. Below this, there are options for AWS CodeCommit and GitHub, with GitHub selected. Another yellow box highlights the GitHub icon with the text '개발툴 건너뛰기 및 설정완료'. At the bottom, there are '이전' and '다음' buttons.

프로젝트 정보

프로젝트 이름
sample-codestar

프로젝트 ID
sample-codestar

어느 리포지토리를 사용하시겠습니까?
AWS CodeStar는 여기서 선택된 서비스를 사용하여 프로젝트의 소스 코드를 저장합니다.

AWS CodeCommit
AWS의 고가용성 Git 소스 제어 서비스입니다. 암호화, IAM 통합 등을 포함합니다.

GitHub
이 프로젝트를 위한 GitHub 소스 리포지토리를 생성합니다. 기존 GitHub 계정이 필요합니다.

리포지토리 이름
sample-codestar

다음

aws

서비스 리소스 그룹 ★

AWS CodeStar > sample-codestar

프로젝트 설정

AWS CodeStar 프로젝트 ✓ 프로젝트가 성공적으로 생성되었습니다.

타일 추가

✓ sample-codestar에 오신 것을 환영합니다!
시작하기를 도와드립니다.

세히 알아보

개발툴 건너뛰기 및 설정완료

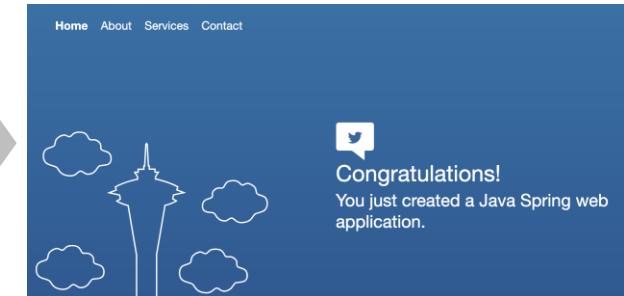
Aws DevOps - CodeStar

- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구

The screenshot shows the AWS CodeStar console. On the left, a sidebar navigation bar includes icons for Dashboard, Code, Build, Deploy, Pipelines, Teams, and Projects. The main area displays:

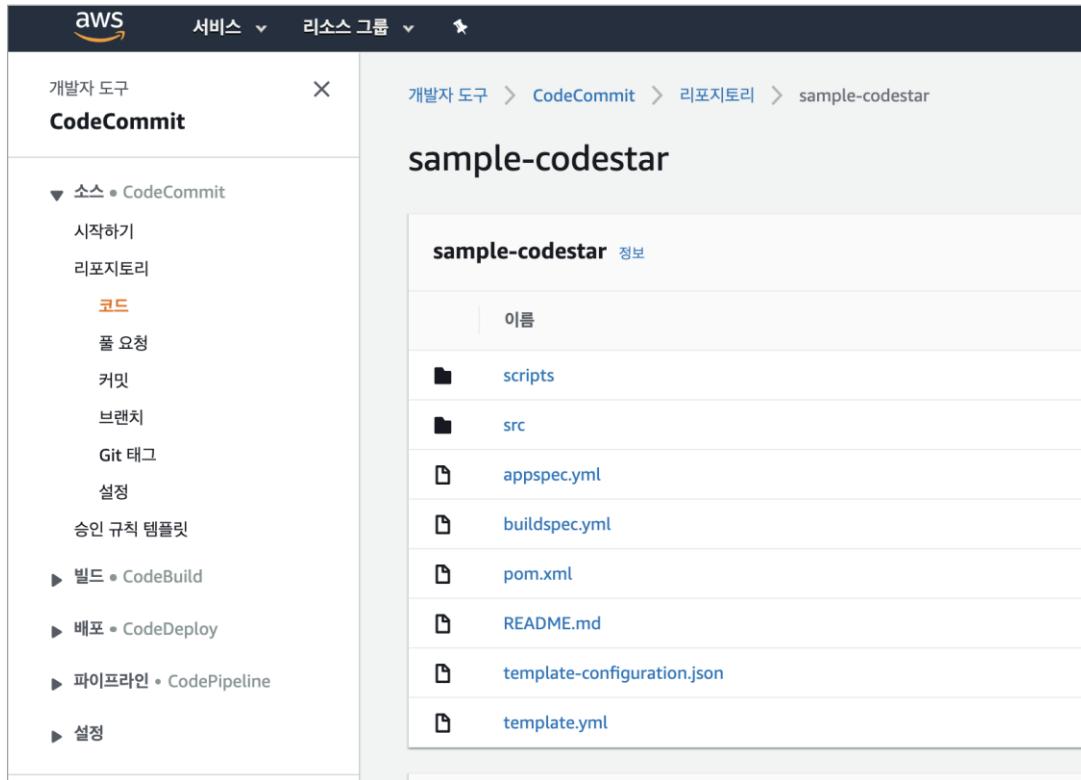
- Project Dashboard:** Shows a commit history for "sample-codestar" on the "master" branch. It lists two commits: "readme" by "kimscott" (1 minute ago) and "Initial commit by AWS CodeCommit" by "AWS CodeCommit" (1 hour ago). A "Deploy" button is visible.
- Application Activity Timeline:** A chart titled "애플리케이션 활동" (Application Activity) showing a sharp spike in activity at 12:00 on March 19, 2020, with a value of approximately 20.0. An arrow points from this chart to the "Deploy" step in the pipeline.
- Pipeline Details:** A vertical stack of three boxes representing the CI/CD pipeline stages:
 - Source:** 2020. 3. 19. 오후 12:52:27 ApplicationSource CodeCommit 성공
 - Build:** 2020. 3. 19. 오후 12:52:27 PackageExport CodeBuild 진행 중
 - Deploy:** 2020. 3. 19. 오전 11:57:43 GenerateChangeSet CloudFormation 성공

1. 자동으로 커밋-빌드-배포로 이루어진 파이프라인이 생성됨
2. EC2 인스턴스를 생성하여 어플리케이션을 자동 배포함



Aws DevOps – CodeCommit

- AWS가 Managed Service로 제공하는 프라이빗 Git 리파지토리



1. Private Repository
2. Git 사용
3. 사용하기 위해서는 User에 권한을 부여해 주어야 함
4. 권한부여 iam 주소
<https://console.aws.amazon.com/iam/>
5. buildspec.yml – 코드 빌드에 사용되는 파일
6. appspec.yml – 코드 디플로이에 사용되는 파일

Aws DevOps – CodeCommit

- AWS가 Managed Service로 제공하는 프라이빗 Git 리파지토리

Identity and Access Management(IAM)

- 대시보드
- 액세스 관리
- 그룹
- 사용자
- 역할
- 정책
- 자격 증명 공급자
- 계정 설정
- 보고서 액세스
- 액세스 분석기

권한 그룹 (1) 태그 (1) 보안 자격 증명 액세스 관리자

AWS CodeCommit에 대한 HTTPS Git 자격 증명

AWS CodeCommit 리포지토리에 대한 HTTPS 연결을 인증하는데 사용할 수 있는 사용자 이름과 암호를 생성합니다. 자격 증명 세트를 최대 100개까지 만들 수 있습니다.

자격 증명 생성 작업 ▾

사용자 이름	상태	생성 완료
uengineDev-at-979050235289	(활성)	2020-03-19 12:36 UTC+0900

```
→ MSA project git clone https://git-codecommit.ap-northeast-1.amazonaws.com/  
  
Cloning into 'sample-codestar'...  
Username for 'https://git-codecommit.ap-northeast-1.amazonaws.com': uengineDev  
Password for 'https://uengineDev-at-979050235289@git-codecommit.ap-northeast-1.amazonaws.com':  
remote: Counting objects: 44, done.  
Unpacking objects: 100% (44/44), done.  
→ MSA project cd sample-codestar  
→ sample-codestar git:(master) █
```

1. IAM 의 사용자 – 보안 자격 증명에서 CodeCommit 자격 생성
2. ID / PW 가 발급되면 해당 계정으로 CodeCommit의 git 접속 가능

Aws DevOps – CodeDeploy

- AWS가 Managed Service로 제공하는 배포 자동화 도구 (**'20. 12 현재, EKS 배포 미지원'**)

The screenshot shows the 'Application Creation' step of the AWS CodeDeploy wizard. It includes fields for the application name ('sample-deploy'), deployment group ('EC2/OnPremises'), and a note about the deployment group being required. A large orange 'Create Application' button is at the bottom.

개발자 도구 > CodeDeploy > 애플리케이션 > 애플리케이션 생성

애플리케이션 생성

애플리케이션 구성

애플리케이션 이름
애플리케이션 이름을 입력합니다.
sample-deploy
100자 이내

컴퓨팅 플랫폼
컴퓨팅 플랫폼 선택
EC2/온프레미스

취소 **애플리케이션 생성**

1. 배포 어플리케이션 – 배포 그룹 생성 순으로 진행
2. 배포 그룹에서 블루/그린 디플로이가 가능함
3. 아직 EKS를 지원 안함

The screenshot shows the 'Application Details' page for 'sample-deploy' and the 'Deployment Groups' page.

Application Details: Name: sample-deploy, Platform: Compute Platform, Region: EC2/OnPremises.

Deployment Groups: No deployment groups listed.

sample-deploy

Application details

이름 컴퓨팅 플랫폼
sample-deploy EC2/온프레미스

배포 **배포 그룹** 개정

배포 그룹

이름 상태 최근 시도한 배포 최근 성공한 배포 트리거 개수

배포 그룹 없음

CodeDeploy를 사용하여 애플리케이션을 배포하기 전에 배포 그룹을 생성해야 합니다.

배포 그룹 생성

Aws DevOps - CodePipeline

- CI/CD의 풀 라이프사이클(코드, 빌드, 배포)을 관리하는 AWS 관리형 서비스

Step 1
파이프라인 설정 선택

Step 2
소스 스테이지 추가

Step 3
빌드 스테이지 추가

Step 4
배포 스테이지 추가

Step 5
검토

파이프라인 설정 선택

파이프라인 설정

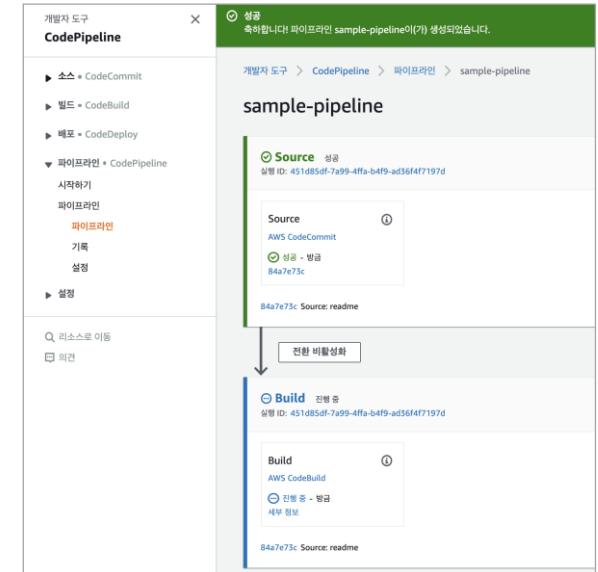
파이프라인 이름
파이프라인 이름을 입력합니다. 생성된 후에는 파이프라인 이름을 편집할 수 없습니다.
 100자를 초과할 수 없습니다.

서비스 역할
 새 서비스 역할
계정에 서비스 역할 생성
 기존 서비스 역할
계정에서 기존 서비스 역할 선택

역할 이름

서비스 역할 이름 입력
 AWS CodePipeline이 이 새 파이프라인에 사용할 서비스 역할을 생성하도록 허용

- CodeStar를 사용하면 자동으로 파이프라인 생성
- 직접 만들기 위해서는 빌드/배포 단계를 생성해야 함



CodeBuild

이점

완전관리형 빌드 서비스

AWS CodeBuild는 기업이 자체적으로 서버 및 소프트웨어를 설정, 패치, 업데이트 및 관리할 필요성을 제거합니다. 설치 또는 관리가 필요한 소프트웨어가 없습니다.

확장 가능

사용자 지정 빌드 환경을 생성하면 CodeBuild에서 지원하는 사전 패키지 형태의 빌드 도구 및 런타임에 더해 자체 빌드 도구와 프로그래밍 런타임을 AWS CodeBuild에서 사용할 수 있습니다.

지속적 크기 조정

AWS CodeBuild는 빌드 볼륨에 따라 자동으로 확장 및 축소됩니다. 제출되는 빌드는 그때마다 즉각적으로 처리되며, 각 빌드를 동시에 실행할 수 있기 때문에 빌드가 대기열에 남겨지는 일이 없습니다.

지속적 통합 및 전달 지원

AWS CodeBuild는 AWS 코드 서비스 제품군에 속합니다. 즉, 이 서비스를 사용하여 CI/CD(지속적 통합 및 전달)를 위한 완전한 자동 소프트웨어 릴리스 워크플로를 생성할 수 있습니다. 또한 CodeBuild를 기준 CI/CD 워크플로에 통합하는 것도 가능합니다. 예를 들어 기존 Jenkins 서버 설정에서 CodeBuild를 작업자 노드로 사용하여 빌드를 분산할 수 있습니다.

사용량에 따라 지불

AWS CodeBuild에서는 빌드를 완료할 때까지 걸리는 시간(분)을 기준으로 비용이 청구됩니다. 이 말은 사용하지 않는 빌드 서버 용량에 대해서는 비용을 지불할 필요가 없다는 것을 의미합니다.

보안

AWS CodeBuild에서는 고객이 지정하고 AWS Key Management Service(KMS)로 관리되는 키를 사용해 빌드 애플리케이션이 암호화됩니다. CodeBuild는 AWS Identity and Access Management(IAM)에 통합되므로 사용자 지정 권한을 빌드 프로젝트에 할당할 수도 있습니다.

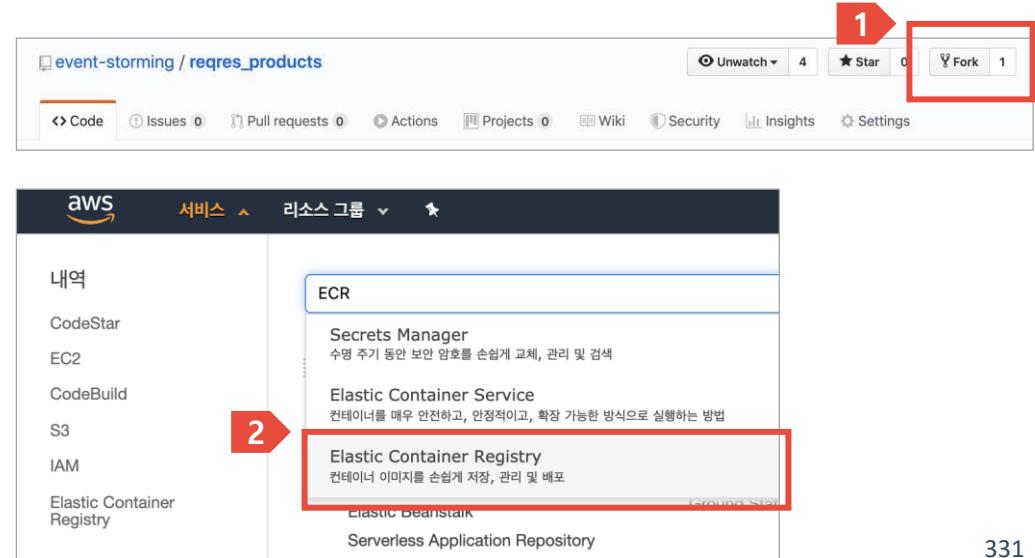
CodeBuild

- Github 의 소스를 빌드 / 패키징 하여 Docker Image 로 변경
- Docker Image 를 ECR 로 업로드
- 업로드된 Image 를 EKS(Amazon Elastic Kubernetes Service) 에 배포

1. Github 리파지토리 Fork
https://github.com/event-storming/reques_products

2. Docker image 를 저장하기 위한 공간인 ECR 에 “admin00-products” Repository 생성

3. Fork 받은 리파지토리의 Root에서 buildspec.yml을 편집하여 5행의 이미지명 수정



CodeBuild : 생성 설정 (1/3)

1. CodeBuild 프로젝트 생성

2. Github 계정 연결 후 Fork 한 리포지토리 연결

1

개발자 도구 > CodeBuild > 빌드 프로젝트 > 빌드 프로젝트 생성

빌드 프로젝트 생성

프로젝트 구성

프로젝트 이름

sample-products

프로젝트 이름은 2~255자여야 합니다. 글자(A-Z 및 a-z), 숫자(0~9) 및 특수 문자(- 및 _)를 포함할 수 있습니다.

설명 - 선택 사항

빌드 배치 - 선택 사항

빌드 배치 활성화

▶ 추가 구성

태그

2

소스

소스 1 - 기본

소스 공급자

GitHub

리포지토리

퍼블릭 리포지토리

내 GitHub 계정의 리포지토리

GitHub 리포지토리

https://github.com/kimscott/reqres_products.git

https://github.com/<user-name>/<repository-name>

연결 상태

OAuth를 사용하여 GitHub에 연결되었습니다.

GitHub에서 연결 해제

소스 버전 - 선택 사항 정보

풀 요청, 브랜치, 커밋 ID, 태그 또는 참조와 커밋 ID를 입력합니다.

▶ 추가 구성

Git clone 깊이, Git 하위 모듈

CodeBuild : 생성 설정 (2/3)

1. Webhook 을 설정하여 Github에 코드가 푸쉬될 때마다 트리거 동작
2. 빌드가 돌아갈 환경 설정
 - 운영체제 : 리눅스
 - 런타임 : Standard
 - 이미지 : Standard3.0 (**이미지별로 환경이 다르니 주의**)
 - 환경유형 : Linux
 - 도커 권한 체크 필수

2

1

기본 소스 Webhook 이벤트 정보

필터 그룹 추가

Webhook - 선택 사항

코드 변경이 이 리포지토리에 푸시될 때마다 다시 빌드

한 개 이상의 Webhook 이벤트 필터 그룹을 추가하여 새 빌드를 트리거하는 이벤트를 지정합니다. Webhook 이벤트 필터 그룹을 추가하지 않은 경우에는 코드 변경이 리포지토리에 푸시될 때마다 새 빌드가 트리거됩니다.

Webhook 이벤트 필터 그룹 1

이벤트 유형

▶ 이러한 조건에서 빌드 시작

▶ 이러한 조건에서 빌드 시작 안 함

환경

환경 이미지

관리형 이미지
AWS CodeBuild에서 관리하는 이미지 사용

사용자 지정 이미지
도커 이미지 지정

운영 체제

Ubuntu

① 이제 프로그래밍 언어 런타임은 Ubuntu 18.04의 표준 이미지에 포함됩니다. 이 이미지는 콘솔에서 생성된 새 CodeBuild 프로젝트에 대해 권장됩니다. 자세한 내용은 CodeBuild에서 제공하는 Docker 이미지 [문서](#)를 참조하십시오.

런타임

Standard

이미지

aws/codebuild/standard:2.0

이미지 버전

이 런타임 버전에 항상 최신 이미지 사용

환경 유형

Linux

권한이 있음

도커 이미지를 빌드하거나 빌드의 권한을 승격하려면 이 플래그를 활성화합니다.

서비스 역할

새 서비스 역할
계정에 서비스 역할 생성

기존 서비스 역할
계정에서 기존 서비스 역할 선택

역할 이름

codebuild-sample-products-service-role

서비스 역할 이름 입력

▶ 추가 구성

제한 시간, 인증서, VPC, 컴퓨팅 유형, 환경 변수, 파일 시스템

CodeBuild : 생성 설정 (3/3)

1. 추가 구성에 환경 변수값을 입력 : AWS 계정 ID
 - AWS_ACCOUNT_ID
 - ECR에서 이미지 주소의 가장 앞에 값임
 - echo \$(aws sts get-caller-identity --query Account --output text) 명령으로 조회 가능
2. buildspec.yml 파일을 사용하겠다고 체크 후, 저장

1

▼ 추가 구성
제한 시간, 인증서, VPC, 컴퓨팅 유형, 환경 변수, 파일 시스템

환경 변수	이름	값	유형	제거
	AWS_ACCOUNT_ID	97905023****	일반 텍스트	▼
환경 변수 추가				

2

Buildspec

빌드 사양

buildspec 파일 사용
YAML 형식의 buildspec 파일에 빌드 명령 저장

빌드 명령 삽입
빌드 명령을 빌드 프로젝트 구성으로 저장

Buildspec 이름 - 선택 사항
기본적으로 CodeBuild는 소스 코드 루트 디렉토리에서 buildspec.yml 파일을 찾습니다. buildspec 파일이 다른 이름 또는 위치를 사용하는 경우 여기에 소스 루트의 경로를 입력하십시오(예: buildspec-two.yml 또는 configuration/buildspec.yml).

아티팩트

아티팩트 1 – 기본

유형

아티팩트 없음

테스트를 실행하거나 도커 이미지를 Amazon ECR에 넣는 경우 [아티팩트 없음]을 선택할 수 있습니다.

▶ 추가 구성
캐시, 암호화 키

아티팩트 추가

CodeBuild : 빌드 시작

The screenshot shows the AWS CodeBuild console interface. On the left, a sidebar menu lists various services: '개발자 도구', 'CodeBuild' (selected), '소스 + CodeCommit', '빌드 + CodeBuild' (selected), '프로젝트 빌드', '설정', '빌드 내역', '보고서 그룹', '보고서 기록', '계정 치료', '배포 + CodeDeploy', and '파이프라인 + CodePipeline'. The main content area displays a build project named 'sample-products'. A red arrow labeled '1' points to the '빌드 시작' (Start Build) button at the top right of the configuration panel. Another red arrow labeled '2' points to the '빌드 상태' (Build Status) section, which shows the build is currently '진행 중' (In Progress). The status table includes columns for '상태', '시작한 사용자', '빌드 ARN', and '해결된 소스 버전'. Below this, it shows '시작 시간' (Start Time: 3월 19, 2020 3:48 오후 (UTC+9:00)), '종료 시간' (End Time: -), '빌드 번호' (Build Number: 1), and '환경 변수' (Environment Variables). At the bottom, a large black box contains the text '빌드 로그의 마지막 1000줄을 표시하는 중입니다. 전체 로그 보기' (Showing the last 1000 lines of the build log. View full log) and a '테일 로그' (Tail Log) button.

개발자 도구 > CodeBuild > 빌드 프로젝트 > sample-products

sample-products

1

구성

소스 공급자 GitHub

기본 리포지토리 kimscott/regres_products

이티택트 업로드 위치 -

빌드 베지 활성

빌드 기록 | 빌드 세부 정보 | 빌드 트리거 | 지표

빌드 기록

C | 빌드 중지 | 이티택트 보기

2

빌드 상태

상태	시작한 사용자	빌드 ARN	해결된 소스 버전
진행 중	root	arn:aws:codebuild:ap-northeast-1:979050235289:build/sample-products:8f6dc424-0415-44f0-9cbc-0e2fab8195a8	-

시작 시간 종료 시간 빌드 번호

3월 19, 2020 3:48 오후 (UTC+9:00) - 1

환경 변수 빌드 세부 정보

빌드 로그 | 단계 세부 정보 | 보고서 | 환경 변수 | 빌드 세부 정보

빌드 로그의 마지막 1000줄을 표시하는 중입니다. 전체 로그 보기

테일 로그

CodeBuild : buildspec.yml

```
1 version: 0.2
2
3 env:
4   variables:
5     IMAGE_REPO_NAME: "products"
6
7 phases:
8   install:
9     runtime-versions:
10    java: openjdk8
11 #     docker: 18
12 pre_build:
13   commands:
14     - echo Logging in to Amazon ECR...
15     - echo $IMAGE_REPO_NAME
16     - echo $AWS_ACCOUNT_ID
17     - echo $AWS_DEFAULT_REGION
18     - echo ${CODEBUILD_RESOLVED_SOURCE_VERSION}
19     - echo start command
20 #     - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION)
21 build:
22   commands:
23     - echo Build started on `date`
24     - echo Building the Docker image...
25     - mvn package -Dmaven.test.skip=true
26 #     - docker build -t $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION .
27 post_build:
28   commands:
29     - echo Build completed on `date`
30     - echo Pushing the Docker image...
31 #     - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
32
33 # cache:
34 #   paths:
35 #     - '/root/.m2/**/*'
```

1. buildspec 에 대한 상세 가이드 -

https://docs.aws.amazon.co/m/ko_kr/codebuild/latest/userguide/build-spec-ref.html

2. phases: 필수 시퀀스입니다. 빌드의 각 단계 동안 CodeBuild가 실행하는 명령을 나타냅니다.
3. Install, pre_build, build, post_build 4개의 시퀀스를 적절히 사용할 수 있음

CodeBuild : cache 적용

The screenshot shows the AWS CodeBuild console with the following interface elements:

- Sidebar:** Lists services: 내역, Elastic Container Registry, IAM, EC2, CloudWatch Metrics.
- Header:** aws logo, 서비스 ▾, 리소스 그룹 ▾, a star icon.
- Main Content Area:**
 - Caching Provider:** A dropdown menu titled "캐시 유형" (Cache Type) is set to "Amazon S3".
 - Bucket Name:** An input field titled "캐시 버킷" (Cache Bucket) contains the value "mvn-cache-codebuild".
 - Cache Location:** A dropdown menu titled "캐시 경로 접두사 - 선택 사항" (Cache Path Prefix - Optional) is empty.
 - Expiration Policy:** A section titled "캐시 수명 주기 - 선택 사항" (Cache Expiration - Optional) includes a note: "경로 접두사에 따라 캐시 버킷 내 전체 또는 일부 객체에 수명 주기 만료 작업을 적용할 수 있습니다." (Cache expiration can be applied to all objects in the bucket or specific prefixes). It features two input fields and a "만료 추가" (Add Expiration) button.
- Buttons at the bottom:** "취소" (Cancel), "아티팩트 업데이트" (Artifact Update).

1. Maven 으로 빌드시, 라이브러리를 캐쉬화 시킴 (캐쉬가 없으면 빌드할때마다 약 5분정도 소요됨)
2. 캐쉬를 저장할 S3 스토리지 생성
3. 버킷 생성 – 리전을 CodeBuild 생성 리전과 일치 해야함
4. 빌드 선택 – 아티팩트 편집
5. 추가구성 – 캐시 유형을 S3 로 선택 후 버킷 선택
6. buildspec.yml 을 에서 cache 부분 주석 해제 후 커밋

CodeBuild : ECR 연결

The screenshot shows the AWS CodeBuild console. On the left, a sidebar menu is open under '빌드' (Build) with 'CodeBuild' selected. The main area is titled '환경' (Environment). It shows the build image as 'aws/codebuild/standard:2.0' and the environment type as 'Linux'. A red box labeled '1' highlights the '서비스 역할' (Service Role) field, which contains the ARN: 'arn:aws:iam::979050235289:role/service-role/codebuild-sample-products-service-role'. Below this, it says '제한 시간' (Time Limit) is set to '1시간 0분' (1 hour 0 minutes). At the bottom, it says '레지스트리 자격 증명' (Registry Authentication). In the bottom right, there's a JSON editor with tabs for '시작적 편집기' (Text Editor) and 'JSON'. The JSON code is:

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Action": [  
6         "ecr:BatchCheckLayerAvailability",  
7         "ecr:CompleteLayerUpload",  
8         "ecr:GetAuthorizationToken",  
9         "ecr:InitiateLayerUpload",  
10        "ecr:PutImage",  
11        "ecr:UploadLayerPart"  
12      ],  
13      "Resource": "*",  
14      "Effect": "Allow"  
15    }  
16  ]  
17 }
```

A red box labeled '2' highlights the 'Statement' section of the JSON code.

1. 프로젝트 빌드 - 빌드 선택 - 빌드 세부정보 - 환경 - 서비스 역할 클릭하여 IAM 역할 페이지로 이동
2. 인라인 정책 추가 > json 하여 ECR 관련 정책을 추가
3. 관련 스크립트는 WorkFlowy에 “CodeBuild 와 ECR 연결 정책” 검색
4. 정책명 입력 후, user00-codebuild-policy 검토 및 생성

CodeBuild : Docker Build/Push

The screenshot shows the AWS CodeBuild interface. At the top, there's a navigation bar with 'reqres_products' and 'buildspec.yml'. Below it is an 'Edit file' section with a 'Preview changes' button and settings for 'Spaces', '2', and 'No wrap'. The main area contains the buildspec.yml configuration:

```
7 phases:
8   install:
9     runtime-versions:
10    java: openjdk8
11    docker: 18
12  pre_build:
13    commands:
14      - echo Logging to Amazon ECR...
15      - echo $IMAGE_REPO_NAME
16      - echo $AWS_ACCOUNT_ID
17      - echo $AWS_DEFAULT_REGION
18      - echo $CODEBUILD_RESOLVED_SOURCE_VERSION
19      - echo start command
20      - $aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION
21  build:
22    commands:
23      - echo Build started on `date`
24      - echo Building the Docker image...
25      - mvn package -Dmaven.test.skip=true
26      - docker build -t $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
27  post_build:
28    commands:
29      - echo Build completed on `date` |
30      - echo Pushing the Docker image...
31      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
32
```

Below this is a 'Commit changes' dialog with fields for 'Update buildspec.yml' and 'Add an optional extended description...'. It includes two radio button options: 'Commit directly to the master branch.' (selected) and 'Create a new branch for this commit and start a pull request.' A red arrow labeled '1' points to the 'Commit changes' button.

1. buildspec.yml 파일의 Docker 관련 주석을 모두 해제 후, 코드 커밋/푸쉬
2. 빌드 성공 후 ECR 에 Docker Image 가 정상적으로 생성되었는지 확인

The screenshot shows the AWS ECR 'products' page. It lists one image entry:

이미지 태그	이미지 URI
6dd5866a8971279f62376425d6f374e6fd682c9b	979050235289.dkr.ecr.ap-northeast-1.amazonaws.com/products:6dd5866a8971279f62376425d6f374e

A red arrow labeled '2' points to the image tag column.

CodeBuild : EKS 연결 (1/4)

1. 쿠버네티스에 접속 하기 위해서는 쿠버네티스 API 주소와 클러스터 어드민 권한이 있는 토큰이 있으면 접속 가능
(실제로 kubectl 명령어가 두개의 조합으로 이루어 져 있음)
2. 쿠버네티스 API 주소 위치

Amazon EKS	Kubernetes 버전	플랫폼 버전
클러스터	1.15	eks.1
Amazon ECR	API 서버 엔드포인트	
리포지토리	https://A4CB98CD955A2E274B343BCEA284621F.yl4.ap-northeast-1.eks.amazonaws.com	

CodeBuild : EKS 연결 (2/4)

1. 쿠버네티스 토큰 생성은 ServiceAccount 생성 후 해당 SA 에 cluster-admin 룰을 주면 된다.
2. WorkFlowy 에서 “CodeBuild 와 EKS 연결” 검색 후 토큰 생성

```
→ ~ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')
Name:           eks-admin-token-hffgq
Namespace:      kube-system
Labels:         <none>
Annotations:   kubernetes.io/service-account.name: eks-admin
               kubernetes.io/service-account.uid: 18c3c8c6-a0dc-4f0b-9ba9-a85b6322aa7f
Type:          kubernetes.io/service-account-token

Data
=====
namespace: 11 bytes
token: eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcmlldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pb9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOjJrdWJlLXN5c3RlbSIsImt1YmVybml0ZXMuaw8vc2VydmljZWFlY29
```

CodeBuild : EKS 연결 (3/4)

- 빌드 파일의 “환경” 영역 편집 버튼을 눌러, 아래 값을 넣은 후에 저장한다.
- 쿠버네티스 API 주소를 “KUBE_URL”이라는 이름으로 저장한다.
- 클러스터 어드민 토큰 값을 “KUBE_TOKEN”이라는 이름으로 저장한다.

환경 변수			
이름	값	유형	제거
AWS_ACCOUNT_ID	979050235***	일반 텍스트	▼ 제거
KUBE_URL	https://A4CB98CD955A	일반 텍스트	▼ 제거
KUBE_TOKEN	eyJhbGciOiJSUzI1NiIsIm	일반 텍스트	▼ 제거
환경 변수 추가			

CodeBuild : EKS 연결 (4/4)

1. Github 소스의 root에 buildspec-kubectl.yml 파일의 내용을 복사하여 buildspec.yml 파일에 붙여 넣은 후 커밋/푸쉬 한다.
2. CodeBuild 가 자동 실행 되어, EKS 클러스 배포 후 서비스가 정상적으로 기동 되었는지 확인 한다.

1

```
post_build:  
  commands:  
    - echo Pushing the Docker image...  
    - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$_PROJECT_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION  
    - echo connect kubectl  
    - kubectl config set-cluster k8s --server="$KUBE_URL" --insecure-skip-tls-verify=true  
    - kubectl config set-credentials admin --token="$KUBE_TOKEN"  
    - kubectl config set-context default --cluster=k8s --user=admin  
    - kubectl config use-context default  
    - |  
      cat <<EOF | kubectl apply -f -  
      apiVersion: v1  
      kind: Service  
      metadata:
```

2 ➔ ~ kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
orders-66f47b6455-pp6dh	1/1	Running	0	40h
products-dbc86b85b-skblg	1/1	Running	0	2m41s

AWS CodeBuild를 활용한 CI/CD 2nd Lab.



From Page.38

1. Github에 있는 소스 fork
https://github.com/event-storming/reqres_delivery
2. Docker image를 저장하기 위한 공간인 ECR에 Repository 생성
3. reqres_products 레파지토리에서 buildspec.yml 파일 내용을 복사해 오기
4. Fork 받은 리파지토리의 Root에서 buildspec.yml을 편집하여 5행의 이미지명 수정
- 위에서 생성한 ECR Repository와 동일한 이름 입력

CodeBuild Webhook Event Filter (1/2)

- 환경 별로 다른 빌드를 하고 싶으면 빌드 파일을 다르게 생성하여 빌드 프로젝트를 각자 생성하면 된다.
(예 : 개발 환경 - buildspec-dev.yml, 운영환경 - buildspec-prod.yml)
- 기존 프로젝트의 webhook 트리거를 잠시 제거하고, buildspec-sample.yml 파일로 빌드 프로젝트를 생성한다.
- 특정 파일 수정일때 트리거 작동 안하도록 설정

- Webhook 이벤트 항목을 편집한다.
- 이러한 조건에서 빌드 시작 안함에서 FILE-PATH 부분에 “^buildspec.*” 값을 넣는다.
- buildspec 으로 시작하는 모든 파일일때 웹훅 이벤트가 동작 안한다.
- 정규식으로 작성해야 한다.
- buildspec-sample.yml 파일을 수정하여 트리거가 작동하는지 확인 한다 (작동하면 안됨)

Webhook 이벤트 필터 그룹 1

이벤트 유형

PUSH X

▼ 이러한 조건에서 빌드 시작

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
------------------	------------------	------------------	-------------------

▼ 이러한 조건에서 빌드 시작 안 함

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
------------------	------------------	------------------	-------------------

^buildspec.*

CodeBuild Webhook Event Filter (2/2)

1. 태그 이벤트일때 트리거 작동하도록 설정

- 이러한 조건에서 빌드 시작 항목 – HEAD_REF 에 “`^refs/tags/.*`” 를 입력한다.
 - > 팁 : (아래 이미지처럼 빌드 시작안함도 같이 있을 경우, buildspec 파일만 변경하고, tag 를 하여도 트리거는 동작하지 않는다.)
 - > 팁 : 필터를 여러개 생성해도 1개만 만족하면 트리거는 작동한다
- 저장 후, README.md 파일을 수정하여 트리거가 작동하는지 확인한다. (빌드 시작 안함)
- github Tag 를 설정 한 후 트리거가 작동하는지 확인 한다. (빌드 시작함)
- HEAD_REF 에 “`^refs/heads/dev$`” 를 추가하여 dev 브랜치에서만 작동하는것을 연습해보자.

Webhook 이벤트 필터 그룹 1

이벤트 유형

PUSH X

▼ 이러한 조건에서 빌드 시작

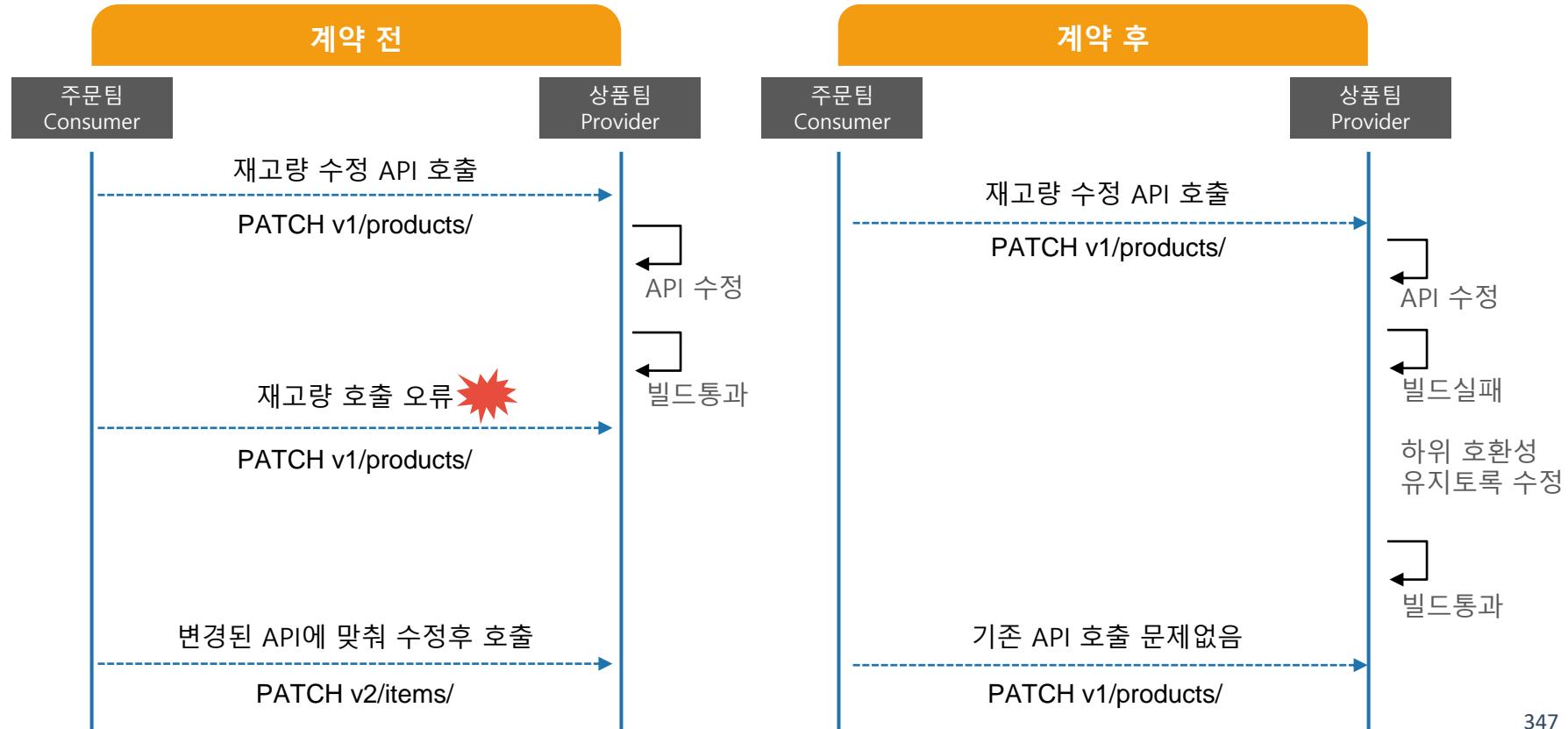
ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
<input type="text"/>	<input type="text" value="^refs/tags/.*"/>	<input type="text"/>	<input type="text"/>

▼ 이러한 조건에서 빌드 시작 안 함

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="^buildspec.*"/>

86 [Container] 2020/03/20
87 tag name2 : tag/v1.0.1
88
89 [Container] 2020/03/20
90
91 [Container] 2020/03/20
92 tag name3 : v1.0.1
93

Contract Test



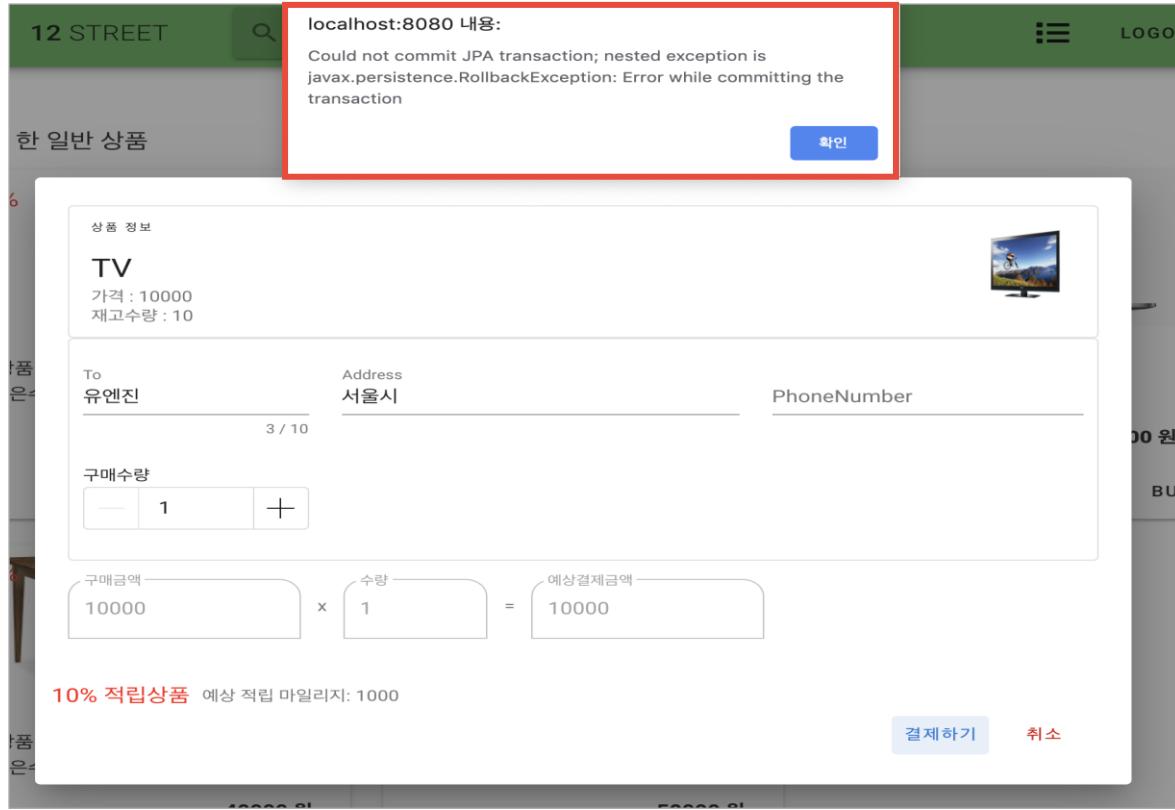
상품팀에서 API를 일방적으로 변경

소스 위치

products / src / main / java / com / example / template / ProductController.java

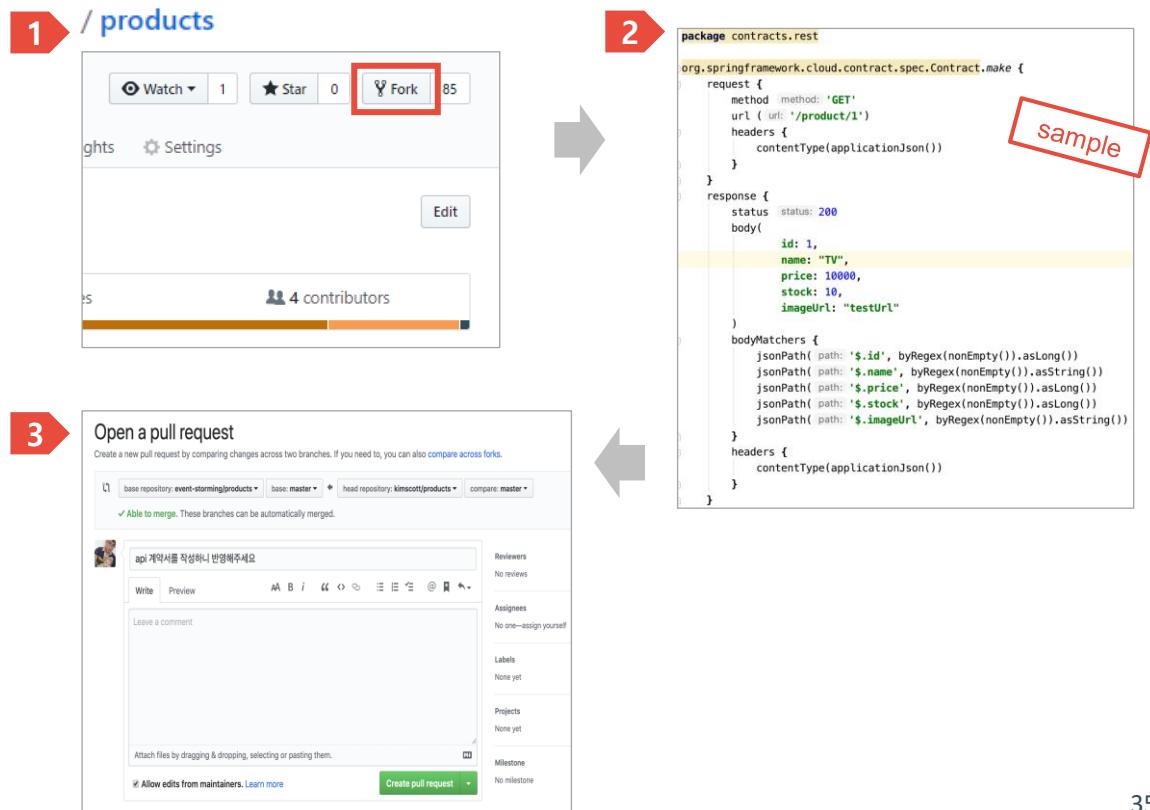
```
13  
14     @GetMapping("/product/{productId}") /item/{productId} 로 변경  
15     Product productStockCheck(@PathVariable(value = "productId") Long productId) {  
16         return this.productService.getProductById(productId);  
17     }  
18 }  
19
```

주문팀의 서비스 장애 발생



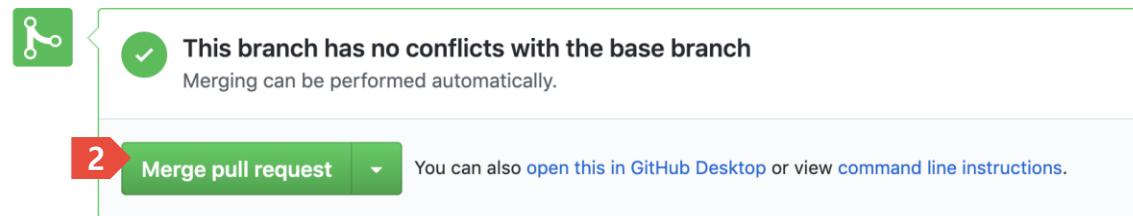
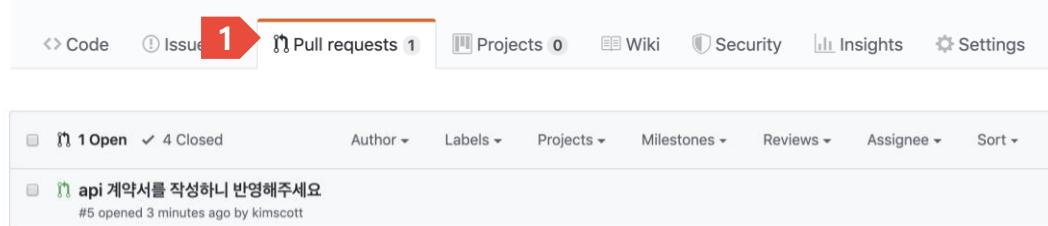
계약체결(1/2) - 주문팀에서 계약서 작성 후, 상품팀에 체결 요청

1. 상품팀 소스 복사 (포크 생성)
2. 주문팀이 계약서 생성
3. 주문팀에서 생성한 계약서 (productGet.groovy) 파일을 상품팀에 Pull request 요청함



계약체결(2/2) - 상품팀에서 계약서 수락

- 상품팀 : Pull Request 메뉴 선택
- 상품팀은 해당 계약서를 accept 하여 반영함
- 상품 서비스는 이제부터는 계약서를 안지켰을때 아예 배포가 안됨



계약체결 후, 상품팀은 계약 위반으로 배포 실패함

CloudBuild에서 mvn package 단계 실패

빌드 정보

상태	빌드 실패
빌드 ID	6db60b82-3b99-4aef-9870-ce85dce23cccd
이미지	-
트리거	master 브랜치에 푸시 (products) GitHub event-storming/products ↗
소스	GIT 커밋 9997fc8cebfff7e444d94b790378bb58dfc324362c ↗
환경 변수	CLOUDSDK_COMPUTE_ZONE=asia-northeast1-a CLOUDSDK_CONTAINER_CLUSTER=standard-cluster-1
시작 시간	2019년 10월 29일 오후 3시 42분 59초 UTC+9
걸린 시간	1분 54초

로그가 너무 커서 이 페이지에 완전히 표시할 수 없습니다. 빌드 단계의 로그가 누락되거나 완료되지 않을 수 있습니다.

모두 펼치기

빌드 단계

build gcr.io/cloud-builders/mvn -- clean package	1분 46초
---	--------

실패 LOG

```
Step #0 - "build": 2019-10-29 06:44:51.547 DEBUG 69 --- [           main] o.s.c.c.v.m.stream.StreamStubMessages      : Picked channel name is [event-out]
Step #0 - "build": [ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 4.976 s <<< FAILURE! - in com.example.template.MessagingTest
Step #0 - "build": [ERROR] validate_productChanged(com.example.template.MessagingTest)  Time elapsed: 0.221 s  <<< ERROR!
Step #0 - "build": com.jayway.jsonpath.PathNotFoundException: No results for path: ${'productName'}
Step #0 - "build":         at com.example.template.MessagingTest.validate_productChanged(MessagingTest.java:43)
Step #0 - "build": 
```

상품팀에서는 하위 호환성을 유지하며, 추가 API 제공

products / src / main / java / com / example / template / ProductController.java

```
@GetMapping("/item/{productId}")
Product productStockCheck(@PathVariable(value = "productId") Long productId) {
    return this.productService.getProductById(productId);
}

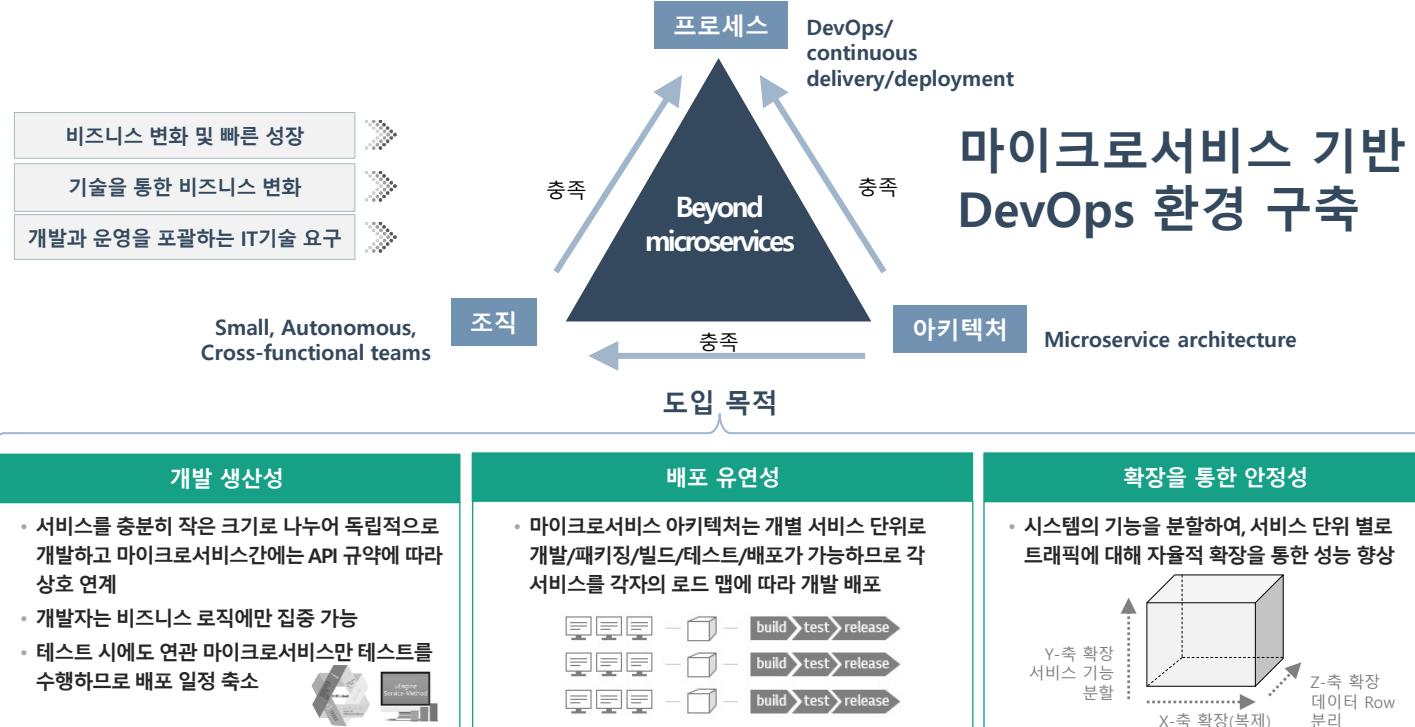
@GetMapping("/product/{productId}")
Product productStockCheck1(@PathVariable(value = "productId") Long productId) {
    return this.productService.getProductById(productId);
}
```

기존의 하위
호환성 API 유지

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 아키텍처입니다.



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 커뮤니케이션 Anti-corruption layer를 통해 연동	이벤트 기반으로 식별된 도메인 영역, 멀티 라이즈 뷰, 읽기/쓰기 (커멘드)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 트랜잭션이 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 총족	이벤트 기반 데이터 관리, 이벤트 소싱 및 커멘드 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 원천한 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 운영	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 로드밸런싱 서버사이드 로드밸런싱	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	마이크로서비스를 사용하여 가벼운 미들웨어 통합 레이어를 접하는 API 게이트웨이	AWS 램다와 같은 이벤트 기반의 단일 목적을 위한 전용의 서비스



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 증가량	데이터 용량이 많아 부하와 성능에 영향을 줄 수 있는지 업무 확인	현행 시스템 기준 핵심 앤터티 데이터 용량 측정	1. 성능에 영향을 줄 수 있는 데이터량 기준 정의 2. 1번 기준 이상인 업무 Y로 표기	500GB 이상
		현행 시스템에 있는 업무 중 데이터 증가량이 많아 부하와 성능에 영향을 줄 수 있는 업무 확인	현행 시스템 기준 핵심 앤터티 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
	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 트랜잭션이 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

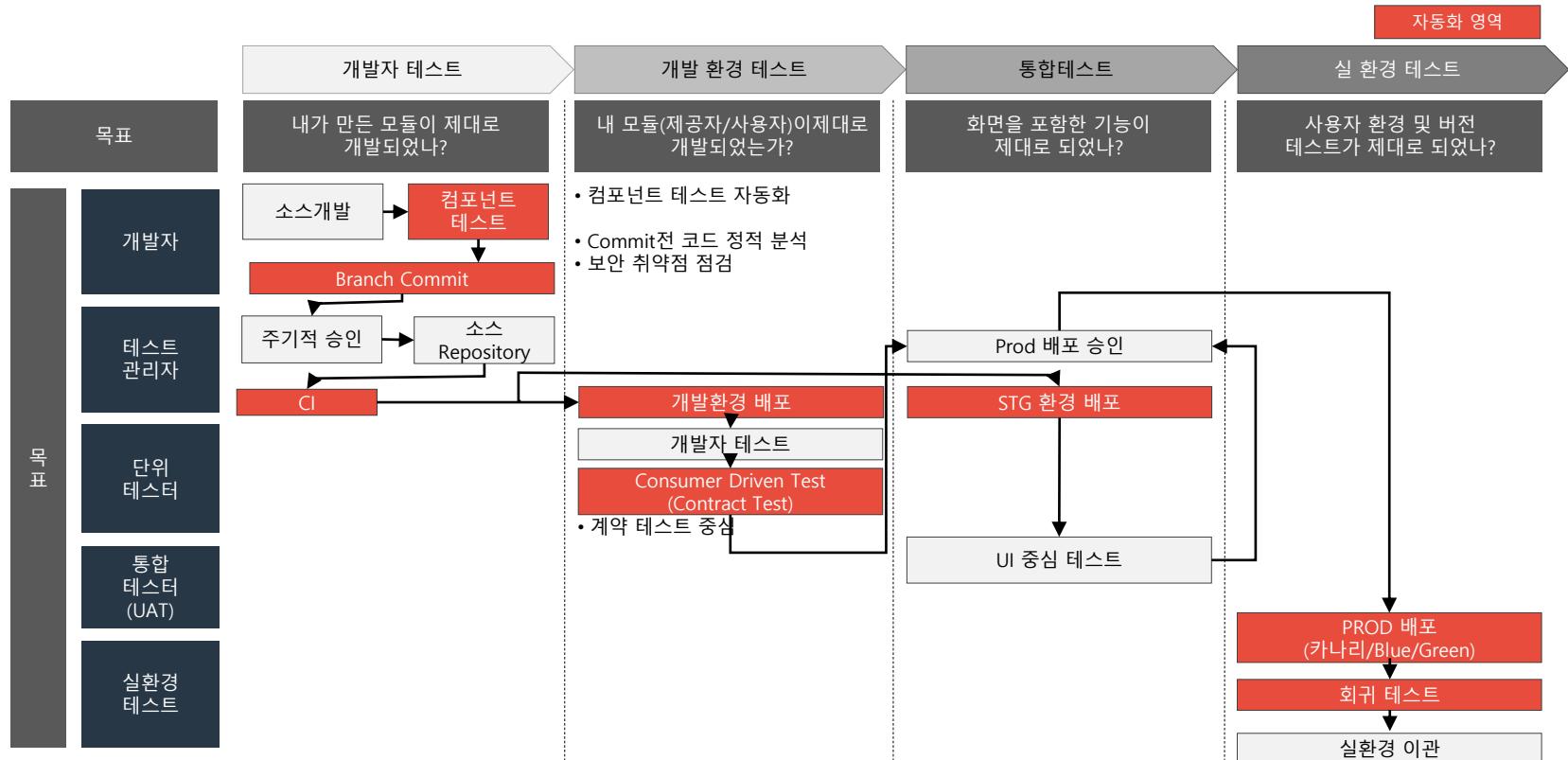
아키텍처 패턴	체크 포인트	회피 패턴
Circuit Breaker	<ul style="list-style-type: none"> 성능저하를 막아주나, Fail-fast 전략으로 사용자 경험이 나빠질 수 있음 적용대상이 비동기, 이벤트 기반으로 처리가능하다면 그 기반으로 전환 	Saga
Database per service	<ul style="list-style-type: none"> ACID 트랜잭션 비용을 포기 ACID 트랜잭션 비용 포기가 불가하다면, Shared database 로 처리해야 하거나 Semantic Locking 통한 Eventual Transaction 상의 Lock 을 구현해야 함 	Schema Per Service (Anti)
Service Registry	<ul style="list-style-type: none"> 서비스 레지스트리의 유형 선택: API 기반(Eureka), DNS 기반(Kube-dns) 	Saga, CQRS
Saga	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 하나 이상의 마이크로서비스에서 추출한 데이터로 뷔를 구성해야 하는경우 잦고 빠른 마이크로서비스 내에서의 Read 가 발생하는 경우에 사용 	HATEOAS
Event Sourcing	<ul style="list-style-type: none"> 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 찾음 이벤트 소싱에서의 이벤트는 Append only 이기 때문에 데이터의 Diff 의 정보를 충실히 포함해야 함 	
Backends for frontends	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Server-side Rendering 은 Microservice 의 장점을 회색하므로 가능한 MVVM 기반 Client-side Rendering 을 적용해야 함 	Server-side rendering (Anti)

MSA 성능 테스트 방안

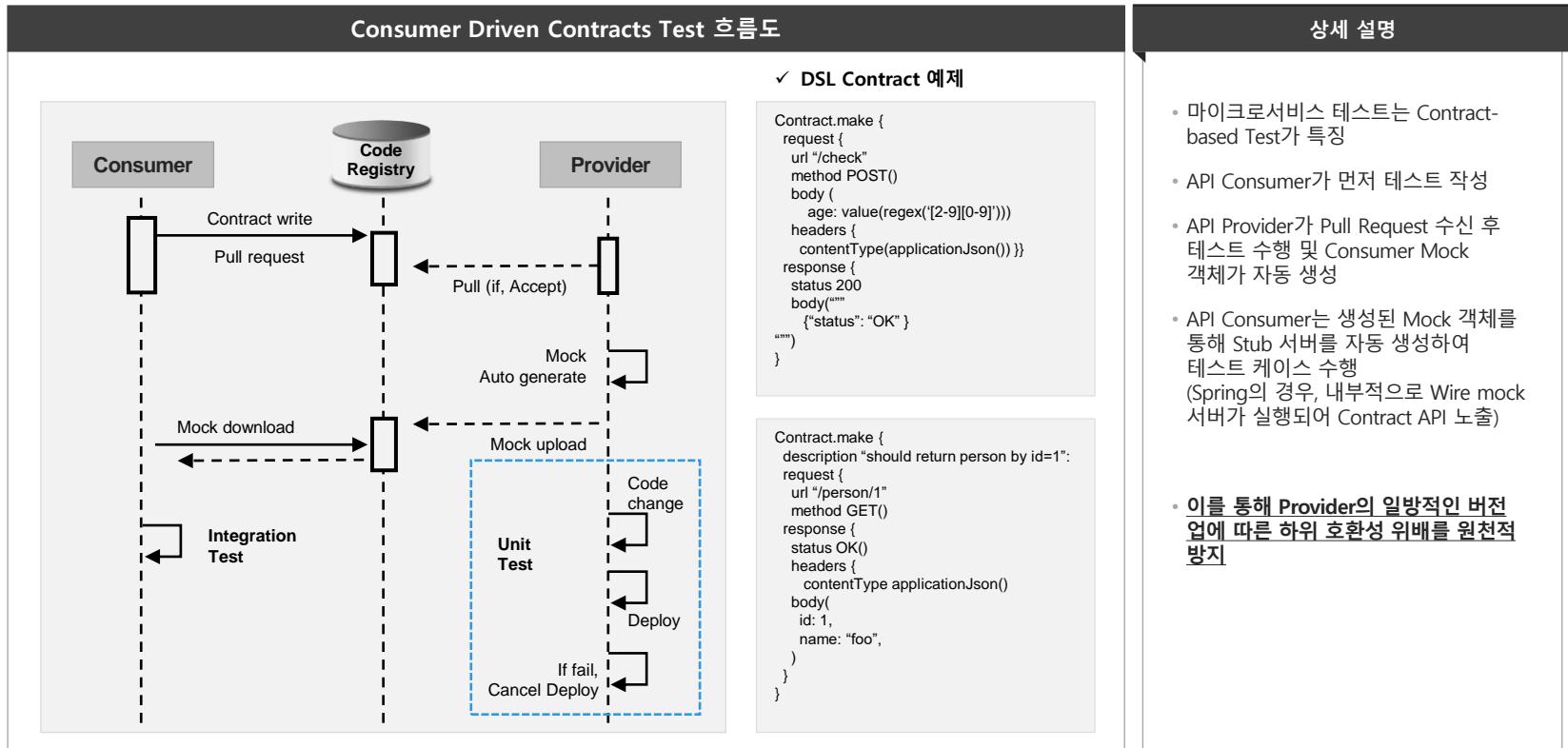
비기능 항목		품질지수 기준	비기능 성능지표 측정 방안	목표
가용성 (Availability)	가용률	가용률 측정	<ul style="list-style-type: none">컨테이너 가용률(URL 접근성) 측정측정 대상 : 컨테이너 기반의 어플리케이션측정 기간 : 7 일 X 24시간	<ul style="list-style-type: none">99.999% 이상 (Five Nines)
응답성 (Responsiveness)	응답시간	응답시간 측정	<ul style="list-style-type: none">컨테이너 배포 시, 평균 응답시간 측정	<ul style="list-style-type: none">평균 3초 이내
확장성 (Scalability)	확장성	리소스 확장 확인	<ul style="list-style-type: none">컨테이너 자동/수동 확장 및 부하분산 처리 여부 확인	<ul style="list-style-type: none">정상 동작
		확장요청 처리시간	<ul style="list-style-type: none">컨테이너 확장요청 식별 시점부터 확장이 완료되기까지 소요된 시간 측정	<ul style="list-style-type: none">평균 1분 이내
신뢰성 (Reliability)	서비스 회복시간	서비스 회복시간 측정	<ul style="list-style-type: none">웹서버 장애(삭제) 발생 시, 서비스 회복 시간 측정	<ul style="list-style-type: none">평균 1분 이내
			<ul style="list-style-type: none">DB 백업 파일 복구를 통한 평균 서비스 회복시간 측정	<ul style="list-style-type: none">평균 10분 이내
	백업 준수율	백업 및 복구 기능	<ul style="list-style-type: none">백업 및 복구 기능의 정상 동작 여부 확인	<ul style="list-style-type: none">정상 동작



MSA 테스트 프로세스



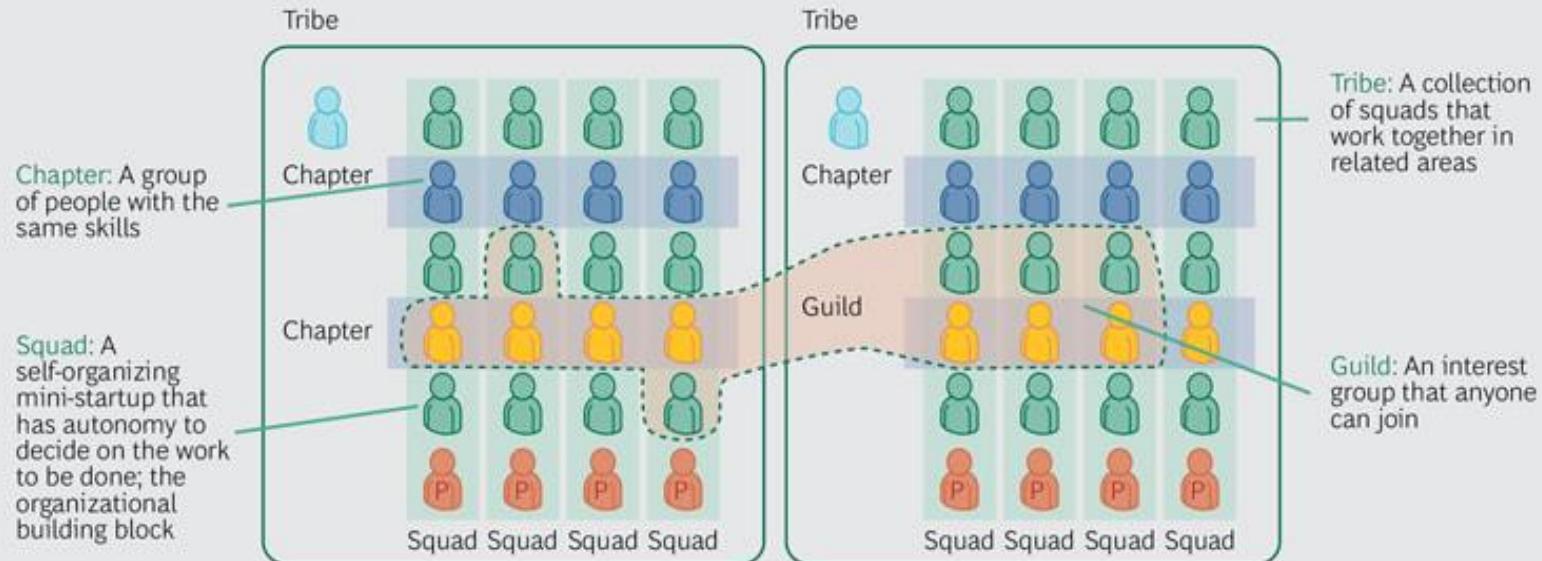
MSA 계약 테스트 방안



MSA 단위 기능 테스트 방안

단위(Unit) 테스트	서비스(통합) 테스트	UI(E2E) 테스트
<ul style="list-style-type: none"> 소스코드의 특정 모듈이 의도된 대로 작동하는지에 대한 검사로 MSA에선 테스트 더블(Mock 객체)을 활용하여 테스트 <p>테스트 더블이란? - '대역'을 의미</p> <pre> graph TD TD[Test Double] --> DO[Dummy Object] TD --> TS[Test Stub] TD --> TSpy[Test Spy] TD --> MO[Mock Object] TD --> FO[Fake Object] </pre> <p>출처 : http://xunitpatterns.com/TestDouble.html</p>	<ul style="list-style-type: none"> 배포 완료된 마이크로서비스에 대해 수행 단위 테스트의 상위 레이어에 위치하며, 구체적인 로직보다는 실제 네트워크 호출이나 데이터베이스 호출을 포함 <p>서비스 테스트 전략의 일환으로, 모든 서비스를 테스트 하는 대신, 서비스간 통신을 담당하는 각 어댑터를 테스트 하는 방안도 고려</p>	<ul style="list-style-type: none"> 시스템의 모든 컴포넌트가 예정된 방식으로 잘 동작하는지를 확인하는 테스트 모든 서비스에 존재하는 엔드 포인트에 대한 부하 및 성능 테스트도 고려 거의 수행하지 않으나, BDD(Behaviour-Driven Development)를 기반으로 레코딩된 테스트를 수행하는 방안도 고려 <p>BDD(Behaviour-Driven Development)란? - TDD와 유사하나, TDD는 테스트 자체에 집중하여 개발하는 반면, BDD는 비즈니스 요구사항에 집중하여 테스트 케이스를 개발</p>

조직 전환 방안 – TO-BE



Source: Spotify

사례1 – ING BANK

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

Principles

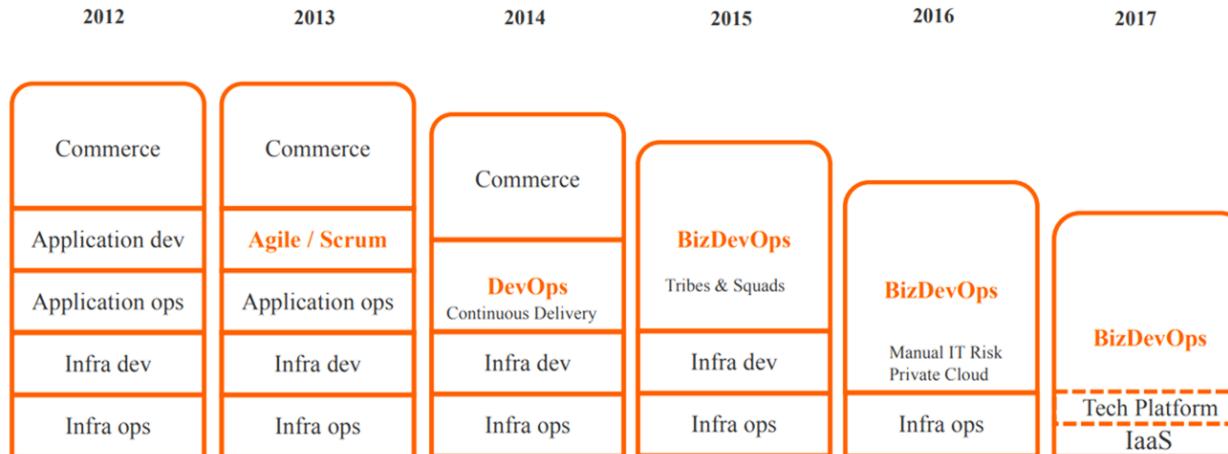
Inspired by the Agile methodology, eight principles are at the heart of our Way of Working:

1. We work in high **performing** teams
2. We **empower** teams
3. We care about talent and **craftmanship**
4. We continuously **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



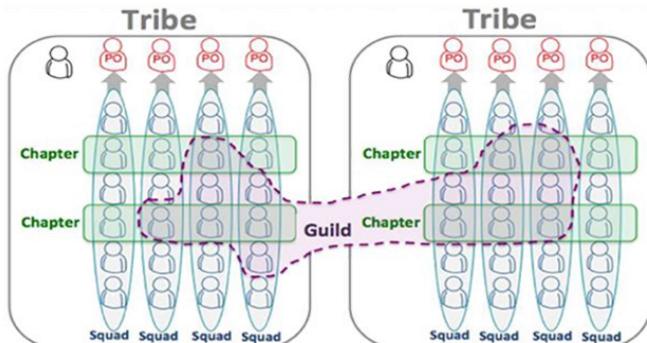
Engineer: From single discipline to **full stack engineers**: designing, coding, test engineering, infra engineering, etc

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 differentiating role: **skill and competency coach**.

사례1 – ING BANK

And this is how we organize ourselves

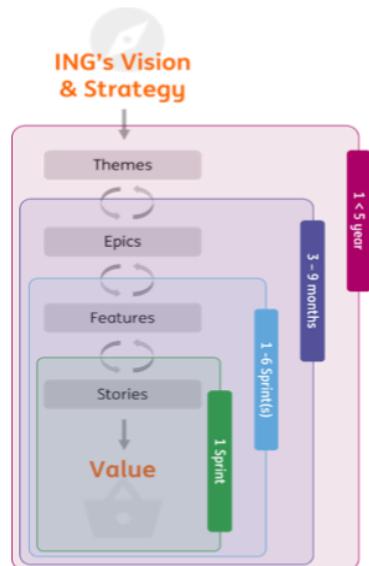


- **Squads**, are high performing “stable” teams who provide the execution power to the One WoW
- A **Tribe** is a collection of Squads organized around the same purpose
- **Chapters** are formal “groups” of members with the same background / expertise deciding on how things need to be done within a 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

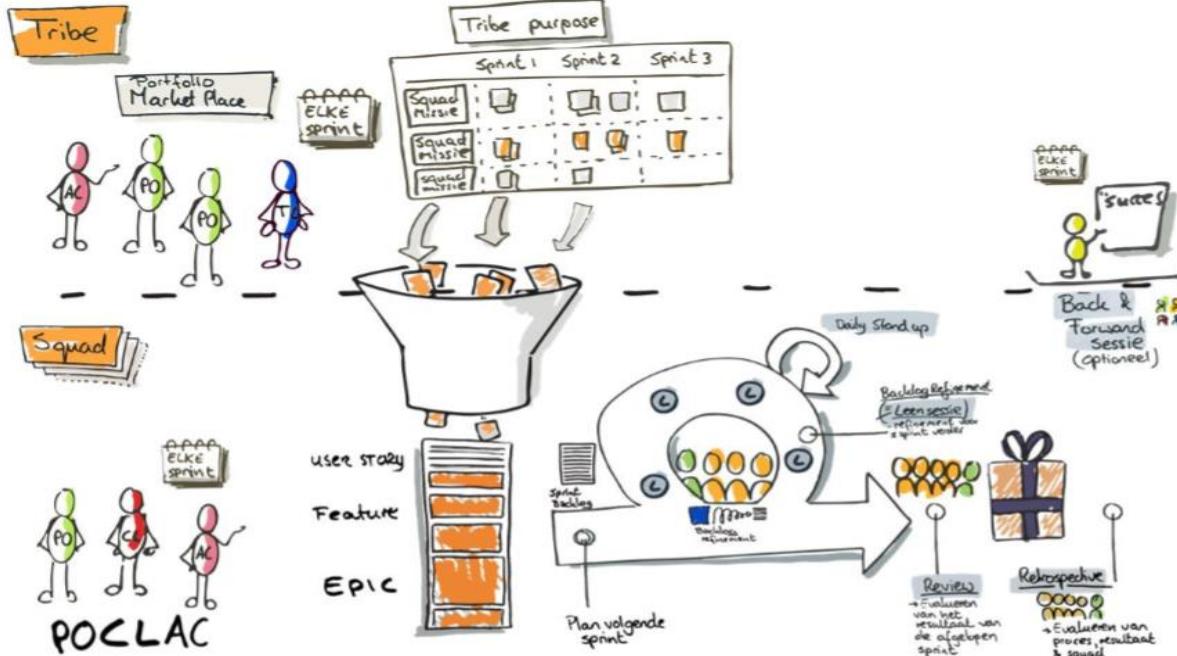


- A **Theme** connects the dots between Strategy and execution. Hence, a Theme represents an (investment) area that supports the strategic vision. Themes are representing one to five year(s).
- A Theme is broken down in multiple **Epics**: large scale bank initiatives that will be delivered in one to 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



THANKS!

Any Question?

You can find me at:

jyjang@uengine.org

<https://github.com/jinyoung>

<https://github.com/TheOpenCloudEngine>

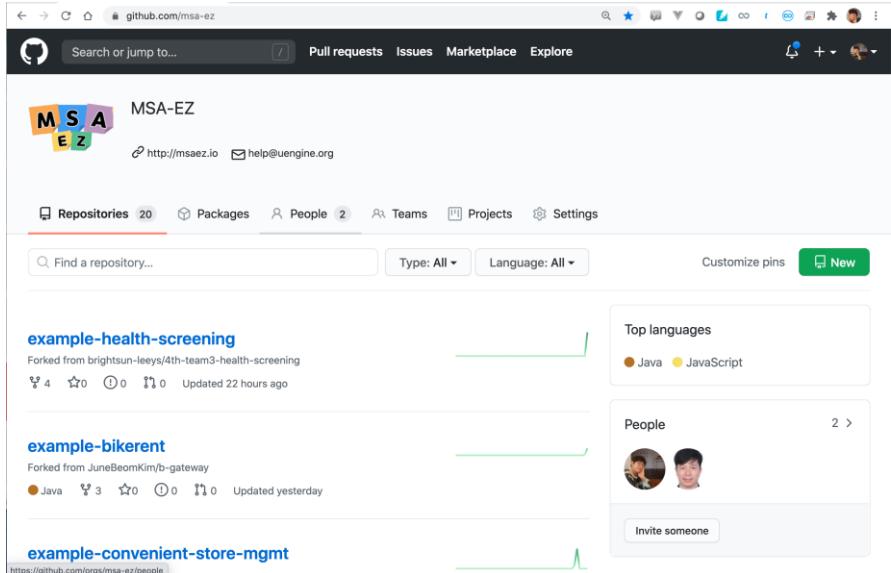
Recommended Books

- **Overall MSA Design patterns:**
<https://www.manning.com/books/microservices-patterns>
- **Microservice decomposition strategy:**
 - DDD distilled: <https://www.oreilly.com/library/view/domain-driven-design-distilled/9780134434964/>
 - Event Storming: https://leanpub.com/introducing_eventstorming
- **API design and REST:**
<http://pepa.holla.cz/wp-cont.../2016/01/REST-in-Practice.pdf>
- **Database Design in MSA:**
 - Lightly:
 - [https://www.confluent.io/wp-content/uploads/2016/08/Making Sense of Stream Processing Confluent 1.pdf](https://www.confluent.io/wp-content/uploads/2016/08/Making_Sense_of_Stream_Processing_Confluent_1.pdf)
 - Deep dive:
 - https://dataintensive.net/?fbclid=IwAR3OSWkhqRjLI9gBoMpbsk-QGxeLpTYVXIJVCSaw_A5eYrBDc0piKSm4pMM

github.com/msa-ez

Reference MSA Projects

github.com/msa-ez



The screenshot shows the GitHub organization page for `github.com/msa-ez`. The page features a header with the organization logo (MSA EZ), a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. Below the header, there's a banner with the URL `http://msaez.io` and an email address `help@uengine.org`. The main content area displays three repository cards:

- example-health-screening**: Forked from `brightsun-leeyi/4th-team3-health-screening`. It has 4 stars, 0 forks, 0 issues, 0 pull requests, and was updated 22 hours ago.
- example-bikernet**: Forked from `JuneBeomKim/b-gateway`. It has 3 stars, 0 forks, 0 issues, 0 pull requests, and was updated yesterday.
- example-convenient-store-mgmt**: A link to the organization's people page: <https://github.com/orgs/msa-ez/people>.

On the right side, there are sections for "Top languages" (Java, JavaScript) and "People" (two user icons). A "Customize pins" button and a "New" button are also visible.

MSA School

msaschool.io

The screenshot shows the 'MSA School 소개' (MSA School Introduction) page. The header includes the site's logo and navigation links for 소개, 계획단계, 설계/구현/운영단계, 참고자료, 커뮤니티 및 교육, and //Engine. The main content area features a large image of a person working on a laptop, overlaid with text: '험난한 MSA 구축 여정의 길라잡이'. To the right of the image is a circular diagram divided into six segments labeled 1단계 (분석), 2단계 (설계), 3단계 (구현), 4단계 (통합), 5단계 (배포), and 6단계 (운영). Below the diagram, the text 'Biz Dev Ops' is prominently displayed. On the left side of the main content area, there is a sidebar with links to MSA School 소개, 예제 도메인, 사용 플랫폼, 예제 애플리케이션 둘러보기, 관련 리소스, 유필리티 및 도구. At the bottom of the page, there is a section titled 'MSA 스쿨에 오신 것을 환영합니다.' followed by a paragraph of text and a note: 'MSA School은 BizDevOps의 전 과정에 걸친 End-to-End 학습을 위한 단계별 실천법 제공입니다.'