

Developing Cloud-native applications

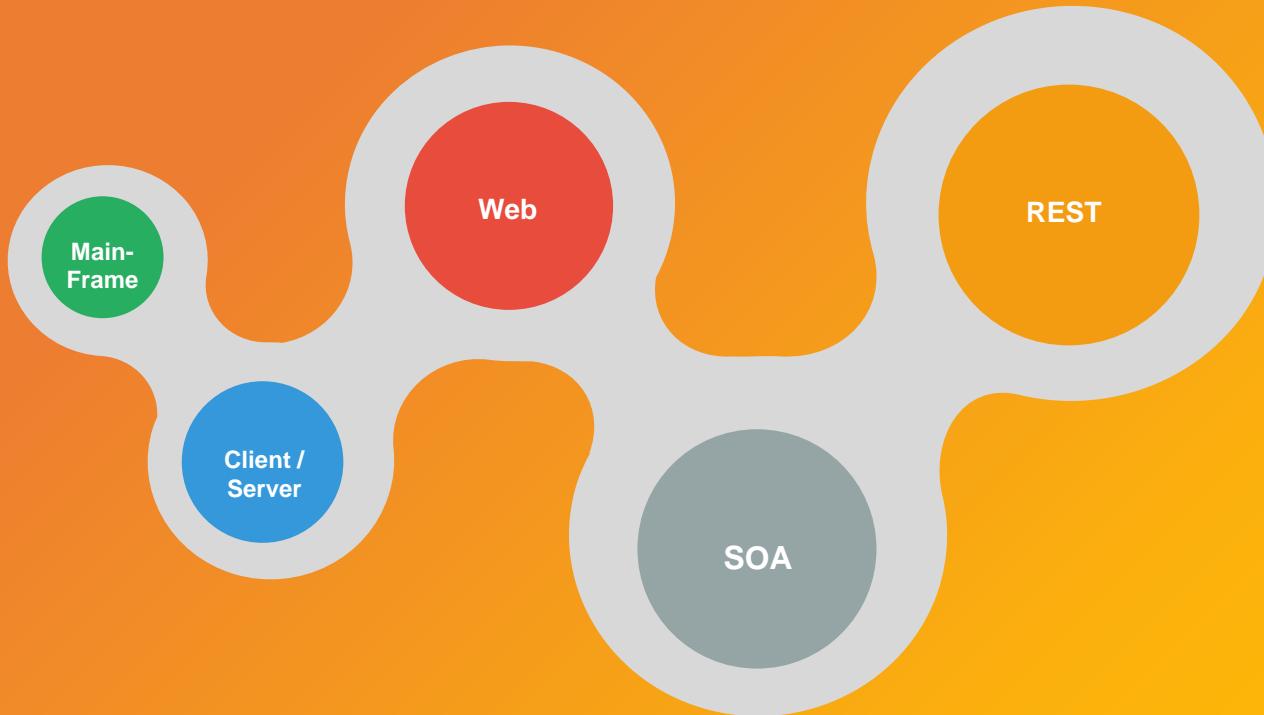
2nd Jan 2023, ver2.0



Table of contents

- Cloud Native Application Definition & Design Strategies ✓
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

소프트웨어 아키텍처의 성장 여정



AS-IS: Pain-points

A 사 의료분야 SaaS 운영

- 서비스 업그레이드가 수시로 요청이 들어와 거의 매일 야근 중. 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, 이로 인해 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 높은 편
- 기존 모놀로직 아키텍처의 한계로 장기적인 발전의 한계에 봉착

B 사 제조분야 SaaS 운영

Agile Delivery

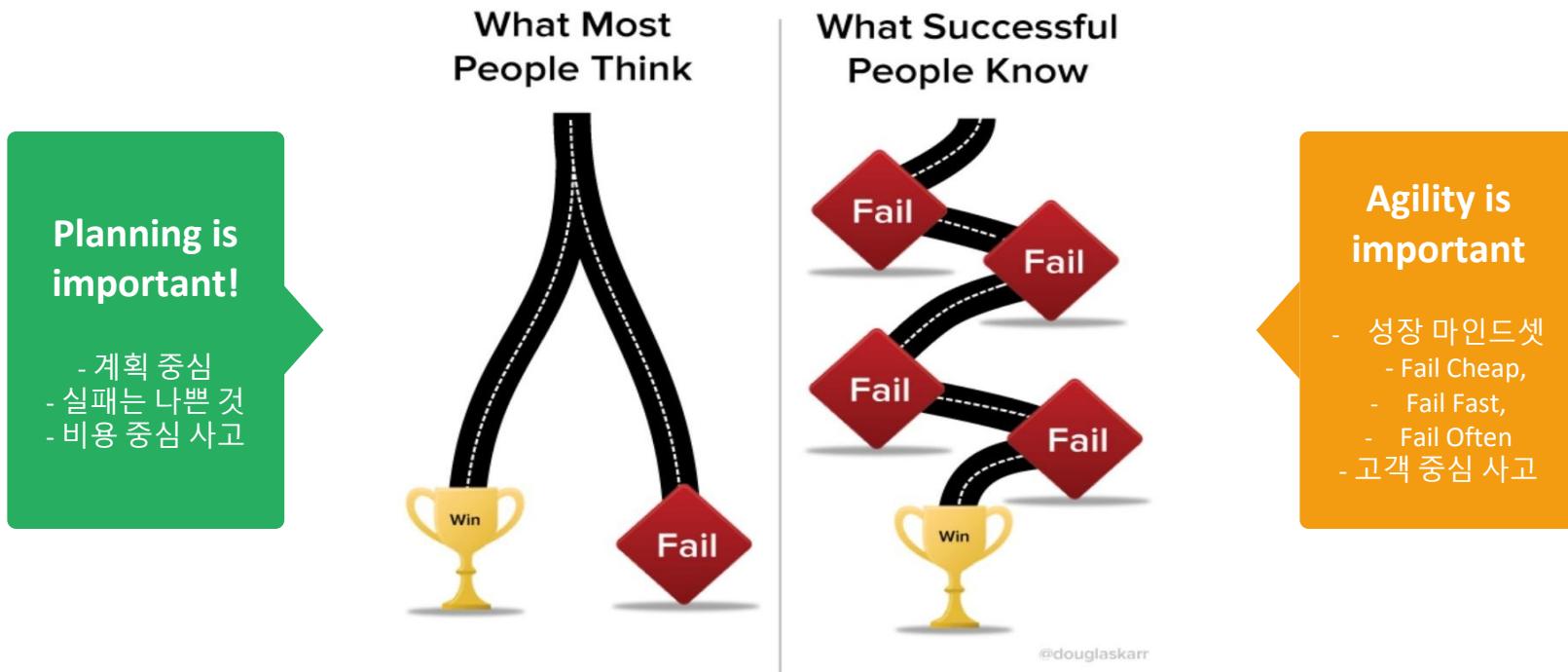
Amazon, Google, Netflix, Facebook, Twitter는 얼마나 자주 배포할까요?



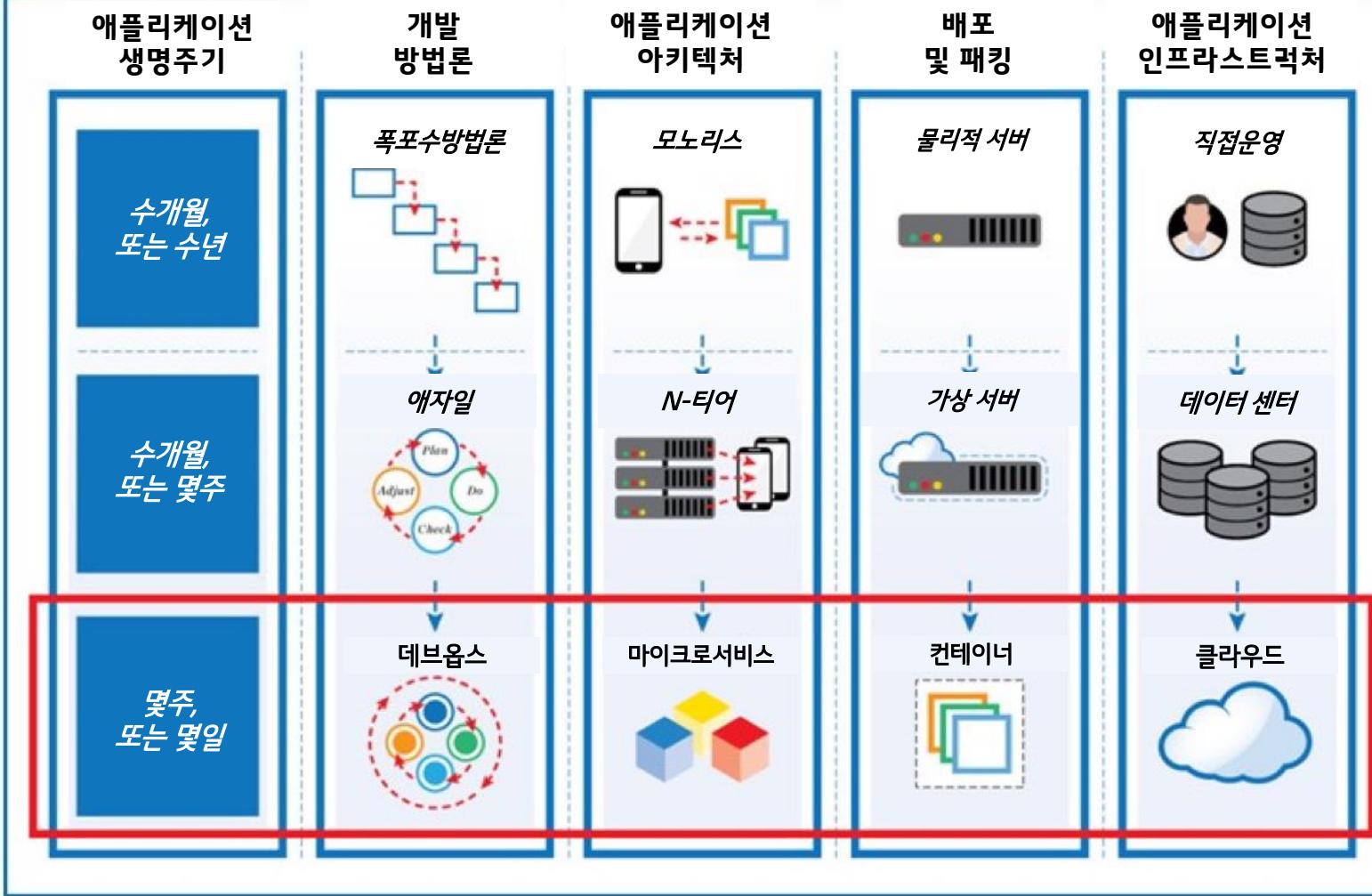
Company	배포 주기 Deploy Frequency	배포 지연 시간 Deploy Lead Time	안정성 Reliability	고객 요구 응답성 Customer Responsiveness
아마존	23,000 / 일	몇 분 (Minutes)	높음	높음
구글	5,500 / 일	몇 분 (Minutes)	높음	높음
넷플릭스	500 / 일	몇 분 (Minutes)	높음	높음
페이스북	1 / 일	몇시간 (Hours)	높음	높음
트위터	3 / 주	몇시간 (Hours)	높음	높음
일반회사	9개월 주기	수개월~수분기별	낮음 / 보통	낮음 / 보통

출처: 도서 The Phoenix Project

Agile 의 정의



애자일에 필요한 것들



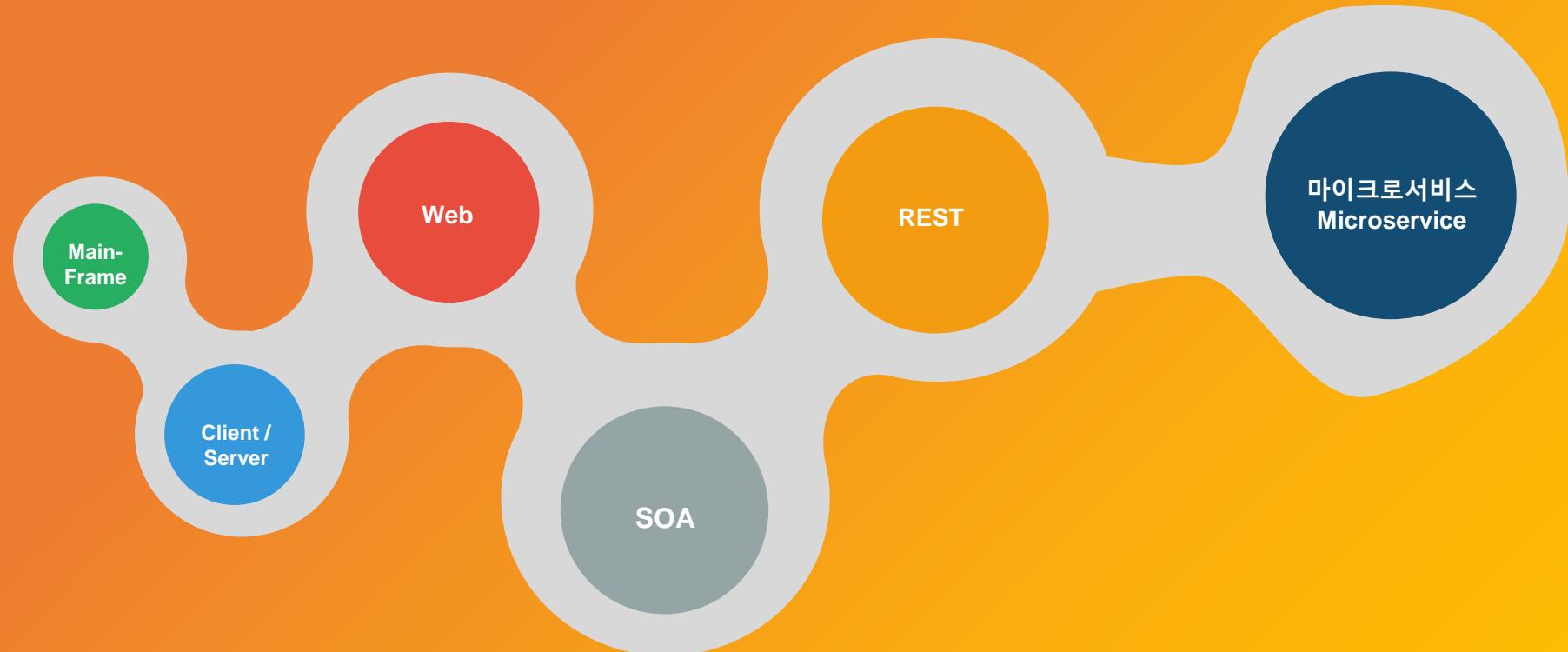
제프 베조스의 의무사항

”



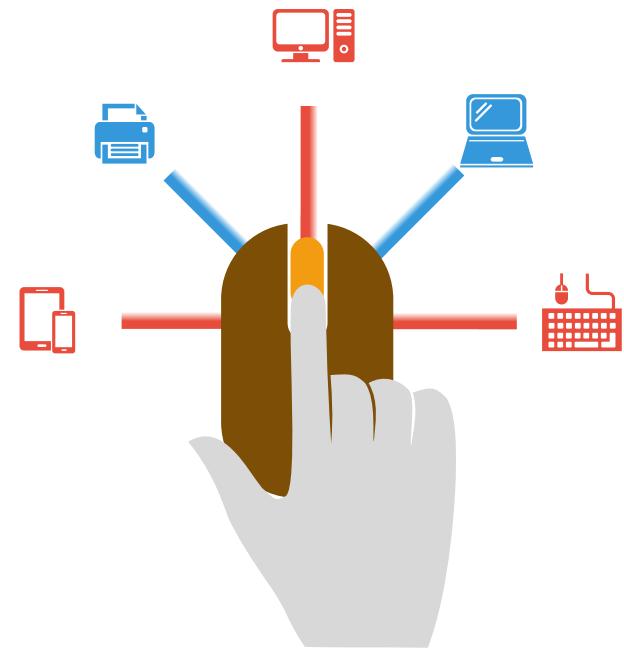
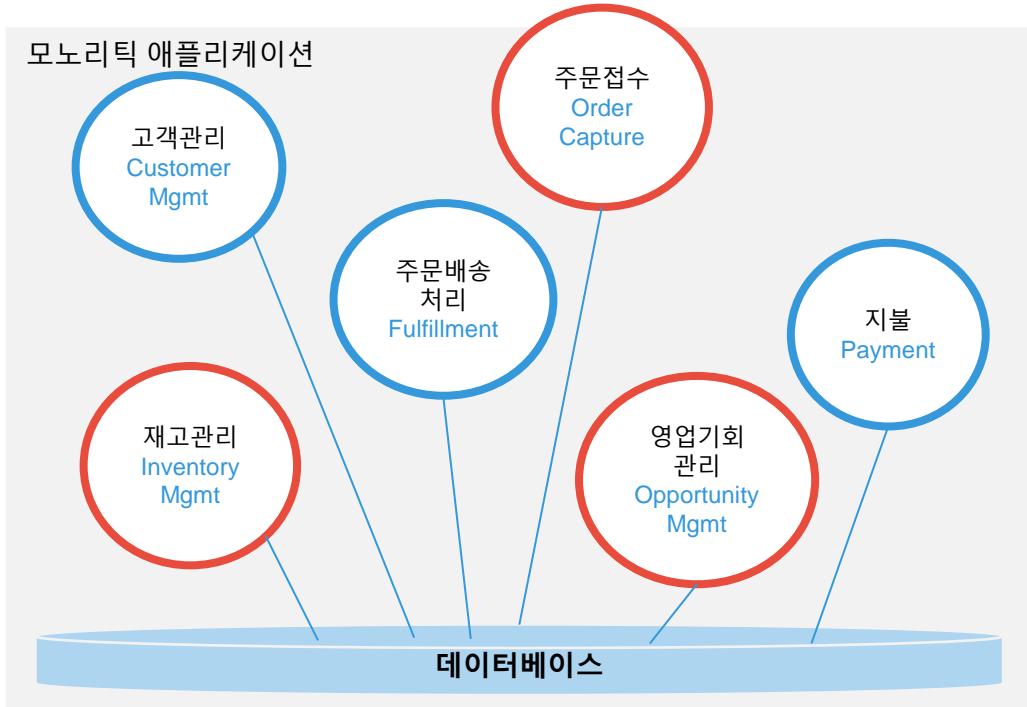
1. All teams will henceforth expose their data andfunctionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. 다음과 같은 그 어떠한 직접적 서비스간의 연동은 허용하지 않겠다:
no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors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

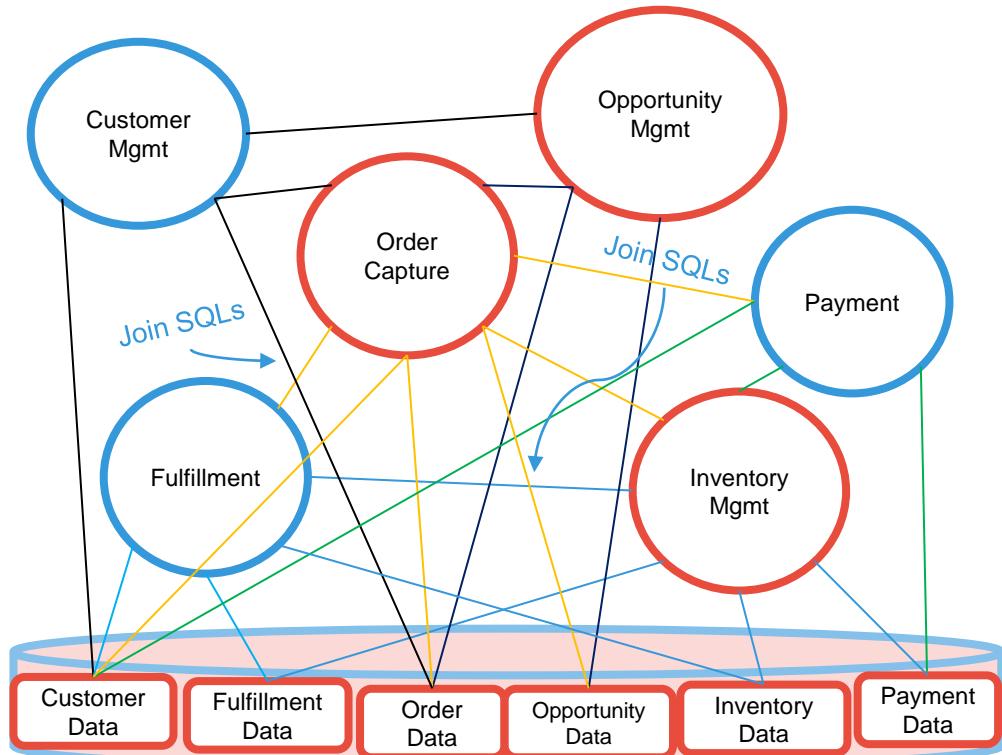
An Enterprise Application or Suite



모노리티 아키텍처의 분석

상호 데이터 참조가 용이하여 빨리 개발하기 위한 초기 아키텍처로 적합

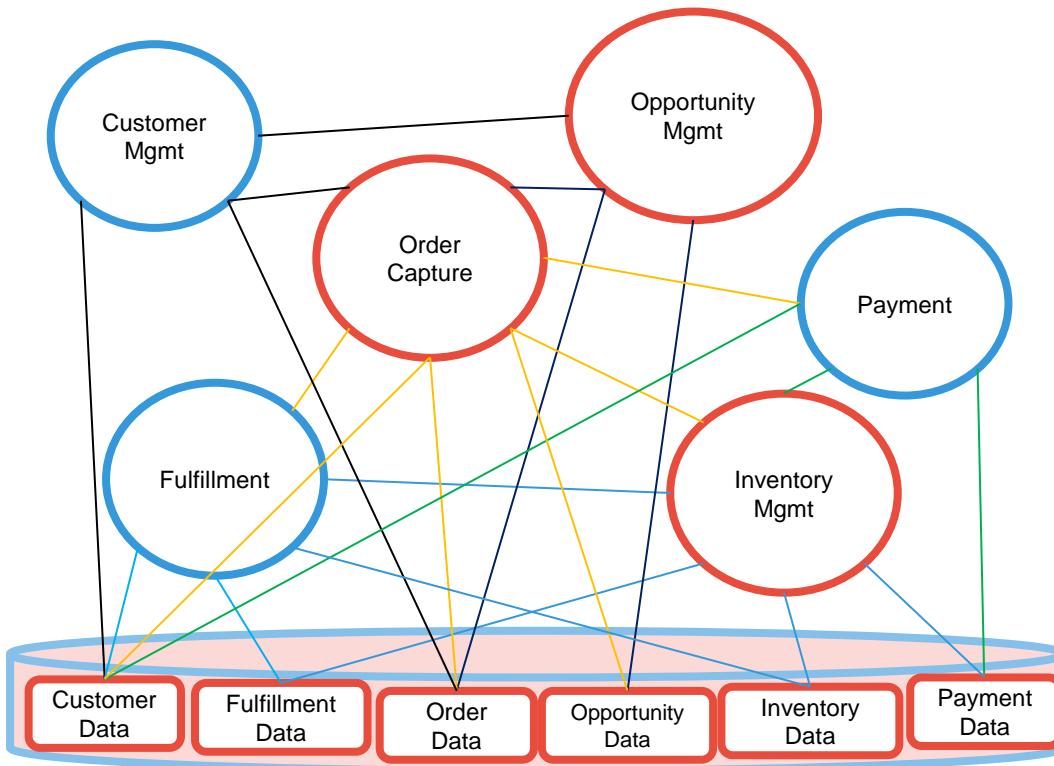
- 그러나, 쉬운 상호연동은 상호의존성을 높힘
But, ease of interaction results in many inter-dependencies
- 시간이 갈수록, 커플링(의존성)은 강해지고 강해짐
Over time, coupling becomes tighter and tighter
- 하나의 컴포넌트를 수정하는 일은...
e.g. Order Data Table 의 field 변경
- 많은 다른 컴포넌트의 수정을 동반하게 됨
e.g. 다른 모듈의 Join SQL 들



모노리틱 아키텍처의 단점

Drawbacks of a Monolithic Architecture

Size matters, costs grow as they grow

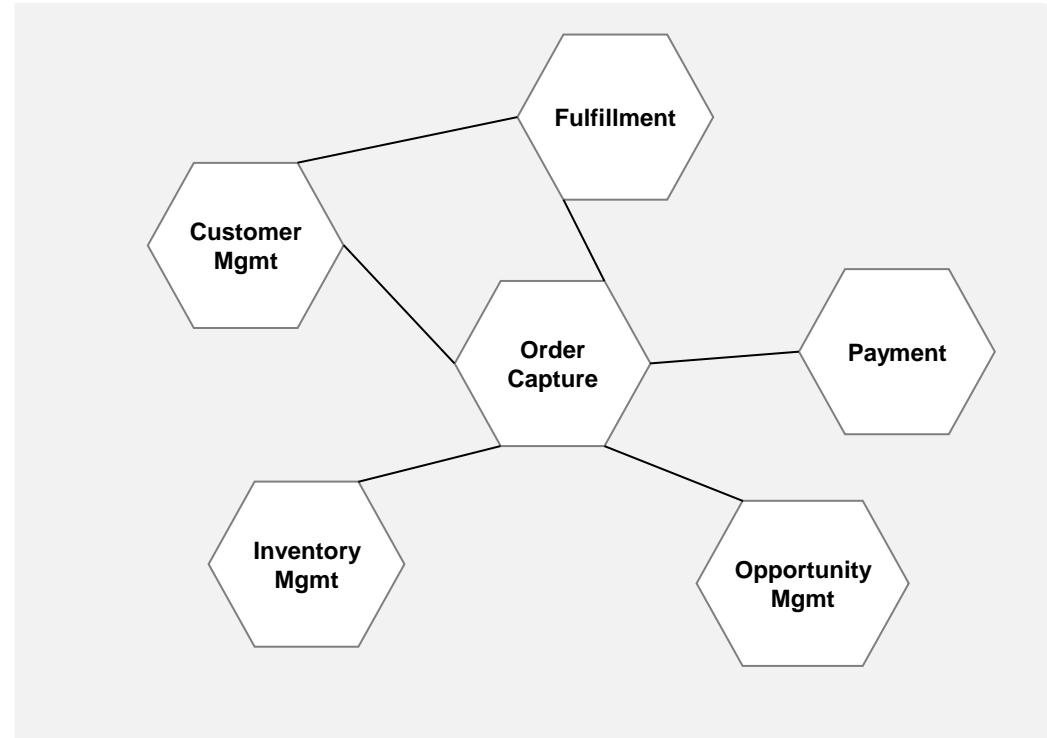


- 코드량이 방대함 (오류발생시 바닷가에서 바늘 찾기)
Large code base
- 개발환경이 무거움 (IDE, WAS 등)
**Overloaded IDE and development environment
(Web container, etc.)**
- 하나의 변경이 나머지의 모든 재배포를 유발함
Deployment of any change requires redeploying everything
- 선별적 확장이 용이하지 않음
Only scales in one dimension
- 하나의 기술 스택 만을 선택 가능 e.g. Java 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

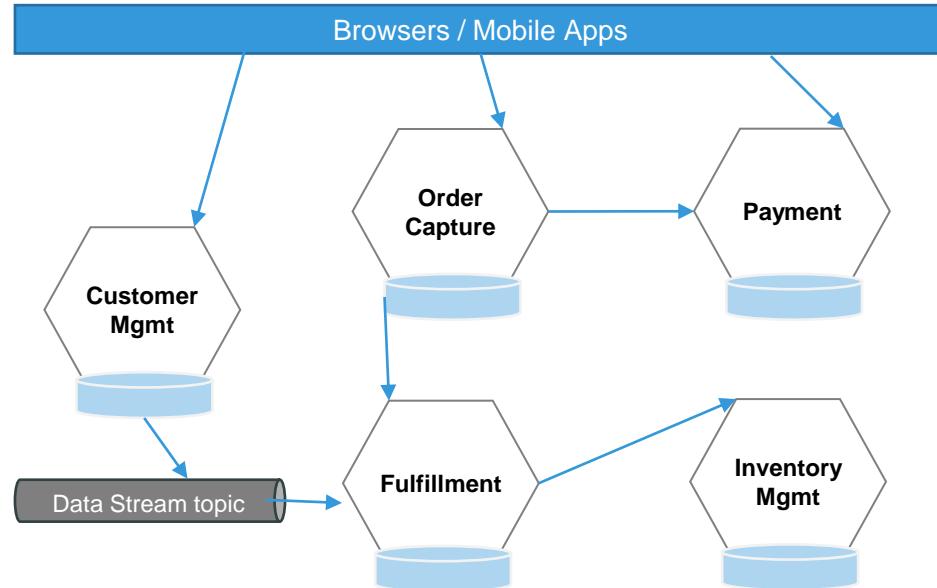
- 모든 기능을 각각의 배포 스택으로 분리
Break each function into separate deployment stacks
- Separate database
- Separate Servers running any technology
- Local or wide-area network



Microservice Architecture

Isolation is the name of the game

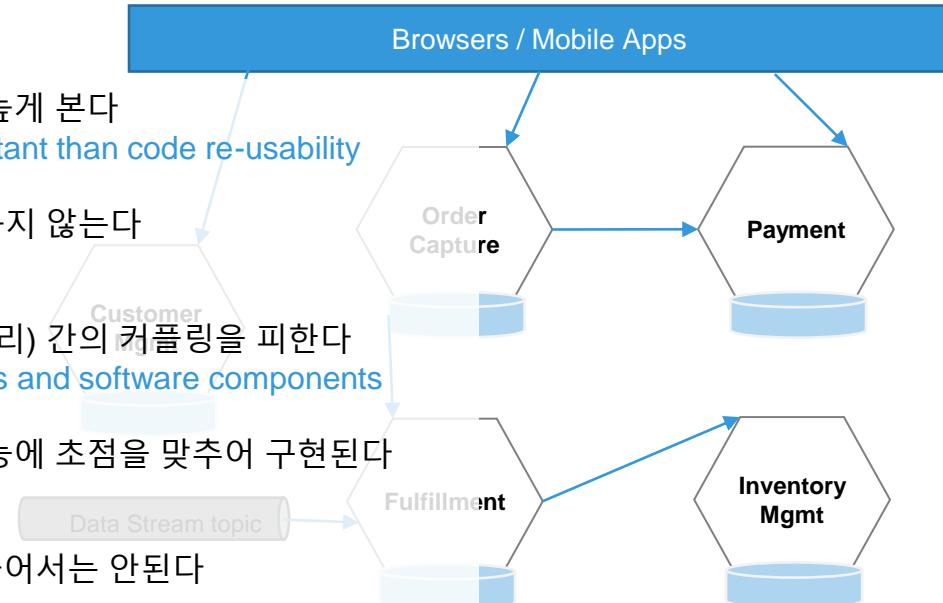
- Service-oriented architecture
- Loosely coupled elements
 - Interaction only through HTTP/REST
 - Or asynchronous streaming / messaging



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크로 서비스는 코드와 데이터를 공유하지 않는다
Microservices should not share code or data
- 불필요한 서비스와 SW 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

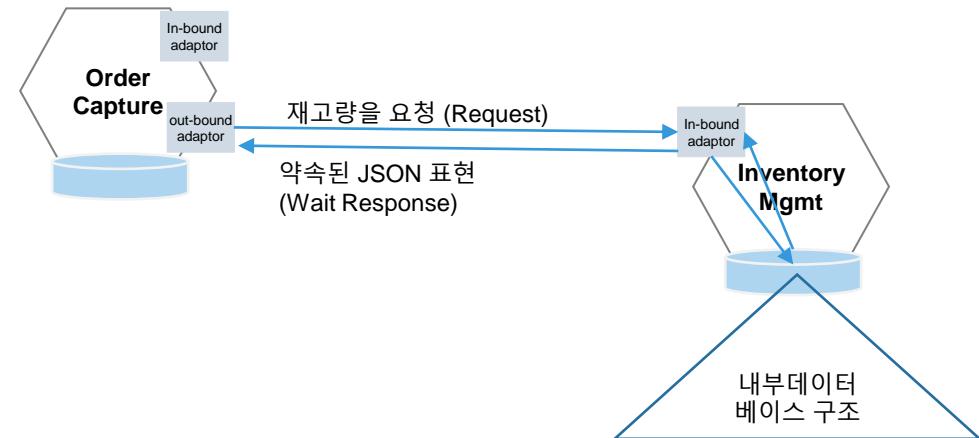


HTTP/REST 를 이용한 약결합 연동 (Loosely-Coupled Interaction via HTTP/REST)

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

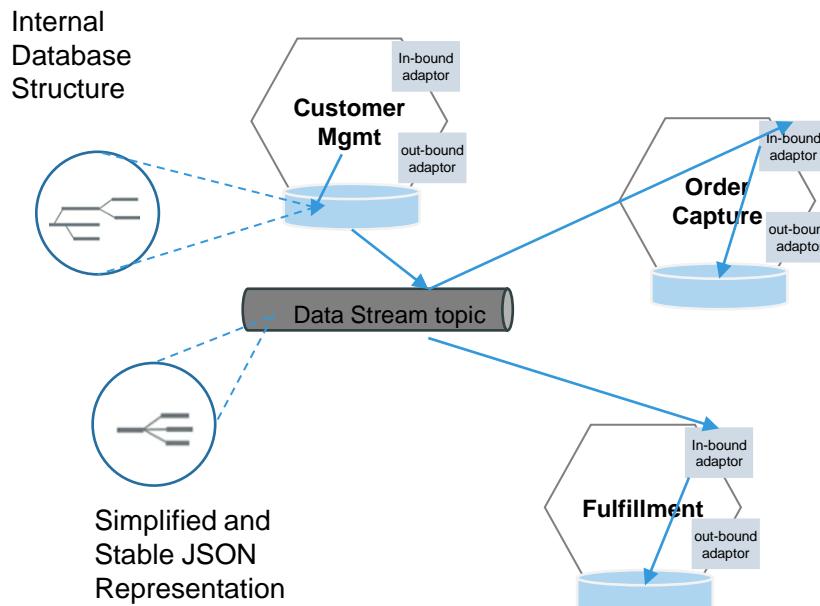
상대의 수신을 기다리지 않는 비동기식 메시징
Asynchronous messaging(streaming)

- 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

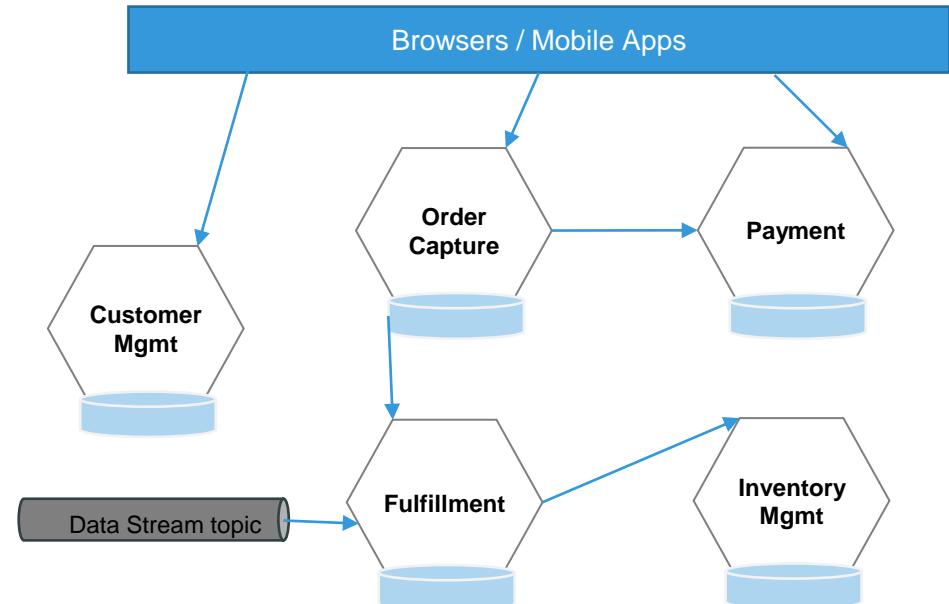
Improved fault isolation

- 혼재된 기술 스택 (신기술)을 사용하기 쉽다

Not committed to one technology stack

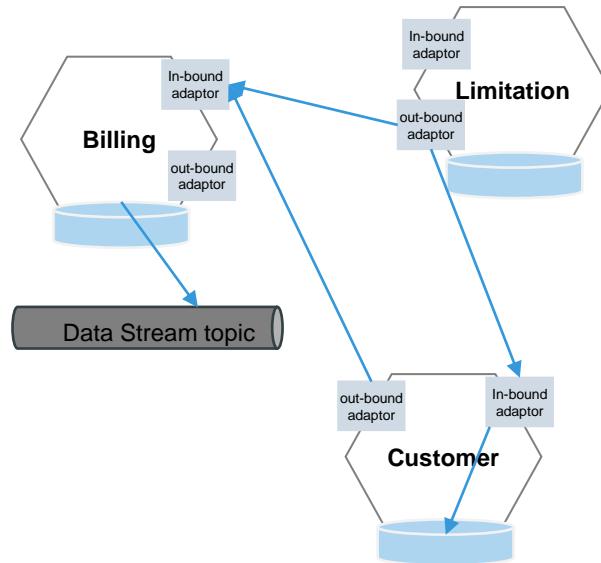
- 개발환경을 구축하기 쉽고 빨리 기동된다

IDE and app startup are faster



Microservice Architecture

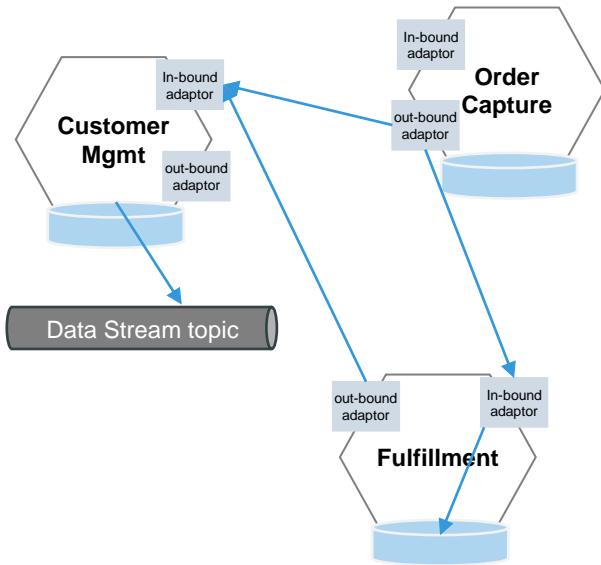
- Updating one service doesn't require changing others
- Ability to upgrade the tech stack (HW/SW/DBMS/NW) Of each service independently



- 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
쿠버네티스 와 같은 DevOps 환경이 필수적이다
Deployment and operations are more complex

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

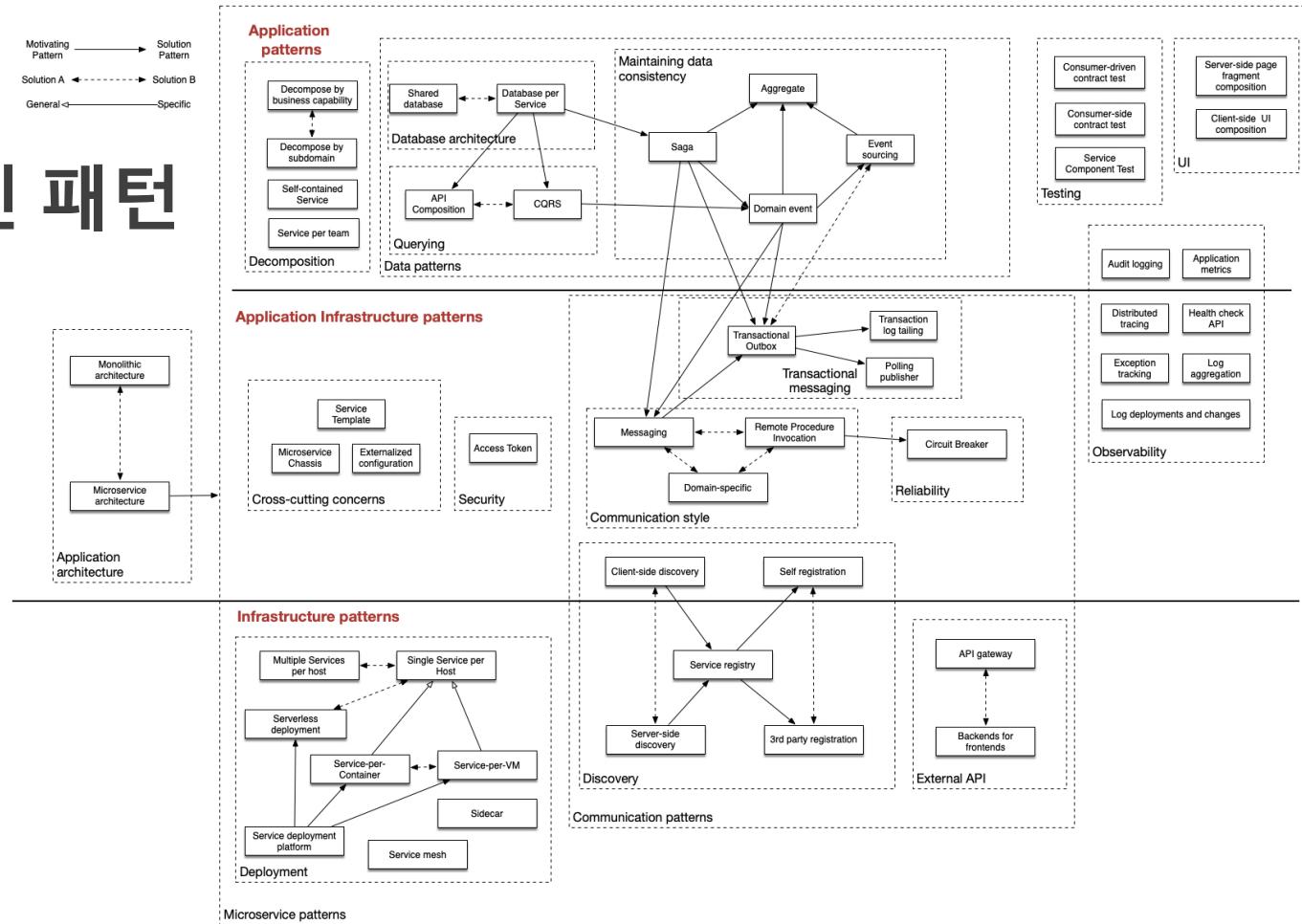
서비스간 연동을 통해서만 구현하는 비용의 상승
Inter-service communication complicates development

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

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

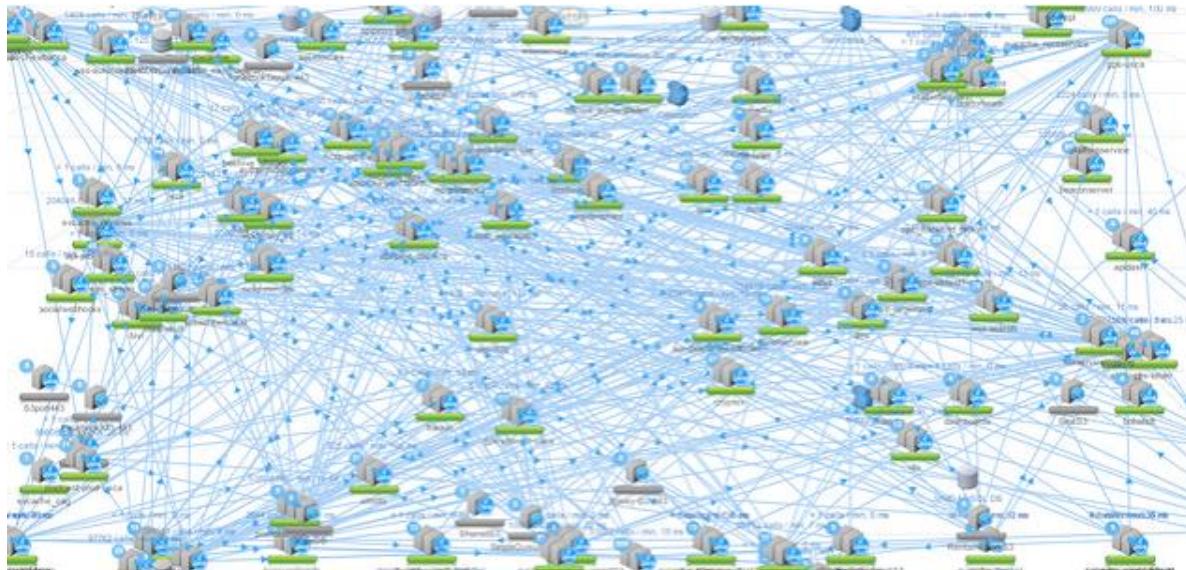
해결책: MSA 디자인 패턴

- microservices.io



넷플릭스 OSS

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



マイクロ 서비스 转换 사례



Mobile Apps and
Serverless Microservices



Pure Play Video OTT - A



Video & Broadcasting



From Monolithic to
Microservices



Gaming Platform



IoT Service

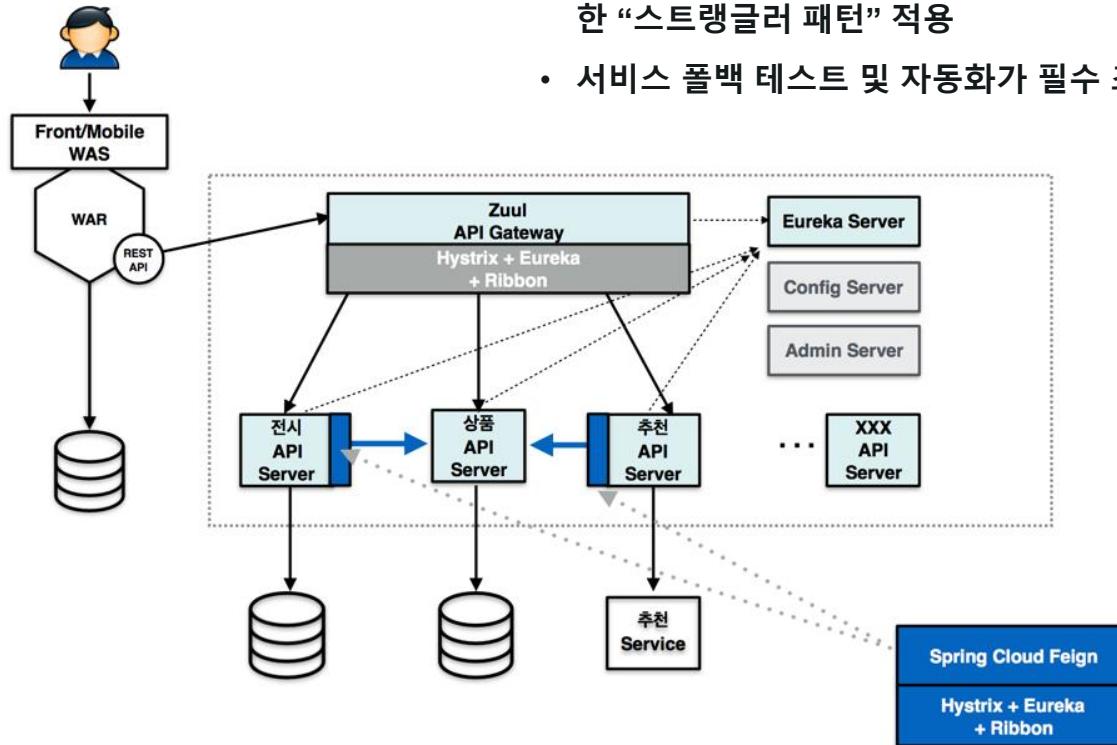


Serverless Microservices

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 contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall ✓
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

Target Domain

: Online Shopping Mall (12 STREET)

- Vision & Mission



Service Resiliency

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



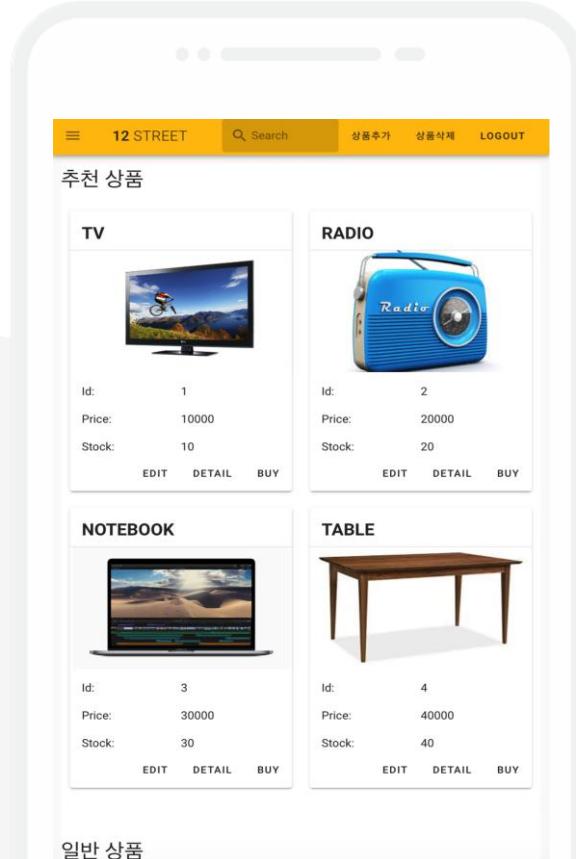
Customer Responsiveness

다양한 고객 기능 요구사항의 탐색과 반영



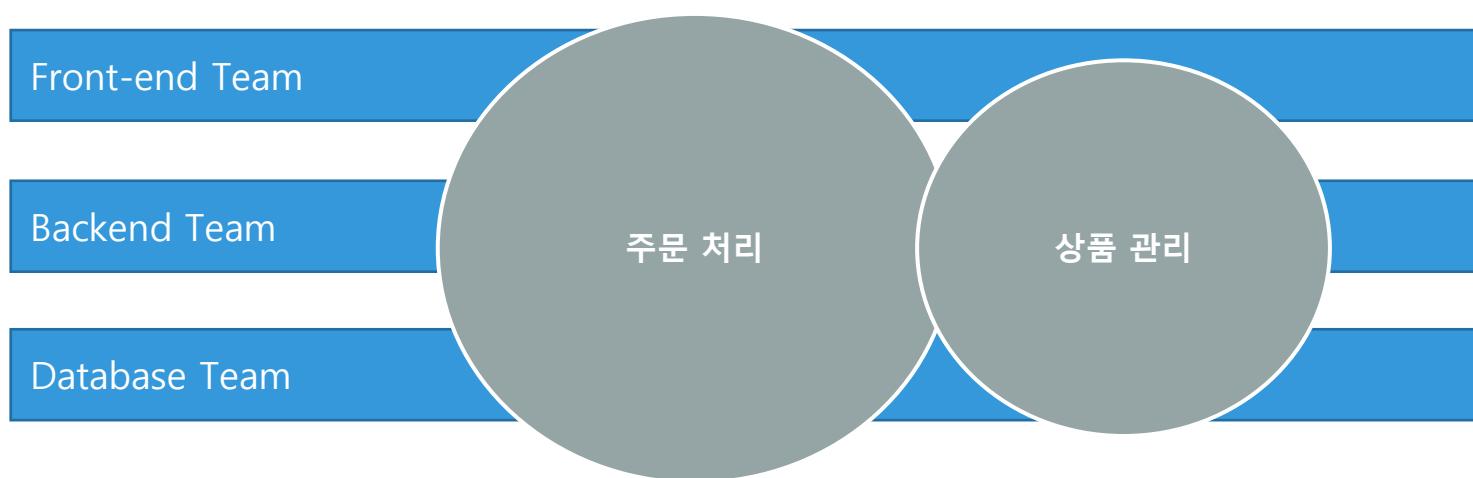
Scalability

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



Organization & KPI Definition - 창업시기

12 Street Team

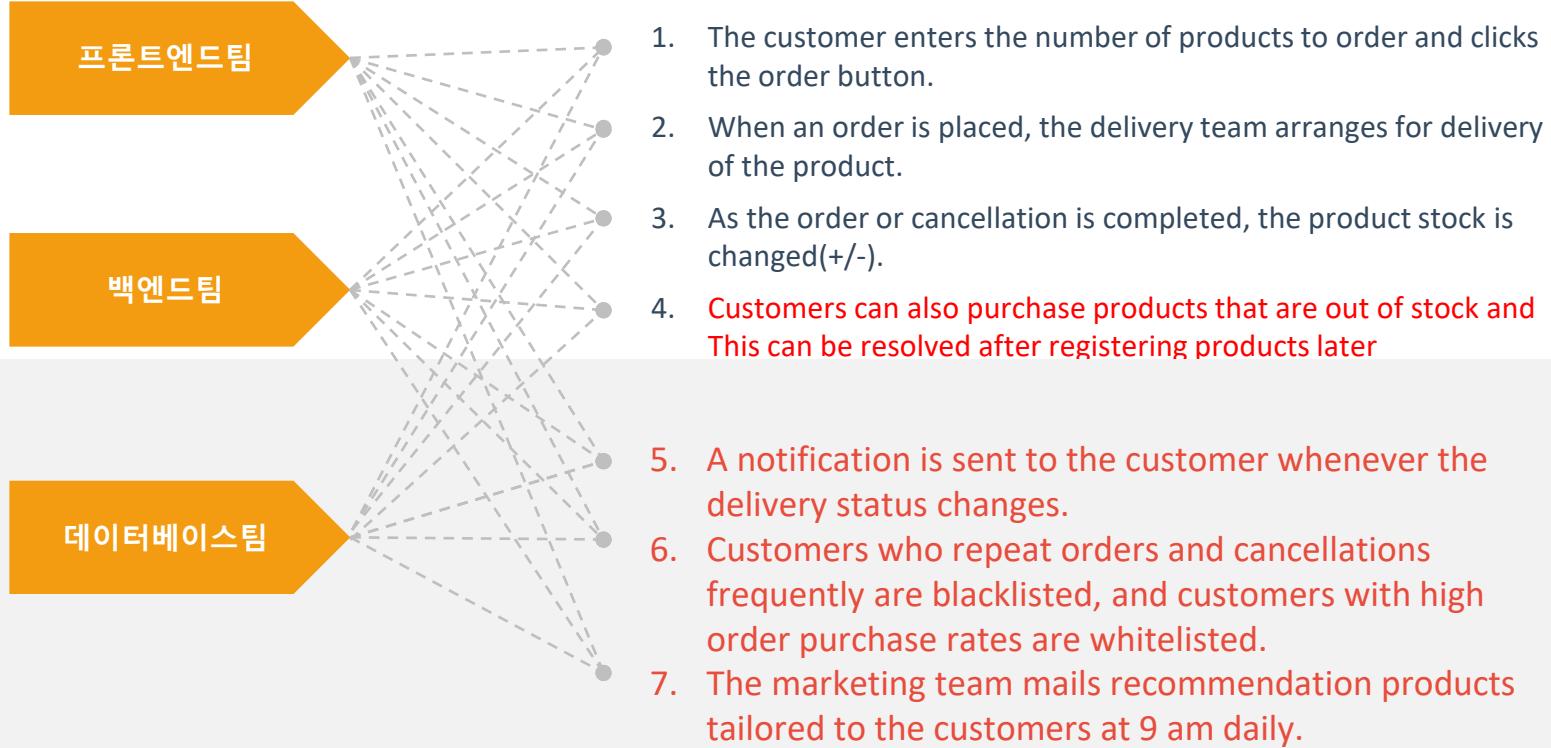


- Multiple Concerns Single Organization → no problem
- Horizontally aligned

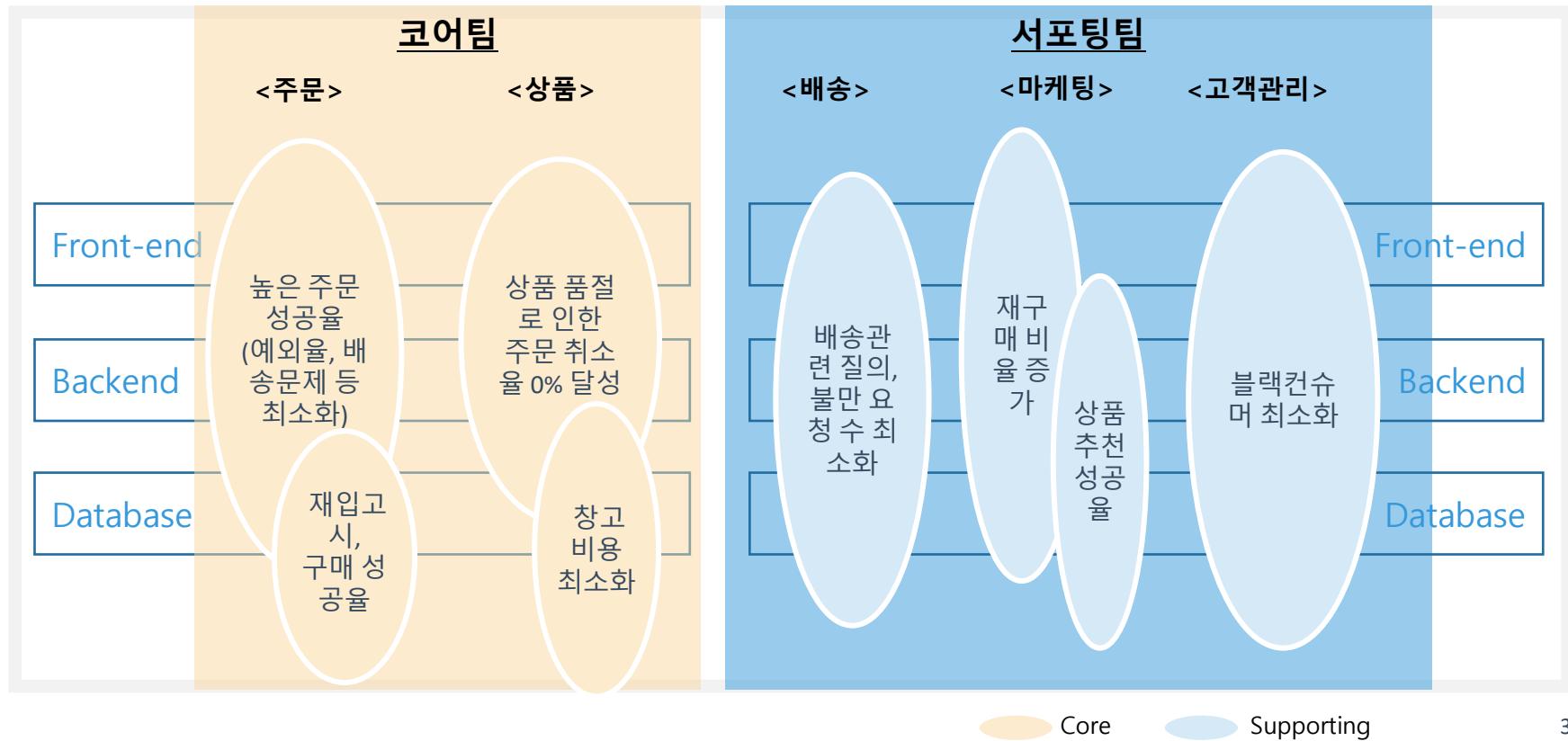
회사의 성장 – 너무 많은 Concerns



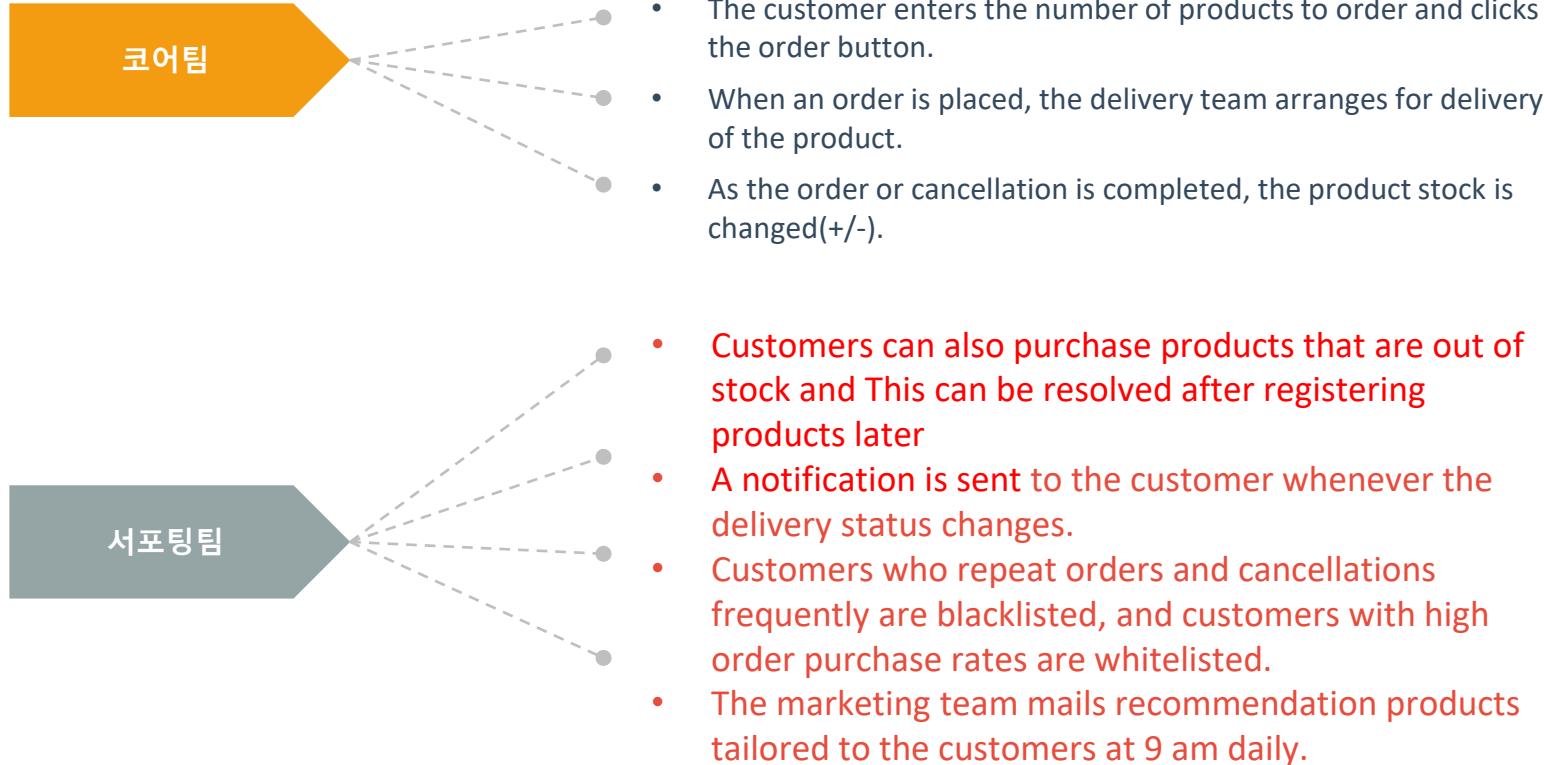
User Stories 와 책임소재 → 너무 많은 Concerns



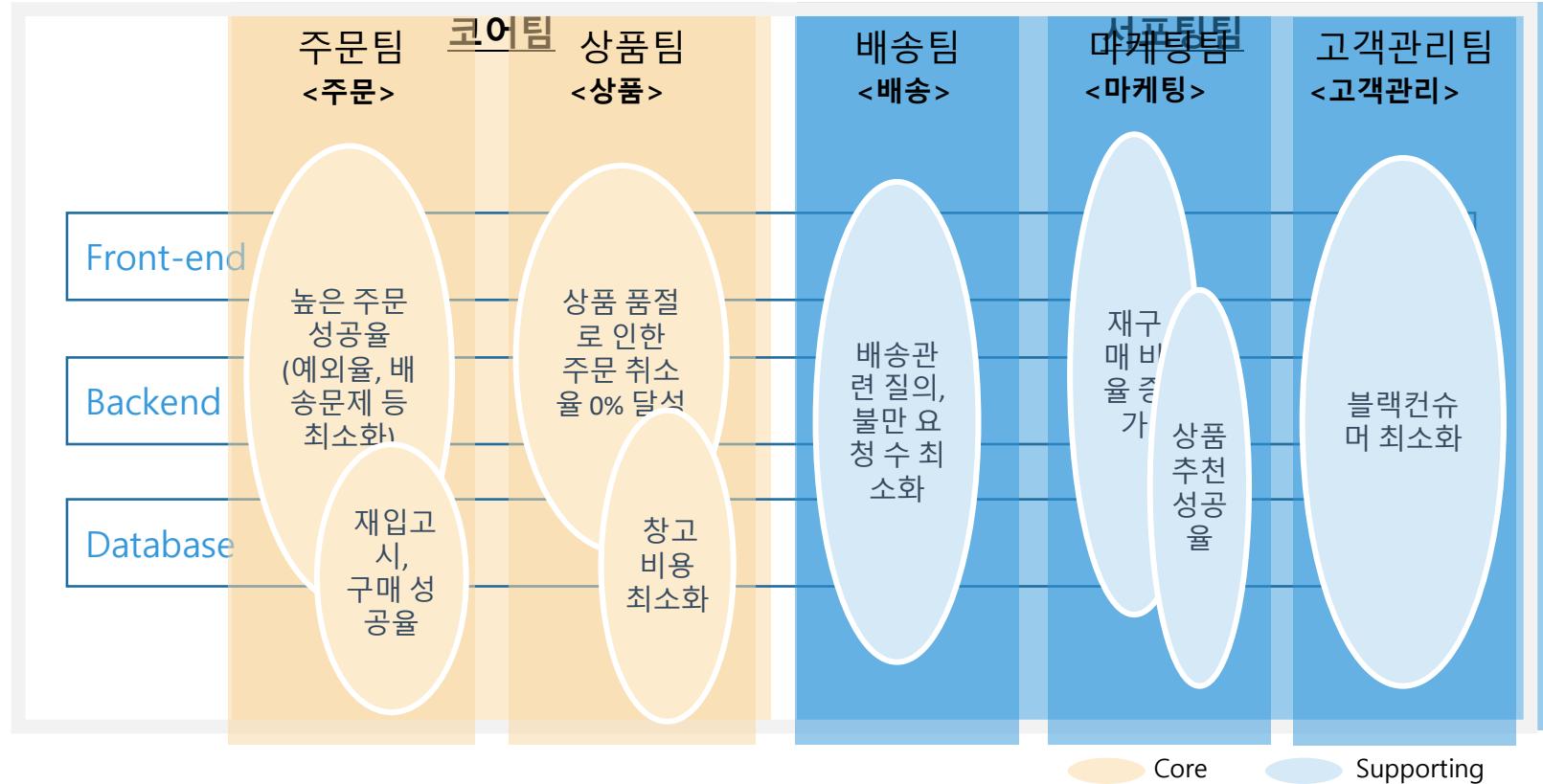
회사의 성장 - Separation of Concerns



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.

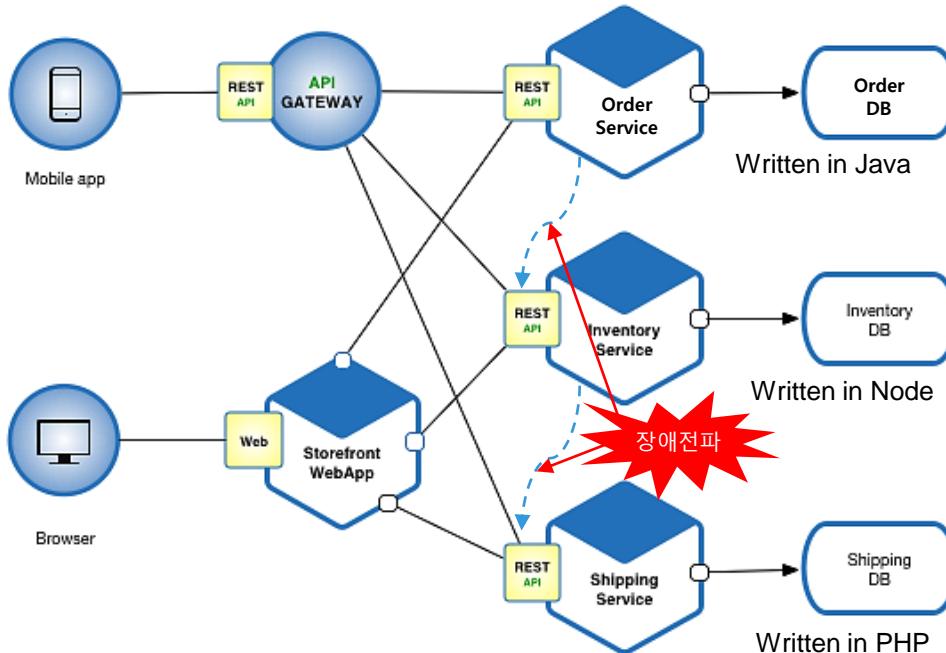
Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview ✓
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

Approach #1 : Micro Service Architecture

Contract based, Polyglot Programming

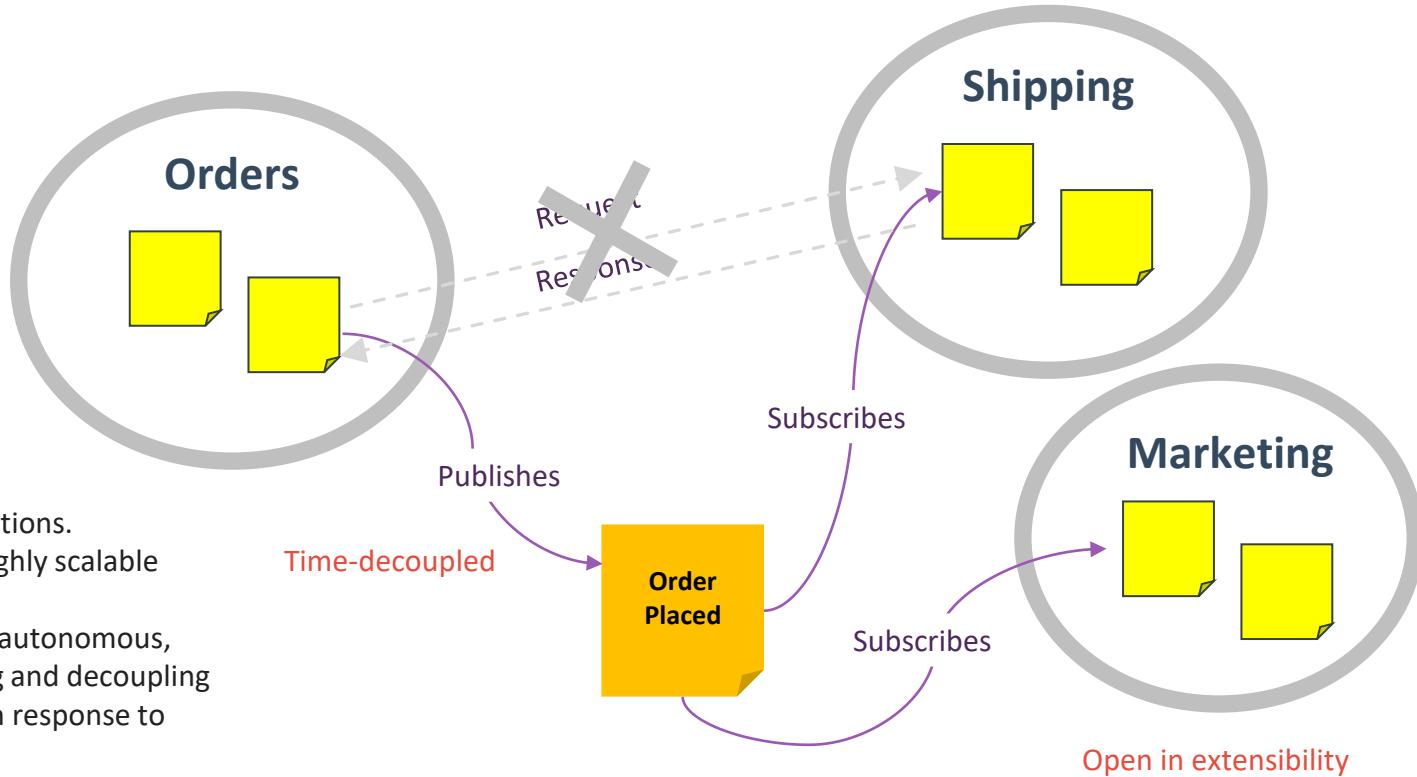
→ Separation of Concerns, Parallel Development, Easy Outsourcing



[Limitation]

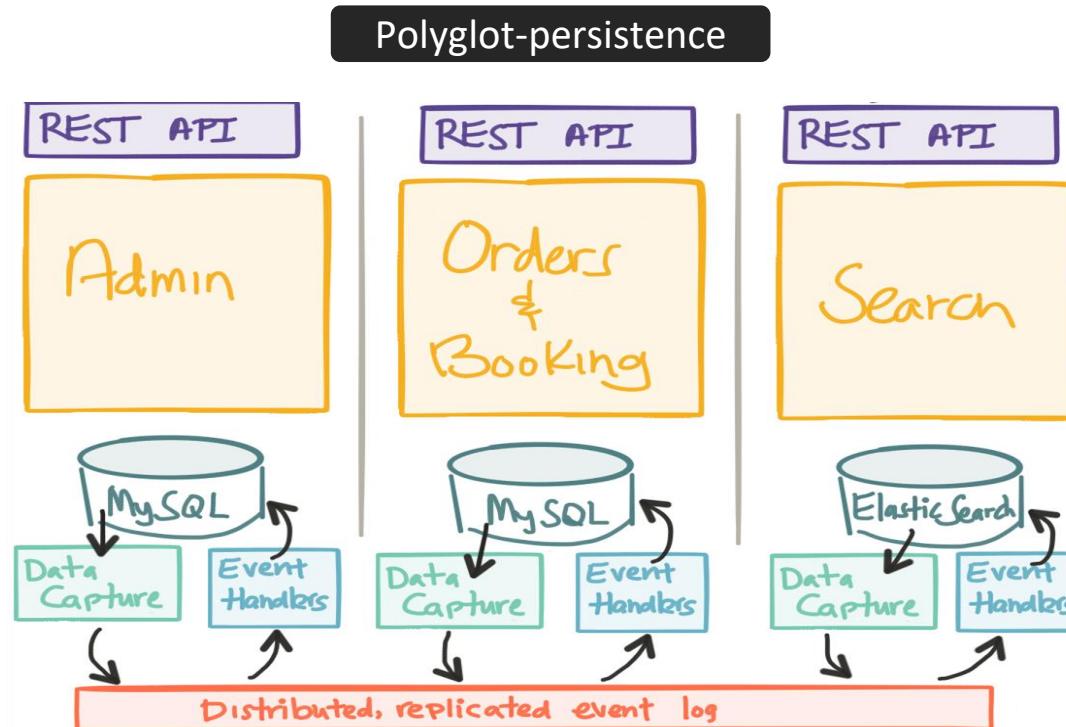
1. Code Coupling 해소
But, Time Coupling 잔존
2. 서비스 Blocking 가능성 내재
3. 장애 전파 우려
4. Point to Point 연결에 따른 복잡한 스파게티 네트워크

Approach #2 : Event Driven Architecture

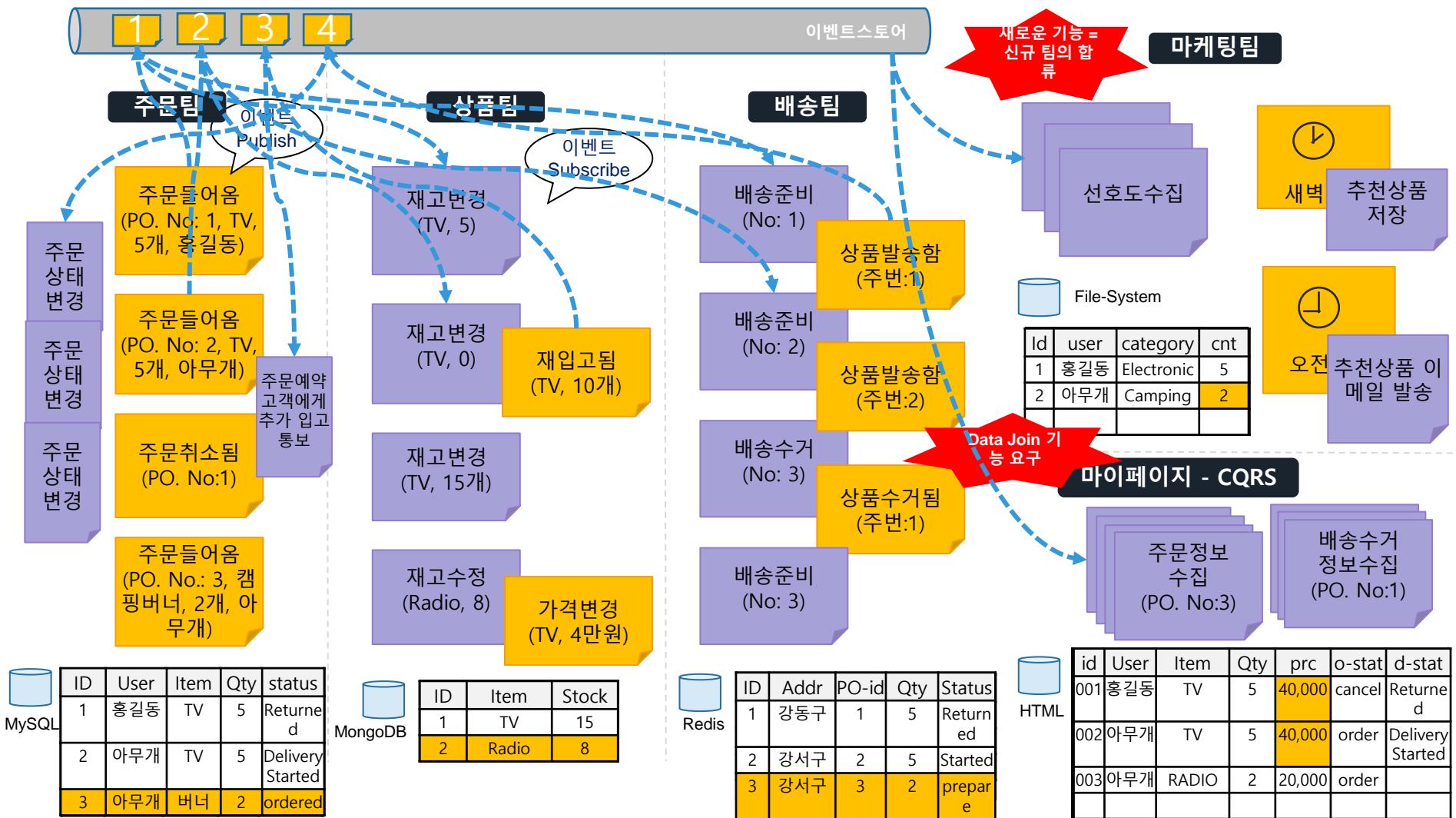


- No point-to-point integrations.
- High performance and highly scalable systems.
- Components can remain autonomous, being capable of coupling and decoupling into different networks in response to different events.
- Advanced analytics can be done using complex event processing.

Approach #2 : Event Driven Architecture



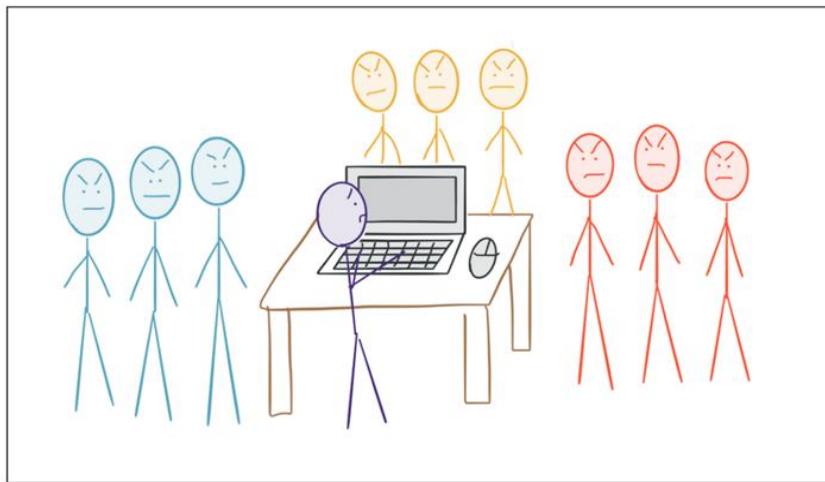
Source: <https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7>, Kevin webber



Benefits of Event-Sourcing?

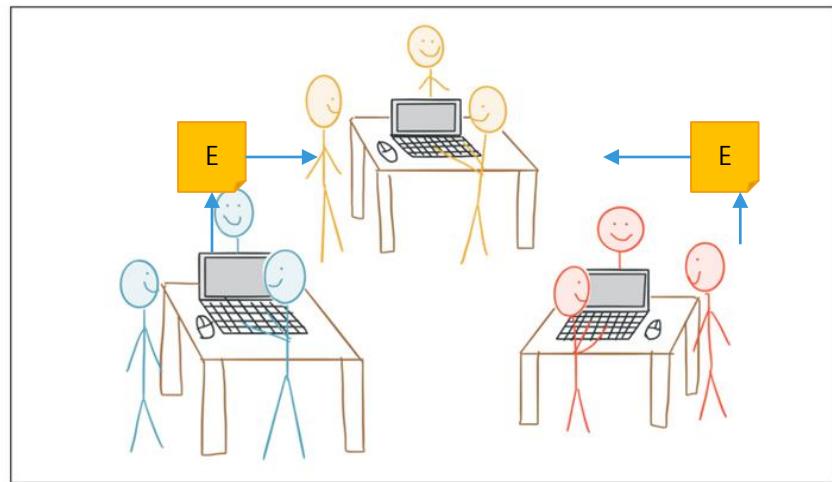
- The event store provides a complete log of every state change, effectively creating an audit trail of the entire system.
- The state of any object can be recreated by replaying the event store.
- It removes the need to map relational tables to objects.
- An event store can feed data into multiple read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



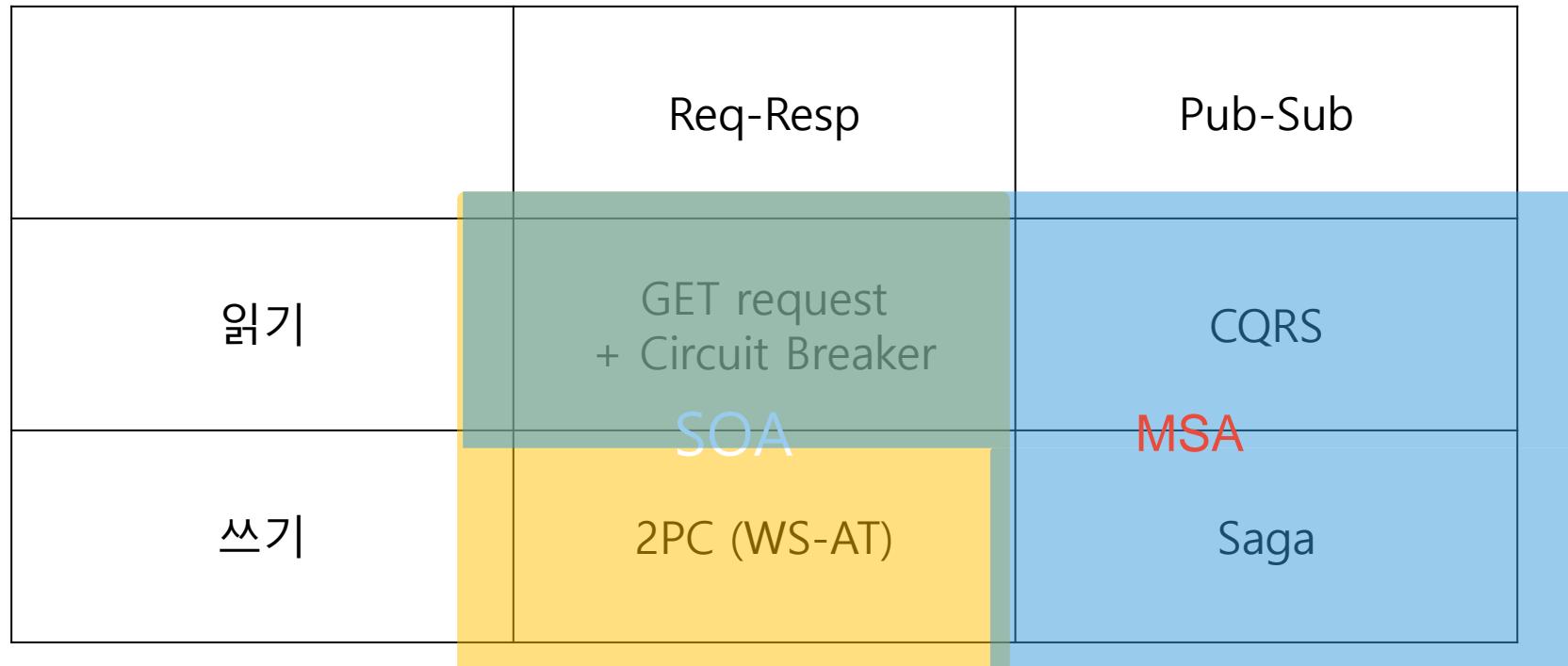
Reactive Microservices:

- Autonomously designed Model(Document)
- Polyglot Persistence

Case Study – 11번가

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

Integration Strategies Compared



Approach #3: BizDevOps Process & Infra

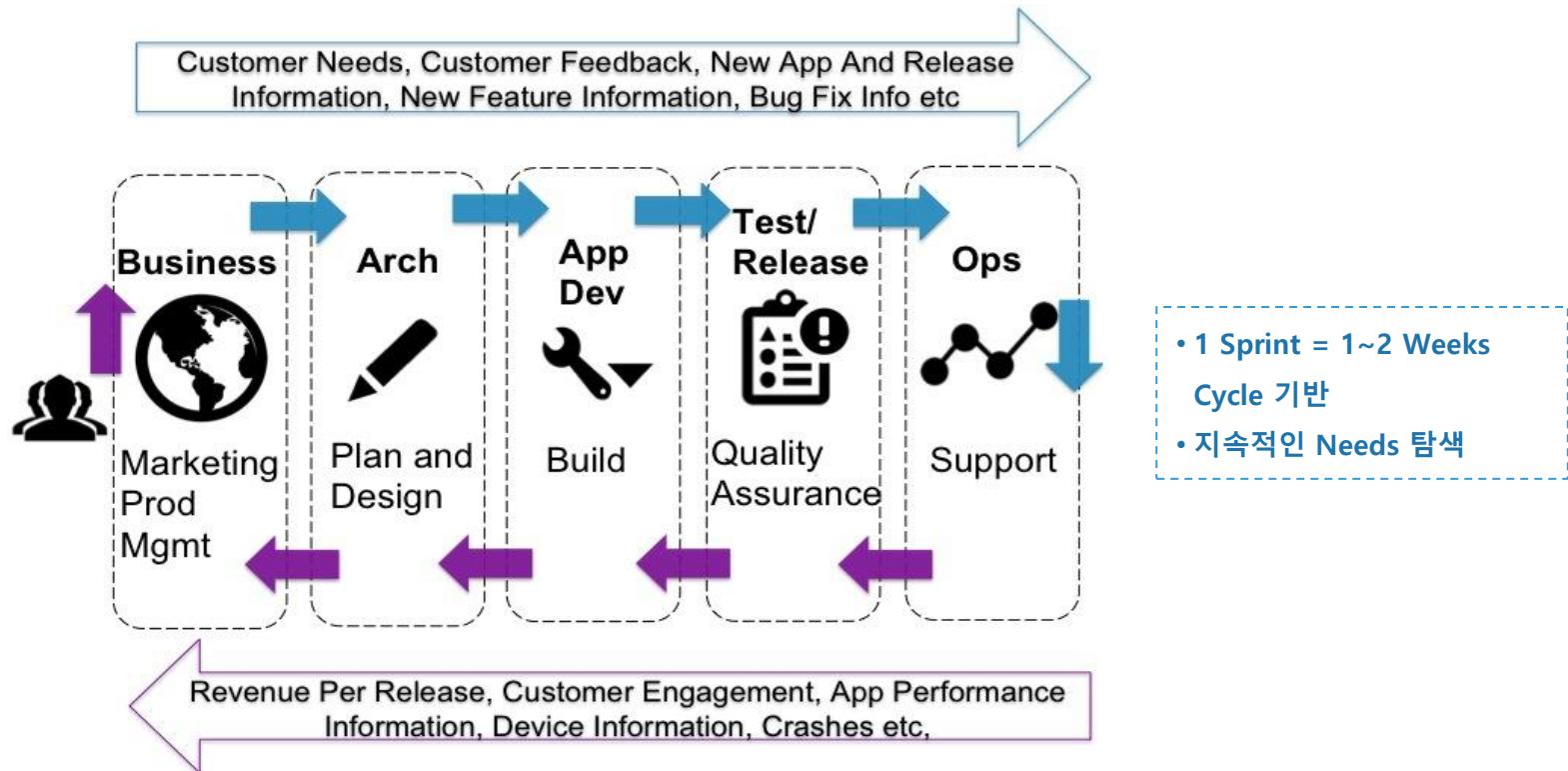
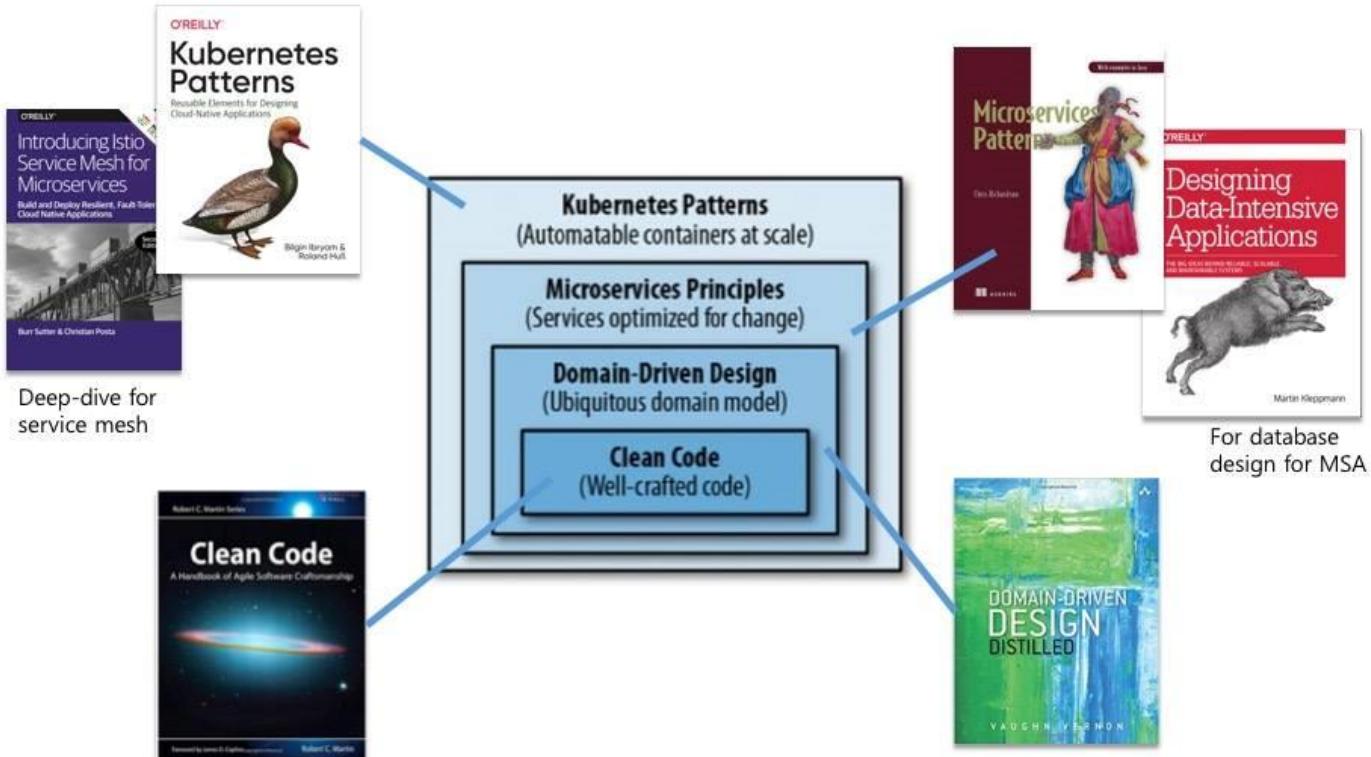


Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming ✓
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

Learning Path to Cloud Native

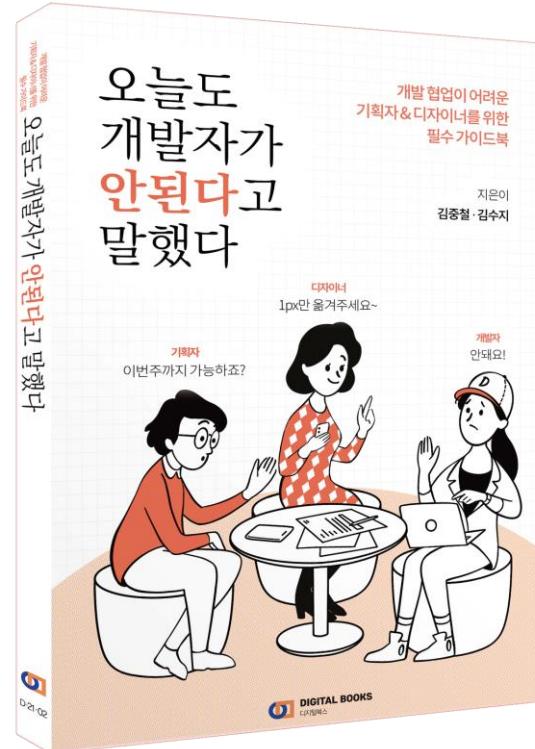


도메인 주도 설계는 소프트웨어의 복잡성을 관리하는 전략적인 도구

- Domain-Driven Design 은 복잡한 문제 영역에 대한 언어이자 도메인 중심의 소프트웨어 디자인 접근법
- 에릭에반스의 저서인 "Domain-driven Design: 소프트웨어의 복잡성 관리하기"에서 언급된 용어
- 팀이 소프트웨어를 저작함에 있어 기술적, 비즈니스적 복잡성을 관리하여 비즈니스를 성공시키기 위한 핵심적인 패턴과 원칙, 그리고 실천법으로 구성



Eric Evans
Domain-Driven Design:
Tackling Complexity in the Heart of Software

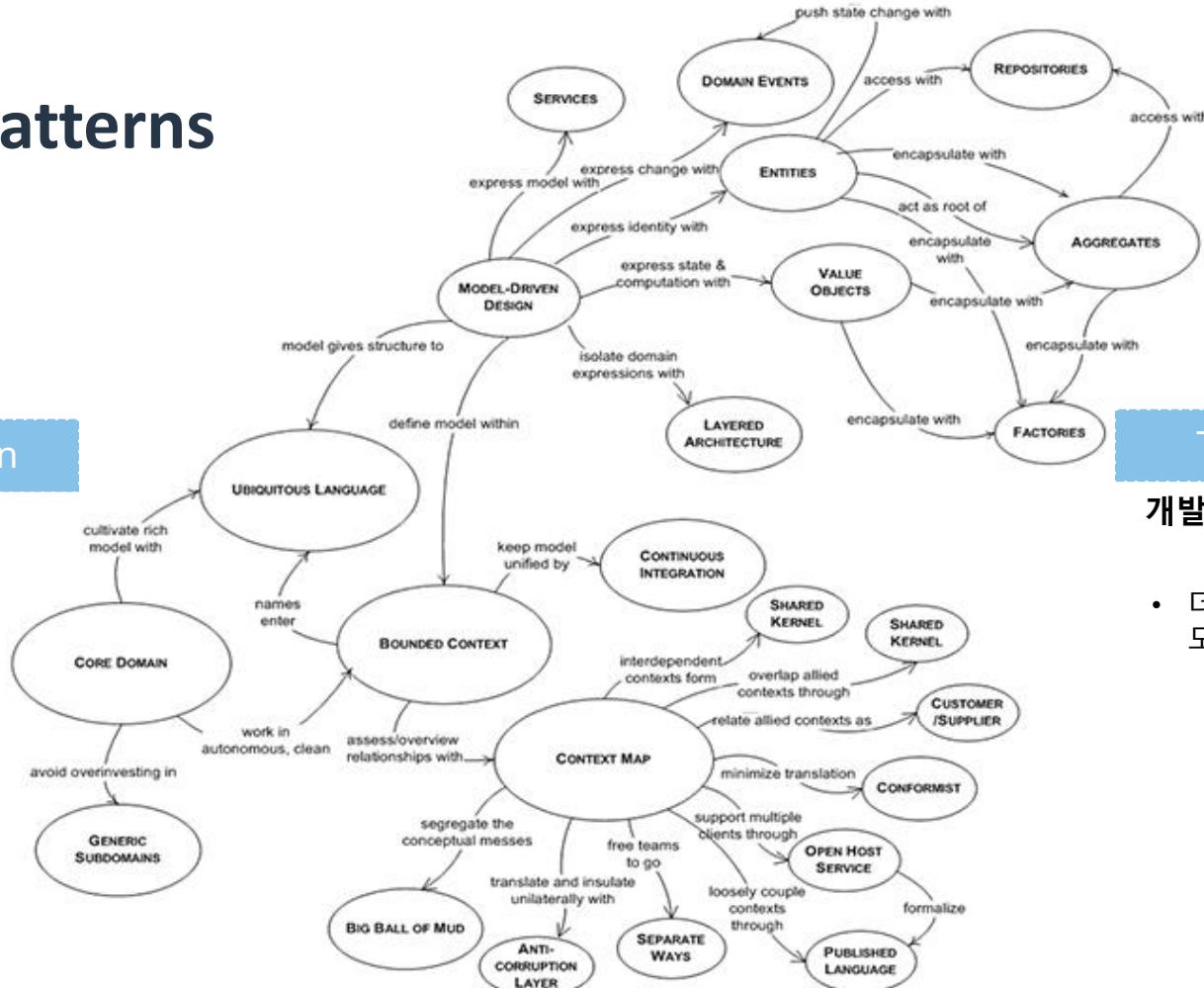


DDD Patterns

Strategic Design

개념설계 (전략적 설계)

- 비즈니스 도메인을 매핑하고 도메인 모델에 대해 경계가 있는 컨텍스트 정의



Tactical Design

개발을 위한 구체적 설계 (전술적 설계)

- 더 상세하게 도메인 모델의 구현영역 정의

DDD Goals

- 기존의 현업에서 IT로의 일방향 소통구조를 탈피하여 현업과 IT의 쌍방향 커뮤니케이션을 매우 중시
- 현업 비즈니스가 생성되는 리소스 코드에 그대로 반영되어야 .. (유비쿼터스 언어)
- 업무 전문가가 이해하는 코드

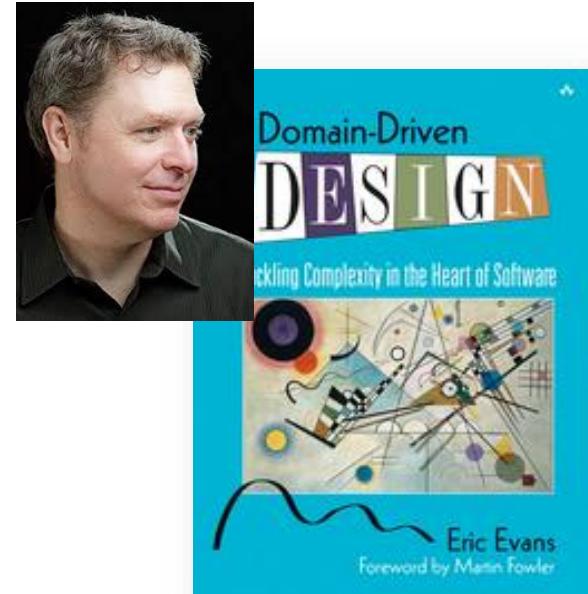
Domain concepts made explicit in the code model

```
public class Guaranteed30MinuteDeliveryOffer
{
    public void After(PizzaDelivered delivery)
    {
        if (delivery.TimeTaken.Exceeded(thirtyMinutes))
        {
            sendCouponTo(delivery.Customer);
        }
    }
}
```

Domain-Driven Design & MSA

DDD for MSA

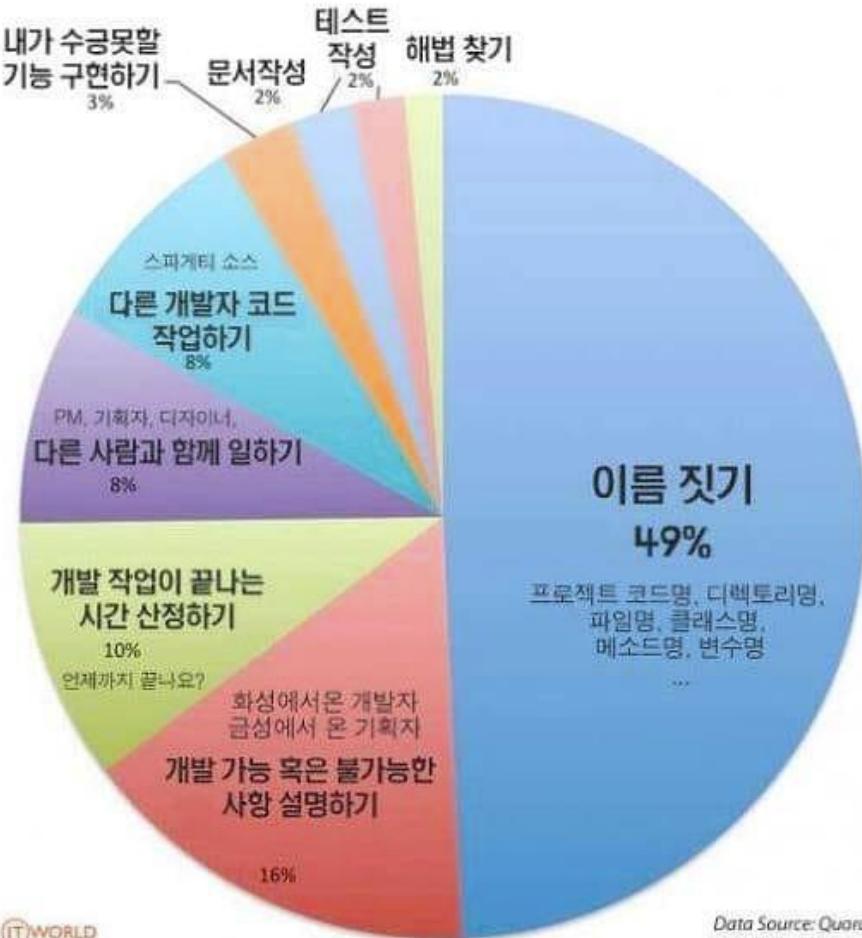
- **Bounded Context 와 Ubiquitous Language**
→ 어떤 단위로 마이크로 서비스를 쪼개면 좋은가?
- **Context Mapping**
→ 서비스를 어떻게 결합할 것인가?
- **Domain Events**
→ 어떤 비즈니스 이벤트에 의하여 마이크로 서비스들이 상호 반응하는가?



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



프로그래머가 가장 힘들어하는 일은?



Data Source: Quora/Ubuntu Forums
Total Votes: 4,522

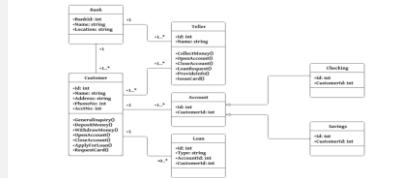
Bounded Context

(한정된 맥락)

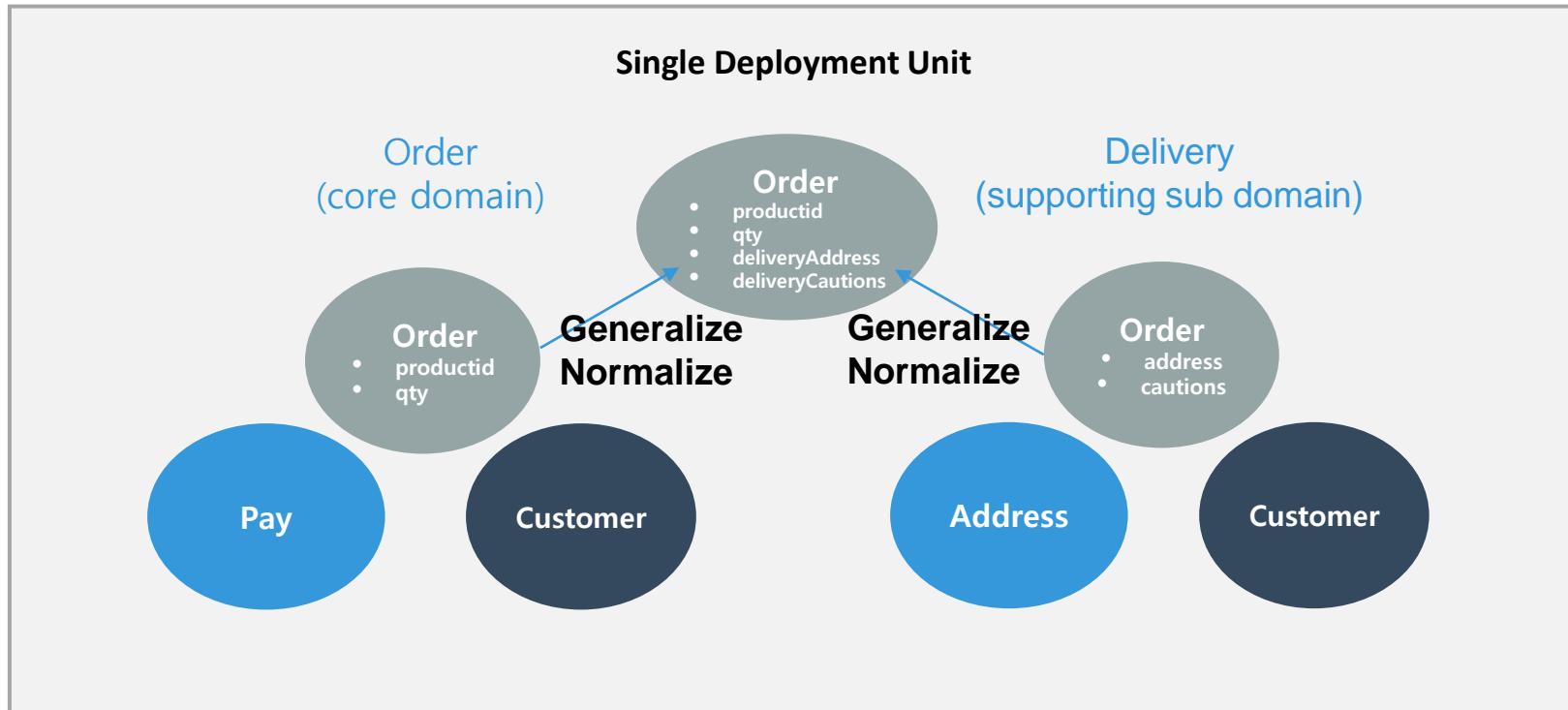
Ubiquitous Language
(도메인 언어)



Bounded Context and Ubiquitous Language

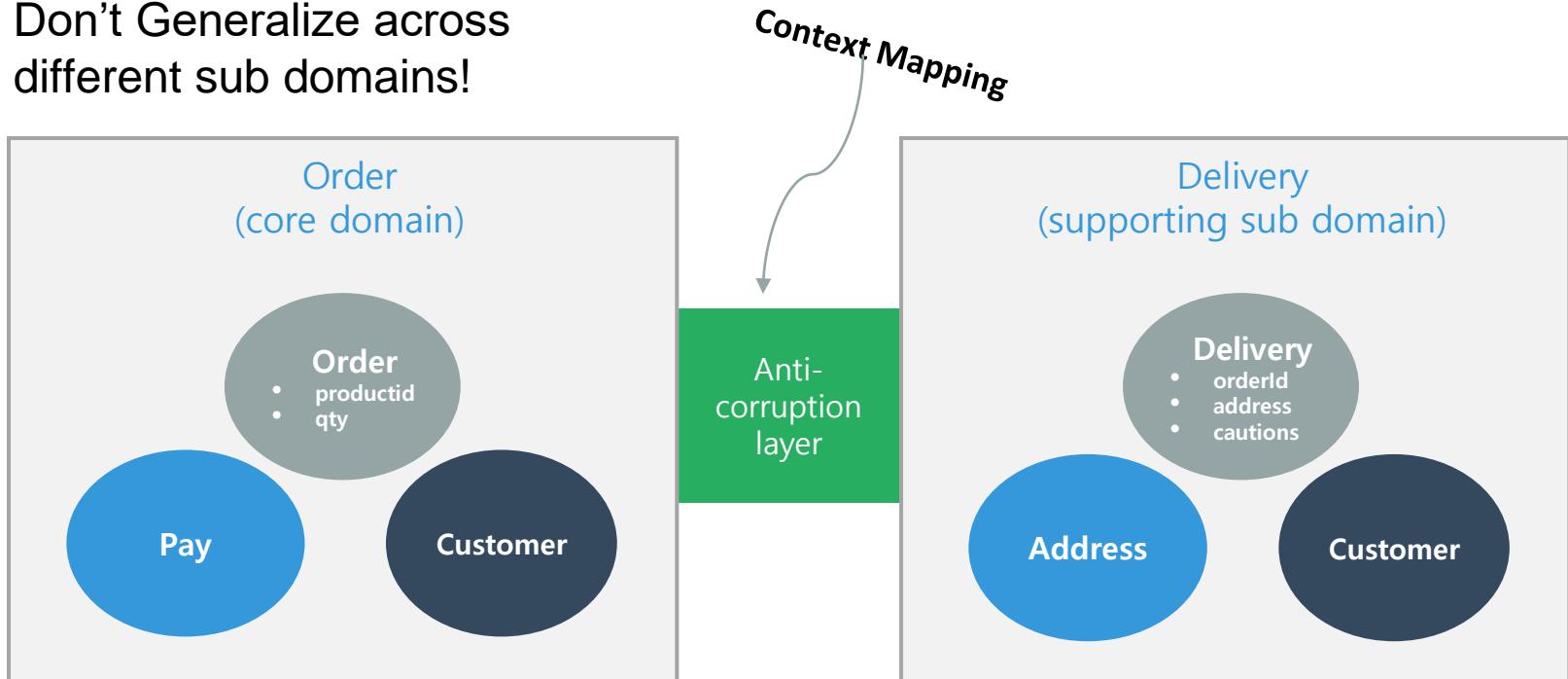
	SW	CONSTRUCTION
A Project		
An Architecture		
A Developer		

In Monolith : Single Model Fits All



In MSA : Multiple Models

Don't Generalize across
different sub domains!

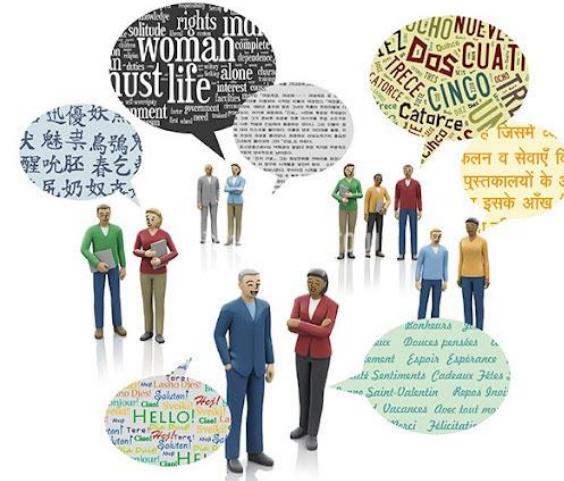


Comparison of Total Communication Cost

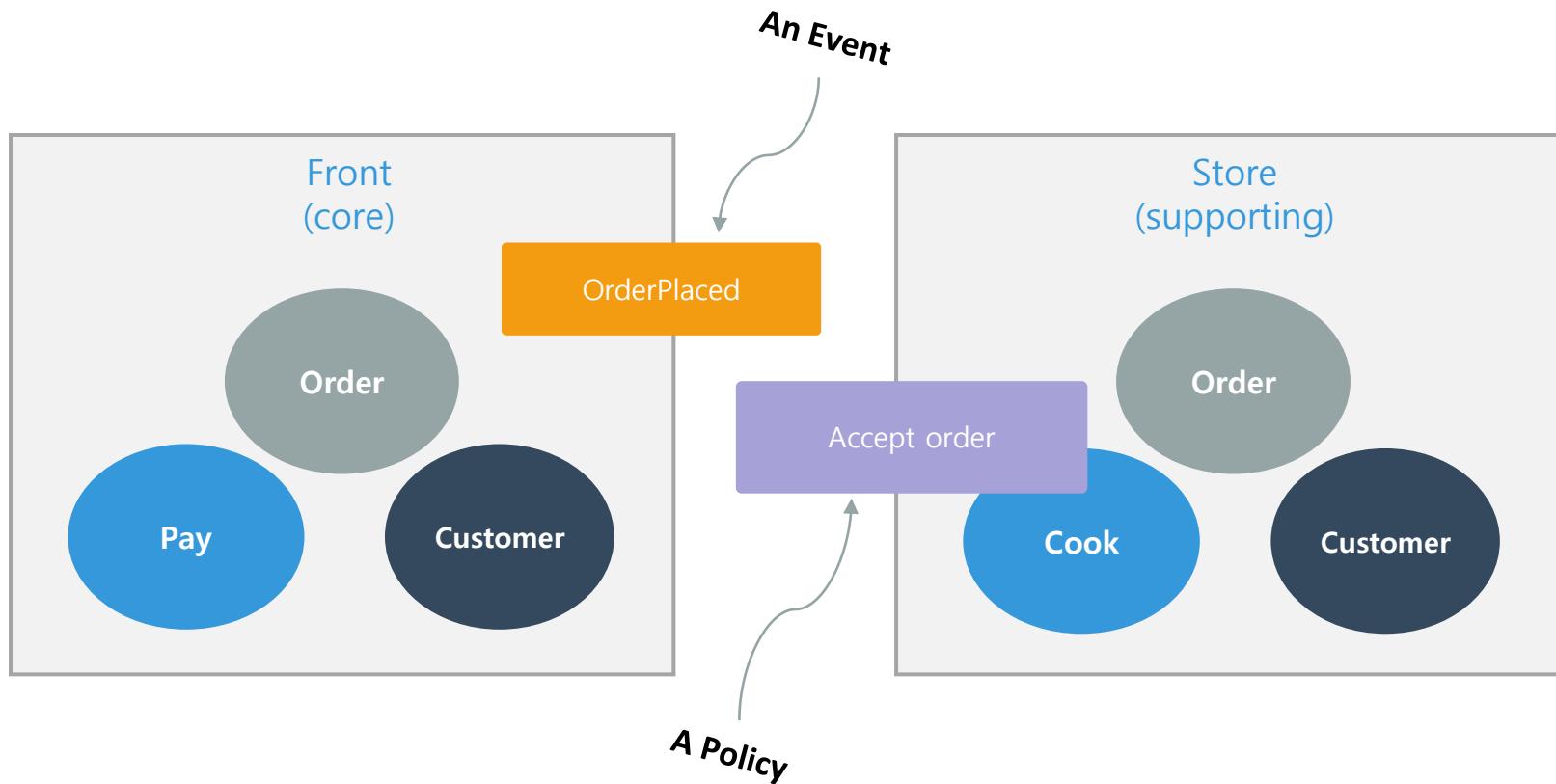
- Generalization and Normalization of single language



- Translators between multiple language
 - Anti-corruption between Multiple languages

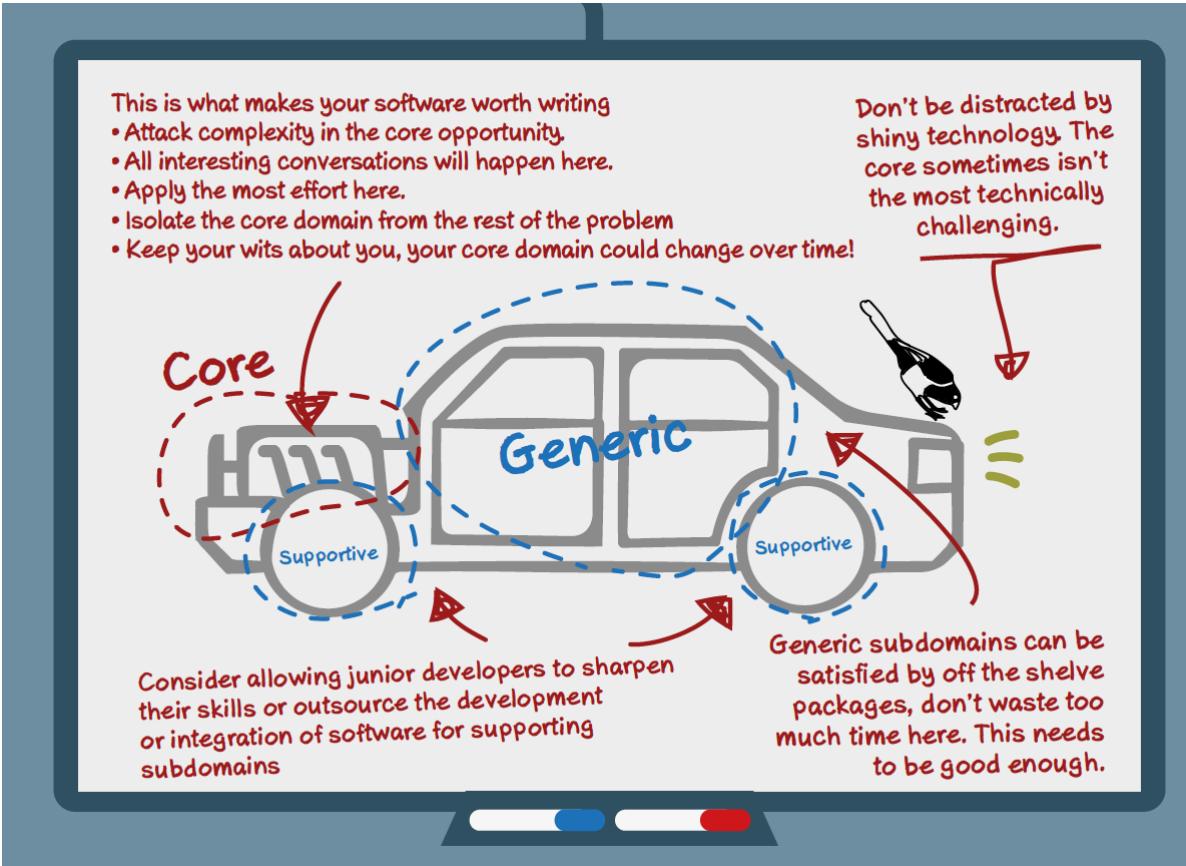


In Microservices :



Class of sub-domains

- Core
- Supportive
- Generic



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

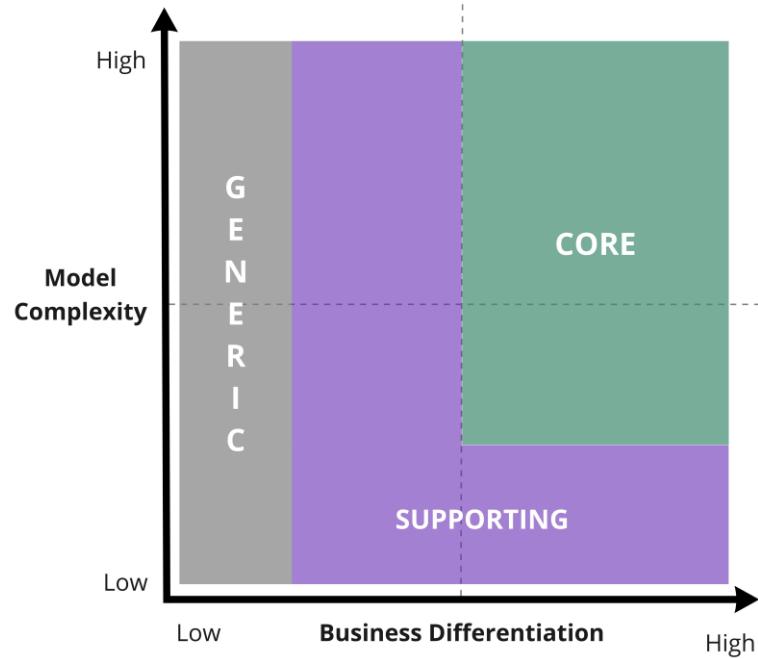
- 기업의 핵심 경쟁력이 아닌, 직접 운영해도 좋지만 상황에 따라 아웃소싱 가능한
- 시스템 리소스가 모자라면 외부서비스를 빌려쓰는 것을 고려할만한
예) 재고관리, 배송, 회원관리 서비스 등



3순위 : General Domain

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

Diff. between Subdomain Types



서브도메인 유형	경쟁적 우위	복잡도	변화성	구현	문제
Core	Yes	High	High	In-house	재미있는
Supporting	No	Low	Low	In-house /아웃소싱	뻔한
Generic	No	High	Low	구매/적용	해결된

Ref : Relation types between services

- **Shared Kernel**

- 두 Bounded Context가 같은 모델을 공유하는 경우로, 임의 모델 변경 불가 등 두 팀간의 밀접한 관계 유지가 필수

- **Customer/ Supplier**

- 가장 흔한 관계로 한쪽에서 API를 제공하고 다른 한쪽에서 그 API를 호출하는 관계로 Downstream BC은 API를 제공하는 Upstream BC에 의존

- **Conformist**

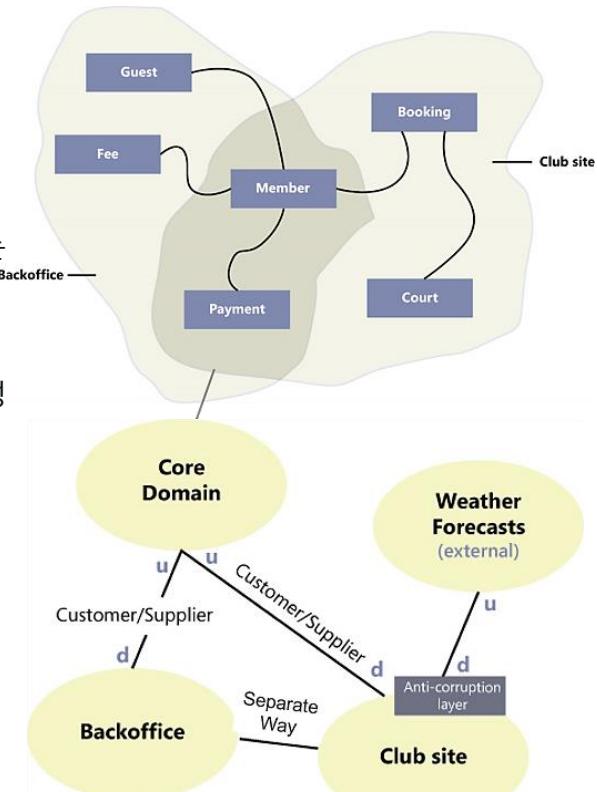
- Upstream BC의 API를 엄격하게 준수하여 Bounded Context간 통신의 복잡성을 제거하는 방식

- **Separate Ways (Anti-Corruption Layer)**

- 두 Bounded Context간 서로 통합하지 않는 방식으로 다른 BC와의 통합은 사용자가 직접 수동으로 이루어 짐 (스케일이 증가할수록 한계)

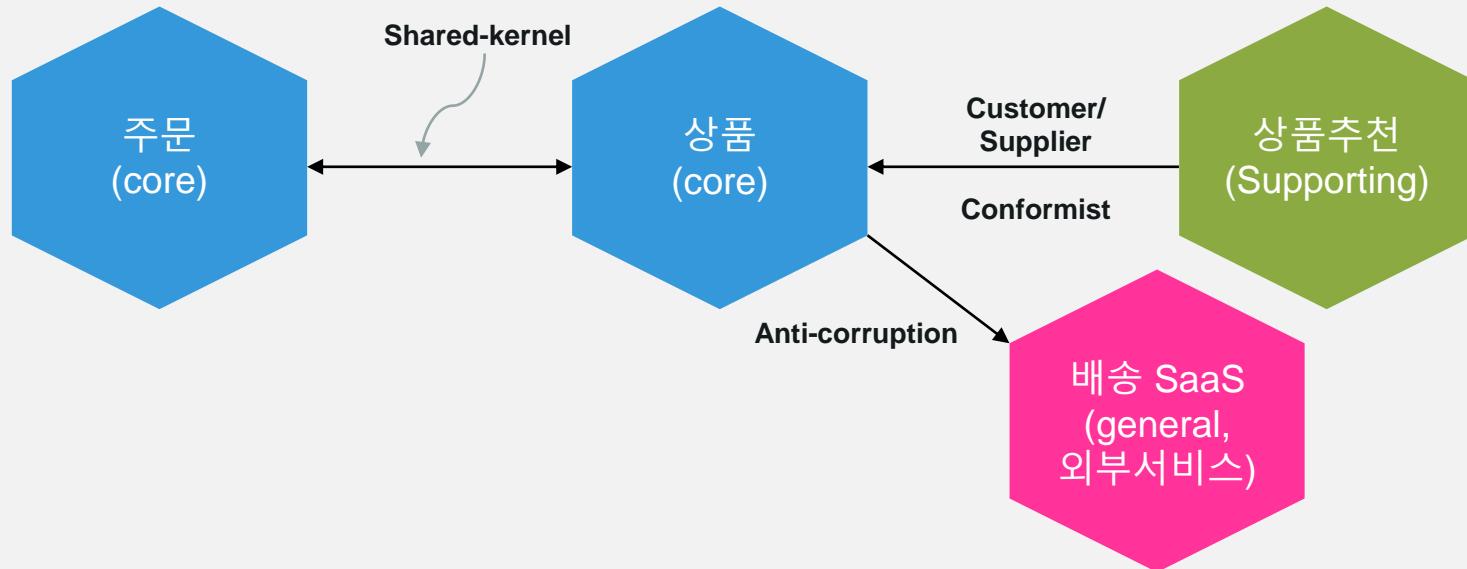
- **Open Host Service**

- 통합검색 서비스처럼 Downstream BC별로 구현하기 보다 여러 종류의 Downstream BC를 위해 Upstream BC가 통합



マイクロ 서비스간의 서열과 역학관계

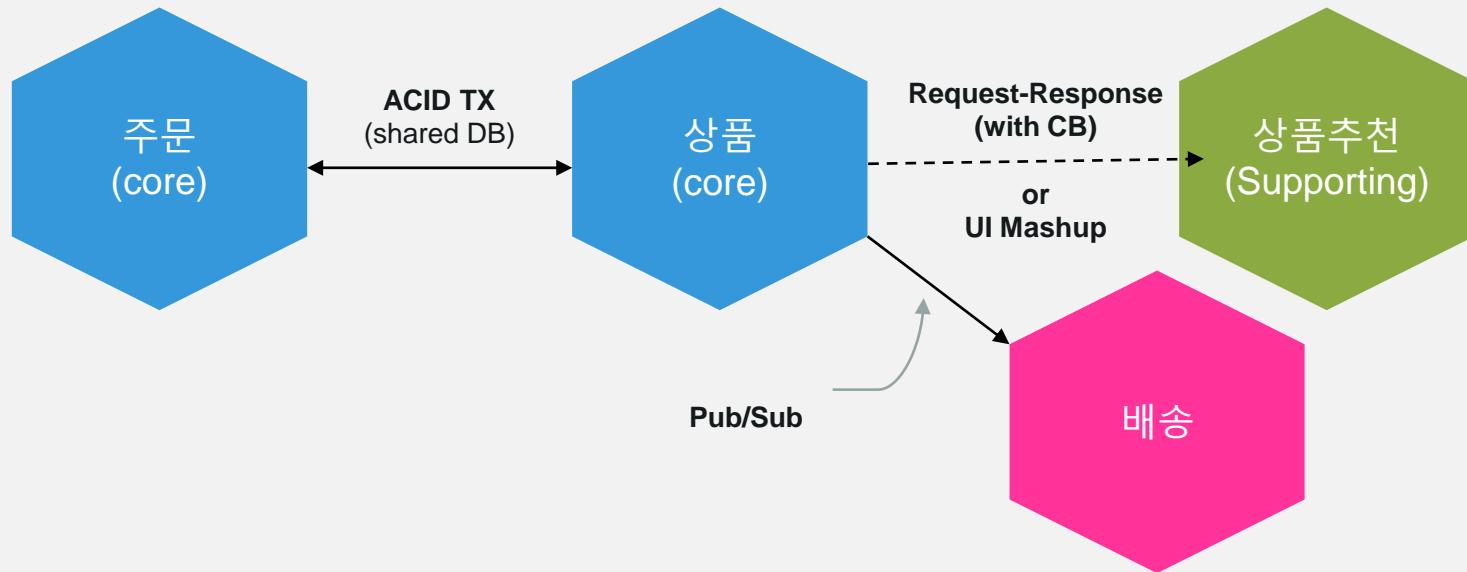
“ 어느 마이크로 서비스의 인터페이스를 더 중요하게 관리할 것인가? ”



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

マイクロ 서비스간의 서열과 역학관계

“ 마이크로서비스간 트랜잭션의 묶음을 어떻게 할 것인가? ”



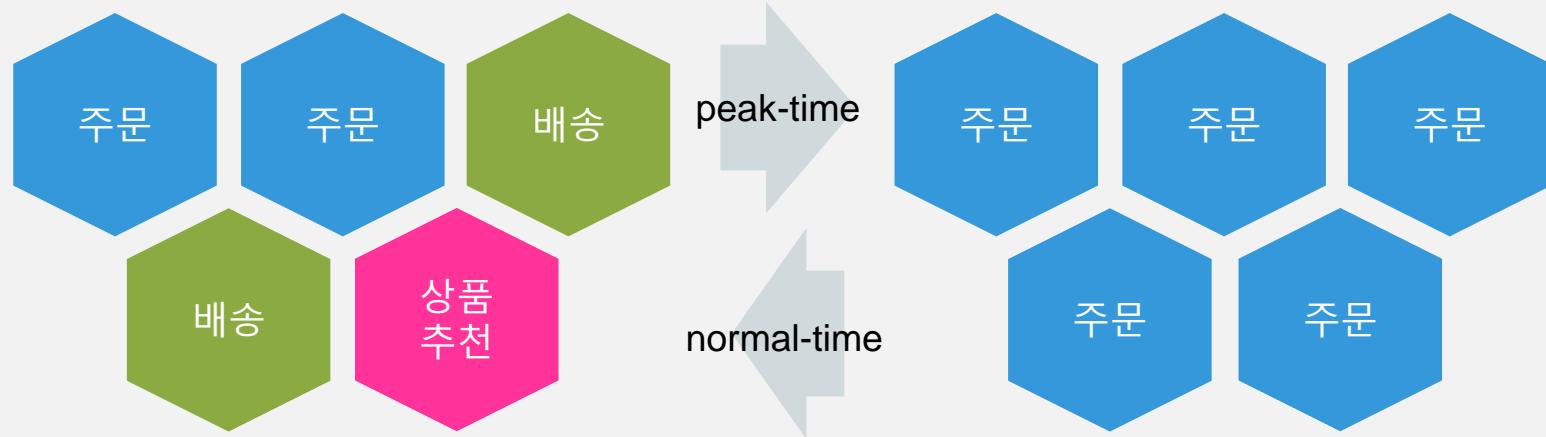
배송기사가 없다고 주문을 안받을 것인가? 상품 추천이 안된다면 주문버튼을 안보여줄 것인가?

Core Domain 간에는 강결합이 요구되는 경우가 생길 수 있다.

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

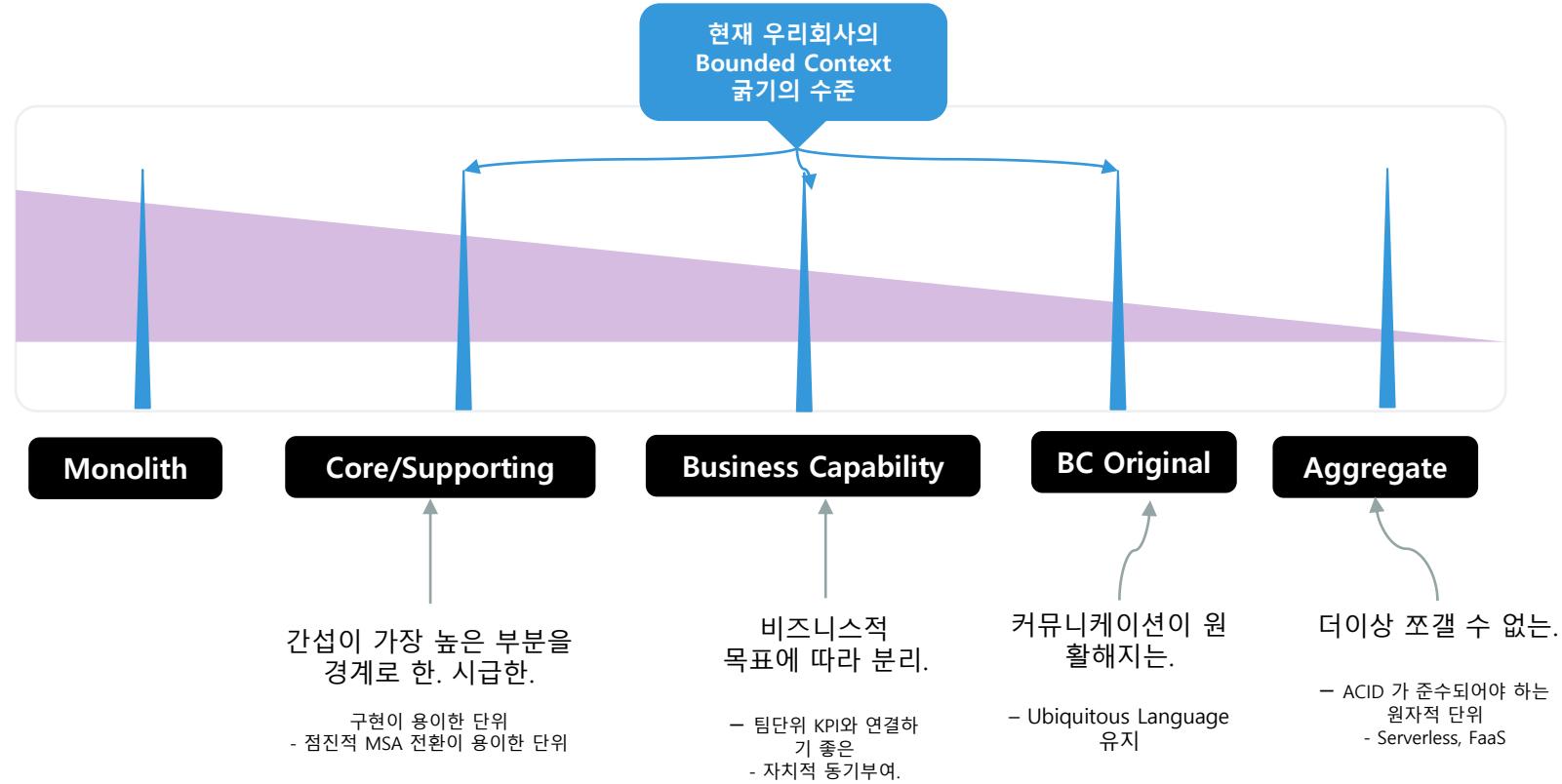
マイクロ 서비스간의 서열과 역학관계

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

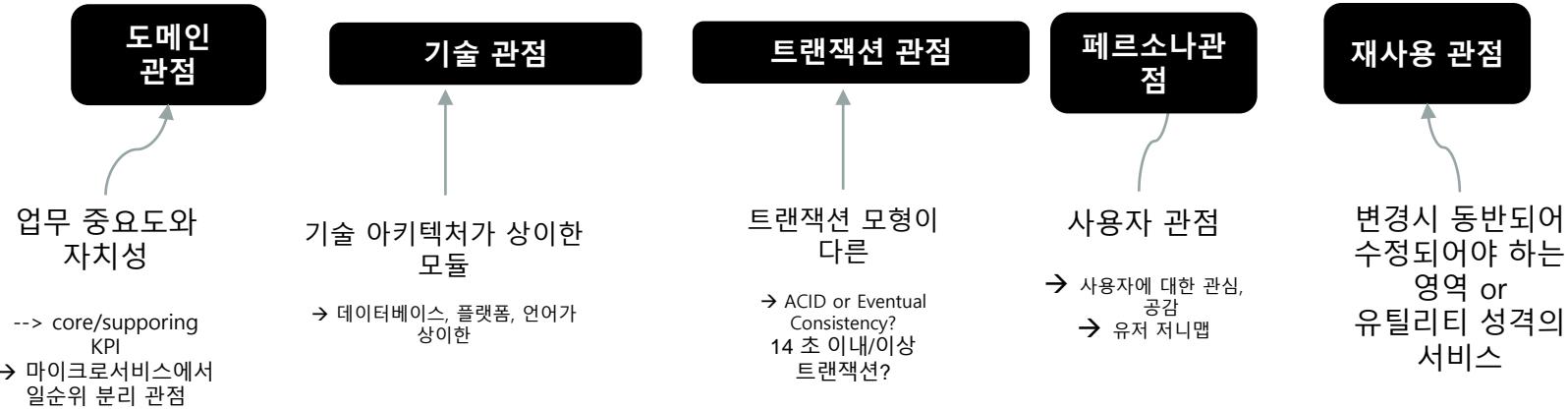


배송서비스는 야간에 올라와서 후에 처리되어도 된다.
주문이 몰려드는 시간에 주문을 못 받으면 배송은 의미가 없다

분리수준 - 어느 굽기로 쪼갤 것인가?



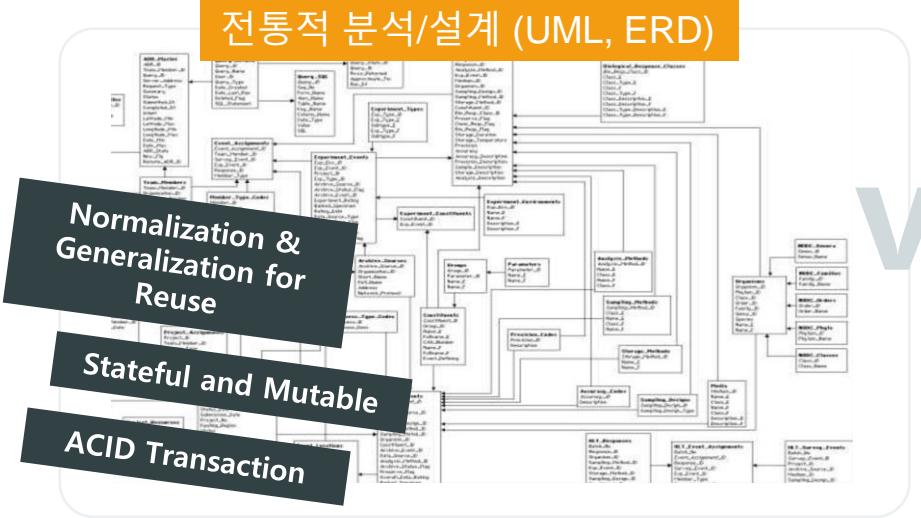
분리관점 - 어떤 관점으로 쪼갤 것인가?



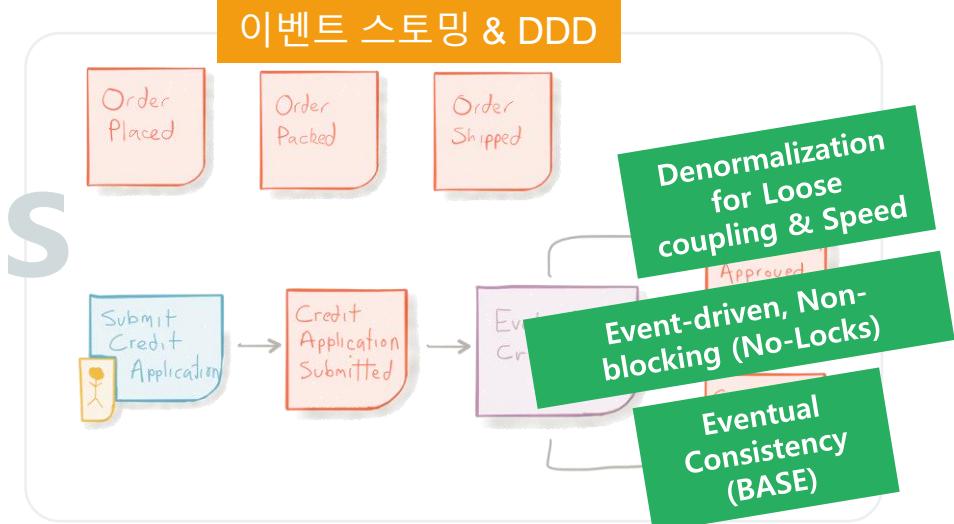
Event Storming : DDD 를 쉽게 하는 방법

- 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)

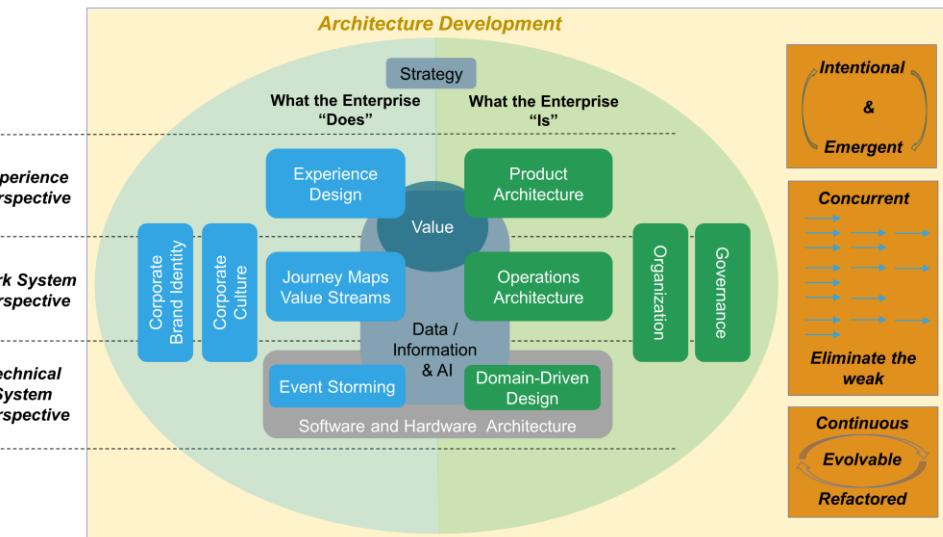


이벤트 스토밍 & DDD



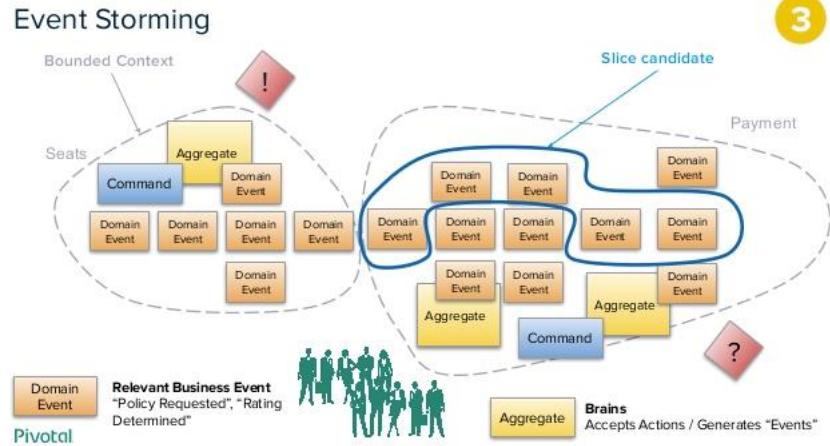
Event Storming 의 위상

The Open Group 의 OAA 의 빌딩블록



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

Pivotal 의 AppTX 방법론



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

IBM 의 Garage 방법론... etc

Event Storming : Prepare

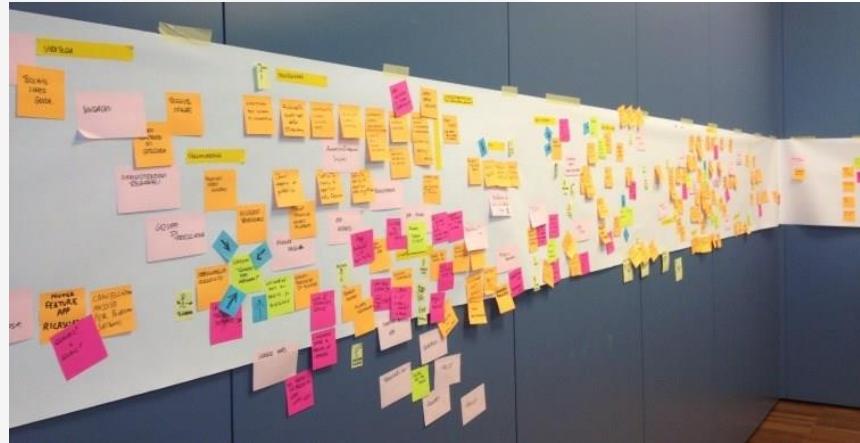
People

- 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)

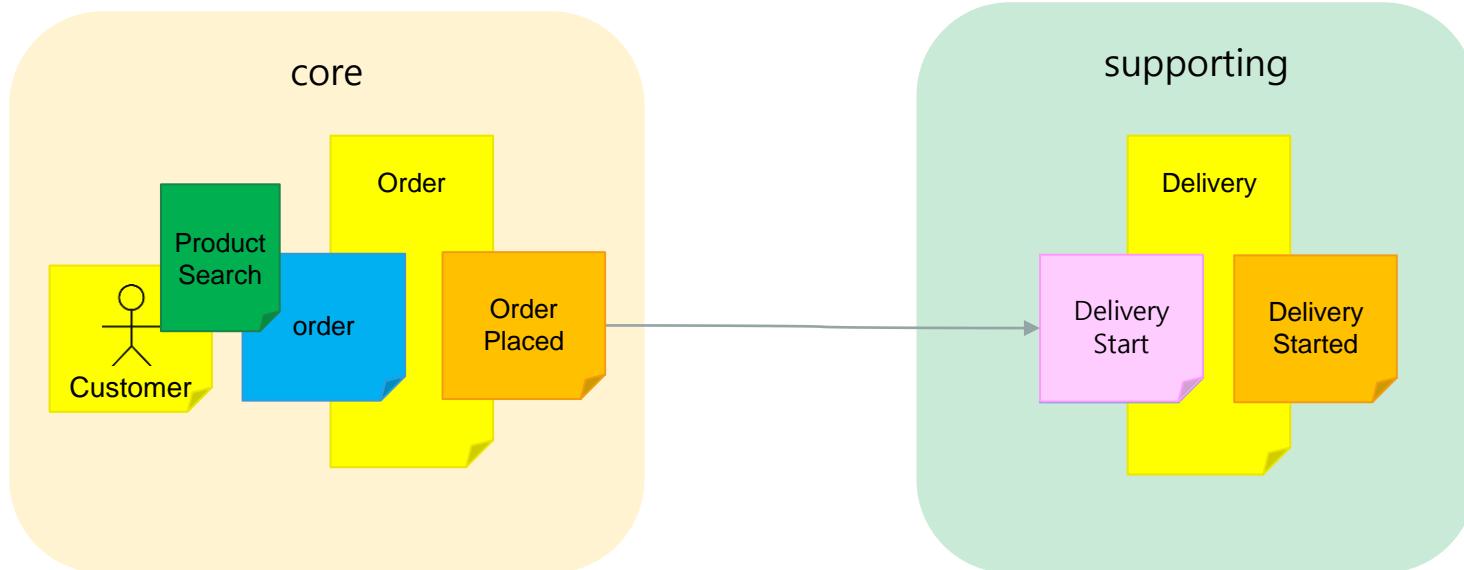
이벤트에 대한 반응
(서브스크라이브)
비즈니스 룰 엔진 등

Comment
Or
Question
(Purple)

의견 또는 질문

추가적인 내용 입력,
예측되는 Risk

Examples



Firstly, Event Discovery

PO 가 주도
(업무전문가)



- 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

Account
Created

Email
Sent

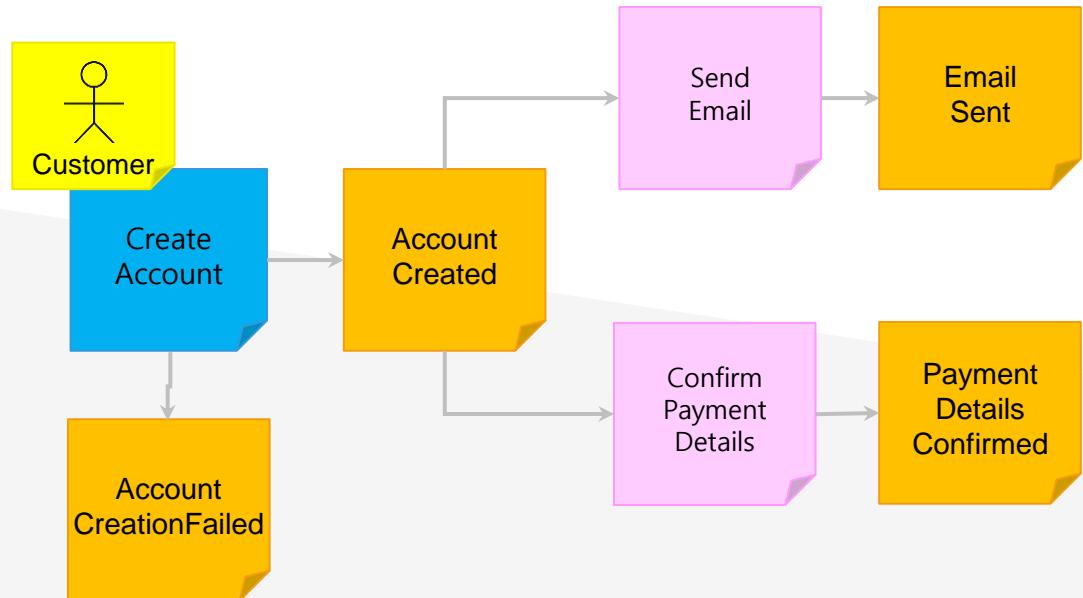
Payment
Details
Confirmed

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 Mgmt

Account Updated

Delete Account

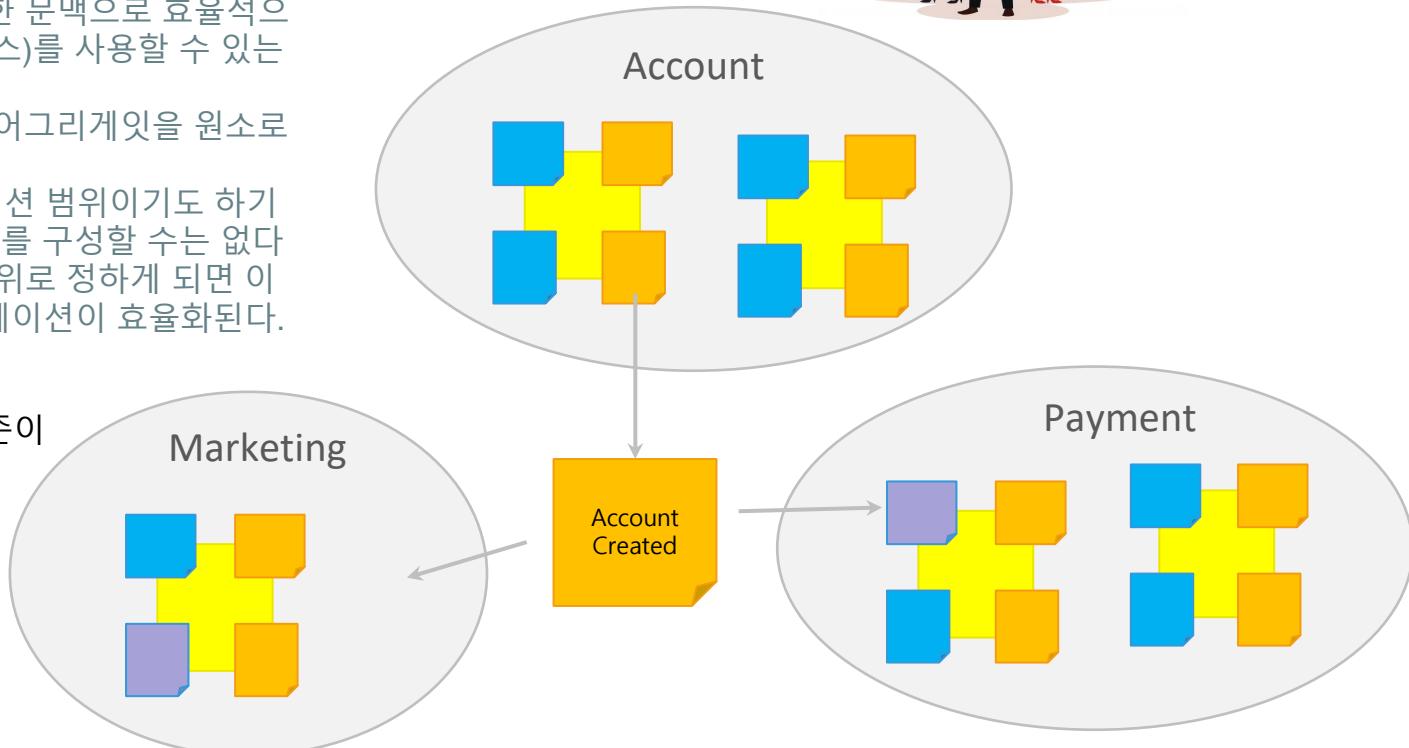
Account Deleted

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
- ...

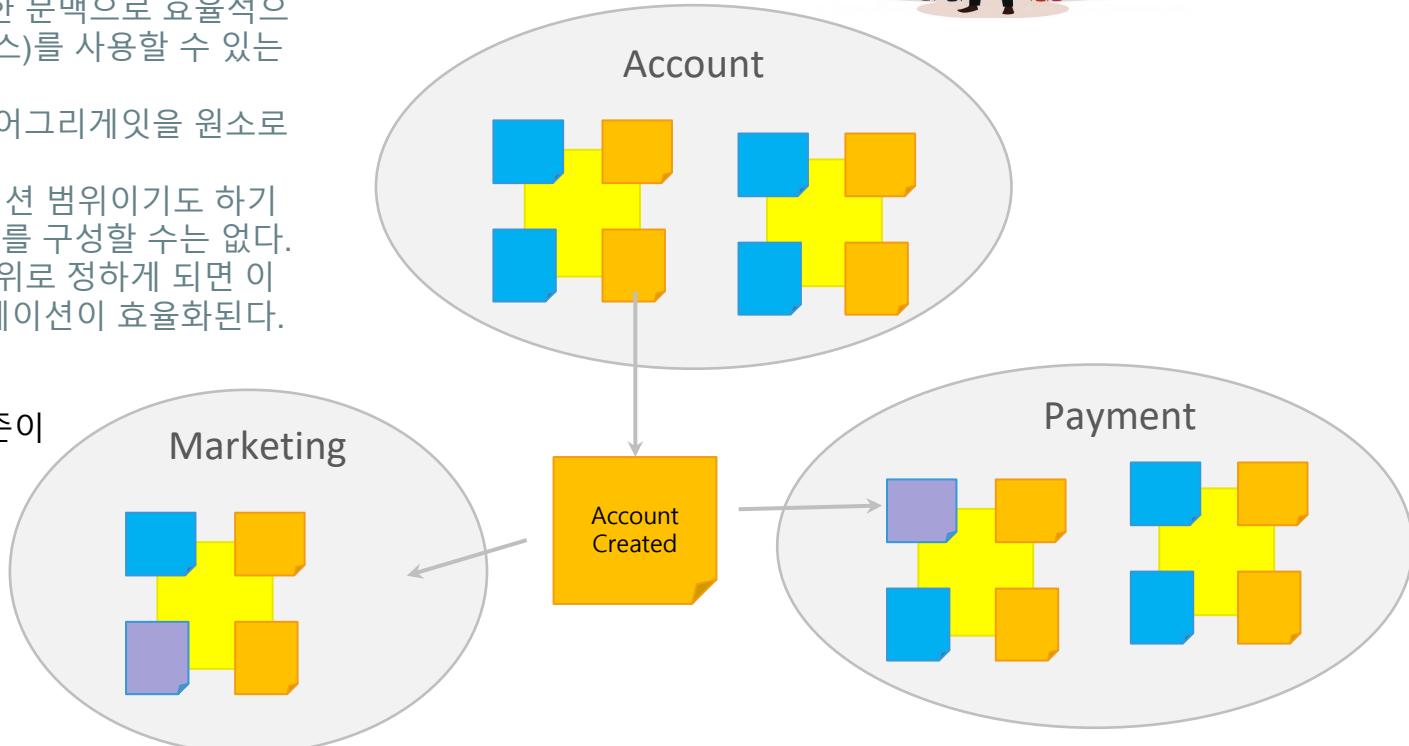


Bounded Contexts 도출

- 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

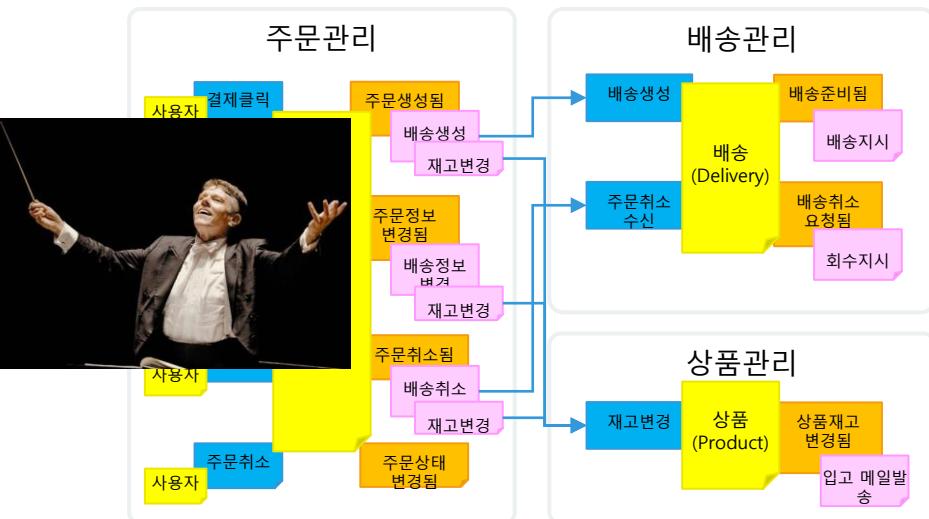
- Domain Boundary
- Transaction Boundary
- Technical Stack
- ...



Context Mapping

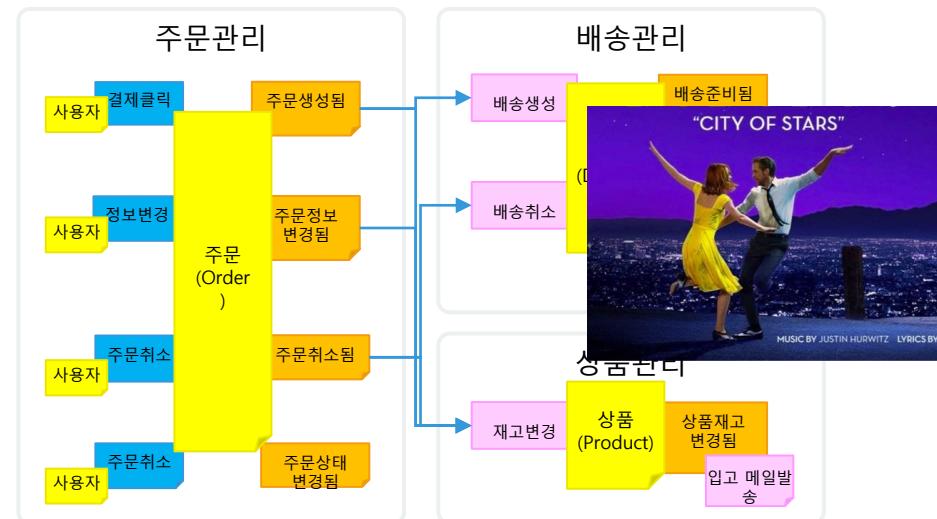
라이락 스티커 (Policy) 를 어디에 둘 것인가? - Orchestration or Choreography? Or Mediation?

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.

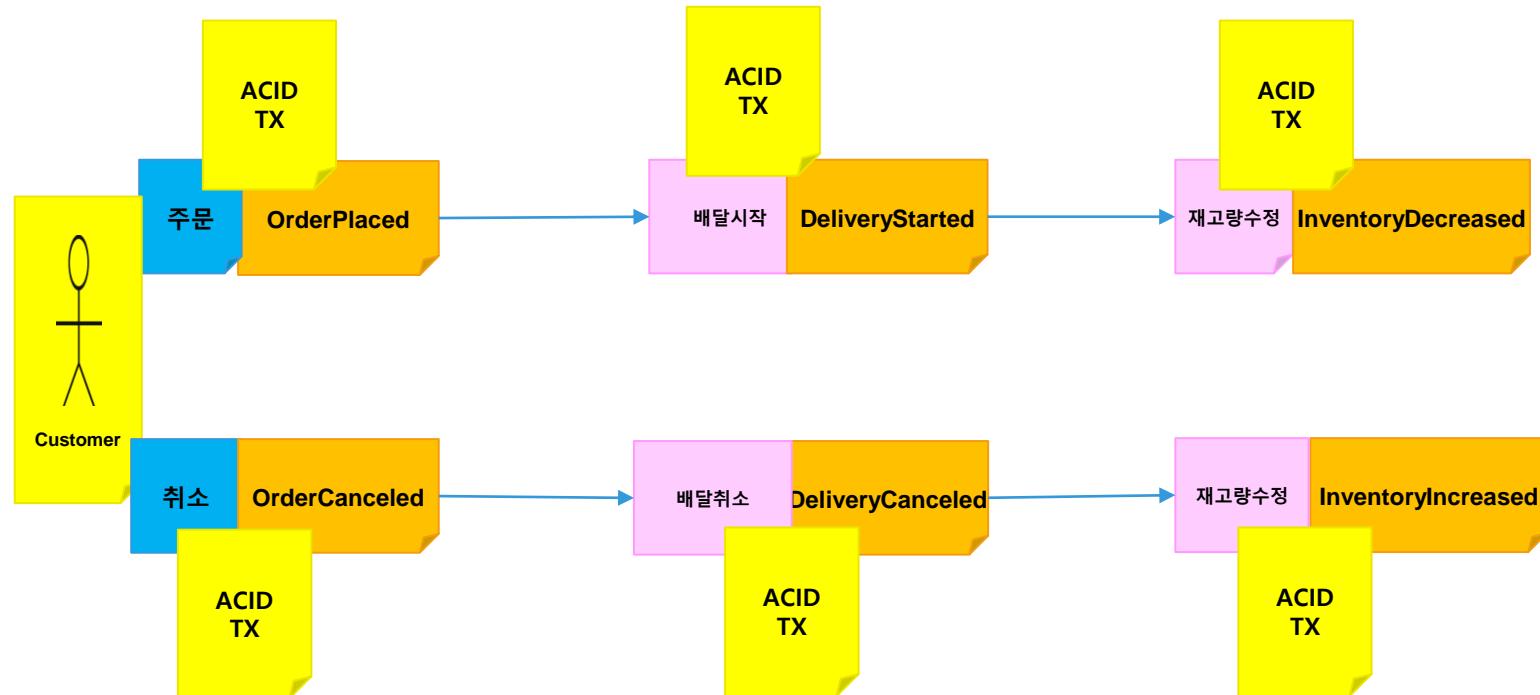


Lab Time : Command, Policy, Actor

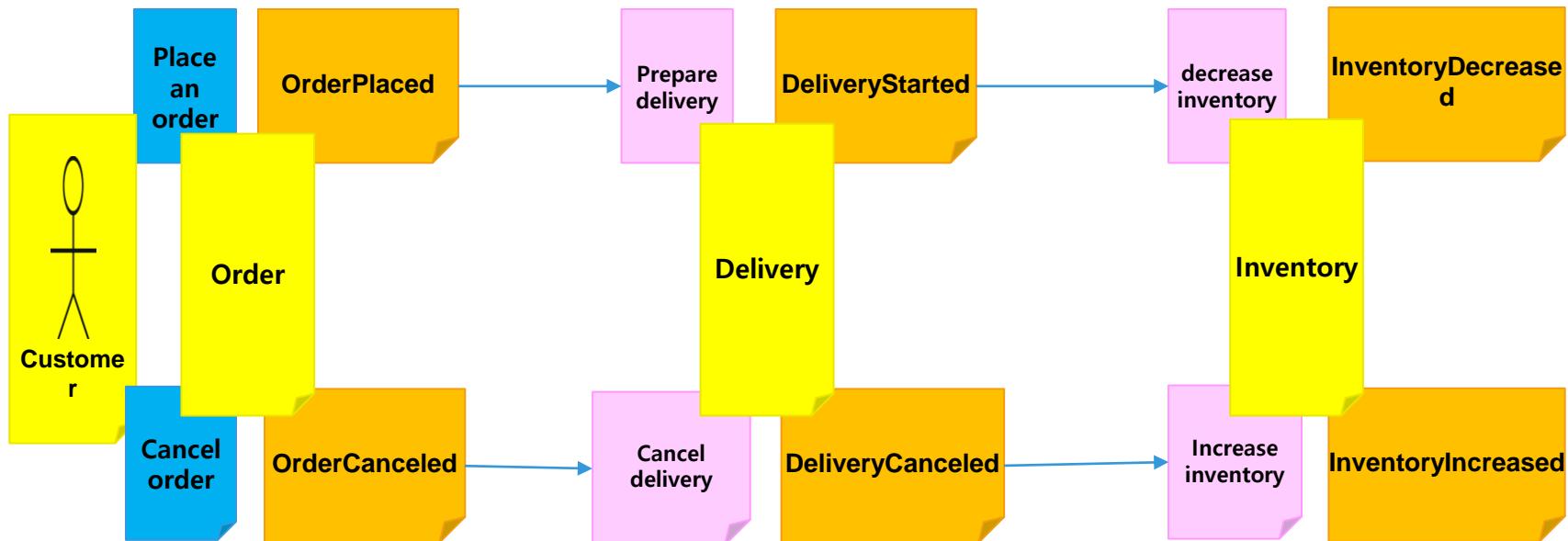
- **Customer** search products and clicks the order button to placed an order.
- When an order is placed, the delivery team prepares for delivery of the product and the delivery will start.
- When delivery has been started, the product inventory is decreased.
- **Customer** can cancel their orders.
-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



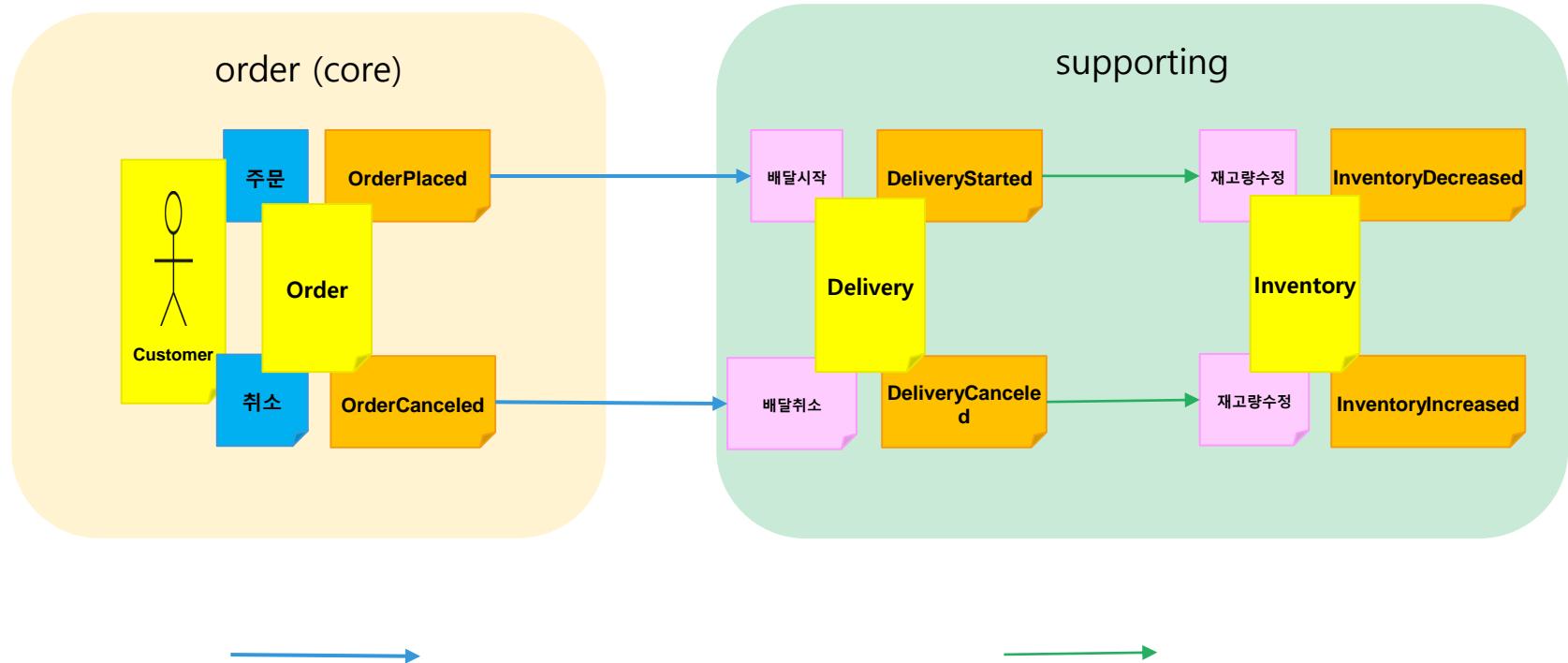
Lab Time : Aggregate



Lab Time : Aggregate (2)



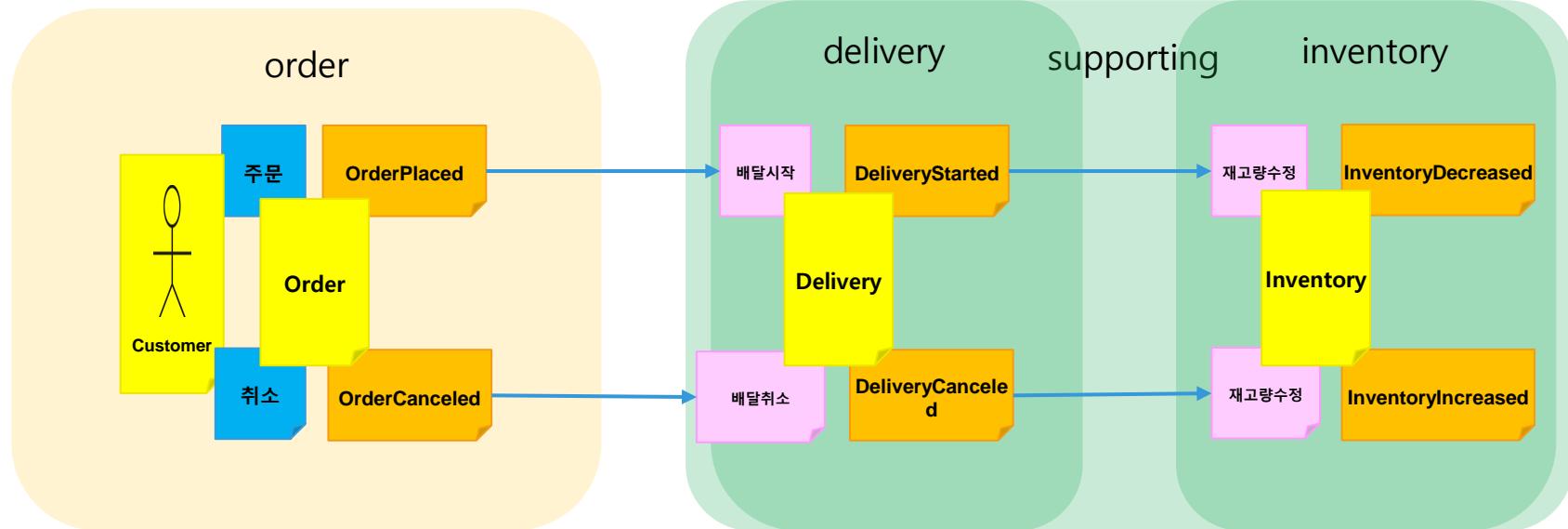
Lab Time : Bounded Context 분리



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

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

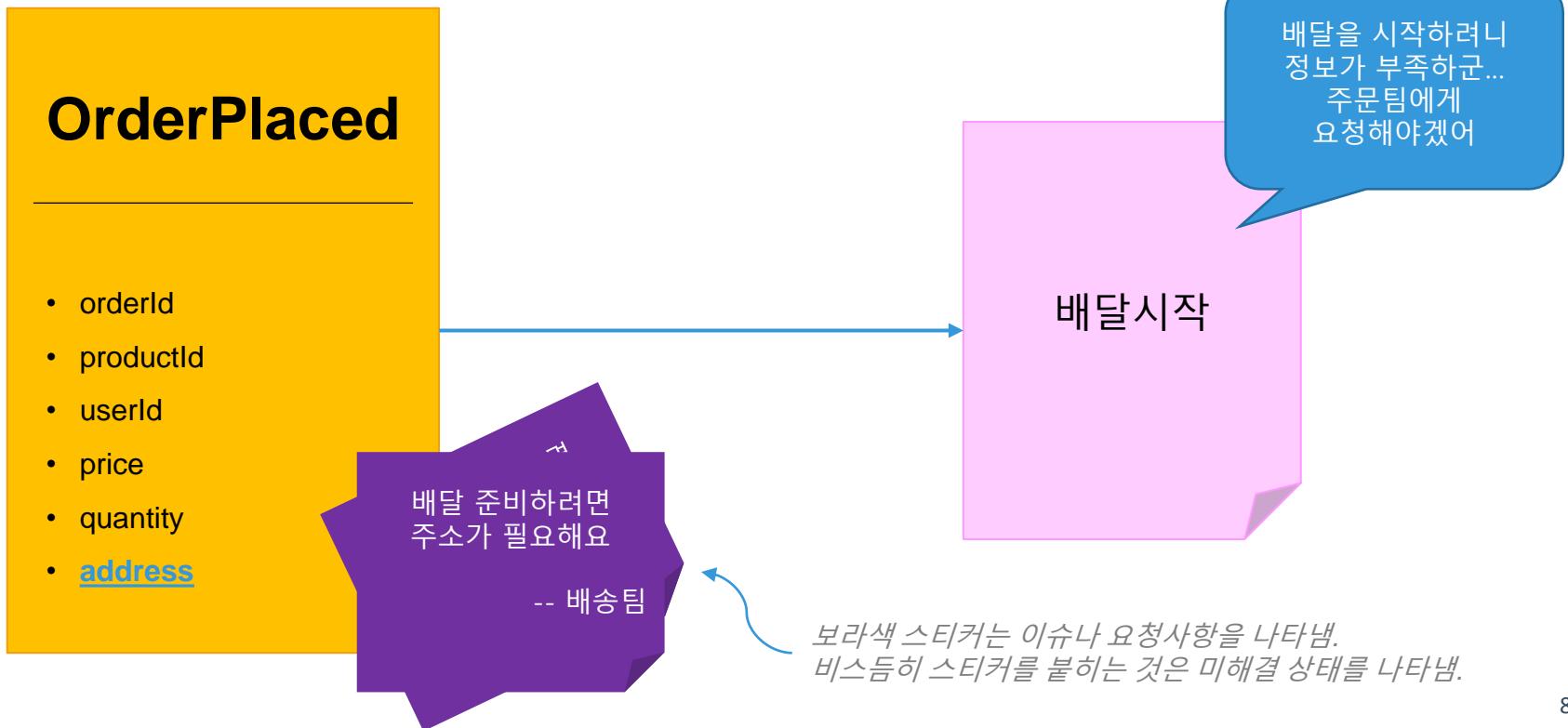
Lab Time : Bounded Context 분리 (다음스프린트)



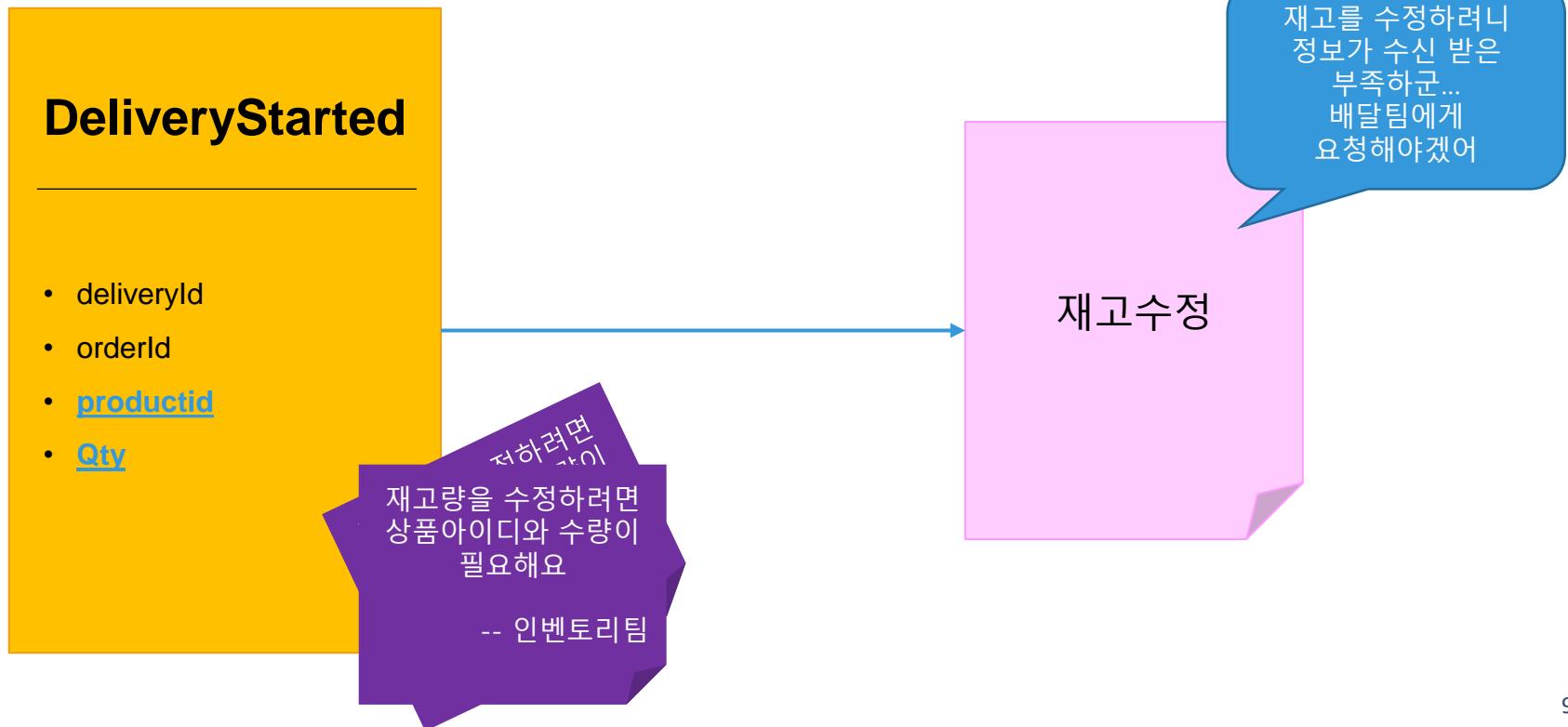
→

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

Lab Time : 도메인 이벤트의 속성 선언



Lab Time : 도메인 이벤트의 속성 선언(2)



Quiz. 다음 중 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 around Event Storming and DDD

Collaboration & Code Generation

- www.msaenz.io
- <https://docs.vlingo.io/xoom-designer>
- <https://contextmapper.org/>

Frameworks

- Xoom Vlingo Framework: <https://docs.vlingo.io/>
- Axon Framework: <https://axoniq.io/>

Collaboration Tools

- <https://miro.com>

Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS ✓
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

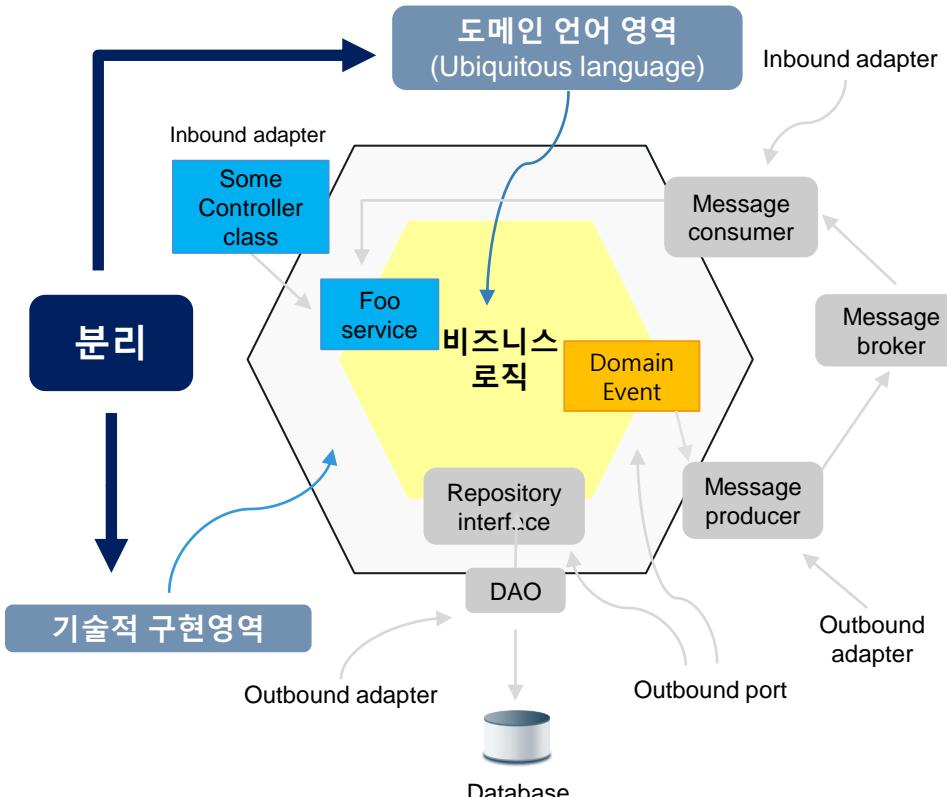
Microservice Implementation Pattern (1)

- **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.

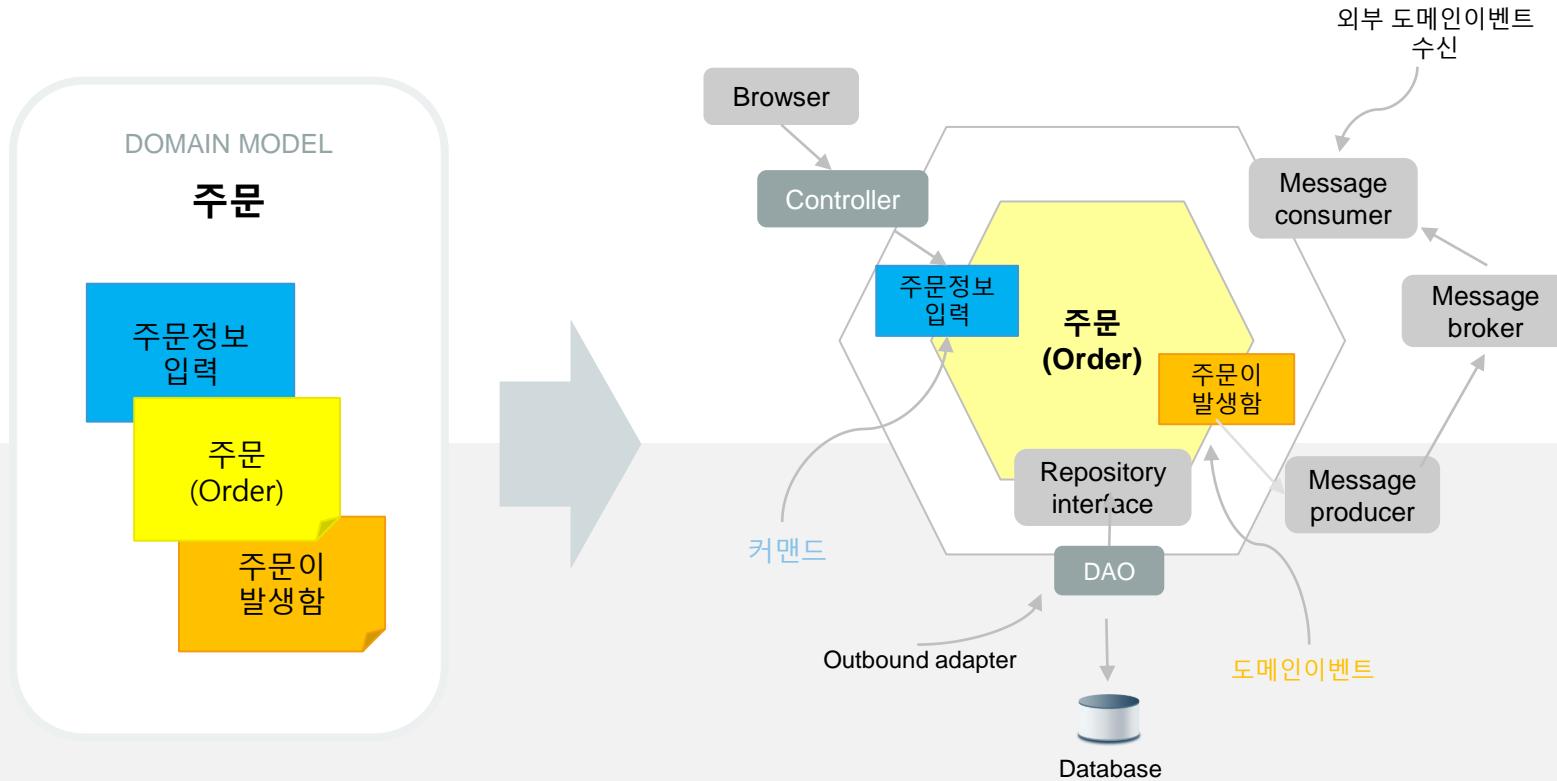
<https://alistair.cockburn.us/Hexagonal+architecture>



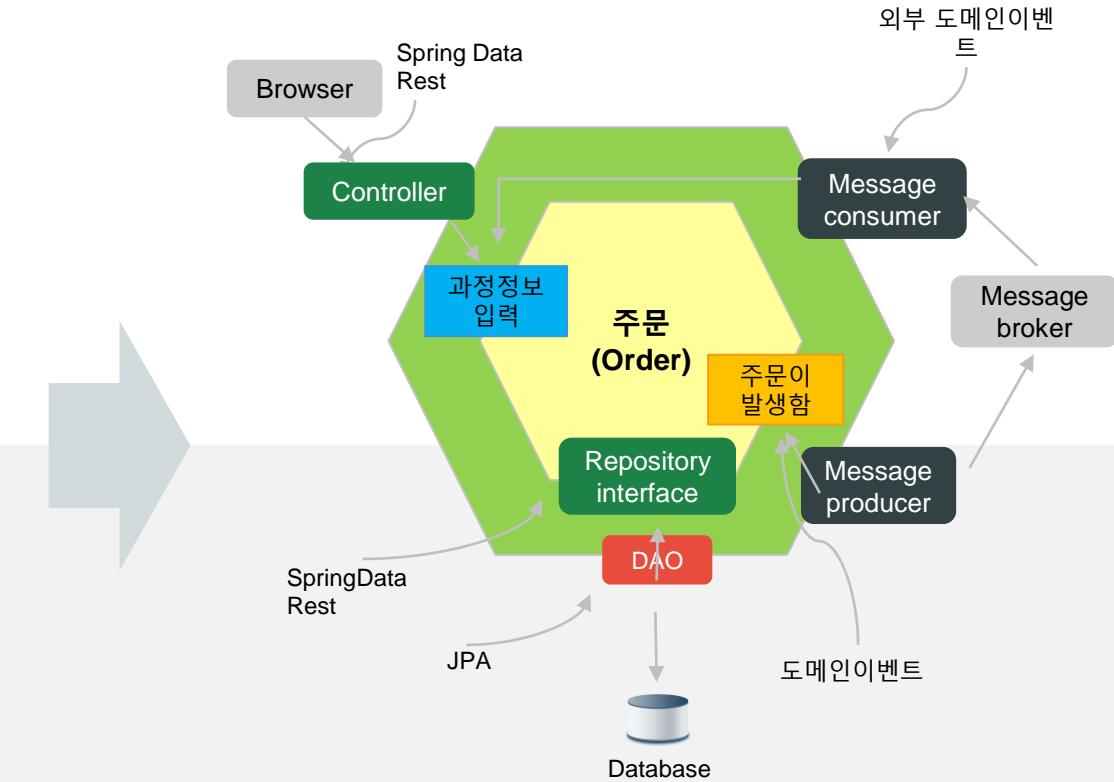
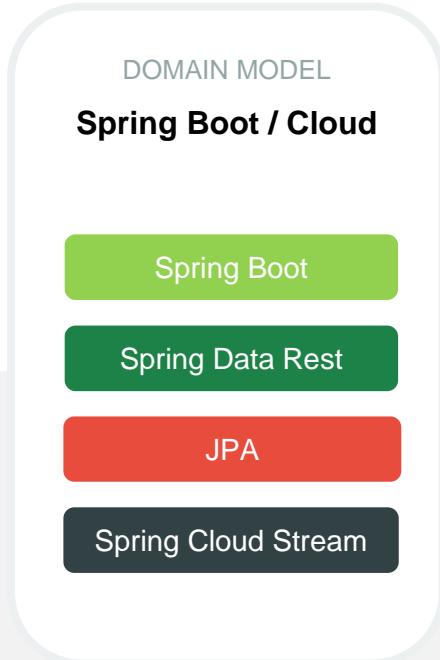
Service Implementation

- 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

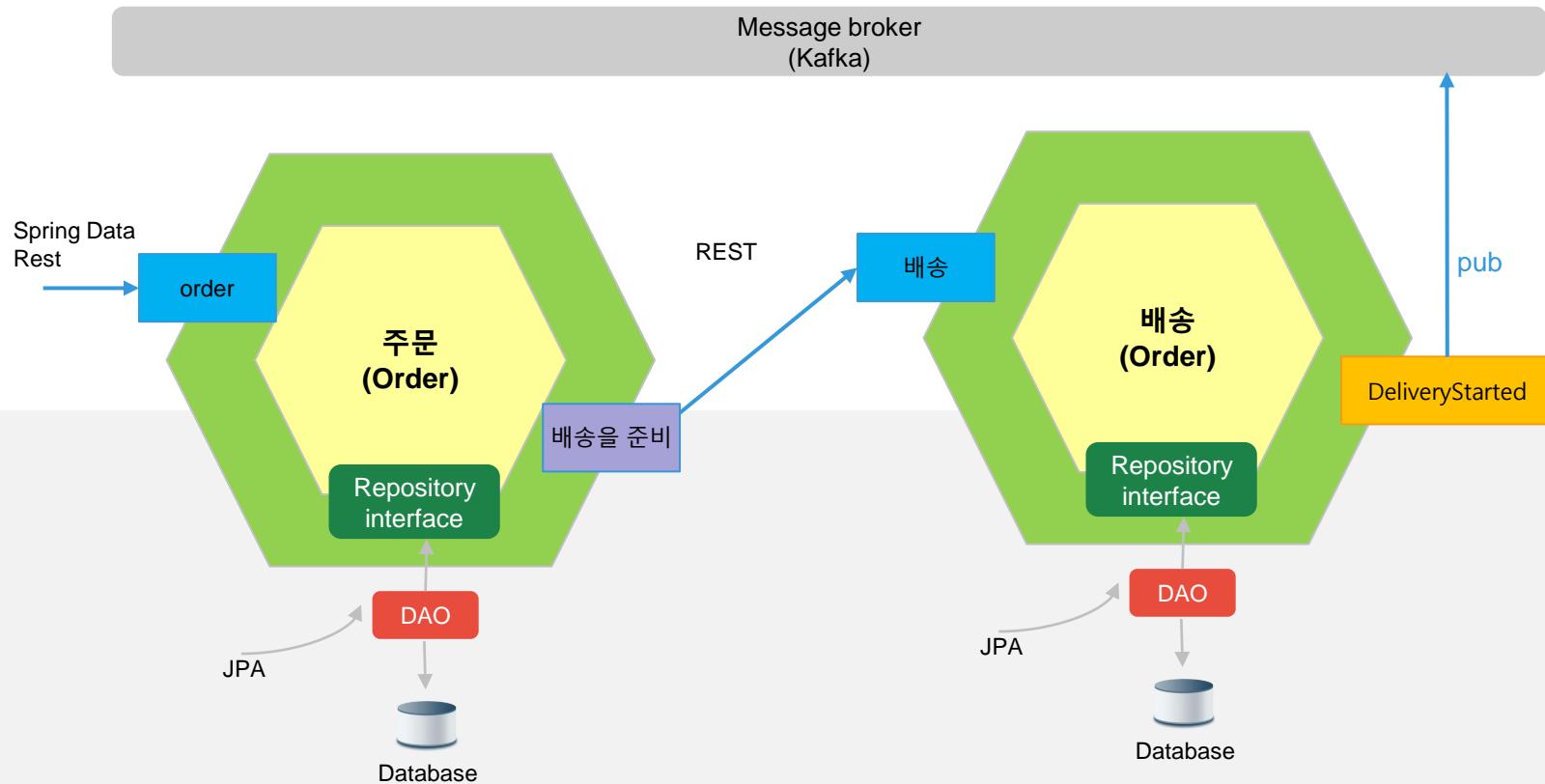
Applying Hexagonal Architecture



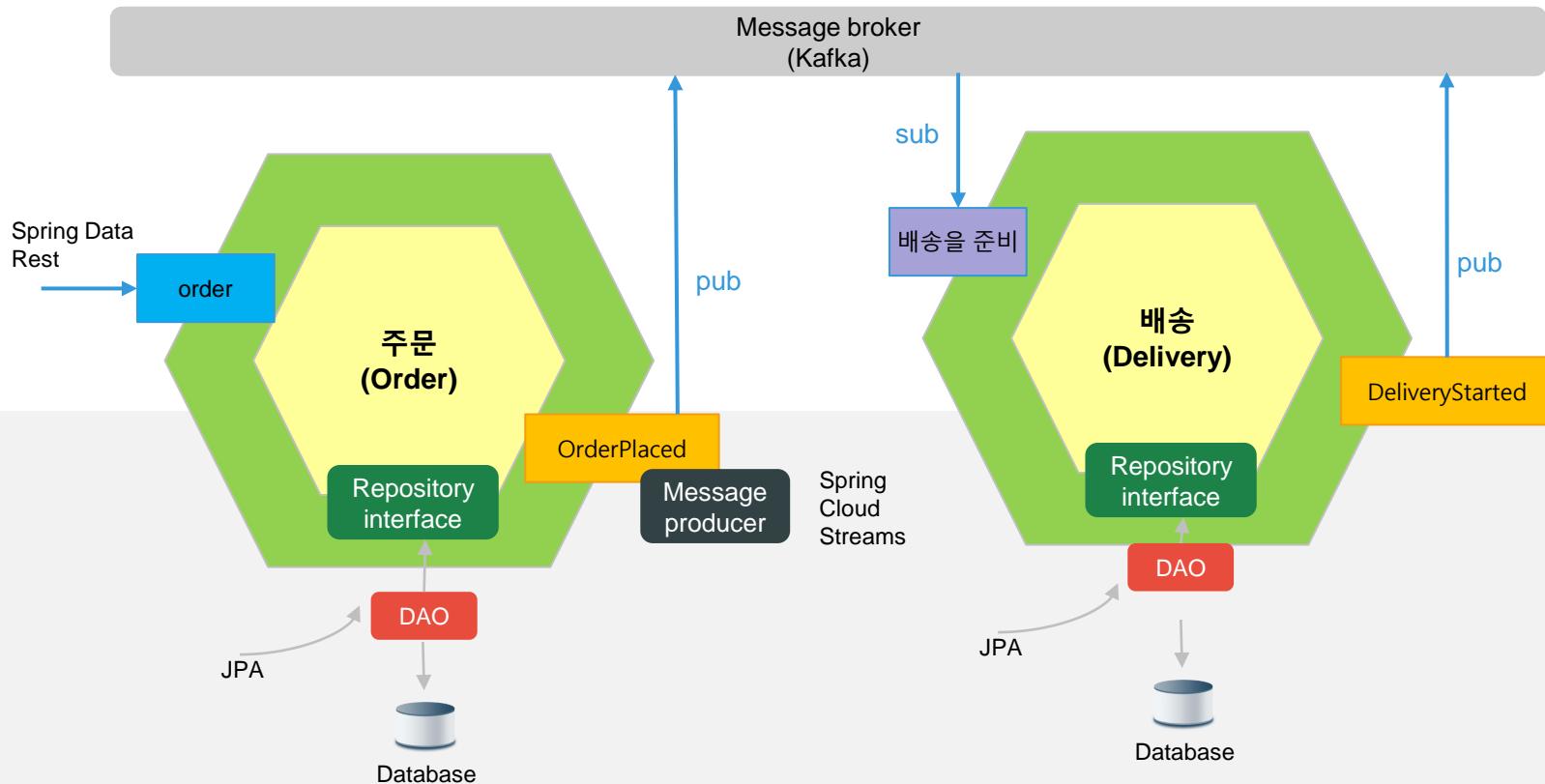
Applying MSA Chassis



Applying MSA Chassis: Request/Response



Applying MSA Chassis: Pub/Sub



이벤트 스토밍 결과에서 구현 기술 연동

- 이벤트스토밍의 결과는 곧바로 코드로 생성될 수 있는 MDD 사상을 기반함

요소	소스코드	미들웨어/프레임워크/라이브러리
바운디드 컨텍스트	<ul style="list-style-type: none">마이크로 서비스 (후보)	<ul style="list-style-type: none">Spring Boot / Quarkus / MicroprofileContainer (Docker)
이벤트 (Domain Event)	<ul style="list-style-type: none">도메인 이벤트 객체	<ul style="list-style-type: none">POJO to Serialized JSON카프카, Rabbit MQ, Active MQ ...CDC (Change Data Capturing)
커맨드 (Command)	<ul style="list-style-type: none">어그리거트의 진입메서드 (Port 메서드)컨트롤러의 인바운드 어댑터 (API)	<ul style="list-style-type: none">RestControllerSpring Data REST, Repository 패턴
결합물 (Aggregate)	<ul style="list-style-type: none">어그리거트 루트 + 도메인 클래스 묶음엔티티 (ORM)	<ul style="list-style-type: none">JPA / My batisEventuate Local, AxonSpring Cloud Stream Publisher included
정책 (Policy)	<ul style="list-style-type: none">이벤트 리스너어그리거트 내의 메서드(커맨드)를 호출	<ul style="list-style-type: none">Spring Cloud Stream ListenerAxon, Eventuate Tram
읽기모델 (Read Model)	<ul style="list-style-type: none">CQRS 의 Query PartMaterialized View Table	<ul style="list-style-type: none">Axon, Eventuate Local

Aggregate

Order

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



Aggregate Root

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

Aggregate Members

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

//Value Objects

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

Repository

```
public interface OrderRepository extends JpaRepository<Order, Long>{  
    // leave it blank  
}
```



Command

주문취소



```
@Service
public class ProductService {

    @RequestMapping(
        method = RequestMethod.DELETE,
        path = "orders/{orderId}/"
    )

    public void cancelOrder(@PathVariable("productId") Long orderId){

        orderRepository.findById(orderId).ifPresent(order -> {
            order.setStatus(Status.CANCELLED);
            order.setOrderDetails(null);
            orderRepository.save(order);
        })
    }
}
```

Anemic Domain Model

Command

주문취소

Command → Aggregate 내부의 메서드

```
@Entity  
public class Order {  
    ...  
    protected void setStatus(Status status){...}  
  
    public void cancelOrder() {  
        setOrderDetails(null);  
        setStatus(Status.CANCELLED);  
    }  
}
```

Command

주문취소



```
@Service  
public class ProductService {  
  
    @RequestMapping(  
        method = RequestMethod.DELETE,  
        path="/orders/{orderId}"  
    )  
  
    public void cancelOrder(@PathVariable("productId") Long orderId){  
  
        orderRepositoy.findById(orderId).ifPresent(order -> {  
            order.cancelOrder();  
            orderRepository.save(order);  
        })  
  
    }  
}
```

Domain Event

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 ...  
  
}
```

카
프
카

실행 (JSON)

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

Publishing Domain Events

Order

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

- publishOrderPlaced()

Event → 이벤트 발사로직

```
@Entity  
public class Order {  
    ...  
  
    @PostPersist // 주문이 저장된 후에  
    private void publishOrderPlaced() {  
        OrderPlaced orderPlaced = new OrderPlaced();  
        orderPlaced.setOrderId(id);  
        ...  
        orderPlaced.publish();  
    }  
}
```



Subscribing Domain Events

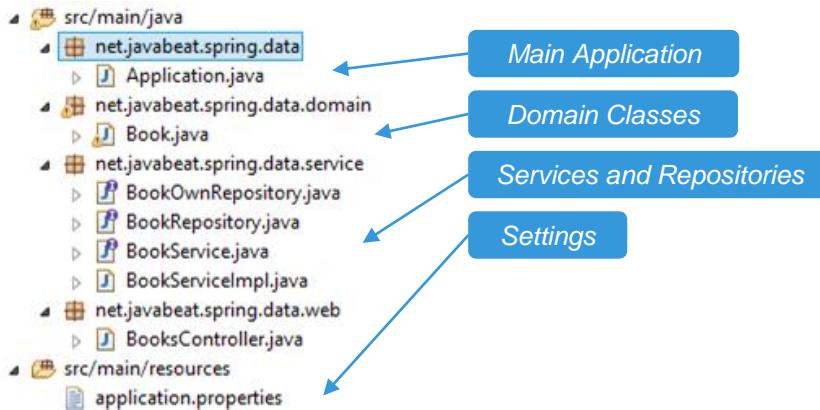
(Whenever OrderPlaced)

배송을 준비함

```
@StreamListener(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
```

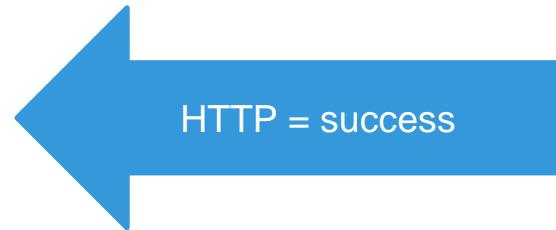
HTTP/1.1 200

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 → 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

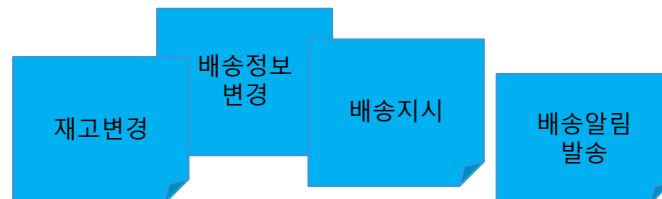
비즈니스 객체(명사)에서 추출 가능한 CRUD의 경우



```
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"
    }
  }
}
```



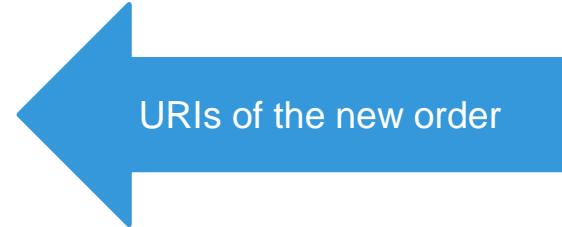
HATEOAS links

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"  
    }  
},  
  "customerId": "1@uengine.org",  
  "price": 10000,  
  "productId": 1,  
  "productName": "TV",  

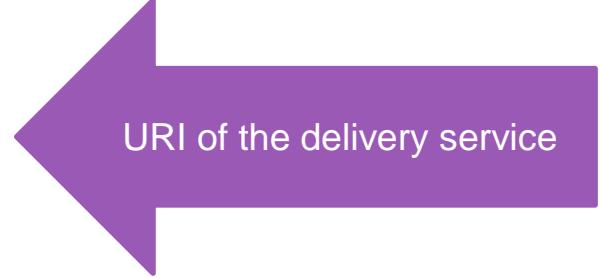
```



Create a delivery for the order

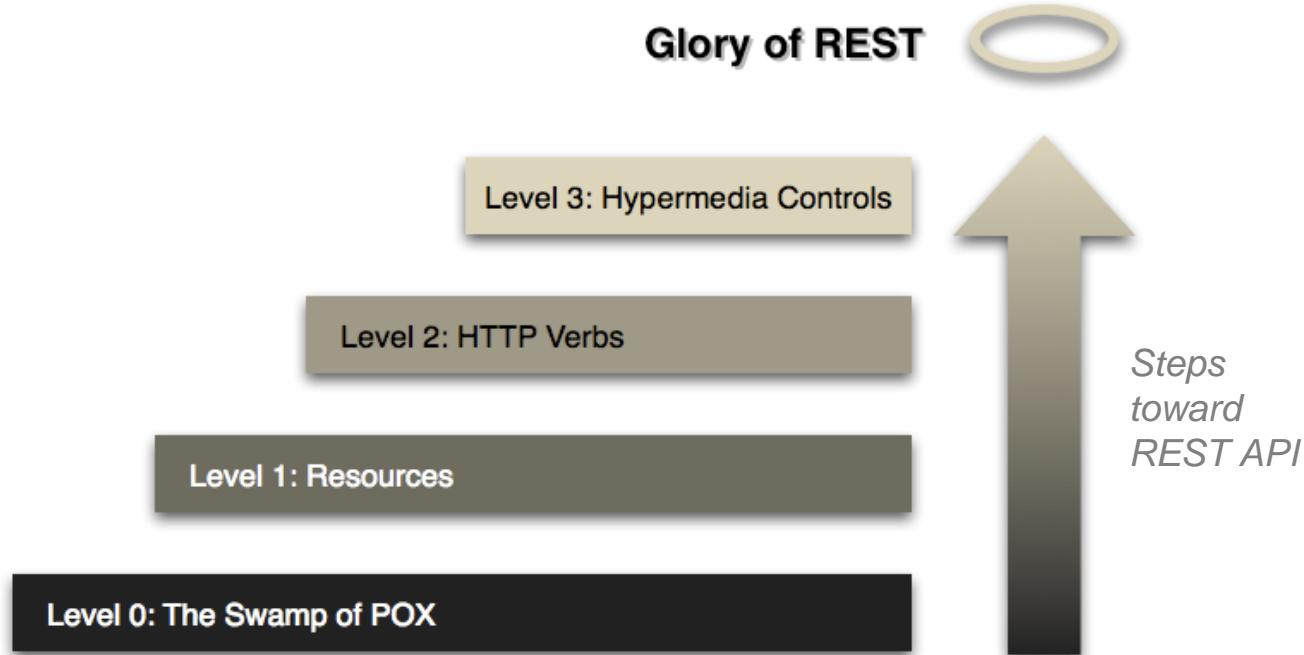
```
$ http http://localhost:8088/orders/1/delivery
```

```
{  
  "_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  
}
```



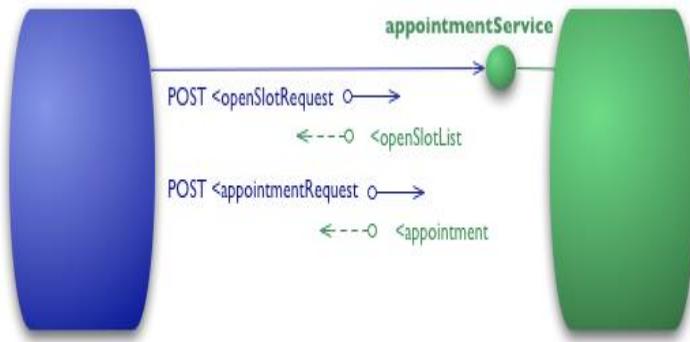
URI of the delivery service

Tip: REST Maturity Model



Level 0: Swamp of POX

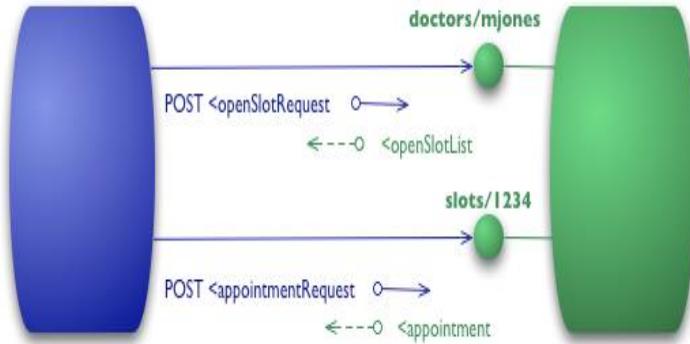
- 웹 매커니즘을 사용하지 않고, HTTP를 원격 호출을 위한 전송 시스템으로 사용하는 경우로 RPC(Remote Procedure Call)처럼 리소스 구분 없이 설계된 HTTP API
- 하나의 End-point를 사용해서 HTTP Method도 POST만 존재하여 서로 다른 매개변수를 통해서만 여러 동작을 표현



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

Level 1: Resources

- 리소스 개념을 도입해 모든 요청을 하나의 End-point로 보내는 것이 아닌 개별 리소스와 통신
- HTTP Method는 GET과 POST만 사용하고 StatusCode는 무조건 200으로 응답



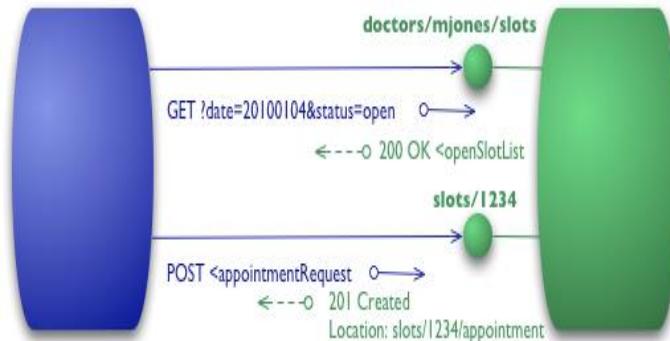
```
<openSlotList>
<slot _link = "http://openslots/1234"/>
<slot _link = "http://openslots/1235"/>
</openSlotList>
```

```
# 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

- Level2는 4가지 HTTP Method를 사용해서 CRUD를 표현하고 StatusCode도 활용하여 반환
- 현재 가장 많은 REST API가 이 단계에 해당

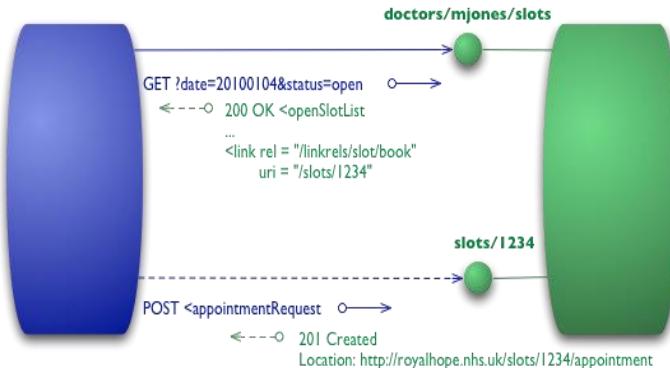


Operation	HTTP / REST
Create	PUT / POST
Read (Retrieve)	GET
Update (Modify)	PUT / PATCH
Delete (Destroy)	DELETE

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

Level 3: Hypermedia Controls

- 마지막 단계인 Level 3는 API 서비스의 모든 End-point를 최초 진입점이 되는 URI를 통해 Hypertext Link 형태로 제공
- 추가적으로 다음 Request에 필요한 End-point까지 제공하는 Uniform HATEOAS Interface



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

Level 3: In another words, HATEOAS

- Hypermedia As The Engine Of Application S 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.

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

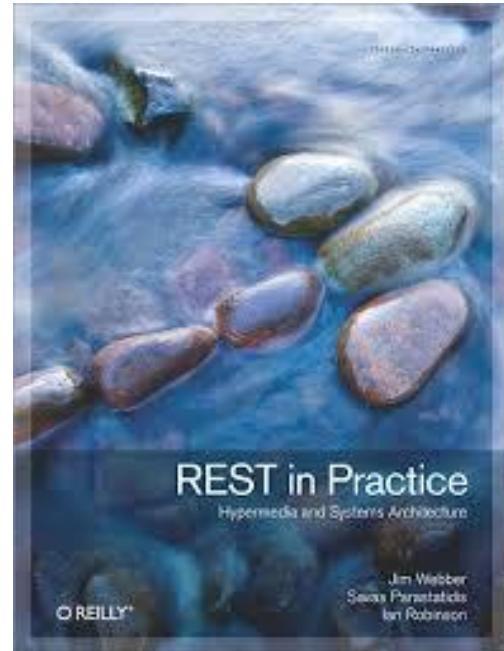
```
{  
  "_links": {  
    "delivery": {  
      "href": "http://localhost:8088/orders/1/delivery"  
    },  
    "order": {  
      "href": "http://localhost:8088/orders/1"  
    },  
    "product": {  
      "href": "http://localhost:8088/orders/1/product"  
    },  
    "self": {  
      "href": "http://localhost:8088/orders/1"  
    }  
  },  
  ...  
}
```

Wrap Up REST Maturity Model

Lvl	Name	Example Request	Response
0	Plain of XML (POX)	method: POST URI: /orders/	JSON
1	Resources with specific URI For action	method: POST URI: /orders/1/delete	JSON
2	Resources + HTTP methods	method: PATCH URI: /orders/1/delivery	JSON
3	HATEOAS (Level 2 + extra links to navigate through API)	method: PATCH URI: /orders/1/delivery	JSON with HAL (extra links)

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

Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices ✓
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

From Monolith to Microservices

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

```
src
└── main
    └── java
        └── com
            └── example
                └── template
                    └── delivery
                        └── Delivery.java
                        └── DeliveryRepository.java
                        └── DeliveryService.java
                        └── DeliveryServiceImpl.java
                        └── DeliveryStatus.java
            └── order
                └── Order.java
                └── OrderException.java
                └── OrderRepository.java
                └── OrderService.java
        └── product
            └── Product.java
            └── ProductController.java
            └── ProductOption.java
            └── ProductOptionRepository.java
            └── ProductRepository.java
            └── ProductService.java
            └── ProductServiceImpl.java
            └── Application.java
    > resources
```

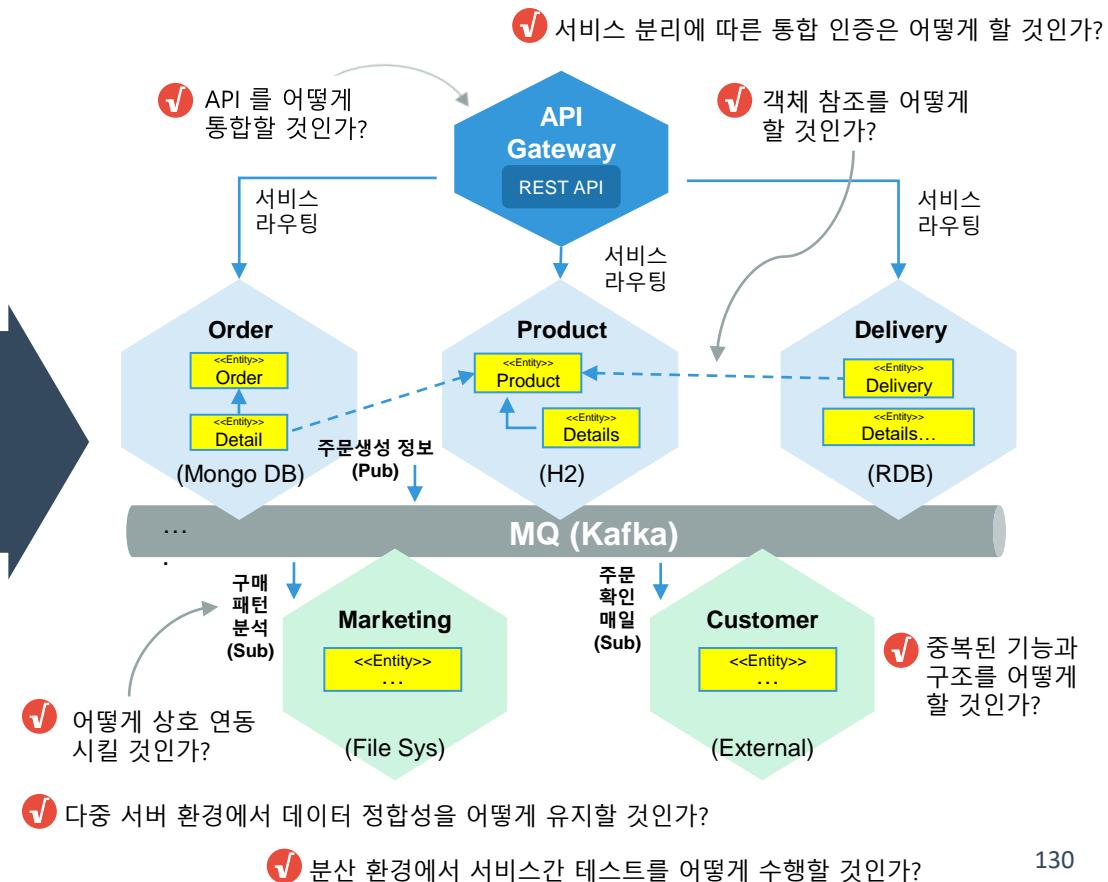
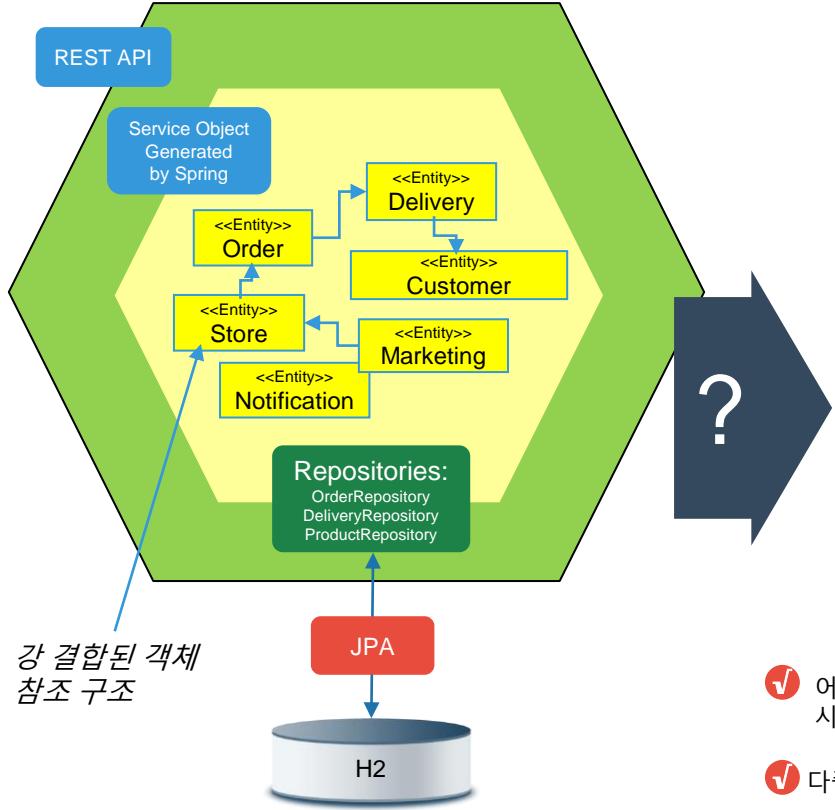
Supporting Domain
First !!

```
delivery
└── Application.java
└── Delivery.java
└── DeliveryRepository.java
└── DeliveryService.java
└── DeliveryStatus.java
└── Order.java
└── Product.java
```

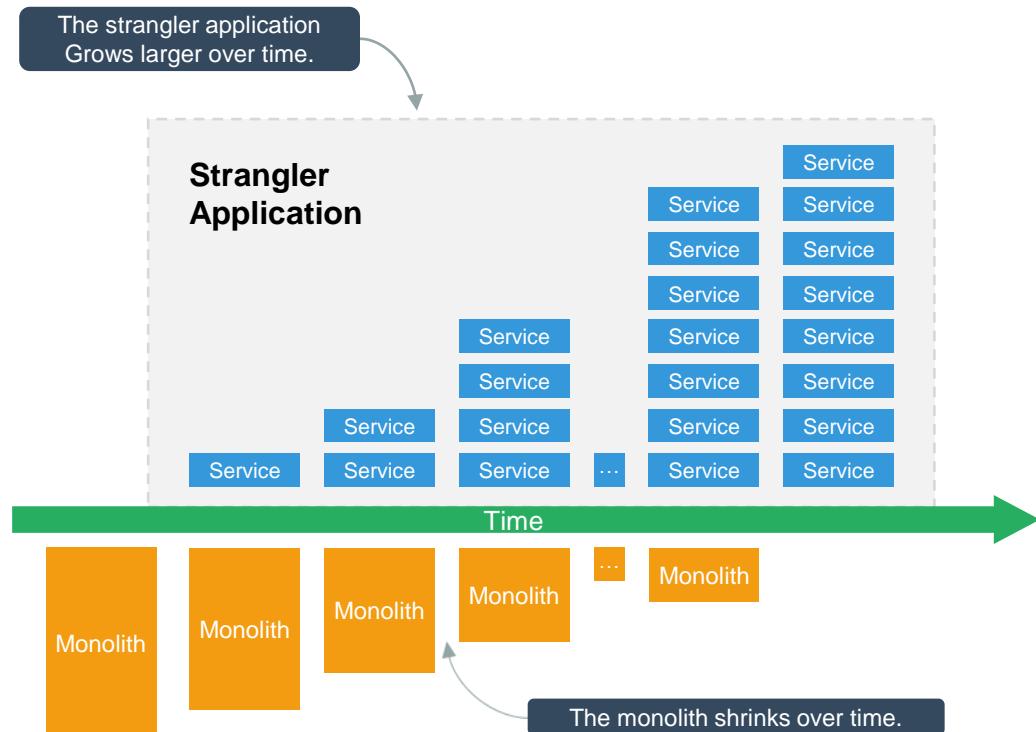
```
product
└── Application.java
└── Product.java
└── ProductRepository.java
└── ProductService.java
└── ProductStatus.java
```

```
order
└── Application.java
└── Order.java
└── OrderRepository.java
└── OrderService.java
└── OrderStatus.java
```

Issues in transforming Monolith to Microservices



Legacy Transformation – Strangler Pattern



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

(Issues #1) API 를 어떻게 통합할 것인가?

- **API Gateway**

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

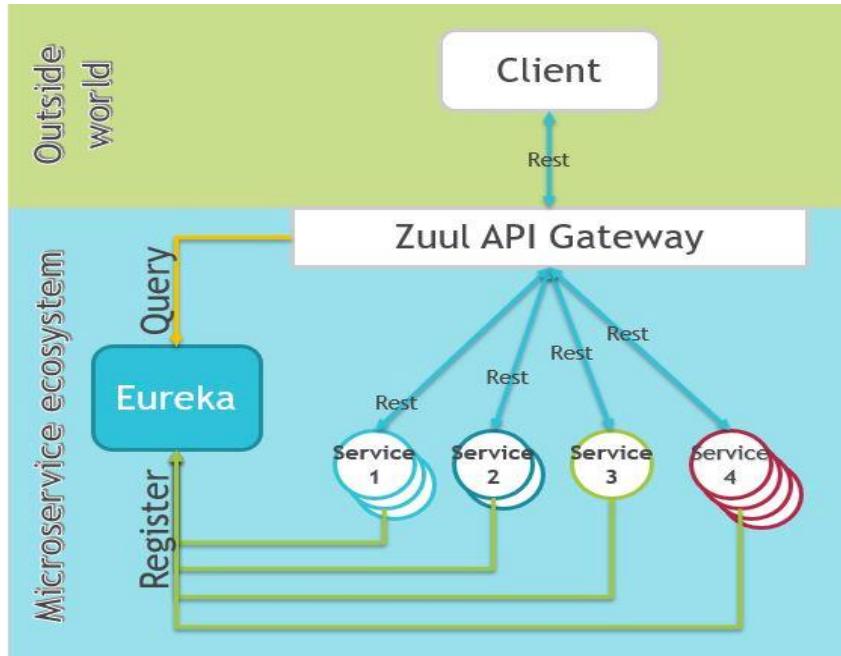
- **Service Registry**

- 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.



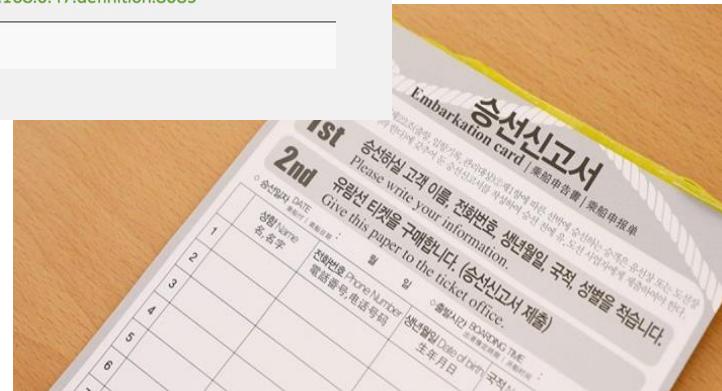
Service Registry: EUREKA or Kube-DNS

 **spring Eureka**

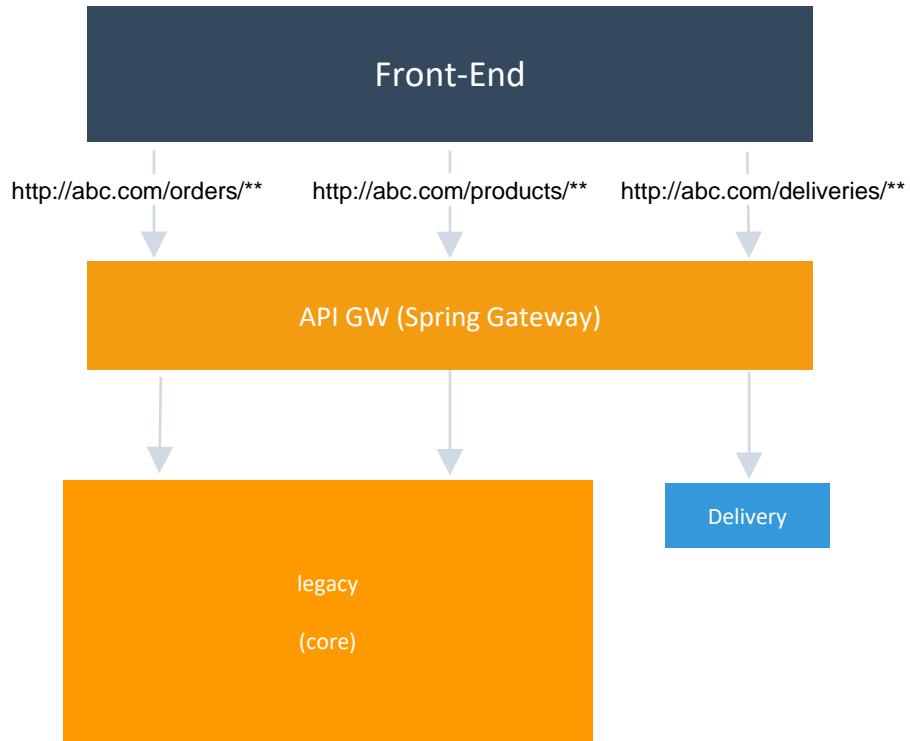
HOME LAST 1000 SINCE STARTUP

Instances currently registered with Eureka

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



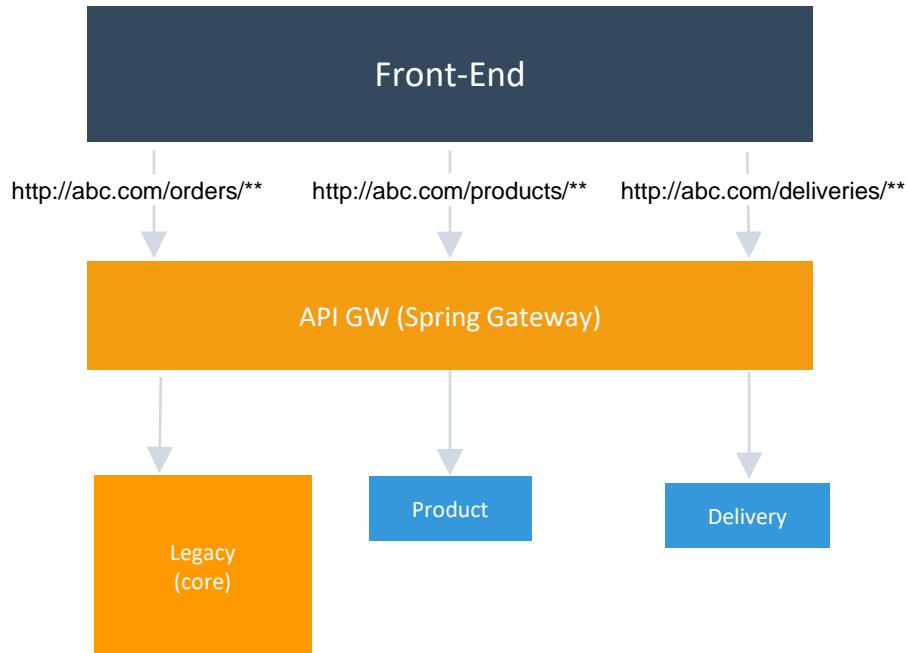
API Gateway : Strangler



#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

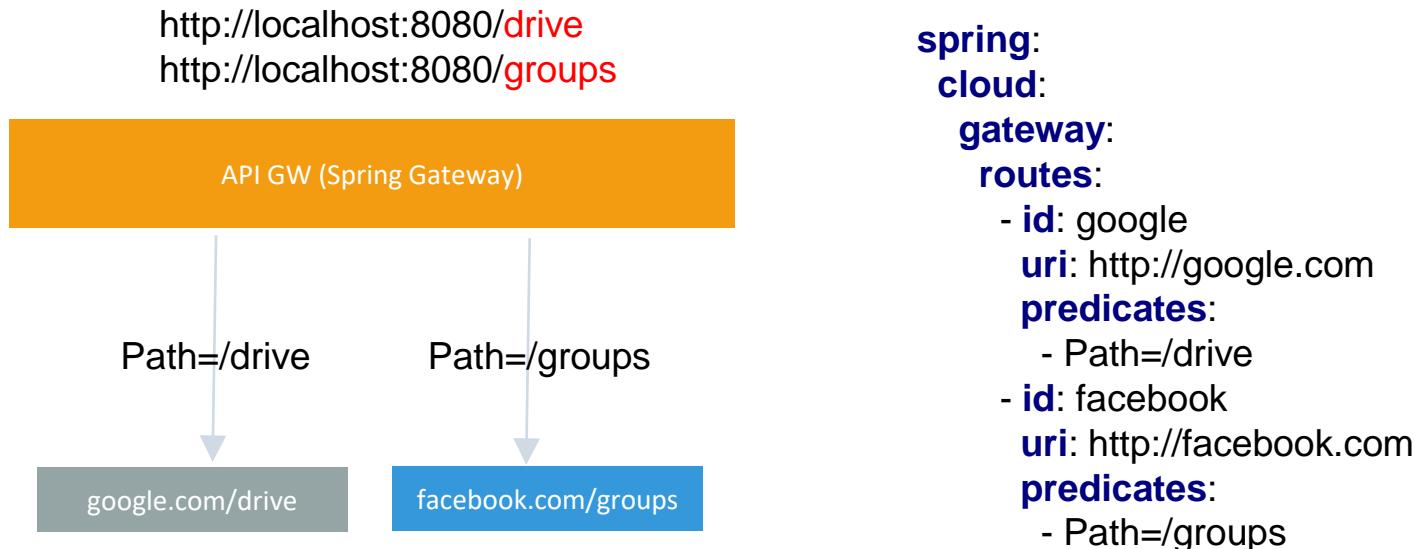


#application.yml

```
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/**
```

Lab: API Gateway – Path-based routing

- 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

- 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

API GW (Spring Gateway)

google/msa
=> /search?q=msa

naver/msa
=> /search.naver?query=msa

google.com/search?q=msa

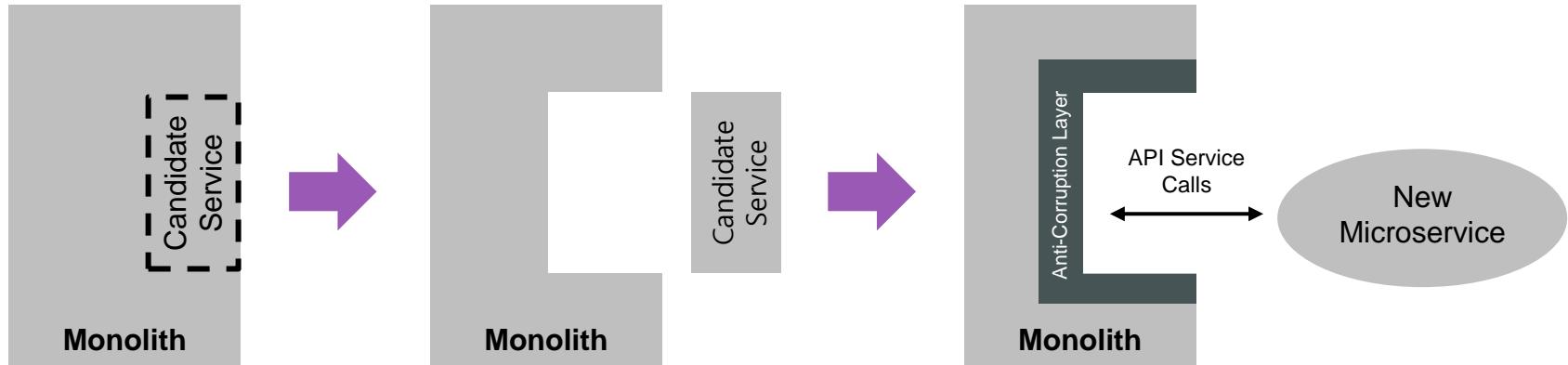
search.naver.com/search.naver?
query=msa

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

(Issues #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
→ 레거시에서 이벤트를 퍼블리시하도록 전환, 신규 서비스가 주요 비즈니스 이벤트에 대하여 반응하도록 처리

Find the seams and replace with Proxy

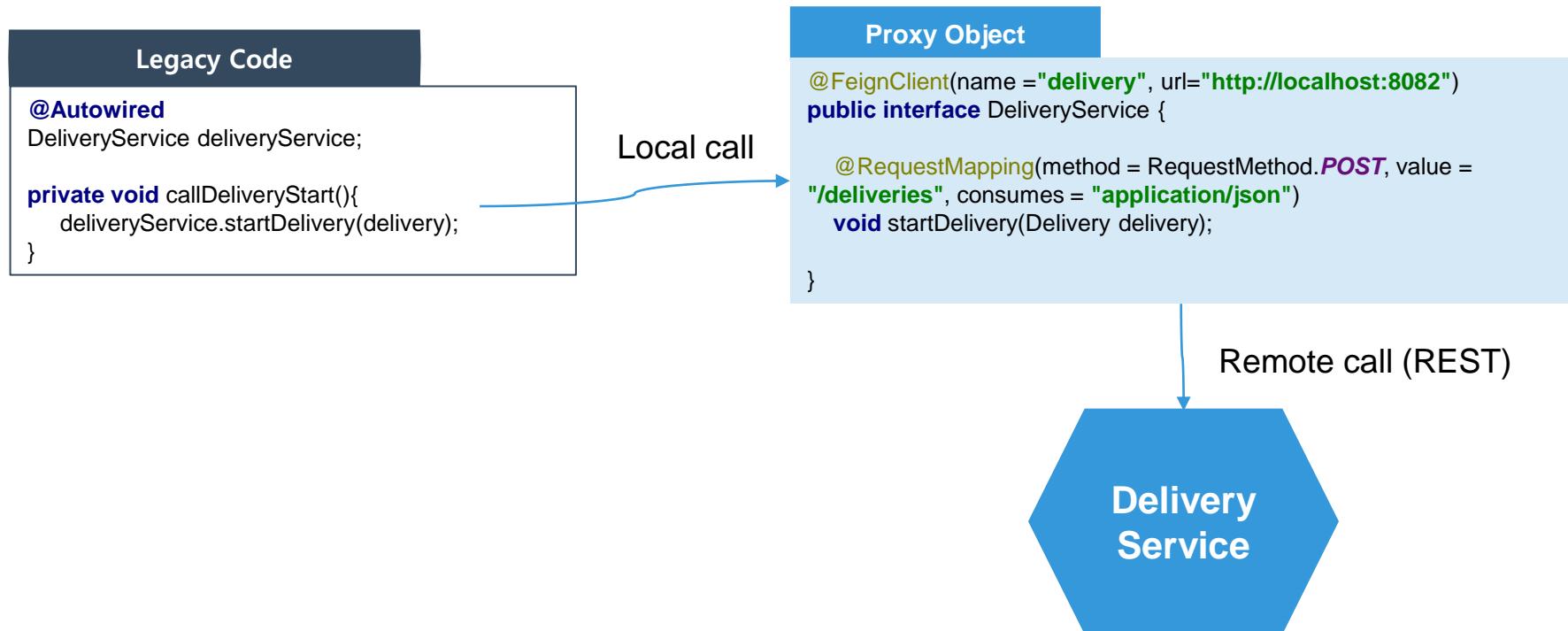


- Determine candidate service from Monolith and strangle it out
- Convert candidate service to Microservice and Add Anti-Corruption Layer to enable communication with Monolith

Candidates for Anti-Corruption Layer Proxy

HTTP Client	프로바이더	특징
HttpURLConnection	JDK에 포함되어 있는 URLConnection의 서브 클래스	<ul style="list-style-type: none">Config 설정정보가 많음Connection을 수동으로 열고 닫음
Apache Commons HttpComponents	Apache에서 만든 오픈소스 라이브러리	<ul style="list-style-type: none">안정적인 오픈소스 라이브러리다소 무겁지만 다양한 API 제공
RestTemplate	HTTP Client로 REST API를 호출위한 함수 제공 Spring 5.0 이후부터 Deprecated	<ul style="list-style-type: none">HttpClient를 추상화하여 제공Restful한 Json 요청/응답 처리 지원
WebClient	Spring에서 추천하는 방식으로 싱글 스레드 Non-Blocking 이벤트에 반응형으로 동작(Spring React 프레임워크)	<ul style="list-style-type: none">RestTemplate은 Blocking이고, WebClient는 Non-Blocking 방식
FeignClient	NetFlix에서 만든 선언적 Rest Client 라이브러리 SpringCloud 프레임워크로 흡수	<ul style="list-style-type: none">Goal : HTTP API 클라이언트 단순화RestTemplate 대체 및 코드 가독성 우수

Proxy using FeignClient



Lab: Proxy with FeignClient (1)

- 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 (2)

- 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

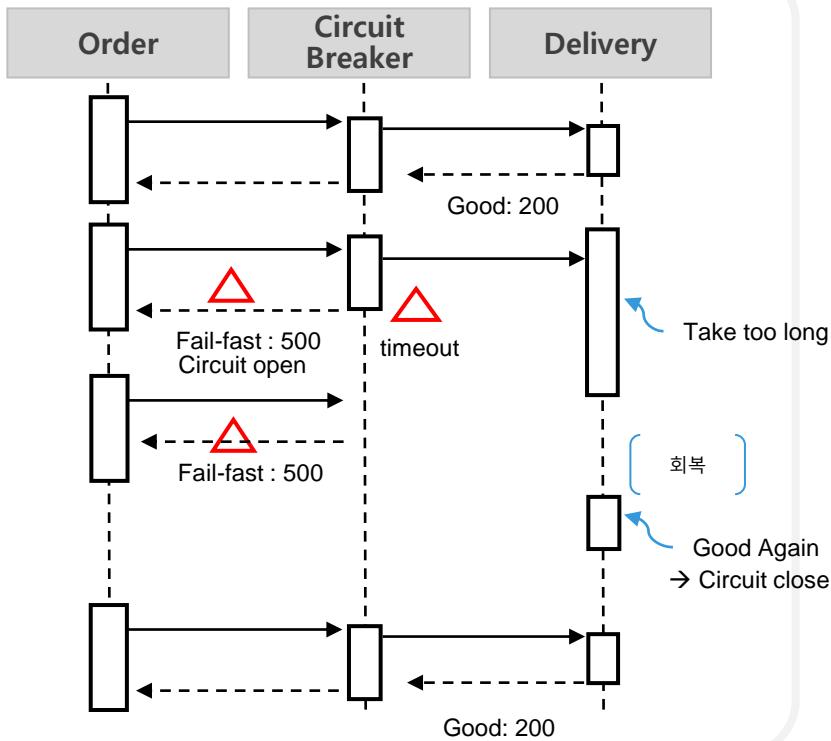
- 주문 후 변경된 상품 수량 확인

http http://localhost:8088/orders/1/product

- 프로젝트 원복

git reset --hard

동기호출에 따른 장애전파 우회: 서킷브레이커 패턴



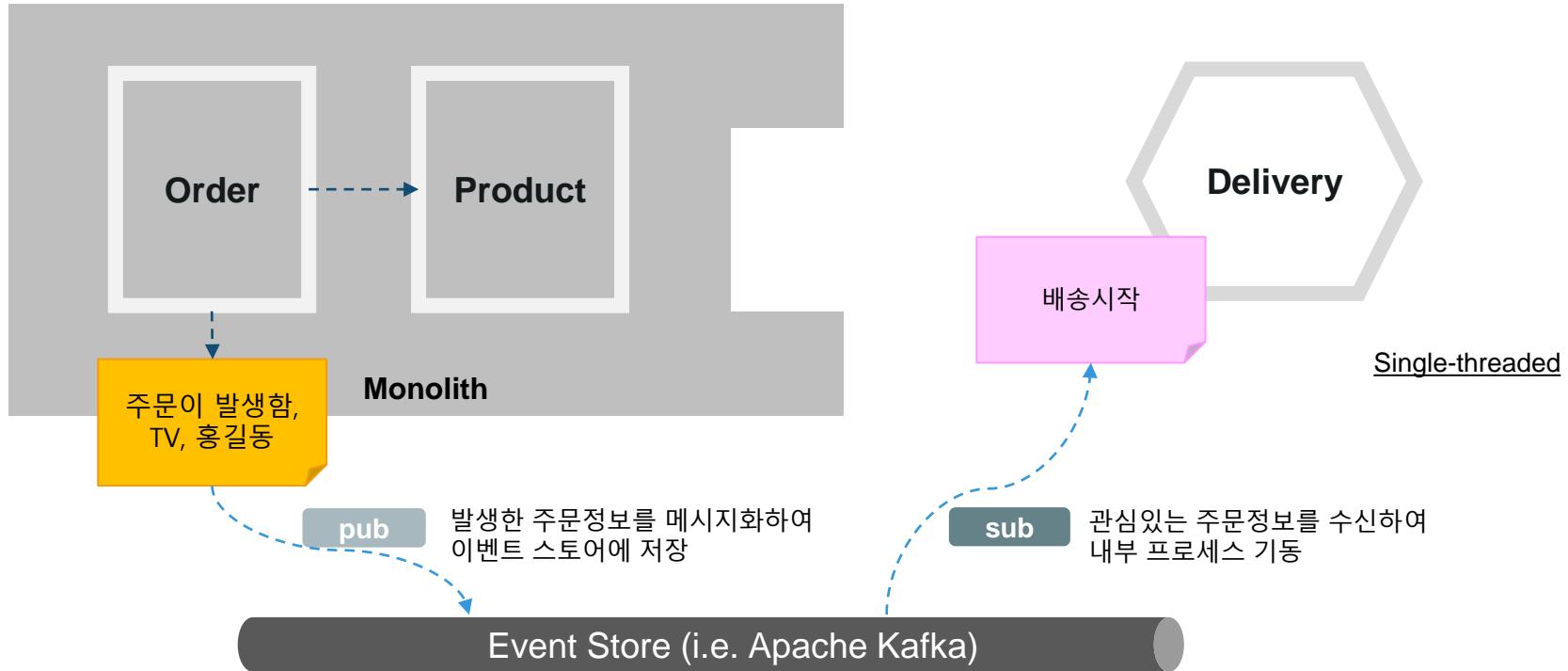
하나의 트랜잭션이 가능한 블로킹이
발생하지 않도록 미리 차단, Fail-Fast 전략

→ 메모리 사용 폭주 막음



Event Shunting

- Communication between monolith and microservice via event store



SAGA Pattern

saga 미국·영국 ['sa:gə]  영국식 

1. [명사] (특히 노르웨이·아이슬란드의) 영웅 전설
2. [동사] 대하소설
3. [명사] 일련의 사건 [모험] (또는 그에 대한 보도)

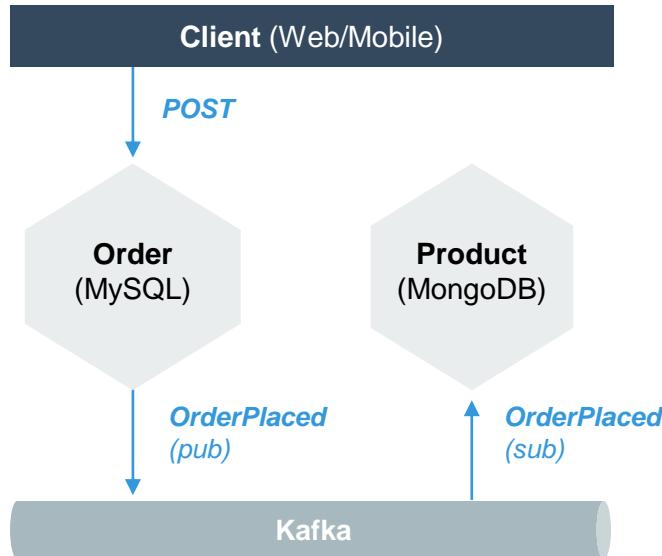
SAGA 패턴이란, 마이크로서비스들끼리 이벤트를 주고 받아 특정 마이크로서비스에서의 작업이 실패하면 이전까지의 작업이 완료된 마이크서비스들에게 보상 (complementary) 이벤트를 소싱함으로써 분산 환경에서 원자성(atomicity)을 보장하는 패턴입니다.

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 ~~

<https://microservices.io/patterns/data/saga.html>

Saga를 활용한 Eventual Transaction

- 서비스구성 (Eventual TX)

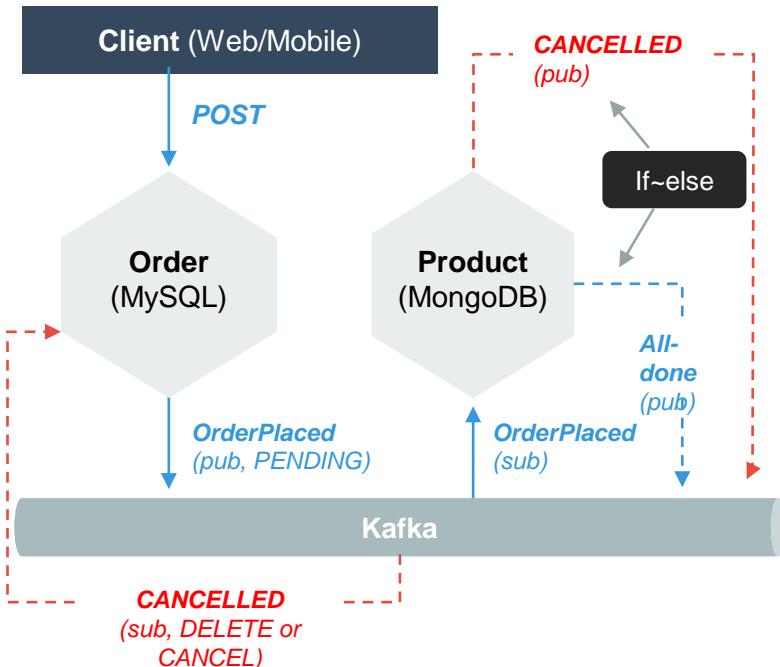


- 조회화면



Rollback in Saga – Compensation transaction

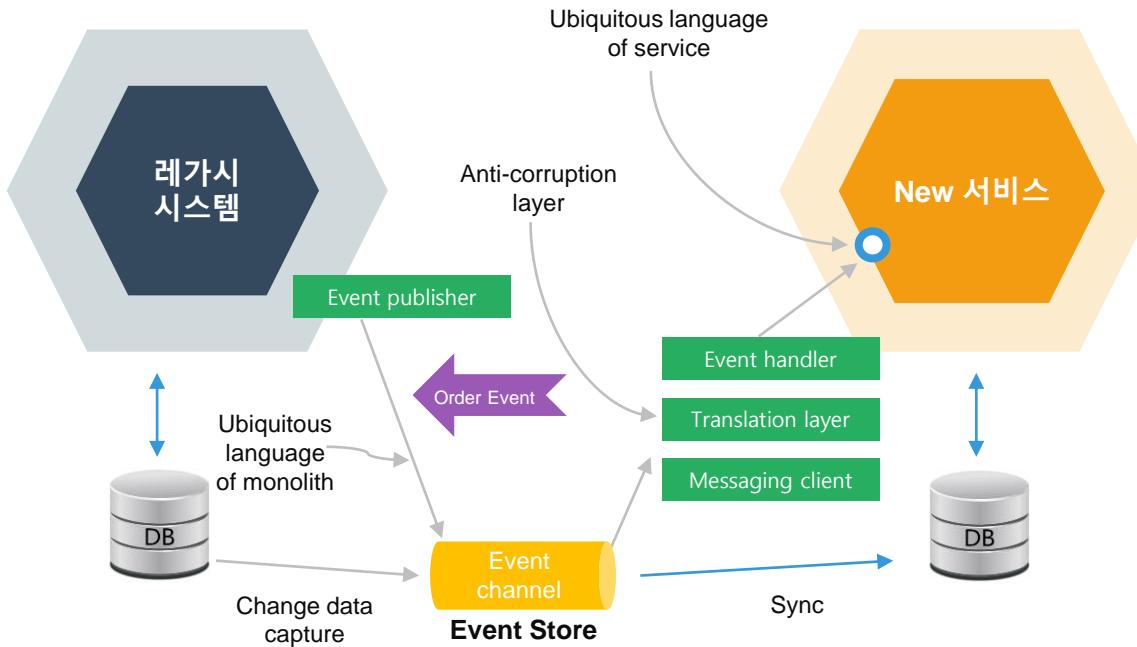
- 서비스구성 (Eventual TX)



- 조회화면



Event Shunting in Persistence Layer



- **Aggregate 내 코드 주입**

→ 호출 코드 직접 주입, Hexagonal Architecture 의 손상
→ JPA 의 Lifecycle Annotation 사용

- **CDC (Change Data Capturing) 가능**

→ DB 의 Change Log 를 Listening, Event 자동퍼블리시 하는 툴
→ Debezium, Eventuate Tram 등이 존재

Event Shunting Chassis

- **Event Store**
 - 이벤트는 추가만 가능하고 수정 및 삭제가 불가능 (Append ONLY)
 - 이벤트 소싱*(Event Sourcing)기반 데이터 퍼시스턴시 및 SnapShot 지원
 - 처리되지 않은 이벤트를 위한 Retry와 DLQ(Dead Letter Queue)
- **Event sourcing Framework – Application Layer**
 - Client에서는 동일 이벤트를 중복처리 하지 않는 멱등성(Idempotence) 보장
 - 비즈 로직과 이벤트간의 Atomic 트랜잭션(Trx) 보장
 - 이벤트기반 시스템이 스케일 아웃에 따른 동시성 처리 보장
- **CDC tools – Persistence Layer**
 - CDC는 데이터베이스 내 데이터에 대한 변경을 식별해 필요한 후속처리(데이터 전송/공유 등)를 자동화하는 기술 또는 이를 구현하는 데이터 파이프라인 도구

* Event Sourcing Pattern은 발생하는 모든 Event를 저장하고, 저장한 Event를 바탕으로 Data 조작을 수행하는 Pattern

Event Stores

Type	특징	단점
RDBMS	<ul style="list-style-type: none"> Transaction에 대한 지원 및 기술적 성숙도가 높음 	<ul style="list-style-type: none"> 확장성에 따른 성능 이슈 Event Notification 기능 부재
Mongo DB	<ul style="list-style-type: none"> NoSQL 기반 대표적 스토어로 하나의 이벤트는 하나의 Document 해당되고 대량의 데이터 처리에 적합 	<ul style="list-style-type: none"> Transaction 지원 문제점 Event Notification 기능 부재
Apache Kafka	<ul style="list-style-type: none"> 대용량 실시간 Message Broker에 적합 파일시스템에 메시지를 저장함으로써 영속성 보장 	<ul style="list-style-type: none"> 관리 및 모니터링 도구 부재 Event Sourcing 처리에는 부적합
Axon Server	<ul style="list-style-type: none"> Event Sourcing 기반 데이터 적재 EventStore와 Event Borker의 두 역할 수행 	<ul style="list-style-type: none"> 엔터프라이즈 적용(HA)을 위한 상용 버전
AWS MSK	<ul style="list-style-type: none"> Amazon Managed Streaming for Apache Kafka (Apache Kafka 및 Kafka Connect 클러스터 제공) 	<ul style="list-style-type: none"> Vanilla Apache Kafka와 동일

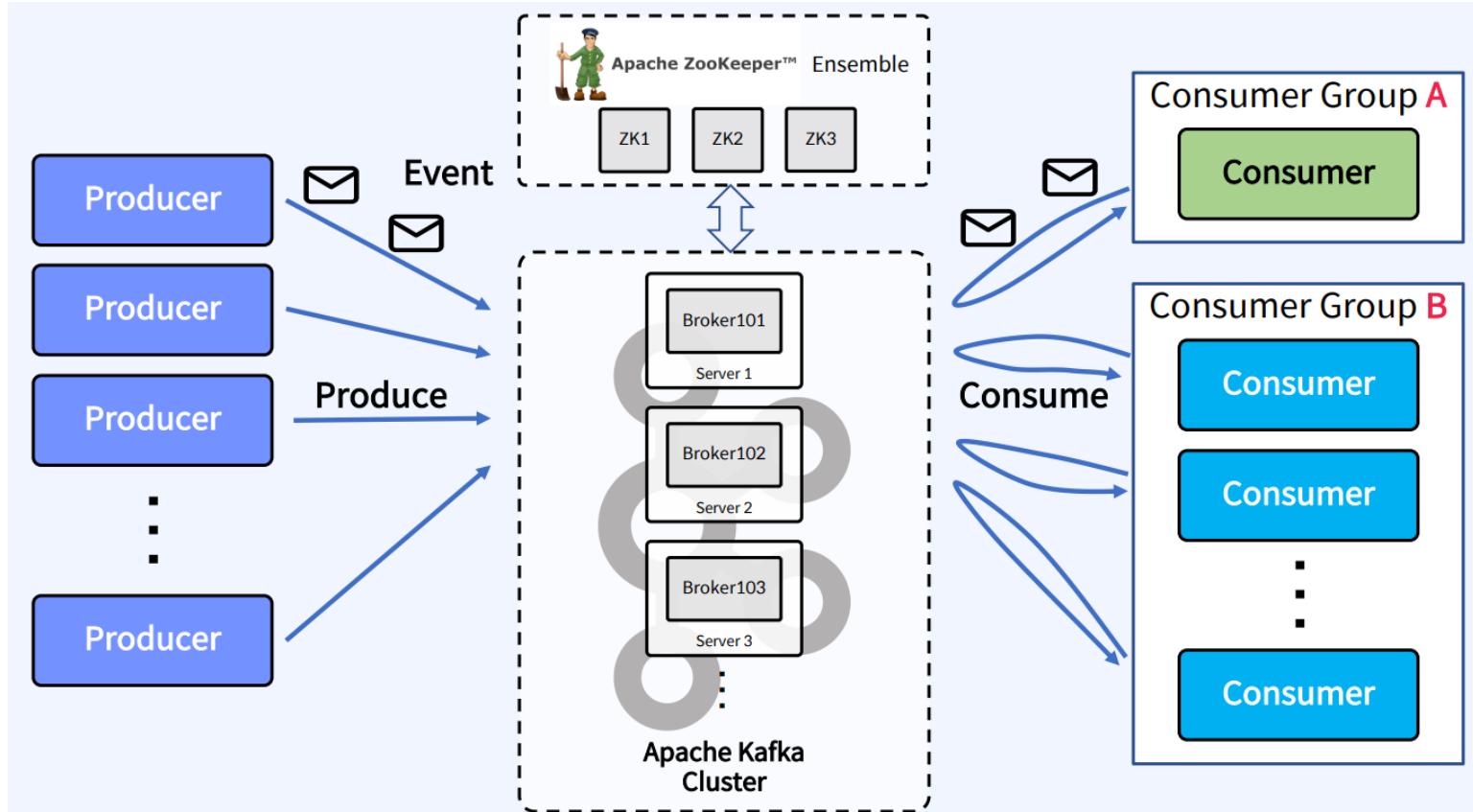
“

Event Store – Apache 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

카프카 아키텍처



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>

Topics, Partitions, Offsets

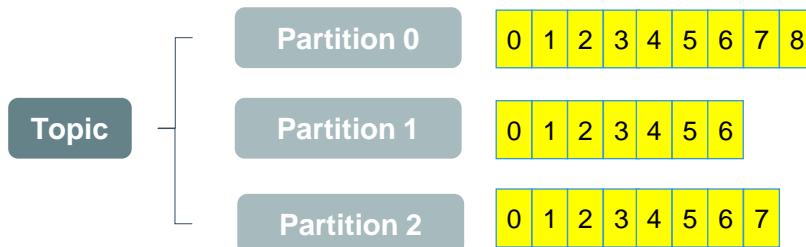
- 토픽

- 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

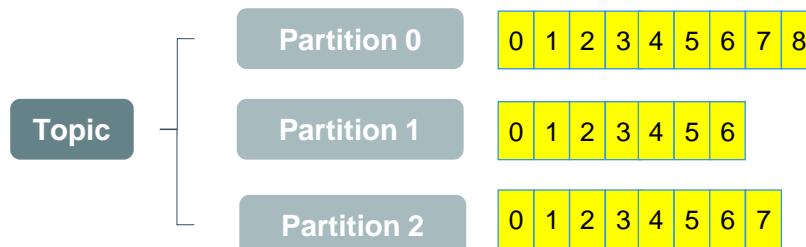
- 토픽은 파티션을 나눌 수 있음
- 토픽당 데이터를 분산 처리하는 단위
- 병렬 처리와 많은 양의 데이터 처리를 위해서 파티션을 늘리 수 있음
- **늘리기만 하고 줄이는 것은 안됨**
- 각각의 파티션은 0부터 1, 2, ... 의 Index를 가짐
- 파티션 별로 증가되는 아이디, 즉 Offset라는 값을 가지게 됨



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)
- 주키퍼 클러스터는 홀수로 동작함 (Leader/Follower 선출때문에 짝수로 동작 못함)
- 주키퍼 Leader 는 write, Follower 는 read 작업을 함

* Zookeeper will be deprecated

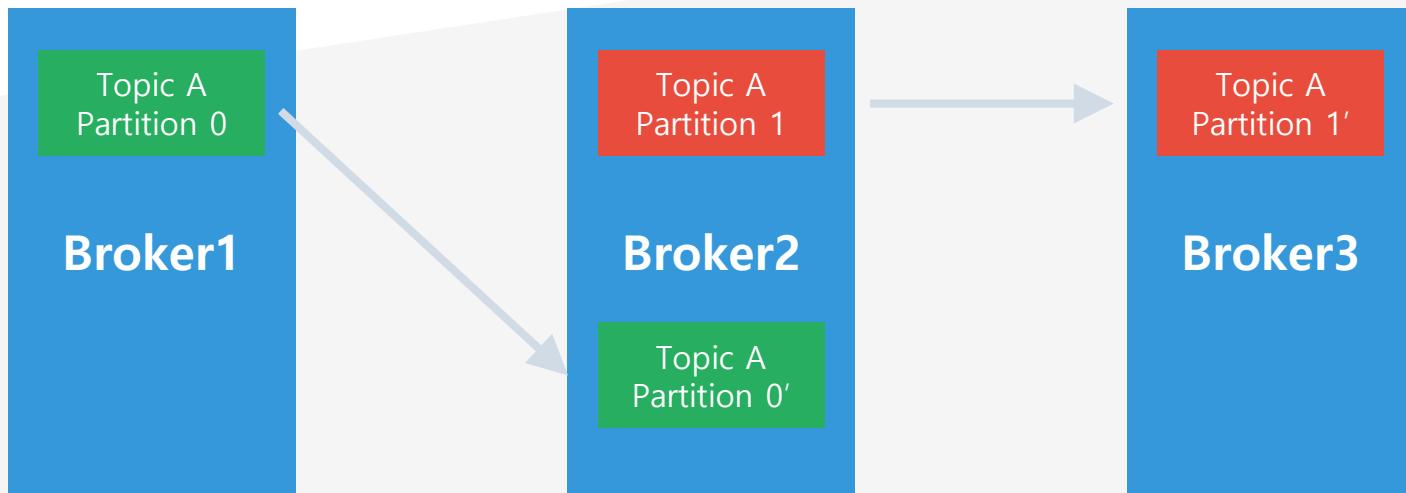
ZooKeeper adds an extra layer of management. Kafka 3.3 would include options for both ZooKeeper and KRaft. ZooKeeper would be deprecated in the release after that, and removed in Kafka 4.0.

Kafka Raft, or KRaft, a protocol for internally managed metadata, will replace ZooKeeper.

출처 : <https://www.infoworld.com/article/3660073/why-apache-kafka-is-dropping-zookeeper-for-kraft.html>

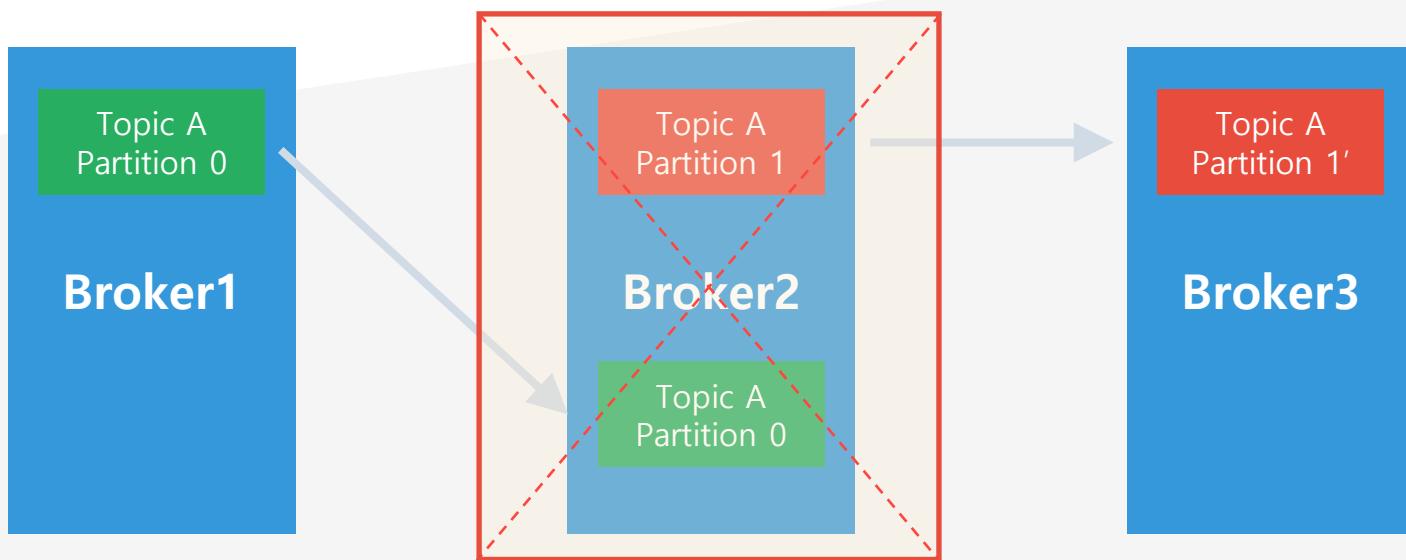
Topic Replication factor

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



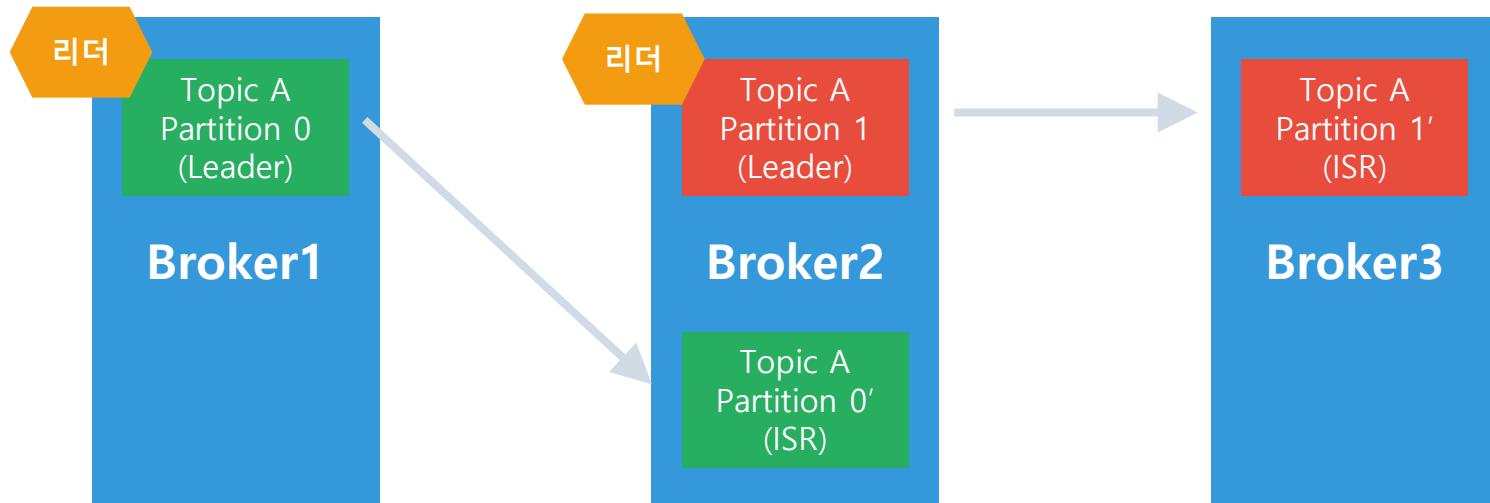
Topic Replication factor

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



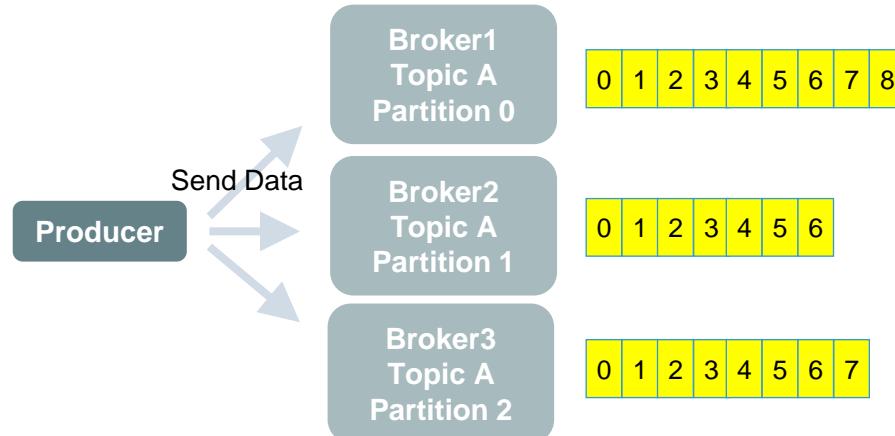
파티션의 Leader

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



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.

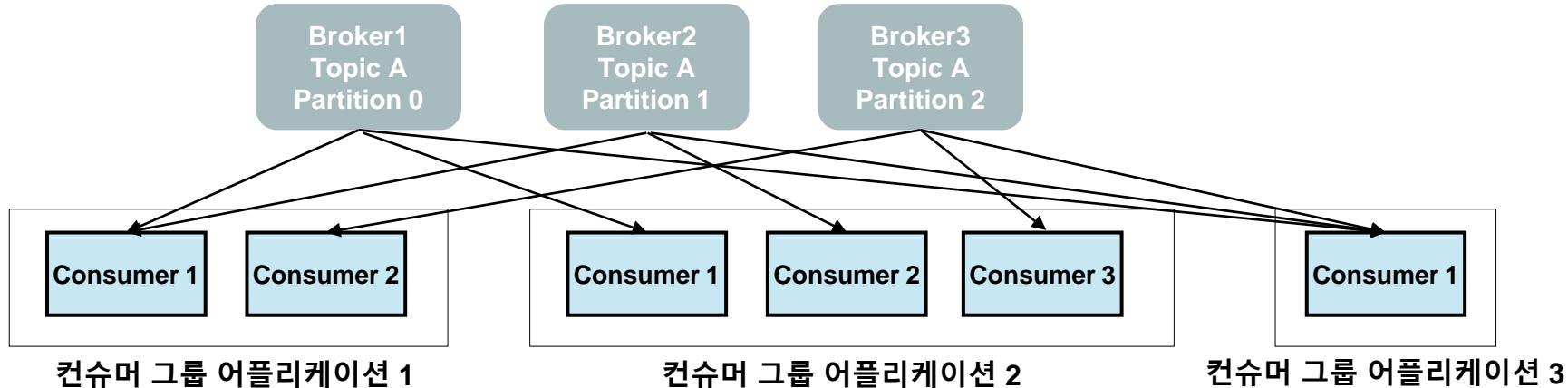


Producer

- 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.

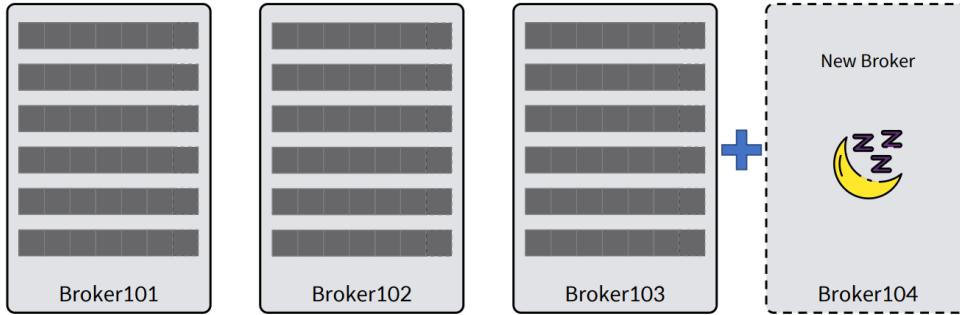


Consumer Offset

- 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



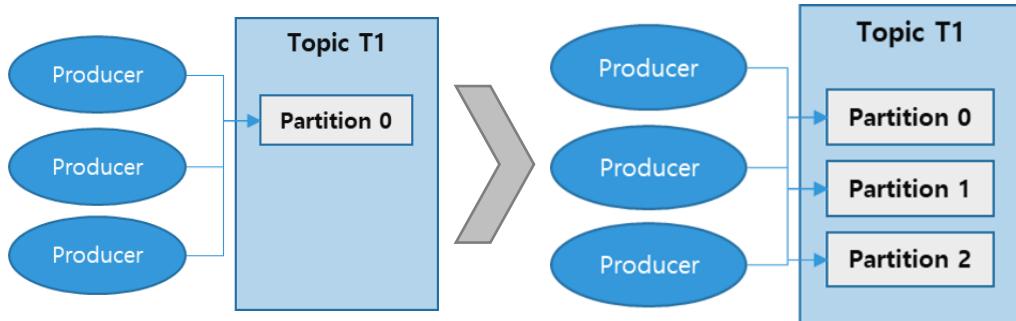
```
# Broker에 파티션 재할당  
kafka-reassign-partitions.sh \  
--bootstrap-server kafka1:9092 --zookeeper zk1:2181 \  
--reassignment-json-file reassignment.json \  
--execute
```

```
# 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



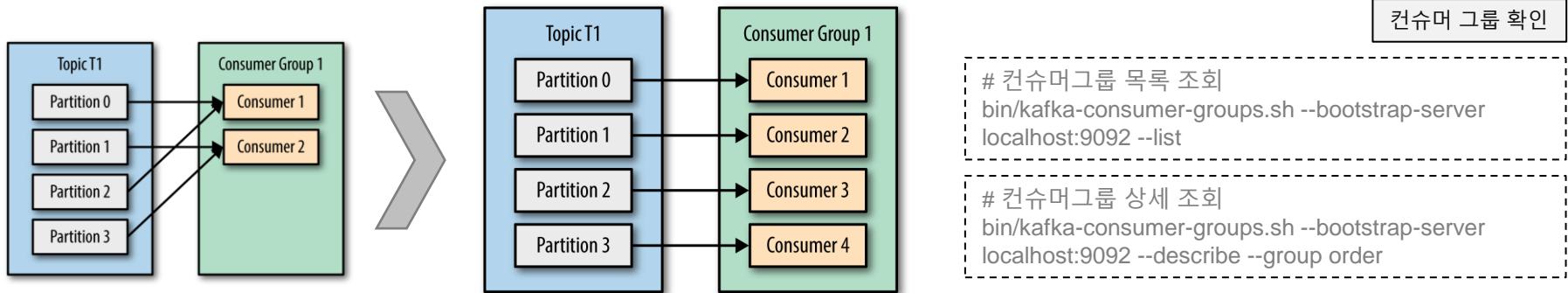
실행명령

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

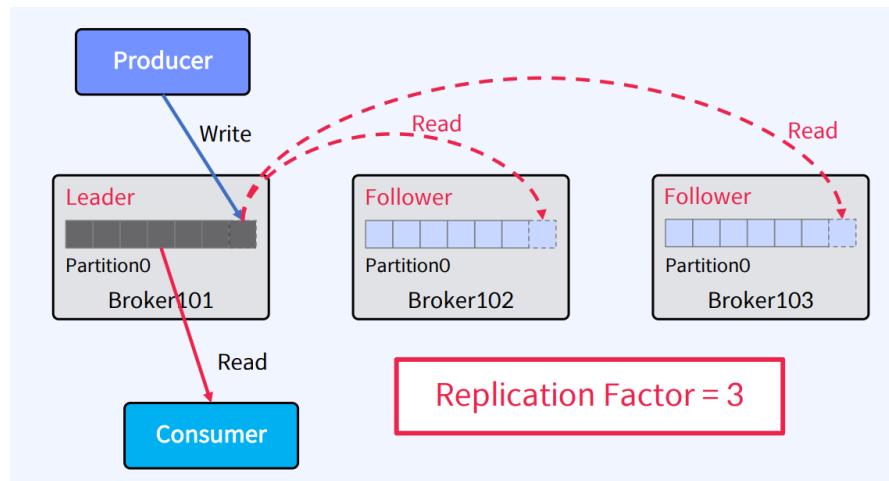
- 카프카 운영시, Message Lagging이 높을 경우, 처리를 담당하는 컨슈머를 증가한다.
 - 컨슈머 그룹별로 그룹 코디네이터(Group Coordinator)가 각 컨슈머 그룹을 관리
 - 컨슈머는 폴링(polling)하거나 커밋할 때 Heartbeat 메시지를 그룹 코디네이터에게 전달
 - 컨슈머 수가 변경되면 그룹내에서 소유권 이관작업이 일어나며 이를 리밸런싱이라 함
 - 그룹 코디네이터가 일정 기간(session.timeout.ms) 컨슈머의 하트비트를 받지 못하면, 해당 컨슈머를 장애로 판단하고 리밸런싱을 수행



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
```

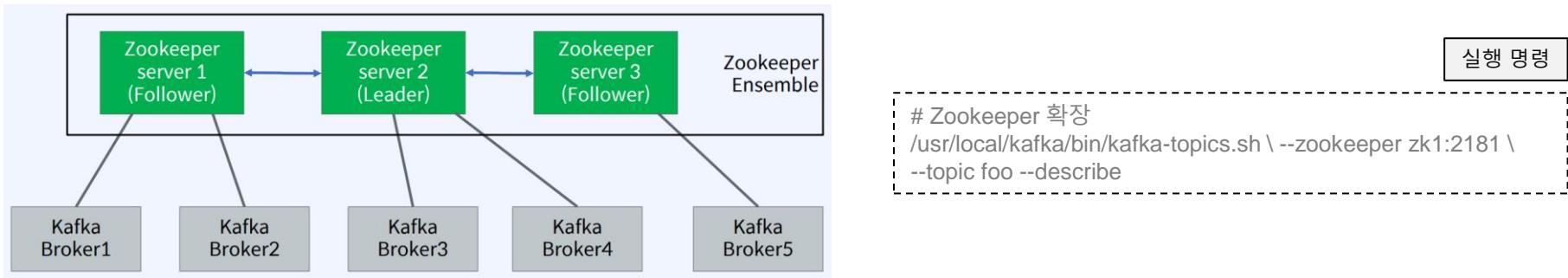
```
{ # RF 변경 JSON 생성
```

```
"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)

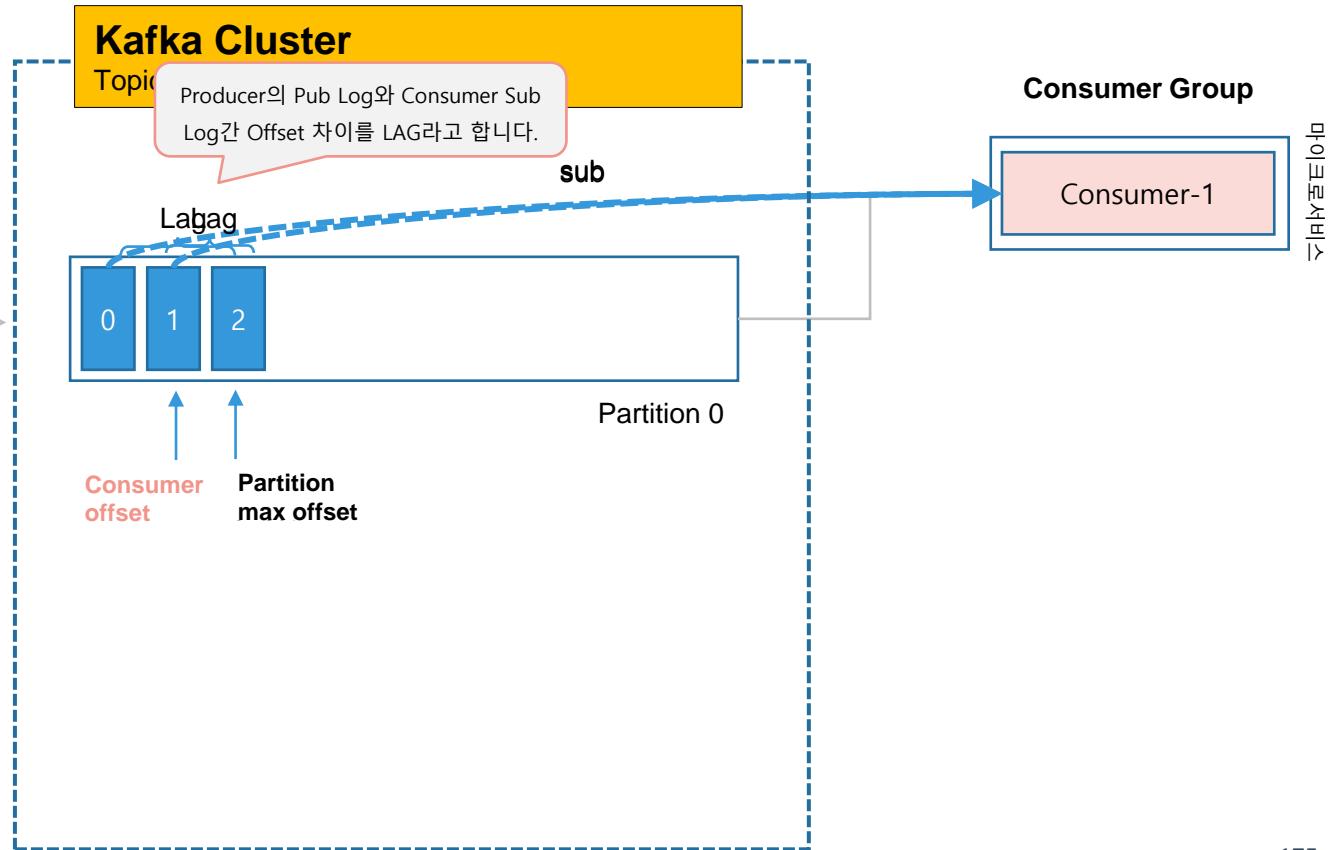


Wrap up Message Flow in Kafka

Kafka Basic

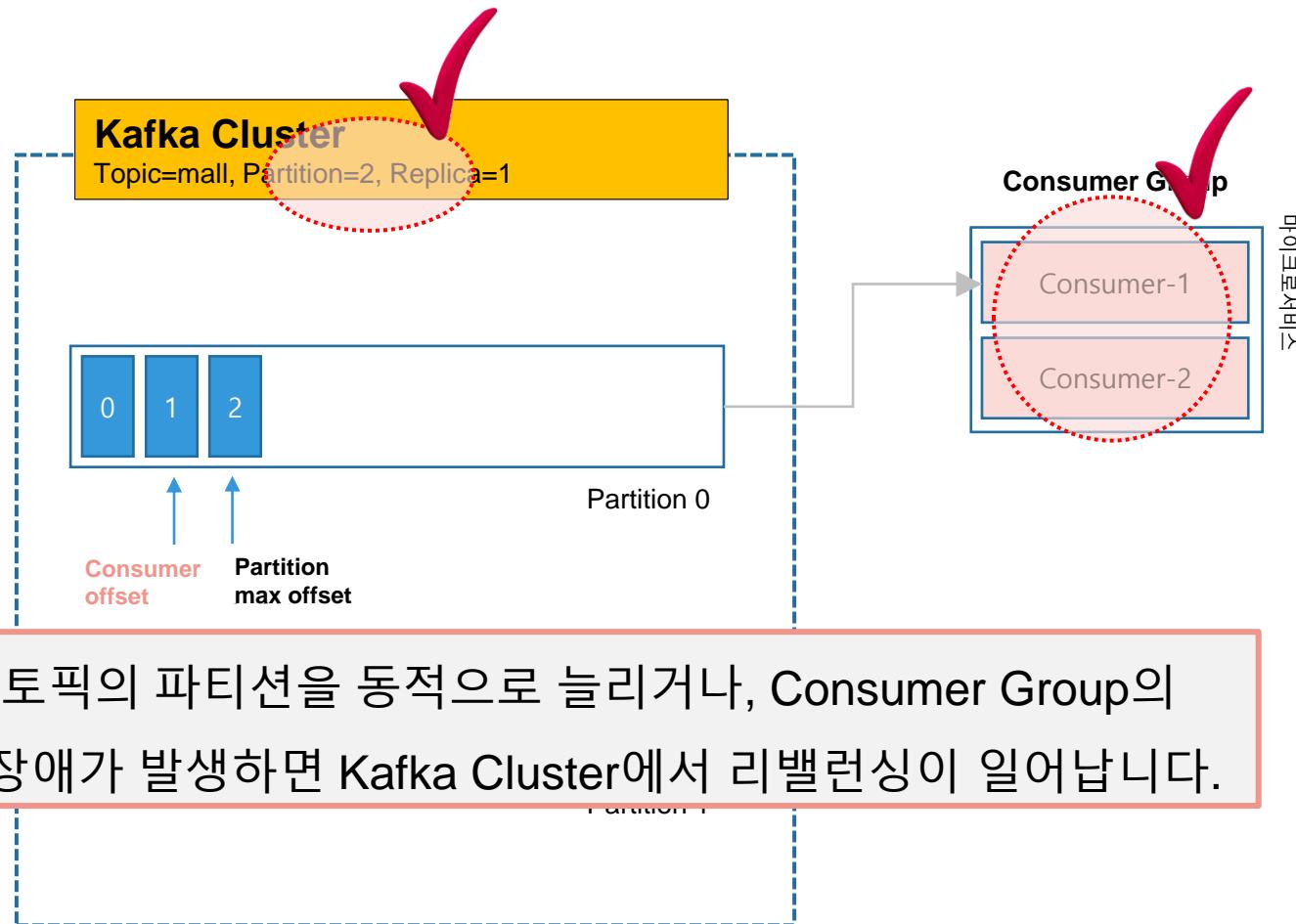
마이크로서버
서버

Producer



Rebalancing

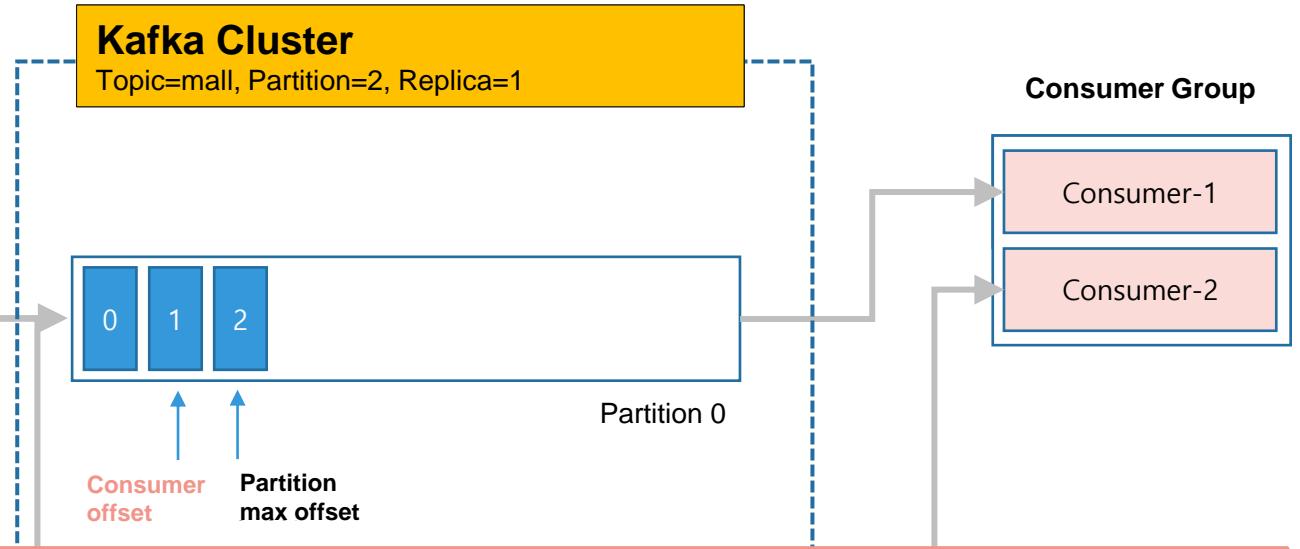
마이크로서비스
설계 원칙



Rebalancing

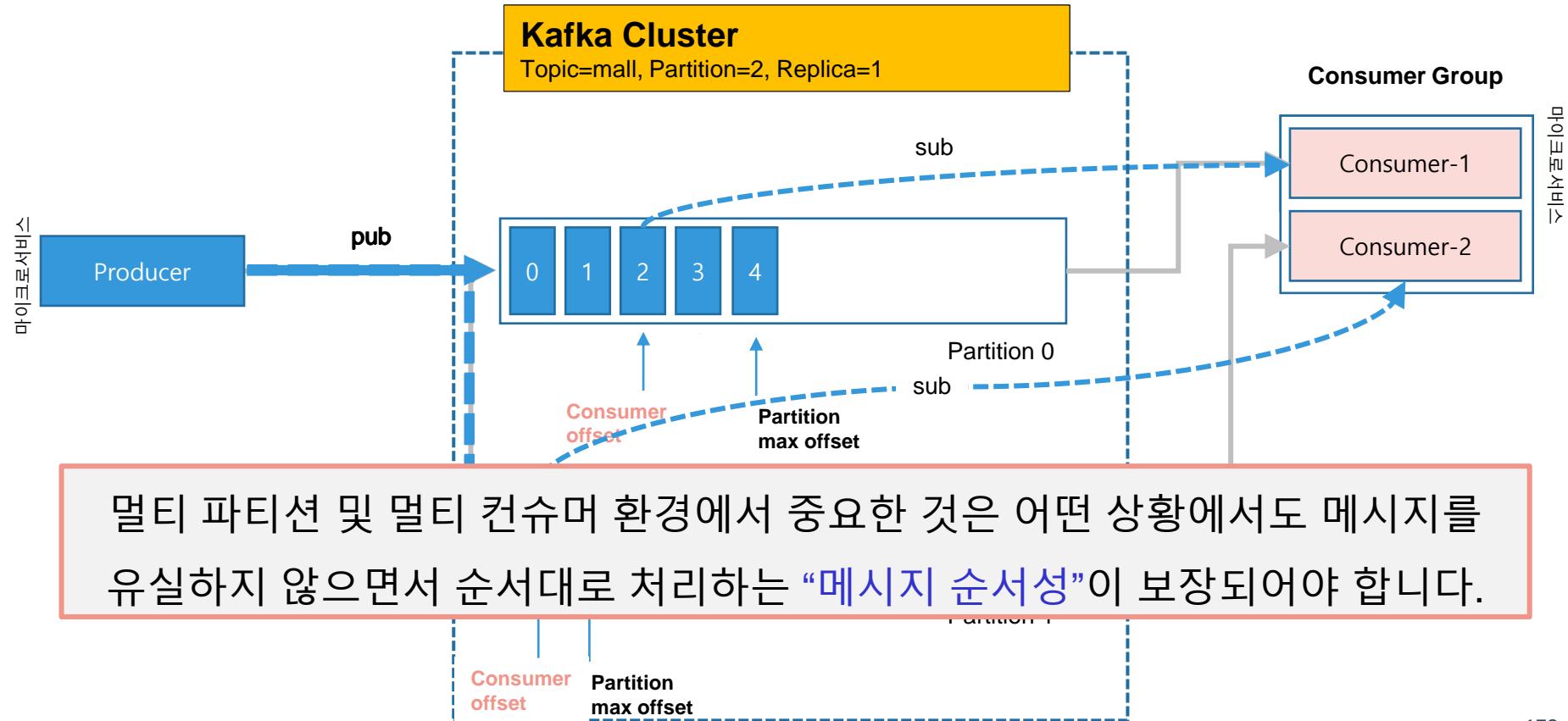
시
스
마
이
크
로
제
이
스

마이크로서비스

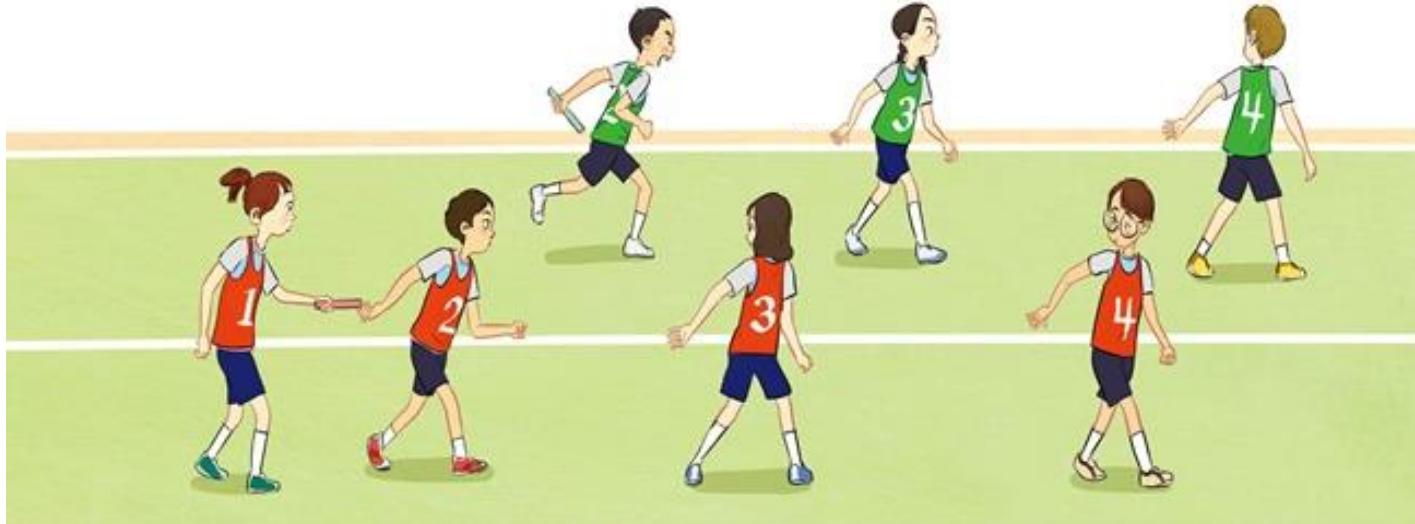


리밸런싱이란 컨슈머그룹의 파티션 소유권을 재할당하는 작업을 말하며, 이때 Producer의 메시지는 모든 파티션에 균등하게 배분되어 저장됩니다.

Multi-Partition

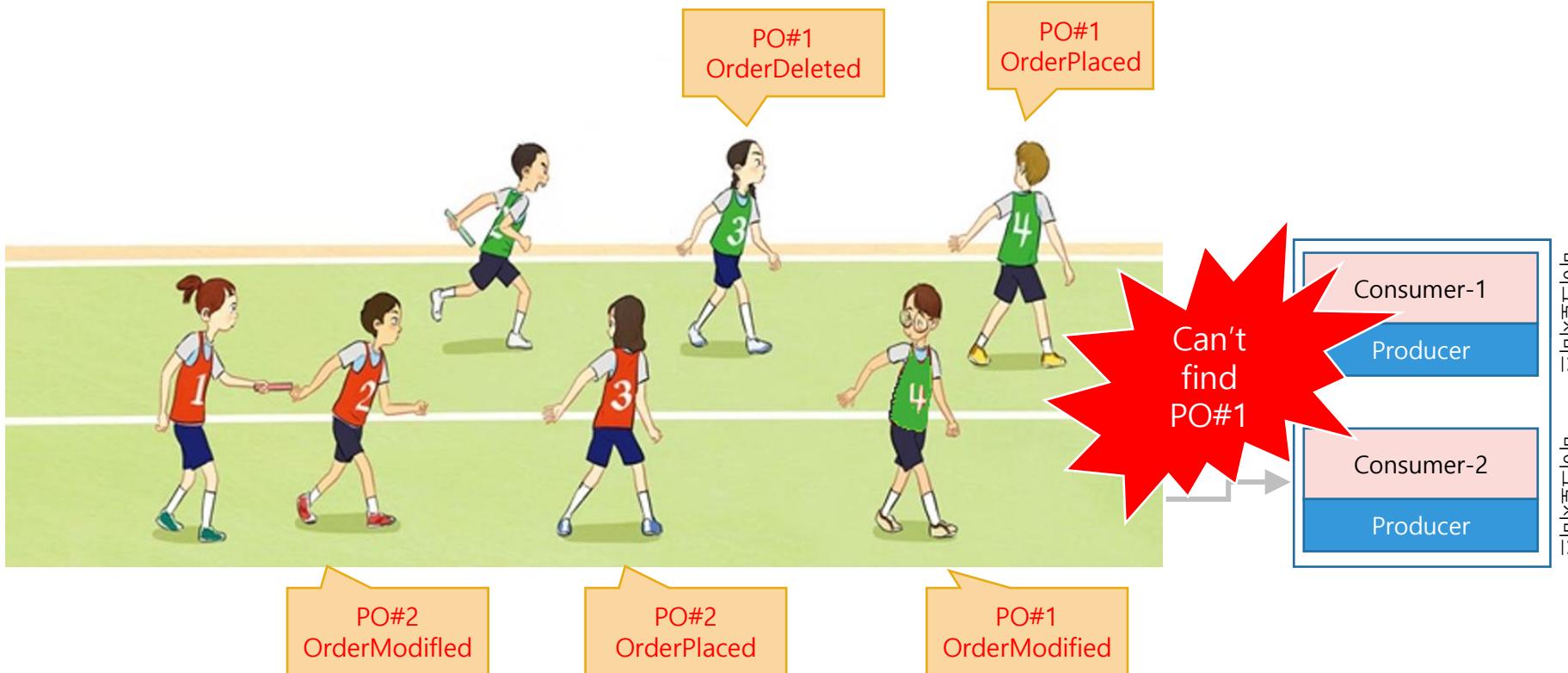


Message Orderliness

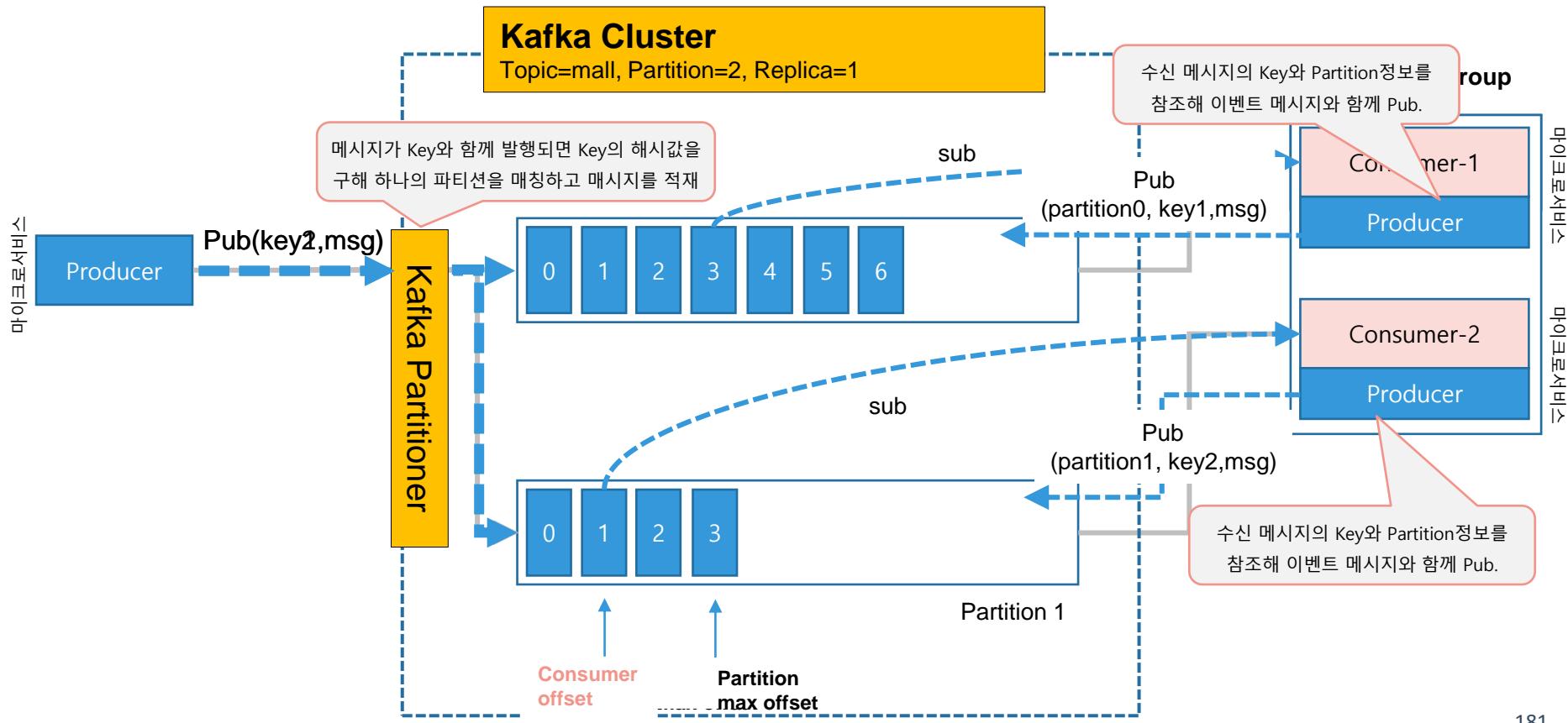


이는 마치, 400계주 이어달리기처럼 카프카 메시지들은 지정된 트랙(파티션)을
유지하며, 메시지간 순서성 보장을 위한 바톤(Key)을 주고 받아야 합니다.

If Message Orderliness broken...



Message Orderliness



“

Event Shunting in the Persistency Layer

- Change Data Capture (CDC)

CDC – Change Data Capture (1/2)

- CDC는 데이터베이스 내 데이터에 대한 변경을 식별해 필요한 후속처리(데이터 전송/공유)를 자동화하는 기술 또는 설계 기법이자 구조
- 필요성
 - 계정계에서 정보계로의 실시간 데이터 동기화 필요성 증대 (금융 도메인)
 - MSA 전환 또는 신규 구축에 따른 원장 데이터의 읽기 전용 모델 구축
 - Open API 확대에 따른 Core 시스템 부하 감소
- CDC Solutions
 - Oracle GoldenGate (OGG)
 - IBM InfoSphere
 - Kafka Connect, Eventuate Tram
 - 인포매티카 cdc, HVR, Strim, etc



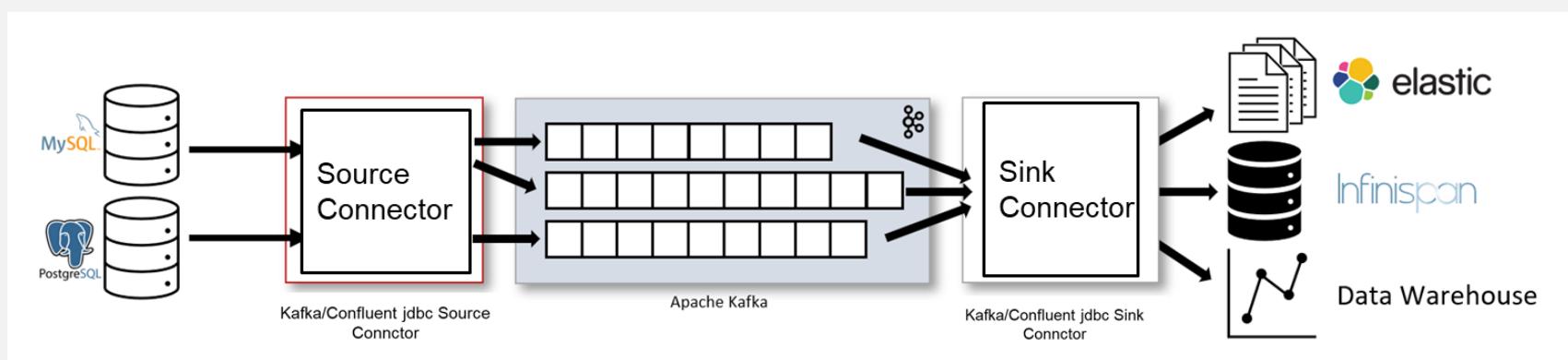
Image source : https://www.megazone.com/armiq_series3/

CDC – Change Data Capture (2/2)

- CDC는 데이터베이스 내 데이터에 대한 변경을 식별해 필요한 후속처리(데이터 전송/공유)를 자동화하는 기술 또는 설계 기법이자 구조
- CDC 구현기법
 - Time Stamp on Rows
 - Version Numbers on Rows
 - Status on Rows
 - Time Version/Status on Rows
 - Triggers on Tables
 - Event Programming
 - Log Scanner on Database
- CDC 구현방식
 - PUSH : 데이터 원천에서 변경을 식별하고 대상 시스템에 변경 데이터를 적재해 주는 방식
 - PULL : 대상 시스템에서 데이터 원천을 정기적으로 Pulling하여 데이터를 동기화 하는 방식

CDC with Apache Kafka

- Kafka Connect는 Producer와 Consumer를 사용해 데이터 파이프라인을 생성
- Source Connector를 사용해 Kafka Broker로 데이터를 보내고, Sink Connector를 사용해 Kafka에 담긴 데이터를 타겟 DB에 저장



→ Connector Type에는 대상 테이블을 직접 조회하는 Query-based CDC Connector와 데이터베이스 로그 파일을 모니터링하는 Log-basedSource Connector로 구분

CDC with Apache Kafka

- Kafka Connect 서버에 REST api를 통해 Sink/Source Connector를 등록한다.
- Query-based 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

CDC with Apache Kafka

- Query based 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

CDC with Apache Kafka

- Log-based CDC는 각 DB 벤더사의 connector를 이용하거나 debezium의 Open Source인 debezium Connector를 사용한다.
- Log-based Source Connector 예제

```
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": "{KAFKA 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

CDC with Apache Kafka

- Log-based 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

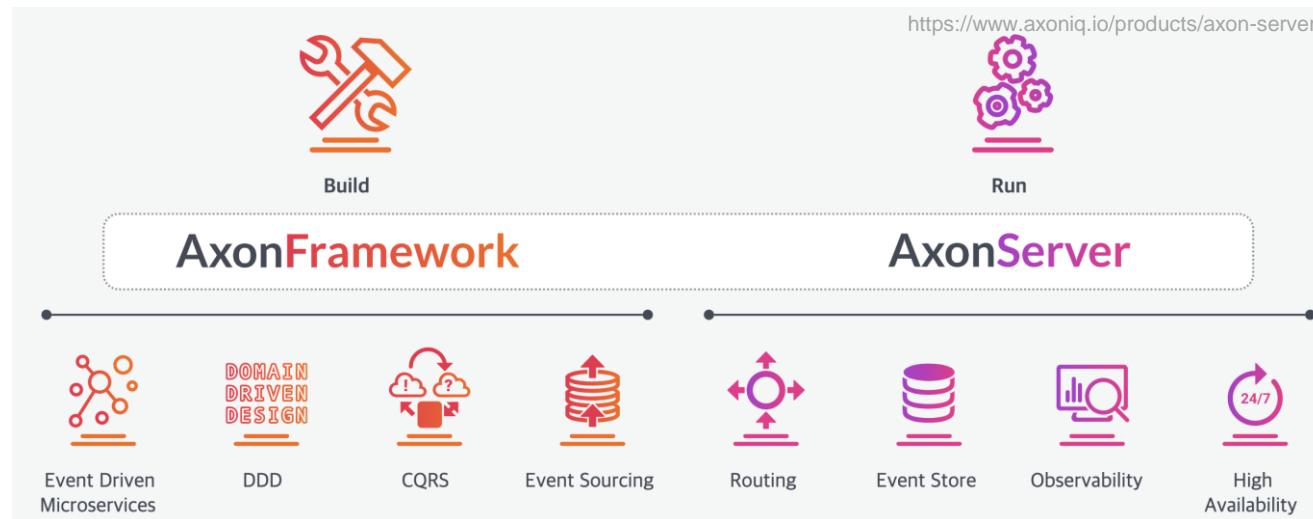
“

Event driven System

- Dedicated Server : Axon

Event driven System - Axon

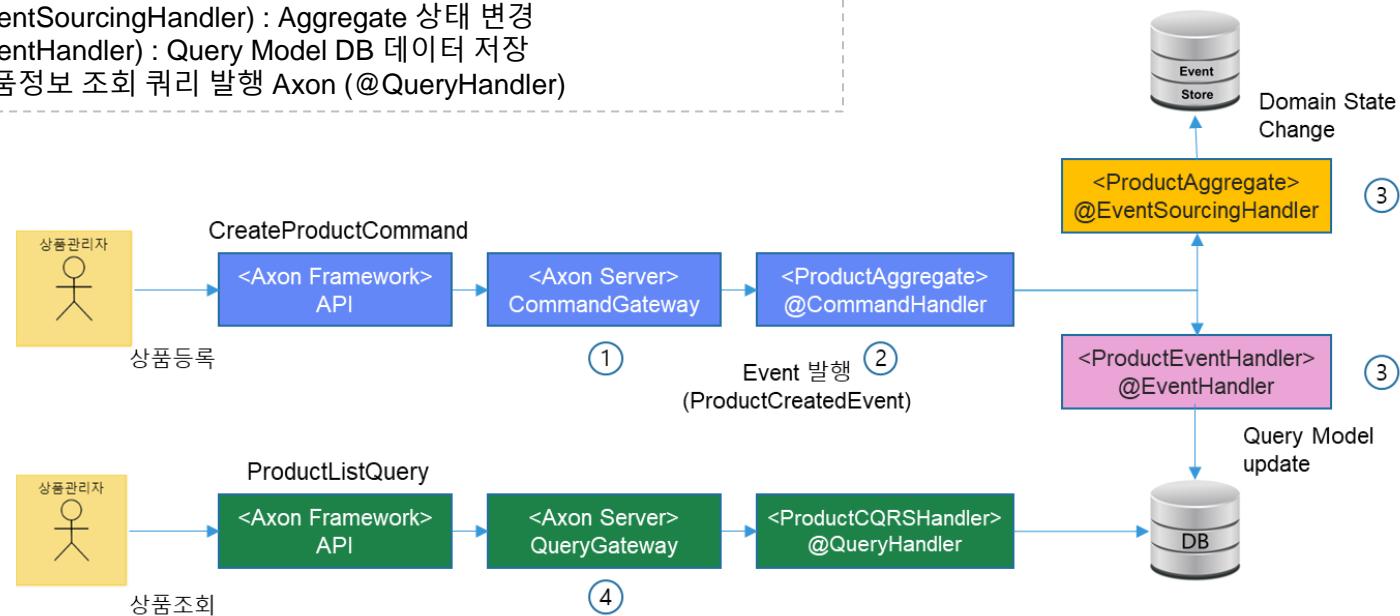
- Axon Framework / Axon Server
- Axon Framework : EventSourcing, CQRS, DDD기반 어플리케이션 개발을 지원하는 Framework
- Axon Server : 메시지 라우팅, 이벤트 저장소 역할 (Kafka, NoSQL DB로 대체가능)



Event driven System - Axon

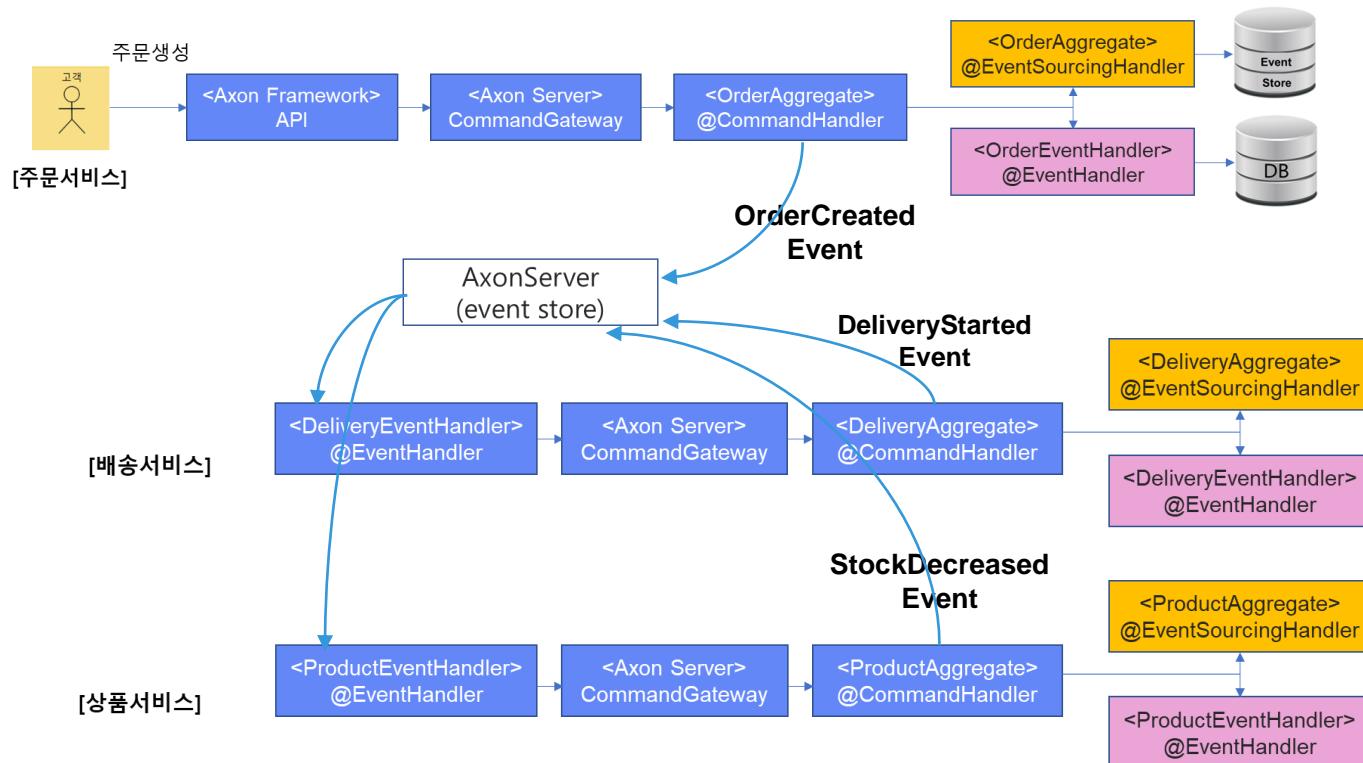
- Axon 동작 방식 – ‘상품등록 및 조회’ in 12st mall

- ✓ 관리자 : 상품등록 커맨드 실행
- ✓ Axon (@CommandHandler) : 수신된 커맨드로 로직을 수행하고 이벤트 발생
- ✓ Axon (@EventSourcingHandler) : Aggregate 상태 변경
- ✓ Axon (@EventHandler) : Query Model DB 데이터 저장
- ✓ 관리자 : 상품정보 조회 쿼리 실행 Axon (@QueryHandler)



Event driven System - Axon

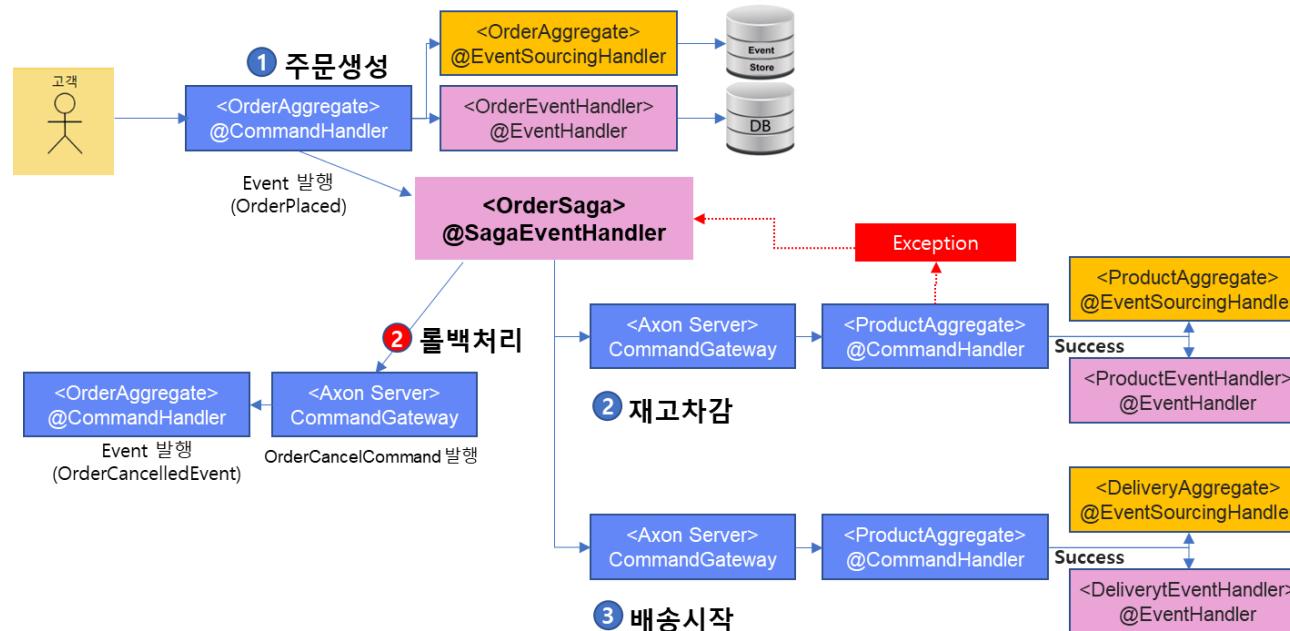
- Choreography Saga – ‘도메인 이벤트에 스스로 반응하는 프로세스’



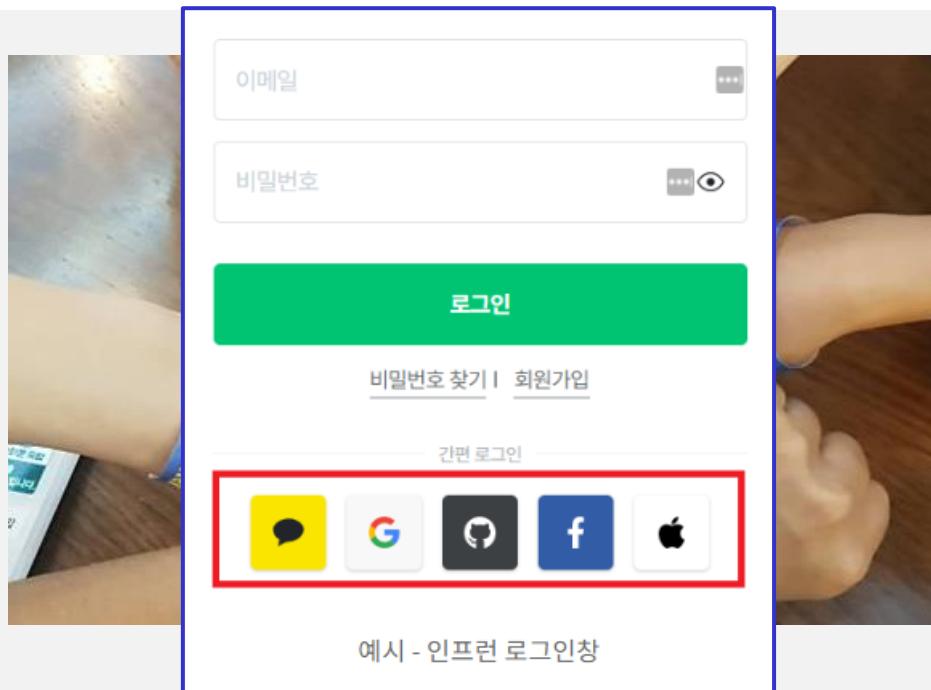
Event driven System - Axon

- Orchestration Saga – ‘오케스트레이터에 의한 프로세스 컨트롤’

✓ 시나리오 : 주문이 일어나면, 주문에 따른 Saga Orchestrator가 생성되고, Orchestrator에 의해 상품 재고차감, 배송시작
프로세스가 순차적으로 호출, 프로세스 실행 중 오류가 발생시, 각 프로세스별 보상처리(Compensation)로직 실행

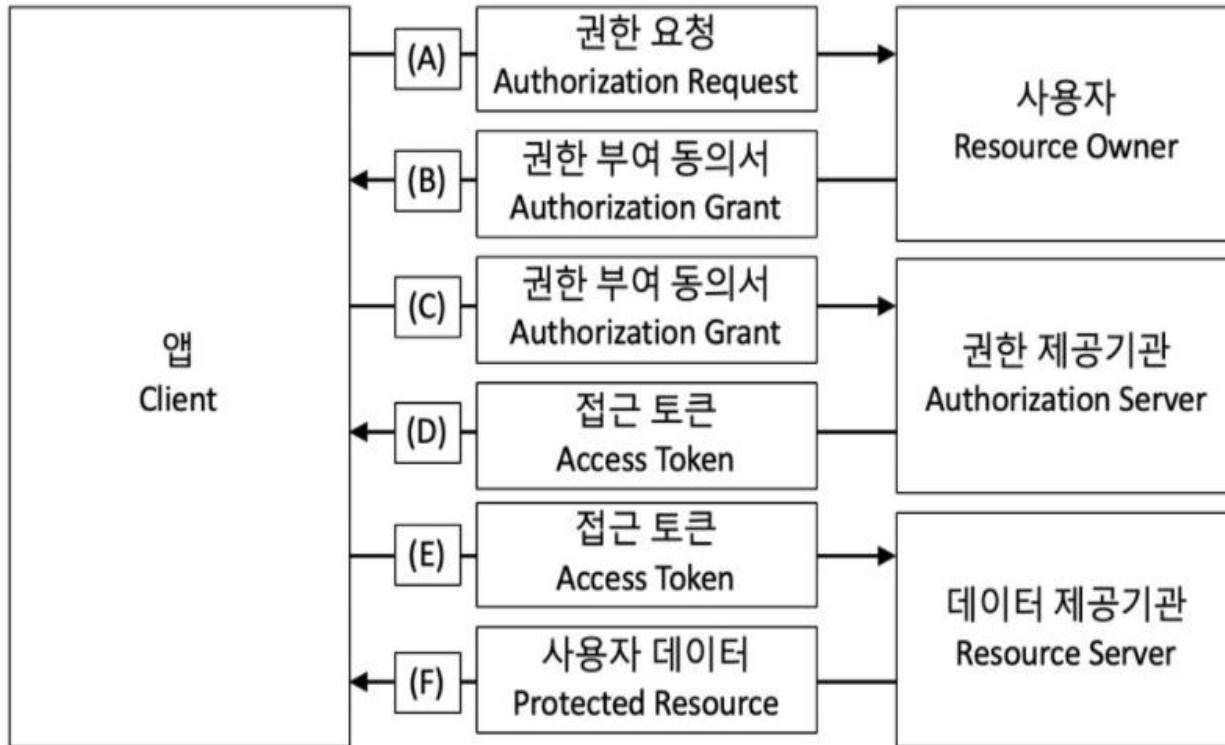


(Issues #3) 서비스 분리에 따른 통합인증은 어떻게 할 것인가?



- **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

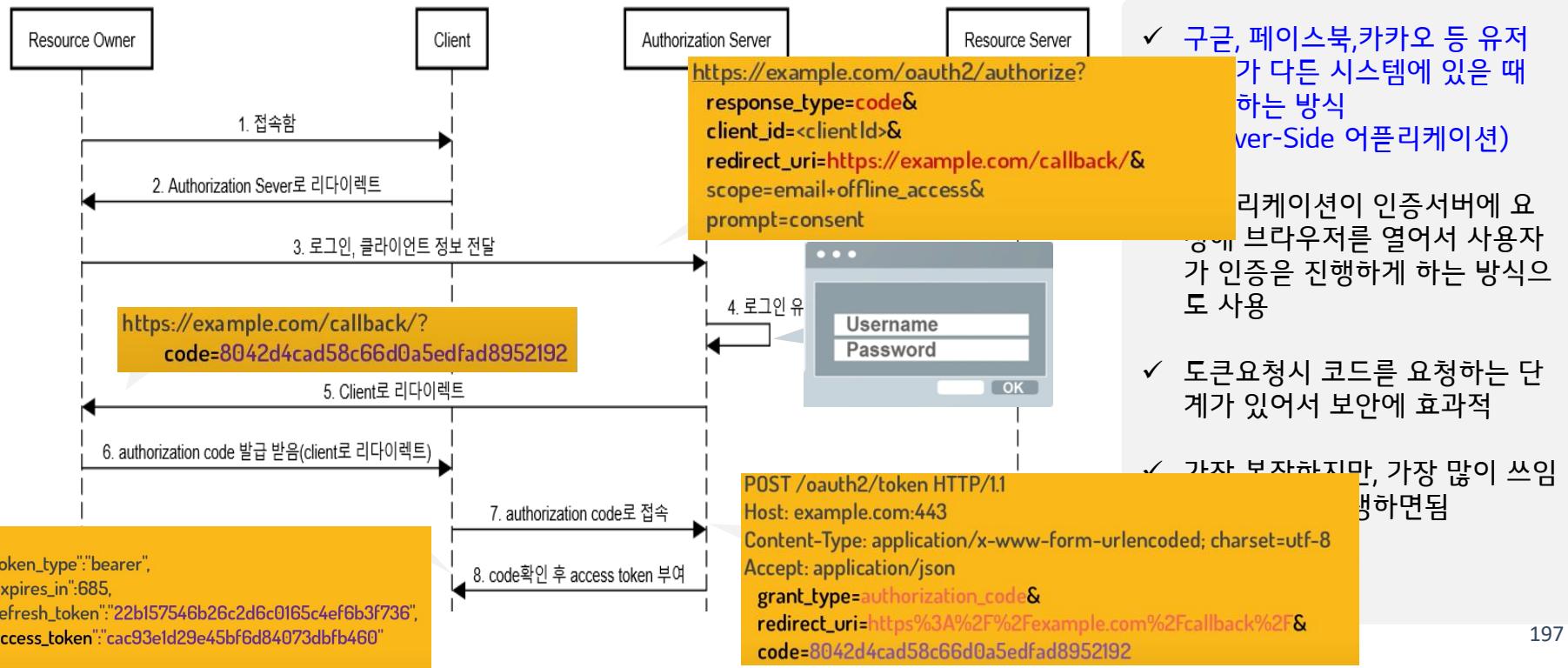
OAuth 2.0: Authorization Flow



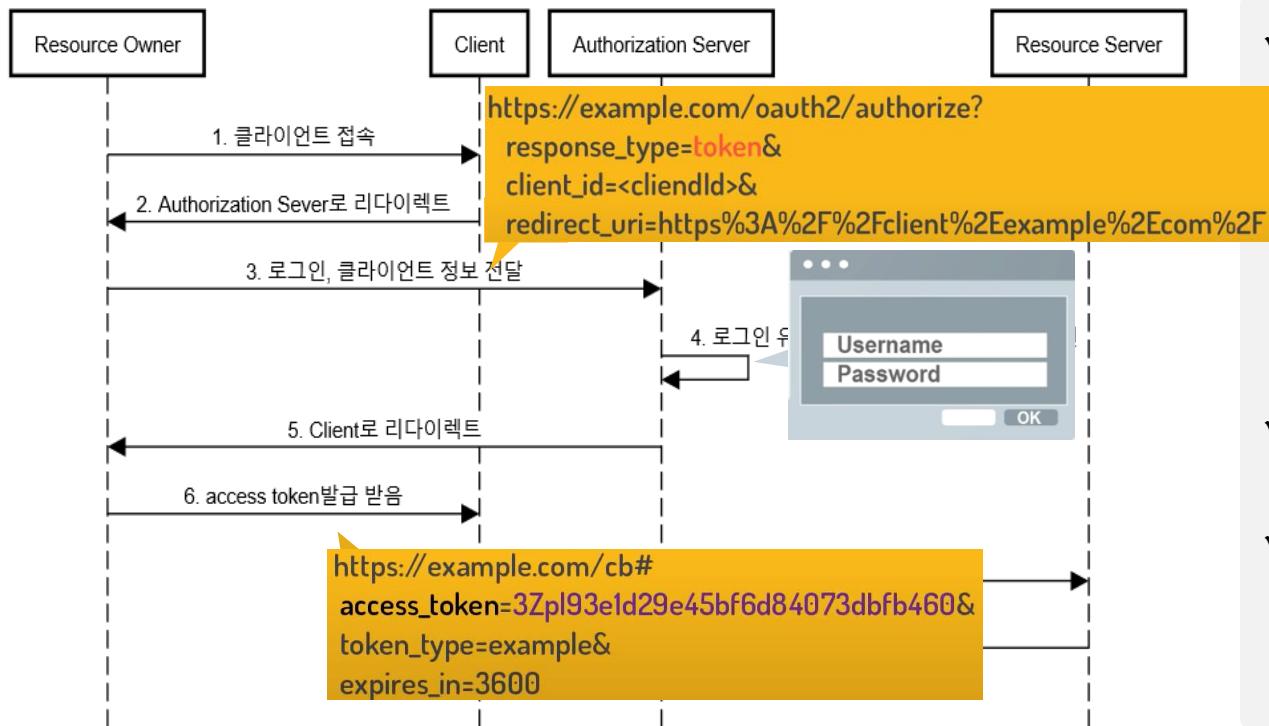
출처: <http://www.2e.co.kr/>

- A. 클라이언트가 자원 소유자에게 권한 요청
- B. 자원 소유자가 권한을 허가시, 클라이언트는 권한 증서를 받급받음
- C. 클라이언트는 권한증서를 가지고 도큰은 권한 서버에 요청
- D. 권한증서의 유효성을 체크하고 도큰은 받급해줌
- E. 클라이언트는 도큰은 사용하여 자원 요청
- F. 도큰 유효성 확인후 요청 처리

Grant Type #1 - Authorization Code

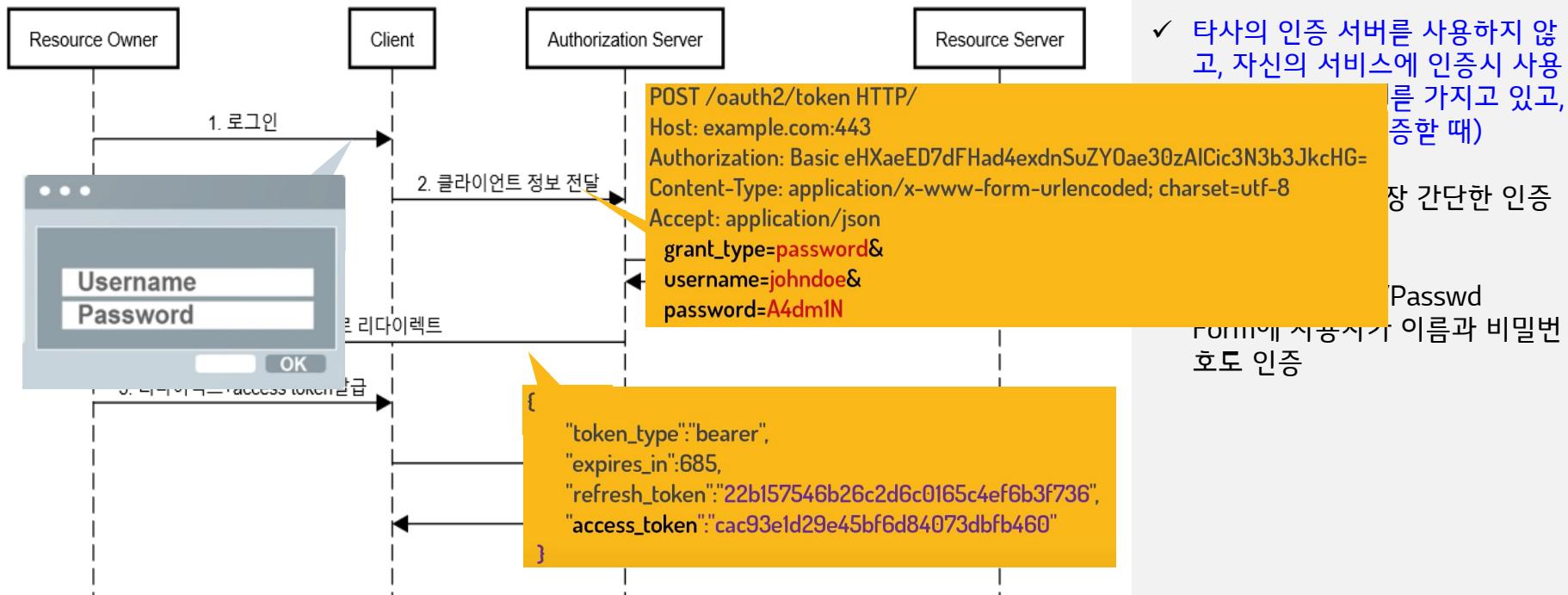


Grant Type #2 - Implicit (암묵적 유형)

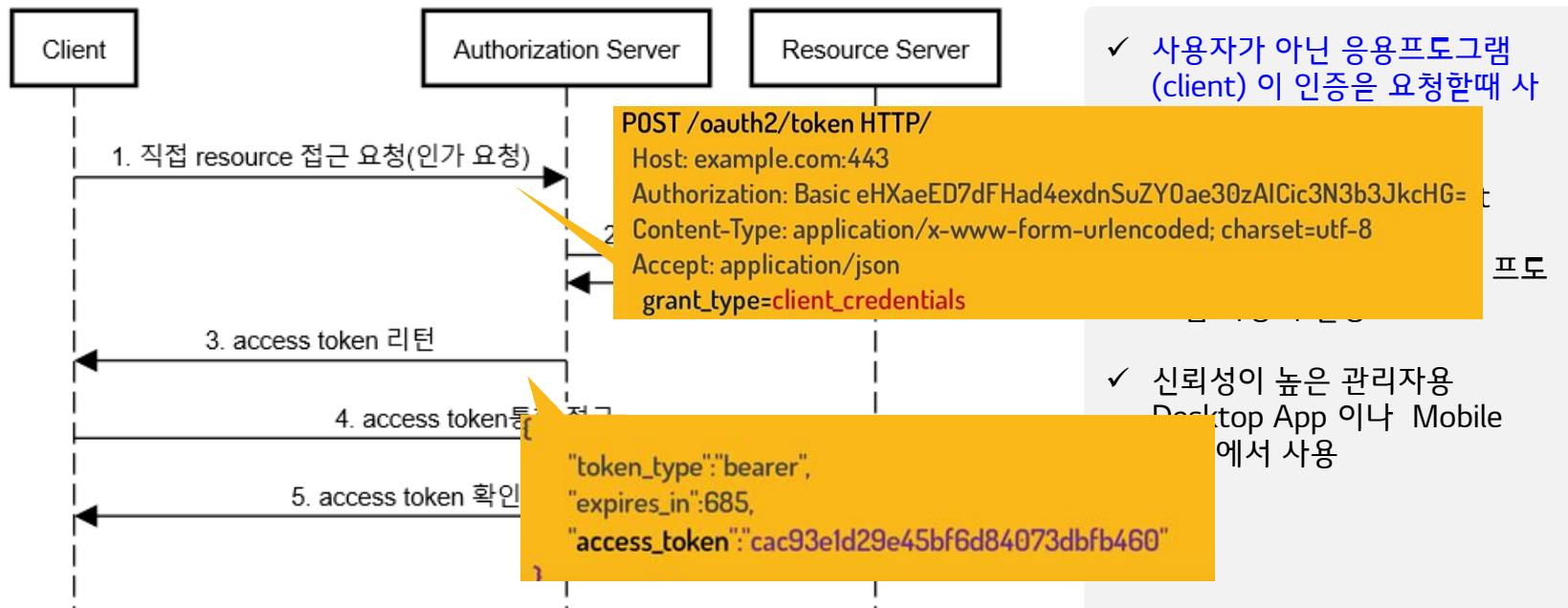


- ✓ Authorization_code 방식에서 코드를 요청하는 프로세스를 제거하고, 바로 도くん을 리턴 (Mobile Apps)
- ✓ Javascript 도 동작하는 SPA(Single Page Application, react, Vue)에서 사용하기 위해 만들어 졌으나 권장하지 않음
- ✓ 신뢰성 있는 앱 또는 단말기에서 사용
- ✓ 외부에 있는 Oauth 서버가 CORS 를 지원하지 않은때 사용

Grant Type #3 - Resource Owner Password Credentials



Grant Type #4 - Client Credentials



Grant type 적용 기준

- Access Token Owner, Client Type(Confidential, Public)에 따라..
- Client가 Credential 정보(Secret, key)를 유지할 수 있느냐?

Client Credentials	<ul style="list-style-type: none">• 리소스 Owner가 없는 타입으로 B2B 트랜잭션이나 배치 프로세스에 적합 (Back channel only)
Resource Owner Credentials	<ul style="list-style-type: none">• 리소스 Owner의 크리덴셜을 관리하는 Client를 신뢰할 경우에 적합• 리소스가 Owner가 Username, Password를 Client에 제출, Client가 Authz 서버에 Owner 크리덴셜과 함께 토큰을 요청하는 방식 (Front channel + Back channel)
Authorization Code	<ul style="list-style-type: none">• Redirect 기반 Grant 유형으로 웹 브라우저와 같은 User Agent가 필요• Client가 리소스 Owner의 크리덴셜을 Authorization 서버에게 위임하고 동의가 필요할 경우 (Front channel + Back channel)
Implicit	<ul style="list-style-type: none">• Authorization Code 타입에서 CODE가 없는 타입으로 Public Client에 적합• Client가 Mobile, 또는 Single Page Application 일 경우 (Front channel only)

JWT(Json Web Token) 토큰



- 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) 토큰

Strength

- URL 파라미터와 헤더로 사용
- 수평 스케일이 용이
- 디버깅 및 관리가 용이
- 트래픽 대한 부담이 낮음
- REST 서비스로 제공 가능
- 만료 시간 내장
- 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>

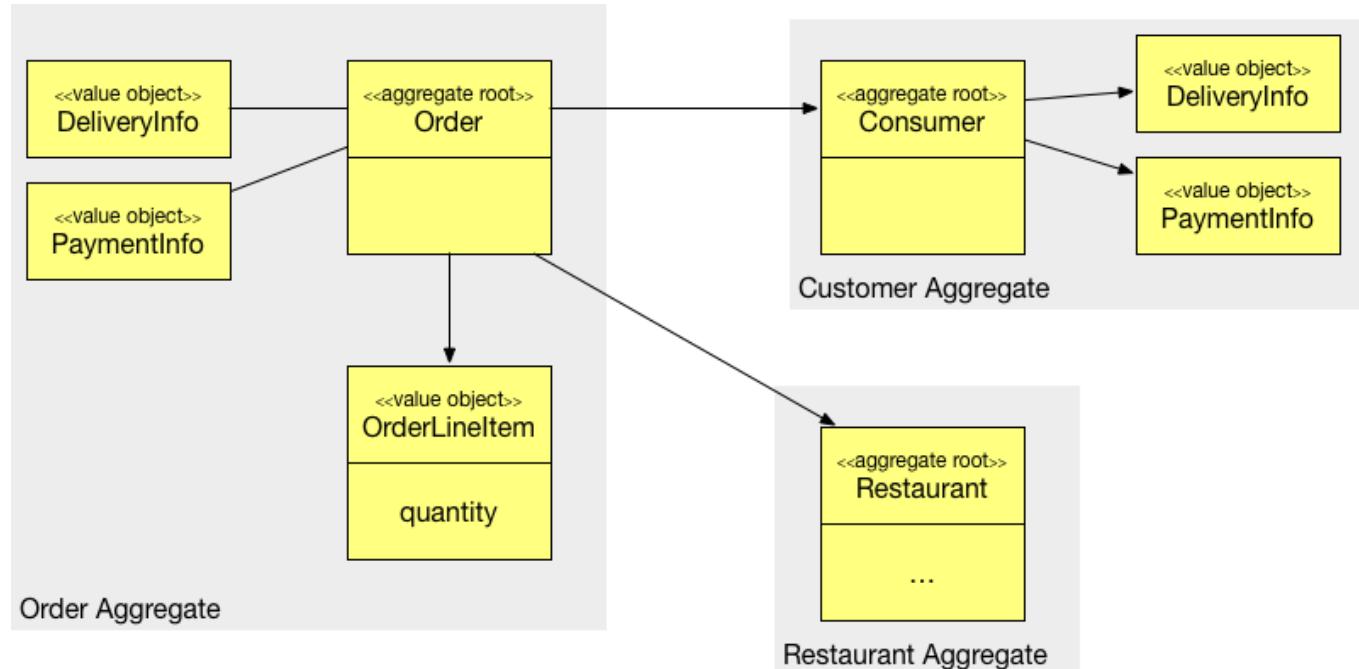
Lab: OAuth Based Authorization (3/3)

- 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

(Issues #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

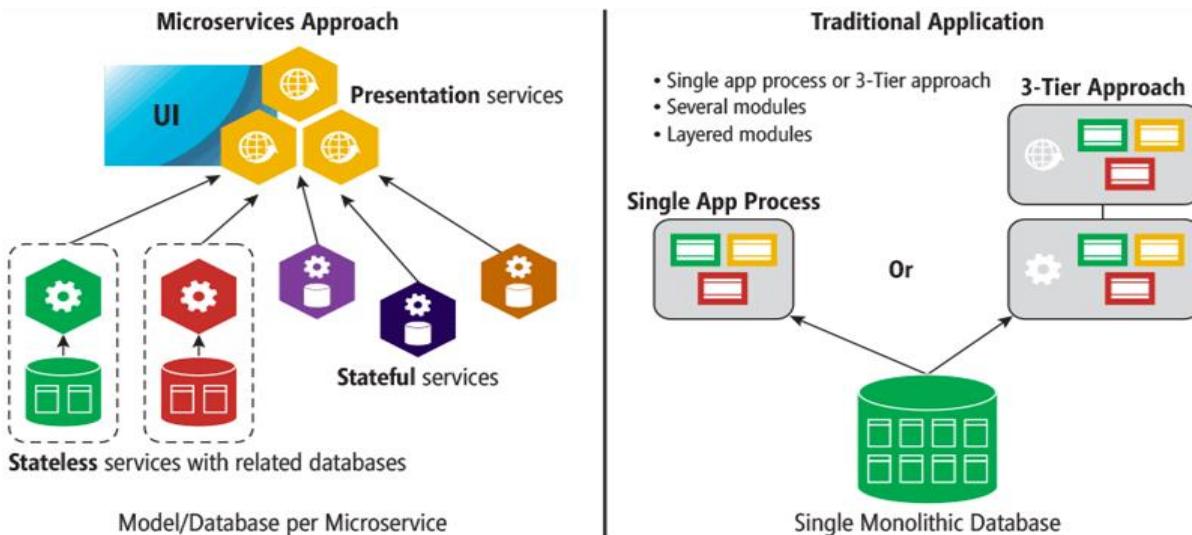
- **HATEOAS** (Hypertext As The Engine Of Application State)

- 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

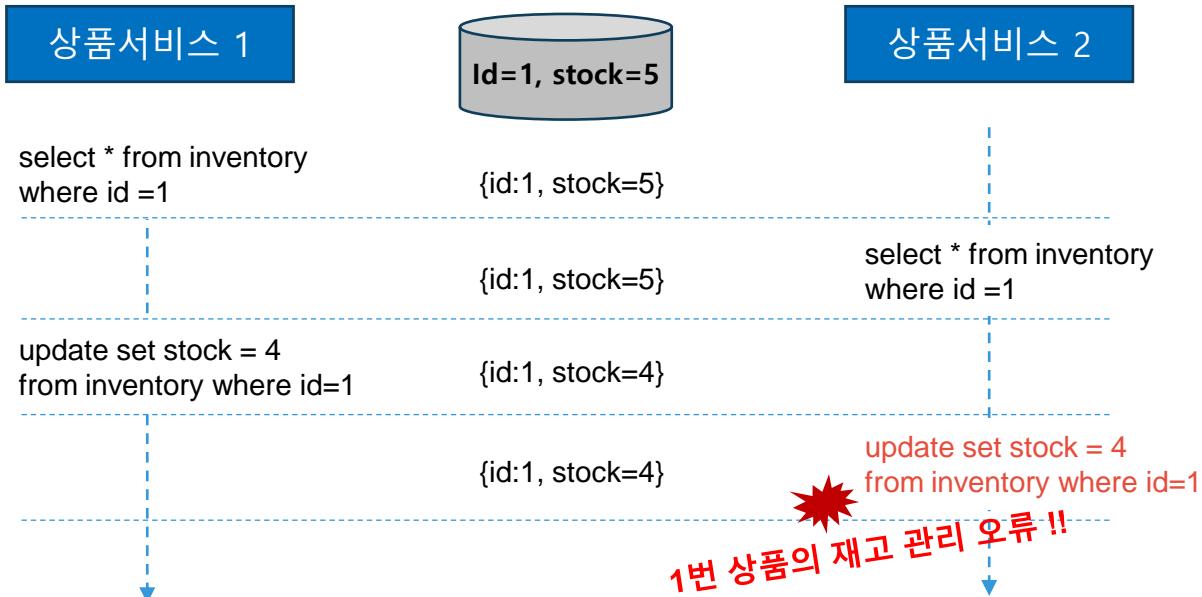
- 서비스의 통합을 위하여 기존에 Join SQL 등을 사용하지 않고 프론트-엔드 기술이나 API Gateway 를 통하여 서비스간 데이터를 통합함
- 프론트엔드에서 데이터를 통합하기 위한 접근 방법으로는 W3C 의 Web Components 기법과 MVVM 그리고 REST API 전용 스크립트가 유용함



(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

- 상품 마이크로서비스에서의 재고수량 동시성 이슈

- Data Consistency를 위한 트랜잭션 관리가 이루어 지지 않을 경우,



Race Condition !!
(Data race)

Java Synchronized 활용 (X)

데이터베이스 락 이용 (O)

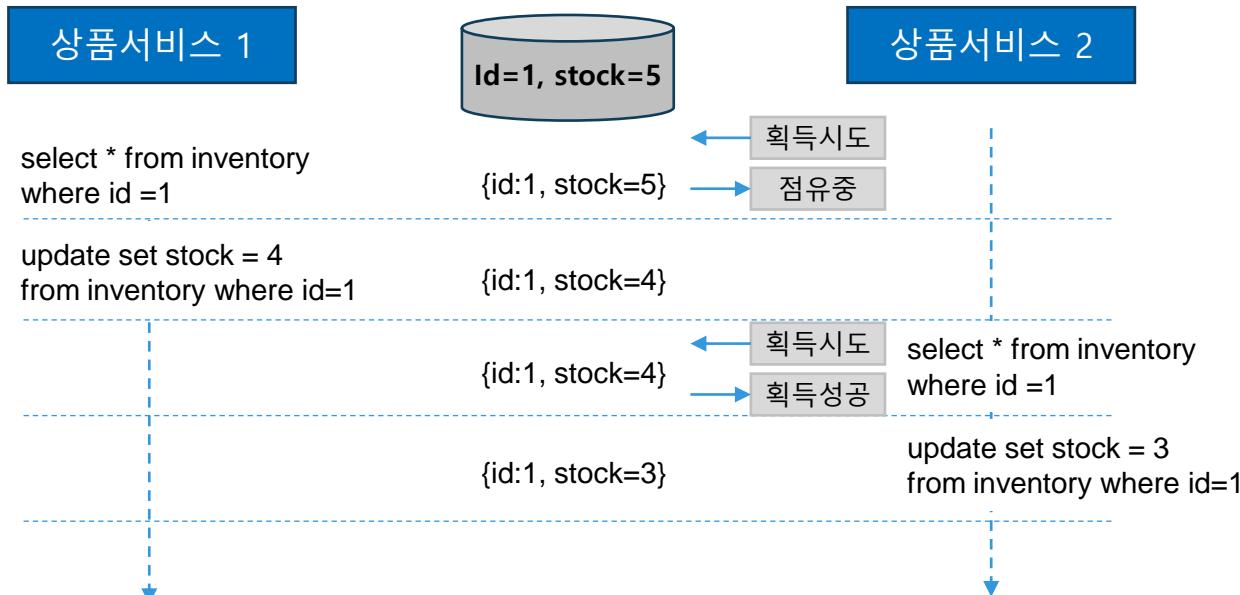
애플리케이션 락 처리 (O)

Global Distributed Lock
활용 (O)

Saga Pattern 활용 (O)

(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

- 데이터베이스 락을 이용한 상품서비스의 재고수량 동시성 이슈 해결
 - 동시성 문제가 발생할 것이라고 예상하고 락을 걸어버리는 **비관적 락(Pessimistic Lock)** 방식



실제로 데이터에 Lock을 걸어
정합성을 맞추는 방식

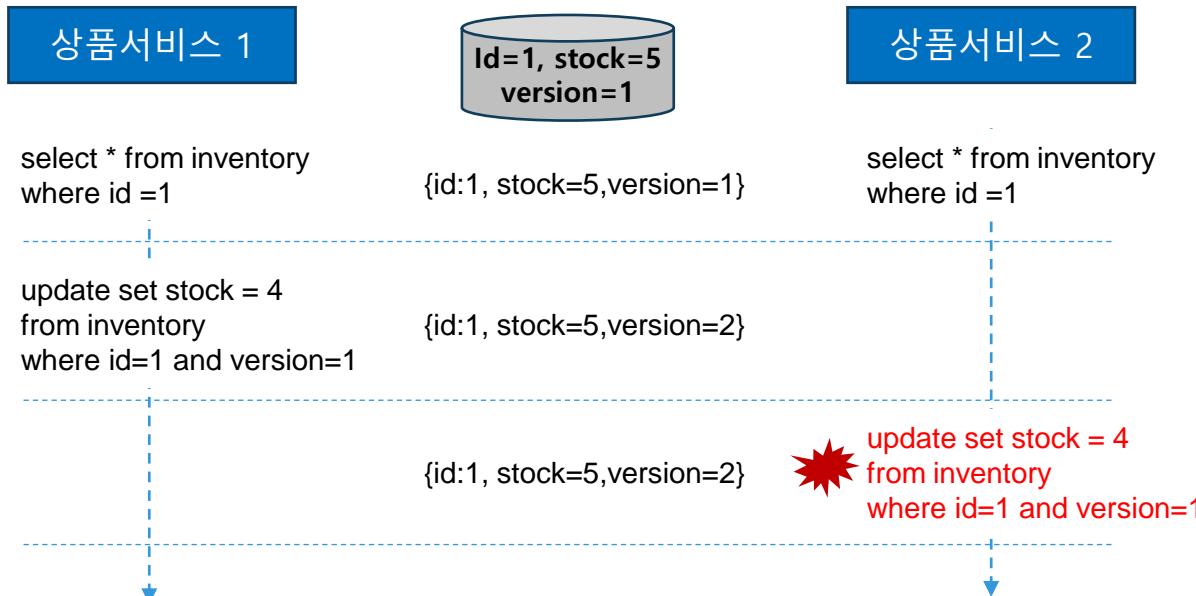
Exclusive Lock(베타적 잠금)
동시성 성능 저하

Lock 해제까지 다른
트랜잭션은 대기

Data의 정합성이 중요한
API에 적용

(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

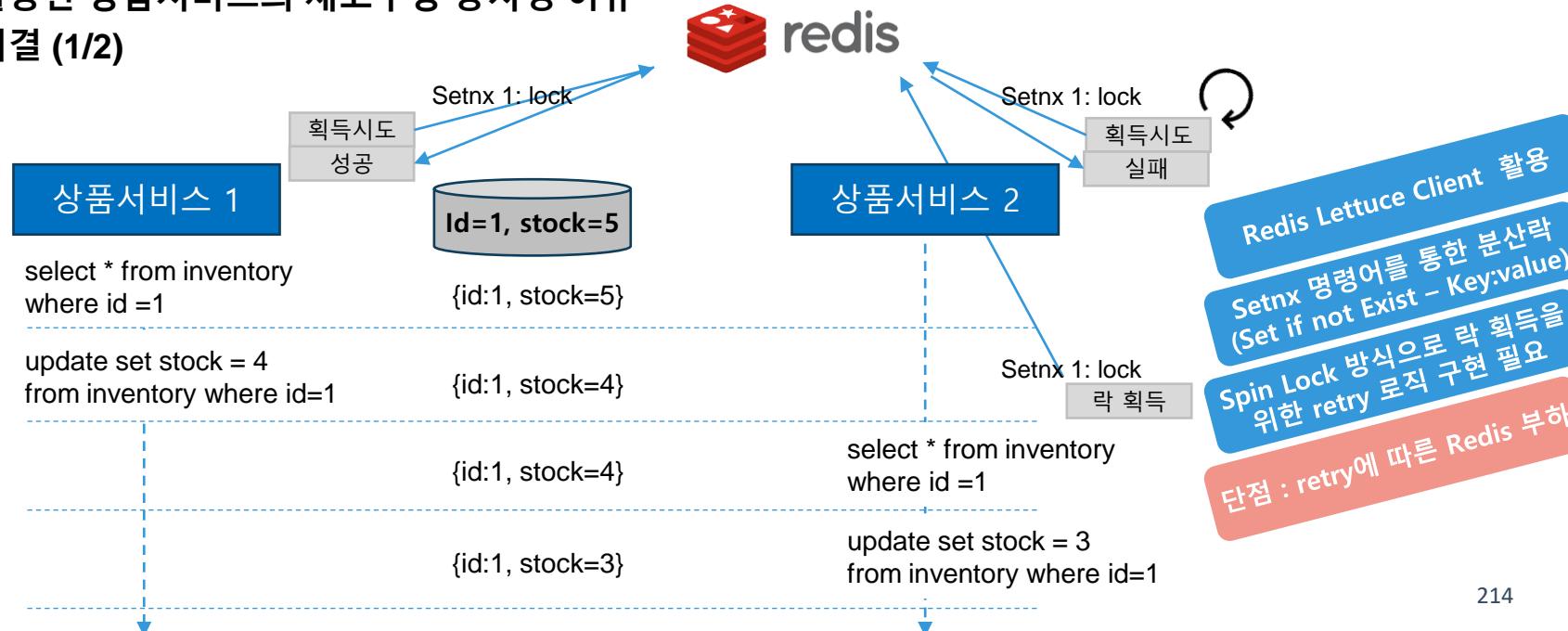
- 애플리케이션에서 락을 처리함으로써 상품서비스의 재고수량 동시성 이슈 해결
 - 동시성 문제가 발생하면 그때가서 처리하는 낙관적 락(Optimistic Lock) 방식



실제로 Lock을 이용하지 않고 버전을 이용해 정합성 맞춤
Update 수행시, 내가 읽은 버전인지 확인 후 업데이트
Pessimistic Lock보다는 성능상의 이점
Update 실패시, 개발자가 직접 재시도 또는 를백로직 작성

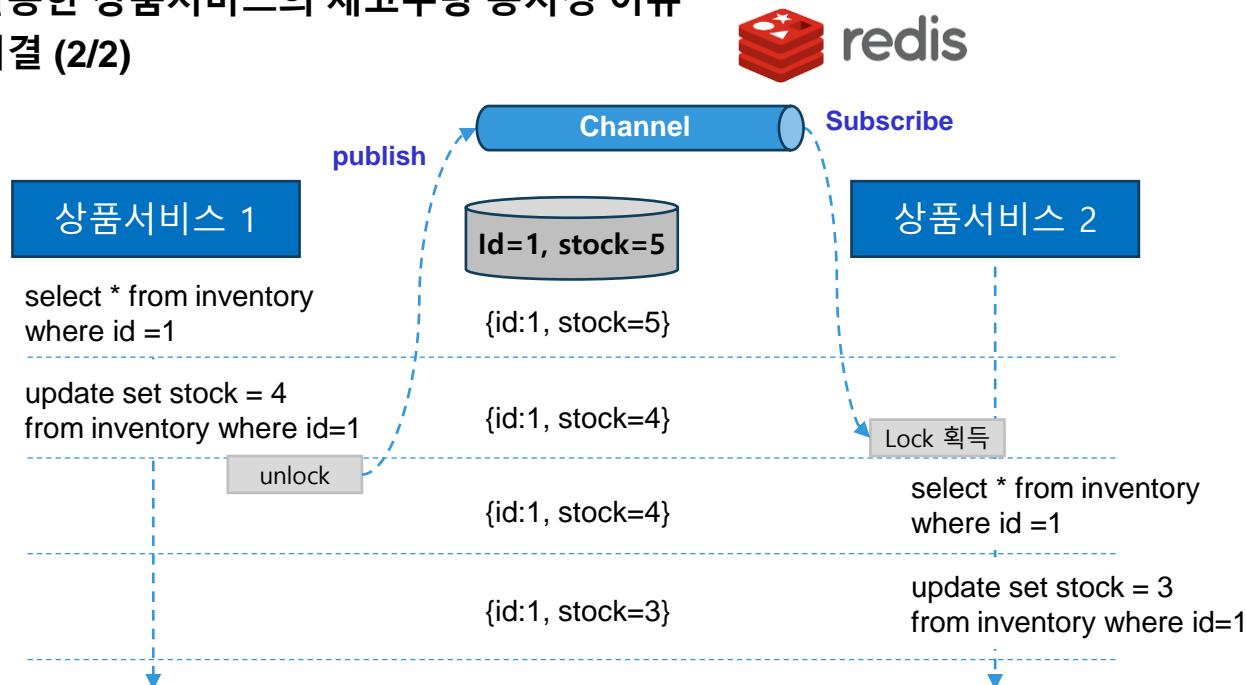
(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

- 글로벌 분산 락(Distributed Lock w/ Redis)를 활용한 상품서비스의 재고수량 동시성 이슈 해결 (1/2)



(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

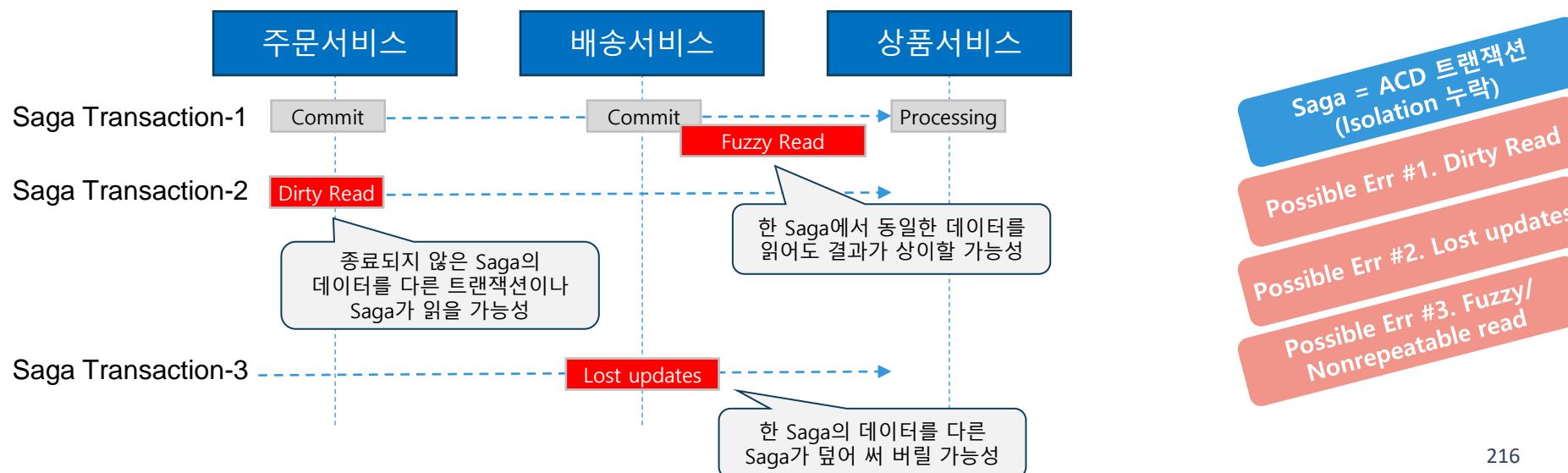
- 글로벌 분산 락(Distributed Lock w/ Redis)를 활용한 상품서비스의 재고수량 동시성 이슈 해결 (2/2)



Redis redisson Client 활용
Key를 중심으로 Pub/Sub
기반 Locking 모델 제공
타임아웃을 설정해 대드락방지
장점 : Spin Lock을 사용하지
않아 redis 부하 없음

(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

- 분산 트랜잭션 없이 Saga를 적용했을 때, 상품서비스의 재고수량 동시성 이슈
 - Saga에 참여하는 로컬 트랜잭션이 커밋과 동시에 외부에 노출되기 때문에 “**격리성 저하**” 발생



(Issues #5) 다중 서버 환경에서 데이터 정합성을 어떻게 유지할 것인가?

- 분산 트랜잭션 없이 Saga를 적용했을 때, 상품서비스의 재고수량 동시성 이슈
 - Saga에 참여하는 로컬 트랜잭션이 커밋과 동시에 외부에 노출되기 때문에 “**격리성 저하**” 발생
- 해결 방안
 - **Semantic Lock** : 애플리케이션 수준에서 data Flag를 설정하는 락
 - **Commutative updates**(교환적 업데이트) : 업데이트 순서에 상관없어도 되게끔..
 - **Pessimistic view**(비관적 관점) : 시가 단계 순서를 재조정하여 비즈니스 리스크 최소화
(**Dirty reads** 방지)
 - **Reread value**(값 다시 읽기) : 데이터 쓰기전 사전 변경 데이터 확인 (**Lost update** 방지)
 - **Version file**(버전 파일) : 순서를 재조정할 수 있게 업데이트를 기록
 - **By value**(값에 의한) : 요청별 위험성을 기준으로 동시성 메카니즘 동적 선택

Isolation이 부재한
Saga ACD “Solutions”

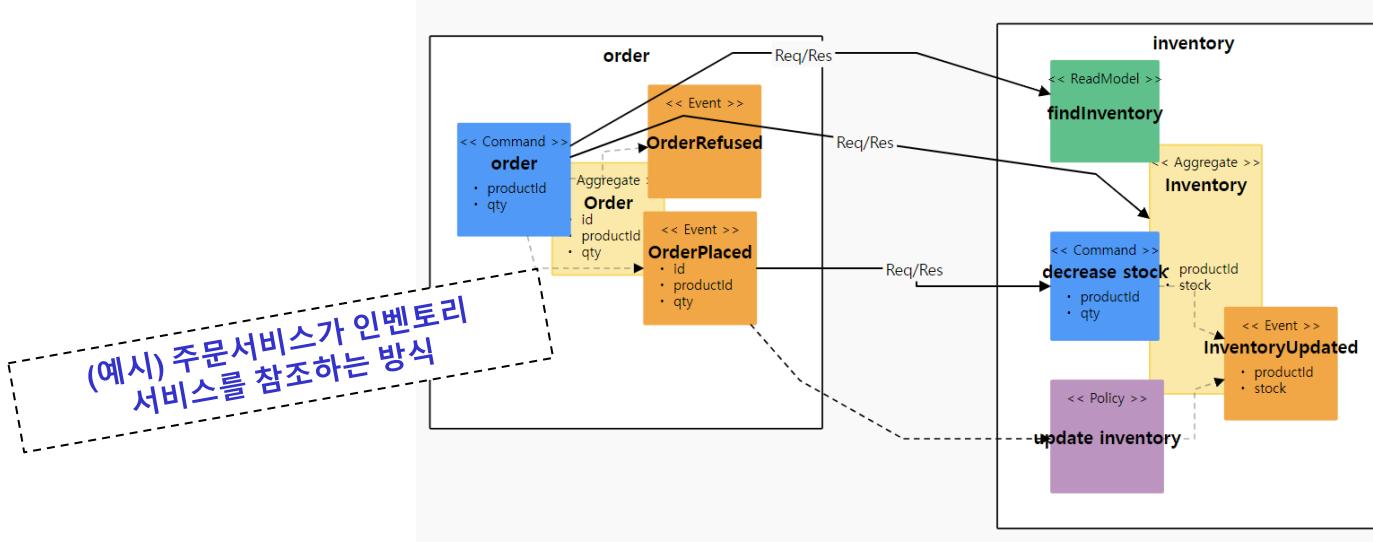
Semantic Lock

Commutative updates

Pessimistic view

(Issues #6) 분산 환경에서 서비스간 테스트를 어떻게 수행할 것인가?

- 마이크로서비스들은 다양한 방식과 API로 상호 참조됨
- 전제는 디펜던시 마이크로서비스가 실행 중이 아니어도 테스트와 병렬 개발이 가능해야 함

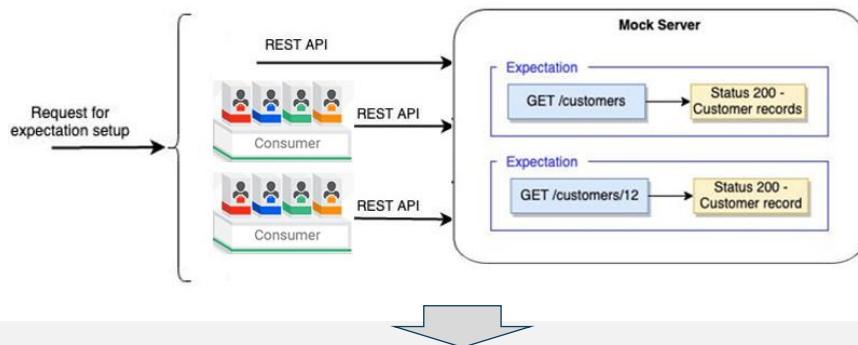


→ 주문 서버는 인벤토리 가짜서버(Mock Server)를 바라보고 테스트 및 병렬 개발이 가능해야 ..

(Issues #6) 분산 환경에서 서비스간 테스트를 어떻게 수행할 것인가?

- 디펜던시 서비스 Mocking기반 테스트 방법론

✓ REST API Mock 서버를 활용한 Test Double

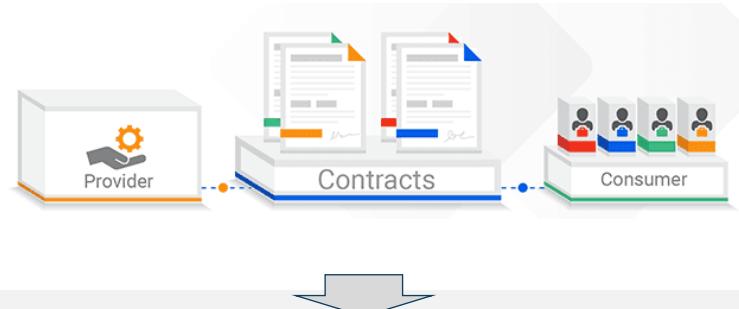


➔ Tools : Microcks, Prism

➔ API 기반 Mock Server 생성 & Kafka Mocking 지원

➔ Swagger UI, Example Spec.(given/when/then) 지원

✓ Contract DSL을 활용한 Test Double



➔ Tools : Pact, Spring Cloud Contract

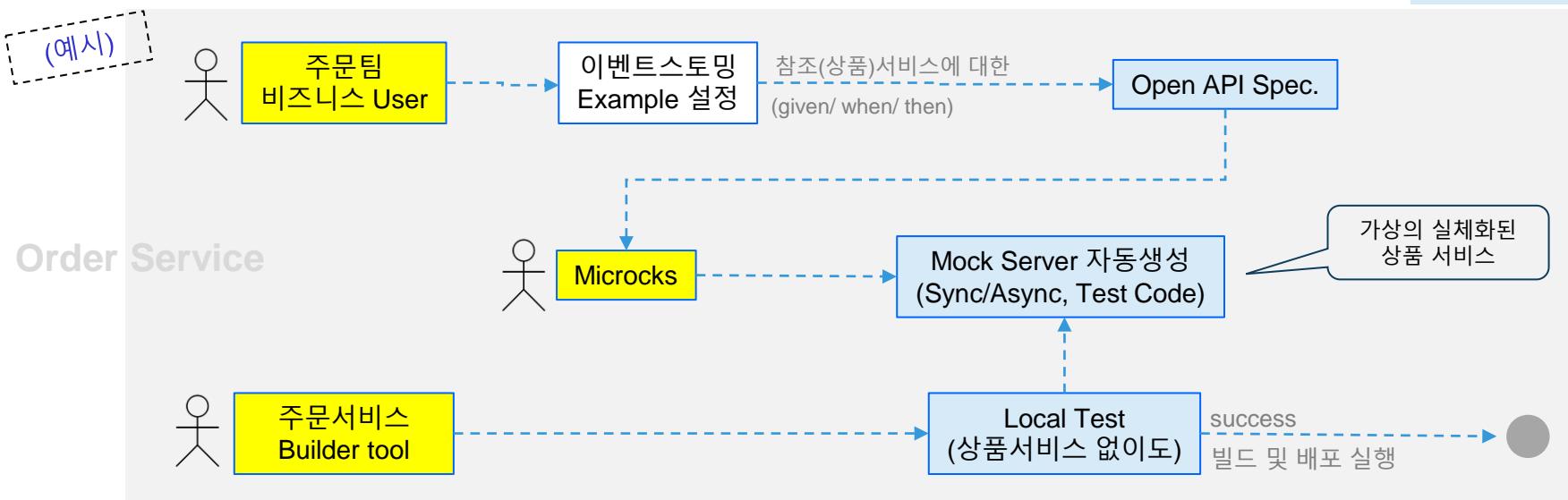
➔ Contract 기반 Test 결과를 Mock Server로 생성

➔ Stub Runner 통해 WireMock 테스트 및 상호 검증

(Issues #6) 분산 환경에서 서비스간 테스트를 어떻게 수행할 것인가?

- REST API Mock Server 기반 테스트 흐름 : **Microcks** 

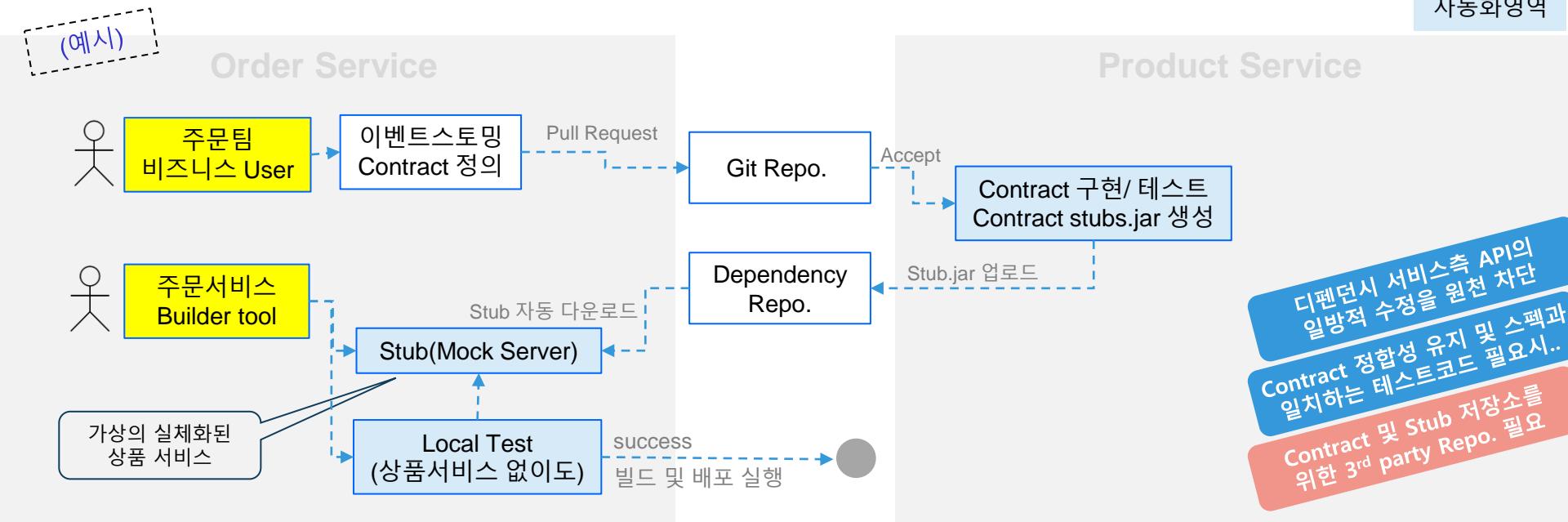
자동화영역



(Issues #6) 분산 환경에서 서비스간 테스트를 어떻게 수행할 것인가?

- Contract DSL 기반 테스트 흐름 : **Spring Cloud Contract** 

자동화영역



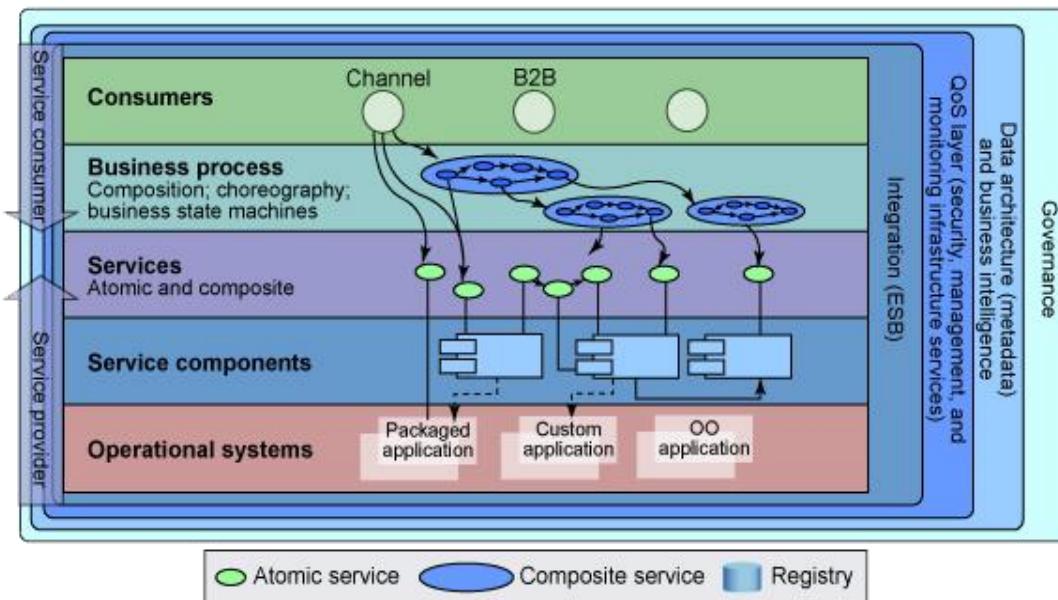
(Issues #7) 중복된 기능과 데이터를 어떻게 할 것인가?

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

Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA ✓
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

Integration Patterns



By UI

By Request-Response

By Event-driven Architecture

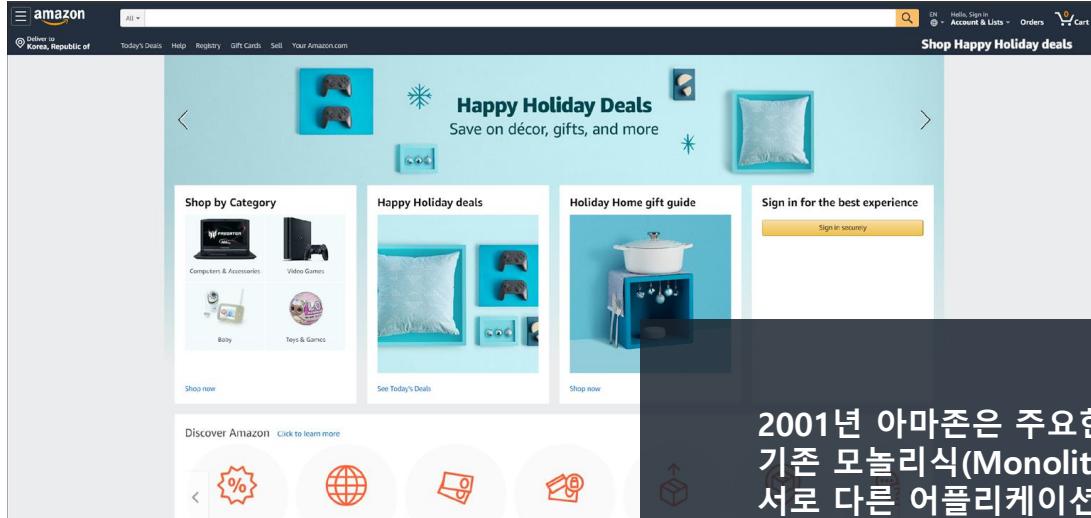
“

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.

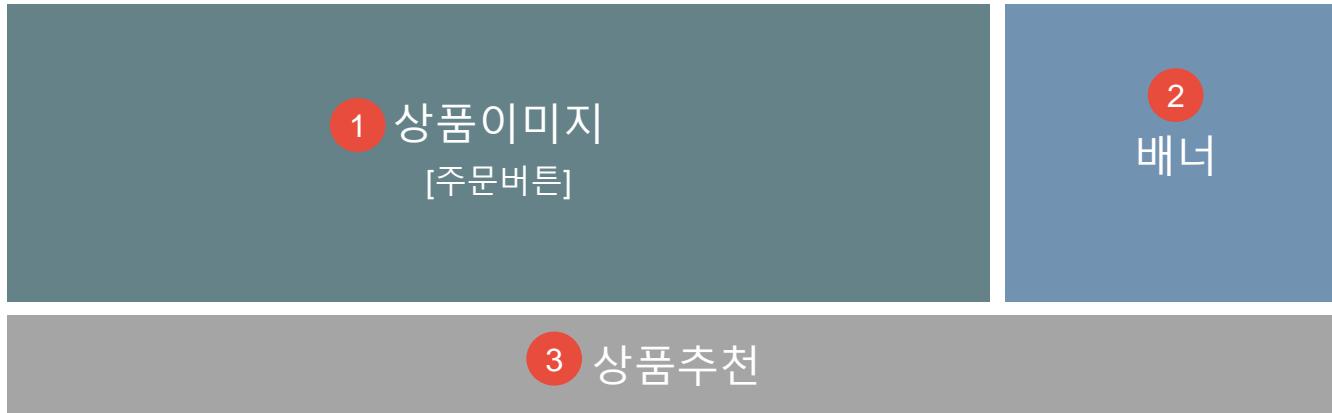


2001년 아마존은 주요한 아키텍쳐 변화가 있었는데
기존 모놀리식(Monolithic) 기반에서
서로 다른 어플리케이션 기능을 제공하는
“분산 서비스 플랫폼”으로 변화 하였습니다.

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

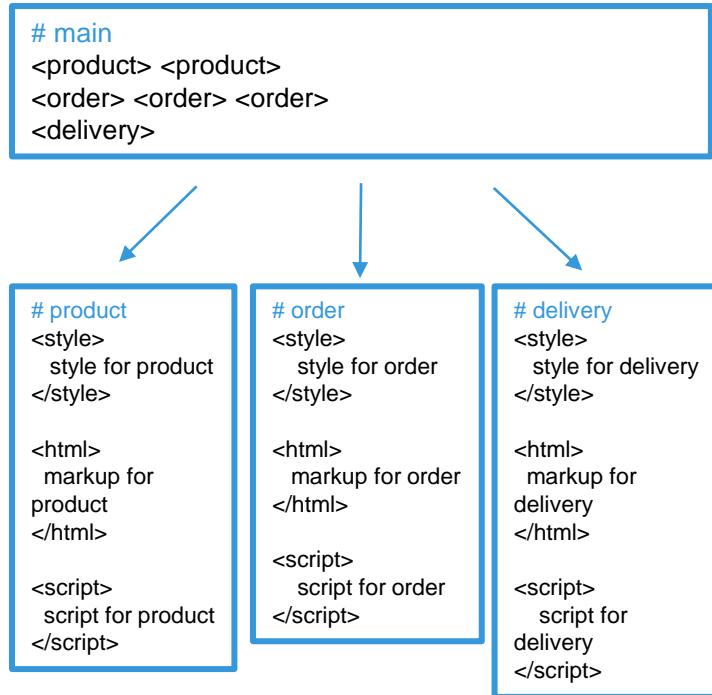
W3C Web Components

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

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

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

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



Almost all the frontend frameworks support Web Components



Polymer



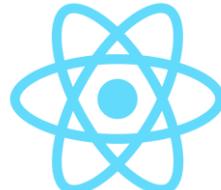
VueJS



AngularJS



ember



React

Custom tag for Domain Class: <product> tag

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

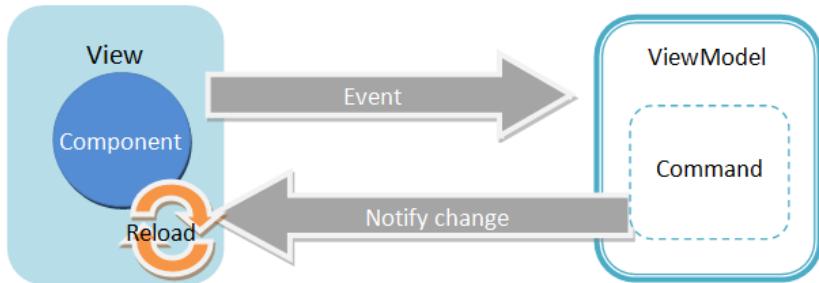
  </tr>
</table>
```

Data binding to Domain Tags

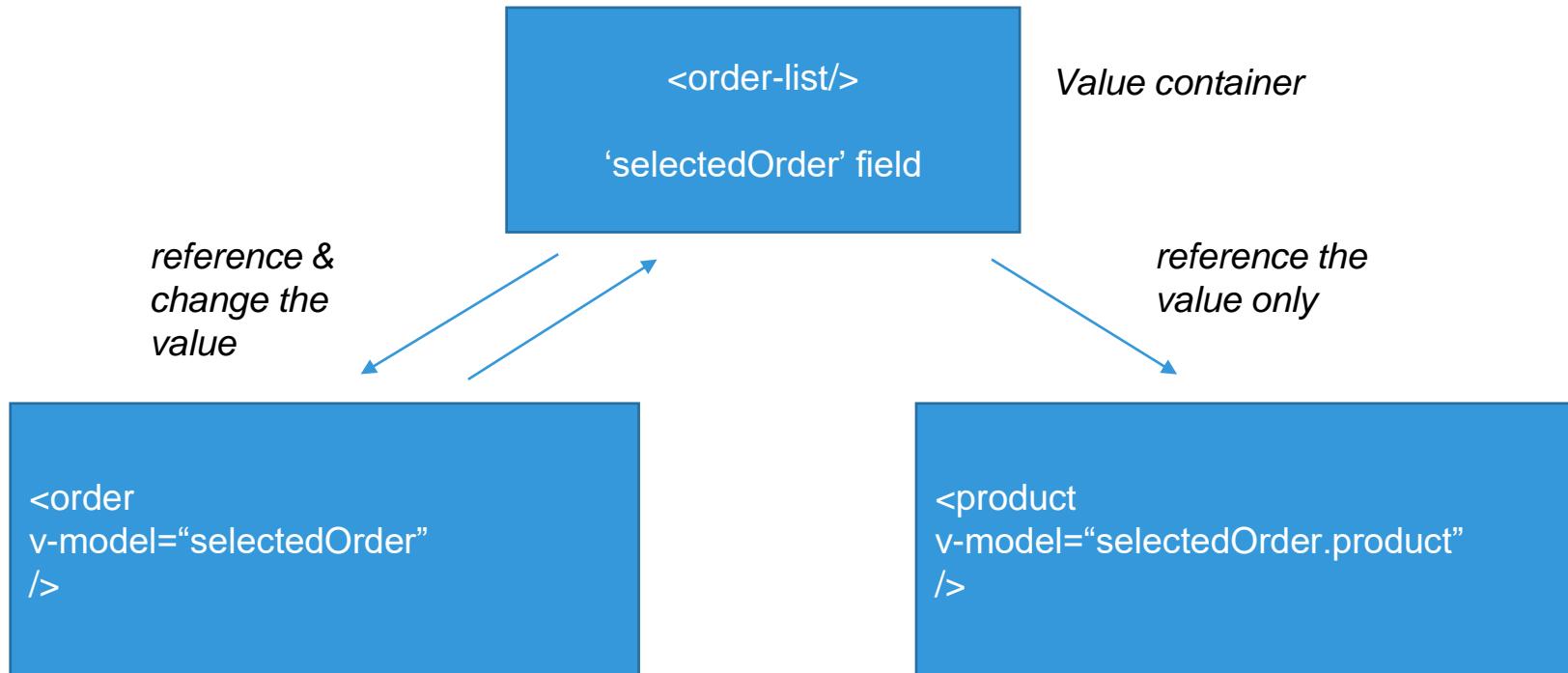
```
<product
  v-model="props.items[index]"
  @inputBuy="showBuy"
  @inputEdit="showEdit"
></product>

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

MVVM



Interaction between Domain Tags

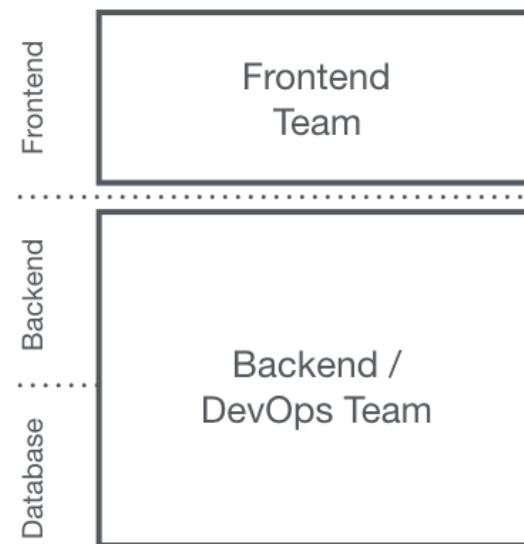


Monolith-Frontend

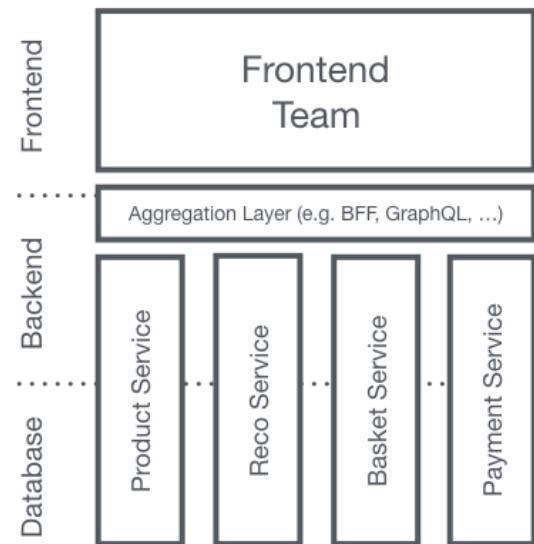
The Monolith



Front & Back

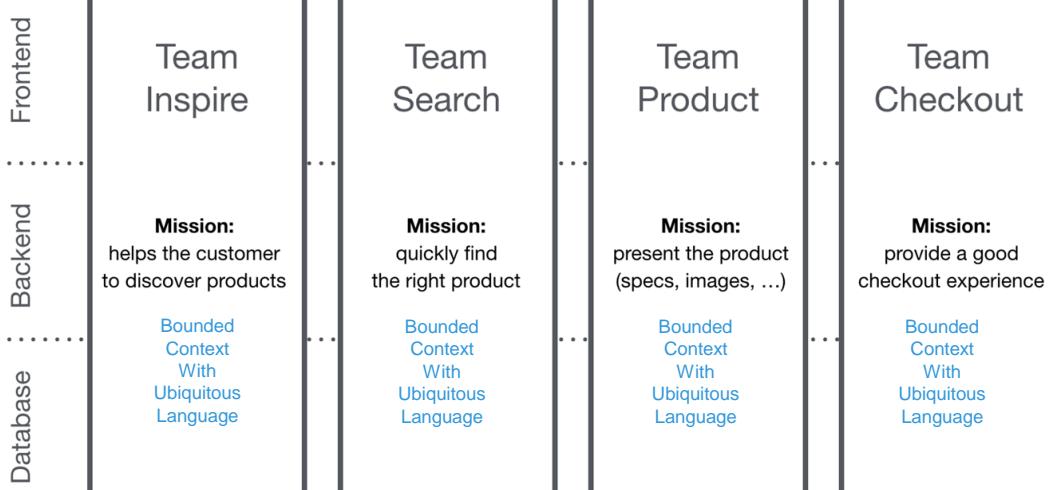


Microservices



<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?

James

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

Tell the components

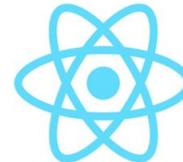
Angular



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

Say hello

React



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

Say hello

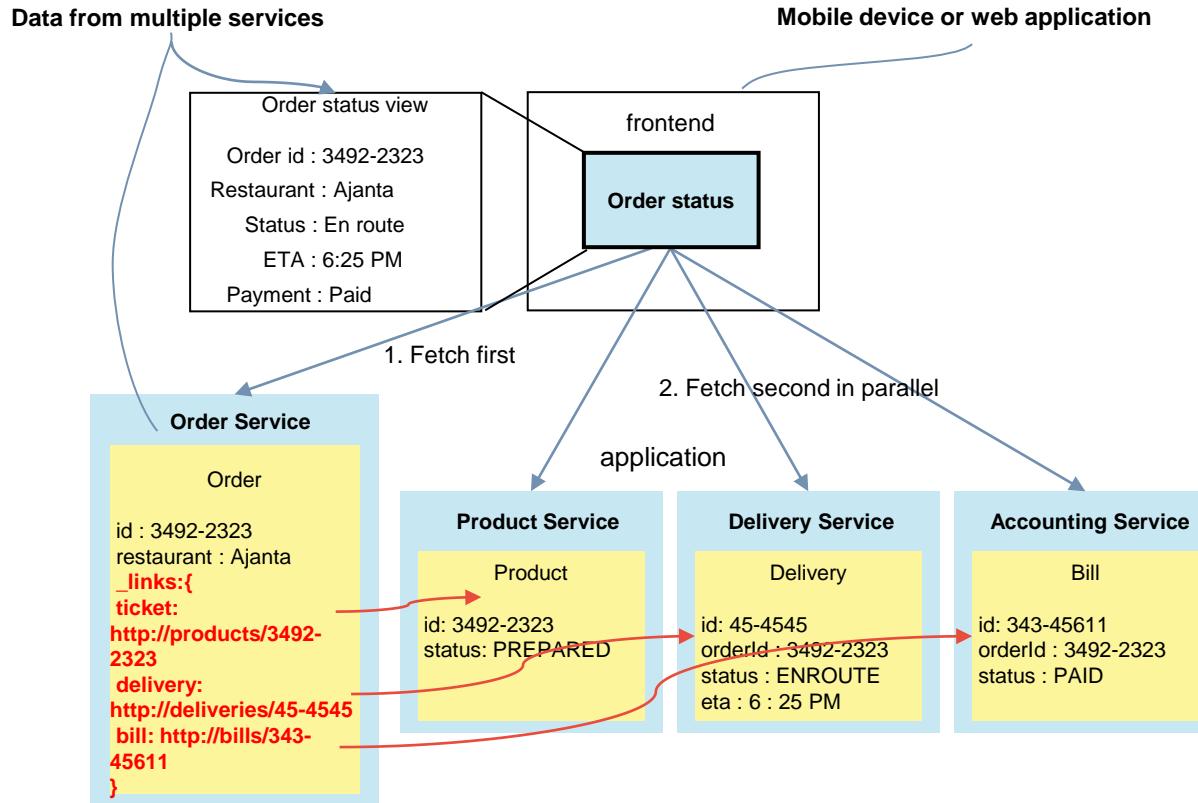
While they provide:

- 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



Issues :

- 단점 : UI 개발 복잡성과 성능

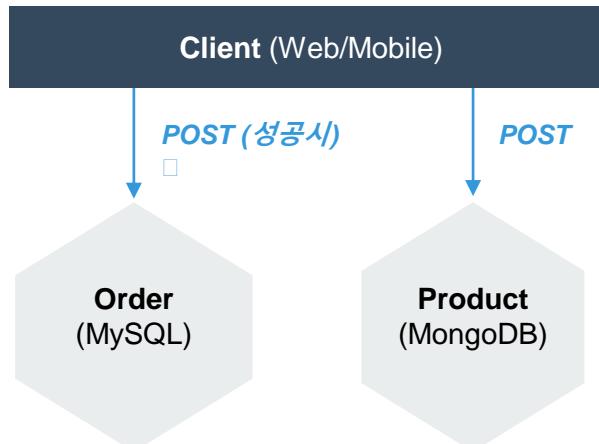
Write-Side using UI:

Distributed Transaction Problem in data mutation by UI composition

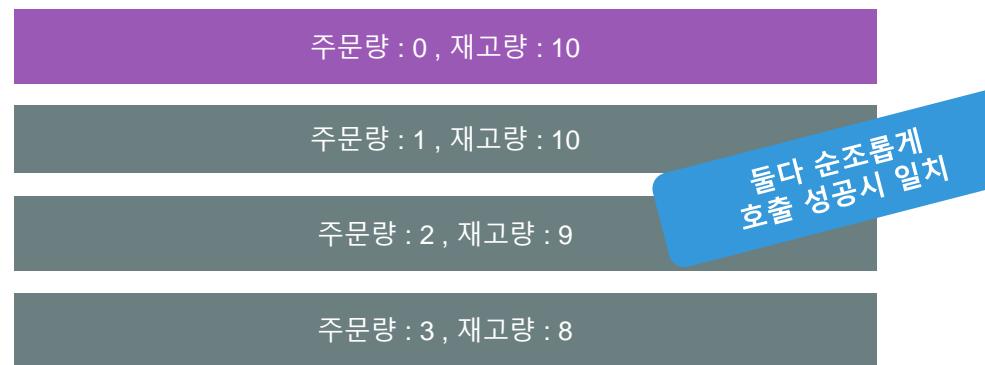
In-consistent

Consistent

- 서비스구성 (클라이언트가 순차 호출)



- 조회화면



Integration Strategies Compared

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

Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven ✓
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

“

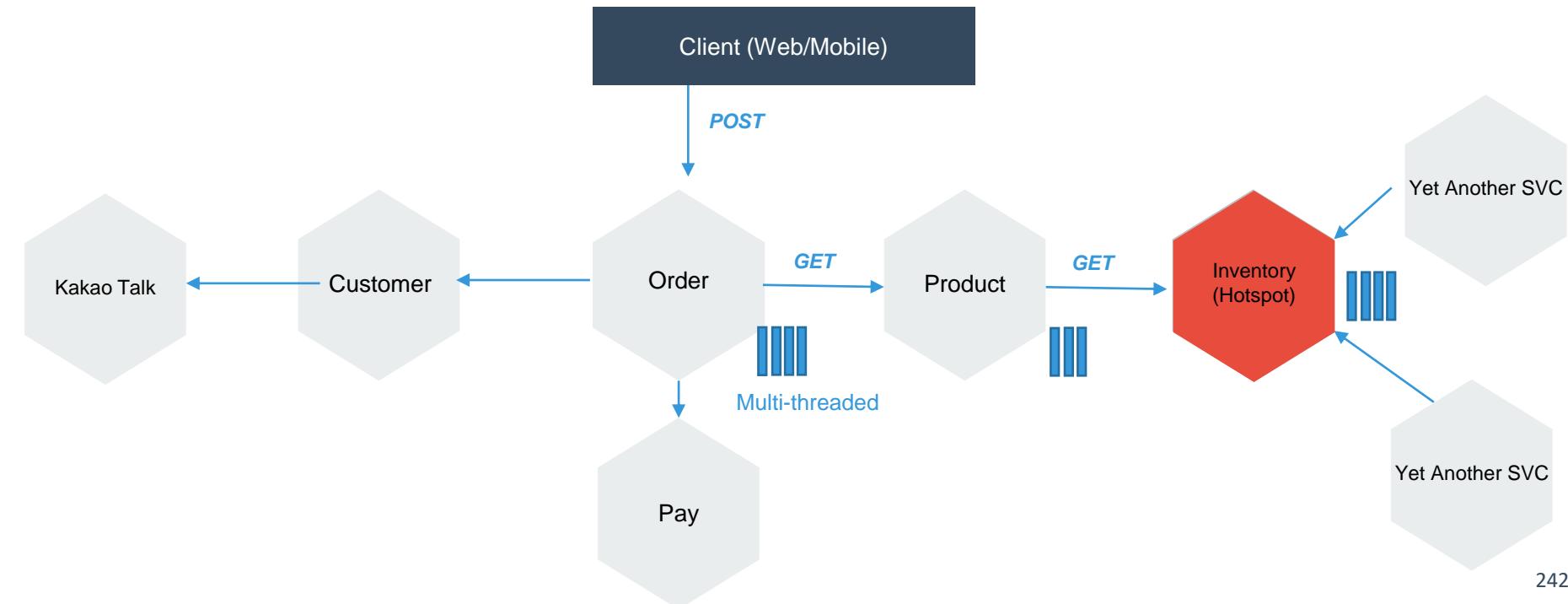
Service Integration by Request-Response

- 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

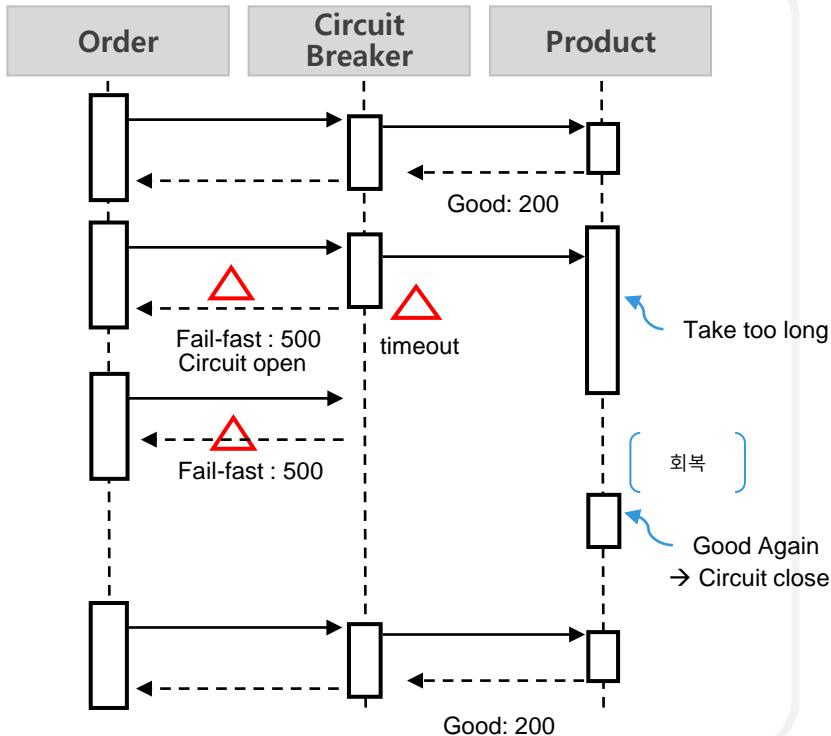
Data Projection in Request-Response model

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



Lab Time: Circuit breaker using istio

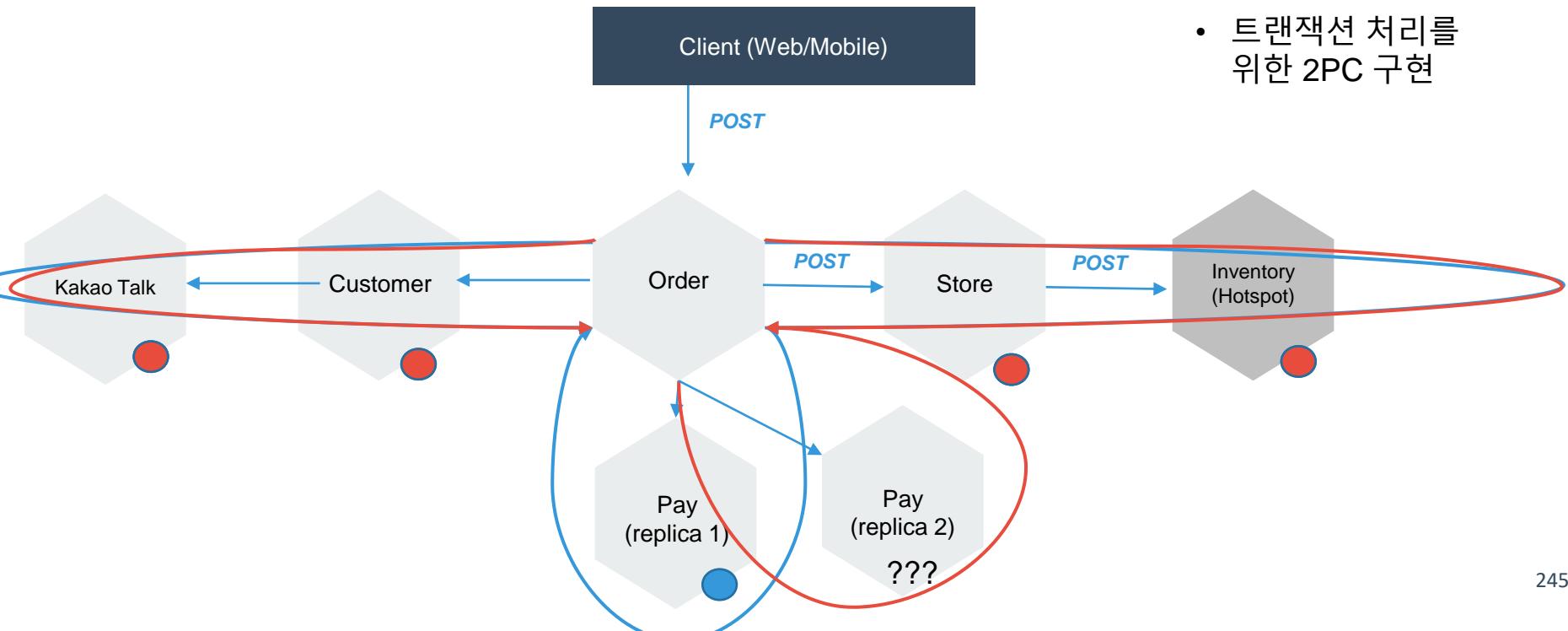
- product 서비스에 서킷 브레이커 적용
 - connectionPool :
지정된 서비스에 connections를 제한하여 가능 용량 이상의 트래픽 증가에 따른 서비스 Pending 상태를 막도록 *circuit breaker*를 작동시키는 방법
 - outlierDetection :
오류 발생하거나 응답이 없는 인스턴스를 탐지하여 *circuit breaker*를 작동시키는 방법
- 부하를 주어서 지정된 커넥션 만큼 차단 확인
 - siege -c2 -t10S -v --content-type "application/json"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

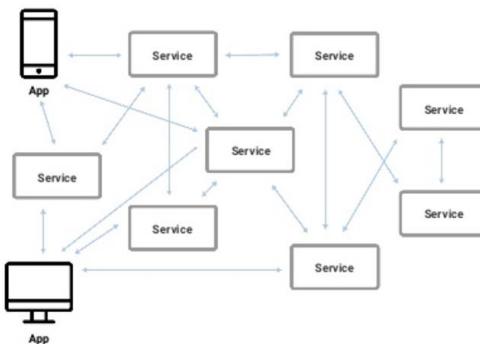
- Event-driven Approach
- ACID Tx. vs Eventual Tx.
- Business Transaction / Compensation (Saga) Pattern

Microservice Integration with Event-driven Architecture

Request-Response Applications

Deterministic
Rigid
Tight coupling

confluent

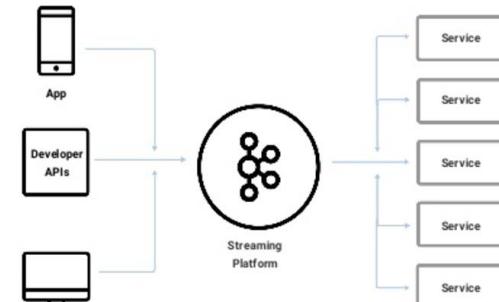


VS

Event-Driven Applications

Responsive
Flexible
Extensible

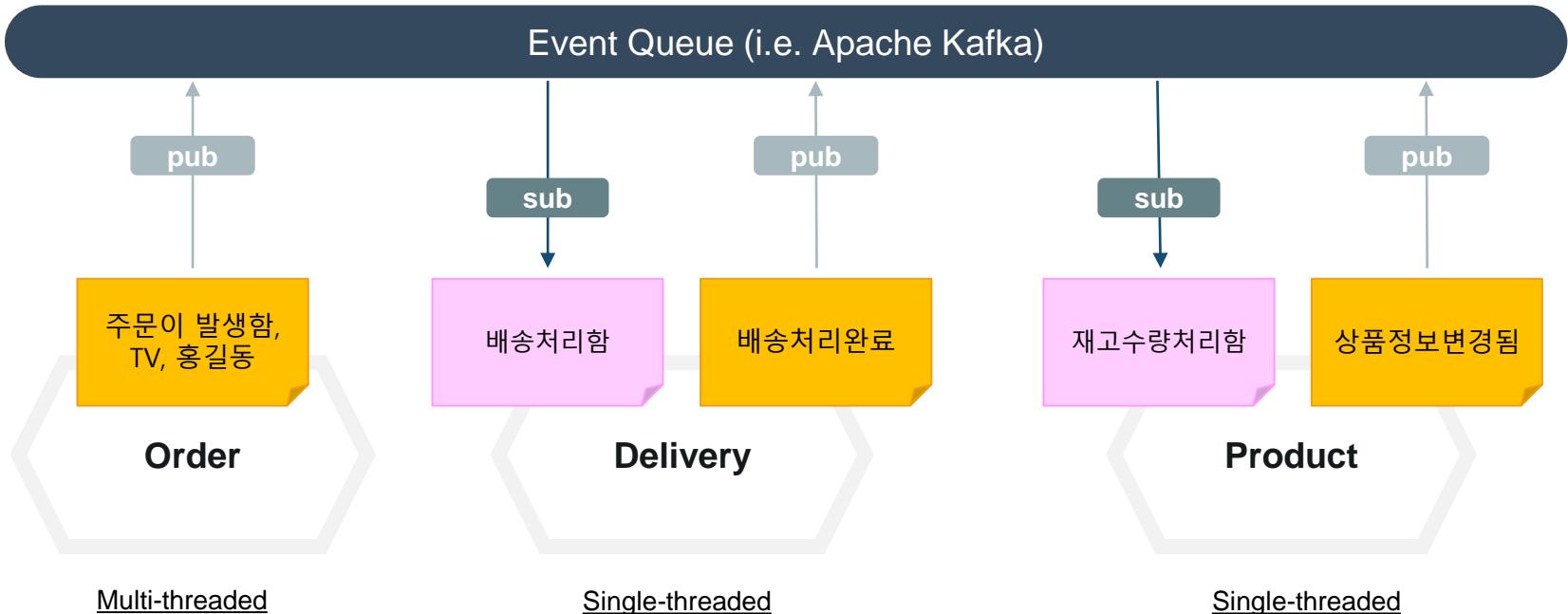
confluent



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

- Broadcasting
- Non-Blocking Model
- System fault isolation

Inter-microservice Call - Event Publish / Subscribe



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 이벤트가 들어오면..
    deliveryService.start(orderPlaced.getOrderId());
}
```

Event Consumer : Product

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

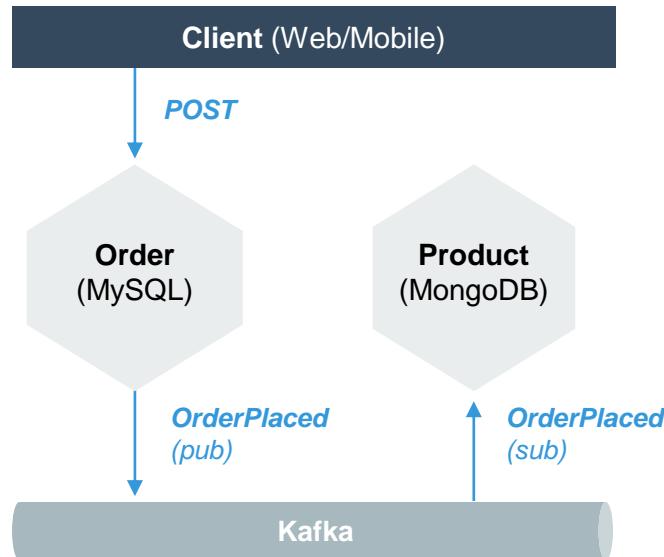
    // 해당 상품의 재고량을 주문량 만큼 줄여서 저장
    Product product = productRepository.findById(orderPlaced.getProductId()).get();
    product.setStock(product.getStock() - orderPlaced.getQuantity());

    productRepository.save(product);

}
```

Eventual Consistency를 통한 분산 트랜잭션 처리

- 서비스구성 (Eventual TX)



- 조회화면

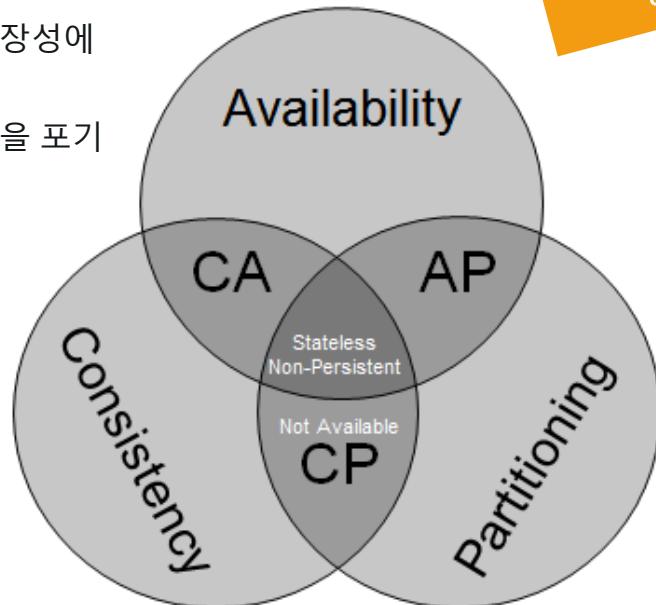


여기서 선택의 길을 만남...

- 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?

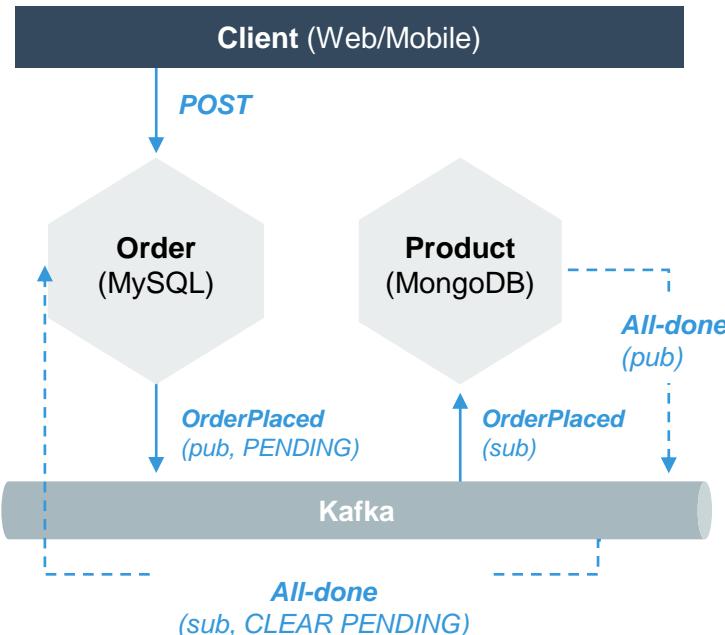
“성공한 내 서비스의 경쟁자들은 무엇을 포기했는가?”

▣ 강력한 의사결정 필요
(무엇으로 태어날 것인가...)



Eventual TX – 불일치가 Mission Critical 한 경우

- 서비스구성 (Eventual TX)

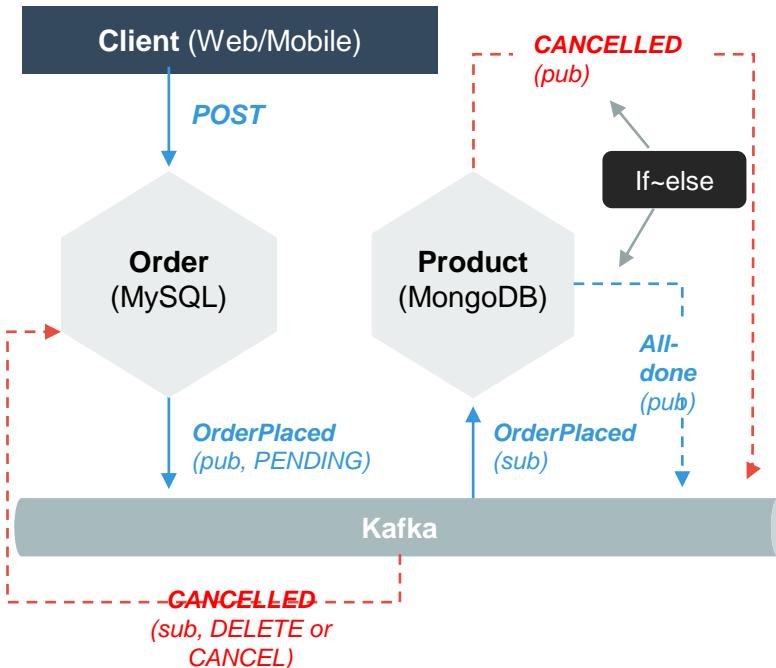


- 조회화면



Eventual TX – Rollback (Saga Pattern)

- 서비스구성 (Eventual TX)



- 조회화면



복구 &
결국 일치

Integration Strategies Compared

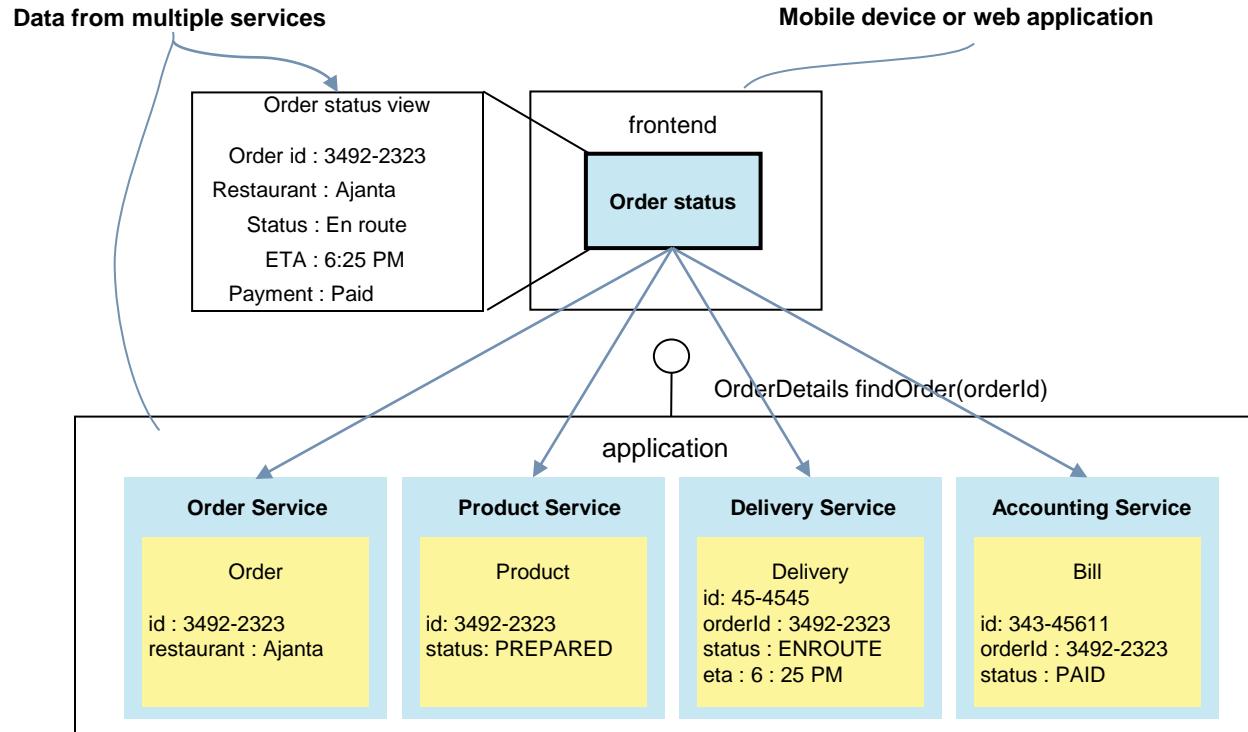
	UI	Req-Resp	Pub-Sub
읽기	UI Composition + HATEOAS	GET Request + Circuit Breaker	
쓰기	Not Recommended	2PC Not Recommended	Saga

“

Data Query (Projection) in MSA

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

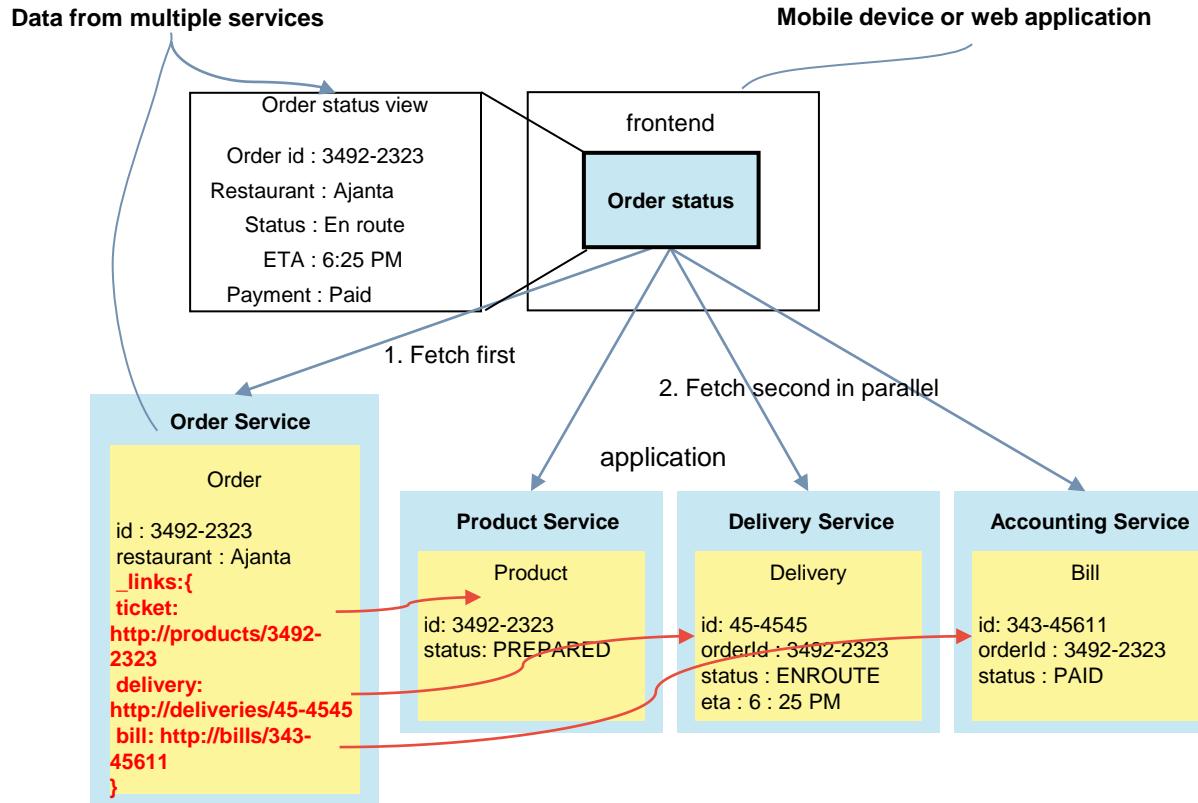
분산서비스에서의 Data Projection Issue



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
        })
    })
})
```

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

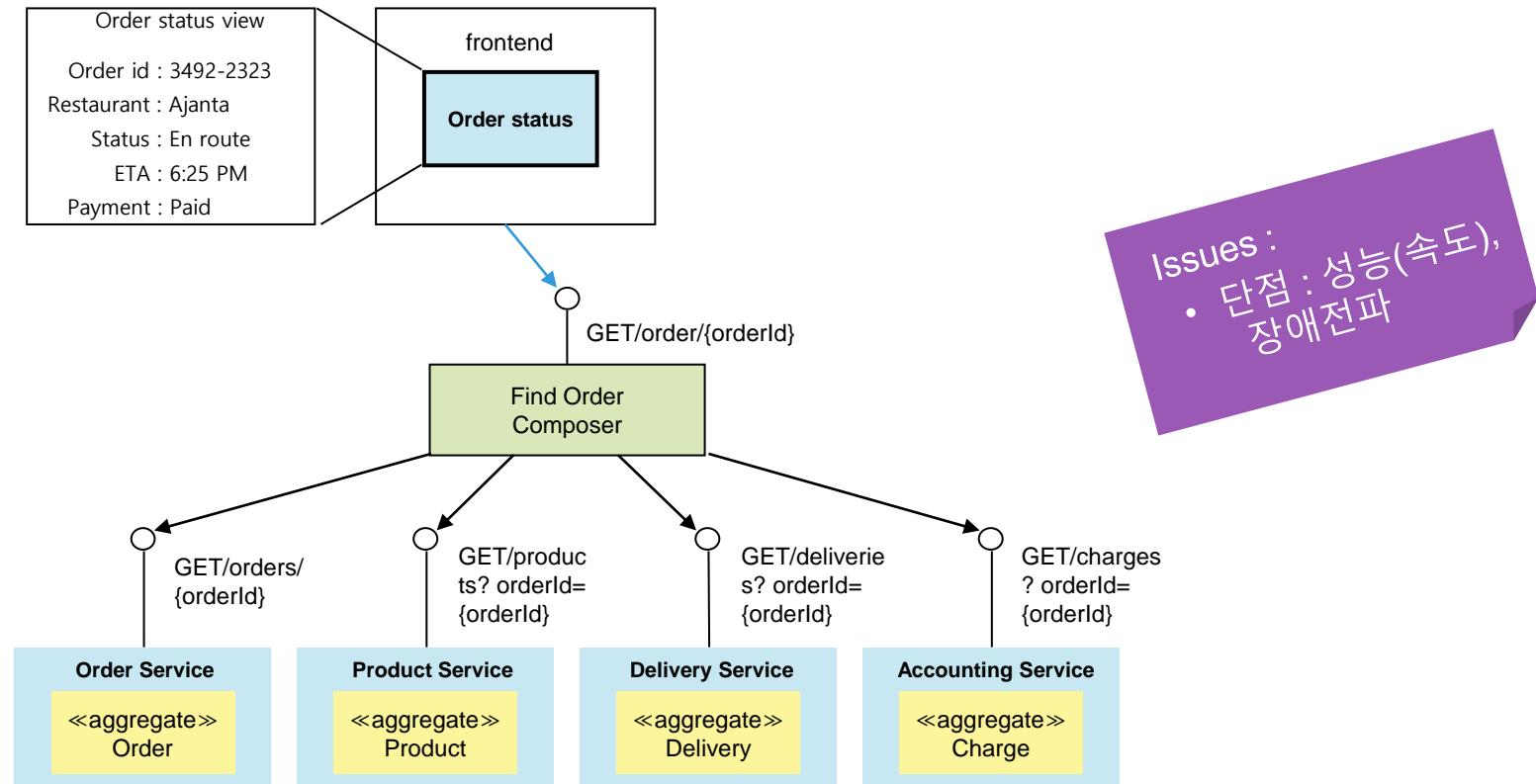
- 주문을 여러건 한다.
- Chrome의 개발자 모드 > Network 템을 연다.
- Menu의 My page (UI Mash Up) 을 열어서 API 날라가는 모습을 확인한다.

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",
      "quantity": 1,
      "unitPrice": 20000
    },
    {
      "productId": 3,
      "productName": "NOTEBOOK",
      "quantity": 1,
      "unitPrice": 30000
    },
    {
      "productId": 1,
      "productName": "TV",
      "quantity": 1,
      "unitPrice": 10000
    },
    {
      "productId": 1,
      "productName": "TV",
      "quantity": 1,
      "unitPrice": 10000
    },
    {
      "productId": 1,
      "productName": "TV",
      "quantity": 1,
      "unitPrice": 10000
    }
  ],
  "links": {
    "self": {
      "href": "http://localhost:8081/orders"
    }
  }
}
```

하위 데이터를 HATEOAS로 유지시 UI 개발이 편리

Data Projection by Composite Service or GraphQL



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

- 역할 : 사용자의 주문이력과 각 주문에 대한 상세 정보 (주문, 상품, 배송) 를 조합하여 출력
- 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());
});
});
```

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 많을 시에 모든 데이터를 조회하기 때문에 시간이 많이 걸림
 - 각 호출 API 별로 return 되는 data를 알고 있어야 함 (각 서비스에서 변경시 잊은 변경 요청)

Data Projection by GraphQL

- GraphQL
 - 페이스북에서 만든 쿼리언어로 “또 하나의 API 스펙” (<https://graphql.org/>)
- SQL과 비교
 - SQL : 데이터베이스 시스템에 저장된 데이터를 효율적으로 가져올 목적으로 주로 백엔드 시스템에서 작성하고 호출
 - GQL : 클라이언트가 데이터를 서버로부터 효율적으로 가져오는 것이 목적으로 주로 클라이언트 시스템에서 작성하고 호출
- REST API와 비교
 - GraphQL API는 주로 하나의 Endpoint를 사용
 - RESTful은 응답의 형태가 기본적으로 정해져 있어 필요한 정보만 부분적으로 요청하는 것이 힘듦
- 장점
 - REST API의 Overfetching 극복 → 열람하고 싶은 리소스의 일부 정보만을 Fetch 가능 (주문자 이름만 열람하고자 할 때)
 - REST API의 Underfetching 극복 → 주문에 따른 상품정보, 배송정보 등 교차정보를 한번에 가져오는 Query로 Fetch 가능
 - HTTP 요청의 횟수와 HTTP 응답의 Size를 줄일 수 있음

Lab Time: Data Projection by GraphQL (1/2)

- 참고 코드 실행
 - git clone <https://github.com/event-storming/graphql.git>
 - 주문/ 상품/ 배송 마이크로서비스 실행 (cd Folder > mvn spring-boot:run)
- 주문 하기
 - http localhost:8081/orders productId=1 quantity=1 customerId="1@uengine.org"
- GraphQL Composite Service 기동 (Lab에서는 npm, Apollo 기반 예제)
 - cd apollo_graphql
 - npm install
 - npm start
- GraphQL Playground 실행
 - http://localhost:8089로 접속한다.
 - Graph QL의 Type Definitions, Resolver 명세를 확인하고
 - Graph QL로 테스트 한다.

Lab Time: Data Projection by GraphQL (2/2)

- GraphQL Playground에서 GQL로 단순조회와 컴포지트 서비스를 조회한다.
- GQL Labs

- 주문 조회

```
query getOrders {  
  orders {  
    productId  
    productName  
    quantity  
    price  
  }  
}
```

- 주문 중 상품명만 조회

```
query getOrders {  
  orders {  
    productName  
  }  
}
```

- 1번 주문 조회

```
query getOrderById {  
  order(orderId: 1) {  
    productId  
    productName  
    quantity  
    price  
  }  
}
```

- 복합 서비스 조회

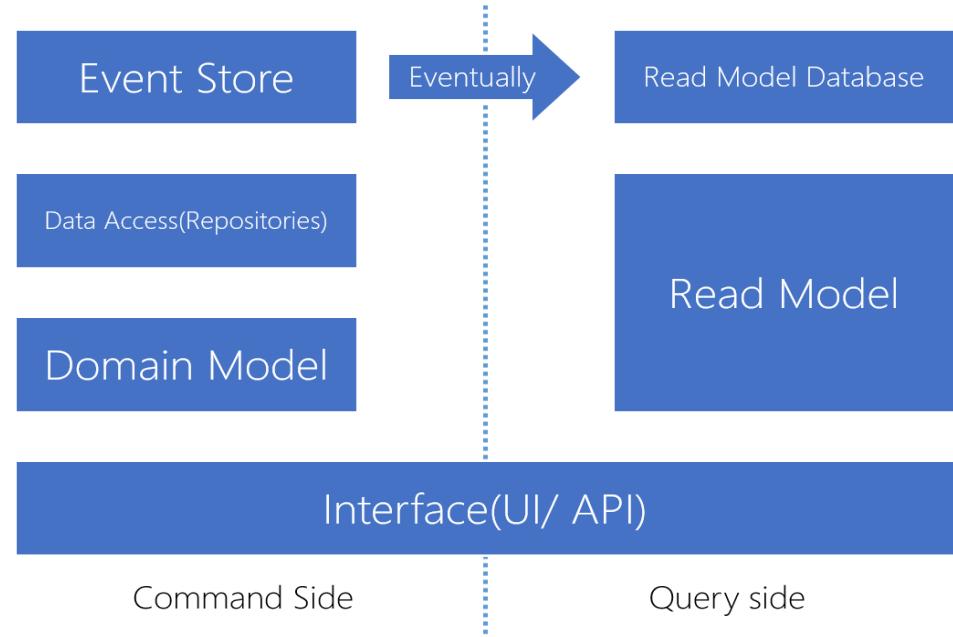
```
query Composite {  
  orders {  
    quantity  
    customerId  
    state  
    product {  
      price  
      name  
    }  
    delivery {  
      deliveryAddress  
    }  
  }  
}
```

- 단점

- 주문, 배송, 상품 서비스가 모두 가동중이어야 데이터 조회가 됨
- RESTful API에 비해 캐싱이나 파일 업로드를 위한 명세가 없어 별도 구현해야 함

생각을 다르게 하자 : CQRS and Event-Sourcing

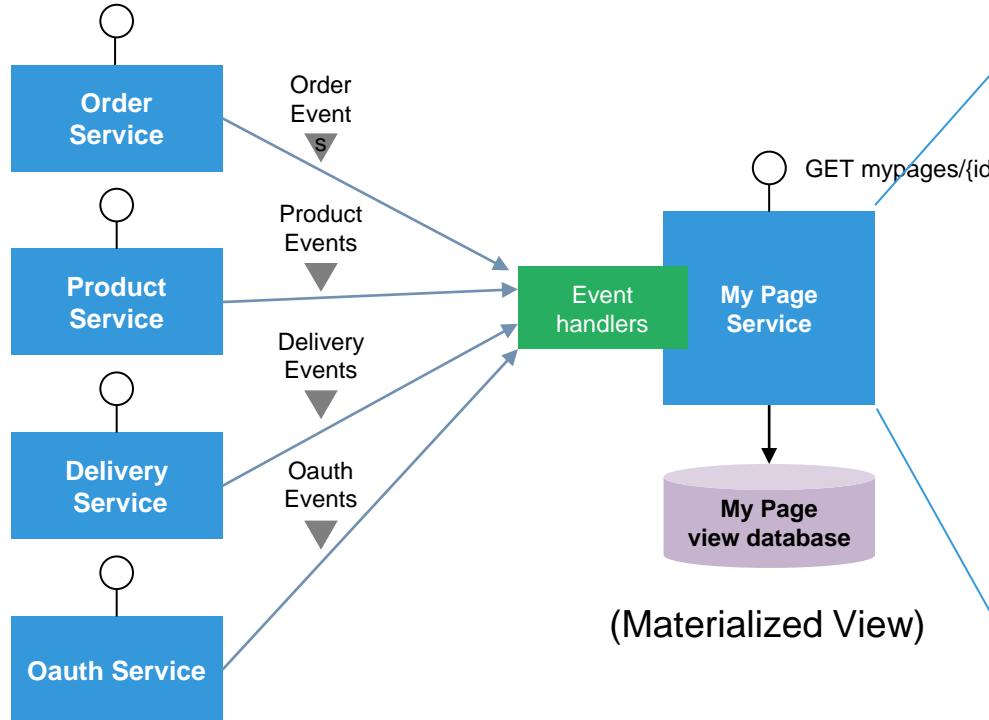
- 데이터 원본에서 매번 가져오지 말고,
취합된 별도의 뷰를 미리 만들어놓자
- Command (Write) / Query (Read) Responsibility Segregation
- 읽기전용 DB와 쓰기 DB를 분리함으로
써 빠른 읽기 구현
- Query 뷰를 다양하게 구성하여 여러 MSA 서비스 목적에 맞추어 각 서비스의
Polyglot Persistence 구현



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

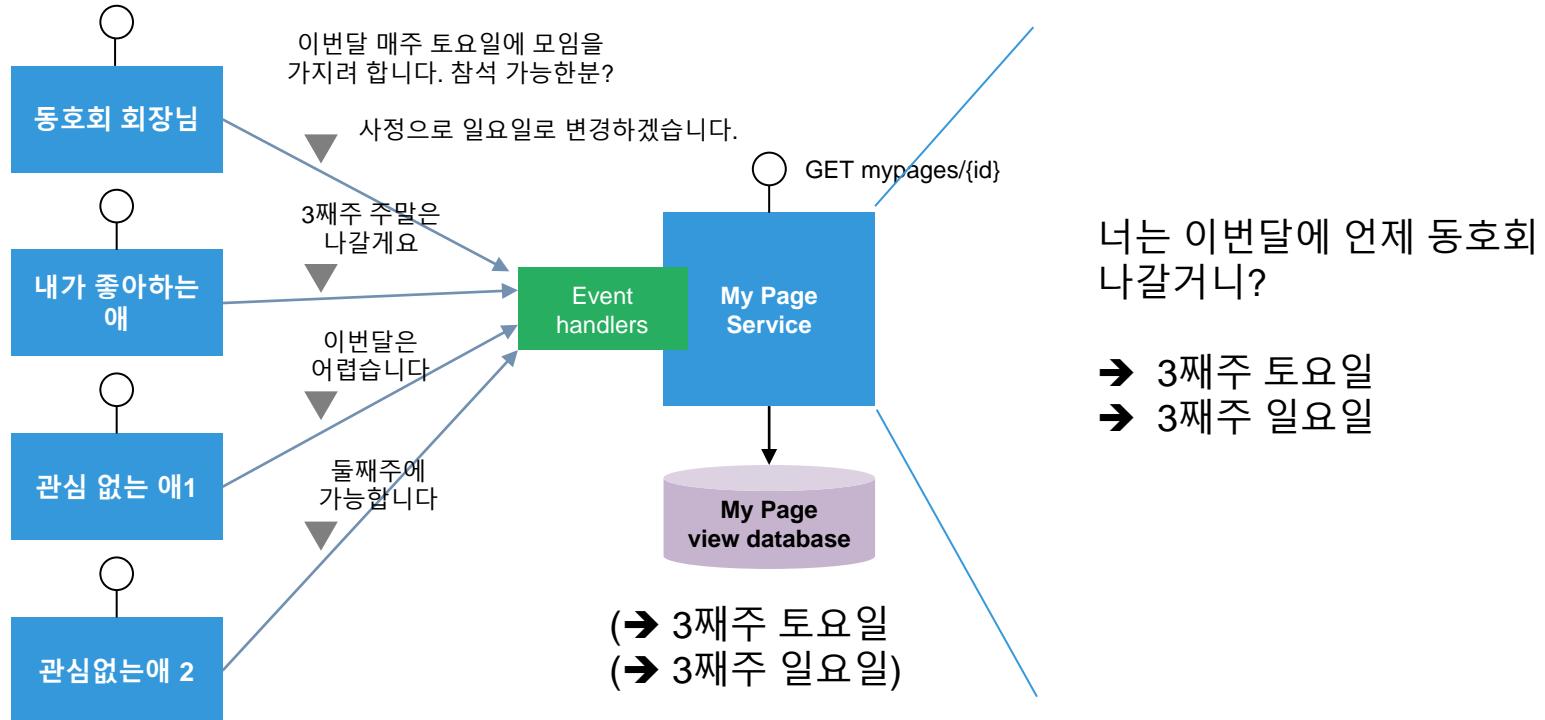
<https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-1-richardson>

CQRS 의 확장 : Multiple Event Sources



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

CQRS 의 확장 : Multiple Event Sources

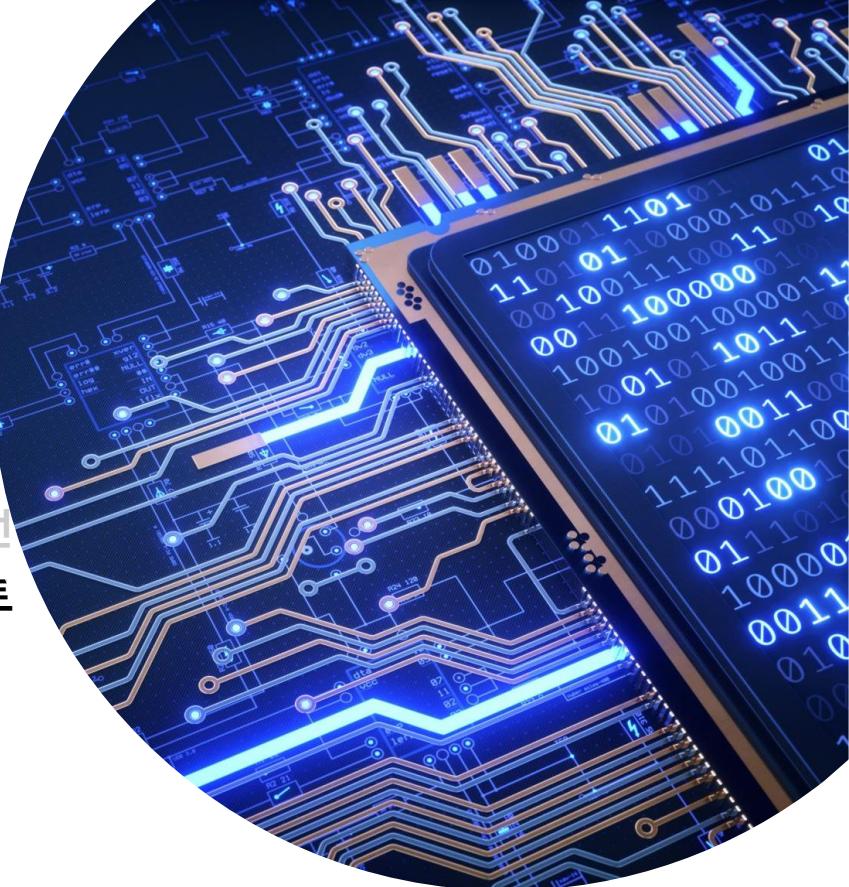


Integration Strategies Compared

	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 데이터든 상태를 재 생성할 수 있다.
 4. 1,2,3 모두 옳다
 5. 1,2,3 모두 틀렸다

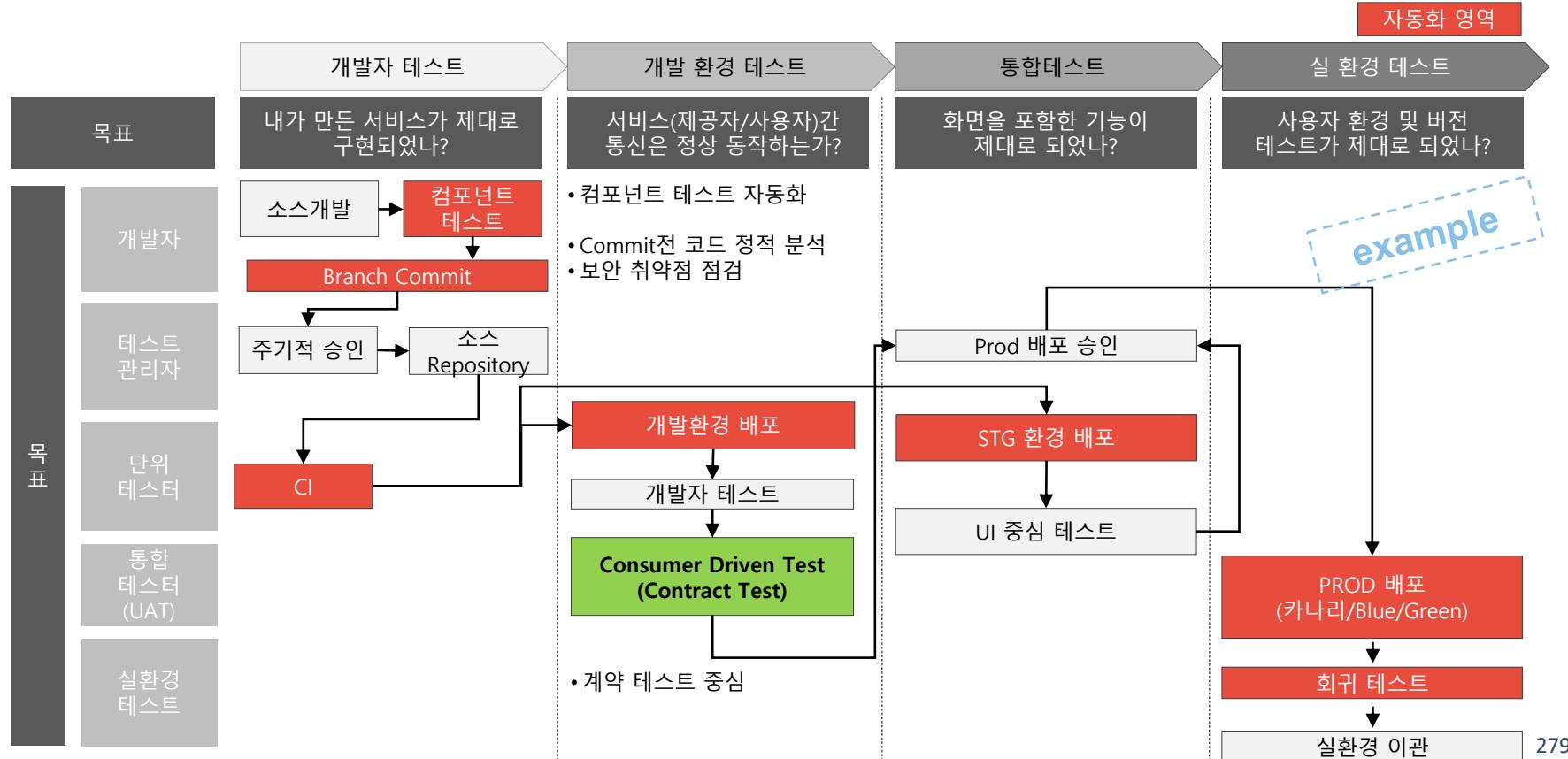
Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test ✓
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

“ Contract Test

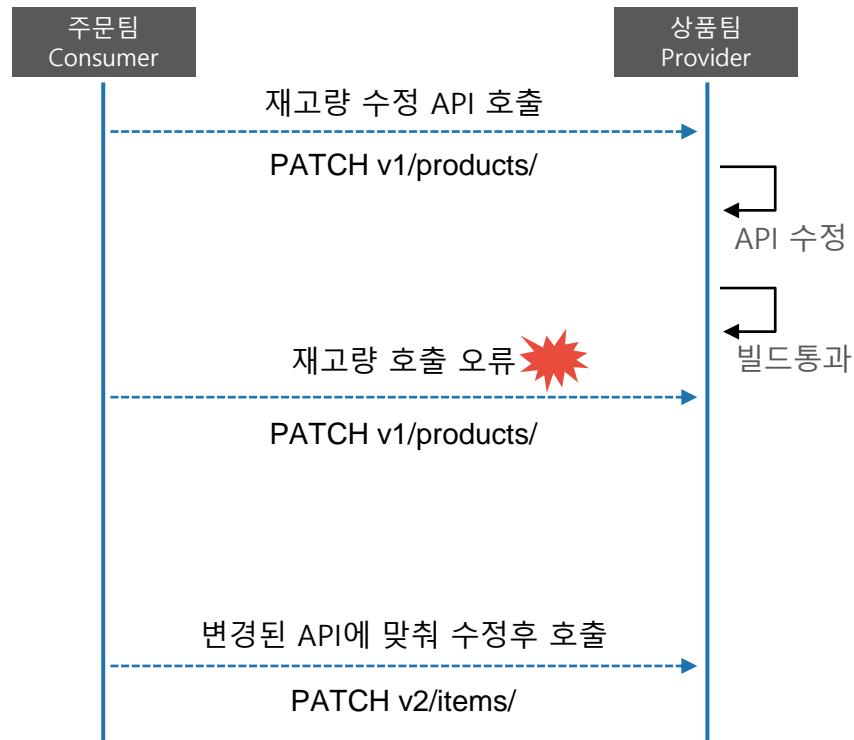
- Contract ?
- Consumer driven Contract (CDC)
- CDC Testing Frameworks and Tools

MSA Test & deploy Process

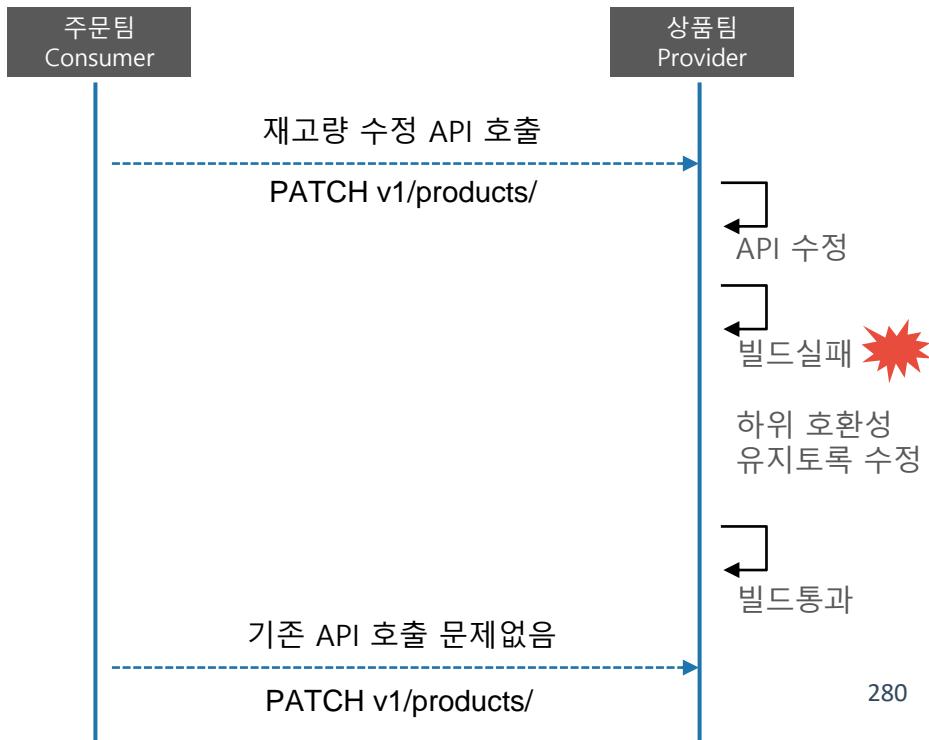


Contract

계약서가 없다면..

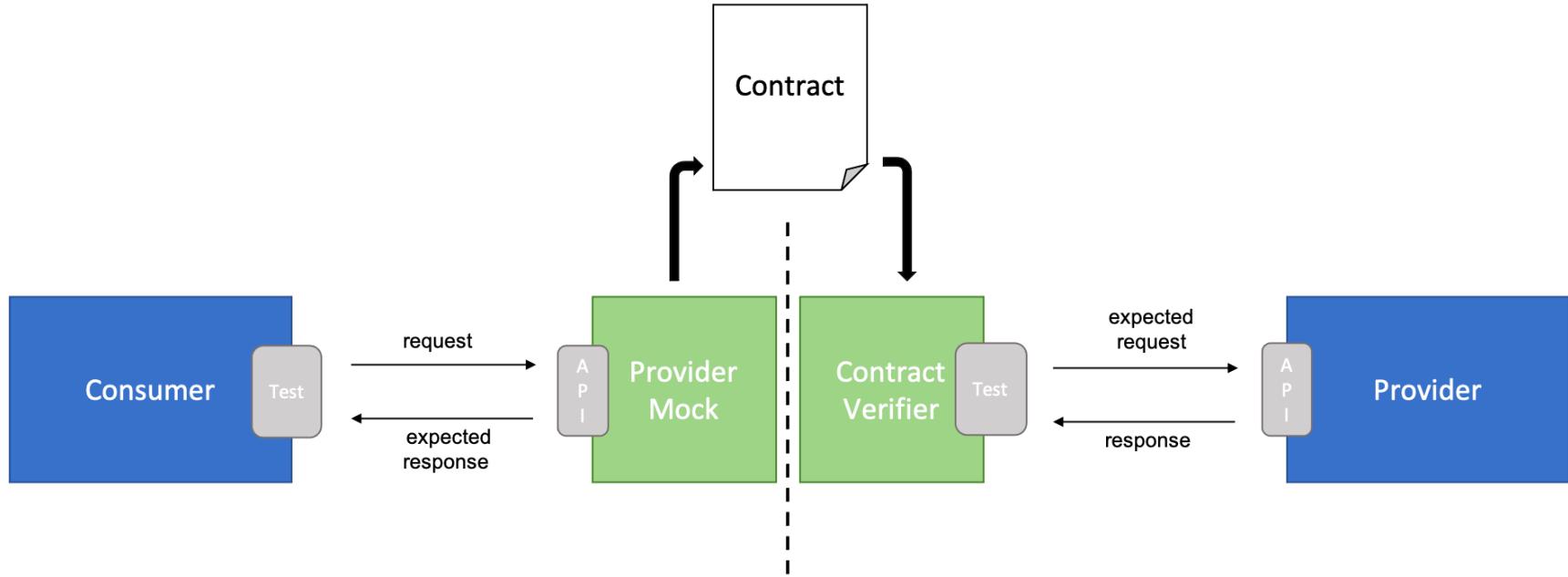


계약으로 합의되어 있으면..



What's Contract Test ?

- 제공자(Producer) 가 소비자(Consumer) 에 대한 약속을 준수하는지 확인하면서 격리된 시스템 구성 요소를 테스트하는데 사용되는 소프트웨어 테스트 방법론
- 소비자의 요구사항을 중심으로 제공자의 서비스를 진화시키기 위한 방법



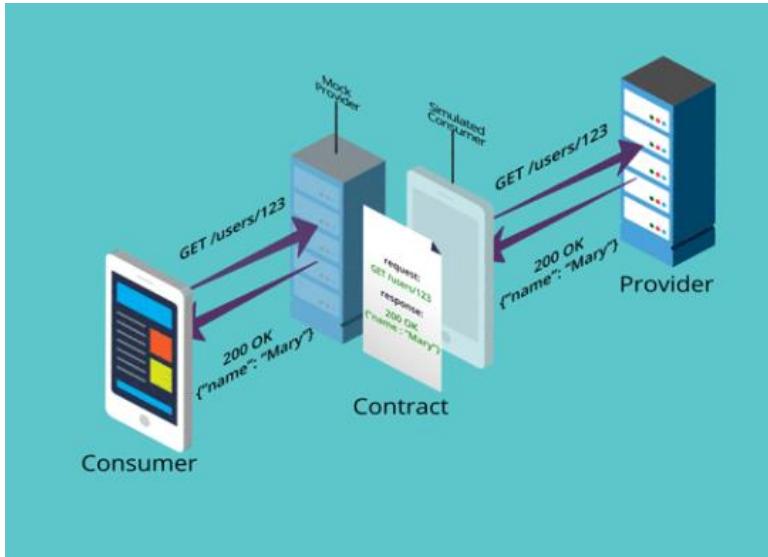
Why Contract Test ?

- E2E (End to End) 테스트는 사용자의 입장 수행하는 테스트로 일반적으로 느림
 - E2E 테스트는 비용이 많이 들고 유지 관리가 어려움
 - 대규모 시스템의 E2E 테스트는 전용 테스트 환경 외부에서 실행하기 어렵거나 불가능
-
- Contract Test는 문서화되어 있는 컨슈머의 규약을 준수하여 테스트하고,
 - Contract을 따르는 Mock*을 통해 격리된 컴포넌트간의 상호 작용을 테스트함으로써 E2E의 한계를 극복
 - 이러한 Contract 을 CDC (**Consumer-Driven Contract**)라고 부른다.

CDC Testing Frameworks and Tools

- Pact

- Contract에 사양을 정의하면서 Consumer에 먼저 Response를 Mocking하고, Producer가 이에 대한 상호 작용을 검증

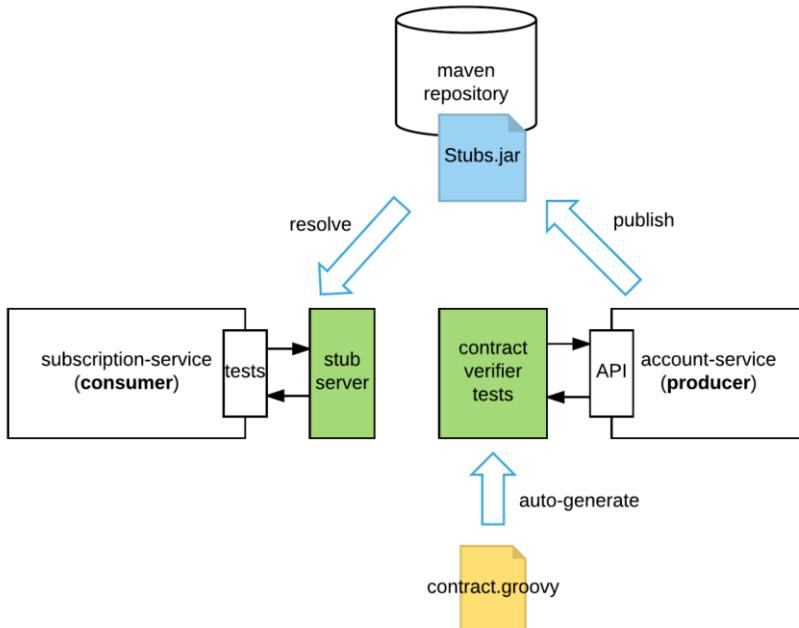


출처 : pactflow.io, how-pact-works

CDC Testing Frameworks and Tools

- Spring Cloud Contract

- Producer에서 테스트후에 생성된 stub을 Consumer에서 실행하여 테스트하므로 정확하고 빠른 테스트가 가능



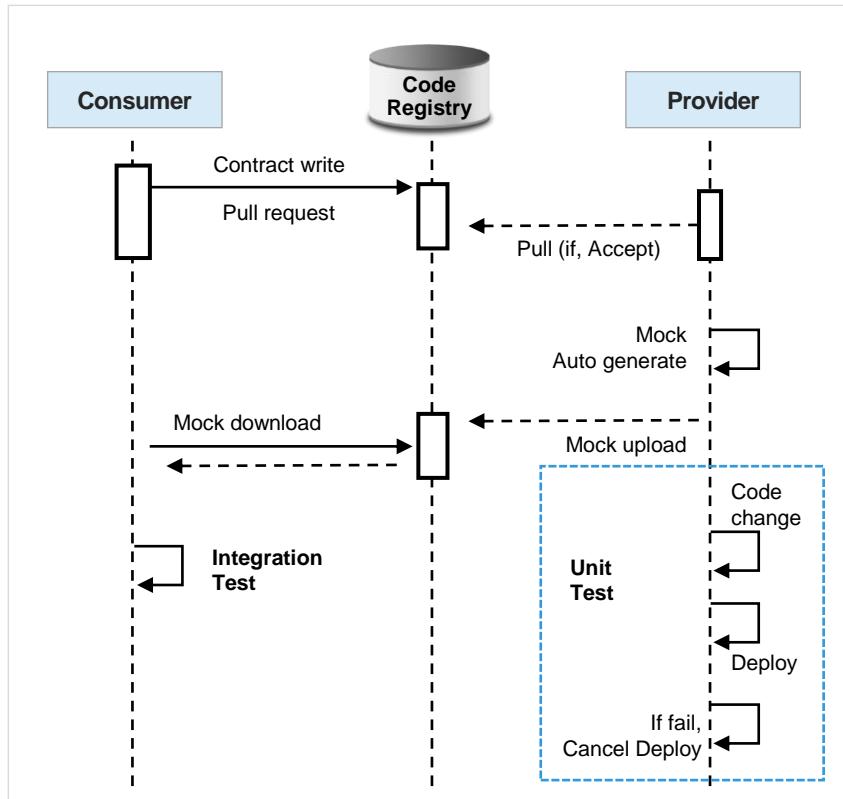
- Spring Cloud Contract Verifier :**
 - (Producer 측) Contract 등록역할
groovy, yaml, kotlin의 Contract에 대한 테스트 코드 자동 생성
- Spring Cloud Contract Stub Runner :**
 - Consumer 측에서 사용하며 Producer가 생성한 Stub으로 테스트 진행
- Spring Cloud Contract Wiremock :**
 - Consumer 측에서 테스트 시, 스텁 서버 역할

CDC, Consumer Driven Contract

✓ Contract Samples

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

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



- API Consumer가 먼저 테스트 작성
- API Provider가 Pull Request 수신 후 테스트 수행 및 Consumer Mock 객체 생성
- API Consumer는 생성된 Mock 객체를 통해 Stub 서버를 자동 생성하여 테스트 케이스 수행
- 이를 통해 Provider의 일방적인 버전 업에 따른 하위 호환성 위배를 원천 방지

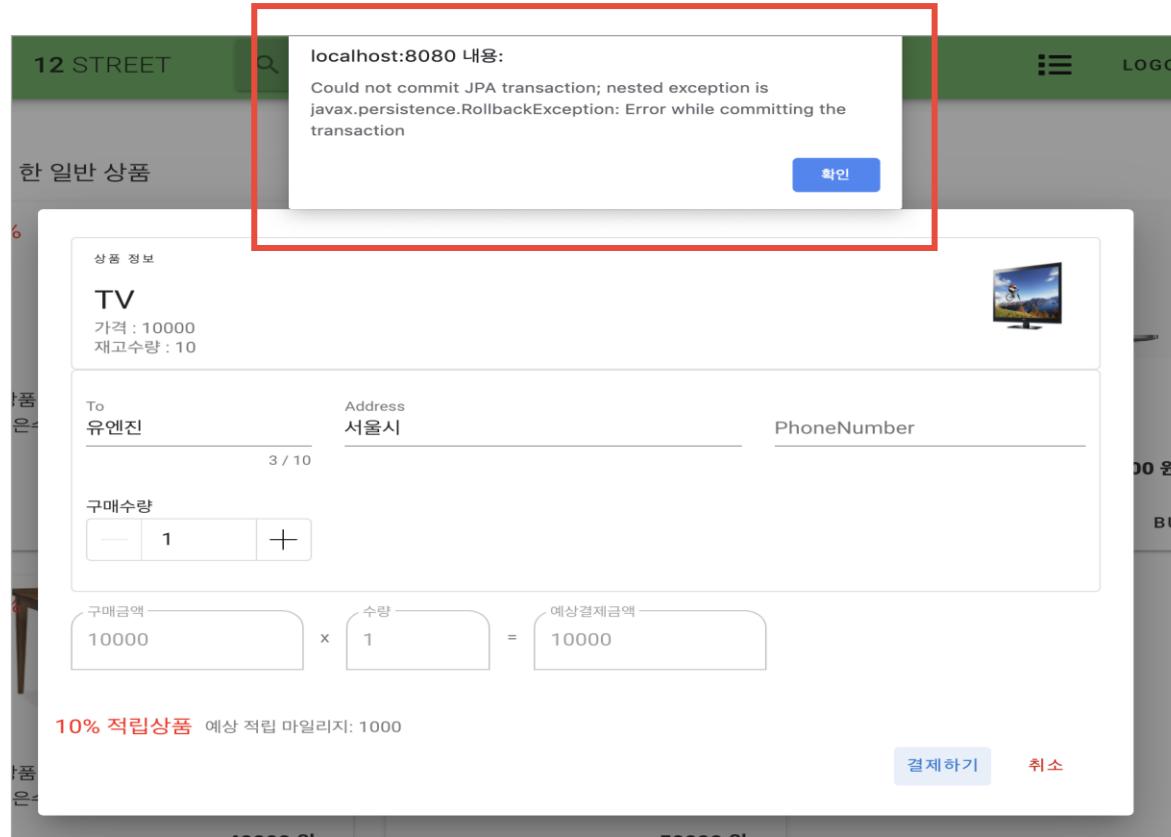
상품팀에서 API를 일방적으로 변경 후 배포한다면..

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 }
```

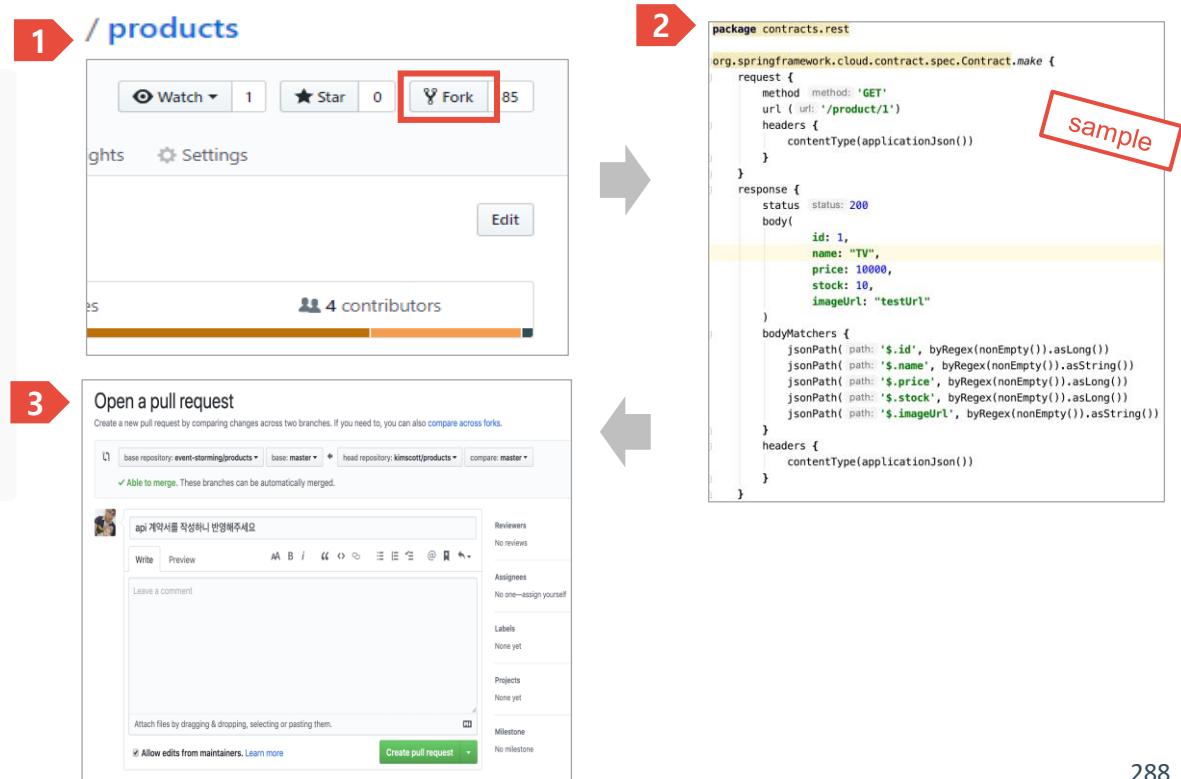
런타임에서 주문 서비스 장애는 불가피...



이를 방지하기 위해, 주문팀과 상품팀간 계약체결

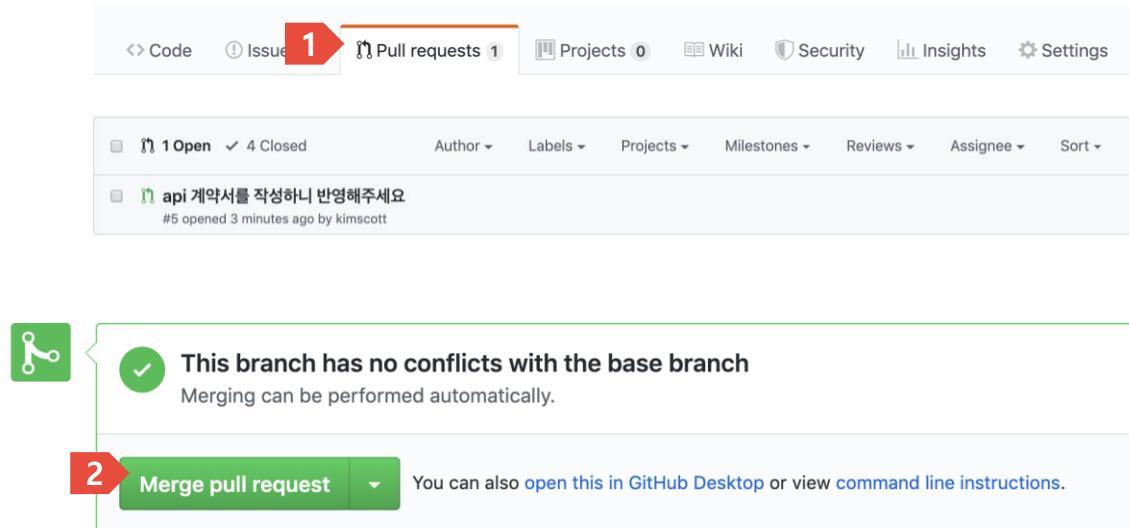
주문팀이 계약서 작성 후,
상품팀에 체결 요청

1. 형상서버에 계약서 저장
2. 주문팀에서 생성한 계약서 (productGet.groovy) 파일을 상품팀에 전달 (Pull Request)



상품팀이 계약서를 수락함으로써 계약 성립

- 상품팀 : Pull Request 수락
- 상품팀은 해당 계약서를 accept하여 반영함
- 상품 서비스는 이제부터는 계약서를 안지켰을때 아예 배포가 안됨



계약체결 후 계약 위반시, 배포도 실패 (상품팀)

Code Build 단계시, 계약위반으로 빌드 실패

빌드 정보

상태	빌드 실패
빌드 ID	6db60b82-3b99-4aef-9870-ce85dce23cccd
이미지	-
트리거	master 브랜치에 푸시 (products) GitHub event-storming/products ↗
소스	9997fc8cebfff7e444d94b790378bb58dfc324362c ↗
Git 커밋	CLOUDSDK_COMPUTE_ZONE=asia-northeast1-a CLOUDSDK_CONTAINER_CLUSTER=standard-cluster-1
환경 변수	-
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": 
```

상품팀은 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

Lab. Contract Test

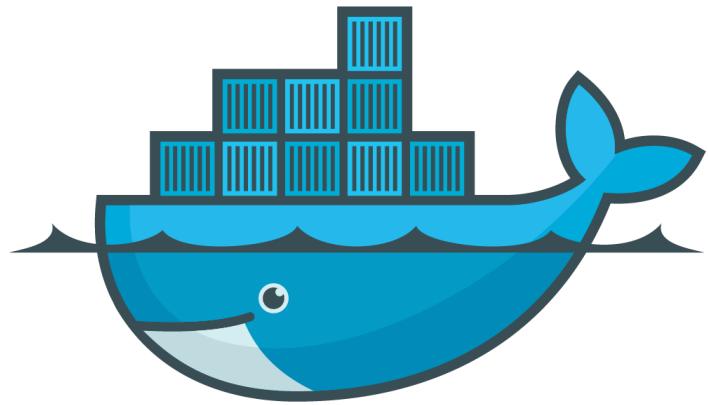


Lab Time

“

Packaging Application with Container

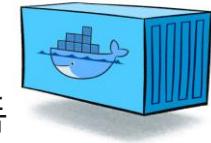
- Docker & Dockerizing
- Using Docker Hub as a Container Image Registry



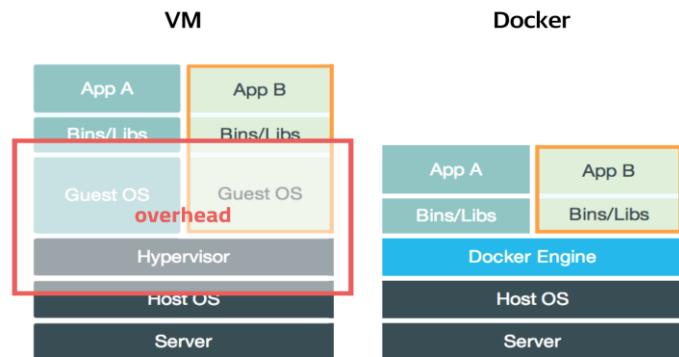
- 2013년 3월,
dotCloud 창업자 Solomon Hykes가 Pycon Conference에서 발표
- Go 언어로 작성된 “The future of linux Containers”

- Container based

- 프로세스 격리 기술
- 오픈소스 가상화 플랫폼



- 가상머신 .vs. Docker



Docker Image & Container

- Docker Image
 - 가상머신 생성시 사용하는 ISO 와 유사한 개념의 이미지
 - 여러 개의 층으로 된 바이너리 파일로 존재
 - 컨테이너 생성시 읽기 전용으로 사용
 - 도커 명령어로 레지스트리로부터 다운로드 가능



- 저장소 이름 : 이미지가 저장된 장소, 이름이 없으면 도커 허브(Docker Hub)로 인식
- 이미지 이름 : 이미지 이름, 생략 불가
- 이미지 버전 : 이미지 버전정보, 생략 시 latest로 인식

Docker Image Path Samples

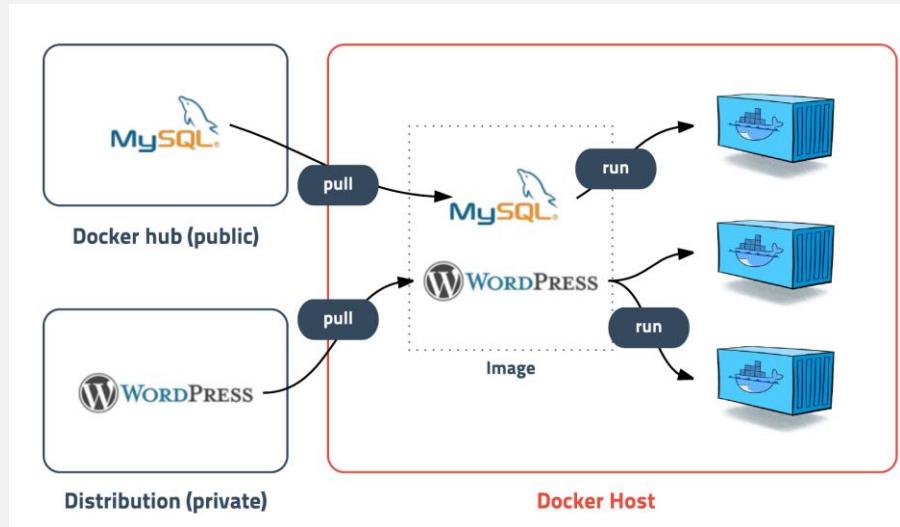


- 이미지 Path는 <URL>/<namespace>/<Image_name>:<tag> 형식
- library는 도커허브 공식 이미지 Namespace로, 여기에 사용자 이름이 위치
- Private 이미지 저장소(Docker, harbor)를 설치하여 운영 가능

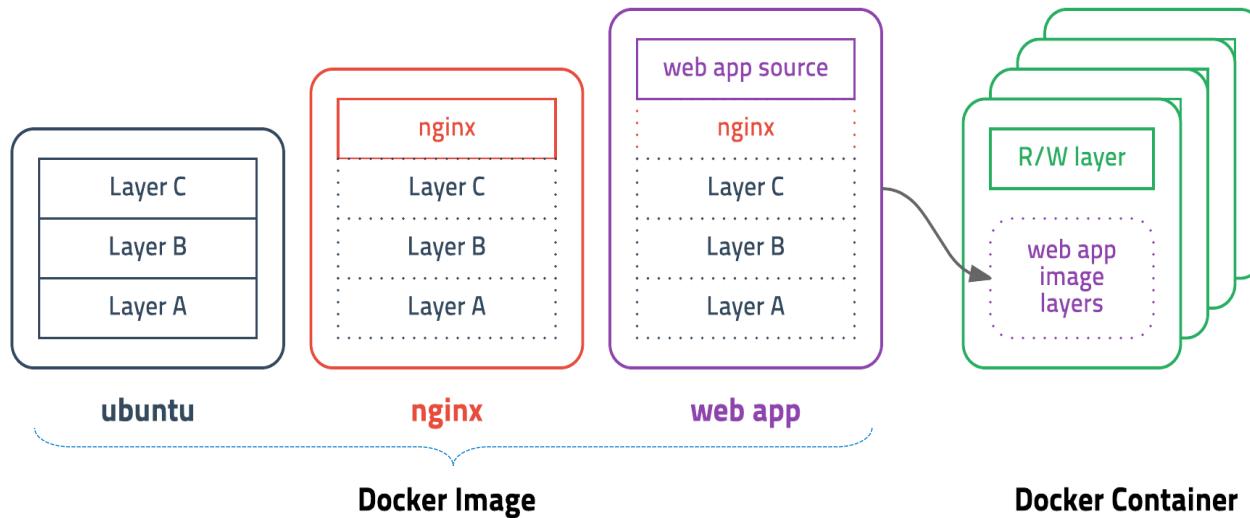
Docker Image & Container

- Docker Container

- 도커 이미지로 부터 생성
- 격리된 파일시스템, 시스템 자원, 네트워크를 사용할 수 있는 독립공간 생성
- 이미지를 읽기 전용으로 사용, 이미지 변경 데이터는 컨테이너 계층에 저장



Layered Architecture



- **Image** : 여러 개의 읽기 전용(Read Only) 레이어로 구성
- **Container** : Image 위에 R/W 레이어를 두고, 실행 중 생성 또는 변경 내용 저장

Image build Process

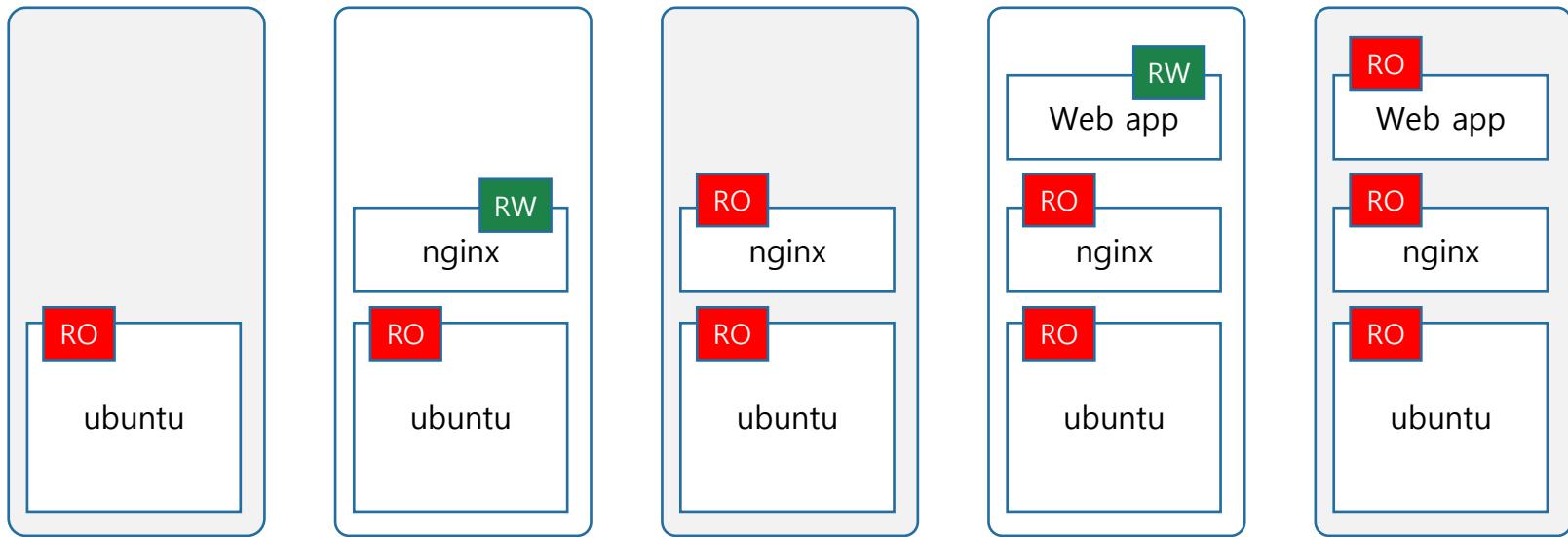


Image → Container → Image → Container → Image
Step 1/3 Step 2/3 Step 3/3

Dockerfile sample

```
FROM openjdk:8-jdk-alpine
```

```
RUN apk --no-cache add tzdata && cp  
/usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

```
WORKDIR /app  
COPY hello.jar hello.jar  
COPY entrypoint.sh run.sh  
RUN chmod 774 run.sh
```

```
ENV PROFILE=local
```

```
ENTRYPOINT ["./run.sh"]
```

- **FROM** : 이미지를 생성할 때 사용할 기반 이미지를 지정한다.
- **RUN** : 이미지를 생성할 때 실행할 코드 지정한다. 예제에서는 패키지를 설치하고 파일 권한을 변경하기 위해 RUN을 사용
- **WORKDIR** : 작업 디렉토리를 지정한다. 해당 디렉토리가 없으면 새로 생성한다. 작업 디렉토리를 지정하면 그 이후 명령어는 해당 디렉토리를 기준으로 동작
- **COPY** : 파일이나 폴더를 이미지에 복사한다. WORKDIR로 지정한 디렉토리를 기준으로 복사
- **ENV** : 이미지에서 사용할 환경 변수 값을 지정한다. 컨테이너를 생성할 때 PROFILE 환경 변수를 따로 지정하지 않으면 local을 기본 값으로 사용
- **ENTRYPOINT** : 컨테이너를 구동할 때 실행할 명령어를 지정한다.

Docker Image Commands

- 도커 이미지 목록 확인
 - \$ docker images
- 도커 이미지 불러오기
 - 컨테이너 run 시에 이미지가 없으면 Docker Hub로부터 자동으로 Pull
 - \$ docker pull [ImageName:태그]
- 도커 이미지 삭제
 - docker image rm [이미지 ID]
 - docker image rm -f [이미지 ID] # 컨테이너를 삭제하기 전에 이미지 삭제
- 도커 모든 이미지 한 번에 삭제
 - \$ docker image rm \$(docker images -q)

Docker Container Commands

- 컨테이너 실행
 - \$ docker run [Options] [Image] [Command]
- 실행 중인 컨테이너 확인
 - \$ docker ps
 - \$ docker ps -a # 정지된 컨테이너 포함
- 컨테이너 시작, 재시작, 종료
 - \$ docker start / restart / stop [컨테이너 이름]
- 컨테이너 삭제
 - \$ docker container rm [컨테이너 ID]
- 모든 컨테이너 한번에 삭제 (중지 후 삭제)
 - \$ docker container rm \$(docker ps -a -q)

Docker Build & Push Commands

- Dockerfile로 이미지 생성
 - docker build --tag [생성할 이미지 이름]:[태그 이름].
 - # 맨 마지막의 .(마침표)은 Dockerfile의 위치
 - 이미지 이름은 URL(Docker hub, Cloud Container registry, Private registry)로 시작
- 이미지 Push
 - \$ docker login # 도커 로그인
 - \$ docker push [이미지 REPOSITORY]:[태그]
- Docker Hub에서 확인
 - <http://hub.docker.com> (login)

Lab. Docker Build & Push

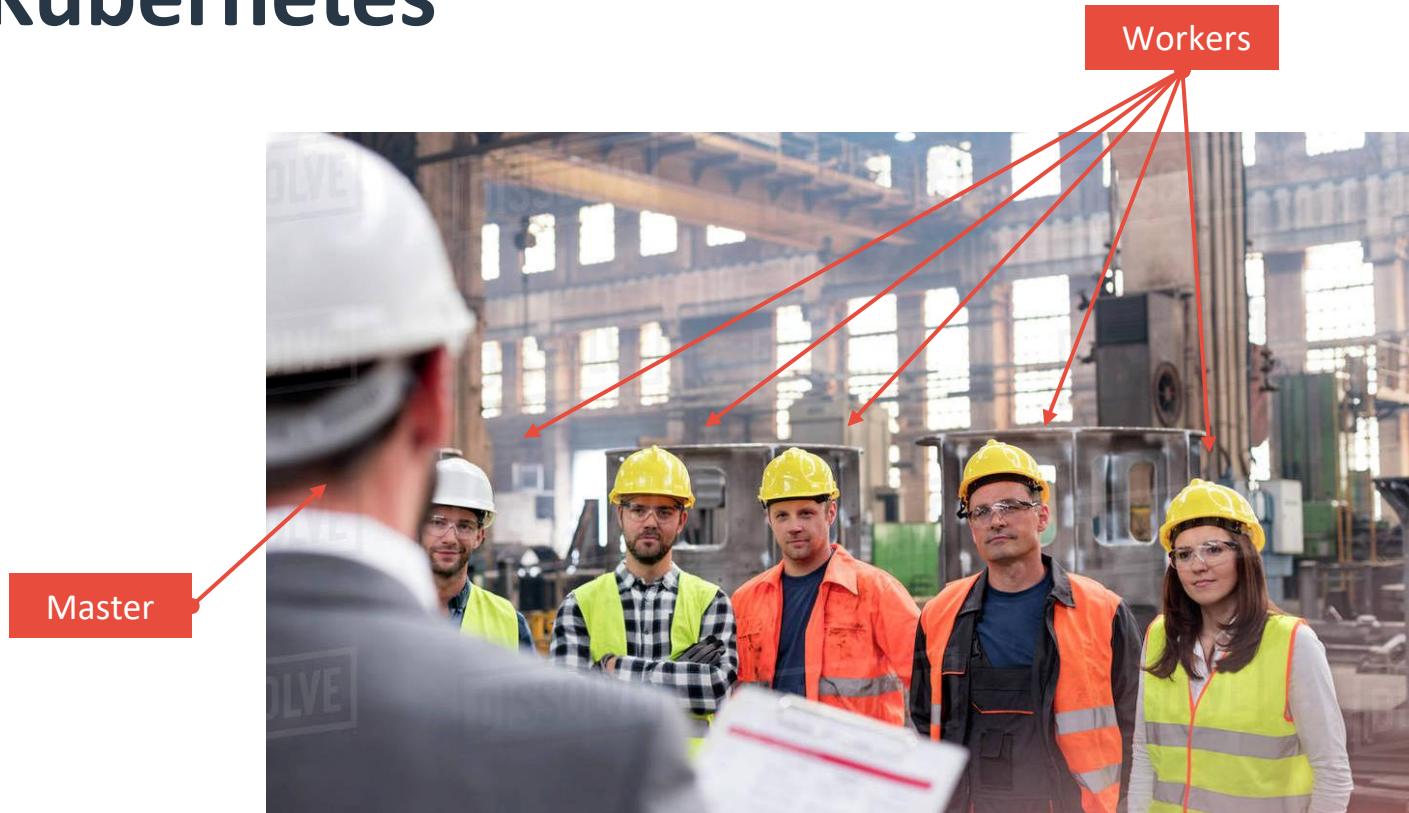


Lab Time

Table of contents

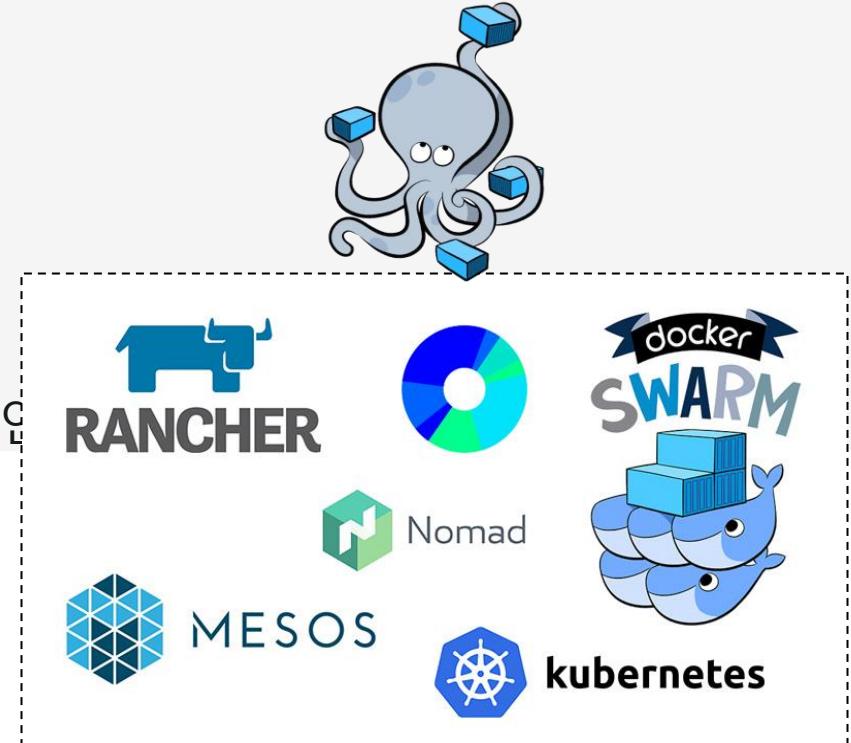
- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes) ✓
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh

Kubernetes



Container Orchestration Features

- 컨테이너 자동 배치 및 복제
- 컨테이너 그룹에 대한 로드 밸런싱
- 컨테이너 장애 복구
- 클러스터 외부에 서비스 노출
- 컨테이너 확장 및 축소
- 컨테이너 서비스간 인터페이스를 통한 C



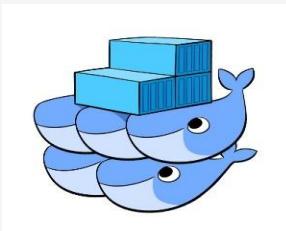
Container Orchestrators

Kubernetes



구글에서 개발, 가장 기능이 풍부하며 널리 사용되는 오케스트레이션 프레임워크
베어 메탈, VM환경, 퍼블릭 클라우드 등의 다양한 환경에서 작동하도록 설계
컨테이너의 롤링 업그레이드 지원

Docker Swarm



여러 개의 Docker 호스트를 함께 클러스터링하여 단일 가상 Docker 호스트를 생성
호스트 OS에 Agent만 설치하면 간단하게 작동하고 설정이 용이
Docker 명령어와 Compose를 그대로 사용 가능

Apache Mesos



수만 대의 물리적 시스템으로 확장할 수 있도록 설계
Hadoop, MPI, Hypertable, Spark 같은 응용 프로그램을 동적 클러스터 환경에서 리소스 공유와 분리를 통해 자원 최적화 가능
Docker 컨테이너를 적극적으로 지원

Kubernetes

“Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.”

“**Kubernetes**는 컨테이너화된 어플리케이션을 자동으로 배포, 조정, 관리할 수 있는 오픈소스 플랫폼이다.

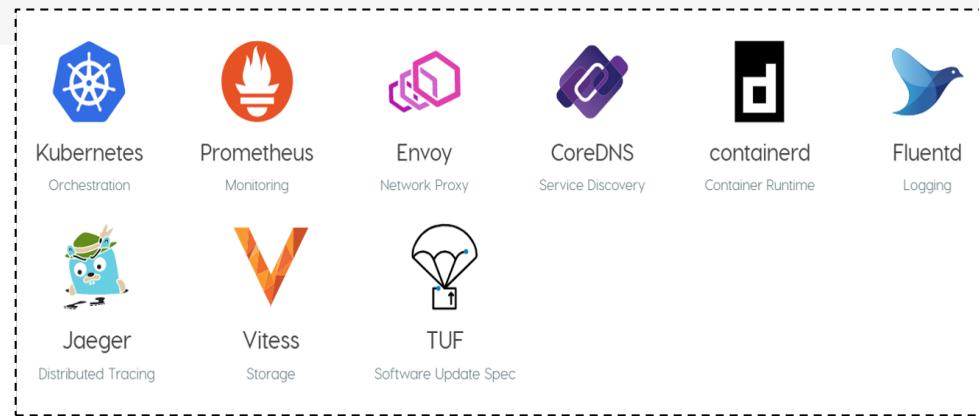
- From Kubernetes Website

Kubernetes Origin

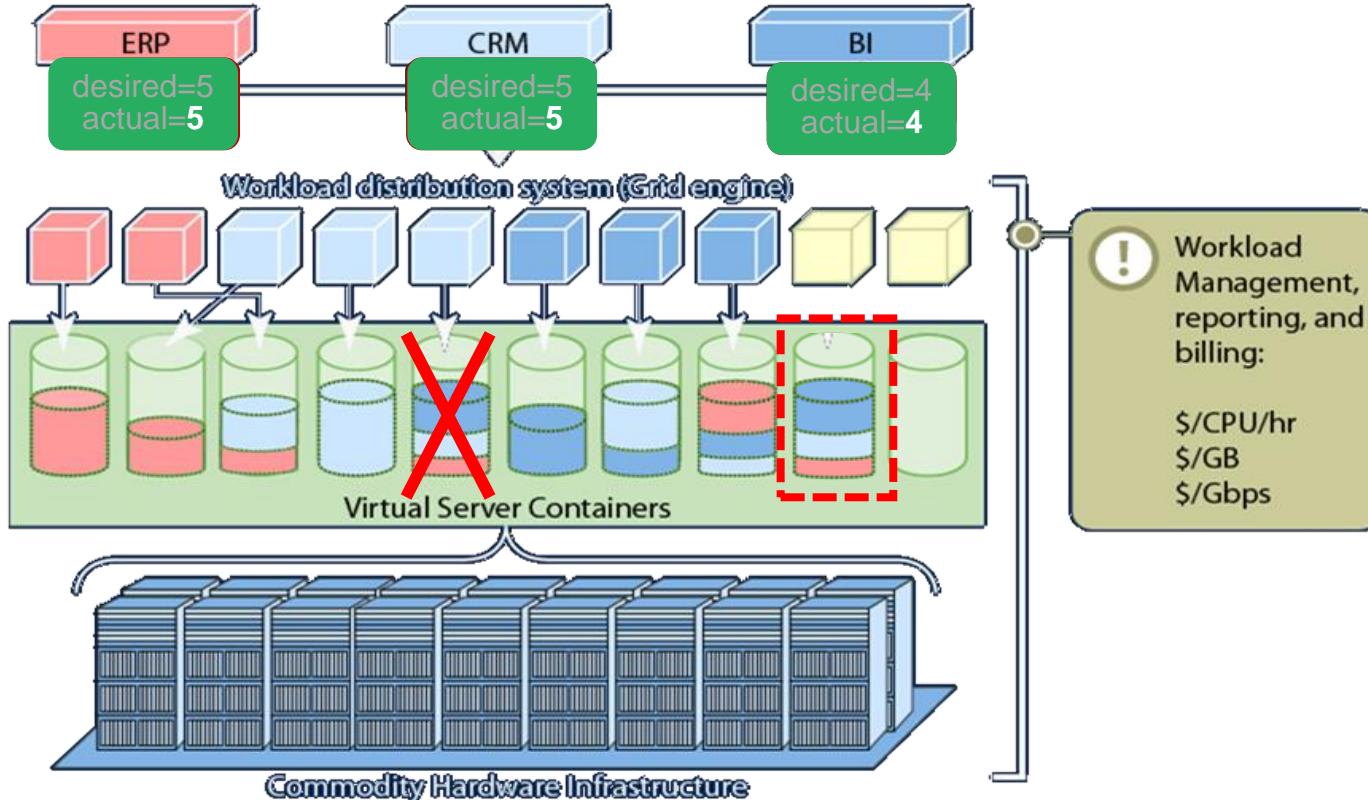
- Borg System 영향을 받아, 2014년 구글의 의해 처음 발표
- 2015년 7월 21일, v1.0 출시
- 리눅스재단과 Cloud Native Computing Foundation(CNCF) 설립
- Kubernetes를 seed 테크놀로지(seed technology)로 제공



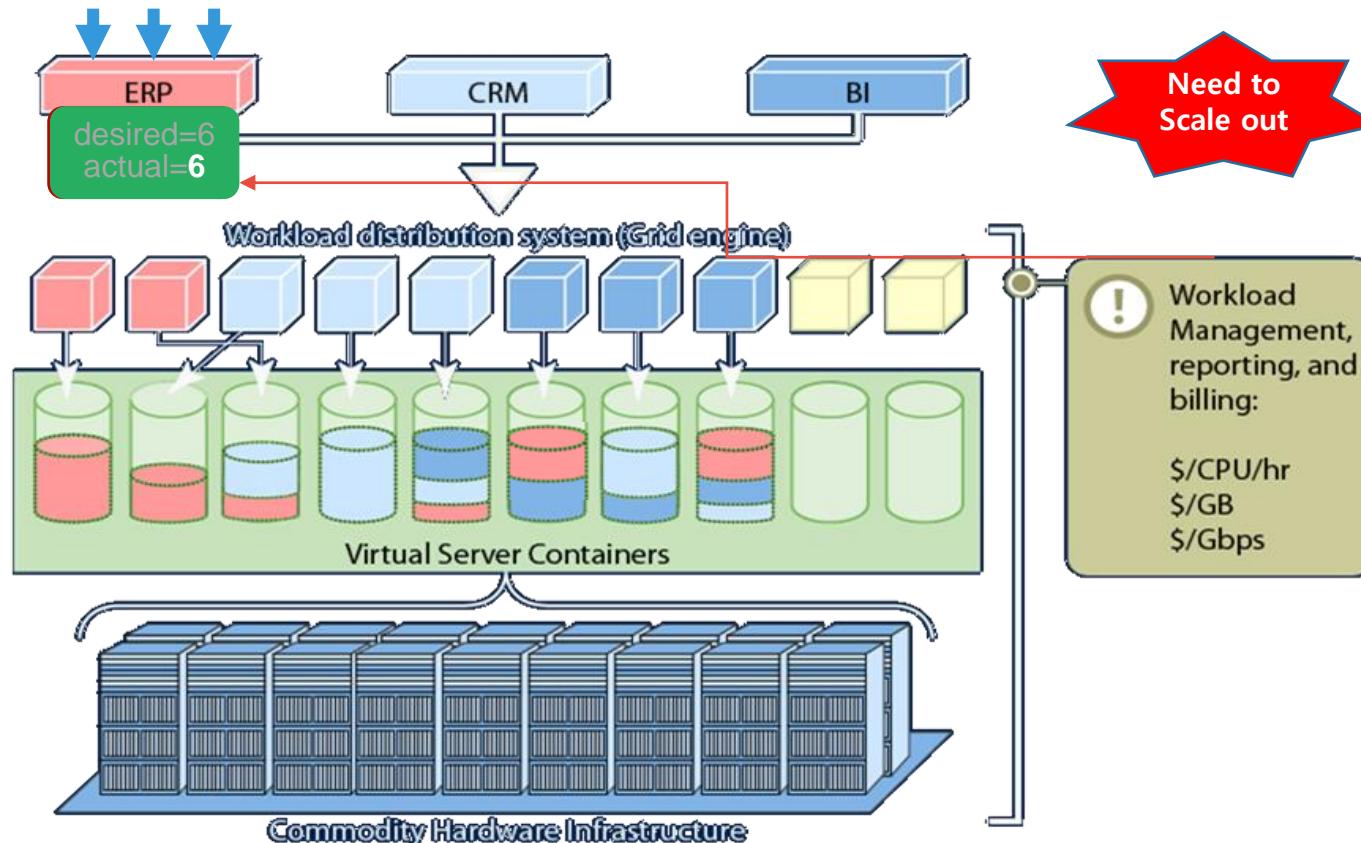
<https://www.cncf.io/>



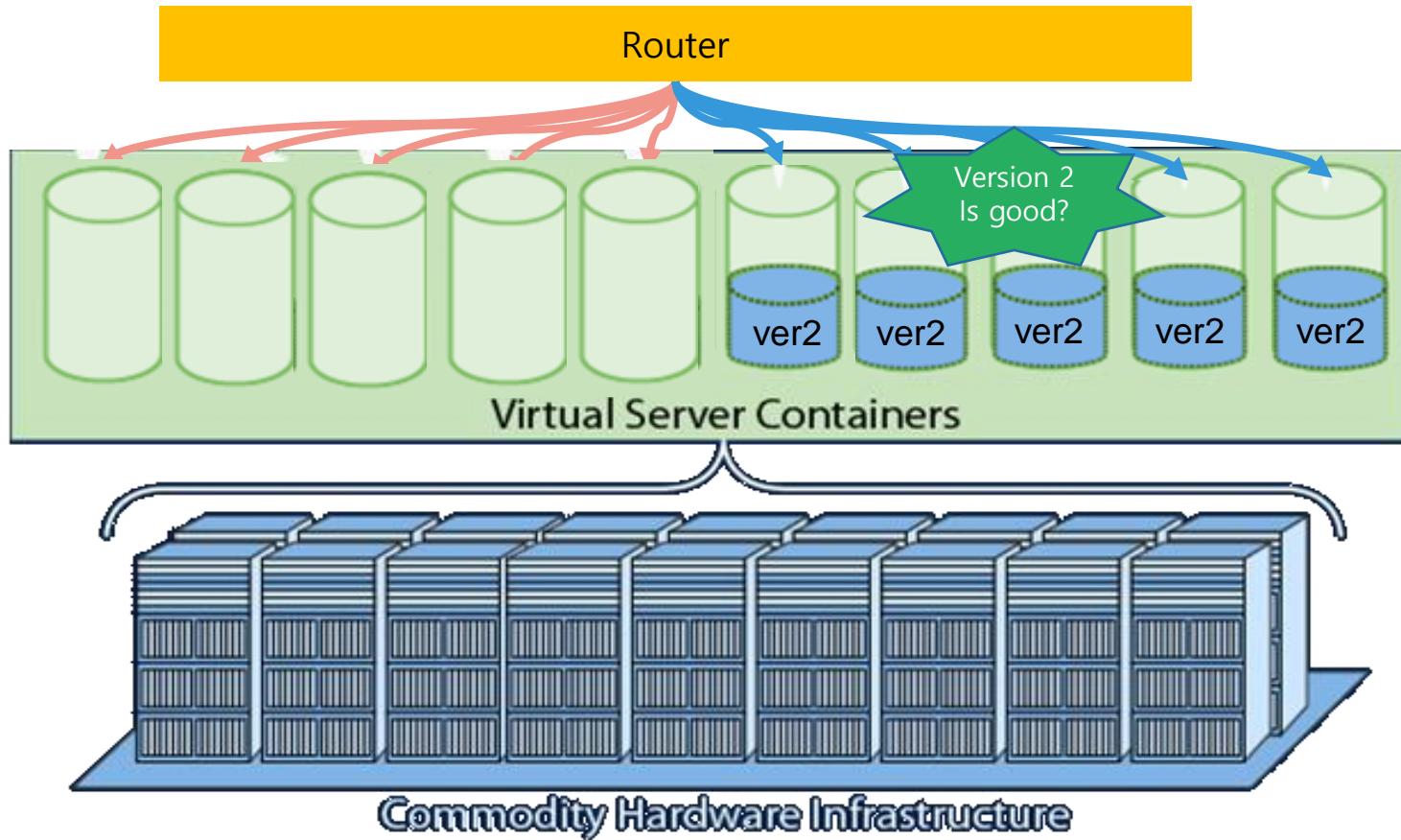
Container Orchestration – Self Healing Mechanism



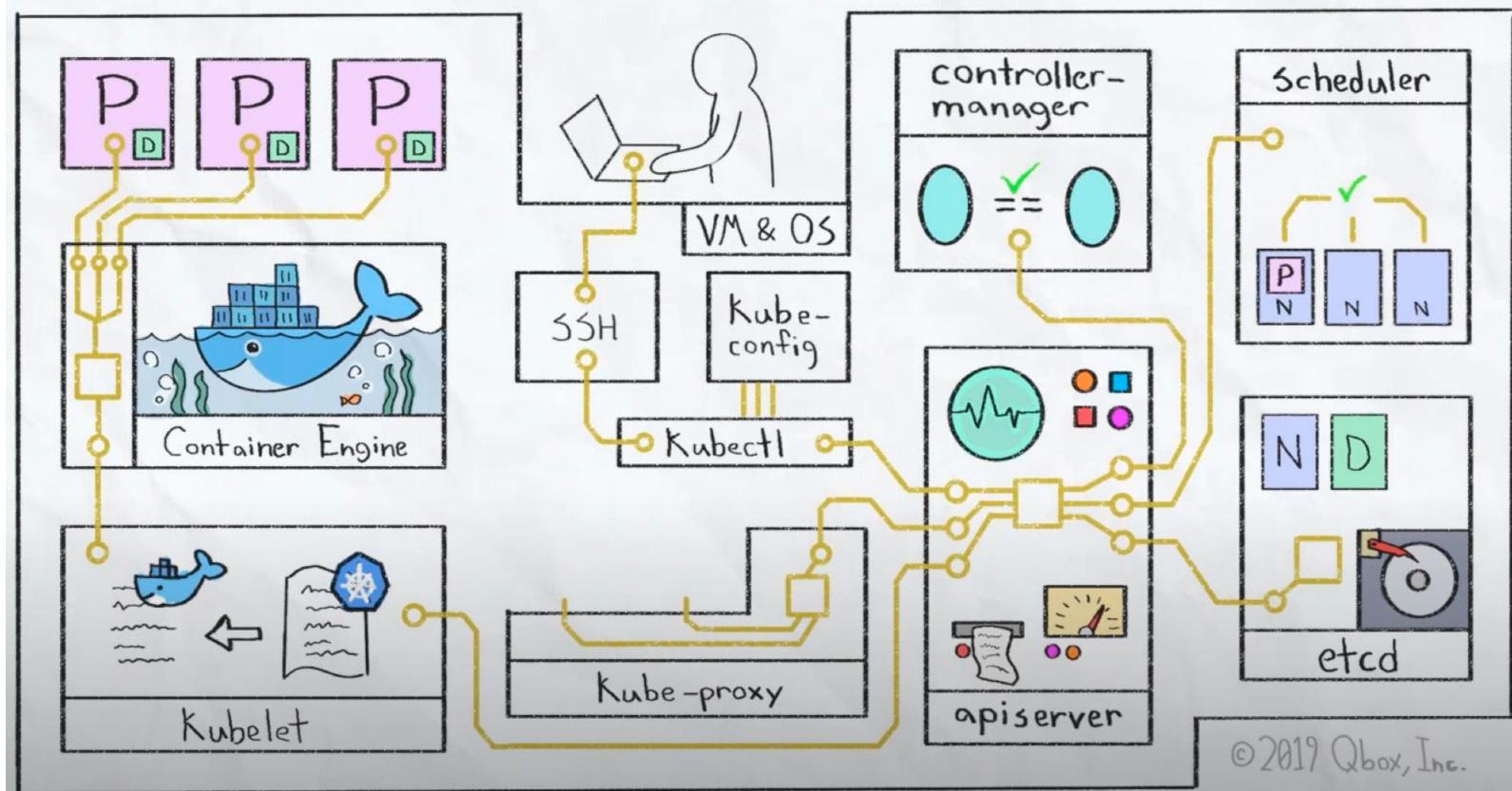
Container Orchestration – Scale out



Container Orchestrator – Zero Down Time Deploy



Anatomy of Kubernetes



Kubernetes Core Concept : “Meet the Desired State”

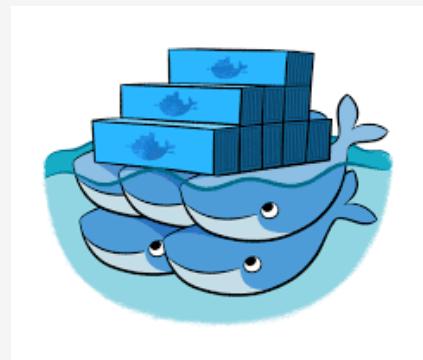


Declarative Model & Desired States

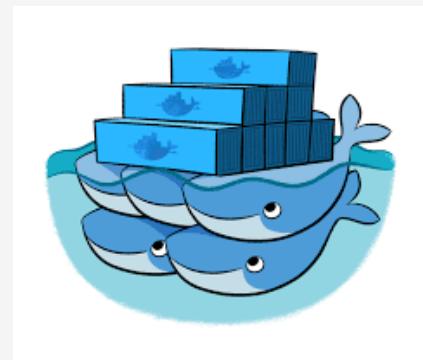
- Kubernetes는 Current State을 모니터링하면서, Desired State를 유지하려는 습성
- 직접적인 동작을 명령하지 않고, 원하는 상태를 선언(Not Imperative, But Declarative)
 - Imperative – “nginx 컨테이너를 3개 실행해줘, 그리고 80포트로 오픈해줘.”
 - Declarative – “80포트를 오픈한 채로, nginx 컨테이너를 3개 유지해줘.”

Pod ; Kubernetes 최소 배포 단위

- Pod : 미국식 [pa:d], 영국식 [pɒd]
 - “물고기, 고래” 작은 떼 (Docker의 심볼이 고래 모양에서 유래)
 - 발음하기 : “팟”, “파드”, “포드”



Pod



Pod

Kubernetes Object Model

`http://serviceld:8080`

Micro-Service

Service
(name: foo)

`http://external_ip:8080`

Load Balancer from
cloud provider

Redirects

Deployment

Creates / Manages

ReplicaSet
(Green)

ReplicaSet
(Blue)

- Zero-down time deployment
- (Kubernetes default is rolling-update)
e.g. Blue / Green

Responsible for :

- Dynamic Service Binding
- e.g. <http://foo:8080>
- Type :
 - LoadBalancer e.g. Ingress (API GW) or front-end
 - ClusterIP e.g. 내부 마이크로 서비스들

- Keep replica count as desired
(replicas=2)

- Service Hosting

Pod

Container
(Main)

Container
(Side-car)

Pod

Container

Container

Pod

Container

Container

Pod

Container

Container

Kubernetes Object Model

- **apiVersion** : 해당 Object description 을 해석할 수 있는 API server 의 버전
- **kind** : 오브젝트의 타입 – 예제는 **Deployment**
- **metadata** : 객체의 기본 정보. 예) 이름
- **Spec (spec and spec.template.spec)** : 원하는 "Desired State" 의 세부 내역. 예제에서는 3개의 replica를 template 내의 pod 정의대로 찍어내어 유지하라는 desired state 설정임
- **spec.template.spec** : defines the desired state of the Pod. The example Pod would be created using **nginx:1.7.9**.

Once the object is created, the Kubernetes system attaches the **status** field to the object

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Deployment object Example

Lab. 기본적인 kubectl 명령어

- 객체 목록 불러오기
 - “[kubectl get \[객체 타입\]](#)” Ex) kubectl get pods : pod 목록을 불러온다.
- 객체 삭제하기
 - “[kubectl delete \[객체 타입\] \[객체 이름\]](#)”
Ex) “kubectl delete pods wordpress-5bb5dddcff-kwgf8”
 - “[kubectl delete \[객체타입,객체타입,...\] --all](#)”
Ex) “kubectl delete services,deployments,pods --all”
 - 실습 중에 잘못 생성되었거나 초기화가 필요할 경우 사용
 - 콤마(,)로 구분하여 붙여 적기
- 객체 상세 설명 확인하기
 - “[kubectl describe \[객체 타입\] \[객체 이름\]](#)”
Ex) “kubectl describe service wordpress”

Lab. 이미지를 통한 어플리케이션 배포

- 현재 작동 중인 pod들이 없는지 확인
 - [kubectl get pods](#)
- Nginx 이미지 예제로 배포하기
 - [kubectl create deploy first-deployment --image=nginx](#)
- 실행된 Pods 확인
 - [kubectl get pods](#)
 - 각 Pod들은 ‘Pod명-{hash}’ 형식의 고유한 이름을 가짐

Lab. Pod에 접속하기

- 로그 보기
 - `kubectl logs [복사된 pod 이름] -f`
- 복사된 pod이름으로 접속하기
 - `kubectl exec -it [복사된 pod 이름] -- /bin/bash`
- Pod 내에서 접속하기 (위 명령어로 진입후에는 리눅스 Shell 명령어 사용)
 - Curl 명령어를 사용하기 위한 업데이트를 한다. (Curl 명령이 없으면 설치)
`apt-get update`
`apt-get install curl`
 - Curl 명령어로 호출하기
`curl localhost`

설정 파일을 통한 pod 배포 (1/2)

- 아래 내용으로 nano editor 를 이용하여 declarative-pod.yaml 파일을 생성
 - Nginx 이미지를 기반으로 pod를 배포하는 설정파일

```
apiVersion: v1
kind: Pod
metadata:
  name: declarative-pod
spec:
  containers:
    - name: memory-demo-ctr
      image: nginx
```

설정 파일을 통한 pod 배포 (2/2)

- nano declarative-pod.yaml 파일을 직접 작성 후 배포
 - `kubectl create -f declarative-pod.yaml`
(-f 는 파일 경로를 설정해서 배포할 수 있는 옵션이다.)
- 배포된 Pod를 검색 후 접속
 - 이름이 설정대로 declarative-pod 로 생성됨을 확인
`kubectl get pods`
 - 생성된 Pod에 접속
`kubectl exec -it declarative-pod -- /bin/bash`
`apt-get update`
`apt-get install curl`
`curl localhost`

Pod Initialization

- `Init.yaml` 파일 생성
 - `nano pod-initialize.yaml`
- Pod 생성
 - `kubectl create -f init.yaml`
- Pod 접속하기
 - `kubectl exec -it init-demo -- /bin/bash`
 - `cd /usr/share/nginx/html`
 - `cat index.html`

```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html
  initContainers:
    - name: install
      image: busybox
      command:
        - wget
        - "-O"
        - "/work-dir/index.html"
        - "http://kubernetes.io"
      volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
      dnsPolicy: Default
      volumes:
        - name: workdir
          emptyDir: {}
```

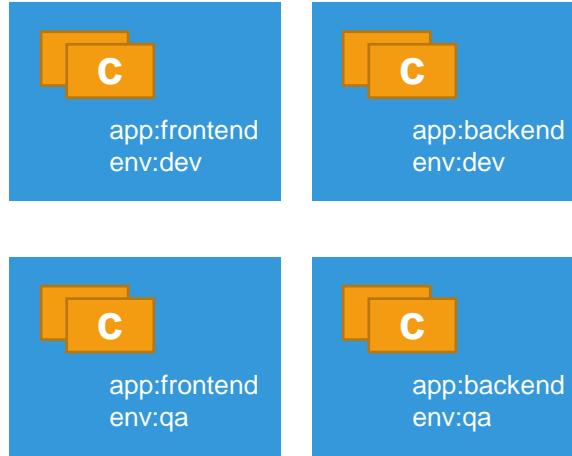
Pod 초기화 시점에
실행

Lab. Pod



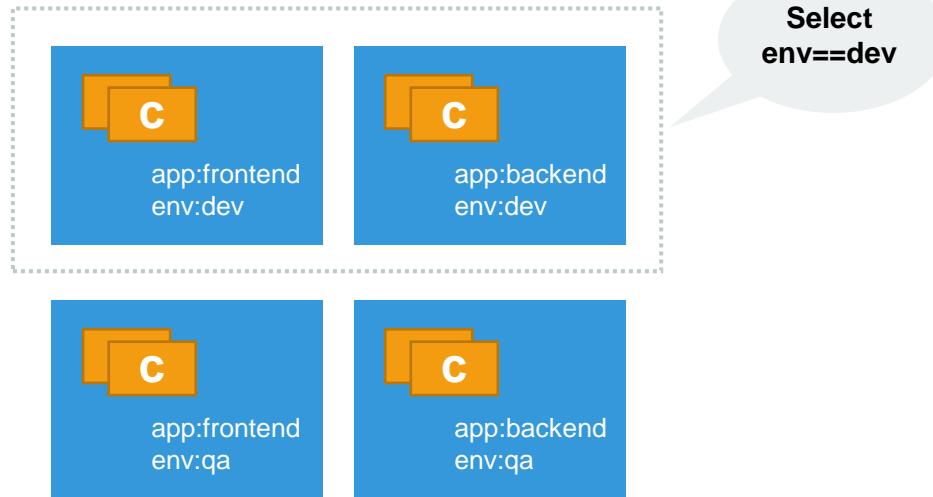
Lab Time

Labels



- Labels 은 객체 식별 정보로서 Kubernetes 객체라면 모두 붙일 수 있다.
- Label들은 요구사항에 맞춰 개체의 하위 집합을 구성하고 선택하는데 사용된다.
- Label들은 객체에 고유성을 제공하지 않아, 여러 객체들은 같은 label을 가질 수 있다.

Label Selectors



- Label Selectors들은 객체들의 집합을 선택하며, kubernetes는 2가지 종류를 지원한다.
 - **Equality-Based Selectors**
Uses the `=`, `==`, `!=` 연산자를 이용하여 Label key와 value 값을 기반으로 객체들을 필터링 할 수 있다.
 - **Set-Based Selectors**
`in`, `notin`, `exist` 연산자를 사용하며, value 값들을 기반으로 객체들을 선택할 수 있다.

Lab. Label

- kubectl get po # 실행 중, Pod list 확인
- kubectl edit po <pod 명> # Pod 인스턴스에 Label 추가

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-02-16T11:22:56Z"
  labels:
    env: test
  name: init-demo
  namespace: default
  resourceVersion: "588586"
```

] ← vi 에디터로 편집

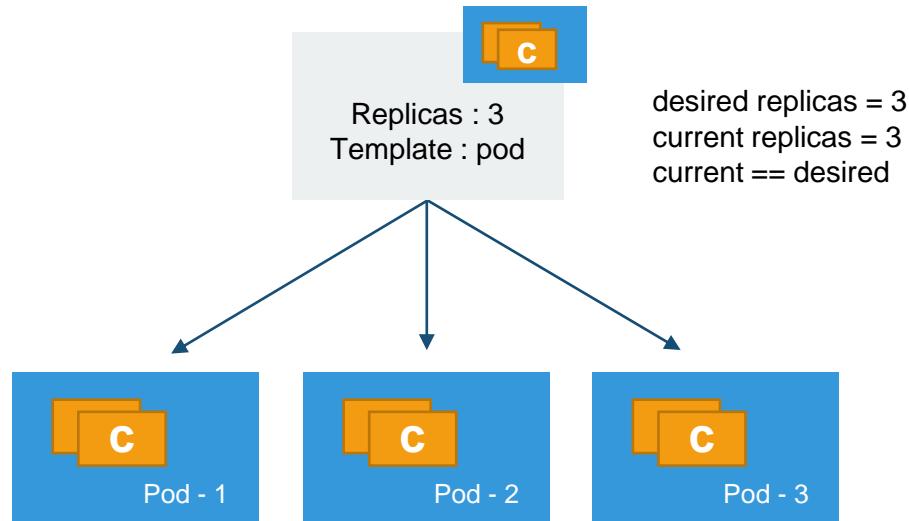
- kubectl get pods -l env=test
- kubectl get pods --selector env=test
- kubectl get pods --selector 'env in (test, test1)' # or 연산
- kubectl get po --selector 'env in(test, test1), app in (nginx, nginx1)' # and 연산
- kubectl get po --selector 'env,app notin(nginx)' # env가 있으면서, app이 nginx가 아닌 Pod

Replication Controllers

- Master node의 Controller Manager 중 하나
- Pod의 복제품이 주어진 개수(Desired State)만큼 작동하고 있는지 확인하고 개수를 조절 한다.
- Replication Controller는 Pod를 생성하고 관리한다.
 - 일반적으로 Pod는 자기 복구가 불가능 하기에 단독으로 배포를 하지 않는다.

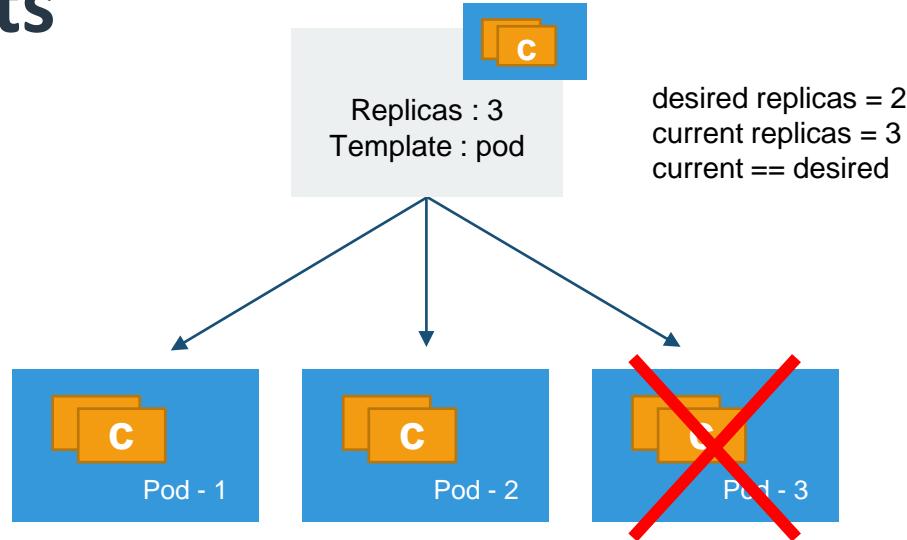


ReplicaSets



- ReplicaSet(rs)은 Replication Controller의 업그레이드 버전
- ReplicaSet은 equal 및 set 기반 Selector를 모두 지원하는 반면, Replication Controller는 equal기반 Selector만 지원

ReplicaSets



- 지정된 수의 Pod (Desired State)가 항상 실행되도록 보장
- ReplicaSets은 단독으로도 사용 가능하지만, 주로 Pod Orchestration에 사용(Pod creation, deletion, updates)
- Deployment가 ReplicaSets을 자동 생성하기 때문에 사용자는 관리에 신경 쓰지 않아도 됨

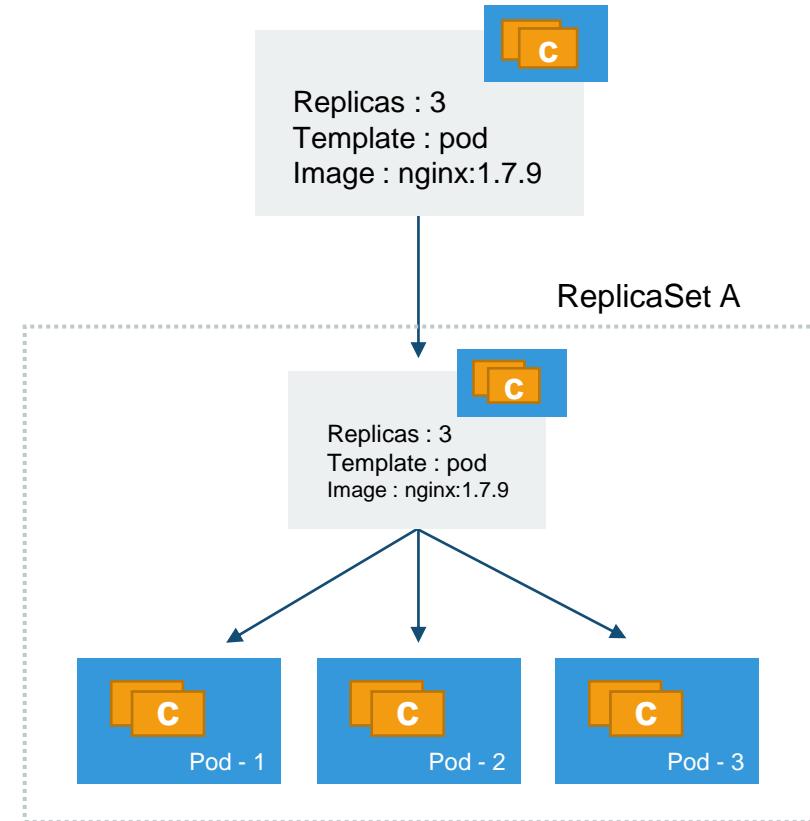
Lab. ReplicaSet

- ReplicaSet 파일 생성
 - [nano frontend.yaml](#)
- 파일을 기반으로 ReplicaSet 배포
 - [kubectl create -f frontend.yaml](#)
- ReplicaSet을 확인
 - [kubectl get pods](#)
 - [kubectl describe rs/frontend](#)
- ReplicaSet을 삭제
 - [kubectl delete rs/frontend](#)
(관련된 pod 전부 제거한다.)
 - [kubectl delete rs/frontend --cascade=false](#)
(ReplicaSet은 제거하지만 Pod는 유지)

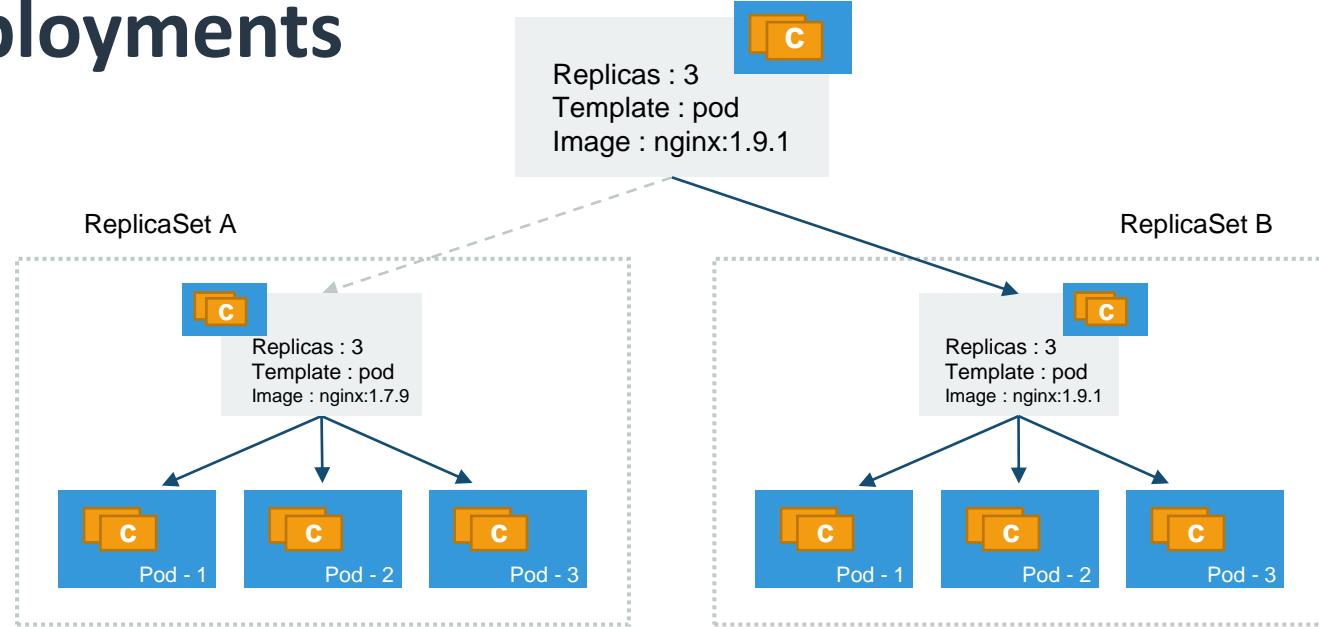
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          ports:
            - containerPort: 80
```

Deployments

- Deployment 객체는 Pods와 ReplicaSets에 대한 선언적 업데이트를 제공한다.
- Deployment Controller는 Master node 컨트롤 관리자의 일부로 Desired state가 항상 만족이 되는지 확인한다.
- Deployment 가 ReplicaSet을 만들고 ReplicaSet은 그 뒤에 주어진 조건만큼의 Pod들을 생성한다.



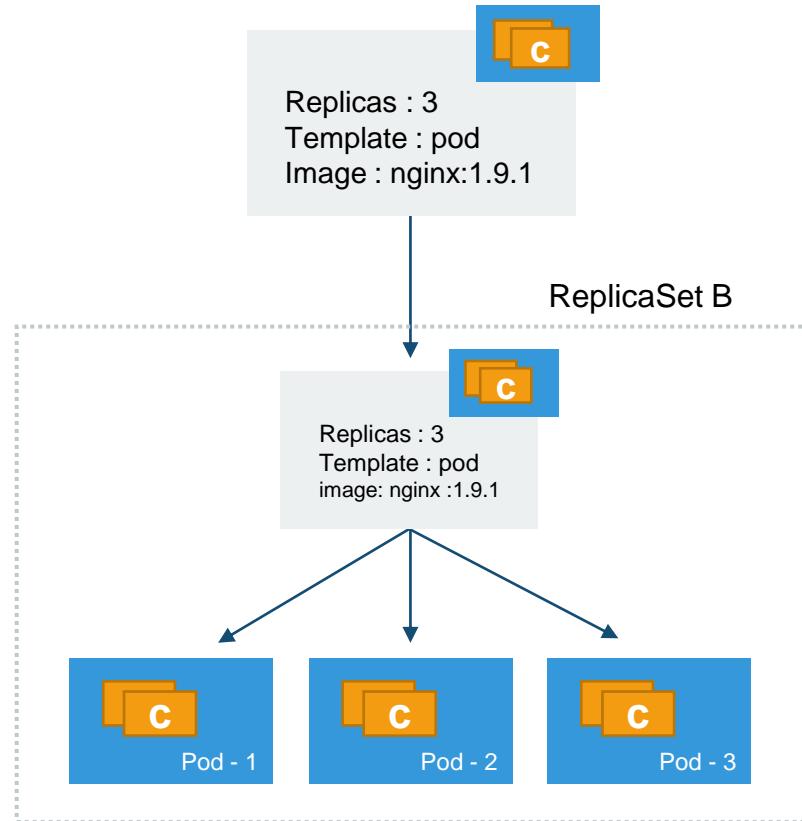
Deployments



- Deployment의 Pod template이 바뀌게 되면, 새로운 ReplicaSet이 생성되는데, 이를 **Deployment rollout**이라고 한다.
- Rollout은 Pod template에 변동이 생겼을 경우에만 동작하며, Scaling등의 작업은 ReplicaSet을 새로 생성하지 않는다.

Deployments

- 새로운 ReplicaSet이 준비되면 Deployment는 새로운 ReplicaSet을 바라본다.
- Deployment들은 Deployment recording등의 rollback 기능을 제공하며, 문제가 발생했을 경우, 이전 단계로 돌릴 수 있다.



Deployment 생성

- 설정 파일을 생성
 - nano nginx-deployment.yaml
- 파일을 기반을 배포
 - kubectl create -f nginx-deployment.yaml
- 생성된 deployment 확인
 - kubectl describe deployment nginx-deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Pod 생성 시, 이 템플릿 참조
도커허브에서 nginx 1.7.9
이미지를 가져와 'nginx' 이름의
컨테이너 생성

Scaling Deployments

- Deployment의 replica의 개수를 확인한다.
 - `kubectl get pods`
- Deployment의 이름을 복사한다.
 - `kubectl get deployments`
- 해당 Deployment의 scale 조정
 - `kubectl scale deployments [deployment 이름] --replicas=3`
- 변경을 확인
 - `kubectl get pods`

Deployments 의 변경

- Deployment 파일을 변경
 - kubectl get deployments
 - nano nginx-deployment.yaml
(spec 아래 항목에 replicas: 5 속성 추가; 있으면 수정)
- 변경한 파일을 적용
 - kubectl apply -f nginx-deployment.yaml
- 변경 내용을 확인
 - kubectl get pods
- 설정파일에 추가된 replicas 속성을 삭제 후 다시 적용
 - Nano nginx-deployment.yaml (replicas)
 - kubectl apply -f nginx-deployment.yaml
 - kubectl delete pods --all
 - kubectl get pods

Rolling update

- 작동중인 pod와 deployments 확인
 - `kubectl get pods`
 - `kubectl get deployments`
- 새로운 버전의 deployments 배포 및 배포 상태 확인
 - `kubectl apply -f nginx-deployment.yaml`
 - `kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1`
 - `kubectl rollout status deployment/nginx-deployment`
- 변경 확인
 - `kubectl get deployments`
 - `kubectl get pods`
 - `kubectl describe pods [pod 이름]`
- Pod에 접속하여 확인
 - `kubectl exec -it podname -- /bin/bash`
 - `apt-get update`
 - `apt-get install curl`
 - `curl localhost`

Rollback

- 현재 실행중인 객체들을 확인
 - `kubectl get pods`
 - `kubectl get deployments -o wide` # deployment에 적용된 Image:버전 추가 표시
- 객체를 롤백 처리
 - `kubectl rollout undo deployment/nginx-deployment`
- 진행을 확인
 - `kubectl get deployments -o wide`

Namespaces

- Kubernetes는 동일 물리 클러스터를 기반으로 하는 복수의 가상 클러스터를 지원하는데 이들 가상 클러스터를 Namespace라고 한다.
- Namespace를 활용하면, 팀이나 프로젝트 단위로 클러스터 파티션을 나눌 수 있다.
- Namespace 내에 생성된 자원/객체는 고유하나, Namespace 사이에는 중복이 발생할 수 있다.



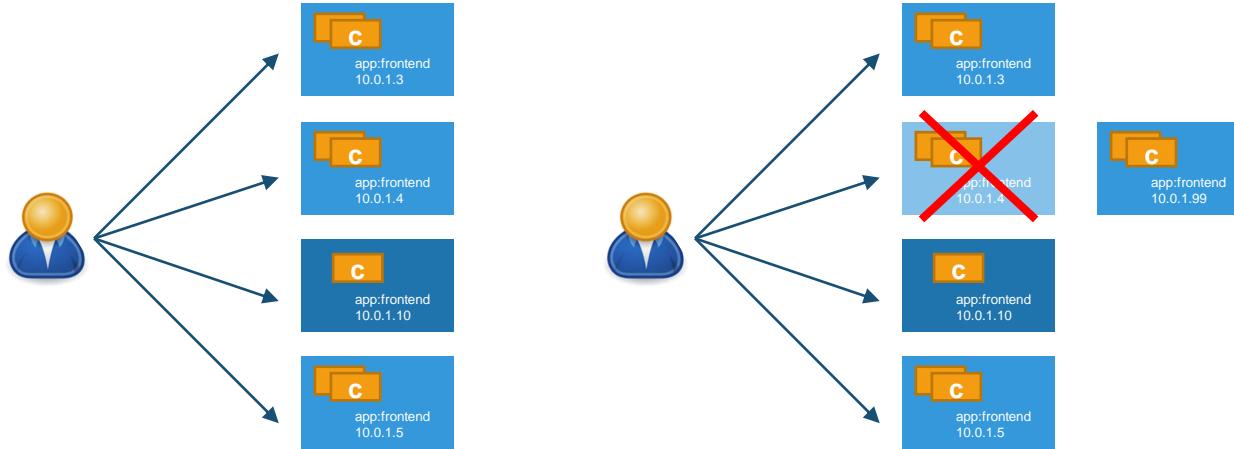
Namespaces

- Namespaces Object 조회

```
$ kubectl get namespaces
  NAME      STATUS   AGE
  default   Active   11h
  kube-public   Active   11h
  kube-system   Active   11h
```

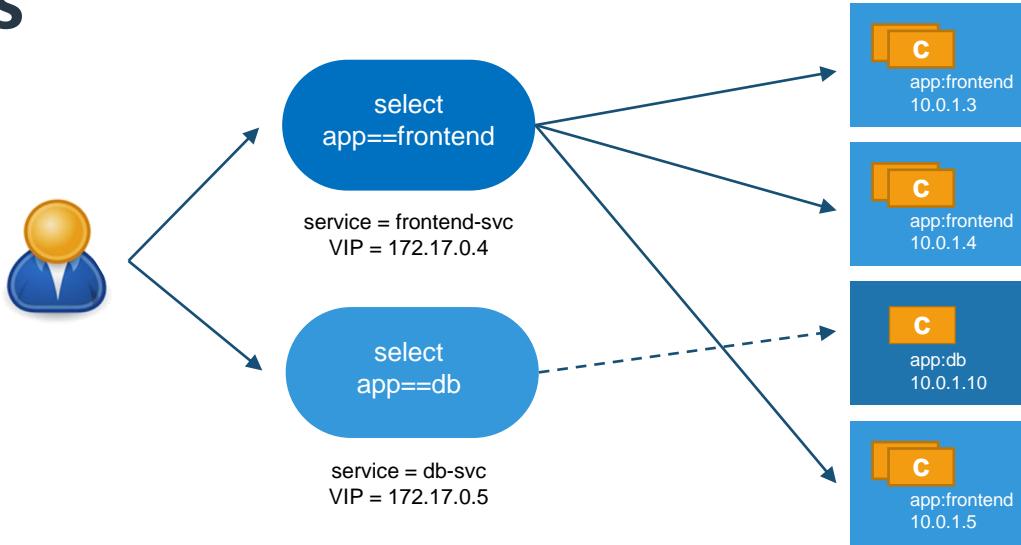
- Kubernetes는 처음에 3개의 초기 네임스페이스를 가진다.
 - default** : 다른 namespace를 갖는 다른 객체들을 가지고 있다.
 - kube-public** 은 클러스터 bootstrapping 같은 모든 유저가 사용 가능한 특별한 namespace 이다.
 - kube-system**: Kubernetes system에 의해서 생성된 객체를 가지고 있다.
- Resource Quotas를 사용하여 namespace 내에 존재하는 자원들을 나눌 수 있다.

Pod Access Issues



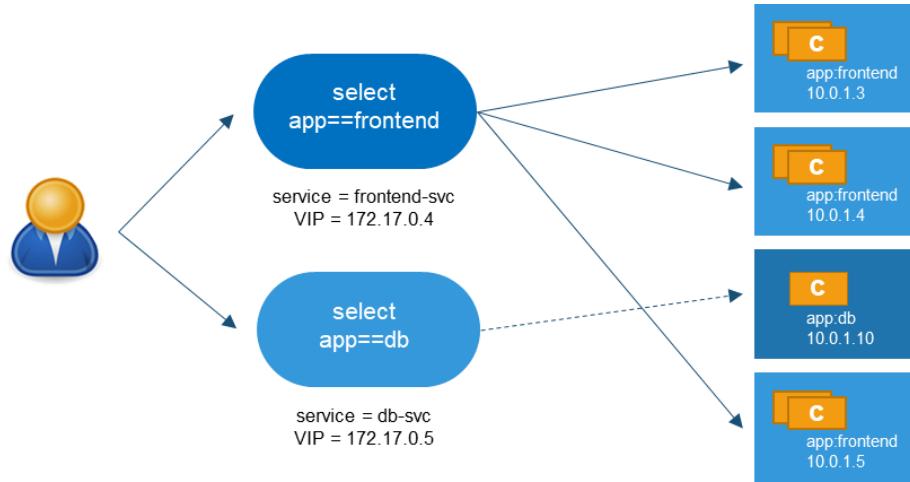
- 어플리케이션에 접근하기 위해서는, 사용자가 Pod에 접근해야 한다.
- Pod들은 언제든지 소멸 가능하기에 IP주소는 고정되어 있지 않다.
- 사용자가 직접 IP주소로 Pod에 연결되어 있을 때, Pod가 죽어서 새로 만들어지면 접속할 방법이 없다.
- 이 상황을 극복하기 위해서 추상화를 통해 Service라는 Pod들의 논리적 집단을 만들어 규칙을 설정하고 사용자들은 여기에 접속을 한다.

Services



- Selector를 사용하여 Pod를 논리적 그룹으로 나눌 수 있다.
- 각 논리적 그룹에 대해서 Service name이라는 이름을 부여할 수 있다.
- 사용자는 Service IP주소를 통해 Pod에 접속하게 된다.
 - 각 Service에 부여된 IP는 클러스터 IP 라고도 부른다.
- Service는 각 Pod에 대해 Load balancing을 자동으로 수행

Service YAML Specification

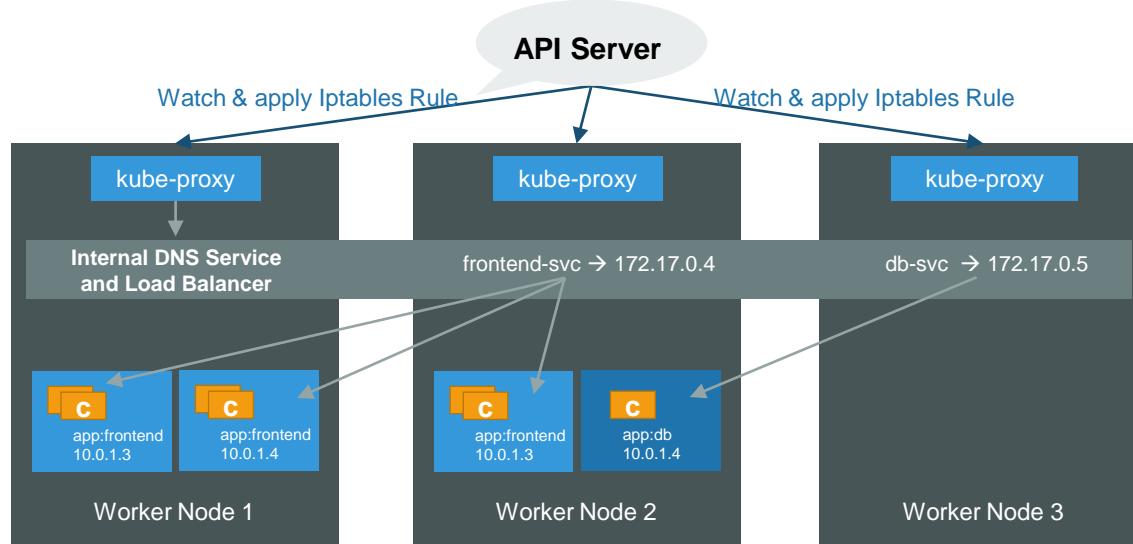


- Selector를 사용하여 Pod를 논리적 그룹으로 그룹핑
- 각 논리적 그룹에 대해서 Service Name 부여
- 사용자는 Service IP주소를 통해 Pod에 접속
 - 각 Service에 부여된 IP는 클러스터 IP 라고도 함
- Service는 각 Pod에 대해 Load balancing을 자동으로 수행

```
apiVersion: v1
kind: Service
metadata:
  name: db-svc
  labels:
    run: db-svc
spec:
  ports:
    - port: 3306
      targetPort: 3306
      protocol: TCP
      name: http
    - port: 63306
      targetPort: 63306
      protocol: TCP
      name: https
  selector:
    app: db
  type: ClusterIP
```

Container Spec.에 설정된 Label

Service Daemon : kube-proxy



- 모든 Worker node들은 Kube-proxy라는 데몬을 실행하는데, 이 데몬은 Service 및 End-point 객체의 추가/삭제를 위해 마스터 노드의 API 서버를 모니터링
- 각 Worker Node의 kube-proxy는 새로운 Service에 대한 Iptables 규칙 설정은 물론, 서비스 ClusterIP의 트래픽을 캡처하고 해당 트래픽을 서비스의 백엔드 Pod 중 하나로 리다이렉트
- Service 객체가 제거되면, kube-proxy는 Worker Node상의 Iptables 규칙도 삭제

Service Discovery

- Service는 클라이언트가 애플리케이션에 접근하는 Kubernetes의 채널로 런타임시, 이를 검색할 수 있는 방법이 필요
- DNS를 이용하는 방법
 - 서비스는 생성되면, [서비스 명].[네임스페이스명].svc.cluster.local이라는 DNS명으로 쿠버네티스 내부 DNS에 등록되고, 쿠버네티스 내부 클러스터에서는 이 DNS명으로 접근 가능한데, 이 때 DNS에서 리턴해 주는 IP는 외부 IP(External IP)가 아니라 Cluster IP(내부 IP) 임
- External IP를 명시적으로 지정하는 방법
 - 외부 IP는 Service의 Spec 부분에서 externalIPs 항목의 Value로 기술

ServiceType

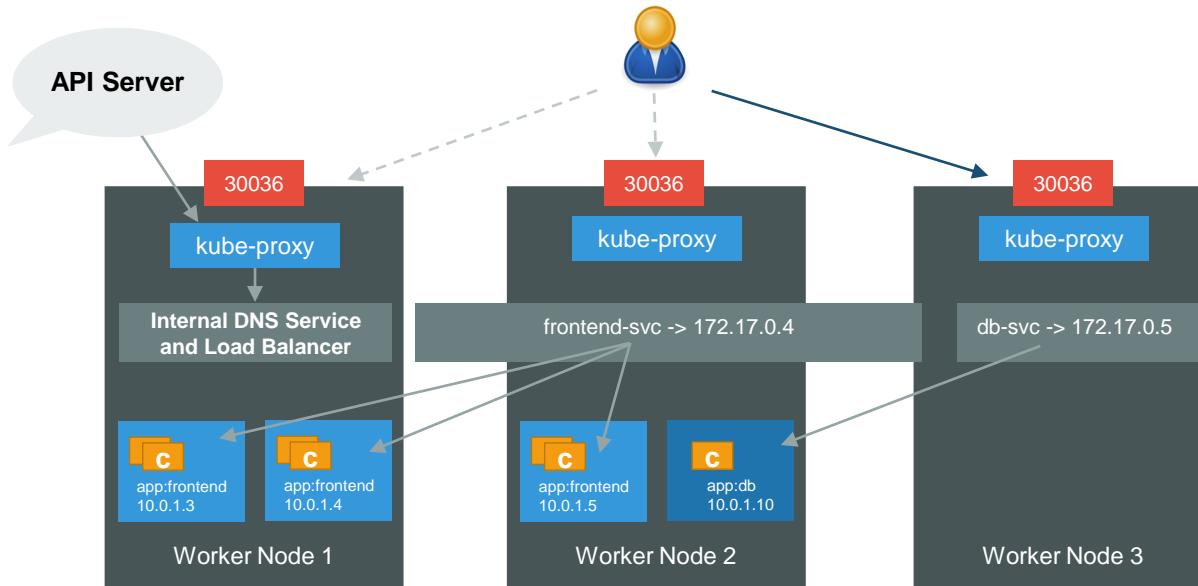
- Service를 정의할 때 Access Scope를 따로 정할 수 있다.
 - 클러스터 내에서만 접근이 가능한가?
 - 클러스터 내와 외부에서 접근이 가능한가?
 - 클러스터 밖의 리소스에 대한 Map을 가지는가?
- Service 생성 시, IP주소 할당 방식과 서비스 연계 등에 따라 4가지로 구분
 - ClusterIP
 - NodePort
 - LoadBalancer
 - ExternalName

ServiceType : ClusterIP

- 디폴트 설정으로 서비스에 클러스터 ip를 할당
- 쿠버네티스 클러스터 내에서만 이 서비스에 접근 가능
- 외부에서는 외부 IP를 할당 받지 못했기 때문에 접근이 불가능

ServiceType : NodePort

- 고정 포트(NodePort)로 각 노드의 IP에 서비스를 노출
- Cluster IP 뿐만 아니라, 노드의 IP와 포트를 통해서도(<NodeIP>:<NodePort>) 접근 가능

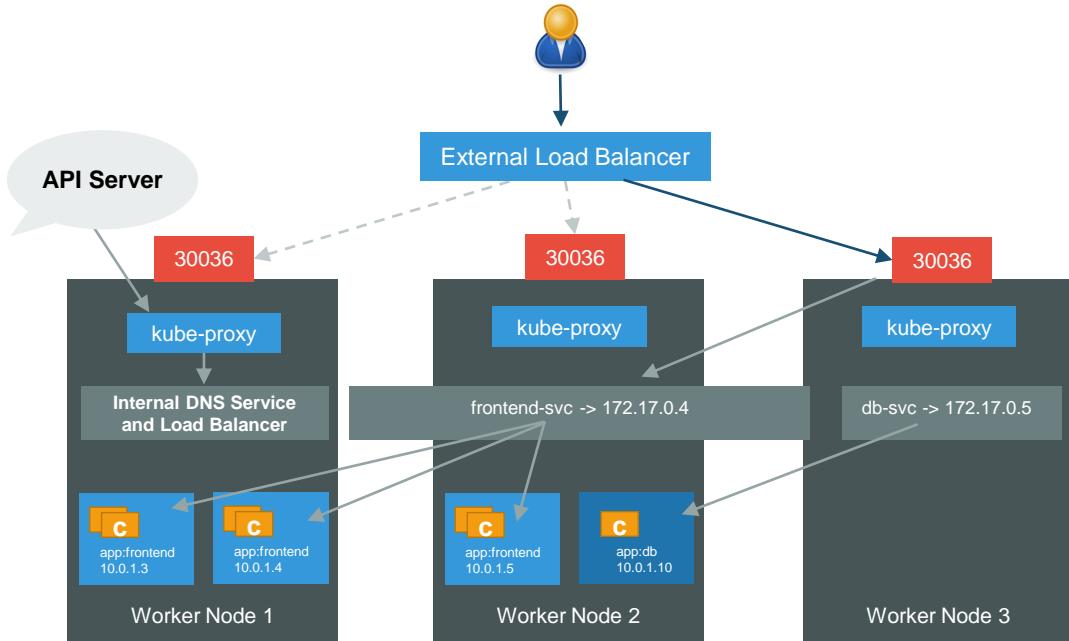


ServiceType : LoadBalancer

- 클라우드 밴더의 로드밸런싱 기능을 사용
 - NodePort와 ClusterIP Service들은 자동으로 생성되어 External Load Balancer가 해당 포트로 라우팅
 - Service들은 각 Worker node에서 Static port로 노출

LoadBalancer ServiceType은 기본 인프라가 Load balancer의 자동 생성을 제공하고, Kubernetes를 지원 할 경우에만 작동

Ex) Azure, Google Cloud Platform, AWS



Ingress

- Service는 L4 레이어로 TCP레벨에서 Pod를 로드밸런싱 함
- MSA에서는 Service 하나가 MSA의 서비스로 표현되는 경우가 많고 서비스는 하나의 URL(/orders, /products, ...)로 대표되는 경우가 많다.
- MSA 서비스간 라우팅을 위해 API Gateway를 두는 경우가 많은데 관리포인트가 생김
- URL기반의 라우팅 정도라면 L7 로드밸런서 정도로 위의 기능을 충족
- Kubernetes에서 제공하는 L7 로드밸런싱 컴포넌트를 'Ingress' 라고 함

kubernetes.io의 의하면,

"An Ingress is a collection of rules that allow inbound connections to reach the cluster Services."

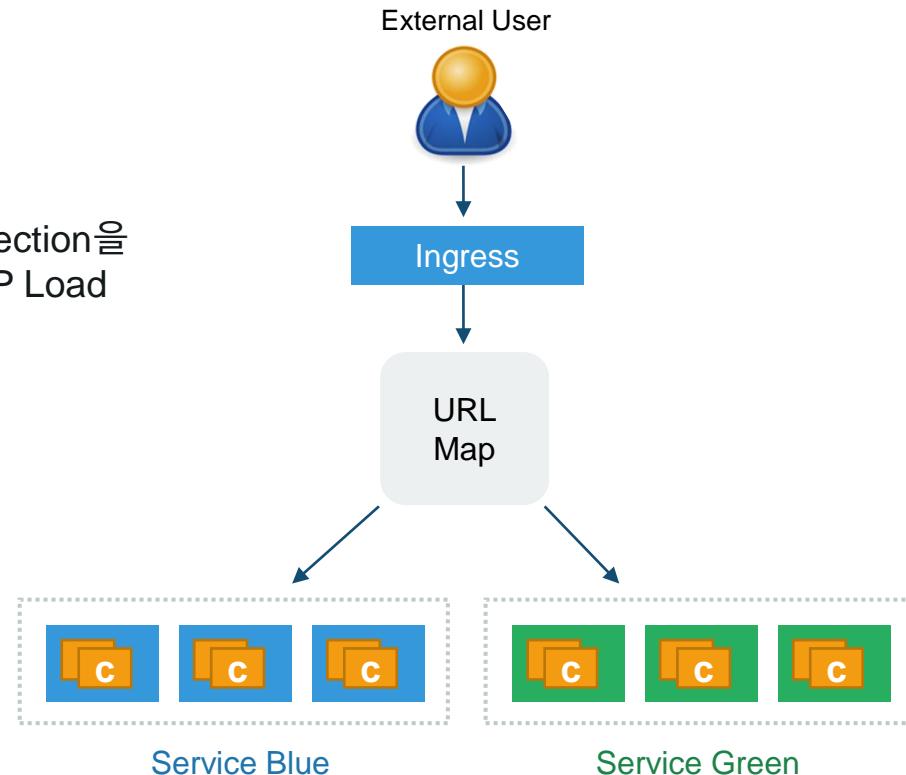
"Ingress는 인바운드 연결이 클러스터의 Service에게 라우팅되도록 하는 규칙의 집합체이다."



Ingress

- 아래와 같은 Service들의 Inbound Connection을 지원하기 위해 Ingress는 Layer7의 HTTP Load balancer 기능 제공

- TLS (SSL)
- Name-based virtual hosting
- Path-based routing
- Custom rules

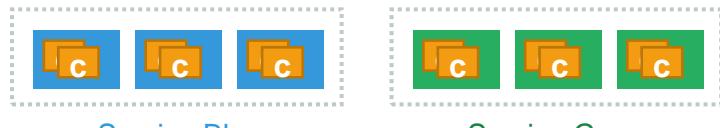


Ingress

URL Path based

example.com/blue
example.com/green

Ingress Controller



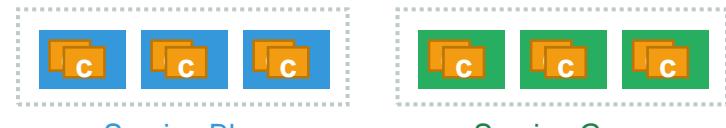
Service Blue

Service Green

Virtual Hosting

blue.example.com
green.example.com

Ingress Controller



Service Blue

Service Green

- 사용자들은 직접 Service에 접속하지 않는다.
- 유저는 Ingress에 먼저 접근하고, 요청은 해당 Service로 포워드 된다.
- Ingress 요청은 Ingress Controller에 의해 처리된다.

Ingress Controller

- "Ingress Controller"는 Ingress 리소스의 변경 사항을 마스터 노드의 API 서버에서 감시하고, 그에 따라 Layer 7로드 밸런서를 업데이트하는 응용 프로그램이다.
- Ingress Controller는 오픈소스 기반 구현체 및 클라우드 벤더사가 직접 구현체를 개발해 사용하기도 함
 - Nginx Ingress Controller, KONG, GCE L7 Load Balancer

Ingress Routing

- Host-based Routing
 - 사용자가 blue.example.com 와 green.example.com에 접근을 하게 되면 같은 Ingress endpoint에서,
 - 각각 nginx-blue-svc와 nginx-green-svc로 요청이 포워드

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web-ingress
  namespace: ingress-basic
spec:
  rules:
    - host: blue.example.com
      http:
        paths:
          - backend:
              serviceName: nginx-blue-svc
              servicePort: 80
    - host: green.example.com
      http:
        paths:
          - backend:
              serviceName: nginx-green-svc
              servicePort: 80
```

<Host-based routing>

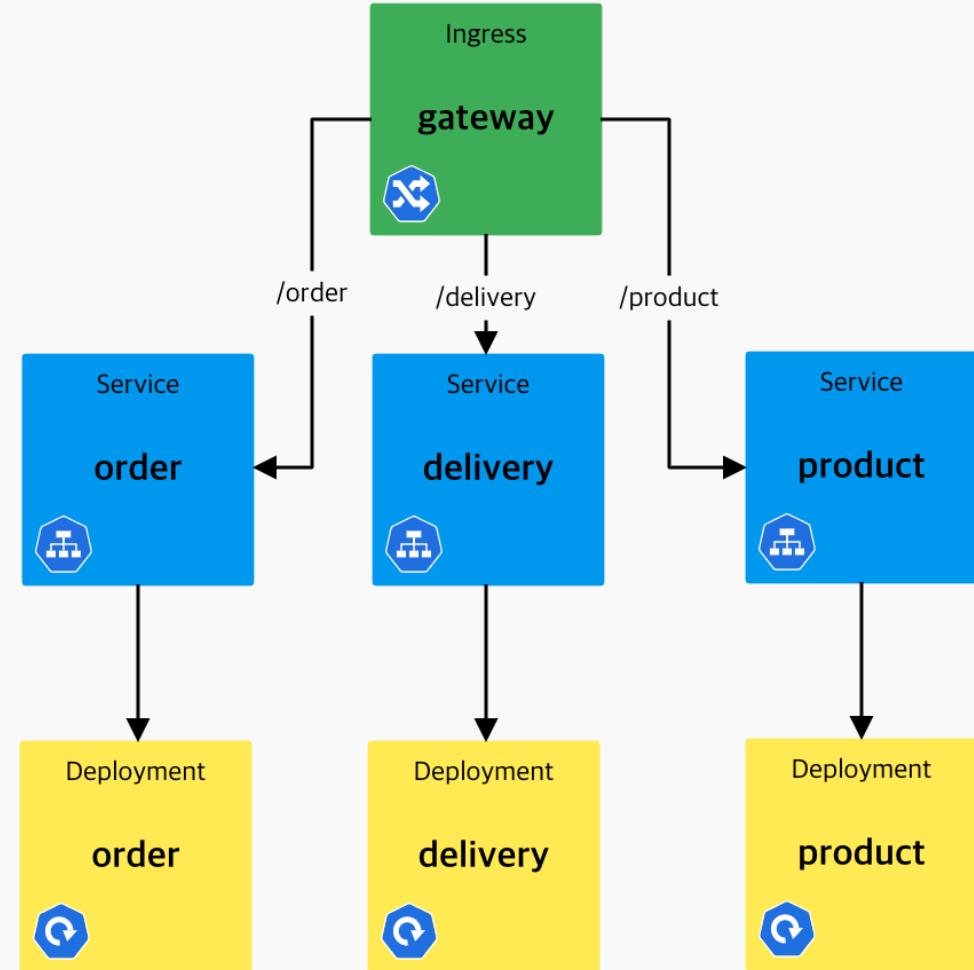
Ingress Routing

- Path-based Routing
 - Ingress는 또한 example.com/blue 와 example.com/green 형태의 요청에 대해,
 - 각각 nginx-blue-svc와 nginx-green-svc로 요청이 라우팅

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web-ingress
  namespace: ingress-basic
spec:
  rules:
    - http:
        paths:
          - path: /blue/*
            backend:
              serviceName: nginx-blue-svc
              servicePort: 80
          - path: /green/*
            backend:
              serviceName: nginx-green-svc
              servicePort: 80
```

<Path-based routing>

Lab. Deploying 12-Street



여우야 여우야 뭐하니?

밥 먹는다

무슨 반찬?

개구리 반찬



살았니 죽었니?

살았다 야!

Probe types

쿠버네티스는 각 컨테이너의 상태를 주기적으로 체크(Health Check)해서, 문제가 있는 컨테이너를 자동으로 재시작하거나, 또는 문제가 있는 컨테이너를 서비스에서 제외하는 등, 실행중인 컨테이너의 상태를 파악하고 라이프사이클을 제어한다.

- Liveness Probe
 - 문제가 발생한 컨테이너를 종료하고, RestartPolicy ([default: Always](#))에 따라 다시 만들어지거나, 종료된 상태로 남는다.
- Readiness Probe
 - 문제가 발생한 컨테이너를 서비스 백엔드에서 일시적으로 제외한다. 해당 컨테이너를 임시적으로 disable 상태로 전환시킨 후, 지속적으로 핸들러를 호출하여 결과를 확인한다.

Probe Actions

- Liveness probe와 readiness probe는 컨테이너가 정상적인지 아닌지를 체크하는 방법으로 다음과 같이 3가지 방식을 제공한다.
 - ExecAction
 - HttpGetAction
 - TCPSocketAction

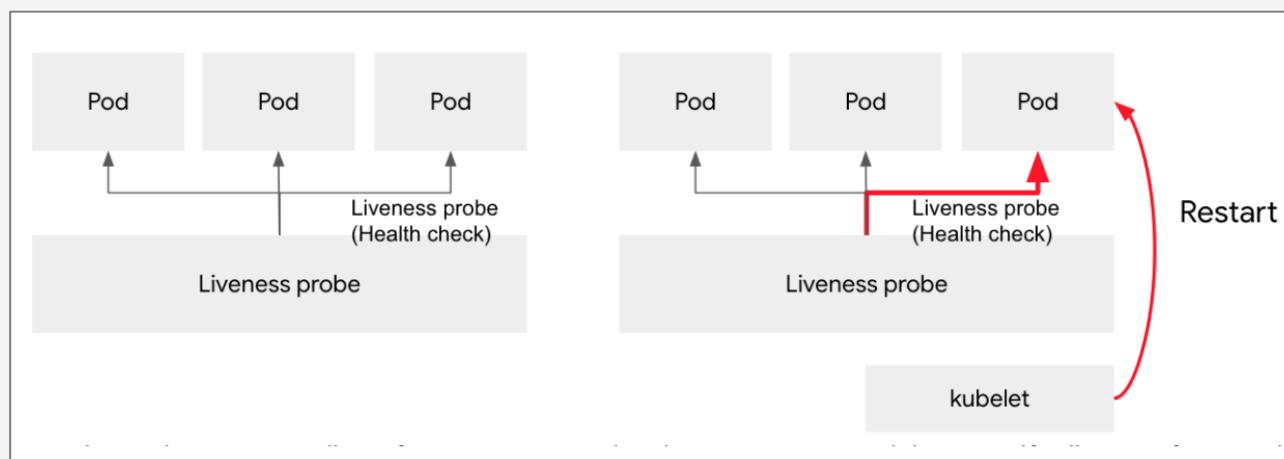
Probe Configuration

- **initialDelaySeconds** : 컨테이너가 시작된 후, Probe가 작동하기까지 딜레이 시간
 - 기본값 : 0초
- **periodSeconds** : Probe를 수행하는 주기(빈도)
 - 기본값 : 10초
- **timeoutSeconds** : Probe가 작동하고 최대 기다리는 시간(타임아웃)
 - 기본값 : 1초
- **successThreshold** : Probe가 한번 실패한 후, 다시 성공으로 간주하는 회수
 - 기본값 : 1회
- **failureThreshold** : Probe가 최종 실패로 간주하는 회수
 - 기본값 : 3회

Health Total Check Time = initialDelaySeconds + (periodSeconds * failureThreshold)

Liveness Probe

- Pod는 정상적으로 작동하지만 내부의 어플리케이션이 반응이 없다면, 컨테이너는 의미가 없다.
 - 위와 같은 경우는 어플리케이션의 Deadlock 또는 메모리 과부화로 인해 발생할 수 있으며, 발생했을 경우 컨테이너를 다시 시작해야 한다.
- Liveness probe는 Pod의 상태를 체크하다가, Pod의 상태가 비정상인 경우 kubelet을 통해서 재시작한다.



Liveness Command probe

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
    - name: liveness
      image: k8s.gcr.io/busybox
      args:
        - /bin/sh
        - -c
        - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
          - cat
          - /tmp/healthy
      initialDelaySeconds: 3
      periodSeconds: 5
```

- 왼쪽은 /tmp/healthy 파일이 존재하는지 확인하는 설정파일이다.
- periodSeconds 파라미터 값으로 5초마다 해당 파일이 있는지 조회한다.
- initialDelaySeconds 파라미터는 kubelet이 첫 체크하기 전에 기다리는 시간을 설정한다.
- 파일이 존재하지 않을 경우, 정상 작동에 문제가 있다고 판단되어 kubelet에 의해 자동으로 컨테이너가 재시작 된다.

Liveness HTTP probe

- Kubelet이 HTTP GET 요청을 /healthz로 보낸다.
- 실패 했을 경우(응답상태가 200~400 구간인 경우), kubelet이 자동으로 컨테이너를 재시작한다.

```
livenessProbe:  
  httpGet:  
    path: /healthz  
    port: 8080  
    httpHeaders:  
      - name: X-Custom-Header  
        value: Awesome  
    initialDelaySeconds: 3  
    periodSeconds: 3
```

- 컨테이너가 참조하는 Data Store가 불능일 경우, 굳이 컨테이너를 재시작할 필요가 없으므로 Spring Actuator(/actuator/health)를 httpGet Action으로 Liveness Probe로 추천하지 않는다.

Liveness TCP Probe

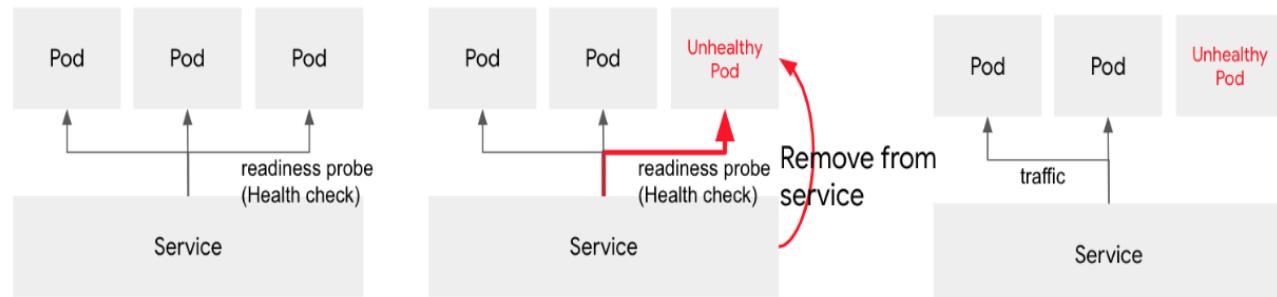
- kubelet은 TCP Liveness Probe를 통해, 지속적으로 어플리케이션이 실행중인 컨테이너의 TCP Socket을 열려고 한다.
- 정상이 아닌 경우 컨테이너를 재시작한다.

```
livenessProbe:  
  tcpSocket:  
    port: 8080  
  initialDelaySeconds: 15  
  periodSeconds: 20
```

Readiness Probe

- Configuration을 로딩하거나, 많은 데이터를 로딩하거나, 외부 서비스를 호출하는 경우에는 일시적으로 서비스가 불가능한 상태가 될 수 있다.
- Readiness Probe를 사용하게 되면 주어진 조건이 만족할 경우, 서비스 라우팅하고, 응답이 없거나 실패한 경우, 서비스 목록에서 제외

```
readinessProbe:  
  exec:  
    command:  
      - cat  
      - /tmp/healthy  
  initialDelaySeconds: 5  
  periodSeconds: 5
```



Difference between Liveness and Readiness

- Liveness probe와 Readiness probe 차이점은
 - Liveness probe는 컨테이너의 상태가 비정상이라고 판단하면,
→ 해당 Pod를 재시작하는데 반해,
 - Readiness probe는 컨테이너가 비정상일 경우에는
→ 해당 Pod를 사용할 수 없음으로 표시하고, 서비스등에서 제외한다.
 - 주기적으로 체크하여, 정상일 경우 정상 서비스에 포함

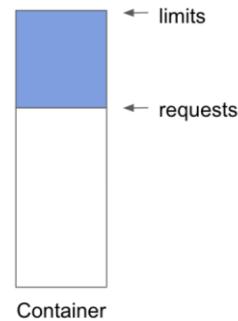
Lab. Liveness, Readiness



Lab Time

Resource Assign & Management

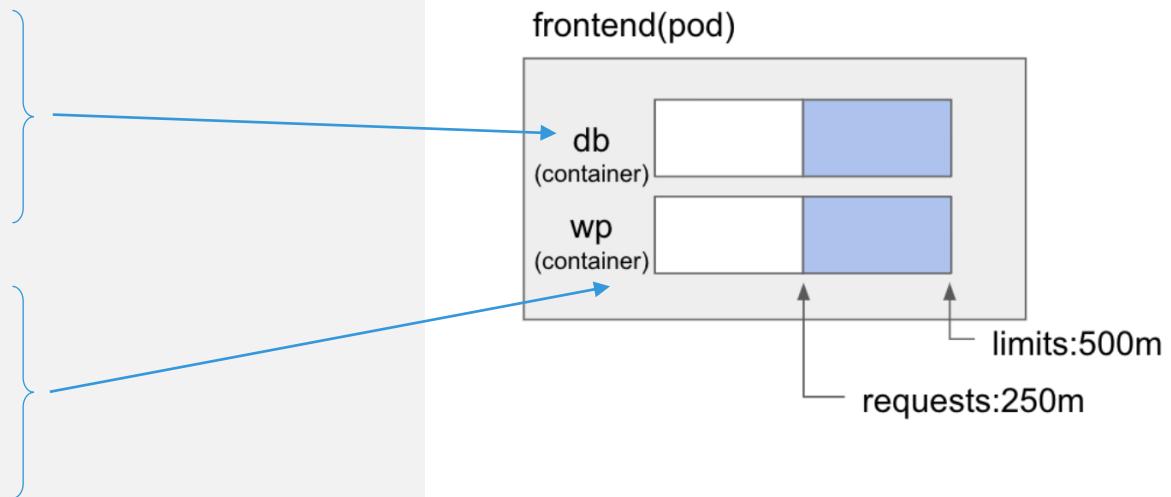
- 쿠버네티스에서 Pod를 어느 노드에 배포할지 결정하는 것을 스케줄링이라 함
- Pod에 대한 스케줄링시에, Pod내의 애플리케이션이 동작할 수 있는 자원(CPU,메모리 등) 정보를 알아야 그만한 자원이 가용한 노드에 Pod 배포 가능
- 리소스 단위
 - CPU의 경우 ms(밀리 세컨드)를 사용하는데 대략 1000ms가 1 vCore (가상 CPU 코어)
 - 메모리의 경우 Mb를 사용하며 64M(64×1000), 또는 64Mi (64×1024)로 계산
 - Request & Limit
 - Request : 컨테이너가 생성될 때 요청하는 리소스 양
 - Limit : 리소스가 더 필요한 경우 추가로 사용 가능한 양



Resource Assign & Management

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    - name: db
      image: mysql
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
    - name: wp
      image: wordpress
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

- 샘플 설정에 따른 Pod내 CPU 리소스 할당



Resource Monitoring (1/2)

- Node의 자원 상태 모니터링을 위한 Metric Server(메트릭 서버) 설치
 - 설치 Commands

```
git clone https://github.com/kubernetes-incubator/metrics-server.git  
cd metrics-server/  
kubectl create -f deploy/kubernetes/
```

- 설치 확인

```
apexacme@APEXACME:~$ kubectl get all -l 'k8s-app in (metrics-server)' -n kube-system  
NAME                 READY   STATUS    RESTARTS   AGE  
pod/metRICS-server-7668599459-9zxb9  1/1     Running   0          2d3h  
  
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE  
deployment.apps/metRICS-server  1/1     1           1          2d3h  
  
NAME                      DESIRED   CURRENT   READY   AGE  
replicaset.apps/metRICS-server-7668599459  1         1         1        2d3h
```

Resource Monitoring (2/2)

- Node의 자원 상태 모니터링

- \$ kubectl get nodes
- \$ kubectl describe nodes

Namespace	Name	CPU Requests	CPU Limits	Memory Requests	Memory Limits	AGE
ingress-basic	nginx-ingress-controller-5b85669986-g4n8t	0 (0%)	0 (0%)	0 (0%)	0 (0%)	5h14m
ingress-basic	nginx-ingress-default-backend-6b8dc9d88f-1bd9g	0 (0%)	0 (0%)	0 (0%)	0 (0%)	5h14m
kube-system	kube-proxy-47g4j	100m (5%)	0 (0%)	0 (0%)	0 (0%)	2d2h
kube-system	omsagent-8cr5p	75m (3%)	150m (7%)	225Mi (4%)	600Mi (13%)	7d
kube-system	tiller-deploy-7b98f7c844-t7cpn	0 (0%)	0 (0%)	0 (0%)	0 (0%)	6h55m
kube-system	tunnelfront-c8df6fcf-c7xz	10m (0%)	0 (0%)	64Mi (1%)	0 (0%)	7d

Allocated resources:

(Total limits may be over 100 percent, i.e., overcommitted.)

Resource	Requests	Limits
cpu	185m (9%)	150m (7%)
memory	289Mi (6%)	600Mi (13%)
ephemeral-storage	0 (0%)	0 (0%)
attachable-volumes-azure-disk	0	0

Events:

<none>

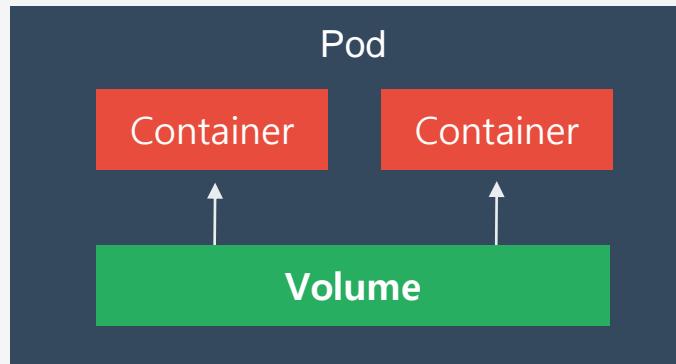
apexacme@APEXACME: ~/yaml/ingress\$

- 현재 사용 중인 리소스 현황 모니터링

- \$ kubectl top nodes
- \$ kubectl top pods

Volumes

- 쿠버네티스는 여러 호스트에 걸쳐 Stateless한 컨테이너를 마이크로서비스로 배포하는 것이 목표이기에 영속성 있는 저장장치(Persistent Volume)를 고려해야 함



- Volume은 Pod에 장착되어, 그 Pod에 있는 Container 간에 공유

Types of Volumes

- Pod에 마운트된 디스크는 Volume type에 따라 사용 유형이 정의
- Volume Type 내 디스크의 크기, 내용 등의 속성 설정
- Types of Volumes

임시 볼륨	로컬 볼륨	네트워크 볼륨	네트워크 볼륨 (Cloud dependent)
emptyDir	hostPath	gitRepo, iSCSI, NFS cephFS, glusterFS	gcePersistentDisk, AzureDisk, Amazon EFS, Amazon EBS ...
✓ Pod내 컨테이너간 공유	✓ Host 디렉토리를 Pod와 공유 해 사용하는 방식	✓ 영구적으로 영속성 있는 데이터 관리 목적	
✓ Pod가 삭제되면, emptyDir도 지워지므로 휘발성 데이터 저장 용도	✓ 컨테이너에 nodeSelector를 지정안하면 매번 다른 호스 트에 할당 (e.g. 호스트의 Metric 수집해야 하는 경우)	✓ 쿠버네티스는 PV와 PVC의 개념을 통해 Persistent 볼 륨을 Pod에 제공	✓ gitRepo, iSCSI, NFS와 같은 표준 네트워크 볼륨과 Cloud Vendor가 제공하는 볼륨으로 구분

Volumes : emptyDir

```
apiVersion: v1
kind: Pod
metadata:
  name: shared-volumes
spec:
  containers:
    - image: redis
      name: redis
      volumeMounts:
        - name: shared-storage
          mountPath: /data/shared
    - image: nginx
      name: nginx
      volumeMounts:
        - name: shared-storage
          mountPath: /data/shared
  volumes:
    - name: shared-storage
      emptyDir: {}
```

- emptyDir의 생명주기는 컨테이너 단위가 아닌 Pod 단위로 Container 재기동에도 계속 사용 가능

- 생성된 Pod 확인

```
apexacme@APEXACME: ~/yaml$ kubectl get all
NAME           READY   STATUS    RESTARTS   AGE
pod/shared-volumes   2/2     Running   0          10m
```

- 지정 컨테이너 접속 후, 파일 생성
 - kubectl exec -it shared-volumes --container redis -- /bin/bash
 - cd /data/shared
 - echo test... > test.txt
- 다른 컨테이너로 접속 후, 파일 확인
 - kubectl exec -it shared-volumes --container nginx - /bin/bash
 - cd /data/shared
 - ls

Volumes : hostPath

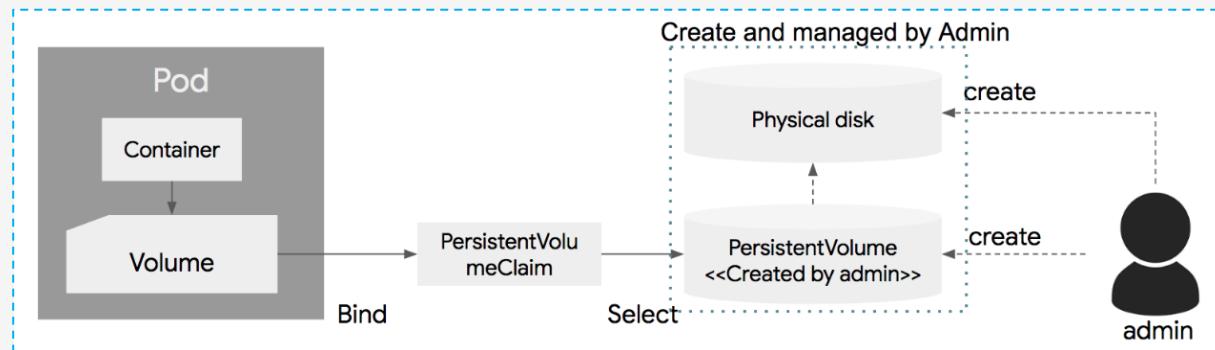
```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath
spec:
  containers:
    - name: redis
      image: redis
      volumeMounts:
        - name: somepath
          mountPath: /data/shared
  volumes:
    - name: somepath
      hostPath:
        path: /tmp
      type: Directory
```

- Node의 Local 디스크 경로를 Pod에 마운트
- 같은 hostPath에 있는 볼륨은 여러 Pod사이에서 공유
- Pod가 삭제되어도 hostPath에 있는 파일은 유지
- Pod가 재기동 되어 다른 Node에서 기동될 경우, 새로운 Node의 hostPath를 사용
- Node의 로그 파일을 읽는 로그 에이전트 컨테이너 등에 사용 가능
- Pod 생성 및 확인 (Pod 내, ls -al /data/shared)

```
drwxrwxrwt 8 redis root 4096 Feb 17 03:08 .
drwxr-xr-x 3 redis redis 4096 Feb 17 03:07 .
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .ICE-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .Test-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .X11-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .XIM-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .font-unix
drwx----- 3 redis root 4096 Feb 11 05:44 systemd-private-fe55104f60e34b2ea4
```

PersistentVolume & PersistentVolumeClaim

- 특정 IT 환경에서는 영속성 있는 대용량 스토리지는 관리자에 의해 관리
 - 쿠버네티스 클러스터를 사용하는 개발자로부터 볼륨 프로비저닝 역할을 분리하는 사상
- 시스템 관리자가 실제 물리 디스크를 생성한 뒤, 이 디스크를 PersistentVolume 이라는 이름으로 Kubernetes에 등록
- 개발자는 Pod 생성 시, 볼륨을 정의하고, 해당 볼륨의 정의 부분에 PVC(PersistentVolumeClaim)를 지정하여 관리자가 생성한 PV와 연결



StorageClass - Dynamic PV Provisioning



- PV는 관리자에 의해 수동으로 생성될 수 있지만, 자동 생성도 가능(Dynamic Provisioning)
- StorageClass(SC) Object
- StorageClass 객체에 의해 PersistentVolumes 동적 제공 가능
 - StorageClass는 PersistentVolume를 만들기 위해 Cloud Provider별 CSI 인터페이스를 구현하여 제공
- PersistentVolumes 스토리지 관리를 제공하는 Volume Types :
 - GCEPersistentDisk, AWSElasticBlockStore, AzureDisk, NFS, iSCSI

AWS Storage Provisioner

- AWS Storage 유형 비교

항 목	EBS (Elastic Block Store)	EFS (Elastic File System)
Zone(존) Accessibility	✓ 한 존에서만 접근 가능	✓ 여러 존에서 접근 가능
Access Mode (Concurrent Connection)	✓ 한번에 한 Pod에서만 연결 ✓ ReadWriteOnce	✓ 한번에 여러 Pod에서 접근 가능 ✓ ReadWriteMany 포함 모든 모드지원
Volume Cost	✓ 처음 볼륨 크기를 설정하고, 그 크기만큼 비용 지불	✓ 사용한 만큼 볼륨사이즈가 결정되고, 사용한 만큼 비용지불

- Default AWS StorageClass (Provisioner) 확인

```
apexacme@APEXACME:~/workdir$ kubectl get storageclass
NAME      PROVISIONER      AGE
gp2 (default)  kubernetes.io/aws-ebs  25h
```

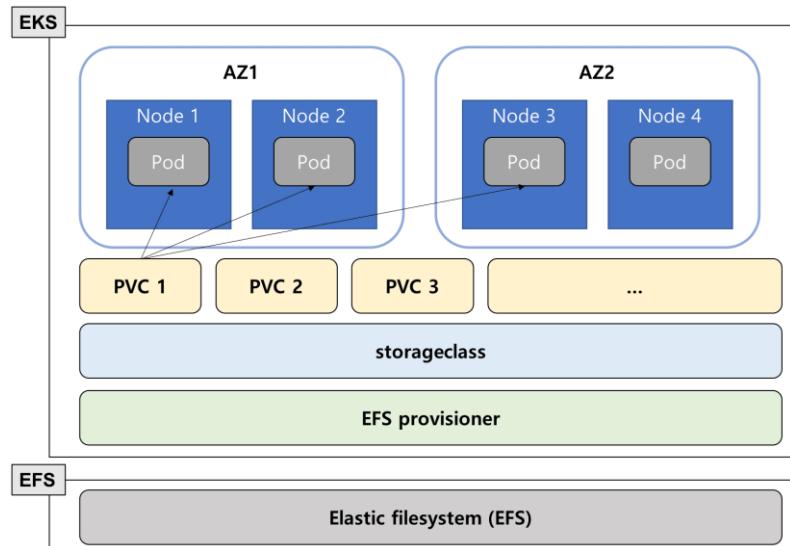
Using EFS in EKS Cluster

- EFS (Elastic File System) 사용을 위한 태스크 오더

- Step. 1: 파일시스템 생성
 - Step. 2: EFS CSI 드라이버 설치
 - Step. 3: StorageClass 생성
 - Step. 4: PVC 생성

- 설정 확인

- PVC를 컨테이너에 마운팅



이미지 출처 : <https://kscorey.com/dev/aws/eks-efs>

Using EFS in EKS Cluster (1/4)

- Step. 1: 파일시스템 생성 (AWS Console에 접속)

The screenshot shows the AWS console interface for creating an EFS file system. On the left, a modal window titled '파일 시스템 생성' (File System Creation) is open. It asks for the file system name ('이름 - 선택 사항') and the VPC ('Virtual Private Cloud(VPC)'). The 'Standard' storage class is selected. At the bottom are '취소' (Cancel), '사용자 지정' (Customize), and a large orange '생성' (Create) button. A red arrow labeled '1' points to the '이름' input field. Another red arrow labeled '2' points to the '내 VPC 선택' dropdown. A third red arrow labeled '3' points to the '생성' button.

On the right, the main Amazon EFS landing page is displayed. It features a dark header with the service name and a light-colored body. The body contains a summary of the service, a question 'Amazon Elastic File System이란 무엇입니까?', and a table of storage classes and their costs.

Amazon Elastic File System

조정 가능하고 탄력적인 클라우드 네이티브 NFS 파일 시스템

Amazon Elastic File System(Amazon EFS)은 AWS 클라우드 서비스 또는 온프레미스 리소스와 함께 사용되는 범용 워크로드를 위한 단순하고, 조정 가능하며, 탄력적인 파일 시스템을 제공합니다.

Amazon Elastic File System이란 무엇입니까?

요금
Standard 스토리지 GB당 US\$0.33
Standard - Infrequent Access 스토리지 GB당 US\$0.027

• EFS 파일시스템을 생성한다. 이때 내 VPC 선택이 중요!

Using EFS in EKS Cluster (1/4)

- Step. 1: 파일시스템 생성 (AWS Console에 접속)

The screenshot shows the AWS EFS console interface. On the left, there is a table for managing network interfaces across three regions (ap-northeast-2a, ap-northeast-2b, ap-northeast-2c). A red box labeled '2' highlights the '네트워크' (Network) tab in the top navigation bar. A red arrow labeled '3' points from the '네트워크' tab to the '관리' (Management) button in the top right corner of the table header.

On the right, a modal window titled 'Amazon EFS Elastic Throughput 소개' (Amazon EFS Elastic Throughput Introduction) is open. It contains a message about elastic throughput and success logs. A green box labeled '성공!' (Success!) indicates that the file system 'fs-054b5c53a97673dd8' is now available.

In the bottom right corner, a callout box contains the text: '• 생성된 파일시스템 내에서 “네트워크 > 관리”를 클릭한다.' (Click on 'Network > Management' in the newly created file system.)

가용 영역	탑재 대상 ID	서브넷 ID	탑재 대상 상태	IP 주소	네트워크 인터페이스 ID	보안 그룹
ap-northeast-2a	fsmt-009f258e3d2d6a325	subnet-0cf18cfaf91a596c9	사용 가능	192.168.10.7.77	eni-07d60bea512ec709a	sg-0e0c9d90bc6140e15 (default)
ap-northeast-2b	fsmt-01fcc766b5d541f07	subnet-04a979bc414d83e0e	사용 가능	192.168.70.100	eni-05e357e9b9fc67826	sg-0e0c9d90bc6140e15 (default)
ap-northeast-2c	fsmt-0a5eaf1e4ffd2d96a	subnet-071d22e8d2c55e4ff	사용 가능	192.168.14.5.107	eni-05e357e9b9fc67826	sg-0e0c9d90bc6140e15 (default)

스템 (1)	C	세부 정보 보기	삭제	파일 시스템 생성
fs-054b5c53a97673dd8	암호화됨	6.00 KiB	6.00 KiB	0바이트

Page: 383

Using EFS in EKS Cluster (1/4)

- Step. 1: 파일시스템 생성 (AWS Console에 접속)

The screenshot shows two separate CloudFormation stacks. The top stack is for the 'ap-northeast-2a' region, and the bottom stack is for the 'ap-northeast-2b' region. Both stacks are associated with the same subnet and IP range (192.168.107.77 and 192.168.70.59 respectively). In the 'ap-northeast-2a' stack, a security group named 'sg-0816157f152384616' is attached, which includes the 'eksctl-sre-eks-cluster-ClusterSharedNodeSecurityGroup-UWZB5AYCJQ5W' rule. A note below the first stack indicates that the 'ClusterSharedNodeSecurityGroup' is added to all security groups in the same availability zone. The bottom stack also lists the same security group and rule.

- 출력되는 모든 가용영역의 보안그룹에 ClusterSharedNodeSecurityGroup을 추가한다.

가용 영역	서브넷 ID	IP 주소	보안 그룹
ap-northeast-2a	subnet-0cf18cfaf91	192.168.107.77	보안 그룹 선택▼ sg-0816157f152384616 eksctl-sre-eks-cluster-ClusterSharedNodeSecurityGroup-UWZB5AYCJQ5W + 자세히 표시 (+1)
ap-northeast-2b	subnet-04a979bc41	192.168.70.59	보안 그룹 선택▼

가용 영역	서브넷 ID	IP 주소	보안 그룹
ap-northeast-2c	subnet-071d22e8d;	192.168.145.107	보안 그룹 선택▼ sg-0e0c9d90bc6 140e15 default

384

Using EFS in EKS Cluster (2/4)

- Step. 2: EFS CSI 드라이버 설치
 - EKS Cluster에 IAM 계정(정책) 설정

```
eksctl create iamserviceaccount \
--override-existing-serviceaccounts \
--region $REGION \
--name efs-csi-controller-sa \
--namespace kube-system \
--cluster $CLUSTER_NAME \
--attach-policy-arn arn:aws:iam::$AWS_ROOT_UID:policy/EFSCSIControllerIAMPolicy \
--approve
```

- Cluster에 EFS CSI 드라이버 설치

```
helm repo add aws-efs-csi-driver https://kubernetes-sigs.github.io/aws-efs-csi-driver
helm repo update
helm upgrade -i aws-efs-csi-driver aws-efs-csi-driver/aws-efs-csi-driver \
--namespace kube-system \
--set image.repository=[리전별 이미지 저장소]/eks/aws-efs-csi-driver \
--set controller.serviceAccount.create=false \
--set controller.serviceAccount.name=efs-csi-controller-sa
```

Using EFS in EKS Cluster (3/4)

- Step. 3: StorageClass 생성

```
kubectl apply -f - <<EOF
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: $FILE_SYSTEM_ID
  directoryPerms: "700"
EOF
```

- 생성 확인

```
apexacme@APEXACME:~$ kubectl get storageclass
NAME      PROVISIONER          AGE
efs-sc    efs.csi.aws.com    118m
gp2 (default)  kubernetes.io/aws-ebs  4d18h
```

Using EFS in EKS Cluster (4/4)

- Step. 4: PVC(PersistentVolumeClaim) 생성

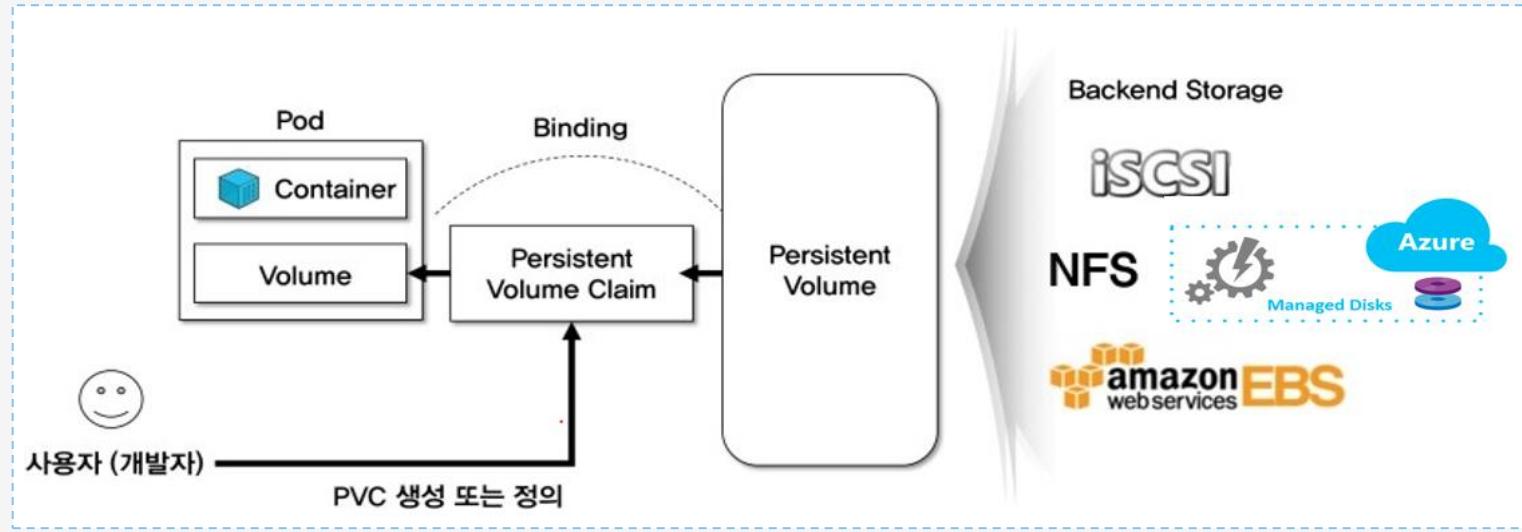
```
kubectl apply -f - <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-pvc
  labels:
    app: efs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
  storageClassName: efs-sc
EOF
```

← Step.3에서 생성한 StorageClass 이름 입력

- 생성 확인

```
apexacme@APEXACME:~/workdir$ kubectl get pvc
NAME      STATUS  VOLUME                                     CAPACITY  ACCESS MODES  STORAGECLASS  AGE
efs-pvc   Bound   pvc-346af366-1ff1-4a47-97c9-467450ac922c  1Mi       RWX          aws-efs     43h
```

PersistentVolume Claims & Binding



- Pod가 크기, 접근 모드에 따라 PVC를 요청, 적합한 PersistentVolume 발견시 PersistentVolume Claim에 바인딩
- PVC 조건을 만족하는 PV가 없을 경우, PV를 StorageClass가 자동으로 Provisioning하여 바인딩

Lab. Create PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-pvc
  labels:
    app: efs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
  storageClassName: efs-sc
```

- **accessMode:**
 - `ReadWriteOnce` : 하나의 Pod에만 마운트되고, 읽고 쓰기 가능
 - `ReadOnlyMany` : 여러 개의 Pod에서 마운트되고, 동시에 읽기만 가능 (쓰기는 불가능)
 - `ReadWriteMany` : 여러 개의 Pod에서 마운트되고, 동시에 읽고 쓰기 가능
- `kubectl apply -f volume-pvc.yaml`
- `kubectl get pvc`

```
apexacme@APEXACME:~/workdir$ kubectl get pvc
NAME      STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS  AGE
aws-efs   Bound   pvc-346af366-1ff1-4a47-97c9-467450ac922c  1Mi       RWX        aws-efs       43h
efs       Bound   pvc-6d1170cf-3bb0-47ea-90b0-30bc3bc50e3e  1Mi       RWX        aws-efs       2d1h
```

- `kubectl describe pvc`

```
apexacme@APEXACME:~/workdir$ kubectl describe pvc aws-efs
Name:           aws-efs
Namespace:      default
StorageClass:   aws-efs
Status:         Bound
Volume:         pvc-346af366-1ff1-4a47-97c9-467450ac922c
Labels:         app=test-pvc
Annotations:    pv.kubernetes.io/bind-completed: yes
                  pv.kubernetes.io/bound-by-controller: yes
                  volume.beta.kubernetes.io/storage-provisioner: apexacme.com/aws-efs
Finalizers:     [kubernetes.io/pvc-protection]
Capacity:       1Mi
Access Modes:   RWX
```

Lab. Create Pod with PersistentVolumeClaim

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: username/order:v1
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      volumeMounts:
        - mountPath: "/mnt/aws"
          name: volume
    volumes:
      - name: volume
        persistentVolumeClaim:
          claimName: efs-pvc
```

- kubectl apply -f pod-with-pvc.yaml
- kubectl get pod
- kubectl describe pod mypod
- kubectl exec -it mypod -- /bin/bash
- cd /mnt/aws
- df -k 로 PVC 마운트 확인

1K-blocks	Used	Available	Use%	Mounted on
20959212	3037876	17921936	15%	/
65536	0	65536	0%	/dev
1988964	0	1988964	0%	/sys/fs/cgroup
20959212	3037876	17921936	15%	/etc/hosts
39254739968	0	9007199254739968	0%	/mnt/aws
65536	0	65536	0%	/dev/shm
1988964	12	1988952	1%	/run/secrets/kubernetes.io-token-4qjwv
1988964	0	1988964	0%	/proc/acpi
1988964	0	1988964	0%	/sys/firmware

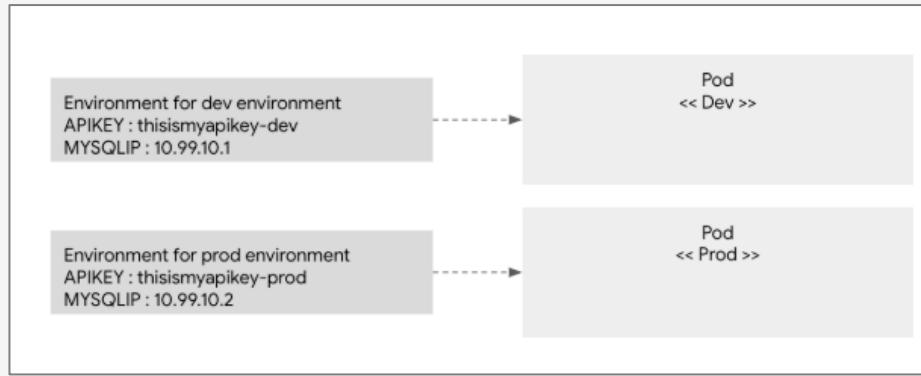
Lab. Volumes



Lab Time

ConfigMaps

- ConfigMaps는 컨테이너 이미지로부터 설정 정보를 분리할 수 있게 해준다.
- 환경변수나 설정값들을 환경변수로 관리해 Pod가 생성될 때 이 값을 주입



- ConfigMaps은 2가지 방법으로 생성
 - 리터럴 값
 - 파일
- ConfigMaps는 etcd에 저장

Pod에서 ConfigMap 추가 사용하기

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: cm-file-deployment
spec:
  replicas: 3
  minReadySeconds: 5
  selector:
    matchLabels:
      app: cm-file
  template:
    metadata:
      name: cm-file-pod
      labels:
        app: cm-file
    spec:
      containers:
        - name: cm-file
          image: (283210891307.dkr.ecr.ap-northeast-2.amazonaws.com)/cm-file:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          env:
            - name: PROFILE
              valueFrom:
                configMapKeyRef:
                  name: cm-file
                  key: profile.properties
```

- 환경변수로 값 전달

- o cm-file configMap에서 키가 "profile.properties" (파일명)인 값을 읽어와서 환경 변수 PROFILE에 저장
- o 저장된 값은 파일의 내용인 아래 문자열이 됨
 - o myname=terry
email=myemail@mycompany.com
address=seoul

Pod에서 ConfigMap 추가 사용하기

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: cm-file-deployment-vol
spec:
  replicas: 3
  minReadySeconds: 5
  selector:
    matchLabels:
      app: cm-file-vol
  template:
    metadata:
      name: cm-file-vol-pod
      labels:
        app: cm-file-vol
    spec:
      containers:
        - name: cm-file-vol
          image: (283210891307.dkr.ecr.ap-northeast-2.amazonaws.com)/cm-file-volume:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: config-profile
              mountPath: /tmp/config
      volumes:
        - name: config-profile
          configMap:
            name: cm-file
```

- 디스크 볼륨으로 마운트 하기
 - ConfigMap을 Volume으로 정의하고, 이 볼륨을 volumeMounts를 이용해 /tmp/config에 마운트 함
 - 이때 중요한점은 마운트 포인트에 마운트 될 때, ConfigMap내의 키가 파일명이 됨

리터럴 값으로부터 ConfigMap 생성

- ConfigMap을 생성하는 명령어

```
$ kubectl create configmap my-config --from-literal=key1=value1 -  
-from-literal=key2=value2  
configmap "my-config" created
```

- 설정된 ConfigMap 정보 가져오기

```
$ kubectl get configmaps my-config -o yaml  
apiVersion: v1  
data:  
  key1: value1  
  key2: value2  
kind: ConfigMap  
metadata:  
  creationTimestamp: 2017-05-31T07:21:55Z  
  name: my-config  
  namespace: default  
  resourceVersion: "241345"  
  selfLink: /api/v1/namespaces/default/configmaps/my-config  
  uid: d35f0a3d-45d1-11e7-9e62-080027a46057
```

- o yaml 옵션은 해당 정보를 yaml형태로 출력하도록 요청한다.
- 해당 객체는 종류가 ConfigMap이며 key-value 값을 가지고 있다.
- ConfigMap의 이름 등의 정보는 metadata field에 들어 있다.

파일로부터 ConfigMap 생성 (1/2)

- 아래와 같은 설정 파일을 만든다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: customer1
data:
  TEXT1: Customer1_Company
  TEXT2: Welcomes You
  COMPANY: Customer1 Company Technology Pct. Ltd.
```

- customer1-configmap.yaml**라는 이름으로 파일을 생성하였을 경우, 아래와 같이 ConfigMap를 생성한다.

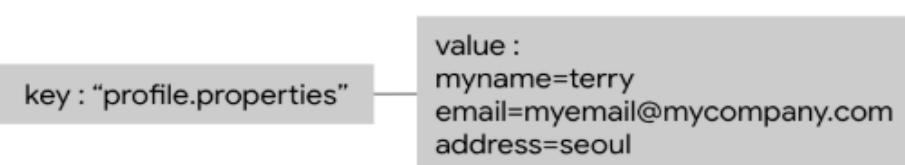
```
$ kubectl create -f customer1-configmap.yaml
configmap "customer1" created
```

파일로부터 ConfigMap 생성 (2/2)

- Userinfo.properties 파일을 생성하고,

```
myname=apexacme  
email=apexacme@uengine.org  
Address=seoul
```

- 파일을 이용해 ConfigMap을 만들 때는 --from-file을 이용해 파일명을 넘긴다.
- kubectl create configmap cm-file --from-file=./properties/profile.properties
 - 이때, 키는 파일명이 되고, 값은 파일 내용이 됨



Secrets

- ConfigMap이 일반적인 환경 설정 정보나 Config정보를 저장하도록 디자인 되었다면, 보안이 중요한 패스워드나 API 키, 인증서 파일들은 Secret에 저장
- Secret은 정보보안 차원에서 추가적인 보안 기능을 제공
 - 예를 들어, API서버나 Node의 파일에 저장되지 않고, 항상 메모리에 저장되므로 상대적 접근이 어려움
 - Secret의 최대 크기는 1MB (너무 커지면, apiserver나 Kubelet의 메모리에 부하 발생)
- ConfigMap과 기본적으로 유사하나, 값(value)에 해당하는 부분을 base64로 인코딩해야 함
 - SSL인증서와 같은 binary파일의 경우, 문자열 저장이 불가능하므로 인코딩 필요
 - 이를 환경변수로 넘길 때나 디스크볼륨으로 마운트해서 읽을 경우 디코딩되어 적용

```
apiVersion: v1
kind: Secret
metadata:
  name: hello-secret
data:
  language: amF2YQo=
```

Pod에서 Secret 사용하기

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: hello-secret-deployment
spec:
  replicas: 1
  minReadySeconds: 5
  selector:
    matchLabels:
      app: hello-secret-literal
  template:
    metadata:
      name: hello-secret-literal-pod
      labels:
        app: hello-secret-literal
    spec:
      containers:
        - name: hello-secret
          image: (283210891307.dkr.ecr.ap-northeast-2.amazonaws.com)/hello-secret:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          env:
            - name: LANGUAGE
              valueFrom:
                secretKeyRef:
                  name: hello-secret
                  key: language
```

- Deployment.yaml 생성/ 실행
- kubectl create -f hello-secret-deployment.yaml
- \$ kubectl get deploy

Kubectl 명령어로 Secret 생성 및 확인

- 명령어로 Secret 만들기
 - \$ kubectl create secret generic my-password --from-literal=password=mysqlpassword
 - my-password라는 Secret을 생성하고, password라는 key와 mysqlpassword라는 value 값을 가지게 된다.
 - Value는 base64로 자동 encoding
 - **generic** : create a secret from a local file, directory or literal value
- Secret 확인 : kubectl get secret my-password -o yaml
 - echo [base64 value] | base64 --decode

Secret을 직접 만들기

- base64 형태로 인코딩하여 YAML파일내에 직접 생성 가능

```
$ echo mysqlpassword | base64  
bXIzcWxwYXNzd29yZAo=
```

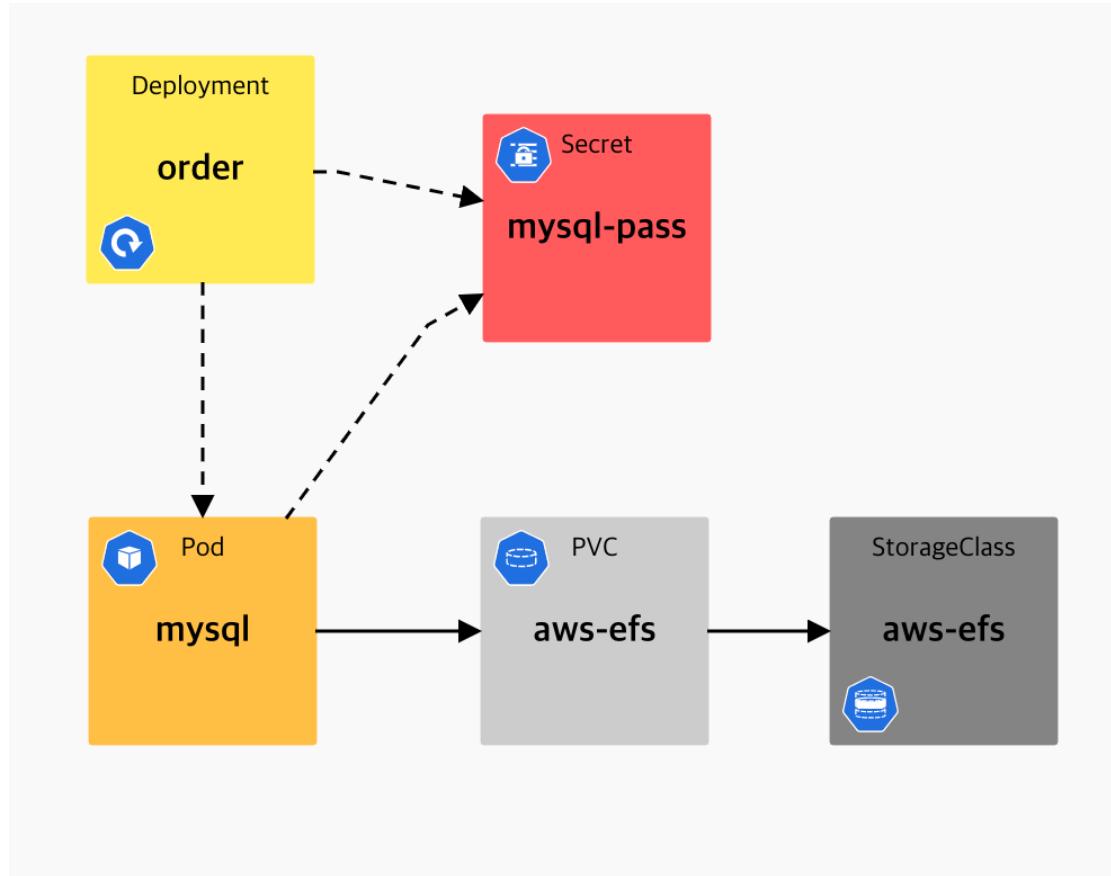
- 위 방식으로 인코딩 된 정보를 사용해 설정파일 생성

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: my-password  
type: Opaque  
data:  
  password: bXIzcWxwYXNzd29yZAo=
```

- base64** 인코딩은 바로 디코딩 됨으로 주의!

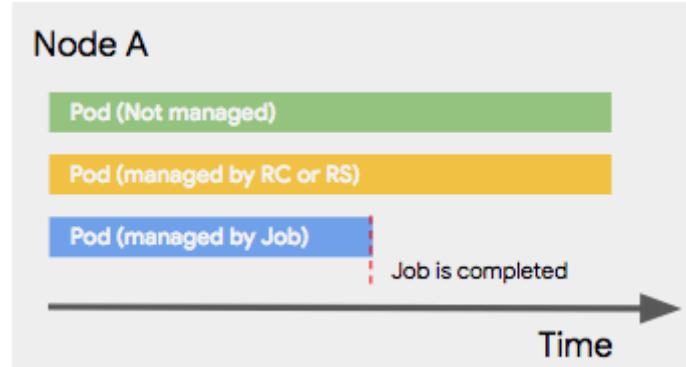
```
$ echo "bXIzcWxwYXNzd29yZAo=" | base64 --decode  
설정파일을 절대 소스코드에 넣지 않도록 주의한다!
```

Lab. Secret



Jobs

- 워크로드 모델 중, 배치나 한번 실행되고 끝나는 형태의 작업이 있을 수 있다.
- 예로, 원타임으로 파일 변환 작업을 하거나, 주기적으로 ETL 배치 작업을 하는 경우, Pod가 계속 떠 있을 필요 없이 작업을 할 때만 Pod를 실행한다.
- Job은 이러한 워크로드 모델을 지원하는 Controller



Jobs

- Job에 의해 관리되는 Pod는 Job이 종료되면 Pod도 같이 종료
- Job정의 시, Container Spec에 image뿐만 아니라, Job을 수행하기 위한 커맨드를 같이 입력

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle",
                     "print bpi(2000)"]
          restartPolicy: Never
          backoffLimit: 4
```

Job실패시, 처음부터 재시작 하지않음

Job실패시, 4번까지 재시도

Cron Jobs

- Job 컨트롤러에 의해 실행되는 작업을 주기적으로 스케줄링 해 주는 컨트롤러

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
  restartPolicy: OnFailure
```

Resource Quota

- ResourceQuota는 네임스페이스별로 사용 가능한 리소스 양을 정의

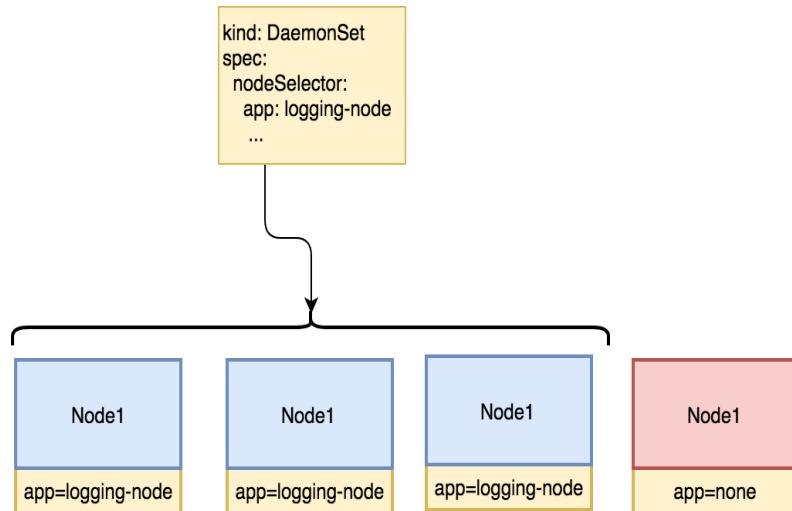
```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

<quota-mem-cpu.yaml>

- 네임스페이스에 ResourceQuota 적용
 - kubectl apply -f quota-mem-cpu.yaml --namespace=a-namespace

DaemonSets

- Pod를 각각의 노드에서 하나씩만 돌게 하는 형태로 Pod를 관리하는 컨트롤러
- 모니터링 데이터를 수집하거나, 스토리지 데몬을 실행하는 등, 모든 노드에서 항상 실행되는 특정 유형의 포드가 필요할 경우에 사용
 - Node 가 클러스터에 추가되면, 주어진 DaemonSet에 의해 Pod가 생성
 - DaemonSet이 삭제 되면, 관련된 모든 Node의 Pod들은 삭제
- 또한 특정 노드에만 Pod를 배포할 수 있도록, Pod의 “node selector”를 이용해서 라벨을 필터링하여 특정 노드만 선택 가능



StatefulSets

- 이전의 컨트롤러(Replica Set, Job) 같은 상태가 유지되지 않는 application을 관리하는 용도지만, StatefulSet은 단어의 의미 그대로 상태를 가지고 있는 포드들을 관리하는 컨트롤러
- 스테이트풀셋을 사용하면 볼륨을 사용해서 특정 데이터를 기록해두고 그걸 포드가 재시작했을 때도 유지할 수 있음
- StatefulSet controller는 이름, 네트워크 인증, 엄격한 순서 등의 독자성이 보장 되어야 할 때 사용
 - RS에 의해서 관리되는 Pod들은 기동이 될때 병렬로 동시에 기동 되나, DB의 경우에는 Master 노드가 기동 된 다음에, Slave 노드가 순차적으로 기동되어야 하는 순차성을 가지고 있는 경우가 있음
 - Ex) MySQL cluster, etcd cluster.
- 여러 개의 포드를 띄울 때 포드 사이에 순서를 지정해 지정된 순서대로 포드 실행 가능

StatefulSets

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: nginx
          image: k8s.gcr.io/nginx-slim:0.8
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
      volumeClaimTemplates:
        - metadata:
            name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: "standard"
        resources:
          requests:
            storage: 1Gi
```

- \$ kubectl get pod

NAME	READY	STATUS	RESTARTS	AGE
nginx-0	1/1	Running	0	13m
nginx-1	1/1	Running	0	12m
nginx-2	1/1	Running	0	12m

- \$ kubectl get pvc

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
www-nginx-0	Bound	pvc-e70627ea-2ecd-11e9-8d43-42010a920009	1Gi	RWO	standard	12m
www-nginx-1	Bound	pvc-f5f2d9d9-2ecd-11e9-8d43-42010a920009	1Gi	RWO	standard	12m
www-nginx-2	Bound	pvc-003c7376-2ece-11e9-8d43-42010a920009	1Gi	RWO	standard	12m

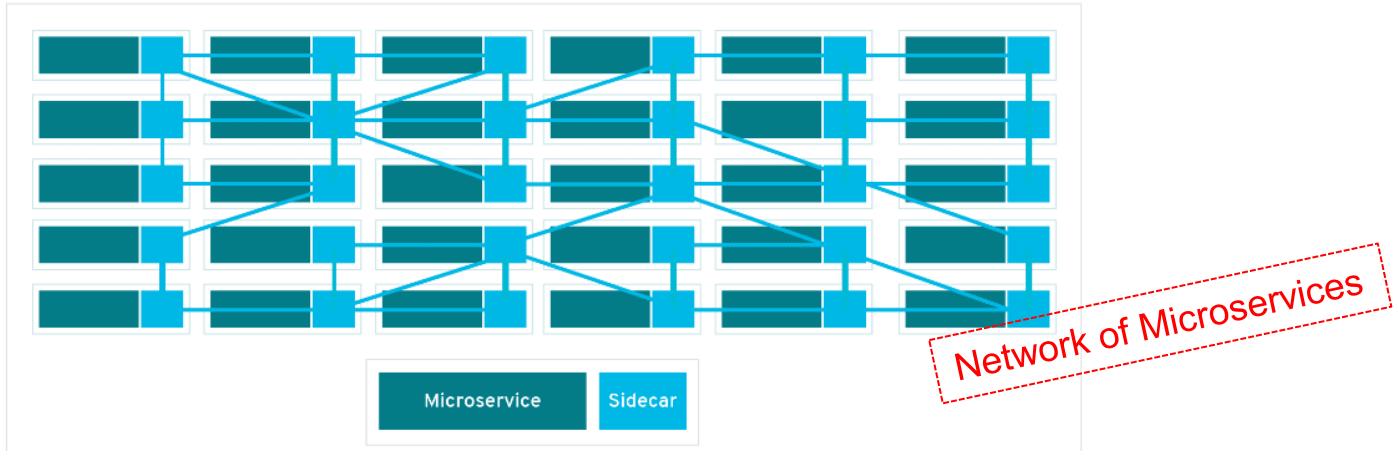
- StatefulSet은 Pod를 생성할 때 순차적으로 기동되고, 삭제할 때도 순차적으로(2 → 1 → 0) 삭제 됨

Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering ✓
- CI/CD and Progressive Deployment Strategy with Service Mesh

What's Service Mesh ?

Mesh Network



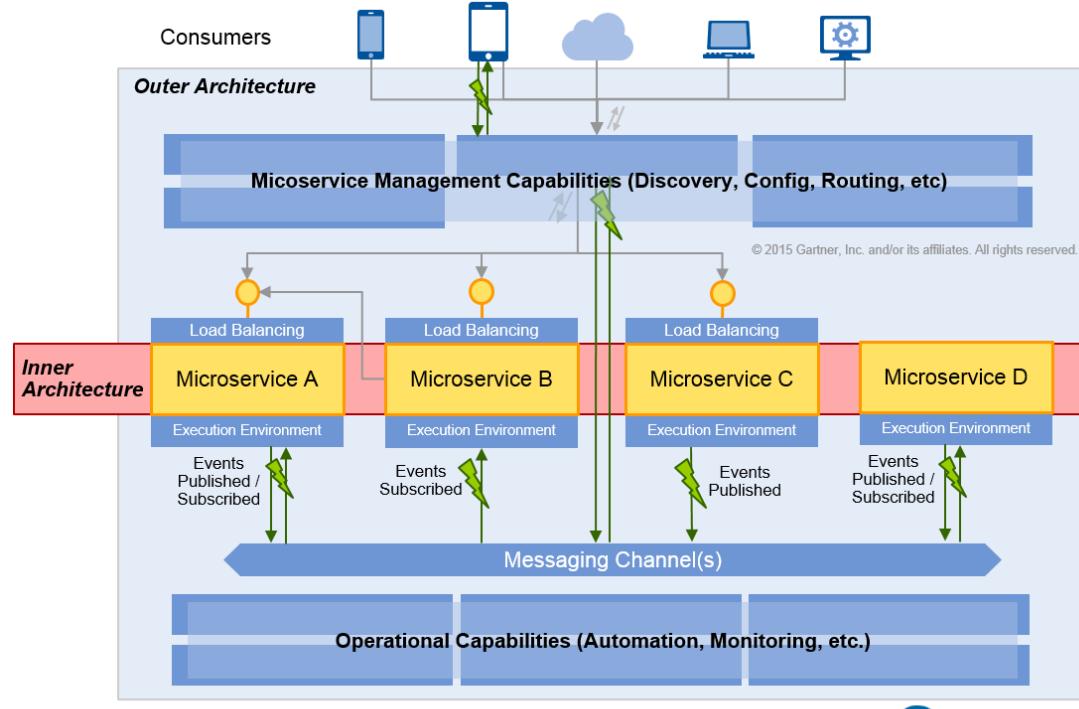
- Service Mesh는 마이크로서비스간 통신(네트워크)을 위한 어플리케이션 레벨의 인프라 계층
- 통신 및 네트워크 기능을 비즈니스 로직과 분리한 네트워크 통신 인프라
- 서비스 메시에서의 호출은 마이크로서비스 개별 인프라 계층의 'Sidecar'라고 하는 Proxy를 통해 수행



Sidecar



Service Mesh Coverage in MSA



Inner Architecture:

개별 마이크로서비스 영역으로 가능한 간결, 명확하고 단순하게 구현하는데 중점

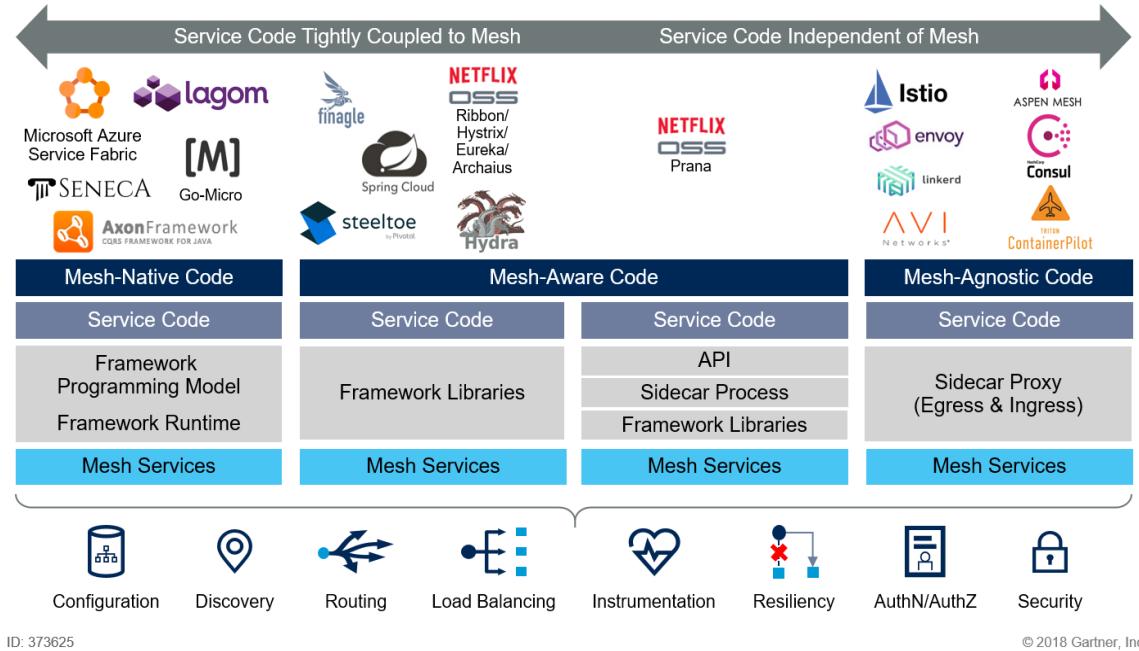
Outer Architecture:

- 마이크로서비스 외부를 아우르는 영역으로 트래픽 통제 및 서비스간 어떻게 커뮤니케이션 하는가 등, 클라우드 담당자가 공히 겪게 되는 복잡성들 (e.g. 서비스 발견, 탄력성, 장애 내선, 등)을 네트워크 레벨에서 처리

(Service Mesh 영역)

Service Mesh Types

A Spectrum of Service Mesh Technologies Based on Code Coupling



✓ 코드 기반

프레임워크 기반 프로그래밍 모델로 서비스 메시지를 구현하는데 특화된 코드가 필요한 유형

✓ 라이브러리 기반

라이브러리로 구현되어 API호출을 통해 Service Mesh에 결합되는 유형

✓ Proxy 기반

Istio/ Envoy, Consul, Linkerd 등 Sidecar Proxy를 통해 마이크로서비스에 주입하는 유형

Service Mesh Comparison

Aspect	Sidecar Proxy (Istio/Envoy)	Library (Spring Cloud & Netflix OSS)
Application Language	<ul style="list-style-type: none">Polyglot	<ul style="list-style-type: none">Java
Architecture	<ul style="list-style-type: none">사이트카 패턴 (uses Envoy as proxy)	<ul style="list-style-type: none">애플리케이션에 라이브러리 추가
Implementation Level	<ul style="list-style-type: none">Platform (using YAML)	<ul style="list-style-type: none">Application (using Code)
Service Discovery	<ul style="list-style-type: none">Implicit (자동 등록)	<ul style="list-style-type: none">Explicit (명시적 선언)
Circuit Breaker	<ul style="list-style-type: none">YAML Configuration	<ul style="list-style-type: none">Coding/ Annotation
Tracing	<ul style="list-style-type: none">Envoy Proxy들을 통한 호출 추적	<ul style="list-style-type: none">추적 데이터를 명시적으로 로깅
Security	<ul style="list-style-type: none">Service to ServiceMutual TLS through Envoy	<ul style="list-style-type: none">Application Specific

Istio Service Mesh key features

- Improved Routing & Deployment Strategy
- Improved Smart Resilience
- Improved Security
- Improved Observability

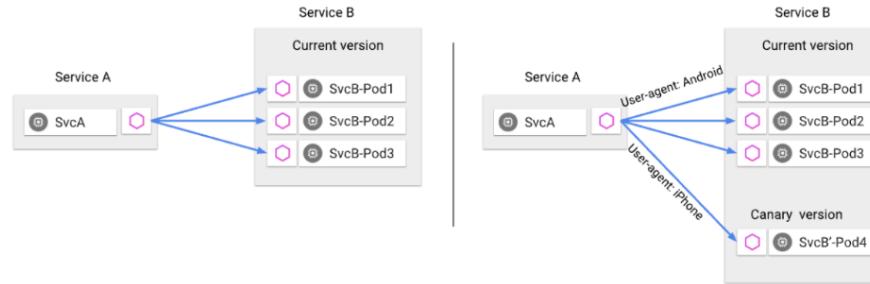


- ✓ Service Discovery
- ✓ Load balancing
- ✓ Dynamic Request Routing
- ✓ Circuit Breaking
- ✓ 암호화 (TLS)
- ✓ 보안
- ✓ Health check, Retry and Timeout
- ✓ Metric 수집



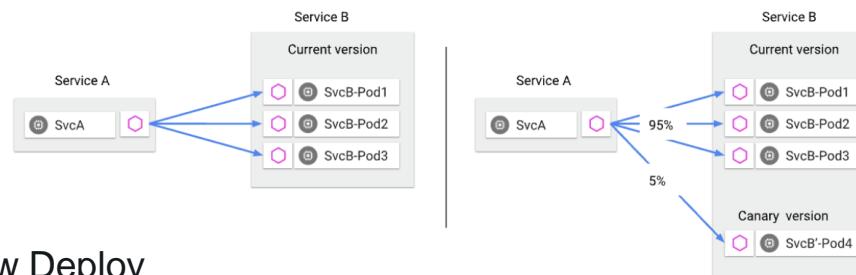
1. Improved Routing & Deployment Strategy

- Traffic Routing Controls



- Canary Deploy

- 특정 유저의 신상, 지역, 권한, 접근 단말에 따른 다른 버전의 노출



- A/B Testing, Shadow Deploy

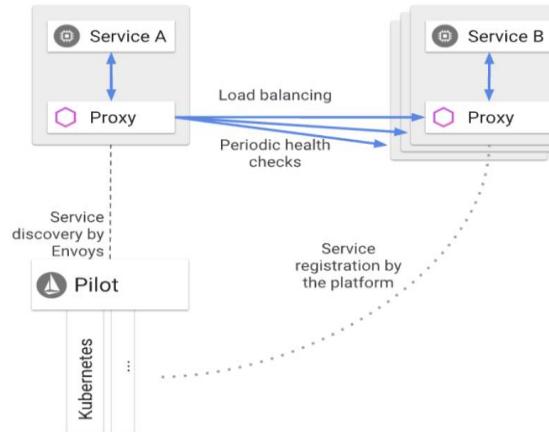
- 신규 버전의 오류 노출 없는 실질적 테스트

2. Improved Resilience

- Retry
 - 서비스간 호출의 실패에 대한 재시도
- Circuit Breaking / Rate Limiting
 - 인스턴스의 보호
 - 전체 서비스 장애 차단
- Pool Ejection
 - 죽은 인스턴스의 제외
- Health Check & Service Discovery
 - 여러 인스턴스에 대한 로드 밸런싱
 - 서비스 앤드 포인트 관리



Circuit Breaking + Pool Ejection + Retry
= High Resilience

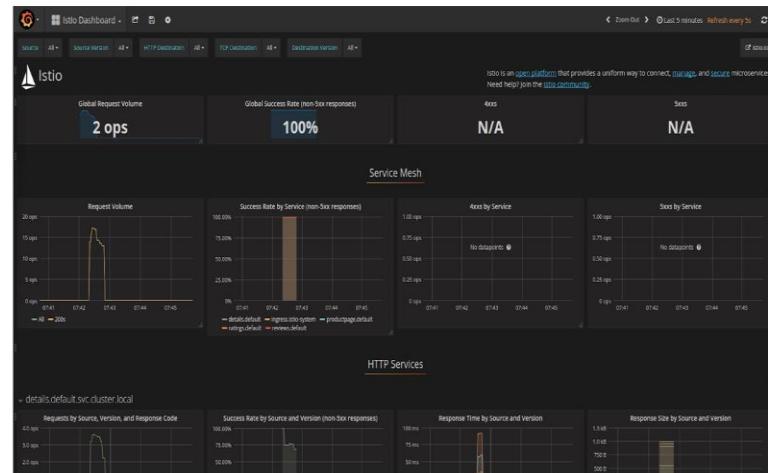
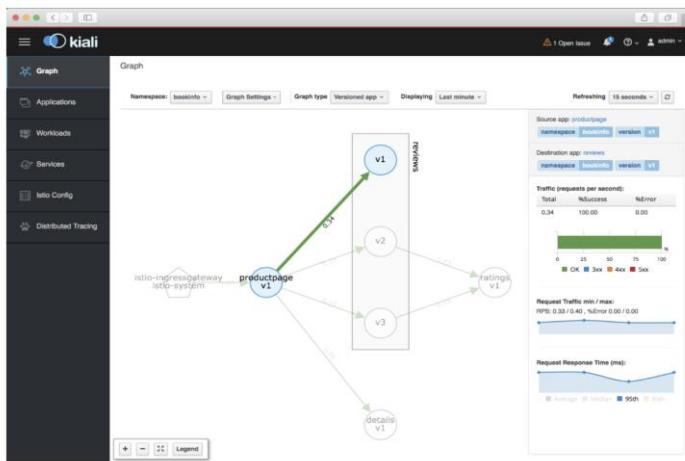


3. Improved Security

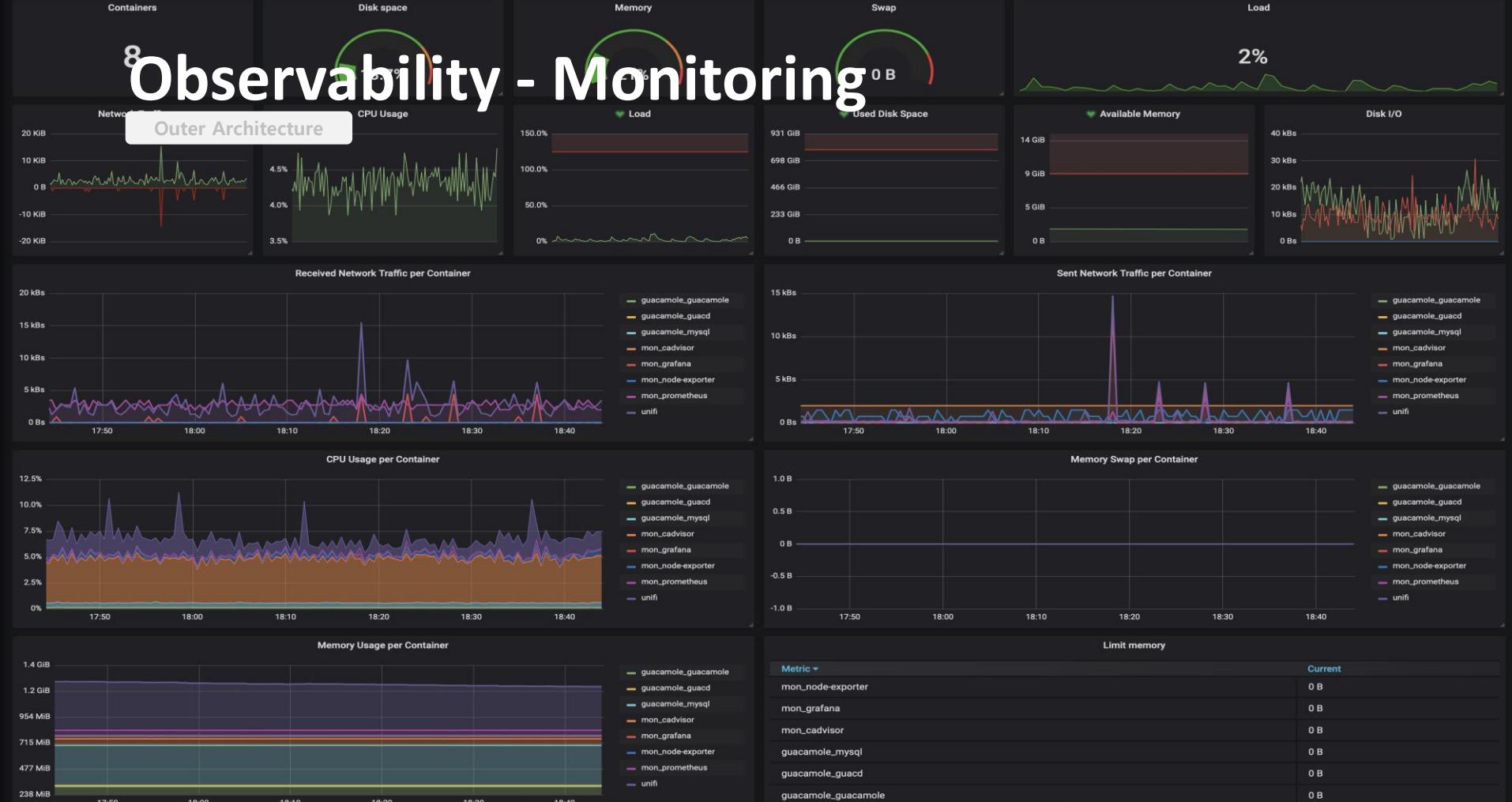
- TLS based Inter Microservices Communication
 - Envoy로 통신하는 모든 트래픽을 자동으로 TLS를 이용해 암호화
- Service Authentication & Authorization
 - JWT Token을 이용한 서비스 접근 Client 인증
 - 역할기반(RBAC) 권한 인증 지원
- Whitelist and Blacklist

4. Improved Observability

- Distributed Tracing and Measure
 - 서비스간 호출 내용 기록
 - Istio 설치 시, Kiali, Jaeger가 default로 설치되어 각각 Monitoring 및 Tracing 지원

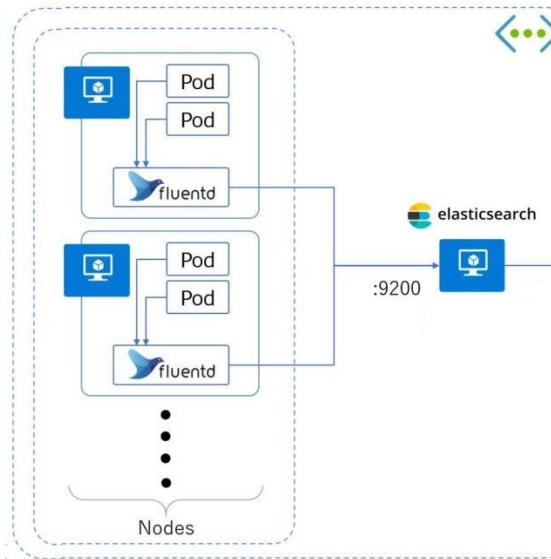


8 Observability - Monitoring



Observability – 통합 로깅

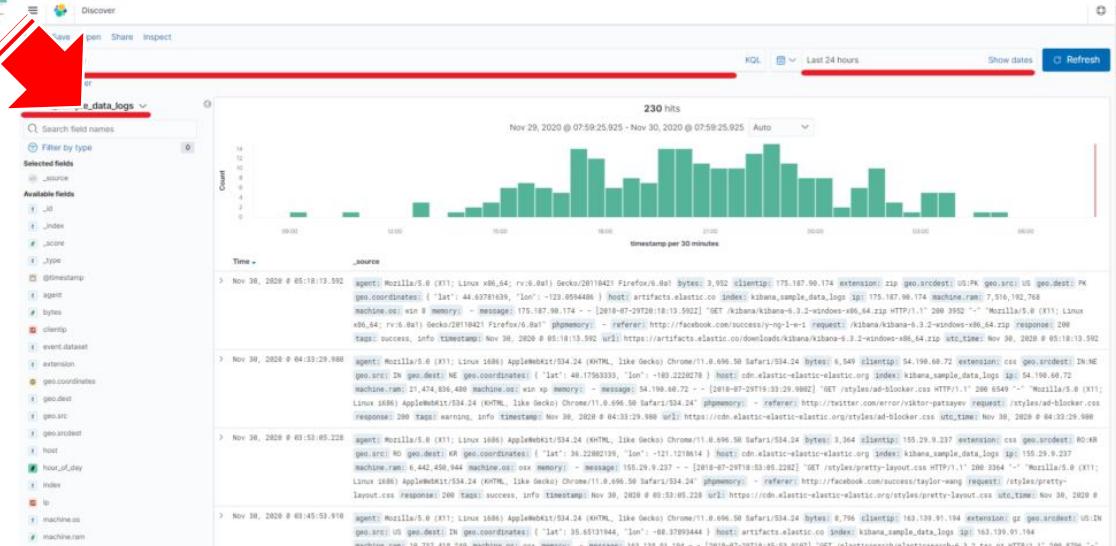
Outer Architecture



 Grafana loki

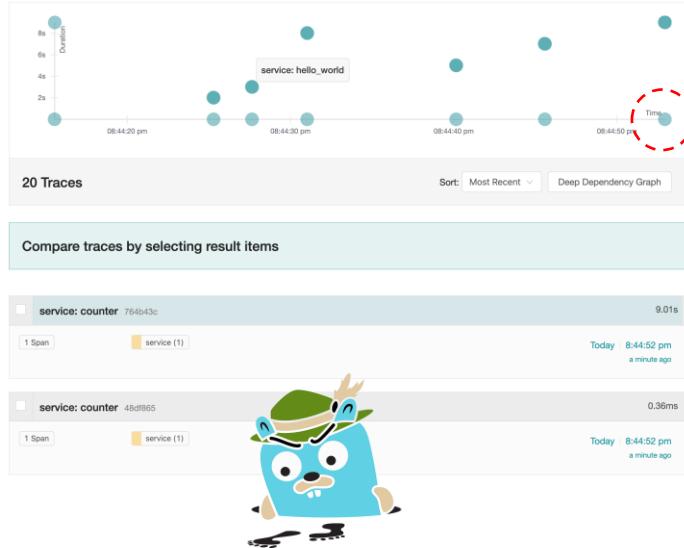
EFK, or PLG(Promtail, Loki, Grafana)

Loggregation = Log + Aggregation



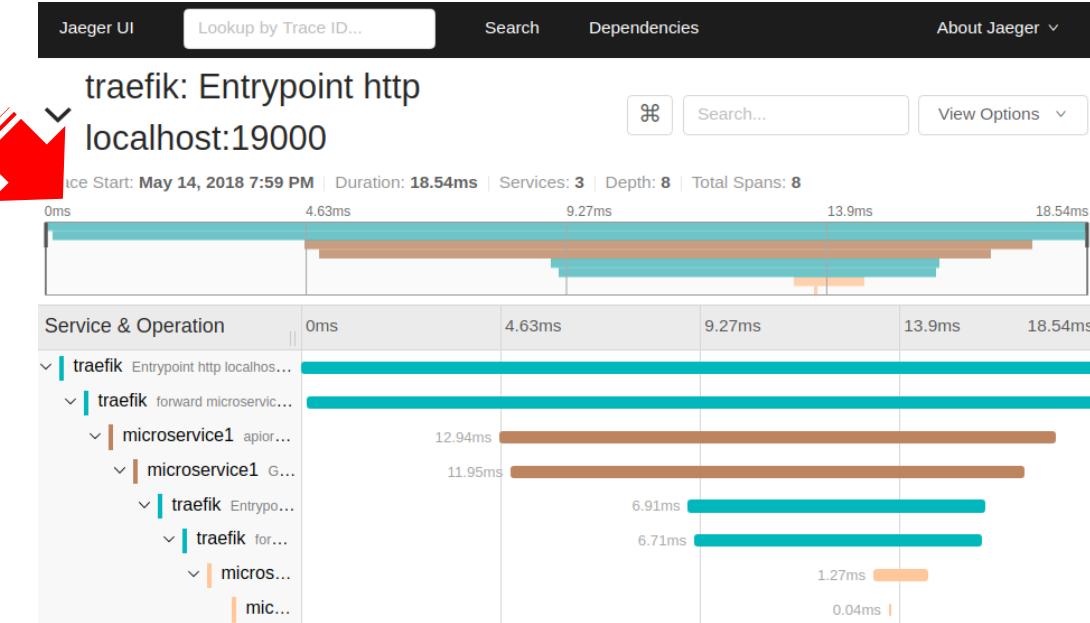
Observability - Tracing

Outer Architecture

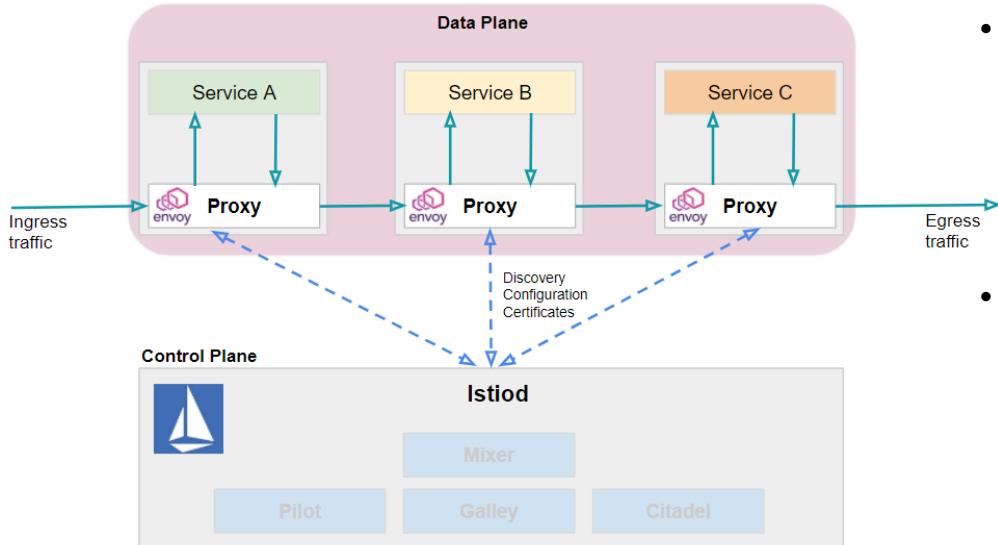


Zipkin, or Jaeger Stack

Tracing between Inner Architectures



Istio Service Mesh architecture



• Data Plane(데이터 플레인)

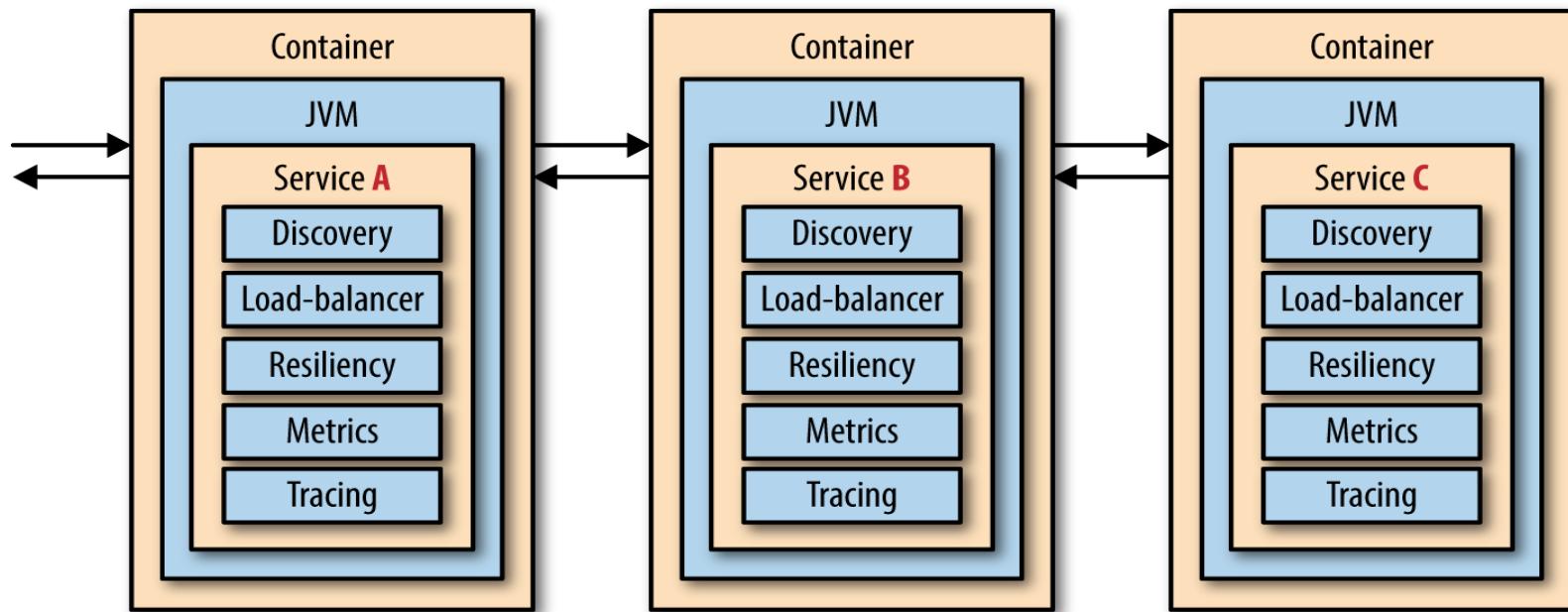
- 서비스 옆에 사이드카(Envoy)를 붙여, 서비스로 들어오고 나가는 트래픽을 Envoy를 통해 통제
- 유입/유출 트래픽 통제 : Ingress/Egress Controller

• Control Plane(컨트롤 플레인)

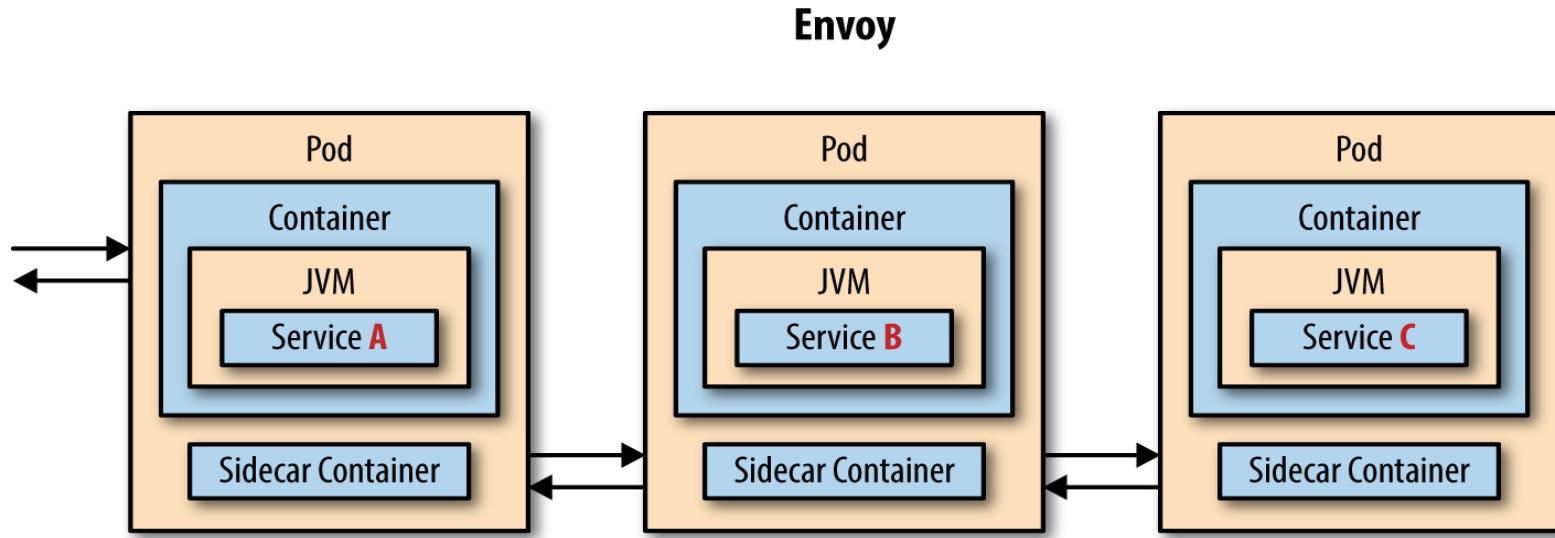
- **Pilot(파일럿)** : Envoy에 대한 설정 관리 및 서비스 디스커버리 기능 제공
- **Mixer(믹서)** : 액세스 컨트롤 및 다양한 모니터링 지표수집 (Pilot으로 대체됨)
- **Citadel(시터덜)** : 보안관련 기능 담당 모듈로 사용자 인증/인가 및 TLS 통신을 위한 인증서 관리
- **Galley(갤러리)** : Istio Configuration의 유효성 검사

Before Istio : Spring Cloud + Netflix

Before Istio



After Istio



좋은점 :

1. TCP Proxy가 아닌 L7 레이어 Proxy를 사용, 다양한 메쉬기능 지원 및 높은 성능
2. Code 변경 없이 Cross-cutting 이슈를 핸들링
3. Main 서비스의 재배포 없이 Sidecar로 관리

Lab. Istio Install

- Istio 설치

- export ISTIO_VERSION=1.18.1
- curl -L https://istio.io/downloadIstio | ISTIO_VERSION=\$ISTIO_VERSION TARGET_ARCH=x86_64 sh -
- cd istio-\$ISTIO_VERSION
- istioctl install --set profile=demo --set hub=gcr.io/istio-release

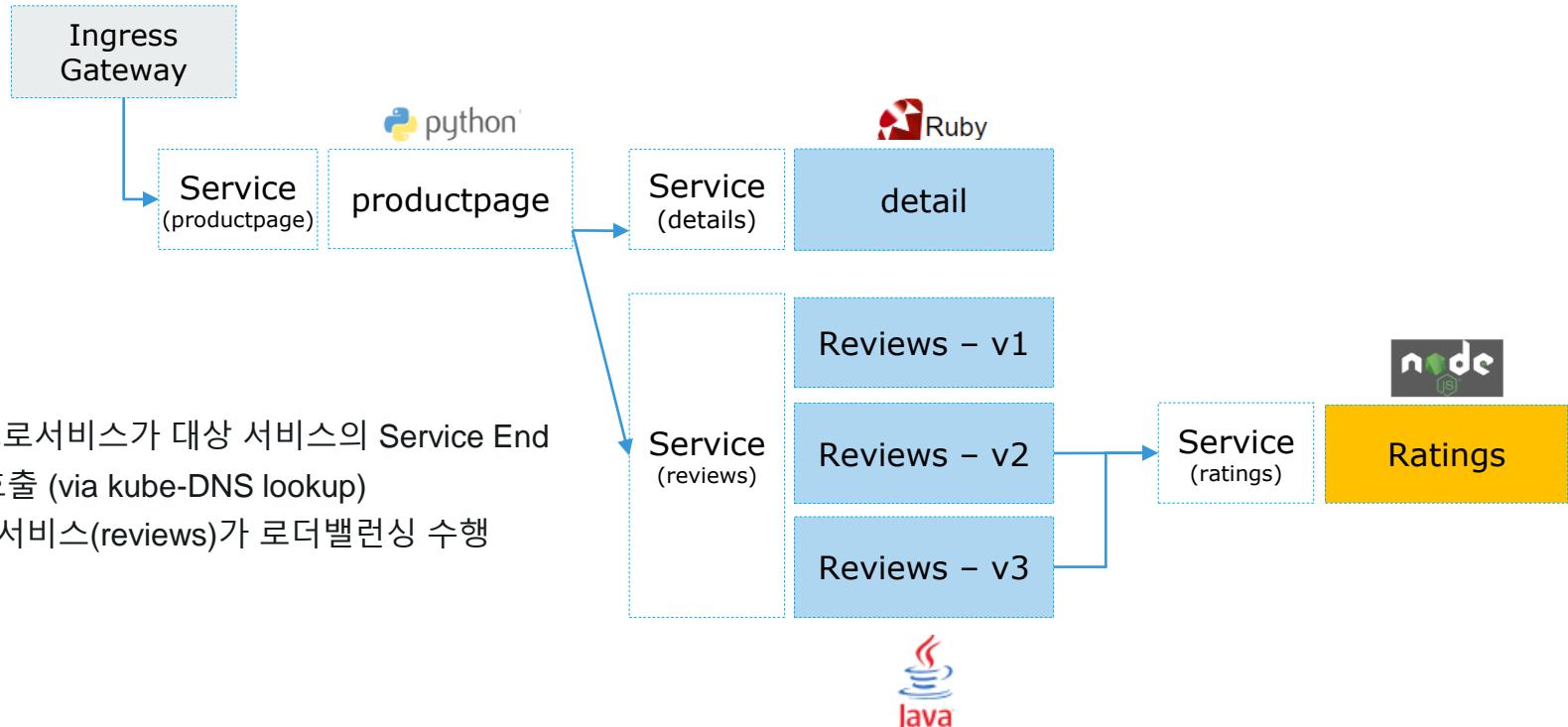
- 설치확인

- \$ kubectl get pod -n istio-system

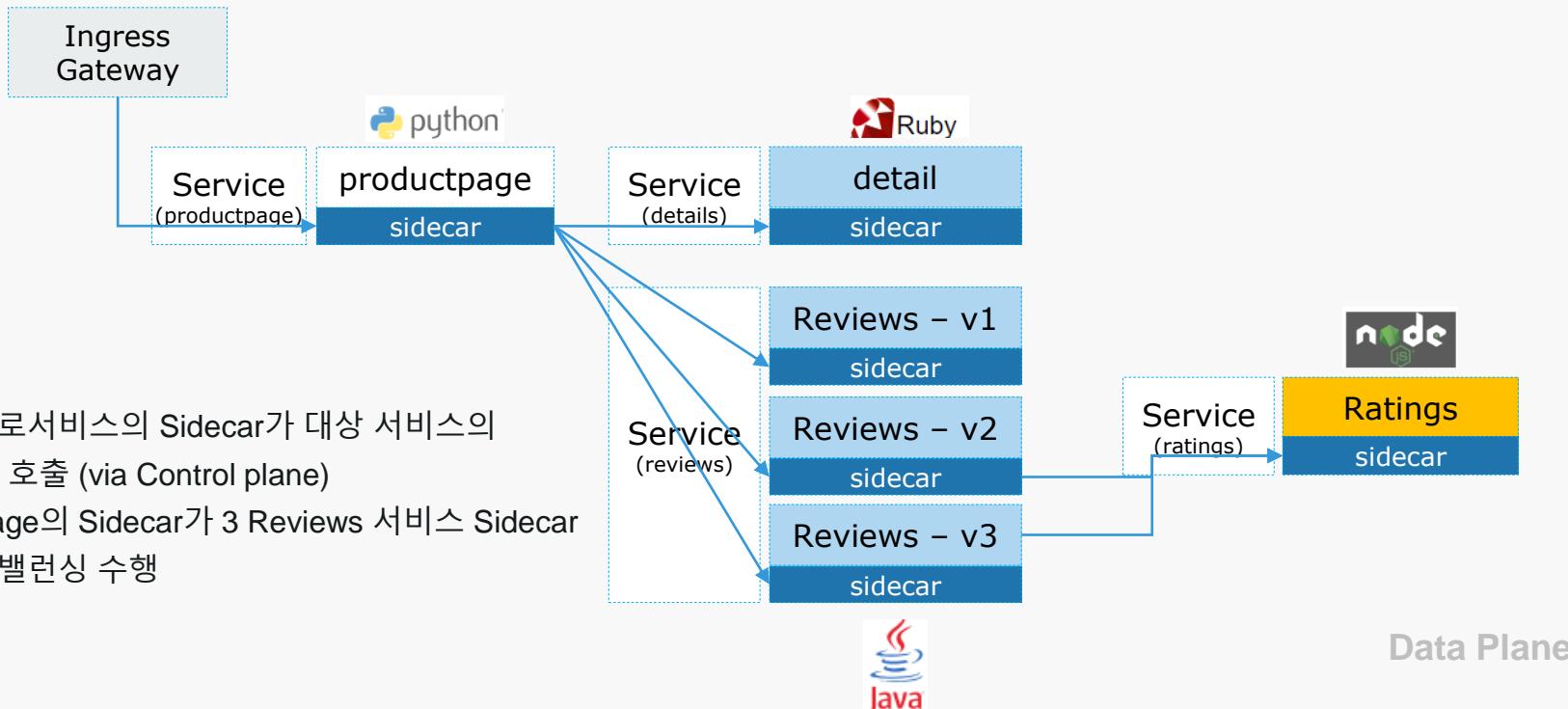


```
apexacme@APEXACME:~/istio-1.4.5$ kubectl get pod -n istio-system
NAME                               READY   STATUS    RESTARTS   AGE
grafana-6bb6bcf99f-78p9q           1/1    Running   0          76s
istio-citadel-597559bf4-lpqxc     1/1    Running   0          75s
istio-egressgateway-5c68d8857-qp14x 0/1    Running   0          76s
istio-galley-769849c5fb-cpvkp      0/1    Running   0          76s
istio-grafana-post-install-1.4.5-frqk4 0/1    Completed  0          78s
istio-ingressgateway-665c7c6b8-k9qhg 0/1    Running   0          76s
istio-pilot-6bd54f544b-65tnf       1/2    Running   1          75s
istio-policy-6b7d856769-xcip2      2/2    Running   2          75s
istio-security-post-install-1.4.5-p9wtc 0/1    Completed  0          78s
istio-sidecar-injector-6d9f967b5-v157x 1/1    Running   0          75s
istio-telemetry-657bbb5677-pg2th     2/2    Running   2          75s
istio-tracing-56c7f85df7-71lkd8     1/1    Running   0          75s
kiali-7b5c8f79d8-dm46s             1/1    Running   0          76s
prometheus-74d8b55f54-r9w7v        1/1    Running   0          75s
apexacme@APEXACME:~/istio-1.4.5$
```

Sample Application – ‘Book info’ (Without Istio)



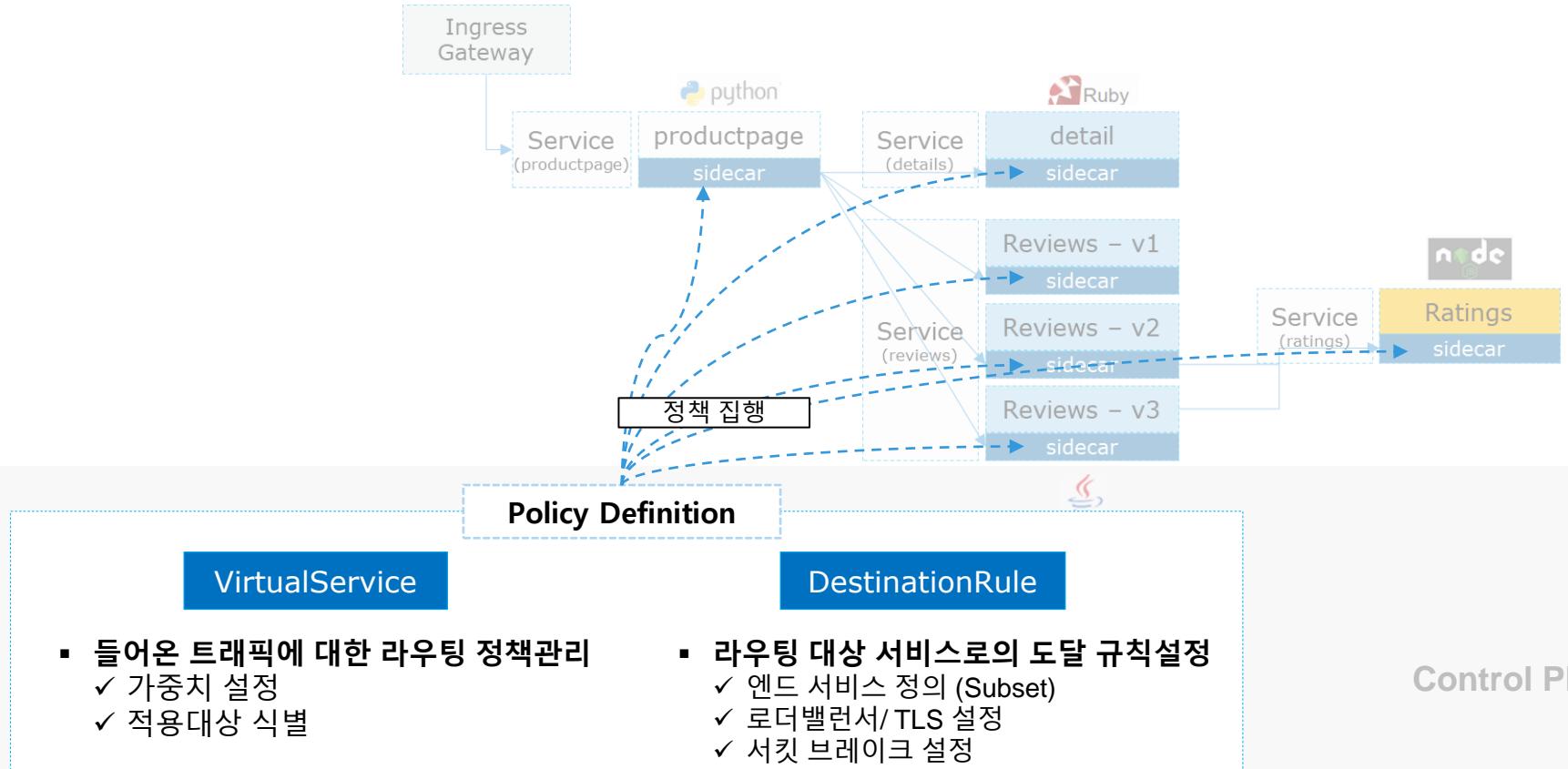
Sidecar injected 'Book info' model



- 각 마이크로서비스의 Sidecar가 대상 서비스의 Sidecar를 호출 (via Control plane)
- Productpage의 Sidecar가 3 Reviews 서비스 Sidecar로의 로더밸런싱 수행

Data Plane

CRDs for Policy Management of Sidecar



“

Service Reliability Engineering

- Istio based telemetry
- About SRE
- Error Budget & SRE Monitoring

SRE 등장 배경

- 개발팀은 시간내 기능 출시를 위해 개발에 무게를 두고, 운영팀은 시스템 안정성을 더 중시
- 개발팀의 무리한 배포로 인한 장애를 최소화 하고, 안정적인 서비스 운영을 위한 운영팀간의 균형있는 운영을 위한 DevOps 실전 철학
- SRE의 역할은 DevOps가 원활하게 운영되기 위한 자동화되고 모니터링 가능한 플랫폼을 지원하는 것이 목표

“*DevOps가 개발과 운영의 사일로(분단) 현상을 해결하기 위한 방법론이자,
하나의 조직문화에 대한 방향성이라면,*

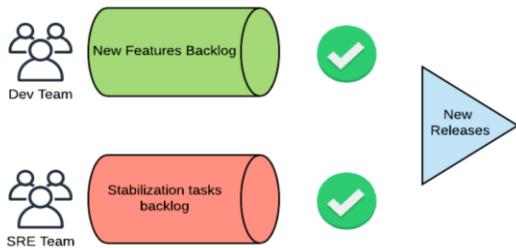
SRE는 DevOps가 정착되기 위한 구체적인 프랙티스(실사례)와 가이드”

– Google –

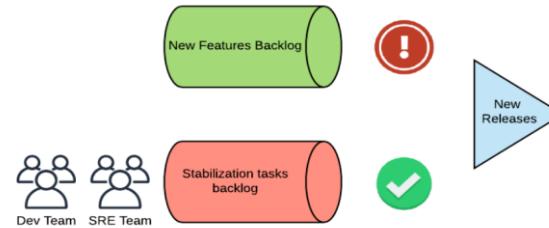
SRE 동작 방식

- **SRE : Site Reliability Engineering**, 서비스 운영에 대한 소프트웨어 엔지니어링 접근 방식
- 서비스 수준 지표(SLI) 및 서비스 수준 목표(SLO)를 사용해 서비스 수준 계약(SLA), 즉 마이크로서비스에 요구되는 신뢰성을 정의함으로써 새로운 기능의 계속적인 출시 여부를 결정
- 서비스가 SLA 수준 내에서 실행 중이면 개발 팀은 언제든지 새 버전 출시 가능, 하지만 SLA 수준에 못 미치는 경우, SLA가 충족될 때까지 신규 버전 출시 불가 → 사이트 신뢰성 향상에 총력

Team meeting the SLA



Team does not meeting the SLA



SRE 측정 기준

- 첫번째, 가용성에 대한 명확한 정의 (Service-Level Indicator, SLI)
- 두번째, 가용성 목표 정의 (Service-Level Objective, SLO)
- 세번째, 장애 발생에 대한 계획 (Service Level Agreement, SLA)



SLIs | Metrics that you use to define the SLO targets



SLOs | Targets you set for the overall health of your services



SLAs | Promises you make about your service's health

SRE 측정 기준

- **SLI (Service-Level Indicator)** : 서비스 수준 측정을 위한 정량적 지표
 - 예시 : 응답시간(Request Latency), 에러율(Error Rate), 처리량(Throughput), 가용성(Availability), etc
 - 모니터링 시스템을 통해 수집 주기, 수집범위를 정해 이를 지표화하고 시각화
- **SLO (Service-Level Objective)** : 지표에 대한 목표값
 - $\text{SLO} = \text{SLI} + \text{목표값(Goal)}$
 - 예시 : “매주, 99% of REST API 호출의 응답 시간은 100ms 이하여야 한다.”에서 “매주, 99% REST API 호출 응답시간”이 SLI가 되고 응답시간 100ms가 목표값이 되어 위의 전체 문장이 SLO가 됨
 - 단순하면서 완벽한 값을 사용하지 말고 가급적 적은 수의 SLO만 정의

Error Budget이란?

- 가용성에 따라서 허용되는 장애 시간
- **Error budget = [100% - availability target]**
 - 예를 들어, 한달에 SLO가 99.999%를 목표치로 설정했다면, 한달간 SLO는 0.001%의 다운 타임을 허용하게되고, 이 0.001%가 Error budget이 된다.
- Error budget 활용
 - Error budget 은 계획된 다운 타임이나, 장애 등에 의한 계획되지 않은 다운 타임에도 차감
 - 만약 Error buget이 없다면 새로운 기능에 대한 업데이트를 중지하고, 시스템 가용성을 높이는 자동화나 프로세스 개선 등의 작업을 수행

– Google –



Lab Time

Lab. SRE Monitoring

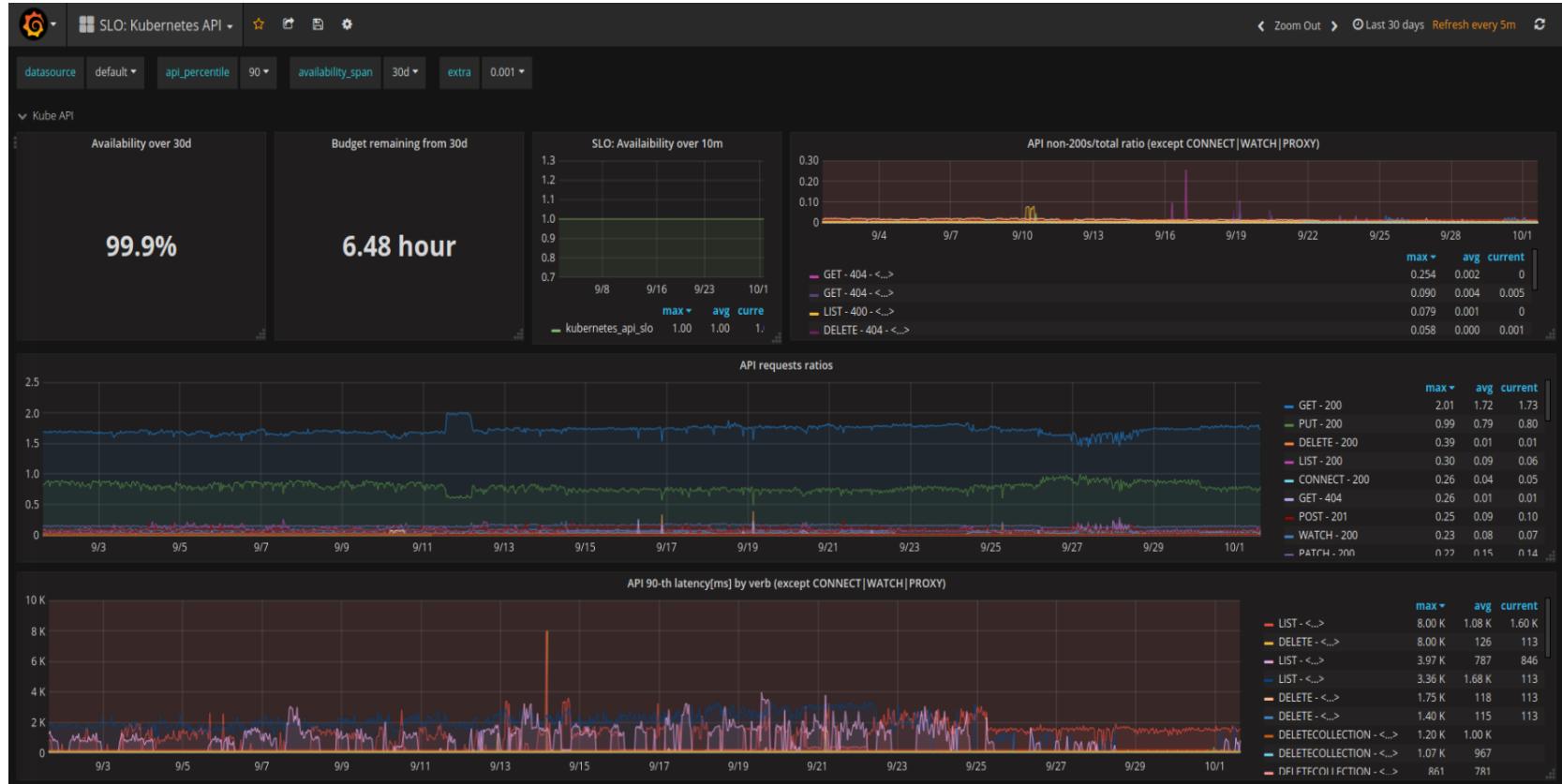


Table of contents

- Cloud Native Application Definition & Design Strategies
- The Domain Problem : A Commerce Shopping Mall
- Architecture and Approach Overview
- Domain Analysis with DDD and Event Storming
- Service Implementation with Spring Boot and Netflix OSS
- Monolith to Microservices
- Front-end Development in MSA
- Service Composition with Request-Response and Event-driven
- Application Packaging and Consumer Driven Contract Test
- Improving SLA with Container Orchestrator (Kubernetes)
- Service Mesh and Service Reliability Engineering
- CI/CD and Progressive Deployment Strategy with Service Mesh ✓

“

마이크로서비스 배포 자동화

- DevOps Process & Container-based Pipeline
- CI/CD Pipeline tool 소개

What's CI/CD

- Continuous Software Development Life Cycle in DevOps
- CI/CD 파이프라인은 새 버전의 소프트웨어를 제공하기 위해 수행해야 할 일련의 단계
- 개발자가 코드를 지속적으로 배포할 수 있어야 하는 필요성
- 초기 단계에서 버그를 감지하고 빈번한 코드 커밋으로 인한 이슈 방지
- CI/CD는 어플리케이션을 보다 짧은 주기로 고객에게 제공하는 방법
- 어플리케이션의 통합 및 테스트 단계에서부터 제공 및 배포에 이르는 어플리케이션의 라이프사이클 전체에 걸쳐 자동화와 모니터링을 제공하는 프로세스들의 묶음을 의미



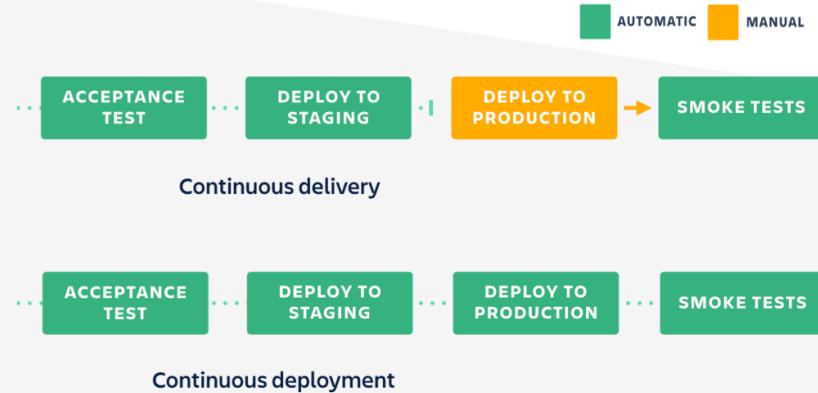
Continuous Integration

- 개발자를 위한 자동화 프로세스인 지속적인 통합(Continuous Integration)을 의미
- **개발코드를 통합할 때의 문제점을 해결하고, 자동화시켜 지속적으로 유지시키는 방법**
- 코드를 커밋에 따른 빌드, 통합, 테스트 과정의 자동화
- 성공적인 CI 를 위해서는..
 - 코드 저장소에 모든것을 넣어야 합니다.
 - 코드는 자주 병합되어야 합니다.
 - 매일 빌드를 성공적으로 실행해야 합니다.
 - 빌드 프로세스는 완전 자동화 되어야 합니다.
 - 빌드 실패시 바로 수정이 되어야 합니다.
 - 빌드에 번호가 매겨지고, 반복 가능해야 합니다.
 - CI 결과물로 만들어진 패키지 혹은 컨테이너는 신뢰 할 수 있어야 합니다.
(테스트 자동화 필수)



Continuous Delivery (Deployment)

- 지속적인 서비스 제공(Continuous Delivery) / 지속적인 배포(Continuous Deployment)
- 어플리케이션을 항상 신뢰 가능한 수준으로 배포 될수 있도록 지속적으로 관리
- **CI 가 이루어지고 난 후에 운영환경 까지 배포를 수행하여, 실제 사용자가 사용할 수 있도록 적용하는 단계**
- 성공적인 CD 를 위해서는..
 - ➔ 개발/스테이징/운영 환경에 동일한 배포 프로세스를 적용
 - ➔ 모든 환경에 동일한 패키지를 사용 (환경별 구성과 패키지를 다르게 유지해야 한다는 것을 의미)
 - ➔ 빠른 롤백이 가능하도록 구성
 - ➔ 모니터링 할 수 있는 도구 필요



Supporting Continuous Delivery

Facebook

- 배포 주기
 - 매일 마이너 업데이트
 - 메이저 업데이트 (매주 화요일 오후)
- Deployment Pipeline



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

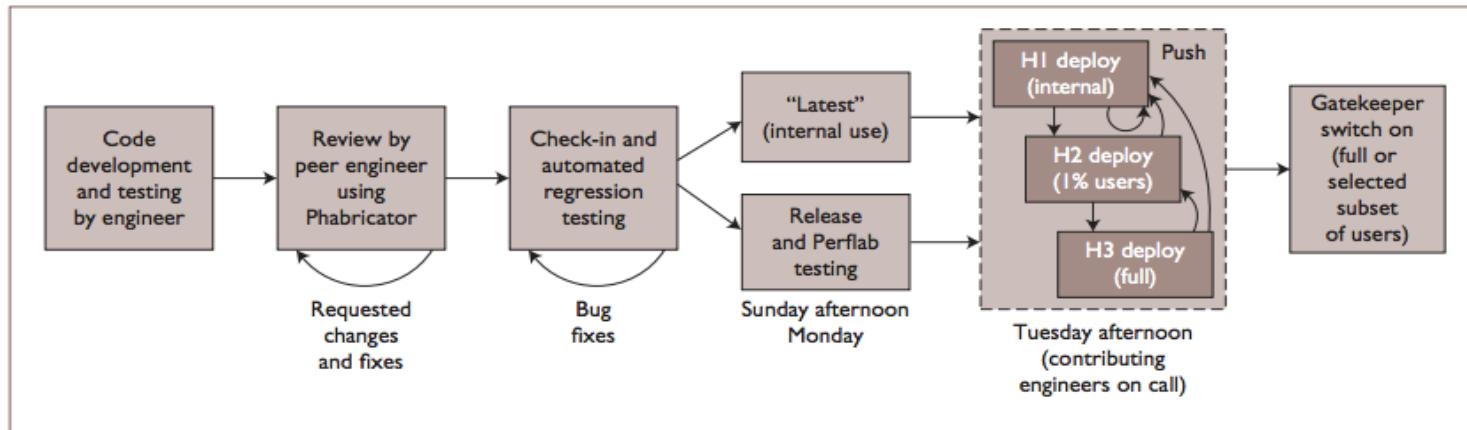


Figure 8. The Facebook deployment pipeline. Multiple controls exist over new code.

Supporting Continuous Delivery

Netflix

- Canary를 통해서 확신 갖기
 - Canary란? 실제 production 환경에서 small subset에서 새로운 코드를 돌려보고 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

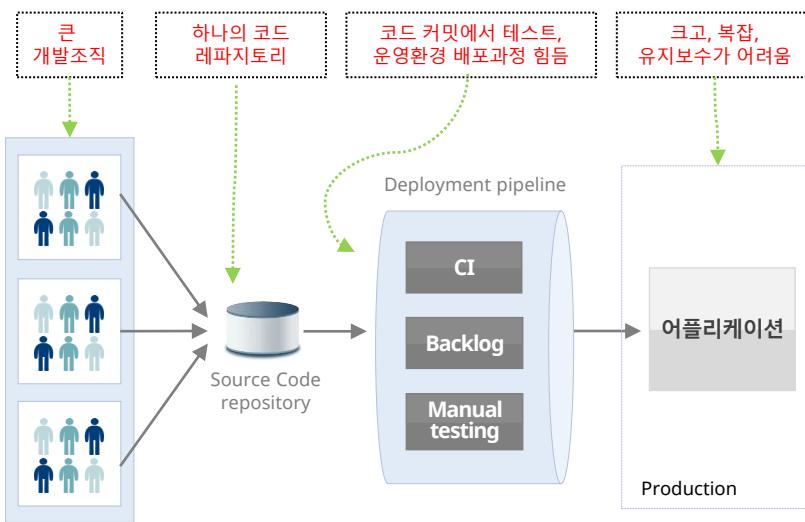


출처: <http://techblog.netflix.com/2013/08/deploying-netflix-api.html>

Process Change : 열차말고 택시를 타라!

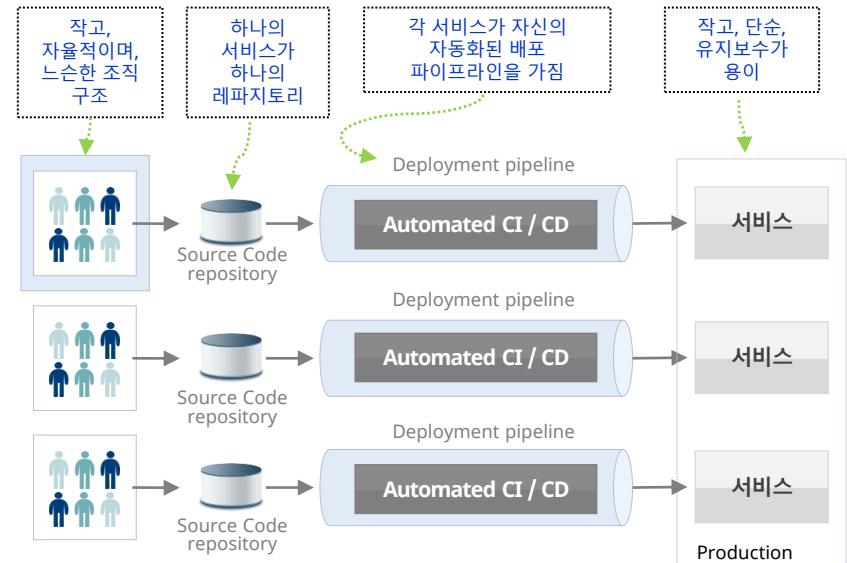
모노리식 개발 및 운영환경

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



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



CI/CD Open Source Tools



- ✓ 무료 및 오픈 소스로 연동 가능한 다양한 플러그인 (1,500개 이상) 제공
- ✓ AWS, Google Cloud, Azure, Digital Ocean 등 퍼블릭 클라우드 플랫폼과 통합
- ✓ 독립 실행형 Java 응용 프로그램이며 즉시 사용할 수 있는 .war 형식



- ✓ Kotlin 기반으로 Docker, Visual Studio Team Services, Maven, NuGet 등과 통합
- ✓ Google Cloud, AWS, VMWare vSphere 클라우드 플랫폼과 통합 가능



- ✓ CircleCI의 CircleCI Server는 GitHub Enterprise, LambdaTest, Coveralls 통합 지원
- ✓ AWS, Google Cloud, Azure 등 퍼블릭 클라우드 플랫폼 연동 지원



- ✓ CI/CD는 GitHub Enterprise 도구와 원활하게 통합되는 독점 YAML 구문을 사용
- ✓ AWS, Google Cloud, Azure 등 퍼블릭 클라우드 플랫폼 연동 지원



- ✓ Jira 소프트웨어 및 Bitbucket 서버와의 통합 기능 내장
- ✓ 최대 100개의 원격 빌드 에이전트와 에이전트에 대한 병렬 테스트 배치 지원

CNCF Projects



Continuous Integration & Delivery

CI/CD Tools - Cloud Vendors'



AWS CodeBuild

- ✓ CI/CD(지속적 통합 및 전달)를 위한 완전한 자동 소프트웨어 릴리스 워크플로를 생성 가능
- ✓ Jenkins 서버 설정에서 CodeBuild를 작업자 노드로 사용하여 빌드를 분산
- ✓ 빌드 볼륨에 따라 자동으로 확장 및 축소
- ✓ 빌드를 완료할 때까지 걸리는 시간(분)을 기준으로 비용이 청구

- ✓ Azure Pipeline에 있는 DevOps는 프로젝트 시작에서 기획, 작업관리, 모니터링 도구포함
- ✓ 부하 테스트를 포함하는 통합 테스트 및 모니터링 환경 제공
- ✓ DevOps를 위해 CI/ CD(Continuous Deployment(Delivery))를 분리하여 구축
- ✓ CI와 CD trigger 설정 및 파이프라인 자동화

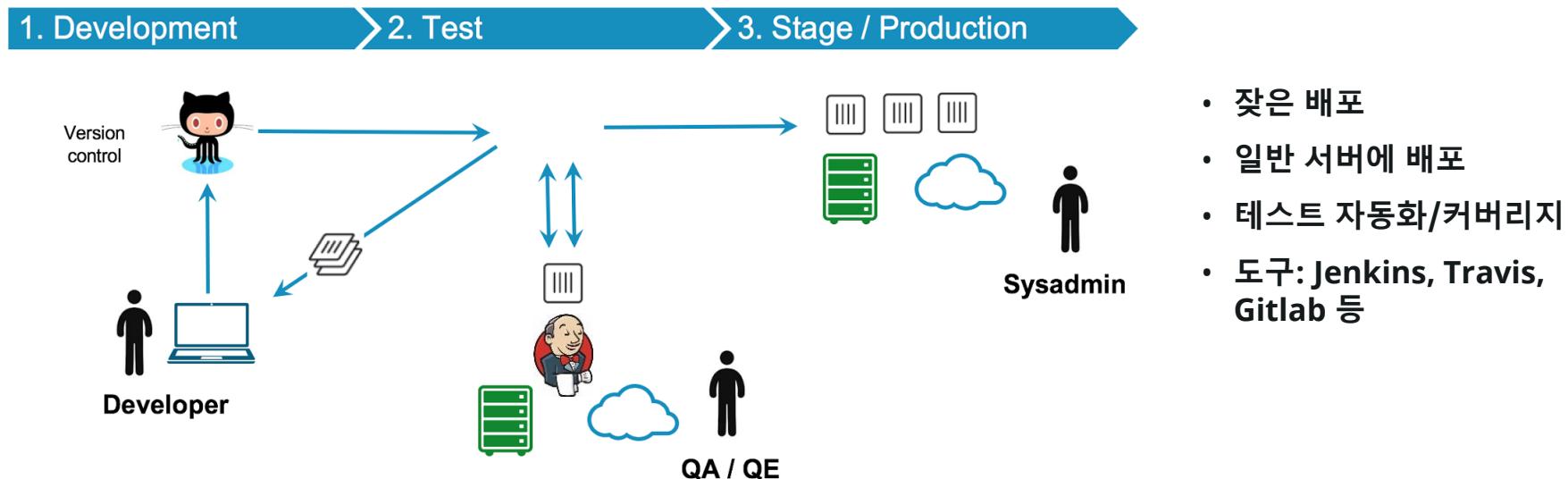


Google Cloud Build

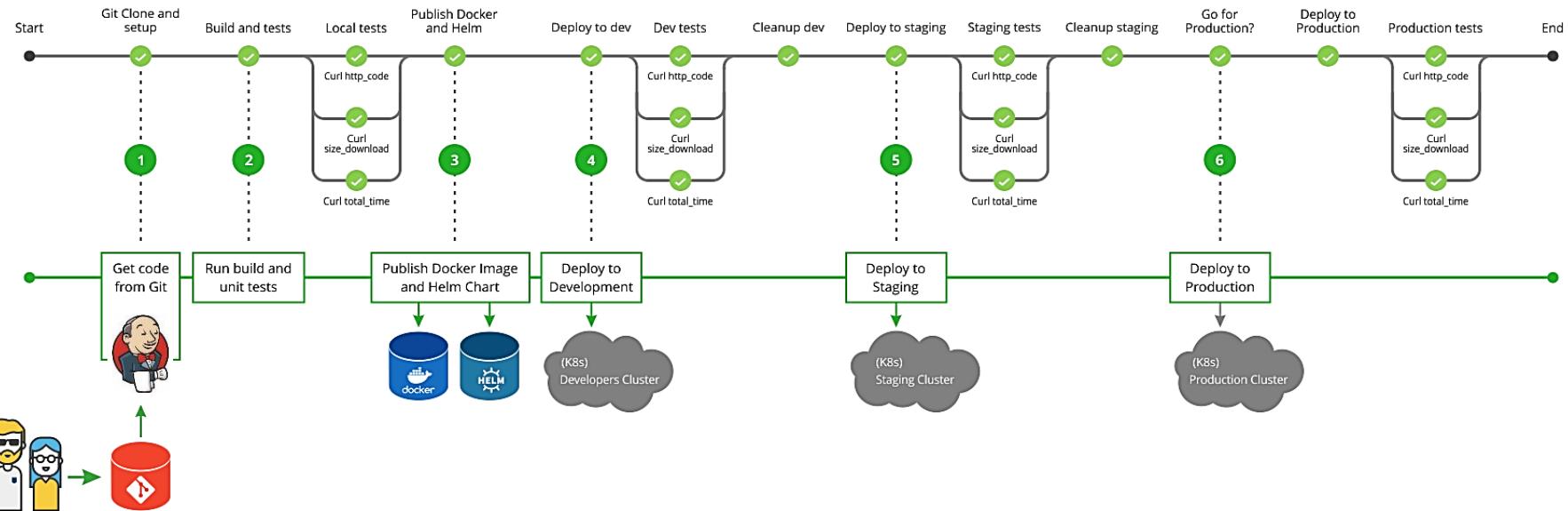
- ✓ Cloud Build는 Google Cloud Platform의 인프라에서 빌드를 실행하는 서비스
- ✓ 매일 120분의 무료 빌드와 최대 10회 동시 빌드 지원
- ✓ Docker, gradle, maven, bazel 같은 빌드 도구로 artifact 생성 가능
- ✓ 컨테이너 이미지 패키지 취약점 자동분석

CI/CD Pipeline : Conventional CI/CD

CI /CD Workflow



CI/CD Pipeline : Container-based



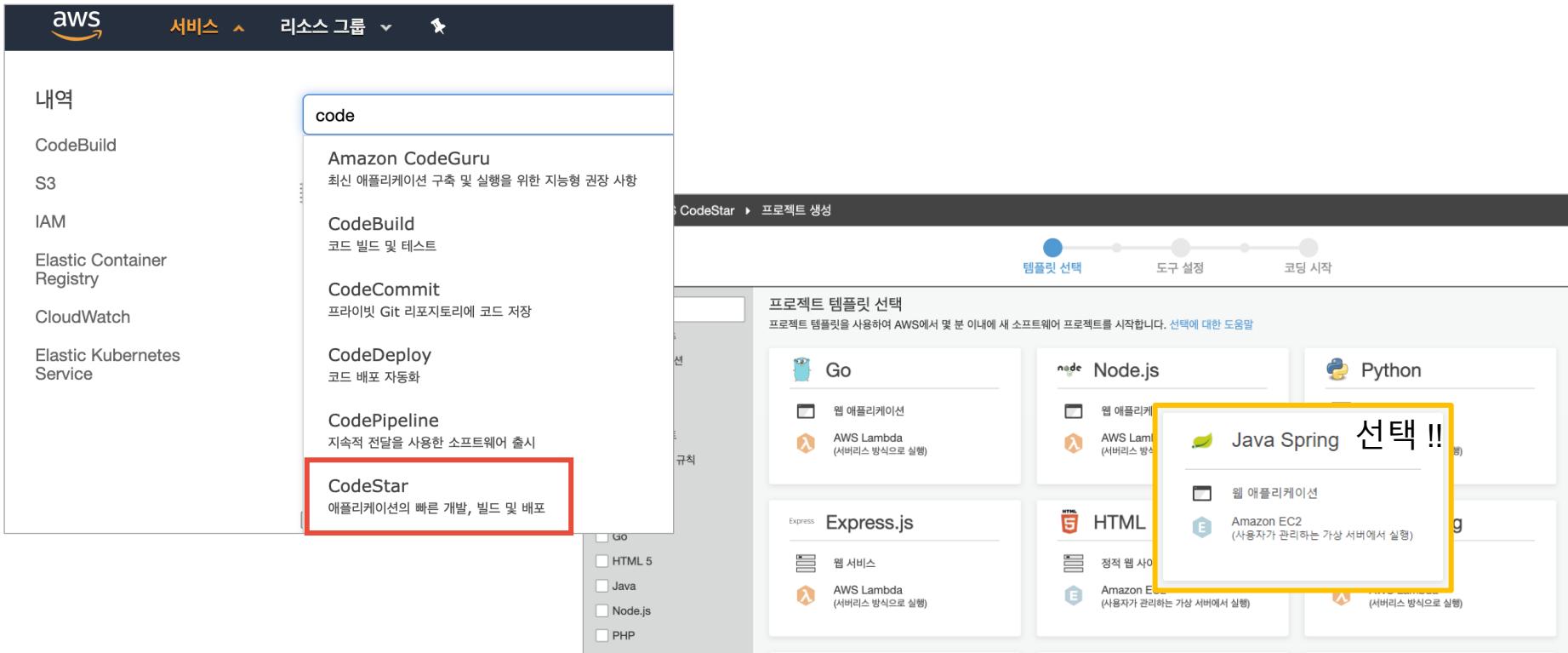
- API Testing
- Blue/Green, Canary
- 도구 : Jenkins + Docker + Argo CD + Canary + Kubernetes
→ 매우복잡

“

AWS Managed Service for CI/CD

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.

Project Creation Interface (Left):

- Progress bar: 템플릿 선택 (Completed), 도구 설정 (In Progress), 코딩 시작 (In Progress).
- Project Information:**
 - Project Name: sample-codestar
 - Project ID: sample-codestar
- Repository Options:**
 - 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.
- Repository Name:** sample-codestar
- Next Step Buttons:** 이전, 다음

Yellow Box Annotation: ◀ user00-codestar 입력 !!

Resulting Dashboard (Right):

- aws AWS CodeStar > sample-codestar
- Project Settings:** AWS CodeStar 프로젝트. **✓** 프로젝트가 성공적으로 생성되었습니다.
- File Addition:** 파일 추가
- Success Message:** sample-codestar에 오신 것을 환영합니다! 시작하기를 도와드립니다.
- Development Tools:** A screenshot of the AWS CodeStar dashboard showing various development tools and metrics.
- Yellow Box Annotation:** ◀ 개발툴 건너뛰기 및 설정완료

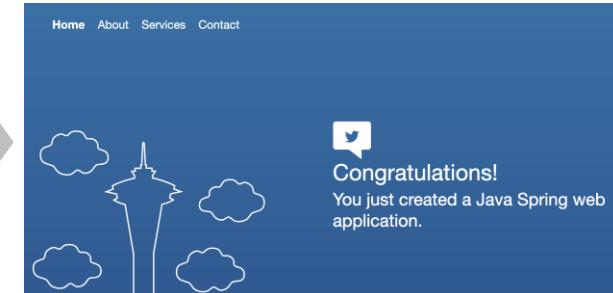
AWS DevOps – CodeStar

- 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:

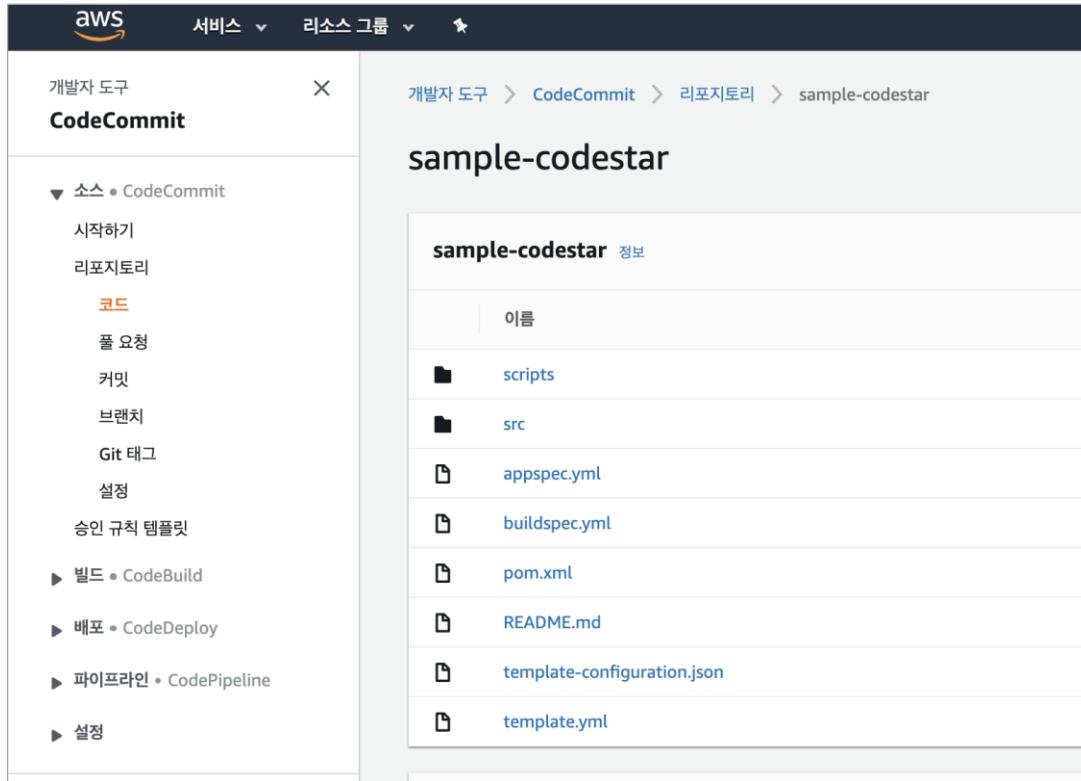
- Project Dashboard:** Shows a commit history for "sample-codestar" on the "master" branch. It lists two commits: "readme" by "kimscott" (1 minute ago) and "Initial commit by AWS CodeCommit" by "AWS CodeCommit" (1 hour ago). A "Deploy" button is visible.
- Application Activity Timeline:** A chart titled "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 View:** A vertical stack of three 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 type ('Compute Platform' selected), and target environment ('EC2/OnPremises'). A note indicates the application name must be 100 characters or less. The 'Create Application' button is visible 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 'Deployment Groups' tab for the 'sample-deploy' application. It lists one deployment group named 'Deployment Group'. A note at the bottom states that CodeDeploy requires a deployment group to be created before proceeding with the deployment. The 'Deployment Groups' button is highlighted.

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

개발자 도구 > CodePipeline > 파이프라인 > 새 파이프라인 생성

파이프라인 설정 선택

파이프라인 설정

파이프라인 이름
파이프라인 이름을 입력합니다. 생성된 후에는 파이프라인 이름을 편집할 수 없습니다.

sample-pipeline

100자를 초과할 수 없습니다

서비스 역할

새 서비스 역할
계정에 서비스 역할 생성

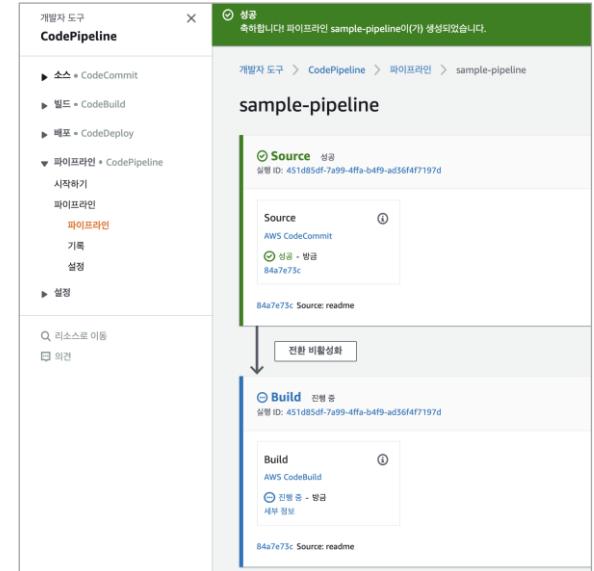
기존 서비스 역할
계정에서 기존 서비스 역할 선택

역할 이름
 AWSCodePipelineServiceRole-ap-northeast-1-sample-pipeline

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.

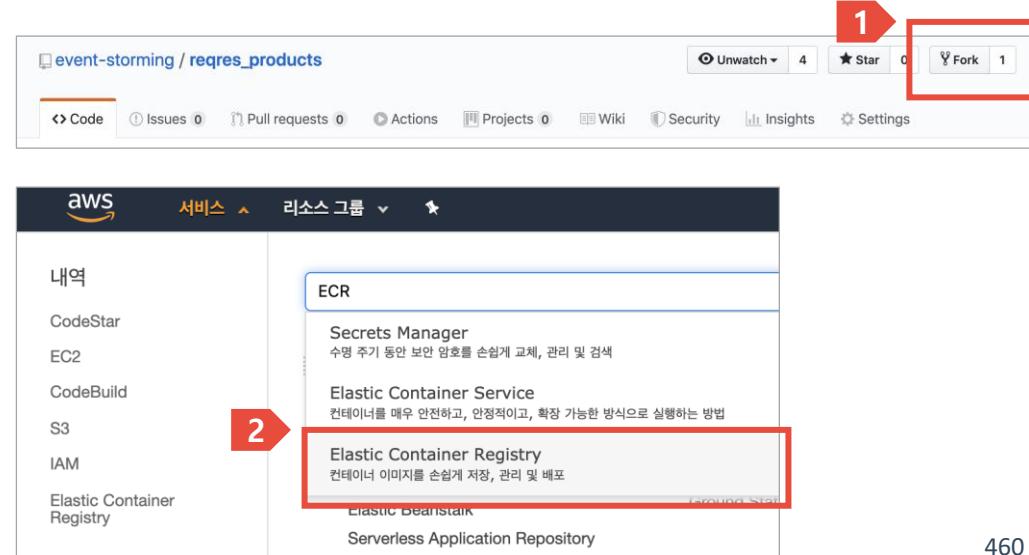
“

Let's Build a Microservice Pipeline

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/reqres_products
2. Docker image 를 저장하기 위한 공간인 ECR 에 “user00-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 : Amazon Linix 2
 - 런타임 : Standard
 - 이미지 : Standard4.0 (**이미지별로 환경이 다르니 주의**)
 - 환경유형 : Linux
 - 도커 권한 체크 필수

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

2

새 환경 이미지

관리형 이미지
AWS CodeBuild에서 관리하는 이미지 사용

사용자 지정 이미지
도커 이미지 지정

운영 체제

Amazon Linux 2

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 [\(를\) 참조](#) 하십시오.

런타임

Standard

이미지

aws/codebuild/amazonlinux2-x86_64-standard:4.0

이미지 버전

이 런타임 버전에 항상 최신 이미지 사용

환경 유형

Linux

권한이 있음

도커 이미지를 빌드하거나 빌드의 권한을 승격하려면 이 플러그를 활성화합니다.

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	
	AWS_ACCOUNT_ID	97905023****	일반 텍스트	제거
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 : 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: corretto8
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.com/ko_kr/codebuild/latest/usingguide/build-spec-ref.html

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

CodeBuild : ECR Connection Policy 설정

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 following details:

- 이미지: aws/codebuild/standard:2.0
- 환경 유형: Linux
- 제한 시간: 1시간 0분
- 서비스 역할: arn:aws:iam::979050235289:role/service-role/codebuild-sample-products-service-role

A red box highlights the '서비스 역할' field, and a red arrow labeled '1' points to it.

Below the environment configuration, there is a JSON editor:

```
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 highlights the JSON code, and a red arrow labeled '2' points to it.

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 : Run Pipeline (Docker Build/Push)

The screenshot shows the AWS CodeBuild buildspec.yml editor. The code content is as follows:

```
reqres_products / buildspec.yml Cancel  
Edit file Preview changes Spaces 2 No wrap  
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
```

Below the code editor is a "Commit changes" section:

- Update buildspec.yml
- Add an optional extended description...
- Commit directly to the `master` branch.
- Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

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

2 ECR에서 파이프라인으로 배포된 이미지 확인

The screenshot shows the AWS ECR console under the "products" tab. It displays a single image entry:

이미지 (1)	
<input type="checkbox"/> 이미지 찾기	
<input type="checkbox"/> 이미지 태그	이미지 URI
<input type="checkbox"/> 6dd5866a8971279f62376425d6f374e6fd682c9b	979050235289.dkr.ecr.ap-northeast-1.amazonaws.com/products:6dd5866a8971279f62376425d6f374e

CodeBuild : 빌드내역 확인

The image shows two screenshots illustrating how to check CodeBuild build logs.

Screenshot 1: Developer Tools - CodeBuild

- Developer Tools menu: CodeCommit, CodeArtifact, CodeBuild (selected), CodeDeploy, CodePipeline, Settings.
- CodeBuild section:
 - Source: CodeCommit
 - Artifacts: CodeArtifact
 - Build: CodeBuild (selected)
 - Start
 - Project Build
 - Build History (highlighted with a red dashed box)
 - Report Groups
 - Report Details
 - Metrics
 - Deploy: CodeDeploy
 - Pipeline: CodePipeline
 - Settings

Screenshot 2: Build History Detail View

Build History for build project sample-products:8f6dc424-0415-44f0-9cbc-0e2fab8195a8

Build Status	Started By	Build ARN	Resolved Source Version
In Progress	root	arn:aws:codebuild:ap-northeast-1:979050235289:build/sample-products:8f6dc424-0415-44f0-9cbc-0e2fab8195a8	-
Start Time	Last Run Time	Build Number	
2020-03-19T03:48:00+09:00	-	1	

Log tab selected. Log output:
Build log output is truncated at 1000 lines. To view the full log, click "Full Log View".

CodeBuild : Cache 적용

The screenshot shows the AWS CodeBuild console with the following steps highlighted:

- 1 S3 서비스에서 Bucket 생성**: A red box highlights the 'Amazon S3' dropdown menu under '캐시 유형' (Cache Type). The selected option is 'Amazon S3'. A red arrow points to the top-left corner of the box.
- 2 파일프라인 - Artifact 편집**: A red box highlights the 'mvn-cache-codebuild' entry in the '캐시 버킷' (Cache Bucket) input field. A red arrow points to the top-left corner of the box.

The interface includes a search bar for buckets and a note about naming conventions for cache buckets. At the bottom, there are '취소' (Cancel) and '아티팩트 업데이트' (Artifact Update) buttons.

1. Maven 으로 빌드시,
라이브러리를 캐쉬화 시킴
(캐쉬가 없으면 빌드때마다
1~2분 정도 더 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 에서 cache
부분 주석 해제 후 커밋

CodeBuild : VM에서 EKS Connection 설정 (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
API 서버 엔드포인트 https://A4CB98CD955A2E274B343BCEA284621F.yl4.ap-northeast-1.eks.amazonaws.com		
Amazon ECR		
리포지토리		

CodeBuild : VM에서 EKS Connection 설정 (2/4)

1. 쿠버네티스 토큰 생성은 ServiceAccount 생성 후 해당 SA에 cluster-admin 룰을 주면 된다.
2. WorkFlowy에서 “CodeBuild와 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.eyJpc3MiOiJrdWJlcmlldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pb9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOjJrdWJlLXN5c3RlbSIsImt1YmVybmv0ZXMuaW8vc2VydmljZWFlY29
```

CodeBuild : VM에서 EKS Connection 설정 (3/4)

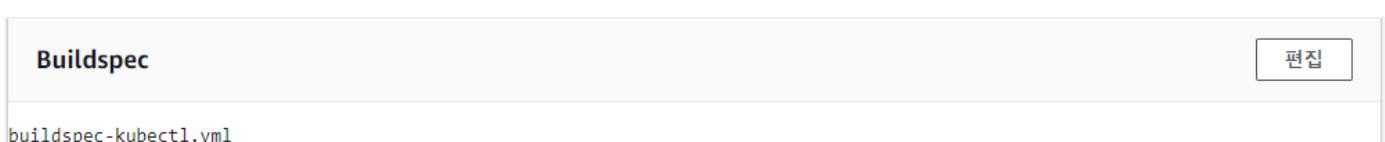
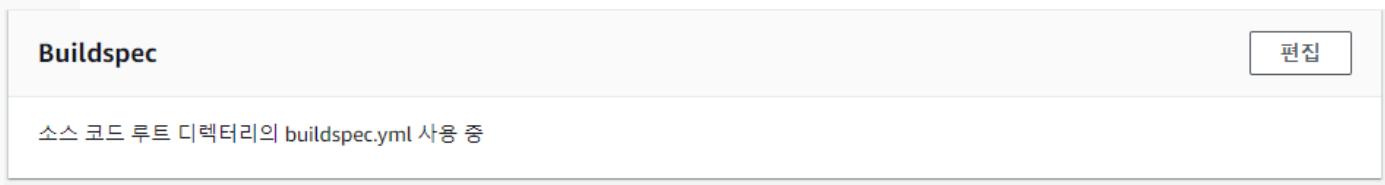
- 빌드 파일의 “환경” 영역 편집 버튼을 눌러, 아래 값을 넣은 후에 저장한다.
- 쿠버네티스 API 주소를 “KUBE_URL” 이라는 이름으로 저장한다.
- 클러스터 어드민 토큰 값을 “KUBE_TOKEN” 이라는 이름으로 저장한다.

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 : VM에서 EKS Connection 설정 (4/4)

1 파이프라인의 Buildspec을 편집하여, 레파지토리에 있는 buildspec-kubectl.yaml로 수정한다.

2 buildspec-kubectl.yaml 파일을 열어, _PROJECT_NAME: 을 내정보에 맞게 수정하고, 저장한다.



CodeBuild : 파이프라인으로 생성된 결과 확인

- ✓ CodeBuild가 자동 실행 되어,
EKS 클러스터에 배포 후
서비스가 정상적으로 기동
되었는지 확인한다.
- ✓ Kubectl get all

```
root@labs--1147635527:/home/project# kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/user30-products-5f4d77645f-2x86p        1/1     Running   0          45m

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
service/kubernetes   ClusterIP   10.100.0.1    <none>       443/TCP     5h15m
service/user30-products   ClusterIP   10.100.202.42  <none>       8080/TCP    45m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/user30-products  1/1     1           1          45m

NAME                           DESIRED  CURRENT  READY   AGE
replicaset.apps/user30-products-5f4d77645f  1        1        1      45m
root@labs--1147635527:/home/project#
```

AWS CodeBuild를 활용한 CI/CD 2nd Lab.



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-kubectl.yml 파일 내용 복사해 오기

4. Fork 받은 리파지토리의 Root에서 buildspec.yml을 편집하여 5행의 이미지명 수정
- 위에서 생성한 ECR Repository와 동일한 이름 입력

참조 : **reqes_product** 파이프라인 생성

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

“

컨테이너 기반 배포 전략들

- Deployment Strategy
- Deploying with Argo Rollout & Istio
- Canary Deploy with Argo CD
- A/B Testing Deploy

Comparison of Deploy Strategies

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.

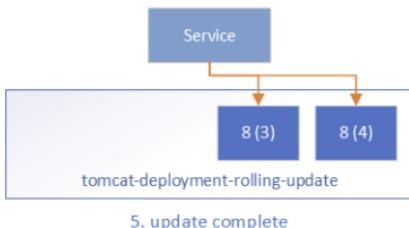
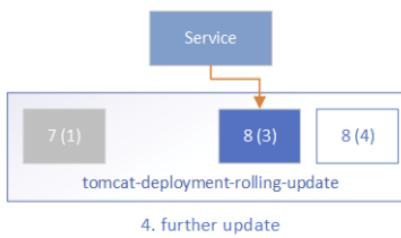
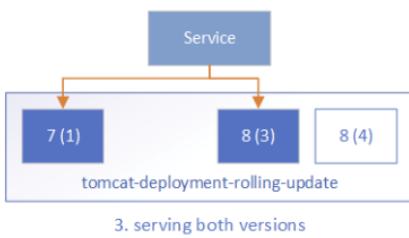
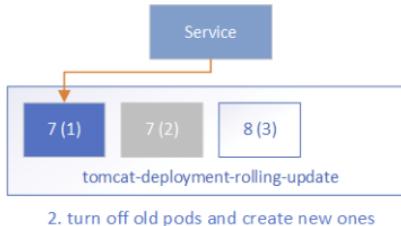
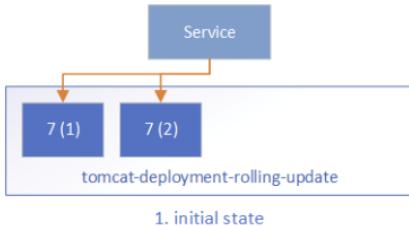
Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

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	✓	✓	✗	■ ■ ■	□ □ □	□ □ □	■ ■ ■

K8s default Deployment Strategy : Rolling Update



새버전의 배포

`kubectl set image deploy order order=order:v2`

방금 디플로이의 롤백 (복구)

`kubectl rollout undo deploy order`

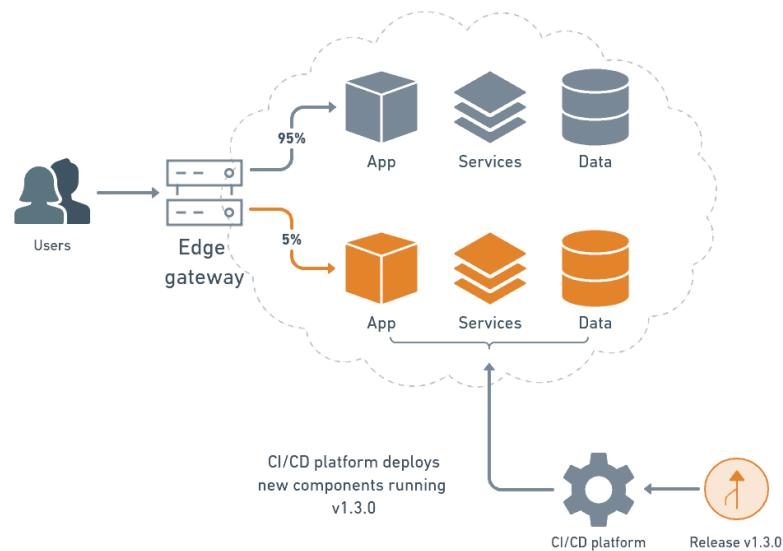
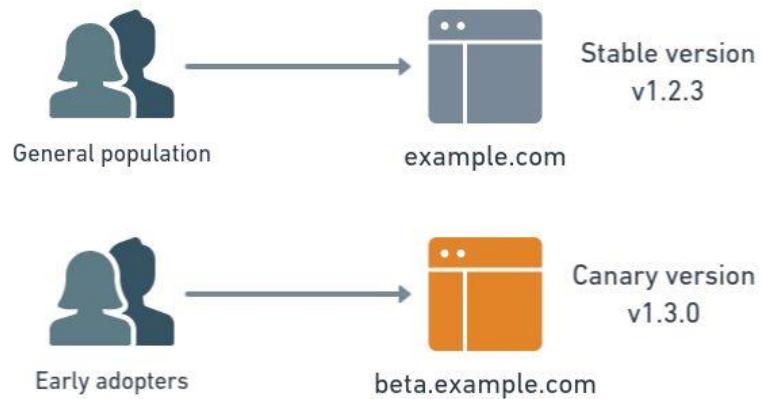
히스토리 보기

`kubectl rollout history deploy order`

특정 버전으로 복구

`kubectl rollout undo deploy order --to-revision 2`

Canary Deployment



Canary Deployment with Argo Rollouts

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollouts-demo
spec:
  replicas: 5
  strategy:
    canary:
      steps:
        - setWeight: 20
        - pause: {}
        - setWeight: 40
        - pause: {duration: 10}
        - setWeight: 60
        - pause: {duration: 10}
        - setWeight: 80
        - pause: {duration: 10}
```

Step Setting

Deploy Command

```
kubectl argo rollouts set image rollouts-demo
order=order:v2
```

Rollback Command

```
kubectl argo rollouts undo rollouts-demo
```

```
$ kubectl argo rollouts get rollout demo
Name:          demo
Namespace:     default
Status:        II Paused
Strategy:      Canary
  Step:        3/4
  SetWeight:   50
  ActualWeight: 33
Images:        argoproj/rollouts-demo:blue (stable)
                argoproj/rollouts-demo:orange (canary)
```

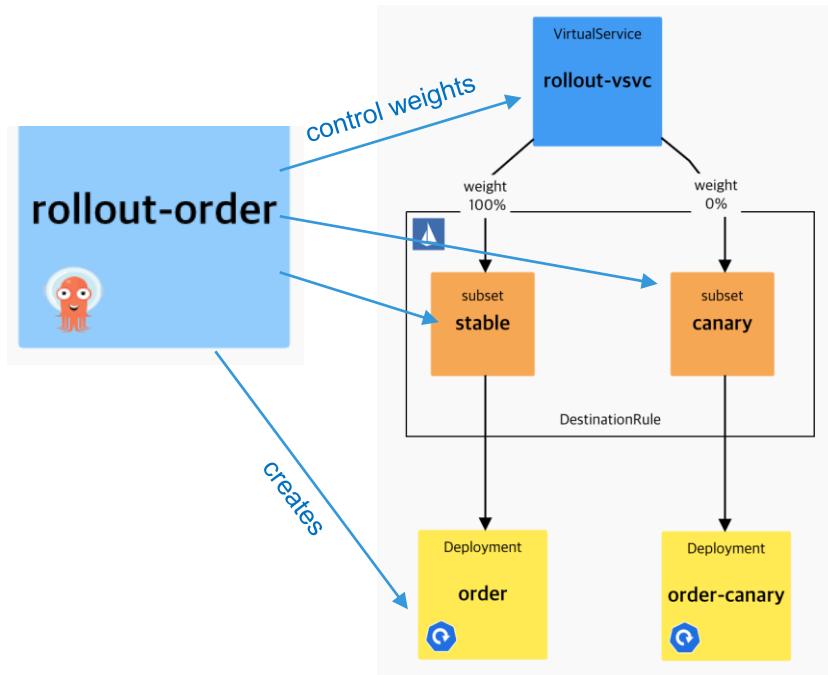
```
Replicas:
  Desired:    4
  Current:    6
  Updated:    2
  Ready:      6
  Available:  6
```

NAME	KIND	STATUS	AGE	INFO
demo	Rollout	II Paused	21m	
# revision:2				
└─demo-f66bdb575	ReplicaSet	✓ Healthy	108s	canary
└─demo-f66bdb575-tzkhf	Pod	✓ Running	108s	ready:2/2
└─demo-f66bdb575-hgrw2	Pod	✓ Running	38s	ready:2/2
# revision:1				
└─demo-8478fd7f57	ReplicaSet	✓ Healthy	21m	stable
└─demo-8478fd7f57-7kmg6	Pod	✓ Running	21m	ready:2/2
└─demo-8478fd7f57-dmmf8	Pod	✓ Running	21m	ready:2/2
└─demo-8478fd7f57-dtmbf	Pod	✓ Running	21m	ready:2/2
└─demo-8478fd7f57-fb66d	Pod	✓ Running	21m	ready:2/2

Monitoring

Argo Rollouts

– Canary with Istio (Blue/Green with Splitting Traffic)



```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollout-order
spec:
  replicas: 5
  strategy:
    canary:
      trafficRouting:
        istio:
          virtualService:
            name: rollout-vsvc      # required
            routes:
              - primary           # required
            destinationRule:
              name: rollout-destrule # required
              canarySubsetName: canary # required
              stableSubsetName: stable # required
  steps:
    - setWeight: 5
    - pause:
        duration: 10s
    - setWeight: 20
    - pause:
        duration: 10s
    - setWeight: 40
    - pause:
        duration: 10s
    - setWeight: 60
```

Argo Rollouts

: Canary Analysis with AnalysisTemplate

카나리 테스트를 통과하지
못하면 자동 롤백

```
---  
apiVersion: argoproj.io/v1alpha1  
kind: AnalysisTemplate  
metadata:  
  name: success-rate  
spec:  
  args:  
    - name: service-name  
  metrics:  
    - name: success-rate  
      successCondition: result[0] >= 0.95  
      interval: 30s  
      failureLimit: 1  
      provider:  
        prometheus:  
          address: http://prometheus.istio-system.svc.cluster.local:9090  
          query: |  
            sum(irate(  
              istio_requests_total{reporter="source",destination_service=""}  
            )) /  
            sum(irate(  
              istio_requests_total{reporter="source",destination_service=""}  
            ))
```

sample

A/B Testing

- 새 버전을 임의의 집단에게 서로 다른 컨텐츠를 노출한 다음, 어느 집단이 더 높은 관심을 보이는지 정량적으로 평가하는 방식으로 분할 테스트라 불리는 방법을 서비스에 적용한 방법론
- 테스트 예시
 - 디자인/ 레이아웃/ 서비스 콘텐츠, 이미지
 - 광고문안, 콘텐츠 제안
 - 로고와 슬로건, 헤드라인
- 테스트 목적
 - 사용자 환경 및 콘텐츠 개선
 - 전환율 증가 (페이지 방문 → 가입 및 구매)
 - 매출 증대 및 이탈율 감소



How to do A/B Testing

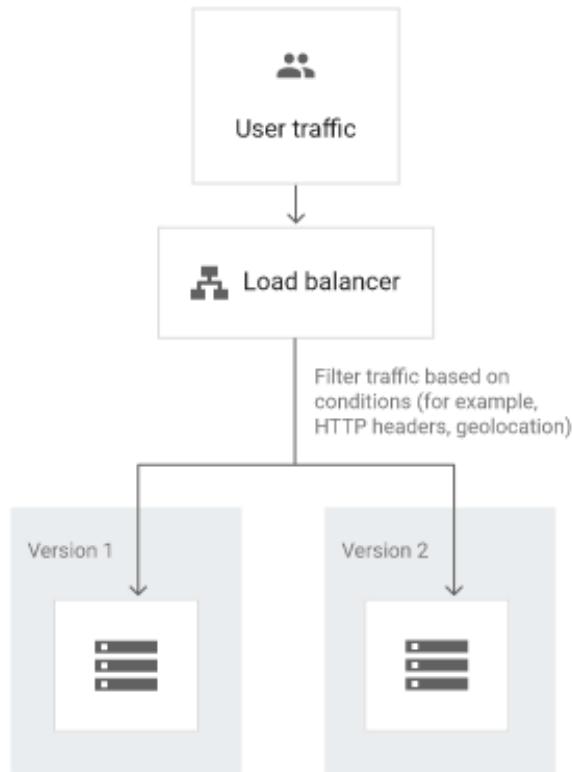
- 서비스의 새 버전 배포
 - 새 버전을 Condition을 만족하는 사용자에게 라우팅
 - Condition : 브라우저 버전, 사용자 에이전트, 위치 정보, 또는 Request Header 기반 Customized Value
 - 각 버전내에서 유의미한 통계정보 모니터링
 - 측정항목 : 방문 시간, 오픈율, 클릭률, 등록 수, 구독 현황
 - 명시적인 피드백 콘텐츠 노출

상품 디자인에 대한 만족도는 어떻습니까?				
<input type="radio"/> 매우불만족	<input type="radio"/> 불만족	<input type="radio"/> 보통	<input type="radio"/> 만족	<input type="radio"/> 매우만족

이 사이트 콘텐츠에 대하여 어느 정도 만족하셨습니까?				
<input type="radio"/> 매우불만족	<input type="radio"/> 불만족	<input type="radio"/> 보통	<input type="radio"/> 만족	<input type="radio"/> 매우만족

- 선호도가 높은 버전으로 Rollout

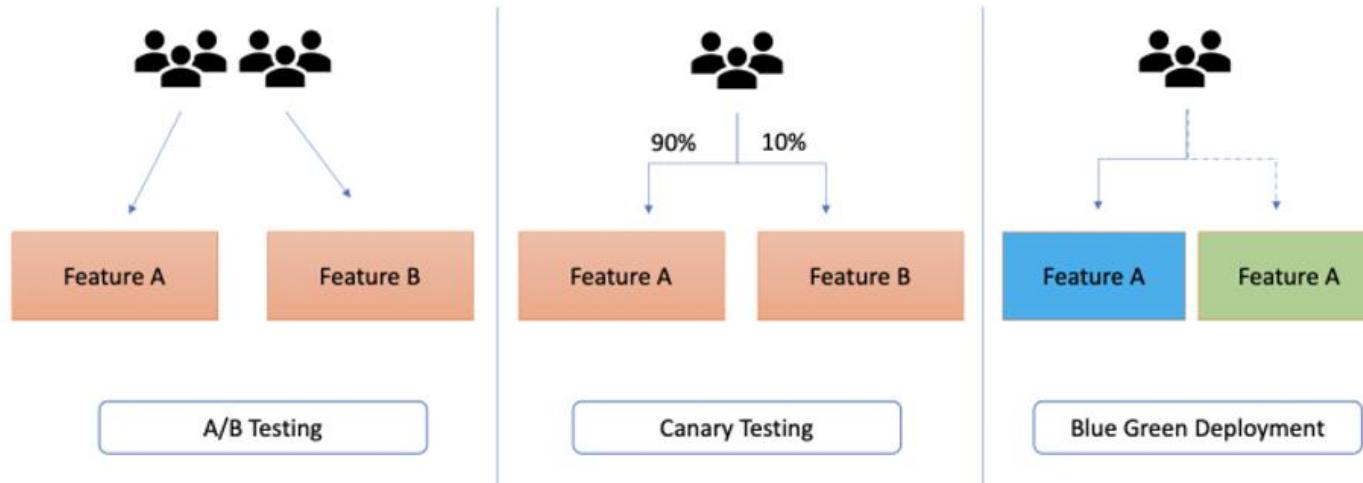
A/B Testing with Istio



```
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: order-ab-testing
spec:
  hosts:
    - order
  http:
    - match:
        - headers:
            gender:
              exact: male
      route:
        - destination:
            host: order
            subset: fontsize-40
            weight: 50
        - destination:
            host: order
            subset: fontsize-20
            weight: 50
      - route:
          - destination:
              host: order
              subset: fontsize-20
EOF
```

*Istio based
Sample yaml*

Comparison of A/B Testing, Canary, Blue/Green



- A/B 테스팅은 대조군과 실험군으로 나뉜 두 그룹의 작용을 비교 분석하여 평가하는 전략
- 카나리 배포는 소수 사용자에게 먼저 새 버전을 릴리즈 한 다음, 서비스를 검증하는 전략
- 블루/그린 배포는 새버전의 릴리즈를 비활성 상태로 배포하고 모든 테스트가 완료되었을 때, 새 버전으로 라우터를 전환하는 배포 전략

“

GitOps, 깃옵스

- GitOps 핵심 개념
- GitOps based Deployment with Argo CD

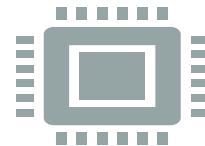
What's GitOps ?

2017년 위브웍스(Weaveworks Inc.)에서 처음 사용한 용어로 프로젝트에 DevOps의 실천 방법



프로젝트에 데브옵스를 적용하는 실천
방법 중 하나

- 클라우드 네이티브 애플리케이션을 대상으로 한
지속적 배포(Continuous Deployment)에 초점
- 선언형 모델(Declarative Model)을 지원하는 최근
도구들이 클라우드 네이티브에 중점
- 애플리케이션의 배포와 운영에 관련된 모든 요소를
코드화하여 깃(Git)에서 관리(Ops)하는 것



기본 개념 :
Single Source Of Truth, SSOT

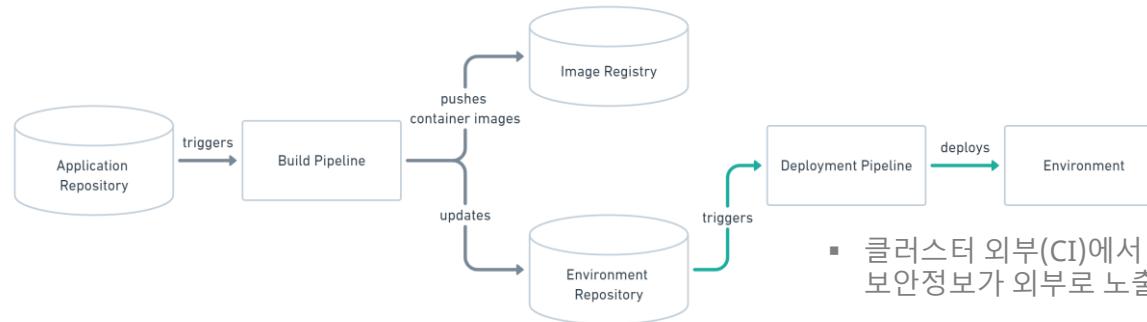
- Git을 단일 진실 공급원으로 지정하고 오직 이
곳에서만 관리하도록 함

GitOps의 원칙

- 모든 시스템은 선언적으로 선언되어야 한다.
 - 쿠버네티스의 manifest들은 모두 선언적으로 작성되었고, 이를 Git으로 관리한다면 versioning과 같은 Git의 장점과 더불어, SSOT(single source of truth)를 구축
- 시스템의 상태는 Git의 버전을 따라간다.
 - Git에 저장된 쿠버네티스 manifest를 기준으로 시스템에 배포되기 때문에 이전 버전의 시스템을 배포하고 싶으면 git revert와 같은 명령어를 사용
- 승인된 변화는 자동으로 시스템에 적용된다.
 - 한번 선언된 manifest가 Git에 등록되고 관리되기 시작하면 변화(코드 수정 등)가 발생할때마다 자동으로 시스템에 적용되어야 함
- 배포에 실패하게 되면, 이를 사용자에게 경고해야 한다.

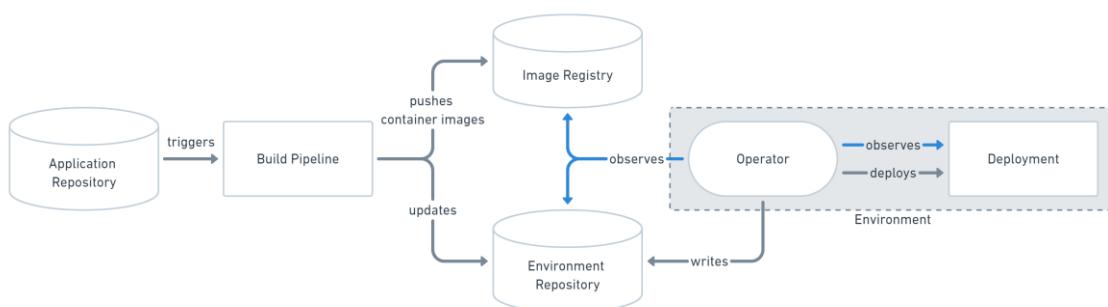
GitOps 배포 전략

- **Push-based Deployments** : Git Repo에 있는 Manifest 파일이 변경되었을 때, 배포 파이프라인을 실행시키는 구조



- 클러스터 외부(CI)에서 읽기/쓰기(RW) 권한이 존재하게 되어, 보안정보가 외부로 노출될 수 있다는 단점

- **Pull-based Deployments** : 배포하려는 클러스터에 위치한 별도의 오퍼레이터가 배포 역할을 대신하는 구조



- 보안상 안전
- 자격증명을 내부에서 관리
- Jenkins X, Bamboo

GitOps 배포 도구



- **Argo CD**

- 직관적인 GUI를 제공한다.
- 수동 Sync 등 모든 조작을 GUI에서 할 수 있다.

- **Flux CD**

- GitOps를 제시한 Weaveworks사의 도구로 제작이 간단하고 경량이지만, GUI를 제공하지 않음

- **Jenkins X**

- 이 도구에서는 CI/CD에 대응한 파이프라인 자체를 구축 가능
- Argo CD, Flux CD는 CD만 가능

Lab. Deployment Strategies



Lab Time

Lab. 12Street Deployment by Argo CD

Argo CD 는 Application 이라고 하는 CRD 를 기반으로 깃설정을 저장하며, 타겟 깃의 변화를 감지하여 자신이 포함된 클러스터에 동기화한다.



“ Appendix

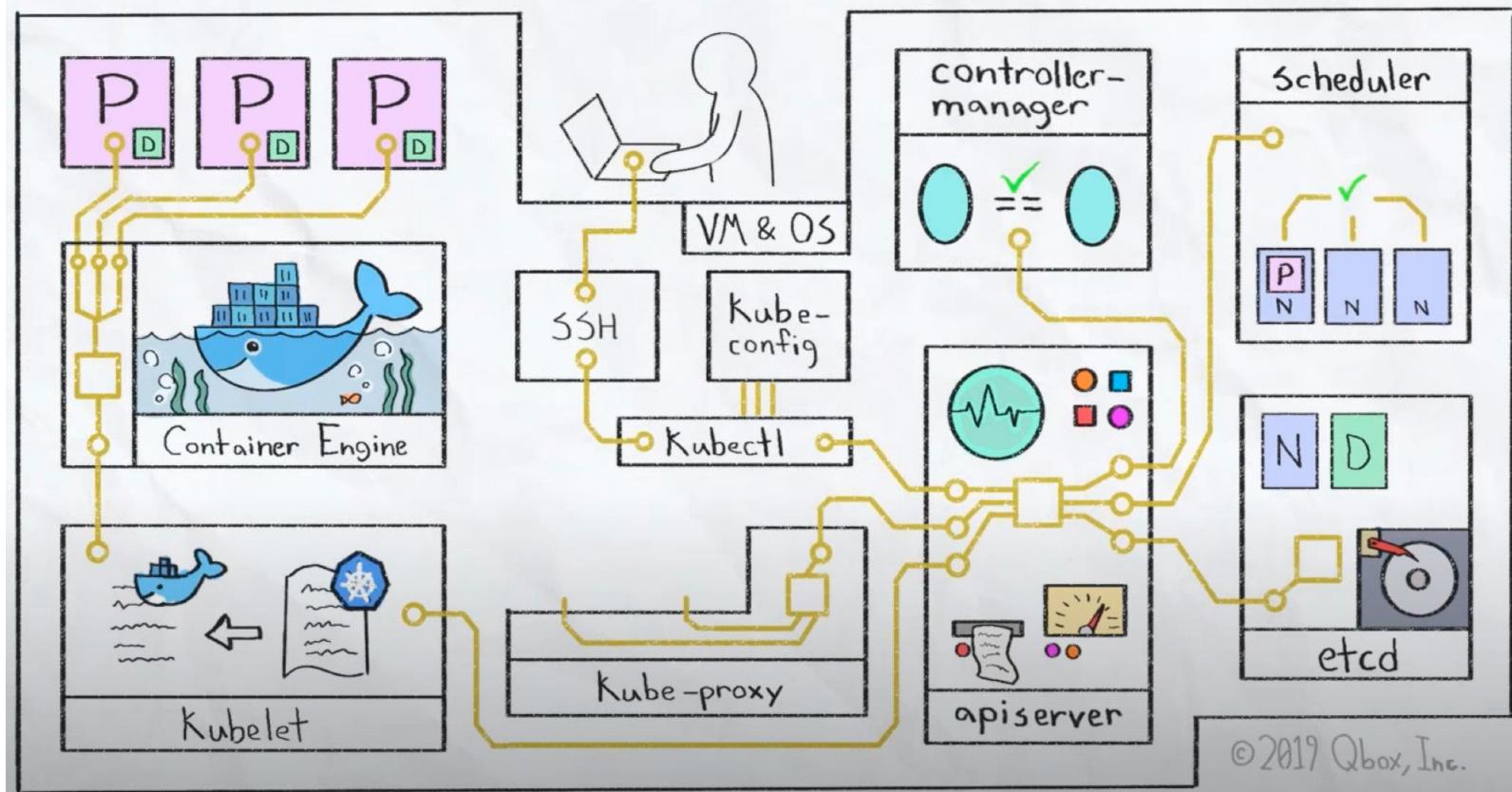
- #1: Kubernetes Architecture
- #2: Case Study - Event Sourcing 기반 예금 입출금
- #3: MSA 프로젝트 관리와 조직 변화관리
- #4: k8s Basic Service Object Model

“ Appendix #1:

Kubernetes Architecture

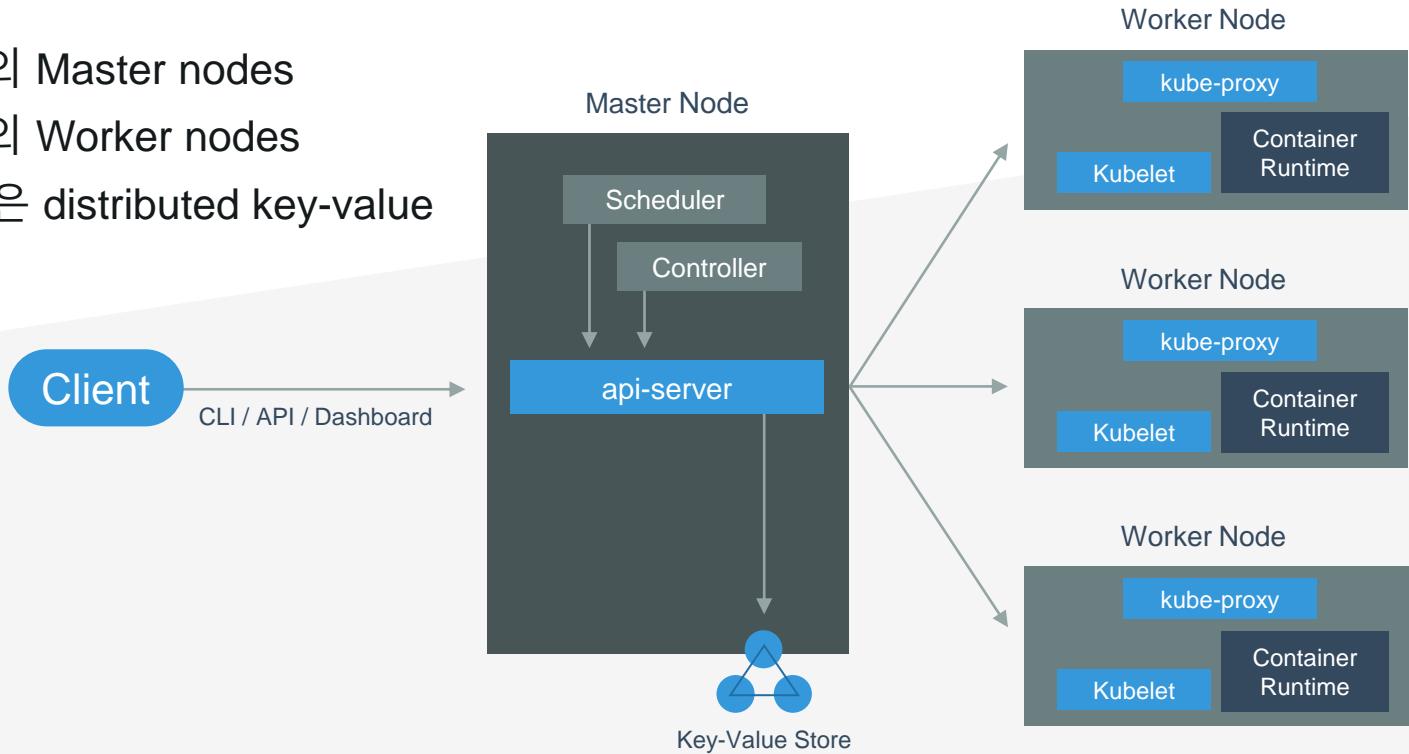
- Master / Worker Node

Anatomy of Kubernetes



Kubernetes Architecture

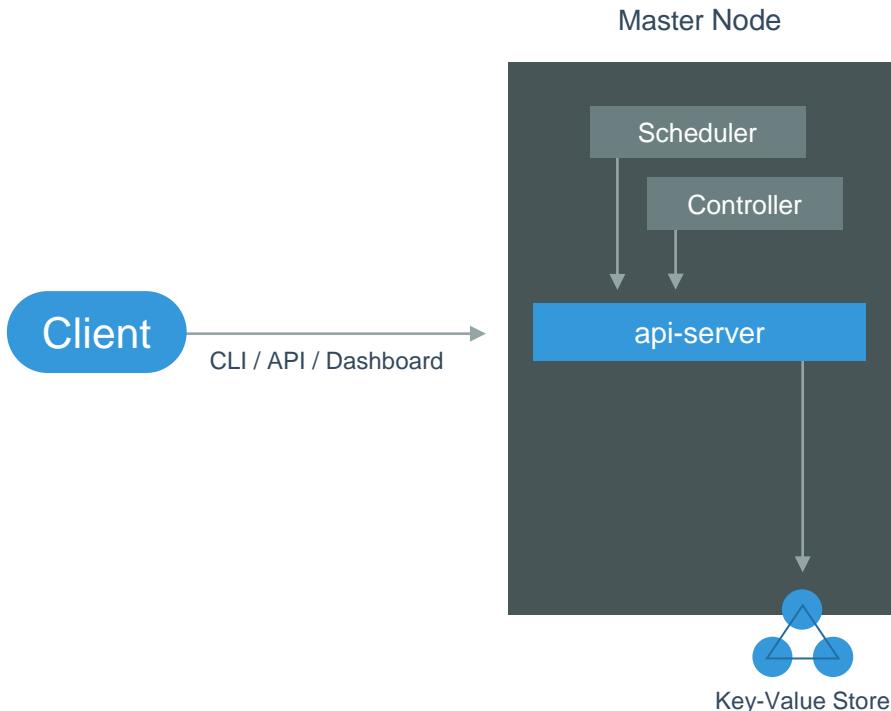
- 하나 이상의 Master nodes
- 하나 이상의 Worker nodes
- etcd 와 같은 distributed key-value store



Master Node

- **Master node는 Kubernetes 클러스터를 관리하며, 모든 관리자 업무의 시작점**
- 다양한 모듈이 확장성을 고려하여 기능별로 분리
- CLI, GUI (Dashboard), APIs 등으로 Master node에 접근 가능

Master Node 구성요소



- API server
- Scheduler
- Controller manager
- Etcd (distributed key-value store)

- ✓ 장애 대비하여, 하나 이상의 Master node를 한 클러스터에 구성하여 안정성을 높임
- ✓ 하나 이상의 Master node 가 존재할 경우, HA (High Availability)모드로 그 중 하나가 리더(leader)로써 작동하고 나머지는 팔로워(followers)로 운영

Master Node 구성요소

#1. API Server

- 원하는 상태를 Key-Value 저장소에 저장하고, 저장된 상태를 조회하는 작업 수행
- 모든 administrative tasks는 master node에 있는 API server를 통해 수행
- 사용자/운영자가 보낸 요청은 API server를 통해 처리되고, 실행 결과는 분산 키-값 저장소 (Distributed key-value store)에 저장

Master Node 구성요소 #2. Scheduler

- Pod를 여러가지 조건(필요한 자원, 라벨)에 따라 적절한 노드에 할당해 주는 모듈
- Scheduler가 개별 worker node에게 업무를 분담
- Scheduler는 각 worker node의 리소스 사용 정보를 가짐
- “disk==ssd” 와 같은 label이 설정된 worker node에 업무 부여하는 등 사용자/운영자가 설정한 제약에 대해 인지하고 있음
- Scheduler는 Pods 와 Services 단위로 업무 수행

#3. Controller Manager

- **Kubernetes Cluster에 있는 모든 오브젝트의 상태를 관리하는 모듈**
- Controller manager는 Kubernetes Cluster의 상태를 조절하는 non-terminating control loops 관리
- 각각의 control loops는 자신이 관리하는 객체의 desired state를 알고 그 객체들의 현 상태를 API 서버를 통해서 모니터링
- 제어 루프에서 관리하는 객체의 현 상태가 원하는 상태와 다르면, 제어 루프는 수정 단계를 수행하여 현 상태와 원하는 상태를 일치시키는 작업 수행
- Kubernetes release 1.6, kube-controller-manager와 cloud-controller-manager로 세분
Cloud-controller-manager is structured using a plugin mechanism that allows different cloud providers to integrate their platforms with Kubernetes.

Master Node 구성요소

#4. Etcd

- RAFT* 알고리즘을 이용한 클러스터 상태를 저장하는 key-value 저장소
- 단순 R/W 기능외, Watch기능이 있어 상태가 변경되면 트리거 로직 실행 가능
- 클러스터의 모든 설정, 상태가 저장되므로 Cluster Backup / Restore 유리
- Etcd는 오직 API 서버와 통신하고, 다른 모듈은 API서버를 통해 etcd에 접근

* RAFT 알고리즘: 분산 환경에서 상태를 공유하는 알고리즘, consensus algorithm(Paxos, RAFT)

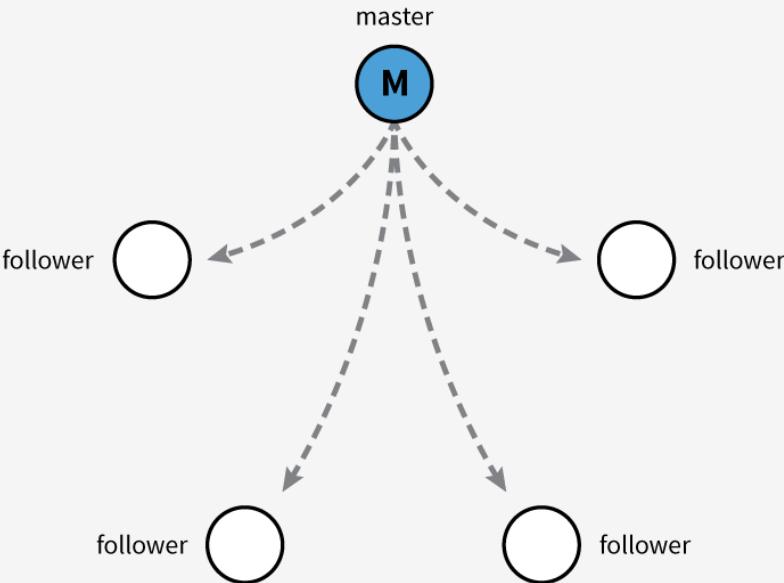
#4. Etcd - State 관리

- Kubernetes는 etcd를 사용해 클러스터의 상태를 저장
- etcd는 ‘Raft Consensus Algorithm’을 기반으로 하는 distributed key-value store
 - Raft는 장치의 집합으로써 몇몇 구성원에 장애가 발생하여도 살아남아 일관적인 집단으로 작동할 수 있도록 해 줌
- etcd는 Go 프로그래밍 언어로 작성
- Kubernetes에서는 클러스터의 상태를 저장하는 기능 말고도 Subnets, ConfigMaps, Secrets 등을 저장하기도 함

Master Node 구성요소

#4. Etcd - State 관리

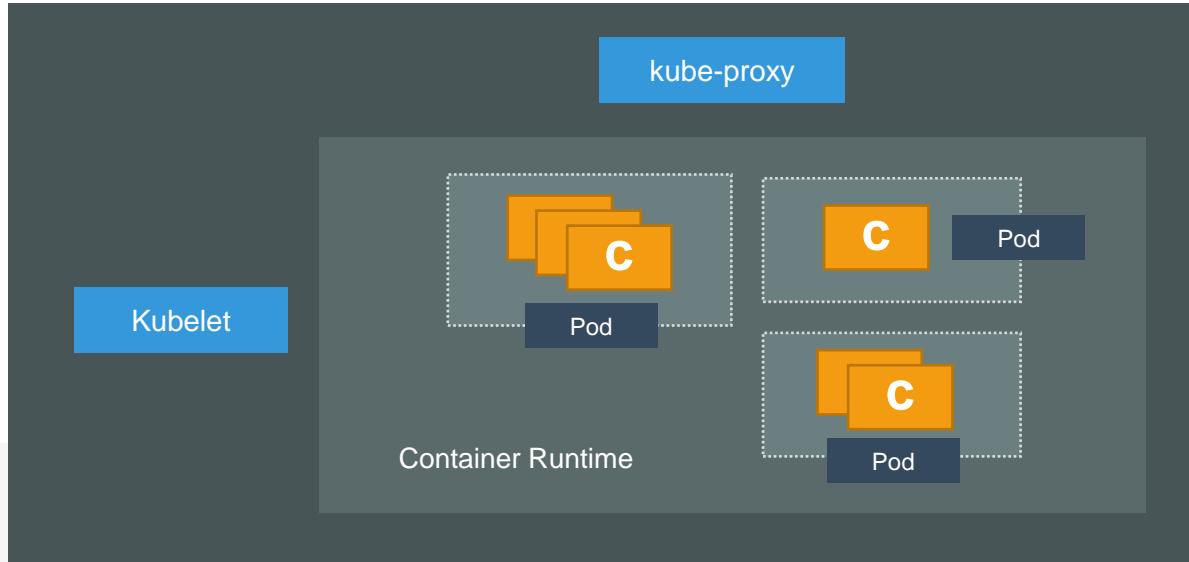
- 그룹의 노드 중 하나는 Master로 작동을 하고, 나머지는 Follower로 작동
- 모든 노드는 Master 후보



Worker Node

- 마스터 서버와 통신하면서 필요한 Pod를 생성하고 네트워크와 볼륨을 설정
- Worker node는 Master node에 의해 관리되며, Pod를 이용 어플리케이션을 실행
- Pod는 Kubernetes의 스케줄링 단위
- 하나 이상의 컨테이너의 논리적 집합이며, 항상 같이 스케줄링 됨

Worker Node 구성요소



- Container runtime
- kubelet
- kube-proxy

#1. Container Runtime

- 컨테이너의 lifecycle을 관리/실행 하기위한 Worker node상의 Container runtime
 - (runtimes: containerd, cri-o, RKT, Docker)
 - ✓ kubernetes는 자유롭게 컨테이너 런타임을 선택 가능
 - ✓ CRI (Container Runtime Interface) 표준의 등장과 함께 Kubernetes에서 사용할 수 있는 런타임은 docker, cri-o, rkt, containerd 등 다양하며, 릴리즈 버전에 따라 지원가능한 런타임이 종속적임

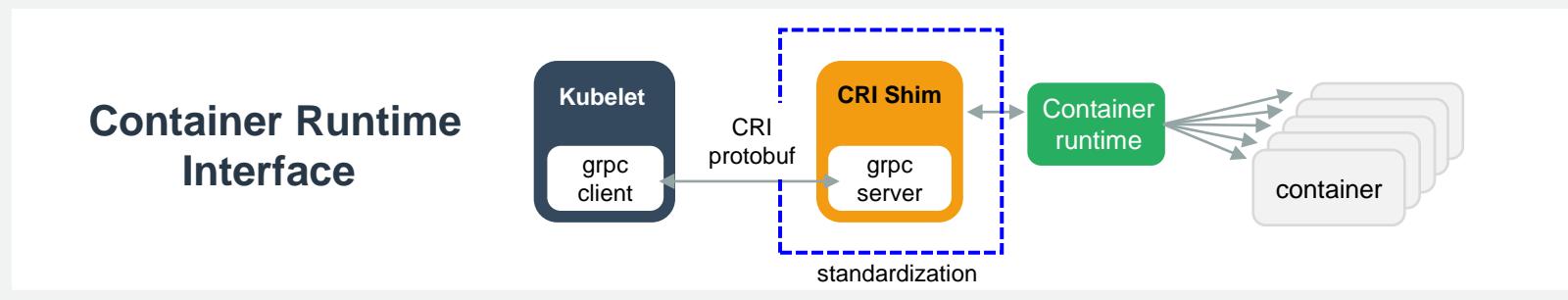
Worker Node 구성요소

#2. Kubelet

- **Node에 할당된 Pod의 생명 주기를 관리**
- kubelet은 master node와 통신하기 위해 worker node에서 작동하는 에이전트
- Pod 설정을 수신하고 해당 Pod와 관련된 컨테이너를 실행
- 항상 컨테이너가 정상 작동하는지 확인
- Kubelet은 Container Runtime Interface(CRI)를 이용하여 컨테이너 런타임에 연결
 - CRI는 protocol buffers, gRPC API, and libraries 등을 포함

Worker Node 구성요소

#2. Kubelet



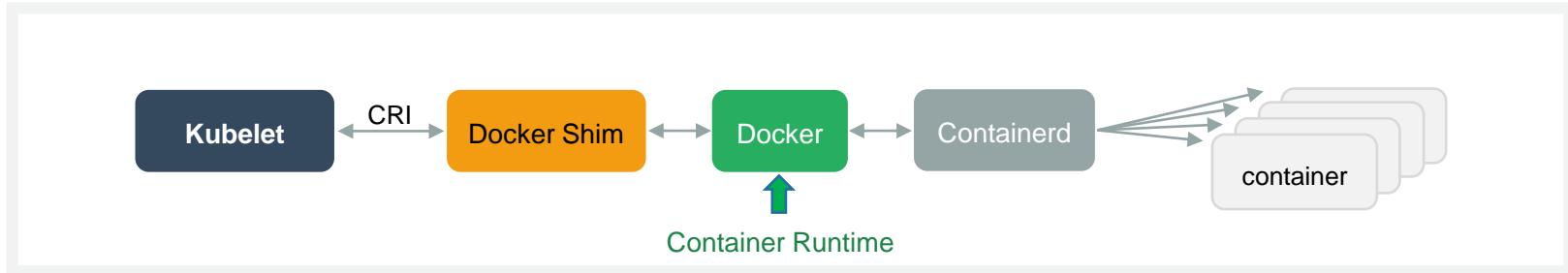
- Kubelet이 명령을 받으면 Docker runtime을 통해 Container를 생성하거나 삭제
- Docker이외의 여러 컨테이너 기술이 나오면서 Kubelet과 Container runtime 사이에 표준 인터페이스가 제정되었는데, 이를 **CRI Shim**이라 함
- 새로운 Container runtime은 CRI Shim스펙에 맞춰 CRI 컴포넌트를 구현

* gRPC : Google에서 처음 개발한 공개 원격 프로시저 호출(RPC) 시스템, 인터페이스 설명언어로 프로토콜 버퍼 사용

* Protocol buffer 줄임말로 크기를 줄인 직렬화 데이터 구조(이진 메시지 형식)

Worker Node 구성요소

#2. Kubelet : CRI Shims



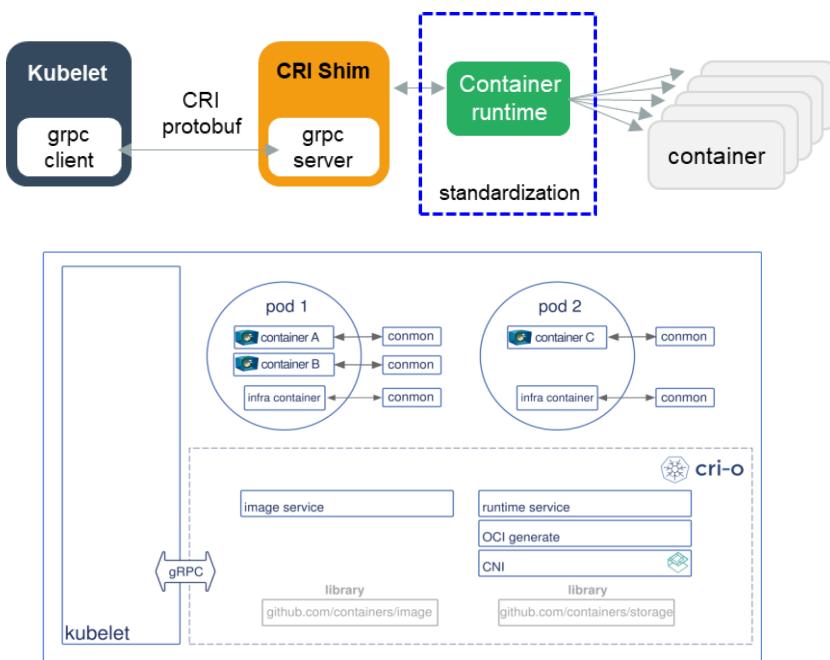
- **Docker shim**

Docker shim은 Worker node에 설치된 Docker를 이용하여 컨테이너를 생성

- 내부적으로는 Docker는 containerd를 이용해 컨테이너를 관리

Worker Node 구성요소

#2. Kubelet : CRI-O



• Container Runtime 표준화

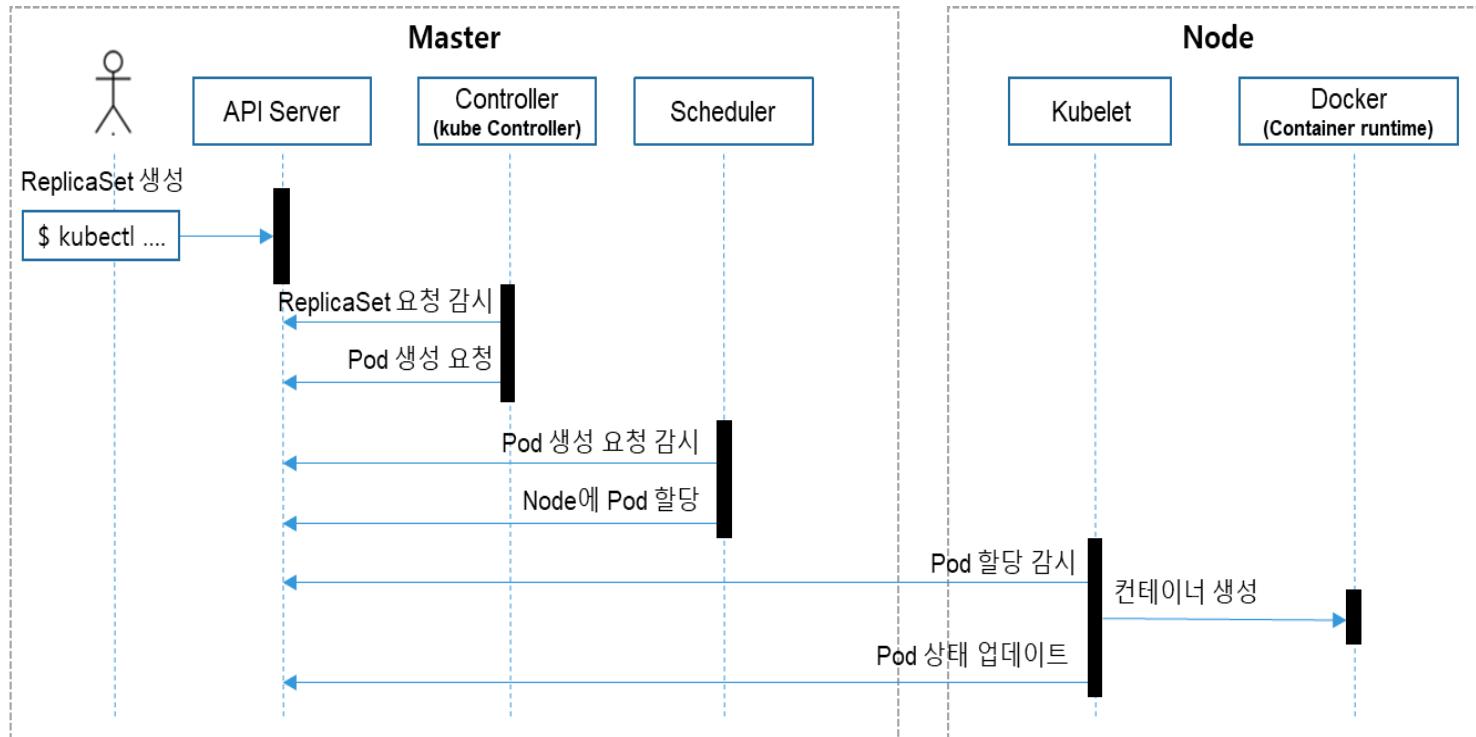
- OCI(Open Container Initiative) 표준에 대한 구현체가 컨테이너 런타임
- 레드햇이 개발한 Kubernetes를 위한 표준 컨테이너 런타임이 CRI-O
- 컨테이너 런타임을 제어하는 오케스트레이션의 표준은 CNCF(Cloud Native Computing Foundation)이며 구현체는 "kubernetes"

Worker Node 구성요소

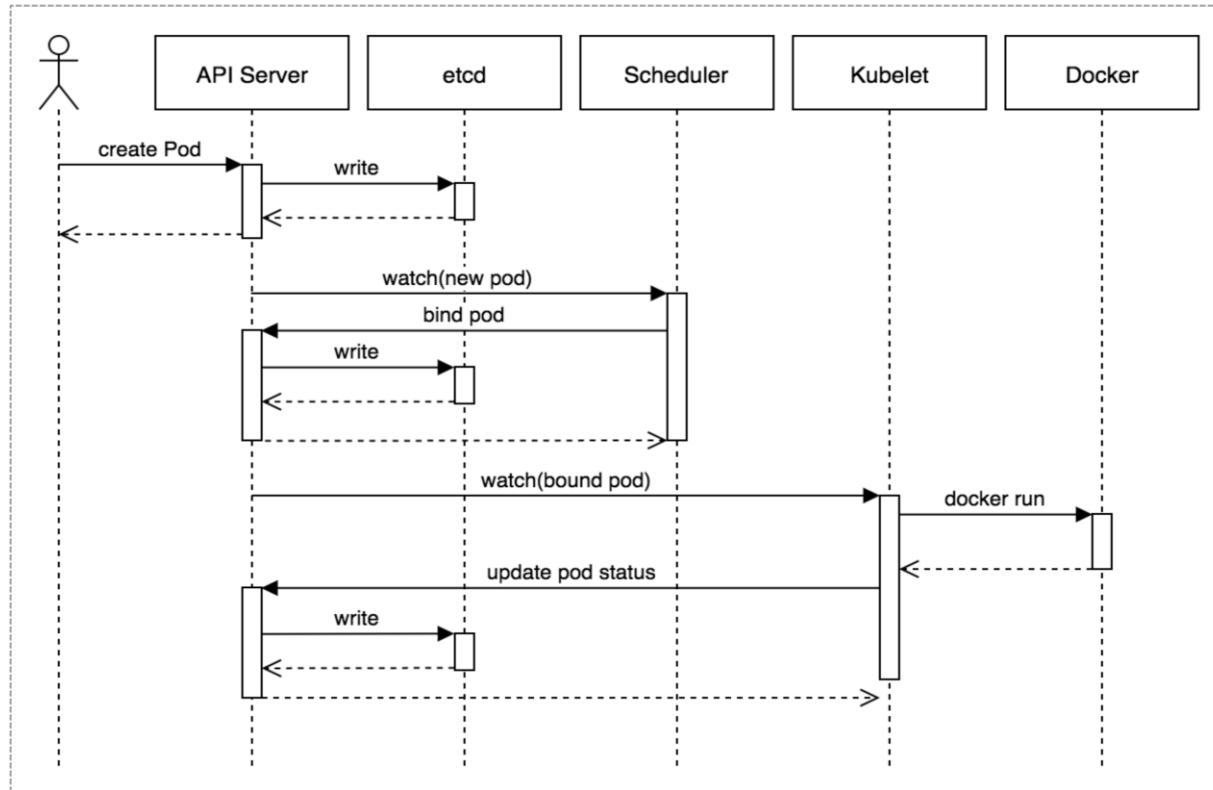
#3. Kube-proxy

- Kubelet이 Pod를 관리한다면, Proxy는 Pod로 연결되는 네트워크를 관리
- 초기, kube-proxy가 프록시 서버로 동작하면서 실제 요청을 받아 각 Pod로 포워딩
 - 성능 이슈로 iptables를 설정하는 방식으로 변경, 최근엔 IPVS 지원 시작
- 애플리케이션에 액세스하기 위해 포드에 직접 연결하는 대신 "서비스"라는 논리적 구성을 연결 엔드포인트로 사용
 - Service 그룹과 관련된 Pod에 접속 시 자동으로 Load Balancing 됨
- kube-proxy는 Network proxy로서 각각의 Worker node에서 작동하며, 각각의 Service endpoint를 생성/삭제하기 위해 API server를 지속적으로 Listening
 - kube-proxy는 각 Service endpoint에 접근할 수 있는 경로 설정

K8s working process #1



K8s working process #2

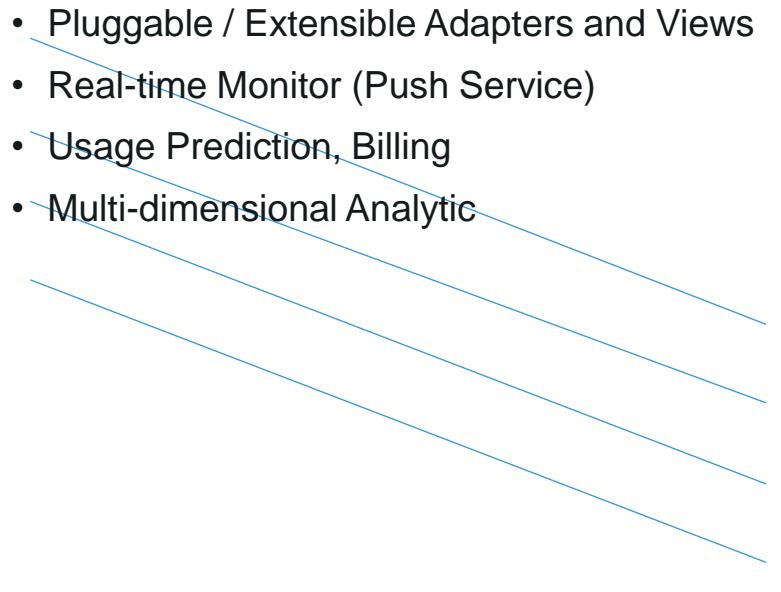


“

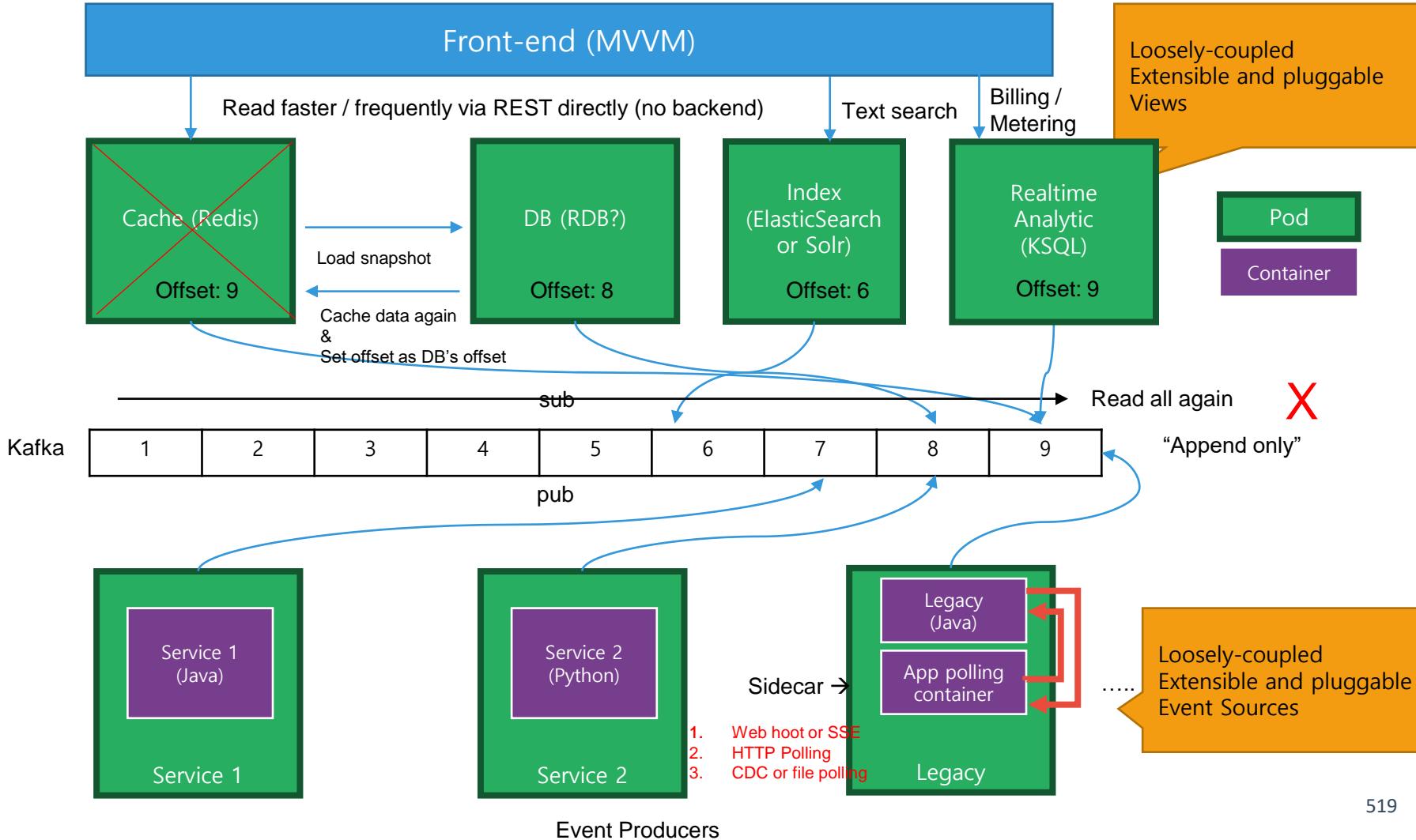
Appendix #2:

Case Study - Event Sourcing기반 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

Front-end (MVVM)



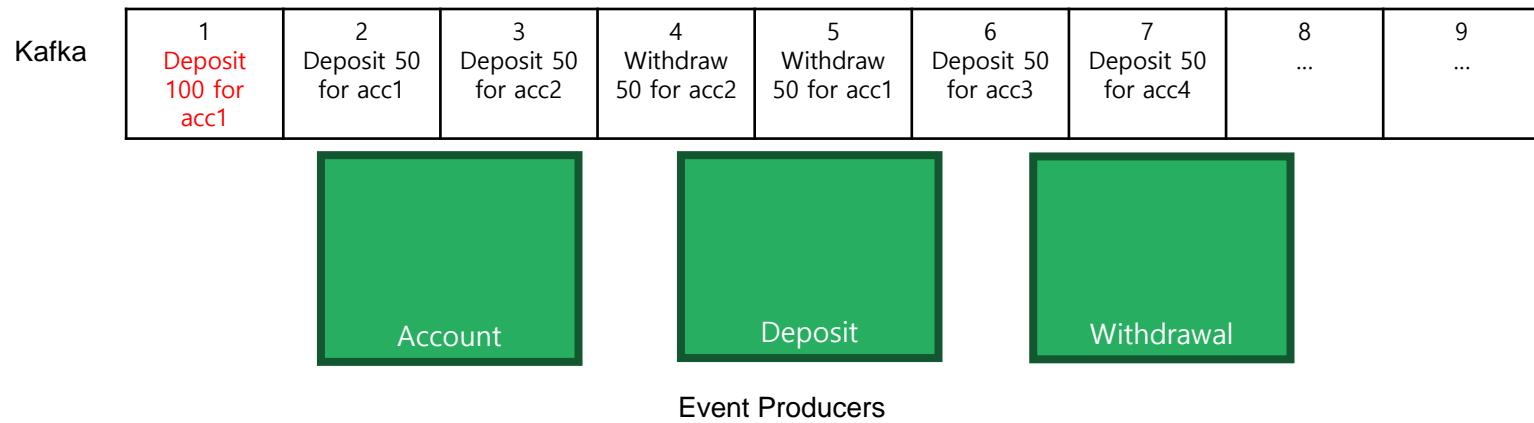
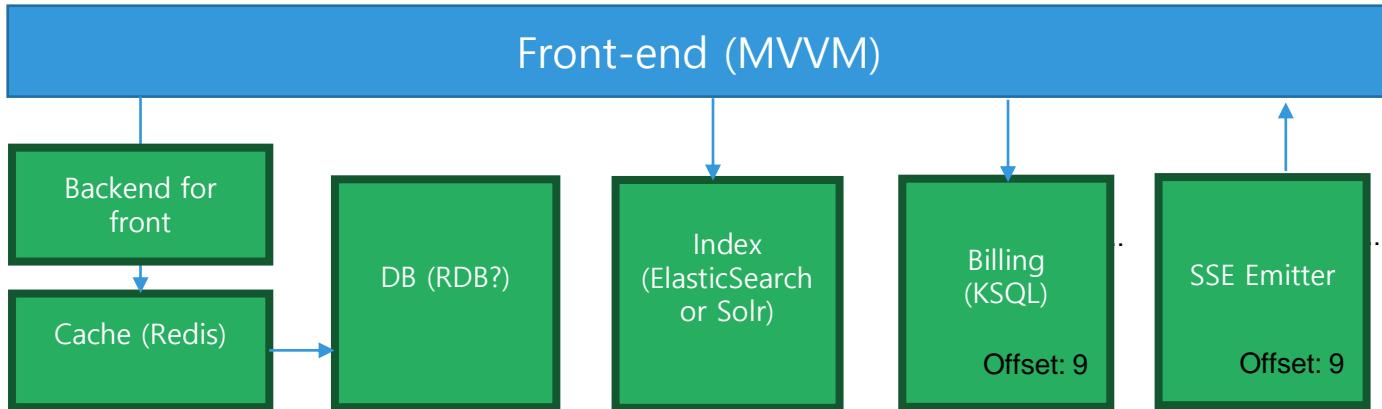
Event Formatting: Event Sourcing

Kafka

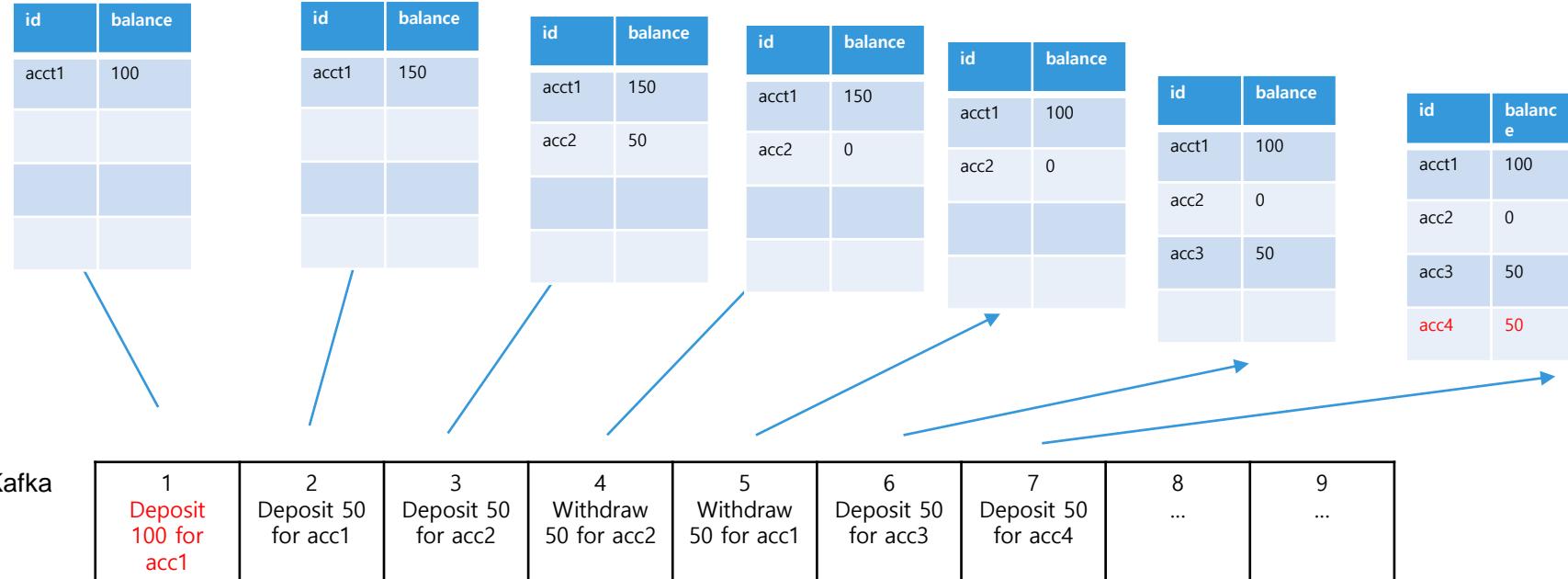
1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Append the “Diff” only

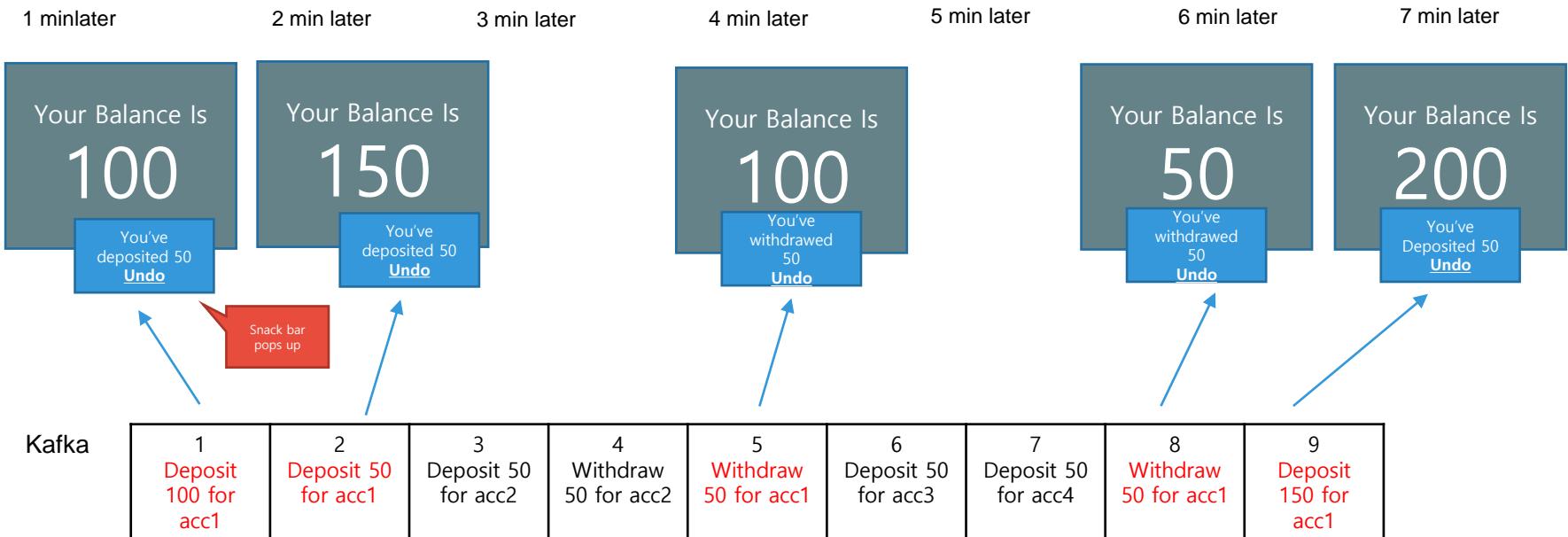
- 1: key: account1, value: { action: Deposit, amount: 100 }
- 2: key: account1, value: { action: Deposit, amount: 50 }
- 3: key: account2, value: { action: Deposit, amount: 50 }
- 4: key: account2, value: { action: Withdraw, amount: 50 }
- 5: key: account1, value: { action: Withdraw, amount: 50 }
- 6: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



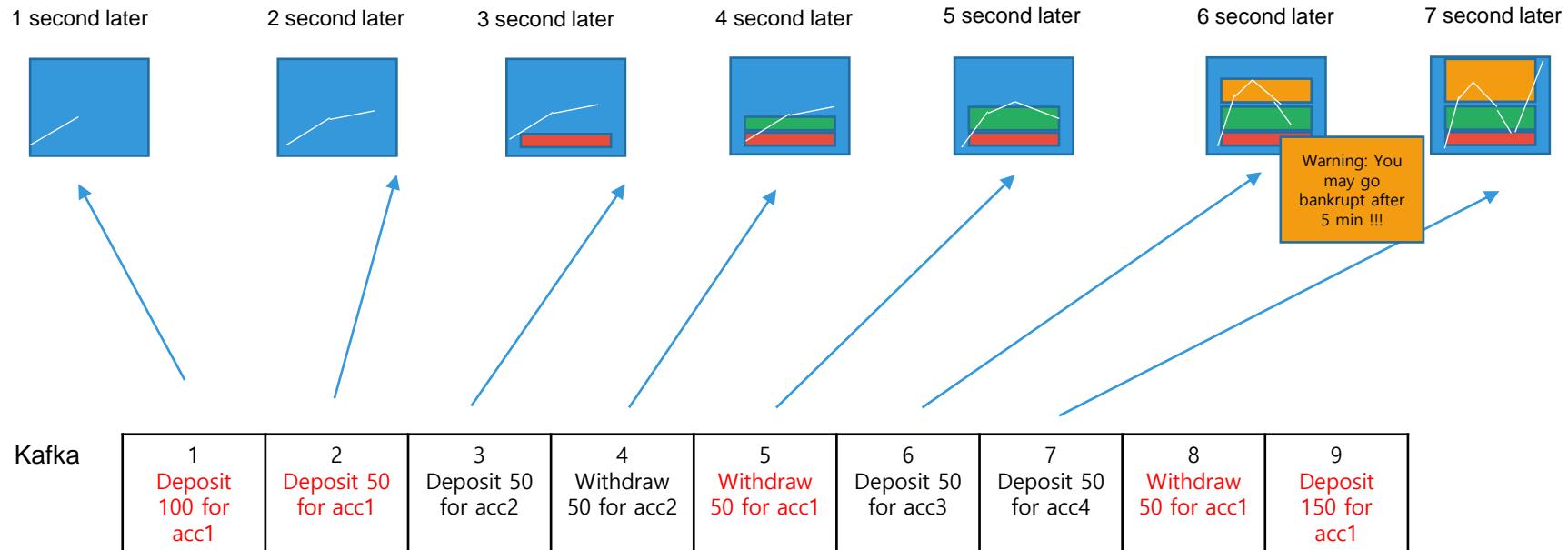
Aggregate view in Database



View in Front-end (Push service)



View in Realtime Analytic (Early Warning)



** Using KSQL or Kstreams:

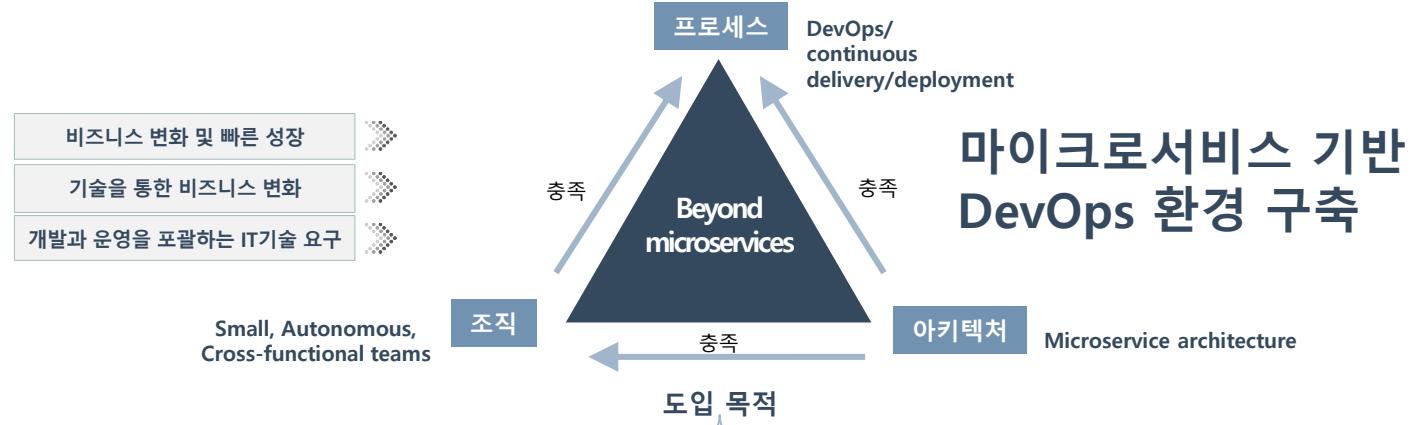
“

Appendix #3:

MSA 프로젝트 관리와 조직 변화관리

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.



개발 생산성	배포 유연성	확장을 통한 안정성
<ul style="list-style-type: none">서비스를 충분히 작은 크기로 나누어 독립적으로 개발하고 마이크로서비스간에는 API 규약에 따라 상호 연계개발자는 비즈니스 로직에만 집중 가능테스트 시에도 연관 마이크로서비스만 테스트를 수행하므로 배포 일정 축소	<ul style="list-style-type: none">마이크로서비스 아키텍처는 개별 서비스 단위로 개발/패키징/빌드/테스트/배포가 가능하므로 각 서비스를 각자의 로드 맵에 따라 개발 배포	<ul style="list-style-type: none">시스템의 기능을 분할하여, 서비스 단위 별로 트래픽에 대해 자율적 확장을 통한 성능 향상

Specific Goal Setting – MSA Maturity Levels

	Early	Inception	Expanding	Mature
기능분해	비즈니스 역량은 도출한 비즈니스 기능과 유즈케이스를 단위로 하여 분리 가능	뭉뚱그리지 말고 확실히 분해해라. 추출 된 각 유스 케이스와 인터페이스를 통해 액세스 할 데이터에 대해 잘 정의 된 인터페이스를 가짐.	컨텍스트매핑의 결과로 도출된 범위가 한정된 문맥 (Bounded context) 을 기준으로 분해한다. ubiquitous language 가 다른 bounded context간의 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

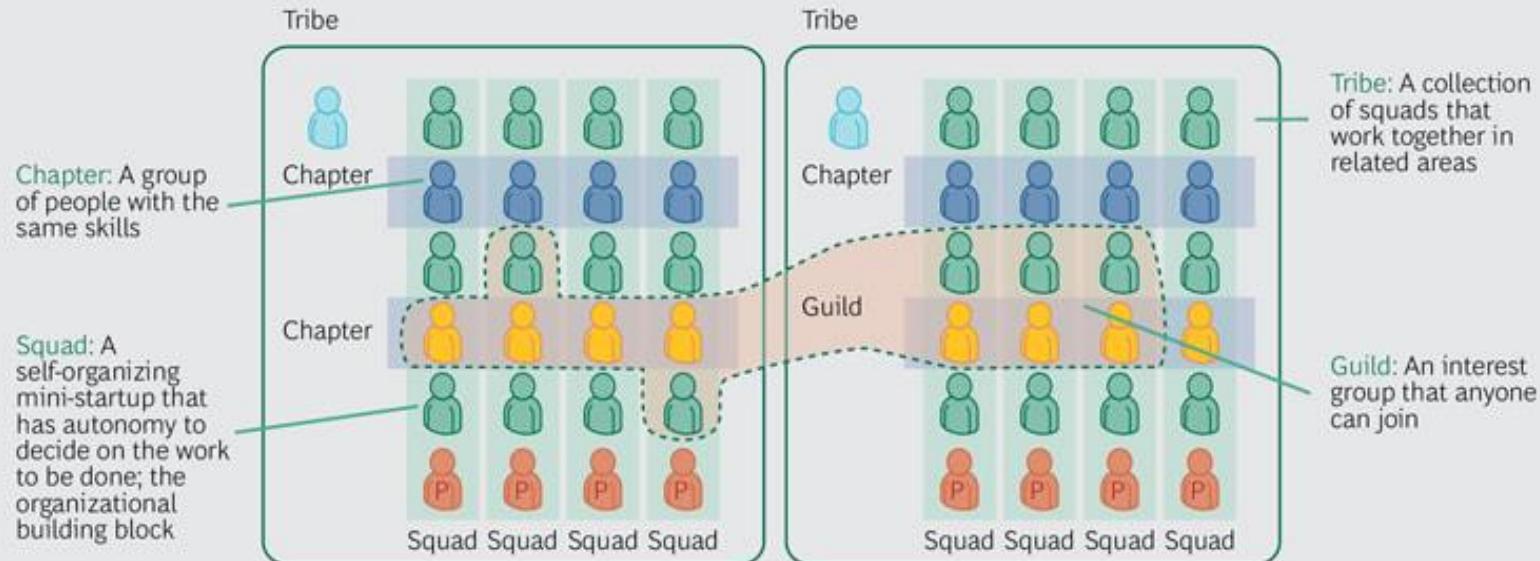
아키텍처 패턴	체크 포인트	회피 패턴
Circuit Breaker	<ul style="list-style-type: none"> 성능저하를 막아주나, Fail-fast 전략으로 사용자 경험이 나빠질 수 있음 적용대상이 비동기, 이벤트 기반으로 처리가능하다면 그 기반으로 전환 	Saga
Database per service	<ul style="list-style-type: none"> ACID 트랜잭션 비용을 포기 ACID 트랜잭션 비용 포기가 불가하다면, Shared database 로 처리해야 하거나 Semantic Locking 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

예시
SAMPLE

조직 전환 방안 – TO-BE



Source: Spotify

사례1 – ING BANK

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

Principles

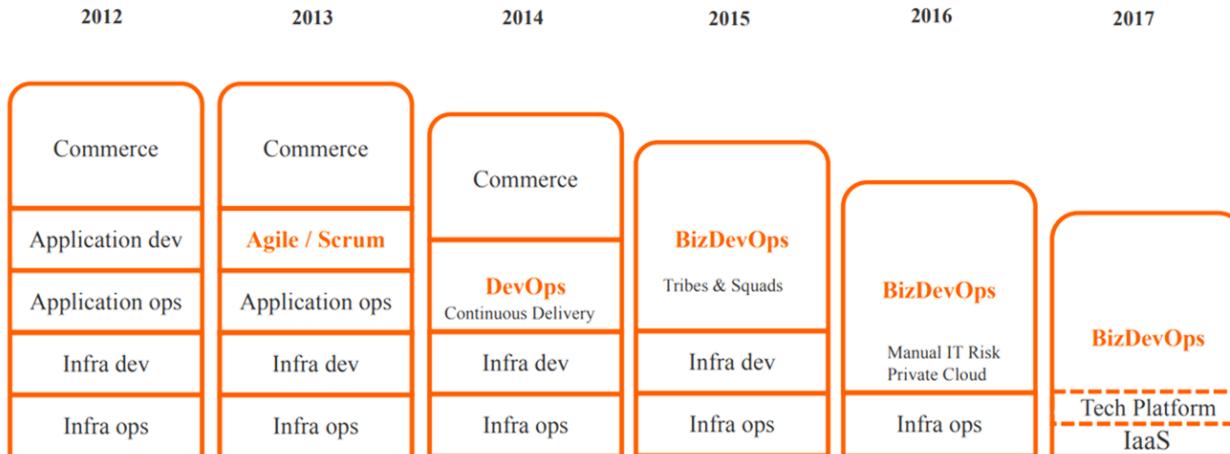
Inspired by the Agile methodology, eight principles are at the heart of our Way of Working:

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



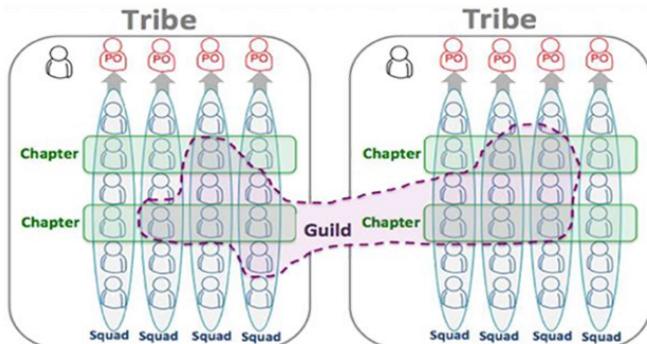
Engineer: From single discipline to **full stack engineers**: designing, coding, test engineering, infra engineering, etc

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

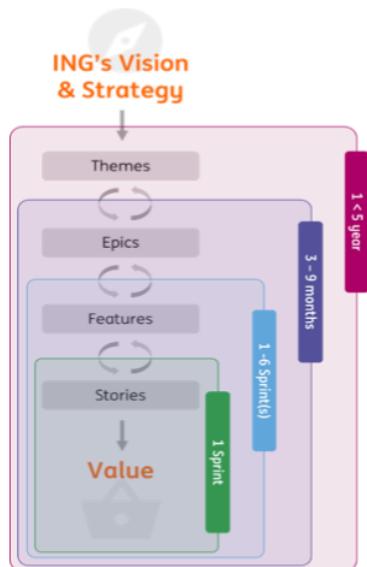


- **Squads**, are high performing “stable” teams who provide the execution power to the One WoW
- A **Tribe** is a collection of Squads organized around the same purpose
- **Chapters** are formal “groups” of members with the same background / expertise deciding on how things need to be done within a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

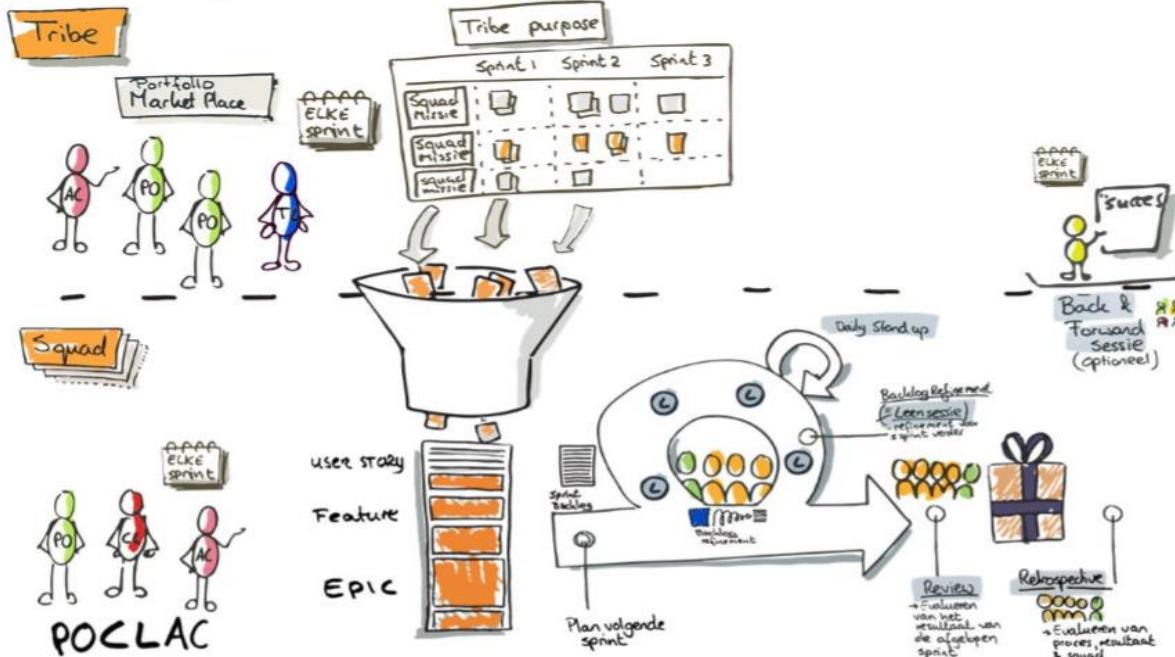


- A **Theme** connects the dots between Strategy and execution. Hence, a Theme represents an (investment) area that supports the strategic vision. Themes are representing one to five year(s).
- A Theme is broken down in multiple **Epics**: large scale bank initiatives that will be delivered in one to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

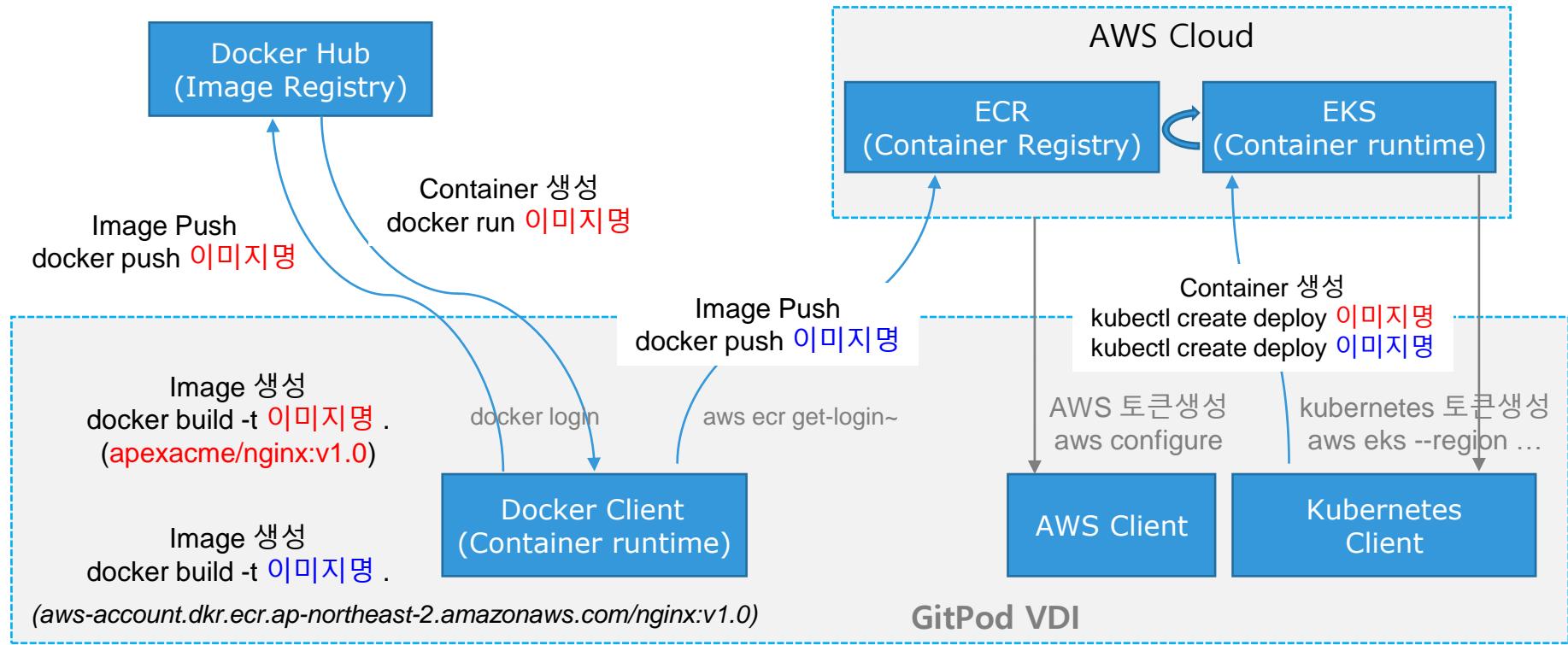


“

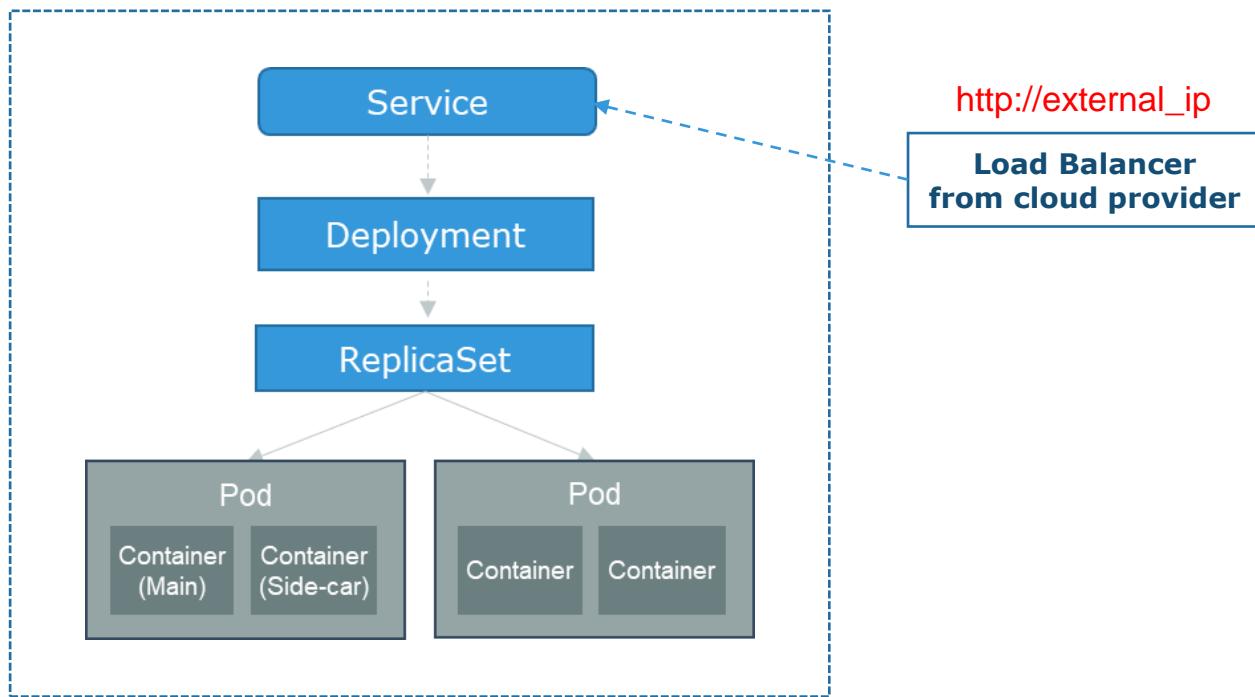
Appendix #4:

k8s Basic Service Object Model

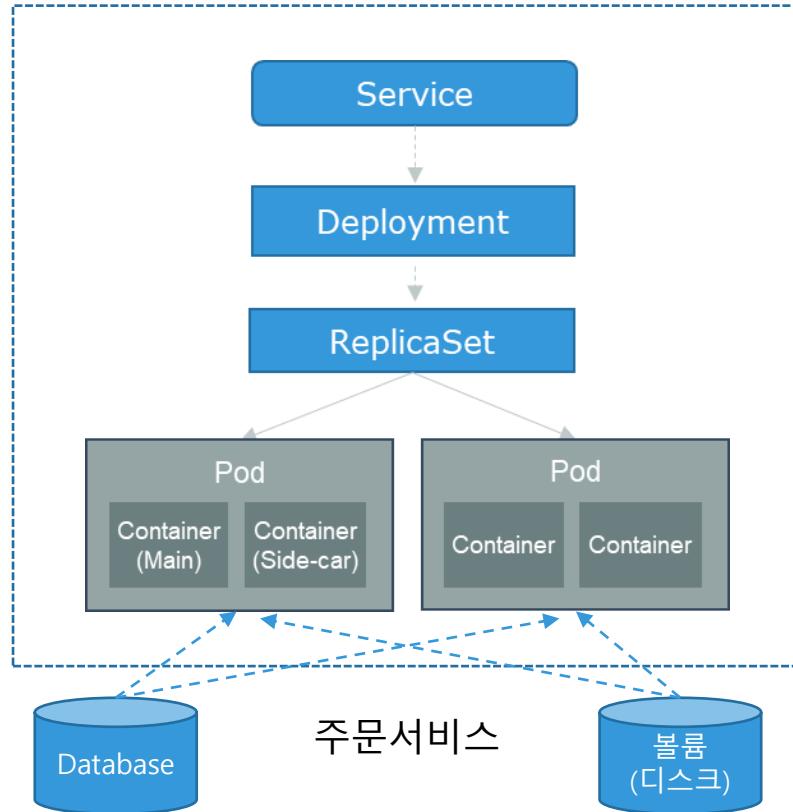
DevOps Environment



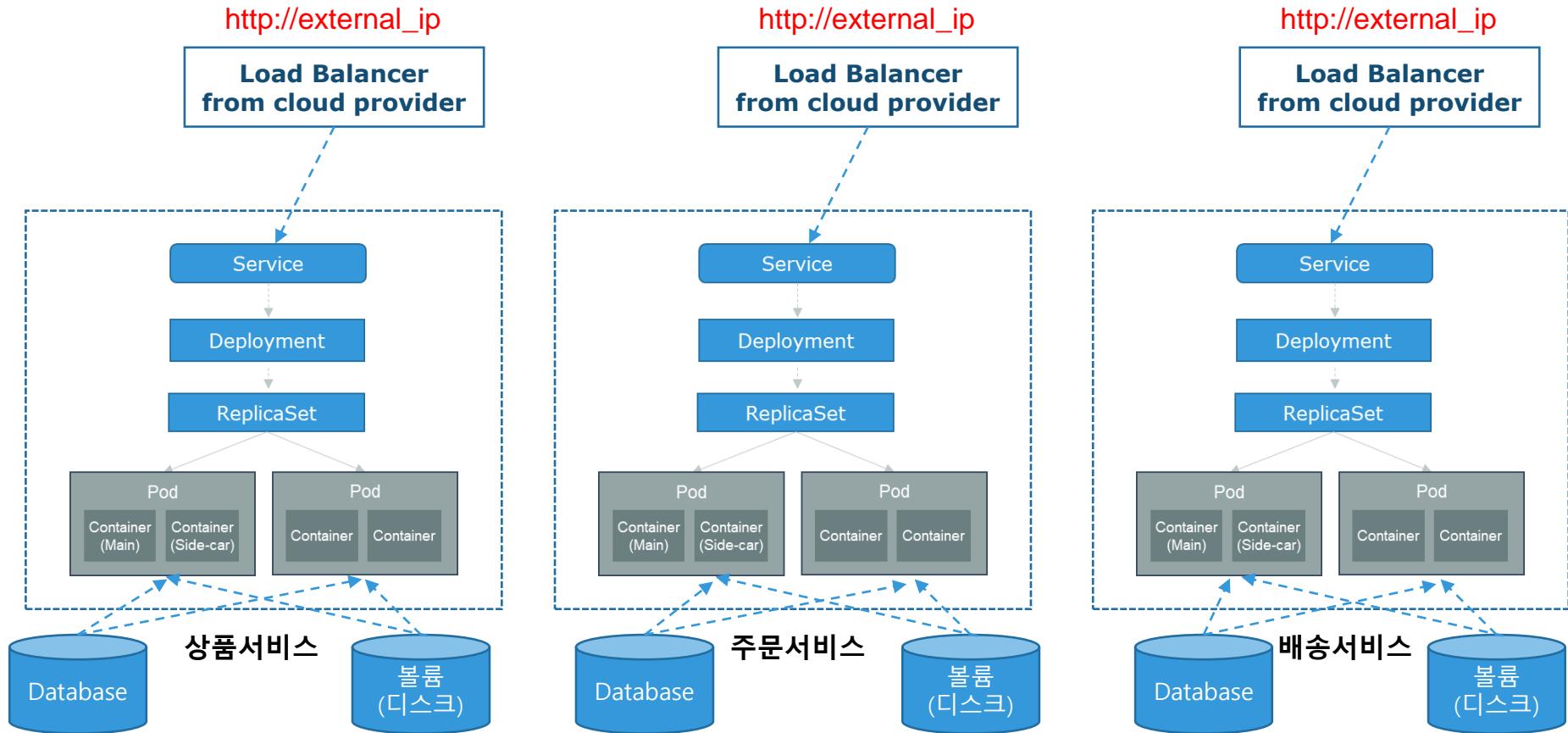
Kubernetes basic Service Model



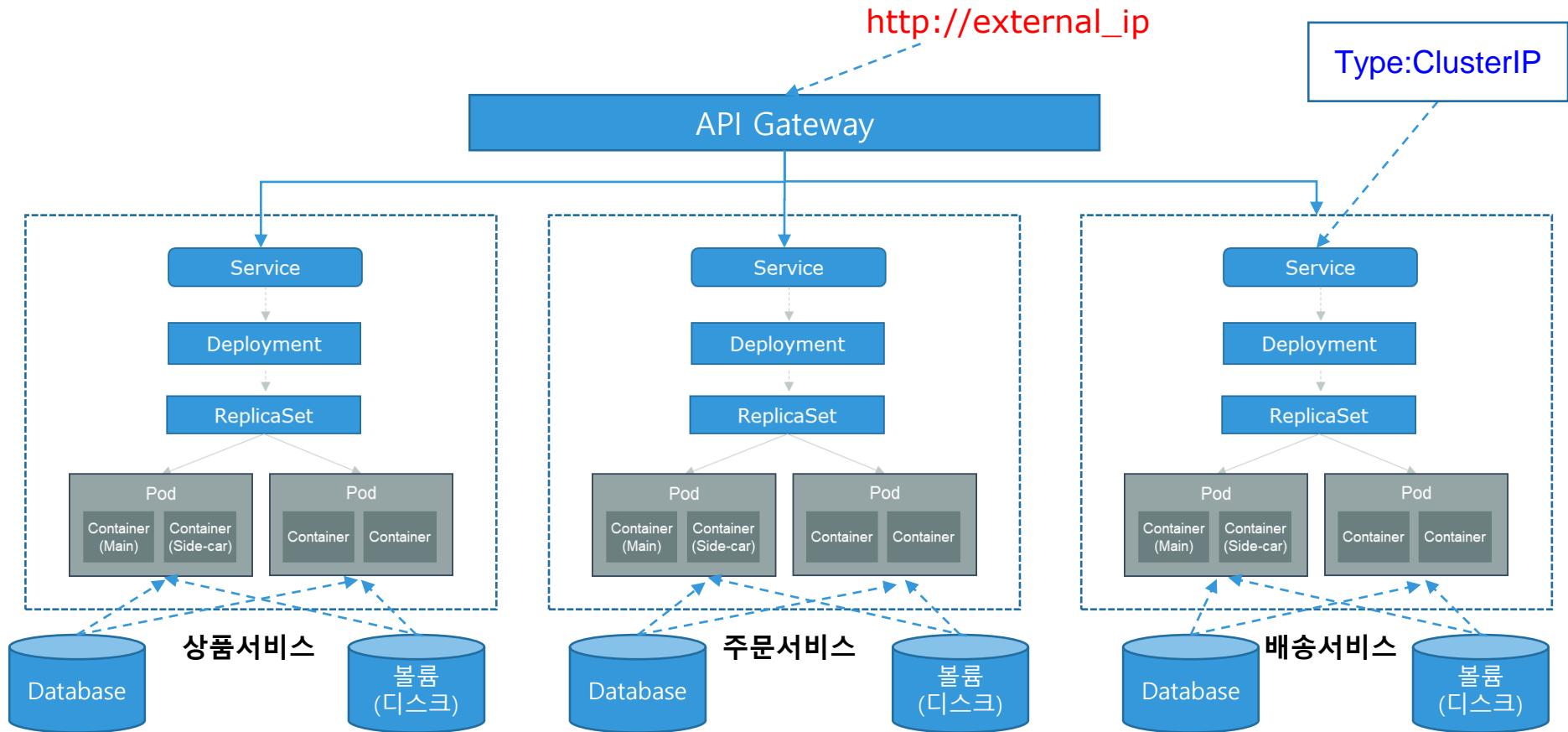
Kubernetes basic Service Model



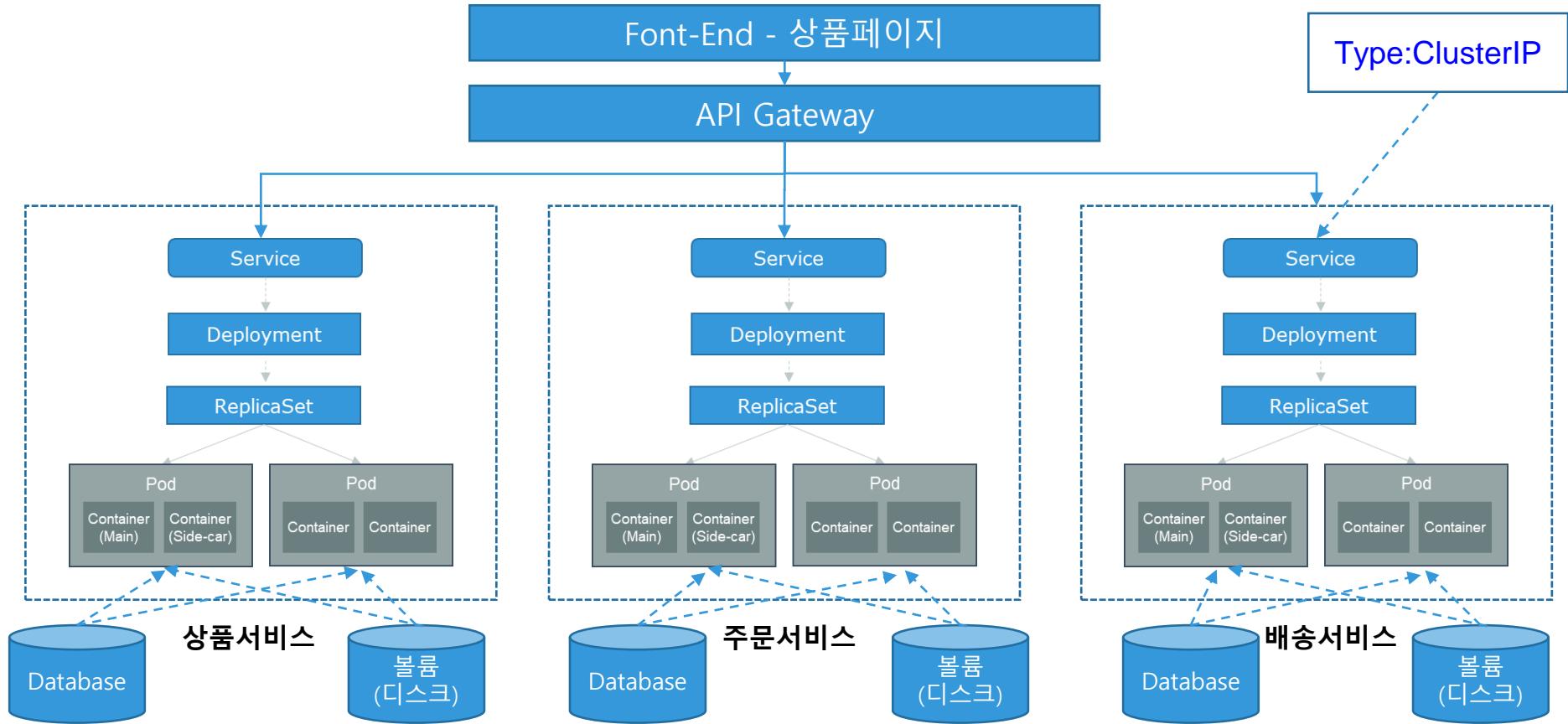
Kubernetes basic Service Model



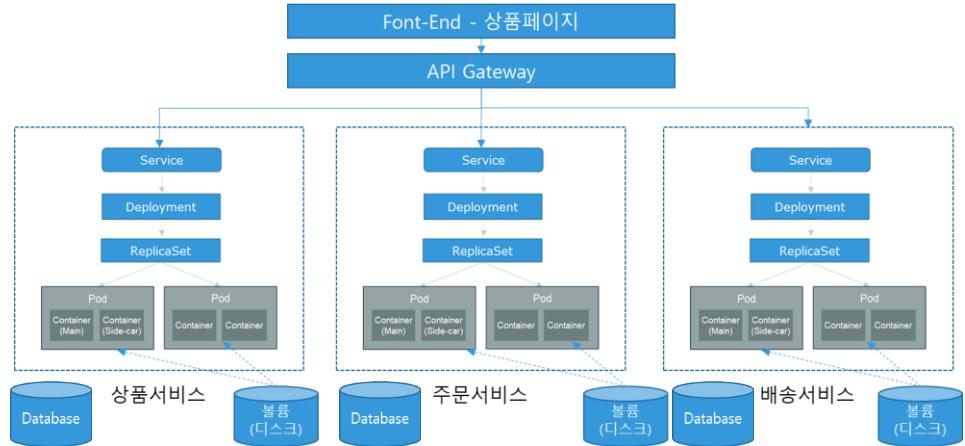
Kubernetes basic Service Model



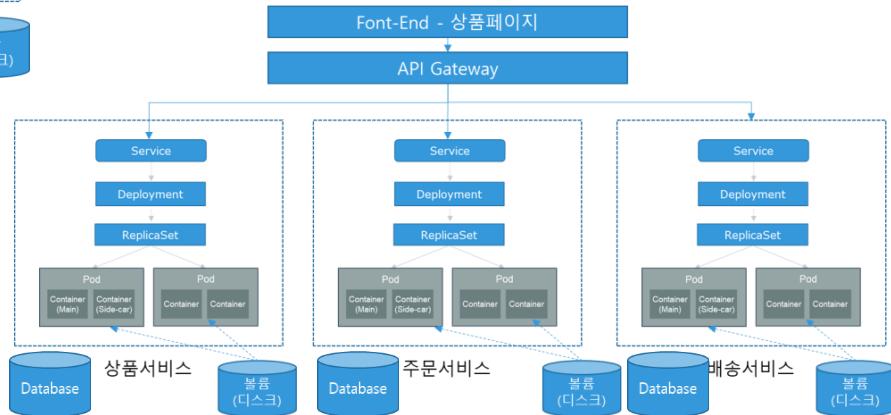
Kubernetes basic Service Model



Kubernetes basic Service Model



<Dev. 클러스터>



<Prod. 러스터>

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](http://pepa.holla.cz/wp-content/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

MSA Open Source Platform

a comprehensive tool designed to assist in the analysis, design, implementation, and operation of microservices.

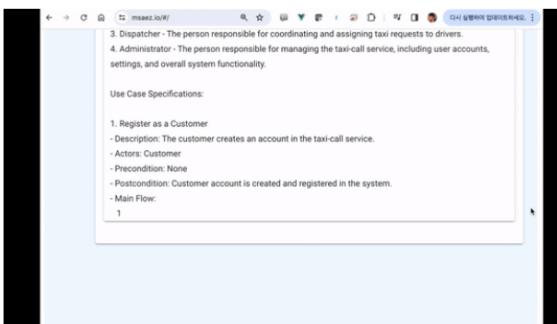
<https://github.com/msa-ez/platform>

README GPL-3.0 license

MSAez (Microservices made easy) Platform

MSA Easy (<https://www.msaez.io/>) is a comprehensive tool designed to assist in the analysis, design, implementation, and operation of microservices. It enables business experts and developers to collaboratively analyze and design software using domain-driven design and event-storming techniques. Furthermore, it facilitates the automatic generation of the "Clean-code". Throughout this entire process, ChatGPT can provide automation support and guidance on how to effectively utilize this platform.

Chat-GPT creates User Stories



The screenshot shows a browser window for 'msaez.io/#'. The page displays a user story for a taxi-call service. The story includes a list of actors and their responsibilities:

- 3. Dispatcher - The person responsible for coordinating and assigning taxi requests to drivers.
- 4. Administrator - The person responsible for managing the taxi-call service, including user accounts, settings, and overall system functionality.

Below this, under 'Use Case Specifications:', there is a numbered list:

1. Register as a Customer
 - Description: The customer creates an account in the taxi-call service.
 - Actors: Customer
 - Precondition: None
 - Postcondition: Customer account is created and registered in the system.
 - Main Flow:
 - 1

At the bottom of the page, the text 'Chat-GPT Auto Software Modeling (Event Storming and UML)' is visible.

MSA School

Provides end-to-end full-process learning including business modeling, implementation, and deployment

<https://www.msaschool.io>

The screenshot shows the homepage of the MSA School website. The header features the MSA School logo and navigation links for '소개' (About), '계획단계' (Planning Phase), '설계/구현/운영단계' (Design/Implementation/Operation Phase), '참고자료' (Reference Materials), '커뮤니티 및 교육' (Community and Education), and a search bar. The main content area has a purple header with the text '험난한 MSA 구축 여정의 길라잡이'. Below this, there's a section titled 'Biz Dev Ops' with a grid of six circles labeled '1단계 분석', '2단계 설계', '3단계 구현', '4단계 통합', '5단계 배포', and '6단계 운영'. A callout box highlights '계획에서 운영까지 End-to-End 학습을 위한 단계별 효율적인 실천법 제시'. At the bottom, there's a section about MSA School's mission and its focus on Microsoft services.

MSA School 소개

예제 도메인

사용 플랫폼

예제 애플리케이션 둘러보기

관련 링크

유필리티 및 도구

첨단한 MSA 구축
여정의 길라잡이

Biz Dev Ops

1단계 분석 2단계 설계 3단계 구현 4단계 통합 5단계 배포 6단계 운영

계획에서 운영까지 End-to-End 학습을 위한 단계별 효율적인 실천법 제시

MSA 스쿨에 오신 것을 환영합니다.

최근, 마이크로서비스를 활용한 비즈니스 구축ニ즈와 이와 관련한 사전 교육 및 컨설팅 요구는 계속적으로 증가하고 있는 추세에 있습니다. 특히, 대기업을 중심으로 모든 신규 프로젝트는 마이크로서비스 아키텍처를 우선 고려할 정도로 IT 시장에서의 MSA는 이미 깊숙히 자리 잡고 있습니다.

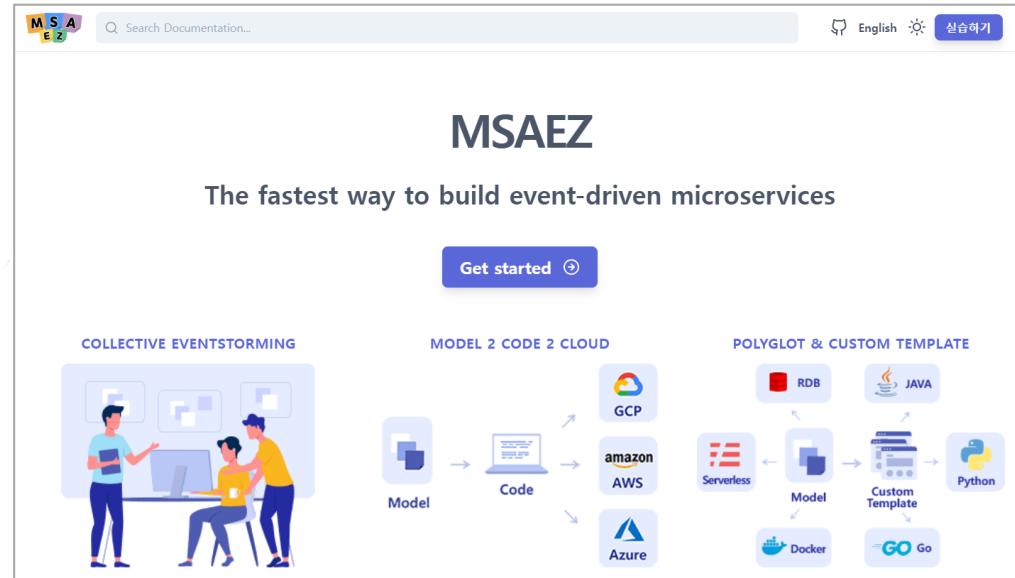
MSA School은 이러한 시장 상황에서 보다 구체적이고 실행 가능한 사례 중심의 구현 전략을 소개하려는 목적으로 설립되었습니다.

MSA 스쿨은 BizDevOps의 전 과정에 걸친 End-to-End 학습을 위한 단계별 실천법을 제공합니다.

Introduction MSAEZ

The fastest way to build event-driven microservices

<https://intro.msaez.io/>



THANKS!

Any Question?

You can find me at:

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