**Developing Cloud Native Applications**

3rd Jan 2020, ver2.0

Copyright © 2019. Jinyoung Jang & uEngine-solutions All rights reserved.

1

**Software System Architecture Journey**

**Main- Frame**

**Client / Server**

**Web**

**SOA**

**REST**

2

**Agile delivery**

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

**Company Deploy Frequency Deploy Lead Time Reliability Customer**

**Responsiveness**

Amazon 23,000 / day Minutes High High

Google 5,500 / day Minutes High High

Netflix 500 / day Minutes High High

Facebook 1 / day Hours High High

Twitter 3 / week Hours High High

Typical enterprise Once every 9 months Months or quarters Low / Medium Low / Medium

출처: 도서 The Phoenix Project

3

**Software System Architecture Journey**

**Main- Frame**

**Client / Server**

**Web**

**SOA**

**REST**

**Microservice**

4

An Enterprise Application or Suite

Monolithic Application

Customer Mgmt

Fulfillment

Payment

Inventory

Opportunit Mgmt

y Mgmt

**Database**

**First, What it is Not: A Monolithic Architecture**

Order Capture

5

**Anatomy of a Monolithic Architecture**

Good initially because interaction is easy

But, ease of interaction results in many inter-dependencies

Customer Mgmt

Over time, coupling becomes tighter and tighter

Change to one Component...

Fulfillmen ....require changes in many

t

Opportunity MgmtOrder Capture

Inventory Mgmt

Customer

Fulfillment Data Data Order Data

Payment

Payment Data Inventory

Data

Opportunity Data

6

Size matters, costs grow as they grow

Customer Mgmt

Order Capture

Fulfillment

Opportunity Mgmt

Inventory Mgmt

Customer

Data Fulfillment

Data Order Data

Opportunity Data

**Drawbacks of a Monolithic Architecture**

Payment Data Inventory

Data

Payment

Large code base

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

Deployment of any change requires redeploying everything

Only scales in one dimension

You are committed to the technology stack

Becomes an obstacle to scaling development

7

**In Contrast: A Microservice Architecture**

Service-oriented architecture of loosely coupled elements with bounded contexts

Break each function into separate deployment stacks

• Separate database

Custome r Mgmt

• Separate Servers running any technology

• Local or wide-area network

Order Capture

Order Capture

Customer Mgmt

Fulfillmen

Fulfillment t Payment

Payment

Inventory Mgmt

Inventory

Mgmt Opportunit

Mgmt y

Opportunity Mgmt

8

**Anatomy of Microservice Architecture**

Isolation is the name of the game

Service-oriented architecture

• Interaction only through HTTP/REST

Browsers / Mobile Apps

Payment

Order Capture

Order Capture

Inventory Mgmt Fulfillment

Loosely coupled elements

• Or asynchronous streaming / messaging

Customer Mgmt

Data Stream topic

9

**Loosely-Coupled Interaction via HTTP/REST**

REST APIs must be stable and hide internals

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

Client ignore parts of representations they don’t understand

In-bound adaptor

out- bound adaptor

Order Capture

Get / product availability

In-bound adaptor Stable JSON Representation

Inventory Mgmt

Internel Database Structure

Only additive changes allowed

10

**Loosely-Coupled Interaction via Asynchronous Messaging**

Data streaming can decouple database with shared data

Internal Database

In-bound adaptor

Asynchronous messaging(streaming)

Structure

out- bound adaptor

Order Capture

Customer Mgmt

Customer Mgmt

Customer Mgmt

• Operation in near real-time

• Operation in near real-time

• Decouples **“timing”**

In-bound adaptor

In-bound adaptor

In-bound adaptor

In-bound adaptor

out- bound adaptor

out- bound adaptor

Data Stream topic

Client ignore parts of the stream they don’t understand

Data streams hide the internal implementation of the service

In-bound adaptor

Fulfillmentout-

bound adaptor

Simplified and Stable JSON Representation

11

Smaller is better

Smaller, independent code base is easier to understand

IDE and app startup are faster

Simpler deployment and testing of just the changed service

Improved fault isolation

Not committed to one technology stack

Testing can be automated since all features are exposed as services

**Microservice Architecture benefits**

Monolithic**Customer Mgmt**

**Inventory** Inventory **Mgmt** Mgmt

Custome r Mgmt

**Order** Order **Capture** Capture

**Fulfillment** Fulfillmen

t **Payment**

Payment

**Opportunity** Opportunit

**Mgmt** Mgmt y

12

**Microservice Architecture**

Updating one service doesn’t require changing others

In-bound adaptor

Billing

In-bound adaptor

Limitation out- bound adaptor

Limitation out- bound adaptor

Good fault isolation

Good fault isolation

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

out- bound adaptor

Data Stream topic

out- bound adaptor

In-bound adaptor

In-bound adaptor

Customer

Smaller, simpler code base

Options for scaling

Options for scaling

Options for scaling

13

Distributed computing adds complexity and slow down initial development

In-bound adaptor

Order

out-

Capture

Dev tools not optimized for In-bound adaptor distributed services Customer

Mgmt

out-

bound adaptor • Potentially unreliable connections require bound adaptor

fault tolerance

Data Stream topic

out- bound adaptor

In-bound adaptor

Fulfillment

**Drawbacks to a Microservice Architecture**

Testing can be more complicated

Deployment and operations are more complex

Where/How to decompose the services?

Inter-service communication complicates development

Maintaining consistency with distributed transactions is hard

What about inter-service security? Identity management?

14

**Micro Service Architecture**

• 변경된 서비스만 재배포 → Side effect 최소화

• 자율성 → 각 서비스에 대한 자유로운 언어, 아키텍처, 아웃소싱 용이

• 병렬 개발, 타임 투 마켓, 린 개발

15

**Micro-Service Architecture Patterns** • microservices.io

16

**Netflix**

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

17

**마이크로 서비스 전환 사례**

Video & Broadcasting

From Mobile Apps and

Pure Play Video OTT- A

Monolithic to Serverless Microservices

Microservices

IoT Service Serverless Microservices

Gaming Platform

18

• 가장 작고 독립적인 서비스로 부터 마이그래이션

• 순차적 레거시의 마이크로서비스 전환을 위한 API GW 적용 한 “스트랭글러 패턴” 적용

• 서비스 폴백 테스트 및 자동화가 필수 조건

19

• 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

20

**Tip: monolithic and MSA**

**monolithic MSA**

Aggregation (데이터 통합) Backend 가 주도 Front 가 주도

Database 통합 데이터베이스 서비스 별 데이터베이스

필수 환경 WAS DevOps, PaaS (Grid Engine)

서비스 굵기 업무 비즈니스 기능별 구현 팀별, 10000 라인 이하로?, 관심사

별

Front 기술 JSP, Struts 등 Server-side rendering MVVM, AJAX 등 Client-side rendering

Container / Packaging WAS / WAR Spring-Boot, Docker

21

**Comparison with SOA**

Source: pwc

22

**MSA Trade-offs (from Martin Fowler)**

23

**MSA Trade-off**

24

**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/

25

1. The Domain Problem : A Commerce Shopping Mall

2. Architecture and Approach Overview

3. Domain Analysis with DDD and Event Storming

4. Service Implementation with Spring Boot and Netflix OSS

5. Monolith to Microservices

6. Front-end Development in MSA

7. Service Composition with Request-Response and Event-driven

8. Implementing DevOps Environment with Kubernetes, Istio

**Microservice and Event-storming-Based DevOps Project Table of content**

26

**Target Domain : Online Shopping Mall** (12 STREET)

**- Vision & Mission**

**Service**

24시간 365일 접속과 주문이 가능 **resiliency**

: 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

27

**Organization & KPI Definition - 창업시기**

**상품 품절로 인한 주문 취소율 0% 달성**

**높은 주문 성공율 (예외율, 배송 문제 등 최소화)**

**배송관련 질의, 재방문 비율 증가**

**불만 요청 수 최소화**

**창고 비용 최소화**

**블랙컨슈머 최소화**

• Multiple Concerns Single Organization

• Horizontally aligned

***12 Street Team***

28

**User Stories**

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

고객에게 인폼하고 처리(재 입고시 지연 배송, 주문 취소) 한다.

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

29

**Separate Core Domain from Supporting Domain**

상품 품절로 인한 주문 취소율 0% 달성

높은 주문 성공율 (예외율, 배송문제 등 최소화)

**12 Street**

**Core Supporting**

배송관련 질의, 불만 요청 수 최소화

창고 비율

재방문 비율 최소화

증가

블랙컨슈머

최소화Core Supporting

30

**Separation of Concerns - 회사의 성장**

**<주문>**

**12 Street**

**<배송>**

높은 주문 성공율 (예외율, 배송문제 등 최소화)

배송관련 질의, 불만 요청 수 최소화

재입고 시, 구매 성공율

**<마케팅>**

**<상품>**

재구매 비율 증가

상품 추천율

상품 품절로 인한 주문 취소율 0% 달성

**<고객센터>** 창고 비용

최소화

블랙컨슈머 최소화

Core Supporting

31

**신설된 User Stories 및 Concerns Mapping**

**배송관리팀**

**상품관리팀**

**주문/결제관리팀**

**고객센터팀**

**마케팅팀**

5. A notification is sent to the customer whenever the

delivery status changes. 6. Customers who repeat orders and cancellations

frequently are blacklisted, and customers with high order purchase rates are whitelisted. 7. The marketing team mails recommendation products

tailored to the customers at 9 am daily.

1. The customer enters the number of products to order and clicks

the order button. 2. When an order is placed, the delivery team arranges for delivery

of the product. 3. As the order or cancellation is completed, the product stock is

changed(+/-). 4. Customers can also purchase products that are out of stock and

This can be resolved after registering products later

32

1. The Domain Problem : A Commerce Shopping Mall

2. Architecture and Approach Overview

3. Domain Analysis with DDD and Event Storming

4. Service Implementation with Spring Boot and Netflix OSS

5. Monolith to Microservices

6. Front-end Development in MSA

7. Service Composition with Request-Response and Event-driven

8. Implementing DevOps Environment with Kubernetes, Istio

**Microservice and Event-storming-Based DevOps Project Table of content**

33

**AS-IS: Pain-points**

• 서비스 업그래이드가 수시로 요청이 들어와 거의 매일 야근중. 개발자 행복지수가 매우 낮음.

• 한팀의 반영이 전체팀의 반영에 영향을 주어 거의 매일 야근해야 함. 행복지수 낮음을 토로함.

• 테넌트별 다형성 지원을 제대로 하지 못하여 가입고객이 늘 때마다 전체 관리 비용이 급수로 올라가는 한계에 봉착함

• 자체 IDC를 구성하여 하드웨어, 미들웨어 구성을 직접해야 하는 비용문제.

• 운영팀과 개발팀이 분리되어 개발팀의 반영을 운영팀이 거부하는 사례 발생

• 현재 미국, 일본, 유럽 등 수요가 늘어나는 상황이나, 상기한 문제로 신규 고객의 요구사항을 받아들이지 못하는 상황

• 수동 운영의 문제로, SLA 준수가 되지 못하여 고객 클레임이 높은편

• 기존 모놀로씩 아키텍처의 한계로 장기적인 발전의 한계에 봉착

• 개발팀은 새로운 요건을 개발했으나, 이로인해 발생하는 오류가 두려워 배포를 꺼려함

34

**What is Agile?**

**Planning is important!**

- Fail is bad - Cost-driven

**Agility is important**

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

35

**To be Agile**

**DELIVERY & REQUIREMENT MANAGEMENT**

**MVP** (Minimum Viable Product) & **Incremental delivery**

**S/W ENGINEERING Test driven Development**

& **Continuous Refactoring**

**TECHNICAL DESIGN Loosely Coupled Architecture**

**PROCESS & COLLABORATION Continuously Improving**

**Loose**

36

**Continuous Delivery**

**Amazon, Google, Netflix, Facebook, Twitter**는얼마나 자주 배포할까요?

**Company Deploy Frequency Deploy Lead Time Reliability Customer Responsiveness**

Amazon 23,000 / day Minutes High High

Google 5,500 / day Minutes High High

Netflix 500 / day Minutes High High

Facebook 1 / day Hours High High

Twitter 3 / week Hours High High

Typical enterprise Once every 9 months Months or quarters Low / Medium Low / Medium

출처: 도서 The Phoenix Project

37

**Monolithic Architecture**

→ **모든 서비스가 한번에 재배포**

→ **한팀의 반영을 위하여 모든 팀이 대기**

→ **지속적 딜리버리가 어려워**

38

**Jeff Bezos** Mandate

1. All teams will henceforth expose their data andfunctionality through service interfaces.

2. Teams must communicate with each other through these interfaces. **3. There will be no other form of interprocess communication allowed:**

**no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network. ...**

4. The team must plan and design to be able to expose the interface to developers in the

outside world. No exceptions. 5. Anyone who doesn't do this will be fired.

39

**Approach #1 : Micro Service Architecture**

**Contract based, Polyglot Programming**

→ **Separation of Concerns, Parallel Development, Easy Outsourcing**

X

장애전파

Written in Java

Written in Node

Written in PHP

**[ 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**

40

**Approach #2 : Event Driven Architecture**

ShippingOrders

Publishes

Subscribes

Marketing

Time-decoupled

**Order Placed**

Subscribes

Open in extensibility

41

**Approach #2 : Event Driven Architecture**

Polyglot-persistence

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

42

**주문팀 상품팀 배송팀 마케팅팀**

이벤트 Publish

이벤트 Subscribe

선호도수집 선호도수집 주문들어옴 (PO. No: 1, TV, 5개, 홍길동)

재고변경 (No:1, TV, 5)

배송준비

새벽 2시

추천상품 (No: 1)

저장

배송준비 (No: 2)

오전 9시추천상품

이메일 발송

**마이페이지**

(PO. 주문정보

(PO. 주문취소 수집 정보수집 No:1)

No:1)

id User Item Qty prc o-stat d-stat

ID Item Stock

1 TV 10 2 Radio 10

선호도수집

주문상태변경주문상태변경재고변경 (No:1, TV, 0)

재입고됨 (ID:1, TV, 10개)

001 홍길동 TV 5 50,000 cancel Delivery

Started 002 아무개 TV 5 50,000 order Delivery

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

주문상태변경

상품발송함 (주번:1)

재고변경 (No:1, TV, 15개)

배송수거 (No: 3)

상품수거됨 (주번:1)

재고수정 (No:2, Radio, 8)

MySQL MongoDB Redis HTML

File-System 주문들어옴 (PO. No: 2, TV, 5개, 아무개)

주문취소됨 (PO. No:1)

주문들어옴 (PO. No.: 3, Radio, 2개, 아무개)

Id user category cnt

id User Item Qty prc o-stat d-stat ID User Item Qty status 1 홍길동 TV 5 ordered ID Item Stock

1 TV 5 001홍길동 TV 5 25,000 Order 2 Radio 10

Id user category cnt 1 홍길동 Electronic 1

ID User Item Qty status

1 홍길동 TV 5 ordered 2 아무개 TV 5 ordered

Id user category cnt 상품발송함

1 홍길동 Electronic 1 (주번:2)

2 아무개 Electronic 1

배송준비 (No: 3)

(PO. 주문정보

(PO. 주문정보

수집

수집 No:2)

No:3)

변경정보수집

변경정보수집 (PO. 배송상태 (PO. 배송상태 (PO. 배송수거 정보수집

No:1)

No:2)

No:1)

id id id id id id User User User User User User Item Item Item Item Item Item Qty Qty Qty Qty Qty Qty prc prc prc prc prc prc o-stat o-stat o-stat o-stat o-stat o-stat d-stat d-stat d-stat d-stat d-stat d-stat ID ID ID ID ID ID User User User User User User Item Item Item Item Item Item Qty Qty Qty Qty Qty Qty status status status status status status

ID Addr PO-id Qty Status

001 001 001 001홍길동 001홍길동 홍길동 홍길동 홍길동 TV TV TV TV TV 5 5 5 5 5 50,000 50,000 50,000 50,000 50,000 cancel cancel Order

order order Delivery Returne Delivery Returne 1 1 1 1 1 홍길동 홍길동 홍길동 홍길동 홍길동 TV TV TV TV TV 5 5 5 5 5 Cancelle Delivery Delivery Returne Returne

Started Started d d Started Started d d d

002아무개 002 002 002아무개 002아무개 아무개 아무개 TV TV TV TV TV 5 5 5 5 5 50,000 50,000 50,000 50,000 50,000 order order order order order

Delivery Delivery Delivery 2 2 2 2 2 아무개 아무개 아무개 아무개 아무개 TV TV TV TV TV 5 5 5 5 5 Delivery ordered Delivery Delivery Delivery

Started Started Started Started Started Started Started

003 아무개 RADIO 2 20,000 order 3 아무개 Radio 2 ordered

ID ID ID ID Item Item Item Item Stock Stock Stock Stock

ID ID ID ID Addr Addr Addr Addr PO-id PO-id PO-id PO-id Qty Qty Qty Qty Status Status Status Status

1 1 1 1 TV TV TV TV 15 10 15 0 2 2 2 2 Radio Radio Radio Radio 10 10 10 8

1 1 1 1 강동구 강동구 강동구 강동구 1 1 1 1 5 5 5 5 Started Started Return Return 2 강서구 2 5 Started

ed ed 2 2 강서구 강서구 2 2 5 5 Started Started 3 강서구 3 2 prepar

e

43

**Event Driven MSA Architecture**

E E

**Monolith:**

• With Canonical Model

• (“Rule Them All” model)

**Reactive Microservices:**

• Autonomously designed Model(Document)

• Polyglot Persistence

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

44

**Approach #3: BizDevOps Process**

• **1 Sprint = 1~2 Weeks**

**Cycle 기반**

• **지속적인 Needs 탐색**

45

**적용 아키텍처**

프론트엔드

프론트엔드

IAM

Gateway

상주배마상품 주문 배송

품

문

송

케팅 (Java/

(Java/ Tomcat)

Spring)

(.NET)

(python)

데이터베이스

RDBRDBRDBNo SQL모놀리식

Message Channel

46

**효과 – 장애 최소화**

프론트엔드

상품 주문 배송

데이터베이스

장애가 전파되며, 수동 복구로 장애 지연

프론트엔드

상품

주문

배송

데이터

데이터

데이터 베이스

베이스

베이스

장애가 격리되며, 자동으로 복구됨(이전버전)

47

**효과 – 기능 확장**

프론트엔드

Java/J2EE 마상품 주문 배송 케팅

데이터베이스 스키마 체인지

기존 코드의 수정 → Big Impact

프론트엔드

RDB상품 (Java/ Tomcat)

RDB주문 (Java/ Spring)

RDB주문 (Java/ Spring)

RDB배송 (.NET)

RDB배송 (.NET)

RDB배송 (.NET)

No SQL마케팅 (python)

No SQL마케팅 (python)

No SQL마케팅 (python)

No SQL마케팅 (python)

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

48

**효과 – 기능 확장**

프론트엔드

Java/J2EE 마상품 주문 배송 케팅

데이터베이스 스키마 체인지

기존 코드의 수정 → Big Impact

프론트엔드

상품

(Java/ Tomcat)

주배마문 송

케팅 (Java/ Spring)

(.NET)

(python)

RDB

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

49

**효과 – 성능 배분**

프론트엔드

상품 주문 배송 마케팅

데이터베이스

스케일업을 통해서만 확장

프론트엔드

상품

주배문

송 마케팅

주문결제

‘상품추천’ 워크로드에 대한 스케일 아웃 용이

50

1. The Domain Problem : A Commerce Shopping Mall

2. Architecture and Approach Overview

3. Domain Analysis with DDD and Event Storming

4. Service Implementation with Spring Boot and Netflix OSS

5. Monolith to Microservices

6. Front-end Development in MSA

7. Service Composition with Request-Response and Event-driven

8. Implementing DevOps Environment with Kubernetes, Istio

**Microservice and Event-storming-Based DevOps Project Table of content**

51

**DDD Patterns**

52

**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*

53

**Bounded Context** (한정된 맥락)

**Ubiquitous Language (도메인 언어)**

54

**Bounded Context and Ubiquitous Language**

**SW CONSTRUCTION**

A Project

An Architecture

A Developer

55

**In Monolith :**

**Single Deployment Unit** / **Single Server Runtime** / **Single Language and Platform**

Construction Project Management (core)

**Project**

**Developer**

**Architecture** SW Project Management

(supportive)

**SW Project**

**SW**

**SW Developer**

**Architecture**

56

**In Microservices :**

Construction Project Management (core)

SW Project Management (supportive)

**Project**

**Developer**

**Architecture** Anti-

corruption layer

**Project**

**Developer**

**Architecture**

57

**In Microservices :**Construction Project Management (core)

Project Started

**Project**

**Project** A Software Project Also Start

SW Project Management (supportive)

**Developer**

**Architecture**

**Developer**

**Developer**

**Architecture**

**Architecture**

**Architecture**

58

**5개의 눈금: 어느 굵기로 쪼갤 것인가? 그것이 문제로다...**

**Aggregate Bounded context Business capability Mini Monolith** 최소 사이즈 –

ACID 가 준수되어야 하는 원자적 단위 - 유연성이 높은

커뮤니케이션이 원활한 단위 – Separation of Concerns 이 좋은

비즈니스적 성과 측정과 피버팅이 용이한 단위 – 팀단위와 연결하기 좋은

구현이 용이한 단위 - 점진적 MSA 전환이 용이한 단위

59

**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 시스템 을 분석에서 개발까지 필요한 도메인에 대한 탁월하게 빠른 이해를 도모**하는데 유리하다.

• 기존의 유즈케이스나 클래스 다이어그래밍 방식은 고객 인터뷰나 엔티티 구조를 인지하는 방식과 다르게 별다른 **사전 훈련된 지식과 도구 없이 진행할 수 있다**.

• 진행과정은 참여자 워크숍 방식의 방법론으로 결과는 **스티키 노트를 벽에 붙힌 것으로 결과가 남으며, 오랜지색 스티키 노트들의 연결로 비즈니스 프로세스가 도출**되며 이들을 이후 BPMN과 UML 등으로 정재하여 전환할 수 있다.

**vs**

60

**Event Storming : Prepare**

• 3팀 이상으로의 구성 및 팀당 최소 2명 이상 구성

• 큰 종이 시트 및 종이 시트를 여러 장 붙일 수 있는 충분한 벽이 있는 넓은 공간

• 여러 종류의 색깔 스티커, 검은 색 펜, 검은색 or 파란색 테이프

• 서서 하는 방식으로 의자 필요 없음.

61

**Type of stickers**

Domain

**P.P 형태의 동사**

**사용자, 페르소나, 스테이크 홀더 정의**Event

도메인 전문가가 정의

도메인에 대한 용어 등의 (Orange)

이벤트 퍼블리싱

설명,기술

**집합체** Command (Sky Blue)

**리드모델**

**외부 시스템**

시스템 호출을 암시 (REST)

이벤트에 대한 반응 (서브스크라이브) 비즈니스 룰 엔진 등

Definition

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

유저 인터페이스를 통해 데이터를 소비하고 명령을 실행하여 시스템과 상호 작용

Aggregate (Yellow)

Read Model (Green)

Actor

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

Comment Or Question (Purple)

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

External System (Pink)

**업무정책** Policy (Lilac)

**의견 또는 질문**

추기적인 네용 입력, 예측되는 Risk

62

**Examples**

order

Product Search Order Placed

Actor

Delivery Start

Delivery

Delivery Completed

**[Meta Model] [12st Sample]**

Command

Policy

Domain Event Read Model

External System

**Delivery-Adm**

Customer

Order

Delivery Start

External System

Delivery Started **Logistics**

63

**Firstly, Event Discovery**

• 오렌지(주황색) 스티커 사용

Order Placed

• 각 도메인 전문가들이 개별 도메인 이벤트 목록 작성

Order Cancelled

Order Cancelled

Order Modified

Order Modified

Order Modified

• 이벤트는 도메인 전문가와 비즈니스 관계자 서로 이해할 수 있는 의미 있는 방식으로 표현하고 동사의 과거형 (p.p.)으로 표현한다.

• 시작 및 종료 이벤트를 식별하고 스티커를 붙일 벽의 시작과 끝의 타임라인에 배치

• 이벤트를 페르소나와 관련시키는 방법에 대해 논의

• 중복된 이벤트를 발견하면 중복된 이벤트를 벽에서 제거

• 불분명한 경우 다른 색상의 스티커 메모를 사용하여 질문이나 의견을 추가(빨간색 스티커)

• 이벤트에 대해서 동사를 과거 시제로 입력하고 다른 이벤트와 명확하게 구분되는 용어를 사용

64

**잘 도출된 이벤트와 아닌 경우**

상품주문이 발생함

**O X** 상품이

상품정보 입고됨

변경됨

**다른 팀에서**

**다른 팀에서**

**적당한**

It’s a command It doesn’t make **관심 가질만함**

**관심 가질만함**

**사이즈임**

any state change

Too technical and too fine-grained

Be business level

Nobody will be interested in this event

65

**Command 와 Actor, UI 도출**

• Command는 파란색 스티커 메모를 사용

• 도메인 분석에서 시스템 설계의 첫 단계

• 관심있는 비즈니스 프로세스를 구현하는 시스템을 구축하려면 이러한 이벤트가 발생하는 방식에 대한 질문 등으로 정의한다.

• 목표는 이벤트가 효과를 기록하는 원인을 찾는 것

• 예상되는 이벤트 트리거 유형의 예

- Operator 결정을 내리고 명령을 내릴 시 - 외부 시스템 또는 센서가 자극될 시 - 정해진 시간 경과 시

• 마이크로 서비스 구현에서 API가 될 수 있다.

• Command를 사용하는 인간 혹은 주체는 노란색 스티커 메모를 사용한다.

66

**Policy 도출**

• 라일락 색의 스티커 사용

• Policy는 이벤트가 발생한 후 발생하는 반응형 논리

• Policy는 다른 Command를 트리거

• Policy는 process 같이 사람이 행하 는 수동 동작 및 자동화 될 수 있다.

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

Account Created

Send Email

Confirm Payment Details

Email Sent

Email Sent

Payment Details Confirmed

Payment Details Confirmed

67

**Read Model 도출**• 녹색 스티커 사용

• 이벤트를 생성하기 위해 Command을 실행하는데 필요한 데이터의 이해

• The Data needed in order to make that decision

사용자의 의사 결정 과정을 지원하는 도구로 사용됨

• 각 Command 및 Event에 대해 필요한 속성 및 데이터 요소에 대한 설명

**“ Read Model Derivation ”**

68

**Aggregate 도출**

• 노란색 스티커 사용

Create

• 같은 Entity를 사용하는 연관 있는 도메인

Account 이벤트들의 집합

• 관련 데이터 (Entity 및 value objects)뿐만 아니라 해당 Aggregates의 Life Cycle에 의해 연결된 작업(Command)으로 구성

Update Account

Delete Account

Account Created

Account Updated

Account Deleted

69

**Bounded Contexts 도출**

• 이벤트의 내용을 정의하고 시스템의 경계를 구분

Payments

• 찾는 방식은 두가지 유형으로 구분되어진다

- Time Boundary - Subject Boundary

Shipping

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

70

**Lab Time – Event 들을 먼저 도출**

상품등록됨 상품재고

변경됨

상품삭제됨

주문생성됨 주문정보

변경됨

주문취소됨

배송출발함

배송취소 요청됨

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

배송준비됨

상품정보 변경됨

배송완료됨 주문상태 변경됨

71

**Lab Time – Policy 도출**

상품등록됨 상품재고

변경됨

홍보 메일발송

주문생성됨 주문정보

변경됨

입고 메일발송

배송생성

배송정보 변경재고변경

재고변경

상품정보

상품삭제됨 주문취소됨 주문상태 변경됨

변경됨

배송취소

재고변경

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

배송출발함 배송준비됨

배송지시 알림메일

발송

배송완료됨

배송취소 요청됨

알림메일

발송 회수지시

72

**Lab Time – Command 도출 (쓰기행위)**

상품등록

재고변경

결제클릭 정보변경 주문생성

출발수신 수신 상품등록됨

상품재고 변경됨 주문생성됨 주문정보

변경됨

배송준비됨 배송출발함 홍보

입고 메일발송

메일발송 배송생성 배송정보

변경 알림메일

배송지시

발송

재고변경 재고변경

정보변경

상품삭제 주문취소

주문취소 완료수신

상품삭제됨 주문취소됨 배송취소

요청됨

재고변경

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

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

주문취소 수신 상품정보 변경됨

주문상태 변경됨

배송완료됨

배송취소 알림메일

발송 회수지시

73

MD

MD

**Lab Time – Command Actor 식별**

재고변경

상품재고

변경됨입고

메일발송 상품등록

결제클릭 정보변경 상품등록됨주문생성됨 주문정보

변경됨

홍보 메일발송

배송준비 배송정보

재고변경 변경 재고변경

정보변경

상품삭제 주문취소 주문취소 상품정보

상품삭제됨 주문취소됨 주문상태 변경됨

변경됨

배송취소

재고변경

Actor 는 Command를 발생시키는 주체 , 담당자 또는 시스템 (외부 (External) 또는 내부 )이 될 수 있음

MD

MD

사용자

사용자

사용자

사용자

시스템

시스템

시스템 주문생성

출발수신 수신 배송출발함 배송준비됨

배송지시

알림메일 발송

완료수신

시스템

주문취소

배송완료됨

수신

배송취소 요청됨 알림메일

발송 회수지시

74

MD

MD

MD

MD

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

**Lab Time – Aggregate 도출**

상품상품등록됨

결제주문생성

배송준비됨 등록

클릭

수신

재고변경

정보변경출발수신

재입고 메일발송 완료수신정보주문변경 취소상품상품삭제됨

주문취소 삭제주문수신취소주문생성됨

홍보 메일발송

사용자

배송생성 배송지시

주문정보 변경됨

알림메일 상품

발송 (Product) 주문

(Order)

배송출발함 상품재고 변경됨

사용자

상품정보 변경됨

사용자

주문상태 변경됨

사용자

재고변경

배송정보

변경재고변경

배송완료됨

주문취소됨

알림메일 발송 배송취소

재고변경

배송취소 요청됨

회수지시

시스템

시스템

시스템

시스템

배송 (Delivery)

75

**Lab Time – Bounded Context 도출**

상품관리 주문관리 배송관리

상품등록

재고변경

상품등록됨

정보변경

상품상품삭제됨 삭제주문생성됨 배송생성

주문정보

변경됨배송정보변경

주문취소됨

배송취소

주문상태 변경됨주문생성 수신

배송출발함

상품정보

결제클릭

홍보 메일발송

정보변경상품 (Product)

변경됨주문취소주문취소배송준비됨 MD

배송지시

재고변경

상품재고 변경됨

출발수신

알림메일

입고 메일발송 MD

주문 (Order)

배송 (Delivery)

발송

재고변경 완료수신재고변경

주문취소

배송취소 수신요청됨

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

• **구현관점:** 어그리게잇은 *ACID* 트랜잭션 범위이기도 하기때문에 이를 더 쪼개서 *BC* 를 구성할 수는 없다*.*

*BC* 내의 어그리게잇들에 대한 보존 *(Persistence)*은 아마도 그 목적에 맞는 최적화된 데이터모델을 독립적으로 설계하고 구현할 수 있다*.*

배송완료됨

알림메일

MD

발송

MD

회수지시

사용자

사용자

사용자

사용자

시스템

시스템

시스템

시스템

76

시스템

시스템

시스템

**12번가 Shopping Mall Bounded Context**

출발수신

완료수신주문취소 수신**REAL CASE**

마케팅관리상품관리 주문관리

배송관리

재고변경

추천상품 저장

고객센터

고객별 주문이력 분석 추천상품 이메일발송

MD

상품상품등록됨 등록

결제주문생성됨 클릭

배송생성

주문생성

배송준비됨 수신 주문생성

주문이력

배송지시 수신

주문이력 추가됨 AI분석

요청지시

정보변경주문정보

변경됨배송출발함

알림메일 배송정보

발송 변경

정보변경

홍보 메일발송

메일로그

상품 (Product)

상품정보 변경됨재고변경 주문정보 조회

MD

재고상품재고 변경

변경됨입고

메일발송

주문주문취소됨 취소배송취소

배송취소

상품상품삭제됨 삭제주문주문상태

취소변경됨요청됨

설문제출

배송 주문

(Delivery) (Order)

재고변경

배송완료됨 MD

알림메일 발송

추천MD

회수지시 상품AI

주문이력

조회추천상품

설문등록

설문등록됨

블랙리스트 등록됨 추천상품AI

설문조사

설문진행 메일발송 (Survey)

고객관리

설문응답

등록됨화이트리스트

등록됨등록

VIP 메일발송

메일발송

담당자

담당자

사용자

사용자

사용자

사용자

시스템

시스템

시스템

시스템

시스템

77

**12st Microservices Event Sample :**

**One ‘Event’ is similar to a Chat in Chatting Room.**

**Chatting Room**

78

1. The Domain Problem : A Commerce Shopping Mall

2. Architecture and Approach Overview

3. Domain Analysis with DDD and Event Storming

4. Service Implementation with Spring Boot and Netflix OSS

5. Monolith to Microservices

6. Front-end Development in MSA

7. Service Composition with Request-Response and Event-driven

8. Implementing DevOps Environment with Kubernetes, Istio

**Microservice and Event-storming-Based DevOps Project Table of content**

79

**Microservice Implementation Pattern (1)**

**도메인 언어 영역**

• **Hexagonal Architecture**

(Ubiquitous language)

Inbound adapter

Inbound adapter 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

**분리**

messaging pattern: As events arrive at a

Some Controller class

Foo service

**비즈니스 로직**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.

**기술적 구현영역**

DAO

Outbound port

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

Outbound port

Database

Domain Event

Repository interface

Outbound adapter

Message consumer

Message producer

Message broker

80