Journal of Korea Multimedia Society Vol. 23, No. 2, February 2020(pp. 328-342)
https://doi.org/10.9717/kmms.2020.23.2.328

Cloud Native환경에서의 생산성 향상을 위한 어플리케이션 개발 방법 연구

김정보†, 김정인††

A Study of Application Development Method for Improving Productivity on Cloud Native Environment

Jung-Bo Kim†, Jung-In Kim††

ABSTRACT

As the cloud-based ICT(Information & Communication Technology) infrastructure matures, the existing monolithic software development method is evolving into a micro-service structure based on cloud native computing. To develop and operate the services efficiently under the cloud native environment, DevOps-based application development plans through MSA(Micro Service Architecture) design based are essential. A cloud native environment is an approach to developing and running applications that take advantage of cloud computing models such as automation of source distribution, container-based virtualization, application scalability, resource efficiency, and flexible maintenance through object independence. To implement this approach, the utilization of key elements such as DevOps, continuous delivery, micro service, and containers is essential, but there are not enough previous studies on case analyses or application methods of these key elements. Therefore, in this paper, we analyze the cases of application development in cloud native environment and propose the optimized application development process and development method through small and medium-sized SI projects.

Key words: MSA, Cloud Native, Project , SW Development, SaaS

1. 서 론

모든 소프트웨어 개발사들은 생산성에 대한 고민이 있으며, 그 고민을 해결하기 위해 소프트웨어 공학이 발달해 왔다. 수학, 물리학, 철학과는 달리 소프트웨어 공학은 역사가 70년 밖에 되지 않아 아직 정립되지 않고 변화해야 할 지식들이 많이 있다. 1969년에 시작된 구조적 프로그래밍에서 1980년대에 구조적 시스템분석과 설계방법론, 1990년대의 객체지향 개발방법론, 그리고 2000년대 들어서 애자일 개발방법론의 대두로 SOA(Sevice Oriented Architecture)시대가 열렸다. 그리고 2010년도는 4차 산업 혁명이란 화두로 대표되는 기술적 이슈인 클라우드 컴퓨팅의 출현과 환경, 문화적 이슈인 고객 경험을 새롭게 정의한 사용자 중심의 서비스 플랫폼등이 확장됨에 따라 기존의 전통적인(Monolithic) 서비스 합성은 오픈API 기반의 요소 기능들을 하나의 연결된 서비스로 합성하는 MSA(Micro Service Architecture) 구조로 바뀌고 있다[1,2]. 기존의 전통적(monolithic) 구조의 서비스 개발에서는 새로운 기능을 추가하거나 개선하기 위해 전체 서비스를 다시 빌드하여 배포해야하기 때문에 구현 대상 서비스가 복잡하

※ Corresponding Author : Jung In Kim, Address: (608-711) Dept. of Computer Engineering Tong-Myong University Sinsun-ro 428, Nam-gu, Busan, Korea, TEL : +82-51-629-1174, FAX : +82-51-629-1169, E-mail : jikim@tu.ac.kr

Receipt date : Oct. 14, 2019, Revision date : Jan. 7, 2020
Approval date : Jan. 31, 2020

† Dept. of Computer Media Engineering, Tong-Myong University (E-mail : jb7541@hanmail.net)
†† Dept. of Computer Engineering, Tong-Myong University

게 되어 유연성과 자원 활용 측면에서 효율성 저하 문제가 나타난다. 또한 일부 요소의 장애에 대해 분리된 대처가 어려워서 전체적인 시스템 마비로 이어질 수 있기 때문에, 장애 발생에 대한 대처도 힘들게 된다. 반면 기존의 전통적 구조와는 다르게 자율적으로 관리되는 마이크로 서비스 구조와 서비스 개발에서는 요소기능별로 해당되는 마이크로 서비스들이 정의된 오픈 API를 통해 다른 요소기능들과 통신하는 단일 목적의 기능 단위로 구성되는 모듈화 방식이기 때문에 전체 서비스를 독립적이고 간단한 기능으로 분리하고 기능 간 연결을 통한 서비스 통합이 가능하다. 이러한 모듈화 기법은 전체적으로 서비스 개발과 시험에서 편이성과 효율성이 높은 것으로 보고되고 있다[3].

이렇듯 MSA에 관한 필요성이 제기되고 있음에도 불구하고 국내에서는 서비스 기능별 분류기준과 DB 구조의 설계 방식에 대한 선행연구가 부족한 상황으로 해외 선진 사례인 넷플릭스 프로젝트나 Sales force.com, 아마존, 마이크로 소프트, IBM 등의 클라우드 사업자들의 프로세스를 도입하고 있는 실정이다. 또한 대부분의 국내 기업들이 클라우드 인프라를 도입하지만 경험과 선행연구 부족으로 MSA 방식이 아닌 기존의 전통적인 개발방식으로 구현함으로써 Cloud Native 환경의 장점을 제대로 활용하지 못하고 있다. 이러한 국내 현실은 IT개발의 유연성, 확장성, 생산성 저해를 야기하고 세계적인 추세인 Cloud 환경 확산을 저해하며, Cloud 서비스 기반 기술인 공유경제나 인공지능(빅데이터) 서비스 산업에도 부정적 영향을 미칠 수 있다.

본 연구는 중소규모의 소프트웨어 개발에 대해 MSA적용 방법을 실제 프로젝트에 적용해 보고, Cloud Native 환경에서 생산성 및 품질을 분석하고자 한다.

2. 관련연구

소프트웨어 공학에서의 생산성에 대한 고민은 1969년부터 2000년대 까지 쭉 이어져 왔다. 하지만 SOA는 죽었다 라고 말하는 개발자들이 있듯이 생산성 향상은 항상 숙제로 남아왔다. 이러한 과정을 겪어오면서 2010년도에는 Cloud 라는 가상화 가 등장했고 객체지향과 SOA 그리고 CBD 개발방법론과 애자일 개발방법론 등이 다시 화두가 되고 있다. 본 논문에서는 2010년 이전의 개발방법을 지칭하는 Monolithic 개발방법과 Cloud 환경에서의 개발방법인 MSA(Micro Service Architecture)방법에서의 생산성 향상을 위한 분석, 설계, 구현, 테스트 과정을 체계화 할 수 있는 방법을 연구하였다.

2.1 전통적 개발 방법(Monolithic)

전통적 개발 방법이란 클라우드 환경 이전의 개발들 즉, 엔터프라이즈 아키텍처 기반의 개발과 엔터프라이즈 아키텍처 개발의 단점을 보완한 객체지향 개발방법과 그리고 품질 강화를 위한 SW Visualization 기반의 개발방법을 통틀어서 말한다. 전통적 방법은 MSA의 반대되는 말로 사용되어 왔는데 본 연구에서는 기존 SI업체들이 클라우드 환경 이전에 사용한 전통적인 개발방법을 통칭한다. 앞서 언급된 방법 중 엔터프라이즈 개발방법은 고객의 요구사항을 받아 DB 및 개발 소스를 하나로 만드는 것이다.

이 방법의 장점은 통합적인 관점에서 운영이 편하다는 것이지만, 장기적으로 봤을 때 유연성이 떨어지고 향후 업그레이드 및 수정 부분에서도 문제가 발생한다. 또한 고객 요구에 의한 수정이 빈번하다보니 프로그램의 재사용성이 떨어져 개발 생산성이 저하되며, 더 나아가 개발 소스의 품질 문제로 이어지게 된다. 프로젝트 규모가 클수록 많은 개발자가 필요하게 되며 처음 설계단계 이후에는 개발자 역량에 따라 프로그램의 질이 좌우되기 때문에 이로 인해 유지보수 및 운영이 어려워지는 사례가 많이 발생한다. 이런 단점을 보완하기 위해서 나온 방법이 객체지향 개발방법이다. 객체지향 개발방법은 CBD(Component Based Development)라고도 하며, 소프트웨어의 재사용성을 높이기 위해서 컴포넌트 단위별로 쪼개서 개발하고 관리하는 방법이다. 객체지향 개발 방법은 엔터프라이즈 개발방법보다 운영 및 소스의 재사용율을 높여 생산성을 향상시키는 장점이 있으나, 로직과 데이터베이스가 따로 떨어져 있어 둘을 함께 복제해야만 작동하는데, 데이터베이스 전체를 복제해야 하는 상황이라면 사실상 엔터프라이즈 개발방식과 차이가 없게 되어, 클라우드 환경에서의 운영방식과는 큰 차이가 있다.

SW Visualization 기반의 Tool-Chain을 활용한 개발방법은 Cloud Native의 핵심 요소인 DevOps 환경개발과 가장 비슷하지만, 로직과 데이터베이스 설계부분은 앞서 설명한 2가지 개발 방법과 사실상 동

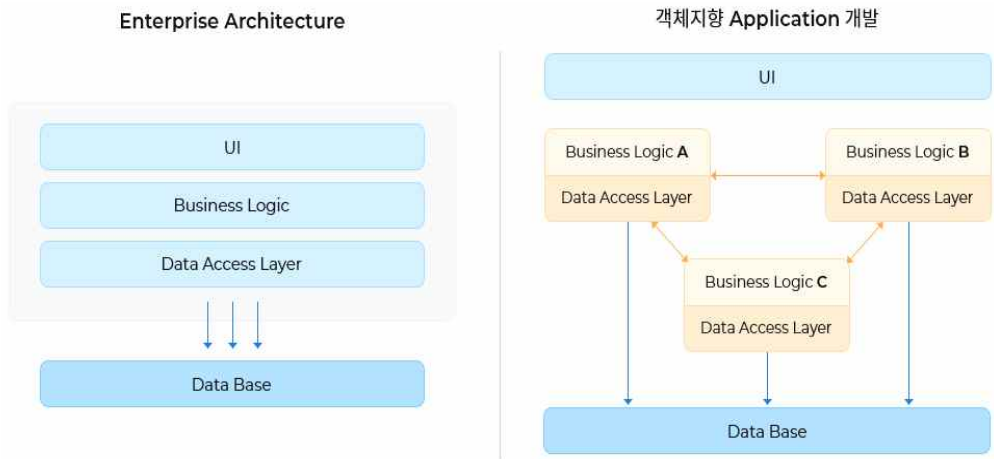


Fig. 1. Traditional development architecture.

일하다. 따라서 본 연구에서 추구하는 Cloud Native 환경의 활용 부분에서는 장점을 찾기가 힘들다. 하지만 전통적인 개발 방법 중에서는 생산성, 품질, 유연성에서 가장 뛰어난 효과를 보여준다. Fig. 1은 엔터프라이즈 개발방법, 객체지향 개발방법에 대한 구조를 나타낸다. 2.1.1은 전통적인 개발 방법 3가지의 생산성 및 품질에 대한 사전 연구의 결과를 보여준다[4]

2.1.1 SW Visualization 기반의 Tool-Chain 개발 방법

소프트웨어 공학센터에서 제안하는 소프트웨어 시각화 기법을 통해 중소규모의 SI 프로젝트의 생산성 향상정도를 알아보기 위해 분석, 설계, 구현, 테스트, 배포, 관리 전 과정을 대상으로 생산성 향상정도를 측정하였다. Table 1 Experimental Conditions 은 2013년 1월부터 2014년 8월까지 진행되었던 프로젝트를 중심으로 A project는 고객사 자체 품질관리

Table 1. Experimental Conditions

Section	Project A	Project B	Project C
Method of Quality Control	Self Quality Control by Customer	CSE's Tool-Chain-based Quality Control	Tool-Chain - based Quality Control for Small and Medium Scale Projects
Quality Index	Customer's Quality Index	Standard Quality Index	Standard Quality Index
Development Methodology	Agile & CBD	CBD	CBD
Language and DB	Java, Gauss, Oracle	Java, Flex, Oracle	Java, Flex, Oracle
Period	2013,6~2014,3 (10 Months)	2013,3~2013,10 (Seven Months)	2014,3~2014,9 (Seven Months)
Input Personnel	7M/M	6M/M	6M/M
Level of Program Difficulty	High	High	High
Program Complexity	High	High	High
Evaluation Methods	Customer Satisfaction, Deadline Observance Rate, Cost Reduction Rate, Developer Satisfaction	Customer Satisfaction, Deadline Observance Rate, Cost Reduction Rate, Developer Satisfaction, Appropriateness of Tool-Chain	Customer Satisfaction, Deadline Observance Rate, Cost Reduction Rate, Developer Satisfaction, Appropriateness of Tool-Chain

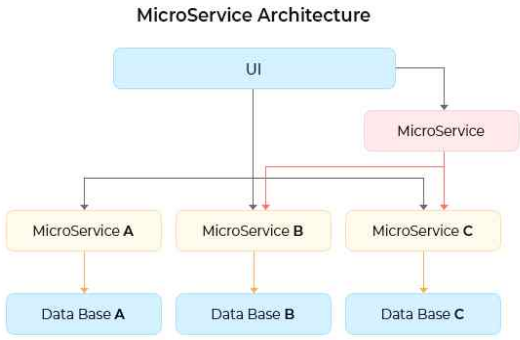


Fig. 2. Micro Service Architecture.

방법론을 기반으로 하며 B Project는 소프트웨어 공학센터에서 제안하는 품질관리 방법을, C Project는 SW Visual 기반의 중소규모 SI 사업에서의 생산성 향상방법 연구[4]를 바탕으로 실험을 진행한 결과 값이다. 이 과정은 Cloud DevOps 환경과 유사함을 보여주지만, 모든 프로세스 및 개발환경을 개발자가 직접 구현해야 하는 번거로움이 있었다. 그리고 지속적인 배포라는 Cloud 배포 개념에 사용되는 CI/CD 툴과도 같은 툴을 사용하였으나 관리자에 의해 수동으로 배포되었다.

2.2 Cloud Native 환경과 MSA (Micro Service Architecture)

Cloud Native 환경이란 클라우드 컴퓨팅 모델의 장점을 모두 활용하여 애플리케이션을 개발하고 실행하는 방식이다. DevOps, Continuous Delivery, Micro service, Containers와 같은 핵심요소들을 활용하여 유연하고 확장성이 확보되는 환경을 의미한다[5].

여기서 말하는 클라우드 컴퓨팅의 대표적인 장점은 지속적인 자동 배포 기능(provisioning)과 서버 자원을 공동으로 활용하는 컨테이너 기반의 가상화 기술 그리고 서비스에 대한 트래픽에 따라 부하를 자동으로 조정하는 Auto Scaling기능을 대표적으로 꼽는다.

이 기능들은 서비스 운영을 유연하게 하며 경량화된 서비스들을 언제나 배포할 수 있으며 24시간 무정지 서비스를 제공한다. 이러한 환경들을 PaaS, IaaS라 하며, 클라우드 PaaS, IaaS의 장점들을 활용하기 위해서는 SaaS 즉, 어플리케이션을 개발하는 방법들이 정의되어야 한다.

본 논문에서는 이 방법을 Cloud Native 환경에서의 어플리케이션 개발이라고 하며, 대표적인 방법으로 MSA(Micro Service Architecture) 기반의 개발방법을 제시한다. MSA 개발방법은 대규모 소프트웨어 개발에 적용하기 위한 것으로 단독으로 실행가능하고 독립적으로 배포될 수 있는 작은 단위(모듈)로 기능을 분해하여 서비스 하는 아키텍처라고 하며, 레고 블록을 조립하여 원하는 모양으로 만드는 것에 비유할 수 있다. MSA 사용으로 개발자들이 클라우드 망을 통해 공유하고 협업하여 자유롭게 소프트웨어를 개발할 수 있으며, 개발 및 유지보수에 소요되는 시간과 비용을 절감할 수 있다. 단일 서비스로 개발하는 기존 모놀리식 방식과는 반대되는 개념이며, 서비스 지향 아키텍처(SOA : Service-Oriented Architecture) 방식보다 더 세분화되어 있다[6].

Table 2는 Cloud Native 환경의 장점을 활용하기 위해 PaaS와 IaaS Cloud 시스템을 활용하여 전통적인 개발 방법으로 개발했을 때와 MSA 개발 방법으로 제안했을 때의 효과를 비교한 자료이다[6]

2.3 Cloud Native 환경에서의 성능 , 품질, 생산성 측정 방안

블로거 짐버드는 코드라인 수, 수익성, 개발속도, 사이클 시간, 코드품질 면에서 계량화하고 정량적으로 평가하는 것은 절대 불가능한 일이라고 주장한다[7]. 하지만 많은 기관 및 업체들은 소프트웨어 품질 및 성능 그리고 생산성 향상에 대해 많은 고민과 연구를 끊임없이 해왔다.

이런 환경에서 Cloud Native 환경에서의 어플리케이션들의 성능, 품질, 생산성 측정방식에 대한 연구도 진행되고 있다. 국내의 경우는 KPI[핵심성과지표]기반의 SW개발 평가 방안이 있는데 일반적인 일정, 요구사항 준수, 비용, 이슈 등에 대해 비즈니스 분석과 의사소통에 효과적인 정보를 제공해 주는 대시보드 구축 방안을 제시하였다[8]. 이외에도 각종 개발자 커뮤니티 및 기관들은 DevOps 이니셔티브의 성공을 측정하기 위해 12가지 KPI에 대한 측정 항목을 수집, 처리 하는 지표들을 제시하고 있다[9].

2.4 클라우드 네이티브 환경에서의 어플리케이션 개발 사례

MSA 기반의 솔루션 개발 사례는 배달의 민족, 쿠팡, 넥플릭스, 카카오 등 많은 기업들이 시도하였으

Table 2. Monolithic VS MSA on the Cloud Native Environment

	Monolithic Method	MSA Method
Development language	Single language configuration	Multiple language combinations available
Update	Lack of Flexibility due to a program-wide update	More flexibility due to an update by service
Elasticity	Scale Out by forking the whole program	Scale out the only service that requires expansion
Resources	Resource management from a single location	Location management by service
Downtime	Downtime on the entire program when an update or problem occurs	Downtime on the only service when an update or problem occurs
CI/CD	Increased deployment time to build/test for the entire program	Decreased deployment time to build and test for each service
Modular design	Due to high dependencies, high cohesion makes it difficult to expand	Due to low dependencies, each service secures independence and is easy to expand
container-based virtualization	Heavy container operation due to the demand on a high specification	configuration with a low specification container per service enables agile container operation
Maintenance	An impact analysis on the entire program is required for maintenance	Modifications/changes by service reduce maintenance burden
Collaboration	Typical silos with a single team Increases team management costs and limites development methodology	Multiple smaller teams reduce team management costs and are able to apply various development methodologies
Quality	Difficulties in maintaining high quality due to high impact of frequent changes	Quality assurance for modifications/ changes by service

며 실제로 운영되고 있다. 하지만 솔루션의 경우 요구사항이 명확하고 작업 기한도 내부적으로 협의가 되는데 비해 중소규모의 SI프로젝트에서는 한정된 기간 내에 다양한 요구사항을 충족시키며, 수시로 바뀌는 요구사항의 risk를 관리해야 하므로 국내 적용 사례는 찾기 힘들다. 그래서 우리는 중소규모의 SI에서의 Cloud Native 환경을 구축하고 그 환경에서의 개발을 경험하기 위해 2016년부터 2019년까지 MS-Azure의 퍼블릭 서비스, IBM의 퍼블릭 클라우드 Blue-Mix, IBM의 Private Cloud인 ICP 활용 및 테스트, Tmax의 Public Cloud인 Prozone 및 DevOps PO환경 하에서 개발을 진행하였다.

국내 클라우드로는 정부의 G-Cloud, 이노그리드, 크로센트 등이 존재하지만 PaaS, IaaS 제품들만 가지고 있으며, Tmax가 소프트웨어 방식의 클라우드 ProZone을 통해 EMB기반의 MSA 어플리케이션 개발을 지원하고 있다. 여러 가지 클라우드 환경에서의 개발을 진행하면서 솔루션 형태의 자사 제품을 개발할 때에는 대부분 성능 및 품질에 문제가 없는 것으로 평가되고 있지만, SI사업에서 여러 제품들과의 통합 및 협업을 위해서는 많은 문제점들이 대두되었다. 아직까지 국내에서는 클라우드 환경에 대한 표준이 없기 때문에 국내 기업들은 각 글로벌 벤더에서 운영하고 있는 제품의 기능을 활용하고 있으며, 개발사는 글로벌 클라우드 벤더에서 제공하는 제품들의 기능에 맞추는 실정이다. 그래서 개발사들은 클라우드 벤더들이 제공하는 각자의 시스템에 맞추어 개발하는 소모적인 환경 세팅을 활용하고 있으며, 이런 문제점들을 해결하기 위해 히로쿠(Heroku)는 모든 Cloud Native 환경에서 배포할 수 있는 분산형 마이크로서비스 구축에 필요한 12가지 요소(Twelve-Factor app)를 제안하였고,[10] 현재 이 12가지 요소는 웹이나 앱 혹은 SaaS(Software as a Service) 개발자들에게 활용되면서 활발한 커뮤니티를 형성하고 있다[11,12].

Table 3. SW Development Evaluation Method with KPI

Evaluation Method of SW Development with KPI	
1. Schedule Adherence	• Timed compliance indicators measure performance from a time-management perspective and plan for actions required in terms of the timing of inputs, accurate estimates, the grasp of the work breakdown due to additional requirements, and avoidance of schedule delays • The measurement formula is the ratio deviation between actual and plan distribution date - [1- (ADD-PDD)/PDD] * 100 - ADD(Actual Delivery Date) = Actual completion date - Planned start date - PDD(Planned Delivery Date) = Planned Completion Date - Planned start date
2. Content Adherence	• The content compliance indicator provides useful information about the team's productivity or performance in handling demand ranges • Evaluate productivity in terms of the timing of inputs and the deployment of the entire plan range • The measurement formula is the ratio deviation between the total completed requirements and the total injected requirements - (ΣCompleted requirements) / (Σinput requirement) * 100
3. Cost Adherence	• As indicated by the index name, this metric provides analytical information to measure cost management efficiency • Focuses on when to make an injection, accurate estimation, and over-cost dues • The measurement formula calculates the deviation between the actual cost and the input cost - [1 - (Expected Costs Upon Completion-Input cost) / Input cost] * 100 - Expected Costs Upon Completion = Actual cost + Estimated cost - Input cost = Base value
4. Deliverables and Team Performance	• These indicators provide useful information for product quality management • It is able to be used as a scale not only for design and development but also for other project phases • The product quality of the development phase can be measured in terms of the accuracy of the functional definition • It also helps to measure the quality of the output during the test phase to help fulfill the bug-free product contract • The measurement of this indicator provides information on the ability to identify issues before development and the prevention of bugs introduced during development • Measuring features and design quality: - [1 - (ΣFunction and Design Issues) / (ΣAll kinds of issues)] * 100 • When measuring bug-free products contracts - [1 - (ΣBug fix)/(ΣDevelopment function)] * 100
5. Cost of Quality	• Quality should be measured during development as well as during sales and customer satisfaction management • If the final distribution is in service, the quality of the product must be measured and the costs associated with the stabilization must be identified • The measurement formula for this indicator is to measure the maintenance costs associated with the efficiency of the team, which manages the accident caused by low quality products and corrects issues that escalate. - (ΣMaintenance cost) / (ΣNumber of incidents including solution modification)

Table 4. Twelve-Factor app

Twelve-Factor app	
I Codebase	One version-managed codebase and multiple deployments
II Dependency	explicitly declared and isolated dependencies
III Setting	Settings stored in the environment
IV Back-end service	Regard back-end services as linked resources
V Build, Release, Execute	Fully separated build and execution phases
VI Process	Run one or several stateless processes for an application
VII Port Binding	Use port binding to disclose services
VIII Concurrency	Extensions with process model
IX Disposability	Maximize reliability with quick start and Graceful shutdown
X Match development / production environments	Keep development, staging, and production environments as similar as possible
XI Log	Regard logs as event stream
XII Admin process	Run the admin/maintenance as a one-time process

3. Cloud Native 환경에서의 어플리케이션 개발 방법 연구

3.1 연구의 목적 및 방법

2장에서는 전통적인 개발 방법과 MSA 개발 방법 및 Cloud Native 환경에서 어플리케이션 운영 및 개발방법, MS, IBM, Tmax의 해외 선진사례를 살펴보았다. 아직 글로벌 기업들도 자사에만 맞는 환경과 어플리케이션 방법들을 제공하기 때문에 표준화가 되지 않아 범용성과 호환성에 다양한 문제점들이 있음을 알 수 있다.

MS의 DevOps 환경에서 IBM의 Blue Mix Cloud

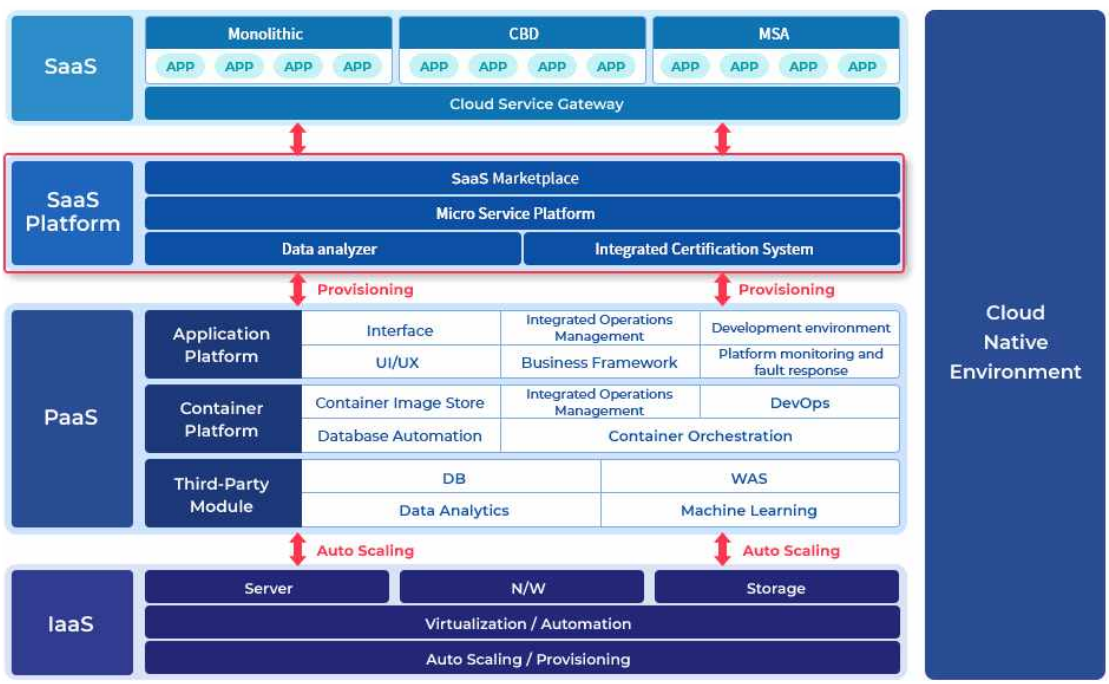


Fig. 3. Diagram of S/W Development Method in the Cloud Native Environment.

사용을 시도해 보았지만 많은 제약이 있었으며, 국내 기업 Tmax의 Prozone에서 제공하는 Was(Jeus) 및 DB(Tibero) 등은 MS와 IBM에 호환성이 없다. 그 외에도 Cloud Native 환경에서의 어플리케이션을 개발하기 위해 필요한 MSA 접근 방법이 제각각이며, 각자의 DevOps 환경을 활용해야만 최적의 성과를 낼 수 있기 때문에 비용 및 효율적인 부분에서 만족스럽지 못하다. 다만, 대규모 프로젝트를 진행할 때 적정한 보수를 지급하면서 기술적인 지원 및 환경을 지원 받는다면 앞서 언급한 기업들의 Cloud 환경은 매우 효율적일 것이다.

본 연구에서는 어떤 Cloud Native 환경에서도 배포가능한 DevOps 및 MSA를 최적화할 수 있는 프로세스 그리고 방법론에 대해서 기술하며, Monolithic 개발 방법과 MSA 방식을 Cloud Native 환경에서 운영했을 때 생산성 및 품질을 비교한다. 정량적인 성과 측정을 위해 아래와 같은 조건을 설정하였다.

첫째, 본 연구는 2.3 에서 설명하였던 KPI를 바탕으로 한 SW 개발 평가 방안으로 비교한다.

둘째, 동일한 Cloud Native 환경에서 Monolithic 방법과 MSA 개발 방법으로 어플리케이션을 구현한다.

셋째, 개발기간이 6개월 이상 12개월 이하의 SI사업 즉 대학교의 종합정보 시스템 및 역량 시스템 프로젝트를 기준으로 하여 인원은 5명 이상 10명 미만으로 구성된 프로젝트를 대상으로 한다.

넷째, 개발 단계 및 방법론은 보편적인 Monolithic, MSA 방법에 따라 적용 한다.

3.2 Cloud Native 환경에서의 어플리케이션 개발 방안 제안

위의 Fig. 3은 보편적인 Cloud 구성을 나타내고 있다. 본 연구를 위해 사용자의 요구에 맞게 시스템 자원을 할당, 배치, 배포해 두었다가 필요시 시스템을 즉시 사용할 수 있는 상태로 미리 준비해두는 Provisioning기능과, 트래픽에 따라 자원의 사용량을 자동적으로 조절하는 Auto Scaling 기능, 그리고 위의 두 기능을 활용하기 위한 컨테이너 기반 가상화에 맞는 PaaS, IaaS 환경의 Cloud Native 환경을 구축하였다.

Monolithic 개발 방법과 MSA 개발 방법을 비교하기 위해 2013년 1월부터 2014년 8월까지 중소규모의 SI 프로젝트에서의 생산성 향상 연구를 하였고 2016년부터 2019년 6월까지 MSA방법을 구현하기 위한 연구를 수행하였다. 연구 결과 Cloud Native 환경의 장점을 활용하기 위해서는 전통적인 개발 방법보다 MSA개발 방법이 효율적인 방법임을 알 수 있었으며, 그 결과를 토대로 중소규모의 SI프로젝트는 9가지 Cloud Native 어플리케이션 구현 과정을 아래와 같이 제시하고 세부적인 구현방법을 제안하였다.

1단계: 개발 문화 및 개발자의 인식 변화
2단계: Cloud Native APP개발을 위한 원칙 설정
3단계: 기존 Monolithic Service를 분석하여 재구조화

Table 5. MSA Requirements

MSA Requirements	
Requirement	Solution
Seperated Frontend/Backend	Synchronize with company's Engineer Tech Tree
Large volume of code, duplicate code generated	to make it easier for anyone to figure out by reducing code
Project for the whole team	Product by Working Group
Increased inter-system connectivity	Enforce API-based Contract Management
Desire to introduce new technology	Empowering autonomy for implementation on a Micro-Service basis (Polyglot)
Improved re-useability and continued development	Micro-Service unit reuse, free refactoring
Support by company's infrastructure	On-Premise Cloud, automate deployments linked to CI, and align with Container technologies such as Docker

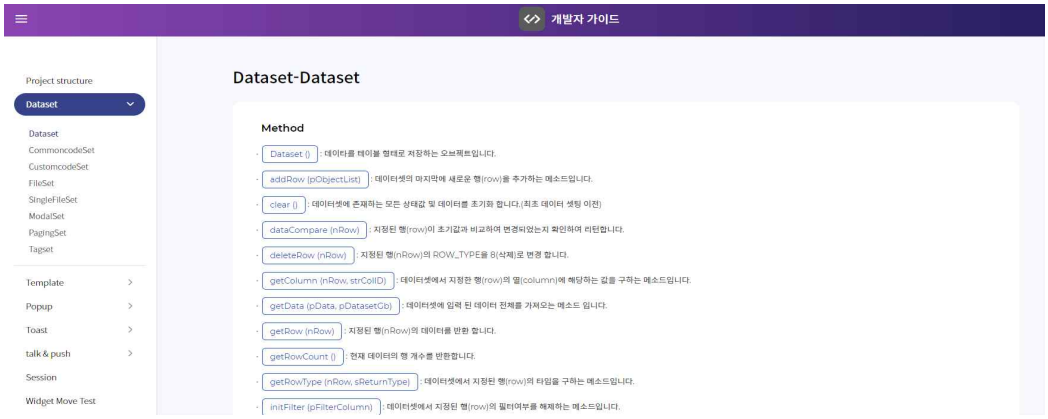


Fig. 4. Developer's Guide based on MSA.

4단계: MSA를 위한 Service Story 정립
5단계: 생산성 향상을 위한 DevOps 환경구현
6단계: Service 간 연계를 위한 API Gateway 활용 및 API 구현
7단계: SaaS Platform을 통한 체계적인 관리 체계 및 IT 자동화
8단계: 사용자 중심의 포틀릿(개인화) 서비스 구현 - 한층 발전된 모듈식 아키텍처 구현
9단계: API, APP Life Cycle 관리 및 통합 관제 실행

가) 1단계: 개발 문화 및 개발자의 인식 변화

전통적인 개발방법으로 오랫동안 SI프로젝트를 진행한 개발자들은 생산성 향상을 위해 새로운 기법을 도입하는데 부정적이다. 실제로 2013년 중소규모 프로젝트에서의 시각화 기반품질관리를 진행할 때도 개발자들의 협조를 얻기가 힘들었다. 그 이유는 SI 프로젝트 특성상 정해진 납기와 명확하지 않은 요구사항, 그리고 잦은 요구사항의 변경 등으로 품질관리 및 생산성을 담보할 수 없으므로 프로젝트 관리자 입장에서는 장기적 성능보다 눈앞의 생산성을 선택하기 때문이다. 이러한 상황은 유지보수나 안정화 기간에 오히려 더 많은 재원을 투입하게 되기 때문에 프로젝트의 생산성이 오히려 나빠지기도 한다.

반면, SI 프로젝트 환경에 Cloud Native APP 개발을 위한 MSA개발 방법을 적용할 경우, 많은 개발자들의 반발과 생산성을 주요하게 생각하는 프로젝트 관리자의 반대에 직면할 수 있다. 따라서 개발자 및 프로젝트 관리자 그리고 경영층까지 전사적으로 지원하지 않으면 개인 또는 팀으로 성과를 내기가 힘든 실정이다.

이러한 문제점을 해결하기 위해 본 프로젝트 팀은 매주 금요일 Tech-day를 통한 자기 주도적 학습과 각 직급별 Skill- Set을 통한 진급 체계 그리고 모든 SI프로젝트 회고를 통해 문제점을 리뷰하고 향후 다른 프로젝트에서의 개선점을 반영하는 프로세스를 구축하였다. 우리는 2년에 걸쳐 [Table 5]와 같은 MSA Requirements을 도출할 수 있었다.

나) 2단계: Cloud Native APP개발을 위한 원칙 설정

중소규모의 SI프로젝트에서 Cloud Native APP개발을 위해서는 개발 문화 및 개발자의 인식변화가 있어야 하며 전사적 개발 원칙이 설정 되어야 한다. 다음은 우리가 작성한 MSA 기반의 개발 원칙이다.

- 전통적인 개발방법(Monolithic)으로 개발된 Service를 나누어서 단계적으로 진행한다.
- 새로운 프로젝트는 전통적인 개발방법으로 진행하고 안정화 되면 Service를 분해한다.
- 독립적인 팀을 구성한다.
- 팀원들을 교육하고 개발 가이드를 제작, 제공한다.

다) 3단계: 기존 Monolithic Service를 분석하여 재구조화

MSA 아키텍처를 설계하기 위해서는 서비스와 DB가 하나로 개발된 Monolithic Service를 잘게 분할해야 Cloud Native 환경의 장점인 프로비져닝 서

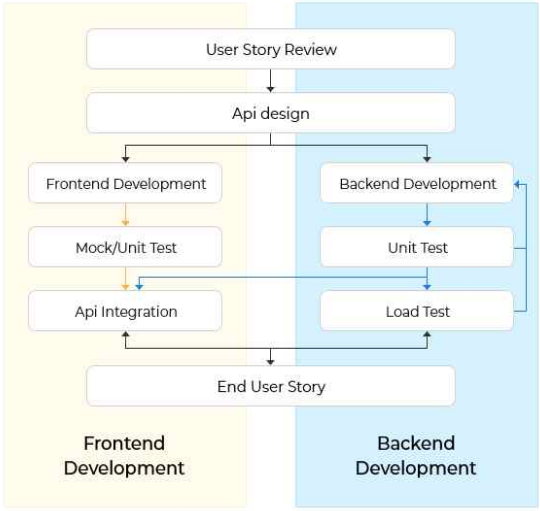


Fig. 5. Service Story Process for MSA.

비스를 활용 할 수 있다. 그러기 위해서는 Monolithic Service를 아래와 같이 재구조화 하여야 한다.

- Project를 Product화하여 파일럿 프로젝트로 진행한다.
- API를 정의한다.
- API를 Rest API Maturity Level 2이상이 되도록 강제화한다.
- API를 문서로 정의한다.
- 다양한 DB를 활용하기 위한 ORM기술을 활용한다.
- DB의존도를 낮춘다.
- 도메인 내부의 의미 있는 값에 대해 외부 노출을 지양한다.
- Micro-Service가 별다른 설정 없이 바로 기동되도록 한다.
- 서비스 간 연계 시 협의 체계를 구성한다.
- 주체 서비스에 대해 명확히 선정한다.

라) 4단계: MSA를 위한 Service Story 정립

3단계의 Monolithic Service를 재구조화 하였다면 이제는 재구조화 된 Service를 연결하고 배치하기 위하서 Service Story가 필요하게 된다. 기존의 Monolithic Service는 Front-end 와 Back-end가 통합된 Story를 가지고 있는 데 여기서는 Front-end 와 Back-end를 분리하여 단위테스트 및 Load Test를 진행하여 서비스를 연계하고 실행하기 위한 필수적인 단계이다.

마) 5단계: 생산성 향상을 위한 DevOps 환경구현

기존에 각 개인이 개발한 환경을 SVN 서버에 통합하여 수동 배포하던 방식을 Redhat의 오픈시프트 플랫폼 서비스를 적용하여 지속적인 배포를 위한 환경으로 자동화시켰다.

Fig. 6과 Fig. 7은 개발자의 개발환경을 자동으로 활용하기 위한 Devops 활용 Process를 보여주고 있다, 여기서는 소셜 프로그래밍을 위한 분산버전 관리 시스템을 활용하기 위해 Git 레파지토리를 활용하고 GitHub를 통해 소셜 프로그래밍을 가능한 개발 환경을 구성하고 있다. 또한 지속적인 배포를 위해 젠킨스를 연동하여 Cloud Native 환경의 장점인 컨테이너 기반의 자동배포 기능을 구현하였다.

바) 6단계: Service 간 연계를 위한 API Gateway 활용 및 서비스 연계 방법구현

MSA 개발 방법을 구현하다 보면 서비스 간 연계를 위한 API 개발 및 오케스트레이션 방식이나 코레

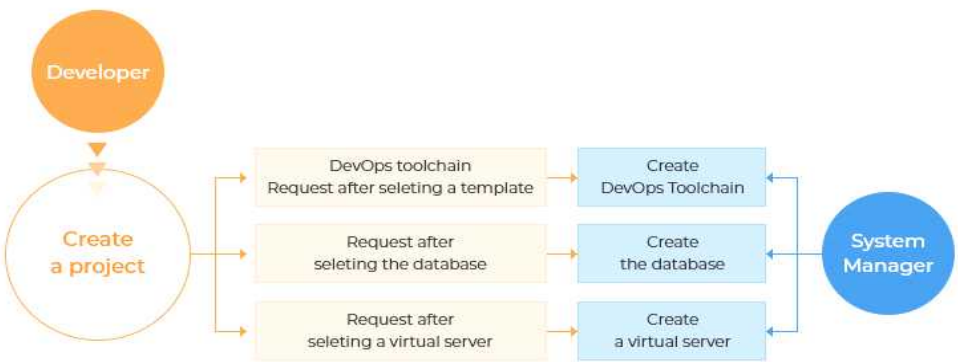


Fig. 6. DevOps Configuration.

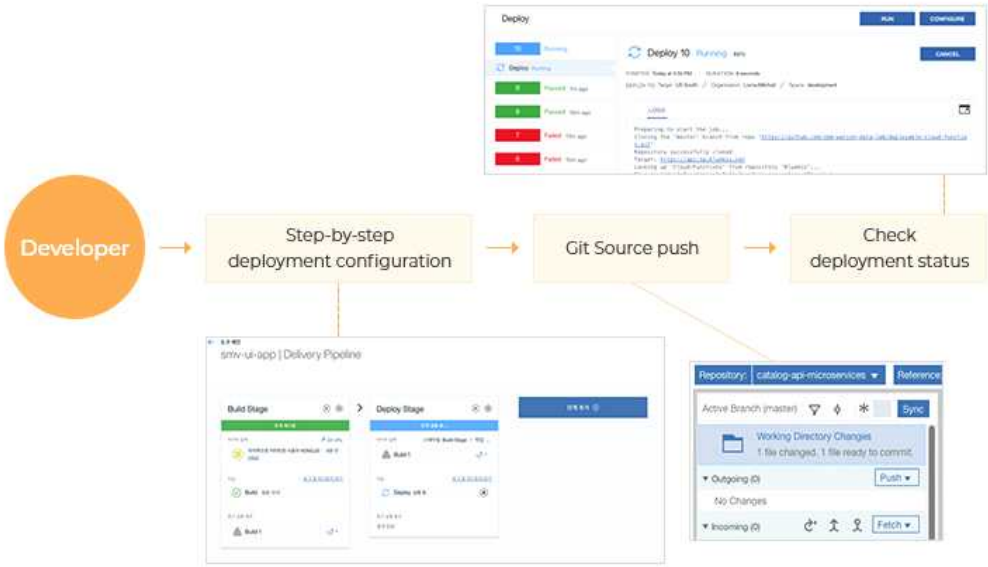


Fig. 7. Automated Deployment of DevOps Source.

오그래피 방식을 고민하게 된다. 또한 이렇게 생성된 API들을 체계적으로 관리하기 위해서는 API Gateway가 필수적으로 필요하다. 구글이나 넥플릭스 줄루같은 오픈소스들도 존재하지만 본 연구에서는 WSO2 오픈소스를 활용해 자체적으로 API Gateway를 제작하여 APP 및 사용자에게 배포하는 위젯 서비스까지 제공하는 life-cycle을 완성하였다 본 프로젝트에서는 서비스 오케스트레이션 방법을 채용하여 테스트 하였다.

사) 7단계: SaaS Platform을 통한 체계적인 관리 체계 및 IT 자동화

MS, 구글, 아마존, IBM등의 글로벌 퍼블릭 클라우드 사업자를 제외하면 MSA개발을 위한 SaaS Platform을 사용하는 개발사는 현재 알려진 바 없다. 본 연구에서는 오픈소스를 기반으로 하여 자체적인 SaaS Platform을 활용하여 체계적인 관리 및 IT 자동화를 구현하였다.

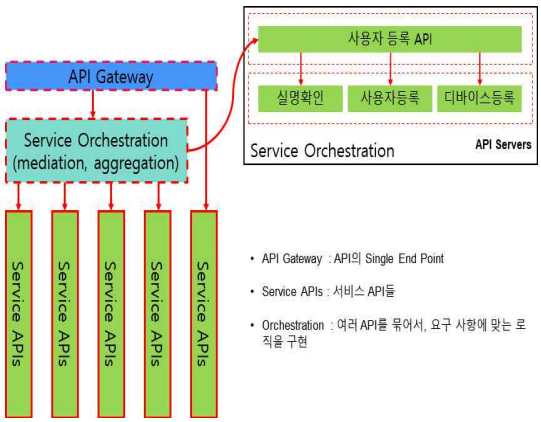


Fig. 8. Service Orchestration.

아) 8단계: 사용자 중심의 포틀릿(개인화) 서비스 구현

클라우드 사업자들은 각자의 포탈을 가지고 있지만 타사의 클라우드 환경을 지원하지 않는다. 우리는 서비스 관점에서의 개인화 서비스를 지원하기 위해 자체적인 포탈이 필요하다고 판단하여 Fig. 9의 SaaS Platform 체계도와 같은 관리자 - 개발자 - 사용자들의 환경을 쉽게 개인화할 수 있는 포탈을 개발하였다.

자) 9단계: API, APP Life Cycle 관리 및 통합 관제 실행

중소규모의 SI 프로젝트에서 통합 테스트 및 실운영을 지원할 경우 많은 문제들이 발생한다. 이런 문제들의 경우는 DB Server가 Pool이 되던, 네트워크가 끊기던 사용자는 서비스가 작동하지 않는다고 생각하여 일단 개발사에 문의한다. 또한 MSA 방식의 개발은 많은 모듈들이 산재하여 문제를 찾기가 쉽지

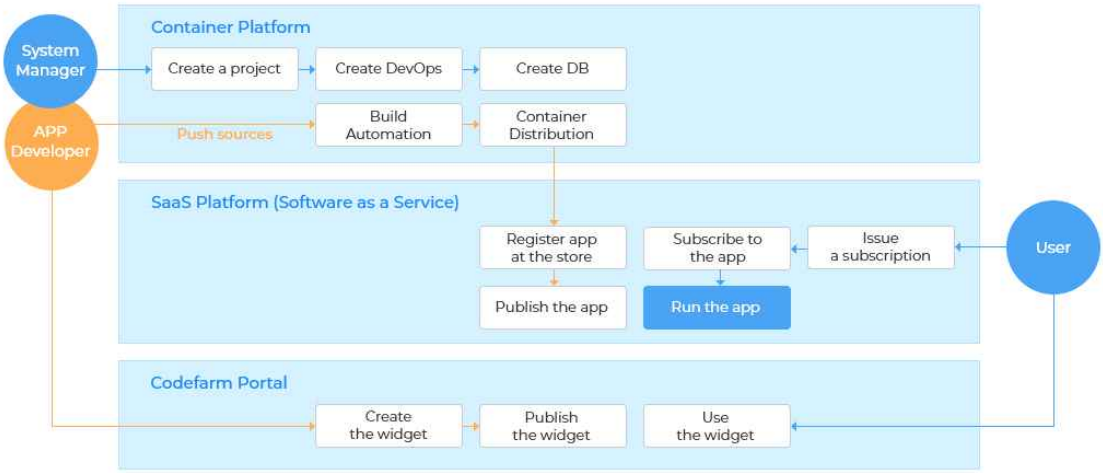


Fig. 9. Scheme of SaaS Platform.

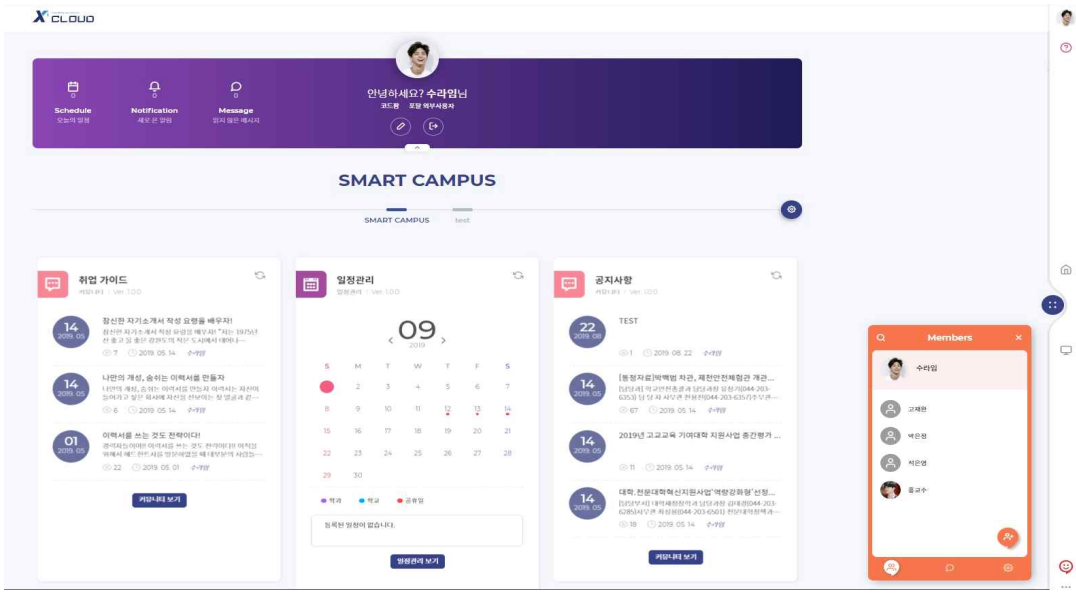


Fig. 10. User-centric Cloud Portlet Service.

않다. 따라서 Cloud Native APP개발은 통합 관제가 매우 중요하다. 개발된 API, APP등의 Life Cycle 관리와 연계된 PaaS, IaaS 와 API를 통한 모니터링 기반의 통합 관제가 실행되어야 한다. Fig. 11은 통합관제에서의 API 실패한 호출에 대한 통계를 보여주는 화면이다.

3.3 실험 및 평가

2016년 6월부터 2.1.1장의 Table 1 중 Project B와 Project C를 Cloud Native 환경으로 전환하기 위하여 자체 Project를 Product화하는 프로젝트를 진행하였으며, 2018년 3월 Project C가 Cloud Native 어플리케이션으로 전환하는 작업을 진행하였다. 그리고 2018년 10월 Project C와 유사한 Project C-1 사업을 2019년 6월까지 진행하여 KPI 기반의 SW개발 평가 방법으로 평가를 실시하였다.

이런 실험값을 토대로 과거 monolithic 사례를 D Project 라 하고 2.1.1장의 Table 1 중 Project B와

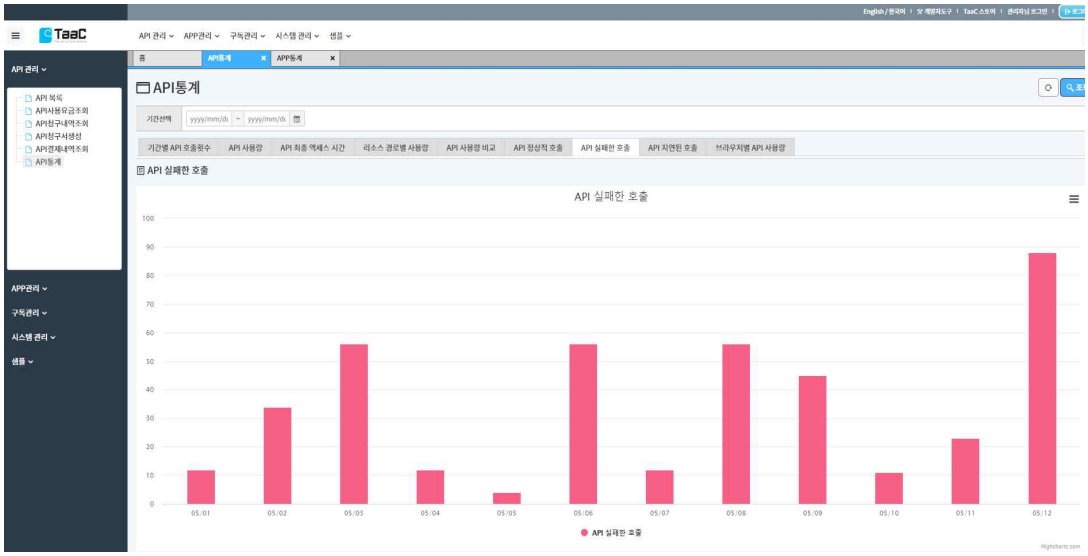


Fig. 11. Graphs of API Failed Call in Total Control Screen.

Project C를 E Project, 가장 최근에 개발이 끝난 프로젝트를 F 프로젝트로 하여 Table 6과 같은 결과를 얻을 수 있었다.

가) D Project

2016년 6월부터 2017년 3월까지 D 대학의 학생 역량 개발 프로젝트를 진행하였다. 이 프로젝트는 기존의 전통적인 개발방법으로 고객의 요구사항을 받아 Service와 DB를 하나로 설계하여 개발하였다. 개발기간 내에 고객이 요구한 내용과 예산을 잘 맞추어 KPI에서의 일정 준수 달성율은 100%에 도달하였으나, 사용하면서 요구사항 변경으로 인한 프로세스의 잦은 변경으로 유지 보수가 빈번하게 일어나고 생산성이 떨어져, D Project의 소스를 30% 이상 활용하기 힘든 상황이 되었다. 그 결과는 산출물 및 팀생산성과 품질 비용부분의 수치로 나타났다. Cloud Native 환경에서 장점은 극대화하지 못하였으며 가상화 환경에 서비스를 올려놓은 구조였다.

나) E Project

2.1.1장 Table 1의 B project를 고도화하는 사업이었다. 원래 이 프로젝트는 Monolithic 개발방법으로 개발되었는데 많은 정책 변경으로 고도화보다 전체를 개편하면서 객체지향 개발방법으로 개발하였다.

Table 6. Cloud Native Application Experimental Conditions

Section	D Project	E Project	F Project
Dev Method	Monolithic	CBD & MSA	MSA
Dev Tool	java	java	java
Delivery	2016.6~2017.3	2018.3~2018.9	2018.10~2019.3
Man/ Month	7M/M	6M/M	6M/M
Program Complexity	上	上	上
Schedule Adherence	86%	77%	70%
Content Adherence	100%	100%	100%
Cost Adherence	93%	87%	85%
Deliverables and Team Performance	47%	93%	95%
Cost of Quality	62%	87%	100%

이전 개발 소스에서 서비스와 DB가 하나로 개발되어 있어서 소스 재활용성이 매우 낮았으며, 업무 화면 정도만 재활용성을 가질 뿐, 내부 프레임워크와 프로시저들도 모두 다 재구성하였다. 향후 MSA로 전환하기 위하여 Service와 DB를 분리하여 설계하였다.

실제로 E Project 진행시 70% 정도 재활용이 가능하여 생산성에 도움이 되었으며, 사업 소요기간을 30일 정도 단축할 수 있었다. 다만 이 프로젝트도 D Project와 마찬가지로 Cloud Native 환경의 장점을 활용하는데 한계가 있었다.

다) F Project

D Project 종료 후, 유지보수의 어려움 및 잦은 프로세스의 변경으로 소스의 품질관리가 어려웠다. 특히 본사 프로젝트 이외의 타사 프로젝트와의 연계 부분에서 많은 문제점이 노출 되었다.

우리는 본 연구를 위해서 많은 표준을 정립하였으나, Cloud 환경을 이해하고 개발한 본사 인력과 Monolithic 환경에 익숙한 타사 개발 프로그램과의 연동부분이 문제가 되어 결국 프로젝트를 종료하였다. 하지만 이 프로젝트를 통해 신규 개발자들의 손쉬운 개발 환경 셋팅과 DevOps를 통한 동일한 품질관리, 그리고 API 와 APP 마켓을 통한 버전 및 라이프 사이클 관리를 손쉽게 할 수 있는 구조를 만들어 향후 유사 프로젝트 진행에 많은 향상을 기대할 수 있는 환경을 구축하였고 자동 배포 및 관제 체계를 형성하여 After Service가 아닌 Before Service를 구현하였다. 그 성과는 Table 6 F Project에서 산출물과 팀 생산성 그리고 품질비용 결과에 잘 나타나 있다.

4. 결 론

전통적인 개발방법과 객체지향 개발방법은 Cloud Native 환경의 특성을 활용하기 힘든 구조이다. Cloud Native 환경의 기술적 특징은 DevOps, 지속적인 배포와 MSA, 컨테이너라 할 수 있다. 앞서의 두 방법은 Service와 DB를 개별적인 컨테이너로 배포하지 않는 구조이기 때문에 Cloud Native란 용어와는 어울리지 않다.

본 연구를 통해서 Cloud Native환경에서의 어플리케이션 개발방법은 MSA 개발방법이 최적임을 알 수 있었고, MSA를 구현하기 위한 많은 조건과 기술적 요구사항들을 정의하였다.

SK, 용감한 형제들, 넥플릭스, Sales force.com등 많은 기술 선도기업들이 MSA 형태의 Service를 구현하고 있고 경험들을 오픈하고 있다. 본 연구에서 타 Cloud Native 환경에서의 어플리케이션 개발방법 및 활용은 확인하였으나, 전통적인 개발방법을 고수하는 솔루션 업체나 SI업체들과의 협업체계에 대한 방안에 대해서는 미제로 남아있다. 본 연구의 후속연구를 통해 이러한 한계를 보완하고 좀 더 나은 생산성 및 품질, 속도 등을 올릴 수 있는 방법을 모색할 것이다.

REFERENCE

[1] J. Han, S. Park, and J. Kim, “Conceptual Design and Verification of OverCloud Approach for Realizing Dynamic IoT-Cloud Services,” *Korean Institute of Information Scientists and Engineers Transactions on Computing Practices*, Vol. 25, No. 1, pp. 9, 2019.

[2] N. Alshuqayran, N. Ali, and R. Evans, “A Systematic Mapping Study in Microservice Architecture,” *Proceeding of IEEE 9th International Conference on Service-oriented Computing and Application*, pp. 44-51, 2016.

[3] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-native Architecture,” *Journal of IEEE Software*, Vol. 33, No. 3, pp. 42-52, 2016.

[4] J.B. Kim, J.Y. Jung, and J.I. Kim, “A Method for Tool-chain-driven Quality Control based on Visualization for Small and Medium Scale Software Development Projects,” *Journal of Korea Multimedia Society*, Vol. 18, No. 4, pp. 546-556, 2015.

[5] MicroService Architecture, https://terms.naver.com/ (accessed August 20, 2019).

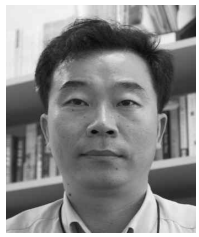
[6] Cloud Native Application, https://pivotal.io/kr/cloud-native (accessed August 12, 2019).

[7] 5 Reasons Why you Can’t Measure Developer Productivity, http://www.itworld.co.kr/news/91431#csidx0239b1e9956ae9da329c66324afd08

3 (accessed July 24, 2019).
[8] NIPA, *SW Development Evaluation Plan with KPI*, NIPA Software Engineering Center Trend Briefing v124, 2015.
[9] Fabio Jose Moraes DevOps KPI, https://dzone.com/articles/DevOps-kpi-in-practicechapter-1deployment-speed-fr (accessed August 20, 2019).
[10] The Twelve-factor App, https://12factor.net/ (accessed August 13, 2019).
[11] A. Mahajan, M.K. Gupta, and S. Sundar, *Cloud Native Spring*, Acorn, Seoul, 2019.
[12] J. Han and J. Kim, "Analytics-leveraged Box Visibility for Resource-aware Orchestration of SmartX Multi-site Cloud," *Proceeding of International Conference on Information Networking*, pp548－549 2016.

김 정 보

2015년 2월 동명대학교 컴퓨터미디어공학과 석사
2018년 2월 동명대학교 컴퓨터미디어공학과 박사 수료
2010년 10월～2014년 5월 해람정보기술(주) 대표이사
2012년 9월～2013년 12월 동원과학기술대학교 겸임교수
2016년 3월～현재 ㈜코드팜 대표이사
2018년 9월～현재 동명대학교 겸임교수
관심분야 : SI개발, 개발방법론, 품질관리, MSA, Cloud Native, Spring Cloud, SaaS



김 정 인

1991년 4월～1993년 3월, 게이오대학 계산기과학전공 공학석사
1993년 4월～1996년 3월, 게이오대학 계산기과학전공 공학박사
1996년 5월～1998년 2월, 포항공과대학교 정보통신연구소 연구원, 기계번역시스템 설계
1998년 3월～현재, 동명대학교 컴퓨터공학과 교수
관심분야 : 기계번역, 기계학습, 시멘틱웹, 웹2.0