Research Paper Informatization Policy Vol. 26, No.3 (2019), pp.090-106
https://doi.org/10.22693/NIAIP.2019.26.3.090

컨테이너 자동편성 플랫폼을 활용한 개방형 클라우드 플랫폼 생태계 전략*

정 기 봉**, 현 재 욱***, 윤 희 근****, 김 은 주*****

요 약 클라우드 서비스 시장은 온프레미스 환경에서 클라우드 컴퓨팅 환경으로의 전환에 힘입어 급속도로 성장하고 있고, 국내 클라우드 소프트웨어 시장 또한 전 세계적인 흐름에 따라 2022년까지 연평균 약 15%로 성장할 것으로 예상한다. 국내에서는 정부 주도하에 오픈소스 소프트웨어를 활용한 개방형 클라우드 플랫폼을 제공하고 있으며, 2019년까지 개방형 클라우드 플랫폼의 안정성 및 기능을 강화하고, 이종 클라우드 인프라 기반으로 운영되고 응용 소프트웨어의 전체 라이프사이클 관리 기능을 제공하는 세계적 수준의 개방형 클라우드 플랫폼 기반 및 개발자 지원환경을 제공하고자 한다. 이에 따라 본 연구에서는 개방형 클라우드 플랫폼 생태계를 활성화하기 위해 컨테이너 자동편성 플랫폼의 접목을 제시하고, 이를 통해 개방형 클라우드 플랫폼에서의 CaaS 활용 방안을 제시한다. 최종적으로 사용자에게 Application Runtime 및 Container Runtime을 제공함으로써 두 개의 플랫폼이 서로 상생할 수 있는 생태계의 방향성을 제시한다는 점에서 의의가 있다.

주제어 : 개방형 클라우드 플랫폼, 클라우드 서비스, PaaS, CaaS, Kubernetes

Open Cloud Platform Ecosystem Strategy Using the Container Orchestration Platform*

Jung, Ki-Bong**·Hyun, Jae-Uk***·Yoon, Hee-Geun****·Kim, Eun-Ju*****

Abstract The cloud services market is growing rapidly from the on-premises environment to the cloud computing environment and the domestic cloud software market in Korea is expected to grow at a CAGR of around 15%. In Korea, research teams are providing open cloud platforms using open source software under the government taking the initiative, which intends to enhance the reliability and functionality of open cloud platforms, provide users with a world-class open cloud platform-based and developer-friendly environment that is managed on heterogeneous cloud infrastructure and supported by full-lifecycle management of application software. In this paper, we propose a method to utilize CaaS in the open cloud platform, through incorporating the platform with the container orchestration platform. Finally, by providing users with the application runtime and container runtime, it presents how the two platforms can coexist and cooperate in the same ecosystem.

Keywords : open cloud platform, cloud service, PaaS, CaaS, Kubernetes

Received Jan 2, 2019; Revised May 13, 2019; Accepted May 17, 2019

* This work was supported by Institute for Information & communications Technology Promotion(IITP) grand funded by the Korea government(MSIT) (2017-0-00382, Advancement of Open PaaS Platform & Support for Development Environments)

** First author, Crossent Inc.(kbongjung@crossent.com)
*** Bluedigm, Inc.(juhyun@bluedigm.com)
**** Senior Manager, National Information Society Agency(hgyoon@nia.or.kr)
***** Director, National Information Society Agency(outframe@nia.or.kr)

Ⅰ. 서론

최근 세계적으로 4차 산업혁명 기술이 대두됨에 따라 클라우드 서비스 시장은 클라우드 컴퓨팅 환경으로의 전환에 힘입어 급속도로 성장하고 있다. 국내 전체 소프트웨어 시장 중 PaaS,[1] SaaS[2]를 포함하는 클라우드 소프트웨어 시장은 2017년 기준 약 4,300억 원 규모로 국내 소프트웨어 시장의 약 10%를 차지하는 거대 시장으로 성장하였다. 또한, 2022년까지 연평균 약 15%로 성장하여 약 8,700억 원 규모가 될 것으로 예상되고 있다(Doh, 2018).

클라우드 시장 또한 최근 IT시장의 흐름에 발맞추어 오픈소스 기반의 개방형 생태계가 확산되는 방향으로 분위기가 변화하고 있다. 클라우드 시장의 변화에 따라, 정부에서도 클라우드 컴퓨팅의 활성화를 위하여 2009년 클라우드 컴퓨팅 활성화 종합계획을 수립하고(Ahn, et al., 2015), 2015년 클라우드 컴퓨팅 발전법을 시행하였다. 또한, 개방형 클라우드 플랫폼(Open PaaS)을 2019년까지 고도화하여 세계적 수준의 개방형 클라우드 플랫폼 기반과 개발·테스트·배포에 필요한 다양한 개발자 지원환경을 제공하는 목표로 정부 주도의 플랫폼 연구·개발 사업을 진행하고 있다. 이 플랫폼은 안정성과 기능을 지속적으로 강화하고, 다양한 이종 클라우드 인프라 기반으로 운영되며 응용 소프트웨어의 전체 라이프사이클(개발-운영-유통) 관리기능을 제공하는 것이 포함된다.

기존의 시장 상황과는 별개로 클라우드 플랫폼의 경우 컨테이너 관리 플랫폼과 애플리케이션 플랫폼 기술이 혼재되어 사용되고 있으며, AWS Lambda를 대표로 하는 서버리스 컴퓨팅(Serverless Computing) 기술들이 출시되면서 사용자들의 클라우드 플랫폼 선택에 혼란을 주고 있다. 클라우드 플랫폼은 개발자의 관점에서 FaaS → PaaS → CaaS[3]의 순으로 이용 자유도가 높아진다. FaaS와 PaaS의 경우 개발자는 애플리케이션 빌드 결과물을 생성하여 배포만 할 수 있으며 애플리케이션 실행 환경에 대한 통제권은 전혀 가지지 못하는 데에 반해서, CaaS의 경우에는 애플리케이션 실행환경 즉 컨테이너 이미지를 개발자가 구축하여 배포한다는 측면에서 개발자의 자유 재량도가 높아진다. 반면 개발 생산성 측면에서는 이와 반대되는 상황이 발생하는데, 개발자가 책임져야 할 업무 범위가 위 순서대로 증가하게 되므로 개발 생산성은 좌측으로 갈수록 높아진다고 볼 수 있다. 특히 엔터프라이즈에서는 개발 업무 프로세스 표준화와 조직에 대한 통제 가능성이 중요하므로 개발자의 자유도보다도 규격화된 개발 업무를 수행할 수 있는 좌측의 클라우드 플랫폼들을 선호하는 상황이다.

이 기술들은 상호 배타적인 경쟁기술들이 아닌 상황과 환경에 맞게 사용하는 보완적인 관계이므로 하나의 플랫폼에서 사용자의 상황에 맞는 필요한 기술을 선택하여 사용하거나 동시 사용이 가능하도록 제공하는 것이 가장 이상적이나 실제에서는 다른 제품으로 제공되거나 같은 제품이더라도 다른 에디션으로 제공되고 통합제품에 대한 요구가 시장에 존재하고 있다.

본 연구는 사용자의 플랫폼 선택 자유도를 높이고 개방형 클라우드 플랫폼 생태계 활성화를 위해 기존 애플리케이션 플랫폼에 컨테이너 관리 플랫폼을 접목하는 방안을 제시하는 것을 목적으로 한다. 이를 통해 클라우드 플랫폼 사용자에게 애플리케이션 런타임(Application Runtime)과 컨테이너 런타임

1) Platform as a Service : 클라우드 인프라 서비스(IaaS, Infrastructure as a Service) 상에서 응용의 개발·실행을 지원하는 미들웨어 레벨의 클라우드 서비스로, 미들웨어·프레임워크 등 응용 개발에 필요한 기반 SW와 사용자 관리 및 보안·가용성·확장성 보장 등 응용의 실행에 필요한 기반 서비스를 제공

2) Software as a Service: 소프트웨어를 구입해서 PC에 설치하지 않아도 웹에서 소프트웨어를 빌려 쓸 수 있는 클라우드 환경에서 운영되는 애플리케이션 서비스를 의미하며 모든 서비스가 클라 우드에서 제공

3) Container as a Service: 사용자가 컨테이너 기반 가상화를 통해 컨테이너, Linux 응용 프로그램 및 클러스터를 관리하고 배포 할 수있는 클라우드 서비스 모델. 컨테이너 엔진, 오케스트레이션 및 기본 컴퓨팅 리소스는 클라우드 공급자로부터 서비스로 사용자에게 제공

(Container Runtime)을 함께 제공함으로써 사용자의 플랫폼 활용 편의성을 높이고, 또한 서로 다른 이종의 플랫폼이 서로 상생할 수 있는 생태계의 방향성을 제시하고, 향후 개방형 클라우드 플랫폼 생태계에 대한 전략적 대책 수립할 수 있도록 도움을 줄 것으로 기대된다. 본 연구의 배경기술들인 컨테이너 자동편성 기술과 대표적 솔루션인 Kubernetes, 개발형 클라우드 플랫폼에 대한 소개와 특성들을 설명하고, 클라우드 플랫폼 접목 방안을 도출하기 위해 애플리케이션 플랫폼에 컨테이너 관리 플랫폼을 접목하는 방안들을 제시하고 가장 적합한 방법을 선정하여 제시하고자 한다.

Ⅱ. 기술적 배경

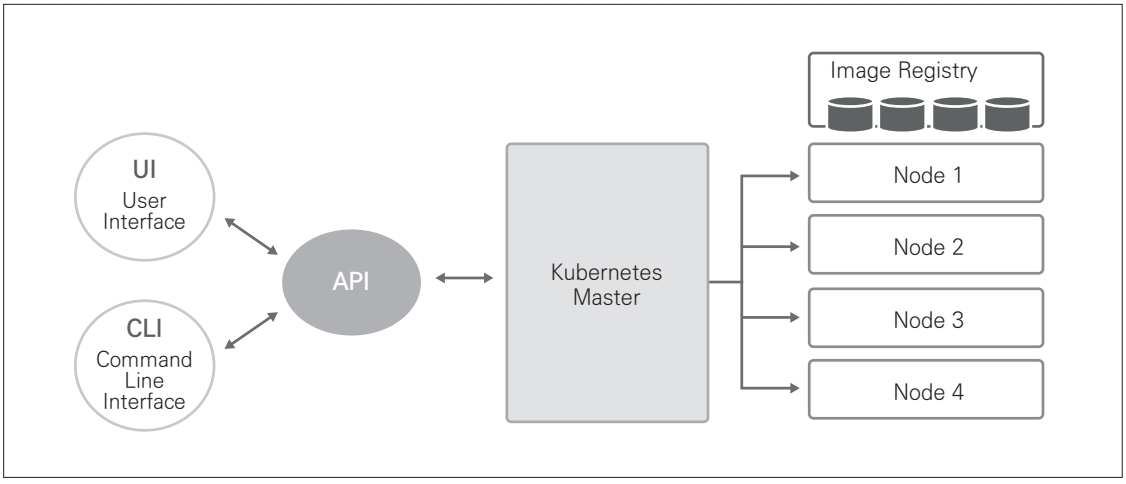
1. Kubernetes의 동작 구조

가상화 기술이 발전하면서 가상 컴퓨터 환경을 기반으로 서비스를 제공하는 컨테이너 기술이 보급되기 시작하였다. Kubernetes는 구글이 개발을 주도하고 대규모로 구성된 컴퓨팅환경에서 컨테이너를 효율적으로 배치하고 관리하기 위하여 핵심적인 기술요소로 등장하였다. Kubernetes를 이용하여 컨테이너를 배치하고 관리하기 위한 핵심적인 시스템이다(Kim, et al., 2018).

Kubernetes는 Master 노드와 Worker 노드로 구성되어 있다. Kubernetes의 아키텍처는 〈그림 1〉을 통해 알 수 있다. Master 노드는 전체 클러스터를 관리하며, Docker, rkt 등의 컨테이너 런타임 실행을 하며 컴퓨팅, 네트워킹 및 저장소(Storage) 등 애플리케이션이 필요로 하는 리소스를 제공한다. 또한, 로깅, 모니터링, 서비스 검색 및 선택적 Add-on을 위한 추가 구성 요소를 실행하며, 클라우드에서 실행되는 가상 머신(VM) 또는 데이터 센터에서 실행 중인 베어 메탈 서버에 설치된다. Worker 노드는 요청받은 리소스를 할당해 실제 컨테이너를 실행하고 운영한다.

Kubernetes는 크게 다음과 같은 5가지 기능을 제공한다. Automatic Bin Packing은 자원 요청 및 제약 조건에 따라 가용성을 유지하며 컨테이너를 Worker 노드에 자동으로 배치한다. Self-healing은 컨테이너 또는 노드 오류 등의 원인으로 상태 확인 실패 시, 컨테이너를 자동으로 복구한다.

Horizontal scaling은 Pod의 CPU 사용량 또는



source: Authors of the Kubernetes (2018)

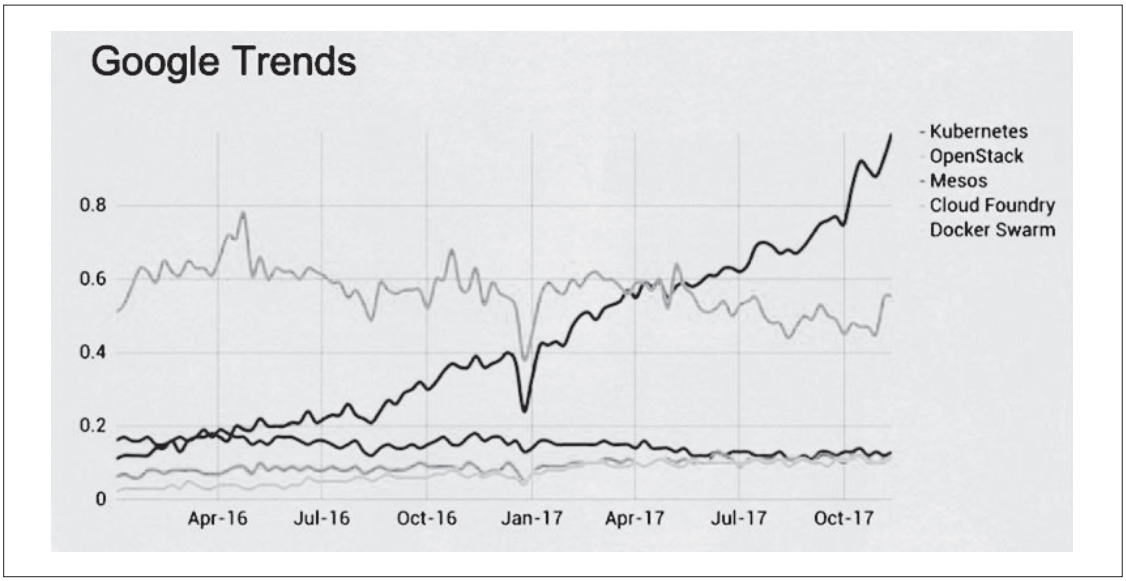
〈그림 1〉 Kubernetes 아키텍처
〈Fig. 1〉 Kubernetes Architecture

애플리케이션 측정지표에 따라 ReplicaSet 리소스를 이용한 컨테이너의 수평적 스케일링을 지원한다. Service discovery and load balancing은 별도의 애플리케이션 작업 없이 Service discovery 메커니즘이 동작하게 하고, 컨테이너에 고유 IP 및 단일 DNS를 제공하여 부하 분산(Load balancing)을 가능하게 한다. 마지막으로 애플리케이션 및 구성 변경 시, 전체 인스턴스의 중단 없이 점진적 롤아웃이 가능하여 Rolling update 문제 발생 시, 배포 리비전에 따라 변경 사항에 대한 자동 롤백을 지원한다.

Kubernetes는 다음과 같은 한계점을 지니고 있다. 먼저 Master 및 Node(Worker), 구성정보를 저장하고 있는 etcd에 대한 자동 장애 복구를 지원하지 않으며, 구동 중인 클러스터의 온라인 업그레이드가 불가능하다. 그리고 Node(Worker)가 5000개까지만 확장 가능하여 그 이상 사용하려면 여러 개의 클러스터가 필요하고, 상태정보를 저장하는 별도의 Stateful 서비스 관리 기능이 부족하다. 마지막으로 소프트웨어 자체의 빠른 버전 업으로 안정성이 떨어지며, 국내 개발자가 극소수인 Go 언어로 작성되어 있어 소스를 수정하기가 힘들다(The Kubernetes Authors, 2018).

2. 국내외 Kubernetes와 컨테이너 기술 동향

최근 클라우드 시장은 SaaS가 큰 비중을 차지하고 있으며, 사용자의 다채로운 요구사항을 즉각적으로 서비스화하기 위해 서비스형 컨테이너인 CaaS와 클라우드 컨테이너의 자동화 관리와 조절이 가능한 컨테이너 자동편성(Container Orchestration) 기술이 급성장하고 있다. 대표적인 컨테이너 자동편성 솔루션은 Docker Swarm, 아파치 재단의 Mesos, 그리고 리눅스 재단의 Kubernetes, 클라우드 파운드리 재단의 Diego 등이 있다. 현재 Kubernetes는 구글, 아마존, 마이크로소프트와 같은 대부분의 클라우드 서비스 제공자들이 서비스로 제공하고 있으며, Docker와 Cloud Foundry 또한 Kubernetes를 지원하기 시작하였고, 자체적인 자동편성 솔루션을 제공하던 AWS 또한 Kubernetes 서비스 제공을 발표했다(Cloud Foundry Foundation,



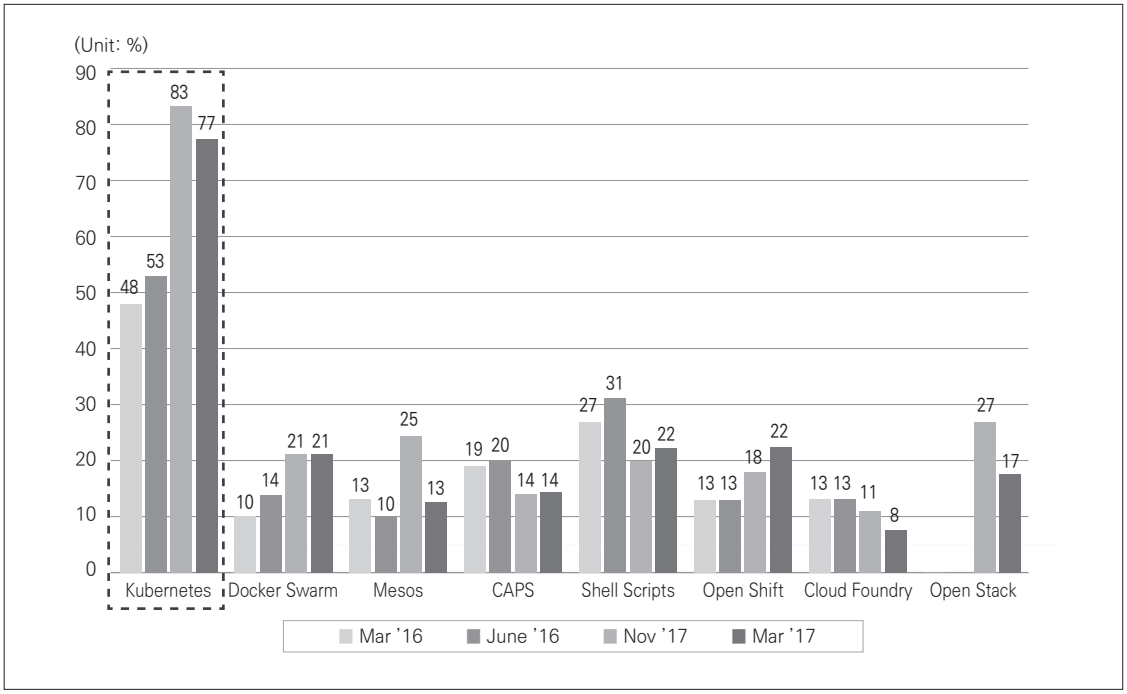
〈그림 2〉 Kubernetes 성장 추이
〈Fig. 2〉 Kubernetes Growth Trend

2018).

Kubernetes는 컨테이너 자동편성 플랫폼 및 도구로 컨테이너화된 애플리케이션(Containerized Application)의 배포, 확장 및 관리를 위한 오픈소스 플랫폼으로, 구글에서 설계 후 기부하여 현재 리눅스 재단에서 관리한다. “쿠버네티스”, “쿠베르네테스”, “K8s”, “쿠베”, “쿠버”, “큐브”라고 부르며 Apache License 2.0 라이선스로 공개한다(Linux Foundation, 2018). Kubernetes는 컨테이너에 대한 패키징, 배포, 서비스 검색, 확장 및 롤링 업그레이드를 위한 개방적이며 효율적인 모델을 제공하며 자동 배치, 자동 재시작, 자동 복제, 자동 확장으로 애플리케이션 상태 확인 및 셀프 복구를 수행한다. 또한, 컨테이너화된 애플리케이션과 해당 리소스의 즉시 확장 기능을 지원하고 스토리지 장착 및 추가로 상태정보 유지(Stateful) 애플리케이션 실행이 가능하다.

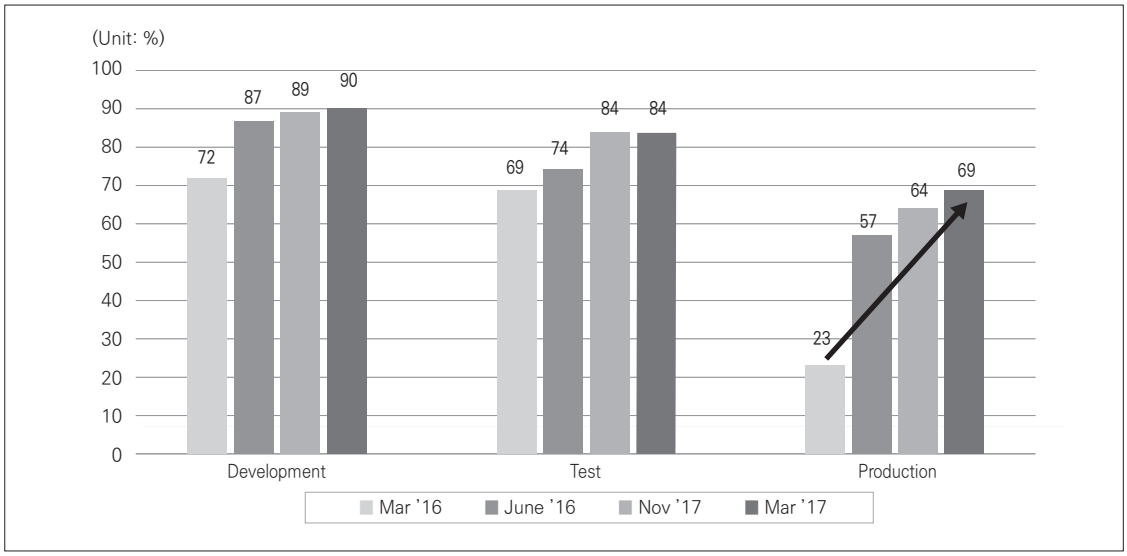
주요 IT 기업들의 지원으로 빠르게 발전하면서 Kubernetes는 대규모의 컨테이너 자동편성이 필요한 상황이라면 최우선으로 검토해야 할 솔루션으로 자리 잡았으며, 〈그림 2〉와 같이 실제로 구글에서의 검색 빈도가 꾸준히 상승하고 있다.

또한 〈그림 3〉에서 보는 바와 같이 조사 대상 기업의 컨테이너 관리 솔루션 선호도에서 Kubernetes가 차지하는 비중이 압도적으로 높음을 확인할 수 있으며, 〈그림 4〉를 보면 컨테이너 환경에서 Kubernetes를 사용하는 기업의 80% 이상이 개발 및 테스트 환경에서 꾸준히 사용하고 있으며, 2017년 3월 기준으로 69%의 기업이 Kubernetes와 컨테이너 기술을 운영 환경에서도 활용하고 있음을 확인할 수 있다(Conway, 2017). 이상의 추세로 인해 국내·외 기업에서 Kubernetes 기반으로 Container 서비스를 출시하고 있고 주요한 사례는 다음과 같다.



source: Conway (2017)

〈그림 3〉 Container Management Platforms Preferences
〈Fig. 3〉 Container Management Platforms Preferences



source: Conway (2017)

〈그림 4〉 The Shift from Development/Test to Production for Kubernetes
〈Fig. 4〉 The Shift from Development/Test to Production for Kubernetes

IBM(2018)은 2017년에 Kubernetes 기반 프라이빗 클라우드 플랫폼을 발표함으로써, IBM 클라우드 프라이빗(IBM Cloud Private)은 애플리케이션을 프라이빗 클라우드 환경에 배치할 때 전통적인 가상머신과 컨테이너 관리 플랫폼 Cloud Foundry에 더하여 Kubernetes에도 배포할 수 있게 함으로써 전체적으로 세 가지 방법을 지원하고 있다.

Microsoft(2018)는 2018년 자사의 콘퍼런스인 'Build 2018'에서 애저 쿠버네티스 서비스(Azure Kubernetes Service, AKS)를 공개하였다. 자사의 클라우드 상에서 Kubernetes 사용 경험이 없는 개발자가 컨테이너 기반 솔루션 개발을 수행할 수 있도록 개발자 도구 및 작업 영역, 데브옵스(DevOps) 지원기능, 네트워킹, 모니터링 등을 통합해 최적의 컨테이너 기반 개발 환경을 제공한다.

AWS의 Kubernetes 기반 컨테이너 서비스인 엘라스틱 컨테이너 서비스(Elastic Container Service for Kubernetes, EKS)는 2018년에 직접 Kubernetes 클러스터를 설치 및 운영할 필요 없이 AWS에서 Kubernetes를 쉽게 실행할 수 있도록 해주는 관리형 서비스를 공개했다(Amazon Web Services, Inc, 2018). 그 이외에 국내에서는 삼성 및 SK텔레콤에서 Kubernetes 기반으로 Container 서비스를 추진하고 있다.

3. 개방형 클라우드 플랫폼

1) 플랫폼의 구성요소

개방형 클라우드 플랫폼은 애플리케이션의 개발 및 운영 시 요구되는 인프라 구축 및 유지관리의 복잡성 없이도 웹 애플리케이션을 개발 및 운영 할 수 있도록 지원하는 클라우드 컴퓨팅 서비스다. 개방형 클라우드 플랫폼은 IaaS[4]를 기반으로 구축·운영되며, Java, Ruby, Python, PHP, Go 등 다양한 언어로 개발된

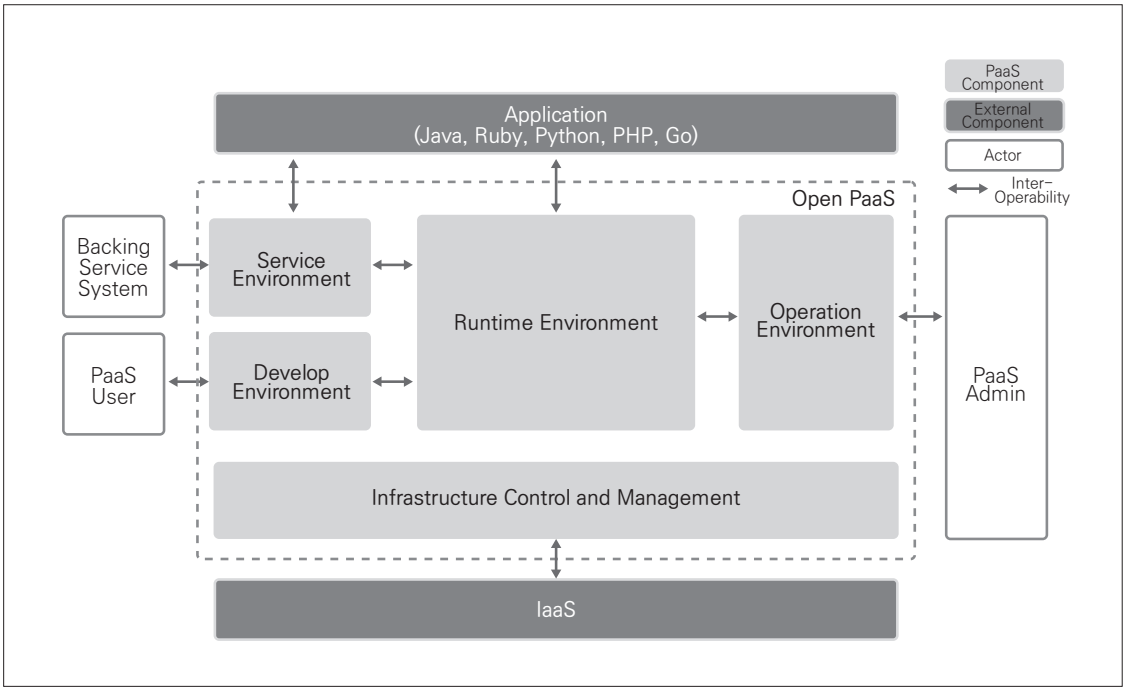
4) Infrastructure as a Service

응용 애플리케이션의 배포와 실행을 지원하는 서비스이다. 클라우드 플랫폼 사용자는 복잡한 인프라 서비스의 관리 대신 플랫폼이 제공하는 개발환경을 이용하여 응용 애플리케이션의 개발·배포·실행 요청만으로 최종 사용자에게 응용서비스를 제공할 수 있다. 또한, 응용서비스 실행 시 필요한 RDBMS, NoSQL, 메시징 서비스 등 다양한 대외의 백엔드 서비스의 손쉬운 연계를 지원한다.

개방형 클라우드 플랫폼은 〈그림 5〉와 같이 IaaS를 제어하는 인프라 제어 및 관리부와 개발자에게 애플리케이션 개발과 배포 서비스를 제공하는 개발/서비스 환경부, 운영자에게 플랫폼 관리 서비스를 제공하는 운영환경으로 구성된다. 개방형 클라우드 플랫폼은 개발자 포탈, 개발도구 및 관리자 포탈과 상호작용 하여 애플리케이션의 배포 및 관리, 모니터링을 수행하기 위한 오픈 API 서비스를 제공하며, 또한 플랫폼 자체적으로 API 게이트웨이 서비스를 제공하여 외부 애플리케이션을 접근 가능한 API 서비스로 제공하거나 플랫폼에 배포된 애플리케이션을 API 서비스로 대외에 제공할 수 있다. 애플리케이션 플랫폼은 다양한 유형의 IaaS를 지원하고 리소스 관리를 수행하는 하위 인프라 제어 및 관리 컴포넌트를 기반으로 운영 관리한다(Seo, et al., 2016).

2) 플랫폼 실행환경

개방형 클라우드 플랫폼은 애플리케이션 실행환경으로 컨테이너 Orchestrator인 Diego를 내부 구조에 포함하고 있다. 〈그림 6〉은 Diego 내부 구조이다. 그림에서 Cell이 실제 애플리케이션 실행환경이 탑재되는 가상머신으로, Garden Linux 컨테이너 형태로 존재하게 된다. Cell은 전체 플랫폼 구성에 따라 가변적이나, 보통 복수 개의 가상머신으로 구성이 되는데,



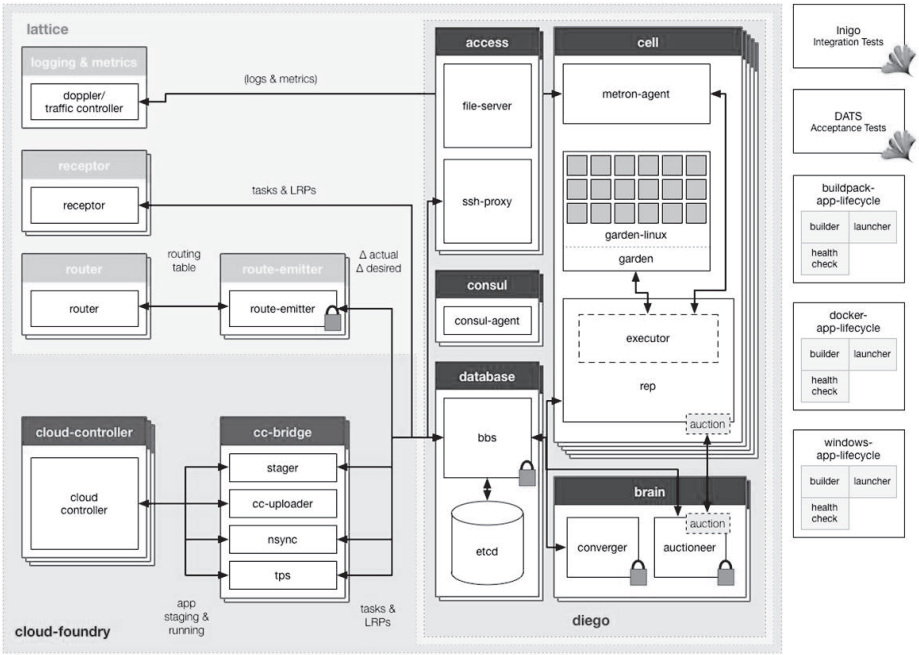
source: Seo, et al.(2016)

〈그림 5〉 개방형 클라우드 플랫폼 비즈니스 컨텍스트 모델
〈Fig. 5〉 Business Context Model for Open Cloud Platform

어떤 Cell에 실행환경을 생성할지를 결정하는 역할을 Brain에 존재하는 Auctioneer가 수행하게 된다. 이 과정에서 경매 알고리즘이 동작하게 되는데, 각각의 Cell이 현재 자신의 상태와 성능 정보 등을 Brain으로 전달하고 Auctioneer는 이 중 유휴 가용 자원이 가장 많이 남아 있는 Cell을 선정하여 신규 실행환경을 위치 시키도록 결정한다. 이러한 과정이 최고가를 제시한 사람이 낙찰을 받는 경매의 과정과 비슷하여 경매 알고리즘이라 명명하는 것이다.

애플리케이션 실행환경을 만드는 과정을 Staging이라 하는데 이 전체 과정은 Cloud Controller에 의해 관장되고 내부적으로는 빌드팩이라는 프로그램에 의해 수행된다. 빌드팩은 세 가지 단계로 동작되는데 Detect → Compile → Command의 순서이다. Detect는 개발자가 배포한 애플리케이션의 유형과 동작에 필요한 실행환경을 감지하는 단계로, 예를 들어 Java 언어의 경우 JRE, 웹 애플리케이션의 경우 WAS, Spring 등의 프레임워크의 경우 필요한 라이브러리 등을 자동으로 감지한다. Compile은 Detect 단계에서 감지된 필요한 실행환경들을 다운로드 또는 복사하여 적절한 곳에 위치시키는 과정이다. Command는 생성된 전체 패키지를 시작하기 위한 명령어를 생성하는 과정으로, 예를 들어 Java 웹 애플리케이션의 경우는 Tomcat 시작 명령어가 여기에 해당하는 것이다.

이런 과정으로 Cell 상에 동작하는 애플리케이션 실행환경의 정상 동작 여부와 자원사용 현황은 주기적으로 정보가 수집되어 체크되고 비정상 상황 발생 시 자동으로 복구가 진행된다. 이 역할은 Brain의 Converger가 수행한다. 이 과정에서 전체 시스템의 상태/성능 정보에 형상정보가 저장 관리되는 저장소



source: Cloud Foundry Foundation (2018)

〈그림 6〉 Diego 아키텍처
〈Fig. 6〉 Diego Architecture

가 필요한데, Working 데이터는 Consul에 메타데이터는 Database에 저장된다.

새로 생성되거나 동적으로 변화하는 애플리케이션 실행환경에 대한 URL 기반의 라우팅 정보는 라우터에 의해 유지 관리되며 실시간 변경 수행은 Router-Emitter에 의해 실행된다. 개방형 클라우드 플랫폼의 라우터는 HTTP와 TCP 프로토콜을 모두 지원한다.

Ⅲ. 클라우드 플랫폼 접목 방안 연구

1. 개방형 클라우드 플랫폼과 Kubernetes

개방형 클라우드 플랫폼은 보통 Application PaaS라고 불리는 Cloud Foundry 계열의 PaaS로 출발했으며 내부적으로 컨테이너 자동편성을 포함한다. (Diego) 반면 Kubernetes 는 Container PaaS(또는 CaaS)라 불리는 컨테이너 자동편성이다.

Cloud Foundry는 애플리케이션 빌드 결과물 또는 소스 코드를 직접 배포하면, 해당 애플리케이션 실행환경을 플랫폼에서 자동으로 생성하고 구동하여 서비스를 제공할 수 있도록 해 주는 반면, Kubernetes는 개발자가 직접 컨테이너를 생성하고 이를 컨테이너 레지스트리에 저장하고 저장된 컨테이너 이미지를 배포하면 배포된 컨테이너 이미지를 애플리케이션 실행환경으로 생성하고 구동하여 서비스를 제공할 수 있도록 해 주는 차이점이 존재한다. 〈표 1〉은 Cloud Foundry와 Kubernetes 간의 차이점을 항목별로 보여준다. 아래에서 보여지는 비교표에서 보는 바와 같이 Kunernetes는 개발자에게 더 많은 자유도와 유연성을 제공하는 반면, Cloud Foundry는 관리 가능한 환경에서 개발

〈표 1〉 Cloud Foundry와 Kubernetes 비교
〈Table 1〉 Comparison Between Cloud Foundry and Kubernetes

Cloud Foundry vs. Kubernetes			
Type	Categoty	Cloud Foundry	Kubernetes
Concept	PasS Type	Application PaaS	Container PaaS(aka CaaS)
Development Environment	Main user interface	CLI	CLI
	Supported IDE	Provide Eclipse Plus-in	Provide Eclipse Plus-in
Runtime Environment	Application Deployment unit	Application Build outputs	Container image
	Image Repository bofore deployment	Artifactory(Optional)	Docker Registry(Required)
	Responsibility of application instant (container) generation	Buildpack within platform	Developer or deployment pipeline
	Container orchestrator	Diego within platform	Kubernetes itself
	Types of sopporting containers	Garden Linux	Docker / RunC / OCI
	Routing System	DNS	IP Address
	Container - Storage connection	External storage and API connection	Direct container - storage connection
Service Environment	Service cataloque connection	Connection through individual back-end service brokers	Helm Chart connection through helm
	Service provisioning / binding method	Using Open Service Broker APIs	Using Open Service Broker APIs
Operation Environment	Responsibility of Infrastructure confituration	Bosh within platform	Operator or IaaS automation tool
	Monitoring method	Separate metric provision, storage and visualization	Separate metric provision, storage and visualization
	Metering framework	CF Abacus	Operator Metering
Security	Support multi-tenancy	Organization / Space, Quota Management	Namespace(Virtual Cluster), Quota Management

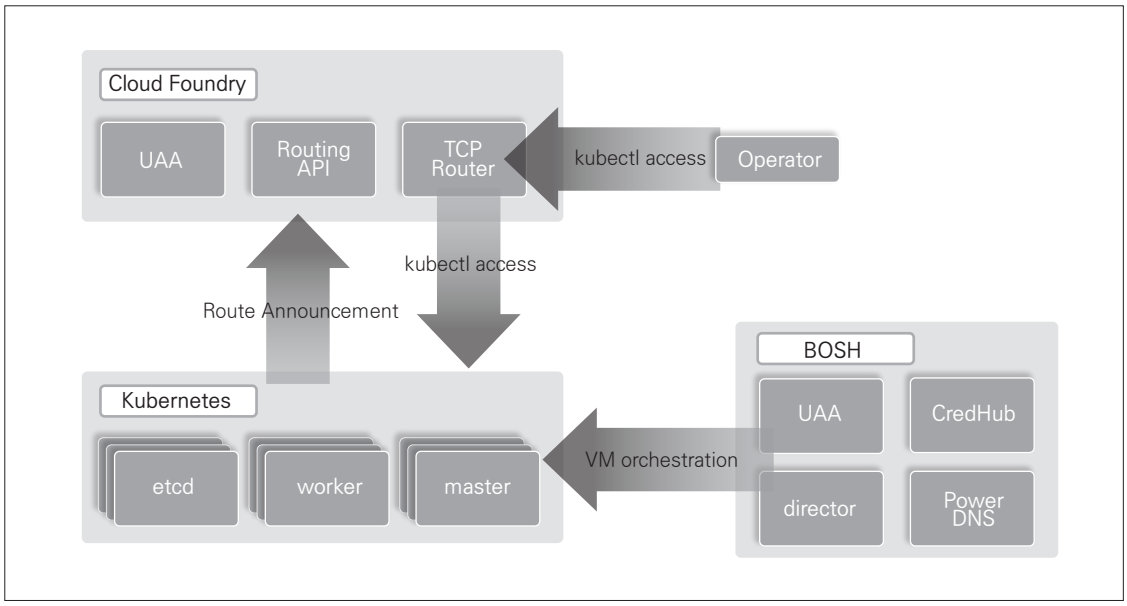
생산성을 극대화 할 수 있도록 지원해 주는 특성이 있다. 또한, 내부구조에서는 Cloud Foundry는 Garden Linux를 컨테이너 기술로 사용하는 반면, Kubernetes는 Docker를 중심으로 표준 컨테이너 기술들을 수용하고 있는 점에서 차이를 보인다. 백엔드 서비스를 연결하는 방법은 동일하게 Open Service Broker 표준을 수용하여, 이 표준에 맞게 개발된 백엔드 서비스를 표준 인터페이스로 프로비저닝 받고 애플리케이션과 바인딩 하여 동작할 수 있도록 해준다.

2. 개방형 클라우드 플랫폼과 Kubernetes 접목 방안

본 장에서는 최근 전 세계적으로 가장 많이 활용되는 컨테이너 플랫폼(CaaS)인 Kubernetes와 애플리케이션 플랫폼(PaaS) 개방형 클라우드 플랫폼의 3가지 접목 방안을 살펴본다. 첫 번째 접목 방안은 Kubernetes의 표준 인프라 인터페이스인 Cloud Provider Interface 활용하여 개방형 클라우드 플랫폼을 Kubernetes 클러스터 위에 Bosh로 배포하여 사용하는 방법이다. Kubernetes 상에 Cloud Foundry의 배포 및 구성은 〈그림 7〉과 같이 이루어진다.

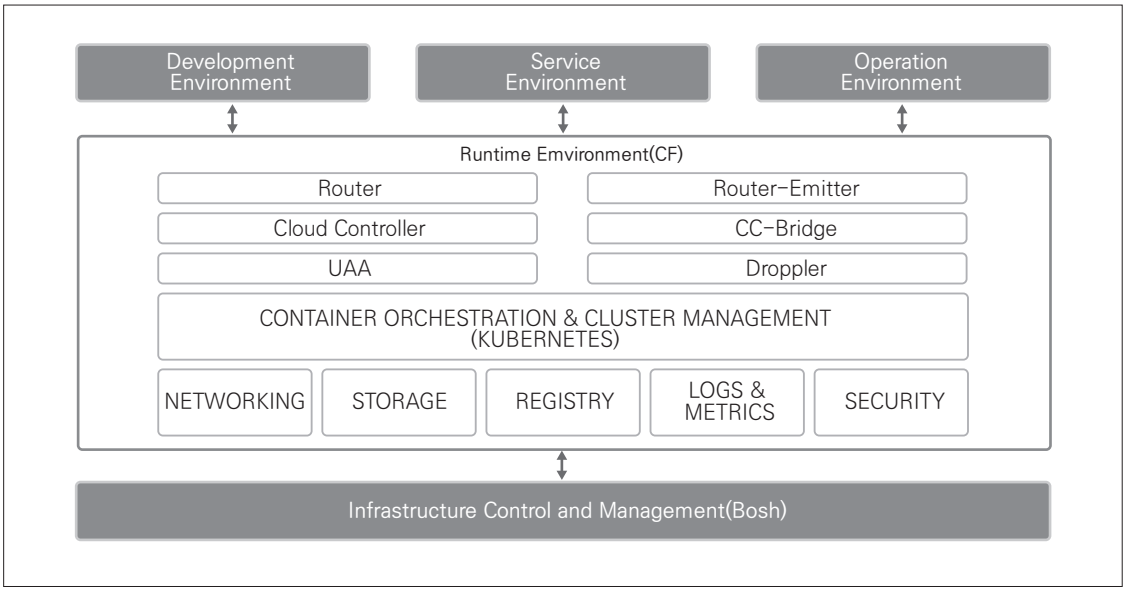
〈그림 7〉에서 Bosh는 Cloud Foundry 내부 구성요소들을 Docker Container로 생성하고, 생성된 Container들이 Kubernetes에 배포되어 Cloud Foundry가 동작하게 되는 방법이다. 위 방법의 장점은 현재 기업에서 운영 중인 Kubernetes 클러스터가 있을 시, 별도의 인프라 확충 없이 손쉽게 Cloud Foundry를 설치하여 활용할 수 있도록 지원한다. 하지만 이 모델은 Kubernetes 상에서 Cloud Foundry가 가상화되어 구축된 후, 그 위에 다시 응용서비스가 올라가게 되므로 Cloud Foundry 상에서 운영되는 응용서비스의 성능 하락이 불가피하게 발생하므로 테스트 환경에서만 사용할 수 있는 구성이다.

두 번째로 〈그림 8〉과 같이 개방형 클라우드 플랫폼에서 사용하는 컨테이너 자동편성인 Diego를 대신하여 Kubernetes를 사용하여 컨테이너를 관리하는 방



source: Strukhoff & Turol(2017)

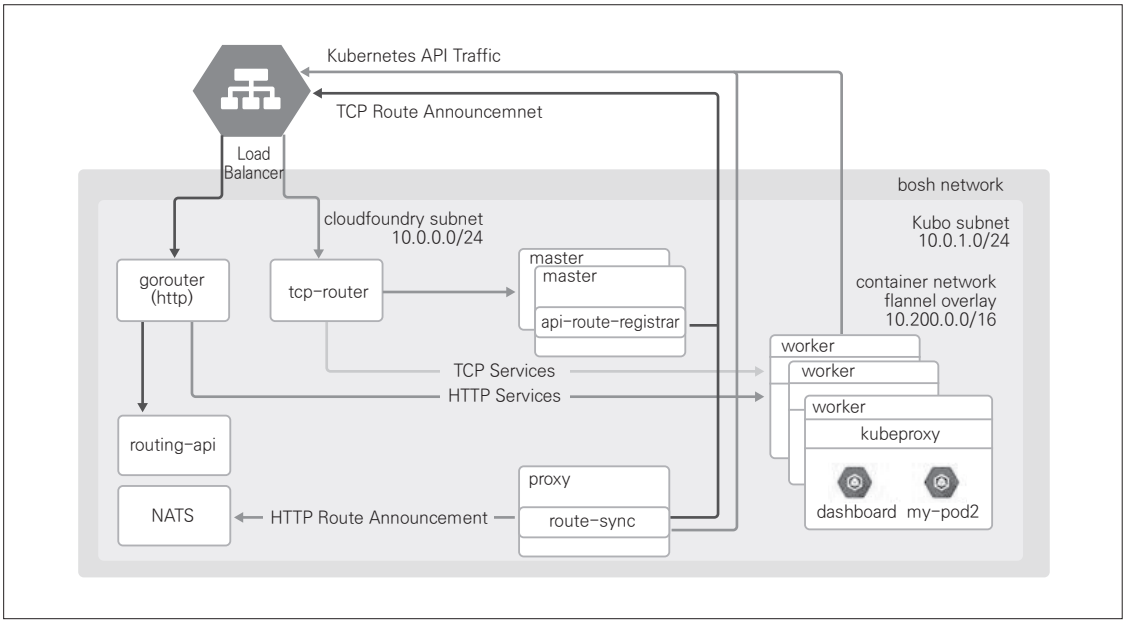
〈그림 7〉 Kubernetes 클러스터에 개방형 클라우드 플랫폼을 배포하는 방안
〈Fig. 7〉 Plan of Deploying Open Cloud Platform on Kubernetes Cluster



source: Seo, et al.(2016)

〈그림 8〉 개방형 클라우드 플랫폼 컨테이너 관리를 Kubernetes로 변경하는 방안

〈Fig. 8〉 Plan of Replacing Container Management of Open Cloud Platform with Kubernetes



source: Strukhoff & Turol(2017)

〈그림 9〉 개방형 클라우드 플랫폼과 Kubernetes의 융합 방안

〈Fig. 9〉 Plan of Fusing Open Cloud Platform and Kubernetes

안으로, Redhat의 Openshift와 유사한 아키텍처로 전환하는 내용이다.

이 방법은 이종의 두 플랫폼(Cloud Foundry, Kubernetes)의 관리비용으로 인한 비용 낭비 없이 Kubernetes의 강력한 컨테이너 관리 기능을 도입할 수 있도록 한다. 하지만 이 모델은 Cloud Foundry의 Diego와 garden컨테이너를 Kubernetes와 Docker 컨테이너로 전환 시 막대한 소스 수정 및 테스트가 필요하며, 또한 Diego와 garden 컨테이너의 장점을 활용할 수 없는 단점이 존재하여 현실적으로 어려운 방안이라 할 수 있다.

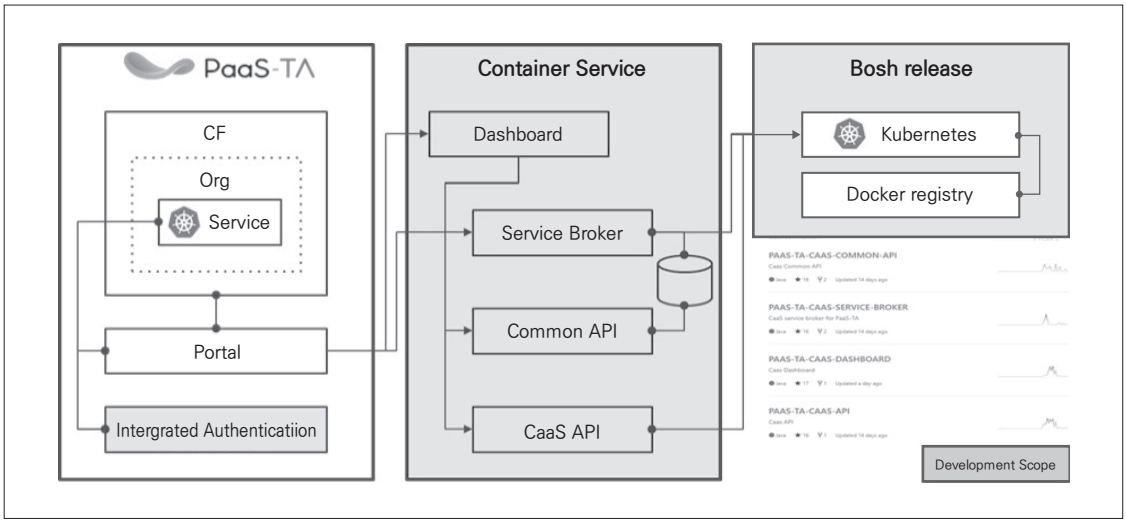
세 번째로 〈그림 9〉와 같이 개방형 클라우드 플랫폼과 Kubernetes를 동등한 레벨의 플랫폼으로 융합하여 사용하는 방안이다. 개방형 클라우드 플랫폼에서 Kubernetes 클러스터를 백엔드 서비스 형태로 제공하며, 시스템 관리 소프트웨어인 Bosh를 이용하여 Kubernetes 클러스터를 자동으로 확장 할 수 있게(Auto Scaleable) 서비스를 구성하고 Multi tenancy 기능을 제공한다. 사용자가 개발하는 최종 애플리케이션은 개방형 클라우드 플랫폼에 배포하고, 개방형 클라우드 플랫폼에 배포한 애플리케이션에서 사용하려는 자원이 개방형 클라우드 플랫폼에서 제공하지 못 할 경우 Docker Hub 등 레지스트리에 존재하는 컨테이너 이미지를 Kubernetes 클러스터에 배포하여 사용가능하도록 서비스를 구성한다. 융합형으로 사용하게 되면 기존 Docker로 개발한 서드파티(Third-party) 서비스의 빠른 개시(launching)가 가능하다.

3. 개방형 클라우드 플랫폼과 Kubernetes의 최적 융합방안 제시

개방형 클라우드 플랫폼은 애플리케이션의 코드를 중심으로 하는 개발 플랫폼으로 개발 언어 혹은 프레임워크에 관계없이 원하는 클라우드 환경에 코드를 실행하고 애플리케이션 전체 라이프사이클(Application Lifecycle)을 관리해 준다. 또한, 오픈 서비스브로커 API(Open Service Broker API)를 통하여 손쉽게 다양한 SaaS 및 백엔드 서비스와 연동할 수 있도록 지원한다. 반면 컨테이너 자동편성 플랫폼인 Kubernetes는 효율적인 컨테이너 관리 기능과 편리한 컨테이너 편성 및 유지보수 기능을 제공한다. 또한, 기존 개발된 애플리케이션을 컨테이너화하여 클라우드 환경에서 손쉽게 운영·재사용될 수 있도록 지원한다.

본 논문에서는 각 플랫폼의 장점을 함께 지원할 수 있는 융합 모델 아키텍처를 제시한다. 개방형 클라우드 플랫폼은 클라우드 기반의 신규 응용서비스를 개발할 때, 표준화된 환경과 다양한 개발지원 도구를 제공하므로 빠르고 효율적인 서비스 개발을 지원한다. 하지만 개발 방법·도구가 플랫폼에 종속(Lock-in) 되므로 자유도가 떨어진다는 단점이 있다. 반면 컨테이너 플랫폼은 응용서비스의 개발과 컨테이너화를 위해 별도의 개발 환경 및 테스트 환경이 필요하다는 단점이 있지만, 컨테이너 관리의 높은 자유도를 제공한다. 또한, 국내의 많은 기업들이 기존 응용서비스의 클라우드화를 위하여 Docker 기반의 컨테이너 기술을 많이 활용하고 있어 Kubernetes를 활용하면 컨테이너화된 기존 응용서비스를 별다른 수정 없이 바로 사용할 수 있는 장점이 있다. 따라서 개방형 클라우드 플랫폼과 Kubernetes를 융합한 방식으로 제공함으로써 개방형 클라우드 플랫폼과 Kubernetes 기반 컨테이너 서비스가 서로 상생할 수 있는 생태계 기반을 제공할 수 있다. 본 논문에서 제안하는 융합 모델의 아키텍쳐는 〈그림 10〉과 같다.

위 그림에서 보는 바와 같이 Kubernetes 클러스터와 컨테이너 레지스트리는 시스템 관리 소프트웨어인 Bosh에 의해 설치 구성되어 개방형 클라우드 플랫폼의 백엔드 서비스로 제공된다.(Bosh 동작에 대한 자세한 설명은 다음 장 참조) Kubernetes 클러스터를 필요로 하는 조직(ORG)은 개방형 클라우드 플랫폼 서비스 프로비저닝을 통해 Kubernetes 가상 클러스터(네임스페이스)를 할당받고 해당 가상 클러스터에 대



〈그림 10〉 융합형 컨테이너 서비스 아키텍처
〈Fig. 10〉 Fusion-type Container Service Architecture

한 독점적 접근이 가능해진다.(상세한 기술적 설명은 다음 장 참조)

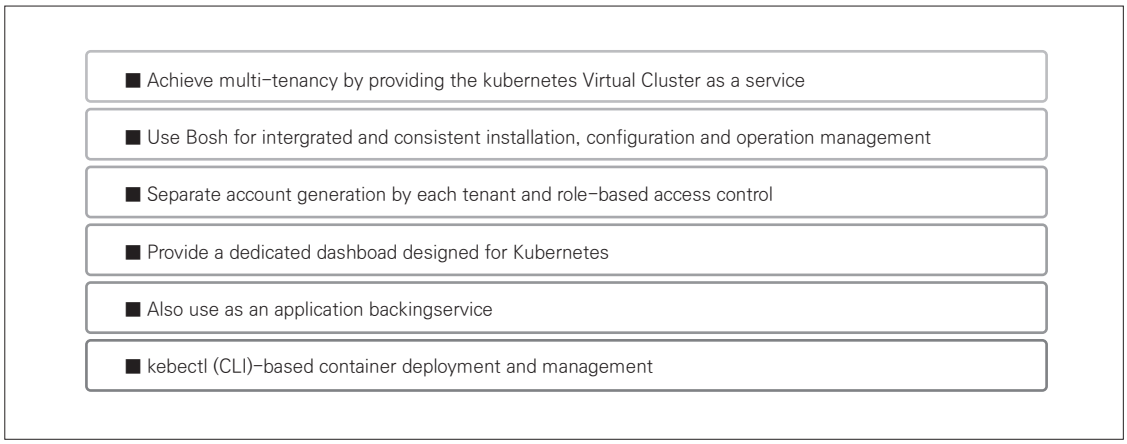
이상과 같은 아키텍처를 통해 개방형 클라우드 플랫폼과 컨테이너 환경을 동시에 제공함으로써 이종의 두 플랫폼 장점만을 가지는 새로운 플랫폼 기능을 제공할 수 있다. 이를 통해 애플리케이션 플랫폼과 컨테이너 플랫폼 간의 유연한 조합 기능을 제공하며, 또한, 서로 독립된 인프라 상에서 구축되므로 가상화 중첩으로 인한 성능 저하 없이 서비스를 제공할 수 있다. 개발자는 특정한 플랫폼 제약 없이 필요에 따라 응용서비스 또는 컨테이너의 Multi-tenancy 환경을 손쉽게 구성할 수 있다. 이를 통해 컨테이너 환경에 익숙하지 않은 개발자 혹은 관리자가 컨테이너 환경을 쉽게 구축하고 모니터링·관리할 수 있도록 지원한다.

3) 설계 원칙 및 기술 구조

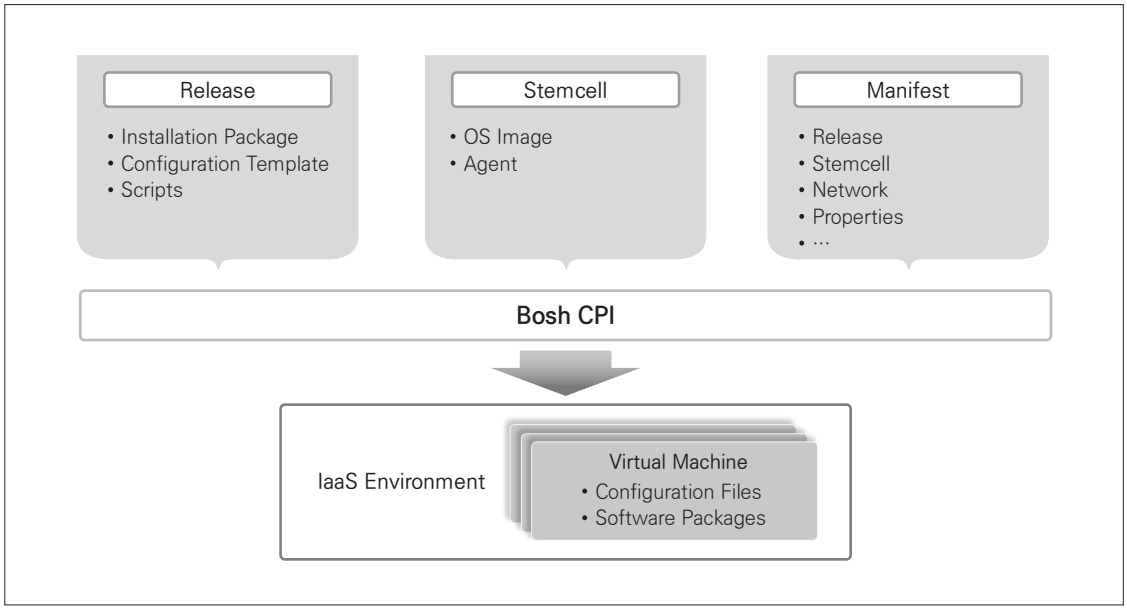
개방형 클라우드 플랫폼과 Kubernetes를 융합하기 위해서 〈그림 11〉과 같은 설계 원칙을 수립하였다. 이 원칙은 상이한 두 이종 플랫폼을 통합 과정에서 각자가 가진 강점을 살리고 상호 간의 부족한 점을 보완하여 시너지를 극대화하기 위하여 도출된 설계 원칙이다.

개방형 클라우드 플랫폼에서는 Kubernetes를 독립적인 클러스터로 제공하지 않고 서비스 형태로 제공함을 설계 원칙으로 정의하였다. 즉 개방형 클라우드 플랫폼 포털 화면에서 서비스 카탈로그로 제공되는 컨테이너 서비스를 신청하면 테넌트 하나가 생성이 되어 지원되는 구조이다. 내부적으로 멀티테넌시를 지원하기 위하여 Kubernetes 가상 클러스터(네임스페이스)를 생성하여 서비스 신청자에게 프로비저닝 되어 제공된다. Kubernetes 가상 클러스터는 Kubernetes 클러스터의 논리적 구분 단위이며 Kubernetes 서비스 어카운트 계정의 권한 관리로 접근이 제어되게 된다. 다른 Kubernetes 가상 클러스터와 구분하기 위하여 엑세스 시 유일한 네임스페이스를 사용하게 된다.

Kubernetes 클러스터를 설치, 구성, 운영하는 소프트웨어인 Bosh는 분산 시스템을 효과적으로 배포·관리하기 위한 오픈소스 프로젝트로 Cloud Foundry는 소프트웨어 설치와 운영, 그리고 자동화된 유지보수를 위하여 Bosh를 활용한다. 〈그림 12〉는 Bosh의



〈그림 11〉 개방형 클라우드 플랫폼과 Kubernetes 융합 원칙
〈Fig. 11〉 Principles of Converging Open Cloud Platform and Kubernetes



〈그림 12〉 Bosh의 구성요소
〈Fig. 12〉 Components of Bosh

4가지 구성요소를 보여준다.

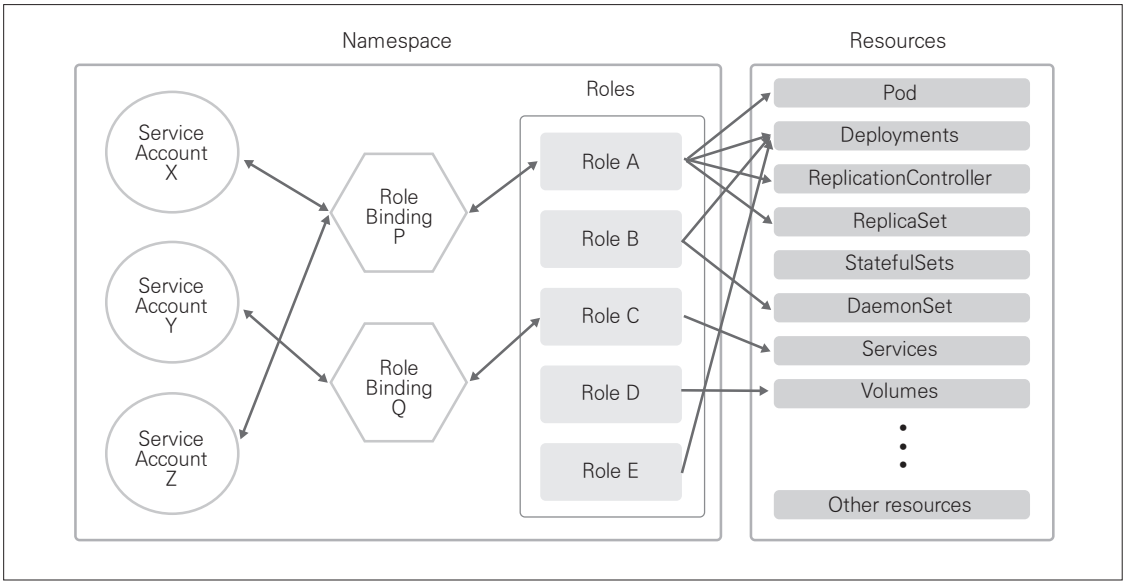
Bosh에는 클라우드 인프라스트럭처를 제어하는 영역이 별도로 존재하며 이 영역의 표준 인터페이스를 통하여 필요한 가상화 인프라를 생성하고 그 수명주기를 관리하게 된다. Bosh는 Release/Stemcell/Manifest의 3가지 구성요소로 이루어지며, 이 세 가지가 유기적으로 맞물려 동작하는 것이 특징이다. Release는 설치 대상 소프트웨어 패키지의 모음이며, Stemcell

은 설치되는 환경인 IaaS의 특수하게 제작된 가상머신 이미지로 이를 통해 설치 과정에서 필요로 하는 가상머신을 생성하게 된다. Manifest는 소프트웨어 패키지 설치 시의 구성정보를 담고 있다. 새로운 관리 대상 소프트웨어 발생하면 Bosh Release를 개발하여 적용한다. 개방형 클라우드 플랫폼 컨테이너 서비스도 Bosh Release로 만들어졌으며, 이에 따라 Bosh를 통한 클러스터 및 서비스 브로커 설치와 운영, 유지보수가 가능하다. Bosh를 통한 운영, 유지보수가 가능함으로써 복잡한 인프라 작업을 단순화 할 수 있으며, 클라우드 인프라 간 이식성을 보장한다.

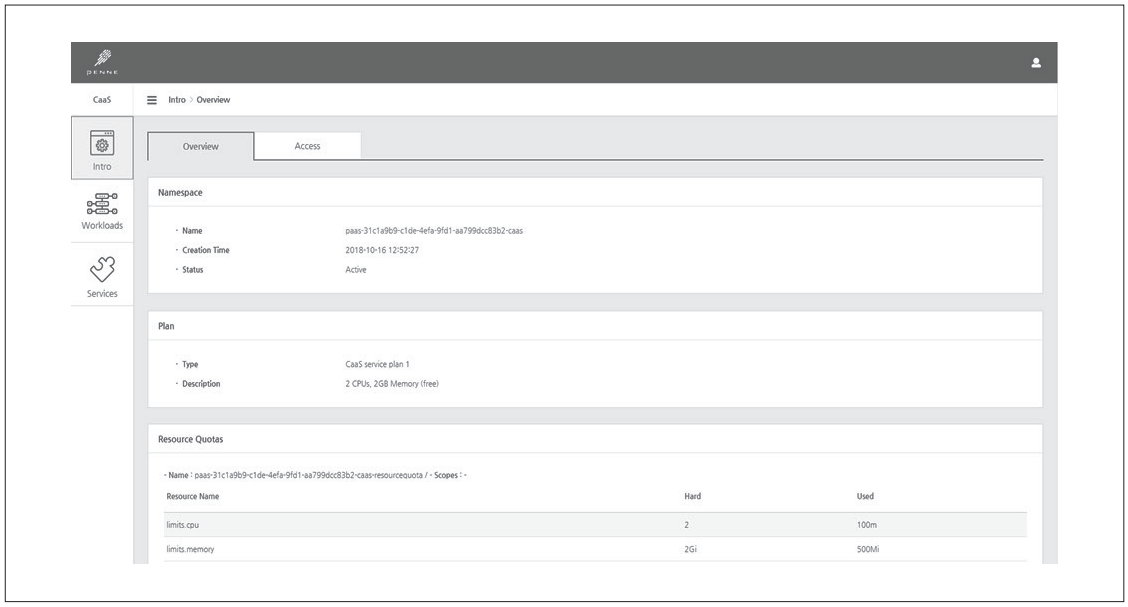
개방형 클라우드 플랫폼의 계정관리 체계와 Kubernetes 계정관리 체계를 통합하기 위하여, Kubernetes 서비스 프로비저닝 시 - 즉 Kubernetes 가상 클러스터 생성 시 -, 서비스 신청자의 개방형 클라우드 플랫폼 계정과 동일한 이름의 Kubernetes 서비스 어카운트를 생성하고 해당 계정에 생성된 Kubernetes 가상 클러스터에 대한 운영자 권한을 할당한다.

이후 신규로 생성되는 Kubernetes 가상 클러스터 내의 계정은, 가상 클러스터 운영자(테넌트 운영자)에 의해 초대되고 권한을 할당받게 된다. 새로운 계정도 테넌트 운영자와 동일하게 Kubernetes 서비스 어카운트로 생성되고, Kubernetes 리소스에 대한 Role을 할당을 받아 접근제어가 구현된다. 〈그림 13〉은 롤 바인딩(Role Binding)이 실행되는 Kubernetes 내부 구조를 도식화 한 것이다.(자세한 내용은 Kubernetes 권한관리 문서 참조)

생성된 Kubernetes 가상 클러스터에 애플리케이션을 배포하는 등 운영을 위해서는 Kubectl CLI를 사용한다. 개방형 클라우드 플랫폼에서는 컨테이너 서비스에 대한 대시보드를 자체적으로 제공하여 현재 컨테이너 서비스 상태를 확인할 수 있도록 해 준다. 〈그림 14〉는 개방형 클라우드 플랫폼에서 제공하는 대시보드를 보여준다. 대시보드에서는 Kubernetes 가상 클러스터의 워크로드(배포된 Pod와 복제 세트 정보)를 중신으로 하는 사용 상태 정보를 디스플레이 하며, Kubectl CLI를 실행하기 위한 환경설정 및 명령어 실



〈그림 13〉 쿠버네티스 롤 바인딩 구조
〈Fig. 13〉 Role Binding Architecture in Kubernetes



〈그림 14〉 개방형 클라우드 플랫폼의 쿠버네티스 대시보드
〈Fig. 14〉 Kubernetes Dashboard on Open Cloud Platform

행 절차를 운영체제별로 제공한다.

Ⅳ. 결론

본 연구에서 언급한 것처럼 한 가지의 컨테이너 관리 체계로는 사용자의 다양한 요구에 대응하기가 힘든 시대로 접어들었다. 또한, 앞으로는 많은 기술이 서로 융합하여 제품 간 서로 상생할 수 있는 생태계를 조성하는 세상이 될 것으로 예상한다. 특히 4차 산업혁명 시대는 빅데이터 분석, 인공지능, 로봇공학, 사물 인터넷, 무인 운송 수단, 나노 기술과 같은 6대 분야에서 새로운 기술 혁신을 바탕에 두고 정보통신기술(ICT)의 융합으로 이루어진 혁명의 시대이며, 이러한 기술은 클라우드 환경에서 제공해야 하는 방향으로 나가고 있다. 이에 따라 클라우드 환경에서 많은 애플리케이션이 컨테이너 기반으로 관리 및 운영되고 있어 특정한 컨테이너 기반으로 애플리케이션을 자동편성 및 관리하는 것보다 애플리케이션 실행환경 기반인 개방형 클라우드 플랫폼과 컨테이너 실행환경 기반인 Kubernetes를 융합하여 사용자에게 제공함으로써 시너지가 나올 것이라 예상된다.

본 연구에서는 개방형 클라우드 플랫폼에 Kubernetes를 접목하기 위하여, 운영관리를 위한 Bosh 기술과 백엔드 서비스를 통합하는 표준 서비스 브로커 기술을 활용하고, Kubernetes의 가상 클러스터의 생성과 접근제어, 자체 대시보드를 제공하는 독자적인 융합 방안을 제공함으로써 플랫폼 사용자와 운영자 모두를 만족시킬 수 있는 통합 플랫폼을 제시하여 국내 클라우드 플랫폼 생태계 활성화에 기여할 것으로 예상한다.

다만 본 연구의 내용의 완성도를 높이기 위해 명령어 기반(CLI) 기반의 사용자 인터페이스를 GUI 기반으로 개선하고, 개방형 클라우드 플랫폼의 CI/CD 배포 파이프라인을 Kubernetes 배포까지 지원할 수 있도록 확장 개발 하는 개선이 후속 작업으로 진행될 필요가 있다. 또한, 새롭게 떠오르는 서버리스 컴퓨팅 연구를 통해 FaaS(Function-as-a-Service) 기술도 플

랫폼과의 융합을 적극적으로 검토해야 한다.

■ References

Ahn, J., Lee, K., So, D., Seo, B. & Kim, E. (2015). *A Strategy for Developing the Open Cloud Platform for Government*.

{안재석·이기영·소대영·서보국·김은주 (2015). 〈정부를 위한 개방형 클라우드 플랫폼 구축 전략〉.}.

Amazon Web Services, Inc (2018). "Amazon EKS-Managed Kubernetes Service." https://aws.amazon.com/ko/eks/ (Retrived on November 29, 2018).

Cloud Foundry Foundation (2018). "Cloud Foundry-Open Source Cloud Application Platform." http://www.cloudfoundry.org/ (Retrived on November 29, 2018).

Conway, S. (2017). "Survey Shows Kubernetes Leading as Orchestration Platform." https://www.cncf.io/blog/2017/06/28/survey-shows-kubernetes-leading-orchestration-platform/ (Retrived on November 29, 2018).

Doh, S. (2018). *Korea Software Forecast, 2018- 2022: 2017 Year-End*. Seoul: IDC Korea.

IBM (2018). "IBM Cloud Container Service." https://www.ibm.com/cloud-computing/bluemix/ko/containers/ (Retrived on November 29, 2018).

Kim, K., Lee, G., Kim, T., Choi, J., Ha, S., Jung, Y. & Jin, S. (2018). "Kubernetes Architecture for Cloud Services." *Information & Communications Magazine, 35*(11), 11-19.

{김경일·이규진·김태현·최제민·하수정·정양재·진성근 (2018). 클라우드 서비스를 위한 쿠버네티스 구조. 한국통신학회논문지 〈정보와 통신〉, 35권 11호, 11-19}.

Linux Foundation (2018). "Home Page-Cloud Native Computing Foundation." https://www.cncf.io/ (Retrived on November 29, 2018).

Microsoft (2018). "AKS(Azure Kubernetes Service)." https://azure.microsoft.com/ko-kr/services/kubernetes-service/ (Retrived on November 29, 2018).

National Information Society Agency (2018). "PaaSTA Inside: Open Ecosystem." https://paas-ta.kr/intro/forwarding_result (Retrived on November 29, 2018).

{한국정보화진흥원 (2018). "파스-타 인사이드: 개방형 생태계." https://paas-ta.kr/intro/forwarding_result (검색일: 2018.11.29.)}.

Pivotal Software, Inc (2018). "Pivotal Cloud Foundry (PCF)." https://pivotal.io/platform/ (Retrived on November 29, 2018).

Seo, B., Kim T. & Kum, D. (2016). "Case Study of Open Cloud Platform Using Open Source." *Korea Information Processing Society, 23*(6), 11-21.

{서보국·김태현·금득규 (2016). 오픈소스를 활용한 개방형 클라우드 플랫폼의 적용 사례. 〈정보처리학회지〉, 23권 6호, 11-21.}.

Strukhoff, R. & Turol, S. (2017). "Kubo Enables Kubernetes Environments Managed by Cloud Foundry's BOSH." https:// www.altoros.com/blog/nt-only-for-cloud-foundry-kubo-enableskubernetesdeployments-with-bosh/ (Retrived on November 29, 2018).

The Kubernetes Authors (2018). "Production-Grade Container Orchestration-Kubernetes." https://kubernetes.