석사학위논문

Cloud 서비스 적용기술 분석 및 CI/CD 기술기반 PaaS 플랫폼 구현 방안 연구

-정보통신 및 제조업 분야 중심으로-

2022년

한성대학교 지식서비스&컨설팅대학원

스 마 트 융 합 컨 설 팅 학 과

스마트융합기술컨설팅전공

김 삼 현

석 사 학 위 논 문
지도교수 원종혁

Cloud 서비스 적용기술 분석 및 CI/CD 기술기반 PaaS 플랫폼 구현 방안 연구

; 정보통신 및 제조업 분야 중심으로

Analysis of applied technology of cloud services and research on how to implement a PaaS platform based on CI/CD technology

; Focusing on information communication and manufacturing sector

2021년 12월 일

한성대학교 지식서비스&컨설팅대학원

스 마 트 융 합 컨 설 팅 학 과

스마트융합기술컨설팅전공

김 삼 현

석 사 학 위 논 문
지도교수 원종혁

Cloud 서비스 적용기술 분석 및 CI/CD 기술기반 PaaS 플랫폼 구현 방안 연구

; 정보통신 및 제조업 분야 중심으로

Analysis of applied technology of cloud services and research on how to implement a PaaS platform based on CI/CD technology

; Focusing on information communication and manufacturing sector

위 논문을 컨설팅학 석사학위 논문으로 제출함

2021년 12월 일

한성대학교 지식서비스&컨설팅대학원

스 마 트 융 합 컨 설 팅 학 과

스마트융합기술컨설팅전공

김 삼 현

김삼현의 컨설팅학 석사학위 논문을 인준함

2021년 12월 일

심사위원장 홍 정완 (인)

심 사 위 원 박 인채 (인)

심 사 위 원 원 종혁 (인)

국 문 초 록

Cloud 서비스 적용기술 분석 및 CI/CD 기술기반 PaaS 플랫폼 구현 방안 연구
; 정보통신 및 제조업 분야 중심으로

한성대학교 지식서비스&컨설팅대학원
스 마 트 융 합 컨 설 팅 학 과
스 마 트 융 합 기 술 컨 설 팅 전 공
김 삼 현

현대의 정보통신기술(ICT)은 하드웨어와 소프트웨어 기술의 발전을 통해 AICBM, Metaverse 같은 새로운 패러다임을 만들어 왔다. 이는 클라우드 산업분야에도 영향을 미치게 되었으며 IaaS, PaaS, SaaS 형태 및 AIaaS, XaaS 형태로의 발전을 거듭하게 되었다. 특히 인터넷의 발달은 클라우드 산업분야의 새로운 시장을 형성하게 되었으며 COVID-19와 같은 글로벌 팬데믹 상황에서도 더욱 기술의 성장을 이어나갈 수 있게 되었다.

클라우드 컴퓨팅의 확산과 지속적 성장에 따라 소프트웨어 개발의 방법론, 동작시키는 방식, 배포하고 수정 후 재배포하는 방식 등 소프트웨어 산업 전반에 걸쳐 패러다임이 변화되고 있다. 이에 따라 클라우드 기술의 활용의 범위는 증가하였고 개발 환경을 지원하는 플랫폼의 요구사항 또한 증가하게 되었다. 이러한 환경은 클라우드 서비스의 확대 과정에서 이를 지원하기 위한

클라우드 서비스에 적용 가능한 플랫폼으로서 PaaS 플랫폼이 부각되게 되었다. PaaS 플랫폼은 IaaS나 SaaS 대비 시장 점유율이 비교적 낮지만 급변하는 시대적 요구사항을 반영하기엔 PaaS 플랫폼이 최우선 검토 대상이 될것으로 전망되어 진다. 그렇기 때문에 향후에는 PaaS 플랫폼의 성장이 IaaS 성장을 이끌어주고 결론적으로 클라우드 시장의 주도적 견인 역할을 할것으로 기대하고 있다.

현재 클라우드 시장을 주도하는 기업들은 클라우드를 활용하여 서비스 구축을 확대하는데 주력하고 있다. 소프트웨어(SW)를 개발하고 운영함에 있어서 필요한 플랫폼을 포함하여 클라우드 환경으로의 변화를 요구하면서 개발 환경이 바뀌게 되었다. 이는 PaaS의 수요가 늘어나고 있다고 판단하는 기준이 되기도 한다. PaaS 플랫폼은 과거 '가상머신(VM) 관리' 기능을 중심으로 구축되어 왔지만 최근에는 인프라를 관리하고 애플리케이션을 개발하며 운영의 지원 부분까지 적용범위가 확대 되고 있다. 이에 본 연구는 개발의 연속성을 위해 필요한 PaaS 플랫폼의 환경적 요소에 대한 수요자 중심의 요구사항을 반영한 가이드라인을 제시할 수 있었다.

애플리케이션을 개발하고 구현하는데 있어서 클라우드 서비스로 발전되는 환경에 대응하기 위해서는 컨테이너 기술과 컨테이너 오케스트레이션의 관리 기능이 필요하다. 이를 잘 활용하여 개발 생산성을 높이기 위한 CI/CD 오픈소스 기술을 적용한 시스템을 설계하고 구축하였다. 구축 시스템은 기업의 실제 책임자급 담당자의 요구사항을 분석하여 플랫폼으로 설계 및 구성하였으며 요구사항 분석을 기반으로 플랫폼 구성의 가이드라인을 설정하였다.

수요처의 요구사항을 분석하여 CI/CD 기술을 적용한 솔루션이 요구사항을 충족할 수 있다는 것을 확인할 수 있었다. CI/CD 기술을 PaaS 플랫폼에 적용하고 클라우드 서비스로 제공한다면 개발 생산성이 좋아지게 되어 PaaS 플랫폼 확산에도 매우 긍정적인 영향을 줄 것으로 기대한다.

주제어: PaaS, Cloud, CI/CD, DevOps, **컨테이너**, MSA, Cloud Native, OpenShift, Open Source

목 차

표 목 차

그 림 목 차

Ⅰ. 서론

1.1 연구의 배경 및 목적

1.1.1 연구의 배경

현대의 정보통신기술(ICT)은 하드웨어와 소프트웨어 기술의 발전을 통해 AICBM, Metaverse 같은 새로운 패러다임을 만들어 왔다. 이러한 변화는 클라우드 산업분야에도 영향을 미치게 되었으며 IaaS, PaaS, SaaS 형태 및 AIaaS, XaaS 형태로의 발전을 거듭하게 되었다. 특히 인터넷의 발달은 클라우드 산업 분야의 새로운 시장을 형성하게 되었으며 COVID-19와 같은 글로벌 팬데믹 상황에서도 더욱 기술의 성장을 이어나갈 수 있게 되었다.

최근 4차 산업혁명을 통한 디지털 전환에 힘입어 5G 통신, 인공지능(AI), 빅데이터(Big Data), 사물 인터넷(IoT) 등과 같은 신기술 적용속도 또한 빨라지게 되었다. 이러한 급변하는 환경에 적응하고 사용자 측면에서의 요구사항을 충족시키기 위해서는 다양한 기술과의 융합을 통한 플랫폼의 커스트마이징이 필요하게 되었다. 그리고 효율화 및 최적화 적용에 대한 완성도가 높아져야 하는 필요성도 높아지게 되었다. 향후 로봇 프로세스 자동화(RPA) 등과 같은 기술과의 접목을 눈앞에 두게 되면서 이러한 요구사항은 더욱 커지게 될 것으로 예상되고 있다.

클라우드 컴퓨팅이 확산되고 지속적으로 성장함에 따라 소프트웨어 개발의 방식, 동작하는 방식, 배포하고 수정 후 재배포하는 방식 등 소프트웨어 산업의 패러다임이 변화되고 있다. 클라우드 환경에서는 빈번하게 사용되는 기능을 위주로 효율성을 높이기 위한 방법으로서 모듈화하여 제공하게 되었다. 따라서 개발자는 제공된 해당 모듈을 조합하면 기존보다 쉽게 소프트웨어를 개발할 수 있고 짧은 기간에 글로벌 서비스 제공이 가능하게 되었다(정근훈, 2020; 락플레이스, 2021).

현재 클라우드 시장을 주도하는 기업들은 클라우드를 활용하여 서비스 구축을 확대하는데 주력하고 있다. 소프트웨어(SW)를 개발하고 운영함에 있어서 필요한 플랫폼을 포함하여 클라우드 환경으로의 변화를 요구하면서 개발 환경이 바뀌게 되었다. 이는 PaaS의 수요가 늘어나고 있다고 판단하는 기준이 되기도 한다. PaaS 플랫폼은 과거 '가상머신(VM) 관리' 기능을 중심으로 구축되어 왔지만 최근에는 인프라를 관리하고 애플리케이션을 개발하며 운영의 지원 부분까지 적용범위가 확대 되고 있다.

한편, IaaS는 기본적인 개념이 기존의 컴퓨팅 자원을 클라우드 환경으로 이관(Migration)하는 것으로서 기술적 이해수준이 어렵지는 않다. 즉, 전문성이 없는 사용자라도 클라우드의 장점을 파악하기가 수월하기 때문에 사용자의 수요가 많다고 할 수 있다. 한편 PaaS는 사용자의 요구사항 정도에 따라 애플리케이션을 수정하는 기간과 개발자 투입이 많이 필요한 단점도 분명이 있다. 그런데도 성장세가 높게 나타나는 것은 이러한 단점보다 인프라를 구축하고 유지하느라 시간과 비용을 투자할 필요 없이 새로운 애플리케이션을 만들고 배포하기 위한 환경을 확보하는 등의 장점이 훨씬 많기 때문에 향후 PaaS를 요구하는 시장이 많아질 것이라는 기대감을 가질수 있게 한다.

PaaS의 기술적인 핵심은 가상화 기술 중에서 컨테이너 기술이 있으며 이를 관리하도록 만들어진 오케스트레이션 기술로는 쿠버네티스가 대표적이라 할 수 있다. 컨테이너 기술은 2000년대 중반에 개념이 처음 소개되었으며 기업에서 SW 개발에 필요한 환경을 구축할때 유연성과 민첩성을 적용할 수 있게 만들어주는 기술이다. PaaS 핵심 기술이 컨테이너인 이유는 개발과 운영 환경을 공통된 환경으로 만들어 줌으로서 방법론적인 틈새를 좁혀주기 때문이다.

쿠버네티스는 구글이 만들었으며 오픈소스로 전환한 컨테이너를 관리하는 오케스트레이션(Orchestration)을 제공하는 오픈소스 기술이다. 쿠버네티스에서 핵심 기술은 '오케스트레이션(Orchestration)' 기능이라고 할 수 있다. 이 오케스트레이션 기능을 이용할 경우 컨테이너의 양이 아무리 많아도 자동으로 관리할 수 있는 큰 장점을 제공한다(Kubernetes, 2019).

하지만 쿠버네티스 솔루션은 매우 많은 오픈소스가 결합하여 있어서 모든

기능을 제대로 다루기란 쉬운 일이 아니며 너무 많은 기능으로 인해 오히려 단점으로 인식되기도 한다. 그리고 쿠버네티스에 여러 가지 다양한 오픈소스가 결합되도록 설계할 수 있으나 이렇게 설계된 솔루션에 기능상 문제가 발생하게 되면 기술지원이 어렵다는 문제점도 가지고 있다. 따라서 쿠버네티스를 솔루션으로 제공한다는 것은 난이도가 상당히 높은 기술력이 필요하다고 할 수 있다.

PaaS 솔루션이 다른 클라우드 솔루션보다 개발이 어려운 이유로는 다양한 오픈소스를 쿠버네티스와 결합하여 관리할 수 있어야만 PaaS 솔루션의 개발이 완료단계라고 할 수 있기 때문이다. 실제 2018년 일부 대기업들이 쿠버네티스 솔루션 상용화를 시도했지만 결론적으로 실패했다는 평가가 거론되고 있는 점에서도 그 난이도가 얼마나 높은지 알 수 있다. 그리고 쿠버네티스는 신규 업데이트 공개를 분기 단위로 하고 있는데, 쿠버네티스를 업데이트하면 결합한 오픈소스들을 지속해서 추적하면서 관리해야 하는 번거로움도 해결해야 한다. 이러한 어려운 관리체계를 가지고 있기 때문에 기업들이 충분한 기술을 확보하고 자본을 투입하지 않으면 성공적인 결론을 얻기 어려운 상황이라고 할 수 있다.

또한 기술지원의 만족도가 낮은 것도 도입이 지연되는 이유 중 하나로 평가되고 있다. 오픈소스는 인터넷에서 내려받아 쉽게 쓸 수 있는 장점이 있지만 오픈소스 특성상 기술지원을 받기가 현실적으로 어렵다.

한편 오케스트레이션의 다양한 기능을 사용하려면 오케스트레이션을 위한 엔진이 필요해 지며 이를 지원하는 쿠버네티스가 엔진 역할을 하게 된다. 쿠버네티스를 통해 컨테이너를 생성하거나 삭제하기, 시작과 중단 시점에 대한 컨트롤, 스케줄링, 클러스터링, 로드 밸런싱 등의 기능을 통해 컨테이너로 구현한 애플리케이션을 서비스하는 모든 과정의 관리가 가능하다. 이러한 다양하고 편리한 기능을 사용하기 위한 연구가 필요하며 이러한 기능을 원활히 구현하기 위한 커스트마이징 작업에 있어서 효율성을 높이고 구축기간을 획기적으로 단축시킬 수 있는 클라우드 기반의 CI/CD 자동화 기술에 대한 연구가 필요해 졌다.

1.1.2 연구의 목적

본 연구는 급변하는 PaaS 플랫폼 요구환경에 더욱 능동적으로 접근이 가능한 오픈소스 기반의 PaaS 플랫폼 개발 환경의 필요성이 커지면서 이에 대한 해결점을 찾고자 함에 있다. PaaS 플랫폼 개발 환경의 능동적인 구현을 위해서는 기획 및 설계 단계부터 전반적인 구성요소의 분석이 필요하며 실질적인 개발 로드맵의 수립이 우선시 되어야 한다.

본 연구는 개발의 연속성을 위해 필요한 PaaS 플랫폼의 환경적 요소와 이에 상응하는 요구사항을 충족시키기 위한 가이드라인을 제시하는 데 목적을 두고 있다. 현재 제기되는 문제점으로서 기존에 구축되어 운영 중인 시스템은 잦은 변경 요청에 응대가 어려우며 개발을 진행하고 적용하는 단계에서 서비스 제공의 연속성이 문제로 주목받고 있다. 이러한 문제점을 해결하기 위해서 다음과 같이 연구를 진행하였다.

첫 번째 전반적인 클라우드 컴퓨팅 환경을 조명하고, 기술 요구사항을 분석한다.

두 번째 클라우드 환경에 적용될 수 있는 PaaS 플랫폼 및 CI/CD 환경을 설계한다.

세 번째 실제 기업의 책임 담당자 수요조사를 통해 필요 요구사항에 대한 분석을 진행하고 개선된 기술 요구사항의 방향성을 정립한다.

네 번째 설계된 내용을 바탕으로 PaaS 플랫폼과 CI/CD 환경을 구현하고 수요처 기업의 개선된 효과를 검증한다.

1.1.3 연구의 차별성

본 연구에서 구현하는 CI/CD 시스템의 경우, 클라우드 환경에서 구현된 PaaS 플랫폼과 연계될 수 있는 환경으로 오픈소스를 활용하여 컨테이너 기술을 통한 CI/CD 파이프라인을 설계 및 구현 방안을 제시하고 실 수요처의 요구사항을 반영한 시스템으로 개선된 효과를 검증한다는 측면에서 다른 연구와 차별성이 있다. 본 연구에서 구현한 PaaS 플랫폼은 CI/CD와 같은 자동화 도구의 기능이 보편화하도록 기능 개선을 함으로써 개발과 운영 측면에서 더

욱 활성화되는 환경을 마련할 수 있을 것으로 기대한다.

1.2 연구의 방법 및 구성

본 논문의 구성은 다음과 같다.

제1장인 서론은 클라우드 시장의 현황과 개발 환경의 변화, 클라우드 네이티브 환경을 위해 필요한 요구사항을 기술하고 연구의 당위성을 설명하였다.

제2장은 본 논문의 이론적 배경을 바탕으로 클라우드 컴퓨팅의 개념과 분류된 클라우드에 따라 장점을 조명하고 기술적 고려를 위한 요소들을 확인하였다.

제3장은 PaaS 플랫폼의 설계를 통해 클라우드 환경에서 요구되는 시스템적인 측면과 애플리케이션 측면의 고려 사항을 도출하여 설계기준을 정의하였다.

제4장은 수요자 중심의 요구사항 설계를 위해 1년 이내 도입예정인 기업의 책임자급을 대상으로 한 설문내용을 분석하여 PaaS 플랫폼과 CI/CD 환경을 구현하였다.

제5장은 본 연구의 결과에 대해 의미 있는 결과도출에 대해 정의하였으며 본 연구의 한계점과 향후 연구 방향을 제시하고 마무리한다.

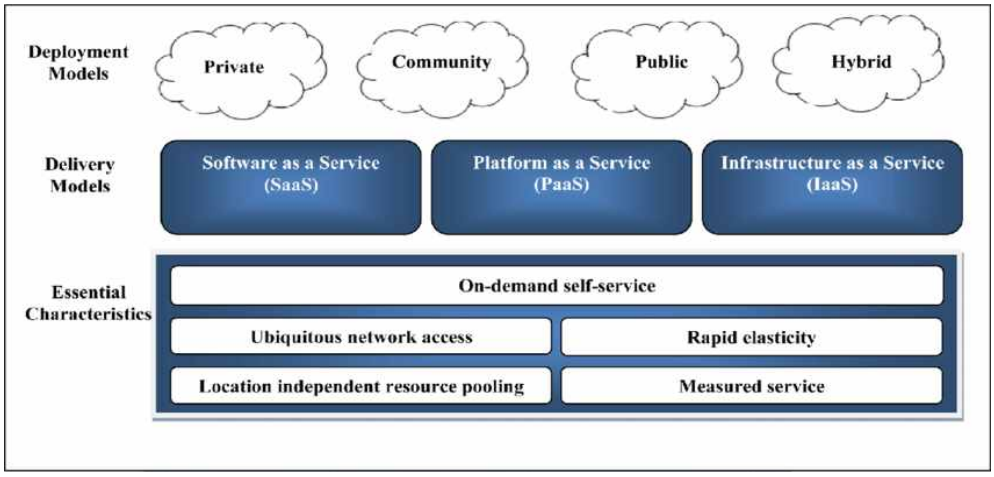
II. 이론적 배경

2.1 클라우드 컴퓨팅

2.1.1 클라우드 컴퓨팅 정의

클라우드라는 용어는 1961년 존 메카시(John McCarthy)가 유틸리티 컴퓨팅을 최초로 예언하는 데서 언급이 되었다. Michael Armbrust(2009)는 인터넷 연결 방식을 통해 서비스로 제공되는 애플리케이션과 이러한 서비스를 제공하는 데이터 센터의 하드웨어 및 시스템 소프트웨어를 모두 의미하는 것을 클라우드 컴퓨팅이라 정의하였다.

그리고 미국 국립 표준 기술원(NIST)에서는 관리적 측면에서 최소한의 노력만이 필요하며 서비스 공급자와 수요자 간의 상호 작용만으로도 신속하게 프로비저닝 및 배포를 통해 구성할 수 있어야 한다고 하였다. 따라서 네트워크, 서버, 스토리지, 애플리케이션 및 서비스 등 컴퓨팅 자원의 공유집합으로서 언제 어디서든 인터넷 연결이 된다면 맞춤형(on-demand) 네트워크 접근이 가능한 모델이라고 설명하였다.



[그림 2-1] 클라우드 컴퓨팅 개념도 (NIST, 2011)

클라우드 컴퓨팅은 위 [그림 2-1]과 같이 필수 특성으로 주문형 셀프서비스가 있으며 유비쿼터스 네트워크 접근을 통한 확장형 컴퓨팅과 위치 독립적 자원을 제공, 신속하게 확장 가능한 서비스와 이렇게 제공하는 환경을 측정하는 서비스 등이 있다(NIST, 2011). 클라우드 서비스의 배포되는 모델로 IaaS, PaaS, SaaS 형태가 있고, 최종 배포 모델 측면으로는 프라이빗 클라우드, 퍼블릭 클라우드, 하이브리드 클라우드 형태가 있다.

국내외 IT 관련 주요 기관들이 클라우드 컴퓨팅에 대한 다양한 정의를 발표하고 있으며 중요 기관의 발표내용은 아래 [표 2-1]과 같다. 이를 종합적으로 요약을 해보면 클라우드 컴퓨팅은 일반적인 컴퓨팅 자원인 서버, 네트워크, 스토리지, Software 등을 필요한 만큼 서비스로 제공하는 기술로 정의할 수 있다.

[표 2-1] IT 관련 주요 기관별 클라우드 컴퓨팅 정의(NIPA, 2015; 연구자 재구성)[1)]

기관명	정의
Gartner	인터넷 기술을 통해 다수의 고객을 위한 높은 수준의 확장성을 갖춘 자원들을 서비스로 제공하는 컴퓨팅 형태
Forrester Research	정보통신 기반의 표준화 된 기능들이 인터넷 프로토콜로 제공되어, 시간과 공간의 제약이 없는 접근이 허용되고, 수요 변화에 따라 가변적이며, 사용량이나 광고를 통해 비용이 지급되고, 웹 또는 프로그램과 같은 인터페이스를 제공
Wikipedia	컴퓨팅 자원들이 인터넷으로 제공되는 형태로 인터넷을 통해 개발을 수행하는 컴퓨팅 기술을 활용
IBM	웹 기반의 애플리케이션을 활용하여 대용량 데이터베이스를 인터넷 가상공간에서 분산되게 하고, 이러한 데이터를 컴퓨터나 휴대전화, 모바일 기기 등 다양한 단말기에서 실행되게 하거나 사용할 수 있게 하여 주는 환경
Google	사용자 중심이나 업무 중심의 수많은(수백에서 수천) 컴퓨터를 연결하여 단일 컴퓨터로는 구현 불가한 대량의 컴퓨팅 자원을 이용과 활용하는 기술
NIST	클라우드 컴퓨팅은 서비스 제공자가 최소한의 관리나 개입만으로 빠르게 자원을 생성·제거·구성하고 공유된 컴퓨팅

1) 미래창조과학부·정보통신산업진흥원, 클라우드컴퓨팅법 해설서(미래부; NIPA, 2015. 12.)

	자원들을 어디서든 어떤 터미널인지 구분할 필요 없이 필요할 때 편리하게 네트워크에 접근하여 사용하도록 제공하는 모델
ETRI	정보통신 자원(서버, 네트워크, 스토리지, SW)을 필요한 만큼 대여하여 사용하고, 서비스 사용량의 부하에 따라 실시간으로 확장할 수 있도록 지원받으며, 원하는 고품질의 서비스를 사용하고 책정된 비용을 지급하는 컴퓨팅
AWS	클라우드 제공 플랫폼에서 계산력, 데이터베이스 저장공간, 애플리케이션과 기타 IT 자원을 필요한 요구사항에 따라 인터넷으로 제공하고 사용한 만큼만 비용을 지급하는 것
Microsoft	인터넷(“클라우드”)을 통해 서버, 스토리지, 데이터베이스, 네트워킹, 소프트웨어, 분석, 인텔리전스 등의 컴퓨팅 서비스를 제공하는 것
Red Hat	`클라우드 내에서 워크로드를 실행하는 활동이라 정의하며, 네트워크 전반에서 수평적으로 확장할 수 있는 리소스를 추상화, 풀링, 공유하는 IT 환경을 의미

최근의 클라우드는 IT 기술과 접목하면서 다양한 산업의 요구사항을 통해 클라우드 제공자 관점의 시스템적인 본질적 특징과 비용을 지급하고 이용하는 서비스 사용자 관점의 고려 사항이 반영된 영역으로 확대되고 있다. 이러한 특징을 미국 국립 표준 기술원(NIST)에서는 다섯 가지 본질적인 클라우드 특징으로 아래 [표 2-2]와 같이 분류하고 있다.

[표 2-2] 클라우드 컴퓨팅 본질적 특징(NIST, 2011)

본질적 특징	설명
주문형 셀프서비스 (On-demand self-service)	• 클라우드 서비스를 사용자가 개입하지 않고 자동으로 할당되는 서비스로 대상은 웹 애플리케이션, 서버 시간, 스토리지, 네트워크 접근 시간이 해당
광대역 네트워크 접근 (Broad Network Access(mobility))	• 사용자들은 다양한 접근 방식(휴대전화, 모바일 등)으로 언제 어디서나 클라우드 서비스에 접근할 수 있음
자원의 공동관리 (Resource Pooling)	• 물리적 및 가상 컴퓨팅 리소스가 클라우드에 집중되고 이러한 자원은 고객이 자신의 위치에 대해 통제할 수 없고 지식도 없다는 점에서 위치의 종속성이 없음
신속한 탄력성(Rapid Elasticity)	• 소비자의 요구에 따라 컴퓨팅 자원을 신속하고 탄력적으로 프로비저닝하고 출시하며 소비자는 이러한 자원을 무한한 자원으로 보고 언제든지

	원하는 수량으로 구매할 수 있음
측정 서비스 (Measured Services)	• 클라우드 자원과 서비스는 CSP에 의해 종량제 비즈니스 모델을 통해 모니터링 제어 및 최적화되고 소비자는 전기, 수도 및 가스를 활용하는 것과 유사한 방식으로 이러한 서비스를 활용함

또한 NIST에서 발표한 본질적 특징과 함께 Rashid, A. & Chaturvedi, A. (2019)는 추가적 특징을 분류하여 [표 2-3]과 같이 비용 효율성, 다중 활용성(Multi Tenancy), 확장성, 안정성, 규모의 경제, 커스터마이징, 효율적인 리소스 활용, 가상화 등으로 분류하였다.

본 연구에서 다루고자 하는 PaaS 플랫폼과 CI/CD 기술은 오케스트레이션 부분을 추가적 특징으로 정의할 수 있으며 대형화되는 클라우드 인프라 환경에서는 서비스 제공자 측면에서 중요한 요소라 할 수 있다.

[표 2-3] 클라우드 컴퓨팅의 추가적 특징(Rashid, & Chaturvedi, A. 2019; 연구자 재구성)

추가적 특징	설명
비용 효율성 (Cost effectiveness)	• 클라우드 서비스 공급자가 제공하는 서비스는 무료가 아닐 경우 매우 비용 효율적이며, 청구 모델은 사용량에 따라 지급되므로 인프라를 살 필요가 없어 유지보수 비용이 절감됨
다중 활용성 (Multi-tenancy)	• 클라우드는 동시에 여러 사용자에게 서비스를 제공하며, 이러한 사용자는 네트워크 수준, 호스트 수준 및 애플리케이션 수준에서 클라우드 리소스를 공유하지만 각 사용자는 사용자 지정된 가상 애플리케이션 인스턴스 내에서 격리됨
확장성 (Scalability)	• 클라우드 컴퓨팅의 인프라는 확장성이 매우 탁월하며, 클라우드 공급자는 클라우드 인프라 및 소프트웨어를 약간만 수정해도 클라우드에 새 노드와 서버를 추가할 수 있음
안정성 (Reliability)	• 여러 개의 중복 사이트를 활용한 높은 안정성으로 인해 클라우드는 재해 복구 및 비즈니스 중대한 작업을 위한 완벽한 솔루션 제공

규모의 경제 (Economies of scale)	• 규모의 경제를 활용하기 위해 클라우드는 최대한 크게 구성되며 클라우드를 저렴한 발전소와 가까운 곳에 배치하거나 저렴한 부동산에서 비용을 절감하는 등의 다른 고려 사항도 있음
커스터마이징 (Customization)	• 클라우드는 사용자 요구에 따라 인프라 및 애플리케이션 측면에서 커스터마이징하고 조정할 수 있는 재구성 가능한 환경 제공
효율적인 리소스 활용 (Efficient resource utilization)	• 필요한 기간만 리소스를 제공하면 이러한 리소스를 효율적으로 활용할 수 있음
가상화(Virtualization)	• 클라우드 컴퓨팅은 사용자가 모든 종류의 터미널을 통해 어디서나 서비스를 받을 수 있도록 하며, 필요한 리소스의 경우, 보이는 실체(entity) 대신 클라우드에서 가져옴 • 노트북이나 휴대전화를 이용하여 넷 서비스를 통해 원하는 모든 것을 할 수 있으며, 사용자는 언제 어디서나 손쉬운 방법으로 안전하게 데이터를 얻거나 공유할 수 있고 사용자는 단일 컴퓨터에서 완료할 수 없는 태스크를 완료할 수 있음
오케스트레이션 (Orchestration)	• 전반적인 컴퓨터 시스템 및 애플리케이션, 자동화된 서비스 관리와 설정 등을 조정하는 것을 말하며, 오케스트레이션은 인터넷 기술팀이 복잡한 태스크 및 워크플로우를 이전보다 쉽게 관리할 수 있도록 기능을 제공함

2.1.2 클라우드 컴퓨팅 기술 동향

클라우드 컴퓨팅 베이스의 서비스 제공은 CSP(Cloud Service Provider) 기준에서 보면 하드웨어 시스템 인프라가 갖춰져 있는 데이터 센터 형태의 구축이 선행적으로 진행되어야 하며 동적 자원할당, 데이터 동기화, 주문형 서비스와 같이 클라우드의 특징이 충족될 수 있는 기술적 요소의 솔루션들이 많이 요구된다.

클라우드 컴퓨팅의 주요기술에 관한 내용을 살펴보면 아래 [표 2-4]와 같이 최근에는 엣지 컴퓨팅 형태의 기술 요구사항으로 인프라의 확장과 연계 등이 중요한 기술요소가 되고 있으며 특히 컨테이너와 같은 가상화 기술을 통한 개발이 중요해 지고 있다.

[표 2-4] 클라우드 컴퓨팅 주요기술(NIPA, 2015)

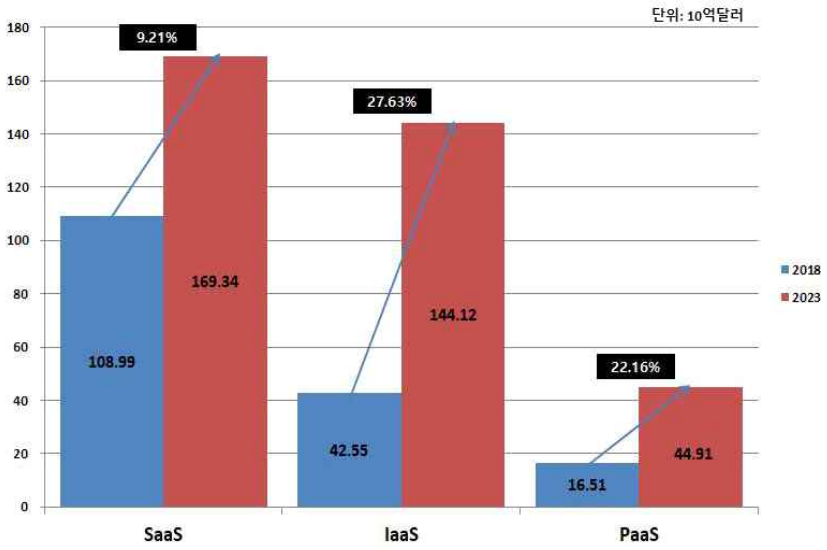
구분	주요특징	요소기술
가상화 기술	• 물리적인 하드웨어의 제약 조건을 가상화 기술을 통해 인프라를 구축하여 시스템 운영 • 한 대의 전산 시스템을 여러 대처럼 운영 또는 역으로 한대처럼 운영하는 기술	Hypervisor, 가상 I/O, Partition Mobility, Container, Resource Pool 등
대규모 분산처리	• 대규모의 서버 환경(수천 노드 이상)에서 대용량의 데이터를 분산처리 하는 기술	분산처리기술
오픈 인터페이스	• 인터넷을 통해 서비스를 이용하고 서비스 간 정보 공유를 지원하는 인터페이스 기술 • 클라우드 기반 SaaS, PaaS에서 기존 서비스에 대한 확장 및 기능 변경에 적용 가능	SOA, Open API, Web Service 등
서비스 프로비저닝	• 서비스 제공업체가 실시간으로 자원을 제공 • 서비스 신청부터 자원 제공까지의 업무 자동화, 클라우드의 경제성과 유연성 향상	자원 제공 기술
자원 유틸리티	• 전산 자원에 대한 사용량 수집을 통해 과금체계를 정립하기 위한 기술	사용량 측정, 과금, 사용자 계정 관리 등
SLA(서비스 수준 관리)	• 외부 컴퓨팅 자원을 활용하는 클라우드 서비스의 특성상 서비스 수준이라는 계량화된 형태의 품질 관리 기술 요구됨	서비스 수준 관리 시스템
보안 및 개인정보 관리	• 민감한 보안 정보를 외부 컴퓨팅 자원에 안전하게 보관하기 위한 기술	방화벽, 침입방지 기술, 접근 권한 관리 기술 등
다중 공유 모델	• 하나의 정보자원 인스턴스를 여러 사용자 그룹이 완전히 분리된 형태로 사용하는 모델 • SaaS를 제공하는 데 필수 요소로 꼽힘	멀티 테넌시

정보통신산업진흥원(NIPA) 클라우드법 해설서에서 정의하는 클라우드 컴퓨팅 주요 기술은 인프라 자원을 서비스로 제공하면서 필요한 기술로 가상화, 대규모 분산처리, 오픈 인터페이스와 같은 연계 기술, 서비스 프로비저닝과 여러 사용자에게 공유하면서 생길 수 있는 보안 및 개인정보 관리 기술 등이 있음을 설명하고 있다.

2.1.3 클라우드 컴퓨팅 산업 동향

2.1.3.1 국외 산업 동향

글로벌 클라우드 컴퓨팅 시장은 2018년 1,680억 5,000만 달러에서 2023년에는 3,583억 7,000만 달러로 증가하여 연평균 성장률 16.35%에 이를 것이라고 글로벌 리서치회사인 TechNavio사 에서는 전망하고 있다. 리서치 보고서에 의하면 클라우드 기술별 컴퓨팅 시장을 분석하면서 [그림 2-2]와 같이 IaaS, PaaS, Saas의 연평균 성장률을 전망 하였다. SaaS는 2018년을 기준으로 64.86%의 점유율을 나타내고 있으며 두 번째로는 IaaS가 25.32%, 그 다음으로는 PaaS가 9.82%의 점유율을 보이고 있다. PaaS의 경우 SaaS나 IaaS보다 점유율이 비교적 낮다는 것을 알 수 있다.

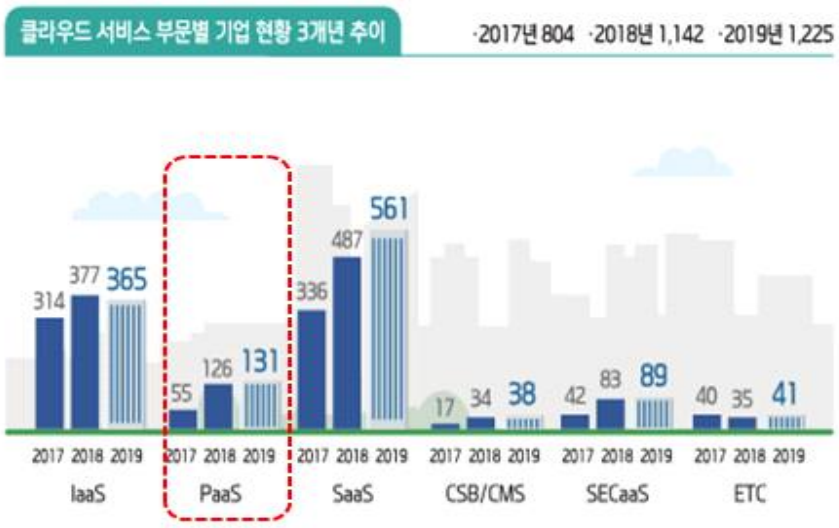


[그림 2-2] 글로벌 클라우드 컴퓨팅 기술별 시장규모(TechNavio, 2019)

2.1.3.1 국내 산업 동향

가트너가 전망한 국내 퍼블릭 클라우드 시장은 2021년까지 평균적으로 20.5%씩 증가하고 전체 매출 규모는 3.44조 원에 달할 것으로 전망하였으며, IDC 자료에서는 2018년 기준으로 IaaS 54.1%, SaaS 38.8%, PaaS 7.1% 순

으로 전망하였다. 또한 국내 정보통신산업진흥원에서 조사한 내용에 따르면 클라우드 공급 기업의 서비스 부문별 기업 현황은 SaaS가 561개로 전체 45.8%의 비중을 나타내고 있다. 그리고 IaaS(365개, 29.8%), PaaS(131개, 10.7%) 등의 순으로 비중의 순위를 나타내고 있다. 서비스 부문별 규모 3개년 평균 성장률을 살펴보면 PaaS 서비스가 54.3%로 가장 높게 조사되었다.



[그림 2-3] 국내 클라우드 컴퓨팅 시장의 서비스별 기업 현황(NIPA, 2020)[2)]

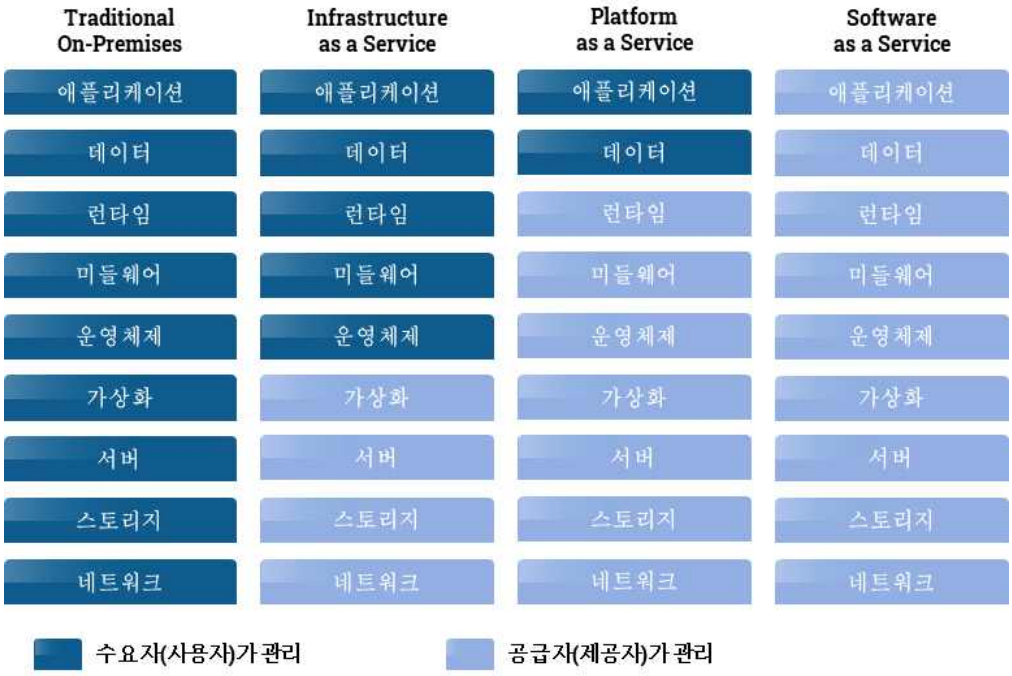
한편 과학기술정보통신부와 정보통신상업진흥원이 공동으로 연구한 2020 클라우드산업 실태조사 결과보고서에 의하면 위 [그림 2-3]과 같이 3개년 기업의 증가 추이를 보면 PaaS의 경우, IaaS나 SaaS 서비스 기업과 대비해서 비교적 낮다고 할 수 있다.

2.2 클라우드 컴퓨팅 서비스 유형

클라우드 컴퓨팅의 서비스 유형은 공급자(제공자)와 수요자(사용자) 측면에서 관리 및 사용이 구분되며, 기업이 사용하는 IT 환경을 기준으로 IaaS,

2) 과학기술정보통신부, 정보통신산업진흥원 2020 클라우드산업 실태조사 결과보고서

PaaS, SaaS 형태로 분류하고 있다. 아래 [그림 2-4]와 같이 전통적인 클라우드의 모델은 인프라 환경부터 애플리케이션까지 모두 지원하는 구조라고 한다면 하드웨어부터 가상화 단계까지 서비스로 제공되는 부분을 IaaS라고 하며, 운영체제, 미들웨어, 런타임 환경을 제공하는 것을 PaaS, 그리고 최종적으로 애플리케이션을 제공하여 필요한 기능에 따라 사용만 할 수 있게 제공하는 서비스를 SaaS라 정의할 수 있다.



[그림 2-4] 클라우드 서비스 모델 차이점(Red Hat, 2015)

2.2.1 Infrastructure as a Service(IaaS)

IaaS 서비스는 물리적인 Hardware(서버, 스토리지, 네트워크)와 가상화 기술을 이용하여 인프라 서비스를 제공하는 형태로 사용자는 필요한 만큼의 자원을 요청하여 할당받고 운영체제(OS), 미들웨어(WAS), 데이터베이스(DB) 등을 설치하여 활용하는 방식이다(위키피디아, 2021).

국제 인터넷 표준화 기구(IETF)에 따르면 기본으로 거론되는 클라우드 서

비스 모델은 컴퓨팅 기반기술로 가상 머신과 이를 서비스하는 데 필요한 서버, 네트워크, 스토리지 등의 자원들을 구독자에 대한 서비스로 제공하는 모델이다. IaaS는 하드웨어에 해당하는 컴퓨팅 자원과 위치, 데이터 파티셔닝, 확장과 보안 그리고 백업과 같은 기반기술의 자세한 요소로부터 사용자를 끌어내는 온라인 서비스들을 말한다.

2.2.2 Platform as a Service(PaaS)

PaaS 플랫폼 공급자는 일반적으로 개발을 위한 도구와 표준 환경, 그리고 배포 및 비용 책정을 위한 채널을 개발한다. PaaS 모델에서 클라우드 공급자들은 일반적으로 운영체제(OS), 프로그래밍 언어 실행 환경(runtime), 데이터베이스(DB), 웹(WEB) 서버를 포함한 컴퓨팅 플랫폼을 구현한다. 애플리케이션 개발자들은 기반이 되는 하드웨어 및 소프트웨어 계층을 직접 구축하는 비용이나 관리적 복잡성 없이도 서비스를 위한 소프트웨어 솔루션을 클라우드 플랫폼에서 개발 및 실행할 수 있다(위키피디아, 2021).

마이크로소프트 Azure와 구글의 App 엔진과 같은 PaaS에서 기반이 되는 컴퓨터와 스토리지 자원은 애플리케이션 수요에 대응하기 위해 자동으로 규모를 조정하고 클라우드 사용자는 수동으로 필요한 자원을 할당하지 않아도 된다. 부연 설명을 하자면 후자의 경우, 클라우드에서 실시간으로 편의성을 받고자 하는 목적이 있는 아키텍처 구성 형태로 제안되고 있다.

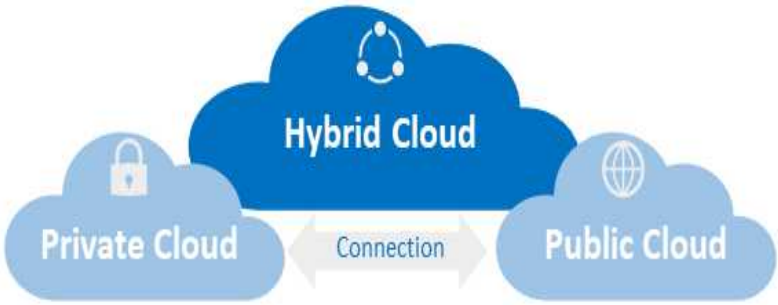
2.2.3 Software as a Service(SaaS)

SaaS 모델에서 사용자들은 애플리케이션과 데이터베이스에 대한 접근 권한을 가진다. 클라우드 공급자들은 애플리케이션을 실행하는 인프라스트럭처와 플랫폼을 관리한다. SaaS는 서비스를 받을 때 "주문형 소프트웨어 "(on-demand software) 방식을 사용하며 가격 정책은 종량제(pay-per-use)로 하거나 구독(Subscription) 비용 방식으로 책정된다. SaaS 모델에서 클라우드 공급자들은 클라우드 환경에 애플리케이션을 설치하고 운영까지 직접 관

리하며 클라우드를 통해 서비스를 받는 사용자들은 클라우드가 제공하는 소프트웨어에 클라이언트로 접근하여 사용한다. 클라우드 사용자들은 이러한 애플리케이션들이 실행되는 클라우드 기반의 인프라스트럭처와 플랫폼의 관리는 신경 쓸 필요가 없다. 이런 방식 때문에 클라우드 사용자 자신의 컴퓨터에 애플리케이션을 설치하고 실행할 필요가 없으므로 관리 및 지원이 단순해지면서 사용이 편리해 진다는 특징이 있다(위키피디아, 2021).

2.3 클라우드 컴퓨팅 서비스 배포 유형

클라우드 서비스를 구성하기 위하여 배포하는 유형으로 컴퓨팅 인프라의 위치적 측면, 사용자 접근 방식, 리소스 및 서비스 제공 형태에 따라 퍼블릭 클라우드(공용 클라우드), 프라이빗 클라우드(내부 또는 사내 구현 클라우드), 하이브리드 클라우드(퍼블릭+프라이빗)와 같은 3가지 유형으로 분류된다.



[그림 2-5] 클라우드 컴퓨팅 서비스 배포 유형(Red Hat, 2019)

클라우드 컴퓨팅 서비스 배포 유형은 위 [그림 2-5]에서 보는 바와 같이 프라이빗 클라우드와 퍼블릭 클라우드가 독립적으로 운영되고 이를 네트워크로 연결하여 서비스를 제공하는 형태가 하이브리드 클라우드이다.

2.3.1 퍼블릭 클라우드(Public Cloud)

퍼블릭 클라우드는 클라우드 컴퓨팅 배포의 가장 일반적인 유형이다. 클라우드 리소스인 서버나 스토리지와 같은 인프라는 타사 클라우드 서비스 공급자가 소유하고 운영하며 인터넷을 통해 제공한다. 퍼블릭 클라우드를 사용할 경우 모든 하드웨어, 소프트웨어 및 기타 지원 인프라를 클라우드 공급자가 소유하고 관리한다.

퍼블릭 클라우드에서는 다른 조직 또는 클라우드 "테넌트"와 같은 하드웨어, 스토리지 및 네트워크 디바이스를 공유하며 웹 브라우저를 사용하여 서비스에 액세스하고 계정을 관리한다. 공용 클라우드 배포는 웹 기반 메일, 온라인 사무실 애플리케이션, 스토리지 테스트 및 개발 환경을 제공하는 데 자주 사용된다.

[표 2-5] 퍼블릭 클라우드 장점(NIPA, 2020)

장점	설명
비용절감	하드웨어 또는 소프트웨어를 구매할 필요가 없으며, 사용한 서비스의 요금만 지급
유지관리 불필요	서비스 공급자가 유지관리를 제공
제한 없는 스케일링 성능	주문형 리소스를 사용하여 비즈니스 요구사항을 충족
높은 안정성	광대한 서버 네트워크를 통해 실패를 방지

퍼블릭 클라우드는 위 [표 2-5]에서 언급된 장점과 같이 인프라 시스템의 관리 영역을 서비스로 제공받기 때문에 비용이 절감되고 유지관리가 불필요하다. 그리고 비용을 지불하고 사용하는 주문형 방식으로 제한 없는 스케일링 성능과 높은 안정성을 제공한다.

2.3.2 프라이빗 클라우드(Private Cloud)

프라이빗 클라우드는 단일 비즈니스 또는 조직에서 독점적으로 사용되는 클라우드 컴퓨팅 리소스로 구성된다. 프라이빗 클라우드는 조직에서 실제로

보유하고 있는 데이터 센터에 있거나 타사 서비스 공급자가 호스팅할 수 있다. 그러나 프라이빗 클라우드에서는 서비스와 인프라가 항상 프라이빗 네트워크에서 유지 관리되며, 하드웨어와 소프트웨어는 해당 조직에서만 전용으로 사용된다.

따라서 프라이빗 클라우드를 사용하여 조직이 특정 IT 요구사항을 만족하게 하도록 리소스를 쉽게 사용자 지정할 수 있다. 프라이빗 클라우드는 환경에 대한 제어 기능을 강화하려는 비즈니스에 중요한 작업이 있는 중견 및 대규모 조직 외에도 정부 기관과 금융 기관 등에서 도입 및 사용하는 경우가 많이 있다.

[표 2-6] 프라이빗 클라우드 장점(NIPA, 2020)

장점	설명
유연성 향상	조직에서 특정 비즈니스 요구사항을 충족하기 위해 클라우드 환경을 사용자 지정
보안성(제어 향상)	리소스가 다른 사용자와 공유되지 않으므로 더 높은 수준의 제어 및 개인정보 보호가 가능
스케일링 성능 향상	프라이빗 클라우드는 온-프레미스 인프라와 비교했을 때 대개 더 큰 스케일링 성능을 제공

프라이빗 클라우드의 경우는 기업 내부에 직접 구축한 환경적 요소 때문에 위 [표 2-6]에서 언급한 장점과 같이 특정 요구사항을 충족할 수 있는 유연성과 다른 사용자와 공유되지 않는 장점을 통해 보안성이 향상된다.

2.3.3 하이브리드 클라우드(Hybrid Cloud)

하이브리드 클라우드는 온-프레미스 인프라(또는 프라이빗 클라우드)를 퍼블릭 클라우드와 결합하는 클라우드 컴퓨팅의 유형이다. 하이브리드 클라우드를 사용하면 두 환경 간에 데이터와 앱을 이동할 수 있다.

많은 조직에서는 규정 및 데이터 주권 요구사항 충족, 온-프레미스 기술

투자 최대한 활용 또는 짧은 대기 시간 문제 해결 같은 비즈니스 요구사항 때문에 하이브리드 클라우드 접근 방식을 선택한다.

하이브리드 클라우드는 엣지(edge) 워크로드도 포함하도록 진화하고 있다. 엣지 컴퓨팅은 데이터가 있는 위치와 더 가까운 IoT 디바이스에 클라우드의 컴퓨팅 성능을 적용한다. 워크로드를 엣지로 이동하면 디바이스가 클라우드와 통신하는 데 걸리는 시간이 줄어들어 대기 시간이 단축될 뿐만 아니라 길어진 오프라인 기간에도 안정적으로 작동할 수 있다.

[표 2-7] 하이브리드 클라우드 장점(NIPA, 2020)

장점	설명
관리성(제어)	조직이 짧은 대기 시간이 필요한 중요한 자산이나 워크로드를 위한 프라이빗 인프라를 유지 관리
유연성	필요할 때 퍼블릭 클라우드에서 추가 리소스를 활용
비용 효율성	퍼블릭 클라우드로 확장 배포하는 기능을 통해 필요할 때만 추가되는 컴퓨팅 성능의 비용을 지급
용이성	여유 시간을 두고 점진적으로 워크로드를 단계적으로 수행하여 점차 전환할 수 있으므로 이용 부담 없이 클라우드로 전환

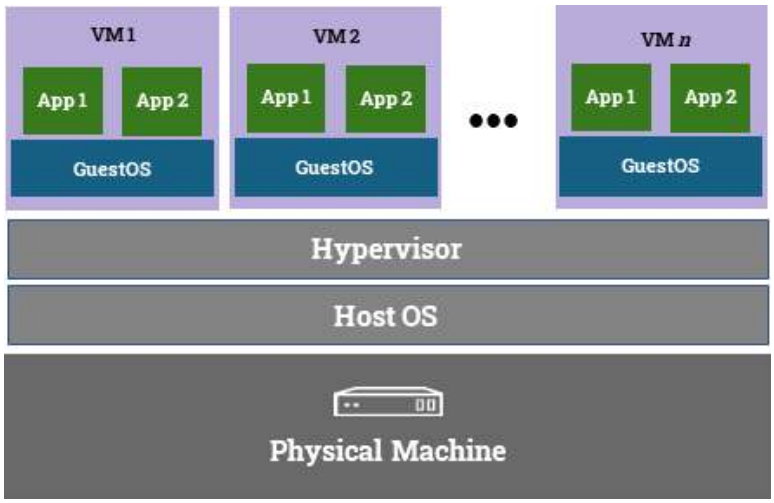
퍼블릭 클라우드와 프라이빗 클라우드에서 제공하는 장점을 [표 2-7]에서 언급한 관리성, 유연성, 비용 효율성 등을 가지고 있으며 단계적으로 업무를 클라우드로 전환할 수 있는 용이성을 제공한다.

2.4 클라우드 컴퓨팅 기술의 유형

클라우드 컴퓨팅은 인프라 환경을 서비스하기 위해 발전된 기술이 대표적이다. 본 장에서는 클라우드 컴퓨팅의 대표적인 기술을 정의하고 본 연구에서 조명하고자 하는 PaaS 플랫폼의 핵심적인 기술을 살펴보고자 한다.

2.4.1 가상화(Virtualization)

가상화 기술은 물리적인 시스템 환경을 논리적으로 나누어 자원을 효율적으로 분배하여 사용하기 위한 기술로 클라우드 컴퓨팅의 핵심 기술 중 하나이다.



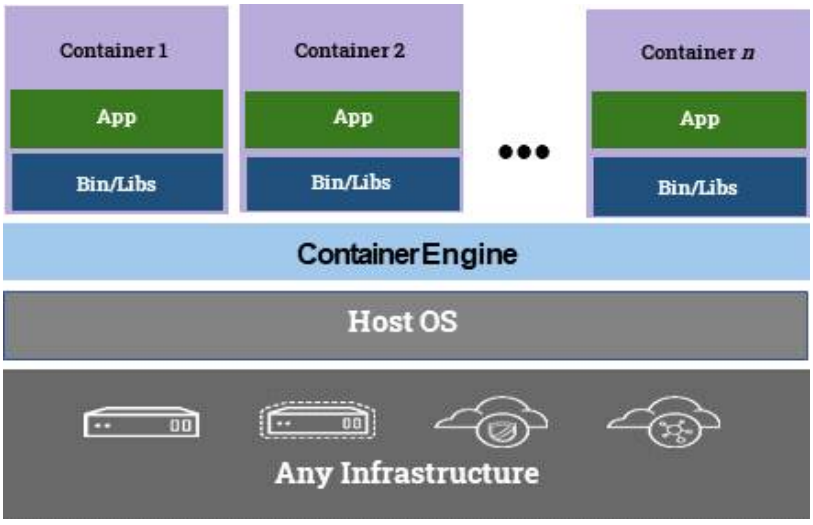
[그림 2-6] 가상화 개념도(Red Hat, 2019)

컴퓨터에서 CPU, 메모리, 디스크와 같은 자원을 추상화하는 개념이다. "물리적인 컴퓨팅 시스템 자원의 특징을 다른 시스템이나 애플리케이션, 최종 사용자들이 컴퓨팅 자원을 제어 및 관리가 필요 없도록 서비스로 제공하는(감추는) 기술"을 말한다. 이는 여러 개의 논리 자원으로서 역할을 하는 것처럼 보이는 서버, OS, 애플리케이션이나 저장 장치와 같은 부분을 마치 하나의 자원으로 사용하듯 만들어준다. 다르게는 단일 논리 자원 같은 서버나 저장 장치 등 여러 개의 물리적 자원을 만들어 낼 수 있다.

가상화 기술은 가상화 대상과 기술의 방식에 따라 다양하게 분류할 수 있으며, 가상화 환경을 위한 대상을 기준으로 서버 가상화, 데스크톱 가상화, 애플리케이션 가상화 등으로 분류할 수 있다.

2.4.2 컨테이너(Container)

컨테이너 기술은 2000년부터 'FreeBSD Jail', 'Unix FreeBSD', '리눅스 LXC'의 하위 시스템 분할의 아이디어로 시작되었으며, 예전부터 사용되어 온 기술이다. 리눅스 컨테이너는 하나의 서버 운영체제 위에 여러 개의 격리된 시스템 환경을 구현할 수 있는 운영체제 수준의 가상화 기술이다. 리눅스 컨테이너는 컨트롤그룹(cgroup)과 네임스페이스(namespace) 기능을 통해 운영 환경을 격리시키고 관리한다.



[그림 2-7] 컨테이너 개념도(Red Hat, 2019)

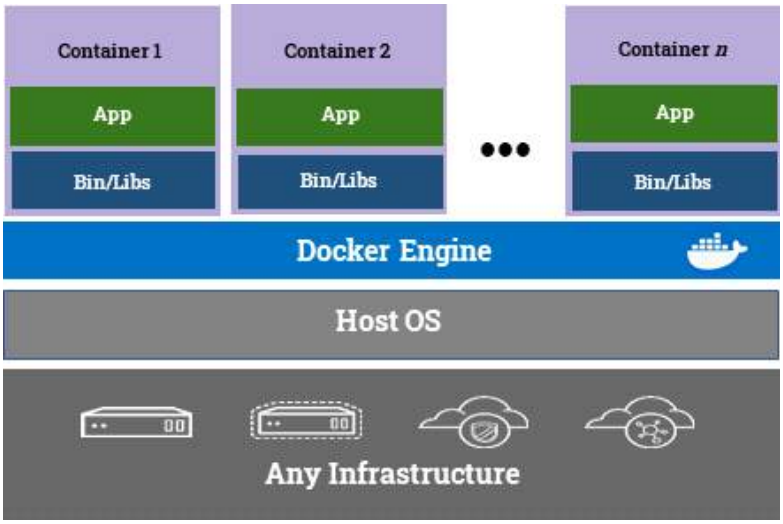
컨트롤그룹은 물리적인 하드웨어의 리소스를 프로세스 그룹 단위로 제어하는 기능이다. 컨트롤그룹을 통해 사용자는 CPU 시간, 시스템 메모리, 네트워크 대역폭과 같은 자원이나 이러한 자원의 조합을 실행 중인 프로세스 간에 할당하는 것이 가능하다. 이러한 사용 환경은 시스템 관리자가 시스템의 자원할당, 우선하는 순위 지정, 거부, 관리, 모니터링과 같은 제어를 가능하게 한다.

[그림 2-7]에서는 컨테이너가 실행되는 인프라 환경을 표현하고 있으며 가상화 개념과 다르게 Host OS에서 컨테이너 엔진이 실행되고 애플리케이션이 실행되는 구조이다.

2.4.2.1 컨테이너 엔진(Container Engine)

1) 도커(DOCKER)

도커는 애플리케이션을 개발, 배포 및 실행하기 위한 개방형 플랫폼을 말한다. 도커를 사용하면 애플리케이션을 인프라에서 분리할 수 있으므로 소프트웨어를 빠르게 제공할 수 있다. 도커를 사용하면 애플리케이션을 관리하는 것과 동일한 방식으로 인프라를 관리할 수 있다. 코드를 빠르게 전달, 테스트하고 배포하기 위해 도커 방법론을 활용하여 코드를 작성하고 프로덕션 환경에서 실행하면 그 사이에서 발생하는 지연을 크게 줄일 수 있다.



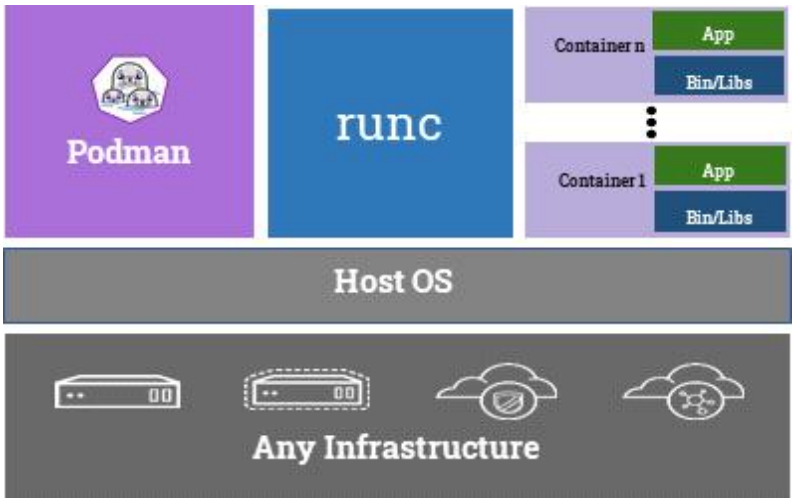
[그림 2-8] 도커 구조(Docker, 2019)

컨테이너가 실행되는 환경과 비교하면 [그림 2-8]과 같이 컨테이너 엔진이 도커 엔진으로 실행되는 부분이 다르다.

2) 포드맨(PODMAN)

포드맨(POD 관리자)은 Linux 시스템에서 OCI 컨테이너를 개발, 관리 및

실행하기 위한 daemon이 없는 컨테이너 엔진이다. 포드맨은 저장소에도 포함된 컨테이너 수명 주기 관리용 라이브러리인 libpod를 기반으로 하며, libpod 라이브러리는 컨테이너, 포드, 컨테이너 이미지 및 볼륨을 관리하기 위한 API를 제공한다.[3] 도커 컨테이너와 다르게 포드맨과 컨테이너가 동등한 위치에서 실행되는 형태로 아래 [그림 2-9]와 같은 구조로 표현할 수 있다.



[그림 2-9] 포드맨 구조(Red Hat, 2020)

포드맨은 컨테이너 엔진 역할을 하지만 계층적 구조로 보았을 때 컨테이너와 동일 선상에서 서비스하는 형태의 구조로 포드맨 엔진 프로세스가 갑자기 응답이 없는 상태가 되면 다른 프로세스가 역할을 대신하는 구조로 동작한다.

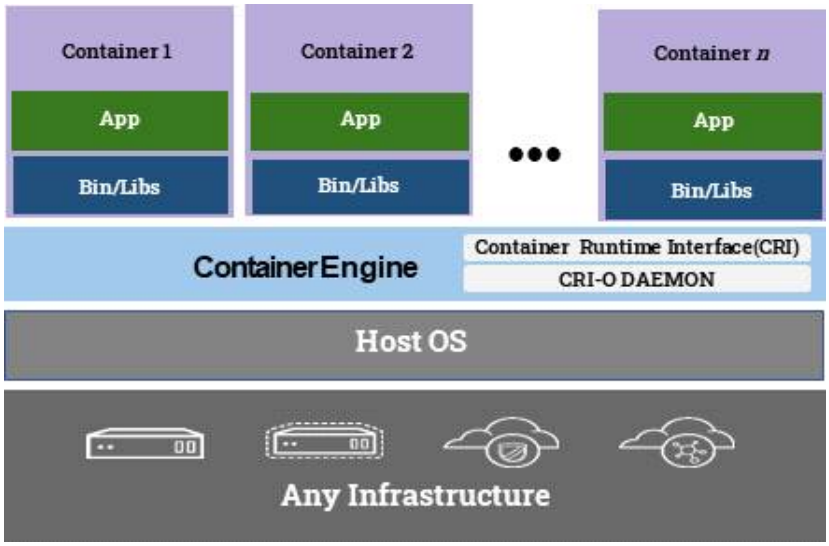
3) CRI-O

CRI-O는 OCI(Open Container Initiative)[4] 호환 런타임을 사용할 수 있

3) containers/podman https://github.com/containers/podman accessed 2021-12-01

4) 2015년 6월 Docker 및 기타 컨테이너 업계 리더에 의해 설립되었으며, 컨테이너 형식 및 런타임에 대한 개방형 산업 표준을 만들기 위한 명시적인 목적을 위한 개방형 거버넌스 구조이다.

도록 하는 Kubernetes CRI(Container Runtime Interface)의 완성이다. Kubernetes의 런타임으로 도커를 사용하는 것보다 가벼운 대안이며, 이를 통해 Kubernetes는 POD를 실행하기 위한 컨테이너 런타임으로 모든 OCI 호환 런타임을 사용할 수 있다.[5)]



[그림 2-10] CRI-O 구조(CRI-O, 2020)

CRI-O는 [그림 2-10]에서 보이는 컨테이너 런타임 인터페이스를 통해 표준성을 제공하고 이러한 표준 구성은 컨테이너 오케스트레이션을 제공하는 쿠버네티스와 쉽게 연계할 수 있다.

2.4.3 컨테이너 오케스트레이션(Container Orchestration)

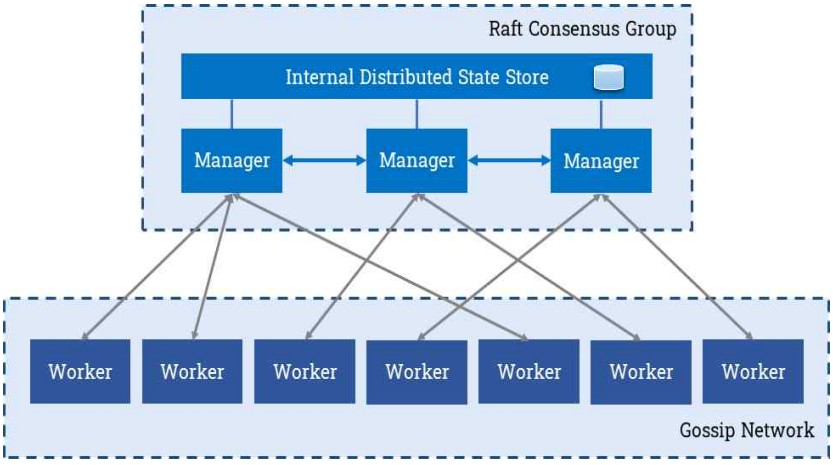
컨테이너 오케스트레이션은 컨테이너의 배포, 관리를 담당하는 것으로 여러 컨테이너의 배포 프로세스를 최적화하고 컨테이너의 자동배치 및 복제, 로드밸런싱 등의 다양한 기능을 담당한다. 컨테이너 오케스트레이션은 그 종류에 따라 차이가 존재하지만, 대표적으로 프로비저닝, 스크립트 작성, 모니터링, 업그레이드 및 롤백, 서비스 탐색 등의 기능을 담당하고 있다(최원석, 정

5) CRI-O 홈페이지 https://cri-o.io/ accessed 2021-12-01

혜진, 나연묵, 2019).

2.4.3.1 도커 스웜(Docker Swarm)

도커 스웜은 오픈 소스 컨테이너 오케스트레이션 플랫폼이며 도커를 위한 기본 클러스터링 엔진이다. 기존의 도커 모델은 자체 관리기능을 스웜 모드라고 부르며 컨테이너 클러스터링, 스케줄링 및 애플리케이션 관리를 제공했다. 도커 스웜은 아래 [그림 2-11]에서 보여주는 아키텍처와 같이 단일 노드 도커를 기본으로 클러스터 개념에서는 스웜으로 확장하는 개념을 사용하고 있다. 도커 스웜의 장점으로는 도커 호스트 클러스터를 조정하기에 편리하며 도커 명령어와 Compose를 그대로 사용할 수 있고 표준 도커 API를 제공하기 때문에 기존의 외부 API들을 그대로 사용할 수 있다.



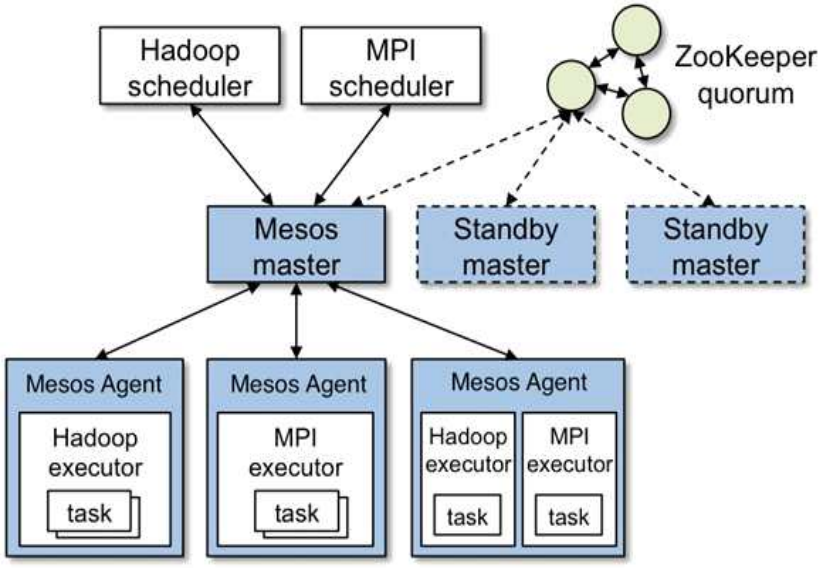
[그림 2-11] 도커 스웜 아키텍처(Docker, 2020)

도커 스웜은 클러스터 그룹 내에서 CPU, 메모리 등 컴퓨팅을 위한 리소스 사용량을 제한하고 특정 호스트에 컨테이너를 배포하거나 컨테이너 또는 호스트에 장애 발생 시 자동 복구 등의 기능을 제공한다(Nitin Naik, 2016). 그렇지만 기본적으로 도커가 단일 컨테이너 관리에 쉽게 만들어졌고, 도커 스웜은 이를 효율적으로 지원하는 컨테이너 오케스트레이션이므로 다양한 환경

의 컨테이너와 애플리케이션의 실행 요구사항이 증가할수록 컴퓨팅 리소스 관리와 로드밸런싱 등 오케스트레이션이 복잡해져 관리의 어려움이 증가할 수 있다.

2.4.3.2 아파치 메소스(Apache Mesos)

아파치 메소스는 UC Berkeley에서 Nexus라는 이름으로 개발이 진행되던 프로젝트이며 아파치 프로젝트에 오픈소스 그룹으로 포함되면서 메소스란 이름으로 변경되었다. 클라우드 기반의 컴퓨팅 자원을 통합 및 관리할 수 있도록 만든 리소스 관리 오케스트레이션이며 Master, Slave, Framework 3단계로 구성 후 Slave 노드에서 Framework를 통해 작업을 실행한다. 아래 [그림 2-12]에서 보는 바와 같이 메소스 구조에서 각각 메소스 Master와 장애 발생 시 이중화 구성을 위한 Standby Master, 자동 복구를 위한 ZooKeeper Quorum, 메소스 Slave로 구성되어 있다(Nitin Naik, 2016).



[그림 2-12] 아파치 메소스 아키텍처(Apache, 2017)

메소스는 기본적으로 컨테이너 오케스트레이션이 아닌 데이터 센터 자원의 공유 및 격리 기능을 관리하는 기술이다(Kakadia, D. 2015).

사용자가 서버 클러스터나 데이터 센터를 하나의 머신처럼 사용할 수 있게 기능을 제공하고 사용이 가능한 자원을 실시간 감시하여 필요조건에 따라 격리 및 할당하는 임무를 수행한다. 또한 각각의 프레임워크에서 자원을 할당하고 격리할 수 있게 기능을 제공하고, 자원의 최적화를 실현할 수 있으며 메소스는 대규모 클러스터 환경에 최적화되고 자원의 활용도를 높이며 시스템 운용 과정을 단순화시킬 수 있다. 그리고 메소스에서 자체적으로 복구 기능도 사용할 수 있으므로 노드의 실행 동작이 멈추더라도 해당 작업을 다른 노드로 전환하여 작업의 연속성을 보장할 수 있으며 외부 클라우드와의 연동도 가능하고 메소스를 위한 컨테이너 오케스트레이션 프레임워크로 다양한 플랫폼을 구성 가능하며 대표적인 것으로 마라톤(Marathon)이 존재한다. 외부 스케줄러나 API를 접목하여 사용할 수 있으며 사용자의 기존 애플리케이션 간의 관리 통합 기능을 제공하고 이를 통해 고가용성, 서비스 탐색, 부하 분산 등을 포함한 여러 가지 기능들을 제공한다. 그러나 메소스의 경우 특정 기능들을 사용하기 위해서는 외부 API나 Framework 등을 지속해서 결합하여 사용이 필요하며 이는 추상적인 계층을 추가해야 함을 의미한다.

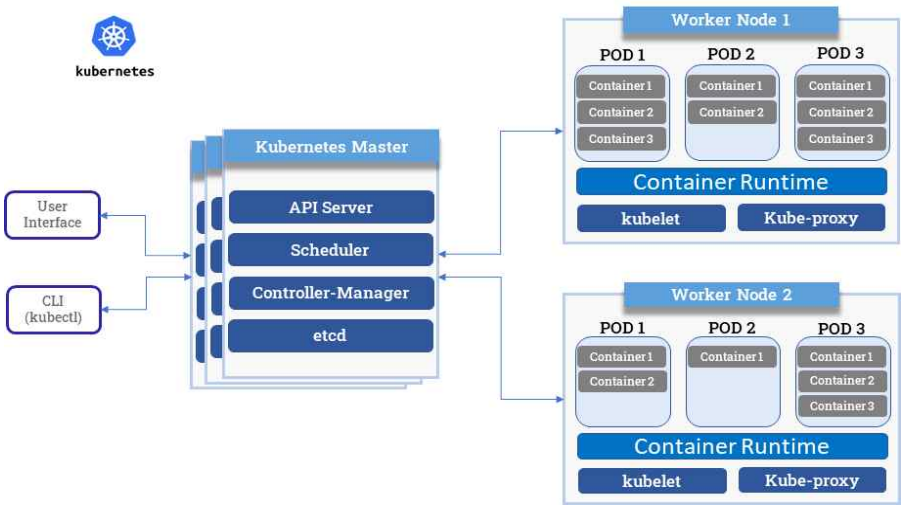
2.4.3.3 쿠버네티스(Kubernetes)

K8s라고도 알려진 쿠버네티스는 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리해주는 오픈소스 시스템이다.[6] 쿠버네티스는 표준 구성과 자동화 구성을 모두 쉽게 하며, 오픈소스 프로젝트와 함께 빠르게 성장하는 생태계를 확보해 가고 있으며, 쿠버네티스의 서비스 기술 지원과 지원 도구는 어디서나 쉽게 이용할 수 있다. 쿠버네티스는 Google에서 애플리케이션 서비스를 제공하기 위해 개발했던 Borg를 CNCF[7]에 기부하면서 오픈소스 프로젝트로 공개가 되었고 이름을 쿠버네티스로 변경하였다.

---

6) 쿠버네티스 오픈소스 프로젝트 페이지 설명 인용 https://kubernetes.io/ko/ accessed 2021-12-1

7) CNCF(Cloud Native Computing Foundation)는 컨테이너 기술을 발전시키고 기술 산업이 발전할 수 있도록 지원하기 위해 2015년에 설립된 Linux Foundation 프로젝트이다.



[그림 2-13] 쿠버네티스 아키텍처(Kubernetes, 2017)

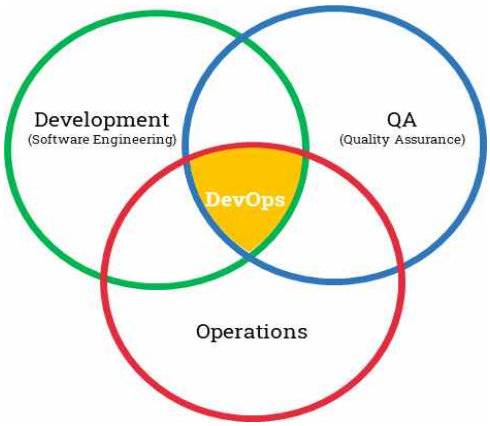
클라우드 인프라의 환경은 보안 구성 정책 및 지원 서비스와 비용에 의해 결정되지만, 어떠한 클라우드 인프라 환경에서도 애플리케이션은 특정 클라우드 인프라 환경에 종속되지 않고 실행될 수 있어야 하므로 클라우드 환경의 제약이 없고 서비스의 이동이 가능한 아키텍쳐를 가져야 한다. 이러한 기능을 실제로 가능하게 표준화를 제공하는 기술은 오픈소스 프로젝트로 진행되고 있는 쿠버네티스가 대표적이다. 쿠버네티스는 컨테이너 오케스트레이션 도구로 출발하였으며 현재는 클라우드 시스템의 전체 인프라를 관리하고 모니터링할 수 있는 서비스까지 발전되고 있다. 일례로 오픈스택 환경의 프라이빗 클라우드를 쿠버네티스 환경에서 운영하도록 구성할 수도 있고 베어메탈 환경에서 프로비저닝도 가능하다. 퍼블릭 클라우드 제공 업체인 아마존 웹 서비스, 마이크로소프트 애저, 구글 GCP 등 주요 글로벌 기업에서 쿠버네티스 기반의 서비스를 제공하고 있기 때문에, 인프라 환경이 달라지더라도 컨테이너 환경으로 개발된 애플리케이션을 쿠버네티스에서 운영이 가능하다.

2.5 DevOps 기술

DevOps는 소프트웨어의 개발(Development)과 운영(Operations)의 합성어

로서, 소프트웨어 개발자와 정보기술 전문가 간의 소통, 협업 및 통합을 강조하는 개발 환경이나 문화를 말한다.[8)]

DevOps는 위 [그림 2-14]와 같이 소프트웨어의 개발 시스템과 운영 시스템이 있는 조직 간의 상호 의존적 대응을 말하며 해당 조직에서 소프트웨어 출시를 위한 제품화와 서비스 제공을 위하여 최대한 빠르게 개발하고 신속하게 배포하는 것에 목적이 있다.



[그림 2-14] DevOps 개념도(위키피디아, 2020)

또한 DevOps는 클라우드 환경의 서비스 모델이 다변화되면서, 개발과 운영 측면에서 다양하게 요구되는 환경을 지원하기 위해 탄생한 개념이라고 할 수 있으며, 프로세스를 정립하고 자동화를 통해 새로운 제품 출시 기간을 단축, 신규 버전의 낮은 실패율, 빠른 복구 시간 등 운영 시스템의 예측 및 유지보수 가능성, 효율성 등을 추구한다.

2.5.1 DevOps 등장 배경

일반적인 소프트웨어 제품의 경우 연구 기획자에 의해 개발 계획을 수립하고 개발자가 제품을 설계 및 구현하며, 지속적 계획에 의해 업데이트가 이루어진다. 기능적인 심각한 오류나 버그(Bug) 및 보안 문제로 긴급 업데이트 패치가 진행되기도 하지만 대부분은, 사전 계획한 로드맵(roadmap)을 통한

8) 위키피디아 https://ko.wikipedia.org/wiki/데브옵스 accessed 2021-12-01

일정대로 업데이트를 진행한다. 하지만 운영 시스템 환경에 적용된 경우 제품 서비스를 시작하면 개발 시스템 환경처럼 계획된 업데이트를 통한 작업 계획을 수립하기 어렵다. 대부분은 서버, 네트워크, 보안 등 전문가 집단으로 구성된 운영조직은 시스템 운영을 하면서 운영 시스템 환경을 수정 및 개선하고 대내 또는 대외의 환경변화에 따른 요구사항이 즉시 또는 일상적으로 발생하기 때문이다.

무중단 운영 시스템은 점검해야 하는 자산(서버, 네트워크, 스토리지, 보안 등의 장비)이나 장애 요소가 증가하였을 경우 서비스 무중단 모니터링 기능의 추가가 필요하고 개별 자산의 특징이나 특성을 확인해서 적용해야 한다. 따라서 개발 환경의 긴급 시스템 변경(업데이트)과 달리 기존 시스템과 상호 유기적으로 연동되어 작동되는 운영환경에서는 소프트웨어의 품질을 충분히 검사하지 않고 시스템에 적용하는 경우가 많이 발생하는 것이 현실이다.

[표 2-8] 개발과 운영의 환경 차이(정근훈, 2020)

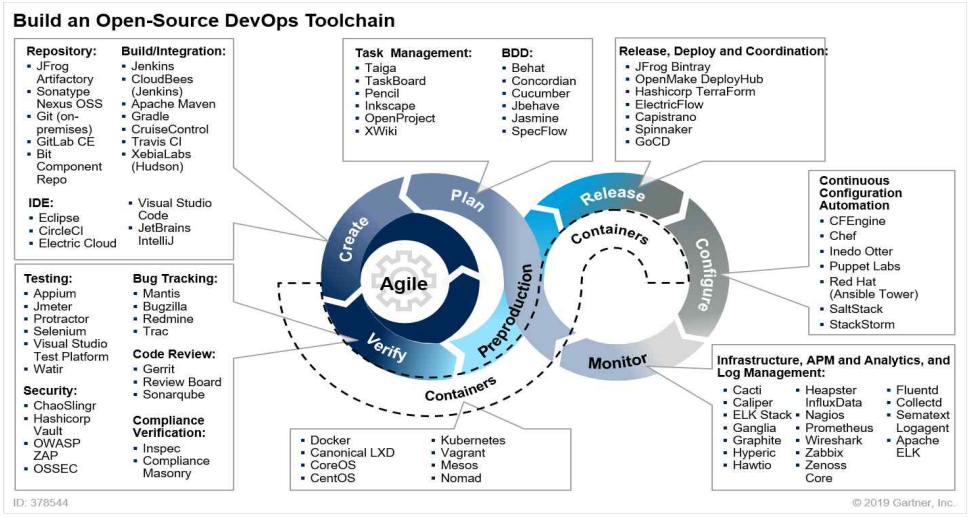
항목	개발	운영
목표	기능을 추가, 버전 출시, 배포	가용성, 안정성, 장애감소
성향	변화추구	안전유지
성과 척도	새로운 기능추가, 개발 기간	서버의 가동시간, 무중단 기간

개발과 운영은 제품의 라이프사이클에서 상호 보완적으로 연결이 되어야 하지만, 위 [표 2-8]과 같이 목표와 성향 그리고 성과에 대한 척도가 다르며 업무와 책임의 분리가 명확하다. 그리고 개발과 운영은 상호 협력하여 업무를 진행함의 어려움이 많은데 이러한 경우는 운영진이 전문성으로 특화된 조직에서 매우 두드러지게 나타난다. 구체적으로 클라우드 컴퓨팅 환경의 경우, 하드웨어에서 컨트롤 하던 영역이 소프트웨어 개발 요구사항이 되면서 개발과 운영에 더 많은 업무 협력이 필요하게 되었다. 대다수의 개발 오류나, 요구사항 재요청은 앞서 언급한 개발과 운영의 원활한 커뮤니케이션이 충족되

지 않는 부분에서 많이 발생하고 있다. 개발조직과 운영조직의 통합에 대한 가장 큰 장점은 개발과 운영의 원활한 의사소통과 협업으로 인해, 적시에 추가 가능을 제공하고, 운영자의 요구사항이 정확하게 반영되어 업무 효율성이 증가되는 선순환 구조로 인하여 넷플릭스, 페이스북, 아마존, 트위터, 구글 등 IT 선도 기업들이 폭넓게 채택하고 있다(정근훈, 2020). 서비스의 규모가 매우 크기 때문에 계속해서 개발이 반영되는 환경이거나, 실시간으로 모니터링하는 환경에서는 개발조직과 운영조직의 통합관리 필요성이 대두되면서 도입이 증가되고 있다(전인석, 2015).

2.5.2 DevOps 툴 체인

DevOps 툴 체인은 DevOps 방식을 사용하는 조직에서 조정하는 대로 시스템 개발 수명 주기 전반에 걸쳐 소프트웨어 애플리케이션의 제공, 개발 및 관리를 지원하는 도구 세트 또는 조합이다.



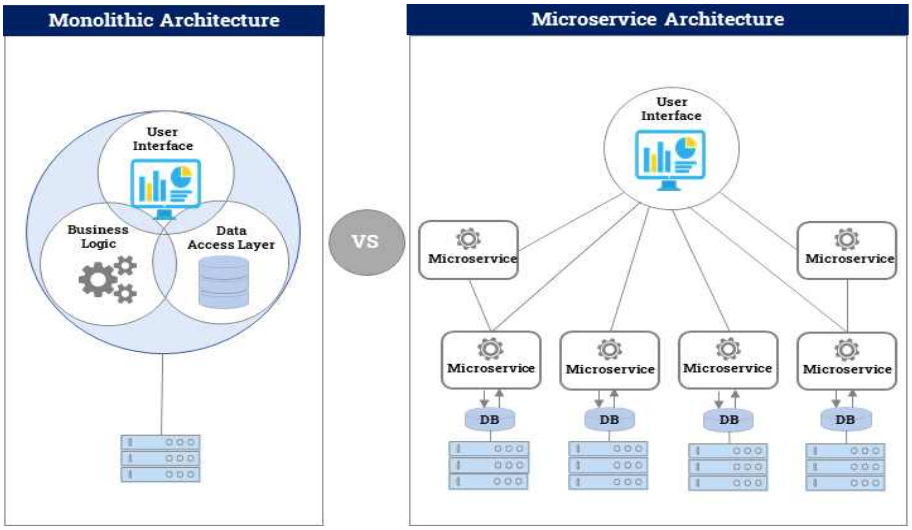
[그림 2-15] 오픈소스 DevOps 툴 체인(Gatner, 2019)

DevOps 구현은 위 [그림 2-15]에서 같이 오픈소스 소프트웨어를 통해 구현할 수 있는 다양한 툴을 포함하고 있으며, 특히 컨테이너 기술을 통해

클라우드 환경에서 지속적 통합(Continuous Integration)과 지속적 전달(Continuous Delivery)을 위한 단계별 플랫폼 서비스를 구현할 수 있다.

2.5.3 마이크로서비스 아키텍처

마이크로서비스 아키텍처는 독립적으로 분리된 소규모의 서비스 단위를 연결하여 전체 애플리케이션을 설계하는 아키텍처이다. 이러한 구조는 애플리케이션 측면, 애플리케이션 인프라스트럭처, 인프라스트럭처 패턴으로 구분할 수 있다(Richardson, C. 2017). 이렇게 서비스를 분리하는 단위를 비즈니스 기준으로 하거나, 테스트를 간소하게 독립적인 규모로 만들 수 있는 팀이 서비스 단위로 분리가 될 수 있고 소스에 대한 배포 단위는 호스트 당 다중 서비스 인스턴스, 호스트별 서비스 인스턴스, VM(Virtual Machine)별 인스턴스, 서버리스(Serverless) 배치 플랫폼, 서비스 배포 플랫폼 등으로 배포 단위를 다양하게 하도록 설계할 수 있다(윤경식, 김영한, 2019).



[그림 2-16] 모놀리식과 마이크로서비스 아키텍처 비교(Oracle, 2017)

전통적인 모놀리식 아키텍처와 마이크로서비스 아키텍처는 위의 [그림 2-16]과 같은 형태의 비교된 구조를 확인할 수 있으며, 가장 큰 차이점은 하

나의 마이크로서비스마다 독립된 업무 프로세스와 데이터베이스를 운영하며 서로 RESTful API 방식의 호출로 연결하는 구조이다(전인석, 2015, 정근훈, 박준석, 이극, 2020).

모놀리식 아키텍처는 한 대의 서버에서 모든 작업을 해야 하는 구조로 되어 있는 형태이고 마이크로서비스 아키텍처는 여러 서버에 분산되고 각각의 서비스를 연결하는 구조이다.

[표 2-9] 모놀리식과 마이크로서비스 비교(김정보, 김정인, 2020)

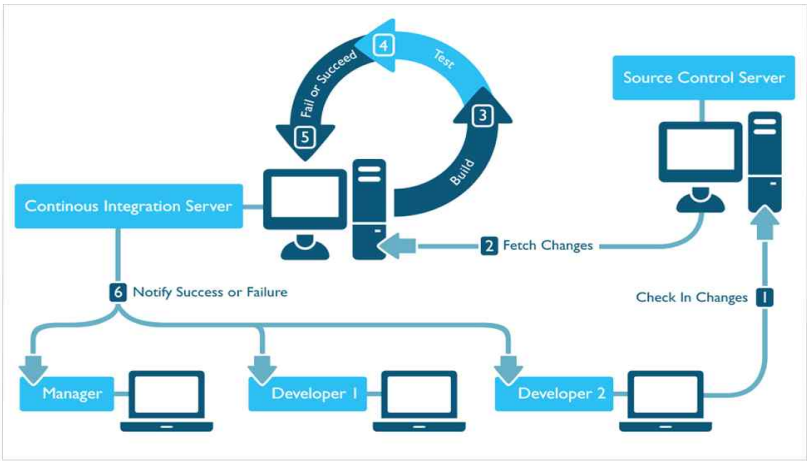
항목	Monolithic 방법	MSA 방법
개발언어	단일 언어 구성	여러 언어 조합 사용 가능
업데이트	프로그램 전체 업데이트로 인한 유연성 부족	서비스별 업데이트로 인한 유연성 향상
탄력성 (Elasticity)	전체 프로그램을 분기하여 확장	확장이 필요한 유일한 서비스 확장
리소스	단일 위치에서 리소스 관리	서비스별 위치 관리
다운 타임	업데이트 또는 문제 발생 시 전체 프로그램의 다운 타임	업데이트 또는 문제 발생 시 유일한 서비스 다운 타임
CI/CD	전체 프로그램에 대한 구축/테스트 배포 시간 증가	각 서비스에 대한 구축 및 테스트를 위한 배포 시간 단축
모듈형 디자인	의존성이 높으므로 응집력이 높으면 확장이 어려움	종속성이 낮아 각 서비스의 독립성을 확보하고 확장이 쉬움
컨테이너 기반 가상화	높은 요구사항(Specification)에 따른 중량 컨테이너 작업	서비스별 낮은 요구사항(Specification)의 컨테이너 구성으로 민첩한 컨테이너 운영 가능
유지보수	유지보수를 위해 전체 프로그램에 대한 영향도 분석이 필요	서비스별 수정/변경 유지보수 부담의 경감
콜라보레이션	단일팀이 있는 일반적인 사일로 팀 관리 비용 증가 및 개발 방법론 제한	여러 소규모 팀들이 팀 관리 비용을 절감하고 다양한 개발 방법론을 적용
품질	잦은 변경으로 인한 높은 영향으로 고품질 유지 어려움	서비스별 수정/변경에 대한 품질 보증

위의 [표 2-9]는 Cloud Native 환경의 장점을 활용하기 위해 PaaS와 IaaS Cloud 시스템을 활용하여 전통적인 개발 방법으로 개발했을 때와 MSA 개발 방법으로 개발했을 때의 효과를 비교한 자료이다(김정보, 김정인, 2020).

2.5.4 CI/CD 개념

전통적인 소프트웨어 개발 방법론은 오늘날의 비즈니스 요구사항을 충족하기에 충분하지 않으며, 일반적으로 빠르게 변화하고 예측할 수 없는 시장, 복잡하고 변화하는 고객 요구사항, 짧은 시장 시간(Time-To-Market) 및 빠르게 발전하는 정보기술은 대부분의 소프트웨어 개발 프로젝트에서 찾아볼 수 있는 특성이다. 애자일 방식을 적용하면 소프트웨어 개발 회사가 선호하는 SDLC(Software Development Life Cycle)의 유연성, 효율성 및 속도가 향상된다. 애자일 선언문에 따르면, 12가지 원칙은 Extreme Programming(XP), Scrum, Kanban, Crystal, Lean Software Development(LSD), FDD(Feature-Driven Development)와 같은 방법론에 적용되는 프로세스 및 관행의 무결성을 정의한다. 신속한 변화를 위한 CICD 파이프라인 구현으로 소프트웨어를 신속하게 제공하고 생산성을 높일 수 있다.

2000년에 Martin Fowler는 지속적 통합(Continuous Integration, CI)의 개념을 제시했고 나중에 J.Humble과 D.Farley는 이러한 아이디어를 배치 파이프라인의 개념으로서 지속적 전달(Continuous Delivery, CD) 접근 방식으로 확장했다. CI 관행의 주요 이점은 위험을 줄이고 소프트웨어 버그가 없고 안정적이게 만들어 빈번한 제공의 장벽을 제거한다는 것이다. 출시 시간 단축, 제품 품질 개선, 고객 만족도 향상, 신뢰할 수 있는 릴리스, 생산성 및 효율성 향상은 기업이 CD에 투자하도록 하는 주요 이점이다.



[그림 2-17] 일반적인 Continuous Integration 프로세스(Red Hat, 2020)[9)]

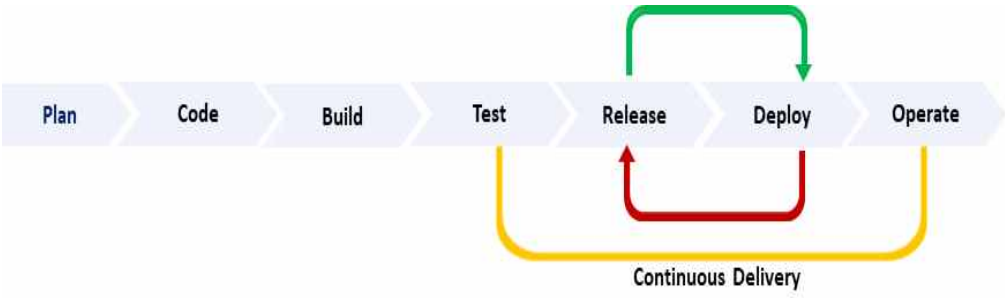
지속적 통합(continuous integration, CI)은 위 [그림 2-17]과 같이 개발하는 과정에서 연속적으로 품질 관리(Quality Control)를 적용하여 개발 프로세스를 실행하는 것이다.[10)] 작은 단위로 작업을 하거나 빈번하게 변경된 내용을 적용하게 되며 지속적인 통합은 모든 개발을 완료한 뒤에 품질 관리를 적용하는 고전적인 방법을 대체하여 소프트웨어의 품질 향상과 소프트웨어를 배포하는데 걸리는 시간을 줄이는 게 핵심이다.

지속적 배포(Continuous delivery, CD)는 팀이 짧은 주기로 소프트웨어를 개발하는 소프트웨어 공학적 접근의 하나로, 소프트웨어가 언제든지 신뢰 가능한 수준으로 출시될 수 있도록 보증하기 위한 것이다.[11)] 소프트웨어를 주기적으로 빌드 및 테스트하고 더 빨리 제품화하여 출시하는 것을 목표로 하며 이러한 접근은 다수의 증분되는 업데이트를 업무용 애플리케이션에 적용할 수 있도록 하여 변경사항에 따른 배포 비용과 시간, 위험을 줄이고 신속한 배포환경을 구성할 수 있게 해 준다.

9) https://pepgotesting.com/continuous-integration/ accessed 2021-12-01

10) 위키피디아 지속적 통합 내용 https://ko.wikipedia.org/wiki/지속적_통합 accessed 2021-12-01

11) 위키피디아 지속적 배포의 내용 https://ko.wikipedia.org/wiki/지속적_배포 accessed 2021-12-01



[그림 2-18] Continuous Delivery 프로세스(위키피디아, 2020; 연구자 재구성)

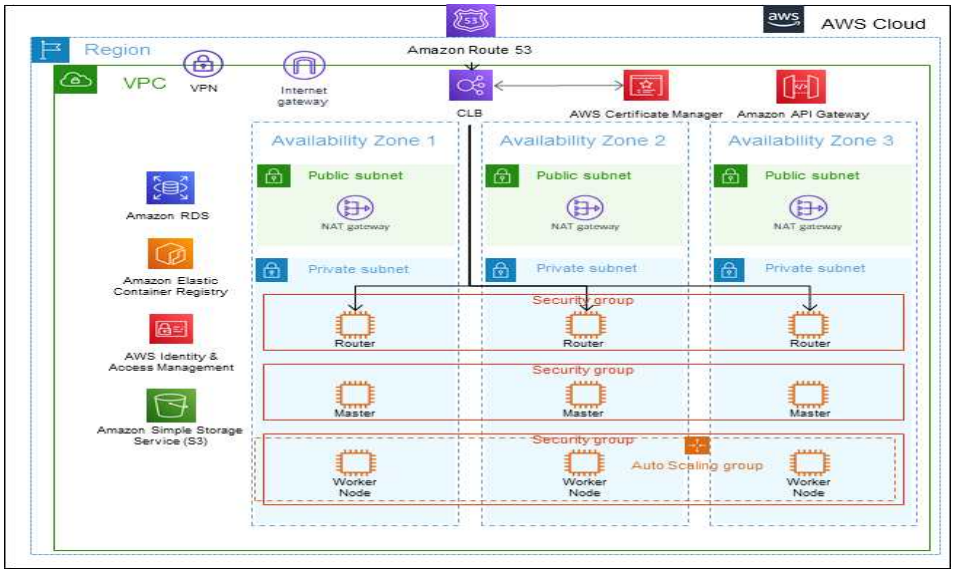
지속적인 전달(Delivery)은 위의 [그림 2-18]과 같은 형태의 프로세스로 진행되며, 제품 최종 배포까지 자동화 구현 여부에 따라 전달(Delivery) 또는 배치(Deployment)로 구분하여 정의하기도 한다.

III. PaaS 플랫폼 설계

3.1 PaaS 플랫폼 설계기준

3.1.1 퍼블릭 클라우드(Public Cloud) 설계

퍼블릭 클라우드를 서비스하는 대표적인 세계적 기업으로 아마존 웹 서비스(AWS), 마이크로소프트 애저(Azure), 구글(Google) GCP 등이 있고 국내의 경우, 네이버 클라우드 플랫폼, KT, SK 등이 있는데 그중 점유율[12]이 가장 높은 AWS를 기준으로 퍼블릭 클라우드에서 PaaS 플랫폼 설계를 구현하면 아래 [그림 3-1]과 같이 Availability Zone을 나누어 분산한다.



[그림 3-1] 퍼블릭 클라우드 계통도(AWS, 2020)

Availability Zone을 3개로 구성하여 Master 노드의 장애가 발생하였을

12) 2020년 4분기 기준으로 AWS 32%, MS 애저 20%, 구글 9% 순이다(시너지리서치그룹, 연합뉴스 기사 https://www.yna.co.kr/view/GYH20210205000400044, accessed 2021-11-29).

경우 서비스 장애로 이어지지 않도록 예방할 수 있는 구성이 필요하며, 다음과 같은 기본 구성요소가 배포된다.

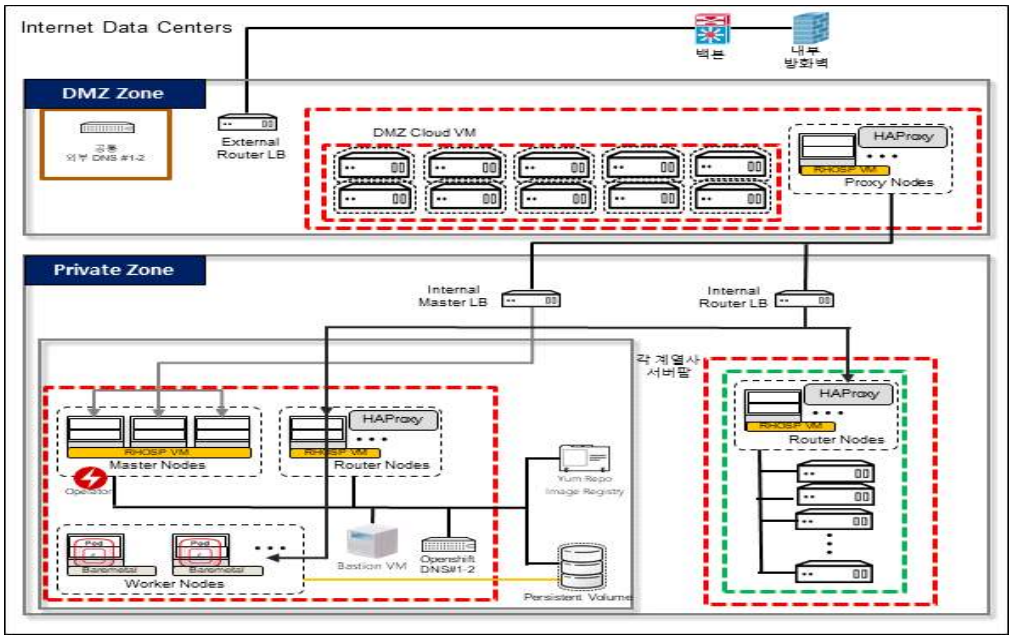
- Control Plane은 다음과 같은 마스터 구성요소를 제공한다.
  - API 서버(노드, 사용자, 관리자 및 오픈시프트에 배포된 기타 인프라 시스템을 포함하여 클라이언트의 요청 처리를 담당)
  - 컨트롤러 관리자 서버(스케줄러 및 복제 컨트롤러 포함)
  - 오픈시프트 클라이언트 도구(oc 및 oadm)
- etcd는 영구 마스터 상태를 저장하는 반면 다른 구성요소는 etcd가 원하는 상태로 전환되는지 확인한다.
- 노드는 컨테이너에 대한 런타임 환경을 제공한다.

3.1.2 프라이빗 클라우드(Private Cloud) 설계

프라이빗 클라우드는 기존 Legacy 환경과의 연계 측면에서 고려가 필요하고 PaaS 플랫폼 구성요소와 네트워크 요구사항에 맞춰 구현해야 한다. 외부와의 연결이 필요한 영역은 DMZ 구간으로 구성하여 방화벽 환경에서 보호하고 내부 네트워크 환경은 Private 구간으로 설계하였다.

본 연구에서 제안 설계한 아키텍처 구성은 물리적인 서버 환경이 많이 요구되는 관리 목적의 서버인 Master 노드, 인프라(Infra) 노드, Router 노드 등은 가상 서버(Virtual Machine)에서 구현하고 실제 컨테이너 실행을 위한 서버는 물리 서버(Physical Machine)로 설계하였다. 관리에 필요한 영역은 가상화 환경으로 다양한 인프라를 구현할 수 있는데 대표적으로 오픈스택을 통한 IaaS 서비스로 제공하면 PaaS 플랫폼과 함께 서버, 네트워크, 스토리지 등의 인프라 영역도 클라우드 서비스를 구현하여 제공할 수 있다. PaaS 플랫폼의 관리를 위해 생각보다 많은 자원이 요구되므로 하드웨어 구성을 안정적으로 구성하면서 자원 사용량을 줄이고자 한다면 가상화 기술이 포함된 아키텍

처를 선택하는 것이 좋다.



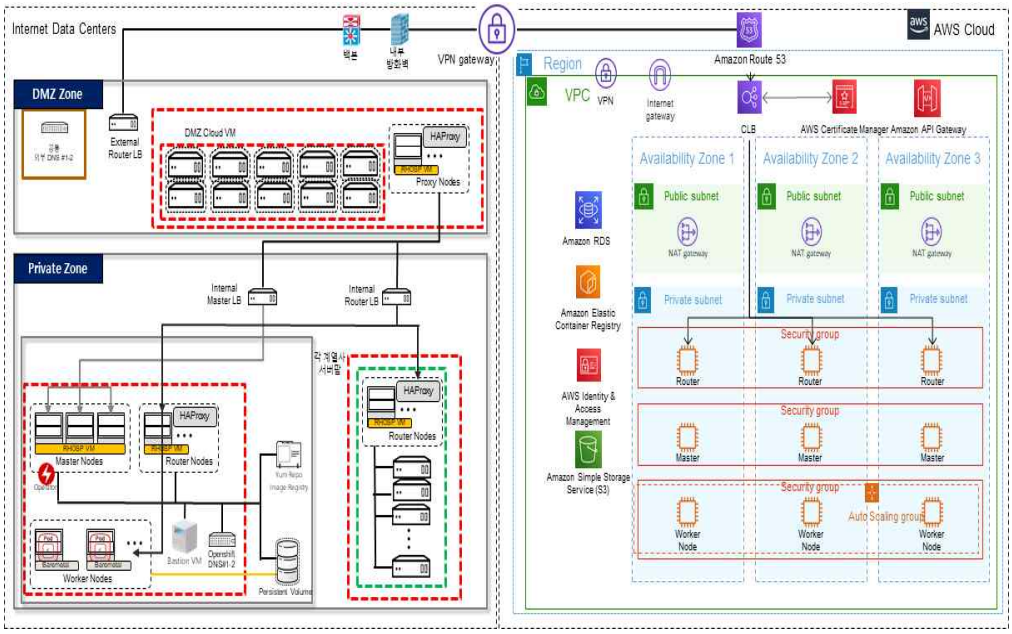
[그림 3-2] 프라이빗 클라우드 계통도 (Red Hat, 2020)

위 [그림 3-2]에서는 프라이빗 클라우드에서 PaaS 플랫폼을 구성할 때 네트워크의 분리와 인프라 자원의 분리를 구성하는 방안으로 설계하였다.

3.1.3 하이브리드 클라우드(Hybrid Cloud) 설계

하이브리드 클라우드는 퍼블릭 클라우드와 프라이빗 클라우드를 연계하여 사용하는 구성으로 퍼블릭 클라우드에서 서비스 중인 애플리케이션을 프라이빗 클라우드로 배포가 가능하거나 반대로 프라이빗 클라우드에서 서비스 중인 애플리케이션을 퍼블릭 클라우드로 배포할 수 있는 환경 구현이 중요하다. 최근에는 Cloud Management Portal(CMP)을 제공하는 형태로 하이브리드 클라우드 환경 요구사항이 증가하고 있는 추세이다. 본 연구에서는 하이브리드 클라우드 환경을 CMP 관점 보다는 서비스 연계 관점으로 접근하여 설계

하는 것을 제시한다. 서비스 연계 측면에서 퍼블릭 클라우드와 프라이빗 클라우드 환경은 전용 네크워크 연결을 통해 보안성 및 네트워크 대역폭(Bandwidth)을 보장하는 구성으로 설계가 필요하다.



[그림 3-3] 하이브리드 클라우드 계통도(Red Hat, 2020)

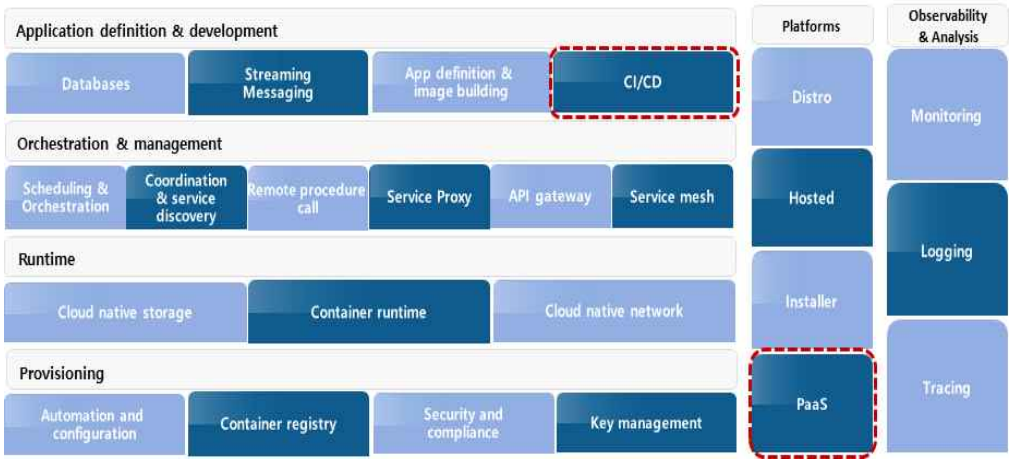
위 [그림 3-3]은 프라이빗 클라우드와 퍼블릭 클라우드를 Virtual Private Network(VPN) 전용선을 연결하여 구성하는 방안으로 네트워크 보안과 성능을 고려하여 설계가 되었다.

3.2 PaaS 플랫폼 환경 설계

3.2.1 오픈소스 기반

본 연구에서는 오픈소스 기반의 소프트웨어 기술을 통해 비즈니스적 가치를 수용하면서 기술적으로 안정화 된 PaaS 플랫폼 환경에서 CI/CD 오픈소스 기술을 적용하여 지속적 개발 및 배포환경을 구현하고자 한다. CNCF(Cloud

Native Computing Foundation)[13]에서 제공하는 Landscape를 기초로 하여 오픈소스를 선택하고 PaaS 플랫폼에서 구현 가능한 환경을 설계하였다. CNCF에서 제공하는 landscape는 아래 [그림 3-4]와 같이 관리되는 주요 그룹별로 분류된 오픈소스 및 상용 소프트웨어가 있으며, 점선 박스로 표시된 CI/CD와 PaaS 영역을 중심으로 소프트웨어를 선택하고 설계를 진행하였다.



[그림 3-4] CNCF Landscape(CNCF, 2021)

Kubernetes에서 인증된 플랫폼 중에서는 아래 [표 3-1]의 목록에서 오픈시프트를 선택하였으며, 해당 플랫폼에서 구현 가능한 CI/CD 파이프라인 도구를 선택하였다.

[표 3-1] Platform으로 Kubernetes 인증된 오픈소스 제품(CNCF, 2021)

구분	상세 설명
AgoraKube	최소 엔터프라이즈급 Kubernetes 클러스터를 부트스트랩 하도록 지원
Airship	클라우드 프로비저닝을 선언적으로 자동화하는 느슨하게 결합하였지만 상호 운용 가능한 오픈소스 도구 모음으로 Airship은 데이터 센터 인프라의 전체 수명 주기를 관리하여 OpenStack-Helm을 포함하여 Helm이 배포한 아티팩트로 프로덕션 등급 Kubernetes 클러스터를 제공

13) CNCFCloud Native Computing Foundation)는 비영리 단체인 리눅스 재단(Linux Foundation)의 일부 프로젝트로 세계 최고의 개발자, 최종 사용자 및 공급업체를 한자리에 모아 최대 규모의 오픈 소스 개발자 컨퍼런스를 운영한다.(https://www.cncf.io/)

Amazon Elastic Kubernetes Service Distro (Amazon EKS-D)	Amazon Elastic Kubernetes Service Distro(Amazon EKS-D)는 안정적이고 안전한 Kubernetes 클러스터를 생성하기 위해 Amazon Elastic Kubernetes Service(EKS)를 기반으로 하고 사용하는 Kubernetes 배포판
Azure (AKS) Engine	Azure(AKS) 엔진 - 커뮤니티가 공동 작업하고 Azure용 최고의 개방형 Docker 컨테이너 인프라를 구축
Canonical Charmed Distribution of Kubernetes	Ubuntu를 배포하는 사람들이 제공하는 최신 메트릭 및 모니터링을 통해 가장 광범위한 클라우드에서 테스트 된 순수한 Kubernetes 배포판
Desktop Kubernetes	단일 Bash 스크립트를 실행한 후 VirtualBox를 사용하여 데스크톱에서 3개의 VM Kubernetes 개발 클러스터를 설정
Elastisys Compliant Kubernetes	Compliant Kubernetes(ck8s)는 보안 및 규정 준수를 염두에 두고 구축된 엔터프라이즈급 Kubernetes 배포판
Flant Deckhouse	Deckhouse Platform을 사용하면 어디에서나 동종 Kubernetes 클러스터를 생성하고 완벽히 관리할 수 있음
Flexkube	유연한 Kubernetes 배포
Fury Distribution	Kubernetes Fury는 순전히 업스트림 Kubernetes를 기반으로 경쟁 테스트를 거친 배포판
k0s	k0s는 모든 호스트에 실행 파일을 복사하고 실행하기만 하면 Kubernetes 클러스터를 구축할 수 있도록 사전 구성된 모든 필수 항목이 포함된 단일 바이너리 포괄적인 Kubernetes 배포판
k3s	가벼운 kubernetes
KubeCube	KubeCube는 기업에 Kubernetes 리소스의 시각화된 관리 및 통합 멀티 클러스터 멀티 테넌트 관리 기능을 제공하는 오픈소스 엔터프라이즈급 컨테이너 플랫폼
Kubermatic Kubernetes Platform	모든 클라우드, 사내 및 에지(Edge) 상에서 수천 개의 Kubernetes 클러스터에 대한 Kubernetes 배포 및 운영 작업을 자동화

Kubesphere	Kubesphere.io는 KubeSphere 컨테이너 관리 플랫폼의 업스트림 프로젝트이며, 우리의 비전은 Kubernetes를 기반으로 하는 개인과 기업을 위한 보다 쉽고 친숙하며 강력한 분산 관리 플랫폼을 제공하고 더 많은 비즈니스 요구 사항을 충족하고 더 많은 사용자가 Kubernetes를 더 빠르고 더 잘 사용할 수 있도록 도움
Kubic	openSUSE Kubic은 openSUSE MicroOS라고 하는 컨테이너 최적화 트랜잭션 업데이트 기본 시스템 위에 최신 kubeadm 버전을 통합하는 커뮤니티 Kubernetes 배포판
kURL	Kubernetes URL(kURL)은 사용자 지정 Kubernetes 배포를 만들기 위한 프레임워크로 URL(curl 및 bash를 통해 설치) 또는 다운로드 가능한 패키지(airgapped 환경에 설치)로 공유함
MetalK8s	장기 온프레미스 배포에 중점을 둔 독창적인 Kubernetes 배포판
MicroK8s	설치가 간편한 단일 노드 로컬 Kubernetes
OpenNESS	OpenNESS는 고도로 최적화된 고성능 에지 플랫폼이 Kubernetes 클러스터를 통해 애플리케이션 및 네트워크 기능을 온보딩하고 관리할 수 있도록 하는 다중 액세스 에지 컴퓨팅 소프트웨어 도구 킷
Rancher Kubernetes	Rancher의 Kubernetes 배포판을 어디에나 배포하거나 Google, Amazon 또는 Microsoft에서 클라우드 Kubernetes 서비스를 시작
Red Hat OpenShift	기업에 docker 및 Kubernetes를 제공하는 컨테이너 애플리케이션 플랫폼
RKE Coverment	연방 정부 규정 준수 기반 사용 사례를 지원하는 데 중점을 둔 Kubernetes 배포판
Typhoon	타이푼은 업스트림 쿠버네티스, 아키텍처 컨벤션, 클러스터 애드온을 배포

vcluster	각 vcluster는 다른 Kubernetes 클러스터의 네임스페이스 내에서 실행되고 vcluster를 사용하는 것은 별도의 완전한 클러스터를 만드는 것보다 훨씬 저렴하며 일반 네임스페이스보다 더 나은 다중 테넌시 및 격리를 제공
VMware Tanzu Community Edition	학습자와 사용자를 위한 모든 기능을 갖춘 관리하기 쉬운 Kubernetes 플랫폼으로, 커뮤니티에서 지원하는 사용 제한 없이 자유롭게 이용

아래 [표 3-2]와 같이 오픈소스로 제공되는 CI/CD 솔루션 목록들을 조사하였으며, 기업에서 가장 많이 활용되면서 오픈시프트에서 지원되는 Jenkins를 선정하였다.

[표 3-2] 오픈소스 기준 CI/CD Landscape(CNCF, 2021)

구분	상세 설명
argo	Kubernetes를 위한 워크플로우 엔진
flux	Kubernetes를 위한 개방적이고 확장 가능한 지속적 전달 솔루션, GitOps 툴킷 제공
agola	고급 기능과 아키텍처를 갖춘 오픈 소스 CI/CD 플랫폼으로 강력하고 재현 가능하며 컨테이너화된 워크플로우(Runs라고 함)
Brigade	Kubernetes용 이벤트 기반 스크립팅
Buildkite	Buildkite 에이전트는 모든 장치 또는 네트워크에서 빌드 작업을 안전하게 실행하기 위해 Golang으로 작성된 오픈 소스 툴킷
Concourse	Concourse는 Go로 작성된 컨테이너 기반의 연속적인 행위자
Drone	컨테이너 네이티브 지속적 전달 플랫폼
flagger	점진적 전달 Kubernetes Operator(카나리, A/B 테스트 및 Blue/Green 배포)
Gitlab	Gitlab CE Mirror

GoCD	GoCD용 메인 리포지토리 - 지속적 전달 서버
Hyscale	K8s을 통한 앱 중심 추상화 프레임워크 및 멀티 클라우드 Kubernetes 배포를 가속화하기 위한 엔터프라이즈 플랫폼
Jenkins	Jenkins 자동화 서버
JenkinsX	Jenkins X는 Tekton의 Cloud Native 파이프라인을 사용하여 pull 요청에 대한 미리 보기 환경과 함께 Kubernetes용 자동화된 CI+CD를 제공
k6	k6은 성능 테스트를 생산적이고 즐거운 경험으로 만들기 위해 구축된 개발자 중심의 무료 오픈소스 부하 테스트 도구
Keptn	클라우드 네이티브 애플리케이션 라이프 사이클 오케스트레이션
OpenGitOps	OpenGitOps 프로젝트에 대한 최상위 정보를 위한 리포지토리
OpenKruise	Kubernetes에서 애플리케이션 관리 자동화
Ortelius	마이크로서비스의 구현을 단순화
Razee	클러스터, 환경 및 클라우드 제공자 전반에 걸쳐 Kubernetes 리소스의 배포를 자동화 및 관리하고 배포 프로세스를 모니터링하고 배포 문제를 찾을 수 있도록 리소스에 대한 배포 정보를 시각화
Screwdriver	지속적인 배포를 위해 설계된 오픈소스 빌드 플랫폼
Spinnaker	빠른 속도와 자신감으로 소프트웨어 변경 사항을 릴리스하기 위한 멀티 클라우드 지속적 배포 플랫폼
Tekton Pipelines	클라우드 네이티브 파이프라인 자원
Travis CI	Travis CI용 Ember 웹 클라이언트
Werf	CI/CD 파이프라인에서 애플리케이션 제공을 단순화하고 가속하도록 설계된 GitOps CLI 도구
Woodpecker CI	Dron CI 시스템의 커뮤니티 포크

CNCF Landscape 목록에서 확인된 오픈소스 제품을 기준으로 분류하여 가상화를 접목할 수 있는 기술 기반의 클라우드 환경과 CI/CD를 위한 애플리케이션을 선택하여 아래 [표 3-3]과 같이 스택 형태로 구성하였다.

[표 3-3] 오픈소스 소프트웨어 구성(락플레이스, 2021)

Layer	Software
CI/CD Application	Jenkins, Maven, Nexus, Gitlab, Sonarqube
Platform Environment	OpenShift(Cri-O, Kubernetes)

또한 오픈소스 기술 기반의 소프트웨어 구성으로 설계하여 Self-Support가 쉽고 범용성과 이식성이 탁월한 시스템 환경을 고려하여 설계하였다. Cloud Native 애플리케이션 실행 환경을 위해 가상 서버 환경은 리눅스 컨테이너를 사용하였으며, 개발 프로세스의 표준화를 위한 CI/CD 파이프라인은 Jenkins를 주축으로 VCS 버전 관리 시스템 체계로 구현하였다.

3.2.2 컨테이너 오케스트레이션

3.2.2.1 컨테이너 오케스트레이션 요구사항

컨테이너 오케스트레이션은 컨테이너로 구성된 서비스들의 라이프사이클을 관리하고 소규모의 컨테이너를 배포하거나 유지하는 일은 어렵지 않지만 수백의 컨테이너와 서비스가 있는 경우 관리가 복잡해지고 어려워져 대규모로 운영하는 경우 컨테이너 배포, 확장, 컨테이너 가용성과 관리를 위한 컨테이너 오케스트레이션 기능은 필수적이다(정은태. 박규동, 최백규, 2019).

본 연구에서는 오픈소스 환경의 Kubernetes를 기준으로 컨테이너 오케스트레이션을 위해 필요한 기능을 [표 3-4]와 같이 정의하였다.

[표 3-4] 컨테이너 오케스트레이션 기능(Red Hat, 2021; 연구자 재구성)

기능	설명
프로비저닝	• 시스템 자원을 할당, 배치, 배포하여 필요시에 예약하고 시작할 수 있도록 준비 • 사용자의 요구사항에 따라 자원, 지리적 위치 등을 고려하여 최상의 장소에 설치
모니터링	• 컨테이너 관리에 주로 사용하는 감시 도구이며 컨테이너 다운 시 새로운 컨테이너로 교체 실행하고 호스트 서버 장애 때 다른 호스트 서버에서 컨테이너를 실행 • 시스템 검사와 컨테이너가 있는 가상머신이나 호스트 서버의 특이점도 기록
빌드 스크립트	• 컨테이너에서 사용될 특정 애플리케이션 구성 정보를 컨테이너에 올려 자동 설치 스크립트를 작성 • 이를 통해 구성된 이미지는 배포 및 빠른 컨테이너 생성을 지원
서비스 탐색	• 기존에는 애플리케이션에 필요한 서비스들을 직접 지정했으나 오케스트레이션을 통해 적합한 자원을 검색
오퍼레이터 허브	• 컨테이너 이미지를 표준화하여 쉽게 연계할 수 있는 허브형 이미지 저장소를 Public이나 Private 형태로 설정

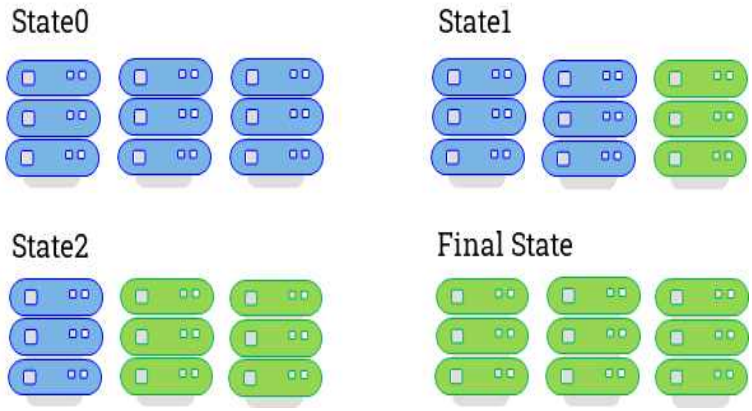
컨테이너 오케스트레이션은 Kubernetes 환경에서 요구되는 사항을 고려하여 시스템 자원 사용과 컨테이너 배포, 모니터링, 컨테이너 생성을 위한 빌드 스크립트와 표준화 된 컨테이너 이미지를 쉽게 연동할 수 있는 기능으로 정의하였다.

3.2.2.2 애플리케이션 배포 방식

사용자에게 제공하는 서비스는 지속해서, 기존 기능을 개선하거나 새로운 기능을 추가하게 되는데 기존의 서비스와 새로 업그레이드되는 서비스를 전환하는 방법 또한 중요한 사항이 되며, Rolling, Blue/Green, Canary 배포라는 방법이 있다(정근훈, 2020).

1) 롤링(Rolling) 배포

롤링 또는 단계 배포는 롤백 없이 배포 위험을 최소화하는 전략이며 롤링 배포는 응용 프로그램의 새로운 버전이 점진적으로 예전 버전을 대체하고 실제 배포는 정해진 기간 안에서 발생하며 이 기간 내에 새로운 버전과 예전 버전은 기능 부분이나 사용자에게 사용 환경적 부분에서 영향을 주지 않으면서 서로 공존하게 되는데 이 방법을 통해 기존 구성요소와 호환되지 않는 새로운 구성요소를 쉽게 되돌릴(rollback) 수 있다.



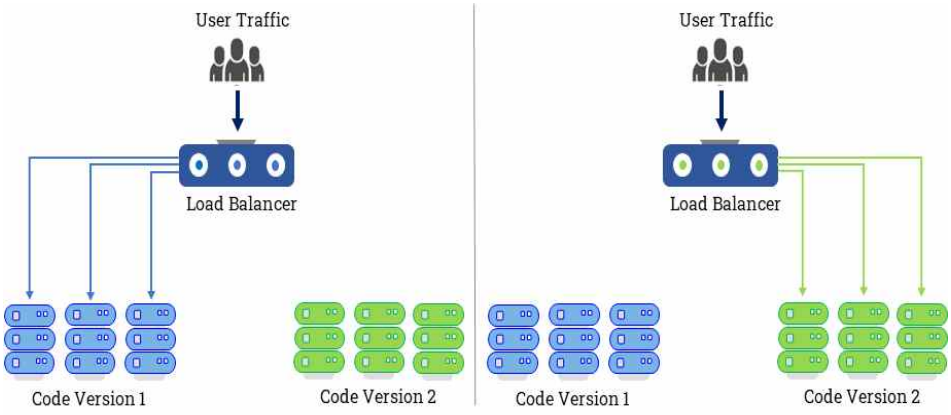
[그림 3-5] 롤링(Rolling) 배포 방식(Red Hat, 2019)

위 [그림 3-5]에서는 롤링 배포의 절차를 단계적으로 표현해 주고 있으며 최종적으로 Final State로 변경되면서 새로운 버전이 적용된다.

2) 블루(Blue)/그린(Green) 배포

블루/그린 배포는 이전 버전에서 신규 버전으로 일제히 전환하는 전략으로 이전 버전의 서버와 신규 버전의 서버를 동시에 구성하고 배포 시점이 되면 트래픽을 일제히 전환하면서 하나의 버전만 최종 제품으로 제공하므로 버전 관리 문제를 방지할 수 있고 빠른 롤백이 가능하다. 또 다른 장점으로 운영환경에 영향을 주지 않고 실제 서비스 환경으로 신규 버전

테스트를 진행할 수 있다. 예를 들어 이전 버전과 신규 버전을 구성하고 네크워크 포트를 다르게 설정하거나 내부 네트워크 트래픽인 경우 신규 버전으로 접근하도록 설정하여 테스트를 진행할 수 있으나 시스템 자원을 많이 요구(거의 두배)하므로 전체 플랫폼에 대한 테스트가 진행되어야 한다.

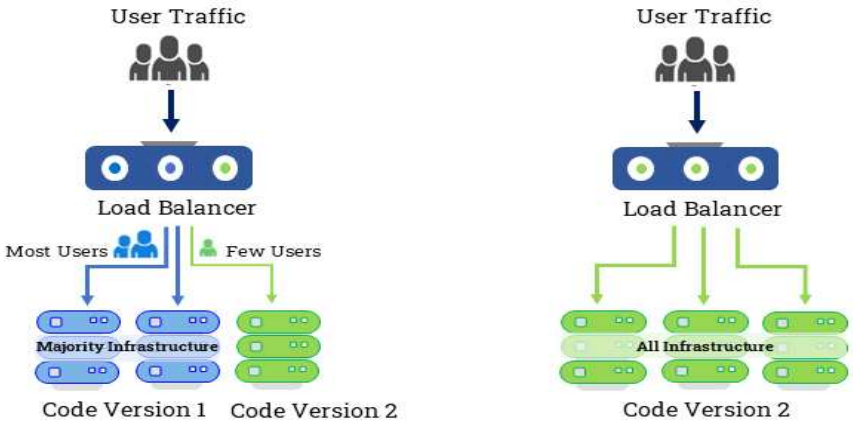


[그림 3-6] 블루(Blue)/그린(Green) 배포 방식(Red Hat, 2019)

블루/그린 배포 방식은 [그림 3-6]과 같이 블루에 연결된 Code Version을 Green에 연결된 Code Version으로 변경하는데 로드밸런서의 역할이 중요하다.

3) 카나리(Canary) 배포

Canary 배포는 이전 버전의 서버와 신규 버전의 서버들을 구성하고 일부 네트워크 트래픽을 새로운 버전으로 분산하여 오류 여부를 판단하는 배포 전략으로 이 방법은 오류 확률과 성능 모니터링을 유용하게 제공할 수 있고 네트워크 트래픽을 분산시킬 라우팅은 랜덤하게 설정할 수 있고 사용자 프로필 등을 베이스로 분류할 수도 있다.



[그림 3-7] 카나리(Canary) 배포 방식(Red Hat, 2019)

[그림 3-7]과 같이 분산 후 배포 결과에 따라 새로운 버전이 운영환경을 대체할 수도 있고, 이전으로 되돌릴 수도 있다.

3.2.3 아마존 웹 서비스(AWS)

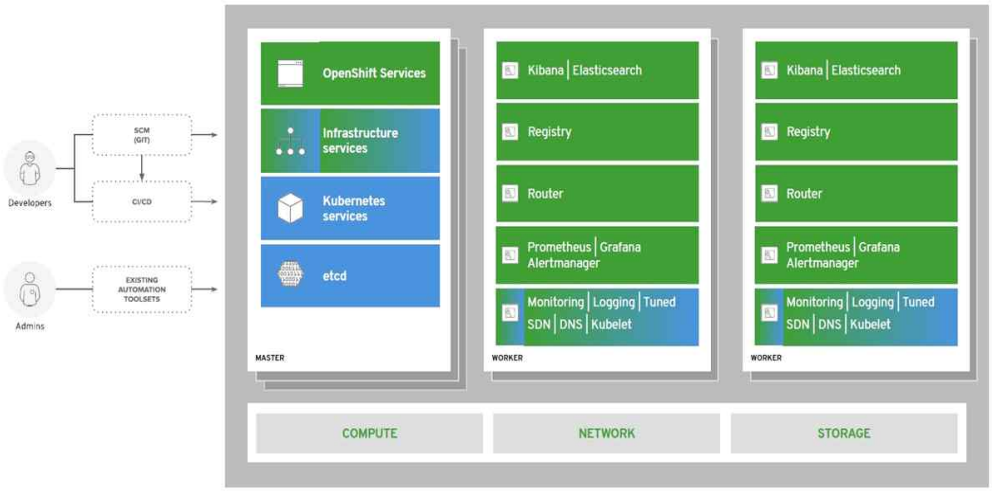
아마존 웹 서비스에서는 대표적인 컨테이너 서비스로 Amazone Elastic Container Service(ECS)와 쿠버네티스 기술을 사용한 Amazon Elastic Kubernetes Service(EKS)를 제공하고 있으며, 레드햇의 오픈시프트를 서비스로 제공하는 Red Hat OpenShift Service on AWS(ROSA)를 제공하고 있다. 본 연구에서는 직접 구축하고 관리하는 PaaS 시스템을 기준으로 오픈시프트를 사용하였다. 향후 직접 관리가 어려운 경우에는 ROSA 서비스를 이용하면 기존 환경에서 기술적 변경 없이 서비스를 이용할 수 있는 장점이 있다.

3.2.4 오픈시프트(OpenShift)

오픈시프트는 컨테이너 기반 소프트웨어의 디플로이 및 관리를 위한 레드햇의 컴퓨터 소프트웨어 제품이다. 구체적으로 말해, 가속화된 애플리케이션 개발을 위해 도커 컨테이너와 데브옵스 도구를 사용하는 쿠버네티스의 지원 배포판이라 할 수 있다.[14] 오픈시프트는 On-Premise와 클라우드 인프라 환

경을 모두 지원하여 클라우드 인프라 구축에 따른 제약사항을 최대한 배제할 수 있는 환경을 제공한다.

오픈시프트는 컨테이너 관리를 위한 Master Node와 실제 컨테이너가 실행되는 Worker Node로 구성된다. Master Node는 etcd라는 키값 방식의 스토리지 기능으로 3중화된 클러스터 환경을 제공하여 서버나 네트워크의 장애가 발생하더라도 관리적 문제가 발생하지 않도록 서비스 연속성을 보장한다. 하드웨어를 지원하는 OS의 경우, 컨테이너 기술을 지원하는 CoreOS를 사용하여 시스템의 관리와 성능을 보장하고 도커 컨테이너로 만들어진 이미지를 포드맨을 통해 이식성이 용이한 호환성을 제공한다.



[그림 3-8] 오프시프트 상세 구성도(Red Hat, 2019)

[그림 3-8]은 오픈시프트에 포함된 상세 소프트웨어 기능을 보여주고 있으며 쿠버네티스와 CRI-O 런타임을 통해 컨테이너 오케스트레이션과 Registry 서버를 통한 컨테이너 버전 관리, Router를 이중화 가능한 구조로 네트워크 통신의 신뢰성을 보장한다.

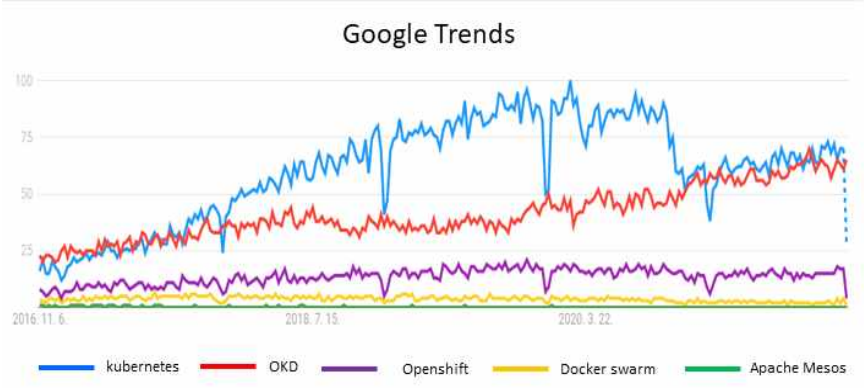
3.3 CI/CD 환경 설계

14) 위키피디아: https://ko.wikipedia.org/wiki/오픈시프트

3.3.1 컨테이너 시스템

오픈소스 프로젝트를 기반으로 많은 컨테이너 관리 기술이 나오고 있으며 클라우드 네이티브 애플리케이션 환경을 지원하는 시스템을 구축하기 위한 플랫폼 솔루션 선택은 매우 중요한 요소라 할 수 있다. 현재 컨테이너 시스템은 리눅스 기반으로 제공되는 플랫폼 형태가 주도적이며 앞서 설명한 Kubernetes 오케스트레이션과 함께 기술을 선도하고 있으며, 컨테이너 엔진의 경우 리눅스 OS에 최적화되고 호환성이 높아야 한다. 선행 연구에서는 대부분 연구가 도커 컨테이너 환경에서 진행되었으나 본 연구에서는 OCI(Open Container Initiative)에서 표준으로 제공하고 Kubernetes에서도 표준 컨테이너 런타임으로 제공되는 CRI-O를 기본으로 지원하는 오픈시프트를 통해 컨테이너 시스템을 설계하였다.

2015년 6월 도커 및 기타 컨테이너 업계 리더에 의해 설립된 OCI는 컨테이너 형식 및 런타임에 대한 개방형 산업 표준을 만들기 위해 명시적인 목적을 통한 개방형 거버넌스 구조를 지원하는 단체로 현재 런타임 사양(runtime-spec) 및 이미지 사양(image-spec)의 두 가지 사양이 포함되어 있다.(OCI, 2020)



[그림 3-9] 컨테이너 관리 시스템 사용 추이 (Google, 2021)

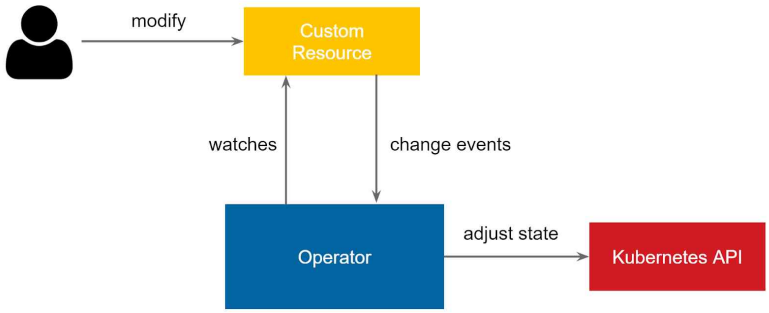
컨테이너 관리 시스템은 [그림 3-9]와 같이 쿠버네티스가 가장 많이 사용되고 있으며 오픈시프트는 쿠버네티스 호환 제품으로 기업에서 사용은 조금씩 증가하고 있다.

3.3.1.1 컨테이너 저장소

컨테이너는 이미지 형태의 속성을 가지고 있어서 저장소를 요구하며, 해당 저장소는 등록된 컨테이너 단위로 구분이 필요하므로 버전 관리를 해야 한다. 컨테이너 저장소로는 다양한 오픈소스 도구가 있는데 인터넷이 연결 가능한 환경에서는 Amazon Elastic Container Registry(Amazon ECR)와 Docker Hub 같은 레지스트리 저장소를 통해 쉽게 구현이 가능하다. 본 연구에서는 외부 네트워크와 단절된 폐쇄망 네트워크 환경의 제약사항으로 Docker Distribution을 사용하여 구현되도록 설계하였다.

3.3.1.2 컨테이너 오퍼레이터(Operator)

PaaS 플랫폼은 다양한 개발 도구와 서비스용 소프트웨어가 컨테이너로 제공되어야 한다. 베이스 컨테이너가 제공되지 않으면 개발자가 컨테이너 이미지를 처음부터 만들어야 하므로 상당한 시간과 노력이 요구될 수 있다. 그러므로 이미 검증한 베이스 컨테이너 이미지를 오퍼레이터 형태로 제공하면 해당 오퍼레이터에서 서비스 구현이 가능한 컨테이너로 쉽게 구현할 수 있다.



[그림 3-10] 쿠버네티스 오퍼레이터 동작 설명(Red Hat, 2021)[15]

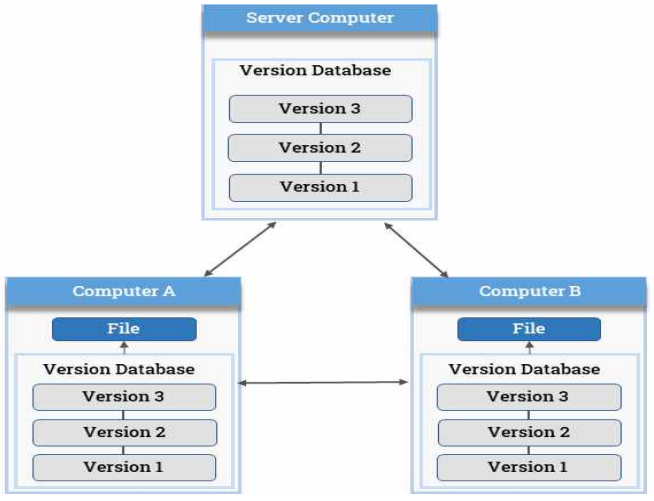
사용자가 애플리케이션 소스(Source)를 수정하면 위 [그림 3-10]과 같이 오퍼레이터는 변경된 이벤트를 감지하고 변경된 사항을 기반으로 쿠버네티스 API를 호출하는 형태로 작업이 수행된다.

3.3.2 CI(Continuous Integration)

3.3.2.1 분산 버전 관리 시스템

DVCS(Distributed Version Control System)는 클라이언트가 해당하는 파일의 마지막 스냅샷을 통해 저장하는 방식이 아닌 버전 관리 서버에 있는 저장소를 그대로 Local 저장소에 복사하여 사용하는 방식이다(이현경, 2020). 각 클라이언트도 서버에 있는 저장소와 자신만의 저장소를 동일하게 가지고 있으며, 아래 [그림 3-11]을 보면 버전 관리 서버에도 저장소를 가지고 있는 것을 확인할 수 있다.

분산 구조 방식은 버전 관리 서버에 특정 문제가 발생하더라도 각 클라이언트에 복사된 Local 저장소를 이용하여 즉시 복구가 가능하다는 장점을 포함하는 버전 관리 방식 중 하나이다.



[그림 3-11] 분산 버전 관리 시스템 구조(Red Hat, 2021)

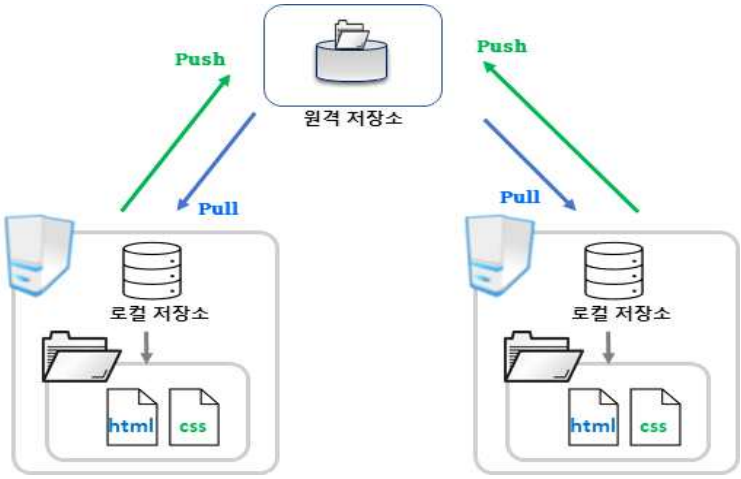
15) https://cloud.redhat.com/blog/build-your-kubernetes-operator-with-the-right-tool (Red hat 공식블로그)

소프트웨어는 개발한 소스 단위로 버전이 존재하고 이러한 버전을 관리하는 역할이 버전 관리 시스템의 역할이라고 할 수 있다. 또한 개발 소스가 저장되는 방식이 분산 구조로 동작하므로 백업 시스템과 소스를 하나로 통합해 주기 위한 기능적 측면이 고려되면 개발 시스템의 안정성도 향상되는 효과를 기대할 수 있다. 이러한 버전 관리 시스템 방식이 클라우드 환경에 적용되면 개발 환경의 편의성과 안정성, 그리고 생산성이 향상될 수 있다.

3.3.2.2 저장소

저장소의 의미는 Git에서 사용하는데, 저장소(Git Repository)라는 말 그대로 어떠한 파일이나 폴더를 저장해 두는 곳을 의미한다. Git에서는 원격 저장소, 로컬 저장소 이 두 종류의 저장소를 제공하고 있다.[16)]

- 원격 저장소(Remote Repository) : Git 서버는 원격 저장소에서 파일이 관리되며 다수의 사용자들이 파일을 공유하고 동시에 사용 가능한 저장소
- 로컬 저장소(Local Repository) : 사용자의 컴퓨터에 파일이 저장되므로 개인별 저장



[그림 3-12] Git Repository 구조(Nulab, 2019)

16) backlog by nulab: https://backlog.com/git-tutorial/kr/intro/intro1_2.html

위 [그림 3-12]와 같이 Git의 Repository 구조를 이용하여 프로젝트 별 원격 저장소를 생성하고 각 개발자 PC에 로컬 저장소를 생성하여 소스 코드에 대한 관리 및 이력 관리를 제공하게 된다. 저장소를 생성하게 되면 생성한 폴더에 '.git'이라는 디렉토리가 만들어진다. '.git'이라는 디렉토리에는 저장소에 필요한 Skeleton이 들어 있다.[17] 따라서 개발자 PC, 테스트를 진행할 서버 내 각 프로젝트, 운영 서버 내 각 프로젝트에는 로컬 저장소를 생성하고 Git 서버인 Bonobo에는 원격 저장소를 생성한다. PaaS 시스템에서는 Git Repository를 컨테이너로 구성하거나 외부 서버를 통해 구성을 해야 한다.

3.3.2.3 Branch

Branch는 Git에서 가지는 일상적인 개발 프로세스의 일부로써 효과적으로 변경사항의 스냅샷을 표시하는 포인터이다.[18] 개발자들은 소프트웨어 개발을 진행할 때 동일한 소스 코드를 가지고 공동으로 작업하는 경우가 많은데 그럴 경우, 동일한 소스 코드에 서로 다른 기능의 추가나 버그를 수정하고 변경된 내용을 반영해야 하는 경우가 많다. 다수의 개발자가 동일한 소스 코드에 대하여 동시에 여러 가지 요구된 작업을 할 수 있도록 만들어 주는 기능이 바로 Branch 기능이다.

3.3.2.4 Jenkins

Jenkins는 소프트웨어 구축, 테스트, 전달 또는 배포와 관련된 모든 종류의 작업을 자동화하는데 사용할 수 있는 독립형 오픈 소스 자동화 서버이며 기본 시스템 패키지를 도커를 통해 설치하거나 JRE(Java Runtime Environment)가 설치된 모든 시스템에서 독립 실행형으로 실행할 수도 있다.[19] 대표적인 특징으로는 1)훌륭한 커뮤니티 지원을 제공하는 오픈 소스 도

17) git web page: https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

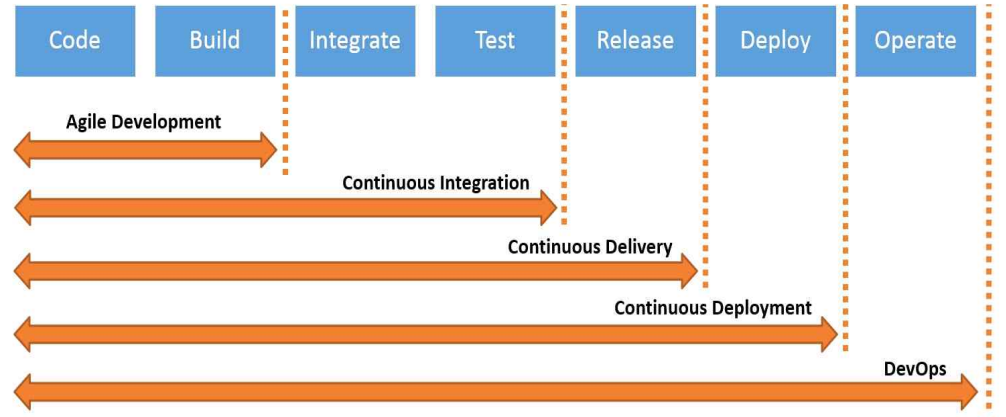
18) 출처: https://www.atlassian.com/git/tutorials/using-branches

19) what is jenkins?(https://www.jenkins.io/doc/)

구이며, 2)설치가 용이하며 작업을 수월하게 만드는 1000개 이상의 플러그인을 제공한다. 3)플러그인이 존재하지 않으면 코드를 작성하고 커뮤니티와 공유도 가능하고 4)Java 기반으로 호환되는 모든 플랫폼으로 이식도 가능하다.

3.3.3 CD(Continuous Delivery)

지속적 전달(CD)은 업데이트/소프트웨어 변경사항을 프로덕션에 점진적으로 전달하는 것이며 CI의 확장 역할을 한다. CD를 사용하면 전체 소프트웨어 릴리스 프로세스를 자동화할 수 있다. 개발한 소프트웨어를 배포하기 위하여 통합 테스트, 기능 테스트, 성능 테스트, 보안 테스트, UI 테스트 등과 같은 절차로 수행하고 단위 테스트를 진행한다. 이는 개발자가 더 완전한 유효성 검사를 수행할 수 있도록 소프트웨어 버그가 없는 배포를 보장하기 위한 CD를 구현하면 새로운 기능을 자주 릴리스할 수 있으므로 고객 피드백이 향상되고 고객 참여도가 향상된다. 따라서 CI/CD는 DevOps 기술에서도 매우 중요한 부분이라 할 수 있다.[20]



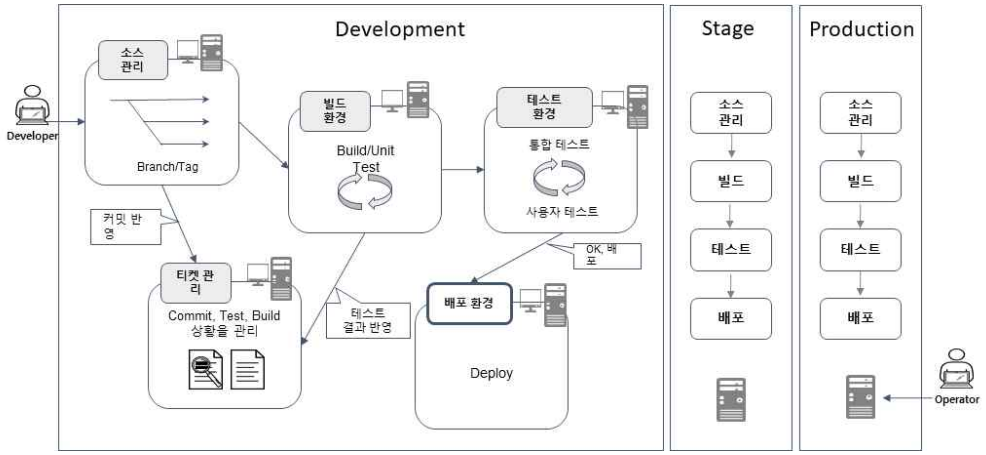
[그림 3-13] CI/CD의 전달 및 배포(bmc, 2017)

20) bmc blog; https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/ accessed 2021-12-01

지속적인 배포 프로세스로 이동하면 여러 자동화 구성요소가 제자리에 있게 된다. 지속적 통합 빌드 서버와 지속적 전달은 모두 자동화되어야 하며 프로덕션에 자동으로 배포할 수 있는 기능이 있어야 한다.

3.3.4 CI/CD 파이프라인

본 연구에서는 CI/CD 파이프라인 구성을 통해 개발 및 운영환경의 효율성을 확보하고자 PaaS 플랫폼 환경으로 기능 구현을 설계하였다. 아래 [그림 3-14]에서는 CI/CD 환경에서 개발/테스트/운영 측면의 절차를 도식화를 하였다. 일반적으로 기업 환경에서는 개발을 위한 시스템 환경과 테스트를 위한 환경, 제품화된 소프트웨어를 서비스로 적용하여 운영하기 위한 시스템 환경으로 분리가 되어 있으며, 조직이 큰 규모에서는 각각의 시스템을 관리하는 담당자도 분리되어 있다.



[그림 3-14] CI/CD 파이프라인 절차(Red Hat, 2020)

개발 환경은 최초 개발부터 안정화 단계를 거쳐 제품화하기까지 시스템 환경에 따라 세부적으로 분리되기도 하고 표준화된 파이프라인 체계가 없다면 개발 조직간의 협업이나 버전관리 등의 문제가 발생하기 때문에 개발단계와 안정화 단계, 그리고 마지막으로 제품화 단계에서의 CI/CD 파이프라인 설계를 진행하였다.

IV. PaaS 플랫폼 구현

4.1 수요자 중심의 요구사항 설계

4.1.1 PaaS 플랫폼 도입 관련 설문조사

본 연구의 개선된 설계기준을 마련하기 위해서 온라인 설문을 시행하였으며 설문대상자 기준은 PaaS 플랫폼을 도입하였거나 1년 이내 도입예정인 기업의 책임자급 이상으로 선정하였다. 설문조사 기간은 2020년 11월에 395명, 2021년 3월에 286명, 총 2회에 걸쳐 681명을 대상으로 실시하였으며 불성실한 답변 및 모집단이 10명 이하인 업종은 제외하였다. 설문조사에 앞서 웹세미나 시간을 통해 충분한 내용전달을 하였으며 설문조사 작성시 데이터의 신뢰성을 높이기 위해 객관적이고 실무적인 관점에서의 설문작성 필요성에 대해 충분한 안내를 시행하였다.

본 연구에서 조사 분석한 설문내용과 모집단의 업종 종류는 [표 4-1]과 같이 공공, 금융, 정보통신, 제조업, 유통·물류, 의료, 교육, 방송·미디어 분야를 대상으로 집계를 하였다.

[표 4-1] PaaS 플랫폼 도입 관련 설문조사(락플레이스, 2021)

업종	Count	백분율
공공	32	5.10%
금융	24	3.82%
정보통신	291	46.34%
제조	192	30.57%
유통/물류	36	5.73%
의료	21	3.34%
교육	14	2.23%
방송/미디어	18	2.87%
Total	628	100%

본 연구에서 설문내용은 글로벌 오픈소스 플랫폼 단체인 CNCF에서 지난 수년간 전 세계 고객사를 대상으로 선정한 요구사항 정의서를 기준으로 선별하였으며 그중에서 비중이 높은 중요한 요구사항들을 제시하였다. 답변 내용은 중복응답이 가능하며 아래 [표 4-2]와 같이 마이크로서비스 구현, 애플리케이션 현대화, 비용절감, 애플리케이션 개발 및 변경속도 향상, 애플리케이션 호환성 및 이식성 향상, 애플리케이션 가용성 및 안정성 향상 등으로 구성된다.

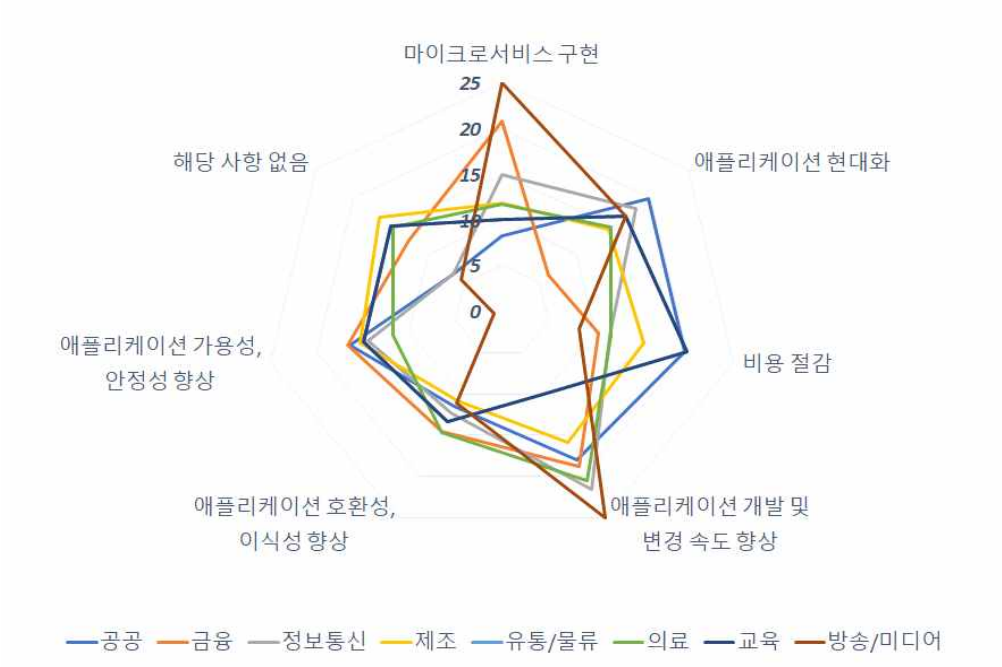
[표 4-2] PaaS 플랫폼 도입시 요구사항 설문조사(락플레이스, 2021)

답변 내용	Count	백분율
마이크로서비스 구현	163	14.20%
애플리케이션 현대화	177	15.42%
비용절감	162	14.11%
애플리케이션 개발 및 변경속도 향상	212	18.47%
애플리케이션 호환성, 이식성 향상	159	13.85%
애플리케이션 가용성, 안정성 향상	153	13.33%
해당 사항 없음	119	10.37%
ETC	3	0.26%
Total(중복응답 포함)	1,148	

4.1.2 요구사항 분포도

요구사항에 대한 업종별 분포도의 조사를 통해 업종별로 어떠한 기능을 강조하고 있는지 알 수 있다. 아래 [그림 4-1]과 같이 방사형 그래프로 표시하면 마이크로서비스 구현 기능은 방송/미디어 분야와 금융 분야에서 많은 요구사항이 있다는 것을 알 수 있다. 애플리케이션 현대화 기능은 금융 분야를 제외한 대부분 분야에서 높은 비중의 요구사항을 보여주고 있음을 알 수 있다. 비용절감 요구사항은 공공 분야와 교육 분야, 제조 분야에서 높은 요구

사항을 보여주고 있다. 애플리케이션 개발 및 변경속도 향상 기능은 교육 분야를 제외하고 모든 분야에서 높은 요구사항이 있으며 특히 방송/미디어 분야에서 강한 요구사항을 보여주는 것을 알 수 있다. 그리고 애플리케이션 호환성 및 이식성 향상에 대한 기능은 거의 모든 분야에서 중요도의 비중이 유사하다는 것을 알 수 있다. 애플리케이션 가용성, 안정성 향상의 기능 또한 모든 분야에서 중요도의 비중이 유사함을 알 수 있다.



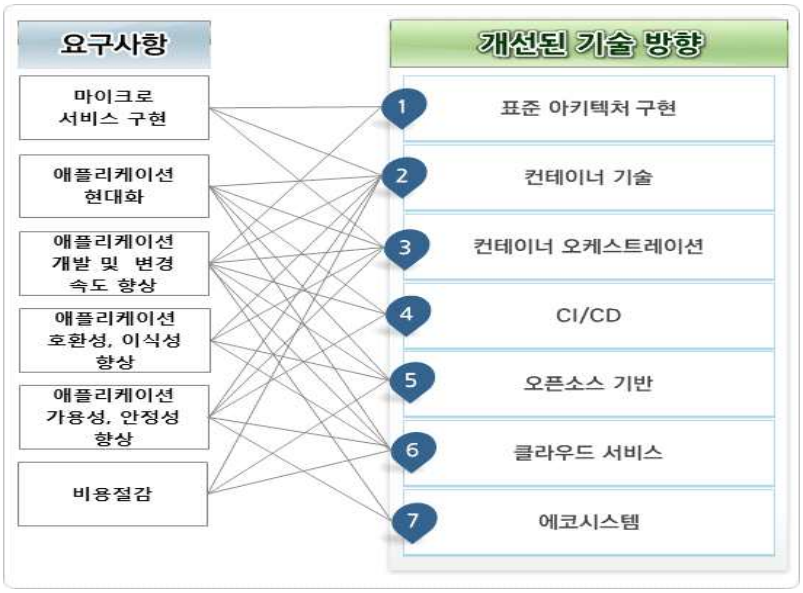
[그림 4-1] 업종별 요구사항 설문조사(락플레이스, 2021)

정보통신 및 제조업 분야의 담당자가 가장 많은 응답률을 보인 설문 결과를 바탕으로 해당 산업 분야 기준 기술의 방향성을 정립하였다.

4.1.3 개선된 기술 방향 정립

요구사항 설문의 결과에 따라 기술의 방향성을 정리하여 아래의 [그림

4-2]처럼 개선된 기술을 정립하였다.



[그림 4-2] 요구사항을 반영한 개선된 기술의 방향(락플레이스, 2021)

개선된 기술에 따른 방향성을 구체적으로 다음과 같이 가이드라인 제시를 위해 정리하였다.

① PaaS 기반의 표준 인프라 및 개발/운영환경 구축

② VM보다 작은 단위의 용량과 의존성을 제거한 애플리케이션 패키징 기술 적용

③ 컨테이너로 서비스하는 기능 측면과 관리 측면에서 오케스트레이션 기술이 요구됨

④ 개발된 소스 코드의 잦은 변경과 개발 소스를 테스트 및 배포하는 자동화 기술 구현

⑤ 오픈소스 기술이 접목된 솔루션으로 구성하여 벤더 종속성과 같은 독점 제약 조건을 제거

⑥ 프라이빗, 퍼블릭 클라우드 환경에서 서비스될 수 있는 환경을 제공

⑦ 다양한 개발 도구가 지원되는 플랫폼 생태계 및 기술 지원 파트너 협력

위 정리된 가이드라인을 바탕으로 본 연구에서는 PaaS 플랫폼과 CI/CD 환경 구현을 진행하였다.

4.2 컨테이너 시스템 구현

컨테이너 시스템은 컨테이너를 구현하는 Engine 기술과 컨테이너를 다양한 요구사항에 맞춰 관리하는 오케스트레이션 기술을 통해 구현이 필요하고 이러한 환경을 지원하는 대표적인 기술이 도커와 Kubernetes이다. 본 연구에서는 이론적 배경에서 언급한 Kubernetes 환경과 이를 지원하기 위한 CRI-O, Podman Engine 기술을 적용한 오픈시프트 플랫폼 소프트웨어를 사용하여 컨테이너 시스템을 구현하였다.

4.2.1 하드웨어 구성

4.2.1.1 서버

범용적인 환경의 서버를 선택하는 것이 무엇보다 중요하며, 특히 리눅스가 잘 지원되는 하드웨어로 선택하였으며 자세한 상세 사양 정보는 아래 [표 4-3]과 같다.

[표 4-3] 서버 세부 사양 정보(락플레이스, 2021)

구 분	상세 내용
x86 CPU	Intel® Xeon® Gold 5218 @ 2.30GHz (2P16C)
Memory	1024GB
HDD	300GB*2ea
Network Interface	Ethernet 1Gb 4-port 366 FLR Adapter - NIC * 1ea Ethernet 10Gb 2-port 562SFP+ Adapter*2ea

CPU는 소켓당 16 Cores로 총 32 Cores가 장착되어 있고 메모리는 1TB

이며, OS 설치용 내장 디스크는 300GB 2장으로 Mirroring(RAID 1)을 적용하였다. 그리고 네트워크 포트의 경우에는 1Gbit는 서버를 관리하는 네트워크로 설정하였고 10Gbit는 컨테이너에 필요한 스토리지 네트워크(NAS 망)와 컨테이너가 통신하기 위하여 사용하는 네트워크를 논리적으로 나누어서 사용하기 위해 필요하다.

[그림 4-3] 구성 서버(락플레이스, 2021)

서버는 위 [그림 4-3]에서 보는 바와 같이 표준 Rack에 장착할 수 있는 서버를 선택하여 공간 및 관리 측면의 효율성을 확보하였다.

4.2.1.2 네트워크 스위치

네트워크는 클라우드 서비스 환경에서 속도가 상당히 중요하기 때문에 10Gbit 이상의 내부 네트워크 환경을 갖춰야 하며, 대용량의 I/O 부하가 요구되는 환경이라면 20Gbit나 40Gbit의 인피니밴드급 네트워크 속도를 보장해야 한다. 본 연구에서 사용된 네트워크 환경은 10Gbit 네트워크 환경으로 4~5년 전부터 클라우드 환경의 구현이 필요한 경우, 내부 시스템 네트워크는 10Gbit로 구성하였다. 본 연구의 환경 구성은 PaaS 환경 구성으로 물리적 서버 및 논리적인 가상 컨테이너 환경이 스케일 아웃 구조로 확장될 수 있는

설계로 구성이 필요하고 네트워크 IP 주소의 경우, 클래스 단위로 할당을 하여 구성을 해야 하므로 공인 사설 IP 주소를 사용하여 내부 시스템에 IP 주소를 설정하였다.

[표 4-4] 네트워크 스위치 기자재 세부 사양 정보(락플레이스, 2021)

구 분	상세 내용
TOR 스위치	48 SFP+ and 4 QSFP28, front to back air flow, AC * 2
GBIC	Quad Small Form Factor Pluggable 40GBase-SR4 Gigabit Ethernet Optics
	Small Form Factor plus Pluggable 10GBase-SR Gigabit Ethernet Optics
DAC 케이블	QSFP+ 40 Gigabit Ethernet Direct Attach Copper

네트워크 스위치의 경우, 속도 및 가용성과 안정성이 상당히 중요하므로 아래 [그림 4-4]와 같이 이중화 구성을 적용하였다.

[그림 4-4] 구성 네트워크 스위치(락플레이스, 2021)

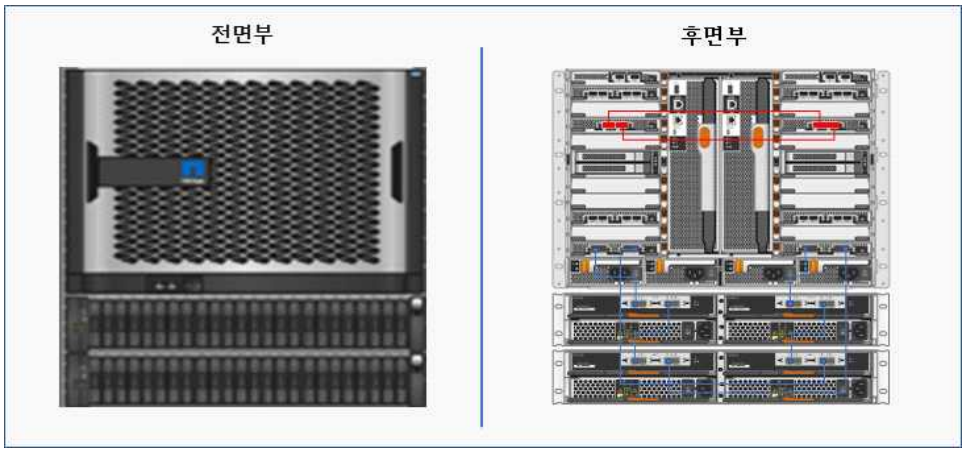
4.2.1.2 스토리지

스토리지는 데이터가 저장되는 공간으로 안정성과 성능이 충족되어야 한다. 컨테이너를 사용하는 시스템에서는 Persistent Volume이라 불리는 디스크 영역이 각각의 컨테이너에 할당되도록 구성이 되어야 하며 Network Attached Storage(NAS) 방식을 사용하여 구성하였다.

[표 4-5] NAS 스토리지 세부 사양 정보(락플레이스, 2021)

구 분	상세 내용
Controller	AFF A700,HA,CTL,Encl,AC PS,40G,SAS,-C
3.8TB 디스크쉘프	SSD Shelf,12G,12x3.8TB,2P,-C
1.2TB 디스크쉘프	Disk Shelf,12G,24x1.2TB,10K,-QS
960GB 디스크쉘프	SSD Shelf,12G,24x960GB,-QS
10Gb IO Module	IO Module,4-PT CNA,10GbE,16GB FC

스토리지의 경우, 데이터 증가량에 따라 추가로 스토리지 증설이 필요하므로 별도의 전용 Rack을 사용하여 확장이 용이하도록 구성하였다.

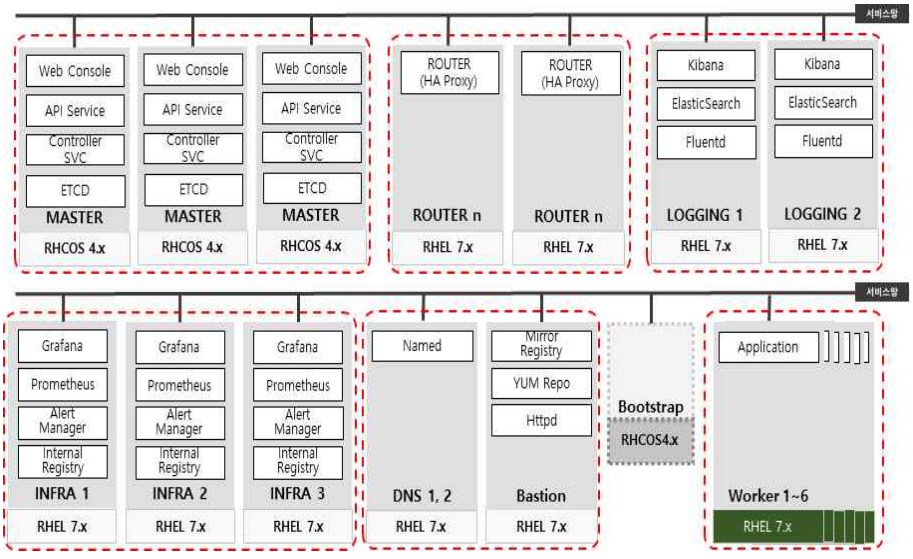


[그림 4-5] 구성 스토리지(락플레이스, 2021)

4.2.2 컨테이너 매니저 구현

컨테이너 관리를 위한 매니저는 사용자 컨테이너의 시작부터 종료까지의 생명주기(Life Cycle)를 관리가 필요하고, 컨테이너의 컴퓨팅 자원을 모니터링, 서비스 컨테이너를 위한 레지스트리(Registry) 저장소와 연결하여 관리하는 기능이 포함된다.

컨테이너 관리를 위해 구성한 Master 서버는 아래 [그림 4-6]의 구성과 같이 3중화 환경으로 구현하여 물리적 또는 네트워크 통신 등의 장애가 생겼을 경우에 전체 장애가 발생하지 않도록 구현하였다. 또한 PaaS 플랫폼에서 요구되는 모니터링과 메트릭(Metric), 컨테이너 이미지 관리용 레지스트리 구성을 위한 인프라 서버를 구성하고, 다양한 상태의 로그 정보 수집을 위한 로깅(Logging) 서버와 CI/CD 환경 구현을 위한 실제 컨테이너 애플리케이션이 실행되는 워커(Worker) 서버를 분리 구성하였다. 컨테이너는 확장이 되면 상당히 많은 수량이 배포되기 때문에 IP 주소를 통한 여러 개의 서비스 제공은 한계가 있으므로 Domain Name Service를 통해 서비스되도록 구현하기 위하여 DNS 서버도 구성하였다.



[그림 4-6] 컨테이너 시스템 Infra 구성(락플레이스, 2021)

각각의 서버에 할당한 서버의 상세 용량 규격은 아래 [표 4-6]과 같이

적용하였으며, PaaS 플랫폼은 Openshift 4.8 버전을 기준으로 요구하는 시스템 자원을 고려하여 적용하였다.

[표 4-6] 컨테이너 시스템 서버 목록(락플레이스, 2021)

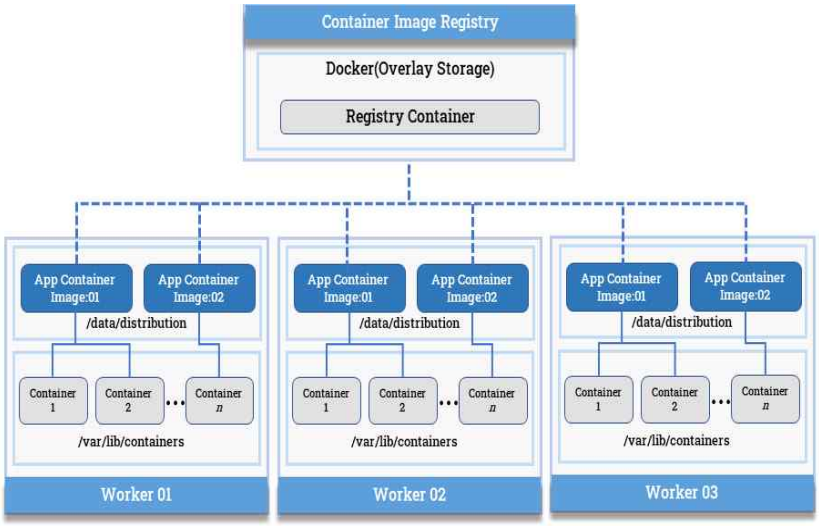
Hostname	OS	CPU (Cores)	Memory (GB)	Local Disk (GB)	Storage (GB)
bastion	RHEL7.x	8	32	200	500
master01	RHCoreOS4.x	4	16	200	
master02	RHCoreOS4.x	4	16	200	
master03	RHCoreOS4.x	4	16	200	
router01	RHCoreOS4.x	2	8	200	
router02	RHCoreOS4.x	2	8	200	
infra01	RHCoreOS4.x	8	32	200	
infra02	RHCoreOS4.x	8	32	200	1,000
infra03	RHCoreOS4.x	8	32	200	
logging01	RHCoreOS4.x	8	32	200	
logging02	RHCoreOS4.x	8	32	200	1,000
logging03	RHCoreOS4.x	8	32	200	
worker01	RHCoreOS4.x	8	64	200	
worker02	RHCoreOS4.x	8	64	200	
worker03	RHCoreOS4.x	8	64	200	10,000 (App Storage)
worker04	RHCoreOS4.x	8	64	200	
worker05	RHCoreOS4.x	8	64	200	
worker06	RHCoreOS4.x	8	64	200	

4.2.3 서비스 컨테이너 구현

서비스를 위한 컨테이너는 Worker 서버에서 컨테이너가 실행되기 위해서 서비스용 컨테이너를 구현하는 작업이 필요하며, 개발자가 컨테이너 이미지를 쉽게 제작할 수 있는 템플릿 기반의 예제 카탈로그(Catalog)를 제공하며, 또한 외부 공인된 저장소에서 표준으로 제공하는 컨테이너 이미지를 연계하여 구성할 수 있도록 하였다. 프라이빗 클라우드 환경은 서비스의 특성상 외부와 단절된 네트워크 환경 구성으로 별도의 서비스 컨테이너를 위한 오퍼레이터를 구현하였다.

4.2.4 컨테이너 저장소 구현

컨테이너가 실행되기 위해서는 실행 가능한 컨테이너 이미지가 필요하며, 컨테이너 빌드(Build) 파일을 통해 해당 이미지는 변경이 되기 때문에 변경된 이미지 관리를 위해서는 저장소가 필요하다. 이 저장소는 아래 [그림 4-7]과 같이 컨테이너 이미지가 저장되어있는 위치 정보를 보관하고 있으며 컨테이너 이미지 실행이 요청되면 이미지 Pull 방식을 이용하여 실행이 필요한 Worker Host 서버에 다운로드가 된다.



[그림 4-7] 컨테이너 저장소 구조(락플레이스, 2021)

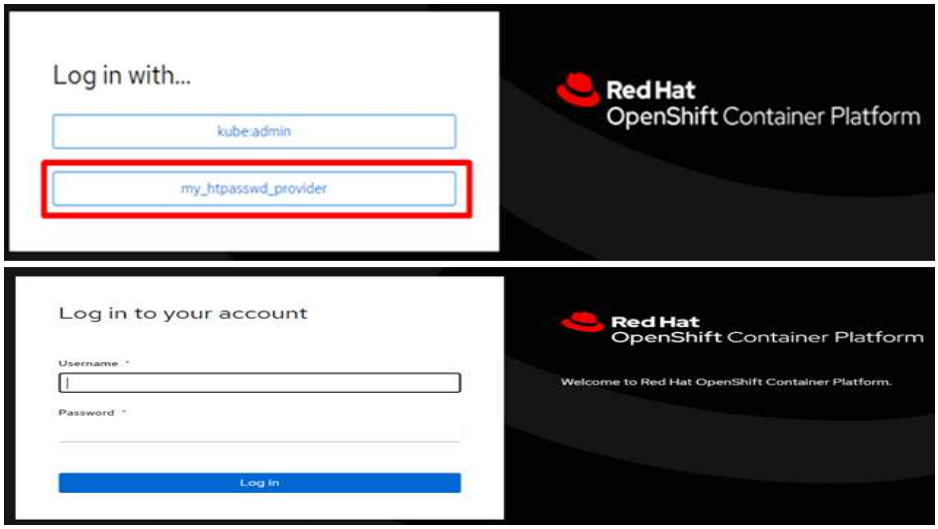
컨테이너 이미지 저장소는 외부 저장소를 사용하는 경우, 사용자가 수정할 수 있으며 본 연구에서는 오픈소스로 제공되는 Docker Distribution을 사용하여 구현하였다.

4.2.5 컨테이너 시스템 구현 결과

컨테이너 시스템은 레드햇의 오픈시프트를 사용하였고, 3중화된 환경으로 구성되어 서버 1대의 장애가 발생하더라도 문제가 발생하지 않도록 동작되고 HA Proxy를 통한 로드밸런싱 기능이 적용되었다.

4.2.5.1 로그인 화면

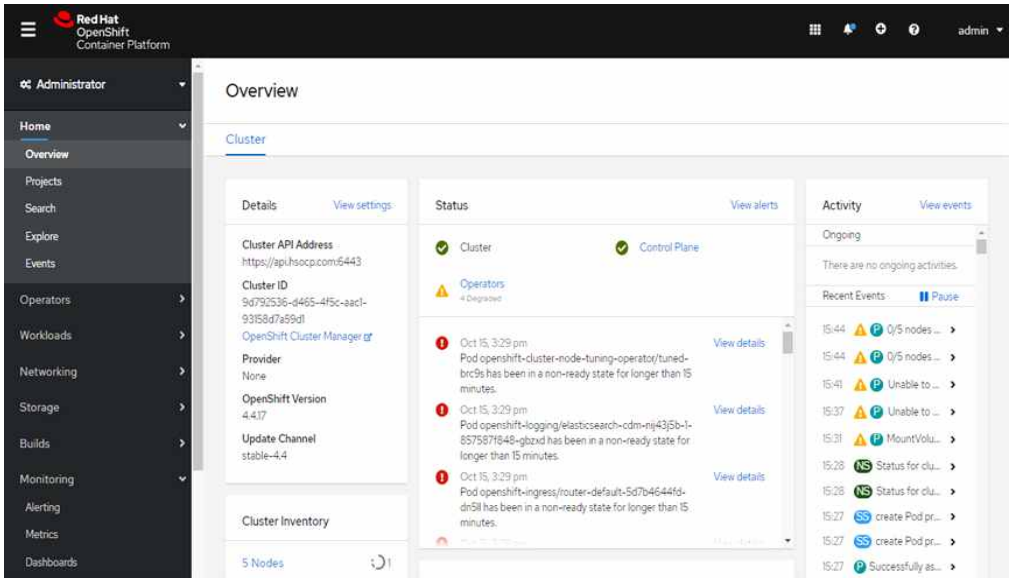
실제 구현된 오픈시프트의 대시보드 화면은 아래 [그림 4-8]과 같이 웹 UI로 접속할 수 있다. 로그인시 my_htpasswd_provider 버튼을 클릭한 후 Username과 Password를 통해 로그인이 가능하도록 구현하였다.



[그림 4-8] 오픈시프트 로그인 화면(락플레이스, 2021)

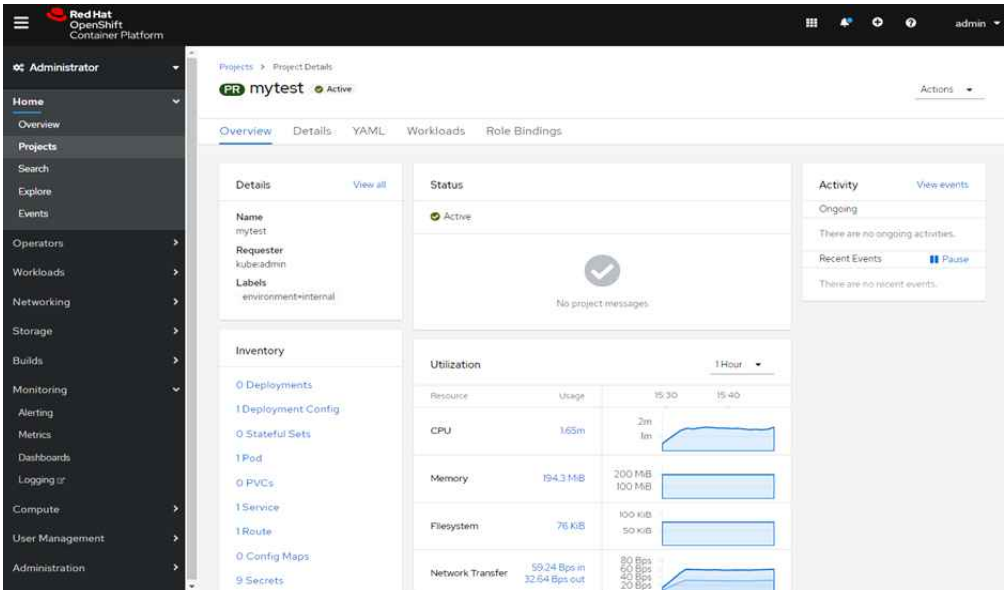
4.2.5.2 웹 대시보드 화면

Administrator 권한으로 로그인하면 아래의 [그림 4-9]와 같은 화면을 볼 수가 있으며, 컨테이너 생성, 삭제, 모니터링 등과 별도 컨테이너 이미지를 등록할 수 있는 기능 등을 사용할 수 있다.



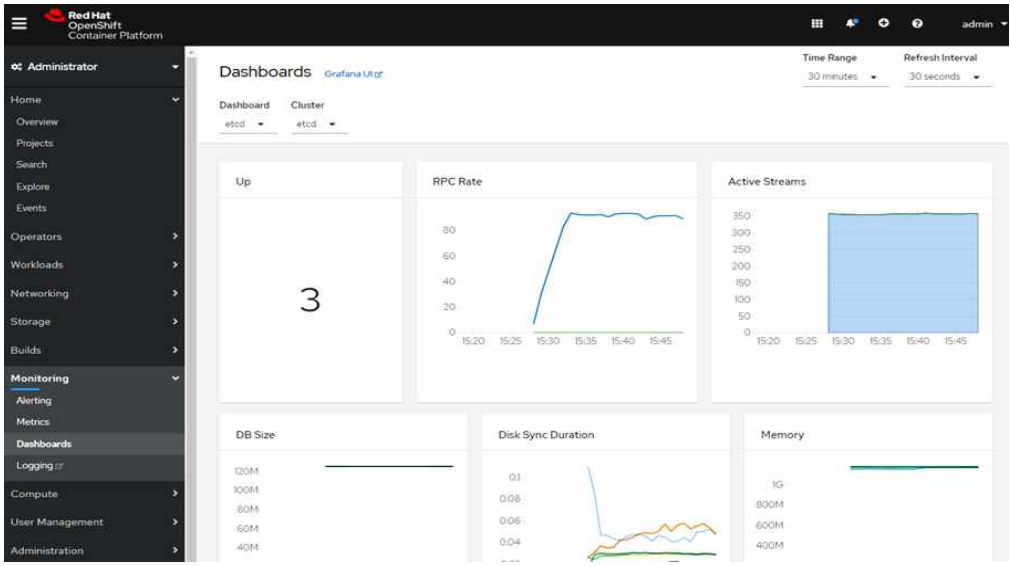
[그림 4-9] 웹 대시보드 초기 화면(락플레이스, 2021)

로그인하면 Overview 화면부터 볼 수가 있으며, 현재 시스템의 상태 정보를 확인할 수 있다. 아래 [그림 4-10]의 경우는 컨테이너 등의 리소스를 묶는 논리적인 단위로 정의된 프로젝트 화면으로 사용자는 권한을 부여받아서 해당 프로젝트에 접근하여 컨테이너 생성 등의 작업을 수행할 수 있는 화면이다.



[그림 4-10] 웹 대시보드 프로젝트 화면(락플레이스, 2021)

추가로 전체 오픈시프트 상태를 모니터링 가능한 화면을 통해 실시간으로 확인할 수 있다. 아래 [그림 4-11]에서와 같이 모니터링 화면을 확인할 수 있다.



[그림 4-11] 웹 대시보드 모니터링 화면(락플레이스, 2021)

모니터링 화면의 경우, 오픈소스 기반의 Grafana를 통해 기능이 구현되어 있어서 API(Application Provider Interface)를 통해 별도의 모니터링 시스템과 연계할 수 있다.

4.3 CI/CD 오픈소스 기술 구현

CI/CD 기술은 클라우드 산업의 성장과 디지털 전환의 중심에서 신속한 개발 및 서비스 운영을 위한 핵심 기술로 부상하고 있다. 3장에서 설계한 내용을 기반으로 PaaS 플랫폼에서 컨테이너 기술을 이용한 CI/CD 파이프라인을 구성하였다.

4.3.1 오픈소스 기술 스택

CI/CD를 지원하는 오픈소스 프로젝트 재단인 CNCF의 제품군에서 구현한 소프트웨어는 아래 [표4-7]과 같이 Jenkins와 GitLab이 대표적이다.

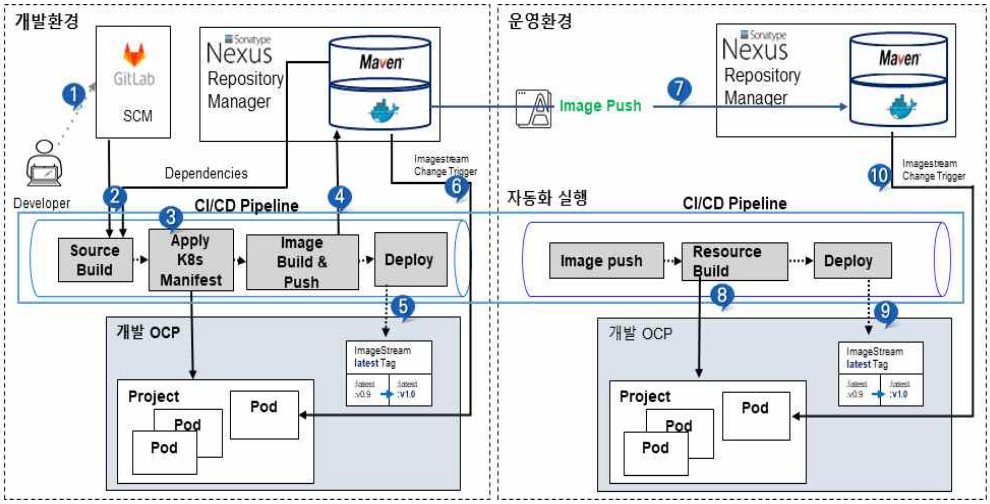
[표 4-7] 구현된 CI/CD 오픈소스 제품군 (락플레이스, 2021)

구분	소프트웨어 이름	제공 사이트
형상관리	GitLab	https://about.gitlab.com/install/
저장소 (Repository)	nexus	https://www.sonatype.com/products/repository-oss-download
품질검사	sonarqube	https://www.sonarqube.org/downloads/
CI Tool	Jenkins	https://www.jenkins.io/download/
Build	Maven	https://maven.apache.org/download.cgi

4.3.2 CI/CD 파이프라인 구성

빌드와 배포를 중심으로 CI/CD를 구성하였으며, PaaS 플랫폼 환경에서 개발과 운영환경의 분리 후 개발 시스템이 운영 시스템에 영향을 미치는

위험 요소를 최소화하여 아래 [그림 4-12]와 같이 구현하였다.



[그림 4-12] CI/CD 파이프라인 구성도(Red Hat, 2021)

위 [그림 4-12]에서 수행되는 CI/CD 절차는 다음과 같은 순서로 진행된다.

① 개발자가 소스 코드를 SCM(Supply Chain Management)에 PUSH 한다.

② Git 서버에서 소스 코드와 Maven에서 의존성 라이브러리를 가져와서 빌드한다.

③ PaaS 환경에 필요한 구성요소들을 배포한다.(deployments, service…등)

④ Docker 빌드를 통해 생성된 이미지를 Docker 저장소에 Push 한다.

⑤ Latest Imagestream tag를 현재 이미지 버전의 tag로 업데이트한다.

⑥ Imagestream change Trigger를 이용하여 자동 배포한다.

⑦ 릴리즈된 Docker 이미지를 운영환경의 Docker 저장소로 Push 한다.

⑧ PaaS 환경에 필요한 구성요소들을 배포한다.(deployments, service…)

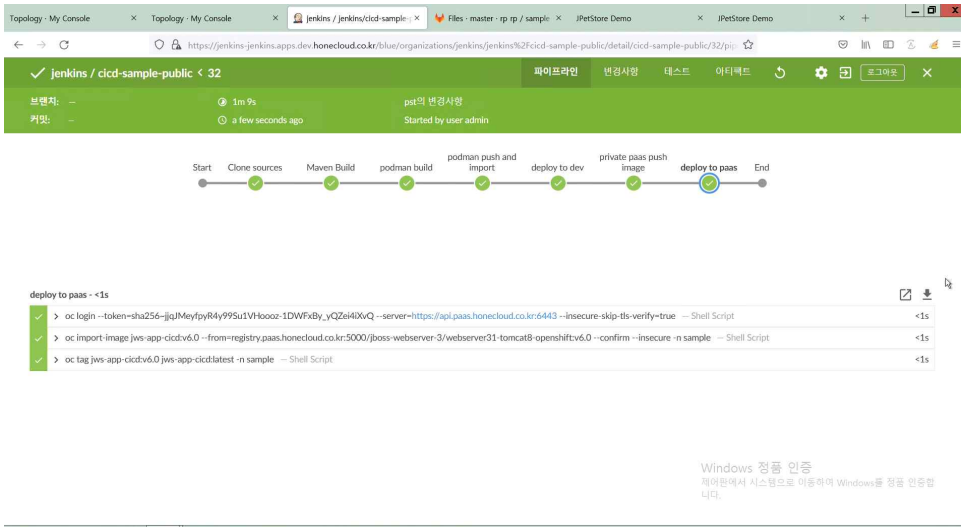
⑨ Latest Imagestream tag를 현재 이미지 버전의 tag로 업데이트한다.

⑩ Imagestream change Trigger를 이용하여 자동 배포한다.

CI/CD는 운영 정책에 따라 변경이 필요하며, 본 연구에서 구현한 시스템은 실제 수요 기업의 시스템 운영환경에 맞추어 구성되었다.

4.3.3 CI/CD 파이프라인 구현 결과

Jenkins 파이프라인을 적용하고 오픈시프트에서 배포된 화면을 통해 CI/CD 환경 구현이 정상적으로 되었음을 확인하였다. 아래 [그림 4-13]은 Jenkins를 통해 CI/CD 파이프라인을 통한 애플리케이션 빌드(build) 과정이 진행되는 화면을 확인한 것이다.

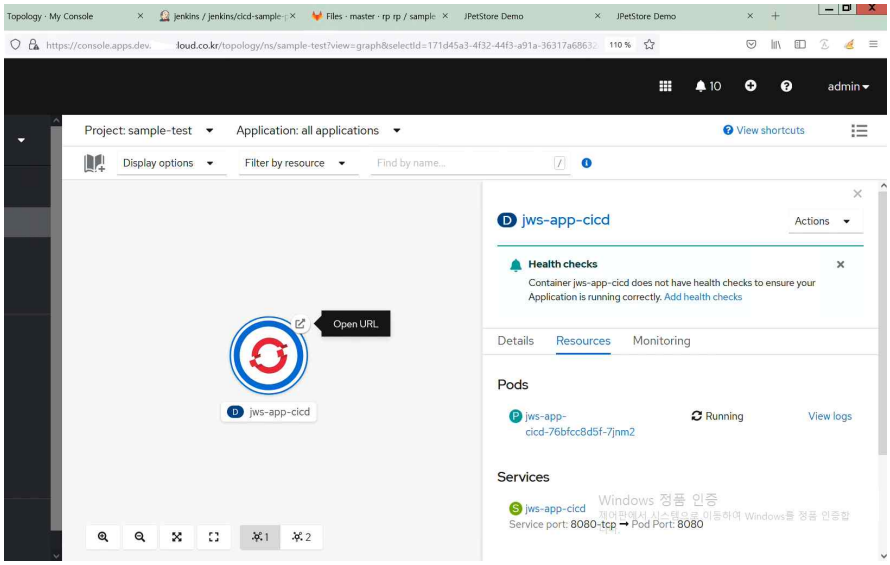


[그림 4-13] Jenkins의 CI/CD 파이프라인 구현 화면(락플레이스, 2021)

위 [그림 4-13]의 Jenkins 화면에서 CI/CD 파이프라인이 실행되는 과정을 보면 소스 코드를 복제하고 Maven을 통한 소스 빌드, 포드맨 컨테이너 빌드, 컨테이너 이미지를 Push하여 적재하고 dev 환경에 디플로이 한다. 그리고 Push된 이미지를 프라이빗으로 전달하고 배포한 후 종료한다.

오픈시프트에서 배포된 애플리케이션을 확인하면 아래 [그림 4-14]와

같이 배포되어 실행 중인 것을 확인할 수 있으며, 해당 애플리케이션이 실행 중인 아이콘을 클릭하면 애플리케이션의 상태 정보도 확인할 수 있다.



[그림 4-14] 오픈시프트에 배포된 애플리케이션 확인 화면(락플레이스, 2021)

4.4 개선된 오픈소스 기술 적용

4.4.1 수요처 기업

4.4.1.1 개선 효과

PaaS 플랫폼 환경에서 오픈소스 CI/CD 파이프라인을 적용하여 개선된 기업의 효과는 아래 [표 4-8]과 같이 차이가 발생하는 것으로 확인되었다.

기존 사용 환경이 클라우드 인프라와 컨테이너 기술이 적용된 환경이 아니었기 때문에 외부 개발자나 개발 프로젝트에 신규로 참여하는 인력이 승인 절차부터 개발을 시작하는 단계까지 10일에서 14일 정도의 시간이

소요되었다. 이에 반하여 PaaS가 도입된 환경에서는 3.5시간 정도로 차이가 크게 나는 것을 확인할 수 있다.

[표 4-8] 수요 기업의 개선 효과(Red Hat, 2020)

기존 환경	소요 기간	PaaS 환경	소요 기간
외부 개발자 계정 및 권한 생성	1일	외부 개발자 계정 및 권한 생성	1시간
개발 시스템 구성	4~6일	웹 브라우저 접속	0.5시간
접속 환경 설정	1일	컨테이너 배포	1시간
개발 시스템 환경 설정	2~3일	Cloud IDE 툴 사용	1시간
IDE 툴 설정	2~3일		
총 시간	10~14일	총 시간	3.5시간

실제 기업 사례는 정보통신과 제조업의 기업 사례가 있으며 현재 퍼블릭 클라우드를 제공하는 해외 및 국내 기업들은 다양한 CI/CD 서비스를 제공하고 있다.

V. 결론

5.1 결론 및 시사점

본 논문에서는 클라우드 서비스 적용을 위해 꼭 필요한 PaaS 플랫폼을 구현함에 있어서 CI/CD 파이프라인 설계를 최적화하기 위한 오픈소스 기반의 소프트웨어 플랫폼 구성방안을 제시하고자 하였다.

PaaS 플랫폼이 IaaS나 SaaS 서비스 대비 시장규모가 작지만, 성장성이 비교적 급증하고 있는 시점에서 기술적인 측면의 접근과 실제 수요자를 대상으로 설문한 결과를 분석하여 PaaS 플랫폼의 구성방안을 제시함으로 향후 PaaS 플랫폼의 수요 증가에 기여를 했다는 데 의의가 있다.

PaaS 플랫폼의 구성은 오픈소스 프로젝트 재단인 CNCF가 관리하는 Kubernetes의 아키텍처 분석을 통해 제공되는 컨테이너 런타임 기술을 분석하였다. 그리고 이러한 기술이 적용된 오프시프트 플랫폼이 제시하는 기술의 방향성에 대한 산업적 근거 마련에 목적을 두고 있다. 컨테이너 관리를 위해 제공되는 오케스트레이션 도구의 최적화 구현을 위한 분석을 위해 PaaS 플랫폼을 최근에 도입하였거나 1년 이내에 도입할 예정이 있는 기업의 수요조사 후 요구사항을 정의하고 이를 반영하였다. 결과로는 산업군별 요구사항의 유형과 사유에 대해 분석을 할 수 있게 되었으며 이를 기반으로 정보통신과 제조업 분야에 최적화된 PaaS 플랫폼 설계에 중요한 기준을 정립할 수 있었다. 설계기준 및 구현을 위한 주요 이슈 사항은 다음과 같다.

첫 번째, 애플리케이션 개발 및 변경속도 향상이 가장 큰 비중을 차지하였다. 두 번째, 애플리케이션 현대화에 대한 요구사항이 그다음으로 높게 나타났다. 세 번째, 애플리케이션 가용성 및 안정성 향상에 대한 요구사항 또한 상당수 있었다. 네 번째, 마이크로서비스 구현에 대한 요구사항도 있었다. 다섯 번째, 비용절감에 대한 요구사항이 있었다. 마지막 여섯 번째, 애플리케이션 호환성 및 이식성 향상에 대한 요구사항이 있었다. 애플리케이션 개발 및 변경속도 향상과 애플리케이션 현대화 측면에서 관심도가 높게 나타났으며

나머지 요구사항은 상대적으로 차이가 크지 않은 수치 결과를 보였다. 이러한 결과로 보았을 때 PaaS 플랫폼은 자동화 도구의 기능이 보편화하도록 기능 개선을 함으로써 개발과 운영 측면에서 더욱 활성화되는 환경을 마련할 수 있을 것으로 기대한다.

5.2 연구의 한계점 및 향후 연구 방향

PaaS 플랫폼은 기업에서 소프트웨어 개발을 위한 시스템으로 구현을 하는데 이러한 시스템 환경은 운영환경도 고려대상에 포함된다. 본 논문에서 다루고 있는 부분은 기술적 관점에서 애플리케이션 요구사항을 분석하였으며 운영환경 기준의 가이드라인을 제시하고 있다. 설문 대상자는 시스템 도입 관련 책임자급을 담당자 대상으로 하는 점에서 실무적 관점의 요구사항은 적합하지만, 기술적 관점에서는 다소 제약사항이 존재할 수 있을 것이다.

PaaS 플랫폼 기준의 CI/CD 파이프라인 설계 및 적용을 위해선 해당 기업 내부의 정보통신 및 정보보안 인프라의 분석이 사전에 이루어져야 좀 더 최적화된 구현이 가능하므로 도입예정인 기업의 기존 인프라 환경 및 시스템의 특성을 잘 파악하여야 하는 이슈가 있었다.

본 연구에서 가장 많은 응답을 보인 정보통신 및 제조업 분야는 급변하는 ICT 기술의 적용과 4차산업혁명과의 연계를 위한 융합형태의 서비스 플랫폼의 필요성이 증가하고 있다. 이러한 기준에서 수요자 관점에서의 요구사항을 좀 더 반영하기 위해서는 도입예정인 산업 분야의 지리적, 시대적, 경제적 특성에 대한 사전 파악 등이 필요하다. 향후 발전된 연구에서는 이러한 필요사항을 반영하고 기술적 실무자 관점에서의 요구사항을 수렴하여 반영한다면 또 다른 의미 있는 결과를 도출할 수 있을 것으로 기대한다. 또한 본 연구를 통해 최적화된 PaaS 플랫폼의 도입 후 일정 기간이 지난 후에 사용자 만족도 조사를 진행한다면 좀 더 수요자 중심의 최적화된 시스템 설계가 가능할 것으로 기대한다.

참 고 문 헌

1. 국내문헌

강다연, 김상현. (2020). 클라우드 컴퓨팅 서비스 도입 시 기대효과 요인의 중요도 우선순위 도출에 관한 연구. 한국콘텐츠학회논문지, 20(4), 564-570.

김광섭. (2018). "공개형 PaaS 클라우드 컴퓨팅 기반 공간정보 처리 시스템 개발 및 성능 평가." 국내박사학위논문 한성대학교 일반대학원.

김정보, 김정인. (2020). Cloud Native 환경에서의 생산성 향상을 위한 어플리케이션 개발 방법 연구. 멀티미디어학회논문지, 23(2), 328-342.

김영재. (2020). “클라우드의 동적 성능 변화에 적응하는 체크포인트 기반의 컨테이너 자율 이동성 지원 연구.” 국내석사학위논문 건국대학교 대학원.

김유성, 공수재, 김선진, 박준한. (2019). MSA에 대한 이해와 컨테이너 기술의 활용 방안. 정보통신기획평가원.

김응석. (2020). "금융 클라우드 서비스 도입 의도에 영향을 미치는 요인에 관한 실증연구." 국내박사학위논문 숭실대학교 대학원.

김철진. (2013). 소프트웨어 재사용성 향상을 위한 PaaS 기반 클라우드 컴포넌트 통합 연구. 한국산학기술학회 논문지, 14(2), 868-877.

박건주. (2018). "클라우드 환경의 IT인프라 비즈니스 모델에 관한 연구." 국내석사학위논문 숭실대학교 대학원.

박상엽. (2010). "PaaS 기반 웹 애플리케이션의 테스트 주도 개발 프로세스.“ 국내석사학위논문 숭실대학교 정보과학대학원.

배유미, 정성재, 소우영. (2014). 웹 서버 구성을 통한 가상머신과 컨테이너 방식 비교 분석. 한국정보통신학회지, 18(11), 2670-2677.

배춘선. (2019), 국내 Cloud와 Global Cloud 비교 분석을 통한 개선 방향 도출, 국내석사학위논문 숭실대학교 정보과학대학원.

이혁주, 김명진, 정종진, 최운. (2017). 다목적 컨테이너기반 서비스 운용을 위한 다중 컨테이너 오케스트레이터 관리 시스템 개발. 한국정보처리학회 학술대회논문집, 24(1), 419-422.

전우진, (2019). PaaS 클라우드 컴퓨팅을 위한 컨테이너 친화적인 파일 시스템 이벤트 탐지 시스템, 국내석사학위논문 세종대학교 대학원.

전인석, (2015). “보안을 고려한 무중단 환경에서 개발운영조직 통합관리(DevOps)”, 정보보호학회지 제25권 제1호, pp48.

정근훈, 박준석, 이극, (2019). 리눅스 컨테이너를 이용한 웹기반의 DevOps 플랫폼 연구, 한국융합학회지, 10(12), 71-80.

정근훈, (2020). "신속한 개발과 운영 전환을 위한 웹기반의 DevOps 플랫폼 국내박사학위논문 한남대학교 대학원.

정일훈, 오정훈, 박정흠, 이상진. (2011). IaaS 유형의 클라우드 컴퓨팅 서비스에 대한 디지털 포렌식 연구. 정보보호학회논문지, 21(6), 55-66.

정유진. (2017). “버전관리 시스템 기반의 상황인지 협업 웹 어플리케이션 설계.“ 국내석사학위논문 숭실대학교 대학원.

윤경식, 김영한. (2019). 마이크로서비스 아키텍처 기반의 통합 콘텐츠 관리 시스템 설계 및 구현. 정보처리학회논문지. 소프트웨어 및 데이터 공학, 8(3), 97-108.

이근열, (2020). 컨테이너 기반의 무중단 배포관리. 국내석사학원논문 숭실대학교 대학원.

이은정. (2015). 개발운영조직 통합관리(DevOps) 프로세스 연구, 국내석사학위논문 고려대학교 융합소프트웨어전문대학원.

이현경. (2021). 효율적인 (Continuous Integration and Continuous Deploy) 자동화 모델 설계. 국내석사학위논문 숭실대학교 대학원.

최원석, 정혜진, 나연묵. (2019). 컨테이너 기반 클러스터 환경에서 효율적인 자원관리를 위한 오케스트레이션 방법. 한국차세대컴퓨팅학회 논문지,

15(2), 71-78.

2. 국외문헌

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., ... & Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing (Vol. 17). Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Cerny Tomas, Donahoo Michael J., Trnka Michal. (2018). Contextual Understanding of Microservice Architecture: Current and Future Directions. SIGAPP Appl. Compute. Rev.. 17(4):29-45.

Johnson, J., Gesmer, L., Poort, J., & Mulder, H. (2015). CHAOS Report 2014. The Standish Group.

Lu, H. K., Lin, P. C., Huang, P. C., & Yuan, A. (2017). Deployment and Evaluation of a Continues Integration Process in Agile Development. Journal of Advances in Information Technology Vol, 8(4).

Ivanov, K. (2017). Containerization with LXC. Packet Publishing Ltd.

Kakadia, D. (2015). Apache Mesos Essentials. Birmingham, UK: Packt Publishing.

Kim, S., Park, S., Yun, J., & Lee, Y. (2008). Automated continuous integration of component-based software: An industrial experience. In 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2008. p. 423-426.

Naik, N. (2016). Building a virtual system of systems using docker swarm in multiple clouds. In 2016 IEEE international symposium on systems engineering (ISSE) (pp. 1-3). IEEE.

Nitin Naik, (2016). *Building a virtual system of systems using Docker*

Swarm in multiple clouds, IEEE International Symposium on Systems Engineering (ISSE). IEEE, pp. 1-3.

Rashid, A., & Chaturvedi, A. (2019). Cloud computing characteristics and services: a brief review. International Journal of Computer Sciences and Engineering, 7(2), 421-426.

Richardson, C. (2017). *A pattern language for microservices*. Retrieved February, 8, 2018.

ABSTRACT

Analysis of applied technology of cloud services and research on how to implement a PaaS platform based on CI/CD technology

Kim, Sam-Hyeon

Major in Master of Consulting

Dept. of Smart Convergence Technology Consulting

The Graduate School of Knowledge Service Consulting

Hansung University

Modern Information and Communication Technology (ICT) has created new paradigms such as AICBM and Metaverse through the development of hardware and software technologies. This has also affected the cloud industry, and has continued to develop into IaaS, PaaS, SaaS, AIaaS, and XaaS. In particular, the development of the Internet has formed a new market in the cloud industry field, and it has become possible to continue the growth of technology even in a global fendemic situation such as COVID 19.

With the spread and continued growth of cloud computing, the

paradigm of the entire software industry is changing, including software development methodologies, operating methods, distribution methods, and redistribution methods after modification. As a result, the scope of utilization of cloud technology has increased, and the requirements for platforms that support the development environment have also increased. In such an environment, the PaaS platform has emerged as a platform applicable to cloud services to support this in the process of expanding cloud services. The PaaS platform has a relatively low market share compared to IaaS and SaaS, but it is expected that the PaaS platform will be the top priority to reflect the rapidly changing requirements of the times. Therefore, we expect that the growth of the PaaS platform will lead the growth of IaaS and will eventually play a leading role in the cloud market.

Currently, the companies that lead the cloud market are focusing on expanding service construction by utilizing the cloud. The development environment has come to change while demanding changes to the cloud environment, including the platform necessary for developing and operating software(SW). This is also a criterion for judging that the demand for PaaS is increasing. The PaaS platform has been built around the past "virtual machine(VM) management" functions, but recently it has expanded its scope to include infrastructure management, application development, and operational support. Therefore, this study was able to provide guidelines that reflect consumer-centric requirements for the environmental factors of the PaaS platform required for development continuity.

Container technology and container orchestration management

capabilities are required to accommodate the evolving environment of cloud services in application development and implementation. We designed and built a system that applies CI/CD open source technology to improve development productivity by making good use of this. The construction system was designed and configured as a platform by analyzing the requirements of the actual responsible self-sufficiency personnel of the company, and the guidelines for platform configuration were set based on the requirements analysis.

By analyzing the requirements of the customer, we were able to confirm that the solution applying CI/CD technology can meet the requirements. Applying CI/CD technology to the PaaS platform and providing it as a cloud service will improve development productivity and will have a very positive impact on the spread of the PaaS platform.