Insight Report

2018. 12. 10. 제2018-004호

클라우드 가상화 기술의 변화

- 컨테이너 기반의 클라우드 가상화와 DevOps

Changes in Cloud Virtualization Technology

- Container-based cloud virtualization and DevOps

안성원 (swahn@spri.kr)†

SPRi 소프트웨어정책연구소

○ 이 보고서는 「과학기술정보통신부 정보통신진흥기금」을 지원받아 제작한 것으로 과학기술정보통신부의 공식의견과 다를 수 있습니다.

○ 이 보고서의 내용은 연구진의 개인 견해이며, 본 보고서와 관련한 의문사항 또는 수정・보완할 필요가 있는 경우에는 아래 연락처로 연락해 주시기 바랍니다.

- 소프트웨어정책연구소 기술·공학연구실 안성원(swahn@spri.kr) 선임연구원

《 요 약 문 》

가상화 기술은 클라우드 컴퓨팅을 가능하게 하기 위한 기반기술이다. 서버 자원의 효율적인 활용을 목적으로 등장한 가상화 기술은 크게 하드웨어 기반의 가상화 기술과 소프트웨어 기술로 각각 발전해 왔으며, 클라우드 컴퓨팅에 활용되고 있다. 최근 클라우드 컴퓨팅을 위한 가상화 기술은 컨테이너 기반의 오픈소스 플랫폼들이 대세를 이루고 있다. 이 리포트에서는 클라우드 컴퓨팅 최신 가상화 기술들을 살펴보고, 이런 기술들이 등장하게 된 배경과 향후 전망에 대해 논해 보고자 한다.

《 Executive Summary 》

Virtualization technology is the foundation technology for enabling cloud computing. Virtualization technology, which has emerged with the aim of efficiently utilizing server resources, has evolved into hardware-based virtualization technology and software technology, respectively, and has been utilized in cloud computing. Recently, virtualization technology for cloud computing has become popular with container-based open source platforms. This report looks at the latest virtualization technologies in cloud computing and discusses the background and future prospects for these technologies.

《 목 차 》

《 Contents 》

1. 논의배경

□ IT 패러다임 변화와 클라우드의 확산

세계의 IT 트랜드가 하드웨어(HW) 및 소프트웨어(SW)등을 직접 구축 및 설치(소유)하던 것에서 서비스의 형태로 빌려 쓰는(활용) 것으로 변화하고 있다. 클라우드 컴퓨팅은 사용자가 컴퓨팅 자원(Computing Resource)[1]을 직접 구축할 필요 없이 필요할 때마다 컴퓨팅 자원에 접근하여 데이터를 처리하고 연산을 수행할 수 있도록 서비스를 제공한다. 이처럼 클라우드 컴퓨팅을 활용하는 세계 IT 패러다임의 변화에 따라 클라우드 컴퓨팅의 확산은 가속화 되고 있다.

또한, 클라우드 컴퓨팅이 4차산업혁명의 핵심 기술 중 하나로 주목 받으면서 글로벌 클라우드 시장은 급성장[2] 추세이다. 전 세계의 기업과 정부는 이러한 패러다임의 변화에 빠르게 대응하며 클라우드 우선주의(Cloud First) 정책[3]을 넘어 클라우드 중심주의(Cloud Only) 정책[4]을 도입하기에 이르렀다. 우리나라도 클라우드 컴퓨팅 산업을 활성화하기 위해 지난 2009년 관계부처 합동으로 「범정부 클라우드 컴퓨팅 활성화 종합계획」을 수립하였고, 2015년에는 세계최초로 '클라우드 컴퓨팅 발전법[5]'을 제정하는 등 현재까지도 부단한 노력을 기울이고 있다.

□ 클라우드 컴퓨팅의 핵심 기반인 가상화 기술

클라우드 컴퓨팅 서비스가 가능하기 위해서는 반드시 가상화 기술이 수반되어야 한다. 클라우드 컴퓨팅이 처음 등장한 배경은 서비스 사업자들의 유휴(Idle) 컴퓨팅 자원의 재활용을 목적으로 시작되었다. 보유하고 있는 하드웨어 장비들을 가상화(Virtualization)를 통해서 여러 개의 장비를 묶어 사용자에게 공유자원으로 제공하며 자원의 활용성을 높이고자 한 것이 시초이다. 가상화 기술은 여러 장비를 하나로 묶을 수도 있고 반대로 하나의

1) CPU, 메모리, 네트워크, 서버, 스토리지, 애플리케이션 등 컴퓨터에서 가용한 컴퓨팅 자원을 의미하며, 이 보고서에서는 자원과 리소스는 같은 의미로 사용한다.
2) '16년 1,030억 달러에서 '21년 2,768억 달러로 연평균 21.9% 성장할 것으로 전망('18.03., IDC)
3) 정부 기관들이 클라우드 컴퓨팅을 선제적으로 도입하는 정책(미국 '10.12, 영국 '13.05.)
4) 미 트럼프 대통령은 「클라우드 Only 행정명령」으로 全 정보화의 클라우드 전환을 의무화('17.05.)
5) 「클라우드컴퓨팅 발전 및 이용자 보호에 관한 법률」('15.03.27.)(시행: '15.09.28.)

장비를 마치 여러 개의 장비인 것처럼 동작시키는 것도 가능하다.

하드웨어 장비를 가상화하면 해당 장비가 제공하는 컴퓨팅 자원의 활용도를 높일 수 있게 된다. 이를 통해, 사업자는 컴퓨팅자원 구매과 유지보수에 들어가는 비용을 절감할 수 있고, 사전에 환경을 구축하기 위한 공간 확보와 인력채용과 같은 고정비용도 절약할 수 있게 된다. 또한, 컴퓨팅 자원을 조달하는 시간을 획기적으로 단축하여 사업을 개시할 수 있고, 용량증설이 필요할 경우 자원을 요청하여 즉시 확장하는 것도 가능해 진다.

가상화 기술을 활용한 현대적 의미의 클라우드 컴퓨팅은 컴팩(Compaq)에서 1996년 용어의 등장 이후 2006년 구글 내부에서 유휴 서버를 활용하는 방식을 처음 제안하고, 같은 해 8월 아마존이 EC2[6]를 개시하면서 대중들에게 널리 알려지기 시작했다.

□ 가상화 기술의 변화와 컨테이너 기반 기술의 부상

가상화 기술은 그 방식과 형태에 따라 전통적으로 하이퍼바이저(Hypervisor)형과 호스트(Host)형으로 나눌 수 있으며, 각각 장단점이 있어 요구되는 상황에 따라 맞춰 활용되어 왔다.

최근에는 클라우드 컴퓨팅 환경을 구축하는 데에 있어서 시스템 환경에 대한 의존성이 없고, 경량화를 통한 속도 및 이식성 향상을 추구하는 컨테이너(Container) 기반의 가상화 기법이 널리 활용되고 있다. 대표적인 플랫폼으로는 공개SW 기반의 쿠버네티스(Kubernetes)[7]나 도커(Docker)[8]가 있으며, 기술과 시장측면을 선도하고 있다.

이 리포트에서는 클라우드 컴퓨팅을 위한 가상화 기술과 최근 부상하는 컨테이너 기반의 가상화 기술에 대해 살펴보고자 한다. 또한, 이런 기술들이 등장하게 된 배경과 향후 전망에 대해 논의하고자 한다.

6) 아마존 일래스틱 컴퓨트 클라우드(Amazon Elastic Compute Cloud)
7) 구글의 컨테이너 기반 가상화 플랫폼 (3장에서 기술)
8) 컨테이너 기반의 클라우드 플랫폼 이름이자 기업이름 (3장에서 기술)

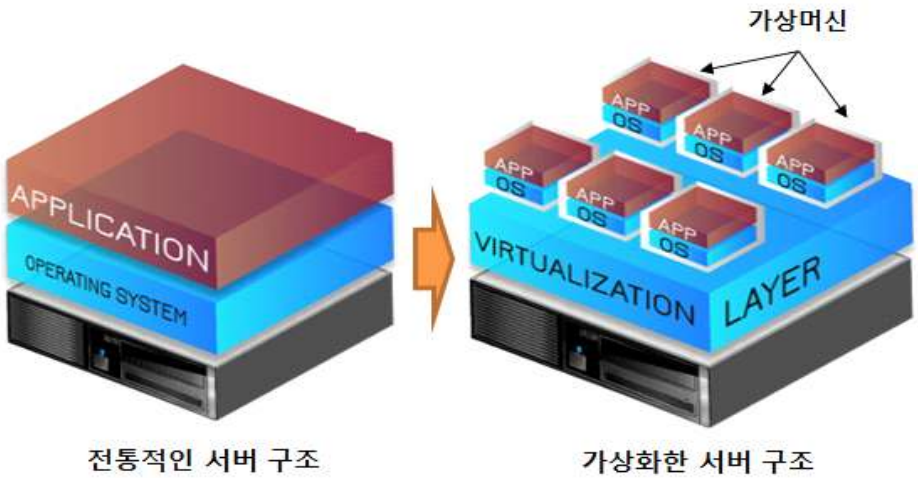
2. 클라우드 컴퓨팅의 가상화 기술

2.1. 가상화 기술의 개요

□ 가상화는 물리적인 컴퓨터 자원을 추상화 하며, 분산컴퓨팅 환경을 가능하게 함

가상화(Virtualization)는 물리적인 컴포넌트(Components, HW장치)를 논리적인 객체로 추상화 하는 것을 의미하는데, 마치 하나의 장치를 여러 개처럼 동작시키거나 반대로 여러 개의 장치를 묶어 마치 하나의 장치인 것처럼 사용자에게 공유자원으로 제공할 수 있어 클라우드 컴퓨팅 구현을 위한 핵심기술이다.

가상화의 대상이 되는 컴퓨팅 자원은 프로세서(CPU), 메모리(Memory), 스토리지(Storage), 네트워크(Network)를 포함하며, 이들로 구성된 서버나 장치들을 가상화함으로써 높은 수준의 자원 사용율과 분산 처리 능력을 제공할 수 있다.



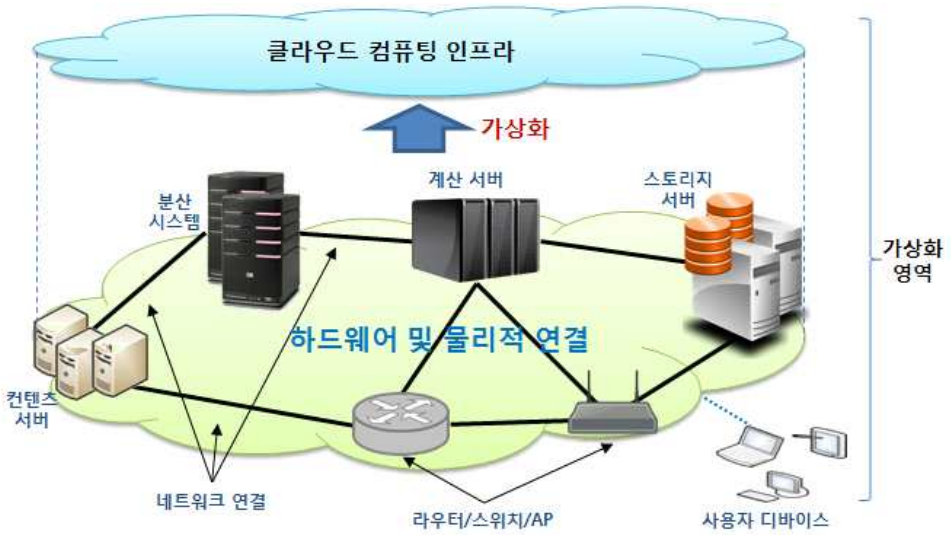
※ 출처 : VM ware (재편집)

[그림 2-1] 가상화의 기본개념(서버 가상화의 예)

일반적으로 서비스 사업자의 서버에서 제공되는 서비스는 항상 많은 양의 컴퓨팅 자원을 소모하지 않는다. 서비스의 종류와 시간, 그리고 서버가 위치한 지리적인 요인에 따라 어떤 서버는 컴퓨팅 자원이 모자란 반면, 다른 서버는 컴퓨팅 자원이 남아도는 상황이 종종 발생한다.

평균적으로 대부분의 서버는 보유한 용량의 일부인 약 10~15% 수준을 사용하는 경우가 많은데, 가상화를 통해 한 대의 서버 컴퓨터에서 동시에 여러 개의 운영체제(OS)를 가동시키고 컴퓨팅 자원이 모자란 서버의 요청 태스크(Task)를 분산처리 하면 사용률을 약 70% 이상까지 끌어올릴 수 있다.[9] 이렇게 되면 서비스 요청이 몰린 다운직전의 서버의 부담을 해소하고, 다른 서버의 남아도는 자원을 끌어다 쓰는 효과를 동시에 얻을 수 있다.

클라우드 컴퓨팅은 [그림 2-2]와 같이 기존의 하드웨어와 이들을 연결하는 네트워크로 구성된 환경을 가상화를 통해 통합된 계산, 저장 및 처리가 가능한 환경으로 제공하는 것으로 정의할 수 있다. 실제 하드웨어의 물리적인 레이어를 가상화함으로써, 데이터센터가 제공할 수 있는 다양한 기능들을 가진 '가상의 데이터센터'를 구현할 수 있다.



※ 출처 : 소프트웨어정책연구소, 클라우드 보안의 핵심이슈와 대응책, 2017.

[그림 2-2] 클라우드 컴퓨팅에서 가상화의 예

□ 가상화 개념은 1960년대부터 등장하여, 2000년 이후 상용화를 거치며 현재 많은 기업들의 비용절감 효과를 창출

가상화의 개념은 1960년대 IBM 메인프레임에서 시도[10]되면서 처음 등장하게

9) Matthew Portnoy, Virtualization-Essentials, SYBEX, 2016.
10) 제랄드 포펙(Gerald Popek)과 로버트 골드버그(Robert Goldberg)에 의해 가상화를 지원하기 위해 필요한 사항을 기술한 프레임워크에서 시작

되었으며, 1974년「가상화 가능한 3세대 아키텍처의 정규 필요사항[11]」이라는 논문을 통해 소개되었다.

컴퓨터와 인터넷 등 IT의 발전은 다수의 서버를 보유한 데이터센터의 증가를 불러왔다. 그런데 시간에 따라 서버의 성능은 무어의 법칙[12]을 따랐고, 하나의 서버에서 한 개의 어플리케이션이 동작하는 상황이 늘어나기 시작했다. 결국 하드웨어의 성능이 증가한 것에 비해서 서버 한 개당 하나의 어플리케이션을 구동하는 것은 서버의 성능을 전부 활용하지 못하는 비효율적인 상황으로 이어졌다.

기술자들은 이 문제를 가상화를 통해서 해결하고자 했다. 이미 70년대에 등장한 가상화의 개념이 점차 현대의 컴퓨터 시스템에 맞게 수정되면서 다양한 벤더(Vendor)들로부터 솔루션이 등장하기 시작했다. 첫 상용 솔루션은 2001년 발표된 VM웨어[13]라는 x86 컴퓨터에서 사용할 수 있는 솔루션이었다. 이후, 2003년에는 시트릭스의 젠(Xen)[14]이라는 병렬 오픈소스 솔루션이 등장했다.

기업들은 가상화를 통한 서버의 통합을 통해 그 수와 종류를 줄이는 것으로 상당한 비용절감 효과를 얻을 수 있다. 고가의 장비 구매비용 뿐만 아니라, 장비의 유지 및 보수, 그리고 운영에서도 큰 매리트를 가져갈 수 있다.

가상화를 통한 비용절감 효과

- **HW구매, 유지·보수 비용의 감소**
  - 기존 서버의 통합으로 추가적인 서버 증설 요구가 감소되며, 서버 개수 및 종류의 단순화를 통한 관리 용이성 증대된다.
  - 서버의 물리적인 양 감소로 데이터센터의 절전효과, 공간효율성, 냉각시설 효율성이 증대되며 쾌적한 데이터센터 운영이 가능하다.

11) Formal Requirements for Virtualizable Third Generation Architecture, 1974.
12) Moore's law, 인텔의 공동설립자 고든 무어(Gordon Moore)가 내놓은 이론으로 반도체 집적회로의 성능이 2년마다 2배씩 증가하며, 컴퓨팅 성능은 18개월 마다 2배씩 향상, 컴퓨팅 가격은 18개월마다 반으로 떨어진다는 법칙
13) VM ware, 대표적인 상용 하이퍼바이저
14) 오픈소스 기반의 최초의 반 가상화 하이퍼 바이저

- **서비스 개발의 용이성 및 운영의 유연성 보장**
  - 다양한 환경에서 동작하는 코드를 한 대의 워크스테이션에서 개발할 수 있어서 개발환경이 단순화되고 효율성이 올라간다.
  - 서비스의 통합(Clustering), 분할(Partitioning), 이동(Migration), 업무량(Workload) 관리 등 다양한 비즈니스 요구에 유연하게 대응할 수 있다.

2.2. 가상화의 종류

□ 가상화는 가상화의 대상, 방식에 따라 다양하게 구분

가상화는 가상화의 대상에 따라 서버 가상화, 데스크톱 가상화, 어플리케이션 가상화로 나눌 수 있다.

①서버 가상화(Server Virtualization)는 앞서 서술한 것처럼 서버의 효율성을 올리기 위해 등장하였으며, 가상화 개념의 시초가 되는 역할을 한 가상화이다. 가상화를 가능하게 하는 하이퍼바이저 (Hypervisor)와 이 하이퍼바이저를 통해 제어되며 각종 어플리케이션을 실행하기 위한 환경인 가상머신(Virtual Machine, VM)으로 이루어진다. 하이퍼바이저는 하드웨어로부터 제공되는 물리적인 레이어를 추상화하여 가상 머신을 통해 이 기능들을 온전하게 사용토록 한다.

[그림 2-1]의 오른쪽 「가상화한 서버 구조」에서 나타난 것처럼 하이퍼바이저는 Virtualization Layer(가상레이어)에 위치하며 하드웨어와 다수의 VM들 간의 인터페이스 역할을 한다. 하이퍼바이저와 가상머신은 이 보고서에서 주로 다루는 내용으로 다음 장에서 좀 더 자세하게 살펴보도록 하겠다.

②데스크톱 가상화(Virtual Desktop Infrastructure, VDI)는 데이터 센터의 서버에서 운영되는 가상의 PC환경을 의미한다[그림 2-3]. 물리적으로는 존재하지 않는 가상의 개별 컴퓨터로 사용자는 모니터, 키보드, 마우스, 스피커 등의 필수적인 입출력 장치만을 활용[15]하거나 매우 단순화된 인터페이스만 가지고 컴퓨터를 활용할 수 있다. 가상의 데스크톱을 마치 로컬 시스템처럼 활용할 수 있으며, 모든 작업의 프로세싱과 저장은 센터에 위치한 서버에서

15) 예로 씬 클라이언트(Thin Client)가 있다.

이루어진다.

VDI 환경에서는 언제 어디서든 네트워크가 가능하다면 서버에 접속하여 자신만의 PC환경을 구동시킬 수 있다. 사용자는 VDI를 통해 보통의 PC보다 5~10%수준의 전력소모로 유사한 컴퓨팅 환경을 보장 받을 수 있다. 데이터가 로컬 머신이 아닌 센터의 서버에 위치하여 PC의 복원, 생성 등의 작업이 쉬워진다.

또한 보안측면에서도 기존 개별 로컬 PC마다 보안솔루션을 설치했던 방식과 달리 데이터 센터 급의 보안 서비스를 보장받으며, 센터 서버의 가상머신을 모니터링하고 캡슐화 함으로써 보안성이 향상될 수 있다. 관리측면에서도 적은 종류의 가상PC 이미지로 수백 대의 가상PC를 만들 수 있으며, 이에 대한 일괄 업데이트가 가능하여 관리 효율성이 올라간다.

※ 출처 : 퓨어스토리지 (재편집)

[그림 2-3] 데스크톱 가상화의 개념

③어플리케이션 가상화(Application Virtualization)는 해당 응용프로그램이 실행되는 운영체제(OS)로부터 응용소프트웨어를 캡슐화 하는 기법이다. 이렇게 캡슐화 된 응용프로그램은 실제 설치되지는 않으나, 마치 설치된 것처럼 실행된다. 예를 들면, 마이크로소프트(MS)의 윈도우(Windows) 7에서 훨씬 이전의 OS인 윈도우 XP 모드를 제공하는데, 현재 OS의 별다른 수정 없이 구형 OS를 구동하는 것이 가능하다. 사용자가 윈도우 XP에서만 구동되는 구형 프로그램을 실행하고자 할 때 용이하게 쓰인다.

이 방식은 각 응용프로그램간의 상호작용에 문제가 발생할 수 있기에

필요한데, 한 응용프로그램을 업데이트하여 관련된 다른 응용프로그램이 동작하지 않는 문제가 발생하는 경우를 방지할 수 있다. 또한, 어플리케이션 관리가 수백~수천 개에 달하는 기업등과 같은 집단의 경우, 가상화된 어플리케이션은 관리자가 배포 및 업데이트를 할 때 상대적으로 용이할 수밖에 없다.

가상화의 대상이 되는 **④하드웨어 리소스에 따라 분류**할 수도 있는데, 하드웨어 자원에 따른 가상화를 정리하면 <표 2-1>과 같다.

<표 2-1> 가상화 대상 하드웨어 리소스에 따른 분류

구분		내용	방법
중앙처리장치(CPU) 가상화	싱글 코어	▪ 각각의 가상머신(VM)에 동적인 vCPU 할당	▪ 가상머신에 vCPU를 할당 시 물리적인 CPU(호스트 CPU)를 시분할 스케줄링 하여 동작
	멀티 코어		▪ 각각의 가상머신의 vCPU를 물리코어에 매핑하여 자원을 할당하여, 기본적으로는 코어의 수만큼 가상머신을 구동 가능 - 가상머신 벤더마다 각 물리 코어당 지원하는 vCPU의 개수가 상이(25~100개) - 하이퍼스레딩[16]을 활용할 경우 더 많은 vCPU의 할당도 가능
메모리(Memory) 가상화		▪ VM에 메모리 영역을 할당하고, 연속된 물리적 메모리가 존재하는 것처럼 인식	▪ 가상머신에 물리메모리의 특정영역을 필요한 용량만큼 할당하고 페이징(Paging), 메모리압축 등의 기법을 통해 관리
저장소(Storage) 가상화		▪ VM에 저장소를 할당하며, 직접 연결된 디스크처럼 인식	▪ 물리디스크의 특정영역을 필요한 용량만큼 할당하고 해당영역에 접근 및 읽기·쓰기 권한을 제공
네트워크(Network) 가상화 HW		▪ 컴퓨터가 여러 개의 네트워크 인터페이스를 보유한 것처럼 인식	▪ VM에 물리적인 네트워크 인터페이스(NIC)를 공유하여, 가상의 NIC을 할당
		▪ 가상네트워크(VLAN)은 물리적 네트워크에서 분리된 가상의 네트워크를 제공함으로써 유동적인 관리와 성능조율이 가능 - 특정 서비스를 제공하는 가상 네트워크를 구성하여, 사용자, 서비스 목적, 과금 체계 별로 독립된 네트워크를 제공	▪ 라우터(Router), 스위치(Switch), 터미널(Terminal) 등의 네트워크 중계기기 및 단말의 가상화를 통해 가상의 네트워크를 만들고 서로 다른 프로토콜도 공존할 수 있도록 함

16) Hyter-Threading : CPU 제조사인 인텔의 기술로, CPU내 각 코어의 연산을 병렬화 하여 두 개의 논리적 코어로 동작하게끔 하는 기술이다.

스토리지영역 네트워크 (SAN, Storage Area Network)	▪ 특정 서버나 저장소에 데이터의 저장 요구가 많아지는 경우 상대적으로 저장 공간이 남아 있는 저장소에 분산 저장 - 물리적 저장 디바이스를 가상화하며 스토리지 리소스의 가용성과 유연성을 향상 - 공유 클러스터 스토리지	▪ 하이퍼바이저에 의해 물리적 저장소를 추상화 하는 파일시스템을 제공 - VMFS(VM웨어), XFS(Xen), CSV(하이퍼-V)

2.3. 가상머신과 하이퍼바이저

□ 가상머신(Virtual Machine)은 가상화를 통하여 구현되는 복제된 컴퓨팅 환경

가상머신은 컴퓨팅 환경을 소프트웨어적으로 구현한 것을 말하는데, 보통 하드웨어를 가상화하면 [그림 2-1]에서와 같이 그 하드웨어 위에서 다수의 VM들이 생성되어 각각 독립적인 운영체제 환경을 구동한다. 시스템내의 가상머신들은 실제 가상화하기 전과 같은 수준의 실행 기능을 제공하는 것을 목적으로 하고, 가상머신들 간의 하드웨어를 공유하게 된다.

가상머신은 내부구조가 물리적인 서버의 컴퓨팅 환경과 매우 유사하다. 실제 서버처럼 CPU, 메모리, 저장소와 같은 하드웨어 자원을 활용하고, 내부에 운영체제를 구동시킬 수 있으며, 각종 응용프로그램을 구동 및 관장한다. 물리적인 서버와의 차이점은 여러 개의 가상머신이 동시에 존재할 수 있고, 각 가상머신마다 서로 다른 구동 환경을 갖출 수 있어서 다양한 어플리케이션을 수행하는 것이 가능하다는 점이다. 가상머신들은 각 머신별로 가상화된 컴퓨팅 리소스를 할당 받거나 접근(Access)한다.

가상머신은 여러 하드웨어 자원에 접근할 수 있지만 가상머신의 관점에서는 접근하는 하드웨어 디바이스가 가상이라는 것을 알지 못하며, 표준 디바이스를 다루는 것처럼 인식한다. 단, 앞서 살펴본 반가상화의 경우에는 가상머신에 탑재되어 있는 게스트OS가 하드웨어가 가상화되었다는 것을 인지하고 하이퍼바이저를 통해 디바이스에 접근한다.

가상머신을 만드는 목적은 여러 가지가 있을 수 있는데, 하나의 하드웨어위에 동시에 여러 종류의 운영체제나 프로토콜을 실행할 때, 하나의 하드웨어 자원을 여러 사용자에게 나누어 줄 때, 가상화를 통해 분할된 시스템 간 상호 간섭이 없는 독립성(Isolation)을 보장하고자 할 때 등 이 있다.

□ 하이퍼바이저(Hypervisor)는 공유 컴퓨팅 자원을 관리하고 가상머신들을 컨트롤 하는 중간관리자

하드웨어를 가상화하기 위해서는 하드웨어들을 관장할 뿐만 아니라 각각의 가상머신들을 관리할 가상머신모니터(VMM: Virtual Machine Monitor)와 같은 중간관리자가 필요하다. 이 중간관리자를 하이퍼바이저(Hyperviser)라고 하며, VM이 동작할 수 있는 환경을 제공 한다. 하이퍼바이저는 하드웨어의 물리적인 리소스를 VM들에게 제공하고, VM과 하드웨어간의 I/O[17] 명령을 처리한다.

하이퍼바이저에 요구되는 사항은 <표 2-2>와 같이 정확성, 독립성, 성능 세 가지가 있는데, 이중 앞 두 가지는 상대적으로 만족하기 쉬우나 성능 문제는 HW자원을 공유하는 관계로 만족시키기가 어렵다.

<표 2-2> 하이퍼바이저에 요구되는 세 가지 요소

요소	내용
정확성(Fidelity)	VM을 위해 만든 환경은 원래 물리적 머신과 본질적으로 동일해야 함
독립성(Isolation)과 안정성(Safety)	하이퍼바이저는 시스템 자원에 대한 완전한 제어권을 가짐
성능(Performance)	VM과 물리적 환경 간의 성능 차이가 없어야 함

※자료 : Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." Communications of the ACM 17.7 (1974): 412-421.

[그림 2-4]는 하이퍼바이저의 위치 및 역할 차이에 따라 Type1과 Type2로 구분됨을 나타낸다. Type1은 베어메탈(Bare-metal) 기반으로 하드웨어 위에서 바로 구동되며, 하이퍼바이저가 다수의 VM들을 관장하는 형태이다. 하이퍼바이저형으로도 잘 알려져 있다. 이 타입은 가상머신에 설치된 게스트 운영체제(Guest O

17) Input Output 명령

S)가 하드웨어 위에서 2번째 수준으로 구동된다. Type2 보다는 더 향상된 성능을 제공하지만, 여러 하드웨어 드라이버를 세팅해줘야 하며 설치가 어렵다.

※이미지 출처 : 안성원, 클라우드 컴퓨팅과 인공지능의 만남, IT데일리 전문가 강좌, 2017.7
소프트웨어정책연구소, 클라우드 보안의 핵심이슈와 대응책, 2017.

[그림 2-4] 하이퍼바이저의 타입

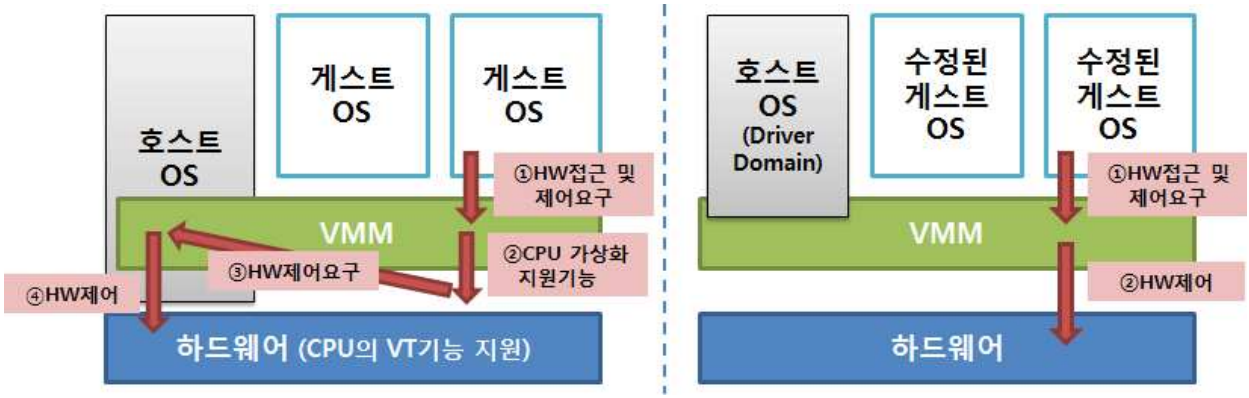
Type2는 하드웨어 위에 호스트 운영체제(Host OS)가 있고, 그 위에서 하이퍼바이저가 다른 응용프로그램과 유사한 형태로 동작한다. 이 타입의 하이퍼바이저에 의해서 관장되는 가상머신의 게스트OS는 하드웨어 위에서 3번째 수준으로 구동된다. 기존의 컴퓨터 환경에서 하이퍼바이저를 활용하는 것이기에 설치가 용이하고 구성이 편리한 장점이 있다. 반면, Type1 보다는 성능이 떨어질 수 있다.

□ 하이퍼바이저의 가상화 방식에 따라 전가상화와 반가상화로 구분

가상화는 가상화 하는 방식에 따라서도 전가상화와 반가상화로 구분할 수 있다. 주로 서버 가상화에서 이 두 개념이 등장하는데, 분류하는 기준은 하드웨어에 대한 I/O 접근을 어디까지 가상화 할 것인가에 달려있다. [그림 2-5]는 전가상화와 반가상화의 개념도를 나타낸다.

①전가상화(Full-Virtualization)는 컴퓨팅 시스템의 하드웨어 리소스를 완전하게 가상화 하는 방식으로 그 위에서 동작하게 될 게스트OS의 수정 없이 구동이 가능하다. MS 윈도우에서 리눅스 까지 다양한 OS를 사용할 수 있기

때문에 적용이 쉬운 편이다. 컴퓨팅환경을 기존의 OS위에서 에뮬레이션 하는 형식으로 지원하기에 호스트(Host)형 가상화 라고도 한다. 다만 CPU의 가상화 지원 기술(VT, Virtualization Technology)[18]과 같은 하드웨어 기능을 일부 지원받기 때문에 모든 기능을 소프트웨어적으로 구동하는 에뮬레이션과는 차이가 있다.



(a)전가상화 (b)반가상화

[그림 2-5] 전가상화와 반가상화

전가상화에서는 게스트OS가 하드웨어에 접근하기 위해 기존의 OS를 통해서 접근한다. [그림 2-5]와 같이 게스트OS에서 발생한 하드웨어 접근 및 제어 요구는 CPU의 VT가 VMM(Virtual Machine Monitor, Hypervisior)에게 HW접근을 요청하는 절차가 필요하다. 처리 단계가 늘어남에 따라 VMM(하이퍼바이저)의 부담이 가중되고 따라서 성능은 반가상화 기법보다 낮다. 전가상화의 대표적인 제품은 VMware의 VMware나 ESX Server, MS의 Hyper-V 등이 있다.

②반가상화(Para-Virtualization)는 하드웨어를 완전히 가상화하지 않는 방식이다. 하이퍼바이저가 하드웨어 위에서 직접 실행되기 때문에 네이티브(Native, Bare-metal) 가상화 라고도 한다. 반가상화는 하드웨어에 대한 제어권을 하이퍼바이저(VMM)가 가지고 있기 때문에, 하드웨어와의 I/O 처리에 있어서 전가상화보다 직접적인 루틴을 사용한다.

18) CPU에서 지원하는 가상화 기술로 Intel의 Intel-VT, AMD의 AMD-V가 있다.

하이퍼바이저는 I/O 디바이스와 직접 통신하기 위해 반드시 로우레벨의 커널드라이버를 보유해야 한다. 이를 통해서 게스트OS의 하드웨어 접근 요청을 수행한다. 게스트 OS는 자신이 직접 하드웨어를 제어하지 못하기 때문에 하이퍼바이저의 커널드라이버와 통신하기 위한 수정이 필요하다.

게스트OS에 대한 수정 때문에 OS소스코드에 대한 접근이 가능해야 하고 도입이 상대적으로 어려운 편이다. 반가상화의 대표적인 제품은 시트릭스의 Xen이 있다. Xen에서는 호스트의 역할을 하는 도메인(Driver Domain, Dom 0)이 하드웨어 및 VM들에 대한 관리를 한다.

<표 2-3> 전가상화와 반가상화 비교

구분	전가상화	반가상화
가상화 범위	▪ 하드웨어 전체 가상화	▪ 하드웨어 일부 가상화
OS 수정여부	▪ 수정 없이 사용가능 - 설치와 구성이 용이	▪ 커널 일부 수정 - 하이퍼바이저가 커널드라이버(로우레벨)를 보유해야하고, 게스트OS는 공유되는 디바이스에 접근할 수 있어야 함 - 또는, 파티셔닝(Partitioning) 된 디바이스를 각VM에 할당
하드웨어 제어	▪ 게스트OS가 직접 통제하는 것처럼 동작 - 실제로는 CPU의 VT에서 하이퍼바이저에게 의뢰하는 형태이며 호스트OS를 통해 접근	▪ 하이퍼바이저가 통제
성능	▪ CPU-VT 지원 여부에 따른 HW제약 ▪ I/O 루틴의 복잡성으로 인한 낮은 성능(RT OS 지원 불가)	▪ 단순한 I/O 루틴으로 상대적으로 고성능 유지 ▪ RT(Real Time) OS 지원 가능
주요제품	▪ VMware ESX Server	▪ XenExpress

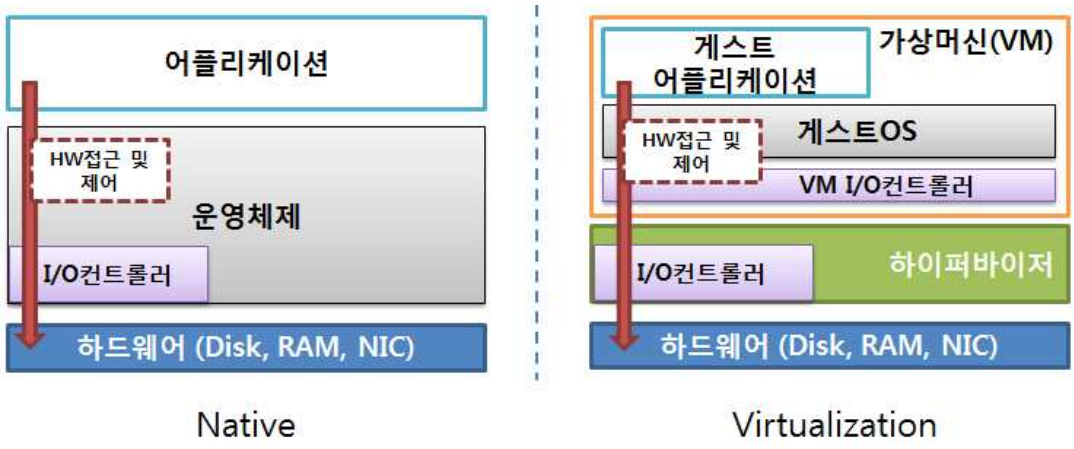
반가상화 하이퍼바이저가 I/O에 접근할 수 있는 방식에는 개별 파티셔닝(partitioning) 디바이스를 특정 VM에 배정하는 것도 있다. 이렇게 하면 각 VM은 네이티브 드라이버를 사용하여 파티션된 I/O 디바이스에 직접 접근이 가능하다. 물론 VMM에 의한 간섭이 적어지며, 그만큼 하이퍼바이저의 부담도 줄어들게 된다. 하이퍼바이저가 필요할 때는 하이퍼콜(HyperCall)을 통해 기능을 활용하고, 이 호출을 최소화함으로써 성능을 향상시킬 수도 있다.

하이퍼콜(HyperCall)

- 하이퍼바이저를 호출하는 명령어로 게스트OS가 직접서비스에 접근할 수 있는 반가상화 인터페이스
  - 운영체제에서 감시자호출(Supervisor call)을 요청하는 것과 유사하다.
    ※ 감시자호출 : 일반 어플리케이션 수준에서 할 수 없는 지정된 서비스를 실행
- 하이퍼콜은 게스트OS가 하드웨어의 가상화 여부를 알고 있어야 하며, 이 기능이 가능하도록 게스트OS의 커널 수정이 필요
  - 따라서, 오픈소스 기반의 OS가 아니라면 반가상화를 이용하기 쉽지 않다.

가상화와 성능문제

- 가상화는 기존의 하드웨어를 하이퍼바이저를 통해 직접 가상화 하거나 호스트OS상에서 소프트웨어적으로 구동하기에 성능의 하락이 발생
  - 가상머신에서 디스크 읽기/쓰기와 같은 하드웨어 접근 시 기존의 Native 시스템에 비하여 거치는 단계가 늘어나므로 명령 처리 루틴이 길어진다.
    ※ 컴퓨팅 성능의 순서 : Native > 반가상화 > 전가상화 > 에뮬레이션



[그림] Native와 가상화의 하드웨어 접근제어 루틴

  - 가상화 연구영역에서는 성능을 개선시키기 위한 다양한 연구가 수행
    ※ SR-IOV[19]를 활용한 NIC 접근제어, GPU 가상화 활용, NUMA effect[20]를 고려한 Multi-core CPU 할당 등

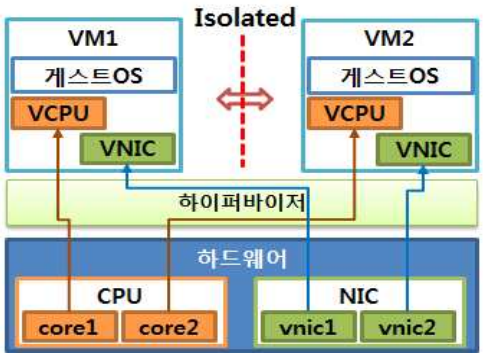
19) Single Root IO Virtualization, 인텔의 Network Interface Card에서 지원하는 기능으로 파티셔닝을 통해 가상머신마다 고유의 NIC 메모리 영역을 접근할 수 있도록 함
20) 다중 코어의 컴퓨터에서 시스템 버스를 공유하는 메모리 접근이 CPU와 메모리의 연결 위치에 따라 성능이 달라지는 현상

에뮬레이션(Emulation)

- 에뮬레이션은 컴퓨터 하드웨어 부품의 모든 기능을 소프트웨어적으로 구현하는 방식
  - 에뮬레이터는 하드웨어를 대신하여 해당 하드웨어와 같은 방식으로 동작하는 환경을 복제하여 구성한다. 즉, 응용프로그램을 구동시킬 어떤 장치가 해당 응용프로그램을 실행시킬 수 있는 장치라고 믿도록 속이는 것이다.
- 에뮬레이션의 예
  - 컴퓨터상에서 프린터 에뮬레이터를 통해 실제 프린터는 연결되어 있지 않으나, 마치 연결되어 있는 것처럼 인식하고 동작하게끔 하는 것이 있다.
  - x86 기반의 컴퓨터에서 ARM 계열의 가상머신을 띄우고 안드로이드를 구동시켜주는 QEMU[21] 또한 에뮬레이션의 일종이다.
- 에뮬레이션은 하드웨어 기능을 직접 지원받는 가상화와는 다르게, 모든 것을 소프트웨어로만 구현하기 때문에 범용성은 높으나 성능이 더 떨어짐
  - 초기의 가상머신들은 게스트의 하드웨어와 명령어를 모두 에뮬레이트 해야 했기에 속도가 매우 느렸다.

가상머신의 독립성(Isolation)

- 하이퍼바이저에 의해 구동되는 가상머신은 각 가상머신별로 독립된 가상의 자원을 할당받음
  - 가상의 자원을 할당받는다는 의미는 하드웨어의 특정 영역에 대한 접근 권한을 보장받는다는 의미이다.
    ※ 메모리의 경우 하드웨어 용량 이상을 할당하기도 하며 페이징(Paging)[22]을 통해 메모리 활용을 지원한다.



[그림] VM간의 독립성

가상머신들은 동일 하드웨어에서 구동되더라도 논리적으로 분리되어 있어서 한 VM에 오류가 발생하거나 작동이 멈추어도 다른 VM 및 시스템으로 확산되지 않는다.

21) Quick Emulator, 가상화 기능을 지원하는 리눅스 기반의 오픈소스 소프트웨어 에뮬레이터로 KVM을 적용할 수 있다.

그 외 클라우드를 위해 필요한 기술들

- **분산처리(Distributed Computing)**는 클라우드를 위한 요소기술로 여러 대의 컴퓨터 계산 및 저장능력을 이용하여 커다란 계산문제나 대용량의 데이터 저장을 해결하는 방식을 의미

  - 광의적으로는 여러 개의 컴퓨팅 디바이스를 하나의 시스템 안에 결합시킨 병렬컴퓨팅을 포함한다. 예로, 그리드컴퓨팅(Grid Computing)은 많은 계산량을 필요로 하는 작업을 위해, 인터넷상으로 분산된 자원을 공유하여 가상의 슈퍼컴퓨터처럼 활용할 수 있다.

- **네트워크(Network)**는 물리적으로 떨어져 있는 다양한 장비들을 연결하기 위한 수단으로 중계장치(라우터, 스위치 등)의 가상화를 통해 가상네트워크(Virtual Network)를 지원

  - 다양한 장비들을 네트워크로 연결하여 하나의 군집(Cluster)을 만들고 리소스를 활용한다. 예로, 네트워크컴퓨팅(Network Computing)은 응용프로그램을 서버(Server) 상에 두되 작동은 사용자(Client)의 자원을 이용하는 방식을 의미한다.

22) 한정된 메모리 용량을 관리하는 기법으로 한 번에 처리할 수 있는 적정크기(페이지) 단위로 분할하여 주기억장치(RAM)와 보조기억장치(HDD)의 페이지 스왑(swap)을 수행

3. 컨테이너 기반의 클라우드 가상화

3.1. 컨테이너의 개요

□ 컨테이너(Container)는 모듈화되고 격리된 컴퓨팅 공간 또는 컴퓨팅 환경을 의미하며, 시스템 환경 의존성을 탈피하고 안정적으로 구동

컨테이너의 사전적인 의미는 '물체를 격리하는 공간'이다. 우리가 흔히 무역관련 뉴스 등에서 접하는 컨테이너선의 그 컨테이너와 같은 의미이다. 컨테이너는 규격화 된 박스에 다양한 화물을 넣을 수 있어서 화물의 보관과 이송에 매우 최적화 되어있다. 컴퓨터 세상에서도 컨테이너는 모듈화 되고 격리된 컴퓨팅 공간 또는 컴퓨팅 환경을 의미한다. 엄밀하게는, 앞서 설명한 가상머신도 기존의 운영체제와 어플리케이션의 컨테이너라고 볼 수 있다.

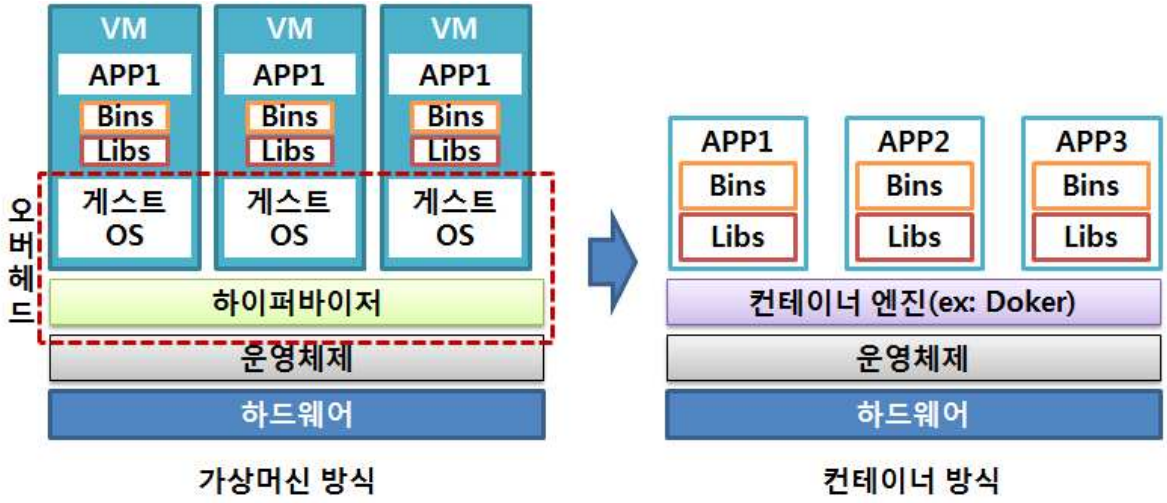
클라우드 컴퓨팅에서 컨테이너는 어플리케이션(App)과 App을 구동하는 환경을 격리한 공간을 의미한다. 가상화의 범주 내에서 컨테이너는 기존 하이퍼바이저와 게스트OS를 필요로 했던 가상머신 방식과는 달리, 프로세스를 격리하여 '모듈화된 프로그램 패키지'로써 수행하는 것을 의미한다. 이렇게 하면 기존의 가상머신에 비해 가볍고 빠르게 동작할 수 있는 장점이 있다.

컨테이너라는 개념이 처음 등장한 것은 2000년대 중반부터 리눅스에 내장된 LXC(LinuX Container)기술로 소개되면서부터이다. 컨테이너 기술이 등장하게 된 계기는 개발한 프로그램이 구동환경의 달라짐에 따라 예상하지 못한 각종 오류를 발생시키는 것을 해결하기 위함이었다. 이 문제는 SW개발자의 오랜 골칫거리였는데, 이런 오류가 발생하는 이유는 구동 환경마다 네트워크, 스토리지, 보안 등의 정책이 각각 다를 수 있기 때문이다. 결국 SW를 하나의 컴퓨팅 환경에서 다른 컴퓨팅 환경으로 이동하더라도 안정적으로 실행하는 방법을 모색하여 나온 방법이 바로 컨테이너이다.

[그림 3-1]은 컨테이너 방식의 개념을 나타낸다. 그림과 같이 어플리케이션의 실행에 필요한 라이브러리(Library, Libs)[23], 바이너리(Binary, Bins)[24], 기타

23) 프로그램 구동 시에 필요하거나 공통으로 사용할 수 있는 특정기능의 서브프로그램 또는 소스 코드

구성파일 등을 패키지로 묶어서 배포하면, 구동환경이 바뀌어도 실행에 필요한 파일이 함께 따라다니기 때문에 오류를 최소화할 수 있다.



[그림 3-1] 컨테이너의 개념

리눅스 컨테이너(LXC)

- LXC(LinuX Containers)는 단일 머신상에 여러개의 독립된 리눅스 커널 컨테이너를 실행하기 위한 OS레벨의 가상화기법으로 컨테이너 개념의 시초
  - IBM의 네임스페이스와 구글 cgroup이 결합되어 리눅스 컨테이너(LXC)가 탄생되었으며, 호스트에서 실행되는 프로세스들 사이에 벽을 만드는 기능
    ※ 네임스페이스는 리눅스 시스템 리소스들을 묶어 프로세스에 전용 할당하는 방식으로 제공되며, 하나의 프로세스의 자원을 관리하는 기능이다.
    ※ cgroup는 CPU, 메모리 등 프로세스 그룹의 시스템 리소스 사용량을 관리하여 특정 어플리케이션이 자원을 과다하게 사용하는 것을 제한할 수 있다.
  - LXC는 대부분의 코드가 GNU(LGPLv2.1+) 라이센스를 따르는 오픈소스 소프트웨어이다.
  - 프로그램 개발·실행을 위한 도구, 템플릿, 라이브러리, 프로그래밍 언어 바인딩이 세트로 구성되어 로우레벨 지원에 유연하며, 최신 커널이 지원하는 모든 컨테이너 기능을 다룸

집합을 의미한다. 라이브러리에는 주로 함수(서브루틴), 클래스에 대한 정의, 구성데이터 등 미리 작성된 코드가 포함될 수 있다.

24) 컴퓨터 저장과 처리 목적을 위해 이진수 형식으로 인코딩된 데이터 파일을 의미하며, 코드의 컴파일(Compile) 또는 압축된 결과물을 말한다.

□ 컨테이너 기술은 경량화로 인한 속도와 이식성 측면에서 각광받는 추세

최근 클라우드 컴퓨팅에서는 컨테이너기반의 가상화가 기존의 하이퍼바이저 기반의 가상화 기술을 대체하며 각광받고 있다. 예로 IT업계의 대표주자인 구글은 Gmail, Google Drive를 포함한 모든 서비스를 컨테이너로 제공한다고 발표하였으며 현재 자사의 컨테이너 플랫폼인 쿠버네티스(Kubernetes)를 통해 이미 2014년부터 매주 20억 개 이상의 컨테이너를 구동하고 있다.

그렇다면 클라우드 컴퓨팅에서 이미 널리 쓰이는 서버 가상화 기술이 있는데 왜 컨테이너가 인기를 끄는 것일까? 가장 큰 이유는 가볍기(경량화) 때문이다. 또한, 가볍기 때문에 파생되는 속도, 이식성 등의 향상효과도 있다.

컨테이너는 가상머신과는 달리 운영체제를 제외하고 어플리케이션 실행에 필요한 모든 파일을 패키징(Packaging) 한다는 점에서 'OS레벨 가상화[25]'라고도 한다. 기존의 서버에 하이퍼바이저를 설치하고, 그 위에 가상OS와 APP을 패키징한 VM을 만들어 실행하는 방식[26]인 HW레벨의 가상화와는 [그림 3-1]과 같이 게스트OS와 하이퍼바이저가 없다는 측면에서 차별성을 보인다.

컨테이너는 가상머신 방식의 가상화보다 시스템에 대한 요구사항이 적다. 먼저 컨테이너 크기가 작다. 일반적으로 컨테이너에는 OS가 포함되지 않아 크기가 수십 MB에 불과하다. 당연히 운영체제 부팅이 필요 없기 때문에 서비스를 시작하는 시간 또한 상대적으로 매우 짧다. 또한, 작은 크기 때문에 컨테이너에 대한 복제와 배포가 좀 더 용이하다.

반면, VM에는 게스트OS가 포함되므로 보통 수 GB를 넘고, 시스템 자원을 많이 소요한다. 그 이유는 각각의 가상머신이 구동하는 게스트 OS를 통하여 운영체제 구동에 필요한 하드웨어의 가상 복제본(CPU, RAM 등)을 모두 구동해야하기 때문이다. 이는 상당한 오버헤드이다.

이 점은 결국 자원에 대한 요구사항 측면으로도 이어진다. 컨테이너의 경우,

25) 운영체제 수준 가상화, 운영체제의 커널이 여러 개의 격리된 사용자 공간 인스턴스를 갖출 수 있도록 하는 가상화 방식
26) 보편적으로 서버가상화 - VMware, Xen, KVM, Hyper-V 등

생성 및 실행되면 마치 운영체제 위에서 하나의 어플리케이션이 동작하는 것과 동일한 수준의 컴퓨팅 자원을 필요로 한다. 시스템은 기존 응용프로그램을 실행시키는 것과 유사하게 이를 구동할 여분의 컴퓨팅 자원만 있으면 된다. 때문에 기존의 가상머신 방식 대비 시스템의 성능 부하가 훨씬 적다.

자원에 대한 배분도 좀 더 유연하다. 컨테이너에서 실행중인 서비스에 더 많은 가용성이 필요하거나 반대로 필요 없을 때, CPU에 대한 사용량이나 사용자가 설정한 임계치에 따라 자동으로 확장 또는 축소가 가능하다.

컨테이너는 구동 방식이 간단하다. 특정 클라우드 어플리케이션이 실행되기 위한 모든 라이브러리와 바이너리파일 등이 패키지화되어 있어서, 그저 기존의 시스템에서 실행하면 된다. 반면, 가상머신 방식은 새로운 서비스를 제공하기 위한 특정 어플리케이션을 실행시키려면, 먼저 새로운 VM을 띄우고 자원을 (-동적 또는 미리 세팅한 대로) 할당한 다음, 필요한 게스트OS를 부팅한 후 어플리케이션을 실행시켜야 한다.

이처럼 컨테이너 방식은 기존의 가상머신 방식보다 시스템이 경량화 되어 있기 때문에 더 많은 응용프로그램을 더 쉽게 하나의 물리적 서버에서 구동시키는 것이 가능하다.

<표 3-1> 가상머신 방식과 컨테이너 기반 가상화 방식의 차이

구분	기존방식의 가상머신(VM)	컨테이너 기반의 가상화
이식성	▪ VM당 모놀리딕(Monolithic)한 서비스 ▪ VM단위의 이동, 복제와 생성 가능	▪ 실행에 필요한 모든 종속성 및 구성을 함께 배포 (실행환경의 일관성) ▪ 마이크로(Micro)서비스 구축에 최적
효율성	▪ 1VM당 1서비스 ▪ 성능오버헤드 존재	▪ 호스트 OS커널 공유이므로 필요한 만큼 자원 사용
서비스 요청에 따른 신속성	▪ 최소 수 GB 이상의 추가 VM을 생성하여 대응	▪ 게스트OS가 없는 수 MB 단위의 컨테이너 생성
라이센스 비용	▪ VM개수만큼 지불	▪ Host 1대의 비용만 지불
안정성	▪ 각각 독립된 VM들로 안정적인 운영 가능(완전한 분리)	▪ 통제된 영역이지만 OS커널을 공유하므로, 장애발생시 같이 영향 받음

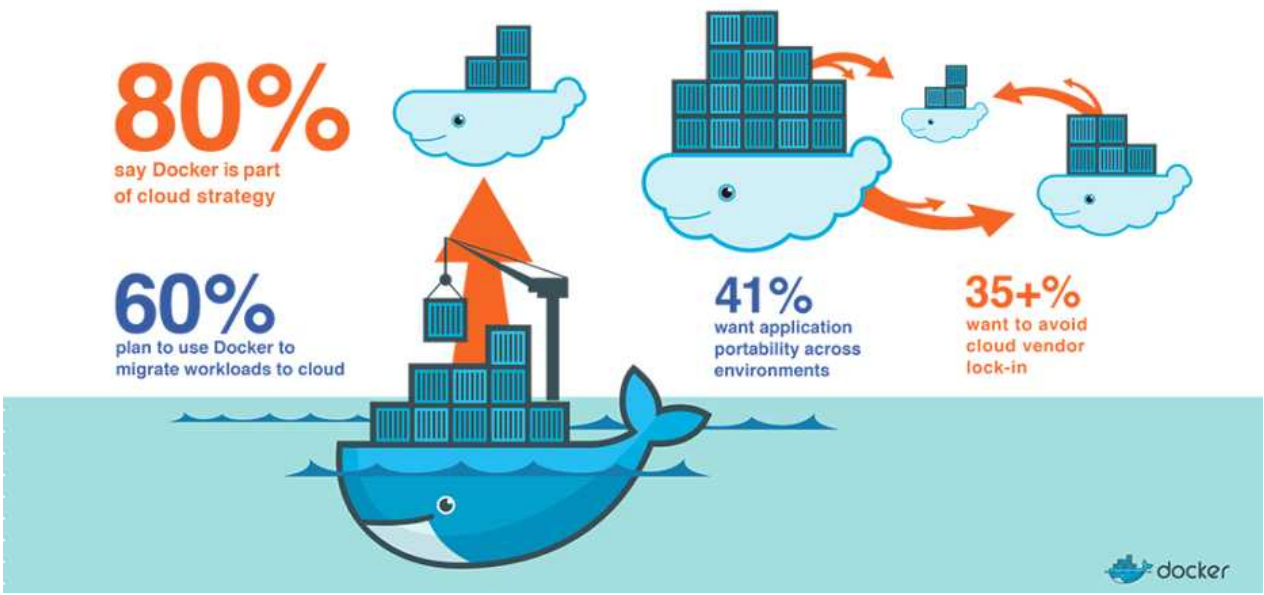
3.2. 도커의 등장과 클라우드 기술의 변화

□ 도커(Docker)는 오픈소스 기반의 컨테이너 관리 플랫폼이자 현재 컨테이너 기반 클라우드 컴퓨팅의 디펙토(de facto)

컨테이너 기반의 가상화 소프트웨어에는 OpenVZ, LXC, Lunix vServer, FreeBSD Jail, Solaris Zones, Docker 등이 있다. 그중에서 최근 가상화 및 클라우드 컴퓨팅 영역에서 가장 각광받고 있는 것이 바로 도커(Docker)이다. 도커는 앞서 살펴본 컨테이너를 관리하는 기능의 오픈소스 플랫폼이다.

도커는 2013년 3월 산타클라라에서 열린 Pycon Conference에서 dotCloud의 창업자인 솔로몬 하익스(Solomon Hykes)가 「The future of Linux Containers」라는 세션을 발표하면서 처음 세상에 알려졌다.

이후 도커가 인기를 끌면서 같은 해 10월 회사이름을 자사의 플랫폼 명과 같은 Docker로 변경하고, 2014년 6월 Docker 1.0을 발표했다. 2013년 오픈소스로 공개된 후 불과 3년 만에 서버 운영체제의 기본기술로 각광받기 시작했다. [그림 3-3]은 도커의 로고인 푸른 고래와 클라우드 활용 지표를 나타낸다.



※출처 : Docker.com, 2016.4.

[그림 3-3] 도커의 로고와 클라우드 활용 지표

도커는 리눅스의 응용프로그램들을 소프트웨어 컨테이너 안에 배치시키는 일을 자동화 하는 오픈소스 프로젝트로, 리눅스 컨테이너(LXC) 기술을 기반으로 만들었다. 기존 리눅스 컨테이너(LXC)기술에 이식성 향상, 데이터와 코드의 분산된 관리, 프로그램 스택의 간결·명료함 등 이동성과 유연성을 높이는 변화를 주었다. 기존의 시스템보다 더 쉽고 빠르게 워크로드를 배포하고 복제하고 이동할 수 있으며 백업도 가능하다.

도커의 특징은 컨테이너 이미지 생성 기능을 제공하는 것에 있다. 이는 특정 컨테이너에서 실행될 소프트웨어와 방식에 대한 '구동사양(컨테이너 실행에 필요한 파일과 설정 값 등을 포함)'을 미리 정의해 놓는 것을 의미한다. 개발자는 도커에서 지원하는 컨테이너 이미지 도구를 활용하여 어플리케이션의 이미지를 만들고, 원격으로 배포하여 실행하는 것도 가능하다. 이렇게 하면 고도의 분산 시스템을 생성하는 일이 단순해진다. 개발자가 일일이 데이터 센터의 서버를 찾아다니면서 하나씩 세팅할 필요가 없다는 뜻이다.

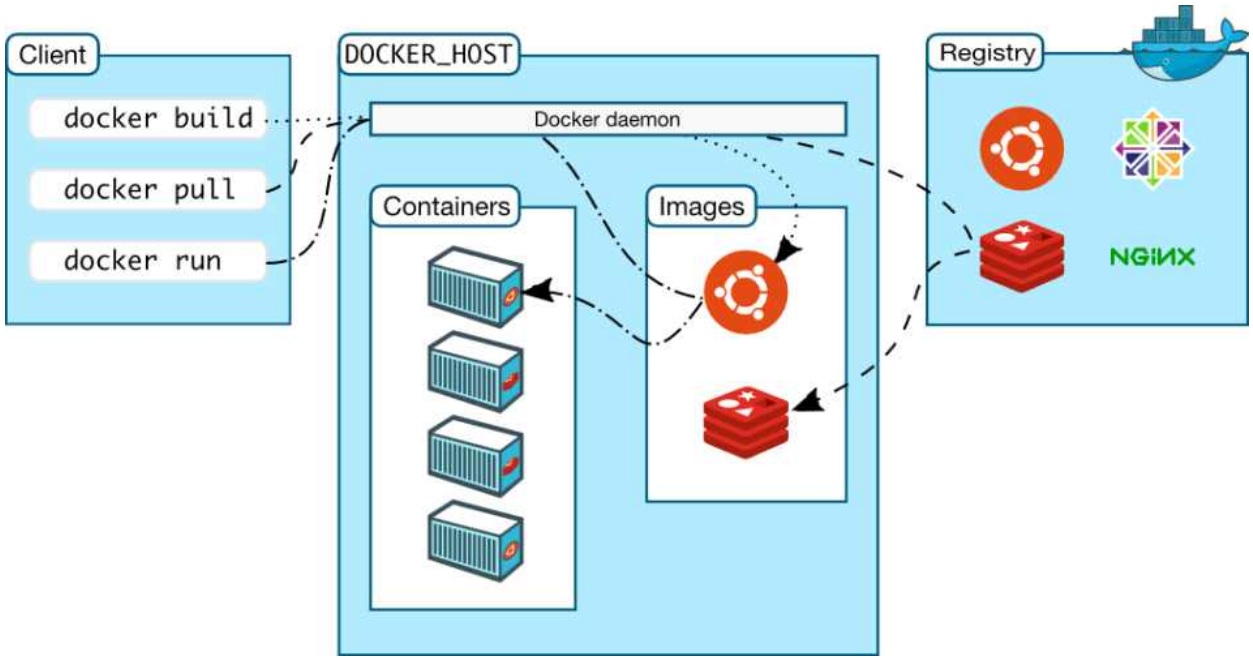
도커를 활용한 이미지는 수정이 불가한 형태로 배포되는데 이것은 단점보다는 장점으로 작용한다. 컨테이너에서 실행되는 앱과 구동하는 시스템이 분리된 형태로 있기 때문에 더 깔끔한 소프트웨어 스택 구현이 가능하다. 이미지에는 컨테이너를 실행하기 위한 모든 필요 요소가 담겨있다. 이로 인해 실행환경에 구애받지 않으며, 환경 의존성을 벗어날 수 있다.

이미지 파일은 항상 원본 상태를 유지한다. [그림 3-4]와 같이 컨테이너는 이미지를 실행한 상태인데, 같은 이미지에서 다수의 컨테이너를 생성할 수 있으며, 컨테이너의 상태가 바뀌거나 삭제되더라도 이미지는 그대로 남아있게 된다. 실행 중 추가되거나 변경된 값은 현재의 컨테이너에 저장된다. 사용자가 해당 파일을 재 구동 할 때는 원본인 이미지와 컨테이너가 가진 설정 값을 조합하여 실행된다.

결과적으로 원본이미지에 대한 중복성을 없애고 수정된 값만을 관리하여 시스템이 경량화 될 수 있다. 이것은 파일이 수정될 때마다 매번 새로운 전체 버전을 다운하거나 설치할 필요가 없다는 것을 의미한다. 도커에서는 이런

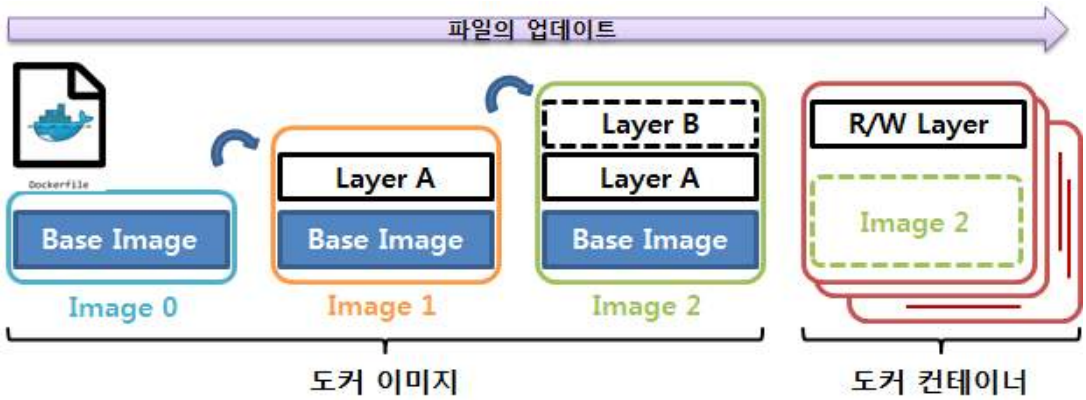
기능을 레이어(Layer)라는 개념을 통해서 지원한다.

도커의 이미지는 컨테이너를 실행하기 위한 모든 정보를 가지고 있어서 용량이 보통 수백 MB이상에 이른다. 대개는 읽기전용(Read Only) 레이어들로 구성되고, 파일이 수정 또는 추가되면 새로운 레이어가 생성된다.



※출처 : Docker.com

[그림 3-4] 도커의 구동 아키텍쳐



[그림 3-5] 도커의 Layer

[그림 3-5]의 예에서처럼 Image 0를 기본으로 하여 만든 Image 1은 Base Image + Layer A 가 되고, 다시 이를 기반으로 만든 Image 2는 Base Image +

Layer A + Layer B로 구성된다. 만일 여기서 Layer B에 해당되는 소스를 수정하고 싶다면, Image 1을 제외한 나머지 Layer B의 v2(버전 2)만 다운받으면 된다.

컨테이너를 생성할 때에도 기존의 이미지 레이어 위에 읽기/쓰기 레이어(R/W Layer)를 추가하면 된다. 컨테이너가 실행 중에 생성 및 수정하는 파일은 바로 이 R/W 레이어에 저장된다. 따라서 여러 개의 컨테이너를 실행하더라도 이미지 레이어는 그대로 사용하기 때문에 컨테이너는 최소한의 용량만을 필요로 한다. 원본이미지에 대한 관리는 [그림 3-4]와 같이 도커 허브에 등록하거나 도커 레지스트리(Docker Registry, 저장소)를 만들어 관리할 수 있다.

도커는 클라이언트-서버 모델처럼 클라이언트에서 빌딩하고 배포하고 실행하는 요청을 도커 데몬(Daemon)[27]에 요청하면서 컨테이너를 구동시킬 수 있다. 도커 클라이언트와 데몬은 같은 머신에서 동작할 수도 있고 원격으로 접속하여 동작하기도 한다.

도커 데몬은 클라이언트에서 빌딩한 요청대로 (- Docker API[28]를 통하여) 도커 레지스트리로부터 가져온 이미지(docker pull 기능)를 기반으로 컨테이너를 생성하고 원하는 서비스를 구동한다. 이때 어떻게 연결을 할 것인지, 어느 정도의 크기로 클라우드를 구성할 것인지도 함께 관리한다.

도커는 앞서 살펴본 것처럼 사실 컨테이너의 장점과 오버레이 네트워크[29]의 활용, 그리고 유니온 파일시스템[30] 등의 현존하는 기술을 잘 조합하여 쉬운 구동환경을 제공하는 플랫폼이다. 또한 프로그램을 작은 단위로 나누어 조합하는 마이크로(micro) 서비스를 지향한다. 도커를 기반으로 하는 오픈소스 프로젝트는 10만개 이상으로 진행되고 있다. 최근에는 머신러닝과 같은 인공지능 프로젝트에도 적극 활용되고 있다.

27) 데몬, 컴퓨터 시스템 운영에 관련된 작업을 background 상태로 실행해 주는 프로그램
28) Application Program Interface, OS와 APP사이의 통신에 사용되는 언어나 메시지 형식으로, 어떤 프로그램을 구동하기 위한 라이브러리에 쉽게 접근하기 위한 규칙들을 정의한 것을 말한다.
29) Overlay Network, 물리 네트워크 위에 생성하는 가상의 네트워크로 오버레이 네트워크내의 노드는 가상의 논리적인 링크로 연결된다.
30) Union File System, 읽기전용의 파일을 수정할 때 쓰기가 가능한 임시파일을 생성하고 수정이 완료되면 기존의 읽기전용 파일을 대체하는 형식의 파일시스템

3.3. 도커의 써드파티 - 쿠버네티스(Kubernetes)

□ 다중 컨테이너에 대한 효율적인 관리와 클러스터링

컨테이너는 기존의 가상머신 구동보다 경량화 되어 있어서 서버 자원을 보다 효율적으로 사용하는 것이 가능하지만, 이 역시 컨테이너의 수가 많아지게 되면 관리와 운영에 있어서 어려움이 따른다. 다수의 컨테이너(서비스)의 실행을 관리 및 조율하는 것을 컨테이너 오케스트레이션(Orchestration)이라고 한다.

특히, 매우 큰 규모의 엔터프라이즈급[31] 컨테이너 배포 및 관리에는 여러 써드 파티(3rd Party)[32] 프로젝트가 제공하고 있는데, 쿠버네티스(Kubernetes, K8S)는 구글에서 공개한 대표적인 컨테이너 관리 시스템으로 최근 가장 주목받고 있는 오케스트레이션 플랫폼이다. 구글은 이미 자사의 Gmail과 Google Drive등의 운영환경을 쿠버네티스를 활용하여 운영 중이며, 오픈소스 커뮤니티에도 최대의 코드 기여를 하고 있다.

개발자 및 사용자는 컨테이너 오케스트레이션 엔진을 통해서 컨테이너의 생성과 소멸, 시작과 중단 시점 제어, 스케줄링, 로드밸런싱, 클러스터링(그룹화) 등 컨테이너를 통한 어플리케이션을 구성하는 모든 과정을 관리하는 것이 가능하다. <표 3-2>는 오케스트레이션 엔진의 기능을 나타낸다.

<표 3-2> 오케스트레이션의 기능

기 능		내 용
서비스 디스커버리 (Service Discovery)		서비스 탐색 기능으로 기본적으로는 클라우드 환경에서 컨테이너의 생성과 배치 이동여부를 알 수 없기에 IP, Port 정보 업데이트 및 관리를 통해 서비스를 지원함
스케일링 (Scaling)	로드밸런싱 (Load Balancing)	생성된 컨테이너의 컴퓨팅자원 사용량의 설정 및 자동배분
	스케줄링 (Scheduling)	늘어난 컨테이너를 적합한 서버에 나누어 배포하고, 서버가 다운될 경우 실행 중이던 컨테이너를 다른 서버에서 구동시킴
클러스터링 (Clustering)		여러 개의 서버를 묶어 하나의 서버처럼 사용할 수 있도록 지원하거나, 가상네트워크를 이용하여 산재된 서버를 연결시켜줌
로깅/모니터링 (Logging/Monitoring)		여러 개의 서버를 동시에 관리할 경우 한 곳에서 서버 상태를 모니터링 하고 로그 관리를 할 수 있도록 함

31) Enterprise-class(grade), 대규모의 서비스망 또는 이를 구축하기 위한 시스템 환경을 의미하며 일반적으로 개별 소비자용 보다는 기업용 솔루션을 지칭한다.
32) 해당 분야에 호환되는 제품 또는 주요기업의 원천기술을 활용한 파생 제품을 생산하는 회사

앞서 살펴본 도커에도 Swarm 모드라는 자체 오케스트레이션 기능을 지원한다. 그 외에도 Apache Mesos, CoreOS fleet, AWS EC2, Redhat Cockpit, HashiCorp Nomad 등 다양한 플랫폼이 있다.

<표 3-3> 주요 오케스트레이션 플랫폼 비교

구분	구글 Kubernetes	도커 Swarm	아파치 Mesos
특징 요약	다양한 테스트를 만족하는 안정적인 솔루션	사용이 용이한 솔루션	UI 수준이 높고 기능이 풍부하나 설치 및 관리가 어려운 솔루션
운영가능 host 머신	1,000 nodes	1,000 nodes	10,000 nodes
관리 서비스	Google Container Engine	Docker Cloud, SDN	Azure Container Service (MS)
기술자료	기술 자료가 매우 풍부하고 CNCF[33]와 협력이 많아 클라우드 친화적임	기술 자료가 풍부하고 개념과 기능이 간결한편	MS와 Mesosphere가 적극적으로 지원하나 기술 자료가 부족한편
라이센스 모델	아파치	아파치	아파치

다만, 쿠버네티스는 규모가 큰 엔터프라이즈급의 컨테이너 관리에 좀 더 안정적인 환경을 제공하며, 오픈소스 기반의 클라우드 친화적인 기술 자료가 풍부하다. [그림 3-5]는 쿠버네티스의 아키텍쳐를 나타낸다.

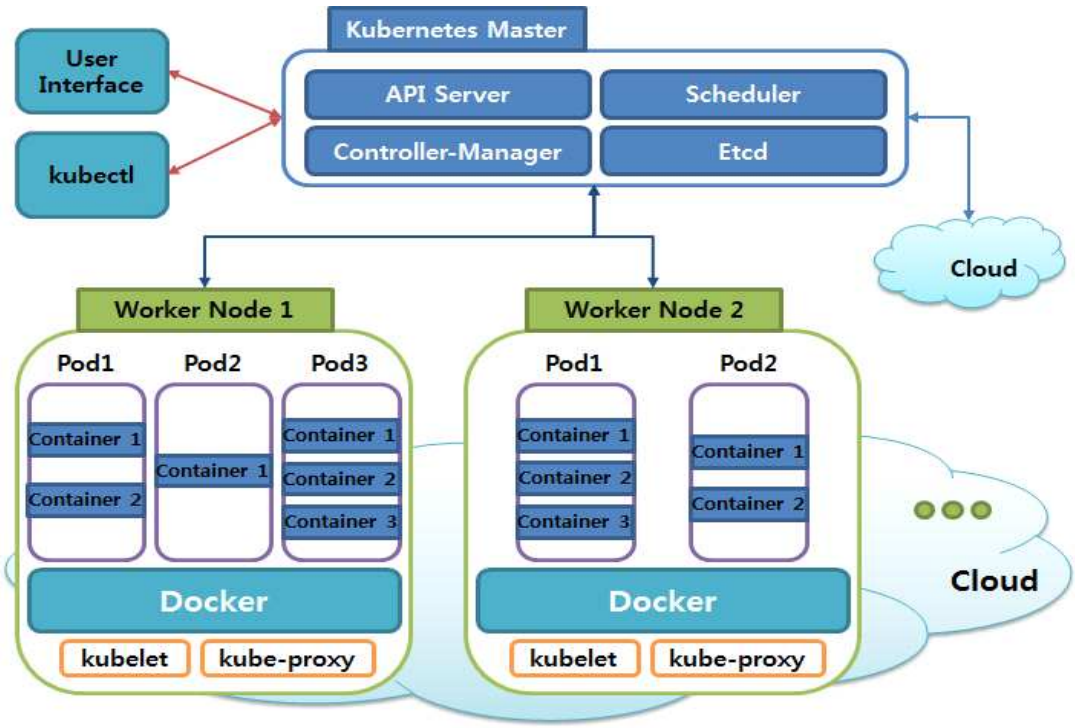
쿠버네티스는 클러스터 구조를 띄는데, 클러스터 전체를 관리하는 마스터(Kubernetes Master)가 있고, 컨테이너가 배포되는 가상 또는 물리 머신인 노드(Worker Node)가 존재한다.

쿠버네티스에 의해서 배포 및 관리되는 컨테이너들은 포드(Pod)라는 단위로 묶여서 관리된다. Pod는 하나 이상의 컨테이너를 포함하고 있고, 같은 Pod내에 속해있는 컨테이너들은 서로 로컬 통신이 가능하며, 디스크 자원도 공유한다. 이렇게 하면 서비스에 따른 컴퓨팅 파워의 스케일링(Scaling)이 쉬워진다.

마스터는 쿠버네티스의 설정 환경을 저장하고 노드로 이루어진 클러스터

33) CNCF: Cloud Native Computing Foundation, 구글이 쿠버네티스 v1.0을 출시하며 리눅스 재단과 제휴를 맺고 설립한 클라우드 컴퓨팅 재단

전체를 관리하며, Kubectl 커맨드 인터페이스를 통해서 세팅될 수 있다. 마스터에는 API 서버, 스케쥴러, 컨트롤러 매니저, etcd 로 구성되어 있다. API 서버는 유저(User Interface, UI)로 부터의 요청이나, 마스터-노드간의 통신을 담당한다. Etcd는 클러스터의 DB(Data Base)역할을 하는데, 노드 및 클러스터의 설정 값과 상태를 저장한다.



※참고 : Docker.com

[그림 3-5] 쿠버네티스의 클라우드 아키텍쳐

스케쥴러는 Pod의 배포 및 서비스에 필요한 리소스 할당 시 적절한 노드에 할당하는 역할을 한다. 컨트롤러 매니저는 Pod의 볼륨(Volum, 저장 공간)조절, 동적으로 추가 또는 삭제되는 Pod에 대한 라벨(Label) 할당, 복제, 여러 Pod의 그룹핑 서비스 시 로드밸런싱 등을 관리한다.

노드는 마스터에 의해 명령을 받고 실제 작업을 수행하는 서비스 컴포넌트로 컨테이너를 보유한 Pod들이 배치된다. Pod는 노드 내에서는 도커 플랫폼과 맞물려 자원을 할당 받아 구동된다. Kubelet는 마스터의 API 서버와 통신을 담당하며 수행할 명령을 받거나 노드의 상태를 마스터로 전달하는 역할을 한다.

Kube-proxy는 노드에 들어오는 네트워크 트래픽을 Pod내의 컨테이너에게 라우팅하고 노드와 마스터간의 네트워크 통신을 관리한다.

쿠버네티스는 다중 머신 환경(클라우드)에서 사용자가 요청한 컨테이너를 어느 머신의 호스트에 설치하는지, 어떤 머신이 컴퓨팅 자원에 여유가 있는지를 판단하여 최적의 서비스 상태를 유지하려 한다.

쿠버네티스의 장점 중 하나가 바로 이와 관련된 Fault-tolerance[34] 기능인데, 기존의 서비스가 긴급점검과 같은 서비스 중단 상태를 보여 왔다면, 쿠버네티스는 점진적인 수시 업데이트를 통해 서비스 중단 없이 서버를 업데이트 할 수 있다. 또한, 특정 컨테이너에 장애가 발생하더라도 바로 복제 컨테이너를 생성해 서비스를 유지할 수 있다.

클라우드 사용자가 서비스 공급자(Vendor)를 변경하고자 할 때, 기존에는 서비스 제품 또는 인프라간의 호환 문제로 서비스 이전이 어려웠다. 이러한 현상을 특정업체에 종속되는 문제 즉, 벤더 종속성(Vendor Lock In) 이라고 한다. 쿠버네티스는 가장 널리 쓰이는 도커 컨테이너를 기반으로 하는 오픈소스 플랫폼이어서 클라우드 이전에 좀 더 자유롭다.

실제로 쿠버네티스는 컨테이너 기반의 가상화에서 오케스트레이션 플랫폼의 표준처럼 여겨지고 있으며, 도커 엔터프라이즈 에디션에 번들로 제공된다. 쿠버네티스는 2018년 12월 초 현재 기준으로 v1.13까지 출시되었다. v1.5부터 윈도우도 지원한다.

34) 무중단 서비스, 시스템 일부에 고장이 발생하더라도 기능의 일부 또는 전체를 유지하는 것.

4. 시사점

4.1. 클라우드 기술의 변혁을 가져온 DevOps

□ DevOps는 소프트웨어 개발과 정보기술 전문 운영자 간의 소통과 협업을 강조하는 개발 환경을 의미하며 컨테이너 기반 클라우드의 초석

DevOps는 소프트웨어 개발(Development)과 운영(Operations)의 합성어이다. DevOps는 SW개발조직과 운영조직간 상호 긴밀한 소통과 협력, 대응을 의미하고, SW제품과 서비스를 빠른 시간에 개발 및 배포하는 것을 목적으로 한다. 일반적으로 프로그램, 게임, 컨텐츠 등과 같은 시스템을 개발하여 출시한 이후에는 반드시 '운영'이라는 과제가 남아있다.

시스템을 직접 개발했기 때문에 시스템에 대하여 잘 알고 있는 개발자가 운영까지 한다면 더할 나위 없이 좋겠지만, 그러기에는 오픈 후에 쏟아지는 다양한 문제들(업데이트, 보안문제, 버그수정, 안정성확보, 확장성이슈 등)을 처리하기에도 역부족인 상황에 놓이게 된다.

또한 시스템의 개발과 운영에는 기본철학에 괴리가 있는데, 이를테면 개발측은 시스템에 문제가 발생했을 때 빠른 수정을 원하는 반면, 운영측은 안정적인 서비스가 목적으로 업데이트로 인한 변화와 혹시 발생할 수 있는 불안정한 문제(서버다운, 지연현상, 버그 등)와 같은 리스크를 기피하고자 한다.

그러나 클라우드의 등장으로 인해 이러한 개발 및 운영 환경에 변화가 생겼다. 기존에는 서버의 구축과 관리가 어려웠다. 우선 고가의 서버를 구매해야하고, 각 서버마다 운영체제를 설치해야하며, 해당 서버가 있는 데이터센터(IDC: Internet Data Center)의 환경에 맞게끔 일일이 서버를 세팅해야만 했다. 각 서버별로 상이한 HW를 사용하게 될 경우 세팅이 조금씩 다를 수밖에 없고, 이러한 세팅으로 인해 서비스는 미묘한 불안정성을 가지게 된다.

클라우드의 등장으로 서버의 세팅과 관리가 자유로워짐에 따라 개발과

운영의 경계는 허물어 졌다. 결국, 클라우드로 인해 DevOps가 구체적으로 실현될 수 있었고, 이것이 컨테이너 기반의 효율적인 시스템 출현을 촉발시켰으며, 이것은 다시 기존의 클라우드를 과거보다 진일보 시키는 결과를 만들었다.

4.2. 클라우드 컴퓨팅 생태계의 주도권 전망

□ 컨테이너 기술은 클라우드 컴퓨팅과 동반성장하며 확산을 가속화

컨테이너 기반의 클라우드 컴퓨팅 환경 구성은 기존의 시스템보다 경량화를 추구하며, 그에 따른 속도향상, 마이크로서비스의 용이함, 컨테이너의 이동성, 서비스의 확산력, 스케일링 향상, 유연한 결합성이라는 효과까지 제공한다.

클라우드 컴퓨팅은 세계적인 IT 트렌드의 변화와 함께, 서비스 벤더나 사용자들에게 이미 당연한 시스템이 되어버렸고, 이 시스템을 발전시키고 있는 컨테이너 기반의 다양한 기술들은 사실상의 표준이 되었다. 국내에서도 컨테이너 기반의 서비스들을 개발하거나 플랫폼을 운영하는 사례가 통신사업자나, SW기업 및 스타트업을 중심으로 늘어나고 있으며 앞으로도 더 증가할 것으로 보인다.

특히, 구글과 리눅스 재단이 만든 클라우드 컴퓨팅 재단(CNCF)의 지원과 오픈소스 기반이라는 장점은 오픈소스 커뮤니티에서 클라우드 컴퓨팅 관련 수많은 프로젝트에 기여하고 있다. 이는 컨테이너 시장의 주도권을 가지면서 후발 주자들의 플랫폼 오픈 소스화를 종용하고 있기도 하다.

2017년 오픈스택(OpenStack) 클라우드 서비스 사업자인 미란티스는 자사의 플랫폼에 쿠버네티스를 통합했고(4월), MS는 쿠버네티스 기반의 컨테이너 배포 플랫폼 개발사(다이스)를 인수하고 Azure에 쿠버네티스 기반 서비스를 통합했다. 이후, IBM도 자사의 클라우드 컨테이너 서비스에서 쿠버네티스를 지원하기 시작했으며(5월), 오라클도 자사의 클라우드 솔루션 코어OS에서 쿠버네티스를 지원할 것이라고 밝혔다(6월). 그러자 클라우드 서비스 업체인 피보탈도 쿠버네티스를 지원하는 피보탈 컨테이너 서비스(PKS) 개발을 발표했고,

아마존도 CNCF에 가입하면서(8월) 적극적으로 코드를 기여하고 있다.

그 후로 현재까지 앞서 언급한 벤더들은 합작한 새 버전의 컨테이너 서비스들을 내놓고 있다.

□ 구글의 시장 주도는 앞으로도 지속될 것

쿠버네티스의 공개와 클라우드 컴퓨팅 재단 설립 등의 적극적인 행보로 구글은 컨테이너 오케스트레이션 시장의 주도권을 장악하고, 클라우드 컴퓨팅 생태계를 선도하고 있다. 구글의 쿠버네티스는 컨테이너 기반의 가상화 기술 개발 뿐 아니라 효과적인 운용 방식에도 표준이 되었다.

사실 구글이 리눅스 재단과의 합작으로 CNCF를 설립한 것은 도커가 리눅스재단과 OCI[35]를 설립한 직후이다[36]. 구글의 이같은 행보는 리눅스 생태계의 대표격인 레드햇의 오픈시프트(OpenShift) 플랫폼에서도 쿠버네티스를 지원하도록 했고, 이로써 더 많은 컨트리뷰터(기여자)의 지원을 받고 있다.

앞서 도커가 컨테이너 기반의 클라우드 가상화 기술의 사실상의 표준이라고 했고 쿠버네티스가 이를 적극 지원하는 형태를 띄고 있다고 했지만, 도커를 거치지 않고 곧바로 컨테이너와 연결되는 바이패스(Bypass)형 인터페이스도 개발되고 있다. 전통적인 가상머신 진영인 오픈스택 재단은 쿠버네티스가 기본으로 탑재된 하이퍼바이저 기반의 컨테이너 환경인 카타(Kata)를 통해 도커와의 경쟁을 시작했다.

도커의 인기는 좀 더 지켜봐야할 일이지만, 구글의 쿠버네티스는 적용되지 않은 플랫폼을 찾아보기가 힘들 정도로 널리 퍼져있다.

4.3. 컨테이너 기술의 전망

□ 한계점도 분명하여 이를 해결하고자 하는 기술개발도 심화될 것

35) Open Container Initiative, 컨테이너 표준과 설계명세서를 수립하기 위한 비영리 단체, ‘15.06.22.설립.
36) ‘15.07.21.에 쿠버네티스 v1.0 출시와 함께 CNCF 설립.

컨테이너 기술은 기존의 가상머신 형태의 서버 가상화가 가졌던 많은 문제를 해결했다. 사용자가 원하는 만큼 수 십대 이상의 서버 컴퓨팅 능력을 비교적 간단한 UI를 통하여 대여할 수 있고, 부하에 따라 자동으로 서버의 개수를 늘이고 줄이는 것을 더 쉽게 해주었다.

그러나 이러한 컨테이너 기술도 기존의 방법보다 클라우드를 구성하는 방법에 있어서 효율적인 것은 맞지만 단점도 가지고 있다. 예컨대 가상머신의 장점인 고수준의 프로세스 분리 기능은 곧 컨테이너 기술의 단점이 된다.

가상머신에서 동작하는 게스트OS는 호스트의 OS와 동일할 필요가 없다. 이는 가상머신마다 서로 다른 독립적인 운영체제 환경을 구축할 수 있단 의미이고, 하나의 머신기준으로 제공할 수 있는 서비스의 다양성 측면에서는 좀 더 유리하다고 볼 수 있다. 또한, 가상머신간의 독립성(Isolation)이 보장되어 한 가상머신의 장애가 다른 가상머신이나 시스템 전체에 영향을 미치지 않는다.

반면, 컨테이너는 호스트OS의 통제된 영역을 사용하지만, 많은 컨테이너들이 동일한 운영체제 커널을 공유한다. 컨테이너들은 가상머신처럼 더 철저하게 분리되어 있지 않기 때문에 보안이나 안정성 측면에서 문제가 발생할 수도 있다. 컨테이너 기반의 도커나 쿠버네티스에서도 Fault Tolerance를 보장하는 안전장치가 있고 대부분의 워크로드에서 충분한 분리성을 제공하지만 이것도 어디까지나 동일 호스트 위에서 이루어지기 때문에, 잘못된 스케줄링이나 예기치 못한 컨테이너간의 충돌 등으로 시스템 장애를 유발하는 시나리오도 가능하다.

컨테이너는 기본적으로 비 저장성을 갖는 이미지로부터 실행된다. 이 점은 시스템을 간결하게 하고 경량화 하는 장점으로 작용했지만 작업 세션의 영속성(Persistence)측면에서는 단점이 된다. 가상머신은 자체 파일시스템을 가지고 있어서 기본적으로 세션에 대한 영속성을 가지고 있다. 컨테이너는 새 인스턴스 (프로세스)를 실행하면 컨테이너가 사라지고 재시작 되는데, 이렇게 생성된 신규 컨테이너는 이전 컨테이너와의 연결된 상태 정보가 없다. 따라서 영속성을 위해서는 이전의 세션 작업을 포함하는 이미지의 새로운 생성이

필요하다. 이 부분은 컨테이너와 연결된 별도의 DB나 독립적인 스토리지가 있어야 하므로 가상머신보다는 불리하다.

성능측면의 문제도 남아있다. 컨테이너 기반의 가상화가 기존 VM 기반의 가상화 보다 구동상에 오버헤드가 적은 것은 분명하지만, 네이티브 수준의 속도를 제공하진 않는다. 사실 성능 문제는 SW를 활용하는 모든 영역의 끝나지 않는 숙제이다. 가상화 영역 또한 오랜 숙원과제였으며 클라우드 컴퓨팅과 같은 주요 IT트렌드와 맞물려 그 중요성과 관심이 매우 높아 다양한 연구주제들이 남아 있다. 예로 최근에는 GPGPU[37]를 가상화 환경에서 활용하여 성능을 올리고자 하는 연구들도 진행되고 있다.

□ 컨테이너 기술은 하이퍼바이저의 잠정적 대체제인가?

많은 전문가들은 컨테이너가 안정적이면서 확장성이 좋은 인프라 수요의 증가에 맞춰서 더 확산할 것으로 예상하고 있다. 그 대표적인 예로, 많은 사람들이 동시에 사용하는 포털 커뮤니티나 메일 서비스와 같은 인터넷 어플리케이션 부문에 컨테이너 기술이 많이 활용되고 있다는 점을 들 수 있다. 또한, 최근 프라이빗이나 퍼블릭 클라우드 인프라가 복잡해지는 것도 컨테이너의 확산을 부추기는 원인으로 여겨진다.

다만 컨테이너 방식이 확산 될 것임에는 이견이 없으나, 적어도 당분간은 컨테이너가 하이퍼바이저를 대체하기에는 한계가 있을 것이라는 시각이 지배적이다. 이미 앞에서 언급하였지만 하이퍼바이저가 관리하는 가상머신 형태의 가상화 기법도 컨테이너 방식보다 보안과 안정성 등과 같은 여러 측면에서 분명 더 큰 장점이 있기 때문이다.

따라서 컨테이너 방식이 최근 각광을 받고 있긴 하나 서버 가상화 기술과 서로의 약점을 보완하면서 공존할 것으로 전망된다. 또한, 하이퍼바이저 진영에서도 컨테이너의 장점을 받아들이며 진화한 버전의 가상화 기법들을 시도하고 있어서 하이브리드 형태의 새로운 기술이 등장할 수도 있다고 본다.

37) General-Purpose computing on Graphics Processing Unit, 다수의 병렬프로세서로 이미지 처리에 특화된 GPU를 전통적으로 CPU가 맡았던 응용프로그램 계산에 활용하는 기술

[참고자료]

1. 국내

[1] Redhat, Linux 컨테이너 : https://www.redhat.com/ko/topics/containers/whats-a-linux-container

[2] Redhat, 가상화 이해 : https://www.redhat.com/ko/topics/virtualization

[3] Redhat, 클라우드 컴퓨팅 이해 : https://www.redhat.com/ko/topics/cloud

[4] 아마존AWS-컨테이너란 무엇입니까?, 2018. : https://aws.amazon.com/ko/what-are-containers/

[5] Kubernetes 튜토리얼 : https://kubernetes.io/ko/docs/tutorials/kubernetes-basics/

[6] IT월드, 도커와 도커 컨테이너의 이해, 2018.09.

[7] KCERN, 한국의 클라우드 전략, 2018.09

[8] 지디넷, 클라우드 전환기, IBM의 승부수 '컨테이너', 2018.05.

[9] IBM Developer, IBM Cloud Private 소개, 2018.03.

[10] 지디넷, 2018년 가장 뜰 소프트웨어는?, 2018.01.

[11] IBM Developer, 쿠버네티스와 컨테이너를 쉽게 이해하기, 2018.01.

[12] IDG, 데브옵스부터 컨테이너까지, 'IT 속도전' 시대의 플랫폼 신기술 도입 전략, 2017.

[13] Mantech, IT 트렌드-쿠버네티스(Kubernetes), 2017.

[14] IT월드, "왜 도커인가?" 도커와 리눅스 컨테이너의 이해, 2017.

[15] 매튜 포트노이, 가상화 세상 속으로, 에이콘출판사, 2016.

[16] IDG, 컨테이너에 대한 6가지 오해, 2016.

[17] 정보통신산업진흥원, 클라우드산업실태조사, 2015.

[18] 오픈스택 커뮤니티, 오픈소스 클라우드 기술의 진화: 가상화와 컨테이너 기술, KRnet, 2015.

[19] CIO, 핵심 오픈소스 SW재단 8곳, 그리고 그들이 중요한 이유, 2015.

[20] 정보통신산업진흥원, 클라우드컴퓨팅 발전 및 이용자 보호에 관한 법률 설명자료, 2013.

2. 국외

[1] Cloud Native Computing Foundation: https://www.cncf.io/

[2] Kubernetes, https://kubernetes.io/

[3] Kubernetes Concepts, https://kubernetes.io/docs/concepts/

[4] Kubernetes Architecture, https://kubernetes.io/docs/concepts/architecture/node

[5] Docker, https://www.docker.com/

[6] Docker Blog, https://blog.docker.com/

[7] Docker, What is a Container - A standardized unit of software, : https://www.docker.com/resources/what-container

[8] https://docker-curriculum.com/

[9] Opensource.com, What is Docker? : https://opensource.com/resources/what-docker

[10] IBM Cloud private documentation: https://www.ibm.com/support/knowledgecenter/SSBS6K_1.2.0

[11] Twistlock, Container Technology Chapter 1 | Docker, VM, LXC & Container Basics : https://www.twistlock.com/resources/container-basics- whitepaper-chapter-1/

[12] Twistlock, Docker Basics Whitepaper | Chapter 2 : https://www.twistlock.com/resources/docker-basics-whitepaper-chapter-2/

[13] VMblog, 2017 Predictions: It's not "containers or virtualization." It's "containers AND virtualization.", 2017.

[14] Twistlock, Containers 101 Infographic: What Are Containers, What's The Difference Between Containers & VMs & More, 2016. : https:// www.twistlock.com/2016/06/23/containers-101-infographic/

[15] Wired, Google Open Sources Its Secret Weapon in Cloud Computing, 2015.

[16] Wired, Google Made Its Secret Blueprint Public to Boost Its Cloud, 2015.

[17] Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." Communications of the ACM 17.7 (1974): 412-421.

주 의

1. 이 보고서는 소프트웨어정책연구소에서 수행한 연구보고서입니다.
2. 이 보고서의 내용을 발표할 때에는 반드시 소프트웨어정책연구소에서 수행한 연구결과임을 밝혀야 합니다.