클라우드 서비스, 도커(Doker), 그리고 도쿠(Dokku)

양 단 희*

◈ 목 차 ◈

1. 서 론

클라우드하면 가상화이다. 클라우드의 본질적인 가치는 개발자가 애플리케이션을 개발할 때 신경 써야 할 위치를 선택이 가능한 옵션으로 만들었다는 것이다. AWS가 나오기 전에는 그런 선택권이 존재하지 않았다. 클라우드 서비스로서 AmazonWS EC2가 IaaS라면 헤로쿠나 구글 앱 엔진은 대표적인 PaaS 서비스이다. 더구나 Heroku는 AWS 상에서 운영되고 있다.

도커는 2013년에 등장한 새로운 컨테이너 기반 가상화 도구이다. 도커는 컨테이너의 특정 상태를 항상 보존해두고, 도커가 설치되어 있으면 때 언제 어디서나 이를 실행할 수 있도록 도와주는 도구이다. 도커 컨테이너에 버전 관리 시스템인 Git을 설치하면 도커는 마치 자신이 VCS인 것처럼 컨테이너 이미지 간의 변경사항을 확인할 수 있는 명령어를 제공한다. 즉 git diff 명령어로 프로젝트의 변경사항을 확인하듯이, 도커 diff 명령어를 사용할 수 있다[1,3].

도커의 활용도는 다양하다. 실제 배포에도 사용할 수 있고, 실험적인 개발을 진행할 수도 있다. Vagrant보다도 훨씬 빠른 가상 환경을 활용할 수 있으며 서버환경을 자동적으로 구성할 수도 있다. 미리 만든 이미지를 자신의 저장소(registry)에 등록해 여러 대의 컴퓨터에 컨테이너들을 자동적으로 배포할 수도 있고, 오픈소스 프로젝트에서 dokerfile을 제공해 다양한 설치방법을 지원할 수도 있다.

본고에서는 클라우드 서비스를 간단히 분류해 보고, PaaS와 연계되어 도커에 대한 자세한 이해를 도모한다. 그리고 도커를 활용해 간단히 애플리케이션 배포 서버를 구축할 수 있게 도와주는 도쿠(Dokku)에 대해 간략히 언급하겠다.

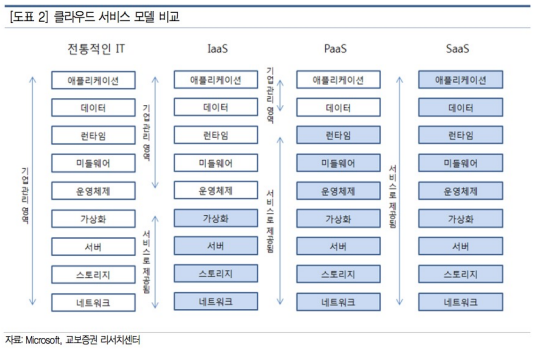
2. 클라우드 서비스(IaaS,PaaS,SaaS)

(그림 1)처럼 초기 클라우드 서비스는 지메일이나 네이버 클라우드처럼 소프트웨어를 웹에서 쓸 수 있는 SaaS(Software as a Service, 서비스로서의 소프트웨어)가 대부분이었다. 그러다가 서버와 스토리지, 네트워크 같은 컴퓨팅 인프라 장비를 빌려주는 IaaS(Infrastracture as a Service, 서비스로서의 인프라스트럭처), 플랫폼을 빌려주는 PaaS(Platform as a Service, 서비스로서의 플랫폼)로 늘어났다. 클라우드 서비스는 어떤 자원을 제공하느냐에 따라 이처럼 크게 3가지로 나뉜다.

직관적으로 얘기하면 SaaS는 웹메일이나 포털 검색 서비스처럼 일반사용자가 클라우드에서 소프트웨어를 빌려 쓰는 것이고, PaaS는 개발자가 클라우드에서 특정 운영체제와 API를 빌려 쓰는 것이고, IaaS는 시스템 운영자가 클라우드에서 서버와 스토리지 등 하드웨어 장비를 빌려 쓰는 것이다. 즉 소프트웨어와 하드웨어 자원을 필요한 기간 동안 빌려 쓰는 개념이기 때문에, 그 서비스를 이용하는 사람에게는 구입비 부담, 그리고 설치

* 평택대학교 융합소프트웨어학과 교수

와 유지보수의 복잡함으로부터 벗어날 수 있게 해주기 때문에 매우 획기적인 서비스가 아닐 수 없다.



(그림 1) 클라우드 서비스 모델 비교[2]

3. 도커(Doker)

가상 머신은 물리적인 환경으로부터 격리된 추상적인 환경을 구축해 주지만 배포용으로 사용할 때 성능 저하를 수반할 수밖에 없다. 특히 한 운영체제 위에 또 다른 운영체제를 돌린다는 것 자체가 자원을 비효율적으로 사용할 수밖에 없다.

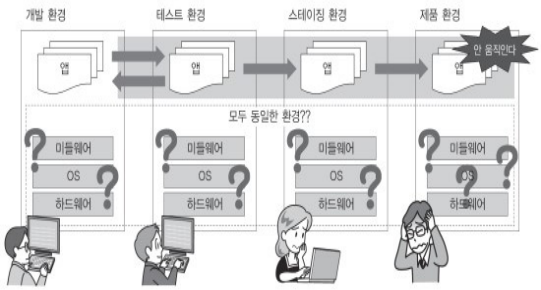
도커는 이런 가상 머신의 단점은 극복하고 장점만을 극대화하기 위한 가상화 애플리케이션이다. 도커는 단순한 가상 머신을 넘어서 어느 플랫폼에서나 재현 가능한 애플리케이션 컨테이너를 만들어주는 걸 목표로 한다. LXC(리눅스 컨테이너)라는 독특한 개념에서 출발하는 도커의 가상화는 기존에 운영체제를 통째로 가상화하는 것과는 접근 자체를 달리한다.

3.1. 개념

웹 시스템 개발 시 애플리케이션을 제품 환경에서 가동시키기 위해서는 다음과 같은 요소가 필요하다.

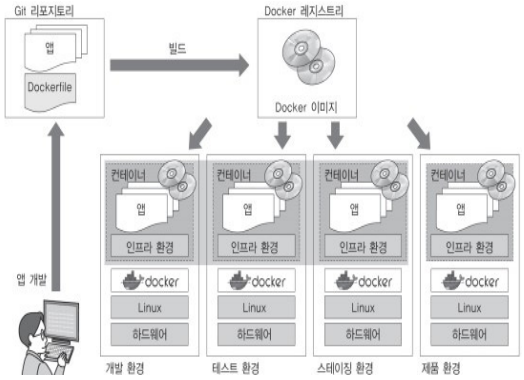
- 애플리케이션의 실행 모듈(프로그램 본체)
- 미들웨어나 라이브러리군
- OS/네트워크 등과 같은 인프라 환경 설정

애플리케이션 개발은 (그림 2)와 같은 흐름으로 진행되는데 '개발 환경'에서 잘 동작했던 것이 오른쪽 환경으로 이동되면 문제가 발생하는 경우가 허다하다. 이런 문제점을 해결하기 위해 도커에서는 애플리케이션의 실행에 필요한 모든 파일 및 디렉터리들을 컨테이너로서 모아 놓고 이 컨테이너를 관리한다. 그리고 이 컨테이너의 바탕이 되는 도커 이미지를 도커 Hub와 같은 리포지토리(repository)에서 공유한다.



(그림 2) 일반적인 시스템 개발 흐름[3]

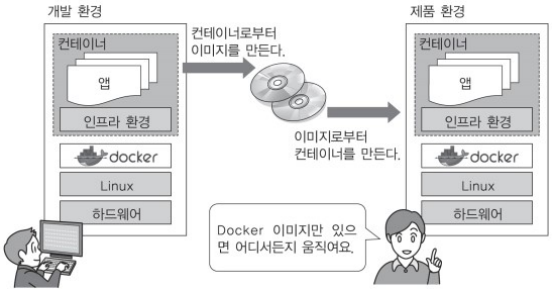
도커와 도커 컨테이너를 사용하면 (그림 3)과 같은 흐름으로 애플리케이션을 개발할 수 있다.



(그림 3) 도커를 사용한 시스템 개발 흐름[3]

(그림 4)에서처럼 개발자는 애플리케이션의 실행에 필요한 모든 것이 포함되어 있는 도커 이미지를 도커를 사용하여 작성한다. 그리고 이 이미지를 바탕으로 컨테이너를 가동시킨다. 이것은 객체지향프로그래밍 자바에서 클래스(이미지)와 인스턴스(컨테이너) 관계에 비유된다. 이 이미지는 도커가 설치되어 있는 환경이라면 기본

적으로 어디서든지 작동되므로 실행 환경차이로 발생하는 문제점들을 깨끗이 해결할 수 있다. 여기서 도커는 자바에서 JDE(Java Development Environment)에 비유될 수 있다.

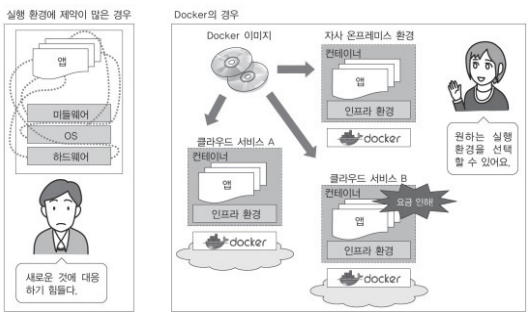


(그림 4) 도커 이미지와 도커 컨테이너[3]

3.2. 애플리케이션의 이식성(portability)

한 소프트웨어가 다양한 환경에서 동작될 수 있는 특성을 이식성(portability)이라고 한다. 도커는 이식성이 높기 때문에 (그림 5)에서처럼 도커 컨테이너의 바탕이 되는 도커 이미지만 있으면 애플리케이션을 동일한 환경에서 가동시킬 수 있다.

기업환경에서는 필요한 시스템을 구축하기 위해 하드웨어, 어플리케이션 등을 구입하고 상황에 맞게 IT환경을 꾸미는 것을 '온프레미스(On-premise)'라 하고 이와 반대로 기업 자체에 구축하는 것이 아니라 인터넷으로 연결된 다른 컴퓨터나 가상공간을 통해 데이터를 처리하는 방식을 '오프프레미스(Off-premise)' 혹은 Cloud computing이라고 한다.

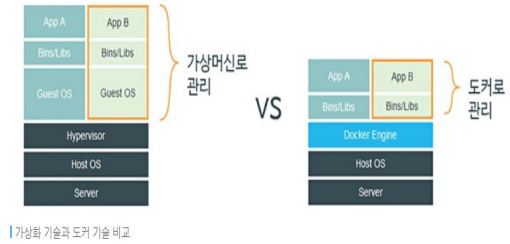


(그림 5) 도커 컨테이너의 이식성[3]

즉 개발한 업무 애플리케이션을 온프레미스 환경에 대한 이전뿐만 아니라 온프레미스 환경에서 클라우드, 클라우드에서 온프레미스 환경 간에도 자유로운 이전이 가능하다.

3.3. 도커 컨테이너

소프트웨어 이식성은 개발자의 골칫거리임에 분명하다. 한 소프트웨어의 사용 환경이 바뀔 때마다 네트워크 기술과 보안 정책, 스토리지가 제각각이어서 예기치 않은 오류에 직면하게 되기 때문이다. 그래서 '소프트웨어를 한 컴퓨팅 환경에서 다른 컴퓨팅 환경으로 이식할 때 안정적인 방법이 없을까?'라는 문제에 대한 해결책으로 제시된 것이 바로 '컨테이너'이다. 컨테이너는 항구에서 다양한 짐을 운반할 때 사용하는 컨테이너와 같이 애플리케이션과 그 실행에 필요한 것들을 패키지로 묶어 놓은 것으로 운영체제 위에서 구현된 가상화, 즉 '운영체제 레벨 가상화'이다. 이런 가상화를 하면 실행에 필요한 파일이 항상 함께 따라 다니므로 오류를 최소화할 수 있다[4].



가상화 기술과 도커 기술 비교

(그림 6) 가상머신(왼쪽)과 컨테이너(오른쪽)

'하드웨어 레벨 가상화'는 서버에 하이퍼바이저를 설치한 후 그 위에 가상 운영체제와 애플리케이션을 패키징한 '가상머신(VM)'을 만들어 실행하는 것으로 이 기술을 '서버 가상화'라고 한다. (그림 6)에서처럼 컨테이너(오른쪽)는 가상머신(왼쪽)과 달리 게스트 운영체제가 없어 용량이 작고 빠르다.

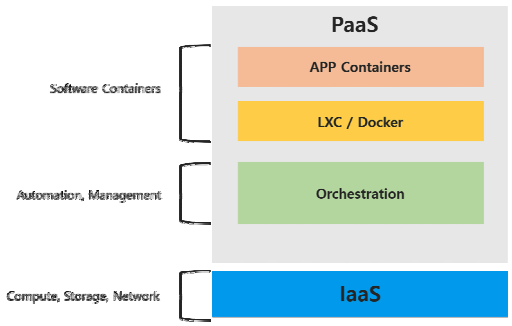
그런데 서버 가상화 기술이 존재하는데도 컨테이너가 인기를 끄는 이유는 크기와 속도 때문이다. 컨테이너에는 운영체제가 포함되지 않아 크기가 수십 MB로 작다. 반면 VM에는 운영체제가 포함되므로 보통 수 GB가

된다. 결국, 컨테이너는 서버 1대에서 실행할 수 있는 개수가 VM보다 최대 10배 이상 많고, 운영체제 부팅이 필요 없어 불과 수초 만에 서비스를 시작할 수 있다. 도커는 컨테이너 기술을 대중화시킨 1등 공신이다. 컨테이너들이 메인 운영체제를 공유하는 구조여서 서버 가상화보다 운영체제와의 접점이 많아 컨테이너가 뚫리면 운영체제가 해킹당할 가능성도 그만큼 커진다[4].

3.4. 도커 컨테이너와 PaaS와의 관계

PaaS 서비스는 플랫폼 기반으로 애플리케이션을 개발하기 때문에 특정 플랫폼에 종속될 수 있다는 단점이 있다. 도커는 "소프트웨어 컨테이너 안에 응용 프로그램들을 배치시키는 일을 자동화해 주는 오픈 소스 프로젝트이자 소프트웨어" 이다[5]. 도커는 컨테이너를 통한 PaaS를 가능하게 한다.

그러나 (그림 7)을 보면 컨테이너를 사용한다는 것 자체가 PaaS 솔루션을 사용하고 있다는 것을 의미하지는 않는다. 도커와 같은 표준 컨테이너 인프라는 컨테이너에서 앱을 운영하고 관리하기 위한 필수적인 툴들을 제공한다. 그러나 자동 확장, 트래픽 라우팅, 로드 밸런싱 등을 용이하게 하고 자동화하는 오케스트레이션 솔루션은 제공하지 않는다[5].



(그림 7) 도커 컨테이너와 PaaS[6]

4. 도쿠(Dokku)

도쿠는 도커 기반으로 작동하는 소형 배포시스템으로 미니 헤로쿠(Heroku)라고도 한다. Jeff Lindsay이 개발한 도쿠는 도커를 활용해 100줄 남짓한 쉘스크립트 본체와 도쿠를 둘러싼 몇 가지 모듈들을 활용해 미니 헤로쿠와 같은 PaaS 환경을 구축할 수 있도록 해준다[7,8].

헤로쿠를 사용하면 개발자가 애플리케이션을 만들고 헤로쿠에 있는 Git 서버에 푸쉬하기만 하면 애플리케이션을 자동으로 빌드하고 연결한 도메인으로 배포해준다. 도쿠는 바로 도커라는 가상화 기술을 활용해 이러한 PaaS를 직접 구축하게 도와주는 절대로 크지 않은 애플리케이션이다[1].

5. 결 론

본고에서는 클라우드 서비스에 대해 살펴보고, 2013년에 등장한 새로운 컨테이너 기반 가상화 도구인 도커, 그리고 도커 기반으로 작동하는 소형 배포시스템으로 미니 헤로쿠(Heroku)라고도 불리는 도쿠에 대해 살펴보았다.

컴퓨터 역사를 돌이켜 볼 때 초창기는 개발자가 하드웨어와 소프트웨어 개발 환경에 대한 풍부한 지식을 가지고 있어야 했고, 가지고 있는 것이 가능했다. 그러나 이제는 너무나 다양한 하드웨어가 나와 있고, 소프트웨어적인 환경도 복잡다단하여 '가상화'는 필수적인 개념이 되었다. 그런 의미에서 도커, 그리고 도쿠는 개발자가 순수 개발에만 집중할 수 있게 해주는 매우 유용한 도구임에 분명하다.

참 고 문 헌

[1] nacyot, 도쿠(Dokku)로 만드는 미니 히로쿠(Heroku), https://blog.nacyot.com/articles/2014-01-30-deploying-application-with-dokku/

[2] BLOTER, SaaS, IaaS, PaaS, http://www.bloter.net/archives/259518

[3] 길라잡이, 프로그래머에게 Docker란?, 정보문화사, https://blog.naver.com/infopub/221356560689

[4] 컨테이너(container), http://www.itworld.co.kr/news/103469

[5] 아코디언, PaaS와 Docker Container, http://blog.accordions.co.kr/221343560647

[6] 도커,
https://terms.naver.com/entry.nhn?docId=3578612&cid=59088&categoryId=59096
[7] The smallest PaaS implementation you've ever seen,
http://dokku.viewdocs.io/dokku/
[8] Docker powered mini-Heroku,
https://github.com/dokku/dokku

◐ 저 자 소 개 ◑

양 단 희
1989년 연세대학교 전산과학과(이학사)
1991년 연세대학교 대학원 전산과학과(이학석사)
1999년 연세대학교 대학원 컴퓨터과학과(공학박사)
1991년~1995년 현대전자 S/W 연구소
2013년 Visiting Scholar at Texas A&M University
2001년 3월~현재 평택대학교 컴퓨터학과 교수
관심분야: 인공지능, 컴퓨터보안, 기계학습, 소프트웨어공학, 자연어처리