JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 19, No. 3, pp. 453-460, Mar. 2018

컨테이너 기반 가상화를 통한 교육 실습환경 구축

윤 준 원[1] · 송 의 성[2*]
[1]한국과학기술정보연구원 슈퍼컴퓨팅본부
[2*]부산교육대학교 컴퓨터교육과

Building Education Practice Environment through Container-based Virtualization

JunWeon Yoon[1] · Ui-Sung Song[2*]
[1]Department of Supercomputing Center, KISTI, Daejeon 34141, Korea
[2*]Department of Computer Education, Busan National University of Education, Busan 47503, Korea

[요 약]

가상화 기술의 대표적인 특징은 사용자의 시스템 환경을 격리시킬 수 있고 컴퓨팅 자원을 사용자가 요구에 따라 유연하고 확장성 있게 지원할 수 있다는 점이다. 하지만 이미 잘 알려진 클라우드 컴퓨팅의 가상화 기술은 게스트 OS와 이를 관리하기 위한 하이퍼바이저 부하를 감수해야 한다. 이런 OS 기반의 가상화 문제점을 해결하기 위해 최근 컨테이너 기술이 부각되고 있다. 이 기술은 어플리케이션이 수행되는 프로세스를 격리화 시킴으로써 부하를 최소화 하면서 가상화와 유사한 환경을 구성할 수 있다.

본 연구에서는 기존 구축했던 클라우드 기반의 교육 실습 테스트 환경을 컨테이너 기반의 도커를 통해 구성하고자 한다. 이를 위해 교육 실습환경 구축시에 필요한 요구 사항을 분석하였다. 또한 컨테이너의 기능들을 분석하여 요구사항을 충족하기 위한 방법을 연구하였다. 이는 실습환경에 맞게 유연하고 확장성 있는 기존의 클라우드의 장점을 살리고, 성능부하의 이슈를 최소화함으로써 한정된 자원의 활용성을 최대화 할 수 있다.

[Abstract]

Virtualization technology is characterized by the ability to isolate the user's system environment and to support the computing resources flexibly and extensively on demand. However, virtualization technology of cloud computing, which is already well known, must overload the guest OS and the hypervisor to manage it. Container technology is emerging to solve such OS-based virtualization problems. This technology can isolate the processes under which the application is running, thus creating a virtualization-like environment with minimal overhead.

In this work, we construct a container-based education practice system using Docker instead of the existing cloud-based environment. To do this, we analyze the requirements for the establishment of the training practice environment. We also analyze the functions of the container and study the method to meet the requirements. This can take advantage of the existing flexible and scalable cloud computing. Also, it maximizes the availability of limited resources by minimizing the performance load.

색인어 : 가상화, 컨테이너, 도커, 클라우드, 교육시스템
Key word : Virtualization, Container, Docker, Cloud, Education System

http://dx.doi.org/10.9728/dcs.2018.19.3.453

Received 06 February 2018; **Revised** 20 February 2018
Accepted 25 March 2018

***Corresponding Author; Ui-Sung Song**

Tel: +82-51-500-7326
E-mail: ussong@bnue.ac.kr

Ⅰ. 서 론

근래 x86기반 멀티코어 아키텍처가 일반화 되고 GPU, INTEL PHI 등의 가속기 장착으로 단위 노드의 성능이 급격히 증가하였다. 가상화 기술은 이런 고성능 자원을 사용자가 요구에 따라 최적화하여 유연하고 확장성 있게 지원할 수 있는 특징을 지니고 있어 다양한 분야에서 각광을 받고 있다[1]. 대표적인 기술인 클라우드 컴퓨팅은 OS 기반의 가상화로 호스트머신과 가상머신 사이에 하이퍼바이저(Hypervisor)를 통해 자원을 할당하고 관리하게 된다. 가상머신을 구현하기 위하여 KVM, Xen, VMware 등의 기술들을 사용하였으나 개수가 많아지고 가상머신에서의 자원 사용량이 많은 작업을 수행할 경우 하이퍼바이저의 부하가 높아져 전체적인 성능 저하를 야기할 수 있다[2]. 이러한 OS 기반의 가상화 문제점을 해결하고자 필수자원에 대해서는 독립적으로 사용하고 그렇지 않은 경우는 공유를 통해 자원의 효율성을 높일 수 있는 컨테이너 기술이 개발되었다. 그 기반은 Linux Container 기술로 보통 LXC 라고 부르며 Linux Kernel을 독립적인 환경에서 사용하기 위한 API를 제공하며, 이 API를 잘 활용해서 만든 것이 도커(Docker)이다. 최근에는 도커로 대표되는 LXC(Linux Containers)와 같은 컨테이너형 가상화 쪽으로 이슈가 높아져가고 있다. 컨테이너형 가상화 기술은 기존의 가상화 기술보다 가벼워지고, 이식성이 뛰어난 특징을 가지고 있다[3].

본 연구에서는 컨테이너 기반의 가상화 기술인 도커를 활용한 교육 실습환경을 구축하고자 한다. 이 환경은 이미지 생성을 통해 다양한 실습환경 지원할 수 있으며 기존에 구축한 클라우드 컴퓨팅의 기반의 가상화 환경에 요구되는 부하를 최소화함으로써 자원의 활용성을 더욱 높일 수 있다[4].

Ⅱ. 관련 연구

2-1 LXC의 개념

OS의 내부는 물리적 자원을 관리하는 "커널 공간"과 사용자 프로세스(어플리케이션)을 실행하는 "사용자 공간"으로 구분된다. 컨테이너형 가상화는 사용자 공간을 여러 개로 나누어 각 사용자 프로세스에서 보이는 리소스를 제한한다. 이와 같이 여러 사용자 프로세스를 정리하여 격리시킨(Isolation) 사용자 공간이 바로 컨테이너이며 운영 시스템 레벨의 가상화 방법이다. (그림 1)과 같이 리눅스 커널은 Cgroups를 절충하여 가상머신을 시작할 필요 없이 자원 할당(CPU, 메모리, 블록 I/O, 네트워크 등)을 한다. Cgroups는 또한 애플리케이션 입장에서 프로세스 트리, 네트워크, 사용자 ID, 마운트 된 파일 시스템 등의 운영 환경을 완전히 고립시키기 위해 네임스페이스 격리화(Namespace Isolation)를 제공한다. LXC는 Cgroups와 네임스페이스를 결합하여 애플리케이션을 위한 고립된 환경을 제공한다[5].

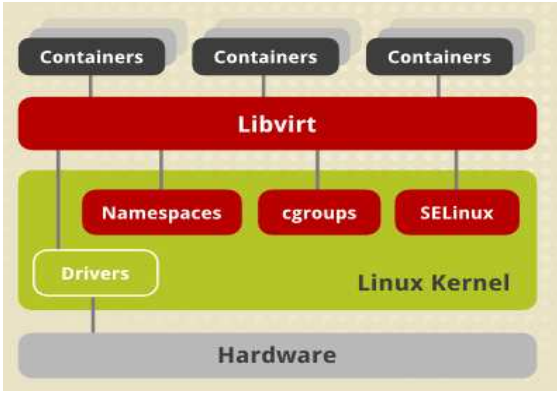

그림 1. LXC(리눅스 컨테이너)
Fig. 1. LXC(Linux Container)

Docker 또한 실행 드라이버의 하나로 LXC를 사용할 수 있으며 이를 통해 이미지 관리와 개발 서비스를 제공한다. 도커는 이미지 생성과 배포에 특화된 기능을 제공한다. 마치 깃(Git)처럼 이것을 이용하면 이미지를 push/pull할 수 있다. 뿐만 아니라 이것들을 공유할 수 있는 깃허브(GitHub)와 유사한 도커허브(Docker Hub)를 사용할 수 있다.

2-2 OS 기반 가상화와 LXC의 차이

잘 알려진 가상화 기술로는 KVM, Xen, VMware 등이 있으며, 호스트 OS상에 하이퍼바이저(Hypervisor)라는 일종의 자원을 관리하는 미들웨어를 두고 그 위에 가상머신 환경을 만들어 OS 환경을 구축하게 된다. 즉, (그림 2)의 좌측과 같이 OS 위에서 또 하나의 OS가 동작하는 형태이다. 이 경우 가상머신을 실행하는 호스트 OS와 게스트 OS를 완전히 분리 할 수있는 장점은 있지만, 호스트 OS에서 관장하고 있는 하드웨어의 자원을 하이퍼바이저(Hypervisor)를 통해 관리하고 할당하기 때문에 오버헤드가 증가하게 된다.

(그림 2) 우측의 Linux Container는 모든 프로세스가 호스트 OS에서 바로 시작된다. 일반적인 프로세스의 동작과 다른 점은 그 과정의 일부를 그룹화하고 다른 그룹이나 그룹에 속하지 않는 프로세스에서 단절된 공간으로 동작하여 컨테이너 간에는 다른 컨테이너의 내부를 볼 수 없게 된다. 화물 컨테이너처럼 독립된 공간에 프로세스가 들어 있기 때문에 이 공간을 '컨테이너'라고 부른다. 결론적으로 Linux Container는 KVM, Xen, VMware 등의 가상화 기술과는 다르게 가상머신을 생성하는 것이 아니라 OS가 사용하는 자원을 분리하여 여러 환경을 만들 수 있도록 하는 것이다. 따라서 가상화 오버 헤드는 거의 존재하지 않으며, 또한 가상 머신 부팅 및 종료라는 개념이 존재하지 않기 때문에 가상 환경의 시작과 종료를 빠르게 실행할 수 있는 장점을 가진다[6]. 더욱이 다양한 시스템 환경을 신속하게 적용하고 배포할 수 있어 교육용 실습환경 접목에 최적의 조건을 갖는다.

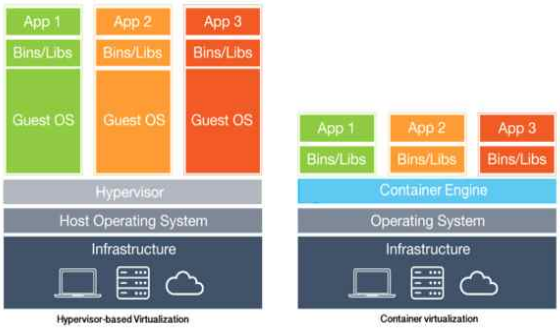

그림 2. OS 기반의 가상화와 LXC 차이
Fig. 2. Differences of OS-based virtualization and LXC

1) Linux Container

가상화의 목적은 사용자의 어플리케이션을 저렴한 비용으로 빠르고, 안정적으로 제공하기 위한 기반 환경이다. 이를 위해 기존에는 호스트 머신에 별도의 독립적인 운영체제를 올리고 네트워크와 디스크 프로세스가 또한 격리시켜 서비스를 제공했다. 하지만 어플리케이션 몇 개를 수행하기 위해 이와 같이 별도의 OS를 구성해서 올리는 것은 엄청난 자원 소모일 수 있다. 컨테이너의 개념은 간단하다. 격리된 네트워크, CPU, 메모리, 디스크를 가진 공간을 만들고 이 공간에서 프로세스 실행해서 사용자 서비스를 하는 것이 목적이다[7]. 이 공간(컨테이너)을 여러개 만들어서 제공하는 것으로 멀티테넌시(Multitenancy) 환경을 만들 수 있다. 격리된 네트워크, CPU, 메모리, 디스크를 가진 공간을 만들고 여기서 프로세스 실행하여 사용자의 어플리케이션을 수행하게 되는데 이런 격리된 환경의 자원을 관리하기 위해서 아래와 같은 기술들이 요구된다.

(1) Contrel Grups(Cgroup)

2006년 구글 엔지니어로부터 시작된 프로젝트로 대량의 프로세스를 운영해야 하는 구글의 요구사항에 필요했던 기술로 Process Containers라는 이름으로 자원을 격리하기 위한 기능 개발을 시작했다. 그리고 2007년 Cgroup(control group)으로 이름을 변경하고 2008년 리눅스 커널 2.6.24의 메인 코드에 추가했다. Cgroup은 아래의 기능들을 제공한다[8].

- Resource limiting: 프로세스 그룹이 사용할 메모리와 파일 시스템 캐시등에 대한 제한 설정
- Prioritization: 프로세스 그룹간 디스크 입출력과 CPU 이용에 대한 우선순위 설정
- Accounting: 프로세스 그룹의 자원 사용량 측정
- Control: 프로세스에 대한 checkpoint를 설정 및 재시작

(2) Namespace

Cgroup이 기능이 추가되는 동안 리눅스 커널에 네임스페이스 격리(Namespace Isolation)라는 기능을 추가한다. 리소스들을 서로 다른 네임스페이스 그룹으로 나누어 다른 네임스페이스에서 볼 수 없게 하는 기술로 가령, PID 네임스페이스는 서로 다른 네임스페이스에서 독립된 PID를 관리 할 수 있도록 한다. 즉, 하나의 운영체제에서 여러 개의 프로세스 tree를 운용할 수 있게 된다. 그 외에도 아래와 같은 자원에 대한 네임스페이스 관리 기능을 제공하며 리눅스 프로세스들은 각 타입별로 네임스페이스에 포함될 수 있다[8].

- UTS Namespace: hostname 관리 공간
- IPC Namespace: IPC(Inter Process Communication)관리 공간
- PID Namespace: PID 즉 프로세스를 관리하는 공간
- MNT Namespace : 파일 시스템 마운트를 관리하는 공간
- NET Namespace : 네트워크를 자원을 관리하는 공간
- User Namespace : 리눅스 어카운트(User, Group)를 관리하는 공간

(3) Cgroup & Namespace

Cgroup과 네임스페이스를 서로 결합하면, 기존의 가상 머신과 비슷한 수준의 가상환경을 만들 수 있다. Cgroup를 이용해서 자원의 사용량을 제한하고 네임스페이스를 이용해서 특정 유저만 자원을 볼 수 있도록 제한할 수 있다.

(4) Storage

클라우드 환경의 가상머신은 각각 별도의 OS로 동작하기 때문에 OS상에 블록디바이스를 할당해주면 된다. 하지만 컨테이너는 좀 다르다. 컨테이너는 근본적으로 프로세스의 실행이기 때문이다. 프로세스는 하나의 프로그램 이미지에서 시작한다. 프로세스의 데이터가 변경되더라도 원본 프로그램 이미지를 변경할 수 없다. 대신 설정파일등을 이용해서 원본 프로그램 이미지의 변경 없이, 격리된 프로세스를 실행 할 수 있다. vim을 예로 들자면, 유저별로 .vimrc를 설정하는 것으로 전혀 다른 환경의 vim 프로세스를 실행할 수 있다.

컨테이너 실행 과정은 프로그램과 각종 설정파일들을 가지고 있는 컨테이너 이미지로 부터 생성된다. 컨테이너 이미지는 프로그램 이미지, 컨테이너는 프로세스에 대응이 된다. 프로그램과 마찬가지로 컨테이너 이미지는 변경 할 수가 없다. 누가 실행 하든지 간에 동일한 프로그램 환경을 보장해야 하기 때문이다. 컨테이너가 실행되면, 원본인 컨테이너 이미지로 부터 변경되는 정보를 저장할 수 있어야 한다. 원본으로 부터 복사(Copy)를 하고 변경되는 정보를 쓰는 (Write)하는 저장 방식이 필요하다. 원본을 수정하지 않고 변경된 정보만 저장하는 것을 COW(Copy on Write)기법이라고 하는데, 컨테이너를 구현하기 위해서는 COW 파일 시스템이 반드시 필요하다.

2) 컨테이너의 취약점

도커 기반으로 배포되고 운영되는 어플리케이션들은 운영체제 및 악의적 내부 침입 등의 취약점을 지니고 있다. 도커 호스트와 그 안에서 운영되는 컨테이너 모두를 보호 하는 지를 파악하고 이해하는 것이 중요하다. 최근에는 이런 도커의 취약점을 해결하기 위해 컨테이너 가시성을 제공하고 컨테이너가 지원하는 호스트 및 컨테이너를 보호할 수 있도록 보안이 강화되고 있다[9].

(1) Host OS 에 종속적

리눅스 컨테이너는 OS에서 Linux Kernel이 관리하는 것이므로, Linux 이외의 다른 OS 에서는 동작하지 않으며, 컨테이너 환경에서도 다른 OS 설치가 불가능하다. 컨테이너에 Linux 계열의 모든 배포판을 설치할 수 있으나 실행중인 커널은 하나이기 때문에 결국 컨테이너는 Host OS 의 커널을 사용하게 된다.

(2) 컨테이너별 커널구성이 불가능

커널에 관련 작업은 가능하나 컨테이너마다 다른 커널 작업을 수행 할 수는 없다. 커널의 기능으로 구성되는 환경이기 때문에 당연히 전체 컨테이너에서 보이는 커널은 동일하다. 따라서 컨테이너에서 보이는 장치나 로드되는 커널 모듈은 모두 동일하다. LXC는 Cgroups 기능을 사용하여 다양한 자원을 분리하여 독립적인 환경을 제공한다. LXC로 만들어진 가상 환경은 호스트 환경에서 완전히 고립된 상태이기 때문에 가상 환경에서 호스트 환경의 리소스를 직접 액세스 할 수 없다. 또한, root 권한으로 동작하기 때문에 컨테이너에서 root 권한으로 실행되는 프로세스는 호스트 환경에서 root 권한을 갖는다. 이 문제를 해결하기 위해 LXC 1.0 컨테이너에서는 root 사용자를 컨테이너 외부의 다른 사용자로 처리하는 방식으로 구현되었다.

본 연구에서는 공용의 실습환경에 주안점을 두었다. 추후 보안이 요구되는 실습환경의 경우 독립적인 수행공간과 더불어 각 환경에 대한 보안이 중요시 된다. 따라서 컨테이너 기반의 가상 실습 구축에 보안 이슈에 대한 연구가 추가적으로 수행되어야 한다[10].

Ⅲ. 컨테이너 기반의 실습환경 구축

3-1 요구사항 분석

컨테이너 기반의 교육 실습환경을 구축하기 위해서는 다음과 같은 기능들이 필요하다.

① 교육용 실습환경을 구성하기 위해 필요한 실습환경 이미지를 만들고 배포하여야 한다. 보통 도커에서 제공하는 Docker Hub[11]를 많이 사용하나 실습환경과 같은 프라이빗한 환경에서는 로컬 저장소를 구축하여 저장하게 된다. 본 연구에서는 초기 베이스 이미지를 Docker Hub를 통해서 다운로드 받고 이 이미지 위에 실습환경 이미지를 생성한 후 로컬 저장소에 저장하게 된다.

② 컨테이너 기반의 실습환경을 구축하기 위해서는 컨테이너를 묶어 구성하고 배포하는 container orchestration 도구가 필요하다. 특히 여러 호스트에서 컨테이터가 구동되기 위해서는 네트워크 설정, 리소스의 분배, 장애 복구 등의 기능이 필수이다. 본 실험에서는 Kubernetes를 설치하여 구성하였다. 이는 Linux 컨테이너 작업을 자동화하는 오픈소스 플랫폼으로 Linux 컨테이너를 실행하는 호스트 그룹을 클러스터링 할 수 있으며 쉽고 효율적으로 관리할 수 있다. Kubernetes는 구글엔지니어들이 개발하고 설계한 플랫폼으로 멀티플 컨테이너의 스케줄링, 확장, 워크로드 관리를 편리하게 제공하고 있다.

③ 본 실험을 위해 가상머신(Virtual Machine)을 구성하고 가상머신 위에 도커를 설치하였다. 컨테이너가 구동되는 호스트머신과 컨테이너들을 관리하기 위해서는 이 모두를 관리하기 위한 네트워크 솔루션을 적용하여야 한다.

④ 컨테이너 실습환경이 구성되면 컨테이너 내부에 공용 파일시스템을 마운트 하여 실습 데이터와 가공한 산출물 데이터를 관리할 수 있도록 설정해주어야 한다.

3-2 컨테이너 기반 실습환경 구축

컨테이너 기반의 교육 실습환경 구축을 위해 (그림 3)과 같이 시스템 환경을 설정하였다. 호스트머신에 3개의 가상머신을 설치하였다. 도커를 실행하기 위한 커널 버전은 3.10.x 이상을 권고하고 있으며, 본 실험환경은 호스트머신에는 CentOS 7.3 가상머신에는 Ubuntu 16.04.3을 설치하였고 컨테이너가 수행되는 환경을 가상머신에서 설정함으로써 추후 보안에 대한 취약부분을 가상머신 단위로 격리할 수 있다. 한 대의 가상머신에서 복수개의 도커 컨테이너들이 생성되고 실습환경의 규모에 따라 호스트 머신과 가상머신의 확장이 가능하다.

실습환경이 수행될 컨테이너 이미지 관리를 위해서 로컬 레지스트리(Registry)를 구성하였다. 보통 컨테이너 이미지는 도커 허브(Docker Hub)와 같은 공식 레지스트리를 사용할 수 있으나 실습환경과 같은 프라이빗한 이미지를 업로드하여 서비스하기에는 보안 및 네트워크 부하 등과 같은 제한적인 요소들이 존재한다. 본 연구에서는 Docker Registry를 이용하여 호스트머신에 이미지 저장소를 직접 구축하였다.

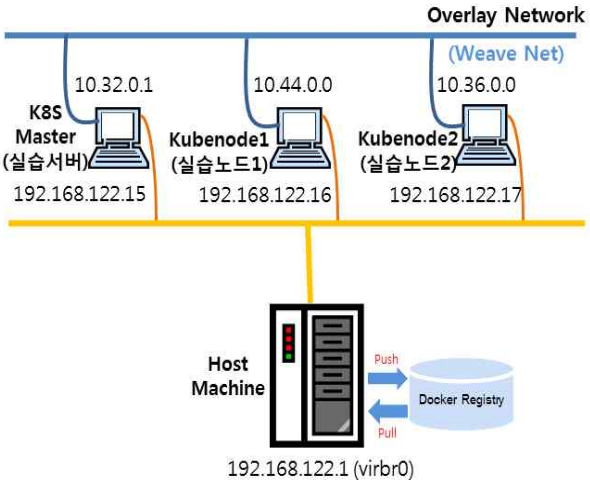

그림 3. 컨테이너 기반 교육 실습환경 구성도

Fig. 3. Container-based education practice system configuration diagram

다음으로는 실습환경 컨테이너 이미지를 생성하였다. 본 실험에서는 “C 컴파일러” 실습환경을 예로 이미지를 생성하였다. Dockerfile이라는 이미지 빌드용 DSL(Domain Specific Language) 파일을 사용하여 (그림 4), <표 1>와 같이 CentOS 베이스 이미지에 GCC를 설치하여 edu_gcc라는 컨테이너 이미지를 생성하였다. (그림 3)의 실습서버와 실습노드에는 호스트머신의 볼륨을 fuse와 sshfs를 사용하여 공유하도록 구성하였다. 각 실습노드에서 수행되는 컨테이너에 이 공유 볼륨을 설정하기 위해 Dockerfile에 volume ["/mybackup"]을 설정한 후 컨테이너 수행시 –v 옵션을 사용하여 마운트 하도록 설정하고 read-only(:ro옵션)을 부여하여 실습 이미지가 사용자에 의해 변경되지 않도록 하였다.

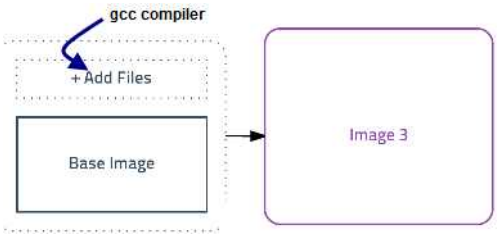

그림 4. 실습환경 이미지 파일 생성
Fig. 4. Create a practice lab image file

표 1. Dockerfile을 통한 실습환경 이미지 구축
Table 1. Create a practice lab image file using Dockerfile

```
843% [root@ut1 hello_docker]# cat Dockerfile  # Dockerfile Creation
FROM centos
MAINTAINER jwyoon <junweon@gmail.com>
RUN yum install -y gcc
RUN mkdir /edu_lab
COPY ./hello.c /edu_lab
WORKDIR /edu_lab
RUN gcc -o hello hello.c
volume ["/mybackup"]
#RUN hello
844% [root@ut1 hello_docker]# docker build -t edu_gcc  # edu_gcc Image Creation
Sending build context to Docker daemon 4.096 kB
Step 1 : FROM centos
 ---> ff426288ea90
Step 2 : MAINTAINER jwyoon <junweon@gmail.com>
 ---> Using cache
 ---> 6dda4836971d
Step 3 : RUN yum install -y gcc
 ---> Using cache
 ---> 960898f3c7bc
Step 4 : RUN mkdir /edu_lab
 ---> Using cache
 ---> e514b4b0c164
Step 5 : COPY ./hello.c /edu_lab
 ---> Using cache
 ---> bfcca870247e
Step 6 : WORKDIR /edu_lab
 ---> Using cache
 ---> 763a98a9dde1
Step 7 : RUN gcc -o hello hello.c
 ---> Using cache
 ---> 304f6a53257c
Step 8 : VOLUME /mybackup
 ---> Using cache
 ---> f2688bbe7290
Successfully built f2688bbe7290
845% [root@ut1 hello_docker]# docker images
REPOSITORY          TAG                 IMAGE ID
edu_gcc             latest              f2688bbe7290
857% [root@ut1 hello_docker]# docker run -it -v /mybackup:/mybackup:ro f2688bbe7290 /bin/sh
                                        # Mount the Shared Volume using Read-only option
sh-4.2# cd /mybackup/; ls
ATLAS       STRIDE_v1.1         hpl-2.2         mpi_hello.c       mpich-3.2_vm1  mpich-3.2_vm4
ATLAS_vm    STRIDE_v1.1.tar.gz  hpl-2.2.tar.gz  mpich-3.2         mpich-3.2_vm2  phi_benchmark
Readme.txt  atlas3.10.2.tar.bz2 hpl-2.2_vm      mpich-3.2.tar.gz  mpich-3.2_vm3  stream
sh-4.2# touch tt
touch: cannot touch 'tt': Read-only file system
```

생성된 GCC 실습용 컨테이너 이미지를 push 명령어를 사용하여(# docker push localhost/edu_gcclab) 로컬 레지스트리에 등록하였다. 이로써 각 실습환경에서는 구축된 프라이빗 레지스트리에서 생성된 실습환경 이미지를 실행할 수 있다.

기본적으로 도커가 구동되면 호스트 시스템에 "docker0"이라는 가상 브리지를 만들고 프라이빗 네트워크가 설정된다. 컨테이너 내에는 가상 이더넷 장치가 브리지에 연결되고 컨테이너 내부의 eth0에 매핑 되며 위에서 언급 한 네트워크 “docker0” 범위 내의 IP가 사용된다. 이 IP는 도커를 실행중인 각 호스트에 대한 고려 없이 설정된다. 따라서 네트워크 범위가 충돌 할 수 있으며 동일한 가상 브리지에 연결된 컨테이너 간에만 통신할 수 있다. 다른 호스트의 다른 컨테이너와 통신하려면 포트 매핑 설정에 따라야한다. 즉, 호스트 시스템의 포트를 각 컨테이너에 할당 한 다음 해당 포트의 모든 트래픽을 해당 컨테이너로 전달해야 한다. 애플리케이션이 자신의 IP 주소를 다른 노드에서 호스팅 되는 컨테이너와 통신하기 위해서는 중복되지 않은 IP와 복잡한 포트 매핑을 설정하여야 한다. 네트워크 기반의 작업 방식을 변경할 필요가 없으므로 응용 프로그램을 가상 시스템에서 컨테이너로 쉽게 포팅하기 위한 솔루션으로 Kubernetes에 적용 가능한 네트워크 솔루션으로 Contiv, Flannel, Nuage Networks, OpenVSwitch, OVN, Project Calico, Romana 및 Weave Net 등이 있다[12].

본 연구에서는 <표 2>과 같이 Weave Net을 설치하여 여러 호스트에서 도커 컨테이너를 연결하고 자동 검색을 가능하게 하는 가상 네트워크(Overlay Networks) 생성하였다. 응용 프로그램은 포트 매핑, 대사 또는 링크를 구성하지 않고 컨테이너가 모두 동일한 네트워크 스위치에 연결되어있는 것처럼 네트워크를 사용할 수 있다.

표 2. Weave Net을 통한 컨테이너 네트워크 환경 설정
Table 2. Container Network configuration using Weave Net

```
3634% [root@peter ~]# pdsh -w ut[1-3] 'ifconfig weave'
ut1: weave     Link encap:Ethernet  HWaddr f2:da:8a:47:23:4b
ut1:           inet addr:10.32.0.1  Bcast:0.0.0.0  Mask:255.240.0.0
ut1:           inet6 addr: fe80::f0da:8aff:fe47:234b/64 Scope:Link
ut1:           UP BROADCAST RUNNING MULTICAST  MTU:1376  Metric:1
ut1:           RX packets:464879 errors:0 dropped:0 overruns:0 frame:0
ut1:           TX packets:538162 errors:0 dropped:0 overruns:0 carrier:0
ut1:           collisions:0 txqueuelen:1000
ut1:           RX bytes:108353590 (108.3 MB)  TX bytes:91742188 (91.7 MB)
ut1:
ut2: weave     Link encap:Ethernet  HWaddr 5a:5f:c2:c5:14:2b
ut2:           inet addr:10.44.0.0  Bcast:0.0.0.0  Mask:255.240.0.0
ut2:           inet6 addr: fe80::585f:c2ff:fec5:142b/64 Scope:Link
ut2:           UP BROADCAST RUNNING MULTICAST  MTU:1376  Metric:1
ut2:           RX packets:2705 errors:0 dropped:0 overruns:0 frame:0
ut2:           TX packets:2247 errors:0 dropped:0 overruns:0 carrier:0
ut2:           collisions:0 txqueuelen:1000
ut2:           RX bytes:199266 (199.2 KB)  TX bytes:199316 (199.3 KB)
ut2:
ut3: weave     Link encap:Ethernet  HWaddr 26:e6:34:1d:2a:1e
ut3:           inet addr:10.36.0.0  Bcast:0.0.0.0  Mask:255.240.0.0
ut3:           inet6 addr: fe80::24e6:34ff:fe1d:2a1e/64 Scope:Link
ut3:           UP BROADCAST RUNNING MULTICAST  MTU:1376  Metric:1
ut3:           RX packets:2653 errors:0 dropped:0 overruns:0 frame:0
ut3:           TX packets:2215 errors:0 dropped:0 overruns:0 carrier:0
ut3:           collisions:0 txqueuelen:1000
ut3:           RX bytes:193168 (193.1 KB)  TX bytes:196280 (196.2 KB)
ut3:
```

클라우드의 가상화 환경에서 가상머신들을 관리하는 IaaS(Infrastructure as a Service) 도구들과 유사하게 컨테이너를 관리하는 도구가 필요하다[13]. 본 논문에서는 컨테이너 관리 도구인 Kubernetes를 설치하여 마스터노드(ut1)와 컨테이너가 수행되는 실습노드 2대(ut2, ut3)를 <표 3>과 같이 구성하였다. 하이퍼바이저를 통해 개별 OS 설치된 가상머신을 생성, 실행, 종료하는 소요시간에 비해 컨테이너 관리는 이런 부하를 줄일 수 있다. 또한 마스터노드- 모든 서비스는 컨테이너로 띄운다. Kubernetes Deployment는 애플리케이션 인스턴스들을 생성하며 Kubernetes 마스터는 클러스터에 존재하는 각각의 노드위에 애플리케이션 인스턴스를 스케줄링 하고 지속적으로 모니터링 한다. Deployment의 생성과 관리는 Kubernetes Command Line Interface(CLI)인 kubectl을 사용하게 된다[14].

Deployment를 생성하면 Kubernetes는 애플리케이션을 호스트 장비의 pod으로 생성하는데 가장작은 deploy를 위한 단위이다. pod에는 한개 또는 여러 개의 애플리케이션 컨테이너가 존재한다. 그리고 각 컨테이너들이 리소스를 공유할 수 있다. <표 3>은 Kubernetes를 사용하여 앞서 생성한 실습환경 이미지(edu_gcc)를 Deployment하고 Pod이 생성되는 모습을 보여주고 있다.

표 3. Kubernetes를 통한 컨테이너 관리
Table 3. Container management using Kubernetes

```
1149%[root@ut1 ~]# kubectl cluster-info
Kubernetes master is running at https://192.168.122.16:6443
KubeDNS is running at https://192.168.122.16:6443/api/v1/namespaces/kube-system/
services/kube-dns:dns/proxy

1151%[root@ut1 ~]# kubectl get nodes
NAME      STATUS    ROLES     AGE       VERSION
ut1       Ready     master    20m       v1.9.2
ut2       Ready     <none>    10m       v1.9.2
ut3       Ready     <none>    10m       v1.9.2

1153%[root@ut1 ~]#  kubectl run edu-lab -image=edu_gcc
deployment "edu-lab" created

1154% [root@ut1 ~]# kubectl get deployment
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
edu-lab        1         1         1            0           15s
1155% [root@ut1 ~]# kubectl get pod
NAME                    READY     STATUS              RESTARTS   AGE
edu-lab-7f75447b5f-9qkj5   0/1       ContainerCreating   0          26s
```

도커 데몬은 TCP port 대신 Unix socket 과 바인드 되어있다. 기본 유저로 root 로 설정이 되어 있어서 sudo 를 이용하여하는데 이를 위해 실습 사용자 그룹을 설정하고 이를 도커 그룹을 생성하여 <표 4>와 같이 추가하였다. 도커 데몬이 실행시 Unix socket 의 읽기/쓰기의 권한을 도커 그룹이 가지고 있기 때문이다.

표 4. 도커 그룹 생성
Table 4. Create a Docker group

```
// added a docker group
[root@host ~]# useradd edu01 －m －s /bin/bash
// added lab users to docker group
[root@host ~]# sudo usermod -aG docker 사용자 계정
```

Ⅳ. 결 론

도커는 컨테이너 기반의 오픈소스 가상화 플랫폼으로 컨테이너 생성과 배포에 있어 다양한 장점을 가지고 있다. 또한 Docker Hub를 통해 요구사항에 맞는 컨테이너 이미지를 무료로 배포 관리하고 있다. 최근에는 개발(Development)과 운영(Operation) 조직이 분리되어 발생할 수 있는 업무 효율성을 줄이기 위해 DevOps(Dev+Ops)라는 용어가 사용되고 있다[15]. 이는 테스트, 운영에 이르는 전 구간을 자동화하여 배포 주기를 줄이기 위한 방법론으로 대표적인 솔루션으로 Docker 컨테이너가 적용되고 있다.

본 논문에서는 이런 컨테이너 기반의 가상화 솔루션인 도커를 활용하여 교육 실습환경을 구축하였다. 이를 위해 컨테이너 환경에서 요구사항을 분석하였고, 각 요구사항에 필요한 솔루션을 적용·설치하였다. 컨테이너 가상화 기술은 앞서 언급했듯이 기존에 OS를 통한 클라우드 환경의 가상머신에 비해 Cgroups와 네임스페이스를 활용하여 프로세스를 격리화 함으로써 더욱 빠르고 유연하게 독립된 어플리케이션 환경을 제공할 수 있다. 이런 장점을 교육 실습환경에 접목함으로써 다양한 실습환경을 쉽게 개발, 구축하고 배포할 수 있어 시간과 비용의 절감을 기대할 수 있다. 현재 구축된 환경은 기능적인 부분을 고려하여 구성한 실습 환경으로 향후, 실제 컨테이너의 보안 및 배포 확장성 등의 취약점을 고려하여 기능을 보완해야 할 것이다.

감사의 글

본 논문은 2017년도 부산교육대학교 교내 연구과제로 지원을 받아 수행된 연구임

참고문헌

[1] Zhang, Y., Zhang, G., Liu, Y., & Hu, D, "Research on services encapsulation and virtualization access model of machine for cloud manufacturing" Journal of Intelligent Manufacturing, 28(5), pp.1109-1123, 2017.

[2] Daniel J. Abadi, “Data Management in the Cloud: Limitations and Opportunities”, In IEEE DE Bulletin, vol 32(1), pp.3-12, Feb 2009.

[3] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014.239.2, 2014.

[4] Y JunWeon, P ChanYeol, S Ui-Sung, "Building the Educational Practice System based on Open Source Cloud Computing." Journal of Digital Contents Society 14.4, pp.

505-511, 2013.

[5] Celesti, A., Mulfari, D., Fazio, M., Villari, M., & Puliafito, A, "Exploring container virtualization in IoT clouds", In Smart Computing(SMARTCOMP), IEEE International Conference, IEEE, pp.1-6, May 2016.

[6] Babu, A., Hareesh, M. J., Martin, J. P., Cherian, S., & Sastri, Y, "System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver", In Advances in Computing and Communications(ICACC), IEEE, pp.247-250, August 2014.

[7] Fink, J., " Docker: a software as a service, operating system-level virtualization framework", Code4Lib Journal, 25, 29, 2014.

[8] Dua, R., Raja, A. R., & Kakadia, D, "Virtualization vs containerization to support paas", In Cloud Engineering(IC2E), 2014 IEEE International Conference, pp.610-614, March 2014.

[9] Bui,T,"Analysis of docker security" arXiv preprint arXiv:1501.02967, 2015.

[10] Combe, T., Martin, A., & Di Pietro, R., "To Docker or not to Docker: A security perspective. IEEE Cloud Computing, 3(5), pp.54-62, 2016.

[11] Docker Hub [Internet]. Available: https://hub.docker.com/.

[12] Weave Net [Internet]. Available: https://www.weave.works/oss/net/.

[13] Manvi, Sunilkumar S., and Gopal Krishna Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey." Journal of Network and Computer Applications, 41, pp.424-440, 2014.

[14] Kubernetes [Internet]. Available: https://kubernetes.io/.

[15] Bass, L., Weber, I., Zhu, L. DevOps: A Software Architect's Perspective. Addison-Wesley Professional,2015.

윤준원(JunWeon Yoon)

2002년～2004년 : 고려대학교 대학원 컴퓨터학과(이학석사)
2010년～2011년 : 고려대학교 대학원 컴퓨터학과 박사 수료

2005년~현 재 : KISTI 국가슈퍼컴퓨팅연구소 선임연구원
※관심분야: 분산컴퓨팅, 결함포용시스템, 슈퍼컴퓨팅, 병렬파일시스템, 배치스케줄링, 벤치마크(BMT)

송의성(Ui-Sung Song)

1991년～1997년 : 고려대학교 컴퓨터학과(학사)
1998년～1999년: 고려대학교 대학원 컴퓨터학과(이학석사)
2000년～2005년: 고려대학교 대학원 컴퓨터학과(이학박사)

2006년～현 재: 부산교육대학교 컴퓨터교육과 교수
※관심분야: 컴퓨터교육, 교육용로봇교육, 컴퓨터네트워크, 스마트러닝