지구물리와 물리탐사 Geophysics and Geophysical Exploration
Vol. 24, No. 2, 2021, p. 45-52 https://doi.org/10.7582/GGE.2021.24.2.45
ISSN 1229-1064 (Print)
ISSN 2384-051X (Online)

해 설

리눅스 컨테이너와 버전 관리 시스템을 이용한 소프트웨어 연구 환경 구축

하완수 *

부경대학교 에너지자원공학과

Building Software Research Environment using Linux Container and Version Control System

Wansoo Ha*

Department of Energy Resources Engineering, Pukyong National University

요 약

소프트웨어 기술 발달에 따라 점점 더 많은 과학자와 공학자들이 연구를 위해 컴퓨터 소프트웨어와 프로그래밍 도구들을 사용하고 있다. 소프트웨어를 이용한 연구에서는 환경 설정, 재현성 및 소스 코드 손실과 같은 문제들이 발생할 수 있다. 이 해설에서는 리눅스 컨테이너와 버전 관리 시스템을 사용하여 이러한 문제를 방지하는 방법에 대해 조사하였다. 연구 프로젝트 단위로 클라우드 저장소를 통해 코드를 관리하고 리눅스 컨테이너에 연구 환경을 구축하면 위의 문제들을 방지하고 협동 연구를 더 쉽게 만들 수 있다. 리눅스 컨테이너 사용 경험이 없는 연구자들을 위해 컨테이너 생성과 실행에 필요한 스크립트를 포함한 연구 프로젝트 템플릿 저장소를 공개하였다.

주요어

소프트웨어, 프로그래밍, 리눅스 컨테이너, 버전 관리

ABSTRACT

With advancements in software technology, more scientists and engineers are employing computer software and programming tools for research. However, several issues can arise in software-based research: environment setting, reproducibility, and loss of source codes. This study investigates the use of Linux containers and version control systems to prevent these problems. Managing research projects using a cloud source-code repository and building a research environment in a Linux container can prevent the abovementioned problems and make research collaboration easier. For researchers with no experience with Linux containers, a repository of project template containing shell scripts for building and running containers has been released.

KEYWORDS

software, programming, Linux container, version control

서 론

소프트웨어가 세상을 집어 삼키고 있다(Andreessen, 2011). 각종 아날로그 매체에 담겨있던 정보들은 디지털 비트가 되어 증발하고 있다(Negroponte, 1995; Tercek, 2015). 레코드판, 카세트테이프, 비디오테이프가 증발하였고, 신문과 책도 증발하고 있다. 증발된 정보는 클라우드에(cloud) 존재하며, 사용자가 원할 때(on-demand) 아날로그 매체 없이 비트만 받아 텔레비전, 컴퓨터, 스마트폰으로 보고 들을 수 있다.

하드웨어 장치들도 증발하고 있다. 비디오플레이어, CD 플레이어, MP3 플레이어, 카메라, 계산기 등은 이제 컴퓨터 소프트웨어나 손바닥 안의 앱으로 존재한다. 각 기관들에서 운영하던 서버 컴퓨터들도 클라우드로 옮겨가고 있다. 소프트웨어 개발자들은 이제 인프라도 코드로 관리한다(Morris, 2016).

코로나19는 회사 사무실들을 증발시켰고, 학교와 학원 교실에서 진행되던 수업들을 증발시켰다. 재택 근무와 온라인 강의

Received: 27 April 2021; Revised: 21 May 2021; Accepted: 25 May 2021
*Corresponding author
E-mail: wansooha@pknu.ac.kr
Address: Department of Energy Resources Engineering, Pukyong National University, 45 Yongso-Ro, Nam-Gu, Busan 48513, Republic of Korea

는 코로나19 이후에도 상당 부분 남아 있을 것으로 예상된다. 앞으로 많은 사회적 경제적 활동들이 가상의 소프트웨어 세상인 메타버스에서 일어날 것이다(Dionisio *et al.*, 2013).

다양한 과학 기술 분야의 연구도 컴퓨터 소프트웨어로 진행되고 있다(Hannay *et al.*, 2009; Prabhu *et al.*, 2011). 컴퓨터 시뮬레이션은 많은 물리 모형 실험을 대체하였다. 디지털 트윈은 현실 상황을 컴퓨터로 재현한다(Negri *et al.*, 2017). 최근 다양한 학문 분야에서 널리 시도하고 있는 머신러닝은 데이터 수집 후 훈련부터 추론까지 컴퓨터 소프트웨어로 진행한다.

외부 소프트웨어나 직접 개발한 프로그램을 이용한 연구에서는 컴퓨터와 관련된 다양한 문제들이 발생할 수 있다. 운영체제나 라이브러리 업데이트 후 잘 작동하던 프로그램이 실행되지 않을 수 있다. 컴퓨터마다 운영체제나 라이브러리 환경이 다르므로 한 컴퓨터에서 잘 실행되던 프로그램이 다른 컴퓨터에서는 실행되지 않는 경우도 발생한다. 심각한 경우에는 다른 컴퓨터에서 프로그램을 실행하면 실행 결과가 달라지기도 한다. 드물게 실수, 컴퓨터 하드웨어 고장 또는 해킹 피해로 인해 작성한 프로그램 소스 코드가 사라지는 경우도 발생한다. 이러한 문제들은 연구의 핵심 내용은 아니지만 문제 발생시 연구가 상당 기간 지연될 수 있다.

본 해설에서는 위의 문제들을 방지하고 각종 소프트웨어와 프로그래밍을 이용한 연구와 교육에 도움이 될 수 있도록 리눅스 컨테이너와 소스 코드 관리 기술을 사용하는 방법에 대해 설명한다. 이를 위해 Fig. 1과 같은 시스템에서 호스트 컴퓨터에 도커(Docker, 2021)와 깃(Git, 2021)이 설치되어 있고, 사용자가 기본적인 리눅스 쉘 명령어와 텍스트 에디터 사용법을 알고 있다고 가정한다. 아래의 내용들은 모두 인터넷에 공개되어 있는 내용들이지만, 경험이 없는 연구자가 찾아서 적용하기에는 많은 시간이 소요될 수 있다. 본 해설에서는 리눅스 컨테이너 사용 경험이 없더라도 쉽게 따라할 수 있도록 리눅스 스크립트들을 만들고 깃허브 저장소에 공개하였다(https://github.com/pkgpl/project_template).

리눅스 컨테이너

컨테이너 기술은 컴퓨터에 가상의 환경을 구축하는 가상화 기술의 일종이다. 가상화 기술은 호스트 가상화, 하이퍼바이저 가상화, 컨테이너 가상화가 대표적인데, 호스트 가상화는 호스트 운영체제 위에서 가상화 소프트웨어를 이용해 게스트 운영체제를 구동하는 방식으로, 가상화를 위한 컴퓨터 자원이 많이 필요한 기술이다. 하이퍼바이저 가상화는 하이퍼바이저라는 가상화 전문 소프트웨어를 이용해 호스트 운영체제 없이 가상 환경을 구현하는 기술이지만, 가상 환경마다 각각 운영체제를 구동하므로 여러 개의 가상 환경을 이용할 경우 부담이 커질 수 있다. 컨테이너 가상화 기술은 호스트 운영체제 상에서 호스트 운영체제 시스템과 격리된 프로세스들을 모아 개별 서버처럼 사용하는 기술로, 다른 가상화 기술들에 비해 가볍고 빠르다. 2000년대 초반부터 다양한 리눅스 컨테이너 기술들이 개발되었으나 2013년 도커의 등장으로 널리 사용되기 시작했다(Linux containers, 2021). 본 해설에서도 도커를 사용하였다. 도커는 리눅스 컨테이너지만 윈도우즈와 맥에서도 실행 가능하다.

컨테이너는 원하는 소프트웨어들을 설치해놓은 이미지 파일을 이용해 실행하는 프로세스로, 하나의 호스트 컴퓨터에서 여러 컨테이너를 동시에 실행할 수도 있다. 도커 이미지는 다른 사용자로부터 받거나 Dockerfile이라는 텍스트파일로부터 생성할 수 있으며 Dockerfile 또는 이미지를 공유하면 다른 시스템에 동일한 연구 환경을 구축할 수 있기 때문에 연구 결과 재현이나 소프트웨어를 이용한 협동 연구에 유용하다. 도커에서는 도커 허브(Docker Hub, 2021)를 통해 이미지를 쉽게 공개

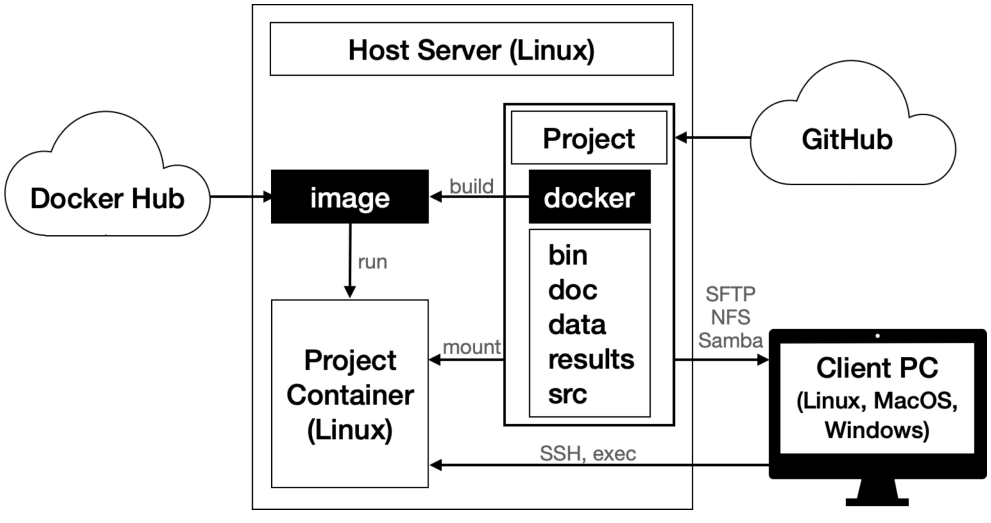

Fig. 1. System configuration used in this study.

Table 1. Important Docker commands (Docker, 2021).

Docker command	Description
docker build	Build an image from a Dockerfile
docker run	Run a command in a new container
docker exec	Run a command in a running container
docker images	List images
docker ps	List containers
docker stop	Stop one or more running containers
docker rmi	Remove one or more images

하고 공유할 수 있도록 하고 있으며, 도커 레지스트리를 이용하면 별도의 서버를 통해 이미지를 공유할 수도 있다.

Table 1에 본 연구와 관련하여 호스트 컴퓨터에서 실행하는 주요 도커 명령어들을 정리하였다. 본 연구에서는 도커를 쉽게 사용할 수 있도록 리눅스 쉘 스크립트들을 작성하여 프로젝트 템플릿과 함께 공개하였다.

버전 관리와 소스 코드 저장소

버전 관리 기술은 프로그램 소스 코드와 같은 디지털 문서의 변경 사항을 기록하고 저장하는 기술로, 이를 이용하면 소스 코드의 변경 사항을 확인하거나 이전 상태로 되돌릴 수 있다(Version control, 2021). 소스 코드 관리 소프트웨어에는 CVS, BitKeeper, Git (Git, 2021), Mercurial, Subversion 등 다양한 프로그램들이 있다. 소스 코드 관리 정보는 기본적으로 프로그램을 개발하고 있는 컴퓨터에 저장하지만 다른 서버나 클라우드에도 함께 저장할 수도 있다. BitBucket, GitHub (GitHub, 2021), GitLab과 같은 클라우드 서비스를 이용하면 프로그램을 이중으로 백업할 수 있고, 다른 사용자와 쉽게 공유할 수도 있다. 본 연구에서는 가장 많이 사용되는 깃(Git)과 깃허브(GitHub)를 기준으로 설명한다. 깃은 리눅스 커널 소스 코드 관리를 위해 개발된 소프트웨어로, 속도가 빠르고 CVS나 Subversion과 같은 중앙 집중식 버전 관리 소프트웨어와 달리 분산 버전 관리를 이용하므로 중앙 서버와의 연결 없이도 개발이 가능하다(Git, 2021). 각각의 클라이언트가 저장소 전체를 복사하여 가지고 있으므로 소스 코드 백업에도 유리하다. 깃허브는 깃 프로젝트를 지원하는 웹호스팅 서비스로, 깃 프로젝트의 원격 저장소로 사용할 수 있으며, 웹 사이트에서의 코드 편집을 지원한다. 또한 여러 사용자가 공동으로 작업하기에 좋도록 다양한 기능을 제공한다(GitHub, 2021). Table 2에 본 연구와 관련된 주요 깃 명령어들을 정리하였다.

버전 관리를 통해 관리할 파일은 소스 코드와 같은 텍스트 파일들이다. 이진 파일은 변경 내용을 추적 하더라도 사람이 변경 사항을 확인하기 어렵고, 파일 용량이 클 경우 클라우드 저장소와 통신하는데 시간이 오래 걸리게 된다. 이진 파일의 백업이 필요할 경우에는 별도의 컴퓨터나 저장 매체에 사본을 저장해놓는 것이 좋다.

Table 2. Important Git commands (Git, 2021).

Git command	Description
git clone	Clone a repository into a new directory
git add	Add file contents to the index
git status	Show the working tree status
git commit	Record changes to the repository
git log	Show commit logs
git push	Update remote references along with associated objects
git pull	Fetch from and integrate with another repository or a local branch
git diff	Show changes between commits, commit and working tree, etc.

연구 프로젝트

컨테이너와 버전 관리는 프로젝트 단위로 통합할 수 있다. 하나의 프로젝트는 하나의 연구 주제나 논문, 동일 소프트웨어 또는 데이터를 이용하는 연구, 수업 과목 등을 기준으로 생성할 수 있다. 하나의 프로젝트는 호스트 컴퓨터 상에서 하나의 디렉토리 내에 존재하는 하위 디렉토리와 파일들로 구성되며, 이는 하나의 소스 코드 저장소와 하나의 컨테이너에 대응한다. 하위 디렉토리들은 Table 3과 같이 구성할 수 있다. 본 연구에서는 Wilson *et al.* (2017)이 제안한 디렉토리 구조에 도커 관리를 위한 docker 디렉토리를 추가하였다. Wilson *et al.* (2017)은 소프트웨어 연구 프로젝트 운영과 관련된 사항들을 설명하였다. 본 연구에서는 도커 컨테이너와 버전 관리 시스템을 이용해 소프트웨어 연구 프로젝트 환경을 구축하는 방법에 대해 다룬다.

연구 작업 절차

본 연구에서는 도커를 사용해본 적이 없더라도 컨테이너를 쉽게 시작할 수 있도록 깃허브에 프로젝트 템플릿을 만들고 Table 1의 도커 명령어들을 기반으로, Dockerfile, Env.sh,

Table 3. Project layout (Wilson *et al.*, 2017).

Directory	Description
project_directory	Project root directory
|-- bin	Executables
|-- data	Research data
|-- doc	Text documents
|-- results	Research analysis results
|-- src	Program source codes
|-- docker	Dockerfile and docker-related scripts

Table 4. Workflow using the project template.

Stage	Workspace	Work
Initialization	GitHub	1. Fork the project template in GitHub
	GitHub	2. Create a new GitHub repository using the template
	Host	3. Clone the repository to a local machine $ git clone <github_path> <host_project_directory>
	Host	4. Edit environment files (Env.sh, Dockerfile, README.md)
	Host	5. Build project image $ ./build.sh
	Host	6. Track changes (git add, commit, and push)
Daily research	Host	7. Start the project container $ ./run.sh
	Host/Container	8. Work in the project container $ ./exec.sh
	Host	9. Stop the project container $ docker stop <project_name>
Finalization	Host	10. Remove the project image $ docker rmi <project_name>

build.sh, run.sh 및 exec.sh 등과 같은 리눅스 스크립트 파일을 작성해 놓았다. 이 파일들을 이용한 새 프로젝트 생성 및 작업 절차는 Table 4와 같다. 1번 과정은 단 한 번만 실행하면 되고, 2번부터 5번까지는 새로운 프로젝트를 시작할 때 한 번씩만 실행한다.

1. 깃허브에 계정을 만들고 로그인한 후 서론에 제시한 프로젝트 템플릿을 본인의 저장소로 복사(fork)한다. 이 작업은 처음 한 번만 수행하면 되고, 두 번째 프로젝트 생성시에는 2번 과정부터 진행하면 된다.

2. 본인의 깃허브 계정에서 새로운 저장소를 생성한다. 이때 1번에서 복사한 저장소를 템플릿으로 지정한다. 새로 만드는 저장소는 원할 경우 비공개로 설정할 수 있다.

3. 새로 만든 클라우드 저장소를 호스트 컴퓨터에 복사한다. 명령어는 다음과 같다.

```
$ Git clone <github_path> [<host_project_directory>]
```

위에서 [ ] 안의 내용은 생략 가능하며, 생략시 깃허브 저장소 이름과 같은 이름의 디렉토리가 생성된다. 만약 git을 처음 사용한다면 다음과 같이 깃 사용 환경을 설정한다. 여기서 지정하는 이름과 메일 주소는 버전 관리 시스템에서 파일을 생성하고 수정한 사람이 누구인지 파악할 때 사용된다.

```
$ git config --global user.name "My Name"
$ git config --global user.email "email@address.com"
```

4. 프로젝트 환경을 설정한다. 3번 과정에서 생성한 프로젝트 디렉토리 내 README.md 파일을 프로젝트에 맞게 수정한다. 이후 docker 디렉토리로 이동 후 Env.sh 파일을 열어 프로젝트명, 기본 도커 이미지 및 주피터 노트북/랩 및 텐서보드 관련 설정을 수정한다. 프로젝트명은 공백 없이 소문자로 지정한다. 여기서 지정한 프로젝트명은 도커 이미지 이름, 컨테이너 이름 및 컨테이너 내 프로젝트 디렉토리명으로 사용된다. 기본 도커 이미지는 도커 허브에서 본인의 연구 목적에 맞는 이미지를 찾아 주소를 적으면 된다. 템플릿에서 기본적으로 사용하는 도커 이미지에는 텐서플로우, 파이토치, 사이킷런, 판다스, 맷플롯립 등 파이썬을 이용한 머신러닝에 필요한 패키지들이 설치되어 있으므로 머신러닝 작업을 위한 프로젝트의 경우 템플릿의 기본 도커 이미지를 그대로 사용할 수 있다.

필요할 경우 Dockerfile 파일을 열어 추가 설치 프로그램 지정한다. 프로그램 설치와 같이 이미지 생성시 실행하고자 하는 명령어는 RUN 명령 뒤에 추가하면 되고, 환경 설정을 변경하고자 하는 경우에는 ENV 명령 뒤에 추가하면 된다. 프로그램 설치를 위해 호스트 디렉토리의 파일을 컨테이너 안으로 복사할 경우 COPY 명령을 사용할 수 있다.

프로젝트 템플릿에 기본으로 제공되는 Dockerfile에서는 컨테이너 내의 사용자를 생성한다. 별도 사용자 없이 관리자 권한으로 컨테이너를 실행할 수도 있지만 리눅스에서 관리자 권한으로 작업하는 것은 일반적으로 권장하지 않는다. 컨테이너 내의 사용자는 호스트 사용자와 동일한 사용자 아이디 및 그룹 아이디를 사용하도록 만들었다. 이를 통해 컨테이너 내에서 마운트한 디렉토리에 생성한 파일이 호스트에서 봤을 때 다른 사용자가 생성한 것처럼 나오는 것을 방지한다. 이 외에도 호스트의 리눅스 명령줄, 컨테이너 사용자의 명령줄, 컨테이너

관리자의 명령줄이 시각적으로 구분되도록 명령줄 프롬프트를 지정하고 프로젝트 내의 bin 디렉토리를 PATH 환경에 추가한다.

5. 프로젝트 환경 설정 후에는 도커 이미지를 생성한다. 명령어는 다음과 같다.

$ bash ./build.sh

리눅스 스크립트들에 실행 권한을 추가하여 놓았으므로 위 명령에서 bash는 생략할 수 있다. 위의 과정을 마치면

$ docker images

를 통해 프로젝트명으로 된 이미지를 확인할 수 있다.

6. 수정한 파일들을 깃 버전 관리 시스템에 등록한다.

$ git add <modified_files>

위의 명령은 수정한 파일들을 관리하겠다는 의미이고, 아래와 같이 커밋을 실행해야 변경 사항이 기록된다.

$ git commit -m “my commit message”

커밋 후에는

$ git log

를 통해 커밋 내역을 확인할 수 있다.

$ git status

를 실행하면 클라우드 저장소에 비해 커밋이 앞서있다고 나올 것이다. 아래 명령을 통해 클라우드 저장소에도 변경 사항을 기록한다.

$ git push

특정 파일의 변경 사항을 확인하기 원할 경우 git diff를 사용하면 된다.

5번까지의 내용은 프로젝트 생성시 한 번만 수행하면 되고, 6번 과정은 주요 수정 사항이 있을 때마다 실행한다. 일상적인 연구 절차는 다음과 같다.

7. 생성한 이미지를 이용해 컨테이너를 백그라운드로 실행한다.

$ ./run.sh [option]

컨테이너 내에서 기본 작업 디렉토리는 /home/<username>/<project_name> 디렉토리로, Env.sh에서 지정한 프로젝트명을 디렉토리 이름으로 사용한다. 이 디렉토리에 호스트의 프로젝트 디렉토리가 마운트된다. 또한 컨테이너 내에서 실행할 주피터 랩 및 텐서보드에 호스트 웹브라우저를 이용해 접근할 수 있도록 해당 포트들을 호스트와 연결한다. 기본 포트는 8888과 6006이지만 필요에 따라 Env.sh 파일에서 호스트 포트 번호를 수정할 수 있다. 컨테이너 내에서 리눅스 X 윈도우를 이용하는 프로그램을 실행할 경우에는 위 명령에 x11 옵션을 추가한다. x11 옵션을 추가할 경우 컨테이너에서 호스트의 네트워크 전체에 접근하게 되는데, 텐서보드 작업을 여러 개 동시에 실행해서 여러 개의 포트가 필요할 경우에도 사용할 수 있다. 컨테이너 실행 상태는

$ docker ps

명령으로 확인할 수 있다.

8. 컨테이너 내에서 연구를 수행한다. 다음 명령이 가장 많이 사용하게 될 명령이다.

$ ./exec.sh [option]

위 명령을 옵션 없이 실행하면 컨테이너 내의 리눅스 쉘에 접속하게 된다. 시작 위치는 7번에서 설명한 작업 디렉토리이고, 목적에 맞는 하위 디렉토리에서 연구를 진행한다(Table 3). 편의를 위해 주피터 노트북 또는 주피터 랩을 백그라운드작업으로 진행할 수 있도록 위 명령 뒤에 notebook 또는 lab 옵션을 추가할 수 있다. 이 때 호스트 컴퓨터의 웹브라우저를 이용해 localhost:8888 주소 또는 Env.sh에서 지정한 포트로 접근할 수 있다. 접근에 필요한 비밀번호는 Env.sh 파일 내에서 지정한다. 컨테이너 내에서 텐서보드 실행시 같은 방식으로 호스트 컴퓨터의 웹브라우저를 이용해 텐서보드 포트(localhost:6006)에 접근할 수 있다.

아래와 같이 컨테이너의 리눅스 쉘을 나가면 컨테이너는 실행중인 상태이기 때문에 다시 exec.sh를 통해 컨테이너에 접속할 수 있다.

(docker) $ exit

주요 변경 사항이 있을 때마다 컨테이너가 아닌, 호스트 컴퓨터에서 6번 과정을 반복한다. 컨테이너 내 프로젝트 디렉토리와 하위 디렉토리들 내에서 작업한 내용은 호스트 컴퓨터의 버전 관리 하에 있는 디렉토리에 기록되므로 호스트에서 변경 사항을 기록한다.

9. 컨테이너를 완전히 종료하고 싶다면 호스트에서

$ docker stop <project_name>

을 실행하면 된다. 이후 컨테이너를 다시 실행할 경우 7번 과정부터 시작하면 된다.

10. 프로젝트 종료로 더이상 해당 이미지가 필요 없을 경우

```
$ docker rmi <project_name>
```

를 실행하면 되고, 필요할 경우 삭제한 이미지는 5번 과정을 통해 다시 생성할 수 있다.

활용 사례

본 해설의 내용은 컴퓨터를 활용하는 다양한 분야의 연구와 교육에서 사용될 수 있다. 서론의 깃허브 저장소에 프로젝트 템플릿을 이용한 교육 및 연구 예시 프로젝트들과 프로젝트 생성 동영상 설명을 공개하였다.

먼저 머신러닝 강의에 사용했던 암상분류 과제 예시를 공개하였다. 이 예시에서 1번 과정은 불필요하였고, 깃허브에서 새로운 저장소를 만든 후(2번 과정) 호스트 컴퓨터에 받았다(3번 과정). 호스트 컴퓨터의 저장소에서 README.md 파일을 수정하고 Env.sh에서 프로젝트명을 수정하였다. 사이킷런과 텐서플로우 등 머신러닝 패키지들을 설치해놓은 기본 도커 이미지를 사용하였으므로 Dockerfile은 따로 변경하지 않았다(4번 과정). 도커 이미지 생성 후(5번 과정) data 디렉토리에 필요한 자료를 받고 호스트 컴퓨터에서 깃을 이용해 변경 사항을 저장하였다. data 디렉토리에서 모든 작업을 진행하였으며, 필요 없는 디렉토리들은 삭제하였다(6번 과정). 컨테이너 실행 후에는(7번 과정) 주피터 노트북으로 데이터를 분석하였다(8번 과정). 분석 후에는 컨테이너를 멈추고(9번 과정), 이미지를 삭제하였다(10번 과정).

풍력 발전을 위한 풍속 자료 분석 사례도 함께 공개하였다. 깃허브에 저장소를 만들고 호스트에 받아오는 과정은 앞의 예제와 동일하다(2, 3번 과정). 프로젝트 명 수정 후 엑셀 파일 입력을 위해 Dockerfile에서 필요한 패키지를 설치하였다(4번 과정). 도커 이미지 생성 후(5번 과정) data 디렉토리에 풍속 자료를 받고 수정 사항을 깃으로 저장하였다(6번 과정). 컨테이너 실행 후(7번 과정) 주피터 노트북을 통해 src 디렉토리에서 와이블 분포와 레일리 분포를 이용하여 주어진 풍력 터빈 정보로부터 풍력 발전량을 예측하였다(8번 과정). 데이터 분석 후에는 컨테이너를 멈췄고(9번 과정), 도커 이미지는 나중에 다시 사용하기 위해 남겨두었다.

탄성파 탐사 완전파형역산 연구 예시도 공개하였다. 먼저 깃허브 저장소를 만들고 호스트 컴퓨터에 받았다(2, 3번 과정). Env.sh에서 프로젝트명을 지정하고 Dockerfile을 수정하여 컨테이너에 파이토치 기반 완전파형역산 패키지를 설치하였다. 기본 도커 이미지는 탄성파 자료처리 패키지들이 설치되어 있는 이미지를 사용하였다(4, 5 번 과정). 호스트 컴퓨터에서 bin 디렉토리에는 파동 전파 모델링 및 역산 프로그램을 복사하고 data 디렉토리에 벤치마킹 모델을 복사하였다. 변경 사항은 깃을 이용해 저장하였다(6번 과정). 컨테이너 실행 후에는(7번 과정) 컨테이너에 접속하여 data 디렉토리에서 관측 자료를 생성하였다. src 디렉토리에서 파이토치의 다양한 옵티마이저를 이용해 완전파형역산을 수행하며 텐서보드를 이용해 모니터링하고 results 디렉토리에서 최종 결과를 정리하였다(8번 과정). 세부 사항은 깃허브 사이트에서 직접 확인할 수 있다.

토 의

도커 이미지 변경 사항

도커 컨테이너 내에서 프로젝트 디렉토리 외의 다른 위치에서 파일을 생성하거나 변경할 경우 컨테이너를 중지하면 해당 변경 사항은 사라지게 된다. 따라서 저장이 필요한 작업은 프로젝트 디렉토리 내에서만 실행하도록 한다. docker commit 명령을 통해 변경 사항을 이미지에 저장할 수도 있지만 이 경우 변경 사항 추적이 안 되므로 재현성을 해치게 된다. Docker 컨테이너 내부 환경에 수정이 필요할 경우 Dockerfile을 통해 변경하고 이미지를 새로 생성하는 것이 바람직하다. 이미지를 처음 생성할 때에는 기본 이미지를 받아오느라 시간이 오래 걸릴 수 있으나 Dockerfile에서 일부만 수정하여 이미지를 다시 생성할 때에는 캐시를 이용하므로 빠르게 생성할 수 있다. 동일한 기본 이미지를 이용하는 새로운 프로젝트 이미지 생성 또한 캐시를 이용해 빠르게 진행된다.

클라이언트에서의 접속

클라이언트 컴퓨터에서 호스트 서버 컴퓨터로 SSH (Secure Shell)를 이용해 접속하고 작업하는 경우 SSH 터널링을 이용하면 컨테이너에서 실행한 주피터 랩 또는 텐서보드에 윈도우즈/맥/리눅스 클라이언트 컴퓨터의 웹브라우저를 통해 접속할 수 있다. SSH 터널링을 이용한 포트 접속은 서버에서 실행한 웹브라우저를 SSH를 통해 클라이언트에서 보는 것보다 속도가 빠르다. 컨테이너에서 주피터 랩을 실행한 후 클라이언트 컴퓨터의 명령줄에서 실행하는 SSH 터널링 명령어는 다음과 같다.

```
$ ssh -N -L 8888:localhost:8888 user_id@host_server
```

위 명령을 이용해 서버에 접속하면 클라이언트 컴퓨터의 웹브라우저에서 localhost:8888 주소로 컨테이너의 주피터 랩에 접속할 수 있다. 텐서보드의 경우 포트 주소만 6006으로 바꾸면 된다. 텐서보드 포트 번호, 주피터 랩의 포트 번호 및 접속 비밀번호는 Env.sh 스크립트에서 수정할 수 있다. 한 번에 여러 개의 컨테이너를 동시에 실행시킬 경우에는 프로젝트마다 주피터 랩 및 텐서보드 연결 포트 번호를 다르게 지정해야 한다. 호스트 서버 앞에서 직접 작업하는 경우 또는 클라이언트 컴퓨터에서 원격 화면 공유 프로그램을 이용해 호스트 서버에 접속한 경우에는 SSH 터널링 과정이 불필요하다.

호스트와 클라이언트 컴퓨터 사이에 파일을 주고 받아야 할

경우 SFTP (SSH File Transfer Protocol)를 이용할 수 있다. 호스트 컴퓨터의 파일을 편집할 경우 vim과 같은 호스트 컴퓨터의 편집기를 이용하게 되는데 경우에 따라 클라이언트의 편집기나 통합 개발 환경(integrated development environment)을 이용하길 원할 수 있다. 통합 개발 환경에서 원격 작업을 지원할 경우 해당 기능을 이용할 수도 있지만 호스트의 디렉토리를 클라이언트 컴퓨터에 마운트하면 클라이언트 컴퓨터에 있는 파일처럼 편리하게 호스트 컴퓨터의 파일을 관리할 수 있다. 이를 위해서는 네트워크 파일 시스템(Network File System, NFS) 또는 삼바(Samba, 2021)와 같은 기술을 사용할 수 있다.

파이썬 가상 환경

파이썬을 사용할 경우 venv (Venv, 2021) 또는 conda (Conda, 2021)를 이용해 자체적으로 격리된 환경을 생성할 수 있다. 본 연구에서는 도커 컨테이너 자체가 격리된 가상 환경이므로 별도의 파이썬 가상 환경을 사용하지 않았다.

이미지 공개

이미지를 만들어 공개할 경우에는 라이센스 문제가 발생하지 않도록 주의해야 한다. 상용 프로그램의 경우 라이센스를 포함해서는 안 된다. 오픈 소스 프로그램도 재배포를 제한하는 경우가 있으므로 Dockerfile 설정시 리눅스, 파이썬 등의 패키지 설치 프로그램을 이용하거나 프로그램 홈페이지 또는 깃허브와 같은 공식적인 경로로 파일을 받아서 설치하도록 만드는 것이 좋다.

생략한 내용

도커와 깃에서는 위에 설명한 것 외에도 다양한 기능들을 제공하므로 다른 기능이 필요할 경우 홈페이지 문서를 참고할 수 있다(Docker, 2021; Git, 2021). 도커 Swarm 또는 쿠버네티스(Kubernetes, 2021)를 이용하면 여러 대의 노드로 구성된 클러스터 서버 시스템에서 컨테이너 작업을 관리할 수 있다. 윈도우즈 호스트의 경우 윈도우즈 도커 컨테이너를 실행하는 것도 가능하므로 윈도우즈 소프트웨어가 필요할 경우에 사용할 수 있다.

프로젝트 템플릿을 이용할 경우 소스 코드 저장소를 깃허브에서 만들기 때문에 깃의 초기화 명령을 따로 사용하지 않았지만, 컴퓨터 내의 임의의 디렉토리에서 저장소를 만들고 싶을 경우 git init 명령을 사용할 수 있다. 규모가 있는 개발 프로젝트의 경우 git branch 기능을 사용하는 것이 좋다. 이전 작업으로 돌아가고자 할 때에는 목적에 따라 checkout, reset, 또는 revert 등의 명령을 사용할 수 있다.

호스트 컴퓨터 관리자 작업

본 해설은 관리자가 아닌 사용자를 위해 작성하였으나 별도의 관리자가 없는 경우를 위해 도커 설치에 대해 간략히 설명한다.

도커는 도커 홈페이지에서 운영체제에 맞는 버전을 받아서 설치한다(Docker, 2021). Nvidia GPU를 사용하는 서버의 경우 nvidia 드라이버를 설치한 후 nvidia-docker (Nvidia Container Toolkit, 2021)를 설치하고 도커 daemon의 default-runtime으로 nvidia를 사용하도록 설정한다. 세부 사항은 프로젝트 템플릿 저장소의 docker/admin에서 찾을 수 있다.

도커 사용자는 리눅스 호스트 컴퓨터의 docker 그룹에 속해 있어야 한다. 도커를 처음 설치했을 경우 docker 그룹을 생성하고 docker 그룹에 컨테이너를 사용할 사용자들의 아이디를 추가한 후 도커 서비스를 시작한다. 도커 서비스 시작 후 docker 그룹에 사용자를 나중에 추가한 경우에는 도커 서비스를 재시작한다. 자세한 사항은 도커 문서(Docker, 2021) ‘Post-installation steps for Linux’에서 찾을 수 있다.

결 론

도커 컨테이너를 이용하면 여러 컴퓨터에서 동일한 연구 환경을 쉽게 생성할 수 있으므로 연구 재현성을 높이고 한 연구 그룹 내에서 또는 타 연구 그룹과의 협동 연구를 쉽게 시작할 수 있다. 클라우드 소스 코드 저장소를 사용하면 프로그램 소스 코드 변경 사항을 쉽게 관리하고 백업 및 공유할 수 있다. 많은 공학 및 과학 연구가 소프트웨어를 이용해 진행되는 상황에서 이러한 기술 없이도 연구를 수행할 수 있지만 컨테이너와 소스 코드 저장소를 이용하면 재현성 및 연구 환경 설정, 소스 코드 백업 등과 관련해 발생할 수 있는 문제들을 예방하고 공동 연구를 쉽게 시작할 수 있다. 해외 대학이나 연구소의 경우 연구 전문가(technician)들이 본 해설의 내용을 제공해줄 수 있지만 연구 전문가들이 부족한 국내 상황에서 연구자들이 연구를 위해 각자 이러한 내용을 찾아서 습득해야 한다면 국가적인 인력 낭비라 할 수 있다. 본 해설에서 정리해놓은 내용들이 소프트웨어와 프로그래밍을 이용한 연구와 교육을 좀 더 시작하기 쉽고, 재현 가능하고, 안전하게 만드는데 도움이 되기를 기대한다.

감사의 글

다양한 오픈 소스 소프트웨어들의 개발자들에게 감사합니다.

References

Andreessen, M., 2011, *Why Software Is Eating The World*, The Wall Street Journal, http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html (April 15, 2021 Accessed)

Conda, 2021, https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html (April 15, 2021 Accessed)

Dionisio, J. D. N., Burns III, W. G., and Gilbert, R., 2013, 3D virtual worlds and the metaverse: Current status and future possibilities, *ACM Comput. Surv.*, **45(3)**, p.1-38, doi: 10.1145/2480741.2480751.

Docker, 2021, https://docs.docker.com (April 15, 2021 Accessed)

Docker Hub, 2021, https://hub.docker.com (April 15, 2021 Accessed)

Git, 2021, https://git-scm.com/docs (April 15, 2021 Accessed)

GitHub, 2021, https://github.com (April 15, 2021 Accessed)

Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl D., and Wilson, G., 2009, How do scientists develop and use scientific software?, *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, Vancouver, BC, Canada, p.1-8, doi: 10.1109/SECSE.2009.5069155.

Kubernetes, 2021, https://kubernetes.io (April 15, 2021 Accessed)

Linux containers, 2021, https://linuxcontainers.org (April 15, 2021 Accessed)

Morris, K., 2016, *Infrastructure as code: Managing servers in the cloud*, O'Reilly.

Negri, E., Fumagalli, L., and Macchi, M., 2017, A review of the roles of digital twin in CPS-based production systems, *Procedia Manufacturing*, **11**, 939-948.

Negroponte, N., 1995, *Being Digital*, Alfred A. Knopf, Inc., USA.

Nvidia Container Toolkit, 2021, https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.htm (April 15, 2021 Accessed)

Prabhu, P., Kim, H., Oh, T., Jablin, T. B., Johnson, N. P., Zoufaly, M., Raman, A., Liu, F., Walker, D., Zhang, Y., Ghosh, S., August, D. I., Huang, J., and Beard, S., 2011, A survey of the practice of computational science, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WA, USA, p.1-12, doi: 10.1145/2063348.2063374.

Samba, 2021, https://www.samba.org (April 15, 2021 Accessed)

Tercek, R., 2015, *Vaporized: Solid stragegies for success in a dematerialized world*, LifeTree Media, USA.

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., and Teal, T. K., 2017, Good enough practices in scientific computing, *PLoS Comput. Biol.*, **13(6)**, e1005510, doi: 10.1371/journal.pcbi.1005510.

Venv, 2021, https://docs.python.org/3/library/venv.html (April 15, 2021 Accessed)

Version control, 2021, https://en.wikipedia.org/wiki/Version_control (April 15, 2021 Accessed)