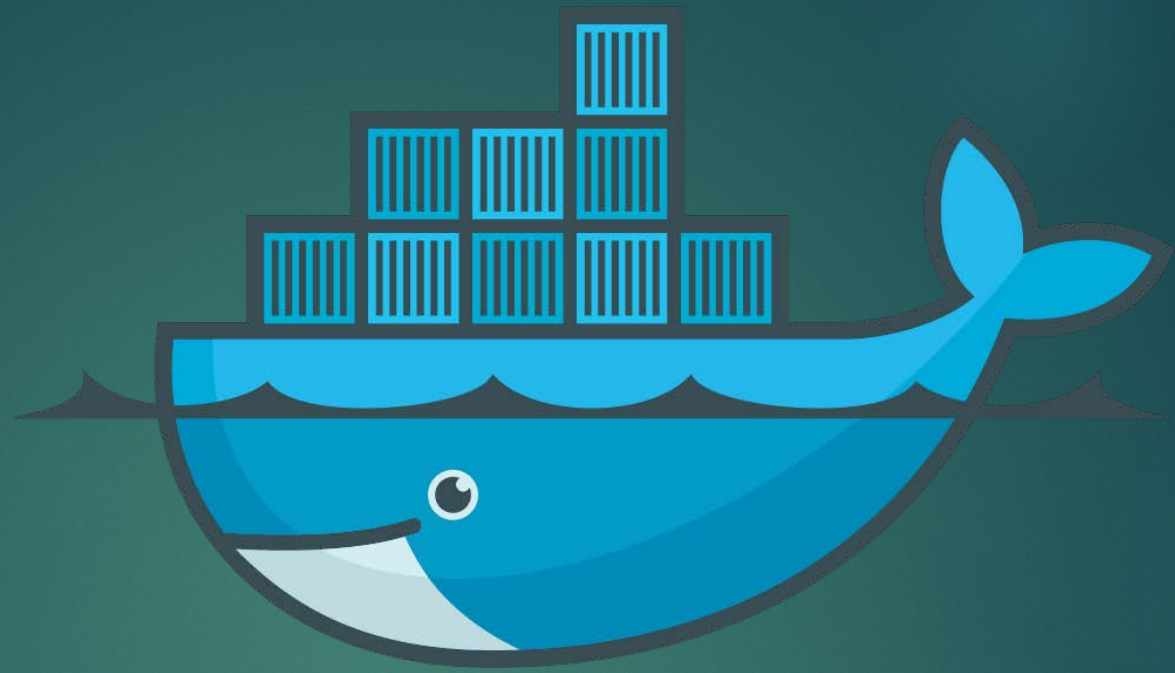


Docker w/ Tutorial

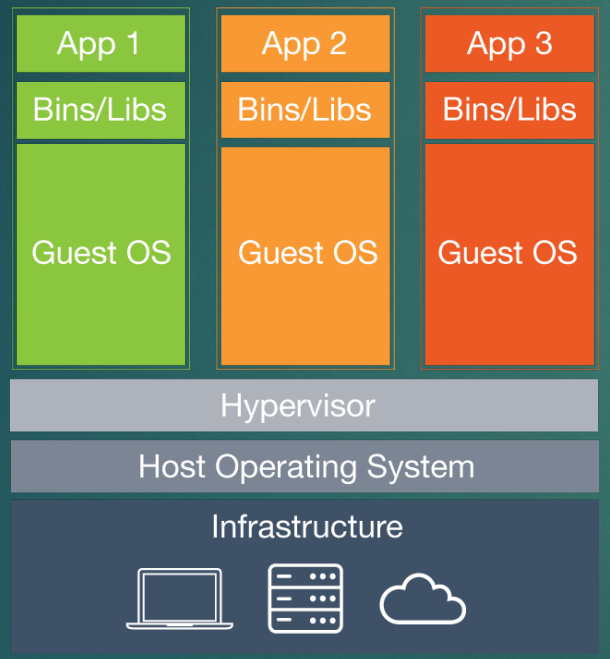
SPARCS WHEEL SEMINAR 2015-07-20 SAM JO

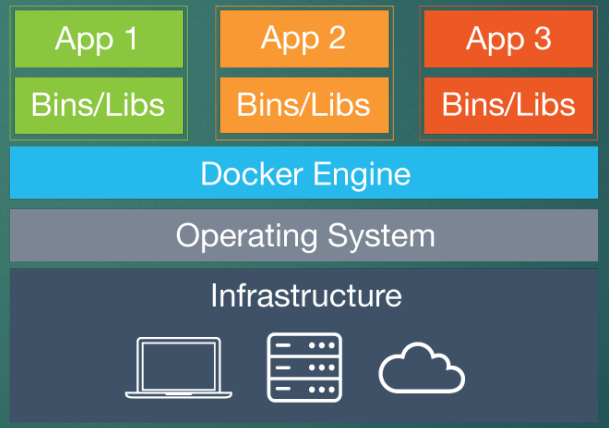
Index

- Docker?
- Tutorials
  - Search & Pull & Run
  - Operations
  - Commit
  - Build with Dockerfile

Container-based Virtualization

- Xen과 같은 Virtual Machine과는 다르게 가상 하드웨어를 시뮬레이트하지 않고 OS가 사용하는 자원을 분리하여 가상화





1 VM = 1 OS + Libs + Apps

1 Container = Libs + Apps

Container-based Virtualization

- Xen과 같이 완전한 guest OS를 구동시키는 것이 아니라 host OS에서 각 container들이 자신만의 process와 network 공간을 가질 수 있도록 하는 가상화 방법

- Kernel에서 해당 기능을 지원해야 사용할 수 있음
  - ex: Debian 6.0에는 Docker 사용 불가!

- LXC, Docker, OpenVZ 등이 이러한 가상화 방법을 사용함

How Container Works?

- File: process의 root directory를 변경하여 특정 directory를 root directory로 보이게 함
- Process: container의 process들을 격리시켜 다른 container에서 실행되는 process들을 보이지 않게 함
- Network: “veth”라는 가상 장치를 만들어 물리적 장치에서 오는 패킷을 포워딩함
- CPU, Memory: cgroups등의 하드웨어 제어 도구를 사용하여 container에서 사용할 수 있는 범위를 제한

Container Software

- LXC (LinuX Container)
  - Developed by Parallels, IBM, Google, ...
  - Written in C, Python, Lua
- Docker
  - Developed by Docker, Inc.
  - Written in Go
- OpenVZ: community project, written in C

Container vs VM

- Container Preferred
  - 속도가 중요할 때 (하드웨어 시뮬레이팅을 하지 않으므로 속도가 빠름)
  - 사용할 수 있는 용량이 제한되어 있을 때 (guest OS가 없으므로 용량이 적음)

- Virtual Machine Preferred
  - 보안이 상당히 중요할 때 (kernel 자체가 분리되어있으므로 특정 kernel의 보안 취약점이 다른 container에 영향을 미치지 않음)
  - Host OS에서 가상화 대상의 Kernel을 지원하지 못할 때

LXC (LinuX Container)

- An **operating-system-level** virtualization environment for running multiple isolated Linux systems on a single Linux host

- Linux kernel은 resource를 제한 및 분리시킬 수 있는 cgroups라는 기능을 제공하는데, 이를 사용하여 container를 구현

- 각 container는 system resource (CPU, Memory) 뿐만 아니라 파일, process 및 network도 제한 및 분리되어 있음

Docker

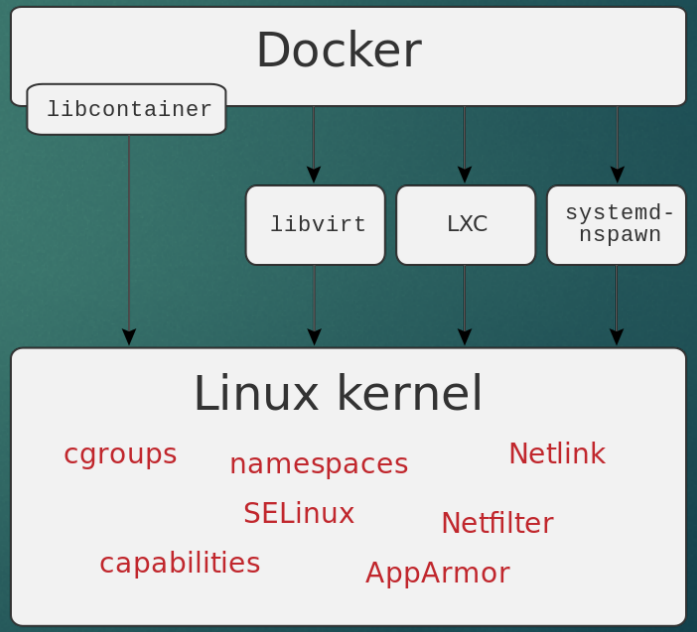
“Docker is an open platform for **building, shipping and running** distributed applications. It gives programmers, development teams and operations engineers the common toolbox they need to take advantage of the distributed and networked nature of modern applications.”

“Docker **containers wrap up** a piece of software in a complete filesystem that **contains everything** it needs to run: **code, runtime, system tools, system libraries** – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.”

Docker How to?

- libvirt, LXC나 system-nspawn 같은 다른 container library를 통해 Linux kernel의 가상화 기능에 접근

- Docker 0.9 버전부터는 자체적인 libcontainer라는 library를 통해 가상화 기능을 구현할 수 있음



LXC? Docker?

- Docker는 여러가지 high-level한 도구들을 지원함
  - Automatic Build: Dockerfile을 사용하여 자동으로 container를 만들 수 있음
  - Versioning: git과 같이 image의 version 관리를 할 수 있음
  - Component re-use: 기본이 되는 image를 여러 개의 container로 분화시킬 수 있음 (ex: Django가 설치되어 있는 image를 가지고 ara, otl을 서비스하는 container를 만들 수 있음)
  - Sharing: docker-hub를 사용하여 유용한 container들을 공유할 수 있음

  - https://docs.docker.com/misc/faq/#what-does-docker-add-to-just-plain-lxc

With Tutorial!

- Docker Version: 1.7.0

- **bold orange color** – 이 명령어를 실행시켜주세요!
  - **sudo** 는 생략되어 있습니다! **docker** 앞에는 **sudo**를 붙여주세요!
  - **(shell #1), (shell #2)** 이 붙은 경우 shell을 여러 개 켜서 각각 실행해주세요!

- **는 휠 세미나 서버에서 따라하지 마세요.**
  - 휠 세미나 서버의 커널 버전이 낮고, Xen위에서 돌아가는 서버라 docker를 실행하는데 무리가 있습니다. (삽질을 경험하실 수 있습니다.)
  - 그러므로 이 세미나는 여러분이 직접 하셔야 하는 실습이 없고, 발표자 서버에서 돌아가는 모습을 구경해주시면 됩니다.

Install

- OS: Ubuntu 14.04 LTS

- **apt-get install docker.io** (1.7 버전을 사용합니다.)
- **docker info**

- http://docs.docker.com/mac/step_one/ (for Mac)
- http://docs.docker.com/windows/step_one/ (for Windows)

Commands (v1.7)

events / **version** / info: docker의 이벤트 / 버전 / 정보 출력

search / **pull** / push [IMGAGE]: docker registry hub에서 image를 검색 / 다운 / 업로드

images / **ps**: local에 있는 image / container의 목록을 출력

run [IMAGE]: 이미지로부터 새 container을 만듬

start / **stop** / restart / kill / **rm** [CONTAINER]: container을 시작 / 정지 / 재시작 / 강제 정지 / 삭제

top / pause / unpause / wait [CONTAINER]: container 안에 있는 프로세스 조회 / 모두 정지 / 해제 / 종료시까지 대기

save / **rmi** / history [IMAGE]: image를 tar 파일로 저장 / 삭제 / 기록 조회

attach / export / **diff** [CONTAINER]: container에 접속 / 파일로 저장 / 변경된 파일 목록 /

logs / inspect / **commit** [CONTAINER]: container 로그 조회 / 저수준 정보 조회 / image 생성

build / import / load [FILE or URL]: dockerfile로부터 이미지 빌드 / 파일 시스템 이미지 생성 / 이미지 로드

cp [CONTAINER]:[PATH] [HOSTPATH]: container으로부터 host로 파일 복사

port [CONTAINER] [PORT]: container안의 특정 port가 어디로 연결되어 있는지 조회

tag [IMAGE] [NAME]: image에 name을 태그

login: docker registry hub에 로그인 또는 가입

Commands

- 전 슬라이드에서 **이 색깔**의 명령어만 실습 예정입니다.
  - **version / search / pull / images / ps / run / start / stop / rm / top / rmi / attach / diff / commit / build / cp**

- 더 자세한 설명은 다음 링크를 참고해 주세요!
  - **https://docs.docker.com/**
  - **https://gist.github.com/nacyot/8366310**

Image? Container?

- image vs container = .exe vs process = object vs class
  - Image: binaries, libraries, applications를 설치한 뒤 파일로 만든 것
  - Container: image를 실행한 상태

- 1 image -> many containers!
  - run [OPTION] IMAGE

- ??????? -> 1 image
  - 네 가지의 방법: import & load, search, commit, build

Docker File System

- Union File System (Union FS)
  - layer을 생성하며 운영할 수 있는 file system으로, 가볍고 빠름
  - Image에서 container을 생성하는데 사용됨

- Data Volume
  - 여러 container에서 동시에 접속 가능하도록 만든 volume
  - Container가 생성 / 변경 / 삭제되는 것에 영향을 받지 않음
  - 어떠한 container도 사용하지 않는다면 garbage collect됨

Docker Network

- docker0라는 이름의 virtual interface를 host에 생성함

```
samjo@samnote:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.56847afe9799       no              veth352e
```

- docker0는 가상 Ethernet bridge으로 다른 interface로 들어오는 패킷을 container들로 포워딩함

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:587           0.0.0.0:*               LISTEN      2073/sendmail: MTA:
tcp6       0      0 :::80                   :::*                    LISTEN      19590/docker.io
```

- Container들은 172.17.42.1/16의 private ip를 할당받음

Image import & load

- import: 빈 파일 시스템 이미지를 만들고, 웹 또는 .tar에서 가져온 파일을 파일 시스템에 추가한다.
  - https://docs.docker.com/reference/commandline/import/
- load: 이미 만들어진 docker image를 local 목록에 추가한다.
  - https://docs.docker.com/reference/commandline/load/
- 이 세미나에서는 직접 실습하지 않음

Image Search & Pull

- Docker registry hub으로부터 image를 검색 / 다운

- search <name> - Docker Hub에서 image 검색
  - Docker Hub: https://registry.hub.docker.com/search?q=library
  - Official images + 수많은 custom images
  - Ubuntu, Debian, CentOS, etc…

- pull <name>:<tag> - 해당하는 image 다운
- **docker search ubuntu**
- **docker pull ubuntu:latest**

Image List

- images – local에 존재하는 image 목록 출력
  - -a: 빌드 과정에 사용된 중간 이미지를 포함하여 모든 이미지 출력
  - Docker는 image를 빌드할 때 RUN 명령마다 image를 생성
  - 빌드 순서를 기억하고 중간 이미지를 보존하기 때문에 같은 과정에서는 캐시된 이미지를 사용하여 빠른 빌드가 가능함

- **docker images**
- **docker images -a**

Container Run!

- run <option> <name> <file>
  - Image로부터 container를 만든 뒤, 해당 container에서 file을 실행
  - -i: interactive / -d: demon
  - -t: pseudo-tty / -p <in>:<out>: port forward, -P: auto port forward
  - -v <host-dir>:<dir>: host-dir을 dir에 mount
  - --name <name>: container에 이름 지정

Container Run!

- **mkdir ~/docker/; touch ~/docker/abc.txt**

- **docker run -v ~/docker:/data -i -t --name=nyang ubuntu:latest /bin/bash**
  - Nyang라는 이름의 Container가 하나 생성됨
  - ~/docker directory가 container 안의 /data directory로 마운트됨
  - Container의 shell이 뜨는 것을 확인할 수 있음

- **cat /etc/issue/**
- **ls -al /data/**
- **touch /home/docker/nyang.nyang**

Container List

- ps – 모든 container 목록을 출력
  - -a: 종료된 container 목록까지 모두 출력

- **(shell #2) docker ps**

- **(shell #1) exit**
- **(shell #2) docker ps**
- **(shell #2) docker ps -a**
  - Nyang container가 종료되었으므로 -a 옵션을 줘야만 목록에 나타남

Operations on Container

- start / stop / restart / kill / rm <name>
  - container 시작 / 정지 / 재시작 / 강제 정지 / 삭제
  - 강제 정지란 SIGKILL을 container에 보내는 것을 뜻함
  - container을 지워도 image는 삭제 안됨!

- rmi <name> - image를 삭제
  - 해당 image로부터 실행중인 container가 1개 이상 존재한다면 삭제 불가

- **docker start nyang**
- **docker ps -a**

Attach

- attach <name> - container에 접속

- **docker attach nyang**
- **apt-get install git**
- **git --version**
- **Ctrl+P, Ctrl+Q**
  - Container을 종료하지 않고 shell을 빠져나오게 해 줌

Copy

- cp <name>:<container-path> <host-path>
  - Container 내부의 파일을 host으로 복사
  - 반대 방향은 불가능 (앞의 mount 기능을 사용하자)

- **docker cp nyang:/home/docker/nyang.nyang ~/**
- **ls -al ~/**

Top / Pause / Unpuase Process

- top <name> <options>
  - 현재 container에 실행중인 프로세스 목록을 보여줌
  - Options은 ps의 option과 동일

- **docker top nyang**

- pause / unpause <name>
  - Container에 있는 모든 프로세스를 멈추거나 다시 시작함
  - 모든 프로세스에게 SIGSTOP을 보내 멈추게 함

End of Basic Operations

- 이제 image를 다운받아 container을 만들고, shell을 띄울 수 있음!
- 이제 attach를 통해 들어가서 이것저것 설치할 수도 있음

- 근데 서버 띄울 때마다 Nginx도 Django도 다 깔고, 설정까지 no-ga-da?

- container를 image로 만들어서, 이 image를 사용하자!! => commit

Diff, Commit & History

- diff <name> - image와 container 사이의 차이점 보기
  - 기준이 되는 image는 이 container를 생성한 이미지

- commit <option> <name> <img-name>
  - container를 commit하여 image만들기
  - -a: user info / -m: commit message
  - 이 외에도 git의 commit과 유사한 옵션들이 있음

Diff & Commit

- **docker diff nyang**
  - Git을 설치했으므로 관련 파일이 추가 / 변경된 것을 볼 수 있음

- **docker commit -m “add git” nyang gguggu:0.1**
  - Nyang container를 gguggu라는 이름의 image로 저장
- **docker history gguggu:0.1**

- **docker run -i -t gguggu:0.1 /bin/bash**
  - gguggu라는 image로부터 container을 하나 생성
- **git –version**

Introducing Dockerfile

- Commit를 이용해서 image를 만들면 편리함
- 하지만 아직 충분히 편리하지 않음

- Image를 생성하는 과정을 자동화시키자!

- Dockerfile은 docker image를 만드는데 필요한 명령을 담은 txt파일

Dockerfile

FROM base-image-name	명령을 실행할 기초 image를 설정함
MAINTAINER name	이 dockerfile을 만든 사람의 정보를 기록함
RUN op arg1 arg2 ...	Shell에서 op arg1 arg2 ... 을 실행함
ADD host-dir dir	Host에 있는 파일을 image에 추가함
EXPOSE port-no	Container가 실행중일 때 특정 포트를 listen하도록 함
CMD op arg1 arg2 ...	Container을 실행하는 기본 방법을 설정함
VOLUME [dir]	Data Volume을 새로 생성함

- 더 많은 명령어는 https://docs.docker.com/reference/builder/

Dockerfile Example

FROM base-image-name	FROM ubuntu:latest
MAINTAINER name	MAINTAINER Sam Jo <samjo@sparcs.org>
RUN op arg1 arg2 ...	RUN apt-get update
ADD host-dir dir	ADD ~/docker /home/docker/file
EXPOSE port-no	EXPOSE 80
CMD op arg1 arg2 ...	CMD ["/usr/sbin/apache2", "FOREGROUND"]

Dockerfile Tutorial

- 목표: Dockerfile을 사용하여 django를 serve해주는 container를 만들어보자!

- **git clone https://github.com/a1sams1a/dockerfile-django.git;**
- **cd dockerfile-django;**
- **docker build –t webapp .**
- **docker run -d -p 80:80 webapp**

Q&A

END OF THIS SEMINAR

References

- https://www.docker.com/
- http://pyrasis.com/Docker/Docker-HOWTO
- http://blog.nacyot.com/articles/2014-01-27-easy-deploy-with-docker/