Virtualization & Docker

+ Portainer, Whale

appleseed

Virtualization

- 가상의 무언가를 만드는 모든 행동
  - 가상 머신, 가상 키보드, 가상 디스크, ...
- 물리적 리소스에 직접 접근하여 조작하는 것을 방지
  - ex) 메모리 가상화: 애플리케이션이 물리적으로 메모리 주소에 접근하는 것은 위험하므로 OS는 페이지 단위로 메모리를 할당하여 모든 애플리케이션에게 메모리에 대한 일관적인 뷰를 제공
- 어떤 소프트웨어가 구동하는 환경을 어디에서나 일관적으로 유지
  - ex) JVM: 모든 Java 바이트코드는 각자 다른 OS에서 각자 다른 JVM(Java Virtual Machine)을 통해 구동되어 일관적인 동작을 수행함

cf. Emulation

- cf. Emulation: 다른 아키텍쳐의 장치를 흉내내는 행동
  - ex) AMD64 기기에서 닌텐도 DS의 명령어 셋을 구현하여 닌텐도 DS 게임을 구동
  - ex) AMD64 기기에서 ARM64 명령어 셋을 구현하여 안드로이드 애플리케이션을 구동
- Emulator는 애플리케이션을 제외한 장치의 모든 부분을 소프트웨어로 구현해야 함
  - 입출력 장치, 디스크, 메모리, CPU, GPU, ...

cf. Emulation

주요 모바일 기기의 ARM 아키텍처			
명령어 세트	아키텍처 명칭	성능	주요 기기
ARMv6	ARM11	1.25 DMIPS/MHz	옵티머스 원, 옴니아 II, **아이폰 오리지널, 아이폰 3G** 등
ARMv7-A	Cortex-A5	1.57 DMIPS/MHz	갤럭시 M 스타일 등
	Cortex-A7	1.9 DMIPS/MHz	갤럭시 S4의 LITTLE 코어, All-winner A31/A20 등
	Cortex-A8	2.0 DMIPS/MHz	갤럭시 S, **아이폰 3GS, 아이폰 4** 등
	Scorpion	1.9~2.1 DMIPS/MHz	옵티머스 Z, 베가 X, 베가 레이서, 옵티머스 LTE 등
	Cortex-A9	2.5 DMIPS/MHz	갤럭시 S II, 아이폰 4, **아이폰 4S**, 옵티머스 2X 등
	Cortex-A12	3.0 DMIPS/MHz	RK3288
	Cortex-A17	3.1~3.3 DMIPS/MHz	MT6595 의 big core
	Krait	3.1/3.39 DMIPS/MHz	옵티머스 G, 베가 R3 등
	Swift	3.5 DMIPS/MHz	**아이폰 5**
	Cortex-A15	3.5~4.01 DMIPS/MHz	갤럭시 S4의 big 코어, 넥서스 10 등

Why Virtualization?

- 프로그래머가 작성한 코드가 어떤 환경에서도 동일하게 작동하는 것을 보장하는 것이 중요하기 때문
  - 메모리가 가상화 되어있지 않다면?
    - 모든 프로세스는 고정된 물리적 메모리 주소를 참조할 수밖에 없음
    - 사용중인 메모리를 overwrite하거나 권한 밖의 메모리를 열람할 수 있음
  - Java 바이트코드가 가상화 되어있지 않다면?
    - Windows/macOS/Linux에서 Java의 동작 방식이 모두 달랐을 것
    - macOS에서 컴파일한 C binary의 Linux 실행 결과를 보장할 수 없음

Why Virtualization?

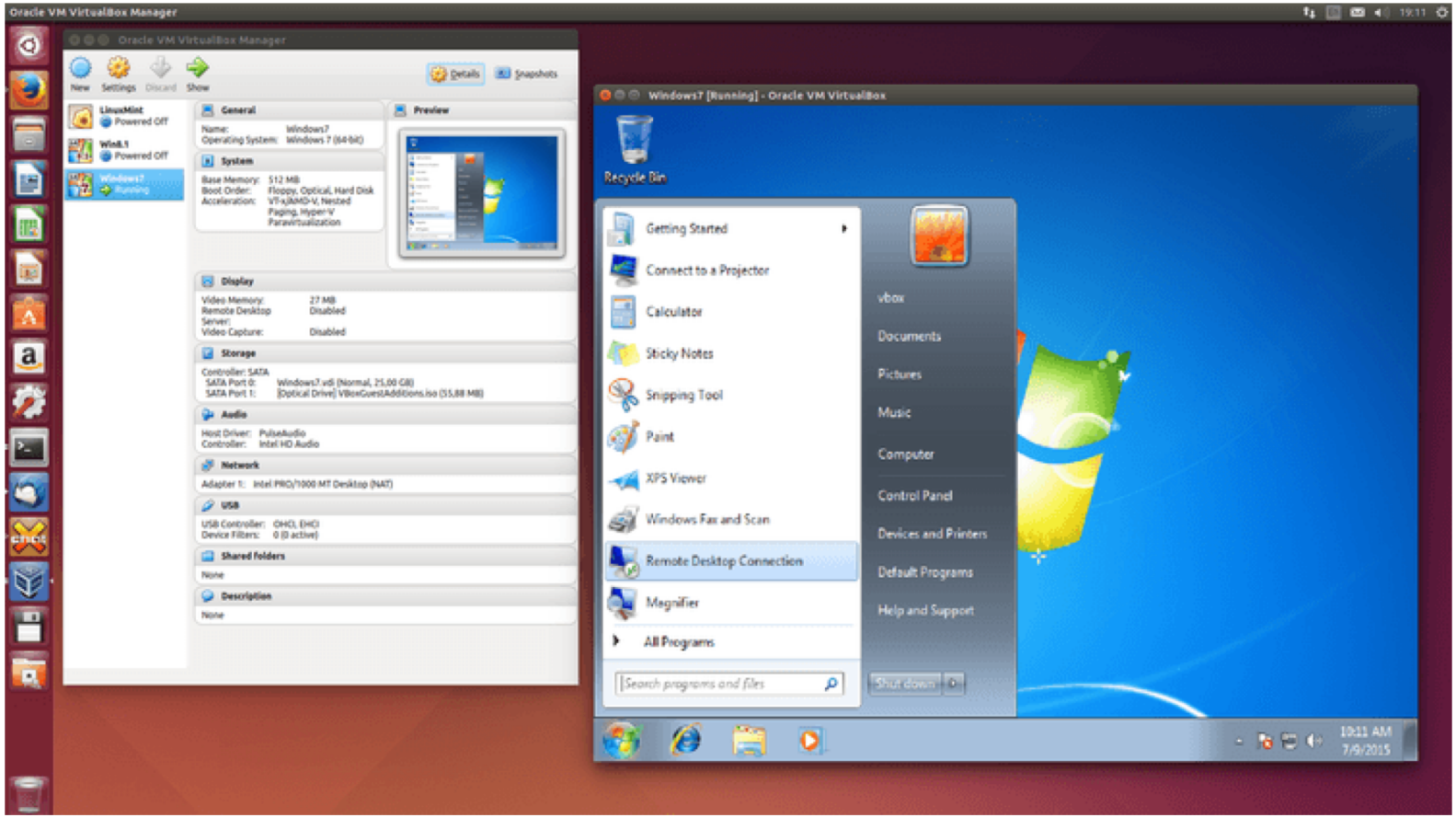
- macOS

```
➜ trash ./test
hello world!
```

- Ubuntu

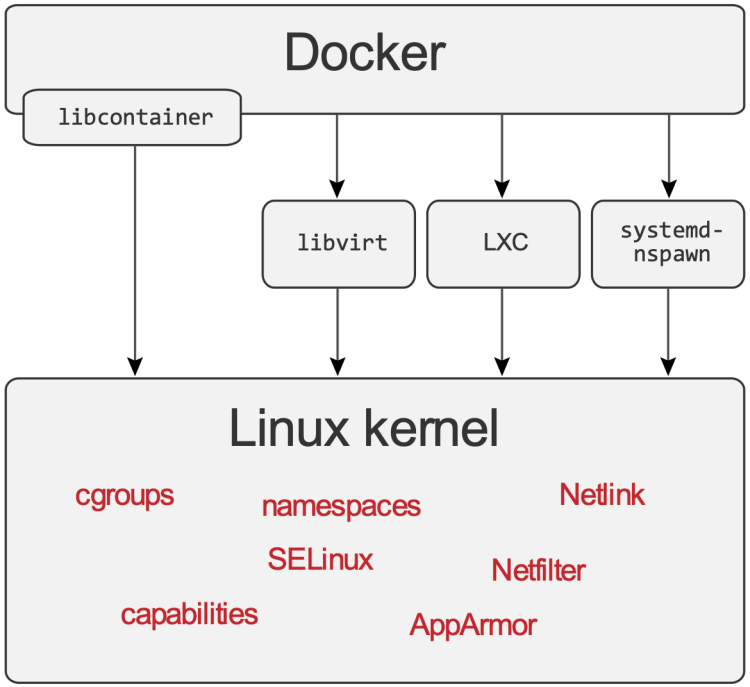
```
appleseed@ssal:~/test$ ./test
-bash: ./test: cannot execute binary file: Exec 형식 오류
```

Why Virtualization?



Why Virtualization?

- 운영 체제 레벨 가상화를 이용한 컨테이너 단위 서비스 배포를 지원하는 가상화 소프트웨어



Full Virtualization

- 완전 가상화 Full virtualization
  - 가상 소프트웨어가 돌아가는 환경을 사용자 영역에서 완전히 분리
  - ex) macOS에서 VirtualBox로 Windows를 구동
    - VB 안에서 Windows의 system call 호출(A)

    → VB는 A를 macOS의 system call로 번역(B)

    → macOS는 B를 실행

    → B의 실행 결과를 VB를 통해 출력
  - System call을 번역하는 과정에서 상당한 overhead가 발생
  - 가상 환경이 접근할 수 있는 하드웨어 또한 모두 가상화되어야 하기 때문에 리소스 낭비가 심각함

Full Virtualization

- 완전 가상화 Full virtualization
  - 가상 소프트웨어가 돌아가는 환경을 사용자 영역에서 완전히 분리
  - ex) macOS에서 Java 바이트코드를 구동
    - Java 바이트코드는 JVM에 의해 macOS system call(B)로 번역됨

    → macOS는 B를 실행

    → B의 실행 결과를 VB를 통해 출력

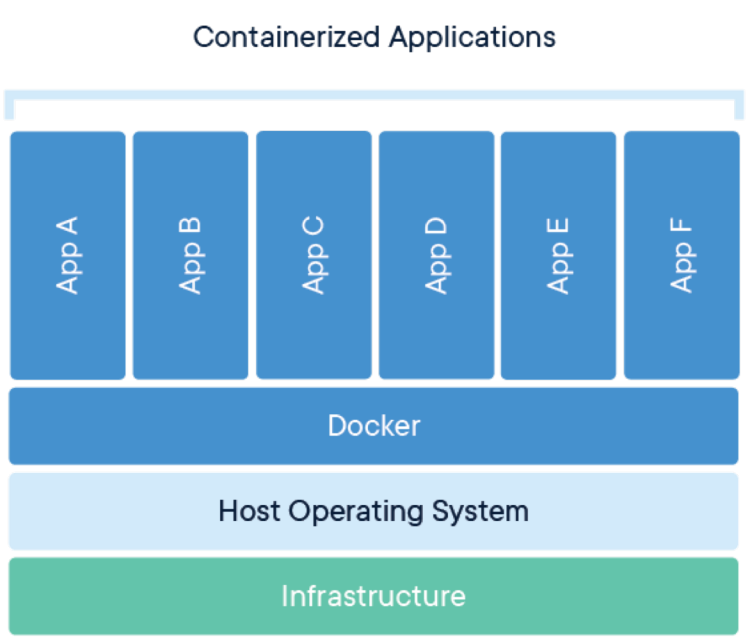
  - System call을 번역하는 과정에서 상당한 overhead가 발생
  - 가상 환경이 접근할 수 있는 하드웨어 또한 모두 가상화되어야 하기 때문에 리소스 낭비가 심각함

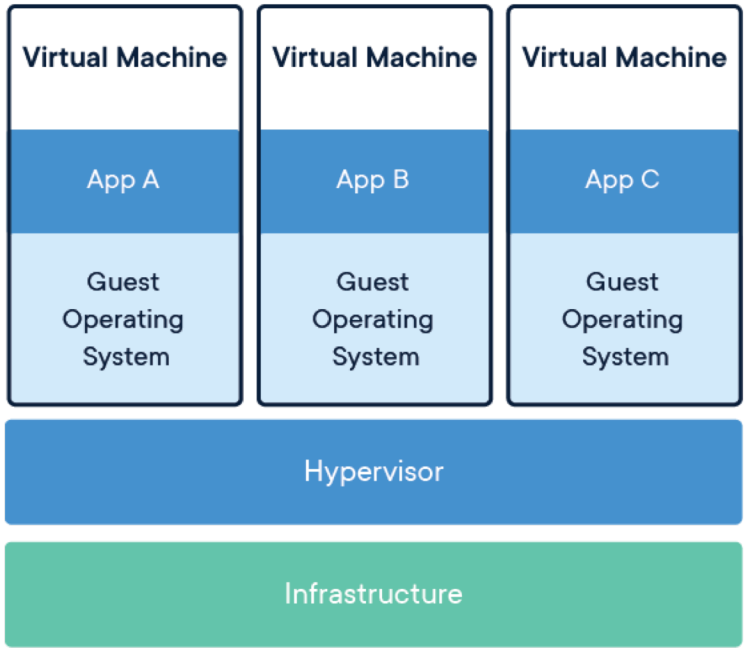
OS-level Virtualization

- 운영체제 레벨 가상화 OS-level Virtualization
  - 가상 환경은 또 다른 사용자 영역처럼 작동
  - ex) CentOS에서 docker container로 Ubuntu 구동
    - CentOS는 Ubuntu system call(A)을 실행

    → A의 실행 결과를 docker container를 통해 출력
  - System call 번역 과정이 없으므로 overhead가 거의 없음
  - OS의 자원/권한 가상화 기능을 그대로 활용하기 때문에 마치 하나의 프로세스처럼 관리 가능

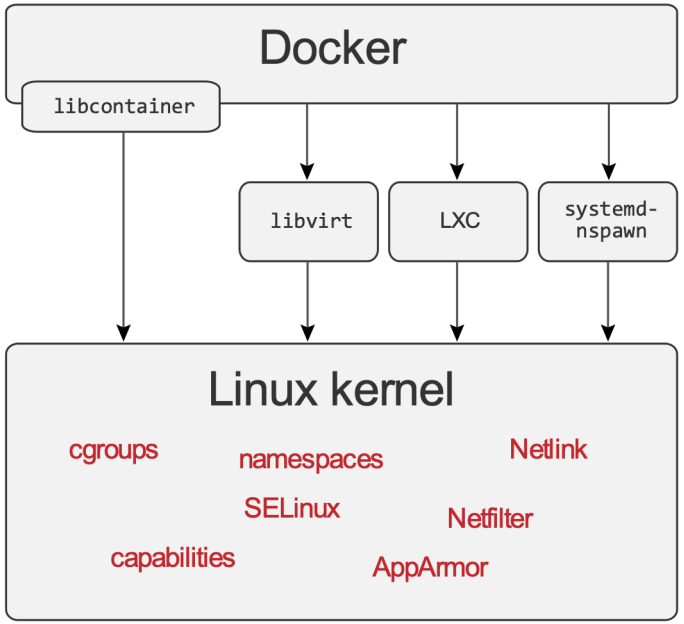
OS-level Virtualization





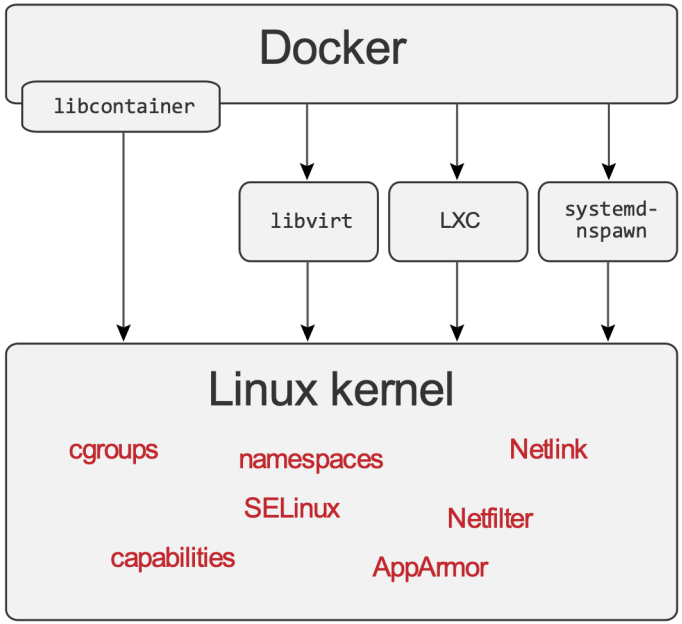
Docker Explained: Fundamentals

- cgroups: 프로세스 묶음이 접근할 수 있는 시스템 자원을 제한하고 관리하는 Linux kernel 기능
  - CPU, 메모리, 디스크, IO 장치, 네트워크, ...
- namespaces: 다른 namespace의 resource를 열람할 수 없도록 하는 Linux kernel 기능
  - PID, UID, 환경변수, 네트워크, IPC, 파일시스템, ...



Docker Explained: Fundamentals

- libcontainer: Linux의 virtualization feature들을 이용하기 쉽도록 묶은 라이브러리
- 추상화된 virtualization interface들인 libvirt, LXC와 systemd-nspawn들을 libcontainer와 함께 사용



Docker Explained: Container & Image

- 컨테이너 Container: 격리된 한 환경의 단위
  - 독립적인 Linux 기기와 매우 유사하게 작동
  - 일부 System call은 사용이 불가능할 수 있음
- 이미지 Image: 시스템의 스냅샷, 디렉토리 구조와 환경변수 등에 대한 시스템의 거의 모든 정보를 담고 있음
  - 특정 상태의 컨테이너를 이미지로 commit한 후 그 이미지를 다른 docker daemon에서 실행시켜도 정확히 같은 환경을 얻을 수 있음
  - [repository]:[tag]의 형태의 이름을 가짐
  - ":[tag]"을 제공하지 않으면 자동으로 latest 사용
  - ubuntu:latest, centos:latest, ubuntu:16.04, ubuntu:xenial, ...

Why Docker?

- 반가상화된 Linux 기기를 생성할 수 있음
  - 같은 배포판, 같은 버전의 Linux에서는 애플리케이션의 실행 환경이 '거의' 동일함
- 다른 가상화에 비해 월등한 performance
  - System call 번역 과정이 없고 기기의 자원에 직접 접근하므로 overhead가 매우 적음
- dockerfile 또는 docker image를 이용해 특정 컨테이너의 환경을 정확히 복제할 수 있음
  - ex) ssal docker로 구동하는 서비스를 AWS로 이전
    - dockerfile이 있다면 AWS에서 build 후 run
    - 또는 구동중인 container를 그대로 image로 만들어서 AWS에서 run

FAQ

- 반가상화?
  - 커널 영역의 리소스는 컨테이너와 호스트 운영체제가 전부 공유
  - Ubuntu 18.04 기기에서 Ubuntu 12.04 docker 띄우고 uname -a 입력해도 커널 버전은 리눅스 최신 커널로 나옴
- macOS나 Windows에서 linux syscall을 가져다 쓰는 docker를 실행 가능한 이유
  - macOS나 Windows의 경우 결국 가상화된 리눅스 커널을 먼저 만들고 그 위에 docker를 띄울 수밖에 없어 전가상화의 단점 보유
  - macOS는 xhyve, Windows는 Hyper-V
  - 최신 Windows는 리눅스 서브시스템을 탑재해 전가상화하지 않아도 docker 실행 가능 (WSL2)

FAQ

- 독립적인 Linux 기기와 가장 큰 차이점
  - PID 0: swapper, sched - 프로세스들을 스케줄링
  - PID 1: /sbin/init - Linux 시스템을 시작함
  - docker는 호스트 Linux의 프로세스 스케줄러를 같이 쓰기 때문에 PID 0 프로세스는 호스트 Linux의 그것과 동일(?)
  - docker의 PID 1 프로세스는 run 할때 넣어준 CMD의 그것
    - /bin/bash를 넣었다면 /bin/bash가 PID 1을 가지고 있음
  - 커널 동작에 필요한 프로세스들에 접근할 수 없으므로 root 권한과 별개로 명령을 거부당할 수 있음
    - --privileged 옵션으로 추가 권한을 부여할 수 있음

Q&A

Let’s practice Docker!

- ssh wheelseminar@15.164.55.89
  - 비밀번호는... 아시죠?
- $ docker --version

```
wheelseminar@ip-172-31-33-32:~$ docker --version
Docker version 18.09.7, build 2d0083d
```

- Ubuntu에서는 apt install docker.io로 설치 가능
  - add-apt-repository 이후 docker-ce로 설치하는 것이 best practice
  - 접속한 instance에는 이미 설치되어 있음

Docker Basics

- docker command는 종류와 옵션이 다양하기 때문에, 도움의 손길을 받도록 하자!
  - $ docker [command] --help | less

```
wheelseminar@ip-172-31-33-32:~$ docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/home/wheelseminar/.docker")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal")
                           (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
                           "/home/wheelseminar/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default
                           "/home/wheelseminar/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/home/wheelseminar/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  engine      Manage the docker engine
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
```

Docker Basics

- docker ps / docker container list: 현재 실행중인 컨테이너 출력
  - --all: 실행중이 아닌 컨테이너까지 모두 출력
- docker images / docker image list: 보유한 이미지 출력
- docker pull [image]: remote 저장소에서 [image]를 가져옴
- docker search [text]: remote 저장소에서 [text] 이미지 검색
  - 저장소인 docker hub는 뒤에 나옴

Docker Basics

- docker run [image] [CMD]: 컨테이너를 시작
  - [CMD]: 컨테이너가 실행할 명령어 - 명령어 실행이 끝나면 stop됨
  - -i (interactive): Keep STDIN open even if not attached
  - -t (tty): Allocate a pseudo-TTY
  - -d (detach): 컨테이너를 실행하는 즉시 detach
  - --name [name]: 컨테이너 이름을 지정, 없으면 랜덤으로 생성
  - --privileged: 컨테이너에 커널을 조작할 추가 권한 부여
  - -p [port] / -P: 포트를 외부로 배정 / 외부로 나가는 포트 모두 개방
- docker exec [CMD]: 컨테이너 안에서 [CMD] 실행
  - -i, -t 등의 옵션 붙일 수 있음

Docker Practice 1

- Ubuntu 위에서 CentOS 컨테이너를 만들어보자!
- docker run -it --name [id]_test centos_update /bin/bash
- docker exec -it centos bash
- uname -a → Ubuntu 버전이 출력됨(?)
  - uname 명령어는 시스템의 커널 정보를 출력하기 때문에, 호스트 운영체제의 커널 정보를 출력하게 됨
- apt update → command not found
- yum update → 패키지 목록을 업데이트
  - CentOS의 패키지 매니저는 yum이기 때문에 Ubuntu의 apt처럼 사용

Docker attach / detach

- attach / detach: 현재 stdin, stdout, stderr을 컨테이너의 것과 연결 / 연결 해제
- docker attach [name]: [name] 컨테이너와 attach
- Ctrl + P + Q: 컨테이너에서 detach
- docker command에서 컨테이너를 가리킬 때에는 container id의 첫 몇 글자를 사용해도 되고, 컨테이너의 이름을 사용해도 됨

```
CONTAINER ID        IMAGE               NAMES
a484ca805628        centos              zen_shaw
```

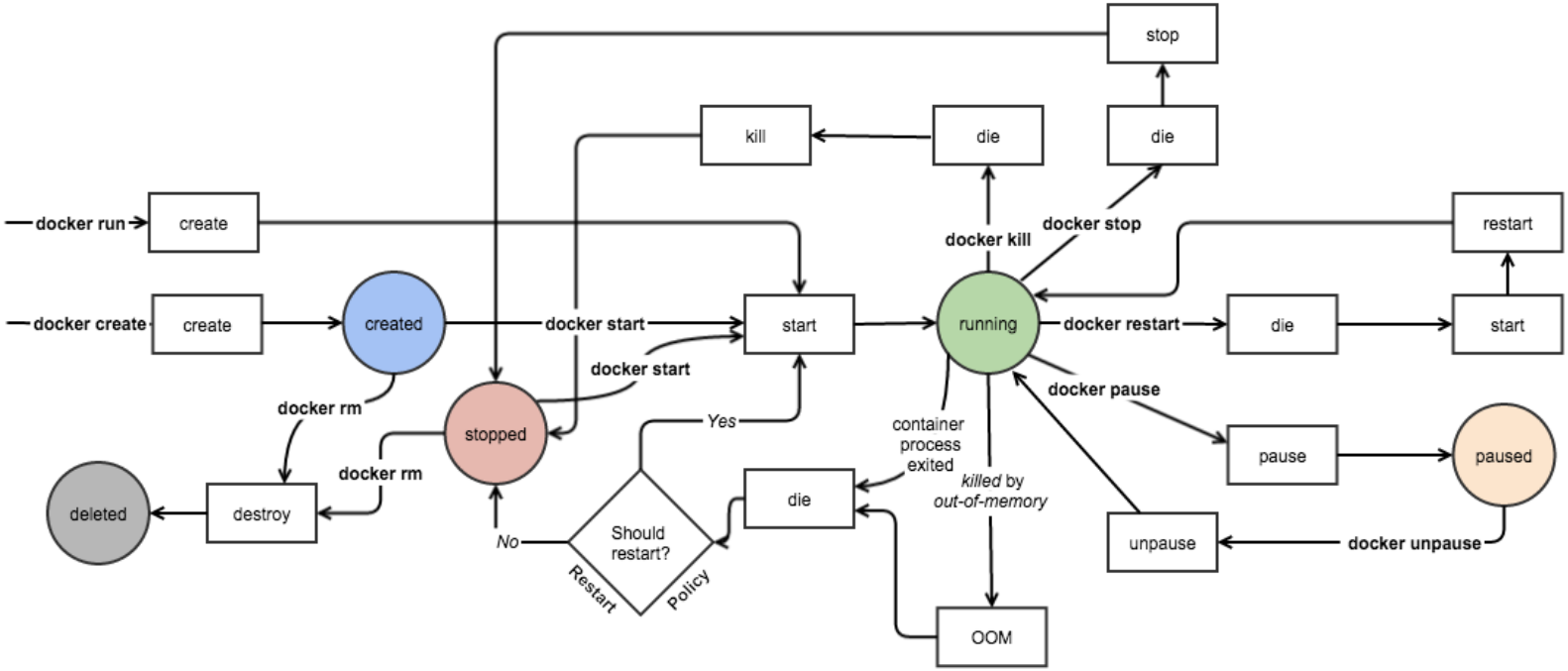
Docker Container Status

- Created: 생성됨, Running: 실행중, Stopped: 정지됨, Paused: 일시정지됨
- 기본적으로 컨테이너는 명령어 하나를 실행한 후 즉시 stopped 상태로 돌입
- 일반적으로 SPARCS에서는 컨테이너를 실행한 후 sshd daemon이 실행중인 상태에서 CMD로 /bin/bash를 실행 - 입력을 대기하기 때문에 Running 상태를 유지
- interactive & tty 옵션을 이용하여 stdin이 계속 입력을 기다리도록 하고 터미널을 통해 컨테이너의 bash와 같은 환경으로 attach될 수 있음

Docker Container Status

- Status management commands
- docker [status] [name]
  - start: stopped 컨테이너를 run
  - stop: running 컨테이너를 stop
  - pause: running 컨테이너를 pause
  - unpause: paused 컨테이너를 run
  - kill: 컨테이너 상태와 관계없이 모든 프로세스를 kill, 컨테이너 stop
  - restart: 컨테이너를 재시작

Docker Container Status



Docker Practice 2

- 오늘의 목표: ssh daemon을 실행하고 컨테이너에 ssh로 접속하기
- 아까 만든 컨테이너를 삭제하자 (CentOS가 실습에 너무 부적합...)
- docker stop [name] → docker rm [name]
- docker run -it --name [name] ubuntu:bionic /bin/bash
- apt update && apt install ssh
- service ssh status
- service ssh start
- 컨테이너의 22번 포트가 공개되어 있지 않으므로 현재는 접속 불가능

Docker Image

- docker commit [name]: 컨테이너를 이미지로 만듦
  - 만들어진 이미지는 이름(레포, 태그)이 없는 상태
  - docker images 실행 시 tag가 비어있는 이미지가 생성된 것을 볼 수 있음
  - 여러 개 존재하는 경우 sha256값 첫 몇 자리를 통해 구분
- docker tag [hash] [repository]:[tag]: 이미지에 레포와 태그를 부여
  - ":[tag]"가 없으면 자동으로 latest 사용
  - docker images 실행 시 레포와 태그가 부여된 것을 볼 수 있음
- 생성된 이미지로 docker run 실행 시 완전히 같은 환경이 재현됨!

Docker Practice 3

- 방금 만들었던 컨테이너를 이미지로 만들고, 포트 개방해서 다시 돌려보자
- 먼저 ssh로 접속할 유저 생성
- adduser wheelseminar
  - 비밀번호 쉽게 갑시다. '1234qwer'
- Ctrl + P + Q로 detach
  - exit 할 경우, 컨테이너는 /bin/bash를 수행 완료한 것이므로 그대로 컨테이너가 stop됨

Docker Practice 3

- docker commit [name]
  - 출력되는 해쉬의 첫 8자리 정도를 아래에서 사용
- docker images → 레포와 태그가 없는 이미지들 생성됨
- docker tag [hash] [id]:1.0 → 이미지에 레포와 태그가 생김
- docker run -it -p 22 [id]:1.0 [id]_ssh /bin/bash
- 컨테이너 안에서, service ssh start 후 Ctrl + P + Q로 detach
- docker ps로 22번 포트에 배정받은 포트 번호 확인
- ssh wheelseminar@localhost -p [port]

Docker Image

- docker save [image]:[tag] -o [output file]
  - 이미지를 tar 파일로 저장
- docker load -i [input file]
  - tar 파일을 이미지 리스트에 로드
- 현재 내 맥북에 떠있는 컨테이너를 그대로 AWS에서 구동하는 방법
  - 내 맥북에서 docker save [my image] -o image.tar
  - scp image.tar [remote] → ssh [remote]
  - docker load -i image.tar → docker run [my image]
  - 다른 머신에서 완전히 동일한 반가상 환경 재현!
  - 네트워크 등 설정만 바꿔준 후 완전히 동일하게 운영 가능

Dockerfile

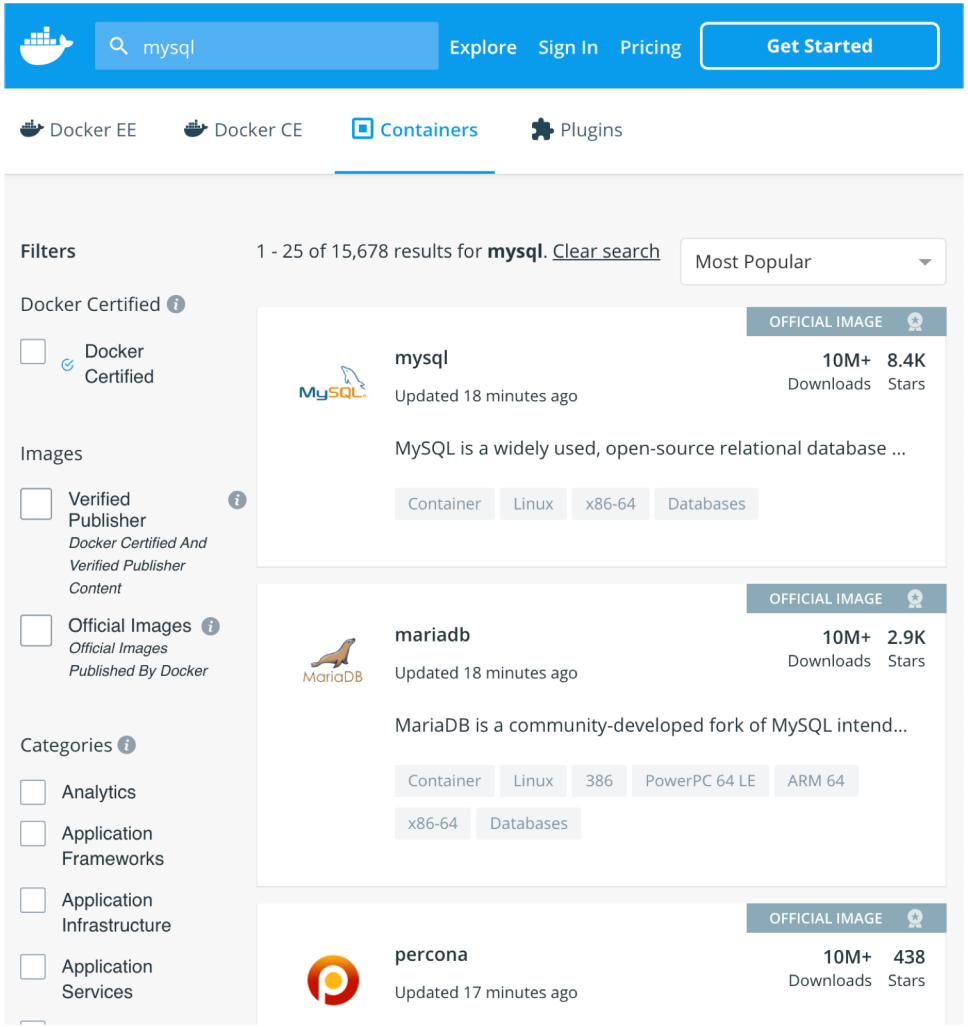
- docker 이미지는 컨테이너의 상태를 거의 그대로 보존하지만, 어떻게 그 컨테이너가 설정되었는지에 대한 정보는 없음
- dockerfile의 형태로 이미지를 만드는 과정을 기록하고 dockerfile로부터 이미지를 생성할 수 있음
- docker build [path]: dockerfile를 이미지로 빌드함
  - [path]/Dockerfile이 default dockerfile 경로
  - -f [file]: 파일명이 Dockerfile이 아닐 경우 [file]로 명시
  - -t [repo]:[version]: 이미지를 태그함
  - -q: 빌드 과정의 stdout을 무시

Docker Practice 4

- mkdir [id] && cd [id]
- cp ../appleseed/Dockerfile ./ && cd ..
- docker build [id] -t [id]:2.0
- docker images
- docker run -d -it -P --name [id]_ssh_2 [id]:2.0 /bin/bash

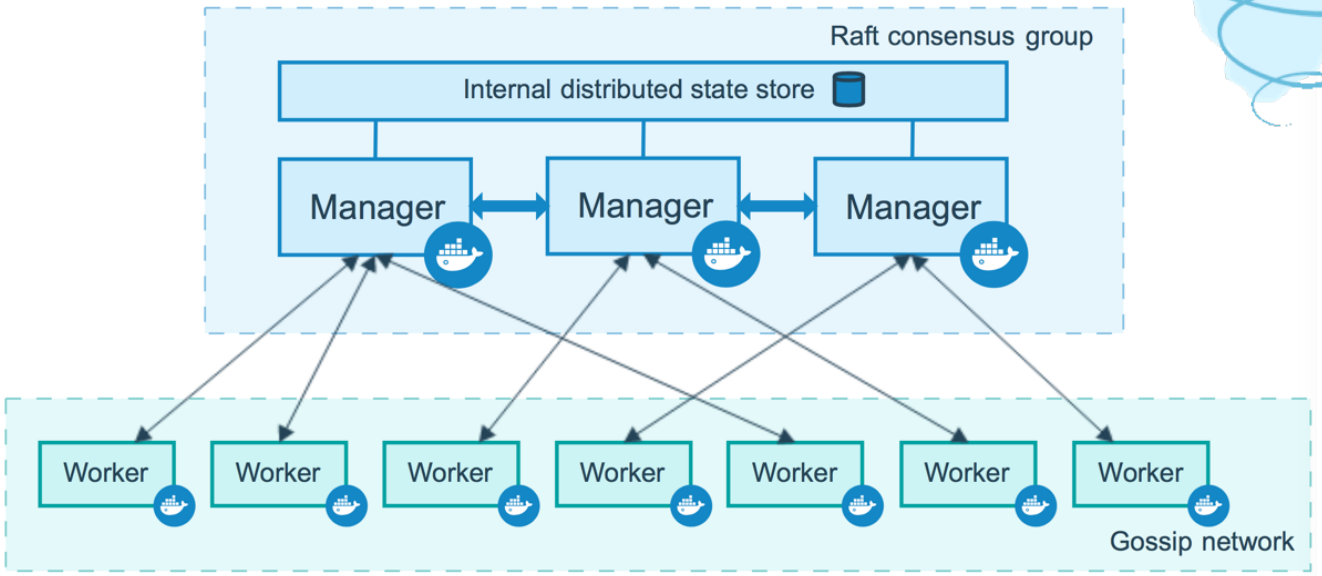
Further - Docker Hub

- https://hub.docker.com
- docker 이미지 저장소
- 기업들은 자신이 만든 서비스를 docker hub에서 컨테이너 단위로 배포
- docker search/pull 등으로 CLI에서도 사용 가능



Further - Docker Swarm

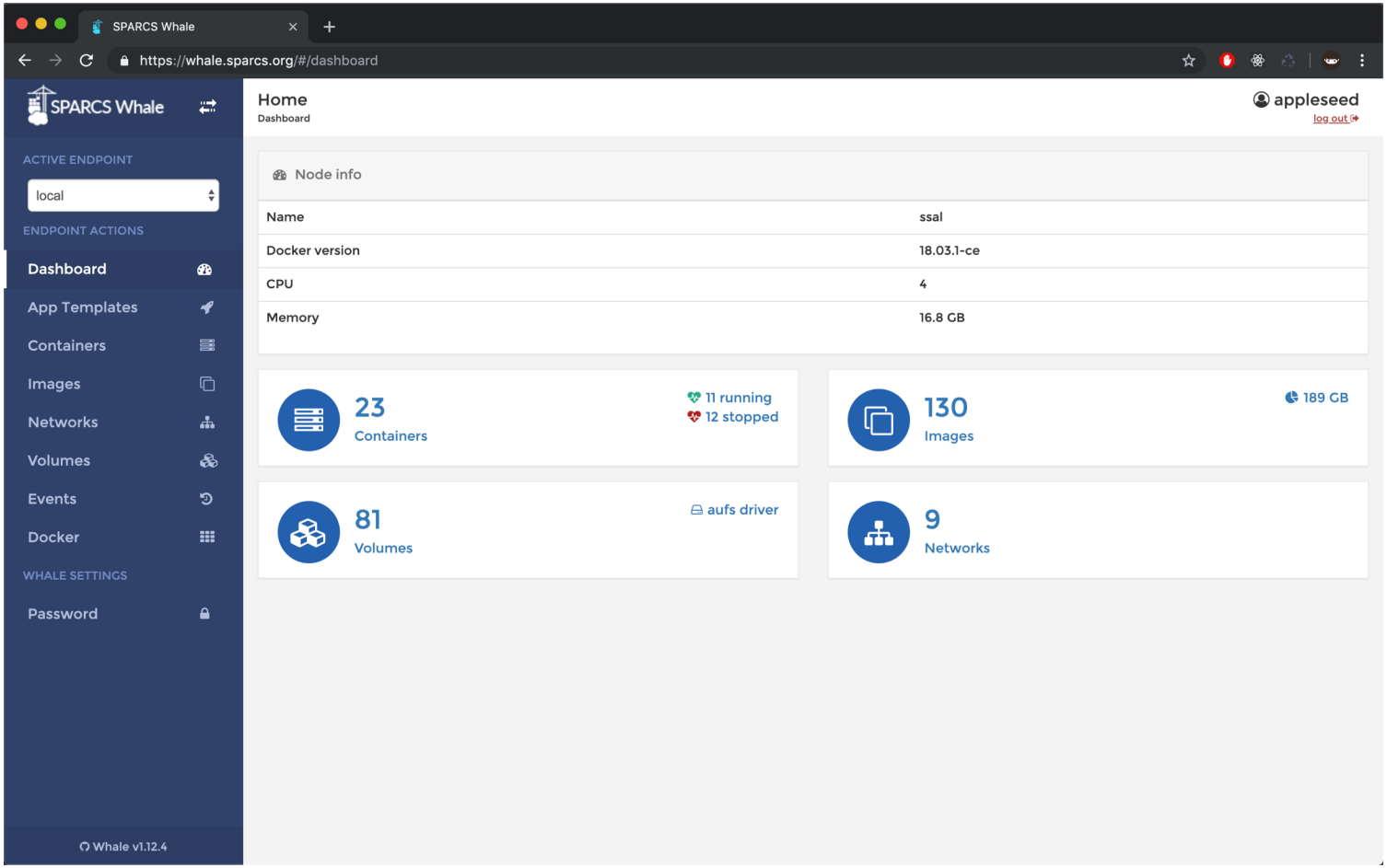
- 컨테이너 오케스트레이션 도구의 하나
- 여러 docker 호스트들을 하나인 것처럼 묶어줌
- 각 docker daemon은 swarm의 manager/node



Q&A

- docker나 docker swarm을 웹 프론트엔드로 관리할 수 있도록 개발된 관리 UI
- portainer 그 자체도 swarm 내의 하나의 컨테이너로서 동작
- docker daemon 자체는 유저 별 컨테이너의 접근 권한을 제한하는 기능이 없으나 portainer를 사용하여 접근 제한 가능
- SPARCS whale은 초창기 portainer를 포크하여 작성되었고 현재 ssal의 standalone docker daemon을 관리하는 중

portainer.io




```
appleseed@ssal:~$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                    NAMES
80f14de106ac        ubuntu_mongo:latest    "/bin/bash"              25 hours ago        Up 25 hours         0.0.0.0:32802->22/tcp, 0.0.0.0:32801->80/tcp
                    acid-rain
112152f883c2        ubuntu_sparcs:latest   "/bin/bash"              2 weeks ago         Up 2 weeks          0.0.0.0:32778->22/tcp
                    joyb_test
306a82e1c182        ubuntu_ssh:latest      "/bin/bash"              3 weeks ago         Up 8 days           0.0.0.0:18012->22/tcp
                    kja_pintos
211158846f12        ubuntu_mongo:latest    "/bin/bash"              7 weeks ago         Up 2 weeks          0.0.0.0:10000->22/tcp, 0.0.0.0:10001->80/tcp, 0.0.0.0:10002->443/tcp
                    zabo-cookie
f1c9f195a250        ubuntu_sparcs:latest   "/bin/bash"              2 months ago        Up 2 weeks          0.0.0.0:32771->22/tcp, 0.0.0.0:32770->80/tcp, 0.0.0.0:32769->27017/tcp
                    kris_reset
3bed93b950d8        sparcs_biseo:1.0       "/bin/bash"              2 months ago        Up 2 weeks          0.0.0.0:32777->22/tcp, 0.0.0.0:32776->80/tcp, 0.0.0.0:32775->443/tcp, 0.0.0.0:32774->27017/tcp
                    biseo_test
da237f7f75dc        ubuntu_mongo:latest    "/bin/bash"              2 months ago        Up 10 days          22/tcp, 0.0.0.0:32797->80/tcp, 0.0.0.0:32796->443/tcp, 0.0.0.0:32795->27017/tcp
                    mango_biseo
46a0fba8a998        ubuntu_kris:latest     "/bin/bash"              2 months ago        Up 2 weeks          0.0.0.0:32773->22/tcp, 0.0.0.0:32772->80/tcp
                    kris_practice
f074d9c5eea7        ubuntu_mongo:1.0       "/bin/bash"              2 months ago        Up 2 weeks          0.0.0.0:32780->22/tcp, 0.0.0.0:32779->80/tcp
                    wika-backend
6cce348fc371        ubuntu_sparcs:latest   "/bin/bash"              2 months ago        Up 3 weeks          0.0.0.0:3000-3001->3000-3001/tcp, 0.0.0.0:5000->5000/tcp, 0.0.0.0:2222->22/tcp, 0.0.0.0:8080->80
/tcp, 0.0.0.0:4433->443/tcp         nothing
8773f306c05b        mailserver:190206      "/bin/bash"              5 months ago        Up 2 weeks          0.0.0.0:25->25/tcp, 0.0.0.0:110->110/tcp, 0.0.0.0:143->143/tcp, 0.0.0.0:993->993/tcp, 0.0.0.0:99
5->995/tcp, 0.0.0.0:20619->80/tcp   mailserver2
b41d8d1c0296        ubuntu_sparcs:latest   "/bin/bash"              7 months ago        Up 2 weeks          0.0.0.0:32783->22/tcp, 0.0.0.0:50056->2404/tcp
                    zackie_mathtech2
083b555fc61f        084ec18124c8           "docker-entrypoint.s…"   9 months ago        Up 3 weeks          5432/tcp
                    davidgram_db
5b40b87252f9        f7b3f317ec73           "/bin/bash"              2 years ago         Up 3 weeks          0.0.0.0:31415->22/tcp
                    leeopop_dev
1af63d50d8d7        whale                  "/whale --no-analyti…"   2 years ago         Up 3 weeks          127.0.0.1:9000->9000/tcp
                    whale
```

Q&A