CHOI MOONHWAN

moonhwan.choi@seven.co.kr

**Docker guide manual**

설치와 운영

Docker의 설치부터 기본적인 운영까지   
다루는 문서입니다.

목차

[**1. 도커 환경 구축을 위한 VM 설정 (VM ware)** 5](#_Toc108883738)

[**1-1. 네트워크 설정** 5](#_Toc108883739)

[**2. 도커 환경 구축을 위한 OS 설치** 6](#_Toc108883740)

[**2-1. docker1 (ubuntu20.04 LTS)** 6](#_Toc108883741)

[**2-2. docker2 (centos7)** 7](#_Toc108883742)

[**3. 도커 엔진 설치** 7](#_Toc108883743)

[**3-1. 도커엔진 설치 (우분투)** 7](#_Toc108883744)

[**3-2. 도커엔진 설치 (센토스)** 7](#_Toc108883745)

[**4. 도커의 기본 개념과 용어 정리** 8](#_Toc108883746)

[**5. 도커 컨테이너 살펴보기** 8](#_Toc108883747)

[**5-1. 도커 컨테이너란?** 9](#_Toc108883748)

[**6. 도커 컨테이너 만들어 보기** 13](#_Toc108883749)

[**6-1. 도커 파일의 기본문법** 13](#_Toc108883750)

[**6-2. 컨테이너 만들기 실습** 14](#_Toc108883751)

[**7. 컨테이너 보관 창고** 20](#_Toc108883752)

[**7-1. 정의 및 개념** 20](#_Toc108883753)

[**7-2. Registry운영 실습** 20](#_Toc108883754)

[**7-3. Private Registry 운영하기** 23](#_Toc108883755)

[**8. Docker Container 사용하기** 24](#_Toc108883756)

[**8-1. 컨테이너 이미지 관리 명령어** 24](#_Toc108883757)

[**8-2. 컨테이너 관리 명령어** 26](#_Toc108883758)

[**9. 컨테이너 관리하기** 28](#_Toc108883759)

[**9-1. 기본옵션 설명 (Memory, CPU, Block I/O 리소스 제한&컨테이너 모니터링)** 28](#_Toc108883760)

[**9-2. 컨테이너 리소스 관리 실습** 30](#_Toc108883761)

[**10. 컨테이너가 사용하는 스토리지** 36](#_Toc108883762)

[**10-1. 개요** 36](#_Toc108883763)

[**10-2. 컨테이너 볼륨 실습** 37](#_Toc108883764)

[**11. 컨테이너 간의 통신 방법** 42](#_Toc108883765)

[**11-1. 기본개념** 42](#_Toc108883766)

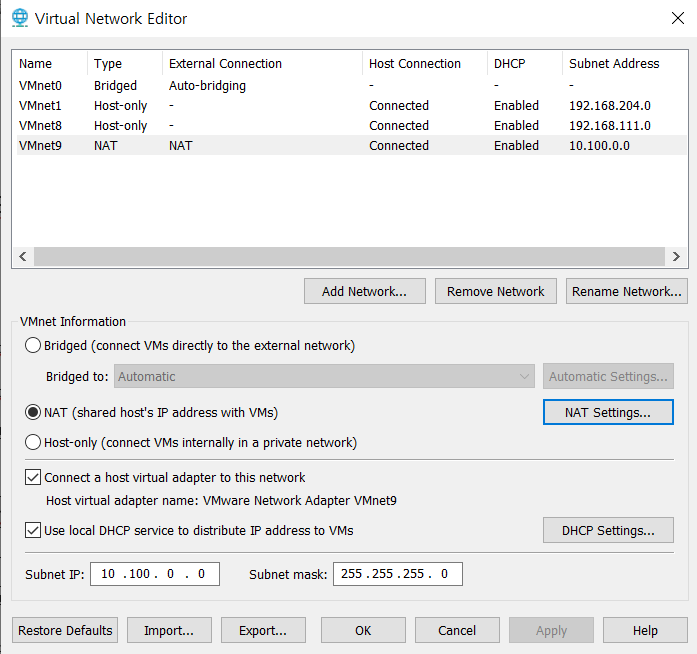
[**11-2. 네트워크 설정 조회, 사용자 네트워크 인터페이스 구성, 포트 포워딩 예제** 43](#_Toc108883767)

**제/개정 이력**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 1.0 | 제 정 | 2022/07/05 |
| 개정번호 | 제/개정 페이지 및 내용 | 제.개정 일자 |

# **1. 도커 환경 구축을 위한 VM 설정 (VM ware)**

## **1-1. 네트워크 설정**



|  |  |
| --- | --- |
| **옵션** | **설정 값** |
| Network type | NAT |
| Subnet IP | 10.100.0.0/24 |
| Subnet mask | 255.255.255.0 |
| Gateway | 10.100.0.1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **hostport** | **type** | **VirtualMachineIP** | **VirtualMachinePort** | **description** |
| 105 | tcp | 10.100.0.105 | 22 | docker1-ubuntu |
| 106 | tcp | 10.100.0.106 | 22 | docker2-centos |

-포트 포워딩 정보 (NAT Settings)

\*우분투 VM – 10.100.0.105 / CentOS VM – 10.100.0.106

# **2. 도커 환경 구축을 위한 OS 설치**

## **2-1. docker1 (ubuntu20.04 LTS)**

-username :

임의대로 (guru)

-hostname : /etc/hostname 에서 호스트 이름 변경

docker-ubuntu.example.com

-hosts파일 : /etc/hosts 에서 아래의 내용을 추가한다.

10.100.0.105 docker-ubuntu.example.com docker-ubuntu

10.100.0.106 docker-centos.example.com docker-centos

-네트워크 설정 :

ip address : 10.100.0.105

gateway : 10.100.0.1

subnetmask : 255.255.255.0

dns : 8.8.8.8

-루트 접속을 위한 비밀번호 설정 :

sudo passwd root 입력 후 비밀번호 설정

-cli 로그인 설정 :

sudo systemctl set-default multi-user.target

-gui 로그인 설정 :

sudo systemctl isolate graphical.target

-필요한 필수 패키지 설치 :

apt-get update

apt-get install -y curl vim tree openssh-server

## **2-2. docker2 (centos7)**

-hosts , hostname 은 우분투 설정과 동일

-네트워크 설정 :

ip address : 10.100.0.106

gateway : 10.100.0.1

subnetmask : 255.255.255.0

dns : 8.8.8.8

-KVM 가상화 머신 비활성화 :

ip addr 실행시켜보면 virbr0 항목이 나오는데 이것이 CentOS 설치 시 자동으로 설치되는 KVM가상 머신 하이퍼 바이저이다. 필요 없으므로 사용 중지시킨다.

systemctl stop libvirtd

systemctl disable libvirtd

-텍스트모드 로그인 :

systemctl set-default multi-user.target

-우분에서 설치한 패키지와 동일 하게 :

yum update

yum install -y curl vim tree openssh-server

yum install -y 프로그램명을 이용하여 설치한다.

# **3. 도커 엔진 설치**

## **3-1. 도커엔진 설치 (우분투)**

<https://docs.docker.com/engine/install/ubuntu/>

## **3-2. 도커엔진 설치 (센토스)**

<https://docs.docker.com/engine/install/centos/>

유저에 도커 권한 추가하기 – usermod -a -G docker username

# **4. 도커의 기본 개념과 용어 정리**

도커란 ?

도커는 컨테이너 기술을 기반으로 한 일종의 가상화 플랫폼이다. 가상화란 물리적 자원인 하드웨어를 효율적으로 활용하기 위해서 하드웨어 공간 위에 가상의 머신을 만드는 기술이고, 컨테이너란 컨테이너가 실행되고 있는 호스트 os의 기능을 그대로 사용하면서 프로세스를 격리해 독립된 환경을 만드는 기술을 뜻한다.

다시 말하자면, 도커는 **독립된 환경을 만들어서 하드웨어를 효율적으로 활용하는 기술**이라고 할 수 있다.

Container Image – 하드디스크 또는 도커 허브에 저장되어 있는 컨테이너 이미지 **파일** (ReadOnly)

Container – 컨테이너 이미지가 실행되어 하나의 **프로세스**가 되어있는 상태 (Read Write가능) , 쉽게 생각해 하나의 어플리케이션으로 보면 되는데, 개발한 어플리케이션(실행파일)과 운영환경이 모두 들어있는 독립된 공간이다.

Docker Host – 도커 프로세스가 실행되고 있는 리눅스 커널

Docker Daemon – 리눅스 커널에서 실행되고 있는 도커 (systemctl start docker로 확인)

Docker Client Command – docker daemon에게 명령어를 전달하는 측

Docker Hub – 컨테이너 이미지가 저장되어 있는 가상의 저장소

ㅅ

# **5. 도커 컨테이너 살펴보기**

## **5-1. 도커 컨테이너란?**

앞에서도 설명했듯이 컨테이너는 쉽게 생각하면 하나의 어플리케이션이다. 그 어플리케이션에 필요한 실행파일과 운영환경이 모두 저장 되어있는 독립적인 공간이라고 생각하면 쉽다. 그렇다면 컨테이너는 어떻게 만들어지는 것일까?

컨테이너는 DockerFile을 이용해 컨테이너를 빌드한다. DockerFile이란 컨테이너를 만들 수 있도록 도와주는 명령어들의 집합이다. 참고로 대소문자는 구분하지는 않지만 가독성을 위해서 사용하는 것이 좋다.

docker version – 도커 설치 상태 체크

▶docker version

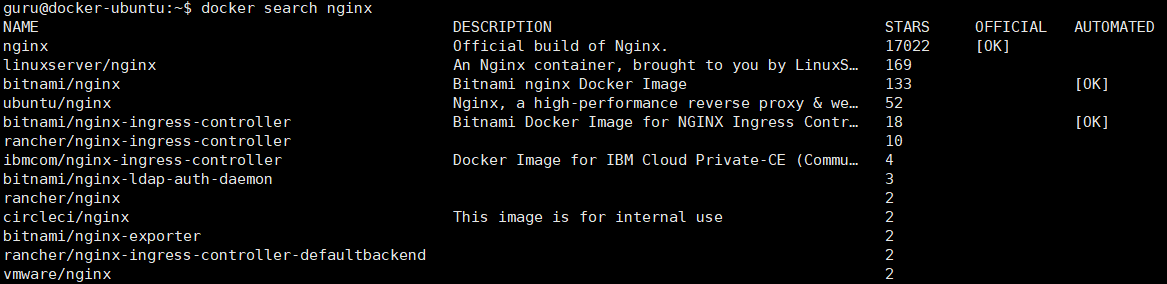
텍스트이(가) 표시된 사진

자동 생성된 설명

Client와 Server가 설치되어 있으면 정상적으로 설치가 된 것이다.

docker search [image name] – 허브에 있는 도커 이미지 검색

▶docker search nginx



현재 도커 허브에 저장되어 있는 nginx의 이미지 리스트이다.

도커 이미지를 다운 받을 때는

docker pull [imageName] – 도커 이미지 다운로드 (단순히 이미지를 다운로드만 받은 상태이다. 컨테이너화 시키는 것은 아니다.)

▶docker pull nginx

텍스트이(가) 표시된 사진

자동 생성된 설명

명령어 실행결과 총 6개의 레이어가 포함된 nginx컨테이너 이미지가 다운로드 되었다.

/var/lib/docker/overlay2 – 도커 이미지의 레이어가 파일 형태로 저장되는 곳

cd /var/lib/docker/overlay2 로 이동하여 ls -l 명령어를 확인 해보면 파일 목록 확인이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

docker image ls (또는 docker images) – 도커 이미지 리스트 보기

▶ docker images

텍스트이(가) 표시된 사진

자동 생성된 설명

위에서 설치한 nginx이미지가 확인된다.

도커 이미지 실행시키기

docker run --[옵션 이미지이름]

▶ docker run -d --name nginxweb -p 80:80 nginx



텍스트이(가) 표시된 사진

자동 생성된 설명

정상적으로 nginx 컨테이너가 실행되고 있음을 확인할 수 있다.

실행시킨 이미지 확인

docker ps -> 실행중인 프로세스 리스트

▶ docker ps



컨테이너 중지시키기

docker stop [컨테이너 이름]

▶ docker stop nginxweb

텍스트이(가) 표시된 사진

자동 생성된 설명

stop 명령어는 단순히 컨테이너를 중지시킨 것이다.

컨테이너를 삭제하기 위해서는 rm 명령어를 사용해야 한다.

docker ps -a 명령어를 실행시켜보면 중지된 컨테이너 목록을 포함하여 목록을 보여준다.

▶ docker ps -a



그리고 도커 컨테이너를 삭제하기 위해서는 다음과 같은 명령어를 사용한다.

docker rm [컨테이너 이름]

▶ docker rm nginxweb

텍스트이(가) 표시된 사진

자동 생성된 설명

완전히 삭제된 것을 확인할 수 있다.

컨테이너 이미지 삭제

docker rmi [이미지 이름]

▶ docker rmi nginx

텍스트이(가) 표시된 사진

자동 생성된 설명

그 후 다시 overlay2 디렉토리를 확인해보면 다음과 같이 레이어들이 삭제된 것을 확인할 수 있다.

텍스트, 표지판, 오렌지이(가) 표시된 사진

자동 생성된 설명

# **6. 도커 컨테이너 만들어 보기**

## **6-1. 도커 파일의 기본문법**

# : comment

FROM : 컨테이너의 Base Image (운영 환경)

MAINTAINER : 이미지를 생성한 사람의 이름 및 정보

LABEL : 컨테이너 이미지에 컨테이너의 정보를 저장

RUN : 컨테이너 빌드를 위해 base image에서 실행할 commands

COPY : 컨테이너 빌드 시 호스트의 파일을 컨테이너로 복사

ADD : 컨테이너 빌드 시 호스트의 파일(tar, url 포함 )을 컨테이너로 복사

WORKDIR : 컨테이너 빌드 시 명령이 실행될 작업 디렉토리 설정

ENV : 환경변수 지정

USER : 명령 및 컨테이너 실행 시 적용할 유저 설정

VOLUME : 파일 또는 디렉토리를 컨테이너의 디렉토리로 마운트

EXPOSE : 컨테이너 동작 시 외부에서 사용할 포트 지정

CMD : 컨테이너 동작 시 자동으로 실행할 서비스나 스크립트 지정

ENTRYPOINT : CMD와 함께 사용하면서 command 지정 시 사용

## **6-2. 컨테이너 만들기 실습**

node.js기반의 어플리케이션 컨테이너를 만들어 본다.

우선 hellojs 디렉토리를 만들고 이동한다. 그리고 간단한 node.js파일을 만든다.

▶ mkdir hellojs

▶ cd hellojs

▶ cat > hello.js

const http = require('http');

const os = require('os');

console.log('Test server starting...');

var handler = function(request, response){

console.log('Received request from '+ request.connection.remoteAddress):

response.writeHead(200);

response.end('Container Hostname: ' + os.hostname() + '\n');

};

var www = http.createServer(handler);

www.listen(8080);

소스코드를 만들었으면 이제 컨테이너를 만들어야 하는데, 컨테이너를 만들기 위해서는 도커파일을 생성해야 한다.

▶ vim dockerfile

FROM node:12 -> 베이스 이미지 설정

COPY hello.js / -> hello.js를 컨테이너의 / 디렉토리로 복사

CMD ["node","/hello.js"] -> node라는 명령을 가지고 / 에 있는 hello.js를 실행시키도록 커맨드 실행

# 작은따옴표 절대안됨!

컨테이너를 실행시키면 마지막 CMD구문이 실행되도록 설정한 것임

이제 이 도커파일을 가지고 컨테이너를 빌드한다.

▶ docker build -t hellojs:latest .

-t : 컨테이너 이름과 태그이름 설정

. : 현재 디렉토리에 있는 파일을 가지고 빌드를 하라 만약 소스파일이 /tmp에 있다면 /tmp로 해주면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

docker images로 만들어진 컨테이너 이미지를 확인한다.

▶ docker images

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 우분투 기반의 웹서버 컨테이너 이미지를 만들어 본다.

마찬가지로 디렉토리를 만들고 이동한다. 그리고 도커파일을 생성한다.

▶mkdir webserver

▶cd webserver

▶vim Dockerfile

FROM ubuntu:18.04

LABEL maintainer="MoonHwan Choi <kkang17859@naver.com>"

# install apache

RUN apt-get update \

&& apt-get install -y apache2

RUN echo "TEST WEB" > /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/apache2ctl","-DFOREGROUND"]

여기서 &&를 쓰는 이유는 RUN을 두 번 쓰게 되면 이미지가 하나 더 생기므로

RUN 한 줄에 명령어를 여러 개 씀으로써 효율성을 높인다.

확인을 했으면 이제 도커파일을 빌드하여 컨테이너 이미지를 만든다.

▶docker build -t webserver:v1 .

정상적으로 완료되었다면 다음과 같은 메시지가 나온다.



docker images로 정상적으로 생성되었는지 확인한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 컨테이너 이미지 파일을 실행시켜 본다.

▶docker run -d -p 80:80 --name web webserver:v1

-d : 데몬 모드 (백그라운드 실행)

-p : 포트 설정

--name : 이름 설정

web webserver:v1 -> web이라는 이름으로 webserver:v1을 실행 최신 버전이면 태그를 생략가능 하지만 그게 아니라면 생략 불가



docker ps 로 프로세스를 확인한다.



curl localhost:80 으로 정상적으로 웹서버가 작동하는지 확인 한다.



정상적으로 TEST WEB을 반환한다.

이제 만든 도커 이미지를 자신의 도커허브에 업로드 해본다.

docker login 명령어를 이용하여 로그인을 진행한다. (계정이 없다면 생성해야 한다.)

▶ docker login

텍스트, 스크린샷, 실내, 닫기이(가) 표시된 사진

자동 생성된 설명

docker images를 입력하여 현재 이미지 목록을 확인한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

앞서 만든 두개의 이미지가 보인다. 하지만 개인의 도커허브에 push를 하기 위해서는 이미지 파일 앞에 본인의 계정이 들어가야 한다. 이미지 태그를 변경해준다.



docker images를 이용하여 이미지 리스트를 확인해보면

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

위와 같이 기존 이미지와 이름을 바꾼 이미지가 같이 있는 것이 확인되는데

이미지 ID를 확인해보면 같은 이미지 라는 것을 확인할 수 있다.

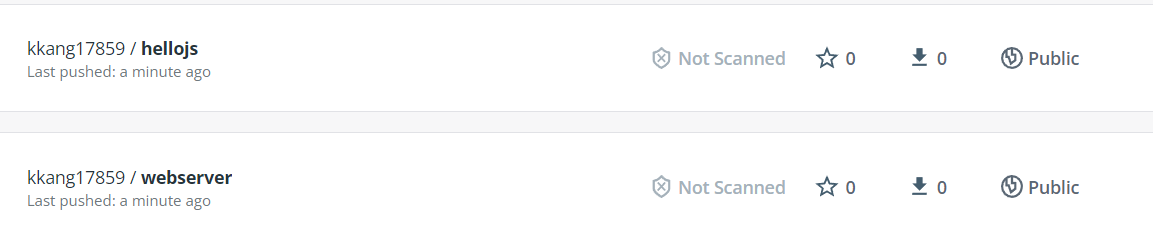
그리고 docker push 명령어를 이용하여 도커허브에 이미지를 업로드 한다.

▶ docker push [이미지 이름]

텍스트이(가) 표시된 사진

자동 생성된 설명

그후 <https://hub.docker.com/> 에 접속하여 본인의 레포지토리를 확인한다.



정상적으로 컨테이너가 잘 배포되었다.

# **7. 컨테이너 보관 창고**

## **7-1. 정의 및 개념**

컨테이너 이미지 보관 창고 – Registry라고 한다.

종류는 크게 2가지가 있는데

1.Docker Hub : 도커에서 직접 운영하는 레지스트리이다. 10만여 개 이상의 이미지들이 있다.

<https://hub.docker.com/>

2.Private Registry : 사내에서 직접 운영하는 레지스트리

docker registry를 이용하면 구축할 수 있다.

## **7-2. Registry운영 실습**

도커 허브에서 이미지 검색시 search 명령어를 사용한다.

docker search [검색 키워드]

▶ docker search httpd

텍스트이(가) 표시된 사진

자동 생성된 설명

도커 허브에서 이미지를 다운로드 받을때는 pull 명령어를 사용한다.

docker pull [이미지 이름:버전] (:버전 생략 시 가장 최신버전이 다운로드 된다.)

▶ docker pull httpd:latest

텍스트이(가) 표시된 사진

자동 생성된 설명

이미지 리스트를 볼 때는 images라는 명령어를 사용한다.

▶ docker images

텍스트, 화면, 스크린샷, 닫기이(가) 표시된 사진

자동 생성된 설명

그리고 이미지를 업로드 하기전에 먼저 도커 허브에 로그인을 해준다.

▶ docker login

입력 후 비밀번호를 입력하면 로그인이 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 이미지를 업로드 할 때는 pull명령어를 사용한다.

▶ docker push [이미지 이름]

여기서 주의할 점은 업로드 하고자 하는 이미지의 이름에 본인의 레포지토리가 포함되어 있어야 한다는 점이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위에서 다운 받은 hpptd에 tag명령어를 이용하여 레포지토리를 정보를 등록해준다.

▶ docker tag httpd kkang17859/httpd

▶docker images

텍스트이(가) 표시된 사진

자동 생성된 설명

kkang17859/httpd 라는 이름의 이미지가 하나 더 생성되었다. 기존 httpd와 이미지ID가 같으므로 같은 이미지이다.

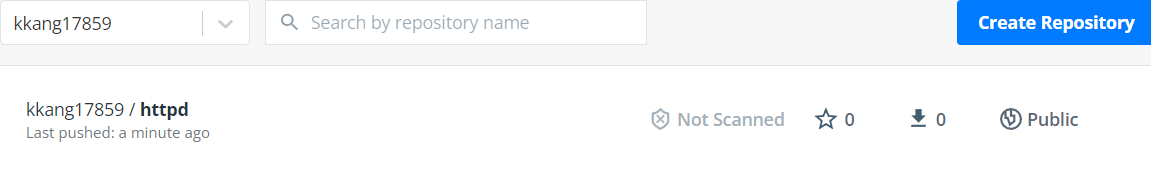
이제 push명령어를 이용해서 이미지를 업로드 한다.

docker push [이미지 이름]

▶ docker push kkang17859/httpd

텍스트이(가) 표시된 사진

자동 생성된 설명



조금 전 push했던 kkang17859/httpd 이미지가 정상적으로 업로드 된 것을 확인할 수 있다.

## **7-3. Private Registry 운영하기**

개인 저장소는 <https://hub.docker.com/_/registry> 해당 링크를 참조하여 구성한다.

다음 명령어를 실행하여 컨테이너를 실행시킨다.

▶ docker run -d -p 5000:5000 –restart always –name registry registry:2

텍스트이(가) 표시된 사진

자동 생성된 설명

그후 프로세스 실행을 체크한다.



그리고 업로드할 이미지를 확인하고 tag를 수정해준다.

▶ docker tag httpd:latest localhost:5000/httpd

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 push명령어를 이용하여 이미지를 업로드 한다.

▶ docker push localhost:5000/httpd

텍스트이(가) 표시된 사진

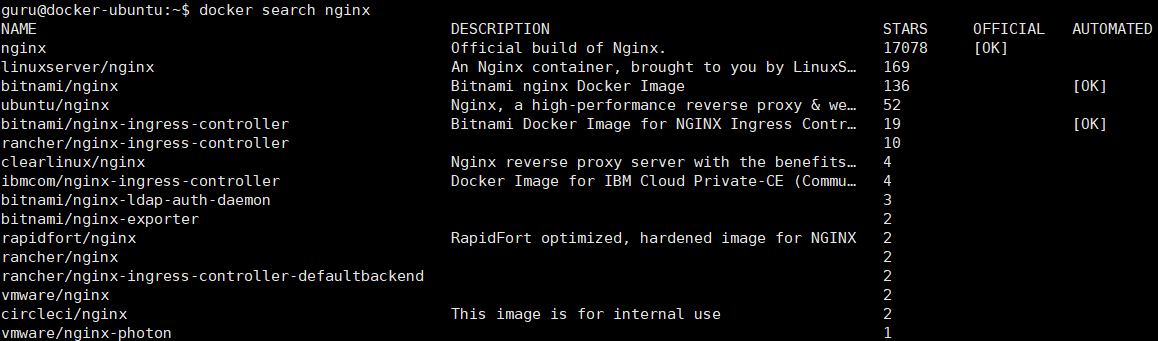
자동 생성된 설명

# **8. Docker Container 사용하기**

## **8-1. 컨테이너 이미지 관리 명령어**

-컨테이너 이미지 검색

▶docker search [이미지 이름]



-컨테이너 이미지 다운로드

▶docker pull [이미지 이름:버전]

버전은 생략 시 최신버전으로 다운로드가 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-다운로드 받은 이미지 확인

▶docker images 또는 docker image ls

▶docker images --no-trunc

: 다운받은 이미지의 ID를 풀 네임으로 표시한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

## **8-2. 컨테이너 관리 명령어**

-현재 동작중인 프로세스 확인

▶docker ps (-a 옵션은 이전 목록까지 모두 조회한다.)

-컨테이너 생성하기

▶docker create --name [컨테이너명 지정] [사용할 이미지 이름:버전]

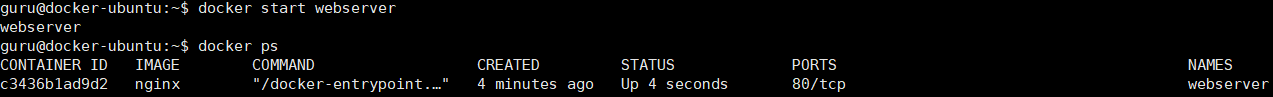




아직 생성만 되었고, running중은 아니다

-생성한 컨테이너 실행하기

▶docker start [컨테이너 이름]



-컨테이너 세부 정보 확인

▶docker inspect [컨테이너 이름]

▶docker inspect --format ‘{{.NetworkSettings.IPAddress}}’ webserver

: --format 옵션으로 원하는 정보만 조회가 가능하다 IP주소 조회의 경우 빈번하게 사용이 되므로 alias에 등록해서 사용하면 좋다.



-컨테이너의 로그 조회

▶docker logs [컨테이너 이름]

▶docker logs -f [컨테이너 이름]

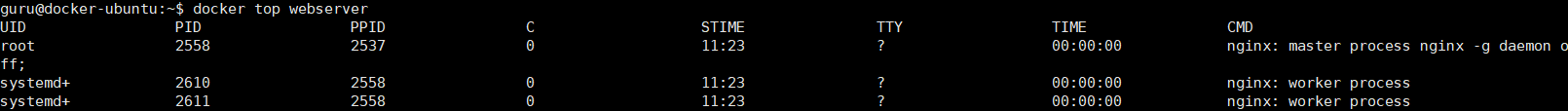
: -f 옵션을 주면 실시간 모니터링이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

-컨테이너에서 실행중인 프로세스 목록 조회

▶docker top [컨테이너 이름]



-컨테이너에 직접 접속하여 쉘 이용하기

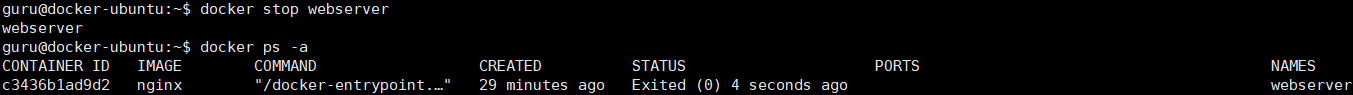
▶docker exec -it [컨테이너 이름] /bin/bash

▶-i : interactive / -t :terminal



-컨테이너 실행 중지시키기

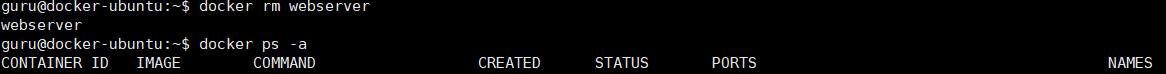
▶docker stop [컨테이너 이름]



-컨테이너 삭제시키기

▶docker rm [컨테이너 이름]

: running중인 컨테이너는 삭제가 불가능하므로 먼저 stop을 시켜 주어야 한다. 또는 -f 옵션을 추가하면 자동으로 stop시키고 삭제까지 시킨다.



# **9. 컨테이너 관리하기**

## **9-1. 기본옵션 설명 (Memory, CPU, Block I/O 리소스 제한&컨테이너 모니터링)**

▶Memory Resource 제한

|  |  |
| --- | --- |
| **옵션** | **의미** |
| --memory, -m | 컨테이너가 사용할 최대 메모리 양을 지정 |
| --memory-swap | 컨테이너가 사용할 스왑 메모리 영역에 대한 설정  스왑 메모리에 일반 메모리가 포함 되어있음  \*실제 스왑 메모리=설정한 스왑 메모리–설정한 메모리 |
| --memory-reservation | --memory값보다 적은 값으로 구성하는 소프트 제한 값 설정(최소 요구 메모리 설정) |
| --oom-kill-disable | OOM killer가 프로세스를  kill하지 못하도록 보호 |

\*제한단위는 b,k,m,g 로 지정한다.

▶CPU 리소스 제한

|  |  |
| --- | --- |
| 옵션 | 의미 |
| --cpus | 컨테이너에 할당할 CPUR core수를 지정  --cpus=”1.5” 컨테이너가 최대 1.5개의 CPU  파워를 사용가능 |
| --cpuset-cpus | 컨테이너가 사용할 수 있는 CPU나 코어를 할당, cpu index는 0부터  --cpust-cpus=0-4 |
| --cpu-share | 컨테이너가 사용하는 CPU비중을 1024값을  기반으로 설정  --cpu-share 2048 기본 값보다 두배 많은  CPU자원을 할당 |

▶Block I/O 리소스 제한

|  |  |
| --- | --- |
| 옵션 | 의미 |
| --blkio-weight  --blkio-weight-device | Block I/O의 Quota를 설정할 수 있으며  100~100까지 선택 (default=500) |
| --device-read-bps  --device-write-bps | 특정 디바이스에 대한 일기와 쓰기 작업의  초당 제한을 kb, mb, gb단위로 설정 |
| --device-read-iops  --device-write-iops | 컨테이너의 read/write속도의 쿼터를 설정  초당 쿼터를 제한해서 I/O를 발생시킴  0이상의 정수로 표기  초당 데이터 전송 량 = IOPS\*블록크기 |

▶컨테이너 모니터링

-docker monitoring commands

docker stats [option] [container….] : 실행중인 컨테이너의 런타임 통계를 확인

docker events(image) -f container=[name] : 도커 호스트의 실시간 이벤트 정보를 수집해서 출력

-cAdvisor (docker 모니터링 툴)

## **9-2. 컨테이너 리소스 관리 실습**

▶stress container 생성

실습을 위해 리눅스의 부하테스트 프로그램 stress를 이용하는 방법을 알아보자

CPU부하 테스트(2개의 코어를 100% 사용하는 부하) : stress --cpu 2

메모리 부하 테스트(프로세스 수 2개와 사용할 메모리 만큼 부하 발생) : stress --vm2 --vm-bytes [사용할 크기]

stress를 설치하고 동작시키는 컨테이너를 빌드 한다.

도커 파일 생성

FROM debian

MAINTAINER moonhwan choi <kkang17859@gmail.com>

RUN apt-get update; apt-get install -y stress

CMD [“/bin/sh”, ”-c”, ”stress -c 2”]

텍스트이(가) 표시된 사진

자동 생성된 설명

기본적으로 2개의 CPU를 사용한다.

▶컨테이너 리소스 제한

실습을 위한 stress 컨테이너 이미지를 만들었으면 컨테이너 실행 시 메모리 설정하여 부하를 테스트한다.

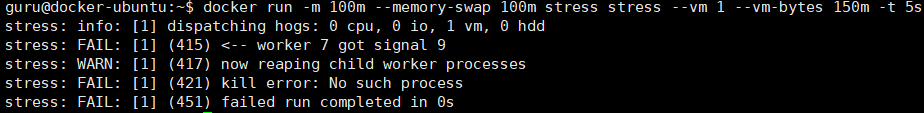
\*메모리 제한 테스트

▶ docker run -m 100m --memory-swap 100m stress stress --vm 1 --vm-bytes 90m -t 5s



100m가 안되는 90m으로 실행되므로 실행 성공

▶ docker run -m 100m --memory-swap 100m stress stress --vm 1 --vm-bytes 150m -t 5s



100m를 넘어서는 150m가 실행되므로 실행 실패

▶ docker run -m 100m stress stress --vm 1 --vm-bytes 150m -t 5s



스왑 메모리를 생략하면 실제 사용가능한 메모리는 200m가 되므로 정상적으로 실행

▶ docker run -d -m 100M --name m4 --oom-kill-disable=true nginx

텍스트이(가) 표시된 사진

자동 생성된 설명

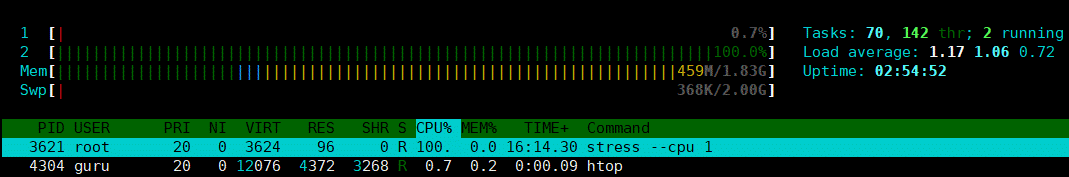
움 킬러 비활성화

\*CPU제한 테스트

▶docker run --cpuset-cpus 1 --name c1 -d stress:latest stress --cpu 1



▶htop



현재 2번째 CPU의 사용량이 100%인 것을 확인할 수 있다.

만약 위 옵션에서 --cpuset-cpus 0 으로 하였다면 위 화면의 1번째 CPU의 사용량이 올라간다.

그리고 --cpuset-cpus 0-1 으로 한다면 0~1번중 랜덤으로 할당이 된다.

▶docker run -c 2048 --name cload1 -d stress:latest

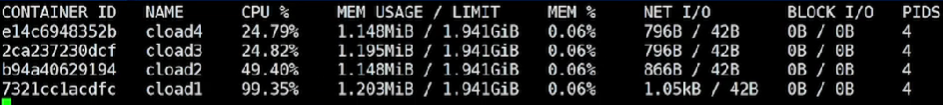
▶docker run --name cload2 -d stress:latest

▶docker run -c 512 --name cload3 -d stress:latest

▶docker run -c 512 --name cload4 -d stress:latest

cload1 컨테이너는 cload2 컨테이너에 비해 2배 많은 CPU할당량을 받게 된다. (default=1024)

그리고 4개의 컨테이너의 리소스 사용 통계를 확인해본다.

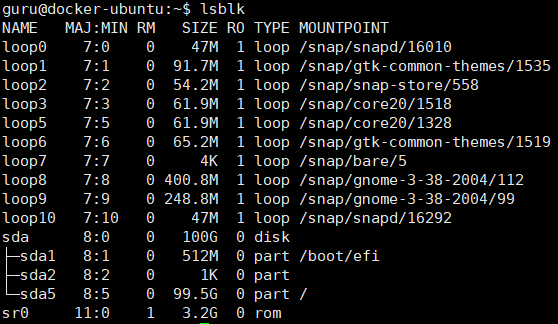


CPU의 %를 보면 설정한 비율과 같이 할당된 것을 볼 수 있다.

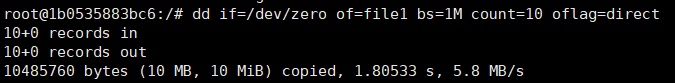
\*Block I/O 제한 테스트

▶docker run -it --rm --device-write-iops /dev/sda:10 ubuntu:latest /bin/bash

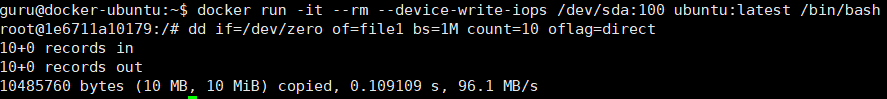
디바이스명은 시스템마다 다를 수 있으므로 lsblk 명령어를 사용하여 확인한다.



▶dd if=/dev/zero of=file1 bs=1M count=10 oflag=direct



약 5.8 MB/s의 속도가 나온다. 그리고 /dev/sda:100 으로 옵션을 주면



약 96.1 MB/s의 속도가 나온다. 이런 방식으로 I/O의 속도를 조정할 수 있다.

▶cAdvisor 설치해서 사용하기

<https://github.com/google/cadvisor> 링크로 접속한다.

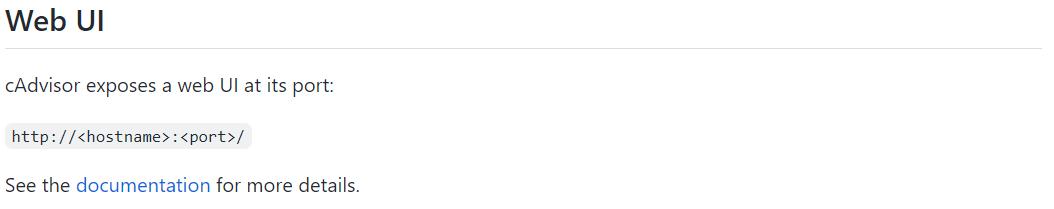
텍스트이(가) 표시된 사진

자동 생성된 설명

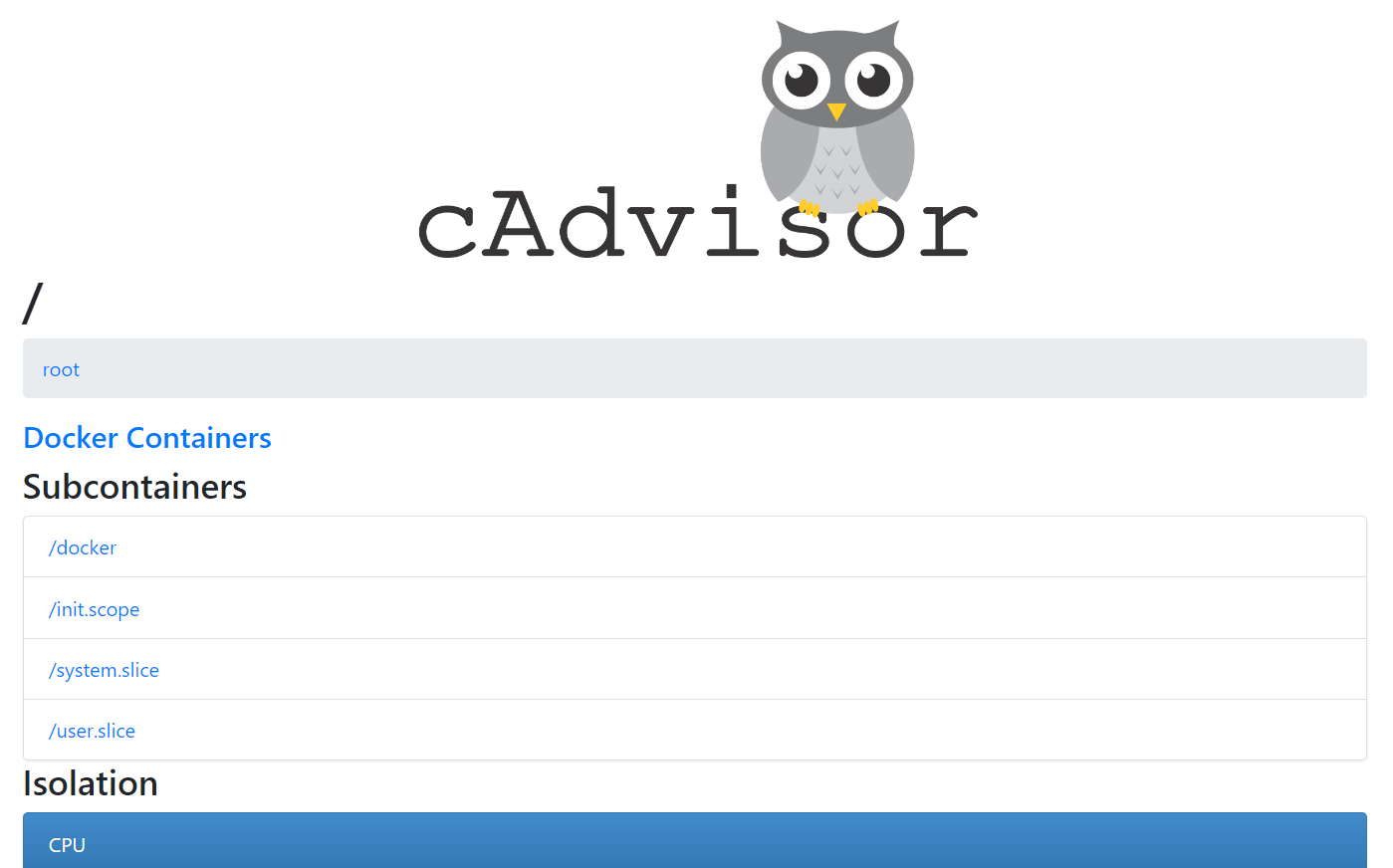
하단에 위와 같은 내용이 있다. 명령어를 복사해서 쉘에서 실행시켜 준다.

텍스트이(가) 표시된 사진

자동 생성된 설명



그리고 <http://docker-ubuntu:8080> 으로 접속한다.



다음과 같은 화면이 나오는데 여기서 여러 정보들을 확인 가능하다.

# **10. 컨테이너가 사용하는 스토리지**

## **10-1. 개요**

-컨테이너 볼륨이란?

도커 컨테이너 이미지는 기본적으로 Read-only 속성을 가지고 있다. 하지만 우리는 컨테이너를 사용하면서 컨테이너에 여러 정보들을 저장하는데 이런 정보들은 Writable layer에 저장이 된다. (이러한 방식을 Union file system 또는 overlay system이라고 한다.)

하지만 어떠한 이유로 컨테이너가 삭제된다면 writable layer의 정보들도 같이 삭제가 되어 복구를 할 수 없는 상황이 생긴다. 이런 상황을 방지하기 위해서 볼륨을 사용하게 된다.

Volume 옵션을 사용하여 실제 호스트 디렉토리에 저장되게 하는데 이것을 볼륨 마운트라고 한다. 이 옵션을 이용하면 컨테이너가 삭제되어도 데이터가 영구적으로 보존이 된다.

그리고 컨테이너끼리 볼륨을 공유할 수도 있다. 각각 다른 컨테이너가 호스트의 같은 디렉토리를 바라본다면 같은 데이터를 공유하게 된다.

-volume 옵션 사용법

-v [host path]:[container mount path]

-v [host path]:[container mount path]:[read write mode]

-v [container mount path]

## **10-2. 컨테이너 볼륨 실습**

**\*호스트 디렉토리의 경로를 지정해주는 경우**

▶docker run -d --name mysql\_db -e MYSQL\_ROOT\_PASSWORD=1234 -v /dbdata:/var/lib/mysql mysql:latest

참고할 것은 여기서 호스트의 /dbdata디렉토리는 현재 생성하지 않았다.



지정한 디렉토리가 없는 경우 자동으로 생성을 해준다.

▶docker exec -it mysql\_db /bin/bash

위의 명령어를 통해 mysql\_db컨테이너에 접속을 한다. 그리고 새로운 디비를 하나 생성한다.

▶mysql -u root -p

텍스트이(가) 표시된 사진

자동 생성된 설명

▶show databases;

텍스트이(가) 표시된 사진

자동 생성된 설명

▶create database test\_db;



텍스트이(가) 표시된 사진

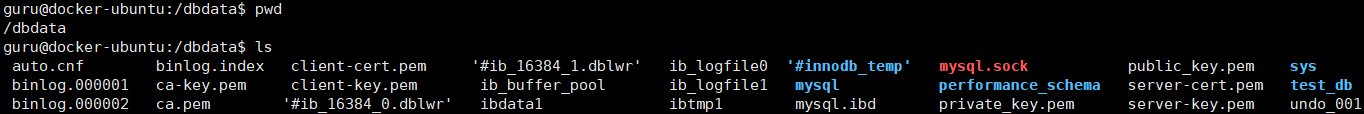
자동 생성된 설명

디비 생성까지 정상적으로 되었다면 컨테이너 접속을 끊고 호스트로 나온다.

그리고 정상적으로 /dbdata에 데이터들이 등록되었는지 확인한다.

▶cd /dbdata

▶ls



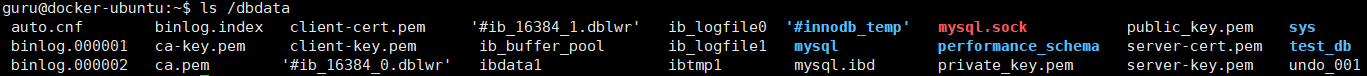
그리고 컨테이너가 삭제되어도 정상적으로 데이터를 보존하고 있는지 확인해보자

▶docker rm -f mysql\_db





▶ls /dbdata



컨테이너가 삭제되어도 정상적으로 데이터를 보존하고 있다.

▶docker volume ls

도커 볼륨 목록 조회

▶docker volume rm [UUID]

도커 볼륨 삭제

**\*호스트 디렉토리의 경로를 지정해주지 않는 경우**

▶docker run -d --name mysql\_db -e MYSQL\_ROOT\_PASSWORD=1234 -v /var/lib/mysql mysql:latest

첫번째 예제와는 달리 호스트 디렉토리의 경로를 지정하지 않았다.

어느 디렉토리에 저장되는지 확인하기 위해서

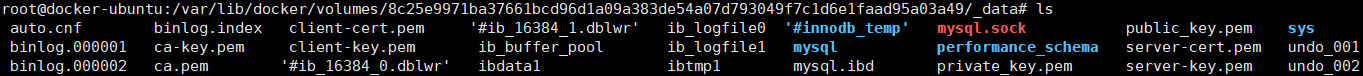
▶docker inspect mysql\_db

텍스트이(가) 표시된 사진

자동 생성된 설명

/var/lib/docker/volumes/8c25e9971ba37661bcd96d1a09a383de54a07d793049f7c1d6e1faad95a03a49/\_data 경로에 마운트가 된 것을 확인할 수 있다.

실제 경로로 이동하여 확인을 해본다.



**\*웹데이터를 컨테이너에 지원하기 (readonly service)**

우선 웹데이터를 저장해 둘 디렉토리를 생성한 후 디렉토리로 이동한다.

▶mkdir /webdata

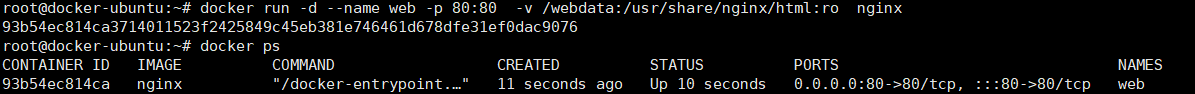
▶cd /webdata

그리고 임시 파일을 하나 만들어 준다.

▶echo "<h1>hello world</h1>" > index.html (root사용자)



▶docker run -d --name web -p 80:80 -v /webdata:/usr/share/nginx/html:ro nginx



▶curl localhost:80



그리고 /webdata에 있는 index.html의 내용을 바꾼 후 다시 테스트해본다.

텍스트이(가) 표시된 사진

자동 생성된 설명

정상적으로 반영이 된다.

# **11. 컨테이너 간의 통신 방법**

## **11-1. 기본개념**

**-docker0**

도커는 컨테이너 네트워크를 위해 docker0 라는 가상 인터페이스를 가진다. docker0는 도커가 자체적으로 지원하는 브리지(Bridge)에 해당한다. 모든 컨테이너는 외부 통신을 docker0를 이용한다. 그리고 컨테이너 실행 시 172.17.X.Y로 IP를 할당하게 된다.

**-container port forwarding**

도커 컨테이너가 사용하는 IP주소는 사설IP주소이기 때문에 기본적으로 컨테이너 외부에서 내부로의 접속이 불가능하다. 이러한 한계를 극복하기 위해 포트 포워딩(포트 맵핑)이라는 기법이 널리 사용되고 있다. 포트가 노출될 때에는 iptables rule이 포트 포워딩을 지원해준다.

-p hostPort:containerPort

-p containerPort

-P

위와 같은 3가지 방식으로 포트 포워딩을 할 수 있다.

**-사용자가 임의로 정의하는 네트워크 인터페이스**

기본적으로 docker0는 컨테이너에 정적IP를 할당해줄 수 없다. 이런 경우 본인이 원하는 네트워크 대역, 정적IP를 설정하고 싶을 때는 user-defined bridge network를 이용하면 된다.

▶docker network create --driver drigde --seubnet [ip address] --gateway [ip address] [interface name]

생성시 위와 같은 형식의 명령어를 사용한다.

▶docker network ls

네트워크 인터페이스 목록 확인

## **11-2. 네트워크 설정 조회, 사용자 네트워크 인터페이스 구성, 포트 포워딩 예제**

**\*네트워크 설정 조회**

▶ip addr

텍스트이(가) 표시된 사진

자동 생성된 설명

docker0 네트워크 인터페이스가 172.17.0.1/16의 IP주소와 네트워크 대역을 가진 것을 볼 수 있다.

▶docker run -it --name con1 busybox

▶ip a

텍스트이(가) 표시된 사진

자동 생성된 설명

처음 실행시킨 컨테이너의 IP주소가 172.17.0.2/16으로 확인된다. 마찬가지로 컨테이너를 하나 더 실행시킨 후 IP주소를 확인하면 172.17.0.3으로 나온다. 순서대로 1씩 증가한다.

▶ping -c 3 8.8.8.8

텍스트이(가) 표시된 사진

자동 생성된 설명

컨테이너 내부에서 외부로 잘 통신이 되는 것을 확인할 수 있다.

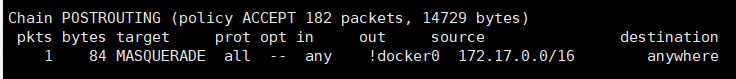
확인을 위해서 con1컨테이너의 설정을 본다.

텍스트이(가) 표시된 사진

자동 생성된 설명

게이트웨이가 docker0의 주소로 되어있다.

▶iptables -t nat -L -v



172.17.0.0 네트워크 대역에서 어디로든 나갈 때 호스트의 IP로 바꿔서 송출하겠다는 내용이다.

**\*포트 포워딩 실습**

-p 옵션에 호스트포트:컨테이너포트 둘 다 적어주는 경우

▶docker run -d -p 80:80 --name web1 nginx



-p 옵션에 컨테이너의 포트만 적어주는 경우

▶docker run -d -p 80 --name web2 nginx



호스트의 80포트가 아닌 49153포트와 연결이 되었다.

-대문자 P 옵션

이 옵션의 경우 해당 컨테이너의 도커 파일에 EXPOSE되어있는 포트가 나와있다.

자동으로 이 포트를 찾아서 열어준다.

▶docker run -d -P --name web3 nginx



**\*유저 정의 네트워크 인터페이스 생성**

▶docker network create --driver bridge --subnet 192.168.100.0/24 --gateway 192.168.100.254 mynet

driver, subnet, gateway 옵션은 설정을 따로 안해주어도 디폴트 값이 자동으로 설정된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

▶docker run --name c1 -it --network mynet --ip 192.168.100.100 busybox

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

## **11-3. 컨테이너를 이용한 server&client 운영 예제**

mysql과 wrodpress컨테이너를 이용한 서비스 예제를 알아본다.

▶docker run -d --name mysql -v /dbdata:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=wordpress -e MYSQL\_PASSWORD=wordpress mysql:5.7

mysql 5.7버전을 이용하는 컨테이너를 실행한다.

/dbdata 디렉터리를 볼륨으로 이용한다.

▶docker run -d --name wordpress --link mysql:mysql -e WORDPRESS\_DB\_PASSWORD=wordpress -p 80:80 wordpress:4

mysql 컨테이너와 연동하기 위해 --link 옵션을 이용한다.

--link 컨테이너명:임의의 이름

다음과 같은 화면이 나오는데 여기서 여러 정보들을 확인 가능하다.