

# docker

## Requirements

- 시스템과 인프라 기초 지식
  - 시스템 기반의 구성 요소; 기능 요구사항(functional requirement), 비기능 요구사항(non-functional requirement): 신뢰성, 확장성, 운용성, 보안 등, 하드웨어, 네트워크, OS, 미들웨어
  - 클라우드와 온프레미스(on-premises; 자사에서 데이터 센터를 보유하고 시스템 구축부터 운용까지를 모두 수행하는 형태)
  - 시스템 기반의 구축/운용 흐름
    1. 시스템 구축 계획 및 요구사항 정의;
      - 시스템 구축 범위 선정
      - 인프라 요구사항 정의
      - 예산 책정
      - 프로젝트 체계화
      - 기존 시스템과의 연계
      - 시스템 마이그레이션 계획
    2. 인프라 설계 단계;
      - 인프라 아키텍처 설계
      - 네트워크 토폴로지 설계
      - 장비 선택, 조달(클라우드인 경우 서비스 선택)
      - OS, 미들웨어 선택, 조달(클라우드인 경우 서비스 선택)
      - 시스템 운용 설계
      - 시스템 마이그레이션 설계
    3. 인프라 구축 단계; (\*표시, 퍼블릭 클라우드에서는 필요 없는 경우가 많다)
      - 네트워크 부설\*
      - 서버 설치\*
      - OS 셋업\*
      - 미들웨어 셋업\*
      - 애플리케이션 및 라이브러리 설치
      - 테스트(네트워크 확인, 부하 테스트, 운용 테스트)
      - 시스템 릴리스 및 마이그레이션
    4. 운용단계;
      - 서버 프로세스, 네트워크, 리소스, 배치 Job 모니터링
      - 데이터 백업 및 정기 유지보수
      - OS, 미들웨어 버전 업그레이드
      - 애플리케이션 버전 업그레이드
      - 시스템 장애 시 대응
      - 사용자 서포트(헬프데스크)
- 하드웨어와 네트워크 기초 지식
  - 서버 장비; CPU, 메모리, 스토리지
  - 네트워크 주소; MAC 주소(물리주소/이더넷 주소), IP 주소
  - OSI 참조 모델과 통신 프로토콜

## OSI 7 Layer

| OSI 참조 모델 |           | 대표 프로토콜              | 대표 통신 기기    | Descriptions   |
|-----------|-----------|----------------------|-------------|--|
| 7계층(L7)   | 응용 계층     | HTTP, DNS, SMTP, SSH | 방화벽, 로드밸런서  | 애플리케이션 특화된 프로토콜 규정   |
| 6계층(L6)   | 표현 계층     |                      |             | 데이터의 저장 형식이나 압축, 문자 인코딩과 같은 데이터의 표현 형식을 규정   |
| 5계층(L5)   | 세션 계층     |                      |             | 커넥션 확립 타이밍이나 데이터 전송 타이밍을 규정. 애플리케이션 간에 일어나는 요청(request)과 응답(response)으로 구성   |
| 4계층(L4)   | 전송 계층     | TCP, UDP             |             | 전송 오류의 검출이나 재전송을 규정. 데이터를 통신 상대의 노드로 확실히 보내는 역할  |
| 3계층(L3)   | 네트워크 계층   | IP, ICMP             | 라우터, L3 스위치 | 서로 다른 네트워크 간에 통신을 하기 위한 규정   |
| 2계층(L2)   | 데이터 링크 계층 | Ethernet             | L2 스위치, 브리지 | 동일한 네트워크 안(동일 세그먼트)에 있는 노드 간의 통신을 규정. MAC 주소로 데이터 전송   |
| 1계층(L1)   | 물리 계층     |                      | 리피터 허브      | 통신 장비의 물리적 및 전기적 특성을 규정. 데이터를 어떻게 전압과 전류의 값으로 할당할지, 케이블이나 커넥터의 모양(RJ) 등을 규정. 트위스트 페어 케이블(STP/UTP), 100BASE-T, IEEE802.11 등 |

- 방화벽; 패킷 필터형, 애플리케이션 게이트웨이 형
- 라우터/레이어3 스위치; 라우팅 테이블 → 정적 경로(Static Route), 라우팅 프로토콜 → 동적 경로(Dynamic Route)
- Linux 기초 지식
- Linux 커널; 디바이스 관리, 프로세스 관리, 메모리 관리

## 셸의 종류

| 셸의 종류 |  |
|-------|--|
| 이름    | 특징   |
| bash  | 명령 이력, 디렉토리 스택, 명령 변환 기능, 명령이나 파일명의 자동보완 기능 등을 지원하는 고기능 셸. 대부분의 Linux 시스템이나 macOS(OS X)에 표준으로 탑재 |
| csh   | C 언어와 매우 비슷한 셸로, BSD 계열 OS에서 주로 이용   |
| tcsh  | csh를 개선한 버전으로 명령이나 파일명 등의 자동보완 기능을 가짐  |
| zsh   | bash와 호환성이 있는 셸로, 고속으로 작동하는 것이 특징  |

- Linux 파일 시스템; VFS(Virtual File System)
- 디렉토리 구성; /bin, /boot, /dev, /etc, /home, /proc, /sbin, /tmp, /usr, /var
- 보안 기능; 계정에 대한 권한 설정, 네트워크 필터링을 사용한 보안 기능 iptables, SELinux(Security-Enhanced Linux)
- 미들웨어 기초 지식
- 웹서버/웹 애플리케이션 서버; Apache HTTP Server, IIS(Internet Information Services), Nginx, ...
- 데이터베이스 서버;
  - RDBMS; MySQL, PostgreSQL, Oracle Database, ...
  - NoSQL; Redis, MongoDB, Apache Cassandra, ...
- 시스템 감시 툴; Zabbix, Datadog, Mackerel, ...
- 인프라 구성 관리 기초 지식
- 인프라 구성 관리; Chef, Ansible, Puppet, Itamae, ... Kubernetes

- 지속적 인티그레이션/지속적 딜리버리;
  - CI(Continuous Integration) 애플리케이션의 코드를 추가 및 수정할 때마다 테스트를 실행하고 확실하게 작동하는 코드를 유지하는 방법; Jenkins, ...

## Linux 디렉토리 구성

| 이름    | 설명   |
|-------|--|
| /bin  | ls 커맨드나 cp 커맨드와 같은 기본 커맨드를 저장하는 디렉토리. 특권 사용자, 일반 사용자 모두 이용하는 명령들이 배치되어 있음.   |
| /boot | Linux 커널 등의 OS의 시작에 필요한 파일을 배치하는 디렉토리. Linux 커널의 정체는 vmlinuz라는 이름의 파일.   |
| /dev  | 하드디스크, 키보드, 디바이스 파일을 저장하는 디렉토리. 예를 들어 /dev/had는 하드디스크, /dev/hda는 IDE 타입 하드디스크, /dev/sda는 SCSI 타입 하드디스크를 나타냄. /dev/tty는 표준입출력이 되는 단말 디바이스. 또한 '아무 것도 아니다'를 나타내는 /dev/null이라는 특수한 디바이스도 마련되어 있음. /dev/null은 필요가 없어진 출력을 버릴 때 사용하거나 빈 파일로 사용. |
| /etc  | OS나 애플리케이션이 작동하는 데 필요한 설정 파일이 저장되어 있는 디렉토리. 예를 들어 /etc/hosts는 IP 주소와 도메인명을 연결하는 파일이며, /etc/passwd는 사용자의 비밀번호가 저장되어 있음. 웹 서버를 시작할 때의 http 데몬 설정 파일도 이 디렉토리 아래에 배치됨.   |
| /home | 일반 사용자의 홈 디렉토리. 시스템 이용자가 자유롭게 사용할 수 있는 디렉토리. 독자적인 셸 설정 파일 등도 여기에 배치될 수 있음. 또한 특권 사용자(root)는 /root를 홈 디렉토리로 사용.   |
| /proc | 커널이나 프로세스에 관한 정보가 저장되어 있는 디렉토리. /proc 아래에 있는 숫자 폴더는 프로세스 ID. 또한 /proc/cpuinfo는 CPU 정보, /proc/partitions는 디스크의 파티션 정보, /proc/version은 Linux 커널의 버전 정보가 저장되어 있음.   |
| /sbin | 시스템 관리용 마운트가 저장되어 있는 디렉토리. 예를 들어 mount 커맨드나 reboot 커맨드 등 관리 커맨드는 /usr/sbin/이나 /usr/local/sbin 등에 배치되는 경우도 있음.  |
| /tmp  | 일시적으로 사용하는 파일 등을 저장하는 임시 디렉토리. 하드디스크에 저장되어 있는 보통의 파일 처럼 보이지만 /tmp는 보통 tmpfs 파일 시스템을 사용하여 메모리상에 전개되기 때문에 서버를 재시작하면 사라져 버림.  |
| /usr  | 각종 프로그램이나 커널 소스가 저장되는 디렉토리. /usr/local은 시스템 관리자가 애플리케이션을 설치하는 장소로 이용.  |
| /var  | 시스템의 가동과 함께 변화하는 파일을 놓아두는 디렉토리. 예를 들어 /var/log에는 가동 로그, /var/spool에는 애플리케이션이 임시 파일로 사용하는 스푼이 저장됨. 또한 메일 등의 큐나 프로세스의 다중 기동을 막기 위한 로그 파일 등도 배치.  |

## 컨테이너 기술과 Docker 개요

- 컨테이너; 호스트 OS 상에 논리적인 구획(컨테이너)을 만들고, 애플리케이션을 작동시키기 위해 필요한 라이브러리나 애플리케이션 등을 하나로 모아, 마치 별도의 서버인 것처럼 사용할 수 있게 만든 것.
- 서버 가상화
  - 호스트형 서버 가상화; Oracle VM Virtual Box, VMware VMware Workstation player 등
  - 하이퍼바이저형 서버 가상화; Microsoft Windows Server의 'Hyper-V', Citrix 'XenServer'
- Docker; 애플리케이션의 실행에 필요한 환경을 하나의 이미지로 모아두고, 그 이미지를 사용하여 다양한 환경에서 애플리케이션 실행 환경을 구축 및 운용하기 위한 오픈소스 플랫폼.  
<https://www.docker.com/>
- 웹 시스템 개발 시 애플리케이션을 제품 환경에서 가동시키기 위해 필요한 요소
  - 애플리케이션의 실행 모듈(프로그램 본체)
  - 미들웨어나 라이브러리
  - OS/네트워크 등과 같은 인프라 환경 설정

- Docker 기능;
  - Docker 이미지를 만드는 기능(Build)
  - Docker 이미지를 공유하는 기능 (Ship)
  - Docker 컨테이너를 작동시키는 기능 (Run)
- Docker 컴포넌트
  - Docker Enging; Docker의 핵심 기능
  - Docker Registry; 이미지 공개 및 공유
  - Docker Compose; 컨테이너 일원 관리
  - Docker Machine; Docker 실행 환경 구축
  - Docker Swarm; 클러스터 관리
- Docker의 작동 구조
  - 컨테이너를 구획하는 장치 (namespace); PID namespace, Network namespace, UID namespace, MOUNT namespace, UTS namespace, IPC namespace
  - 릴리스 관리 장치 (cgroups);
  - 네트워크 구성(가상 브리지/가상 NIC)
  - Docker 이미지의 데이터 관리 장치; AUFS, Btrfs, Device Mapper, OverlayFS, ZFS
- NAT(Network Address Translation); 프라이빗 네트워크 상의 컴퓨터와 인터넷 상의 서버간 통신이 성립되도록 변환하는 기술 글로벌 IP 주소와 프라이빗 IP 주소를 1:1로 변환.
- NATP(Network Address Port Translation) (IP 마스커레이드); 프라이빗 IP 주소와 함께 포트 번호도 함께 변환하는 기술

## Getting started docker

- 설치
- 작동 확인
  - hello world
  - 버전 확인 (docker version)
  - 실행 환경 확인 (docker system info)
  - 디스크 이용 상황 (docker system df)
- nginx 동작 예제; docker 이미지 다운로드 → nginx 작동 → nginx 작동 확인 → nginx 기동 정지

### 'hello world' on docker

docker container run <Docker 이미지명> <실행할 명령>

```
$ docker container run ubuntu:latest /bin/echo 'Hello world' # ubuntu 이미지를 바탕으로 docker 컨테이너를 작성 및 실행한 후 작성한 컨테이너 안에서 "Hello world" 표시
$ docker version # docker 버전, go 언어 버전, os, 아키텍처 등을 확인
$ docker system info # docker 실행 환경의 상세 설정 표시
$ docker system df # docker가 사용하고 있는 디스크의 이용 상황 표시
```

### nginx 작동 예제

```
$ docker pull nginx # nginx 이미지 다운로드
```

```
$ docker image ls # 다운로드 한 이미지 확인
$ docker container run --name webserver -d -p 80:80 nginx # 이미지를 사용하여
nginx 서버를 가동, 웹 브라우저에서 http://localhost:80 으로 접속하여 작동 확인
$ docker container ps # nginx 서버의 상태를 확인
$ docker container stats webserver # 컨테이너 가동 확인
$ docker stop webserver # 컨테이너 정지
$ docker start webserver # 컨테이너 가동
```

## Commands

### 이미지 조작

#### Docker Hub

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

#### 이미지 다운로드(**docker image pull**)

##### docker image pull

```
$ docker image pull [옵션] 이미지명[:태그명]
```

```
$ docker image pull centos:7 # CentOS의 이미지 취득
$ docker image pull -a centos # CentOS의 모든 태그 이미지 취득
$ docker image pull gcr.io.tensorflow/tensorflow # TensorFlow의 URL을 지정하여
이미지 취득
```

#### 이미지 목록 표시(**docker image ls**)

##### docker image ls

```
$ docker image ls [옵션] [리포지토리명]
```

| 옵션         | 설명                |
|------------|-------------------|
| -all, -a   | 모든 이미지를 표시        |
| -digests   | 다이제스트를 표시할지 말지    |
| -no-trunc  | 결과를 모두 표시         |
| -quiet, -q | Docker 이미지 ID만 표시 |

```
$ docker image ls
```

결과

| 항목         | 설명      |
|------------|---------|
| REPOSITORY | 이미지 이름  |
| TAG        | 이미지 태그명 |
| IMAGE ID   | 이미지 ID  |
| CREATED    | 작성일     |
| SIZE       | 이미지 크기  |

- DCT(Docker Content Trust)

```
$ export DOCKER_CONTENT_TRUST = 1 # DCT 기능의 유효화, 서명된 이미지를 다운로드 할 때
이미지 작성자의 공개키를 사용하여 이미지가 진짜인지 아닌지 확인. 만일 변조된 경우 그 이미지를 무효
로 만듦. 이 공개키를 Tagging Key라고 함.
$ export DOCKER_CONTENT_TRUST = 0 # DCT 기능의 무효화
```

## 이미지 상세 정보 확인(docker image inspect)

```
$ docker image inspect centos:7 # centos:7 이미지 상세 정보 확인
$ docker image inspect --format="{{ .Os }}" centos:7 # OS 정보 취득
$ docker image inspect --format="{{ .ContainerConfig.Image }}" centos:7 #
image 정보 취득
```

- 결과는 JSON(JavaScript Object Notation) 형식으로 표시
  1. 이미지 ID
  2. 작성일
  3. Docker 버전
  4. CPU 아키텍처

## 이미지 태그 설정(docker image tag)

### docker image tag

```
<Docker Hub 사용자명>/이미지명:[태그명]
```

```
$ docker image tag nginx alexlevine/webserver:1.0 # alexlevine 사용자명의
webserver 이미지에 1.0의 태그 설정
```

## 이미지 검색(docker search)

### docker search

```
docker search [옵션] <검색 키워드>
```

| 지정할 수 있는 주요 옵션   |                   |
|------------------|-------------------|
| 옵션               | 설명                |
| -no-trunc        | 결과를 모두 표시         |
| -limit           | n건의 검색 결과를 표시     |
| -fileter=stars=n | 즐겨찾기의 수(n 이상)를 지정 |

```
$ docker search nginx # Docker Hub에 공개되어 있는 이미지 검색
```

| docker search 명령 결과 |                                   |
|---------------------|-----------------------------------|
| 항목                  | 설명                                |
| NAME                | 이미지 이름                            |
| DESCRIPTION         | 이미지 설명                            |
| STARS               | 즐겨찾기 수                            |
| OFFICIAL            | 공식 이미지인지 아닌지                      |
| AUTOMATED           | Dockerfile을 바탕으로 자동 생성된 이미지인지 아닌지 |

## 이미지 삭제(docker image rm)

### docker image rm

```
docker image rm [옵션] 이미지명 [이미지명]
```

| 지정할 수 있는 주요 옵션 |                 |
|----------------|-----------------|
| 옵션             | 설명              |
| -force, -f     | 이미지를 강제로 삭제     |
| -no-prune      | 중간 이미지를 삭제하지 않음 |

```
$ docker image rm nginx # nginx 이미지 삭제
```

### docker image prune

- 사용하지 않은 Docker 이미지 삭제

```
docker image prune [옵션]
```

| 지정할 수 있는 주요 옵션 |                    |
|----------------|--------------------|
| 옵션             | 설명                 |
| -all, -a       | 사용하지 않은 이미지를 모두 삭제 |
| -force, -f     | 이미지를 강제로 삭제        |

### Docker Hub에 로그인(docker login)

#### docker login

```
docker login [옵션] [서버]
```

| 지정할 수 있는 주요 옵션 |      |
|----------------|------|
| 옵션             | 설명   |
| -password, -p  | 비밀번호 |
| -username, -u  | 사용자명 |

```
$ docker login
Username: [등록한 사용자명]
Password: [등록한 비밀번호]
Login Succeeded
```

### 이미지 업로드(docker image push)

#### docker image push

```
docker image push 이미지명[:태그명]
```

- 이미지명; <Docker Hub 사용자명>/이미지명[:태그명]

```
$ docker image push alexlevine/webserver:1.0
```

### Docker Hub에서 로그아웃(docker logout)

#### docker logout

```
docker logout [서버명]
```

### 컨테이너 생성/시작/정지

## Docker 컨테이너의 라이프 사이클

- 컨테이너 생성; docker container create
- 컨테이너 생성 및 시작; docker container run
- 컨테이너 시작; docker container start
- 컨테이너 정지; docker container stop
- 컨테이너 삭제; docker container rm

### 컨테이너 생성 및 시작(docker container run)

#### docker container run

docker container run [옵션] 이미지명[:태그명] [인수]

| 지정할 수 있는 주요 옵션   |   |
|------------------|---|
| 옵션               | 설명  |
| -attach, -a      | 표준 입력(STDIN), 표준 출력(STDOUT), 표준 오류 출력(STDERR)에 어태치한다. |
| -cidfile         | 컨테이너 ID를 파일로 출력한다.                                    |
| -detach, -d      | 컨테이너를 생성하고 백그라운드에서 실행한다.                              |
| -interactive, -i | 컨테이너의 표준 입력을 연다.                                      |
| -try, -t         | 단말기 디바이스를 사용한다.                                       |

```
$ docker container run -it --name "test1" centos /bin/cal
# docker container run; 컨테이너를 생성 및 실행
# -it; 콘솔에 결과를 출력하는 옵션
# --name "test1"; 컨테이너 명
# centos; 이미지명
# /bin/cal; 컨테이너에서 실행할 명령

$ docker container run -it --name "test2" centos /bin/bash # bash 실행
```

### 컨테이너의 백그라운드 실행(docker container run)

#### docker container run

docker container run [실행 옵션] 이미지명[:태그명] [인수]

| 지정할 수 있는 주요 옵션  |                           |
|---|---------------------------|
| 옵션  | 설명                        |
| -detach, -d   | 백그라운드에서 실행                |
| -user, -u   | 사용자명을 지정                  |
| -restart=[no on-failure on-failure:횟수n always unless-stopped] | 명령의 실행 결과에 따라 재시작을 하는 옵션  |
| -rm   | 명령 실행 완료 후에 컨테이너를 자동으로 삭제 |

```
$ docker container run -d centos /bin/ping localhost
# docker container run; 컨테이너를 생성 및 실행
```

```
# -d; 백그라운드에서 실행하는 옵션
# centos; 이미지명
# /bin/ping localhost; 컨테이너에서 실행할 명령
fbcdab0e9417.....
$ docker container logs -t fbcdab0e9417 # 컨테이너의 로그 확인
```

| restart 옵션     |                                |
|----------------|--------------------------------|
| 설정값            | 설명                             |
| no             | 재시작하지 않는다.                     |
| on-failure     | 종료 스테이터스가 0이 아닐 때 재시작한다.       |
| on-failure:횟수n | 종료 스테이터스가 0이 아닐 때 n번 재시작한다.    |
| always         | 항상 재시작한다.                      |
| unless-stopped | 최근 컨테이너가 정지 상태가 아니라면 항상 재시작한다. |

```
$ docker container run -it --restart=always centos /bin/bash # 컨테이너를 항상 재시작
```

### 컨테이너의 네트워크 설정(docker container run)

#### docker container run

docker container run [네트워크 옵션]: 이미지명[:태그명] [인수]

| 지정할 수 있는 옵션   |                                   |
|---|-----------------------------------|
| 옵션  | 설명                                |
| -add-host=[호스트명:IP 주소]                              | 컨테이너의 /etc/hosts에 호스트명과 IP 주소를 정의 |
| -dns=[IP 주소]  | 컨테이너용 DNS 서버의 IP 주소 지정            |
| -expose   | 지정한 범위의 포트 번호를 할당                 |
| -mac-address=[MAC 주소]                               | 컨테이너의 MAC 주소를 지정                  |
| -net=[bridge none container:<name id> host NETWORK] | 컨테이너의 네트워크를 지정                    |
| -hostname, -h                                       | 컨테이너 자신의 호스트명을 지정                 |
| -publish, -p[호스트의 포트 번호]:[컨테이너의 포트 번호]              | 호스트와 컨테이너의 포트 매핑                  |
| -publish-all, -p                                    | 호스트의 임의의 포트를 컨테이너에 할당             |

```
$ docker container run -d --dns 192.168.1.1 nginx # 컨테이너의 DNS 서버 지정
$ docker container run -d --mac-address="92:d0:c6:0a:29:33" centos # MAC 주소 지정
2a4f6cf4da30a...
$ docker container inspect --format="{{ .Config.MacAddress }}" 2a5f
92:d0:c6:0a:29:33
$ docker container run -it --add-host test.com:192.168.1.1 centos # 호스트명과 IP 주소 정의
$ docker container run -it --hostname www.test.com --add-host node1.test.com:6 192.168.1.1 centos # 호스트명 설정
```

| -net 옵션의 지정         |                           |
|---------------------|---------------------------|
| 설정값                 | 설명                        |
| bridge              | 브리지 연결(기본값)을 사용한다.        |
| none                | 네트워크에 연결하지 않는다.           |
| container:[name id] | 다른 컨테이너의 네트워크를 사용한다.      |
| host                | 컨테이너가 호스트 OS의 네트워크를 사용한다. |
| NETWORK             | 사용자 정의 네트워크를 사용한다.        |

- 사용자 정의 네트워크 작성

```
$ docker network create -d bridge webap-net
$ docker container run --net=webap-net -it centos
```

## 자원을 지정하여 컨테이너 생성 및 실행(docker container run)

### docker container run

docker container run [자원 옵션] 이미지명[:태그명] [인수]

| 지정할 수 있는 주요 옵션                       |   |
|--------------------------------------|---|
| -cpu-shares, -c                      | CPU의 사용 배분(비율)                            |
| -memory, -m                          | 사용할 메모리를 제한하여 실행<br>(단위는 b, k, m, g 중 하나) |
| -volume=[호스트의 디렉토리]:[컨테이너의 디렉토리], -v | 호스트와 컨테이너의 디렉토리를 공유                       |

```
$ docker container run --cpu-shares=512 --memory=1g centos # CPU 시간의 상대
비율과 메모리 사용량을 지정
$ docker container run -v /Users/alex/webap:/usr/share/nginx/html nginx # 디
렉토리 공유
```

## 컨테이너를 생성 및 시작하는 환경을 지정(docker container run)

### docker container run

docker container run [환경설정 옵션] 이미지명[:태그명] [인수]

| 지정할 수 있는 주요 옵션          |                            |
|-------------------------|----------------------------|
| 옵션                      | 설명                         |
| -env=[환경변수], -e         | 환경변수를 설정한다.                |
| -env-file=[파일명]         | 환경변수를 파일로부터 설정한다.          |
| -read-only=[true false] | 컨테이너의 파일 시스템을 읽기 전용으로 만든다. |
| -workdir=[패스], -w       | 컨테이너의 작업 디렉토리를 지정한다.       |
| -u, -user=[사용자명]        | 사용자명 또는 UID를 지정한다.         |

```
$ docker container run -it -e foo=bar centos /bin/bash # 환경변수 foo 설정
```

```
$ cat env_list # env_list 파일 생성
hoge=fuga
foo=bar
```

```
$ docker container run -it --env-file=env_list centos /bin/bash # evn_list  
파일로 환경변수의 일괄 설정  
  
$ docker container run -it -w=/tensorflow centos /bin/bash # 작업 디렉토리 설정
```

### 가동 컨테이너 목록 표시(**docker container ls**)

---

### 컨테이너 가동 확인(**docker container stats**)

---

### 컨테이너 시작(**docker container start**)

---

### 컨테이너 정지(**docker container stop**)

---

### 컨테이너 재시작(**docker container restart**)

---

### 컨테이너 삭제(**docker container rm**)

---

### 컨테이너 중단/재개(**docker container pause/docker container unpause**)

---

## 컨테이너 네트워크

### 네트워크 목록 표시(**docker network ls**)

### 네트워크 작성(**docker network create**)

### 네트워크 연결(**docker network connect/docker network disconnect**)

### 네트워크 상세 정보 확인(**docker network inspect**)

네트워크 삭제(**docker network rm**)

가동중인 컨테이너 조작

가동 컨테이너 연결(**docker container attach**)

가동 컨테이너에서 프로세스 실행(**docker container exec**)

가동 컨테이너의 프로세스 확인(**docker container top**)

가동 컨테이너의 포트 전송 확인(**docker container port**)

컨테이너 이름 변경(**docker container rename**)

컨테이너 안의 파일을 복사(**docker container cp**)

컨테이너 조작의 차분 확인(**docker container diff**)

이미지 생성

컨테이너로부터 이미지 작성(**docker container commit**)

컨테이너를 **tar** 파일로 출력(**docker container export**)

**tar** 파일로부터 이미지 작성(**docker image import**)

이미지 저장(**docker image save**)

이미지 읽어 들이기(**docker image load**)

불필요한 이미지/컨테이너를 일괄 삭제(**docker system prune**)

---

## Dockerfile을 사용한 코드에 의한 서버 구축

### Dockerfile을 사용한 구성 관리

## Dockerfile이란?

## Dockerfile의 기본 구문

## Dockerfile 작성

## Dockerfile의 빌드와 이미지 레이어

## Dockerfile로부터 Docker 이미지 만들기

## Docker 이미지의 레이어 구조

## 멀티스테이지 빌드를 사용한 애플리케이션 개발

## Dockerfile 만들기

## Docker 이미지의 빌드

## Docker 컨테이너의 시작

## 명령 및 데몬 실행

## 명령 실행(RUN 실행)

## 데몬 실행(CMD 명령)

## 데몬 실행(ENTRYPOINT 명령)

## 빌드 완료 후에 실행되는 명령(ONBUILD 명령)

## 시스템 콜 시그널의 설정(STOPSIGNAL 명령)

## 컨테이너의 헬스 체크 명령(HEALTHCHECK 명령)

## 환경 및 네트워크 설정

환경변수 설정(**ENV** 명령)

작업 디렉토리 지정(**WORKDIR** 명령)

사용자 지정(**USER** 명령)

라벨 지정(**LABEL** 명령)

포트 설정(**EXPOSE** 명령)

**Dockerfile** 내 변수의 설정(**ARG** 명령)

기본 셸 설정(**SHELL** 명령)

파일 설정

파일 및 디렉토리 추가(**ADD** 명령)

파일 복사(**COPY** 명령)

블륨 마운트(**VOLUME** 명령)

---

## Docker 이미지 공개

Docker 이미지의 자동 생성 및 공개

Automated Build의 흐름

GitHub에 공개하기

Docker Hub의 링크 설정

Dockerfile의 빌드

Docker 이미지 확인

## Docker Registry를 사용한 프라이빗 레지스트리 구축

로컬 환경에 Docker 레지스트리 구축하기

Docker 이미지 업로드

Docker 이미지의 다운로드와 작동 확인

클라우드 서비스를 사용한 프라이빗 레지스트리 구축

Google Container Registry 준비하기

Docker 이미지의 업로드

Docker 이미지의 다운로드와 작동 확인

---

## 여러 컨테이너의 운용 관리

여러 컨테이너 관리의 개요

웹 3계층 시스템 아키텍처

영구 데이터의 관리

### Docker Compose

웹 애플리케이션을 로컬에서 움직여 보자

Compose 구성 파일의 작성

여러 Docker 컨테이너 시작

여러 Docker 컨테이너 정지

Docker Compose를 사용한 여러 컨테이너의 구성 관리

## **docker-compose.yml의 개요**

이미지 지정(**image**)

이미지 빌드(**build**)

컨테이너 안에서 작동하는 명령 지정(**command/entrypoint**)

컨테이너 간 연결(**links**)

컨테이너 간 통신(**ports/expose**)

서비스의 의존관계 정의(**depends\_on**)

컨테이너 환경변수 지정(**environment/env\_file**)

컨테이너 정보 설정(**container\_name/labels**)

컨테이너 데이터 관리(**volumes/volumes\_from**)

## **Docker Compose를 사용한 여러 컨테이너의 운용**

**Docker Compose**의 버전 확인

**Docker Compose**의 기본 명령

여러 컨테이너의 생성(**up**)

여러 컨테이너 확인(**ps/logs**)

컨테이너에서 명령 실행(**run**)

여러 컨테이너 시작/정지/재시작(**start/stop/restart**)

여러 컨테이너 일시 정지/재개(**pause/unpause**)

서비스의 구성 확인(**port/config**)

여러 컨테이너 강제 정지/삭제(**kill/rm**)

여러 리소스의 일괄 삭제(**down**)

---

## 멀티호스트 환경에서 **Docker** 실행 환경 구축

멀티호스트 환경에서 컨테이너 관리의 개요

멀티호스트 환경과 클러스터링

**Docker Machine**이란?

웹 애플리케이션을 서비스 공개해 보자

**Docker** 실행 환경 작성

웹 애플리케이션 전개

**Docker** 실행 환경 삭제

**Docker Machine**을 사용한 실행 환경 구축

**Docker Machine**의 기본 명령

실행 환경 작성(**create**)

실행 환경 목록 표시(**ls/status/url**)

실행 환경에 대한 **SSH** 연결(**ssh**)

실행 환경 시작/정지/재시작(**start/stop/restart**)

실행 환경으로부터 파일 다운로드(**scp**)

실행 환경 삭제(**rm/kill**)

실행 환경 정보 확인(**ip/inspect**)

---

## 클라우드를 사용한 **Docker** 실행 환경 구축

클라우드 환경에서 **Docker** 오케스트레이션하기

분산 환경에서의 컨테이너 운용 관리

퍼블릭 클라우드가 제공하는 매니지드 서비스

**Google Cloud Platform**의 컨테이너 관련 서비스

**Kubernetes**의 개요

**Kubernetes**의 서버 구성

애플리케이션 구성 관리(**Pod, ReplicaSet, Deployment**)

네트워크 관리(**Service**)

**Label**을 사용한 리소스 식별

**Kubernetes**의 구조

**GCP**를 사용한 **Docker** 애플리케이션 개발

애플리케이션 개발 흐름

소스코드 관리(**Cloud Source Repositories**)

**Docker** 이미지 빌드(**Cloud Container Builder**)

**GCP**를 사용한 **Docker** 애플리케이션 실행 환경 구축

## **Kubernetes 클러스터 구축**

애플리케이션의 설정 정보 관리(**ConfigMap, Secrets**)

앱의 전개(**Deployment**)

서비스 공개(**Service**)

앱의 버전업(**Blue-Green Deployment**)

배치 잡 실행(**CronJob**)

---

## **클라우드를 사용한 Docker 실행 환경의 운용 관리**

시스템 운용의 기초 지식

가용성 관리

수용성(**Capacity**) 관리

시스템 감시

## **GKE를 사용한 Docker 실행 환경의 운용**

**Kubernetes**의 스테이터스 확인

**Kubernetes**의 Pod 관리

**Kubernetes**의 노드 관리

**Kubernetes**의 리소스 작성/삭제/변경

**Kubernetes**의 업그레이드/다운그레이드

**Stackdriver**에서 로그 확인

## node docker image

```
$ docker exec -it node bash
```

## nginx-php-fpm docker image

```
richarvey/nginx-php-fpm
```

```
$ docker run --name ngx-php -d richarvey/nginx-php-fpm
```

```
$ docker exec -e 'DOMAIN=theta5912.net' -e 'GIT_EMAIL=alex@theta5912.net' -e 'WEBROOT=/var/www/html' -t ngx-php /usr/bin/letsencrypt-setup
```

```
$ docker exec -t -i ngx-php /bin/bash
```

```
$ docker exec -e 'DOMAIN=theta5912.net'
```

```
$ docker exec -e 'GIT_EMAIL=alex@theta5912.net'
```

```
$ docker exec -t ngx-php /usr/bin/letsencrypt-setup (90days)
```

```
$ docker exec -t ngx-php /usr/bin/letsencrypt-renew
```

```
$ docker exec -e 'DOMAIN=theta5912.net' -t ngx-php /usr/bin/letsencrypt-renew
```

```
$ docker start ngx-php
```

```
$ docker commit -a "Alex Levine<alex@theta5912.net>" -m "update dokuwiki, December 29, 2017. Friday" ngx-php
```

```
---
```

```
setting the timezone
```

```
# apk add tzdata
```

```
# ls /usr/share/zoneinfo
```

```
# cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

```
# echo "Asia/Seoul" > /etc/timezone
```

```
# date
```

```
# apk del tzdata
```

```
in dokuwiki
```

```
dokuwiki/inc/init.php 88
```

```
date_default_timezone_set("Asia/Seoul");
```

```
---
```

```
# apk update
```

```
# apk upgrade
```

```
# rm -rf /var/cache/apk/*
```

```
---
# export DOMAIN=theta5912.net
# export GIT_EMAIL=alex@theta5912.net
# export WEBROOT=/var/www/html
# /usr/bin/letsencrypt-setup

---
# wget http://download.dokuwiki.org/src/dokuwiki/dokuwiki-stable.tgz
# tar xvf dokuwiki-stable.tgz --strip 1

---
cp

host -> container
$ docker cp /path/foo.txt mycontainer:/path/foo.txt

container -> host
$ docker cp mycontainer:/path/foo.txt /path/foo.txt

---
$ docker run -i -t ---name <container name> -v <host directory>
```

---

## Google Cloud Platform 사용법

### A.1 계정 등록

[1] 등록 시작 [2] 계정 정보 등록

### A.2 프로젝트 작성과 삭제

[1] 프로젝트 작성 [2] 프로젝트명 설정 [3] 프로젝트 삭제

### A.3 Cloud Console 사용법

틀과 서비스 대시보드

### A.4 Cloud Shell 사용법

### A.5 Cloud SDK 설치하기

From:

<http://theta5912.net/> - reth

Permanent link:

<http://theta5912.net/doku.php?id=public:computer:docker&rev=1628328886>

Last update: **2021/08/07 18:34**

