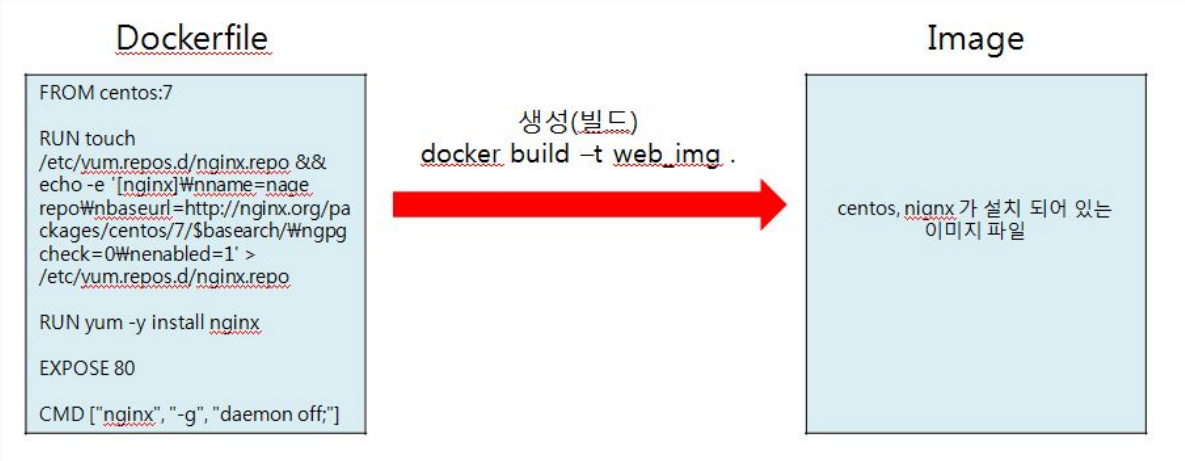
Docker 7

7.Dockerfile

- **Dockerfile 이란?**

도커는 기본적으로 이미지가 있어야 컨테이너를 생성하고 동작시킬 수 있습니다. dockerfile은 필요한 최소한의 패키지를 설치하고 동작하기 위한 자신만의 설정을 담은 파일이고, 이 파일로 이미지를 생성(빌드)하게 됩니다.

패키지 설치, 환경 변수 변경, 설정 파일 변경 등 다양한 작업을 하나하나 컨테이너를 만들고 설정을 적용할 필요 없이 dockerfile을 사용하여 적용할 수 있고, 유저의 실수로 인한 설정 누락 예방 등 다양한 장점이 있습니다.



7.Dockerfile

- Dockerfile은 다음과 같이 `<명령> <매개 변수>` 형식으로 작성합니다.
- #은 주석입니다.
- 명령은 대소문자를 구분하지 않지만 보통 대문자로 작성합니다.
- Docker는 Dockerfile에 작성된 명령을 순서대로 처리합니다.
- Dockerfile에서 명령은 항상 FROM으로 시작해야 합니다.
  FROM이 없거나 FROM 앞에 다른 명령이 있으면 이미지가 생성되지 않습니다.
- 각 명령은 독립적으로 실행됩니다.
  예를 들면 `RUN cd /home/hello`로 디렉터리를 이동하더라도 뒤에 오는 명령에는 영향을 주지 않습니다.
- 이미지를 생성할 때는 Dockerfile이 있는 디렉터리에서 `docker build`명령을 사용합니다.

```
# 주석
FROM scratch
```

```
$ sudo docker build --tag example .
$ sudo docker build --tag pyrasis/example .
```

- `--tag` 또는 `-t` 옵션으로 이미지 이름을 설정할 수 있습니다. Docker Hub에 이미지를 올리려면 **pyrasis/example**처럼 / 앞에 사용자명을 붙이면 됩니다.
- 이미지 이름을 설정하지 않아도 이미지는 생성됩니다. 이때 이미지를 사용하려면 이미지 ID를 지정하면 됩니다.

7-1.dockerignore

Dockerfile과 같은 디렉터리에 들어있는 모든 파일을 컨텍스트(context)라고 합니다.
특히 이미지를 생성할 때 컨텍스트를 모두 Docker 데몬에 전송하므로 필요 없는
파일이나 디렉토리를 제외하고 싶을 때는 .dockerignore 파일을 사용하면 됩니다.

.dockerignore

```
example/hello.txt
example/*.cpp
wo*
*.cpp
.git
.svn
```

특정 파일이나 디렉터리를 제외할 수 있고, 보통 *를 주로 사용합니다.
버전 관리 시스템을 이용하여 Dockerfile과 필요한 파일을 관리할 때 .git, .svn과 같은 디렉터리는 제외해줍니다.

7-2.FROM

- FROM은 어떤 이미지를 기반으로 이미지를 생성할지 설정합니다.
- Dockerfile로 이미지를 생성할 때는 항상 기존에 있는 이미지를 기반으로 생성하기 때문에 FROM은 반드시 설정해야 합니다.
- 이미지 이름과 태그를 함께 설정할 수도 있습니다.
- 이미지 이름만 설정하면 latest를 사용합니다.
- 이미지 이름은 생략할 수 없습니다.

Dockerfile

```
FROM ubuntu
```

Dockerfile

```
FROM ubuntu:14.04
```

`FROM <이미지>` 또는 `FROM <이미지>:<태그>` 형식입니다.

FROM은 항상 설정해야 하고 맨 처음에 와야 합니다. 이미지를 생성할 때 FROM에 설정한 이미지가 로컬에 있으면 바로 사용하고, 없으면 Docker Hub에서 받아옵니다.

Dockerfile 파일 하나에 FROM을 여러 개 설정할 수 있습니다. FROM을 두 개 설정했다면 이미지가 두 개 생성됩니다. `--tag` 옵션으로 이미지 이름을 설정했다면 맨 마지막 FROM에 적용됩니다.

7-3.MAINTAINER

MAINTAINER는 이미지를 생성한 사람의 정보를 설정합니다. 형식은 자유이며 보통 다음과 같이 이름과 이메일을 입력합니다.

Dockerfile

```
MAINTAINER    Hong, Gildong <gd@yuldo.com>
```

`MAINTAINER <작성자 정보>` 형식입니다.
MAINTAINER는 생략할 수 있습니다.

7-4.RUN

RUN은 FROM에서 설정한 이미지 위에서 스크립트 혹은 명령을 실행합니다.
여기서 RUN으로 실행한 결과가 새 이미지로 생성되고, 실행 내역은 이미지의 히스토리에 기록됩니다.

셸(/bin/sh)로 명령 실행하기

Dockerfile

```
RUN apt-get install -y nginx
RUN echo "Hello Docker" > /tmp/hello
RUN curl -sSL https://golang.org/dl/go1.3.1.src.tar.gz | tar -v -C /usr/local -xz
RUN git clone https://github.com/docker/docker.git
```

`RUN <명령>` 형식이며 셸 스크립트 구문을 사용할 수 있습니다. FROM으로 설정한 이미지에 포함된 **/bin/sh** 실행 파일을 사용하게 되며 **/bin/sh** 실행 파일이 없으면 사용할 수 없습니다.

셸 없이 바로 실행하기

Dockerfile

```
RUN ["apt-get", "install", "-y", "nginx"]
RUN ["/user/local/bin/hello", "--help"]
```

`RUN ["<실행 파일>", "<매개 변수1>", "<매개 변수2>"]` 형식입니다. 실행 파일과 매개 변수를 배열 형태로 설정합니다. FROM으로 설정한 이미지의 **/bin/sh** 실행 파일을 사용하지 않는 방식입니다. 셸 스크립트 문법이 인식되지 않으므로 셸 스크립트 문법과 관련된 문자를 그대로 실행 파일에 넘겨줄 수 있습니다.

RUN으로 실행한 결과는 캐시되며 다음 빌드 때 재사용합니다.
캐시된 결과를 사용하지 않으려면 `docker build` 명령에서 `--no-cache` 옵션을 사용하면 됩니다.

7-5.CMD

CMD는 컨테이너가 시작되었을 때 스크립트 혹은 명령을 실행합니다.
즉 `docker run` 명령으로 컨테이너를 생성하거나, `docker start` 명령으로 정지된 컨테이너를 시작할 때 실행됩니다.
CMD는 Dockerfile에서 한 번만 사용할 수 있습니다.

셸(/bin/sh)로 명령 실행하기

Dockerfile

```
CMD touch /home/hello/hello.txt
```

`CMD <명령>` 형식이며 셸 스크립트 구문을 사용할 수 있습니다.
FROM으로 설정한 이미지에 포함된 **/bin/sh** 실행 파일을
사용하게 되며 **/bin/sh** 실행 파일이 없으면 사용할 수 없습니다.

셸 없이 바로 실행하기

Dockerfile

```
CMD ["redis-server"]
```

셸 없이 바로 실행할 때 매개 변수 설정하기

Dockerfile

```
CMD ["mysqld", "--datadir=/var/lib/mysql", "--user=mysql"]
```

`CMD ["<실행 파일>", "<매개 변수1>", "<매개 변수2>"]` 형식입니다. 실행 파일과 매개 변수를 배열 형태로 설정합니다. FROM으로 설정한 이미지의 **/bin/sh** 실행 파일을 사용하지 않는 방식입니다. 셸 스크립트 문법이 인식되지 않으므로 셸 스크립트 문법과 관련된 문자를 그대로 실행 파일에 넘겨줄 수 있습니다.

7-5.CMD

ENTRYPOINT를 사용하였을 때

Dockerfile

```
ENTRYPOINT ["echo"]
CMD ["hello"]
```

`CMD ["<매개 변수1>", "<매개 변수2>"]` 형식입니다. ENTRYPOINT에 설정한 명령에 매개 변수를 전달하여 실행합니다. Dockerfile에 ENTRYPOINT가 있으면 CMD는 ENTRYPOINT에 매개 변수만 전달하는 역할을 합니다. 그래서 CMD 독자적으로 파일을 실행할 수 없게 됩니다.

```
$ sudo docker build --tag example .
$ sudo docker run example
hello
```

7-6.ENTRYPOINT

ENTRYPOINT는 컨테이너가 시작되었을 때 스크립트 혹은 명령을 실행합니다.
즉 **docker run** 명령으로 컨테이너를 생성하거나, **docker start** 명령으로 컨테이너를 시작할 때 실행됩니다.
ENTRYPOINT는 **Dockerfile**에서 단 한번만 사용할 수 있습니다.

셸(/bin/sh)로 명령 실행하기

Dockerfile

```
ENTRYPOINT touch /home/hello/hello.txt
```

`ENTRYPOINT <명령>` 형식이며 셸 스크립트 구문을 사용할 수 있습니다. FROM으로 설정한 이미지에 포함된 **/bin/sh** 실행 파일을 사용하게 되며 **/bin/sh** 실행 파일이 없으면 사용할 수 없습니다.

Dockerfile

```
ENTRYPOINT ["/home/hello/hello.sh"]
```

Dockerfile

```
ENTRYPOINT ["/home/hello/hello.sh", "--hello=1", "--world=2"]
```

`ENTRYPOINT ["<실행 파일>", "<매개 변수1>", "<매개 변수2>"]` 형식입니다. 실행 파일과 매개 변수를 배열 형태로 설정합니다. FROM으로 설정한 이미지의 **/bin/sh** 실행 파일을 사용하지 않는 방식입니다. 셸 스크립트 문법이 인식되지 않으므로 셸 스크립트 문법과 관련된 문자를 그대로 실행 파일에 넘겨줄 수 있습니다.

7-6.ENTRYPOINT

CMD와 ENTRYPOINT는 컨테이너가 생성될 때 명령이 실행되는 것은 동일하지만 `docker run` 명령에서 동작 방식이 다릅니다.

다음과 같이 Dockerfile에서 CMD로 `echo` 명령을 사용하여 **hello**를 출력합니다.

CMD

Dockerfile

```
FROM ubuntu:latest
CMD ["echo", "hello"]
```

```
$ sudo docker build --tag example .
$ sudo docker run example echo world
world
```

컨테이너를 생성할 때 `docker run <이미지> <실행할 파일>` 형식인데 이미지 다음에 실행할 파일을 설정할 수 있습니다. `docker run` 명령에서 실행할 파일을 설정하면 CMD는 무시됩니다.

`CMD ["echo", "hello"]`는 무시되고 `docker run` 명령에서 설정한 `echo world`가 실행되어 **world**가 출력되었습니다. `docker run` 명령에서 설정한 <실행할 파일>과 Dockerfile의 CMD는 같은 기능입니다.

CMD가 먼저 읽히고 docker run의 매개변수가 덮어쓴다

ENTRYPOINT

Dockerfile

```
FROM ubuntu:latest
ENTRYPOINT ["echo", "hello"]
```

```
$ sudo docker build --tag example .
$ sudo docker run example echo world
hello echo world
```

Dockerfile을 빌드하여 `docker run` 명령으로 실행합니다.
`docker run` 명령에서 실행할 파일을 설정하면 ENTRYPOINT 무시되지 않고,
실행할 파일 설정 자체를 매개 변수로 받아서 처리합니다.

```
$ echo hello echo world
hello echo world
```

`ENTRYPOINT ["echo", "hello"]`에서 `echo hello`가 실행되어 **hello**가 출력되고, `docker run` 명령에서 설정한 내용이 `ENTRYPOINT ["echo", "hello"]`의 매개 변수로 처리되어 **echo world**도 함께 출력됩니다.
셸에서는 다음과 같이 표현할 수 있습니다.

7-6.ENTRYPOINT

```
$ sudo docker run --entrypoint="cat" example /etc/hostname
9efe43ea4d40
```

`docker run` 명령에서 `--entrypoint` 옵션으로도 설정할 수 있습니다. `--entrypoint` 옵션으로 `cat`을 실행하고 **/etc/hostname** 파일의 내용을 출력합니다.

`--entrypoint` 옵션을 설정하면 Dockerfile에 설정한 ENTRYPOINT는 무시됩니다.

7-7.EXPOSE

EXPOSE는 호스트와 연결할 포트 번호를 설정합니다. docker run 명령의 --expose 옵션과 동일합니다.

`EXPOSE <포트 번호>` 형식입니다. EXPOSE 하나로 포트 번호를 두 개 이상 동시에 설정할 수도 있습니다.

EXPOSE는 호스트와 연결만 할 뿐 외부에 노출은 되지 않습니다. 포트를 외부에 노출하려면 `docker run` 명령의 `-p`, `-P` 옵션을 사용해야 합니다.

Dockerfile

```
EXPOSE 80
EXPOSE 443
```

Dockerfile

```
EXPOSE 80 443
```

7-8.ENV

ENV는 환경 변수를 설정합니다. ENV로 설정한 환경 변수는 RUN, CMD, ENTRYPOINT에 적용됩니다.

Dockerfile

```
ENV GOPATH /go
ENV PATH /go/bin:$PATH
```

`ENV <환경 변수> <값>` 형식입니다. 환경 변수를 사용할 때는 `$`를 사용하면 됩니다.

다음은 ENV에서 설정한 환경 변수를 CMD로 출력합니다.

Dockerfile

```
ENV HELLO 1234
CMD echo $HELLO
```

```
$ sudo docker build --tag example .
$ sudo docker run example
1234
```

ENV에서 설정한 **HELLO**의 값 **1234**가 출력됩니다.

```
$ sudo docker run -e HELLO=4321 example
4321
```

환경 변수는 `docker run` 명령에서도 설정할 수 있습니다.

`-e <환경 변수>=<값>` 형식입니다.
`-e` 옵션은 여러 번 사용할 수 있고, `--env` 옵션과 같습니다.

7-9.ADD

ADD는 파일을 이미지에 추가합니다.

Dockerfile

```
ADD hello-entrypoint.sh /entrypoint.sh
ADD hello-dir /hello-dir
ADD zlib-1.2.8.tar.gz /
ADD hello.zip /
ADD http://example.com/hello.txt /hello.txt
ADD *.txt /root/
```

`ADD <복사할 파일 경로> <이미지에서 파일이 위치할 경로>` 형식입니다.

- `<복사할 파일 경로>`는 컨텍스트 아래를 기준으로 하며 컨텍스트 바깥의 파일, 디렉터리나 절대 경로는 사용할 수 없습니다.
- `<복사할 파일 경로>`는 파일뿐만 아니라 디렉터리도 설정할 수 있으며, 디렉터리를 지정하면 디렉터리의 모든 파일을 복사합니다.
- 와일드카드를 사용하여 특정 파일만 복사할 수 있습니다.
- `<복사할 파일 경로>`에 인터넷에 있는 파일의 URL을 설정할 수 있습니다.
- 로컬에 있는 압축 파일(tar.gz, tar.bz2, tar.xz)은 압축을 해제하고 tar를 풀어서 추가됩니다. 단, 인터넷에 있는 파일 URL은 압축만 해제한 뒤 tar 파일이 그대로 추가됩니다.
- `<이미지에서 파일이 위치할 경로>`는 항상 절대 경로로 설정해야 합니다. 그리고 마지막이 /로 끝나면 디렉터리가 생성되고 파일은 그 아래에 복사됩니다.
- `ADD ./ /hello`와 같이 현재 디렉터리를 추가할 때 **.dockerignore** 파일에 설정한 파일과 디렉터리는 제외됩니다.

7-10.COPY

COPY는 파일을 이미지에 추가합니다. ADD와는 달리 COPY는 압축 파일을 추가할 때 압축을 해제하지 않고, 파일 URL도 사용할 수 없습니다.

Dockerfile

```
COPY hello-entrypoint.sh /entrypoint.sh
COPY hello-dir /hello-dir
COPY zlib-1.2.8.tar.gz /zlib-1.2.8.tar.gz
COPY *.txt /root/
```

`COPY <복사할 파일 경로> <이미지에서 파일이 위치할 경로>` 형식입니다.

- `<복사할 파일 경로>`는 컨텍스트 아래를 기준으로 하며 컨텍스트 바깥의 파일, 디렉터리나, 절대 경로는 사용할 수 없습니다.
- `<복사할 파일 경로>`는 파일뿐만 아니라 디렉터리도 설정할 수 있으며, 디렉터리를 지정하면 디렉터리의 모든 파일을 복사합니다.
- 와일드카드를 사용하여 특정 파일만 복사할 수 있습니다.
- `<복사할 파일 경로>`에 인터넷에 있는 파일의 URL은 사용할 수 없습니다.
- 압축 파일은 압축을 해제하지 않고 그대로 복사됩니다.
- 마지막이 /로 끝나면 디렉터리가 생성되고 파일은 그 아래에 복사됩니다.
- `COPY ./ /hello`와 같이 현재 디렉터리를 추가할 때 **.dockerignore** 파일에 설정한 파일과 디렉터리는 제외됩니다.

7-11.VOLUME

VOLUME은 디렉터리의 내용을 컨테이너에 저장하지 않고 호스트에 저장하도록 설정합니다.

Dockerfile

```
VOLUME /data
VOLUME ["/data", "/var/log/hello"]
```

`VOLUME <컨테이너 디렉터리>` 또는 `VOLUME ["컨테이너 디렉터리 1", "컨테이너 디렉터리2"]` 형식입니다.
/data처럼 바로 경로를 설정할 수도 있고, **["/data", "/var/log/hello"]**처럼 배열 형태로 설정할 수도 있습니다.
단, VOLUME으로는 호스트의 특정 디렉터리와 연결할 수는 없습니다.

데이터 볼륨을 호스트의 특정 디렉터리와 연결하려면 `docker run` 명령에서 `-v` 옵션을 사용해야 합니다.

```
$ sudo docker run -v /root/data:/data example
```

옵션은 `-v <호스트 디렉터리>:<컨테이너 디렉터리>` 형식입니다.

7-12.USER

USER는 명령을 실행할 사용자 계정을 설정합니다. RUN, CMD, ENTRYPOINT에 적용됩니다.

```
USER nobody
```

`USER <계정 사용자명>` 형식입니다.
USER 뒤에 오는 모든 RUN, CMD, ENTRYPOINT에 적용되며,
중간에 다른 사용자를 설정하여 사용자를 바꿀 수 있습니다.

Dockerfile

```
USER nobody
RUN touch /tmp/hello.txt

USER root
RUN touch /hello.txt
ENTRYPOINT /hello-entrypoint.sh
```

7-13.WORKDIR

WORKDIR은 RUN, CMD, ENTRYPOINT의 명령이 실행될 디렉터리를 설정합니다

Dockerfile

```
WORKDIR /var/www
```

`WORKDIR <경로>` 형식입니다.
WORKDIR 뒤에 오는 모든 RUN, CMD ENTRYPOINT에 적용되며 중간에 다른 디렉터리를 설정하여 실행 디렉터리를 바꿀 수 있습니다.

Dockerfile

```
WORKDIR /root
RUN touch hello.txt

WORKDIR /tmp
RUN touch hello.txt
```

WORKDIR은 절대 경로 대신 상대 경로도 사용할 수 있습니다. 상대 경로를 사용하면 먼저 설정한 WORKDIR의 경로를 기준으로 디렉터리를 변경합니다. 최초 기준은 /입니다.

```
WORKDIR var
WORKDIR www

RUN touch hello.txt
```

/var/www/hello.txt

7-14.ONBUILD

ONBUILD는 생성한 이미지를 기반으로 다른 이미지가 생성될 때 명령을 실행(trigger)합니다.
최초에 ONBUILD를 사용한 상태에서는 아무 명령도 실행하지 않습니다.
다음 번에 이미지가 FROM으로 사용될 때 실행할 명령을 예약하는 기능이라 할 수 있습니다.

Dockerfile

```
ONBUILD RUN touch /hello.txt
ONBUILD ADD world.txt /world.txt
```

`ONBUILD <Dockerfile 명령> <Dockerfile 명령의 매개 변수>` 형식입니다.
FROM, MAINTAINER, ONBUILD를 제외한 모든 Dockerfile 명령을 사용할 수 있습니다.

7-14.ONBUILD

Dockerfile

```
FROM ubuntu:latest
ONBUILD RUN touch /hello.txt
```

```
$ sudo docker build --tag example .
$ sudo docker run -i -t example /bin/bash
root@891ccb6749e9:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

ONBUILD로 설정했기 때문에 **example** 이미지에는 `/hello.txt` 파일이 생성되지 않았습니다.

이제 FROM을 사용하여 **example** 이미지를 기반으로 새 이미지를 생성합니다.

Dockerfile

```
FROM example
```

```
$ sudo docker build --tag example2 .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM example
# Executing 1 build triggers
Step onbuild-0 : RUN touch /hello.txt
 ---> Running in 7e77e38db77c
 ---> 967feb106636
 ---> 967feb106636
Removing intermediate container 7e77e38db77c
Successfully built 967feb106636
$ sudo docker run -i -t example2 /bin/bash
root@874d3e1fdd6f:/# ls
bin  boot  dev  etc  hello.txt  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

ONBUILD는 바로 아래 자식 이미지를 생성할 때만 적용되고,
손자 이미지에는 적용되지 않습니다. 즉 ONBUILD 설정은
상속되지 않습니다.

7-14.ONBUILD

`docker inspect` 명령으로 이미지의
ONBUILD 설정을 확인할 수 있습니다.

```
$ sudo docker inspect -f "{{ .ContainerConfig.OnBuild }}" example
[RUN touch /hello.txt]
```