02_Dockerfile_command

FROM

- 기본적으로 Dockerfile은 어떤 이미지를 기반으로 시작하는지에 대해서 정의를 시작한다.
- FROM 뒤에 이미지의 이름이 오고 그 뒤에 이미지의 label이 붙여진다.

```
#FROM <image-name>:<label>
FROM debian:buster
```

RUN

- Dockerfile 이미지 내부에서 실행되어야 하는 명령어를 작성한다.
- 명령어를 실행한 후 만들어지는 이미지를 레이어 별로 저장을 하여 dockerfile에서 수정을 해도 이전 레이어를 활용하여 빠르게 빌드가 가능하다.
- RUN 뒤에 실행이 필요한 명령어를 작성하여 사용한다.

```
#RUN <command>
RUN apt-get -y update
```

- 실행되어야 하는 경로를 직접 적어주는 방법과 인자에 대해 확실히 배열의 형태로 정의해주는 방법도 존재한다.

```
RUN /bin/bash -c 'echo "Hello World!"'
RUN ["/bin/bash", "-c" "echo 'Hello World!'"]
```

COPY

- Docker image를 만들면서 내부에 sourcefile을 복사하여 이동하는 경우 사용한다.

```
#COPY <src> <dst>
COPY init.sh /
```

ADD

- COPY와 달리 파일의 src 부분에 url을 적으면 자동으로 파일을 다운받아 복사를 한다.
- 압축파일의 경우 압축을 해제해서 전달한다.

```
# ADD <src(url)> <dest>
ADD <https://www.docker.com/sites/default/files/d8/2019-07/Moby-logo.png> /
```

ENV

- Dockerfile 내부 이미지에서 설정되는 환경변수에 대해 설정한다.

```
# ENV name=value
ENV DEV_ENV=true
```

WORKDIR

- Dockerfile에서 작업하는 경로에 대해 정의할 때 사용한다.

```
# WORKDIR <dir>
WORKDIR /root
```

USER

- 작업하는 사용자에 대해서 정의할 때 사용한다.
- 사용되는 user에 대해서는 미리 RUN을 이용하여 user와 group을 추가하거나 이미지에 등록된 user여만 작동한다.

```
# USER <user>[:<group>] | <UID>[:<GID>]
USER 0:0
```

SHELL

- RUN을 할 때 사용이 되는 기본 shell에 대해 정의한다.

```
# SHELL [<executable>, <parameters>]
SHELL ["/bin/bash", "-"]
```

ARG

- Dockerfile은 image를 만들어주는 script 이므로 동적으로 할당이 가능한 ARG가 사용 가능을 하다.
- ARG명령어로 정의한 환경변수는 동일한 ARG변수를 재정의 한다.

```
# ARG <name>[=<default value>]
ARG user=root
```

- Build 시점에 쓰는 변수를 동적으로 재정의할 수 있다.

```
$ docker build --build-arg user=hkwon .
```

- 환경변수를 ARG를 이용하여 재정의하여 사용할 수 있다.

```
ARG TEST
ENV TEST=${TEST:-hello_world}
RUN echo $TEST
```

EXPOSE

- Container 내부에서 network port를 수신 대기 중인 상태를 docker에게 미리 알려줄 수 있다.
- 기본적으로 protocol에 대해 정의를 하지않으면 tcp로 작동을 한다.
  - 정의를 하였다고 해도 host와 port가 연결이 되는 것이 아니므로 run time 시 port binding 작업이 수행되어야 한다.

```
# EXPOSE <port> | <port><protocol>
EXPOSE 80
EXPOSE 443/tcps
```

CMD

- Dockerfile에서 container가 시작되었을 때 script 혹은 명령을 실행.
- 만약 docker run 으로 실행될 때 사용자가 직접 실행할 명령어를 입력하면 CMD의 명령어는 실행되지 않는다.

- Container를 실행할 때 인자값을 주면 Dockerfile에 지정된 CMD 값을 대신해 지정한 인자값으로 변경하여 실행.
    - docker run test:cmd echo hi 와 같이 마지막에 실행 인자를 전달 하게 되면 echo hi 가 실행이 된다.
    - inspect 명령어로 확인을 해보면 CMD 부분의 정보가 새로 정의한 내용으로 변경이 되어 있는 것을 볼 수 가 있다.

```
CMD ["echo", "hello"]
```

ENTRYPOINT

- Dockerfile에서 container가 시작되었을 때 script 혹은 명령을 실행.
- 해당 컨테이너가 수행될 때 반드시 ENTRYPOINT에 속한 명령을 수행하도록 지정된다.
    - docker run test:entrypoint echo world 로 실행을 하게 되면 hello echo world 가 나오는 것을 볼 수 있다.
- Dockerfile에서 한 번만 정의 가능하다.

```
ENTRYPOINT ["echo", "hello"]
```

RUN vs CMD vs ENTRYPOINT

```
1) RUN
- 새로운 layer를 생성하거나, 생성된 layer 위에서 command를 실행.
- (package를 설치하는 명령어를 주로 사용)

2) CMD
- docker가 실행될때 실행되는 명령어 정의
- 만약 docker run 으로 실행될 때 사용자가 직접 실행할 명령어를 입력시
  CMD의 명령어는 실행되지 않는다.
- ex) docker run <image_name> echo world
  "world" 가 출력됨.
  Dockerfile에 있는 CMD는 실행되지 않음

3) ENTRYPOINT
- docker run으로 생성하거나,
  docker start로 중지된 container를 시작할 때 실행되는 명령어
  (CMD와 동일한 역할)
- Dockerfile 내에서 1번만 정의 가능함.
- CMD와 다른 점으로
  docker run으로 실행시 command를 입력하면,
```

```
  ENTRYPOINT의 파라미터로 인식한다.
- ex) docker run <image_name> echo world
  "echo world"를 파라미터로 인식함

[출처] [docker] RUN vs CMD vs ENTRYPOINT | 작성자 freepsw
```

VOLUME

- Docker에서는 기본적으로 데이터를 이미지 레이어에 저장을 한다.
  - 새로운 것을 올릴 때마다 초기 설정된 데이터로 초기화 되는 휘발성이 특징.
- log file이나 db와 같은 데이터는 휘발성이 아닌 비 휘발성으로 관리 되어야 한다.
- 특정 폴더를 기본 volume으로 저장 하지 않고 따로 저장하여 비 휘발성 데이터를 보존한다.

```
# VOLUME [<dir>]
VOLUME ["/data"]
```

ONBUILD

- 만들어진 image를 가지고 FROM 할 때 미리 수행 되어야 하는 작업이 있는 경우 사용한다.
  - 예를 들어 python build에 관련된 이미지를 FROM 으로 사용하여 build를 하기 전 src 파일들을 가져와 사용할 수 있다.

```
# ONBUILD <INSTRUCTION>
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
```

STOPSIGNAL

- 기본적으로 Docker Container를 종료하게 되면 내부에서 작동하는 프로세스에게 SIGTERM을 보내게 되는데 이 때 보내지는 SIGNAL에 대해서 정의한다.

```
# STOPSIGNAL signal
STOPSIGNAL SIGKILL
```

HEALTHCHECK

- DockerContainer가 올라갔을 때 실제로 프로세스가 정상적으로 동작 하는 지에 대해서 프로세스 상태를 체크할 수 있다.
- Option
  - `interval=DURATION` (default: `30s`)
  - `timeout=DURATION` (default: `30s`)
  - `start-period=DURATION` (default: `0s`)
  - `retries=N` (default: `3`)

```
# HEALTHCHECK [OPTIONS] CMD command
HEALTHCHECK --interval=5m --timeout=3s \\
  CMD curl -f <http://localhost/> || exit 1
```

LABEL

- 생성된 image에 대해 여러가지 정보를 담을 때 사용.

```
# LABEL <name>=<value>
LABEL maintainer="hkwon <hkwon.student.42seoul.kr>"
```

출처

- https://jaeseokim.github.io/Etc/docker기초부터-wordpress서비스-직접구현까지/
- https://yeosong1.github.io/ft_server도커파일문법
- https://nirsa.tistory.com/68