Dockerfile

IaC (Infrastructure as a Code)

- 직역 : '코드로 작성한 인프라'
- 수작업에 의한 프로세스가 아닌 코드에 의해 관리 및 배포되는 인프라
  동일한 환경을 배포하도록 보장
- 버전 관리가 용이하다.

Dockerfile

- 사용자가 원하는 이미지를 작성하기 위하여 사용하는 코드\
- 주요 항목
- 기초 이미지
- 컨테이너 볼륨
- 이미지에 추가할 파일
- 컨테이너 작성자 정보
- 빌드 단계에서 실행할 명령
- 빌드 후 실행될 명령
- 컨테이너 구동 시 실행할 명령
- 컨테이너 구동 시 환경변수
- 컨테이너에서 노출할 포트
- 컨테이너 구동 시 작업 디렉토리 위치

Dockerfile 작성방법

```
<명령어> <인수>
# 명령어는 대문자로
# 주석 사용가능
```

- Dockerfile을 사용한 이미지 빌드
  - -t, --tag
  - -f, --file : Dockerfile의 이름이 Dockerfile이 아닐 경우 이름 지정
  - --build-arg : 빌드 시 사용할 변수 지정

```
$ docker build [OPTION] <Dockerfile_PATH>
```

FROM

- 기초 이미지 지정

```
FROM <이미지명>
FROM <이미지명>:<태그>
FROM <이미지명>@<DIGEST>
```

- 태그명 생략 시 베이스 이미지의 최신 버전(latest) 적용
- 다이제스트(Digest) : Docker Hub 업로드 시 자동으로 부여되는 식별자
- 다이제스트 확인 : docker image ls --digests [이미지명]

```
# 베이스 이미지 설정
FROM centos:7
```

RUN

- 베이스 이미지로부터 빌드하기 위하여 생성한 컨테이너에서 명령 실행

```
RUN [COMMAND]
```

- 여러 항목 사용 시 순차적으로 실행 (빌드 시 step으로 표시됨)
- 실행 단계 마다 파일시스템 레이어 생성 (한 명령이 아닌 한 줄 마다 생성)
- 파일시스템에 변경이 가해지지 않을 경우는 파일시스템 미생성
- "\" 기호를 사용하여 Dockerfile 명령 내에서 줄바꿈 가능

ENTRYPOINT

- 빌드 한 이미지를 바탕으로 생성된 컨테이너 내에서 실행되는 명령

```
ENTRYPOINT [실행할 명령]
```

CMD

- 빌드 한 이미지를 바탕으로 생성된 컨테이너 내에서 실행되는 명령

```
CMD [실행할 명령]
```

- 명령 기술 방식 (RUN, CMD, ENTRYPOINT)
- Shell 형식 : 지정한 명령을 Shell 내에서 실행
- Exec 형식 : 명령을 직접 실행

- Shell 형식
- 실행할 명령을 그대로 입력
- RUN [실행할 명령, 옵션, 인수]

```
# Nginx 설치
RUN apt-get -y install nginx
```

- Exec 형식
- 실행할 명령을 JSON 포맷으로 지정

```
RUN ["명령어","옵션","인수"]
# Nginx 설치
RUN ["apt-get", "-y", "install", "nginx"]
```

참고

ENTRYPOINT와 CMD는 둘 다 컨테이너 내에서 실행될 명령을 지정하는데 사용됩니다. 그러나 둘의 차이점은 실행되는 방식입니다.

- ENTRYPOINT는 실행될 명령 자체를 지정합니다.
- CMD는 ENTRYPOINT의 인자로 사용될 수 있습니다. 예를 들어,

  ```
  ENTRYPOINT touch
  CMD /tmp/hello
  ```

  - 위와 같은 경우, 실제 실행되는 명령은 `touch /tmp/hello`입니다.

- 만약 CMD가 단독으로 사용된다면, 그 명령이 실행된다.

  ```
  CMD touch /tmp/hello
  ```

  - 위와 같은 경우, 컨테이너가 시작될 때 `/tmp/hello` 파일이 생성됩니다.

- ENTRYPOINT와 CMD는 각각 `--entrypoint`와 `docker run` 명령어의 인자로 재지정될 수 있습니다.
- ENTRYPOINT와 CMD는 Dockerfile 내에서 실행되는 명령의 기본값을 설정하는 데 사용됩니다. 그러나 이러한 명령은 `docker run` 명령어를 통해 변경될 수 있습니다. `docker run` 명령어를 사용하여 ENTRYPOINT와 CMD를 재지정할 수 있으며, 이를 통해 Docker 컨테이너에서 실행할 명령을 지정할 수 있습니다.

CMD 항목의 Shell 형식과 Exec 형식의 차이점

- 실행되는 방식에 차이가 존재한다.
- Shell 형식은 실행할 명령을 Shell 내에서 실행
  - 실행할 명령을 그대로 입력하면 된다.
  - 여러 항목을 사용할 경우, RUN 명령어와 마찬가지로 순차적으로 실행됨
- Exec 형식은 명령을 직접 실행합니다. 실행할 명령을 JSON 포맷으로 지정된다.

```
RUN ["apt-get", "-y", "install", "nginx"]
#  'apt-get'을 직접 실행
```

장단점

- Shell 형식은 간단하고 쉽게 사용할 수 있으며, 쉘 환경 변수와 명령어 치환 기능을 지원합니다. 하지만 실행 속도가 느리고, 쉘 스크립트에서 발생하는 오류가 Dockerfile 빌드에서 발견되지 않을 수 있습니다.
- Exec 형식은 빌드 속도가 빠르고, 명령의 실행 결과를 정확하게 확인할 수 있습니다. 또한, JSON 배열 형태로 인자를 전달하기 때문에, 명령어와 인자 사이의 오류를 발견할 수 있습니다. 하지만 쉘 환경 변수와 명령어 치환 기능을 사용하려면, 쉘을 명시적으로 지정해야 하기 때문에 조금 더 복잡합니다.
- **따라서, 실행 속도와 실행 결과의 정확성을 중요시하는 경우에는 Exec 형식을 사용하는 것이 좋습니다. 반면에, 동적으로 명령어를 결정해야 하거나, 쉘 스크립트에서 환경 변수나 명령어 치환 기능을 사용해야 하는 경우에는 Shell 형식을 사용하는 것이 좋습니다.**

ONBUILD

- 이미지를 다음 빌드에서 베이스 이미지로 사용시 실행하도록 설정

```
ONBUILD [실행하고 싶은 명령]
```

- 사용예 : 팀 단위 개발
  - 최종단계의 빌드된 이미지가 아닌 데이터 추가가 필요한 이미지
  - 실행환경 Dockerfile 작성 담당자
  - OS/미들웨어 설치 및 설정
  - 라이브러리 검증 및 도입
  - 베이스 Dockerfile 작성
  - Git을 통한 Dockerfile 관리
  - 웹 컨텐츠 개발자(들)
  - 각자 웹 컨텐츠 소스코드 작성
  - 베이스 Dockerfile을 바탕으로 각자 테스트 수행

STOPSIGNAL

- 컨테이너 중지 명령을 내릴 때, 실행프로세스에 전달할 Signal 유형 지정

```
STOPSIGNAL [Signal종류]
```

- Signal 종류 : kill -l, man -s7 signal
- 주요 시그널
  - SIGINT(2) : Ctrl + C
  - SIGKILL(9) : 강제종료

- SIGTERM(15) : 정상종료
- SIGTSTP(20) : Ctrl + Z

HEALTHCHECK

- 컨테이너 내 프로세스의 정상동작여부 확인

```
HEALTHCHECK [옵션] CMD [실행할 명령]
```

- 옵션
  - --interval : 체크 간격 지정 (기본값 30초)
  - --timeout : 체크 타임아웃 (기본값 30초)
  - --retries : 타임아웃 회수 (기본값 3회)

ENV

- 실행된 컨테이너에서 사용할 환경변수 지정

```
ENV [key] [value]
```

- 한번에 하나의 환경변수 지정

```
ENV [key]=[value]
```

- 한번에 여러 개의 환경변수 지정 가능 (개행 \)

WORKDIR

- 실행된 컨테이너에서 프로세스를 실행할 위치 지정 (기본값 /)

```
WORKDIR [경로]
```

- 상대경로 및 절대경로 사용 가능
- 반복해서 사용할 경우 각 단계에서 위치 이동
- ENV로 설정한 환경변수 사용 가능
- 작업 위치에 영향을 받는 항목: RUN, CMD, ENTRYPOINT, COPY, ADD

USER

- 컨테이너 내에서 작업을 실행할 사용자 지정
- USER [사용자명/UID]
- 사용자가 존재하여야 함

```
RUN : useradd / adduser
```

LABEL

- 이미지에 대한 부가정보를 기술

```
LABEL [항목이름]=[값]
```

- 기능에는 영향을 주지 않음
- 이미지 정보에서 확인 가능 (docker image inspect)

EXPOSE

- 컨테이너에서 네트워크로 노출할 포트 지정

```
EXPOSE [포트번호]
```

- 포트의 노출정보를 표시하는 용도
- 실제 사용할 포트를 지정하는 부분은 내부 어플리케이션임에 주의

- 어플리케이션의 정보에 맞게 포트 노출정보를 설정

ARG

- Dockerfile 내에서만 유효한 변수 지정

```
ARG [KEY]=[VALUE]
```

- 빌드 시 --build-arg 옵션을 사용하여 변경 가능

SHELL

- Shell 방식으로 명령 실행(RUN, CMD, ENTRYPOINT)시 사용할 기본 쉘

```
SHELL ["SHELL_PATH", "PARAMETER"]
```

- 기본값: /bin/sh -c

ADD

- 이미지에 호스트 또는 네트워크 위치의 파일을 추가

```
ADD <ORIGIN_PATH> <IMAGE_PATH>
ADD ["<ORIGIN_PATH>" "<IMAGE_PATH>"]
```

- 경로 지정 시 절대경로 및 상대경로 사용 가능
- 아카이브 파일(tar, gzip, bzip2) 자동 압축해제
- 네트워크 위치 파일 추가 가능
- 파일 권한: 600 (RW-------)
- 수정시간 정보: HTTP Last-Modified 헤더 항목이 있을 경우 파일 수정시간 설정

- https 미지원
- 원격 아카이브 파일은 압축해제 미지원

COPY

- 이미지에 호스트의 파일을 추가

```
COPY <ORIGIN_PATH> <IMAGE_PATH>
COPY ["<ORIGIN_PATH>" "<IMAGE_PATH>"]
```

VOLUME

- 컨테이너 실행 시 볼륨 마운트 위치 지정

```
VOLUME </MOUNT_POINT>
VOLUME ["</MOUNT_POINT>", "</MOUNT_POINT>", "</MOUNT_POINT>"]
```

- 임의의 이름의 볼륨을 생성하여 지정한 마운트 포인트에 연결
- 특정 볼륨 사용 또는 다른 컨테이너의 볼륨 사용이 필요할 경우 런타임 옵션 사용
    - --volume, --volumes-from