[Dockerfile 의 작성 방법]

Docker가 대중화되면서 많은 프로젝트들이 개발 환경을 컨테이너화(containerization)시키고 있다. 이러한 프로젝트의 최상위 디렉터리에서는 Dockerfile이 위치하게 되며, Dockerfile의 작성 방법을 이해하는 것은 그 프로젝트의 개발 환경이 어떻게 구성되는지 이해하는 첫걸음이다.

Dockerfile은 Docker 이미지(image)가 어떤 단계를 거쳐 빌드(build)되어야 하는지를 담고 있는 텍스트 파일이다. Docker는 Dockerfile에 나열된 명령문을 차례대로 수행하여 이미지를 생성한다.
다음은 자주 쓰이는 명령어 위주로 Dockerfile을 작성하는 방법에 대해서 정리한 내용이다.

- # 주석(Comment)
명령어(INSTRUCTION) 인자(arguments)
각 명령문은 명령어로 시작하고 여러 개의 인자가 정의될 수 있으며, 해당 명령문에 대한 주석도 달 수 있다. 인자와 구분이 쉽도록 명령어는 모두 영문 대문자로 써주는 것이 관례이다.

- FROM 명령문

FROM <이미지>
FROM <이미지>:<태그>

하나의 Docker 이미지는 base 이미지부터 시작해서 기존 이미지 위에 새로운 이미지를 중첩해서 여러 단계의 이미지 층(layer)을 쌓아가며 만들어진다.

FROM 명령문은 이 base 이미지를 지정하기 위해서 사용되는데, 보통 Dockerfile 내에서 최상단에 위치한다. base 이미지는 일반적으로 Docker Hub와 같은 Docker repository에 올려놓은 잘 알려진 공개 이미지인 경우가 많다.

Ubuntu 최신 버전을 base 이미지로 사용

```
FROM ubuntu:latest
```

NodeJS 12를 base 이미지로 사용

FROM node:12

Python 3.8 (alpine 리눅스 기반)을 base 이미지로 사용

FROM python:3.8-alpine

아파치 웹 서버를 base 이미지로 사용

FROM httpd

최신 버전의 MySQL 을 base 이미지로 사용

FROM mysql:latest

openjdk 17 버전을 base 이미지로 사용

FROM openjdk:17-jdk

- WORKDIR 명령문

WORKDIR 명령문은 쉘(shell)의 cd 명령문처럼 컨테이너 상에서 작업 디렉토리로 전환을 위해서 사용된다. WORKDIR 명령문으로 작업 디렉터리를 전환하면 그 이후에 등장하는 모든 RUN, CMD, ENTRYPOINT, COPY, ADD 명령문은 해당 디렉터리를 기준으로 실행된다. 즉, WORKDIR은 도커 이미지의 어디에서 작업을 할지를 의미한다.

WORKDIR <이동할 경로>

/usr/app으로 작업 디렉터리 전환

WORKDIR /usr/app

- RUN 명령문

RUN ["<커맨드>", "<파라미터1>", "<파라미터2>"]

RUN <전체 커맨드>

RUN 명령문은 마치 쉘(shell)에서 커맨드를 실행하는 것처럼 이미지 빌드 과정에서 필요한 커맨드를 실행하기 위해서 사용된다. 쉘(shell)을 통해 거의 못하는 작업이 없는 것처럼 RUN 명령문으로 할 수 있는 작업은 제한 없지만 보통 이미지 안에 특정 소트프웨어를 설치하기 위해서 많이 사용된다.

curl 도구 설치
RUN apk add curl
npm 패키지 설치
RUN npm install --silent
pip 패키지 설치
RUN pip install -r requirements.txt

- ENTRYPOINT 명령문

ENTRYPOINT ["<커맨드>", "<파라미터1>", "<파라미터2>"]
ENTRYPOINT <전체 커맨드>

ENTRYPOINT 명령문은 이미지를 컨테이너로 띄울 때 항상 실행되어야 하는 커맨드를 지정할 때 사용한다. ENTRYPOINT 명령문은 Docker 이미지를 마치 하나의 실행 파일처럼 사용할 때 유용한다. 컨테이너가 기동 될때 ENTRYPOINT 명령문으로 지정된 커맨드가 실행되고, 이 커맨드로 실행된 프로세스가 죽을 때, 컨테이너도 따라서 종료되기 때문이다.

npm start 스크립트 실행
    ENTRYPOINT ["npm", "start"]
nginx 서버 기동
    ENTRYPOINT ["nginx", "-g", "daemon off;"]

- CMD 명령문

CMD ["<커맨드>","<파라미터1>","<파라미터2>"]
CMD ["<파라미터1>","<파라미터2>"]
CMD <전체 커맨드>

CMD 명령문은 해당 이미지를 컨테이너로 띄울 때 디폴트로 실행할 커맨드나, ENTRYPOINT 명령문으로 지정된 커맨드에 디폴트로 넘길 파라미터를 지정할 때 사용한다.

CMD 명령문은 많은 경우, ENTRYPOINT 명령문과 함께 사용하게 되는데, ENTRYPOINT

명령문으로는 커맨드를 지정하고, CMD 명령문으로 디폴트 파리미터를 지정해주면 매우 유연하게 이미지를 실행할 수 있게 된다.

예를 들어, node 커맨드로 디폴트로는 index.js를 실행하되, docker run 커맨드에 인자가 있는 경우, 해당 인자를 실행하고 싶은 경우, 다음과 같이 Dockerfile을 작성한다.

ENTRYPOINT ["node"]
CMD ["index.js"]
그러면 다음과 같이 docker run 커맨드의 인자 유무에 따라 node 커맨드로 다른 파일이 실행되게 할 수 있다.
node index.js 실행
$ docker run test
node main.js 실행
$ docker run test main.js
CMD 명령문과 RUN 명령문이 기능이 비슷해 보이는데 RUN 명령문은 이미지 빌드시 항상 실행되며, 한 Dockerfile에 여러 개의 RUN 명령문을 선언할 수 있다. 반면에, CMD 명령문은 이미지를 컨테이너로 기동시킬 때 딱 한 번 실행 기회를 가지게 된다.

- EXPOSE 명령문

EXPOSE <포트>
EXPOSE <포트>/<프로토콜>

EXPOSE 명령문은 네트워크 상에서 컨테이너로 들어오는 트래픽(traffic)을 리스닝(listening)하는 포트와 프로토콜를 지정하기 위해서 사용된다. 프로토콜은 TCP와 UDP 중 선택할 수 있는데 지정하지 않으면 TCP가 기본값으로 사용된다.

여기서 주의할 점은 EXPOSE 명령문으로 지정된 포트는 해당 컨테이너의 내부에서만 유효하며, 호스트(host) 컴퓨터에서는 이 포트를 바로 접근을 할 수 없다. 호스트 컴퓨터로부터 해당 포트로의 접근을 허용하려면, docker run 커맨드에서 -p 옵션을 통해 호스트 컴퓨터의 특정 포트를 포워딩(forwarding)시켜주어야 한다.

80/TCP 포트로 리스닝

EXPOSE 80

9999/UDP 포트로 리스닝

EXPOSE 9999/udp

- COPY/ADD 명령문

COPY <src>... <dest>

COPY ["<src>" ... "<dest>"]

COPY 명령문은 호스트 컴퓨터에 있는 디렉터리나 파일을 Docker 이미지의 파일 시스템으로 복사하기 위해서 사용된다. 절대 경로와 상대 경로를 모두 지원하며, 상대 경로를 사용할 때는 이 전에 등장하는 WORKDIR 명령문으로 작업 디렉터리를 어디로 변경했는지 고려해야 한다.

package.json 파일만 복사

COPY package.json package.json

이미지 빌드를 수행한 디렉터리의 모든 파일을 컨테이너의 app/ 디렉터리로 복사

WORKDIR app/

COPY . .

ADD 명령문은 좀 더 파워풀한 COPY 명령문이라고 할 수 있다. ADD 명령문은 일반 파일뿐만 아니라 압축 파일이나 네트워크 상의 파일도 사용할 수 있다. 이렇게 특수한 파일을 다루는 게 아니라면 COPY 명령문을 사용하는 것이 권장된다.

- ENV 명령문

ENV <키> <값>

ENV <키>=<값>

ENV 명령문은 환경 변수를 설정하기 위해서 사용한다. ENV 명령문으로 설정된 환경 변수는 이미지 빌드 시에도 사용되고, 해당 컨테이너 안에서 실행되는 애플리케이션에서도 접근할 수 있다.

NODE_ENV 환경 변수를 production으로 설정

ENV NODE_ENV production

자바 버전을 11로 설정

ENV JAVA_VERSION 11

- ARG 명령문

ARG <이름>

ARG <이름>=<기본값>

ARG 명령문은 docker build 커맨드로 이미지 빌드 시, --build-arg 옵션을 통해 넘길 수 있는 인자를 정의하기 위해 사용한다.

예를 들어, Dockerfile에 다음과 같이 ARG 명령문으로 port를 인자로 선언해주면,

ARG port

다음과 같이 docker build 커맨드에 --build-arg 옵션에 port 값을 넘길 수가 있다.

$ docker build --build-arg port=8080 .

인자의 디폴트값을 지정해주면, --build-arg 옵션으로 해당 인자가 넘어오지 않았을 때 사용된다.

ARG port=8080

설정된 인자 값은 다음과 같이 ${인자명} 형태로 읽어서 사용할 수 있다.

CMD start.sh -h 127.0.0.1 -p ${port}

ENV와 달리 ARG로 설정한 값은 이미지가 빌드되는 동안에만 유효하다.

- .dockerignore 파일

명령문은 아니지만 .dockerignore 파일도 알아두면 Dockerfile을 작성할 때 유용한다.

Docker 이미지를 빌드할 때 제외 시키고 싶은 파일이 있다면, .dockerignore 파일에 추가한다.

예를 들어, .git 디렉터리와 마크다운(markdown) 파일을 모두 제외 시키고 싶다면 다음과 같이 .dockerignore 파일을 작성해주면 된다.

.dockerignore 파일

.git

*.md

이렇게 설정을 해주면 Docker는 프로젝트 최상위 디렉터리에 위치하고 있는 markdown 파일들을 무시하게 되므로, RUN과 CMD, COPY와 같은 명령문이 해당 파일을 사용할 수 없게 된다.

FROM	base 이미지 설정
WORKDIR	작업 디렉터리 설정
RUN	이미지 빌드 시 커맨드 실행
ENTRYPOINT	이미지를 가지고 컨테이너를 기동할 때 실행되어야 하는 커맨드 설정
CMD	ENTRYPOINT 에 지정된 명령 실행시 전달될 파라미터 설정
EXPOSE	컨테이너가 리스닝할 포트 및 프로토콜 설정
COPY/ADD	이미지의 파일 시스템으로 파일 또는 디렉터리 복사
ENV	환경 변수 설정

(*) Docker 파일 작성시 참고할 사이트

https://docs.docker.com/reference/

https://github.com/komljen/dockerfile-examples

https://github.com/topics/dockerfile-examples?o=desc&s=updated