[이전 글](https://javacan.tistory.com/529)에서 도커 이미지에 대해 알아봤는데 이어서 Dockerfile을 도커 이미지 파일을 생성하는 방법을 살펴보자.

**초 간단 Dockerfile**

다음은 매우 간단한 Dockerfile 예이다.

FROM alpine:3.10

ENTRYPOINT ["echo", "hello"]

FROM은 새로운 이미지를 생성할 때 기반으로 사용할 이미지를 지정한다. 위 코드는 alpine:3.10 이미지를 기반 이미지로 사용한다.

ENTRYPOINT는 컨테이너를 시작할 때 실행할 명령어를 입력한다. 위 코드는 'echo hello'를 실행 명령어로 사용한다. ENTRYPOINT는 두 방식으로 입력할 수 있는데 이에 대한 내용은 뒤에서 다시 설명한다.

Dockerfile을 작성했으면 docker build 명령어로 이미지를 생성할 수 있다.

docker build --tag echoalpine:1.0 .

--tag(또는 -t) 옵션은 새로 생성할 이미지 이름을 지정한다. 여기서는 리포지토리 이름으로 echoalpine을 사용하고 태그로 1.0을 사용했다. 마지막에 점(.)은 Dockerfile의 위치를 경로를 지정한다. 파일 이름이 Dockerfile이 아닌 경우 --file(또는 -f) 옵션을 사용해서 파일 이름을 지정한다.

Dockerfile이 위치한 디렉토리에서 위 명령어를 실행하면 다음과 같이 새로운 도커 이미지를 생성한다. docker images 명령어로 새로 생긴 이미지를 확인할 수 있다.

vagrant@ubuntu-bionic:~/sample$ docker build --tag echoalpine:1.0 .

Sending build context to Docker daemon 2.048kB

Step 1/2 : FROM alpine:3.10

---> 961769676411

Step 2/2 : ENTRYPOINT ["echo", "hello"]

---> Running in 943587678aaf

Removing intermediate container 943587678aaf

---> 109b1eea8279

Successfully built 109b1eea8279

Successfully tagged echoalpine:1.0

새로 생성한 이미지를 사용해서 컨테이너를 생성하고 실행해보자. hello가 콘솔에 출력되고 컨테이너가 종료되는 것을 알 수 있다.

$ docker run --rm echoalpine:1.0

hello

컨테이너는 ENTRYPOINT로 지정한 echo hello 명령어를 실행한다. echo는 콘솔에 문자열을 출력하고 종료되므로 컨테이너도 함께 종료된다.

**간단한 예제 구성**

앞의 Dockerfile은 너무 간단했다. 조금 더 현실적인 Dockerfile을 만들어보자. 이를 위해 사용할 파일은 다음고 같다.

* hello.jar : 스프링 부트로 만든 간단한 웹 어플리케이션 ([다운로드](https://github.com/madvirus/docker-start))
* entrypoint.sh : hello.jar를 실행하기 위한 간단한 쉘 파일

entrypoint.sh 파일은 다음과 같다.

#!/bin/sh

ACTIVE\_PROFILE="${PROFILE:-dev}"

echo "ACTIVE\_PROFILE=${ACTIVE\_PROFILE}"

exec java -Djava.security.egd=file:/dev/./urandom \

-Dspring.profiles.active=${ACTIVE\_PROFILE} \

-jar hello.jar

이 코드는 spring.profiles.active의 값으로 PROFILE 환경 변수를 사용한다. 이 환경 변수가 존재하면 그 값을 사용하고 존재하지 않으면 dev를 사용한다. hello.jar는 웹 어플리케이션으로 선택한 프로필에 따라 제공하는 URL이 달라진다.

* prod 프로필 : /actuator/env, /actuator/metric, /actuator/info, /actuator/health 제공
* 그 외 프로필 : /actuator/beans 등 거의 모든 /actuator 엔드포인트 제공

**주요 명령: FROM, RUN, ENV, COPY, ENTRYPOINT**

hello.jar와 entrypoint.sh 파일로 도커 이미지를 만들기 위해 사용할 Dockerfile은 다음과 같다.

FROM openjdk:8-jdk-alpine

RUN apk --no-cache add tzdata && cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime

WORKDIR /app

COPY hello.jar hello.jar

COPY entrypoint.sh run.sh

RUN chmod 774 run.sh

ENV PROFILE=local

ENTRYPOINT ["./run.sh"]

FROM, ENTRYPOINT 외에 몇 가지 명령어를 추가로 사용했다. 각 명령어는 다음과 같다.

* FROM : 이미지를 생성할 때 사용할 기반 이미지를 지정한다. 예제에서는 openjdk:8-jdk-alpine 이미지를 사용했다. 이 이미지는 알파인 OS에 JDK 8을 설치한 이미지이다.
* RUN : 이미지를 생성할 때 실행할 코드를 지정한다. 예제에서는 패키지를 설치하고 파일 권한을 변경하기 위해 RUN을 사용했다.
* WORKDIR : 작업 디렉토리를 지정한다. 해당 디렉토리가 없으면 새로 생성한다. 작업 디렉토리를 지정하면 그 이후 명령어는 해당 디렉토리를 기준으로 동작한다.
* COPY : 파일이나 폴더를 이미지에 복사한다. 위 코드에서 두 번째 COPY 메서드는 entrypoint.sh 파일을 이미지에 run.sh 이름으로 복사한다. 상대 경로를 사용할 경우 WORKDIR로 지정한 디렉토리를 기준으로 복사한다.
* ENV : 이미지에서 사용할 환경 변수 값을 지정한다. 위 코드는 PROFILE 환경 변수의 값으로 local을 지정했는데 이 경우 컨테이너를 생성할 때 PROFILE 환경 변수를 따로 지정하지 않으면 local을 기본 값으로 사용한다.
* ENTRYPOINT : 컨테이너를 구동할 때 실행할 명령어를 지정한다. 위에서는 run.sh을 실행하도록 설정했다.

위 Dockerfile을 이용해서 생성한 이미지는 /app 디렉토리에 hello.jar 파일과 run.sh 파일을 포함하며 컨테이너를 시작할 때 run.sh 파일을 실행한다.

먼저 docker build 명령어를 이용해서 이미지를 만들어보자.

$ docker build --tag hello:1.0 .

Sending build context to Docker daemon 18.39MB

Step 1/8 : FROM openjdk:8-jdk-alpine

---> a3562aa0b991

Step 2/8 : RUN apk --no-cache add tzdata && cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime

---> Running in a6973d93f849

fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/main/x86\_64/APKINDEX.tar.gz

fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/community/x86\_64/APKINDEX.tar.gz

(1/1) Installing tzdata (2019b-r0)

Executing busybox-1.29.3-r10.trigger

OK: 107 MiB in 55 packages

Removing intermediate container a6973d93f849

---> d5c2dda85cbc

Step 3/8 : WORKDIR /app

---> Running in c1b187ed05b6

Removing intermediate container c1b187ed05b6

---> 85f74df29ec8

Step 4/8 : COPY hello.jar hello.jar

---> be5100270e6c

Step 5/8 : COPY entrypoint.sh run.sh

---> cca2ad0f28fb

Step 6/8 : RUN chmod 774 run.sh

---> Running in dd9b59be1c12

Removing intermediate container dd9b59be1c12

---> 854ba56e2a00

Step 7/8 : ENV PROFILE=local

---> Running in eae1bb4e9b5c

Removing intermediate container eae1bb4e9b5c

---> f4ba7e11a824

Step 8/8 : ENTRYPOINT ["./run.sh"]

---> Running in 449c98a2bdf9

Removing intermediate container 449c98a2bdf9

---> 023caa2dd38d

Successfully built 023caa2dd38d

Successfully tagged hello:1.0

이미지를 성공적으로 만들었으면 docker run 명령어로 이미지를 이용해서 컨테이너를 시작하자.

$ docker run --rm -p 9090:9090 hello:1.0

ACTIVE\_PROFILE=local

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v2.1.9.RELEASE)

...생략

docker run 명령어를 실행하면 ACTIVE\_PROFILE=local 문자열이 출력되는데 이 문자열은 run.sh이 출력한 것이다. run.sh 파일은 PROFILE 환경 변수의 값을 ACTIVE\_PROFILE에 할당하므로 출력 결과를 통해 PROFILE 환경 변수의 값으로 local을 사용한 것을 알 수 있다(run.sh 파일은 PROFILE 환경 변수가 없으면 dev를 기본 값으로 사용하도록 구현했다). 이 값은 Dockerfile에서 ENV 명령어를 사용해서 PROFILE 환경 변수에 지정한 값을 사용한 것이다.

PROFILE 환경 변수의 값을 바꾸고 싶다면 컨테이너를 시작할 때 -e 옵션을 사용해서 다른 값을 지정하면 된다.

$ docker run --rm -p 9090:9090 -e PROFILE=prod hello:1.0

ACTIVE\_PROFILE=prod

ENTRYPOINT의 두 형식

ENTRYPOINT는 exec 형식과 shell 형식의 두 가지 형식으로 지정할 수 있다. 이 중 exec 형식을 추천한다. 예에서 사용한 앞서 예에서도 ["echo", "hello"]나 ["./run.sh"]는 모두 exec 형식이다.

exec는 다음 형식을 갖는다.

* ENTRYPOINT ["실행명령어", "인자1", "인자2", ... ]

배열에서 첫 번째는 실행할 명령어이다. 두 번째부터는 명령어의 인자로 전달된다.

shell 형식은 실행 명령어를 문자열로 입력한다.

ENTRYPOINT echo hello

exec 형식을 사용할 때와의 차이점은 shell 형식을 사용하면 /bin/sh -c로 명령을 실행한다는 것이다. 즉 위와 같은 ENTRYPOINT를 사용하면 실제 실행하는 명령어는 다음과 같다.

/bin/sh -c 'echo hello'

여러 개 ENTRYPOINT를 지정해도 마지막 한 개만 적용된다.

COPY와 ADD 명령어

COPY는 파일을 이미지에 복사한다. 형식은 다음과 같다.

* COPY src1 src2 ... dest
* COPY ["src1", "src2", ..., "dest"]

dest가 상대 경로면 WORKDIR 명령어로 지정한 경로에 복사한다. 예를 들어 다음은 entry.sh 파일을 이미지의 /app/run.sh 파일로 복사한다.

WORKDIR /app

COPY entry.sh run.sh

다음은 dest로 절대 경로를 사용했다. 이 경우 entry1.sh 파일은 /usr/bin/ 디렉토리에 복사하고 entry2.sh 파일은 /usr/bin/ 디렉토리에 run.sh로 복사한다.

WORKDIR /app

COPY entry1.sh /usr/bin/

COPY entry2.sh /usr/bin/run.sh

src는 "\*"이나 "?"와 같은 와일드카드를 포함한 GO 언어의 [매치 패턴](https://golang.org/pkg/path/filepath/#Match)을 이용할 수 있다.

ADD 명령어는 COPY 명령어와 비슷하게 이미지에 복사한다. COPY 명령어와의 차이라면 복사한 파일의 압축을 푼다는 것이다. 아래 명령어는 pinpoint-agent-1.8.4.tar.gz 파일을 /pinpoint 디렉토리에 풀어서 복사한다.

ADD pinpoint-agent-1.8.4.tar.gz /pinpoint

빌드 컨텍스트 주의 사항

도커 이미지를 빌드할 때 출력되는 메시지를 보자.

$ docker build --tag hello:1.0 .

Sending build context to Docker daemon 18.39MB

...생략

빌드 컨텍스트를 전송한다는 메시지가 나온다. 빌드 컨텍스트는 빌드를 실행할 때 사용할 파일 집합이다. docker build 명령어는 도커 데몬에 빌드 컨텍스트를 전송한다. 빌드 컨텍스트는 빌드 명령을 실행하는 디렉토리와 그 하위 디렉토리에 포함된 전체 파일이다.

빌드 컨텍스트에는 COPY나 ADD에서 사용하지 않는 파일도 포함되므로 빌드 컨텍스트에는 필요한 파일만 포함해야 한다. 부득이하게 빌드 과정에서 사용하지 않는 파일이 존재할 경우 .dockerignore 파일을 만들어 컨텍스트에서 제외할 대상을 지정하면 된다.

도커 허브를 이용한 이미지 공유

이미지를 생성하면 도커 리포지토리를 이용해서 공유할 수 있다. 다양한 도커 리포지토리가 존재하는데 이 중 대표적인 것이 도커 허브이다. 지금까지 예에서 사용한 nginx나 alpine과 같은 이미지가 위치한 곳이기도 하다. [https://hub.docker.com](https://hub.docker.com/) 사이트에 가입하면 도커 허브에 이미지를 푸시해서 공유할 수 있다.

푸시 방법은 어렵지 않다. 먼저 생성한 이미지에 추가로 태그를 붙인다. 추가 태그에는 리포지토리 계정 이름을 사용한다. 예를 들어 도커 허브 계정이 madvirus라면 madvirus/리포지토리이름:태그 형태를 사용한다. 다음은 simplenode:0.1 이미지에 madvirus/simplenode:0.1 태그를 추가하는 예인데 이때 madvirus라 도커 허브 계정이다.

$ docker tag simplenode:0.1 madvirus/simplenode:0.1

docker login 명령어를 사용해서 도커 허브에 로그인한다.

$ docker login

Login with your Docker ID to push and pull images from Docker Hub. ...생략

Username: madvirus

Password:

...생략

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

로그인에 성공했다면 docker push로 이미지를 푸시한다.

$ docker push madvirus/simplenode:0.1

The push refers to repository [docker.io/madvirus/simplenode]

8eac20925476: Pushed

a3f62211f277: Pushed

e7ae04d3f37c: Mounted from library/node

e29ab5067804: Mounted from library/node

ae4ceb8dc557: Mounted from library/node

f1b5933fe4b5: Mounted from library/node

0.1: digest: sha256:d0299960fb25aae581200d143291e3fd48a18dfbdd53333df8557755b00bef14 size: 1572

푸시가 끝나면 도커 허브에서 이미지를 가져와 사용할 수 있다.

docker run -d --rm -p 5000:5000 madvirus/simplenode:0.1

관련 자료

* hello.jar 파일 다운로드: <https://github.com/madvirus/docker-start>