# **Dockerfile 명령어**

* 도커 파일에 사용되는 문법

## **FROM**

* 빌드 프로세스를 시작하는 기본 이미지를 정의
* FROM 명령어는 도커 런타임에 사용할 도커 이미지를 지정

## **USER**

* 명령 및 컨테이너 실행시 적용할 유저 설정

## **MAINTAINER**

* 이미지 생성한 사람의 이름 및 정보

## **LABEL**

* 도커 이미지에 메타데이터를 추가
* 메타데이터는 키-값 쌍으로 되어 있음
* 이미지 작성 목적으로 버전, 타이틀, 설명, 라이선스 정보 등을 작성
* 1개 이상 작성 가능

## **ARG**

* 사용자가 docker build 명령을 사용하여 빌더에 전달할 수 있는 변수를 정의
* docker bulid 시점에서 변숫값을 전달하기 위해 --build-arg=인자를 정의하여 사용

ARG db\_name

docker build --build-arg db\_name=mysql .

## **COPY**

* 컨테이너 빌드시 호스트의 파일을 컨테이너로 복사
* 원본의 새 파일, 디렉터리 또는 리모트 파일 URL을 복사하고 지정된 대상 경로에 생성 중인 이미지의 파일 시스템에 추가

## **ADD**

* 컨테이너 빌드시 호스트의 파일을 컨테이너로 복사
* 호스트에서 파일을 복사하고 컨테이너 대상 위치에 추가
* COPY와의 차이점
  + ADD와 COPY의 차이점은 없는 것처럼 보이지만(복사의 기능은 동일)
  + COPY는 로컬의 파일만 이미지에 추가할 수 있고, ADD는 외부 URL 및 tar 파일에서도 파일을 추가할 수 있다는 점이 다름
  + COPY의 기능이 ADD에 포함되어 있다고 보면 됨

## **VOLUME**

* 컨테이너의 마운트 지점을 만듬
* 파일 또는 디렉터리를 컨테이너의 디렉터리로 마운트
* 동일한 이미지를 사용해서 새 컨테이너를 만들 때 이전 볼륨과 격리되는 새 볼륨을 생성

## **RUN**

* 컨테이너 빌드를 위해 base images에서 실행할 commands
* 명령과 해당 매개변수를 받아 이미지에서 컨테이너를 실행
* 보통 소프트웨어 패키지를 설치하는데 이 명령어를 사용

## **CMD**

* 컨테이너 동작 시 자동으로 실행할 서비스나 스크립트 지정
* ENTRYPOINT에 매개변수를 제공
* 이 명령어는 docker run과 유사하지만 컨테이너가 인스턴스화 된 후에만 실행될 수 있음
* 여러 개의 CMD를 작성해도 마지막 하나만 처리됨

## **ENTRYPOINT**

* 실행 파일로 실행할 컨테이너를 구성
* CMD와 마찬가지로 생성된 이미지가 컨테이너로 실행될 때 사용하지만, 컨테이너가 실행될 때 명령어 및 인자 값을 전달하여 실행한다는 점

## **WORKDIR**

* 컨테이너상에서 작업할 경로(디렉터리) 전환을 위해 작성
* 따라서 컨테이너 빌드시 명령이 실행될 작업 디렉터리 설정
* WORKDIR을 설정하면 RUN, CMD, ENTRYPOINT, COPY, ADD 명령문은 해당 디렉터리를 기준으로 실행

## **ENV**

* 환경 변수를 설정
* 이미지 안에 각종 환경 변수를 지정하는 경우 작성
* 애플리케이션 사용을 쉽게 하려면 사전에 구성되어야 하는 환경 변수들이 있음
  + 자바 홈 디렉터리
  + 특정 실행 파일의 경로를 보장하기 위해 절대 경로 지정을 위한 PATH 설정
  + 프로그램 버전 등을 사전에 설정
  + 또한, 반복된 표현이 사용되는 경우에도 환경 변수 설정을 권장

ENV JAVA\_HOME /usr/lib/jvm/java-8-oracle

## **EXPOSE**

* 컨테이너가 호스트 네트워크를 통해 들어오는 트래픽을 리스닝(listening)하는 포트와 프로토콜을 지정하기 위해 작성
* docker run 사용시 -p 옵션을 통해 사용
* 컨테이너 생성 시에는 반드시 -p 옵션을 사용해야 실제 포트가 바인딩 되기 때문에 EXPOSE는 다른 사용자에게 해당 dockerfile에서 포트를 사용한다고 알려주는 관점이 강함
* 컨테이너 동작 시 외부에서 사용할 포트 지정
  + EXPOSE에서 PORT를 지정해도 컨테이너 실행할 때 반드시 -p 옵션을 사용해야 port가 바인딩 됨

# **ENTRYPOINT와 CMD의 차이점**

* 커맨드와 동일하게 컨테이너가 시작할 때 수행할 명령을 지정한다는 점은 동일
* entrypoint는 커맨드를 인자로 사용할 수 있는 스크립트의 역할을 할 수 있음

$ docker run -it --name no\_entrypoint ubuntu:14.04 /bin/bash

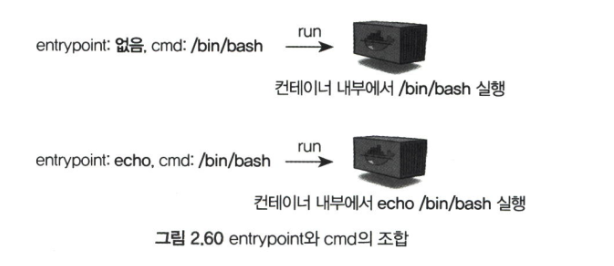
root@be7ce6c45fb5:/#

-------------------------------------------------

$ docker run -it --entrypoint="echo" --name yes\_entrypoint ubuntu:14.04 /bin/bash

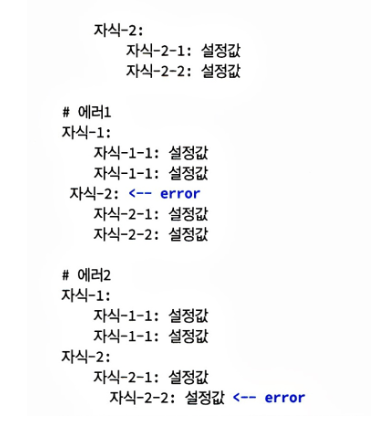
/bin/bash

$



* entrypoint로 전달한 값이 터미널에 출력
* 컨테이너에 entrypoint가 설정되면 run 명령어의 맨 마지막에 입력된 cmd를 인자로 삼아 명령어를 출력
* 위의 예제에서는 echo /bin/bash가 컨테이너에서 실행되어 /bin/bash가 출력
* CMD와 entrypoint가 둘 다 설정되지 않으면 컨테이너는 생성되지 않고 에러를 출력하므로 반드시 둘 중 하나는 설정해야 함
* ENTRYPOINT는 도커 컨테이너 실행 시 항상 수행해야 하는 명령어를 지정(웹 서버나 데이터베이스등의 데몬 실행)
* CMD는 도커 컨테이너 실행 시 다양한 명령어를 지정하는 경유 유용

**도커 컴포트 주의 사항**

* 야믈 코드의 계층 구조는 부모-자식 간의 레벨을 들여쓰기로 엄격하게 구분해야 함
* 주의할 것은 들여쓰기를 탭이 아닌 공백으로 정확히 구분해야 한다는 점  
  

# **항목**

* services : 컨테이너를 정의
* networks : 네트워크를 정의
* volumes : 볼륨을 정의

# **컴포즈 문법**

* service
  + 배포할 서비스를 지정
  + 서비스 이름은 도커 인스턴스에 대한 DNS 엔트리이며, 다른 서비스에서 엑새스하는 데 사용
* container\_name
  + 생략 시 자동으로 부여.
  + 디렉터리\_서비스명\_n으로 생성
  + docker run --name 옵셥과 동일
* image : 사용할 이미지를 지정
* network
  + --net
  + 접속할 네트워크 지정
  + 복잡한 토폴로지를 만들 수 있도록 커스텀 네트워크를 지정
  + 타입(host, overlay, macvlan, none)을 지정하지 않았다면 디폴트 타입은 bridge
  + 브리지 네트워크를 사용하면 동일한 넽워크 내 컨테이너 연결을 관리할 수 있고, 이 네트워크는 동일한 도커 데몬 호스트에서 실해되는 컨테이너에만 적용됨
* alias
  + 네트워크 내 서비스에 대한 호스트 별명을 지정
* ports
  + 서비스 내부 포트와 외부 호스트 포트를 지정하여 바인드
  + 외부 노출 포트 지정
  + docker run -p 옵셥과 동일
* expose
  + 호스트 운영체제와 직접 연결하는 포트를 구성하지 않고 서비스만 포트를 노출
  + 필요시 링크로 연결된 서비스와 서비스 간의 통신만 사용
  + ports가 호스트 외부와 통신이라면 expose는 컨테이너들 사이의 포트임
* networks
  + 최상위 레벨의 networks에 정의된 네트워크 이름을 작성
  + docker run의 --net 옵션과 동일
* volumes
  + 서비스 내부 디렉터리와 호스트 디렉터리를 연결하여 데이터 지속성 설정
  + docker run -v 옵션과 동일
* environment
  + 서비스 내부 환경 변수 설정
  + 환경 변수가 많은 경우에는 파일 .env 로 만들어 env\_file 옵션에 파일명을 지정
  + docker run -e 옵션과 동일
* command
  + 서비스가 구동 이후 실행할 명령어 작성
  + docker run의 마지막에 작성되는 명령어
* restart
  + 서비스 재시작 옵션 지정
    - no : 수동 재시작
    - always : 컨테이너 수동 제어를 제외하고 항상 재시작
    - on-failure : 오류가 있을 시 재시작
  + docker run --restart 옵션과 동일
* depends\_on
  + 서비스 간의 종속성을 의미하며 먼저 실행해야 하는 서비스를 지정하여 순서 지정
  + 이 옵션에 지정된 서비스가 먼저 시작됨

# **명령어**

* docker-compose up -d
  + 애플리케이션 이미지를 빌드하고 정의된 서비스를 시작
  + 이 명령은 필요한 모든 이미지를 내려받아 배포하고 컨테이너를 시작
  + -d 매개변수는 백그라운드 모드에서 도커를 실행하도록 지정
* docker-compose logs
  + 최신 배포에 대한 모든 정보를 볼 수 있음
* docker-compose logs <service\_id>
  + 특정 서비스에 대한 로그를 볼 수 있음
* docker-compose ps
  + 시스템에 배포한 모든 컨테이너 목록을 출력
* docker-compse stop
  + 서비스를 마치고 나서 서비스를 중지
  + 이렇게 하면 컨테이너도 중지됨
* docker-compse down
  + 모든 것을 종료하고 컨테이너도 모두 제거