Dockerize Everythings!

우여명 (Matholic)

Index

- 이번 발표의 목적
- Dockerfile 관련 개념 소개
- nodejs express dockerize
- spring-boot dockerize
- python flask dockerize
- tesseract dockerize
- docker push

이번 발표의 목적

도커라이징의 기본 개념과 구현방법에 대해서 알아보고

개인적으로 도커 이미지를 만들어본 경험을 공유합니다.

Dockerfile

> 도커는 도커파일의 명령어들을 읽어서 자동으로 도커 이미지를 만들 수 있다. 도커파일은 텍스트 문서로 사용자가 이미지를 어셈블하기 위해 명령 행에서 호출 할수 있는 모든 명령을 포함한다. `docker build` 명령어를 사용하여 사용자는 여러가지 명령어들을 연속적으로 실행하는 자동화된 빌드를 만들 수 있다.

Dockerize

> 실행가능 한 프로그램을 도커이미지로 만들어 컨테이너 실행만으로 실행이 가능하게 하는 것

간단하게 빌드하기

```
echo "<h1> Hello Docker </h1>" > index.html
cat index.html
echo "FROM nginx:alpine\nCOPY . /usr/share/nginx/html" > Dockerfile
cat Dockerfile
docker build -t nginx-hello-docker .
docker run -d --rm -p 8888:80 nginx-hello-docker
docker ps
curl http://localhost:8888
```

https://asciinema.org/a/kd9n3WEc8rXLdpUt5hguuSMb6

Dockerfile 빌드과정 살펴보기

1. build context 불러오기

```
$ docker build -t nginx-hello-docker .
Sending build context to Docker daemon  3.072kB
```

- `docker build` 명령어의 맨 마지막에 지정된 위치에 있는 파일을 전부 포함.
- 위 코드에서는 현재 디렉토리(`.`)
- 단순 파일뿐 아니랑 하위 디렉토리도 전부 포함하므로 불필요한 파일이 포함된다면 빌드 속도가 느려짐. `.dockerignore` 파일을 이용하여 제외시킬 수 있음

2. Dockerfile을 이용한 컨테이너 생성과 커밋

```
Step 1/2 : FROM nginx:alpine
 ---> bc7fdec94612 # Step1 이미지 레이어 ID
Step 2/2 : COPY . /usr/share/nginx/html
 ---> 87f5503dfb50 # Step2 이미지 레이어 ID
Successfully built 87f5503dfb50 # 최종 이미지 레이어 ID
Successfully tagged nginx-hello-docker:latest
```

- Dockerfile에서 명령어 한 줄이 실행될 때마다 이전 단계에서 생성된 이미지에 의해서 새로운 컨테이너가 생성되며, 그 컨테이너에서 명령어를 수행하고 다시 새로운 이미지로 레이어로 저장(커밋)함
- 이미지 빌드가 완료되면 Dockerfile의 명령어 줄 수 만큼 레이어가 존재하게 되며, 중간에 컨테이너도 같은 수 만큼 생성하고 삭제됨.

3. 한번 이미지 빌드를 마쳤다면 다시 할때는 캐싱

```
$ docker build -t nginx-hello-docker .
Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM nginx:alpine
 ---> bc7fdec94612
Step 2/2 : COPY . /usr/share/nginx/html
 ---> Using cache
 ---> 87f5503dfb50
Successfully built 87f5503dfb50
Successfully tagged nginx-hello-docker:latest
```

만약 캐시를 하고 싶지 않다면 `--no-cache` 옵션 사용

```
docker build -t nginx-hello-docker --no-cache .
```

Docker layer

도커 레이어를 확인해보는 명령어 : docker history image_name:tag

이미지 레이어와 해당 Dockerfile 명령어를 볼 수 있다.

```
$ docker history nginx-hello-docker:latest
IMAGE               CREATED             CREATED BY
463893708fe8        7 minutes ago       /bin/sh -c #(nop) COPY dir:44
bc7fdec94612        2 weeks ago         /bin/sh -c #(nop)  CMD ["ngin
<missing>           2 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL
<missing>           2 weeks ago         /bin/sh -c #(nop)  EXPOSE 80/
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:1
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:a
<missing>           2 weeks ago         /bin/sh -c GPG_KEYS=B0F425337
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV NGINX_
<missing>           4 months ago        /bin/sh -c #(nop)  LABEL main
<missing>           5 months ago        /bin/sh -c #(nop)  CMD ["/bin
<missing>           5 months ago        /bin/sh -c #(nop) ADD file:09
```

기본적인 Dockerfile 명령어

- `FROM` : 생성할 이미지의 베이스가 될 이미지 ex) `FROM ubuntu:16.04`
- `RUN` : 이미지를 만들기 위해서 컨테이너 내부에서 명령으를 실행함 ex) `RUN npm install`
- `ADD` or `COPY` : 컨텍스트에서 컨테이너 내부로 파일을 추가 ex) `COPY . /root/app`
- `WORKDIR` : working dir 이동, `cd` 명령어와 같다
- `EXPOSE` : 실행되는 컨테이너에서 노출할 포트를 설정
- `ENV` : 실행되는 컨테이너에서 사용될 환경변수 설정
- `VOLUME` : 실행되는 컨테이너와 호스트가 공유할 디렉토리 설정
- `ARG` : 도커 빌드시에 추가로 입력받아 Dockerfile 내에서 사용될 변수의 값을 설정

Dockerfile `CMD` & `ENTRYPOINT`

- Docker는 기본적으로 RUNTIME시에 foreground mode 만 가능
- foreground에 실행되는 프로세스가 없으면 도커는 컨테이너 프로세스를 종료함
- 즉, Dockerfile에서 `CMD` & `ENTRYPOINT` 명령어를 통해서 foreground로 프로세스를 실행해야함

```
CMD [ "npm", "start" ]

or

ENTRYPOINT npm
CMD ["start" ]
```

[참고] ENTRYPOINT vs CMD

- ENTRYPOINT가 있을 경우에는 컨테이너 실행시 ENTRYPOINT로 정의한 스크립트 혹은 명령어를 수행하고 CMD는 해당 명령의 인자가 됨.
- ENTRYPOINT가 없을 경우에는 CMD로 정의한 스크립트 혹은 명령어를 컨테이너 실행시에 수행함

Dockerize process

1. 어떤 파일, 설정, 환경 변수 및 명령어 실행이 필요한지 (pseudo) 스크립트로 정리한다.
2. 그 스크립트로 도커 파일로 수정한다.
3. `ENTRYPOINT` 또는 `CMD` 를 이용해 컨테이너가 실행될 때 수행할 명령어를 작성한다.
4. `docker build` 로 이미지를 빌드한다.

python flask dockerize

1. 스크립트 정리

```
소스 전체를 컨테이너에 추가

pip install -e .

서버 실행
```

2. Dockerfile 작성

```
FROM python:alpine3.6
MAINTAINER voyagerwoo

ADD . /root/
WORKDIR /root/
RUN pip install -e .

ENTRYPOINT ["gunicorn"]
CMD ["-w", "4", "-b", "0.0.0.0:9460", "app:app"]
```

실습

```
git clone https://github.com/voyagerwoo/flask-boilerplate
cd flask-boilerplate
docker build -t flask-boilerplate .
docker run -d --rm --name flask-boilerplate -p 9460:9460 \
        flask-boilerplate
```

https://asciinema.org/a/HSXOFbmbixtiJhwa8b1ITyUDZ

spring-boot dockerize 1

스프링예제 사이트에서 알려준 Dockerfile

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
RUN echo ${JAR_FILE}
ARG JAR_FILE
ADD ${JAR_FILE} app.jar
ENTRYPOINT ["java"]
CMD ["-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

빌드된 jar파일을 ADD하고 실행하면 끝!

(참고 링크 : https://github.com/voyagerwoo/vw.demo.helloworld)

spring-boot dockerize 2

maven plugin

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.4.13</version>
  <configuration>
    <imageName>
      ${docker.registry.host}/${project.artifactId}
    </imageName>
    <dockerDirectory>src/main/docker</dockerDirectory>
    <useConfigFile>true</useConfigFile>
    <registryUrl>${docker.registry.host}</registryUrl>
    <!--dockerHost>https://${docker.registry.host}</dockerHost-->
    <resources>
      <resource>
        <targetPath>/</targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.build.finalName}.jar</include>
      </resource>
    </resources>
    <forceTags>false</forceTags>
    <imageTags>
      <imageTag>${project.version}</imageTag>
    </imageTags>
  </configuration>
</plugin>
```

spring-boot dockerize 2

Dockerfile

```
FROM openjdk:8-jdk-alpine
RUN mkdir /root/app
COPY petclinic-rest-0.0.1.jar /root/app/app.jar
WORKDIR /root/app
ENTRYPOINT ["java"]
CMD ["-Djava.security.egd=file:/dev/./urandom","-jar","app.jar"]
```

```
./mvnw clean package docker:build -Dmaven.test.skip=true
```

(참고 링크 : https://github.com/voyagerwoo/petclinic-rest)

참고

개인적으로, 자바앱의 경우에는 host에서 빌드하고 그 결과 jar 또는 war를 image에 추가하는 것을 선호하는데,

이유는 maven, gradle이 의존하는 라이브러리를 가지고 오는데 시간이 많이 들기 때문이며, 호스트에서 그것을 캐싱할 수 있기 때문이다.

tesseract dockerize

https://github.com/tesseract-ocr/tesseract

- ocr 엔진
- C++ 작성됨
- python을 이용해서 tesseract rest service를 만들려고 함.
- 로컬에 설치하고 싶지 않음

tesseract dockerize

Dockerfile

```
FROM dimg.matholic.com:5000/base-ubuntu:latest
MAINTAINER voyagerwoo

LABEL "name"="base-tesseract"

RUN apt-get update

# Tesseract 의존 라이브러리 설치
RUN apt-get install -y g++ && \
    apt-get install -y autoconf automake libtool && \
    apt-get install -y autoconf-archive && \
    apt-get install -y pkg-config && \
    apt-get install -y libpng-dev && \
    apt-get install -y libjpeg8-dev && \
    apt-get install -y libtiff5-dev && \
    apt-get install -y zlib1g-dev && \
    apt-get install -y libicu-dev && \
    apt-get install -y libpango1.0-dev && \
    apt-get install -y libcairo2-dev
```

이어서..

```
# leptonica 1.74 버전 설치 및 빌드
WORKDIR /
COPY leptonica.tar.gz /
RUN tar xvzf leptonica.tar.gz
WORKDIR leptonica-1.74.4
RUN ./configure; make; make install
WORKDIR /

# Tesseract 다운로드 및 설치
RUN git clone https://github.com/tesseract-ocr/tesseract.
WORKDIR tesseract
RUN ./autogen.sh && ./configure && \
    LDFLAGS="-L/usr/local/lib" CFLAGS="-I/usr/local/inclu
    make install && ldconfig
WORKDIR /

# add traning data download
COPY my.traineddata /usr/local/share/tessdata

RUN apt-get clean && \
        apt-get autoremove -y && \
        rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

Docker push

docker tag 명령어를 통해서 푸시할 위치에 맞게 이미지 명을 변경해줘야한다.

- dockerhub : {username}/{imagename}:{tag}
- custom registry : {registry_host}/{imagename}:{tag}

참고 : https://github.com/voyagerwoo/vw.demo.helloworld/blob/master/.travis.yml

```
docker login -u ${DOCKER_USER} -p ${DOCKER_PASS}
docker tag vw.demo.helloworld:latest \
        voyagerwoo/vw.demo.helloworld:latest
docker tag vw.demo.helloworld:latest \
        voyagerwoo/vw.demo.helloworld:${TRAVIS_JOB_NUMBER}

docker push voyagerwoo/vw.demo.helloworld:latest
docker push voyagerwoo/vw.demo.helloworld:${TRAVIS_JOB_NUMBER}
```

감사합니다.