Dockek 04

[간단한 **Node.js** 어플만들기]

따라하며 배우는 도커

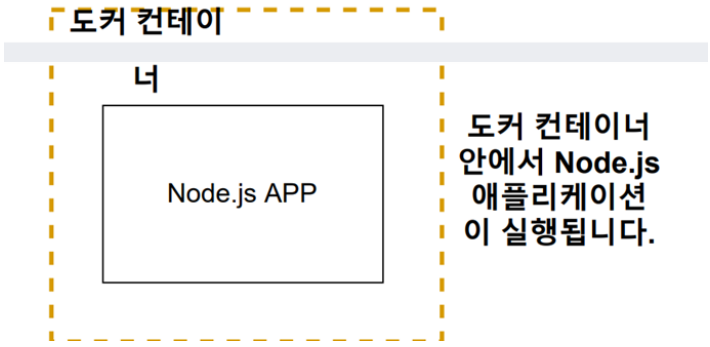
John Ahn

2023.02.12

간단한 **Node.js** 어플을 만들어보기

https://nodejs.org/fr/docs/guides/nodejs-docker-webapp/

노드js를 도커환경에서 실행하는 메뉴얼



이런 환경에서 실행되는 모습을 만들어 보고자 한다.

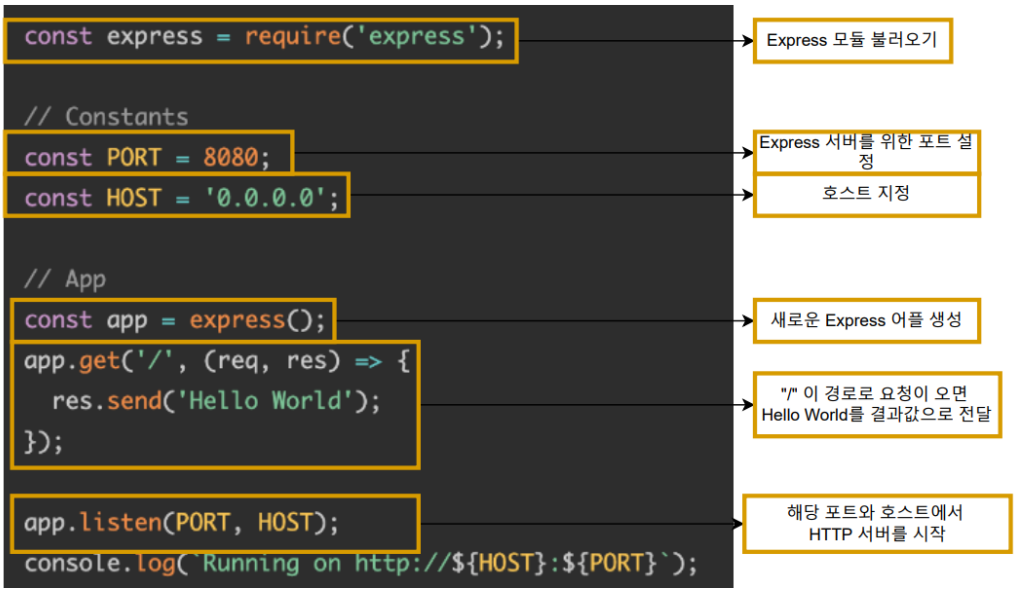
Node.js

package.json : 프로젝트 정보와 프로젝트에서 사용 중인 패키지의 의존성을 관리하는 곳.

→ npm init 명령어로 생성한다.

server.js : 시작과 실행한 관한 정보

server.js(시작점) 만들기



express

더 편하게 사용하기 위해 express 의존을 넣어준다.

Javascript 와 jQuery 관계처럼
Node.js의 API를 더 쉽고 유용하게 만들어준다.

도커파일 만들기

node.js 어플을 실행하기 위한 도커파일을 생성한다.

```
# 베이스 이미지를 명시해준다.
FROM baseImage

# 추가적으로 필요한 파일들을 다운로드 받는다.
RUN command

# 컨테이너 시작시 실행 될 명령어를 명시해준다.
CMD [ "executable" ]
```

```
#베이스 이미지를 명시해 준다. (현재 노드 버전 10)
#베이스 이미지가 노드환경이라는 것.
#From alpine 로 실행시 /bin/sh: npm: not found 에러 발생
#node 베이스 이미지는 npm을 가지고 있다.
From node:10

#추가적으로 필요한 파일을 다운받는다.
#package.json 내에 필요한 의존을 웹에서 다운받아준다.
RUN npm install

#실행될 명령어
CMD ["node", "server.js"]

# #여기까찌 작성시 에러가 발생한다.
```

```
 ---> Running in ee5fa9e668d0
npm WARN saveError ENOENT: no such file or directory, open '/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/package.json'
npm WARN !invalid#2 No description
npm WARN !invalid#2 No repository field.
npm WARN !invalid#2 No README data
npm WARN !invalid#2 No license field.
```

package.json이 없다는 경고가 발생하게 된다.

도커 파일을 만들때 Node 베이스 이미지 → 임시 컨테이너 로 만들때

파일 스냅샷이 임시 컨테이너로 넘어가지 않았기 때문에 RUN npm install부분이 실행 되지 않아서 그런다. 그래서 cpoy로 파일스냅샷을 복사해준다.

```
FROM node:10

COPY package.json ./

RUN npm install

CMD [ "node", "server.js" ]
```

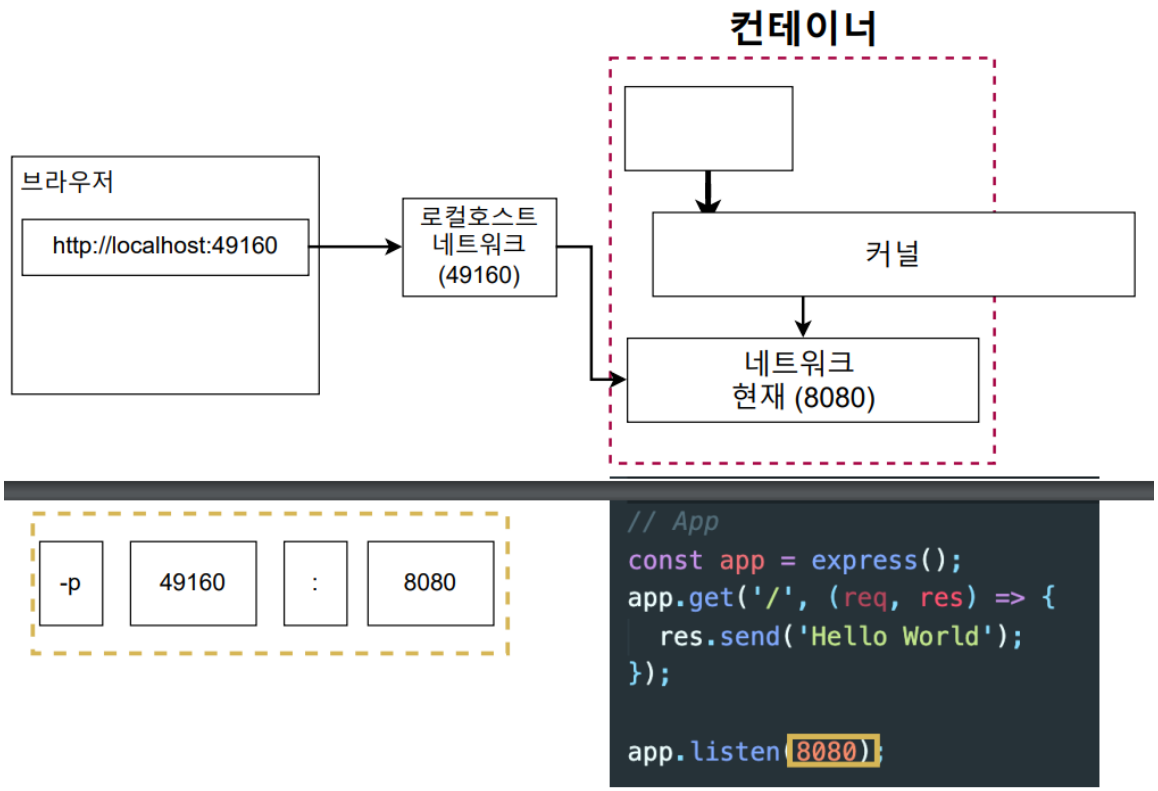
```
From node:10

#파일 스냅샷을 복사해준다.
COPY ./ ./

#추가적으로 필요한 파일을 다운받는다.
#package.json 내에 필요한 의존을 웹에서 다운받아준다.
RUN npm install

#실행될 명령어
CMD ["node", "server.js"]
```

http://localhost:9090/

server.js 호스트 번호를 바꿔줬다. (오라클 때문) 하지만 실행되지 않는다.

컨테이너 네트워크와 연결이 되지않으면 제대로 실행이 되지 않는다.

docker run -p 49160:8080 이미지 이름 으로 실행해야 한다**.**



포트를 매핑을 시켜줘야 한다.

즉, 지금 host 가 9090이고 매핑도 같게 매핑하고 싶다면

docker run -p 9090:9090 lece619/nodejs

이런 식으로 실행 해줘야 매핑이 된다.

도커 파일에 **workdirectory**

WORKDIR ?

이미지 안에서 어플리케이션 소스 코드를 가지고 있을 디렉토리를 생성하는 것.

```
D:\StudyResouce\dockerResource\docker_example\testapp>docker run -it lece619/nodejs ls
Dockerfile  dev   lib    mnt          package-lock.json  root  server.js  tmp
bin         etc   lib64  node_modules  package.json       run   srv        usr
boot        home  media  opt          proc               sbin  sys        var
```

워크 디렉토리를 생성 안하면 더럽다.

나누지 않는다면 문제점?

- 중복되는 파일이 덮어 씌워진다.
- 더럽다.

```
From node:10

WORKDIR /usr/src/app

#파일 스냅샷을 복사해준다.
COPY ./ ./

#추가적으로 필요한 파일을 다운받는다.
#package.json 내에 필요한 의존을 웹에서 다운받아준다.
RUN npm install

#실행될 명령어
CMD ["node", "server.js"]
```

실시간 소스 코드 변경

소스코드를 수정하면 다시 빌드 시켜주고 실행 시켜줬는데 매우 불편하다.

소스 수정시 → 이미지 빌드 → 다시 실행 → 확인

너무 긴 과정

재 빌드를 효율적으로 반영하는 법

```
#node 베이스 이미지는 npm을 가지고 있다.
From node:10

WORKDIR /usr/src/app

#파일 스냅샷을 복사해준다.
COPY package.json ./

#추가적으로 필요한 파일을 다운받는다.
#package.json 내에 필요한 의존을 웹에서 다운받아준다.
RUN npm install

COPY ./ ./

#실행될 명령어
CMD ["node", "server.js"]

# #여기까찌 작성시 에러가 발생한다.
```

COPY를 두개로 나눈이유는 RUN 이 실행 되었을 때 변경되었을 때는 CACHE를 사용하지 않지만
변경이 없다면 CACHE를 사용하여 빌드 과정이 효율적을 이뤄진다.

맥과 윈도우에서 다른 명령어

맥 : -v $(psd):/usr/src/app

윈도우 : -v %cd%:/usr/src/app

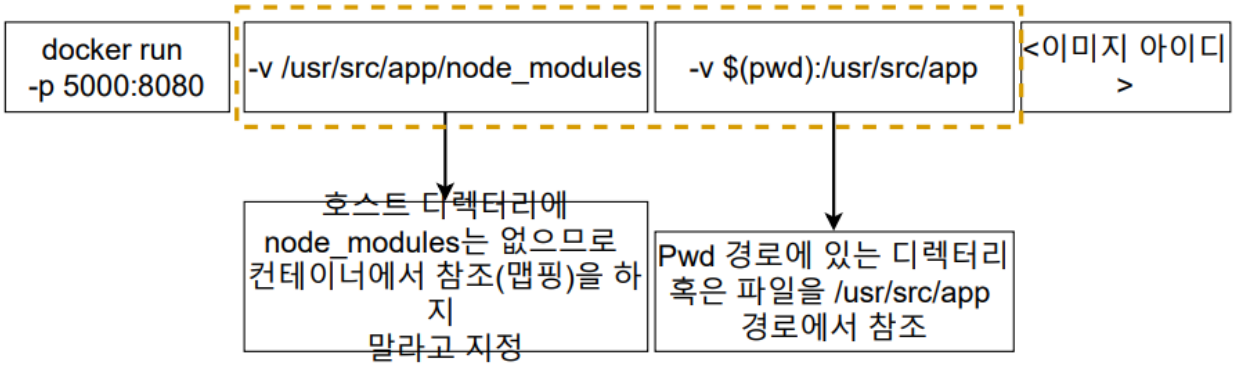
$(psd) → %cd%

Docker Volume 기능

COPY 와 다른가?

COPY : 로컬의 파일의 스냅샷을 도커 컨테이너에 복사해주는 것. 변경이 있을 때 복사해 주는것.

Volume : 도커 컨테이너가 로컬에 있는 파일을 참조(Mapping)를 해서 동작하게 된다.



명령어가 길어진다.

-v 참조될로컬폴더**:**참조할폴더 **(**로컬에 없으면 **:** 생략후 컨테이너 위치만 지정**)**

윈도우는 명령어가 %cd%

docker run -d -p 9090:9090 -v /usr/src/app/node_modules -v %cd%:/usr/src/app lece619/nodejs

실행시 빌드없이 재 실행만으로 소스코드 수정이 돌아간다.