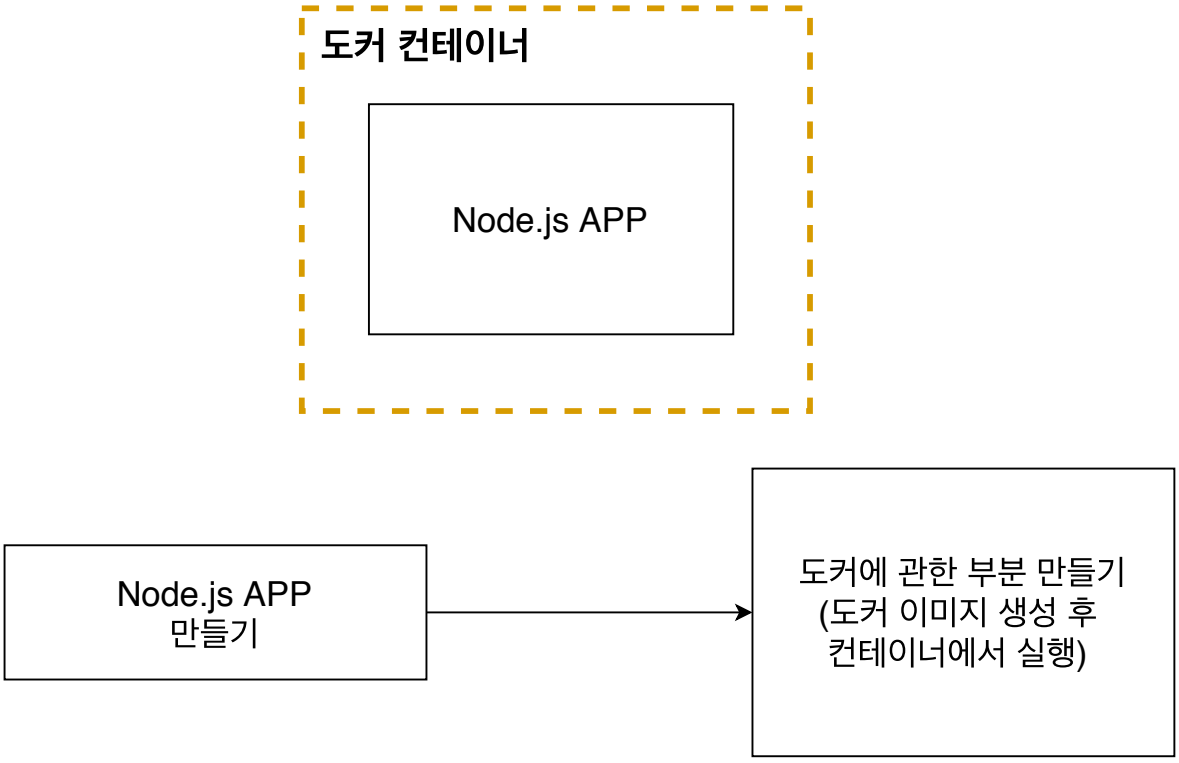
도커를 이용한 간단한 Node.js 어플 만들기

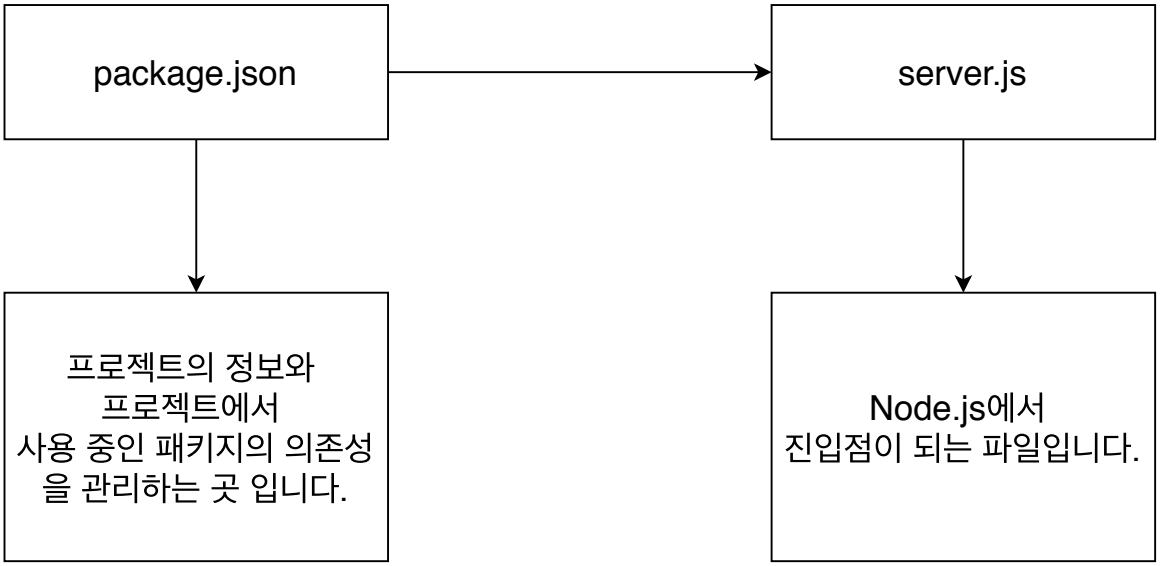
섹션 설명

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```



Node.js 앱 만들기

Node.js 앱 만들기 순서



package.json 만들기

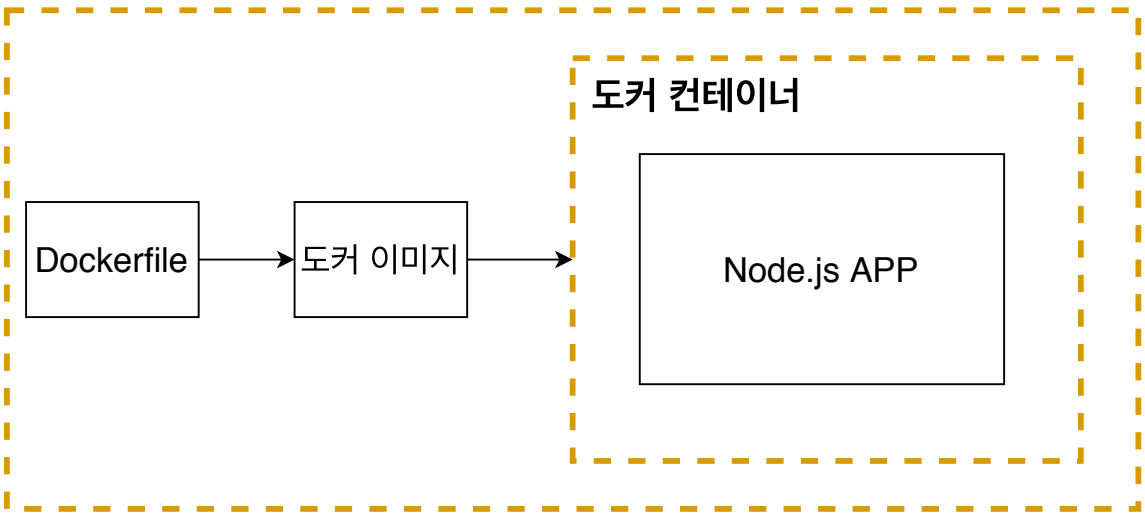
```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

server.js(시작점) 만들기

```
const express = require('express');
```

```
// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

Dockerfile 작성하기



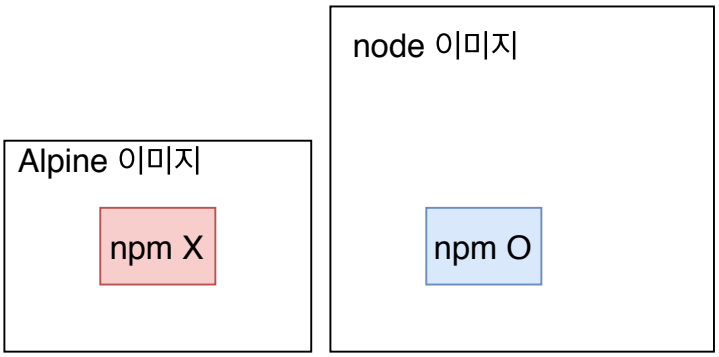

```
# 베이스 이미지를 명시해준다.
FROM baseImage

# 추가적으로 필요한 파일들을 다운로드 받는다.
RUN command

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

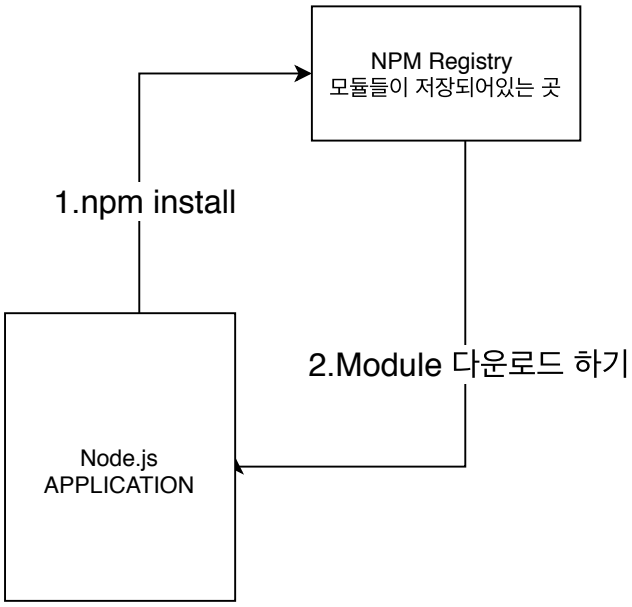
```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

먼저 저번에 했던것 처럼 가장 근본이 되는것 부터 작성해주겠습니다.

```
# 베이스 이미지를 명시해준다.
FROM baseImage

# 추가적으로 필요한 파일들을 다운로드 받는다.
RUN command

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

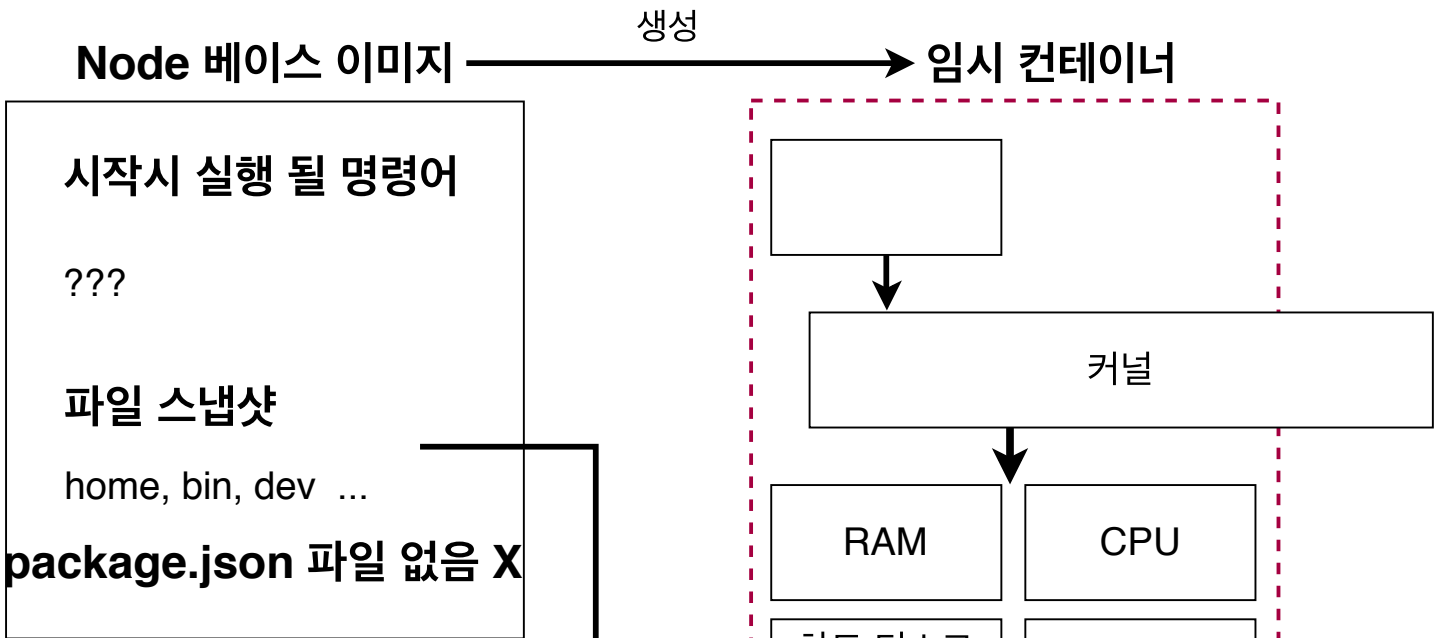
FROM	node:10
RUN	npm install
CMD	"node", "server.js"

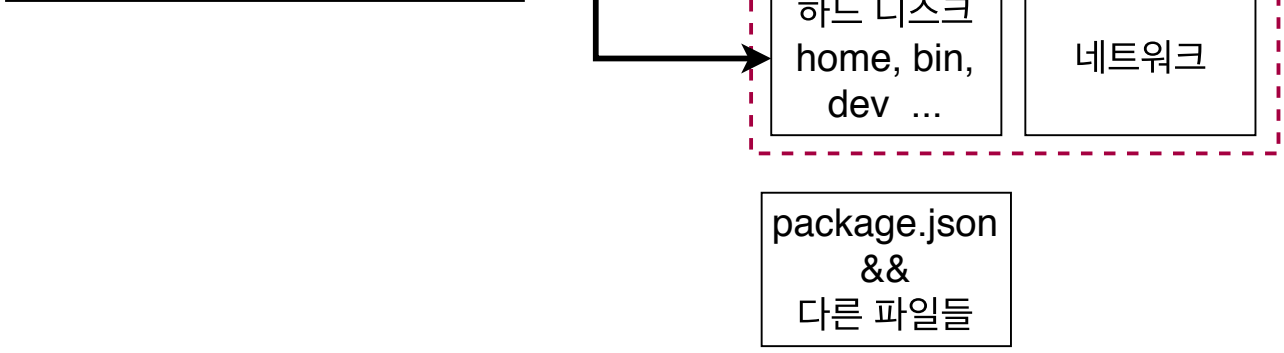




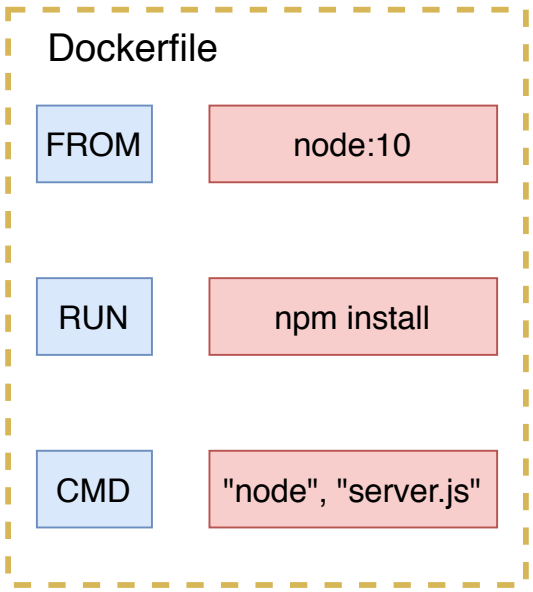
Package.json이 없다고 나오는 이유(dockerfile copy)

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

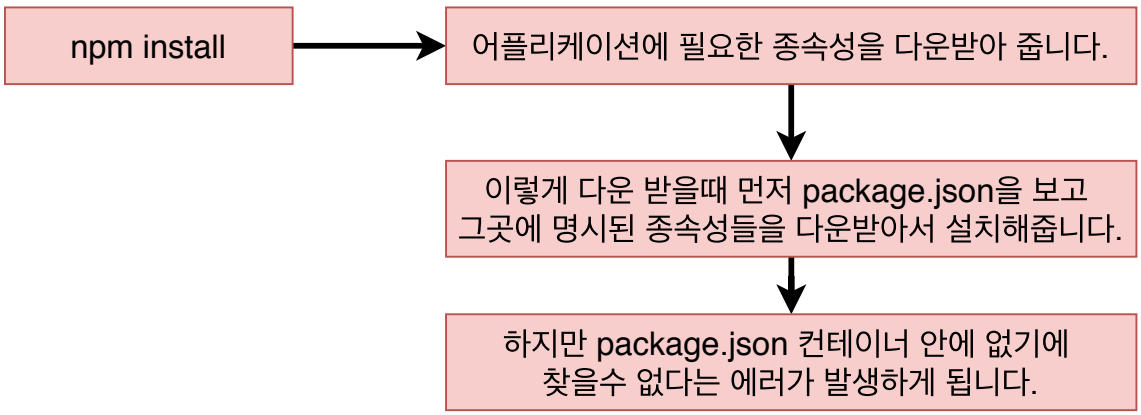


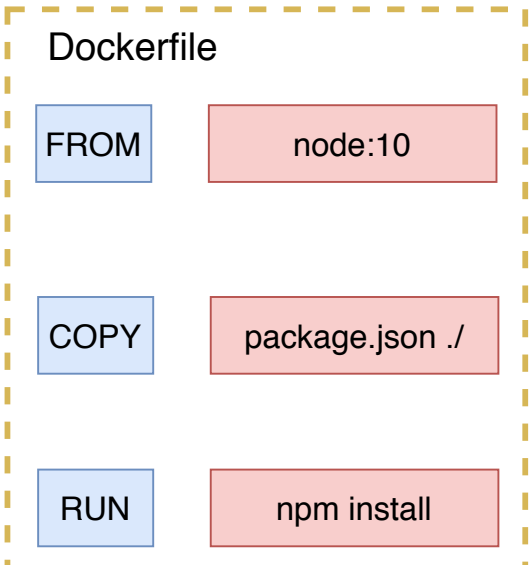


package.json
&&
다른 파일들

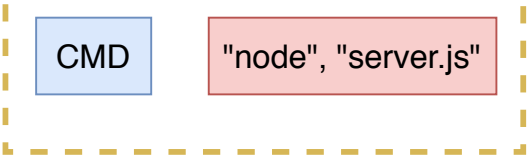


에러가 발생하는 이유

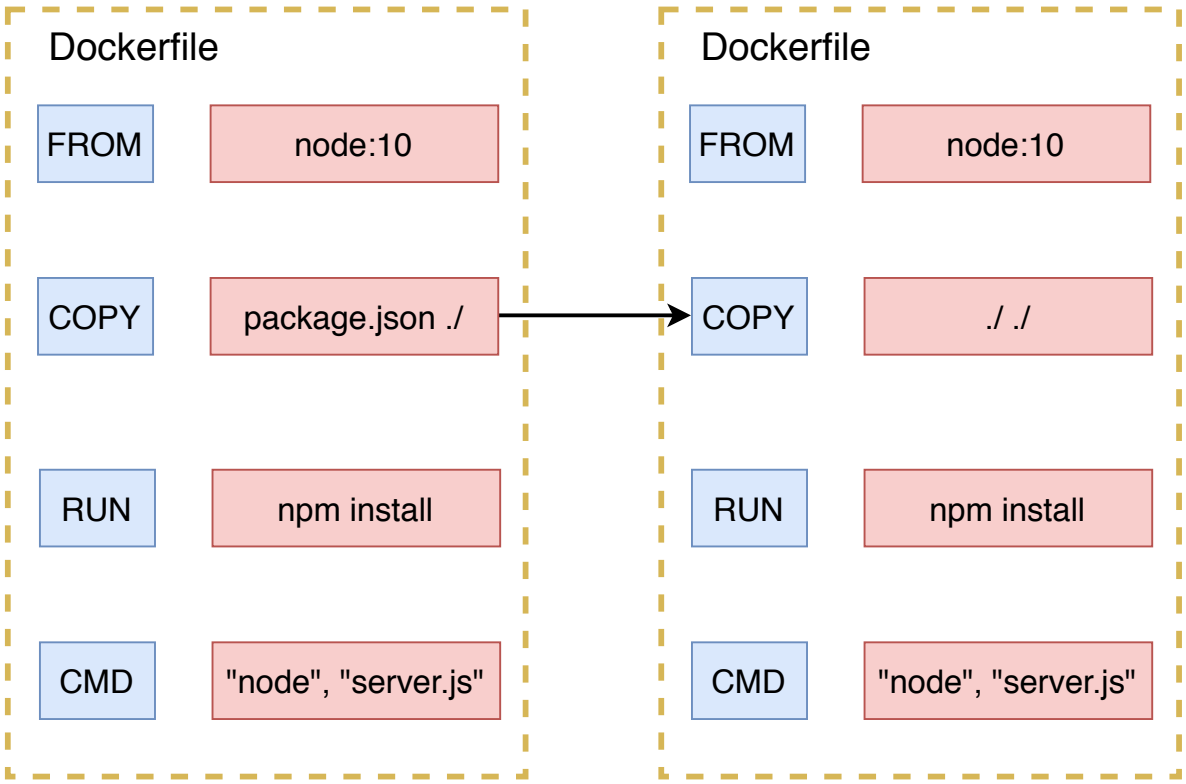





```
FROM node:10

COPY package.json ./

RUN npm install

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




```
jaewon@Jaewonui-MacBookPro node-app-docker % docker build -t  johnahn/node-app .
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM node:10
 ---> 4d698635068f
Step 2/4 : COPY package.json ./
 ---> 46ce4908e1d8
Step 3/4 : RUN npm install
 ---> Running in e7c62f6b2321
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN docker_web_app@1.0.0 No repository field.
npm WARN docker_web_app@1.0.0 No license field.

added 50 packages from 37 contributors and audited 50 packages in 1.671s
found 0 vulnerabilities

Removing intermediate container e7c62f6b2321
 ---> 00756abb244d
Step 4/4 : CMD [ "node", "server.js" ]
 ---> Running in 1a0c35325a47
Removing intermediate container 1a0c35325a47
 ---> f10e3b4500f1
Successfully built f10e3b4500f1
Successfully tagged johnahn/node-app:latest
```



생성한 이미지로 어플리케이션 실행 시 접근이 안 되는 이유 (포트 맵핑)

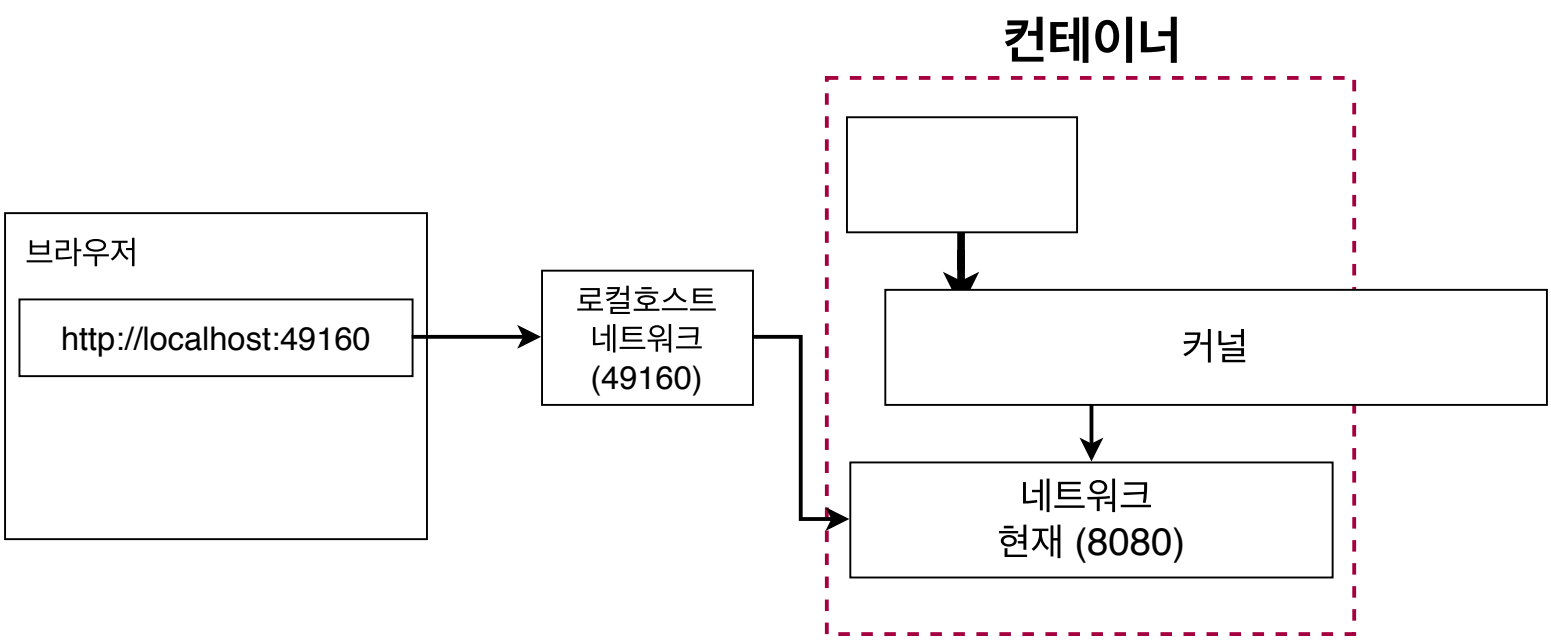
현재까지 컨테이너를 실행하기 위해 사용한 명령어

docker run 이미지 이름

앞으로 컨테이너를 실행하기 위해 사용 할 명령어

docker run -p 49160 : 8080 이미지 이름

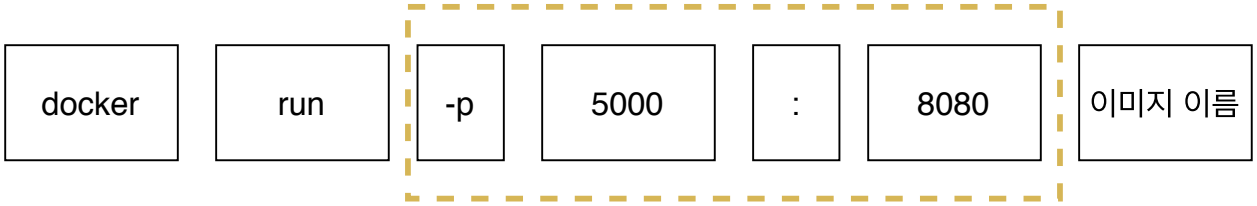
새롭게 추가된 부분



-p 49160 : 8080

```
// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(8080);
```

이 명령어를 이용해서 다시 실행



WORKING DIRECTORY 명시해주기

도커 파일에 이 WORKDIR이라는 부분을 추가해주어야 합니다.
하지만 이부분은 무엇을 위해서 추가해주어야 할까요?

```
# Create app directory
WORKDIR /usr/src/app
```

Node 이미지

시작시 실행 될 명령어

???

파일 스냅샷

home, bin, dev ...

→

현재 Node 이미지속의 Root디렉토리에는

home, bin, dev 등의 파일들이 있다.

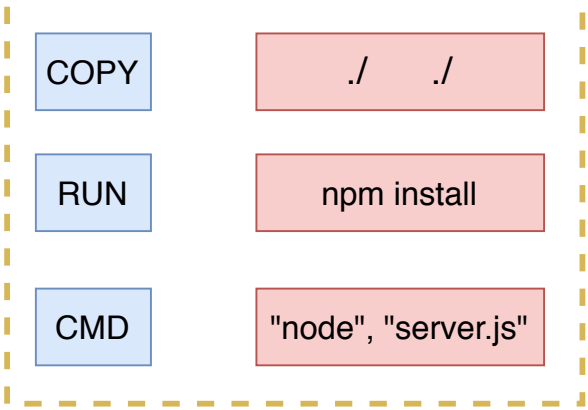
```
jaewon@Jaewonui-MacBookPro node-app-docker % docker run -it johnahn/node-app sh
# ls
Dockerfile  dev   lib    mnt           package-lock.json  root  server.js  tmp
bin         etc   lib64  node_modules  package.json       run   srv        usr
boot        home  media  opt           proc               sbin  sys        var
```

만드는 방법

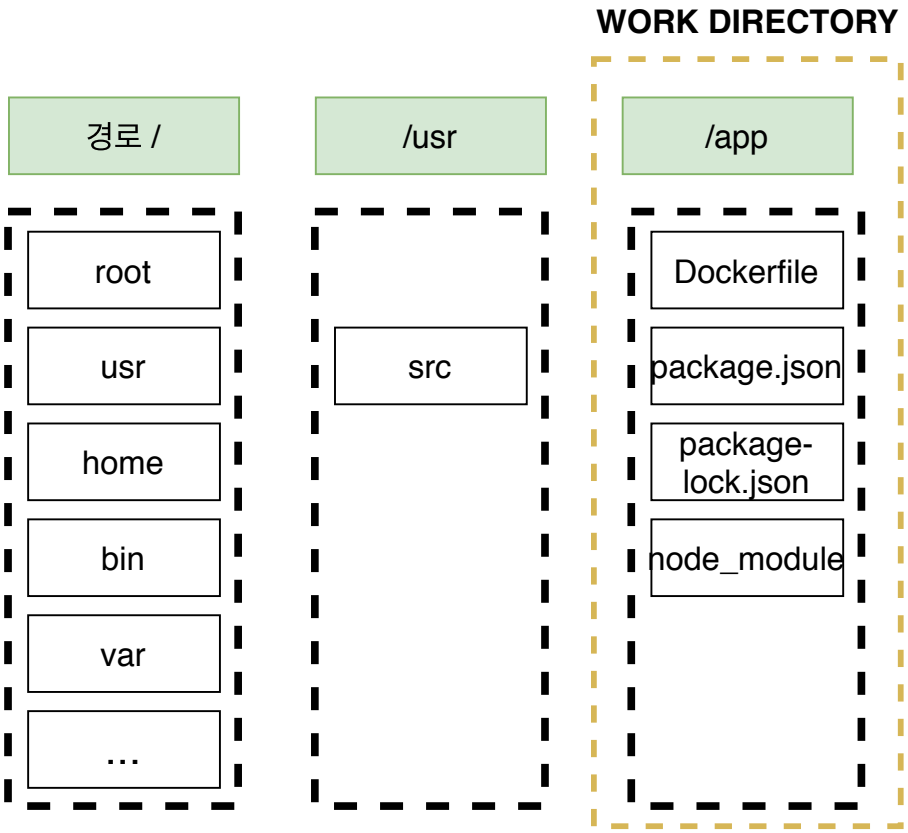
Dockerfile

FROM	node:10
WORKDIR	/usr/src/app

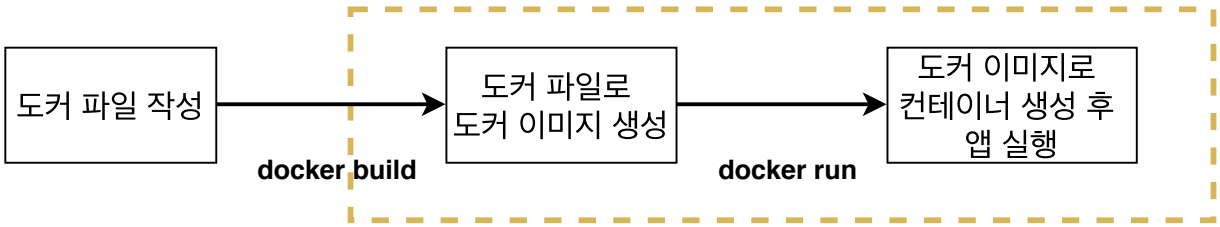
```
FROM node:10

WORKDIR /usr/src/app
```




```
COPY ./ ./

RUN npm install

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



어플리케이션 소스 변경으로 다시 빌드하는 것에 대한 문제점



실제로 이러한 방식으로 해보기

먼저 server.js에 코드 변경

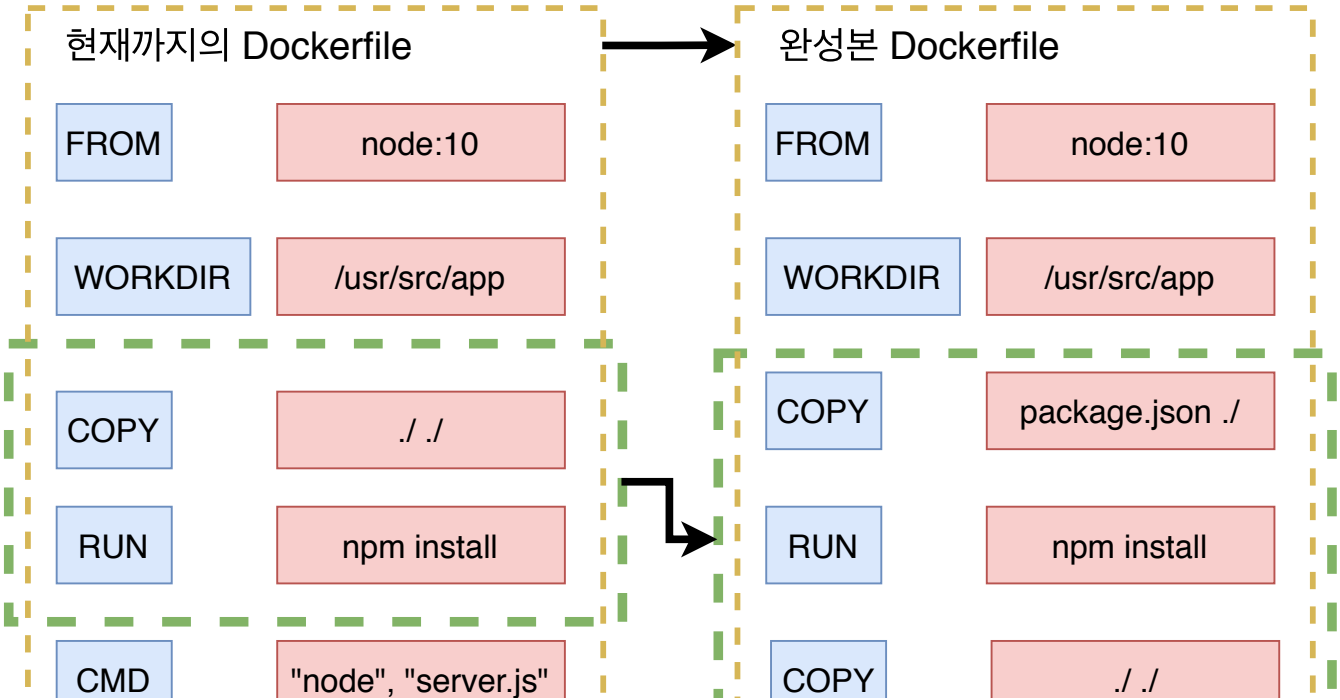
```
// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World');
});
```

```
// App
const app = express();
app.get('/', (req, res) => {
  res.send('안녕하세요');
});
```

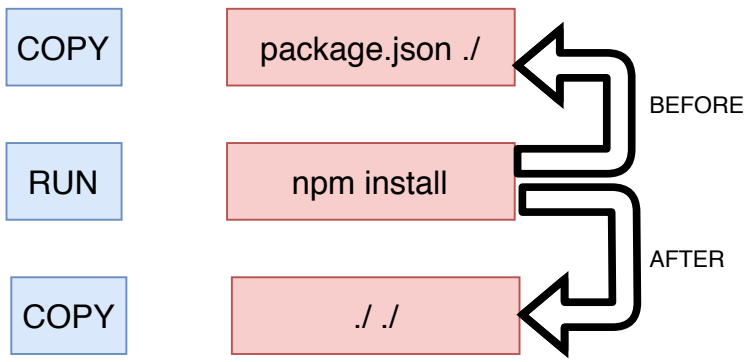
어플리케이션 소스 변경으로 재빌드시 효율적으로 하는 법

우선 이미 완성 된 Dockerfile을 살펴보겠습니다.

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

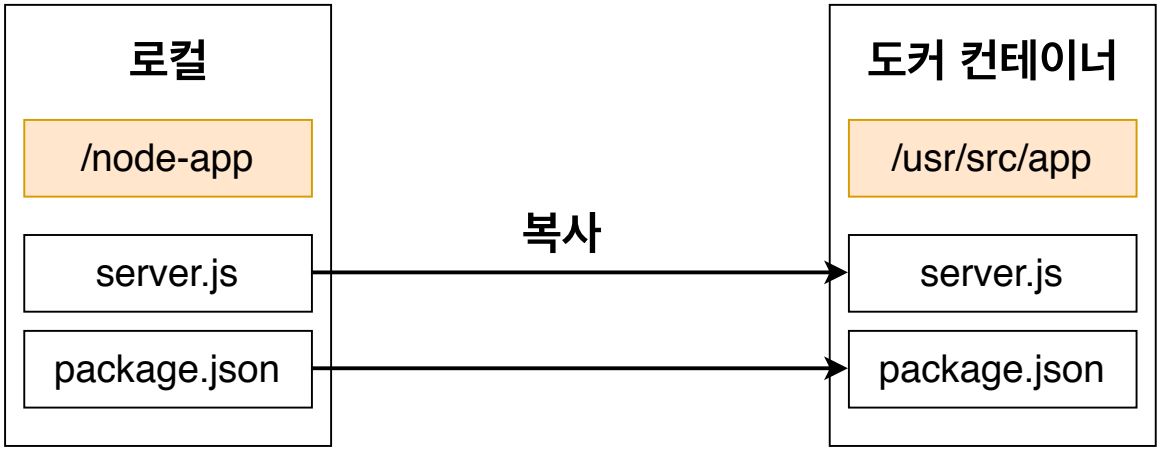



CMD
"node", "server.js"


COPY
package.json ./
BEFORE
RUN
npm install
AFTER
COPY
./ ./

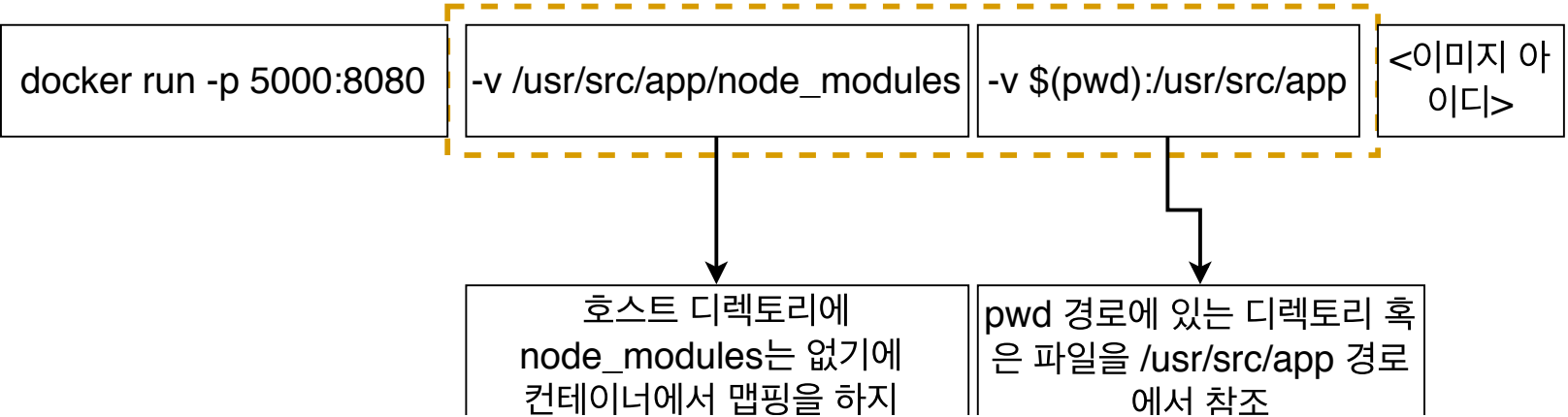
Docker Volume에 대하여

COPY



Volume

Volume 사용해서 어플리케이션 실행하는 법



말라고 하는것

Volume