

1 Lab. Nodejs Web Application Container

1. <https://www.fastify.io/docs/latest/Guides/Getting-Started/> 접속

2. 사전 Test

```
$ mkdir demo
$ cd demo
$ sudo apt install npm
$ wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
$ export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
$ [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
$ nvm install node
$ npm init
$ npm i fastify --save
```

```
$ vim app.js
// Require the framework and instantiate it

// CommonJs
const fastify = require('fastify')({
  logger: true
})

// Declare a route
fastify.get('/', function (request, reply) {
  reply.send({ hello: 'world' })
})

// Run the server!
fastify.listen(3000, '0.0.0.0', function (err, address) {
  if (err) {
    fastify.log.error(err)
    process.exit(1)
  }
  fastify.log.info(`server listening on ${address}`)
})
```

```
$ node app.js
```

```
-다른 세션에서 결과 확인
$ curl localhost:3000
{"hello":"world"}
```

3. Docker Image 생성하기

1) Dockerfile 생성하기

```
$ vim Dockerfile

# 1. nodejs 설치
FROM ubuntu:22.04
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install nodejs npm

# 2. Source File 복사
COPY . /usr/src/app

# 3. Nodejs Packages 설치
WORKDIR /usr/src/app
RUN npm install

# 4. Web Server 실행
EXPOSE 3000
CMD node app.js
```

2). dockerignore 파일 생성

```
$ vim .dockerignore
node_modules/*
```

3) Docker Image 생성

```
$ docker build -t myweb .
$ docker images

$ docker run -d -p 3000:3000 myweb

-Web Browser 확인
{"hello":"world"}

$ docker stop {{CONTAINER ID}}
```

```

84
85
86 4)Source Code 수정
87   -app.js
88     // Require the framework and instantiate it
89
90     //CommonJs
91     const fastify = require('fastify')({
92       logger: true
93     })
94
95     // Declare a route
96     fastify.get('/', function (request, reply) {
97       reply.send({ hello: 'docker world' })      <---여기 수정
98     })
99
100    // Run the server!
101    fastify.listen(3000, '0.0.0.0', function (err, address) {
102      if (err) {
103        fastify.log.error(err)
104        process.exit(1)
105      }
106      fastify.log.info(`server listening on ${address}`)
107    })
108
109
110 5)Build Again
111    $ sudo docker build -t myweb .
112    $ sudo docker images
113
114    $ docker run -d -p 3001:3000 --name myweb myweb
115
116    -Web Browser 확인
117    {"hello":"docker world"}
118
119
120
121 4. Dockerfile 최적화
122 1)Dockerfile 수정
123    # 1. nodejs 설치
124    FROM node:18      <---누군가 ubuntu위에 설치된 node:18를 사용하자.
125
126    # 2. Source File 복사
127    COPY      . /usr/src/app
128
129    # 3. Nodejs Packages 설치
130    WORKDIR   /usr/src/app
131    RUN       npm install
132
133    # 4. Web Server 실행
134    EXPOSE    3000
135    CMD       node app.js
136
137
138 2)Docker Image build
139
140    $ sudo docker build -t myweb .
141
142
143 3)한번 build 하면 cache에 남아있기 때문에 소스코드가 변경되지 않으면 Cache 그냥 사용한다. 그래서 빨리 끝난다.
144
145    $ sudo docker build -t myweb .
146    $ sudo docker build -t myweb .
147    $ sudo docker build -t myweb .
148
149    ...
150    ...
151    => CACHED [2/4] COPY      . /usr/src/app      <-----여기
152    => CACHED [3/4] WORKDIR   /usr/src/app      <-----여기
153    => CACHED [4/4] RUN       npm install      <-----여기
154
155 4)Docker가 build할 때 코드가 바뀌지 않았다면 cache를 사용하고 코드가 수정됐다면 cache를 사용하지 않는다.
156 5)Source Code 수정
157   -app.js
158     // Require the framework and instantiate it
159
160     //CommonJs
161     const fastify = require('fastify')({
162       logger: true
163     })
164
165     // Declare a route
166     fastify.get('/', function (request, reply) {
167       reply.send({ hello: 'world' })      <---여기 수정

```

```

168     })
169
170     // Run the server!
171     fastify.listen(3000, '0.0.0.0', function (err, address) {
172       if (err) {
173         fastify.log.error(err)
174         process.exit(1)
175       }
176       fastify.log.info(`server listening on ${address}`)
177     })
178

```

6)다시 build 해본다.

```

180
181     ...
182     ...
183     => [2/4] COPY      ./usr/src/app      <-----여기
184     => [3/4] WORKDIR   /usr/src/app      <-----여기
185     => [4/4] RUN       npm install        <-----여기
186     <--- [CACHED] 글자가 사라짐. 왜냐하면 cache를 사용하지 않기 때문이다.
187
188

```

7)그런데, 자세히 보면 지금 느껴지는 부분은 바로 **Nodejs Packages** 설치의 **npm install** 부분이다.

8)이것을 최적화할 수 있는데, 먼저 패키지 설치하고 소스코드 복사하면 그만큼 속도가 더 빨라진다.

```

191
192     # 1. nodejs 설치
193     FROM      node:18
194
195     # 2. 패키지 우선 복사
196     COPY      ./package* /usr/src/app/
197     WORKDIR   /usr/src/app
198     RUN       npm install
199
200     # 3. 소스코드 복사
201     COPY      . /usr/src/app
202
203     # 4. Web Server 실행
204     EXPOSE    3000
205     CMD       node app.js
206
207

```

9)다시 빌드

```

209     $ sudo docker build -t myweb .
210     $ sudo docker build -t myweb .
211     $ sudo docker build -t myweb .
212
213

```

10)dockerfile 수정 후 처음 빌드하면 처음 빌드이기 때문에 조금 시간이 걸리지만, 그 다음부터는 **npm install**까지 **cache**되어서 빌드가 빨라짐

```

215     ...
216     ...
217     ...
218     => CACHED [2/5] COPY      ./package* /usr/src/app/
219     => CACHED [3/5] WORKDIR   /usr/src/app
220     => CACHED [4/5] RUN       npm install
221     => CACHED [5/5] COPY      . /usr/src/app
222
223

```

11)그 후 **app.js**의 소스코드는 자주 바뀔 수 있기 때문에 **cache**를 사용하지 않겠지만 가장 시간이 많이 걸리는 **install npm**은 계속 **cache**를 사용하기 때문에 전체적으로 속도가 빨라진다.

12)build한 용량이 너무 크면 **alpine** 버전을 사용할 것

-현재 하나의 이미지 용량이 거의 **1.01GB**

```

228
229     # 1. nodejs 설치
230     FROM      node:12-alpine
231
232

```

-거의 **1/10**으로 줄어듦 --> **192MB**