

```
1 Lab. Nodejs Web Application Container
2
3 1. https://www.fastify.io/docs/latest/Getting-Started/ 접속
4
5
6 2. 사전 Test
7 $ mkdir demo
8 $ cd demo
9 $ sudo apt install npm
10 $ npm init
11 $ npm i fastify --save
12
13 $ vim app.js
14 // Require the framework and instantiate it
15 const fastify = require('fastify')({
16   logger: true
17 })
18
19 // Declare a route
20 fastify.get('/', function (request, reply) {
21   reply.send({ hello: 'world' })
22 })
23
24 // Run the server!
25 fastify.listen(3000, function (err, address) {
26   if (err) {
27     fastify.log.error(err)
28     process.exit(1)
29   }
30   fastify.log.info(`server listening on ${address}`)
31 })
32
33 $ node app.js
34
35 -다른 세션에서 결과 확인
36 $ curl localhost:3000
37 {"hello":"world"}
38
39
40
41 3. Docker Image 생성하기
42 1)app.js 수정하기
43
44 $ vim app.js
45 // Require the framework and instantiate it
46 const fastify = require('fastify')({
47   logger: true
48 })
49
50 // Declare a route
51 fastify.get('/', function (request, reply) {
52   reply.send({ hello: 'world' })
53 })
54
55 // Run the server!
56 fastify.listen(3000, '0.0.0.0', function (err, address) { <---여기 수정
57   if (err) {
58     fastify.log.error(err)
59     process.exit(1)
60   }
61   fastify.log.info(`server listening on ${address}`)
62 })
63
64
65 2)Dockerfile 생성하기
66 # 1. nodejs 설치
67 FROM ubuntu:20.04
```

```

68     RUN      apt-get update
69     RUN      DEBIAN_FRONTEND=noninteractive apt-get -y install nodejs npm
70
71     # 2. Source File 복사
72     COPY      . /usr/src/app
73
74     # 3. Nodejs Packages 설치
75     WORKDIR   /usr/src/app
76     RUN      npm install
77
78     # 4. Web Server 실행
79     EXPOSE    3000
80     CMD       node app.js
81
82
83 3).dockerignore 파일 생성
84     node_modules/*
85
86
87 4)Docker Image 생성
88     $ sudo docker build -t myweb .
89     $ sudo docker images
90
91     $ sudo docker run -d -p 3000:3000 myweb
92
93     -Web Browser 확인
94     {"hello":"world"}
95
96     $ sudo docker stop {{CONTAINER ID}}
97
98
99 5)Source Code 수정
100     -app.js
101         // Require the framework and instantiate it
102         const fastify = require('fastify')({
103             logger: true
104         })
105
106         // Declare a route
107         fastify.get('/', function (request, reply) {
108             reply.send({ hello: 'docker world' })
109         })
110
111         // Run the server!
112         fastify.listen(3000, '0.0.0.0', function (err, address) {
113             if (err) {
114                 fastify.log.error(err)
115                 process.exit(1)
116             }
117             fastify.log.info(`server listening on ${address}`)
118         })
119
120
121 6)재 Build
122     $ sudo docker build -t myweb .
123     $ sudo docker images
124
125     $ docker run -d -p 3001:3000 --name myweb myweb
126
127     -Web Browser 확인
128     {"hello":"docker world"}
129
130
131
132 4. Dockerfile 최적화
133     1)Dockerfile 수정
134         # 1. nodejs 설치

```

FROM node:12 <---누군가 ubuntu위에 설치된 node:12를 사용하자.

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) Docker Image build

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

3) 한번 build 하면 cache에 남아있기 때문에 소스코드가 변경되지 않으면 Cache 그냥 사용한다. 그래서 빨리 끝난다.

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

```
...  
...  
Step 2/6 : COPY . /usr/src/app  
---> Using cache <---바로 이 부분
```

4) Docker가 build할 때 코드가 바뀌지 않았다면 cache를 사용하고 코드가 수정됐다면 cache를 사용하지 않는다.

5) Source Code 수정

```
-app.js  
// Require the framework and instantiate it  
const fastify = require('fastify')({  
  logger: true  
})  
  
// Declare a route  
fastify.get('/', function (request, reply) {  
  reply.send({ hello: 'world' }) <---여기 수정  
})  
  
// 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}`)  
})
```

6) 다시 build 해본다.

```
...  
...  
Step 2/6 : COPY . /usr/src/app  
---> Using cache <---바로 이 부분이 없어졌다. 왜냐하면 cache를 사용하지 않기 때문이다.
```

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

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

1. nodejs 설치
FROM node:12

```
# 2. 패키지 우선 복사
COPY      ./package* /usr/src/app/
WORKDIR   /usr/src/app
RUN        npm install
COPY      . /usr/src/app

# 3. 소스코드 복사
COPY      . /usr/src/app

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

9)다시 빌드

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

10)dockerfile 수정 후 처음 빌드하면 처음 빌드이기 때문에 조금 시간이 걸리지만, 그 다음부터는 npm install까지 cache되어서 빌드가 빨라짐

```
...
...
Step 2/8 : COPY      ./package*      /usr/src/app/
---> Using cache

Step 3/8 : COPY      WORKDIR   /usr/src/app
---> Using cache

Step 4/8 : COPY      RUN    npm install
---> Using cache
```

11)그 후 app.js의 소스코드는 자주 바뀔 수 있기 때문에 cache를 사용하지 않겠지만 가장 시간이 많이 걸리는 install npm은 계속 cache를 사용하기 때문에 전체적으로 속도가 빨라진다.

12)build한 용량이 너무 크면 alpine 버전을 사용할 것
-현재 하나의 이미지 용량이 거의 925MB

```
# 1. nodejs 설치
FROM      node:12-alpine

-거의 1/10으로 줄어듦 --> 96.4MB
```