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}}
```

4)Source Code 수정
-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: 'docker 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}`)
})
```

5)Build Again

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

$ docker run -d -p 3001:3000 --name myweb myweb
```

-Web Browser 확인

```
{"hello":"docker world"}
```

4. Dockerfile 최적화

1)Dockerfile 수정

```
# 1. nodejs 설치
FROM  node:18      <---누군가 ubuntu위에 설치된 node:18를 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 .

...
...
 => CACHED  [2/4] COPY    . /usr/src/app          <-----여기
 => CACHED  [3/4] WORKDIR    /usr/src/app         <-----여기
 => CACHED  [4/4] RUN       npm install           <-----여기
```

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

//CommonJs
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.

```
...
...
=> [2/4] COPY    . /usr/src/app          <-----여기
=> [3/4] WORKDIR    /usr/src/app        <-----여기
=> [4/4] RUN       npm install          <-----여기
<--- [CACHED] 글자가 사라짐. 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:18

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

# 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 길어지지만, 그 다음부터는 npm install까지 cache되어서 빌드가 빨라짐

```
...
...
 => CACHED [2/5] COPY        ./package* /usr/src/app/
 => CACHED [3/5] WORKDIR      /usr/src/app
 => CACHED [4/5] RUN          npm install
 => CACHED [5/5] COPY        .  /usr/src/app
```

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 1.01GB

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

-거의 1/10으로 줄어듦 --> 192MB