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/v0.39.7/install.sh | bash
$ export NVM_DIR="$HOME/.nvm"
$ [ -s "$NVM_DIR/nvm.sh" ] && ₩. "$NVM_DIR/nvm.sh"  # This loads nvm
$ [ -s "$NVM_DIR/bash_completion" ] && ₩. "$NVM_DIR/bash_completion"  # This loads nvm bash_completion
$ 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

    FROM        node:14
    WORKDIR     /app
    COPY        package.json .
    RUN         npm install
    COPY        . .
    EXPOSE      3000

    ENV         ADDRESS=0.0.0.0 PORT=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

FROM node:18   <----여기만 수정

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 [2/5] WORKDIR /app              <---여기 주목
=> CACHED [3/5] COPY package.json .
=> CACHED [4/5] RUN npm install
=> CACHED [5/5] COPY . .
```

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.

```
...
...
```

7)build한 용량이 너무 크면 alpine 버전을 사용할 것

-현재 하나의 이미지 용량이 거의 1.11GB

```
FROM        node:18-alpine        <----여기만 수정
```

-거의 1/10으로 줄어듦 --> 147MB