

도커를 이용하여 Node.js 서버 만들기

이영호

국립목포대학교 컴퓨터공학과
youngho.lee@gmail.com

September 7, 2024

Contents

1	시작하기	2
2	Node.js 설치	2
3	app.js 작성	3
4	Dockerfile 생성 및 빌드	7
5	컨테이너 실행	9
A	Docker 명령어	11
B	Dockerfile 작성법	12

1 시작하기

이번 과정은 우분투에 node.js를 설치하고 간단한 자바 스크립 서버를 만든다. 그리고 dockerfile을 만들어 도커 이미지를 만들어 사용하는 것이다. 실습을 위해서 자신의 컴퓨터의 OS를 사용할 수도 있지만, 가상 머신을 사용하기 바란다. 윈도우면 virtual box, 맥OS는 UTM을 다운받아 설치 한 후 우분투를 설치한다. 우분투 설치후 sudo apt-get update를 실행한다.

반드시 필요한건 아니지만 visual studio code 를 우분투에 설치하여 사용하여도 좋다.

2 Node.js 설치

먼저 Node.js가 설치되어 있지 않다면 사이트(<https://nodejs.org/en/download/package-manager>)에서 설치 방법을 확인하고 따라해주세요. 여러가지 방법이 있습니다. 네이버나 다른 검색엔진에서 한글로 된 친절한 설명을 찾아 읽어 보아도 좋습니다.

우분투에서 폴더를 하나 생성하고, test.js라는 이름으로 파일을 하나 생성 후 다음 코드를 작성합니다. 리눅스에서 폴더를 생성하는 명령어는 mkdir입니다. 그리고 파일 작성은 vi 에디터나 비주얼 스튜디오 코드를 이용하세요.

```

1  'use strict';
2
3  var printFunc = function(name) {
4      console.log(name);
5  }
6
7  printFunc('vscode & node.js');

```

Listing 1: test.js 예제

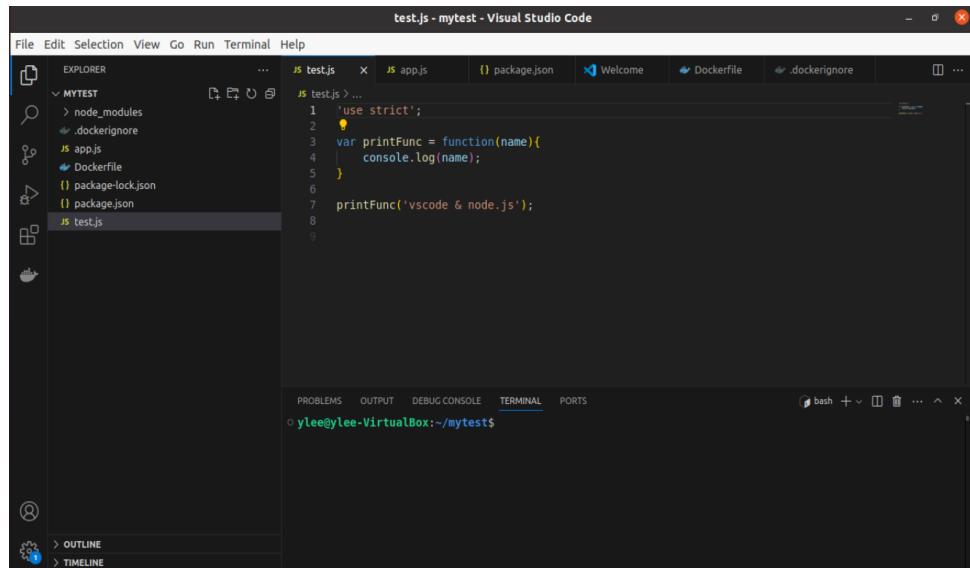


Figure 1: 비주얼 스튜디오 코드에서 test.js를 작성하는 스크린샷

F5를 눌러 실행시켜주면 다음과 같은 화면이 뜨는데 Node.js를 눌러 실행시켜주면 됩니다. 정상 작동이 되면 Node.js 개발환경이 준비되었습니다. 그럼 1처럼 비주얼 스튜디오 코드를 이용하여 코드를 작성한 후 F5를 누릅니다. 그리고 웹브라우저를 실행하여 <http://localhost:3000> 주소창에 입력하면 그림 2처럼 자바스크립트로 작성한 홈페이지를 볼 수 있습니다.

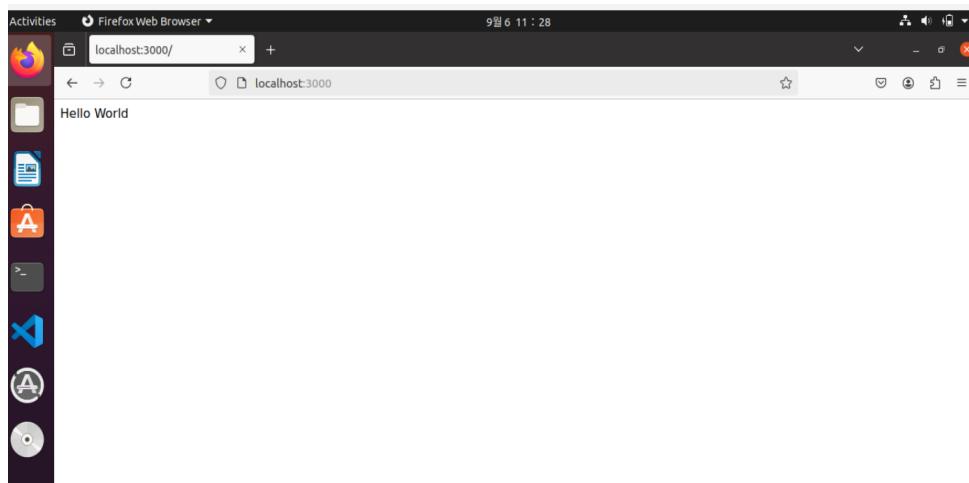


Figure 2: test.js를 nodejs 서버로 실행한 결과

3 app.js 작성

자, 이제 테스트를 마쳤으니 본격적으로 node js 서버를 만들어 봅시다.

```

1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => res.send('Hello World'));
5 app.get('/home', (req, res) => res.send('My name is Youngho Lee'));
6
7 app.listen(3000, () => {
8   console.log('My REST API running on port 3000!');
9 });

```

Listing 2: app.js 예제

작성 후, 터미널에서 해당 폴더로 이동 후 명령어를 입력합니다. VS Code일 경우 Ctrl + ` 또는 TERMINAL을 누르면 해당 폴더로 이동된 터미널 사용이 가능합니다.

```
$ npm init
```

Listing 3: npm 초기화 명령어

npm init 명령어를 입력하고 나면, 패키지명을 입력하라는 창이 나옵니다. 원하는 패키지 명을 입력하는 엔터키를 누릅니다. 완료되면 패키지 정보를 담고 있는 package.json 파일이 생성됩니다. 이 파일은 폴더에 있습니다. 중요한 파일입니다. 아래 실행 결과와 그림3을 보면 이해될 겁니다.

```

1 ylee@ylee-VirtualBox:~/mytest$ npm init
2 This utility will walk you through creating a package.json file.
3 It only covers the most common items, and tries to guess sensible defaults.
4
5 See 'npm help json' for definitive documentation on these fields
6 and exactly what they do.
7
8 Use 'npm install <pkg>' afterwards to install a package and
9 save it as a dependency in the package.json file.
10
11 Press ^C at any time to quit.
12 package name: (mytest) my-nodejs-docker-v0.1
13 version: (1.0.0)
14 description:
15 entry point: (app.js)
16 test command:
17 git repository:
18 keywords:

```

```

19 author:
20 license: (ISC)
21 About to write to /home/yLee/mytest/package.json:
22
23 {
24   "name": "my-nodejs-docker-v0.1",
25   "version": "1.0.0",
26   "description": "",
27   "main": "app.js",
28   "dependencies": {
29     "express": "^4.19.2"
30   },
31   "devDependencies": {},
32   "scripts": {
33     "test": "echo \"Error: no test specified\" && exit 1"
34   },
35   "author": "",
36   "license": "ISC"
37 }
38
39 Is this OK? (yes)
yLee@yLee-VirtualBox:~/mytest$
```

Listing 4: npm init 실행 예제

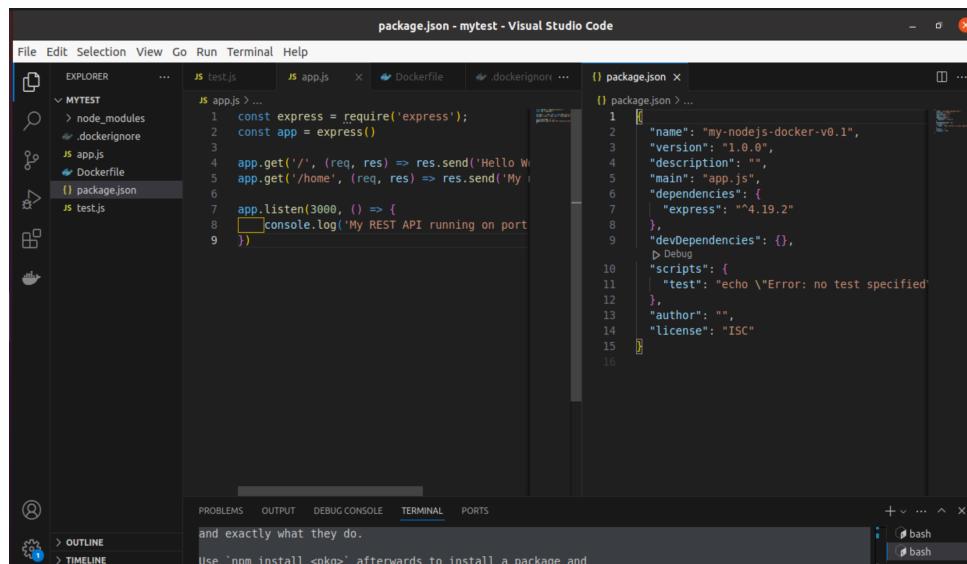


Figure 3: npm init 실행한 결과

그 다음, Express를 설치해줍니다. express가 무엇인지는 다음 홈페이지를 참고하세요. (<https://expressjs.com/ko/starter/installing.html>) 완료되면 package.json 파일에 Express가 추가됩니다.

```
1 $ npm install --save express
```

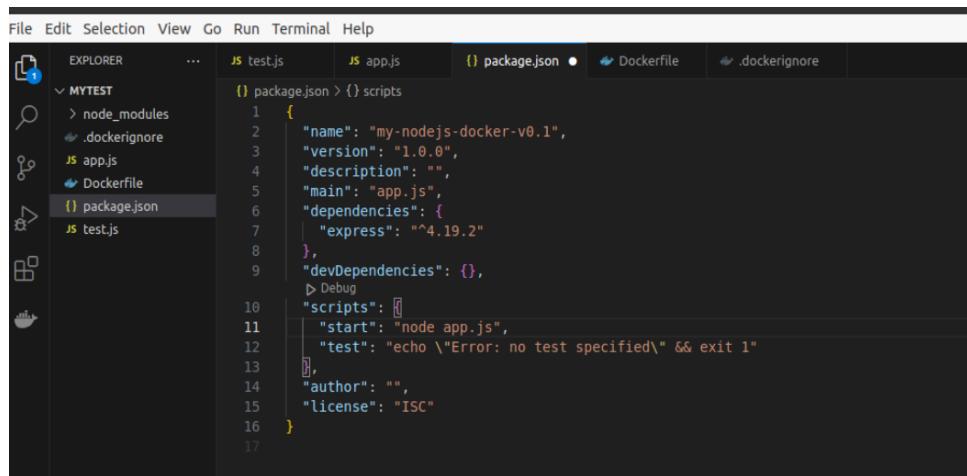
Listing 5: Express 설치

package.json 파일을 열어 시작 앱을 그림4 처럼 app.js로 지정합니다.

```
1 npm run start
```

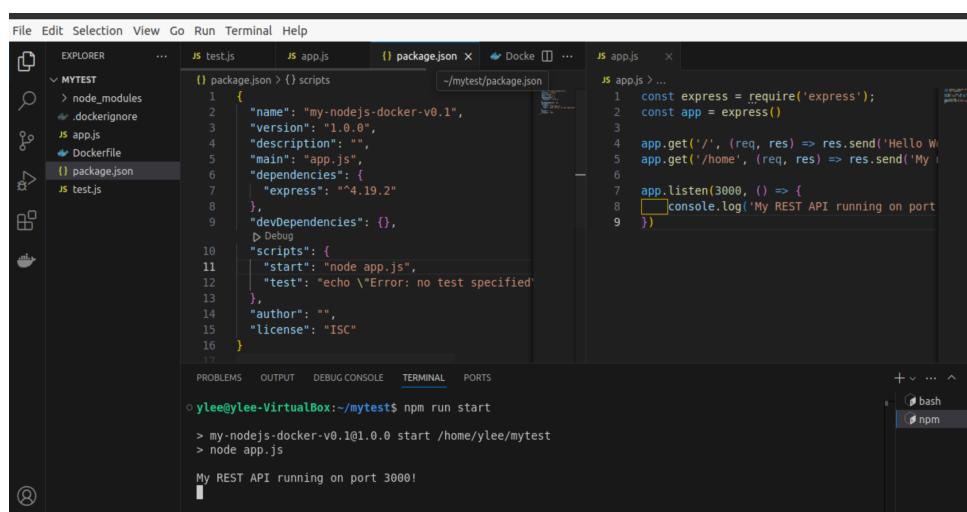
Listing 6: Node.js 서버 시작

실행 후, 크롬 브라우저로 접속 후, localhost:3000과 localhost:3000/home 으로 들어가면, app.js 화면이 띄워지게 됩니다. 그림4와 그림6b를 보면 우리가 만든 서버가 잘 동작함을 알 수 있습니다.



```
File Edit Selection View Go Run Terminal Help
EXPLORER ... JS test.js JS app.js {} package.json Dockerfile .dockerignore
MYTEST
node_modules
.dockerignore
JS app.js
Dockerfile
{} package.json
JS test.js
package.json > {} scripts
1 {
  "name": "my-nodejs-docker-v0.1",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "dependencies": {
    "express": "^4.19.2"
  },
  "devDependencies": {},
  "scripts": {
    "start": "node app.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
17
```

Figure 4: package.json 파일에 start node app.js를 추가합니다.



```
File Edit Selection View Go Run Terminal Help
EXPLORER ... JS test.js JS app.js {} package.json Dockerfile ...
MYTEST
node_modules
.dockerignore
JS app.js
Dockerfile
{} package.json
JS test.js
package.json > {} scripts
1 {
  "name": "my-nodejs-docker-v0.1",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "dependencies": {
    "express": "^4.19.2"
  },
  "devDependencies": {},
  "scripts": {
    "start": "node app.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
17
app.js > ...
1 const express = require('express');
2 const app = express()
3
4 app.get('/', (req, res) => res.send('Hello World'))
5 app.get('/home', (req, res) => res.send('My REST API'))
6
7 app.listen(3000, () => {
8   console.log('My REST API running on port 3000')
9 })
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

y lee@y lee-VirtualBox:~/mytest\$ npm run start

> my-nodejs-docker-v0.1@1.0.0 start /home/y lee/mytest

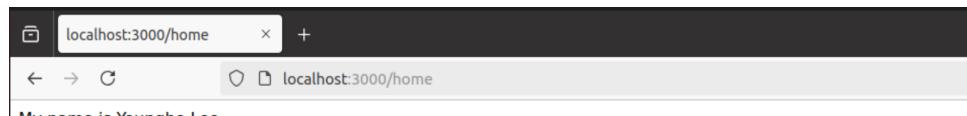
> node app.js

My REST API running on port 3000!

Figure 5: terminal에서 npm run start를 실행하면 app.js 서버가 실행됩니다.



Hello World

(a) `http://localhost:3000` 접속

My name is Youngho Lee

(b) `http://localhost:3000/home` 접속

Figure 6: 서버에 접속한 화면

4 Dockerfile 생성 및 빌드

비주얼 스튜디오 코드의 Extension에서 Docker 검색해서 Docker 플러그인을 설치합니다. 그럼 /refig:dockerextension 을 보면 왼쪽의 탭에서 확장을 선택하여 검색하는 것을 볼 수 있습니다.



Figure 7: 비주얼 스튜디오 코드에서 docker extension을 찾아 설치

이제 같은 폴더에서 비주얼 스튜디오 코드에서 Dockerfile을 생성하고 다음과 같이 작성해 줍니다.

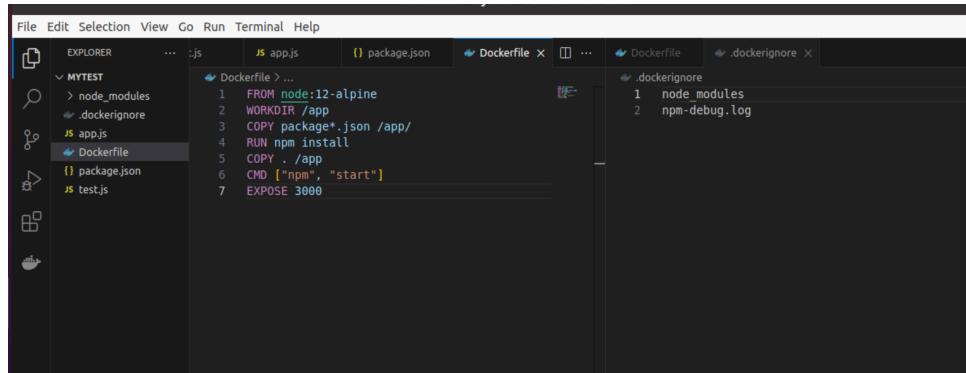


Figure 8: dodkerfile과 .dockerextension 파일

```
1 FROM node:12-alpine
2 WORKDIR /app
3 COPY package*.json /app
4 RUN npm install
5 COPY . /app
6 CMD [ "npm", "start" ]
7 EXPOSE 3000
```

Listing 7: Dockerfile 예제

- Docker Hub에 있는 `node:12-alpine` 이미지 사용
- 이미지 안에 애플리케이션 코드를 넣기 위한 디렉터리 생성. 애플리케이션의 작업 디렉터리가 됩니다.
- `node:12` 이미지에 Node.js와 npm은 설치되어 있으므로 npm 바이너리로 앱 의존성만 설치합니다.
- npm 설치 (RUN은 새로운 레이어 위에서 명령어를 실행, 주로 패키지 설치용)
- Docker 이미지 안에 앱의 소스코드를 넣기 위함

- CMD는 도커가 실행될 때 실행되는 명령어를 정의합니다.
- 3000번 포트로 실행

이는 Dockerfile에 위 스텝들을 캐싱해 놓는다고 생각하시면 됩니다. 애플리케이션을 수정하거나 rebuild 할 때마다 위 Dockerfile은 위 스텝들을 다시 진행하지 않게 해줍니다.

빌드 전, .dockerignore 파일을 생성하여 Docker image의 파일 시스템의 node_modules 디렉터리가 현재 로컬 작업 디렉터리의 node_modules 디렉터리로 덮어지지 않도록 합니다.

```

1  $ sudo docker build -t my-nodejs-docker-v0.1 .
2  [sudo] password for ylee:
3  [+] Building 1.6s (10/10) FINISHED
4
5      docker:
6          default
7      => [internal] load build definition from Dockerfile
8          0.0s
9
10     => => transferring dockerfile: 155B
11
12     0.0s
13     => [internal] load metadata for docker.io/library/node:12-alpine
14             1.5s
15     => [internal] load .dockerignore
16
17     0.0s
18     => => transferring context: 66B
19
20     0.0s
21     => [1/5] FROM docker.io/library/node:12-alpine@sha256:
22         d4b15b3d48f42059a15bd659be60afe21762aae9d6cbea6f1244408 0.0s
23     => [internal] load build context
24
25     0.0s
26     => => transferring context: 147B
27
28     0.0s
29     => CACHED [2/5] WORKDIR /app
30
31     0.0s
32     => CACHED [3/5] COPY package*.json /app/
33             0.0s
34     => CACHED [4/5] RUN npm install
35
36     0.0s
37     => CACHED [5/5] COPY . /app
38
39     0.0s
40     => exporting to image
41
42     0.0s
43     => => exporting layers
44
45     0.0s
46     => => writing image sha256:41
47         a33e535d06a14739f19c83c75f6b93ad7464e193c4700901a448b89f5d3a60
48             0.0s
49     => => naming to docker.io/library/my-nodejs-docker-v0.1

```

Listing 8: Docker 이미지 빌드

그리고 Container Image를 도커로 빌드합니다. 이때 도커가 실행되고 있어야 하니, Docker QuickStart Terminal을 연 상태에서 빌드해주어야 합니다.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-nodejs-docker-v0.1	latest	41a33e535d06	12 minutes ago	95.6MB

4	<none>	<none>	67536ede950e	16 hours ago	94MB
5	mynewserver	latest	3372a92d9e3f	10 days ago	134MB
6	mynginx	latest	3ee48dfc2712	10 days ago	249MB
7	nginx	latest	5ef79149e0ec	3 weeks ago	188MB
8	busybox	latest	65ad0d468eb1	15 months ago	4.26MB
9	hello-world	latest	d2c94e258dcb	16 months ago	13.3kB
10	prakhar1989/static-site	latest	f01030e1dcf3	8 years ago	134MB

Listing 9: 도커 이미지 조회

방금 빌드한 이미지를 확인합니다.

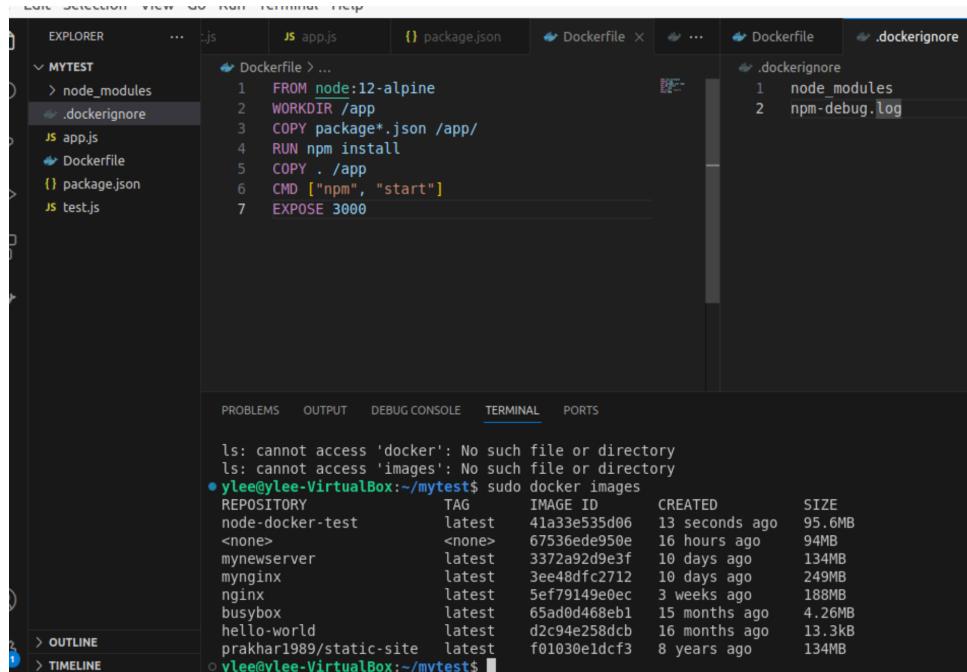


Figure 9: 만들어진 도커 이미지

5 컨테이너 실행

다음 명령어로 Docker Container로 빌드된 이미지를 실행시킵니다. -p 5000:3000의 의미는 Host 포트 5000으로 들어오는 트래픽을 Container의 포트 3000으로 포워딩 시키는 것입니다.

app.js에서 작성한 로그 My REST API running on port 3000! 문구와 함께 app.js가 실행되었다는 화면을 볼 수 있습니다.

```

1 $ sudo docker run -p 5000:3000 my-nodejs-docker-v0.1
2
3 > my-nodejs-docker-v0.1@1.0.0 start /app
4 > node app.js
5
6 My REST API running on port 3000!

```

Listing 10: Docker 컨테이너 실행

지금까지 여러분이 실행한 명령어와 옵션을 극히 일부입니다. 더 많은 옵션과 명령어가 있습니다. 하나하나 찾아보면서 실행해 보시기 바랍니다. 참고로 명령어에 '-h' 옵션을 붙이면 모든 명령어에 필요한 정보를 얻을 수 있습니다.

```

1 $ docker -h
2 Flag shorthand -h has been deprecated, use --help
3
4 Usage: docker [OPTIONS] COMMAND
5

```

```
6 A self-sufficient runtime for containers
7
8 Common Commands:
9   run      Create and run a new container from an image
10  exec     Execute a command in a running container
11  ps       List containers
12  build    Build an image from a Dockerfile
13  pull     Download an image from a registry
14  push     Upload an image to a registry
15  images   List images
```

Listing 11: Docker 명령어 도움말

A Docker 명령어

도커 기본 명령어

- docker --version : 도커 버전을 확인합니다.
- docker info : 도커 시스템에 대한 자세한 정보를 출력합니다.
- docker help : 도커 명령어에 대한 도움말을 표시합니다.

이미지 관련 명령어

- docker pull [이미지 이름] : 도커 허브에서 이미지를 다운로드합니다.
- docker images : 다운로드된 모든 이미지를 나열합니다.
- docker rmi [이미지 ID] : 특정 이미지를 삭제합니다.
- docker build -t [이미지 이름] . : 현재 디렉토리의 Dockerfile을 사용해 이미지를 빌드합니다.

컨테이너 관련 명령어

- docker run [옵션] [이미지 이름] : 컨테이너를 생성하고 실행합니다.
 - 예: docker run -it --name my_container ubuntu /bin/bash
- docker ps : 현재 실행 중인 모든 컨테이너를 나열합니다.
- docker ps -a : 중지된 컨테이너를 포함한 모든 컨테이너를 나열합니다.
- docker stop [컨테이너 ID] : 실행 중인 컨테이너를 중지합니다.
- docker start [컨테이너 ID] : 중지된 컨테이너를 다시 시작합니다.
- docker restart [컨테이너 ID] : 컨테이너를 재시작합니다.
- docker rm [컨테이너 ID] : 특정 컨테이너를 삭제합니다.
- docker exec -it [컨테이너 이름] /bin/bash : 실행 중인 컨테이너에 접속합니다.

네트워크 관련 명령어

- docker network ls : 모든 네트워크를 나열합니다.
- docker network create [네트워크 이름] : 새로운 네트워크를 생성합니다.
- docker network connect [네트워크 이름] [컨테이너 이름] : 컨테이너를 네트워크에 연결합니다.
- docker network disconnect [네트워크 이름] [컨테이너 이름] : 컨테이너를 네트워크에서 분리 합니다.

볼륨 관련 명령어

- docker volume ls : 모든 볼륨을 나열합니다.
- docker volume create [볼륨 이름] : 새로운 볼륨을 생성합니다.
- docker volume rm [볼륨 이름] : 특정 볼륨을 삭제합니다.

로그 및 상태 확인 명령어

- docker logs [컨테이너 이름] : 컨테이너의 로그를 출력합니다.
- docker top [컨테이너 이름] : 컨테이너 내부의 프로세스를 나열합니다.
- docker stats : 모든 실행 중인 컨테이너의 실시간 리소스 사용량을 표시합니다.

도커 컴포즈 관련 명령어

- docker-compose up : 정의된 모든 서비스(컨테이너)를 시작합니다.
- docker-compose down : 정의된 모든 서비스를 중지하고 네트워크를 삭제합니다.
- docker-compose ps : 컴포즈로 실행 중인 서비스(컨테이너)를 나열합니다.

B Dockerfile 작성법

Dockerfile이란?

Dockerfile은 도커 이미지를 생성하기 위한 명령어들의 집합을 포함한 텍스트 파일입니다. Dockerfile은 특정 이미지의 빌드 과정을 자동화하고 일관된 환경을 제공하기 위해 사용됩니다.

Dockerfile 작성 방법

- FROM : 베이스 이미지를 설정하는 명령어로, Dockerfile의 첫 번째 명령어로 사용됩니다. 예: FROM ubuntu:latest
- LABEL : 이미지에 메타데이터(작성자 정보 등)를 추가합니다.
- WORKDIR : 작업 디렉토리를 설정하며, 이후 명령어들은 설정된 디렉토리에서 실행됩니다.
- COPY : 파일이나 디렉토리를 호스트에서 컨테이너로 복사합니다. 예: COPY . /app
- RUN : 컨테이너 빌드 중에 실행할 명령어를 지정합니다. 패키지 설치 등에 사용됩니다. 예: RUN apt-get update && apt-get install -y python3
- EXPOSE : 컨테이너가 사용할 포트를 명시합니다.
- CMD : 컨테이너가 시작될 때 실행할 명령어를 지정합니다. 이 명령어는 Dockerfile에서 한 번만 사용되며, 일반적으로 컨테이너의 기본 프로세스를 지정합니다.
- ENTRYPOINT : CMD와 유사하게 컨테이너 실행 시 실행할 명령어를 지정하지만, 항상 실행됩니다. CMD 와 함께 사용될 수 있습니다.

Dockerfile 예제

아래는 간단한 Python 애플리케이션을 위한 Dockerfile의 예제입니다.

```

1  FROM python:3.9
2  WORKDIR /app
3  COPY requirements.txt .
4  RUN pip install --no-cache-dir -r requirements.txt
5  COPY . .
6  EXPOSE 5000
7  CMD ["python", "app.py"]

```

Listing 12: Python Dockerfile 예제

설명:

- FROM python:3.9 : 베이스 이미지로 Python 3.9 이미지를 설정합니다.
- WORKDIR /app : 컨테이너 내의 작업 디렉토리를 /app으로 설정합니다.
- COPY requirements.txt . : 호스트의 requirements.txt 파일을 컨테이너의 현재 작업 디렉토리로 복사합니다.
- RUN pip install --no-cache-dir -r requirements.txt : requirements.txt 파일에 명시된 Python 패키지를 설치합니다.
- COPY . . : 호스트의 현재 디렉토리의 모든 파일을 컨테이너의 작업 디렉토리로 복사합니다.
- EXPOSE 5000 : 컨테이너가 사용할 포트 5000을 명시합니다.
- CMD ["python", "app.py"] : 컨테이너가 시작될 때 python app.py 명령어를 실행합니다.