


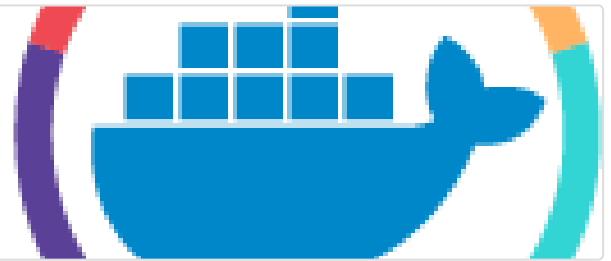
2.1 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

도커 설치

Install Docker Engine

Docker Engine is available on a variety of Linux platforms, macOS and Windows 10 through Docker Desktop, and as a static binary installation. Find your preferred operating system below. Docker provides .deb and .rpm packages from the following Linux distributions and architectures: Note While the

 <https://docs.docker.com/engine/install/>



여기서 설치가 가능함

- `brew install --cask docker` 로도 설치가 가능하다.

설치 완료되면 도커 클라이언트 실행파일로 다양한 도커 명령을 실행할 수 있다.

- 예를 들어, 도커 허브에 있는 이미지 풀하거나 실행 가능

설치했으면 `docker --version` 을 통해 설치가 잘 되었는지 확인하자.

도커를 사용한 컨테이너 이미지 생성, 실행, 공유

간단한 node.js 애플리케이션

```
// app.js
const http = require('http');
const os = require('os');

console.log("Kubia server starting ...");

var handler = function(request, response) {
  console.log("Received request from " + request.connection.remoteAddress);
  response.writeHead(200);
  response.end("You've hit " + os.hostname + "\n");
};

var www = http.createServer(handler);
www.listen(8080);
```

- 포트 8080로 요청하면 200 OK와 명시된 텍스트를 반환하는 애플리케이션

Dockerfile 생성

애플리케이션을 이미지로 패키징하기 위해선 Dockerfile이 필요하다.

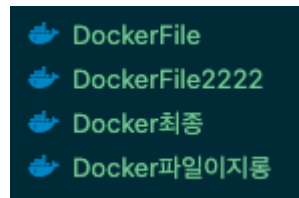
- 도커가 이미지를 생성하기 위한 지시사항이 담긴 파일
- 패키징 할 파일과 같은 디렉토리에 위치해야함

```
# 사용할 컨테이너 이미지 정의
FROM node:7

# 로컬 파일을 루트 파일로 추가
ADD app.js /app.js

# 이미지 실행되었을 때, 수행할 명령어 "node app.js"
ENTRYPOINT ["node", "app.js"]
```

- Dockerfile 생성 시, 중요한 점이 확장자도 없이 파일명이 정확히 `Dockerfile` 이어야 한다는 것
 - `dockerfile`, `Dockerfile.txt`, `DockerFile`, `Docker파일` 안됨^^;
- 바보 같이 `DockerFile` 로 작성해서 빌드를 연속적으로 실패했는데, v
- vscode에 여러 익스텐션 꽂아서 활용하는 편인데, dockerfile을 고래모양으로 보여주길래 오타 없는지 알았더니 오타였다 (`DockerFile`).
- 여러가지 해보니깐 그냥 Docker로 시작하면 다 고래모양 띄어준다 (extension bug)



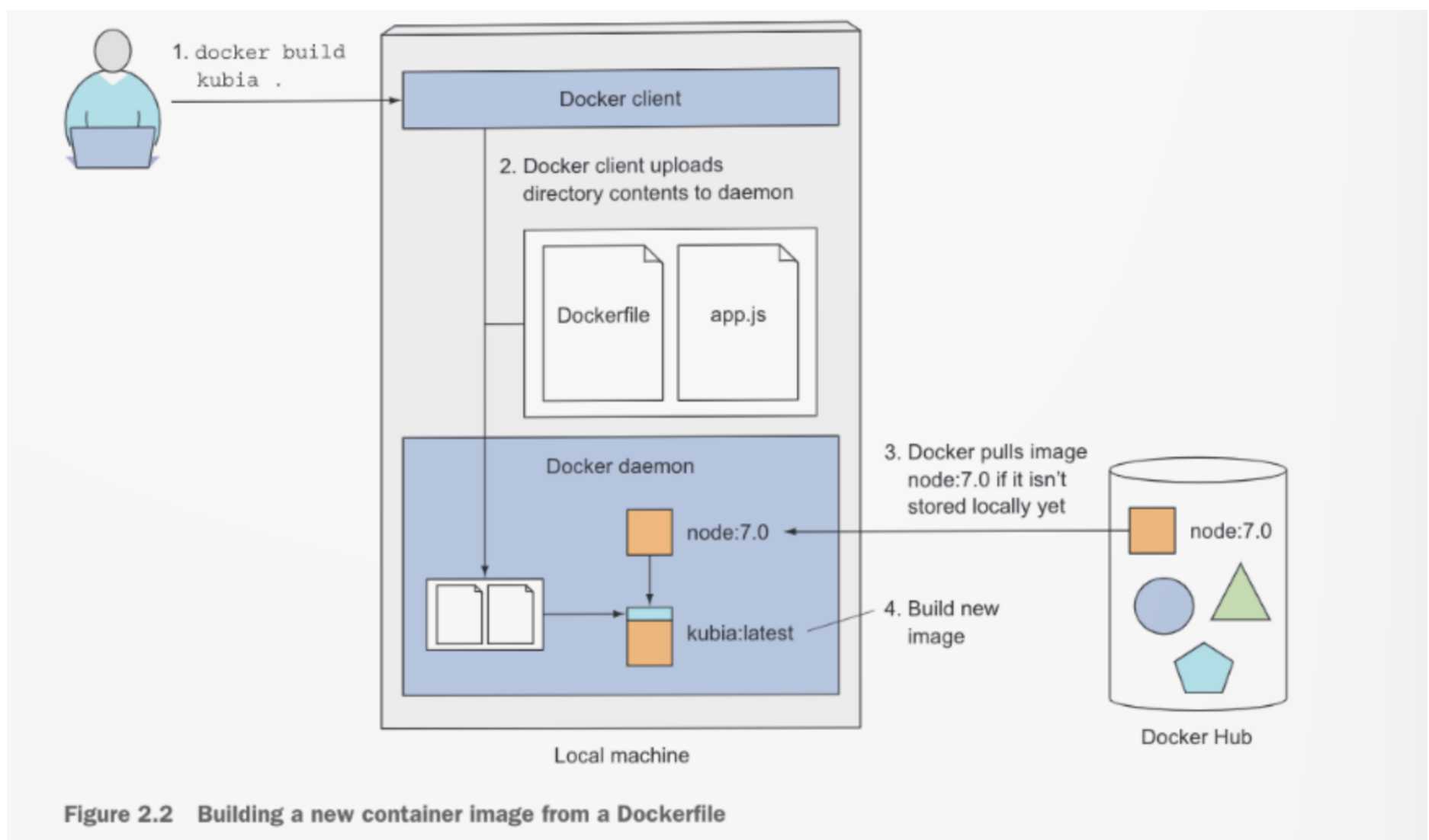
Container image 생성

이제 이미지를 빌드하면 된다.

```
docker build -t kubia .
```

- `docker build`: build를 위한 명령어
- `-t name`: tag(이름) 지정
- `.`: 현재 디렉토리를 컨테이너로 !

전체적인 이미지 빌드 과정을 그림으로 나타내면 다음과 같다.



- 디렉터리의 전체 콘텐츠가 도커 데몬에 업로드되고, 그곳에서 이미지가 빌드된다.
 - 리눅스 아닌 경우, 데몬은 가상머신 내부에서 실행됨
- 사용하려는 이미지가 로컬에 없으면, 이미지 리포지터리(docker hub)에서 가져온다.

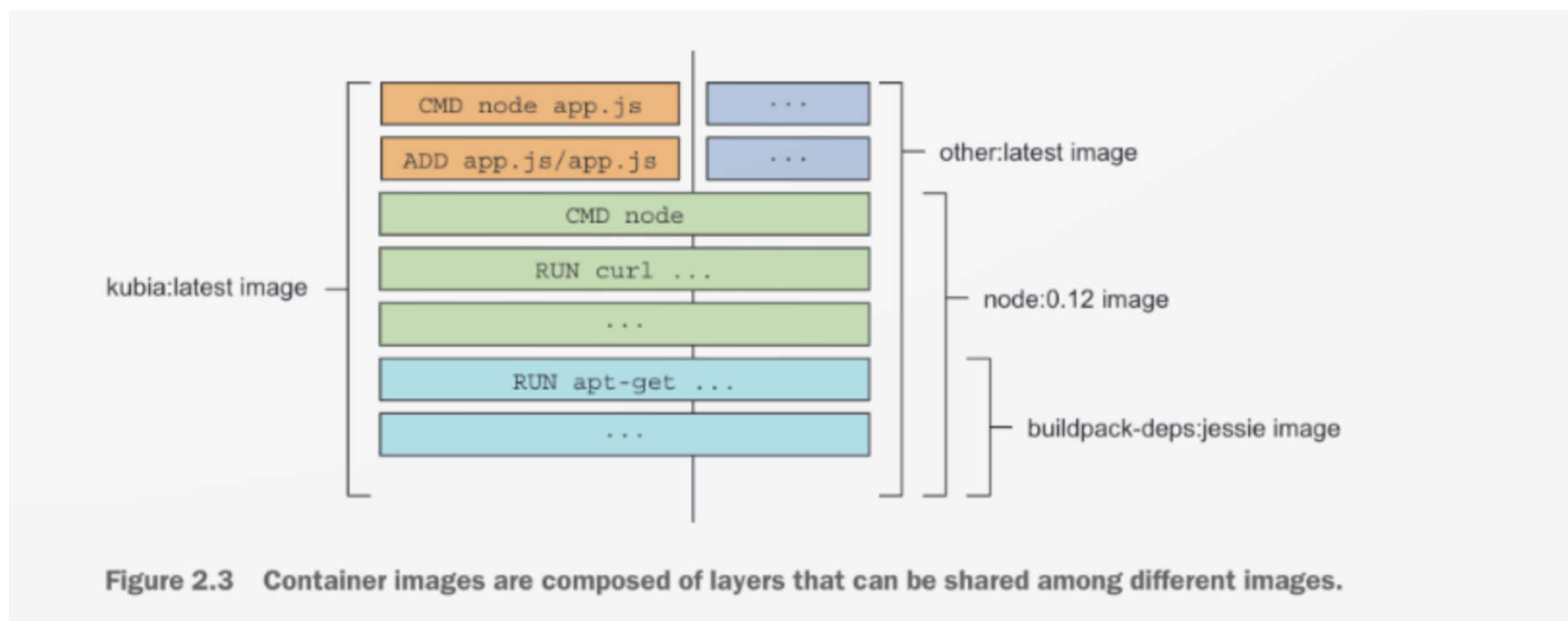
이미지 레이어

이미지라는 것은 여러 개의 레이어로 구성된다.

- 서로 다른 이미지가 여러 개의 레이어를 공유할 수 있음 → 이미지 저장과 전송에 효과적
- 공통적인 레이어가 있다면 다수의 이미지를 생성하더라도 단 한 번만 저장됨
- 도커는 저장되지 않은 레이어만 다운로드함

```
~/Documents/dev/docker/docker-with-k8s/step2/1_host_name_response
master docker build -t kubia .
[+] Building 38.6s (8/8) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 102B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/node:7 3.3s
=> [auth] library/node:pull token for registry-1.docker.io 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 400B 0.0s
=> [1/2] FROM docker.io/library/node:7@sha256:af5c2c6ac8bc3fa372ac031ef60c45a285eeba7bce9ee9ed66dad3a01e29ab8d 35.1s
=> => resolve docker.io/library/node:7@sha256:af5c2c6ac8bc3fa372ac031ef60c45a285eeba7bce9ee9ed66dad3a01e29ab8d 0.0s
=> => sha256:d9aed20b68a4a40a396dc03a881c43531097404838e4dc60dbb7ed29f95974aa 7.21kB / 7.21kB 0.0s
=> => sha256:ad74af05f5a24bcf9459ae1cf7718628c2aeb6b587eb51b6eeaf639aca3e566f 52.61MB / 52.61MB 22.7s
=> => sha256:2b032b8bbe8bc215ad3337035d0183fc353841ec6478d1c481e6e13628ad9e00 19.26MB / 19.26MB 2.3s
=> => sha256:a9a5b35f6ead105e66a9af969454ac09b5772eeb0c6281972c48d2ce882e8eba 43.23MB / 43.23MB 13.7s
=> => sha256:af5c2c6ac8bc3fa372ac031ef60c45a285eeba7bce9ee9ed66dad3a01e29ab8d 2.01kB / 2.01kB 0.0s
=> => sha256:3245b5a1c52cbf0ae23d948fb94ef7b321e3dc54e13c3f6cf79951ed8237f03e 131.86MB / 131.86MB 23.3s
=> => sha256:afa075743392fc2e79375c44e3ef285f775b722bbf27d01f8fe3789144e97f7c 4.38kB / 4.38kB 15.2s
=> => sha256:9fb9f21641cdf120516153e477e21bad887648cdced5e870cb6eed1c1a2aefa3 119.15kB / 119.15kB 15.5s
=> => sha256:3f40ad2666bcec6baf03f4322c4736464de25d351740037d3ecae8dfd46d9693 15.68MB / 15.68MB 20.1s
=> => sha256:49c0ed396b49ee88dec473e3277b89a80d79a5eefebb96c0e1a649069630cfbb 900.58kB / 900.58kB 20.6s
=> => extracting sha256:ad74af05f5a24bcf9459ae1cf7718628c2aeb6b587eb51b6eeaf639aca3e566f 2.4s
=> => extracting sha256:2b032b8bbe8bc215ad3337035d0183fc353841ec6478d1c481e6e13628ad9e00 0.7s
=> => extracting sha256:a9a5b35f6ead105e66a9af969454ac09b5772eeb0c6281972c48d2ce882e8eba 2.2s
=> => extracting sha256:3245b5a1c52cbf0ae23d948fb94ef7b321e3dc54e13c3f6cf79951ed8237f03e 4.9s
=> => extracting sha256:afa075743392fc2e79375c44e3ef285f775b722bbf27d01f8fe3789144e97f7c 0.1s
=> => extracting sha256:9fb9f21641cdf120516153e477e21bad887648cdced5e870cb6eed1c1a2aefa3 0.1s
=> => extracting sha256:3f40ad2666bcec6baf03f4322c4736464de25d351740037d3ecae8dfd46d9693 0.8s
=> => extracting sha256:49c0ed396b49ee88dec473e3277b89a80d79a5eefebb96c0e1a649069630cfbb 0.1s
=> [2/2] ADD app.js /app.js 0.2s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:738d4b42aabea3af535c3f1618d5ff951952d3685e2f8d5ce6ecf24eca55b6d0 0.0s
=> => naming to docker.io/library/kubia 0.0s
```

- 첫 빌드 장면 → node:7 이미지가 없어서 다 다운로드 받는다.
- 다음에 또 다른 이름으로 빌드하게 되면, 로컬에 이미 node:7 이미지가 있기 때문에 다운로드를 하지 않는다.



- Dockerfile 을 통해 이미지를 빌드하면 위 그림처럼 구성된다.
 - 공유 가능한 레이어를 가져오고 그 위에 작성된 명령에 따른 레이어를 쌓는 식

로컬 이미지 확인

`docker images` : 로컬에 저장된 이미지 리스트 출력

컨테이너 이미지 실행

```
docker run --name kubia-container -p 8080:8080 -d kubia
```

- `docker run` : 이미지 실행
- `--name kubia-container` : 컨테이너 이름을 kubia-container로 지정
- `-p 8080:8080` : local 8080 포트와 컨테이너 내부의 8080 포트 매핑시킴
- `-d` : 백그라운드 실행
- `kubia` : kubia 이미지를 실행

실행중인 컨테이너 조회

```
docker ps
```

- 컨테이너 아이디, 이미지 명, 만들어진 날짜, 상태, 포트 등 간단 정보 제공
- `-a` 옵션을 주면 실행 및 중지된 모든 컨테이너 출력

```
docker inspect
```

- 컨테이너의 상세정보를 JSON으로 출력

실행 중인 컨테이너 내부에서 셸 실행

```
docker exec -it kubia-container bash
```

- `-i` : 표준입력 오픈상태 유지 → 셸에 명령어 입력하기 위해 필요
- `-t` : 의사 터미널 할당 → 셸 화면출력을 위해 필요

컨테이너 종지와 삭제

종지: `docker stop kubia-container` → 재실행 가능

삭제: `docker rm kubia-container`

이미지 푸시

외부 이미지 저장소(레지스트리)는 도커 허브, 구글 컨테이너 레지스트리 등이 있다.

도커 허브를 이용해 이미지를 푸시할 것.

- 이미지 푸시를 위해서는 이미지 태그를 지정해주어야 한다.
- 이미지의 리포지터리 이름이 도커 허브 ID로 시작해야만 이미지 푸시 가능
 - 만약 아니라면 denied 뜬다.

이미지 태그 지정

```
docker tag kubia dockerHubID/kubia
```

- 도커 로컬 이미지에 태그를 추가
- kubia라는 컨테이너 이미지에 레포지터리 이름만 dockerHubId/kubia로 지정해 이미지를 만드는 것

도커 허브에 이미지 푸시

```
docker push dockerHubID/kubia
```

다른 머신에서 이미지 실행하기

```
docker run -p 8080:8080 -d dockerHubID/kubia
```

도커 허브에 올라갔으면, 다른 머신에서도 쉽게 docker를 통해 이미지를 받아 실행할 수 있다.

지금까지의 docker 명령어 정리

```
docker --version

docker build -t kubia .

docker images

docker run --name kubia-container -p 8080:8080 -d kubia

docker ps
docker ps -a

docker inspect

docker exec -it kubia-container bash

docker stop kubia-container

docker rm kubia-container

docker tag kubia dockerHubID/kubia

docker push dockerHubID/kubia

docker run -p 8080:8080 -d dockerHubID/kubia
```