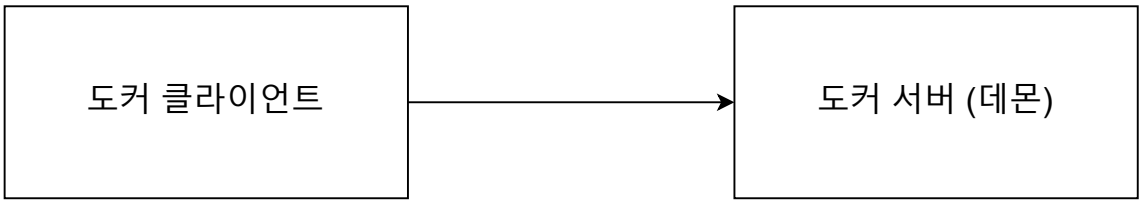
Docker를 사용할 때의 흐름 감잡기

항상 도커를 사용할 때는...

1. 먼저 도커 CLI에 커맨드를 입력한다.
2. 그러면 도커 서버 (도커 Daemon)이 그 커맨드를 받아서 그것에 따라 이미지를 생성하든 컨테이너를 실행하든 모든 작업을 하게 된다.



실제로 커맨드를 입력해보기

1. docker run hello-world

```
jaewon@Jaewonui-MacBookPro ~ % docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

설명

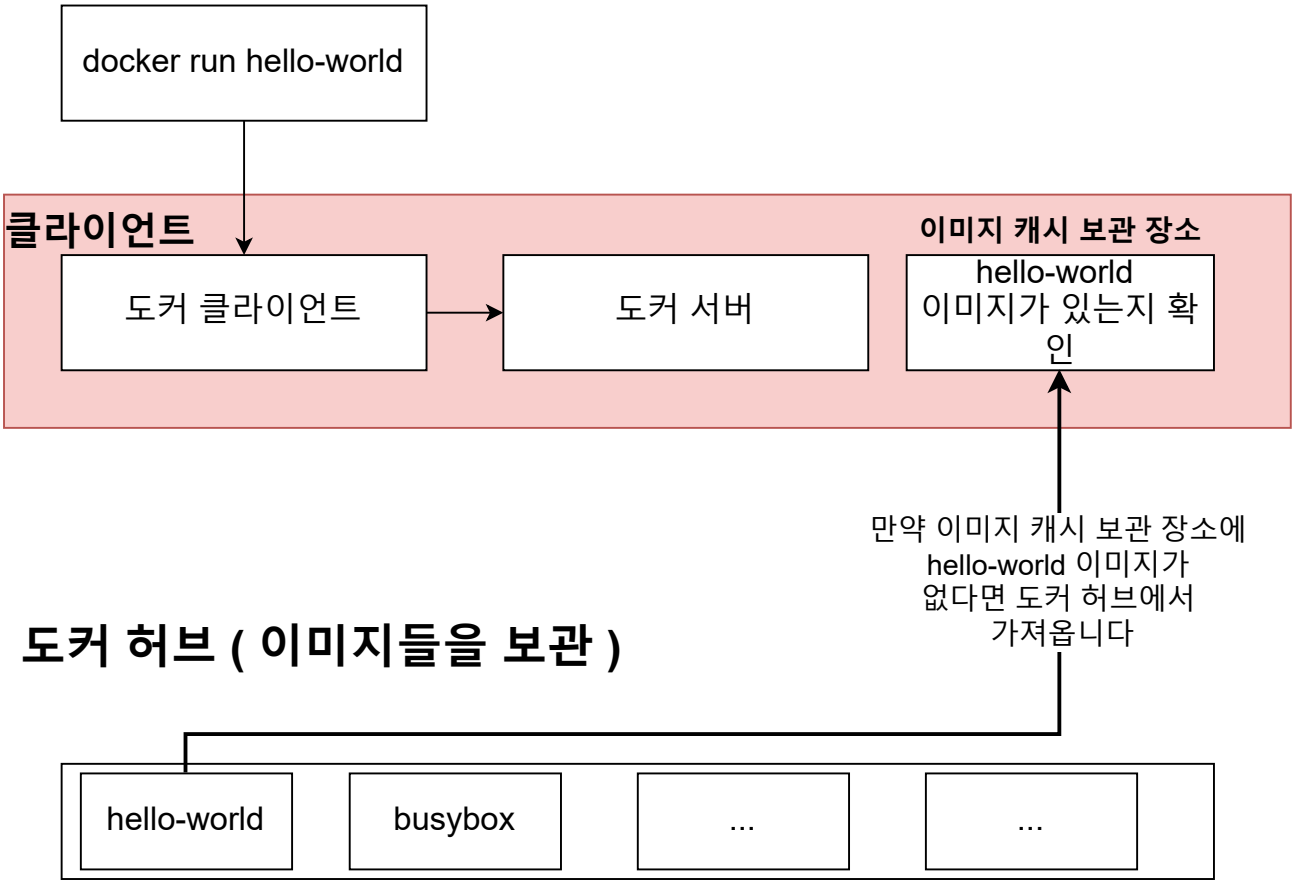
1. 도커 클라이언트에 커맨드를 입력하니 클라이언트에서 도커 서버로 요청을 보냄

2. 서버에서 hello-world라는 이미지가 이미 로컬에 cache가 되어 있는지 확인

3. 현재는 없기에 Unable to find image ~ 라는 문구가 2번째 줄에 표시

4. 그러면 Docker Hub이라는 이미지가 저장되어 있는 곳에 가서 그 이미지를
가져오고 로컬에 Cache로 보관한다.

5. 그 후 이제는 이미지가 있으니 그 이미지를 이용해서 컨테이너를 생성한다.



이제 hello-world 이미지가 캐시가 되어있으니 한번 더 docker run hello-world 하면 어떻게 될까?

1. Unable to find image ~ 라는 문구가 없이 프로그램이 실행됨.

이미지로 컨테이너를 만들기

저번 강의에서 이미지를 이용해서 컨테이너를 생성한다고 배웠습니다.
하지만 어떻게 해서 이미지를 이용해 컨테이너를 생성하는지는
다루지 않았기 때문에 이번 시간에
어떻게 컨테이너가 생성이 되게 되는지 자세히 알아보겠습니다.

시작하기 전에 기억해야 할 것

이미지는 응용 프로그램을 실행하는데 **필요한 모든 것**을 포함하고 있습니다.

그러면 그 필요한 것이 무엇일까요 ?

1. 컨테이너가 시작될 때 실행되는 명령어 ex) run kakaotalk

2. 파일 스냅샷 ex) 컨테이너에서 카카오톡을 실행하고 싶다면
카카오톡 파일 (카카오톡을 실행하는데 필요한 파일)
스냅샷

* 파일 스냅샷은 디렉토리나 파일을 캡처한 것

이미지

1. 시작 시 실행할 명령어

2. 파일 스냅숏

카카오톡 파일

이미지로 컨테이너 만드는 순서

1. Docker 클라이언트에 docker run <이미지> 입력해줍니다.

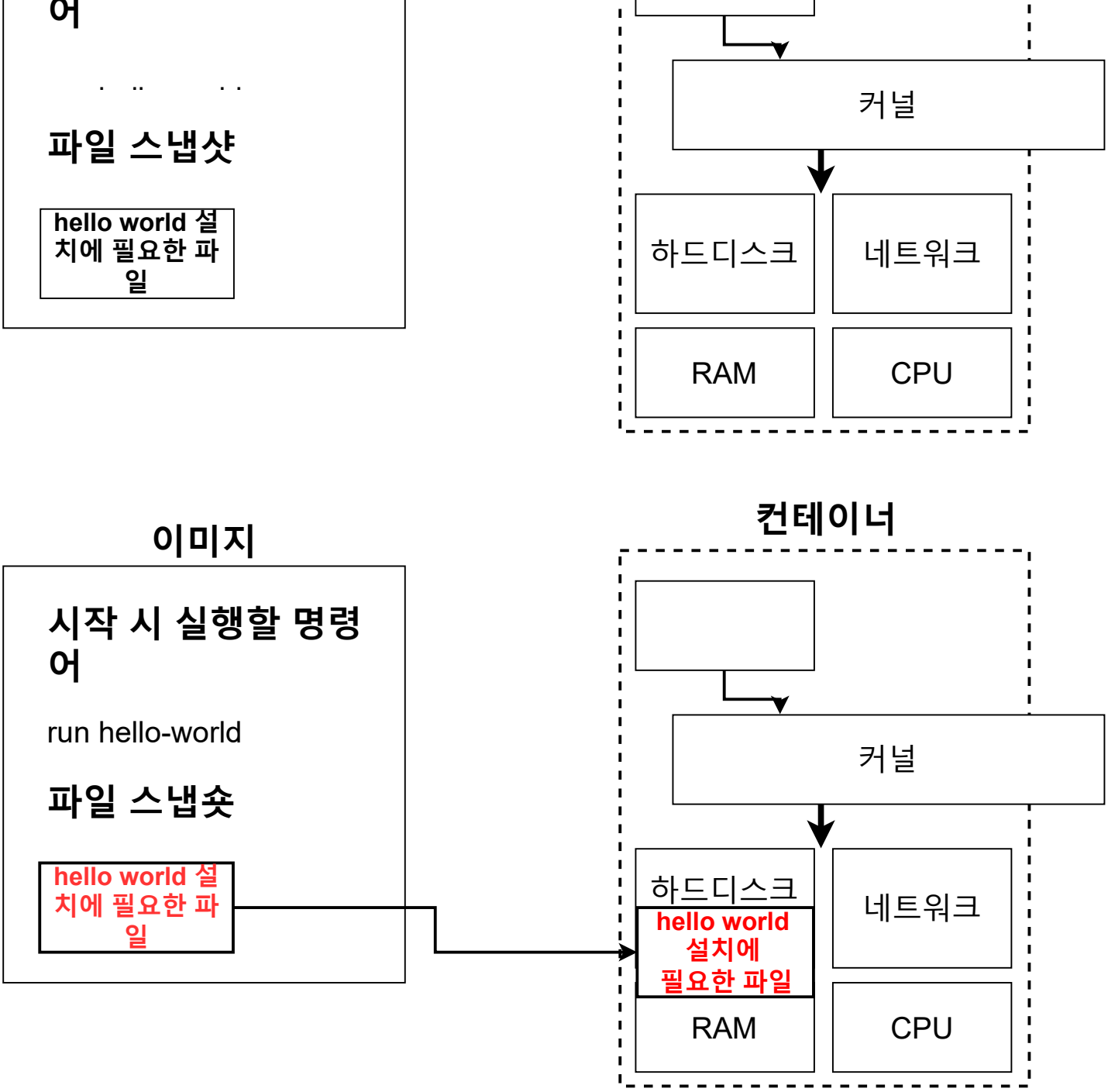
2. 도커 이미지에 있는 파일 스냅샷을 컨테이너 하드 디스크에 옮겨 줍니다

이미지

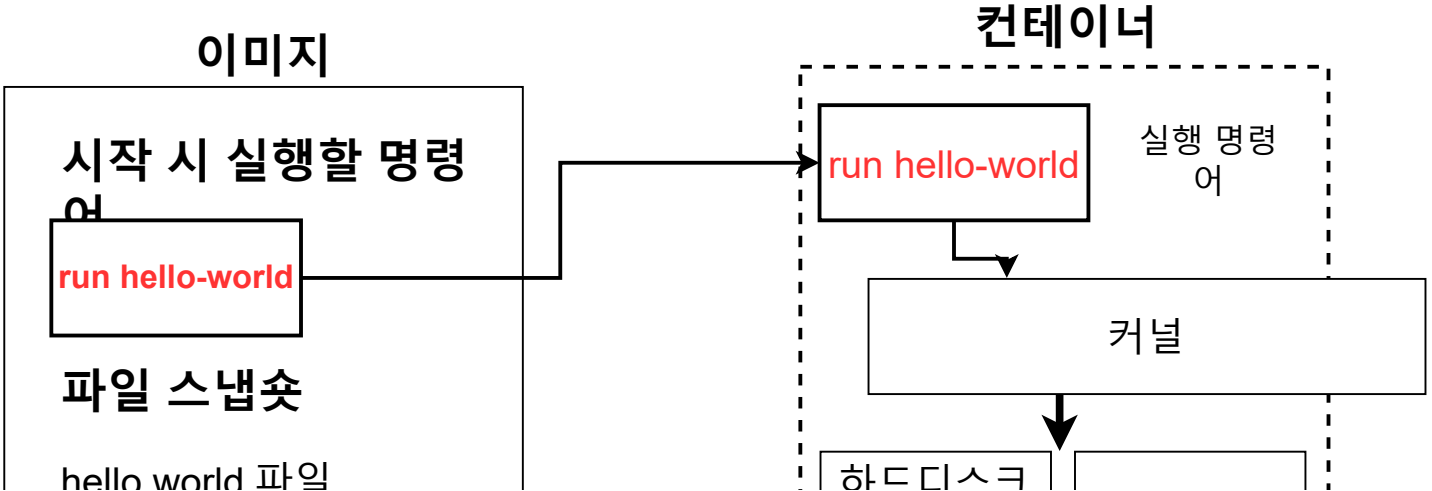
시작 시 실행할 명령

컨테이너

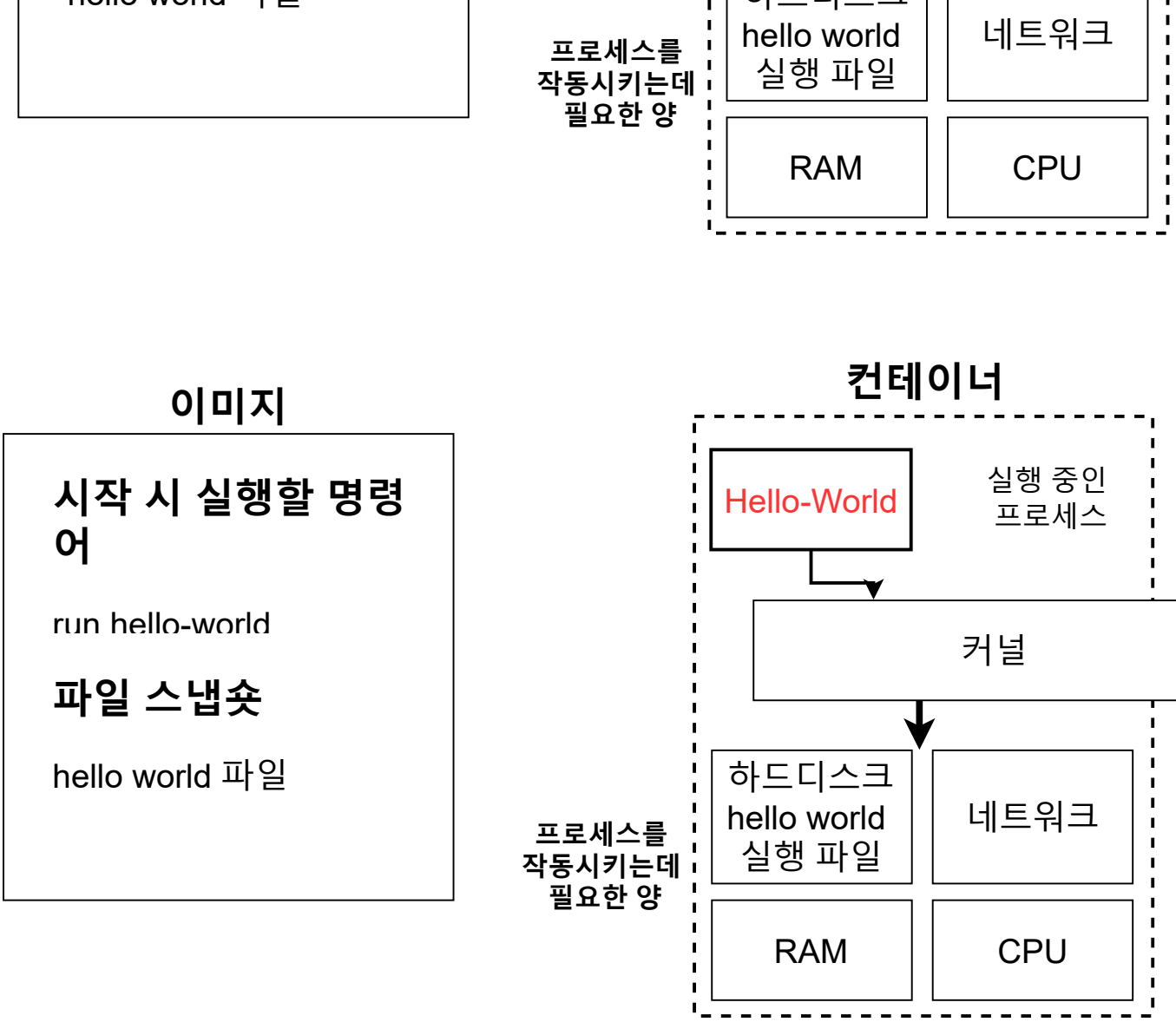
.



3. 이미지에서 가지고 있는 명령어 (컨테이너가 실행될때 사용될 명령어를 이용해서 카카오톡을 실행시켜줍니다.



)를


이미지
시작 시 실행할 명령어
run hello-world
파일 스냅숏
hello world 파일
컨테이너
Hello-World
실행 중인 프로세스
커널
하드디스크
hello world
실행 파일
네트워크
RAM
CPU
프로세스를 작동시키는데 필요한 양

도커 이미지 생성하는 순서

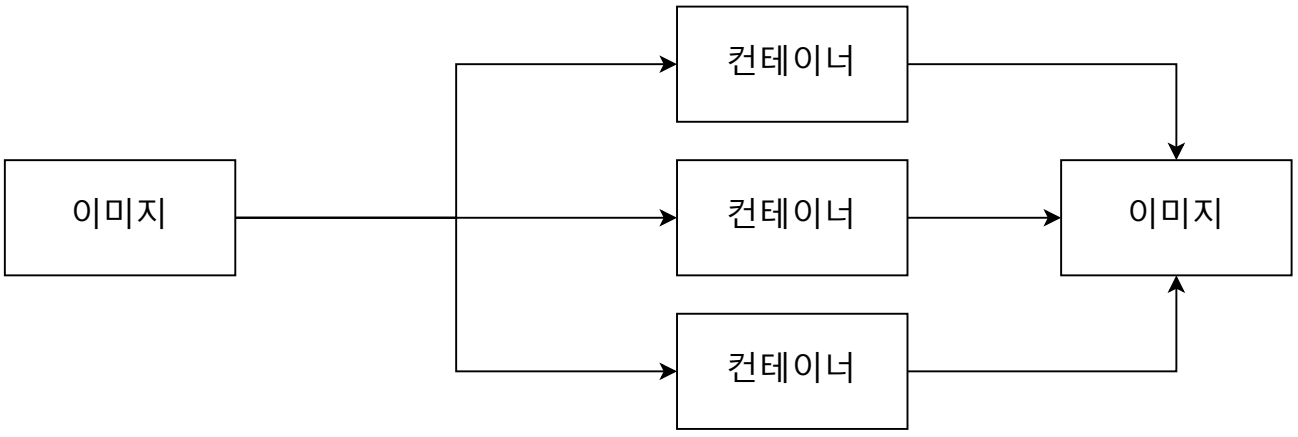
**현재까지는 도커 이미지를 항상 도커 허브에 이미 있던 것들만
가져와서 사용했습니다. 하지만 저희가 직접 도커 이미지를 만들어서 사용할 수도 있고 직접 만든 도커 이미지를 도커 허브에 올려서 공유할 수도 있는데요.
그래서 어떻게 직접 이미지를 만들 수 있는지 알아보겠**

간단하게 도커 이미지 다시 복습하기

1.도커 이미지는 컨테이너를 만들기 위해 필요한 설정이나 종속성들을 갖고 있는 소프트웨어 패키지입니다.

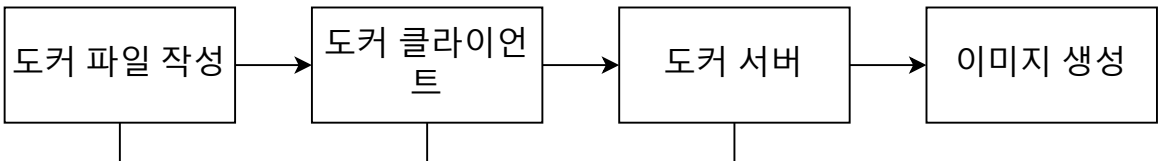
2.지금까지 해왔듯이 도커 이미지는 Dockerhub에 이미 다른 사람들이 만들어 놓은 것을 이용할 수도 있으며, 직접 도커 이미지를 만들어서 사용할 수도 있고 직접 만든 것을 Dockerhub에 업로드할 수도 있습니다.

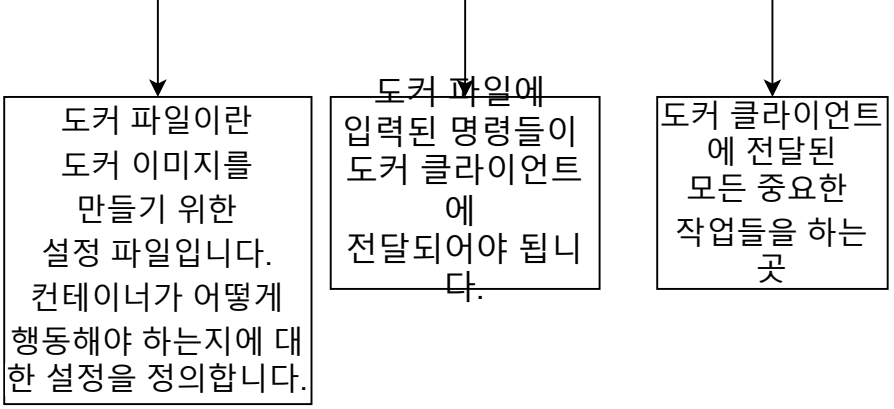
도커 이미지를 이용해서 도커 컨테이너를 생성.
ex) docker create <이미지 이름>



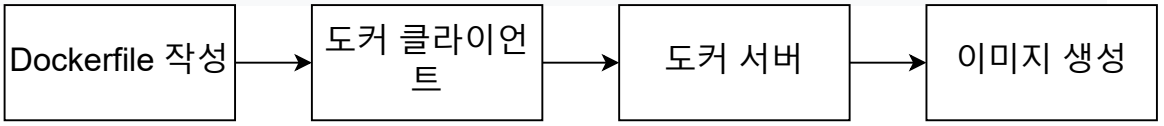
**컨테이너는 도커 이미지로 생성...
그럼 도커 이미지는 어떻게 생성할까요?**

도커 이미지 생성하는





그럼 이제 먼저 Dockerfile을 만드는 법을 알아보겠습니다



Dockerfile 만들기

도커 파일(Docker file)이란 ?

- 도커 이미지를 만들기 위한 설정 파일이며, 컨테이너가 어떻게 행동해야 하는지에 대한 설정들을 정의해 주는 곳 입니다.

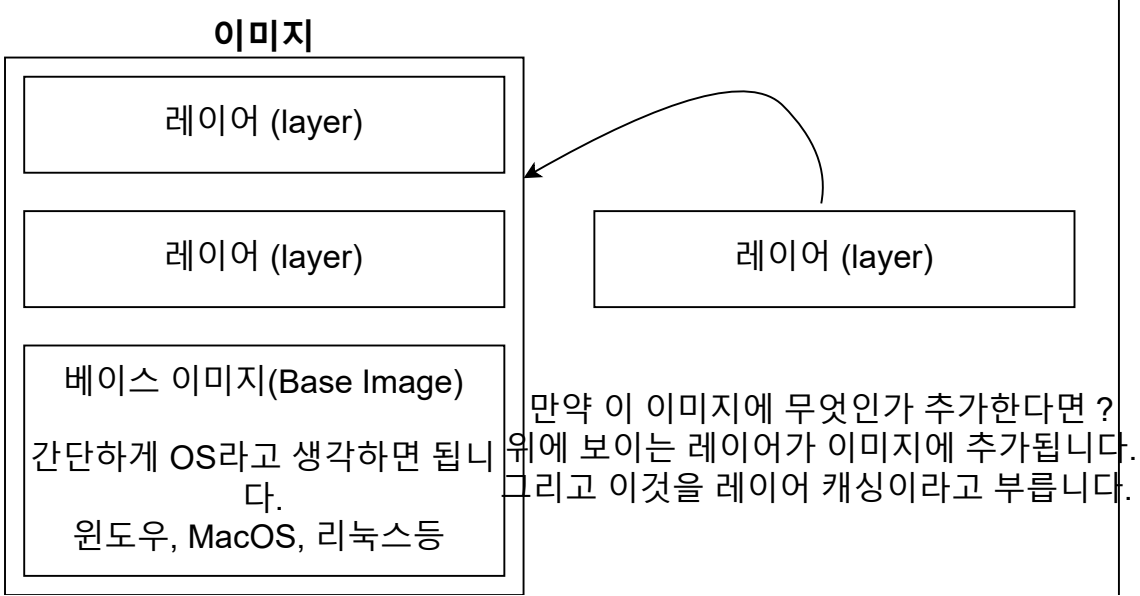
도커 파일 만드는 순서 (도커 이미지가 필요한 것이 무엇인지를 생각하기)

1. 베이스 이미지를 명시해준다. (파일 스냅샷에 해당)

2. 추가적으로 필요한 파일을 다운 받기 위한 몇 가지 명령어를 명시해준다. (파일 스냅샷에 해당)

베이스 이미지는 무엇인가?

- 도커 이미지는 여러개의 레이어들로 되어 있습니다.
  그 중에서 베이스 이미지는 이 이미지의 기반이 되는 부분입니다.



만약 이 이미지에 무엇인가 추가한다면 ?
위에 보이는 레이어가 이미지에 추가됩니다.
그리고 이것을 레이어 캐싱이라고 부릅니다.

코코아톡 이미지

작 시 실행할 명령

일 스냅숏

코아톡 파일

실제로 만들면서 배우기!

도커 이미지의 목표는 "hello 문구" 출력하기

순서

1. 도커 파일을 만들 폴더 하나 만들기 ex) dockerfile-folde

2. 방금 생성한 도커 파일 폴더를 에디터를 이용해서 실행
   (Visual Studio Code 추천)

3. 파일 하나를 생성, 이름은 dockerfile

4. 그 안에 먼저 어떻게 진행해 나갈지 기본적인 토대를 명시.

FROM RUN CMD 등은 도커 서버
것이다.

```
# 베이스 이미지를 명시해준다.
FROM baseImage

# 추가적으로 필요한 파일들을 다운로드 받는다.
RUN command

# 컨테이너 시작시 실행 될 명령어를 명시해준다.
CMD [ "executable" ]
```

FROM
이미지 생성 시 기반이 되는 이미
<이미지 이름>:<태그> 형식으로
태그를 안 붙이면 자동적으로 가
ex) ubuntu:14.04

RUN
도커 이미지가 생성되기 전에 수

CMD
컨테이너가 시작되었을 때
실행할 실행 파일 또는 셸 스크립
해당 명령어는 DockerFile 내 1회

5. 이제 베이스 이미지부터 실제 값으로 추가해주기.

6. 베이스 이미지는 ubuntu를 써도 되고 centos 등을 써도 되지만
   hello를 출력하는 기능은 굳이 사이즈가 큰 베이스 이미지를 쓸 필요가 없기에
   사이즈가 작은 alpine 베이스 이미지를 사용.

7. hello 문자를 출력해주기 위해 echo를 사용하여야 하는데
   이미 alpine 안에 echo를 사용하게 할 수 있는 파일이 있기에 RUN 부분은 생략함.

```
# 베이스 이미지를 명시해준다.
FROM alpine

# 추가적으로 필요한 파일들을 다운로드 받는다.
# RUN command

# 컨테이너 시작시 실행 될 명령어를 명시해준다.
```

에게 무엇을 하라고 알려주는

지 레이어입니다.
작성
상 최신 것으로 다운 받음

행할 쉘 명령어

트입니다.
만 쓸 수 있습니다

FROM

기반이 되는 이미지 레이어입니다.

<이미지 이름>:<태그> 형식으로 작성

ex) ubuntu:14.04

MAINTAINER

메인테이너 정보입니다.

RUN

도커이미지가 생성되기 전에 수행할 쉘 명령어

VOLUME

VOLUME은 디렉터리의 내용을 컨테이너에 저장하지 않고 호
록 설정합니다.

데이터 볼륨을 호스트의 특정 디렉터리와 연결하려면 docke
옵션을 사용해야 합니다.

ex) -v /root/data:/data

CMD

컨테이너가 시작되었을 때 실행할 실행 파일 또는 셸 스크립

해당 명령어는 DockerFile내 1회만 쓸 수 있습니다.

호스트에 저장하도

r run 명령에서 -v

트입니다.

```
CMD [ "echo", "hello" ]
```

이렇게 해서 도커 파일 만들기는 완성되었습니다.
이제 이미지는 어떻게 만들어야 하는지 더욱 알아보겠

WORKDIR

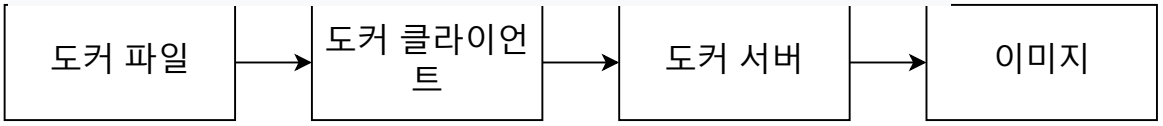
CMD에서 설정한 실행 파일이 실행될 디렉터리입니다.

EXPOSE

호스트와 연결할 포트 번호입니다.

도커 파일로 도커 이미지 만들기

완성된 도커 파일로 어떻게 이미지를 생성하나요?



도커 파일에 입력된 것들이 도커 클라이언트에 전달되어서 도커 서버가 인식하게 하여야 합니다.

그렇게 하기 위해서

docker build ./ 또는 docker build .

Build 명령어는

- 해당 디렉토리 내에서 dockerfile이라는
  파일을 찾아서 도커 클라이언트에 전달시켜준다.

- docker build 뒤에 ./ 와 . 는 둘 다 현재 디렉토리를 가리킨다.

그래서 docker build . 를 해보면

```
jaewon@Jaewonui-MacBookPro dockerfile-folder % docker build .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM alpine
 ---> a24bb4013296
Step 2/2 : CMD [ "echo", "hello" ]
 ---> Running in fa6c364fdc39
Removing intermediate container fa6c364fdc39
 ---> aff4e9afc538
Successfully built aff4e9afc538
jaewon@Jaewonui-MacBookPro dockerfile-folder % docker run aff4e9afc538
hello
```

build 과정 설명

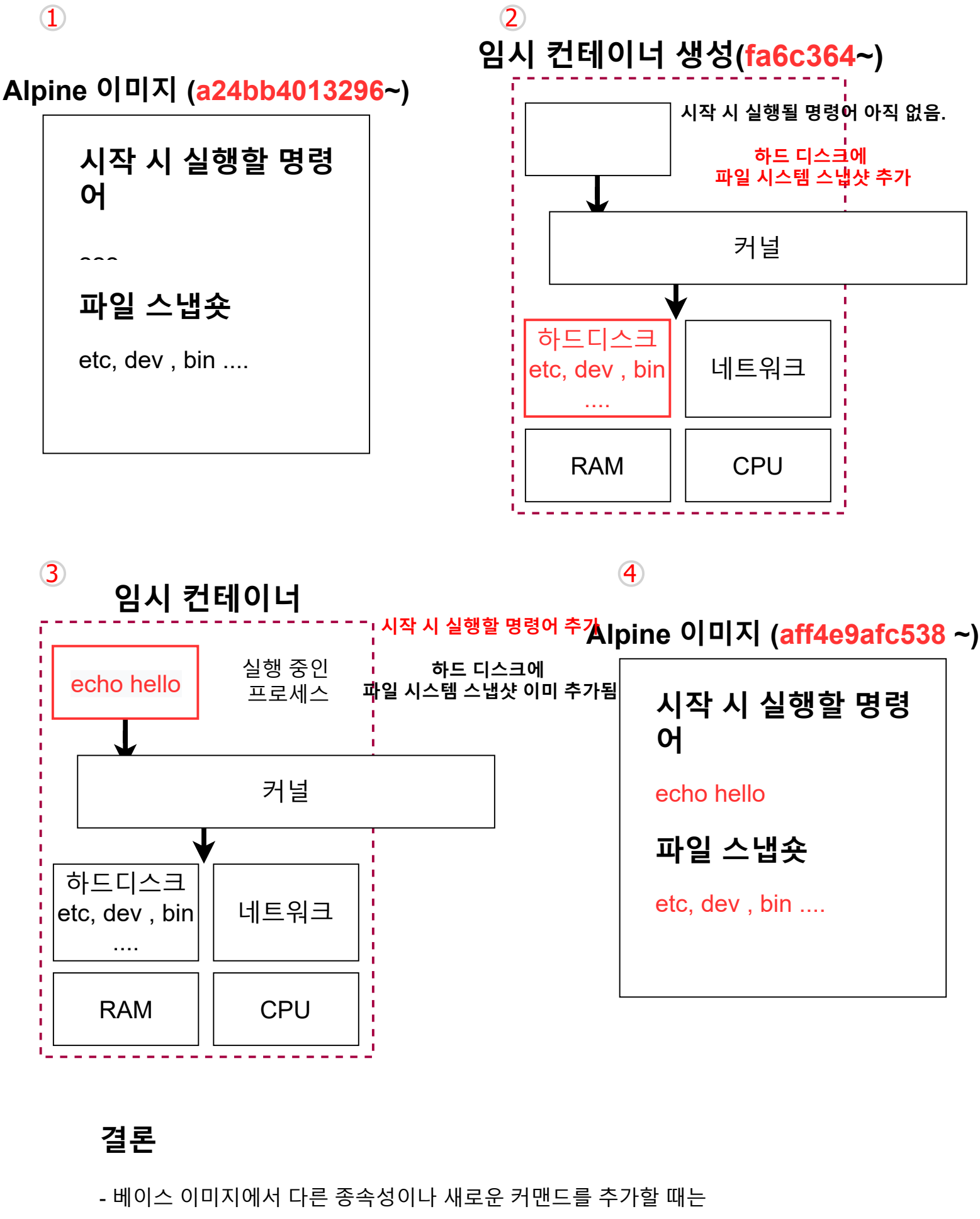
Step 1/2

- alpine 이미지 가져오기 a24bb~ 는 alpine 이미지 아이디

Step 2/2
- 임시 컨테이너 생성 후 그 컨테이너에 시작 시 사용할 명령어 포함시키기.
- 그런 후 방금 생성 한 임시 컨테이너를 지우고 새로운 이미지 만들기

Step 2/2 자세히 보기

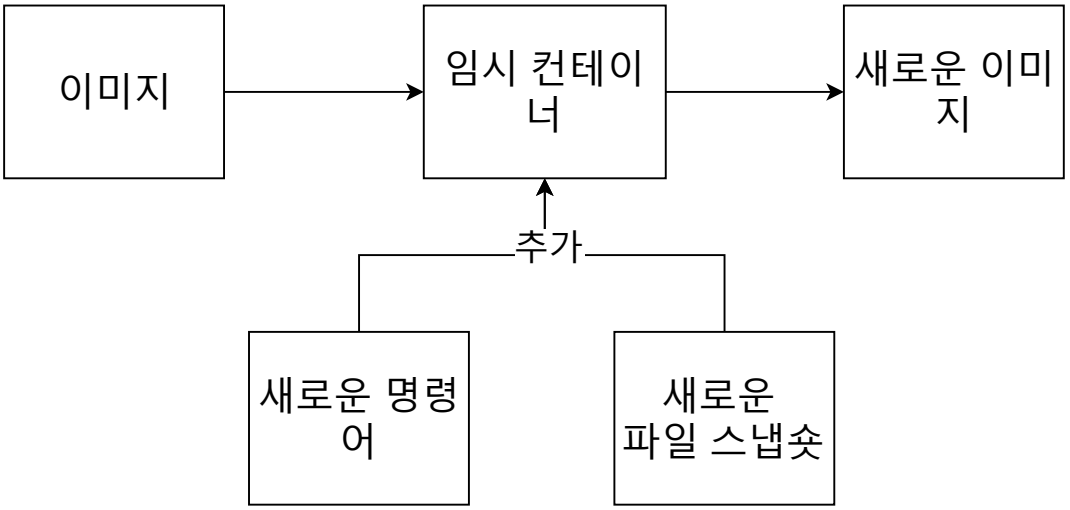
- Step 2에 보면 임시 컨테이너를 생성하고 무언가를 하고 다시 지우는데
  왜 그렇게 하는 걸까 ?



결론

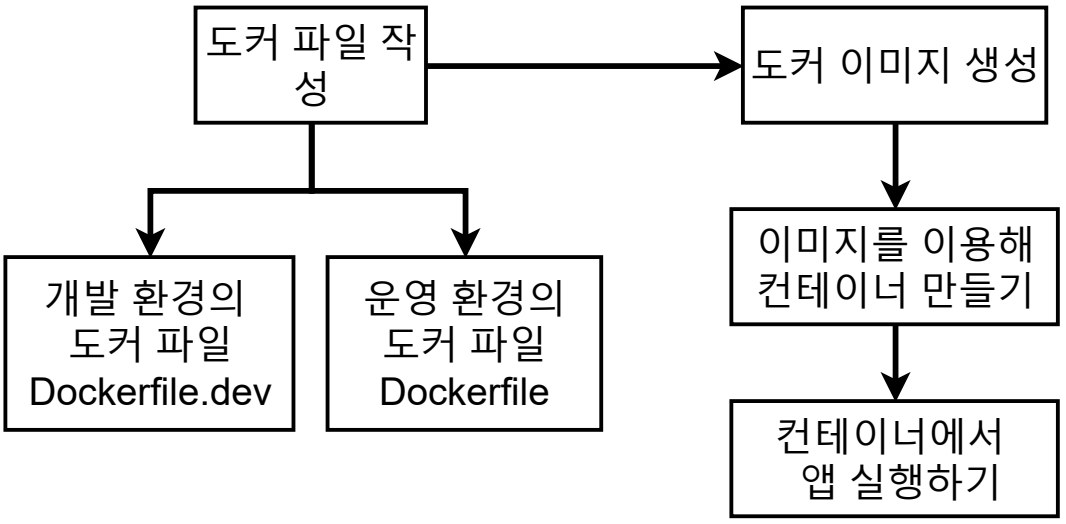
- 베이스 이미지에서 다른 종속성이나 새로운 커맨드를 추가할 때는

임시 컨테이너를 만든 후 그 컨테이너를 토대로 새로운 이미지를 만든다!!!
그리고 그 임시 컨테이너는 지워준다.



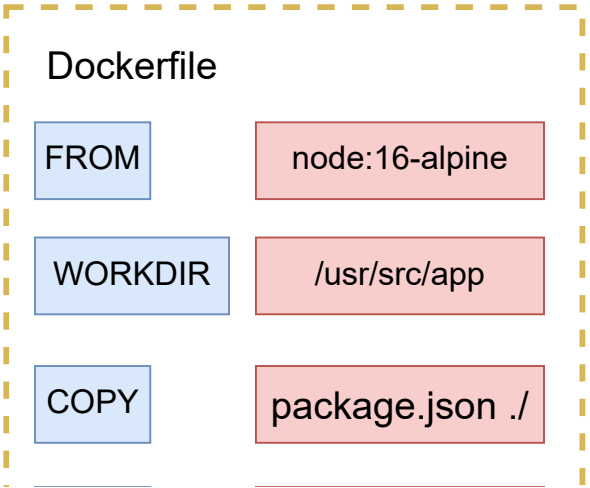
리액트를 위한 도커 파일 작성하기

도커로 어플을 실행하기 위해서



현재까지는 Dockerfile을 그냥 한 가지만 만들었지만
실제로는 Dockerfile을 개발단계를 위한 것과
실제 배포 후를 위한 것을 따로 작성하는 게 좋습니다.
그러므로 개발단계를 위해서
Dockefile이 아닌 Dockerfile.dev 라는 파일을 작성해보겠습

Dockerfile



개발 환경에서의 도커 파일 작성은

왜 저번에는 alpine 베이스 이미지를 썻는데 이번엔 node 이미지를 쓰나요 ?

- 먼저 한번 베이스 이미지를 alpine으로 해서 build를 해보겠습니다. !!!

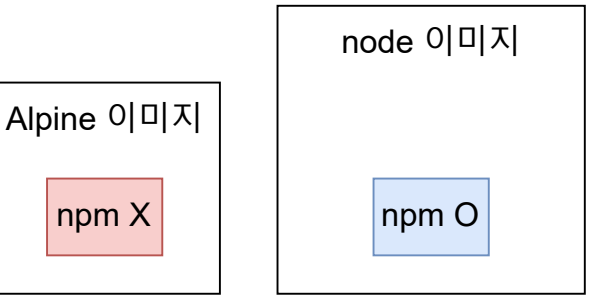
=> 그렇게 해보면 npm not found 라는 에러가 나옵니다.
그 이유는 alpine 이미지는 가장 최소한의 경량화된 파일들이 들어있기에
npm을 위한 파일이 들어있지 않아서 RUN 부분에 npm install을 할수가 없습니다.
알파인 이미지의 사이즈는 5MB정도 밖에 안됩니다.

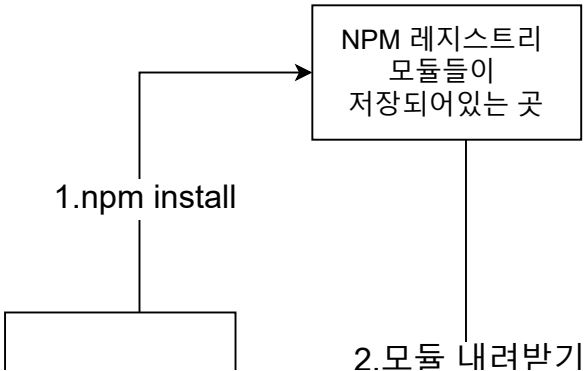
npm install은 무엇인가요?

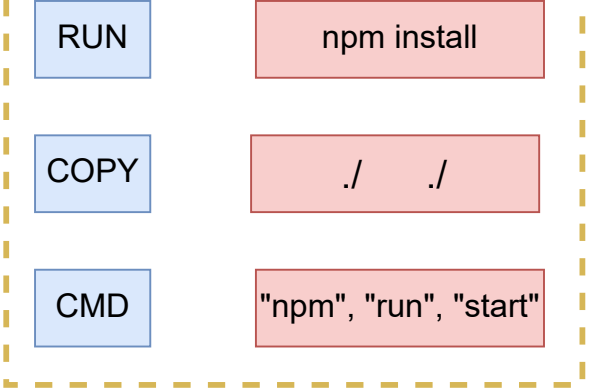
- npm은 Node.js로 만들어진 모듈을 웹에서 받아서 설치하고 관리해주는 프로그램입니다

- npm install은 package.json에 적혀있는 종속성들을 웹에서
자동으로 다운 받아서 설치해주는 명령어입니다.

- 결론적으로는 현재 리액트 JS앱을 만들때 필요한 모듈들을 다운받아 설치하는 역할을 합니다.


node 이미지
Alpine 이미지
npm X
npm O


NPM 레지스트리
모듈들이
저장되어있는 곳
1.npm install
2.모듈 내려받기



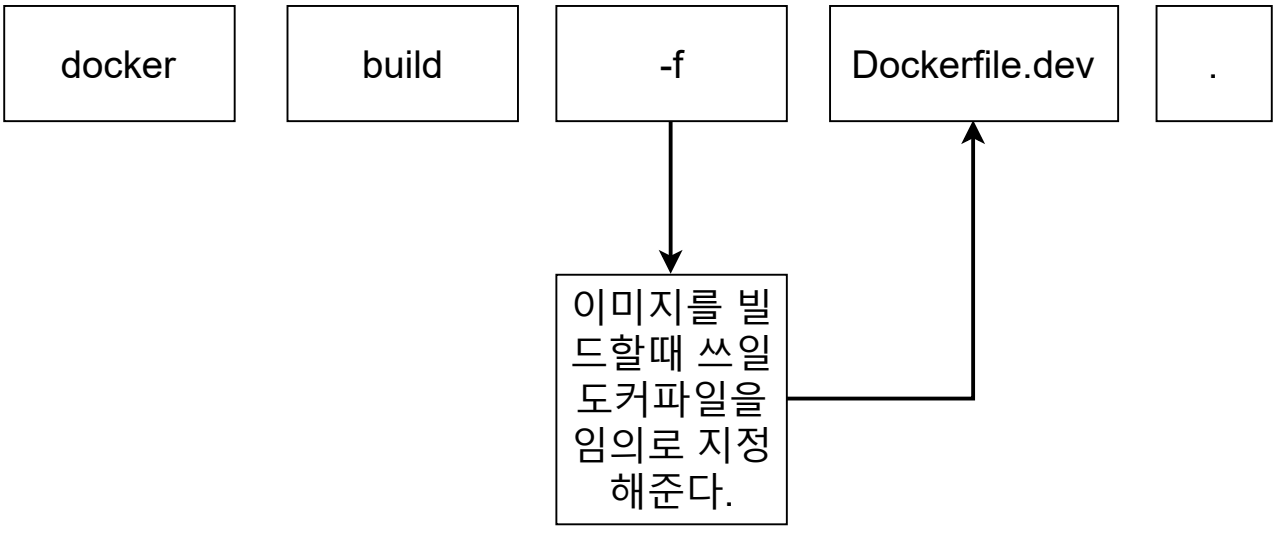
현재까지 도커 파일 작성했던

이렇게 Dockerfile.dev를 작성한 후,
이 도커 파일로 이미지를 생성하면 됩니다.
그래서 docker build ./ 으로 이미지 생성해보겠습니다.

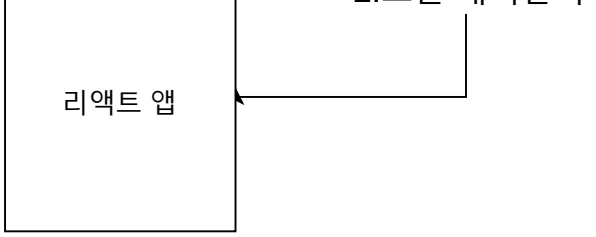
unable to evaluate symlink ...이러한 에러가 보입니다.
그 이유는 원래는 이미지를 빌드할 때 해당 디렉토리만 정해주면 dockerfile을 자동으로 찾아서 빌드하는데 현재는 dockerfile이 없고 dockerfile.dev밖에 없습니다. 그러기에 자동으로 올바른 도커 파일을 찾지 못하여 이런 에러가 발생하게 되는 것입니다.

해결책은 임의로 build 할 때 어떠한 파일을 참조할지 알

임의로 알려주는 방법은 빌드를 할 때 그냥 docker build .으로 하는 게 아니라 이 아래처럼 해줘야 한다.



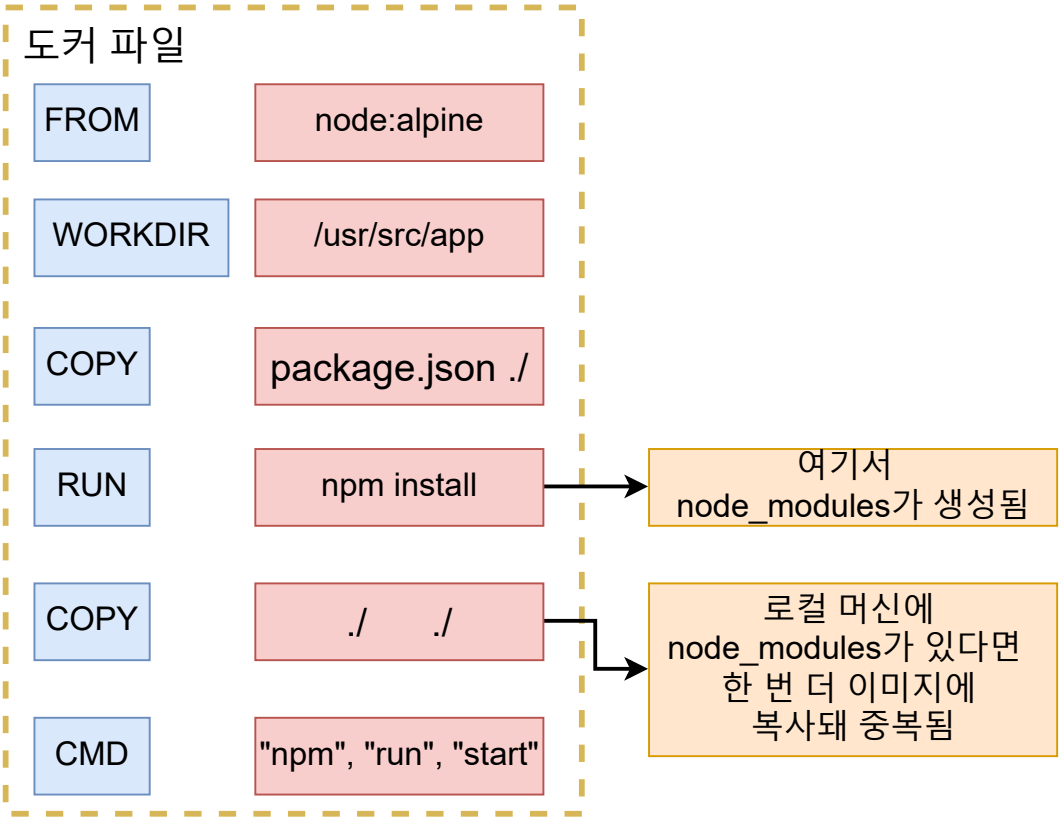
그래서 이런 식으로 -f 옵션을 이용하여서 다시 해보면 개발단계에서 리액트를 실행하게 해 줄
이미지 빌드가 가능해졌습니다.


리액트 앱

한가지 더 팁이 있다면

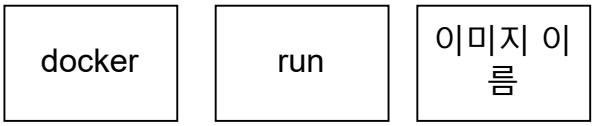
**현재 로컬 머신에 node_modules 폴더가 있습니다.
이곳에는 리액트 앱을 실행할 때 필요한 모듈들이 들어 있지만
이미지를 빌드할 때 이미 npm install로 모든 모듈들을 도커 이미지에 다운로드하기 때문에
굳이 로컬 머신에 node_modules을 필요로 하지 않는**

Dockerfile.dev

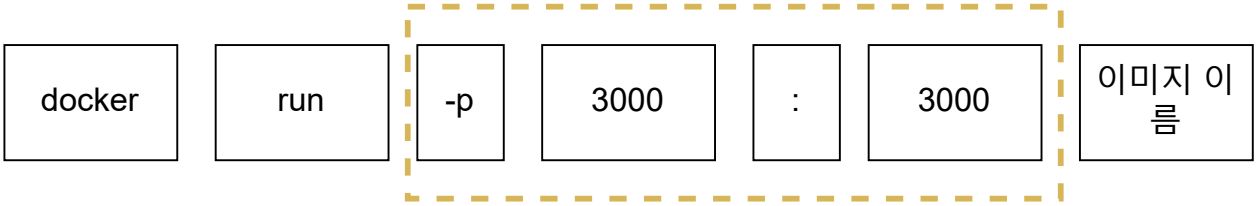


생성한 이미지로 어플리케이션 실행 시 접근이 안 되는 이유 (포트 맵핑)

현재까지 컨테이너를 실행하기 위해 사용한 명령어



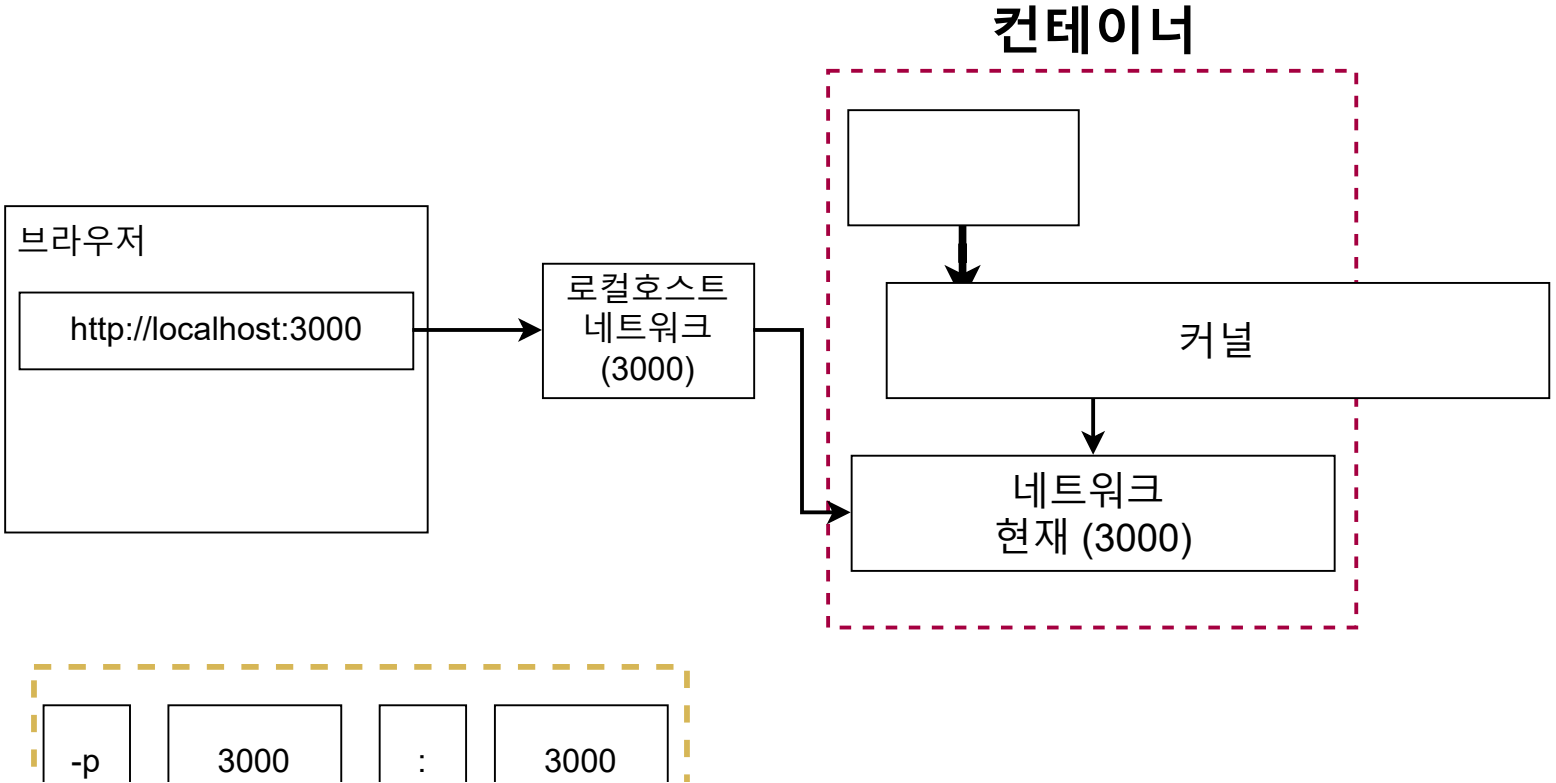
앞으로 컨테이너를 실행하기 위해 사용 할 명령어



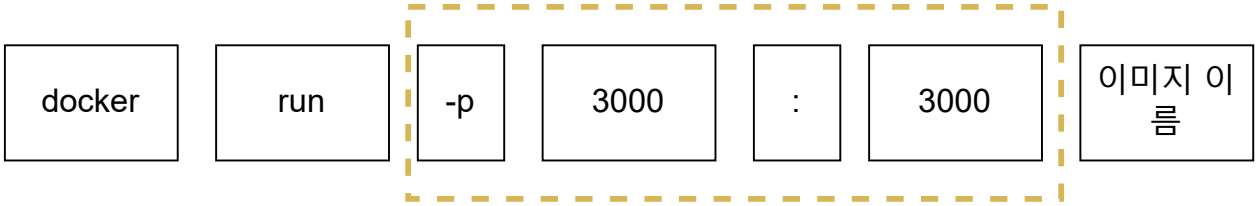
새롭게 추가된 부분

새롭게 추가된 부분은 무엇을 위한 부분인가 ?

우리가 이미지를 만들 때 로컬에 있던 파일(package.json)등을 컨테이너에 복사해줘야 했었다.
그것과 비슷하게 네트워크도 로컬 네트워크에 있던 것을 컨테이너 내부에 있는 네트워크에 연결을 시켜주어야 한다.



이 명령어를 이용해서 다시 실행



5000번 포트를 이용해서 애플리케이션에 접속하기

도커 컴포즈 파일 작성

앞서 리액트 앱을 실행할때 너무나 긴 명령어를 치는게 많이 불편했었습니다.
그러한 불편을 해소하기 위해서 도커 Compose를 이용해서

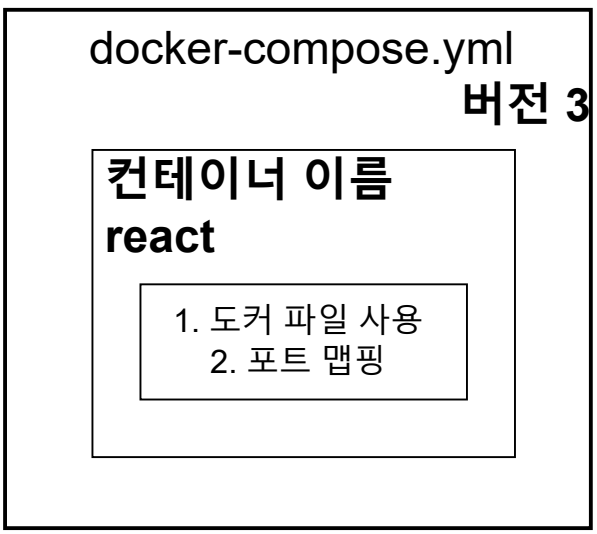
docker run -p 3000:3000 | -v /usr/src/app/node_modules | -v $(pwd):/usr/src/app | <이미지 아이디>

너무 길다....

이걸 간단히 하기 위해서 Docker Compose 파일을 작성해 보겠습니다.

1.먼저 docker-compose.yml 파일 생

2.docker-compose.yml 파일 작성하기

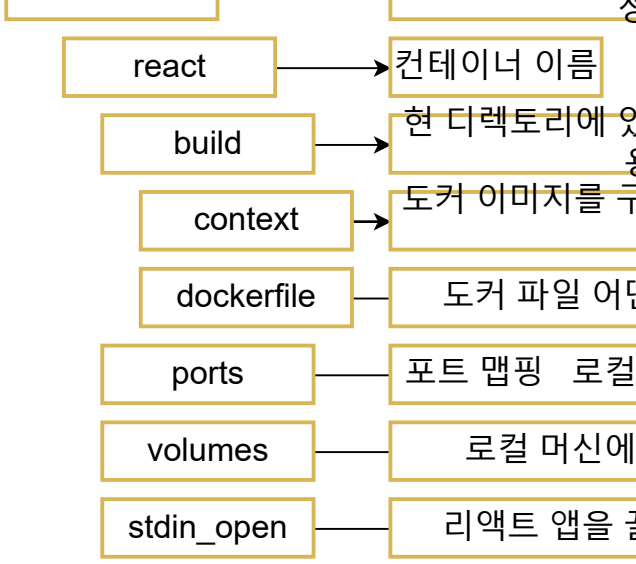



```yaml
version: "3"
services:
```

version → 도커 컴포즈의 버전

services → 이곳에 실행하려

```yaml
react:
  build:
    context: .
    dockerfile: Dockerfile.dev
  ports:
   - "3000:3000"
  volumes:
   - /usr/src/app/node_modules
   - ./:/usr/src/app
  stdin_open: true
```



3. docker-compose를 이용한 어플리케이션 실

docker-compose	up

정의

있는 Dockerfile 사용

구성하기 위한 파일과 폴더들이 있는 위치

던 것인지 지정

포트 : 컨테이너 포트

있는 파일들 맵핑

할때 필요(버그 수정)