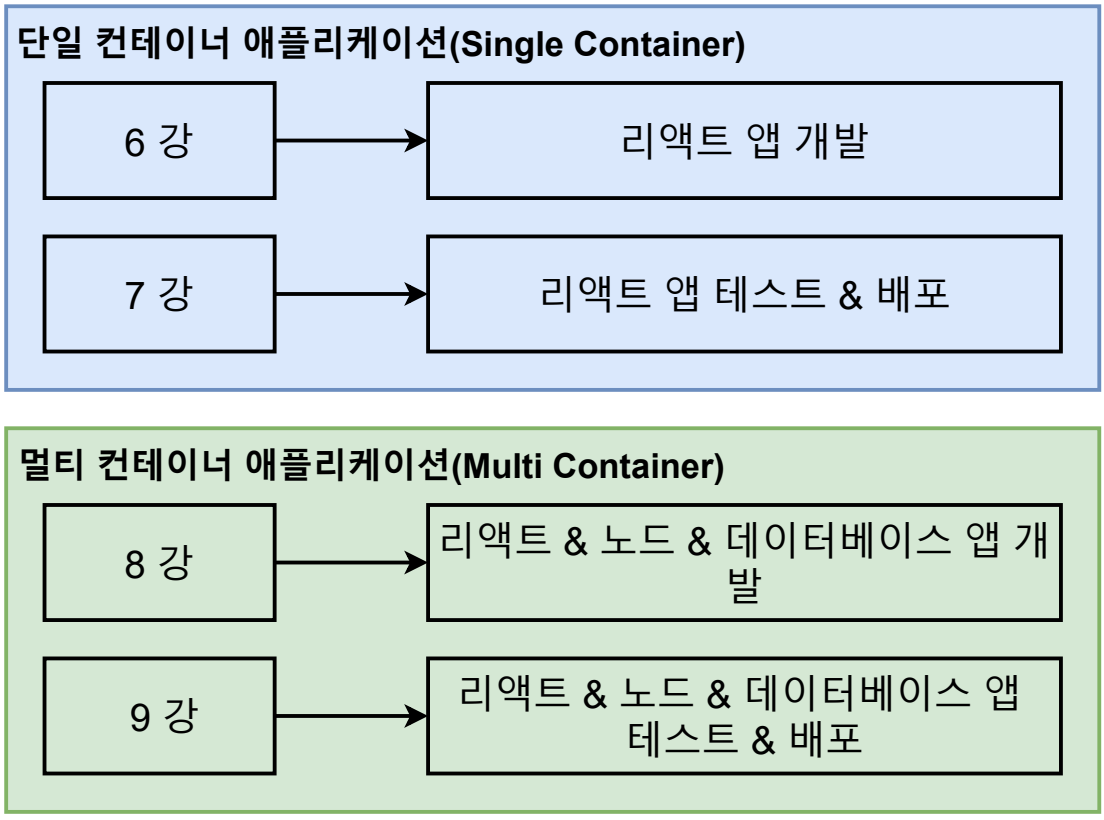
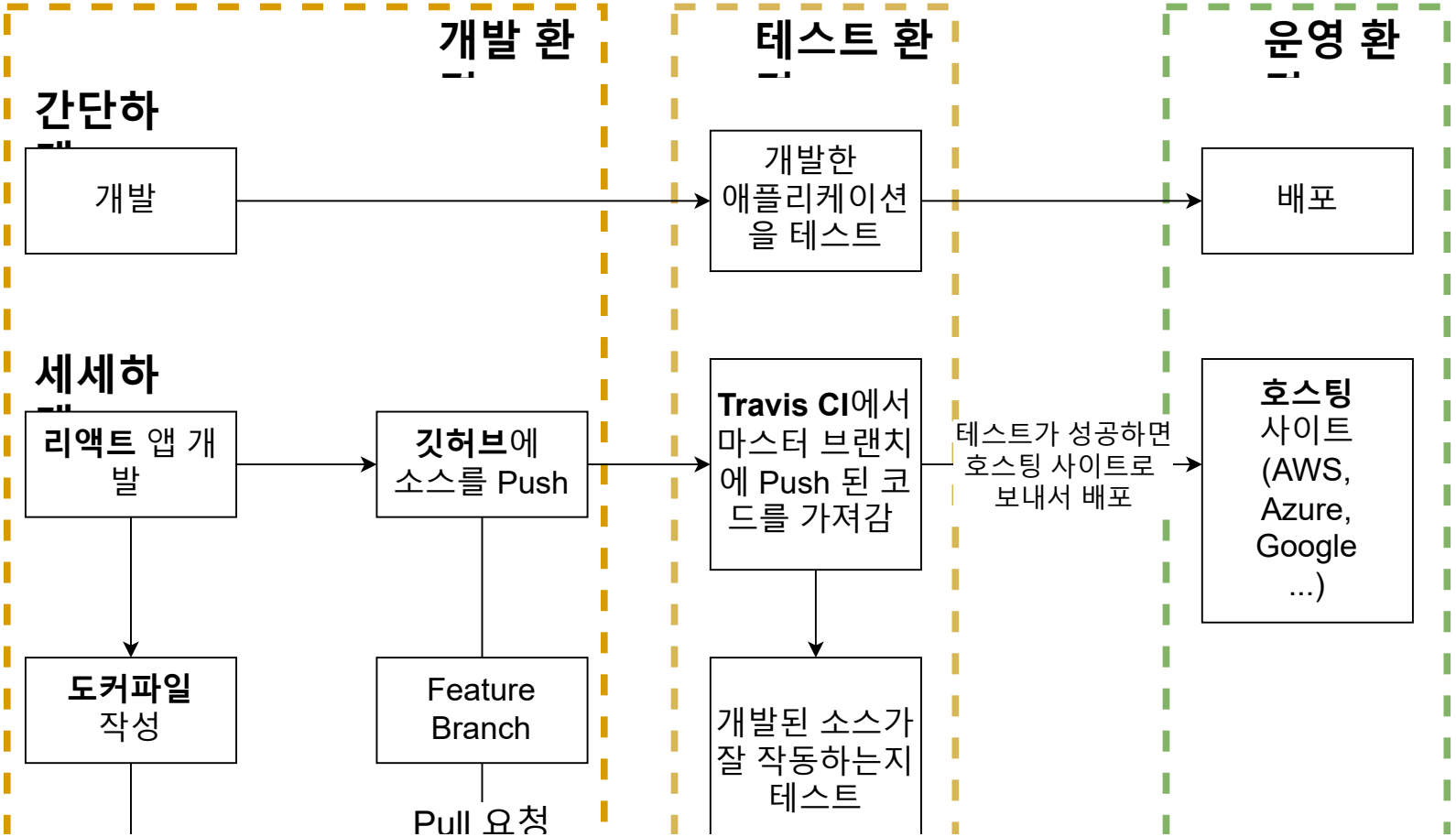

단일 컨테이너 애플리케이션(Single Container)
6 강
리액트 앱 개발
7 강
리액트 앱 테스트 & 배포
멀티 컨테이너 애플리케이션(Multi Container)
8 강
리액트 & 노드 & 데이터베이스 앱 개발
9 강
리액트 & 노드 & 데이터베이스 앱 테스트 & 배포

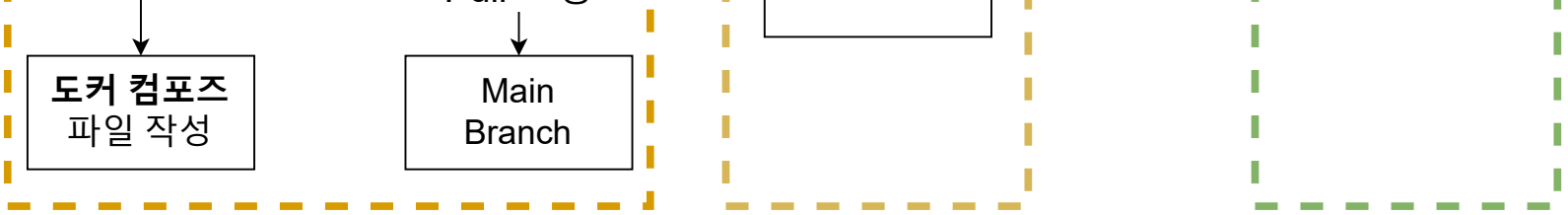
간단한 어플을 실제로 배포해보기(개발 부분)

이번 섹션 설명

지금부터는 4개의 섹션을 (섹션 6, 7, 8, 9) 통해 2가지 어플리케이션을 만들어보려 합니다. 하나는 간단하게 리액트만을 이용해서 처음부터 배포까지 도커를 이용해서 해볼게요. (Single Container) 그리고 두 번째는 리액트와 노드 그리고 데이터 베이스를 이용해서 처음부터 배포까지 하는 걸 해보며 도커를 배우는 시간을 갖겠습니다. (Multi Container)

먼저는 처음부터 배포까지 하는데 어떠한 흐름인지 보겠




도커 컴포즈
파일 작성
Main
Branch

리액트 앱 설치하기

전 시간에 본 흐름을 토대로
아주 간단한 리액트 앱을 만들어 보겠습니다.

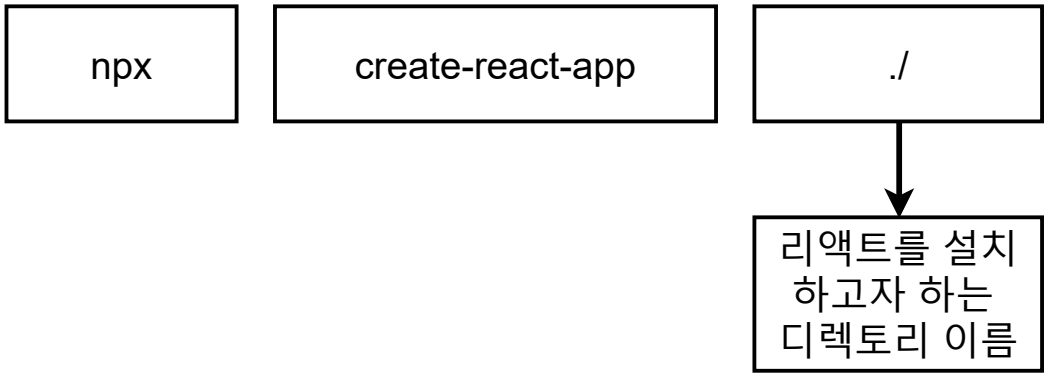
먼저는 리액트를 다운 받기 위해서
노드가 컴퓨터에 다운받아져있어야 합니다.

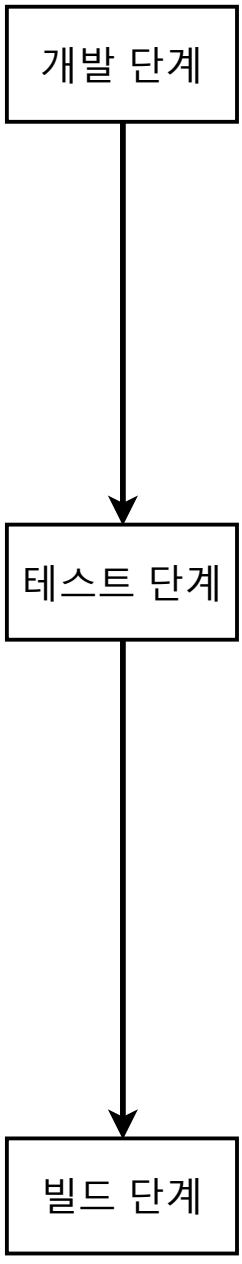
이미 컴퓨터에 노드가 받아져있는지 확인 하기 위해서는

만약 다운이 안받아져 있다면 node 공식 웹사이트에 가서

노드가 받아져 있다면 이제는 리액트를 설치할 차례입니다.
리액트를 다운받아서 설치하기 위해서는 굉장히 간단하게
아래에 보이는 명령어를 입력하면 됩니다.

리액트를 설치하기 위한 명





리액트 애플리케이션을 실행할 때는 아래
명령어를 입력하시면 됩니다.

npm run start

개발이 완료되었다면
개발한 것에 문제가 있는지 없는지
테스트를 해볼 수 있습니다.
테스트는 아래 명령어로 할 수 있습니다.

npm run test

이렇게 테스트까지 완성이 된다면 이제는 배포를 해서
다른 사람들도 이용할 수 있게 해 줘야겠죠.
그러기 위해서는 배포를 위한 폴더와 파일들을 생성해
야 합니다.
생성하기 위해서는 아래 명령어를 입력하시면 됩니다.

npm run build

이렇게 배포를 위한 명령어까지 작성하면

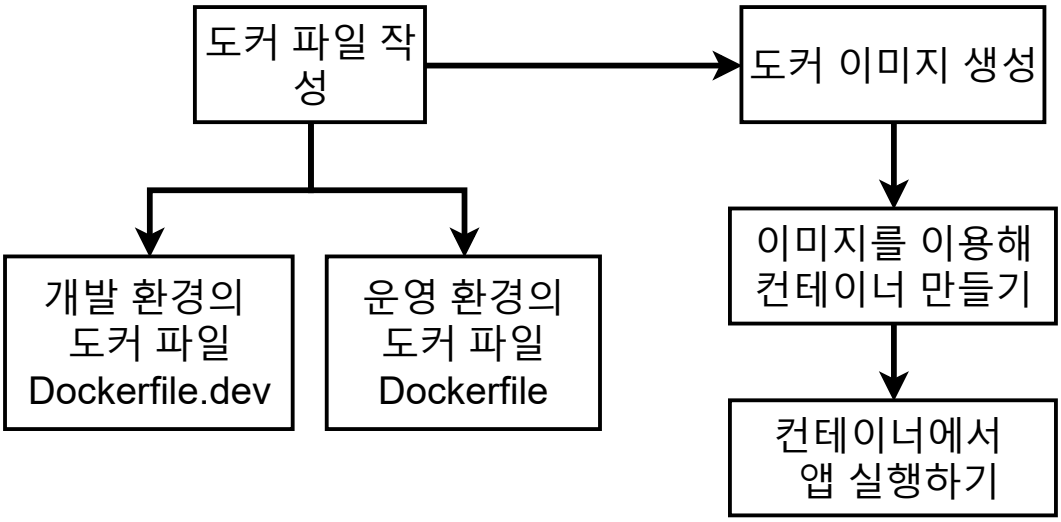
배포를 할 때 사용할 수 있는 build폴더와

그 안에 많은 파일들이 생성이 됩니다.

이렇게 기본적인 리액트 애플리케이션 설치를 완성했습

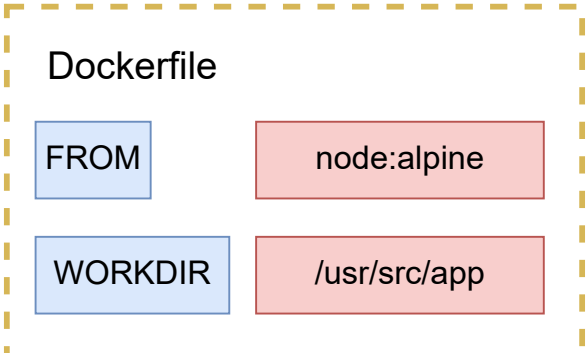
도커를 이용하여 개발단계에서 리액트 실행하기

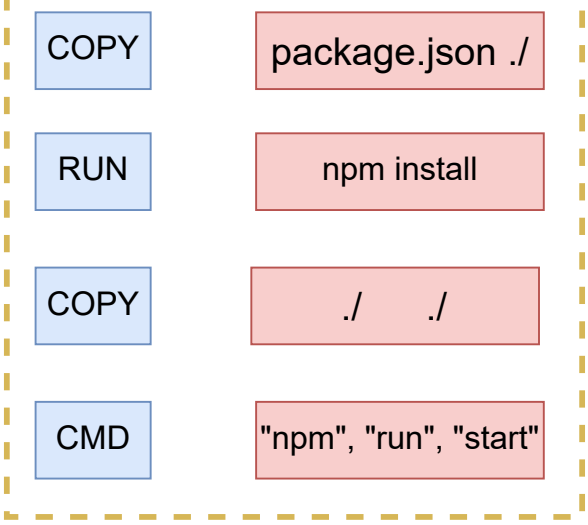
도커로 어플을 실행하기 위해서



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.dev라는 파일을 작성해보겠습

Dockerfile.dev





개발 환경에서의 도커 파일 작성은

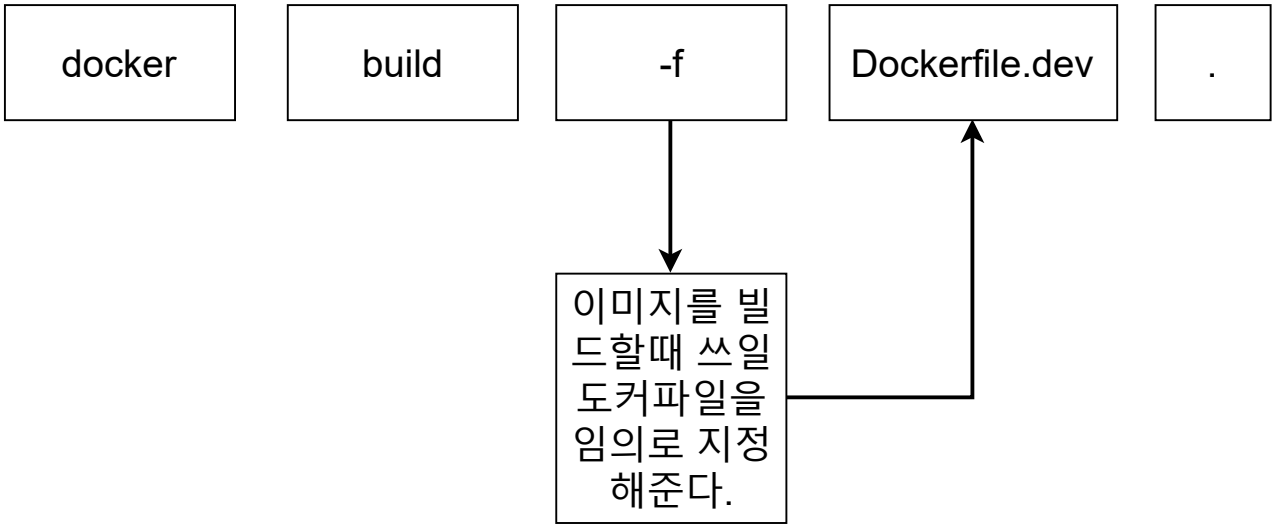
현재까지 도커 파일 작성했던 것과

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 할때 그냥 docker build .으로 하는게 아니라 이 아래 처럼 해줘야 한다.

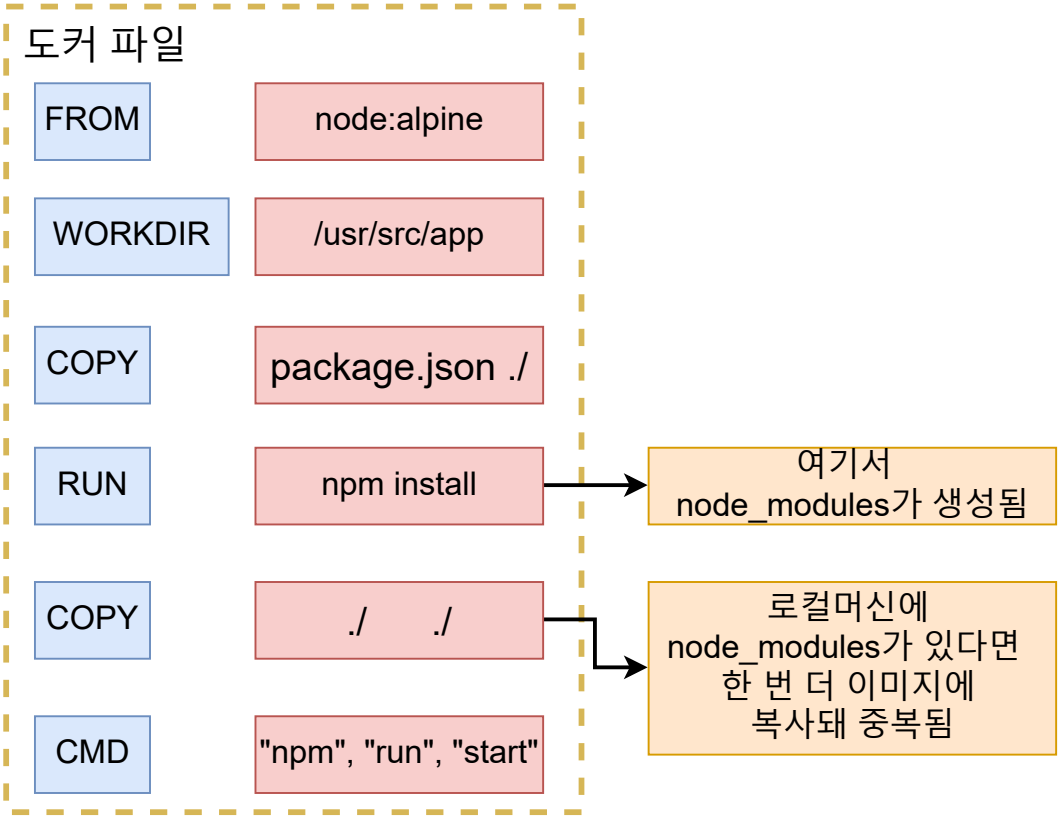


그래서 이런 식으로 -f 옵션을 이용하여서 다시 해보면
개발단계에서 리액트를 실행하게 해 줄
이미지 빌드가 가능해졌습니다.

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 다운 받기 때문에
굳이 로컬 머신에 node_modules을 필요로 하지 않는

Dockerfile.dev

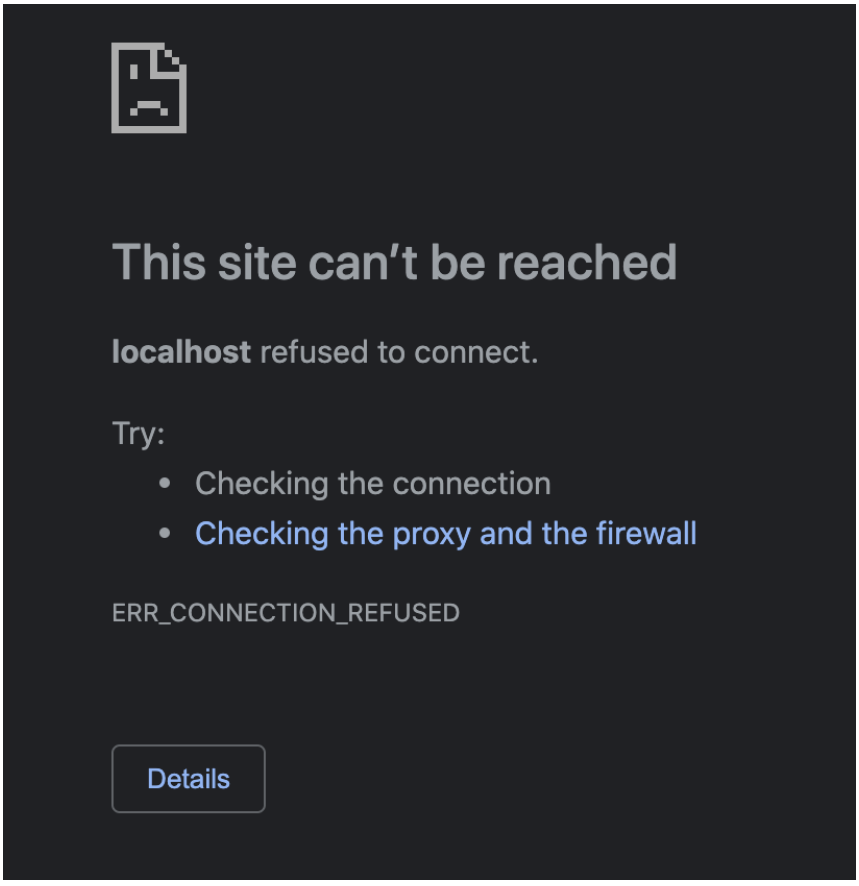


생성된 도커 이미지로 리액트 실행해보기

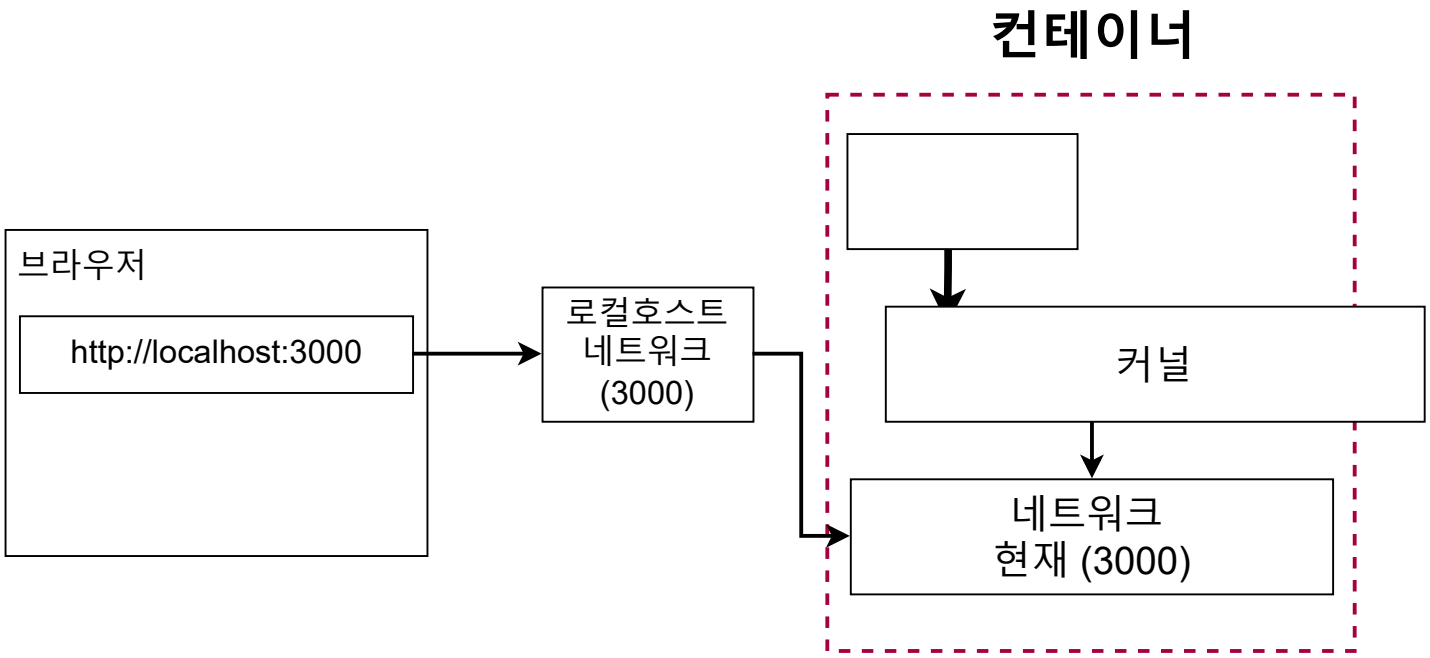
**전 강의에서 리액트를 실행할 수 있는 도커 이미지를 생성했습니다.
그래서 그 이미지로 컨테이너를 실행 해서**

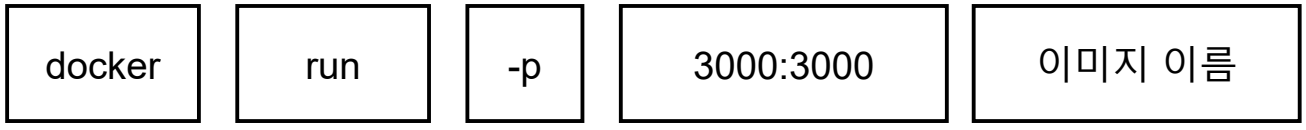
docker	run	이미지 이름

리액트는 기본적으로 3000번 포트에서 실행이 되기 때문에

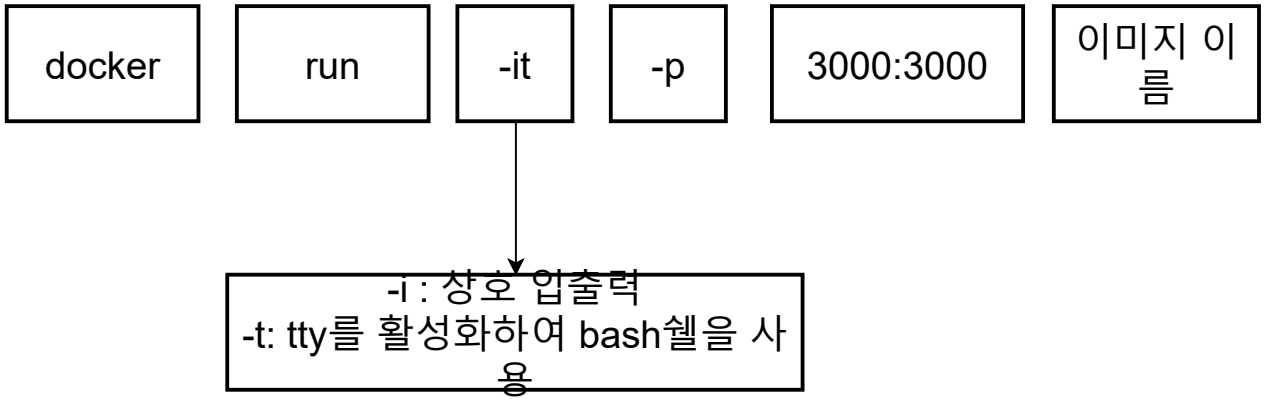


**이유는 예전에 했던 것과 같이 포트 매핑을 해줘야 하는데
안 해줬기 때문에 컨테이너 안에서 실행되고 있는 리액**





원래는 이렇게 해서 실행되어야 하는데
리액트 쪽에서 업그레이드로 인해
-it 붙여야만 실행 가능하게 됨.

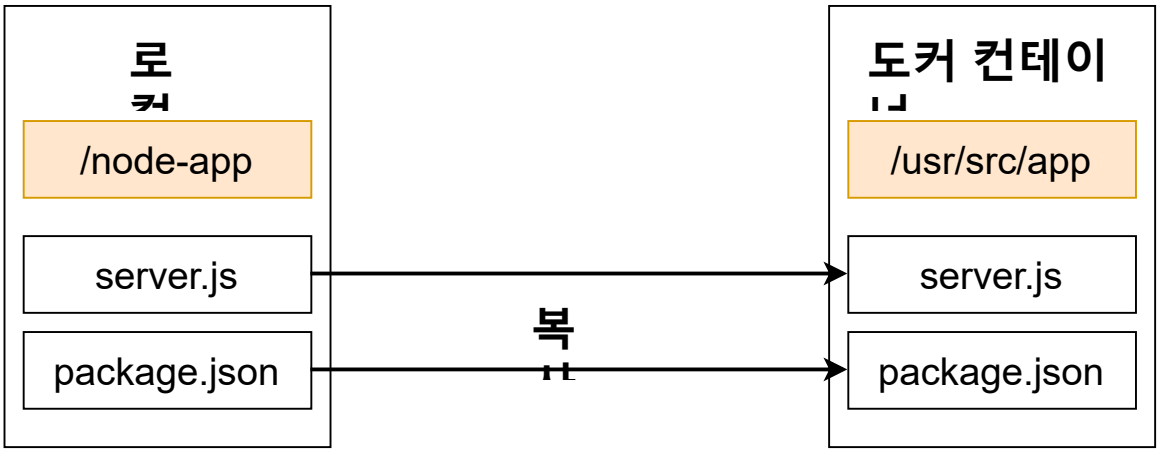


볼륨을 이용한 소스 코드 변경

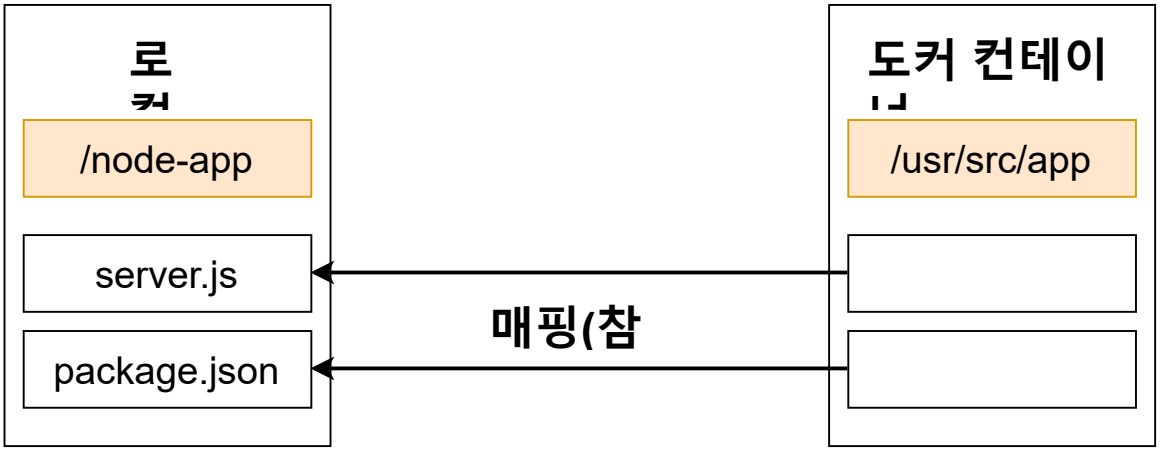
예전 강의에서 이미 COPY대신 VOLUME을 이용하므로 소스를 변경했을 때 다시 이미지를 빌드하지 않아도 변경한 소스 부분이 애플리케이션에 반영되는 부분을 해보았습니다.

COPY와 Volume의 차이 복

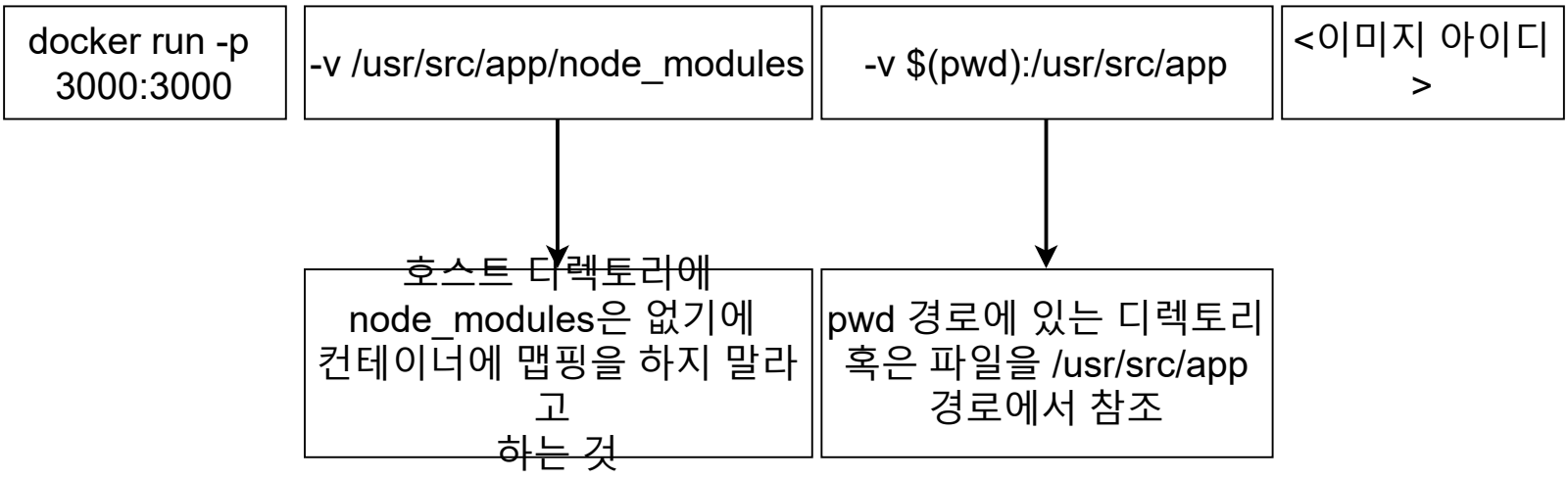
COPY



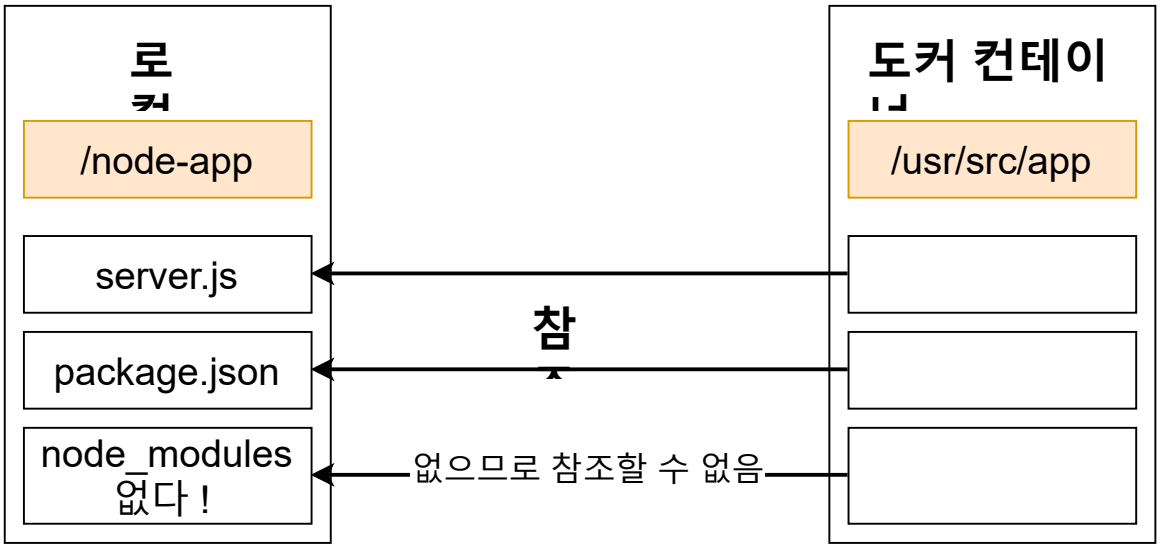
Volume



Volume 사용해서 어플리케이션 실행하

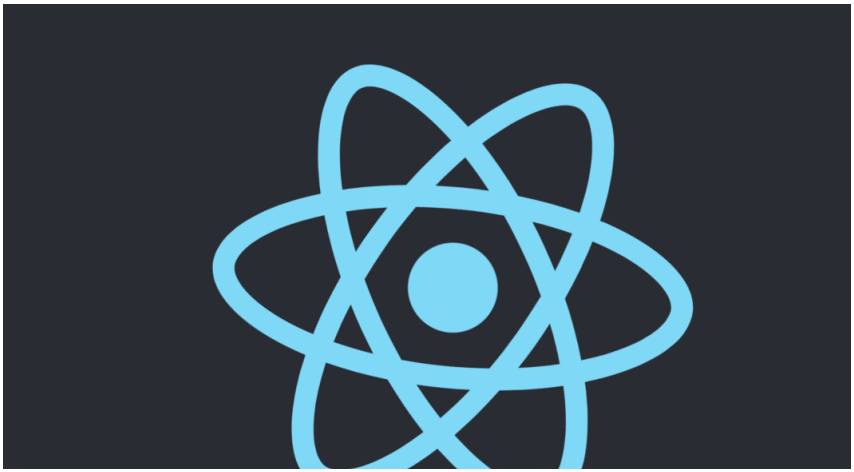


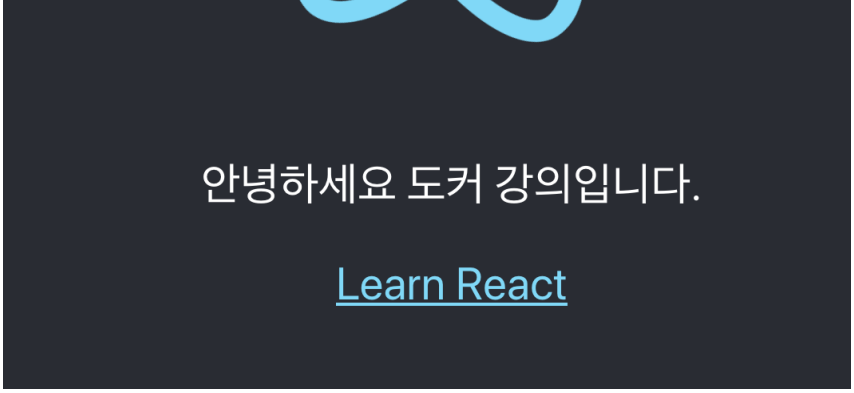
Volume



이렇게 Volume까지 설정했다면 이제는

소스를 변경해서 바로 변경된 것이 반영이 되는지를 확





안녕하세요 도커 강의입니다.

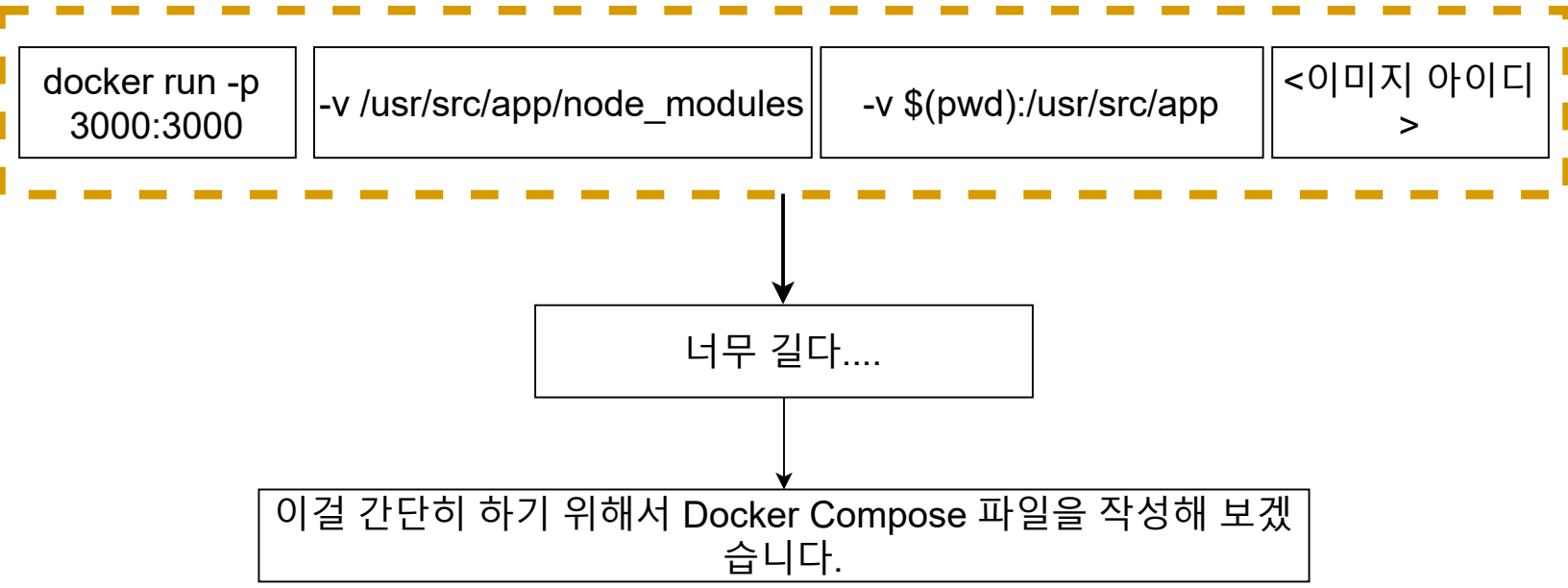
Learn React

도커 컴포즈로 좀 더 간단하게 앱 실행하기

앞서 리액트 앱을 실행할 때 너무나 긴 명령어를 치는 게

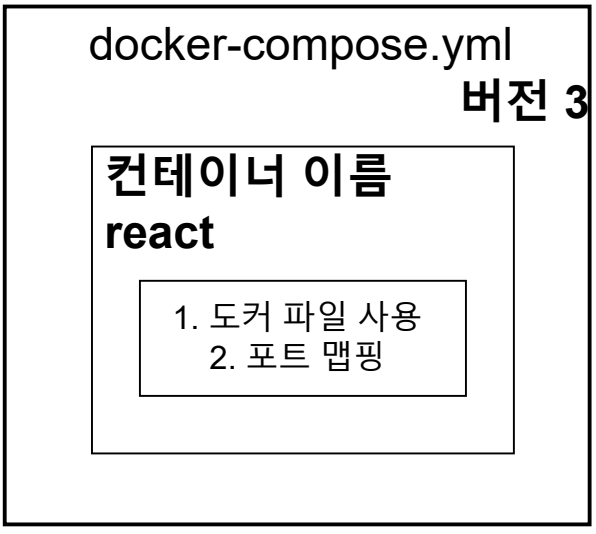
많이 불편했었습니다.

그러한 불편을 해소하기 위해서 도커 Compose를 이용해서

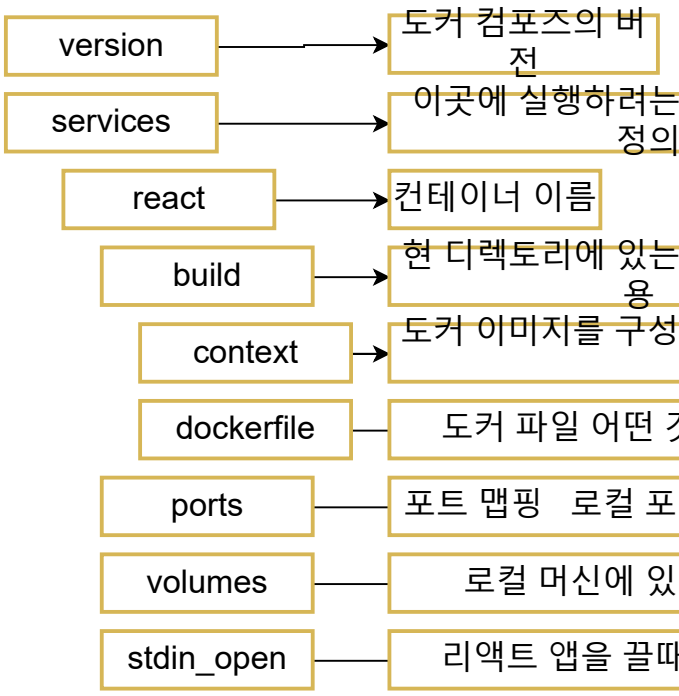


1.먼저 docker-compose.yml 파일 생성

2.docker-compose.yml 파일 작성하기



```yaml
version: "3"
services:
  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 애플리케이션 실

docker-compose	up

컨테이너들을

Dockerfile 사

하기 위한 파일과 폴더들이 있
는 위치

것인지 지정

트 : 컨테이너 포트

는 파일들 맵핑

필요(버그 수정)

리액트 앱 테스트 하기

보통 리액트 앱에서 테스트를 진행하려

```
npm run test
```

도커를 이용한 리액트 앱에서 테스트를 진행하려

이미지 생

```
docker build -f dockerfile.dev .
```

앱 실

```
docker run -it 이미지 이름 npm run test
```

```
PASS src/App.test.js
  ✓ renders learn react link (27ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.612s
Ran all test suites.

Watch Usage
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

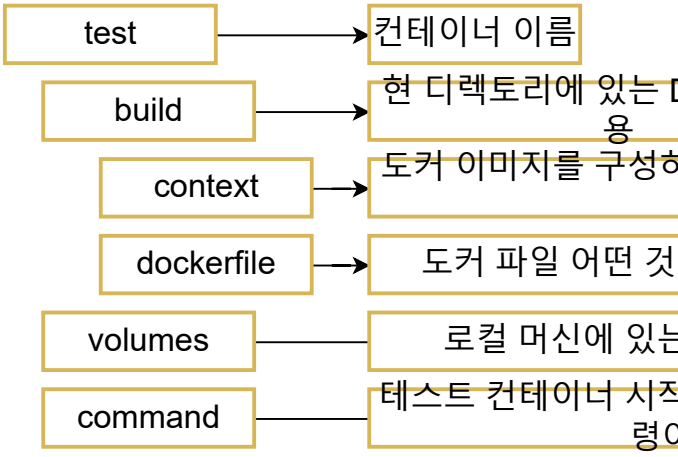
테스트도 소스 코드 변경하면 자동으로 반영되는 것처럼

테스트 소스도 추가하면 바로 반영되었으면 좋겠다!

소스 코드 변경을 위해서 Volume을 이용하였듯이
이번에도 Volume을 이용하지만 Test를 위한 컨테이너를 Compose 파일에 하나 더 만들어 주면 됩니다.

docker-compose.yml에 이 부분 추

```yaml
tests:
  build:
    context: .
    dockerfile: Dockerfile.dev
  volumes:
   - /usr/src/app/node_modules
   - ./:/usr/src/app
  command: ["npm", "run", "test"]
```



이렇게 되면, 앱을 시작할 때 두 개의 컨테이너를 다 시작하게 되니,

그러면 먼저 두 개의 컨테이너를 모두 다 실행을 시켜보겠습니

docker-compose	up	--build

이렇게 실행시킨 후 테스트를 변경시켜보겠습니다.

그러면 자동으로 변경된 부분도 다시 테스트하게 됩니다.

Dockerfile 사

하기 위한 파일과 폴더들이 있
는 위치

인지 지정

는 파일들 맵핑

작할때 실행 되는 명
어

운영환경을 위한 Nginx

현재까지는 리액트 앱을 개발 환경에서 다뤄보았

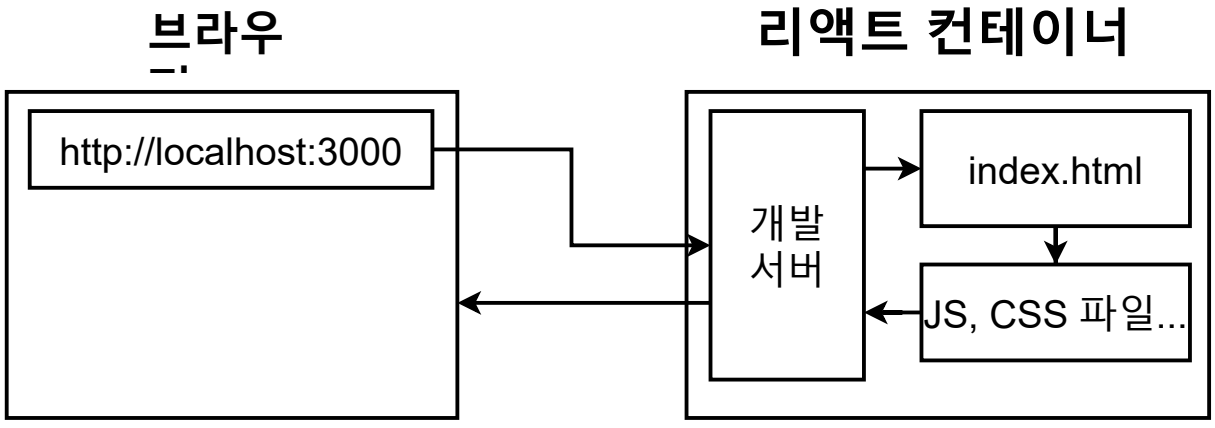
그래서 이제는 운영 환경(배포 후)을

하나하나 다뤄보려고 하는데요.

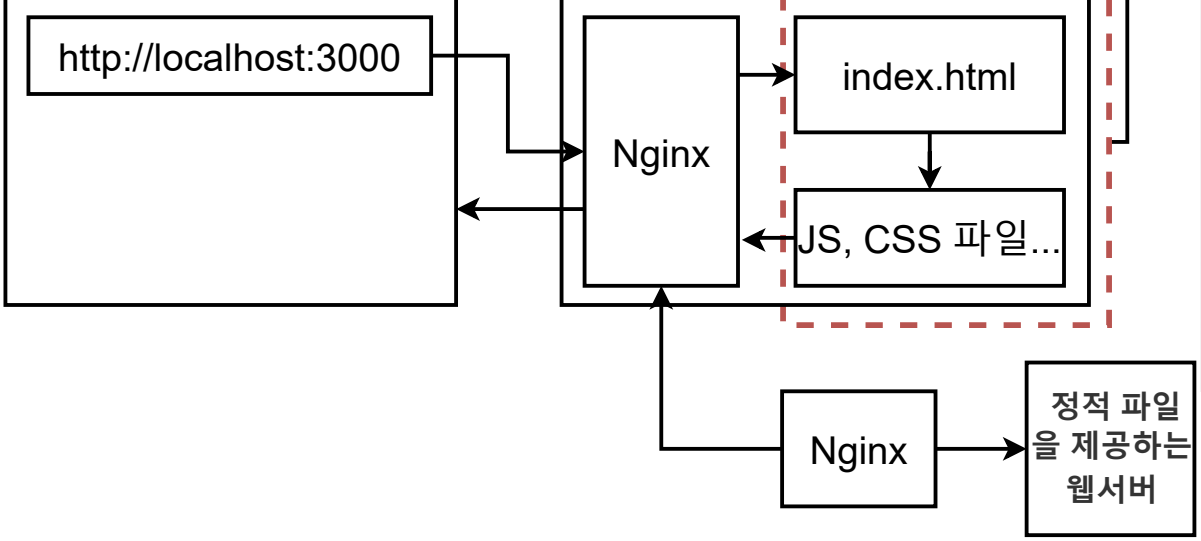
먼저는 Nginx라는 것을 살펴보겠습니다.

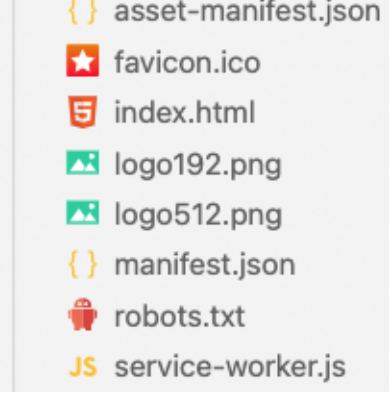
그래서 우선 Nginx가 왜 필요한지 봐보겠습니다.

개발 환경에서 리액트가 실행되는



운영 환경에서 리액트가 실행되는





Tips 왜 개발환경 서버와 운영환경 서버를 다른 거 써야

개발에서 사용하는 서버는 소스를 변경하면 자동으로 전체 앱을

다시 빌드해서 변경 소스를 반영해주는 것 같이 개발 환경에 특화된 기능들이 있기에

그러한 기능이 없는 Nginx 서버보다 더욱 적합합니다.

그리고 운영환경에서는 소스를 변경할 때

다시 반영해 줄 필요가 없으며

개발에 필요한 기능들이 필요하지 않기에 더 깔끔하고 빠

운영환경 도커 이미지 위한 Dockerfile 작성하기

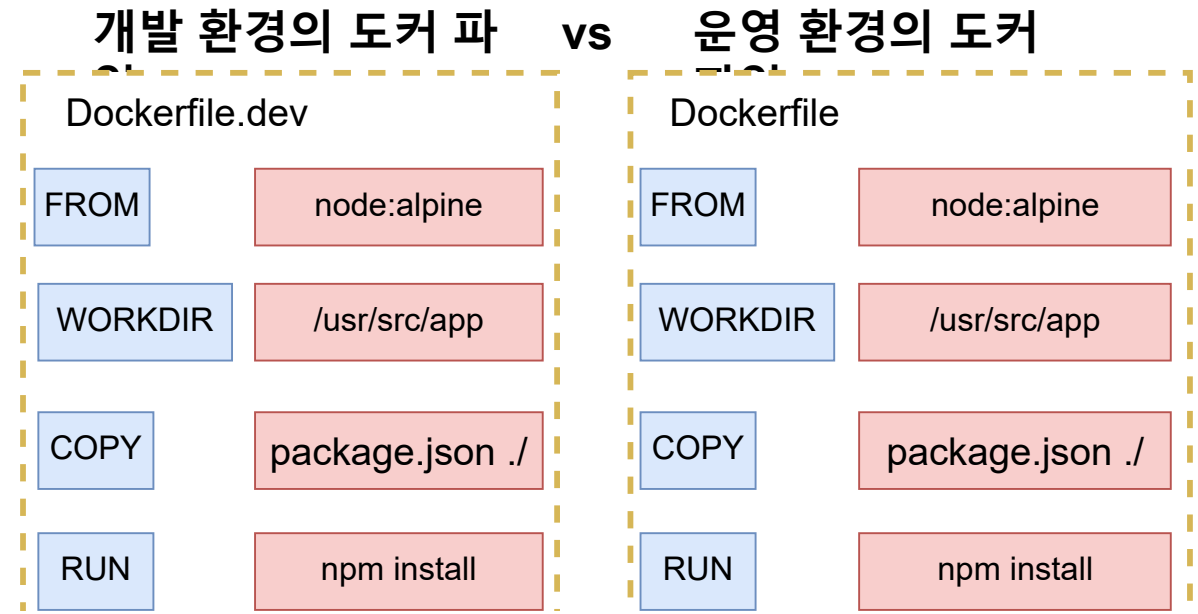
전 시간을 통해서 운영환경에 Nginx가 필요한 걸 알게 되었습니다. 그래서 이제는 Nginx를 포함하는 리액트 운영환경 이미지를 생성해보겠습니다.

이미지를 생성하기 위해서 Dockerfile을 작성하겠습니

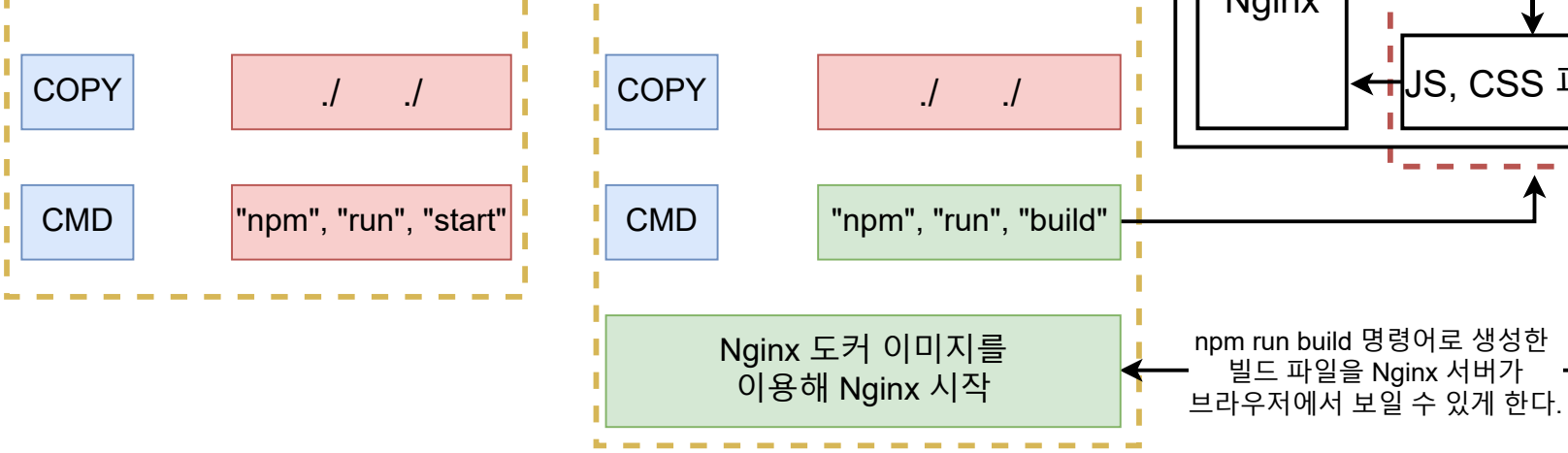
리액트 개발환경에서 필요한 이미지를 생성하기 위해서는 → Dockerfile.dev

리액트 운영환경에서 필요한 이미지를 생성하기 위해서는 → Dockerfile

개발환경 도커파일과 운영환경 도커 파




tml



개발환경과 운영환경의 도커 파일 차이점은

개발 환경에서는 build를 할 필요 없이 실행이 가능 하지만 운영 환경에서는 build를 해줘야 하므로

CMD에 npm run build로 빌드 파일들을 생성해주며 그 이후에 Nginx를 시작해줘야 합니다.

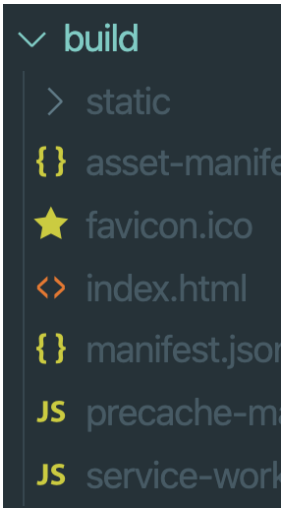
운영환경 도커 파일 자세히

운영환경을 위한 Dockerfile을 요약하자면 2가지 단계로 이루어져 있다.

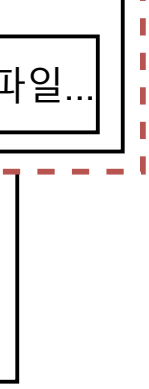
첫 번째 단계는 빌드 파일들을 생성합니다. (Builder Stage)

두 번째 단계는 Nginx를 가동하고 첫 번째 단계에서 생성된 빌드 폴더의 파일들을 웹 브라우저의 요청에 따라 제공하여 준다. (Run Stage)

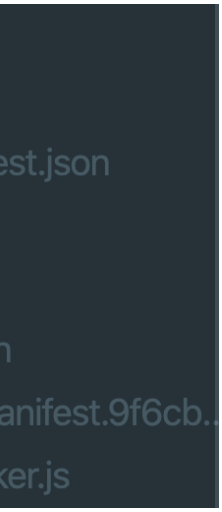
빌드해서 생성된




```
FROM node:alpine as builder
WORKDIR '/usr/src/app'
COPY package.json .
RUN npm install
COPY ./ ./
RUN npm run build
```



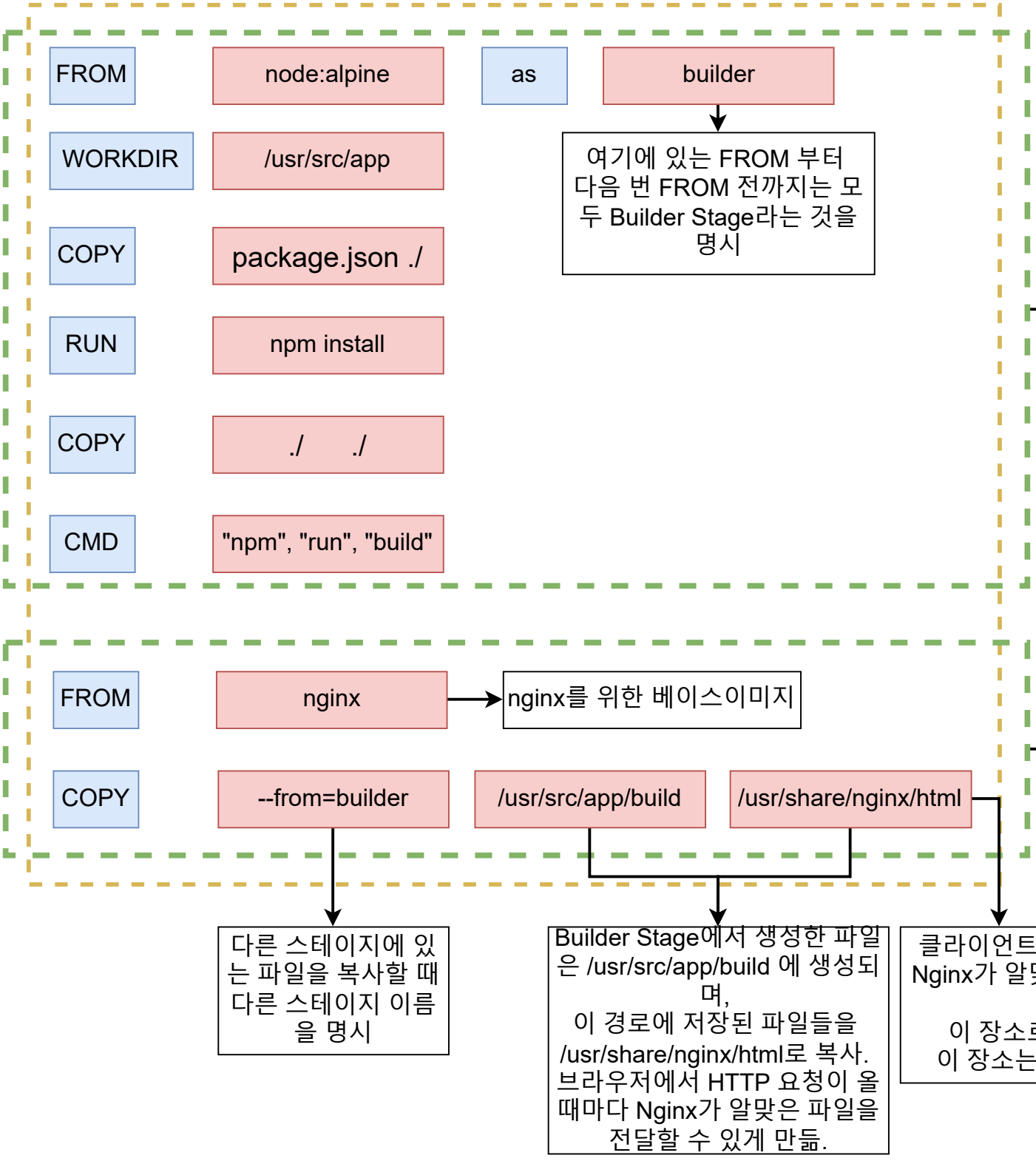
성



```
RUN npm run build

FROM nginx
COPY --from=builder /usr/src/app/build /usr/share/nginx/html
```

도커 파일



Builder Stage

Builder Stage의 목표는
빌드 파일을 생성하는 것

생성된 파일과 폴더들은
usr/src/app/build에 생성됨

Run stage

Run Stage의 목표는
Nginx를 이용해
웹 브라우저의 요청에 따라
정적 파일들을 제공해주는
것

요청이 들어올 때마다
정적 파일을 제공하기
위해
드된 파일을 복사.
정을 통해 변경 가능.

Dockerhub 공식 nginx 설

```
FROM nginx
COPY static-html-directory /usr/share/nginx/html
```

이렇게 다 했다면 다 작성된 Dockerfile로 이미지를 생성해

docker	build	-t 이미지이름	.

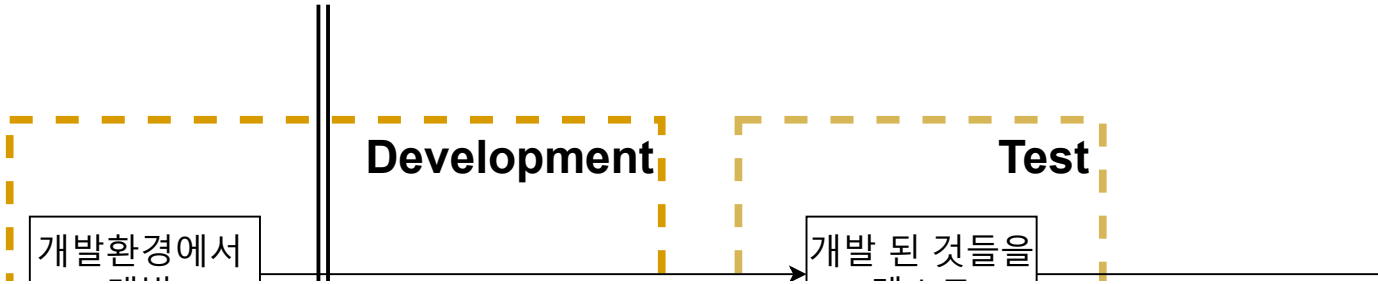
이미지를 생성 했다면 그 이미지를 이용해서 앱을 실행해보

docker	run	-p	8080:80	이미지 이름

Nginx의 기본 사용포트는 80 입니다.

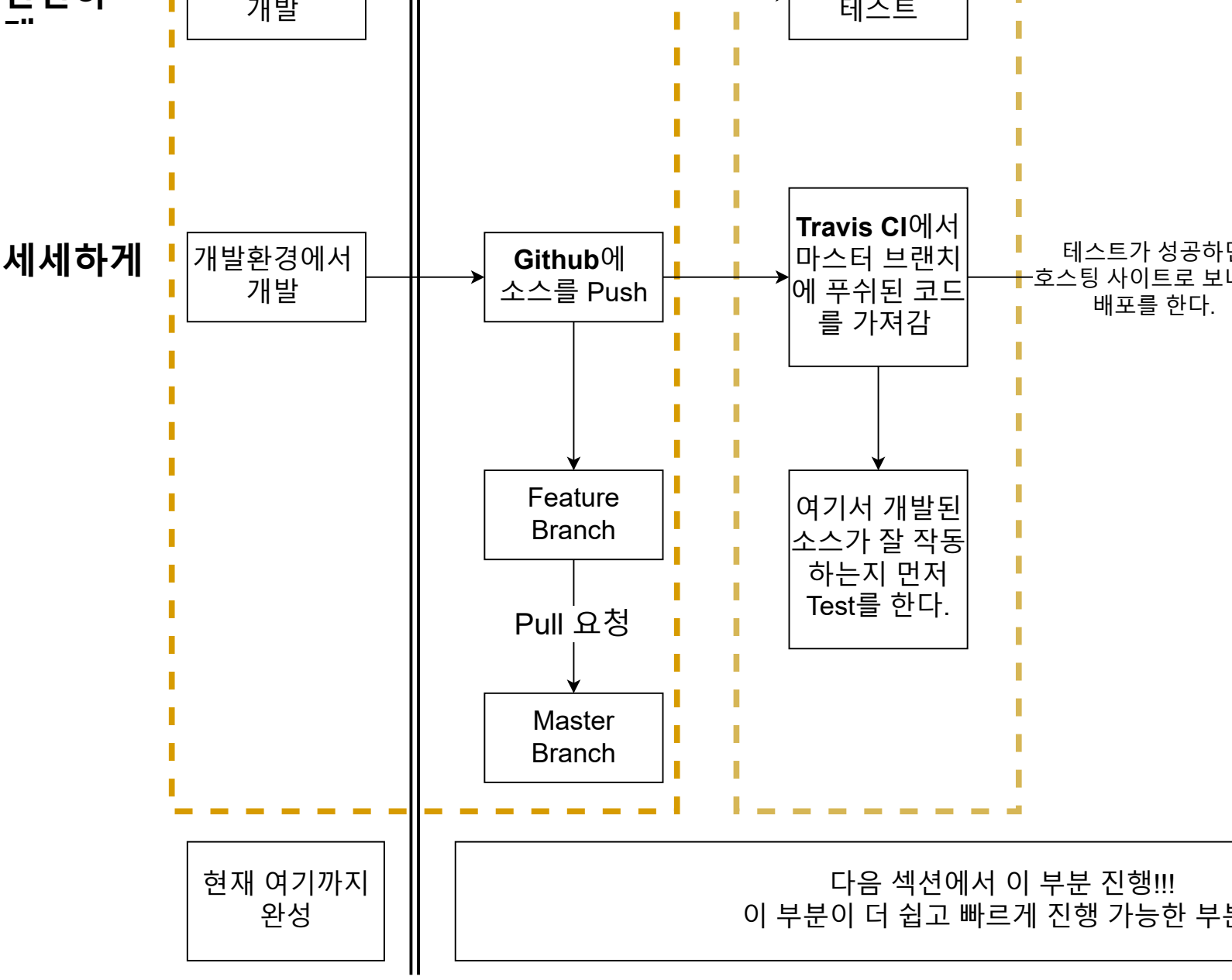
Nginx 서버를 이용한 운영 환경의 react 앱 실행 성공!!!

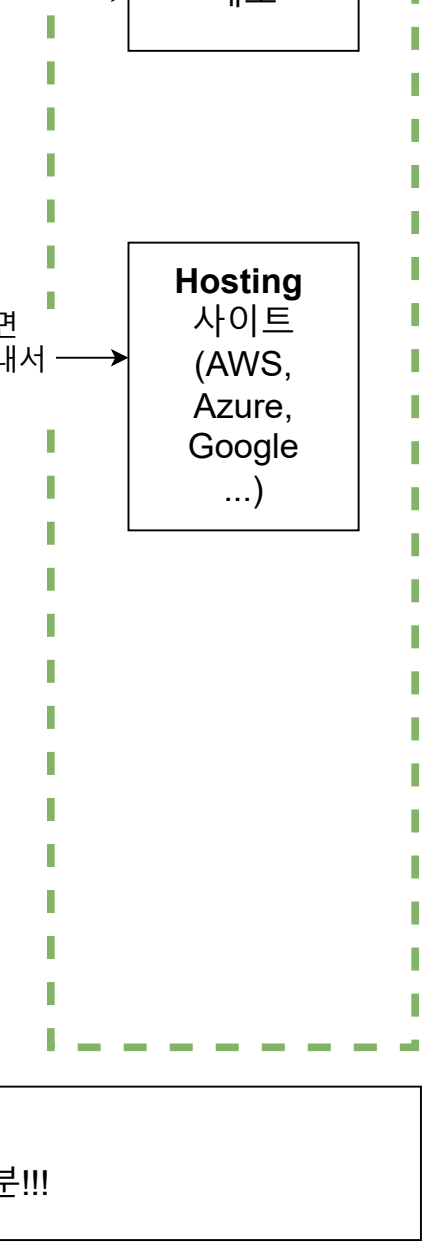
현재까지 진행 상황과 앞으로 해야 할 부분 보



간단하


Production
배포


개발
테스트
세세하게
개발환경에서
개발
Github에
소스를 Push
Travis CI에서
마스터 브랜치
에 푸쉬된 코드
를 가져감
테스트가 성공하면
호스팅 사이트로 보내
배포를 한다.
Feature
Branch
Pull 요청
Master
Branch
여기서 개발된
소스가 잘 작동
하는지 먼저
Test를 한다.
현재 여기까지
완성
다음 섹션에서 이 부분 진행!!!
이 부분이 더 쉽고 빠르게 진행 가능한 부분


Hosting
사이트
(AWS,
Azure,
Google
...)