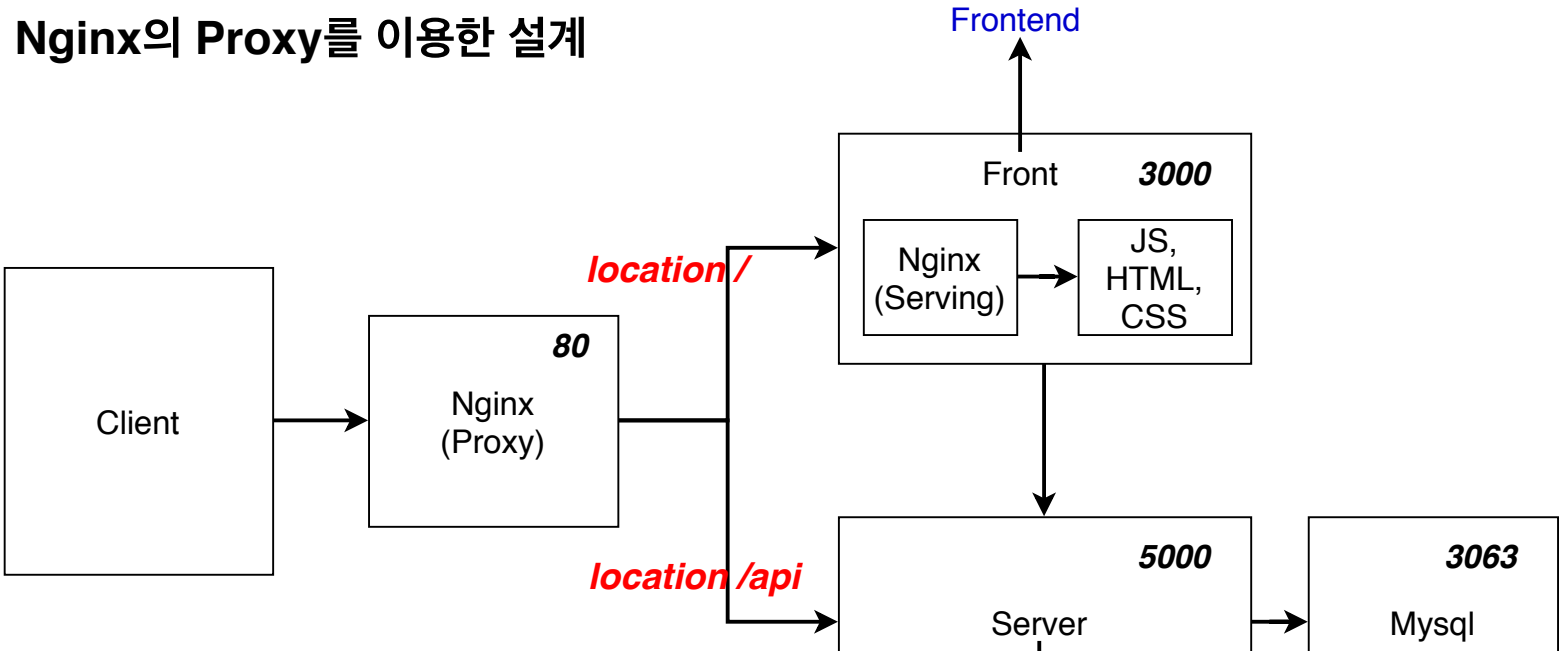
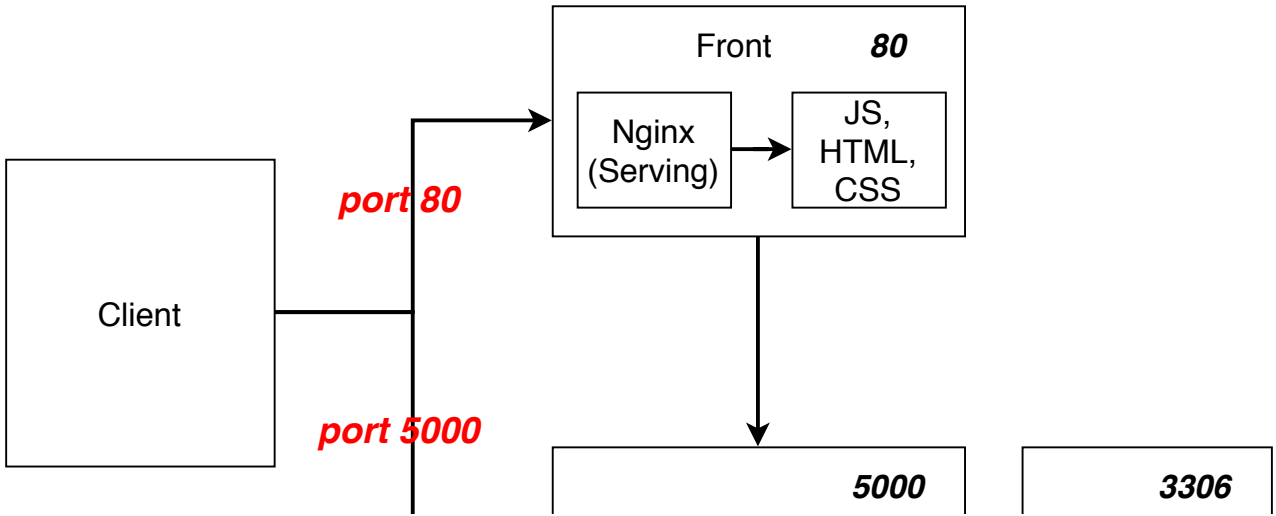
# Docker를 이용한 복잡한 어플리케이션 만들기

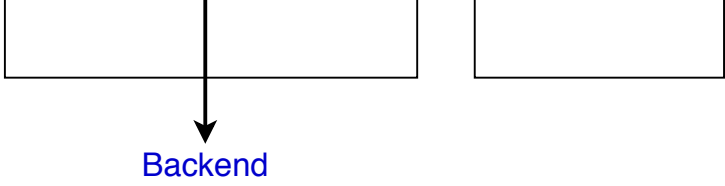
## 섹션 설명


## Nginx의 Proxy를 이용한 설계



## Nginx는 정적파일을 제공만 해주는 설계






## 구현 순서

*전체 소스 코드 작성* → *Dockerfile 작성* → *Do*
*compo*

NodeJS

React

**개발 환경**
**Dockerfile.dev**

Front

Server

Mysql

Nginx

**운영 환경**
**Dockerfile**

Front

Server

Mysql

Nginx

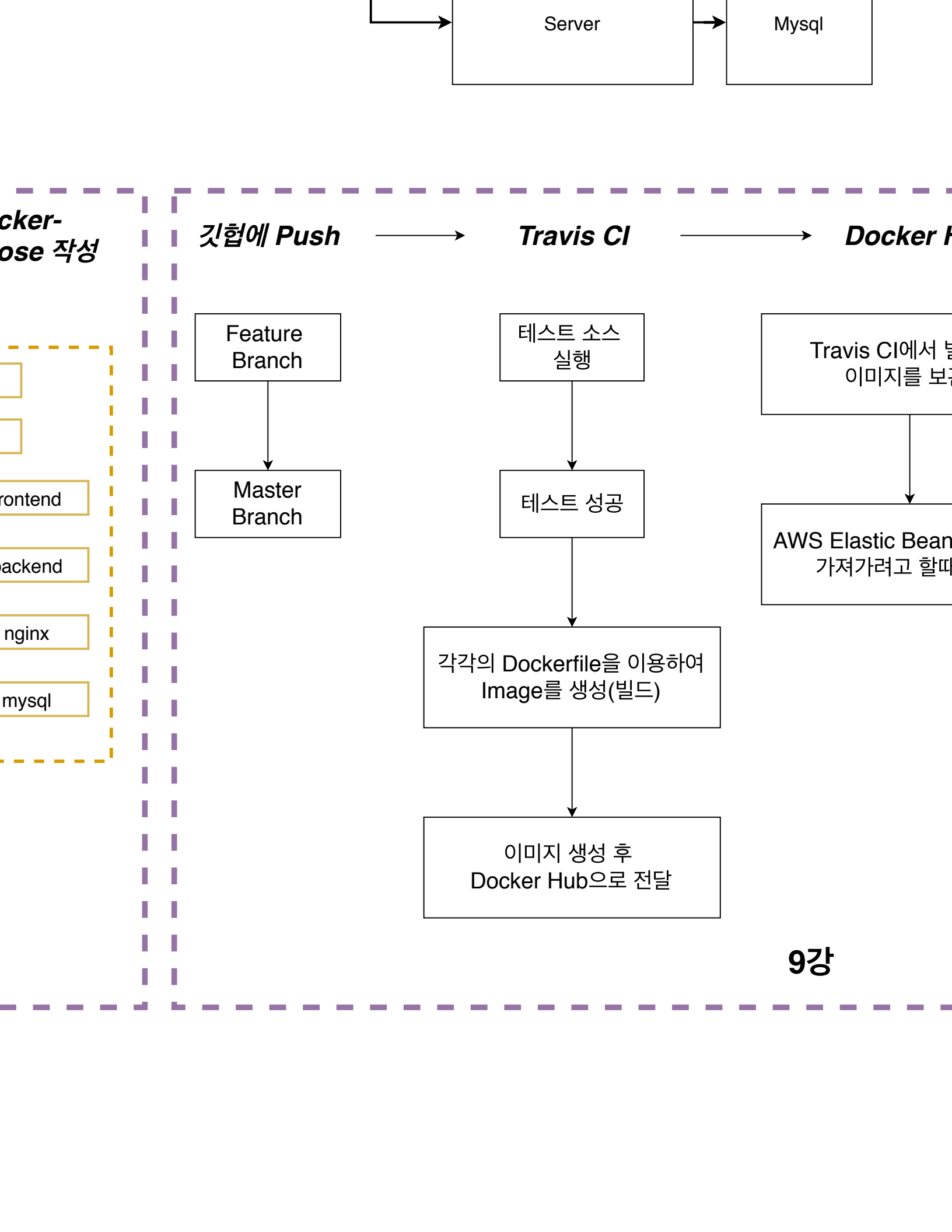
version

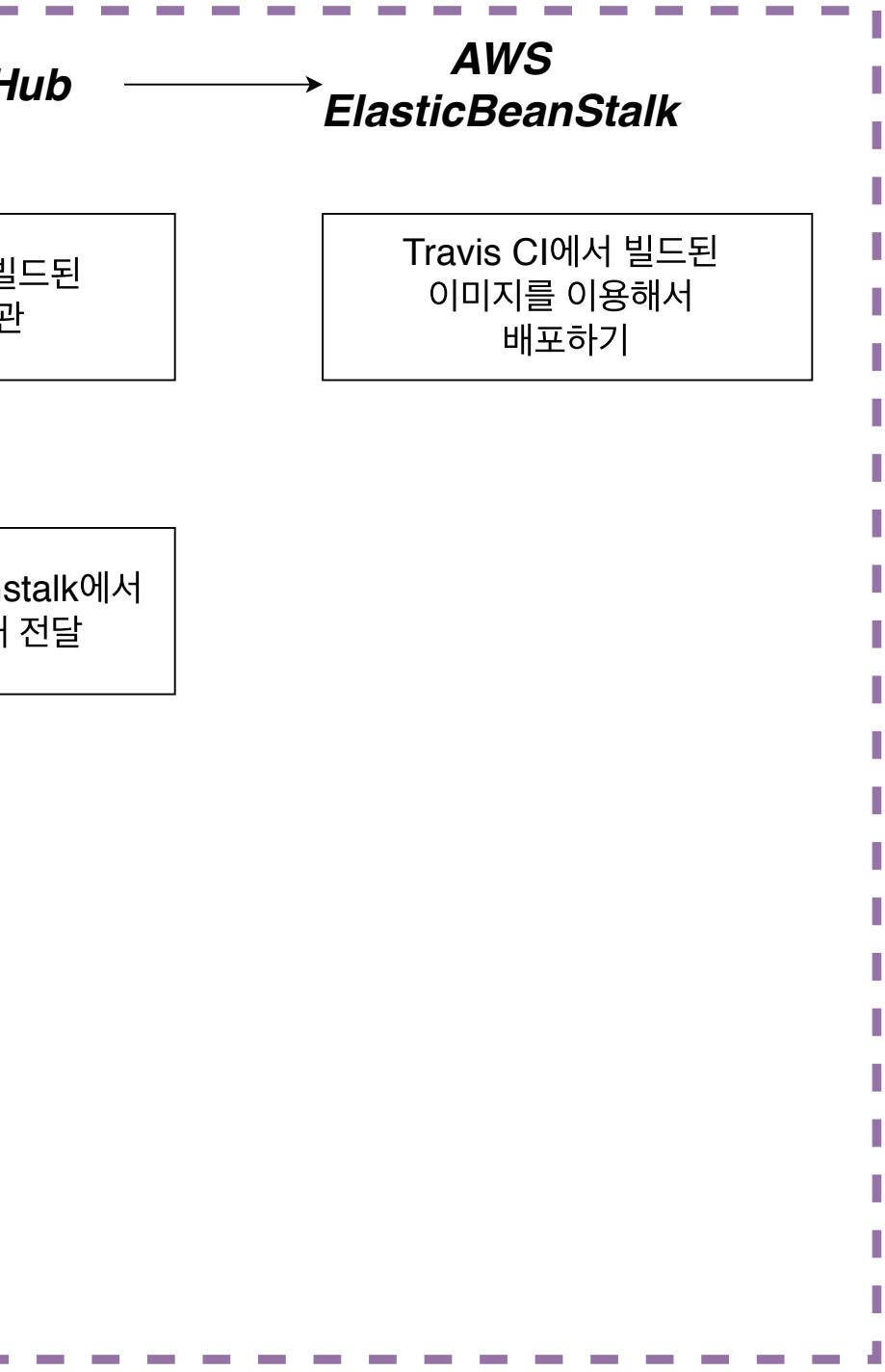
services

f

b

**8강**


Server
Mysql
cker-
ose 작성
깃헙에 Push
Travis CI
Docker H
Feature
Branch
Master
Branch
테스트 소스
실행
테스트 성공
각각의 Dockerfile을 이용하여
Image를 생성(빌드)
이미지 생성 후
Docker Hub으로 전달
Travis CI에서 빌
이미지를 보관
AWS Elastic Bean
가져가려고 할때
rontend
ackend
nginx
mysql
9강


Hub
AWS
ElasticBeanStalk
빌드된
관
Travis CI에서 빌드된
이미지를 이용해서
배포하기
stalk에서
서 전달

# Node JS 구성하기

1. backend 라는 폴더 만들기

2.package.json 파일 만들기

- test → test 코드를 실행할때 사용
- start → express 서버를 시작할때 사용
- dev → nodemon을 이용하여 express서버를 시작할때 사용

3. package.json 파일안에 스크립트와 사용할 모듈들 명시해주기

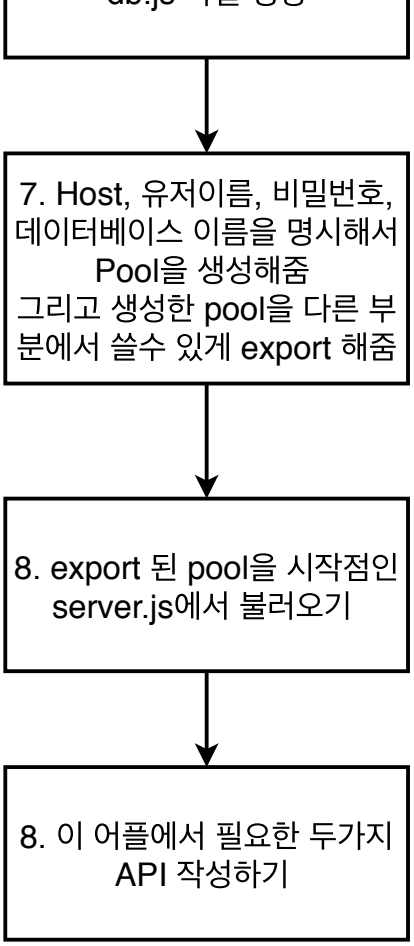
- express → 웹 프레임워크 모듈
- mysql → mysql을 사용하기 위한 모듈
- body-parser → 클라이언트에서 오는 요청의 본문을 해석해주는 미들웨어

4. 시작점이 되는 server.js 만들기

5. server.js의 기본 구조 작성

```
//필요한 모듈들을 가져오기
const express = require('express');
const bodyParser = require('body-parser');

// Express 서버를 생성
const app = express();

// json 형태로 오는 요청의 본문을 해석해줄수있게 등록
app.use(bodyParser.json());

// Express 서버 포트 5000에서 시작
app.listen(5000, () => {
  console.log('어플리케이션이 서버 5000번 포트에서 되었습니다.')
});
```

6. mysql을 연결하기 위한 db.js 파일 생성


db.js 파일 생성
7. Host, 유저이름, 비밀번호, 데이터베이스 이름을 명시해서 Pool을 생성해줌 그리고 생성한 pool을 다른 부분에서 쓸수 있게 export 해줌
8. export 된 pool을 시작점인 server.js에서 불러오기
8. 이 어플에서 필요한 두가지 API 작성하기

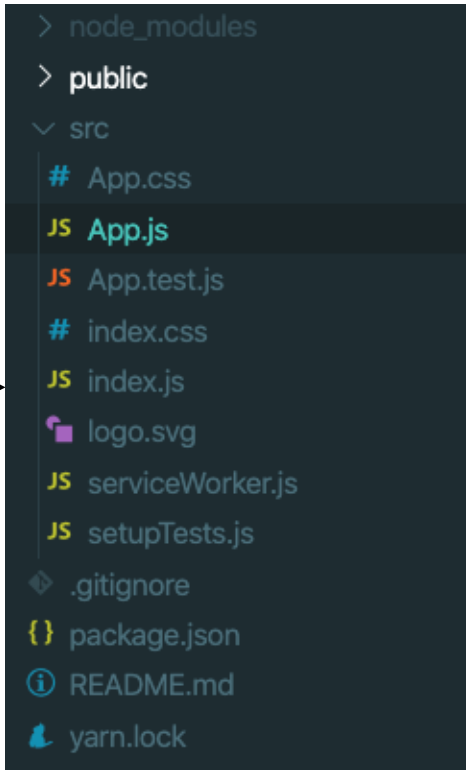
# React JS 구성하기

**이번 시간에는 빠른 속도로 React JS 부분을 구성해보겠습니다.**

1. Create-React-App으로 리액트 앱 생성하기

```
npx create-react-app frontend
```

2.이런 구조의 리액트 앱이 생성됨




3. 먼저는 App.js 파일안에 원하는 UI를 생성

Input 박스와 Button추가


4. UI를 위한 코드 작성

5. UI를 위한 CSS 작성

6. 데이터의 흐름을 위한
State 생성

```jsx
import React, { useState } from 'react'
import axios from 'axios';
import logo from './logo.svg';
import './App.css';

function App() {

  const [lists, setLists] = useState([])
  const [value, setValue] = useState("")
```

7. 데이터 베이스에서 데이터를
가져오기 위해
필요한 useEffect 넣어주기

```jsx
import React, { useState, useEffect } from 'react'
import axios from 'axios';
import logo from './logo.svg';
import './App.css';

function App() {

  const [lists, setLists] = useState([])
  const [value, setValue] = useState("")

  useEffect(() => {
    //여기서 데이터베이스에 있는 값을 가져온다.

  }, [])
```

# 리액트 앱을 위한 도커 파일 만들기

1.frontend 폴더안에
Dockerfile 생성

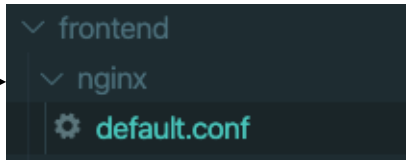
2. 개발 환경을 위한
Dockerfile 작성

```
FROM node:alpine

WORKDIR /app

COPY package.json ./

RUN npm install

COPY ./ ./

CMD [ "npm", "run", "start" ]
```

3. 운영 환경을 위한
Dockerfile 작성

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

FROM nginx
EXPOSE 3000
COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/build /usr/share/nginx/html
```

정적 파일을 제공해주기 위한
Nginx를 위해서
frontend안에 nginx 폴더 생성하고
default.conf 파일 생성

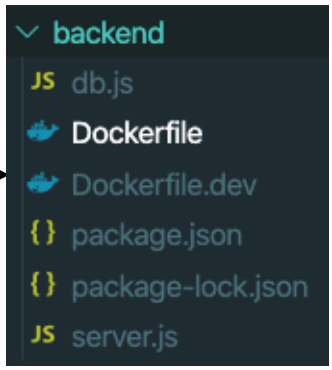



default.conf 파일 작성

```
server {
  listen 3000;

  location / {

    root /usr/share/nginx/html;

    index index.html index.htm;

    try_files $uri $uri/ /index.html;
  }
}
```

# 노드 앱을 위한 도커 파일 만들기

1. backend 폴더 안에 Dockerfile 생성




2. 개발 환경을 위한 Dockerfile 작성

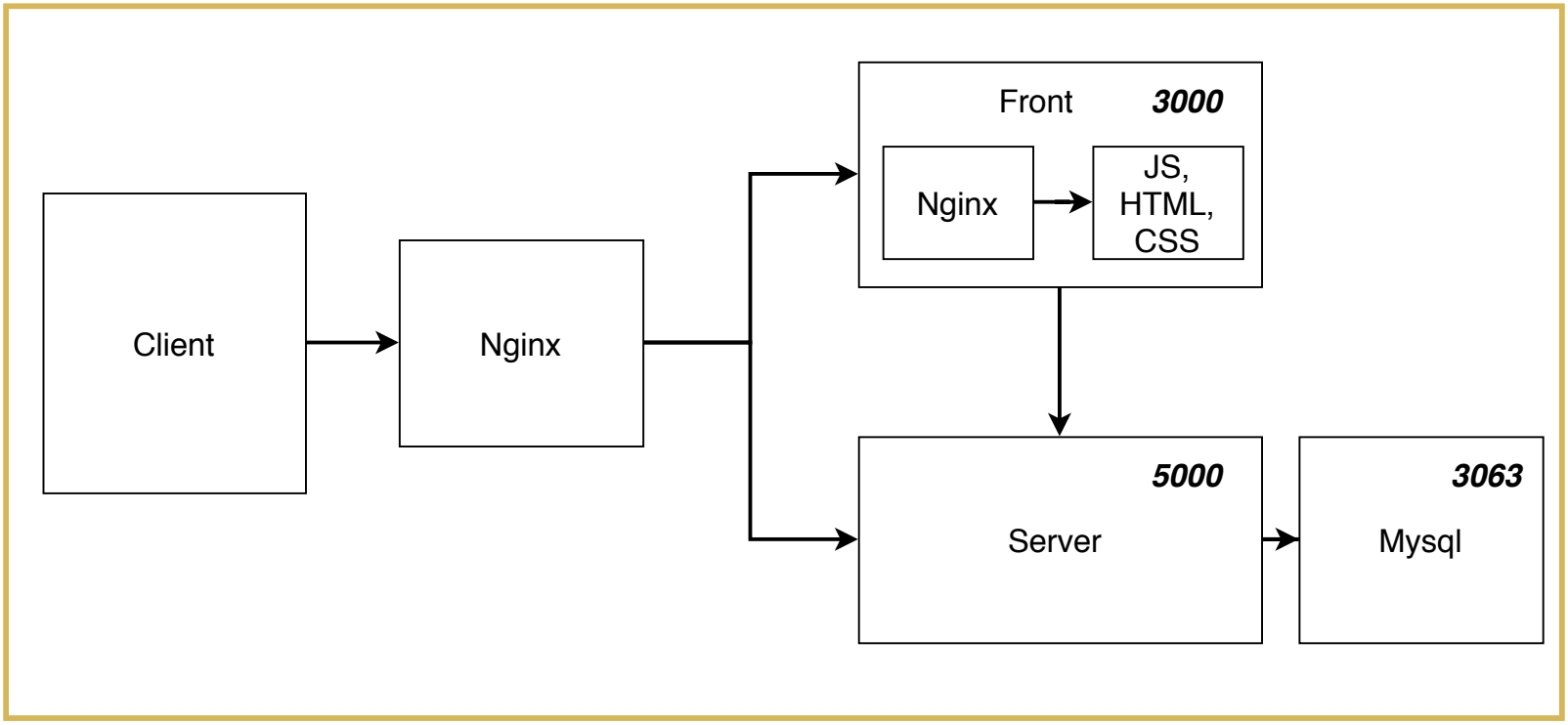
```
FROM node:alpine

WORKDIR /app

COPY ./package.json ./

RUN npm install

COPY . .

CMD ["npm", "run", "dev"]
```

3. 운영 환경을 위한 Dockerfile 작성

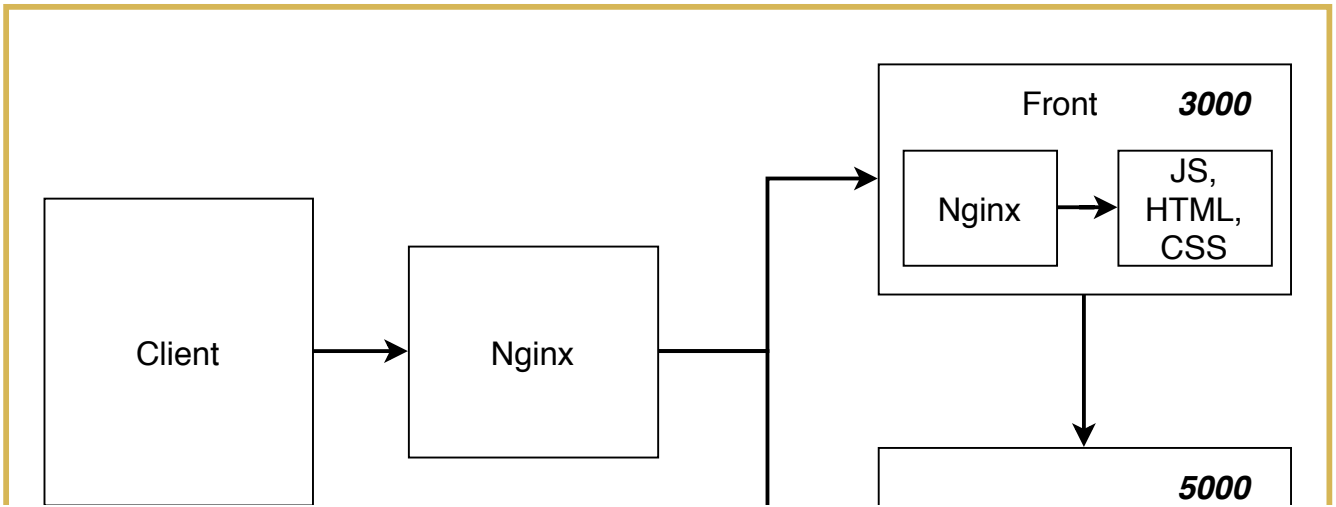
```
FROM node:alpine

WORKDIR /app

COPY ./package.json ./

RUN npm install

COPY . .

CMD ["npm", "run", "start"]
```

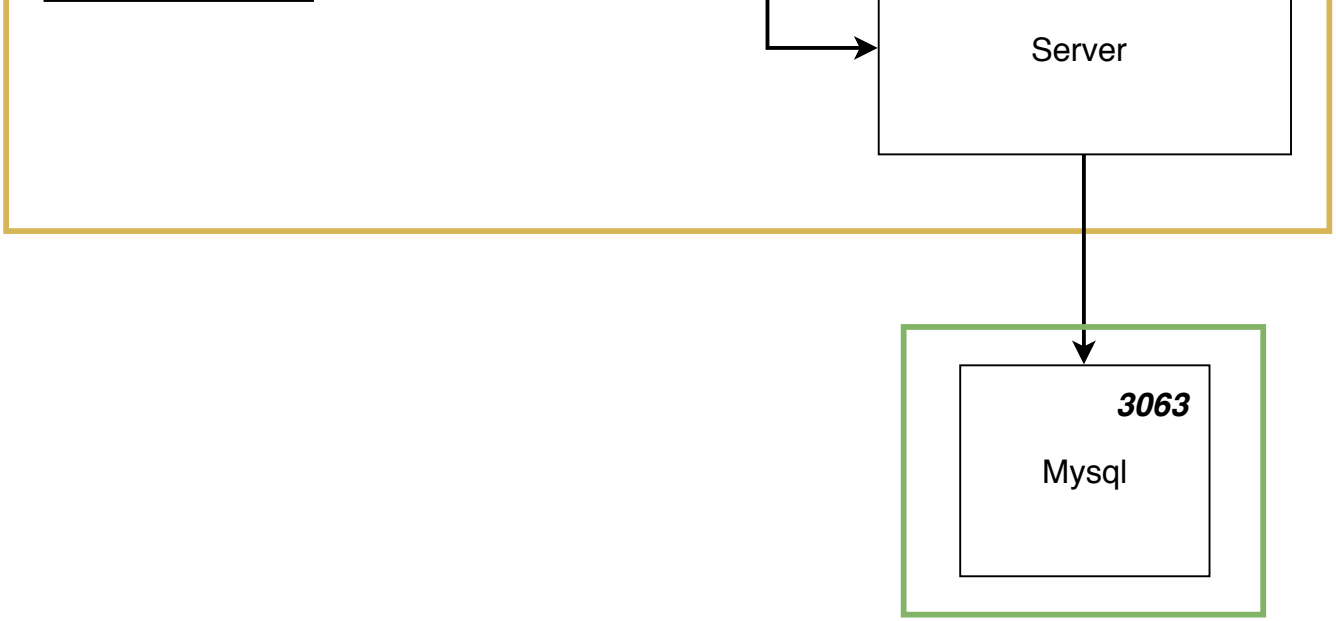
# DB에 관해서

개발 환경

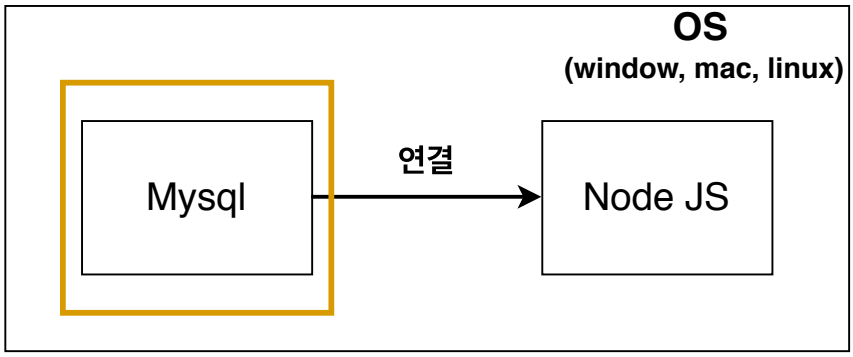



운영 환경




Server
3063
Mysql

# Mysql을 위한 도커 파일 만들기




1. mysql이라는 폴더를 Root 디렉토리에 생성한후 그 안에 Dockerfile 생성

2. Dockerfile 작성

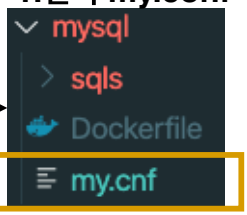
3. Mysql을 시작할때 Database와 Table이 필요한데 그것들을 만들 장소를 만들어 줍니다.

4. Database와 Table을 만들어줍니다.

```
DROP DATABASE IF EXISTS myapp;

CREATE DATABASE myapp;
USE myapp;

CREATE TABLE lists (
  id INTEGER AUTO_INCREMENT,
  value TEXT,
  PRIMARY KEY (id)
);
```

5. 마지막으로 한가지
더 해줘야 할게있습니다.
현재 상태에서 어떠한 글을 데이터베이스에 넣어줄때 한글이 깨지게 되어서 저장을 할때 오류가 나게 됩니다.
그래서 한글도 저장할수 있게 설정을 해주겠습니다.

**1.먼저 my.conf 라는 파일을 생성해준 후에**



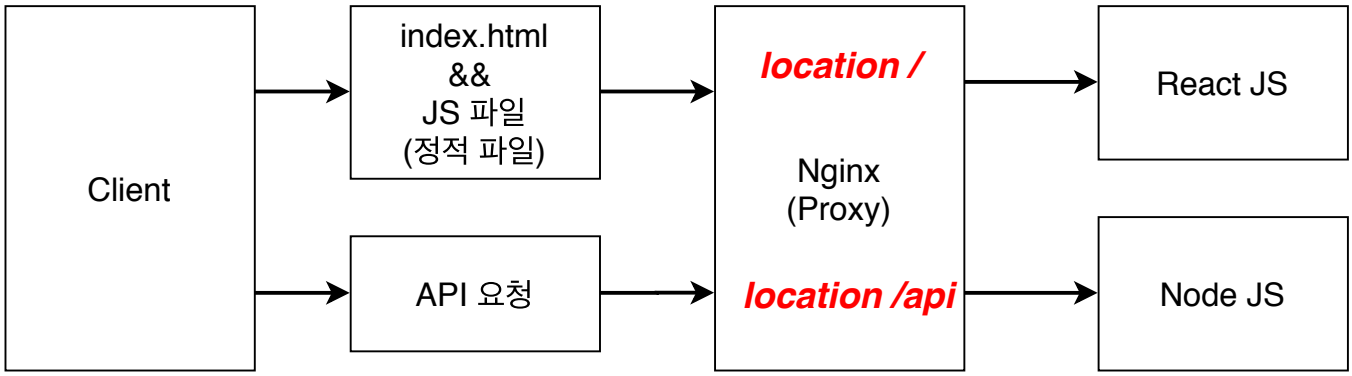

**2.한글이 깨지는 현상을 막기 위해 utf8로 인코딩할수 있게 설정을 해줌**

```
[mysqld]
character-set-server=utf8

[mysql]
default-character-set=utf8

[client]
default-character-set=utf8
```

**3. 위에 해준 설정을 실제 mysql 설정을 해주는 my.conf 파일로 덮어씌어준다.**

```
FROM mysql:5.7

ADD ./my.cnf /etc/mysql/conf.d/my.cnf
```

# Nginx를 위한 도커 파일 만들기




1. nginx 폴더와 default.conf 파일, Dockerfile 생성

2.default.conf 파일에
위에 프록시 기능 작성

```
upstream frontend {
  server frontend:3000;
}

upstream backend {
  server backend:5000;
}

server {
  listen 80;

  location / {
    proxy_pass http://frontend;
  }

  location /api {
    proxy_pass http://backend;
  }

  location /sockjs-node {
    proxy_pass http://frontend;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
  }
}
```

3.Nginx를 위한 Dockerfile 작성

```
FROM nginx
COPY ./default.conf /etc/nginx/conf.d/default.conf
```

# Docker Compose 파일 작성하기

1. 먼저 docker-compose.yml 파일을 생성하겠습니다.

2. 각각의 서비스들을 위한 틀을 만들어 주겠습니다.

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

  nginx:

  backend:

  mysql:
```

3. frontend 서비스를 위한 설정을 해주겠습니다.

```yaml
frontend:
  build:
    dockerfile: Dockerfile.dev
    context: ./frontend
  volumes:
    - /app/node_modules
    - ./frontend:/app
  stdin_open: true
```

4. nginx 서비스를 위한 설정을 해주겠습니다.

```yaml
nginx:
  restart: always
  build:
    dockerfile: Dockerfile.dev
    context: ./nginx
  ports:
    - "3000:80"
```

5. backend 서비스를 위한 설정을 해주겠습니다.

```yaml
backend:
  build:
    dockerfile: Dockerfile.dev
    context: ./backend
  container_name: app_backend
  volumes:
    - /app/node_modules
    - ./backend:/app
```

6. mysql 서비스를 위한 설정을 해주겠습니다.

```yaml
mysql:
  build: ./mysql
  restart: unless-stopped
  container_name: app_mysql
  ports:
    - "3306:3306"
  volumes:
    - ./mysql/mysql_data:/var/lib/mysql
    - ./mysql/sqls/:/docker-entrypoint-initdb.d/
  environment:
    MYSQL_ROOT_PASSWORD: johnahn
    MYSQL_DATABASE: myapp
```

7. 모든 설정이 끝났다면 이제 docker compose로 앱을 시작해보기

# Volume을 이용한 데이터 베이스 데이터 유지하기

```yaml
mysql:
  build: ./mysql
  restart: unless-stopped
  container_name: app_mysql
  ports:
    - "3306:3306"
  volumes:
    - ./mysql/mysql_data:/var/lib/mysql
    - ./mysql/sqls/:/docker-entrypoint-initdb.d/
  environment:
    MYSQL_ROOT_PASSWORD: johnahn
    MYSQL_DATABASE: myapp
```

## 볼륨을 이용한 데이터 영속성 구조