포팅 매뉴얼

프로젝트 기술 스택

협업 툴

- Notion
- Discord
- Mattermost
- Google Meet
- Figma
- Jira

OS

- Window10
- Ubuntu 20.04 LTS

IDE

-
- Gitlab

DB

- MySQL 8.0.29

Server

- AWS EC2
- Ubuntu 20.04 LTS
- Docker 23.0.0
- Jenkins 2.375.2

Front-end

- React 18.2.0
- Noe.js 18.13.0
- Recoil 0.7.6

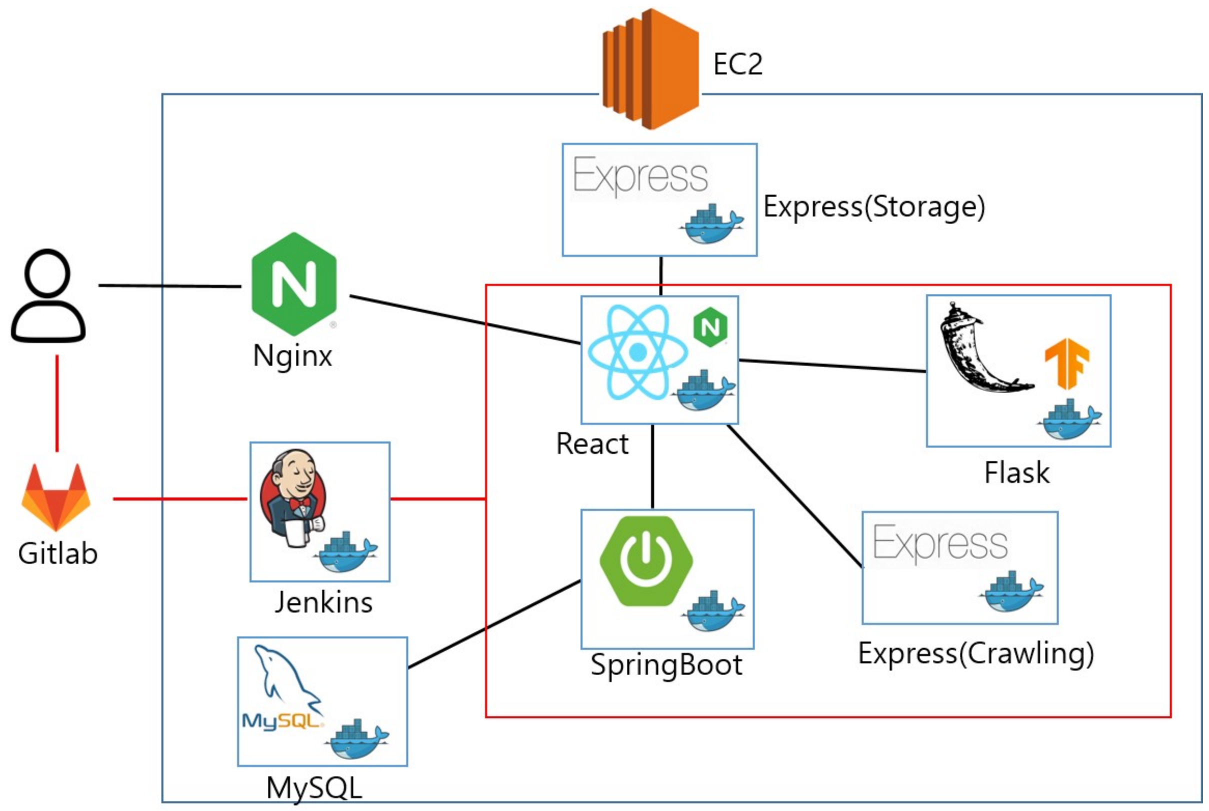
Back-end

- java 11.0.17
- Springboot 2.7.8
- Spring Data JPA
- Spring Security
- Spring Validation
- Express 4.18.2

편의툴

- Swagger 3.0.0
- Postman
- Talend API Testser

아키텍처



port

- FrontEnd: 80
- BackEnd: 8080
- MySQL: 3306:3306
- Jenkins:9090
- Crawling: 4040
- Storage: 4041

도커 초기 설정

1. 도커 설치

```
## docker
apt-get update

apt-get install sudo

apt-get install vim
```

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-pro
perties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add

sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable"

sudo apt-get update && sudo apt-get install docker-ce docker-ce-cli containerd.io

## docker-compose
sudo apt install jq
VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest |
 jq .name -r)
DESTINATION=/usr/bin/docker-compose

sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-com
pose-$(uname -s)-$(uname -m) -o $DESTINATION

sudo chmod 755 $DESTINATION

## docker network
sudo docker network create cd_network
```

2. https 설정

```
## certbot for ssl

sudo snap install --classic certbot

sudo certbot certonly --standalone

## nginx
sudo apt install nginx

cd /etc/nginx/conf.d

sudo vim default.conf
```

```
# default.conf
server {
    server_name j8a402.p.ssafy.io;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    location / {
        proxy_pass http://localhost:3000;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/j8a402.p.ssafy.io/fullchain.pem; # managed b
y Certbot
    ssl_certificate_key /etc/letsencrypt/live/j8a402.p.ssafy.io/privkey.pem; # managed
by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = j8a402.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen 80;
    listen [::]:80;
    server_name j8a402.p.ssafy.io;
    return 404; # managed by Certbot
}
```

- certbot nginx 적용 및 nginx 시작

```
sudo certbot --nginx

sudo nginx
```

3. Jenkins 설정

```
sudo docker pull jenkins/jenkins:lts-jdk11

docker run -u 0 -d -p 9090:8080 -p 50000:50000 -v /var/jenkins:/var/jenkins_home -v /v
ar/run/docker.sock:/var/run/docker.sock --name jenkins jenkins/jenkins:lts-jdk11
```

- 이후 `sudo docker logs jenkins` 명령어를 통해 jenkins 암호 확인
- http://(url):9090 포트를 통해 jenkins에 접속 후 필요한 플러그인 설치
- Gitlab 플러그인
  - Generic Webhook Trigger Plugin
  - GitLab
  - Gitlab API Plugin
  - GitLab Authentication plugin

- Docker 플러그인
  - Docker API Plugin
  - Docker Commons Plugin
  - Docker Compose Build Step Plugin
  - Docker Pipeline
  - Docker plugin
  - docker-build-step
- jenkins 컨테이너의 bash로 접근 후 jenkins 내부에서 docker에 접근할 수 있도록 설정

```
sudo docker exec –it jenkins bash

apt-get update
apt-get install \
        ca-certificates \
      curl \
        gnupg \
              sb-release
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/ke
yrings/docker.gpg
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] http
s://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

- docker-compose를 사용하기 위해 추가 설정

```
sudo apt install jq
VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest |
 jq .name -r)
DESTINATION=/usr/bin/docker-compose
sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-com
pose-$(uname -s)-$(uname -m) -o $DESTINATION
sudo chmod 755 $DESTINATION
```

- 이후 jenkins webhook을 이용하여 gitlab과 연동

```
# jenkins pipeline script
pipeline {
    agent any
    environment {
        SKIP_BUILD = "${env.gitlabBranch != 'dev'}"
        HEALTH_STATE = "green"
    }

    stages {
        stage("Print Event Info") {
            when {
                environment name: 'SKIP_BUILD', value: 'false'
            }
            steps {
                echo "Branch : ${env.gitlabBranch}"
                echo "URL : ${env.gitlabSourceRepoURL}"
                echo "User : ${env.gitlabUserName}"
                echo "Action : ${env.gitlabActionType}"
            }
        }
        stage('Abort Prvious Build') {
            when {
                environment name: 'SKIP_BUILD', value: 'false'
            }
            steps {
                echo "abort previous build"
                abortPreviousRunningBuilds()
            }
        }
        stage('Clone') {
            when {
                expression { SKIP_BUILD == "false" }
            }
            steps {
                git branch: "${env.gitlabBranch}", url: "https://lab.ssafy.com/s08-ai-
image-sub2/S08P22A402", credentialsId: 'GitlabAccount'
            }
        }
        stage('HelathCheck') {
            when {
                expression { SKIP_BUILD == "false" }
            }
            steps {
                echo SKIP_BUILD
                echo HEALTH_STATE
                script {
                    try{
                        def response = sh(returnStdout: true, script: "curl -s http
s://j8a402.p.ssafy.io:4040")
                        def isBlue = response.contains("blue")

                        if (isBlue) {
                            HEALTH_STATE = "blue"
                        }
                    } catch (e) {
                        echo "Server downed"
                    }
                }
            }
        }

        stage('Blue build') {
            when {
                expression { SKIP_BUILD == "false" && HEALTH_STATE == "green" }
            }
            steps {
                echo SKIP_BUILD
                echo HEALTH_STATE
                echo "docker compose blue build"
                sh "sudo docker-compose -f docker-compose-blue.yml build --no-cache"
            }
            post {
                success {
                    echo "Success to build"
                }
                failure {
                    echo "Docker build blue failed. Remove all test images"
                    sh "sudo docker-compose -f docker-compose-blue.yml down --rmi all"
                    error "pipeline aborted"
                }
            }
        }
        stage('Green build') {
            when {
                expression { SKIP_BUILD == "false" && HEALTH_STATE == "blue" }
            }
            steps {
                echo SKIP_BUILD
                echo HEALTH_STATE
                echo "docker compose green build"
                sh "sudo docker-compose -f docker-compose-green.yml build --no-cache"
            }
            post {
                success {
                    echo "Success to build"
                }
                failure {
                    echo "Docker build green failed. Remove all test images"
                    sh "sudo docker-compose -f docker-compose-green.yml down --rmi al
l"
                    error "pipeline aborted"
                }
            }
        }

        stage('Blue down') {
            when {
                expression { SKIP_BUILD == "false" && HEALTH_STATE == "blue" }
            }
            steps {
                echo "docker compose down"
                sh "sudo docker-compose -f ../fast/docker-compose-blue.yml down --rmi
 all"
                sh "sudo docker-compose -f ../fast@2/docker-compose-blue.yml down --rm
i all"
            }
        }
        stage('Green down') {
            when {
                expression { SKIP_BUILD == "false" && HEALTH_STATE == "green" }
            }
            steps {
                echo "docker compose down"
                sh "sudo docker-compose -f ../fast/docker-compose-green.yml down --rmi
all"
                sh "sudo docker-compose -f ../fast@2/docker-compose-green.yml down --r
mi all"
            }
        }
        stage('Blue up') {
            when {
                expression { SKIP_BUILD == "false" && HEALTH_STATE == "green" }
            }
            steps {
                echo "docker compose up"
                sh "sudo docker-compose -f docker-compose-blue.yml up -d"
            }
            post {
                success {
                    echo "Success to run docker-compose"
                }
                failure {
                    echo "Docker run failed. remove all test images"
                    sh "sudo docker-compose -f docker-compose-blue.yml down --rmi all"
                    error "pipeline aborted"
                }
            }
        }
        stage('Green up') {
            when {
                expression { SKIP_BUILD == "false" && HEALTH_STATE == "blue" }
            }
            steps {
                echo "docker compose up"
                sh "sudo docker-compose -f docker-compose-green.yml up -d"
            }
            post {
                success {
                    echo "Success to run docker-compose"
                }
                failure {
                    echo "Docker run failed. remove all test images"
                    sh "sudo docker-compose -f docker-compose-green.yml down --rmi al
l"
                    error "pipeline aborted"
                }
            }
        }

    }
}

def abortPreviousRunningBuilds() {
    def buildNumber = env.BUILD_NUMBER as int
    if (buildNumber > 1) milestone(buildNumber - 1)
    milestone(buildNumber)
}
```

4. MySQL 설정

- mysql 컨테이너 실행

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=<password> -d -p 3306:3306 mysql:latest
```

- mysql 컨테이너 bash에 접속

```
docker exec -it mysql bash
```

- 한글 인코딩 설정

```
vi /etc/mysql/my.cnf
```

```
# /etc/mysql/my.cnf
[client]
default-character-set=utf8

[mysql]
default-character-set=utf8

[mysqld]
collation-server = utf8_unicode_ci
init-connect='SET NAMES utf8'
character-set-server = utf8
```

- 초기 데이터 생성

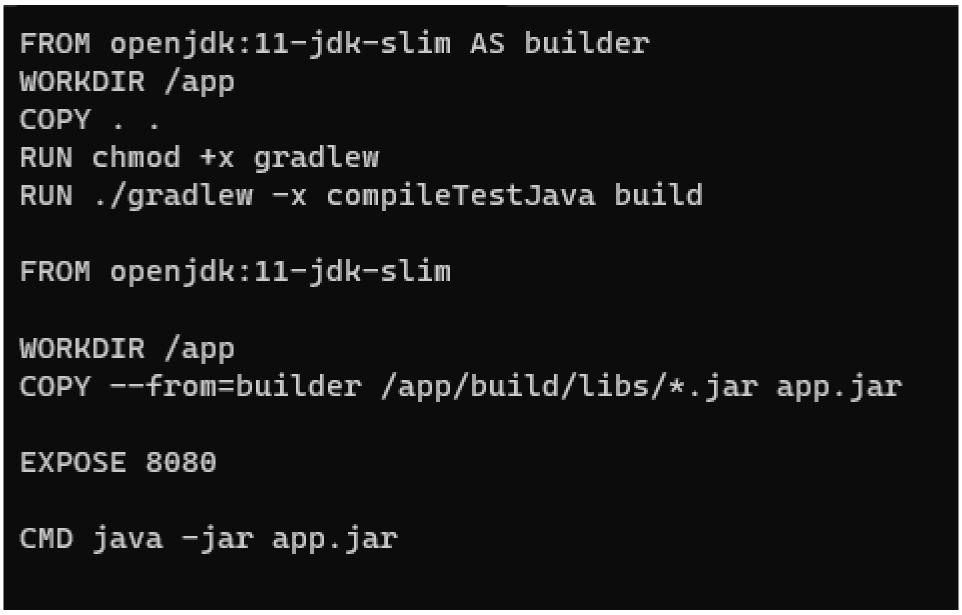
```
create database fast

CREATE USER 'a402'@'%' identified by <password>;
GRANT ALL PRIVILEGES ON . TO 'a402'@'%';

INSERT INTO fast.authority (authority_name) VALUES ('ROLE_USER');
INSERT INTO fast.authority (authority_name) VALUES ('ROLE_ADMIN');

INSERT INTO fast.tag (name) VALUES ('바다');
INSERT INTO fast.tag (name) VALUES ('산');
INSERT INTO fast.tag (name) VALUES ('액티비티');
INSERT INTO fast.tag (name) VALUES ('강');
INSERT INTO fast.tag (name) VALUES ('유적지');
INSERT INTO fast.tag (name) VALUES ('계곡');
INSERT INTO fast.tag (name) VALUES ('호캉스');
INSERT INTO fast.tag (name) VALUES ('캠핑');
INSERT INTO fast.tag (name) VALUES ('힐링');
INSERT INTO fast.tag (name) VALUES ('배낭여행');
INSERT INTO fast.tag (name) VALUES ('박물관');
INSERT INTO fast.tag (name) VALUES ('자전거');
INSERT INTO fast.tag (name) VALUES ('등산');
INSERT INTO fast.tag (name) VALUES ('휴양지');
INSERT INTO fast.tag (name) VALUES ('도심');
```

빌드 상세 설정

- 프로젝트에서 사용할 도커 컨테이너는 총 7개
- Jenkins, mysql, frontend, backend, crawling server, storage server, ai
- 이 중 frontend, backend, crawling server는 jenkins webhook을 이용해 gitlab의 push 이벤트를 감지하여 프로젝트 clone 후 build
- storage server는 이미지를 저장하기 위한 서버이므로 clone 후 새로 build하는 과정에서 제외
- build에 필요한 Dockerfile은 각 컨테이너의 설정에 맞게 작성한 후 각각의 폴더에 위치
- 이후 docker-compose를 이용해 한번에 실행
- blue-green 무중단 배포를 위해 storage를 제외한 모든 dockerfile과 docker-compose는 각각 2개씩 생성하여 -blue, -green으로 구분
- 현재 blue 컨테이너들이 실행 중이면 green을 빌드 한 뒤에 blue 컨테이너를 내리고 green 컨테이너를 실행시킨다

Back-end

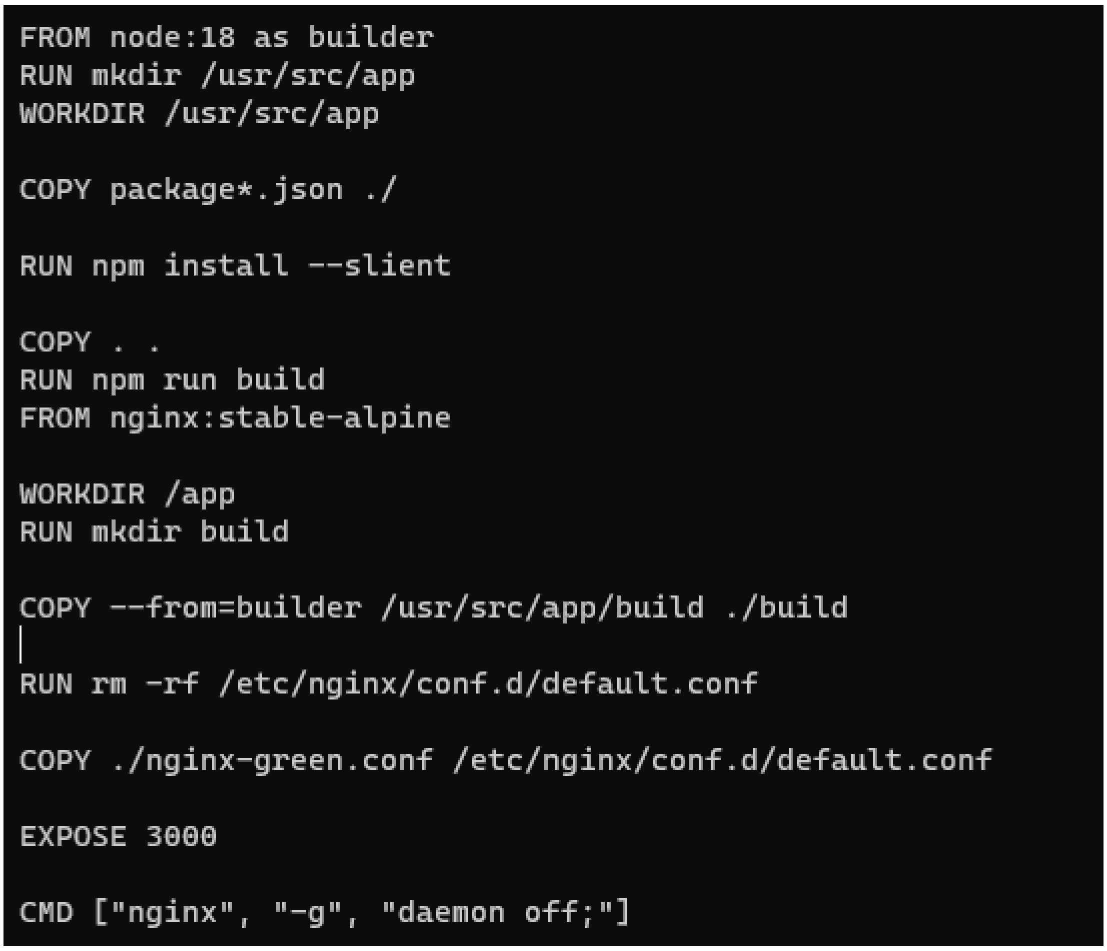
./backend/Dockerfile



```
FROM openjdk:11-jdk-slim AS builder
WORKDIR /app
COPY . .
RUN chmod +x gradlew
RUN ./gradlew -x compileTestJava build

FROM openjdk:11-jdk-slim

WORKDIR /app
COPY --from=builder /app/build/libs/*.jar app.jar

EXPOSE 8080

CMD java -jar app.jar
```

Front-end

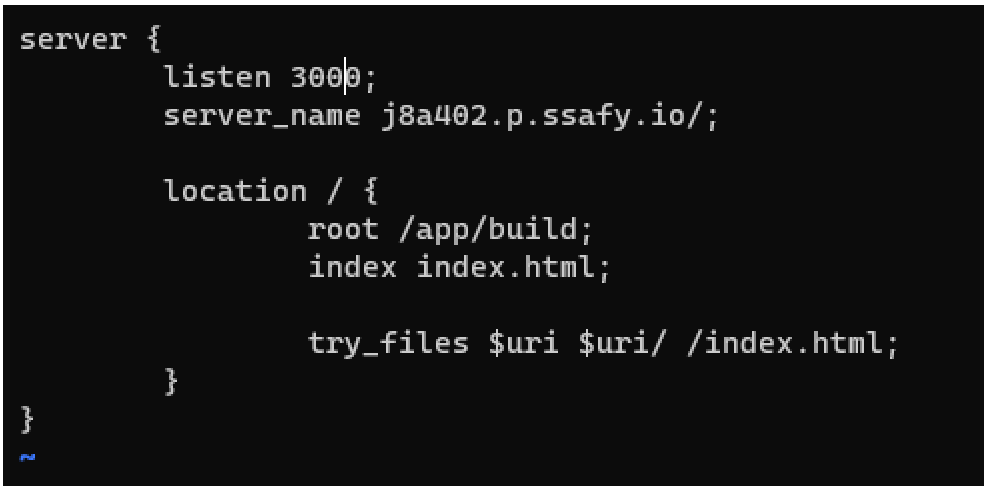
./frontend/Dockerfile



```
FROM node:18 as builder
RUN mkdir /usr/src/app
WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install --slient

COPY . .
RUN npm run build
FROM nginx:stable-alpine

WORKDIR /app
RUN mkdir build

COPY --from=builder /usr/src/app/build ./build

RUN rm -rf /etc/nginx/conf.d/default.conf

COPY ./nginx-green.conf /etc/nginx/conf.d/default.conf

EXPOSE 3000

CMD ["nginx", "-g", "daemon off;"]
```

- frontend의 경우 외부의 80포트를 내부 3000포트로 포워딩하기 위해 frontend 컨테이너 내부에 gninx 설치 후 설정 파일 COPY

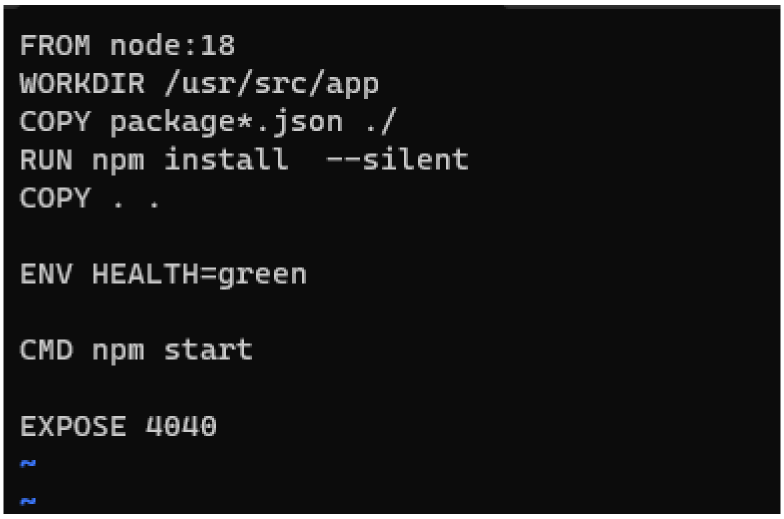
./frontend/nginx.conf



```
server {
        listen 3000;
        server_name j8a402.p.ssafy.io/;

        location / {
                root /app/build;
                index index.html;

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

Crawling

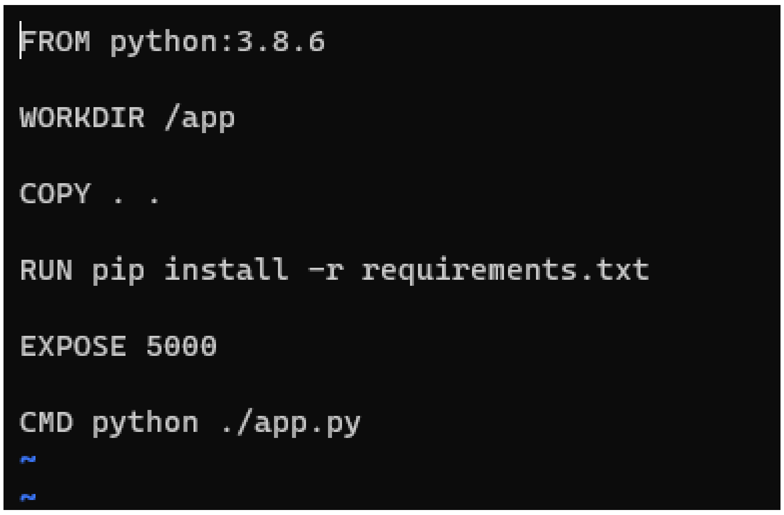
./crawling/Dockerfile



```
FROM node:18
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install  --silent
COPY . .

ENV HEALTH=green

CMD npm start

EXPOSE 4040
~
~
```

AI

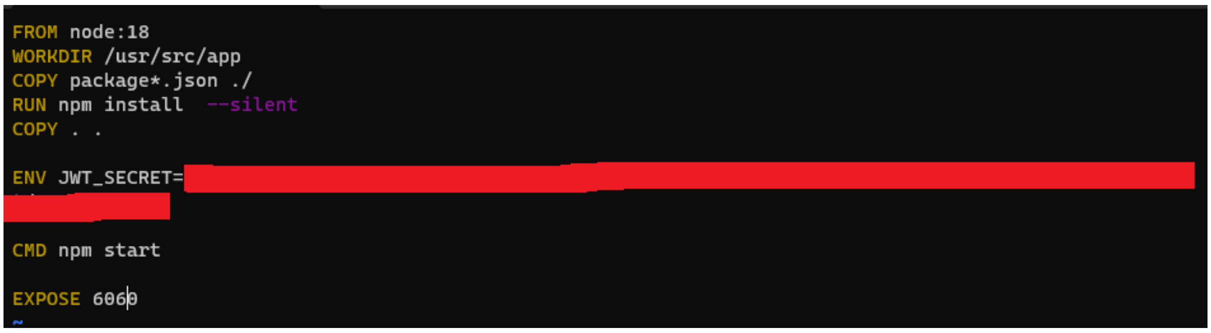
./ai/Dockerfile



```
FROM python:3.8.6

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

EXPOSE 5000

CMD python ./app.py
~
~
```

Storage

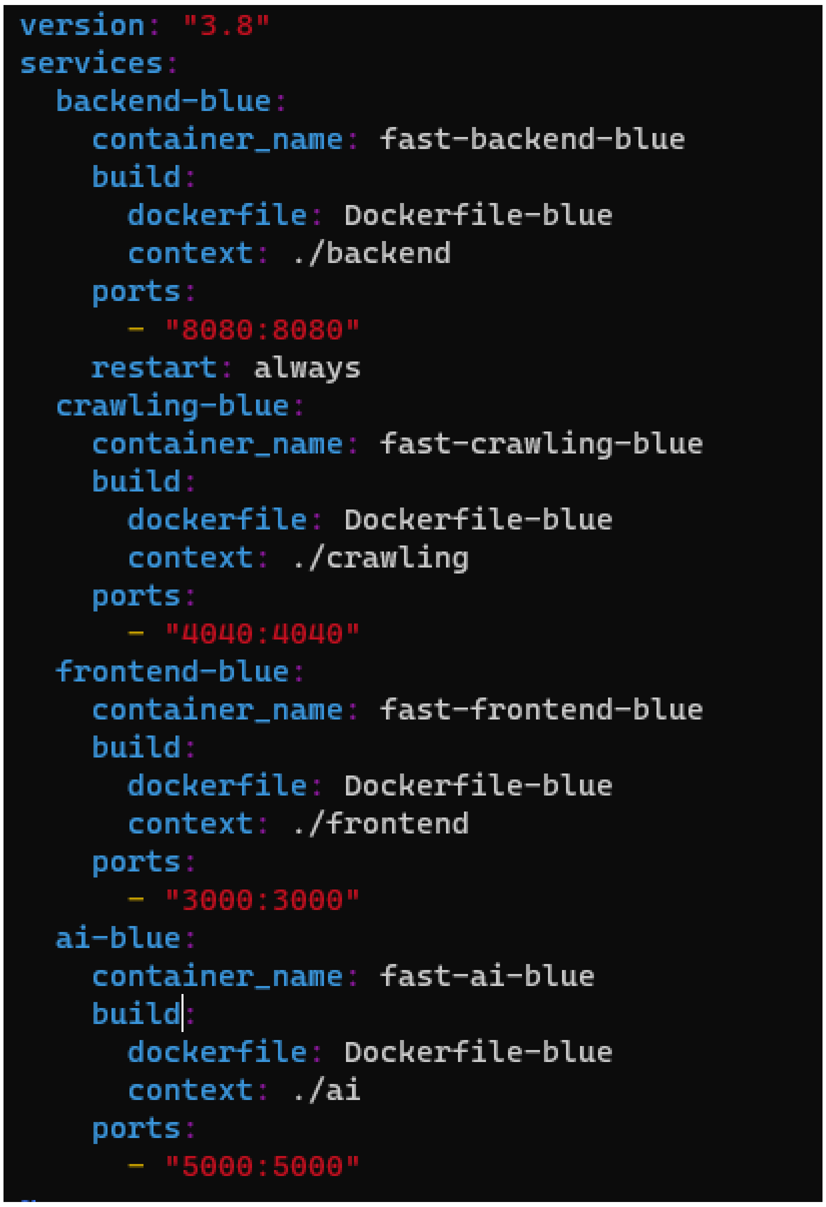
./storage/Dockerfile-blue



```
FROM node:18
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install  --silent
COPY . .

ENV JWT_SECRET=[redacted]

CMD npm start

EXPOSE 6060
~
```

Docker-compose

./docker-compose.yml



```
version: "3.8"
services:
  backend-blue:
    container_name: fast-backend-blue
    build:
      dockerfile: Dockerfile-blue
      context: ./backend
    ports:
      - "8080:8080"
    restart: always
  crawling-blue:
    container_name: fast-crawling-blue
    build:
      dockerfile: Dockerfile-blue
      context: ./crawling
    ports:
      - "4040:4040"
  frontend-blue:
    container_name: fast-frontend-blue
    build:
      dockerfile: Dockerfile-blue
      context: ./frontend
    ports:
      - "3000:3000"
  ai-blue:
    container_name: fast-ai-blue
    build:
      dockerfile: Dockerfile-blue
      context: ./ai
    ports:
      - "5000:5000"
```

외부 서비스

- 카카오 로그인 API
  - https://developers.kakao.com/docs/latest/ko/kakaologin/common
- 네이버 로그인 API
  - https://developers.naver.com/docs/login/api/api.md
- 카카오 맵 API
  - https://apis.map.kakao.com
  - 방문한 여행지를 마커로 지도에 표시하기 위해 사용