포팅 메뉴얼

사용 도구

- 이슈 관리: `Jira`
- 형상 관리: `GitLab`
- 커뮤니케이션: `Notion` `MatterMost`
- 디자인: `Figma`
- CI/CD: `Jenkins`
- 모니터링: `Prometheus` `Grafana`

개발 도구

- `Visual Studio Code` : 1.76.0
- `Visual Studio Code` : 1.86.2 (Universal)
- `Intellij` : 2022.3.2 (Ultimate Edition)
- `jupyter lab` : 4.1.2

개발 환경

FE

- `React` : 18.2.0
- `Node` : v20.11.1
- `TypeScript` : 5.4.2
- `Styled-components` : 6.1.8
- `Zustand` : 4.5.2

BE

Java

- `JDK` : Oracle OpenJDK version 17.0.10
- `Spring Boot` : 3.2.3

Python

- `Python` : 3.8.19
- `FastAPI` : 0.74.1

Database

- `Redis` : 7.2.4
- `MariaDB` : 10.11.6
- `MongoDB` : 6.0
- `InfluxDB` : 2.7.5

AI

- `Python` : 3.8.19
- `Pytorch` : 2.2.1

Service

- `Docker` : 25.0.4
- `Jenkins` : 2.440.1
- `Promethus` : 2.51
- `Grafana` : 10.4.0
- `Nginx` : 1.18.0 (Ubuntu), 1.25.4
- `Sonarqube` : 9.9.4

배포 환경

- `AWS EC2` CPU : Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz, RAM : 16GB, OS: Ubuntu 20.04 LTS
- `AWS S3`
- `Firebase`

환경 변수

FE

directory 구조

.env

```
# 백엔드 주소
REACT_APP_BASE_URL = { Spring Server }

# 카카오 키
REACT_APP_API_KEY = { Kakao API KEY }
REACT_APP_REDIRECT_URI = { Kakao Redirect URI }

#
# 파이어베이스 키
REACT_APP_API_FIRE_KEY = { Firebase KEY }
REACT_APP_AUTH_DOMAIN = { Firebase Project URL }
REACT_APP_PROJECT_ID = { Firebase Project ID }
REACT_APP_STORAGE_BUCKET = { Firebase Bucket URL }
REACT_APP_MESSAGING_SENDER_ID = { Firebase Message Sencer ID }
REACT_APP_APP_ID = { Firebase Service ID }
REACT_APP_MEASUREMENT_ID = { Firebase Measurement ID }
```

BE (Java)

directory 구조

application.yml

```
spring:
  encoding: UTF-8
  config:
    import:
      - optional:env/env.yml
      - optional:env/firebase_service_key.json
  server:
    port: 8080
    servlet:
      context-path: /
      encoding:
        charset: UTF-8
        force: true

  servlet:
    multipart:
      max-file-size: 50MB
      max-request-size: 50MB
  jpa:
    hibernate:
          ddl-auto: none
      naming:
        physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
    show-sql: true
    format_sql: true
    use_sql_comments: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQLDialect
  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: ${ KAKAO_CLIENT_ID }
            redirect-uri: ${ KAKAO_REDIRECT_URL }
            client-authentication-method: POST
            client-secret: ${ KAKAO_CLIENT_SECRET }
            authorization-grant-type: authorization_code
```

```yaml
            scope: account_email
            client_name: kakao
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id
  datasource:
    url: ${ MARIADB_URL }
    username: ${ MARIADB_USERNAME }
    password: ${ MARIADB_PASSWORD }
    driver-class-name: org.mariadb.jdbc.Driver
  data:
    mongodb:
      host: ${ MONGODB_HOST }
      port: ${ MONGODB_PORT }
      database: OUI
      username: ${ MONGODB_USERNAME }
      password: ${ MONGODB_PASSWORD }
      authentication-database: admin
    redis:
      host: ${ REDIS_HOST }
      port: ${ REDIS_PORT }
jwt:
  header: Authorization
  secret: ${ JWT_KEY }

spotify:
  client-id: ${ SPOTIFY_CLIENT_ID }
  client-secret: ${ SPOTIFY_CLIENT_SECRET }

youtube:
  key: ${ YOUTUBE_API_KEY }

cloud:
  aws:
    s3:
      bucket: ${ S3_BUCKET }
    credentials:
      access-key: ${ S3_ACCESS }
      secret-key: ${ S3_SECRET }
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false
server:
  servlet:
    encoding:
      charset: UTF-8
      enabled: true
      force: true
  port: 8080
  ssl:
    enabled: true
    enabled-protocols:
      - TLSv1.1
      - TLSv1.2
    key-store: "classpath:env/keystore.p12"
    key-store-password: ${ SSL_KEY }
    key-store-type: "PKCS12"

openai:
  model: gpt-3.5-turbo
  api:
    key: ${ OPEN_API_KEY }
    url: ${ OPEN_API_URI }
```

env.yml

```
REDIS_HOST: localhost
REDIS_PORT: { #port }

JWT_KEY: {JWT KEY}

KAKAO_CLIENT_ID: { 소셜로그인 Client ID }
KAKAO_CLIENT_SECRET: { 소셜로그인 Client Secret }
KAKAO_REDIRECT_URL: { 소셜로그인 redirect URL }

DB_NAME: oui
MARIADB_USERNAME: { 계정명 }
MARIADB_PASSWORD: { 계정 비밀번호 }
MARIADB_URL: jdbc:mariadb://{ host }:{ #port }/${ DB_NAME }?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8&al
DB_ROOT_PASSWORD: {비밀번호}

# MONGODB
MONGODB_HOST: { host }
MONGODB_PORT: { #port }
MONGODB_USERNAME: { 계정명 }
MONGODB_PASSWORD: { 계정 비밀번호 }

# Spotify
SPOTIFY_CLIENT_ID: {스포티파이 Client ID}
SPOTIFY_CLIENT_SECRET: {스포티파이 Client Secret}

# S3
S3_BUCKET: emotionoui
S3_ACCESS: { access key }
S3_SECRET: { secret key }

# YOUTUBE API KEY
YOUTUBE_API_KEY: { api key }

# ChatGPT
OPEN_API_KEY: { api key }
OPEN_API_URI: { url }

# SSL KEYSTORE KEY
SSL_KEY: {ssl keystore key}
```

firebase_service_key.json

사이트에서 다운로드 받아서 사용

```
{
  "type": { type },
  "project_id": { project id },
  "private_key_id": { private key id },
  "private_key": { private key },
  "client_email": { client email },
  "client_id": { client id },
  "auth_uri": { auth uri },
  "token_uri": { token uri },
  "auth_provider_x509_cert_url": { auth cert url },
  "client_x509_cert_url": { client cert url },
  "universe_domain": { universe domain }
}
```

BE (Python)

directory 구조

application-secrets.json

```
{
    "MONGODB_URL": mongodb 주소,
    "MONGODB_DBNAME": db명,
}
```

EC2 인스턴스 초기 설정

swap 설정

- 디스크 용량 확인 및 스왑 영역 설정
  - `df -h` 확인 → 디스크 용량이 많아서 16G 할당

```
fallocate -l 16G /swapfile
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
# fstab 파일에 시작할 때 마운트할 공간 저장
```

계정 접근

- id, pw로 접속 허용

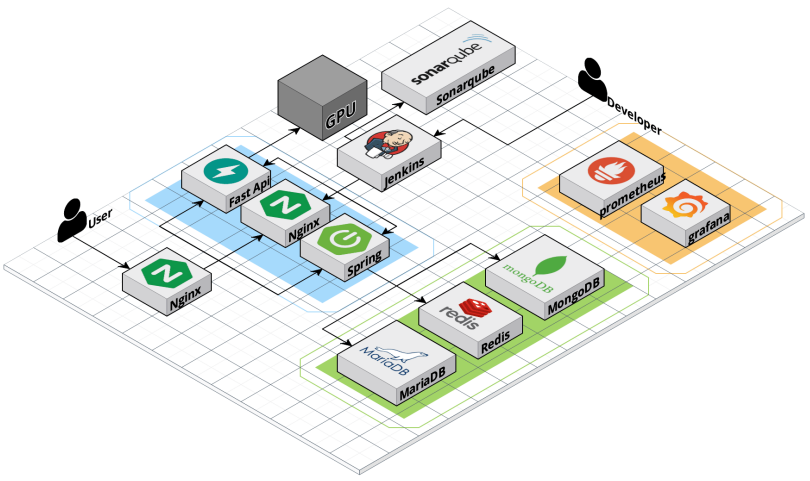
```
sudo passwd root
su root

cd /etc
chmod 660 sudoers
vi sudoers
chmod 440 sudoers

adduser [New Id]
passwd [New Id Pw]
su [New Id]

cd /etc/ssh
sudo vi sshd_config    # PasswordAuthentication-> yes
sudo service sshd restart
```

Architecture 설계



사용 포트 정보

```
22: ssh
80: nginx - proxy
443: SSL

3306: mariaDB
3333: sonarqube

6379: redis

8000: nginx -react
8008: fastapi (ai)
```

```
8323: docker-exporter
8888: jenkins

8900: grafana
8901: prometheus
8902: node-expoter

8989: gerrit(stop)

27017: mongodb
```

```
8080: spring
8086: influxdb
```

포트 상세

[Domain]

https://j10a506.p.ssafy.io/
(172.26.14.196)

[LoadBalancing]

Nginx 80

http://j10a506.p.ssafy.io

[CI/CD]

Jenkins 8888:8080

http://j10a506.p.ssafy.io:8888

엔진엑스 8000:80

http://j10a506.p.ssafy.io:8000

Spring 8080:8080

https://j10a506.p.ssafy.io:8080

[DB]

MaraiDB 3306:3306

http://j10a506.p.ssafy.io:3306

mongoDB 27017:27017

http://j10a506.p.ssafy.io:27017

redis 6379:6379

[Monitoring]

Grafana 8900:3000

http://j10a506.p.ssafy.io:8900

Prometheus 8901:9090

http://j10a506.p.ssafy.io:8901

influxDB 8086:8086

http://j10a506.p.ssafy.io:8086/

Node-exporter8902:9100

http://j10a506.p.ssafy.io:8902

[QA]

sonarqube 3333:9000

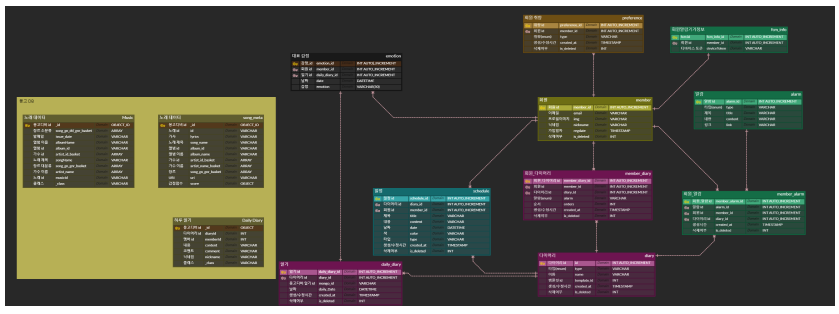
http://j10a506.p.ssafy.io:3333

---

redis 6379:6379

togeball

ERD



Docker 설치

도커 설치

```
sudo apt-get update

sudo apt-get install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-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 -y docker-ce docker-ce-cli containerd.io \
    docker-compose docker-compose-plugin
젠킨스 컨테이너 생성
```

- jenkins/jenkins:lts 컨테이너 생성
  - JDK 17로 작업하기 위해 JAVA 설치 및 JAVA_HOME 환경 변수 생성
  - 컨테이너 데이터 유지를 위한 마운트 & DooD 방식을 위한 소켓 마운트
  - jenkins 유저가 default이기 때문에 root 유저로 생성

```
mkdir -p /var/jenkins_home

chown -R 1000:1000 /var/jenkins_home/

docker run --restart=on-failure --user='root' \
    -p 8888:8080 -p 50000:50000 \
    --env JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64 \
    -v /var/jenkins_home:/var/jenkins_home \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -d --name jenkins jenkins/jenkins:lts \
```

- 젠킨스 환경 구축
  - 로컬과 마찬가지로 설정(컨테이너 OS가 데비안인 것 주의)

```
apt-get update
apt-get install openjdk-17-jdk -y

apt-get install -y \
apt-transport-https \
ca-certificates \
curl \
gnupg2 \
software-properties-common

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

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

apt-get update
apt-get install docker-ce docker-ce-cli containerd.io
```

리액트 프로젝트를 빌드할 것이므로 node설치 (최신 버전은 apt로 설치 불가)

```
curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key\
 | gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg
export NODE_MAJOR=20
sudo tee /etc/apt/sources.list.d/nodesource.list
sudo apt update && sudo apt install nodejs -y
```

Nginx 구성

- 역할
  - **로드밸런싱, 웹서버**

    로드밸런싱 역할을 하는 서버는 로컬, 웹서버 역할을 하는 서버는 도커로 구분하여 설치

웹서버

- 도커 컨테이너를 이용해 nginx 생성 (debian ver)
- 기본 root 디렉토리가 /usr/share/nginx로 되어 있음.

이 위치에 프론트 프로젝트의 빌드된 build 폴더를 옮겨줄 것이므로 /usr/share/nginx/build로 기본 경로를 바꿔준다.

```
docker run --restart=on-failure -p 8000:80 -d --name nginx nginx
```

- /etc/nginx/conf.d

```
server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    location / {
        root   /usr/share/nginx/html/build/;
        index  index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
```

로드밸런싱

- **nginx 설치**

```
apt install nginx
```

- **SSL 인증 받기**
  - Certbot 설치

```


sudo apt-get -y install python3-certbot-nginx
apt install letsencrypt

sudo snap install --classic certbot

// sudo ln -s /snap/bin/certbot /usr/bin/certbot
// sudo apt-add-repository -r ppa:certbot/certbot
```

  - SSL 인증서 받기

```
sudo certbot --nginx
# 이메일 입력 > N > Domain 작성
```

- 리버스 프록시 설정
  - /etc/nginx/conf.d/service-url.inc

```
 set $service_url http://127.0.0.1:8000;
```

- /etc/nginx/sites-enabled/default
  - CORS policy를 위한 8080 ssl proxy 사용

```
server {
    if ($host = j10a506.p.ssafy.io) {
        return 308 https://$host$request_uri;
    }

    listen 80 ;
    listen [::]:80 ;
    server_name j10a506.p.ssafy.io;
    return 404;

}


server {
```

```
    server_name j10a506.p.ssafy.io;

        include /etc/nginx/conf.d/service-url.inc;
        include /etc/nginx/conf.d/port.inc;

        location / {
                proxy_pass $service_url;
                proxy_set_header Host $host;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-Proto $scheme;
        }


#       location /reload {
#                 allow 172.0.0.1;
#                 deny all;
#
#               post_action /restart_nginx.sh;
#       }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/j10a506.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/j10a506.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

}
```

Spring SSL

- PKCS12 생성
  - /etc/letsencrypt/live/<도메인> 아래에 있는 fullchain.pem과 privkey.perm을 묶어서 스프링 프로젝트에 적용할 pkcs12 형식의 파일을 만든다.

```
sudo openssl pkcs12 -export -in fullchain.pem \
            -inkey privkey.pem -out keystore.p12 \
            -name ttp -CAfile chain.pem -caname root
# 패스워드 생성
```

- SSL/TLS 인증서 설정
  - resources/ssl 경로에 keystore.p12 파일 생성
  - application.yml

```
server:
  servlet:
    encoding:
      charset: UTF-8
      enabled: true
      force: true
port: 8080
ssl:
  enabled: true
  enabled-protocols:
    - TLSv1.1
    - TLSv1.2
  key-store: "classpath:env/keystore.p12"
  key-store-password: [키 만들 때 입력한 패스워드]
  key-store-type: "PKCS12"
```

DB 생성

MariaDB

- 도커 볼륨 생성

```
docker volume create mariadb -volume
```

- 도커 볼륨 조회

```
docker volume ls
```

- DB 실행

```
docker run -d --restart=on-failure \
    -p 3306:3306 --name mariadb \
    --env MARIADB_ROOT_PASSWORD=[Password] \
    -v maraidb:/var/libs/mariadb mariadb:lts
docker exec -it mariadb mariadb -u root -p
```

MongoDB

- AI 학습 데이터 저장. 작성한 일기의 내용 저장
- docker-compose

```
mongodb:
    image: "mongo"
    environment:
      MONGO_INITDB_ROOT_USERNAME: [User]
      MONGO_INITDB_ROOT_PASSWORD: [User Password]
      MONGO_INITDB_DATABASE: [Init DB]
    ports:
      - "27017:27017"
    volumes:
      - "mongodb_data:/data/db"
    restart: on-failure
```

Redis

- AccessToken, RefreshToken 저장
- 볼륨 생성

```
docker volume create redisdb
```

- redis config 생성 (/etc/redis/redis.conf)

```
bind 0.0.0.0

port 6379

requirepass [사용하고자 하는 비밀번호]

maxmemory 1g

maxmemory-policy volatile-ttl


save 900 1
save 300 10
save 60 10000
```

- 실행

```
docker run -d \
--restart=on-failure \
--name=redis \
-p 6379:6379 \
-e TZ=Asia/Seoul \
-v /etc/redis/redis.conf:/etc/redis/redis.conf \
-v redisdb:/data \
redis:latest --requirepass [Password]
```

InfluxDB

- 그라파나 젠킨스 연동을 위해 influxdb 설치

```
docker volume create influxdb_data
```

- 도커 컨테이너 실행

```
docker run -d -p 8086:8086 \
--restart=on-failure \
-v /var/lib/influxdb:/var/lib/influxdb2 \
-e DOCKER_INFLUXDB_INIT_USERNAME=[User] \
-e DOCKER_INFLUXDB_INIT_PASSWORD=[Password] \
-e DOCKER_INFLUXDB_INIT_ORG=[Organization] \
-e DOCKER_INFLUXDB_INIT_BUCKET=[Bucket] \
-e DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=[Token] \
--name influxdb influxdb
```

CI/CD 구축

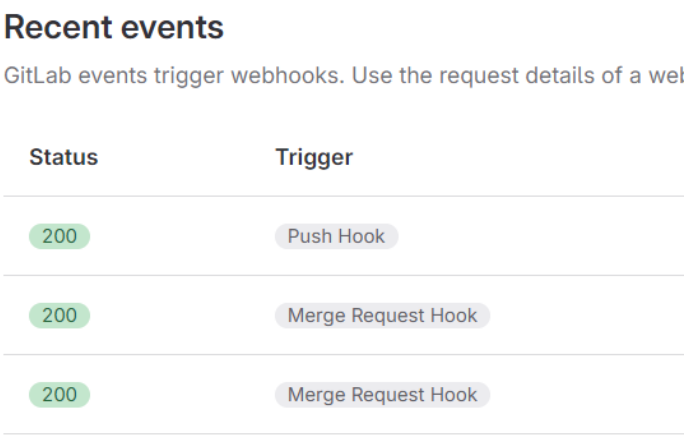
파이프 라인 생성

Enabled GitLab triggers에서 **Accepted Merge Request Events** 선택

[GitLab] Settings> Webhooks > Add WebHooks

URL에는 젠킨스 BuildTrigger에서 보여지는 깃랩 주소를 작성

토큰에는 젠킨스에서 생성한 토큰을 넣어주고 웹훅 완성



- Pipeline

파이프 라인 스크립트 작성

깃 브랜치 전략을 변형한 깃 전략 사용으로 dev 브랜치 특정하여 실행

⇒ GitLab API 사용

```
# BRANCH
curl --header "PRIVATE-TOKEN: `PRIVATE-TOKEN`" \
    "https://lab.ssafy.com/api/v4/projects \
    /${ project ID }/merge_requests?state=opened" \
    | jq '.[0] | .source_branch'

# ASSIGNEE
curl --header "PRIVATE-TOKEN: `PRIVATE-TOKEN`" \
    "https://lab.ssafy.com/api/v4/projects \
    /${ project ID }/merge_requests?state=opened" \
    | jq '.[0] | .assignees[0] | .name'

# REVIWER
curl --header "PRIVATE-TOKEN: `PRIVATE-TOKEN`" \
    "https://lab.ssafy.com/api/v4/projects \
    /${ project ID }/merge_requests?state=opened" \
    | jq '.[0] | .reviewers[0] | .name'
```

Backend CI/CD

- 이미지 빌드 후 사용

```
FROM gradle:8.6.0-jdk17 as builder
WORKDIR /build

# 그래들 파일이 변경되었을 때만 새롭게 의존패키지 다운로드 받게함.
COPY build.gradle settings.gradle /build/
```

```
RUN gradle build -x test --parallel --continue > /dev/null 2>&1 || true

# 빌더 이미지에서 애플리케이션 빌드
COPY . /build
RUN gradle build -x test --parallel

# APP
FROM openjdk:17-ea-4-jdk-slim
WORKDIR /app

# 빌더 이미지에서 jar 파일만 복사
COPY --from=builder /build/build/libs/oui.backend-0.0.1-SNAPSHOT.jar .

EXPOSE 8080
ENV TZ Asia/Seoul
# root 대신 nobody 권한으로 실행
USER nobody
ENTRYPOINT [                                                  \
   "java",                                                    \
   "-jar",                                                    \
   "-Djava.security.egd=file:/dev/./urandom",                 \
   "-Dsun.net.inetaddr.ttl=0",                                \
   "oui.backend-0.0.1-SNAPSHOT.jar"                                \
]
```

- 젠킨스 파이프라인 작성

```
pipeline{
    agent any
    environment {
        def BRANCH = sh(script: '''curl --fail --header "PRIVATE-TOKEN: ${ private Token }" "https://lab.ssafy.com/api/v
    }

    stages {
        stage('gitlab Connect'){
            steps{
                git branch: ${ gitlab branch },
                credentialsId: ${ credential ID },
                url: ${ gitLab URL }
            }
        }
        stage('build'){
            steps{
                sh 'cd /var/jenkins_home/workspace/backend-dev'
                dir('backend'){
                    sh 'cp -r /var/jenkins_home/back/env /var/jenkins_home/workspace/backend-dev/backend/src/main/resour
                    sh 'chmod +x gradlew'
                    sh './gradlew clean sonar -Dsonar.projectKey= ${ project Key } -Dsonar.host.url= ${ sonarqube URL }
                    sh './gradlew build -x test'
                }
            }
        }
        stage('deploy'){
            steps{
                sh 'docker stop ${container name} && docker rm ${container name} && docker rmi ${ container img }'
                dir('backend'){
                    // sh 'sh /blue-green/deploy.sh'
                    sh 'docker build -t backend ./'
                    sh 'docker run --restart=on-failure -p 8080:8080 -d --name spring ${ container img }'
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_U
                endpoint: ${ mattermost webhook },
```

```
                channel: ${ mattermost channel name }
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'danger',
                message: "배포 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_U
                endpoint: ${ mattermost webhook },
                channel: ${ mattermost channel name }
                )
            }

        }
    }
}
```

Frontend CI/CD

- 젠킨스 파이프라인 작성
  - 빌드 폴더 Nginx 컨테이너로 카피
  - CI=false 환경변수 설정 → 이걸 안 하면 경고 메시지를 오류로 인식

```
pipeline{
    agent any
    tools {nodejs "nodejs"}

    stages {
        stage('gitlab Connect'){
            steps{
                git branch: ${ gitlab branch },
        credentialsId: ${ credential ID },
        url: ${ gitLab URL }
            }
        }
        stage('build'){
            steps{
                sh 'cd /var/jenkins_home/workspace/frontend-dev/frontend/'
                dir('frontend'){
                    sh 'cp /var/jenkins_home/front/env /var/jenkins_home/workspace/frontend-dev/frontend/.env'
                    sh 'npm install -g yarn'
                    sh 'yarn install'
                    sh '/var/jenkins_home/${ sonar-scanner path } -Dsonar.projectKey=${ project Key } -Dsonar.sources=.
            sh 'CI=false yarn build'
                }
            }
        }
        stage('deploy'){
            steps{
                dir('frontend'){
                    sh 'docker cp ./build nginx:/usr/share/nginx/html/'
                }
            }
        }

    }
    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_U
                endpoint: ${ mattermost webhook },
                channel: ${ mattermost channel name }
                )
            }
        }
```

```
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_U
                endpoint: ${ mattermost webhook },
                channel: ${ mattermost channel name }
                )
            }
        }
    }
}
```

AI CI/CD

- 젠킨스 파이프라인 작성
  - python 프로젝트 카피 후 실행

```
pipeline{
agent any

stages {
    stage('gitlab Connect'){
        steps{
            git branch: ${ gitlab branch },
    credentialsId: ${ credential ID },
    url: ${ gitLab URL }
        }
    }
    stage('deploy'){
        steps{
            sh 'cd /var/jenkins_home/workspace/ai-dev/'
            sh 'docker cp ./ai ai:/app/'
            sh 'docker cp /var/jenkins_home/ai/model.zip ai:/app/ai/models/'
            sh 'docker cp /var/jenkins_home/ai/application-secrets.json ai:/app/ai/'
            sh 'docker exec ai bash -c "unzip /app/ai/models/model.zip | sleep 30"'
            sh 'docker cp /var/jenkins_home/ai/ai_exec.sh ai:/app/ai/'
            sh 'docker exec ai bash -c "cd /app/ai && ./ai_exec.sh"'
        }
    }
}
post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            mattermostSend (color: 'good',
            message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUIL
            endpoint: ${ mattermost webhook },
            channel: ${ mattermost channel name }
            )
        }
    }
    failure {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            mattermostSend (color: 'danger',
            message: "배포 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUIL
            endpoint: ${ mattermost webhook },
            channel: ${ mattermost channel name }
            )
        }
    }
}
}
```

  - ai_exec.sh

```
#!/bin/bash

ps -ef | grep 0.0.0.0 > /tmp/py.txt ; awk '{print $2}' < /tmp/py.txt | xargs kill
python main.py --host 0.0.0.0 --port 8080 &
```

모니터링 인프라 구축

conf.d 아래에 port.inc 파일 만들어서 아래와 같이 작성 후 include

```
set $port 8080
```

순서

- 블루 서버와 그린 서버 판별
- 그린 서버에 새로운 버전을 배포
- 그린 서버 헬스 체크 → Spring boot actuator

```
#application.yml
management:
    endpoints:
        web:
            base-path: /management
            path-mapping:
                health: health_check
```

  - 서버IP:8080/management/health_check 접속 → UP 확인
- Nginx 설정을 통해 블루 서버와 연결을 끊고 그린 서버와 연결
- 블루 서버 종료

```
#!/bin/bash

# 1. Discrimination Blue&Green
ports=("8080" "8081")
ip= { server ip }

GREEN_PORT="none"

for ports in ${ports[@]}
do
        RESPONSE=$(curl -s http://$ip:$port/management/health_check)
        IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)

        if [ $IS_ACTIVE -eq 1 -a $port -eq "8080" ]
        then
                BLUE_PORT="8080"
                GREEN_PORT="8081"
        elif [ $IS_ACTIVE -eq 1 -a $port -eq "8081" ]
        then
                BLUE_PORT="8081"
                GREEN_PORT="8080"
        fi
done

# 2. If Green is None...Randome Select
if [ $GREEN_PORT ==='none' ]
then
        echo "Random Select"
        BLUE_PORT="8080"
        GREEN_PORT="8081"
fi

# 3. GREEN Health Check
RESPONSE=$(curl -s http://$ip:$GREEN_PORT/management/health_check)
IS_ACTIVE=$(echo ${RESPONSE} | grep 'UP' | wc -l)

if [ $IS_ACTIVE -eq 1 ]
then
        echo -e "All Green Server Must Be Closed..(PortNo. $GREEN_PORT)"
```

```
        exit 0
fi
echo -e "Green(PortNo. $GREEN_PORT) & BLUE(PortNo. $BLUE_PORT) Health Check Complete"


# 4. deploy...
echo -e "Green is $GREEN_PORT"
docker rm spring$GREEN_PORT
docker rmi spring$GREEN_PORT
docker build -t spring$GREEN_PORT /var/jenkins_home/workspace/backend-dev/backend/
docker run --restart=on-failure -p $GREEN_PORT:8080 -d --name spring$GREEN_PORT spring$GREEN_PORT


# 5. GREEN Double Health Check
for retry in {1 ..10}
do
        RESPONSE=$(curl -s http://$ip:$GREEN_PORT/management/health_check)
        GREEN_HEALTH=$(echo ${RESPONSE} | grep 'UP' | wc -l)
        if [ $GREEN_HEALTH -eq 1 ]
        then
                break
        else
                echo -e "$ip:$GREEN_PORT is not ON. Recheck health check after 10s sleep...."
                sleep 10
        fi
done

if [ $GREEN_HEALTH -eq 0 ]
then
        echo -e "$ip:$GREEN_PORT is not working.."
        exit 0
else
        echo -e "$ip:$GREEN_PORT is working!!"
fi

# 6. Switch from Blue to Green
echo "set \$port $GREEN_PORT;" | tee /etc/nginx/conf.d/port.inc
curl -X POST $ip/reload

# 7. Blue Server Stop
echo -e "BLUE CONTAINER STOP PROCESS.."
docker stop spring$BLUE_PORT
echo -e "CONTAINER PortNo. $BLUE_PORT Stop Complete!!!...Ok"
```

모니터링 인프라 구축

Prometheus

- /etc/prometheus 폴더에 prometheus.yml 작성

```
global:
  scrape_interval: 15s

scrape_configs:

  - job_name: 'prometheus'
    static_configs:
            - targets: ['i10a610.p.ssafy.io:8901']

  - job_name: 'node-exporter'
    static_configs:
            - targets: ['i10a610.p.ssafy.io:8902']

  - job_name: 'jenkins'
    metrics_path: /prometheus
    static_configs:
            - targets: ['i10a610.p.ssafy.io:8888']
```

- 도커 컨테이너 생성

```
sudo docker run --restart=on-failure -d \
        -e TZ=Asia/Seoul -p 8902:9100 --name node-exporter \
        prom/node-exporter:latest
sudo docker run --restart=on-failure -d \
        -e TZ=Asia/Seoul -p 8901:9090 \
        -v /etc/prometheus:/etc/prometheus --name prometheus \
        prom/prometheus:latest
sudo docker run --restart=on-failure \
        -e TZ=Asia/Seoul -d -p 8900:3000 \
        --name grafana grafana/grafana:latest
```

Grafana

Prometheus 연결 & metric 쿼리 작성

jenkins

- jenkins에서 prometheus 플러그인 추가
- influxDB 설정

  influxdb 접속해서 API Tokens > admin’s Token이 Active 되어 있는지 확인

  그라파나에서 influxDB를 DataSource에 추가
- prometheus, influxDB 연결