B208 WEVI 포팅 메뉴얼

> **SSAFY 12기 공통프로젝트 대전 2반 8팀**

0. 목차

1. 개발 환경

1. 프로젝트 기술 환경 스택

- Frontend
  - Visual Studio Code 1.97.2
  - HTML5, CSS3, Javascript(ES6)
  - React 18.3.0
  - Vite 6.0.5
  - Tailwind CSS 4.0.0
  - Nodejs 20
  - Firebase 11.3.0
- Backend
  - Intellij 2025.1.6
  - Oracle Open JDK 17
  - JWT
  - SpringBoot 3.3.7
    - JAVA Spring Data JPA 3.1
    - Spring Security 6.4.2
  - Firebase 11.3.0
  - Gradle 8.11.1
- CI/CD
  - AWS EC2
    - NGINX 1.27.4
    - Ubuntu 24.04.1 LTS
    - Docker 27.5.1
    - Jenkins 2.479.3
  - S3
- Database
  - MySQL 8.0.41

2. 환경변수 설정

Frontend

- .env

```
# 로컬용
VITE_API_BASEURL=
```

```
VITE_VAPIDKEY=
VITE_APIKEY=
VITE_AUTHDOMAIN=
VITE_PROJECTID=
VITE_STORAGEBUCKET=
VITE_MESSAGINGSENDERID=
VITE_APPID=
VITE_MEASUREMENTID=

# 배포용
VITE_API_BASEURL=
VITE_VAPIDKEY=
VITE_APIKEY=
VITE_AUTHDOMAIN=
VITE_PROJECTID
VITE_STORAGEBUCKET=
VITE_MESSAGINGSENDERID=
VITE_APPID=
VITE_MEASUREMENTID=
```

Backend

- application.yml

```
spring:
  profiles:
    active: prod
  servlet:
    multipart:
      enabled: true
      max-file-size: 50MB
      max-request-size: 50MB

logging:
  level:
    root: info
    sql: debug
    org.hibernate.SQL: debug
    org.springframework.security: debug
    com.ssafy.wevi: debug
```

- application-prod.yml

```
spring:
  datasource:
    url: jdbc:mysql:///wevi?serverTimezone=Asia%2FSeoul&characterEncoding=UTF-8
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver

  mail:
    host: smtp.gmail.com  # Gmail의 SMTP 서버 호스트
    port: 587 # Gmail SMTP 서버는 587번 포트를 사용
    username:
    password:
```

```
  properties:
    mail:
      smtp:
        auth: true # SMTP 서버에 인증 필요한 경우 true로 지정 Gmail은 요구함
        starttls:
          enable: true # SMTP 서버가 TLS를 사용하여 안전한 연결을 요구하는 경우 true로 설정
          required: true
        connectiontimeout: 5000 # 클라이언트가 SMTP 서버와의 연결을 설정하는 데 대기해야 하는 시간
        timeout: 5000 # 클라이언트가 SMTP 서버로부터 응답을 대기해야 하는 시간
        writetimeout: 5000 # 클라이언트가 작업을 완료하는데 대기해야 하는 시간
  auth-code-expiration-millis: 1800000  # 30 * 60 * 1000 == 30분 이메일 인증 코드의 만료 시간(Millisecond)

sql:
  init:
    mode: always

jpa:
  hibernate:
    ddl-auto: create
  properties:
    hibernate:
      format_sql: true
  defer-datasource-initialization: true

cloud:
  aws:
    credentials: # IAM으로 생성한 시크릿키 정보를 입력한다.
      access-key:
      secret-key:
    S3:
      bucket: my-vendor-images # bucket 이름을 설정한다.
    region:
      static: ap-northeast-2 # bucket이 위치한 AWS 리전을 설정한다.
    stack:
      auto: false # 자동 스택 생성 기능 사용여부를 설정한다. (자동 스택 생성: 애플리케이션이 실행,
      배포될 때 인프라 리소스를 자동으로 생성하고 설정하는 것)
```

- firebase-service-account.json

```
{
  "type":
  "project_id":
  "private_key_id":
  "private_key": "-----BEGIN PRIVATE KEY-----
  "client_email":
  "client_id":
  "auth_uri":
  "token_uri":
  "auth_provider_x509_cert_url":
  "client_x509_cert_url":
  "universe_domain":
}
```

3. 설정 파일

NGINX : default.conf

- 설정 파일 위치

```
/home/ubuntu/nginx/conf.d/
```

- default.conf

```
server {
        root /usr/share/nginx/html;

        index index.html index.htm index.nginx-debian.html;

        server_name "";

        client_max_body_size 50M; # 파일 입출력시 최대크기 설정

        location / {
                try_files $uri $uri/ /index.html;
    }

    location /api {
            proxy_pass http://localhost:8080;  # 스프링 부트 서버 주소
            proxy_set_header X-Real_IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Host $host;
            proxy_redirect off;
            charset utf-8;
    }

    location /.well-known/acme-challenge/ {
            allow all;
            root /var/www/certbot;
    }

    # https 적용 관련

       listen [::]:443 ssl ipv6only=on; # managed by Certbot
       listen 443 ssl; # managed by Certbot
            ssl_certificate /etc/letsencrypt/live/i12b208.p.ssafy.io/fullchain.pem; # managed by Certbot
            ssl_certificate_key /etc/letsencrypt/live/i12b208.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 ($host = i12b208.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 default_server;
        listen [::]:80 default_server;

        server_name i12b208.p.ssafy.io;
    return 404; # managed by Certbot
```

```
}
```

Jenkins : Jenkinsfile

- 설정 파일 위치 : 프로젝트 최상단
- Jenkinsfile

```
pipeline {
    agent any

    environment {
        BRANCH_NAME = "${GIT_BRANCH}"
        NODE_VERSION = 'node20'
        DEPLOY_PATH = '/home/ubuntu/nginx/html'
    JAVA_VERSION = 'jdk17'
    APP_NAME = 'jenkins-test'
    DOCKER_IMAGE = 'jenkins-test:latest'
}

stages {
    stage('Checkout') {
        steps {
            checkout scm
            script {
                echo "현재 브랜치: ${BRANCH_NAME}"
            }
        }
    }

    stage('Backend Build & Deploy') {
        when {
            expression { BRANCH_NAME == 'origin/back' }
        }
        tools {
            jdk "${JAVA_VERSION}"
        }
        steps {
            dir('backend') {
                script {
                    sh '''
                        echo "===== Build Environment ====="
                        echo "JDK Version:"
                        java --version
                        echo "Docker Version:"
                        docker --version
                        echo "Current Directory:"
                        pwd
                        ls -la
                    '''

                    // Prepare Environment
                    sh '''
                        rm -rf src/main/resources
                        mkdir -p src/main/resources
```

```
                chmod 777 src/main/resources
            '''

            // 시크릿 파일 설정 부분 (필요시 주석 해제)
            withCredentials([
                file(credentialsId: 'prod-yaml', variable: 'prodFile'),
                file(credentialsId: 'firebase-json', variable: 'fireFile')
                // file(credentialsId: 'secret-yaml', variable: 'secretFile')
            ]) {
            sh '''
                cp "$prodFile" src/main/resources/application-prod.yml
                cp "$fireFile" src/main/resources/firebase-service-account.json
                chmod 644 src/main/resources/application-*.yml
                chmod 644 src/main/resources/firebase-*.json
            '''
            }

            // Gradle 빌드
            sh '''
                chmod +x gradlew
                ./gradlew clean build -x test --no-daemon
            '''

            // Docker 배포
            sh '''
                docker rm -f ${APP_NAME} || true
                docker rmi ${DOCKER_IMAGE} || true
                docker build -t ${DOCKER_IMAGE} .
                docker run -d \
                    --name ${APP_NAME} \
                    -e SPRING_SERVLET_MULTIPART_MAX_FILE_SIZE=50MB \
                    -e SPRING_SERVLET_MULTIPART_MAX_REQUEST_SIZE=100MB \
                    --network my-network \
                    --restart unless-stopped \
                    -p 8080:8080 \
                    ${DOCKER_IMAGE}
            '''
        }
    }
}

    post {
        success {
            echo '백엔드 빌드 및 배포 성공'
        }
        failure {
            echo '백엔드 빌드 및 배포 실패'
        }
    }
}

stage('Frontend Build & Deploy') {
    when {
        expression { BRANCH_NAME == 'origin/front' }
    }
    tools {
        nodejs "${NODE_VERSION}"
    }
    steps {
```

```
            dir('frontend') {
                script {
                    // 빌드 전 상태 출력
                    sh '''
                        echo "===== Build Environment ====="
                        echo "Node Version:"
                        node --version
                        echo "NPM Version:"
                        npm --version
                        echo "Current Directory:"
                        pwd
                        ls -la
                    '''
                    // 시크릿 파일 설정 부분 (필요시 주석 해제)
                    withCredentials([
                        file(credentialsId: 'react-env', variable: 'envFile')
                    ]) {
                    sh '''
                        cp "$envFile" .env
                        chmod 644 .env
                    '''
                    }

                    sh '''
                        echo "===== Starting Build Process ====="
                        rm -rf node_modules
                        npm install
                        CI=false npm run build
                    '''

                    // 배포
                    sh '''
                        echo "===== Starting Deployment ====="
                        echo "Cleaning deployment directory..."
                        rm -rf ${DEPLOY_PATH}/*

                        echo "Copying build files..."
                        cp -r dist/* ${DEPLOY_PATH}/

                        echo "Verifying deployment..."
                        ls -la ${DEPLOY_PATH}
                    '''
                }
            }
        }
        post {
            success {
                echo '프론트엔드 빌드 및 배포 성공'
            }
            failure {
                echo '프론트엔드 빌드 및 배포 실패'
            }
        }
    }
}

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()
            withCredentials([string(credentialsId: 'mattermost-webhook', variable: 'WEBHOOK_URL')]) {
                mattermostSend(color: 'good',
                    message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${
                    endpoint: WEBHOOK_URL,
                    channel: 'f1f632e18102627b0737ddbefcf0c505'
                )
            }
        }
    }
    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()
            withCredentials([string(credentialsId: 'mattermost-webhook', variable: 'WEBHOOK_URL')]) {
                mattermostSend(color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${
                    endpoint: WEBHOOK_URL,
                    channel: 'f1f632e18102627b0737ddbefcf0c505'
                )
            }
        }
    }
}
```

Dockerfile : Backend

- 설정 파일 위치

```
backend/Dockerfile
```

- Dockerfile

```
FROM openjdk:17-jdk

ARG JAR_FILE=build/libs/*.jar
# jar 파일 복제
COPY ${JAR_FILE} app.jar

# 실행 명령어 test
ENTRYPOINT ["java", "-Dspring.profiles.active=prod,secret", "-jar", "app.jar"]
```

2. 배포 방법

1. EC2 접속

1. window teminal 설치

   www.microsoft.com/ko-kr/p/windows-terminal/9n0dx20hk701?activetab=pivot:overviewtab
2. window openSSH 사용 설정 (재부팅)
3. aws 접속하여 인스턴스 접속정보 확인

ssh -i I12B208T.pem ubuntu@i12b208.p.ssafy.io

4. 프라이빗 키(pem 키) 설정
5. window terminal에서 ssh 로 ec2 접속하기

2. 방화벽 설정

포트허용

로컬에서 개발중인 프론트엔드의 접근을 위해 8080을 열어주었습니다.

9090은 젠킨스 포트로 쓰기위해 열었습니다.

https 접근을 위해 443포트도 열었습니다.

```
sudo ufw allow 8080
sudo ufw allow 9090
sudo ufw allow 443
sudo ufw status numbered
```

3. 도커 설치

java 설치 (springboot프로젝트 버전)

```
# Java 17 설치
sudo apt update
sudo apt install openjdk-17-jdk -y

# 설치된 버전 확인
java -version
```

도커설치

```
$ sudo apt-get update

# 패키지 인덱스 업데이트
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common

# Docker의 공식 GPG 키 추가
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

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

# 패키지 인덱스 업데이트
$ sudo apt-get update

# Docker CE 설치
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

```
# Docker 서비스 시작 및 자동 시작 설정
$ sudo systemctl start docker
$ sudo systemctl enable docker

# Docker 그룹에 사용자 추가
$ sudo usermod -aG docker ${USER}

# 확인
$ docker --version
```

스왑메모리 지정

현재 사용중인 메모리 양(16GB)의 두배(32GB)로 설정

→ 하나의 EC2에서 여러 어플리케이션 서버를 동작하는 경우 필수

```
# 시스템 업데이트
sudo apt update

# Swap Space 생성하기
sudo fallocate -l 32G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile

# RAM swap 하기
sudo swapon /swapfile

# 확인
sudo swapon --show
free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi        12Gi       300Mi       1.0Mi       3.3Gi       3.2Gi
Swap:           31Gi          0B        31Gi
```

4. Jenkins 설치

```
# GPG 키 다운로드
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# 리포지토리 추가
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

# 패키지 목록 업데이트
sudo apt-get update

# Jenkins 설치
sudo apt-get install jenkins

# 확인
$ sudo systemctl status jenkins
```

```
# jenkins 권한 주기
sudo usermod -aG docker jenkins

$ sudo systemctl start jenkins

# 초기 비밀번호 확인
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Jenkins 환경설정 변경

```
# 젠킨스 설정 변경을 위해 서비스 중지
sudo systemctl stop jenkins

# jenkins 설치 경로로 이동
cd /var/lib/jenkins

# update-center에 필요한 CA파일을 다운로드하고 권한을 변경합니다.
sudo mkdir update-center-rootCAs
sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -O \
./update-center-rootCAs/update-center.crt
sudo chown -R jenkins:jenkins update-center-rootCAs

# default 설정에서 특정 미러사이트로 대체하도록 아래 명령어를 실행
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com
/lework/jenkins-update-center/master/updates/tencent/update-center.json#' ./hudson.model.UpdateCenter.xml

# URL이 위에꺼로 바뀌었는지 확인
cat hudson.model.UpdateCenter.xml

# !다시 jenkins 구동하기
sudo systemctl restart jenkins
```

Jenkins 포트 변경

/usr/lib/systemd/system/jenkins.service에 위치한 Environment-”JENKINS_PORT=8080” 부분을 9090으로 변경하면된다.

```
sudo vi /usr/lib/systemd/system/jenkins.service
```

재실행 하기

```
sudo systemctl daemon-reload

# 재시작
sudo systemctl restart jenkins
```

5. Jenkins 접속

이제 `http://ip주소:9090`로 들어가서 초기비밀번호 입력 후

jenkins id, pw, name, email 작성

`suggested plugins`를 설치한 후 admin 생성하고 URL 또한 `http://ip주소:9090`으로 설정

> [설치 플러그인]

- Generic Webhook Trigger
- Gitlab
- Gitlab API
- Gitlab Authentication
- Mattermost Notification
- Docker pipeline

6. Gitlab과 Jenkins 연동

Gitlab에서 토큰 발급

Gitlab 프로젝트에 접속해서 Settings관리 → Access Tokens

Select a role → Maintainer

Select scopes → api, read_api, read_repository 체크

이후 나오는 토큰 복사해두기

gitlab access token

jenkins_access_token

Jenkins에서 Gitlab 연결

Jenkins 관리 → System 설정 → Gitlab탭에서 다음과 같이 설정

- Connection name : 원하는 이름 입력
- Gitlab host URL : 깃랩 메인 주소 입력
- Credentials : +Add 버튼을 눌러 Credential 추가페이지로 이동
- Kind : GitLab API token
- Scope : Global
- API token : 깃랩에서 생성한 엑세스 토큰 입력

id, description은 설정해주지 않아도 괜찮다.

Webhook 설정

jenkins

이제 깃랩에서 특정 브랜치에 업데이트가 되면 자동으로 젠킨스가 감지하여 빌드할 수 있도록 Web-hook을 설정

1. Jenkins에서 '새로운 Item' → 'Pipeline'
2. 'Build Triggers'에서 'Build when a change is push to Gitlab~'
3. 뒤에 URL 기록해두고
4. 고급 버튼 에서 `Secret token`에서 Generate해서 토큰 저장

gitlab webhook url

jenkins secret token

Gitlab 설정

1. webhook탭에서 기록한 url과 secret toekn 넣음
2. branch는 정규식 브랜치 선택해서 `^(front|back)$` 넣음

7. 프론트엔드(React) 연결

React를 빌드한 결과물을 Nginx의 정적파일로 마운트하여 사용. Nginx에 https 적용, `/api` uri는 localhost:8080포트로 프록시하도록 적용

- **순서**

nginx 설치 -> nginx 설정 수정 -> Https 발급(Let's Encrypt) -> nginx 중지 -> Docker로 nginx 생성 후 실행(기존 nginx설정, https 파일 마운트)

Nginx 설치

```
# Nginx 설치
sudo apt update
sudo apt install nginx

# Nginx 상태 확인
sudo systemctl status nginx
```

Nginx 설정 수정

`sudo vi /etc/nginx/sites-available/default`

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    server_name 도메인이름(ex.tripggukgguk.site);

    root /var/www/html;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Nginx 시작

```
# nginx 설정 문법 체크
sudo nginx -t

sudo systemctl start nginx
sudo systemctl enable nginx
```

https 적용

자동 갱신은 적용하지 않았음

```
# certbot 설치
sudo apt install certbot python3-certbot-nginx

# 인증서 발급 (자동으로 Nginx 설정도 수정됨)
sudo certbot --nginx -d i12b208.p.ssafy.io
```

Docker로 전환

```
# 디렉토리 생성
mkdir -p /home/ubuntu/nginx/html
mkdir -p /home/ubuntu/nginx/conf

# 소유권 설정
sudo chown -R ubuntu:ubuntu /home/ubuntu/nginx

# 호스트의 nginx 중지
sudo systemctl stop nginx
sudo systemctl disable nginx
```

Docker Nginx 설정 파가

```
sudo mkdir -p /home/ubuntu/nginx/conf.d
sudo chown -R ubuntu:ubuntu /home/ubuntu/nginx
sudo cp /etc/nginx/sites-available/default /home/ubuntu/nginx/conf.d/default.conf
sudo vi /home/ubuntu/nginx/conf.d/default.conf
```

root 경로를 `/usr/share/nginx/html`로 변경 (도커 컨테이너 내부 경로)

Nginx 컨테이너 실행

```
# 이미지 받기
docker pull nginx

docker run -d --name nginx \
--network host \
-v /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d:ro \
-v /etc/letsencrypt:/etc/letsencrypt:ro \
-v /home/ubuntu/nginx/html:/usr/share/nginx/html \
nginx:stable

# 테스트
echo "<h1>HTTPS Test</h1>" > /home/ubuntu/nginx/html/index.html
```

- v는 마운팅nginx 설정 파일과, https파일
- -network 해야지, localhost호출이 가능함(안하면 docker와 호스트의 network가 따로)

트러블 슈팅

현재 Jenkinsfile을 못 읽어옴

- SCM에서 Git 주소를 입력하고 Credentials에 아이디와 비밀번호를 적용해야 함 (토큰 사용 불가)
- Branch Specifier를 `*/master`에서 `*/front`, `*/back`으로 변경

프론트엔드만 실행하기로 결정 후 Node.js를 찾을 수 없음

1. EC2에서 Node.js 직접 설치

   ```
   curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
   sudo apt install -y nodejs
   ```

2. Node.js가 설치되어 있지만 `sudo su - jenkins` 상태에서 `node --version`이 작동하지 않는 경우 Jenkins 실행 계정의 `PATH` 설정 필요
   a. Node.js 설치 경로 확인

```
which node
```

b. Jenkins 사용자 환경변수 설정

```
sudo nano /etc/default/jenkins
```

맨 아래에 다음 줄 추가:

```
PATH=$PATH:/usr/bin
export PATH
```

c. Jenkins 재시작

```
sudo systemctl restart jenkins
```

Node.js 18버전 호환 불가, 20버전으로 업그레이드

- React Router가 Node.js 20 이상을 요구함
- 현재 설치된 Node.js 버전이 `v18.20.6`이라 호환되지 않음

Node.js 20으로 업데이트

```
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs
node --version
```

Vite 빌드 중 파일을 찾을 수 없음

- `src/App.jsx`에서 `./components/TopNavigationBar/topNavigationBar.jsx` 파일을 찾을 수 없음
- 원인: 파일 이름 대소문자 문제, 파일이 실제로 없는 경우, Git이 파일 변경을 추적하지 못한 경우

Jenkinsfile에서 빌드 디렉터리를 변경

- `build/*` 디렉터리가 없다는 오류 발생
- Vite의 기본 빌드 결과물은 `dist/` 디렉터리에 생성됨
- Jenkinsfile 또는 배포 스크립트 수정

```
# 기존 (잘못된 경로)
cp -r build/* /home/ubuntu/nginx/html/

# 수정 (올바른 경로)
cp -r dist/* /home/ubuntu/nginx/html/
```

배포 경로 문제 해결

- Jenkins는 `jenkins` 사용자의 권한으로 실행되므로 `/home/ubuntu/nginx/html/` 접근 시 권한 문제 발생 가능
- 파일 권한을 조정하여 `jenkins` 사용자도 접근 가능하도록 변경

```
sudo chown -R jenkins:jenkins /home/ubuntu/nginx/html/
sudo chmod -R 755 /home/ubuntu/nginx/html/
```

새로고침 시 404 오류 발생

- 웹 서버 리다이렉션 설정 필요
- Nginx 설정 예시:

```
location / {
    try_files $uri $uri/ /index.html;
```

```
}
```

환경 변수 설정

- Git Credentials에 `.env` 파일 추가
- Jenkinsfile에서 `.env` 파일 복사 및 권한 설정 추가

8. 백엔드(Springboot) 연결

nginx 설정 변경 /api

docker에 띄워져있는 nginx에 주소/api 로 온다면 localhost:8080으로 보내주기 위해 환경변수 파일 변경

`sudo vi /home/ubuntu/nginx/conf.d/default.conf`

```
server {
    listen 80;
    server_name _;  # 여기에 실제 도메인 입력
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    # API 요청을 백엔드로 프록시
    location /api {
        proxy_pass http://localhost:8080;  # 스프링 부트 서버 주소
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

```
# 도커 재시작
docker restart nginx
```

gitlab credential 설정

숨기려는 파일 업로드

`Jenkins 관리` -> `Credentials` → `Stores scoped to Jenkins의 (glabal)` -> `Add Credentails`

`Secret file`선택 후 업로드

Jenkins file에 적용된 부분:

```
stage('Secrets Setup') {
    steps {
        withCredentials([
            file(credentialsId: 'prod-yaml', variable: 'prodFile'),
        ]) {
            sh '''
                cp "$prodFile" src/main/resources/application-prod.yml
```

```
            chmod 644 src/main/resources/application-*.yml
        '''
      }
    }
}
```

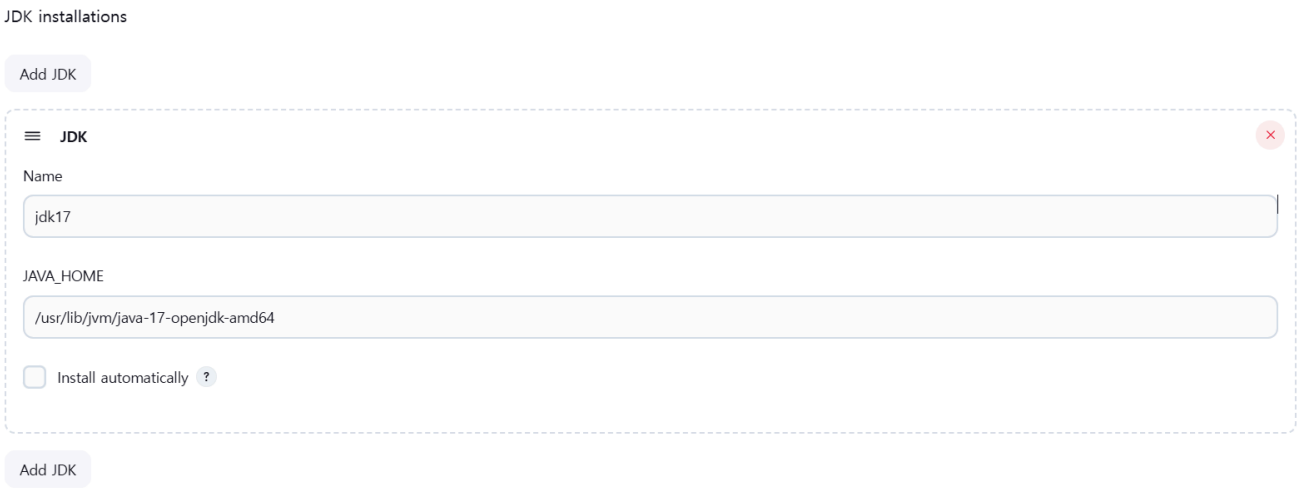
Docker file

```
FROM openjdk:17-jdk-slim

ARG JAR_FILE=build/libs/*.jar
# jar 파일 복제
COPY ${JAR_FILE} app.jar

# 실행 명령어
ENTRYPOINT ["java", "-Dspring.profiles.active=prod,secret", "-jar", "app.jar"]
```

JDK 설정



9. Mattermost와 연결

Credentials 설정

Jenkins 파일에서 숨길 mattermost url 과 이메일값을 숨기기 위해 Credential Secret text를 사용하였습니다.

방법

Jenkins 관리 → Credentials → Credentail 추가 → Secret text → Secret에 숨길 키값 적기 → ID에 호출할 ID값 적기

Jenkinsfile에 적용 부분

```
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_URL}|Details>)",
                endpoint: '',
                channel: ''
                    )
        }
    }
    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_URL}|Details>)",
                endpoint: '',
                channel: ''
                    )
        }
    }
}
```

10. DB 연결

도커를 이용해서 Mysql을 띄우고 docker network를 이용하여 같은 네트워크로 통신 할 수 있도록 구축하였습니다.

Docker images

```
# mysql(8.0.41) 버전 내려받기
docker pull mysql:8.0.41

# Docker Container 볼륨 설정



docker run -d --name mysql-container -p 3306:3306 -v mysql-volume:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD=  mysql:8.0.41
```

Docker Container Volume

```
# volume 생성
docker volume create mysql-volume

# volume 확인
docker volume ls
```

Docker network

`--link`는 Docker가 공식적으로 비추천하는 기능이므로, 대신 Docker 네트워크를 사용하는 것이 좋습니다. `docker network create` 명령어로 새로운 네트워크를 생성하고 두 컨테이너가 같은 네트워크 내에서 실행되도록 설정합니다.

```
# docker network 생성
docker network create my-network

# MySQL 컨테이너 실행 시 네트워크 지정
docker run -d --name mysql-container --network my-network -p 3306:3306 -v mysql-volume:/var/lib/mysql -e MYSQL_R

# Spring Boot 애플리케이션 컨테이너 실행 시 동일한 네트워크 지정 -> Jenkins file에 설정완료
docker run -d --name ${APP_NAME} --network my-network --restart unless-stopped -p 8080:8080 ${DOCKER_IMAGE}
```

위와 같이 `--network my-network` 옵션을 사용해 두 컨테이너가 같은 네트워크에 속하게 하면, MySQL 컨테이너의 이름인 `mysql-container`로 접근할 수 있습니다.

`application-prod.yml` **에서 MySQL 호스트 수정**:

`localhost` 대신 MySQL 컨테이너의 이름인 `mysql-container` 를 사용해야 합니다. Spring Boot 애플리케이션은 이제 `mysql-container` 라는 호스트 이름을 통해 MySQL에 접근할 수 있습니다.

```
spring:
  datasource:
    url: jdbc:mysql://mysql-container:3306/wevi?serverTimezone=UTC&characterEncoding=UTF-8
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver
```

컨테이너 네트워크 확인:
각 컨테이너가 동일한 네트워크에 속하고 있는지 확인하려면 아래 명령어를 실행하여 네트워크를 확인할 수 있습니다.

```
docker network inspect my-network
```

트러블 슈팅

루트 비밀번호를 단순하게 설정하고 root 계정을 외부에서 접속할 수 있도록 설정해두엇더니 해킹이 발생하였다.

```
mysql> select * from README;
+----+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
------------------------------------------+-----------------------------+-------------------------------------+
| id | Message

                                          | Email                       | Bitcoin_Address                     |
+----+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
------------------------------------------+-----------------------------+-------------------------------------+
|  1 | To recover your lost databases pay 0.004 BTC (Bitcoin) to this address: 18LQt7caPBbMLpVXNb5DeGMDancSv8YBvS. After
 your payment contact us at boris.mueller@tutamail.com with your server IP (43.201.24.149) and transaction ID. We will r
eply with your backup within few minutes. | boris.mueller@tutamail.com | 18LQt7caPBbMLpVXNb5DeGMDancSv8YBvS |
+----+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
------------------------------------------+-----------------------------+-------------------------------------+
1 row in set (0.01 sec)
```

기존에 도커에 올라가있던 DB를 Volume까지 전부 삭제하고 Chkrootkit과 ClamAV를 이용하여 서버에 rootkit과 악성코드가 있는지 철저히 검사하였고 현재 실행중인 도커 프로세스와 EC2 내부 프로세스, 외부와 연결되어있는 포트, 네트워크를 전부 검사하고 다시 DB를 생성하였다.

DB 내부 설정

컨테이너 내부로 접속

docker exec -it mysql-container mysql -u root -p

현재 MySQL 사용자 계정 확인

먼저, 현재 존재하는 사용자 계정을 확인

```
SELECT Host, User FROM mysql.user;
```

중복된 root 사용자 계정 확인 및 정리

만약 `root@%` 과 `root@localhost` 가 함께 존재한다면, 원격 접근을 막기 위해 `root@%` 계정을 삭제하면 됩니다

```
DELETE FROM mysql.user WHERE User='root' AND Host='%';
FLUSH PRIVILEGES;
```

이렇게 하면 외부에서 `root` 계정으로 접근할 수 없게 됩니다.

추가적인 보안 조치

1. **임의의 포트로 변경**
   - 예: `13306`, `23306`, `4306` 등
   - 이렇게 하면 단순한 스캐닝 공격을 피할 수 있음.
2. **root 계정 대신 별도의 사용자 생성**

```
sql
복사편집
CREATE USER 'youruser'@'%' IDENTIFIED BY 'yourpassword';
GRANT ALL PRIVILEGES ON yourdb.* TO 'youruser'@'%';
FLUSH PRIVILEGES;
```

   - `youruser` 계정을 만들고, 특정 데이터베이스(`yourdb`)만 접근 가능하도록 설정.
   - `root` 계정은 `localhost`에서만 사용하도록 유지.
3. **도커에서 MySQL 실행 시 포트 매핑 변경**

```
bash
복사편집
docker run -d \
  --name mysql-container \
  -e MYSQL_ROOT_PASSWORD=yourpassword \
  -p 13306:3306 \
  --restart unless-stopped \
  mysql:latest
```

   - 이렇게 하면 **로컬 13306 → 컨테이너 내부 3306**으로 연결됨.
   - 외부에서는 `13306` 포트를 통해 접근해야 하므로 보안성이 향상됨.
4. **방화벽 설정 (IP 제한)**
   - 특정 IP만 MySQL에 접근하도록 제한 가능:

```
bash
복사편집
sudo ufw allow from YOUR_IP to any port 13306
```

   - AWS 보안 그룹 사용 시에도 `13306` 포트에 특정 IP만 허용 가능.

DATABASE 생성

CREATE DATABASE wevi;

11. AI server

ufw 방화벽 설정

AI서버랑 통신하기 위한 포트 열기

```
sudo ufw allow 8001
```

프로젝트 폴더 생성

```
cd ~  # 홈 디렉토리 이동
mkdir ai-server
cd ai-server
```

`app.py` **(FastAPI 서버 코드)**

```
nano main.py
```

코드 입력 후 `Ctrl + X` **→** `Y` **→** `Enter` **로 저장**

AI 서버는 기존 EC2 백엔드(Spring Boot)에서 호출할 예정!

Whisper 모델을 `small` **로 설정하여 속도를 최적화함.**

`requirements.txt` **(Python 패키지 목록)**

```
nano requirements.txt
```

아래 내용 입력 후 저장 → 반드시 엔터로 구분

```
fastapi
uvicorn
boto3
torch
torchaudio
openai-whisper
openai
pydub
soundfile
requests
huggingface_hub
pdfplumber
pyannote.audio
```

Docker 컨테이너에서 필요한 패키지를 자동 설치할 수 있도록 설정

Docker 파일 생성

이제 Docker 컨테이너에서 FastAPI 서버를 실행할 수 있도록 `Dockerfile` 을 작성

`Dockerfile` **생성**

```
nano Dockerfile
```

```
# 1. Python 3.9 기반 이미지 사용
FROM python:3.12.8

# 2. 작업 디렉토리 설정
WORKDIR /app

# 필수 패키지 설치
RUN apt update && apt install -y ffmpeg
```

```
# 3. 필요 패키지 복사 및 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 4. FastAPI 실행 코드 복사
COPY . .

# 5. FastAPI 실행
# uvicorn 서버 실행 (포트 8001에서)
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8001", "--reload"]
```

이 Dockerfile은 Python 3.9 기반 컨테이너에서 FastAPI를 실행하도록 설정함.

`uvicorn` **을 사용해 FastAPI 서버를 실행 (포트 8001).**

Docker 이미지 빌드 및 실행

Docker 이미지 빌드

```
docker build -t fastapi-app .
```

현재 디렉토리 (`.`)에 있는 `Dockerfile`을 기반으로 `fastapi-ai-server`라는 이미지를 생성

시간이 좀 걸릴 수 있음.

Docker 컨테이너 실행

```
docker run -d \
--name fastapi-container \
--network my-network \
-p 8001:8001 \
fastapi-app
```

설명

- `d` → 백그라운드 실행
- `p 8001:8001` → 호스트(EC2)와 컨테이너의 8001번 포트 연결
- `-name ai-server` → 컨테이너 이름 설정
- `fastapi-ai-server` → 실행할 Docker 이미지 이름

실행된 컨테이너 확인

```
docker ps
```

실행된 컨테이너 목록이 보이면 성공!

AI 서버는 이제 `http://your-ec2-ip:8001/process-audio` 에서 실행 중

트러블 슈팅

1. 젠킨스에서 깃 클론을 못해옴

   → 워크스페이스를 다 밀어버리고 다시 실행

```
sudo rm -rf /var/lib/jenkins/workspace/wevi*
sudo systemctl restart jenkins
```

2. 한박자 늦은 커밋푸쉬를 빌드실행함

   → 젠킨스에서 직접 빌드버튼을 한번 누름

3, 대소문자 이슈

   → 윈도우에서는 대소문자 구분이 없지만 linux에서는 대소문자를 구분한다.

   → import 할때 대소문자 잘 구분해서 적기