포팅 매뉴얼

상태 시작 전

1. 개발 환경

Frontend

- Node 20.16.0
- React 18.3.1
- Styled-Components 6.1.13
- Zustand 5.0.0-rc.2
- VSCode 1.90.2

Backend

- JDK 17
- Spring Boot 3.3.4
- Intellij IDEA 2024.1.6 (Ultimate Edition)

AI

- Python 3.10.15
- tensorflow 2.17.0
- scikit-learn 1.5.2
- keras 3.5.0

DB

- MariaDB
- MongoDB
- Redis

Infra

- Docker 27.3.1
- Docker Compose 2.21.0
- Nginx 1.18.0

URL

- Client
    - https://j11a210.p.ssaf
- Server
    - https://j11a210.p.ssaf

- Jenkins 2.477

2. 빌드 및 실행

1) Docker Engine 설정

1. .pem 파일이 존재하는 경로로 이동
2. ssh 실행

```
ssh -i J11A210T.pem ubuntu@j11a210.p.ssafy.io
```

3. ufw 포트 추가

```
# 포트 추가
sudo ufw allow 8080/tcp

# 재부팅
sudo ufw enable
```

4. docker engine 설치

```
# 기존 docker 관련 패키지 삭제
for pkg in docker.io docker-doc docker-compose docker-compose-v

# https://docs.docker.com/engine/install/ubuntu/
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/ke
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

```
# docker install
sudo apt-get install docker-ce docker-ce-cli containerd.io dock

# test
sudo docker run hello-world

sudo wget -O /usr/local/bin/ufw-docker \
  https://github.com/chaifeng/ufw-docker/raw/master/ufw-docker
sudo chmod +x /usr/local/bin/ufw-docker
sudo ufw-docker install
sudo systemctl restart ufw
sudo systemctl restart docker
```

2) Jenkins Container 설정

1. Jenkins Container 생성 및 구동

```
# 최신 버전 jenkins 받기
sudo docker run -d -p 9090:8080 -p 50000:50000 -v /var/run/dock
```

2. 컨테이너 안에 docker 설치

```
# 1. jenkins 컨테이너로 접속
sudo docker exec -it jenkins /bin/bash

# 2. docker 설치
curl https://get.docker.com/ > dockerinstall && chmod 777 docke
# "https://get.docker.com/" 에서 docker를 설치하는 스크립트 다운로드 ㅎ
# "dockerinstall" 파일에 777 권한(모든 사용자(user, group, others)에

# 참고 : 이 시점에서 "ls" 명령어를 실행하면 "dockerinstall" 파일이 생성된
# 참고 : "cat dockerinstall" 명령어로 스크립트 파일을 볼 수 있음.
# 참고 : "docker version" 명령어로 설치된 도커의 버전을 확인할 수 있음.

# 3. docker-compose 설치
apt update
apt install docker-compose
```

```
# 4. 컨테이너 쉘에서 나오기
exit
```

3. HostOS의 /var/run/docker.sock 파일에 권한 부여

```
# 현재 터미널 경로가 HostOS인지 확인
pwd
# 실행 결과 : /home/ubuntu

# /var/run/docker.sock 파일에 user, group 사용자에게 읽기, 쓰기 권한
sudo chmod 666 /var/run/docker.sock
```

4. 초기 패스워드 별도 기록

```
sudo docker logs jenkins
```

3) 환경 설정 변경

1. jenkins data 폴더로 이동

```
cd /home/ubuntu/jenkins-data
mkdir update-center-rootCAs
```

2. update center에 필요한 CA 파일 다운로드

```
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/r
```

3. jenkins의 default 설정에서 특정 미러사이트로의 대체

```
sudo sed -i 's#https://updates.jenkins.io/update-center.json#ht
sudo docker restart jenkins
```

4) Docker-compose.yml 설정

```
services:
  echonote-mysql:
    container_name: echonote-mysql
```

```
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=210nullnull
      - MYSQL_DATABASE=echonote
      - MYSQL_USER=ssafy
      - MYSQL_PASSWORD=210nullnull
    ports:
      - "3306:3306"
    volumes:
      - mydata:/var/lib/mysql
    networks:
      - echonote_network
    restart: always

  echonote-mongo:
    container_name: echonote-mongo
    image: mongo
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: ssafy
      MONGO_INITDB_ROOT_PASSWORD: 210nullnull
    volumes:
      - mongodb-data:/data/db
    networks:
      - echonote_network
    restart: always

  echonote-redis:
    container_name: echonote-redis
    image: redis
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data
    networks:
      - echonote_network
    restart: always

  echonote-backend:
    container_name: echonote-backend
```

```
    image: echonote-backend
    ports:
      - "8080:8080"
    depends_on:
      - echonote-mysql
      - echonote-mongo
      - echonote-redis
    environment:
      - DB_URL=jdbc:mysql://stg-yswa-kr-practice-db-master.mariadb
      - DB_USERNAME=S11P21A210@stg-yswa-kr-practice-db-master
      - DB_PASSWORD=ezy6ULxfYW
      - MONGO_URI=mongodb+srv://S11P21A210:c8bl1A50dd@ssafy.ngivl.
      - MONGO_USERNAME=S11P21A210
      - MONGO_PASSWORD=c8bl1A50dd
      - JWT_KEY=lawsehjlkfhlzxioucvhlzxkjcvlzxcvoihasdkjfhvzlxcjvh
      - S3_BUCKET_NAME=timeisnullnull
      - S3_ACCESS_KEY=AKIAYEKP5MK4EFCVBHH6
      - S3_SECRET_KEY=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
      - ANALYSIS_SERVER_URL=https://grub-beloved-elephant.ngrok-fr
    networks:
      - echonote_network
    restart: on-failure
  echonote-frontend:
    user: root
    container_name: echonote-frontend
    image: echonote-frontend
    depends_on:
      - echonote-backend
    ports:
      - '80:80'
      - '443:443'
    environment:
      - VITE_API_URL=https://j11a210.p.ssafy.io/api/
    networks:
      - echonote_network
    volumes:
      - type: bind
        source: /etc/letsencrypt
        target: /etc/letsencrypt
    restart: on-failure
volumes:
```

```
  mydata:
  mongodb-data:
  redis-data:

networks:
  echonote_network:
    external: true
```

5) Server 설정 및 빌드

1. 백엔드 프로젝트 안에 Dockerfile 작성

```
# Dockerfile

# 빌드 단계: Gradle 이미지로 빌드 수행
FROM gradle:8.10.1-jdk17 AS build

# 작업 디렉토리 설정
WORKDIR /app

# Gradle 설정 파일을 먼저 복사하여 의존성 캐싱
COPY build.gradle settings.gradle ./

# Gradle 의존성 설치 (캐싱 용도)
RUN gradle dependencies --no-daemon

# 나머지 소스 파일을 복사하여 빌드 수행
COPY . .

# Gradle wrapper에 실행 권한 부여
RUN chmod +x gradlew

# Gradle을 이용하여 프로젝트 빌드 (테스트 제외)
RUN ./gradlew clean build -x test --no-daemon

# 실행 단계: JRE를 사용하여 런타임 이미지 생성
FROM openjdk:17-jdk

# 작업 디렉토리 설정
WORKDIR /app
```

```
# 빌드 단계에서 생성된 JAR 파일을 복사
COPY --from=build /app/build/libs/*.jar /app/echonote.jar

# 애플리케이션이 사용할 포트 노출
EXPOSE 8080

# 애플리케이션 실행
CMD ["java", "-jar", "/app/echonote.jar"]
```

- ssh-key 설정

```
$ docker exec -it jenkins bash

$ ssh-keygen -t rsa

$ cat /root/.ssh/id_rsa.pub

$ exit

# 기존 내용 삭제하면 안됨
$ cat id_rsa.pub >> ~/.ssh/authorized_keys
$ docker restart jenkins
```

3. docker network 설정

```
$ docker network create --driver=bridge echonote_network

$ docker network ls

# br-xxxx 주소를 구한다
$ docker inspect echonote_network
$ ip link show

# 규칙 수정
$ sudo ufw route allow in on eth0 out on docker0
$ sudo ufw route allow in on eth0 out on br-xxxx
$ sudo ufw reload
$ sudo ufw status
```

6) Client 설정 및 빌드

1. frontend 프로젝트 안에 Dockerfile 작성

```
# Node.js 공식 이미지 사용. 경량화된 Alpine Linux 기반
FROM node:20.16.0 AS build

# 작업 디렉토리 설정. 컨테이너 내 앱의 기본 경로
WORKDIR /app

ARG VITE_API_URL

# 라이브러리 설치에 필요한 파일만 복사
COPY package.json .
COPY package-lock.json .

# 라이브러리 설치
RUN npm ci

# 소스코드 복사
COPY . /app

ENV VITE_API_URL=${VITE_API_URL}

# 소스 코드 빌드
RUN npm run build

# 프로덕션 스테이지
FROM nginx:1.27.1

# nginx 실행 전 default.conf 파일 수정
COPY nginx.conf /etc/nginx/conf.d/default.conf

# 빌드 이미지에서 생성된 dist 폴더를 nginx 이미지로 복사
COPY --from=build /app/dist /usr/share/nginx/html

EXPOSE 80
CMD [ "nginx", "-g", "daemon off;" ]
```

2. frontend 프로젝트 안에 nginx.cnf 작성

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

    root   /usr/share/nginx/html;
    index  index.html index.htm;

    location / {
        try_files $uri $uri/ /index.html =404;
    }

    location /api/ {
        proxy_pass http://echonote-backend:8080/;
        proxy_set_header Host $host;
        proxy_set_header Origin "";
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded
        proxy_set_header X-Forwarded-Proto $scheme;

        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, P
        add_header 'Access-Control-Allow-Headers' 'Origin, Auth
        add_header 'Access-Control-Allow-Credentials' 'true';

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET, POS
            add_header 'Access-Control-Allow-Headers' 'Origin,
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Access-Control-Allow-Credentials' 'true
            add_header 'Content-Type' 'text/plain charset=UTF-8
            add_header 'Content-Length' 0;
            return 204;
        }
    }
}
```

```
server {
    listen       80;
    listen  [::]:80;
    server_name  j11a210.p.ssafy.io;

    # HTTP에서 HTTPS로 리디렉션
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name  j11a210.p.ssafy.io;

    root   /usr/share/nginx/html;
    index  index.html index.htm;

    # SSL 설정
    ssl_certificate /etc/letsencrypt/live/j11a210.p.ssafy.io/fu
    ssl_certificate_key /etc/letsencrypt/live/j11a210.p.ssafy.i
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    location / {
        # SPA 새로고침 처리
        try_files $uri $uri/ /index.html =404;
    }

    location /api/ {
        proxy_pass http://echonote-backend:8080/;
        proxy_set_header Host $host;
        proxy_set_header Origin "";
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded
        proxy_set_header X-Forwarded-Proto $scheme;

        # CORS 설정
        add_header 'Access-Control-Allow-Origin' '*';
```

```
        add_header 'Access-Control-Allow-Methods' 'GET, POST, P
        add_header 'Access-Control-Allow-Headers' 'Origin, Auth
        add_header 'Access-Control-Allow-Credentials' 'true';

        # wss(web-socket) 설정
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # OPTIONS 메소드에 대한 프리플라이트 요청 처리
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET, POS
            add_header 'Access-Control-Allow-Headers' 'Origin,
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Access-Control-Allow-Credentials' 'true
            add_header 'Content-Type' 'text/plain charset=UTF-8
            add_header 'Content-Length' 0;
            return 204;
        }
    }
}
```

3. EC2 Server에 SSL 인증서 발급받기 (Native 설치)

```
sudo apt update
sudo apt-get install letsencrypt
sudo apt install certbot python3-certbot-nginx

# 인증서 발급
sudo certbot certonly --standalone -d j11a210.p.ssafy.io
```

4. 인증서 접근 권한 주기

```
sudo chmod 644 /etc/letsencrypt/live/j11a210.p.ssafy.io/fullc
hain.pem
sudo chmod 600 /etc/letsencrypt/live/j11a210.p.ssafy.io/privk
ey.pem
```

5. 인증서 갱신 (`/usr/local/bin/renew_certificates.sh`)

```
#!/bin/bash

# 컨테이너 이름 (또는 ID)
CONTAINER_NAME="react"

# Docker 컨테이너 중지
echo "Stopping Docker container $CONTAINER_NAME..."
sudo docker compose stop $CONTAINER_NAME

# j11a210.p.ssafy.io 도메인 인증서 갱신
echo "Renewing certificate for i11a609.p.ssafy.io..."
sudo certbot certonly --standalone -d j11a210.p.ssafy.io --quie
    echo "Certbot renewal failed for j11a210.p.ssafy.io!" >&2
    exit 1
}

# Docker 컨테이너 다시 시작
echo "Starting Docker container $CONTAINER_NAME..."
sudo docker compose up -d $CONTAINER_NAME

echo "Certificate renewal and Docker container restart complete
```

6. 인증서 갱신 설정

```
$ sudo apt install cron -y

$ sudo crontab -e
# Select an editor.  To change later, run 'select-editor'.
#  1. /bin/nano        <---- easiest
#  2. /usr/bin/vim.basic
#  3. /usr/bin/vim.tiny
#  4. /bin/ed
# Choose 1-4 [1]: 2

0 12 * * * /usr/local/bin/renew_certificates.sh --quiet
```

7) Jenkins Pipeline 설정

- Plugins

    - gitlab
    - gitlab Authentication
    - generic webhook trigger
    - Docker
    - Docker Commons
    - Docker Pipeline
    - Docker API
    - NodeJS
    - Stage View
    - mattermost notification
- Jenkins Global Credential 설정
    - Kind : Secret Text
    - Username : VITE_API_URL_ID
    - Password : https://j11a210.p.ssafy.io/api/
    - ID : VITE_API_URL_ID
- CI/CD 과정
    - Gitlab의 dev 브랜치로 변경사항 발생 시 파이프라인 실행
    - frontend/backend 프로젝트 변경 여부에 따라 관련 파이프라인만 진행

```
pipeline {
    agent any

    tools {
        nodejs 'nodejs'
    }

    environment {
        ECHONOTE_BACKEND_IMAGE = 'echonote-backend'
        ECHONOTE_FRONTEND_IMAGE = 'echonote-frontend'
        MATTERMOST_URL = 'https://meeting.ssafy.com/hooks/bkrtik7z
        MATTERMOST_CHANNEL = 'Echonote'
    }

    stages {
```

```
        stage('GitLab Repository Checkout') {
            steps {
                git branch: 'dev',
                    credentialsId: 'S11P21A210',
                    url: 'https://lab.ssafy.com/s11-ai-speech-sub1
            }
        }

        stage('Check Changes') {
            steps {
                script {
                    def changes = sh(script: 'git diff --name-only

                    if (changes.contains('echonote_backend')) {
                        env.BUILD_BACKEND = 'true'
                    }

                    if (changes.contains('echonote_frontend')) {
                        env.BUILD_FRONTEND = 'true'
                    }
                }
            }
        }

        stage('Build Backend') {
            when {
                expression { return env.BUILD_BACKEND == 'true' }
            }
            steps {
                script {
                    dir('echonote_backend') {
                        sh 'docker build -t $ECHONOTE_BACKEND_IMAG
                    }
                }
            }
        }

        stage('Build Frontend') {
            when {
                expression { return env.BUILD_FRONTEND == 'true' }
            }
```

```
            steps {
                script {
                    withCredentials([string(credentialsId: 'VITE_A
                        dir('echonote_frontend') {
                            sh """
                            docker build \
                            --build-arg VITE_API_URL=$VITE_API_URL
                            -t $ECHONOTE_FRONTEND_IMAGE:latest .
                            """
                        }
                    }
                }
            }

        stage('Backend Docker Compose Up') {
            when {
                expression { return env.BUILD_BACKEND == 'true' }
            }
            steps {
                sh 'ssh -i /var/jenkins/id_rsa ubuntu@j11a210.p.ss
            }
        }

        stage('Frontend Docker Compose Up') {
            when {
                expression { return env.BUILD_FRONTEND == 'true' }
            }
            steps {
                sh 'ssh -i /var/jenkins/id_rsa ubuntu@j11a210.p.ss
            }
        }

        stage('Delete unnecessary Docker images') {
            when {
                expression { return env.BUILD_BACKEND == 'true' ||
            }
            steps {
                echo '********** Delete unnecessary Docker images
                sh 'docker image prune -a -f'
                echo '********** Delete unnecessary Docker images
```

```
            }
        }
    }

    post {
        success {
            script {
                // 커밋 작성자 이름과 메시지를 가져옴
                def commitAuthor = sh(script: "git log -1 --pretty
                def commitMessage = sh(script: "git log -1 --prett

                // Jenkins 콘솔 출력 페이지로 연결하는 URL 생성
                def consoleUrl = "${env.BUILD_URL}console"

                def msg = """Build succeeded: [View Build Logs](${
                            |Author: ${commitAuthor}
                            |Message: ${commitMessage}""".stripMa

                mattermostSend(
                    color: 'good',
                    message: msg,
                    channel: "${env.MATTERMOST_CHANNEL}",
                    endpoint: "${env.MATTERMOST_URL}"
                )
            }
        }
        failure {
            script {
                // 커밋 작성자 이름과 메시지를 가져옴
                def commitAuthor = sh(script: "git log -1 --pretty
                def commitMessage = sh(script: "git log -1 --prett

                // Jenkins 콘솔 출력 페이지로 연결하는 URL 생성 (여기서도
                def consoleUrl = "${env.BUILD_URL}console"

                def msg = """Build failed: [View Build Logs](${con
                            |Author: ${commitAuthor}
                            |Message: ${commitMessage}""".stripMa

                mattermostSend(
                    color: 'danger',
```

```
                    message: msg,
                    channel: "${env.MATTERMOST_CHANNEL}",
                    endpoint: "${env.MATTERMOST_URL}"
                )
            }
        }
    }
}
```