**SSAFYROMEDA**

**D205 : SSAFYROMEDA**

삼성청년SW아카데미 구미캠퍼스 8기

공통프로젝트[ 2023.01.03. – 2023.02.17. ]

**포팅 매뉴얼**

담당 컨설턴트 : 박세영

강모현, 김성욱, 박정은, 이창민, 이해솜, 장예은

# 프로젝트 기술 스택

## 이슈관리 : JIRA

## 형상관리 : GITLAB

## 커뮤니케이션 : MATTERMOST, NOTION

## 개발환경

### OS : WINDOWS10

### IDE

#### INTELLIJ 2022.3.1(ULTIMATE EDITION)

#### VISUAL STUDIO CODE

### DATABASE : MYSQL 5.7.39

### OS : WINDOWS10

# 빌드 상세내용

* 코드 병합 과정

1. 프로젝트 파일의 작업 내용을 Git 명령어를 통해 원격 저장소에 병합 요청합니다.

2. 작업 내용을 각 파트 (Front-End, Back-End)의 팀장이 리뷰하고 컨벤션에 적합하지 않다면 피드백을 작성합니다.

3. 각 파트원은 피드백을 반영하여 코드를 수정한 후 재병합을 요청합니다.

4. 코드가 병합된다면 GItLab의 WebHook을 통해서 오픈소스 CI 툴인 Jenkins를 통해서 빌드합니다.

* CI/CD 과정

1. Jenkins는 싸피에서 제공하는 Amazon EC2에 설치하였습니다
2. EC2에 Docker를 설치합니다.
3. Docker Hub에서 Jenkins 공식 이미지를 설치합니다.
4. Jenkins 이미지를 컨테이너화하고 회원 설정과 플러그인을 설치합니다.
5. WebHook 연결을 위해 GitLab 관련 플러그인도 설치합니다.
6. Develop 이라는 이름으로 Item을 생성합니다.
7. Develop의 환경설정을 통해 GitLab WebHook과 연결합니다. Jenkins의 Build Steps을 Execute shell을 통해 설정합니다.
8. Jenkins 설정이 모두 완료되었다면 Git 원격 저장소에 브랜치를 병합하게 되면 도커 소켓을 통해 EC2 Ubuntu에 있는 Docker에 프로젝트 빌드 파일이 컨테이너화되어 배포됩니다.

* OpenVidu 활용 배포 설정

1. WebRTC 사용을 위해 OpenVidu 라이브러리를 사용하였습니다.
2. EC2 Docker에 온프로미스 형식으로 필요한 이미지를 컨테이너화합니다.
3. 주요 컨테이너로는 Kurento Media Server와 OpenVidu-server, OpenVidu-Nginx 등이 있습니다.
4. 특히, OpenVidu-Nginx는 자체적으로 쉽게 SSL을 제공하는 기능이 있어 유용하게 사용할 수 있습니다.
5. 저희는 이 OpenVidu-Nginx를 자체적으로 메인스트림 서버로 분류하였습니다.
6. Docker run을 통해 볼륨 설정된 폴더로 이동합니다.
7. Openvidu는 nginx-conf의 원본의 편집을 허용하지 않습니다.
8. 하지만, 복사본을 수정하여 설정을 변경하는 방법은 허용하고 있습니다. 상세한 환경 설정 방법은 하단 링크를 참고하시길 바랍니다. <https://docs.openvidu.io/en/stable/troubleshooting/#16-how-can-i-customize-deployed-nginx>
9. 연결된 nginx-conf파일을 커스터마이징해서 nginx를 사용할 수 있게 되었습니다.
10. nginx-conf 내부의 upstream 된 url을 서비스 도메인으로 변경합니다.
11. 그리고 API 서버도 upstream 설정하고 리버스 프록시 설정을 해줍니다.
12. 그러면 우리는 따로 인증서를 설정하지 않고 OpenVidu-Nginx를 통해 HTTPS를 사용할 수 있게 됩니다.
13. 만약, 빌드 시 정적 파일을 따로 빌드해주는 방식을 사용하고 있다면 NGINX를 하나 더 생성해서 그곳에서 SPA 관련 처리를 한 후에 클라이언트 프로젝트와 연결해주는 방법을 사용하면 됩니다.

# 배포 특이사항

* Jenkins Build Steps (Execute shell)

docker build -t backend ./backend/ssafyromeda

if (docker ps | grep "backend"); then docker stop backend; fi

docker run -it -d --rm -p 8000:8080 --name backend backend

echo "Run Gradle Project"

docker build -t frontend ./frontend

if (docker ps | grep "frontend"); then docker stop frontend; fi

docker run -it -d --rm -p 5000:80 --name frontend frontend

echo "Run React Project"

* Spring Boot Dockerfile
* FROM adoptopenjdk/openjdk11 AS builder  
  VOLUME /tmp  
  COPY gradlew .  
  COPY gradle gradleCOPY build.gradle .   
  COPY settings.gradle .  
  COPY src src  
  RUN chmod +x ./gradlew  
  RUN ./gradlew bootJAR  
    
  FROM adoptopenjdk/openjdk11  
  COPY --from=builder build/libs/\*.jar app.jar  
  EXPOSE 8080  
  ENTRYPOINT ["java", "-jar", "/app.jar"]
* React Dockerfile
* FROM node:16.13.0 as build-stage
* WORKDIR /var/jenkins\_home/workspace/Deploy/frontend
* COPY package\*.json ./
* RUN npm install
* COPY . .
* RUN npm run build
* FROM nginx:stable-alpine as production-stage
* COPY --from**=**build-stage /var/jenkins\_home/workspace/Deploy/frontend/build /usr/share/nginx/html
* COPY nginx.conf /etc/nginx/conf.d/default.conf
* EXPOSE 80
* CMD ["nginx", "-g","daemon off;"]