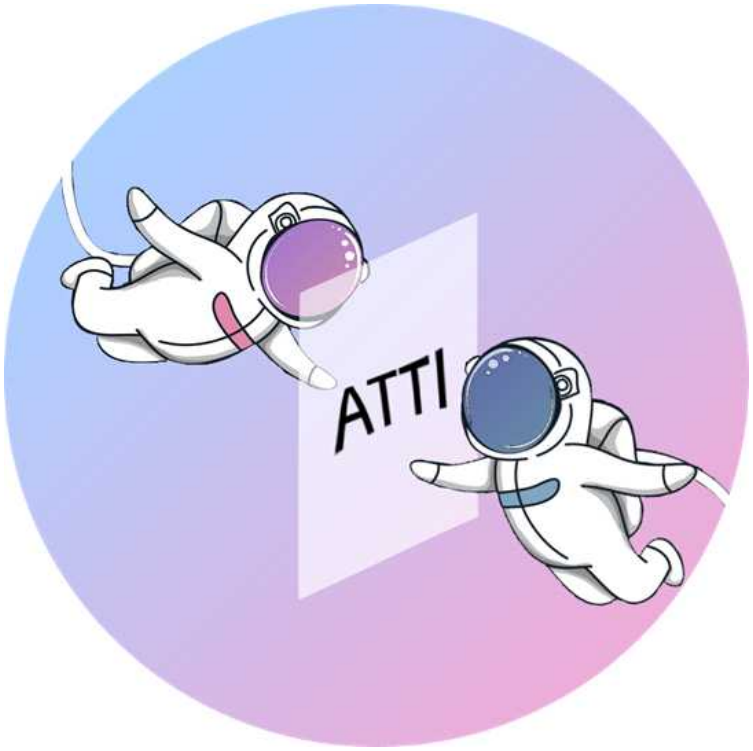


빌드 및 배포

진행 기간 : 2022.07.11 ~ 2020.08.19(6주)

삼성SW청년아카데미 7기 대전 공통 1반 B107
정 진 이현태 김연수 이현정 이주희 박범수

1. 기술 스택 및 버전

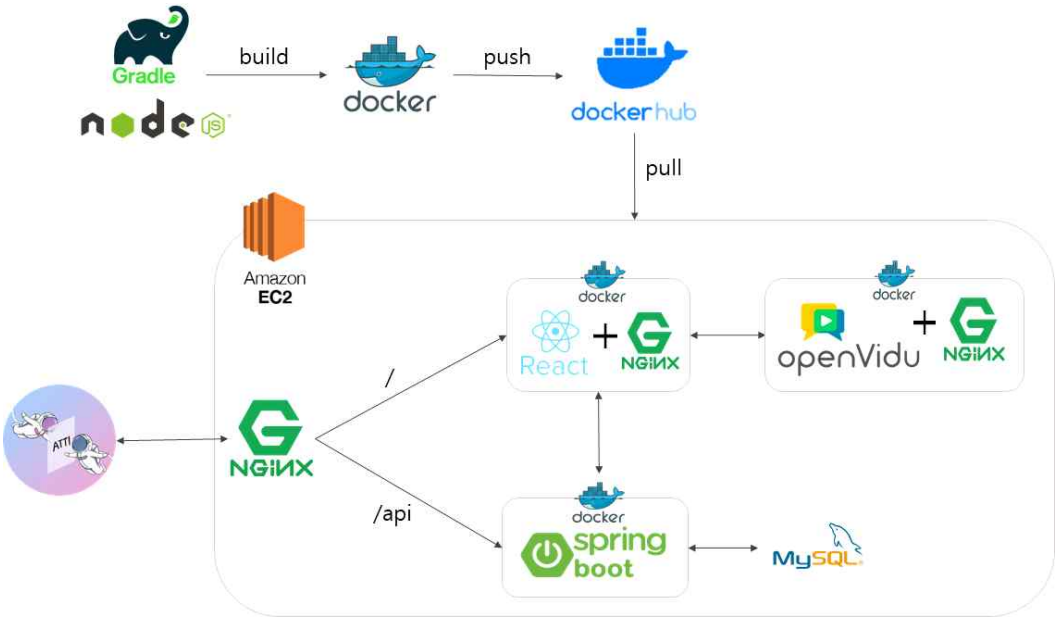
[표 1] 기술 스택 상세 내용

구분	사용 목적	사용 기술	기술 스택	버전
협업	코드 형상 관리	GitLab		
	이슈관리	JIRA		
	커뮤니케이션	Mattermost		
		Notion		
		Webex		
		Google Meet		
		Discord		
BackEnd	개발	DBMS	MySQL	8.0.28
		DB API	JPA(Java Persistence API)	2.7.2
		Java	Zulu	1.8
		Framework	Spring	5.3.22
			Spring Boot	2.7.2
		JWT(JSON Web Token)		
		WAS	Tomcat	9.0.60
		Build	Gradle	7.5
		WebRTC	Openvidu	2.22.0
FrontEnd	개발	CSS		
		JavaScript		
		TypeScript		4.4.2
		Library	React	1.8.0
			Redux	8.0.2
			MUI(Material-UI)	5.9.1
		Style-components		5.3.5
		IDE	Visual Studio Code	1.69
Server	배포	서버	AWS EC2	5.4.0
			nginx	1.18.0
		OS	Ubuntu	20.04 LTS
		배포	Docker	20.10.17

2. 상세 내용

1) 배포 흐름

배포 환경 및 배포 흐름은 아래의 [그림 1]과 같습니다.



[그림 1] 배포 흐름도

각 프로젝트를 빌드하여 Docker 이미지를 만듭니다. 이를 Docker Hub에 push합니다. 배포 서버에서 해당 이미지를 pull하여 컨테이너로 띄웁니다. Nginx를 설치하여 리버스 프록시 서버로 설정하였습니다.

2) 포트 번호

포트 번호는 아래의 [표 2]와 같습니다.

[표 2] EC2 Port

No.	포트 번호	이름
1	443	HTTPS
2	80	HTTP-HTTPS redirect
3	8443	Openvidu
4	3306	MySQL
5	8080	Spring Boot Docker Container
6	3000	React, NginX Docker Container

3) 환경 설정

(1) FrontEnd : src/constant/index.tsx

```
// 백엔드 통신
export const BACKEND_URL = "https://i7b107.p.ssafy.io:8080/api";

// 소셜 로그인 - 카카오톡
const CLIENT_ID = "발급받은 Client ID";
const REDIRECT_URI = "https://i7b107.p.ssafy.io/oauth/callback/kakao";

export const KAKAO_AUTH_URL =
`https://kauth.kakao.com/oauth/authorize?client_id=${CLIENT_ID}&redirect_uri
=${REDIRECT_URI}&response_type=code`;
```

(2) BackEnd : src/main/resources/application.properties

```
# 서버 배포 정보
build.date=@build.date@
server.port=8080
server.address=0.0.0.0
server.servlet.contextPath=/api

# ssl 인증 설정
server.ssl.enabled: true
server.ssl.key-store: classpath:keystore.p12
server.ssl.key-store-password:
server.ssl.key-store-type: PKCS12
```

3. 배포 과정

1) MySQL

아래 모든 과정은 Amazon EC2에서 진행됩니다.

(1) Ubuntu 업데이트

```
sudo apt update
```

(2) MySQL 설치

```
sudo apt install mysql-server
```

(3) MySQL 접속

```
sudo mysql -u root -p
```

(4) DB 변경

```
use mysql
```

(5) 계정 생성 및 권한 부여

```
create user 'b107'@'%' identified by 'b107zzangzzang';
grant all privileges on *.* to 'b107'@'%';
flush privileges;
```

(6) MySQL 접속 해제

```
exit
```

(7) MySQL 접속 권한 수정 : 경로 이동

```
cd /etc/mysql/mysql.conf.d/
```

(8) MySQL 접속 권한 수정 : 접속

```
sudo nano mysqld.cnf
```

(9) MySQL 접속 권한 수정 : 파일 수정

```
bind-address = 0.0.0.0
```

(10) MySQL을 재시작하여 변경 내용 적용

```
sudo service mysql restart
```

(11) MySQL Workbench에서 접속 확인

```
Hostname : i7b107.p.ssafy.io
Port : 3306
Username : b107
Pasword : b107zzangzzang
```

2) FrontEnd(React + Nginx)

(1) [local] Dockerfile 생성

```
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# root 에 app 폴더를 생성
RUN mkdir /app

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 도커에서 80 포트 오픈
EXPOSE 80

# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

(2) [local] nginx 설정 : nginx.conf 파일 생성

```
# 80포트에 /경로로 들어오면 /app/build/에서 index.html 파일을 찾음
server {
    listen 80;
    location / {
        root    /app/build;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

(3) [local] 파일 설치

```
yarn install
```

(4) [local] 빌드 파일 생성

```
yarn run build
```

(5) [local] docker 이미지 생성

```
docker build -t [도커허브이아이디]/[도커저장소이름]:[이미지 태그명] .
```

(6) [local] 도커 로그인

```
docker login
```

(7) [local] 도커 허브에 생성한 이미지 push

```
docker push [도커허브이아이디]/[도커저장소이름]:[이미지 태그명]
```

(8) [server] 도커 로그인

```
docker login
```

(9) [server] 도커 이미지 pull

```
docker pull [도커허브이아이디]/[도커저장소이름]:[이미지 태그명]
```

(10) [server] 도커 이미지 컨테이너 실행

```
docker run —name atti-front -d -p 3000:80 [도커허브이아이디]/[도커저장소이름]:[이미지 태그명]
```

(11) 접속 확인

```
http://i7b107.p.ssafy.io:3000/
```

(12) 수동 배포 시, [server]에서 동작 중인 컨테이너와 이미지를 삭제한 후, (3)~(10) 과정을 재진행함

3) BackEnd(Spring Boot)

(1) [local] Dockerfile 생성

```
# java 8 위에서 스프링 부트 앱이 실행
FROM java:8

# /tmp 아래에 이 컨테이너가 필요한 여러가지 데이터를 저장
VOLUME /tmp

# 도커 노출 포트
EXPOSE 8080

# 어떤 어플리케이션을 실행하는지 실행파일을 연결
# gradle build를 통해 생성한 .jar 파일의 위치
# 경로 확인이 중요
# 상대 경로로 적어서 다른 팀원의 개발환경에 적용
ARG JAR_FILE=build/libs/atti-0.0.1-SNAPSHOT.jar

COPY ${JAR_FILE} app.jar

# 어플리케이션 실행 명령어 : java 실행 명령어
ENTRYPOINT ["java","-jar","/app.jar"]

ARG DEBIAN_FRONTEND=noninteractive

ENV TZ=Asia/Seoul

RUN apt-get install -y tzdata
```

(2) [local] 빌드 파일 생성

```
gradlew clean build
```

(3) [local] docker 이미지 생성

```
docker build -t [도커허브이아이디]/[도커저장소이름]:[이미지 태그명] .
```

(4) [local] 도커 로그인

```
docker login
```

(5) [local] 도커 허브에 생성한 이미지 push

```
docker push [도커허브이아이디]/[도커저장소이름]:[이미지 태그명]
```

(6) [server] 도커 로그인

```
docker login
```

(7) [server] 도커 이미지 pull

```
docker pull [도커허브이아이디]/[도커저장소이름]:[이미지 태그명]
```

(8) [server] 도커 이미지 컨테이너 실행

```
docker run —name atti-backend -d -p 8080:8080 [도커허브이아이디]/[도커저장소이름]:[이미지 태그명]
```

(9) 접속 확인

```
http://i7b107.p.ssafy.io:8080
```

(10) 수동 배포 시, [server]에서 동작 중인 컨테이너와 이미지와 [local]의 build/를 삭제한 수, (2)~(10) 과정을 재진행함

4) Openvidu

아래 모든 과정은 Amazon EC2에서 진행됩니다.

(1) 도커 설치

```
sudo apt install docker-ce docker-ce-cli containerd.io
```

(2) 도커 컴포즈 설치 및 권한 부여

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

(3) 방화벽 설정

```
ufw allow ssh
ufw allow 80/tcp
ufw allow 443/tcp
ufw allow 3478/tcp
ufw allow 3478/udp
ufw allow 40000:57000/tcp
ufw allow 40000:57000/udp
ufw allow 57001:65535/tcp
ufw allow 57001:65535/udp
# 방화벽 적용
ufw enable
# 방화벽 상태 확인
ufw status verbose
```

(4) 권장 폴더로 이동

```
cd /opt
```

(5) 설치 스크립트 다운로드 및 실행

```
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash
```

(6) 환경 설정 : 경로 이동

```
cd openvidu
```

(7) 환경 설정 : 접속

```
nano .env
```

(8) 환경 설정 : 파일 수정

```
DOMAIN_OR_PUBLIC_IP=i7b107.p.ssafy.io
OPENVIDU_SECRET=atti
```

(9) 동작 확인

```
./openvidu start
```

5) ssl 인증서 발급 및 nginx 리버스 프록시 설정

아래 모든 과정은 Amazon EC2에서 진행됩니다.

(1) Ubuntu 업데이트

```
sudo apt-get update
sudo apt-get upgrade
```

(2) nignx 설치

```
sudo apt-get install nginx
```

(3) nginx 중단

```
sudo systemctl stop nginx
```

(4) letsencrypt 설치

```
sudo apt-get install letsencrypt
```

(5) ssl 인증서 발급 : 정상적으로 발급되었을 경우 /etc/letsencrypt/live/{도메인 네임}안에 인증서 파일이 있음

```
sudo letsencrypt certonly --standalone -d i7b107.p.ssafy.io
```

(6) 서버 블록 파일 생성(리버스 프록시 설정) : 경로 이동

```
cd /etc/nginx/sites-available
```

(7) 서버 블록 파일 생성(리버스 프록시 설정) : 접속

```
sudo vi /etc/nginx/sites-available/XXX.conf
```

(8) 서버 블록 파일 생성(리버스 프록시 설정) : 파일 수정

```
server {

        server_name i7b107.p.ssafy.io;

        location /{
                proxy_pass http://i7b107.p.ssafy.io:3000;
        }

        location /api {
                proxy_pass http://i7b107.p.ssafy.io:8080/api;
        }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/i7b107.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/i7b107.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 ($host = i7b107.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80 default_server;

        server_name i7b107.p.ssafy.io;

    return 404; # managed by Certbot
}
```

(9) 각 서버 블록에 대한 심볼릭 링크 생성

```
sudo ln -s /etc/nginx/sites-available/xxx.conf  /etc/nginx/sites-enabled/xxx.conf
```

(10) nginx 설정 체크

```
sudo nginx -t
```

(11) nginx 재시작

```
sudo systemctl restart nginx
```

6) ssl 인증서 적용

(1) FrontEnd는 별도의 설정 과정이 없음

```
https://i7b107.p.ssafy.io
```

(2) BackEnd

① 인증서 파일이 있는 곳으로 이동

```
cd /etc/letsencrypt/live/i7b107.p.ssafy.io
```

② key 파일(keystore.p12) 얻기

```
openssl pkcs12 -export -in fullchain.pem -inkey privkey.pem -out keystore.p12 -name b107 -CAfile chain.pem -caname root
```

③ key 파일을 홈 디렉토리로 이동

```
mv keystore.p12 /home/ubuntu/
```

④ key 파일을 filezilla의 ftp프로그램을 통해 로컬로 옮기기

⑤ key 파일 복사를 위해 서버에 권한 부여

```
chmod 404 keystore.p12
```

⑥ spring boot의 resource/에 key 파일 복사

⑦ application에 key파일 적용을 위한 코드 추가

```
server.ssl.enabled: true
server.ssl.key-store: classpath:keystore.p12
server.ssl.key-store-password: b107zzangzzang
server.ssl.key-store-type: PKCS12
```

⑧ 접속 확인

```
https://i7b107.p.ssafy.io/api
```

(3) Openvidu

① Openvidu 기본 포트로 연결 확인

```
./openvidu start
```

② certificates.conf, [도메인 주소] 파일 확인 : 경로 이동

```
cd /opt/openvidu/certificates/live
```

③ certificates.conf, [도메인 주소] 파일 확인 : 파일 확인

```
ls -l
```

④ 환경 설정 변경 : 접속

```
nano .env
```

⑤ 환경 설정 : 파일 수정

```
CERTIFICATE_TYPE=letsencrypt
```

⑥ 동작 확인

```
./openvidu start
```

⑦ 환경 설정 : 파일 수정

```
HTTP_PORT = 8081
HTTP_PORT = 8443
```

⑧ 최종 동작 확인

```
./openvidu start
```

4. 주요 속성 정보

1) MySQL 계정 정보

```
Database : atti
username : b107
password : b107zzangzzang
```

2) Spring Boot Database 설정

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://i7b107.p.ssafy.io:3306/atti?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&zeroDateTimeBehavior=convertToNull&rewriteBatchedStatements=true

spring.datasource.hikari.username=b107
spring.datasource.hikari.password=

spring.data.web.pageable.one-indexed-parameters=true
```