🌎 배포

🛠️1. 시스템 환경

AWS EC2

Ubuntu 20.04 LTS

Jenkins 2.334

Nginx 1.18.0

Docker 20.10.14

Database

MySQL 8.0.28

RDS(Amazon RDS)

👑2. 기술 스택

Frontend

IDE: VSCode

Node JS 16.13.x

Language: HTML5, Javascript, CSS3

Library: React 17.0.2, SCSS, TypeScript, Axios,

Framework: Material-UI

Backend

IDE: IntelliJ 2021.3.1

Language: Java 1.8

Framework: Spring Boot 2.4.5

Library: JWT, Spring-Boot-JPA,

Model

IDE : Google colab, VSCode, jupyternotebook

Language : python

Framework : tensorflow, pytorch, onnx, tensorflow-js

Library : teachablemachine

🍷 3. Safetykeeper 배포 순서(EC2)

1. nginx를 설치 후 설정해 리버스 프록시 역할을 하도록 합니다.

2. openvidu를 설치해 실행시킵니다.
3. mysql을 백엔드를 이용해 빌드 후 실행 시킵니다.
4. frontend를 빌드 하여 nginx와 빌드된 파일을 이미지화 시켜 컨테이너를 만듭니다.
5. backend를 빌드 하여 배포 파일인 jar 파일을 생성해 이미지화 후 컨테이너로 만듭니다.
6. Jenkins를 이용해 위에 만든 빌드 파일을 docker hub에 업로드 합니다
7. 우분투 환경에서 docker-image-pull.sh를 실행하여 이미지를 다운하여 컨테이너를 실행시킵니다.

🏹4. Nginx

웹서버와 리버스 프록시 역할을 하는 nginx를 사용했습니다.

Nginx 설치

```
$ sudo apt install nginx
```

1. Safetykeeper 프로젝트에 대한 설정파일을 만들기 위해 다음과 같이 작성합니다.

```
$ vi /etc/nginx/sites-available/프로젝트명
```

```
server {

        location /{ # 프론트앤드
                proxy_pass http://localhost:3000;
        }

        location /api { # 백엔드
                proxy_pass http://localhost:8080/api;
        }

        location /swagger-ui {
                proxy_pass http://localhost:8080/swagger-ui/#/;
        }

        location /vonovono{
                proxy_pass http://localhost:8080/api;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "Upgrade";
        }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/j6d101.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/j6d101.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 ($host = j6d101.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80;
        server_name j6d101.p.ssafy.io;
    return 404; # managed by Certbot
}
```

2. 저장 후, 해당 파일의 심볼릭 링크를 다음 경로에 추가합니다.

```
$ ln -s /etc/nginx/sites-available/프로젝트명 /etc/nginx/sites-enabled/프로젝트명
```

3. 올바른 문법인지 검사하고, 다시 재실행 합니다.

```
$ sudo nginx -t
$ sudo service nginx restart
$ sudo systemctl status nginx
```

4. 단, HTTPS를 사용하기 위해 인증서를 발급받아야 합니다. 인증서는 최초 1회만 발급받습니다.

```
$ sudo apt-get install letsencrypt
$ sudo letsencrypt certonly --standalone -d j6d101.p.ssafy.io
```

📷5. OpenVidu

WebRTC를 이용한 화상회의를 이용하기 위해 OpenVidu를 사용했습니다.

설치와 사용법은 다음과 같습니다.

OpenVidu 설치

```
cd /opt   # openvidu는 /opt 디렉토리에 설치되는게 권장
sudo curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | sudo bash
```

OpenVidu 설정파일

```
cd /opt # openvidu는 /opt 디렉토리에 설치되는게 권장
sudo vi .env
DOMAIN_OR_PUBLIC_IP=<Linux 서버의 public ip 주소 또는 도메인>
OPENVIDU_SECRET=<사용할 비밀번호 입력>
CERTIFICATE_TYPE=letsencrypt # default 값은 selfsigned지만 selfsigned 방식 사용시 보안 문제를 야기
               # SSL 키가 있다면 owncert 방식으로 하되, /owncert 디렉토리 안에 키가 있어야 함
LETSENCRYPT_EMAIL=<이메일>
HTTP_PORT=80
HTTPS_PORT=443
# HTTP_PORT와 HTTPS_PORT는 letsencrypt 방식의 키를 발급 받기 전까진 기본 포트인 80, 443을 사용해야 함.
# 키를 발급받고 난 후부터는 포트 변경해도 무방
```

OpenVidu 실행

```
cd /opt/openvidu
sudo ./openvidu start
```

📦6. docker와

어플리케이션을 컨테이너에 담아 배포하기 위해 docker를 사용했습니다.

docker 설치 방법

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
$ echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

docker 사용 예시

```
$ docker run -itd --name jenkins -p 8080:8080 -p 50000:50000 \
-v /docker/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock \
-e TZ=Asia/Seoul -u root jenkins/jenkins:latest
```

docker hub로 image 설치(shell 파일/docker-image-pull.sh**)**

```
#!/bin/bash

# docker container stop
docker stop fe-react
docker stop be-spring

# docker container delete
docker rm fe-react
docker rm be-spring

#docker image delete
docker rmi ... #fe-react image
docker rmi ... #be-spring image

# jenkins build image pull
docker pull ... #fe-react image
docker pull ... #be-spring image

#docker run container
docker run -it -d --privileged --restart always --name fe-react -p 3000:3000 ... #fe-react image
docker run -it -d --privileged --restart always --name be-spring -p 8080:8080 ... #be-spring image
```

docker-image-pull **사용 예시**

docker-image-pull.sh 파일이 있는 위치에서 다음과 같은 명령어를 실행하여 프론트 및 백엔드 이미지 파일을 설치하고 컨테이너를 실행합니다.

```
~$ vi ./vonovono/docker-image-pull.sh
```

👴7. Jenkins Pipeline

파이프 라인을 통하여 빌드 및 도커허브에 이미지 업로드를 진행합니다.

```
pipeline {
    agent any

    stages {
        stage('gitlab clone') {
            steps {
                git branch : 'develop',
                credentialsId: 'SafeKeeper', url: 'https://lab.ssafy.com/s06-ai-image-sub2/S06P22D101'
            }
        }
        stage('Build') {
            parallel {
                 stage('Frontend Build'){
                    steps {
                        dir('fe') {
                            sh 'ls -al'
                            sh 'docker build . -t ...{fe image}'
                        }
                    }
                }
                stage('Backend Build'){
                    steps {
                        dir('backend-java') {
                            sh "chmod +x gradlew"
                            sh'''
                                echo build start
                                ./gradlew clean bootJar
                            '''
                            sh 'docker build . -t ...{be image}'

                        }
                    }
                }
```

```
            }
        }
        stage('Upload') {
            parallel {
                 stage('Frontend Upload'){
                    steps {
                        dir('fe') {
                            sh 'docker push ...{fe image}'
                        }
                    }
                }
                stage('Backend Upload'){
                    steps {
                        dir('backend-java') {
                            sh 'JARNAME=$(find ./build/libs/*.jar | cut -d - -f 4)'
                            sh 'docker push  ...{be image}'

                        }
                    }
                }
            }
        }

        stage('Deploy') {
            steps {
                dir('backend-java') {
                  sh 'docker rmi $(docker images -q)'
                }

            }
        }
    }
}
```

🔐7. 프로젝트 속성 파일 목록

DB 정보, 토큰 정보 및 각종 API의 KEY 값 등을 따로 보관하였습니다.

해당 파일은 backend-java/src/main/resources/application.properties에 위치해야 합니다.

```
#it will be set build date by gradle. if this value is @build.date@, front-end is development mode
build.date=@build.date@
server.port=8080
server.address=0.0.0.0
server.servlet.contextPath=/
# Charset of HTTP requests and responses. Added to the "Content-Type" header if not set explicitly.
server.servlet.encoding.charset=UTF-8
# Enable http encoding support.
server.servlet.encoding.enabled=true
# Force the encoding to the configured charset on HTTP requests and responses.
server.servlet.encoding.force=true


# for SPA
spring.resources.static-locations=classpath:/dist/
spa.default-file=/dist/index.html
spring.mvc.throw-exception-if-no-handler-found=true
spring.resources.add-mappings=false

# Swagger
springfox.documentation.swagger.use-model-v3=false

#database
spring.jpa.hibernate.naming.implicit-strategy=org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
spring.data.web.pageable.one-indexed-parameters=true
spring.datasource.url={url}
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.hikari.username={username}
spring.datasource.hikari.password={password}

# jwt
jwt.secret={jwt}
```

```
# unit is ms. 15 * 24 * 60 * 60 * 1000 = 15days
jwt.expiration=1296000000

#logging
logging.file.name=./ssafy-web.log
logging.level.root=INFO
logging.level.com.samsung.security=DEBUG
logging.level.org.springframework.web=DEBUG
logging.level.org.apache.tiles=INFO
logging.level.org.sringframework.boot=DEBUG
logging.level.org.sringframework.security=DEBUG

spring.devtools.livereload.enabled=true

#gzip compression
server.compression.enabled=true
server.compression.mime-types=application/json,application/xml,text/html,text/xml,text/plain,application/javascript,text/css

#for health check
management.servlet.context-path=/manage
management.health.db.enabled=true
management.health.default.enabled=true
management.health.diskspace.enabled=true
```