포팅 메뉴얼

기술 스택 및 버전

프로젝트 기술 스택

Aa 기술	역할	버전
Docker	Infra	23.0.1
제목 없음	FE	
MariaDB	BE	10.3.37
Redis	BE	5.0.7
Java SDK	BE	11.0.18
SpringBoot	BE	2.7.8
Oauth2	BE	
Spring Security	BE	
aws-java-sdk-s3	BE	1.12.281
SpringBoot gradle	BE	7.6
jjwt	BE	0.9.1
Spring Data JPA	BE	
redis	BE	
querydsl	BE	5.0.0
jasypt	BE	3.0.3
Swagger	BE	2.9.2
Figma	UI/UX	
Intellij	IDE	22.3.1
Gitlab	형상 관리	
Jira	이슈 관리	
Visulaul Studio Code	IDE	

EC2 세팅

도커 설치

1. 패키지 업데이트 진행

```
sudo apt-get update
```

2. 필요 패키지 설치

```
sudo apt-get install \
ca-certificates \
curl \
gnupg \
lsb-release
```

3. Docker의 Official GPG key 등록

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

4. stable repositoy 등록

```
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
```

5. 도커 엔진 설치

```
# 다시 업데이트
sudo apt-get update
# 도커 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

6. 도커 버전 확인

```
docker --version
```

docker-compose 설치

1. 도커 컴포즈 설치

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

2. 실행할 수 있는 권한 부여

```
sudo chmod +x /usr/local/bin/docker-compose
```

3. 도커 컴포즈 버전 확인

```
docker-compose --version
```

Openvidu 배포

- 오픈비두 배포를 위한 root 권한 설정

```
sudo su
```

- 오픈비두를 설치하기 위해 권장되는 경로인 /opt로 이동

```
cd /opt
```

- 오픈비두 설치 후 경로 이동

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

- 도메인 또는 퍼블릭 ip와 오픈비두와 통신을 위한 환경 설정

```
$ vim .env

# OpenVidu configuration
# ----------------------
# 도메인 또는 퍼블릭IP 주소
DOMAIN_OR_PUBLIC_IP=i8b301.p.ssafy.io

# OpenVidu SECRET
OPENVIDU_SECRET=MY_SECRET

# Certificate type:
CERTIFICATE_TYPE=letsencrypt

# If CERTIFICATE_TYPE=letsencrypt, you need to configure a valid email for notifications
LETSENCRYPT_EMAIL=eodms090@naver.com

# HTTP port
HTTP_PORT=8444

# HTTPS port
HTTPS_PORT=8445
```

- 설절 후 오픈비두 서버 실행

```
$ ./openvidu start
```

Nginx 설정 및 ssl 인증서 발급 적용

```
sudo apt-get install nginx

nginx -v

sudo apt-get install letsencrypt

sudo service stop nginx

sudo letsencrypt certonly --standalone -d www제외한 도메인 이름
```

- nginx 설정 경로로 이동

```
cd /etc/nginx/sites-avilable
```

- 적절한 파일을 생성

```
server {

        location /{
                proxy_pass http://localhost:3000;
        }
```

```
        location /api/ {
                proxy_pass http://localhost:8080/;
        }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/i8b301.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/i8b301.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 ($host = i8b301.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80;
        server_name i8b301.p.ssafy.io;
    return 404; # managed by Certbot
}
```

- 파일 복사 및 nginx 재 시작

```
sudo ln -s /etc/nginx/sites-available/[파일명] /etc/nginx/sites-enabled/[파일명]

# 상태 확인 후
sudo nginx -t

sudo systemctl restart nginx
```

Jenkins 설정

- Jenkins LTS 버전 설치

```
$ docker pull jenkins/jenkins:lts
```

- Jenkins 컨테이너 실행

```
$ docker run -d --name jenkins -p 9090:8080 -v /jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/
```

- **-v /jenkins:/var/jenkins_home**

  젠킨스 컨테이너의 설정을 호스트 서버와 공유함으로써, 컨테이너가 삭제되는 경우에도 설정을 유지할수 있게 해줍니다.

- **-v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock**

  젠킨스 컨테이너에서도 호스트 서버의 도커를 사용하기 위한 바인딩입니다.이렇게 컨테이너 내부에서 설치없이, 외부의 도커를 사용하는 방식을 **DooD(Dock out of Docker)** 라고 합니다.

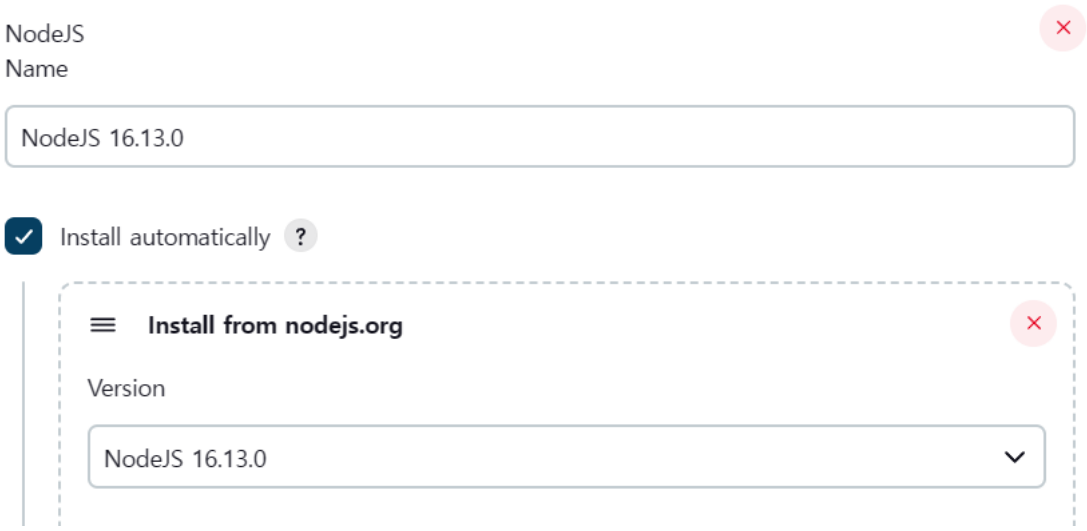
- Jenkins 접속

```
http:i8b301.p.ssafy.io:9090
```

- 키 값 설정

```
$ docker logs jenkins
```

- NodeJS 플로그인 설치 및 적용



프론트엔드 배포

- 파이프라인 작성

```
pipeline {
    agent any

    environment {
        GIT_URL = "https://lab.ssafy.com/s08-webmobile1-sub2/S08P12B301.git"
    }

    tools {
        nodejs "NodeJS 16.13.0"
    }

    stages {
        stage('Pull') {
            steps {
                git url: "${GIT_URL}", branch: "front_merge", credentialsId: "gitlab-jenkins"
            }
        }

        stage('React Build') {
            steps {
                dir('Frontend') {
                    sh 'pwd'
                    sh 'npm i --legacy-peer-deps'
                    sh 'CI=false npm run build'
                }

            }
        }

        stage('Build') {
            steps {
                dir('Frontend') {
                    sh 'docker build -t nginx-react:0.1 .'
                }

            }
        }

        stage('Deploy') {
            steps{
                script {
                    try {
                        sh 'docker ps -q -f name=nginx-react | grep . && docker stop nginx-react && docker rm nginx-react'
                    } catch (e) {
                        sh 'exit 0'
                        sh 'echo docker container stop and remove Fail!!'
                    }
                }
                sh 'docker run --name nginx-react -d -p 3000:3000 nginx-react:0.1'
            }
        }

      stage('Finish') {
            steps{
```

```
                sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
            }
        }
    }
}
```

백엔드 빌드 및 배포

application.yml 파일 설정

redis및 mariaDB 해킹으로 인한 jasypt 설정

- intellij내에서는 VM options를 통해 key 설정

- 설정 후 암호화된 값으로 yml파일 수정

```
spring:
  main:
    allow-bean-definition-overriding: true
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: ENC(w+v9nu+Ntv0+s9QUaiz4VgcGlo089gs0IVh+7m+THm3OuJjK6H6voJLoK2lcQWzmT6V/JfOoVNTieFMe3ZlrqnfWpguucxpUEn5ILLqbUIjry03hypQrmF3irwKlX7hC)
    username: ENC(IOvBivSpocQ38UJMe1yl9g==)
    password: ENC(09vfPMIkaeR8S2PSSSqPTA==)
```

- dockerfile entrypoint에 JAVA_OPTS라는 명으로 변수를 설정

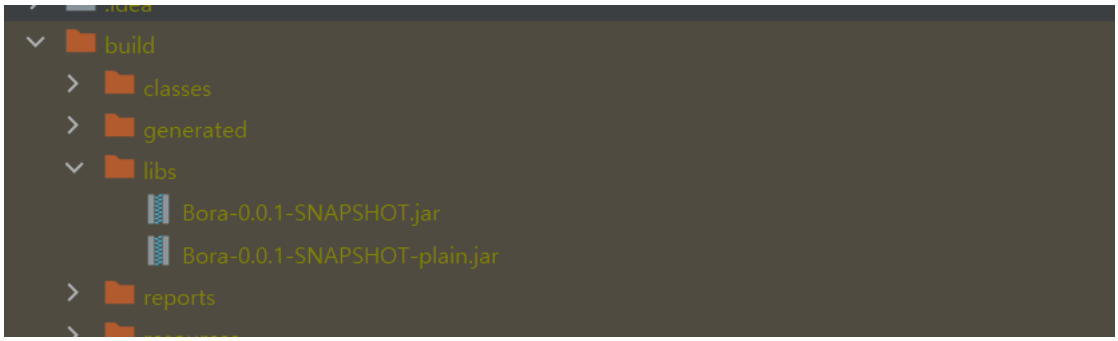
```
ENTRYPOINT ["sh", "-c", "java ${JAVA_OPTS} -jar /app.jar"]
```

- docker run시에 변수값을 설정

```
docker run -p 8080:8080 -e JAVA_OPTS=-Djasypt.encryptor.password=[비밀번호] 프론트엔드 배포
```

jar 파일로 빌드

- IntelliJ의 우측 상단에 Gradle 를 눌러 bootJar 를 실행한다.
- 그럼 프로젝트 폴더에 build 라는 폴더가 생기고 build/libs/ 에 jar 파일이 생긴다.



- 파이프라인 작성

```
pipeline {
    agent any

    environment {
        GIT_URL = "https://lab.ssafy.com/s08-webmobile1-sub2/S08P12B301"
    }
```

```
    stages {
        stage('Pull') {
            steps {
                git url: "${GIT_URL}", branch: "kde", credentialsId: "gitlab-jenkins"
            }
        }

        stage('Build') {
            steps {

                dir('Backend') {
                    sh 'javac -version'
                    sh 'chmod +x gradlew'
                    sh './gradlew clean build -x test'
                    sh 'docker build -t bobs_backend:0.1 .'
                }
            }
        }

        stage('Deploy') {
            steps{
                script {
                    try {
                        sh 'docker ps -q -f name=bora_backend | grep . && docker stop bora_backend && docker rm bora_backend'
                    } catch (e) {
                        sh 'exit 0'
                        sh 'echo bora_backend docker container stop and remove Skip!!'
                    }
                }
                sh 'docker run --name bora_backend -d -p 8080:8080 -e JAVA_OPTS=-Djasypt.encryptor.password=0836895 bora_backend:0.1'
            }
        }

      stage('Finish') {
            steps{
                sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
            }
         }
    }
}
```

DB 및 ERD 접속 정보

```
# mariadb 사용
host : i8b301.p.ssafy.io
사용자 : ssafy
암호 : ssafi
포트 : 8306

# redis
Address : i8b301.p.ssafy.io
Auth : eodms0519y
포트 : 33016
```