

시나브로

포팅 매뉴얼

D203

팀장: 윤선영

팀원: 김호균, 양동기, 이아현, 이진우

내용

[1.프로젝트 개발 환경 4](#_Toc127476879)

[협업 도구 4](#_Toc127476880)

[Back-end 4](#_Toc127476881)

[Front-end 4](#_Toc127476882)

[Media Server 5](#_Toc127476883)

[Server 5](#_Toc127476884)

[2. 백엔드 빌드 방법 6](#_Toc127476885)

[이클립스에 프로젝트 임포트 6](#_Toc127476886)

[프로젝트 빌드 9](#_Toc127476887)

[3. 프론트엔드 빌드 방법 12](#_Toc127476888)

[프로젝트 열기 12](#_Toc127476889)

[프로젝트 빌드 12](#_Toc127476890)

[4. EC2 세팅 14](#_Toc127476891)

[EC2에 필요한 도구 설치 14](#_Toc127476892)

[5. 자동 빌드 및 배포 방법(with Jenkins) 14](#_Toc127476893)

[Jenkins 설치 및 설정 14](#_Toc127476894)

[프론트엔드, 백엔드 Docker 설정 23](#_Toc127476895)

[MySQL Docker 설정 24](#_Toc127476896)

[OpenVidu CE설치 및 실행 25](#_Toc127476897)

[nginx 설정 26](#_Toc127476898)

[결론 27](#_Toc127476899)

[6. MySQL 워크벤치 사용법 28](#_Toc127476900)

# 프로젝트 개발 환경

협업 도구

1) 형상 관리: GitLab

2) 이슈 관리: Jira

3) 소통 및 자료 공유: MatterMost, Webex, Figma

4) 와이어프레임 (UI / UIX): Figma

Back-end

1) OS: Window 10

2) JAVA: OpenJDK 11.0.15 버전

3) IDE: 이클립스 2022-06 버전

4) Framework: Spring Boot 2.7.8 버전

* + - * Spring Boot Data JPA
      * QueryDSL 5.0.0 버전
      * Spring Boot Security
      * Swagger-ui 3.0.0 버전
      * ModelMapper 3.1.1 버전
      * Lombok
      * jjwt 0.11.5 버전

Front-end

1. OS: Window 10
2. Node.js: 14.17.x 버전
3. IDE: Visual Studio Code 1.74 버전
4. Framework: React 18.2.0버전

* Axios 1.3.1버전
* Openvidu-browser 2.25.0버전
* React-dom 18.2.0 버전
* React-icons 4.7.1버전
* React-redux 8.0.5버전
* React-router-dom 6.8.0버전
* React-scripts 5.0.1버전
* Styled-component 5.3.6버전

Media Server

1. OS: Ubuntu 버전
2. OpenVidu: 2.25.0 버전

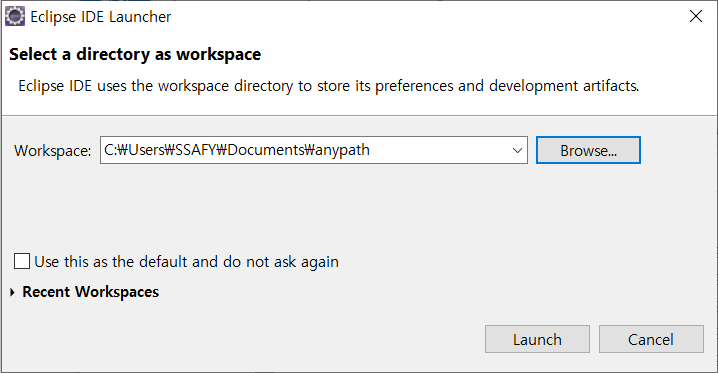
Server

1. OS: Ubuntu
2. 웹서버: nginx
3. 컨테이너: docker
4. 자동화 도구: Jenkins

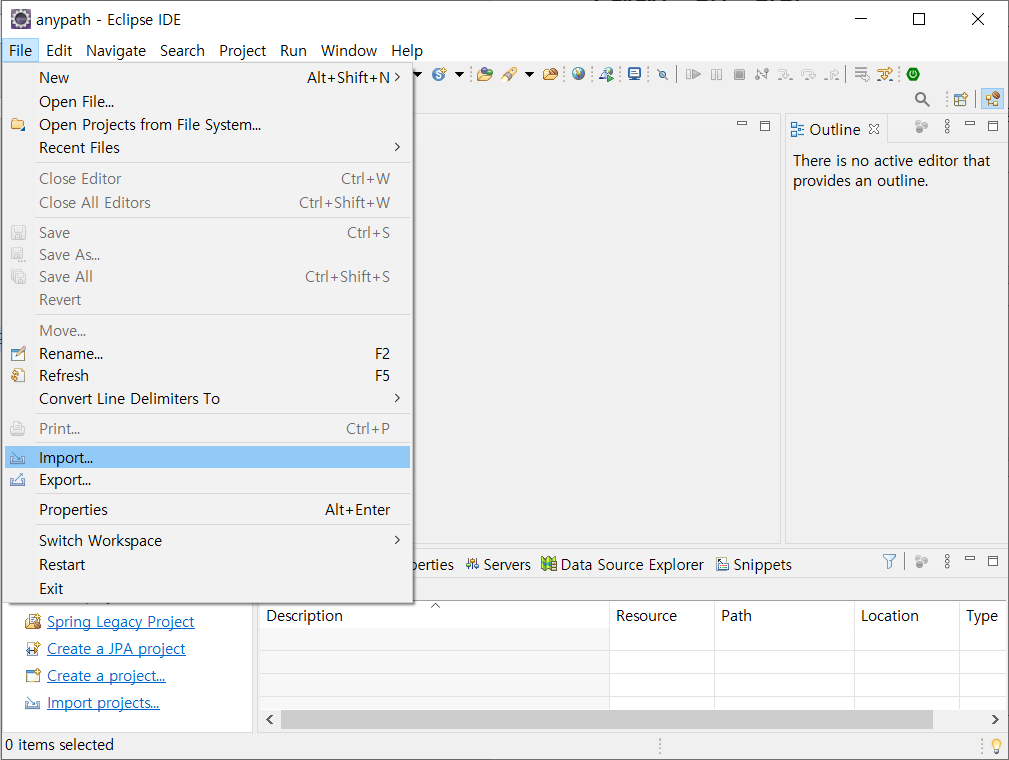
# 백엔드 빌드 방법

이클립스에 프로젝트 임포트

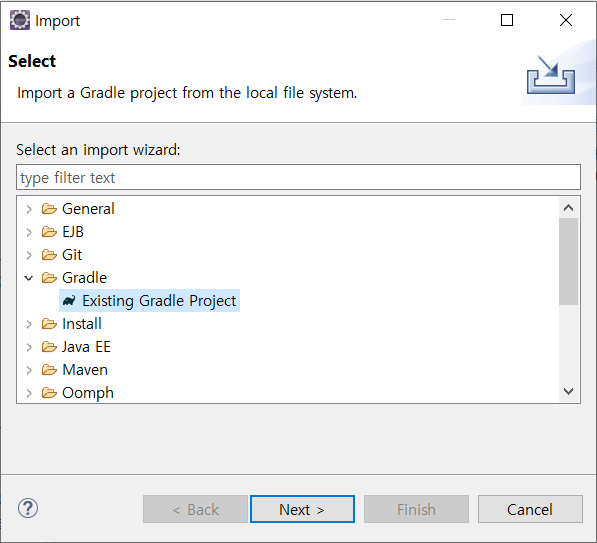
1. 이클립스 실행, Workspace 경로는 클론한 프로젝트 경로와 다르기만 하면 된다.



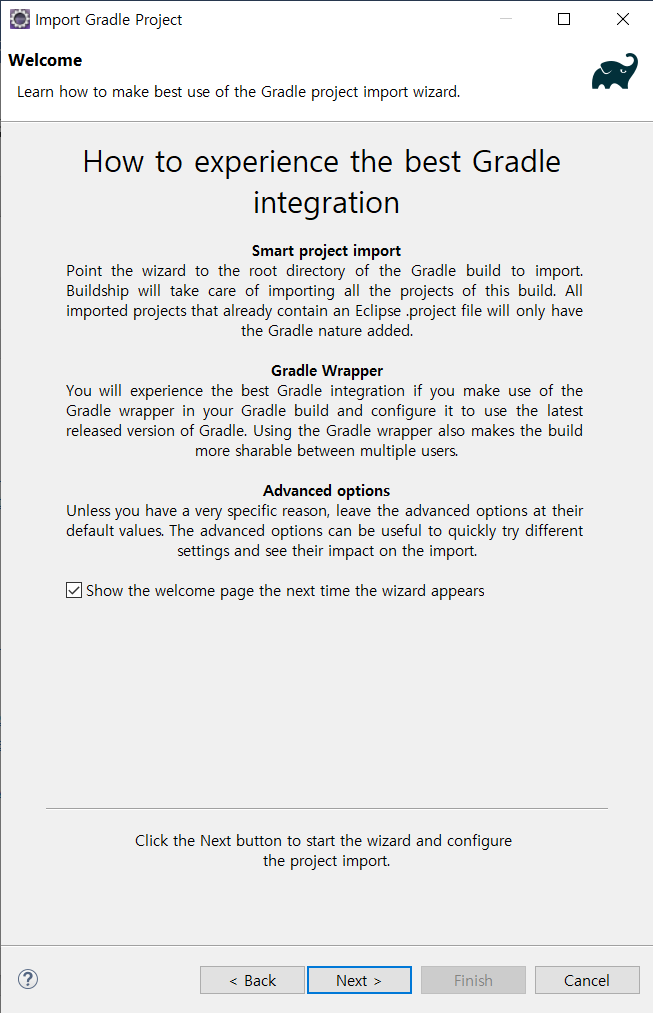
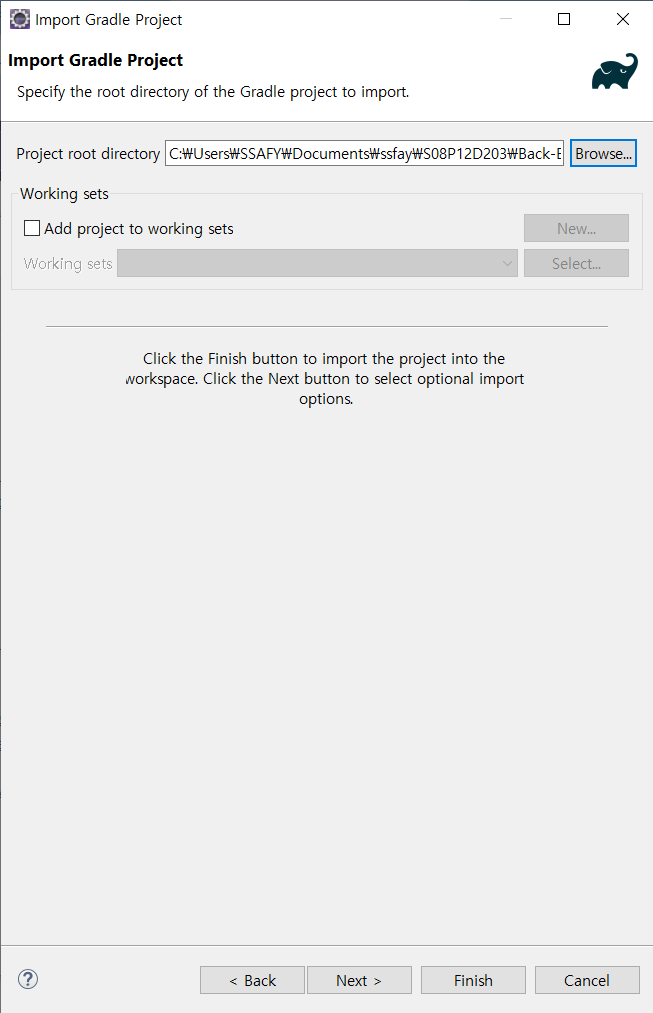
1. File > Import 로 진입



1. Existing Gradle Project 클릭

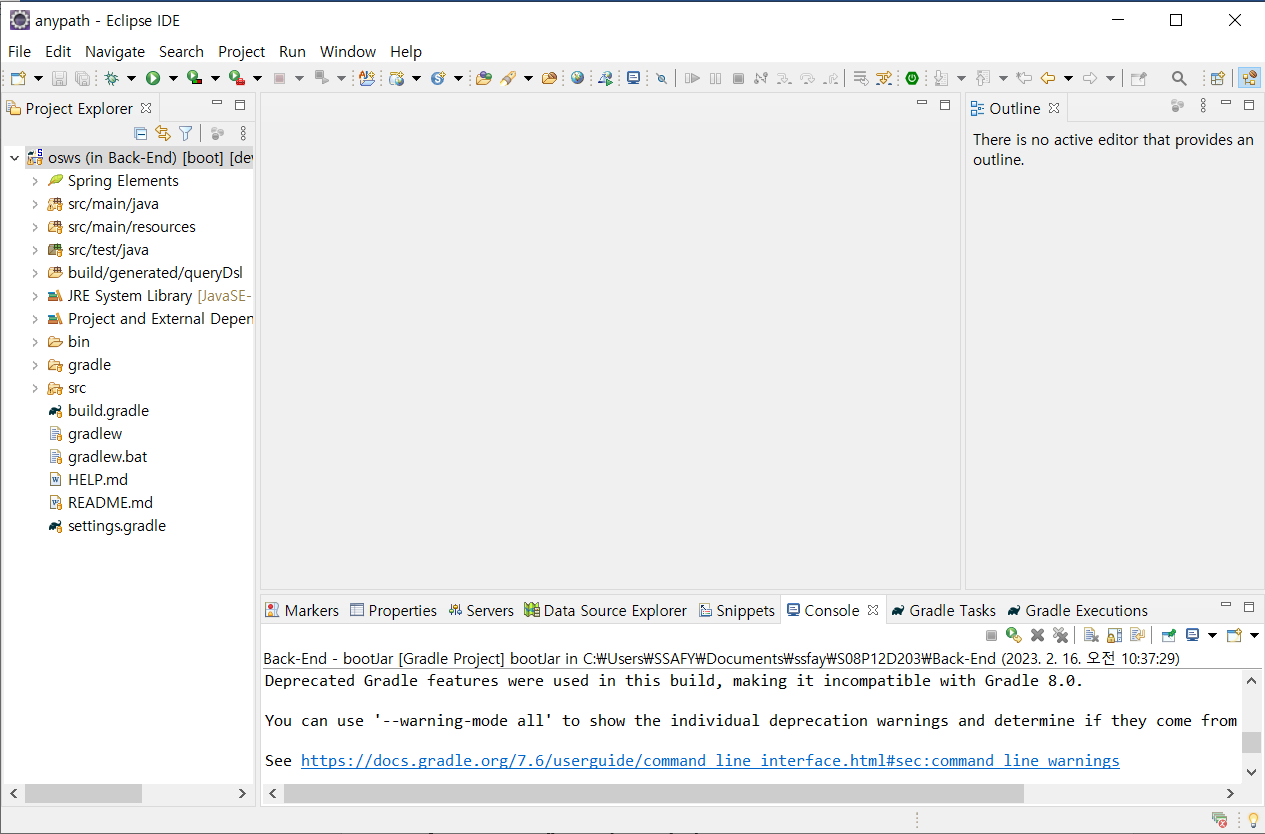


1. Next > Project Root Directory를 프로젝트가 있는 폴더\S08P12D203\Back-End로 지정후 Finish 클릭

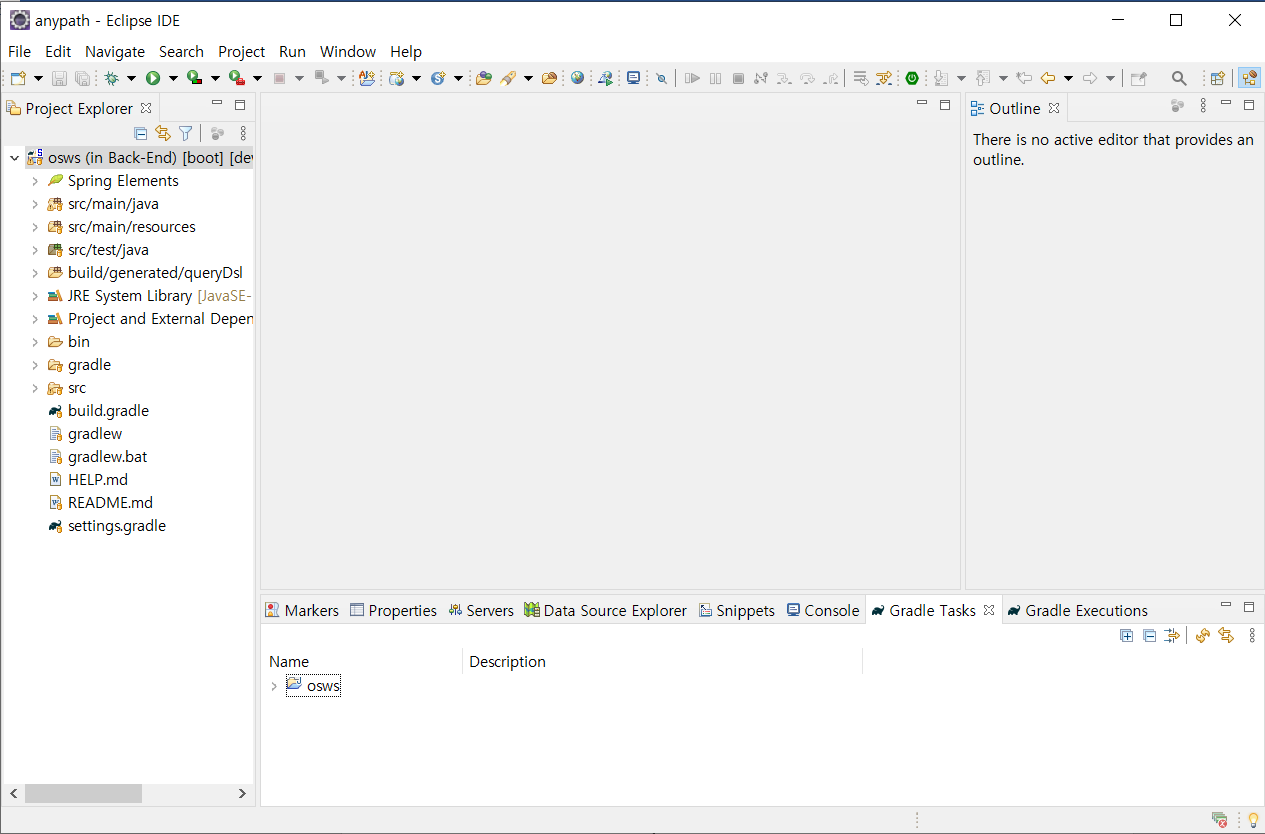
 → 

프로젝트 빌드

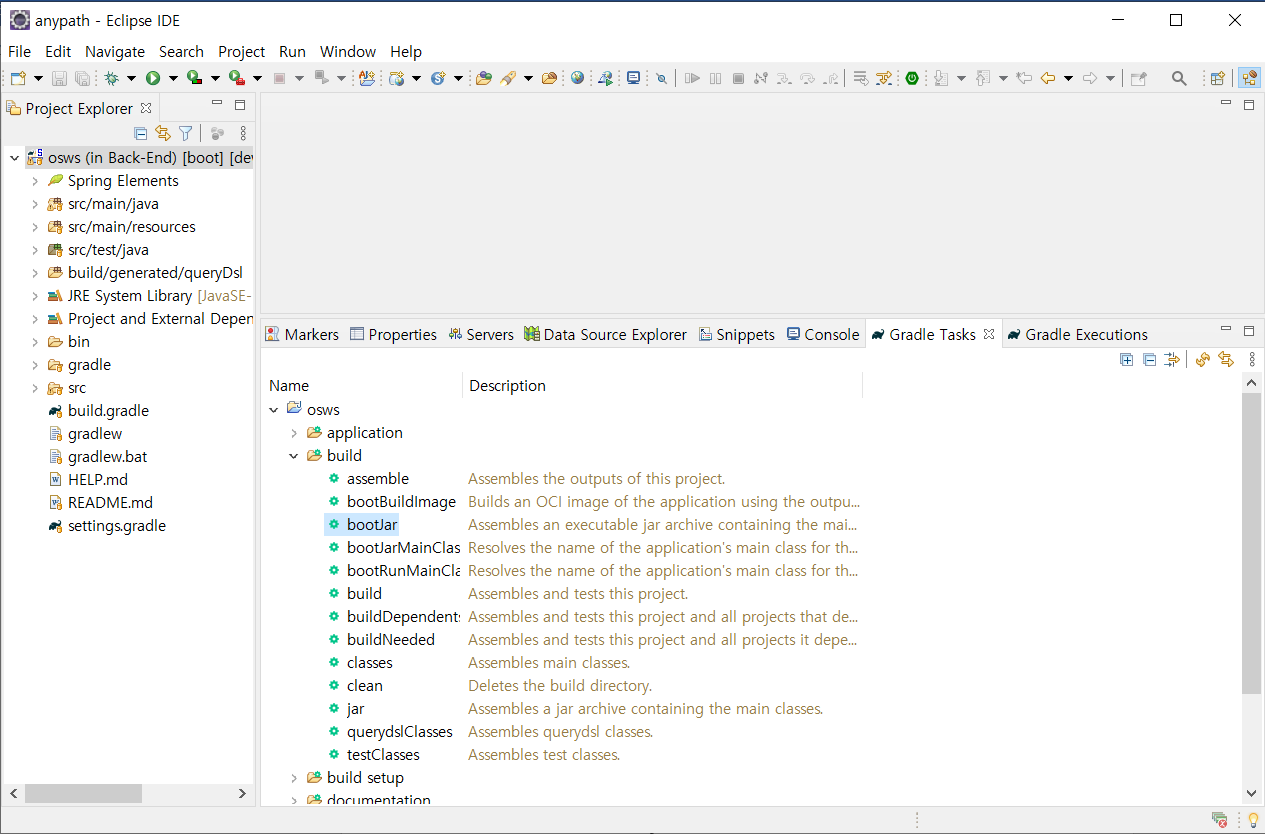
1. 임포트된 osws 프로젝트 클릭



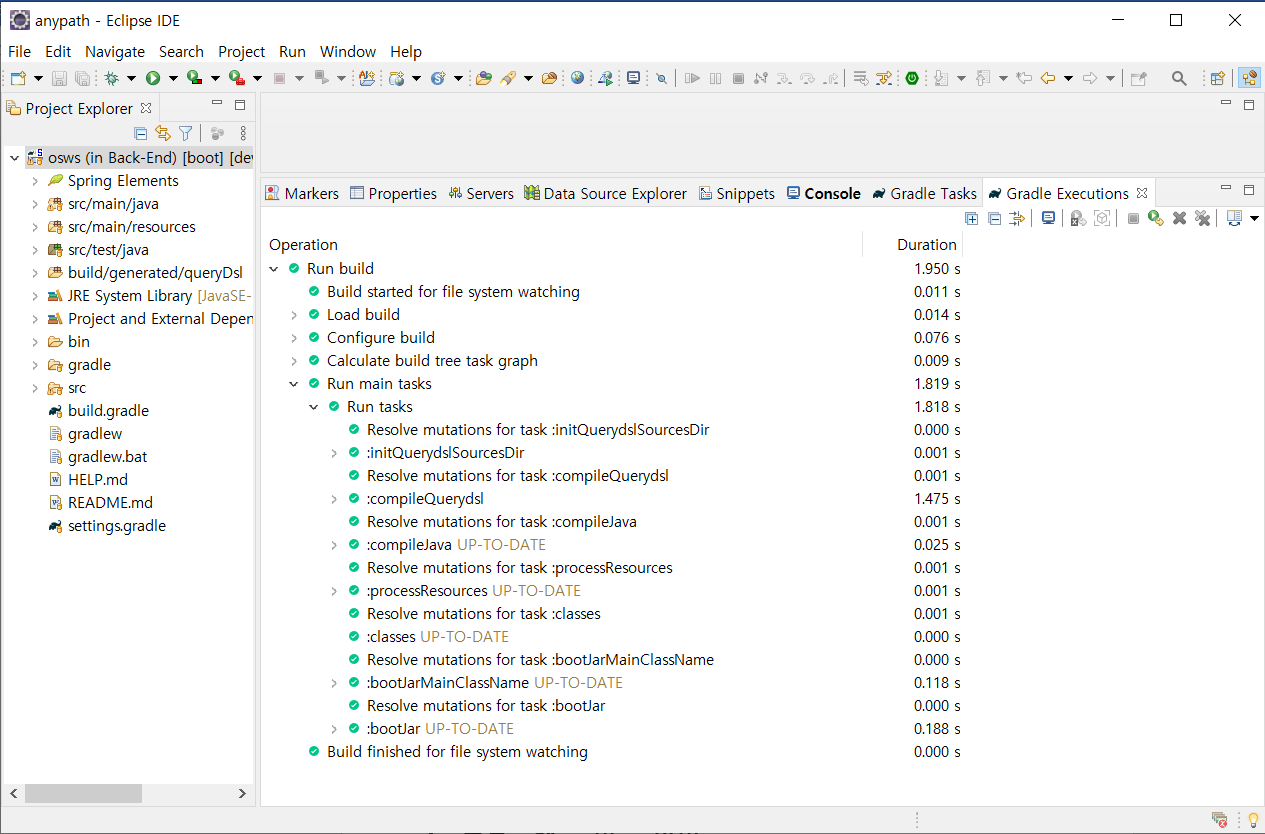
1. 하단에 GradleTasks 클릭



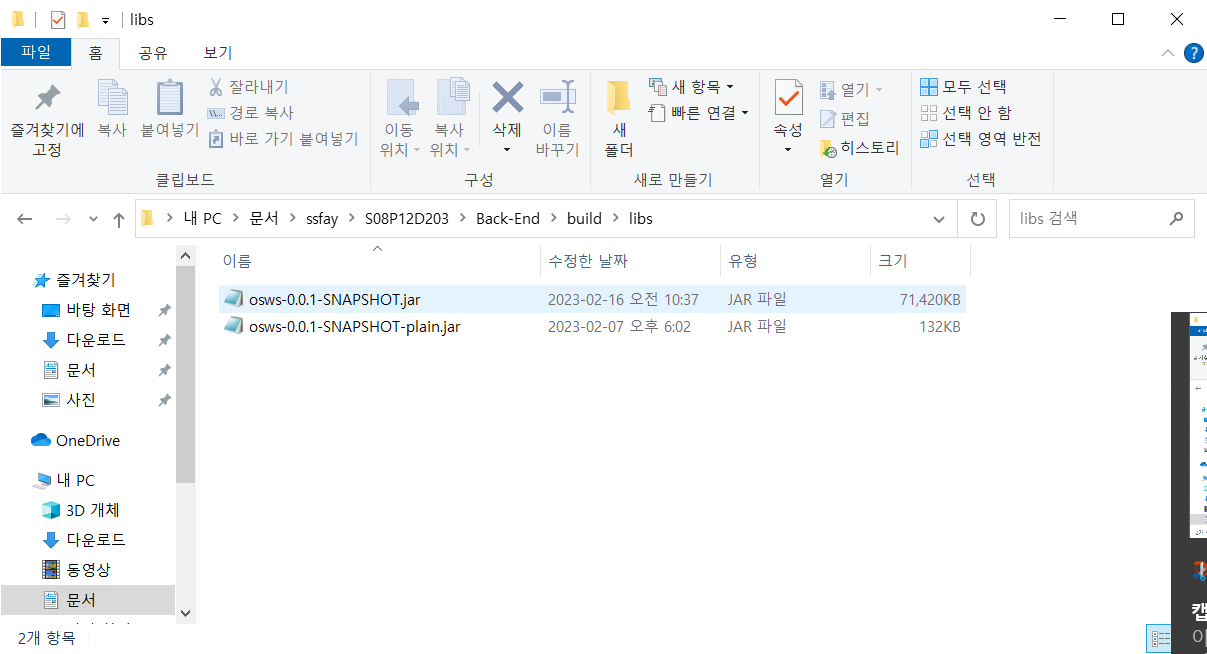
1. osws > build > bootJar 클릭



1. build가 완료되면 {클론한 경로}\S08P12D203\Back-End\build\libs 에서 .jar 파일을 확인할 수 있다.



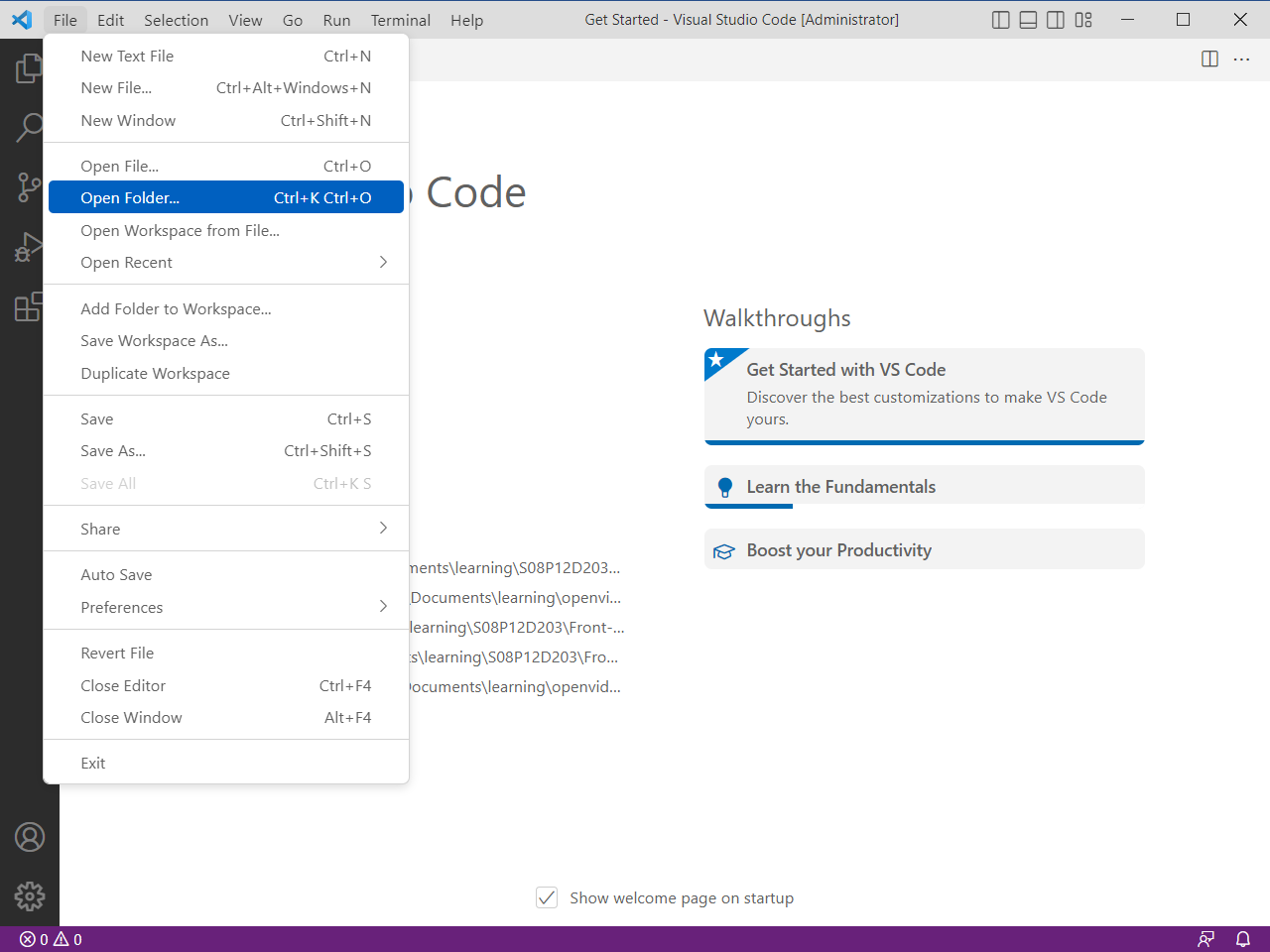
↓



# 프론트엔드 빌드 방법

프로젝트 열기

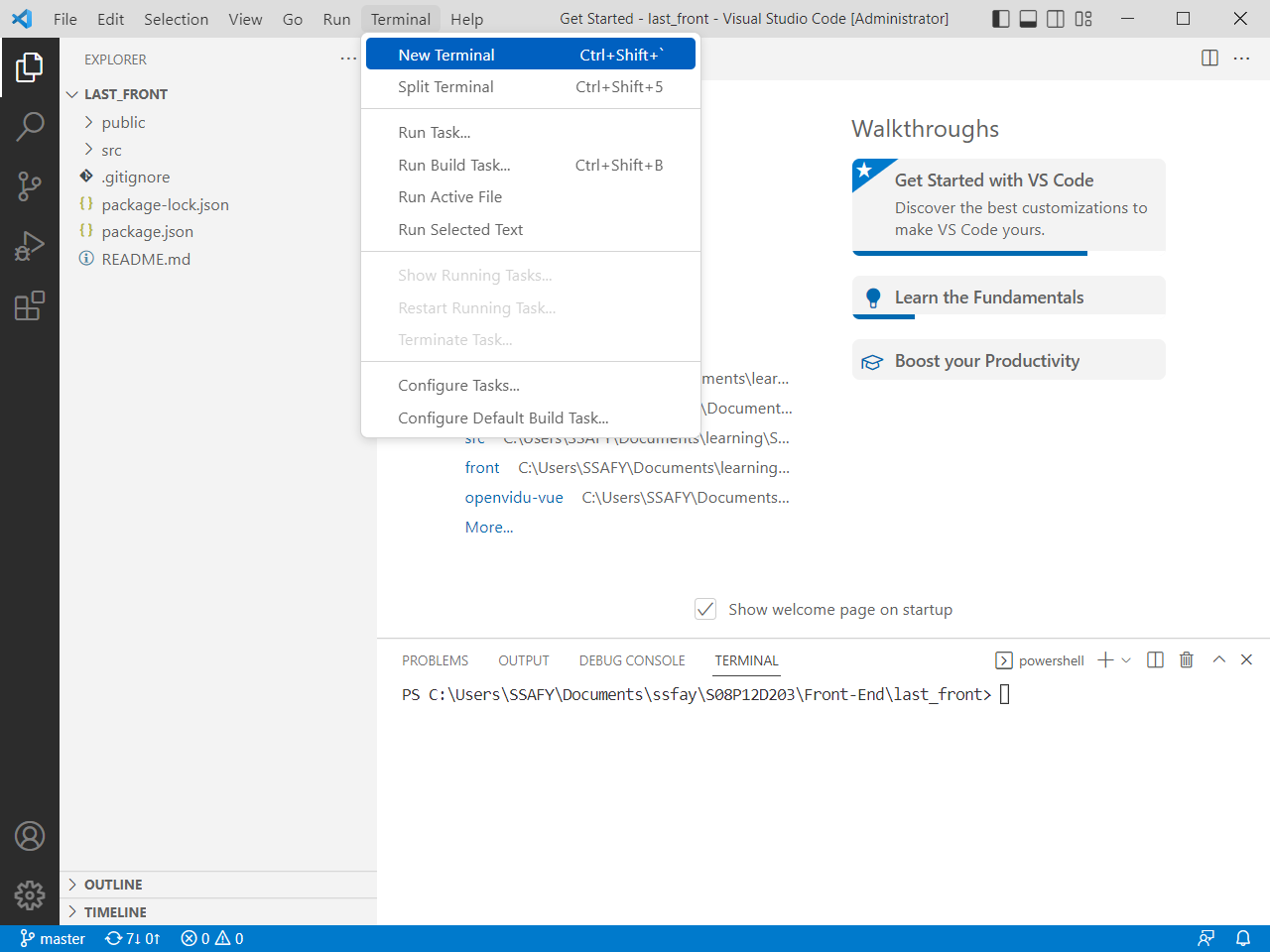
1. Visual Studio Code 실행 > File > Open Folder 클릭



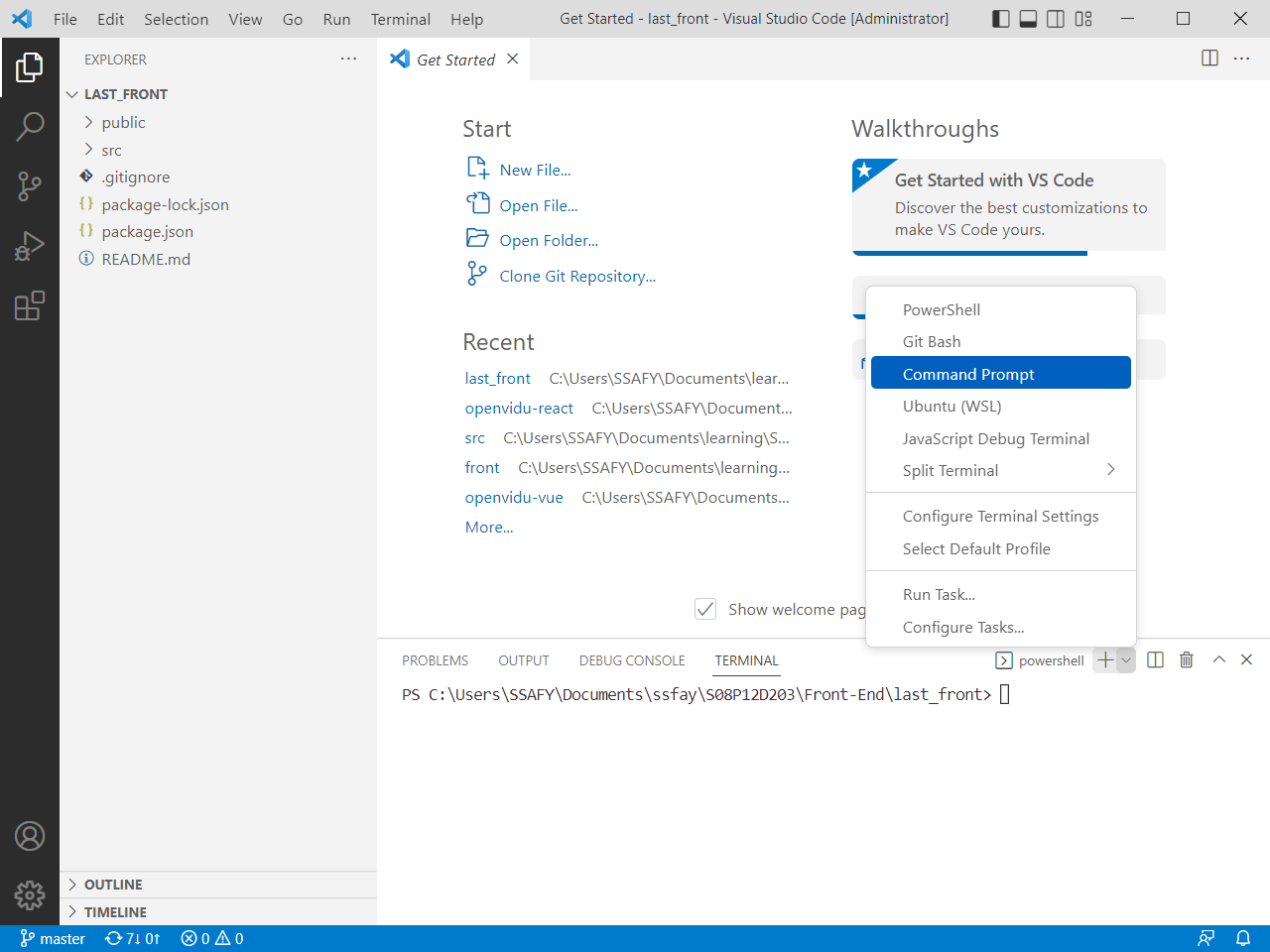
1. {클론한 경로}\S08P12D203\Front-End\last\_front 열기

프로젝트 빌드

1. Terminal > New Terminal 클릭



1. 하단 터미널창 우측 ⋁ 모양 클릭 후 Command Prompt 선택

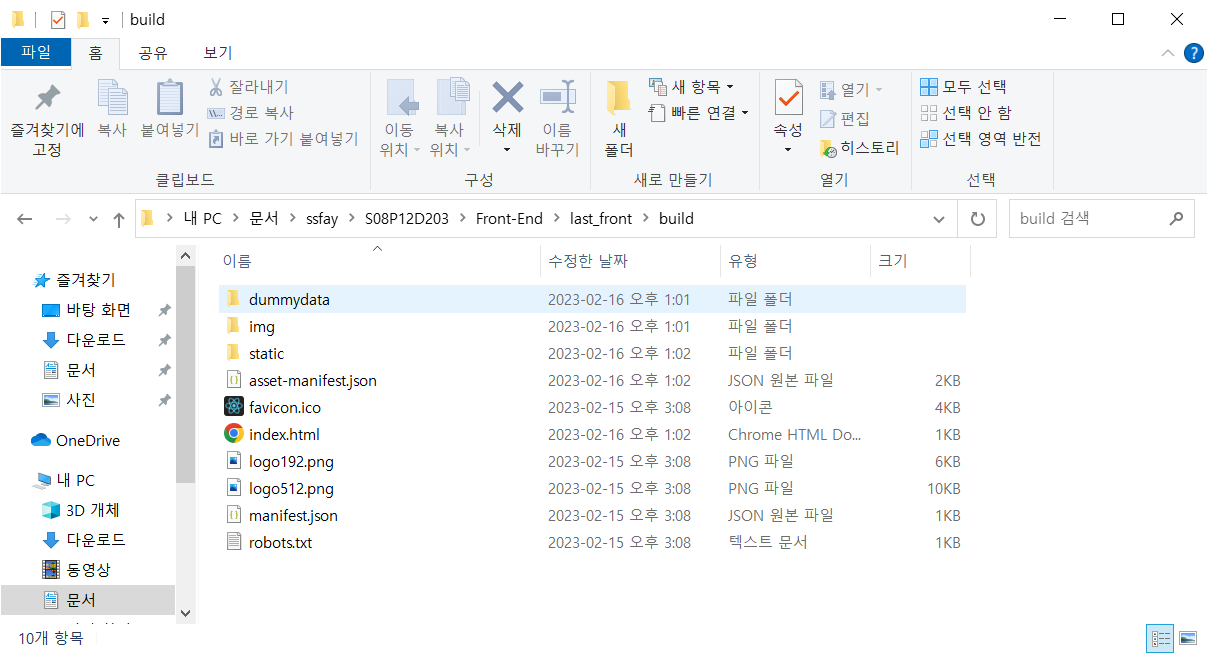


1. Terminal이 cmd 창으로 변경되면 아래 명령어를 순서대로 입력하면 된다.

$npm install

$npm run build

1. {클론한 경로} \S08P12D203\Front-End\last\_front\build 내부에서 build한 파일을 확인 할 수 있습니다.



# EC2 세팅

EC2에 필요한 도구 설치

EC2에 Docker를 설치한다.

$sudo apt-get update

$sudo apt-get install docker.io

개발 환경과 호환성을 위해 Node.Js 14.17.4 버전을 설치한다.

$sudo apt-get install curl

$curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash source ~/.bashrc

$nvm -v

$nvm install 14.17.4

Nginx를 설치한다.

$sudo apt-get install nginx

# 자동 빌드 및 배포 방법(with Jenkins)

Jenkins 설치 및 설정

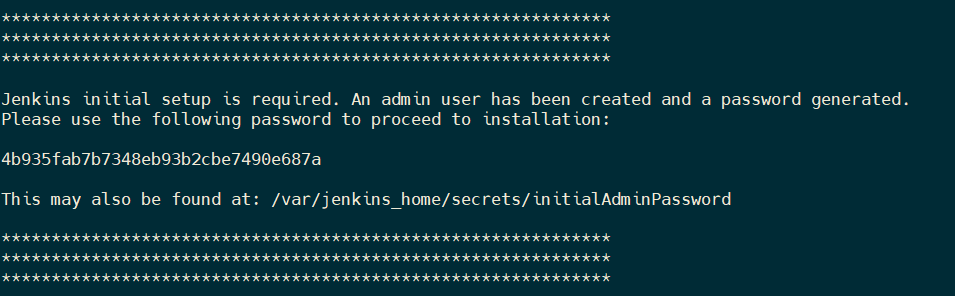
1. Jenkins를 Docker로 설치한다.

$sudo docker pull jenkins/jenkins:lts

$sudo docker run -d \ -p 9000:8080 \ -p 50000:50000 \ -v Jenkins volume:/var/jenkins\_home \ -v /var/run/docker.sock:/var/run/docker.sock:ro \ -v /var/lib/docker/containers:/var/lib/docker/containers:ro \ --net=bridge \ --name jenkins \ jenkins/jenkins:lts

1. Jenkins 접속 주소인 <http://i8d203.p.ssafy.io:9000>에 접속한다.
2. 아래 명령어를 통해 Jenkins container 로그를 확인하고 로그에 적혀있는 admin 암호를 복사한다.

$sudo docker logs jenkins



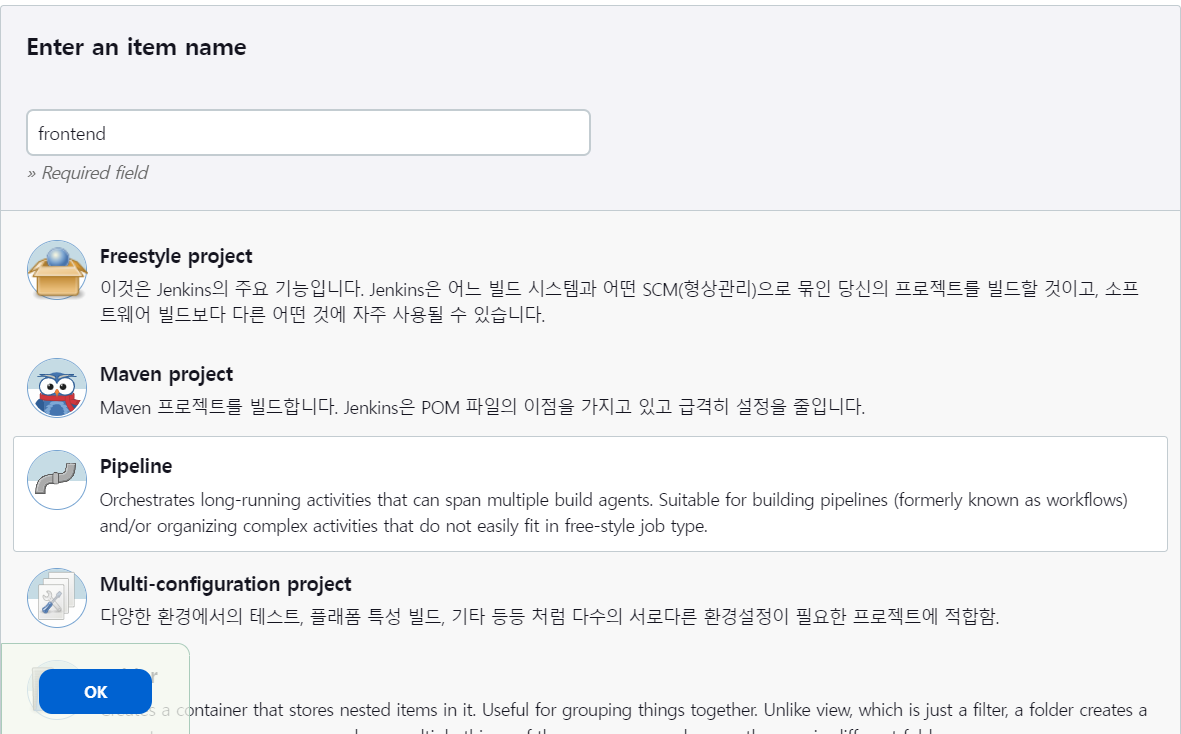
1. 암호 입력 후 다음 단계에서 Install suggested plugins 설치한다.



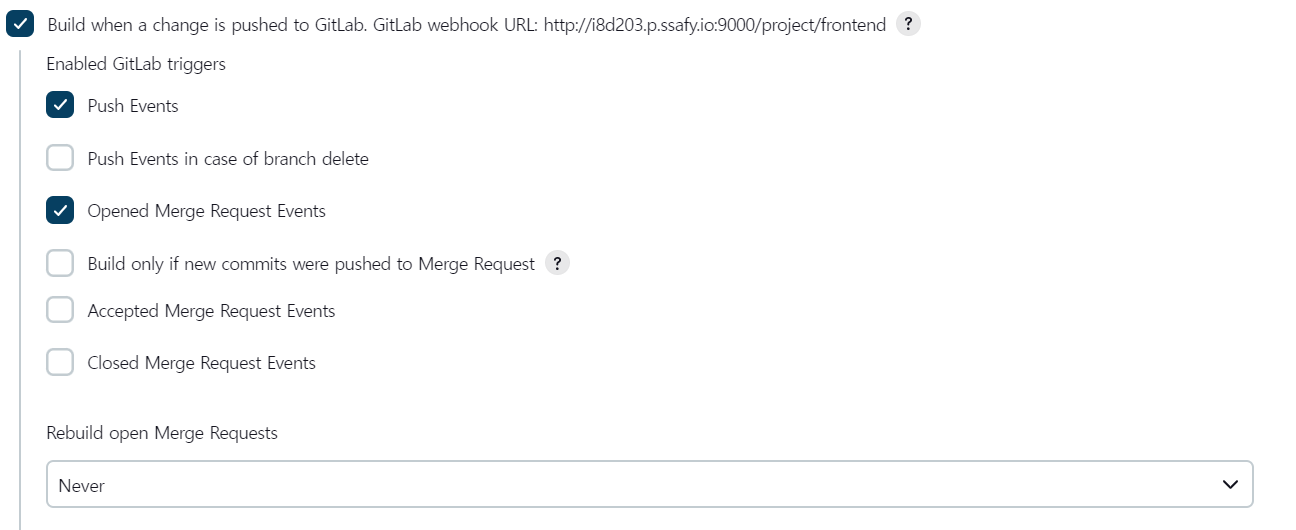
1. 이후 나오는 Dashboard 화면에서 Jenkins 관리 > Plugin Manager > Available plugins에서 아래 목록의 plugins를 설치한다.

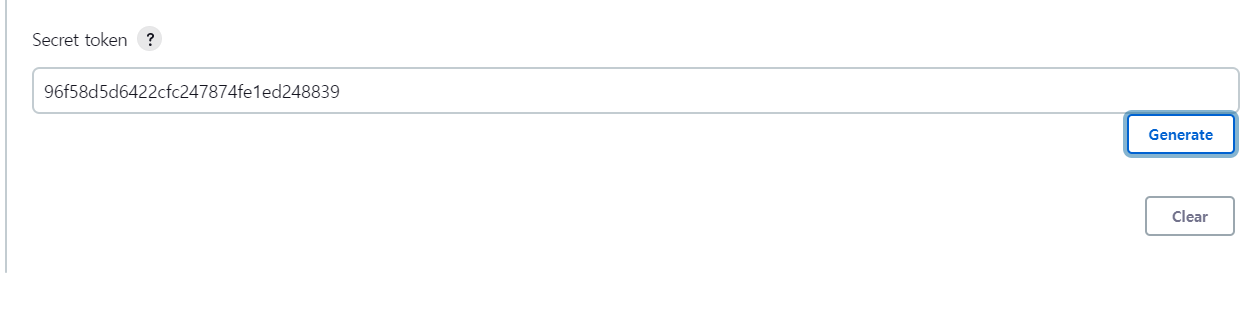
* Docker API Plugin
* Docker Commons Plugin
* Docker Pipeline
* Docker plugin
* Git plugin
* GitLab
* GitLab API Plugin
* GitLab Authentification plugin
* GitLab Branch Source Plugin
* Gradle Plugin
* NodeJS Plugin
* nvm-wrapper
* SCM API Plugin
* SSH Agent Plugin
* SSH Credentials Plugin
* SSH server

1. Jenkins dashboard에서 새로운 Item을 생성한다. (frontend 기준)

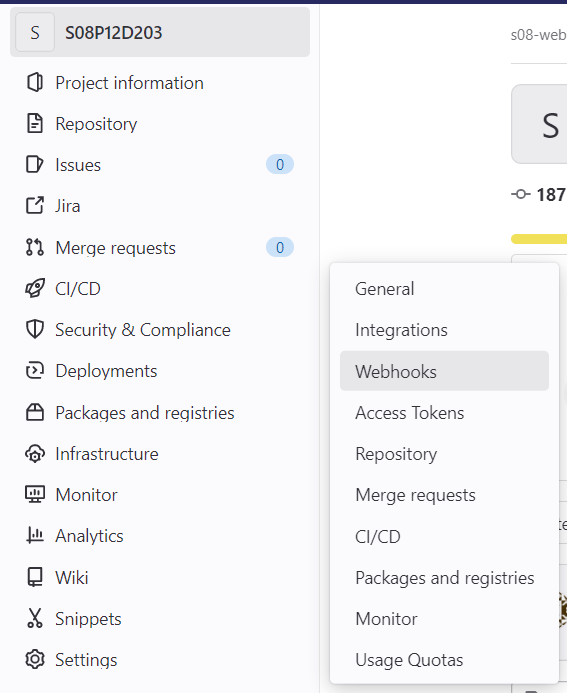


1. Build Triggers 에서 ‘Build when a change is pushed to Gitlab’ 을 체크하고, 우측에 나오는 webhook 주소와 하단의 Secret token을 다시 확인할 수 있는 곳에 저장한다.

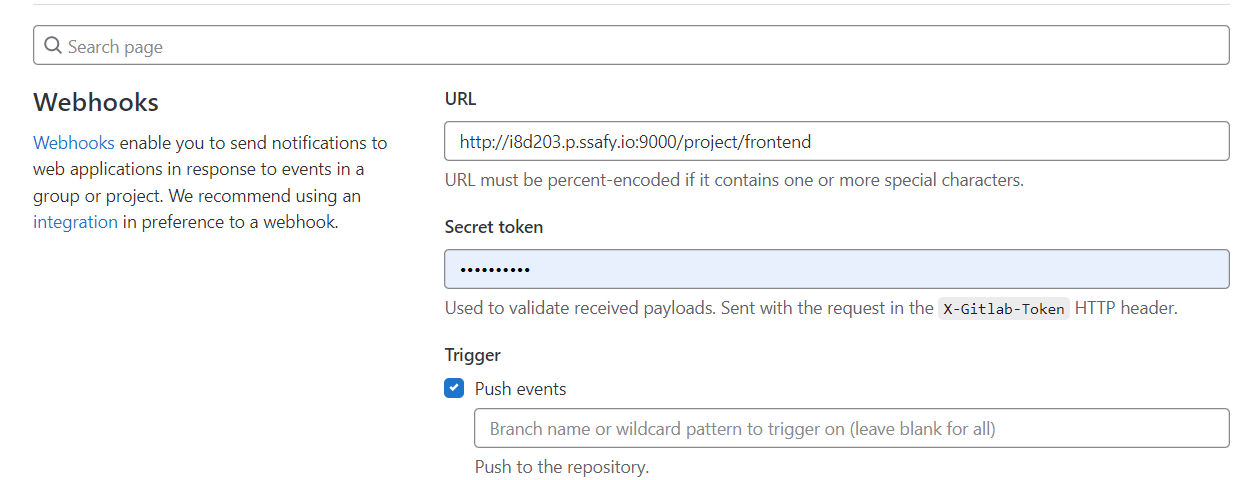




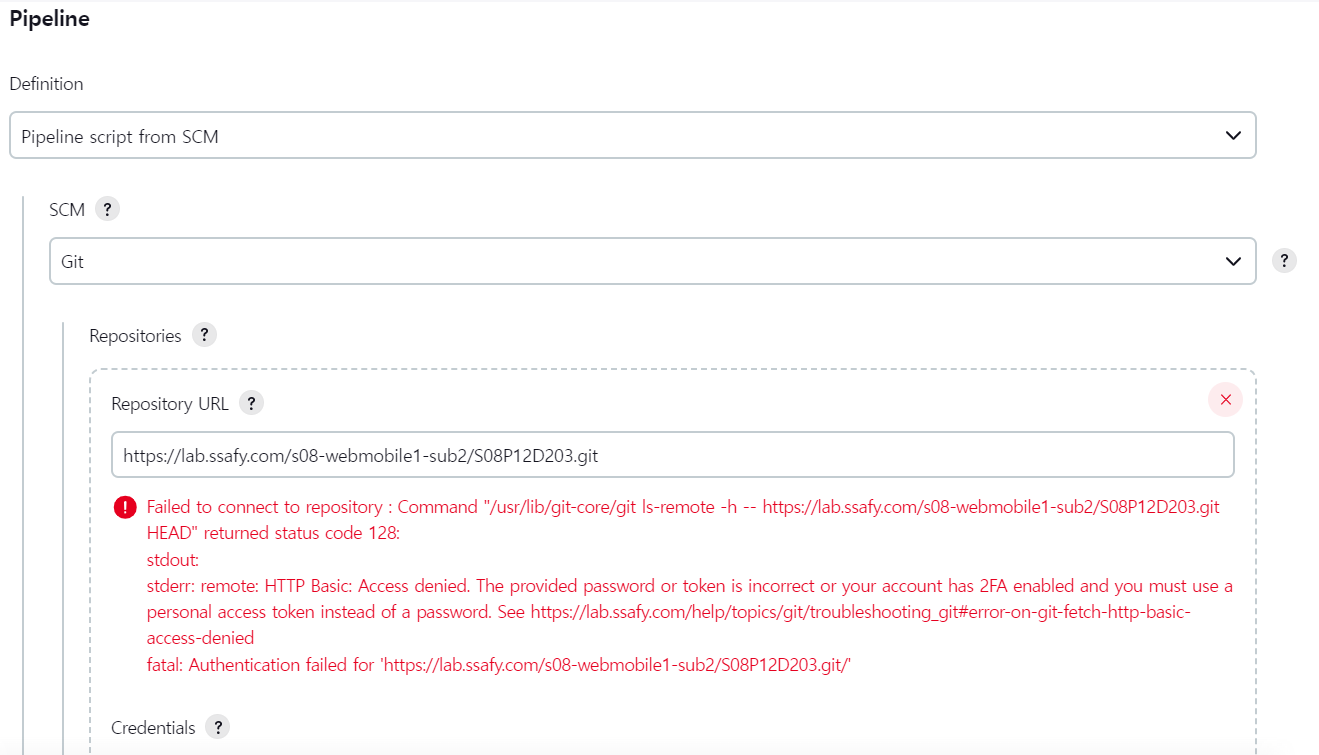
1. Gitlab에서 Settings > Webhook에 들어간다.



1. Webhook 설정 화면에서 10번에서 따왔던 webhook 주소와 secret token을 기입하고 Webhook을 추가한다.



1. Jenkins 화면으로 돌아가서, 하단의 Pipeline 메뉴에서 Definition에 ‘Pipeline script from SCM’을 선택 후, SCM에 ‘Git’을 선택하고, Repository URL에 프로젝트의 Gitlab 주소를 입력한다.



1. Credentials 아래의 Add 버튼을 통해 credentials을 추가한다. Kind에 ‘Username with password’를 선택하고, Username에 GitLab의 유저이름을, Password에 본인 계정의 비밀번호를 기입하고, ID에는 이 credential이 어떤 건지를 표시할 수 있도록 적어준다.



1. 만들어준 Credential을 선택한다.



1. Script Path에 작성한 Jenkinsfile 이름을 적어준다.



1. item을 저장한다.
2. Jenkinsfile에 필요한 credentials을 만든다. (Docker hub, EC2 ssh)
3. Jenkinsfile에 pipeline을 작성한다.

pipeline {

agent any

tools {nodejs "nodejs"}

stages {

stage('Gitlab') {

steps {

script {

git branch: 'master',

credentialsId: 'GitLab',

url: 'https://lab.ssafy.com/s08-webmobile1-sub2/S08P12D203.git'

}

}

}

stage('Build') {

steps {

dir('Front-End/last\_front') {

script {

sh "npm install"

sh "npm run build"

}

}

dir('Front-End') {

script {

dockerImage = docker.build "xronace/d203-frontend:latest"

}

}

}

}

stage('Docker hub push') {

steps {

script {

docker.withRegistry('', 'Docker-Hub') {

dockerImage.push("$BUILD\_NUMBER")

}

sh "docker rmi xronace/d203-frontend:$BUILD\_NUMBER"

}

}

}

stage('Deploy') {

steps {

sshagent(credentials: ['ec2-ssh']) {

sh 'ssh -o StrictHostKeyChecking=no ubuntu@i8d203.p.ssafy.io "docker stop d203-frontend"'

sh 'ssh -o StrictHostKeyChecking=no ubuntu@i8d203.p.ssafy.io "docker rm -f d203-frontend"'

sh 'ssh -o StrictHostKeyChecking=no ubuntu@i8d203.p.ssafy.io "docker run -dp 3000:80 --net=bridge --name d203-frontend xronace/d203-frontend:latest"'

sh 'docker image prune -f'

}

}

}

}

}

1. 9)부터 19)까지의 절차를 반복 수행하여 백엔드 부분을 담당할 item을 새로 생성한다.

프론트엔드, 백엔드 Docker 설정

pipeline 상으로 작동할 수 있도록, Git의 해당 경로에 프론트엔드 React를 빌드할 Dockerfile과 백엔드 Spring Boot를 빌드할 Dockerfile을 저장한다. 프론트엔드는 nginx container 기반으로 작성하기 때문에, 해당 nginx 설정파일인 nginx.conf 파일 역시 해당 경로에 맞추어 작성한다.

* 프론트엔드용 Dockerfile

FROM nginx

RUN mkdir /app

WORKDIR /app

RUN mkdir ./build

ADD ./last\_front/build ./build

RUN rm /etc/nginx/conf.d/default.conf

COPY ./nginx.conf /etc/nginx/conf.d/

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

* 백엔드용 Dockerfile

FROM openjdk:11

ARG JAR\_FILE=\*.jar

COPY ${JAR\_FILE} osws.jar

ENTRYPOINT ["java","-jar","/osws.jar"]

EXPOSE 8080

* 프론트엔드용 nginx.conf

server {

listen 80;

location / {

root /app/build;

index index.html;

try\_files $uri $uri/ /index.html;

}

}

MySQL Docker 설정

1. EC2 내에서 MySQL docker image를 빌드할 Dockerfile을 생성한다.

FROM mysql:5.7

ADD ./mysql-init-files /docker-entrypoint-initdb.d

ENV MYSQL\_USER [유저 이름]

ENV MYSQL\_PASSWORD [유저 비밀번호]

ENV MYSQL\_ROOT\_PASSWORD [루트 비밀번호]

ENV MYSQL\_DATABASE [DB 이름]

EXPOSE 3306

1. Dockerfile에 써둔 것 처럼, mysql-init-files 라는 폴더를 해당 경로에 맞게 생성한 후 내부에 image를 빌드할 때 DB에 필요한 sql 파일을 작성해서 저장해 둔다.
2. MySQL용 Docker image를 만든다.

$sudo docker build -t xronace/d203-mysql:latest .

1. MySQL docker container를 생성 및 실행한다.

$sudo docker run -dp 3306:3306 -v mysql-volume --net=bridge --name d203-mysql xronace/d203-mysql:latest

OpenVidu CE설치 및 실행

1. EC2 내부에서 OpenVidu CE를 다운로드 받는다. 이때 저장 경로는 /opt 를 추천한다. root 권한이 필요하다.

$sudo su

$cd /opt

$curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install\_openvidu\_latest.sh | bash

1. 다운로드 받은 경로에 들어가서 .env 파일을 편집한다.

$cd {다운로드 받은 경로}

$vi .env

1. .env 파일에 아래 설정을 수정한다. HTTP\_PROT와 HTTPS\_PORT는 주석처리 되어 있으니 잘 확인해야 한다.

10. DOMAIN\_OR\_PUBLIC\_IP=i8d203.p.ssafy.io

13. OPENVIDU\_SECRET={스프링 application.yml 내에 OPENVIDU\_SECRET 값}

24. CERTIFICATE\_TYPE=letsencrypt

27. LETSENCRYPT\_EMAIL={유효한 이메일}

36. HTTP\_PORT=8442

40. HTTPS\_PORT=8443

1. , openvidu 폴더에서 openvidu 관련docker들을 다음 명령어로 실행한다.

$./openvidu start

nginx 설정

1. 설치한 nginx를 임시로 중지한다.

$sudo systemctl stop nginx

1. Let’s Encrypt를 설치한다.

$sudo apt-get install letsencrypt

1. 무료 인증서를 발급받는다.

$sudo letsencrypt certonly --standalone -d i8d203.p.ssafy.io

1. 사용할 d203\_nginx.conf 환경설정 파일을 /etc/nginx/sites-available에 작성한다. 아래 두 명령어를 실행하면 수정할 수 있다.

$cd /etc/nginx/sites-available

$sudo vi d203\_nginx.conf

* d203\_nginx.conf 파일

server {

location /{

proxy\_pass http://localhost:3000;

}

location /api{

proxy\_pass http://localhost:5000/api;

}

listen 443 ssl;

ssl\_certificate /etc/letsencrypt/live/i8d203.p.ssafy.io/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/i8d203.p.ssafy.io/privkey.pem;

# include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot

# ssl\_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {

if ($host = i8d203.p.ssafy.io) {

return 301 https://$host$request\_uri;

}

listen 80;

server\_name i8d203.p.ssafy.io;

return 404;

}

1. 적용을 위해 해당 파일을 링크로 연동한다.

$sudo ln -s /etc/nginx/sites-available/d203\_nginx.conf /etc/nginx/sites-enabled/d203\_nginx.conf

1. nginx를 테스트 후, 성공적이면 nginx 서버를 재시작한다.

$sudo nginx –t

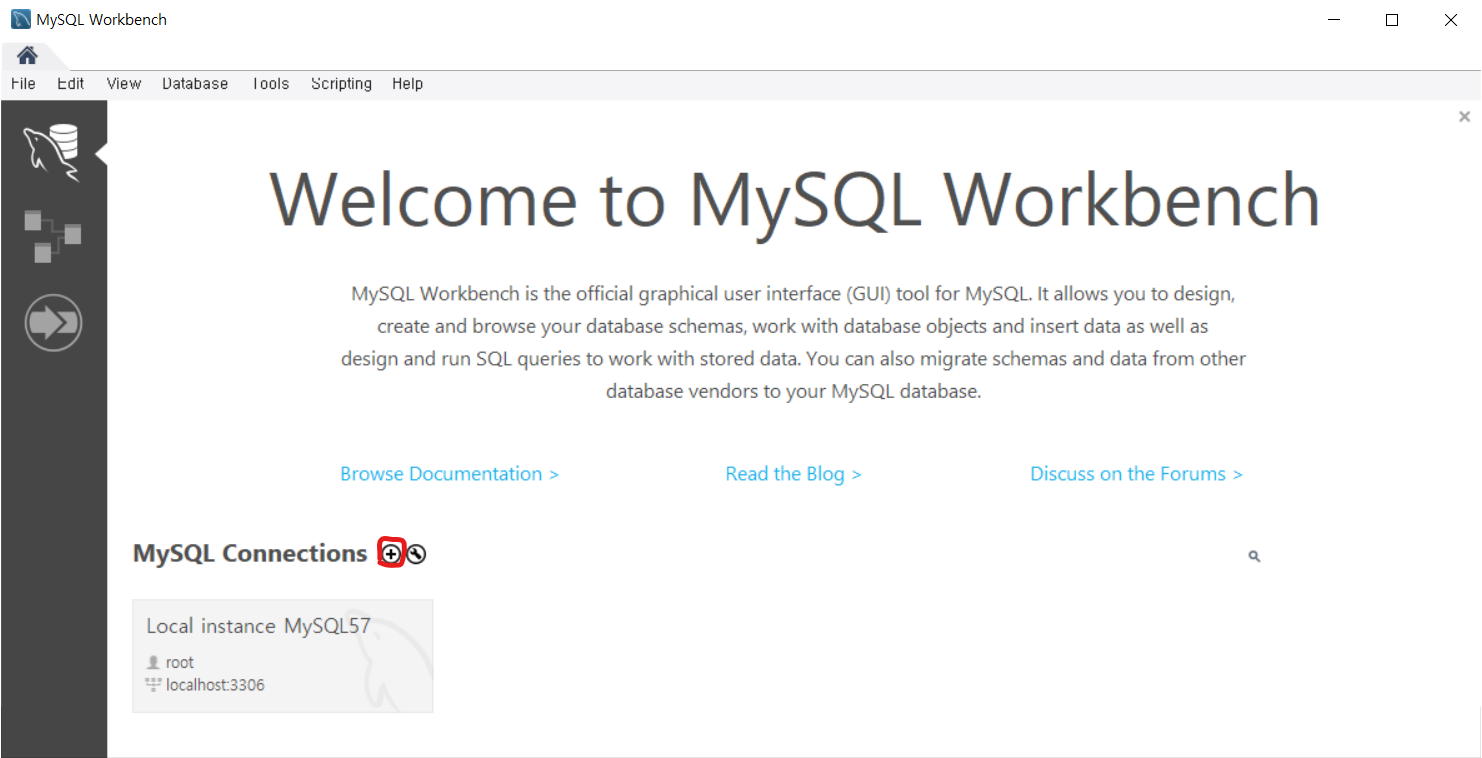
$sudo systemctl restart nginx

결론

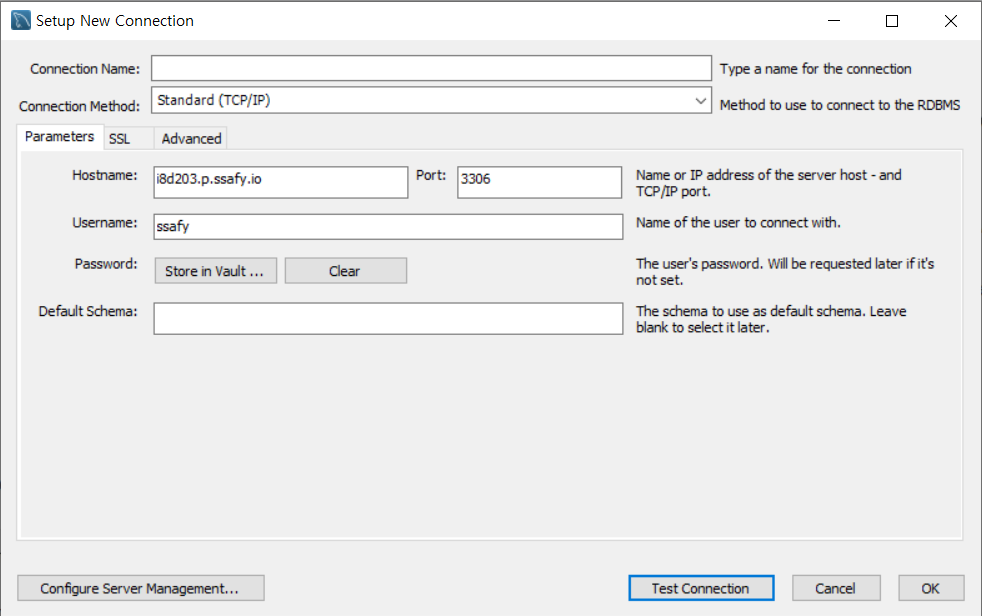
위 과정을 모두 거치면 Gitlab의 master 브랜치에 push가 발생할 때 마다 자동으로 빌드 및 배포가 진행된다.

# MySQL 워크벤치 사용법

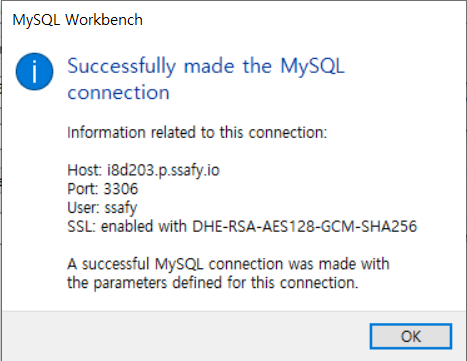
1. 워크벤치 실행 후 나온 메인 화면에서 MySQL Connections 우측 ⊕ 버튼을 클릭한다.



1. 아래 사진과 같이 정보를 입력하고 Test Connection을 실시합니다. 비어있는 Connection Name 창에는 원하는 이름을 입력하시면 됩니다. (password는 [여기로](https://meeting.ssafy.com/s08p10d2/pl/ipngscab6tb4dbdjrtnjh3k3dr) 들어가 확인하면 됩니다.)



1. Test Connection 결과가 아래 사진처럼 나오면 성공적으로 DB에 접속할 수 있습니다.



1. 메인 페이지에서 추가된 데이터베이스 계정을 확인할 수 있습니다.

