



B302 : HOOPAA

삼성 청년 SW 아카데미 대전캠퍼스 7기
공통 프로젝트 (2022.07.11 ~ 2022.08.19)

포팅 매뉴얼

주혜령(팀장), 김현주, 노정현, 박정현, 최원재

목차

1. 프로젝트 기술 스택.....	2
2. CI/CD.....	3
3. 빌드 상세내용	7
4. 배포 특이사항	11
5. DB 계정	12
6. 프로퍼티 정의	14
7. 외부 서비스.....	16

온라인으로 즐기는 완벽한 토론. 코로나 19 시대가 도래하며 대부분의 수업들이 온오프라인을 반복하는 가운데 대한민국 교육에서 빠지지 않는 토론 수업은 온라인으로 진행하기 까다로운 형태의 수업 중 하나입니다.

HOOPAA는 이러한 문제를 해결하고자 개발되었습니다. 이제 토론 맞춤 UI와 사회자의 컨트롤로 원활한 토론을 진행할 수 있습니다. 토론 왕 선정과 다양한 애니메이션 효과는 토론에 재미를 느껴 더 집중할 수 있게 하죠. 많은 사람들과 다양한 주제로 토론하고 토론 왕이 되어 보세요. 당신의 도전을 응원합니다!

1. 프로젝트 기술 스택

가. 이슈 관리 : Jira

나. 형상 관리 : GitLab

다. 커뮤니케이션 : Mattermost, Notion

라. 개발 환경

1) OS : Window 10

2) IDE

A. IntelliJ IDEA Community Edition 2022.1.3

B. Visual Studio Code 1.70.0

C. UI/UX : Adobe Photoshop, Figma

3) Database : MySQL Workbench 8.0 CE

4) Server : AWS EC2 (MobaXterm)

A. Ubuntu 20.04.4 LTS

B. Docker 20.10.17

C. Jenkins 2.346.2 LTS

D. Nginx 1.18.0

마. 상세 내용

1) Backend

A. Java (Open JDK (Zulu 8.33.0.1))

B. Spring Boot Gradle 6.7

C. Lombok 1.18.20, OpenVidu 2.22.0, Swagger3.0.0, QueryDSL, JPA

2) Frontend

A. HTML5, CSS, JavaScript (ES6)

B. Vue 3.0.0, Vuex 4.0.2

C. Node.js 16.16.0


2. CI/CD

Jenkins 이미지를 이용해 Jenkins 를 도커상에서 이용합니다.

1. Git 레포지토리를 받아 Jenkins 폴더로 이동
2. docker-compose up -d 명령어를 사용하여 Jenkins 를 실행
3. GitLab 과 Docker 에 관한 초기 Jenkins 세팅 완료 후 새로운 item 을 Freestyle project 로 생성

Enter an item name

» *Required field*

 **Freestyle project**

이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

4. 프로젝트의 구성에서 소스코드 관리 -> Git 설정에서 Git 레포지토리에 관한 설정을 진행

Git ?

Repositories ?

Repository URL ?

https://lab.ssafy.com/s07-webmobile1-sub2/S07P12B302/

Credentials ?

jhno96@naver.com/*****

+ Add

Name ?

Refspec ?

Add Repository

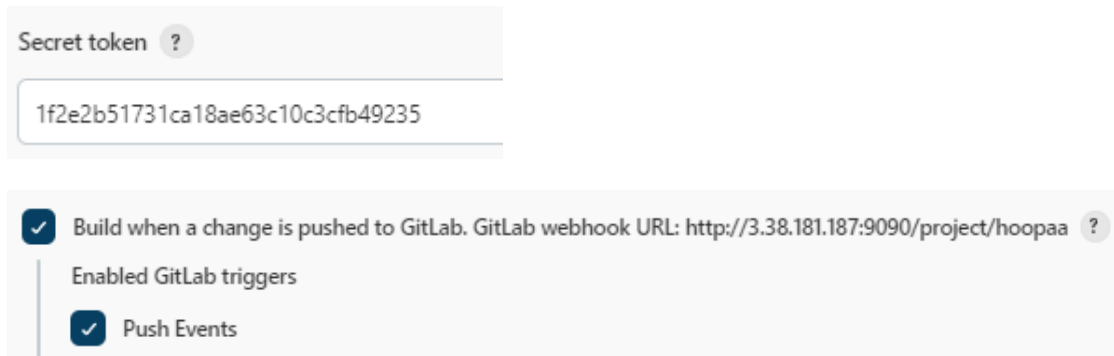
Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

5. 다음 빌드 유발 항목에서 Build when a change is pushed to GitLab 항목을 체크하고 다음과 같이 설정 후 GitLab 의 Webhook 토큰을 입력



Secret token ?

1f2e2b51731ca18ae63c10c3cfb49235

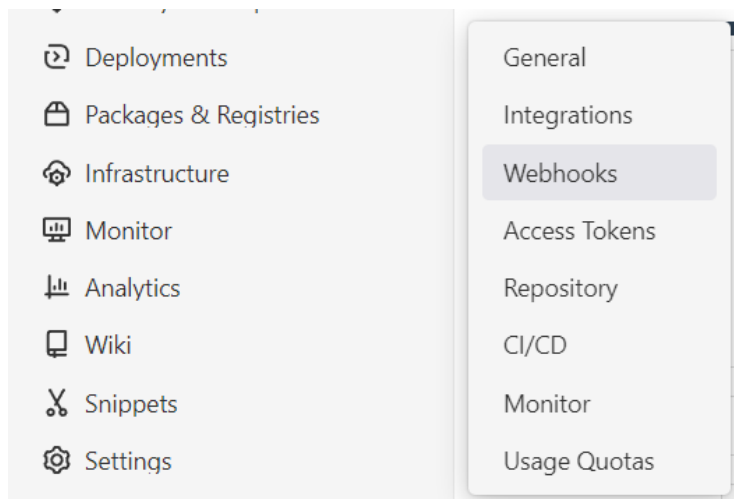
☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://3.38.181.187:9090/project/hoopaa ?

Enabled GitLab triggers

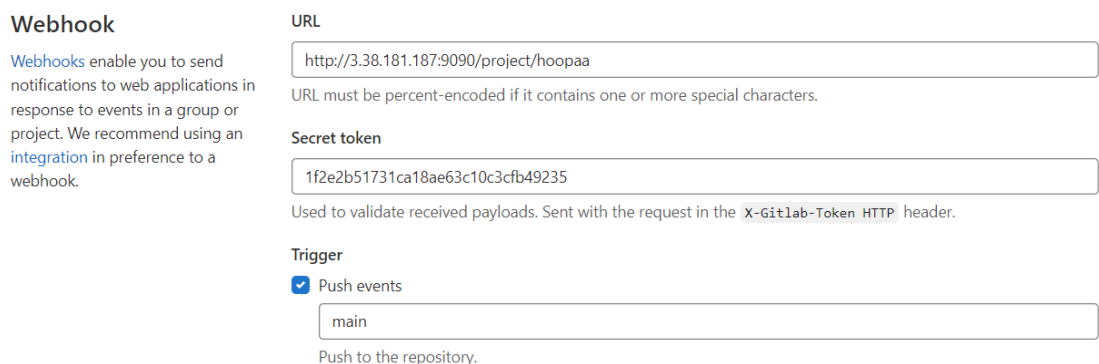
☒ Push Events

A. GitLab Webhook 설정

- i. GitLab setting 탭의 Webhooks 를 클릭



- ii. Jenkins 의 주소와 토큰, 트리거 등을 입력



Webhook

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

http://3.38.181.187:9090/project/hoopaa

URL must be percent-encoded if it contains one or more special characters.

Secret token

1f2e2b51731ca18ae63c10c3cfb49235

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger

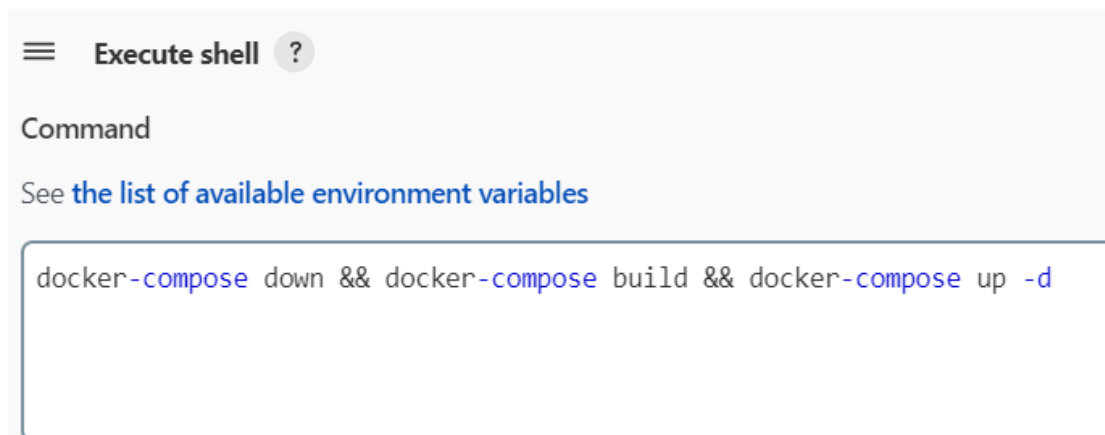
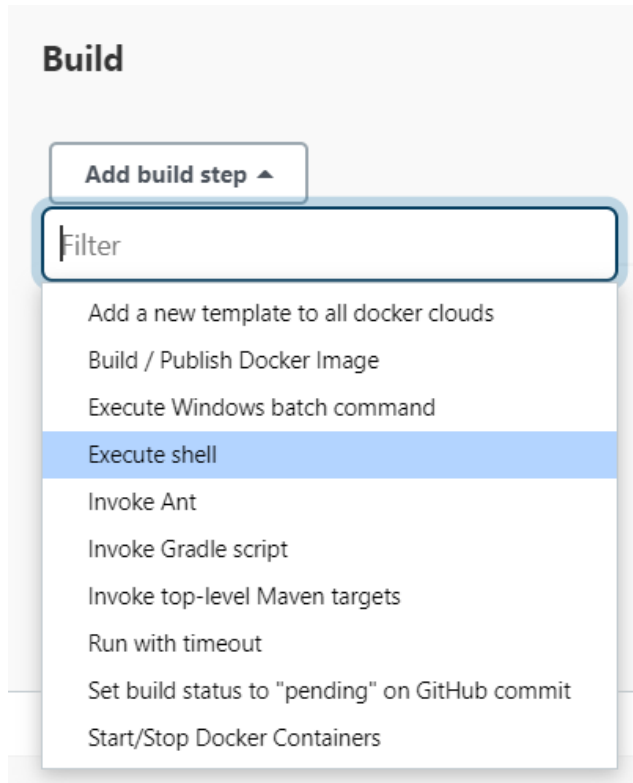
☒ Push events

main

Push to the repository.

- iii. 완료 후 test 로 정상 작동하는지 확인

6. Build 항목의 Execute shell 로 다음 명령어를 입력



7. 저장으로 현재 설정 값을 저장

3. 빌드 상세내용

Dockerfile 과 docker-compose 를 이용해 빌드 합니다.

1) ./ docker-compose.yml

```
version: "3.8"

services:
  db:
    build: DB
    container_name: db
    restart: always
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: ssafy
      TZ: "Asia/Seoul"
    networks:
      - my_network
  spring:
    build: backend
    container_name: spring
    restart: always
    volumes:
      - ./backend/app:/app/app
    expose:
      - "8080"
    environment:
      TZ: "Asia/Seoul"
      SPRING_DATASOURCE_URL: "jdbc:mysql://db:3306/debate?useUnicode=true&characterEncoding=utf8"
    depends_on:
      - db
    networks:
      - my_network
  nginx:
    build: frontend
    container_name: nginx
    ...|
```


service : 컨테이너를 실행하기 위해 정의, 컨테이너는 곧 서비스라는 개념으로 접근

image : 컨테이너에 사용될 이미지

build : Dockerfile의 경로 지정, docker-compose.yml로 부터 상대적인 위치

container_name : 컨테이너 이름을 지정

restart : **no**(수동으로 재시작), **always**(컨테이너를 수동으로 끄기전까지 항상 재시작), **on-failure**(오류가 있을 시에 재시작)

ports : 호스트와 컨테이너 포트를 **설정** (호스트OS와 컨테이너의 포트를 바인딩) Dockerfile에서 **expose**와 동일

depends_on : services 간 종속을 설정합니다, 종속성 순서대로 서비스 시작

environment : 환경 변수를 설정

env_file : 환경 변수를 모아둔 파일

logging : options: max-size: 로그 파일당 최대 용량

command : 해당 서비스가 올라올 때 Dockerfile의 **CMD** 명령문을 무시하고 실행할 명령어 설정

entrypoint : Dockerfile의 entrypoint보다 docker-compose의 entrypoint가 우선 순위가 **높음** (command/entrypoint 둘이 비슷)

2) ./ backend / Dockerfile

```
# Build Stage
FROM openjdk:8-jdk-alpine as build-stage                                # openjdk:8-jdk-alpine 이미지를 build-stage라고 지칭

COPY gradlew .                                                         # gradlew을 이미지에 복사
COPY gradle gradle                                                      # gradle을 이미지에 복사
COPY build.gradle .                                                     # build.gradle을 이미지에 복사
COPY settings.gradle .                                                  # settings.gradle을 이미지에 복사
COPY src src                                                            # 웹 어플리케이션 소스를 이미지에 복사
RUN chmod +x ./gradlew                                                  # gradlew 실행할 수 있는 권한 부여(없으면 Permission Denied 발생)
RUN ./gradlew bootJar                                                   # gradlew를 사용해서 실행 가능한 jar 파일 생성

# Deploy Stage
FROM openjdk:8-jdk-alpine                                              # 새로 생성할 이미지의 기반 이미지 지정

WORKDIR /app                                                           #/app 경로로 작업 디렉토리 전환
COPY --from=build-stage build/libs/*.jar app.jar                      #build-stage 이미지에서 build/libs/*.jar 파일을 app.jar로 복사

EXPOSE 8080                                                            #컨테이너 포트 지정
ENTRYPOINT [ "java", "-jar", "/app/app.jar" ]                        # 컨테이너 생성 후 최초 실행될 때 명령어 지정
```

3) ./ frontend / Dockerfile

```
FROM node:lts-alpine as build-stage                # node:lts-alpine 이미지를 build-stage라고 지칭

WORKDIR /app                                       #/app 경로로 작업 디렉토리 전환

COPY package*.json ./                             # package*.json를 이미지에 복사

RUN npm install --save --legacy-peer-deps         # npm install --save --legacy-peer-deps 실행

COPY . .                                           # 전부 복사

RUN npm run build                                  # npm run build 실행

FROM nginx:stable-alpine as production-stage       #nginx:stable-alpine 이미지를 production-stage로 지칭
COPY --from=build-stage /app/dist /usr/share/nginx/html #build-stage의 /app/dist 파일을 /usr/share/nginx/html에 복사

EXPOSE 80                                          # 포트 지정
CMD ["nginx", "-g", "daemon off;"]               # 컨테이너 실행되면 nginx실행 global 디렉티브로 daemon off; 옵션 적용
```

4) ./ DB / Dockerfile

```
FROM mysql:5.7.35                                  #mysql:5.7.35를 base 이미지로 지정
COPY HooPaa.sql /docker-entrypoint-initdb.d/init.sql #/docker-entrypoint-initdb.d/init.sql에 HooPaa.sql 복사 (시작하면 구동되는 초기화 sql)
EXPOSE 3306                                         # 포트 3306 지정
CMD ["mysqld"]                                     # 컨테이너를 생성하여 최초로 mysqld 실행
```

5) ./ jenkins / docker-compose.yml

```
version: '3.8'

services:
  jenkins:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: 'jenkins_docker'
    restart: always
    user: root
    ports:
      - '9090:8080'
      - '50000:50000'
    volumes:
      - './jenkins_home:/var/jenkins_home'
      - '/var/run/docker.sock:/var/run/docker.sock'

# 컨테이너를 실행하기 위해 정의, 컨테이너는 곧 서비스라는 개념으로 접근
# Dockerfile의 경로 지정, docker-compose.yml로 부터 상대적인 위치
# 컨테이너 이름을 지정
# no: 수동, always: 컨테이너를 수동으로 끄기전까지, on-failure: 오류가 있을 시 재시작
# 호스트와 컨테이너 포트를 설정, Dockerfile에서 expose와 동일
# 마운트(mount)가 필요한 호스트의 경로와 컨테이너의 경로를 명시
```

6) ./jenkins/ Dockerfile

```
FROM jenkins/jenkins:lts                                     # jenkins/jenkins:lts base이미지로

USER root                                                     # 사용자 지정

# docker 설치
RUN apt-get update && \
  apt-get -y install apt-transport-https \
  ca-certificates \
  curl \
  gnupg2 \
  zip \
  unzip \
  software-properties-common && \
  curl -fsSL https://download.docker.com/linux
/$(. /etc/os-release; echo "$ID")/gpg >
/tmp/dkey; apt-key add /tmp/dkey && \
  add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
  $(lsb_release -cs) \
  stable" && \
  apt-get update && \
  apt-get -y install docker-ce                                # 우분투 패키지 업데이트
                                                           # apt-transport-https 설치 : 패키지 관리자가 https를 통해 데이터 및 패키지에 접근
                                                           # ca-certificates 설치 : SSL 기반 앱이 SSL 연결이 되어있는지 확인
                                                           # curl 설치 : 웹사이트에서 데이터를 다운로드 받을 때 사용
                                                           # gnupg2 설치 : 통신상에서 혹은 데이터를 저장할 때 보안
                                                           # zip 설치 : 압축
                                                           # unzip 설치 : 압축 풀기
                                                           # software-properties-common : 개인 패키지 저장소(PPA)를 추가하거나 제거

# docker-compose 설치
RUN curl -L
  "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)"
-o /usr/local/bin/docker-compose && \
  chmod +x /usr/local/bin/docker-compose && \
  ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

# add-apt-repository 명령어로 아래 3라인 주소의 repository 추가
# 우분투 패키지 업데이트
# docker community edition 설치
```

4. 배포 특이사항

다음 명령어로 Docker-compose 를 이용한 자동 배포를 진행합니다.

1. GitLab Clone

```
git clone https://lab.ssafy.com/s07-webmobile1-sub2/S07P12B302.git
```

2. 다음 명령으로 실행 권한 부여 및 실행으로 인증서 발급

```
chmod +x init-letsencrypt.sh  
sudo ./init-letsencrypt.sh
```

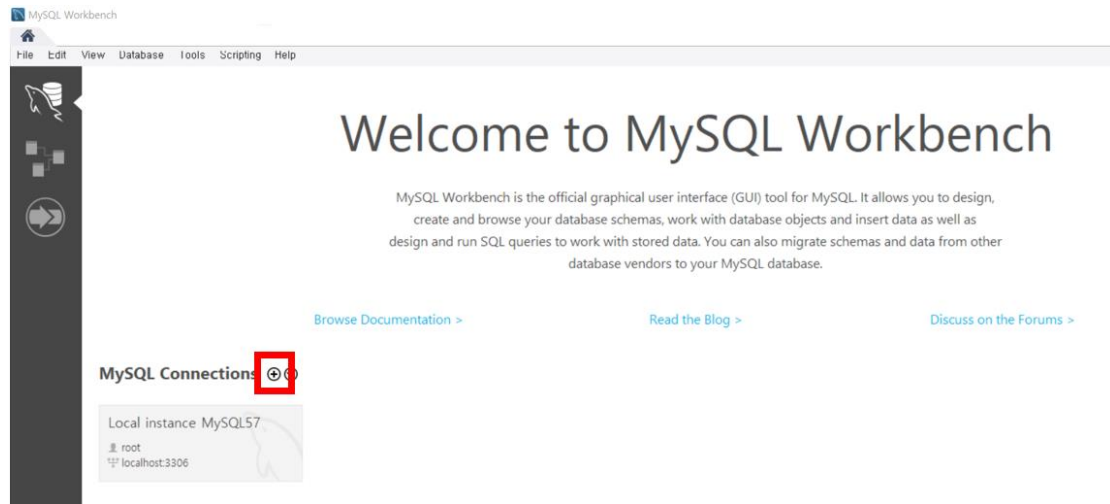
3. 다음 명령으로 도커 컨테이너 실행
(OpenVidu Server, Nginx, Spring, DB, Coturn Server, KMS, Certbot)

```
docker-compose up -d
```

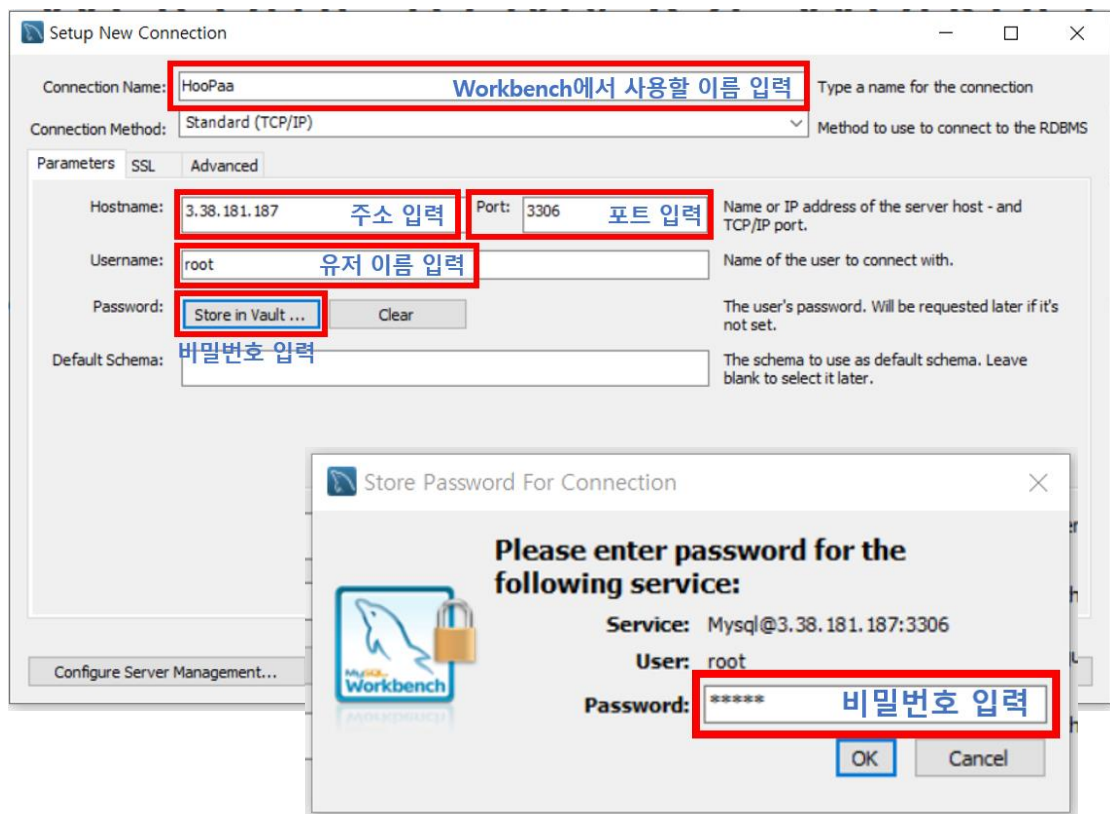
4. 배포가 정상적으로 진행된 후 서버에서 확인 (<https://hoopaa.site/>)

5. DB 계정

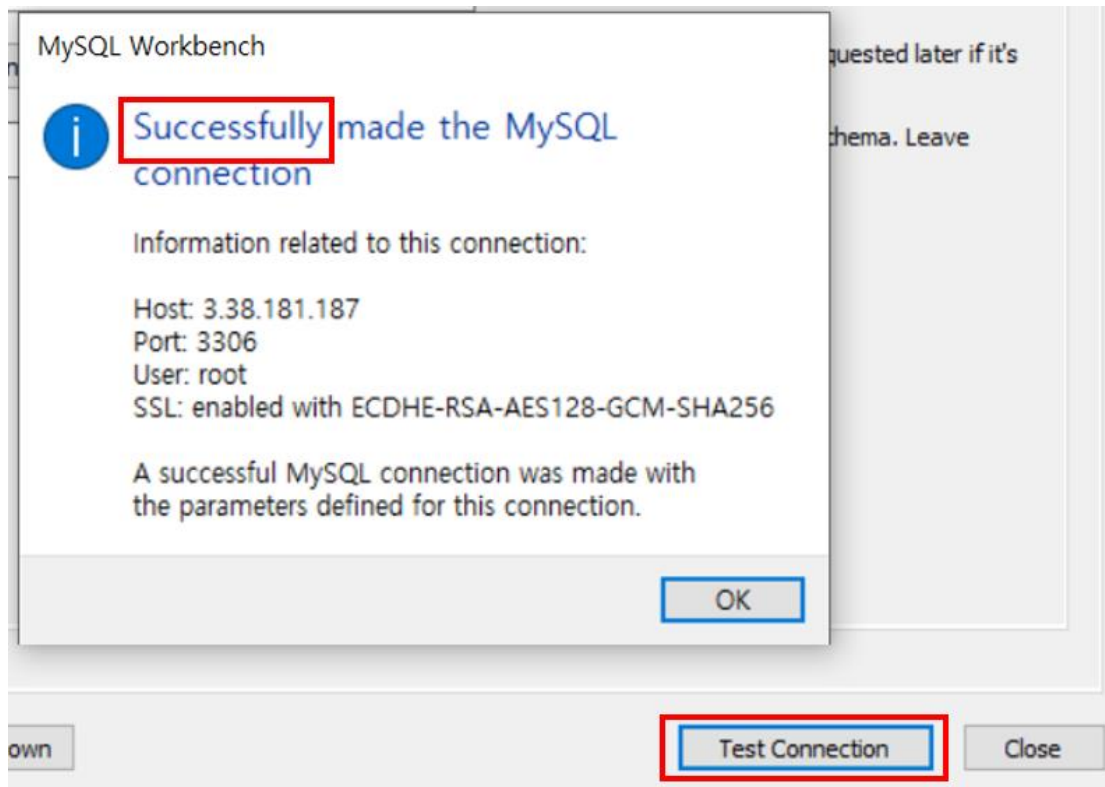
가. MySQL Workbench 추가



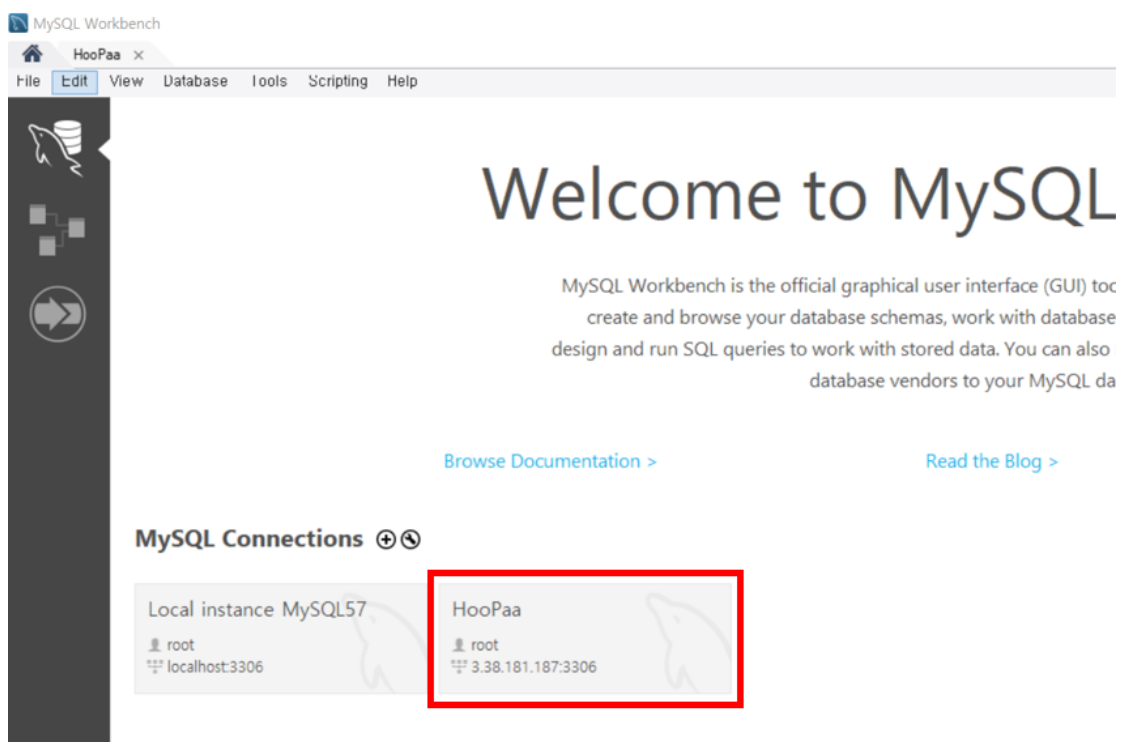
나. EC2 계정 정보 작성



다. 테스트 커넥션 확인



라. 추가된 커넥션 확인



6. 프로퍼티 정의

Nginx 세팅

```
...
server {
    listen 80;

    server_name hoopaa.site, 3.38.181.187, i7b302.p.ssafy.io;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name hoopaa.site;
    server_tokens off;

    # SSL Config
    ssl_certificate /etc/letsencrypt/live/hoopaa.site/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/hoopaa.site/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Proxy
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Proto https;
    proxy_headers_hash_bucket_size 512;
    proxy_redirect off;

    # Websockets
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

```
location / {
    root /usr/share/nginx/html;
    index index.html;
    try_files $uri $uri/ /index.html;
}

location /api {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_pass https://spring:8080;

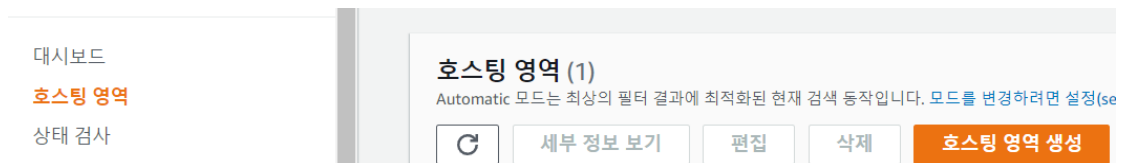
    proxy_connect_timeout      150;
    proxy_send_timeout         100;
    proxy_read_timeout         100;

    proxy_buffer_size          8k;
    proxy_buffers               4 32k;
    proxy_busy_buffers_size     64k;
    proxy_temp_file_write_size 64k;
}
...
```


7. 외부 서비스

AWS Route 53 의 호스팅과 가비아의 도메인을 활용하여 배포합니다.

1. 가비아 이용해 도메인 발급
2. AWS Route 53 에 접속하여 호스팅 영역 탭의 호스팅 영역 생성 클릭



3. 발급받은 도메인 입력하여 호스팅 시작

호스팅 영역 생성 정보

호스팅 영역 구성

호스팅 영역은 example.com 같은 도메인과 관련 하위 도메인에 대한 트래픽을 라우팅하는 방식에 대한 정보를

도메인 이름 정보
트래픽을 라우팅할 도메인의 이름입니다.

유효한 문자: a-z, 0-9 및 ! " # \$ % & ' () * + , - / : ; < = > ? @ [\] ^ _ ` { | } . ~

설명 - 선택 사항 정보
이 값을 사용하면 이름이 동일한 호스팅 영역을 구별할 수 있습니다.

설명은 최대 256자입니다. 0/256

유형 정보
유형은 인터넷 또는 Amazon VPC에서 트래픽을 라우팅할지 여부를 가리킵니다.

☒ 퍼블릭 호스팅 영역
퍼블릭 호스팅 영역은 인터넷에서 트래픽을 라우팅하는 방식을 결정합니다.

☐ 프라이빗 호스팅 영역
프라이빗 호스팅 영역은 Amazon VPC 내에서 트래픽을 라우팅하는 방식을 결정합니다.

4. 생성된 호스팅 영역을 클릭하여 EC2 의 IP 주소를 기준으로 레코드 생성

빠른 레코드 생성 [정보](#) [마법사로 전환](#)

▼ 레코드 1 삭제

레코드 이름 [정보](#)
 hoopaa.site

레코드 유형 [정보](#)

A - IPv4 주소 및 일부 AWS 리소스로 트래픽 라우팅 ▼

☒ 별칭

값 [정보](#)

3.38.181.197

별도의 줄에 여러 값을 입력합니다.

TTL(초) [정보](#)

300 1분 1시간 1일

 라우팅 정책 [정보](#)

단순 라우팅 ▼

권장 값: 60~172,800(2 일)

5. 가비아의 도메인 통합 관리툴에서 네임서버에 AWS Route 53 의 NS 유형의 주소 입력

네임서버 설정

1차	ns-1512.awsdns-61.org
2차	ns-1888.awsdns-44.co.uk
3차	ns-434.awsdns-54.com
4차	ns-1003.awsdns-61.net