



# 포팅 메뉴얼

[포트번호](#)

[버전](#)

[빌드 메뉴얼](#)

[1. Jenkins 배포](#)

[도커파일](#)

[도커컴포즈](#)

[Jenkinsfile](#)

[2. 백엔드, 프론트 이미지 빌드](#)

[스프링 도커파일](#)

[장고 도커파일](#)

[노드 도커파일](#)

[3. Nginx 설정 및 프론트 서빙](#)

[nginx.conf](#)

[도커 컴포즈](#)

[4. DB 컨테이너 실행](#)

[MySQL 배포DB서버](#)

[MySQL 개발DB서버](#)

[레디스 캐시 서버](#)

[MySQL 컨테이너 실행 방법](#)

[5. 비밀키 세팅](#)

[Wear OS Watch Install](#)

[Android 설정](#)

[주요 라이브러리 및 플러그인 버전](#)

[프로젝트 설치](#)

[사전 요구 사항](#)

[Android Studio 사용](#)

[SeoLo\\_Watch.apk 설치](#)

[Flutter App Install](#)

## 포트번호

	PORT 번호
http	80
https	443
Spring	8080

	PORT 번호
Django	8000
Jenkins	9000
MySQL-Dev	3306
MySQL-Prod	3316
Redis-Jwt	6378
Redis-Jwt-Dev	6379
Redis-Jwt-News	6380

## 버전

- node 20.10.0
- Recoil 0.7.7
- TypeScript 5.2.2
- Nginx 1.25.5
- Java 17
- Gradle 8.5
- Spring 3.2.5
- Django 5.0.4
- Flutter 3.19.5
- Dart 3.3.3
- Kotlin 1.9.0

## 빌드 메뉴얼

### 1. Jenkins 배포

#### 도커파일

```
FROM jenkins/jenkins
```

```
USER root
```

```

ENV DEBIAN_FRONTEND noninteractive
ENV DEBCONF_NOWARNINGS="yes"

RUN apt-get -y update && \
    apt-get install -y --no-install-recommends \
    vim \
    apt-utils \
    ca-certificates \
    curl \
    gnupg \
    lsb-release \
    sudo

# Docker CE 설치
RUN curl -fsSL https://download.docker.com/linux/debian/gpg | gpg
    echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-ar
    apt-get -y update && \
    apt-get -y install docker-ce docker-ce-cli containerd.io

# Docker Compose 설치
RUN curl -L "https://github.com/docker/compose/releases/download/
    chmod +x /usr/local/bin/docker-compose

# Jenkins 사용자에게 sudo 권한 부여
RUN echo "jenkins ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

USER jenkins
EXPOSE 8080

```

## 도커컴포즈

```

version: '3.8'

services:
  jenkins:
    container_name: seolo-jenkins
    build:
      context: .
      dockerfile: Dockerfile # 사용할 Dockerfile의 이름, 폴더 내 다른
    ports:

```

```

    - "9000:8080"
volumes:
  - /var/jenkins_home:/var/jenkins_home
  - /var/run/docker.sock:/var/run/docker.sock
networks:
  - jenkins-network

networks:
jenkins-network:
  external: true

```

## Jenkinsfile

```

pipeline {
  agent any
  environment {
    // 환경 변수 정의
    DOCKER_COMPOSE_FILE = 'DevOps/docker-compose.yml'
    REPO_URL = 'https://lab.ssafy.com/s10-final/S10P31C104.git'
    BRANCH_NAME = 'feature/be-auth'
    CREDENTIALS_ID = 'seolo-git-token'
    GITHUB_CREDENTIALS_ID = 'seolo-github-token'
    GITHUB_REPO_URL = 'https://github.com/makeUgreat/seolo-cc'
    GITHUB_REPO_URL_DJANGO = 'https://github.com/makeUgreat/seolo-django'
    PROFILES_ACTIVE = 'dev'
  }

  stages {
    stage('Set Permissions') {
      steps {
        script {
          // 권한 설정
          sh 'sudo chmod -R 777 /var/jenkins_home/workspace'
          echo "Permissions have been set to 777 for all"
        }
      }
    }
    stage('Prepare Workspace') {
      steps {
        script {

```

```

env.PROFILES_ACTIVE = (env.BRANCH_NAME == 'ma

// 현재 작업 공간 전체를 정리
echo "Attempting to clean the workspace..."
sh 'pwd'
try {
    deleteDir()
    echo "Workspace cleaned successfully."
} catch (Exception e) {
    echo "Failed to clean the workspace."
    echo "Error: ${e.getMessage()}"
    sh 'ls -la' // 현재 디렉토리의 파일 목록과 권
}
}
}

stage('Clone Repository') {
    steps {
        script {
            // Git 저장소에서 develop 브랜치 클론
            git branch: env.BRANCH_NAME, credentialsId: e

            // Mattermost에 빌드 시작 메시지 보내기
            def Author_ID = sh(script: "git show -s --pre
            def Author_Name = sh(script: "git show -s --p
            mattermostSend(color: 'warning',
                message: "빌드 시작: ${env.JOB_NAME} #${env
                endpoint: 'https://meeting.ssafy.com/hook
                channel: 'C104Build'
            )

            // SECRET KEY 저장소 clone
            dir('backend/seolo/src/main/resources') {
                echo "Active profile is ${env.PROFILES_AC

                // dev로 profile 변경
                sh "sed -i 's/active: local/active: ${env
                sh "cat application.yml"
                sh 'mkdir -p confidence'
                dir('confidence') {

```

```

        git branch: 'master', credentialsId:
    }
}

// 새로운 서브모듈 클론
dir('backend/news/news') {
    sh 'mkdir -p setting'
    dir('setting') {
        git branch: 'master', credentialsId:
    }
}
}
}

stage('Prepare Docker Compose') {
    steps {
        script {
            // docker-compose 설치 확인 및 설치
            sh '''
            if ! command -v docker-compose &> /dev/null
            then
                echo "Installing docker-compose..."
                sudo curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
                sudo chmod +x /usr/local/bin/docker-compose
            else
                echo "docker-compose is already installed"
            fi
            '''
        }
    }
}

stage('Shutdown Existing Services') {
    steps {
        script {
            // 이미 실행 중인 서비스를 종료
            sh "docker-compose -f ${env.DOCKER_COMPOSE_FILE} down"
        }
    }
}
}

```

```

stage('Build Nginx & Frontend') {
    steps {
        script {
            dir('frontend/web') {
                sh "pwd"
                sh "docker build -t seolo-nginx/front:latest"
            }
        }
    }
}

stage('Compose') {
    steps {
        script {
            sh "docker-compose -f ${env.DOCKER_COMPOSE_FILE} up"
        }
    }
}

stage('Cleanup') {
    steps {
        script {
            // 미사용 Docker 이미지 정리
            sh 'docker image prune --force --filter "until=24h"'
        }
    }
}

post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=format:%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=format:%an", returnStdout: true).trim()
            mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
                endpoint: 'https://meeting.ssafy.com/hooks/xc1nh4',
                channel: 'C104Build'
            )
        }
    }
}

```

```

    )
  }
}

failure {
  script {
    def Author_ID = sh(script: "git show -s --pretty=
    def Author_Name = sh(script: "git show -s --prett
    mattermostSend (color: 'danger',
    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_N
    endpoint: 'https://meeting.ssafy.com/hooks/xc1nh4
    channel: 'C104Build'
  )
}
}
}
}
}

```

## 2. 백엔드, 프론트 이미지 빌드

### 스프링 도커파일

```

# 1단계: Gradle을 이용해 빌드
FROM gradle:8.5.0-jdk17-alpine AS build

WORKDIR /app

# Gradle 설정 파일들을 Docker 이미지 안으로 복사
COPY gradlew .
COPY gradle gradle
# 프로젝트의 소스 코드와 빌드 파일 복사
COPY build.gradle .
COPY settings.gradle .
COPY src src

RUN chmod +x ./gradlew
# bootJar를 통해 테스트없이 빌드 수행

```



```

RUN ./gradlew bootJar

# 2단계: Java Runtime 이미지 안에 빌드 결과물을 넣어 실행
FROM azul/zulu-openjdk:17

WORKDIR /app

# 빌드 결과물을 이전 단계에서 복사
COPY --from=build /app/build/libs/*.jar /app.jar

# 서버 실행
ENTRYPOINT ["java", "-jar", "-Djava.security.egd=file:/dev/./urand"]

```

## 장고 도커파일

```

# 베이스 이미지로 Python 공식 이미지를 사용합니다.
FROM python:3.10

# 환경 변수 설정
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# 작업 디렉토리를 /app으로 설정합니다.
WORKDIR /app

# 의존성 설치
COPY requirements.txt /app

# 최신 버전의 pip를 설치합니다.
RUN /usr/local/bin/python -m pip install --upgrade pip

# 필요한 라이브러리 설치
RUN pip install --no-cache-dir -r requirements.txt

# 소스 코드 복사
COPY ./ /app

# gunicorn 설치
RUN pip install gunicorn

```

```
# Python 경로 설정 (my_setting.py 파일이 있는 디렉토리를 포함)
ENV PYTHONPATH=/app/news

# Docker 컨테이너 내에서 실행될 명령을 지정합니다.
CMD ["unicorn", "--bind", "0.0.0.0:8000", "news.wsgi:application"]
```

## 노드 도커파일

```
# 기본 이미지로 Node.js 버전 20.11.0 사용
FROM node:20.11.0 as build

# 작업 디렉토리 설정
WORKDIR /usr/src/app

# package.json 및 package-lock.json을 복사하여 종속성 설치
COPY package*.json ./

# 종속성 설치
RUN npm install

# 나머지 애플리케이션 코드 복사
COPY . .

# 프론트엔드 코드 빌드
RUN npm run build

# NGINX 이미지 생성
FROM nginx:latest

# NGINX에서 작업 디렉토리 설정
WORKDIR /usr/share/nginx/html

# 기본 NGINX 정적 콘텐츠 제거
RUN rm -rf /*

# Node.js 빌드 단계에서 빌드된 프론트엔드 코드 복사
COPY --from=build /usr/src/app/dist/ .
```

### 3. Nginx 설정 및 프론트 서빙

#### nginx.conf

```
events {
    worker_connections 1024;
}

http {
    client_max_body_size 20M;

    # WebSocket 연결을 위한 변수 설정
    map $http_upgrade $connection_upgrade {
        default upgrade;
        '' close;
    }

    upstream backend {
        server back:8080;
    }

    upstream news {
        server seolo-news:8000;
    }

    server {
        listen 80;
        server_name k10c104.p.ssafy.io;
        return 301 https://$server_name$request_uri; # HTTP 요청을
    }

    server {
        listen 443 ssl;
        http2 on;
        server_name k10c104.p.ssafy.io;
        include /etc/nginx/mime.types;

        ssl_certificate /etc/ssl/certs/fullchain1.pem;
        ssl_certificate_key /etc/ssl/private/privkey1.pem;
```

```

# FRONTEND WEB 프록시
location / {
    root /usr/share/nginx/html; # 정적 파일 서빙
    index index.html index.htm;
    try_files $uri $uri/ /index.html =404;
}

# BACKEND 프록시 (웹소켓 및 SSE 헤더 포함)
location /api/ {
    rewrite ^/api(.*) $1 break; # /api 제거
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    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_buffering off;
    proxy_cache off;
    proxy_read_timeout 7200s; # SSE를 위한 설정
}

location /django/ {
    rewrite ^/django(.*) $1 break; #
    proxy_pass http://news;
    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;
}
}
}

```

## 도커 컴포즈

```
version: '3.8'
```

```

services:
  nginx:
    container_name: seolo-nginx
    image: seolo-nginx/front:latest
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - /home/ubuntu/certificates/k10c104.p.ssafy.io/fullchain1.p
      - /home/ubuntu/certificates/k10c104.p.ssafy.io/privkey1.pem
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - back
    environment:
      - TZ=Asia/Seoul
    networks:
      - jenkins-network

  back:
    container_name: seolo-back
    build:
      context: ../backend/seolo
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      - TZ=Asia/Seoul
    networks:
      - jenkins-network

  news-server:
    container_name: seolo-news
    build:
      context: ../backend/news
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    environment:
      - TZ=Asia/Seoul
    networks:
      - jenkins-network

```

```
networks:
  jenkins-network:
    external: true
```

## 4. DB 컨테이너 실행

### MySQL 배포DB서버

ID : root

PWD: sEoLo202404181046

```
docker run --name seolo-product-mysql -e MYSQL_ROOT_PASSWORD=sEoL
```

### MySQL 개발DB서버

ID : root

PWD: develop1234

```
docker run --name seolo-dev-mysql -e MYSQL_ROOT_PASSWORD=develop1
```

### 레디스 캐시 서버

```
docker run -d --name redis-token-local -p 6378:6379 redis/redis-s
```

```
docker run -d --name redis-token -p 6379:6379 redis/redis-stack-s
```

```
docker run -d --name redis-token-local -p 6380:6379 redis/redis-s
```

## MySQL 컨테이너 실행 방법

### 1. Docker 설치 확인:

우선, 시스템에 Docker가 설치되어 있고 실행 중인지 확인합니다. 설치되어 있지 않다면,

Docker 공식 웹사이트에서 설치 지침을 따라 설치할 수 있습니다.

## 2. MySQL 컨테이너 실행:

다음 명령어를 사용하여 MySQL 컨테이너를 실행합니다. 이 명령어는 MySQL의 최신 버전 컨테이너를 실행하며, 필요한 환경 변수(예: root 비밀번호)를 설정합니다.

```
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -p 8282:3306 -v /opt/datadir:/var/lib/mysql -d mysql:latest
```

볼륨매핑

```
docker run -v /opt/datadir:/var/lib/mysql mysql
```

이 명령어에서 `some-mysql` 은 컨테이너의 이름을 지정하며, `MYSQL_ROOT_PASSWORD` 는 MySQL root 사용자의 비밀번호를 설정합니다. `mysql:latest` 는 사용할 MySQL 이미지와 태그를 지정합니다(여기서는 최신 버전을 사용합니다).

## 3. 컨테이너 상태 확인:

MySQL 컨테이너가 성공적으로 실행되었는지 확인하기 위해 다음 명령어를 사용합니다.

```
docker ps
```

## 4. MySQL 컨테이너에 접속:

컨테이너가 실행되고 나면, 다음 명령어를 사용하여 MySQL 컨테이너 내부의 MySQL 쉘에 접속할 수 있습니다.

```
sudo docker exec -it fodongDB bash -c "mysql -u root -p"
```

여기서 `some-mysql` 은 컨테이너의 이름이며, `mysql -uroot -p` 는 MySQL 쉘에 접속하기 위한 명령어입니다. 비밀번호 입력을 요청하면 앞서 설정한 `MYSQL_ROOT_PASSWORD` 값을 입력합니다.

## MySQL 컨테이너 설정하기

```
sudo docker exec -it fodongDB bash -c "mysql -u root -p"
```

명령어를 통해 MySQL 설정 파일 들어가기

```
CREATE USER 'username'@'%' IDENTIFIED BY 'password';  
GRANT ALL PRIVILEGES ON *.* TO 'username'@'%' WITH GRANT OPT
```

```
ION;  
FLUSH PRIVILEGES;
```

사용자를 생성할 때 접근 허용 범위를 다음과 같이 설정 할 수 있습니다.

'Username'@'%' : 해당 사용자는 외부에서 접근가능

'Username'@'localhost' : 해당 사용자는 내부에서만 접근 가능

'Username'@'xxx.xxx.xxx.xxx' : 해당 사용자는 지정한 ip주소로만 접근 가능


## 싸피 EC2 로 접속하는 경우 인텔리제이 데이터그립 설정

포트 열어주기 (UFW) SSH로 EC2 서버 접속해서 포트 열어준다

## MySQL 외부접근 IP열어주기

\*Ubuntu 기준

1. 도커를 통해 mysql 컨테이너 설치하고
2. 컨테이너 들어가서 my.cnf 파일을 수정한다( 보통 경로 /etc/my.cnf )
3. [mysqld] 섹션에 bind-address = 0.0.0.0 을 추가하거나 수정한다

 이때 편집 명령어 vi, nano 등 안먹히면 골치아픔 (아래 링크 참조)

 [해결방법. Docker Volume](#)

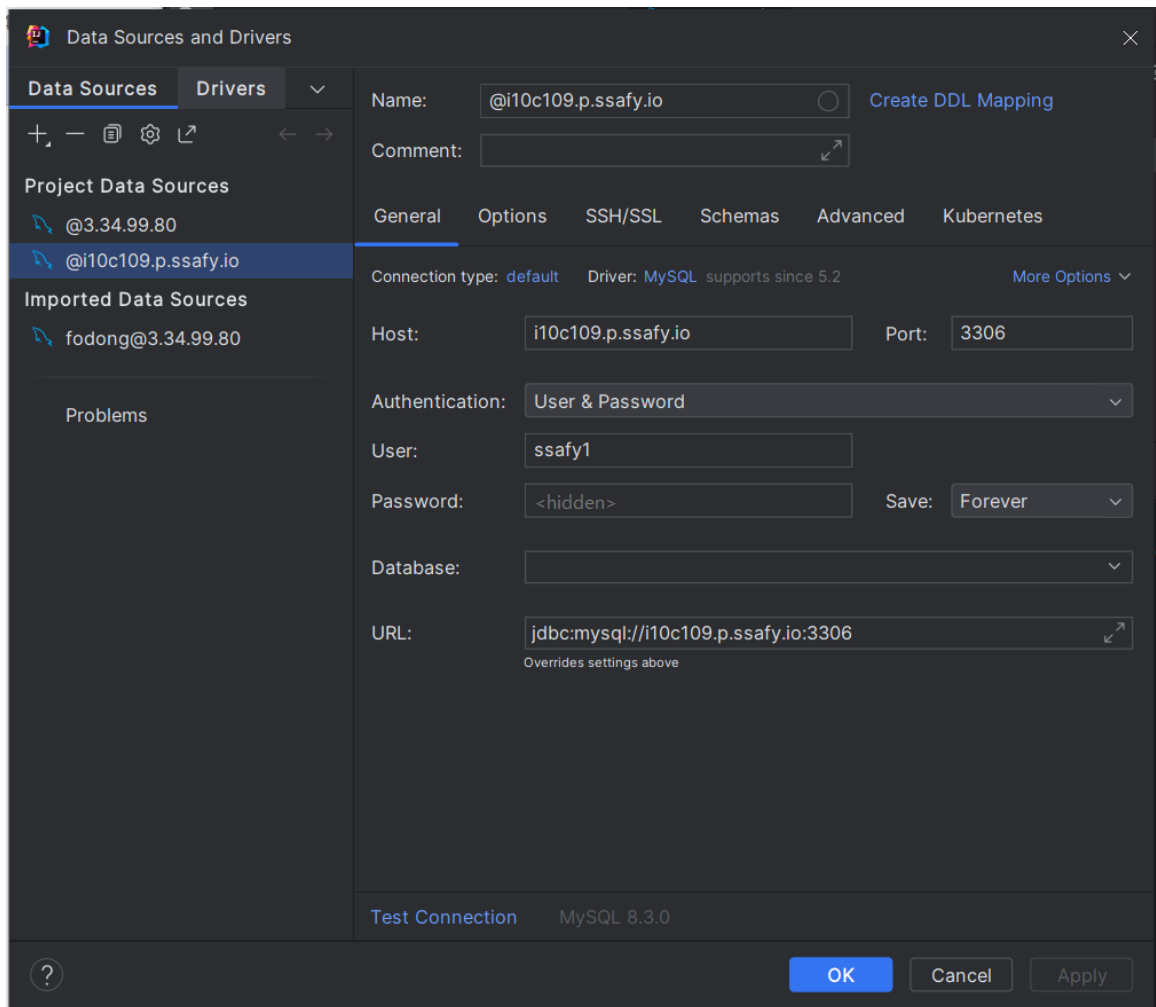
## AWS 웹 콘솔을 이용하는 경우

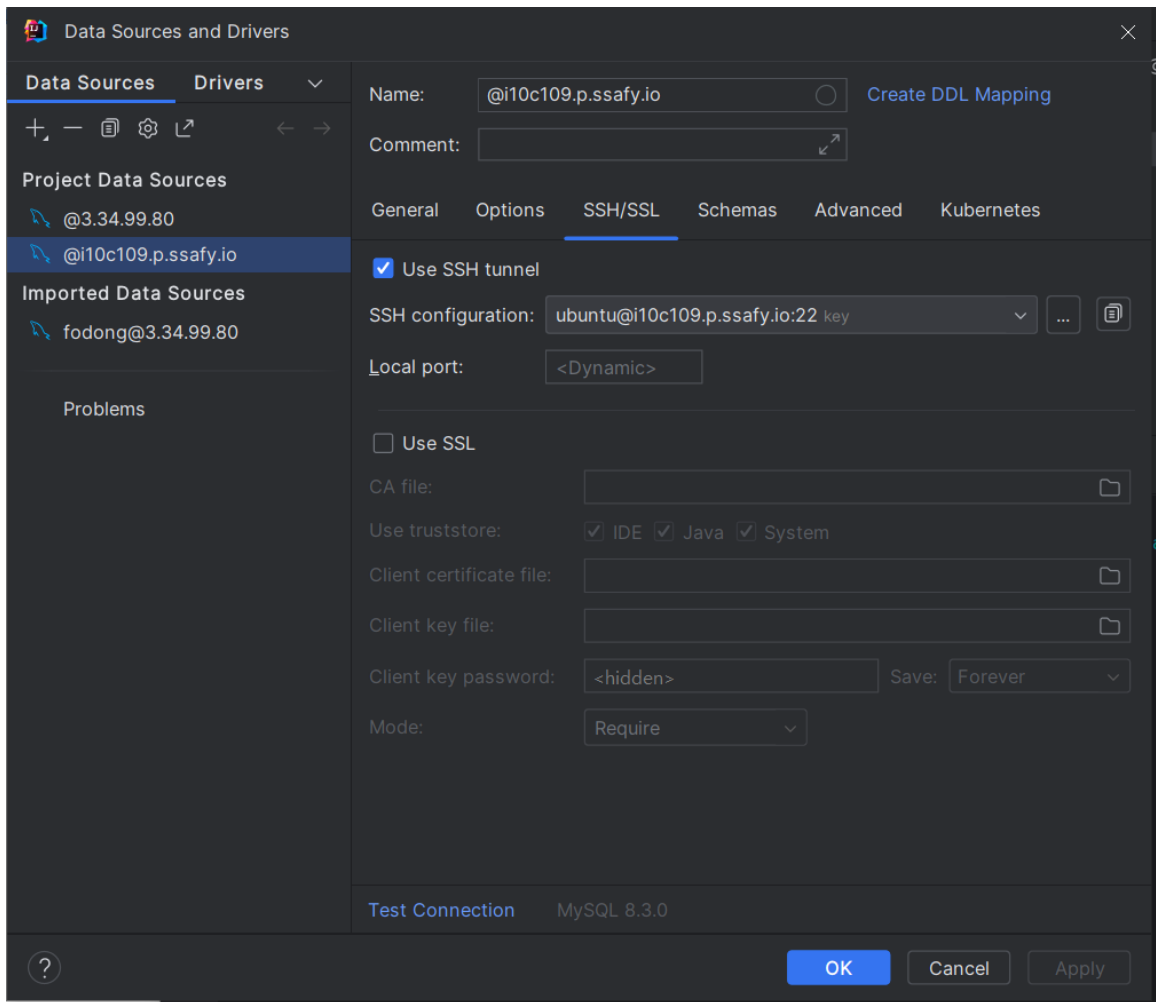
포트 열어주기

- AWS 관리 콘솔에 로그인하여 EC2 대시보드로 이동합니다.
- 사용 중인 EC2 인스턴스가 속한 보안 그룹을 찾습니다.
- 해당 보안 그룹의 인바운드 규칙을 편집하여 로컬 컴퓨터의 IP 주소 또는 필요한 범위의 IP 주소로부터 MySQL 포트(기본적으로 3306)에 대한 접근을 허용합니다.

## 인텔리제이 데이터그립 연결







#### 프로젝트 데이터베이스 연결 정보 설정:

- 프로젝트의 데이터베이스 연결 설정에서 EC2 MySQL 서버의 주소, 사용자 이름, 비밀번호, 데이터베이스 이름을 입력합니다.
- EC2 인스턴스의 공개 IP 주소나 도메인 이름을 사용하여 서버 주소를 설정합니다.



IntelliJ에서 데이터베이스에 연결하기 위한 설정 과정은 다음과 같습니다. 이 과정은 IntelliJ의 데이터베이스 툴을 사용하여 원격 또는 로컬 데이터베이스에 접속하는 방법을 설명합니다:

#### 1. 데이터 소스 추가:

- IntelliJ IDE를 열고, 오른쪽 상단에 있는 'Database' 탭을 클릭합니다.
- 'Database' 툴 윈도우에서 '+' 버튼을 클릭하고 'Data Source'를 선택한 후, 사용할 데이터베이스 유형 (예: MySQL)을 선택합니다.

#### 2. 연결 정보 입력:

- 'Host' 필드에 MySQL 서버가 실행 중인 호스트의 IP 주소나 도메인 이름을 입력합니다. EC2 인스턴스를 사용하는 경우, EC2 인스턴스의 공개 IP 주소를 입력합니다.
- 'Port' 필드에는 MySQL 서버의 포트 번호를 입력합니다 (기본 값은 3306입니다).
- 'Database' 필드에는 연결하고자 하는 MySQL 데이터베이스의 이름을 입력합니다.
- 'User'와 'Password' 필드에는 데이터베이스에 접속할 사용자 이름과 비밀번호를 입력합니다. 이 정보는 MySQL 서버에 설정한 사용자 계정 정보와 일치해야 합니다.

#### 3. 연결 테스트:

- 설정을 완료한 후, 'Test Connection' 버튼을 클릭하여 데이터베이스 연결을 테스트합니다. 연결이 성공하면 'Successful' 메시지가 표시됩니다.

#### 4. 저장 및 사용:

- 연결 정보를 모두 입력하고 테스트가 성공적이면, 'OK' 버튼을 클릭하여 설정을 저장합니다.
- 이제 IntelliJ의 'Database' 탭에서 해당 데이터베이스를 볼 수 있으며, 테이블을 조회하거나 SQL 쿼리를 실행하는 등의 작업을 할 수 있습니다.

IntelliJ에서 데이터베이스 연결을 설정하는 이 과정을 통해 개발자는 애플리케이션 개발 중 데이터베이스와의 상호작용을 손쉽게 관리할 수 있습니다.

## 5. 비밀키 세팅

비밀키 외부저장소에 credential 있는 계정을 등록해

`resources/confidence/application-secret.yml` 파일을 클론해야한다.



민감정보가 있어 공유 불가

## Wear OS Watch Install

### Android 설정

- compileSdk = 34
- minSdk = 30
- targetSdk = 34

### 주요 라이브러리 및 플러그인 버전

```
agp = "8.4.0"
kotlin = "1.9.0"
playServicesWearable = "18.1.0"
composeBom = "2023.08.00"
composeMaterial = "1.2.1"
composeFoundation = "1.2.1"
activityCompose = "1.7.2"
coreSplashscreen = "1.0.1"
constraintlayout = "2.1.4"
appcompat = "1.6.1"
viewpager2 = "1.0.0"
material = "1.11.0"
securityCryptoKtx = "1.0.0"
```

## 프로젝트 설치

### 사전 요구 사항

- Android Studio 최신 버전 설치
- Java Development Kit (JDK) 8 이상 설치
- Android SDK 설치 및 환경 변수 설정

## Android Studio 사용

- Android Studio를 실행하고, **File > Open** 을 선택하여 프로젝트 디렉토리를 엽니다.
- 상단 메뉴에서 **Sync Project with Gradle Files** 버튼을 클릭하여 필요한 의존성을 설치합니다.
- **Run** 버튼을 클릭하여 Wear OS 에뮬레이터 또는 실제 기기에서 애플리케이션을 실행합니다.

## SeoLo\_Watch.apk 설치

- 다운로드 경로
  - <https://drive.google.com/file/d/1nZ5mIgbr4J2fttegI1MDPyTAo2awxMgp/view>


## Flutter App Install

- build.gradle

```
defaultConfig {
    minSdkVersion 21
}
```

- app 다운로드 경로

SeoLo\_App.apk

 [https://drive.google.com/file/d/1udeSx9Hdvu-tOLqIKKp5\\_wNsBOWIPJtL/view?usp=drive\\_link](https://drive.google.com/file/d/1udeSx9Hdvu-tOLqIKKp5_wNsBOWIPJtL/view?usp=drive_link)

- code에서 설치

```
- flutter build apk
- flutter install (기기 연결 시)
```

