

# 산출물

≡ 태그

## 1. Gitlab 소스 클론 이후 빌드 및 배포할 수 있도록 정리한 문서

### 사용 툴 및 버전

- 웹서버
  - nginx/1.18.0 (Ubuntu)
- 개발환경 IDE
  - IntelliJ IDEA 2023.1.3 - 백엔드(SpringBoot) 개발
    - jdk : OpenJDK - 17.0.7
  - Visual Studio Code 1.81.1 - 프론트엔드(React, Typescript) 개발
    - nodeJS : 18.17.0
- EC2
  - Ubuntu 20.04.6
- Docker
  - Docker Desktop (로컬 개발시 사용)
  - EC2 내부 Docker 버전 : 24.0.5
- Jenkins
  - 도커 컨테이너 Jenkins
  - 이미지 : jenkins/jenkins:jdk17

### 프론트, 백엔드 설정파일

#### 백엔드 설정파일

##### application.yml

- {DB\_URL} 등의 정보들은 Jenkins에 등록해둔 Credential이
- 서버에 배포할 때 sed 명령어를 이용해 치환이 된다.
- Jenkins에 Credential들을 등록해두고 빌드 시 값이 대입되도록 설정했는데, 아래 빌드 & 배포 과정에서 설정을 어떻게 했는지 설명한다.

```
server:
  port: 9999
  servlet:
    context-path: /api

spring:
  datasource:
    url: ${DB_URL}
    username: ${DB_USER}
    password: ${DB_PW}
    driver-class-name: com.mysql.cj.jdbc.Driver

jpa:
  hibernate:
    ddl-auto: update
  properties:
```

```

hibernate:
  dialect: org.hibernate.dialect.MySQL8Dialect
  format_sql: true
  default_batch_fetch_size: 1000 # 최적화 옵션
sql:
  init:
    encoding: UTF-8

redis:
  host: tagyou.site
  port: ${REDIS_PORT}

security:
  oauth2:
    client:
      registration:
        kakao:
          client-id: ${KAKAO_CLIENT}
          redirect-uri: https://tagyou.site/api/login/oauth2/code/kakao
          authorization-grant-type: authorization_code
          client-authentication-method: client_secret_post
          client-name: Kakao
          scope: profile_nickname, profile_image, account_email

      provider:
        kakao:
          authorization-uri: https://kauth.kakao.com/oauth/authorize
          token-uri: https://kauth.kakao.com/oauth/token
          user-info-uri: https://kapi.kakao.com/v2/user/me
          user-name-attribute: id

  servlet:
    multipart:
      maxFileSize: 10MB
      maxRequestSize: 20MB

logging:
  level:
    org.hibernate.SQL: debug

cloud:
  aws:
    s3:
      bucket: tag-u #tagyou
      stack.auto: false
      region.static: ap-northeast-2
      credentials:
        accessKey: ${S3_ACCESS}
        secretKey: ${S3_SECRET}

#OPENVIDU_URL: http://localhost:4443/
OPENVIDU_URL: ${OPENVIDU_URL}
OPENVIDU_SECRET: ${OPENVIDU_SECRET}

#

```

## build.gradle

```

plugins {
    id 'java'
    id 'org.springframework.boot' version '3.1.1'
    id 'io.spring.dependency-management' version '1.1.0'
}

group = 'com.ssafy'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '17'
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
}

```

```

implementation 'org.springframework.boot:spring-boot-starter-jdbc'
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.boot:spring-boot-starter-websocket'
implementation 'org.springframework.boot:spring-boot-starter-data-redis'
implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.0.2'
compileOnly 'org.projectlombok:lombok'
developmentOnly 'org.springframework.boot:spring-boot-devtools'
runtimeOnly 'com.mysql:mysql-connector-j'
annotationProcessor 'org.projectlombok:lombok'
// querydsl 패키지
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
annotationProcessor "com.querydsl:querydsl-apt:${dependencyManagement.importedProperties['querydsl.version']}:jakarta"
annotationProcessor "jakarta.annotation:jakarta.annotation-api"
annotationProcessor "jakarta.persistence:jakarta.persistence-api"

testImplementation 'org.springframework.boot:spring-boot-starter-test'

implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.springframework.boot:spring-boot-starter-validation'
implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.2'
testImplementation 'org.springframework.security:spring-security-test'

implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE'
implementation 'commons-io:commons-io:2.6'

implementation 'io.openvidu:openvidu-java-client:2.28.0'

implementation 'org.webjars:sockjs-client:1.5.1'
implementation 'org.webjars:stomp-websocket:2.3.4'
implementation 'org.springframework:spring-messaging:6.0.3'
implementation 'org.springframework.security:spring-security-messaging:6.0.2'

}

tasks.named('test') {
    useJUnitPlatform()
}

// Querydsl 설정
def generated = 'src/main/generated'

// querydsl QClass 파일 생성 위치 지정
tasks.withType(JavaCompile) {
    options.getGeneratedSourceOutputDirectory().set(file(generated))
}

// java source set 에 querydsl QClass 위치 추가
sourceSets {
    main.java.srcDirs += [ generated ]
}

// gradle clean 시에 QClass 디렉토리 삭제
clean {
    delete file(generated)
}

```

## 프론트 설정파일

### package.json

```

{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.14.0",
    "@mui/material": "^5.14.2",
    "@mui/styled-engine-sc": "^5.12.0",
    "@react-spring/web": "^9.7.3",
    "@stomp/stompjs": "^7.0.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/antd": "^1.0.0",

```

```

"@types/express": "^4.17.17",
"@types/jest": "^27.5.2",
"@types/kakao-js-sdk": "^1.39.1",
"@types/node": "^16.18.38",
"@types/react": "^18.2.15",
"@types/react-cookies": "^0.1.0",
"@types/react-dom": "^18.2.7",
"@types/react-modal": "^3.16.0",
"@types/react-motion": "^0.0.34",
"@types/react-router-dom": "^5.3.3",
"@types/stompjs": "^2.3.5",
"@types/yajireo__tagify": "^4.17.0",
"@yajireo/tagify": "^4.17.8",
"axios": "^1.4.0",
"emotion-normalize": "^11.0.1",
"express": "^4.18.2",
"framer-motion": "^10.15.1",
"history": "^5.3.0",
"normalize.css": "^8.0.1",
"openvidu-browser": "^2.28.0",
"openvidu-react": "^2.27.0",
"react": "^18.2.0",
"react-cookie": "^4.1.1",
"react-cookies": "^0.1.1",
"react-dom": "^18.2.0",
"react-hook-form": "^7.45.2",
"react-modal": "^3.16.1",
"react-query": "^3.39.3",
"react-router-dom": "^6.14.2",
"react-scripts": "5.0.1",
"react-slick": "^0.29.0",
"react-transition-group": "^4.4.5",
"react-use-websocket": "^3.0.0",
"recoil": "^0.7.7",
"recoil-persist": "^5.1.0",
"slick-carousel": "^1.8.1",
"socket.io-client": "^4.7.2",
"sockjs-client": "^1.6.1",
"stompjs": "^2.3.3",
"styled-components": "^5.3.11",
"v6": "^0.0.0",
"web-vitals": "^2.1.4",
"ws": "^8.13.0"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject",
  "storybook": "storybook dev -p 6006",
  "build-storybook": "storybook build"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest",
    "plugin:storybook/recommended"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
},
"devDependencies": {
  "@storybook/addon-essentials": "^7.0.26",
  "@storybook/addon-interactions": "^7.0.26",
  "@storybook/addon-links": "^7.0.26",
  "@storybook/blocks": "^7.0.26",
  "@storybook/preset-create-react-app": "^7.0.26",
  "@storybook/react": "^7.0.26",
  "@storybook/react-webpack5": "^7.0.26",
  "@storybook/testing-library": "^0.0.14-next.2",
  "@types/react-slick": "^0.23.10",
  "@types/sockjs-client": "^1.5.1",
  "babel-plugin-named-exports-order": "^0.0.2",
  "eslint-plugin-storybook": "^0.6.12",
  "http-proxy-middleware": "^2.0.6",
  "msw": "^1.2.3",
  "prop-types": "^15.8.1",

```

```

"storybook": "^7.0.26",
"typescript": "^4.9.5",
"typescript-styled-plugin": "^0.18.3",
"webpack": "^5.88.1"
},
"msw": {
  "workerDirectory": "public"
}
}

```

## 사용 기술 스택

### 백엔드



- SpringBoot 3.1.1
- Spring security 6.1.1
- JPA

### 프론트



- "react": "^18.2.0",
- "recoil": "^0.7.7",
- "storybook": "^7.0.26",
- "typescript": "^4.9.5",
- "@emotion/react": "^11.11.1"
- sourcetree 3.4.14

### 데이터베이스



- MySQL 8.0.34

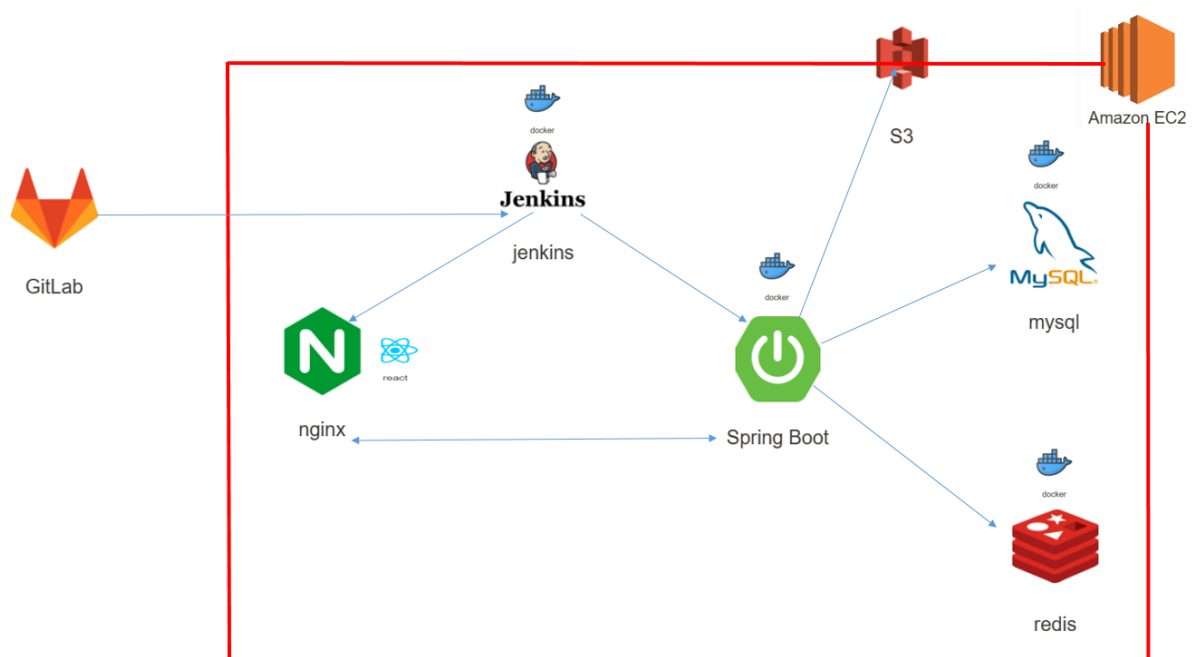
- redis 7.0.12

## 인프라, CI/CD



- EC2 (Ubuntu 20.04.6)
- S3
- jenkins (jenkins/jenkins:jdk17)
- nginx(1.18.0 Ubuntu)
- docker (24.0.5)
- Gitlab (ssafy gitlab )

## 인프라 구조



### 목적 : 빌드 & 배포 자동화

- Gitlab에서 push가 일어나면 Jenkins가 감지하여 빌드한다.
- Jenkins에서 빌드한 파일 배포 : nginx, (Docker)SpringBoot
- SpringBoot는 mysql, redis, S3와 연동상태
  - SpringBoot와 mysql, redis는 같은 도커 네트워크를 공유

## CI/CD 과정



아래의 내용은 처음부터 CI/CD를 구축하는 과정이다.

## 환경

- Gitlab
- Jenkins : openjdk 17버전
  - Jenkins의 jdk 경우 Build할 SpringBoot 프로젝트에서 사용하는 jdk와 동일하게 설정해주었다.
- EC2
  - Ubuntu 환경에서 실행하였음.
- Docker
  - Jenkins를 Docker Container로 구동시킬 것이기 때문에 Docker 필요하다.
  - Jenkins를 이용하여 Build한 결과물(SpringBoot, React) 역시 Docker Container로 구동시켜야 한다.

### Webhook이 필요한 이유.

- Jenkins가 Gitlab에서 **변경이 일어난 것을 감지해야하기 때문**
  - 그래서, Gitlab이 Jenkins에게 **나 변동이 일어났다!** 라고 알려줘야함.
  - 언제 알려줄건가?
    - 여기서 세팅할 환경은 **Push Events**, 즉 **Push가 일어났을 때.**



Gitlab에서 Push를 하면 Jenkins가 감지하게 한다.

- 굳이 Push Events가 아니어도, 목적에 따라 다를 수 있음.
- **결론적으로**



Gitlab → Jenkins (**Webhook 필요**)

## Ubuntu에서 도커 설치하기

- 공식 문서의 내용과 동일한 블로그 참고.

### [Linux] 우분투(Ubuntu)에서 도커(Docker) 설치 하기

가상화 플랫폼인 Docker를 Ubuntu에서 설치하는 방법을 알아보겠습니다. 도커 공식 홈페이지에도 설치 방법이 자세  
히 나와 있습니다. <https://docs.docker.com/engine/install/ubuntu/> Install Docker Engine on Ubuntu  
[docs.docker.com](https://docs.docker.com) 우분투에서 도커 설치하기 우선 apt 패키지 리스트를 업데이트 합니다. \$ sudo apt-get update

🔗 <https://jkim83.tistory.com/166>



// 순서대로 입력한다.

```
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=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/
$(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
```

## Jenkins

### 목적

- Build → Deploy
- Gitlab에서 받은 파일을 Build하여 **이상이 없음을 체크**하고 (build가 되지 않으면 Jenkins에서 막힌다.)
- 이상이 없다면 만들어진 결과물을 **서버**에 전송한다
  - **EC2**에 전송하는 것.

### Jenkins 이미지를 가진 도커 컨테이너

- jdk 버전은 **본인의 프로젝트에 맞게** 설정.
- ec2에서 아래 명령어를 사용

```
sudo docker run -d \
-u root \
-p 9090:8080 \
--name=jenkins \
-v /home/ubuntu/docker/jenkins-data:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
jenkins/jenkins:jdk17

#docker run : Docker 컨테이너 실행
# -d : 백그라운드 모드로 실행
# -u root : 컨테이너 안에서 실행되는 프로세스 사용자 root로 설정
# -p 9090:8080 : 호스트가 9090으로 접속할 경우 8080포트로 포워딩.
# --name=jenkins : 실행 중인 컨테이너의 이름

# -v /home/ubuntu/docker/jenkins-data:/var/jenkins_home
# 호스트 디렉터리 /home/ubuntu/docker/jenkins-data를
# /var/jenkins_home 디렉터리와 마운트

# -v /var/run/docker.sock:/var/run/docker.sock
# 호스트의 Docker 소켓을 컨테이너 내부의 Docker 소켓으로 마운트
# 이를 통해 컨테이너 내부에서 호스트의 Docker 데몬을 사용하여
# 다른 컨테이너를 관리

#jenkins/jenkins:jdk17 : jdk17 버전의 jenkins 이미지
```

### Jenkins 컨테이너 접속 및 설정

💡 순서대로 따라가면 된다.

- 참고한 블로그

#### [CI/CD] Jenkins 설치하기 (Docker)

이번 포스트에서는 Docker를 이용하여 아무런 설정이 되어있지 않은 젠킨스를 설치하는 방법을 정리하겠습니다.  
테스트 환경 OS: Ubuntu Server 22.04 LTS (HVM) 성능: t2.micro(free tier) 만약, EC2에서 Jenkins를 설치하  
신다면 외부에서 Jenkins 접속을 위해서 Security group의 인바운드 규칙 설정을 통해 8080 포트를 열어주셔야  
🔗 <https://seosh817.tistory.com/287>



- ec2에서 아래 명령어 실행.

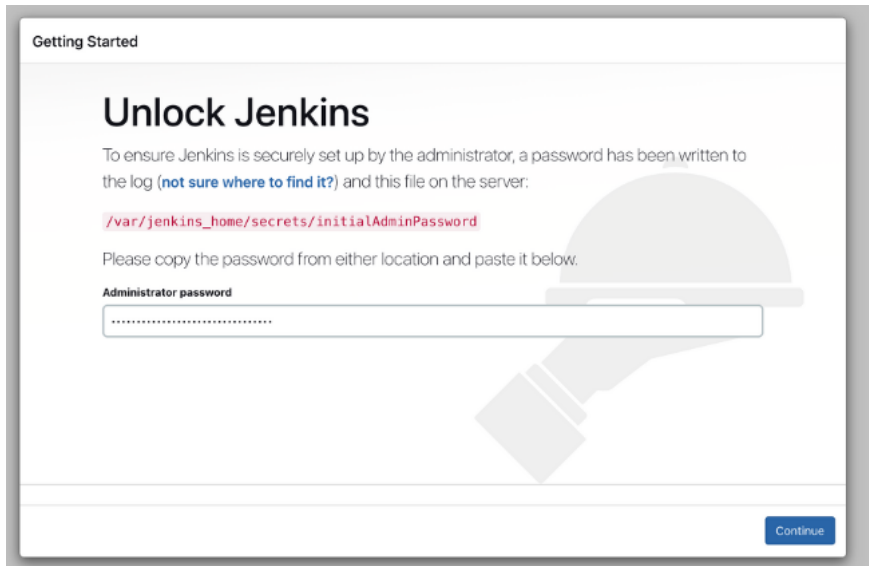


```
docker exec -it jenkins /bin/bash
# jenkins 컨테이너에 접속하는 명령어.

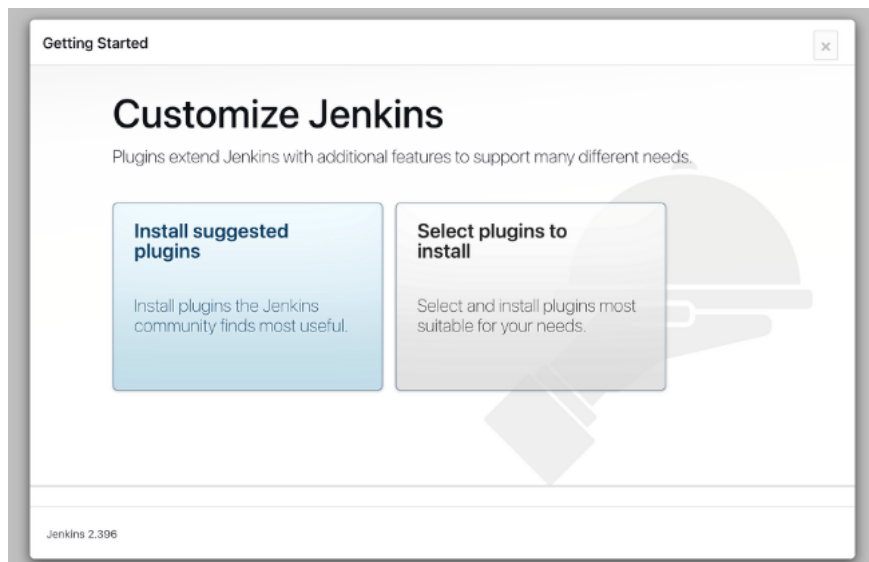
cat /var/jenkins_home/secrets/initialAdminPassword

# cat 명령어의 결과물로 무슨 password가 나타남.
```

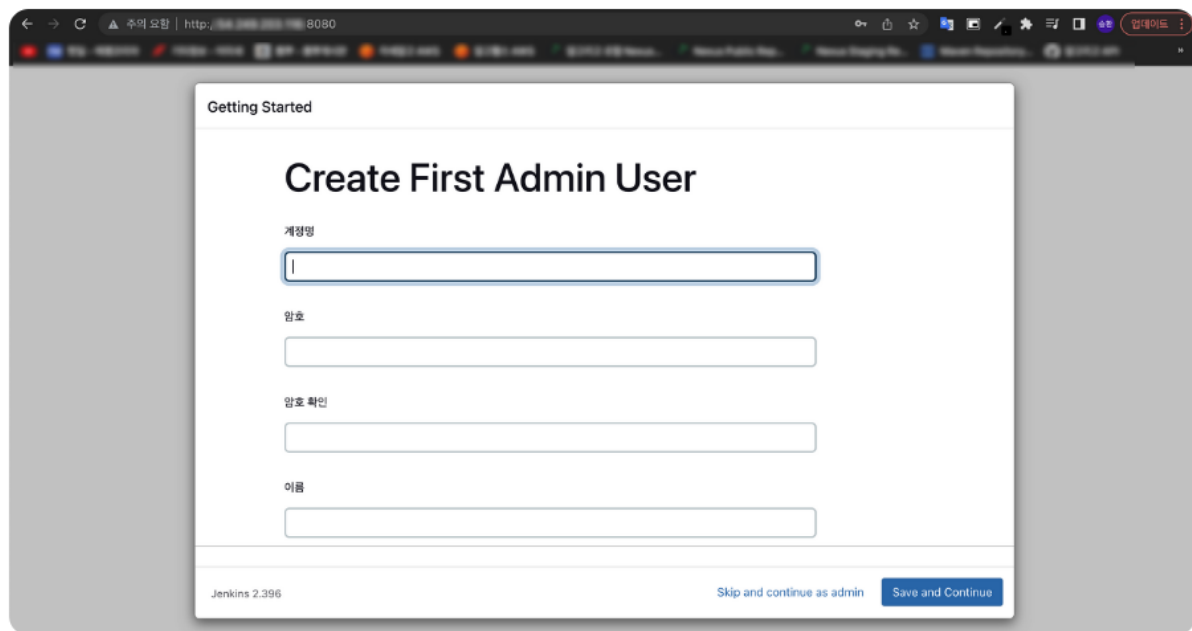
- <http://도메인:9090> 을 이용하여 jenkins에 접속한다.



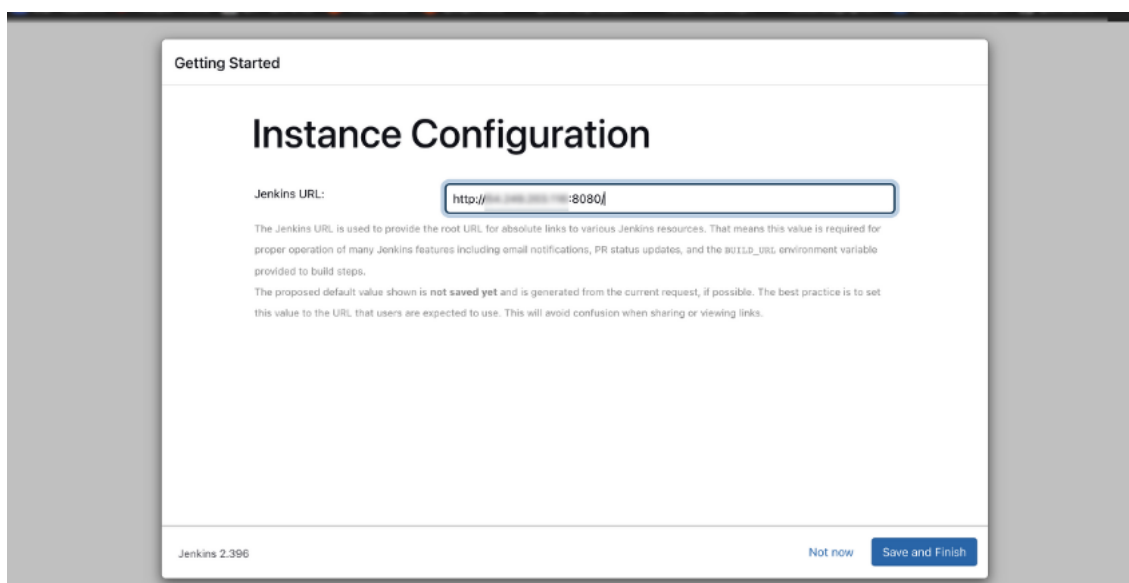
- cat 명령어의 결과를 복사해서 위 화면에 입력 후 Continue



- 비밀번호 입력 후 다음 화면에서 Install suggested plugins 선택



- 설치가 되면 계정을 생성한다.
  - 이 계정은 Jenkins에 로그인 할 때 필요한 계정임



- Jenkins 접속 URL을 입력한다.
- 현재 젠킨스 도커 포트는 9090으로 접속하게 해둔 상태이다.
- 그러므로, [http://도메인이름: 9090](http://도메인이름:9090) 으로 설정한다.

위의 과정을 마치면 jenkins의 기본 홈페이지가 뜰 것임.

## Webhook

### Gitlab과 Webhook이 통신하기 위해 Gitlab에서 Personal Access Token 발급

- 우측 상단 본인 아이디 → Edit Profile → Access Token 들어간다.

#### Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

#### Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

##### Token name

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

##### Expiration date

##### Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☐ api  
Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- ☐ read\_api  
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- ☐ read\_user  
Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- ☐ read\_repository  
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- ☐ write\_repository  
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

- scopes를 선택하는데, 설명을 읽어보고 본인에게 맞는 api를 선택
- 필자의 경우 api를 select 하였음
- Create personal access token 클릭



발급 받은 Access token의 값을 기억해두어야 한다. 다시 보지 못한다.

### Jenkins → Jenkins 관리 → plugins에 들어간다.

+ 새로운 Item

사람 사람

일기상 빌드 기록

설정 Jenkins 관리

문서 My Views

블루 오션 열기

All

+

S	W	Name ↓	최근 성공
✓	☀	Deploy Test	7 days 1 hr #1
✓	☀	test	44 min #345

빌드 대기 목록

▼

빌드 대기 항목이 없습니다.

아이콘: S M L

빌드 실행 상태

▼

1 대기 중

2 대기 중

System Configuration

⚙️

System

환경변수 및 경로 정보등을 설정합니다.

🔧

Tools

Configure tools, their locations and automatic installers.

27

⚙️

Plugins

Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.

🖥️

Nodes

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

☁️

Clouds

Add, remove, and configure cloud instances to provision agents on-demand.

📄

Managed files

e.g. settings.xml for maven, central managed scripts, custom files, ...

- plugins로 들어가서 **Gitlab**을 검색하여 Install 받는다.

## 이후 다시 위의 사진에서 System을 들어간다.

- Gitlab이라 들어와 있는 부분에서 본인에 맞게 작성한다.
- Gitlab host URL은 gitlab 서버임.

GitLab connections

Connection name ?  
A name for the connection

GitLab host URL ?  
The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)

Credentials ?  
API Token for accessing GitLab

- Credential은 Add해줘야 한다. 아래 참고

### Add Credentials

Domain

Global credentials (unrestricted)

Kind

GitLab API token

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

API token

ID ?

Description ?

- GitLab API token을 설정 후,
- API token에 발급받은 Gitlab Access token을 입력 후
- ID는 본인이 관리하고 싶은 ID를 설정한다.
  - 예시 : Gitlab-Jenkins Personal Access token

## 위의 토큰은 Personal Access Token이다.

## Freestyle Project를 생성한다.

### 소스 코드 관리

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://gitlab-jenkins-personaltoken:6KzT3o

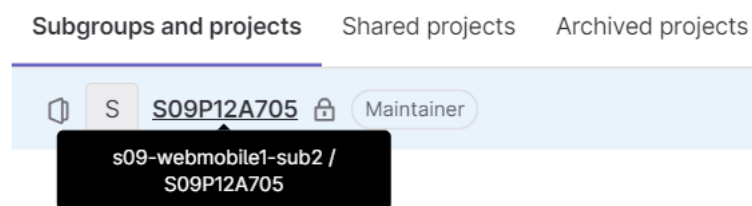
Credentials ?

- none -

Add ▾

고급 ▾

- Git이라 나와있는 부분에서 작성해야 한다.
- Repository URL 형식 (Gitlab 페이지 참고)
  - https://[token-id]:[token-value]@lab.ssafy.com/프로젝트/프로젝트세부명
  - s09-webmobile1-sub2/S09P12A705



- 위의 URL은 http 방식의 git을 clone해도 상관없지 않을까..? 테스트 해봐야 알 듯.
- 아래에 Build할 Branch를 설정한다.
- 아래 예시로 보면,
  - feature/nginx라는 branch에서 push를 하면 Jenkins가 감지하여 빌드하는 개념이라고 생각하면 된다.

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/feature/nginx

## 빌드 유발 설정

### 빌드 유발

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://i9a705.p.ssafy.io:9090/project/test ?

Enabled GitLab triggers

☒ Push Events ?

☐ Push Events in case of branch delete ?

☒ Opened Merge Request Events ?

☐ Build only if new commits were pushed to Merge Request ?

☒ Accepted Merge Request Events ?

☐ Closed Merge Request Events ?

Rebuild open Merge Requests ?

Never

☒ Approved Merge Requests (EE-only) ?

☒ Comments ?

Comment (regex) for triggering a build ?

Jenkins please retry a build

- Gitlab에 push가 일어날 시 WebHook을 걸기 때문에 위와 같이 설정한다.
- Comment 바로 아래에 고급이라는 부분을 눌러주자.

☒ Approved Merge Requests (EE-only) ?

☒ Comments ?

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update ?

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

46N4771204d81362a1e1103ae07c1d5a

- 맨 아래에 보이는 Secret Token은 Generate하여 Gitlab에서 Webhook할 때 넣어줘야 한다.
  - Generate는 계속 가능. 그러나 선택한 Secret Token과 Gitlab에 등록되는 것은 동일해야함
- Secret Token 값을 복사하여 Gitlab으로 간다.
- Gitlab → 프로젝트 선택 → 왼쪽 화면에서 Settings → Webhooks

s09-webmobile1-sub2 > S09P12A705 > Webhook Settings

Q Search page

### Webhooks

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**

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

☒ Show full URL

☐ Mask portions of URL  
Do not show sensitive data such as tokens in the UI.

**Secret token**

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

**Trigger**

☐ Push events

☐ Tag push events  
A new tag is pushed to the repository.

☐ Comments  
A comment is added to an issue or merge request.

☐ Confidential comments  
A comment is added to a confidential issue.

☐ Issues events  
An issue is created, updated, closed, or reopened.

☐ Confidential issues events  
A confidential issue is created, updated, closed, or reopened.

☐ Merge request events  
A merge request is created, updated, or merged.

☐ Job events  
A job's status changes.

☐ Pipeline events  
A pipeline's status changes.

☐ Wiki page events

- 위와 같은 화면이 나오는데 URL에는 jenkins의 경로를 적어준다.
  - 이 URL의 경로는 어디있나?



- 빌드 유발 부분에 보이는 **GitLab webhook URL** 이다.

#### 빌드 유발

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?  
☐ Build after other projects are built ?  
☐ Build periodically ?  
☒ Build when a change is pushed to GitLab. GitLab webhook URL: `http://i9a705.p.ssafy.io:9090/project/test` ?

- 복사해서 가져온 Secret Token도 위에 붙여준다.
- Trigger에서 Push events를 설정한다.
  - 모든 브랜치에서 푸시가 일어날 때마다 선택할 수도 있으나, 이러면 Push가 일어나기만 해도 빌드가 된다.
    - jenkins에서 감지하는 것이 모든 브랜치가 되는 것임.
  - 그래서
    - **Regular expression** 선택하고 감지할 브랜치 설정

#### Trigger

☒ Push events  
☐ All branches  
☐ Wildcard pattern  
☒ Regular expression

Regular expressions such as `^(feature|hotfix)/` are supported.

SSL verification  
☒ Enable SSL verification

Project Hooks (2)	
<a href="#">http://i9a705.p.ssafy.io:9090/project/test</a>	<input type="button" value="Test"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>
Push events Tag push events Comments Merge request events SSL Verification: enabled	

- Add Webhook을 누르면 위와 같은 Project Hooks가 생긴다.
- Test를 눌러 200을 반환받으면 Webhook이 성공적으로 이루어진다.

## 빌드를 위한 준비물

- 본인이 빌드할 환경에 맞는 **Tool**을 설정해야한다.
- SpringBoot를 gradle 형식으로 했다면 gradle이 필요하고,
- Maven으로 했다면 Maven이 필요하다.
- React라면 node 설정해주기
- 현재 환경
  - SpringBoot gradle 8.1.1
  - nodeJs 18.17.0
- 두 가지 모두 필요하기 때문에 모두 설치한다.

빌드 실행 상태 ▾

1 대기 중

2 대기 중

## System Configuration



### System

환경변수 및 경로 정보등을 설정합니다.



### Plugins

Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.



### Clouds

Add, remove, and configure cloud instances to provision agents on-demand.



### Tools

Configure tools, their locations and automatic installers.



### Nodes

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.



### Managed files

e.g. settings.xml for maven, central managed scripts, custom files, ...

위 화면에서 Tools로 들어가서 설치한다.

## JDK

### JDK installations

JDK installations ^

✎ Edited

Add JDK

JDK  
Name

openjdk

JAVA\_HOME

/opt/java/openjdk

☐ Install automatically ?

Add JDK


- jenkins에서 이미 jdk-17 버전을 가지고 있는 컨테이너를 만들었기 때문에 생략해도 무방하다.
- 그러나 jdk를 미리 깔지 않았다면 Jenkins에서 별도의 설치 작업을 해야한다.
- Jenkins에서 jdk가 필요한 이유는 Jenkins는 Java 기반이기 때문.
- 참고로 자동 설치의 경우 (install automatically)
  - Oracle JDK만 제공, 11이상 버전은 라이선스가 있어야 받을 수 있음.
- docker 환경의 jenkins에서 java— version을 쳐서 확인한다

```
root@68d3d571b4f8:/opt/java/openjdk# java --version
openjdk 17.0.7 2023-04-18
OpenJDK Runtime Environment Temurin-17.0.7+7 (build 17.0.7+7)
OpenJDK 64-Bit Server VM Temurin-17.0.7+7 (build 17.0.7+7, mixed mode)
```

## Gradle, NodeJS

- 본인이 Build할 프로젝트와 맞게 설정한다.
- 내가 진행한 환경은 Springboot Gradle 8.1.1과 Nodejs 18.7.0

### Gradle installations

Gradle installations ^  Edited

Add Gradle

Gradle name ?

Gradle 8.1.1

☒ Install automatically ?

≡ Install from Gradle.org

Version

Gradle 8.1.1

Add Installer ▾

Add Gradle

## NodeJS installations

NodeJS installations ^

✎ Edited

Add NodeJS

NodeJS

Name

nodejs 18

☒ Install automatically ?

≡ Install from nodejs.org

Version

NodeJS 18.7.0

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

☐ Force 32bit architecture

Global npm packages to install

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax `packageName@version`

Global npm packages refresh hours

Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache

72

Add Installer ▾

## 생성해둔 Freestyle 프로젝트 → 구성으로 들어간다.

### Build steps

- Gradle 프로젝트를 빌드할 것이기 때문에 설정해준다.
- Tasks에 clean build -x test를 입력한다.
  - 본인이 원하는 방식으로 바뀌도 무방하다.



FROM ChatGPT

1. **clean**: 이는 Gradle 빌드 프로세스에서 이전 빌드에서 생성된 모든 빌드 출력을 제거하고 "클린(clean)"한 상태로 시작하라는 것을 의미합니다. 이는 불필요한 파일이나 디렉토리가 남지 않도록 하기 위해 사용됩니다.
2. **build**: 이는 실제 빌드 작업을 수행하라는 것을 나타냅니다. 프로젝트 소스 코드를 컴파일하고 라이브러리를 생성하며 실행 가능한 파일을 생성하는 등의 작업이 이 단계에서 수행됩니다.
3. **x test**: 이는 빌드 작업을 수행할 때 "test" 태스크를 제외하라는 것을 의미합니다. "test" 태스크는 유닛 테스트를 실행하는데 사용되므로, 이 옵션을 사용하면 빌드 중에 테스트가 실행되지 않습니다. 이를 통해 빌드 시간을 단축할 수 있습니다. 만약에 빌드 이후에 테스트를 별도로 실행하려면 이 옵션을 사용할 수 있습니다.

따라서, "clean build -x test"는 Gradle을 사용하여 프로젝트를 클린(clean)하게 빌드(build)하되, 테스트는 실행하지 않는다는 것을 나타냅니다.

## Build Steps

≡ Invoke Gradle script ?

● Invoke Gradle ?

Gradle Version

Gradle 8.1.1

☐ Use Gradle Wrapper ?

Tasks ?

clean build -x test

- 바로 아래 고급을 누르면 아래와 같은 화면이 보인다.

고급 ^ Edited

Switches ?

System properties

☐ Pass all job parameters as System properties

Project properties

☐ Pass all job parameters as Project properties

Root Build script ?

/var/jenkins\_home/workspace/test/Sub-PJT2/backend

Build File ?

Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)

☐ Force GRADLE\_USER\_HOME to use workspace ?

- 위의 화면에서 Root Build script의 위치는,
  - Jenkins 내부에서 빌드할 프로젝트가 위치하는 곳.
  - 기본적으로 /var/jenkins\_home/workspace가 잡힌다.
- React

Execute NodeJS script ?

NodeJS Installation

Specify nodejs installation where npm installed packages to execute the script

nodejs 18

Script

See [the list of available environment variables](#) accessible by process.env.ENV\_VARIABLE.

npmrc file

- use system default -

Cache location

Default (~/.npm or %APP\_DATA%\npm-cache)

Execute shell ?

Command

See [the list of available environment variables](#)

```
cd /var/jenkins_home/workspace/test/Sub-PJT2/frontend

pwd
npm install
CI=false npm run build
```

- 여기까지 설정하면, Push가 일어났을 시 정상적으로 빌드가 되도록 설정한 것.

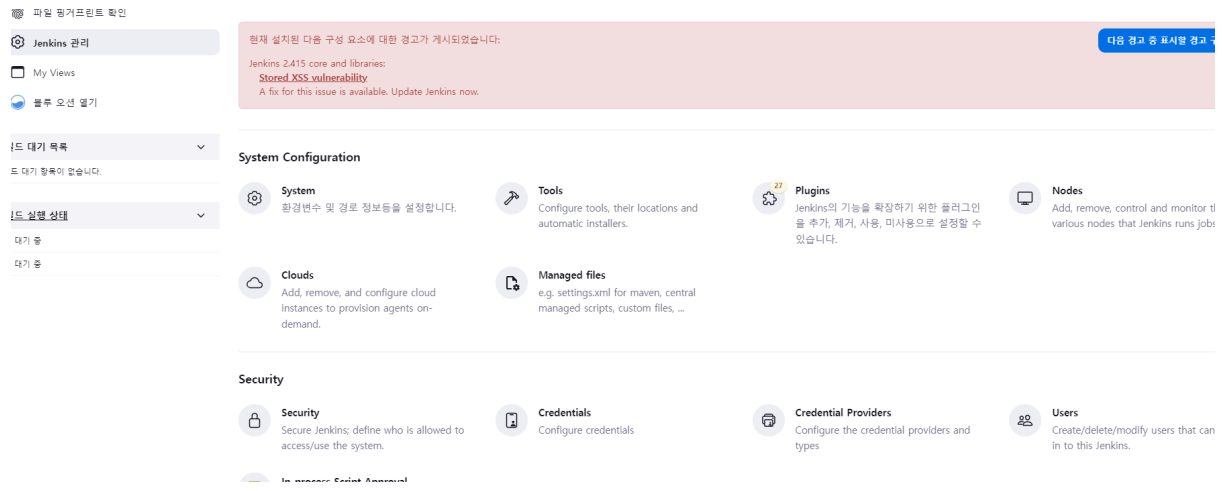


아래의 설정파일은 application.yml에 들어갈 정보들을 숨김 처리하기 위한 설정이다.

Credential을 Jenkins에 설정하여 Jenkins가 빌드 시 application.yml 내용 안에 {DB\_URL}, {DB\_PW} 등에 대해서 읽고 **치환**하도록 하는 방식을 사용한다.

## Jenkins의 Credentials 등록 방법

- Credentials로 들어간다.



- Add Credentials를 누른다.

**Global credentials (unrestricted)** [+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
<a href="#">gitlab-jenkins-personaltoken</a>	GitLab API token (gitlab-jenkins-personaltoken)	GitLab API token	gitlab-jenkins-personaltoken
<a href="#">DB_PW</a>	DB_PW	Secret text	DB_PW
<a href="#">DB_USER</a>	DB_USER	Secret text	DB_USER
<a href="#">S3_ACCESS</a>	S3_ACCESS	Secret text	S3_ACCESS
<a href="#">S3_SECRET</a>	S3_SECRET	Secret text	S3_SECRET
<a href="#">DB_URL</a>	DB_URL	Secret text	DB_URL
<a href="#">KAKAO_CLIENT</a>	KAKAO_CLIENT	Secret text	KAKAO_CLIENT
<a href="#">OPENVIDU_URL</a>	OPENVIDU_URL	Secret text	OPENVIDU_URL
<a href="#">OPENVIDU_SECRET</a>	OPENVIDU_SECRET	Secret text	OPENVIDU_SECRET

- script에서 text를 secret할 것이기 때문에 kind는 secret text로 설정하고
- secret에 본인이 실제로 넣을 값을 넣어주고 이름 설정하면 된다.
- Scope는 별도로 건드리지 않음.
- 예를 들어, Secret 값이 DB\_PW라고 하자.
  - DB\_PW는 실제로 1234라고 하면,
  - Secret에 1234가 들어가고
  - ID에 DB\_PW를 넣어준다.

### New credentials

Kind  
Secret text

Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

Secret

ID ?

Description ?

Create

## 추가한 크레덴셜을 빌드환경에서 설정

- 생성해둔 Freestyle project에서 구성을 들어간다.
- 빌드 환경에서 아래 버튼을 클릭 (Use secret ...)
- Bindings가 나오면 Add를 누르며 하나하나 추가해주면 된다.

### 빌드 환경

☐ Delete workspace before build starts

☒ Use secret text(s) or file(s) ?

#### Bindings

≡ Secret text ?

Variable ?  
DB\_URL

Credentials ?  
☒ Specific credentials ☐ Parameter expression  
DB\_URL

Add

≡ Secret text ?

Variable ?  
DB\_USER

Credentials ?  
☒ Specific credentials ☐ Parameter expression  
DB\_USER

Add

추가한 크레덴셜들을 Build steps의 '첫' 번째로 시작.



## Build Steps



- sed 명령어를 통해 치환할 문자열 → 등록된 문자열(Credential)로 바꾸는 과정.
- application.yml 파일이 위치하는 곳으로 이동 후
- sed 명령어를 통해 변경하고 싶은 문자열이 있는 곳에 크레덴셜 값을 주입
- Gitlab에는 {DB\_URL} 이런 식으로 올라가고
- 서버가 실제로 EC2에서 작동될 때는 정상 값들이 들어간 채로 되게 하는 것

```
echo ">application.yml 파일 위치로 이동"  
cd /var/jenkins_home/workspace/test/Sub-PJT2/backend/src/main/resources  
  
echo ">application.yml 정보 삽입"  
sed -i "s#\${DB_URL}#\${DB_URL}#" application.yml  
sed -i "s#\${DB_USER}#\${DB_USER}#" application.yml  
sed -i "s#\${DB_PW}#\${DB_PW}#" application.yml  
sed -i "s#\${S3_ACCESS}#\${S3_ACCESS}#" application.yml  
sed -i "s#\${S3_SECRET}#\${S3_SECRET}#" application.yml  
sed -i "s#\${KAKAO_CLIENT}#\${KAKAO_CLIENT}#" application.yml  
sed -i "s#\${OPENVIDU_URL}#\${OPENVIDU_URL}#" application.yml  
sed -i "s#\${OPENVIDU_SECRET}#\${OPENVIDU_SECRET}#" application.yml  
  
cat application.yml
```

## SED 명령어를 사용할 때 주의할 점

- sed 명령 실행 시 &를 만나는 경우
- &는 일치하는 문자열 자체를 의미하므로 \${DB\_URL} 이런 식으로 값이 들어가버림.
- 그래서 \& 이렇게 &이 출현하는 곳 바로 뒤에 \을 붙여서 무시할 수 있도록 한다.
- 아래는 Chatgpt

sed에서 &를 만나면 저게 자꾸 \${DB\_URL}로 바뀐다.

죄송합니다, 제 답변이 혼돈을 드린 것 같습니다. `sed` 명령어에서 `&`는 일치하는 문자열 자체를 나타내기 때문에 `\${DB\_URL}`로 대체되는 것이 맞습니다. 제가 제시한 방법은 옳바르지 않은 것이었습니다.

`\${DB\_URL}`을 원하는 URL 값으로 올바르게 대체하려면 다음과 같이 `sed` 명령어를 사용할 수 있습니다:

```
bash Copy code  
g?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8#" application.yml
```

이렇게 하면 `\${DB\_URL}`이 주어진 URL 값으로 올바르게 대체될 것입니다. 위 명령어를 실행하면 `application.yml` 파일의 `datasource` 섹션이 다음과 같이 변경될 것입니다:

```
yaml Copy code  
datasource:  
  url: jdbc:mysql://tagyou.site:3306/meeting?serverTimezone=UTC&useUnicode=t
```

위와 같이 수정한 뒤에 애플리케이션을 실행하거나 설정을 적용하면 `\${DB\_URL}`이 제대로 대체되어 올바른 설정이 적용될 것입니다.

## 배포

### 생성해둔 Freestyle Project에서 빌드 후 조치


- SSH 방식을 이용할 것이기 때문에
- Publish over SSH 플러그인을 설치해야한다.
- tools에서 install한다.
  - Gitlab 플러그인을 설치한 과정을 그대로 따라가서 Publish over SSH 플러그인을 설치해주면 된다.

### 설치 후 Jenkins 관리로 들어간다.

## Publish over SSH

Jenkins SSH Key ?

Passphrase ?

 Concealed

Path to key ?

Key ?

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAg4Mx/ASWBseh3bSsOTKHkNY+yUHHd6en0uqaE+7iMLBtY8Ew
dk2kJZzcgANTWSTg84dIYA/q+N2Z+KcuKJHzJSv19qqChjlvDSyCk4zYbwuwNmc2
b/jJSKEU75Q3PK4h0Q+qYzvByLSFR8nW96iwfqku8ecxpFkuuBKgcFlaEEaOX5X
ayfNU+0OmrCxYmp14eN5H3eHgwPmY+4Qp6faNQqQxSwJENfWQ5rqrfqJKsglTeYu
m5xYDjhptNiMIWHz//0sLmC8H2aMzWnMjVfVxAjnhbMdatKCCn+b+F/9IT9EG7v
NtkExwyDUIB5+6neeYeCgJmm+GHEXn54ZTToQIDAQABAoIBAG9XWrEwPMg+qbbn
aMwAELA6AuwnkbYPFIHGV2461/s601IaVGpEb418kv+cJEubomvrmJNlxrmt+9Z9
y+GDN2SoJFSSpXD4HSH9Jgu/udnB/o0AsyD3eykMX4o6ZM2V64FxDuN2yyieVRyj
MMF6avrBR8J75xZqfrlLFGloj+9DvIEt9IP1z9wjOP32qw011SsSn8Cs4uRYdaYX
sAls3qtYp5xSGAdH9WlqHiaLH+vSw2PWEnRpXE02MKGFFx/3NZYsiX2MRYa3nYlg
keOLLSNiyUwNr7re4GVmB+foyD0Kq/7/K4i3qShf8pN8aPxsTbWn2ZNRrqVuVtN
l5sO+cECgYEA1jVAMcfYdrMRyUNaOnMWQGqZXmOV8R9h6MTEGs5bhCSCZKUMKkgS
wbPrPQtuWsbMTkchar5i9lPw5gdrOnu9V0zYcjCtMteCw0S4aRk5o28HGQja8CKa
3LG+BfZ2mjf+D7R1ifHjBx5Fq+LMIHn6FIL4OV6WoFsSrqiITbHD3SkCgYEAAnSuq
gXtvtNM8iQd7jiqNj97Q8yxdrOPGRO7Lb4leQ9XaUhh/btnPpa6VqQphQEdhlHgr8
+FOh7RwWF7fErI4whPrGMvndvRmncrOa9C+7+OVY5en99vkzXhevulIHlwMFIHnfnN
```

- 내리다 보면 Publish over SSH라는 화면이 보이게 된다.
- Key에는 EC2에 접속하는데 사용하는 .pem 키를 복사 붙여넣기

### SSH Servers

SSH Server

Name ?

docker-server

Hostname ?

i9a705.p.ssafy.io

Username ?

root

Remote Directory ?

/

☐ Avoid sending files that have not changed ?

고급 ▾

Test Configuration

- 추가 버튼을 누르면, SSH Servers가 나온다.
- Name은 마음대로 설정해도 된다.
- Hostname은 domain 이름을 설정해준다.
  - ssh hostname
- Username같은 경우, 필자는 root로 접속하기 때문에 root
  - ssh username
- Remote Directory는, 전송이 되었을 때 EC2의 시작 경로이다.

## 백엔드 관련 배포

- 구성한 Freestyle Project로 들어간다.

### 빌드 후 조치

- 위와 같이 설정할 것이다. 따라하기 전에 설명부터 봐야 한다.
- SSH Server : 실제로 전송이 될 EC2 이름을 설정한다.
  - Publish over SSH에서 지정한 Server name을 적어준다.
- Source files는 jar파일만 서버에 보낼것이다.
- Remove prefix에서 jar파일 이전 경로를 다 지워준다.
- Remote directory는 실제로 서버에 전송했을 때 저장이 되는 경로이다.

- 필자같은 경우는 /root에 jar파일이 저장된다.

```
root@ip-172-26-11-118:~# pwd
/root
root@ip-172-26-11-118:~# ls
Dockerfile  project-0.0.1-SNAPSHOT.jar
```

Remote directory ?

Exec command ?

```
docker rm -f BS;
docker build -t project:latest -f Dockerfile .;
docker run -d --rm -p 9999:9999 --name BS --network springboot-
mysql-net project:latest
```

All of the transfer fields (except for Exec timeout) support substitution of [Jenkins environment variables](#)

고급 ▼

- Exec command는 전송 후 실행하는 명령어라고 생각하면 된다.

```
docker rm -f BS;
docker build -t project:latest -f Dockerfile .;
docker run -d --rm -p 9999:9999 --name BS --network springboot-mysql-net project:latest

# 존재하는 BS (백엔드 서버)를 지우고
# root 경로에 있는 Dockerfile에서 project 이미지를 최신화하고
# BS 서버를 다시 로드한다.

# 계속해서 BS 서버를 지워주는 것은 최신으로 바꾸고 다시 구동시키기 위함.
```

## 프론트 관련 배포

- 빌드 후 조치 추가를 눌러 아래와 같이 작성

SSH Server

Name ?

docker-server

고급

Transfers

Transfer Set

Source files ?

Sub-PJT2/frontend/build/\*\*

Remove prefix ?

Sub-PJT2/frontend

Remote directory ?

/usr/share/nginx

Exec command ?

All of the transfer fields (except for Exec timeout) support substitution of

- 이 경우는 별도로 Exec command 하지 않았다.
- 프론트 파일 역시 별도의 컨테이너화 해서 관리하고 싶다면
  - Exec command가 백엔드처럼 진행이 되어야하고
  - Dockerfile을 React 파일에 넣어주던, React 파일이 저장되는 위치에 넣어두던 설정해주어야 한다.
- 현재 본인 서버에는 nginx가 컨테이너화 되어있지 않고 바로 존재함.
  - 이 nginx가 바라보고 있는 정적 파일의 경로가
  - /usr/share/nginx임.
- 그래서 react 파일을 바로 /usr/share/nginx에 전송시켜준다.
- 이후의 프록시 설정은 nginx에서 한다.

## nginx 설정 및 SSL 적용

- nginx 관련 SSL을 적용하는 방식은 ubuntu 버전에 따라 다를 수 있다.
- 아래 블로그를 보고 참고했다.

### Let's Encrypt 인증서로 NGINX SSL 설정하기

이 모범 사례에서는 Let's Encrypt 클라이언트를 사용하여 인증서를 생성하는 방법과 이를 사용하도록 NGINX 오픈소스 및 NGINX Plus를 사용하여 nginx ssl 설정을 자동으로 구성하는 방법을 다룹니다.

 <https://nginxstore.com/blog/nginx/lets-encrypt-인증서로-nginx-ssl-설정하기/>



### • SSL 적용이 된 nginx 구성파일임

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    if ($host = www.tagyou.site) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    if ($host = tagyou.site) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    server_name tagyou.site www.tagyou.site;
    return 404; # managed by Certbot
}

server {
    location / {
        root /usr/share/nginx/build;
        index index.html index.htm;
        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;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://localhost:9999;
        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;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/tagyou.site/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/tagyou.site/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_name tagyou.site www.tagyou.site; #managed by Certbot
}
```

## location /

- root가 /usr/share/nginx/build로 되어있다.
  - nginx가 맨 처음 바라보는 곳이 root → /usr/share/nginx/build이다.
  - 아무런 설정을 하지 않은 경우, 홈페이지 화면에서 **Welcome to nginx!** 등의 모습이 보인다.
  - root가 바라보는 곳이 nginx가 기본적으로 설정한 index.html로 되어있기 때문.
  - 우리는 root를 react파일을 배포한 위치로 설정해주어야 한다.
- 이전에 react파일의 빌드 결과물을 /usr/share/nginx로 전송했다.
- 그러므로, /usr/share/nginx/build를 바라보게 하고

- 그 안에 있는 `index.html`을 바라보게 설정한 것 → `index index.html index.htm;`
- `try_files $uri $uri/ /index.html;`
  - 이거 빼면 `api` 통신 못한다.

## location /api

- `proxy_pass http://localhost:9999;`
  - `/api`로 오는 요청에 대해 `localhost:9999`, 즉 BS 컨테이너로 요청을 보내주는 것임.

## 2. 프로젝트에서 사용하는 외부 서비스 정보를 정리한 문서

### KAKAO 로그인

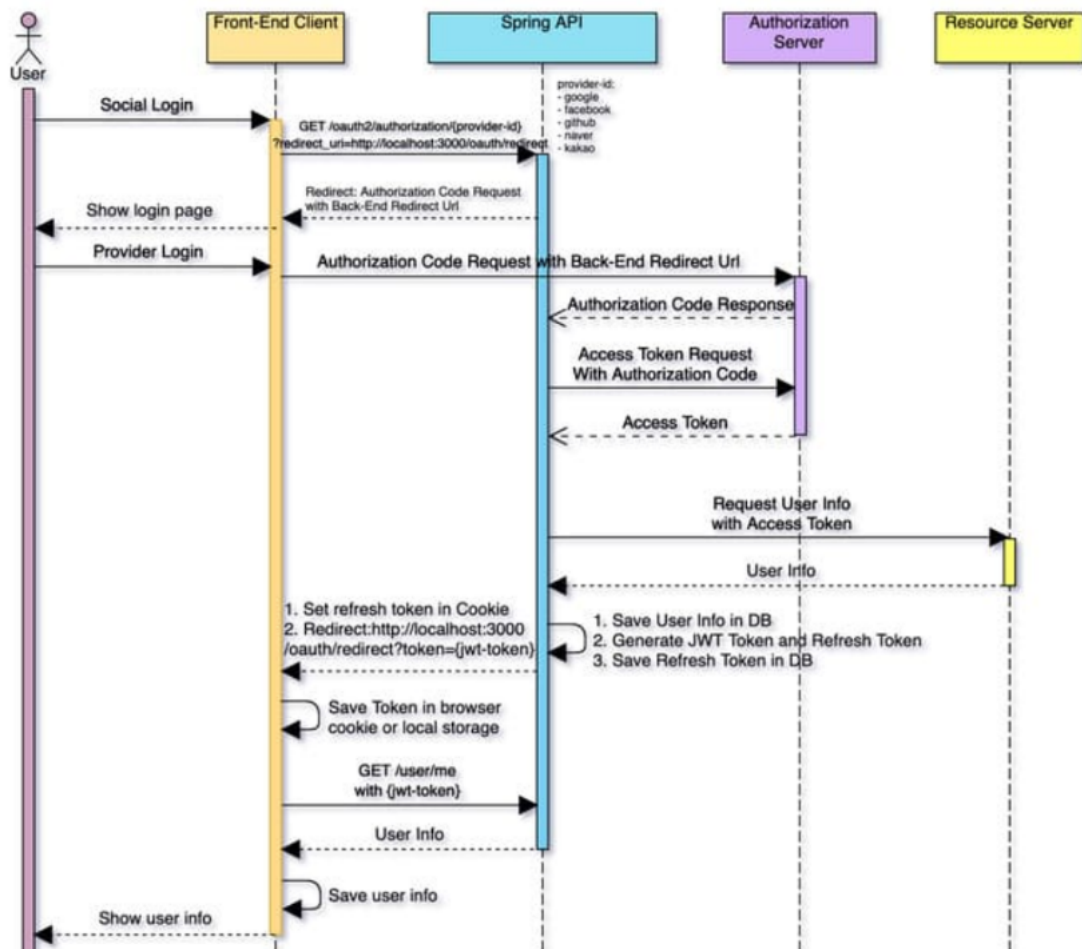
#### Kakao Developers

카카오 API를 활용하여 다양한 어플리케이션을 개발해보세요. 카카오 로그인, 메시지 보내기, 친구 API, 인공지능 API 등을 제공합니다.

<https://developers.kakao.com/docs/latest/ko/kakaologin/rest-api>

kakao developers

- 로그인 로직





- application.yml

```
spring:
  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: ${KAKAO_CLIENT} # 카카오 REST_API KEY가 들어가야 하는 곳
            redirect-uri: https://tagyou.site/api/login/oauth2/code/kakao
            # redirect-uri는 카카오 홈페이지에서 등록해두어야함
            authorization-grant-type: authorization_code
            client-authentication-method: client_secret_post
            client-name: Kakao
            scope: profile_nickname, profile_image, account_email

        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id
```

- 카카오 REST API Key

## 앱 키

네이티브 앱 키	4f63732517e1042a9111
REST API 키	fbb08611d97dc25ae84

- 위의 REST API 키를 application.yml에 등록해줘야한다.
- redirect-uri : 우리 서버를 통해서 kakao server로 uri를 redirect 시켜야 한다.
  - 해당 redirect uri는 카카오 홈페이지에서 등록해놓아야 한다.

## Redirect URI

삭제

수정

Redirect URI	
	http://localhost:9999/api/login/oauth2/code/kakao
	http://localhost:9999/login/oauth2/code/kakao
	http://tagyou.site/login/oauth2/code/kakao
	http://tagyou.site/api/login/oauth2/code/kakao
	https://tagyou.site/api/login/oauth2/code/kakao

- 서비스에서 필수적으로 사용해야 하는 것은 닉네임, 프로필 사진이고 이메일은 default

## 개인정보

항목 이름	ID	상태	
닉네임	profile_nickname	● 필수 동의	설정
프로필 사진	profile_image	● 필수 동의	설정
카카오계정(이메일)	account_email	● 선택 동의 [수집]	설정

### 3. DB 덤프 파일 최신본

- 덤프 파일 없음.

### 4. 시연 시나리오

- tagyou.site에 접속하여 카카오 로그인 진행
- 로그인 진행 후 처음 맞이하는 화면 : 추가 정보 입력
- 입력 이후 메인화면
  - 현재 내 정보를 볼 수 있는 UI가 떠있다.
  - 좌측 상단의 달 아이콘을 클릭하면 UI의 색깔이 변경된다.
    - 수정이 가능하다.
  - 좌측 하단 미팅
    - 일대일 매칭
    - 다대다 매칭
  - 우측 바에 친구 목록
    - 친구 검색
      - 친구 검색 결과 사용자가 존재하는 경우 친구추가 가능
    - 친구 추가

### 매칭

- 매칭은 내가 들어가 있는 어떤 랜덤방에 사람이 가득 찰 때까지 대기한다. (대기중 이란 화면이 계속 떠있는 상태)
- 매칭완료 시
  - 게임 방 이동 후 게임에 참여한 사용자의 모습이 보인다
  - 게임 방 내에서 채팅이 가능하다.
  - 미니게임을 눌러 실행하면 게임이 된다.
    - 아직 미니게임이 될지 안될지 모르겠다.