METASSAFY Backend 배포 매뉴얼

본 문서는 스프링 부트로 개발된 metassafy 를 사용하기 위한 가이드로 백엔드 스프링 서버를 배포하는 과정에 대해 서술하고 있습니다

# 스프링 개발

1.1 개발 환경

IntelliJ IDEA : 2022.3 버전 사용

Spring boot : 2.7.7 버전 사용

Java : 11버전 사용

1.2 사용 라이브러리

**SockJS**

프론트엔드에서 통신을 주고 받을 때 사용

Spring boot에서 네이버 SockJS를 사용하기 위한 pom.xml 설정

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-websocket</artifactId>  
</dependency>

**MySQL**

백엔드를 MySQL 데이터베이스와 연동하기 위해서 필요한 라이브러리

Spring boot에서 네이버 MySQL를 사용하기 위한 pom.xml 설정

<dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
</dependency>

MySQL 데이터베이스를 이용하기 위한 변수

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://i8d211.p.ssafy.io:3305/metassafy?useUniCode=yes&characterEncoding=UTF-8&serverTimezone=Asia/Seoul&allowMultiQueries=true  
spring.datasource.username=ssafy  
spring.datasource.password=P@ssw0rd

**Mybatis**

Mybatis는 자바 오브젝트와 SQL사이의 자동 매핑 기능을 지원하는 ORM 프레임워크

Spring boot에서 네이버 Mybatis를 사용하기 위한 pom.xml 설정

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>2.3.0</version>  
</dependency>

Mybatis를 이용하기 위한 변수

mybatis.mapper-locations=classpath:mappers/\*\*/\*.xml

**네이버 클라우드 서비스**

네이버 클라우드 회원가입

Spring boot에서 네이버 클라우드 서비스를 사용하기 위한 pom.xml 설정

<dependency>  
 <groupId>com.amazonaws</groupId>  
 <artifactId>aws-java-sdk-s3</artifactId>  
 <version>1.11.238</version>  
 </dependency>

클라우드 서비스를 이용하기 위해 필요한 변수

naver.cloud.endpoint=https://kr.object.ncloudstorage.com  
naver.cloud.region.name=kr-standard  
naver.cloud.bucket.name=metassafy  
naver.cloud.access.key=ZTVuL0QGNF5Po0x6lr5U  
naver.cloud.secret.key=GgMLSB18SgVeVOAgeyZw9QqQcJReZNnl2fUpL7vM

**OpenVidu**

웹 또는 모바일 애플리케이션에서 화상 통화를 쉽게 추가할 수 있는 플랫폼

OpenVidu 내부에 있는 nginx를 통해서 리버스 프록시 역할 수행 및 서버 내부 경로 설정

Spring boot에서 OpenVidu 서비스를 사용하기 위한 pom.xml 설정

<dependency>  
 <groupId>io.openvidu</groupId>  
 <artifactId>openvidu-java-client</artifactId>  
 <version>2.25.0</version>  
 </dependency>

OpenVidu 서비스를 이용하기 위해 필요한 변수

server.ssl.enabled=false  
OPENVIDU\_URL=http://i8d211.p.ssafy.io:5443  
OPENVIDU\_SECRET=ssafy

**JWT**

JWT토큰을 사용하기 위한 라이브러리

Spring boot에서 jwt를 사용하기 위한 pom.xml 설정

<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.1</version>  
</dependency>

# 백엔드 배포 과정

## **배포 환경**

배포 서버 : Ubuntu 20.04.5 LTS

Docker : 20.10.12

Maven : Apache Maven 3.8.7

Tomcat : tomcat 9.0 사용

Openvidu : Openvidu 2.25.0

Nginx : Openvidu 자체 nginx

## **도커 설치**

EC2 환경(Ubuntu)에서 도커 최신 버전 설치

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugi

## **OpenVidu 설치**

Docker-compose 다운로드

DOCKER\_CONFIG=${DOCKER\_CONFIG:-$HOME/.docker}

mkdir -p $DOCKER\_CONFIG/cli-plugins

curl -SL https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-linux-x86\_64 -o $DOCKER\_CONFIG/cli-plugins/docker-compose

바이너리 실행권한 적용

chmod +x $DOCKER\_CONFIG/cli-plugins/docker-compose

설치 테스트

docker compose version

포트 개방

OpenVidu에서 사용하는 port 개방

**22 TCP**

**80 TCP**

**443 TCP**

**3478 TCP+UDP**

**5442 TCP+UDP**

**5443 TCP+UDP**

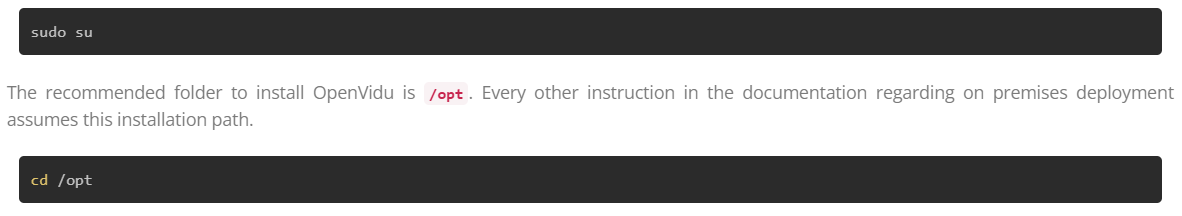
**6379 TCP+UDP**

**8888 TCP+UDP**

**40000:57000 TCP+UDP**

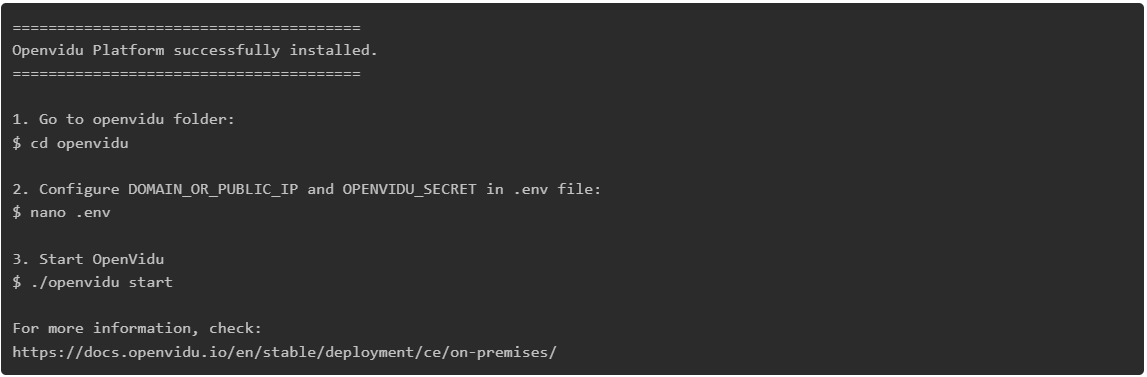
**57001:65535 TCP+UDP**

OpenVidu를 설치 하기 위해서 관리자로 전환



아래 명령어 실행 시 Openvidu가 설치되고 아래 가이드에 따라 진행

****



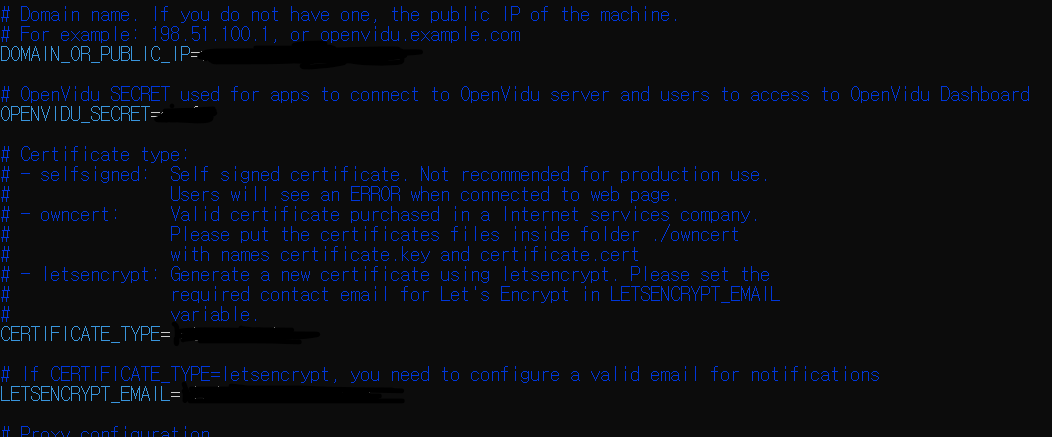
vi .env 명령어 실행 시 아래의 4개의 영역에 값을 입력

DOMAIN\_OR\_PUBLIC\_IP= 나의 도메인 이름

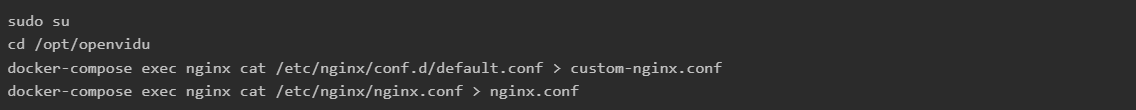
OPENVIDU\_SECRET=아무 비밀번호

CERTIFICATE\_TYPE=letsencrypt

LETSENCRYPT\_EMAIL=자기 이메일 주소



OpenVidu를 실행시킬 때마다 컨테이너가 새롭게 올라가서 기존에 설정했던 nginx 설정 값들이 날아가기 때문에 밖에 파일(custom-nginx.conf)을 컨테이너의 기본 설정값으로 연동



docker-compose.yml nginx 부분에 추가



시작 or 재시작을 하면 custom-nginx.conf 설정값으로 nginx 작동



## **Jenkins 설치**

도커 안에 Jenkins 컨테이너 설치

docker run -d -v jenkins\_home:/var/jenkins\_home -p 8080:8080 -p 50000:50000 --restart=on-failure --name jenkins-server jenkins/jenkins:lts-jdk11

1. Jenkins container의 아이디 확인

docker ps

1. Jenkins container log에 찍혀있는 Serial number값 복사

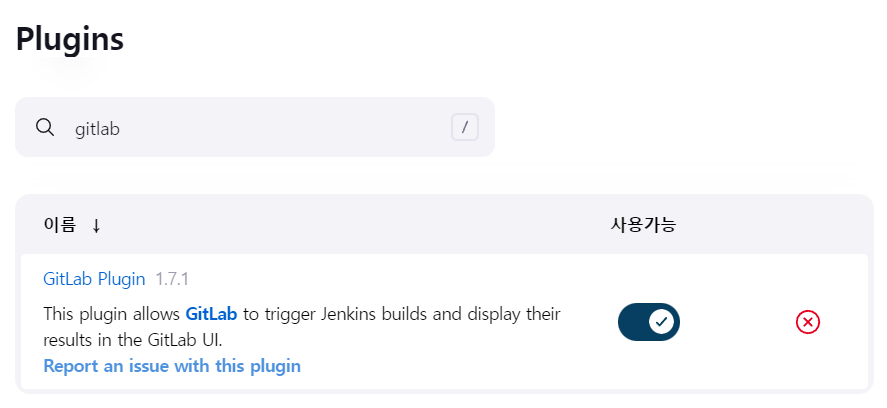
docker logs Jenkins\_id

1. 도메인<URL:8080>으로 접속해서 key값 입력
2. 기본 설치 및 계정 생성

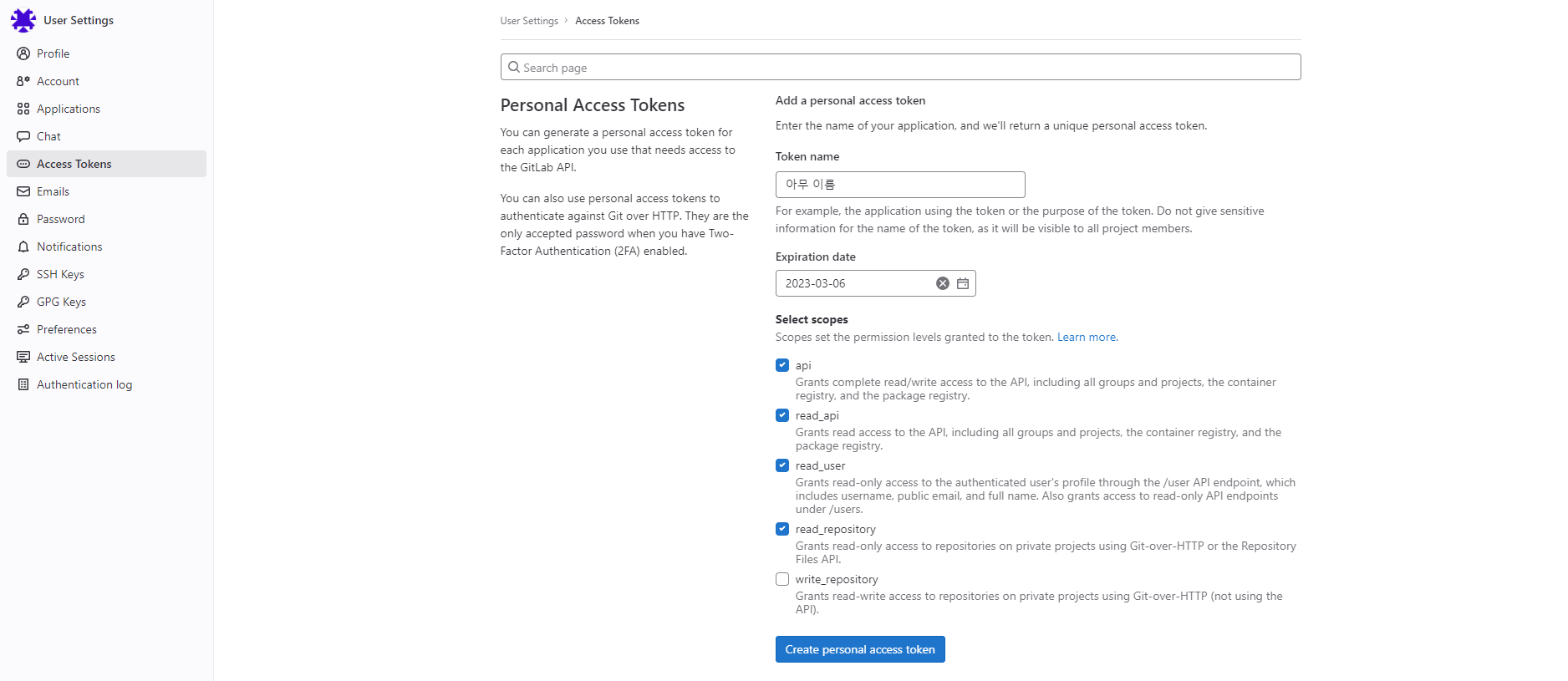
## **GitLab과 연동**

### GitLab Plugin 설치

Dashboard > Jenkins 관리 > Available plugins 에서 GitLab Plugin 설치

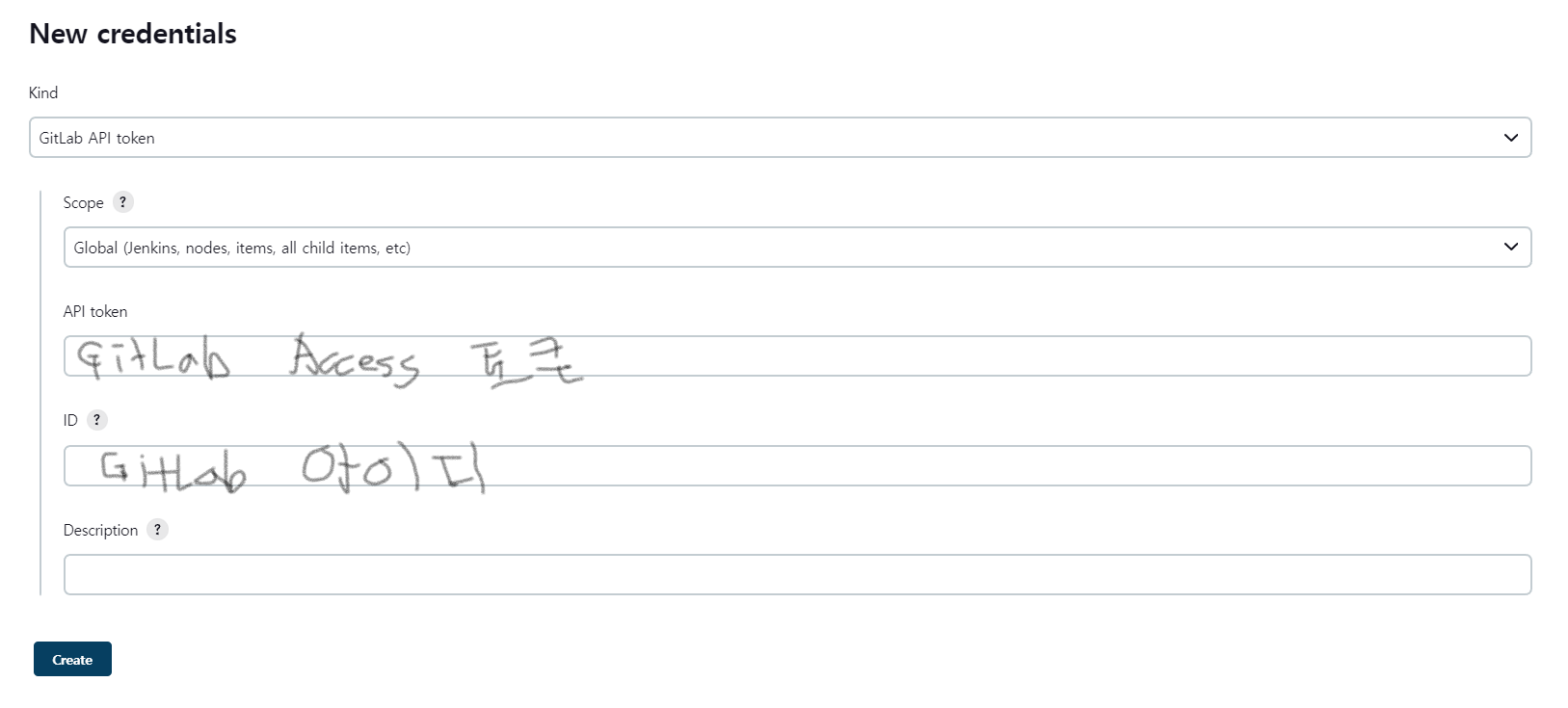
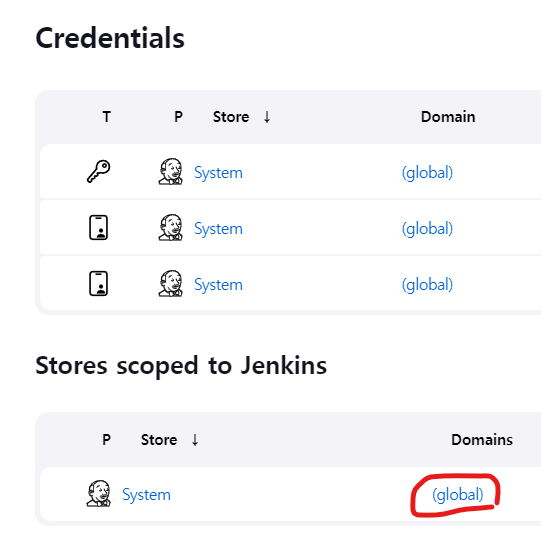
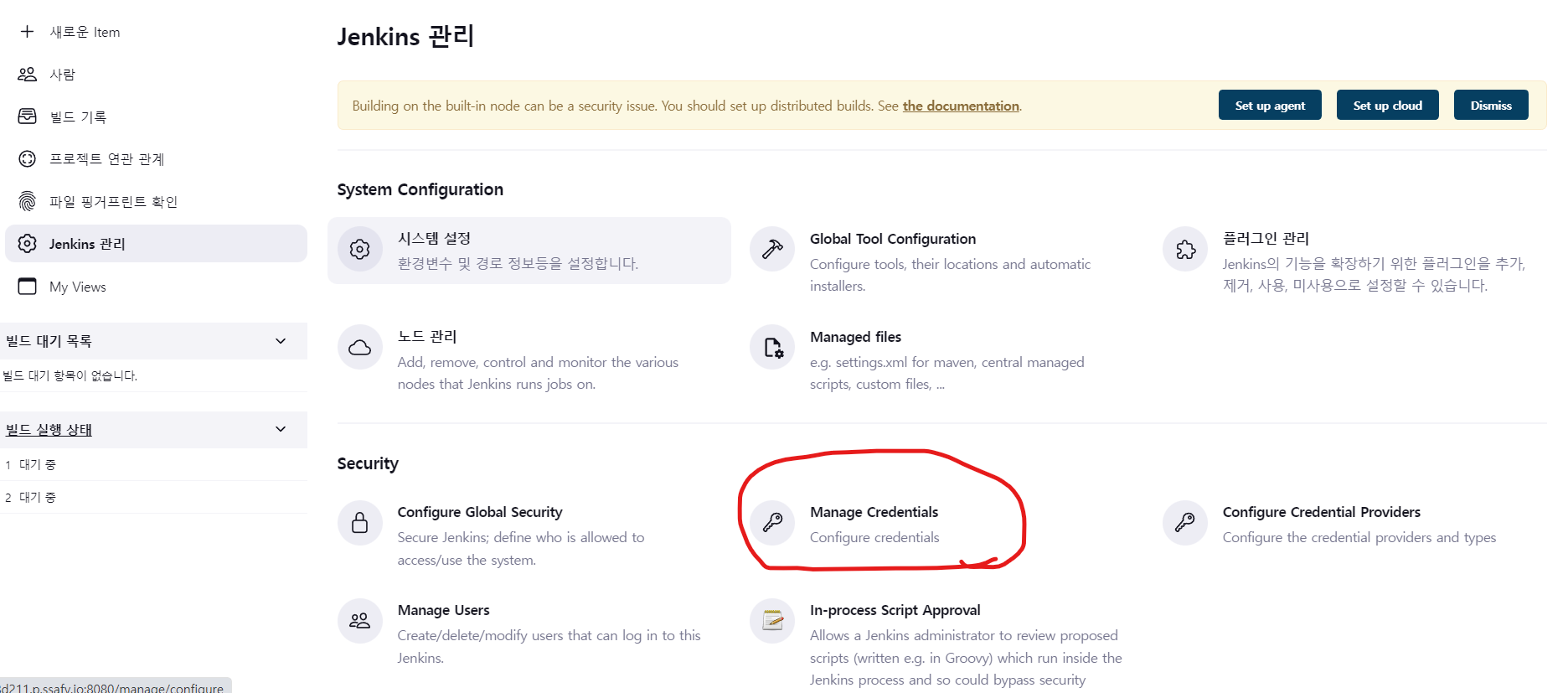


### GitLab Access Token 생성



* GitLab에 Access Token 으로 이동
* Token name은 아무거나 입력
* Expiration date는 기존에 적혀있는걸 사용하거나 지정하지 않고 사용 가능(지정 x 사용)
* api, read\_api, read\_user, read\_repository 체크 후 생성

### Credentials 생성

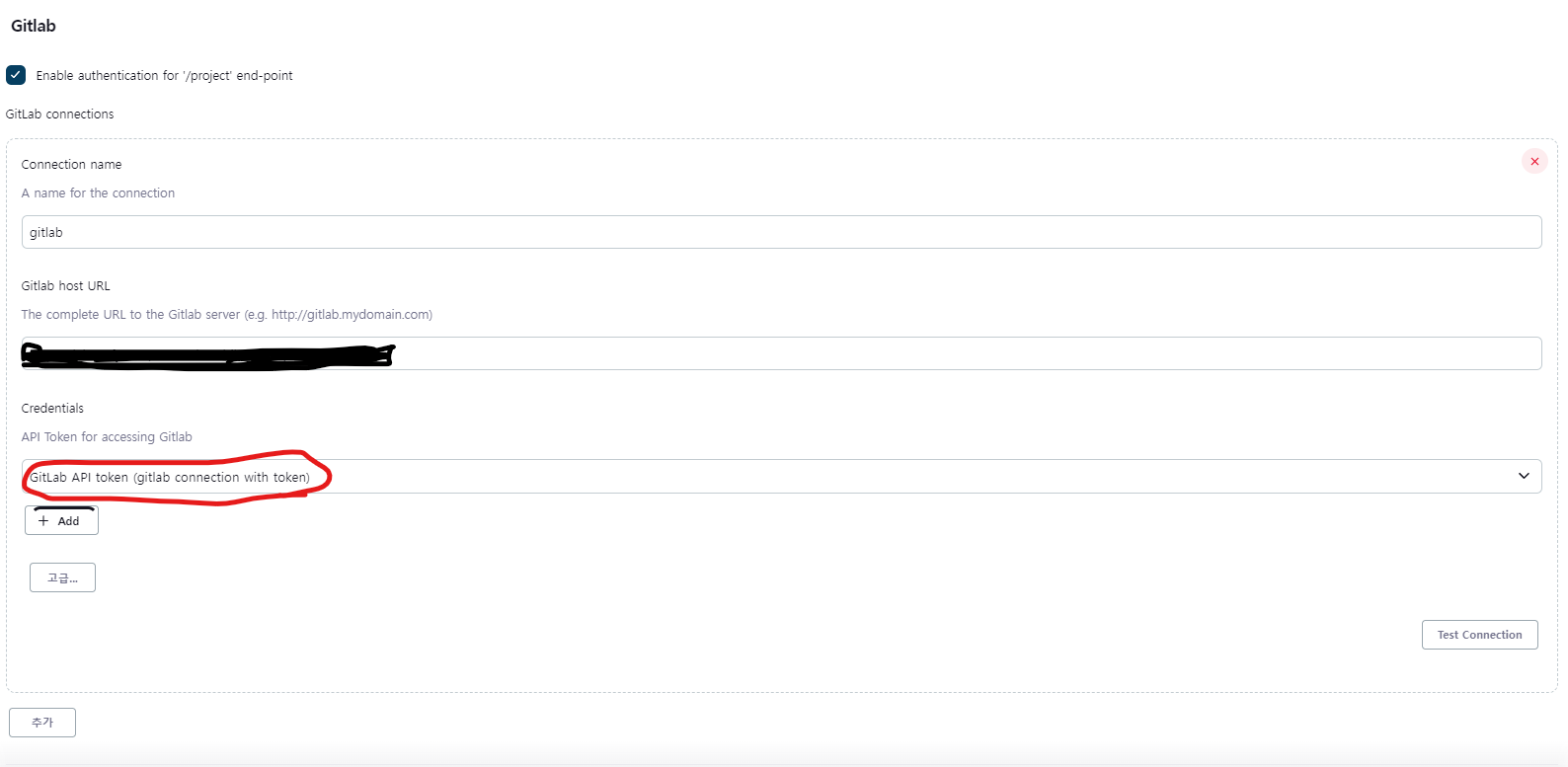


Dashboard > Jenkins 관리 > Configure System에서 gitlab 연결

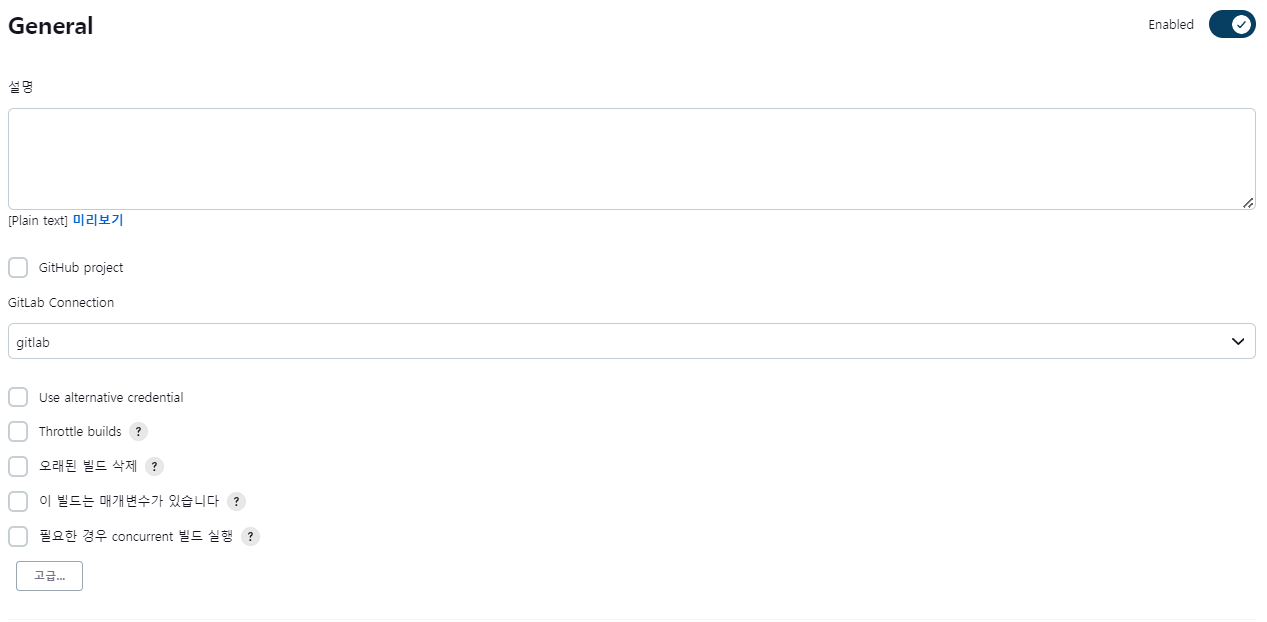
Connection name은 아무거나

Gitlab host URL은 연결할 GitLab URL

Credentials는 위에서 만든 Credential을 연결



GitLab Connection에서 위에 생성한 connection 연결



Repository URL에 GitLab URL 기입

Credentials에 생성한 Credential 적용

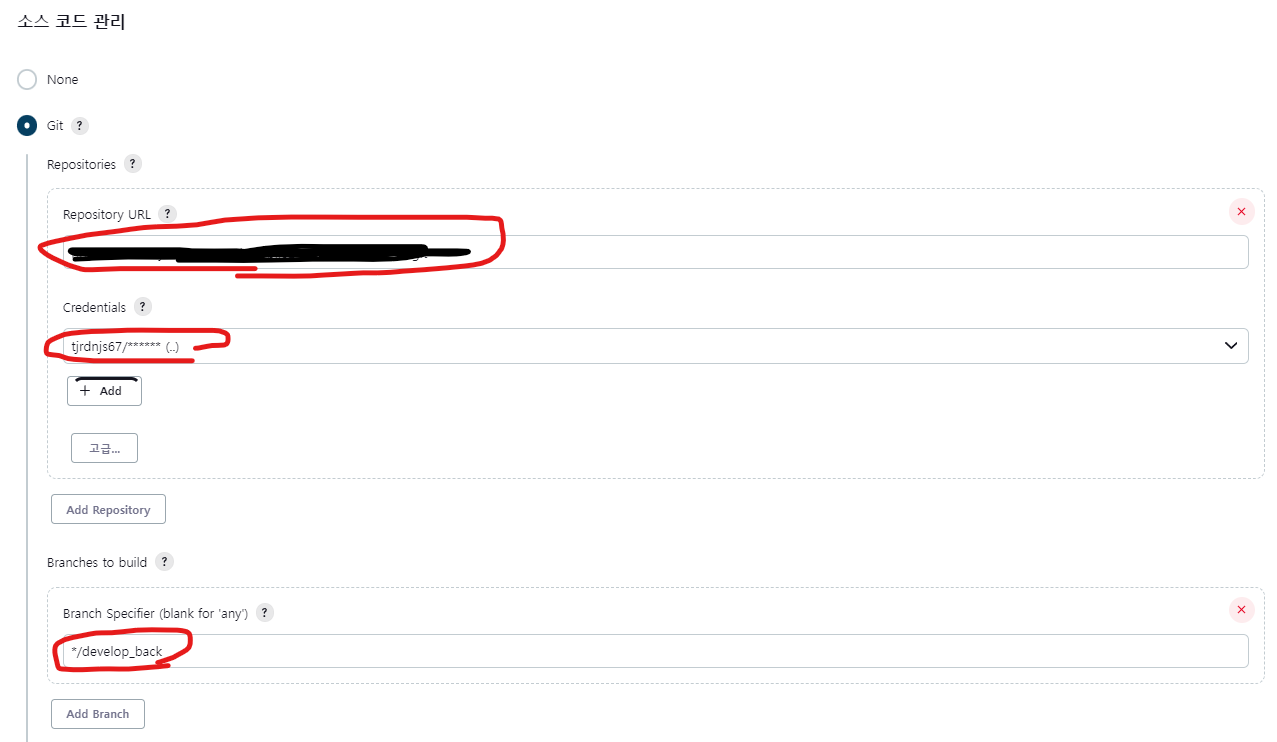
+ Add 클릭

Jenkins 클릭



* Username에 GitLab 아이디, Password에 GitLab 비밀번호 입력
* ID는 Username과 같게 하거나 아무거나

Branch Specifier에는 연동할 git branch 기입



## **Jenkins 사용법**

Item이란, jenkins에서 사용하는 작업의 최소단위

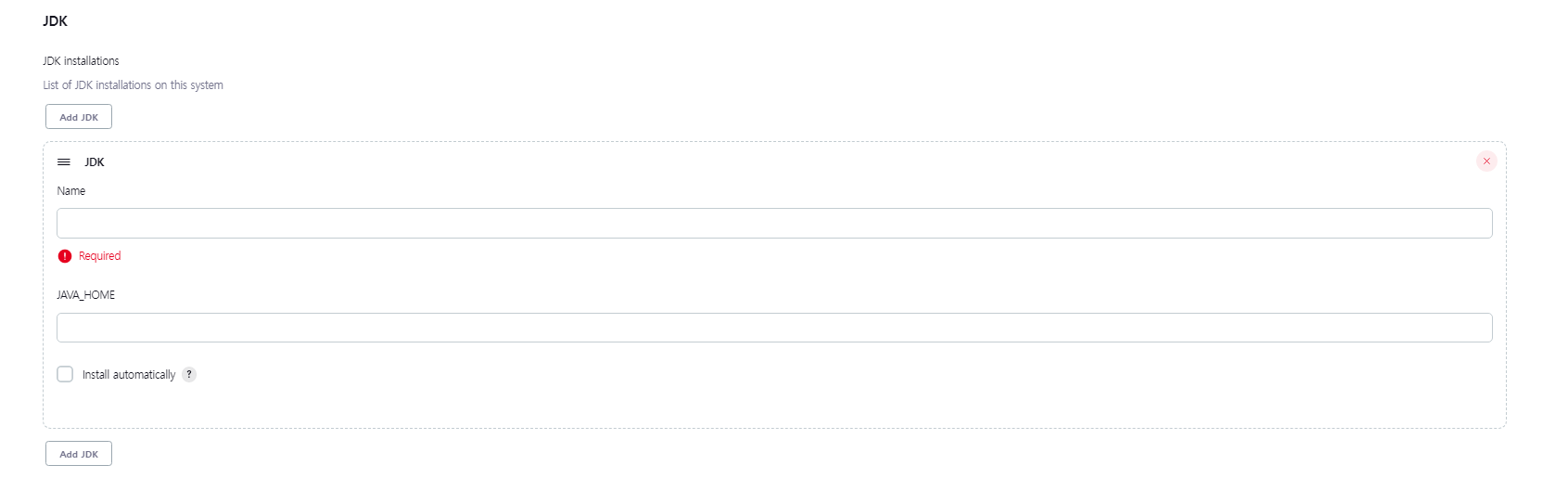
jenkins 관리를 통해 jdk, maven 설정

### jenkins 관리

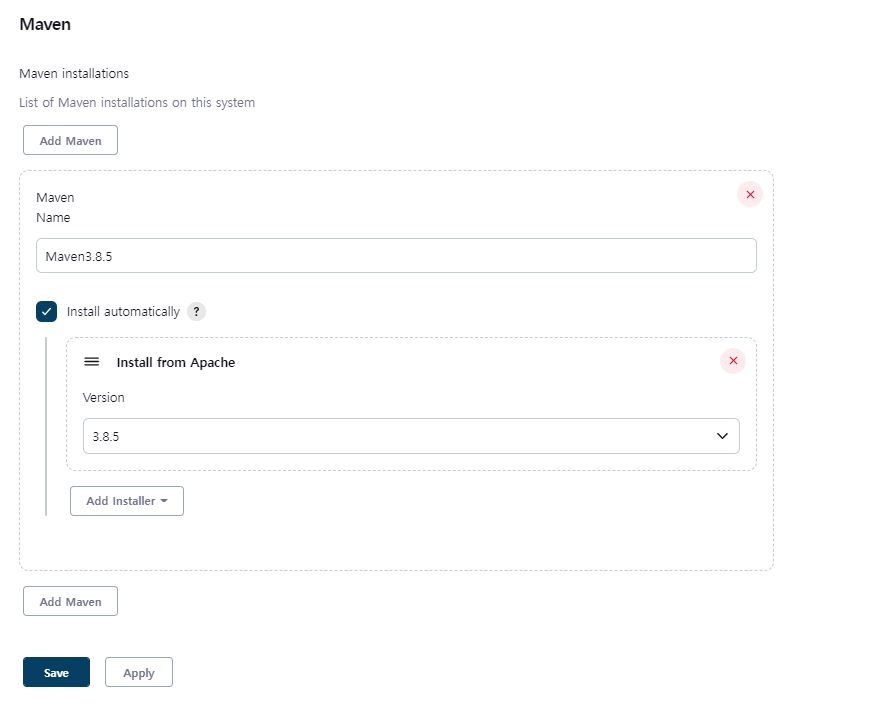


jenkins에 자바를 설치해주는 목록

* docker에서 jenkins를 java11로 설치해주었기 때문에 따로 설정을 할 필요는 없다.
* docker desktop → jenkins 클릭 → terminal 접속 → echo $JAVA\_HOME



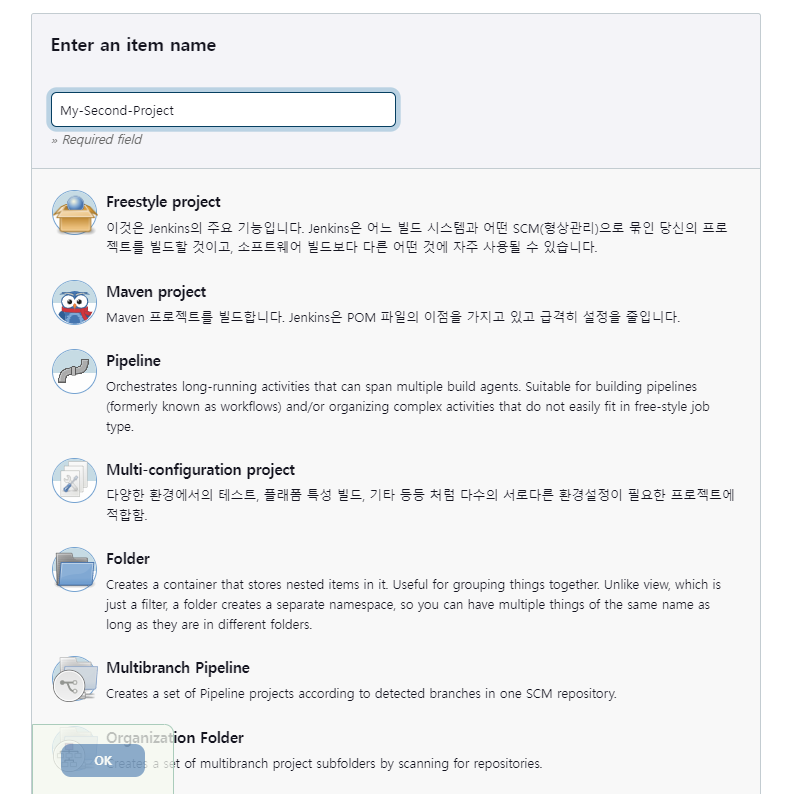
* echo를 통해서 나온 path를 JAVA\_HOME 빈칸에 넣어준다.
* 이름은 아무거나



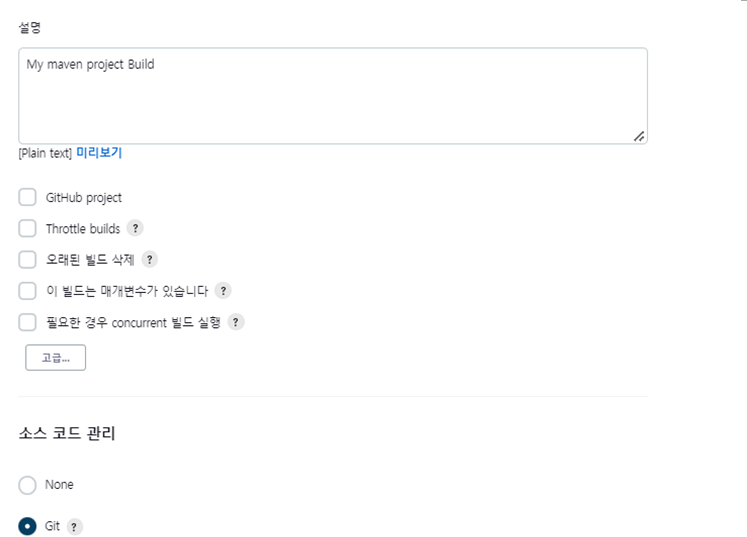
Maven과 연동할 수 있게 해주는 목록

* Maven은 처음 Pulg-In에 포함된 프로그램이 아니므로, 직접 Plug-In을 설치해주어야 한다.
* 플러그 인 관리 → Available plugins → maven intergration 선택 → install without restart

### 프로젝트 생성

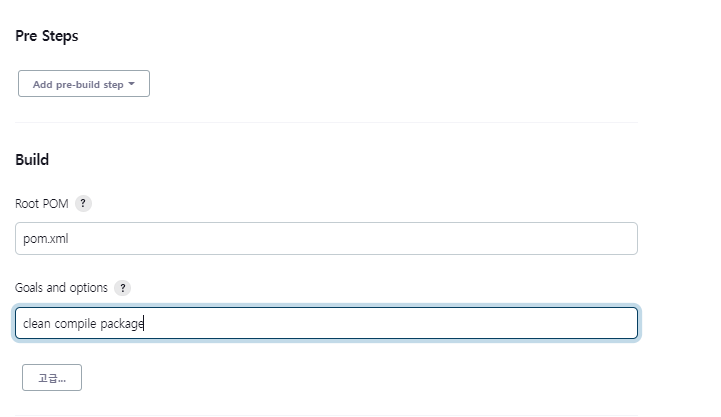


Maven 플러그인을 설치하였으므로 Maven을 통해 Project를 생성해준다.



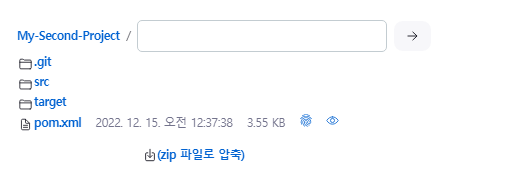
프로젝트의 간단한 설명과 연동할 깃허브 주소를 입력한다.

* private 환경의 git 레퍼지토리라면 계정을 추가해야한다.



* clean : build를 한게 있으면 지우고 새로 만들겠다는 뜻
* compile : 빌드하겠다.
* pakage : 컴파일 되어진 내용을 통해 poem.xml을 가지고 package를 만들겠다는 의미(jar, war)

지금 빌드 후 → 작업공간 확인

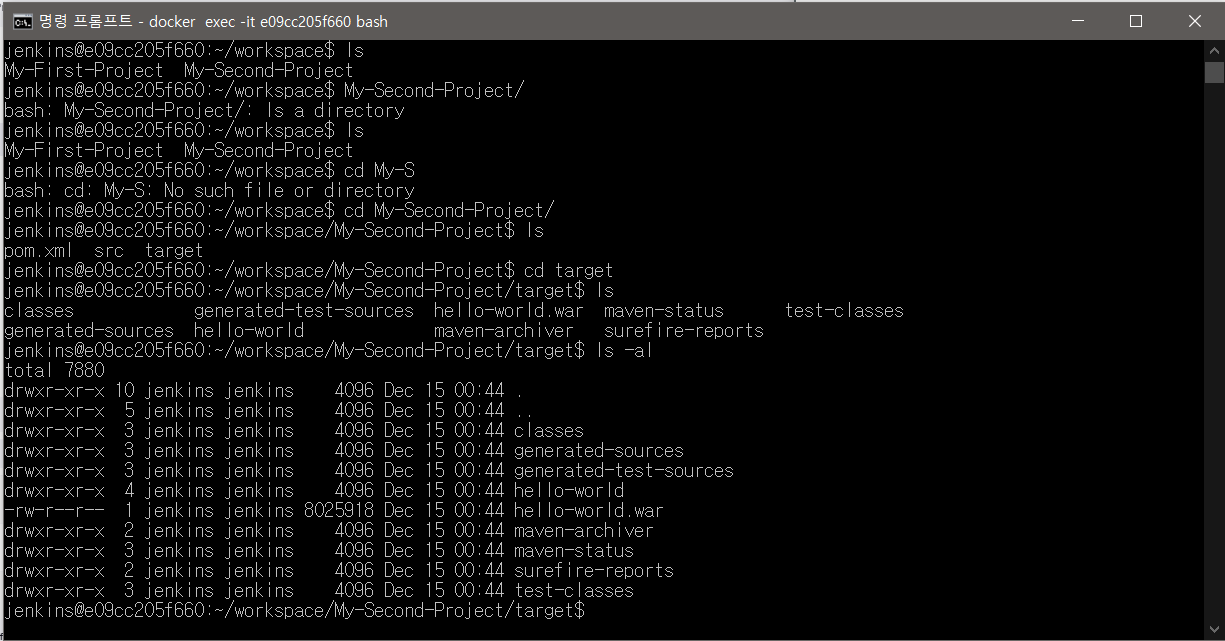


→

docker container exec -it [container\_id or container\_name] bash

를 통해 container에 접속하여 jenkins\_home을 확인한다.

→



War 파일이 존재하는 것을 볼 수 있다.

### 플러그인 설치

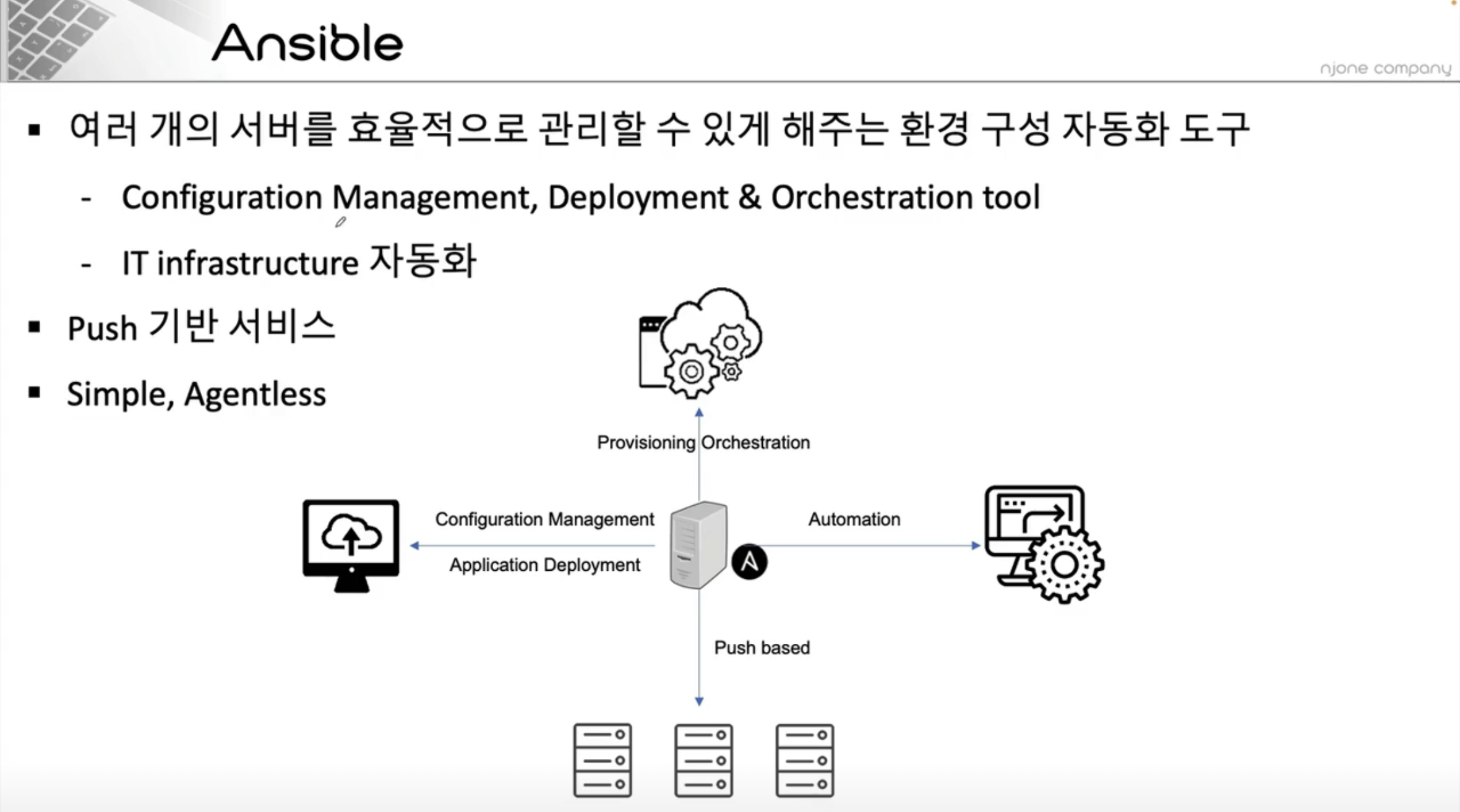
Jenkins 관리 → plugin Manager → Available plugins → Publish Over SSH → 설치

## **Jenkins + Infrastructure as Code**

## Infrastructure as Code

* 시스템, 하드웨어 또는 인터페이스의 구성정보를 파일을 통해 관리 및 프로비저닝
* IT 인프라스트럭처, 베어 메탈 등의 물리 장비 및 가상 머신과 관련된 구성 리소스를 관리
* 버전 관리를 통한 리소스 관리

## Ansible



Ansible을 통해서 기존에 있는 도커를 중지하고 다시 생성하거나 이미지를 다시 배포하여 인프라를 운영할 수 있도록한다.

Ansible과 Server는 서로 Python의 SSH 모듈을 통해 통신하기 때문에 별도의 Agent가 필요하지 않다.

이미지 준비하기

docker pull edowon0623/ansible

Docker Out Of Docker(DOOD)

docker run -itd --name ansible-server -p 20022:22 -e container=docker --tmpfs /run --tmpfs /tmp -v /sys/fs/cgroup:/sys/fs/cgroup:ro -v /var/run/docker.sock:/var/run/docker.sock edowon0623/ansible:latest /usr/sbin/init

각 컨테이너의 ip주소 확인

docker network inspect bridge

ansible-server : 172.17.0.3, jenkins-server : 172.17.0.2

Ansible이 접속을 각 컨테이너에 접속을 시도할때, pw 정보가 있어야지 접속이 가능한데 매번 pw를 입력하는게 번거로움므로 key를 생성하여 2개의 컨테이너에 해당 정보를 전달한 후 pw 인증시 사용하도록한다.

docker exec –it ansible-server bash

ssh-keygen

ssh-copy-id root@172.17.0.3

## **Jenkins + Ansible 연동하기**

Jenkins 서버에서 Ansible 서버로 접속할 수 있어야한다.

### 젠킨스 서버 접속



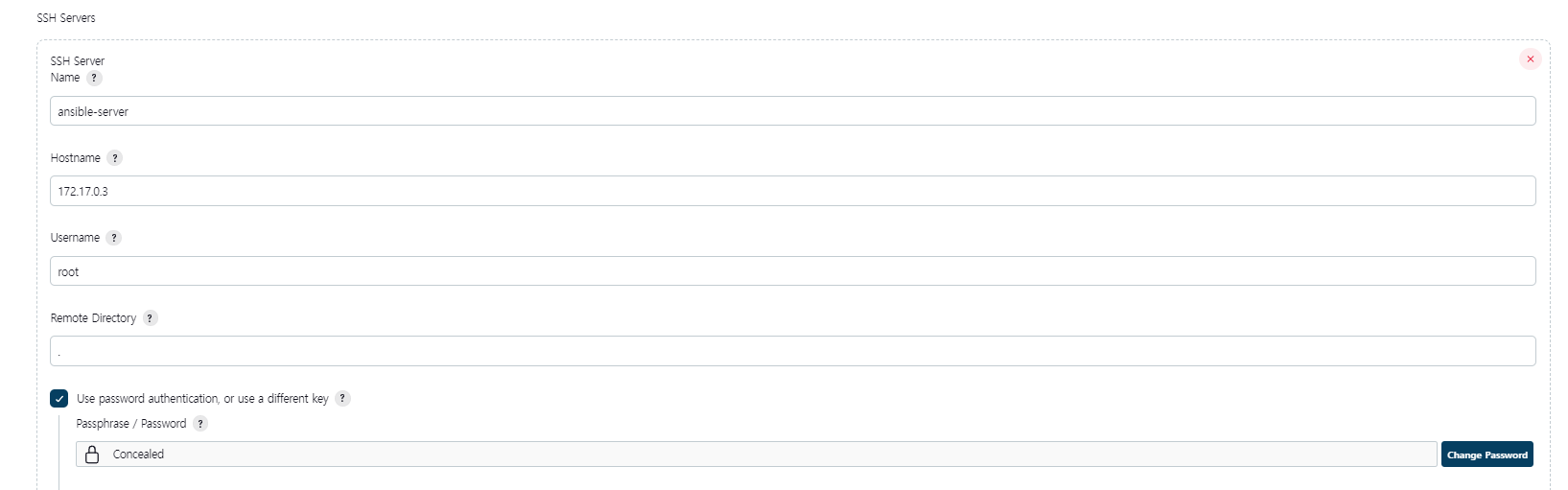
### 젠킨스 서버에서 Ansible 접속



* 접속 성공

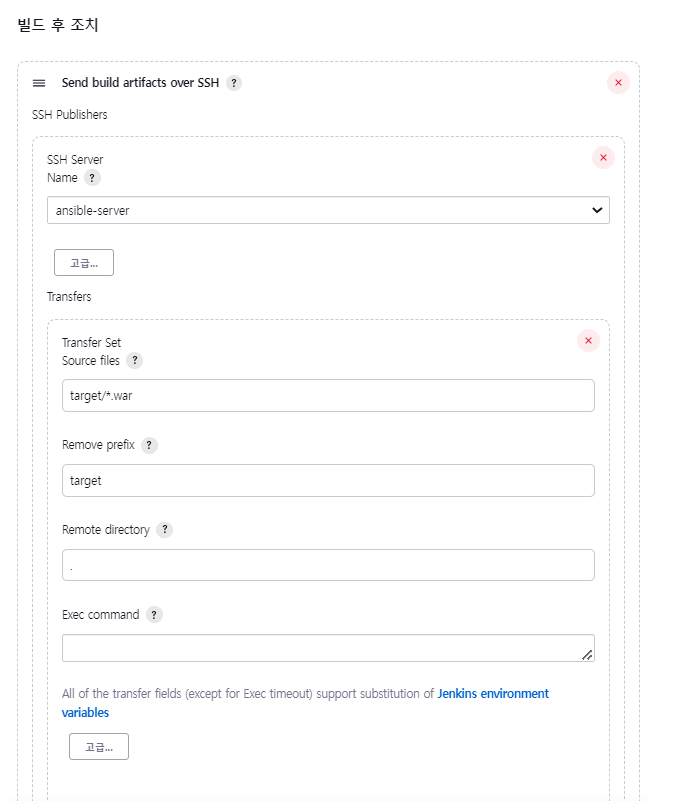
### 젠킨스 SSH 추가 설정하기

Dashboard > Jenkins 관리 > Configure System > SSH Servers



## **Jenkins + Ansible Playbook 사용하기**

jenkins가 ansible-server에 접속할 수 있도록 한다.



### hosts 만들기

mkdir /etc/ansible

vi /etc/ansible/hosts

hosts 의 내용은

[devops]

ansible ip 주소

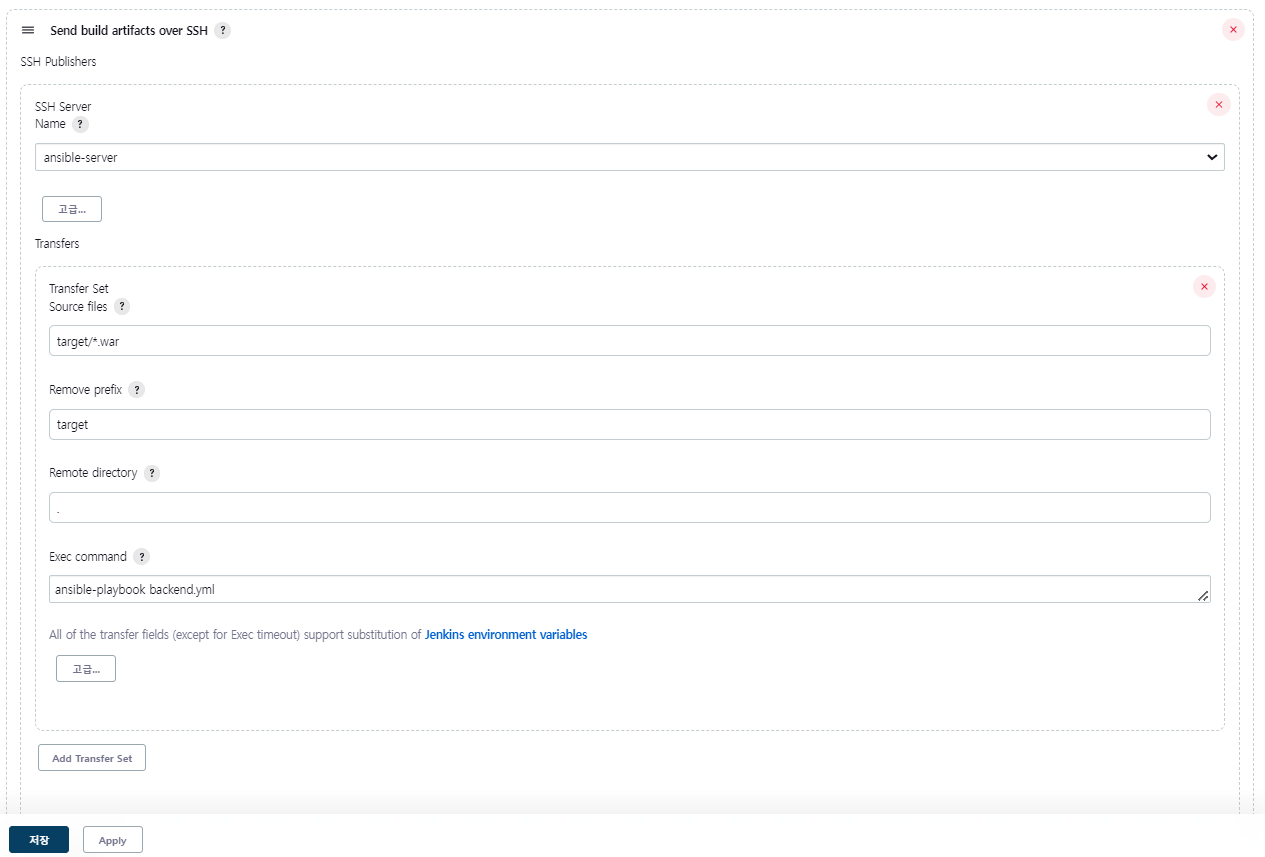
vi [아무이름].yml

**DockerFile 생성**



**Back-end.yml 파일 생성**





* exec command에 ansible-server에서 실행해야하는 명령어 적용
* ansible-playbook 아무이름.yml
  + playbook으로 hosts에 있는 ip주소에 아무이름.yml을 실행한다는 뜻

METASSAFY 메타버스 매뉴얼

본 문서는 멀티플레이를 위한 포톤 서버 구축과 유니티-리액트 통신 방법을 설명합니다.

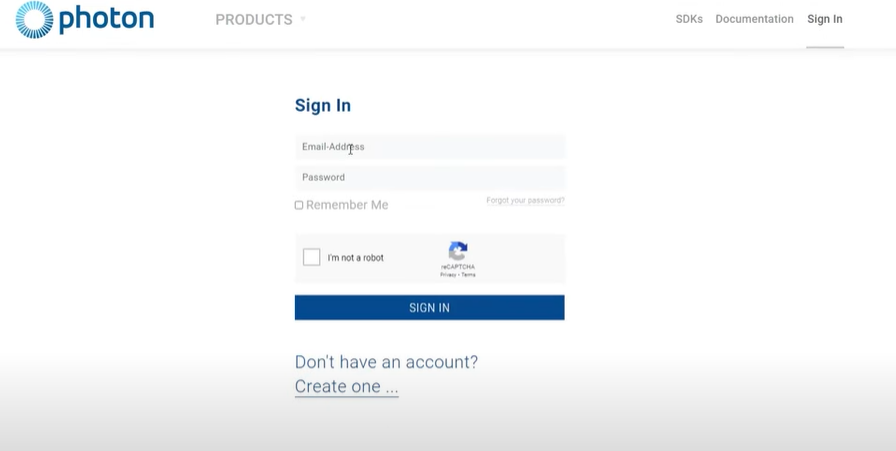
개발 환경

Unity 2021.3.16f1

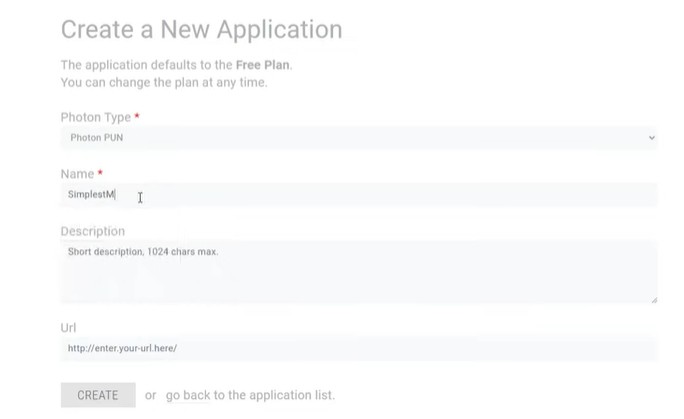
Photon Engine 2.4.1

Visual Studio 2021

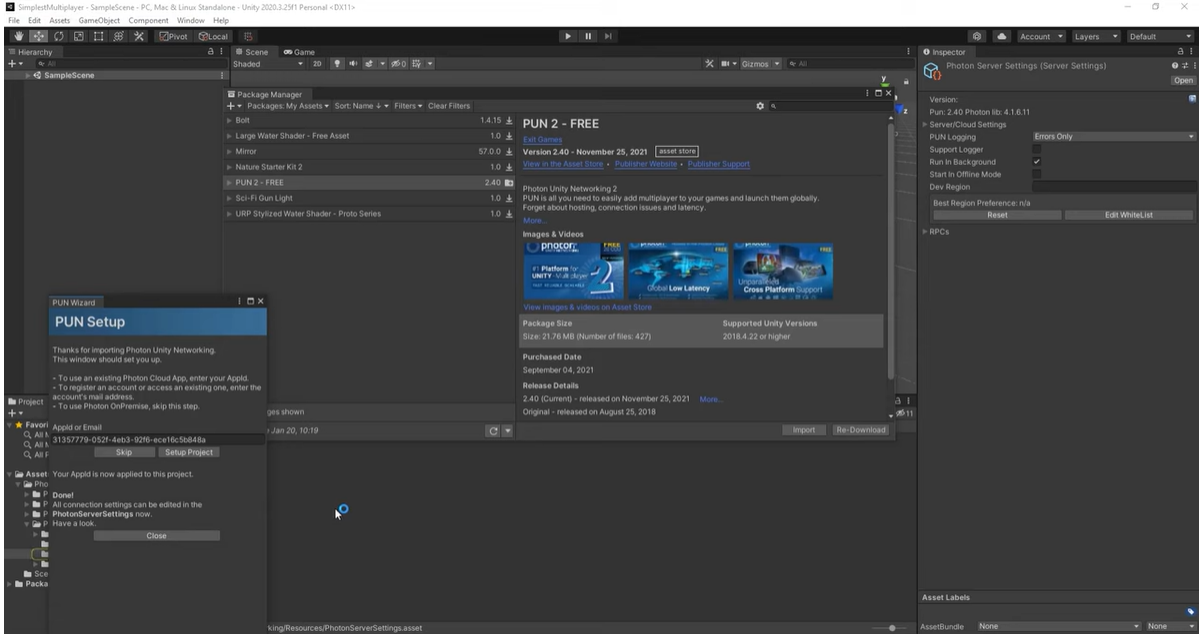
**포톤 서버**



포톤에서 제공하는 멀티플레이 서버를 사용하기 위해 회원가입, 로그인 진행



Photon PUN 타입의 새 Application을 추가한 뒤 app id를 복사한다.

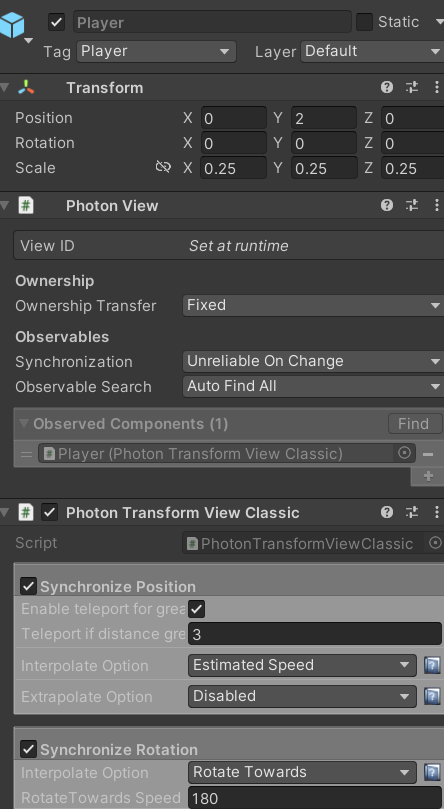


Unity asset store에서 pun2를 my asset에 추가한다.

Unity 에디터를 열어 Window-package manager- my asset을 열어 pun2를 import 한다.

Import 과정에서 복사한 app id를 입력하여 생성한 photon application과 연동한다.

**플레이어 설정**



플레이어가 멀티플레이가 가능하게 하기 위해서 플레이어 프리팹에 기본적인 포톤 속성들을 추가한다.

PhotonView: 멀티 플레이 환경에서 플레이어의 모습이 상대에게 보이도록 한다.

Photon Transform View Classic: 멀티 플레이 환경에서 플레이어들의 움직임이 동기화되도록 한다.

\*프리팹-유니티 상에서 쓰이는 객체 오브젝트

**방 생성 및 접속**

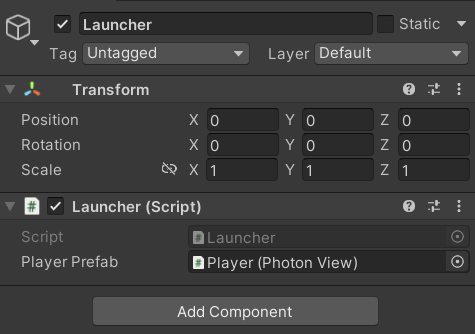


방 생성 및 접속에 관한 가장 기본적인 코드는 위와 같다.

씬이 시작되면 “Gumi” 라는 방에 접속한다. “Gumi” 방이 없으면 본인이 직접 생성하여 참가한다.

방의 옵션은 따로 설정하지 않는다.

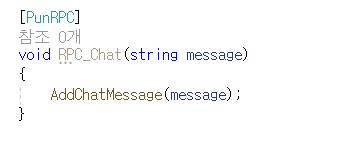
접속이 완료되면 x0, y0, z0 위치에 플레이어를 스폰시킨다.



PhotonManager 라는 GameObject를 생성한 뒤 위 코드를 붙이면 멀티플레이 환경 구축이 완료된다.

**상호작용**





채팅, 감정 표현 등의 상호작용을 하기 위해서는 PunRPC를 사용한다.

방의 다른 사용자들에게 보내고 싶은 이벤트를 PunRPC로 정의하고, 호출한다.

**React에 Unity 띄우기**

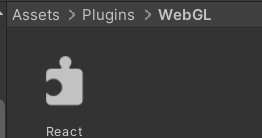
npm i react-unity-webgl

로 react-unity 통신에 필요한 라이브러리를 다운받는다.

****

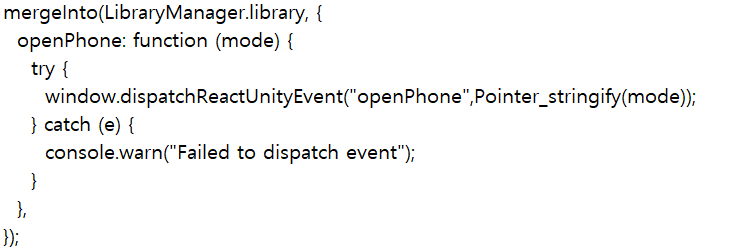
유니티 화면을 띄우기 위한 코드를 작성하고, 유니티 webgl 빌드 결과물을 react의 public 폴더 내에 넣는다.

**Unity->React 통신**

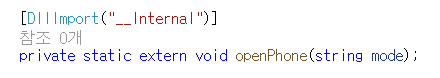


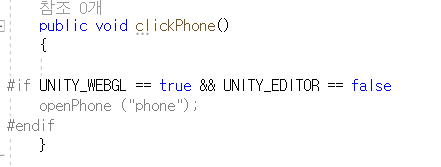
Unity 에디터 내에 Plugins-WebGL 폴더를 만들어 React.jslib 파일을 만든다.

(주의: 폴더명, 확장자명 대소문자 반드시 맞춰야 함)



jslib 파일에 openPhone이라는 함수를 작성한다.

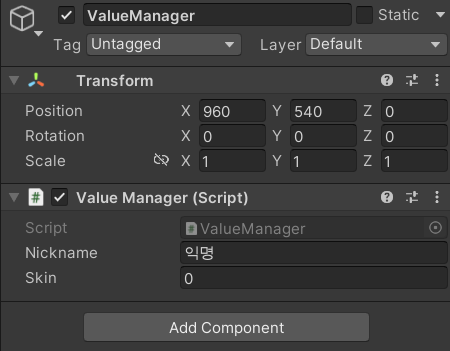




이후 Unity에서 openPhone이라는 함수를 사용해 React에 이벤트를 보낼 수 있다.

React에서는 addEventListener('openPhone', (mode) => {}); 를 통해 이벤트를 받을 수 있다.

**React->Unity 통신**

****

React로부터 이벤트를 받기 위한 ValueManager 오브젝트와 ValueManager 스크립트를 작성한다.(이름은 상관 없음)



SendMessage를 useUnityContext로부터 가져온다.



React에서 Unity로 이벤트를 보내고 싶을 때 sendMessage(“유니티 오브젝트명”,”유니티 내의 함수명”) 을 이용해 이벤트를 보낼 수 있다.

METASSAFY Frontend 설명서

프론트 빌드 정리

1. Node.js 설치

* npm 설치를 위해 필요
* Node.js 버전 : v18.13.0
* npm 버전: 8.19.3

1. 빌드 및 배포 (CI/CD에서 진행)

* > npm run build

**package.json dependencies**

|  |
| --- |
| 1. Css 관련 라이브러리   "@emotion/react": "^11.10.5",      "@emotion/styled": "^11.10.5",      "@mui/icons-material": "^5.11.0",      "@mui/lab": "^5.0.0-alpha.119",      "@mui/material": "^5.11.7",      "@mui/x-date-pickers": "^5.0.17",   1. Three.js의 React 버전 관련 라이브러리       "@react-three/drei": "^9.56.12",      "@react-three/fiber": "^8.10.1",   1. Redux Toolkit 관련 라이브러리       "@reduxjs/toolkit": "^1.9.1",   1. Socket 통신 라이브러리       "@stomp/stompjs": "^6.1.2",   1. HTTP 통신 라이브러리       "axios": "^1.2.4",      "express": "^4.18.2",   1. 애니메이션 라이브러리       "gsap": "^3.11.4",      "html2canvas": "^1.4.1",      "masonic": "^3.7.0",      "nodemon": "^2.0.20",   1. WebRTC OpenVide 라이브러리       "openvidu-browser": "^2.25.0",   1. React 기본 라이브러리       "react": "^18.2.0",      "react-dom": "^18.2.0",      "react-icons": "^4.7.1",      "react-redux": "^8.0.5",      "react-router-dom": "^6.7.0",      "react-scripts": "5.0.1",   1. React 로딩 스피너 라이브러리       "react-spinners": "^0.13.8",   1. Unity WebGL 라이브러리       "react-unity-webgl": "^9.4.0",   1. Redux 라이브러리       "redux": "^4.2.0",      "redux-logger": "^3.0.6",   1. SockJS 소켓 통신 라이브러리       "sockjs": "^0.3.24",      "sockjs-client": "^1.6.1",   1. 컴포넌트 CSS 라이브러리       "styled-components": "^5.3.6",   1. Three.js 라이브러리       "three": "^0.148.0",      "web-vitals": "^2.1.4" |

React 폴더 구조

|  |
| --- |
| S08P12D211  ├─ public  │  ├─ Build (Unity 빌드 결과 삽입)  │  ├─ index.html  └─ src     ├─ App.jsx (메인 컴포넌트)     ├─ assets (화면 단에 필요한 자료들)     │  ├─ fonts (폰트)     │  ├─ icons (svg 아이콘)     │  └─ images (이미지)     ├─ components (기능별로 분리한 컴포넌트)     │  ├─ auth (회원 관리 컴포넌트 폴더)     │  ├─ board (게시판)     │  ├─ developers (개발자 소개)     │  ├─ layout (Navbar 등의 레이아웃)     │  ├─ main (메인 페이지 구성 컴포넌트 폴더)     │  ├─ openvidu (WebRTC)     │  ├─ phone (휴대폰 모달 내 컴포넌트)     │  ├─ profile (프로필 관리)     │  └─ UI (중복 사용되는 UI)     ├─ hooks (Custom Hooks)     ├─ index.css     ├─ index.js     ├─ libs (라우터 컴포넌트 등 커스텀 라이브러리들)     ├─ pages (라우팅 적용할 화면 단위의 컴포넌트)     ├─ services (API Javascript)     ├─ store (전역 상태 관리 Redux)     └─ utils (스토리지 관련 등 유틸들) |