

포팅매뉴얼

개발 환경

형상 관리 도구	협업 도구	IDE	UI/UX
<ul style="list-style-type: none">• Git• Gitlab	<ul style="list-style-type: none">• Jira• Mattermost• Notion	<ul style="list-style-type: none">• VSCode (v.1.75.1)	<ul style="list-style-type: none">• Figma

사용 기술 정보

웹 프론트엔드	웹 백엔드	Embedded
<ul style="list-style-type: none">• Javascript• React.js (v.18.2.0)• Redux (v.1.9.1)• Socket.io(v.4.5.4)	<ul style="list-style-type: none">• Typescript(v.4.7.4)• Nest.js (v.9.0.0)• Fastify (v.9.2.1)• Typeorm (v.0.3.11)• MariaDB (v.10.10.3)• Node.js(v.18.04)	<ul style="list-style-type: none">• C• Arduino• ESP8266Wifi• Adafruit_NeoPixel• Raspberry Pi• Javascript• Socket.io(v.4.5.4)• Node.js(v.18.04)

인프라
<ul style="list-style-type: none">• AWS EC2• Docker(v.20.10.23)• Jenkins(v.2.375.3-lts-jdk11)• nginx(v.1.23.3)

배포 문서



웹 배포 과정은 **직접 배포**와 **도커 & 젠킨스 이용 배포** 두가지 방법으로 나누어 소개하고 있습니다. **둘 중 하나만 진행** 하면 됩니다.

[1] 배포 과정 (도커 및 젠킨스 이용 시)



다음 내용을 점검해야 한다.

- os환경이 리눅스 ubuntu 이며 20.04 LTS 버전인가?
- 인터넷이 연결 되어 있는가?
- 최소 20gb의 스토리지 용량, 4gb이상의 메모리 환경이 제공되어 있는가?

✓ 위 조건을 만족한다면 설정 준비가 완료되었다.!!

- 방화벽

먼저 방화벽을 열어야 한다.

아래에 있는 포트 및 프로토콜은 실제 프로젝트에서 사용한 방화벽이다.

port	protocol	ACTION	FROM	비고	필수?
22	TCP/UDP	ALLOW	Anywhere	SSH	O
80	TCP/UDP	ALLOW	Anywhere	nginx 서버	O
8080	TCP/UDP	ALLOW	Anywhere	jenkins 서버	O
3000	TCP/UDP	ALLOW	Anywhere	api 배포서버	X
3306	TCP/UDP	ALLOW	Anywhere	mariadb	O
123	UDP	ALLOW	Anywhere	NTP	X
3001	TCP/UDP	ALLOW	Anywhere	node.js backend 개발서버	X
3080	TCP/UDP	ALLOW	Anywhere	nginx 개발서버	X
3002	TCP/UDP	ALLOW	Anywhere	테스트용	X

아래 명령어를 통해 방화벽을 열 수 있다.

```
sudo ufw allow 22
sudo ufw allow 80
sudo ufw allow 8080
sudo ufw allow 3000
sudo ufw allow 3306
sudo ufw allow 123/udp
sudo ufw allow 3001
sudo ufw allow 3080
sudo ufw allow 3002
```

- 도커 설치

먼저 도커를 설치한다.

아래 명령어를 실행하여 Version: 20.10.23을 설치한다.

```
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

sudo mkdir -m 0755 -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo apt-get update

sudo VERSION_STRING=5:20.10.23-3-0-ubuntu-focal
sudo apt-get install docker-ce=$VERSION_STRING docker-ce-cli=$VERSION_STRING containerd.io docker-buildx-plugin docker-compose-plugin

sudo chmod 777 /var/run/docker.sock
```

그 다음 도커 이미지를 설치한다.

사용한 도커이미지는 다음과 같다.

Image 이름	설명	docker hub 이름
nginx	nginx 서버	nginx:1.23.3
node	node.js REFL	node:19.6.0-alpine3.17
jenkins	CI/CD 도구	jenkins/jenkins:2.375.3-lts-jdk11
mariadb	데이터베이스	mariadb:10.10.3-jammy

아래 명령어를 실행하여 도커 이미지를 설치한다.

```
sudo docker pull nginx:1.23.3
sudo docker pull node:19.6.0-alpine3.17
sudo docker pull jenkins/jenkins:2.375.3-lts-jdk11
sudo docker pull mariadb:10.10.3-jammy
```

• 젠킨스 설치

먼저 젠킨스를 설치한다. 다음 명령어를 입력한다.

```
sudo docker run -d -v jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -p 8080:8080 -p 5000:5000
```

젠킨스가 설치되었다면, 먼저 다음 명령어를 통해 초기 비밀번호를 확인한다.

```
sudo docker logs jenkins
```

아래와 같이 초기 비밀번호 값이 제공된다.

```
*****
*****
*****

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

[초기 비밀번호]

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
*****
```

초기 비밀번호를 기억해둔다.

- 젠킨스 내부 필요 프로그램 설치

해당 명령어를 통해 젠킨스 내부에 접근한다.

```
docker exec -it -u 0 jenkins bash
```

그 다음 특정 프로그램을 설치해야 한다.

도커와 git을 설치해야 한다.

다음 명령어를 순서대로 입력한다. (Y/N 을 물어보면 Y를 고른다)

```
apt-get update

apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release \
    git

mkdir -m 0755 -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \
    $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

VERSION_STRING=5:20.10.23~3-0~debian-bullseye
sudo apt-get install docker-ce=$VERSION_STRING docker-ce-cli=$VERSION_STRING containerd.io docker-buildx-plugin docker-compose-plugin
```


- 젠킨스 초기 설정

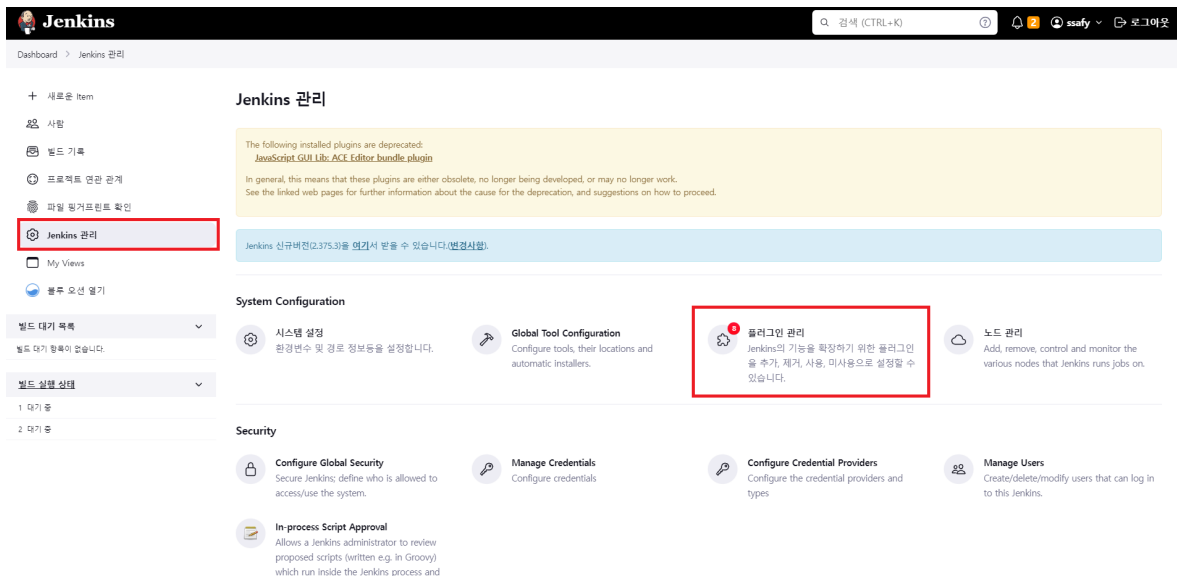
먼저 localhost:8080 로 들어간다.



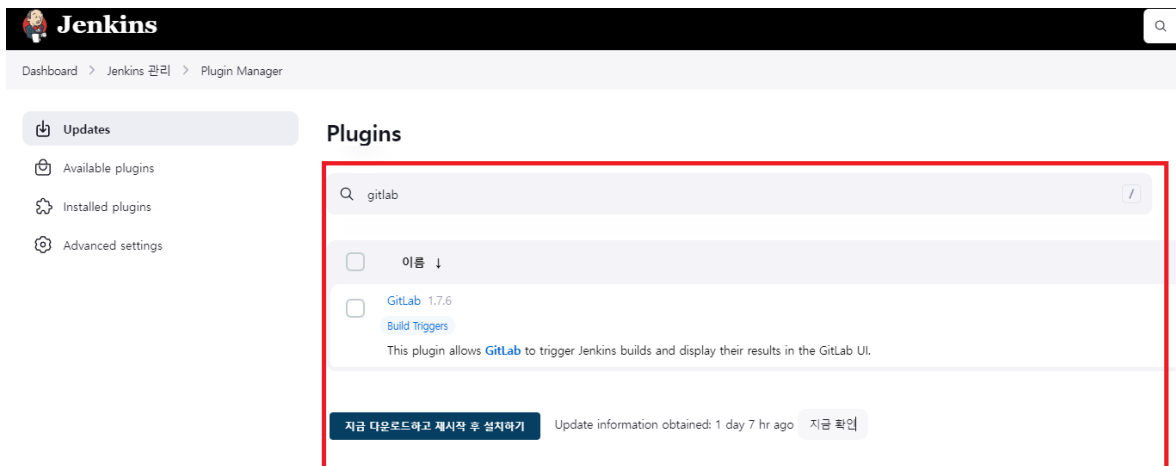
초기 비밀번호를 입력하고 로그인 한 뒤 Install suggested plugins를 선택한다.

초기 플러그인 설치를 기다린다.

 중간에 Admin User 계정을 만들라고 물어보는데, 마음껏 만든다.



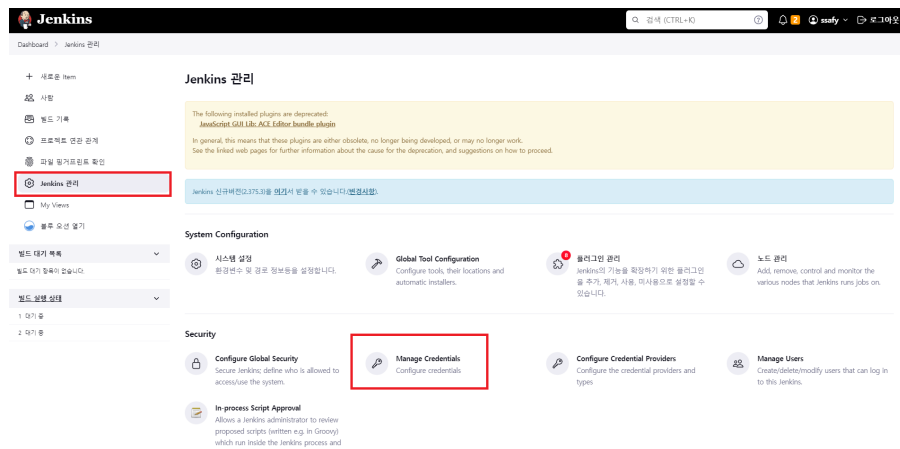
설치가 완료되었다면 Jenkins관리 → 플러그인 관리 (Plugin Manager) 로 이동한다.



그 다음, 플러그인 이름을 검색하여 플러그인을 설치한다.

설치해야 할 플러그인 목록은 다음과 같다.

1. Pipeline
2. Generic Webhook Trigger
3. Gitlab
4. Docker
5. Docker Commons
6. Docker Pipeline
7. Docker API

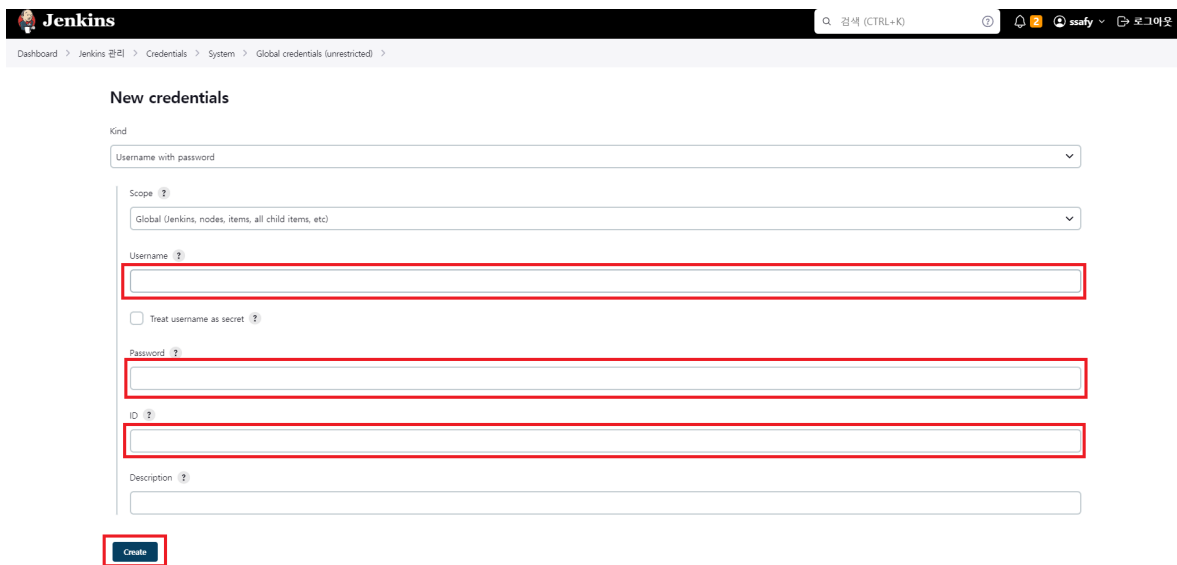


Jenkins관리 → Security → Manage Credentials에 들어간다.

Stores scoped to Jenkins아래에 System이 보이는데,



(global)을 누르고 [+ Add Credentials]를 누른다.

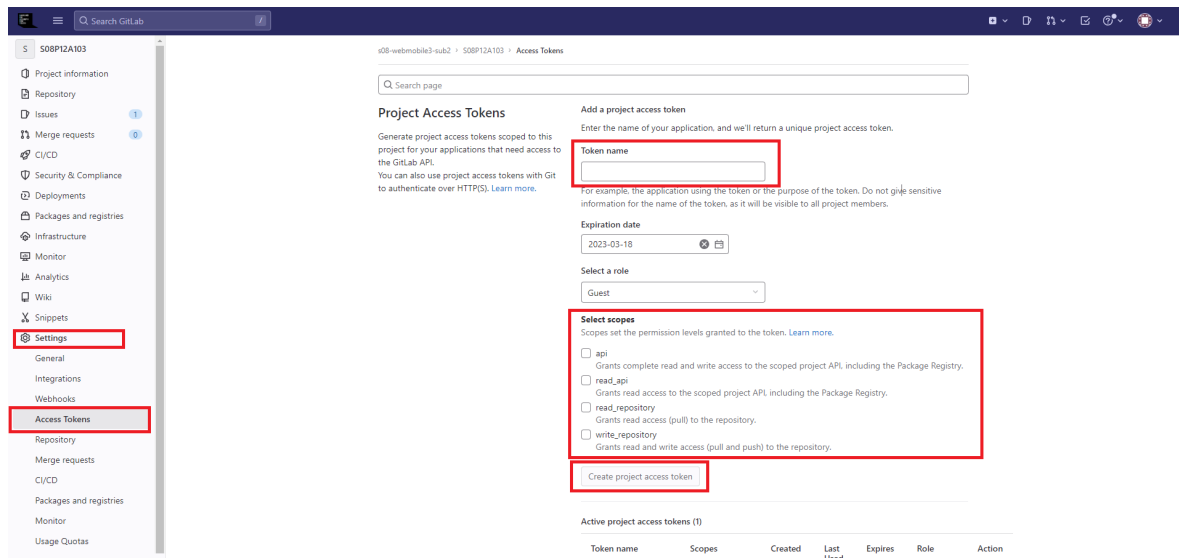


username with password를 선택하고 Scope를 Global로 둔 뒤,
gitlab repository 접근이 허용된 username과 password를 입력한다.

이때 ID는 "git_access_credential" 으로 짓는다.
(원치 않으면 변경 가능하나 pipeline내 변수도 변경해야 한다.)

username과 password가 노출되기 원치 않는다면, access token을 이용하는 방법이 있다.

먼저 Gitlab에 repository로 접속한다.



왼쪽 메뉴바에서 Settings → Access Token을 선택한뒤,
Token name에 username을 적고,

Select scopes에

☐ api, ☐ read_api, ☐ read_repository를 선택하고

Create project access token을 선택한다.

i Your new project access token has been created.
×

Q Search page

Project Access Tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API.

You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

Your new project access token

👁
📄

Make sure you save it - you won't be able to access it again.

Add a project access token

Enter the name of your application, and we'll return a unique project 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

2023-03-18
✕ 📅

Select a role

Developer
▼

Select scopes

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

☐ api

Grants complete read and write access to the scoped project API, including the Package Registry.

☐ read_api

Grants read access to the scoped project API, including the Package Registry.

☐ read_repository

Grants read access (pull) to the repository.

☐ write_repository

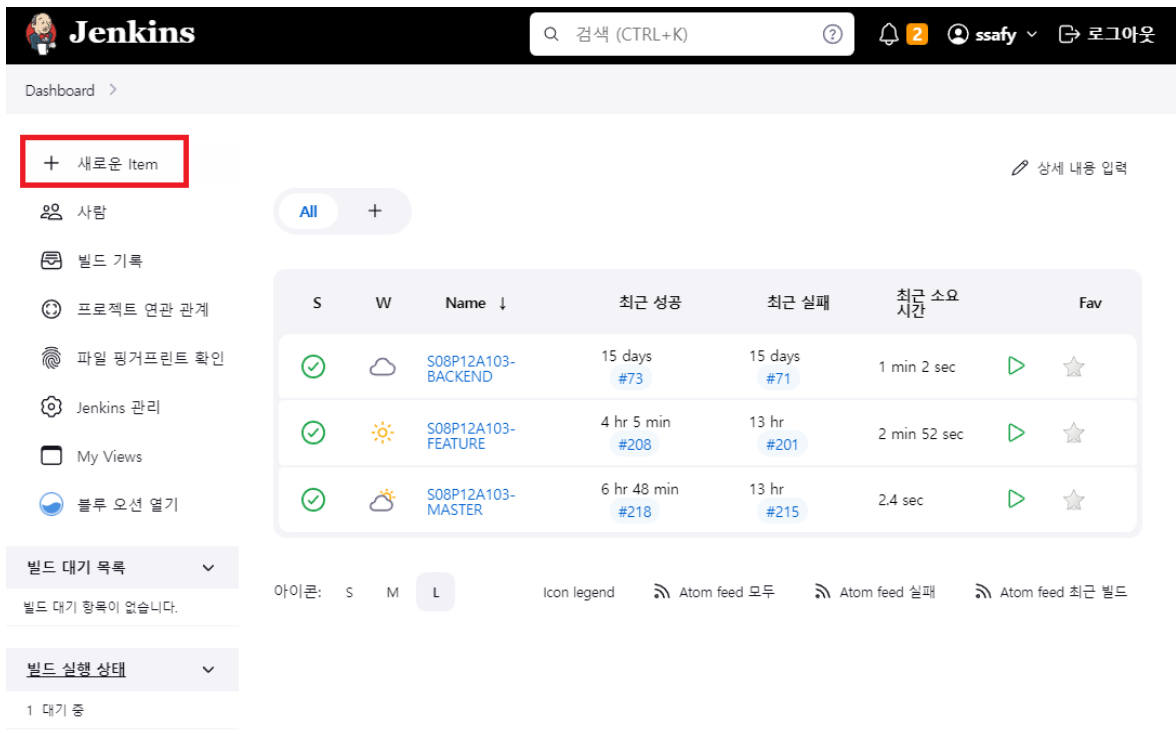
Grants read and write access (pull and push) to the repository.

이때 짤 하고 비밀 password를 주는데,

절대 다른 화면으로 이동하지 말고 그 password를 복사한다. (한번밖에 알려주지 않음)

이렇게 얻은 username(Token name) 과 password(비밀 password)를 사용하면 된다.

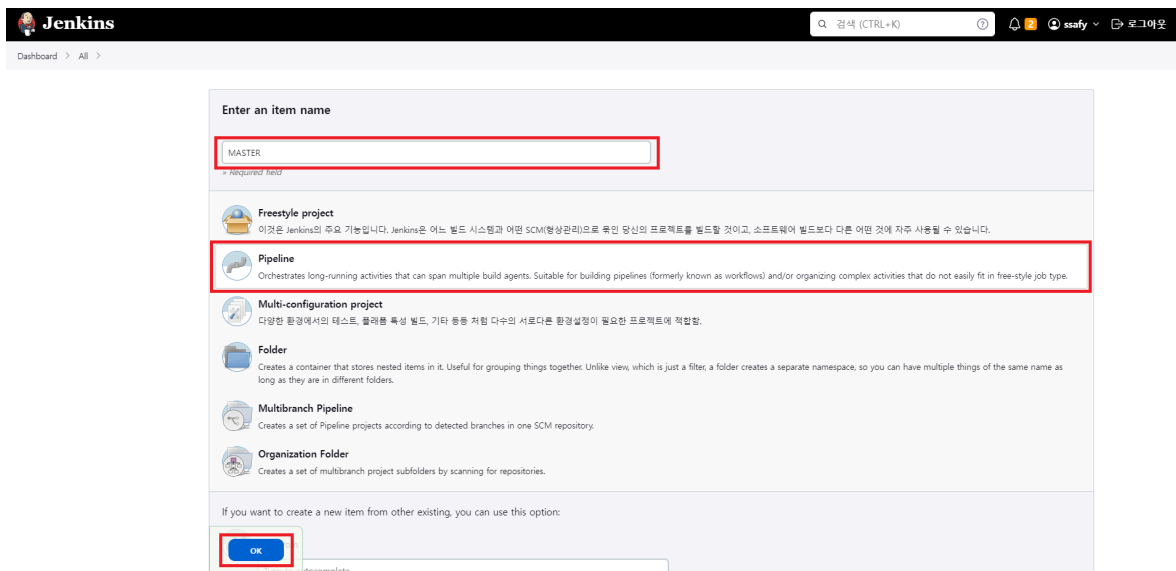
- Jenkins Job 만들기



The screenshot shows the Jenkins Dashboard. At the top, there's a search bar and user information. The left sidebar contains navigation links: '새로운 Item' (highlighted with a red box), '사람', '빌드 기록', '프로젝트 연관 관계', '파일 핑거프린트 확인', 'Jenkins 관리', 'My Views', and '블루 오션 열기'. The main area displays a table of build jobs:

S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간	Fav
✓	☁	S08P12A103-BACKEND	15 days #73	15 days #71	1 min 2 sec	▶ ☆
✓	☀	S08P12A103-FEATURE	4 hr 5 min #208	13 hr #201	2 min 52 sec	▶ ☆
✓	☁	S08P12A103-MASTER	6 hr 48 min #218	13 hr #215	2.4 sec	▶ ☆

Below the table, there are filters for '아이콘: S M L', 'Icon legend', and 'Atom feed' options. On the left, there are sections for '빌드 대기 목록' (1 대기 중) and '빌드 실행 상태' (1 대기 중).



The screenshot shows the 'Enter an item name' dialog in Jenkins. The 'MASTER' text is entered in the input field (highlighted with a red box). Below the input field, there are several project type options: 'Freestyle project', 'Pipeline' (highlighted with a red box), 'Multi-configuration project', 'Folder', 'Multibranch Pipeline', and 'Organization Folder'. The 'Pipeline' option is selected. At the bottom, there's an 'OK' button (highlighted with a red box) and a link to 'Jotomcolete'.

새로운 아이템(new Jobs) → Pipeline선택 및 이름을 짓고 OK를 눌러서 Job 하나를 만든다.

Configuration에서 General과 Advanced Project Options를 자유롭게 설정한다.

[해당 프로젝트는 git 설정을 pipeline에서 명시하므로 Gitlab Connection은 사용하지 않는다.]

Configure

General

Advanced Project Options

Pipeline

고급...

Pipeline

Definition

Pipeline script

Script ?

```

1 def repository = 'https://lab.ssafy.com/s08-webmobile3-sub2/S08P12A103'
2 def credentials = '68907cfb-7c81-4fe4-9b04-f5c46ed05f99'
3
4
5 def branch = 'master';
6 def branch_decorator = 'master'
7
8 def backend_container = "node-backend-${branch_decorator}"
9 def backend_location = './Backend'
10 def backend_port = 3000
11
12 def frontend_container = "nginx-frontend-${branch_decorator}"
13 def frontend_location = './FrontEnd/f_pjt'
14 def frontend_port = 80
15
16 pipeline {
17     agent any
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Use Groovy Sandbox ?

Pipeline Syntax

저장

Apply

설정 이후 Pipeline내에 아래와 같이 jenkins-pipeline을 입력한다.

```

def repository = 'https://lab.ssafy.com/s08-webmobile3-sub2/S08P12A103'
def credentials = 'git_access_credential'

def branch = 'master';
def branch_decorator = 'master'

def backend_container = "node-backend-${branch_decorator}"
def backend_location = './Backend'
def backend_port = 3001

def frontend_container = "nginx-frontend-${branch_decorator}"
def frontend_location = './FrontEnd/f_pjt'
def frontend_port = 80

pipeline {
    agent any
    stages {
        stage('Git Checkout') {
            steps {
                git branch: "${branch}", credentialsId: "${credentials}", url: "${repository}"
            }
        }
        stage("Deploy"){
            parallel {
                stage('Frontend') {
                    when{
                        expression { sh( returnStdout: true, script: "git show --online --first-parent -- ${frontend_location}").trim().length() > 0 }
                    }
                    stages{
                        stage('Build Image') {
                            steps {
                                sh "docker build -f ${frontend_location}/Dockerfile-prod -t ${frontend_container}:latest ${frontend_location}"
                            }
                        }
                        stage('Drop Container') {
                            when {
                                expression { sh( returnStdout: true, script: "docker ps -f name=${frontend_container} -q").trim().length() > 0 }
                            }
                            steps {
                                sh "docker stop ${frontend_container}"
                                sh "docker rm ${frontend_container}"
                            }
                        }
                        stage('Run Container') {
                            steps {
                                sh "docker run -it -d -p ${frontend_port}:80 --name ${frontend_container} --add-host=host.docker.internal"
                            }
                        }
                    }
                }
                stage('Backend') {
                    when{

```


구성(Configuration) → General → Build Trigger로 이동한다.

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab, GitLab webhook URL: https://gitlab.com/api/v4/projects/1234567890/hooks/1234567890 ?

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

이때 **Build when a change is pushed to GitLab**. GitLab webhook URL: [외워야할 URL]
을 체크 하고, **Push Event**를 체크한다.

Dashboard > S08P12A103-MASTER > Configuration

Configure

General

Advanced Project Options

Pipeline

☒ 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

고급...

☐ Generic Webhook Trigger ?

☐ GitHub hook trigger for GITSCM polling ?

☐ Poll SCM ?

☐ Quiet period ?

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

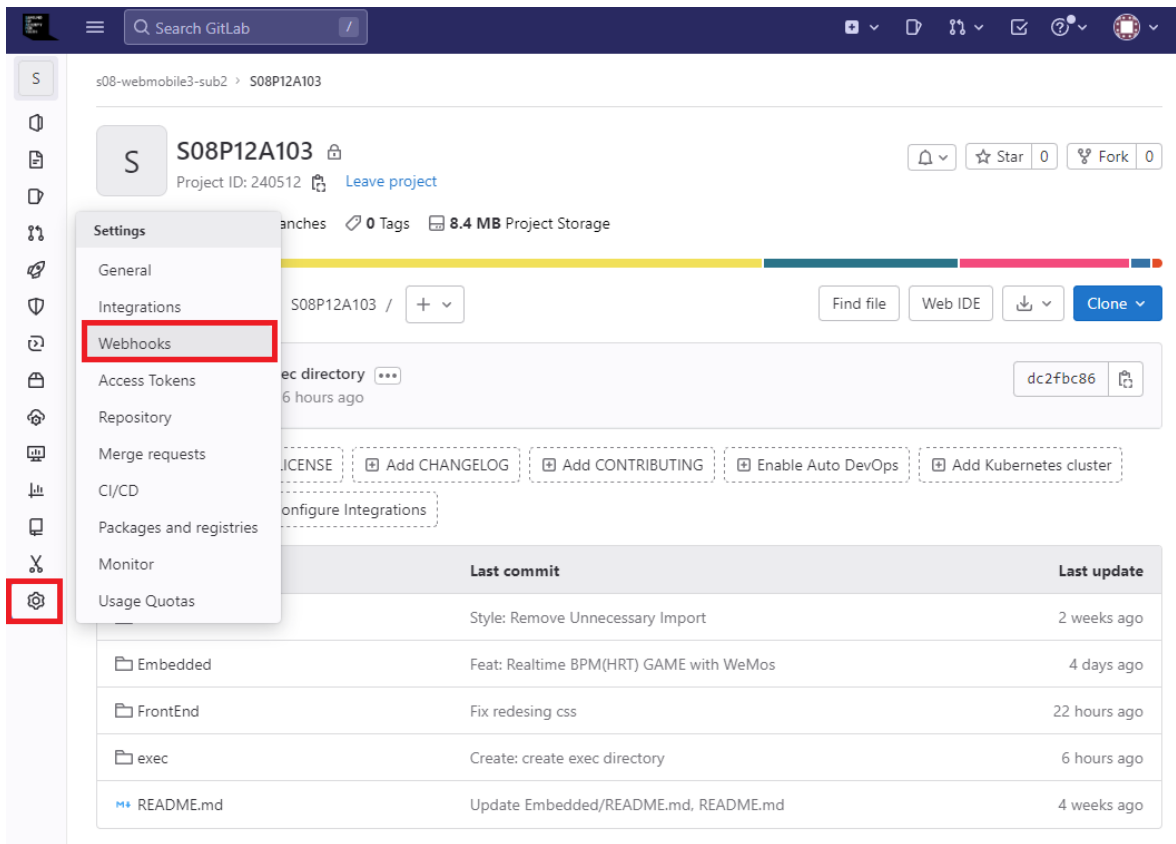
Secret token ?

Generate

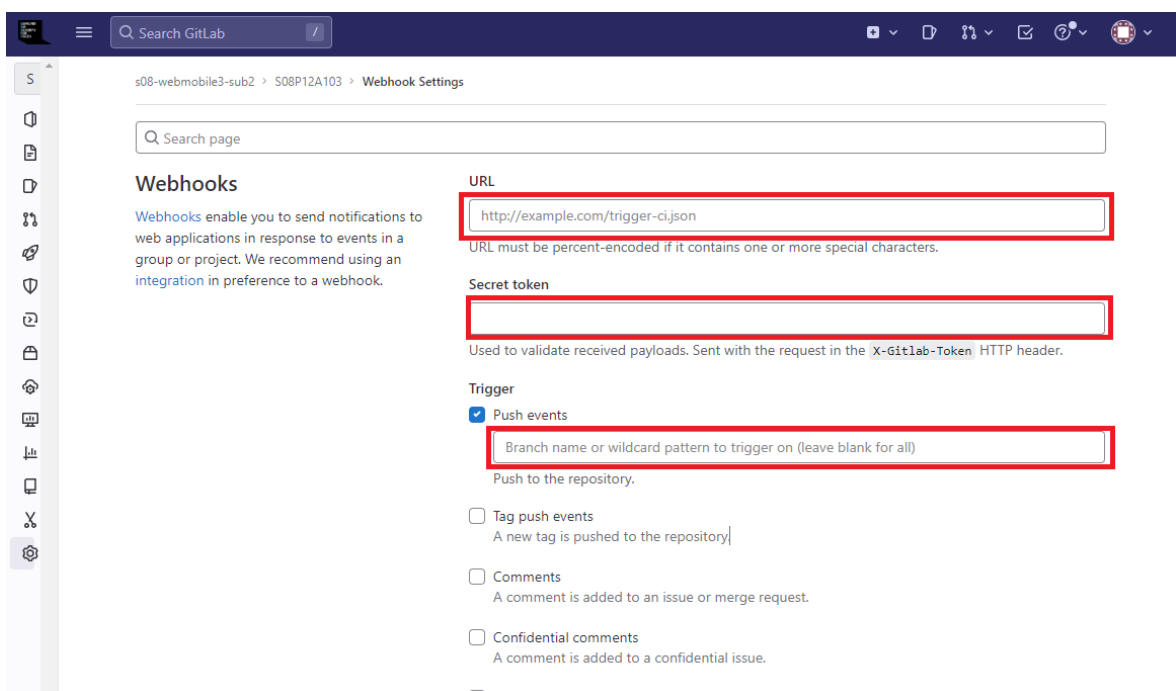
Clear

하단에 **[고급(Advance)]**를 누르고 Secret Token 항목으로 이동하여 **[Generate]**을 누른다.
그때 나온 값을 **[외워야할 Token]** 이라고 정의하자.

이제 획득한 값을 가지고 Gitlab에 repository로 접속한다.



왼쪽 메뉴바에서 Settings → Web Hooks를 선택한다.



URL 항목에 **[외워야할 URL]** 을 입력하고, Secret token에 **[외워야할 Token]** 을 입력한다.

Push Event를 체크하고, WebHook을 걸을 브랜치(master)이름을 입력한다.

하단에 **Save Changes** 을 누르면 WebHook 설정이 완료된다.

- mariadb 설치

다음 명령어를 입력하여 mariadb를 설치한다.

```
sudo docker run --detach -p 3306:3306 --network ssafy-project-network --name database --env MARIADB_USER=ssafy --env MARIADB_PASSW
```

상단에서 설정한 **MARIADB_PASSWORD**, **MARIADB_ROOT_PASSWORD** 는 데이터베이스 비밀번호, **MARIADB_USER**는 기본 계정 아이디이다.

원하는 것으로 바뀌도 된다. 단, 바꿀 시, secret에 있는 db.json파일 역시 변경해야 한다.

그 다음 mariadb에 접속하여 i8a103dump.sql을 실행시킨다.

[2] 배포 과정 (직접 배포시)



빌드 환경 테스트는 리눅스 환경(ubuntu) 진행되었으며, 다른 OS에서의 방법에 차이가 있을 수 있습니다.

- 사전작업

(1) 아래 프로그램을 설치한다. 설치시 선택한 OS에 맞게 적절히 설치한다.

이름	설명	버전	설치위치
git	git	최신버전	웹 서버
nginx	웹 서버	최신버전	웹 서버
node	API 서버	18.14.0	웹 서버
node	술디스펜서 서버	18.14.0	라즈베리파이 서버
mariadb	데이터베이스	최신버전	데이터베이스

(2) git clone등을 통해 repository내용을 다운 받는다.

(3) exec폴더 내 제공된 Secret 폴더내 파일을 전부 복사하여 다운 받은 폴더 위에 붙여넣는다.

(4) 작업 디렉토리를 프로젝트 최상단 루트로 이동한다.

- 백엔드빌드

(1) ./Backend 위에 다음 명령어를 실행한다.

```
npm install
npm run start
```

(2) localhost:3000 으로 접속하면 된다.

- 프론트엔드 빌드

(1) ./FrontEnd/f_pjt/src/url.js파일내 URL을 다음과 같이 수정한다.

```
export const URL= 'http://localhost/api';
```

(2) ./FrontEnd/f_pjt 위에서 다음 명령어로 빌드한다.

```
npm ci
npm install react-scripts@latest -g
npm run build
```

(3) default.conf과 build 폴더를 nginx에 덮어 쓴다.

아래는 linux 환경에서의 예시이며, 운영체제에 따라 해당 내용은 달라 질 수 있다.

```
./FrontEnd/f_pjt/conf.prod/conf.d/default.conf 파일을 /etc/nginx/conf.d/default.conf 위에 덮어쓴다.
yes |cp -arpf ./FrontEnd/f_pjt/conf.prod/conf.d/default.conf /etc/nginx/conf.d/default.conf

./FrontEnd/f_pjt/build 디렉토리 통째로 /usr/share/nginx/html 디렉토리 위에 덮어쓴다.
yes |cp -arpf ./FrontEnd/f_pjt/build /usr/share/nginx/html
```

(4) default.conf 파일을 수정한다. 아래 참조.

```
"host.docker.internal"를 "localhost"로 수정
"/usr/share/nginx/html"를 실제 nginx/html 위치로 수정
```

(5) nginx를 실행시킨다.

다음은 linux환경에서 예시이며, 운영체제에 따라 해당 내용은 달라 질 수 있다.

```
sudo systemctl stop nginx
sudo systemctl start nginx
```

(5) localhost:80으로 접속하면 된다.

- 데이터베이스 설정

(1) mariadb를 설치하고 실행한다. 구체적인 설치방법은 생략한다.

(2) ./BackEnd/src/env/db.json파일을 수정한다.

하단 내용 중 “호스트주소”, “접속아이디”, “접속패스워드” 를 수정하면 된다.

```
{
  "host": "호스트주소",
  "port": 3306,
  "username": "접속아이디",
  "password": "접속패스워드",
  "database": "ssafy_project",
  "test_database": "ssafy_project_test",
  "timezone": "+09:00",
  "charset": "utf8mb4"
}
```

(3) mariadb에 접속하여 i8a103dump.sql을 실행시킨다.

배포시 특이 사항

- SQL dump내 beverage 테이블내 삽입된 데이터들의 column 값중 beverage_image_url이 실제로 개인이 사용하고 있는 clouldinary CDN에서 가져온 이미지를 사용하였는데, CDN내 이미지의 expired 일자가 지나게 되면 이미지가 찢릴 수 있습니다. 이 경우 이미지 url을 변경해서 사용해야 합니다.
- git 주소가 변경되거나, 브랜치 이름이 변경되었을시 반드시 jenkins pipeline코드내 선언 변수중 branch와 repository를 변경해야 합니다.
- ./FrontEnd/f_pjt/src/url.js파일내

```
export const URL= 'http://i8a103.p.ssafy.io:3001';
```

이라고 되어있는 URL 주소를 서버호스팅 url로 변경해야 합니다. 만약 로컬인 경우 localhost로 변경해야 합니다.

DB 접속 정보 등 프로젝트에 사용되는 프로퍼티 파일 목록

아래 파일들은 “secret폴더” 내 정의 되어 있으며, 프로젝트에 복사하여 사용한다.

파일명	설명
/BackEnd/src/auth/constant.ts	nestjs에서 사용하는 auth 환경변수
/BackEnd/src/env/db.json	mariadb 데이터베이스 정보
/FrontEnd/f_pjt/src/url.js	프론트엔드에서 접근하는 API서버 URL 정보