

# 포팅 메뉴얼

## 사용 프로그램 버전



프로젝트에서 사용한 프로그램의 버전을 정리합니다.

### Backend

- SpringBoot 2.7.13
  - JDK 11
- FastAPI
  - python 3.9
  - fastapi 0.104.0
  - uvicorn 0.24.0

### Frontend

- Node.js 18.16.1
- React 18.2.0
- React-Calendar 4.6.1
- React-Webcam 7.2.0
- Recoil 0.7.7
- React-Speech-Recognition 3.10.0

### Database

- Mysql 8.0
- Redis 5.0.7
- MongoDB 4.4.25

## 배포 환경




서버구성에 사용된 코드를 정리합니다.

### 서버구성(AWS)


### Terminus


**Address**


 k9b107a.p.ssafy.io


**General**

silapmyeon


 Parent Group


 Tags



 Backspace Default


[Share this host](#) 


**SSH on** 22 **port**


 ubuntu

 K9B107T.pem

 Certificate  FIDO2

 Termius Light

[Show more](#) 

 Add Telnet

**Connect**

- Label은 호스트명 원하는 이름으로 입력
- Address는 EC2의 도메인 주소를 입력
- SSH Username에는 ubuntu 입력
- Set a Key를 눌러 EC2 .pem키를 등록

## EC2에 Mysql 설치

### 1. mysql 설치하기

```
sudo apt-get update
sudo apt-get install mysql-server

#mysql 접속
sudo mysql -u root //초기에는 root로 -p없이 로그인 하면 돼
```

## 2. 설치한 mysql 접속

```
#사용할 database 만들기
create database DB이름;

#database 만들어 졌는지 확인하기
show databases;
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sam |
| sys |
+-----+
5 rows in set (0.00 sec)
```

## 3. 사용할 user 생성하기

```
#db바꾸기
use mysql;

#사용자에게 권한 주기 '%'->모든 권한 주기
create user '사용자 이름'@'%' identified by '비밀번호';
grant all on 'db이름'.* to '권한 줄 사용자 이름'@'%';

#권한 잘 주어져는지 확인
show grants for 'db이름'@'%';
```

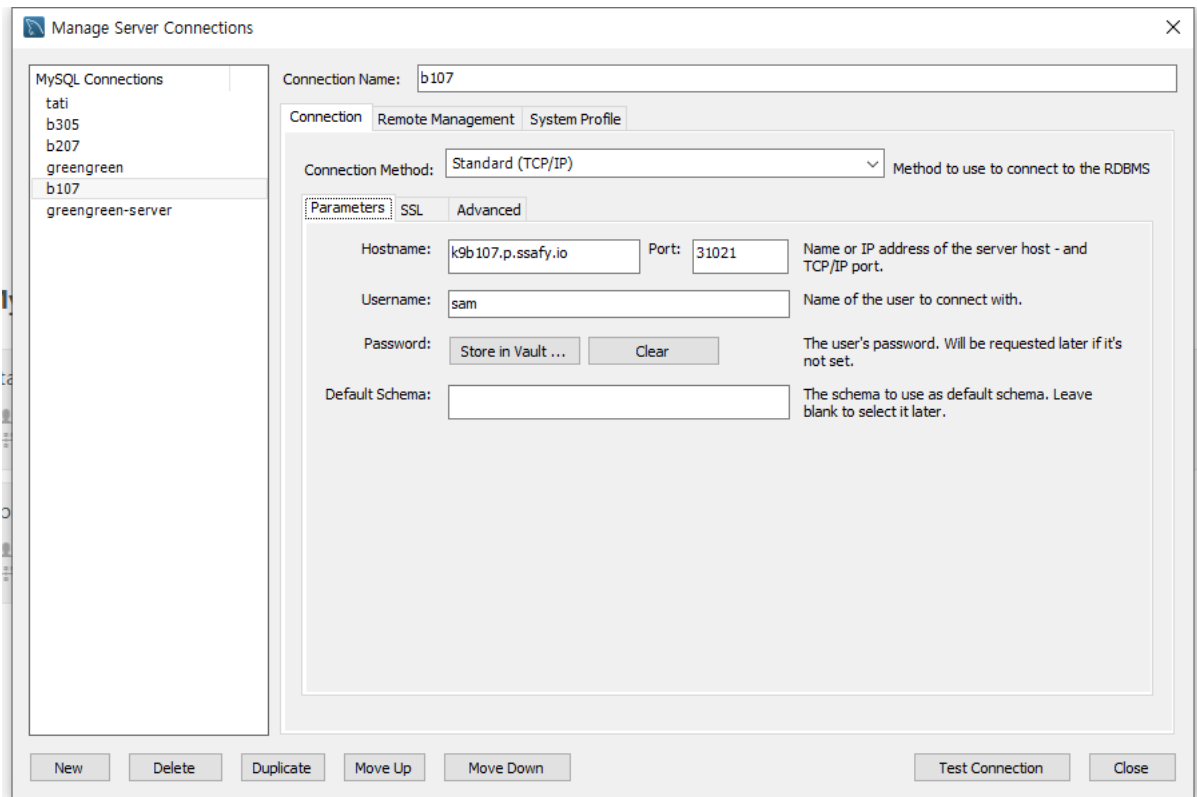
```
mysql> show grants for 'sam'@'%';
+-----+
| Grants for sam@% |
+-----+
| GRANT USAGE ON *.* TO `sam`@`%` |
| GRANT ALL PRIVILEGES ON `sam`.* TO `sam`@`%` |
+-----+
2 rows in set (0.00 sec)
```

## 4. local workbench 설정

- 로컬 workbench에서 ec2 DB에 접속하기 위해 추가적으로 설정해줘야 하는 부분  
→ 127.0.0.1로 설정 되어있는 bind-address값을 0.0.0.0으로 수정해서 외부 접속 허용해준다.(i 눌러서 수정해주기 → ctrl+c (insert 종료) → :wq+enter (저장))
- sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf

```
# If MySQL is running as a replication slave, this should be
# changed. Ref https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_tmpdir
# tmpdir = /tmp
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address = 0.0.0.0
mysqlx-bind-address = 127.0.0.1
#
```

```
sudo service mysql stop
sudo service mysql start
```



- password의 store in vault ...를 눌러서 비밀번호 입력 후

```
#EC2의 포트번호 열어주기
sudo ufw allow 3306
```

## Docker + Jenkins

- EC2에 docker 설치

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg

sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \
"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
"${. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- docker engine과 그에 따른 plugin설치

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

sudo apt install docker-compose

#정상 설치 되었는지 확인
sudo docker -v
sudo docker compose version
```

## Jenkins 설치

### | docker-compose를 사용해서 jenkins container를 실행시키기

- jenkins container를 실행시킬 dockcer-compose 만들기

```
sudo vim docker-compose.yml

## 아래 작성 된 사항은 jenkins 기본이미지로 jdk 11버전을 지원해주기 때문에jdk 17이상을 설치 해야 한다면
## -> image: jenkins/jenkins:jdk17 이렇게 수정해 주면 됨
## jenkins 기본 포트는 8080인데 9090포트 사용하도록 지정해줌

# (i) : docker - compose 파일에 필요한 스펙을 작성 // 아래 캡처 화면 작성하기
# (ctrl+c) : 작성 완료
# (:wq) : 저장, 나오기
```

```
version: '3'

services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /jenkins:/var/jenkins_home
    ports:
      - "9090:8080"
    user: root
```

```
sudo docker-compose up -d

##정상적으로 jenkins container가 실행 되고 있는지 확인
sudo docker ps
##정상적으로 실행 되고 있지 않다면 해당 container가 어떤 상태인지 확인
sudo docker ps -a
```

## Jenkins 컨테이너 내부에 접속해서 docker 설치

```
sudo docker exec -it jenkins bin/bash
docker
## root~~ 나오면
## jenkins 컨테이너 내부에 접속 완료

##docker 설치
apt-get update
apt-get install ca-certificates curl gnupg lsb-release
mkdir -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
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-compose
```

## Jenkins, GitLab → SpringBoot 배포하기

### | jenkins : http://도메인:9090포트로 접속

- 처음 접속 시 admin password는 Jenkins container의 log에서 확인하기

```
sudo docker logs jenkins
```

```
2023-07-11 23:59:07.921+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: Started initialization
2023-07-11 23:59:08.811+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: Listed all plugins
2023-07-11 23:59:09.332+0000 [id=31] INFO jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
2023-07-11 23:59:09.339+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: Started all plugins
2023-07-11 23:59:09.353+0000 [id=29] INFO jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
2023-07-11 23:59:09.846+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: System config loaded
2023-07-11 23:59:09.847+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: System config adapted
2023-07-11 23:59:09.847+0000 [id=29] INFO jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
2023-07-11 23:59:09.849+0000 [id=29] INFO jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs
pdated
2023-07-11 23:59:10.005+0000 [id=38] INFO jenkins.install.SetupWizard#init:

*****
*****
*****

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

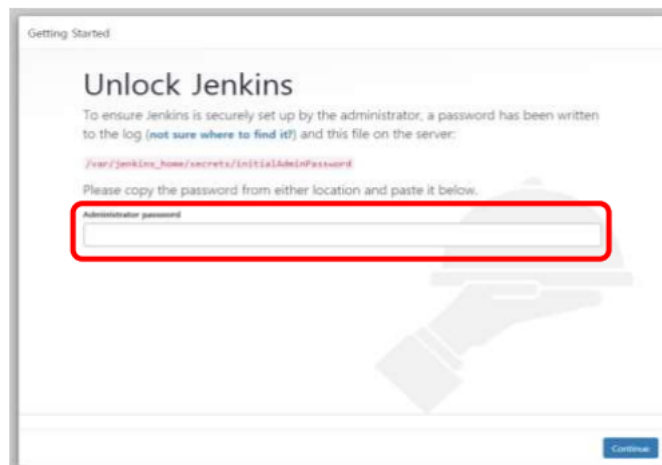
feal7cd928e942e386c36788ea14efce

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

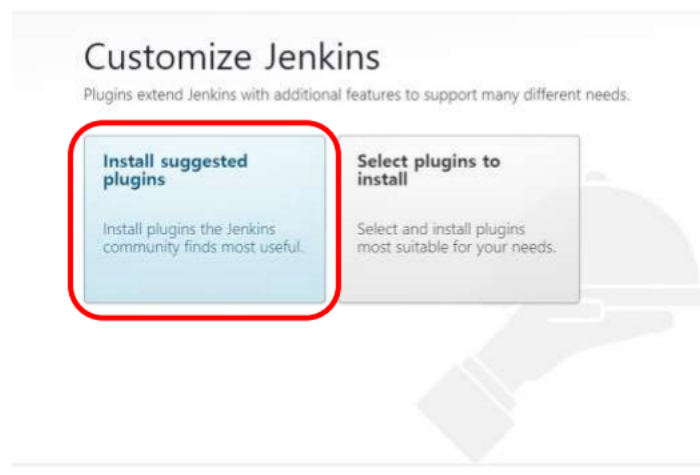
*****
*****
*****

WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1 (file:/var/jenkins_home/war/WEB-INF/
b/groovy-all-2.4.21.jar) to constructor java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)
```

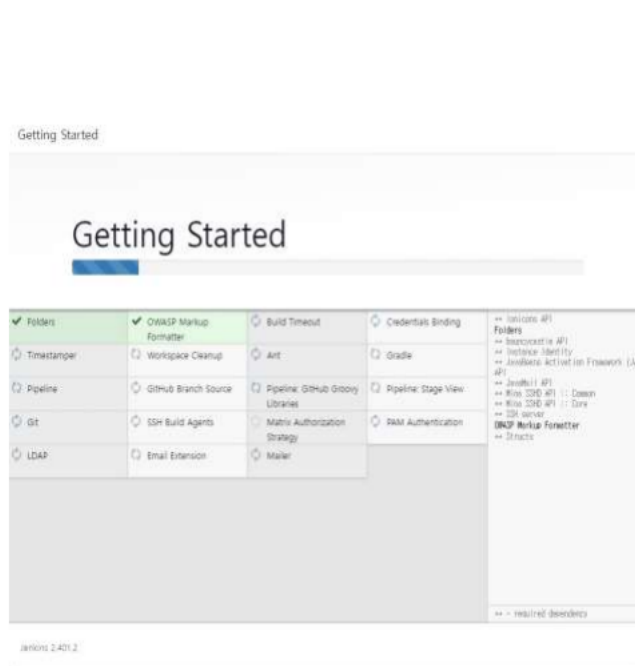
- 해당 부분 ctrl + shift + c 해서 아래 입력에 넣기



- Plugin 설치



- 추후에도 원하는 Plugin 설치 가능



- Jenkins 접속 계정 만들기(Jenkins ID, PW 어렵게 해서 해킹 위험 방지)

- Jenkins URL 변경 설정 (URL 원하는 대로 바꾸기 가능 ⇒ 해킹 위험 감소)

- http://도메인주소:9090으로 접속 후 위에서 설정한 ID, PW로 로그인
- 추가 plugin 설치하기, Jenkins 관리 접속

- + 새로운 Item
- 사람 사람
- 빌드 기록
- 프로젝트 연관 관계
- 파일 핑거프린트 확인
- Jenkins 관리
- My Views

- Plugins 접속 후 Docker, Docker Commons, Docker Pipeline, Docker API 설치
- GitLab 관련도 설치

**System Configuration**

**System**  
 환경변수 및 경로 정보등을 설정합니다.

**Tools**  
 Configure tools, their locations and automatic installers.

**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, ...

- GitLab
- Generic Webhook Trigger
- GitLab API
- GitLab Authentication
- Loading plugin extensions

→ [메인 페이지로 돌아가기](#)  
 (설치된 플러그인을 바로 이용하실 수 있습니다.)

→ 설치가 끝나고 실행중인 작업이 없으면 Jenkins 재시작.

- 인증 정보 만들기 Jenkins 관리 → Security → Credentials

**Security**

**Security**  
 Secure Jenkins; define who is allowed to access/use the system.

**Credentials**  
 Configure credentials

**Credential Providers**  
 Configure the credential providers and types









**Users**  
 Create/delete/modify users that can log in to this Jenkins.

**In-process Script Approval**  
 Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions. 1 scripts pending approval. 1 signatures pending approval.

- GitLab으로 jenkins\_token, .pem키의 값으로 ssh\_key 설정



## Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	jenkins_token	holizon9@naver.com/*****
		System	(global)	3e9e0d46-a1ab-4fbb-83fa-03a67e2f77c7	/*****
		System	(global)	af10982d-67a8-42c0-bd8e-7a09afe262d5	/*****
		System	(global)	ssh_key	ubuntu (ssh_key)

- jenkins\_token 설정

### Update credentials

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

Username ?  
holizon9@naver.com

☐ Treat username as secret ?

Password ?  
Concealed Change Password

ID ?  
jenkins\_token

Description ?

Save

- ssh\_key 설정

### Update credentials

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

ID ?  
ssh\_key

Description ?  
ssh\_key

Username  
ubuntu

☐ Treat username as secret ?

Private Key  
☒ Enter directly


Key  
Concealed for Confidentiality Replace


Passphrase


Save


- Jenkins 관리에서 System Configuration → System


## System Configuration


**System**  
환경 변수 및 경로 정보들을 설정합니다.

**Tools**  
Configure tools, their locations and automatic installers.

**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, ...

## • GitLab

**GitLab**

☒ Enable authentication for '/project' end-point

GitLab connections

**원하시는 Connection명으로 설정하세요.**

Connection name  
name for the connection  
deplot-test

GitLab host URL  
The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)  
**https://lab.ssafy.com/**

Credentials  
API Token for accessing GitLab  
- current -  
Add +

**Credentials는 앞서 발급받은 Personal Access Token을 Gitlab API Token으로 연결하면 Test Connection은 되지만, 우리는 Username with password 인증 방식을 사용하기 때문에 설정하지 않아도 됩니다.**

그룹 ▼

Test Connection

**GitLab**

☒ Enable authentication for '/project' end-point ?

GitLab connections

Connection name ?  
A name for the connection  
silapmyeon

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

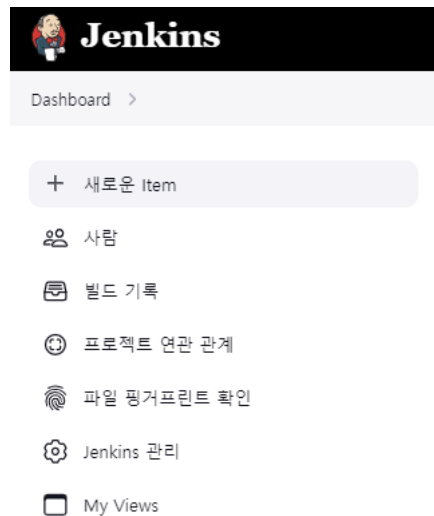
Credentials ?  
API Token for accessing GitLab  
- none -  
Add +

**API Token for GitLab access required**

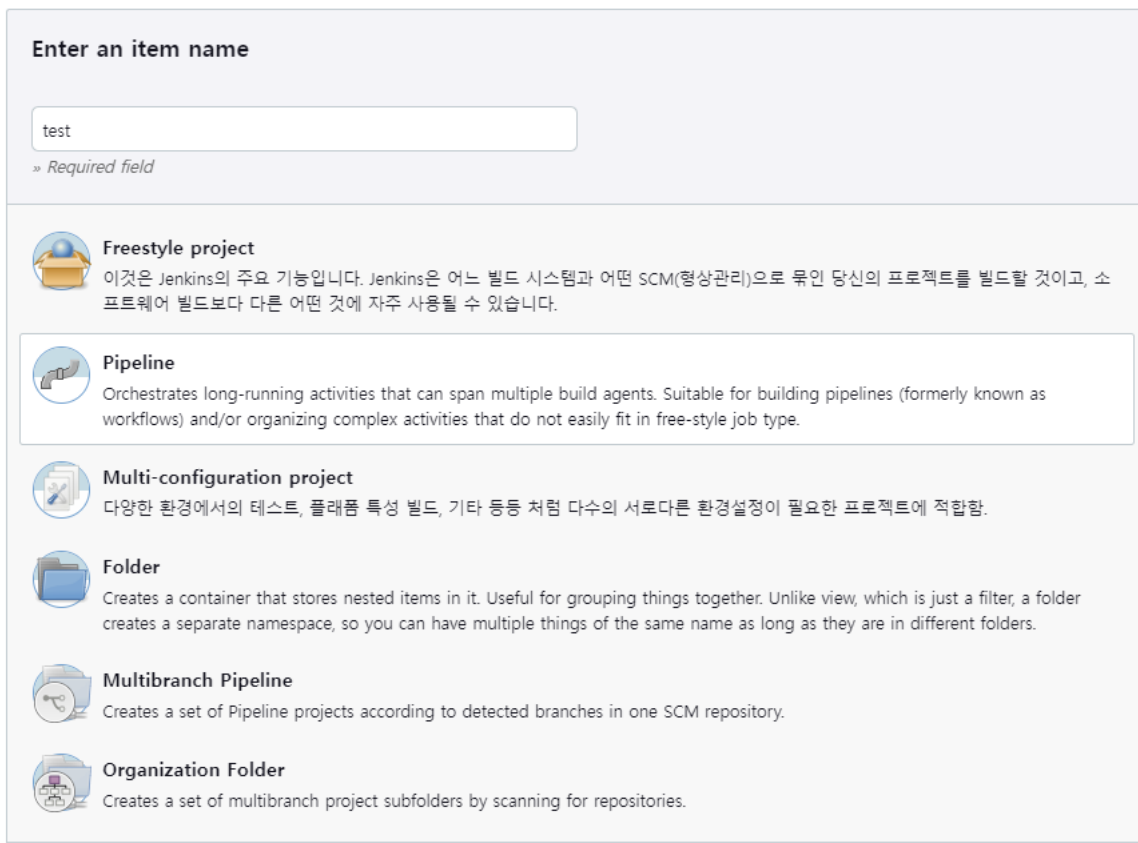
그룹 ▼

Test Connection

## • Jenkins 홈에서 새로운 Item 클릭



- item name 설정 후 Pipeline 클릭



- 만든 item의 구성 클릭

Dashboard > test >

Status

</> Changes  
▶ 지금 빌드

구성

Pipeline 삭제  
Full Stage View  
Rename  
Pipeline Syntax

Build History

추이 ▼

## Pipeline test

### Stage View

No data available. This Pipeline has not yet run.

### 고정링크

- GitLab Connection project 설정

GitLab Connection

silapmyeon

- Build Triggers 설정

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://k9b107a.p.ssafy.io:9090/project/silapmyeon-8E> ?

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

고급 ▼

- 고급 설정 Webhook 등록을 위한 Secret Token 생성하기 (Generate)



## 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

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 UL.

### Secret token

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

### Trigger

☒ Push events

☐ All branches

☒ Wildcard pattern

Wildcards such as `*-stable` or `production/*` are supported.

☐ Regular expression

☐ 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.

- Jenkins Pipeline 설정 (develop 브랜치의 BE/interview/Jenkinsfile)

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssfy.com/s09-final/S09P318107.git

Credentials ?

holizon9@naver.com/\*\*\*\*\*

Add ~

고급 ▼

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/develop

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add ~

Script Path ?

BE/interview/Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

저장 Apply

- 이런식으로 SpringBoot, React, FastAPI를 develop브랜치에서 폴더로 구분하여 설정
- SpringBoot는 /jenkins/workspace/Jenkins-item이름/프로젝트이름/src/main/으로 접속 후

```
cd /jenkins/workspace/Jenkins-item이름/프로젝트이름/src/main/
sudo mkdir resources
cd resources

#applicaion.yml 복사해서 넣은 후 springboot 빌드 다시해야 적용됨.
sudo vim applicaion.yml
```

- applicaion.yml

```
server:
  port: 8080
  servlet:
    context-path: /
    encoding:
      charset: UTF-8
      enabled: true
      force: true

spring:
  datasource:
```

```

driver-class-name: com.mysql.cj.jdbc.Driver
url: jdbc:mysql://k9b107.p.ssafy.io:31021/sam?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
username: sam
password: dlsxjqbaktmxj107
output:
  ansi.enabled: always
jpa:
  database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
  show-sql: true
  hibernate:
    ddl-auto: validate
  properties:
    hibernate:
      format_sql: true
data:
  mongodb:
    uri: mongodb://ssafy:ssafy@k9b107.p.ssafy.io/test?authSource=admin

redis:
  host: k9b107a.p.ssafy.io
  port: 6379
  password: dlsxjqbaktmxj107

jwt:
  secret: dlsxjqbaktmxj107dlsxjqbaktmxj107dlsxjqbaktmxj107
  atk-live: 600000 # 10분 = 1000 * 60 * 10
  #   atk-live: 60000 # 1분 = 1000 * 60 * 1
  rtk-live: 1209600000 # 2주 = 1000 * 60 * 60 * 24 * 14
servlet:
  multipart:
    max-file-size: 200MB
    max-request-size: 200MB
    enabled: true

security:
  oauth2:
    client:
      registration:
        google: # /oauth2/authorization/google 이 주소를 동작하게 한다.
          client-id: 727026468286-lbbjio1iv67c0apom1hs63gjc0e124qf.apps.googleusercontent.com
          client-secret: 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
          scope:
            - email
            - profile

        # 네이버는 OAuth2.0 공식 지원대상이 아니라서 provider 설정이 필요
        # 요청주소도 다르고, 응답 데이터도 다름
        naver:
          client-id: Wrd4kcDJIuRrce8KHy9
          client-secret: zn4RKoXAMb
          scope:
            - name
            - email
            - profile_image
          client-name: Naver
          authorization-grant-type: authorization_code
          redirect-uri: http://silapmyeon.com:8080/login/oauth2/code/naver

        kakao:
          client-id: 6aa51cf8b9ad58569c97442122a5e93e
          redirect-uri: http://silapmyeon.com:8080/login/oauth2/code/kakao
          client-authentication-method: POST
          authorization-grant-type: authorization_code
          scope: profile_nickname, profile_image, account_email #동의 항목
          client-name: Kakao

      provider:
        naver:
          authorization-uri: https://nid.naver.com/oauth2.0/authorize
          token-uri: https://nid.naver.com/oauth2.0/token
          user-info-uri: https://openapi.naver.com/v1/nid/me
          user-name-attribute: response # 회원정보를 json의 response 키값으로 리턴

        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

cloud:
  aws:
    credentials:
      access-key: AKIAZTK30LBDMODIVKOV
      secret-key: Qed50cW9nyNkv+1f0d016pLCxcyZicHujjETBsk

```



```

s3:
  bucket: yeongki
  region:
    static: ap-northeast-2
    auto: false
  stack:
    auto: false
logging:
  level:
    org.springframework.data.mongodb.core.MongoTemplate: DEBUG

url:
  frontend: "https://silapmyeon.com"
  backend: "http://localhost:8080"

notification:
  mattermost:
    user-name: 실전압축면접 에러 알림 봇
    enabled: true # mmSender를 사용할 지 여부, false면 알림이 오지 않는다
    webhook-url: https://meeting.ssafy.com/hooks/qzb71x9xjjdebye57dm4hhoh4y
    channel: # 기본 설정한 채널이 아닌 다른 채널로 보내고 싶을 때 기입한다
    pretext: Back-end Exception # attachments의 상단에 나오게 되는 일반 텍스트 문자
    color: red # attachment에 왼쪽 사이드 컬러. default=red
    author-name: silapmyeon # attachment의 상단에 나오는 이름
    author-icon: https://www.radware.com/RadwareSite/MediaLibraries/Images/Cyberpedia/Bot%20Manager/types-of-bots.jpg # author-icon 왼쪽
    footer: # attachment에 하단에 나올 부분. default=현재 시간

```

### ▼ 1. Jenkinsfile : pipeline 작성 (옆에 화살표 누르면 토글로 파일 코드 있음)

- spring boot 폴더에 Jenkinsfile만들어서 pipeline작성하기

#### - pipeline 구성

1. Springboot build
  - jar 파일 생성!!
2. Docker image build
  - docker 이미지 생성
3. Deploy 배포!
  - 백엔드 Jenkinsfile

```

//자동배포 버전
pipeline {
  agent any

  stages {
    stage('Springboot build') {
      steps {
        dir('BACKEND'){
          sh '''
            echo 'springboot build'
            chmod +x gradlew
            ./gradlew clean build //스프링 부트 빌드, .jar파일 생성
          '''
        }
      }
    }
    stage('Dockerimage build') {
      steps {
        dir('BACKEND'){
          sh '''
            echo 'Dockerimage build'
            docker build -t docker-springboot:0.0.1 ./도커 이미지파일 빌드
          '''
        }
      }
    }
    stage('Deploy') {
      steps {
        dir('BACKEND'){
          sh '''
            echo 'Deploy' //배포
            // 스프링 구동

            //최초 처음에 터미널에서 run한번 해줘야해 그 다음부터는 빌드할때마다 자동으로 멈추고 재실행 반복해줌
            //docker run -d -p 8080:8080 --name springboot docker-springboot:0.0.1

            docker stop springboot
          '''
        }
      }
    }
  }
}

```

```
}  
}  
}  
}  
}    docker rm springboot  
      docker run -d -p 8080:8080 --name springboot docker-springboot:0.0.1  
      ''
```

- 프론트엔드 Jenkinsfile

```

pipeline {
    agent any

    tools {
        nodejs "Node18" // 여기서 "Node18"은 위에서 설정한 Node.js의 이름입니다.
    }

    stages {

        stage('React build') {
            steps {
                dir('frontend') {
                    echo 'React build'
                    sh 'npm install' // --save 옵션은 더 이상 필요하지 않습니다.
                    sh 'CI=false npm run build'
                }
            }
        }

        stage('Dockerimage build') {
            steps {
                dir('frontend') {
                    sh '''
                    echo 'Dockerimage build for React'
                    docker build -t docker-react:0.0.1 .
                    '''
                }
            }
        }

        stage('Deploy') {
            steps {
                sh '''
                echo 'Deploy React' // 배포
                docker stop react
                docker rm react
                docker run -d -p 3000:3000 --name react docker-react:0.0.1

                '''
            }
        }
    }
}

```

- FastAPI Jenkinsfile

```

pipeline {
    agent any
    stages {
        stage('GitHub Clone') {
            steps {
                // Git 클론 작업
                git branch: 'develop', credentialsId: 'jenkins_token', url: 'https://lab.ssafy.com/s09-final/S09P31B107.git'
            }
        }
        stage('Deployment') {
            steps {
                sshagent(credentials: ['ssh_key']) {
                    sh '''
                        echo "Connecting to remote server"
                        ssh -o StrictHostKeyChecking=no ubuntu@3.36.76.73

                        echo "Creating remote directory if it doesn't exist"
                        ssh ubuntu@3.36.76.73 "mkdir -p /home/ubuntu/fastapi"

                        echo "Copying code to remote server"
                        scp -r $WORKSPACE/FastAPI/ ubuntu@3.36.76.73:/home/ubuntu/fastapi/
                    '''
                }
            }
        }
    }
}

```

```
    echo "Executing deploy script on remote server"
    ssh -t ubuntu@3.36.76.73 "./deploy_fastapi.sh"
  }
}
}
```

## ▼ 2. Dockerfile

- spring boot 폴더에 Dockerfile만들기

Dockerfile: Docker build 할 때 Docker가 자동으로 Dockerfile읽어서 도커 이미지 빌드 할 수 있도록 함

- 백엔드 Dockerfile

```
FROM openjdk:17-jdk-slim //본인 jdk로 설정!
VOLUME /tmp
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- 프론트엔드 Dockerfile

```
FROM node:18.16.1

RUN npm install -g serve

RUN mkdir ./build
ADD ./build ./build

ENTRYPOINT ["serve", "-s", "build"]
```

- FastAPI Dockerfile

```
FROM python:3.9

COPY ./FastAPI/requirements.txt /fastapi/requirements.txt

RUN ls -al

RUN pip install --upgrade pip

RUN apt-get update

WORKDIR /fastapi

RUN pip install --no-cache-dir --upgrade -r /fastapi/requirements.txt

COPY ./FastAPI /fastapi

RUN ls -al

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

## nginx 설치

```
#nginx 설치
sudo apt install nginx

#nginx 설치 상태 확인
sudo systemctl status nginx
```

```
#nginx 실행 시작/중지
sudo systemctl start nginx
sudo systemctl stop nginx
```

```
#ssl설정

#let's Encrypt 설치
sudo apt-get install letsencrypt

#certbot 설치
sudo apt-get install certbot python3-certbot-nginx

#certbot 동작 (이메일 입력, 약관 동의Y, 이메일 동의 Y or N, 도메인입력)
sudo certbot --nginx

##방화벽 기본 포트 설정
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
# 필요한 포트 열어주기
sudo ufw allow 3306(mysql), 8080(Springboot), 8000(fastapi), 6379(redis), 27017(mongoDB)
```

## • 도메인 적용 전

```
sudo vim /etc/nginx/sites-available/nginx.conf

#파일 수정
server {
    listen 80; #80포트로 받을 때
    server_name k9b107a.p.ssafy.io; # 없을 경우 localhost
    return 301 https://k9b107a.p.ssafy.io$request_uri;
}

server {
    listen 443 ssl http2;
    server_name k9b107a.p.ssafy.io;

    # ssl 인증서 적용하기
    ssl_certificate /etc/letsencrypt/live/k9b107a.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k9b107a.p.ssafy.io/privkey.pem;

    location / {
        proxy_pass http://localhost:3000;
    }

    location /api { # location 이후 특정 url을 처리하는 방법을 정의
        proxy_pass http://localhost:8080; # Request에 대해 어디로 리다이렉트하는지
        proxy_redirect off;
        charset utf-8;

        proxy_http_version 1.1;
        proxy_set_header Connection "upgrade";
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Host $http_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-NginX-Proxy true;
    }
}
```

## • 도메인 적용 후

```
server {
    if ($host = silapmyeon.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80; # 80포트로 받을 때
    server_name silapmyeon.com; # 없을 경우 localhost
    return 301 https://silapmyeon.com$request_uri;
}

server {
    listen 443 ssl http2;
    server_name silapmyeon.com;

    # SSL 인증서 적용하기
    ssl_certificate /etc/letsencrypt/live/silapmyeon.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/silapmyeon.com/privkey.pem; # managed by Certbot
```

```

location / {
    proxy_pass http://localhost:3000;

}

location /api/ {
    proxy_pass http://localhost:8080/;
    charset utf-8;
    proxy_redirect off;
    proxy_http_version 1.1;
    proxy_set_header Connection "upgrade";
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Host $http_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-NginX-Proxy true;
}

location /api/interview {
    proxy_pass http://localhost:8080/interview;
}
}

```

```

#sites-enabled에 심볼릭 링크 생성
sudo ln -s /etc/nginx/sites-available/nginx.conf /etc/nginx/sites-enabled

#conf 파일 오류 없는지 확인 하고 nginx 실행
sudo nginx -t

```

```

#!!!!!!nginx 완전히 날릴때
sudo apt-get purge nginx nginx-common

```