



# 포팅 매뉴얼

## 목차

### 목차

1. 개발 환경
2. 설정파일 및 환경 변수 정보
3. 인증서 발급
4. Docker 설치
  - [Sudo 명령어 없이 치기](#)
  - [docker 모드](#)
  - [Docker-compose 설치](#)
  - [docker-compose.yml에 작성된 모든 컨테이너 생성, 실행](#)
  - [docker-compose.yml에 작성된 모든 컨테이너 중지, 종료](#)
5. Jenkins 설치
  - [Jenkins 플러그인 설치](#)
6. mysql 컨테이너 생성
7. nginx 컨테이너 생성
8. 프론트 Vue 배포
9. 백 Spring 배포
10. 깃랩 웹훅 적용

## 1. 개발 환경

- **Server** : Ubuntu 20.04 LTS
- **JDK** : OpenJDK8
- **Node.js** : 19.6.0
- **Nginx** : 1.23.3
- **MySQL** : 8.0.31
- **Jenkins** : 2.375.2
- **Vscode** : 1.73.1
- **IntelliJ** : 2022.03

## 2. 설정파일 및 환경 변수 정보

### Spring

application.properties

```
# MySQL 설정
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# DB Source URL
spring.datasource.url=<DB url>
# DB Username
spring.datasource.username= =<DB 유저>
# DB Password
spring.datasource.password= =<DB 비밀번호>
# true 설정 시 JPA 쿼리문 확인 가능
spring.jpa.show-sql=false
# DDL(Create, Alter, Drop) 정의 시 DB의 고유 기능 사용 가능
spring.jpa.hibernate.ddl-auto=update
# JPA의 구현체인 Hibernate가 동작하면서 발생한 SQL의 가독성 높여줌
```

```

spring.jpa.properties.hibernate.format_sql=false

# Swagger (OpenApi)
springdoc.api-docs.path=/api-docs
springdoc.swagger-ui.path=/swagger-ui
springdoc.swagger-ui.disable-swagger-default-url=true
# JWT
jwt.salt=<JWT salt 값>
jwt.expiration=5
jasypt.encryptor.bean=jasyptStringEncryptor
# SMTP
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=<SMTP 주소>
spring.mail.password=<SMTP 비밀번호>
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
# multipart 설정
spring.servlet.multipart.max-file-size: 100MB
spring.servlet.multipart.max-request-size: 100MB
# aws 설정
cloud.aws.region.static=ap-northeast-2
cloud.aws.stack.auto=false
# IAM Access key
credentials.iam.accessKey=<IAM access 키>
credentials.iam.secretKey=<IAM secret 키>
# S3 정보
s3.bucket=<S3 이름>
s3.region=<S3 지역>
s3.url=<S3 url>

```

### 3. 인증서 발급

```

$ sudo apt-get install letsencrypt
$ sudo letsencrypt certonly standalone d www 제외한 도메인 이름
이메일 작성 후 Agree
뉴스레터 수신 여부 Yes/No
$ ssl_certificate /etc/letsencrypt/live/ 도메인이름 /fullchain.pem;
$ ssl_certificate_key /etc/letsencrypt/live/ 도메인이름 /privkey.

```

### 4. Docker 설치

도커는 도커데몬 위에서 이미지를 컨테이너 상태로 실행시키고, 호스트 서버 혹은 컨테이너 간의 상호작용을 목표로 합니다.

▶ apt 업데이트 `$ apt-get update`

▶ 도커 설치 `$ apt-get install docker.io`

▶ 링크 생성 `$ ln -sf /usr/bin/docker.io /usr/local/bin/docker`

▶ 도커 버전 확인 `$ docker -v`

▶ 도커 데몬 실행 `$ systemctl start docker`

#### Sudo 명령어 없이 치기

`sudo usermod -aG docker ubuntu`

#### docker 모드

`sudo chmod 666 /var/run/docker.sock`

#### Docker-compose 설치

```
sudo curl -L https://github.com/docker/compose/releases/latest/\
download/docker-compose-$(uname -s) sudo chmod +x usr/local/bin/docker-compose
```

## 설치 확인

```
$docker-compose -v
```

## docker-compose.yml에 작성된 모든 컨테이너 생성, 실행

```
$docker-compose up
```

## docker-compose.yml에 작성된 모든 컨테이너 중지, 종료

```
$docker-compose down
```

# 5. jenkins 설치

### ▶ 도커허브로 부터 jenkins/jenkins:its 이미지 pull

```
$ docker pull jenkins/jenkins:its
```

### ▶ 젠킨스 컨테이너 실행

```
$ docker run -d --name jenkins -p 외부포트:내부포트 -v /jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock -u root jenkins/jenkins:its .
```

- **v /jenkins:/var/jenkins\_home**
  - 젠킨스 컨테이너의 설정을 호스트 서버와 공유함으로써, 컨테이너가 삭제되는 경우에도 설정을 유지
- **v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock**
  - 젠킨스 컨테이너에서도 호스트 서버의 도커를 사용하기 위한 바인딩
- **p 외부포트:내부포트**
  - 젠킨스의 포트를 호스트 서버와 매핑.

### ▶ 도커 컨테이너의 실행 상태 확인 \$ docker ps

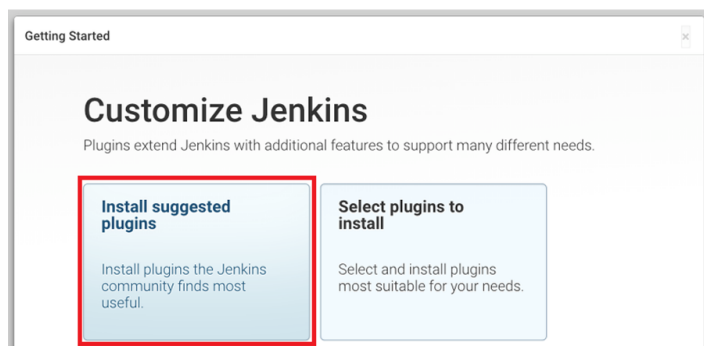
## 도커 로그에서 확인 가능

```
$ docker logs jenkins
```

## \$docker logs jenkins 비밀번호 확인후 복사

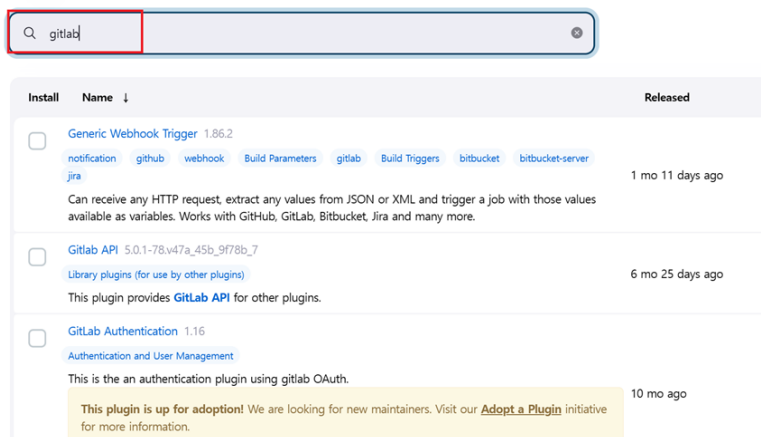


## 좌측 클릭(추천 플러그인 설치)



## Jenkins 플러그인 설치

### Plugins



- Gitlab API
- Gitlab

## 6. mysql 컨테이너 생성

파일 위치 : /home/ubuntu/mysql

## docker-compose.yml

```
version : "3"
services :
  mysql:
    image: mysql
    container_name : <컨테이너 이름>
    environment:
      MYSQL_ROOT_PASSWORD: <mysql 비밀번호>
      MYSQL_DATABASE: <mysql 스키마>
    ports:
      - 3306:3306
    restart: unless-stopped
    networks:
      - <네트워크 이름>

networks:
  <네트워크 이름>:
    external: true
```

## 컨테이너 생성, 실행

```
$docker-compose up
```

## 7. nginx 컨테이너 생성

nginx.conf

- 파일 위치 : home/ubuntu/nginx/conf

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

    server_name <서버 도메인>;
    server_tokens off;

    access_log /var/log/nginx/reverse-access.log;
    error_log /var/log/nginx/reverse-error.log;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name <서버 도메인>;
    server_tokens off;

    ssl_certificate      /etc/letsencrypt/live/<서버 도메인>/fullchain.pem;
    ssl_certificate_key  /etc/letsencrypt/live/<서버 도메인>/privkey.pem;

    location / {
        proxy_pass http://<서버 도메인>:<vue 포트번호>;
        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;
    }

    location /api/ {
        rewrite ^/api(.*)$ $1 break;
        proxy_pass http://<서버 도메인>:<springboot 포트번호>;
        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;
    }
}

}
```

docker-compose.yml

- 파일 위치: /home/ubuntu/nginx

```

version: '3.4'
services:
  nginx:
    image: nginx:latest
    container_name : nginx
    volumes:
      - ./conf:/etc/nginx/conf.d
      - /etc/letsencrypt:/etc/letsencrypt
    ports:
      - 80:80
      - 443:443
    networks:
      - ssafy-network

networks:
  ssafy-network:
    external: true

```

## 컨테이너 생성, 실행

```
$docker-compose up
```

## 8. 프론트 Vue 배포

### Dockerfile

- 파일 위치 front/Dockerfile

```

# Use an official Node.js runtime as the base image
FROM node:19

# Set the working directory in the container
WORKDIR /app

# Copy the package.json and package-lock.json files
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the React project files
COPY . .

# Build the Vue project
RUN npm run build

# Use Nginx as the web server
FROM nginx

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

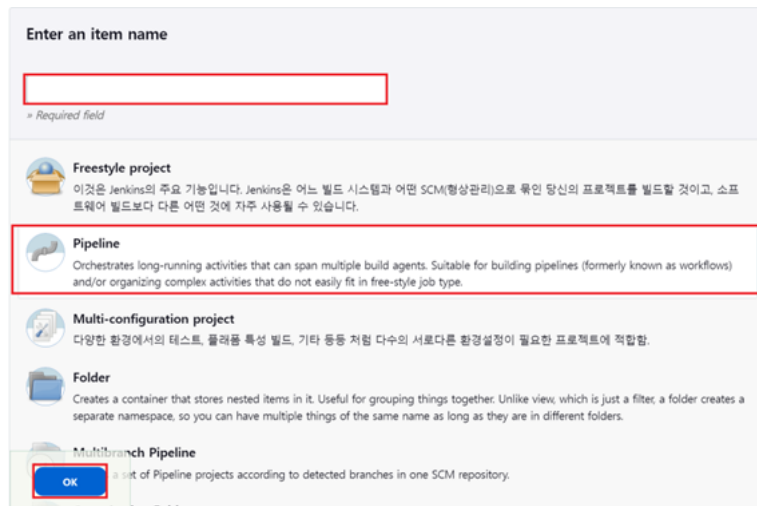
# Copy the build files to the Nginx web root directory
COPY --from=0 /app/dist /usr/share/nginx/html

# Expose port 80
EXPOSE 80

# Start Nginx
CMD ["nginx", "-g", "daemon off;"]

```

### Jenkins에서 item 생성



[프로젝트 이름]작성후 **pipeline** 선택

*Vue pipeline* 작성

```

pipeline { // 파이프라인의 시작
    agent any

    environment {
        GIT_URL = "깃 주소"
    }

    stages {
        stage('Pull') {
            steps {
                git url: "${GIT_URL}", branch: "fe", poll: true, changelog: true, credentialsId: '크래덴셜 id'
            }
        }
        stage('Stop and Remove Old Container - Front') {
            // 안되면 try catch 적용해보기
            steps {
                script{
                    try {
                        sh 'docker stop $(docker ps -q --filter ancestor=front:latest)'
                        sh 'docker rm $(docker ps -a -q --filter ancestor=front:latest)'
                    } catch (Exception e) {
                        echo "An error occurred: ${e}"
                    }
                }
            }
        }
        post {
            success {
                echo 'Stop and Remove success!'
            }
        }
    }

    stage('Build Frontend') {
        // 도커 빌드
        // agent any
        steps {
            echo 'Build Frontend'

            dir ('front/logit'){
                sh """
                docker build . -t front:latest
                """
            }
        }
        post {
            // steps 끝나면 post온다
            // 빌드하다 실패하면 error 뽐고, 나머지 과정 하지말고 종료
            failure {
                error 'This pipeline stops here...'
            }
        }
    }
    stage('Deploy New Frontend Container') {

```

```

    steps {
        sh 'docker run --network <네트워크 이름> --name front -p 포트번호 -d <이미지 이름>'
    }

    post {
        success {
            echo 'Deploy Frontend success!'
        }
    }
}

stage('Stop and Remove Old Container - Back') {
    // 안되면 try catch 적용해보기
    steps {
        script{
            try {
                sh 'docker stop $(docker ps -q --filter ancestor=backend:latest)'
                sh 'docker rm $(docker ps -a -q --filter ancestor=backend:latest)'
            } catch (Exception e) {
                echo "An error occurred: ${e}"
            }
        }
    }
    post {
        success {
            echo 'Stop and Remove success!'
        }
    }
}

stage('Finish') {
    steps{
        sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
    }
}
}
}

```

## 9. 백 Spring 배포

### Dockerfile

- back/Dockerfile

```

FROM adoptopenjdk/openjdk8
ARG JAR_FILE=/build/libs/back-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /build/libs/back-0.0.1-SNAPSHOT.jar

ENTRYPOINT ["sh", "-c", "java ${JAVA_OPTS} -jar /build/libs/back-0.0.1-SNAPSHOT.jar"]

```

Jenkins에서 item 생성





**Enter an item name**


back\_project


» Required field


---


 **Freestyle project**  
이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

 **Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

 **Multi-configuration project**  
다양한 환경에서의 테스트, 플래폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

 **Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

 **Multibranch Pipeline**  
a set of Pipeline projects according to detected branches in one SCM repository.

 **Organization Folder**

**OK**

프로젝트 이름 작성후 > pipeline 선택!

Springboot pipeline

```

pipeline {
    agent any

    environment {
        GIT_URL = "깃 주소"
    }

    stages {
        stage('Pull') {
            steps {
                git url: "${GIT_URL}", branch: "be", poll: true, changelog: true, credentialsId: 'plzzzzzz'
            }
        }

        stage('GradleBuild') {
            steps {
                dir('back'){
                    sh 'chmod +x gradlew'
                    sh './gradlew -Djasypt.encryptor.password=SPRINGLOGIT clean bootJar'
                }
            }
        }

        stage('Build') {
            steps {
                dir('back'){
                    sh 'docker stop back || true && docker rm back || true'
                    sh 'docker build -t back .'
                }
            }
        }

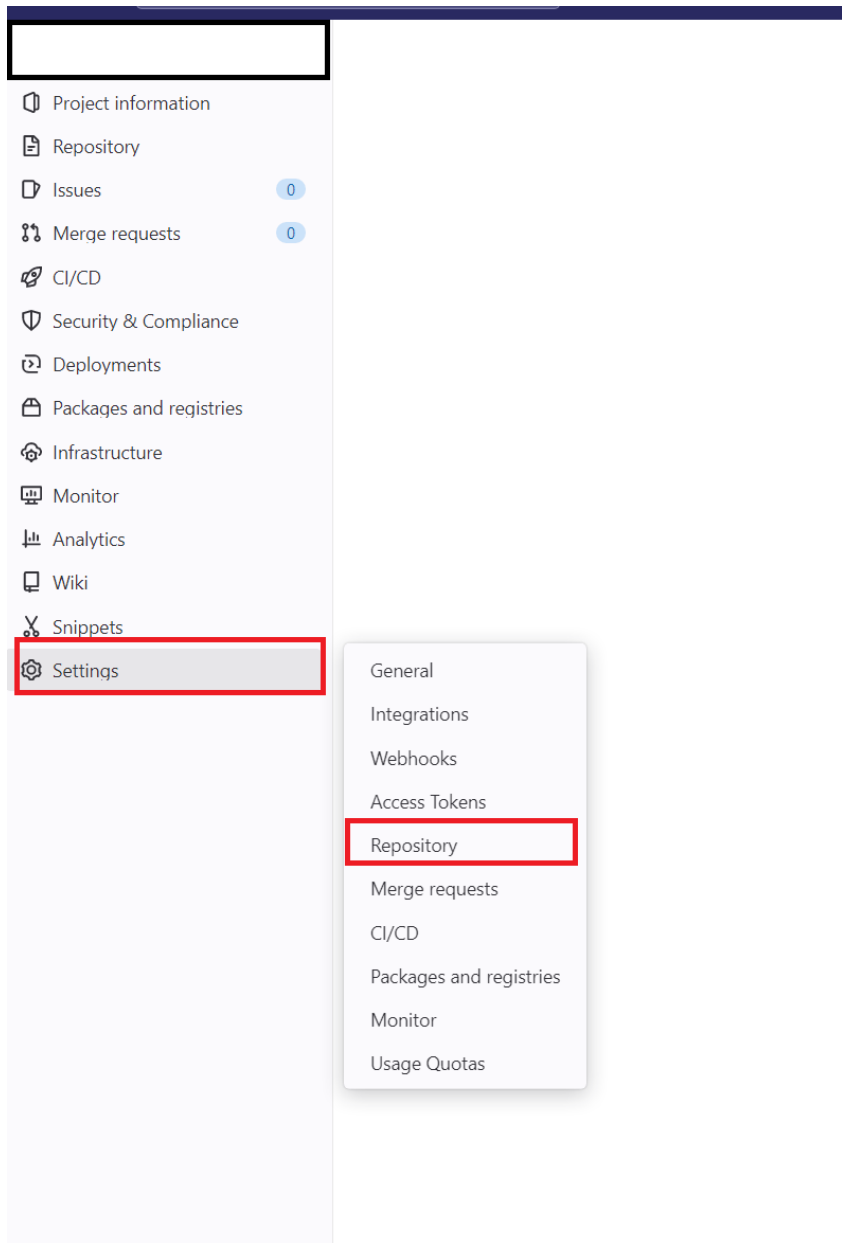
        stage('Deploy') {
            steps{
                sh 'docker run --name back -d --network ssafy-network -e JAVA_OPTS="-Djasypt.encryptor.password={jasypt 설정한 비밀번호}.'
            }
        }

        stage('Finish') {
            steps{

```

```
}  
  }  
}  
}  
sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
```

## 10. 깃랩 웹훅 적용



jenkins Spring project 레포 주소 참조

### Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: [http://\[REDACTED\]project/springboot\\_repo](http://[REDACTED]project/springboot_repo) ?

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

저장

Apply

고급 → Secret Token [Generate](#) 버튼 클릭후 저장!

Secret token ?

[REDACTED]

Generate

Clear

깃랩 → Webhooks

## 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.

### Secret token

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

### Trigger

☒ Push events

Push to the repository.

☐ 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

A wiki page is created or updated.

☐ Deployment events

`url` : 젠킨스 프로젝트

`Secret token` : 젠킨스 프로젝트에서 생성한 Secret Token