



# 포팅 메뉴얼

## 기술 스택

[이슈 관리](#)  
[형상 관리](#)  
[커뮤니케이션](#)  
[개발 환경](#)  
[상세 사용](#)

## AWS EC2 Ubuntu 서버 세팅

[Docker](#)  
[NGINX](#)  
[Cerybot\(https 설정\)](#)  
[Jenkins](#)

## Database

[MySQL](#)  
[Redis](#)

## 프로퍼티 정의

[nginx 설정 파일](#)  
[백엔드 설정 파일](#)

## Jenkins Pipeline 구축

[Jenkins 환경 설정](#)  
[Pipeline 구성](#)

## 기술 스택

### 이슈 관리

Jira

### 형상 관리

Girllab

### 커뮤니케이션

Mattermost, Notion, Figma

### 개발 환경

1. OS
  - Windows 10
2. Web Server
  - NGINX 1.18.0
3. WAS
  - Apache Tomcat 9.0.65
4. IDE
  - IntelliJ IDEA 2021.2.4
  - Android Studio Dolphin 2021.3.1
5. DB
  - MySQL 8.0.31
  - Redis 5.0.7
6. 배포

- AWS EC2 Ubuntu 20.04 LTS
- Docker 20.10.20
- Jenkins 2.374

## 상세 사용

### 1. Backend

- Java 11
- Spring Boot 2.7.5
- Spring Data JPA 2.7.5
- Spring Data Redis 2.7.5
- Gradle 7.5.1
- Lombok 1.18.24
- Websocket 5.3.23

### 2. Frontend

- kotlin 1.7.20
- retrofit2 2.2.0
- glide 4.12.0
- gradle 7.4.2
- SDK
  - min sdk 28
  - target sdk 32

## AWS EC2 Ubuntu 서버 세팅

- root 계정 로그인

```
su root
```

## Docker

- 설치
  - 사전 패키지 다운로드

```
# apt 패키지 업데이트
apt update

# 사전 패키지 설치
apt-get install -y ca-certificates \
    curl \
    software-properties-common \
    apt-transport-https \
    gnupg \
    lsb-release

# 위의 명령어로 패키지 설치가 잘 안된다면 각각 나눠서 설치해도 됨
apt-get install -y ca-certificates
apt-get install -y curl
apt-get install -y software-properties-common
```

```
apt-get install -y apt-transport-https
apt-get install -y gnupg
apt-get install -y lsb-release
```

#### ◦ gpg 키 다운로드

리눅스 패키지 툴이 프로그램 패키지가 유효한지 확인하기 위해, 프로그램 설치 전에 gpg 키로 검증하는 과정을 거치기 때문에 gpg 키를 받아야 함

```
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/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/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

#### ◦ 도커 설치

```
apt update
apt install docker-ce docker-ce-cli containerd.io docker-compose
```

위의 명령어로 설치하는 과정에서 문제가 발생할 경우 다음 명령어들을 순서대로 수행 후 재시도

```
# 패키지 리스트 업데이트
sudo apt update

# apt가 https를 통해 패키지를 사용하기 위한 선행 패키지들을 설치
sudo apt install apt-transport-https ca-certificates curl software-properties-common

# 시스템 상에 공식 도커 저장소 접속을 위한 GPG 키를 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# 도커 저장소를 apt 소스에 추가
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 추가된 내용을 적용하기 위해 다시 패키지 리스트 업데이트
sudo apt update

# ubuntu 기본 저장소가 아닌 도커 저장소를 사용하도록 설정
apt-cache policy docker-ce
```

#### ◦ 관련 파일

##### ■ `Dockerfile` (Backend용)

###### ▼ 코드 보기

```
FROM gradle:7.5WORKDIR /var/jenkins_home/workspace/nawanolza-test/nawanolza/server

COPY ./ ./

ENV SPRING_DATASOURCE_URL=${SPRING_DATASOURCE_URL} \
    SPRING_DATASOURCE_USERNAME=${SPRING_DATASOURCE_USERNAME} \
    SPRING_DATASOURCE_PASSWORD=${SPRING_DATASOURCE_PASSWORD} \
    SPRING_REDIS_HOST=${SPRING_REDIS_HOST} \
    SPRING_REDIS_PORT=${SPRING_REDIS_PORT} \
    SPRING_REDIS_PASSWORD=${SPRING_REDIS_PASSWORD} \
    ENV_IDLE_PROFILE=${ENV_IDLE_PROFILE}

RUN gradle clean bootJar --no-daemon

CMD ["java", "-jar", "-Dspring.profiles.active=${ENV_IDLE_PROFILE}", "-Duser.timezone=Asia/Seoul", "./build/libs/server-0.0.1-SNAPSHOT.jar"]
```

##### ■ `docker-compose.yml` (Jenkins용)

###### ▼ 코드 보기

```

version: "3"

services:
  jenkins:
    image: jenkins/jenkins:jdk11
    container_name: jenkins
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /jenkins:/var/jenkins_home
    ports:
      - "9090:8080"
    privileged: true
    user: root

```

- jenkins: 만들어지는 컨테이너의 이름. 자유롭게 설정 가능
- image: 컨테이너에 올라갈 이미지
- volumes: 컨테이너가 내려갔을 때 파일이 소실되는걸 방지하기 위한 로컬 저장 경로
- ports: {외부 ↔ 컨테이너 통신용 포트}:{컨테이너 ↔ 프로그램 통신용 포트}
- privileged: 컨테이너가 시스템 주요 자원에 접근할 수 있도록 권한을 부여하는 옵션
- user: 컨테이너 접속시 디폴트 사용자
- `docker-compose.yml` 사용 명령어

```

# docker-compose 명령어로 docker-compose.yml 파일을 사용할 수 있음
# 기본적으로 현재 폴더를 기준으로 하고, -f 옵션을 통해 파일의 위치를 지정할 수 있음
# up으로 컨테이너를 올릴 수 있고, down으로 컨테이너를 내릴 수 있음
# --build는 캐싱된 이미지를 사용하지 않고 무조건 빌드를 하고 시작한다는 의미
# -d는 컨테이너를 백그라운드에서 올리겠다는 의미
docker-compose -f docker-compose.yml up -d --build
docker-compose -f docker-compose.yml down

```

## NGINX

- 설치

```
apt install nginx
```

- NGINX 상태확인/실행/중지

```

// nginx 상태확인
service nginx status

// nginx 실행
service nginx start

// nginx 재실행
service nginx restart

// nginx 중지
service nginx stop

```

nginx 실행 과정에서 80번 포트가 이미 사용중이라고 나오면서 실행에 실패할 경우 다음 명령어를 수행

```

# 80번 포트를 사용중인 프로세스들을 확인
lsof -i :80

# 80번 포트를 사용중인 프로세스들을 종료
fuser -k 80/tcp

```

- 삭제

진행 과정에서 문제가 생겨nginx를 완전히 삭제할 필요가 있을 경우 다음 명령어 수행

```
apt-get remove --purge nginx nginx-full nginx-common
```

## Cerybot(https 설정)

- 사전 준비

https가 적용될 서버 블록이 필요하기 때문에 우선 NGINX를 이용해서 서버 블록을 지정

```
# 도메인 이름으로는 k7d103.p.ssafy.io를 사용
mkdir -p /var/www/{도메인이름}/html
chown -R $USER:$USER /var/www/{도메인이름}/html
chmod -R 755 /var/www/{도메인이름}
```

/etc/nginx/sites-available 에 있는 default 파일을 삭제한 뒤 k7d103.p.ssafy.io.conf 를 만들고 아래 명령어로 /etc/nginx/sites-enabled 에도 심볼릭 링크로 연결

```
ln -s /etc/nginx/sites-available/k7d103.p.ssafy.io.conf /etc/nginx/sites-enabled/
```

- 설치

```
snap install --classic certbot
```

- SSL 인증서 발급

```
certbot certonly --nginx -d k7d103.p.ssafy.io
```

만약 certbot 명령어를 인식하지 못하면 다음 명령어로 환경변수에 등록한 후 재시도

```
export PATH=$PATH:/snap/bin
```

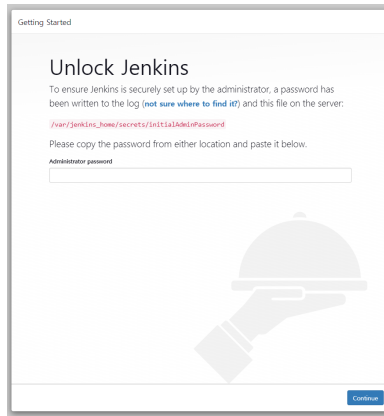
명령어를 입력하면 간단한 설정을 하고 /etc/letsencrypt/live/{도메인이름} 디렉토리에 다음 파일들이 생김

- cert.pem : yourdomain.com의 인증서(public key)
- chain.pem : Let's encrypt의 Intermediate 인증서
- fullchain.pem : cert.pem 과 chain.pem 을 합친 결과
- privkey.pem : cert.pem publickey에 대응하는 비밀키

파일들이 성공적으로 생성됐다면 본 문서의 '프로퍼티 정의' → 'nginx 설정 파일' → 'k7d103.p.ssafy.io'의 내용을 k7d103.p.ssafy.io 파일에 붙여넣으면 됨

## Jenkins

- 본 문서의 'Docker'에서 작성한 docker-compose.yml 파일을 docker-compose up -d 로 실행하면 젠킨스 컨테이너를 올릴 수 있음
- 컨테이너가 올라가면 브라우저에서 k7d103.p.ssafy.io:9090 으로 젠킨스 페이지에 접속 가능



- 첫 젠킨스 페이지 접속 시 입력해야하는 암호는 다음 명령어로 확인 가능

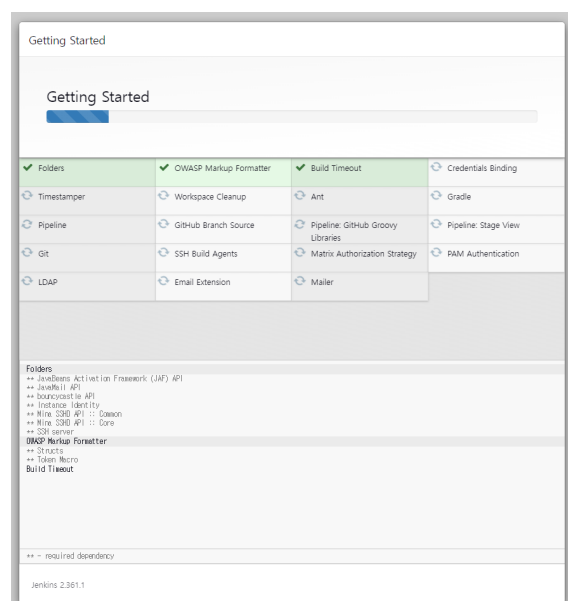
```
docker logs {컨테이너 이름}
```

# 또는

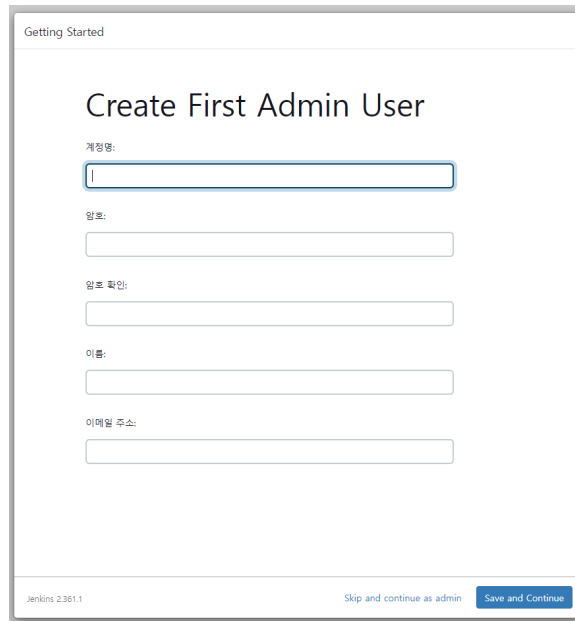
```
docker-compose logs
```

```
jenkins_cicd | 2022-10-06 07:55:21.853+0000 [id=28] INFO jenkins.install.SetupWizard#init:
jenkins_cicd |
jenkins_cicd | *****
jenkins_cicd | *****
jenkins_cicd | *****
jenkins_cicd |
jenkins_cicd | Jenkins initial setup is required. An admin user has been created and a password generated.
jenkins_cicd | Please use the following password to proceed to installation:
jenkins_cicd | {암호}
jenkins_cicd |
jenkins_cicd | This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
jenkins_cicd |
jenkins_cicd | *****
jenkins_cicd | *****
jenkins_cicd | *****
jenkins_cicd |
jenkins_cicd | 2022-10-06 07:55:55.640+0000 [id=29] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
```

- 암호를 입력하면 플러그인 설치 창이 나오는데, 왼쪽의 **Install suggested plugins**로 플러그인 설치



- 플러그인 설치가 끝나면 계정명, 암호, 이름, 이메일 주소 등을 입력해서 관리자 계정을 설정  
이때 계정명과 암호는 젠킨스 페이지에 로그인 할 때 ID와 비밀번호로 사용됨



The image shows the 'Getting Started' page of Jenkins 2.361.1. The main heading is 'Create First Admin User'. Below the heading are five input fields: '계정명:' (Username), '암호:' (Password), '암호 확인:' (Confirm Password), '이름:' (Name), and '이메일 주소:' (Email address). At the bottom right, there are two buttons: 'Skip and continue as admin' and 'Save and Continue'.

- 이후 url은 수정할 필요 없이 **save and finish** 를 하면 젠킨스 초기 설정이 끝나고 젠킨스 페이지에 접속 가능
- 젠킨스 메인에서 **Jenkins 관리** → **플러그인 관리** → **설치 가능** 으로 이동해서 다음 플러그인들을 설치
  - Gitlab
  - Gitlab API
  - Gitlab Authentication
  - Generic Webhook Trigger
  - Docker
  - Docker Commons
  - Docker Pipeline
  - Docker API
  - SSH Agent Plugin

## Database

### MySQL

- AWS EC2 서버에서 MySQL Docker Image를 받아서 container 실행

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=${password} -d -p 3306:3306 mysql
```

- 실행 중인 MySQL container에 접속

```
docker exec -it mysql bash
```

- MySQL container에서 mysql 접속

```
mysql -uroot -p${password}
```

## Redis

- 설치

```
# 패키지 업데이트
apt-get update

# Redis 설치
apt-get install redis-server

# Redis 버전 확인
redis-server --version
```

- Redis 상태확인/실행/중지

```
// redis 상태확인
service redis status

// redis 실행
service redis start

// redis 재실행
service redis restart

// redis 중지
service redis stop
```

- 설정 변경

`/etc/redis/redis.conf` 파일을 수정해서 설정을 변경할 수 있고, 설정을 수정한 뒤에는 redis를 재시작해야 함  
단, 파일 내용이 길어서 수정하고 싶은 내용을 찾기 어려울 수 있으므로 `grep` 명령어를 사용하는 것을 추천

- 최대 사용 메모리(maxmemory)와 메모리 초과 시 삭제 정책(maxmemory-policy) 설정

`grep -n "maxmemory" /etc/redis/redis.conf` 명령어로 수정하고 싶은 옵션의 위치를 찾은 뒤 수정

```
root@ip-172-26-3-19:/home/ubuntu# grep -n "maxmemory" /etc/redis/redis.conf
545:# according to the eviction policy selected (see maxmemory-policy).
555:# WARNING: If you have replicas attached to an instance with maxmemory on,
563:# limit for maxmemory so that there is some free RAM on the system for replica
566:# maxmemory <bytes>
568:# MAXMEMORY POLICY: how Redis will select what to remove when maxmemory
597:# maxmemory-policy noeviction
608:# maxmemory-samples 5
610:# Starting from Redis 5, by default a replica will ignore its maxmemory setting
622:# memory than the one set via maxmemory (there are certain buffers that may
626:# the configured maxmemory setting.
628:# replica-ignore-maxmemory yes
654:# 1) On eviction, because of the maxmemory and maxmemory policy configurations,
1040:# e      Evicted events (events generated when a key is evicted for maxmemory)
1255:# Redis LFU eviction (see maxmemory setting) can be tuned. However it is a good
```

위의 경우 566번 줄과 597번 줄의 주석을 풀고 내용을 수정하거나 그 아래에 내용을 작성해서 설정을 변경할 수 있음


```
# 메모리를 1g로 설정
maxmemory 1g

# 가장 오래된 데이터를 삭제
maxmemory-policy allkeys-lru
```

삭제 정책의 종류에 관해서는 아래 링크를 참고

### Key eviction

Overview of Redis key eviction policies (LRU, LFU, etc.) When Redis is used as a cache, it is often convenient to let it automatically evict old data as you add new data. This behavior is well known in the developer community, since it is the default behavior for the popular memcached system.

 <https://redis.io/docs/manual/eviction/>



- springboot에서 redis를 사용하기 위한 설정
  - build.gradle 의존성 추가

```
implementation 'org.springframework.boot:spring-boot-starter-data-redis'
```

- application.yml 설정 추가

```
spring:
  redis:
    host: k7d103.p.ssafy.io
    port: 6379
    password: ${spring.redis.password}
```

## 프로퍼티 정의

### nginx 설정 파일

1. `nginx.conf` (nginx 디폴트 설정)

`/etc/nginx` 폴더에 기본으로 존재하는 디폴트 설정 파일

설피했을 때 설정에서 변경 X

▼ 코드 보기

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    # multi_accept on;
}

http {

    ##
    # Basic Settings
    ##

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    # server_tokens off;

    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ##
    # SSL Settings
    ##

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: P00DLE
    ssl_prefer_server_ciphers on;

    ##
    # Logging Settings
    ##

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

    ##
    # Gzip Settings
    ##

    gzip on;
```

```

# gzip_vary on;
# gzip_proxied any;
# gzip_comp_level 6;
# gzip_buffers 16 8k;
# gzip_http_version 1.1;
# gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss

##
# Virtual Host Configs
##

include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
}

#mail {
#   # See sample authentication script at:
#   # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
#   # auth_http localhost/auth.php;
#   # pop3_capabilities "TOP" "USER";
#   # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
#   server {
#       listen     localhost:110;
#       protocol   pop3;
#       proxy      on;
#   }
#
#   server {
#       listen     localhost:143;
#       protocol   imap;
#       proxy      on;
#   }
#}

```

## 2. k7d103.p.ssafy.io (서버 설정)

/etc/nginx/sites-available 폴더에 도메인 이름에 맞춰서 새로 만들어 줄 설정 파일

/etc/nginx/sites-enabled 폴더에도 심볼릭 링크로 연결되어 있음

### ▼ 코드 보기

```

upstream backend {
    ip_hash;
    server 172.26.2.49:9002;
}

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

    server_name k7d103.p.ssafy.io;

    if ($host = k7d103.p.ssafy.io) {
        return 308 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name k7d103.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/k7d103.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k7d103.p.ssafy.io/privkey.pem;

    location / {
        root /home/ubuntu/client;
        index index.html;
    }

    location /api {
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://backend;
        proxy_redirect off;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-Proto $scheme;
        proxy_http_version 1.1;
        proxy_set_header    Host $http_host;
        proxy_set_header    Upgrade $http_upgrade;
        proxy_set_header    Connection "Upgrade";
        add_header P3P 'CP="ALL DSP COR PSAa PSDa OUR NOR ONL UNI COM NAV"';
    }
}

```

```
}
}
```

## 백엔드 설정 파일

1. `application-local.yml` (로컬 개발용 springboot 설정)

로컬에서 개발하는 개발자들을 위한 springboot 설정 파일

▼ 코드 보기

```
spring:
  profiles:
    active:
      on-profile: local
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3307/nawanolza?serverTimezone=Asia/Seoul
    username: root
    password: nawanolza1234
    hikari:
      pool-name: jpa-hikari-pool
      maximum-pool-size: 5
      jdbc-url: ${spring.datasource.url}
      username: ${spring.datasource.username}
      password: ${spring.datasource.password}
      driver-class-name: ${spring.datasource.driver-class-name}
      data-source-properties:
        rewriteBatchedStatements: true
  jackson:
    serialization:
      fail-on-empty-beans: false
  redis:
    host: localhost
    port: 6380
    password:

# JPA ??
jpa:
  generate-ddl: true
  hibernate:
    ddl-auto: validate
  show-sql: true
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MySQL8Dialect
      hbm2ddl.import_files_sql_extractor: org.hibernate.tool.hbm2ddl.MultipleLinesSqlCommandExtractor
      current_session_context_class: org.springframework.orm.hibernate5.SpringSessionContext
      default_batch_fetch_size: ${chunkSize:100}
      jdbc.batch_size: 20
      order_inserts: true
      order_updates: true
      format_sql: true
  server:
    servlet:
      context-path: /api

security:
  jwt:
    token:
      secret-key: nawanolza
      expire-length: 1800000

kakao:
  client_id: 79a60227e8d4d465f7b303d472b3cd99
  redirect_uri: http://localhost:8080/api/auth/kakao/callback

management:
  endpoints:
    web:
      base-path: /management
      path-mapping:
        health: health_check
      exposure:
        include: health, info
```

2. `docker-compose-local.yml` (로컬 개발용 데이터베이스 컨테이너)

로컬에서 개발하는 개발자들의 데이터베이스 환경을 일치시키기 위한 `docker compose` 파일

로컬 환경에서 `docker desktop` 또는 `docker-compose` 명령어를 설치하고 파일을 실행

▼ 코드 보기

```
version: '3'

services:
  redis:
    image: redis
    ports:
      - 6380:6379

  mysql:
    image: mysql
    environment:
      MYSQL_DATABASE: nawanolza
      MYSQL_ROOT_PASSWORD: nawanolza1234
      MYSQL_ROOT_HOST: '%'
    ports:
      - 3307:3306
    restart: always
    volumes:
      - ../db/docker-volume-mysql:/var/lib/mysql
      - ../db/mysql-sql:/docker-entrypoint-initdb.d
```

3. `application-prod.yml` (배포용 springboot 설정)

외부에 노출되어선 안되는 정보들을 환경변수로 관리하기 위한 springboot 설정 파일

▼ 코드 보기

```
spring:
  profiles:
    active:
      - on-profile: prod
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
  hikari:
    pool-name: jpa-hikari-pool
    maximum-pool-size: 5
    jdbc-url: ${spring.datasource.url}
    username: ${spring.datasource.username}
    password: ${spring.datasource.password}
    driver-class-name: ${spring.datasource.driver-class-name}
    data-source-properties:
      rewriteBatchedStatements: true

# JPA 설정
jpa:
  generate-ddl: true
  hibernate:
    ddl-auto: validate
  show-sql: true
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MySQL8Dialect
      hbm2ddl.import_files_sql_extractor: org.hibernate.tool.hbm2ddl.MultipleLinesSqlCommandExtractor
      current_session_context_class: org.springframework.orm.hibernate5.SpringSessionContext
      default_batch_fetch_size: ${chunkSize:100}
      jdbc.batch_size: 20
      order_inserts: true
      order_updates: true
      format_sql: true
  jackson:
    serialization:
      fail-on-empty-beans: false
  redis:
    host: k7d103.p.ssafy.io
    port: 6379
    password: ${spring.redis.password}
  server:
    servlet:
      context-path: /api

security:
  jwt:
    token:
      secret-key: nawanolza
      expire-length: 1800000

kakao:
  client_id: 79a60227e8d4d465f7b303d472b3cd99
```

```
redirect_uri: http://localhost:8080/api/auth/kakao/callback
```

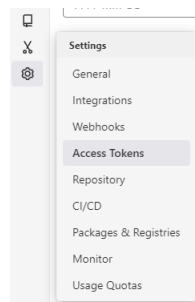
```
management:  
  endpoints:  
    web:  
      base-path: /management  
      path-mapping:  
        health: health_check  
      exposure:  
        include: health, info
```

## Jenkins Pipeline 구축

### Jenkins 환경 설정

- Gitlab 연결

- Gitlab에서 Access Token을 발행을 위해 프로젝트의 설정에서 **Access Token**으로 이동



- 토큰 이름, 만료일자, 권한, 스코프 등을 설정하고 토큰을 생성

**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.](#)

**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**

**Select a role**

Maintainer

**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\_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).

**Active project access tokens (1)**

Token name	Scopes	Created	Last Used	Expires	Role
Jenkins_Connection	api, read_api, read_repository, write_repository	Nov 3, 2022	2 weeks ago	Never	Maintainer

[Revoke](#)

- 위에서 생성한 토큰을 Jenkins에 등록하기 위해 **Jenkins 관리** → **Manage Credentials** → **Add Credentials**로 이동
- Kind에서 GitLab API token을 선택하고 API Token에 앞서 생성한 토큰값을 입력해서 Credential 생성

## New credentials

Kind

GitLab API token

Scope ?

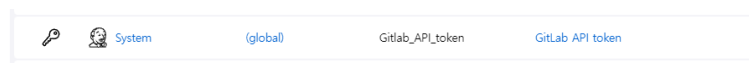
Global (Jenkins, nodes, items, all child items, etc)

API token

ID ?

Description ?

Create



- Access Token으로 Jenkins와 Gitlab을 연결하기 위해 **Jenkins 관리** -> **시스템 설정** -> **Gitlab** 으로 이동
- 식별을 위한 이름과 GitLab URL, 연결에 사용할 토큰을 입력

## Gitlab

☒ Enable authentication for '/project' end-point

### GitLab connections

Connection name

A name for the connection

SSAFY

Gitlab host URL

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

https://lab.ssfafy.com

Credentials

API Token for accessing Gitlab

GitLab API token

+ Add

고급...

Test Connection

추가

## • 환경변수 설정

- 배포에 사용할 환경변수를 정의하기 위해 **Jenkins 관리** → **시스템 설정** → **Global properties** 로 이동
- 환경변수의 이름과 값을 아래와 같이 지정
  - 이름: ENV\_IDEL\_PROFILE, 값: prod
  - 이름: SPRING\_DATASOURCE\_PASSWORD, 값: k7nawashfwkd103
  - 이름: SPRING\_DATASOURCE\_URL, 값: jdbc:mysql://k7d103.p.ssfafy.io:3306/nawanolza?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=Asia/Seoul&useLegacyDatetimeCode=false
  - 이름: SPRING\_DATASOURCE\_USERNAME, 값: root
  - 이름: SPRING\_REDIS\_HOST, 값: k7d103.p.ssfafy.io
  - 이름: SPRING\_REDIS\_PASSWORD, 값: k7nawashfwkd103

- 이름: SPRING\_REDIS\_PORT, 값: 6379

## Pipeline 구성

- 젠킨스 메인페이지에서 **새로운 Item** 을 누르고 server 빌드를 위한 아이템을 **Pipeline** 으로 생성

- **General** 의 **GitLab Connection** 에 앞서 설정한 GitLab을 선택

- **Build Triggers** 에서 **Generic Webhook Trigger** 를 선택

- Webhook의 트리거로 사용할 토큰을 설정

Token

server

Optional token. If it is specified then this job can only be triggered if that token is supplied when invoking `http://JENKINS_URL/generic-webhook-trigger/invoke`. It can be supplied as a:

- Query parameter `/invoke?token=TOKEN_HERE`
- A token header `token: TOKEN_HERE`
- A Authorization: Bearer header `Authorization: Bearer TOKEN_HERE`

- Webhook 트리거 주소를 GitLab에 등록
  - GitLab 프로젝트의 설정에서 Webhook으로 이동



- URL 에 `http://{젠킨스주소}/generic-webhook-trigger/invoke?token={토큰이름}` 형식으로 경로를 입력하고, Trigger 에서 Merge request events 를 선택

**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**

`http://k7d103.p.ssafy.io:9090/generic-webhook-trigger/invoke?token=server`

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  
Branch name or wildcard pattern to trigger on (leave blank for all)  
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.

- Post content parameters를 추가
  - Post 요청을 받았을 때 요청에 있는 값들 중에서 트리거로 사용할 변수들을 지정하는 것

Post content parameters

추가

If you want value of **param1** from post content { "param1": "value1" } to be contributed, you need to add **\$.param1** here.



Variable

Name of variable

Expression

☐ JSONPath

☐ XPath

Expression to evaluate in POST content. Use [JSONPath](#) for JSON or [XPath](#) for XML.

Value filter

Optional. Anything in the evaluated value, matching this [regular expression](#), will be removed. Having `[^0-9]` would only allow numbers. The regexp syntax is documented [here](#).

Default value

Optional. This value will be used if expression does not match anything.

- **Name of variable**: 사용자가 설정하는 변수명
- **Expression**: **JSONPath** 또는 **XPath** 형식으로 나타나는 표현식
  - **JSONPath**: JSON 객체 탐색에 사용되는 형식
  - **XPath**: XML 데이터 탐색에 사용되는 형식
- ⇒ Gitlab의 Webhook으로 받는 정보를 JSON으로 처리할 것이므로 **JSONPath**를 사용
- Gitlab의 **merge request events**로 받는 데이터의 형식은 아래 링크를 참고

Webhook events | GitLab  
 Documentation for GitLab Community Edition, GitLab Enterprise Edition, Omnibus GitLab, and GitLab Runner.  
[https://docs.gitlab.com/ee/user/project/integrations/webhook\\_events.html](https://docs.gitlab.com/ee/user/project/integrations/webhook_events.html)

- Gitlab에서 머지가 완료됐을 때, Label을 기준으로 Frontend와 Backend의 빌드 분기가 발생하도록 설정하기 위해 다음 3가지 parameter를 받아서 사용

Variable

Name of variable

Expression

☒ JSONPath

☐ XPath

Variable

Name of variable

Expression

☒ JSONPath

☐ XPath

Variable

Name of variable

Expression

☒ JSONPath

☐ XPath

머지 결과

Label

타겟 branch

- Optional filter를 설정해서 parameter의 값에 따라 트리거 발생 여부를 결정

#### Optional filter

Expression

(?!=.\*merged)(?!=.\*develop)(?!=.\* BACK)

Regular expression to test on the evaluated text specified below. The regexp syntax is documented [here](#).

Text

\$ACTION \$REF \$LABEL\_0

Text to test for the given [expression](#). You can use any combination of the variables you configured above.

- Expression: 정규식으로 정의된 트리거 발생 조건
- Text: 표현식의 각 부분에 매핑될 사용자 정의 변수들
- 젠킨스에서 도커 빌드를 하기 위해 젠킨스 컨테이너에 도커를 설치
  - EC2에서 젠킨스 컨테이너에 접속

```
# 도커 컨테이너에 접속하는 명령어
docker exec -it {컨테이너명} bash

# jenkins 컨테이너 명이 jenkins_cicd일 경우
docker exec -it jenkins_cicd bash
```

- 젠킨스 컨테이너 안에서 도커 설치

전체적인 설치 방법은 ec2에 docker를 설치하는 방법과 동일하지만, sudo를 제거하고, ubuntu라고 된 부분을 debian으로 변경해야 함

#### ■ 설치

- 사전 패키지 다운로드

```
# apt 패키지 업데이트
apt update

# 사전 패키지 설치
apt-get install -y ca-certificates \
    curl \
    software-properties-common \
    apt-transport-https \
    gnupg \
    lsb-release

# 위의 명령어로 패키지 설치가 잘 안된다면 각각 나눠서 설치해도 됨
apt-get install -y ca-certificates
apt-get install -y curl
apt-get install -y software-properties-common
apt-get install -y apt-transport-https
apt-get install -y gnupg
apt-get install -y lsb-release
```

- gpg 키 다운로드

리눅스 패키지 툴이 프로그램 패키지가 유효한지 확인하기 위해, 프로그램 설치 전에 gpg 키로 검증하는 과정을 거치기 때문에 gpg 키를 받아야 함

```
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/
    $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- 도커 설치

```
apt update
apt install docker-ce docker-ce-cli containerd.io docker-compose
```

위의 명령어로 설치하는 과정에서 문제가 발생할 경우 다음 명령어들을 순서대로 수행 후 재시도

```
# 패키지 리스트 업데이트
apt update

# apt가 https를 통해 패키지를 사용하기 위한 선행 패키지들을 설치
apt install apt-transport-https ca-certificates curl software-properties-common

# 시스템 상에 공식 도커 저장소 접속을 위한 GPG 키를 추가
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# 도커 저장소를 apt 소스에 추가
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian gpg" > /etc/apt/sources.list.d/docker.list

# 추가된 내용을 적용하기 위해 다시 패키지 리스트 업데이트
apt update

# ubuntu 기본 저장소가 아닌 도커 저장소를 사용하도록 설정
apt-cache policy docker-ce
```

- 빌드 및 배포

Pipeline에서 Pipeline script를 선택하고 스크립트를 작성

- ▼ 코드 보기

```
node {
  stage('clone') {
    git branch: 'develop', credentialsId: '1217jdk', url: 'https://lab.ssafy.com/s07-final/S07P31D103.git'

    sh "echo 'clone'"
  }

  stage('build') {
    dir("nawanolza/server") {
      try {
        mattermostSend (
          color: "#2A42EE",
          message: "Build STARTED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
        )
        app = docker.build("server")
      } catch(e) {
        currentBuild.result = "FAILURE"
      } finally {
        if(currentBuild.result == "FAILURE") {
          mattermostSend (
            color: "danger",
            message: "Build FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
          )
        } else {
          mattermostSend (
            color: "good",
            message: "Build SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
          )
        }
      }
    }
    sh "echo 'build'"
  }
}

stage('deploy') {
  def deployResult = "SUCCESS"

  try {
    mattermostSend (
      color: "#2A42EE",
      message: "Deploy STARTED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
    )
    sshagent(credentials: ['jenkins-ssh']) {
      sh "ssh -o StrictHostKeyChecking=no ubuntu@172.26.2.49 './deploy.sh'"
    }
  } catch(e) {
    deployResult = "FAILURE"
  } finally {
    if(deployResult == "FAILURE") {
      mattermostSend (
        color: "danger",
        message: "Deploy FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
      )
    } else {
      mattermostSend (
        color: "good",
        message: "Deploy SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
      )
    }
  }
}
```

```

        color: "good",
        message: "Deploy SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER} (<${env.BUILD_URL}|Link to build>)"
    )
}
}
}
}
}

```

Pipeline script에 의해 실행 될 deploy.sh 파일을 작성해서 /home/ubuntu에 저장

#### ▼ 코드 보기

```

#!/bin/sh

response=$(curl -s http://k7d103.p.ssafy.io:9001/api/management/health_check)
echo $response

up=$(echo $response | grep 'UP' | wc -l)
echo $up

if [ $up -eq 1 ]
then
    green_port=9002
    blue_port=9001
else
    green_port=9001
    blue_port=9002
fi

echo `sudo docker run \
    -d --rm -p $green_port:8080 \
    --name spring_$green_port \
    -e SPRING_DATASOURCE_URL='jdbc:mysql://k7d103.p.ssafy.io:3306/nawanolza?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC' \
    -e SPRING_DATASOURCE_USERNAME=root \
    -e SPRING_DATASOURCE_PASSWORD=k7nawashfwkd103 \
    -e SPRING_REDIS_HOST='k7d103.p.ssafy.io' \
    -e SPRING_REDIS_PORT=6379 \
    -e SPRING_REDIS_PASSWORD=k7nawashfwkd103 \
    -e ENV_IDLE_PROFILE=prod \
    server`

response=$(curl -s http://k7d103.p.ssafy.io:$blue_port/api/management/health_check)

up=$(echo $response | grep 'UP' | wc -l)

if [ $up -eq 1 ]
then
    sudo docker stop spring_$blue_port
fi

sudo sed -i "s/${blue_port}/${green_port}/g" /etc/nginx/sites-available/k7d103.p.ssafy.io.conf

sudo nginx -s reload

```

- 위의 설정들을 모두 마치면 빌드 및 배포 관련 모든 설정이 끝나기 때문에 Gitlab에서 라벨을 붙여서 머지를 성공하면 정상적으로 빌드 후 배포됨