

# SGBG 포팅 메뉴얼

## 1. 개발환경

### 1.1. Frontend

### 1.2. Backend

#### Language

#### Frameworks

#### Database

#### APIs

#### Testing

#### Build Tools

#### Libraries

#### Security

#### Cloud

#### ElasticSearch

#### Image Processing

#### Scheduler

#### Other

### 1.3. Server

### 1.4. Database

### 1.5. UI/UX

### 1.6. IDE

### 1.7. 형상/이슈관리

### 1.8. 기타 툴

## 2. 환경변수

### 2.1. Backend(API)

### 2.2. 민감 환경변수 관리

## 3. EC2 세팅

### 3.1. Docker 설치

### 3.2. Docker-compose 설정

### 3.3. Nginx 설정

### 3.4. EC2 Port

### 3.5. 방화벽(UFW) 설정

## 4. CI/CD 구축

### 4.1. Jenkins 도커 이미지 + 컨테이너

### 4.2. Jenkins 설정

#### 4.2.1 GitLab Credentials 설정

#### 4.2.2 Jenkins Item 생성

[4.2.3. GitLab Webhook 설정](#)

[4.2.4. 빌드 및 배포](#)

[5. Redis 설정](#)

[6. 시연 시나리오](#)

[6.1 랜딩 페이지](#)

[6.2 웹 상 확장 프로그램](#)

[6.3 데스크톱 앱 - 홈 화면](#)

[6.4 데스크톱 앱 - 내 디렉토리](#)

[6.5 데스크톱 앱 - 설정 화면](#)

# 1. 개발환경

## 1.1. Frontend

- Node JS 20.15.1
- React 18.2.0
- electron 29.1.1
- Axios 1.7.7
- styled-components 6.1.13

## 1.2. Backend

### Language

- **Java:** OpenJDK 17

### Frameworks

- **Spring Boot:** 3.3.2
  - Web, Validation, WebFlux, Actuator (for health checks)
- **Spring Data JPA:** 3.3.2
- **Spring Security:** 3.3.2
- **Spring Retry:** 1.3.1 (for retry functionality in schedulers)
- **Spring Cloud AWS:** 2.2.6.RELEASE (for AWS S3 integration)
- **Springdoc OpenAPI:** 2.0.4 (for Swagger documentation)

- **Spring Data Elasticsearch:** Latest compatible with Elasticsearch High-Level Client 7.17.10

## Database

- **MySQL:** Driver version 8.0.33
- **Redis:** Spring Boot Starter Data Redis (includes Redis caching)
- **MongoDB:** Not explicitly included in the dependencies but extendable through Spring Data MongoDB if needed

## APIs

- **Jakarta Persistence API:** 3.1.0
- **Jakarta Servlet API:** 6.0.0

## Testing

- **JUnit:** 5.8.2
- **Spring Security Test:** Integrated for security-related testing
- **Thymeleaf Security Extras:** Thymeleaf templates with Spring Security support

## Build Tools

- **Gradle:** Version 7.6 (as the build automation tool)

## Libraries

- **Lombok:** 1.18.26 (for boilerplate code reduction)
- **QueryDSL:** 5.1.0 (for type-safe queries with JPA and Jakarta support)
- **Guava:** 29.0-jre (for utility functions)
- **JSON (org.json):** 20210307
- **Jsoup:** 1.7.2 (for parsing and extracting HTML)
- **Thymeleaf:** For server-side rendering

## Security

- **OAuth 2.0:** Client and Resource Server
- **JWT:** Java JWT (jjwt-api 0.11.5) with Jackson support for JSON parsing

## Cloud

- **AWS S3:** Integration via AWS Java SDK (Version 1.11.1000) and software.amazon SDK (Version 2.20.80)

## ElasticSearch

- **Spring Data Elasticsearch:** Integrated for ease of use with Elasticsearch
- **Elasticsearch High-Level Client:** Version 7.17.10
- **Elastic Java Client:** Version 8.9.0 (for advanced use cases)

## Image Processing

- **TwelveMonkeys ImageIO:** Core (3.9.4) and WebP support

## Scheduler

- **Spring Retry:** 1.3.1 (with Spring Aspects)

## Other

- **Base64 Processing:** JAXB API 2.3.1

## 1.3. Server

- Ubuntu 20.04 LTS
- Docker-compose 2.6.1
- Nginx 1.18.0
- Docker 27.3.1
- Jenkins 2.452.3

## 1.4. Database

- MySQL 9.0.1
- Redis 7.4.0

## 1.5. UI/UX

- Figma

## 1.6. IDE

- Visual Studio Code 1.91.1
- IntelliJ IDEA 2024.01

## 1.7. 형상/이슈관리

- GitLab
- Jira

## 1.8. 기타 툴

- Postman 11.6.2
- Termius 9.2.0

# 2. 환경변수

## 2.1. Backend(API)

application.yml

```
spring:
  application:
    name: backend

  security:
    oauth2:
      client:
        registration:
          google:
            client-id: 466186867370-iosqitk6mjeaa8u6tpm7lqnlq
            client-secret: GOCSPX-yoZ_mqQQEcDLufgbQ4BhHTmNrzd
            scope:
              - profile
              - email
            redirect-uri: "{baseUrl}/oauth2/code/google" # 2
            authorization-grant-type: authorization_code
        provider:
```

```

        google:
            authorization-uri: https://accounts.google.com/o/
            token-uri: https://oauth2.googleapis.com/token #
            user-info-uri: https://www.googleapis.com/oauth2/
            user-name-attribute: sub # 사용자 정보를 식별하는 속성

datasource:
    url: jdbc:mysql://k11b205.p.ssafy.io:3306/sgbg?createData
    username: sgbg
    password: sgbg
    driver-class-name: com.mysql.cj.jdbc.Driver

jpa:
    hibernate:
        ddl-auto: update
    properties:
        hibernate:
            dialect: org.hibernate.dialect.MySQLDialect
            naming:
                physical-strategy: org.hibernate.boot.model.naming.

elasticsearch:
    uris: k11b205.p.ssafy.io:9200
    username: sgbg
    password: sgbg
# main:
#     allow-bean-definition-overriding: true

cache:
    type: redis

redis:
    host: k11b205.p.ssafy.io
    port: 6379
    password: sgbg

# redis-keyword 인스턴스 설정
redis-keyword:

```

```

    host: k11b205.p.ssafy.io
    port: 6380
    password: sgbg

# redis-user 인스턴스 설정
redis-user:
    host: k11b205.p.ssafy.io
    port: 6381
    password: sgbg

jwt:
    secretkey: IWJqam6tBDpBs7bm1/yreAjl3xVlGYf1C1glzfKGn4=

openai:
    model: gpt-4o
    api:
        key: sk-proj-TfyB5rabijdf9mfudrIyr_BjwAGHdrAvQPckyTI34tUu
        url: https://api.openai.com

google:
    vision:
        api-key: AIzaSyDqwJw6qknWfRBqmdzVyPuvpxi2zDh2o1s
        credential-json: |
            {
                "type": "service_account",
                "project_id": "genial-moon-439615-a5",
                "private_key_id": "fb42dec2d74e5fd60422e4cfd41b0a589b",
                "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIB,
                "client_email": "singblebungle@genial-moon-439615-a5..",
                "client_id": "105542841769930751826",
                "auth_uri": "https://accounts.google.com/o/oauth2/aut",
                "token_uri": "https://oauth2.googleapis.com/token",
                "auth_provider_x509_cert_url": "https://www.googleapi",
                "client_x509_cert_url": "https://www.googleapis.com/r",
                "universe_domain": "googleapis.com"
            }

server:

```

```
port: ${SERVER_PORT:8081} # 기본값은 8081, 환경 변수 SERVER_PORT
address: 0.0.0.0
servlet:
  context-path: /api
  encoding:
    charset: UTF-8
    enabled: true
    force: true

management:
  endpoints:
    web:
      exposure:
        include: health

# S3 bucket connection
cloud:
  aws:
    credentials:
      accessKey: AKIA6GBMGD5ZBLFCK5WX
      secretKey: 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
    s3:
      bucketName: sgbgbucket
      region:
        static: ap-northeast-2
      stack:
        auto: false

springdoc:
  swagger-ui:
    path: /swagger-ui.html

logging:
  level:
    org.elasticsearch.client.RestClient: ERROR
    org.springframework.web: WARN
    org.springframework.security: WARN
    org.apache.tomcat: WARN
```



## 2.2. 민감 환경변수 관리

Jenkins credentials 로 민감 환경변수 설정 파일 자동 저장 및 관리  
(.gitignore에 추가하여 GitLab에 푸시되는 일이 없도록 함)

### Security



#### Security

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



#### Users

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



#### Credentials

Configure credentials



#### Credential Providers

Configure the credential providers and types

Jenkins 설정의 Credentials 로 간다.

### Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

아이콘: S M L

(global) 도메인으로 만든다.

### Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
jenkins-ssh-key-id	ubuntu (jenkins-ssh-key-id)	SSH Username with private key	jenkins-ssh-key-id
gitlab-token	gitlab-token/***** (gitlab-token)	Username with password	gitlab-token

Add Credentials로 새로운 Credential을 만든다.

### New credentials

Kind

Secret file

Scope ?

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

File

파일 선택 선택된 파일 없음

ID ?

application-backend-properties

❗ This ID is already in use

Description ?

application-backend-properties

Create

Secret 파일에 민감한 환경변수 파일을 넣고, 파이프라인 구성시 필요한 환경변수 파일을 복사해서 이미지를 빌드하는 형식으로 진행

## 3. EC2 세팅

### 3.1. Docker 설치

```
# 1. 리눅스 업데이트
sudo apt update -y && sudo apt upgrade -y

# 1.1 필수 패키지 설치
sudo apt-get install -y ca-certificates curl gnupg lsb-release

# 2. Docker의 공식 GPG 키를 추가할 디렉토리 생성
sudo mkdir -p /etc/apt/keyrings

# 3. Docker의 GPG 키 다운로드 및 바이너리 형식으로 변환하여 저장
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo

# 4. Docker 저장소를 추가
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list
```

```
# 5. 패키지 목록 업데이트
sudo apt-get update

# 6. Docker 패키지 설치
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# 7. Docker 데몬을 시작하고 부팅 시 자동으로 시작하도록 설정
sudo systemctl start docker
sudo systemctl enable docker
```

## 3.2. Docker-compose 설정

```
version: '3.8'

services:
  nginx:
    image: nginx:latest
    restart: always
    environment:
      - LISTEN_PORT=443 # 환경 변수 LISTEN_PORT 추가
    volumes:
      - ./data/nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    ports:
      - "80:80"
      - "443:443"
    command: "/bin/sh -c 'while :; do sleep 320h & wait ${!}'"
    depends_on:
      - landing_frontend # 랜딩 페이지 프론트엔드 의존성 추가
      - other_frontend # 다른 페이지 프론트엔드 의존성 추가
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "1"
```

```

certbot:
  image: certbot/certbot
  restart: unless-stopped
  environment:
    TZ: Asia/Seoul
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  entrypoint: "/bin/sh -c 'trap exit TERM; while :; do cert
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "1"

```

```

certbot:
  image: certbot/certbot
  restart: unless-stopped
  environment:
    TZ: Asia/Seoul
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  entrypoint: "/bin/sh -c 'trap exit TERM; while :; do cert
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "1"

```

```

jenkins:
  image: jenkins/jenkins:lts
  restart: unless-stopped
  volumes:
    - jenkins_home:/var/jenkins_home
    - ./data/jenkins:/var/jenkins_shared
    - /var/run/docker.sock:/var/run/docker.sock # Docker 소
    - /usr/local/bin/docker-compose:/usr/local/bin/docker-c

```

```

    - /usr/bin/docker:/usr/bin/docker # Docker CLI 바이너리
    - /home/ubuntu/plog:/home/ubuntu/plog # 호스트의 프로젝트
ports:
  - "8080:8080"
  - "50000:50000"
environment:
  JENKINS_OPTS: --prefix=/jenkins
  JAVA_OPTS: -Djava.util.logging.config.file=/var/jenkins
  TZ: Asia/Seoul
logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "1"

backend:
  image: backend:latest
  restart: unless-stopped
  expose:
    - "8081" # 내부포트 8081
  environment:
    SPRING_PROFILES_ACTIVE: prod
    SERVER_PORT: 8081 # 환경 변수로 포트 설정
    TZ: Asia/Seoul
    LOG_FILE_PATH: backend

other_frontend: # 다른 페이지 프론트엔드
  image: other_frontend:latest
  restart: unless-stopped
  environment:
    NODE_ENV: production
    TZ: Asia/Seoul
    LISTEN_PORT: 3001 # Nginx에서 사용할 포트를 환경 변수로 설정
  expose:
    - "3001" # 내부포트 3001

landing_frontend: # 랜딩 페이지 전용 프론트엔드
  image: landing_frontend:latest

```

```
restart: unless-stopped
environment:
  NODE_ENV: production
  TZ: Asia/Seoul
  LISTEN_PORT: 3000 # Nginx에서 사용할 포트를 환경 변수로 설정
expose:
  - "3000" # 내부포트 3000
```

```
mysql:
  image: mysql:latest
  container_name: mysql
  restart: unless-stopped
  environment:
    MYSQL_ROOT_PASSWORD: sgbg
    MYSQL_DATABASE: sgbg
    MYSQL_USER: sgbg
    MYSQL_PASSWORD: sgbg
    TZ: Asia/Seoul
  volumes:
    - ./data/mysql_data:/var/lib/mysql
  expose:
    - "3306"
  ports:
    - "3306:3306"
```

```
redis:
  image: redis:latest
  restart: unless-stopped
  command: redis-server --requirepass sgbg
  expose:
    - "6379"
  ports:
    - "6379:6379"
```

```
redis-keyword:
  image: redis:latest
  restart: unless-stopped
  command: redis-server --requirepass sgbg
```

```

    expose:
      - "6379"
    ports:
      - "6380:6379"

redis-user:
  image: redis:latest
  restart: unless-stopped
  command: redis-server --requirepass sgbg
  expose:
    - "6379"
  ports:
    - "6381:6379"

redisinsight:
  image: redis/redisinsight:latest
  restart: unless-stopped
  ports:
    - "5540:5540"
  volumes:
    - ./data/redis/insight:/db
    - ./data/redis/insight/log:/db/logs
  depends_on:
    - redis
  environment:
    TZ: Asia/Seoul

elasticsearch:
  image: elasticsearch:7.17.0
  restart: unless-stopped
  ports:
    - "9200:9200"
  environment:
    - discovery.type=single-node
    - network.host=0.0.0.0 # 모든 인터페이스
    - ES_JAVA_OPTS=-Xms2g -Xmx2g # 메모리
    - TZ=Asia/Seoul
  volumes:

```

```

        - ./data/elasticsearch:/usr/share/ela
    expose:
        - "9200"

    kibana:
        image: kibana:7.17.0
        restart: unless-stopped
        ports:
            - "5601:5601"
        depends_on:
            - elasticsearch
        environment:
            - ELASTICSEARCH_HOSTS=http://elastics
            - TZ=Asia/Seoul
            - SERVER_BASEPATH=/kibana
            - SERVER_REWRITEBASEPATH=true
            - SERVER_PUBLICBASEURL=https://k11b20
            - XPACK_SECURITY_ENABLED=false
        expose:
            - "5601"

    volumes:
        jenkins_home:
        mysql_data:
        redis:
        redis-keyword:
        redis-user:
        redisinsight:

```

### 3.3. Nginx 설정

```

user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

```



```

events {
    worker_connections 1024;
}

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

    log_format  main  '$remote_addr - $remote_user [$time_local]
                      "$status $body_bytes_sent" "$http_referer
                      "$http_user_agent" "$http_x_forwarded_for"
                      "$host" "$server_name" "$request_uri"
                      "$request_body" "$args" "$upstream_addr';

    access_log  /var/log/nginx/access.log  main;

    sendfile    on;
    keepalive_timeout 65;

    # 요청 제한 설정
    limit_req_zone $binary_remote_addr zone=mylimit:10m rate=10r/s;

    server {
        listen 80;
        server_name k11b205.p.ssafy.io;
        server_tokens off;

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

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

    }
}

```

```

server {
    listen 443 ssl;
    server_name k11b205.p.ssafy.io;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/k11b205.p.ssafy
    ssl_certificate_key /etc/letsencrypt/live/k11b205.p.s
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # 요청 제한 설정 적용
    location / {
        limit_req zone=mylimit burst=10 nodelay;
        limit_req_status 429;
        # 요청이 너무 많을 경우 429 에러 반환

        # 프록시 설정 주석 처리
        # proxy_pass https://your_backend_server;
        # proxy_set_header Host $host;
        # proxy_set_header X-Real-IP $remote_addr;
        # proxy_set_header X-Forwarded-For $proxy_add_x_f
        # proxy_set_header X-Forwarded-Proto $scheme;
        proxy_pass http://other_frontend:3001; # 다른 페이
        #proxy_pass http://landing_frontend:3000; # 랜딩
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_for
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Port 443;
    }

    # 다른 경로로 접속 시 다른 프론트엔드 컨테이너로 연결
    location /landing/ {
        limit_req zone=mylimit burst=10 nodelay;
        limit_req_status 429;

        # /landing 경로를 다른 프론트엔드 컨테이너로 프록시
        proxy_pass http://landing_frontend:3000/landing/;
    }
}

```

```

        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;
        proxy_set_header X-Forwarded-Port 443;
    }

# Jenkins 리버스 프록시 설정 (8080 포트)
location /jenkins {
    proxy_pass http://jenkins:8080/jenkins;
    proxy_set_header Host $host:443;
    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-Forwarded-Port 443;

    proxy_http_version 1.1;

}

# Jenkins의 50000 포트에 대한 리버스 프록시 설정
location /jenkins-jnlp {
    proxy_pass http://jenkins:50000;
    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;
}

# backend 리버스 프록시 설정
location /api/ {
    proxy_pass http://backend:8081;
    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;
}

```

```

# Kibana 리버스 프록시 설정
location /kibana/ {
    proxy_pass http://kibana:5601;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    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 /elasticsearch/ {
    proxy_pass http://elasticsearch:9200;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    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 /redisinsight/ {
    proxy_pass http://redisinsight:5540/redisinsight/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    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;
}
}
}

```

## 3.4. EC2 Port

Port 번호	내용
22	SSH
80	HTTP (HTTPS로 redirect)
443	HTTPS
3000, 3001	Nginx, React(Docker)
5601	kibana
5540	redisInsight(GUI)
6379	redis, redis-keyword, redis-user
8080	Jenkins
8081	spring boot(api)
9200	elasticsearch

### 3.5. 방화벽(UFW) 설정

ufw 상태 확인

```
sudo ufw status
```

Status: active

To	Action	From
--	-----	----
22	ALLOW	Anywhere
8989	ALLOW	Anywhere
443	ALLOW	Anywhere
80	ALLOW	Anywhere
8080	ALLOW	Anywhere
9200	ALLOW	Anywhere
5601	ALLOW	Anywhere
3306	ALLOW	Anywhere
5540	ALLOW	Anywhere
6379	ALLOW	Anywhere
6380	ALLOW	Anywhere
6381	ALLOW	Anywhere
22 (v6)	ALLOW	Anywhere (v6)
8989 (v6)	ALLOW	Anywhere (v6)
443 (v6)	ALLOW	Anywhere (v6)
80 (v6)	ALLOW	Anywhere (v6)

8080 (v6)	ALLOW	Anywhere (v6)
9200 (v6)	ALLOW	Anywhere (v6)
5601 (v6)	ALLOW	Anywhere (v6)
3306 (v6)	ALLOW	Anywhere (v6)
5540 (v6)	ALLOW	Anywhere (v6)
6379 (v6)	ALLOW	Anywhere (v6)
6380 (v6)	ALLOW	Anywhere (v6)
6381 (v6)	ALLOW	Anywhere (v6)

사용할 포트 허용하기

```
sudo ufw allow 포트번호
```

ufw 활성화하기

```
sudo ufw enable
```

등록한 포트 삭제하기

```
sudo ufw status numbered
```

```
sudo ufw delete 4
```

## 4. CI/CD 구축

### 4.1. Jenkins 도커 이미지 + 컨테이너

docker-compose.yml 내부

```
jenkins:
  image: jenkins/jenkins:lts
  restart: unless-stopped
  volumes:
    - jenkins_home:/var/jenkins_home
    - ./data/jenkins:/var/jenkins_shared
    - /var/run/docker.sock:/var/run/docker.sock # Docker 소
    - /usr/local/bin/docker-compose:/usr/local/bin/docker-c
    - /usr/bin/docker:/usr/bin/docker # Docker CLI 바이너리
    - /home/ubuntu/plog:/home/ubuntu/plog # 호스트의 프로젝트
  ports:
    - "8080:8080"
```

```

- "50000:50000"
environment:
  JENKINS_OPTS: --prefix=/jenkins
  JAVA_OPTS: -Djava.util.logging.config.file=/var/jenkins
  TZ: Asia/Seoul

```

## 4.2. Jenkins 설정

### 4.2.1 GitLab Credentials 설정

1. 아이디 → "Credentials" 클릭
2. "Store : System" → "(global)" → "+ Add Credentials" 클릭

**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.

**Stores scoped to Jenkins**

P	Store ↓	Domains
	System	(global)

아이콘: S M L

**Global credentials (unrestricted)** + Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
jenkins-ssh-key-id	ubuntu (jenkins-ssh-key-id)	SSH Username with private key	jenkins-ssh-key-id
gitlab-token	gitlab-token/***** (gitlab-token)	Username with password	gitlab-token

3. "Kind"에 "Username with password" 입력 → "Username"에 GitLab ID 혹은 원하는 ID 입력(gitlab-token) → "Password"에 Gitlab Personal Access Tokens 입력 → "ID"에 임의 아이디 입력(gitlab-token) → 생성  
 \*\*\* Personal Access Token은 Gitlab > User Settings > Access Tokens에서 생성

## New credentials

Kind  
Username with password

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

Username ?  
[Redacted]

☐ Treat username as secret ?

Password ?  
[Redacted]

ID ?  
[Redacted]

## 4.2.2 Jenkins Item 생성

1. "새로운 Item" 클릭
2. "Enter an item name"에 임의 Item 이름 입력 → "Pipeline" 클릭

Enter an item name

test-item

» 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.

3. "General" → "Do not allow concurrent builds" 클릭  
(한 빌드를 진행중이면 동시에 빌드를 진행하지 않게 한다)

☒ Do not allow concurrent builds

☐ Abort previous builds ?

4. "Build Triggers" → "Build when a change is pushed to GitLab" 클릭  
(WebHook 설정 : GitLab 특정 브랜치 merge 시 자동 빌드 + 배포 설정)  
(해당 URL 복사 → WebHook 설정 시 사용 예정)



☒ Build when a change is pushed to GitLab. GitLab webhook URL: <https://111b308.p.ssafy.io/jenkins/project/backend-pipeline> ?

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

## 5. "Build when a change is pushed to GitLab" 하위의 "고급..." 클릭

고급 ^ Edited

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update ?

Allowed branches

☐ Allow all branches to trigger this job ?

☐ Filter branches by name ?

## 6. 특정 브랜치에서 타겟 브랜치로 머지를 할 경우 빌드 + 배포가 진행되도록 설정 Secret token의 "Generate" 클릭 후 생성된 토큰값 복사

● Filter branches by regex ?

Source Branch Regex

Target Branch Regex

☐ Filter merge request by label

Secret token ?

Generate

Clear

7. "Pipeline" → "Definition"에 Pipeline script from SCM 설정 → "SCM"에 "Git" 설정 → "Repository URL"에 프로젝트 GitLab URL 입력 → "Credentials"에 사전에 추가한 Credentials 입력

#### Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

Credentials ?

+ Add

고급

8. "Branch Specifier"에 빌드 할 브랜치명 입력 (master일 시 "\*/master)

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/develop-be

## 9. "Script Path"에 Jenkinsfile 경로 입력

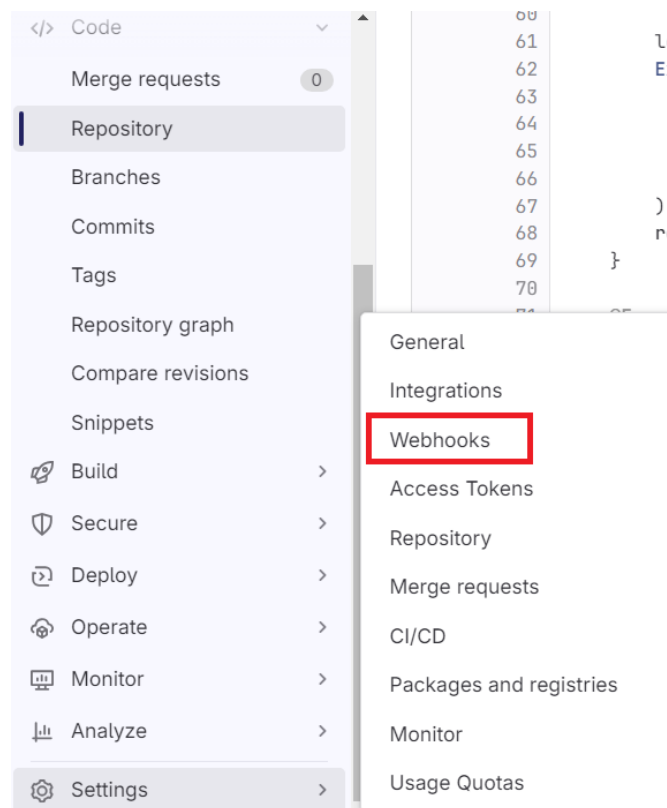
Script Path ?

backend/Jenkinsfile

☐ Lightweight checkout ?

## 4.2.3. GitLab Webhook 설정

### 1. 프로젝트 GitLab → "Settings" → "Webhooks" 클릭



### 2. "URL"에 사전에 복사해놓은 Jenkins URL 입력 → "Secret token"에 사전에 복사해놓은 Secret token 입력 → "Merge request events" 클릭 후 WebHook 적용 브랜치 입력 (Jenkins Branch Specifier과 일치하여야 함)

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 UI.

Custom headers </> 0

No custom headers configured.

Name (optional)

Backend Realtime Webhook

Description (optional)

Backend Realtime Webhook

Secret token

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

#### 4.2.4. 빌드 및 배포

Option 1. 상기 WebHook 설정한 브랜치로 merge

Option 2. Jenkins 홈 화면 → Jenkins Item 클릭 → “지금 빌드” 클릭

## 5. Redis 설정

`docker-compose.yml` 내부

```
redis:
  image: redis:latest
  restart: unless-stopped
  command: redis-server --requirepass sgbg
  expose:
    - "6379"
  ports:
    - "6379:6379"
```

```

redis-keyword:
  image: redis:latest
  restart: unless-stopped
  command: redis-server --requirepass sgbg
  expose:
    - "6379"
  ports:
    - "6380:6379"

redis-user:
  image: redis:latest
  restart: unless-stopped
  command: redis-server --requirepass sgbg
  expose:
    - "6379"
  ports:
    - "6381:6379"

redisinsight:
  image: redis/redisinsight:latest
  restart: unless-stopped
  ports:
    - "5540:5540"
  volumes:
    - ./data/redis/insight:/db
    - ./data/redis/insight/log:/db/logs # 로그 디렉토리
  depends_on:
    - redis
  environment:
    TZ: Asia/Seoul

```

## 6. 시연 시나리오

### 6.1 랜딩 페이지

- 확장 프로그램 설치 버튼 클릭

- 구글 확장 프로그램 설치
- 확장 프로그램에서 구글 소셜 로그인 진행

## 6.2 웹 상 확장 프로그램

- 웹에서 드래그 앤 드롭으로 이미지 저장
- 서비스 내 생성되어있는 자신의 디렉토리에서 원하는 폴더에 저장

## 6.3 데스크톱 앱 - 홈 화면

- 확장 프로그램 내 경로를 통해 데스크톱 앱 열기
- 전체 이미지로 이동하여 저장된 자신의 이미지와 다른 유저들이 저장한 이미지 조회
- 이미지를 클릭하여 이미지 상세화면 조회
  - 생성된 키워드, 원본 이미지 URL
  - 이미지 URL, 파일 복사 가능
- 생성된 키워드를 이미지 상세화면에서 클릭하거나 검색창에서 검색하면 해당 키워드를 가진 이미지들을 조회 가능
- 조회, 검색, 저장 횟수를 기준으로 키워드 랭킹 조회 가능

## 6.4 데스크톱 앱 - 내 디렉토리

- 이미지를 저장했던 자신의 디렉토리에서 저장된 이미지 확인
- 저장한 이미지 삭제 시 휴지통으로 이동하고 휴지통에서 영구삭제 가능
- 디렉토리 생성, 이름 수정, 순서 변경, 삭제 가능

## 6.5 데스크톱 앱 - 설정 화면

- 시작 앱 설정 가능
- 트레이 최소화 설정 가능