하팅! 포팅매뉴얼

1. 개발환경

1.1. Frontend

- Node JS 18.13.0 (LTS)
- React 18.2.0
  - Recoil 0.7.7
- Typescript 4.9.5
- Axios 1.3.6
- Tailwind CSS 3.3.1

1.2. Backend

- Java
  - Java OpenJDK 11
  - Spring Boot 2.7.10
    - Spring Data JPA 2.7.10
    - Spring Security 2.7.10
    - JUnit 5.8.2
    - Lombok 1.18.26
  - Gradle 7.6
- Node JS 18.13.0 (LTS)
- Socket IO 4.6.1

1.3. Server

- Ubuntu 20.04 LTS
- Nginx 1.18.0
- Docker 23.0.4
- Docker Compose 2.17.2
- MicroK8s (Kubernetes) 1.26.4
- Sonarqube 3.4.0
- Jenkins 2.387.3

1.4. Database

- MySQL 8.0.30
- Redis 7.0.11

1.5. UI/UX

- Figma 93.4.0

1.6. IDE

- Visual Studio Code 1.78.2
- IntelliJ IDEA 2023.1

1.7. 형상 / 이슈관리

- Gitlab
- Jira

1.8. 웹사이트 분석/관리

- Google Analytics 4
- Google Tag Manager
- Hotjar

1.9. 기타 툴

- Postman 10.14.2
- Termius 7.58.7

2. 환경변수

2.1. Frontend

> REACT_APP_API
> REACT_APP_KAKAO_API
> REACT_APP_KAKAO_CLIENT_ID
> REACT_APP_KAKAO_JAVASCRIPT_ID
> REACT_APP_KAKAO_CLIENT_SECRET
> REACT_APP_KAKAO_REDIRECT_URI
> REACT_APP_GOOGLE_API
> REACT_APP_GOOGLE_CLIENT_ID
> REACT_APP_GOOGLE_REDIRECT_URI

2.2. Backend

> SPRING_DATASOURCE_URL
> SPRING_DATASOURCE_USERNAME
> SPRING_DATASOURCE_PASSWORD
> SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_KAKAO_CLIENTID
> SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_KAKAO_CLIENTSECRET
> SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_GOOGLE_CLIENTID
> SPRING_SECURITY_OAUTH2_CLIENT_REGISTRATION_GOOGLE_CLIENTSECRET
> SPRING_REDIS_HOST
> SPRING_REDIS_PORT
> SPRING_REDIS_PASSWORD
> JWT_SECRET
> APP_AUTH_TOKENSECRET
> CLOUD_AWS_CREDENTIALS_ACCESSKEY

> CLOUD_AWS_CREDENTIALS_SECRETKEY
> TWITTER_CONSUMERKEY
> TWITTER_CONSUMERSECRET

2.3. 민감 환경변수 관리

2.3.1. Frontend

> Jenkins Pipeline의 workspace에 위치한 프로젝트 Git Repository에서 .env 수동 저장 및 관리 (.gitignore에 추가하여 GitLab에 푸시되는 일이 없도록 함)

```
# 경로
/home/jenkins/workspace/hearting-pipeline-docker

### hearting-pipeline-docker는 Jenkins Pipeline 이름
```

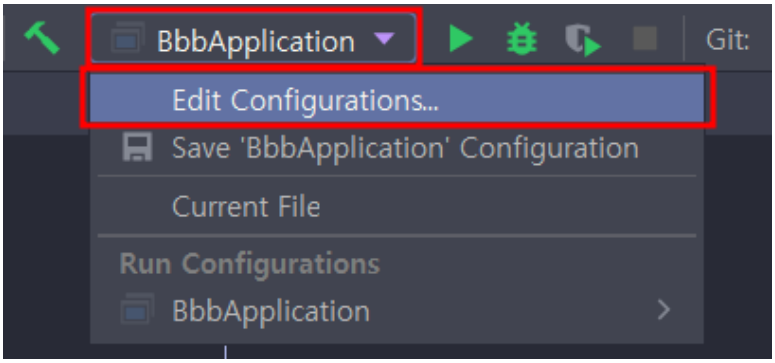
2.3.2. Backend

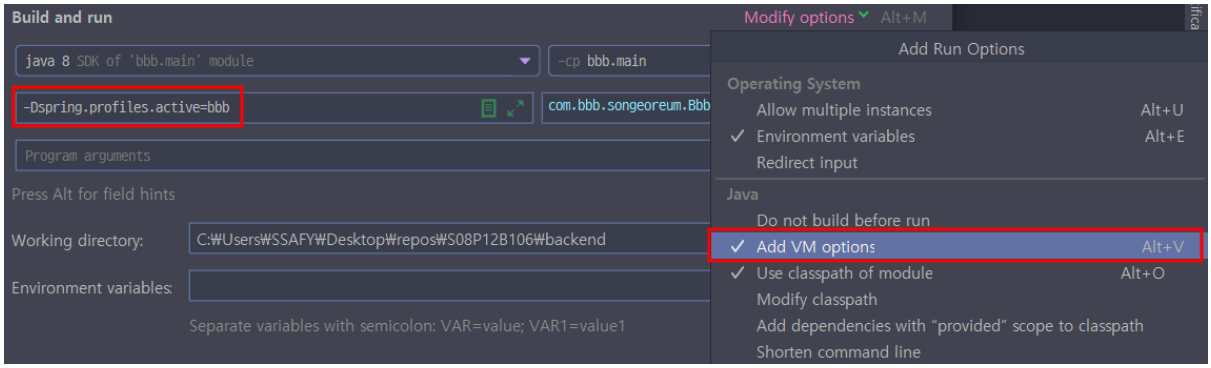
> Kubernetes ConfigMap을 활용하여 민감정보 저장하여 위와 동일하게 해당 Jenkins workspace에 수동 저장 및 관리 (.gitignore에 추가하여 GitLab에 푸시되는 일이 없도록 함)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hearting-back-configmap
data:
  # 여기에 정보 입력
```

2.3.3. Intellij Setting

✅ 로컬환경에서의 사용을 위해 intelliJ `[프로젝트명]Application` → `Edit Configurations` → `Add VM Options` 선택→ `Dspring.profiles.active=[위에 설정한 임시명]` 설정 → .gitignore에 해당 yml 파일 추가





3. EC2 세팅

3.1. Docker 설치

```
### install-docker.sh

# remove docker if exists
sudo apt-get remove docker docker-engine docker.io containerd runc

# update apt packages
sudo apt-get update
sudo apt-get -y install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# add docker's GPG key
sudo rm -r /etc/apt/keyrings
sudo mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

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

# install docker engine
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# add docker group
sudo usermod -aG docker $USER

# resetting the permissions
sudo chown -R $USER:$USER /var/run/docker.sock
sudo chmod -R 660 /var/run/docker.sock
```

3.2. MySQL(Docker) 설치

```
### install-mysql.sh

#!/bin/bash

if [ $# -ne 1 ]; then
  echo "Usage: $0 <MYSQL_ROOT_PASSWORD>"
  exit 1
fi

MYSQL_ROOT_PASSWORD=$1

docker run --name mysql \
        -v /var/lib/mysql-data:/var/lib/mysql \
        -e MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD} \
        -d -p 3306:3306 mysql:8.0.30

for i in {1..6}; do
  # Check if the MySQL server is ready
  if docker exec mysql mysqladmin -u root -p${MYSQL_ROOT_PASSWORD} status > /dev/null 2>&1; then
```

```
    echo "MySQL server is ready."
    break
  else
    echo "MySQL server is not ready. Retrying in 5 seconds..."
    sleep 5
  fi
done

############
# 해당 .sh 파일 경로에서
./install-mysql [DB 비번]
```

3.3. Nginx 설치

```
# 1. Nginx 설치
sudo apt-get install nginx
nginx -v

# 2. Let's Encrypt 설치 및 SSL 발급
sudo apt-get install letsencrypt
sudo systemctl stop nginx
sudo letsencrypt certonly --standalone -d 도메인명

# 3. Nginx 설정파일 생성
cd /etc/nginx/sites-available
vi configure
############

### DDoS 방어
# 시간 당 request 비율 제한 (클라이언트 IP에 대한 요청 1초에 최대 5개)
# 시간 당 request 비율 제한 (클라이언트 IP에 대한 요청 1초에 최대 5개)
#limit_req_zone $binary_remote_addr zone=ddos_req:50m rate=20r/s;

server {

        location /sonarqube {
                proxy_pass http://localhost:9000;
                #limit_req zone=ddos_req burst=10;
        }

        location /k8sdashboard/ {
            proxy_pass https://localhost:10443/;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_ssl_verify off;
        }

        location /jenkins/ {
                proxy_pass http://localhost:8081;
#               proxy_redirect $scheme://$http_host/jenkins/ $scheme://$host/jenkins/;
                proxy_redirect off;

                #limit_req zone=ddos_req burst=10;

                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header X-Forwarded-Port $server_port;
                proxy_set_header X-Forwarded-Host $host;

                proxy_cache_bypass 1;
                proxy_request_buffering off;
                proxy_buffering off;

        }

        location / {
                #proxy_pass http://localhost:3000;
                proxy_pass http://10.152.183.124:3000;
                #limit_req zone=ddos_req burst=10;
        }

        location /api {
                #proxy_pass http://localhost:8080/api;
                proxy_pass http://10.152.183.140:8080/api;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                #limit_req zone=ddos_req burst=10;
        }

        location /wssample {
```

```
                #proxy_pass http://localhost:8000;
                proxy_pass http://10.152.183.165:8000;
                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-Forwarded-For $proxy_add_x_forwarded_for;
        }

        location /ws {
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header X-Forwarded-Port $server_port;
                proxy_set_header X-Forwarded-Host $host;

                #proxy_pass http://localhost:8000;
                proxy_pass http://10.152.183.165:8000;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
                #limit_req zone=ddos_req burst=10;

                proxy_redirect off;

                # Additional headers for WebSocket
                proxy_set_header Sec-WebSocket-Extensions $http_sec_websocket_extensions;
                proxy_set_header Sec-WebSocket-Key $http_sec_websocket_key;
                proxy_set_header Sec-WebSocket-Version $http_sec_websocket_version;

         }

    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/heart-ing.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/heart-ing.com/privkey.pem;

}

server {

    if ($host = heart-ing.com) {
        return 301 https://$host$request_uri;
    }

    listen 80;
    server_name heart-ing.com;

    return 404;
}
###########

sudo ln -s /etc/nginx/sites-available/configure /etc/nginx/sites-enabled/configure

sudo nginx -t # ok 시 성공

sudo systemctl restart nginx
```

3.4. EC2 Port

Port 번호	내용
22	SSH
80	HTTP (HTTPS로 redirect)
443	HTTPS
3000	Nginx, React (Docker)
3306	MySQL
6379	Redis (Cache)
6380	Redis (Node.js 관리)
8000	Node.js
8080	Spring Boot (Docker)
8081	Jenkins
9000	Sonarqube
10443	K8S Dashboard

16443	Microk8s

3.5. 방화벽(UFW) 설정

```
# 1. 해당 포트 개방
# 22 TCP
# 80 TCP
# 443 TCP
# 3000 TCP
# 3306 TCP
# 8080 TCP
# 8081 TCP
# 16443 TCP
# 예시
sudo ufw allow 22/tcp

# 2. Firewall 활성화 / 상태 확인
sudo ufw enable
sudo ufw status verbose

# 3. Nginx reverse proxy 설정 후 Frontend, Backend, Jenkins 서버 포트 닫기
sudo ufw deny 3000/tcp # React Nginx
sudo ufw deny 8000/tcp # NodeJS
sudo ufw deny 8080/tcp # Spring Boot
sudo ufw deny 8081/tcp # Jenkins
sudo ufw deny 9000/tcp # Sonarqube
sudo ufw deny 10443/tcp # K8sDashboard
```

3.6. Git Username/Password 설정

```
# 1시간 동안 git 설정 (username, pw) 캐시에 저장
git config --global credential.helper "cache --timeout=3600"
```

4. CI/CD 구축

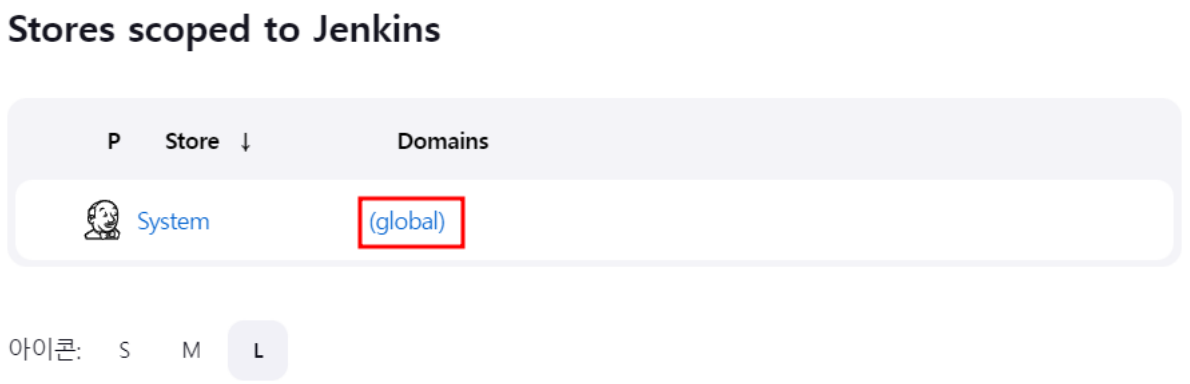
4.1. Jenkins 도커 이미지 + 컨테이너 생성

```
docker run --name jenkins -e JENKINS_OPTS="--prefix=/jenkins" -d -p 8081:8080 -p 50000:50000 -v /home/jenkins:/var/jenkins_home -v
 /var/run/docker.sock:/var/run/docker.sock -v /var/snap/microk8s/current/credentials/mykubeconfig.yml:/var/jenkins_home/kubeconfig.
yml -u root youngmookk/hearting-jenkins:v2

### 추가 설치 플러그인
- GitLab
- NodeJS

### youngmookk/hearting-jenkins:v2
FROM jenkins/jenkins:lts

ENV DEBIAN_FRONTEND noninteractive
ENV DEBCONF_NOWARNINGS="yes"

USER root
RUN apt-get -y update && apt-get install -y --no-install-recommends \
    vim \
    apt-utils
RUN apt-get install ca-certificates curl gnupg lsb-release -y
RUN mkdir -p /etc/apt/keyrings
RUN curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
RUN 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
RUN apt-get -y update
RUN apt-get install docker-ce docker-ce-cli containerd.io docker-compose docker-compose-plugin -y

# Install kubectl
RUN curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
RUN echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | tee -a /etc/apt/sources.list.d/kubernetes.list
RUN apt-get update && apt-get install -y kubectl

RUN if [ -e /var/run/docker.sock ]; then chown jenkins:jenkins /var/run/docker.sock; fi
RUN usermod -aG docker jenkins
USER jenkins
```

4.2. Jenkins 설정

4.2.1. GitLab Credentials 설정

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

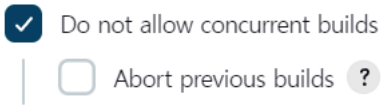


3. “Kind”에 “Username with password” 입력 → “Username”에 GitLab ID 입력 → “Password”에 Gitlab Personal Access Tokens 입력 → “ID”에 임의 아이디 입력 → 생성
   *** Personal Access Token은 Gitlab > User Settings > Access Tokens에서 생성

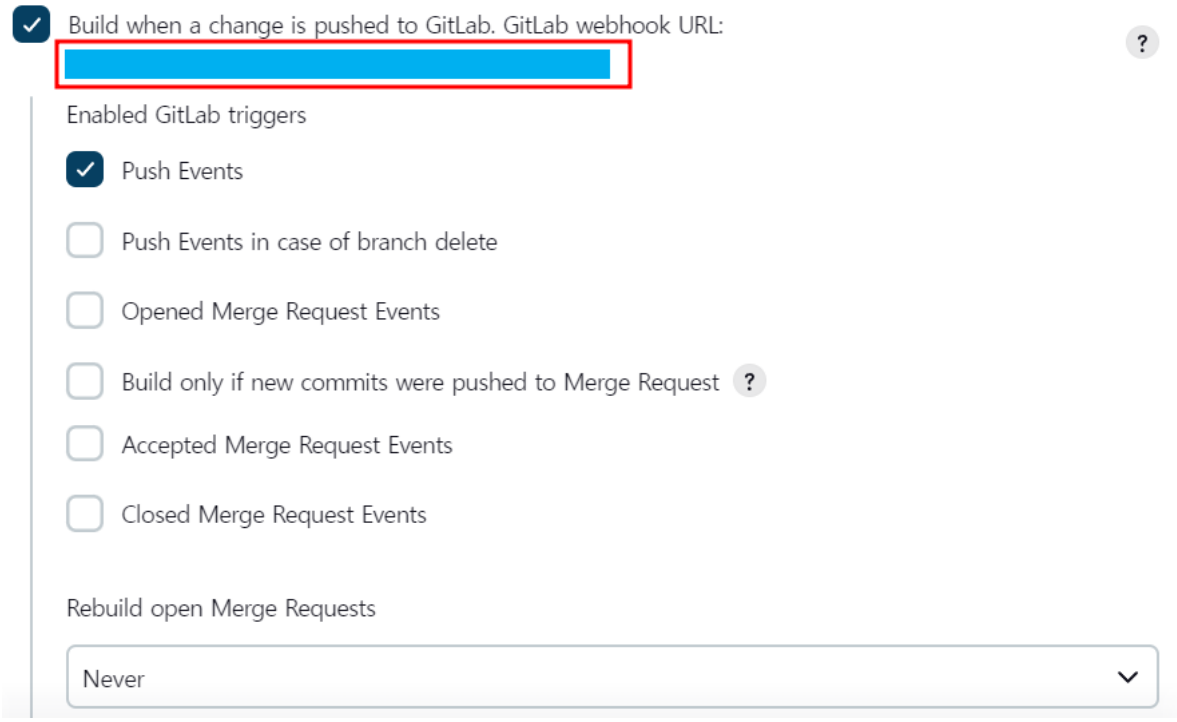
4.2.2. Jenkins Item 생성

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

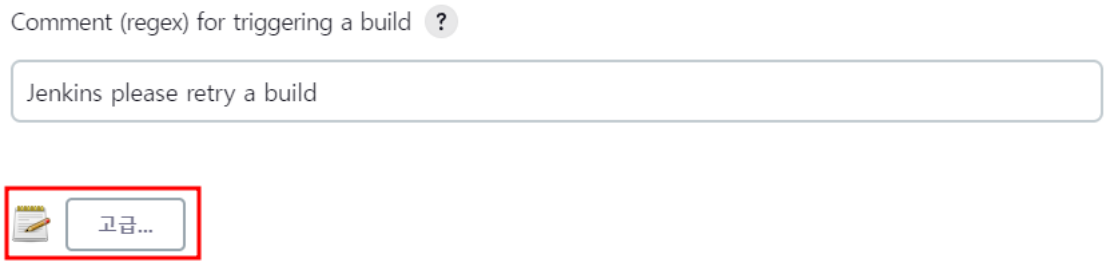
3. “General” → “Do not allow concurrent builds” 클릭
   (한 빌드를 진행중이면 동시에 빌드를 진행하지 않게 한다)



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

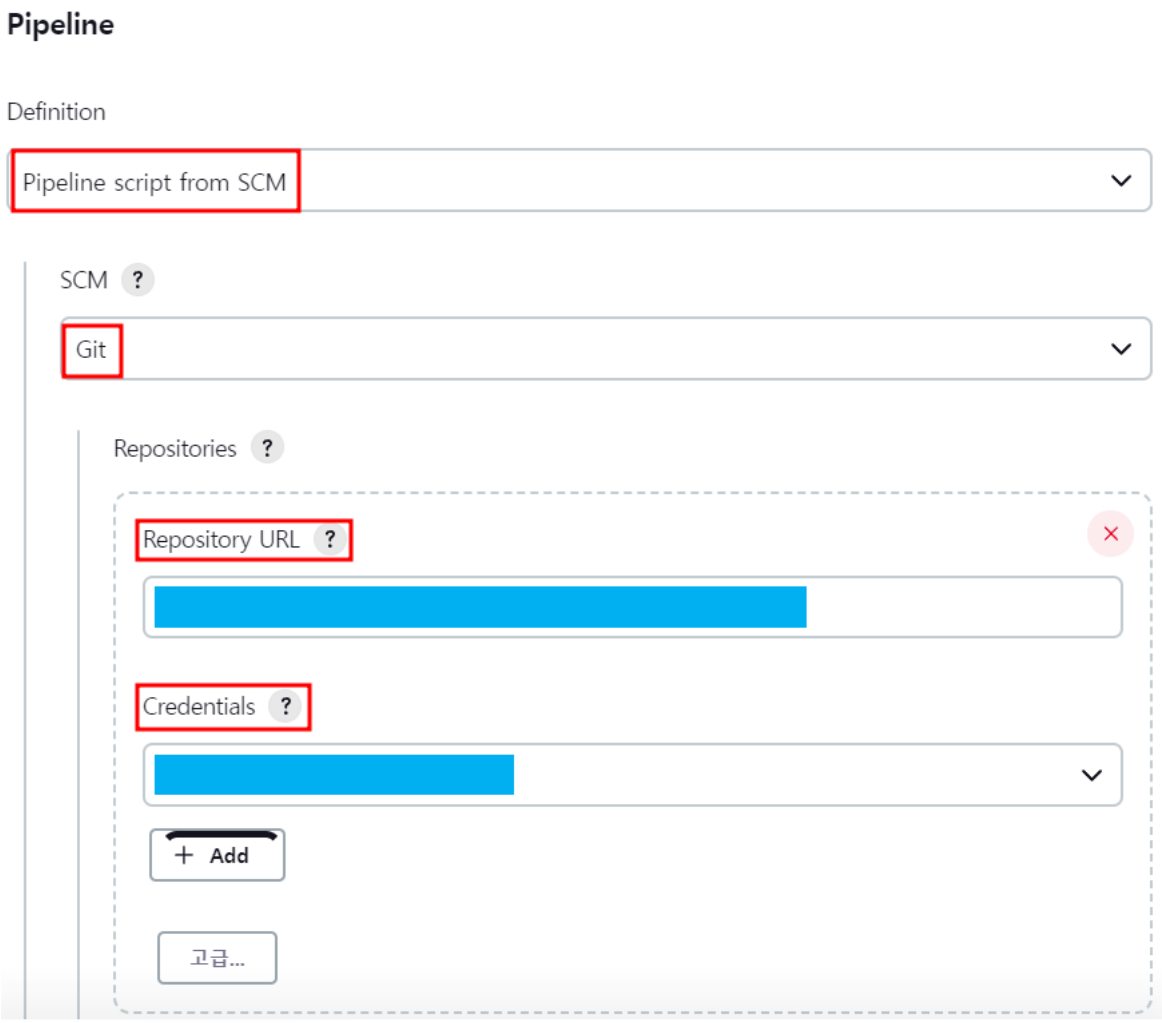


5. “Build when a change is pushed to GitLab” 하위의 “고급…” 클릭



6. “Secret token”의 “Generate” 클릭 후 생성된 토큰값 복사

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

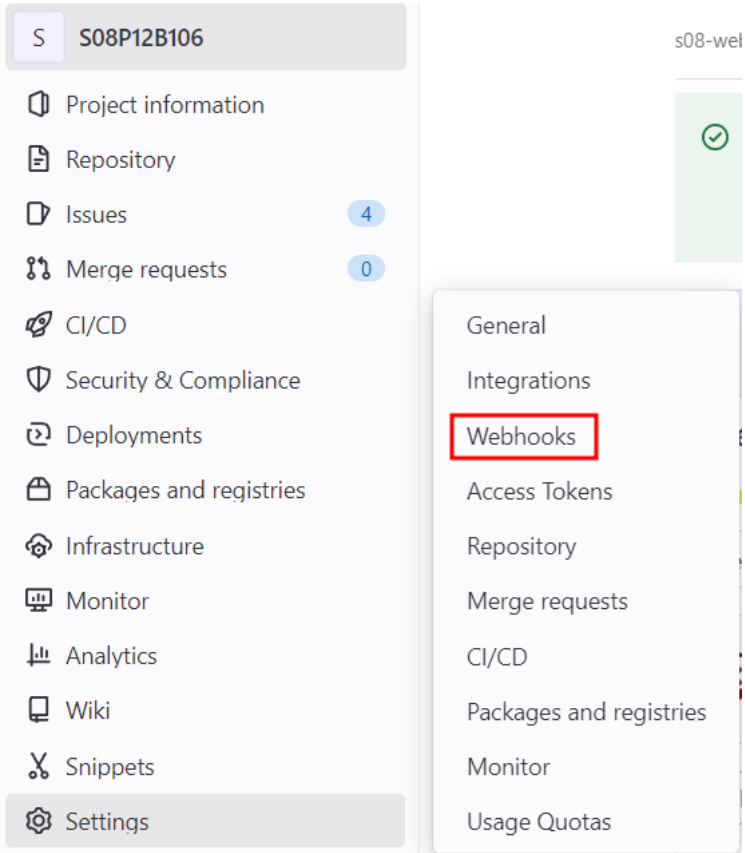


8. “Branch Specifier”에 빌드 할 브랜치명 입력 (master일 시 “*/master)

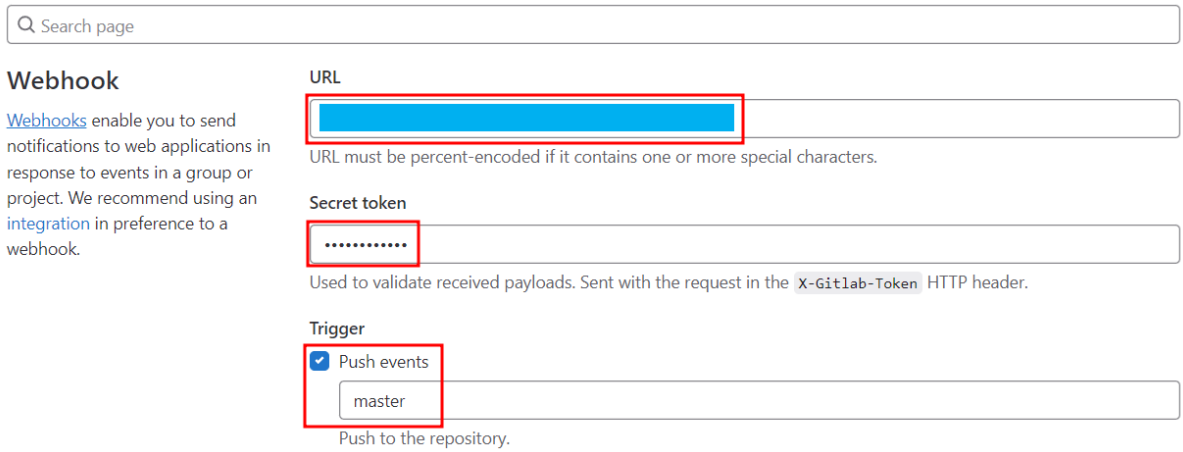
9. “Script Path”에 Jenkinsfile 경로 입력 → “Lightweight checkout” 해제 → 저장

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 → “Push events” 클릭 후 WebHook 적용 브랜치 입력 (Jenkins Branch Specifier과 일치하여야 함)



4.2.4. 빌드 및 배포

> Option 1. 상기 WebHook 설정한 브랜치로 푸시

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

4.2.5. Jenkins Gradle + NodeJS 설정

> Jenkins 관리 → Global Tool Configuration → Gradle, NodeJS 설정

5. Redis 설정

5.1. Redis 설정

```
docker pull redis

docker network create redis-net

docker run -p 6379:6379 --name hearting-redis -v /docker/redis/data:/data --network redis-net -d redis redis-server --appendonly ye
s --requirepass "hEARTING307"
```

5.2. Docker redis-cli 접속

```
docker run -it --network redis-net --rm redis redis-cli -h hearting-redis -a hEARTING307
```

6. Sonarqube 설정

6.1. Jenkins-Sonarqube 기본 설정

- 참고: 소나큐브 공식문서

Jenkins integration

SonarScanners running in Jenkins can automatically detect branches and pull requests in certain jobs.

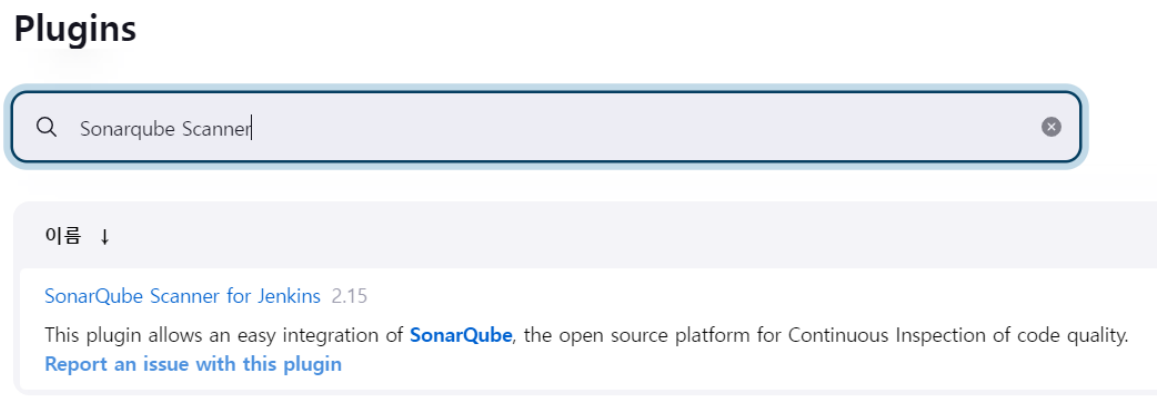
https://docs.sonarqube.org/9.7/analyzing-source-code/ci-integration/jenkins-integration/

Jenkins extension for SonarQube

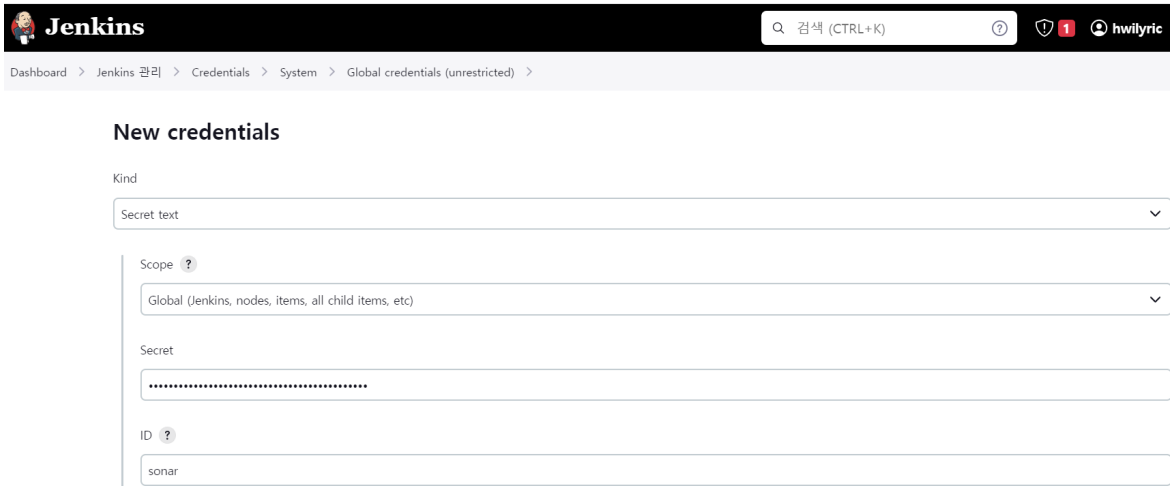
This plugin lets you centralize the configuration of SonarQube server connection details in Jenkins global configuration.

https://docs.sonarqube.org/9.7/analyzing-source-code/scanners/jenkins-extension-sonarqube/

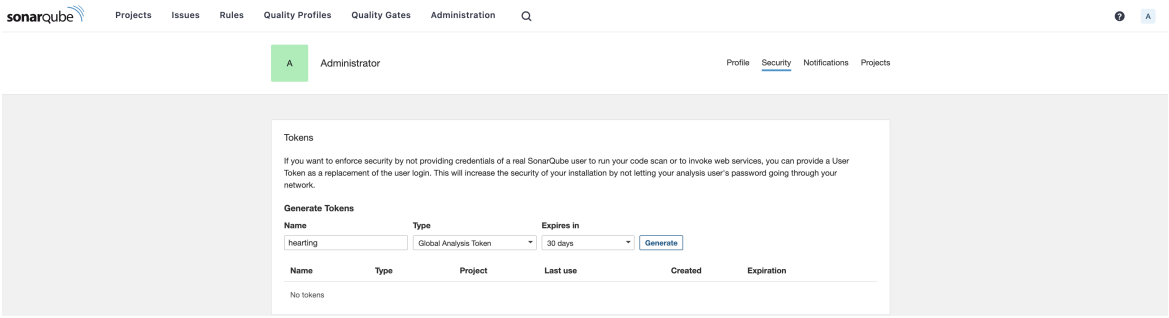
1. **(Jenkins) Dashboard > Jenkins 관리 > Plugin Manager 에서 Sonarqube Scanner 설치**



2. **(Jenkins) Dashboard > Jenkins 관리 > Credentials > System > Global credentials 계정 생성**

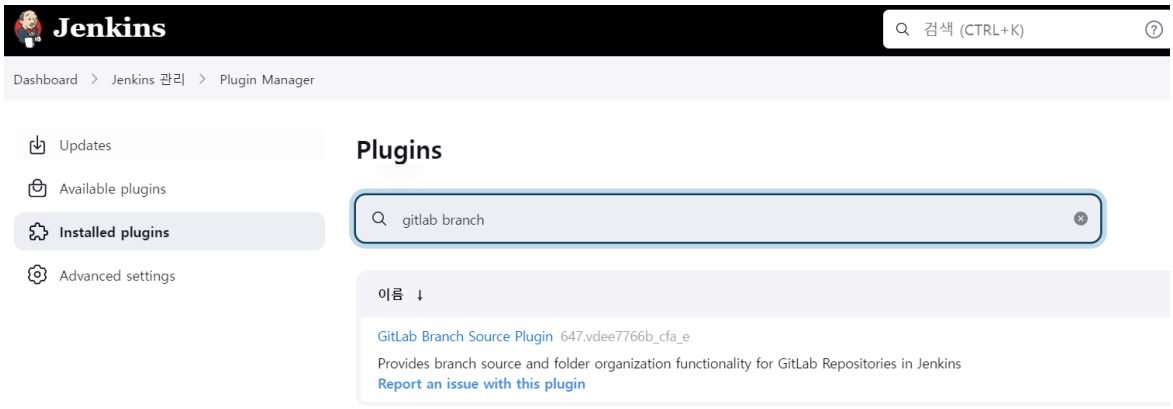


(Sonarqube) Sonarqube 에서 토큰 발급 후 Secret에 입력



3. **(Jenkins) Dashboad > Manage Jenkins > Configure System > SonarQube servers 세팅**

4. **(Jenkins) Dashboad > Manage Jenkins > Manage Plugins > Gitlab Branch source 설치**



5. **(Jenkins) Dashboad > Manage Jenkins > Configure System > Gitlab(Gitlab Servers) 서버 추가**

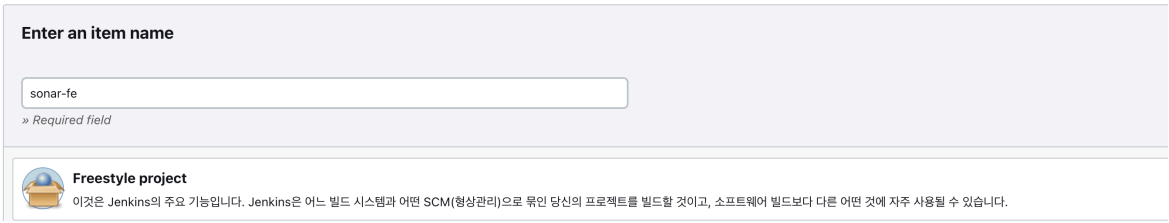
- Display Name, Server URL 입력
- Manage Web Hooks 체크
- 하단에 Test Connection 해보기

6. **(Jenkins) Dashboard > Global Tool Configuration > SonarQube Scanner**
   a. 소나큐브 등록

7. **(Jenkins) Dashboard > Global Tool Configuration > Gradle, NodeJS 세팅 확인**

6.2. Jenkins-Sonarqube with Freestyle Project

1. **New Items> Freestyle project 생성**



2. **소스코드 관리 > Git 설정**
   a. Repository URL: 분석할 레포지토리 url

b. Credentials: 레포지토리에 권한을 가진 계정 선택 (없으면 생성)

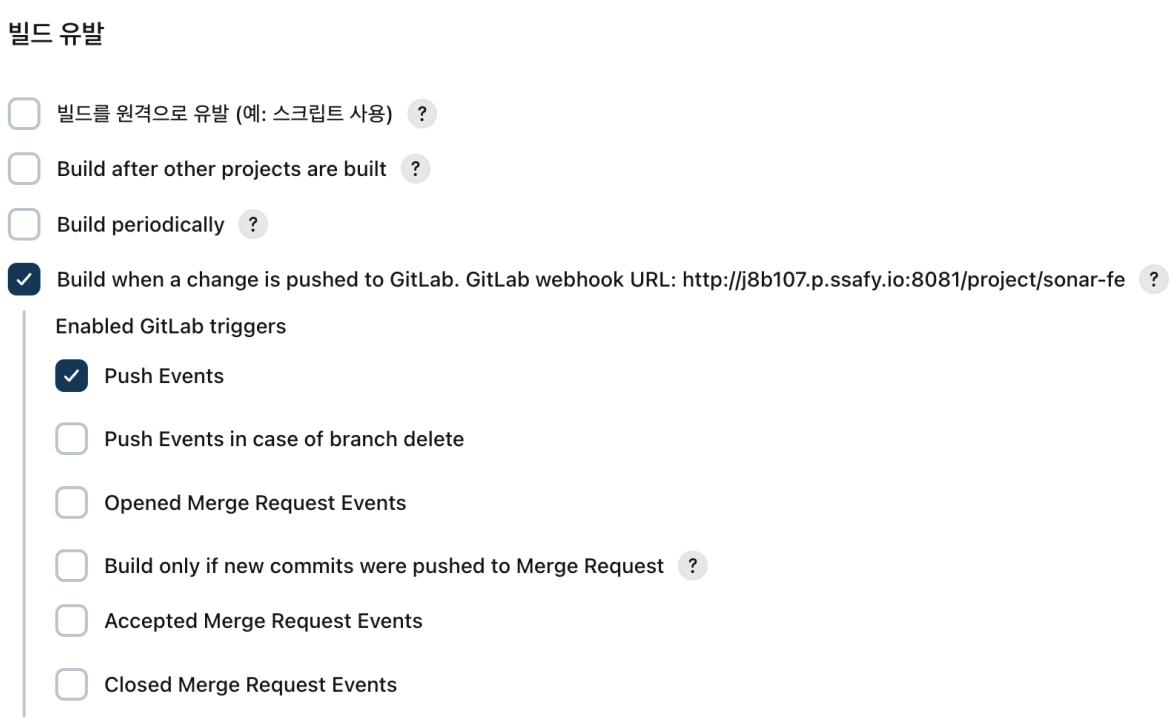
c. Branch Specifier: 분석할 브랜치 선택

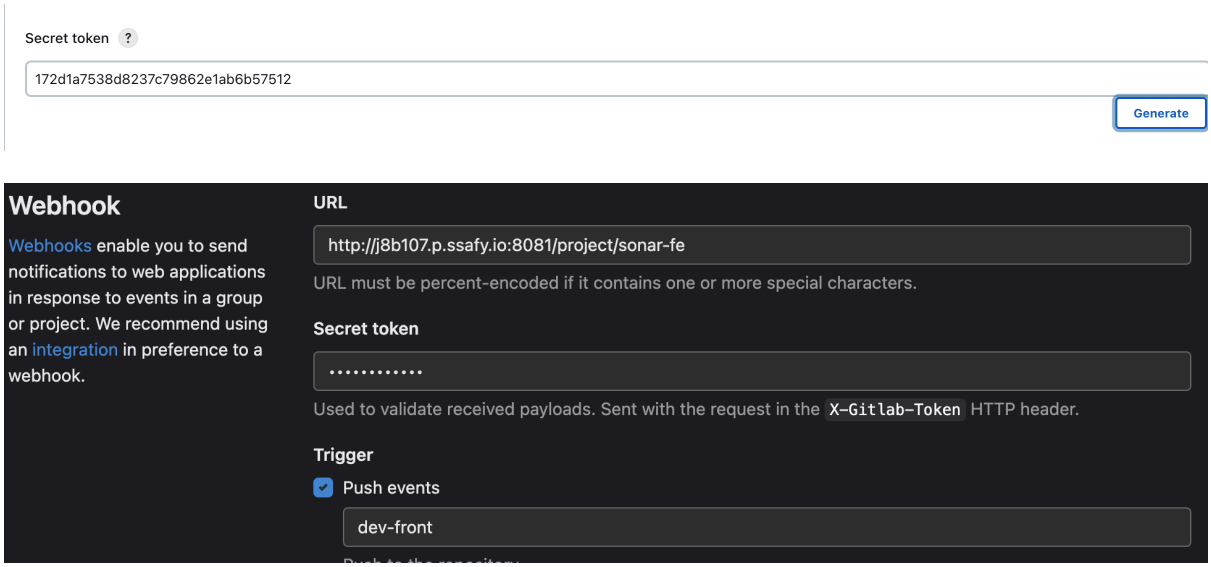
3. **빌드유발 > Build when a change is pushed to Gitlab 선택**

a. Gitlab webhook URL 복사 후 Gitlab > Settings > Webhooks 에 붙여넣기

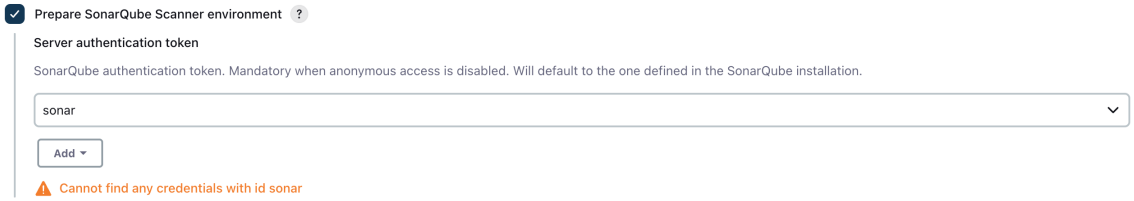
b. Push Events 선택

c. 고급 > secret token 생성 후 Gitlab > Settings > Webhooks에 붙여넣기



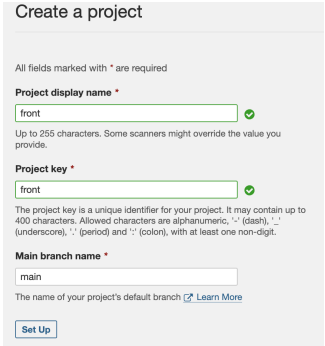


4. **빌드 환경 > Prepare SonarQube Scanner environment 체크**
   a. sonarqube 권한 가진 계정 선택



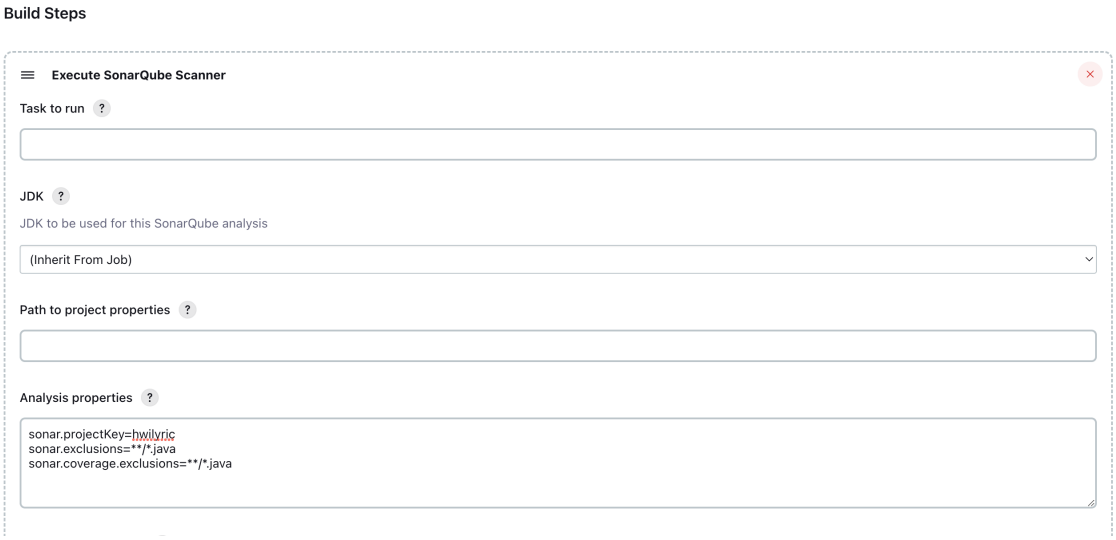
5. **빌드 환경 > Provide Node & npm bin/ folder to PATH 체크**

6. **Sonarqube로 이동 후 Sonarqube 프로젝트 생성**
   a. sonarqube 접속 (ex: 도메인:9000) > create Project > manually 선택
   b. Project display name 설정 후 Project Information > Project key 복사



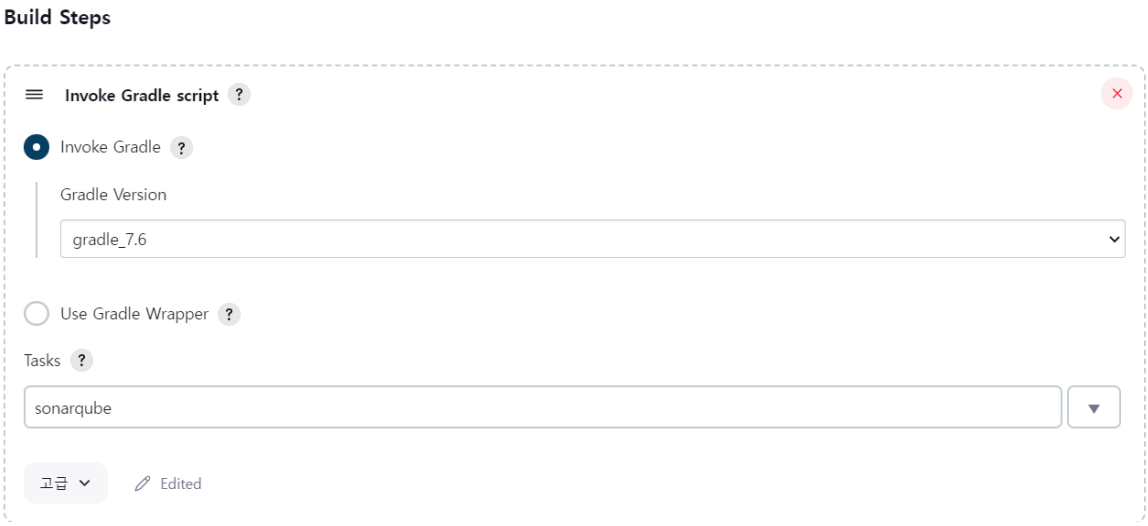
7. **Build Steps > Add build step 후 Execute SonarQube Scanner 선택**
    a. Analysis properties 작성

    sonar.projectKey={PROJECT_KEY}
    sonar.exclusions=****/*.java**
    sonar.coverage.exclusions=**/*.java
    sonar.python.version=3.8



8. **Build Steps > Add build step 후 Invoke Gradle script 선택**

- Invoke Gradle script 선택
- gradle 버전 선택
- Task에 sonarqube 입력



7. Kubernetes 설치 및 적용

7.1. Microk8s 설치

```
sudo snap install microk8s --classic
sudo usermod -a -G microk8s ubuntu
sudo chown -R ubuntu ~/.kube
newgrp microk8s
echo "alias kubectl='microk8s.kubectl'" >> ~/.bashrc
microk8s enable dns
```

7.2. Microk8s IP 설정

> Jenkins 내부에서 microk8s의 설정을 활용할 수 있게 하기위해 client.config파일에 IP만 바꿔서 다른 이름으로 저장

```
cd /var/snap/microk8s/current/credentials

sudo cp client.config mykubeconfig.yml

sudo vi mykubeconfig.yml

###

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: 데이터
    # 빨간 IP는 EC2의 내부IP로 설정; 기존 client.config는 localhost로 되어있지만, Jenkins에서 localhost로 접근할 수 없다
    server: https://172.26.12.124:16443
  name: microk8s-cluster
contexts:
- context:
    cluster: microk8s-cluster
    user: admin
  name: microk8s
current-context: microk8s
kind: Config
preferences: {}
users:
- name: admin
  user:
    token: a3pYaXVDRm4raXFGTVord2NFNEorOHY2T0NrVkhhWS8zR1VEQy93ODZOdz0K

###
# Jenkins 컨테이너에 만든 mykubeconfig.yml을 볼륨마운트로 추가
docker run --name jenkins -e JENKINS_OPTS="--prefix=/jenkins" -d -p 8081:8080 -p 50000:50000 -v /home/jenkins:/var/jenkins_home -v
 /var/run/docker.sock:/var/run/docker.sock -v /var/snap/microk8s/current/credentials/mykubeconfig.yml:/var/jenkins_home/kubeconfig.
yml -u root youngmookk/hearting-jenkins:v2
```

7.3. Microk8s IP 설정

```
# Nginx config 파일에서 해당 서버들의 clusterIP로 분기

kubectl get svc
# 각 서버에 해당하는 clusterIP주소를 아래 nginx.conf 파일에 명시

###
server {

        location / {
                proxy_pass http://10.152.183.124:3000;
        }

        location /api {
                proxy_pass http://10.152.183.140:8080/api;
        }

        location /ws {
                proxy_pass http://10.152.183.165:8000;
         }
}

###
```

7.4. Kubernetes Dashboard 설정

1. Microk8s Dashboard 활성화

```
microk8s enable dashboard
```

2. Microk8s Token 추출

```
token=$(microk8s kubectl -n kube-system get secret | grep default-token | cut -d " " -f1)
microk8s kubectl -n kube-system describe secret $token

# 토큰값 복사해놓기
```

3. Port Forwarding 적용

```
microk8s kubectl port-forward -n kube-system service/kubernetes-dashboard 10443:443
```

4. heart-ing.com/k8sdashboard로 접속해서 토큰 등록

8. Redis Cluster 설정 (NodeJS Websocket)

💡 **구축 사유** : Kubernetes Deployment를 활용하여 다수의 NodeJS 서버를 관리한다. NodeJS 서버는 Client와 websocket 프로토콜을 활용하여 통신하기 때문에 하나의 NodeJS 서버에 연결되어있는 동안에는 그 서버에 종속된다 (stateful한 커넥션을 한다). 따라서, Client A가 Client B에게 event를 날리려고 하는데 서로 다른 NodeJS 서버에 연결되어 있으면 그 event를 전달해 줄 미들웨어가 필요하다. 따라서, Redis Cluster를 구축하고 NodeJS 서버들과 연동시켜 websocket 기능이 차질없이 한다.

1. Kubernetes StatefulSet 으로 Redis Cluster 구축 yml 파일 생성

```
apiVersion: v1
kind: Service
metadata:
  name: hearting-redis-cluster
spec:
  ports:
    - port: 6380
      name: client
  clusterIP: None
  selector:
    app: hearting-redis-cluster

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: hearting-redis-cluster
spec:
  serviceName: "hearting-redis-cluster"
  replicas: 6
  selector:
    matchLabels:
      app: hearting-redis-cluster
  template:
    metadata:
      labels:
        app: hearting-redis-cluster
    spec:
      containers:
        - name: redis
          image: redis:6.2
          command:
            - "redis-server"
            - "--port"
            - "6380"
            - "--cluster-enabled"
            - "yes"
          ports:
```

```
            - containerPort: 6380
              name: client
```

💡 **StatefulSet를 사용하는 이유**
1. 각 노드에게 안정적인 hostname이 정해진다 (예: hearting-redis-cluster-0, hearting-redis-cluster-1 등등) / Deployment는 노드의 hostname이 배포할때마다 바뀐다.
2. 시작 종료 순서가 순차적으로 진행된다 (시작은 0부터 N-1까지, 종료는 반대방향)
(Redis 같은 분산 DB같은 경우 빠른 quorum 구축이 필요하기 때문에 순차적으로 시작을 하는게 빠른 시스템 구축에 더 도움이 된다)

2. kubectl apply -f hearting-redis-cluster.yml로 Kubernetes StatefulSet 배포 (추후에는 신규버전 배포 시 Jenkins Pipeline에서 갱신할 예정)

3. Redis Cluster 구축

```
kubectl exec -it hearting-redis-cluster-0 -- redis-cli --cluster create --cluster-replicas 1 $(kubectl get pods -l app=hearting-red
is-cluster -o jsonpath='{range .items[*]}{.status.podIP} ' | xargs -n 1 echo | awk '{print $1":6380"}' | xargs)

###
# kubectl get pods -l app=hearting-redis-cluster -o jsonpath='{range .items[*]}{.status.podIP} ' | xargs -n 1 echo | awk '{print
 $1":6380"}' | xargs

# => 해당 command는 현재 redis pod의 내부IP주소와 :6380을 붙힌 값을 띄어쓰기 단위로 나눠서 출력한다. 즉 Redis Cluster 구성원들의 주소를 출력한다.
```

4. NodeJS에서 redis 및 @socket.io/redis-adapter 패키지를 설치하고 구축한 redis cluster과 NodeJS를 연결한다.

```
const { createClient } = require("redis");
const { createAdapter } = require("@socket.io/redis-adapter");

const pubClient = createClient({
  url: "redis://hearting-redis-cluster:6380",
});
const subClient = pubClient.duplicate();

Promise.all([pubClient.connect(), subClient.connect()]).then(() => {
  io.adapter(createAdapter(pubClient, subClient));
});
```

redis client를 publisher (socket event를 발송하는 역할), subscriber(socket event를 받는 역할) 하나씩 생성하여 NodeJS 서버에 redis adapter를 생성하여 부착한다.

9. 외부서비스

9.1. 소셜 로그인 - Kakao

> **Kakao Developers REST API (카카오 로그인)**

✔️ OAuth 기반 소셜 로그인 API 제공

https://developers.kakao.com/docs/latest/ko/kakaologin/rest-api

9.1.1. 애플리케이션 생성

1. Kakao developer에서 로그인 후 애플리케이션 추가
2. 카카오 로그인을 활성화

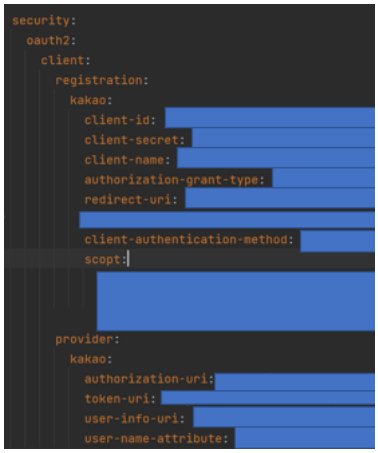
3. 사이트 도메인, Redirect URI 등록

9.1.2. 키 생성, 동의 항목 선택

1. 앱 키 메뉴에서 REST API 키를 가져와 git에 올리지 않을 yml 파일 작성
2. 카카오 로그인 동의 항목 설정

9.1.3. application-oauth.yml 작성

1. 외부로 노출 안 되도록 application-oauth.yml파일에 키와 관련 내용 작성



9.1.4. 카카오로부터 사용자 정보 얻어오기

1. 카카오가 보내준 `인가코드` 를 Front에서 Back으로 전달

2. Back에서 카카오에게 인가코드를 전달하고 `access-token` 받아오기
3. access-token으로 카카오에 저장되어 있는 `user 정보`를 받아오기

9.2. 소셜 로그인 - Google

Google Cloud (구글 로그인)

✔️ OAuth 기반 소셜 로그인 API 제공

https://console.cloud.google.com/

9.1.1. 애플리케이션 생성

1. Google Cloud에서 로그인 후 애플리케이션 추가
2. OAuth 동의 화면에서 애플리케이션 정보 입력, 동의 항목 선택 후 저장
2. 사이트 도메인, Redirect URI 등록

9.1.2. 키 생성

1. 사용자 인증 정보에서 클라이언트 ID, 클라이언트 보안 비밀번호 git에서 올리지 않을 yml에 작성

9.1.3. application-hearting.yml 작성

1. 외부로 노출 안 되도록 application-hearting.yml파일에 키와 관련 내용 작성

9.1.4. 구글로부터 사용자 정보 얻어오기

1. 구글이 보내준 `인가코드` 를 Front에서 Back으로 전달
2. Back에서 구글에게 인가코드를 전달하고 `access-token` 받아오기
3. access-token으로 구글에 저장되어 있는 `user 정보`를 받아오기

9.2. 소셜 로그인 - Twitter

Twitter Developer Portal (트위 로그인)

✔️ OAuth 기반 소셜 로그인 API 제공

https://developer.twitter.com/en/portal/dashboard

9.1.1. 애플리케이션 생성

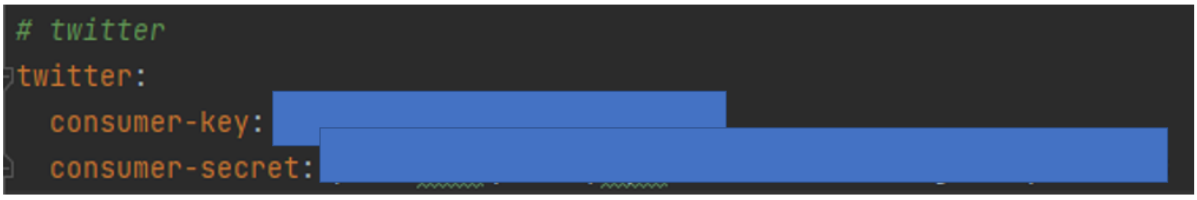
1. Twitter Developer Portal에서 로그인 후 애플리케이션 추가
2. User authentication settings 화면에서 애플리케이션 정보 입력, 동의 항목 선택 후 저장
2. Request email from users 활성화 해두기 사이트 도메인, Redirect URI 등록

9.1.2. 키 생성

1. Projects & Apps > 프로젝트명 에서 consumer-key, consumer-secret git에서 올리지 않을 yml에 작성

9.1.3. application-hearting.yml 작성

1. 외부로 노출 안 되도록 application-hearting.yml파일에 키와 관련 내용 작성



9.1.4. 트위터로부터 사용자 정보 얻어오기

1. Front에서 Back으로 redirect URL을 get 요청
2. Back에서 트위터에게 `consumer-key`, `consumer-secret`를 담아서 `request token` 받아오기
3. Back에서 requset token으로 redirectURL 생성해서 Front로 반환하기
4. Front에서는 redirectURL 로 바로 redirect 하기
5. 사용자는 트위터 로그인 진행
6. 트위터에서 보내준 oauthToken, oauthVerifier를 가지고 `access token`을 받아오기
7. TwitterTemplate 객체 생성
8. TwitterTemplate 객체로 `user 정보` 받아오기