|  |
| --- |
|  |

|  |
| --- |
| **에그로그** |
| **포팅 메뉴얼** |
| 광주 자율 C204 |
|  |
|  |
| C:\Users\SSAFY\AppData\Local\Microsoft\Windows\INetCache\Content.Word\호남향우회.png |
|  |



목차

[I. 개요 2](#_Toc163053853)

[1. 프로젝트 개요 2](#_Toc163053854)

[2. 프로젝트 사용 도구 2](#_Toc163053855)

[3. 개발환경 2](#_Toc163053856)

[4. 외부 서비스 2](#_Toc163053857)

[5. Gitgnore 처리한 핵심 키들 3](#_Toc163053858)

[II. 빌드 3](#_Toc163053859)

[1. 환경변수 형태 3](#_Toc163053860)

[2. 빌드하기 5](#_Toc163053861)

[3. 배포하기 6](#_Toc163053862)

[4. 서비스 이용 방법 27](#_Toc163053863)

[가) 포트원 27](#_Toc163053864)

[나) Firebase 실시간 알림 28](#_Toc163053865)

# 개요

## 프로젝트 개요

간호사를 위한 근무, 일정 관리 서비스 에그로그는 간호사들의 복잡한 근무 일정(3교대 근무)로 인해 야기되는 약속 잡기의 어려움을 개선하기 위해 개발되었습니다. 간호사에게 다음과 같은 편의성을 제공합니다.

1. 복잡한 근무를 직관적으로 확인 및 관리

2. 친구의 일정과 본인의 일정을 한눈에 비교

3. 커뮤니티를 통한 정보 공유

4. 공유 캘린더 스펙의 외부 캘린더간 일정 동기화

## 프로젝트 사용 도구

이슈 관리 : JIRA

형상 관리 : Gitlab

커뮤니케이션 : Notion, Mattermost, Miro

디자인 : Figma

UCC : 모바비

CI/CD : Jenkins, Docker

## 개발환경

Kotlin : 14.16.0

IntelliJ : 17.0.9+7-b1087.9 amd64

Android Studio

JVM : OpenJDK 64-Bit Server VM by JetBrains s.r.o (스프링은 17로 빌드)

SERVER : AWS EC2 Ubuntu 20.04.3 LTS

RDS: AWS RDS

DB : MySQL 8.0.33, Firestore

## 외부 서비스

Google Cloud Storage : serviceAccountKey.json에 해당 내용 있음  
(과금이 발생할 수 있는 키입니다. 취급 주의)

Firebase Firestore :

Firebase Storage

-- kotlin --

1. jetpack compose
2. hilt
3. material3 & material
4. datastore
5. retrofit & okhttp
6. splash
7. ui controller
8. serialization
9. paging3
10. oauth 관련 google, naver, kakao
11. firebase messaging, auth, storage,
12. coil
13. navigation
14. orbit
15. webview

-- web --

1. react
2. typeScript
3. d3.js
4. styled-component

*(수정됨)*

kotlin 설정

1. minSdk = 26
2. maxSdk = 34
3. jdk version = 17
4. gradle agp version = 8.3.1
5. hilt version = 2.49.0
6. coil = 2.6.0
7. retrofit = 2.9.0
8. orbit = 6.1.0

## Gitgnore 처리한 핵심 키들

Spring : application.yml, gradle.properties

(\src\main\resources, 또는 classPath에 위치)

Flutter : google-services.json (android\app에 위치, firebase 통신을 위해 필요)

kakaoKey (location를 통한 lat, lng 값 얻어오기 위해 필요)

# 빌드

## 환경변수 형태

**1-1. .env**

REDIS\_PASSWORD=레디스 비밀번호

CONFIG\_USERNAME= 스프링 클라우드 컨피그 서버 관리자명

CONFIG\_PASSWORD= 스프링 클라우드 컨피그 서버 관리자 비밀번호

GIT\_USERNAME= 깃허브 사용자명

GIT\_PASSWORD= 깃허브 사용자 토큰

**1-2. application.yml – spring cloud config server 통해 관리(암호화 및 조회)**

**1-2-1. release**

**1-2-1-1. config server**

spring:

application:

name: config-service

profiles:

active: dev

cloud:

config:

server:

git:

uri: https://github.com/honam-hyanguhoe/duty-app-setting.git

skipSslValidation: true

force-pull: true

searchPaths:

- back/\*\*

default-label: main

username: 깃허브 계정

password: 깃허브 액세스 토큰

server:

port: 9000

admin:

username: 접속 계정

password: 접속 비밀번호

**1-2-1-2. config client**

server:

port: {서비스별 상이}

spring:

application:

name: gateway-service

config:

import: optional:configserver:http://${CONFIG\_USERNAME}:${CONFIG\_PASSWORD}@{호스트}:9000

profiles:

active: prod

cache:

type: redis

main:

web-application-type: reactive

management:

endpoints:

enabled-by-default: false

web:

base-path: /actuator

exposure:

include: ['refresh','info', 'health']

jmx:

exposure:

exclude: ['\*']

**1-2-2. main – spring cloud config server를 통해 통신**

**파일 구조: {프로젝트명}-{프로필}.yml**

## 빌드하기

1. Front

gradle build

2) Back(Spring cloud)

2-1) config -> service

2-2) 실행 방법

Gradle 실행

Bootjar 실행

## 배포하기

**3-1. Nginx 설정**

**nginx.conf**

worker\_processes auto;

events { worker\_connections 2048; }

http {

limit\_req\_zone $binary\_remote\_addr zone=limit\_request\_per\_ip:10m rate=10r/s;

map $http\_user\_agent $bad\_bot {

default 0;

~\*(^MJ12bot|^MJ12bot/v1.4.5|masscan|Mass\ Downloader|maverick|SemrushBot|SemrushBot-SA|DomainCrawler|MegaIndex.ru|AlphaBot|Paros|ZmEu|nikto|dirbuster|sqlmap|openvas|w3af|Morfeus|Zollard|Arachni|Brutus|bsqlbf|Grendel-Scan|Havij|Hydra|N-Stealth|Netsparker|Pangolin|pmafind|webinspect) 1;

}

include mime.types;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /dev/stdout main;

error\_log /dev/stderr;

upstream api {

# 배포 중인 서버들

server k10c204.p.ssafy.io:8010;

#health-check

# interval - 3초씩에 한 번씩 확인

# rise - 2번 이상 응답에 성공하면 서버가 살은 것으로 판단

# fall - 5번 이상 응답에 실패할 경우 서버가 죽은 것으로 판단

# timeout - 응답 시간 초과 1초

check interval=10000 rise=2 fall=5 timeout=1000 type=http;

# GET /actuator/health으로 healthcheck 요청

check\_http\_send "GET /actuator/health HTTP/1.0\r\n\r\n";

check\_http\_expect\_alive http\_2xx http\_3xx;

}

# 80 포트 설정 포함

include /usr/local/nginx/conf/nginx\_80.conf;

# 443 포트 설정 포함

include /usr/local/nginx/conf/nginx\_443.conf;

}

**nginx\_80.conf**

server {

listen 80;

server\_name api.egg-log.org;

server\_tokens off;

location /.well-known/acme-challenge/ {

allow all;

root /var/www/certbot;

}

return 301 https://$host$request\_uri;

}

server {

listen 80;

server\_name jenkins.egg-log.org;

server\_tokens off;

location /.well-known/acme-challenge/ {

allow all;

root /var/www/certbot;

}

return 301 https://$host$request\_uri;

}

server {

listen 80;

server\_name www.egg-log.org;

server\_tokens off;

location /.well-known/acme-challenge/ {

allow all;

root /var/www/certbot;

}

return 301 https://$host$request\_uri;

}

server {

listen 80;

server\_name ngrinder.egg-log.org;

server\_tokens off;

location /.well-known/acme-challenge/ {

allow all;

root /var/www/certbot;

}

return 301 https://$host$request\_uri;

}

**nginx\_443.conf**

server {

listen 443 ssl;

server\_name jenkins.egg-log.org;

server\_tokens off;

location /{

# 나쁜 봇을 차단

if ($bad\_bot) {

return 403;

}

# app 서비스로 라우팅

proxy\_pass http://jenkins/;

proxy\_redirect off;

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;

}

}

server {

listen 443 ssl;

server\_name api.egg-log.org;

server\_tokens off;

location /{

# 나쁜 봇을 차단

if ($bad\_bot) {

return 403;

}

# app 서비스로 라우팅

proxy\_pass http://api;

proxy\_redirect off;

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;

}

}

server {

listen 443 ssl;

server\_name www.egg-log.org;

server\_tokens off;

location /{

# 나쁜 봇을 차단

if ($bad\_bot) {

return 403;

}

# app 서비스로 라우팅

proxy\_pass http://web;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

location /ws{

# 나쁜 봇을 차단

if ($bad\_bot) {

return 403;

}

# app 서비스로 라우팅

proxy\_pass http://web;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

proxy\_set\_header Host $host;

proxy\_set\_header Origin "";

}

}

server {

listen 443 ssl;

server\_name ngrinder.egg-log.org;

server\_tokens off;

location /{

if ($bad\_bot){

return 403;

}

proxy\_pass http://ngrinder;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

**3-2. Jenkins Jobs**

**3-2-1. 젠킨스 파이프라인**

**3-2-1-1. 스프링부트 기반 전체 파이프라인 구조**

pipeline {

agent any

stages {

stage('Checkout Main Repo') {

steps {

checkout scm

}

}

stage('Move .env File') {

steps {

script{

sh "cp ../environments/back/.env api/.env"

sh "mkdir -p api/src/main/resources"

// find 명령어를 사용하여 .yml 파일을 찾고, 각 파일에 대해 cp 명령어를 실행

//sh "cp ../environments/back/egg-log-service.yml api/src/main/resources/application.yml"

sh "cp ../environments/back/egg-log-service-test.yml api/src/main/resources/application.yml"

sh "cp ../environments/back/eggLogPrivateKey.jks api/src/main/resources/eggLogPrivateKey.jks"

sh "cp ../environments/back/firebaseSocon.pem api/src/main/resources/firebaseSocon.pem"

sh "cp ../environments/gradle.properties api/gradle.properties"

}

}

}

stage('Setup Environment') {

steps {

script{

sh "chmod +x ./api/gradlew"

}

}

}

stage('Docker Build & Push to Nexus') {

steps {

dir("api") {

script {

def dockerImageTag = "${env.DOCKER\_REGISTRY\_URL}/${env.DOCKER\_IMAGE\_NAME}-test:latest"

sh """

docker build -t ${dockerImageTag} .

echo ${env.DOCKER\_REGISTRY\_CREDENTIALS\_PASSWORD} | docker login -u ${env.DOCKER\_REGISTRY\_CREDENTIALS\_USERNAME} --password-stdin ${env.DOCKER\_REGISTRY\_URL}

docker push ${dockerImageTag}

"""

}

}

}

}

stage('Deploy Main') {

steps {

dir("api") {

script {

try {

sh "docker-compose up -d --build"

} catch (Exception e) {

sh "echo '빌드에 실패했습니다.'"

sh "docker-compose down"

sh "docker-compose up -d --build"

}

}

}

}

}

stage('Wait for Main to be Healthy') {

steps {

script {

waitForHealthy("main-test")

}

}

}

stage('Deploy Main-GCP on Remote Server') {

steps {

script {

try{

sh """

ssh -i ${env.KEY\_PATH} -o StrictHostKeyChecking=no ${REMOTE\_FULL\_PATH} '

export DOCKER\_REGISTRY\_URL=${env.DOCKER\_REGISTRY\_URL} &&

export DOCKER\_REGISTRY\_USERNAME=${env.DOCKER\_REGISTRY\_CREDENTIALS\_USERNAME} &&

export DOCKER\_REGISTRY\_PASSWORD=${env.DOCKER\_REGISTRY\_CREDENTIALS\_PASSWORD} &&

sh ./login.sh &&

docker-compose up -d

'

"""

}catch (Exception e){

sh """

ssh -i ${env.KEY\_PATH} -o StrictHostKeyChecking=no ${REMOTE\_FULL\_PATH} '

docker-compose down &&

sh ./login.sh &&

docker-compose up -d

'

"""

}

}

}

}

stage('Wait for Main-GCP to be Healthy') {

steps {

script {

waitForHealthy("Main-gcp")

}

}

}

stage('Prune old images'){

steps{

script{

sh "yes | docker system prune --filter until=1h"

}

}

}

}

post {

always {

script {

def Author\_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()

def Author\_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()

mattermostSend (color: currentBuild.currentResult=='SUCCESS'?'good':'danger',

message: "빌드 ${currentBuild.currentResult}: ${env.JOB\_NAME} #${env.BUILD\_NUMBER} by ${Author\_ID}(${Author\_Name})\n(<${env.BUILD\_URL}|Details>)",

endpoint: 'https://meeting.ssafy.com/hooks/65u7o3o7kfb87dqcs8cxyzbdpo',

channel: 'Jenkins'

)

}

}

}

}

// 백엔드 인스턴스의 health check를 기다리는 함수

def waitForHealthy(String service) {

int port = 8020

def maxRetries = 15

def retries = 0

def isHealthy = false

if(service == "Main-gcp"){

service = env.REMOTE\_HOST

}

while (retries < maxRetries && !isHealthy) {

def result = sh(script: "curl -s http://${service}:${port}/actuator/health | jq -r .status", returnStdout: true).trim()

if (result == 'UP') {

isHealthy = true

} else {

sleep 10

retries++

}

}

if (!isHealthy) {

error("${service} 서버가 정상적으로 작동되지 않았습니다.")

}

}

**3-2-2. 도커 설정**

**3-2-2-1. 도커 네트워크 연결 설정**

동일 인스턴스 내 존재하는 컨테이너는 모두 동일한 bridge 네트워크로 연결됩니다.

**3-2-2-2. Spring boot**

**docker-compose.yml**

version: '3.8'

services:

main-test:

container\_name: main-test

image: 54.180.55.106:5000/main-test

build:

context: .

dockerfile: Dockerfile

restart: unless-stopped

volumes:

- /home/ubuntu/services/main:/var/logs/main:rw

environment:

- TZ=Asia/Seoul

ports:

- "8020:8020"

networks:

- jenkins\_egg-log-net

networks:

jenkins\_egg-log-net:

external: true

**Dockerfile – 기본 프로젝트**

# 빌드 스테이지

FROM amazoncorretto:17.0.7-alpine AS builder

USER root

WORKDIR /config

COPY gradlew .

COPY gradle gradle

COPY build.gradle .

COPY settings.gradle .

COPY src src

# gradlew 실행 권한 부여

RUN chmod +x ./gradlew

RUN ./gradlew bootJar

# 실행 스테이지

FROM openjdk:17

WORKDIR /config

COPY --from=builder /config/build/libs/\*.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]

**3-2-2-3. Nginx**

**docker-compose.yml**

version: '3.8'

services:

nginx:

container\_name: nginx

image: egg-log/nginx

build:

context: .

dockerfile: Dockerfile

restart: always

# 볼륨 매핑은 젠킨스 워크스페이스 기준

volumes:

- /etc/letsencrypt:/etc/letsencrypt

ports:

- "80:80"

- "443:443"

networks:

- test-net

networks:

test-net:

**Dockerfile**

FROM debian:bullseye-slim AS builder

ENV TZ=Asia/Seoul

# Install necessary tools

RUN apt-get update && \

apt-get install -y git wget build-essential \

libpcre3-dev openssl libssl-dev zlib1g zlib1g-dev make && \

rm -rf /var/lib/apt/lists/\*

# Set Nginx version

ENV NGINX\_VERSION 1.25.5

# Download the Nginx source code and the health check module

RUN cd /usr/local/src && \

wget http://nginx.org/download/nginx-${NGINX\_VERSION}.tar.gz && \

tar -zxvf nginx-${NGINX\_VERSION}.tar.gz && \

git clone https://github.com/yaoweibin/nginx\_upstream\_check\_module.git

# Compile Nginx with the health check module

RUN cd /usr/local/src/nginx-${NGINX\_VERSION} && \

patch -p1 < /usr/local/src/nginx\_upstream\_check\_module/check\_1.20.1+.patch && \

./configure --add-module=/usr/local/src/nginx\_upstream\_check\_module \

--with-http\_ssl\_module \

--with-http\_stub\_status\_module && \

make && make install

# Copy the built Nginx to a new clean image

FROM debian:bullseye-slim

COPY --from=builder /usr/local/nginx /usr/local/nginx

# Create symbolic links for log files to standard output and error

RUN ln -sf /dev/stdout /usr/local/nginx/logs/access.log && \

ln -sf /dev/stderr /usr/local/nginx/logs/error.log

# Remove default configuration and copy custom configuration

RUN rm -rf /etc/nginx/conf.d/default.conf

COPY nginx.conf /usr/local/nginx/conf/nginx.conf

COPY nginx\_80.conf /usr/local/nginx/conf/nginx\_80.conf

COPY nginx\_443.conf /usr/local/nginx/conf/nginx\_443.conf

CMD ["/usr/local/nginx/sbin/nginx", "-g", "daemon off;"]

**3-2-2-4. Redis**

**docker-compose.yml**

version: '3.8'

services:

redis:

container\_name: redis

hostname: redis

image: 54.180.55.106:5000/redis

build:

context: .

dockerfile: Dockerfile

restart: unless-stopped

networks:

- jenkins\_egg-log-net

volumes:

- redis-data:/data

user: root

command: redis-server /usr/local/etc/redis/redis.conf --requirepass ${REDIS\_PASSWORD}

ports:

- "6379:6379"

networks:

jenkins\_egg-log-net:

external: true

volumes:

redis-data:

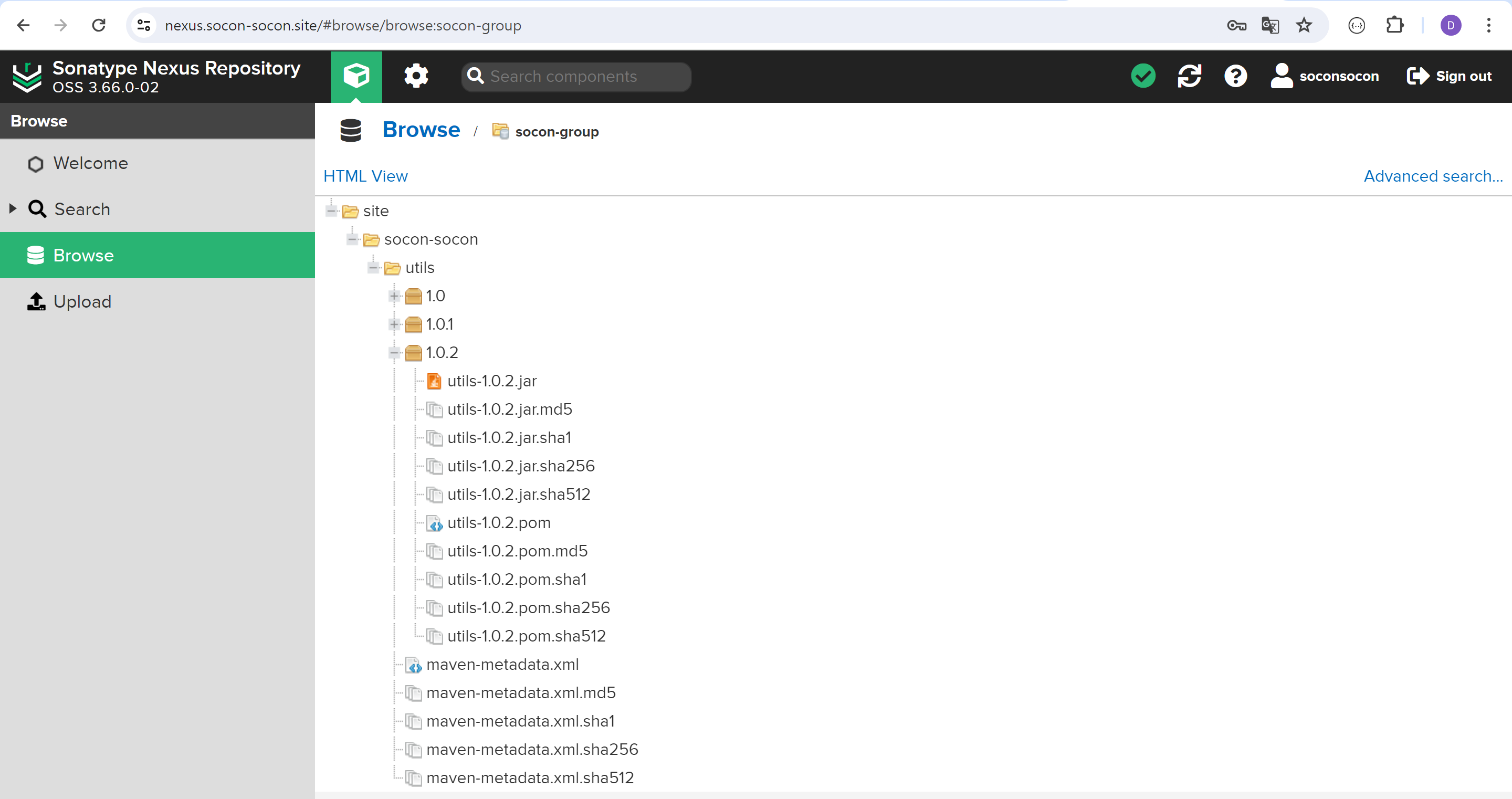
**Dockerfile**

FROM redis:7.2.4

ENV TZ=Asia/Seoul

COPY redis.conf /usr/local/etc/redis/redis.conf

**3-3. 공통 라이브러리 넥서스 사설 레포지토리 배포**



**3-3-1. 도커 설정**

**docker-compose.yml**

version: '3.3'

services:

nexus:

container\_name: nexus

image: sonatype/nexus3

restart: always

# 볼륨 매핑은 젠킨스 워크스페이스 기준

volumes:

- nexus:/sonatype-work

ports:

- "8081:8081"

networks:

- jenkins-project\_socon-net

volumes:

nexus:

networks:

jenkins-project\_socon-net:

external: true

**3-3-2. 스프링 부트 설정**

**3-3-2-1. gradle.properties**

nexusUrl=https://nexus.socon-socon.site/repository

nexusUsername=

nexusPassword=

project.name=

version=

queryDslVersion=

**3-3-2-2. build.gradle**

**publish – 주요 설정**

plugins {

id 'java'

id 'maven-publish'

id "io.github.gradle-nexus.publish-plugin" version "2.0.0-rc-2" //넥서스 플러그인

…

}

//그룹명과 버전 지정

group = 'site.socon-socon'

version = '1.0.2'

…

bootJar.enabled = false // 실행가능한 아카이브로 main-class 가 있는 모듈인경우

jar.enabled = true // PLAIN 으로 생성 실행이 불가능한 일반 아카이브

repositories {

mavenCentral()

//nexus 레포지토리 연결

maven {

credentials(PasswordCredentials) {

username = "${nexusUsername}"

password = "${nexusPassword}"

}

url "${nexusUrl}/socon-group/"

allowInsecureProtocol true

// 가능한 경우 allowInsecureProtocol 제거하고 HTTPS 사용

}

}

jar {

archiveFileName = "${project.name}-${version}.jar"

}

//배포 단계

publishing {

publications {

mavenJava(MavenPublication) {

groupId group

artifactId project.name

version version

artifact("build/libs/$project.name-$version" + ".jar") {

extension 'jar'

}

}

}

repositories {

maven {

credentials(PasswordCredentials) {

username = "${nexusUsername}"

password = "${nexusPassword}"

}

def releasesRepoUrl = "${nexusUrl}/socon-release"

def snapshotsRepoUrl = "${nexusUrl}/socon-snapshot"

url = version.endsWith('SNAPSHOT') ? snapshotsRepoUrl : releasesRepoUrl

allowInsecureProtocol true

// 가능한 경우 allowInsecureProtocol 제거하고 HTTPS 사용

}

}

}

dependencies {

…

}

…

**client 주요 설정**

…

repositories {

mavenCentral()

maven {

credentials(PasswordCredentials) {

username = "${nexusUsername}"

password = "${nexusPassword}"

}

url "${nexusUrl}/socon-group/"

allowInsecureProtocol true

// 가능한 경우 allowInsecureProtocol 제거하고 HTTPS 사용

}

}

…

dependencies {

implementation 'site.socon-socon:utils:1.0.2' //배포한 라이브러리 추가

…

}

…

## 서비스 이용 방법

### Firebase 실시간 알림

1. Firebase 콘솔 프로젝트 만들기

2. 설정 → 프로젝트 설정 → 서비스 계정 → 새 비공개 키 생성